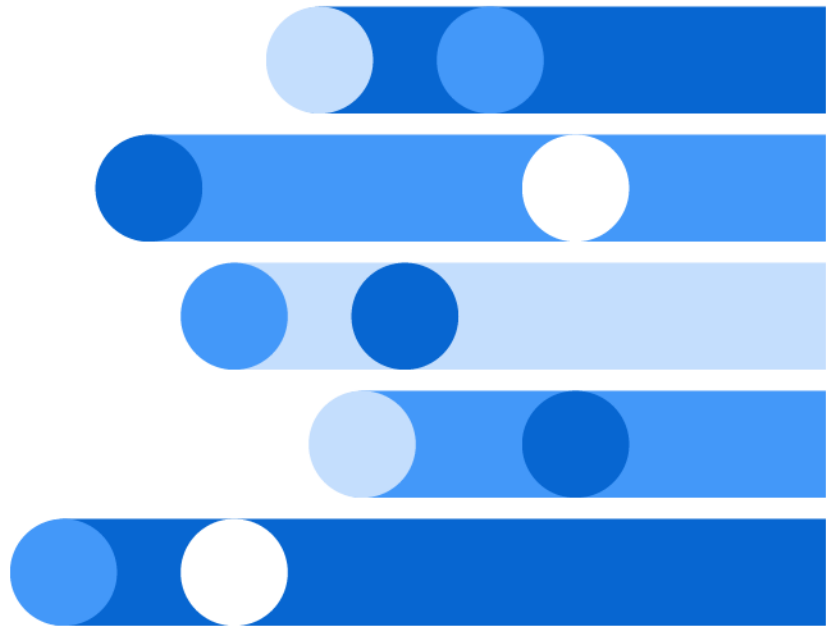




SAS[®] Viya[®] Platform: Machine Learning Python API Guide

2026.04*



* This document might apply to additional versions of the software. Open this document in [SAS Help Center](#) and click on the version in the banner to see all available versions.



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2026. *SAS® Viya® Platform: Machine Learning Python API Guide*. Cary, NC: SAS Institute Inc.

SAS® Viya® Platform: Machine Learning Python API Guide

Copyright © 2026, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

April 2026

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

v_006-P1:ecpymplpg

Contents

Chapter 1 / Data Processing	1
Chapter 2 / Model Assessment	33
Chapter 3 / Bayesian Network Models	45
Chapter 4 / Decomposition Models	55
Chapter 5 / Gaussian Mixture Models	65
Chapter 6 / K-Means Clustering	75
Chapter 7 / Linear Models	85
Chapter 8 / Nominal Dimension Reduction	121
Chapter 9 / Support Vector Models	137
Chapter 10 / Tree Models	153
Chapter 11 / Examples	203
DecisionTreeClassifier	204
DecisionTreeRegressor	222
ForestClassifier	234
ForestRegressor	240
ElasticNet	252
Lasso	255
LinearRegression	259
Ridge	262
GradientBoostingClassifier	266
GradientBoostingRegressor	271
LogisticRegression	283
PCA	293
SVC	296
SVR	302

Data Processing

<i>Classes</i>	1
----------------------	---

Classes

Class Name	Description
Cardinality	The Cardinality class determines the cardinality or limited cardinality of a variable.
CategoricalEncoder	Represents the CategoricalEncoder class, which enables users to group and encode categorical variables by using unsupervised and supervised grouping techniques.
Discretize	Represents the Discretize class, which enables users to perform supervised and unsupervised variable discretization.
Impute	Represents the Impute class, which enables users to impute both interval and nominal variables in various ways.

Cardinality

The Cardinality class determines the cardinality or limited cardinality of a variable.

API: Cardinality

```
class sasviya.ml.cardinality.Cardinality(*, level_order='ASC', level_threshold=20, verbose=0)
```

The cardinality of a variable is the number of its distinct values, and the limited cardinality of a variable is the number of its distinct values that do not exceed a specified threshold.

Parameters

level_order : {"ASC", "DESC"}, optional

Specifies the order in which variable levels are sorted. The level order determines how variable levels are sorted in the details output table as well as in the list that is returned when the `get_variable_levels()` method is called. The default is "ASC".

level_threshold : int in [5,254], optional

Specifies the maximum number of levels of the variables to consider in the input data. When the `summarize()` method is run, only up to the specified `level_threshold` value of variable levels, or distinctive values, are considered. If the number of levels exceeds the `level_threshold` value for a variable, the `_MORE_` column in the summary output table will be 'Y'. The default is 20. Allowed values: [5,254].

verbose : int in {0,1,2}, optional

Specifies the amount of information to print when the `summarize()` method is run. The default is 0. A value of 1 prints the cardinality summary table to the log. A value of 2 prints the cardinality summary and details tables to the log.

Attributes

details_ : Results

Specifies the details table that contains information about the model and its results.

variables_in_ : list(str)

Specifies the variables to include in the summary.

n_variables_in_ : int

Specifies the number of variables.

level_threshold_in_ : int

Specifies the maximum number of variable levels.

level_order_in_ : str

Specifies the order in which variable levels are sorted.

nominal_variables_ : list(str)

Specifies the names of nominal variables to include.

interval_variables_ : list(str)

Specifies the names of interval variables to include.

binary_variables_ : list(str)

Specifies the names of binary variables to include.

id_variables_ : list(str)

Specifies the names of ID variables to include.

Methods

Method Name	Description
customize_results	Generate plots for all actions that support graphics, as well as filter tables from the result list and create custom result orders.
get_details_table	Return the cardinality details table.
get_levels	Return the levels of a variable.
get_n_levels	Return the cardinality of a variable.
get_params	Gets parameters for this model.
get_summary_table	Return the cardinality summary table.
set_params	Updates the parameters of this summarizer.
summarize	Summarize the cardinality of the input data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generate plots for all actions that support graphics, as well as filter tables from the result list and create custom result orders.

Parameters

plots : str or str list

Request plots. This can be a single plot or list of plots. This request can also be “all” or “none”. Parenthesized arguments are also allowed with each plot request. In addition, automatic plots from models can be disabled by setting the environment variable `AUTO_GRAPHICS=false`. However, this variable does not disable automatic plot generation through the `customize_results()` method. You will need to set `plots="none"` on the method to guaranteed that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics* <https://documentation.sas.com/doc/en/pyautogrplots/v_001/p11t293hhzoo85n1r73e8aring2y.htm>.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default dots per inch of the graph. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme used in the graph. The default is “light”. The session default can be set by setting the `GRAPHICS_THEME` environment variable.

path : str

If specified, the plot results are written to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the display results. This can be a keyword (“none”, “all”), a single table, or list of tables. The default is “all”, which keeps all tables. The “none” keyword drops all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names can seen in the top-left corner of all displayed results. If this option is specified with the “exclude_tables” option, then the “include_tables” option has precedence.

exclude_tables : str or str list

Excludes tables from the display results. This can be a keyword (“none”, “graph_tables”, “all”), a single table, or list of tables. The default is “none”, so that all tables are shown. The “graph_tables” keyword drops all tables that were used to produce a graph. The “all” keyword drops all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names can seen in the top-left corner of all displayed results. If this option is specified with the “include_tables” option, then the “include_tables” option has precedence.

order : str or str list

Orders plot and tabular results for display. Any result not specified in the option will be excluded in the new results. The names for the results can be seen in the top-left corner of all displayed results. If this option is specified along with the “include_tables” or “exclude_tables” option, those options are applied BEFORE the ordering is done. If the ordered list contains no results, then the “order” option is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, The CustomResults are typically assigned to a variable for later usage or passed passed directly into the display() method for viewing in a Jupyter notebook cell.

get_details_table

```
get_details_table()
```

Return the cardinality details table.

Returns

Returns an array-like object of the same type as the summarized data.

get_levels

```
get_levels(var)
```

Return the levels of a variable.

Parameters

variable : str

Specifies the variable name.

Returns

list(str)

get_n_levels

```
get_n_levels(var)
```

Return the cardinality of a variable.

Parameters

variable : str

Specifies the variable name.

Returns

int

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

get_summary_table

```
get_summary_table()
```

Return the cardinality summary table.

Returns

Returns an array-like object of the same type as the summarized data.

set_params

```
set_params(**params)
```

Updates the parameters of this summarizer.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

summarize

```
summarize(X)
```

Summarize the cardinality of the input data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform cardinality analysis on.

Returns

self

CategoricalEncoder

Represents the CategoricalEncoder class, which enables users to group and encode categorical variables by using unsupervised and supervised grouping techniques.

API: CategoricalEncoder

```
class sasviya.ml.preprocess.CategoricalEncoder(*, strategy='group_rare',
n_bins=None, max_bins=None, min_bins=None, min_bin_obs=None,
min_bin_percent=5, tree_criteria='gain_ratio', bin_missing=False,
preprocess_rare=False, rare_threshold=None, rare_threshold_percent=5)
```

Parameters

strategy : {'group_rare', 'label', 'classification_tree', 'regression_tree', 'woe'}, default='group_rare'

Specifies which of the following grouping or encoding strategies to use:

- **'group_rare'**: groups rare levels of the analysis variable. This is an unsupervised technique.
- **'label'**: uses one-hot encoding of categorical variables. This is an unsupervised technique.
- **'classification_tree'**: groups on the basis of a one-level decision tree. This is a supervised technique.
- **'regression_tree'**: groups on the basis of a one-level regression tree. This is a supervised technique.
- **'woe'**: groups on the basis of maximizing the information value (IV). This is a supervised technique. The 'event' parameter in the fit and fit_transform methods is required for this strategy.

n_bins : int or list(int), optional

Specifies a list of the number of bins to create for each variable. This parameter can override the following parameters: *max_bins*, *min_bins*, *min_bin_obs*, and *min_bin_percent*.

max_bins : int or list(int), optional

Specifies the maximum number of bins for supervised strategies. The default is computed as $\log_2$ of the number of distinct values.

min_bins : int or list(int), optional

Specifies the minimum number of bins for supervised strategies.

min_bin_obs : int, optional

Specifies the minimum number of observations to include in a bin when the *strategy* parameter value is 'classification_tree', 'regression_tree', or 'woe'. This parameter takes precedence over the *min_bin_percent* parameter.

min_bin_percent : double, optional

Specifies the minimum percentage of all observations to include in a bin when the *strategy* parameter value is 'classification_tree', 'regression_tree', or 'woe', with the range (0,50).

tree_criteria : {'gain', 'gain_ratio', 'gini', 'sse'}, default='gain_ratio'/'sse'

Specifies tree criteria for the *strategy* parameter value 'classification_tree' or 'regression_tree'. The default criterion for 'classification_tree' is 'gain_ratio', and the default criterion for 'regression_tree' is 'sse'.

bin_missing : boolean, optional

Bins missing values in a separate bin.

preprocess_rare : boolean, optional

Groups rare levels before implementing the specified binning strategy. This parameter is not applicable when the *strategy* parameter value is 'label'.

rare_threshold : int, optional

Specifies the rare frequency threshold. This parameter takes precedence over the *rare_threshold_percent* parameter. The *preprocess_rare* parameter value must be True.

rare_threshold_percent : float, optional

Specifies the rare threshold percentage, with the range (0,100). The *preprocess_rare* parameter value must be True.

Attributes

details_ : Results

Stores information about the fitting process and its results.

n_features_in_ : int

Stores the number of input features.

feature_names_in_ : list(str)

Stores the list of input features.

n_bins_ : list(int)

Store the list of the number of bins per feature. Not available when the *strategy* parameter value is 'label'.

n_levels_ : list(int)

Stores the number of levels per feature.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the data.
<code>fit_transform</code>	Fits and transforms the data.
<code>get_feature_names_out</code>	Retrieves the output feature names.
<code>get_params</code>	Gets parameters for this model.
<code>save</code>	Saves the model to a file.
<code>set_params</code>	Updates the parameters of the model.
<code>transform</code>	Transforms the data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does

not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` or `exclude_tables` parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the `order` parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, weight=None, y=None, event=None)
```

Fits the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a fit analysis on.

y : array-like of shape (n_samples,)

Specifies the target labels.

weight : str, optional

Specifies a column from a NumPy ndarray or a Pandas DataFrame.

event : str, optional

Specifies the event level for the target that is used in supervised strategies.

Returns

None

fit_transform

```
fit_transform(X, weight=None, y=None, event=None)
```

Fits and transforms the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a fit analysis on.

y : array-like of shape (n_samples,)

Specifies the target labels.

weight : str, optional

Specifies a column from a NumPy ndarray or a Pandas DataFrame.

event : str, optional

Specifies the event level for the target that is used in supervised strategies.

Returns

result : array-like

Returns an array-like result of transformed values.

get_feature_names_out

```
get_feature_names_out()
```

Retrieves the output feature names.

Returns

list(str)

Returns a list of strings.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

transform

```
transform(X)
```

Transforms the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a transform analysis on.

Returns

result : array-like

Returns an array-like result of transformed values.

Discretize

Represents the Discretize class, which enables users to perform supervised and unsupervised variable discretization.

API: Discretize

```
class sasviya.ml.preprocess.Discretize(*, strategy='uniform', n_bins=[5],
cut_points=None, max_bins=None, min_bins=None, min_bin_obs=None,
min_bin_percent=5, tree_criteria='gain_ratio', bin_missing=False, alpha=0.05,
initial_strategy='uniform', initial_bins=100)
```

Parameters

strategy : {'bucket', 'uniform', 'quantile', 'custom', 'chimerge', 'mdlp', 'classification_tree', 'regression_tree', 'woe'}, default='bucket'|'uniform'
Specifies which of the following discretization strategies to use:

- 'bucket' or 'uniform': creates bins of equal width.
- 'quantile': creates bins of equal frequency.
- 'custom': creates bins according to the user-specified cutpoints. The *cut_points* parameter is required for this strategy.
- 'chimerge': creates bins on the basis of chi-square merging of neighboring bins. This is a bottom-up supervised discretization technique.
- 'mdlp': creates bins on the basis of the minimum description length. This is a top-down supervised discretization technique.
- 'classification_tree': creates bins on the basis of a one-level decision tree. This is a top-down supervised discretization technique.
- 'regression_tree': creates bins on the basis of a one-level regression tree. This is a top-down supervised discretization technique.
- 'woe': creates bins on the basis of the WOE criterion. This is a top-down supervised discretization technique. The *event* parameter in the fit and fit_transform methods is required for this strategy.

n_bins : int or list(int), optional

Specifies a list of the number of bins to create for each variable. This parameter can override the following parameters: *max_bins*, *min_bins*, *min_bin_obs*, and *min_bin_percent*. The default is 5 for unsupervised strategies.

initial_strategy : {'bucket', 'quantile'}, default='bucket', optional

Specifies the initial binning method for interval inputs to start supervised strategies.

initial_bins : int, optional

Specifies the initial number of bins for interval inputs to start supervised strategies.

cut_points : list(int), optional

Specifies the user-provided cutpoints when the *strategy* parameter value is 'custom'.

max_bins : int or list(int), optional

Specifies the maximum number of bins for supervised strategies. The default is 5 for supervised strategies.

min_bins : int or list(int), optional

Specifies the minimum number of bins for supervised strategies. The default is 1 for supervised strategies.

min_bin_obs : int, optional

Specifies the minimum number of observations to include in a bin when the *strategy* parameter value is 'classification_tree', 'regression_tree', or 'woe'. This parameter takes precedence over the *min_bin_percent* parameter.

min_bin_percent : double, optional

Specifies the minimum percentage of all observations to include in a bin when the *strategy* parameter value is 'classification_tree', 'regression_tree', or 'woe', with the range (0,50).

tree_criteria : {'gain', 'gain_ratio', 'gini', 'sse'}, default='gain_ratio'|'sse'

Specifies tree criteria for the *strategy* parameter value 'classification_tree' or 'regression_tree'. The default criterion for 'classification_tree' is 'gain_ratio', and the default criterion for 'regression_tree' is 'sse'.

bin_missing : boolean, optional

Bins missing values in a separate bin.

alpha : double, optional

Specifies the significance level for the *strategy* parameter value 'chimerge', with the range (0,1).

Attributes

details_ : Results

Stores information about the fitting process and its results.

n_features_in_ : int

Stores the number of input features.

feature_names_in_ : list(str)

Stores the list of input features.

n_bins_ : list(int)

Stores the list of the number of bins per feature.

bin_edges_ : ndarray of ndarray of shape (n_features,)

Stores an array of bin edges per feature.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the data.
<code>fit_transform</code>	Fits and transforms the data.
<code>get_feature_names_out</code>	Retrieves the output feature names.
<code>get_params</code>	Gets parameters for this model.
<code>save</code>	Saves the model to a file.
<code>set_params</code>	Updates the parameters of the model.
<code>transform</code>	Transforms the data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, weight=None, y=None, event=None)
```

Fits the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a fit analysis on.

y : array-like of shape (n_samples,)

Specifies the target labels.

weight : str, optional

Specifies a column from a NumPy ndarray or a Pandas DataFrame.

event : str, optional

Specifies the event level for the target that is used in supervised strategies.

Returns

None

fit_transform

```
fit_transform(X, weight=None, y=None, event=None)
```

Fits and transforms the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a fit analysis on.

y : array-like of shape (n_samples,)

Specifies the target labels.

weight : str, optional

Specifies a column from a NumPy ndarray or a Pandas DataFrame.

event : str, optional

Specifies the event level for the target that is used in supervised strategies.

Returns

result : array-like

Returns an array-like result of transformed values.

get_feature_names_out

```
get_feature_names_out()
```

Retrieves the output feature names.

Returns

list(str)

Returns a list of strings.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

transform

```
transform(X)
```

Transforms the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a transform analysis on.

Returns

result : array-like

Returns an array-like result of transformed values.

Impute

Represents the Impute class, which enables users to impute both interval and nominal variables in various ways.

API: Impute

```
class sasviya.ml.preprocess.Impute(*, strategy='mean', values_numeric=None,
values_string=None)
```

Parameters

strategy : {'mean', 'median', 'min', 'max', 'midrange', 'mode', 'most_frequent', 'value', 'constant'}, default='mean'

Specifies which of the following imputation strategies to use:

- *'mean'*: replaces missing values with the mean of the column.
- *'median'*: replaces missing values with the median of the column.
- *'min'*: replaces missing values with the minimum of the column.
- *'max'*: replaces missing values with the maximum of the column.
- *'midrange'*: replaces missing values with the midrange of the column.
- *'mode'* or *'most_frequent'*: replaces missing values with the mode of the column.
- *'value'*: replaces missing values with a specified value (see the *values_numeric* and *values_string* parameters).
- *'constant'*: replaces missing values with a constant value (see the *values_numeric* and *values_string* parameters).

values_numeric : float or list(float), optional

Specifies the value to use in numeric columns when the *strategy* parameter value is 'value' or 'constant'.

values_string : str or list(str), optional

Specifies the value to use in character columns when the *strategy* parameter value is 'value' or 'constant'.

Attributes

details_ : Results

Stores information about the fitting process and its results.

n_features_in_ : int

Stores the number of input features.

feature_names_in_ : list(str)

Stores the list of input features.

imputed_values_ : list(float) or list(str)

Stores the list of imputed values.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fits the data.
fit_transform	Fits and transforms the data.
get_feature_names_out	Retrieves the output feature names.
get_params	Gets parameters for this model.
save	Saves the model to a file.
set_params	Updates the parameters of the model.
transform	Transforms the data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left

corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, weight=None, y=None)
```

Fits the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a fit analysis on.

weight : str

Specifies a column from a NumPy ndarray or a Pandas DataFrame.

Returns

None

fit_transform

```
fit_transform(X, weight=None, y=None)
```

Fits and transforms the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a fit analysis on.

weight : str

Specifies a column from a NumPy ndarray or a Pandas DataFrame.

Returns

Returns an array-like result of transformed values.

get_feature_names_out

```
get_feature_names_out()
```

Retrieves the output feature names.

Returns

list(str)

Returns a list of strings.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

transform

```
transform(X)
```

Transforms the data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform a transform analysis on.

Returns

result : array-like

Returns an array-like result of transformed values.

Model Assessment

<i>Classes</i>	33
----------------------	----

Classes

Class Name	Description
Assess	The Assess class enables you to measure and evaluate the performance of supervised learning models.
AssessBias	The AssessBias class calculates bias metrics for predictive models.

Assess

The Assess class enables you to measure and evaluate the performance of supervised learning models.

API: Assess

```
class sasviya.ml.assess.Assess(*, input=None, y_true=None, pos_label=None,
p_var=None, p_pos_label=None, cut_step=None, user_cutoff=None, verbose=0)
```

The Assess class supports both regression and classification models. It provides the ROC-and-fit-statistics-related metrics.

Parameters

input : str

Specifies the predicted variable to assess. For classification models, it is the predicted probability of the specified target event. For regression models, it is the predicted target.

y_true : str

Specifies the true target variable.

pos_label : str

Specifies the target event for classification model assessment. If the pos_label is not specified, the assessment is for regression models.

p_pos_label : list(str), optional

Specifies the target levels, excluding the pos_label. This parameter is for classification models.

p_var : list(str)

Specifies the predicted probabilities of the p_pos_label. This parameter is for classification models.

cut_step : float, optional

Specifies an interval to generate a set of thresholds between 0 and 1 for evaluating metrics related to confusion matrices. This parameter is for classification models. By default, cut_step=0.01.

user_cutoff : float, optional

Specifies the threshold for evaluating metrics related to confusion matrices. The user_cutoff value is in (0, 1). This is for classification models. By default, user_cutoff=0.5.

verbose : int, optional

Specifies the amount of information to print during summarize(). Allowed values: [0,1,2]. A verbosity level of 1 prints the fit statistic table to the log. A verbosity level of 2 prints the ROC table and the fit statistic table to the log. By default, verbose=0.

Attributes

The following attributes are associated with parameters specified in the constructor:

details_ : Results

Additional information about the summarize process and the final results.

input_in_ : str

The predicted probability variable.

y_true_in_ : str

The specified target.

pos_label_in_ : str

The specified target event.

cut_step_in_ : float

The specified value of option cut_step, default is 0.01.

user_cutoff_in_ : float

The specified value of option user_cutoff, default is 0.5.

The following attributes are associated with regression models:

mean_absolute_error_ : float

The returned mean absolute error for regression models.

mean_squared_error_ : float

The returned mean square error for regression models.

The following attributes are associated with classification models:

accuracy_score_ : float

The classification score.

log_loss_ : float

The Log Loss metric for classification models.

roc_curve_ : array list

The returned array of TPR and FPR.

auc_ : float

The returned area under the curve for classification model.

precision_recall_curve_ : array list

The returned array of the precision score and recall score.

The following attributes are associated with user-defined cutoffs:

f1_score_ : float

The returned f1 score for classification models.

confusion_matrix_ : array

The returned 2x2 matrix for the specified event and threshold.

precision_score_ : float

The returned precision score for classification models.

recall_score_ : float

The returned recall score for classification models.

classification_report_ : dict

The returned f1 score, precision, and recall for the target event and overall precision score.

Methods

Method Name	Description
<code>customize_results</code>	Generate plots for all actions that support graphics, as well as filter tables from the result list and create custom result orders.
<code>get_params</code>	Gets parameters for this model.
<code>set_params</code>	Updates the parameters of this summarizer.
<code>summarize</code>	Summarize the scored data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                  theme=None, path=None, output_type=None, include_tables=None,
                  exclude_tables=None, order=None)
```

Generate plots for all actions that support graphics, as well as filter tables from the result list and create custom result orders.

Parameters

plots : str or str list

Request plots. This can be a single plot or list of plots. This request can also be “all” or “none”. Parenthesized arguments are also allowed with each plot request. In addition, automatic plots from models can be disabled by setting the environment variable `AUTO_GRAPHICS=false`. However, this variable does not disable automatic plot generation through the `customize_results()` method. You will need to set `plots="none"` on the method to guaranteed that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics* <https://documentation.sas.com/doc/en/pyautogrplots/v_001/p11t293hhzoo85n1r73e8aring2y.htm>.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default dots per inch of the graph. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme used in the graph. The default is "light". The session default can be set by setting the GRAPHICS_THEME environment variable.

path : str

If specified, the plot results are written to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the display results. This can be a keyword ("none", "all"), a single table, or list of tables. The default is "all", which keeps all tables. The "none" keyword drops all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names can be seen in the top-left corner of all displayed results. If this option is specified with the "exclude_tables" option, then the "include_tables" option has precedence.

exclude_tables : str or str list

Excludes tables from the display results. This can be a keyword ("none", "graph_tables", "all"), a single table, or list of tables. The default is "none", so that all tables are shown. The "graph_tables" keyword drops all tables that were used to produce a graph. The "all" keyword drops all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names can be seen in the top-left corner of all displayed results. If this option is specified with the "include_tables" option, then the "include_tables" option has precedence.

order : str or str list

Orders plot and tabular results for display. Any result not specified in the option will be excluded in the new results. The names for the results can be seen in the top-left corner of all displayed results. If this option is specified along with the "include_tables" or "exclude_tables" option, those options are applied BEFORE the ordering is done. If the ordered list contains no results, then the "order" option is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, The CustomResults are typically assigned to a variable for later usage or passed passed directly into the display() method for viewing in a Jupyter notebook cell.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

set_params

```
set_params(**params)
```

Updates the parameters of this summarizer.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

summarize

```
summarize(X)
```

Summarize the scored data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to perform assess analysis on

Returns

self

AssessBias

The AssessBias class calculates bias metrics for predictive models.

API: AssessBias

```
class sasviya.ml.assessbias.AssessBias(*, target, target_type, sensitive_var,
pred_vars, levels=None, event=None, reference_level=None, cutoff=None,
n_bins=None, depth=None, n_cuts=None, verbose=0)
```

In predictive modeling, bias occurs when the prediction or performance of a model differs for different values of a given variable, which is referred to here as the “sensitive” variable. Prediction bias is measured by calculating the difference in average model prediction for different values of the sensitive variable. Performance bias is measured by calculating the difference in model performance for different values of the sensitive variable.

The AssessBias class calculates performance statistics and average model predictions for each level of a nominal variable. The difference in these statistics is then calculated either by reporting the maximum pairwise difference among all levels or by comparing each level to a reference group in order to report a single number that represents model bias.

Parameters

target : str

Specifies the target variable.

target_type: {"INTERVAL", "NOMINAL"}

Specifies the target type. The default is INTERVAL.

sensitive_var : str

Specifies the sensitive variable to use in bias calculations.

pred_vars : list(str)

Specifies the list of variables that contain the model's predictions. The order of the variables must match the order that you specify in the levels parameter.

levels : list(str), optional

Specifies the list of formatted values of the target variable. The order of the variables must match the order that you specify in the pred_vars parameter.

event : str, optional

Specifies the formatted value of the target variable that represents the event of interest. For an interval target, this parameter is ignored.

reference_level : str, optional

Specifies the reference level for the sensitive variable.

cutoff : float, optional

Specifies the cutoff for the confusion matrix. The default is 0.5. The allowed values are (0, 1).

n_bins : int, optional

Specifies the number of bins to use in lift calculations. The default is 20. The allowed values are [2, 100].

depth : int, optional

Specifies the depth to use in lift calculations. The default is 10. The allowed values are (0, 100].

n_cuts : int, optional

Specifies the number of cuts to use in the receiver operating characteristic (ROC) calculation. The default is 100. The allowed values are [2, infinity].

verbose : int, optional

Specifies the amount of information to print during summarize(). The default is 0. The allowed values are [0, 1, 2].

Attributes

details_ : Results

The details table that contains information about the model and its results.

Methods

Method Name	Description
customize_results	Generate plots for all actions that support graphics, as well as filter tables from the result list and create custom result orders.
get_params	Gets parameters for this model.
set_params	Updates the parameters of this summarizer.

Method Name	Description
<code>summarize</code>	Assess the bias of the scored input data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
theme=None, path=None, output_type=None, include_tables=None,
exclude_tables=None, order=None)
```

Generate plots for all actions that support graphics, as well as filter tables from the result list and create custom result orders.

Parameters

plots : str or str list

Request plots. This can be a single plot or list of plots. This request can also be “all” or “none”. Parenthesized arguments are also allowed with each plot request. In addition, automatic plots from models can be disabled by setting the environment variable `AUTO_GRAPHICS=false`. However, this variable does not disable automatic plot generation through the `customize_results()` method. You will need to set `plots="none"` on the method to guaranteed that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics* <https://documentation.sas.com/doc/en/pyautogrplots/v_001/p11t293hhzoo85n1r73e8aring2y.htm>.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default dots per inch of the graph. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme used in the graph. The default is “light”. The session default can be set by setting the `GRAPHICS_THEME` environment variable.

path : str

If specified, the plot results are written to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the display results. This can be a keyword (“none”, “all”), a single table, or list of tables. The default is “all”, which keeps all tables. The “none” keyword drops all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names can be seen in the top-left corner of all displayed results. If this option is specified with the “exclude_tables” option, then the “include_tables” option has precedence.

exclude_tables : str or str list

Excludes tables from the display results. This can be a keyword (“none”, “graph_tables”, “all”), a single table, or list of tables. The default is “none”, so that all tables are shown. The “graph_tables” keyword drops all tables that were used to produce a graph. The “all” keyword drops all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names can be seen in the top-left corner of all displayed results. If this option is specified with the “include_tables” option, then the “include_tables” option has precedence.

order : str or str list

Orders plot and tabular results for display. Any result not specified in the option will be excluded in the new results. The names for the results can be seen in the top-left corner of all displayed results. If this option is specified along with the “include_tables” or “exclude_tables” option, those options are applied BEFORE the ordering is done. If the ordered list contains no results, then the “order” option is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, The CustomResults are typically assigned to a variable for later usage or passed passed directly into the display() method for viewing in a Jupyter notebook cell.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

set_params

```
set_params(**params)
```

Updates the parameters of this summarizer.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

summarize

```
summarize(X)
```

Assess the bias of the scored input data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the input data to assess the bias of. Does not support NumPy arrays.

Returns

self

Examples

```
from sasviya.ml.tree import GradientBoostingRegressor
from sasviya.ml import AssessBias
# Train a gradient boosting model
model = GradientBoostingRegressor()
model.fit(simdata.drop(['intervalTarget', 'mySensitiveVariable'], axis=1),
          simdata['intervalTarget'])
# Score the data to get predictions
```

```
scored_data = model.predict(simdata)
# Assess bias in the model predictions
bias_assessor = AssessBias(
    target='intervalTarget',
    target_type='INTERVAL',
    sensitive_var='mySensitiveVariable',
    pred_vars=['P_intervalTarget']
)
bias_assessor.summarize(scored_data)
```


Bayesian Network Models

Classes 45

Classes

Class Name	Description
BNET	Bayesian Network Classification Model

BNET

Bayesian Network Classification Model

API: BNET

```
class sasviya.ml.bnet.BNET(num_bin=5, pre_screening=1, var_select=1,
missing_int='ignore', missing_nom='ignore', indep_test='chigsquare', alpha=0.05,
mi_alpha=0.05, structure='pc', parenting='bestset', max_parents=5,
best_model=False, in_network=None, verbose=0)
```

Parameters

num_bin : int, optional

Specifies the number of binning levels for all continuous variables.

pre_screening : {0, 1}, optional

Specifies the initial screening for the input variables.

var_select : {0, 1, 2, 3}, optional

Specifies the selection for the input variables.

missing_int : {"ignore", "impute"}, optional

Specifies how to handle missing values for continuous variables.

missing_nom : {"ignore", "impute", "level"}, optional

Specifies how to handle missing values for nominal variables.

indep_test : {"all", "chigsquare", "chisquare", "gsquare", "mi"}, optional

Specifies the methods for independence tests.

alpha : float, optional

Specifies the significance level for independence tests by using chi-square or G-square statistics.

mi_alpha : float, optional

Specifies the significance level for independence tests by using mutual information.

structure : {"GENERAL", "GN", "NAIVE", "MB", "TAN", "PC"}, optional

Specifies the network structure types.

parenting : {"bestone", "bestset"}, optional

Specifies the structure learning methods.

max_parents : int, optional

Specifies the maximum number of parents that are allowed for each node in the network.

best_model : bool, optional

Requests that the best model be selected.

verbose: int, optional

Specifies the amount of information to print during model fitting.

Attributes

classes_ : list of str

Stores class labels that are seen during model fitting.

details_ : Results

Stores additional information about the model fitting process and the final results.

feature_names_in : list of str

Stores the column names of the input features that are seen during fitting.

n_features_in_ : int

Stores the number of input features that are seen by the model.

outnetwork_ : DataFrame

Stores the parent-child connections that the fitted network contains.

target_name_in_ : str

Stores the column name of the target variable.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
decision_function	Computes the decision function given input features.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fits the model to the training data.
get_params	Gets parameters for this model.
predict	Predicts class labels given input features.
predict_proba	Estimates probabilities for each class label.
save	Saves the model to a file.
score	Determines model accuracy by using the data set X and the targets y.
set_params	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see SAS Viya: Managing Plots in Python Automatic Graphics.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with

the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

decision_function

```
decision_function(X)
```

Computes the decision function given input features.

Parameters

X : array-like

Specifies training features in the shape (n_samples, n_features).

Returns

array-like

Returns the estimated posterior probability for the target class. This is (n_samples, n_classes) for the multiclass case and (n_samples,) with just `self.classes_[1]` returned for the binary case.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,). The target is assumed to be categorical.

nominals: list of str, optional

Specifies the names of columns in *X* to treat as categorical.

Returns

self : object

Returns the instance itself.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts class labels given input features.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in *X*.

Returns

array-like

Returns predicted class labels in the shape (n_samples,).

predict_proba

```
predict_proba(X, copy_var=None)
```

Estimates probabilities for each class label.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns estimated class label probabilities in the shape (n_samples, n_classes).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Decomposition Models

<i>Classes</i>	55
----------------------	----

Classes

Class Name	Description
PCA	Principal component analysis.

PCA

Principal component analysis.

API: PCA

```
class sasviya.ml.decomposition.PCA(n_components=None, *, svd_solver='auto',
iterated_power=1, random_state=None, scale=True)
```

A class that applies a principal component analysis technique for examining relationships among several quantitative variables. The class provides an optimal way to reduce dimensionality by projecting the data onto a lower-dimensional orthogonal subspace that explains as much variation as possible in those variables.

You can use principal component analysis to reduce the number of variables in regression, clustering, and so on.

Parameters

n_components : int or None, optional

Specifies the number of principal components to extract. The value must be less than $\min(n_samples, n_features)$. The default is $\min(n_samples, n_features)$.

svd_solver : {"auto", "full", "randomized"}, optional

Specifies the solver to be used to calculate the principal components. If the specified value is "auto", then "full" is used unless the input data is larger than 500×500 and the *n_components* parameter value is $< 80\%$ of the smallest dimension of the data.

iterated_power : int, optional

Specifies the number of solver iterations when the *svd_solver* parameter value is "randomized". Otherwise this parameter is ignored.

random_state : int, optional

Specifies the seed to use for random number generation. This parameter is ignored if the *svd_solver* parameter value is not "randomized".

scale : bool, optional

When set to True, scales variables. Interval inputs are always centered by their corresponding means. When this parameter is set to True, the method divides the inputs by their corresponding standard deviations.

Attributes

details_ : Results

Specifies additional information about the model fitting process and the final results.

components_ : ndarray of shape (n_components, n_features)

Specifies the principal axes in feature space, which represent the directions of maximum variance.

explained_variance_ : ndarray of shape (n_components,)

Specifies the variance explained by each of the selected components.

explained_variance_ratio_ : ndarray of shape (n_components,)

Specifies the percentage of variance explained by each of the selected components.

feature_names_in_ : list(str)

Specifies the number of features to use in the model. This parameter value is equal to $\text{len}(\text{feature_names_in_})$.

mean_ : ndarray of shape (n_features,)

Specifies the per-feature empirical mean, which is estimated from the training data.

n_components_ : int

Specifies the number of principal components to extract.

n_features_in_ : int

Specifies the number of features to use in the model. This parameter value is equal to `len(feature_names_in_)`.

n_samples_ : int

Specifies the number of samples in the training data.

singular_values_ : ndarray of shape (n_components,)

Specifies the singular values that correspond to each of the selected components.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fit the model with X.
fit_transform	Fit the model and apply dimensionality reduction to X.
get_covariance	Get the covariance of the data.
get_feature_names_out	Get column names for the transformer's output.
get_params	Gets parameters for this model.
inverse_transform	Transform data back to its original space.
save	Saves the model to a file.
set_params	Updates the parameters of the model.
transform	Reduce the dimensionality of input data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left

corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y=None)
```

Fit the model with X.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the training data, where *n_samples* is the number of samples and *n_features* is the number of features.

y : any

This parameter is ignored (for API consistency only).

Returns

self : object

Returns the instance itself.

fit_transform

```
fit_transform(X, y=None)
```


Fit the model and apply dimensionality reduction to X .

Parameters

X : array-like of shape $(n_samples, n_features)$

Specifies the training data, where $n_samples$ is the number of samples and $n_features$ is the number of features.

y : any

This parameter is ignored (for API consistency only).

Returns

array-like

Returns X reduced to $(n_samples, n_components)$.

get_covariance

```
get_covariance()
```

Get the covariance of the data.

Returns

ndarray of shape $(n_features, n_features)$.

Returns the covariance matrix.

get_feature_names_out

```
get_feature_names_out()
```

Get column names for the transformer's output.

Returns

ndarray of str

Returns the column names.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

inverse_transform

```
inverse_transform(X, copy_var=None)
```

Transform data back to its original space.

In other words, return an input *X_{original}* whose transform would be X.

Parameters

X : array-like of shape (n_samples, n_components)

Specifies the transformed data, where *n_samples* is the number of samples and *n_components* is the number of components.

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

X_{original} : array-like of shape (n_samples, n_features)

Returns the original data, where *n_samples* is the number of samples and *n_features* is the number of features.

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

transform

```
transform(X, copy_var=None)
```

Reduce the dimensionality of input data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the data to project onto the first principal components.

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns X reduced to (n_samples, n_components).

Gaussian Mixture Models

Classes 65

Classes

Class Name	Description
GaussianMixtureModel	The GaussianMixtureModel class fits a probabilistic GMM clustering model to data and predicts cluster assignments for new values.

GaussianMixtureModel

The GaussianMixtureModel class fits a probabilistic GMM clustering model to data and predicts cluster assignments for new values.

API: GaussianMixtureModel

```
class
sasviya.ml.gaussianmixturemodel.GaussianMixtureModel.GaussianMixtureModel(*, n_components=100, covariance_type='diagonal', max_iter=100, weight_concentration_prior=1, tol=0.01, random_state=None, verbose=0)
```

Parameters

n_components : int, optional

Specifies the number of mixture components. The default is 100.

covariance_type : {"diagonal|diag", "full"}, optional

Specifies the covariance matrix type of the Gaussian mixtures. The default is "diagonal".

max_iter : int, optional

Specifies the maximum number of variational Bayesian (VB) iterations to perform. The default is 100.

weight_concentration_prior : float, optional

Specifies the Dirichlet concentration of each component on the weight distribution. The default is 1.

tol : float, optional

Specifies the convergence threshold of the VB algorithm. The default is 0.01.

random_state : float, optional

Specifies the seed to use for random number generation for initialization.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

details_ : Results

Specifies additional information about the model fitting process and the final results.

cluster_summary_ : ndarray of shape (n_components, n_features+3)

Specifies the cluster size, neighbor, and center of each mixture component.

cluster_info_ : ndarray of shape (n_components+2, n_features*2 + 2)

Specifies the cluster weight, mean, and variance of each mixture component.

cluster_covariance_ : ndarray of shape (n_components*n_features, n_features + 1)

Specifies the covariance of each mixture component.

labels_ : ndarray of shape (n_samples,)

Specifies the labels of each observation.

converged_ : bool

When set to True, convergence is reached in fit().

n_iter_ : int

Specifies the number of model fitting iterations to perform.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during fitting.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fit the model to the training data.
<code>fit_predict</code>	Fit the model to the provided training data and predict cluster assignments.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predict the closest cluster for each given input feature.
<code>predict_proba</code>	Predict the cluster probabilities for each given input feature.
<code>save</code>	Saves the model to a file.
<code>score</code>	Score clustering results on data X, based on the learned model.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns**CustomResults**

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns**Results**

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters**file : str, pathlib.Path, or file-like, optional**

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns**bytes or None**

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y=None)
```

Fit the model to the training data.

Parameters**X : array-like of shape (n_samples, n_features)**

Specifies the training data, where *n_samples* is the number of samples and *n_features* is the number of features.

y : any

This parameter is ignored (included for API consistency only).

Returns**self : object**

Returns the instance itself.

fit_predict

```
fit_predict(X, y=None)
```

Fit the model to the provided training data and predict cluster assignments.

Equivalent to calling `fit(X)` followed by `predict(X)`.

Parameters**X : array-like**

Specifies the samples to fit and predict the closest cluster for, with the shape (n_samples, n_features).

y : any

This parameter is ignored (included for API consistency only).

Returns

type(X)

Returns the predicted index of the cluster for each input in the shape (n_samples,).

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X)
```

Predict the closest cluster for each given input feature.

Parameters

X : array-like

Specifies the samples to predict, with the shape (n_samples, n_features).

Returns

type(X)

Returns the predicted index of the cluster for each input in the shape (n_samples,).

predict_proba

```
predict_proba(X)
```

Predict the cluster probabilities for each given input feature.

Parameters

X : array-like

Specifies the samples to predict, with the shape (n_samples, n_features).

Returns

type(X)

Returns the predicted probabilities of the cluster for each input in the shape (n_samples, n_components).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y=None)
```

Score clustering results on data X, based on the learned model.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies new data, where *n_samples* is the number of samples and *n_features* is the number of features.

y : any

This parameter is ignored (included for API consistency only).

Returns

type(X)

Returns the index of the cluster that each observation belongs to, along with the probability that it belongs to the cluster.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

K-Means Clustering

Classes 75

Classes

Class Name	Description
KMeans	K-Means clustering.

KMeans

K-Means clustering.

API: KMeans

```
class sasviya.ml.kmeans.KMeans(*, n_clusters=6, init='random_centroids',
max_iter=10, random_state=None, distance='Euclidean', distance_nom='Binary',
impute=None, impute_nom=None, verbose=0, standardize=None,
estimate_n_clusters=None, stop_criterion=None, gamma=None)
```

Parameters

n_clusters : int, optional

Specifies the number of clusters to form and centroids to generate.

init : {"forgy", "rand|random"}, optional

Specifies the method for obtaining the initial estimate of cluster centers. The "forgy" method selects n_cluster data points at random to use as the cluster centroids. The "rand" method assigns observations to a cluster at random. By default, init = "forgy".

max_iter : int, optional

Specifies the maximum number of iterations for the algorithm to perform.

random_state : int, optional

Specifies the seed to use for random number generation for initialization.

verbose : int, optional

Specifies the amount of information to send to the log.

standardize: None, "Range", "STD"

Specifies the method for standardizing interval input variables.

estimate_n_clusters: Specifies the ABC params in a dictionary.

noc: "abc", None

Specifies if the ABC method will be used to infer the number of clusters.

align: "pca", None

Specifies the method for aligning the reference data based on the input data.

B: int, numpy.int32

Specifies the number of reference data to be created for each cluster candidate.

criterion: "all", "firstmaxwithstd", "firstpeak", "globalpeak"

Specifies the criterion to be used to estimate the number of clusters that use the statistics obtained by the ABC method.

min_clusters: int, numpy.int32

Specifies the minimum number of clusters for searching for the best number of clusters.

stop_criterion: Specifies the stop criterion for the clustering, a dictionary.

change: "cluster_change", "wcscs_change"

Specifies the method to use for convergence. If you do not specify this option, the algorithm stops after it reaches the maximum number of iterations (which is specified in the max_iter parameter).

value: the threshold value for the change.

gamma: Specify the gamma value for the kprototypes.

Attributes

details_ : Results

Additional information about the model fitting process and the final results.

cluster_centers_ : ndarray of shape (n_clusters, n_features)

The coordinates of cluster centers.

labels_ : ndarray of shape (n_samples,)

The labels of each observation.

inertia_ : float

The sum of squared distances of observations to their closest cluster center.

n_iter_ : int

The number of model fit iterations performed.

n_features_in_ : int

The number of input features seen by the model.

feature_names_in_ : list of str

The column names of the input features seen during fitting.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fits the model on the training data.
fit_predict	Fits the model on the provided training data and predicts cluster assignments
get_params	Gets parameters for this model.
predict	Predicts the closest cluster for each given input feature.
save	Saves the model to a file.

Method Name	Description
<code>score</code>	Estimates model fitness by calculating the negative of the K-Means objective.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
theme=None, path=None, output_type=None, include_tables=None,
exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see SAS Viya: Managing Plots in Python Automatic Graphics.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the *display()* method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, nominals=None, y=None)
```

Fits the model on the training data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the training data, where *n_samples* is the number of samples and *n_features* is the number of features.

nominals: iterable of str, optional

Specifies the nominal features in the input data.

y : any

Ignored (included for API consistency only).

Returns

self : object

Returns the instance itself.

fit_predict

```
fit_predict(X, nominals=None, y=None)
```

Fits the model on the provided training data and predicts cluster assignments

Equivalent to calling fit(X) followed by predict(X).

Parameters

X : array-like

Specifies the samples to fit and predict the closest cluster for, with shape (n_samples, n_features).

y : any

Ignored (included for API consistency only).

Returns

array-like

Predicted index of the cluster for each input in the shape (n_samples,).

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts the closest cluster for each given input feature.

Parameters

X : array-like

Specifies the samples to predict, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns a predicted index of the cluster for each input in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X=None)
```

Estimates model fitness by calculating the negative of the K-Means objective.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Linear Models

<i>Classes</i>	85
----------------------	----

Classes

Class Name	Description
ElasticNet	Linear regression with a mix of L1 and L2 regularization.
Lasso	Linear regression with L1 regularization.
LinearRegression	Ordinary least squares linear regression.
LogisticRegression	Logistic Regression classifier.
Ridge	Linear Regression with L2 regularization.

ElasticNet

Linear regression with a mix of L1 and L2 regularization.

API: ElasticNet

```
class sasviya.ml.linear_model.ElasticNet(alpha=None, *, l1_ratio=0.5,
fit_intercept=True, tol=1e-08, solver='lbfgs', verbose=0)
```

Parameters

alpha : float, optional

Specifies the strength of the regularization that is applied. A constant value is used to scale the regularization term of the cost function. If None is specified, the regularization term is computed from the data. “Alpha” is also referred to as “lambda” in fit results.

l1_ratio : float, optional

Specifies the elastic net mixing parameter. Specifies the ratio of L1:L2 regularization. If the specified value is 0, only L2 regularization is used. If the specified value is 1, only L1 regularization is used.

fit_intercept : bool, optional

Specifies whether or not to include a bias term in the model.

solver : {“admm”, “bfgs”, “lbfgs”, “nlp”}, optional

Specifies the optimization algorithm to use.

tol : float, optional

Specifies the tolerance threshold for early stopping. Model fitting stops when the size of the gradient is less than the value of *tol*.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

coef_ : ndarray of shape (n_parameters,)

Specifies the model coefficients. The number of parameters can be greater than the number of features if additional parameters are introduced to handle missing values, or less than the number of features if variable selection is used.

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

intercept_ : ndarray of shape (, n_intercepts)

Specifies the bias term(s) to be added to the model.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts continuous values given input features.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots=“none”` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {“light”, “dark”, “opal”, “midnight”, “raven”, “htmlencore”, “highcontrast”, “grayscale”, “monochrome”}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {“svg”, “png”}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns**CustomResults**

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns**Results**

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters**file : str, pathlib.Path, or file-like, optional**

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Lasso

Linear regression with L1 regularization.

API: Lasso

```
class sasviya.ml.linear_model.Lasso(*, fit_intercept=True, verbose=0)
```

Parameters

fit_intercept : bool, optional

Specifies whether or not to include a bias term in the model.

verbose : int

Specifies the amount of information to print about the fitted model.

Attributes

coef_ : ndarray of shape (n_parameters,)

Specifies the model coefficients. The number of parameters can be greater than the number of features if additional parameters are introduced to handle missing values, or less than the number of features if variable selection is used.

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

intercept_ : ndarray of shape (, n_intercepts)

Specifies the bias term(s) to be added to the model.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fits the model to the training data.

Method Name	Description
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts continuous values given input features.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
theme=None, path=None, output_type=None, include_tables=None,
exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots=“none”` in the method to guarantee that only tables are returned. For more information, see SAS Viya: Managing Plots in Python Automatic Graphics.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {“light”, “dark”, “opal”, “midnight”, “raven”, “htmlencore”, “highcontrast”, “grayscale”, “monochrome”}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the *display()* method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

LinearRegression

Ordinary least squares linear regression.

API: LinearRegression

```
class sasviya.ml.linear_model.LinearRegression(*, fit_intercept=True,
verbose=0)
```

Parameters

fit_intercept : bool, optional

Specifies whether or not to include a bias term in the model.

verbose : int

Specifies the amount of information to print about the fitted model.

Attributes

coef_ : pandas.DataFrame of shape (n_parameters,)

Specifies the model coefficients. The number of parameters can be greater than the number of features if additional parameters are introduced to handle missing values, or less than the number of features if variable selection is used.

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

intercept_ : pandas.DataFrame of shape (, n_intercepts)

Specifies the bias term(s) to be added to the model.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts continuous values given input features.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method.

You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` or `exclude_tables` parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the `order` parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

LogisticRegression

Logistic Regression classifier.

API: LogisticRegression

```
class sasviya.ml.linear_model.LogisticRegression(*, tol=1e-08,  
fit_intercept=True, solver='nrridg', selection=None, max_iter=50,  
max_time=None, verbose=0)
```

Parameters

tol : float, optional

Specifies the relative gradient convergence criterion.

fit_intercept : bool, optional

Specifies whether to include a bias term in the model.

max_iter : float, optional

Specifies the maximum number of training iterations.

max_time : float, optional

Specifies the maximum training time, in seconds.

solver : {"congra", "dbldog", "lbfgs", "newrap", "nmsimp", "nrridg", "quanew", "duquanew"}, optional

Specifies the optimization algorithm to use. Valid options are:

- "congra": conjugate gradient method
- "dbldog": double-dogleg method
- "lbfgs": limited-memory BFGS solver
- "newrap": Newton-Raphson method with line search and ridging
- "nmsimp": Nelder-Mead simplex method
- "nrridg": Newton-Raphson method with ridging
- "quanew": dual quasi-Newton optimization
- "duquanew": dual quasi-Newton optimization

selection : {"backward", "forward", "lasso", "stepwise"}, optional

Specifies the variable selection method to use.

verbose : int, optional

Specifies the information to print about the fitted model:

0: None 1: Iteration history 2: Iteration history and fit statistics

Attributes

classes_ : ndarray of shape (n_classes,)

Class labels seen during model fitting.

coef_ : pandas.DataFrame of shape (n_parameters,)

Model coefficients. The number of parameters may be greater than the number of features if additional parameters are introduced to handle missing values, or less than the number of features if variable selection was used.

details_ : Results

Additional information about the model fitting process and the final results.

intercept_ : pandas.DataFrame of shape (n_classes-1,)

Bias term(s) added to the model.

n_iter_ : int

The number of model fit iterations performed.

n_features_in_ : int

The number of input features seen by the model.

feature_names_in_ : list(str)

The column names of the input features seen during fitting.

target_name_in_ : str

Column name of the target variable.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>decision_function</code>	Computes the decision function given input features.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the model on the provided training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts class labels given input features.
<code>predict_proba</code>	Estimates probabilities for each class label.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots=“none”` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {“light”, “dark”, “opal”, “midnight”, “raven”, “htmlencore”, “highcontrast”, “grayscale”, “monochrome”}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {“svg”, “png”}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns**CustomResults**

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

decision_function

```
decision_function(X)
```

Computes the decision function given input features.

Parameters**X : array-like**

Specifies training features in the shape (n_samples, n_features).

Returns**array-like**

Returns the estimated posterior probability for the target class. This is (n_samples, n_classes) for the multiclass case and (n_samples,) with just `self.classes_[1]` returned for the binary case.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns**Results**

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model on the provided training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Names of columns in X that should be treated as categorical. The target is always treated as categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts class labels given input features.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted class labels in the shape (n_samples,).

predict_proba

```
predict_proba(X, copy_var=None)
```

Estimates probabilities for each class label.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns estimated class label probabilities in the shape (n_samples, n_classes).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Ridge

Linear Regression with L2 regularization.

API: Ridge

```
class sasviya.ml.linear_model.Ridge(alpha=None, *, fit_intercept=True,  
tol=1e-08, solver='lbfgs', verbose=0)
```

Parameters

alpha : float, optional

Specifies the strength of the regularization that is applied. A constant value is used to scale the regularization term of the cost function. If None is specified, the regularization term is computed from the data. “Alpha” is also referred to as “lambda” in fit results.

fit_intercept : bool, optional

Specifies whether or not to include a bias term in the model.

solver : {“admm”, “bfgs”, “lbfgs”, “nlp”}, optional

Specifies the optimization algorithm to use.

tol : float, optional

Specifies the tolerance threshold for early stopping. Model fitting stops when the size of the gradient is less than the value of *tol*.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

coef_ : ndarray of shape (n_parameters,)

Specifies the model coefficients. The number of parameters can be greater than the number of features if additional parameters are introduced to handle missing values, or less than the number of features if variable selection is used.

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

intercept_ : ndarray of shape (, n_intercepts)

Specifies the bias term(s) to be added to the model.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts continuous values given input features.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method.

You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` or `exclude_tables` parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the `order` parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Nominal Dimension Reduction

Classes 121

Classes

Class Name	Description
LPCA	Nominal Variables Dimension Reduction with Logistic Principal Component Analysis
MCA	Nominal Variables Dimension Reduction with Multiple Correspondence Analysis

LPCA

Nominal Variables Dimension Reduction with Logistic Principal Component Analysis

API: LPCA

```
class sasviya.ml.nominaldr.LPCA(dimensions, prefix='rv', epsilon=0.0001,
max_iter=100, m=4, verbose=0)
```

This object performs nominal variables dimension reduction via the logistic principal component analysis (LPCA) method.

Parameters

dimensions : int

Specifies the number of reduced variables.

prefix : str, optional

Specifies a prefix to apply to the names of the reduced variables. The default is 'rv'.

epsilon : float, optional

Specifies the tolerance to use in determining the convergence of the iterative algorithm. The default is 0.0001.

max_iter : int, optional

Specifies the maximum number of iterations for the iterative algorithm. The default is 100.

m : int, optional

Specifies a finite positive value to approximate the logit function's infinite limits. $\text{logit}(1)$ is approximated by m , and $\text{logit}(0)$ is approximated by $-m$. The default is 4.

verbose : int, optional

Specifies the amount of information to print about the fitted model. The default is 0. A value of 0 prints nothing. A value of 1 prints the model information table to the log. A value of 2 prints the model information and number of observations tables to the log. A value of 3 prints the model information, number of observations, and variable information tables to the log. A value of 4 prints the model information, number of observations, variable information, and iteration history tables to the log. A value of 5 prints the model information, number of observations, variable information, iteration history, and explained variance summary tables to the log.

Attributes

details_ : Results

Stores additional information about the model fitting process and the final results.

dimensions_ : int

Stores the number of reduced variables (which can be less than the specified parameter dimensions).

n_features_ : int

Stores the number of features that appear in the training data.

n_features_in_ : int

Stores the number of input nominal features that the model uses.

n_samples_ : int

Stores the number of samples that appear in the training data.

n_samples_in_ : int

Stores the number of samples that the model uses.

n_all_categories_in_ : int

Stores the total number of categories for all the input nominal features.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fits the model to the training data.
fit_transform	Fits a transformation model to data set X and then runs the transformation model on that data set.
get_params	Gets parameters for this model.
save	Saves the model to a file.
set_params	Updates the parameters of the model.
transform	Applies the model to new input data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
theme=None, path=None, output_type=None, include_tables=None,
exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y=None, nominals=None, freq=None, id=None)
```

Fits the model to the training data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the training data, where *n_samples* is the number of samples and *n_features* is the number of features.

y : any

This parameter is ignored (included for API consistency only).

nominals: str | iterable of str | None, optional

Specifies the columns in the input to the X parameter to treat as nominal data.

freq : str | None, optional

Specifies the column in the input to the X parameter that contains the frequency of each observation.

id: str | iterable of str | None, optional

Specifies the columns in the input to the X parameter that contain the ID of each observation.

Returns

self : object

Returns the instance itself.

fit_transform

```
fit_transform(X, y=None, **kwargs)
```

Fits a transformation model to data set X and then runs the transformation model on that data set.

Parameters

X : array-like

Specifies the input data set that is used to train the model.

y : array-like, optional

Specifies the target data set that is used to train the model. This parameter is ignored if the transformation model does not use a target data set.

Returns

array-like

Returns the X data set after it has been transformed.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

transform

```
transform(X)
```

Applies the model to new input data.

Parameters

X : array-like

Specifies new input data.

Returns

array-like

Returns a table that contains the reduced data for the input to the X parameter.

MCA

Nominal Variables Dimension Reduction with Multiple Correspondence Analysis

API: MCA

```
class sasviya.ml.nominaldr.MCA(dimensions, prefix='rv', verbose=0)
```

This object performs nominal variables dimension reduction via the multiple correspondence analysis (MCA) method.

Parameters

dimensions : int

Specifies the number of reduced variables.

prefix : str, optional

Specifies a prefix to apply to the names of the reduced variables. The default is 'rv'.

verbose : int, optional

Specifies the amount of information to print about the fitted model. The default is 0. A value of 0 prints nothing. A value of 1 prints the model information table to the log. A value of 2 prints the model information and number of observations tables to the log. A value of 3 prints the model information, number of observations, and variable information tables to the log. A value of 4 prints the model information, number of observations, variable information, and explained variance summary tables to the log.

Attributes

details_ : Results

Stores additional information about the model fitting process and the final results.

dimensions_ : int

Stores the number of reduced variables (which can be less than the specified parameter dimensions).

n_features_ : int

Stores the number of features that appear in the training data.

n_features_in_ : int

Stores the number of input nominal features that the model uses.

n_samples_ : int

Stores the number of samples that appear in the training data.

n_samples_in_ : int

Stores the number of samples that the model uses.

n_all_categories_in_ : int

Stores the total number of categories for all the input nominal features.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the model to the training data.
<code>fit_transform</code>	Fits a transformation model to data set X and then runs the transformation model on that data set.
<code>get_params</code>	Gets parameters for this model.
<code>save</code>	Saves the model to a file.
<code>set_params</code>	Updates the parameters of the model.
<code>transform</code>	Applies the model to new input data.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with

the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y=None, nominals=None, freq=None, id=None)
```

Fits the model to the training data.

Parameters

X : array-like of shape (n_samples, n_features)

Specifies the training data, where *n_samples* is the number of samples and *n_features* is the number of features.

y : any

This parameter is ignored (included for API consistency only).

nominals: str | iterable of str | None, optional

Specifies the columns in the input to the X parameter to treat as nominal data.

freq : str | None, optional

Specifies the column in the input to the X parameter that contains the frequency of each observation.

id: str | iterable of str | None, optional

Specifies the columns in the input to the X parameter that contain the ID of each observation.

Returns

self : object

Returns the instance itself.

fit_transform

```
fit_transform(X, y=None, **kwargs)
```

Fits a transformation model to data set X and then runs the transformation model on that data set.

Parameters

X : array-like

Specifies the input data set that is used to train the model.

y : array-like, optional

Specifies the target data set that is used to train the model. This parameter is ignored if the transformation model does not use a target data set.

Returns

array-like

Returns the X data set after it has been transformed.

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

transform

```
transform(X)
```

Applies the model to new input data.

Parameters

X : array-like

Specifies new input data.

Returns

array-like

Returns a table that contains the reduced data for the input to the X parameter.

Support Vector Models

Classes 137

Classes

Class Name	Description
SVC	Support vector classification model.
SVR	Support vector regression model.

SVC

Support vector classification model.

API: SVC

```
class sasviya.ml.svm.SVC(*, max_iter=25, C=1.0, kernel='linear', degree=2,
sigma='auto', coef0=-1, method=None, tol=None, random_state=1, scale=True,
verbose=0)
```

Parameters

max_iter : int, optional

Specifies the maximum number of training iterations.

C : float, optional

Specifies the cost regularization parameter.

kernel : {"linear", "poly", "polynomial", "rbf", "sigmoid"}, optional

Specifies the type of kernel to use.

degree : int, optional

Specifies the degree of the polynomial kernel. This parameter is ignored if the *kernel* parameter value is not "polynomial".

sigma : "auto", "scale" or float, optional

Specifies the kernel parameter for the RBF and Sigmoid kernels.

coef0 : float, optional

Specifies the second parameter for the Sigmoid kernel.

method : {"activeset", "cd", "ipoint"} or None, optional

Specifies the optimization method to be used to fit the model.

- The "activeset" method can be used with all kernels and is the default for the RBF and Sigmoid kernels, as well as for polynomial kernels of more than degree 3.
- The "ipoint" (interior point) method can be used with the linear and polynomial kernels and is the default for linear kernels and polynomial kernels of degree 3 or less.
- Coordinate descent can be used only with the linear kernel.

tol : float, optional

Specifies the tolerance threshold for early stopping.

random_state : int, optional

Specifies the seed to use for random number generation.

scale : bool, optional

When set to True, scales the numeric variables prior to model fitting.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

classes_ : ndarray of shape (2,)

Specifies the class labels that are seen during model fitting.

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

n_iter_ : int

Specifies the number of training iterations. This attribute is present only when the `ipoint` method is used.

n_support_ : int

Specifies the number of support vectors. This attribute is not present if the `method` parameter value is "cd".

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
decision_function	Computes the decision function given input features.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fits the model to the training data.
get_params	Gets parameters for this model.
predict	Predicts class labels given input features.
predict_proba	Estimates probabilities for each class label.
save	Saves the model to a file.
score	Determines model accuracy by using the data set X and the targets y.
set_params	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left

corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

decision_function

```
decision_function(X)
```

Computes the decision function given input features.

Parameters

X : array-like

Specifies training features in the shape (n_samples, n_features).

Returns

array-like

Returns the estimated posterior probability for the target class. This is (n_samples, n_classes) for the multiclass case and (n_samples,) with just `self.classes_[1]` returned for the binary case.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as nominal values.
For classifier models, the target is assumed to be categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts class labels given input features.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted class labels in the shape (n_samples,).

predict_proba

```
predict_proba(X, copy_var=None)
```

Estimates probabilities for each class label.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns estimated class label probabilities in the shape (n_samples, n_classes).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

SVR

Support vector regression model.

API: SVR

```
class sasviya.ml.svm.SVR(*, max_iter=25, C=1.0, kernel='linear', degree=2,  
tol=None, epsilon=0.01, random_state=1, scale=True, verbose=0)
```

Parameters

max_iter : int, optional

Specifies the maximum number of training iterations.

C : float, optional

Specifies the cost regularization parameter.

kernel : {"linear", "poly", "polynomial"}, optional

Specifies the type of kernel to use.

degree : int, optional

Specifies the degree of the polynomial kernel. This parameter is used only when the *kernel* parameter value is "polynomial".

tol : float, optional

Specifies the tolerance threshold for early stopping.

epsilon : float, optional

Specifies the epsilon-tube distance. Points that have a predicted value within this distance from their actual value are not counted toward the training loss.

random_state : int, optional

Specifies the seed to use for random number generation.

scale : bool, optional

When set to True, scales the numeric variables prior to model fitting.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
<code>customize_results</code>	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
<code>describe</code>	Describes the type of model, its inputs, and its output.
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fits the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts continuous values given input features.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                  theme=None, path=None, output_type=None, include_tables=None,
                  exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method.

You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the `exclude_tables` parameter, then the `include_tables` parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` parameter, then the `include_tables` parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the `include_tables` or `exclude_tables` parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the `order` parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fits the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as nominal values.
For classifier models, the target is assumed to be categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Tree Models

<i>Classes</i>	153
----------------------	-----

Classes

Class Name	Description
DecisionTreeClassifier	A decision tree classification model.
DecisionTreeRegressor	A decision tree regression model.
ForestClassifier	A random forest classification model.
ForestRegressor	A random forest regression model.
GradientBoostingClassifier	Gradient boosting classifier.
GradientBoostingRegressor	Gradient boosting regression model.

DecisionTreeClassifier

A decision tree classification model.

API: DecisionTreeClassifier

```
class sasviya.ml.tree.DecisionTreeClassifier(criterion='IGR', max_depth=10,
min_samples_leaf=5, ccp_alpha=0, verbose=0)
```

Parameters

criterion : {"chaid", "chisquare", "entropy|gain", "igr|gainratio", "gini"}

Specifies the split criterion for each tree node.

max_depth : int, optional

Specifies the maximum depth of the tree.

min_samples_leaf : int, optional

Specifies the minimum number of observations on each node.

ccp_alpha : float, optional

Specifies the value to use for minimal cost-complexity pruning for regression trees.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

details_ : Results

Specifies additional information about the model fitting process and the final results.

n_features_in_ : int

Specifies the number of input features that are seen by the model. This can be a subset of the input columns if variable selection is used.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during fitting.

n_features_used_ : int

Specifies the number of input features to be used by the model.

target_name_in_ : str

Specifies the column name of the target variable.

n_nodes_ : int

Specifies the number of tree nodes.

max_n_branches_ : int

Specifies the maximum number of branches of the tree.

depth_ : int

Specifies the depth of the tree.

n_leaves_ : int

Specifies the number of leaves of the tree.

n_bins_ : int

Specifies the number of bins for numeric variables.

min_n_leaves_ : int

Specifies the minimum size of leaves.

max_n_leaves_ : int

Specifies the maximum size of leaves.

n_obs_ : int

Specifies the number of observations to be used by the model.

classes_ : list(str)

Specifies the class labels.

n_classes_ : int

Specifies the number of classes.

feature_importances_ : pandas.DataFrame()

Specifies the ranking of features by importance.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
decision_function	Computes the decision function given input features.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fit the model to the training data.
get_params	Gets parameters for this model.
predict	Predicts class labels given input features.
predict_proba	Estimates probabilities for each class label.
save	Saves the model to a file.
score	Determines model accuracy by using the data set X and the targets y.

Method Name	Description
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                  theme=None, path=None, output_type=None, include_tables=None,
                  exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see SAS Viya: Managing Plots in Python Automatic Graphics.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is “svg”, which contains image map information for tooltips. The “png” output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword (“none”, “all”), a single table, or a list of tables. The default is “all”, which keeps all tables. The keyword “none” removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword (“none”, “graph_tables”, “all”), a single table, or a list of tables. The default is “none”, which displays all tables. The keyword “graph_tables” removes all tables that were used to produce a graph. The keyword “all” removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

decision_function

```
decision_function(X)
```

Computes the decision function given input features.

Parameters

X : array-like

Specifies training features in the shape (n_samples, n_features).

Returns

array-like

Returns the estimated posterior probability for the target class. This is (n_samples, n_classes) for the multiclass case and (n_samples,) with just self.classes_[1] returned for the binary case.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None, weight=None)
```

Fit the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical. For classifier models, the target is assumed to be categorical.

weight : str, optional

Specifies the name of a column in X that contains the weight of each observation.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts class labels given input features.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted class labels in the shape (n_samples,).

predict_proba

```
predict_proba(X, copy_var=None)
```

Estimates probabilities for each class label.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns estimated class label probabilities in the shape (n_samples, n_classes).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

DecisionTreeRegressor

A decision tree regression model.

API: DecisionTreeRegressor

```
class sasviya.ml.tree.DecisionTreeRegressor(criterion='RSS', max_depth=10,  
min_samples_leaf=5, ccp_alpha=0, verbose=0)
```

Parameters

criterion : {"chaid", "ftest", "variance|rss"}

Specifies the criterion to be used to measure the quality of a split in each tree node.

max_depth : int, optional

Specifies the maximum depth of any tree.

min_samples_leaf : int, optional

Specifies the minimum number of observations on each node.

ccp_alpha : float, optional

Specifies the alpha value to use for minimal cost-complexity pruning for regression trees.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

details_ : Results

Specifies additional information about the model fitting process and the final results.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during fitting.

n_features_used_ : int

Specifies the number of input features to be used by the model.

target_name_in_ : str

Specifies the column name of the target variable.

n_nodes_ : int

Specifies the number of tree nodes.

max_n_branches_ : int

Specifies the maximum number of branches of the tree.

depth_ : int

Specifies the depth of the tree.

n_leaves_ : int

Specifies the number of leaves of the tree.

n_bins_ : int

Specifies the number of bins for numeric variables.

min_n_leaves_ : int

Specifies the minimum size of leaves.

max_n_leaves_ : int

Specifies the maximum size of leaves.

n_obs_ : int

Specifies the number of observations to be used by the model.

feature_importances_ : pandas.DataFrame()

Specifies the ranking of features by importance.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fit the model to the training data.
get_params	Gets parameters for this model.
predict	Predicts continuous values given input features.
save	Saves the model to a file.

Method Name	Description
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see SAS Viya: Managing Plots in Python Automatic Graphics.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the *display()* method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None, weight=None)
```

Fit the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in *X* that should be treated as categorical. For classifier models, the target is assumed to be categorical.

weight : str, optional

Specifies the name of a column in *X* that contains the weight of each observation.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in *X*.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self
Returns itself.

ForestClassifier

A random forest classification model.

API: ForestClassifier

```
class sasviya.ml.tree.ForestClassifier(n_estimators=100, *, bootstrap=0.6,
criterion='IGR', max_depth=20, max_features=None, min_samples_leaf=5,
n_bins=50, oob_score=True, random_state=None, verbose=0)
```

Parameters

n_estimators : int, optional

Specifies the number of trees to create.

bootstrap : float, optional

Specifies the fraction of the data to use for the bootstrap sample.

criterion : {"chaid", "chisquare", "entropy|gain", "igr|gainratio", "gini"}

Specifies the criterion to be used to measure the quality of a split in each tree node.

max_depth : int, optional

Specifies the maximum depth of any tree.

max_features : int, optional

Specifies the number of input variables to consider in splitting a node (also referred to as "M" in model output). The variables are selected at random from the input variables for each tree. By default, the value of this parameter is equal to the square root of the number of input variables.

min_samples_leaf : int, optional

Specifies the minimum number of observations at each node.

n_bins : int, optional

Specifies the number of bins to use for numeric variables.

oob_score : bool, optional

Specifies whether or not to compute the out-of-bag error during model training.

random_state : int, optional

Specifies the seed to use for random number generation.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

classes_ : list(str) with length n_classes

Specifies the class labels that are seen during model fitting.

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_importances_ : DataFrame

Specifies the feature importances to calculate during model fitting, scaled to the range [0, 1].

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

n_classes_ : int

Specifies the number of target classes.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

oob_score_ : float

Specifies the out-of-bag error that is observed during model fitting. This value is computed only if the *oob_score* parameter is set to True.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
decision_function	Computes the decision function given input features.
describe	Describes the type of model, its inputs, and its output.

Method Name	Description
<code>export</code>	Exports the internal SAS analytic store from a fitted model.
<code>fit</code>	Fit the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts class labels given input features.
<code>predict_proba</code>	Estimates probabilities for each class label.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
theme=None, path=None, output_type=None, include_tables=None,
exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the GRAPHICS_THEME environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

decision_function

```
decision_function(X)
```

Computes the decision function given input features.

Parameters

X : array-like

Specifies training features in the shape (n_samples, n_features).

Returns

array-like

Returns the estimated posterior probability for the target class. This is (n_samples, n_classes) for the multiclass case and (n_samples,) with just self.classes_[1] returned for the binary case.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fit the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in *X* that should be treated as categorical. For classifier models, the target is assumed to be categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts class labels given input features.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted class labels in the shape (n_samples,).

predict_proba

```
predict_proba(X, copy_var=None)
```

Estimates probabilities for each class label.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns**array-like**

Returns estimated class label probabilities in the shape (n_samples, n_classes).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters**path : str or pathlib.Path**

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

ForestRegressor

A random forest regression model.

API: ForestRegressor

```
class sasviya.ml.tree.ForestRegressor(n_estimators=100, *, bootstrap=0.6,
criterion='RSS', max_depth=20, max_features=None, min_samples_leaf=5,
n_bins=50, oob_score=True, random_state=None, verbose=0)
```

Parameters

n_estimators : int, optional

Specifies the number of trees to create.

bootstrap : float, optional

Specifies the fraction of the data to use for the bootstrap sample.

criterion : {"chaid", "ftest", "variance|rss"}

Specifies the criterion to be used to measure the quality of a split in each tree node.

max_depth : int, optional

Specifies the maximum depth of any tree.

max_features : int, optional

Specifies the number of input variables to consider in splitting a node (also referred to as "M" in model output). The variables are selected at random from the input variables for each tree. By default, the value of this parameter is equal to the square root of the number of input variables.

min_samples_leaf : int, optional

Specifies the minimum number of observations on each node.

n_bins : int, optional

Specifies the number of bins to use for numeric variables.

oob_score : bool, optional

Specifies whether or not to compute the out-of-bag error during model training.

random_state : int, optional

Specifies the seed to use for random number generation.

verbose : int, optional

Specifies the amount of information to print about the fitted model.

Attributes

details_ : Results

Specifies additional information about the model fitting process and the final results.

feature_importances_ : DataFrame

Specifies the feature importances to calculate during model fitting, scaled to the range [0, 1].

feature_names_in_ : list(str)

Specifies the column names of the input features that are seen during model fitting.

n_features_in_ : int

Specifies the number of input features that are seen by the model.

oob_score_ : float

Specifies the out-of-bag error that is observed during model fitting. This value is computed only if the *oob_score* parameter is set to True.

target_name_in_ : str

Specifies the column name of the target variable.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.

Method Name	Description
<code>fit</code>	Fit the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts continuous values given input features.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlcore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the GRAPHICS_THEME environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the display() method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fit the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical. For classifier models, the target is assumed to be categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns**array-like**

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters**path : str or pathlib.Path**

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

GradientBoostingClassifier

Gradient boosting classifier.

API: GradientBoostingClassifier

```
class sasviya.ml.tree.GradientBoostingClassifier(*, min_samples_leaf=5,
n_bins=50, n_estimators=100, max_depth=4, max_features=None,
learning_rate=0.1, subsample=0.5, l1_penalty=0, l2_penalty=1,
random_state=None, verbose=0, n_iter_no_change=0, tol=0,
calc_feature_importances=False, validation_fraction=0)
```

Parameters

n_bins: int, optional

Specifies the number of bins to use for numeric variables. The default is 50.

min_samples_leaf : int, optional

Specifies the minimum number of observations on each node. The default is 5.

n_estimators : int, optional

Specifies the number of trees to create. The default is 100.

max_depth : int, optional

Specifies the maximum depth of the tree. The default is 4.

max_features : float, optional

Specifies the number of input variables to consider for splitting on a node. The default is the number of input variables.

learning_rate : float, optional

Specifies the learning rate of each tree. The default is 0.1.

subsample : float, optional

Specifies the fraction of the data to use for building each tree. The default is 0.5.

l1_penalty : float, optional

Specifies the L1 norm regularization on prediction. The default is 0.

l2_penalty : float, optional

Specifies the L2 norm regularization on prediction. The default is 1.

random_state : float, optional

Specifies the seed for the random number generator. By default, the random number stream is based on the computer clock. If you want a reproducible random number sequence between runs, specify a value greater than 0.

verbose : int, optional

Specifies the amount of information to display about the fitted model.

calc_feature_importances : bool, optional

Specifies whether or not to calculate feature importances.

validation_fraction: float, optional

Specifies the fraction of training samples to use for validation. A value greater than 0 enables early stopping mode.

n_iter_no_change : int, optional

Specifies the maximum number of training iterations for early stopping. This parameter is ignored unless a validation fraction is specified.

tol : float, optional

Specifies the tolerance value for early stopping as the change relative to the previous iteration. This parameter is ignored unless a validation fraction is specified.

Attributes

details_ : Results

Specifies information about the model fitting process and the final results.

n_estimators_ : int

Specifies the number of trees in the fitted model. The number could be less than the specified value because of early stopping.

avg_n_leaves_ : float

Specifies the average number of leaves among trees in the fitted model.

min_n_leaves_ : int

Specifies the minimum number of leaves among trees in the fitted model.

max_n_leaves_ : int

Specifies the maximum number of leaves among trees in the fitted model.

max_n_nodes_ : int

Specifies the maximum number of nodes among trees in the fitted model.

min_n_nodes_ : int

Specifies the minimum number of nodes among trees in the fitted model.

max_depth_ : int

Specifies the maximum depth among trees in the fitted model.

min_depth_ : int

Specifies the minimum depth among trees in the fitted model.

min_samples_leaf_ : int

Specifies the minimum leaf size among trees in the fitted model.

max_samples_leaf_ : int

Specifies the maximum leaf size among trees in the fitted model.

n_features_in_ : int

Specifies the number of input features.

feature_names_in_ : list(str)

Specifies the column names of the input features.

n_features_used_ : int

Specifies the number of features to be used by the fitted model.

target_name_in_ : str

Specifies the column name of the target variable.

classes_ : list(str)

Specifies the class labels.

n_classes_ : int

Specifies the number of classes of the target variable.

feature_importances_ : pandas.DataFrame()

Specifies the ranking of features by importance.

distribution_ : str

Specifies the distribution function of the fitted model.

validation_scores_ : pandas.DataFrame()

Specifies the misclassification rate for the validation sample by iteration.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
decision_function	Computes the decision function given input features.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.

Method Name	Description
<code>fit</code>	Fit the model to the training data.
<code>get_params</code>	Gets parameters for this model.
<code>predict</code>	Predicts class labels given input features.
<code>predict_proba</code>	Estimates probabilities for each class label.
<code>save</code>	Saves the model to a file.
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
                 theme=None, path=None, output_type=None, include_tables=None,
                 exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see SAS Viya: Managing Plots in Python Automatic Graphics.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlcore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is "light". You can specify the session default by setting the GRAPHICS_THEME environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the `display()` method for viewing in a Jupyter notebook cell.

decision_function

```
decision_function(X)
```

Computes the decision function given input features.

Parameters

X : array-like

Specifies training features in the shape (n_samples, n_features).

Returns

array-like

Returns the estimated posterior probability for the target class. This is (n_samples, n_classes) for the multiclass case and (n_samples,) with just self.classes_[1] returned for the binary case.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fit the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical. For classifier models, the target is assumed to be categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts class labels given input features.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted class labels in the shape (n_samples,).

predict_proba

```
predict_proba(X, copy_var=None)
```

Estimates probabilities for each class label.

Parameters

X : array-like

Specifies the samples to predict class labels for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns**array-like**

Returns estimated class label probabilities in the shape (n_samples, n_classes).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters**path : str or pathlib.Path**

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

GradientBoostingRegressor

Gradient boosting regression model.

API: GradientBoostingRegressor

```
class sasviya.ml.tree.GradientBoostingRegressor(*, min_samples_leaf=5,
n_bins=50, n_estimators=100, max_depth=4, max_features=None,
learning_rate=0.1, subsample=0.5, l1_penalty=0, l2_penalty=1,
random_state=None, verbose=0, n_iter_no_change=0, tol=0,
calc_feature_importances=False, validation_fraction=0)
```

Parameters

n_bins: int, optional

Specifies the number of bins to use for numeric variables. The default is 50.

min_samples_leaf : int, optional

Specifies the minimum number of observations on each node. The default is 5.

n_estimators : int, optional

Specifies the number of trees to create. The default is 100.

max_depth : int, optional

Specifies the maximum depth of the tree. The default is 4.

max_features : float, optional

Specifies the number of input variables to consider for splitting on a node. The default is the number of input variables.

learning_rate : float, optional

Specifies the learning rate of each tree. The default is 0.1.

subsample : float, optional

Specifies the fraction of the data to use for building each tree. The default is 0.5.

l1_penalty : float, optional

Specifies the L1 norm regularization on prediction. The default is 0.

l2_penalty : float, optional

Specifies the L2 norm regularization on prediction. The default is 1.

random_state : float, optional

Specifies the seed for the random number generator. By default, the random number stream is based on the computer clock. If you want a reproducible random number sequence between runs, specify a value greater than 0.

verbose : int, optional

Specifies the amount of information to display about the fitted model.

calc_feature_importances : bool, optional

Specifies whether or not to calculate feature importances.

validation_fraction: float, optional

Specifies the fraction of training sample to use for validation. A value greater than 0 enables early stopping mode.

n_iter_no_change : int, optional

Specifies the maximum number of training iterations for early stopping. This parameter is ignored unless a validation fraction is specified.

tol : float, optional

Specifies the tolerance value for early stopping as the change relative to the previous iteration. This parameter is ignored unless a validation fraction is specified.

Attributes

details_ : Results

Specifies information about the model fitting process and the final results.

n_estimators_ : int

Specifies the number of trees in the fitted model. The number could be less than the specified value because of early stopping.

avg_n_leaves_ : float

Specifies the average number of leaves among trees in the fitted model.

min_n_leaves_ : int

Specifies the minimum number of leaves among trees in the fitted model.

max_n_leaves_ : int

Specifies the maximum number of leaves among trees in the fitted model.

max_n_nodes_ : int

Specifies the maximum number of nodes among trees in the fitted model.

min_n_nodes_ : int

Specifies the minimum number of nodes among trees in the fitted model.

max_depth_ : int

Specifies the maximum depth among trees in the fitted model.

min_depth_ : int

Specifies the minimum depth among trees in the fitted model.

min_samples_leaf_ : int

Specifies the minimum leaf size among trees in the fitted model.

max_samples_leaf_ : int

Specifies the maximum leaf size among trees in the fitted model.

n_features_in_ : int

Specifies the number of input features.

feature_names_in_ : list(str)

Specifies the column names of the input features.

n_features_used_ : int

Specifies the number of features to be used by the fitted model.

target_name_in_ : str

Specifies the column name of the target variable.

feature_importances_ : pandas.DataFrame()

Specifies the ranking of features by importance.

distribution_ : str

Specifies the distribution function of the fitted model.

validation_scores_ : pandas.DataFrame()

Specifies the average square error for the validation sample by iteration.

Methods

Method Name	Description
customize_results	Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.
describe	Describes the type of model, its inputs, and its output.
export	Exports the internal SAS analytic store from a fitted model.
fit	Fit the model to the training data.
get_params	Gets parameters for this model.
predict	Predicts continuous values given input features.
save	Saves the model to a file.

Method Name	Description
<code>score</code>	Determines model accuracy by using the data set X and the targets y.
<code>set_params</code>	Updates the parameters of the model.

customize_results

```
customize_results(plots=None, width=None, height=None, dpi=None,
theme=None, path=None, output_type=None, include_tables=None,
exclude_tables=None, order=None)
```

Generates plots for all actions that support graphics, as well as filter tables from the results list, and creates custom result orders.

Parameters

plots : str or str list

Requests plots. You can request a single plot or a list of plots. The request can also be “all” or “none”. Arguments in parentheses are also allowed with each plot request. In addition, you can disable automatic plots from models by setting the environment variable `AUTO_GRAPHICS` to false. However, this variable does not disable automatic plot generation through the `customize_results()` method. You need to specify `plots="none"` in the method to guarantee that only tables are returned. For more information, see *SAS Viya: Managing Plots in Python Automatic Graphics*.

width : int

Overrides the default width of the graph in pixels.

height : int

Overrides the default height of the graph in pixels.

dpi : int

Overrides the default resolution of the graph in dots per inch. The default is 96 dpi.

theme : {"light", "dark", "opal", "midnight", "raven", "htmlencore", "highcontrast", "grayscale", "monochrome"}

Overrides the default theme that the graph uses. The default is “light”. You can specify the session default by setting the `GRAPHICS_THEME` environment variable.

path : str

Writes the plot results to the specified path in addition to the notebook.

output_type : {"svg", "png"}

Overrides the default output type. The default type is "svg", which contains image map information for tooltips. The "png" output type does not support tooltip information.

include_tables : str or str list

Includes only the specified tables from the displayed results. You can specify a keyword ("none", "all"), a single table, or a list of tables. The default is "all", which keeps all tables. The keyword "none" removes all tables, so that the results contain only plots that were generated. You can include tables by name (case-sensitive), and wildcarding is supported. These names appear in the top left corner of all displayed results. If you specify this parameter along with the *exclude_tables* parameter, then the *include_tables* parameter takes precedence.

exclude_tables : str or str list

Excludes tables from the displayed results. You can specify a keyword ("none", "graph_tables", "all"), a single table, or a list of tables. The default is "none", which displays all tables. The keyword "graph_tables" removes all tables that were used to produce a graph. The keyword "all" removes all tables, so that the results contain only plots that were generated. You can also exclude tables by name (case-sensitive), and wildcarding is supported. Those names appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* parameter, then the *include_tables* parameter takes precedence.

order : str or str list

Orders plot and tabular results for display. Any result that you do not specify in this parameter is excluded from the new results. The names for the results appear in the top left corner of all displayed results. If you specify this parameter along with the *include_tables* or *exclude_tables* parameter, those parameters are applied before the ordering is done. If the ordered list contains no results, then the *order* parameter is ignored.

Returns

CustomResults

Returns CustomResults that are a combination of both model results and plot results. These results do not replace the original model results. Instead, the CustomResults are typically assigned to a variable for later use or passed directly to the *display()* method for viewing in a Jupyter notebook cell.

describe

```
describe()
```

Describes the type of model, its inputs, and its output.

Returns

Results

Returns a mapping of various properties that describe aspects of the model.

export

```
export(file=None, replace=False)
```

Exports the internal SAS analytic store from a fitted model.

Generates an analytic store file that is common to other SAS products or procedures.

Parameters

file : str, pathlib.Path, or file-like, optional

Specifies the location where the exported model is saved. The location can be a path that specifies a file or an existing file-like object. It must be writable in binary format.

replace : bool, optional

Specifies whether or not to overwrite the *file* parameter if it already exists. This parameter is ignored unless the *file* parameter value is a file path.

Returns

bytes or None

Returns the analytic store in binary form if the *file* parameter is omitted. Otherwise None is returned.

fit

```
fit(X, y, nominals=None)
```

Fit the model to the training data.

Parameters

X : array-like

Specifies the training features in the shape (n_samples, n_features).

y : array-like

Specifies the target labels in the shape (n_samples,).

nominals : list(str), optional

Specifies the names of columns in X that should be treated as categorical. For classifier models, the target is assumed to be categorical.

Returns

self

get_params

```
get_params(deep=True)
```

Gets parameters for this model.

Parameters

deep : bool, default=True

This parameter is unused but is provided for compatibility with scikit-learn.

Returns

dict

Returns parameter names and current values.

predict

```
predict(X, copy_var=None)
```

Predicts continuous values given input features.

Parameters

X : array-like

Specifies the samples to predict values for, with the shape (n_samples, n_features).

copy_var : str or list of str, optional

Specifies variable(s) to be copied from the data. Can be a single variable name as a string or a list of variable names. For DuckDB queries, these variables will be added to the query if they don't already exist in X.

Returns

array-like

Returns predicted continuous values in the shape (n_samples,).

save

```
save(path)
```

Saves the model to a file.

Save uses pickle and can be loaded with the `sasviya.load_model`.

Parameters

path : str or pathlib.Path

Specifies the location where the model is to be saved.

Returns

None

score

```
score(X, y)
```

Determines model accuracy by using the data set X and the targets y.

set_params

```
set_params(**params)
```

Updates the parameters of the model.

Parameters

****params : dict**

Specifies name:value pairs of the parameters to update.

Returns

self

Returns itself.

Examples

<i>DecisionTreeClassifier</i>	204
DecisionTreeClassifier and Others from SAS® Viya® on Heart Disease	204
<i>DecisionTreeRegressor</i>	222
DecisionTreeRegressor from SAS® Viya® on Bike Sharing Demand	222
<i>ForestClassifier</i>	234
ForestClassifier from SAS® Viya® on Handwritten Digits	235
<i>ForestRegressor</i>	240
ForestRegressor from SAS® Viya® on Bike Sharing Demand	240
<i>ElasticNet</i>	252
Train an Elastic Net Regression Model	252
<i>Lasso</i>	255
Train a Lasso Regression Model	255
<i>LinearRegression</i>	259
Train a Linear Regression Model	259
<i>Ridge</i>	262
Train a Ridge Regression Model	262
<i>GradientBoostingClassifier</i>	266
GradientBoostingClassifier from SAS® Viya® on Banking	266
<i>GradientBoostingRegressor</i>	271
GradientBoostingRegressor from SAS® Viya® on Bike Sharing Demand	271
<i>LogisticRegression</i>	283
LogisticRegression from SAS Viya on Heart Disease	283
<i>PCA</i>	293
PCA from SAS® Viya® on Iris	293
<i>SVC</i>	296
svm.SVC from SAS® Viya® on Breast Cancer with Grid Search	296
<i>SVR</i>	302
svm.SVR from SAS® Viya® to Predict Bonuses	303

DecisionTreeClassifier

Enables you to fit a decision tree model to your data and predict new class labels and probabilities.

DecisionTreeClassifier and Others from SAS® Viya® on Heart Disease

Source

This example is adapted from [Heart Disease prediction Random forest Classifier](#) by Mau Rua.

Data Preparation

About the data set

The original data contains 76 different attributes of patients from four different hospital databases. The goal is to determine if the attributes can be used to predict whether patients are diagnosed with heart disease. However, this data has been subset to contain only 14 factors from only the Cleveland database. Here are the variables included and their interpretations:

- age
- trestbps: resting blood pressure
- chol: serum cholesterol
- thalch: maximum heart rate achieved
- ca: number of major vessels (0-3) colored by fluoroscopy
- sex
- cp: chest pain type
- exang: exercise-induced angina
- slope: slope of the peak exercise ST segment
- thal: thalassemia (3 = normal; 6 = fixed defect; 7 = reversable defect)
- restecg: resting electrocardiographic results

- fbs: fasting blood sugar
- target: diagnosis of heart disease
- oldpeak: ST depression induced by exercise relative to rest

Importing the data set

This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this data set is to predict based on the given attributes of a patient that whether that particular person has heart disease or not and other is the experimental task to diagnose and find out various insights from this data set, which could help in understanding the problem more.

The `pd.read_csv()` function reads the `heart_disease.csv` file and loads it to a DataFrame named `heart_df`. The `shape` method prints the number of rows and columns in the DataFrame.

```
import os
import pandas as pd
import numpy as np
import warnings

from sasviya.ml.linear_model import LogisticRegression
from sasviya.ml.tree import DecisionTreeClassifier, ForestClassifier,
GradientBoostingClassifier

import matplotlib.pyplot as plt
import seaborn as sn

from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)

heart_df = pd.read_csv("https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/heart_disease.csv")
print(heart_df.shape)

(304, 14)

print(heart_df.head(3))
```

	age	trestbps	chol	thalch	oldpeak	...	slope	thal	restecg	fbs	target
0	63	145	233	150	2.3	...	0	1	0	1	0
1	67	160	286	108	1.5	...	1	2	0	0	1
2	67	120	229	129	2.6	...	1	0	0	0	1

[3 rows x 14 columns]

Data Preprocessing

Print information about the data set

Start by getting some general characteristics about the data set. The `info()` method prints information about the `heart_df` DataFrame.

```
print(heart_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 304 entries, 0 to 303
Data columns (total 14 columns):
#   Column      Non-Null Count  Dtype
---  -
0   age         304 non-null    int64
1   trestbps    304 non-null    int64
2   chol        304 non-null    int64
3   thalch      304 non-null    int64
4   oldpeak     304 non-null    float64
5   ca          299 non-null    float64
6   sex         304 non-null    int64
7   cp          304 non-null    int64
8   exang       304 non-null    int64
9   slope       304 non-null    int64
10  thal        304 non-null    int64
11  restecg     304 non-null    int64
12  fbs         304 non-null    int64
13  target      304 non-null    int64
dtypes: float64(2), int64(12)
memory usage: 33.4 KB
```

Replacing NaN values with mean

The data must be preprocessed before running the algorithm. To address missing values in the data, the following code calculates the sum of missing values for each column, identifies columns with missing values, replaces the missing values with the mean of the nonmissing values in their respective columns, and then counts the sum of the missing values in each column to confirm there are no missing values.

```
cols_sum_null = heart_df.isnull().sum()
print('Missing value counts (before imputation)\n', cols_sum_null)
```

```

age      0
trestbps 0
chol     0
thalch   0
oldpeak  0
ca       0
sex      0
cp       0
exang    0
slope    0
thal     0
restecg  0
fbs      0
target   0
dtype: int64

```

```

hasnull_cols = cols_sum_null[cols_sum_null != 0]
for col in hasnull_cols.index:
    mean = heart_df[col].mean()
    heart_df[col].fillna(mean, inplace=True)
print('Missing value counts (after imputation)\n',
      heart_df.isnull().sum())

```

```

Missing value counts (after imputation)
age      0
trestbps 0
chol     0
thalch   0
oldpeak  0
ca       0
sex      0
cp       0
exang    0
slope    0
thal     0
restecg  0
fbs      0
target   0
dtype: int64

```

Visualizing the Data

To help understand the data, graphs will be generated, and additional analysis will be performed.

- a histogram of target for "Heart Disease" versus "No disease"
- a pie chart visualization of the same information
- a pairplot of quantitative features
- a heat map of feature correlations

```

disease = len(heart_df[heart_df['target'] == 1])
no_disease = len(heart_df[heart_df['target'] == 0])

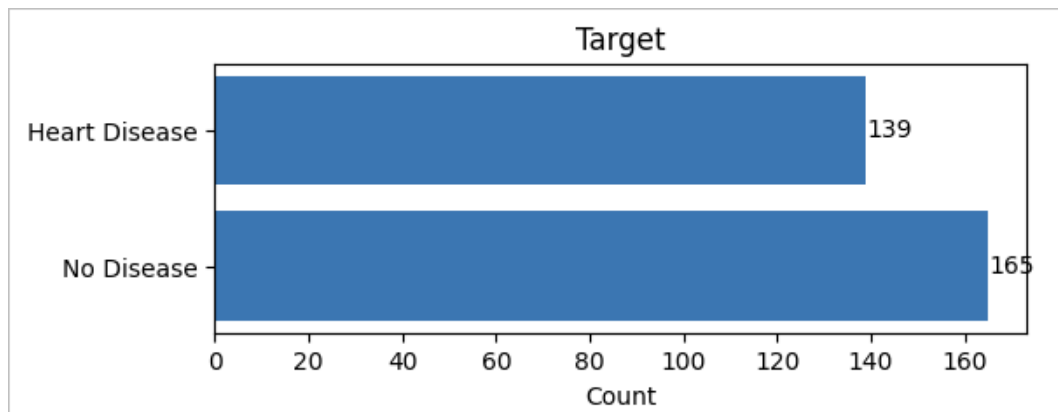
x = [disease, no_disease]
y = ['Heart Disease', 'No Disease']

```

```

fig, ax = plt.subplots(figsize=(6, 2))
y_pos = np.arange(len(y))
ax.barh(y_pos, x, align='center')
ax.set_yticks(y_pos)
ax.set_yticklabels(y)
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Count')
ax.set_title('Target')
for i, v in enumerate(x):
    ax.text(v, i, str(v), color='black', va='center',
            fontweight='normal')
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)

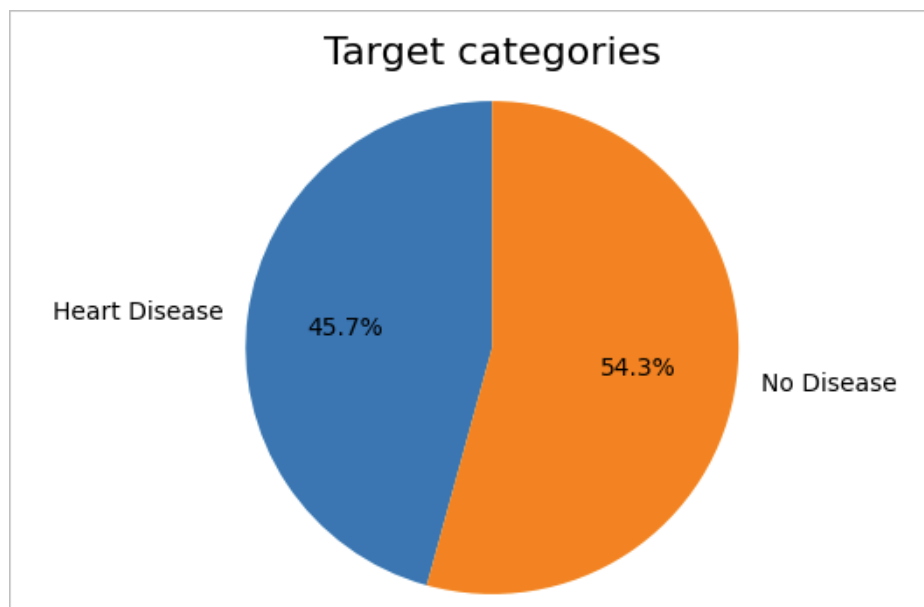
```



```

fig1, ax1 = plt.subplots(figsize=(4, 4))
ax1.pie(x, labels=y, autopct='%1.1f%%', startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.title('Target categories', size=16)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)

```



```

#identify quantitative (continuous) features based on number of unique
categories
cat_threshold = 8

```

```

qualitative = [c for c in heart_df.columns if
len(heart_df[c].unique()) <= cat_threshold]
quantitative = [c for c in heart_df.columns if
len(heart_df[c].unique()) > cat_threshold]

print('Qualitative features:\n', qualitative)

```

```

Qualitative features:
['ca', 'sex', 'cp', 'exang', 'slope', 'thal', 'restecg', 'fbs', 'target']

```

```

print('Quantitative features:\n', quantitative)

```

```

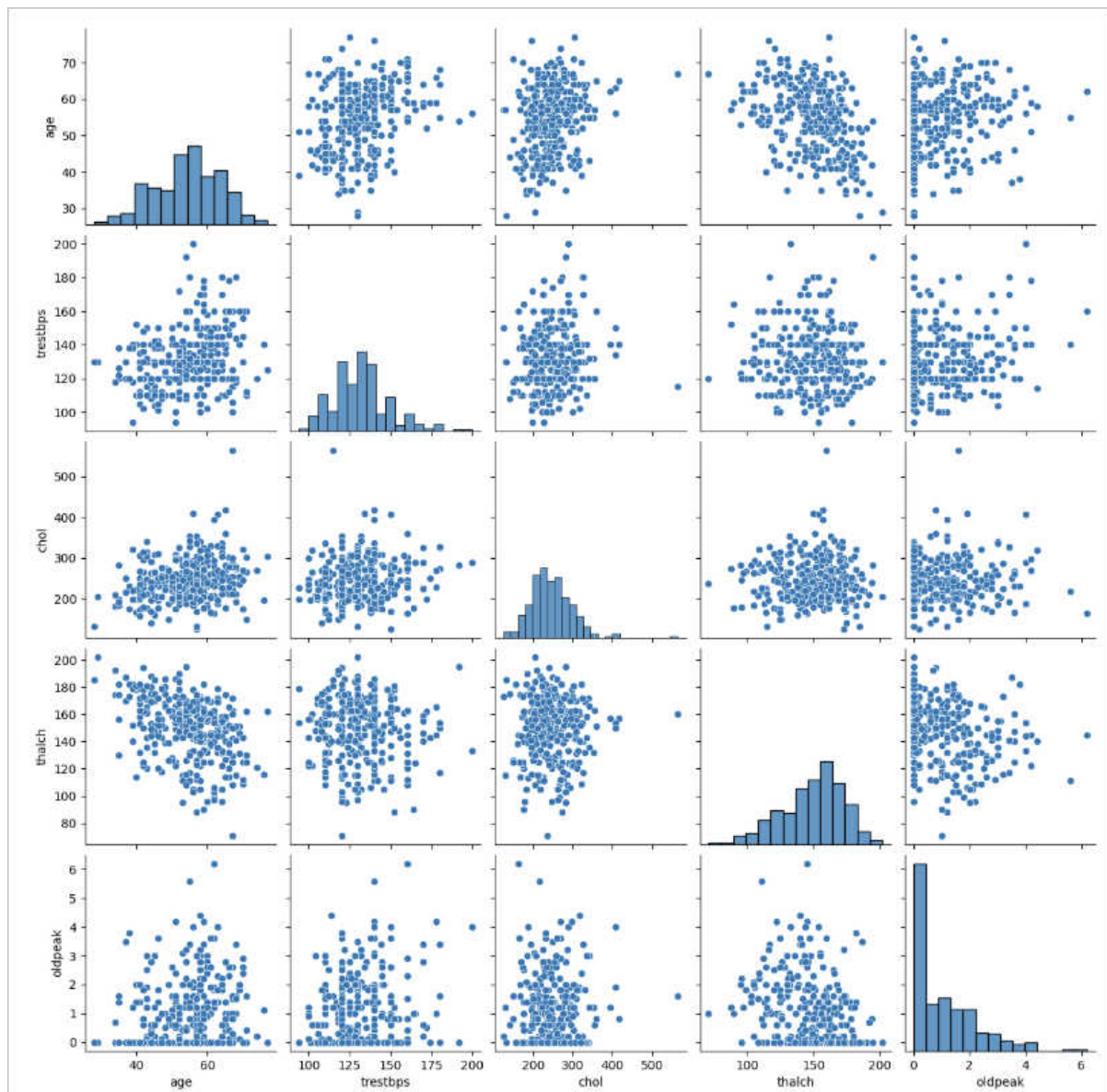
Quantitative features:
['age', 'trestbps', 'chol', 'thalch', 'oldpeak']

```

```

sn.pairplot(heart_df[quantitative])
SAS.pyplot(plt)

```

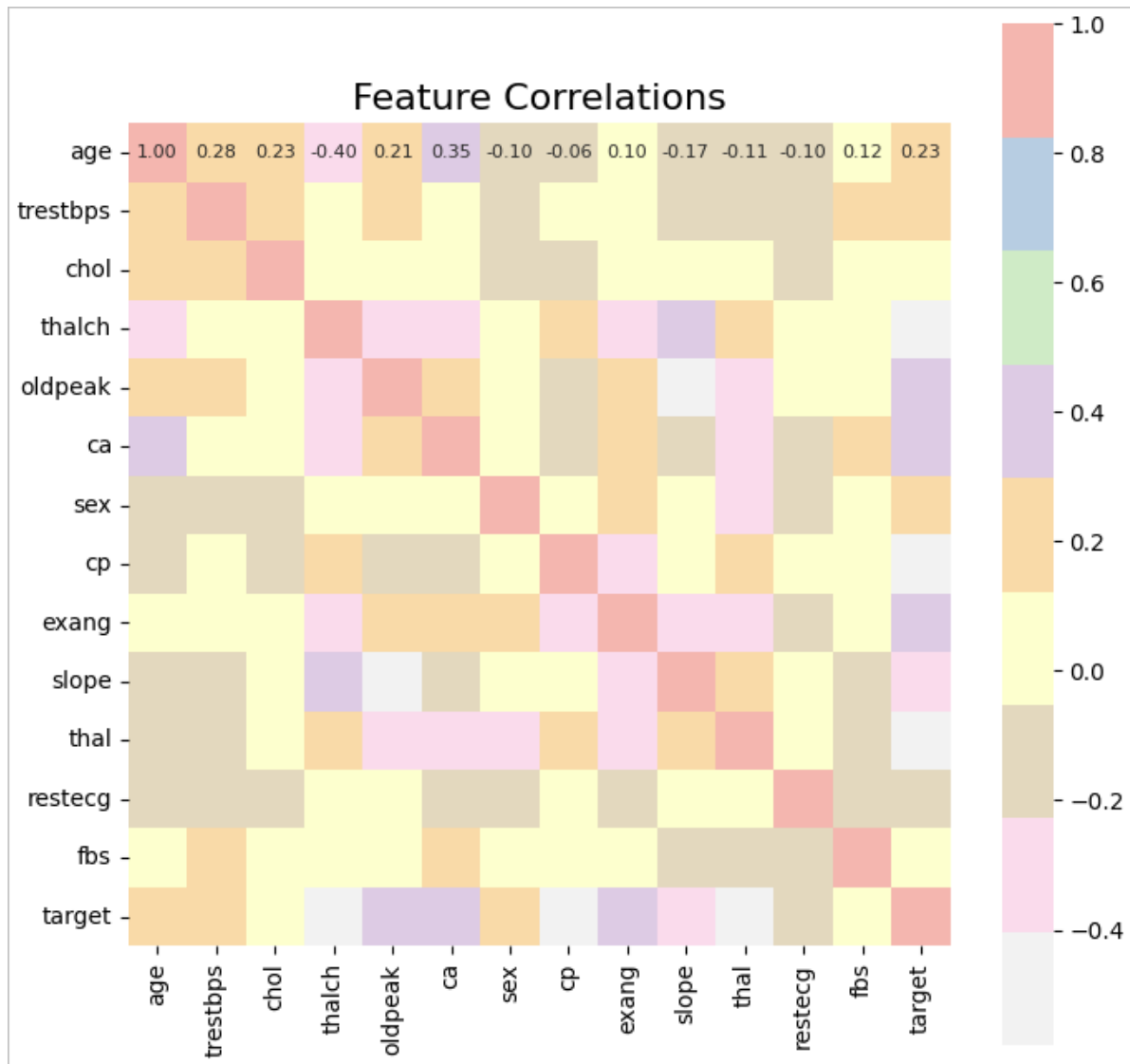


```

corr = heart_df.corr()
f,ax = plt.subplots(figsize=(8, 8))
hm = sn.heatmap(corr, square=True, ax=ax, annot=True,
cmap='Pastell_r', fmt='.2f', annot_kws={'size':8})
plt.title('Feature Correlations', size=16)
SAS.pyplot(plt)

```

```
Text(0.5, 1.0, 'Feature Correlations')
```



Creating Training and Test Data

The original data is split by putting 80% into the training set and 20% into the test set. The shape of the full data and the two new subsets will also be examined.

```

X = heart_df.drop('target', axis=1)
y = heart_df['target']
print('Full data')

```

```
print('Shape of X and y respectively :', X.shape, y.shape)
```

```
Full data
Shape of X and y respectively : (304, 13) (304,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
print('After split')
print('Shape of X and y respectively (train) :', X_train.shape,
y_train.shape)
print('Shape of X and y respectively (test) :', X_test.shape,
y_test.shape)
```

```
After split
Shape of X and y respectively (train) : (243, 13) (243,)
Shape of X and y respectively (test) : (61, 13) (61,)
```

```
print(X_train.head())
```

```
   age  trestbps  chol  thalch  oldpeak  ...  exang  slope  thal  restecg  fbs
176   52      108   233    147     0.1  ...    0       2     0         1     1
26    58      120   340    172     0.0  ...    0       2     2         1     0
74    44      110   197    177     0.0  ...    0       2     2         0     0
101   34      118   182    174     0.0  ...    0       2     2         0     0
154   64      120   246     96     2.2  ...    1       0     2         0     0
[5 rows x 13 columns]
```

```
print(X_test.head())
```

```
   age  trestbps  chol  thalch  oldpeak  ...  exang  slope  thal  restecg  fbs
66    60      140   185    155     3.0  ...    0       1     2         0     0
156   51      140   299    173     1.6  ...    1       2     0         1     0
247   47      110   275    118     1.0  ...    1       1     2         0     0
267   59      126   218    134     2.2  ...    0       1     1         1     1
145   47      108   243    152     0.0  ...    0       2     2         1     0
[5 rows x 13 columns]
```

Building and Training Models with Four Different Algorithms

Overview

Four different models are trained against the training set: LogisticRegression, DecisionTreeClassifier, ForestClassifier, and GradientBoostingClassifier. In each case, the following information is examined for each run:

- Model accuracy
- Confusion matrix
- ROC plot

```

# Common model fitting and assessment code

def fit_model(X_train, y_train, X_test, y_test, model):
    print(model.name.split('_')[0], 'Accuracy\n')
    model.fit(X_train, y_train)
    y_pred_test = model.predict(X_test)

    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)

    print(f'Training accuracy: {train_score:.2f}')
    print(f'Test accuracy: {test_score:.2f}')

    return y_pred_test, test_score

def accuracy_report(y_test, y_pred_test, model):
    output = pd.DataFrame({'Predicted':y_pred_test}) # Heart-Disease
    yes or no? 1/0
    people = output.loc[output.Predicted == 1]["Predicted"]
    rate_people = 0
    if len(people) > 0 :
        rate_people = len(people)/len(output)
    print(f"% of people predicted with heart-disease:
{rate_people*100:.2f}\n")
    print('Classification Report for', model.name.split('_')[0], '\n',
classification_report(y_test,y_pred_test))

def confusion_plot(y_test, y_pred_test, model):
    confusion_results = confusion_matrix(y_test,y_pred_test)
    class_names = [0,1]
    fig,ax = plt.subplots() #
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks,class_names)
    plt.yticks(tick_marks,class_names)
    sn.heatmap(pd.DataFrame(confusion_results), annot = True, cmap =
'Pastell_r', fmt = 'g')
    ax.xaxis.set_label_position('top')
    plt.tight_layout()
    plt.title('Confusion Matrix for ' + model.name.split('_')[0])
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')

def roc_plot(X_test, y_test, model):
    probs = model.predict_proba(X_test).to_numpy()
    preds = probs[:,1]
    fpr, tpr, threshold = roc_curve(y_test, preds)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, label='Model (area = %0.3f)' % roc_auc)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')

```



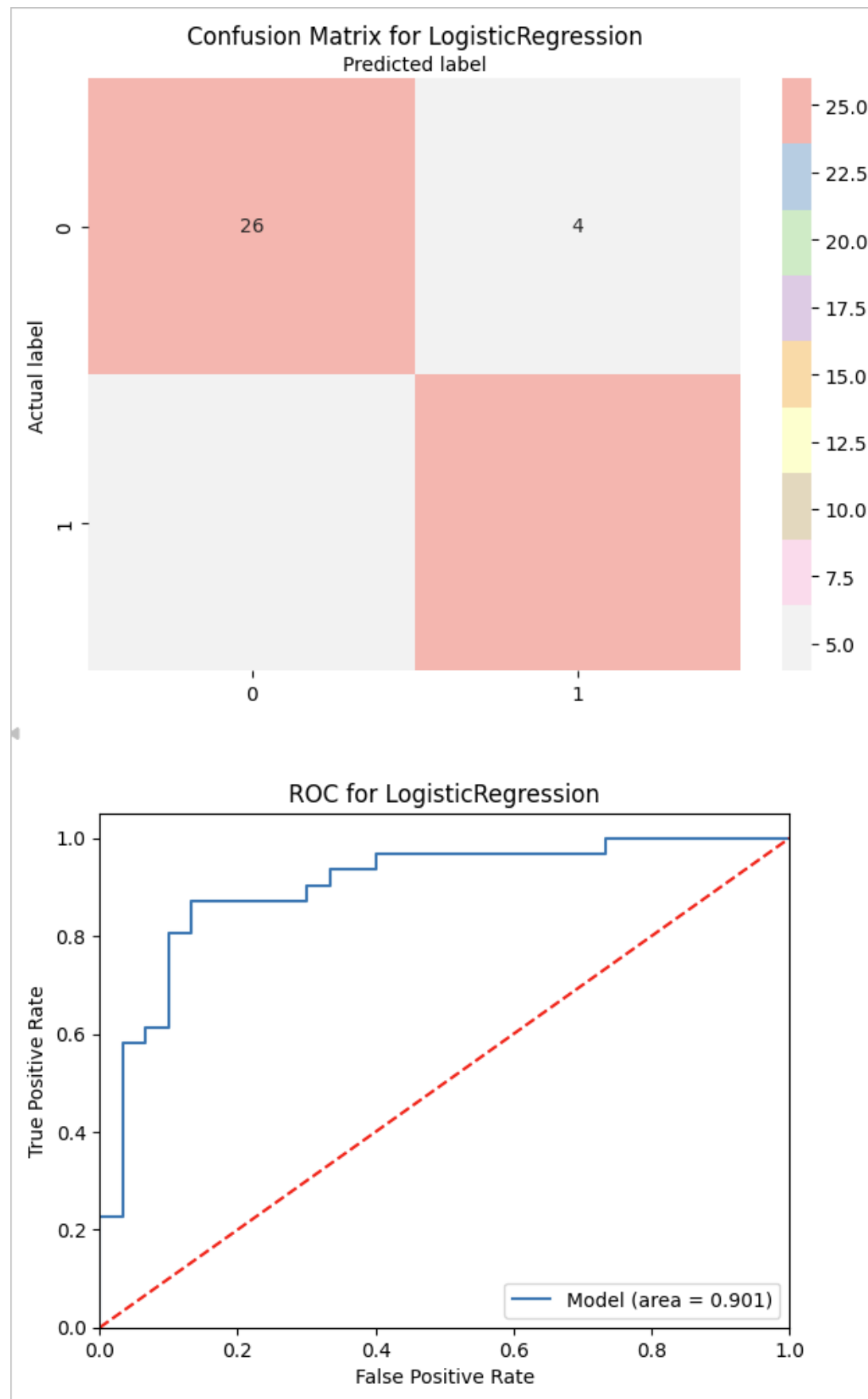
```
plt.ylabel('True Positive Rate')
plt.title('ROC for ' + model.name.split('_')[0])
plt.legend(loc="lower right")
```

LogisticRegression

For details about using the LogisticRegression class in the sasviya package, see the [LogisticRegression documentation](#).

```
logreg = LogisticRegression(
    solver='lbfgs',
    tol=1e-4,
    max_iter=1000)

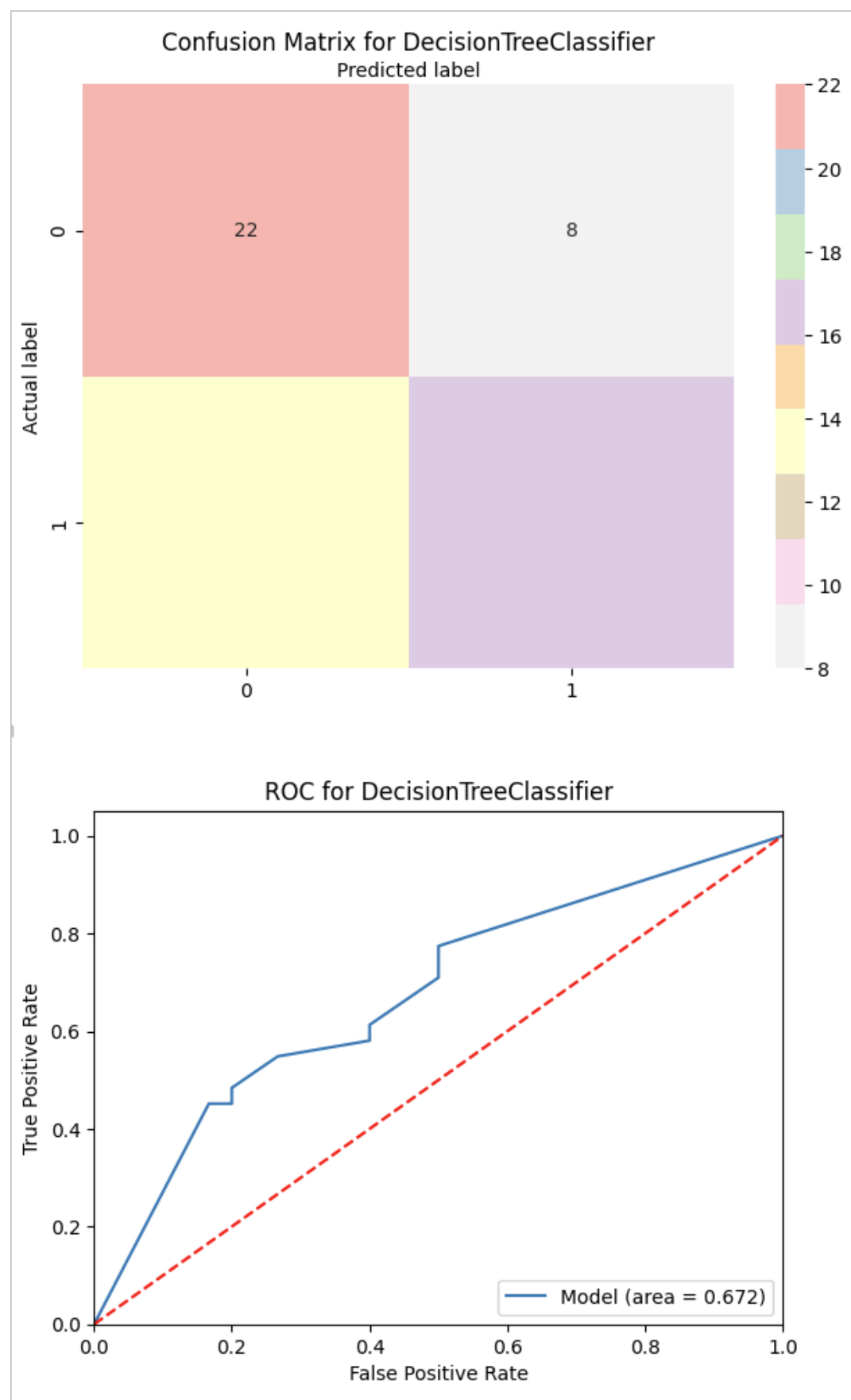
y_pred_test, score_logreg = fit_model(X_train, y_train, X_test,
y_test, model=logreg)
accuracy_report(y_test, y_pred_test, model=logreg)
confusion_plot(y_test, y_pred_test, model=logreg)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)
roc_plot(X_test, y_test, model=logreg)
SAS.pyplot(plt)
```



DecisionTreeClassifier

For details about using the `DecisionTreeClassifier` class in the `sasviya` package, see the [DecisionTreeClassifier documentation](#).

```
dt = DecisionTreeClassifier(max_depth=5)
y_pred_test, score_dtc = fit_model(X_train, y_train, X_test, y_test,
model=dt)
accuracy_report(y_test, y_pred_test, model=dt)
confusion_plot(y_test, y_pred_test, model=dt)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)
roc_plot(X_test, y_test, model=dt)
SAS.pyplot(plt)
```

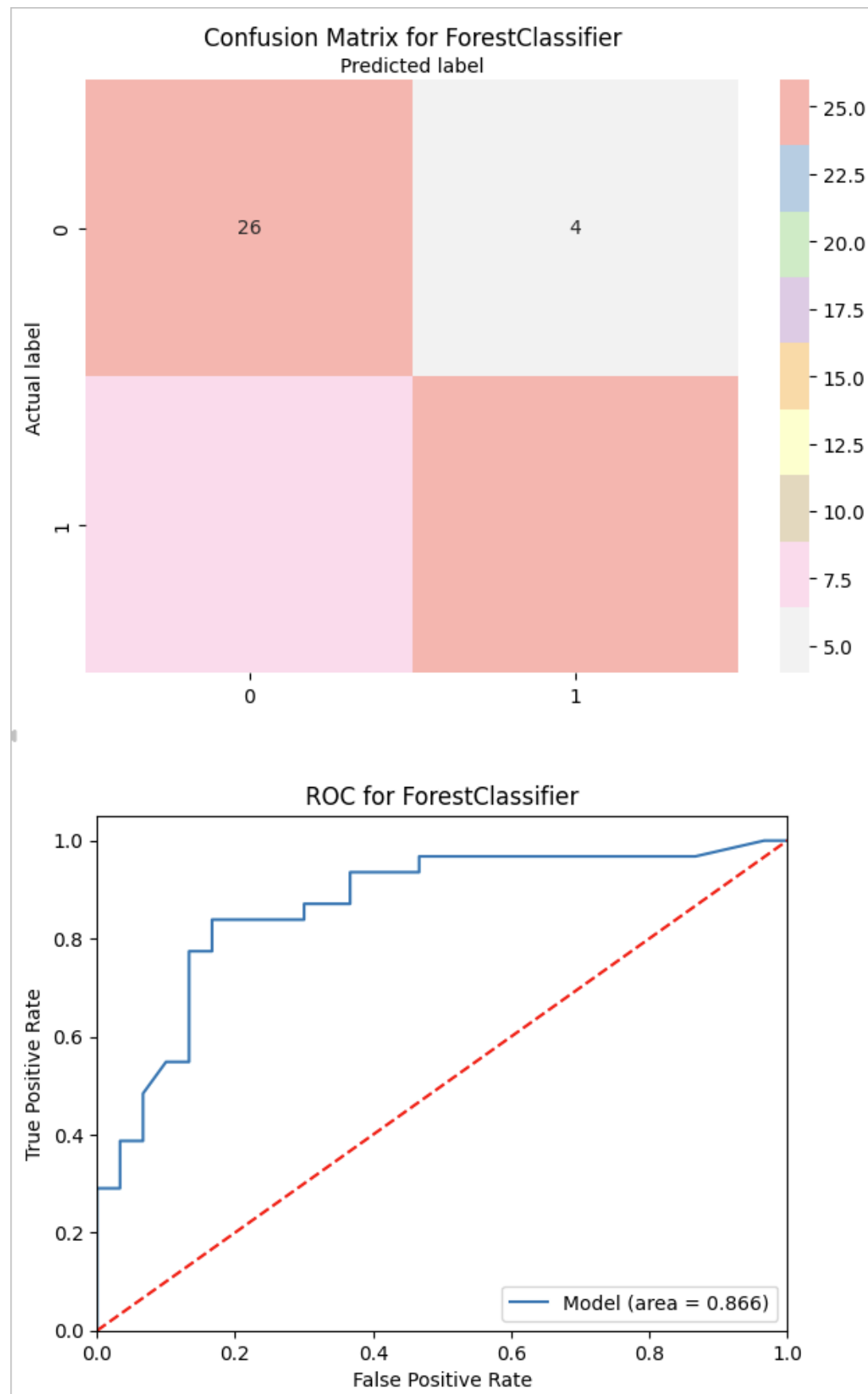


ForestClassifier

For details about using the ForestClassifier class in the sasviya package, see the [ForestClassifier documentation](#).

```
fc = ForestClassifier(
    n_estimators=100,
    max_depth=5,
    min_samples_leaf=1,
    max_features=None,
    criterion='gini',
    random_state=1
)

y_pred_test, score_fc = fit_model(X_train, y_train, X_test, y_test,
model=fc)
accuracy_report(y_test, y_pred_test, model=fc)
confusion_plot(y_test, y_pred_test, model=fc)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)
roc_plot(X_test, y_test, model=fc)
SAS.pyplot(plt)
```

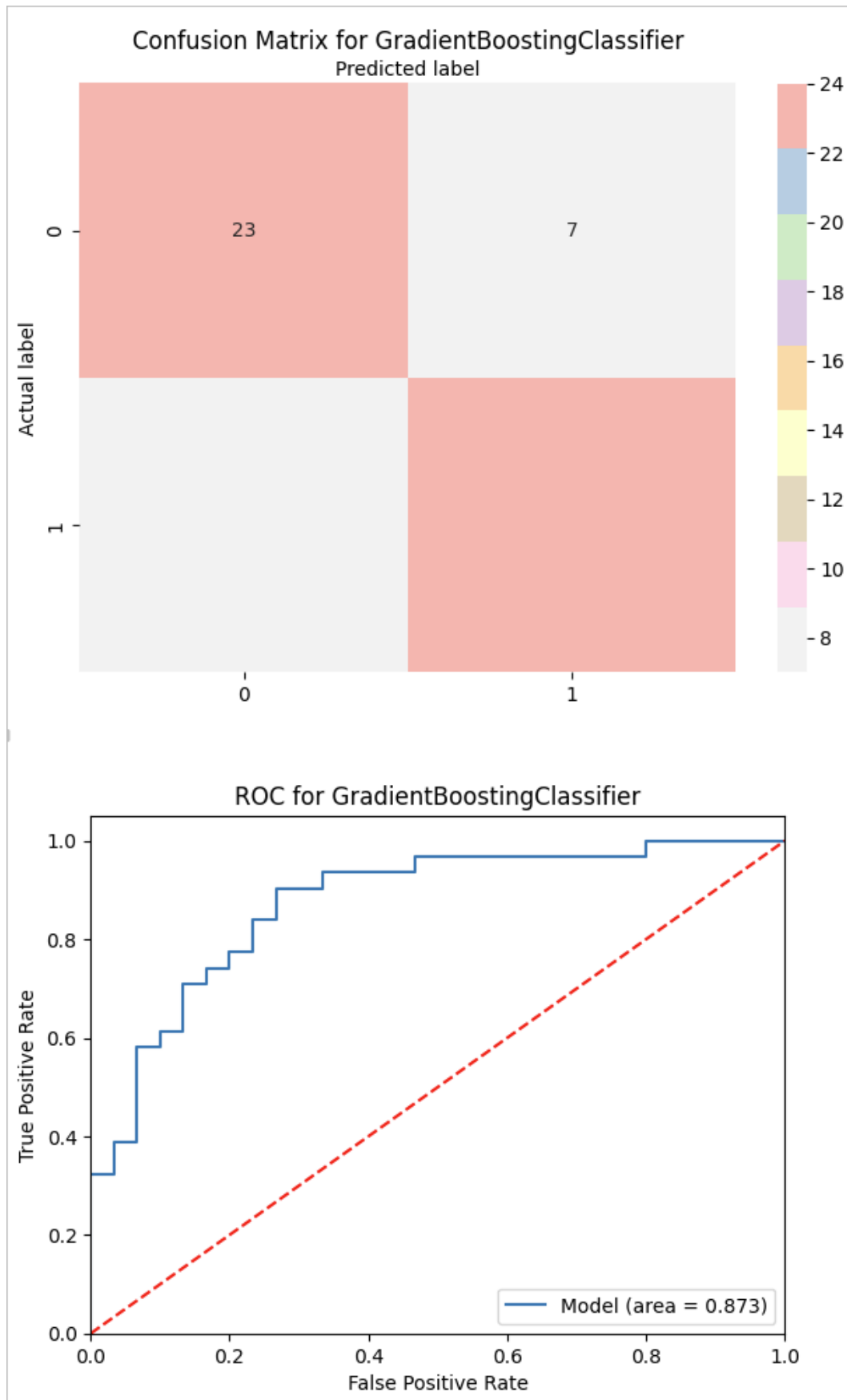


GradientBoostingClassifier

For details about using the GradientBoostingClassifier class in the sasviya package, see the [GradientBoostingClassifier documentation](#).

```
gb = GradientBoostingClassifier(n_estimators=100,
                                max_depth=5,
                                min_samples_leaf=1,
                                max_features=None,
                                learning_rate = 0.1,
                                subsample = 1.0,
                                random_state=1)

y_pred_test, score_gbc = fit_model(X_train, y_train, X_test, y_test,
model=gb)
accuracy_report(y_test, y_pred_test, model=gb)
confusion_plot(y_test, y_pred_test, model=gb)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)
roc_plot(X_test, y_test, model=gb)
SAS.pyplot(plt)
```



Summarizing the Results

A histogram is used to examine the accuracy scores of each model and identify which model performed best on this training and test set.

```

algorithms = ['LogisticRegression', 'GradientBoosting', 'Forest',
              'DecisionTree']
scores = [round(s,2) for s in [score_logreg, score_gbc, score_fc,
                              score_dtc]]

scores_df = pd.DataFrame(zip(algorithms, scores))
scores_df.columns = ['Name', 'Test Accuracy']

scores_df.sort_values(by='Test Accuracy', inplace=True, ascending =
False)

plt.rcdefaults()
fig, ax = plt.subplots()

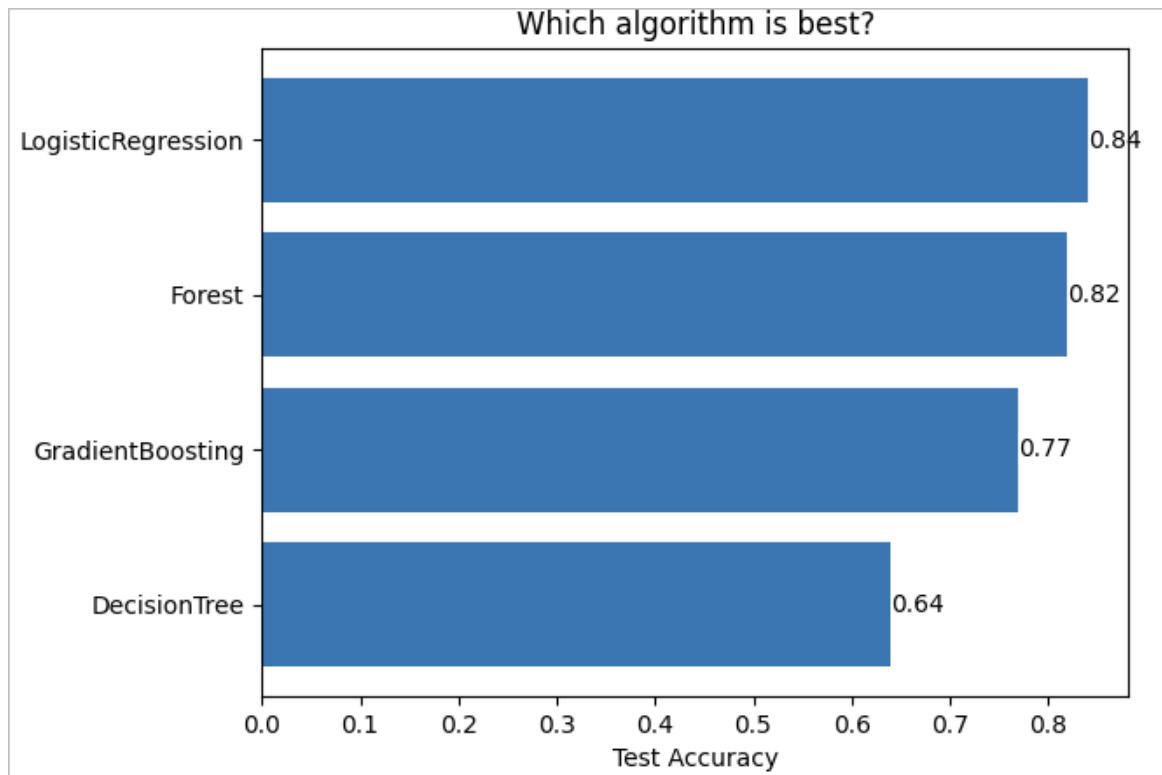
y_pos = np.arange(len(scores_df)) # scores
bars = ax.barh(y_pos, scores_df['Test Accuracy'], align='center')
ax.set_yticks(y_pos)
ax.set_yticklabels(scores_df['Name'])
ax.invert_yaxis() # labels read top-to-bottom
ax.set_xlabel('Test Accuracy')
ax.set_title('Which algorithm is best?')
ax.bar_label(bars)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)

```

```

[Text(0, 0, '0.84'),
Text(0, 0, '0.82'),
Text(0, 0, '0.77'),
Text(0, 0, '0.64')]

```



DecisionTreeRegressor

Enables you to fit a decision tree to your data and predict new values.

DecisionTreeRegressor from SAS[®] Viya[®] on Bike Sharing Demand

Source

This example has been adapted from [EDA & Ensemble Model \(Top 10 Percentile\)](#) by Vivek Srinivasan.

Data Preparation

About the data set

A bicycle sharing system is a system of linked rental locations throughout a metropolitan area that allows individuals to share a pool of bicycles for their use. Bicycles rented from one location can be returned to any other location in the network. Joining the network and renting or returning bicycles are automated. As of 2022, there are around 3,000 cities offering such systems.

The data set contains information about the number of bikes rented by different types of users over several years tracked by hour. Additional information about each rental period includes weather information and categorization of the type of day.

The import statement loads the os, pandas, and numpy Python libraries.

```
import os
import pandas as pd
import numpy as np
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

Importing the data

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
print(bikeData.shape)
```

```
(17379, 16)
```

Sample of the first few rows

The `head()` method prints the first five rows.

```
print(bikeData.head(5))
```

	date	season	year	month	...	windspeed	casual	registered	count
0	1/1/2011	1	0	1	...	0.0	3	13	16
1	1/1/2011	1	0	1	...	0.0	8	32	40
2	1/1/2011	1	0	1	...	0.0	5	27	32
3	1/1/2011	1	0	1	...	0.0	3	10	13
4	1/1/2011	1	0	1	...	0.0	0	1	1

[5 rows x 16 columns]

Variable data types

The `dtypes` method prints the data type of each column.

```
print(bikeData.dtypes)
```

```
date           object
season         int64
year           int64
month          int64
hour           int64
holiday        int64
weekday        int64
workingday     int64
weather        int64
temp           float64
atemp          float64
humidity       float64
windspeed     float64
casual         int64
registered     int64
count          int64
dtype: object
```

Data Analysis

From the above results, the columns `hour`, `weekday`, `month`, `season`, `holiday`, `workingday`, and `weather` should be of categorical data type. The columns will be coerced from integers into appropriate categorical types, and the date column will be dropped.

Coercing to categorical types and dropping unnecessary columns

```
bikeData["month"] =
bikeData.month.map({1:"January",2:"February",3:"March",4:"April",5:"May",
6:"June",7:"July",

8:"August",9:"September",10:"October",11:"November",12:"December"})
bikeData["weekday"] =
bikeData.weekday.map({0:"Sunday",1:"Monday",2:"Tuesday",3:"Wednesday",4
:"Thursday",5:"Friday",6:"Saturday"})
bikeData["season"] = bikeData.season.map({1: "Spring", 2 : "Summer",
3 : "Fall", 4 : "Winter" })
bikeData["weather"] = bikeData.weather.map({1: " Clear + Few clouds +
Partly cloudy + Partly cloudy",\
2 : " Mist + Cloudy, Mist +
Broken clouds, Mist + Few clouds, Mist ", \
3 : " Light Snow, Light Rain +
Thunderstorm + Scattered clouds, Light Rain + Scattered clouds", \
4 : " Heavy Rain + Ice Pallets
+ Thunderstorm + Mist, Snow + Fog " })
```

```
bikeData["workingday"] = bikeData.workingday.map({0: "Non-Working
Day", 1: "Working Day"})
```

The for loop coerces the variables in the `bikeData` `DataFrame` that are in the `categoryVariableList` list to have a category data type. The `drop()` method drops the unnecessary date column.

```
categoryVariableList =
["hour", "weekday", "month", "season", "holiday", "workingday", "weather"]
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("category")
bikeData = bikeData.drop(["date"], axis=1)
```

Outlier Analysis

Visualizing the outliers

To get a sense of how outliers might be distributed in the data, you can run a series of box plots, first for `count` by itself and then for `count` versus `season`, `hour`, and `workingday`.

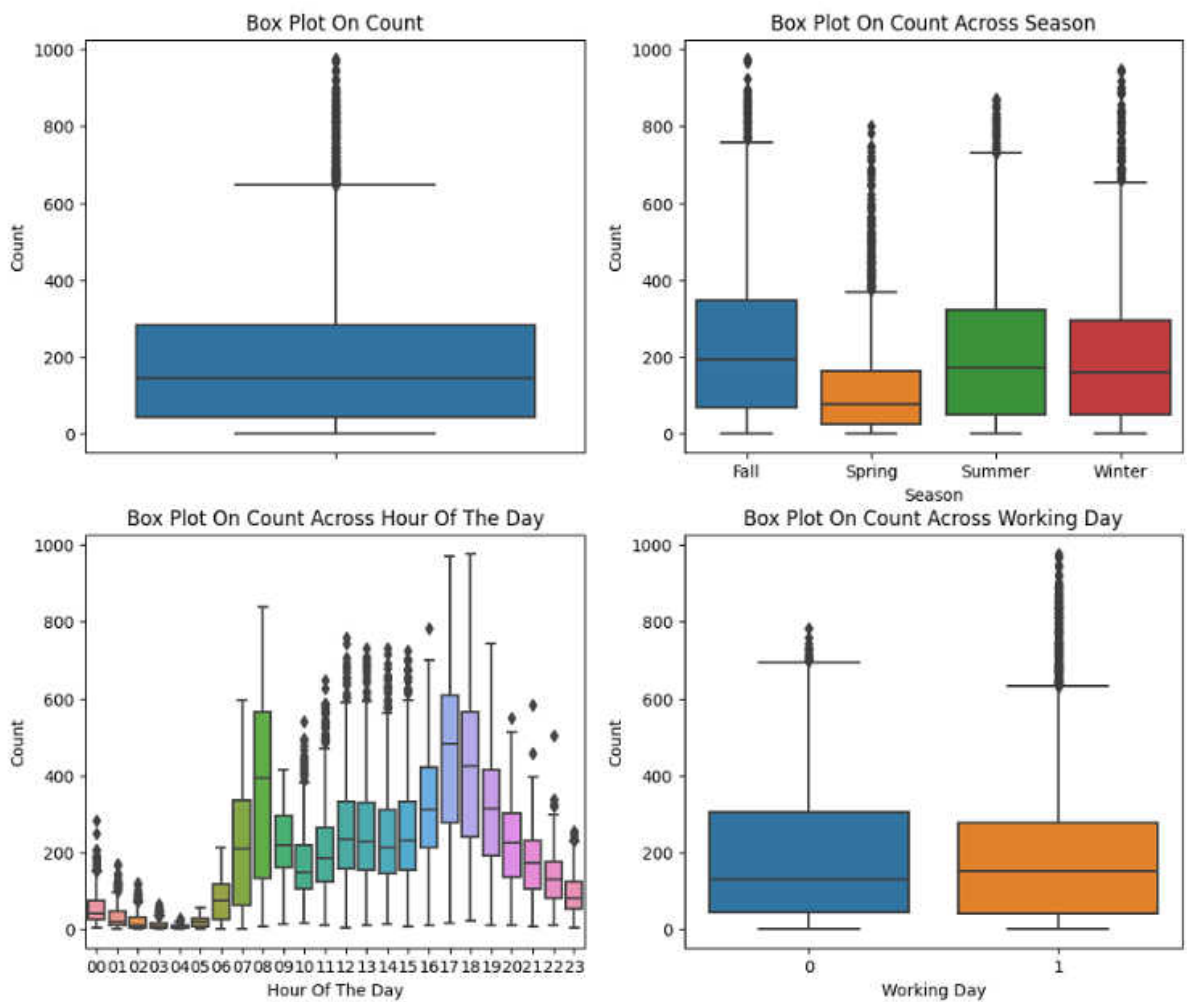
The plots enable you to draw the following inferences about the data:

- `count` contains many outliers beyond the upper quartile limit, which skews the distribution toward higher values.
For `season`, *Spring* has a relatively lower count, demonstrated by the lower median value in its box plot.
- The box plot for *Hour Of The Day* has relatively higher median values for 0700, 0800, 1700, and 1800 hours. This likely matches up with commuter traffic for the start and end of school and office workdays.
- *Working Day* observations contribute a majority of the outliers, as opposed to *Non-Working Day* ones.

```
import matplotlib.pyplot as plt
import seaborn as sn
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_size_inches(12, 10)
sn.boxplot(data=bikeData, y="count", orient="v", ax=axes[0][0])
sn.boxplot(data=bikeData, y="count", x="season", orient="v", hue="season", ax=axes[0][1])
sn.boxplot(data=bikeData, y="count", x="hour", orient="v", hue="hour", ax=axes[1][0])
sn.boxplot(data=bikeData, y="count", x="workingday", hue="workingday", orient="v", ax=axes[1][1])

axes[0][0].set(ylabel='Count', title="1. Box Plot On Count")
axes[0][1].set(xlabel='Season', ylabel='Count', title="2. Box Plot On Count Across Season")
axes[1][0].set(xlabel='Hour Of The Day', ylabel='Count', title="3. Box Plot On Count Across Hour Of The Day")
axes[1][1].set(xlabel='Working Day', ylabel='Count', title="4. Box Plot On Count Across Working Day")
```

SAS.pyplot (plt)



Removing outliers in the count variable

This code removes outliers in the count column.

```
bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape of the data with outliers: ",bikeData.shape)
print ("Shape of the data without outliers:
",bikeDataWithoutOutliers.shape)
```

```
Shape Of The Before Ouliers: (17379, 15)
Shape Of The After Ouliers: (17135, 15)
```

Correlation Analysis

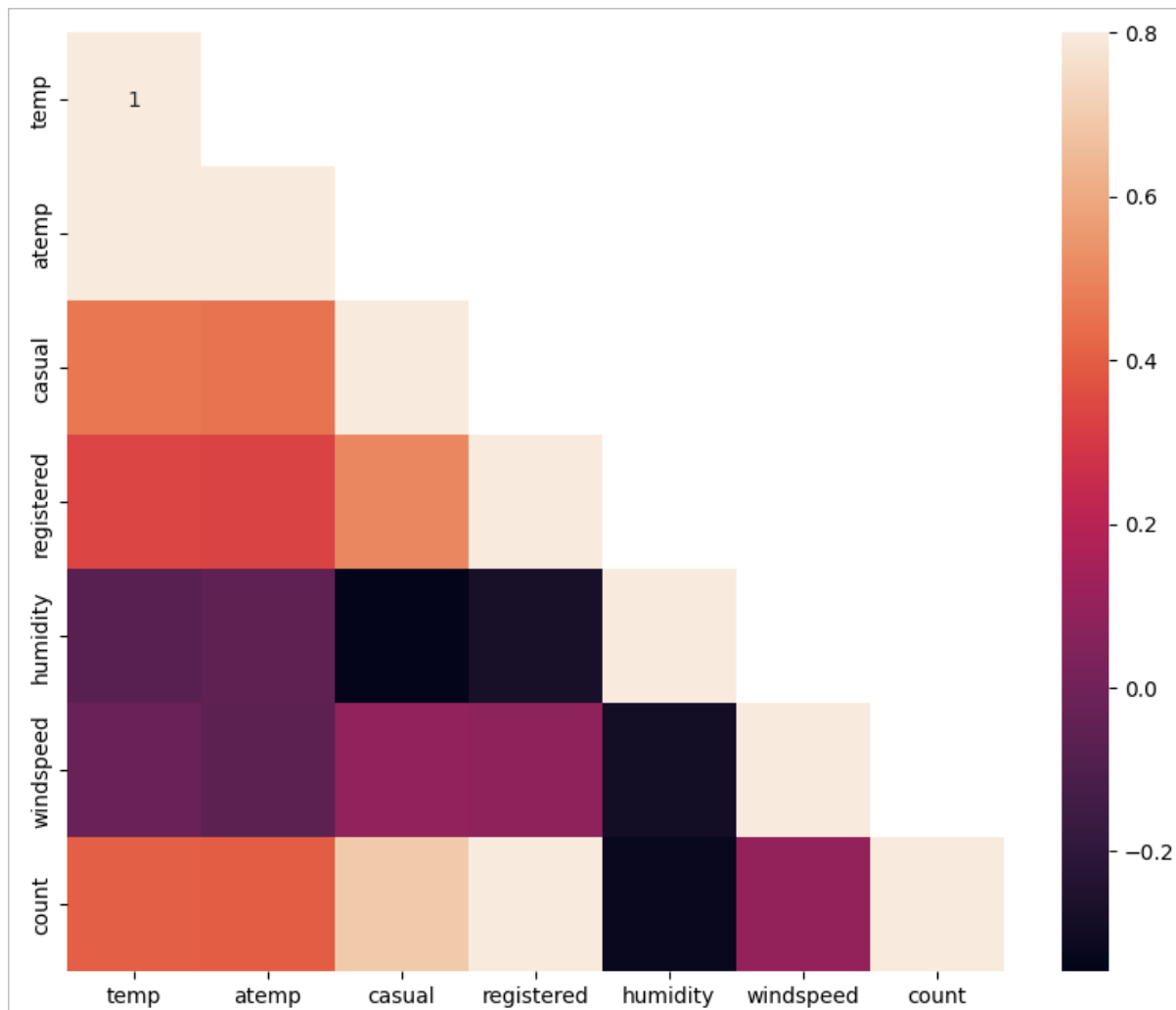
A common method for understanding numeric variables influence dependent variables is through constructing a correlation matrix. The correlation of `count` versus `temp`, `atemp`, `humidity`, and `windspeed` will be examined.

Viewing a heat map

By examining the heat map below, you can make the following observations:

- `temp` and `humidity` have a positive and negative correlation with `count`, respectively. Since the values are noticeably large, `count` does not depend much on either.
- As `windspeed` has a very low correlation with `count`, it adds little to the analysis.
- Since `atemp` (measurement of what the temperature "feels like") and `temp` (thermometer reading) have a strong correlation with each other, including them both would introduce multicollinearity into the analysis. With those definitions, one would expect this relationship `atemp` to be dropped.
- As `casual` and `registered` are leakage variables in nature, they should also be dropped.

```
corrMatt =
bikeData[["temp", "atemp", "casual", "registered", "humidity", "windspeed", "
count"]].corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig, ax = plt.subplots()
fig.set_size_inches(10, 8)
sns.heatmap(corrMatt, mask=mask, vmax=.8, square=True, annot=True)
SAS.pyplot(plt)
```



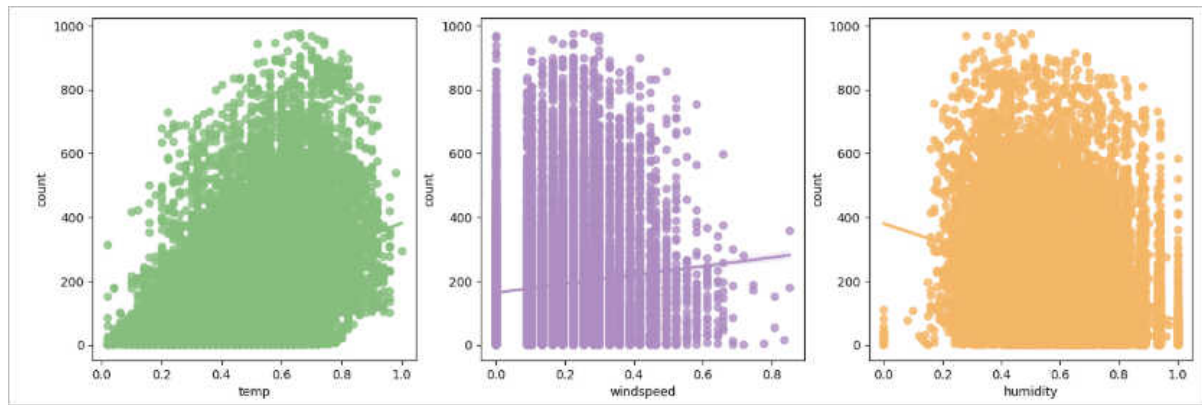
Regression plots

Regression plots from `seaborn` are a useful way to depict the relationship between two features. These will also be viewed for `count` versus `temp`, `atemp`, `humidity`, and `windspeed`.

```
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3)
fig.set_size_inches(16, 5)

# Assigning softer colors to each plot
color1 = "#7fbf7b" # Light green
color2 = "#af8dc3" # Light purple
color3 = "#fdb462" # Light orange

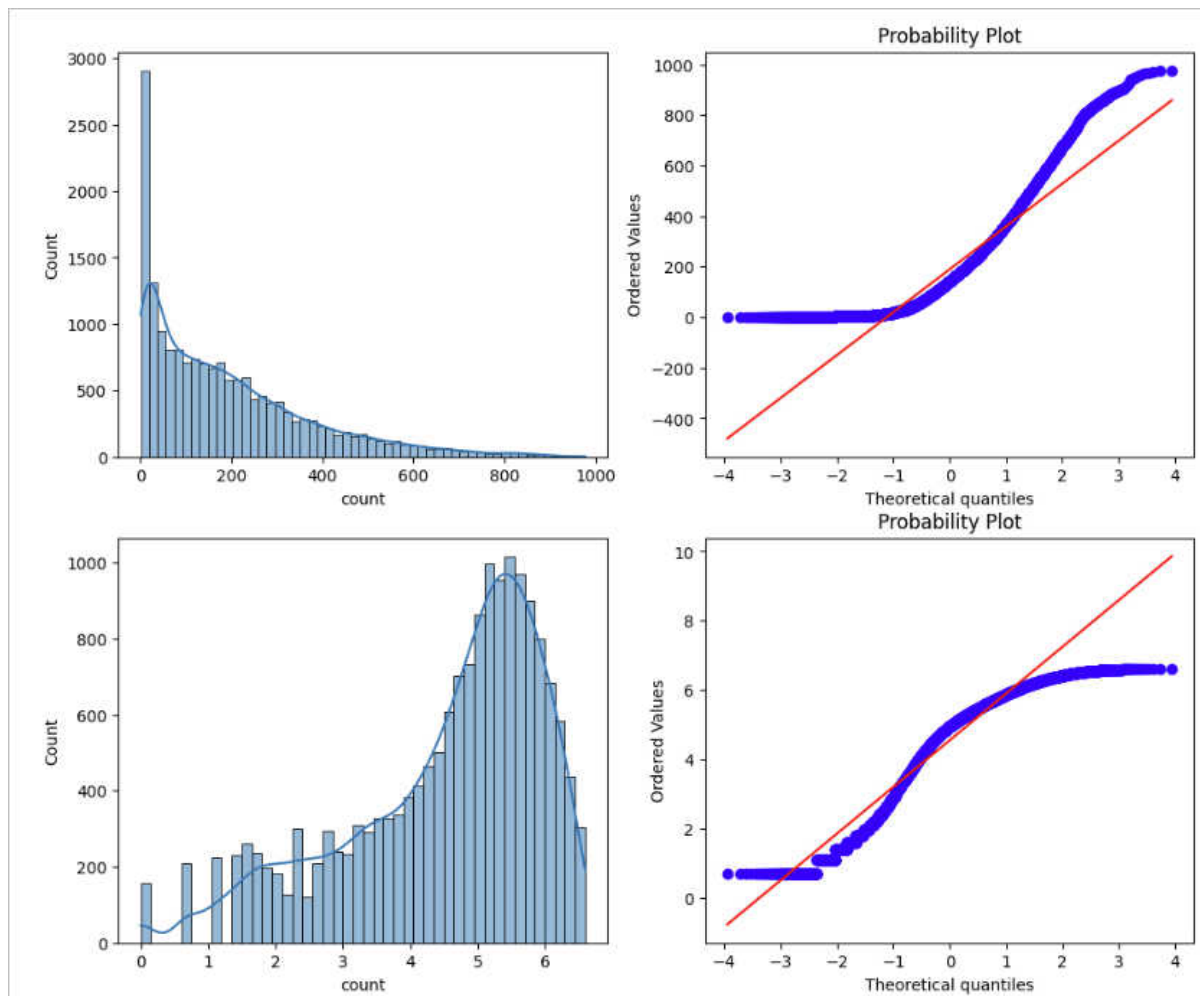
sn.regplot(x="temp", y="count", data=bikeData, ax=ax1, color=color1)
sn.regplot(x="windspeed", y="count", data=bikeData, ax=ax2,
color=color2)
sn.regplot(x="humidity", y="count", data=bikeData, ax=ax3, color=color3)
SAS.pyplot(plt)
```

Visualizing the Distribution of Data

As is visible from the figures below, "count" is skewed toward the right and not a normal distribution. Since many machine learning algorithms require a normal distribution, you will need to pursue options to address this. A common technique is to take a log transformation of "count" after removing the outliers. Afterward, the data distribution looks better but is still not an ideal normal.

```
from scipy import stats
fig, axes = plt.subplots(ncols=2, nrows=2)
fig.set_size_inches(12, 10)
sn.histplot(x = bikeData["count"], kde = True, ax = axes[0,0])
stats.probplot(bikeData["count"], dist='norm', fit=True, plot=axes[0]
[1] )
sn.histplot(x = np.log(bikeDataWithoutOutliers["count"]), kde = True,
ax = axes[1,0])
stats.probplot(np.log1p(bikeDataWithoutOutliers["count"]),
dist='norm', fit=True, plot=axes[1][1])
fig.show()
SAS.pyplot(plt)
```



Visualizing Count versus Month, Season, Hour, and Weekday

The plots help show the following information:

- People tend to rent bikes during summer leading June, July, and August to have relatively higher count (demand).
- On weekdays, more people rent bicycles for the 0700, 0800, 1700, and 1800 hours. This overlaps with normal school and office schedules.
- The renters during this time are registered users.
- On weekend days, more bicycles are rented between 1000 and 1600.

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=4)
fig.set_size_inches(12, 20)
sortOrder =
["January", "February", "March", "April", "May", "June", "July", "August", "Sep
tember", "October", "November", "December"]
hueOrder =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
]
```

```

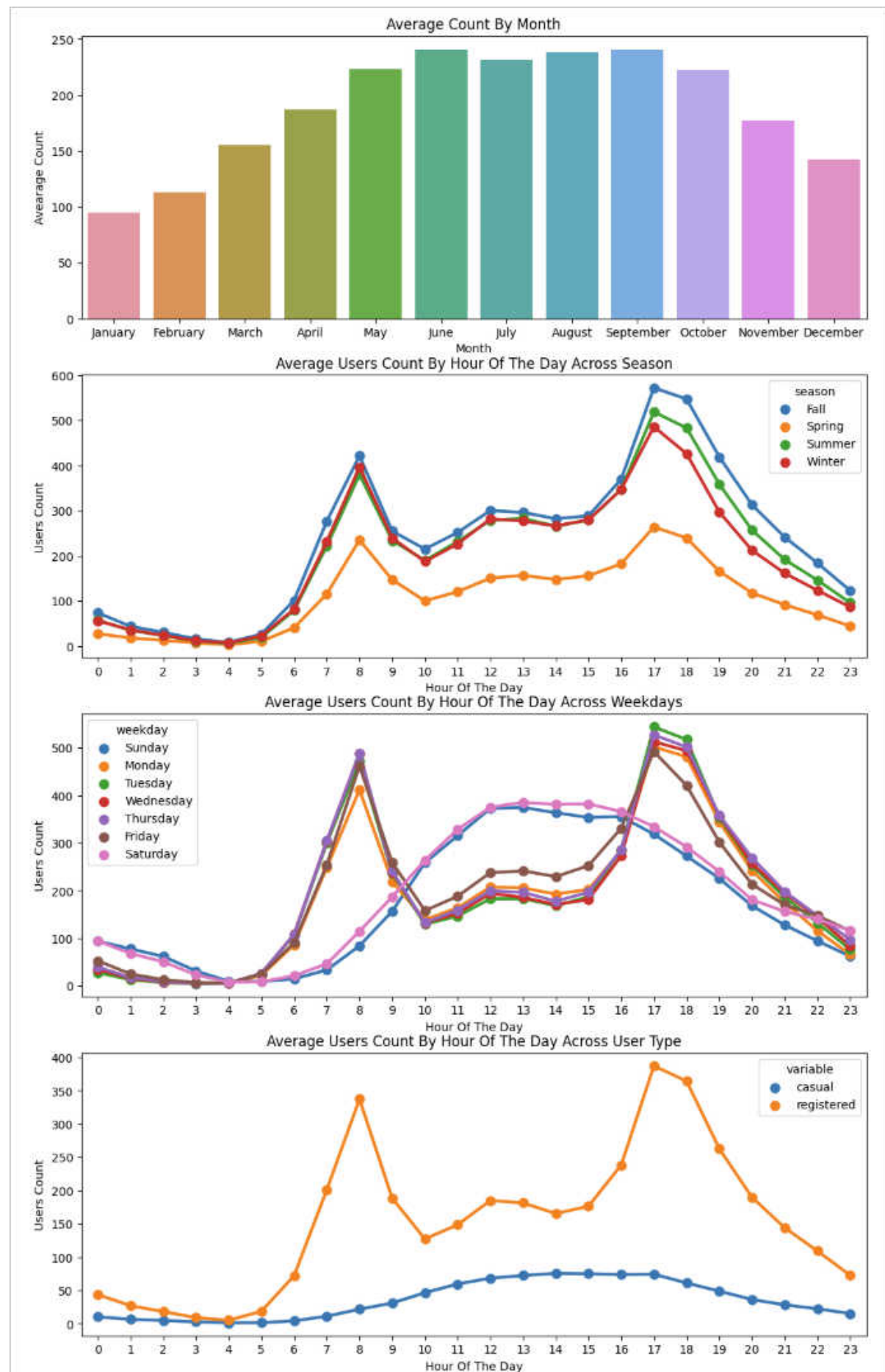
monthAggregated = pd.DataFrame(bikeData.groupby("month")
["count"].mean()).reset_index()
monthSorted = monthAggregated.sort_values(by="count",ascending=False)
sn.barplot(data=monthSorted,x="month",y="count",hue="month",ax=ax1,orde
r=sortOrder)
ax1.set(xlabel='Month', ylabel='Average Count',title="Average Count By
Month")

hourAggregated =
pd.DataFrame(bikeData.groupby(["hour","season"],sort=True)
["count"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["count"],hue=hourAggregated["season"],
data=hourAggregated, join=True,ax=ax2)
ax2.set(xlabel='Hour Of The Day', ylabel='Users Count',title="Average
Users Count By Hour Of The Day Across Season",label='big')

hourAggregated =
pd.DataFrame(bikeData.groupby(["hour","weekday"],sort=True)
["count"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["count"],hue=hourAggregated["weekday"],hue_order=hueOr
der, data=hourAggregated, join=True,ax=ax3)
ax3.set(xlabel='Hour Of The Day', ylabel='Users Count',title="Average
Users Count By Hour Of The Day Across Weekdays",label='big')

hourTransformed = pd.melt(bikeData[["hour","casual","registered"]],
id_vars=['hour'], value_vars=['casual', 'registered'])
hourAggregated =
pd.DataFrame(hourTransformed.groupby(["hour","variable"],sort=True)
["value"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["value"],hue=hourAggregated["variable"],hue_order=["ca
sual","registered"], data=hourAggregated, join=True,ax=ax4)
ax4.set(xlabel='Hour Of The Day', ylabel='Users Count',title="Average
Users Count By Hour Of The Day Across User Type",label='big')
fig.show()
SAS.pyplot(plt)

```



Integrating Data Discoveries

Load the data

After this visualization, the discoveries about the data will be integrated in preparation for the analysis.

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Defining feature lists

This code defines three lists that are used to identify categorical features, numeric features, and features to be dropped. The for loop converts the variables in the data DataFrame that are in the `categoricalFeatureNames` list to variables with categorical data types.

```
categoricalFeatureNames =
["season", "holiday", "workingday", "weather", "weekday", "month", "year", "hour"]
numericalFeatureNames = ["temp", "humidity", "windspeed", "atemp"]
dropFeatures = ["casual", "date", "registered"]
```

Dropping unnecessary variables

The `drop()` method drops unnecessary features from the `bikeData` dataframe based on the variables in the `dropFeatures` list.

```
bikeData = bikeData.drop(dropFeatures,axis=1)
```

Creating training and test data

The original data was split by putting 75% into the training set and 25% into the test set. The `bikeData` dataframe is split into two parts, `X` and `y`, representing the input features and the target variable (`count`) respectively.

The import command loads the `train_test_split` function from the sci-kit learn library. The `train_test_split()` function splits the data set into training and testing subsets for both features (`X`) and labels (`y`). The `test_size` parameter specifies that 25% of the total data set will be the testing set. The `random_state` parameter specifies that different runs of the code return the same data split.

```
X = bikeData.drop('count', axis=1)
Y = bikeData['count']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=3)
```

Decision Tree Model

The import command imports the `DecisionTreeRegressor` class from the `sasviya.ml.tree` module. The `DecisionTreeRegressor()` function creates an instance of the `DecisionTreeRegressor` model with the specified hyperparameters `max_depth`, `min_samples_leaf`, and `criterion`. The `fit()` method trains a decision tree regressor model on the training data. The `print()` function then prints the value of the test set score.

For details about using the `DecisionTreeRegressor` class, see the [DecisionTreeRegressor documentation](#).

Note that `DecisionTreeRegressor` (and most other SAS® Viya® modeling algorithms) provide native support for categorical features. So instead of coercing features to categorical, you simply specify categorical features using the `nominals` keyword.

```
from sasviya.ml.tree import DecisionTreeRegressor

dtModel = DecisionTreeRegressor(max_depth=5,
                                min_samples_leaf=5,
                                criterion='variance'
                                )

dtModel.fit(X_train, y_train, nominals = categoricalFeatureNames )
print("Decision Tree Regressor -Test Set Score:
{:.2f}".format(dtModel.score(X_test, y_test)))
```

```
Decision Tree Regressor -Test Set Score: 0.72
```

ForestClassifier

Enables you to fit random forest models to your data and then predict new class labels and probabilities.

ForestClassifier from SAS® Viya® on Handwritten Digits

Source

This example is adapted from [Example: Random Forest for Classifying Digits](#) and [Logistic Regression in Python: Handwriting Recognition](#).

Data Preparation

About the Data Set

This data is a set of 1,7971 images of digits that have been processed into 32 x 32 pixel bitmaps, divided into non-overlapping 4 x 4 pixel blocks. The number of pixels in each block is counted and each image is classified by the integer between 0 and 9 that it represents. This example will use this input format to classify additional images as the correct integer.

Importing the Data

scikit-learn includes a copy of this data and it can be loaded through `load_digits()`. It returns a tuple of the inputs and output.

```
from sklearn.datasets import load_digits
digits = load_digits()
print(digits.keys())
```

```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images',
'DESCR'])
```

Examining the Data

As these are images, viewing a representation of the images can help clear understand the data.

```
import matplotlib.pyplot as plt

# set up the figure
fig = plt.figure(figsize=(6, 6)) # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05,
wspace=0.05)
```

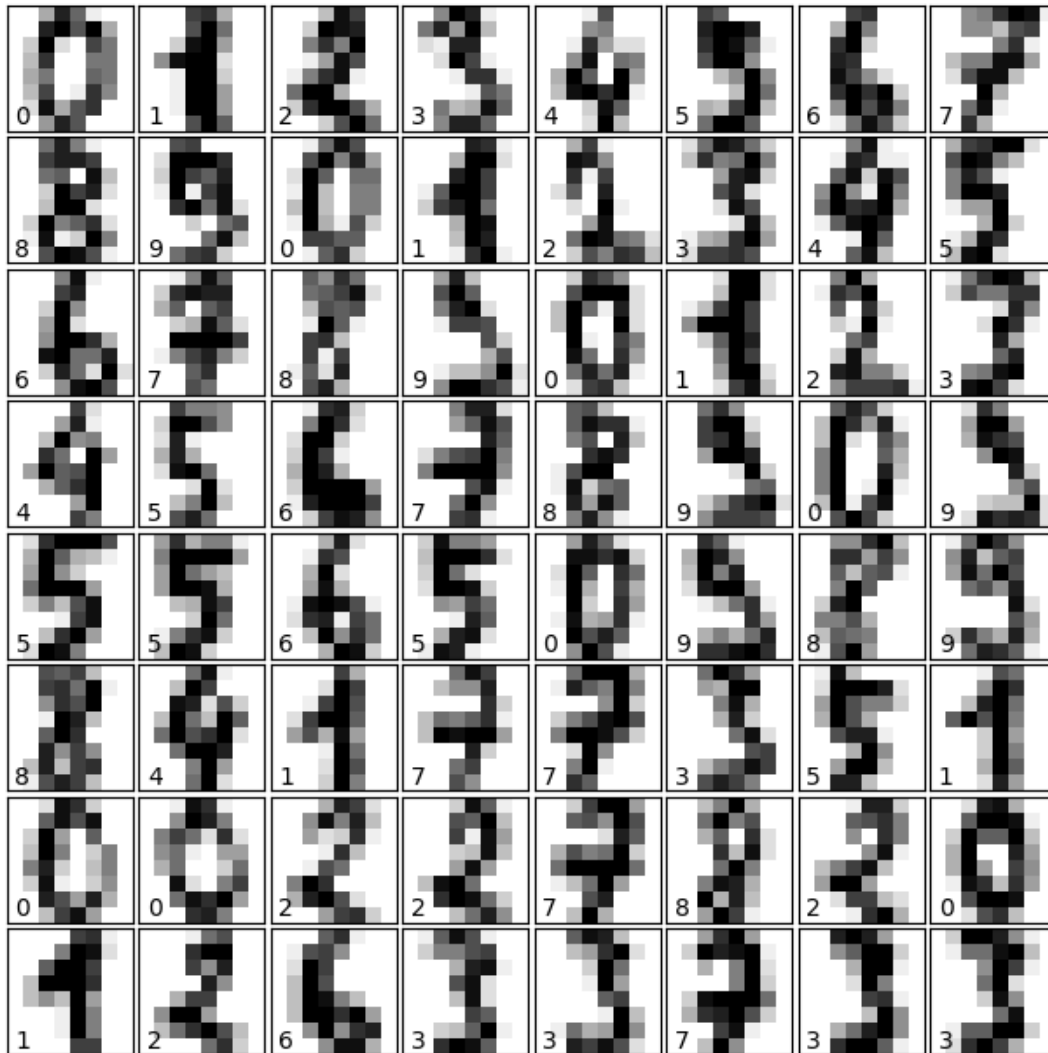
```

# plot the digits where each the color of each block is represented by
a pixel on the grayscale
for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(digits.images[i], cmap=plt.cm.binary,
interpolation='nearest')

    # label the image with the target value
    ax.text(0, 7, str(digits.target[i]))

SAS.pyplot(plt)

```



```

x, y = load_digits(return_X_y=True)
print(x)

```

```

[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]

```



```
print(y)
```

```
[0 1 2 ... 8 9 8]
```

Partitioning the Data

In order to train a model and test its accuracy, you will partition the data into two subsets randomly. Use a training set to create the model and the test set to evaluate how well the model performs.

scikit-learn provides `train_test_split()` to make this partitioning easy. In addition to the input and outcome data, two other important parameters are `test_size` for controlling the size of the test and `random_state` to define the state of the pseudo-random number generator used to split the data. The function returns four arrays--the input training and test data and the results for the training and test data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2, random_state=0)
```

Training a ForestClassifier Model with SAS® Viya®

In order to fit the model, create an instance of `ForestClassifier` and call `.fit()` with the `X_train` and `y_train` data.

For details about using the `ForestClassifier` class in the SAS Viya package, see the [ForestClassifier documentation](#).

```
from sasviya.ml.tree import ForestClassifier

rf = ForestClassifier(
    n_estimators=100,
    max_depth=5,
    min_samples_leaf=1,
    max_features=None,
    criterion='gini',
    random_state=0
)
rf.fit(X_train, y_train)
```

Examining the Results

Model parameters

Since `.fit()` returns the model, you can view the parameters used to train the model with `.get_params()`.

```
print(rf.get_params())
```

```
{'n_estimators': 100, 'bootstrap': 0.6, 'criterion': 'gini', 'max_depth': 5,
 'max_features': None, 'min_samples_leaf': 1, 'n_bins':
 50, 'oob_score': True, 'random_state': 0, 'verbose': 0}
```

Predicting results

Use the `predict()` method on the `X_test` data set to get the predicted class labels for each of the handwritten digits. Each input is assigned to a specific class (an integer from 0 to 9) based on the model's learned decision boundaries.

```
y_pred = rf.predict(X_test)
print(y_pred)
```

```
[2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
 1 0 1 8 6 7 7 9 0 7 6 2 1 9 6 7 9 0 0 9 1 6 3 0 2 3 4 1 9 7 6 9 1 8 3 5 1
 2 1 2 2 9 7 2 3 6 0 0 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 9 2 5 9 0 7 1 4 7 3
 4 8 9 7 7 8 0 6 5 2 5 3 4 1 7 0 6 1 5 5 3 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
 5 9 9 1 5 3 6 1 8 9 7 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 3 6 3 0
 3 3 3 0 7 7 9 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 3 3 1 4 2 1 6 8 9 9 4
 9 3 7 6 2 3 3 1 6 9 3 6 3 9 2 0 7 6 1 1 9 7 2 7 1 5 5 7 5 2 3 7 2 7 9 5 7
 0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 3 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
 4 6 2 9 0 1 7 2 0 3 6 0 4 2 0 7 5 8 5 7 8 2 8 4 3 7 2 6 5 1 5 1 0 8 2 1 9
 5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8]
```

Evaluate the accuracy of the model

You can obtain the accuracy of the model by specifying the `.score()` method on both the training set and the test set. It is helpful to compare the results for both the training and the test set because a higher accuracy score for the training data can indicate overfitting.

Use the `score()` method and pass in the training set as arguments:

```
print(f'Accuracy: {rf.score(X_train, y_train):.2f}')
```

```
Accuracy: 0.96
```

```
print(f'Accuracy: {rf.score(X_test, y_test):.2f}')
```

```
Accuracy: 0.93
```

Viewing the confusion matrix

Although the confusion matrix can be obtained with `confusion_matrix`, it is often more helpful to visualize the results through a heat map than a table of numbers. In the heat map, purple represents numbers two or less, while green and yellow represent numbers 10 or above.

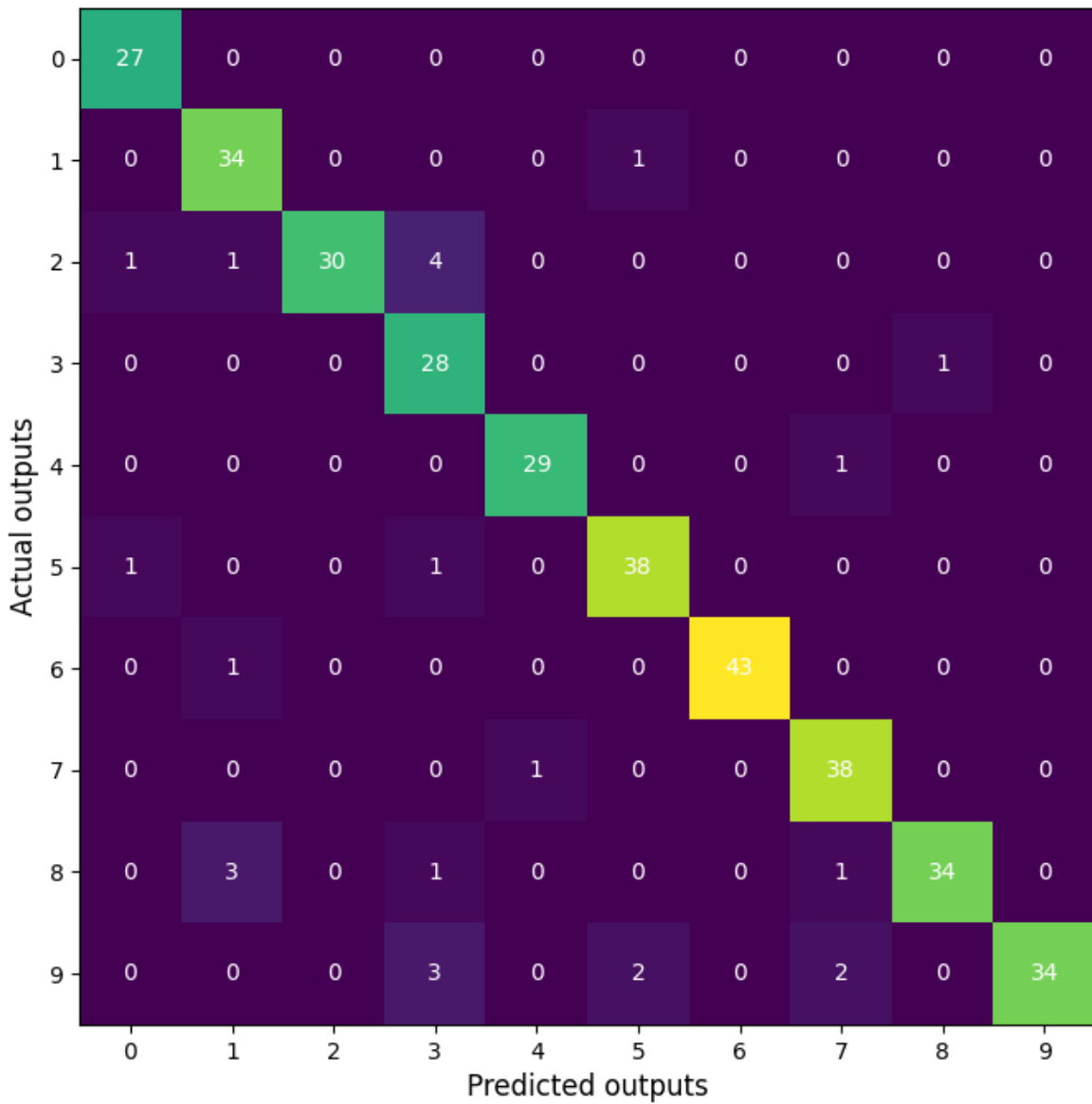
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(8, 8))
```

```

ax.imshow(cm)
ax.grid(False)
ax.set_xlabel('Predicted outputs', fontsize=12, color='black')
ax.set_ylabel('Actual outputs', fontsize=12, color='black')
ax.xaxis.set(ticks=range(10))
ax.yaxis.set(ticks=range(10))
ax.set_ylim(9.5, -0.5)
for i in range(10):
    for j in range(10):
        ax.text(j, i, cm[i, j], ha='center', va='center',
                color='white')
plt.show()
SAS.pyplot(plt)

```



Obtaining the classification report

Scikit-learn produces a report of the results of classifications by comparing the actual results of the test data in `y_test` with the predicted values in `y_pred`. The report provides information such as the support and precision of the classifications.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.93	1.00	0.96	27
1	0.92	0.97	0.94	35
2	1.00	0.83	0.91	36
3	0.81	1.00	0.89	29
4	1.00	0.97	0.98	30
5	0.92	0.85	0.88	40
6	1.00	1.00	1.00	44
7	0.91	1.00	0.95	39
8	1.00	0.87	0.93	39
9	0.83	0.83	0.83	41
accuracy			0.93	360
macro avg	0.93	0.93	0.93	360
weighted avg	0.93	0.93	0.93	360

ForestRegressor

Enables you to fit random forest models to your data and then predict new values.

ForestRegressor from SAS® Viya® on Bike Sharing Demand

Source

This example has been adapted from [EDA & Ensemble Model \(Top 10 Percentile\)](#) by Vivek Srinivasan.

Data Preparation

About the data set

A bicycle sharing system is a system of linked rental locations throughout a metropolitan area that allows individuals to share a pool of bicycles for their use. Bicycles rented from one location can be returned to any other location in the network. Joining the network and renting or returning bicycles are automated. As of 2022, there are around 3,000 cities offering such systems.

The data set contains information about the number of bikes rented by different types of users over several years tracked by hour. Additional information about each rental period includes weather information and categorization of the type of day.

```
import os
import pandas as pd
import numpy as np
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

Importing the data

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
print(bikeData.shape)
```

```
(17379, 16)
```

Sample of the first few rows

```
print(bikeData.head(5))
```

	date	season	year	month	...	windspeed	casual	registered	count
0	1/1/2011	1	0	1	...	0.0	3	13	16
1	1/1/2011	1	0	1	...	0.0	8	32	40
2	1/1/2011	1	0	1	...	0.0	5	27	32
3	1/1/2011	1	0	1	...	0.0	3	10	13
4	1/1/2011	1	0	1	...	0.0	0	1	1

[5 rows x 16 columns]

Variable data types

```
print(bikeData.dtypes)
```

```
[5 rows x 16 columns]
date          object
season        int64
year          int64
month         int64
hour          int64
holiday       int64
weekday       int64
workingday    int64
weather       int64
temp         float64
atemp        float64
humidity      float64
windspeed     float64
casual        int64
registered    int64
count         int64
dtype: object
```

Data Analysis

The results above show that the columns `hour`, `weekday`, `month`, `season`, `holiday`, `workingday`, and `weather` as `int64` data types when they should be of type categorical. In the following code, these columns are coerced from integers into the appropriate categorical types. In addition, the `date` column is dropped since it is not necessary for this analysis.

Coercing to categorical types and dropping unnecessary columns

Use the following code to coerce the data types to categorical type and drop the unnecessary columns.

```
bikeData["month"] =
bikeData.month.map({1:"January",2:"February",3:"March",4:"April",5:"May",
6:"June",7:"July",

8:"August",9:"September",10:"October",11:"November",12:"December"})
bikeData["weekday"] =
bikeData.weekday.map({0:"Sunday",1:"Monday",2:"Tuesday",3:"Wednesday",4
:"Thursday",5:"Friday",6:"Saturday"})
bikeData["season"] = bikeData.season.map({1: "Spring", 2 : "Summer",
3 : "Fall", 4 : "Winter" })
bikeData["weather"] = bikeData.weather.map({1: " Clear + Few clouds +
Partly cloudy + Partly cloudy",\

2 : " Mist + Cloudy, Mist +
Broken clouds, Mist + Few clouds, Mist ", \
3 : " Light Snow, Light Rain +
Thunderstorm + Scattered clouds, Light Rain + Scattered clouds", \
4 : " Heavy Rain + Ice Pallets
+ Thunderstorm + Mist, Snow + Fog " })
bikeData["workingday"] = bikeData.workingday.map({0: "Non-Working
Day", 1: "Working Day"})
```

The `drop()` method drops the unnecessary date column.

```
categoryVariableList =
["hour", "weekday", "month", "season", "holiday", "workingday", "weather"]
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("category")
bikeData = bikeData.drop(["date"], axis=1)
```

Outlier Analysis

Visualizing the outliers

To get a sense of how outliers might be distributed in the data, you can run a series of box plots, first for `count` by itself and then for `count` versus `season`, `hour`, and `workingday`.

The plots enable you to draw the following inferences about the data:

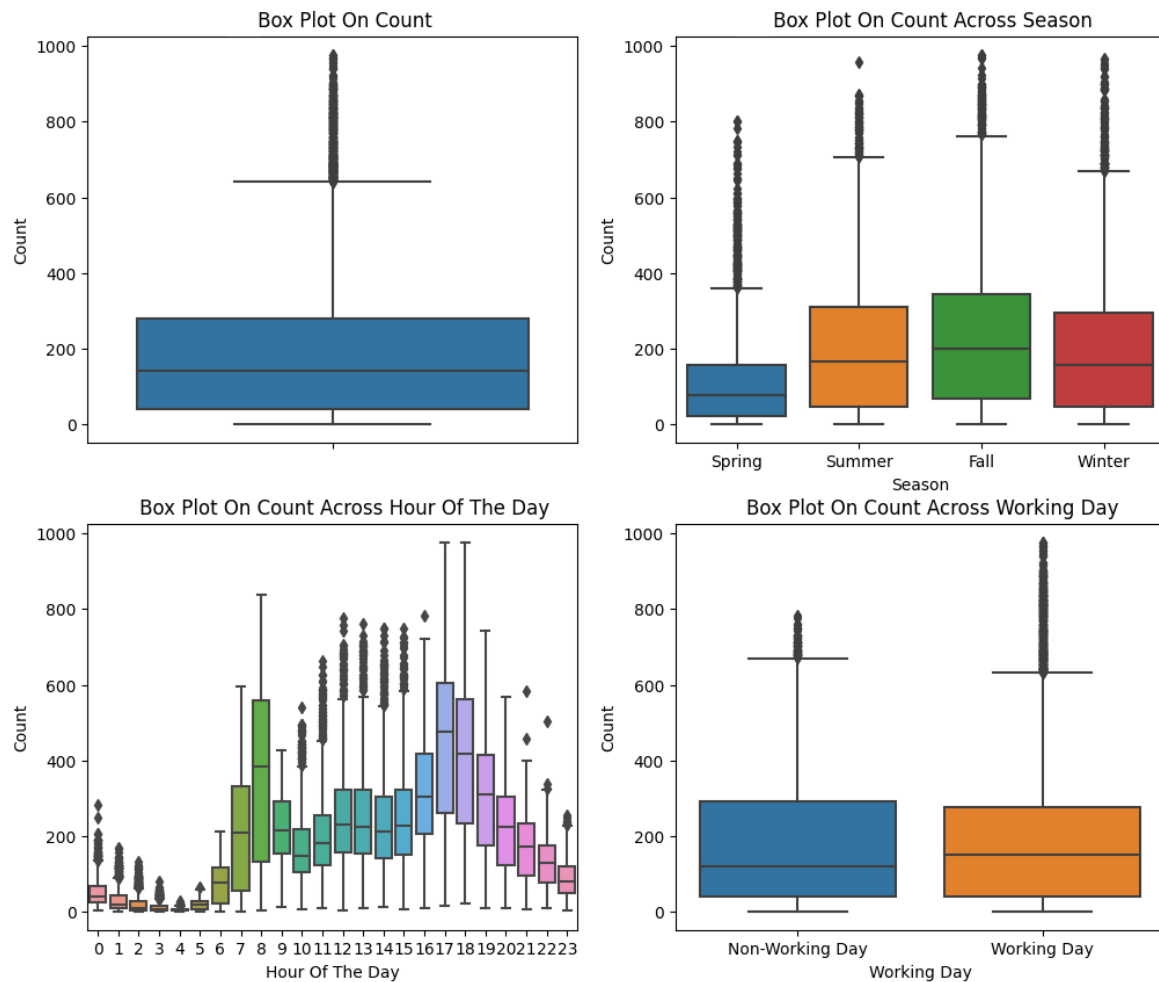
- `count` contains many outliers beyond the upper quartile limit, which skews the distribution toward higher values.

For `season`, *Spring* has a relatively lower `count`, demonstrated by the lower median value in its box plot.

- The box plot for *Hour Of The Day* has relatively higher median values for 0700, 0800, 1700, and 1800 hours. This likely matches up with commuter traffic for the start and end of school and office workdays.
- *Working Day* observations contribute a majority of the outliers, as opposed to *Non-Working Day* ones.

```
import matplotlib.pyplot as plt
import seaborn as sn
fig, axes = plt.subplots(nrows=2, ncols=2)
fig.set_size_inches(12, 10)
sn.boxplot(data=bikeData, y="count", orient="v", ax=axes[0][0])
sn.boxplot(data=bikeData, y="count", x="season", hue="season", orient="v", ax=axes[0][1])
sn.boxplot(data=bikeData, y="count", x="hour", hue="hour", orient="v", ax=axes[1][0])
sn.boxplot(data=bikeData, y="count", x="workingday", hue="workingday", orient="v", ax=axes[1][1])

axes[0][0].set(ylabel='Count', title="Box Plot On Count")
axes[0][1].set(xlabel='Season', ylabel='Count', title="Box Plot On Count Across Season")
axes[1][0].set(xlabel='Hour Of The Day', ylabel='Count', title="Box Plot On Count Across Hour Of The Day")
axes[1][1].set(xlabel='Working Day', ylabel='Count', title="Box Plot On Count Across Working Day")
SAS.pyplot(plt)
```



Removing outliers in the count variable

This code removes outliers in the count column.

```
import numpy as np
bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape of the data with outliers: ",bikeData.shape)
print ("Shape of the data without outliers:
",bikeDataWithoutOutliers.shape)
```

```
Shape of the data with outliers: (17379, 15)
Shape of the data without outliers: (17135, 15)
```

Correlation Analysis

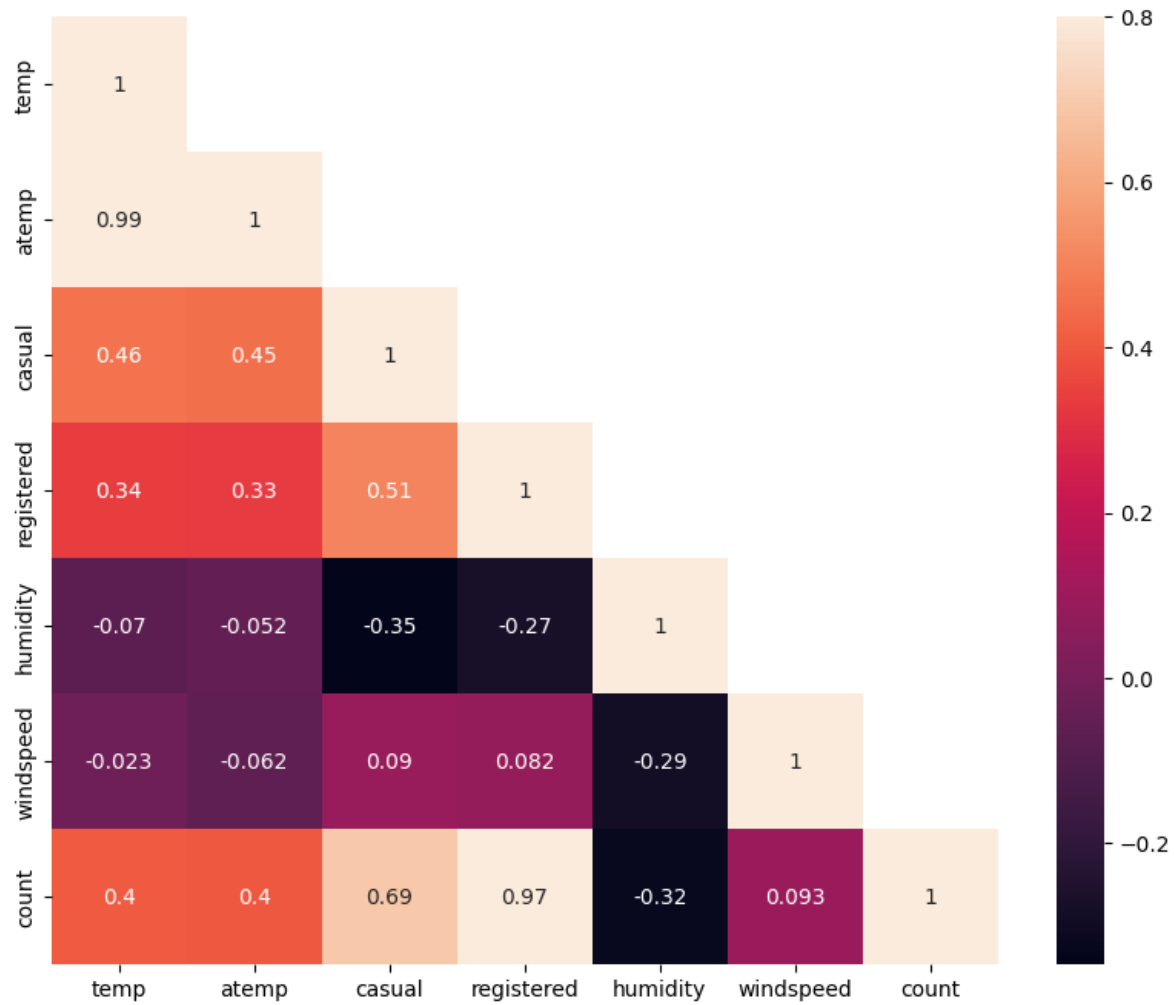
A common method for understanding numeric variables influence dependent variables is through constructing a correlation matrix. The correlation of count versus temp, atemp, humidity, and windspeed will be examined.

Viewing a heat map

By examining the heat map below, you can make the following observations:

- `temp` and `humidity` have a positive and negative correlation with `count`, respectively. Since the values are noticeably large, `count` does not depend much on either one of these features.
- Since `windspeed` has a very low correlation with `count`, the correlation adds little to the analysis.
- Since `atemp` (the measurement of what the temperature “feels like”) and `temp` (the thermometer reading) have a strong correlation with each other; however, including both of them would introduce multicollinearity into the analysis. With those definitions, one would expect this relationship, so `atemp` will be dropped.
- Because `casual` and `registered` are leakage variables in nature, they should also be dropped.

```
corrMatt =
bikeData[["temp", "atemp", "casual", "registered", "humidity", "windspeed", "
count"]].corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig, ax= plt.subplots()
fig.set_size_inches(10,8)
sn.heatmap(corrMatt, mask=mask, vmax=.8, square=True, annot=True)
SAS.pyplot(plt)
```



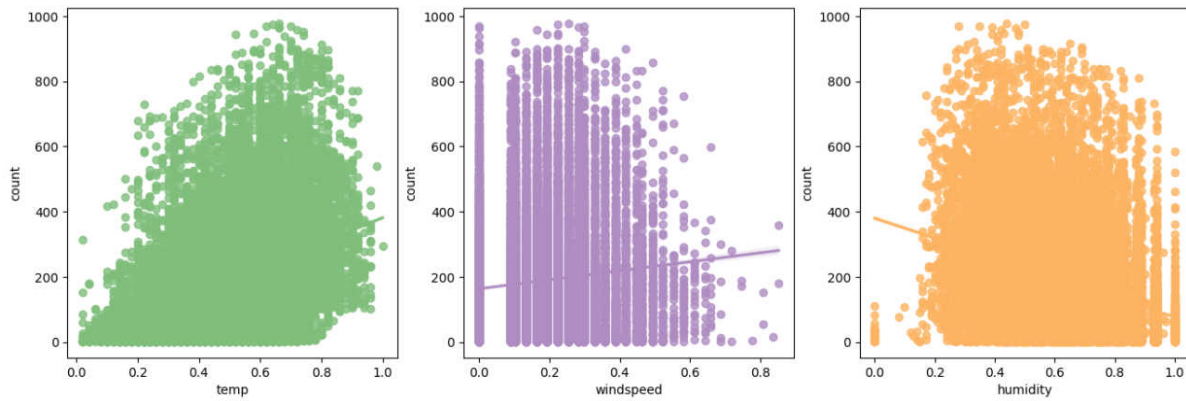
Regression plots

Regression plots from [Seaborn](#) are a useful way to depict the relationship between two features. You can also view these for count versus temp, atemp, humidity, and windspeed.

```
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3)
fig.set_size_inches(16, 5)

# Assigning softer colors to each plot
color1 = "#7fbf7b" # Light green
color2 = "#af8dc3" # Light purple
color3 = "#fdb462" # Light orange

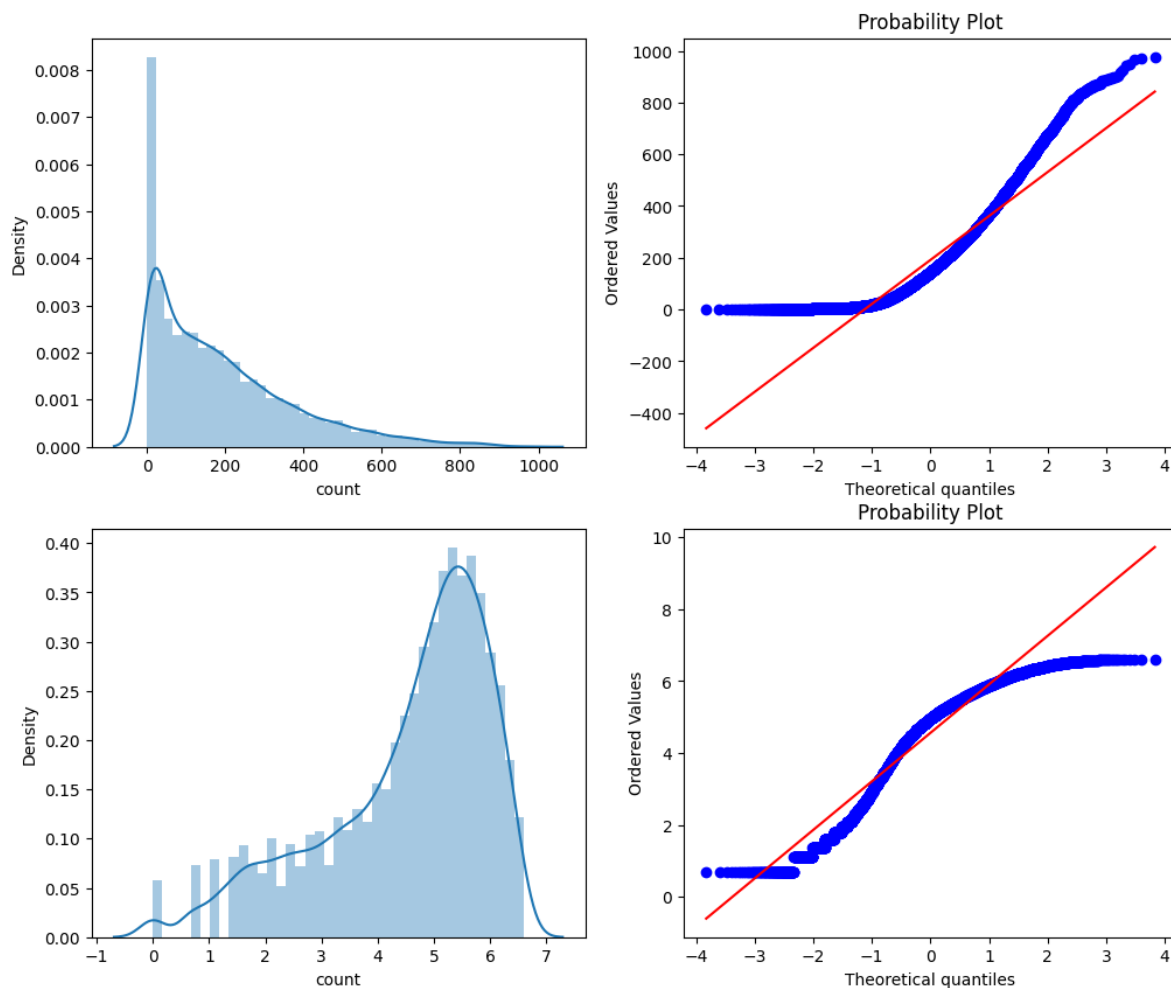
sn.regplot(x="temp", y="count", data=bikeData, ax=ax1, color=color1)
sn.regplot(x="windspeed", y="count", data=bikeData, ax=ax2,
color=color2)
sn.regplot(x="humidity", y="count", data=bikeData, ax=ax3, color=color3)
SAS.pyplot(plt)
```



Visualizing the Distribution of Data

As is visible from the figures below, "count" is skewed toward the right and not a normal distribution. Since many machine learning algorithms require a normal distribution, you will need to pursue options to address this. A common technique is to take a log transformation of "count" after removing the outliers. Afterward, the data distribution looks better but is still not an ideal normal.

```
from scipy import stats
fig, axes = plt.subplots(ncols=2, nrows=2)
fig.set_size_inches(12, 10)
sn.distplot(bikeData["count"], ax=axes[0][0])
stats.probplot(bikeData["count"], dist='norm', fit=True, plot=axes[0][1])
sn.distplot(np.log(bikeDataWithoutOutliers["count"]), ax=axes[1][0])
stats.probplot(np.log1p(bikeDataWithoutOutliers["count"]),
               dist='norm', fit=True, plot=axes[1][1])
fig.show()
SAS.pyplot(plt)
```



Visualizing Count versus Month, Season, Hour, and Weekday

The plots help show the following information:

- People tend to rent bikes during the summer so that June, July, and August have a relatively higher count (or demand).
- On weekdays, more people rent bicycles for the 0700, 0800, 1700, and 1800 hours. This overlaps with normal school and office schedules.
- The renters during this time are registered users.
- On weekend days, more bicycles are rented between 1000 and 1600.

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=4)
fig.set_size_inches(12, 20)
sortOrder =
["January", "February", "March", "April", "May", "June", "July", "August", "Sep
tember", "October", "November", "December"]
hueOrder =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

```

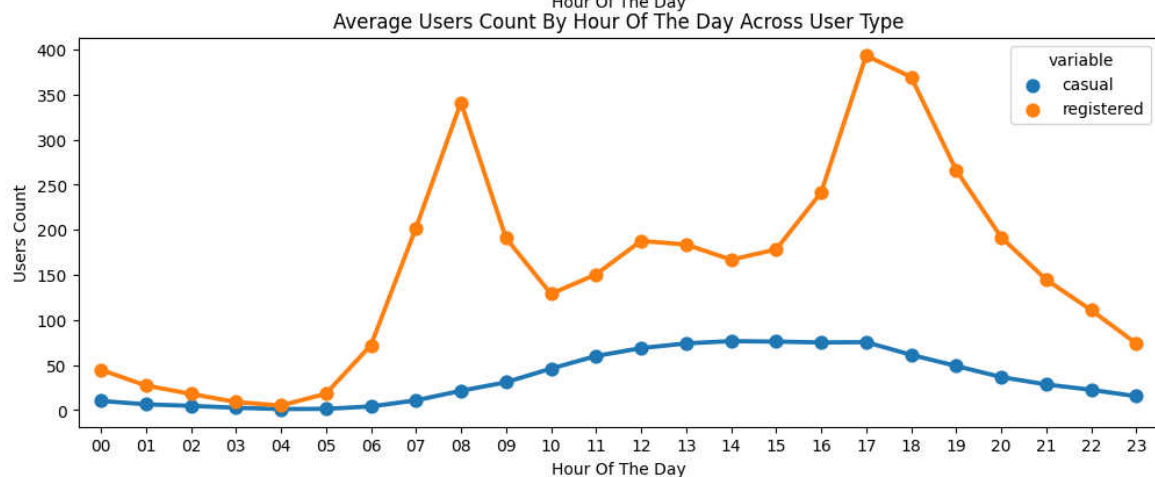
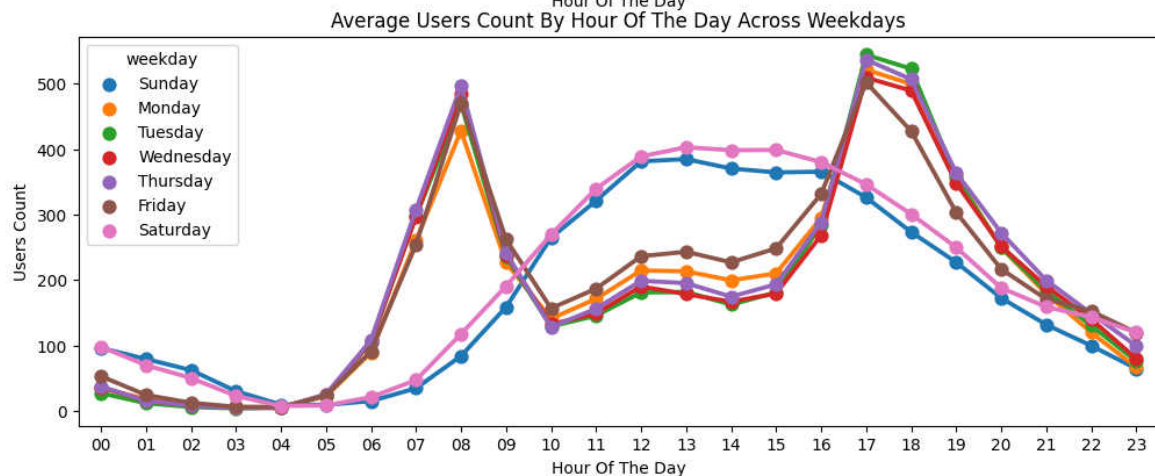
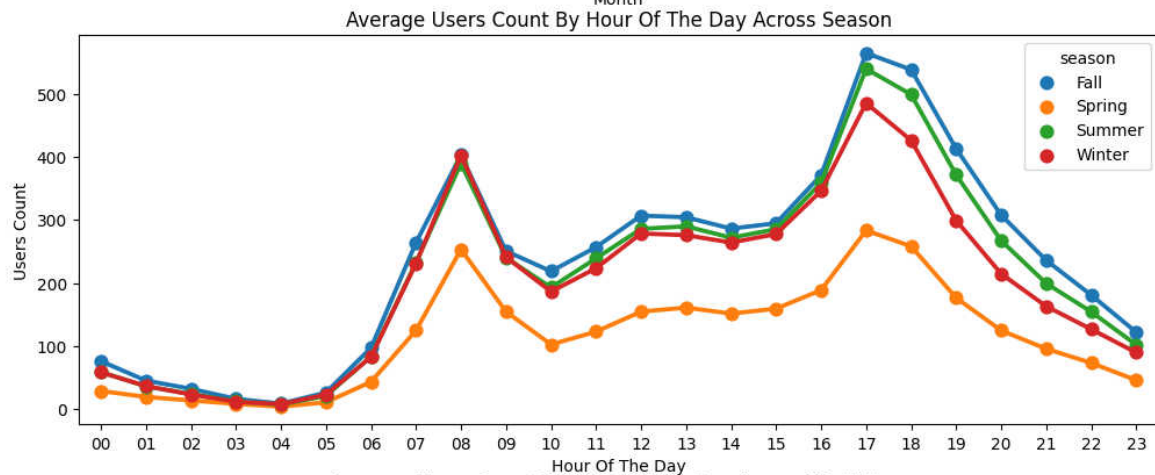
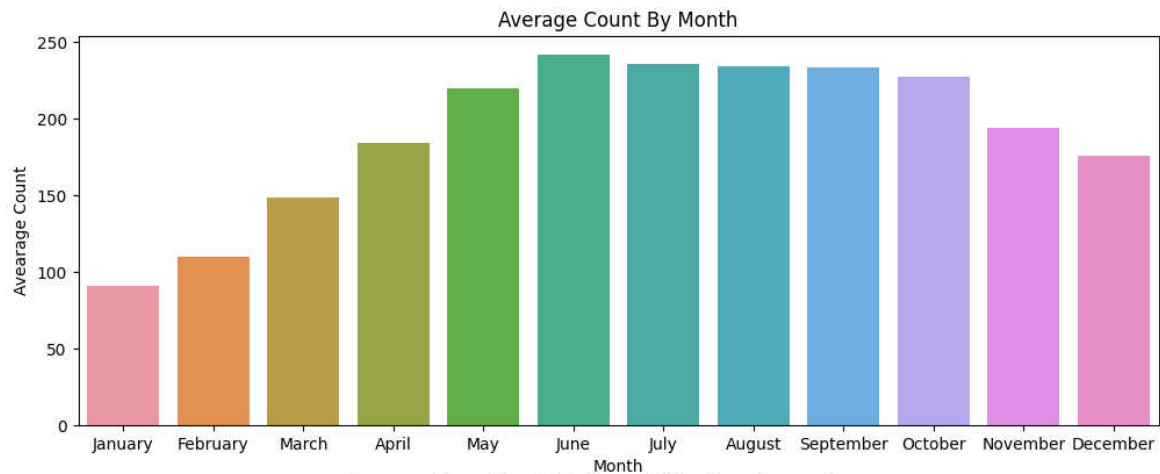
monthAggregated = pd.DataFrame(bikeData.groupby("month")
["count"].mean()).reset_index()
monthSorted = monthAggregated.sort_values(by="count",ascending=False)
sn.barplot(data=monthSorted,x="month",y="count",hue="month",ax=ax1,order=sortOrder)
ax1.set(xlabel='Month', ylabel='Average Count',title="Average Count By Month")

hourAggregated =
pd.DataFrame(bikeData.groupby(["hour","season"],sort=True)
["count"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["count"],hue=hourAggregated["season"],
data=hourAggregated, join=True,ax=ax2)
ax2.set(xlabel='Hour Of The Day', ylabel='Users Count',title="Average Users Count By Hour Of The Day Across Season",label='big')

hourAggregated =
pd.DataFrame(bikeData.groupby(["hour","weekday"],sort=True)
["count"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["count"],hue=hourAggregated["weekday"],hue_order=hueOrder, data=hourAggregated, join=True,ax=ax3)
ax3.set(xlabel='Hour Of The Day', ylabel='Users Count',title="Average Users Count By Hour Of The Day Across Weekdays",label='big')

hourTransformed = pd.melt(bikeData[["hour","casual","registered"]],
id_vars=['hour'], value_vars=['casual', 'registered'])
hourAggregated =
pd.DataFrame(hourTransformed.groupby(["hour","variable"],sort=True)
["value"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["value"],hue=hourAggregated["variable"],hue_order=["casual","registered"], data=hourAggregated, join=True,ax=ax4)
ax4.set(xlabel='Hour Of The Day', ylabel='Users Count',title="Average Users Count By Hour Of The Day Across User Type",label='big')
fig.show()
SAS.pyplot(plt)

```



Integrating Data Discoveries

Loading the data

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Defining feature lists

```
categoricalFeatureNames =
["season", "holiday", "workingday", "weather", "weekday", "month", "year", "hour"]
numericalFeatureNames = ["temp", "humidity", "windspeed", "atemp"]
dropFeatures = ["casual", "date", "registered"]
```

Dropping unnecessary variables

```
bikeData = bikeData.drop(dropFeatures, axis=1)
```

Creating training and test data

Split the original data by putting 75% into the training set and 25% into the test set.

```
X = bikeData.drop('count', axis=1)
Y = bikeData['count']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=3)
```

Ensemble Models - ForestRegressor

```
from sasviya.ml.tree import ForestRegressor

rfModel = ForestRegressor(n_estimators=100,
                           max_depth=25,
                           min_samples_leaf=1,
                           max_features=None,
                           criterion='variance',
                           random_state=0)

rfModel.fit(X_train, y_train, nominals = categoricalFeatureNames)
print("Forest Regressor -Test Set Score:
{:.2f}".format(rfModel.score(X_test, y_test)))
```

```
Forest Regressor -Test Set Score: 0.91
```

ElasticNet

Enables you to fit a linear regression model with a mix of L1 and L2 regularization and then predict new values.

Train an Elastic Net Regression Model

Introduction

This example applies the Elastic Net Regression technique to the Bike Sharing Demand data set and compares the training set score and test set score before and after using the technique.

Set up environment

It is first necessary to set up the environment for working with machine learning models. The `import` commands load the `os` and `pandas`, and `numpy` libraries, the `train_test_split` function from the `scikit-learn` library, the `warnings` library, and the `ConvergenceWarning` exception from the `scikit-learn` library. The `warnings.filterwarnings` command filters out `ConvergenceWarning` messages, which are not critical for this analysis.

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

import warnings
from sklearn.exceptions import ConvergenceWarning

# Filter out ConvergenceWarning messages
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

Data Preparation

Importing the data

The `pd.read_csv()` function reads the `bike_sharing_demand.csv` file and loads it to a `DataFrame` named `bikeData`.


```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Dropping unnecessary variables

```
dropFeatures = ["casual", "date", "registered"]
bikeData = bikeData.drop(dropFeatures, axis=1)
```

Coercing variables to category type

```
categoryVariableList =
["hour", "weekday", "month", "season", "holiday", "workingday", "weather"]
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("object")
print(bikeData.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 13 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   season      17379 non-null  object
1   year        17379 non-null  int64
2   month       17379 non-null  object
3   hour        17379 non-null  object
4   holiday     17379 non-null  object
5   weekday     17379 non-null  object
6   workingday  17379 non-null  object
7   weather     17379 non-null  object
8   temp        17379 non-null  float64
9   atemp       17379 non-null  float64
10  humidity    17379 non-null  float64
11  windspeed   17379 non-null  float64
12  count       17379 non-null  int64
dtypes: float64(4), int64(2), object(7)
memory usage: 1.7+ MB
```

Applying OneHotEncoder to convert nominal features to numeric values

```
from sklearn.preprocessing import OneHotEncoder

### OneHotEncoder
OHEncoder = OneHotEncoder(sparse_output=False)

# Get the list of nominal columns
nominal_columns =
bikeData.select_dtypes(include=['object']).columns.tolist()
# Get the list of numeric columns
numeric_columns =
bikeData.select_dtypes(exclude=['object']).columns.tolist()

nominal_X = OHEncoder.fit_transform(bikeData[nominal_columns])
nominal_X_df = pd.DataFrame(nominal_X,
columns=OHEncoder.get_feature_names_out(nominal_columns))
```

```
bikeData = pd.concat([nominal_X_df, bikeData[numeric_columns]], axis=1)

print(bikeData.columns.tolist())
```

```
['season_1', 'season_2', 'season_3', 'season_4', 'month_1', 'month_2', 'month_3',
'month_4', 'month_5', 'month_6', 'month_7', 'month_8', 'month_9', 'month_10',
'month_11', 'month_12', 'hour_0', 'hour_1', 'hour_2', 'hour_3', 'hour_4',
'hour_5', 'hour_6', 'hour_7', 'hour_8', 'hour_9', 'hour_10', 'hour_11',
'hour_12', 'hour_13', 'hour_14', 'hour_15', 'hour_16', 'hour_17', 'hour_18',
'hour_19', 'hour_20', 'hour_21', 'hour_22', 'hour_23', 'holiday_0', 'holiday_1',
'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4', 'weekday_5',
'weekday_6', 'workingday_0', 'workingday_1', 'weather_1', 'weather_2',
'weather_3', 'weather_4', 'year', 'temp', 'atemp', 'humidity', 'windspeed',
'count']
```

Removing outliers in the count column

```
bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape Of The Before Outliers: ", bikeData.shape)
print ("Shape Of The After Outliers: ", bikeDataWithoutOutliers.shape)
```

```
Shape Of The Before Outliers: (17379, 61)
Shape Of The After Outliers: (17135, 61)
```

Creating Training and Testing Data

The `bikeDataWithoutOutliers` DataFrame is split into two parts, `X` and `y`, representing the input features and the target variable, respectively. The `train_test_split()` function splits the data set into training and testing subsets for both features (`x`) and labels (`y`). The `test_size` parameter specifies that 25% of the total data set will be the testing set. The `random_state` parameter specifies that different runs of the code return the same data split.

```
X = bikeDataWithoutOutliers.drop(columns=['count'])
y = bikeDataWithoutOutliers['count']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=0)
```

Creating the Model

The `import` command imports the `ElasticNet` class from the `sasviya.ml.linear_model` module. The `ElasticNet().fit()` function runs the elastic net regression algorithm on the training data to create a model. The `alpha` parameter controls the strength of regularization. The `l1_ratio` parameter controls the balance between L1 and L2 regularization. The two `print()` functions print the

training set score and the test set score, respectively. The `score()` method calculates and returns the coefficient of determination (R-squared) for the model on the training and test sets.

```
from sasviya.ml.linear_model import ElasticNet

elastic_net = ElasticNet(alpha=0.01, l1_ratio=0.01).fit(X_train,
y_train)

print(f"Elastic Net-Training set score: {elastic_net.score(X_train,
y_train):.2f}")
print(f"Elastic Net-Test set score: {elastic_net.score(X_test,
y_test):.2f}")
```

```
Elastic Net-Training set score: 0.67
Elastic Net-Test set score: 0.67
```

Lasso

Enables you to fit a linear regression model with L1 regularization and then predict new values.

Train a Lasso Regression Model

Introduction

This example applies the Lasso Regression technique to the Bike Sharing Demand data set and compares the training set score and test set score before and after using the technique.

Set up environment

It is first necessary to set up the environment for working with machine learning models. The `import` commands load the `os` and `pandas`, and `numpy` libraries, the `train_test_split` function from the `scikit-learn` library, the `warnings` library, and the `ConvergenceWarning` exception from the `scikit-learn` library. The `warnings.filterwarnings` command filters out `ConvergenceWarning` messages, which are not critical for this analysis.

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
import warnings
from sklearn.exceptions import ConvergenceWarning

# Filter out ConvergenceWarning messages
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

Data Preparation

Importing the data

The `pd.read_csv()` function reads the `bike_sharing_demand.csv` file and loads it to a `DataFrame` named `bikeData`.

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Dropping unnecessary variables

```
dropFeatures = ["casual", "date", "registered"]
bikeData = bikeData.drop(dropFeatures, axis=1)
```

Coercing variables to category type

```
categoryVariableList =
["hour", "weekday", "month", "season", "holiday", "workingday", "weather"]
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("object")
print(bikeData.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 13 columns):
#   Column      Non-Null Count  Dtype
---  -
0   season      17379 non-null  object
1   year        17379 non-null  int64
2   month       17379 non-null  object
3   hour        17379 non-null  object
4   holiday     17379 non-null  object
5   weekday     17379 non-null  object
6   workingday  17379 non-null  object
7   weather     17379 non-null  object
8   temp        17379 non-null  float64
9   atemp       17379 non-null  float64
10  humidity    17379 non-null  float64
11  windspeed   17379 non-null  float64
12  count       17379 non-null  int64
dtypes: float64(4), int64(2), object(7)
memory usage: 1.7+ MB
```

Applying OneHotEncoder to convert nominal features to numeric values

```
from sklearn.preprocessing import OneHotEncoder

### OneHotEncoder
OHEncoder = OneHotEncoder(sparse_output=False)

# Get the list of nominal columns
nominal_columns =
bikeData.select_dtypes(include=['object']).columns.tolist()
# Get the list of numeric columns
numeric_columns =
bikeData.select_dtypes(exclude=['object']).columns.tolist()

nominal_X = OHEncoder.fit_transform(bikeData[nominal_columns])
nominal_X_df = pd.DataFrame(nominal_X,
columns=OHEncoder.get_feature_names_out(nominal_columns))
bikeData = pd.concat([nominal_X_df, bikeData[numeric_columns]], axis=1)

print(bikeData.columns.tolist())
```

```
['season_1', 'season_2', 'season_3', 'season_4', 'month_1', 'month_2', 'month_3',
'month_4', 'month_5', 'month_6', 'month_7', 'month_8', 'month_9', 'month_10',
'month_11', 'month_12', 'hour_0', 'hour_1', 'hour_2', 'hour_3', 'hour_4',
'hour_5', 'hour_6', 'hour_7', 'hour_8', 'hour_9', 'hour_10', 'hour_11',
'hour_12', 'hour_13', 'hour_14', 'hour_15', 'hour_16', 'hour_17', 'hour_18',
'hour_19', 'hour_20', 'hour_21', 'hour_22', 'hour_23', 'holiday_0', 'holiday_1',
'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4', 'weekday_5',
'weekday_6', 'workingday_0', 'workingday_1', 'weather_1', 'weather_2',
'weather_3', 'weather_4', 'year', 'temp', 'atemp', 'humidity', 'windspeed',
'count']
```

Removing outliers in the count column

```
bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape Of The Before Outliers: ", bikeData.shape)
print ("Shape Of The After Outliers: ", bikeDataWithoutOutliers.shape)
```

```
Shape Of The Before Outliers: (17379, 61)
Shape Of The After Outliers: (17135, 61)
```

Creating Training and Testing Data

The `bikeDataWithoutOutliers` DataFrame is split into two parts, `x` and `y`, representing the input features and the target variable, respectively. The `train_test_split()` function splits the data set into training and testing subsets for both features (`x`) and labels (`y`). The `test_size` parameter specifies that 25% of

the total data set will be the testing set. The `random_state` parameter specifies that different runs of the code return the same data split.

```
X = bikeDataWithoutOutliers.drop(columns=['count'])
y = bikeDataWithoutOutliers['count']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=0)
```

Creating the Model

The `import` command imports the `Lasso` class from the `sasviya.ml.linear_model` module. The `Lasso().fit()` function runs the lasso regression algorithm on the training data to create a model. The two `print()` functions print the training set score and the test set score, respectively. The `score()` method calculates and returns the coefficient of determination (R-squared) for the model on the training and test sets.

```
from sasviya.ml.linear_model import Lasso

lasso = Lasso().fit(X_train, y_train)

print(f"Lasso Regression-Training set score: {lasso.score(X_train,
y_train):.2f}")
print(f"Lasso Regression-Test set score: {lasso.score(X_test,
y_test):.2f}")
```

```
Lasso Regression-Training set score: 0.67
Lasso Regression-Test set score: 0.67
```

As shown, lasso regression performs quite disappointingly, and it is a sign of underfitting. The lasso model does not work well because most of the coefficients have become exactly zero. The following code shows the exact number of features that have been used in the model:

```
print(f"Number of features: {sum(lasso.coef_.to_numpy()[0] != 0)}")
```

```
Number of features: 48
```

This indicates that only 48 of the 60 features in the training set are used in the lasso regression model, whereas the rest are ignored.

```
lasso = Lasso().fit(X_train, y_train)
print("Lasso Regression-Training set score:
{:.2f}".format(lasso.score(X_train, y_train)))
print("Lasso Regression-Test set score:
{:.2f}".format(lasso.score(X_test, y_test)))
```

```
Lasso Regression-Training set score: 0.67
Lasso Regression-Test set score: 0.67
```

LinearRegression

Enables you to fit a linear regression model and then predict new values.

Train a Linear Regression Model

Introduction

This example applies the Linear Regression technique to the Bike Sharing Demand data set and compares the training set score and test set score before and after using the technique.

Set up environment

It is first necessary to set up the environment for working with machine learning models. The `import` commands load the `os` and `pandas`, and `numpy` libraries, the `train_test_split` function from the `scikit-learn` library, the `warnings` library, and the `ConvergenceWarning` exception from the `scikit-learn` library. The `warnings.filterwarnings` command filters out `ConvergenceWarning` messages, which are not critical for this analysis.

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

import warnings
from sklearn.exceptions import ConvergenceWarning

# Filter out ConvergenceWarning messages
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

Data Preparation

Importing the data

The `pd.read_csv()` function reads the `bike_sharing_demand.csv` file loads it to a `DataFrame` named `bikeData`.

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Dropping unnecessary variables

```
dropFeatures = ["casual", "date", "registered"]
bikeData = bikeData.drop(dropFeatures, axis=1)
```

Coercing variables to category type

```
categoryVariableList =
["hour", "weekday", "month", "season", "holiday", "workingday", "weather"]
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("object")
print(bikeData.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 13 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   season      17379 non-null  object
1   year        17379 non-null  int64
2   month       17379 non-null  object
3   hour        17379 non-null  object
4   holiday     17379 non-null  object
5   weekday     17379 non-null  object
6   workingday  17379 non-null  object
7   weather     17379 non-null  object
8   temp        17379 non-null  float64
9   atemp       17379 non-null  float64
10  humidity    17379 non-null  float64
11  windspeed   17379 non-null  float64
12  count       17379 non-null  int64
dtypes: float64(4), int64(2), object(7)
memory usage: 1.7+ MB
```

Applying OneHotEncoder to convert nominal features to numeric values

```
from sklearn.preprocessing import OneHotEncoder

### OneHotEncoder
OHEncoder = OneHotEncoder(sparse_output=False)

# Get the list of nominal columns
nominal_columns =
bikeData.select_dtypes(include=['object']).columns.tolist()
# Get the list of numeric columns
numeric_columns =
bikeData.select_dtypes(exclude=['object']).columns.tolist()

nominal_X = OHEncoder.fit_transform(bikeData[nominal_columns])
nominal_X_df = pd.DataFrame(nominal_X,
columns=OHEncoder.get_feature_names_out(nominal_columns))
```



```
bikeData = pd.concat([nominal_X_df, bikeData[numeric_columns]], axis=1)

print(bikeData.columns.tolist())
```

```
['season_1', 'season_2', 'season_3', 'season_4', 'month_1', 'month_2', 'month_3',
'month_4', 'month_5', 'month_6', 'month_7', 'month_8', 'month_9', 'month_10',
'month_11', 'month_12', 'hour_0', 'hour_1', 'hour_2', 'hour_3', 'hour_4',
'hour_5', 'hour_6', 'hour_7', 'hour_8', 'hour_9', 'hour_10', 'hour_11',
'hour_12', 'hour_13', 'hour_14', 'hour_15', 'hour_16', 'hour_17', 'hour_18',
'hour_19', 'hour_20', 'hour_21', 'hour_22', 'hour_23', 'holiday_0', 'holiday_1',
'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4', 'weekday_5',
'weekday_6', 'workingday_0', 'workingday_1', 'weather_1', 'weather_2',
'weather_3', 'weather_4', 'year', 'temp', 'atemp', 'humidity', 'windspeed',
'count']
```

Removing outliers in the count column

```
bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape Of The Before Outliers: ", bikeData.shape)
print ("Shape Of The After Outliers: ", bikeDataWithoutOutliers.shape)
```

```
Shape Of The Before Outliers: (17379, 61)
Shape Of The After Outliers: (17135, 61)
```

Creating Training and Testing Data

The `bikeDataWithoutOutliers` DataFrame is split into two parts, `X` and `y`, representing the input features and the target variable, respectively. The `train_test_split()` function splits the data set into training and testing subsets for both features (`x`) and labels (`y`). The `test_size` parameter specifies that 25% of the total data set will be the testing set. The `random_state` parameter specifies that different runs of the code return the same data split.

```
X = bikeDataWithoutOutliers.drop(columns=['count'])
y = bikeDataWithoutOutliers['count']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=0)
```

Creating the Model

The `import` command imports the `LinearRegression` class from the `sasviya.ml.linear_model` module. The `LinearRegression().fit()` function runs the linear regression algorithm on the training data. The two `print()` functions print the training set score and the test set score, respectively. The `score()`

method calculates and returns the coefficient of determination (R-squared) for the model on the training and test sets.

```
from sasviya.ml.linear_model import LinearRegression

lr = LinearRegression().fit(X_train, y_train)

print(f"Linear Regression-Training set score: {lr.score(X_train,
y_train):.2f}")
print(f"Linear Regression-Test set score: {lr.score(X_test,
y_test):.2f}")
```

```
Linear Regression-Training set score: 0.67
Linear Regression-Test set score: 0.67
```

Ridge

Enables you to fit a linear regression model with L2 regularization and then predict new values.

Train a Ridge Regression Model

Introduction

This example applies the Ridge Regression technique to the Bike Sharing Demand data set and compares the training set score and test set score before and after using the technique.

Set up environment

It is first necessary to set up the environment for working with machine learning models. The `import` commands load the `os` and `pandas`, and `numpy` libraries, the `train_test_split` function from the `scikit-learn` library, the `warnings` library, and the `ConvergenceWarning` exception from the `scikit-learn` library. The `warnings.filterwarnings` command filters out `ConvergenceWarning` messages, which are not critical for this analysis.

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

import warnings
from sklearn.exceptions import ConvergenceWarning
```

```
# Filter out ConvergenceWarning messages
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

Data Preparation

Importing the data

The `pd.read_csv()` function reads the `bike_sharing_demand.csv` file and loads it to a `DataFrame` named `bikeData`.

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Dropping unnecessary variables

```
dropFeatures = ["casual", "date", "registered"]
bikeData = bikeData.drop(dropFeatures, axis=1)
```

Coercing variables to category type

```
categoryVariableList =
["hour", "weekday", "month", "season", "holiday", "workingday", "weather"]
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("object")
print(bikeData.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17379 entries, 0 to 17378
Data columns (total 13 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   season      17379 non-null  object
1   year        17379 non-null  int64
2   month       17379 non-null  object
3   hour        17379 non-null  object
4   holiday     17379 non-null  object
5   weekday     17379 non-null  object
6   workingday  17379 non-null  object
7   weather     17379 non-null  object
8   temp        17379 non-null  float64
9   atemp       17379 non-null  float64
10  humidity    17379 non-null  float64
11  windspeed   17379 non-null  float64
12  count       17379 non-null  int64
dtypes: float64(4), int64(2), object(7)
memory usage: 1.7+ MB
```

Applying OneHotEncoder to convert nominal features to numeric values

```
from sklearn.preprocessing import OneHotEncoder
```

```

### OneHotEncoder
OHEncoder = OneHotEncoder(sparse_output=False)

# Get the list of nominal columns
nominal_columns =
bikeData.select_dtypes(include=['object']).columns.tolist()
# Get the list of numeric columns
numeric_columns =
bikeData.select_dtypes(exclude=['object']).columns.tolist()

nominal_X = OHEncoder.fit_transform(bikeData[nominal_columns])
nominal_X_df = pd.DataFrame(nominal_X,
columns=OHEncoder.get_feature_names_out(nominal_columns))
bikeData = pd.concat([nominal_X_df, bikeData[numeric_columns]], axis=1)

print(bikeData.columns.tolist())

```

```

['season_1', 'season_2', 'season_3', 'season_4', 'month_1', 'month_2', 'month_3',
'month_4', 'month_5', 'month_6', 'month_7', 'month_8', 'month_9', 'month_10',
'month_11', 'month_12', 'hour_0', 'hour_1', 'hour_2', 'hour_3', 'hour_4',
'hour_5', 'hour_6', 'hour_7', 'hour_8', 'hour_9', 'hour_10', 'hour_11',
'hour_12', 'hour_13', 'hour_14', 'hour_15', 'hour_16', 'hour_17', 'hour_18',
'hour_19', 'hour_20', 'hour_21', 'hour_22', 'hour_23', 'holiday_0', 'holiday_1',
'weekday_0', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4', 'weekday_5',
'weekday_6', 'workingday_0', 'workingday_1', 'weather_1', 'weather_2',
'weather_3', 'weather_4', 'year', 'temp', 'atemp', 'humidity', 'windspeed',
'count']

```

Removing outliers in the count column

```

bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape Of The Before Outliers: ", bikeData.shape)
print ("Shape Of The After Outliers: ", bikeDataWithoutOutliers.shape)

```

```

Shape Of The Before Outliers: (17379, 61)
Shape Of The After Outliers: (17135, 61)

```

Creating Training and Testing Data

The `bikeDataWithoutOutliers` DataFrame is split into two parts, `X` and `y`, representing the input features and the target variable, respectively. The `train_test_split()` function splits the data set into training and testing subsets for both features (`x`) and labels (`y`). The `test_size` parameter specifies that 25% of the total data set will be the testing set. The `random_state` parameter specifies that different runs of the code return the same data split.

```

X = bikeDataWithoutOutliers.drop(columns=['count'])
y = bikeDataWithoutOutliers['count']

```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=0)
```

Creating the Model

Train the Linear regression model and then print the training set score and the test set score. Score returns the coefficient of determination R^2 of the prediction.

```
from sasviya.ml.linear_model import LinearRegression

lr = LinearRegression().fit(X_train, y_train)

print(f"Linear Regression-Training set score: {lr.score(X_train,
y_train):.2f}")
print(f"Linear Regression-Test set score: {lr.score(X_test,
y_test):.2f}")
```

```
Linear Regression-Training set score: 0.67
Linear Regression-Test set score: 0.67
```

To avoid overfitting and control the complexity of the model, use ridge regression (L2 regularization) and see how well it does on the data set:

```
from sasviya.ml.linear_model import Ridge

ridge = Ridge(alpha=0.7).fit(X_train, y_train)
print(f"Ridge Regression-Training set score: {ridge.score(X_train,
y_train):.2f}")
print(f"Ridge Regression-Test set score: {ridge.score(X_test,
y_test):.2f}")
```

The import command imports the Ridge class from the `sasviya.ml.linear_model` module. The `Ridge().fit()` function runs the ridge regression algorithm on the training data to create a model. The `alpha` parameter controls the strength of regularization. The two `print()` functions print the training set score and the test set score, respectively. The `score()` method calculates and returns the coefficient of determination (R-squared) for the model on the training and test sets.

```
Ridge Regression-Training set score: 0.59
Ridge Regression-Test set score: 0.59
```

The training set and test set scores of ridge regression are significantly lower than the linear regression score. These scores confirm that ridge regression reduces the model's complexity, leading to a less-overfit-but-more-general model.

```
from sasviya.ml.linear_model import Ridge

ridge = Ridge().fit(X_train, y_train)
print(f"Ridge Regression-Training set score: {ridge.score(X_train,
y_train):.2f}")
print(f"Ridge Regression-Test set score: {ridge.score(X_test,
y_test):.2f}")
```

```
Ridge Regression-Training set score: 0.67
Ridge Regression-Test set score: 0.67
```

The alpha parameter specifies a trade-off between the model's performance on the training set and its simplicity. So, increasing the alpha value (its default value is 1.0) simplifies the model by shrinking the coefficients.

GradientBoostingClassifier

Enables you to fit a gradient boosting model to your data and then predict and score new observations.

GradientBoostingClassifier from SAS® Viya® on Banking

Source

This example is adapted from [Random Forest Classification with Scikit-Learn](#) by Adam Shafi.

About the Data Set

The data set for this example is sourced from the [UC Irvine Machine Learning Repository](#). It represents direct marketing campaigns (phone calls) conducted by a Portuguese banking institution. The objective of classification is to predict whether a client subscribes (1/0) to a term deposit (variable *y*).

The data set contains customer information consisting of 41,188 records and 21 fields.

```
import os
import pandas as pd
import numpy as np

df = pd.read_csv('https://raw.githubusercontent.com/sassoftware/sas-
viya-workbench-examples/refs/heads/main/data/banking_raw.csv')

print(df.head())
```

```

   age      job  marital  ... euribor3m  nr_employed  y
0   44  blue-collar  married  ...    4.963    5228.1    0
1   53  technician  married  ...    4.021    5195.8    0
2   28   management   single  ...    0.729    4991.6    1
3   39   services    married  ...    1.405    5099.1    0
4   55    retired    married  ...    0.869    5076.2    1
[5 rows x 21 columns]

```

Gradient Boosting Workflow

To fit and train this model, perform the following steps:

- split the data
- train the model
- perform hyperparameter tuning
- assess the model's performance

Note that there is no need to convert all non-numeric features (for example, month, education) into numeric ones for model fitting. The GradientBoostingClassifier provides native support for categorical features.

Split into predictor and response dataframes

```

X_df = df.drop('y', axis=1)
y = df['y']

print(X_df.shape, y.shape)

```

```
((41188, 20), (41188,))
```

```
print(y.value_counts())
```

```

y
0    36548
1     4640
Name: count, dtype: int64

```

Splitting the Data

When training any supervised learning model, it is important to split the data into training and test data. The training data is used to fit the model. The algorithm uses the training data to learn the relationship between the features and the target. The test data is used to evaluate the performance of the model.

The code below splits the data into separate variables for the features and target. It then splits the data into training and test data.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_df, y, test_size
= 0.2, random_state = 10)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(32950, 20) (8238, 20) (32950,) (8238,)
```

Fitting and Evaluating the Model

Create an instance of the gradient boosting model with the default parameters. Then, fit this to your training data. Pass both the features and the target variable to the model so that the model can learn.

For details about using the `GradientBoostingClassifier` class in the `sasviya` package, see the [GradientBoostingClassifier documentation](#).

```
from sasviya.ml.tree import GradientBoostingClassifier

gb = GradientBoostingClassifier(n_estimators=100,
                                max_depth=5,
                                min_samples_leaf=1,
                                max_features=None,
                                learning_rate = 0.1,
                                subsample = 1.0,
                                random_state=0,
                                calc_feature_importances=True)

gb.fit(X_train, y_train)
```

```
GradientBoostingClassifier(min_samples_leaf=1, max_depth=5, subsample=1.0,
calc_feature_importances=True)
```

At this point, you have a trained gradient boosting model, but you still need to determine whether the model is making accurate predictions.

```
y_pred = gb.predict(X_test)
```

The simplest way to evaluate this model is by using accuracy; that is, you check the predictions against the actual values in the test set and count how many predictions the model predicted correctly.

```
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred.astype(type(y_test[0])))
print("Accuracy:", accuracy)
```

```
Accuracy: 0.92
```

ROC Curve

You can visualize the performance using an ROC curve.

```
from sklearn import metrics
import matplotlib.pyplot as plt
```

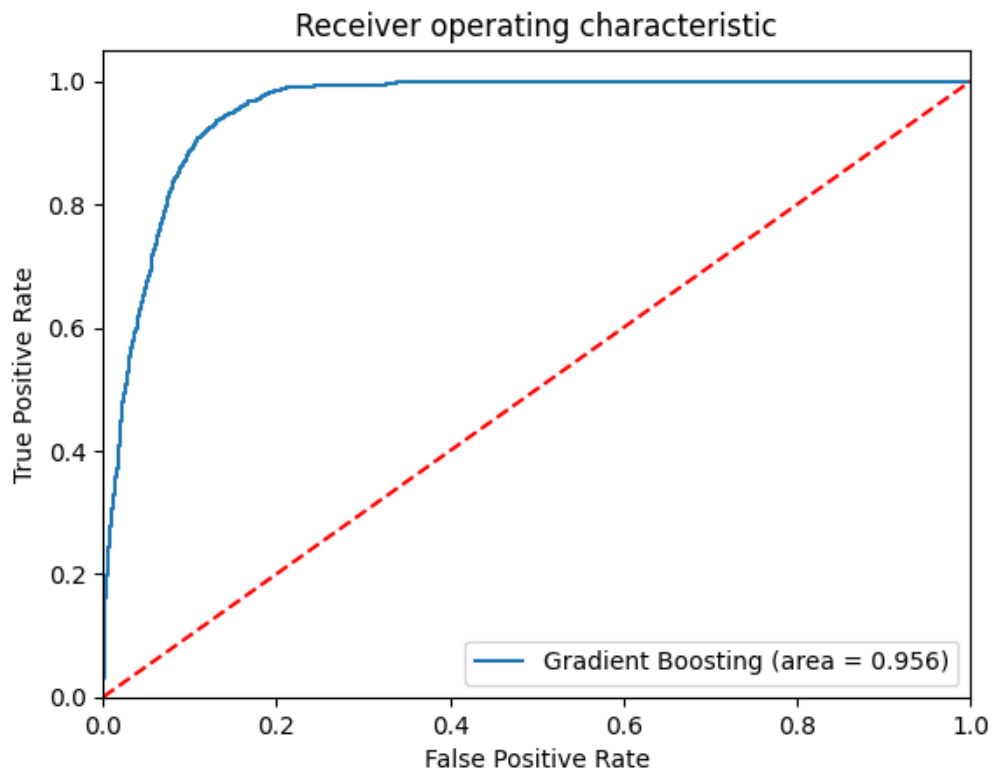


```

probs = gb.predict_proba(X_test).to_numpy()
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
roc_auc = metrics.auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='Gradient Boosting (area = %0.3f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
SAS.pyplot(plt)

```



Confusion Matrix

The confusion matrix plots what the model predicts against what the correct prediction is. You can use this to understand the trade-off between false positives (top right) and false negatives (bottom left). To plot the confusion matrix, use the following code:

```

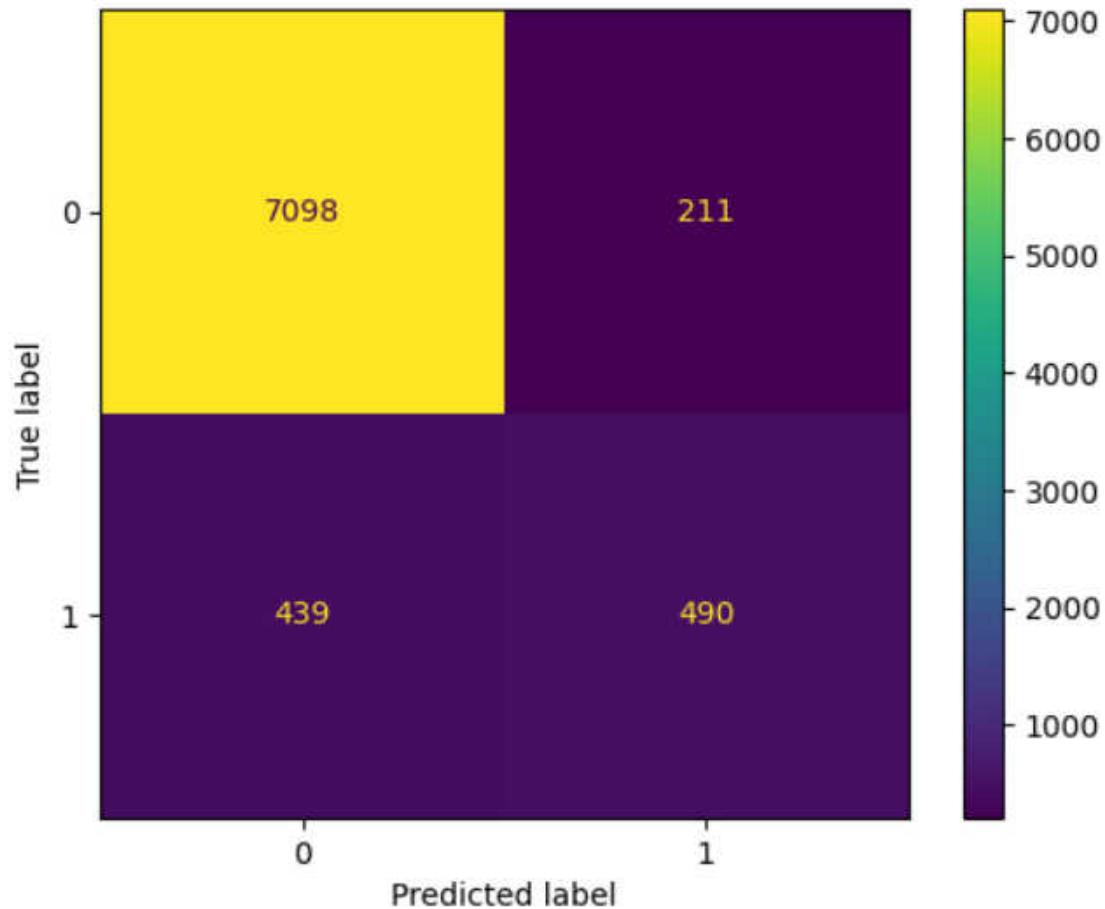
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate test set predictions with the model
y_pred = gb.predict(X_test).astype(type(y_test[0]))

```

```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot()
SAS.pyplot(plt)
```



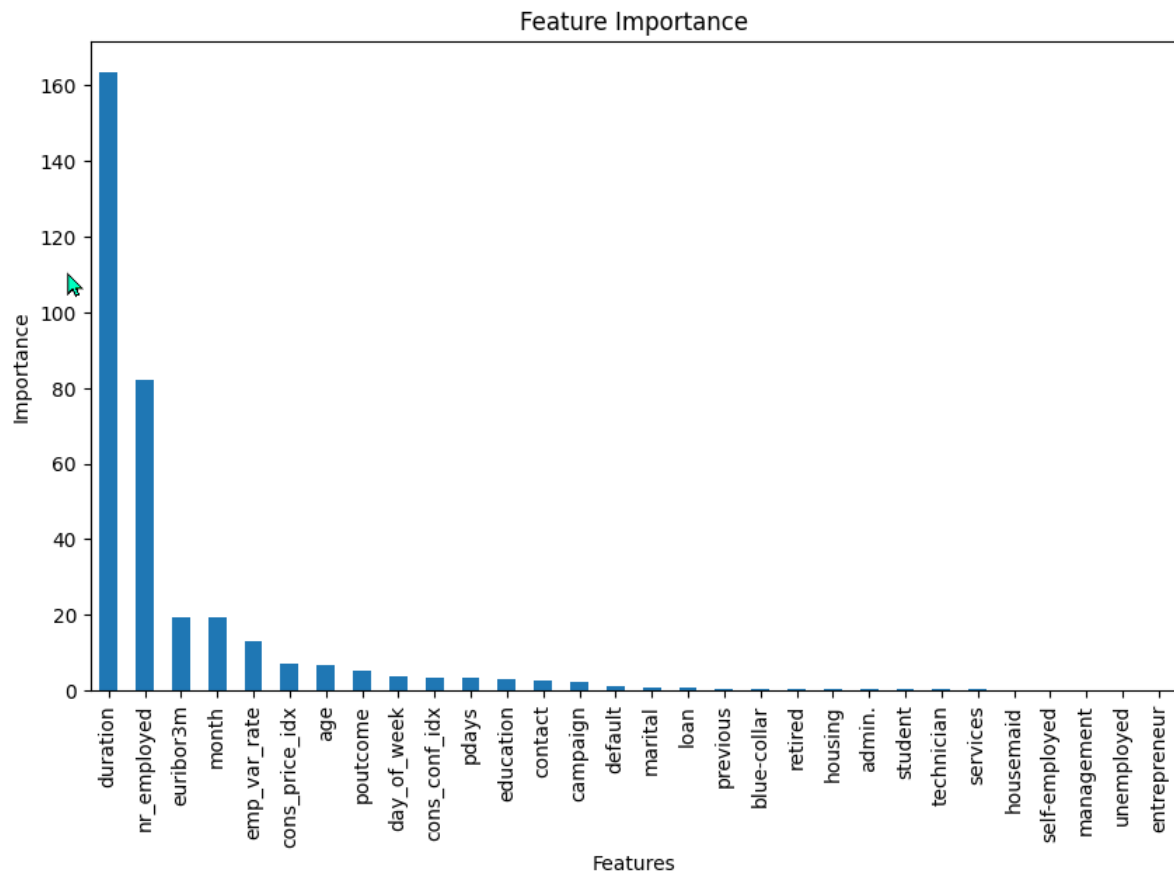
Plot Feature Importances

```
feature_importances = gb.feature_importances_.set_index('Variable')
['Importance']

# Set the figure size
plt.figure(figsize=(10, 6)) # Adjust the width (10) and height (6) as
desired

# Plot a simple bar chart
feature_importances.plot.bar();

# Add labels and title
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance')
SAS.pyplot(plt)
```



GradientBoostingRegressor

Enables you to fit a gradient boosting model to your data and then predict and score new observations.

GradientBoostingRegressor from SAS® Viya® on Bike Sharing Demand

Source

This example has been adapted from [EDA & Ensemble Model \(Top 10 Percentile\)](#) by Vivek Srinivasan.

Data Preparation

About the data set

A bicycle sharing system is a system of linked rental locations throughout a metropolitan area that allows individuals to share a pool of bicycles for their use. Bicycles rented from one location can be returned to any other location in the network. Joining the network and renting or returning bicycles are automated. As of 2022, there are around 3,000 cities offering such systems.

The data set contains information about the number of bikes rented by different types of users over several years tracked by hour. Additional information about each rental period includes weather information and categorization of the type of day.

```
import os
import pandas as pd
import numpy as np
import warnings
pd.options.mode.chained_assignment = None
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

Importing the data

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
print(bikeData.shape)
```

```
(17379, 16)
```

Sample of the first few rows

```
print(bikeData.head(5))
```

	date	season	year	month	...	windspeed	casual	registered	count
0	1/1/2011	1	0	1	...	0.0	3	13	16
1	1/1/2011	1	0	1	...	0.0	8	32	40
2	1/1/2011	1	0	1	...	0.0	5	27	32
3	1/1/2011	1	0	1	...	0.0	3	10	13
4	1/1/2011	1	0	1	...	0.0	0	1	1

[5 rows x 16 columns]

Variable data types

```
print(bikeData.dtypes)
```

```
[5 rows x 16 columns]
date          object
season        int64
year          int64
month         int64
hour          int64
holiday       int64
weekday       int64
workingday    int64
weather       int64
temp         float64
atemp        float64
humidity      float64
windspeed     float64
casual        int64
registered    int64
count         int64
dtype: object
```

Data Analysis

As you can see from the above results, the columns `hour`, `weekday`, `month`, `season`, `holiday`, `workingday`, and `weather` should be of categorical data type. But the current data type is `int` for those columns. You must transform the data set by performing the following steps:

- Coerce the data type of `hour`, `weekday`, `month`, `season`, `holiday`, `workingday`, and `weather` to category.
- Drop the `date` column.

Coercing to categorical types and dropping unnecessary columns

```
bikeData["month"] =
bikeData.month.map({1:"January",2:"February",3:"March",4:"April",5:"May",
6:"June",7:"July",
8:"August",9:"September",10:"October",11:"November",12:"December"})
bikeData["weekday"] =
bikeData.weekday.map({0:"Sunday",1:"Monday",2:"Tuesday",3:"Wednesday",4:
:"Thursday",5:"Friday",6:"Saturday"})
bikeData["season"] = bikeData.season.map({1: "Spring", 2 : "Summer",
3 : "Fall", 4 : "Winter" })
bikeData["weather"] = bikeData.weather.map({1: " Clear + Few clouds +
Partly cloudy + Partly cloudy",\
2 : " Mist + Cloudy, Mist +
Broken clouds, Mist + Few clouds, Mist ", \
3 : " Light Snow, Light Rain +
Thunderstorm + Scattered clouds, Light Rain + Scattered clouds", \
4 : " Heavy Rain + Ice Pallets
+ Thunderstorm + Mist, Snow + Fog " })
categoryVariableList =
["hour","weekday","month","season","holiday","workingday","weather"]
```

```
for var in categoryVariableList:
    bikeData[var] = bikeData[var].astype("category")
```

Outlier Analysis

Visualizing the outliers

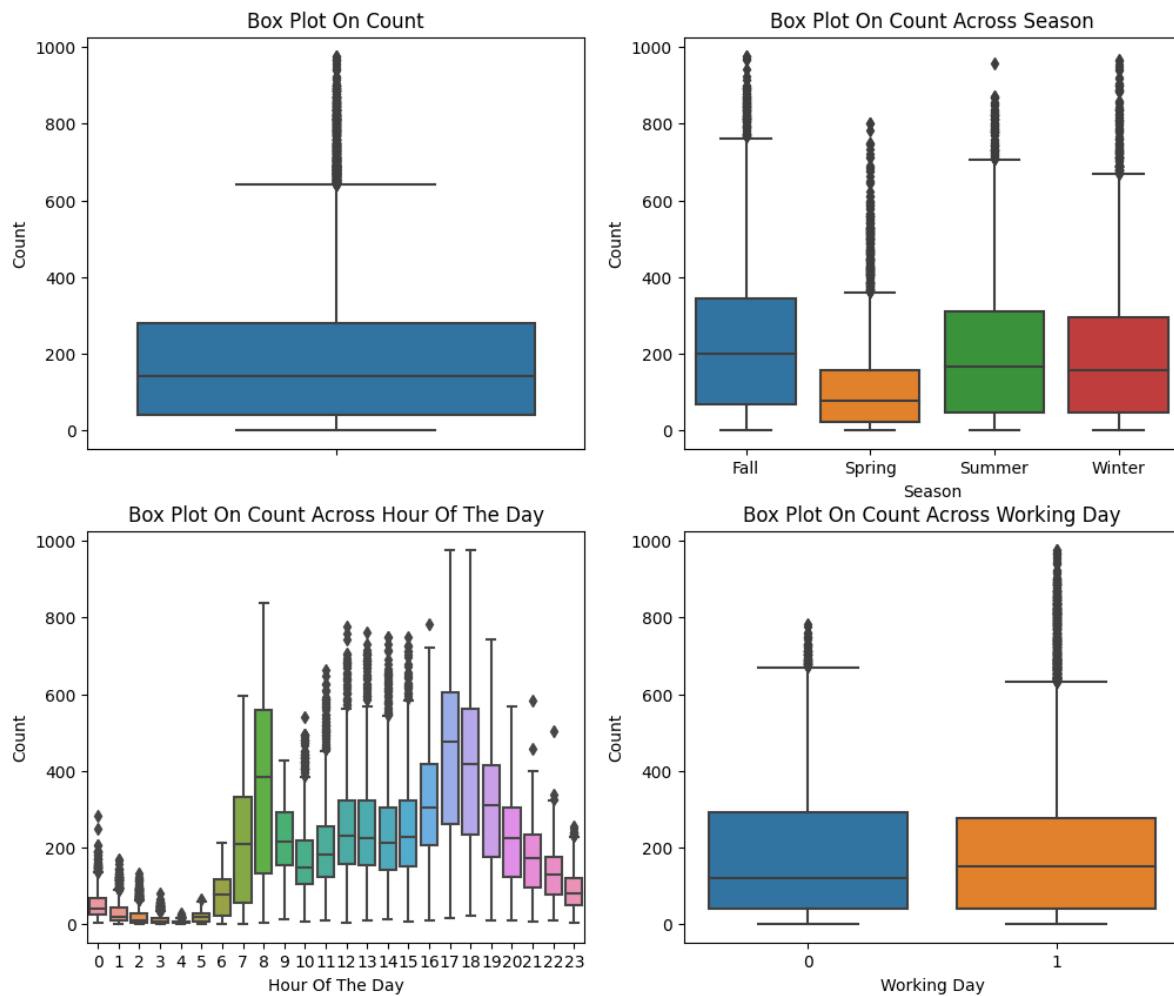
In order to get a sense of how outliers might be distributed in the data, you will run a series of box plots for `count` by itself and `count` versus `season`, `hour`, and `workingday`.

The plots enable you to draw the following inferences about the data:

- `count` contains many outliers beyond the upper quartile limit which skew the distribution toward higher values.
- For `season`, Spring has a relatively lower of `count` demonstrated by the lower median value in its box plot.
- The box plot for "Hour Of The Day" has relatively higher median values for 0700, 0800, 1700, and 1800 hours. This likely matches up with commuter traffic for the start and end of school and office workdays.
- "Working Day" observations contribute a majority of the outliers, as opposed to "Non-Working Day" ones.

```
import matplotlib.pyplot as plt
import seaborn as sn
fig, axes = plt.subplots(nrows=2,ncols=2)
fig.set_size_inches(12, 10)
sn.boxplot(data=bikeData,y="count",orient="v",ax=axes[0][0])
sn.boxplot(data=bikeData,y="count",x="season",hue="season",orient="v",ax=axes[0][1])
sn.boxplot(data=bikeData,y="count",x="hour",hue="hour",orient="v",ax=axes[1][0])
sn.boxplot(data=bikeData,y="count",x="workingday",hue="workingday",orient="v",ax=axes[1][1])

axes[0][0].set(ylabel='Count',title="Box Plot On Count")
axes[0][1].set(xlabel='Season', ylabel='Count',title="Box Plot On Count Across Season")
axes[1][0].set(xlabel='Hour Of The Day', ylabel='Count',title="Box Plot On Count Across Hour Of The Day")
axes[1][1].set(xlabel='Working Day', ylabel='Count',title="Box Plot On Count Across Working Day")
SAS.pyplot(plt)
```



Removing outliers in the count variable

```
bikeDataWithoutOutliers = bikeData[np.abs(bikeData["count"] -
bikeData["count"].mean()) <= (3*bikeData["count"].std())]

print ("Shape Of The Before Ouliers: ",bikeData.shape)
print ("Shape Of The After Ouliers: ",bikeDataWithoutOutliers.shape)
```

```
Shape of the data with outliers: (17379, 15)
Shape of the data without outliers: (17135, 15)
```

Correlation Analysis

One way to understand how a dependent variable is influenced by features (numerical) is to find a correlation between the dependent variable and the features. To do this, you can plot a correlation matrix that plots the relationship between the dependent variable count and the features temp, atemp, humidity and windspeed.

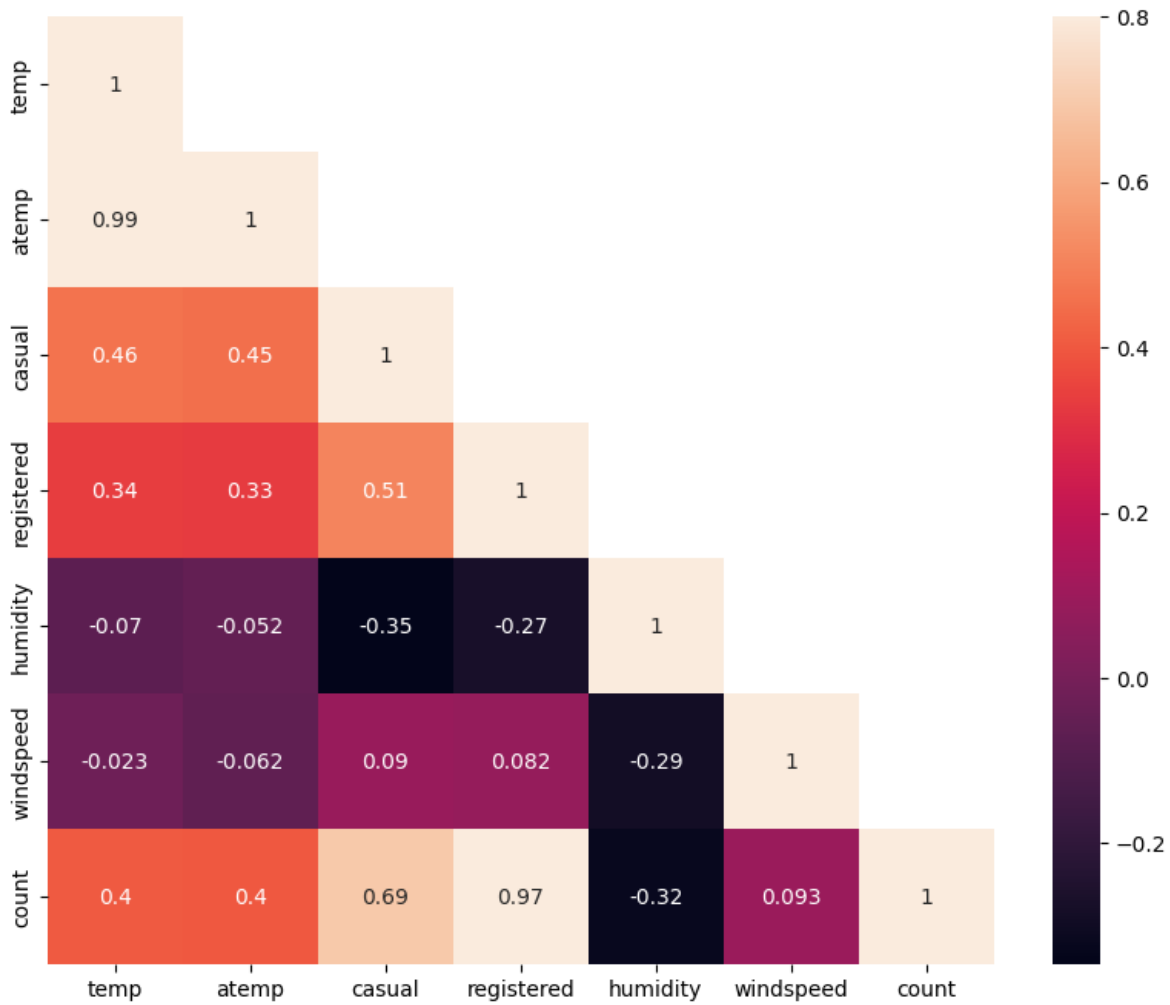
Viewing a heat map

Here are a few key points to note about the correlation matrix:

- The `temp` feature has a positive correlation with `count`, and `humidity` has a negative correlation with `count`. Although these correlations are not very prominent, `count` has a low dependency on `temp` and `humidity`.
- The `windspeed` feature is not a particularly useful numerical feature. It has a correlation value with `count`.
- The `atemp` feature is not relevant because it has a strong correlation with `temp`. During model building one of these variables must be dropped because they will otherwise exhibit multicollinearity in the data.
- The `casual` and `registered` features are not appropriate for this model since they introduce leakage variables in nature and need to be dropped during model building.

Note that the *seaborn* library also provides a useful way to depict the relationship between two features. Consider `count` as compared with `temp`, `humidity`, and `windspeed`.

```
corrMatt =
bikeData[["temp", "atemp", "casual", "registered", "humidity", "windspeed", "
count"]].corr()
mask = np.array(corrMatt)
mask[np.tril_indices_from(mask)] = False
fig, ax = plt.subplots()
fig.set_size_inches(10, 8)
sn.heatmap(corrMatt, mask=mask, vmax=.8, square=True, annot=True)
SAS.pyplot(plt)
```

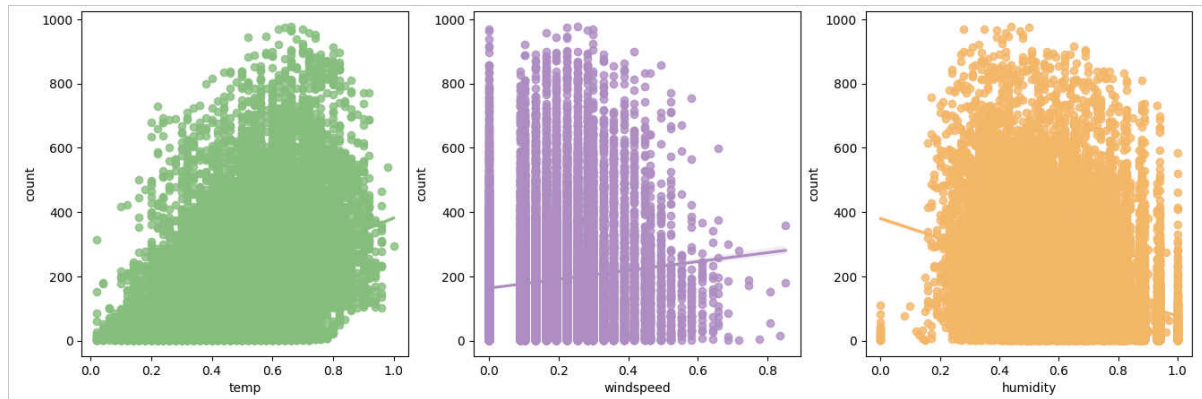
Regression plots

Regression plots from seaborn are a useful way to depict the relationship between two features. You will also view these for count versus temp, atemp, humidity, and windspeed.

```
fig, (ax1, ax2, ax3) = plt.subplots(ncols=3)
fig.set_size_inches(12, 5)

# Assigning softer colors to each plot
color1 = "#7fbf7b" # Light green
color2 = "#af8dc3" # Light purple
color3 = "#fdb462" # Light orange

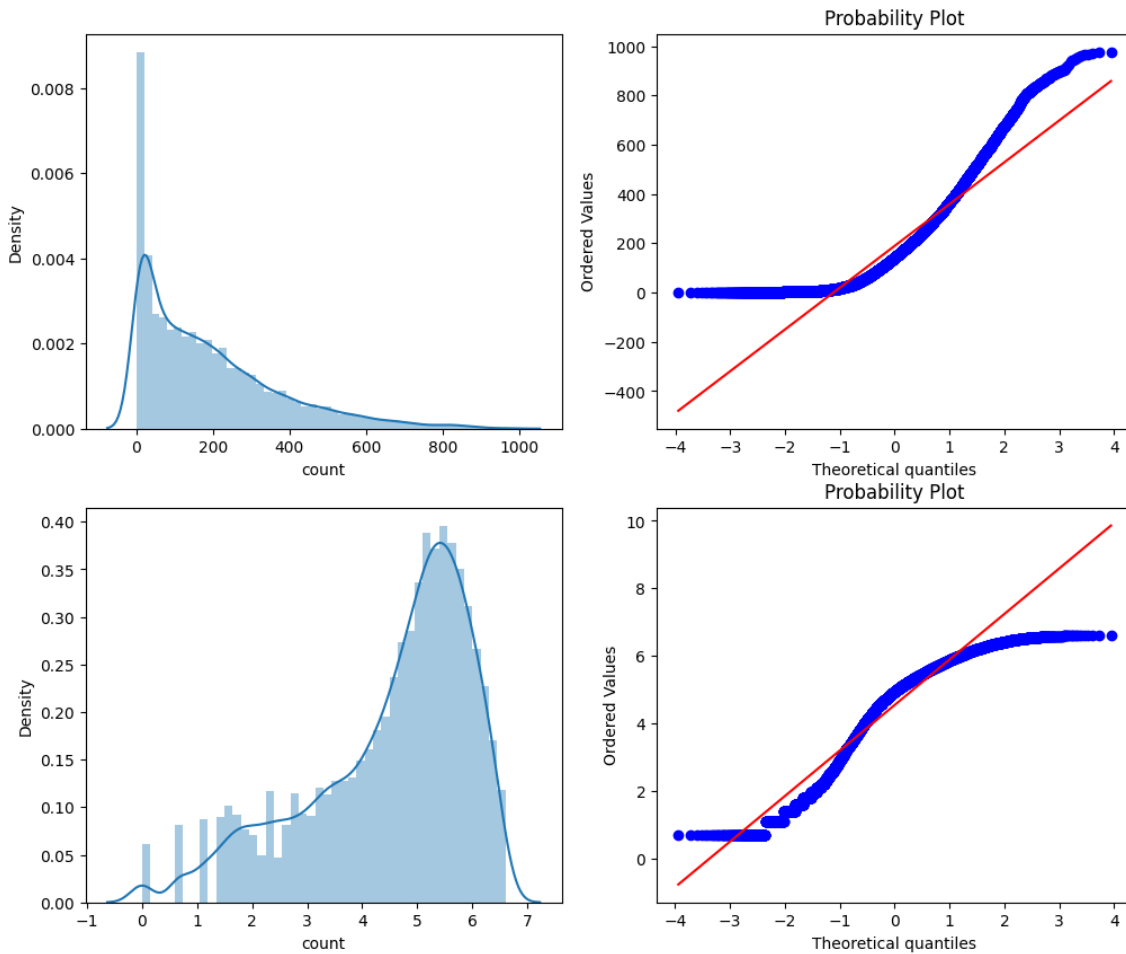
sn.regplot(x="temp", y="count", data=bikeData, ax=ax1, color=color1)
sn.regplot(x="windspeed", y="count", data=bikeData, ax=ax2,
color=color2)
sn.regplot(x="humidity", y="count", data=bikeData, ax=ax3, color=color3)
SAS.pyplot(plt)
```



Visualizing the Distribution of Data

As is visible from the figures below, `count` is skewed toward the right and not a normal distribution. Since many machine learning algorithms require a normal distribution, you will need to pursue options to address this. A common technique is to take a log transformation of `count` after removing the outliers. Afterward, the data distribution looks better but is still not an ideal normal.

```
from scipy import stats
fig, axes = plt.subplots(ncols=2, nrows=2)
fig.set_size_inches(12, 10)
sn.distplot(bikeData["count"], ax=axes[0][0])
stats.probplot(bikeData["count"], dist='norm', fit=True, plot=axes[0][1])
sn.distplot(np.log(bikeDataWithoutOutliers["count"]), ax=axes[1][0])
stats.probplot(np.log1p(bikeDataWithoutOutliers["count"]),
               dist='norm', fit=True, plot=axes[1][1])
fig.show()
SAS.pyplot(plt)
```



Visualizing Count versus Month, Season, Hour, and Weekday

Here are some key ideas to note from this visualization:

- People tend to rent bikes during the summer season. Therefore, June, July, and August have relatively higher demands for bicycles.
- On weekdays, more people tend to rent bicycles between 7AM to 8AM and 5PM to 6PM. As mentioned earlier, this can be attributed to regular school and office commuters.
- This weekday pattern is not observed on the weekends ("Saturday" and "Sunday"). In other words, on the weekends, more people tend to rent bicycles between 10AM and 4PM.
- The peak user count, which is between 7AM to 8AM and 5PM to 6PM, consists of registered users.

```
fig, (ax1, ax2, ax3, ax4) = plt.subplots(nrows=4)
fig.set_size_inches(12, 20)
```

```

sortOrder =
["January", "February", "March", "April", "May", "June", "July", "August", "Sep-
tember", "October", "November", "December"]
hueOrder =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"
]

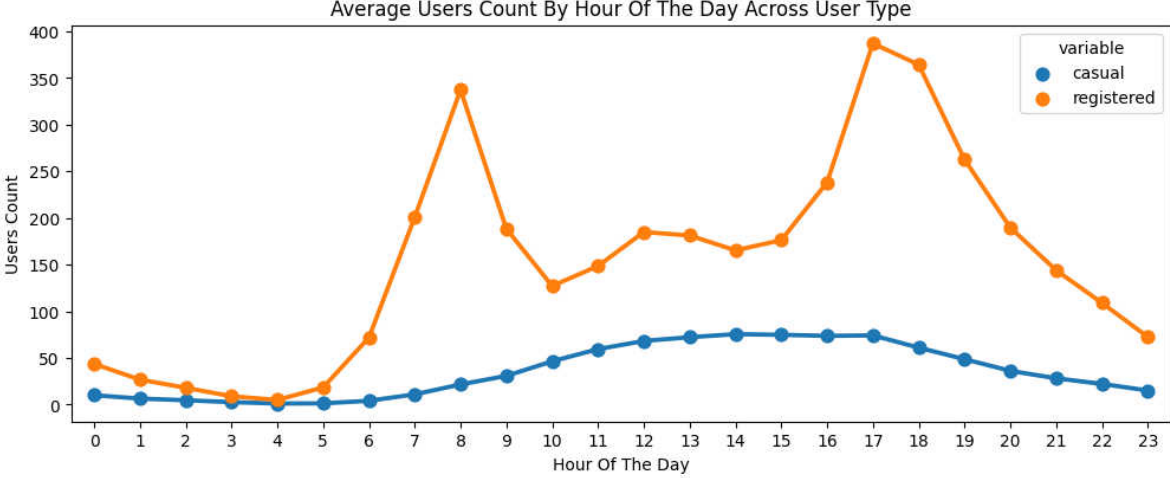
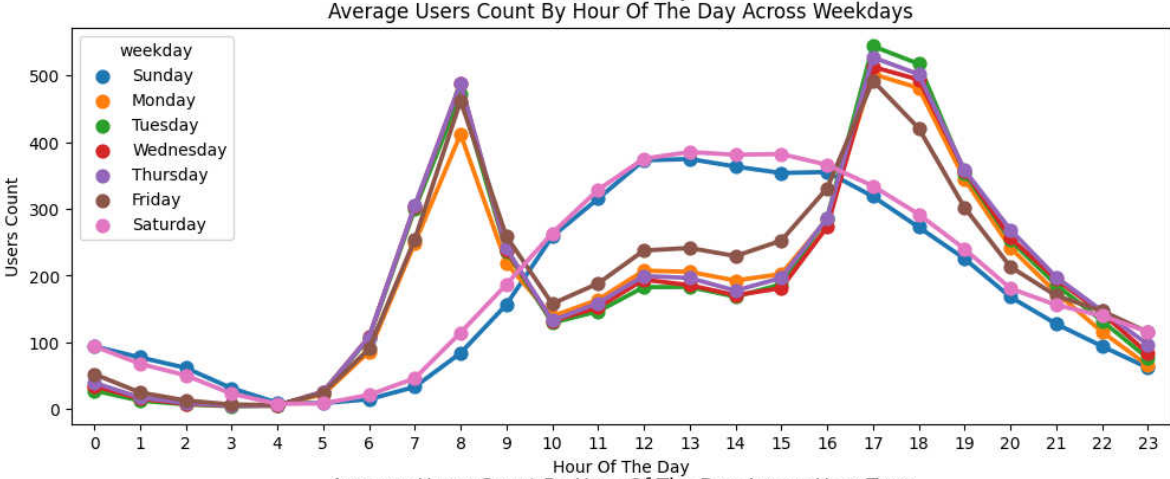
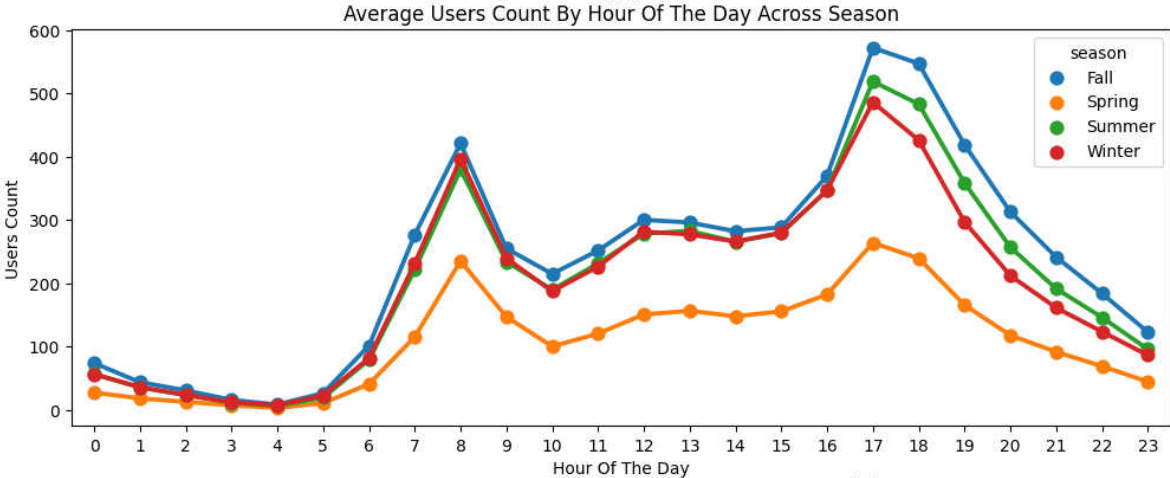
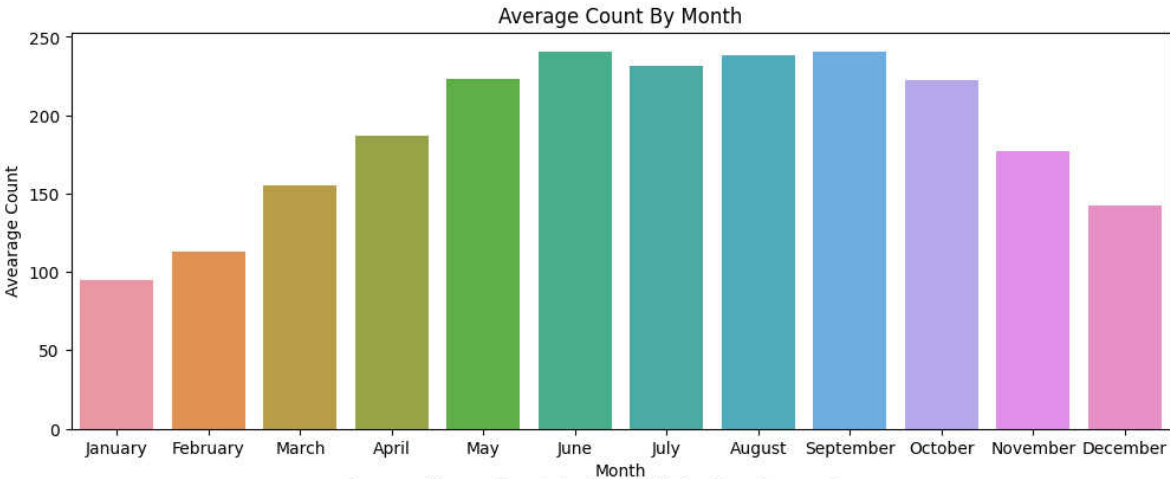
monthAggregated = pd.DataFrame(bikeData.groupby("month")
["count"].mean()).reset_index()
monthSorted = monthAggregated.sort_values(by="count", ascending=False)
sn.barplot(data=monthSorted, x="month", y="count", hue="month", ax=ax1, orde
r=sortOrder)
ax1.set(xlabel='Month', ylabel='Average Count', title="Average Count By
Month")

hourAggregated =
pd.DataFrame(bikeData.groupby(["hour", "season"], sort=True)
["count"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["count"], hue=hourAggregated["season"],
data=hourAggregated, join=True, ax=ax2)
ax2.set(xlabel='Hour Of The Day', ylabel='Users Count', title="Average
Users Count By Hour Of The Day Across Season", label='big')

hourAggregated =
pd.DataFrame(bikeData.groupby(["hour", "weekday"], sort=True)
["count"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["count"], hue=hourAggregated["weekday"], hue_order=hueOr
der, data=hourAggregated, join=True, ax=ax3)
ax3.set(xlabel='Hour Of The Day', ylabel='Users Count', title="Average
Users Count By Hour Of The Day Across Weekdays", label='big')

hourTransformed = pd.melt(bikeData[["hour", "casual", "registered"]],
id_vars=['hour'], value_vars=['casual', 'registered'])
hourAggregated =
pd.DataFrame(hourTransformed.groupby(["hour", "variable"], sort=True)
["value"].mean()).reset_index()
sn.pointplot(x=hourAggregated["hour"],
y=hourAggregated["value"], hue=hourAggregated["variable"], hue_order=["ca
sual", "registered"], data=hourAggregated, join=True, ax=ax4)
ax4.set(xlabel='Hour Of The Day', ylabel='Users Count', title="Average
Users Count By Hour Of The Day Across User Type", label='big')
fig.show()
SAS.pyplot(plt)

```



Integrating Data Discoveries

Now that you have visualized the data, you can build models to predict results.

Load the data

```
bikeData = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/
bike_sharing_demand.csv')
```

Defining feature lists

```
categoricalFeatureNames =
["season", "holiday", "workingday", "weather", "weekday", "month", "year", "hour"]
numericalFeatureNames = ["temp", "humidity", "windspeed", "atemp"]
dropFeatures = ["casual", "date", "registered"]
```

Dropping unnecessary variables

```
bikeData = bikeData.drop(dropFeatures,axis=1)
```

Creating training and test data

Split the original data by putting 75% into the training set and 25% into the test set.

```
X = bikeData.drop('count', axis=1)
Y = bikeData['count']

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X,Y,test_size=0.25,random_state=3)
```

Ensemble Models - Gradient Boosting

For details about using the `GradientBoostingRegressor` class of the `sasviya` package, see the [GradientBoostingRegressor documentation](#).

```
from sasviya.ml.tree import GradientBoostingRegressor

gbModel = GradientBoostingRegressor(n_estimators=100,
                                     max_depth=10,
                                     min_samples_leaf=1,
                                     max_features=None,
                                     learning_rate=0.1,
                                     subsample = 1.0,
                                     random_state=0)

gbModel.fit(X_train, y_train)
```

```
print("Gradient Boosting Regressor -Test Set Score:
{:.2f}".format(gbModel.score(X_test, y_test)))
```

```
Gradient Boosting Regressor -Test Set Score: 0.95
```

LogisticRegression

Enables you to fit a logistic regression model to your data and then predict new class labels and probabilities.

LogisticRegression from SAS Viya on Heart Disease

Source

This example is adapted from [Heart Disease UCI](#) by Harshal Gadhe.

Data Preparation

About the data set

The original data contains 76 different attributes of patients from four different hospital databases. The goal is to determine if the attributes can be used to predict whether patients are diagnosed with heart disease. However, this data has been subset to contain only 14 factors from only the Cleveland database. Here are the variables included and their interpretations:

- age
- trestbps: resting blood pressure
- chol: serum cholesterol
- thalch: maximum heart reate achieved
- ca: number of major vessels (0-3) colored by fluoroscopy
- sex
- cp: chest pain type
- exang: exercise-induced angina
- slope: slope of the peak exercise ST segment

- thal: thalassemia (3 = normal; 6=fixed defect; 7 = reversable defect)
- restecg: resting electrocardiographic results
- fbs: fasting blood sugar
- target: diagnosis of heart disease
- oldpeak: ST depression induced by exercise relative to rest

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn

import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)
warnings.filterwarnings("ignore", category=UserWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.options.mode.chained_assignment = None
```

Importing the data set

```
heart_df = pd.read_csv("https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/heart_disease.csv")
print(heart_df.head())
```

	age	trestbps	chol	thalch	oldpeak	...	slope	thal	restecg	fbs	target
0	63	145	233	150	2.3	...	0	1	0	1	0
1	67	160	286	108	1.5	...	1	2	0	0	1
2	67	120	229	129	2.6	...	1	0	0	0	1
3	37	130	250	187	3.5	...	0	2	1	0	0
4	41	130	204	172	1.4	...	2	2	0	0	0

Data Preprocessing

Print information about the data

Start by getting some general characteristics about the data set.

```
print(heart_df.info())
```



```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 304 entries, 0 to 303
Data columns (total 14 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   age         304 non-null    int64
 1   trestbps    304 non-null    int64
 2   chol        304 non-null    int64
 3   thalch      304 non-null    int64
 4   oldpeak     304 non-null    float64
 5   ca          299 non-null    float64
 6   sex         304 non-null    int64
 7   cp          304 non-null    int64
 8   exang       304 non-null    int64
 9   slope       304 non-null    int64
10   thal        304 non-null    int64
11   restecg     304 non-null    int64
12   fbs         304 non-null    int64
13   target      304 non-null    int64
dtypes: float64(2), int64(12)
memory usage: 33.3 KB

```

Replacing NaN values with mean

There are some missing values in the data set. Replace the missing values with the mean for the column.

```

cols_sum_null = heart_df.isnull().sum()
print(cols_sum_null)

```

```

age         0
trestbps    0
chol        0
thalch      0
oldpeak     0
ca          5
sex         0
cp          0
exang       0
slope       0
thal        0
restecg     0
fbs         0
target      0
dtype: int64

```

```

hasnull_cols = cols_sum_null[cols_sum_null != 0]
for col in hasnull_cols.index:
    mean = heart_df[col].mean()
    heart_df[col].fillna(mean, inplace=True)
print(heart_df.isnull().sum())

```

```

age      0
trestbps 0
chol     0
thalch   0
oldpeak  0
ca       0
sex      0
cp       0
exang    0
slope    0
thal     0
restecg  0
fbs      0
target   0
dtype: int64

```

Correlations

```
print(heart_df.corr())
```

```

          age  trestbps    chol  ...  restecg    fbs    target
age      1.000000  0.281916  0.225333  ... -0.102142  0.120826  0.228467
trestbps 0.281916  1.000000  0.129773  ... -0.119188  0.175419  0.150904
chol     0.225333  0.129773  1.000000  ... -0.142105  0.012782  0.091021
thalch   -0.401508 -0.045660 -0.014568  ...  0.034189 -0.009941 -0.419619
oldpeak  0.209235  0.189202  0.052615  ... -0.050019  0.006968  0.426069
ca       0.354519  0.098706  0.117576  ... -0.107101  0.143925  0.456962
sex      -0.102636 -0.064622 -0.203126  ... -0.057334  0.046870  0.274144
cp       -0.064347  0.041611 -0.075577  ...  0.044318  0.097061 -0.433479
exang    0.096937  0.064930  0.065811  ... -0.063887  0.026593  0.433054
slope   -0.168549 -0.117505 -0.003035  ...  0.083403 -0.061122 -0.341159
thal     -0.105685 -0.123009 -0.011651  ...  0.025765 -0.065833 -0.510246
restecg  -0.102142 -0.119188 -0.142105  ...  1.000000 -0.080011 -0.134547
fbs      0.120826  0.175419  0.012782  ... -0.080011  1.000000  0.026484
target   0.228467  0.150904  0.091021  ... -0.134547  0.026484  1.000000
[14 rows x 14 columns]

```

Visualizing the Data

Look at a variety of plots to get a sense of the data set.

- a pair plot of each pair of variables
- a heat map of the correlations
- scatterplots of the target versus each factor
- a histogram of the target's values

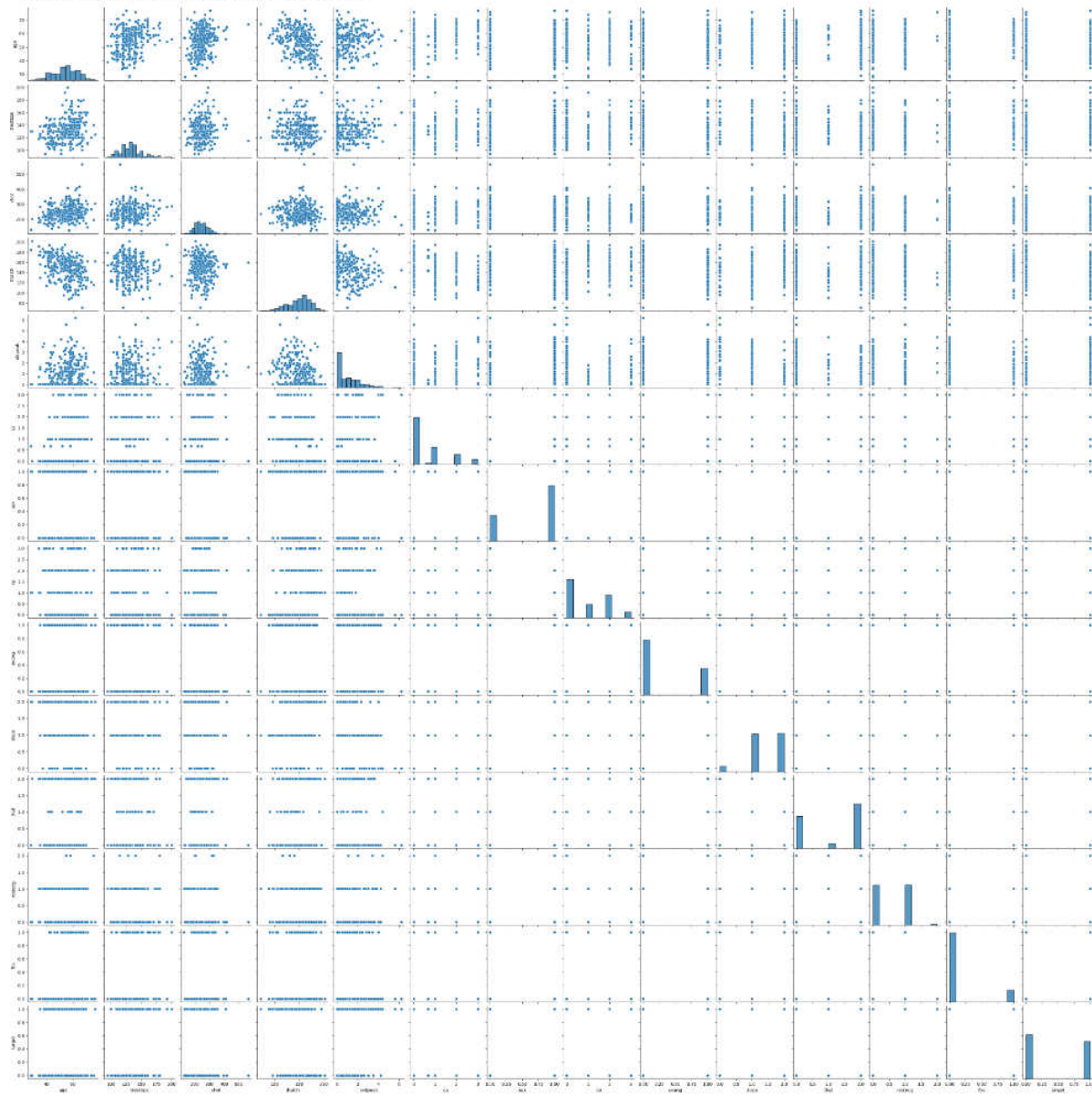
```

cat_threshold = 8
quantitative = [c for c in heart_df.columns if
len(heart_df[c].unique()) > cat_threshold]

sn.pairplot(heart_df[quantitative])
SAS.pyplot(plt)

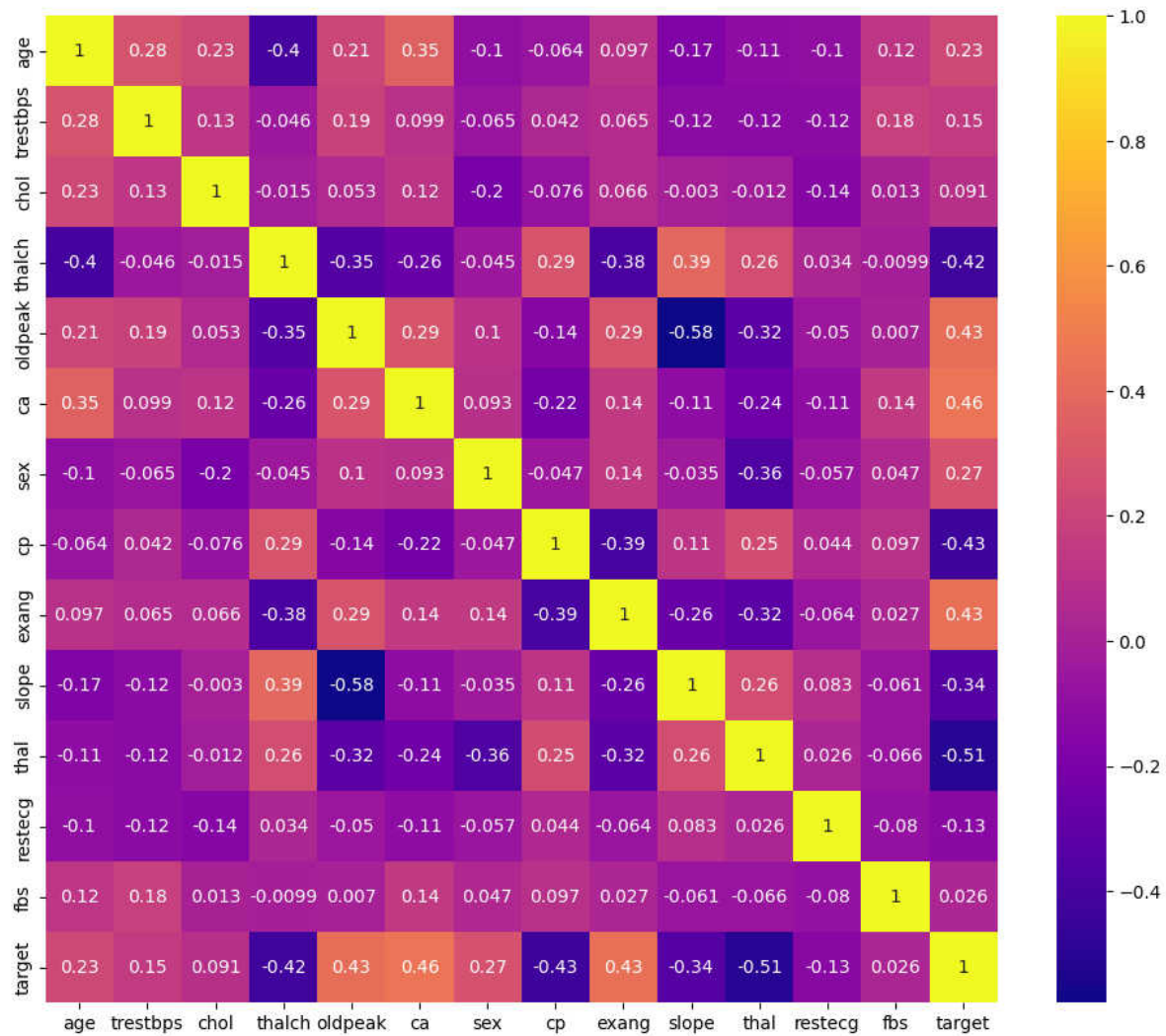
```

```
<seaborn.axisgrid.PairGrid at 0x7f3da6dbbb10>
```



```
plt.figure(figsize=(12,10))
sn.heatmap(heart_df.corr(),annot=True,cmap=plt.cm.plasma)
SAS.pyplot(plt)
```

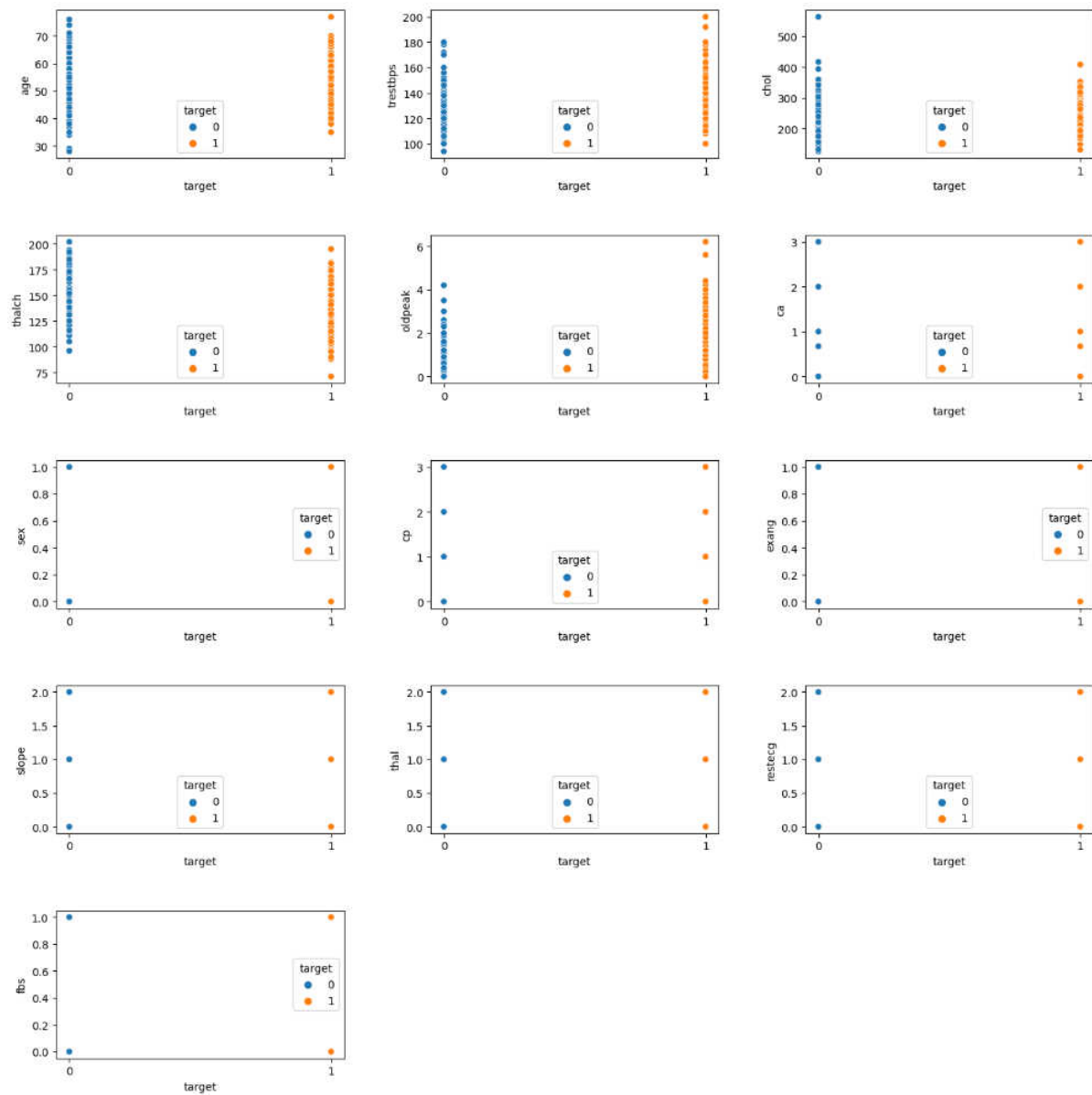
<Axes: >



```

plt.figure(figsize=(15,15))
for i in range(len(heart_df.columns)-1):
    plt.subplot(5, 3, i+1)
    sns.scatterplot(data=heart_df, x='target', y=heart_df.columns[i],
                    hue='target')
    plt.xticks([0, 1])
plt.tight_layout(pad=4.0)
plt.show()
SAS.pyplot(plt)

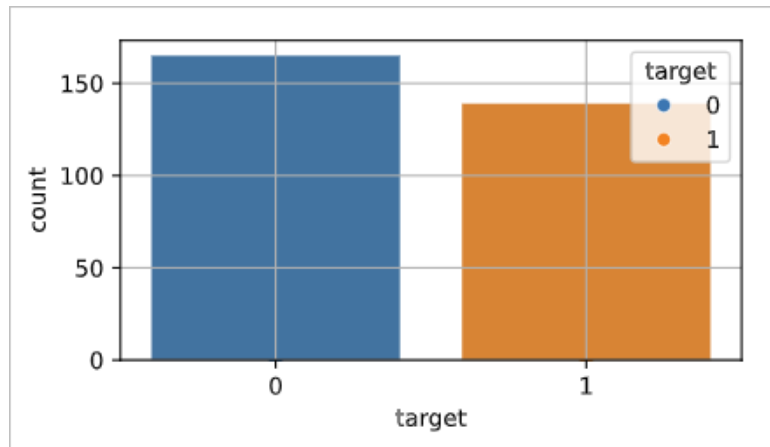
```



```

sn.countplot(x='target', hue='target', legend=False, data=heart_df)
plt.grid()
plt.ylabel("count")
SAS.pyplot(plt)

```



Building and Training the Model

For details about using the classes in `sasviya.ml`, see the [Python API documentation](#).

Data preprocessing

```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
X=heart_df.drop('target',axis=1)
Y=heart_df['target']
heart_df=sc.fit(X).transform(X)
```

Creating training and test data

Split the original data by putting 75% of the data in the training set and 25% of the data into the test set.

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=3)
```

Training the model for six different algorithms

You will train six different models against the training set: `LogisticRegression`, `DecisionTreeClassifier`, `SVC`, `ForestClassifier`, `KNeighborsClassifier`, and `GaussianNB`. The latter two are sklearn classifiers.

```
from sasviya.ml.linear_model import LogisticRegression
from sasviya.ml.tree import DecisionTreeClassifier
from sasviya.ml.svm import SVC
from sasviya.ml.tree import ForestClassifier

from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

def model(X_train,y_train):
    models=[]
```

```

lr = LogisticRegression(
    solver='lbfgs',
    tol=1e-4,
    max_iter=1000)
lr.fit(X_train,y_train)
models.append(lr)

tree=DecisionTreeClassifier()
tree.fit(X_train,y_train)
models.append(tree)

svm=SVC(kernel='rbf', coef0=0.1, C=1.0)

svm.fit(X_train,y_train)
models.append(svm)

knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
models.append(knn)

rfc=ForestClassifier()
rfc.fit(X_train,y_train)
models.append(rfc)

nb=GaussianNB()
nb.fit(X_train,y_train)
models.append(nb)

return models

models=model(X_train,y_train)

```

Gathering the accuracy scores

After running the six models, you should examine how well they did relative to each other. For each, gather the accuracy scores for the training and test data and add them into a summary dataframe.

```

from sklearn.metrics import accuracy_score
train_accuracy=[]
test_accuracy=[]

for m in models:
    train_accuracy.append(round(m.score(X_train, y_train),2))
    test_accuracy.append(round(m.score(X_test, y_test),2))

Accuracy_score=pd.DataFrame({
    'Model': [str(m.__repr__()).split('(')[0] for m in models],
    'Train_Accuracy':train_accuracy,
    'Test_Accuracy':test_accuracy
})

```

Finding the Best Model

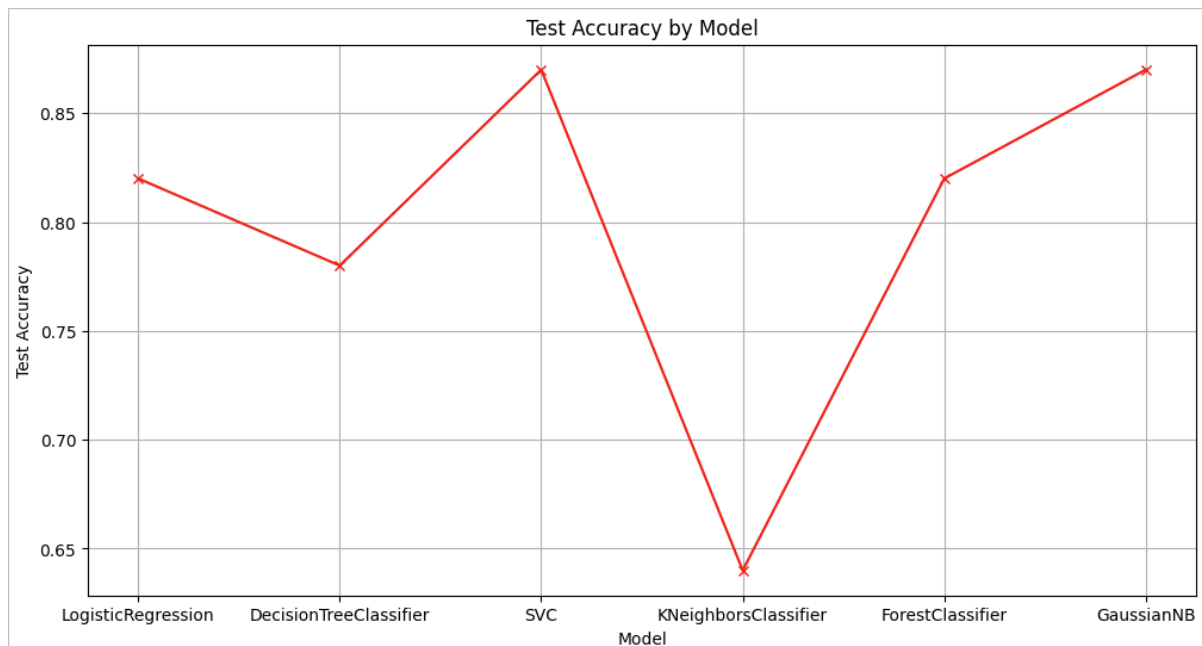
After displaying the table of all accuracy scores for the training and test data, graph the test accuracy for each algorithm to see how they all fared on the data.

```
print(Accuracy_score)
```

```
[14 rows x 4 columns]
```

	Model	Train_Accuracy	Test_Accuracy
0	LogisticRegression	0.88	0.82
1	DecisionTreeClassifier	0.90	0.78
2	SVC	0.85	0.87
3	KNeighborsClassifier	0.75	0.64
4	ForestClassifier	0.92	0.83
5	GaussianNB	0.86	0.87

```
plt.figure(figsize=(12,6))
plt.plot(Accuracy_score['Model'],Accuracy_score['Test_Accuracy'],marker
='x',color='red')
plt.xlabel('Model')
plt.ylabel('Test Accuracy')
plt.title('Test Accuracy by Model')
plt.grid()
SAS.pyplot(plt)
```



PCA

Enables you to examine relationships among several quantitative variables by applying principal component analysis.

PCA from SAS® Viya® on Iris

Source

This example is adapted from [Reduce Data Dimensionality using PCA – Python](#) by user suvratarora06.

Data Preparation

About the data set

The Iris data set comes Ronald Fisher's 1936 paper *The use of multiple measurements in taxonomic problems*. It contains 50 samples with five features: identification of one of three species of irises and the length and width of both sepals and petals. This version of the iris data is available through *scikit-learn*.

```
from sklearn import datasets
import pandas as pd
import seaborn as sn
from sasviya.ml.decomposition import PCA
```

Importing the data set

To load the data, use the `datasets.load_iris()` function from *sklearn*. In this case, you will also convert the data into a Pandas dataframe.

```
iris = datasets.load_iris()
df = pd.DataFrame(iris['data'], columns = iris['feature_names'])
print(df.head(5))
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Standardizing the data

It is commonly recommended to standardize data when using machine learning techniques. The PCA class automatically scales the input features by default, so there is no need for a separate scaling step.

Visualizing the Data

Heat maps provide a convenient way to visualize the correlation between variables. In a default seaborn heat map, darker colors indicate stronger negative correlations and lighter colors show stronger positive correlations. Since petal length has a high correlation with both sepal length and petal width, applying dimensionality reduction could be in order.

```
sn.heatmap(scaled_data.corr(), annot=True)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)
```



Applying Principal Component Analysis

To reduce dimensionality, use principal component analysis (PCA). Because the original data has four columns, you can use `n_components` to reduce it three columns.

For details about using the PCA class of the `sasviya` package, see the [PCA documentation](#).

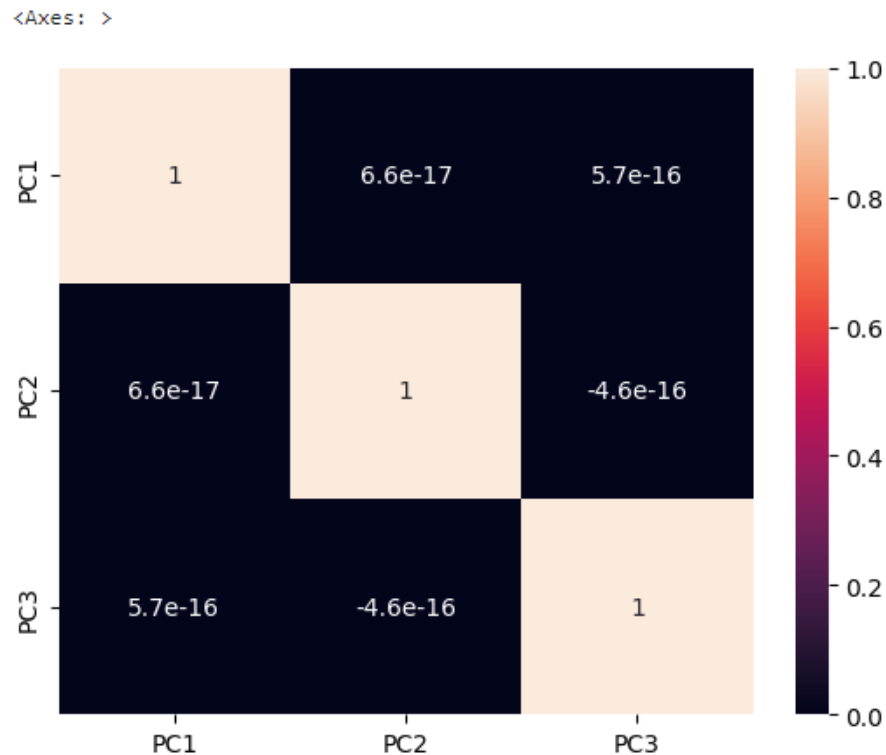
```
pca = PCA(n_components = 3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca, columns=['PC1', 'PC2', 'PC3'])
print(data_pca.head())
```

	PC1	PC2	PC3
0	2.257141	0.478424	0.127280
1	2.074013	-0.671883	0.233826
2	2.356335	-0.340766	-0.044054
3	2.291707	-0.595400	-0.090985
4	2.381863	0.644676	-0.015686

Visualizing Data after PCA

As the brief output shows, PCA transforms the original data. Examine the correlation matrix for the transformed data with a heat map. The principal components have almost no correlation with each other.

```
sn.heatmap(data_pca.corr(), annot=True)
SAS.pyplot(plt, dpi = 96, bbox_inches='tight', pad_inches=0.05)
```



SVC

Enables you to fit a support vector machine model to your data and then predict and score new observations.

svm.SVC from SAS[®] Viya[®] on Breast Cancer with Grid Search

Source

This example is adapted from [SVM Hyperparameter Tuning using GridSearchCV](#) by user tyagikartik4282.

Data Preparation

About the data set

Thirty features describing cell nuclei were calculated from 569 digitized images of fine needle aspirations of a breast mass. Along with an identifier, each image is classified as malignant (M) or benign (B).

In addition to the parameters discovered by training a model, algorithms rely on hyperparameters to control the training process and the resulting parameters. The reasons for choosing particular values might involve a priori knowledge, intuition, or guessing, but can have a significant impact on the quality of the generated model for whatever reason they were selected. A more methodical approach is to use a grid search algorithm that examines the space of parameter combinations to help identify the best performing hyperparameters. This example uses `GridSearchCV` from scikit-learn to show one mechanism for doing this where the grid search is supplemented by cross validation for model evaluation.

```
import pandas as pd
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
from sasviya.ml.svm import SVC
```

Importing the data

```
from sklearn.datasets import load_breast_cancer
dataset = load_breast_cancer()
print(dataset.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

To analyze the data, extract the features and target into dataframes. You can also examine the information about the features and show the general format of the observations. One important thing to note is that there are no missing values for any variables.

```
df_feat = pd.DataFrame(dataset['data'], columns =
dataset['feature_names'])
df_target = pd.DataFrame(dataset['target'], columns = ['Cancer'])
print(df_feat.info())
print(df_feat.head())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 30 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   mean radius                               569 non-null    float64
1   mean texture                              569 non-null    float64
2   mean perimeter                           569 non-null    float64
3   mean area                                569 non-null    float64
4   mean smoothness                          569 non-null    float64
5   mean compactness                         569 non-null    float64
6   mean concavity                           569 non-null    float64
7   mean concave points                      569 non-null    float64
8   mean symmetry                            569 non-null    float64
9   mean fractal dimension                   569 non-null    float64
10  radius error                             569 non-null    float64
11  texture error                            569 non-null    float64
12  perimeter error                         569 non-null    float64
13  area error                              569 non-null    float64
14  smoothness error                        569 non-null    float64
15  compactness error                       569 non-null    float64
16  concavity error                         569 non-null    float64
17  concave points error                    569 non-null    float64
18  symmetry error                          569 non-null    float64
19  fractal dimension error                 569 non-null    float64
20  worst radius                            569 non-null    float64
21  worst texture                           569 non-null    float64
22  worst perimeter                         569 non-null    float64
23  worst area                              569 non-null    float64
24  worst smoothness                       569 non-null    float64
25  worst compactness                      569 non-null    float64
26  worst concavity                        569 non-null    float64
27  worst concave points                   569 non-null    float64
28  worst symmetry                         569 non-null    float64
29  worst fractal dimension                 569 non-null    float64
dtypes: float64(30)
memory usage: 133.5 KB

```

```

mean radius  mean texture  ...  worst symmetry  worst fractal dimension
0           17.99         10.38  ...           0.4601              0.11890
1           20.57         17.77  ...           0.2750              0.08902
2           19.69         21.25  ...           0.3613              0.08758
3           11.42         20.38  ...           0.6638              0.17300
4           20.29         14.34  ...           0.2364              0.07678
[5 rows x 30 columns]

```

Creating training and test data

Split the original data by putting 70% into the training set and 30% into the test set. This split could be considered a hyperparameter, because it impacts the data being used for training and testing; however, the split data will not be included in this examination.

```

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_feat,
np.ravel(df_target),

                                                    test_size = 0.30,

random_state = 101)

```

Training a Support Vector Classifier Model Without Tuning

Before trying hypertuning, use all the defaults for a Support Vector Classifier (SVC) model to see what happens.

To see the effects that tuning hyperparameters can have on model performance, first fit the model using the `SVC()` defaults. You can then print the confusion matrix to assess how well the model performed.

```
model = SVC()
model.fit(X_train, y_train)

# results analysis
y_pred = model.predict(X_test)
print("Classification report:\n", classification_report(y_test,
y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification report:
              precision    recall  f1-score   support

     0               1.00      0.92    0.96         66
     1               0.95      1.00    0.98        105

 accuracy               0.97
 macro avg              0.98
weighted avg              0.97
```

```
Confusion matrix:
[[ 61   5]
 [   0 105]]
```

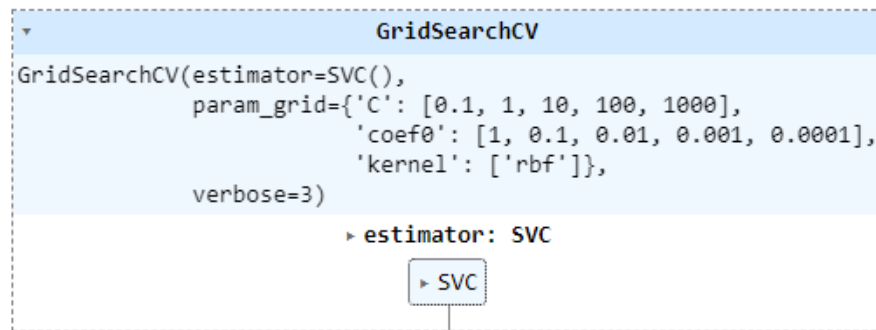
Training a Support Vector Classifier Model With Hyperparameter Tuning

Create a model

Although there are multiple parameters that could be tuned for `SVC()`, examine only two of them: `C` and `coef0` due to the amount of time a full search would take. Since the search space of both is continuous, discretize it into a handful of values of interest. The `fit()` method for a model defined with `GridSearchCV` runs the grid search first, using cross validation to select the best option combination, and then creates a model using those options.

```
from sklearn.model_selection import GridSearchCV
# Define the parameters to examine and tune
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'coef0': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
```

```
grid.fit(X_train, y_train)
```

Viewing the hyperparameter values selected

Afterward, you can list out the selected option combination and generate a model call with the parameters to use directly.

```
print('Best parameters:', grid.best_params_)
print('Model call:', grid.best_estimator_)
```

```
Best parameters: {'C': 100, 'coef0': 1, 'kernel': 'rbf'}
Model call: SVC(C=100, kernel="rbf", coef0=1)
```

Examining the results

Since `grid` is the model with the selected option combination, it can be interacted with like any other model object.

```
grid_pred = grid.predict(X_test)

# print classification report
print("Classification report:\n", classification_report(y_test,
y_pred))
```

```
Classification report:
              precision    recall  f1-score   support

    0               1.00      0.92      0.96         66
    1               0.95      1.00      0.98        105
 accuracy               0.97
 macro avg              0.98      0.96      0.97         171
weighted avg              0.97      0.97      0.97         171
```

SVR

Enables you to fit a support vector machine model to your data and then predict new values.

svm.SVR from SAS® Viya® to Predict Bonuses

Data Preparation

About the data set

This data set contains the bonuses for 10 job positions at a hypothetical company.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import warnings

# Filter out UserWarning messages
warnings.filterwarnings("ignore", category=UserWarning)
# Filter out DeprecationWarning messages
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Importing the data set

Import the data. Load the data into a DataFrame and select the features and target.

```
dataset = pd.read_csv('https://raw.githubusercontent.com/sassoftware/
sas-viya-workbench-examples/refs/heads/main/data/bonus.csv')

X = dataset.iloc[:,1:2].values.astype(float)
y = dataset.iloc[:,2:3].values.astype(float)

print(dataset)
```

	Title	JobCode	Bonus
0	Call Center Agent	1	2000
1	Technical Support Engineer	2	2500
2	Cashier	3	3000
3	Concierge	4	3500
4	Customer Service Representative	5	4000
5	Help Desk Agent	6	4500
6	Public Relation Specialist	7	5000
7	Marketing Consultant	8	5500
8	Accountant	9	6000
9	Sales Representative	10	7000

Building the Support Vector Regression Model

For details about using the `SVR` class of the `sasviya` package, see the [SVR documentation](#).

The data set contains three columns: a string-valued column that identifies an employee's job title, an integer-valued job code ID, and an integer-valued dollar amount that indicates the employee bonus.

Import the `SVR` class so that you can perform regression with a support vector machine. Build the model with a polynomial kernel and fit the model. Use `y.ravel()` to flatten the data into a one-dimensional array.

```
from sasviya.ml.svm import SVR

regressor = SVR(kernel='polynomial')
regressor.fit(X, y.ravel())
```

```
SVR(kernel="poly")
```

You can verify the values used to train the support vector machine by using the `get_params()` method.

```
print(regressor.get_params())
```

```
{'max_iter': 25, 'C': 1.0, 'kernel': 'poly', 'degree': 2, 'tol': None, 'epsilon':
0.01, 'random_state': 1, 'scale': True,
'verbose': 0}
```

Evaluating the Model

Finally, you can make a prediction for the bonus of an employee whose job code ID is halfway between a Help Desk Agent and a Public Relation Specialist.

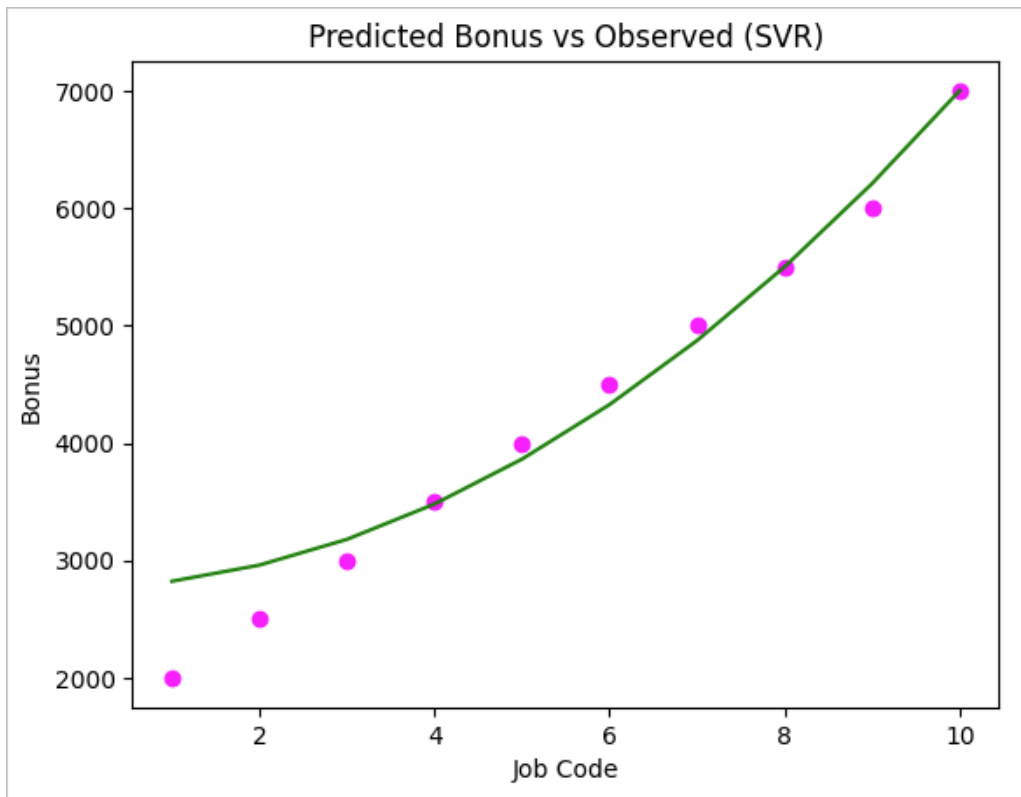
```
y_pred = regressor.predict(np.array([6.5]).reshape(1, -1))
print(y_pred)
```

```
[3059.2063263]
```

The prediction output is about 3059. The model seems to be overpredicting bonuses for lower job classifications.

```
plt.scatter(X, y, color = 'magenta')
plt.plot(X, regressor.predict(X), color = 'green')
plt.title('Truth or Bluff (Support Vector Regression Model)')
plt.xlabel('Job Code')
plt.ylabel('Bonus')
plt.show()
```

SAS.pyplot(plt)



Updating the Model

Processing the data

By default, SVR automatically uses min-max scaling to scale the features to [0,1] range. Use the sklearn `StandardScaler` class to normalize features prior to modeling.

```
from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```

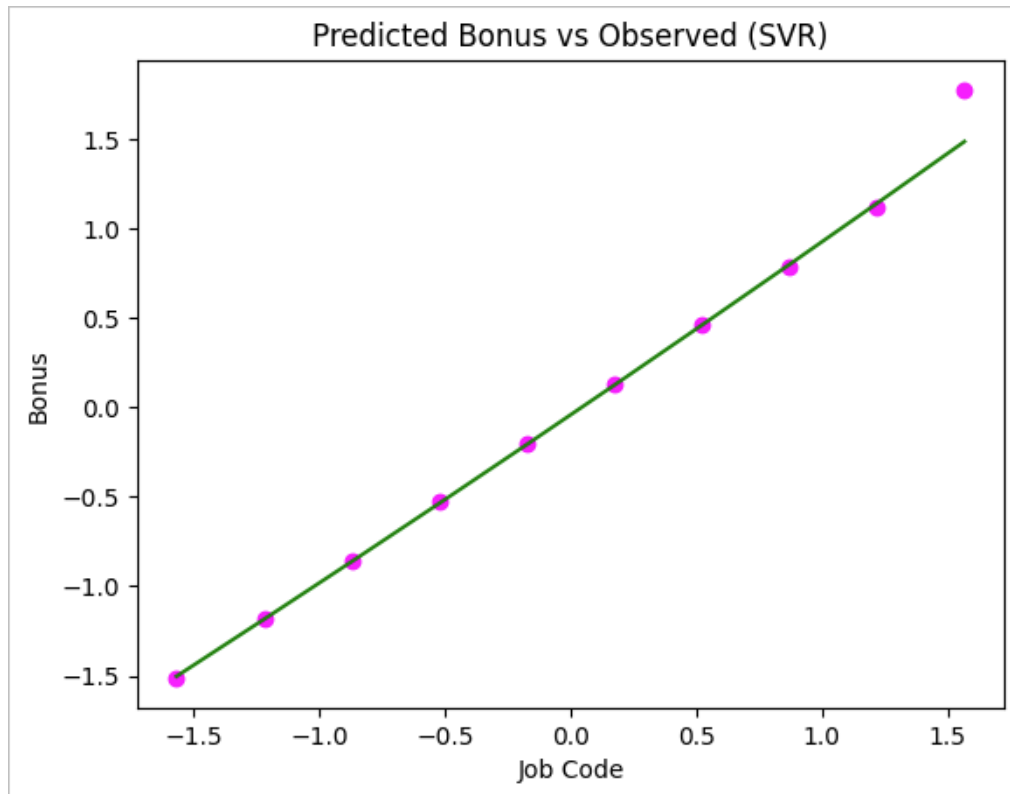
Creating the model with transformed data

```
regressor = SVR(kernel='poly', scale = False)
regressor.fit(X, y.ravel())
```

```
SVR(kernel="poly", scale=False)
```

```
plt.scatter(X, y, color = 'magenta')
plt.plot(X, regressor.predict(X), color = 'green')
```

```
plt.title('Predicted Bonus vs Observed (SVR)')
plt.xlabel('Job Code')
plt.ylabel('Bonus')
plt.show()
SAS.pyplot(plt)
```



```
# Predicting a new result for the input value 2.5
input_value = np.array([[2.5]]) # Reshape the input to a 2D array
with shape (1, 1)
input_value_scaled = sc_X.transform(input_value.reshape(-1, 1))
y_pred_scaled = regressor.predict(input_value_scaled)

# Convert y_pred_scaled from Series to NumPy array and reshape
y_pred_scaled_array = y_pred_scaled.reshape(-1, 1)

# Inverse transform the scaled prediction to the original scale
y_pred = sc_y.inverse_transform(y_pred_scaled_array)
print(y_pred)
```

```
[[2744.29981007]]
```

By using `.transform()` and `.inverse_transform()` methods to convert the scaled values into the original ranges, the new prediction value is about 2744. This value is between the bonuses of Job 2 and Job 3, so it is likely a better fit for the original data.