dvb.datascience Documentation

de Volksbank

Oct 02, 2018

Contents:

CHAPTER 1

dvb.datascience

A python data science pipeline package. At de Volksbank, our data scientists used to write a lot of overhead code for every experiment from scratch. To help them focus on the more exciting and value added parts of their jobs, we created this package. Using this package you can easily create and reuse your pipeline code (consisting of often used data transformations and modeling steps) in experiments. This package has (among others) the following features:

- Make easy-to-follow model pipelines of fits and transforms (what exactly is a pipeline?)
- Make a graph of the pipeline
- Output graphics, data, metadata, etc from the pipeline steps
- Data preprocessing such as filtering feature and observation outliers
- Adding and merging intermediate dataframes
- Every pipe stores all intermediate output, so the output can be inspected later on
- Transforms can store the outputs of previous runs, so the data from different transforms can be compared into one graph
- Data is in Pandas DataFrame format
- Parameters for every pipe can be given with the pipeline fit_transform() and transform() methods

de volksbank

1.1 Scope

This package was developed specifically for fast prototyping with relatively small datasets on a single machine. By allowing the intermediate output of each pipeline step to be stored, this package might underperform for bigger datasets (100,000 rows or more).

1.2 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. For a more extensive overview of all the features, see the docs directory.

1.2.1 Prerequisites

This package requires Python3 and has been tested/developed using python 3.6

1.2.2 Installing

The easiest way to install the library (for using it), is using:

```
pip install dvb.datascience
```

Development

(in the checkout directory): For installing the checkouts repo for developing of dvb.datascience:

```
pipenv install --dev
```

For using dvb.datascience in your project:

```
pipenv install dvb.datascience
```

Development - Anaconda

(in the checkout directory): Create and activate an environment + install the package:

```
conda create --name dvb.datascience
conda activate dvb.datascience
pip install -e .
```

or use it via:

```
pip install dvb.datascience
```

Jupyter table-of-contents

When working with longer pipelines, the output when using a jupyter notebook can become quite long. It is advisable to install the nbextensions for the toc2 extension:

```
pip install jupyter_contrib_nbextensions
jupyter contrib nbextension install
```

Next, start a jupyter notebook and navigate to edit > nbextensions config and enable the toc2 extension. And optionally set other properties. After that, navigate back to your notebook (refresh) and click the icon in the menu for loading the toc in the side panel.

1.3 Examples

This example loads the data and makes some plots of the Iris dataset

```
import dvb.datascience as ds


p = ds.Pipeline()
p.addPipe('read', ds.data.SampleData('iris'))
p.addPipe('split', ds.transform.TrainTestSplit(test_size=0.3), [("read", "df", "df")])
p.addPipe('boxplot', ds.eda.BoxPlot(), [("split", "df", "df")])
p.fit_transform(transform_params={'split': {'train': True}})
```

This example shows a number of features of the package and its usage:

- Adding 3 steps to the pipeline using `addPipe()`.
- Linking the 3 steps using `[("read", "df", "df")]`: the `'df'` output (2nd parameter) of the `"read"` method (1st method) to the `"df"` input (3rd parameter) of the split method.
- The usage of 3 subpackages: `ds.data`, `ds.transform` and `ds.eda`. The other 2 packages are: `ds.predictor` and `ds.score`.
- The last method `p.fit_transform()` has as a parameter additional input for running the defined pipeline, which can be different for each call to the `p.fit_transform()` or `p.transform()` method.

This example applies the KNeighborsClassifier from sklearn to the Iris dataset

```
import dvb.datascience as ds

from sklearn.neighbors import KNeighborsClassifier
p = ds.Pipeline()
p.addPipe('read', ds.data.SampleData('iris'))
p.addPipe('clf', ds.predictor.SklearnClassifier(KNeighborsClassifier, n_neighbors=3),␣
↪[("read", "df", "df"), ("read", "df_metadata", "df_metadata")])
p.addPipe('score', ds.score.ClassificationScore(), [("clf", "predict", "predict"), (
↪"clf", "predict_metadata", "predict_metadata")])
p.fit_transform()
```

This example shows:

- The use of the `KNeighborsClassifier` from `sklearn`
- The usage of coupling of multiple parameters as input: `[("read", "df", "df"), ("read", "df_metadata", "df_metadata")]`

For a more extensive overview of all the features, see the docs directory.

1.4 Unittesting

The unittests for the project can be run using pytest:

```
pytest
```

1.4.1 Code coverage

Pytest will also output the coverage tot the console.

To generate an html report, you can use:

```
py.test --cov-report html
```

1.5 Code styling

Code styling is done using Black

1.6 Built With

For an extensive list, see setup.py

- scipy / numpy / pandas / matplotlib - For calculations and visualizations
- sklearn - Machine learning algorithms
- statsmodels - Statistics
- mlxtend - Feature selection
- tabulate - Printing tabular data
- imblearn - SMOTE

1.7 Contributing

Please read CONTRIBUTING.md for details on our code of conduct, and the process for submitting pull requests to us.

1.8 Versioning

We use SemVer for versioning. For the versions available, see the tags on this repository.

1.9 Authors

- **Marc Rijken** - *Initial work* - mrijken
- **Wouter Poncin** - *Maintenance* - wpbs
- **Daan Knoope** - *Contributor* - daanknoope
- **Christopher Huijting** - *Contributor* - chuijting

See also the list of contributors who participated in this project.

1.10 License

This project is licensed under the MIT License - see the LICENSE file for details

1.11 Contact

For any questions please don't hesitate to contact us at tc@devolksbank.nl

1.12 Work in progress

- Adding support for multiclass classification problems
- Adding support for regression problems
- Adding support for Apache Spark ML

1.12.1 dvb.datascience

dvb package

Subpackages

dvb.datascience package

Subpackages

dvb.datascience.data package

Submodules

dvb.datascience.data.arff module

class dvb.datascience.data.arff.**ARFFDataExportPipe**

Bases: *dvb.datascience.pipe_base.PipeBase*

Exports ARFF files and writes it to file.

Args: file_path (str): String with a path to the file to import wekaname (str): The wekaname to be used

Returns: A file.

file_path = None

fit(*data: Dict[str, Any]*, *params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('file_path', None, None), ('wekaname', None, None)]

input_keys = ('df',)

output_keys = ()

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

wekaname = None

class dvb.datascience.data.arff.**ARFFDataImportPipe**
Bases: *dvb.datascience.pipe_base.PipeBase*

Imports ARFF files and returns a dataframe.

Args: file_path (str): String with a path to the file to import

Returns: A dataframe

file_path = None

fit(*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('file_path', None, None)]

input_keys = ()

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.data.csv module

class dvb.datascience.data.csv.**CSVDataExportPipe**(*file_path: str = None, sep: str = None, **kwargs*)
Bases: *dvb.datascience.pipe_base.PipeBase*

Exports a dataframe to CSV. Takes as input filepath (str), sep (str). Returns a CSV file at the specified location.

input_keys = ('df',)

output_keys = ()

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.data.csv.**CSVDataImportPipe**(*file_path: str = None, content: str = None, sep: bool = None, engine: str = 'python', index_col: str = None*)
Bases: *dvb.datascience.pipe_base.PipeBase*

Imports data from CSV and creates a dataframe using pd.read_csv().

Args: filepath (str): path to read file content (str): raw data to import sep (bool): separation character to use engine (str): engine to be used, default is "python" index_col (str): column to use as index

Returns: A dataframe with index_col as index column.

input_keys = ()

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.data.excel module

class dvb.datascience.data.excel.**ExcelDataImportPipe**(*file_path: str = None, sheet_name=0, index_col: str = None*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Imports data from excel and creates a dataframe using pd.read_excel().

Args: filepath(str): path to read file sheet_name(int): sheet number to be used (default 0) index_col(str): index column to be used

Returns: A dataframe with index_col as index column.

input_keys = ()

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.data.teradata module

class dvb.datascience.data.teradata.**TeraDataImportPipe**

Bases: *dvb.datascience.pipe_base.PipeBase*

Reads data from Teradata and returns a dataframe.

Args: file_path(str): path to read file containing SQL query sql(str): raw SQL query to be used

Returns: A dataframe using pd.read_sql_query(), sorts the index alphabetically.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.data.teradata.**customDataTypeConverter**

Bases: teradata.datatypes.DefaultDataTypeConverter

Transforms data types from Teradata to datatypes used by Python. Replaces decimal comma with decimal point. Changes BYTEINT, BIGINT, SMALLINT and INTEGER to the Python type int.

convertValue(*dbType*, *dataType*, *typeCode*, *value*)

Converts the value returned by the database into the desired python object.

Module contents

class dvb.datascience.data.**DataPipe**(*key: str = 'data', data=None*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Add some data to the pipeline via fit or transform params. The data can be added on three different moments:

```
>>> pipe = DataPipe(data=[1,2,3])
>>> pipeline.fit_transform(fit_params={"data": [4,5,6]})
>>> pipeline.transform(transform_params={"data": [7,8,9]})
```

The last data will be used.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.data.**GeneratedSampleClassification**(*n_classes: int = 10, n_features: int = 20, n_samples: int = 100, random_state: int = None*)
Bases: *dvb.datascience.pipe_base.PipeBase*

input_keys = ()

output_keys = ('df', 'df_metadata')

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.data.**SampleData**(*dataset_name: str = 'iris'*)
Bases: *dvb.datascience.pipe_base.PipeBase*

input_keys = ()

output_keys = ('df', 'df_metadata')

possible_dataset_names = ['iris', 'diabetes', 'wine', 'boston', 'breast_cancer', 'digi

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda package

Submodules

dvb.datascience.eda.andrews module

class dvb.datascience.eda.andrews.**AndrewsPlot**(*column: str*)
Bases: *dvb.datascience.pipe_base.PipeBase*

Create an andrews curves plot of the data in the dataframe

Args: data: Dataframe with the used data column: Target column to be used in the andrews curves plot

Returns: The plot.

input_keys = ('df',)

output_keys = ('figs',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.base module

class dvb.datascience.eda.base.**AnalyticsBase**
Bases: *dvb.datascience.pipe_base.PipeBase*

fit(*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

get_number_of_dfs()

set_fig(*idx: Any*)
Set in plt the figure to one to be used.

If 'idx' has already been used, the data will be added to the plot used with this idx. If not, a new figure will be created.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.boxplot module

class dvb.datascience.eda.boxplot.**BoxPlot**
Bases: *dvb.datascience.pipe_base.PipeBase*

Create boxplots of every feature in the dataframe.

Args: data: Dataframe to be used in the plotting. Note that only dataframes consisting entirely out of integers or floats can be used, as strings cannot be boxplotted.

Returns: Displays the boxplots .

input_keys = ('df',)

output_keys = ('figs',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.corrmatrix module

class dvb.datascience.eda.corrmatrix.**CorrMatrixPlot**
Bases: *dvb.datascience.pipe_base.PipeBase*

Make a plot of the correlation matrix using all the features in the dataframe

Args: data: Dataframe to be used

Returns: Plot of a correlation matrix

input_keys = ('df',)

output_keys = ('fig', 'corr')

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.describe module

class dvb.datascience.eda.describe.**Describe**
Bases: *dvb.datascience.eda.base.AnalyticsBase*

Describes the data.

Args: data: Dataframe to be used.

Returns: A description of the data.

input_keys = ('df',)

output_keys = ('output',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.dimension_reduction module

class dvb.datascience.eda.dimension_reduction.**DimensionReductionPlots**(*y_label*)
Bases: *dvb.datascience.eda.base.AnalyticsBase*

Plot dimension reduction graphs of the data.

Args: data: Dataframe to be used.

Returns: Dimension reduction (PCA, ISOMAP, MDS, Spectral Embedding, tSNE) plots of the data

input_keys = ('df',)

output_keys = ('figs',)

scatterPlot(*X*, *y*, *title*)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.dump module

class dvb.datascience.eda.dump.**Dump**
Bases: *dvb.datascience.eda.base.AnalyticsBase*

Dump the read data

Args: data: Dataframe to be used

Returns: Empty dataframe.

input_keys = ('df',)

output_keys = ('output',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.ecdf module

class dvb.datascience.eda.ecdf.**ECDFPlots**
Bases: *dvb.datascience.pipe_base.PipeBase*

Creates an empirical cumulative distribution function (ECDF) plot of every feature in the dataframe.

Args: data: Dataframe to be used in the plotting.

Returns: Plots of an ECDF for every feature.

static ecdf(*data*)
Compute ECDF for a one-dimensional array of measurements.

input_keys = ('df',)

output_keys = ('figs',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.hist module

class dvb.datascience.eda.hist.**Hist**(*show_count_labels=True, title='Histogram', groupBy: str = None*)
Bases: *dvb.datascience.pipe_base.PipeBase*

Create histograms of every feature.

Args: data: Dataframe to be used in creating the histograms show_count_labels (Boolean): determines of the number is displayed above every bin (default = True) title (str): what title to display above every histogram (default = "Histogram") groupBy (str): this string will enable multiple bars in every bin, based on the groupBy column (default = None)

Returns: Plots of all the histograms.

input_keys = ('df',)

output_keys = ('figs',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.logit_summary module

class dvb.datascience.eda.logit_summary.**LogitSummary**(*use_regularized: bool = True*, ***kwargs*)

Bases: *dvb.datascience.classification_pipe_base.ClassificationPipeBase*

Run a statsmodels logit for coefficient interpretation on the training set

Args: use_regularized (Boolean): Will determine if data is fitted regularized or not (default = True) data: Dataframe to be used for this function

Returns: A summary of the logit.

input_keys = ('df', 'df_metadata')

output_keys = ('summary',)

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.scatter module

class dvb.datascience.eda.scatter.**ScatterPlots**

Bases: *dvb.datascience.pipe_base.PipeBase*

Create scatterplots of all the features in a dataframe. This function generates scatterplots on every unique combination of features. As the number of features grows, so does the loading time of this function, so this can take a long time.

Args: data: Dataframe, whose features will be used to create swarm plots.

Returns: Plots of all the scatterplots.

input_keys = ('df',)

output_keys = ('figs',)

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.eda.swarm module

class dvb.datascience.eda.swarm.**SwarmPlots**

Bases: *dvb.datascience.pipe_base.PipeBase*

Create swarmplots of all the features in a dataframe. This function generates swarmplots on every unique combination of features. As the number of features grows, so does the loading time of this function.

Args: data: Dataframe, whose features will be used to create swarm plots.

Returns: Plots all the swarmplots.

input_keys = ('df',)

output_keys = ('figs',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

Module contents

dvb.datascience.predictor package

Module contents

class dvb.datascience.predictor.**CostThreshold**(*costFalseNegative: float = 1.0*, *costFalsePositive: float = 1.0*)
Bases: *dvb.datascience.predictor.ThresholdBase*

class dvb.datascience.predictor.**GridSearchCVProgressBar**(*estimator*, *param_grid*, *scoring=None*, *fit_params=None*, *n_jobs=None*, *iid='warn'*, *refit=True*, *cv='warn'*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score='raise-deprecating'*, *return_train_score='warn'*)
Bases: sklearn.model_selection._search.GridSearchCV

Monkey patch to have a progress bar during grid search

class dvb.datascience.predictor.**PrecisionRecallThreshold**
Bases: *dvb.datascience.predictor.ThresholdBase*

class dvb.datascience.predictor.**SklearnClassifier**(*clf*, ***kwargs*)
Bases: *dvb.datascience.classification_pipe_base.ClassificationPipeBase*

Wrapper for inclusion of sklearn classifiers in the pipeline.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('clf', 'pickle', 'pickle'), ('threshold', None, None)]

input_keys = ('df', 'df_metadata')

output_keys = ('predict', 'predict_metadata')

threshold = None

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.predictor.**SklearnGridSearch**(*clf*, *param_grid*, *scoring: str = 'roc_auc'*)
Bases: *dvb.datascience.predictor.SklearnClassifier*

fit (*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

input_keys = ('df', 'df_metadata')

output_keys = ('predict', 'predict_metadata')

class dvb.datascience.predictor.**TPOTClassifier**(***kwargs*)
Bases: *dvb.datascience.predictor.SklearnClassifier*

fit (*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

class dvb.datascience.predictor.**ThresholdBase**
Bases: abc.ABC

What does this do?

set_y (*y_true*, *y_pred_proba*, *y_pred_proba_labels*, ***kwargs*)

y_pred_proba = None

y_pred_proba_labels = None

y_true = None

dvb.datascience.transform package

Submodules

dvb.datascience.transform.classes module

class dvb.datascience.transform.classes.**LabelBinarizerPipe**
Bases: *dvb.datascience.pipe_base.PipeBase*

Split label column in different columns per label value

fit (*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('lb', 'pickle', 'pickle')]

input_keys = ('df',)

lb = None

output_keys = ('df', 'df_metadata')

transform (*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.core module

class dvb.datascience.transform.core.**GetCoreFeatures**(*model=None*, *n_features: int = 10*, *method='RFE'*)

Bases: *dvb.datascience.classification_pipe_base.ClassificationPipeBase*

Get the features (maximum *n_features*) which are key to the label.

fit(*data: Dict[str, Any]*, *params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('core_features', None, None)]

get_core_features(*X*, *y*) → List[str]

input_keys = ('df', 'df_metadata')

output_keys = ('features',)

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.features module

class dvb.datascience.transform.features.**ComputeFeature**(*column_name*, *f: Callable*, *c: Callable = None*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Add a computed feature to the dataframe

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.features.**DropFeatures**(*features: List[str] = None*, *features_function: Callable = None*)

Bases: *dvb.datascience.transform.features.DropFeaturesMixin*, *dvb.datascience.transform.features.SpecifyFeaturesBase*

class dvb.datascience.transform.features.**DropFeaturesMixin**

Bases: *dvb.datascience.transform.features.FeaturesBase*

Mixin for classes which will drop features. Superclasses needs to set self.features, which contains the features which will be dropped.

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.features.**DropHighlyCorrelatedFeatures**(*threshold: float = 0.9, absolute: bool = True*)

Bases: *dvb.datascience.transform.features.DropFeaturesMixin*, *dvb.datascience.transform.features.FeaturesBase*

When two columns are highly correlated, one will be removed. From a pair of correlated columns the one that is the latest one in the list of columns, will be removed.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

class dvb.datascience.transform.features.**DropNonInvertibleFeatures**

Bases: *dvb.datascience.transform.features.DropFeaturesMixin*, *dvb.datascience.transform.features.FeaturesBase*

Drops features that are not invertible, to prevent singularity.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

static is_invertible(*a*)

class dvb.datascience.transform.features.**FeaturesBase**

Bases: *dvb.datascience.pipe_base.PipeBase*, abc.ABC

features = None

fit_attributes = [('features', None, None)]

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.features.**FilterFeatures**(*features: List[str] = None, features_function: Callable = None*)

Bases: *dvb.datascience.transform.features.SpecifyFeaturesBase*

FilterFeatures returns a dataframe which contains only the specified columns. Note: when a request column does not exists in the input dataframe, this will be silently ignored.

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.features.**FilterTypeFeatures**(*type_=<class 'numpy.number'>*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Keep only the columns of the given type (np.number is default)

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.features.**SpecifyFeaturesBase**(*features: List[str] = None, features_function: Callable = None*)

Bases: *dvb.datascience.transform.features.FeaturesBase*

Base class for classes which can be initialised with a list of features or a callable which compute those features. The superclass needs to speficify what will be done with the feautures during transform.

features_function = None

fit(*data: Dict[str, Any], params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.filter module

class dvb.datascience.transform.filter.**FilterObservations**(*filter_: Callable*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Filter observations by row based on a function

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.impute module

class dvb.datascience.transform.impute.**CategoricalImpute**(*missing_values='NaN', strategy='mode', replacement=''*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Impute missing values from a categorical/string np.ndarray or pd.Series with the most frequent value on the training data.

Args:

missing_values [string or "NaN", optional (default="NaN")] The placeholder for the missing values. All occurrences of *missing_values* will be imputed. None and np.nan are treated as being the same, use the string value "NaN" for them.

strategy [string, optional (default = 'mode')] If set to 'mode', replace all instances of *missing_values* with the modal value. Otherwise, replace with the value specified via *replacement*.

replacement [string, optional (default='?')] The value that all instances of *missing_values* are replaced with if *strategy* is not set to 'mode'. This is useful if you don't want to impute with the mode, or if there are multiple modes in your data and you want to choose a particular one. If *strategy* is set to *mode*, this parameter is ignored.

fill [str] Most frequent value of the training data.

fit (*data: Dict[str, Any], params: Dict[str, Any]*)
Get the most frequent value.

fit_attributes = [('fill', 'pickle', 'pickle')]

input_keys = ('df',)

output_keys = ('df',)

transform (*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Replaces missing values in the input data with the most frequent value of the training data.

class dvb.datascience.transform.impute.**ImputeWithDummy**(*strategy: str = 'median', impValueTrain=None*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Impute missing values with the mean, median, mode or set to a value. Takes as input strategy (str). Possible strategies are "mean", "median", "mode" and "value". If the strategy is "value", an extra argument impValueTrain can be given, denoting which value should be set.

fit (*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('impValueTrain', 'pickle', 'pickle')]

impValueTrain = None

input_keys = ('df',)

output_keys = ('df',)

possible_strategies = ['mean', 'median', 'mode', 'value']

transform (*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.metadata module

class dvb.datascience.transform.metadata.**MetadataPipeline**(*file_path: str, remove_vars: List = None*)

Bases: *dvb.datascience.sub_pipe_base.SubPipelineBase*

Read metadata and make some pipes for processing the data

input_keys = ('df',)

output_keys = ('df',)

dvb.datascience.transform.outliers module

class dvb.datascience.transform.outliers.**RemoveOutliers**(*nr_of_std: int = 6, skip_columns: List[str] = None, min_outliers: int = 1*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Remove observations when at least one of the features has an outlier.

Args: nr_of_std (int): the number of standard deviations the outlier has to be higher/lower than, to be removed (default = 6) skip_columns (List[str]): columns to be skipped min_outliers (int): minimum number of outliers a row must have, to be removed from the dataframe (default = 1)

Returns: The dataframe, minus any rows with min_outliers of outliers.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('boundaries', 'pickle', 'pickle')]

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.outliers.**ReplaceOutliersFeature**(*method: str = 'median', nr_of_std: float = 1.5*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Replace all outliers in features with the median, mean or a clipped value.

Args:

method (str): what method to use when replacing. (default = median) Options are: - median , replace outliers with median of feature - mean , replace outliers with mean of feature - clip , replace outliers with nr_of_std standard deviations +/- mean

nr_of_std (int): minimum number of outliers a row must have, to be removed from the dataframe (default = 1)

Returns: The dataframe, with outliers replaced by the method indicated.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('features_mean', 'pickle', 'pickle'), ('features_median', 'pickle',

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.pandaswrapper module

class dvb.datascience.transform.pandaswrapper.**PandasWrapper**(*s: Callable[pandas.core.frame.DataFrame, pandas.core.frame.DataFrame]*)
Bases: *dvb.datascience.pipe_base.PipeBase*

Generic Wrapper for Pandas operations. The callable will get the DataFrame from the input 'df' and the returned DataFrame will be put in the output 'df'.

Besides the DataFrame, the callable gets the transform_params, so these can be used to change the operation.

input_keys = ('df',)

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.sklearnwrapper module

class dvb.datascience.transform.sklearnwrapper.**SKLearnBase**
Bases: object

fit(*data: Any*)

transform(*data: Any*)

class dvb.datascience.transform.sklearnwrapper.**SKLearnWrapper**(*cls*, ***kwargs*)
Bases: *dvb.datascience.pipe_base.PipeBase*

Generic SKLearn fit / transform wrapper Geen idee wat dit precies doet...

fit(*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = [('s', 'pickle', 'pickle')]

input_keys = ('df',)

output_keys = ('df',)

s = None

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.smote module

class dvb.datascience.transform.smote.**SMOTESampler**(***kwargs*)
Bases: *dvb.datascience.classification_pipe_base.ClassificationPipeBase*

Resample the dataset.

Note: the new df will not the indexes, because of extra creates row, the indexes won't be unique anymore.

fit(*data: Dict[str, Any], params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

input_keys = ('df', 'df_metadata')

output_keys = ('df',)

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.transform.split module

class dvb.datascience.transform.split.**CallableTrainTestSplit**(*c: Callable[Any, int]*)
Bases: *dvb.datascience.transform.split.TrainTestSplitBase*

Return the train, the test set or the complete set, as defined in params['split'].

For every row, the callable will be called with the row as single argument and returns - CallableTrainTestSplit.TEST - CallableTrainTestSplit.TRAIN

When the return value is not equal to TEST or TRAIN, the row will be excluded from both sets.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.split.**RandomTrainTestSplit**(*random_state: int = 42, test_size: float = 0.25*)
Bases: *dvb.datascience.transform.split.TrainTestSplitBase*

Return the train, the test set or the complete set, as defined in params['split']. The split will be random. A random state is present at default, to make the pipeline reproducable.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.transform.split.**TrainTestSplit**(**args*, ***kwargs*)
Bases: *dvb.datascience.transform.split.RandomTrainTestSplit*

class dvb.datascience.transform.split.**TrainTestSplitBase**
Bases: *dvb.datascience.pipe_base.PipeBase*

ALL = -1

TEST = 1

TRAIN = 0

input_keys = ('df',)

output_keys = ('df',)

dvb.datascience.transform.union module

class dvb.datascience.transform.union.**Union**(*number_of_dfs*, *join: str = 'outer'*, *axis=1*, *remove_duplicated_columns: bool = False*)

Bases: *dvb.datascience.pipe_base.PipeBase*

Merge the result of different Pipes. Merge can be done based on columns (default, axis=1) or rows (axis=0). When columns are merged, it's possible that columns are present in more than one input dataframe. At default, the second occurence of the column will be renamed with an underscore as suffix. Optional, duplicated colums are removed.

The input_keys are generated at initialisation based on the the number of dfs, like:

input_keys = ('df0', 'df1', 'df2', ...)

input_keys = ()

output_keys = ('df',)

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

Module contents

Submodules

dvb.datascience.classification_pipe_base module

class dvb.datascience.classification_pipe_base.**ClassificationPipeBase**

Bases: *dvb.datascience.pipe_base.PipeBase*

Base class for classification pipes, so classification related attributes and methods are reusable for different kind of classification based pipes.

X = None

X_labels = None

classes = None

fit_attributes = [('classes', None, None), ('n_classes', None, None), ('y_true_label',

n_classes = 0

threshold = None

transform(*data: Dict[str, Any]*, *params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

y_pred = None

y_pred_label = ''

y_pred_proba = None

y_pred_proba_labels = None

y_true = None

y_true_label = ''

dvb.datascience.pipe_base module

class dvb.datascience.pipe_base.**PipeBase**

Bases: object

Common base class for all pipes

figs = None

fit(*data: Dict[str, Any], params: Dict[str, Any]*)

Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

fit_attributes = ()

fit_transform(*data: Dict[str, Any], transform_params: Dict[str, Any], fit_params: Dict[str, Any]*) → Dict[str, Any]

get_fig(*idx: Any*)

Set in plt the figure to one to be used.

When *idx* has already be used, it will set the same Figure so data can be added to that plot. Otherwise a new Figure will be set

get_transform_data_by_key(*key: str*) → List[Any]

Get all values for a certain key for all transforms

input_keys = ('df',)

load(*state: Dict[str, Any]*)

load all fitted attributes of this Pipe from *state*.

Note: All PipeBase subclasses can define a *fit_attributes* attribute which contains a tuple for every attribute which is set during the fit phase. Those are the attributes which needs to be saved in order to be loaded in a new process without having to train (fit) the pipeline. This is useful ie for model inference. The tuple for every attribute consist of (name, serializer, deserializer).

The (de)serializer are needed to convert to/from a JSON serializable format and can be: - None: No conversion needed, ie for str, int, float, list, bool - 'pickle': The attribute will be pickled and stored as base64, so it can be part of a json - callable: a function which will get the object to be (de)serialized and need to return the (de)serialized version

name = None

output_keys = ('df',)

save() → Dict[str, Any]

Return all fitted attributes of this Pipe in a Dict which is JSON serializable.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]

Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.pipeline module

class dvb.datascience.pipeline.**Pipeline**

Bases: object

A connector specifies which Pipe (identified by its name) and which output from that Pipe (identified by the key of the output) will be input to a Pipe (identified by its name) and which input for that Pipe (identified by its key)

Example

```
>>> pipeline = Pipeline()
>>> pipeline.addPipe('read', ReadCSV())
>>> pipeline.addPipe('impute', Impute(), [("read", "df", "df")])
```

```
>>> pipeline.fit()
```

```
>>> pipeline.transform()
```

addPipe(*name: str, pipe: dvb.datascience.pipe_base.PipeBase, inputs: List[Tuple[Union[str, dvb.datascience.pipe_base.PipeBase], str, str]] = None, comment: str = None*) → dvb.datascience.pipeline.Pipeline

Add a pipe *pipe* to the pipeline with the given name. Optionally add the input connectors by adding them to *inputs*. *inputs* is a list of the inputs whith for each input a tuple with (output_pipe, output_key, input_key).

current_transform_nr = -1

draw_design()

Returns an image with all pipes and connectors.

end()

When all fit and transforms are finished, end the pipeline, so some clean up can be done. At this moment, that is mainly needed to close plots, so they won't be shown twice in the notebook

fit_transform(*data: Optional[Dict[str, Any]] = None, transform_params: Optional[Dict[str, Any]] = None, fit_params: Optional[Dict[str, Any]] = None, name: str = 'fit', close_plt: bool = False*) → None

Train all pipes in the pipeline and run the transform for the first time

fit_transform_try(**args, **kwargs*)

static get_params(*params: Dict, key: str, metadata: Dict = None*) → Dict

Get a dict with the contents of params only relevant for the pipe with the given key as name. Besides that, also the params['default'] and metadata will be added.

get_pipe(*name*) → Optional[dvb.datascience.pipe_base.PipeBase]

get_pipe_input(*name*) → Optional[Dict]

Get the input for the pipe with *name* from the transformed outputs. Returns a dict with all data when all data for the pipe are collectable. Returns None when not all data is present yet

get_pipe_output(*name: str, transform_nr: int = None*) → Dict

Get the output of the pipe with *name* and the given *transform_nr* (which default to None which will selects the last one). When no output is present, an empty dict is returned

get_processable_pipes() → List[dvb.datascience.pipe_base.PipeBase]

get the pipes which are processable give the status of the pipeline

input_connectors = None

static is_valid_name(*name*)

load(*file_path: str*) → None
Load the fitted parameters from the file in *file_path* and load them in all Pipes.

output_connectors = None

pipes = None

reset_fit()

save(*file_path: str*) → None
Save the fitted parameters from alle Pipes to the file in *file_path*.

transform(*data: Optional[Dict[str, Any]] = None*, *transform_params: Optional[Dict[str, Any]] = None*, *fit_params: Optional[Dict[str, Any]] = None*, *fit: bool = False*, *name: Optional[str] = None*, *close_plt: bool = False*)
When transform_params or fit_params contain a key 'default', that params will be given to all pipes, unless it is overridden by a specific value for that pipe in transform_params or fit_params. The default can be useful for params which are needed in a lot of pipes.

transform_outputs = None

transform_status = None

transform_try(**args*, ***kwargs*)

class dvb.datascience.pipeline.**Status**
Bases: enum.Enum

An enumeration.

FINISHED = 3

NOT_STARTED = 1

PROCESSING = 2

dvb.datascience.score module

class dvb.datascience.score.**ClassificationScore**(*score_methods: List[str] = None*)
Bases: *dvb.datascience.classification_pipe_base.ClassificationPipeBase*

Some scores for classification problems

accuracy() → float

auc() → Optional[float]

classification_report()

confusion_matrix()

fit(*data: Dict[str, Any]*, *params: Dict[str, Any]*)
Train on a dataset *df* and store the learnings so *transform* can be called later on to transform based on the learnings.

input_keys = ('predict', 'predict_metadata')

log_loss()

mcc(*threshold: float = None*) → float

output_keys = ('scores',)

params = None

plot_auc()

plot_confusion_matrix()

plot_model_performance()

possible_predict_methods = ['plot_model_performance']

possible_score_methods = ['auc', 'plot_auc', 'accuracy', 'mcc', 'confusion_matrix', 'p

precision_recall_curve()

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

dvb.datascience.sub_pipe_base module

class dvb.datascience.sub_pipe_base.**PassData**(*subpipeline*, *output_keys*)
Bases: *dvb.datascience.pipe_base.PipeBase*

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

class dvb.datascience.sub_pipe_base.**SubPipelineBase**(*output_pipe_name: str*)
Bases: *dvb.datascience.pipe_base.PipeBase*

fit_transform(*data: Dict[str, Any], transform_params: Dict[str, Any], fit_params: Dict[str, Any]*) → Dict[str, Any]

load(*state: Dict[str, Any]*) → None
load all fitted attributes of this Pipe from *state*.

save() → Dict[str, Any]
Return all fitted attributes of this Pipe in a Dict which is JSON serializable.

transform(*data: Dict[str, Any], params: Dict[str, Any]*) → Dict[str, Any]
Perform an operations on *df* using the kwargs and the learnings from training. Transform will return a tuple with the transformed dataset and some output. The transformed dataset will be the input for the next plumber. The output will be collected and shown to the user.

Module contents

dvb.datascience.**load_module**(*name: str*, *disable_warnings: bool = True*, *random_seed: Optional[int] = 1122*) → Any
Convenience function for running an experiment. This function reloads the experiment when it is already loaded, so any changes in the [.. missing word ..] of that experiment will be used. Usage:

import dvb.datascience as ds p = ds.load_module('experiment').run()

p can be used to access the contents of the pipeline, like:

p.get_pipe_output('predict')

in case you define a 'run()' method in 'experiment.py' returning the pipeline object

dvb.datascience.**run_module**(*name: str*, *disable_warnings: bool = True*) → Any

Module contents

1.12.2 Introduction dvb.datascience

In this tutorial, you will learn what the basic usage of dvb.datascience is and how you can use in in your datascience activities.

If you have any suggestions for features or if you encounter any bugs, please let us know at tc@devolksbank.nl

```
In [1]: # %matplotlib inline
        import dvb.datascience as ds
```

Defining a pipeline

Defining a pipeline is just adding some Pipes (actions) which will be connected

Every Pipe can have 0, 1 or more inputs from other pipes. Every Pipe can have 0, 1 or more outputs to other pipes. Every Pipe has a name. Every input and output of the pipe has a key by which the input/output is identified. The name of the Pipe and the key of the input/output are used to connect pipes.

```
In [2]: p = ds.Pipeline()
        p.addPipe("read", ds.data.SampleData(dataset_name="iris"))
        p.addPipe("metadata", ds.data.DataPipe("df_metadata", {"y_true_label": "label"}))
        p.addPipe("write",ds.data.CSVDataExportPipe("dump_input_to_output.csv", sep=",", index_label=
```

```
Out[2]: <dvb.datascience.pipeline.Pipeline at 0x7fcb09083898>
```

A pipeline has two main methods: `fit_transform()` and `transform()`.`fit_transform()` is training the pipeline. Depending on the Pipe, the training can be computing the mean, making a decision tree, etc. During the transform, those learnings are used to transform() the input to output, for example by replacing outliers by means, predicting with the trained model, etc.

```
In [3]: p.fit_transform()
```

```
'Drawing diagram using blockdiag'
```

```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=448x280 at 0x7FCB08F99358>
```

```
<IPython.core.display.HTML object>
```

```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

After the transform, the output of the transform is available.

```
In [4]: p.get_pipe_output('read')
```

```
Out[4]: {'df':      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
          5                  5.4               3.9                1.7               0.4
          6                  4.6               3.4                1.4               0.3
          7                  5.0               3.4                1.5               0.2
          8                  4.4               2.9                1.4               0.2
          9                  4.9               3.1                1.5               0.1
          10                 5.4               3.7                1.5               0.2
          11                 4.8               3.4                1.6               0.2
          12                 4.8               3.0                1.4               0.1
          13                 4.3               3.0                1.1               0.1
          14                 5.8               4.0                1.2               0.2
```

15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
120	6.9	3.2	5.7	2.3
121	5.6	2.8	4.9	2.0
122	7.7	2.8	6.7	2.0
123	6.3	2.7	4.9	1.8
124	6.7	3.3	5.7	2.1
125	7.2	3.2	6.0	1.8
126	6.2	2.8	4.8	1.8
127	6.1	3.0	4.9	1.8
128	6.4	2.8	5.6	2.1
129	7.2	3.0	5.8	1.6
130	7.4	2.8	6.1	1.9
131	7.9	3.8	6.4	2.0
132	6.4	2.8	5.6	2.2
133	6.3	2.8	5.1	1.5
134	6.1	2.6	5.6	1.4
135	7.7	3.0	6.1	2.3
136	6.3	3.4	5.6	2.4
137	6.4	3.1	5.5	1.8
138	6.0	3.0	4.8	1.8
139	6.9	3.1	5.4	2.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

	target
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

```
11         0
12         0
13         0
14         0
15         0
16         0
17         0
18         0
19         0
20         0
21         0
22         0
23         0
24         0
25         0
26         0
27         0
28         0
29         0
..       ...
120        2
121        2
122        2
123        2
124        2
125        2
126        2
127        2
128        2
129        2
130        2
131        2
132        2
133        2
134        2
135        2
136        2
137        2
138        2
139        2
140        2
141        2
142        2
143        2
144        2
145        2
146        2
147        2
148        2
149        2

[150 rows x 5 columns],
'df_metadata': {'classes': ['Setosa', 'Versicolour', 'Virginica'],
 'y_true_label': 'target'},
'y_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10')}
```

Multiple inputs

Some Pipes have multiple inputs, for example to merge two datasets we can do the following.

```
In [5]: p = ds.Pipeline()
        p.addPipe('read1', ds.data.CSVDataImportPipe())
        p.addPipe('read2', ds.data.CSVDataImportPipe())
        p.addPipe('merge', ds.transform.Union(2, axis=0, join='outer'), [("read1", "df", "df0"), ("re
```

```
Out[5]: <dvb.datascience.pipeline.Pipeline at 0x7fcb0b26df60>
```

```
In [6]: p.fit_transform(transform_params={'read1': {'file_path': '../test/data/train.csv'}, 'read2':
```

```
'Drawing diagram using blockdiag'
```

```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=448x280 at 0x7FCB08829550>
```

```
<IPython.core.display.HTML object>
```

```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

```
In [7]: p.get_pipe_output('merge')
```

```
Out[7]: {'df':    age gender  length   name
         0   20      M     180    jan
         1   21      W     164  marie
         2   23      M     194   piet
         3   24      W     177  helen
         4   60      U     188    jan
         0   25      W     161    gea
         1   65      M     181   marc}
```

Plots

It's easy to get some plots of the data:

```
In [8]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('split', ds.transform.RandomTrainTestSplit(test_size=0.3), [("read", "df", "df")])
        p.addPipe('boxplot', ds.eda.BoxPlot(), [("split", "df", "df")])
        p.fit_transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.TR
```

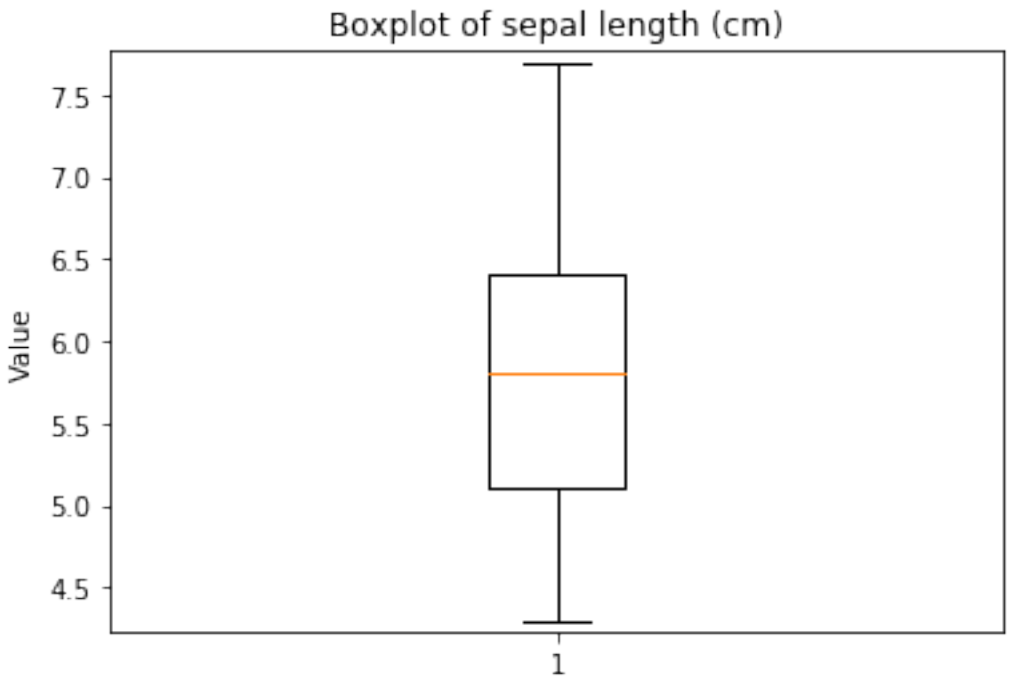
```
'Drawing diagram using blockdiag'
```

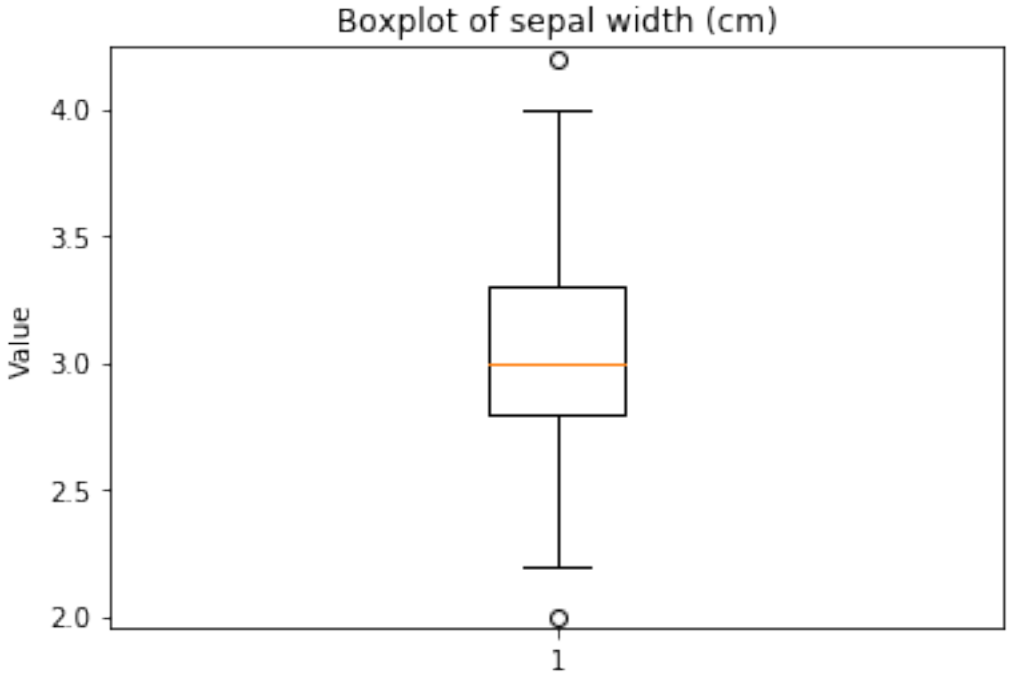
```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB087FD978>
```

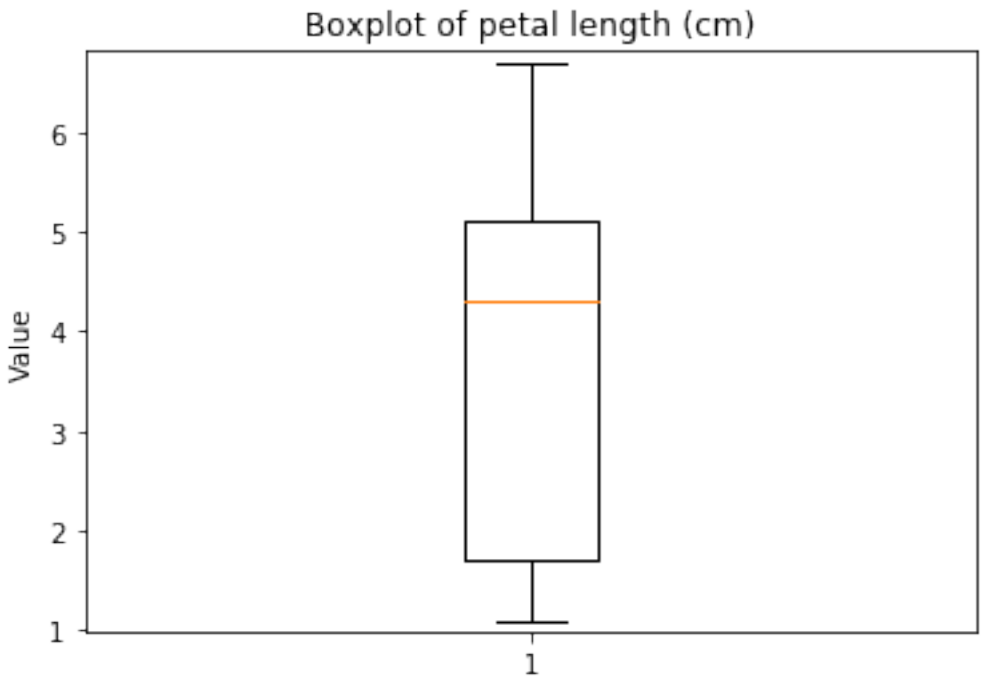
```
<IPython.core.display.HTML object>
```

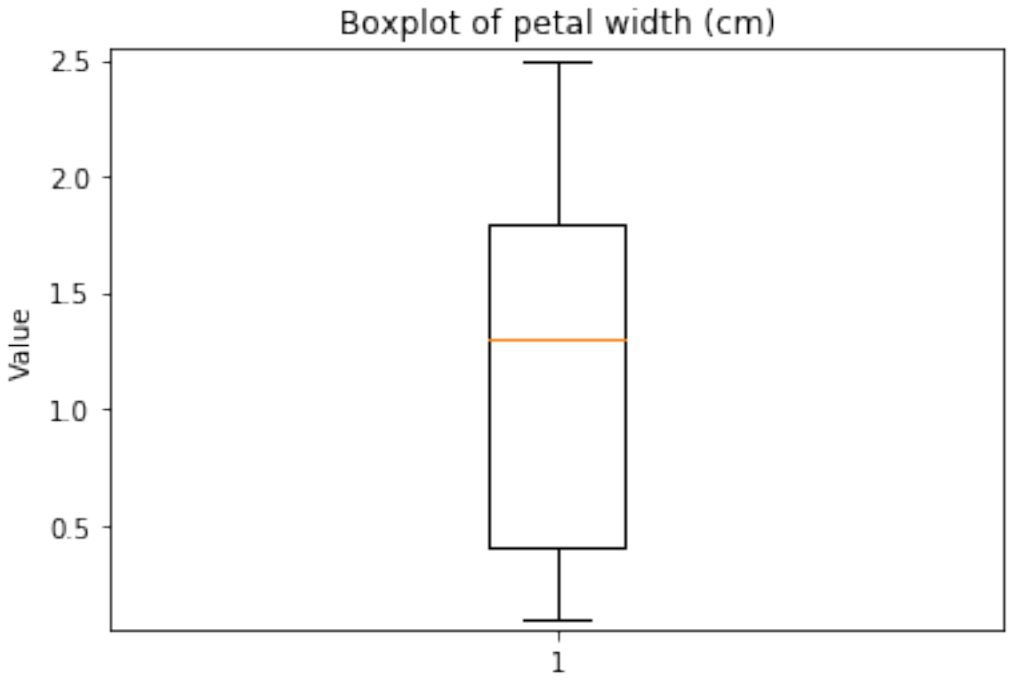
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

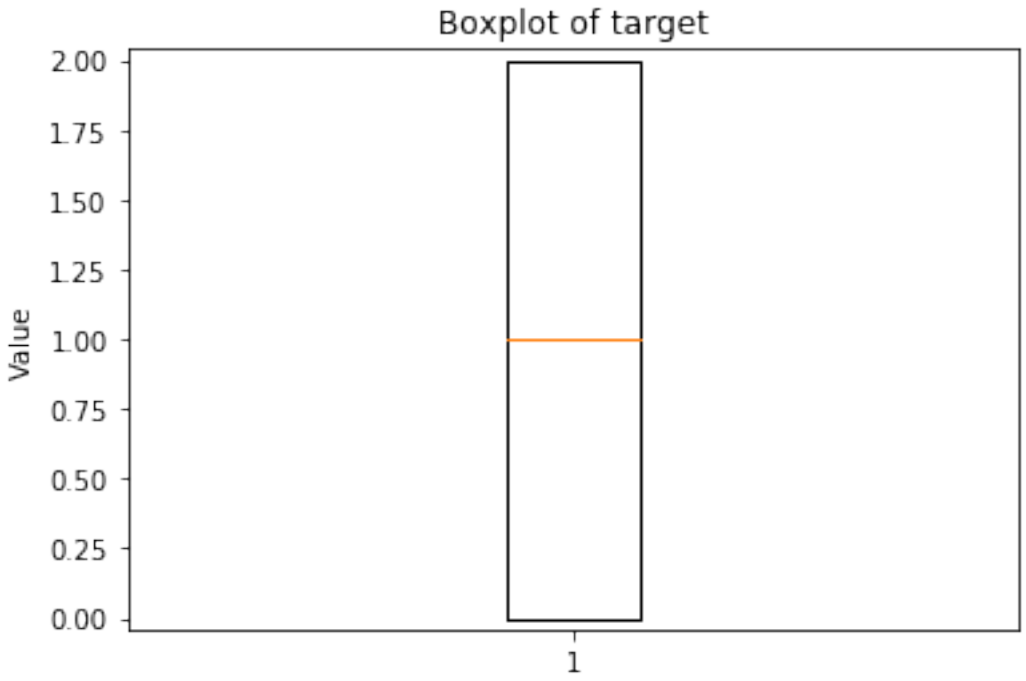
```
<IPython.core.display.HTML object>
```

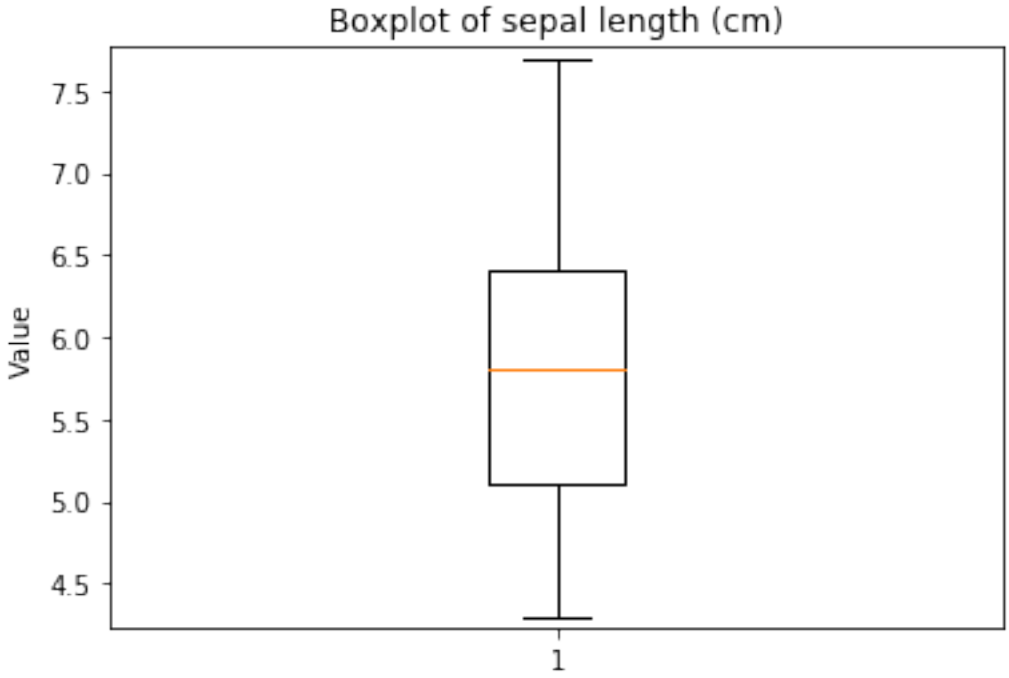

Boxplot of sepal length (cm)
Value
7.5
7.0
6.5
6.0
5.5
5.0
4.5
1

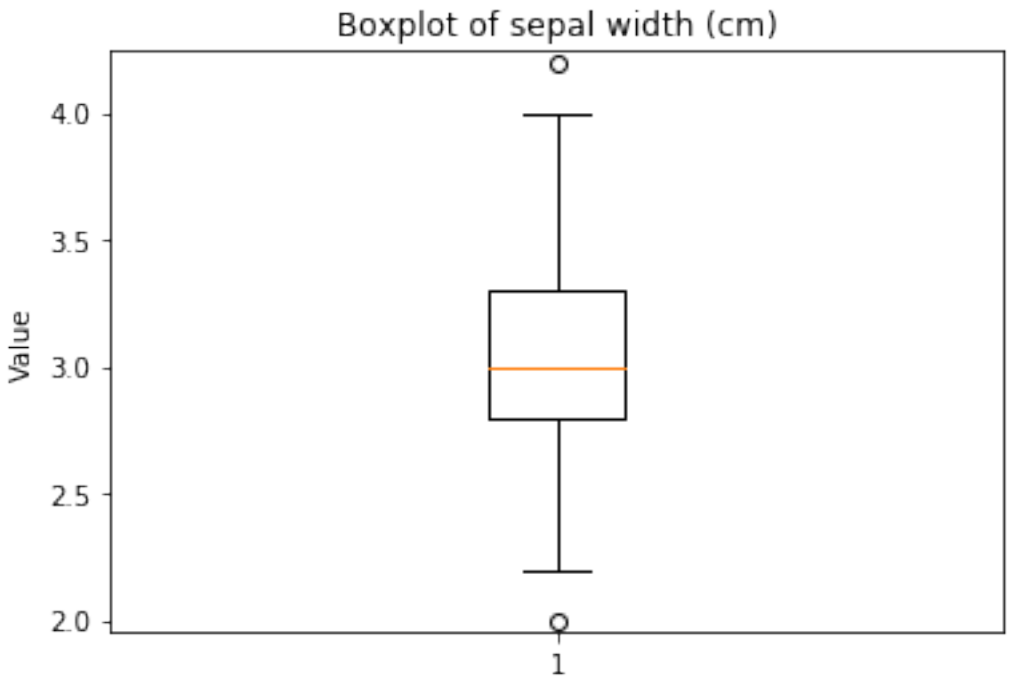

Boxplot of sepal width (cm)
Value
4.0
3.5
3.0
2.5
2.0
1

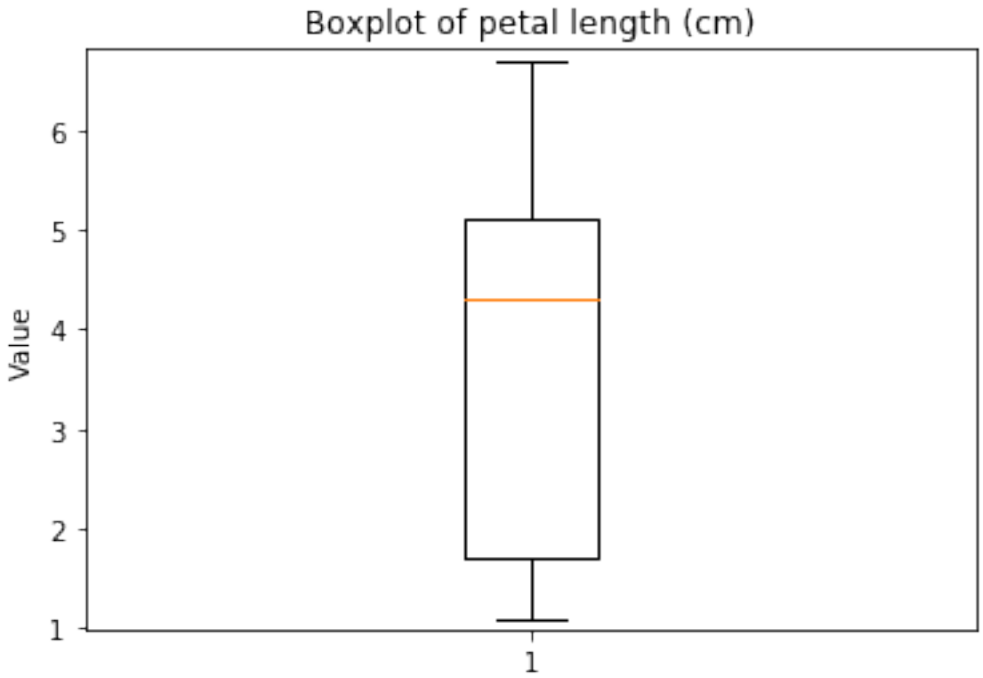

Boxplot of petal length (cm)
Value
6
5
4
3
2
1
1

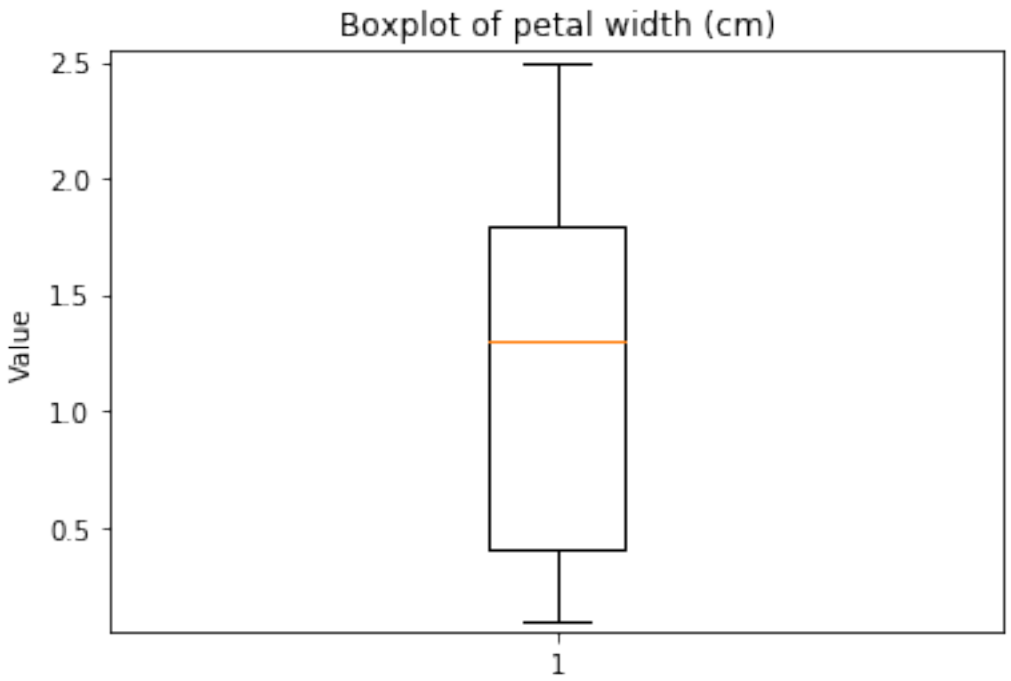

Boxplot of petal width (cm)
Value
2.5
2.0
1.5
1.0
0.5
1

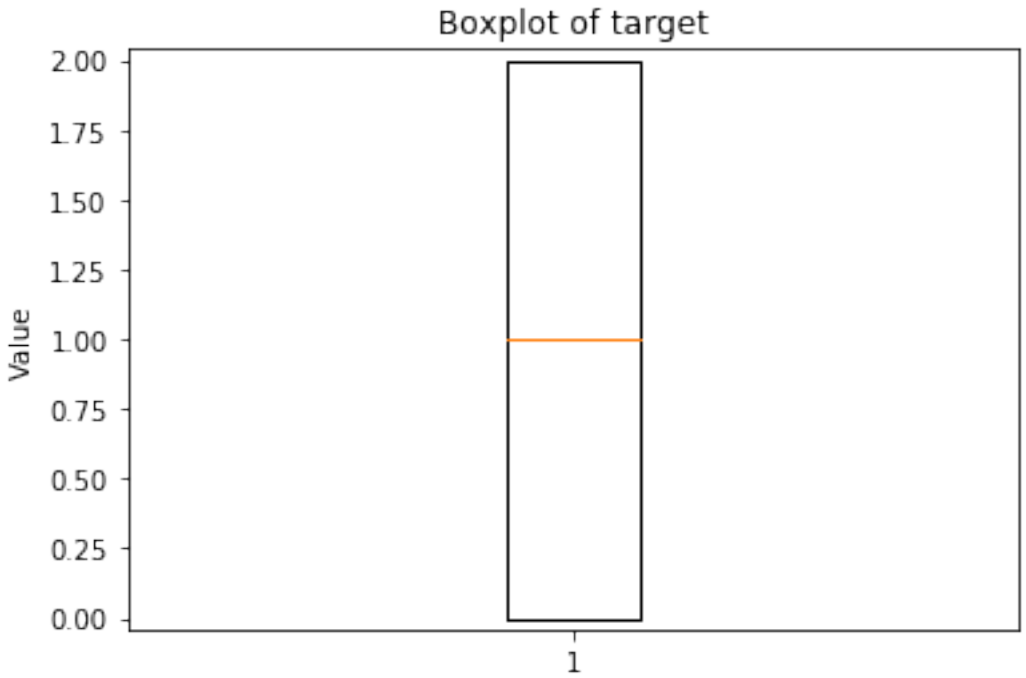

Boxplot of target
Value
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1


Boxplot of sepal length (cm)
Value
7.5
7.0
6.5
6.0
5.5
5.0
4.5
1


Boxplot of sepal width (cm)
Value
4.0
3.5
3.0
2.5
2.0
1


Boxplot of petal length (cm)
Value
6
5
4
3
2
1
1






```
In [9]: p.get_pipe_output('read')
```

```
Out[9]: {'df':      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
          5                  5.4               3.9                1.7               0.4
          6                  4.6               3.4                1.4               0.3
```

```
7                  5.0                3.4                1.5                0.2
8                  4.4                2.9                1.4                0.2
9                  4.9                3.1                1.5                0.1
10                 5.4                3.7                1.5                0.2
11                 4.8                3.4                1.6                0.2
12                 4.8                3.0                1.4                0.1
13                 4.3                3.0                1.1                0.1
14                 5.8                4.0                1.2                0.2
15                 5.7                4.4                1.5                0.4
16                 5.4                3.9                1.3                0.4
17                 5.1                3.5                1.4                0.3
18                 5.7                3.8                1.7                0.3
19                 5.1                3.8                1.5                0.3
20                 5.4                3.4                1.7                0.2
21                 5.1                3.7                1.5                0.4
22                 4.6                3.6                1.0                0.2
23                 5.1                3.3                1.7                0.5
24                 4.8                3.4                1.9                0.2
25                 5.0                3.0                1.6                0.2
26                 5.0                3.4                1.6                0.4
27                 5.2                3.5                1.5                0.2
28                 5.2                3.4                1.4                0.2
29                 4.7                3.2                1.6                0.2
..                 ...                ...                ...                ...
120                6.9                3.2                5.7                2.3
121                5.6                2.8                4.9                2.0
122                7.7                2.8                6.7                2.0
123                6.3                2.7                4.9                1.8
124                6.7                3.3                5.7                2.1
125                7.2                3.2                6.0                1.8
126                6.2                2.8                4.8                1.8
127                6.1                3.0                4.9                1.8
128                6.4                2.8                5.6                2.1
129                7.2                3.0                5.8                1.6
130                7.4                2.8                6.1                1.9
131                7.9                3.8                6.4                2.0
132                6.4                2.8                5.6                2.2
133                6.3                2.8                5.1                1.5
134                6.1                2.6                5.6                1.4
135                7.7                3.0                6.1                2.3
136                6.3                3.4                5.6                2.4
137                6.4                3.1                5.5                1.8
138                6.0                3.0                4.8                1.8
139                6.9                3.1                5.4                2.1
140                6.7                3.1                5.6                2.4
141                6.9                3.1                5.1                2.3
142                5.8                2.7                5.1                1.9
143                6.8                3.2                5.9                2.3
144                6.7                3.3                5.7                2.5
145                6.7                3.0                5.2                2.3
146                6.3                2.5                5.0                1.9
147                6.5                3.0                5.2                2.0
148                6.2                3.4                5.4                2.3
149                5.9                3.0                5.1                1.8

     target
0         0
1         0
2         0
```

```
3          0
4          0
5          0
6          0
7          0
8          0
9          0
10         0
11         0
12         0
13         0
14         0
15         0
16         0
17         0
18         0
19         0
20         0
21         0
22         0
23         0
24         0
25         0
26         0
27         0
28         0
29         0
..       ...
120        2
121        2
122        2
123        2
124        2
125        2
126        2
127        2
128        2
129        2
130        2
131        2
132        2
133        2
134        2
135        2
136        2
137        2
138        2
139        2
140        2
141        2
142        2
143        2
144        2
145        2
146        2
147        2
148        2
149        2
```

```
         [150 rows x 5 columns],
         'df_metadata': {'classes': ['Setosa', 'Versicolour', 'Virginica'],
          'y_true_label': 'target'},
         'y_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10')}
```

```
In [10]: p.transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.TEST
```

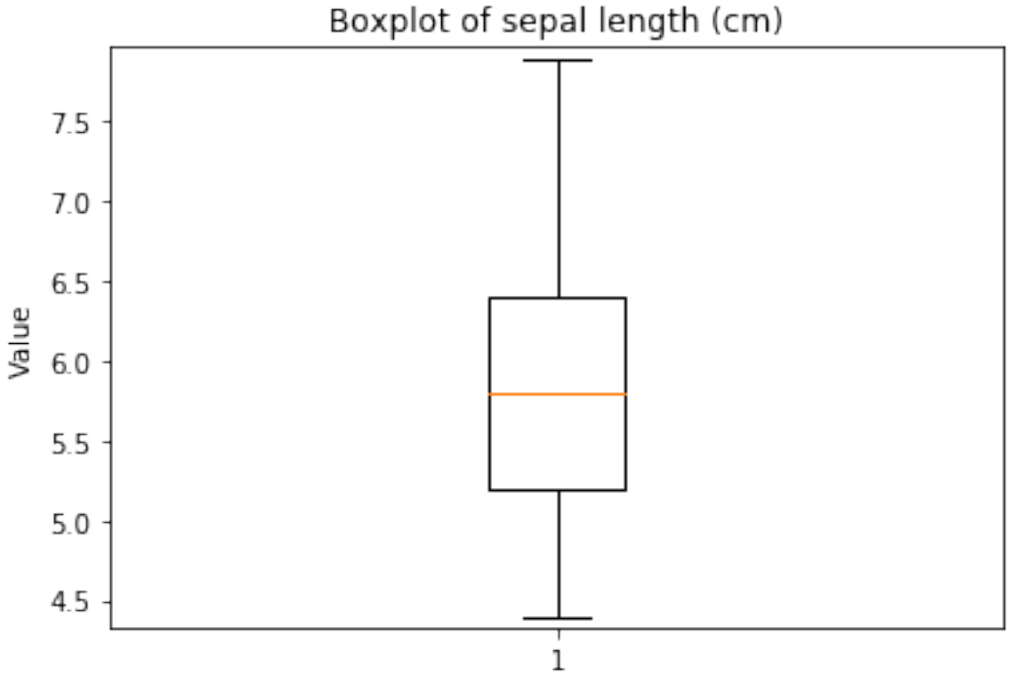
```
'Drawing diagram using blockdiag'
```

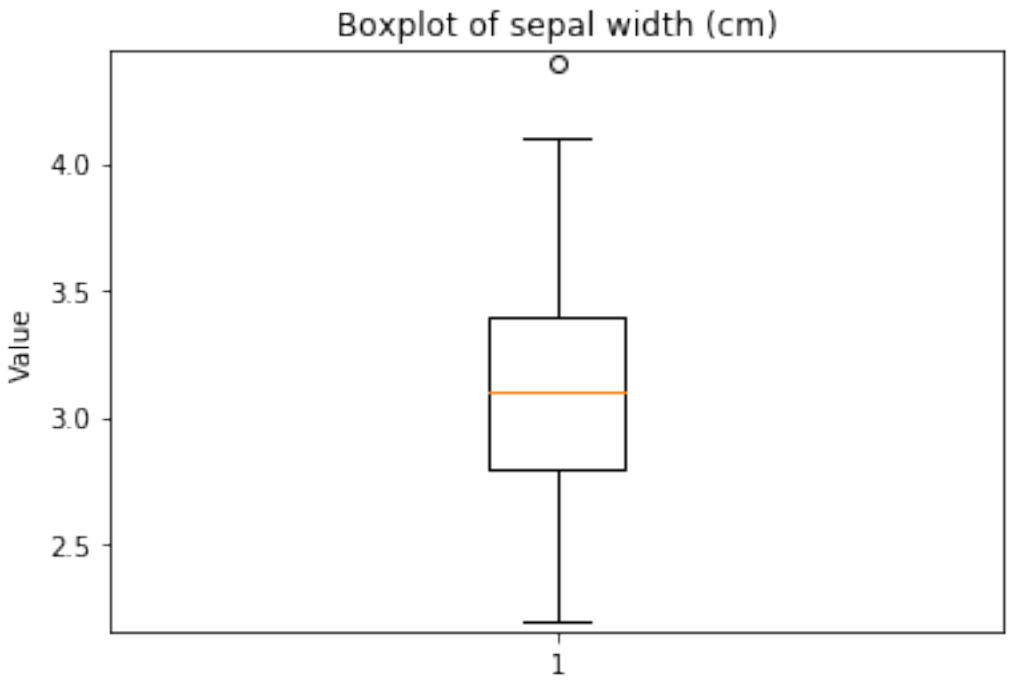
```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB08772D68>
```

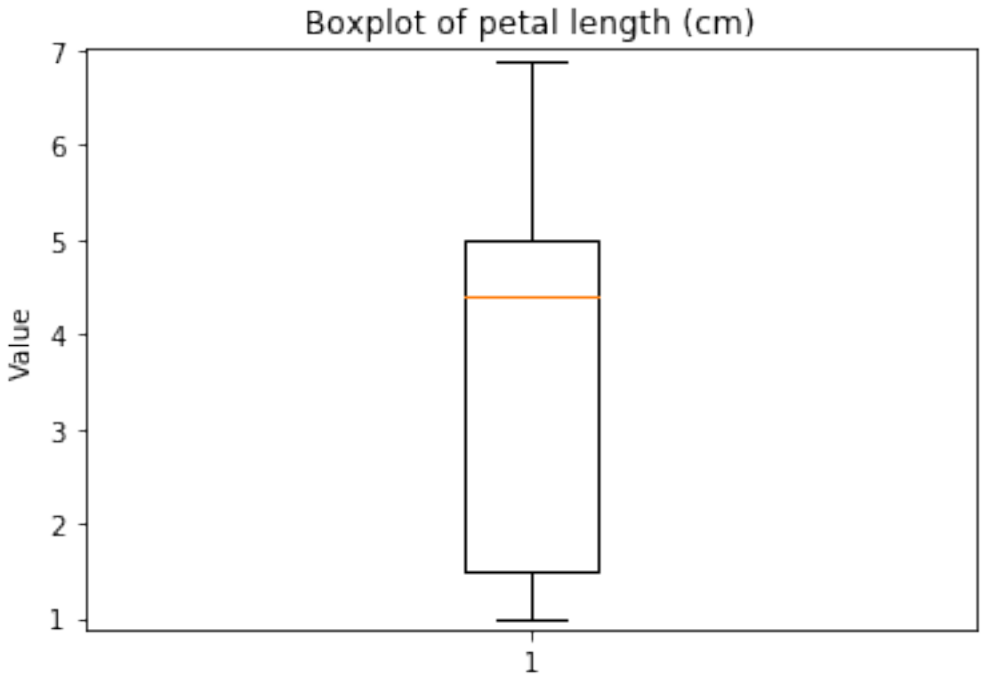
```
<IPython.core.display.HTML object>
```

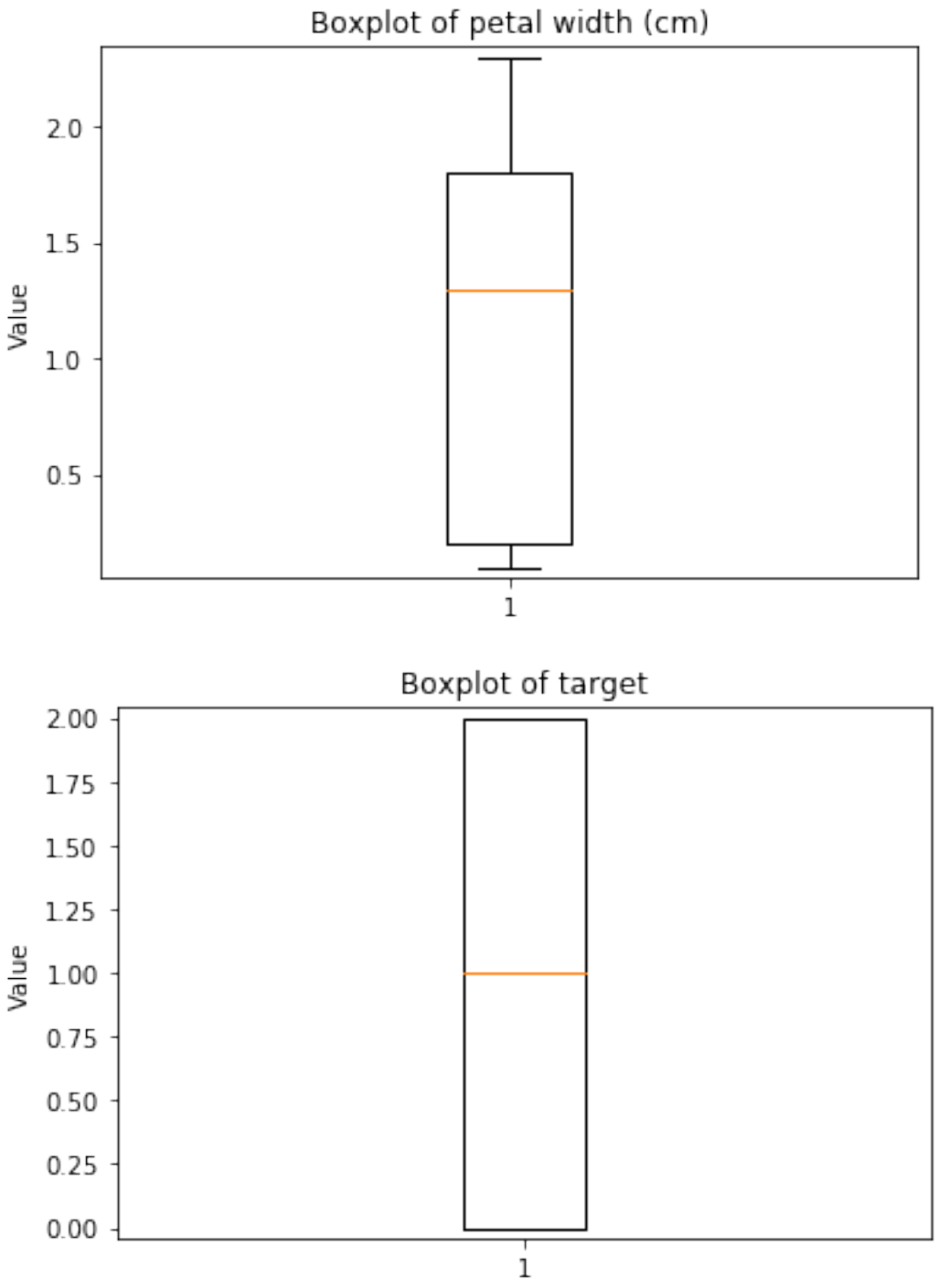
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

```
<IPython.core.display.HTML object>
```




Boxplot of sepal width (cm)
Value
4.0
3.5
3.0
2.5
1


Boxplot of petal length (cm)
Value
7
6
5
4
3
2
1
1



Some plots can combine transforms to one plot

You can add a name to the transform in order to add it to the legend. By default, the transform won't close the plots. So when you leave out close_plt=True in the call of (fit_)transform, plots of the next transform will be integrated in the plots of the previous transform. Do not forget to call close_plt=True on the last transform, otherwise all plots will remain open and will be plotted by jupyter again.

```
In [11]: p = ds.Pipeline()
         p.addPipe('read', ds.data.SampleData('iris'))
         p.addPipe('split', ds.transform.RandomTrainTestSplit(test_size=0.3), [("read", "df", "df")])
         p.addPipe('ecdf', ds.eda.ECDFPlots(), [("split", "df", "df")])
         p.fit_transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.T
         p.transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.TEST)
```

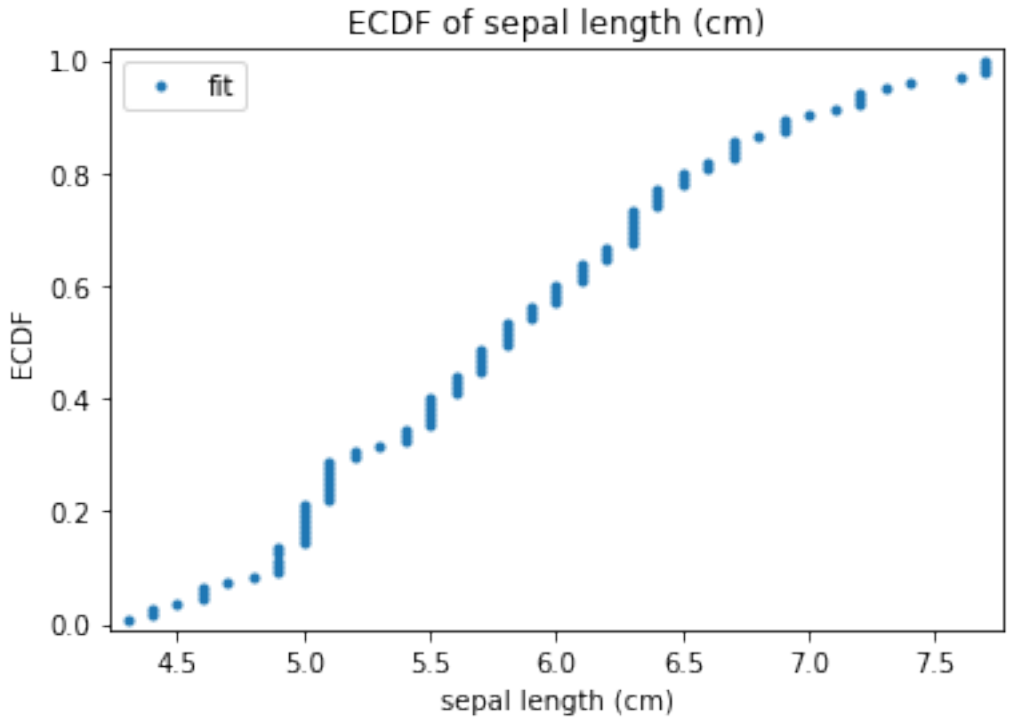
```
'Drawing diagram using blockdiag'
```

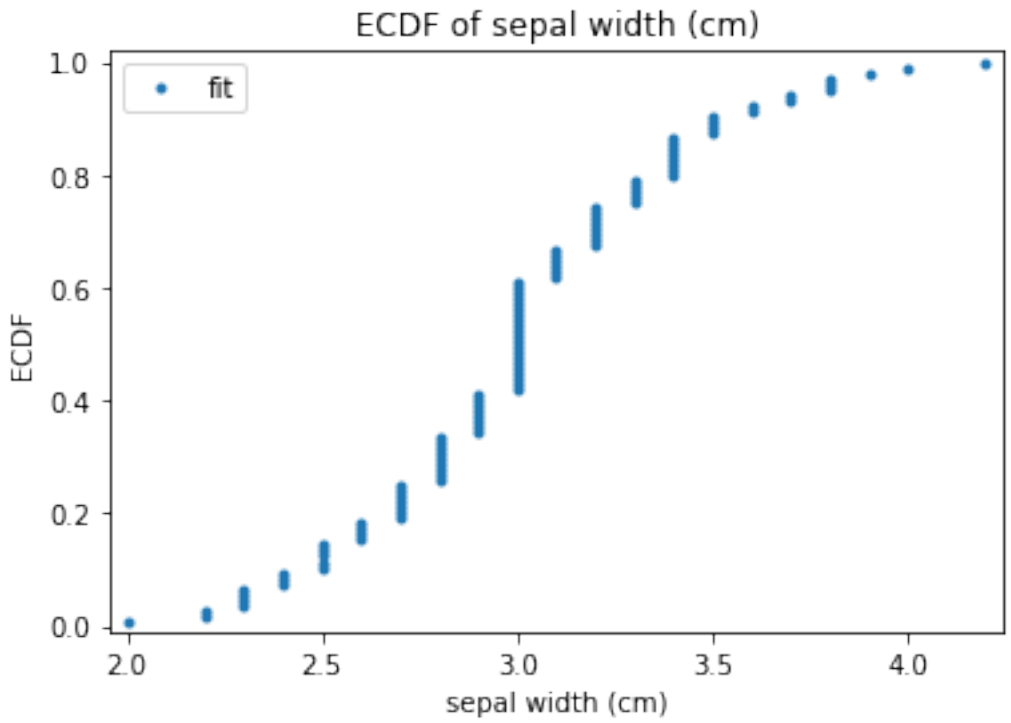
```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB06706EF0>
```

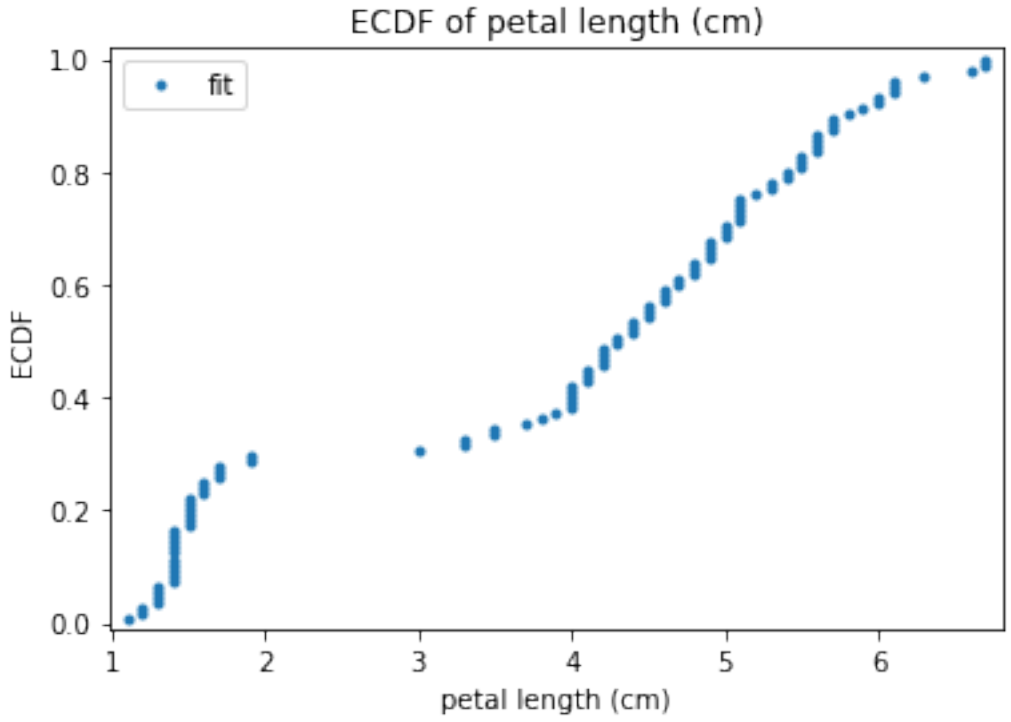
```
<IPython.core.display.HTML object>
```

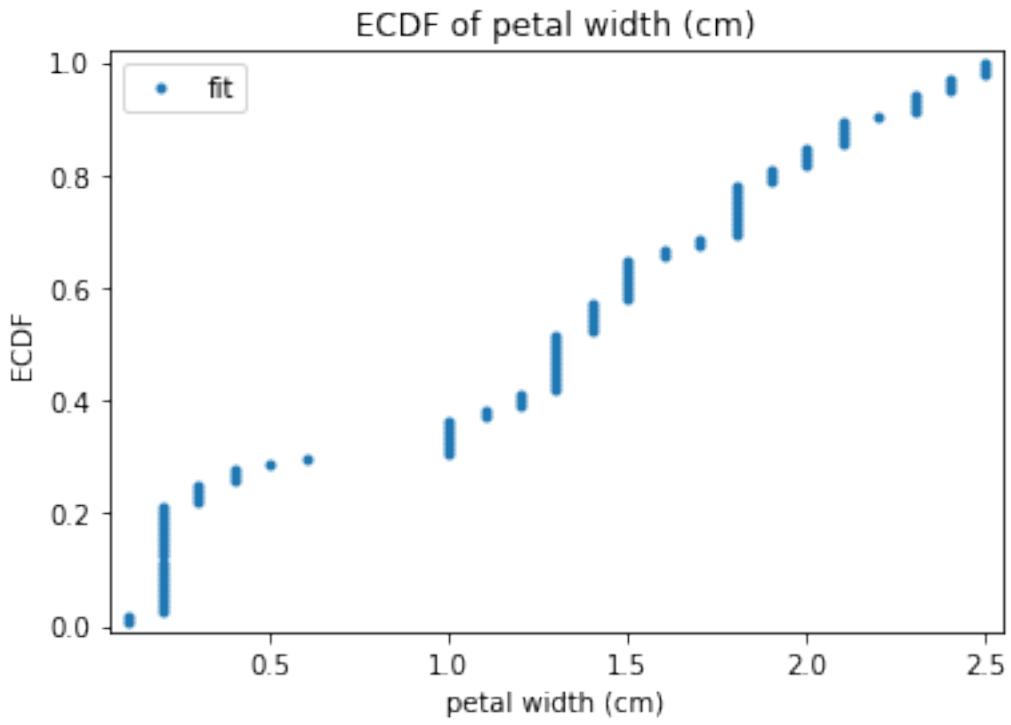
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

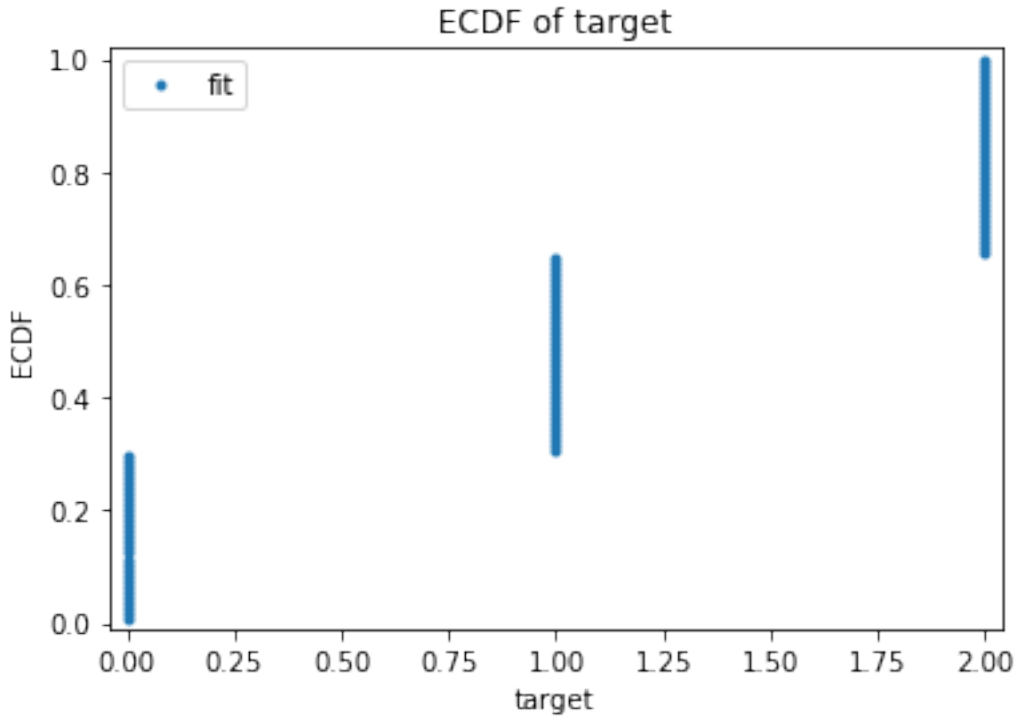
```
<IPython.core.display.HTML object>
```



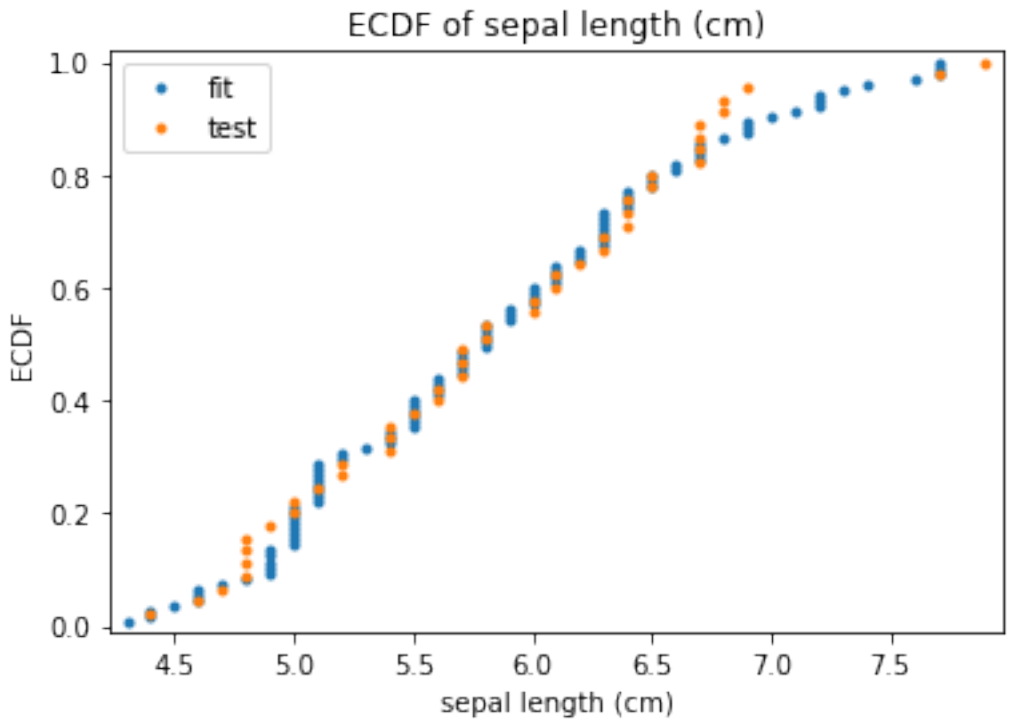

ECDF of sepal width (cm)
fit
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
2.0
2.5
3.0
3.5
4.0
sepal width (cm)

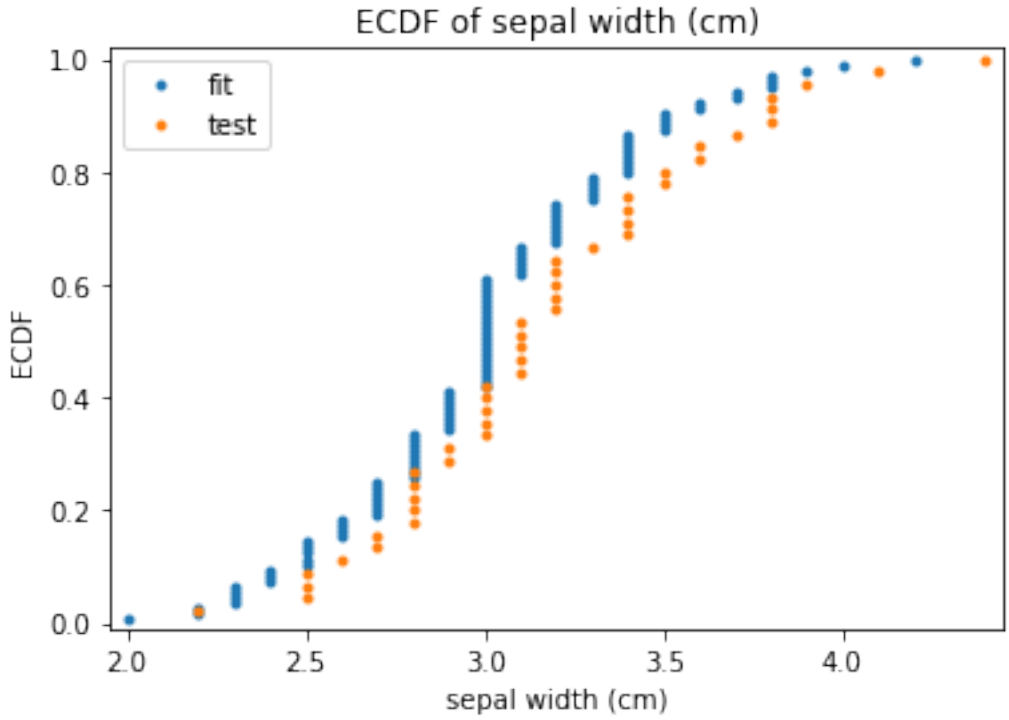

ECDF of petal length (cm)
fit
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
petal length (cm)

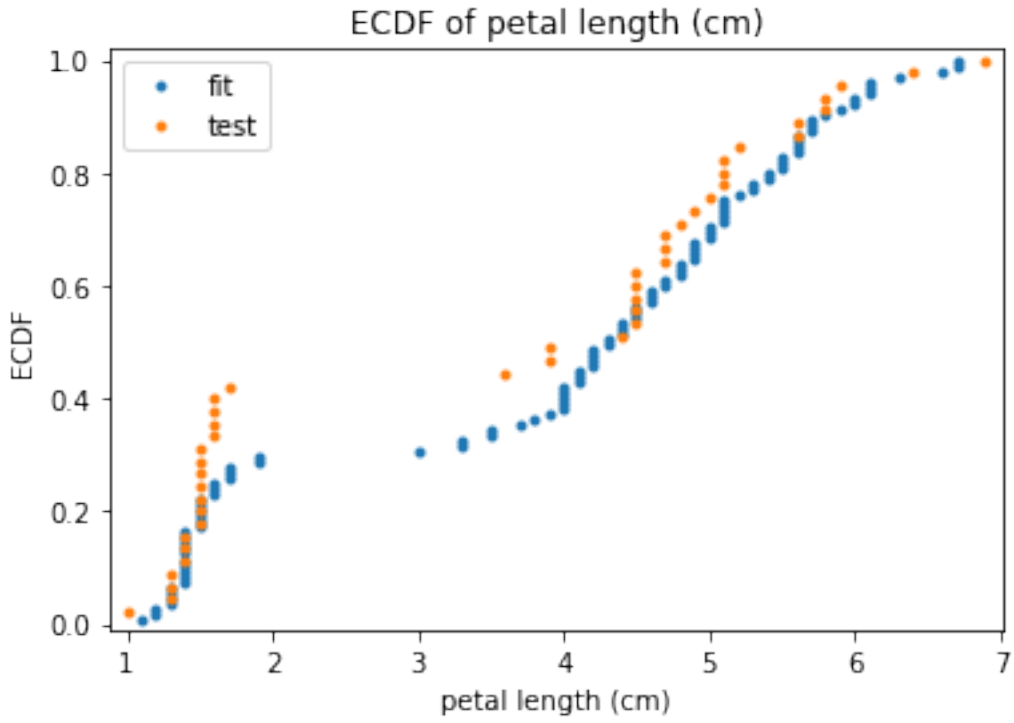


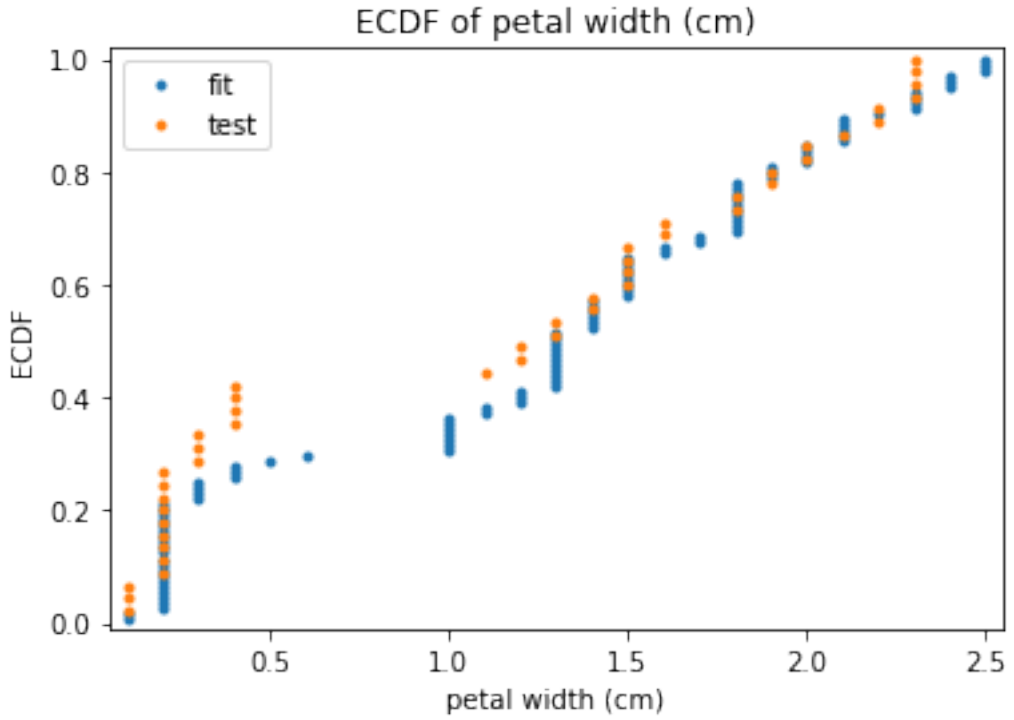


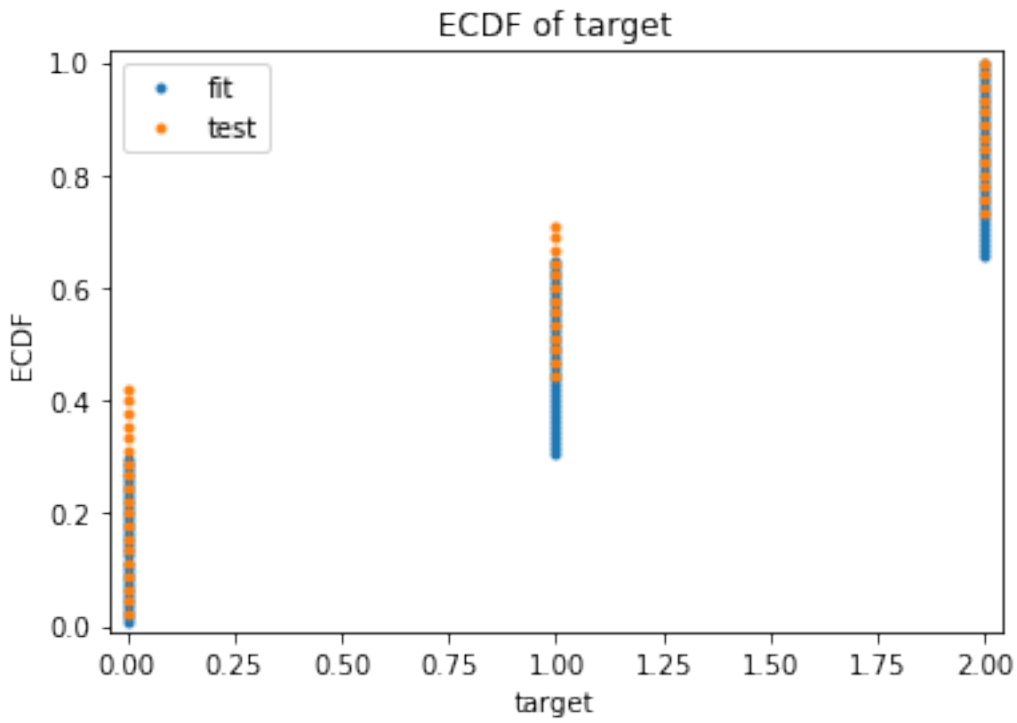

```
'Drawing diagram using blockdiag'
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB06193F60>
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
<IPython.core.display.HTML object>
```


ECDF of sepal length (cm)
fit
test
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
sepal length (cm)


ECDF of sepal width (cm)
fit
test
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
2.0
2.5
3.0
3.5
4.0
sepal width (cm)


ECDF of petal length (cm)
fit
test
ECDF
petal length (cm)
0.0
0.2
0.4
0.6
0.8
1.0
1
2
3
4
5
6
7


ECDF of petal width (cm)
fit
test
ECDF
petal width (cm)
0.0
0.2
0.4
0.6
0.8
1.0
0.5
1.0
1.5
2.0
2.5




```
In [12]: p = ds.Pipeline()
         p.addPipe('read', ds.data.SampleData('iris'))
         p.addPipe('split', ds.transform.RandomTrainTestSplit(test_size=0.3), [("read", "df", "df")])
         p.addPipe('scatter', ds.eda.ScatterPlots(), [("split", "df", "df")])
         p.fit_transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.T
         p.transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.TEST}
```

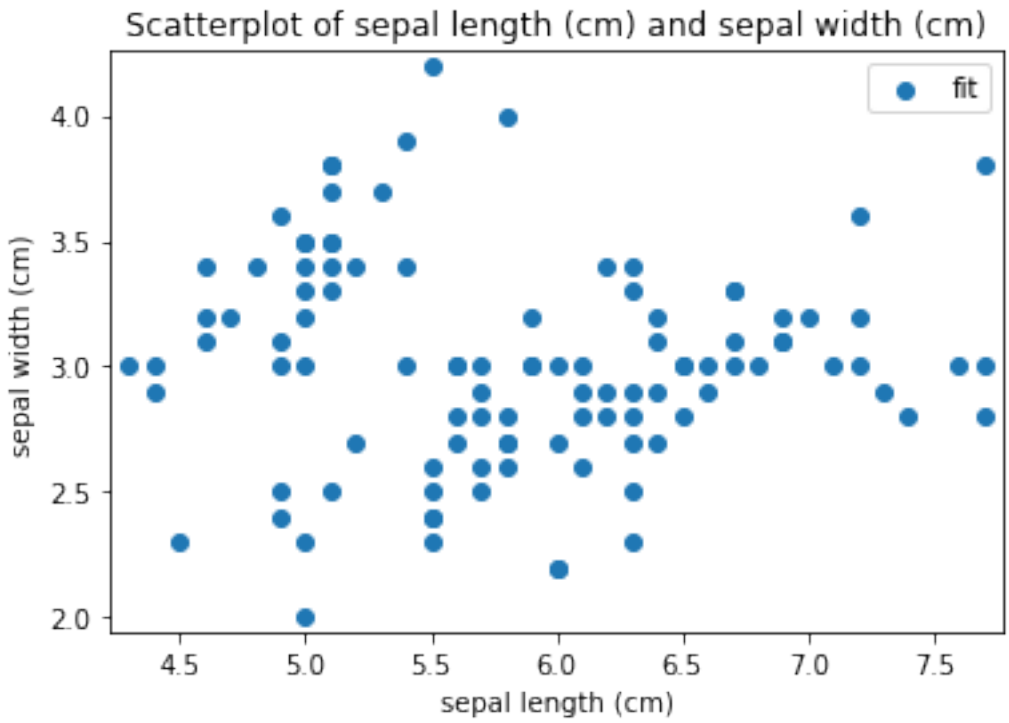
```
'Drawing diagram using blockdiag'
```

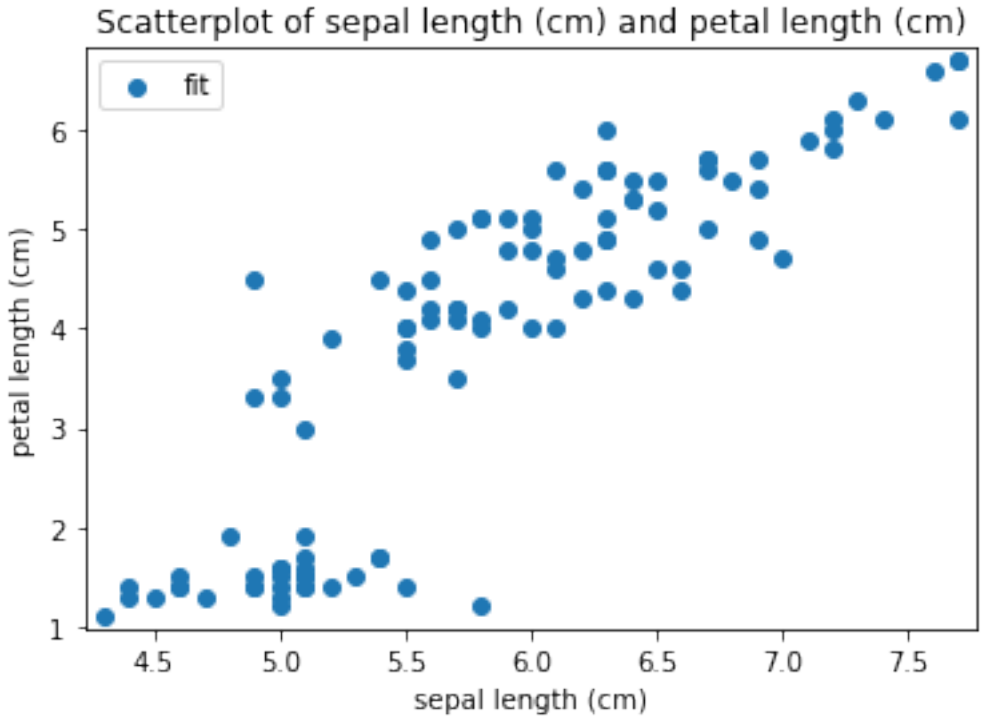
```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB06361E10>
```

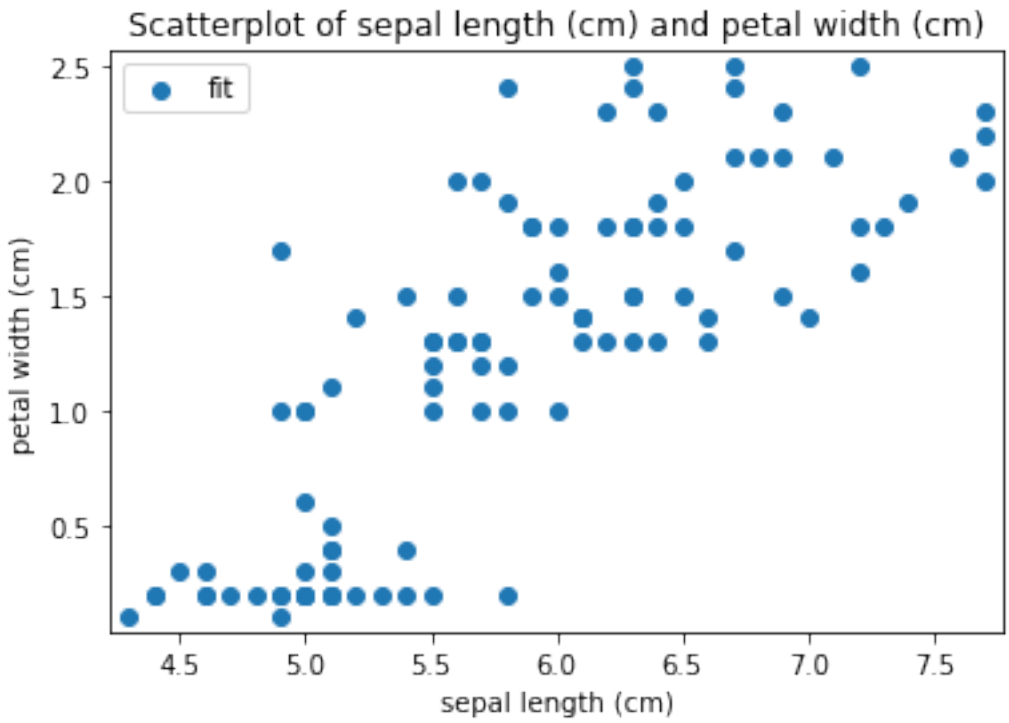
```
<IPython.core.display.HTML object>
```

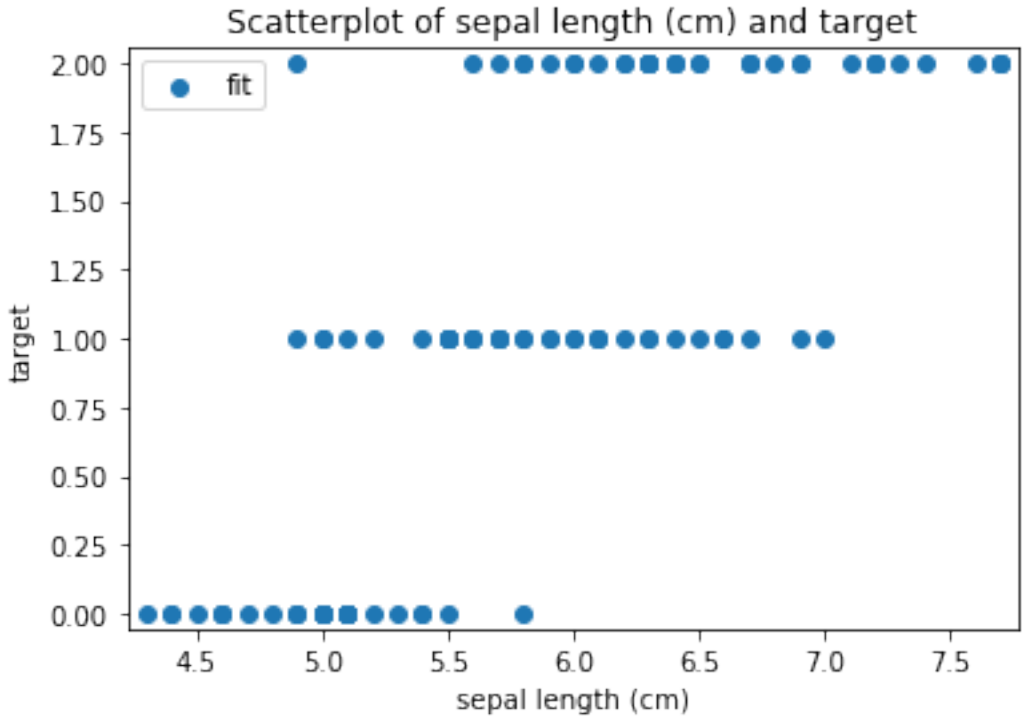
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

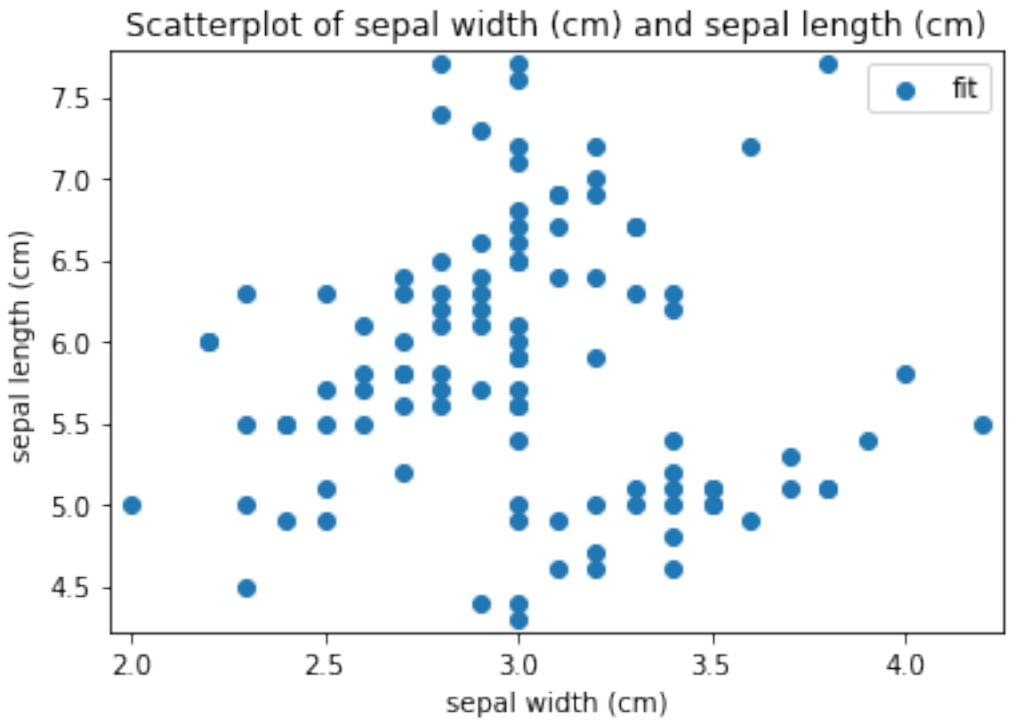
```
<IPython.core.display.HTML object>
```

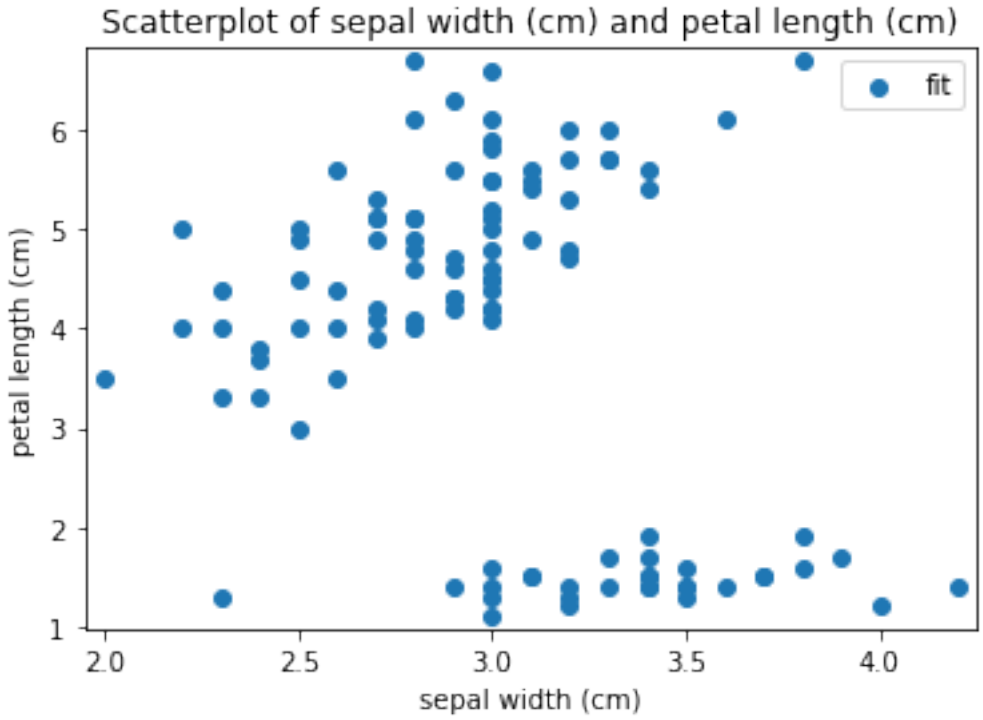

Scatterplot of sepal length (cm) and sepal width (cm)
fit
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5

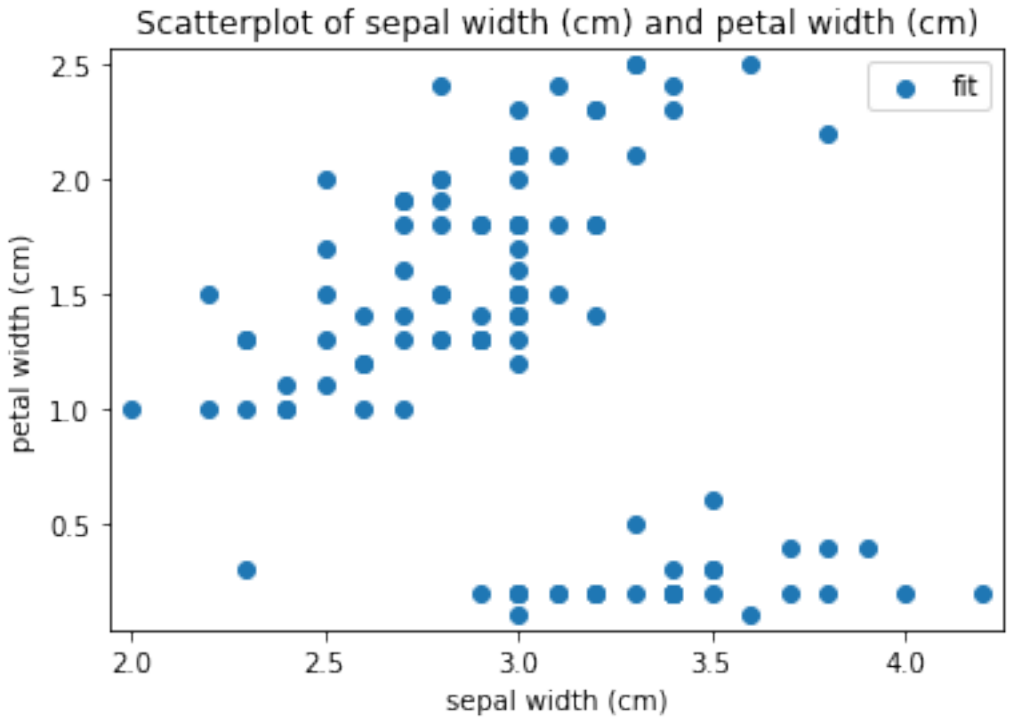

Scatterplot of sepal length (cm) and petal length (cm)
fit
petal length (cm)
sepal length (cm)
6
5
4
3
2
1
4.5
5.0
5.5
6.0
6.5
7.0
7.5

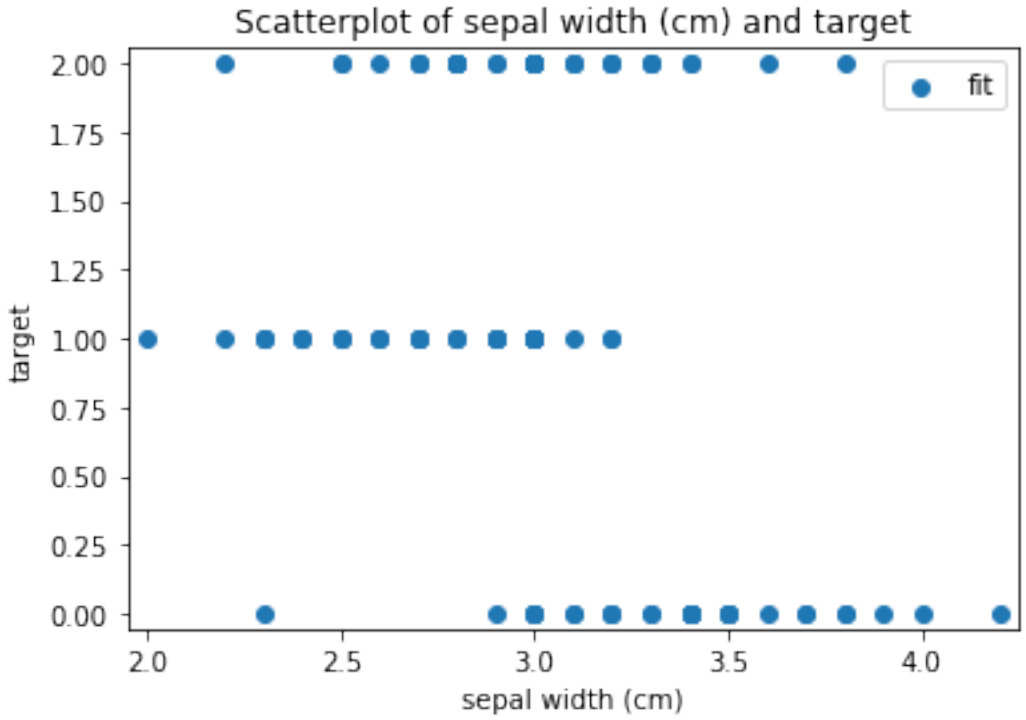

Scatterplot of sepal length (cm) and petal width (cm)
fit
petal width (cm)
sepal length (cm)
2.5
2.0
1.5
1.0
0.5
4.5
5.0
5.5
6.0
6.5
7.0
7.5

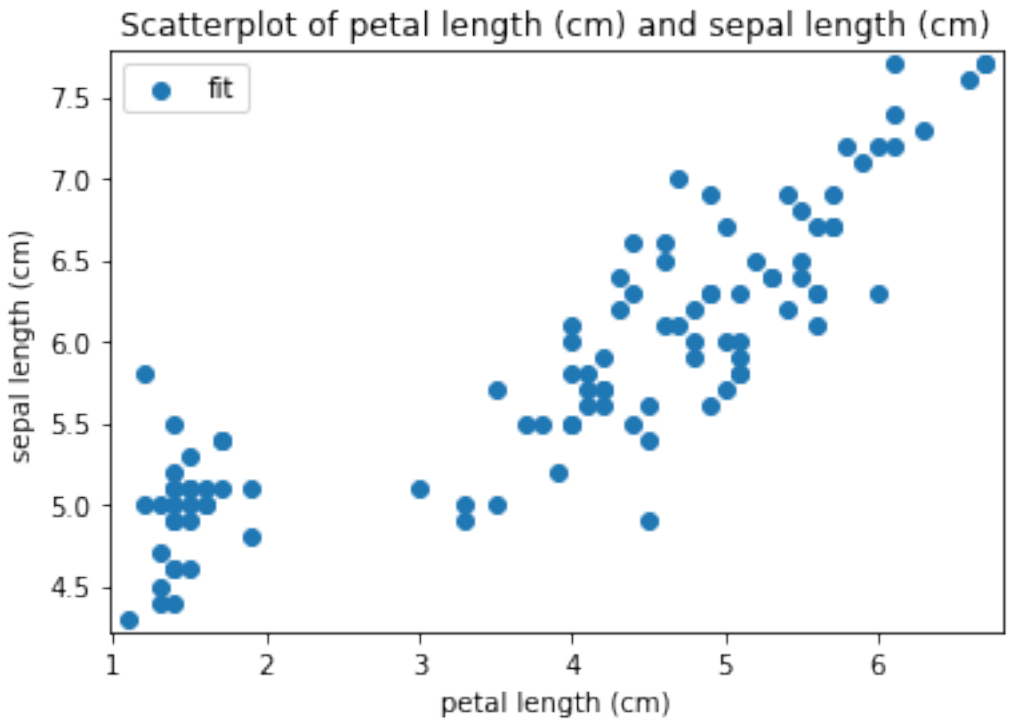

Scatterplot of sepal length (cm) and target
fit
target
sepal length (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4.5
5.0
5.5
6.0
6.5
7.0
7.5

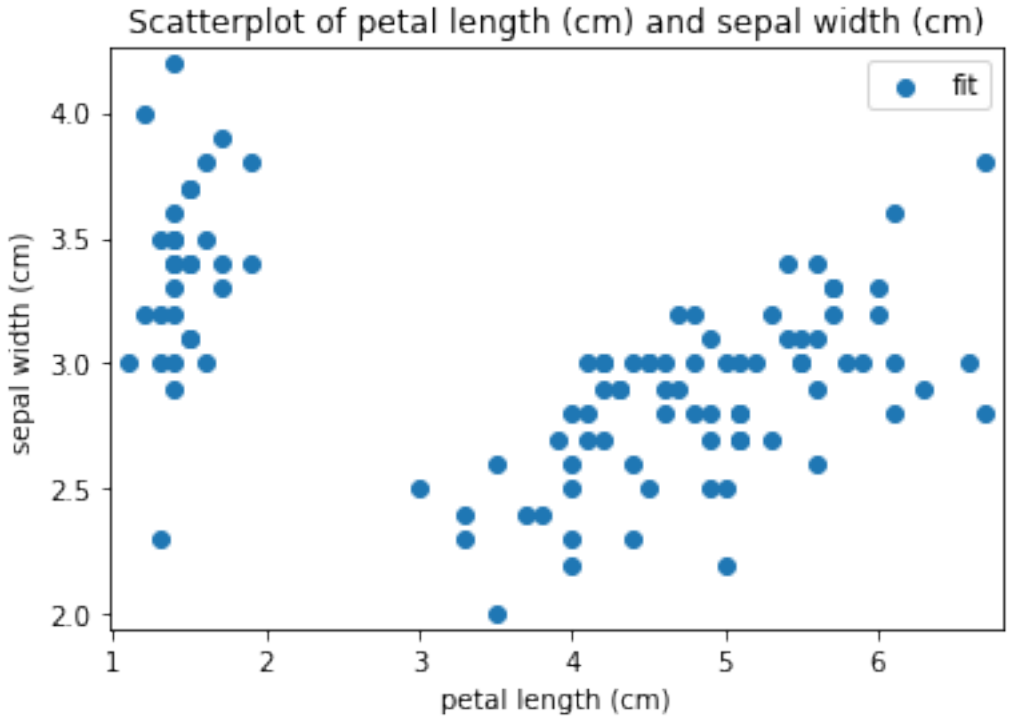

Scatterplot of sepal width (cm) and sepal length (cm)
fit
sepal length (cm)
sepal width (cm)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
2.0
2.5
3.0
3.5
4.0

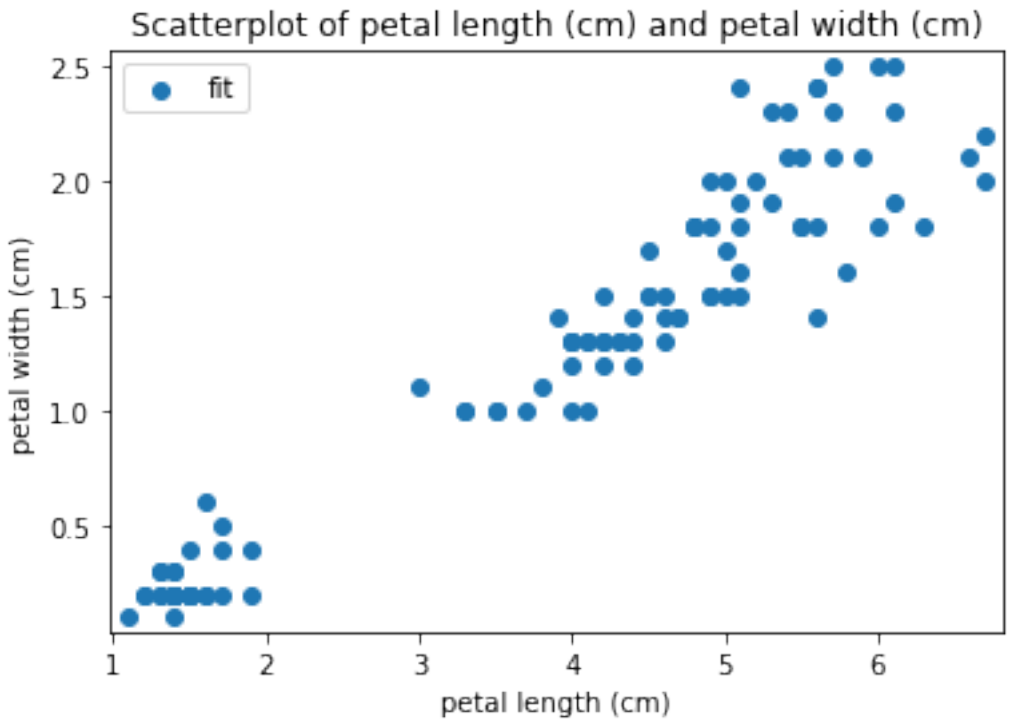

Scatterplot of sepal width (cm) and petal length (cm)
fit
petal length (cm)
sepal width (cm)
6
5
4
3
2
1
2.0
2.5
3.0
3.5
4.0

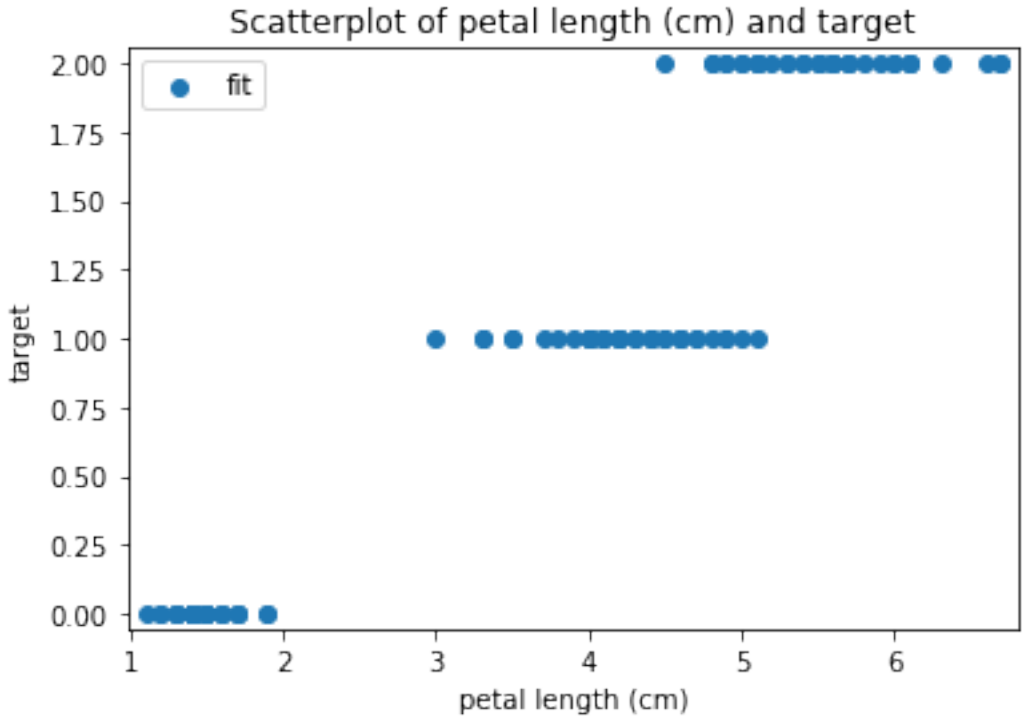

Scatterplot of sepal width (cm) and petal width (cm)
fit
petal width (cm)
sepal width (cm)
2.5
2.0
1.5
1.0
0.5
2.0
2.5
3.0
3.5
4.0

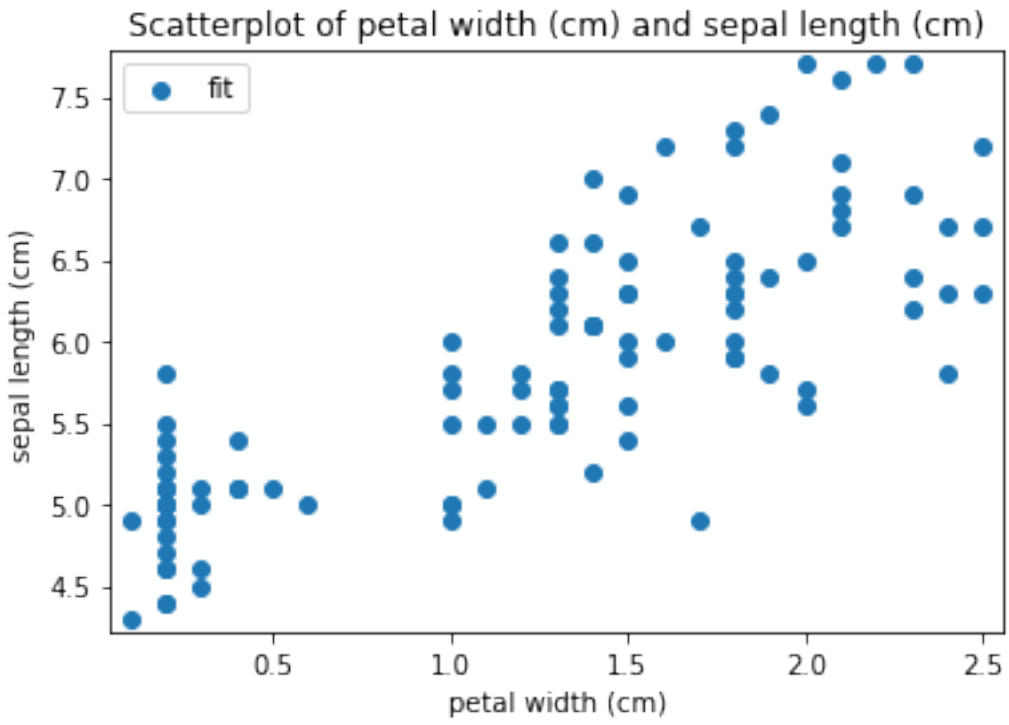

Scatterplot of sepal width (cm) and target
fit
target
sepal width (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2.0
2.5
3.0
3.5
4.0

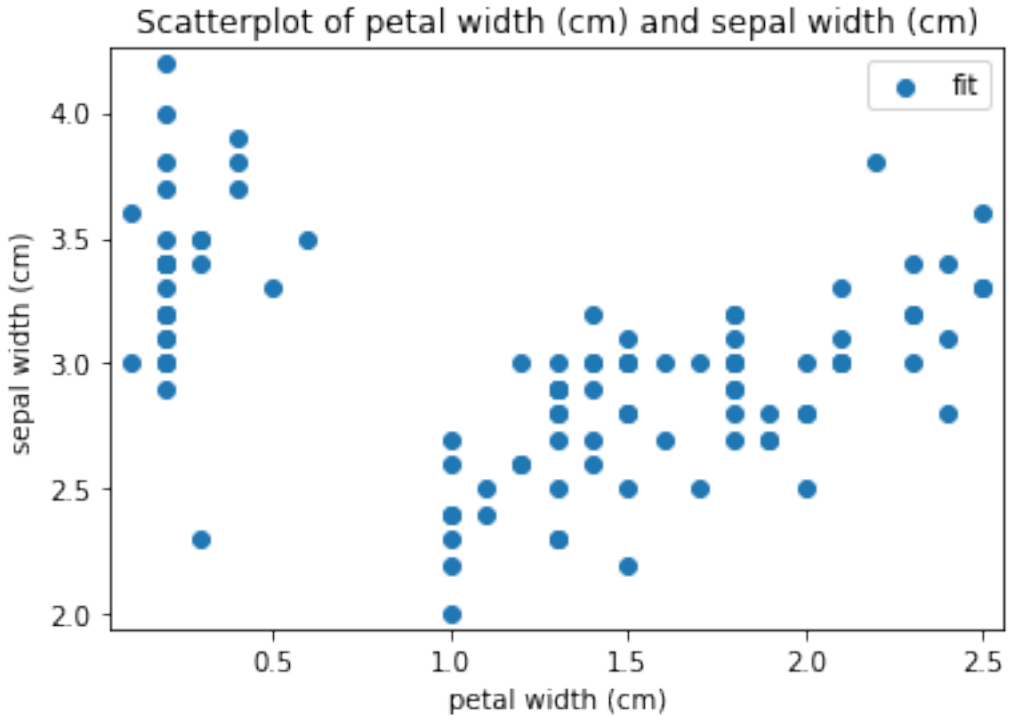

Scatterplot of petal length (cm) and sepal length (cm)
fit
sepal length (cm)
petal length (cm)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
1
2
3
4
5
6

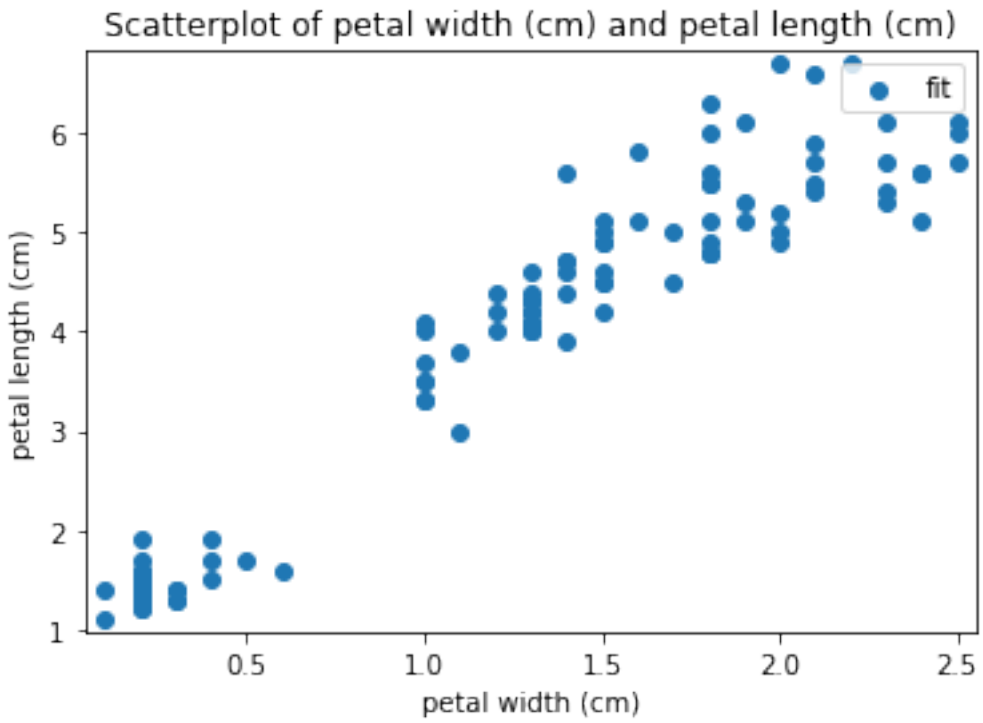

Scatterplot of petal length (cm) and sepal width (cm)
fit
sepal width (cm)
petal length (cm)
4.0
3.5
3.0
2.5
2.0
1
2
3
4
5
6

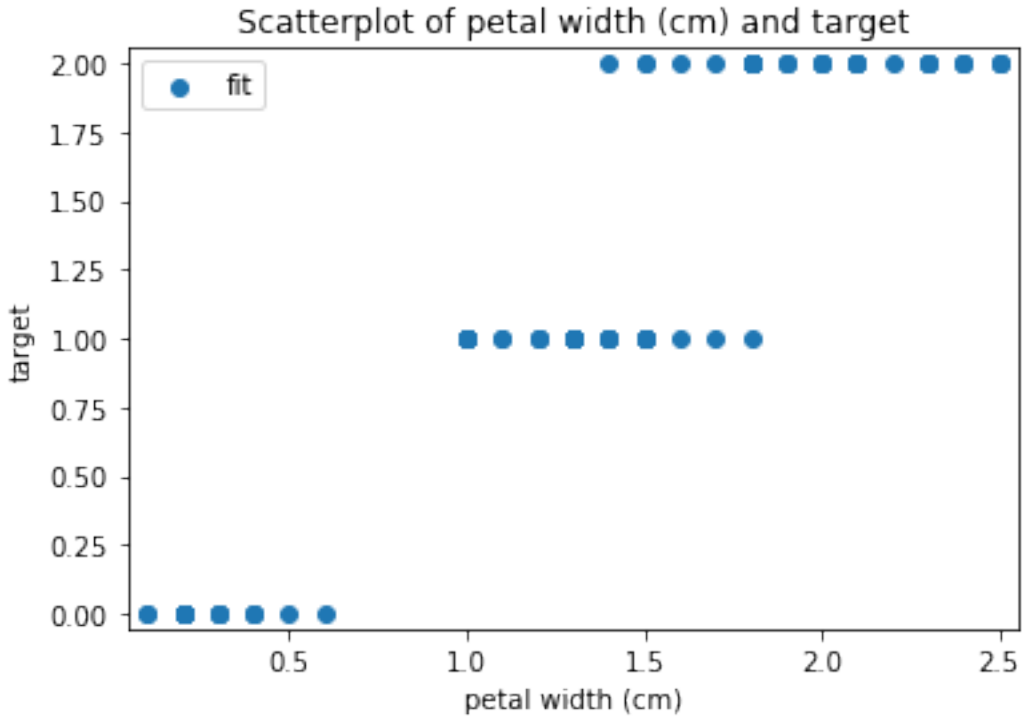

Scatterplot of petal length (cm) and petal width (cm)
fit
petal width (cm)
petal length (cm)
2.5
2.0
1.5
1.0
0.5
1
2
3
4
5
6

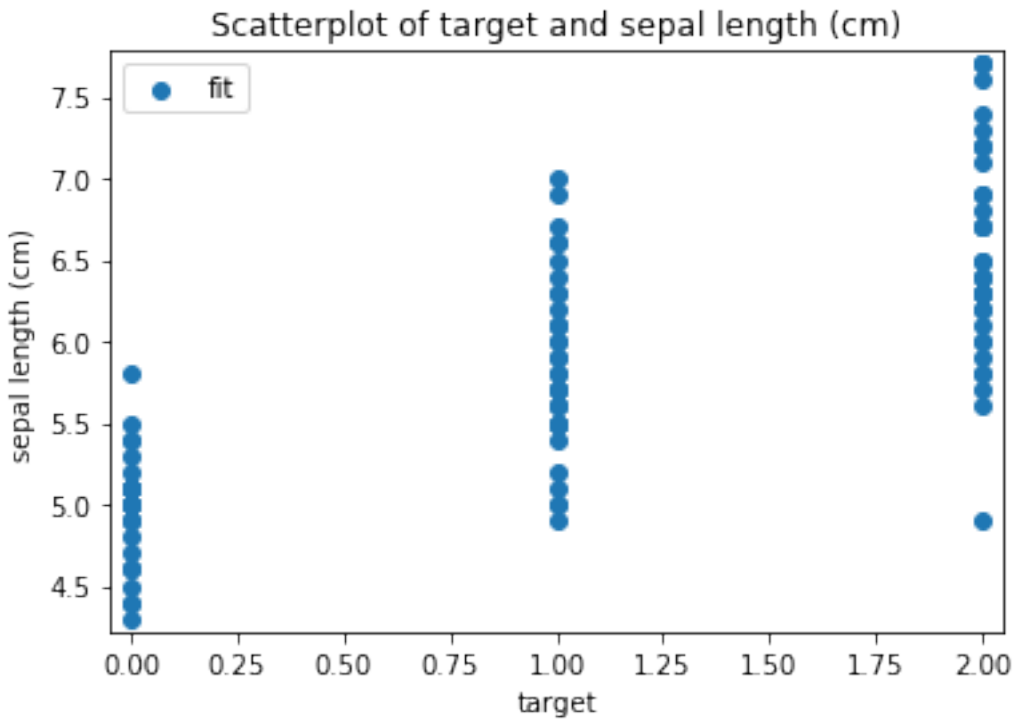

Scatterplot of petal length (cm) and target
fit
target
petal length (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6

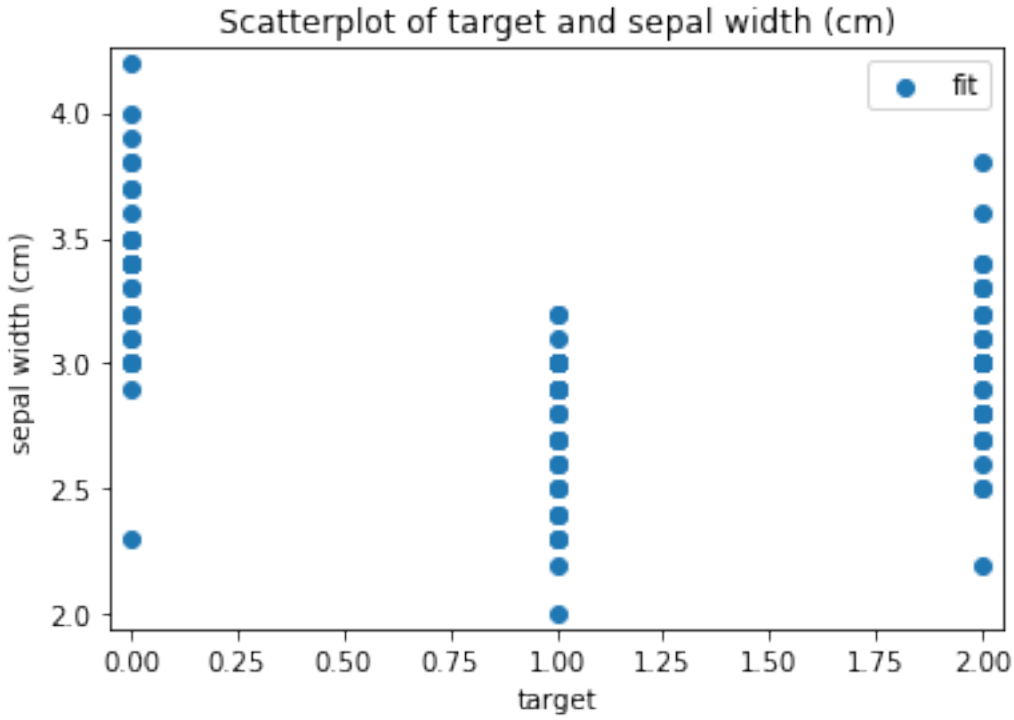

Scatterplot of petal width (cm) and sepal length (cm)
fit
sepal length (cm)
petal width (cm)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.5
1.0
1.5
2.0
2.5

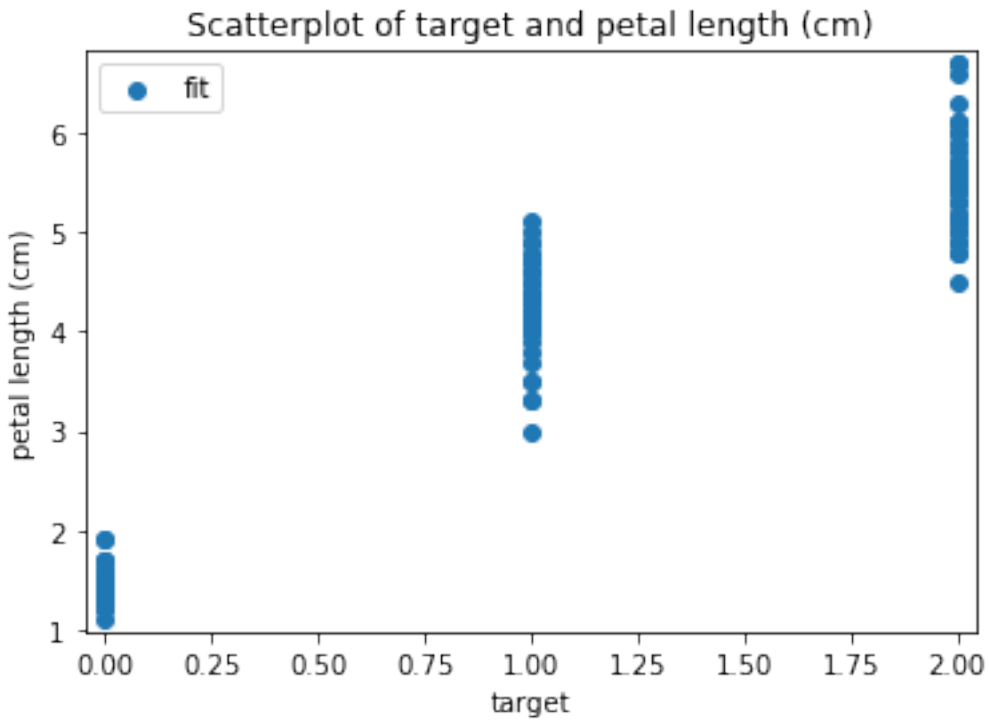

Scatterplot of petal width (cm) and sepal width (cm)
fit
sepal width (cm)
petal width (cm)
4.0
3.5
3.0
2.5
2.0
0.5
1.0
1.5
2.0
2.5

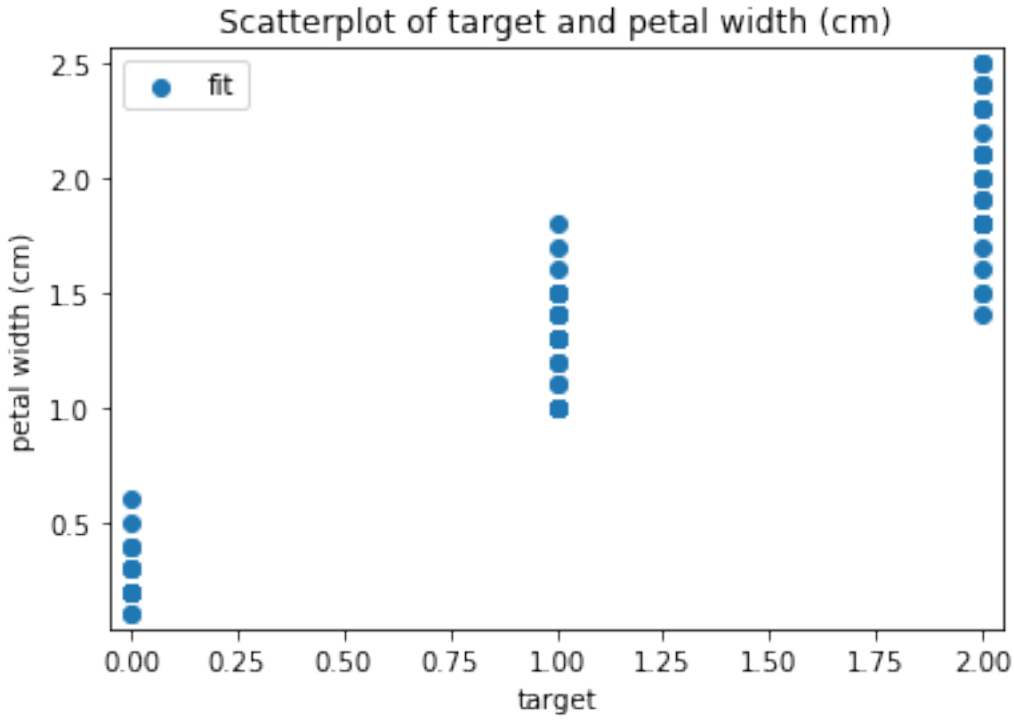

Scatterplot of petal width (cm) and petal length (cm)
fit
petal length (cm)
petal width (cm)
1
2
3
4
5
6
0.5
1.0
1.5
2.0
2.5


Scatterplot of petal width (cm) and target
fit
target
petal width (cm)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.5
1.0
1.5
2.0
2.5

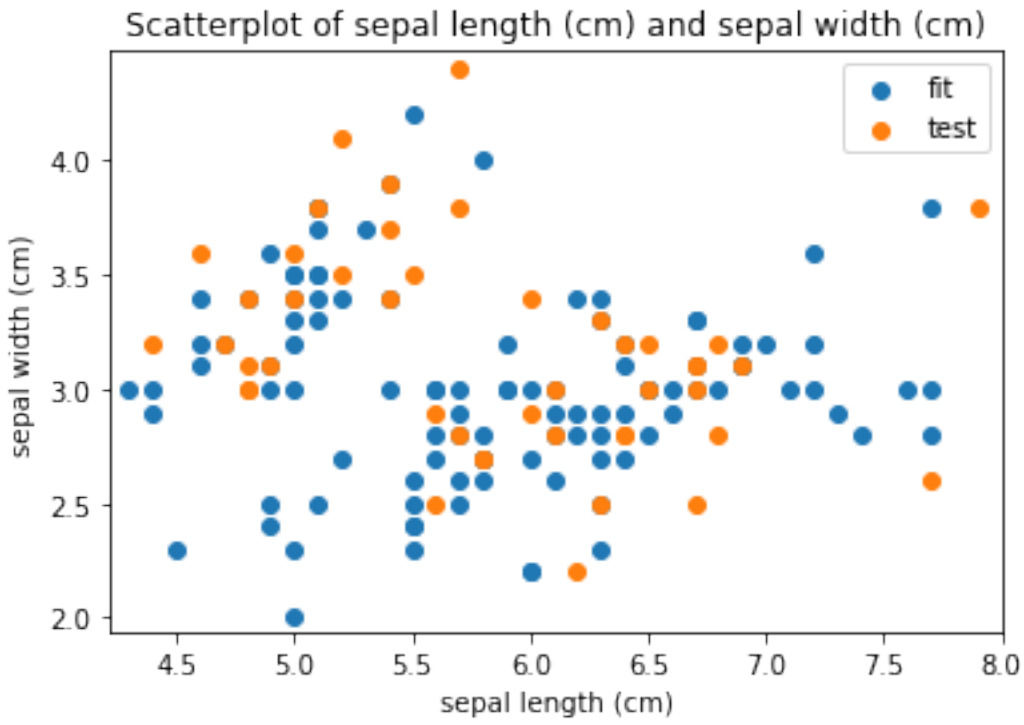

Scatterplot of target and sepal length (cm)
fit
sepal length (cm)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
target

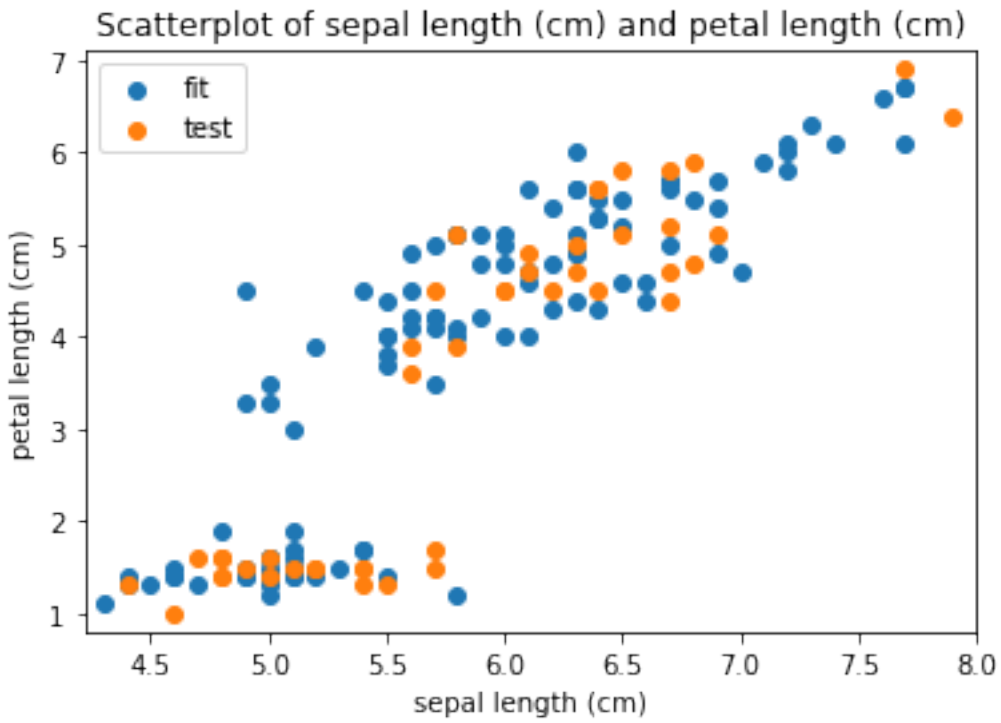

Scatterplot of target and sepal width (cm)
fit
sepal width (cm)
4.0
3.5
3.0
2.5
2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
target

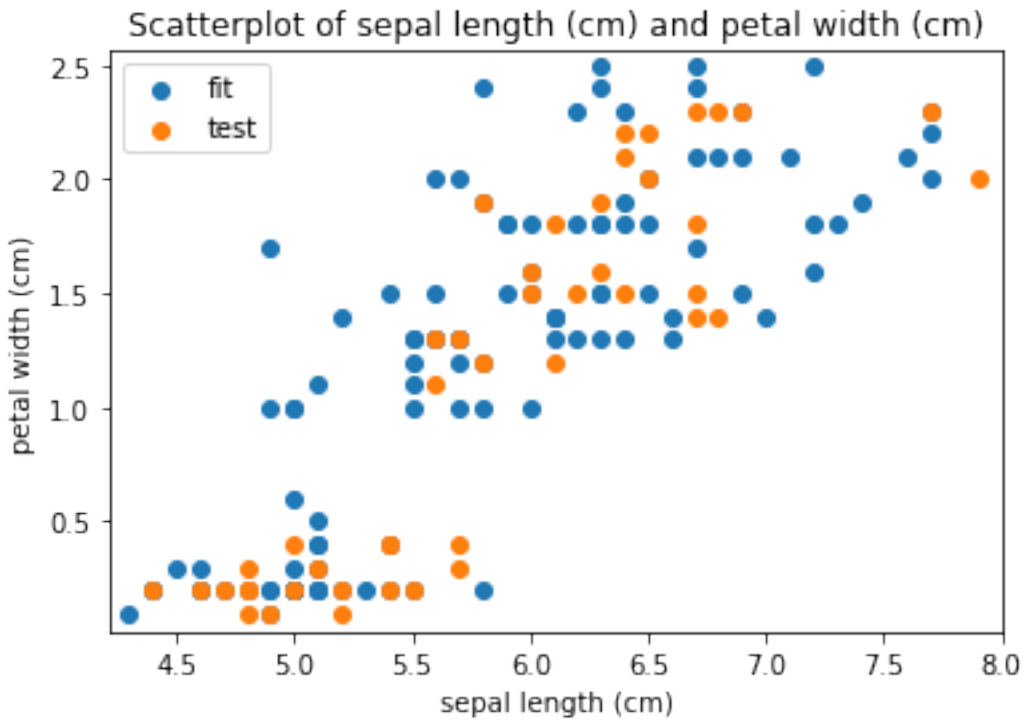


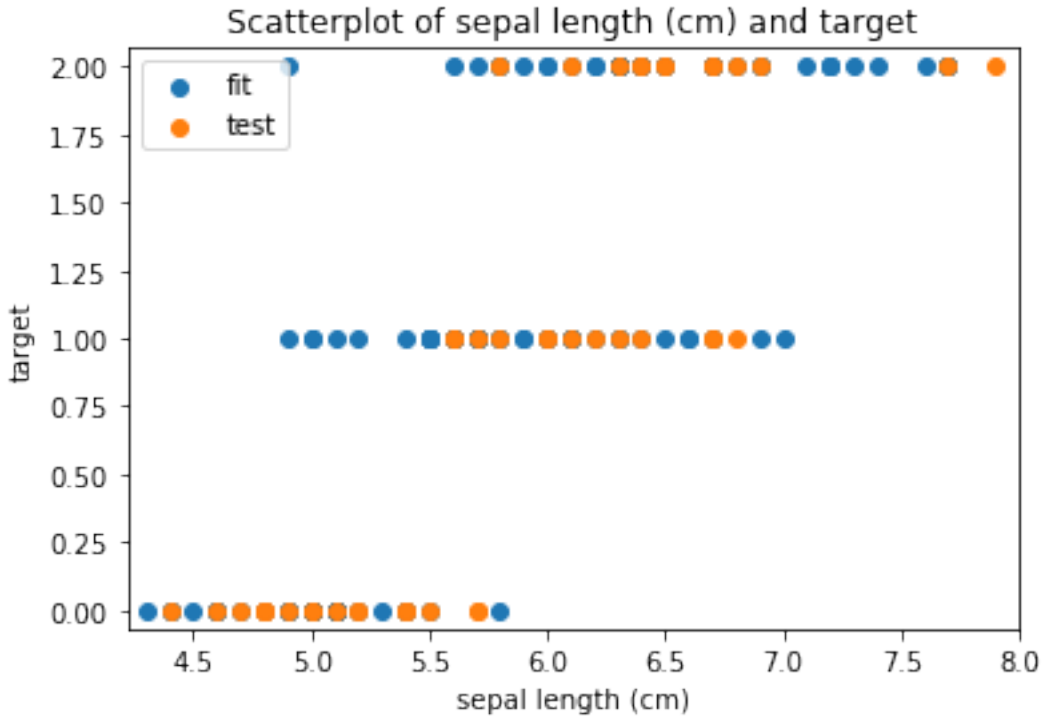


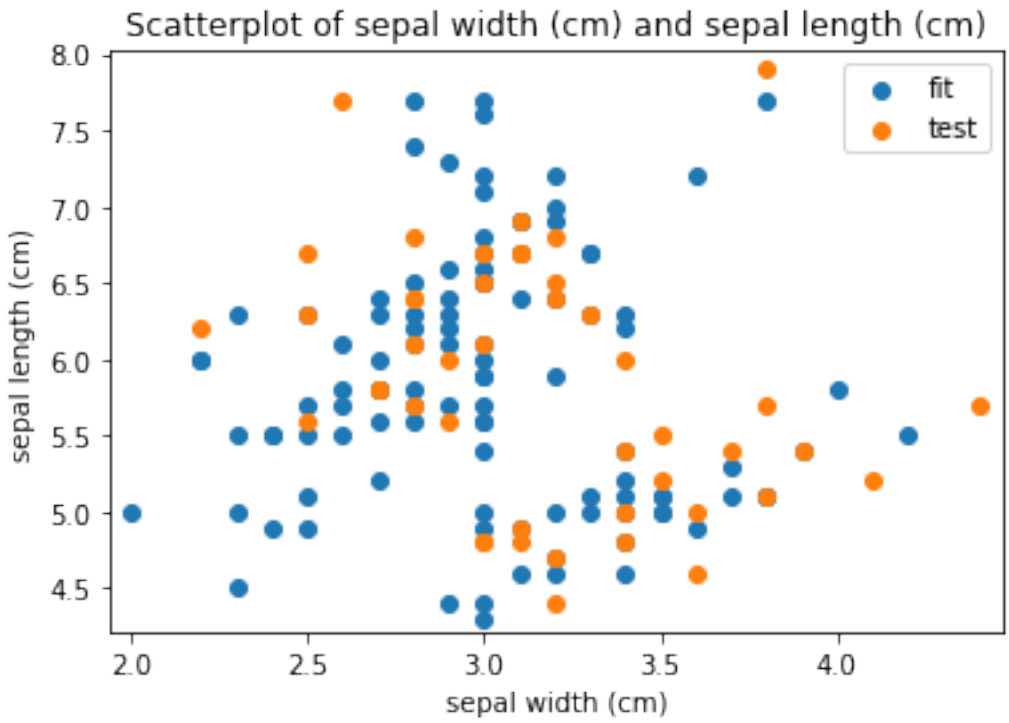

```
'Drawing diagram using blockdiag'
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB06420710>
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
<IPython.core.display.HTML object>
```

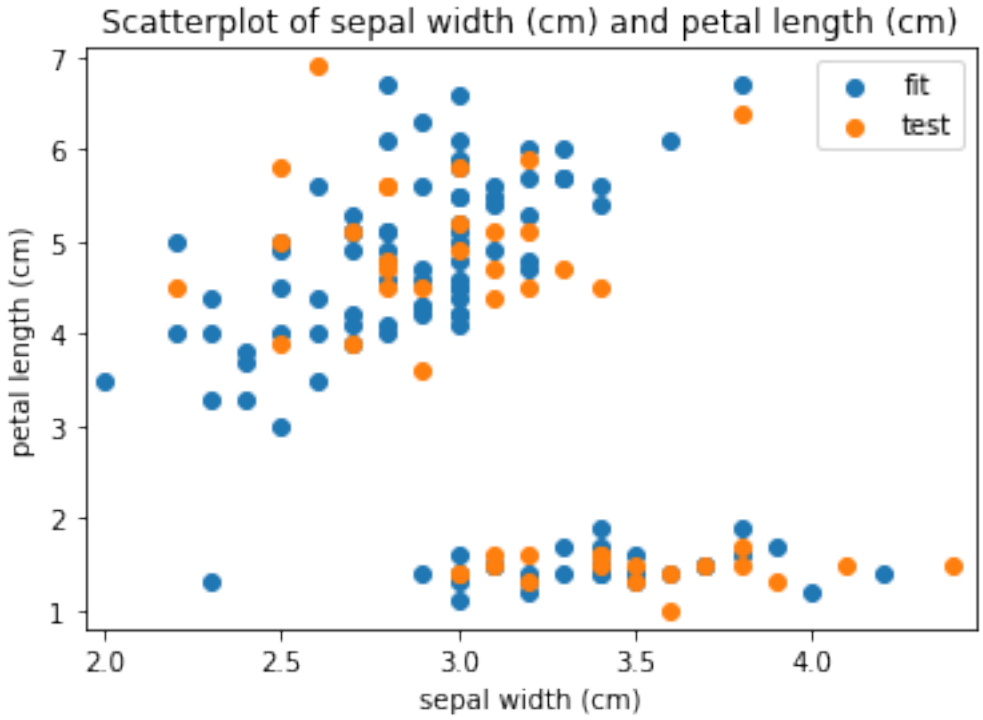

Scatterplot of sepal length (cm) and sepal width (cm)
fit
test
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

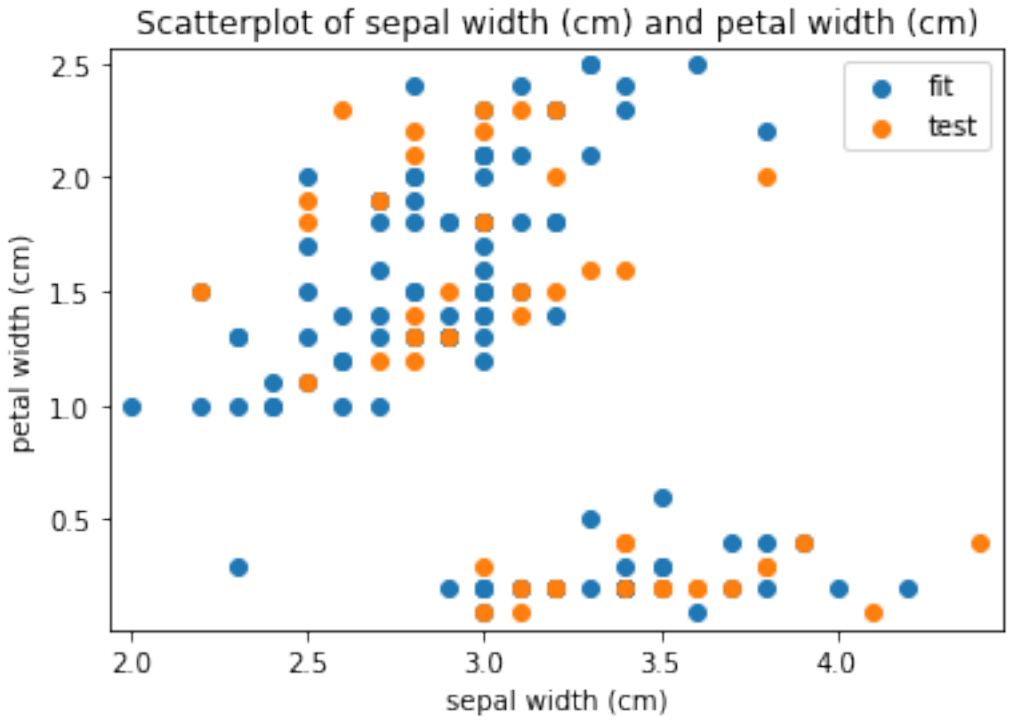

Scatterplot of sepal length (cm) and petal length (cm)
fit
test
petal length (cm)
sepal length (cm)
7
6
5
4
3
2
1
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

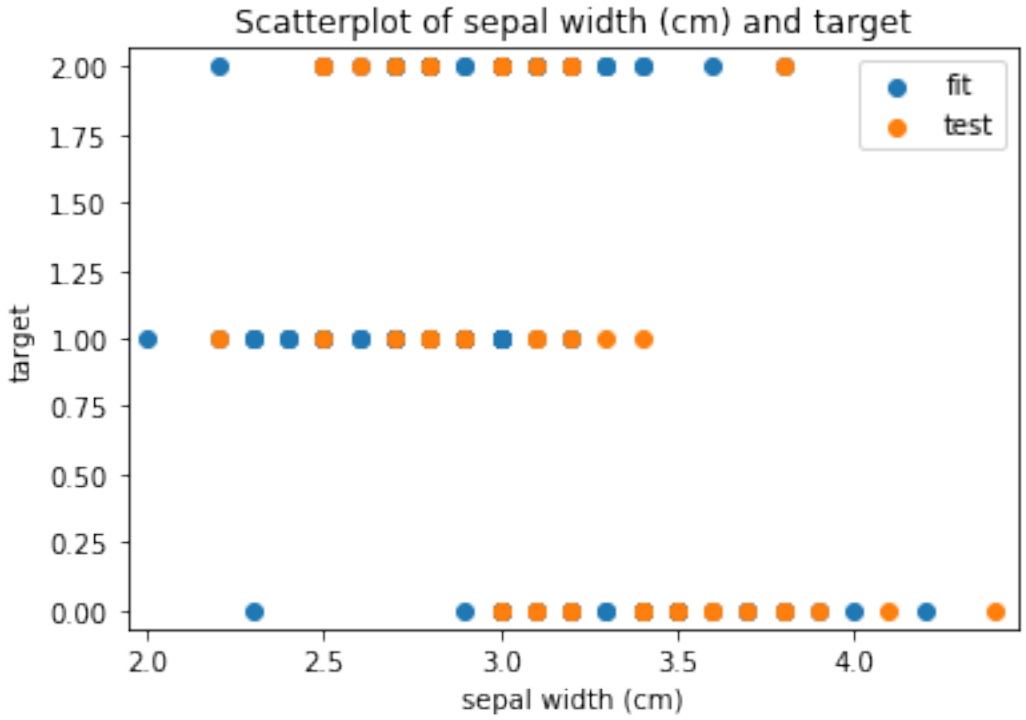

Scatterplot of sepal length (cm) and petal width (cm)
fit
test
petal width (cm)
sepal length (cm)
2.5
2.0
1.5
1.0
0.5
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

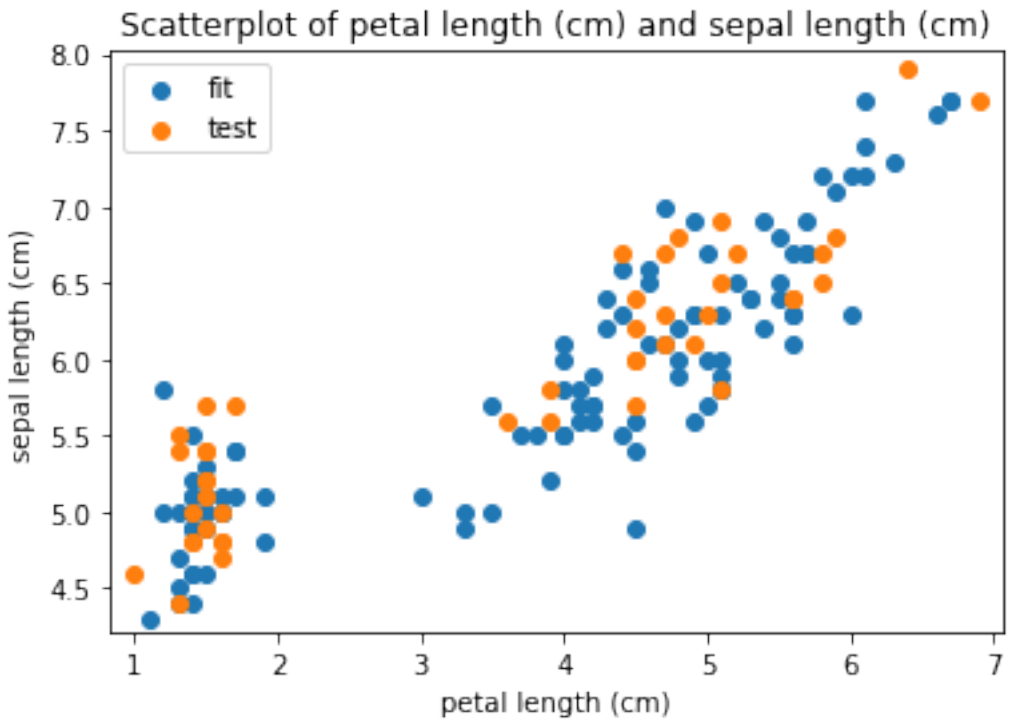

Scatterplot of sepal length (cm) and target
fit
test
target
sepal length (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

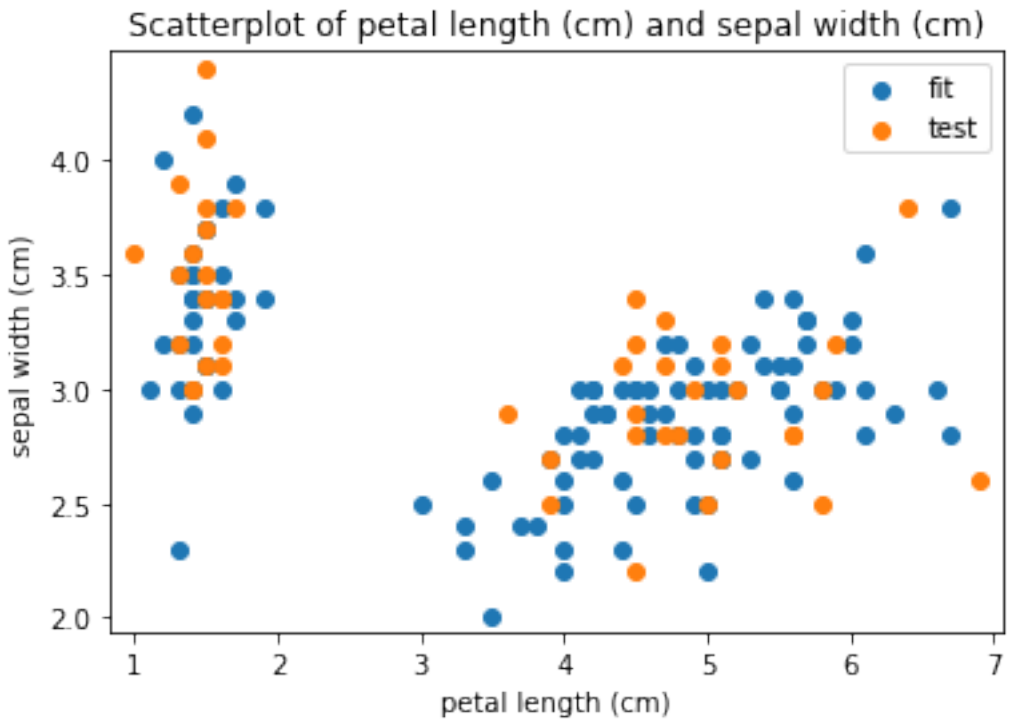

Scatterplot of sepal width (cm) and sepal length (cm)
fit
test
sepal length (cm)
sepal width (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
2.0
2.5
3.0
3.5
4.0

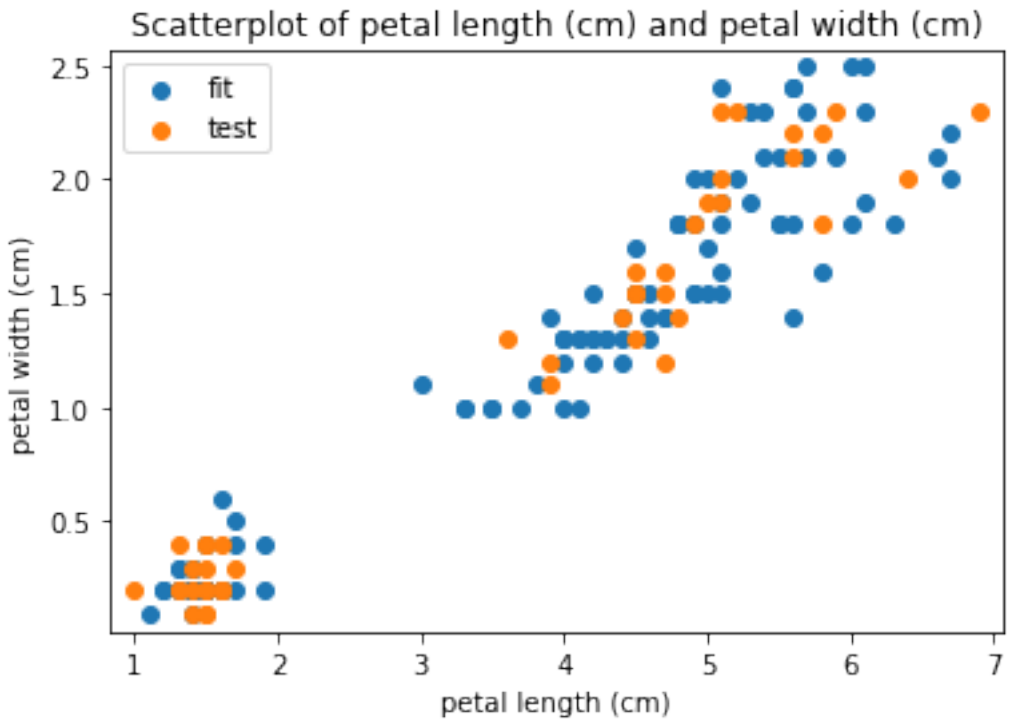

Scatterplot of sepal width (cm) and petal length (cm)
fit
test
petal length (cm)
sepal width (cm)
7
6
5
4
3
2
1
2.0
2.5
3.0
3.5
4.0

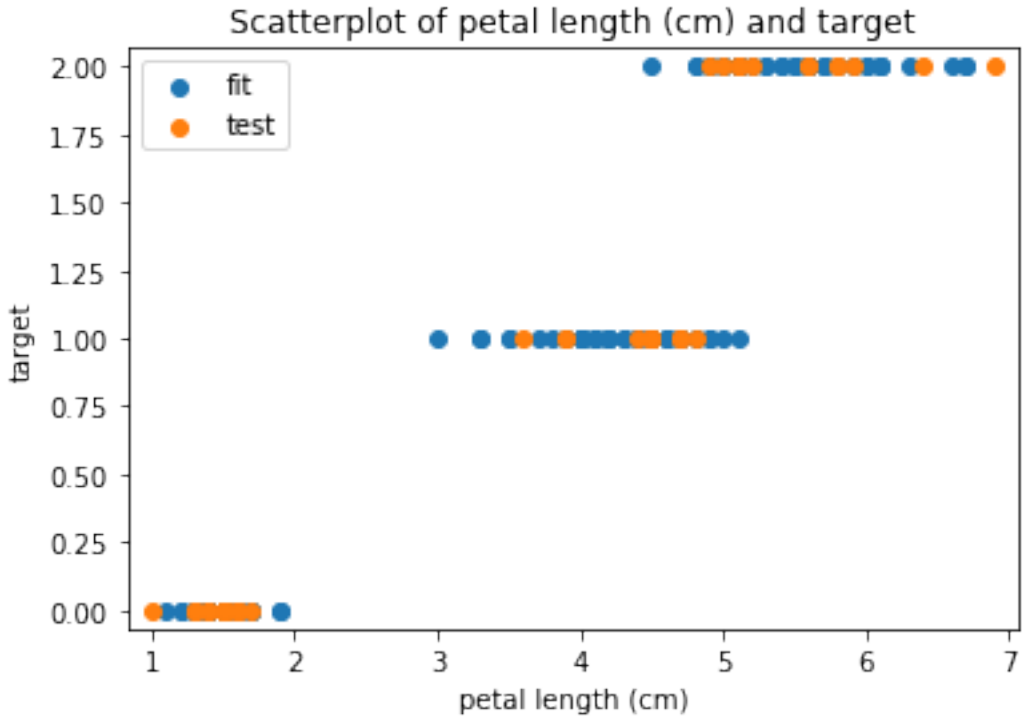

Scatterplot of sepal width (cm) and petal width (cm)
fit
test
petal width (cm)
sepal width (cm)
2.5
2.0
1.5
1.0
0.5
2.0
2.5
3.0
3.5
4.0

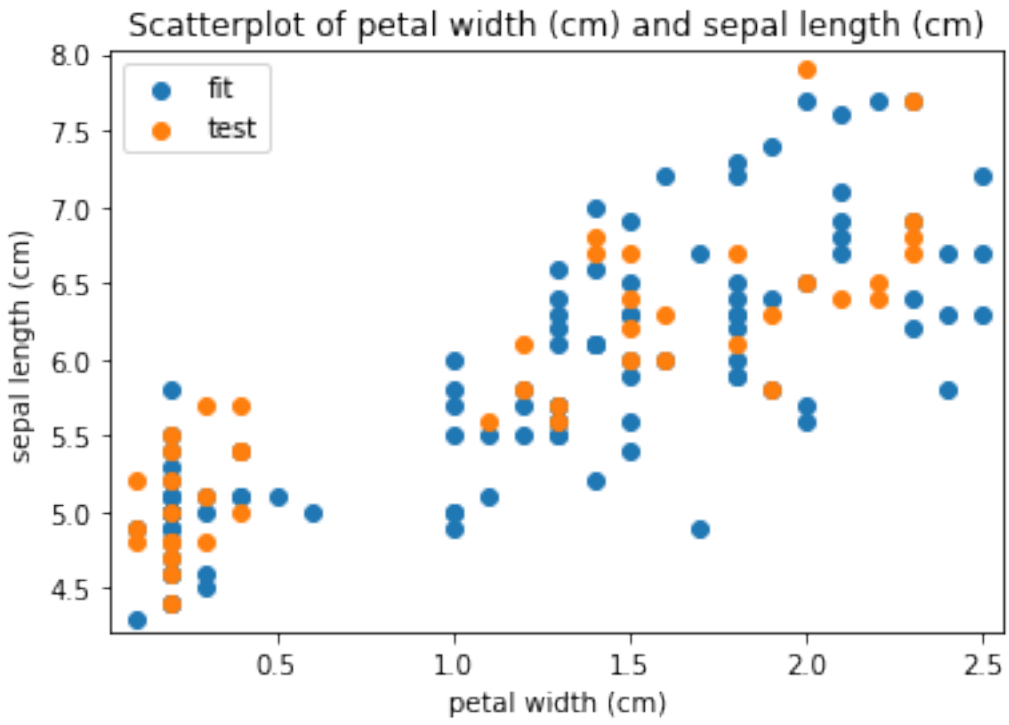

Scatterplot of sepal width (cm) and target
fit
test
target
sepal width (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2.0
2.5
3.0
3.5
4.0

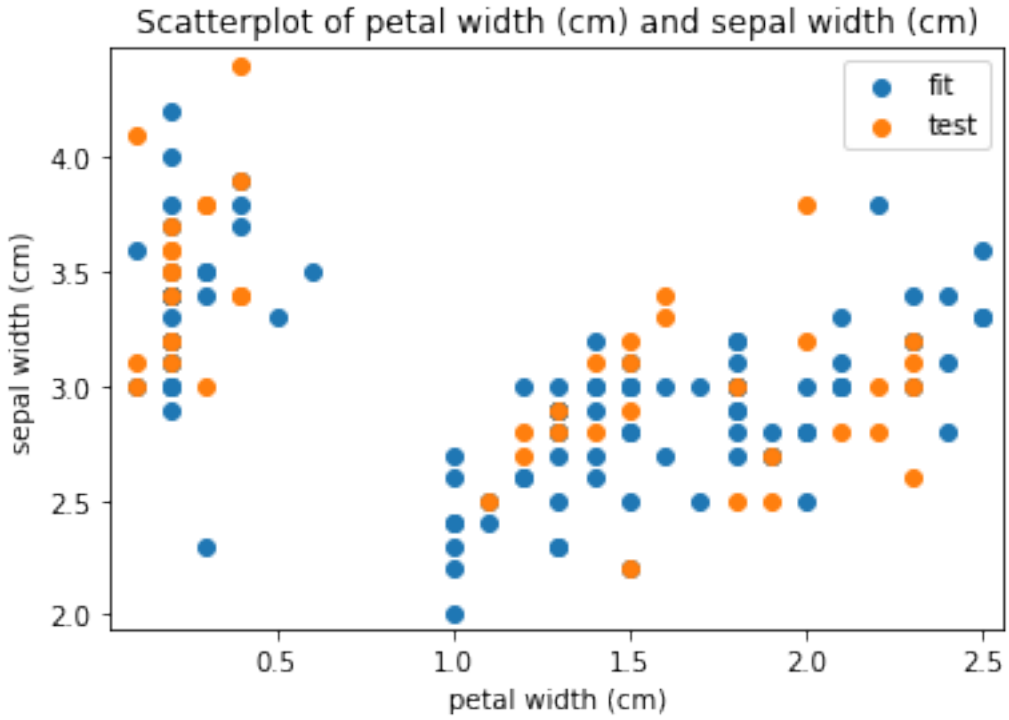

Scatterplot of petal length (cm) and sepal length (cm)
fit
test
sepal length (cm)
petal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
1
2
3
4
5
6
7

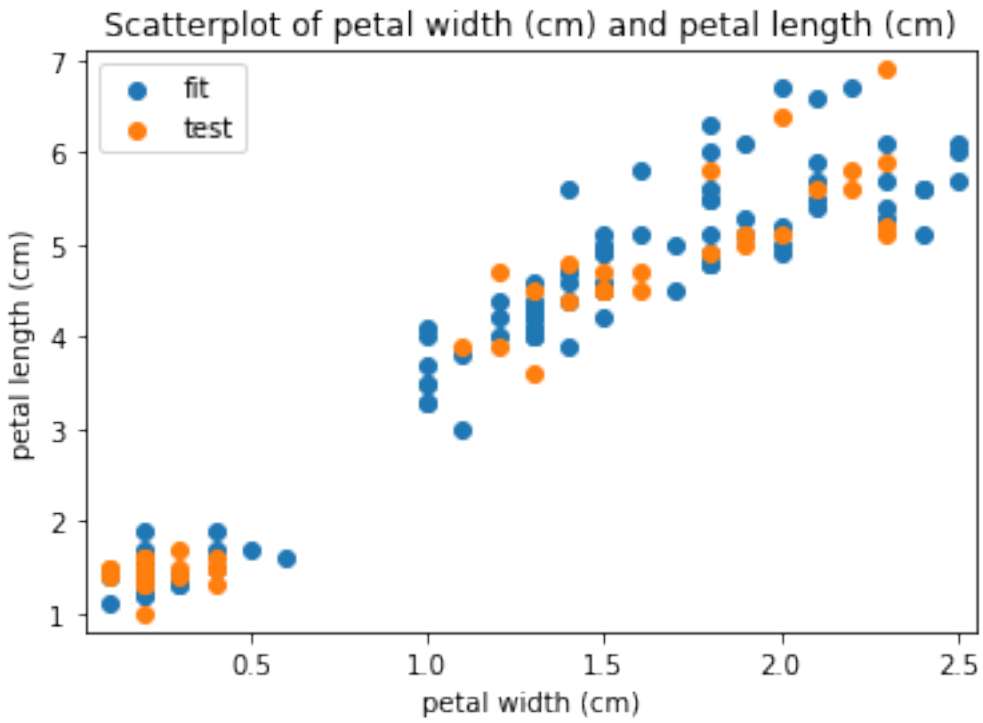

Scatterplot of petal length (cm) and sepal width (cm)
fit
test
sepal width (cm)
petal length (cm)
4.0
3.5
3.0
2.5
2.0
1
2
3
4
5
6
7

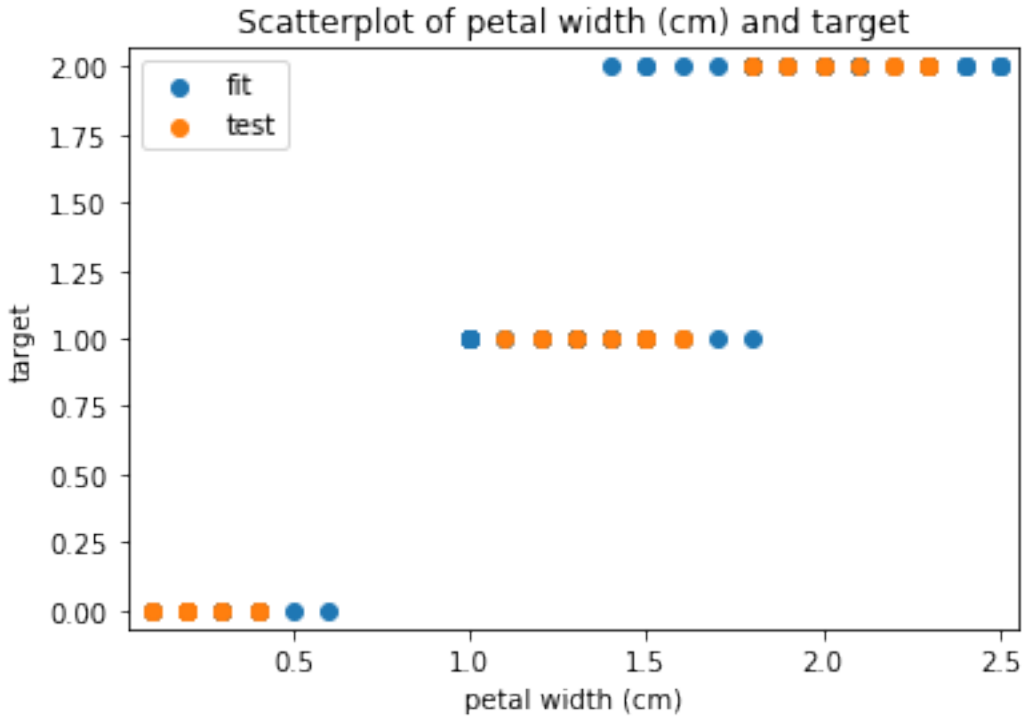

Scatterplot of petal length (cm) and petal width (cm)
fit
test
petal width (cm)
petal length (cm)
2.5
2.0
1.5
1.0
0.5
1
2
3
4
5
6
7

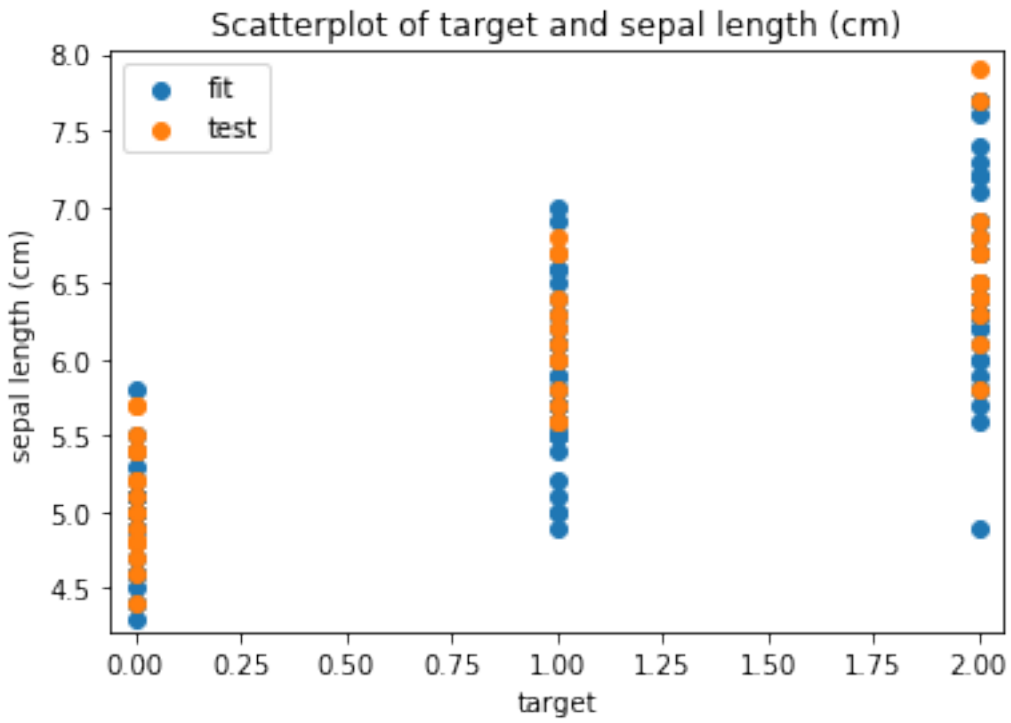

Scatterplot of petal length (cm) and target
fit
test
target
petal length (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6
7

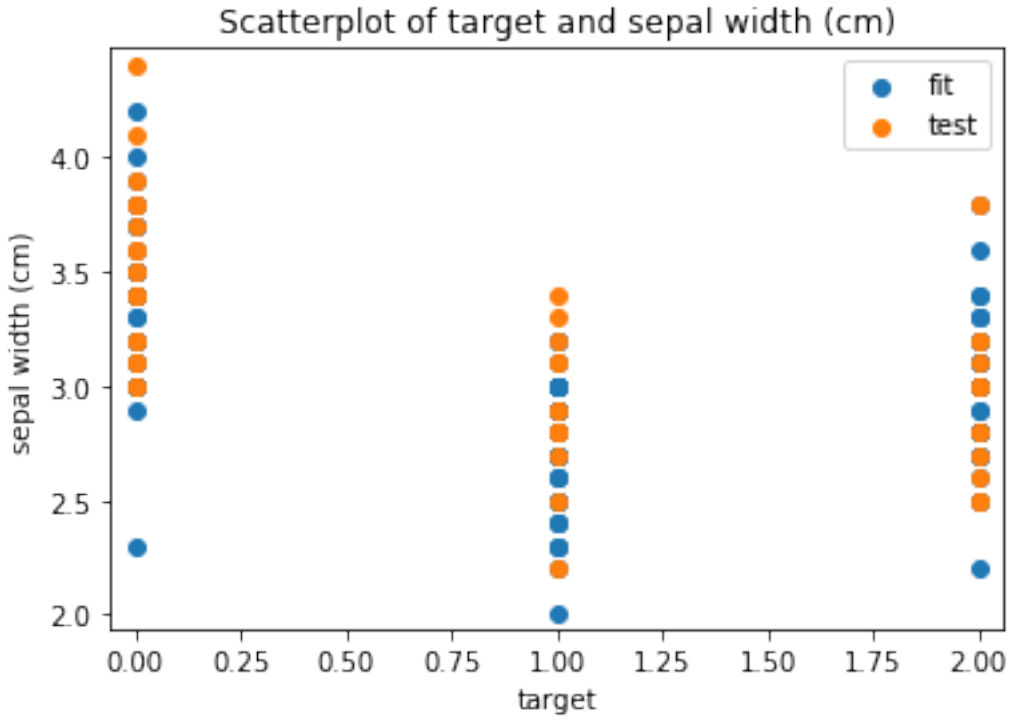

Scatterplot of petal width (cm) and sepal length (cm)
fit
test
sepal length (cm)
petal width (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.5
1.0
1.5
2.0
2.5

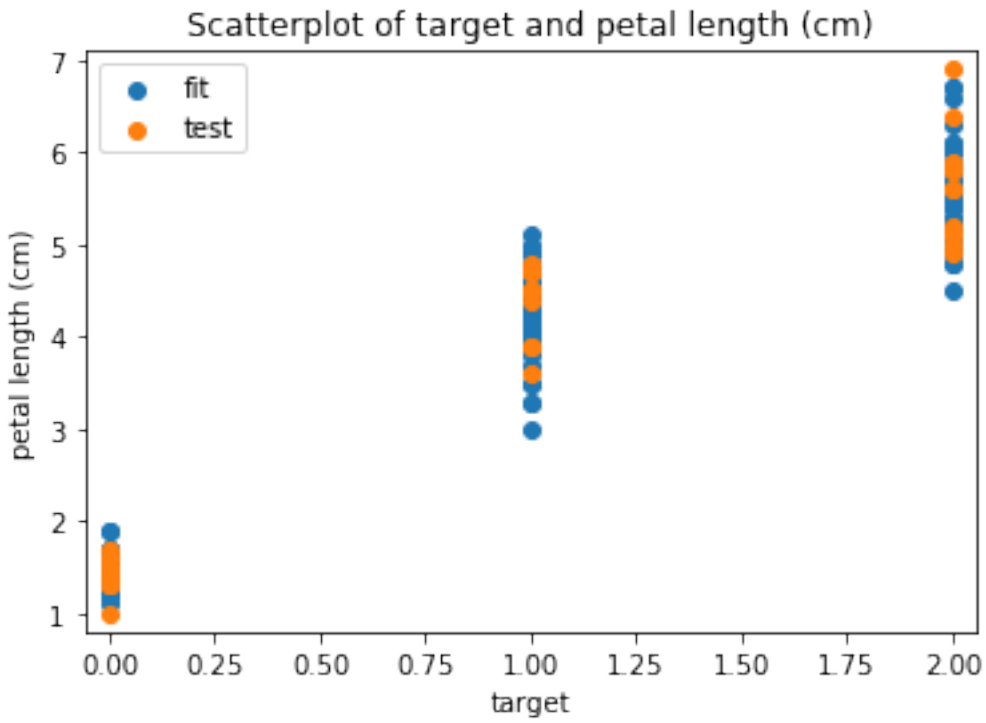

Scatterplot of petal width (cm) and sepal width (cm)
fit
test
sepal width (cm)
petal width (cm)
4.0
3.5
3.0
2.5
2.0
0.5
1.0
1.5
2.0
2.5

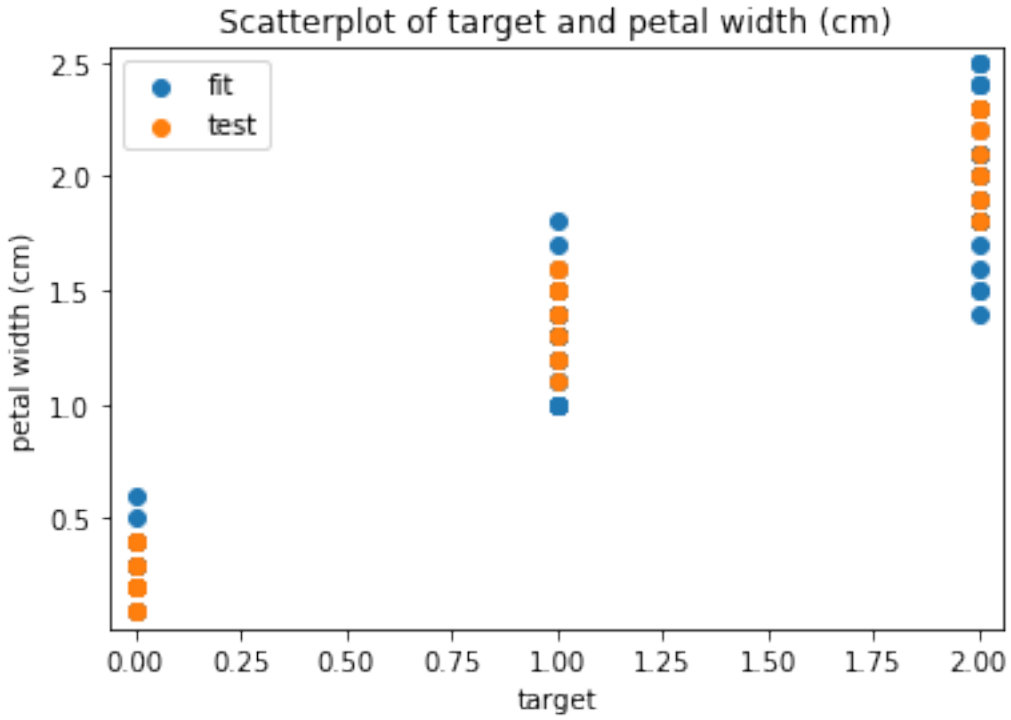

Scatterplot of petal width (cm) and petal length (cm)
fit
test
petal length (cm)
petal width (cm)
1
2
3
4
5
6
7
0.5
1.0
1.5
2.0
2.5


Scatterplot of petal width (cm) and target
fit
test
target
petal width (cm)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.5
1.0
1.5
2.0
2.5


Scatterplot of target and sepal length (cm)
fit
test
sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
target

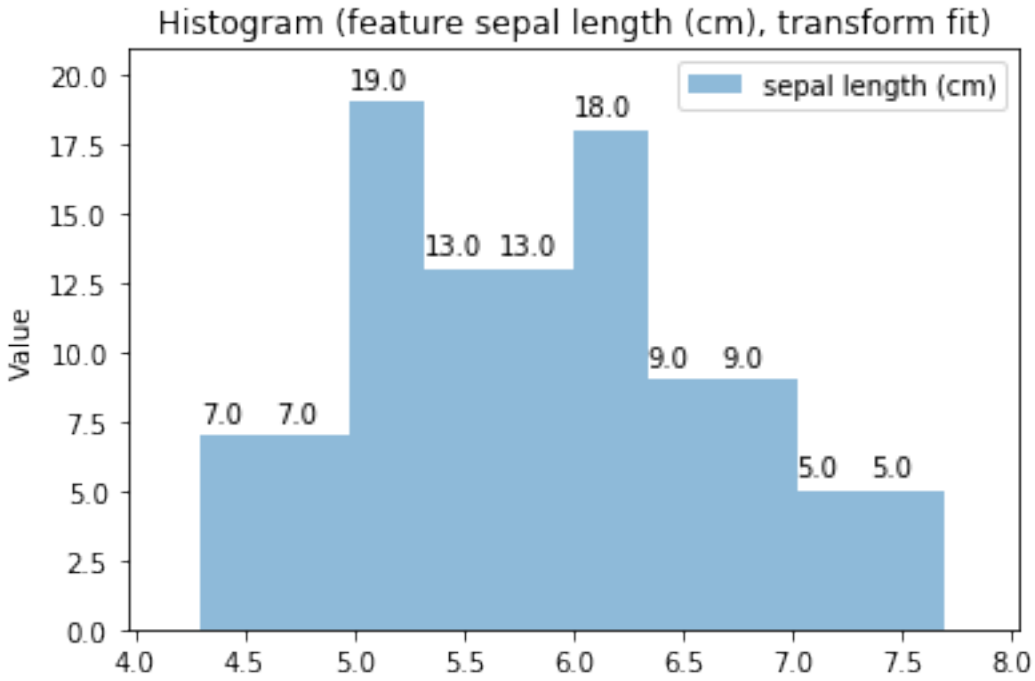

Scatterplot of target and sepal width (cm)
fit
test
sepal width (cm)
4.0
3.5
3.0
2.5
2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
target

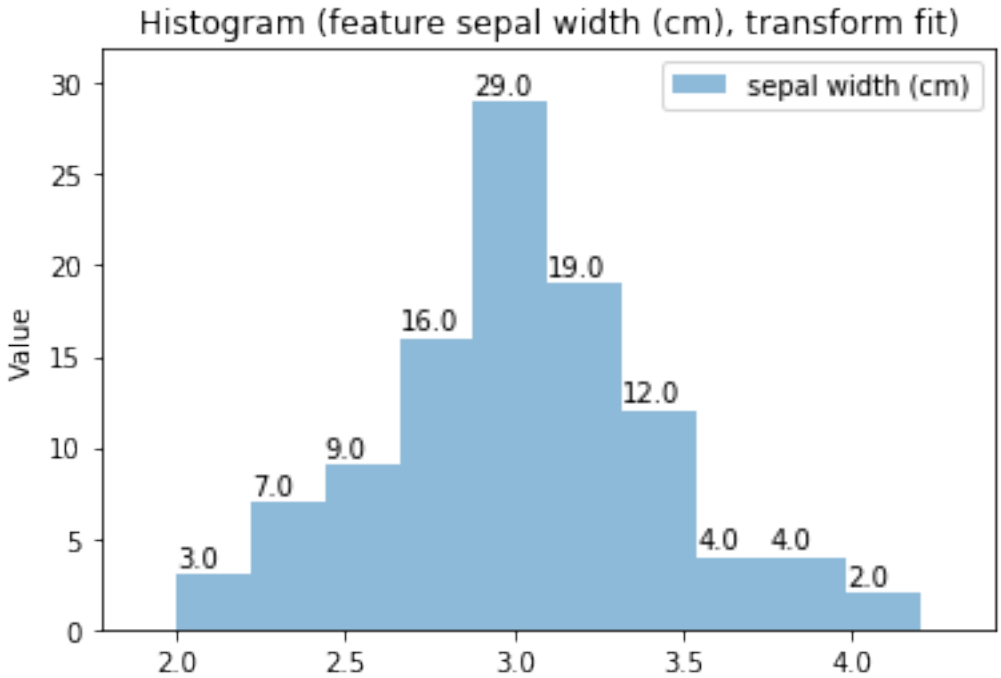


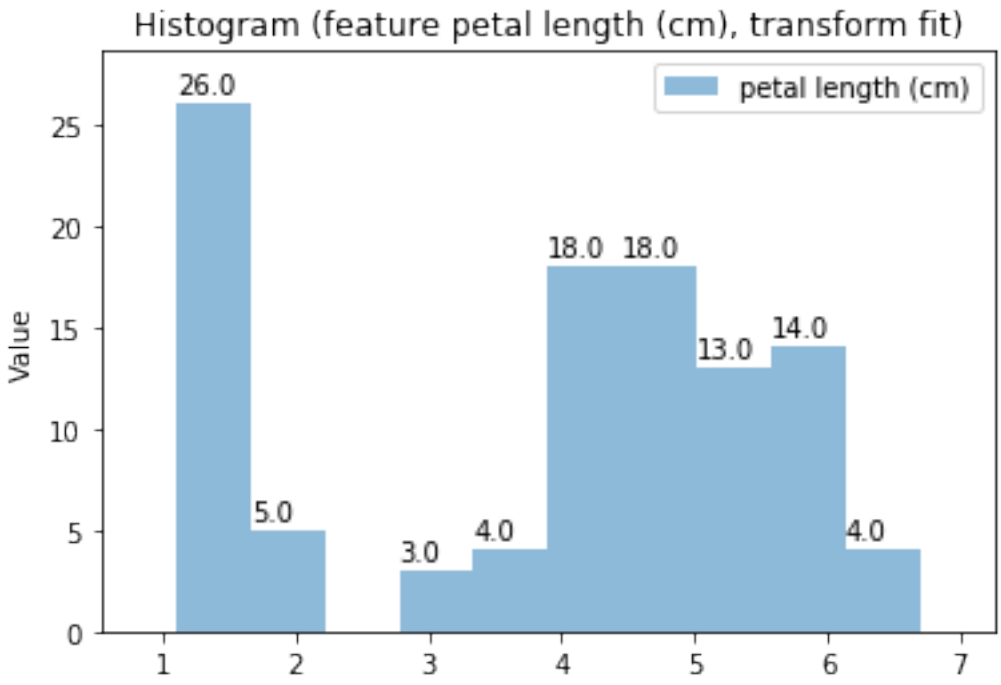


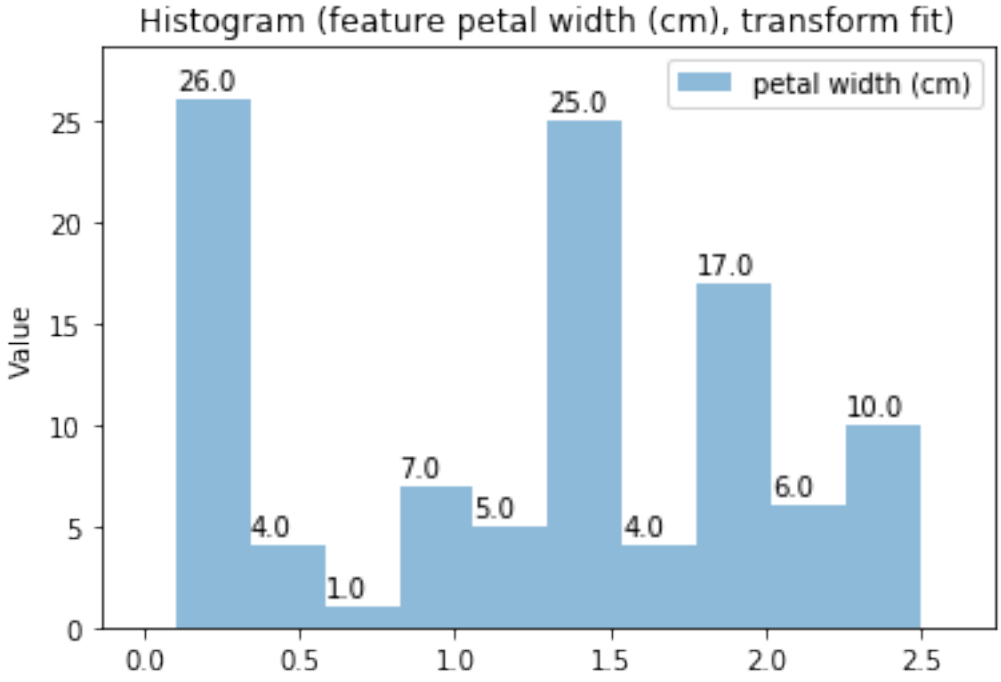

```
In [13]: p = ds.Pipeline()
         p.addPipe('read', ds.data.SampleData('iris'))
         p.addPipe('split', ds.transform.RandomTrainTestSplit(test_size=0.3), [("read", "df", "df")])
         p.addPipe('hist', ds.eda.Hist(), [("split", "df", "df")])
         p.fit_transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.T
         p.transform(transform_params={'split': {'split': ds.transform.split.TrainTestSplitBase.TEST}
```

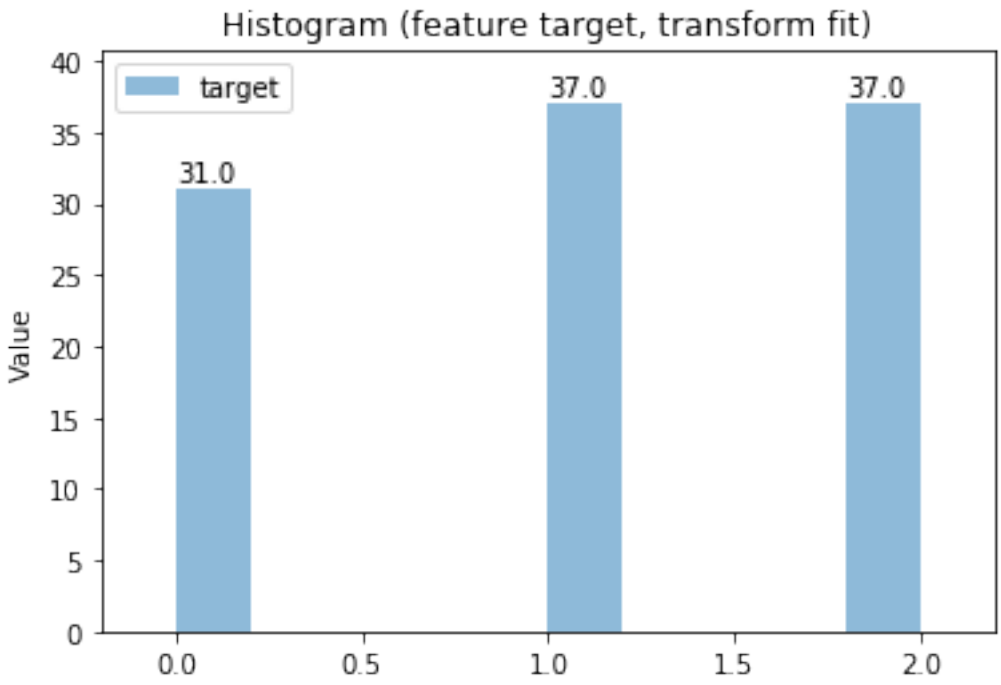
```
'Drawing diagram using blockdiag'
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB0646F6A0>
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
<IPython.core.display.HTML object>
```



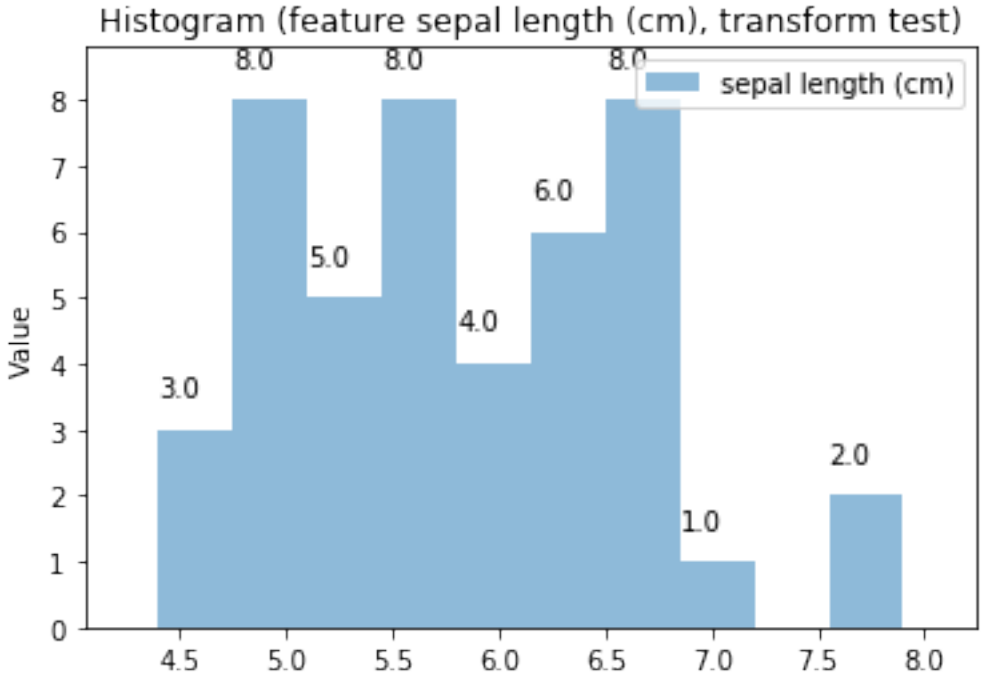


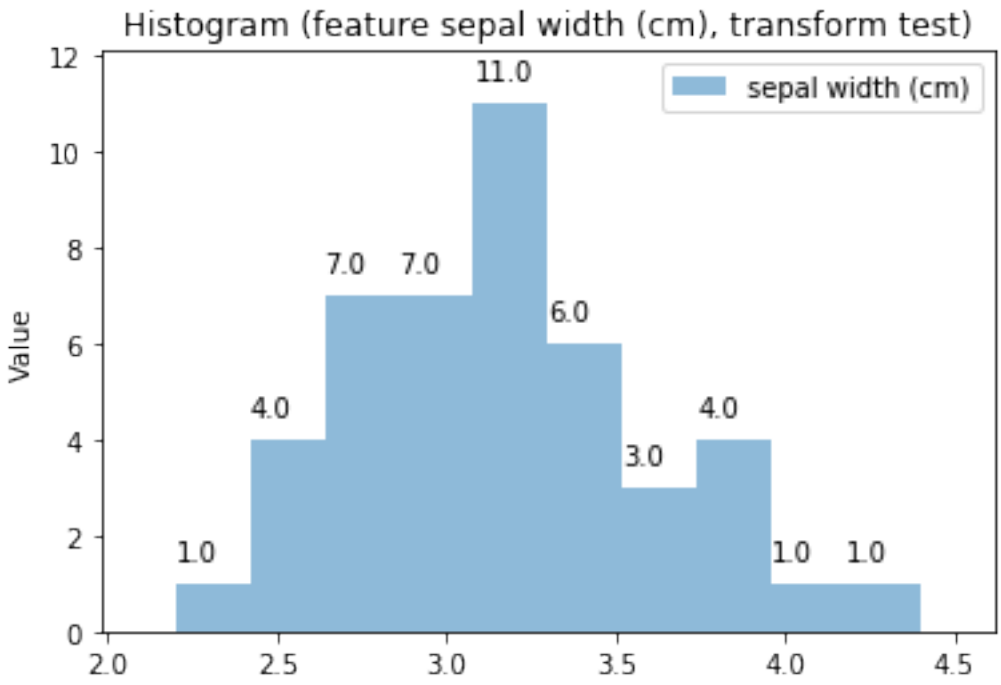

Histogram (feature petal length (cm), transform fit)
petal length (cm)
Value
26.0
5.0
3.0
4.0
18.0
18.0
13.0
14.0
4.0
0
5
10
15
20
25
1
2
3
4
5
6
7

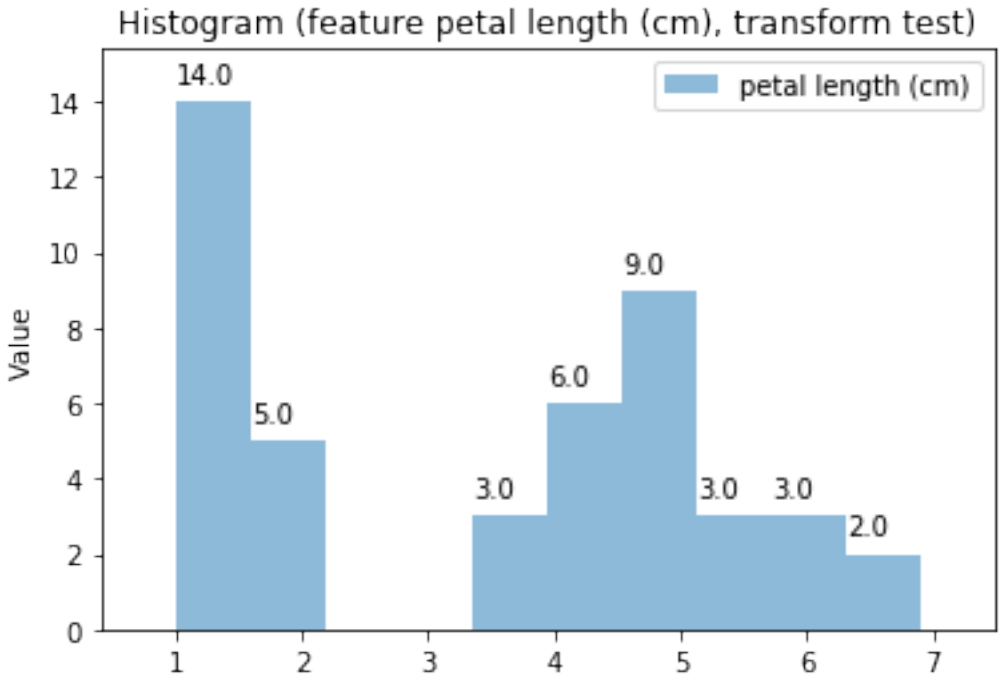

Histogram (feature petal width (cm), transform fit)
petal width (cm)
Value
26.0
4.0
1.0
7.0
5.0
25.0
4.0
17.0
6.0
10.0
0
5
10
15
20
25
0.0
0.5
1.0
1.5
2.0
2.5

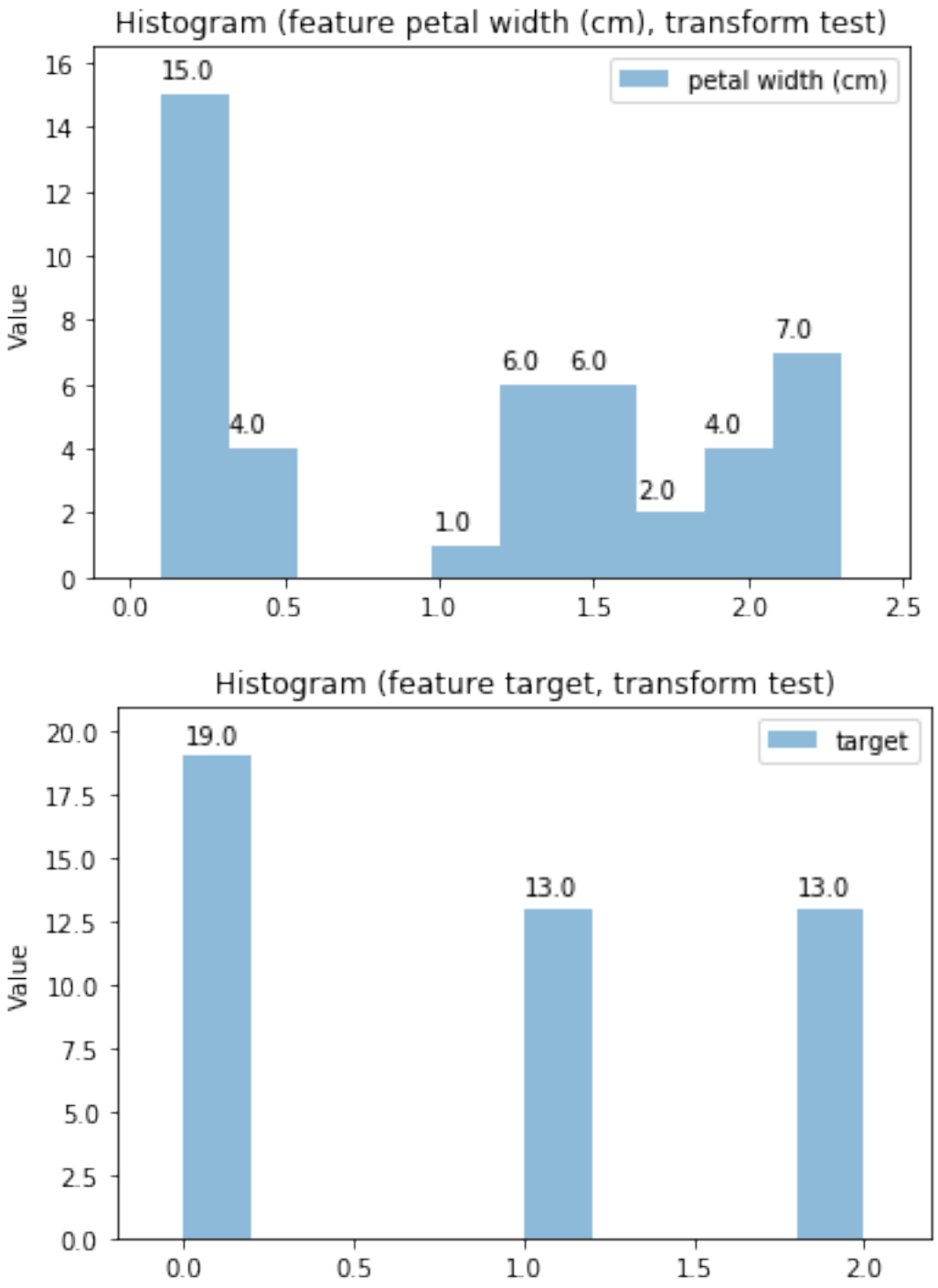



```
'Drawing diagram using blockdiag'
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB05B95208>
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
<IPython.core.display.HTML object>
```




Histogram (feature sepal width (cm), transform test)
sepal width (cm)
Value
11.0
7.0
7.0
6.0
4.0
4.0
3.0
1.0
1.0
1.0


Histogram (feature petal length (cm), transform test)
petal length (cm)
Value
14.0
9.0
6.0
5.0
3.0
3.0
3.0
2.0



Drawing a pipeline

Once defined, a pipeline can be drawn.

```
In [14]: p = ds.Pipeline()
         p.addPipe('read', ds.data.CSVDataImportPipe())
         p.addPipe('read2', ds.data.CSVDataImportPipe())
         p.addPipe('numeric', ds.transform.FilterTypeFeatures(), [("read", "df", "df")])
```

```
        p.addPipe('numeric2', ds.transform.FilterTypeFeatures(), [("read2", "df", "df")])
        p.addPipe('boxplot', ds.eda.BoxPlot(), [("numeric", "df", "df"), ("numeric2", "df", "df")])
        p.draw_design()
```

```
'Drawing diagram using blockdiag'
```

```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=448x360 at 0x7FCB05BA46A0>
```

Predicting

```
In [15]: from sklearn.neighbors import KNeighborsClassifier
         p = ds.Pipeline()
         p.addPipe('read', ds.data.SampleData('iris'))
         p.addPipe('clf', ds.predictor.SklearnClassifier(KNeighborsClassifier, n_neighbors=3), [("rea
         p.fit_transform()
         p.get_pipe_output('clf')
```

```
'Drawing diagram using blockdiag'
```

```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x280 at 0x7FCB063D8630>
```

```
<IPython.core.display.HTML object>
```

```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
```

```
Out[15]: {'predict':      y_pred_Setosa  y_pred_Versicolour  y_pred_Virginica  y_pred  target
          0              1.0            0.000000          0.000000     0.0       0
          1              1.0            0.000000          0.000000     0.0       0
          2              1.0            0.000000          0.000000     0.0       0
          3              1.0            0.000000          0.000000     0.0       0
          4              1.0            0.000000          0.000000     0.0       0
          5              1.0            0.000000          0.000000     0.0       0
          6              1.0            0.000000          0.000000     0.0       0
          7              1.0            0.000000          0.000000     0.0       0
          8              1.0            0.000000          0.000000     0.0       0
          9              1.0            0.000000          0.000000     0.0       0
          10             1.0            0.000000          0.000000     0.0       0
          11             1.0            0.000000          0.000000     0.0       0
          12             1.0            0.000000          0.000000     0.0       0
          13             1.0            0.000000          0.000000     0.0       0
          14             1.0            0.000000          0.000000     0.0       0
          15             1.0            0.000000          0.000000     0.0       0
          16             1.0            0.000000          0.000000     0.0       0
          17             1.0            0.000000          0.000000     0.0       0
          18             1.0            0.000000          0.000000     0.0       0
          19             1.0            0.000000          0.000000     0.0       0
          20             1.0            0.000000          0.000000     0.0       0
          21             1.0            0.000000          0.000000     0.0       0
          22             1.0            0.000000          0.000000     0.0       0
          23             1.0            0.000000          0.000000     0.0       0
          24             1.0            0.000000          0.000000     0.0       0
          25             1.0            0.000000          0.000000     0.0       0
          26             1.0            0.000000          0.000000     0.0       0
          27             1.0            0.000000          0.000000     0.0       0
          28             1.0            0.000000          0.000000     0.0       0
          29             1.0            0.000000          0.000000     0.0       0
          ..             ...                 ...               ...     ...     ...
          120            0.0            0.000000          1.000000     0.0       2
          121            0.0            0.000000          1.000000     0.0       2
          122            0.0            0.000000          1.000000     0.0       2
          123            0.0            0.000000          1.000000     0.0       2
```

```
124            0.0            0.000000          1.000000     0.0       2
125            0.0            0.000000          1.000000     0.0       2
126            0.0            0.000000          1.000000     0.0       2
127            0.0            0.000000          1.000000     0.0       2
128            0.0            0.000000          1.000000     0.0       2
129            0.0            0.000000          1.000000     0.0       2
130            0.0            0.000000          1.000000     0.0       2
131            0.0            0.000000          1.000000     0.0       2
132            0.0            0.000000          1.000000     0.0       2
133            0.0            0.666667          0.333333     1.0       2
134            0.0            0.333333          0.666667     0.0       2
135            0.0            0.000000          1.000000     0.0       2
136            0.0            0.000000          1.000000     0.0       2
137            0.0            0.000000          1.000000     0.0       2
138            0.0            0.333333          0.666667     0.0       2
139            0.0            0.000000          1.000000     0.0       2
140            0.0            0.000000          1.000000     0.0       2
141            0.0            0.000000          1.000000     0.0       2
142            0.0            0.000000          1.000000     0.0       2
143            0.0            0.000000          1.000000     0.0       2
144            0.0            0.000000          1.000000     0.0       2
145            0.0            0.000000          1.000000     0.0       2
146            0.0            0.000000          1.000000     0.0       2
147            0.0            0.000000          1.000000     0.0       2
148            0.0            0.000000          1.000000     0.0       2
149            0.0            0.000000          1.000000     0.0       2

[150 rows x 5 columns],
'predict_metadata': {'classes': ['Setosa', 'Versicolour', 'Virginica'],
 'y_true_label': 'target',
 'threshold': 0.5}}
```

Scoring

```
In [16]: from sklearn.neighbors import KNeighborsClassifier
         p = ds.Pipeline()
         p.addPipe('read', ds.data.SampleData('iris'))
         p.addPipe('clf', ds.predictor.SklearnClassifier(KNeighborsClassifier, n_neighbors=3), [("rea
         p.addPipe('score', ds.score.ClassificationScore(), [("clf", "predict", "predict"), ("clf", '
         p.fit_transform()
```

```
'Drawing diagram using blockdiag'
```

```
<PIL.PngImagePlugin.PngImageFile image mode=RGB size=256x360 at 0x7FCB058D8048>
```

```
<IPython.core.display.HTML object>
```

```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
```

```
'auc() not yet implemented for multiclass classifiers'
```

```
'plot_auc() not yet implemented for multiclass classifiers'
```

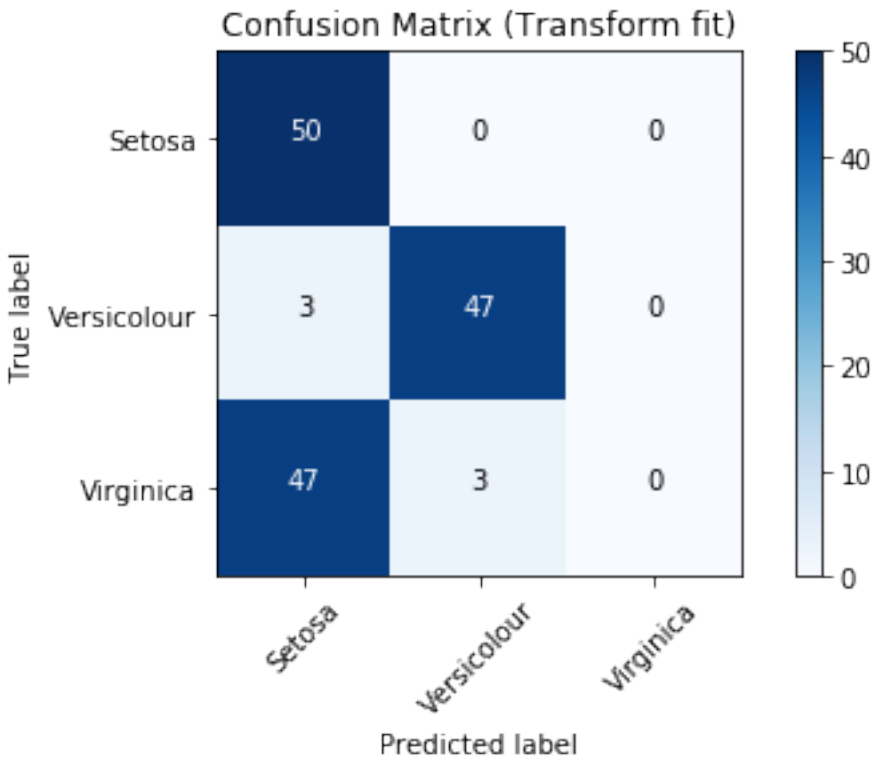
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```




```
'Precision-recall-curve not yet implemented for multiclass classifiers'
```

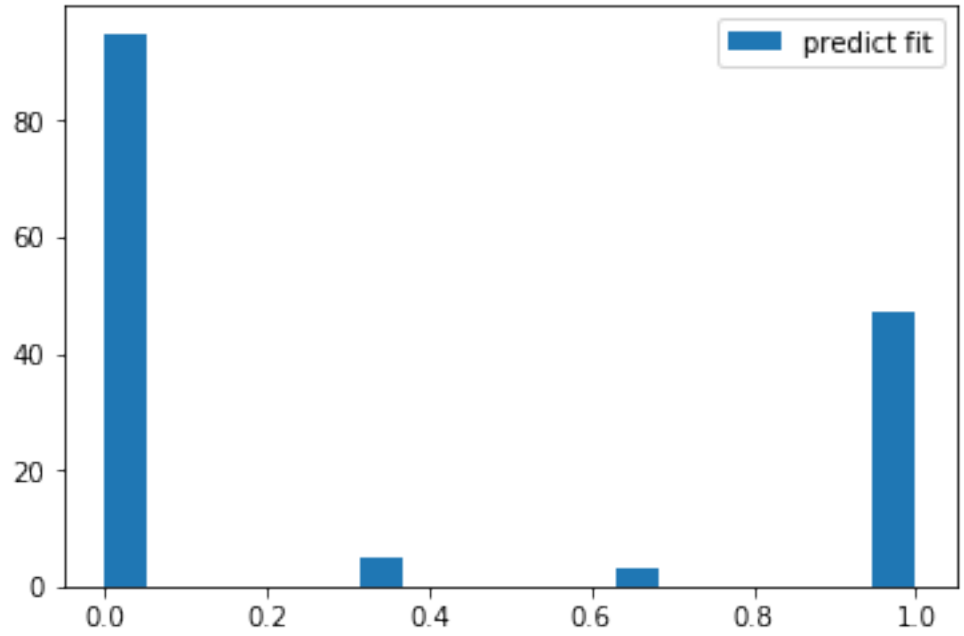
```
'log_loss() not yet implemented for multiclass classifiers'
```

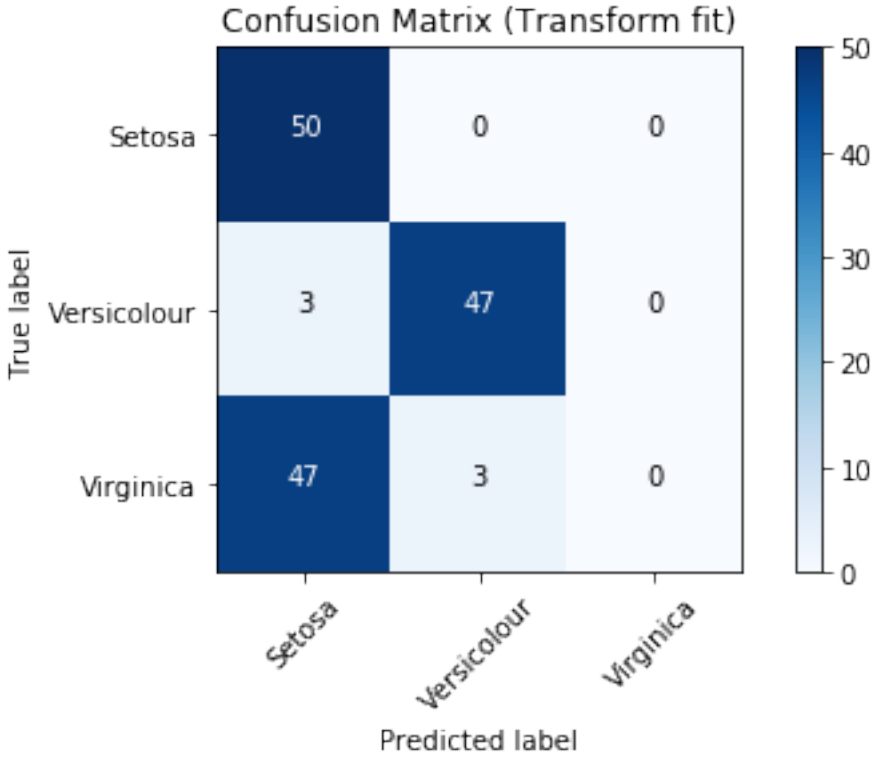
```
<IPython.core.display.HTML object>
```

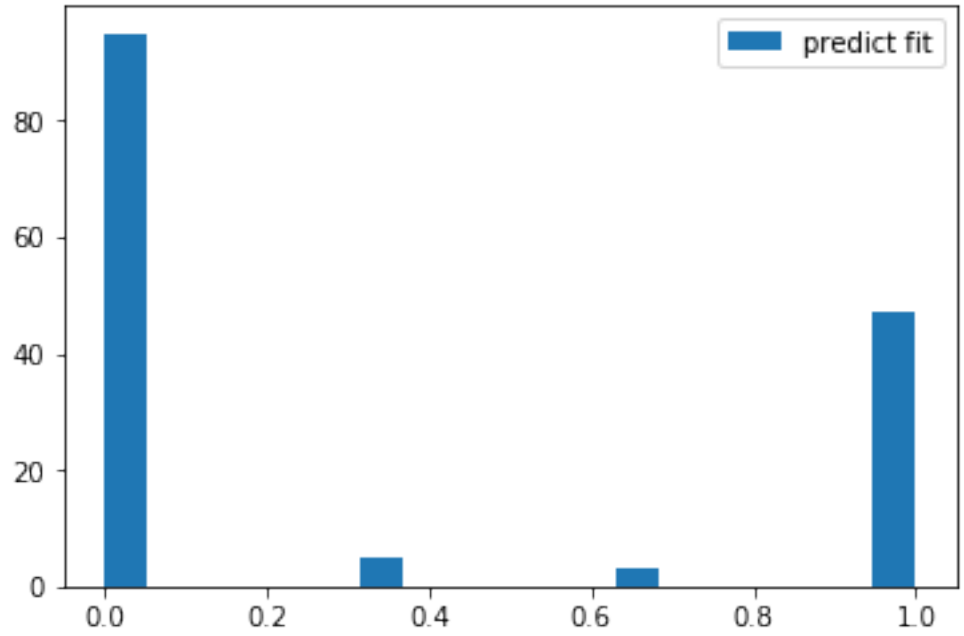
```
/home/docs/checkouts/readthedocs.org/user_builds/dvb-datascience-test/envs/latest/lib/python3.6/site-
  'precision', 'predicted', average, warn_for)
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```


predict fit
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix (Transform fit)
Setosa
Versicolour
Virginica
True label
50
0
0
3
47
0
47
3
0
50
40
30
20
10
0
Setosa
Versicolour
Virginica
Predicted label



1.12.3 Fetching the output

You can fetch the output of a pipe using the following:

```
In [17]: p.get_pipe_output('clf')
```

```
Out[17]: {'predict':      y_pred_Setosa  y_pred_Versicolour  y_pred_Virginica  y_pred  target
          0               1.0            0.000000          0.000000     0.0       0
          1               1.0            0.000000          0.000000     0.0       0
          2               1.0            0.000000          0.000000     0.0       0
          3               1.0            0.000000          0.000000     0.0       0
          4               1.0            0.000000          0.000000     0.0       0
          5               1.0            0.000000          0.000000     0.0       0
          6               1.0            0.000000          0.000000     0.0       0
          7               1.0            0.000000          0.000000     0.0       0
          8               1.0            0.000000          0.000000     0.0       0
          9               1.0            0.000000          0.000000     0.0       0
          10              1.0            0.000000          0.000000     0.0       0
          11              1.0            0.000000          0.000000     0.0       0
          12              1.0            0.000000          0.000000     0.0       0
          13              1.0            0.000000          0.000000     0.0       0
          14              1.0            0.000000          0.000000     0.0       0
          15              1.0            0.000000          0.000000     0.0       0
          16              1.0            0.000000          0.000000     0.0       0
          17              1.0            0.000000          0.000000     0.0       0
          18              1.0            0.000000          0.000000     0.0       0
          19              1.0            0.000000          0.000000     0.0       0
          20              1.0            0.000000          0.000000     0.0       0
          21              1.0            0.000000          0.000000     0.0       0
          22              1.0            0.000000          0.000000     0.0       0
          23              1.0            0.000000          0.000000     0.0       0
          24              1.0            0.000000          0.000000     0.0       0
          25              1.0            0.000000          0.000000     0.0       0
          26              1.0            0.000000          0.000000     0.0       0
```

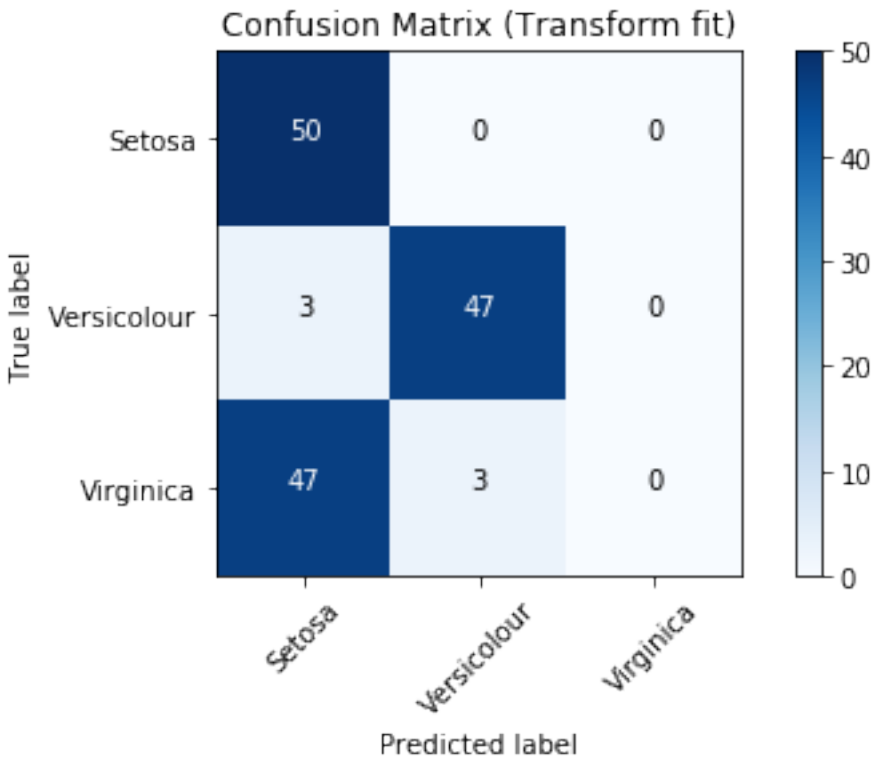
```
27             1.0            0.000000          0.000000     0.0       0
28             1.0            0.000000          0.000000     0.0       0
29             1.0            0.000000          0.000000     0.0       0
..             ...                 ...               ...     ...     ...
120            0.0            0.000000          1.000000     0.0       2
121            0.0            0.000000          1.000000     0.0       2
122            0.0            0.000000          1.000000     0.0       2
123            0.0            0.000000          1.000000     0.0       2
124            0.0            0.000000          1.000000     0.0       2
125            0.0            0.000000          1.000000     0.0       2
126            0.0            0.000000          1.000000     0.0       2
127            0.0            0.000000          1.000000     0.0       2
128            0.0            0.000000          1.000000     0.0       2
129            0.0            0.000000          1.000000     0.0       2
130            0.0            0.000000          1.000000     0.0       2
131            0.0            0.000000          1.000000     0.0       2
132            0.0            0.000000          1.000000     0.0       2
133            0.0            0.666667          0.333333     1.0       2
134            0.0            0.333333          0.666667     0.0       2
135            0.0            0.000000          1.000000     0.0       2
136            0.0            0.000000          1.000000     0.0       2
137            0.0            0.000000          1.000000     0.0       2
138            0.0            0.333333          0.666667     0.0       2
139            0.0            0.000000          1.000000     0.0       2
140            0.0            0.000000          1.000000     0.0       2
141            0.0            0.000000          1.000000     0.0       2
142            0.0            0.000000          1.000000     0.0       2
143            0.0            0.000000          1.000000     0.0       2
144            0.0            0.000000          1.000000     0.0       2
145            0.0            0.000000          1.000000     0.0       2
146            0.0            0.000000          1.000000     0.0       2
147            0.0            0.000000          1.000000     0.0       2
148            0.0            0.000000          1.000000     0.0       2
149            0.0            0.000000          1.000000     0.0       2

[150 rows x 5 columns],
'predict_metadata': {'classes': ['Setosa', 'Versicolour', 'Virginica'],
 'y_true_label': 'target',
 'threshold': 0.5}}
```

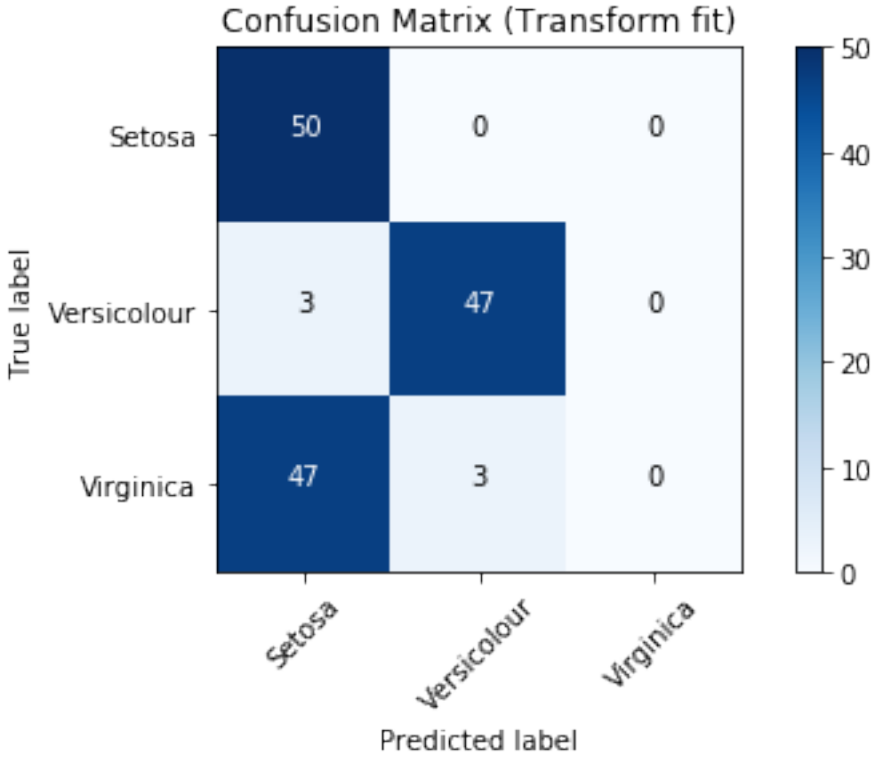
1.12.4 Confusion matrix

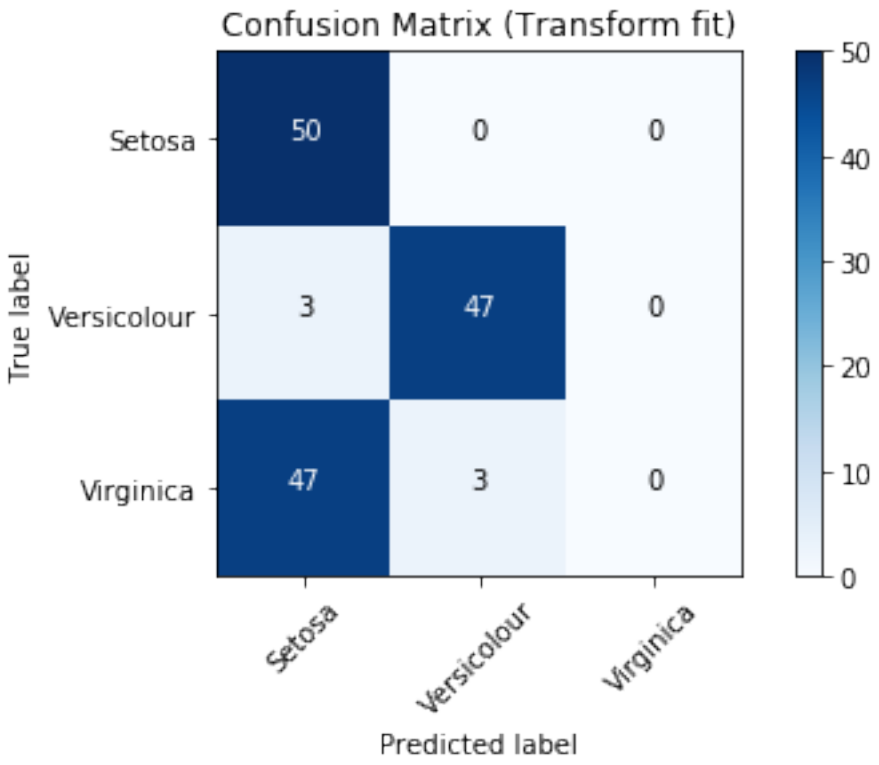
You can print the confusion matrix of a score pipe using the following:

```
In [18]: p.get_pipe('score').plot_confusion_matrix()
```

```
<IPython.core.display.HTML object>
```


Confusion Matrix (Transform fit)
Setosa
Versicolour
Virginica
True label
50
0
0
3
47
0
47
3
0
50
40
30
20
10
0
Setosa
Versicolour
Virginica
Predicted label


Confusion Matrix (Transform fit)
Setosa
Versicolour
Virginica
True label
50
0
0
3
47
0
47
3
0
50
40
30
20
10
0
Setosa
Versicolour
Virginica
Predicted label



1.12.5 Precision Recall Curve

And the same holds for the precision recall curve:

```
In [19]: p.get_pipe('score').precision_recall_curve()
```

```
'Precision-recall-curve not yet implemented for multiclass classifiers'
```

1.12.6 AUC plot

As well as the AUC plot

```
In [20]: p.get_pipe('score').plot_auc()
```

```
'plot_auc() not yet implemented for multiclass classifiers'
```

1.12.7 Plots

This notebook presents some example of possible plots you can make using the package.

First, import the package

```
In [1]: import dvb.datascience as ds
```

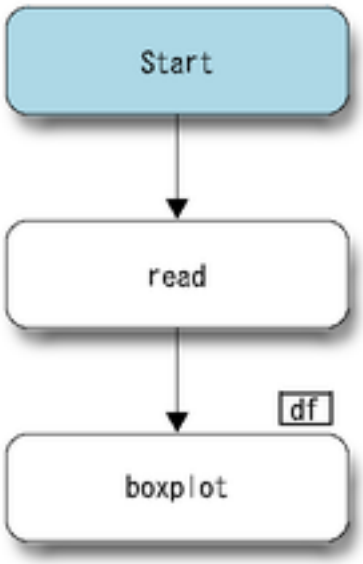
```
C:\ProgramData\Anaconda3\lib\site-packages\deap\tools\_hypervolume\pyhv.py:33: ImportWarning: Falling
  "module. Expect this to be very slow.", ImportWarning)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap_external.py:426: ImportWarning: Not importing direc
  _warnings.warn(msg.format(portions[0]), ImportWarning)
```

```
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap_external.py:426: ImportWarning: Not importing direc
  _warnings.warn(msg.format(portions[0]), ImportWarning)
```

Boxplots

```
In [2]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('boxplot', ds.eda.BoxPlot(), [("read", "df", "df")])
        p.transform(name="boxplot_example", close_plt=True)
```

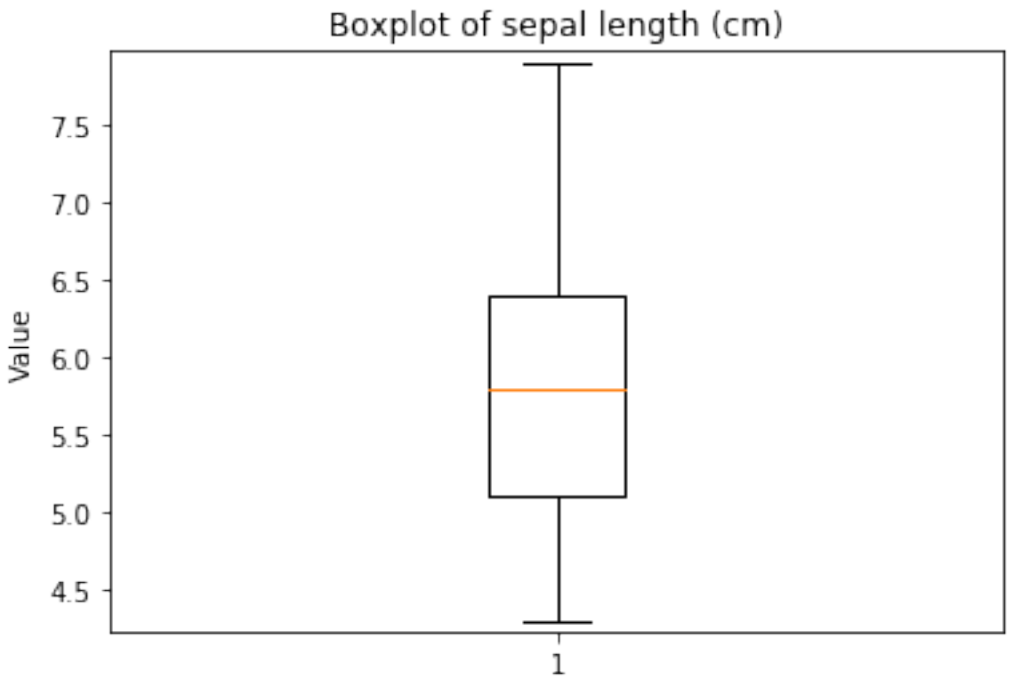
```
'Drawing diagram using blockdiag'
```

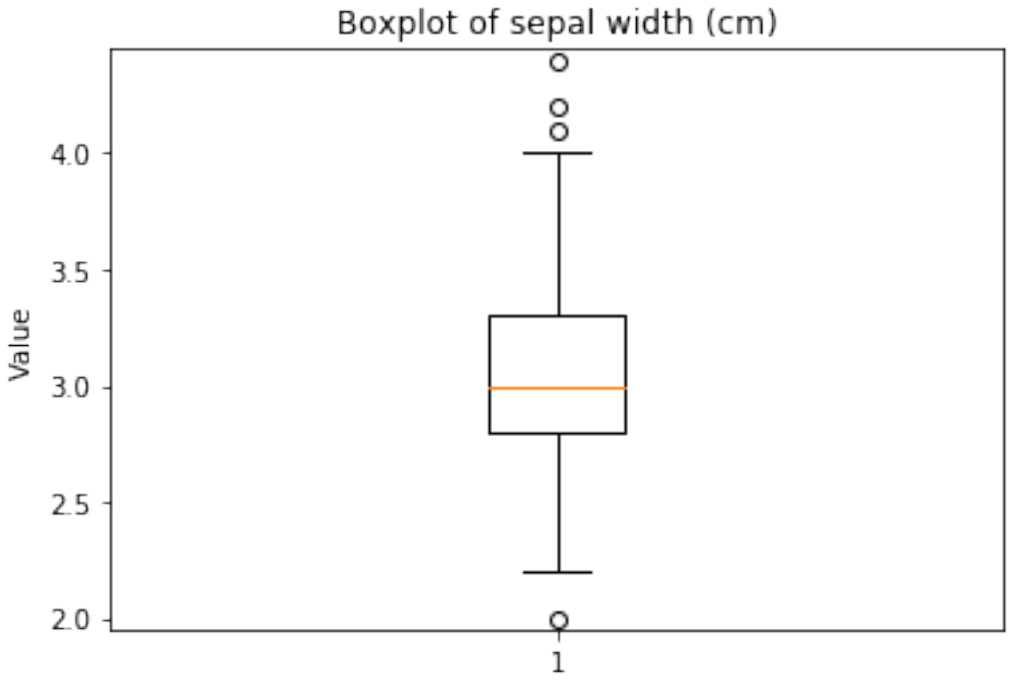


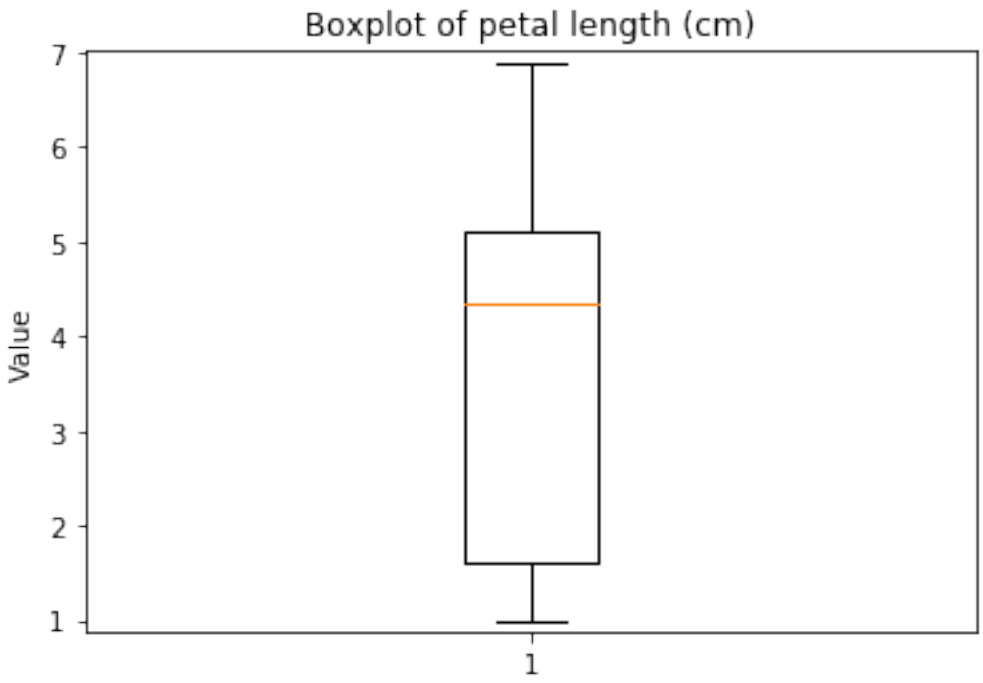

```
<IPython.core.display.HTML object>
```

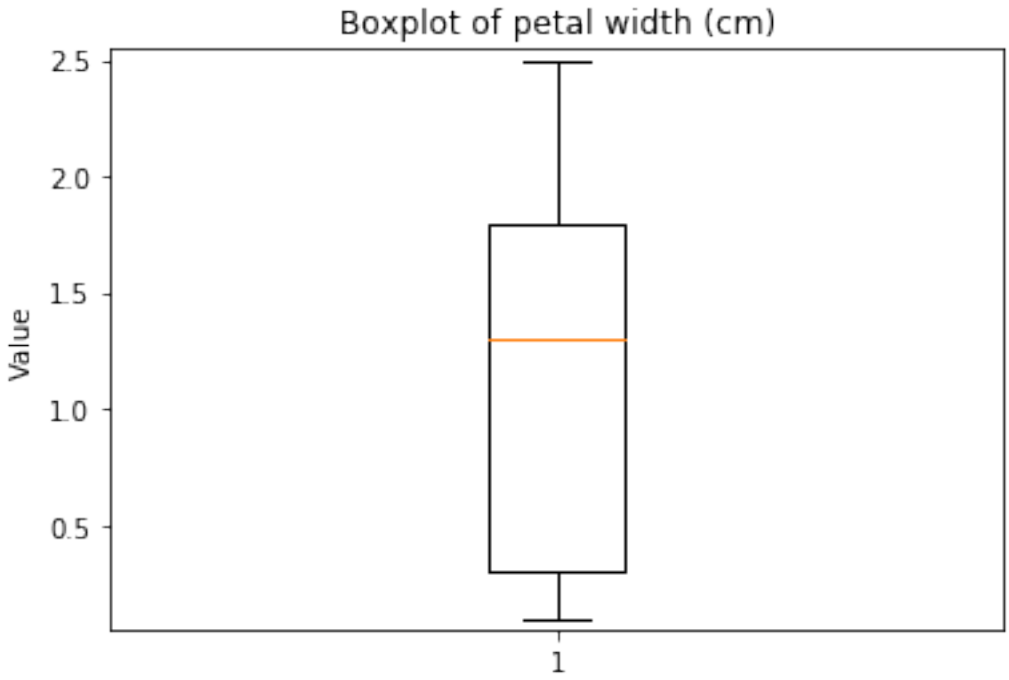
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
```

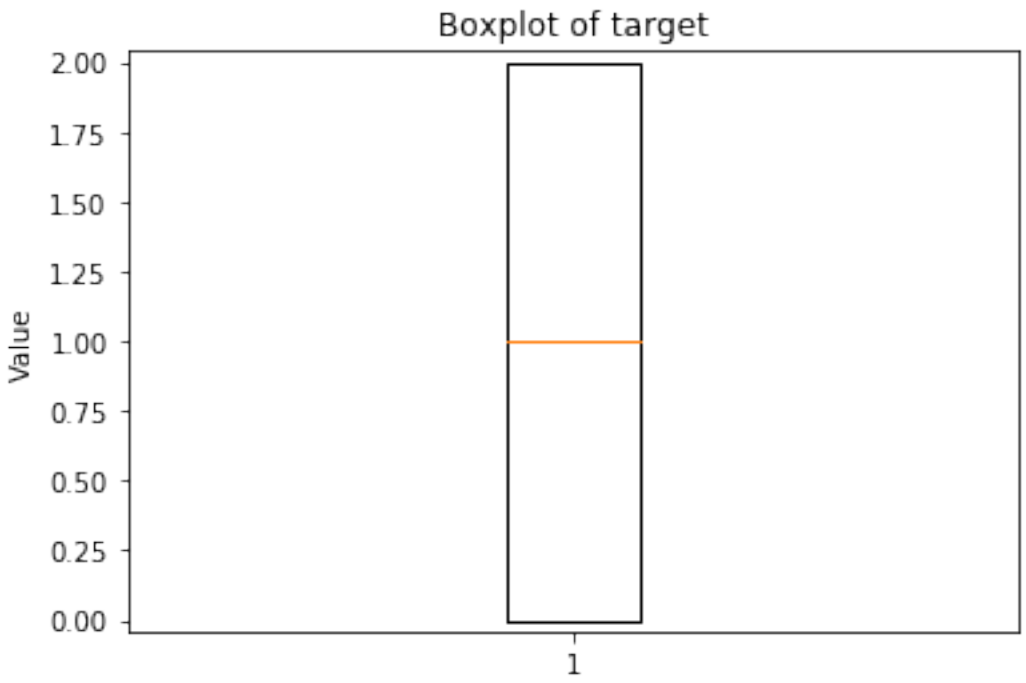
```
<IPython.core.display.HTML object>
```


Boxplot of sepal length (cm)
Value
7.5
7.0
6.5
6.0
5.5
5.0
4.5
1


Boxplot of sepal width (cm)
Value
4.0
3.5
3.0
2.5
2.0
1


Boxplot of petal length (cm)
Value
7
6
5
4
3
2
1
1

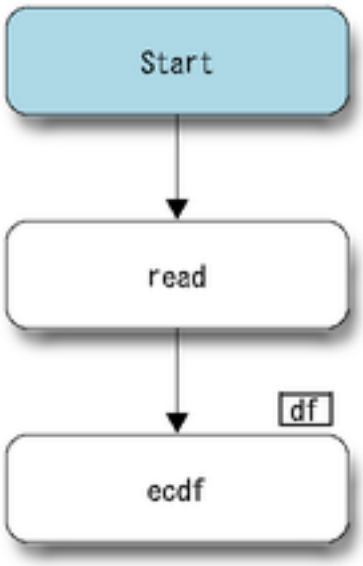

Boxplot of petal width (cm)
Value
2.5
2.0
1.5
1.0
0.5
1



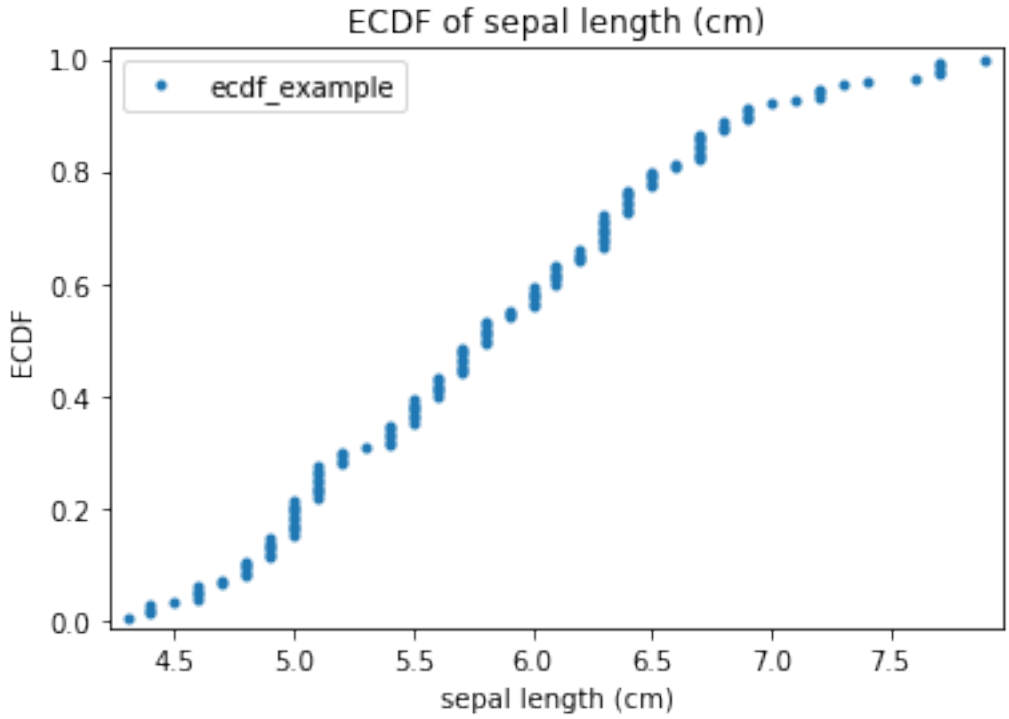
ECDF plots

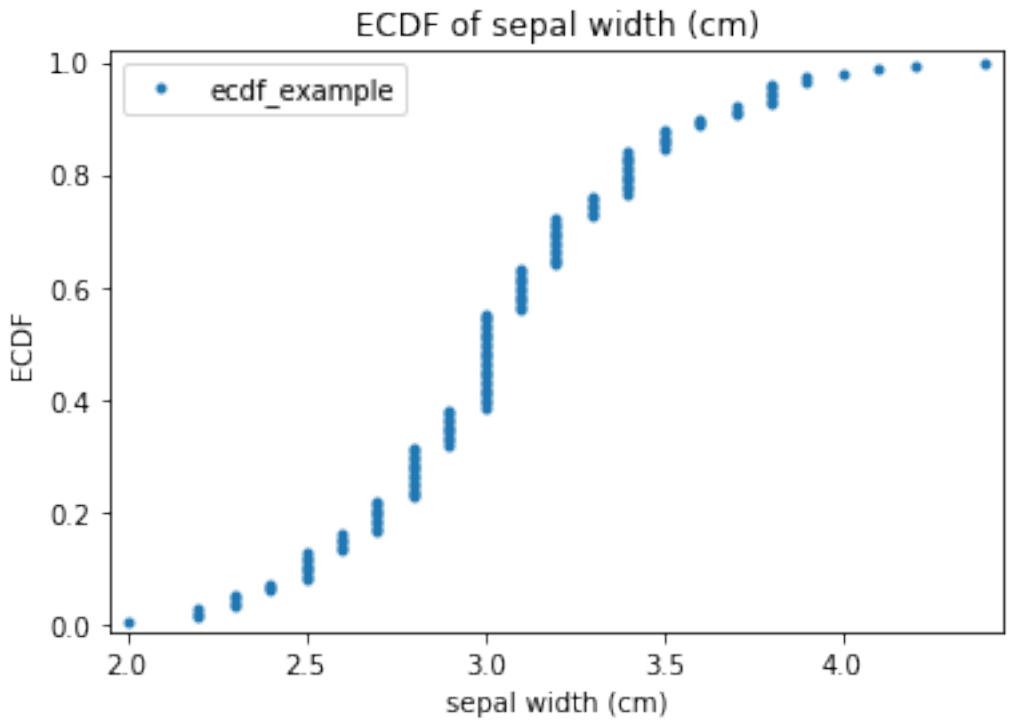
```
In [3]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('ecdf', ds.eda.ECDFPlots(), [("read", "df", "df")])
        p.transform(name="ecdf_example", close_plt=True)
```

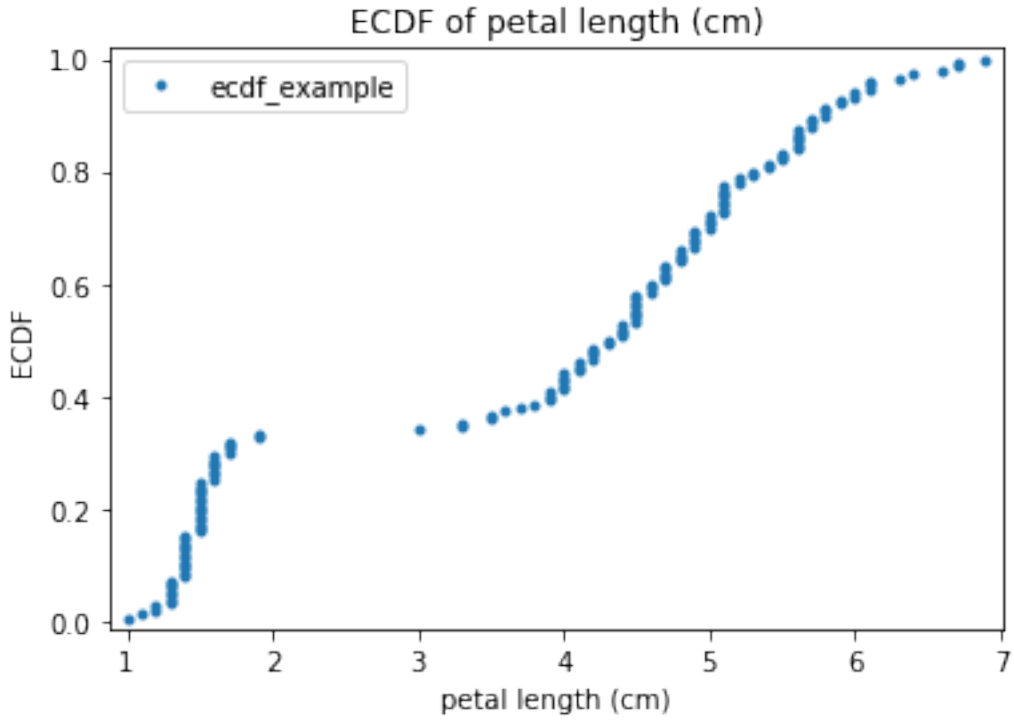
```
'Drawing diagram using blockdiag'
```

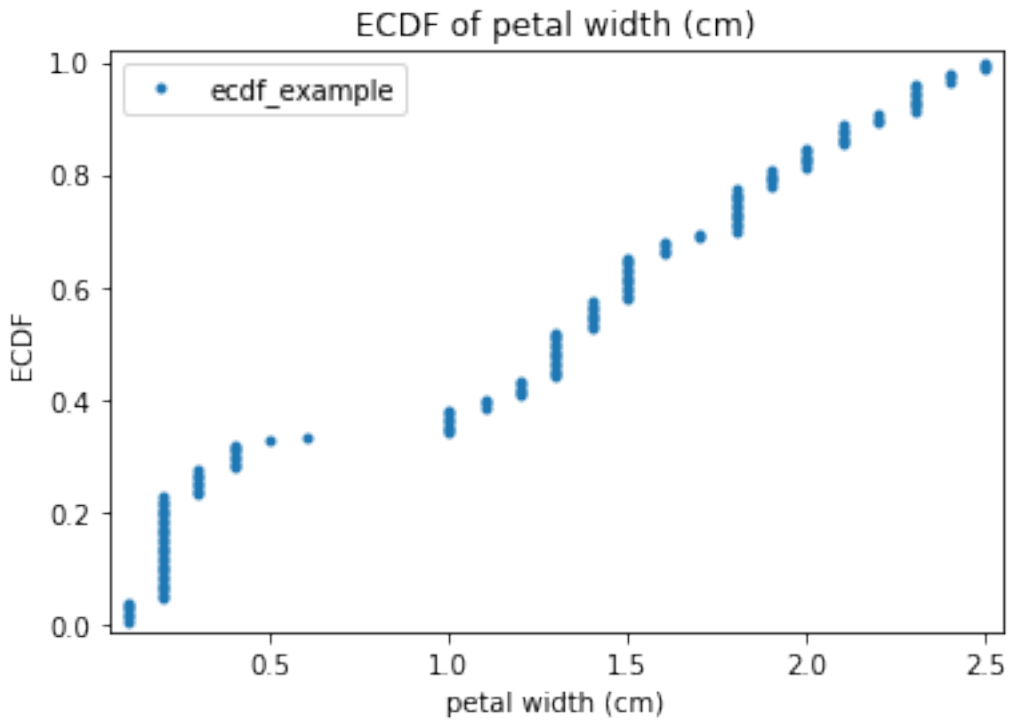


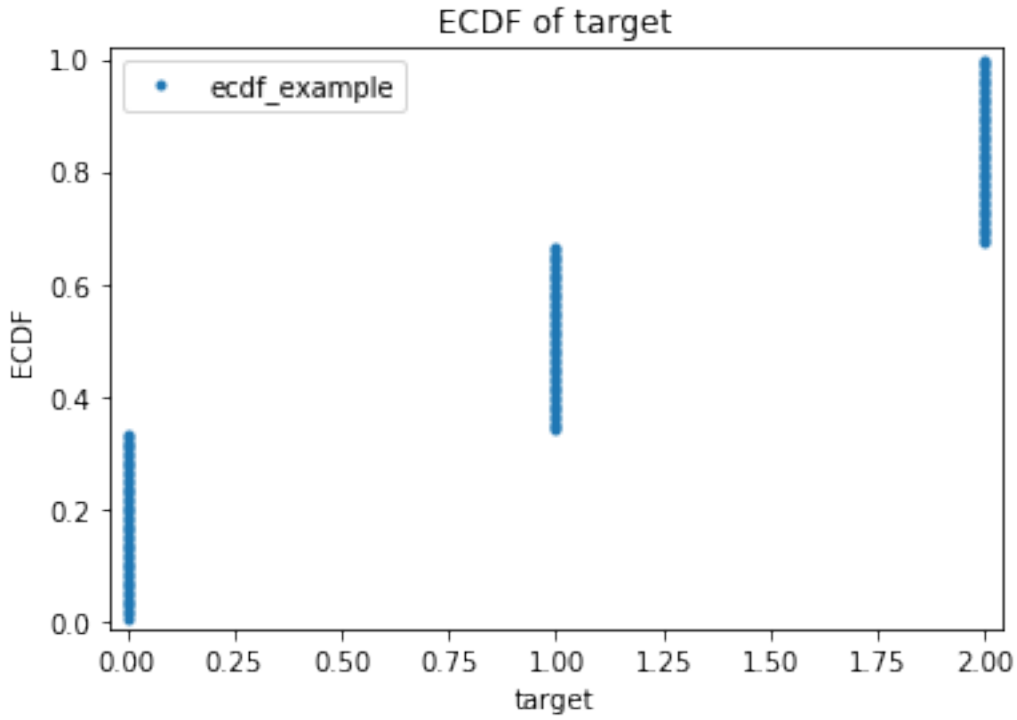

```
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
<IPython.core.display.HTML object>
```




ECDF of sepal width (cm)
ecdf_example
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
2.0
2.5
3.0
3.5
4.0
sepal width (cm)


ECDF of petal length (cm)
ecdf_example
ECDF
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
7
petal length (cm)

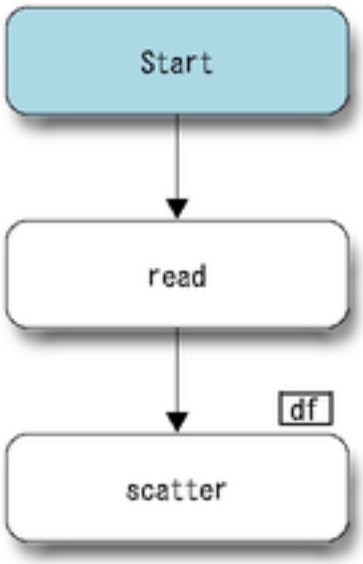




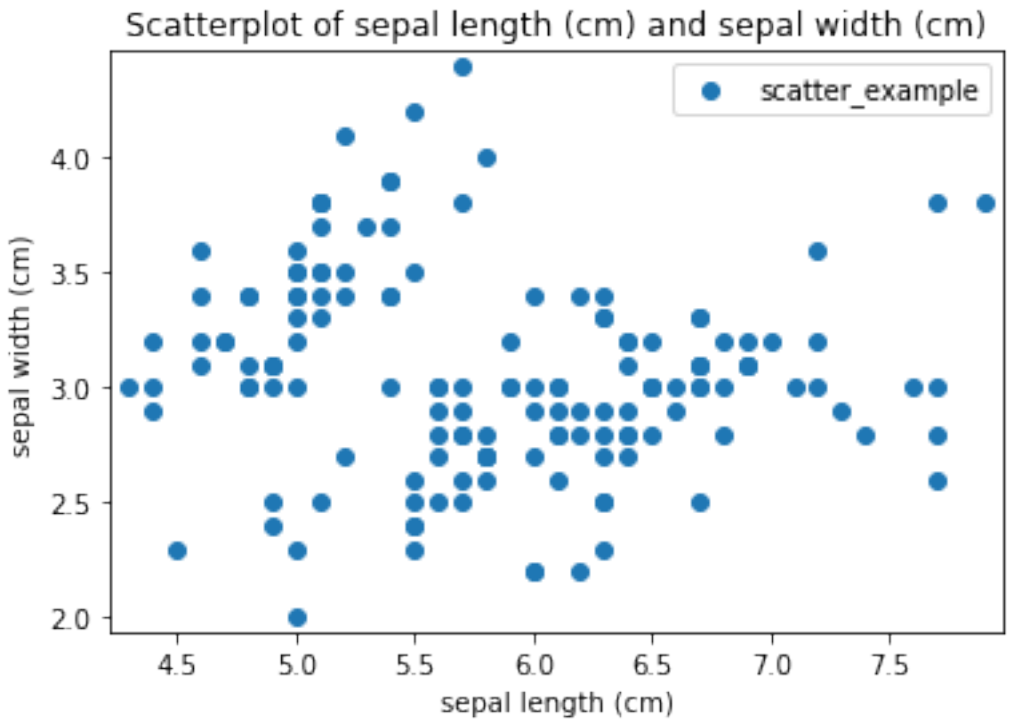
Scatter plots

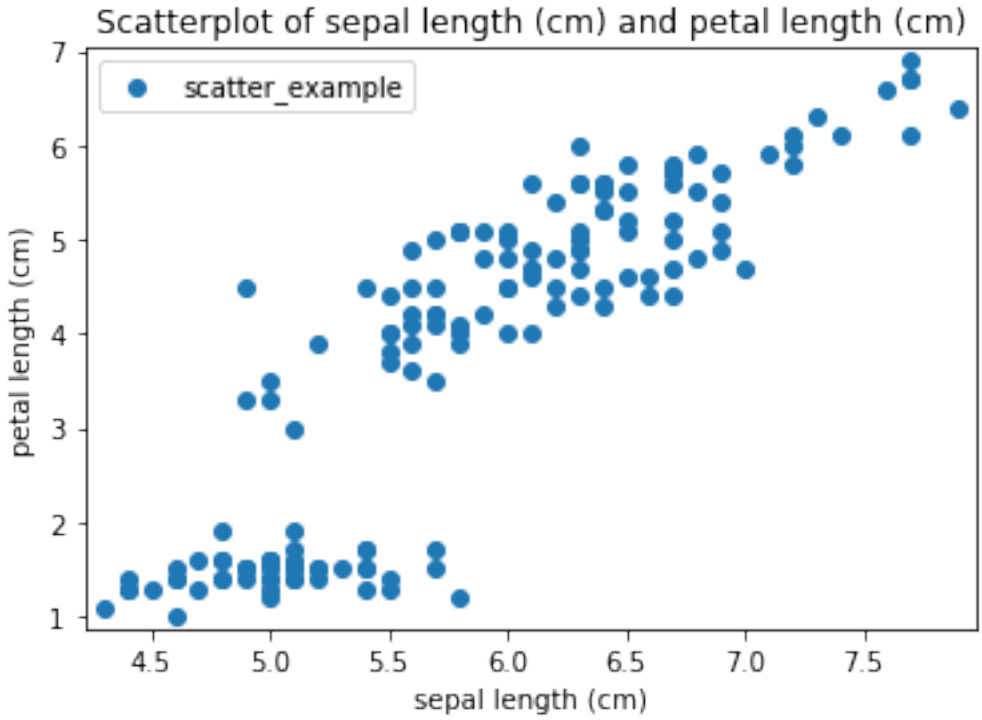
```
In [4]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('scatter', ds.eda.ScatterPlots(), [("read", "df", "df")])
```

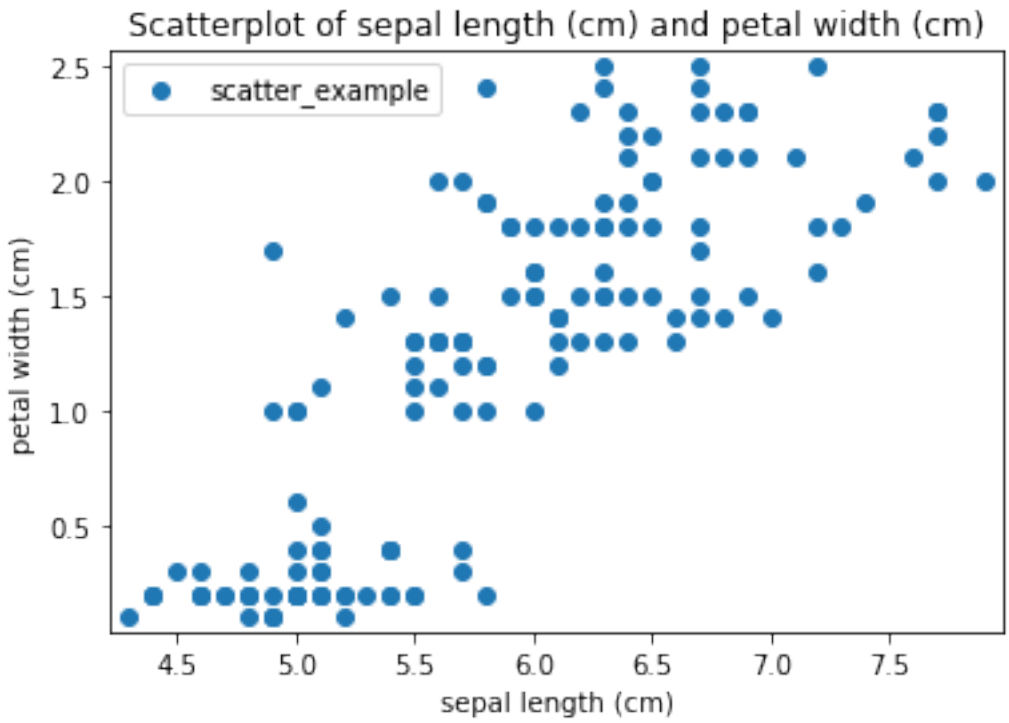
```
        p.transform(name="scatter_example", close_plt=True)
'Drawing diagram using blockdiag'
```

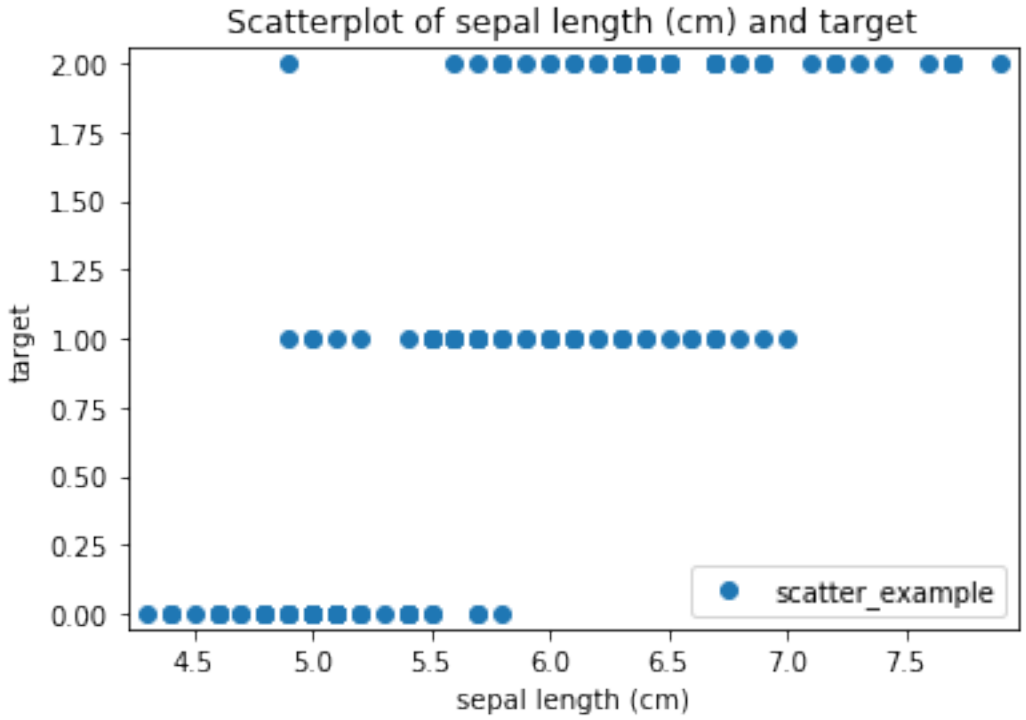


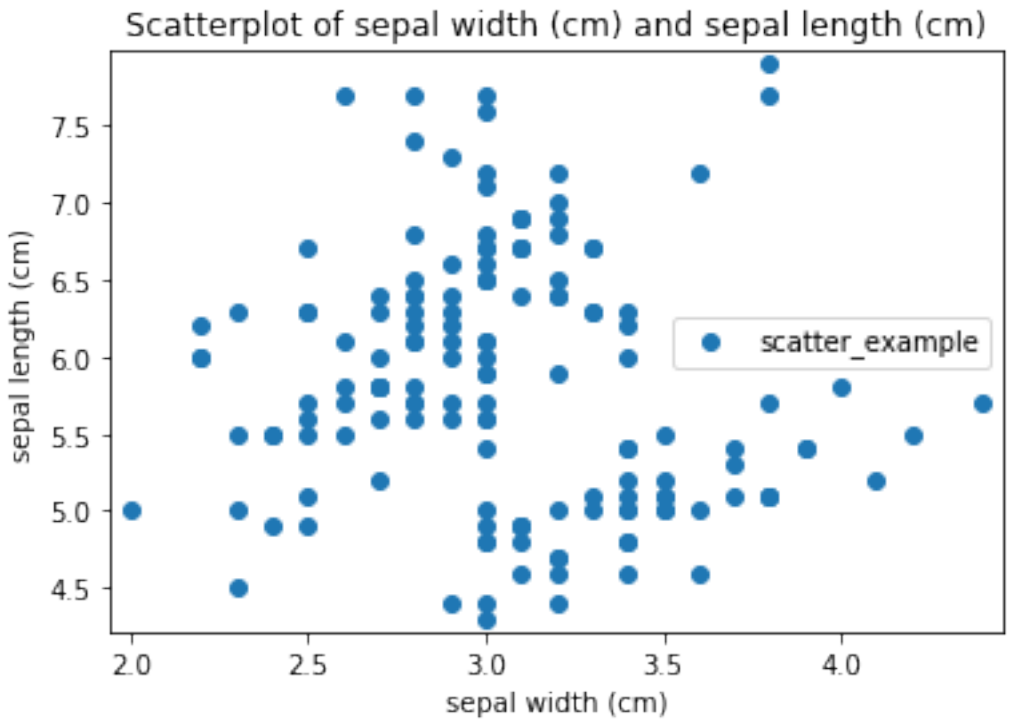

```
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
<IPython.core.display.HTML object>
```

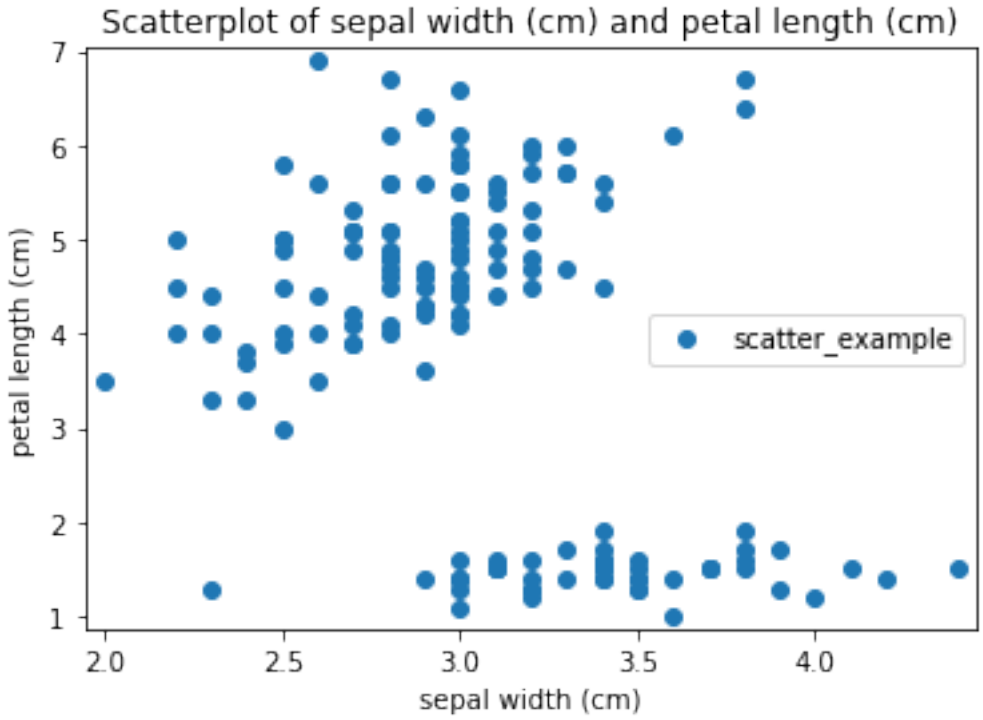

Scatterplot of sepal length (cm) and sepal width (cm)
scatter_example
sepal width (cm)
sepal length (cm)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5

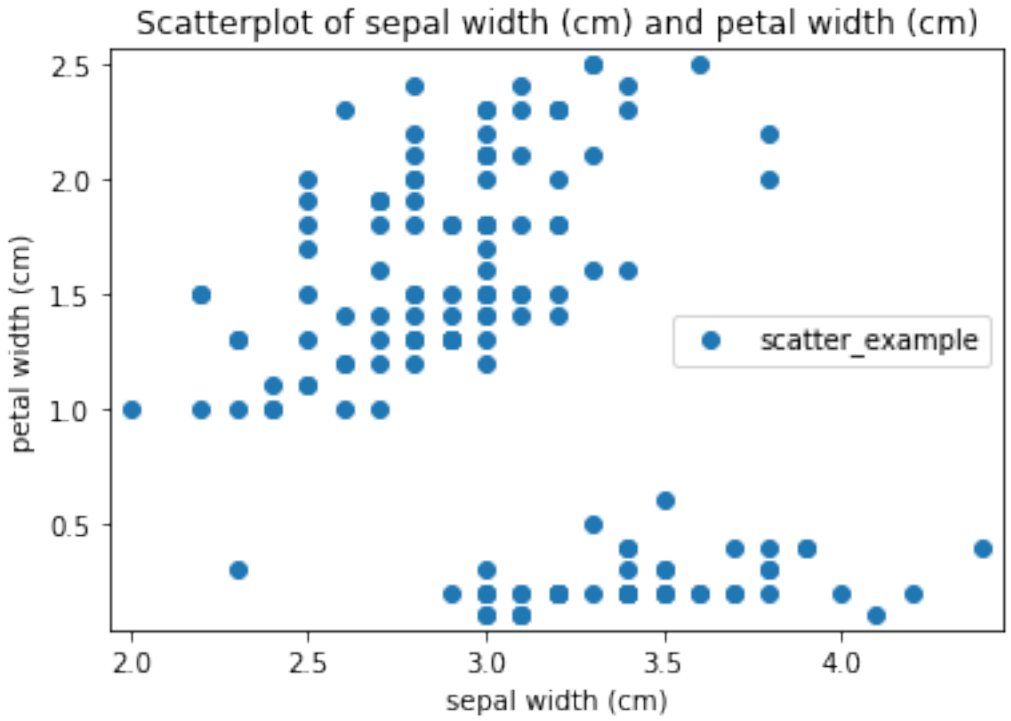

Scatterplot of sepal length (cm) and petal length (cm)
scatter_example
petal length (cm)
sepal length (cm)
7
6
5
4
3
2
1
4.5
5.0
5.5
6.0
6.5
7.0
7.5

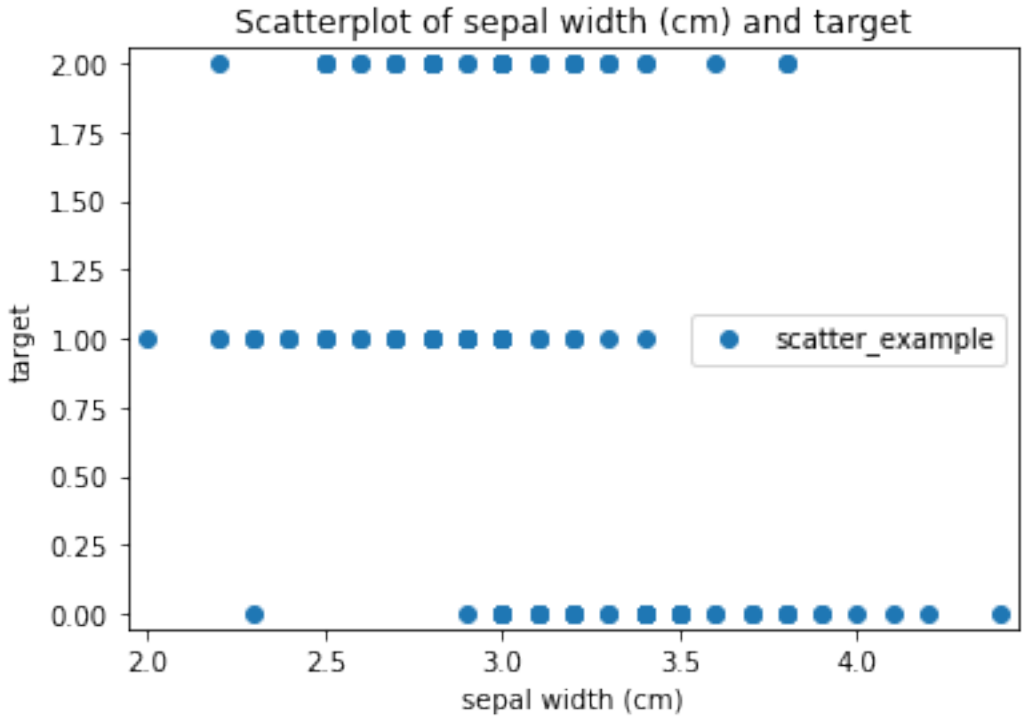

Scatterplot of sepal length (cm) and petal width (cm)
scatter_example
petal width (cm)
sepal length (cm)
2.5
2.0
1.5
1.0
0.5
4.5
5.0
5.5
6.0
6.5
7.0
7.5

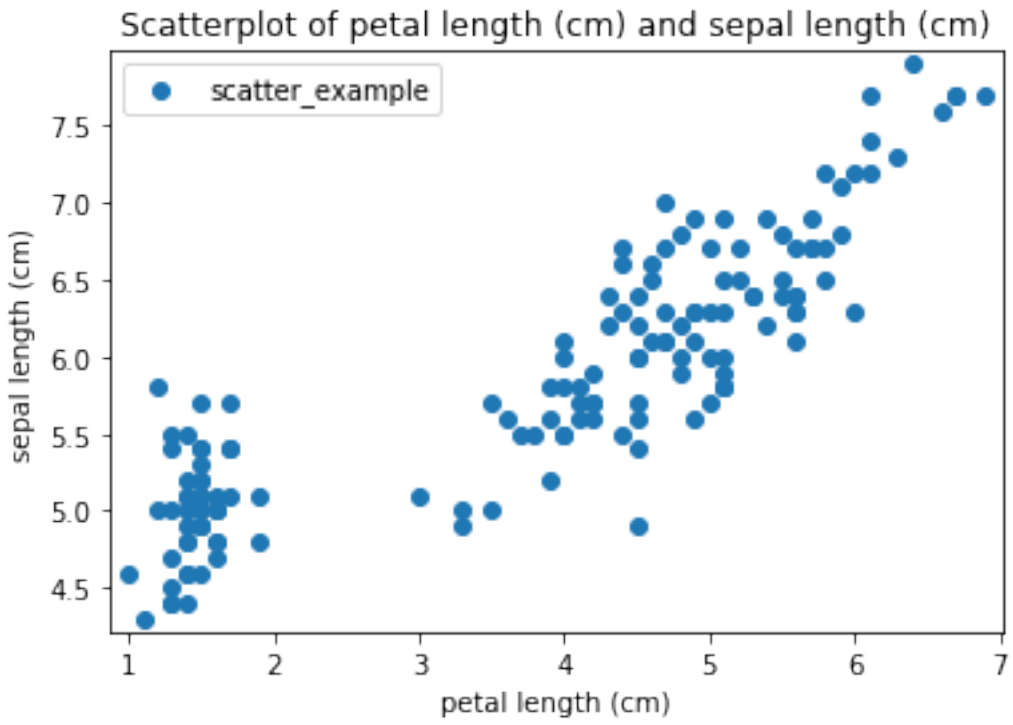

Scatterplot of sepal length (cm) and target
scatter_example
target
sepal length (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
4.5
5.0
5.5
6.0
6.5
7.0
7.5

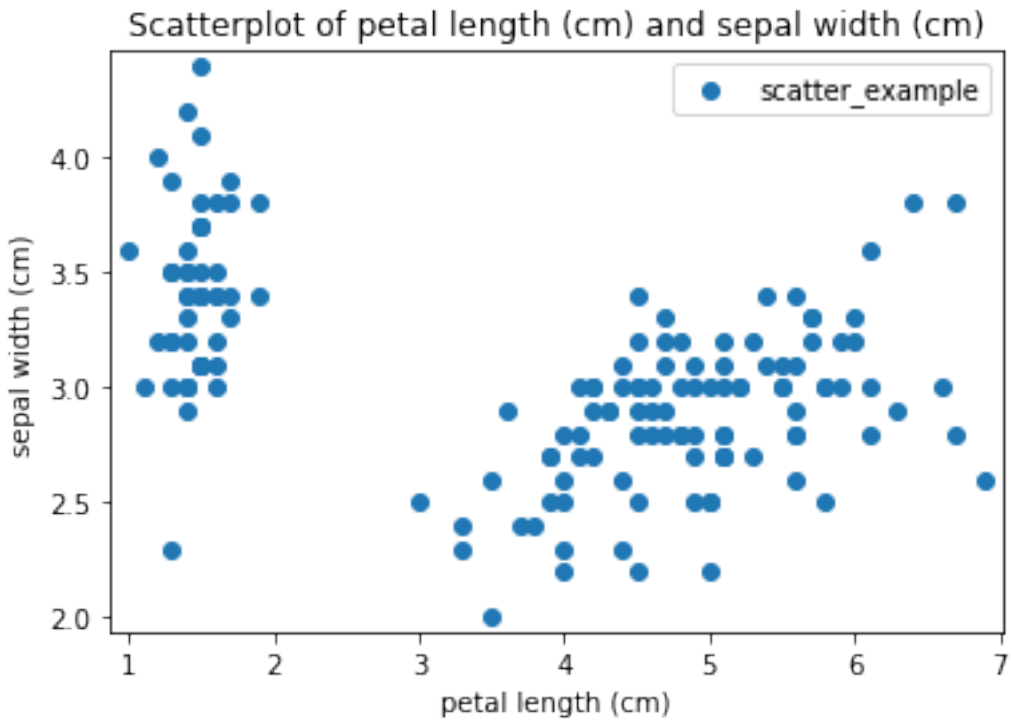

Scatterplot of sepal width (cm) and sepal length (cm)
sepal length (cm)
sepal width (cm)
scatter_example
7.5
7.0
6.5
6.0
5.5
5.0
4.5
2.0
2.5
3.0
3.5
4.0

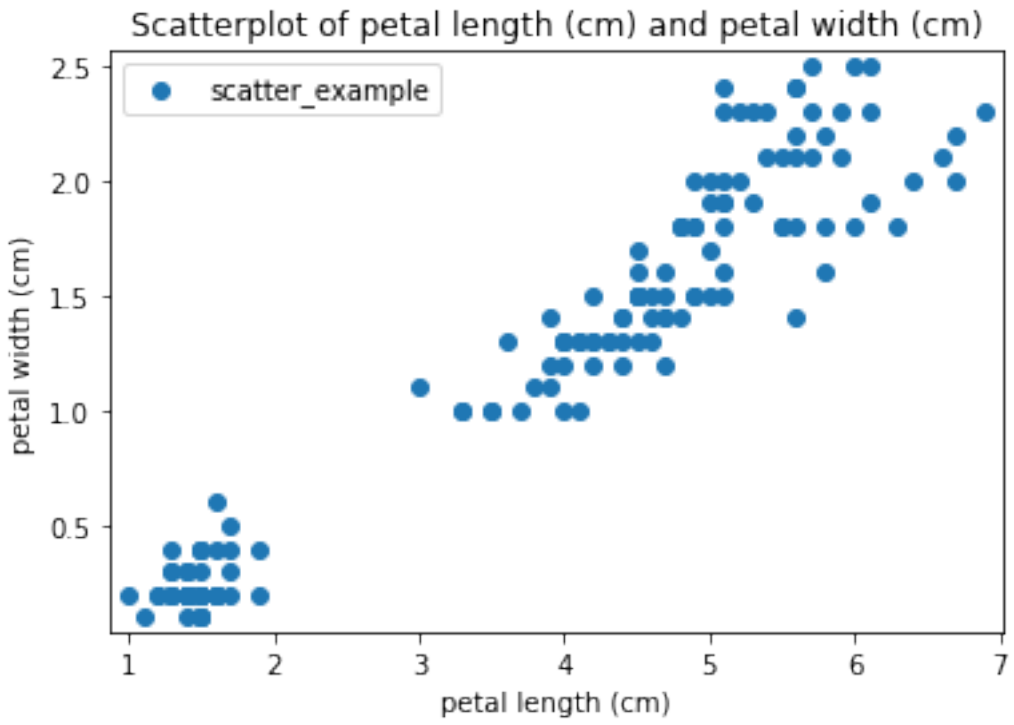

Scatterplot of sepal width (cm) and petal length (cm)
petal length (cm)
sepal width (cm)
scatter_example
7
6
5
4
3
2
1
2.0
2.5
3.0
3.5
4.0

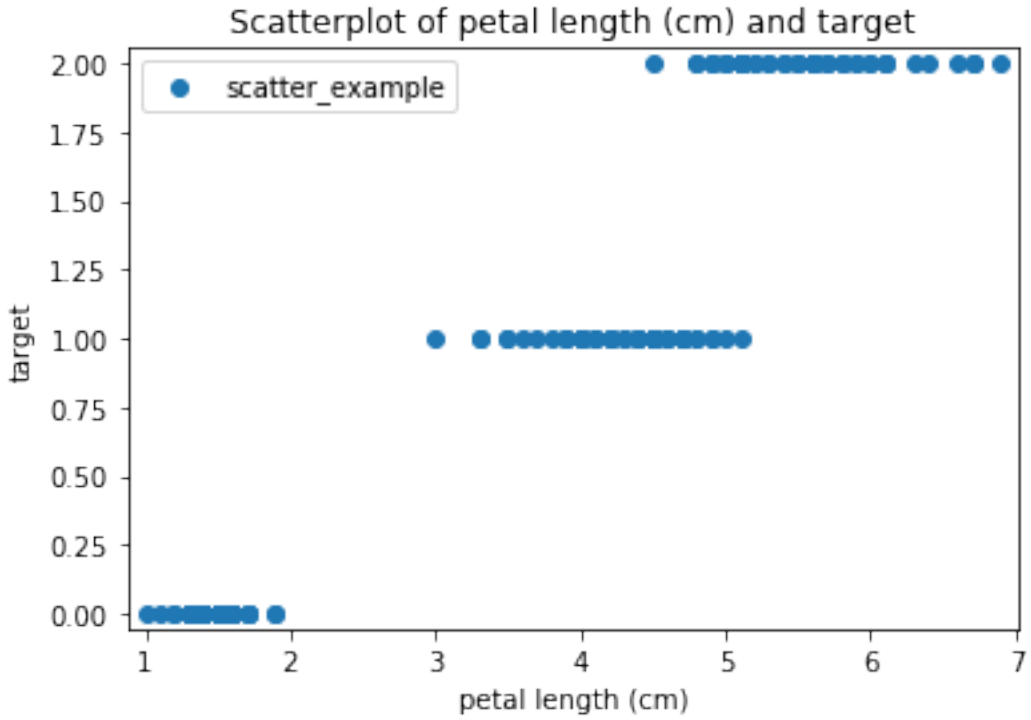

Scatterplot of sepal width (cm) and petal width (cm)
petal width (cm)
sepal width (cm)
scatter_example
2.5
2.0
1.5
1.0
0.5
2.0
2.5
3.0
3.5
4.0

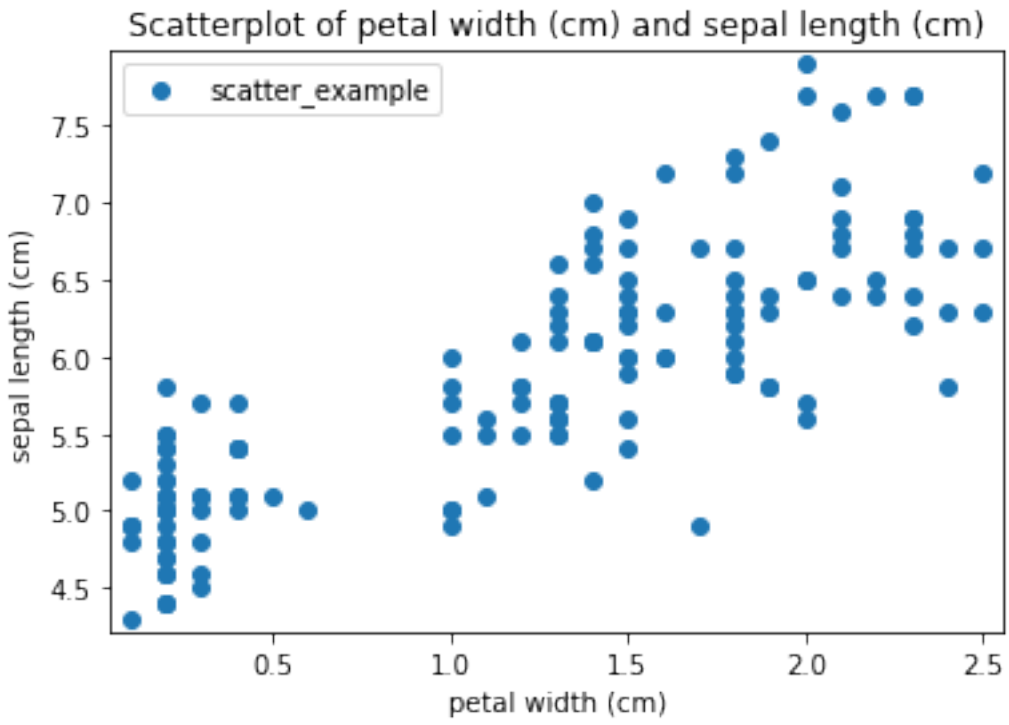

Scatterplot of sepal width (cm) and target
target
sepal width (cm)
scatter_example
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2.0
2.5
3.0
3.5
4.0

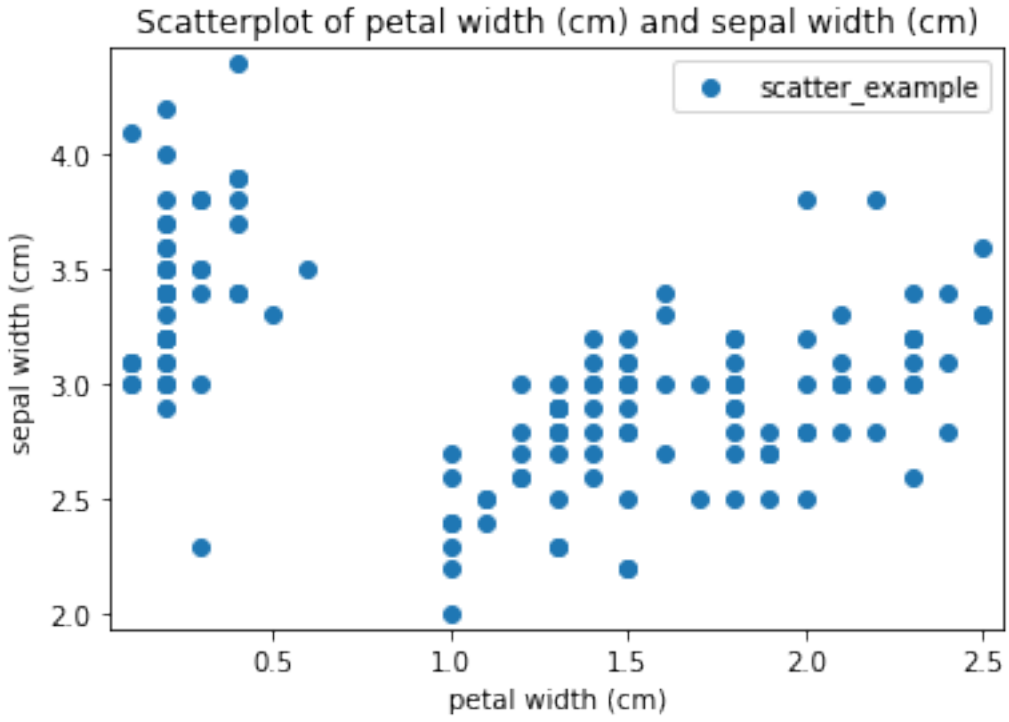

Scatterplot of petal length (cm) and sepal length (cm)
scatter_example
sepal length (cm)
petal length (cm)
4.5
5.0
5.5
6.0
6.5
7.0
7.5
1
2
3
4
5
6
7

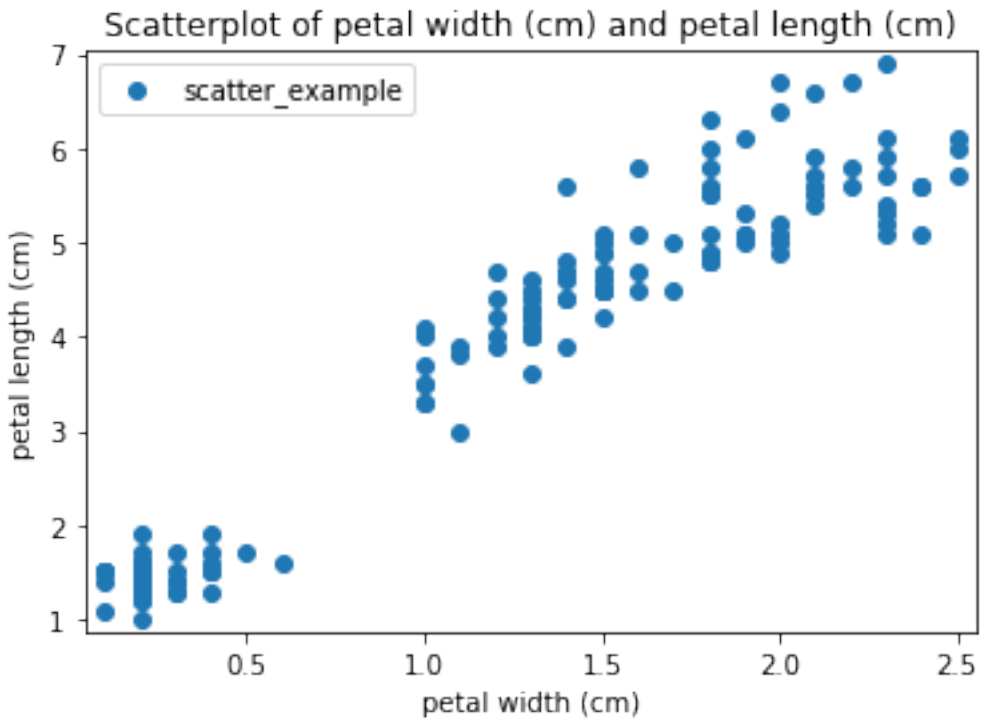

Scatterplot of petal length (cm) and sepal width (cm)
scatter_example
sepal width (cm)
petal length (cm)
2.0
2.5
3.0
3.5
4.0
1
2
3
4
5
6
7

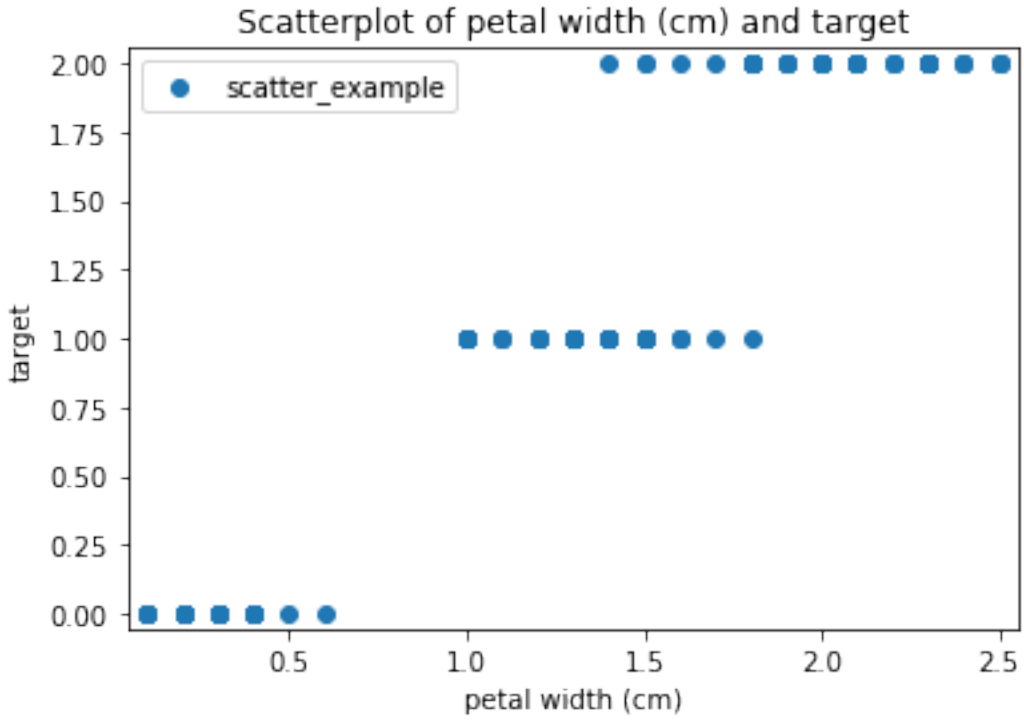

Scatterplot of petal length (cm) and petal width (cm)
scatter_example
petal width (cm)
petal length (cm)
2.5
2.0
1.5
1.0
0.5
1
2
3
4
5
6
7

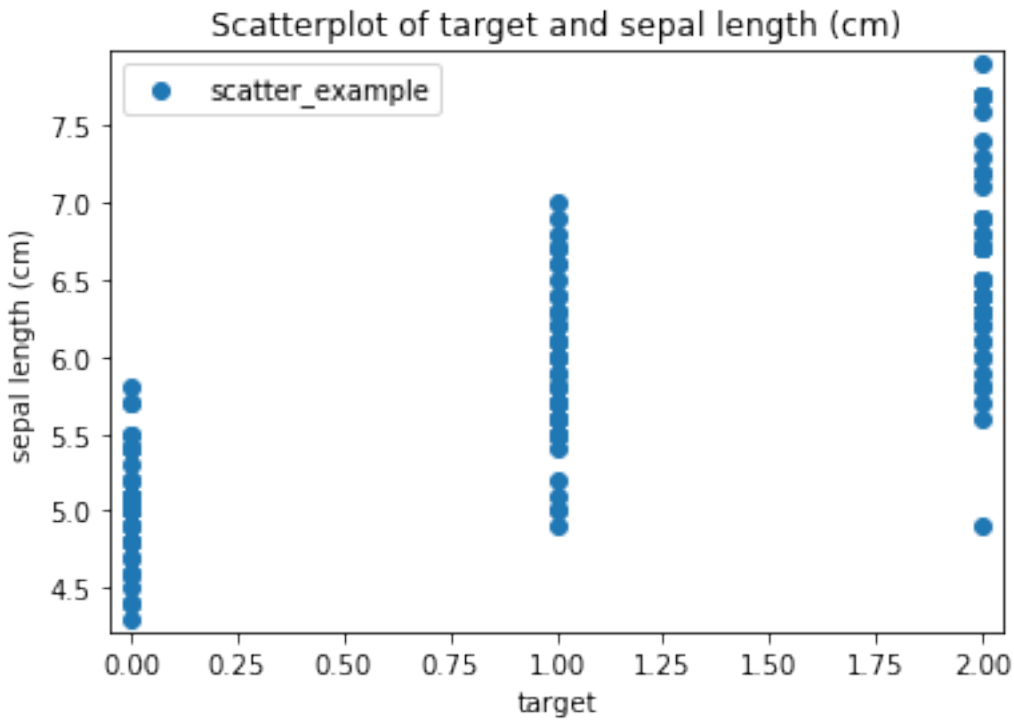

Scatterplot of petal length (cm) and target
scatter_example
target
petal length (cm)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6
7

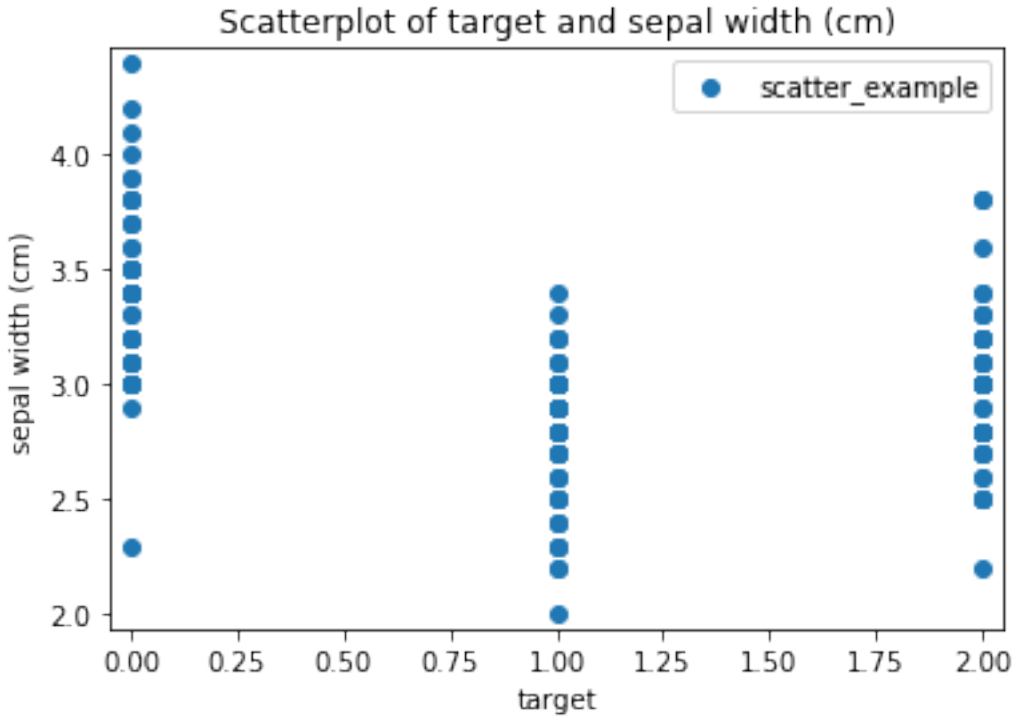

Scatterplot of petal width (cm) and sepal length (cm)
scatter_example
sepal length (cm)
petal width (cm)
4.5
5.0
5.5
6.0
6.5
7.0
7.5
0.5
1.0
1.5
2.0
2.5

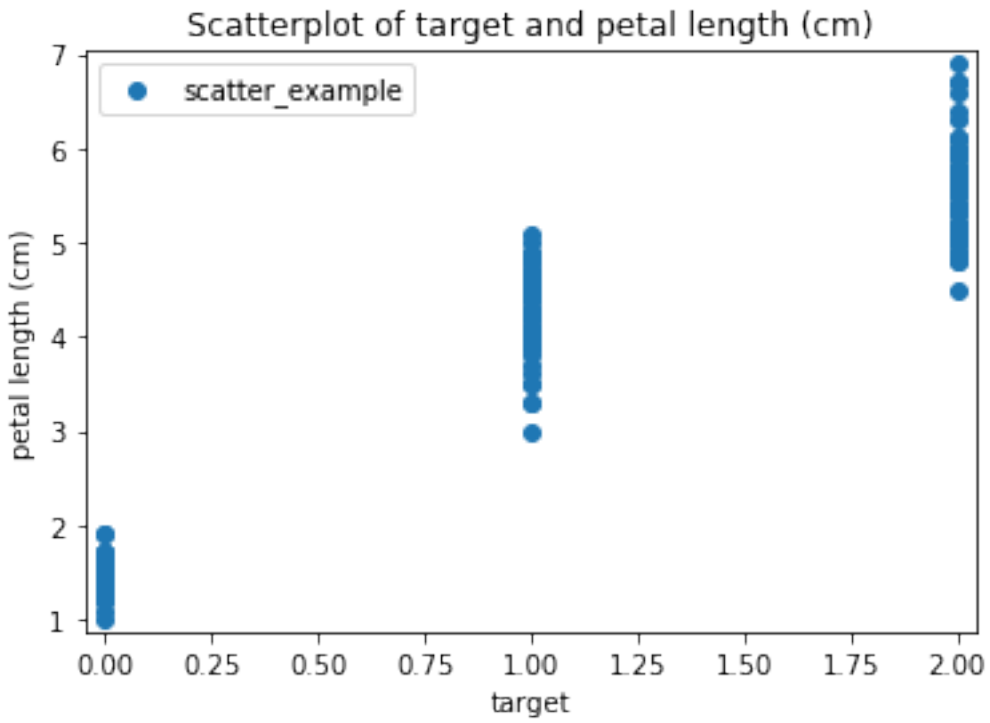

Scatterplot of petal width (cm) and sepal width (cm)
scatter_example
sepal width (cm)
petal width (cm)
2.0
2.5
3.0
3.5
4.0
0.5
1.0
1.5
2.0
2.5

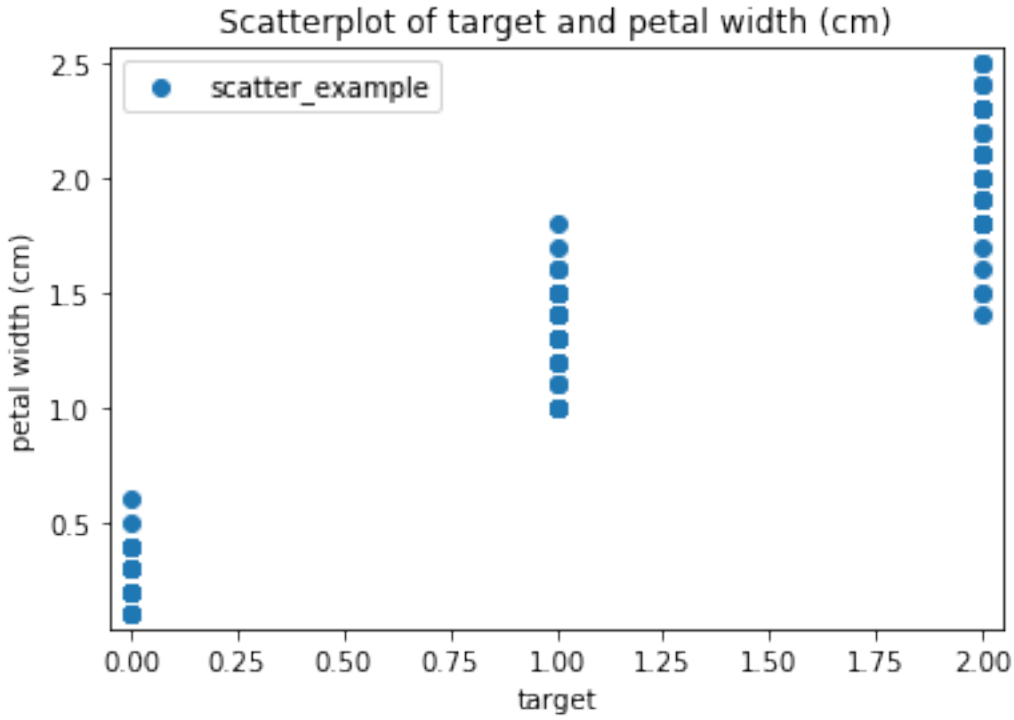

Scatterplot of petal width (cm) and petal length (cm)
scatter_example
petal length (cm)
petal width (cm)
1
2
3
4
5
6
7
0.5
1.0
1.5
2.0
2.5


Scatterplot of petal width (cm) and target
scatter_example
target
petal width (cm)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.5
1.0
1.5
2.0
2.5


Scatterplot of target and sepal length (cm)
scatter_example
sepal length (cm)
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
target


Scatterplot of target and sepal width (cm)
scatter_example
sepal width (cm)
4.0
3.5
3.0
2.5
2.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
target

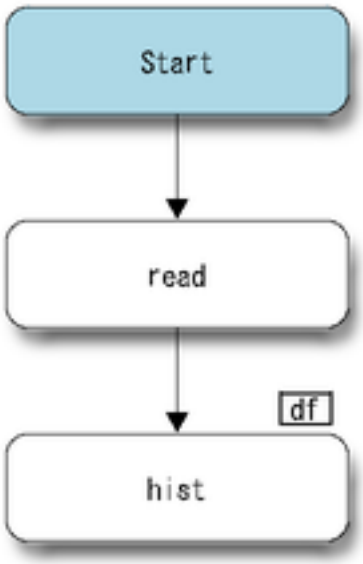




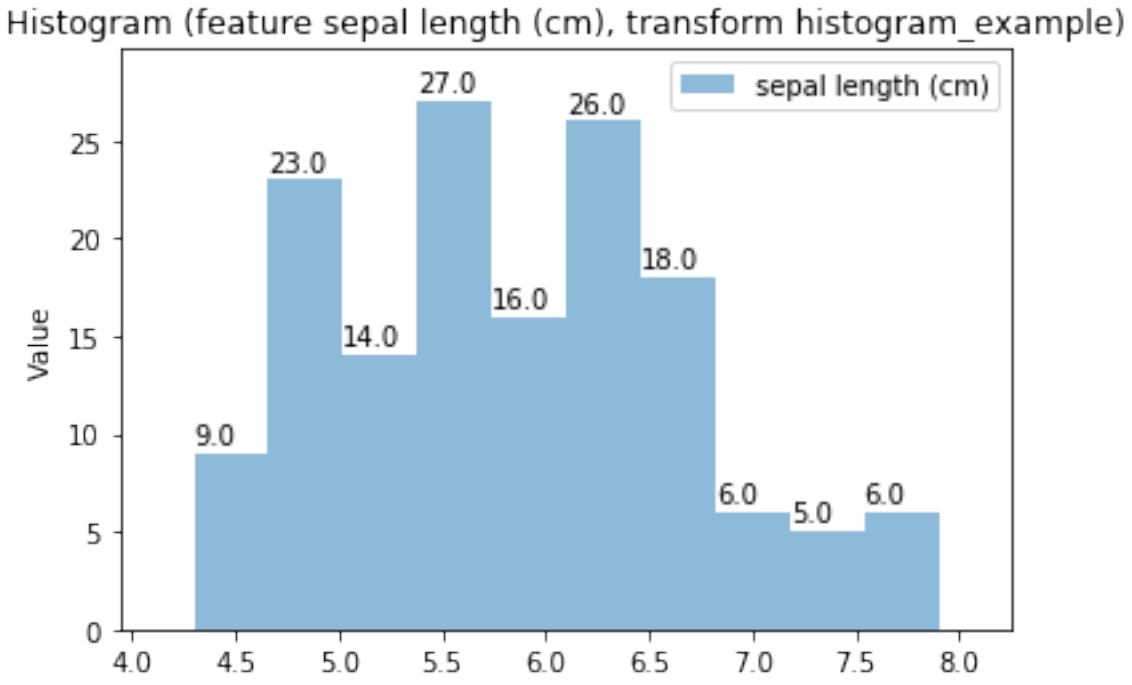
Histogram plots

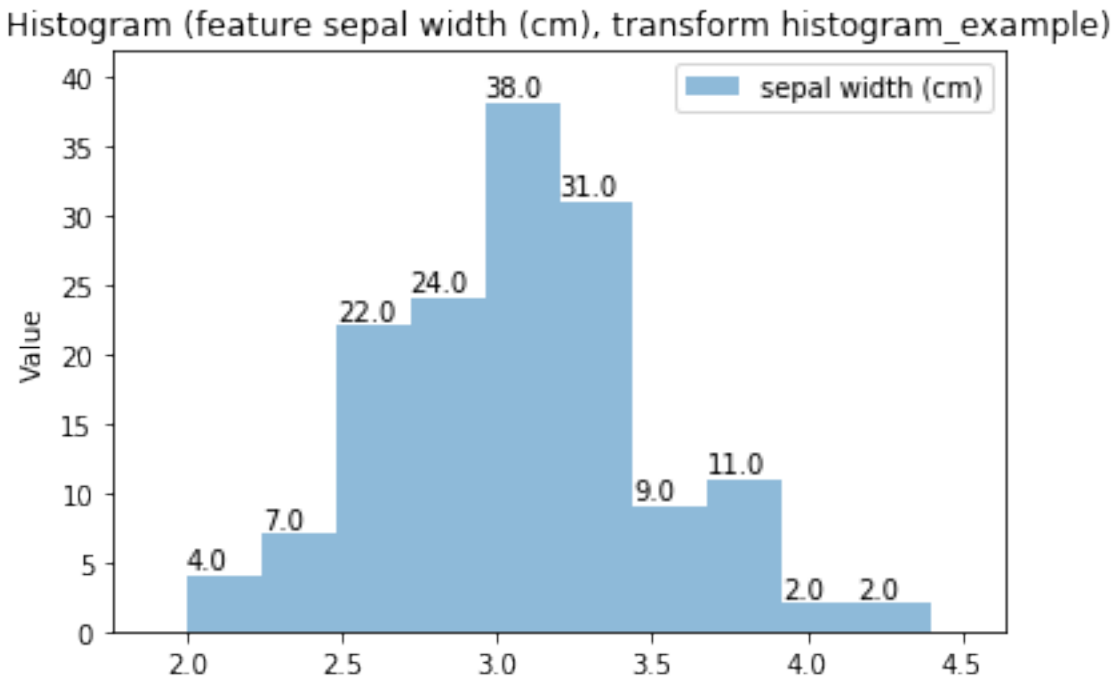
```
In [5]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('hist', ds.eda.Hist(), [("read", "df", "df")])
```

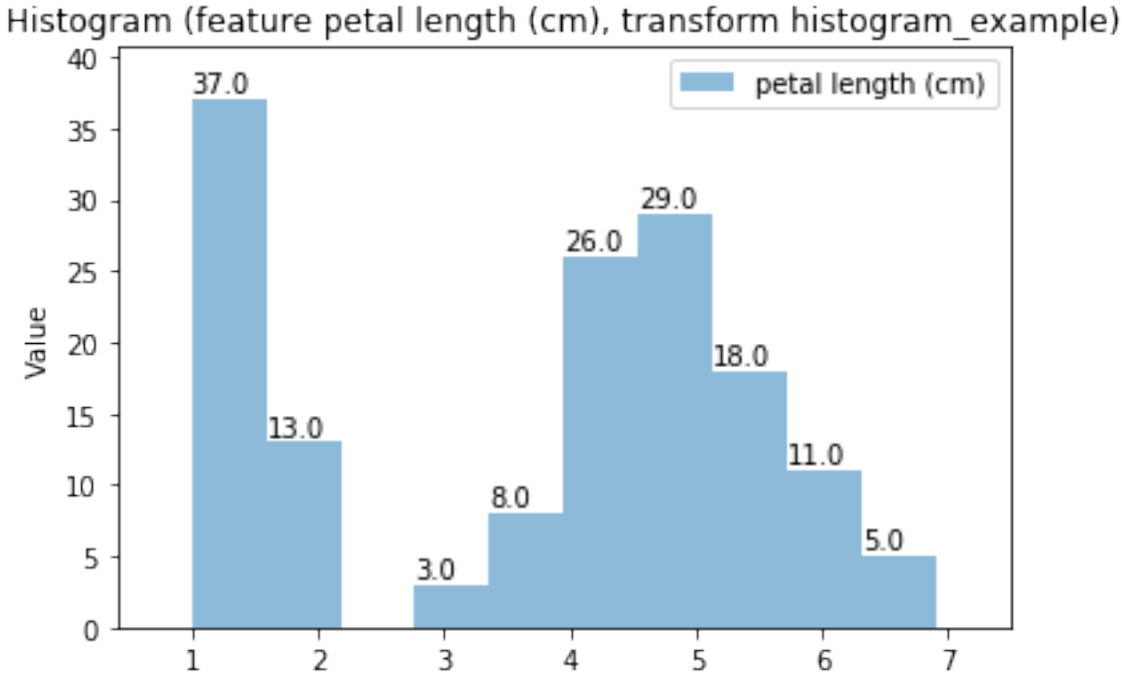
```
        p.transform(name="histogram_example", close_plt=True)
'Drawing diagram using blockdiag'
```

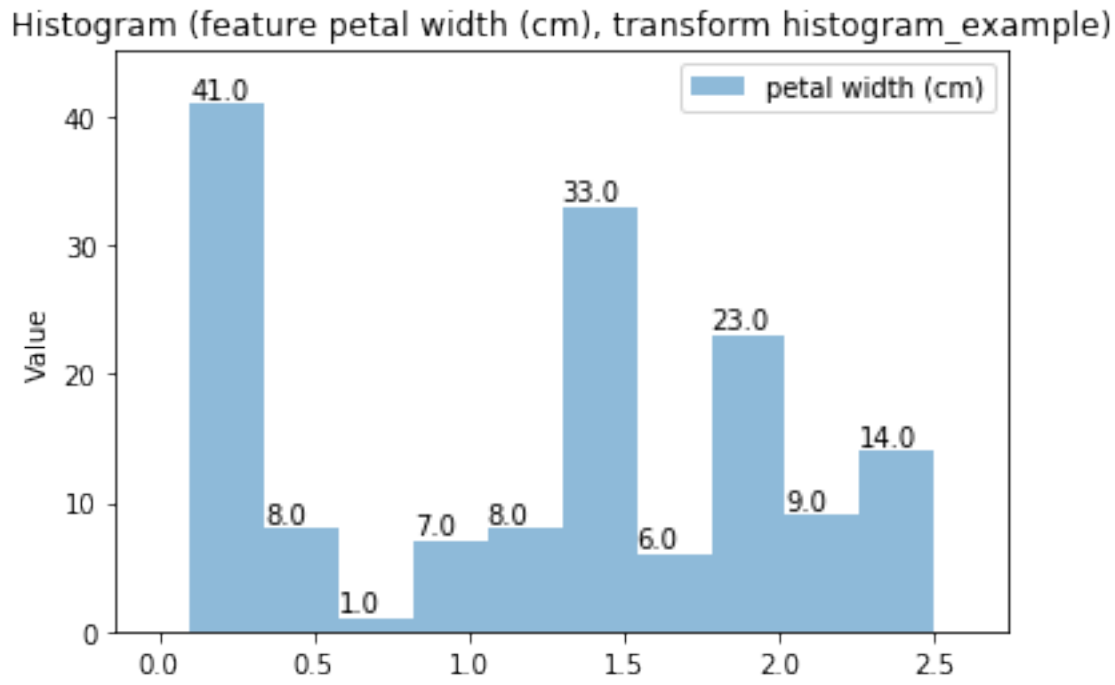


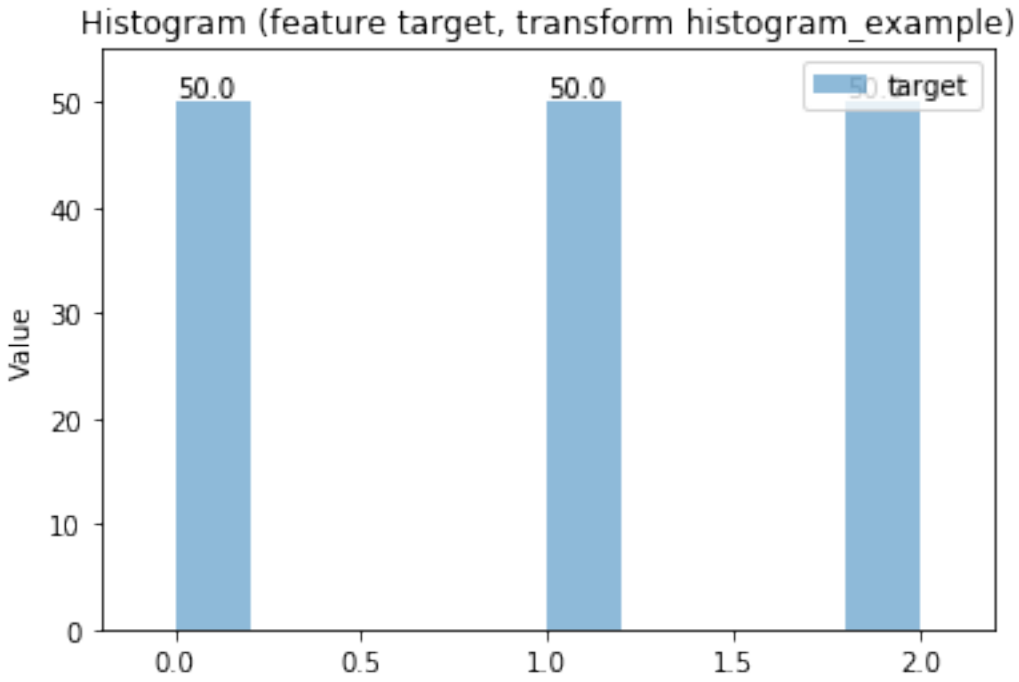

```
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
<IPython.core.display.HTML object>
```




Histogram (feature sepal width (cm), transform histogram_example)
sepal width (cm)
Value
40
35
30
25
20
15
10
5
0
38.0
31.0
24.0
22.0
11.0
9.0
7.0
4.0
2.0
2.0
2.0
2.5
3.0
3.5
4.0
4.5


Histogram (feature petal length (cm), transform histogram_example)
petal length (cm)
Value
40
35
30
25
20
15
10
5
0
37.0
29.0
26.0
18.0
13.0
11.0
8.0
5.0
3.0
1
2
3
4
5
6
7

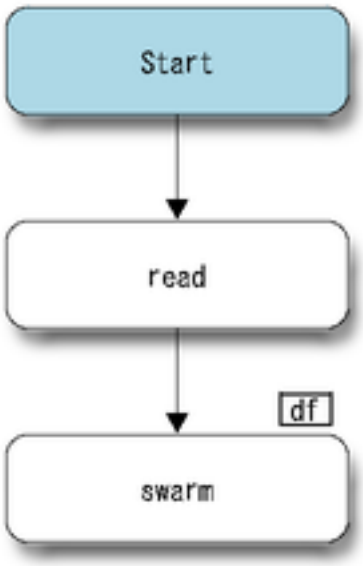




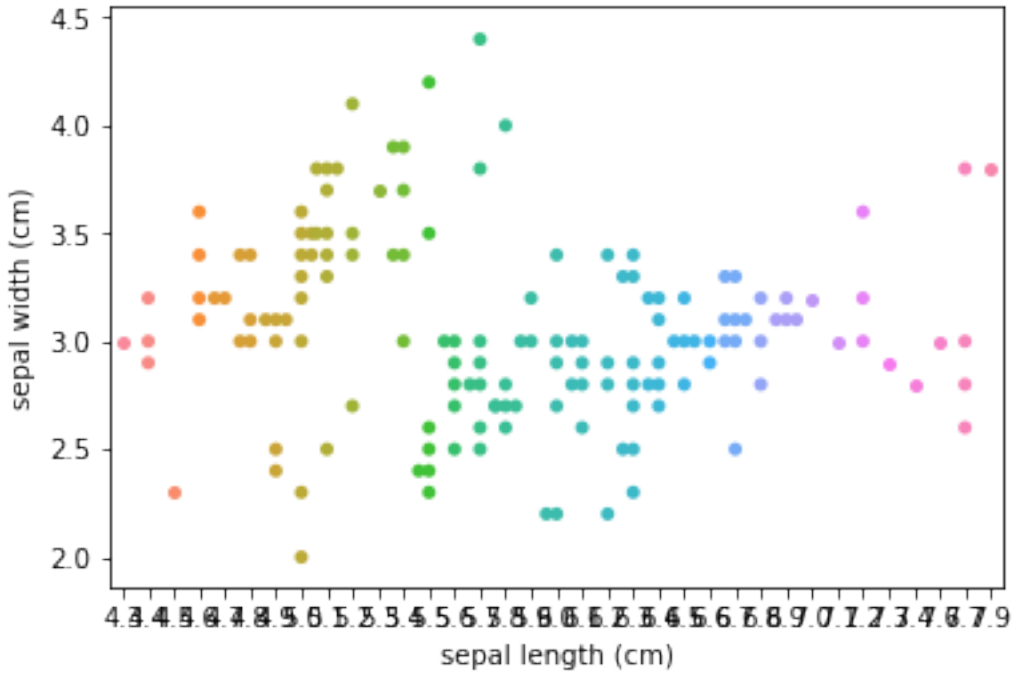
Swarm plots

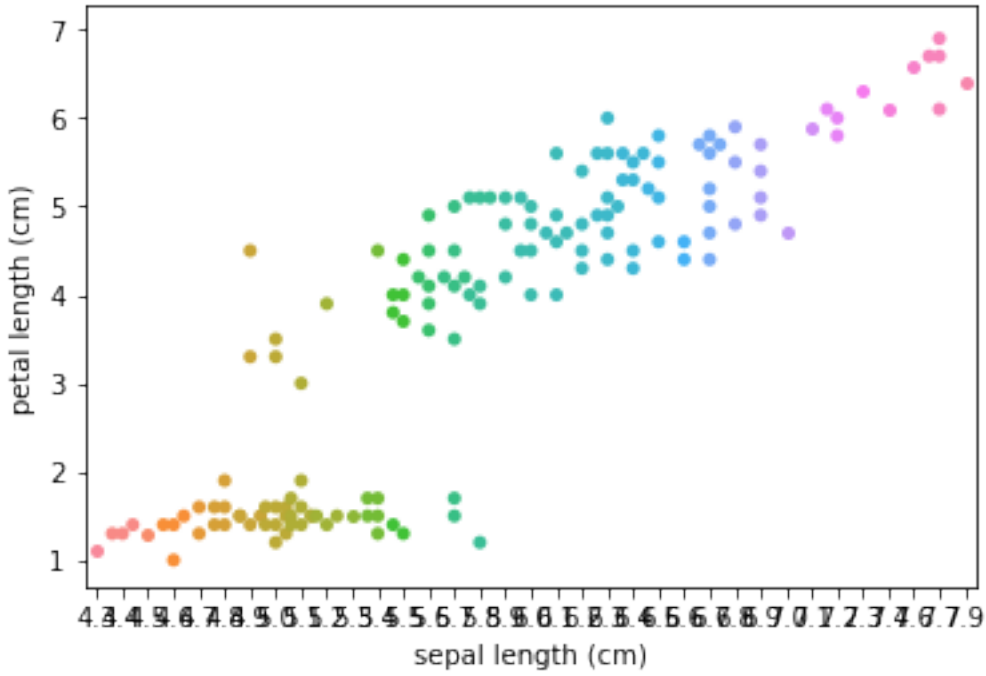
```
In [6]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('swarm', ds.eda.SwarmPlots(), [("read", "df", "df")])
        p.transform(name="swarm_example")
```

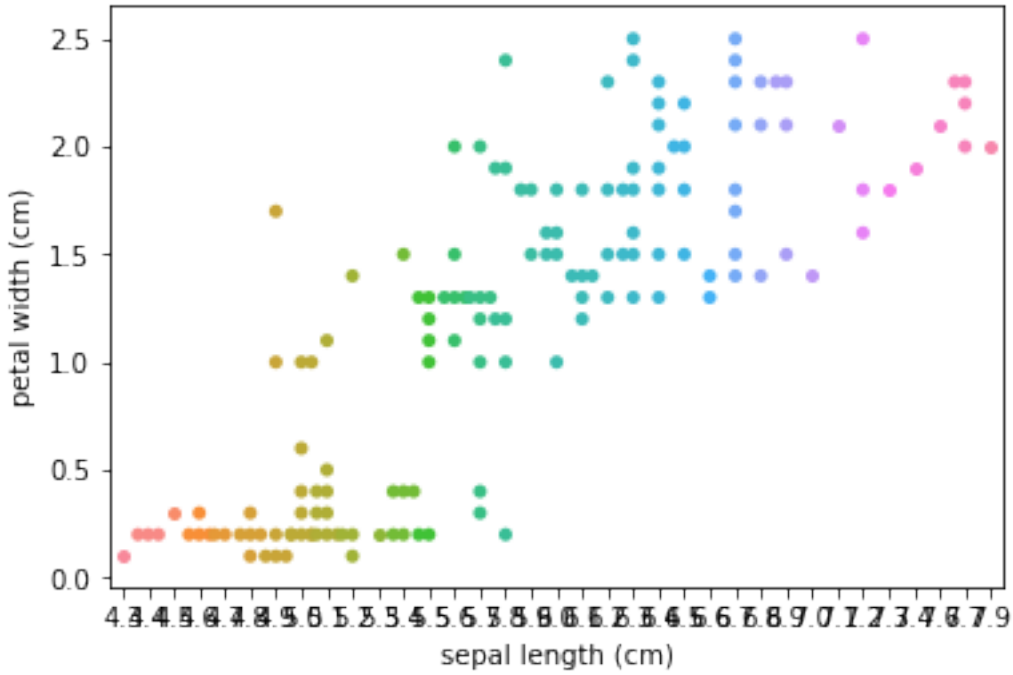
```
'Drawing diagram using blockdiag'
```

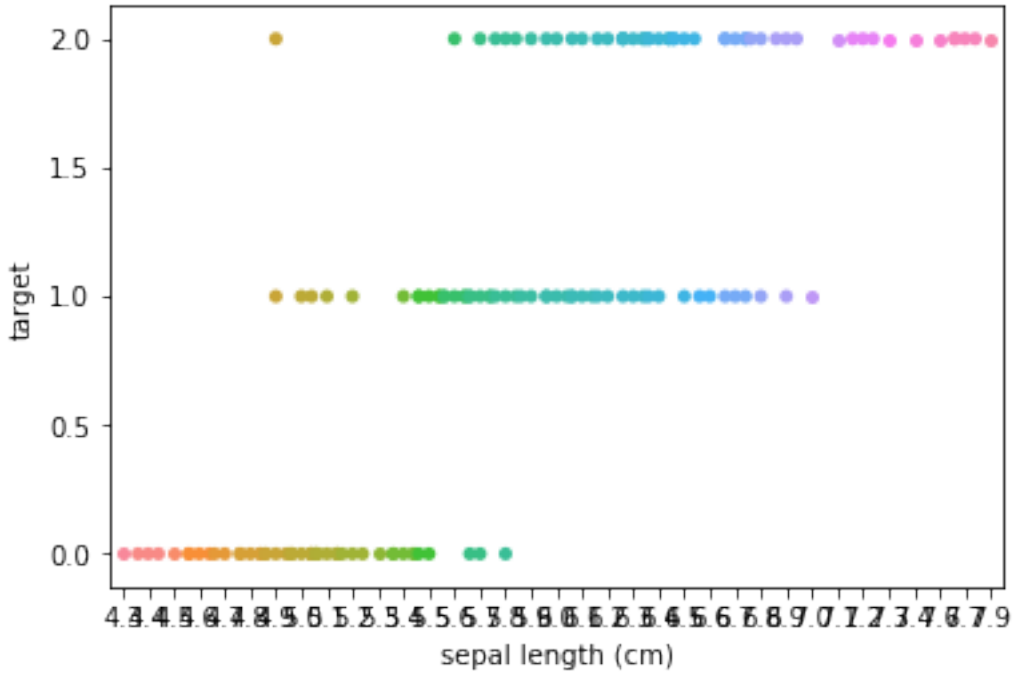


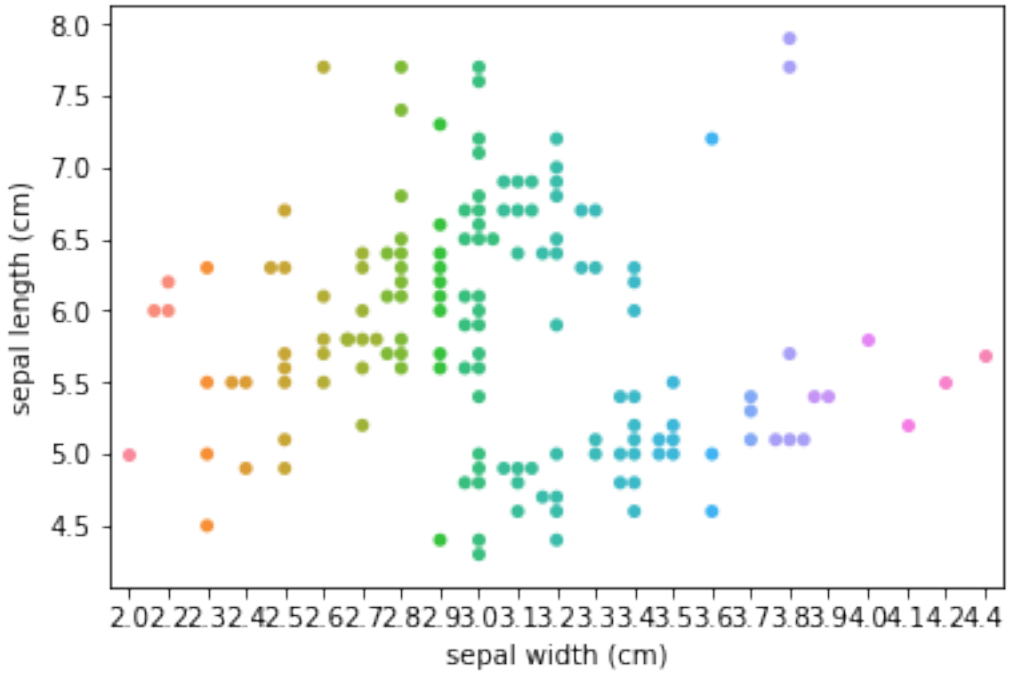

```
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
<IPython.core.display.HTML object>
```

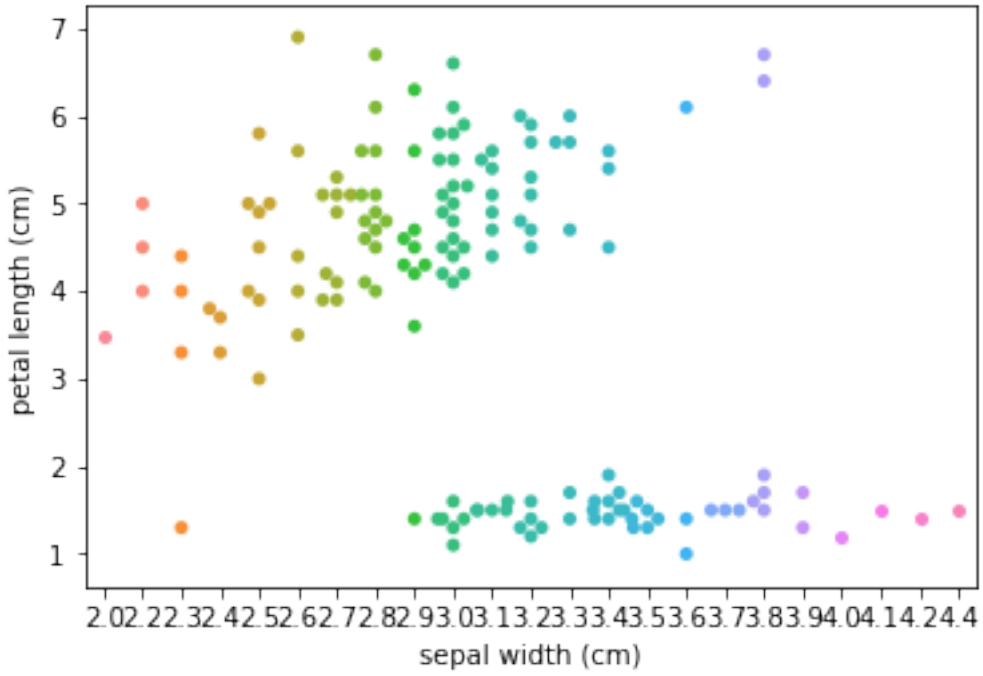


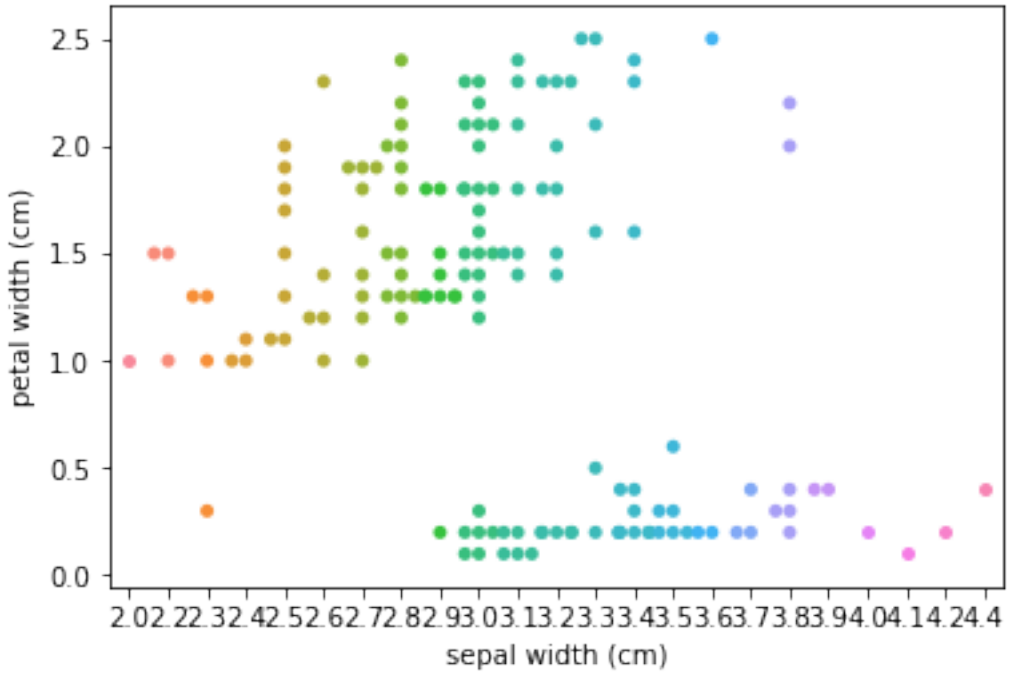

petal length (cm)
sepal length (cm)

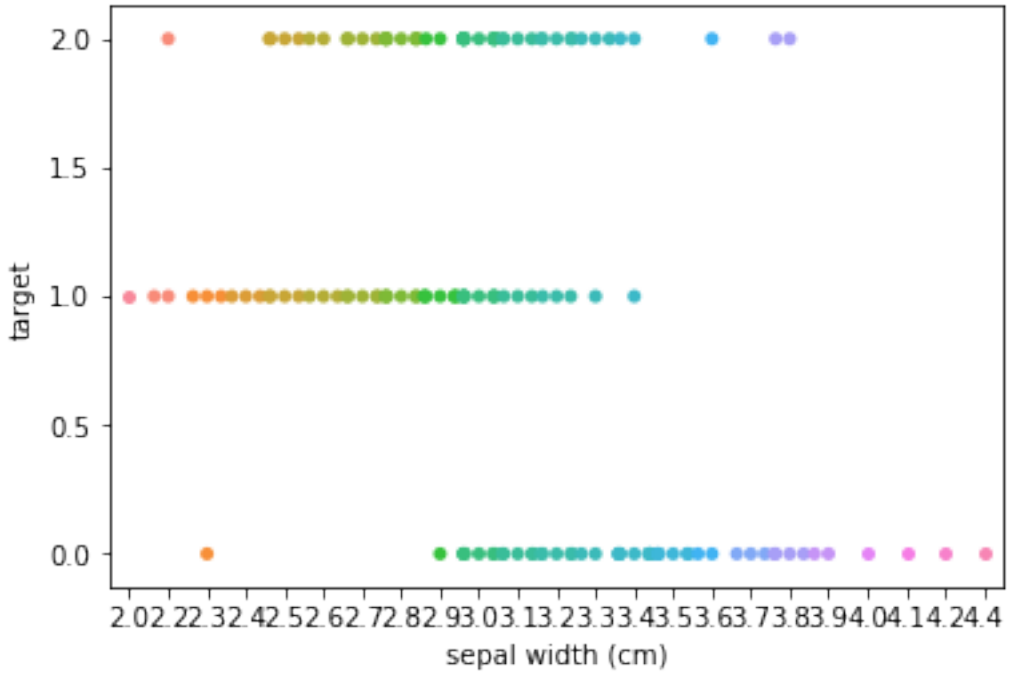

petal width (cm)
sepal length (cm)

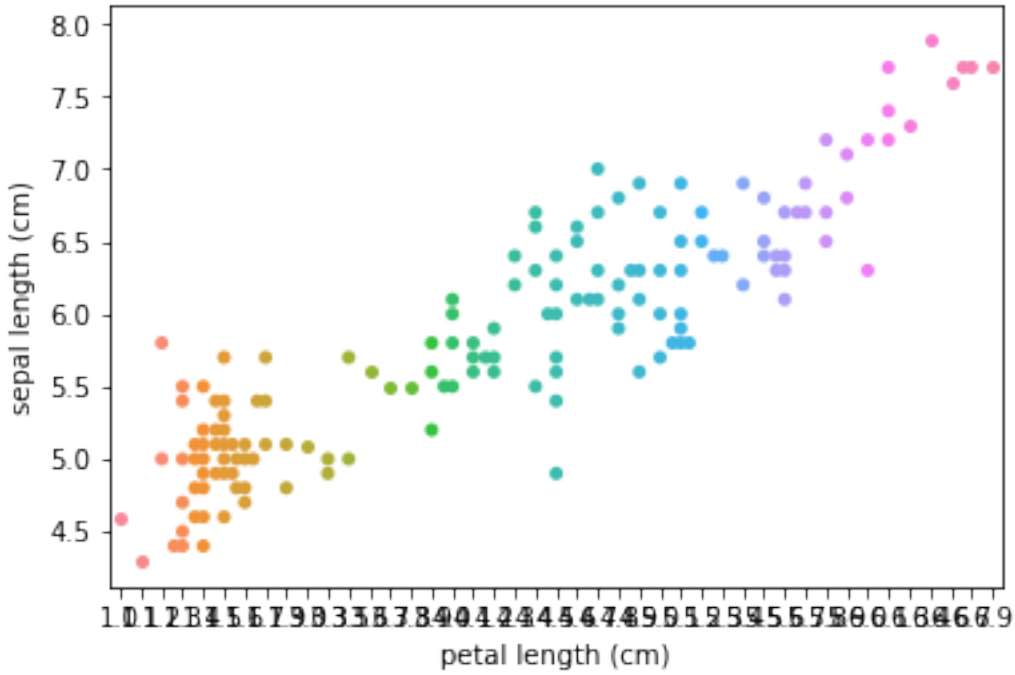

target
2.0
1.5
1.0
0.5
0.0
sepal length (cm)

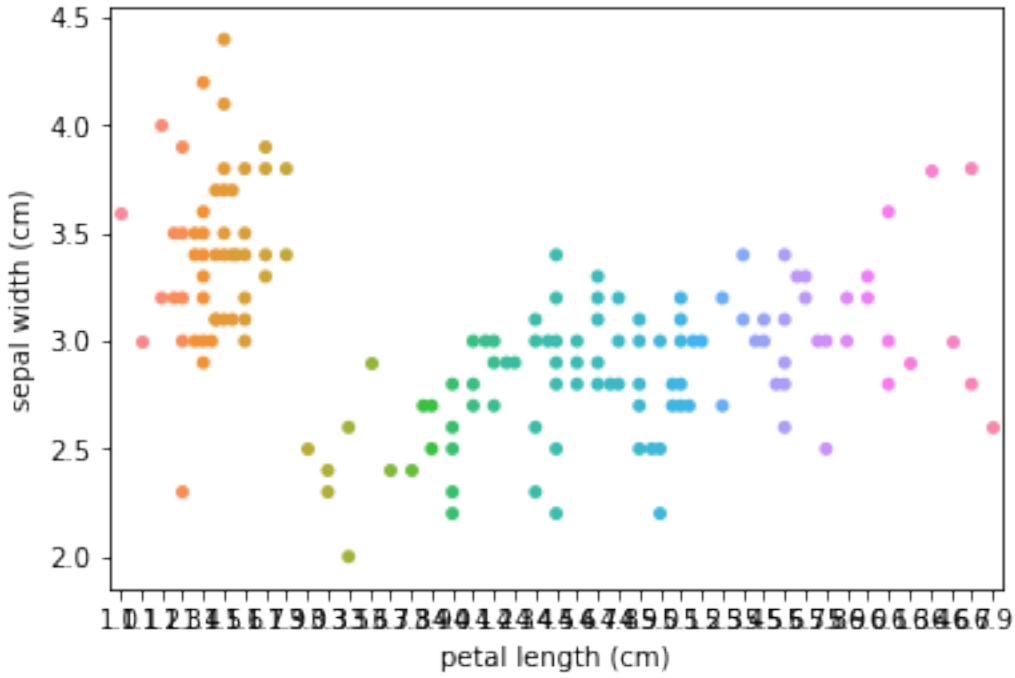

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
sepal width (cm)

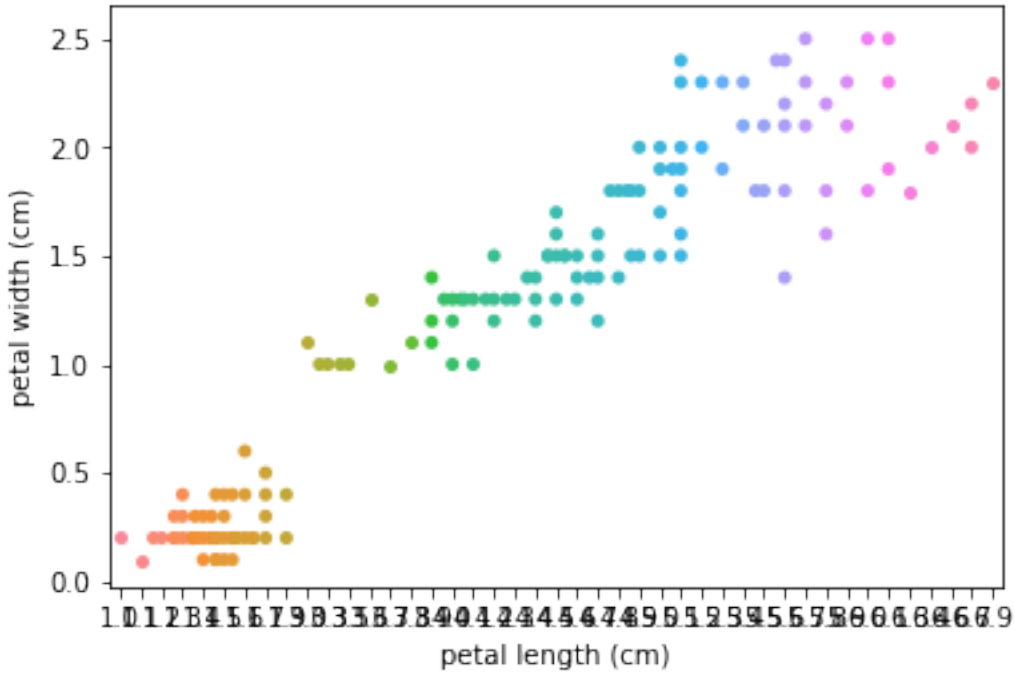

petal length (cm)
sepal width (cm)

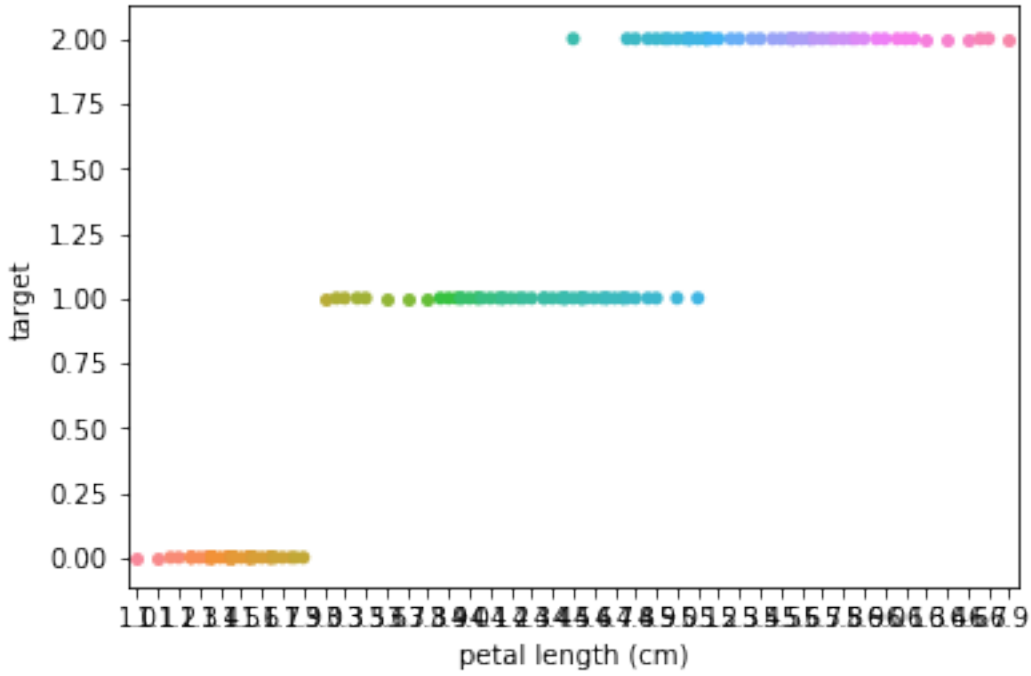

petal width (cm)
sepal width (cm)

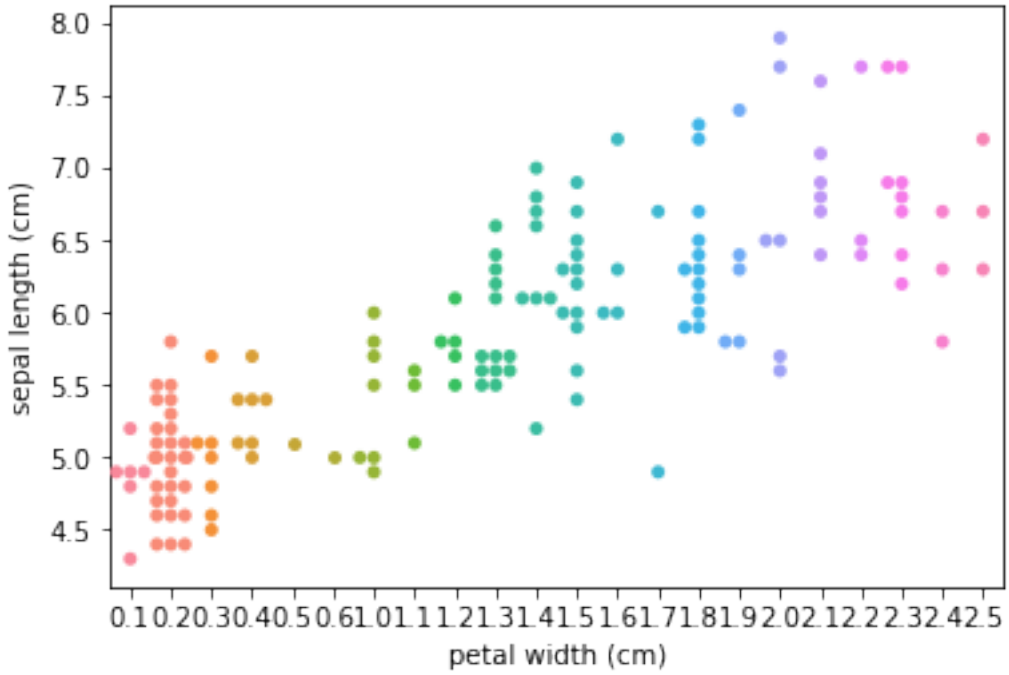

target
2.0
1.5
1.0
0.5
0.0
sepal width (cm)

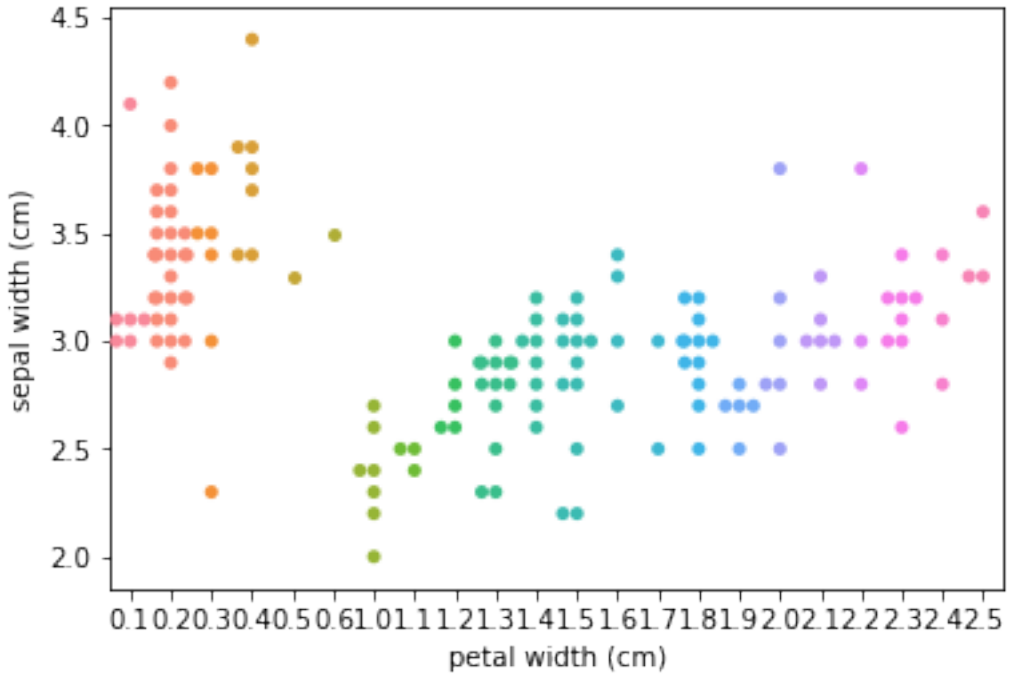

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
petal length (cm)

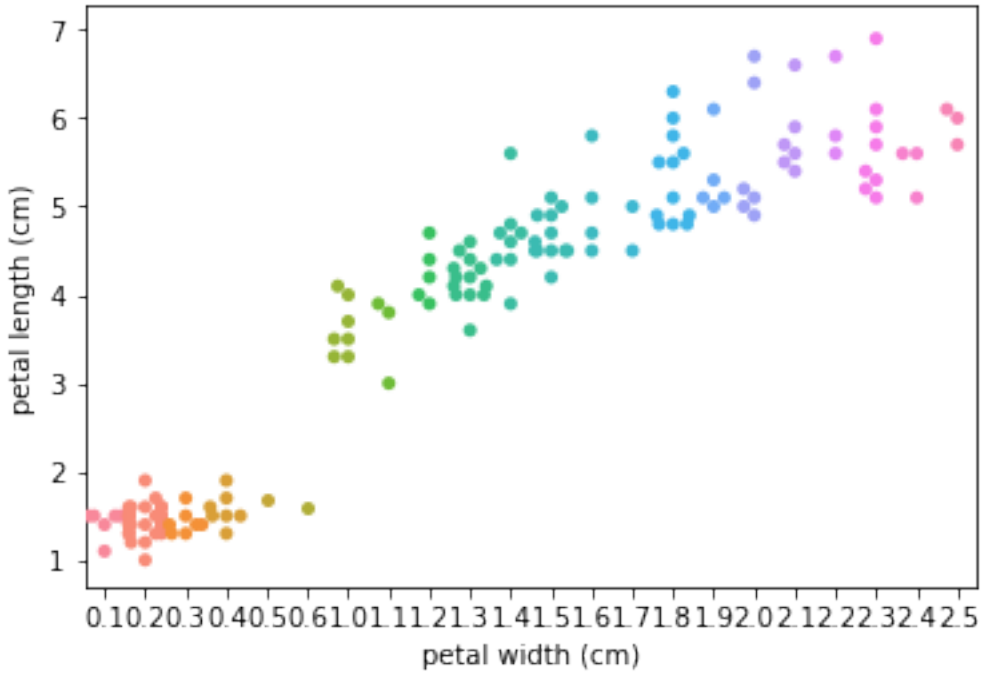

sepal width (cm)
petal length (cm)

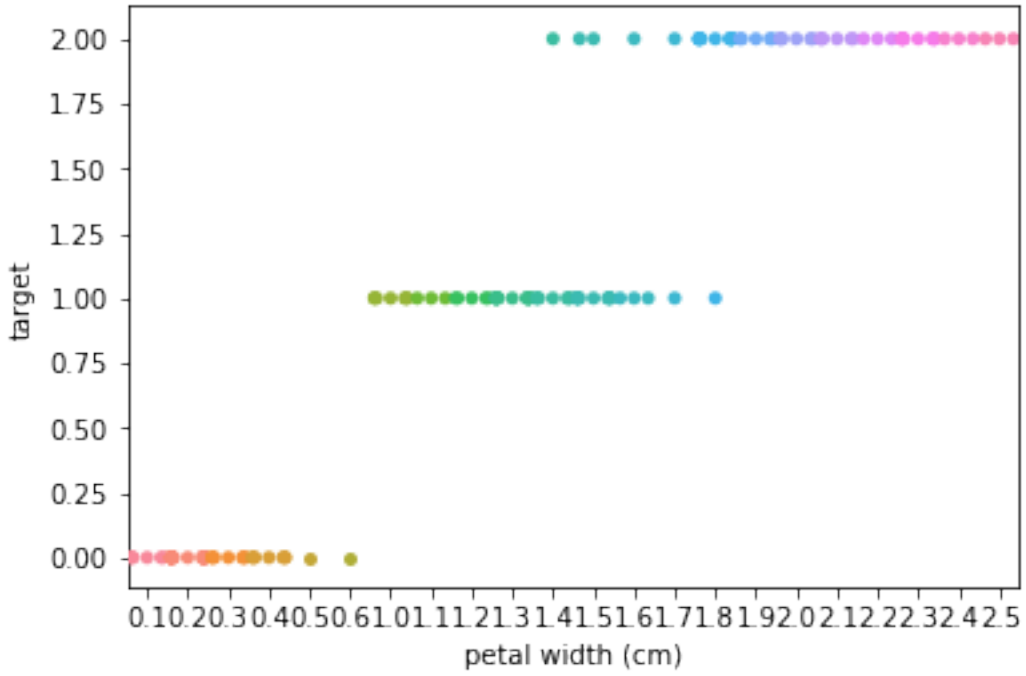

petal width (cm)
petal length (cm)

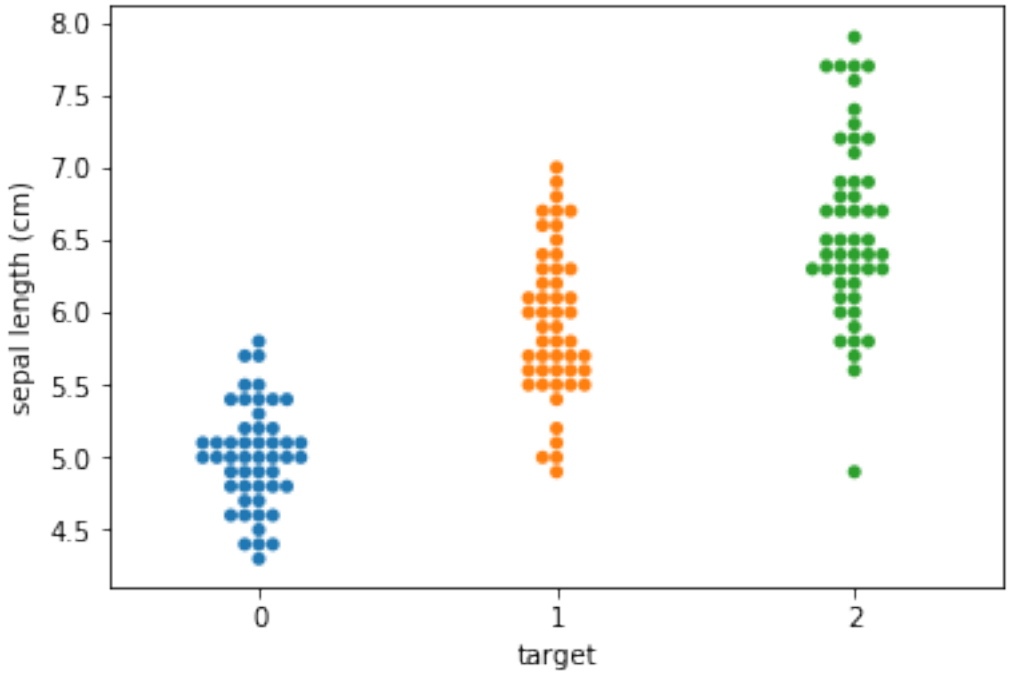

target
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
petal length (cm)

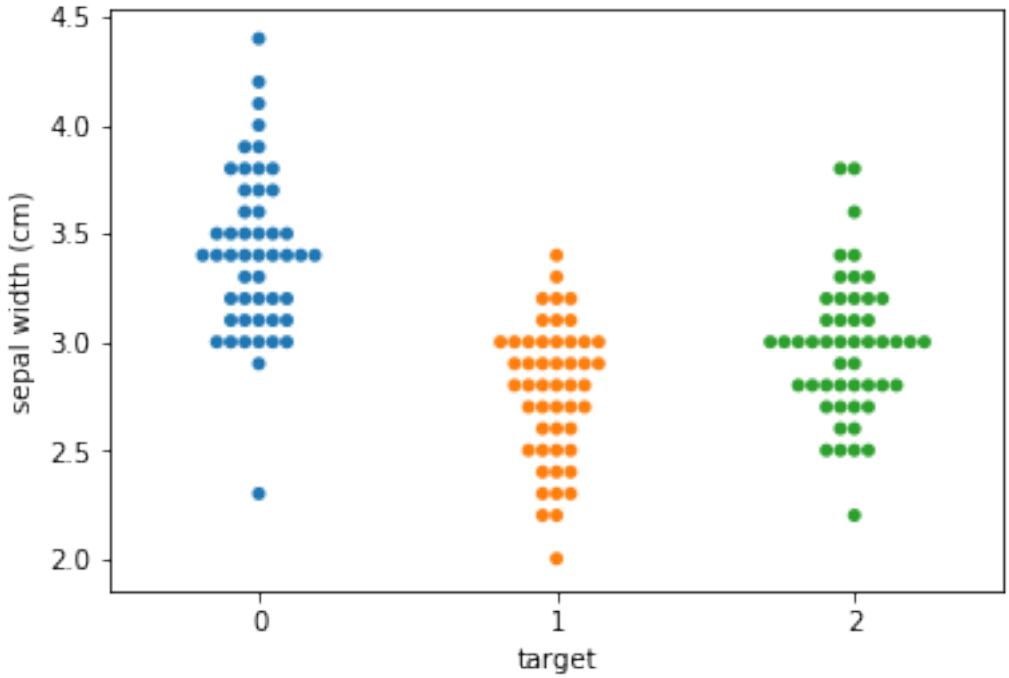

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
petal width (cm)

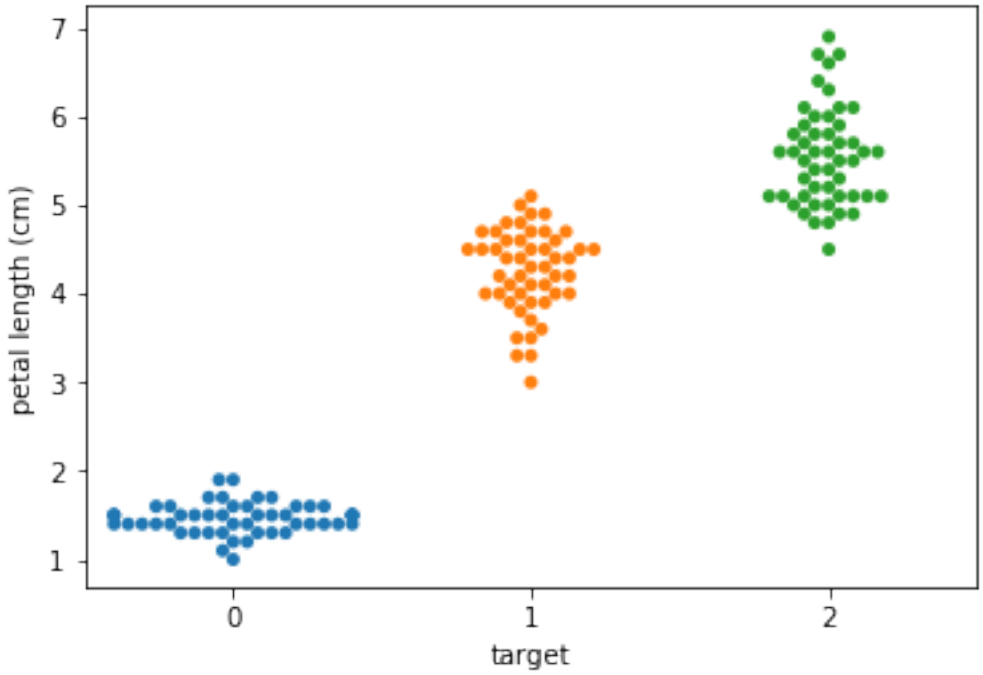

sepal width (cm)
petal width (cm)

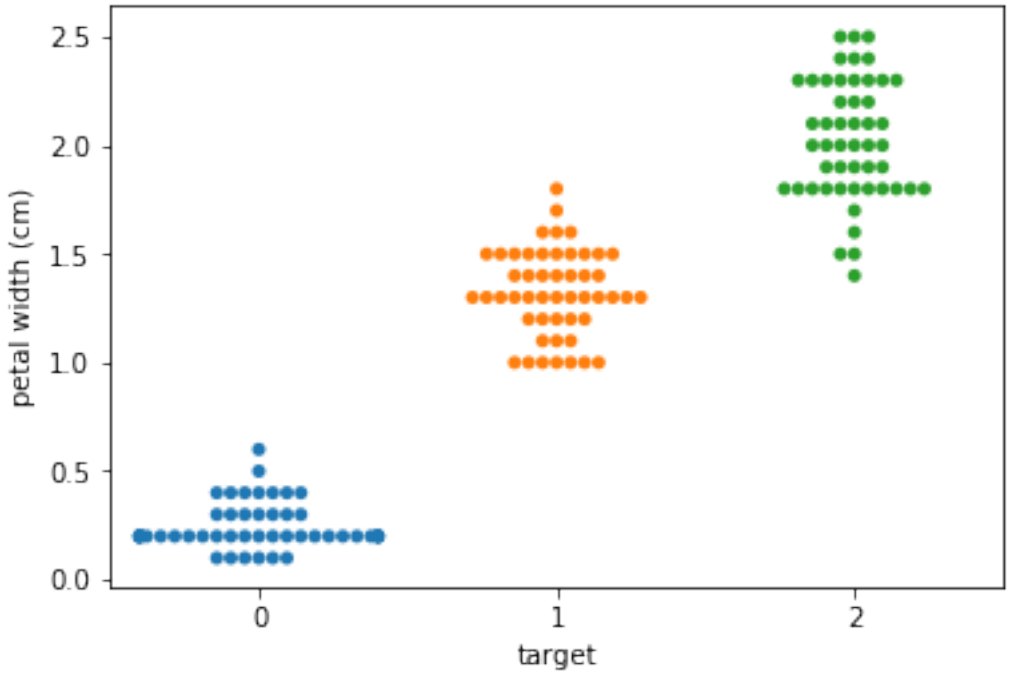

petal length (cm)
petal width (cm)

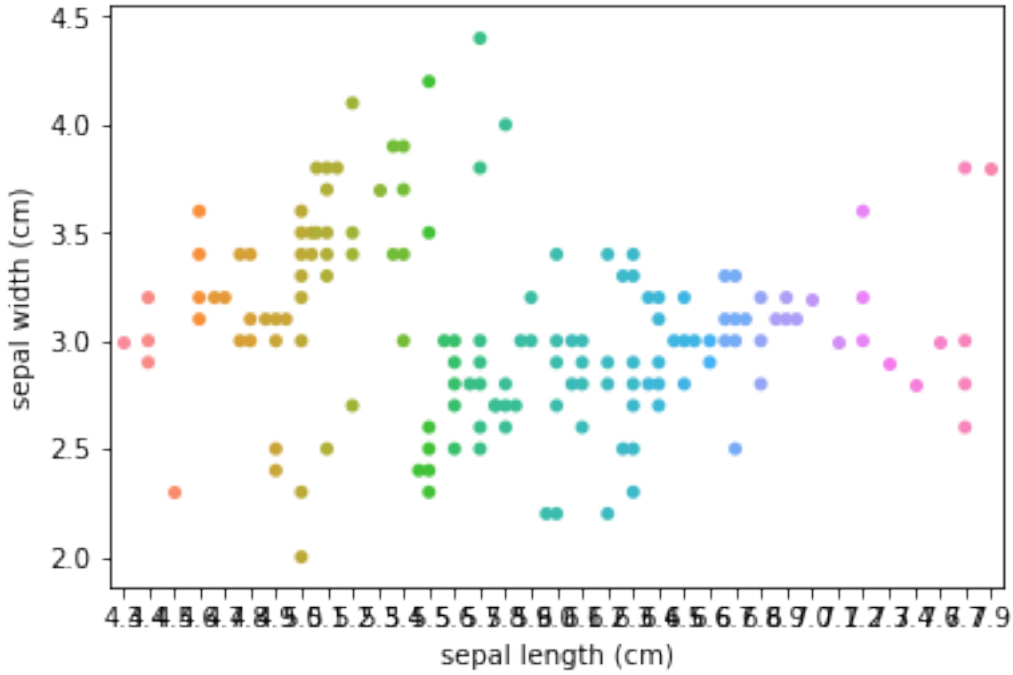

target
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.1 0.2 0.3 0.4 0.5 0.6 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5
petal width (cm)

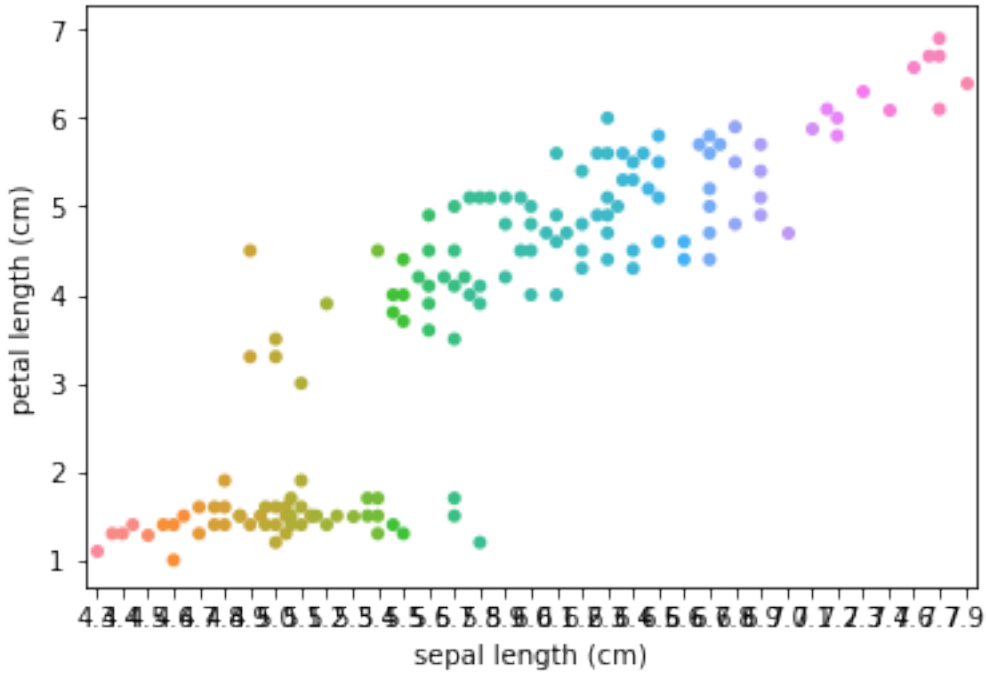

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0
1
2
target

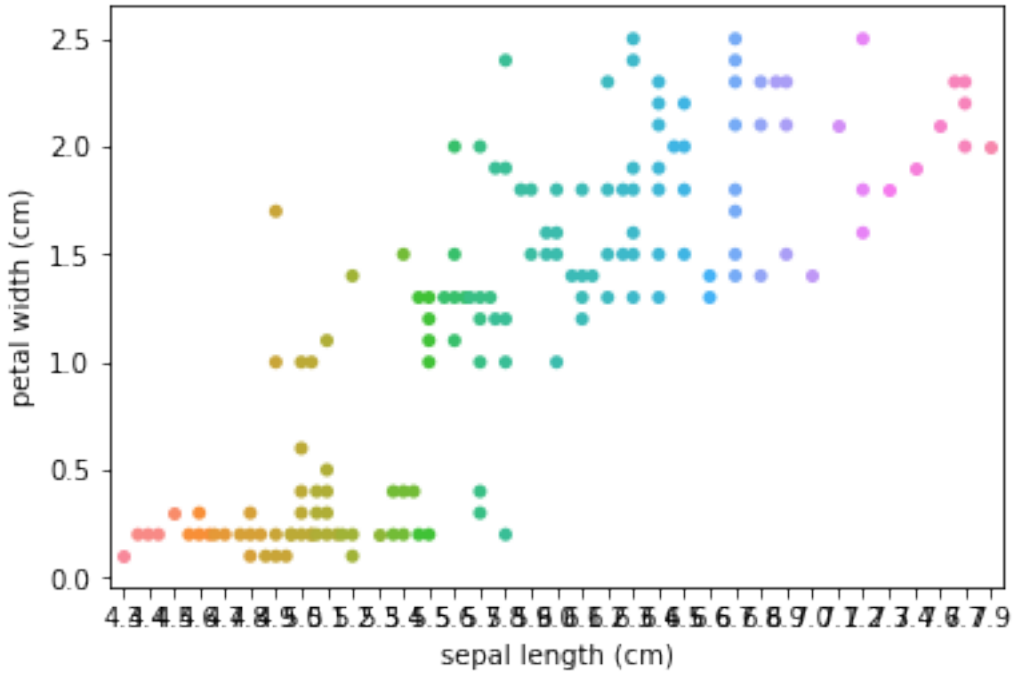

4.5
4.0
3.5
3.0
2.5
2.0
sepal width (cm)
0
1
2
target

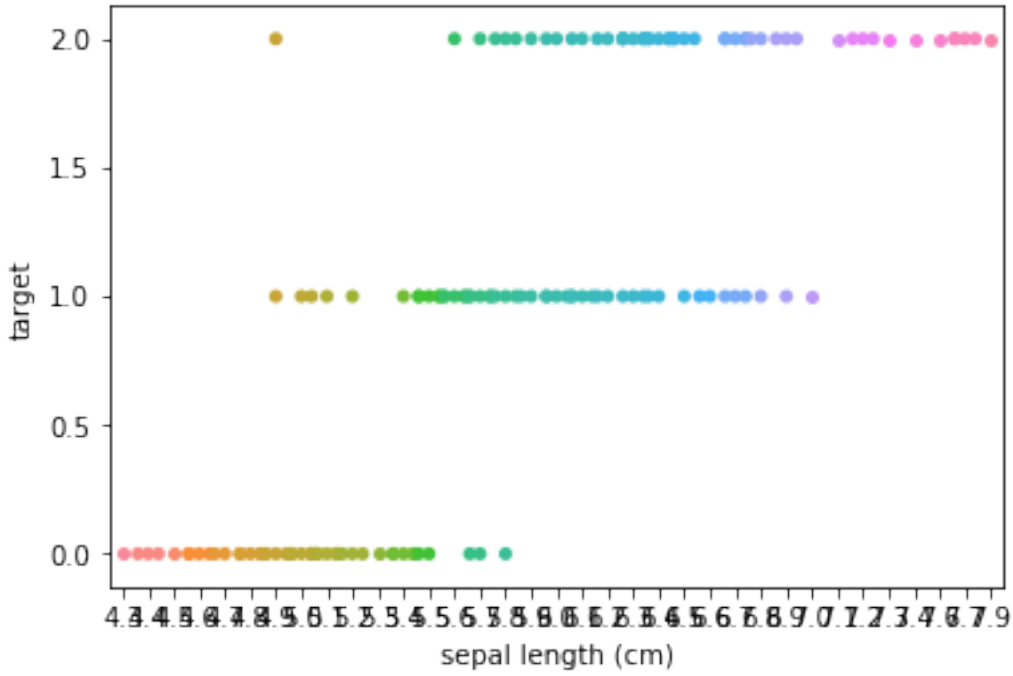

7
6
5
4
3
2
1
petal length (cm)
0
1
2
target

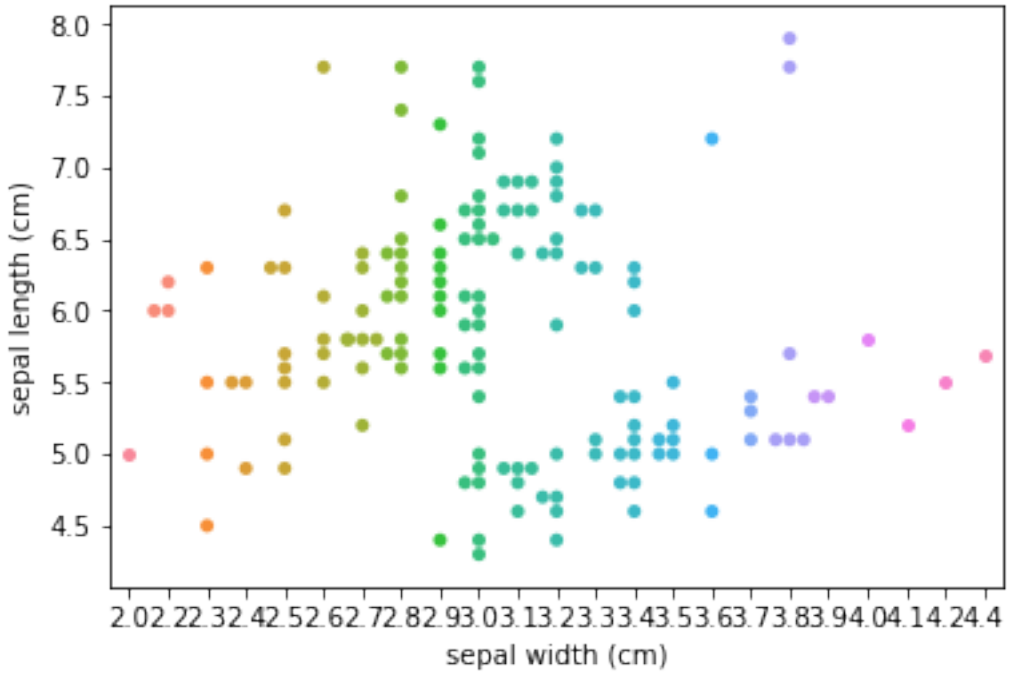

petal width (cm)
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
target

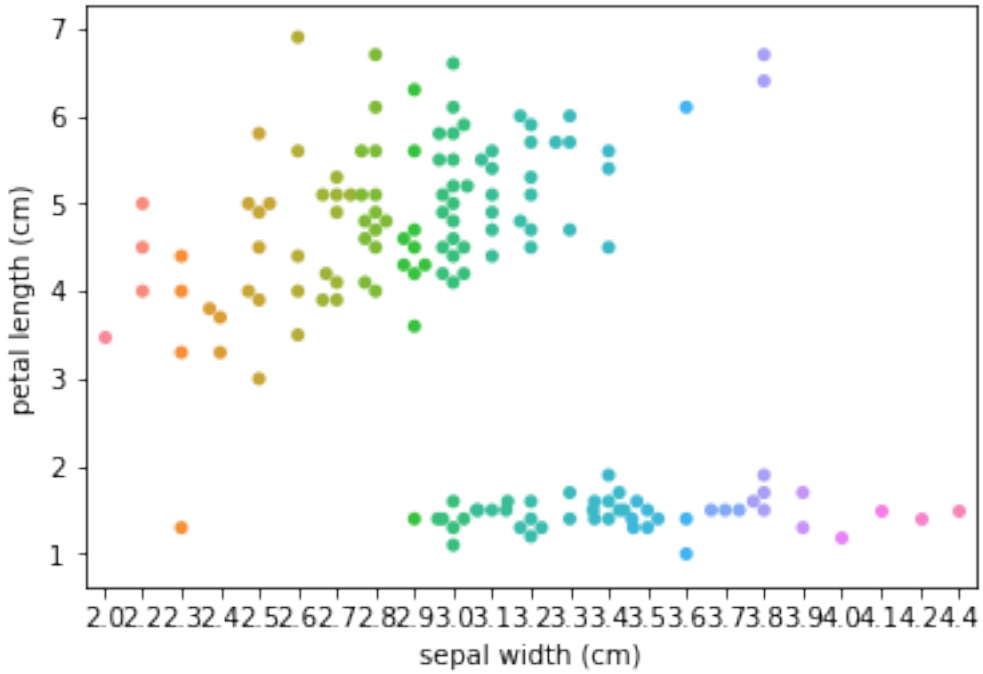

sepal width (cm)
4.5
4.0
3.5
3.0
2.5
2.0
sepal length (cm)

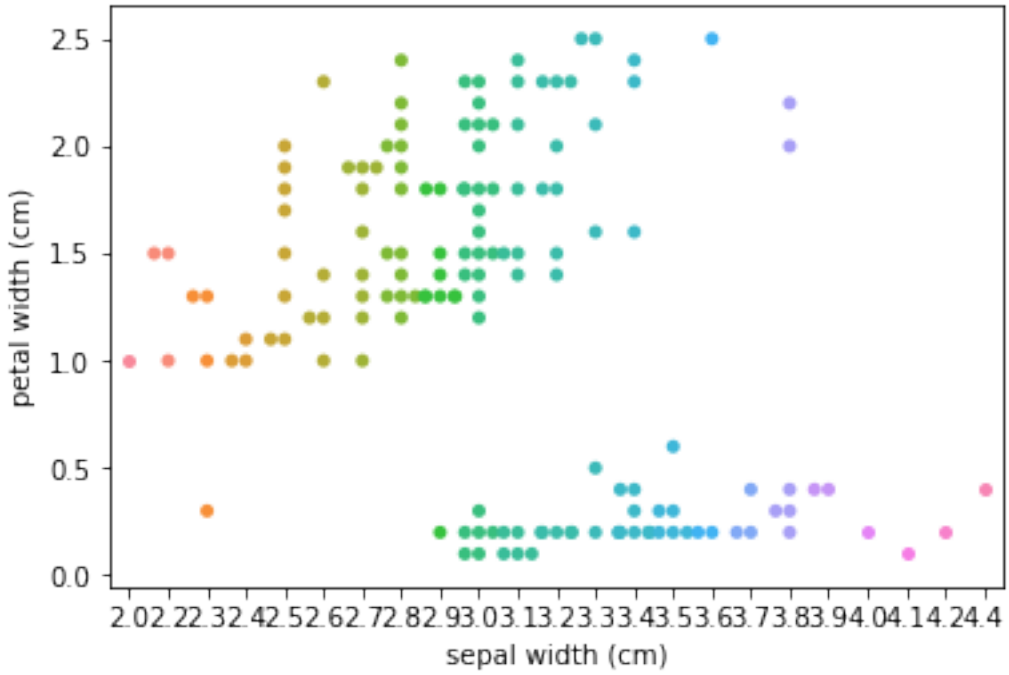

petal length (cm)
sepal length (cm)

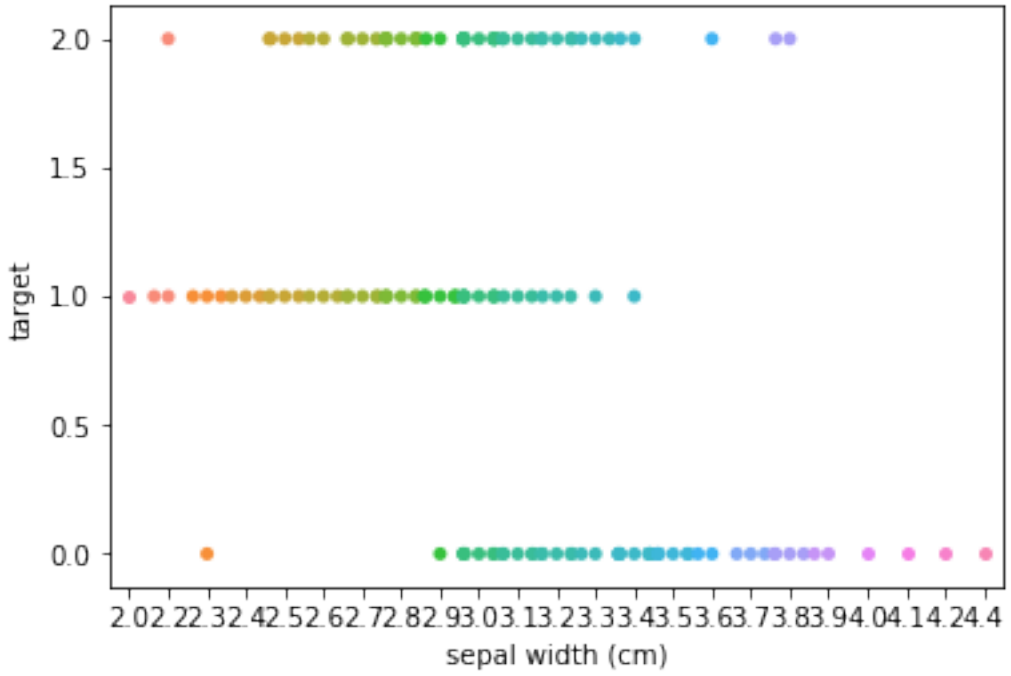

petal width (cm)
sepal length (cm)

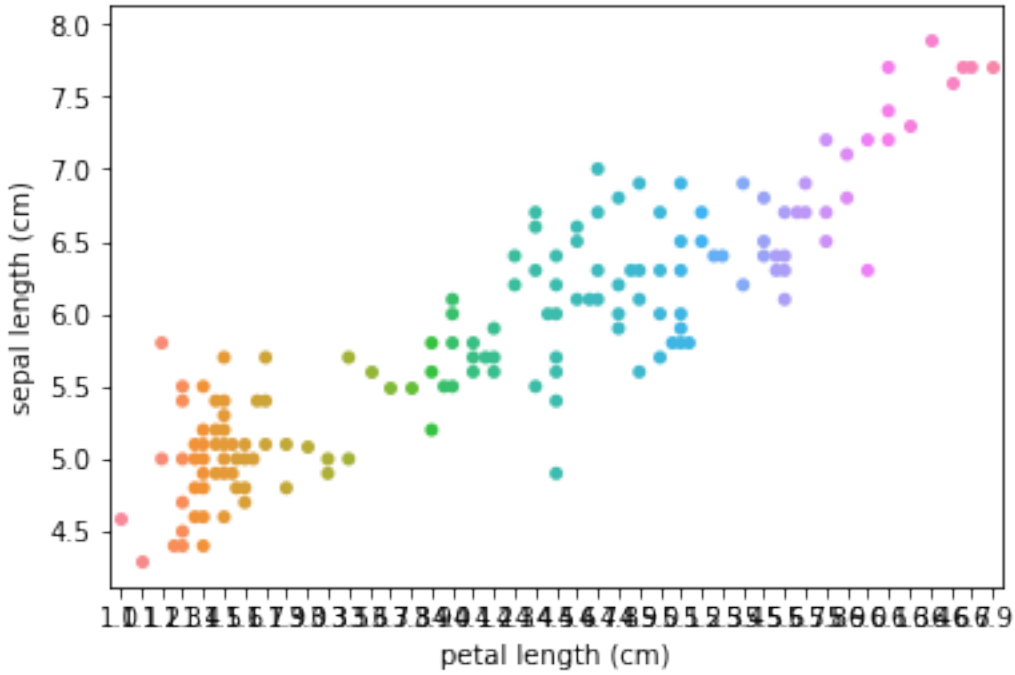

target
2.0
1.5
1.0
0.5
0.0
sepal length (cm)

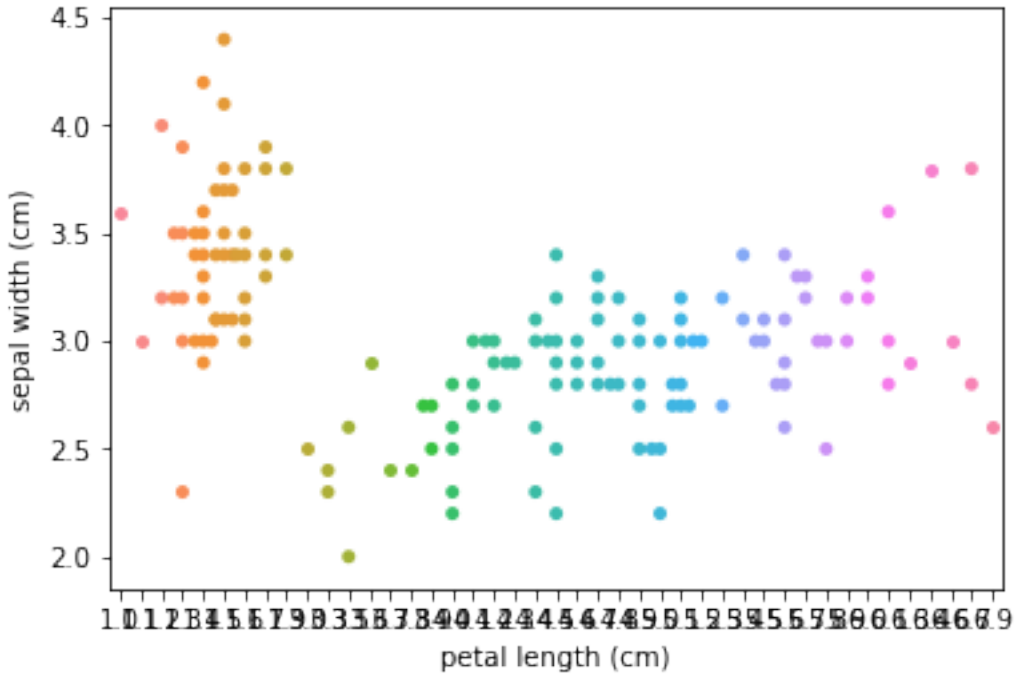

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
sepal width (cm)

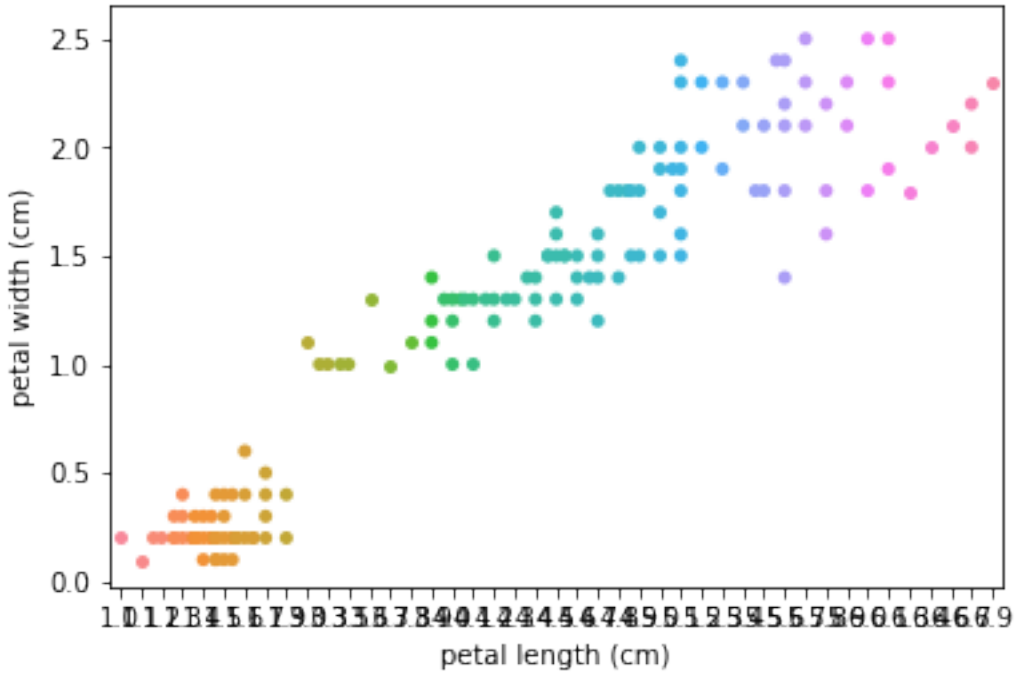

petal length (cm)
sepal width (cm)

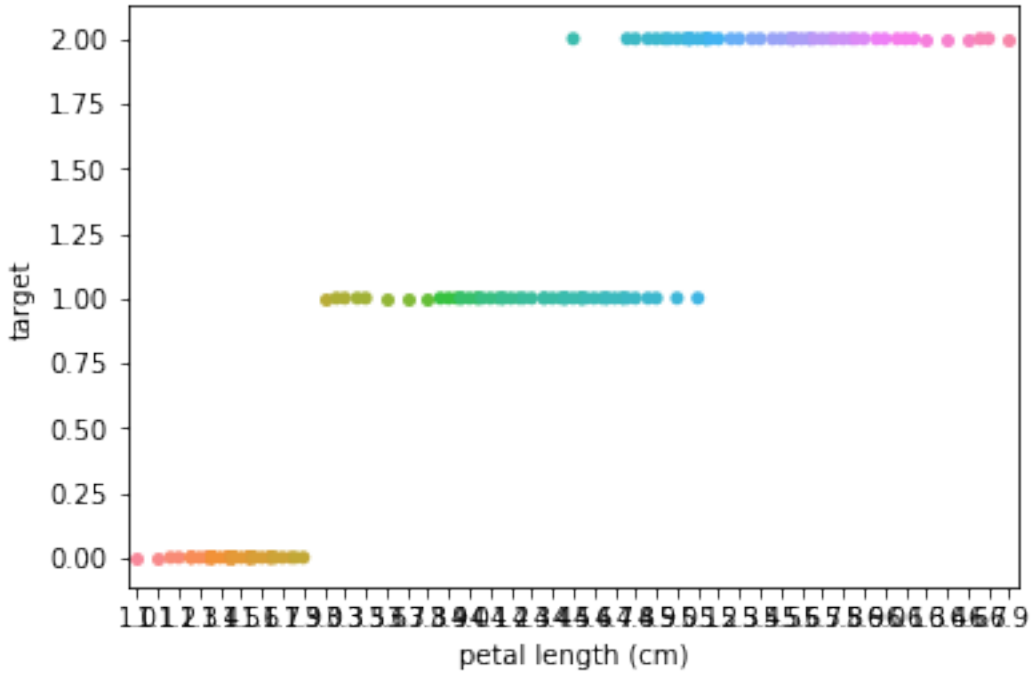

petal width (cm)
sepal width (cm)

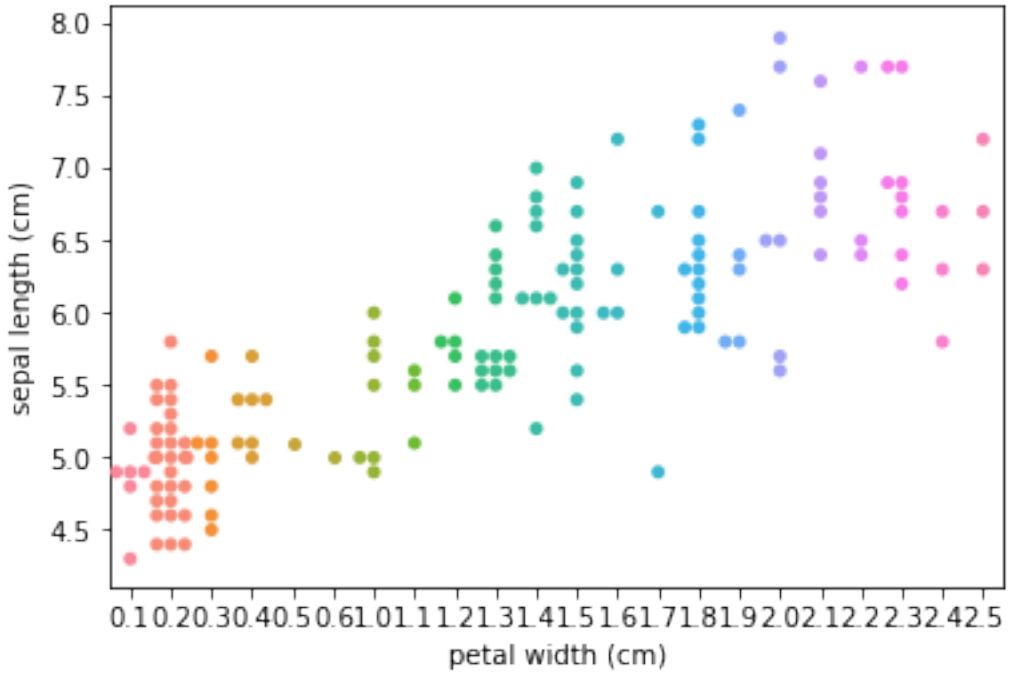

target
2.0
1.5
1.0
0.5
0.0
2.0 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8 3.9 4.0 4.1 4.2 4.4
sepal width (cm)

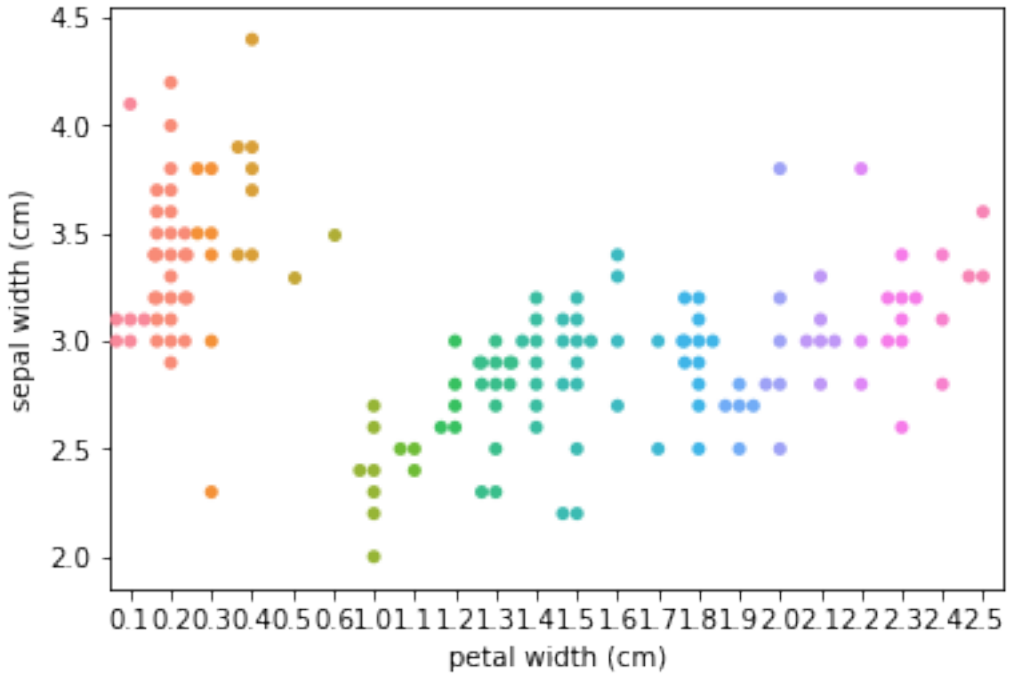

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
petal length (cm)

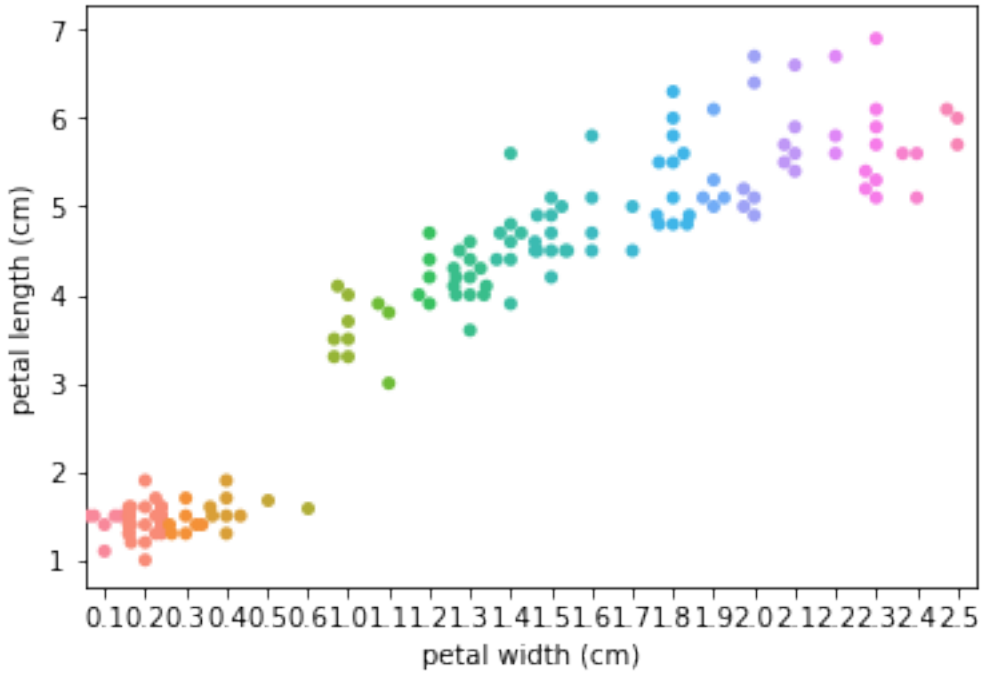

4.5
4.0
3.5
3.0
2.5
2.0
sepal width (cm)
petal length (cm)

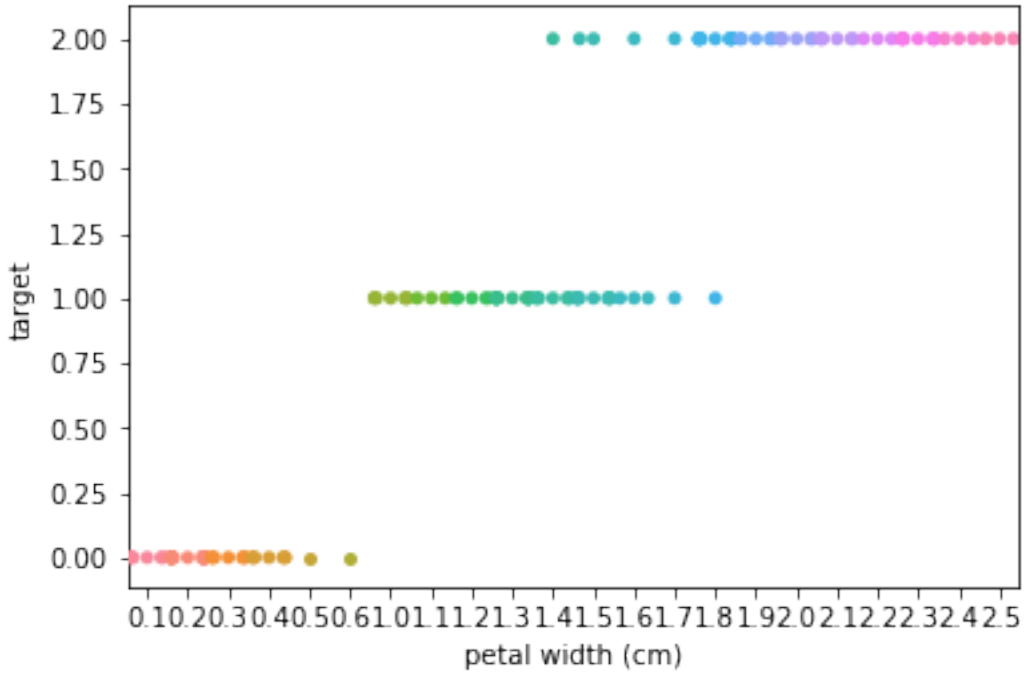

2.5
2.0
1.5
1.0
0.5
0.0
petal width (cm)
petal length (cm)

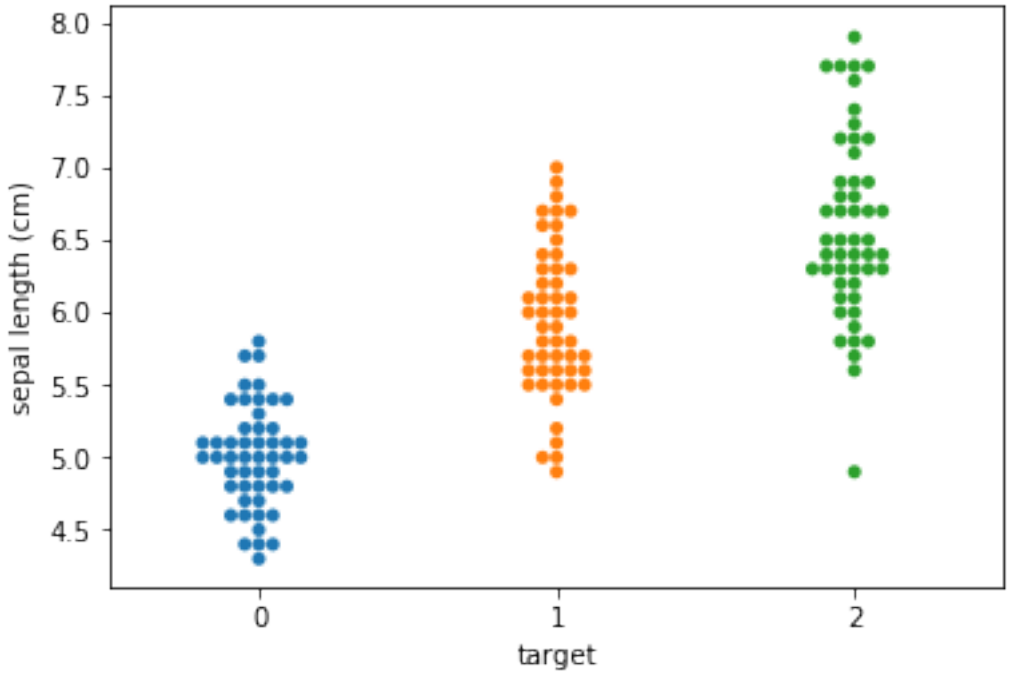

target
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
petal length (cm)

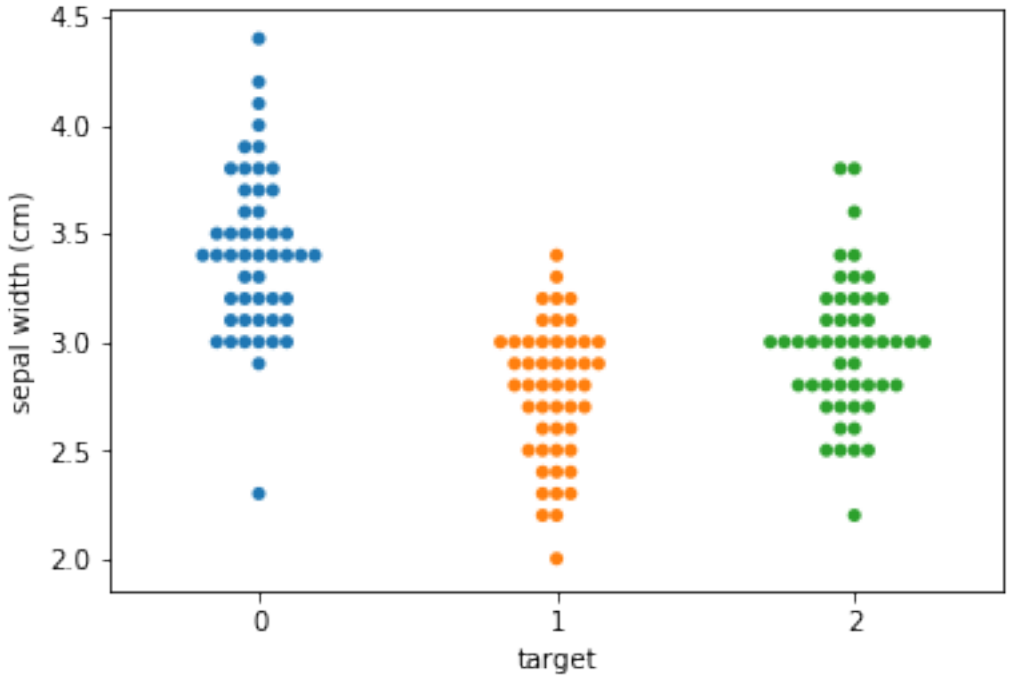

sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
petal width (cm)

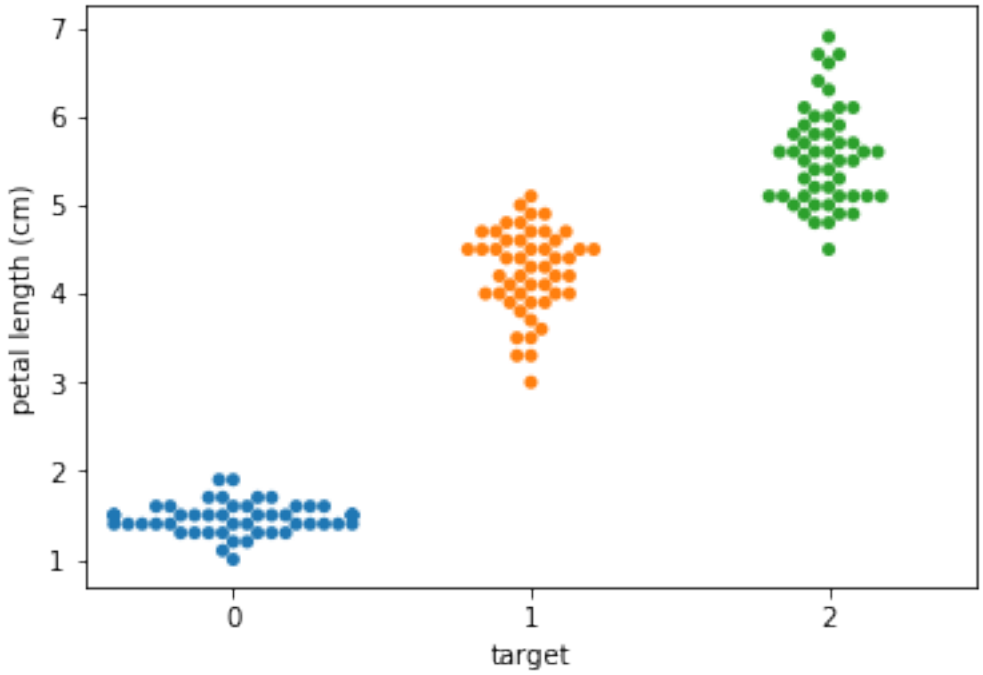

sepal width (cm)
petal width (cm)

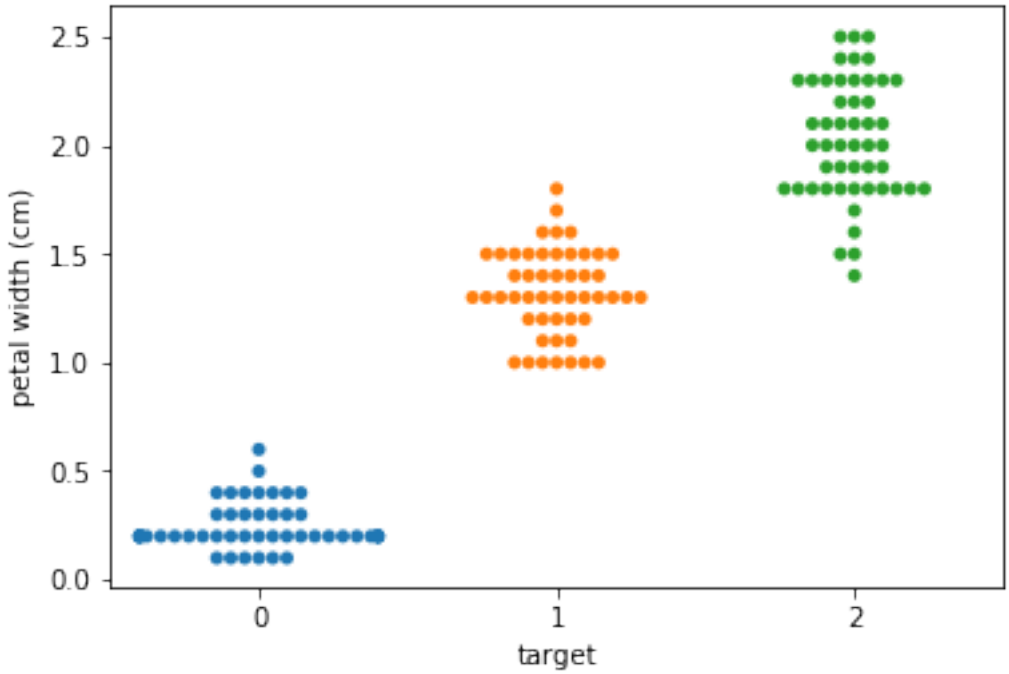

petal length (cm)
petal width (cm)


target
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.1 0.2 0.3 0.4 0.5 0.6 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0 2.1 2.2 2.3 2.4 2.5
petal width (cm)


sepal length (cm)
8.0
7.5
7.0
6.5
6.0
5.5
5.0
4.5
0
1
2
target


4.5
4.0
3.5
3.0
2.5
2.0
sepal width (cm)
0
1
2
target

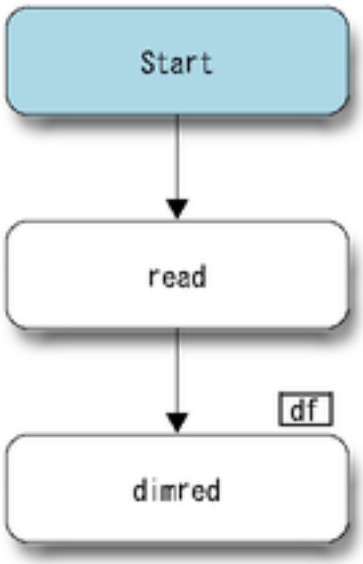

7
6
5
4
3
2
1
petal length (cm)
0
1
2
target



Dimensionality reduction plots

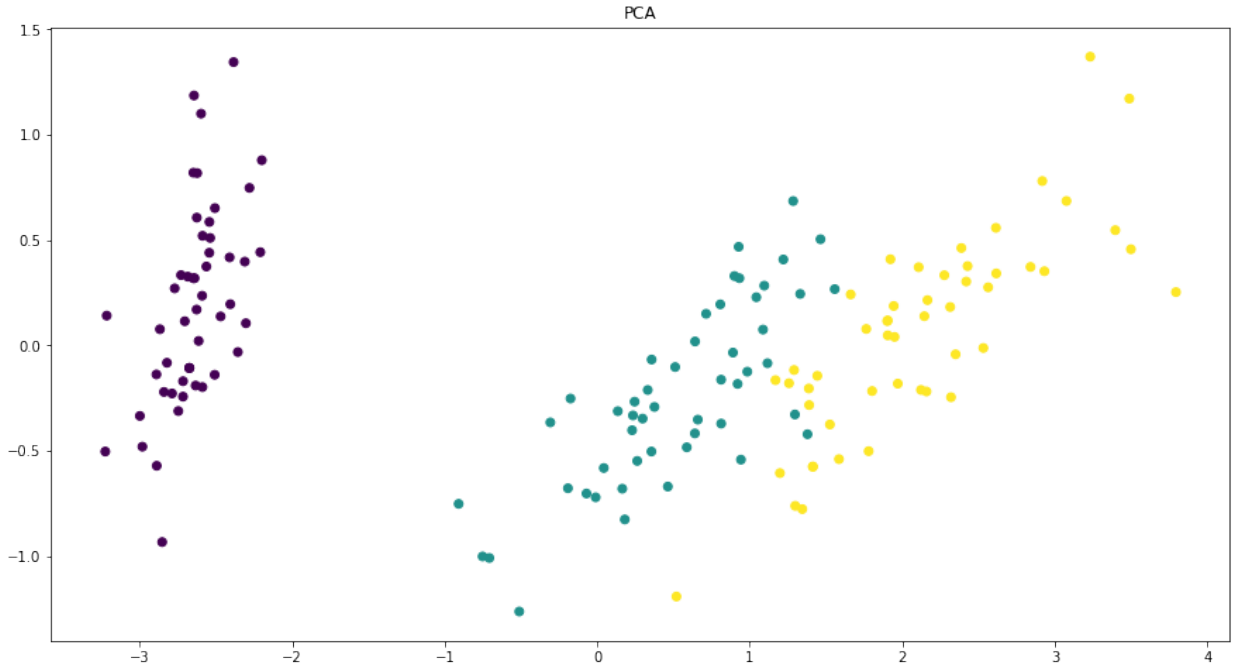
```
In [7]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('dimred', ds.eda.DimensionReductionPlots("target"), [("read", "df", "df")])
        p.fit_transform(name="dimensionality_reduction_example", close_plt=True)
```

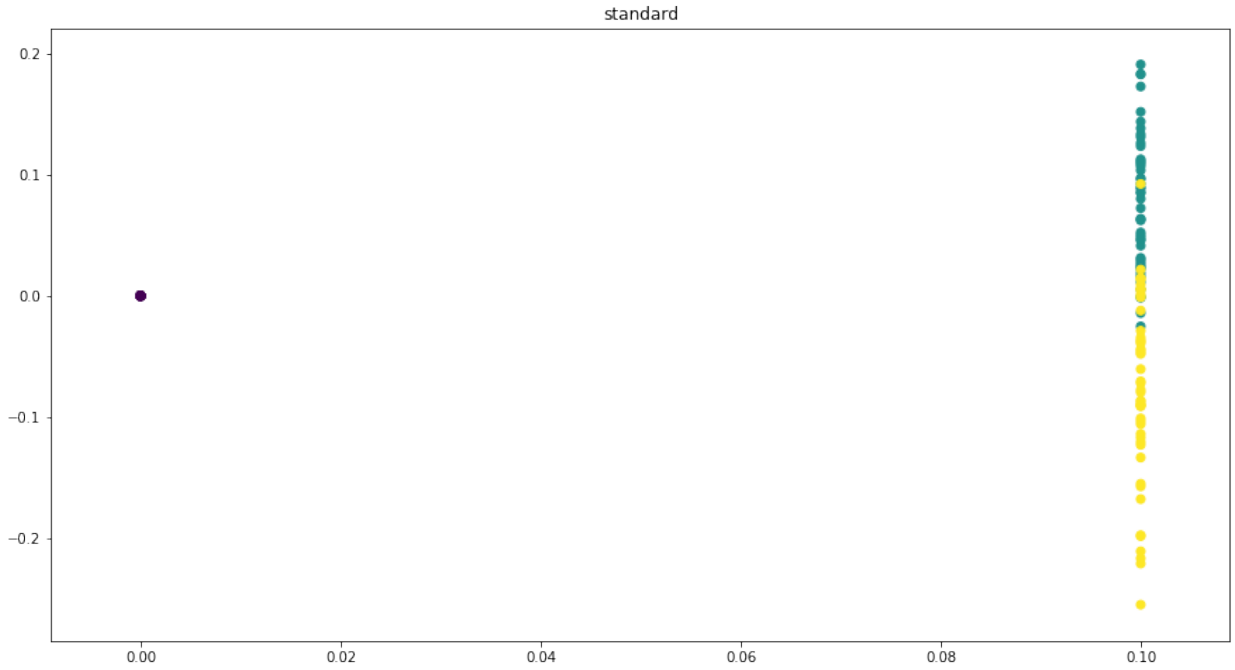
```
'Drawing diagram using blockdiag'
```

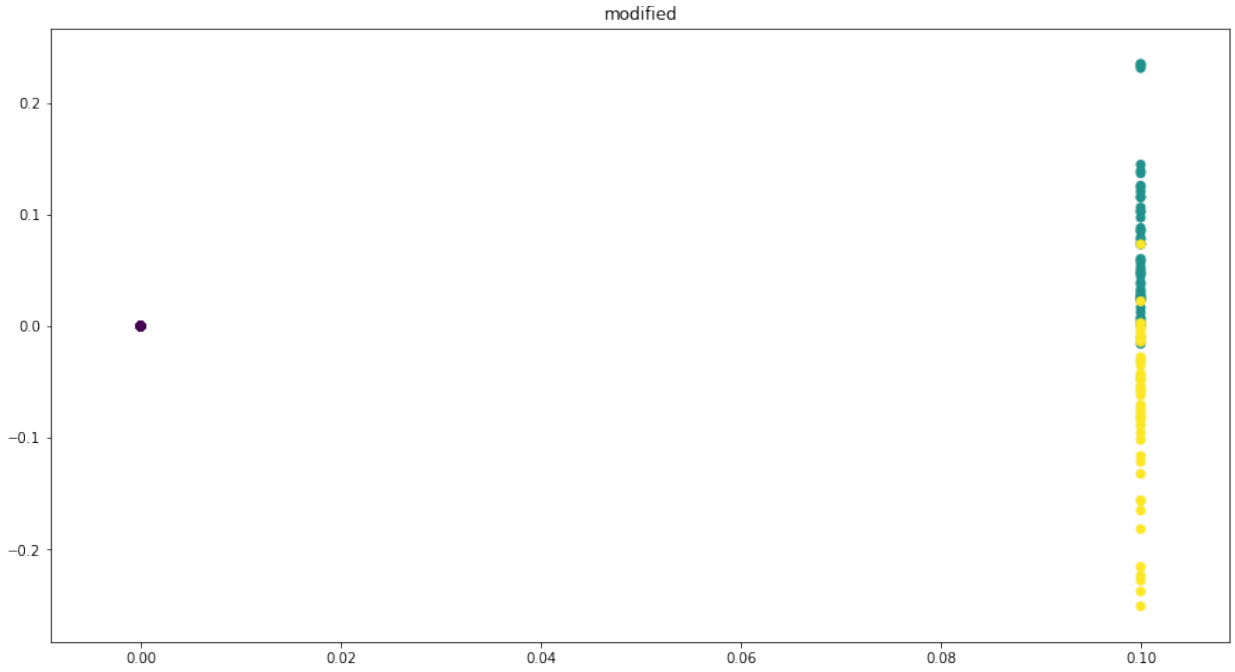


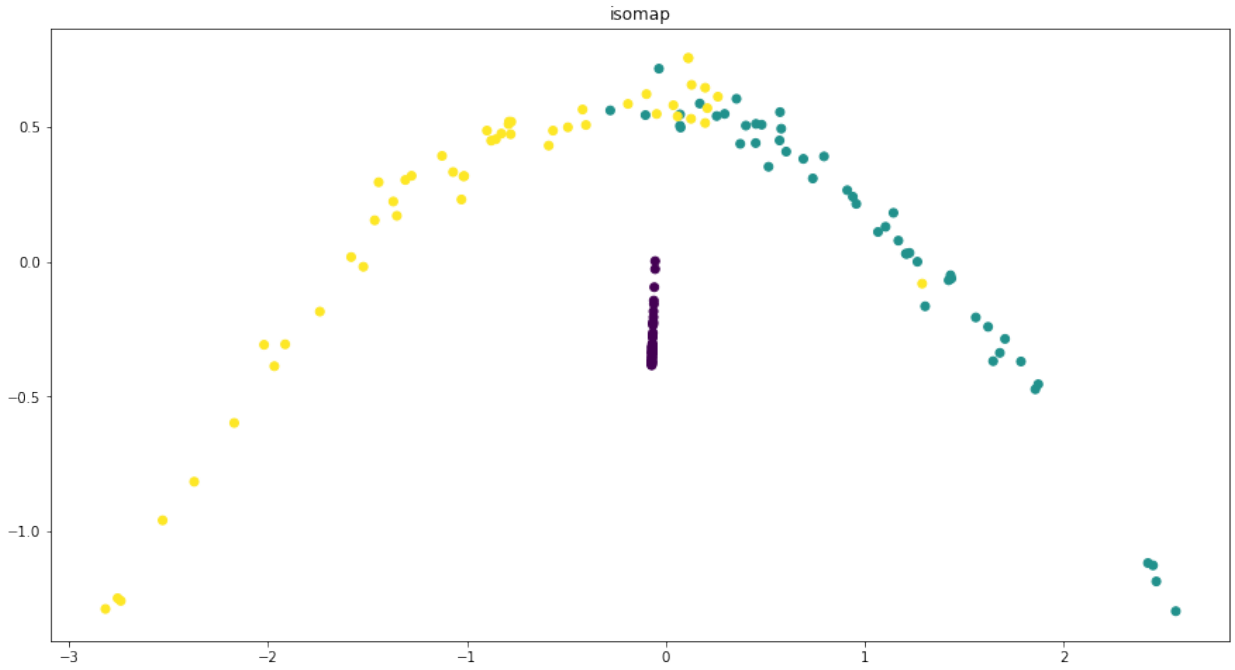

```
<IPython.core.display.HTML object>
```

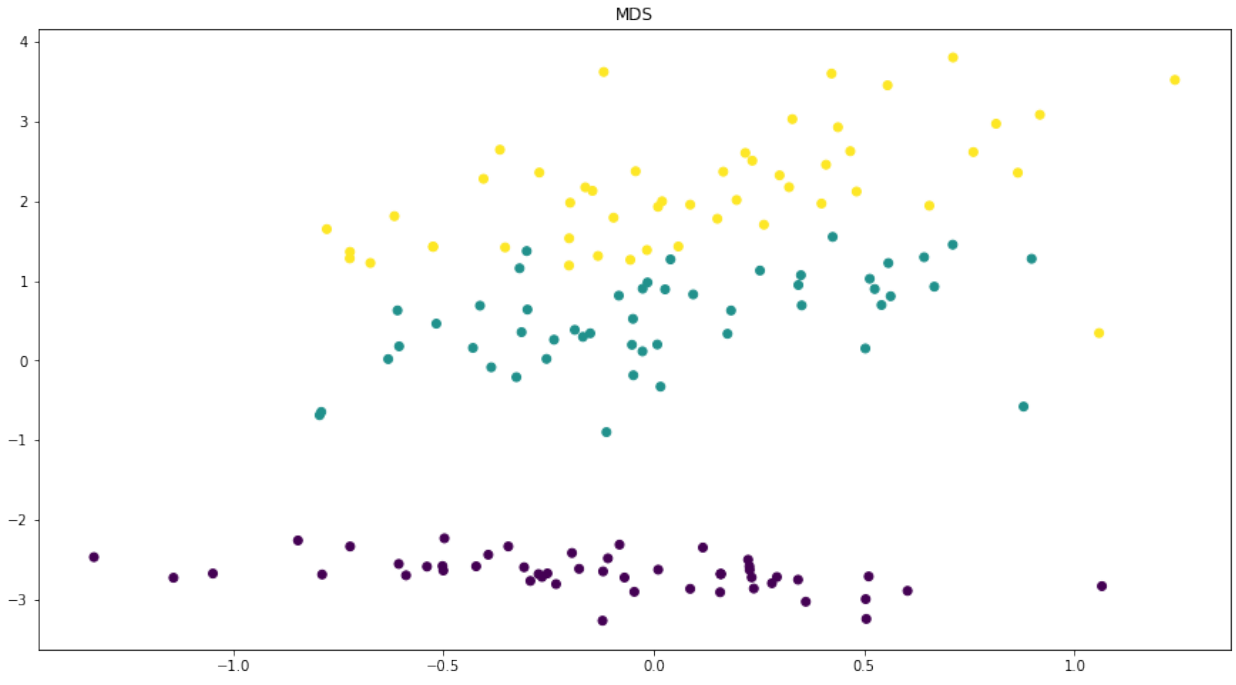
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
```



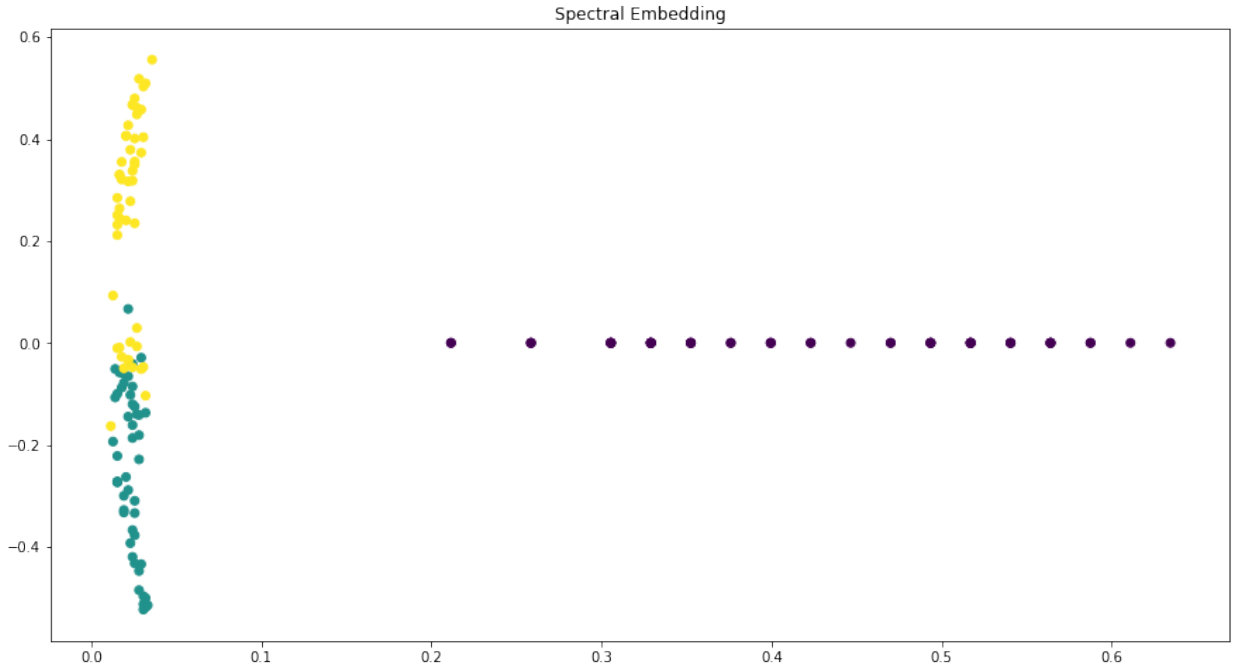

standard
0.2
0.1
0.0
−0.1
−0.2
0.00
0.02
0.04
0.06
0.08
0.10


modified
0.2
0.1
0.0
−0.1
−0.2
0.00
0.02
0.04
0.06
0.08
0.10

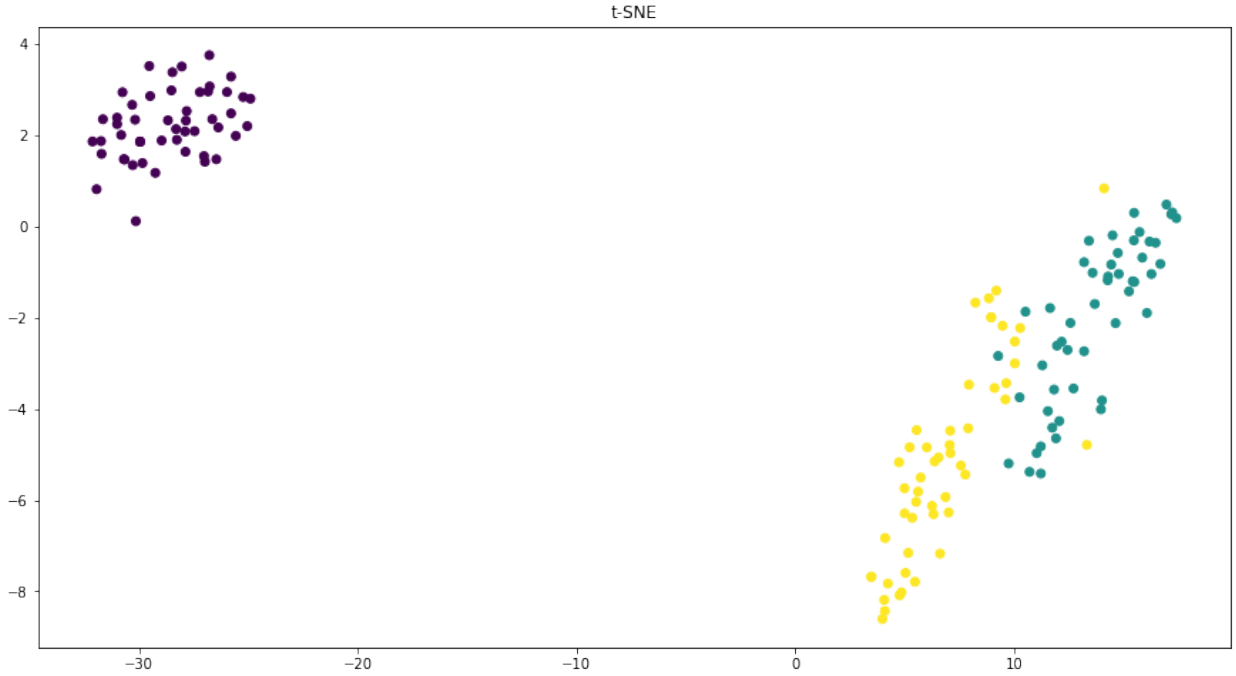





```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\manifold\spectral_embedding_.py:234: UserWarning:
  warnings.warn("Graph is not fully connected, spectral embedding"
```



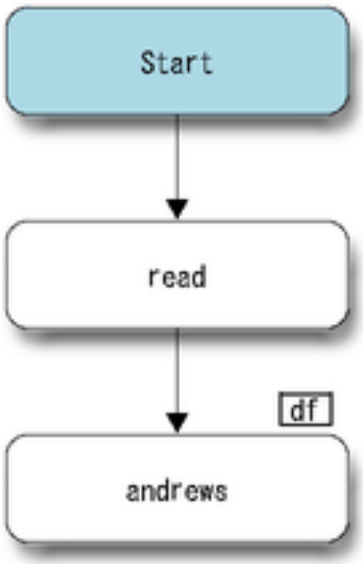

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\base.py:371: RuntimeWarning: invalid val
  result = np.sqrt(dist[sample_range, neigh_ind]), neigh_ind
```



Andrews curves plots

```
In [8]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('andrews', ds.eda.AndrewsPlot(column="target"), [("read", "df", "df")])
        p.transform(name="andrews", close_plt=True)
```

```
'Drawing diagram using blockdiag'
```




```
<IPython.core.display.HTML object>
```

```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
```

```
<IPython.core.display.HTML object>
```

```
In [ ]:
```

1.12.8 Example Data Details

This notebook presents some ways to use the package to give insights of the data

First, import the package

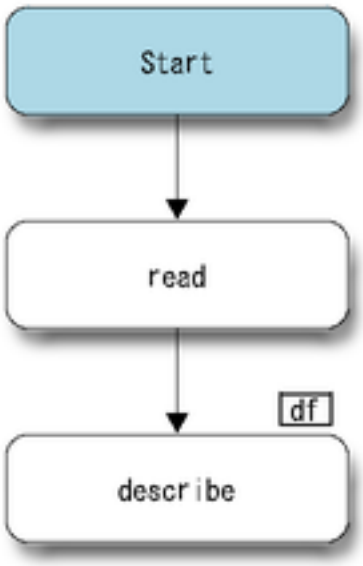
```
In [1]: import dvb.datascience as ds
```

```
C:\ProgramData\Anaconda3\lib\site-packages\deap\tools\_hypervolume\pyhv.py:33: ImportWarning: Falling
  "module. Expect this to be very slow.", ImportWarning)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap_external.py:426: ImportWarning: Not importing direc
  _warnings.warn(msg.format(portions[0]), ImportWarning)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap_external.py:426: ImportWarning: Not importing direc
  _warnings.warn(msg.format(portions[0]), ImportWarning)
```

Describe the data

```
In [2]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('describe', ds.eda.Describe(), [('read', 'df', 'df')])
        p.transform(name="describe", close_plt=True)
```

```
'Drawing diagram using blockdiag'
```




```
<IPython.core.display.HTML object>
```

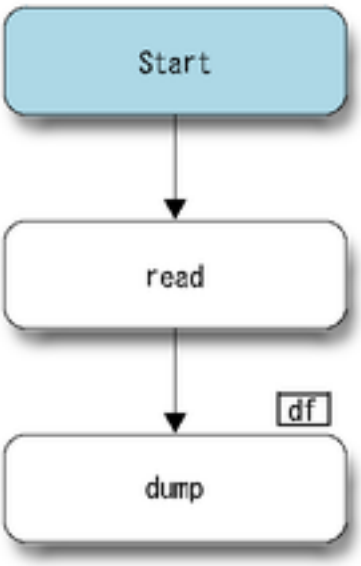
```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.054000           3.758667
std             0.828066          0.433594           1.764420
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.198667    1.000000
std            0.763161    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

Dump the data

```
In [3]: p = ds.Pipeline()
        p.addPipe('read', ds.data.SampleData('iris'))
        p.addPipe('dump', ds.eda.Dump(), [('read', 'df', 'df')])
        p.transform(name="dump", close_plt=True)
```

```
'Drawing diagram using blockdiag'
```




```
<IPython.core.display.HTML object>
```

```
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=2.0), HTML(value='')))
```

	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	
5	5.4	3.9	1.7	0.4	
6	4.6	3.4	1.4	0.3	
7	5.0	3.4	1.5	0.2	
8	4.4	2.9	1.4	0.2	
9	4.9	3.1	1.5	0.1	
10	5.4	3.7	1.5	0.2	
11	4.8	3.4	1.6	0.2	
12	4.8	3.0	1.4	0.1	
13	4.3	3.0	1.1	0.1	
14	5.8	4.0	1.2	0.2	
15	5.7	4.4	1.5	0.4	
16	5.4	3.9	1.3	0.4	
17	5.1	3.5	1.4	0.3	
18	5.7	3.8	1.7	0.3	
19	5.1	3.8	1.5	0.3	
20	5.4	3.4	1.7	0.2	
21	5.1	3.7	1.5	0.4	
22	4.6	3.6	1.0	0.2	
23	5.1	3.3	1.7	0.5	
24	4.8	3.4	1.9	0.2	
25	5.0	3.0	1.6	0.2	
26	5.0	3.4	1.6	0.4	
27	5.2	3.5	1.5	0.2	

28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
30	4.8	3.1	1.6	0.2
31	5.4	3.4	1.5	0.4
32	5.2	4.1	1.5	0.1
33	5.5	4.2	1.4	0.2
34	4.9	3.1	1.5	0.1
35	5.0	3.2	1.2	0.2
36	5.5	3.5	1.3	0.2
37	4.9	3.1	1.5	0.1
38	4.4	3.0	1.3	0.2
39	5.1	3.4	1.5	0.2
40	5.0	3.5	1.3	0.3
41	4.5	2.3	1.3	0.3
42	4.4	3.2	1.3	0.2
43	5.0	3.5	1.6	0.6
44	5.1	3.8	1.9	0.4
45	4.8	3.0	1.4	0.3
46	5.1	3.8	1.6	0.2
47	4.6	3.2	1.4	0.2
48	5.3	3.7	1.5	0.2
49	5.0	3.3	1.4	0.2
50	7.0	3.2	4.7	1.4
51	6.4	3.2	4.5	1.5
52	6.9	3.1	4.9	1.5
53	5.5	2.3	4.0	1.3
54	6.5	2.8	4.6	1.5
55	5.7	2.8	4.5	1.3
56	6.3	3.3	4.7	1.6
57	4.9	2.4	3.3	1.0
58	6.6	2.9	4.6	1.3
59	5.2	2.7	3.9	1.4
60	5.0	2.0	3.5	1.0
61	5.9	3.0	4.2	1.5
62	6.0	2.2	4.0	1.0
63	6.1	2.9	4.7	1.4
64	5.6	2.9	3.6	1.3
65	6.7	3.1	4.4	1.4
66	5.6	3.0	4.5	1.5
67	5.8	2.7	4.1	1.0
68	6.2	2.2	4.5	1.5
69	5.6	2.5	3.9	1.1
70	5.9	3.2	4.8	1.8
71	6.1	2.8	4.0	1.3
72	6.3	2.5	4.9	1.5
73	6.1	2.8	4.7	1.2
74	6.4	2.9	4.3	1.3
75	6.6	3.0	4.4	1.4
76	6.8	2.8	4.8	1.4
77	6.7	3.0	5.0	1.7
78	6.0	2.9	4.5	1.5
79	5.7	2.6	3.5	1.0
80	5.5	2.4	3.8	1.1
81	5.5	2.4	3.7	1.0
82	5.8	2.7	3.9	1.2
83	6.0	2.7	5.1	1.6
84	5.4	3.0	4.5	1.5
85	6.0	3.4	4.5	1.6
86	6.7	3.1	4.7	1.5

87	6.3	2.3	4.4	1.3
88	5.6	3.0	4.1	1.3
89	5.5	2.5	4.0	1.3
90	5.5	2.6	4.4	1.2
91	6.1	3.0	4.6	1.4
92	5.8	2.6	4.0	1.2
93	5.0	2.3	3.3	1.0
94	5.6	2.7	4.2	1.3
95	5.7	3.0	4.2	1.2
96	5.7	2.9	4.2	1.3
97	6.2	2.9	4.3	1.3
98	5.1	2.5	3.0	1.1
99	5.7	2.8	4.1	1.3
100	6.3	3.3	6.0	2.5
101	5.8	2.7	5.1	1.9
102	7.1	3.0	5.9	2.1
103	6.3	2.9	5.6	1.8
104	6.5	3.0	5.8	2.2
105	7.6	3.0	6.6	2.1
106	4.9	2.5	4.5	1.7
107	7.3	2.9	6.3	1.8
108	6.7	2.5	5.8	1.8
109	7.2	3.6	6.1	2.5
110	6.5	3.2	5.1	2.0
111	6.4	2.7	5.3	1.9
112	6.8	3.0	5.5	2.1
113	5.7	2.5	5.0	2.0
114	5.8	2.8	5.1	2.4
115	6.4	3.2	5.3	2.3
116	6.5	3.0	5.5	1.8
117	7.7	3.8	6.7	2.2
118	7.7	2.6	6.9	2.3
119	6.0	2.2	5.0	1.5
120	6.9	3.2	5.7	2.3
121	5.6	2.8	4.9	2.0
122	7.7	2.8	6.7	2.0
123	6.3	2.7	4.9	1.8
124	6.7	3.3	5.7	2.1
125	7.2	3.2	6.0	1.8
126	6.2	2.8	4.8	1.8
127	6.1	3.0	4.9	1.8
128	6.4	2.8	5.6	2.1
129	7.2	3.0	5.8	1.6
130	7.4	2.8	6.1	1.9
131	7.9	3.8	6.4	2.0
132	6.4	2.8	5.6	2.2
133	6.3	2.8	5.1	1.5
134	6.1	2.6	5.6	1.4
135	7.7	3.0	6.1	2.3
136	6.3	3.4	5.6	2.4
137	6.4	3.1	5.5	1.8
138	6.0	3.0	4.8	1.8
139	6.9	3.1	5.4	2.1
140	6.7	3.1	5.6	2.4
141	6.9	3.1	5.1	2.3
142	5.8	2.7	5.1	1.9
143	6.8	3.2	5.9	2.3
144	6.7	3.3	5.7	2.5
145	6.7	3.0	5.2	2.3

```
146                6.3               2.5               5.0               1.9
147                6.5               3.0               5.2               2.0
148                6.2               3.4               5.4               2.3
149                5.9               3.0               5.1               1.8

     target
0         0
1         0
2         0
3         0
4         0
5         0
6         0
7         0
8         0
9         0
10        0
11        0
12        0
13        0
14        0
15        0
16        0
17        0
18        0
19        0
20        0
21        0
22        0
23        0
24        0
25        0
26        0
27        0
28        0
29        0
30        0
31        0
32        0
33        0
34        0
35        0
36        0
37        0
38        0
39        0
40        0
41        0
42        0
43        0
44        0
45        0
46        0
47        0
48        0
49        0
50        1
51        1
52        1
```

```
53        1
54        1
55        1
56        1
57        1
58        1
59        1
60        1
61        1
62        1
63        1
64        1
65        1
66        1
67        1
68        1
69        1
70        1
71        1
72        1
73        1
74        1
75        1
76        1
77        1
78        1
79        1
80        1
81        1
82        1
83        1
84        1
85        1
86        1
87        1
88        1
89        1
90        1
91        1
92        1
93        1
94        1
95        1
96        1
97        1
98        1
99        1
100       2
101       2
102       2
103       2
104       2
105       2
106       2
107       2
108       2
109       2
110       2
111       2
```

```
112       2
113       2
114       2
115       2
116       2
117       2
118       2
119       2
120       2
121       2
122       2
123       2
124       2
125       2
126       2
127       2
128       2
129       2
130       2
131       2
132       2
133       2
134       2
135       2
136       2
137       2
138       2
139       2
140       2
141       2
142       2
143       2
144       2
145       2
146       2
147       2
148       2
149       2
```

1.12.9 Example running pipeline from script

This notebook gives an example of how the package can be used for running it interactively using a notebook.

First, import the package

```
In [1]: import dvb.datascience as ds
```

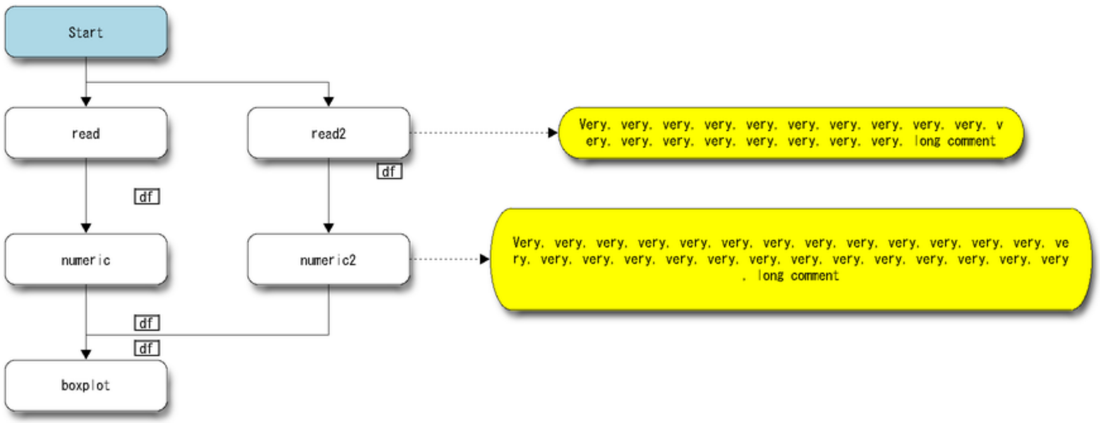
```
C:\ProgramData\Anaconda3\lib\site-packages\deap\tools\_hypervolume\pyhv.py:33: ImportWarning: Falling
  "module. Expect this to be very slow.", ImportWarning)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap_external.py:426: ImportWarning: Not importing direc
  _warnings.warn(msg.format(portions[0]), ImportWarning)
C:\ProgramData\Anaconda3\lib\importlib\_bootstrap_external.py:426: ImportWarning: Not importing direc
  _warnings.warn(msg.format(portions[0]), ImportWarning)
```

Next, use the package to run the file 'example.py' (from the same directory as this notebook) using the `run()` method of the example.py file

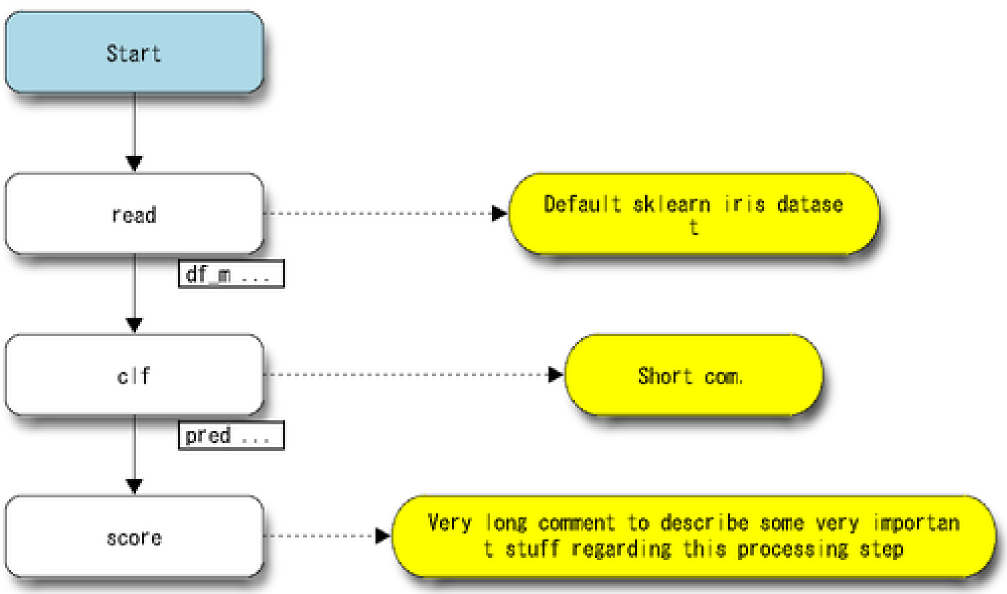
```
In [2]: p = ds.run_module('example').run()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
'Drawing diagram using blockdiag'
```

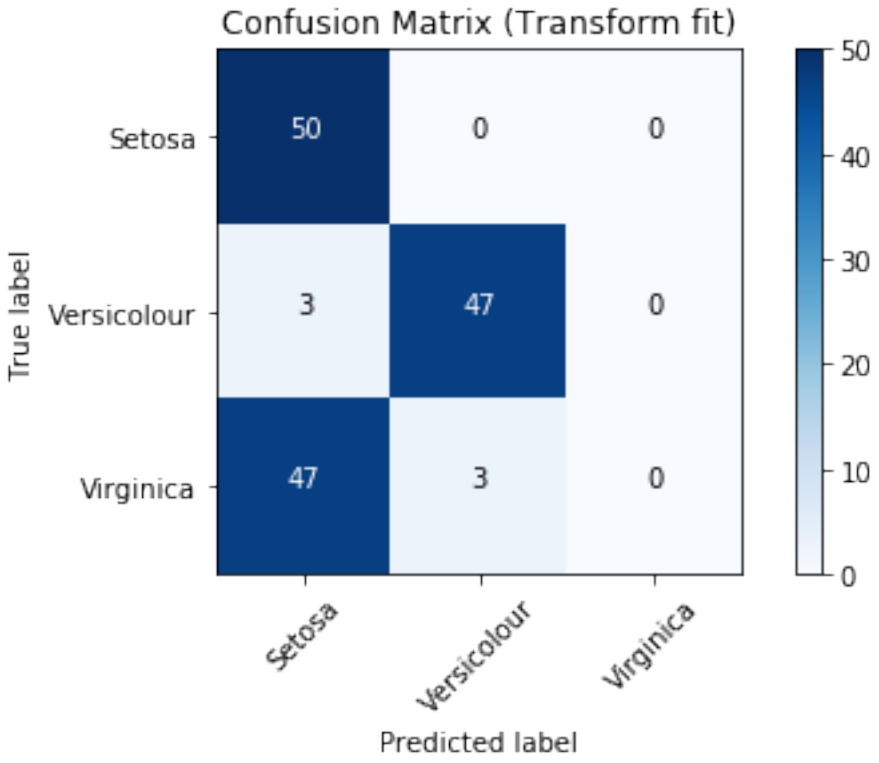



```
<IPython.core.display.HTML object>
'Drawing diagram using blockdiag'
```

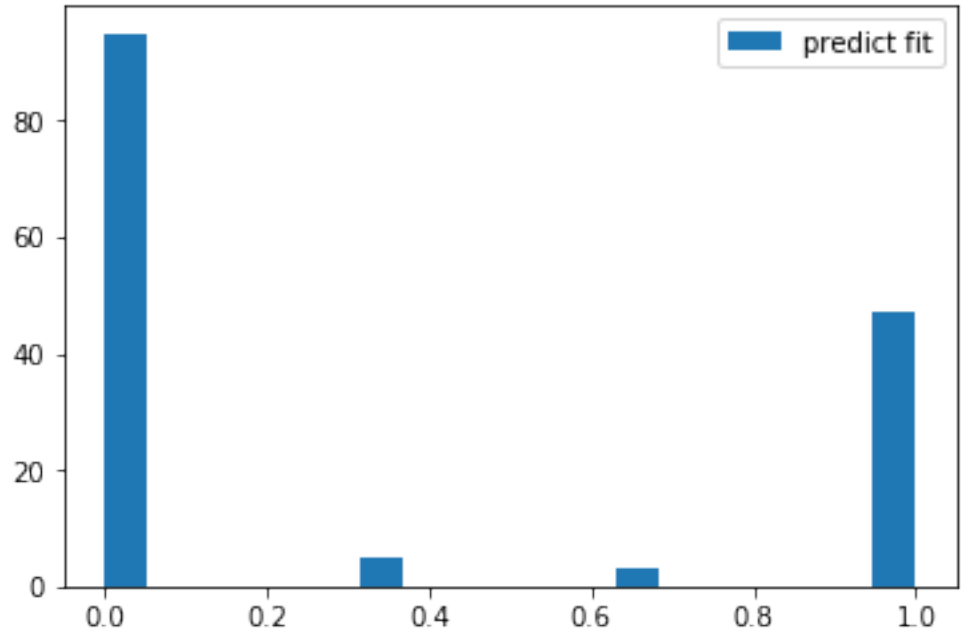



```
<IPython.core.display.HTML object>
HBox(children=(FloatProgress(value=0.0, description='Progress:', max=3.0), HTML(value='')))
'auc() not yet implemented for multiclass classifiers'
'plot_auc() not yet implemented for multiclass classifiers'
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

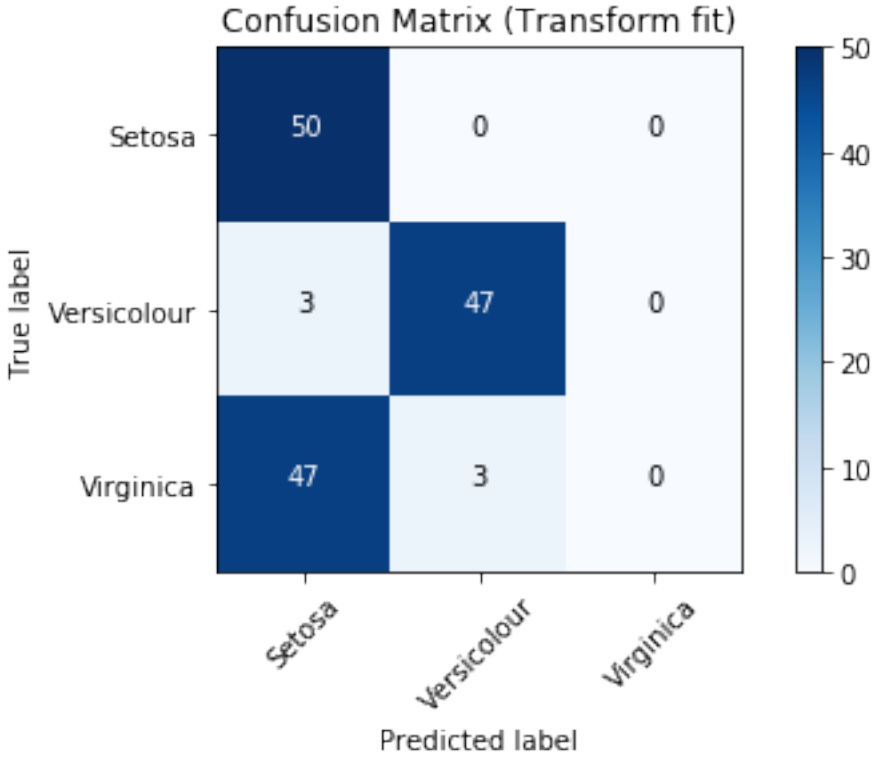
```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

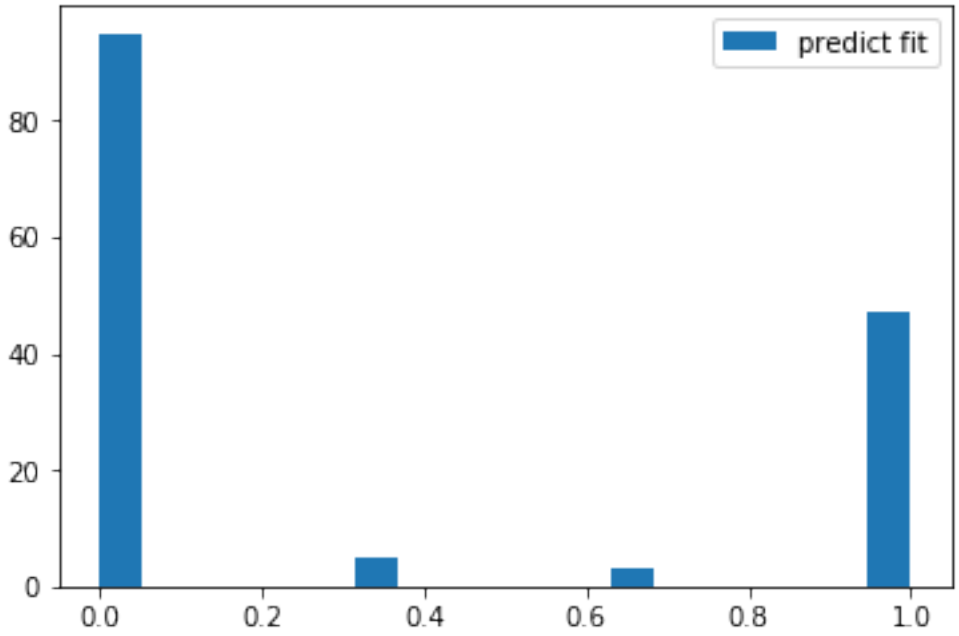



```
'Precision-recall-curve not yet implemented for multiclass classifiers'
'log_loss() not yet implemented for multiclass classifiers'
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```




```
<IPython.core.display.HTML object>
```




predict fit
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W

X

Y