IT8302 APPLIED MACHINE LEARNING

Practical 3
Python Machine Learning

What you will learn / do in this lab

1. *Explore using Python for Machine Learning*
2. *Explore Scikit-learn*
3. *Conduct an experiment with the iris dataset*

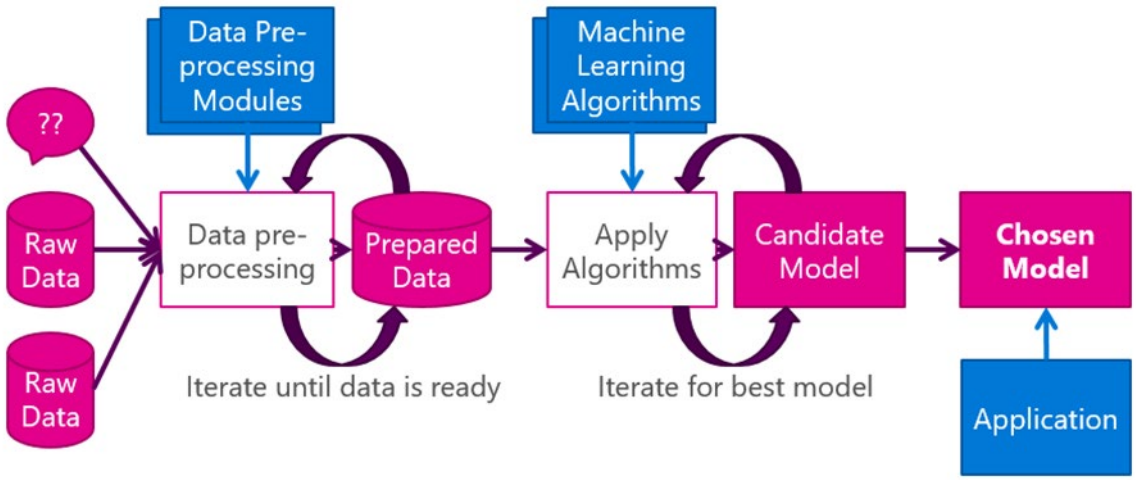
TABLE OF CONTENTS

1.
OVERVIEW

In this practical we will learn how to conduct the machine learning workflow using the python programming language and the scikit-learn machine learning library.

SCIKIT-LEARN

Scikit-learn is a python library that contains classes and functions that help us with most the most common tasks in a machine learning workflow. The machine learning workflow which we are referring to should be familiar to you by now; it is the same as the one we used with Azure ML Studio. We start with formulating the questions first, then data preparation, etc.



Relationship between Numpy, Scipy, Pandas and Scikit-learn

Numpy and Scipy add arrays/vectors and matrix computations to python. Pandas brings DataFrames to python. DataFrames is a core data structure in data analysis. Building on top of these data structures,

Scikit-learn brings implementations of best-of-breed machine-learning algorithms, all under a standardized library.

☐ *PREPROCESSING WITH PANDAS*
- Reading data
- Selecting columns and rows
- Filtering
- Vectorized string operations
- Missing values
- Handling time
- Time series

☐ *NUMPY, SCIPY*
- Arrays
- Indexing, Slicing, and Iterating
- Reshaping
- Shallow vs deep copy
- Broadcasting
- Indexing (advanced)
- Matrices
- Matrix decompositions

☐ *SCIKIT-LEARN*
- Feature extraction
- Classification
- Regression
- Clustering
- Dimension reduction
- Model selection

INSTALLATION

If you are using Anaconda, scikit-learn is already included in the distribution; you don't need to install scikit-learn separately.

To install scikit-learn with other python distributions, refer to the instructions on the scikit-learn site:
http://scikit-learn.org/stable/install.html#

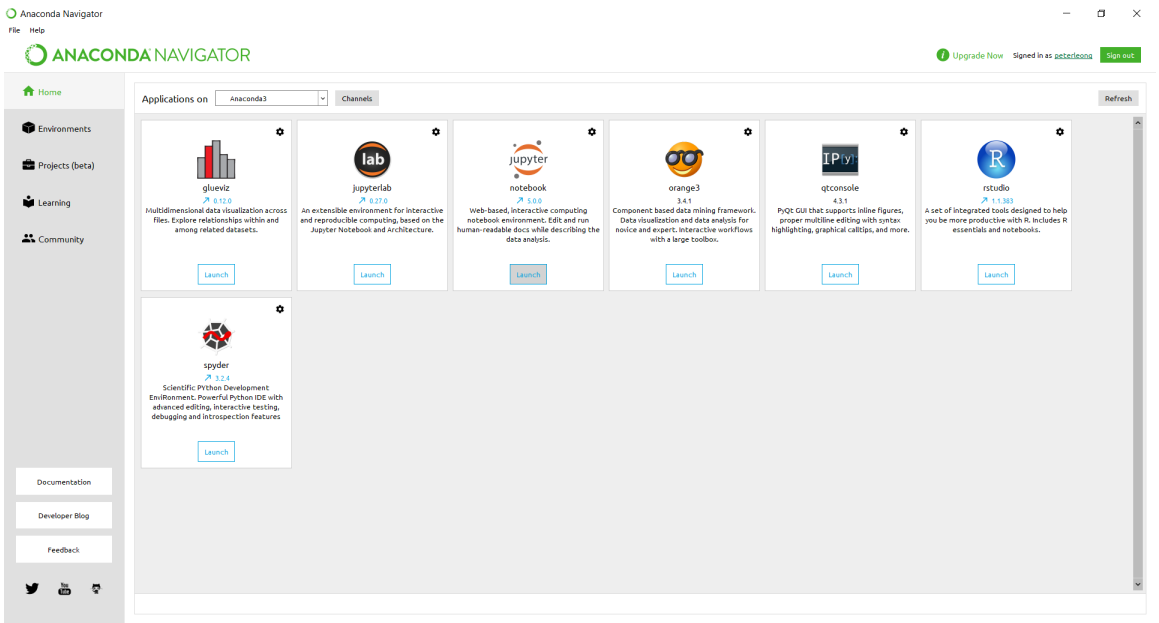
```
pip install scikit-learn
```

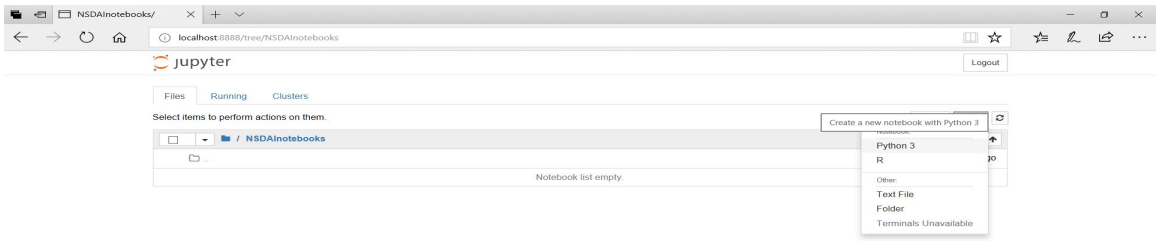
2.
USING SCIKIT-LEARN (SKLEARN)

In this section, we will go through the basics of programming using scikit-learn.

USING JUPYTER

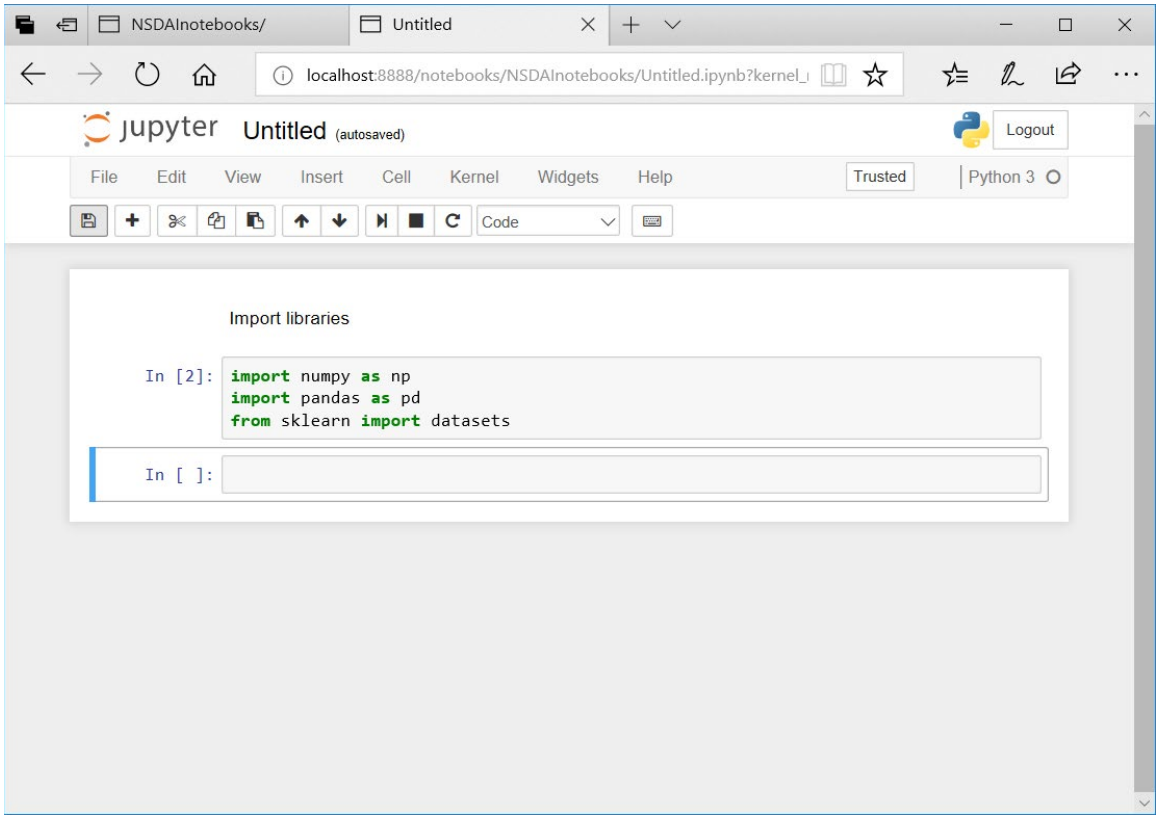
You can use scikit-learn within a jupyter notebook. Launch a python notebook from within the Anaconda Navigator.





Enter into a cell and run

```
import numpy as np
import pandas as pd
from sklearn import datasets
```



DATA LOADING

Scikit-learn comes with a few standard datasets such as iris dataset. Enter the following code to load the iris dataset.

```
# load the data
iris = datasets.load_iris()
```

Next we load the data matrix in X and the target vector in y.

```
# get the data (X) and target (y)
X, y = iris.data, iris.target
```

Print out X, y, and check the shapes.

Explore the documentation for sklearn on the load_iris() function.

3. DATA VISUALISATION AND FEATURE SELECTION

PAIRPLOT

Let us plot some charts:

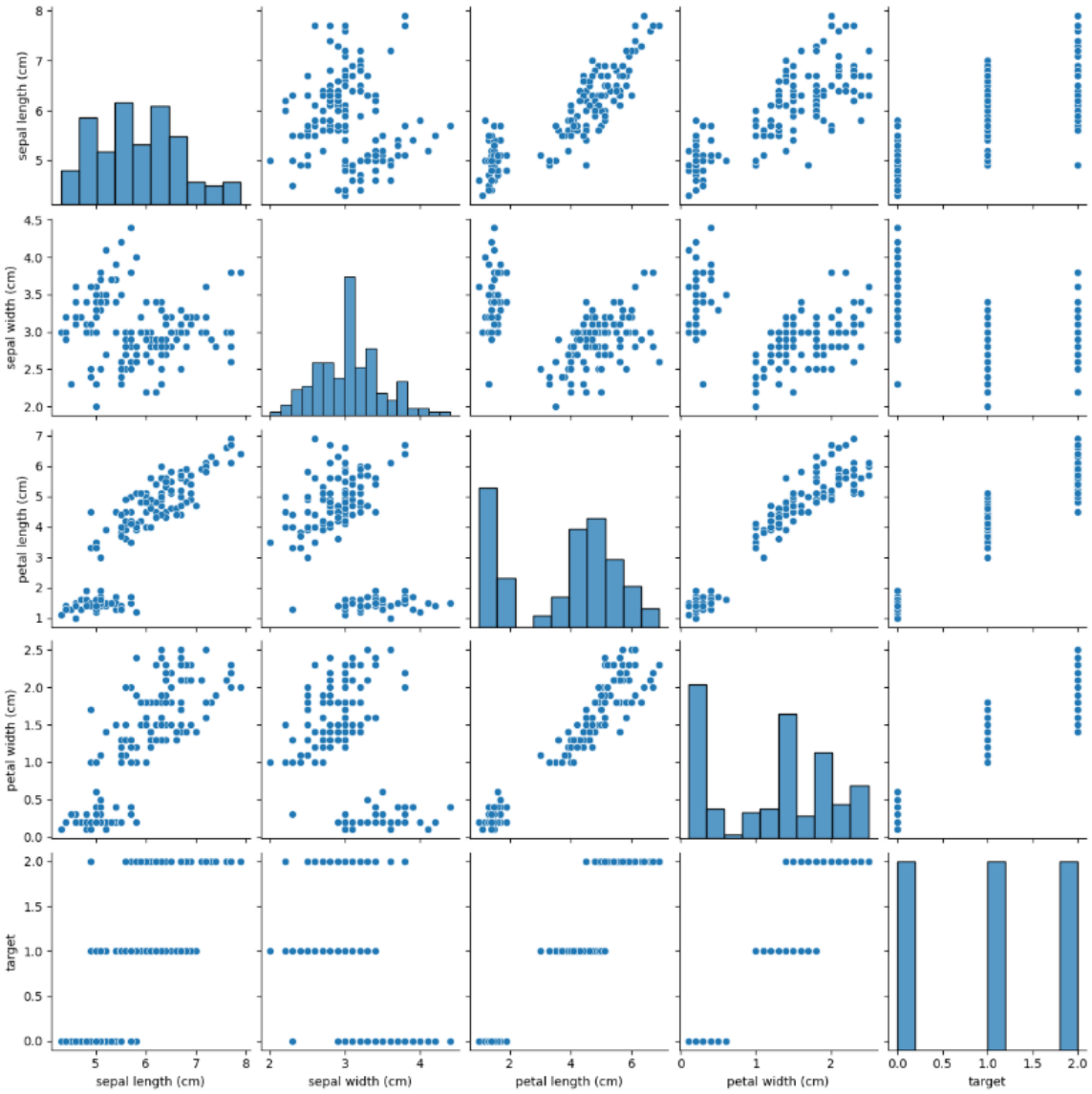
```
import seaborn as sns

data, target = datasets.load_iris(return_X_y=True, as_frame=True)
features_labels = pd.concat([data, target], axis=1)
sns.pairplot(features_labels)
```

Here, we are using the seaborn library to produce a pairplot. This pairplot shows every possible combination of features in the two-dimensional scatter plots.

This sns.pairplot() function requires a dataframe as an input, instead of numpy arrays. This is why we loaded the iris dataset as a dataframe. We then create a dataframe that combines both features and labels.

The diagonal plots are histograms of single features. If there is significant skewing, you may consider applying a log or exponential

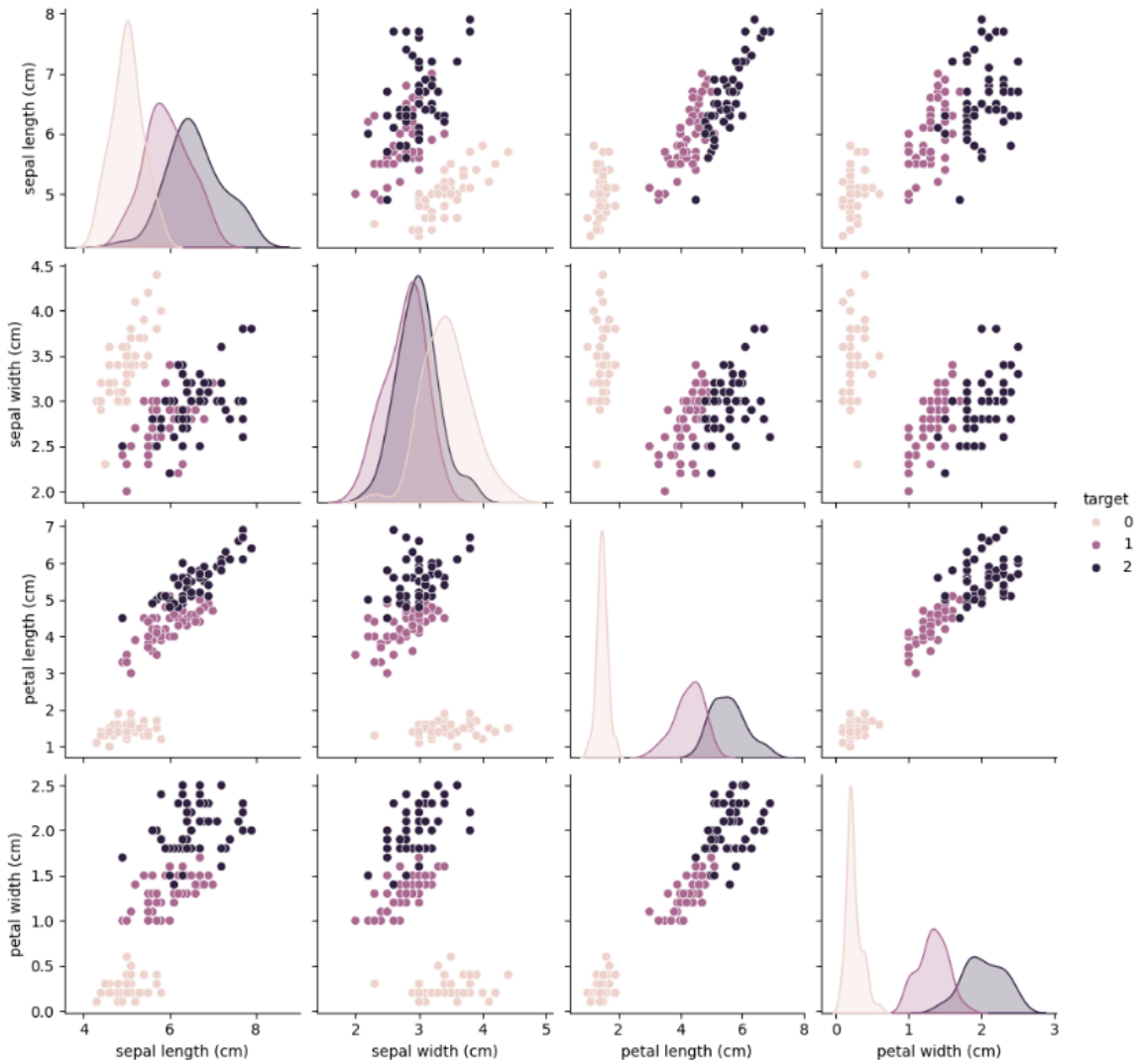
transformation (or some other appropriate transformations) to make the peak more centered.



You may choose to colour each datapoint with the target. Since we know there are three types of species, doing so would allow us to visualise how these species may be distributed with respect to each feature.

To do so, we can use seaborn to produce the pairplot, specifying hue="target" so that the datapoints are coloured by the species:

```
sns.pairplot(features_labels, hue='target')
```


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
0
1
2

CORRELATION HEATMAP

Are there features which are correlated? If so, then perhaps we can consider removing one of them, since keeping both such correlated features would be redundant, and result in more complicated models.

We can observe for such correlations from the plots produced above using pairplot. Another useful chart is the heatmap. First, let us calculate such correlations:

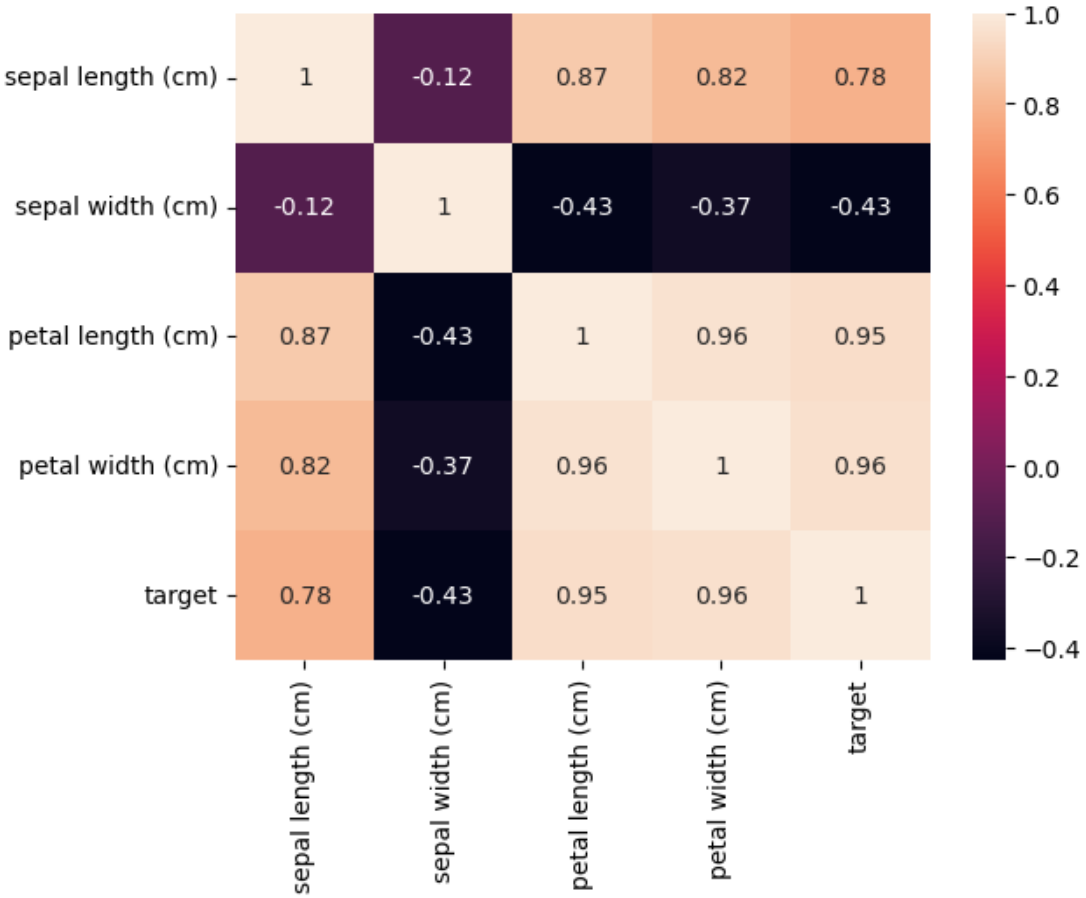
```
features_labels.corr()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126	-0.426658
petal length (cm)	0.871754	-0.428440	1.000000	0.962865	0.949035
petal width (cm)	0.817941	-0.366126	0.962865	1.000000	0.956547
target	0.782561	-0.426658	0.949035	0.956547	1.000000

The above dataframe summaries the correlations amongst various features, as well as the correlations between each feature and the target.

Instead of looking at numbers, we can also plot a heatmap to visualise this:

```
sns.heatmap(features_labels.corr(), annot=True)
```



FEATURE SELECTION

We note that petal length and petal width have high correlation, so we may consider removing them, if we want.

On the other hand, petal width is highly correlated with the target (similarly petal length is highly correlated with the target). This suggests that we may want to keep petal width (or petal length), as it is a good predictor of the target.

3.
ML EXPERIMENT WITH SKLEARN

We will load the iris data set and perform a similar experiment using a logistic regression classifier in scikit-learn. The main purpose is build a classifier to group a new sample of iris into the correct species.

DATA PREPARATION

```
# load the data
iris = datasets.load_iris()
```

Next we load the data matrix in X and the target vector in y. After that, we split the dataset into a training set and a test set using the `train_test_split` function from sklearn.model_selection.

```
# get the data (X) and target (y)
X, y = iris.data, iris.target

# split dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.20)
```

MODELLING

In this section we will show how to train the model using scikit-learn `LogisticRegression` estimator. In scikit-learn, an estimator for

classification is a Python object that implements the methods `fit(X, y)` and `predict(T)`.

```
# get the LogisticRegression estimator
from sklearn.linear_model import LogisticRegression

# training the model
# apply algorithm  to data using fit()
clf = LogisticRegression(solver='newton-cg',multi_class='multinomial')
clf.fit(X_train,y_train)
```

Model Persistence

It is possible to save a model in the scikit by using Python's built-in persistence model, namely `pickle`:

```
# We can save the trained model clf using pickle (step 4)
import pickle
pickle.dump(clf, open("iris_trained_model.p", "wb" ) )
clf2 = pickle.load(open("iris_trained_model.p", "rb" ) )
```

SCORING/PREDICTING

Scoring the model is using the trained model to score/predict the target value of new data or test data.

```
# Score the model (step 5)
y_hat = clf.predict(X_test)
```

EVALUATION

The last step is to evaluate the performance of the model for the training and test set using the confusion matrix.

```
# Evaluation of the model (step 6)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_hat)
print(cm)
```