CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Menjelaskan konsep-konsep dari pembelajaran mesin untuk kemudian bisa diterapkan pada berbagai permasalahan– (C2);
2. Mengimplementasikan algoritma dan/atau metode di pembelajaran mesin sehingga bisa mendapatkan pemecahan masalah dan model yang sesuai – (C3)
3. Menghasilkan suatu model terbaik dari permasalahan dan terus melakukan perbaikan jika terdapat perkembangan di bidang rekayasa cerdas – (C6)

MINGGU 1
Pengenalan Dasar Pembelajaran Mesin

DESKRIPSI TEMA

Mahasiswa mempelajari dasar pembelajaran mesin dan juga teknik pemrosesan dengan menggunakan Bahasa pemrograman Python menggunakan Anaconda Python.

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

Mahasiswa dapat menggunakan dan menerapkan Bahasa pemrograman untuk Pembelajaran Mesin – SCPMK-01

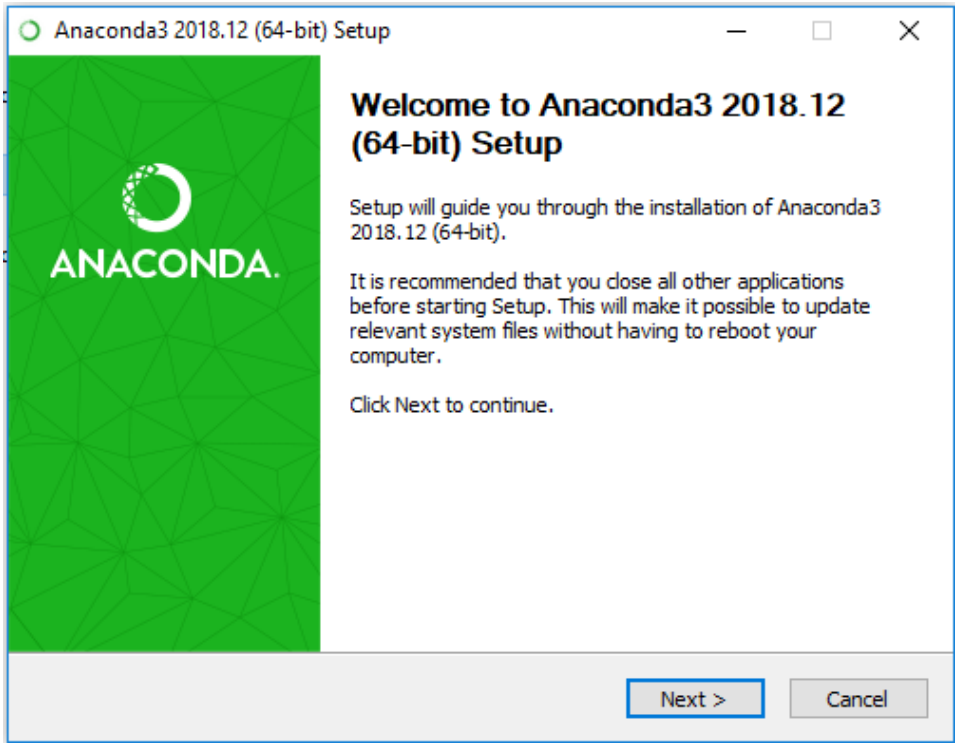
PERALATAN YANG DIGUNAKAN

Anaconda Python 3
Laptop atau Personal Computer

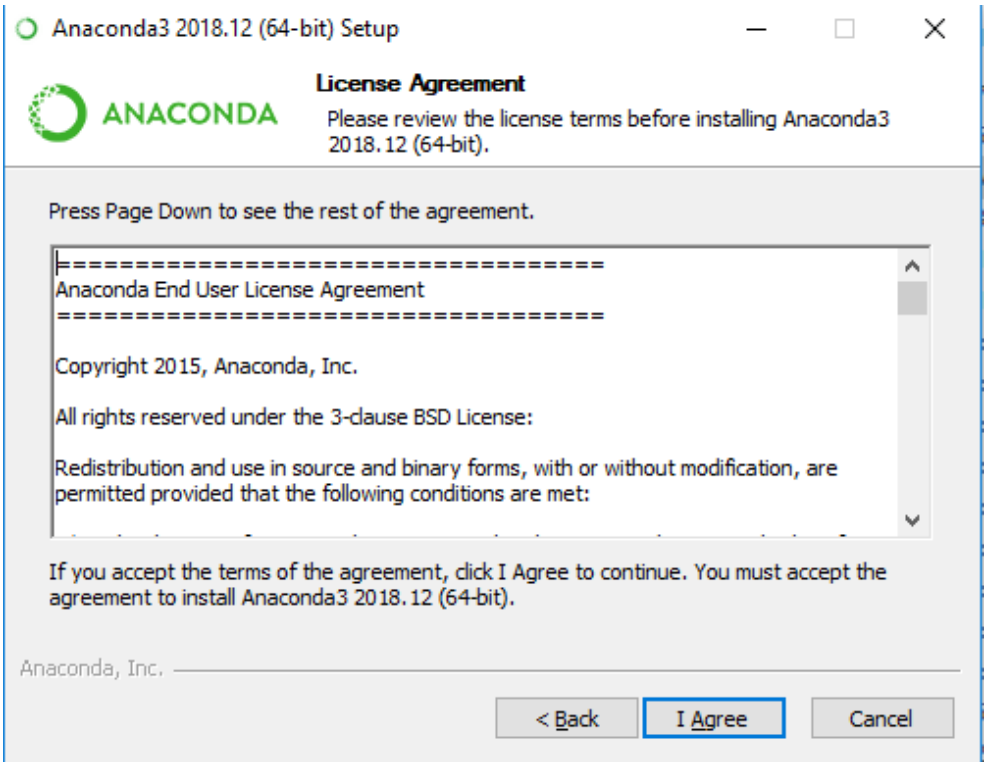
LANGKAH-LANGKAH PRAKTIKUM

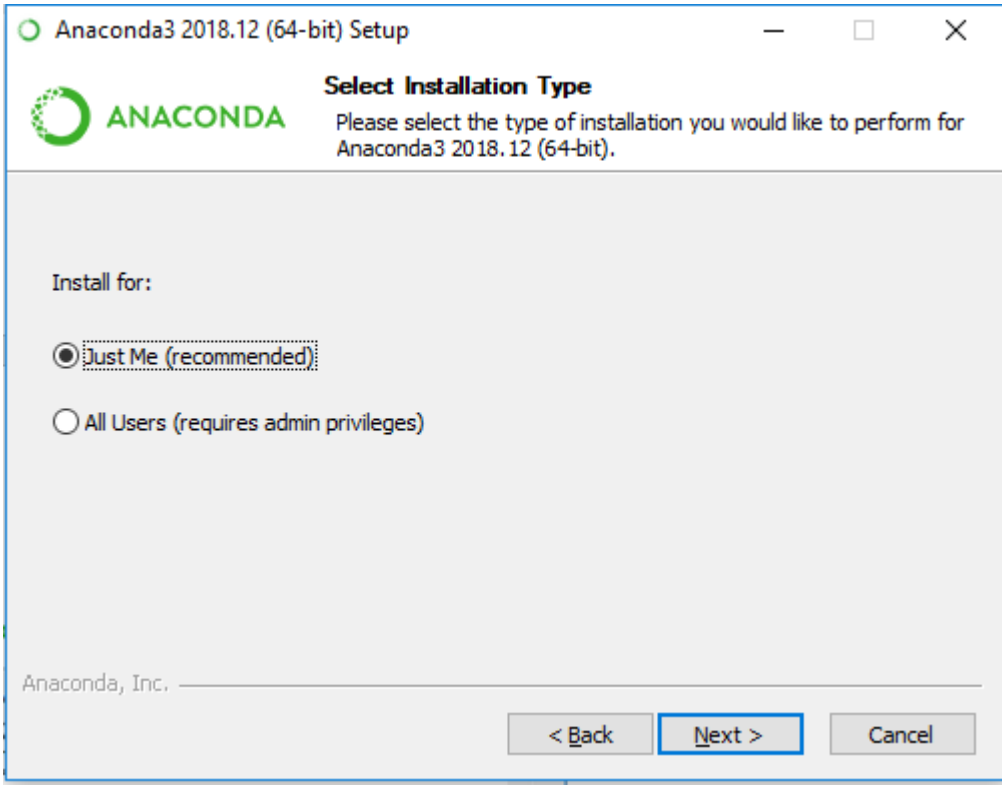
Installation

1. Download Anaconda with Python version 3.7 from the link: https://www.anaconda.com/download/
2. Run the installer.
3. Click Next.

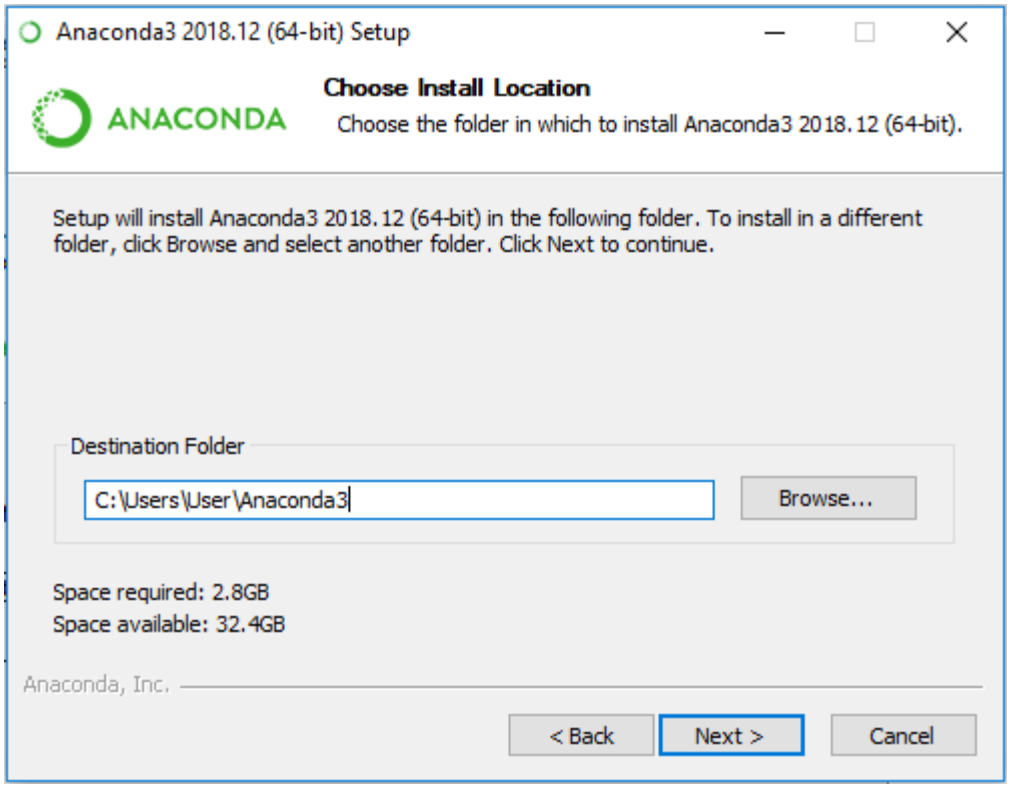


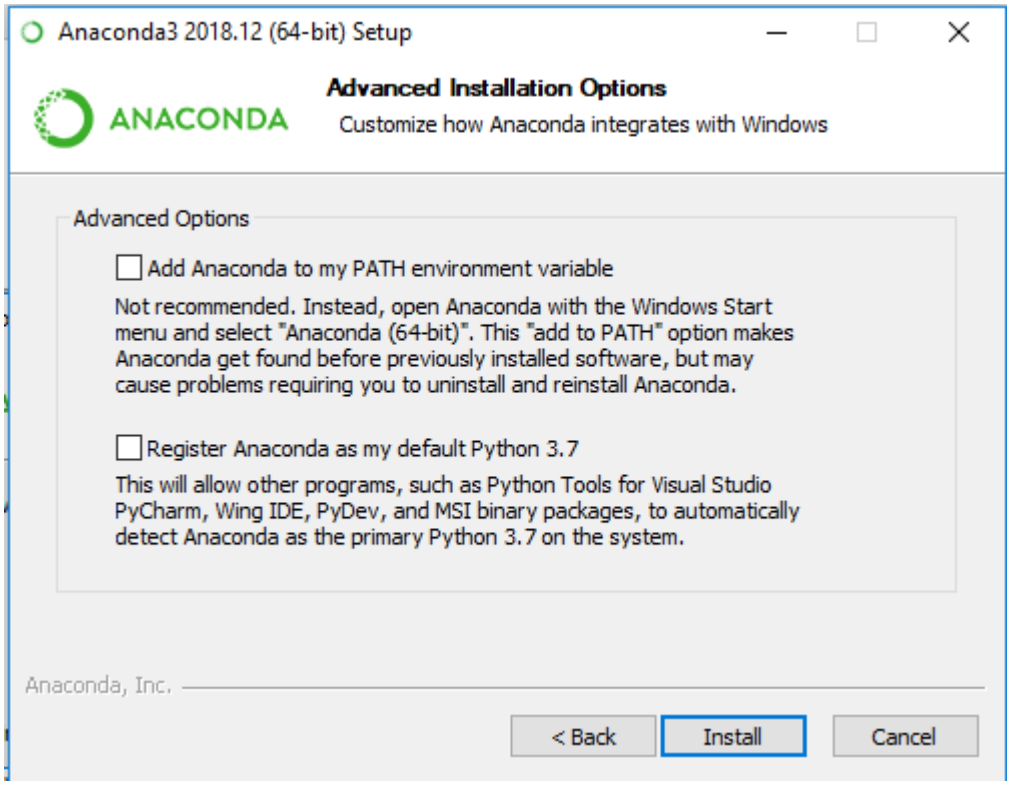
4. Choose I Agree dan Next.



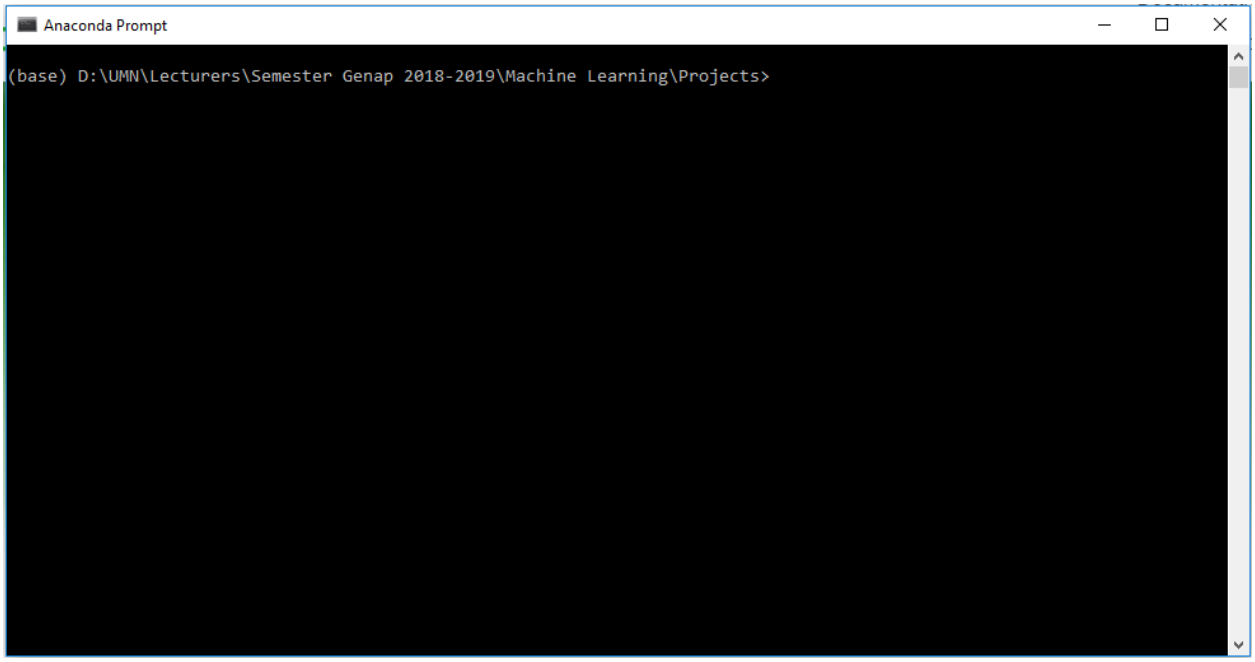


5. Choose your destination folder of Anaconda's installation, then click Install.


Anaconda3 2018.12 (64-bit) Setup
ANACONDA
Choose Install Location
Choose the folder in which to install Anaconda3 2018.12 (64-bit).
Setup will install Anaconda3 2018.12 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:\Users\User\Anaconda3
Browse...
Space required: 2.8GB
Space available: 32.4GB
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 2018.12 (64-bit) Setup
ANACONDA
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda to my PATH environment variable
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda as my default Python 3.7
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.
Anaconda, Inc.
< Back
Install
Cancel

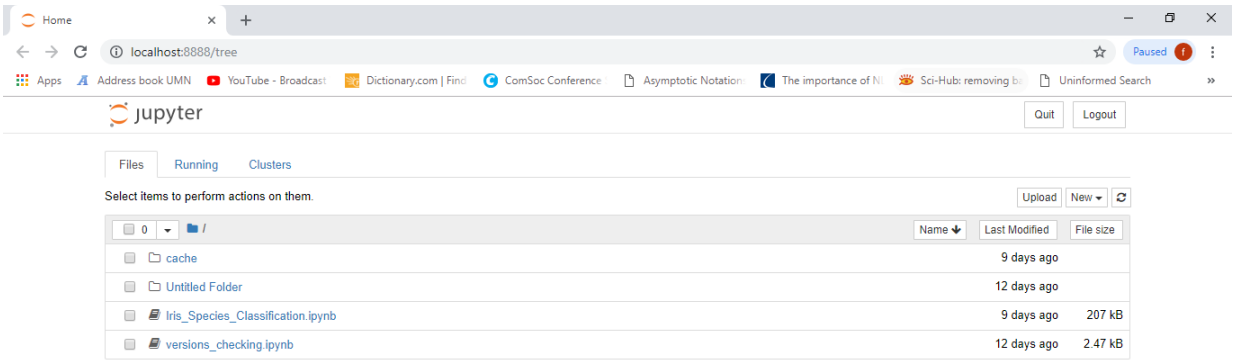
6. After the installation, run Anaconda prompt.



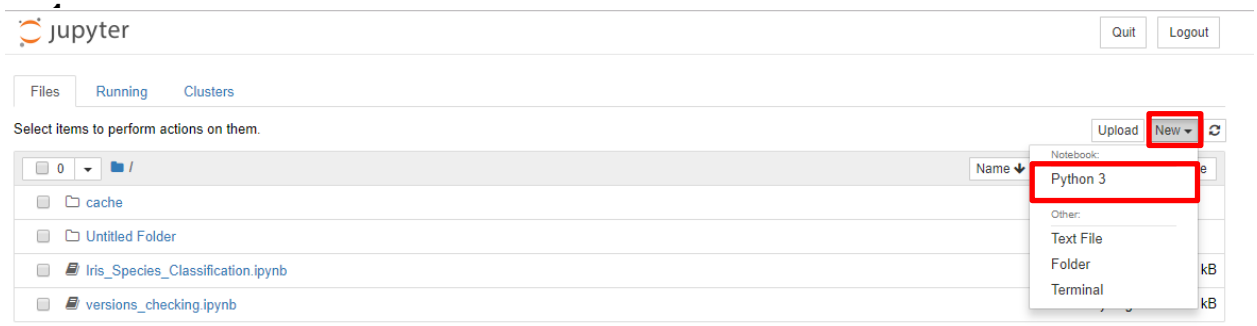
7. Run jupyter notebook from Anaconda prompt.

```
(base) D:\UMN\Lecturers\Semester Genap 2018-2019\Machine Learning\Projects>jupyter notebook
```

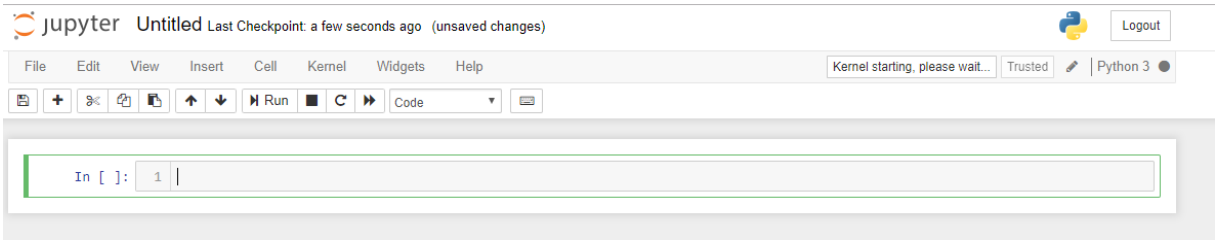
Screen will be appeared as below in web browser.



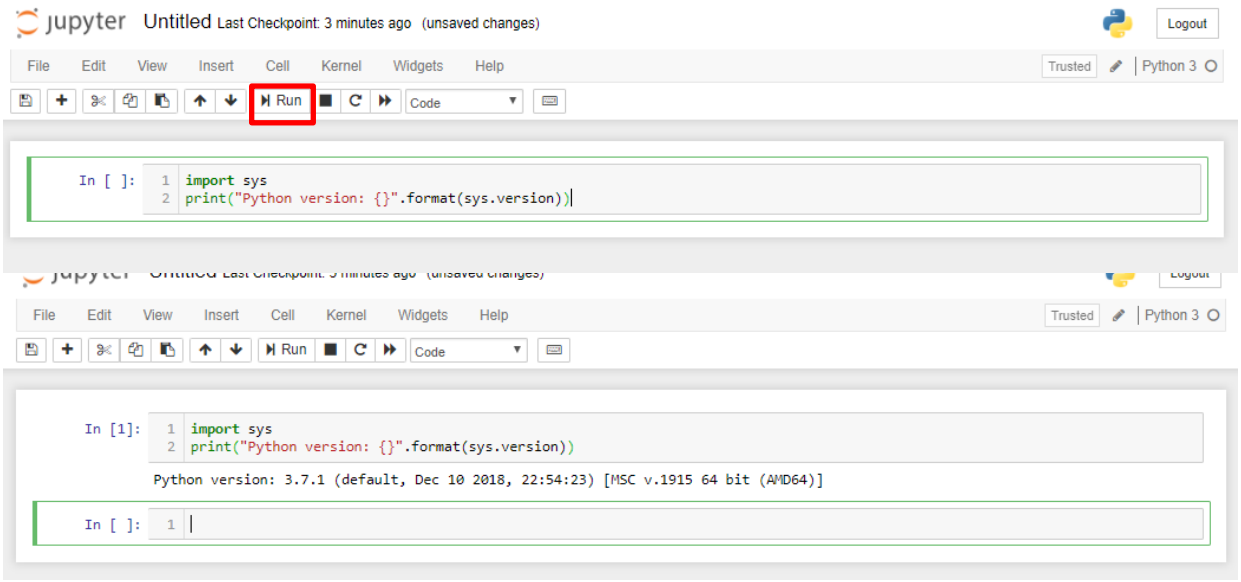
8. To create a new file of Python 3, click New on the right top of jupyter notebook. Then, choose Python 3.



Below is the new file of Python 3 on jupyter notebook.



9. Check all the libraries. Choose Run on toolbar or Alt + Enter on keyboard to run the code.



Run code below to check the version of all libraries:

.

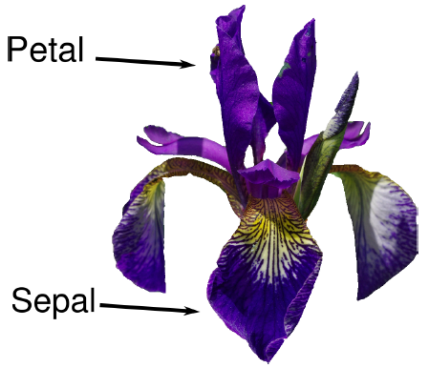
```
import pandas as pd
print("Pandas version: {}". format(pd.__version__))

import matplotlib as mp
print("Matplotlib version: {}". format(mp.__version__))

import numpy as np
print("Numpy version: {}".format(np.__version__))

import scipy as sp
print("Scipy version: {}".format(sp.__version__))

import sklearn
print("Scikit-learn version:
{}".format(sklearn.__version__))
```

```
In [1]:
import sys
print("Python version: {}".format(sys.version))
```
```
Python version: 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)]
```
```
In [2]:
import pandas as pd
print("Pandas version: {}". format(pd.__version__))
```
```
Pandas version: 0.23.4
```
```
In [3]:
import matplotlib as mp
print("Matplotlib version: {}". format(mp.__version__))
```
```
Matplotlib version: 3.0.2
```
```
In [4]:
import numpy as np
print("Numpy version: {}".format(np.__version__))
```
```
Numpy version: 1.15.4
```
```
In [6]:
import scipy as sp
print("Scipy version: {}".format(sp.__version__))
```
```
Scipy version: 1.1.0
```
```
In [7]:
import sklearn
print("Scikit-learn version: {}".format(sklearn.__version__))
```
```
Scikit-learn version: 0.20.1
```

Case 1: Classifying Iris Species

Let's assume that a hobby botanist is interested in distinguishing the species of some iris flowers that she has found. She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimeters. She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica. For these measurements, she can be certain of which species each iris belongs to.

Let's assume that these are the only species our hobby botanist will encounter in the wild. Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.



1. Load iris data from sklearn datasets.

```
In [2]:
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

The iris object that is returned by load_iris is a Bunch object, which is very similar to a dictionary. It contains keys and values.

2. Get the keys of iris dataset.
3. Show a description of iris dataset.
   The value of the key DESCR is a short description of the dataset. We show the beginning of the description here:

```
In [4]: print(iris_dataset['DESCR'][:193]+"\n...")

Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive att
...
```

4. Print target names of iris dataset. The value of the key target_names is an array of strings, containing the species of flower that we want to predict:

```
In [5]: print("Target names: {}".format(iris_dataset['target_names']))

Target names: ['setosa' 'versicolor' 'virginica']
```

5. Print feature names of iris dataset. The value of feature_names is a list of strings, giving the description of each feature:

```
In [6]: print("Feature names: \n{}".format(iris_dataset['feature_names']))

Feature names:
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

6. Print type and shape of data. The data itself is contained in the target and data fields. data contains the numeric measurements of sepal length, sepal width, petal length, and petal width in a NumPy array:

```
In [8]: print("Shape of data: {}".format(iris_dataset['data'].shape))

Shape of data: (150, 4)
```

```
In [7]: print("Type of data: {}".format(type(iris_dataset['data'])))

Type of data: <class 'numpy.ndarray'>
```

We see that the array contains measurements for 150 different flowers. Remember that the individual items are called samples in machine learning, and their properties are called features. The shape of the data array is the number of samples multiplied by the number of features.

7. Print the feature values for the first five samples.

```
In [9]: print("First five columns of data:\n{}".format(iris_dataset['data'][:5]))

First five columns of data:
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
```

From this data, we can see that all of the first five flowers have a petal width of 0.2 cm and that the first flower has the longest sepal, at 5.1 cm.

8. Print type and shape of target. The target array contains the species of each of the flowers that were measured, also as a NumPy array:

```
In [10]: print("Type of target:{}".format(type(iris_dataset['target'])))

Type of target:<class 'numpy.ndarray'>
```

9. Target is a one-dimensional array, with one entry per flower:

```
In [11]: print("Shape of target: {}".format(iris_dataset['target'].shape))

Shape of target: (150,)
```

9. Show all the target data

```
In [12]: print("Target:\n{}".format(iris_dataset['target']))

Target:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

10. The meanings of the numbers are given by the iris['target_names'] array: 0 means setosa, 1 means versicolor, and 2 means virginica.

Training and Testing Data

11. We cannot use the data we used to build the model to evaluate it. This is because our model can always simply remember the whole training set, and will therefore always predict the correct label for any point in the training set. This “remembering” does not indicate to us whether our model will generalize well (in other words, whether it will also perform well on new data).

To assess the model's performance, we show it new data (data that it hasn't seen before) for which we have labels. This is usually done by splitting the labeled data we have collected (here, our 150 flower measurements) into two parts. One part of the data is used to build our machine learning model, and is called the training data or training set. The rest of the data will be used to assess how well the model works; this is called the test data, test set, or hold-out set.

Scikit-learn contains a function that shuffles the dataset and splits it for you: the train_test_split function. This function extracts 75% of the rows in the data as the training set, together with the corresponding labels for this data. The remaining 25% of the data, together with the remaining labels, is declared as the test set.

X is data is usually denoted with a capital X, while labels are denoted by a lowercase y.

```
In [13]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'], iris_dataset['target'], random_state=0)
```

```
In [14]: print("X_train shape: {}".format(X_train.shape))
         print("y_train shape: {}".format(y_train.shape))
         print("X_test shape: {}".format(X_test.shape))
         print("y_test shape: {}".format(y_test.shape))

         X_train shape: (112, 4)
         y_train shape: (112,)
         X_test shape: (38, 4)
         y_test shape: (38,)
```

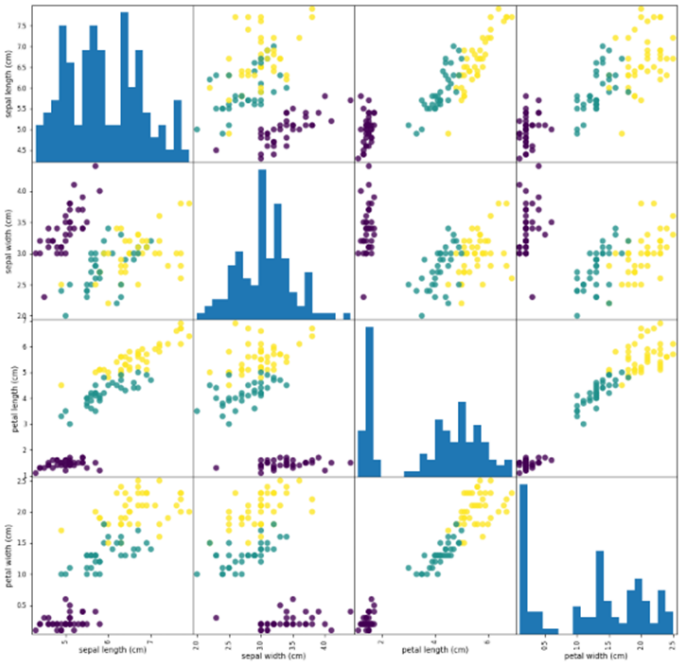
The output of the train_test_split function is X_train, X_test, y_train, and y_test, which are all NumPy arrays. X_train contains 75% of the rows of the dataset, and X_test contains the remaining 25%:

Look at Your Data

Inspecting your data is a good way to find abnormalities and peculiarities. Maybe some of your irises were measured using inches and not centimeters, for example. In the real world, inconsistencies in the data and unexpected measurements are very common. One of the best ways to inspect data is to visualize it. One way to do this is by using a scatter plot.
Build scatter plot using pandas data frame.

```
In [20]: iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
         grr = pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15,15), marker='o', hist_kwds={'bins':20}, s=60, alpha=.8)
```

12. The data points are colored according to the species the iris belongs to. To create the plot, we first convert the NumPy array into a pandas DataFrame. Pandas has a function to create pair plots called scatter_matrix. The diagonal of this matrix is filled with histograms of each feature.



From the plots, we can see that the three classes seem to be relatively well separated using the sepal and petal measurements. This means that a machine learning model will likely be able to learn to separate them.

Building the First Model: k-Nearest Neighbors

Building this model only consists of storing the training set. To make a prediction for a new data point, the algorithm finds the point in the training set that is closest to the new point. Then it assigns the label of this training point to the new data point.

13. The k-nearest neighbors classification algorithm is implemented in the KNeighborsClassifier class in the neighbors module. Before we can use the model, we need to instantiate the class into an object. This is when we will set any parameters of the model.

```
In [21]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=1)
```

14. To build the model on the training set, we call the fit method of the knn object, which takes as arguments the NumPy array X_train containing the training data and the NumPy array y_train of the corresponding training labels:

```
In [22]: knn.fit(X_train, y_train)

Out[22]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=1, n_neighbors=1, p=2,
                              weights='uniform')
```

Making Predictions

We can now make predictions using this model on new data for which we might not know the correct labels. Imagine we found an iris in the wild with a sepal length of 5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm. What species of iris would this be?

15. We can put this data into a NumPy array, again by calculating the shape—that is, the number of samples (1) multiplied by the number of features (4):

```
In [23]: X_new = np.array([[5, 2.9, 1, 0.2]])
         print("X_new.shape: {}".format(X_new.shape))

         X_new.shape: (1, 4)
```

16. To make a prediction, we call the predict method of the knn object:

```
In [24]: prediction = knn.predict(X_new)
         print("Prediction: {}".format(prediction))
         print("Predicted target name: {}".format(iris_dataset['target_names'][prediction]))

         Prediction: [0]
         Predicted target name: ['setosa']
```

Evaluating the Model

This is where the test set that we created earlier comes in. This data was not used to build the model, but we do know what the correct species is for each iris in the test set.

17. We can make a prediction for each iris in the test data and compare it against its label (the known species). We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted:

```
In [25]: y_pred = knn.predict(X_test)
         print("Test set predictions:\n {}".format(y_pred))

         Test set predictions:
          [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
          2]

In [26]: # print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
         print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))

         Test set score: 0.97
```

For this model, the test set accuracy is about 0.97, which means we made the right prediction for 97% of the irises in the test set. Under some mathematical assumptions, this means that we can expect our model to be correct 97% of the time for new irises.

REFERENSI

1. Geron A. 2017. Hands on Machine Learning wirh Scikit Learn and TensorFlow. O Reilly Media Inc
2. Van derPlas J. 2016. Pyton Data Science Handbook. Oreilly Media Inc