เอกสารประกอบการอบรมเชิงปฏิบัติการ
การเขียนโปรแกรมภาษาไพธอนสำหรับวิทยาการข้อมูล
(Python Programming for Data Science)

โดย
ผศ.ดร.โอฬาริก สุรินต๊ะ
Multi-agent Intelligent Simulation Laboratory (MISL)
Department of Information Technology, Faculty of Informatics
Mahasarakham University

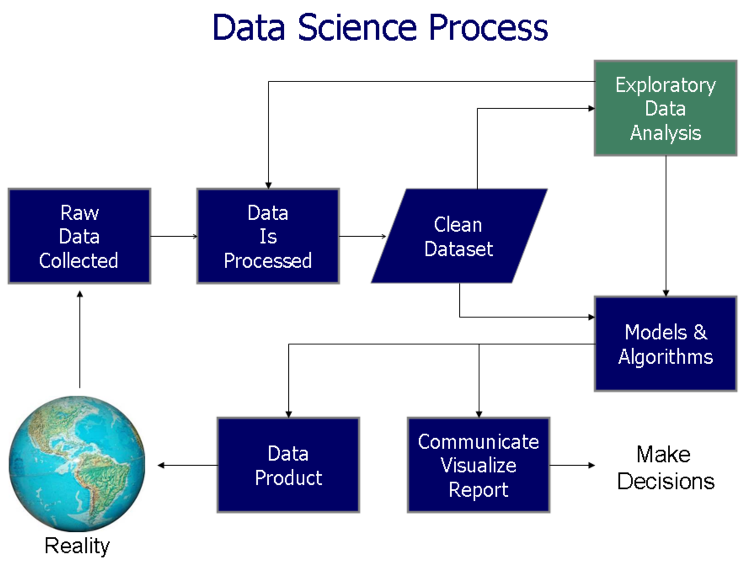
รายละเอียดการอบรม

เวลา	หัวข้ออบรม	
	วันที่ 1	วันที่ 2
9.00 - 10.30	- Basics of Python for data science - Python libraries and data structures	- Feature Selection - Cross validation - Evaluation measures - Visualizing model results
10.30 - 10.45	พักช่วงที่ 1	
10.45 - 12.00	- Exploratory data analysis - Dictionary - Pandas - First machine learning model; K-Nearest Neighbor (KNN)	- Clustering problem - K-Means - Workshop: Water Quality
12.00 - 13.00	พักรับประทานอาหารกลางวัน	
13.00 - 14.30	- Building a predictive model in Python - Regression problem - workshop: House price prediction	- Workshop: Text classification
14.30 - 14.45	พักช่วงที่ 2	

14.45 - 16.00	- Classification problem
	    - Decision tree
	    - MLP
	    - SVM
	- Workshop: weather classification (rain, no rain)
		- Deep learning for image classification
		- Workshop: image classification (handwritten digit recognition)

Table of Contents

Exploratory Data Analysis



Data type

ข้อมูลประเภท Dictionary

- สร้างตัวแปรประเภท dictionary

```python
# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

print(type(data))
print(data.keys())
```

```
<class 'dict'>
dict_keys(['Name', 'Age', 'Address', 'Qualification'])
```

- เรียกดูข้อมูลใน dictionary

```
print(data['Name'])
print(data['Age'])
```

```
['Jai', 'Princi', 'Gaurav', 'Anuj']
[27, 24, 22, 32]
```

- เรียกดูข้อมูลโดยใช้ for-loop

```
for i in range(4):
  print('Name: ', data['Name'][i])
  print('Age: ', data['Age'][i])
  print()
```

```
Name:  Jai
Age:  27

Name:  Princi
Age:  24

Name:  Gaurav
Age:  22

Name:  Anuj
Age:  32
```

pandas – Python Data Analysis Library

```
# creating pandas dataframe
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
df[['Name', 'Qualification']]
```

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

- Importing data from csv file

```python
import pandas as pd

url = 'https://www.biz.uiowa.edu/faculty/jledolter/datamining/weather.csv'
weather_data = pd.read_csv(url)

weather_data
```

แสดงข้อมูลใน DataFrame

```python
weather_data
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm
0	11/1/2007	Canberra	8.0	24.3	0.0	3.4	6.3	NW	30.0	SW	NW
1	11/2/2007	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	39.0	E	W
2	11/3/2007	Canberra	13.7	23.4	3.6	5.8	3.3	NW	85.0	N	NNE
3	11/4/2007	Canberra	13.3	15.5	39.8	7.2	9.1	NW	54.0	WNW	W
4	11/5/2007	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	50.0	SSE	ESE
...	...	...	...	...	...	...	...	...	...	...	...
361	10/27/2008	Canberra	9.0	30.7	0.0	7.6	12.1	NNW	76.0	SSE	NW
362	10/28/2008	Canberra	7.1	28.4	0.0	11.6	12.7	N	48.0	NNW	NNW
363	10/29/2008	Canberra	12.5	19.9	0.0	8.4	5.3	ESE	43.0	ENE	ENE
364	10/30/2008	Canberra	12.5	26.9	0.0	5.0	7.1	NW	46.0	SSW	WNW
365	10/31/2008	Canberra	12.3	30.2	0.0	6.0	12.6	NW	78.0	NW	WNW

366 rows × 24 columns

- Dealing with pandas library

แสดงรายชื่อของคอลัมน์ใน DataFrame

```
[ ]  1 weather_data.columns
```

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation',
       'Sunshine', 'WindGustDir', 'WindGustSpeed', 'WindDir9am', 'WindDir3pm',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
       'Temp3pm', 'RainToday', 'RISK_MM', 'RainTomorrow'],
      dtype='object')
```

แสดงขนาดของ DataFrame

```
[ ]  1 # count row and column
     2 weather_data.shape
```

```
(366, 24)
```

ตัวอย่างการ Query ข้อมูลด้วย pandas
- นับจำนวน

```
[ ]  1 # count frequency value in column
     2 print('WindGustDir = N,',len(weather_data[weather_data['WindGustDir'] == 'N']))
```

```
WindGustDir = N, 21
```

- กำหนดเงื่อนไขการค้นหา

```
[ ]  1 print(len(weather_data[weather_data['MaxTemp'] > 20]))
     2 weather_data[weather_data['MaxTemp'] > 20]
```

176

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Suns
0	11/1/2007	Canberra	8.0	24.3	0.0	3.4	
1	11/2/2007	Canberra	14.0	26.9	3.6	4.4	
2	11/3/2007	Canberra	13.7	23.4	3.6	5.8	
9	11/10/2007	Canberra	8.4	22.8	16.2	5.4	
10	11/11/2007	Canberra	9.1	25.2	0.0	4.2	
...	...	...	...	...	...	...	
360	10/26/2008	Canberra	7.9	26.1	0.0	6.8	
361	10/27/2008	Canberra	9.0	30.7	0.0	7.6	
362	10/28/2008	Canberra	7.1	28.4	0.0	11.6	

- and operation

```python
# and operation
weather_data[(weather_data['MaxTemp'] > 10) & (weather_data['MaxTemp'] < 20)]
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGu
3	11/4/2007	Canberra	13.3	15.5	39.8	7.2	9.1	NW	
4	11/5/2007	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	
5	11/6/2007	Canberra	6.2	16.9	0.0	5.8	8.2	SE	
6	11/7/2007	Canberra	6.1	18.2	0.2	4.2	8.4	SE	
7	11/8/2007	Canberra	8.3	17.0	0.0	5.6	4.6	E	
...	...	...	...	...	...	...	...	...	
342	10/8/2008	Canberra	0.5	17.9	0.0	5.8	11.5	N	

- sort

```python
# sort
weather_data.sort_values(by=['MinTemp'], ascending=True)
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshi
292	8/19/2008	Canberra	-5.3	13.1	0.0	2.2	7
297	8/24/2008	Canberra	-3.7	14.4	0.0	2.6	10
313	9/9/2008	Canberra	-3.7	14.7	0.0	3.4	10
265	7/23/2008	Canberra	-3.5	11.2	0.0	1.6	7
283	8/10/2008	Canberra	-3.5	7.6	0.4	2.4	4
...	...	...	...	...	...	...	
76	1/16/2008	Canberra	17.9	33.2	0.0	10.4	8
90	1/30/2008	Canberra	18.0	34.9	0.0	9.2	9

Missing value

- วิธี KNNImputer

```python
# missing value
import numpy as np
from sklearn.impute import KNNImputer

nan = np.nan
X = [[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]
print(X)
print()

imputer = KNNImputer(n_neighbors=3, weights="uniform")
X = imputer.fit_transform(X)

df = pd.DataFrame(X)
df
```

```
[[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]
```

	0	1	2
0	1.0	2.0	5.0
1	3.0	4.0	3.0
2	4.0	6.0	5.0
3	8.0	8.0	7.0

- วิธี SimpleImputer

```python
from sklearn.impute import SimpleImputer

nan = np.nan
X = [[1, 2, nan], [3, 4, 3], [nan, 6, 5], [8, 8, 7]]

imp = SimpleImputer(missing_values=nan, strategy='mean')
imp.fit(X)
X = imp.transform(X)

df = pd.DataFrame(X)
df
```

	0	1	2
0	1.0	2.0	5.0
1	3.0	4.0	3.0
2	4.0	6.0	5.0
3	8.0	8.0	7.0

Cleaning data

- ลบ column ที่ไม่เกี่ยวข้อง

```
[ ]
# Getting rid of the columns with objects which will not be used in our model:
weather_data.drop(['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RISK_MM'], axis=1, inplace=True)
weather_data.head(5)
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pre
0	8.0	24.3	0.0	3.4	6.3	30.0	6.0	20	68	29	
1	14.0	26.9	3.6	4.4	9.7	39.0	4.0	17	80	36	
2	13.7	23.4	3.6	5.8	3.3	85.0	6.0	6	82	69	

- แทนค่าข้อมูลที่เป็น NaN values

```
[ ]
# And we need to replace NaN values with mean values of each column:
weather_data.fillna(weather_data.mean(), inplace=True)
weather_data.head(5)
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9a
0	8.0	24.3	0.0	3.4	6.3	30.0	6
1	14.0	26.9	3.6	4.4	9.7	39.0	4
2	13.7	23.4	3.6	5.8	3.3	85.0	6

Converting predictions to binary for machine learning algorithm

```
[ ]
weather_data.RainToday = [1 if each == 'Yes' else 0 for each in weather_data.RainToday]
weather_data.RainTomorrow = [1 if each == 'Yes' else 0 for each in weather_data.RainTomorrow]
#weather_data.sample(5)
weather_data.head()
```

WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RainTomorrow
6.0	20	68	29	1019.7	1015.0	7	7	14.4	23.6	0	1
4.0	17	80	36	1012.4	1008.4	5	3	17.5	25.7	1	1
6.0	6	82	69	1009.5	1007.2	8	7	15.4	20.2	1	1
30.0	24	62	56	1005.5	1007.0	2	7	13.5	14.1	1	1
20.0	28	68	49	1018.3	1018.5	7	7	11.1	15.4	1	0

Create label (y)

```
[ ]  1 y = weather_data.RainTomorrow.values
     2 x_data = weather_data.drop('RainTomorrow', axis=1)
     3 x_data.head()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am
0	8.0	24.3	0.0	3.4	6.3	30.0	6.0
1	14.0	26.9	3.6	4.4	9.7	39.0	4.0
2	13.7	23.4	3.6	5.8	3.3	85.0	6.0
3	13.3	15.5	39.8	7.2	9.1	54.0	30.0
4	7.6	16.1	2.8	5.6	10.6	50.0	20.0

แสดงข้อมูล label

```
[ ]  1 print(y)
```

```
[1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1
 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0
 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1
 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Data Normalization

Normalization formula = (x - min(x)) / (max(x) - min(x))

```
[ ]  1 # In order to scale all the features between 0 and 1:
     2 x_norm = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data))
     3 x_norm.head(5)
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpee
0	0.507634	0.592199	0.000000	0.235294	0.463235	0.200000	0.14
1	0.736641	0.684397	0.090452	0.308824	0.713235	0.305882	0.09
2	0.725191	0.560284	0.090452	0.411765	0.242647	0.847059	0.14
3	0.709924	0.280142	1.000000	0.514706	0.669118	0.482353	0.73
4	0.492366	0.301418	0.070352	0.397059	0.779412	0.435294	0.48

[ ]
```python
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
scaler.fit(x_data)
x_scaler = scaler.transform(x_data)

x_scaler = pd.DataFrame(data=x_scaler, columns=weather_data.columns[0:-1])
x_scaler.head()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	Wi
0	0.122047	0.561221	-0.338485	-0.420844	-0.464807	-0.756615	-0.464339	
1	1.119129	0.950363	0.514591	-0.045713	0.517159	-0.064635	-0.718645	
2	1.069275	0.426518	0.514591	0.479471	-1.331248	3.472147	-0.464339	
3	1.002802	-0.755874	9.092744	1.004655	0.343871	1.088663	2.587333	
4	0.055575	-0.666072	0.325018	0.404445	0.777092	0.781117	1.315803	

[ ]
```python
norm = preprocessing.Normalizer()
norm.fit(x_data)
x_scaler = norm.transform(x_data)

x_scaler = pd.DataFrame(data=x_scaler, columns=weather_data.columns[0:-1])
x_scaler.head()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	W:
0	0.005549	0.016856	0.000000	0.002358	0.004370	0.020809	0.004162	
1	0.009770	0.018772	0.002512	0.003071	0.006769	0.027216	0.002791	
2	0.009559	0.016328	0.002512	0.004047	0.002303	0.059311	0.004187	
3	0.009314	0.010855	0.027872	0.005042	0.006373	0.037817	0.021009	
4	0.005262	0.011148	0.001939	0.003877	0.007339	0.034620	0.013848	

Dividing data: Training and Test sets

```
# importing sklearn's library for splitting our dataset:
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_norm, y, test_size=0.2, random_state=75)

print('x_train shape is: ', x_train.shape)
print('y_train shape is: ', y_train.shape)
print('x_test shape is: ', x_test.shape)
print('y_test shape is: ', y_test.shape)
```

```
x_train shape is:  (292, 17)
y_train shape is:  (292,)
x_test shape is:  (74, 17)
y_test shape is:  (74,)
```

First machine learning

K-Nearest Neighbor (KNN)

```
[ ]
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=7)
clf.fit(x_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=7, p=2,
                     weights='uniform')
```

```
[ ]
y_pred = clf.predict(x_test)
```

```
[ ]
print('data', x_test[0:1])
print('label', y_test[0])
```

```
data      MinTemp  MaxTemp  Rainfall  ...   Temp9am   Temp3pm  RainToday
243  0.40458  0.14539  0.050251  ...  0.276423  0.204082        1.0

[1 rows x 17 columns]
label 0
```

- แสดงผลลัพธ์

```
[ ]
#0 = No, 1 = Yes
if(clf.predict(x_test[1:2])):
  print('It will rain tomorrow')
else:
  print('It will not rain tomorrow')
```

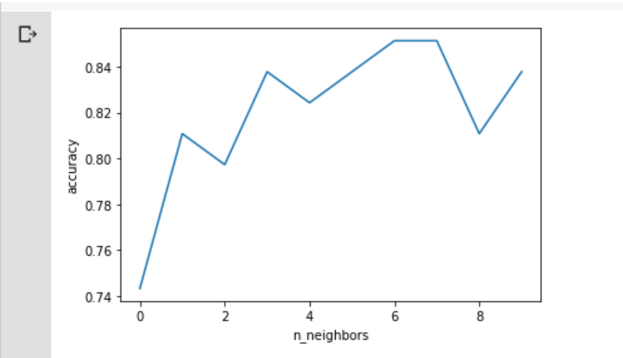
```
It will not rain tomorrow
```

Accuracy score

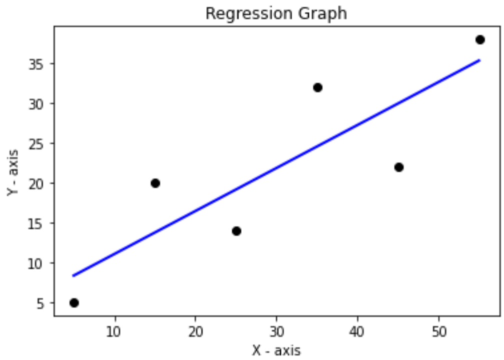
```
[ ]
from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

```
0.8513513513513513
```

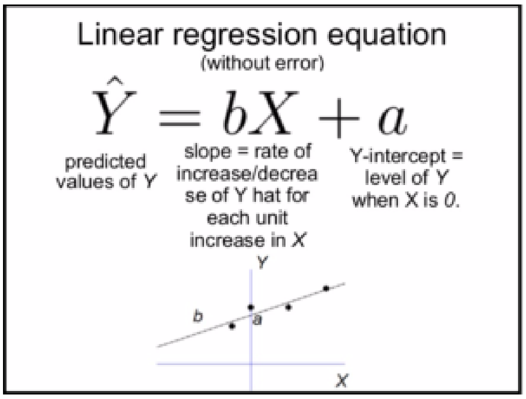
```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

ss = []
for i in range(1,11):
  clf = KNeighborsClassifier(n_neighbors=i)
  clf.fit(x_train, y_train)
  y_pred_knn = clf.predict(x_test)
  ss.append(accuracy_score(y_test, y_pred_knn))

plt.plot(ss)
plt.ylabel('accuracy')
plt.xlabel('n_neighbors')
plt.show()
```



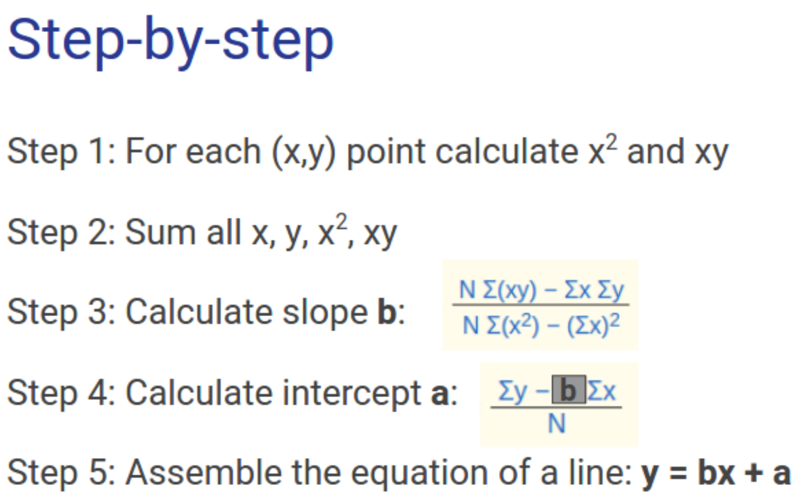
Building a predictive model



Linear regression using least squares method



Step-by-step



- Data

"x" Hours of Sunshine	"y" Ice Creams Sold
2	4
3	5
5	7
7	10
9	15

- Calculation

Step 1: For each (x,y) calculate x^2 and xy:

x	y	x^2	xy
2	4	4	8
3	5	9	15
5	7	25	35
7	10	49	70
9	15	81	135

Step 2: Sum x, y, x^2 and xy (gives us Σx, Σy, Σx^2 and Σxy):

x	y	x^2	xy
2	4	4	8
3	5	9	15
5	7	25	35
7	10	49	70
9	15	81	135
Σx: 26	**Σy: 41**	**Σx^2: 168**	**Σxy: 263**

Also **N** (number of data values) = **5**

Step 3: Calculate Slope $\boxed{b}$

$$\boxed{b} = \frac{N\,\Sigma(xy) - \Sigma x\,\Sigma y}{N\,\Sigma(x^2) - (\Sigma x)^2}$$

$$= \frac{5 \times 263 - 26 \times 41}{5 \times 168 - 26^2}$$

$$= \frac{1315 - 1066}{840 - 676}$$

$$= \frac{249}{164} = 1.5183...$$

Step 4: Calculate Intercept $\boxed{a}$

$$\boxed{a} = \frac{\Sigma y - \boxed{b}\,\Sigma x}{N}$$

$$= \frac{41 - 1.5183 \times 26}{5}$$

$$= 0.3049...$$

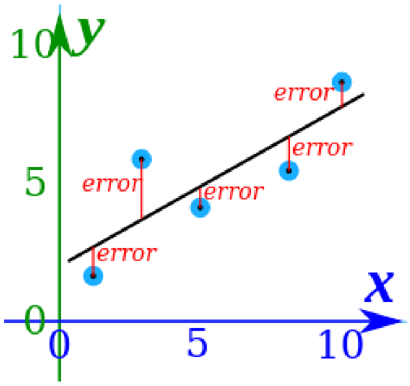
Step 5: Assemble the equation of a line:

y = bx + a

y = 1.518x + 0.305

Evaluation

Square of the errors



error = y' - y

x	y	y = 1.518x + 0.305	error
2	4	3.34	−0.66
3	5	4.86	−0.14
5	7	7.89	0.89
7	10	10.93	0.93
9	15	13.97	−1.03

Mean square error - MSE

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

where N is the number of data points, f_i the value returned by the model and y_i the actual value for data point i.

MSE

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

x	y	y = 1.518x + 0.305	error	(y' - y) ^2
2	4	3.34	−0.66	0.4356
3	5	4.86	−0.14	0.0196
5	7	7.89	0.89	0.7921
7	10	10.93	0.93	0.8649
9	15	13.97	−1.03	1.0609

Sum = 3.1731

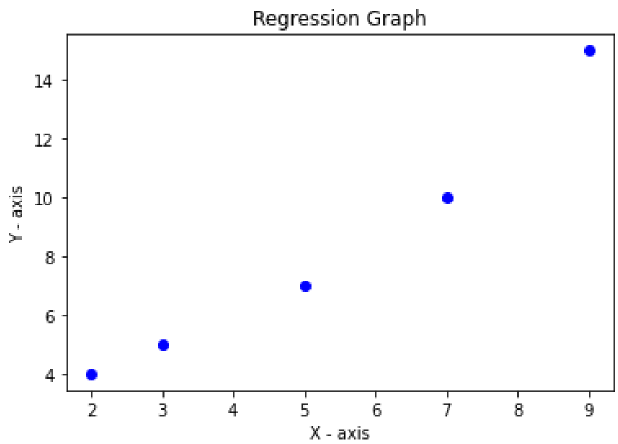
MSE = 3.1731/5
= 0.63462

5 steps to linear regression

```
[ ]
import numpy as np

# create dataset and label
x = np.array([2, 3, 5, 7, 9]).reshape((-1, 1))
y = np.array([4, 5, 7, 10, 15])

print('training set', x, sep='\n')
print('label', y, sep='\n')
```

```
training set
[[2]
 [3]
 [5]
 [7]
 [9]]
label
[ 4  5  7 10 15]
```

```
# regression graph
import matplotlib.pyplot as plt
import math

plt.plot(x,y,'o',color='blue')
plt.xlabel('X - axis')
plt.ylabel('Y - axis')
plt.title('Regression Graph')
plt.show()
```



Step 1: calculate x^2 and xy

```
# step 1: For each (x,y) point calculate x^2 and xy
x2 = np.power(x, 2)
xy = x * y.reshape((-1,1))
#xy = np.multiply(x,y.reshape((-1,1)))

print('x2', x2, sep='\n')
print('xy', xy, sep='\n')
```

```
x2
[[ 4]
 [ 9]
 [25]
 [49]
 [81]]
xy
[[  8]
 [ 15]
 [ 35]
 [ 70]
 [135]]
```

Step 2: sum all x, y, x^2, xy

```
[ ]
# step 2: sum all x, y, x^2, xy
sum_x = np.sum(x)
sum_y = np.sum(y)
sum_x2 = np.sum(x2)
sum_x_2 = np.power(sum_x, 2)
sum_xy = np.sum(xy)
N = np.count_nonzero(x)

print('sum_x ', sum_x)
print('sum_y ', sum_y)
print('sum_x2 ', sum_x2)
print('sum_x_2 ', sum_x_2)
print('sum_xy ', sum_xy)
print('N ', N)
```

```
sum_x  26
sum_y  41
sum_x2  168
sum_x_2  676
sum_xy  263
N  5
```

Step 3: calculate slop - b

```
[ ]
# step 3: calculate slope - b
b = ((N * sum_xy) - (sum_x * sum_y)) / ((N * sum_x2) - sum_x_2)
print('slope (b): ', b)
```

```
slope (b):  1.5182926829268293
```

Step 4: calculate intercept - a

```
[ ]
# step 4: calculate intercept - a
a = (sum_y - (b * sum_x)) / N
print('intercept (a): ', a)
```

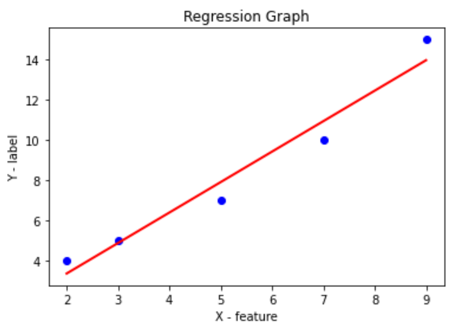
```
intercept (a):  0.30487804878048763
```

Step 5: assemble the equation of a line: y = bx + a

```
[ ]
# step 5: assemble the equation of a line: y = bx + a

# new data
new_data = np.copy(x)
y_pred = (b * new_data) + a
print('Prediction value ', y_pred, sep='\n')
```

```
Prediction value
[[ 3.34146341]
 [ 4.8597561 ]
 [ 7.89634146]
 [10.93292683]
 [13.9695122 ]]
```

```
# regression graph
import matplotlib.pyplot as plt
import math

plt.plot(x,y,'o',color='blue')
plt.plot(x, y_pred, color='red', linewidth=2)
plt.xlabel('X - feature')
plt.ylabel('Y - label')
plt.title('Regression Graph')
plt.show()
```



Predict new data

```
[ ]
new_data = np.array([8])
y_pred_new = (b * new_data) + a
print('Prediction value', y_pred_new, sep='\n')
```

```
Prediction value
[12.45121951]
```

Create linear regression class and function

Create class

Method
- fit – Fit linear model
- predict – Predict using the linear model

```
[ ]
class lr():
  """
  Linear regression using least squares method
  """

  def fit(self, x, y):
    """
    Fit model

    Parameters
    ----------
    X : array_like, shape (n_samples, n_features)
        Training data
    y : array_like, shape (n_samples, )
        Target value

    Return
    ----------
    self : returns an instance of self
    """
```

```

    x2 = np.power(x, 2)
    xy = x * y.reshape((-1,1))
    sum_x = np.sum(x)
    sum_y = np.sum(y)
    sum_x2 = np.sum(x2)
    sum_x_2 = np.power(sum_x, 2)
    sum_xy = np.sum(xy)
    self.n_samples = np.count_nonzero(x)

    self.slope_ = ((self.n_samples * sum_xy) - (sum_x * sum_y)) / ((self.n_samples * sum_x2) - sum_x_2)
    self.intercept_ = (sum_y - (self.slope_ * sum_x)) / self.n_samples

    return self
```

```
    def predict(self, x):
      """
      Predict using the linear model

      Parameters
      ----------
      X : array_like or spare matrix, shape (n_samples, n_features)
          Sample

      Returns
      ----------
      C : array, shape (n_samples, )
          Returns predicted value
      """

      self.y_pred = (self.slope_ * x) + self.intercept_

      return self.y_pred
```

Create an instance of class and calling method

- สร้างข้อมูล training data

```
# create training data
x_train = np.array([2, 3, 5, 7, 9]).reshape((-1, 1))
y_train = np.array([4, 5, 7, 10, 15])
```

- สร้าง instance of class และ calling methods

```
# creating an instance of class
clf = lr()

# calling methods
# fit model
clf.fit(x_train, y_train)
# predict using linear model
y_pred = clf.predict(x_train)

print('x_train \t predict', np.hstack((x_train.reshape(-1,1), y_pred.reshape(-1,1))), sep='\n')
```

```
x_train          predict
[[ 2.          3.34146341]
 [ 3.          4.8597561 ]
 [ 5.          7.89634146]
 [ 7.         10.93292683]
 [ 9.         13.9695122 ]]
```

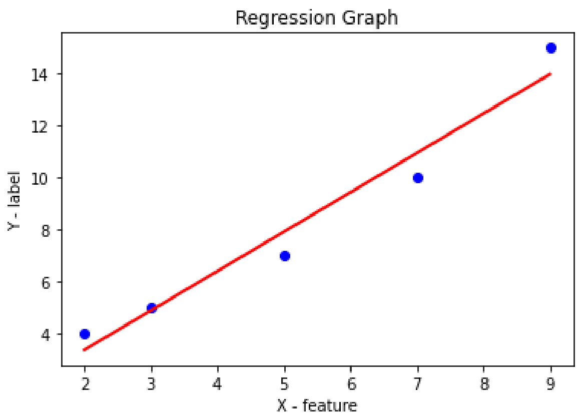
Create visualize function

- สร้างฟังก์ชัน – def

```
# regression graph
import matplotlib.pyplot as plt
import math

def lr_visual(x, y, y_pred):
  plt.plot(x,y,'o',color='blue')
  plt.plot(x, y_pred, color='red', linewidth=2)
  plt.xlabel('X - feature')
  plt.ylabel('Y - label')
  plt.title('Regression Graph')
  plt.show()
```

- เรียกใช้ฟังก์ชัน

```
[ ]  lr_visual(x_train, y_train, y_pred)
```



Predict unknown data

```
[ ]  # create test data
     x_test = np.array([1,6,8,8,9])

     # predict using linear model
     y_test_pred = clf.predict(x_test)

     print('x_test \t\t predict', np.hstack((x_test.reshape(-1,1), y_test_pred.reshape(-1,1))), sep='\n')
```

```
x_test          predict
[[ 1.          1.82317073]
 [ 6.          9.41463415]
 [ 8.         12.45121951]
 [ 8.         12.45121951]
 [ 9.         13.9695122 ]]
```

Using linear regression package from scikit-learn

Import packages and classes

```
[ ]
# import packages and classes
import numpy as np
from sklearn.linear_model import LinearRegression
```

Provide data

```
[ ]
# provide data
x = np.array([5, 15, 25, 35, 45, 55]).reshape((-1, 1))
y = np.array([5, 20, 14, 32, 22, 38])

print(x)
print(y)
```

```
[[ 5]
 [15]
 [25]
 [35]
 [45]
 [55]]
[ 5 20 14 32 22 38]
```

Create linear model and fit

```
# create linear model and fit
lr = LinearRegression()
lr.fit(x, y)
#lr = LinearRegression().fit(x, y)

print('slope =', lr.coef_[0], 'intercept =', lr.intercept_)
```

```
slope = 0.54 intercept = 5.633333333333329
```

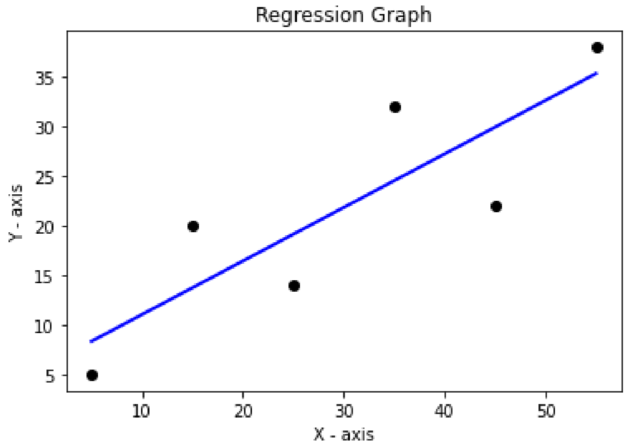
Predict unknown data

```python
# predict
y_pred = lr.predict(x)
print('predicted:', y_pred, sep='\n')

# equation : y = bx + a
y_pred = (lr.coef_*x)+lr.intercept_
print('predicted:', y_pred.T[0], sep='\n')
```

```
predicted:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
predicted:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

Visualize linear regression

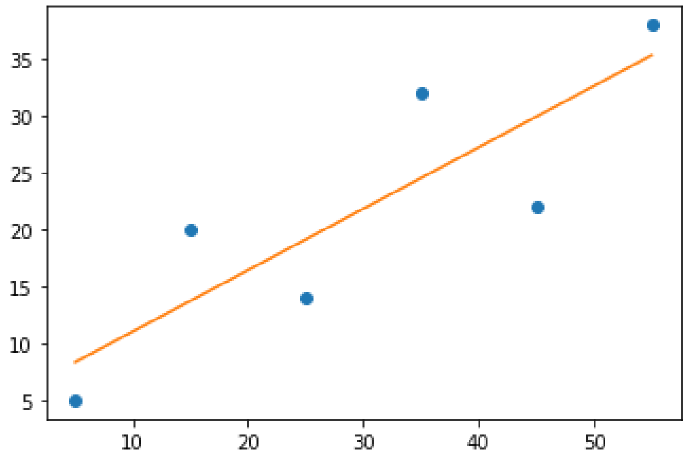
```python
# regression graph
import matplotlib.pyplot as plt
import math

plt.plot(x,y,'o',color='black')
plt.plot(x, y_pred, color='blue', linewidth=2)
plt.xlabel('X - feature')
plt.ylabel('Y - label')
plt.title('Regression Graph')
plt.show()
```



Using linear regression package from numpy

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([5, 15, 25, 35, 45, 55])
y = np.array([5, 20, 14, 32, 22, 38])

# b = slope, a = intercept
b, a = np.polyfit(x, y, 1)
print('slope =',b, 'intercept =',a)
y_pred = (b*x)+a
print('predicted:', y_pred, sep='\n')

# plot
plt.plot(x, y, 'o')
# add line of best fit
plt.plot(x, (b*x)+a)
plt.show()
```

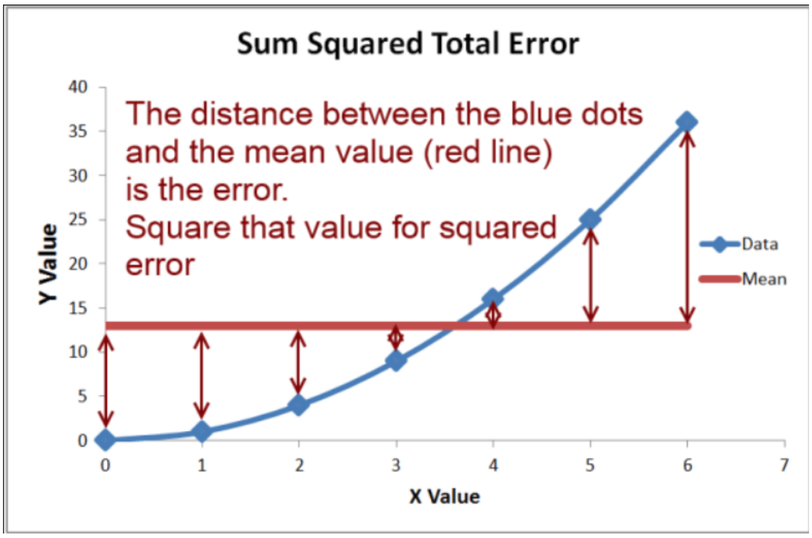
```
slope = 0.54 intercept = 5.633333333333347
predicted:
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```




```
y_pred = (b*x)+a
print(y_pred)
```

```
[ 8.33333333 13.73333333 19.13333333 24.53333333 29.93333333 35.33333333]
```

R-square error – coefficient of determination



$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

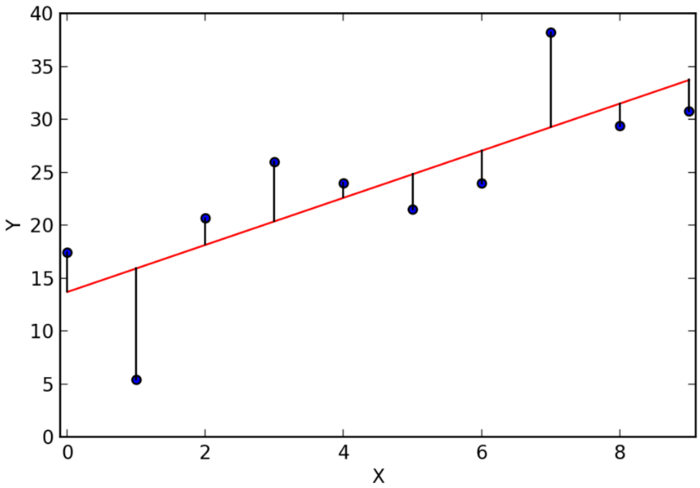
```
# r_sq should close to 1

sum_y_y_pred_2 = np.sum(np.power(y-y_pred, 2))
sum_y_y_bar_2 = np.sum(np.power(y-np.average(y), 2))

r_sq = 1 - (sum_y_y_pred_2 / sum_y_y_bar_2)
print('coefficient of determination:', r_sq)
```

```
coefficient of determination: 0.7158756137479542
```

```
# get results - R_2 (R square) score
# r_sq should close to 1
r_sq = lr.score(x, y)
print('coefficient of determination:', r_sq)
```

```
coefficient of determination: 0.7158756137479542
```

Mean Square Error (MSE)



$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$$

```
N = y.shape[0]
MSE = (1/N) * np.sum(np.power((y-y_pred),2))
print(MSE)
```

```
33.755555555555546
```

```
from sklearn.metrics import mean_squared_error

print(mean_squared_error(y, y_pred))
```

```
33.75555555555555
```

```
# MSE using Numpy module

import numpy as np

MSE = np.square(np.subtract(y, y_pred)).mean()
print(MSE)
```

```
33.75555555555555
```

Boston housing prices prediction using linear regression

Loading data

```
[ ]  from sklearn import datasets ## imports datasets from scikit-learn

     data = datasets.load_boston() ## loads Boston dataset from datasets library
```

```
[ ]  data.keys()
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
[ ]  data.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

```
print(data.data.shape)
data.data
```

```
(506, 13)
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

Preparing data

```python
import pandas as pd

boston = pd.DataFrame(data.data, columns = data.feature_names)
print(boston.shape)
boston.head()
```

```
(506, 13)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Data description

```python
print(data.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

```
print(data.target.shape)
data.target
```

```
(506,)
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
```

```
boston['PRICE'] = data.target

boston.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Data preprocessing

Checking missing values in the data

```
boston.isnull()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATI
0	False	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	...	...	.
501	False	False	False	False	False	False	False	False	False	False	Fals
502	False	False	False	False	False	False	False	False	False	False	Fals

```
1 boston.isnull().sum()
```

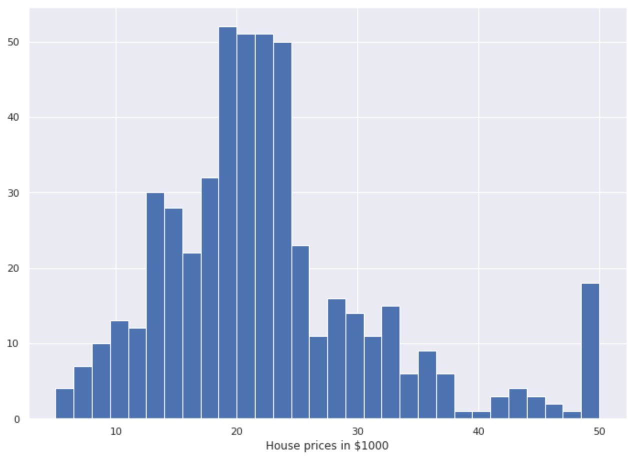
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
```

```
1 boston.describe()
```

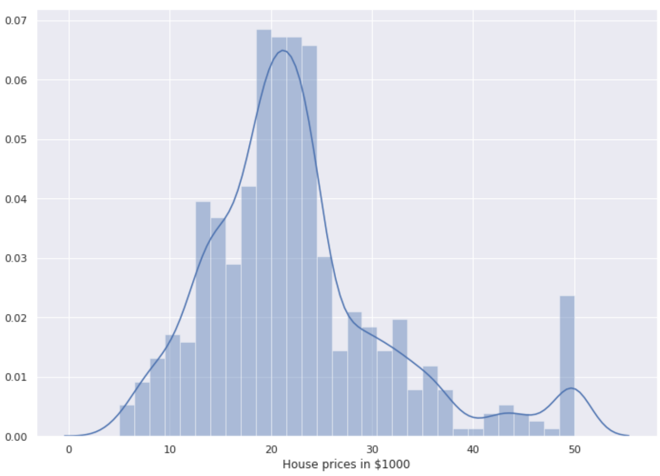
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000

Exploratory data analysis

```
import seaborn as sns

sns.set(rc={'figure.figsize':(11.7,8.27)})
plt.hist(boston['PRICE'], bins=30)
plt.xlabel("House prices in $1000")
plt.show()
```




```
import seaborn as sns

sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(boston['PRICE'], bins=30)
plt.xlabel("House prices in $1000")
plt.show()
```



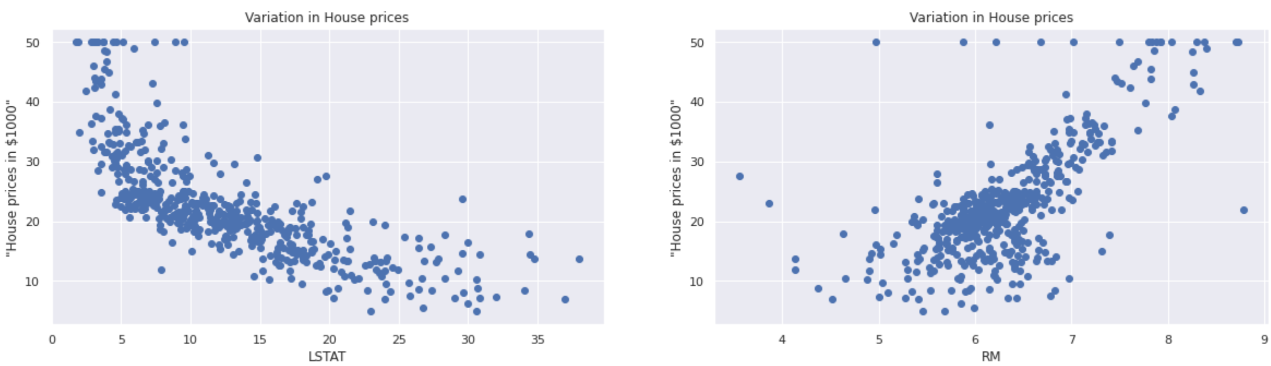
Create a correlation matrix

```
bos = pd.DataFrame(data.data, columns = data.feature_names)
correlation_matrix = bos.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

Notice

- By looking at the correlation matrix we can see that RM has a strong positive correlation with PRICE (0.7) where as LSTAT has a high negative correlation with PRICE (-0.74).
- An important point in selecting features for a linear regression model is to check for multicolinearity. The features RAD, TAX have a correlation of 0.91. These feature pairs are strongly correlated to each other. This can affect the model. Same goes for the features DIS and AGE which have a correlation of -0.75.

```python
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = boston['PRICE']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```



Since you saw that 'RM' shows positive correlation with the House Prices we will use this variable.

```python
X_rooms = boston.RM
y_price = boston.PRICE

X_rooms = np.array(X_rooms).reshape(-1,1)
y_price = np.array(y_price).reshape(-1,1)

print(X_rooms.shape)
print(y_price.shape)
```

```
(506, 1)
(506, 1)
```

Splitting the data into training and test sets

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X_rooms, y_price, test_size = 0.2, random_state=5)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(404, 1)
(102, 1)
(404, 1)
(102, 1)
```

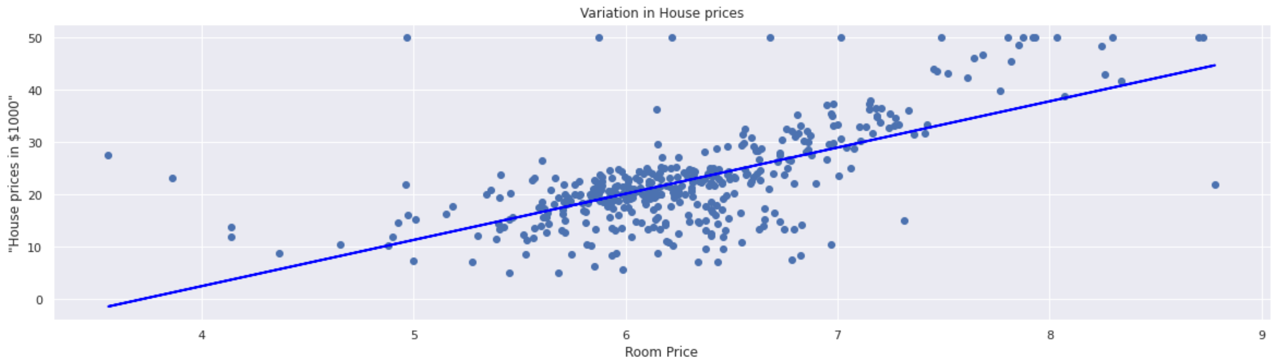
Creating linear model and predicting

Model evaluation for training set

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, Y_train)

# model evaluation for training set
y_train_pred = lr.predict(X_train)
rmse = (np.sqrt(mean_squared_error(Y_train, y_train_pred)))
r2 = round(lr.score(X_train, Y_train),2)

print("The model performance for test set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

```
The model performance for test set
--------------------------------------
Root Mean Squared Error: 6.972277149440585
R^2: 0.43
```

```
# evaluate on test set
plt.figure(figsize=(20, 5))
plt.scatter(X_train, Y_train, marker='o')
plt.plot(X_train, y_train_pred, color='blue', linewidth=2)
plt.title("Variation in House prices")
plt.xlabel('Room Price')
plt.ylabel('"House prices in $1000"')
plt.show()
```

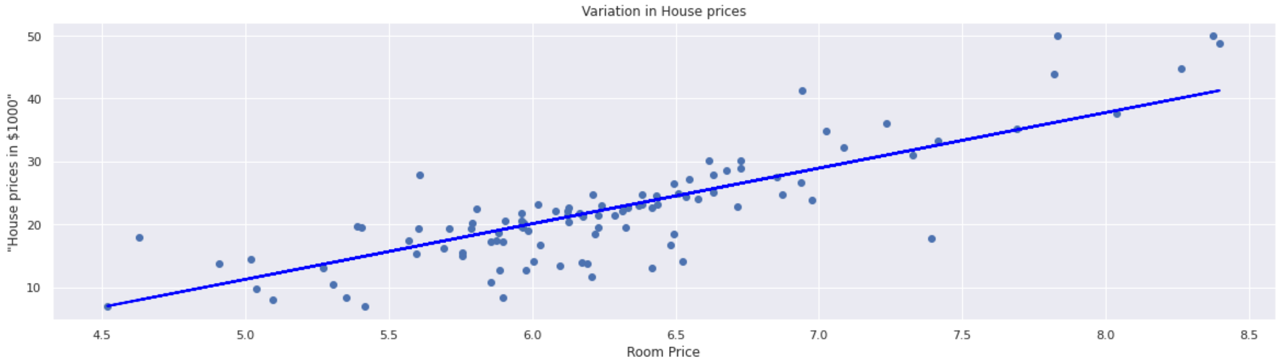


Model evaluation for test set

```python
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, Y_train)

# model evaluation for test set
y_pred = lr.predict(X_test)
rmse = (np.sqrt(mean_squared_error(Y_test, y_pred)))
r2 = round(lr.score(X_test, Y_test),2)

print("The model performance for test set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
print("\n")
```

```
The model performance for test set
--------------------------------------
Root Mean Squared Error: 4.895963186952216
R^2: 0.69
```

```python
# evaluate on test set
plt.figure(figsize=(20, 5))
plt.scatter(X_test, Y_test, marker='o')
plt.plot(X_test, y_pred, color='blue', linewidth=2)
plt.title("Variation in House prices")
plt.xlabel('Room Price')
plt.ylabel('"House prices in $1000"')
plt.show()
```



Linear regression for Boston dataset

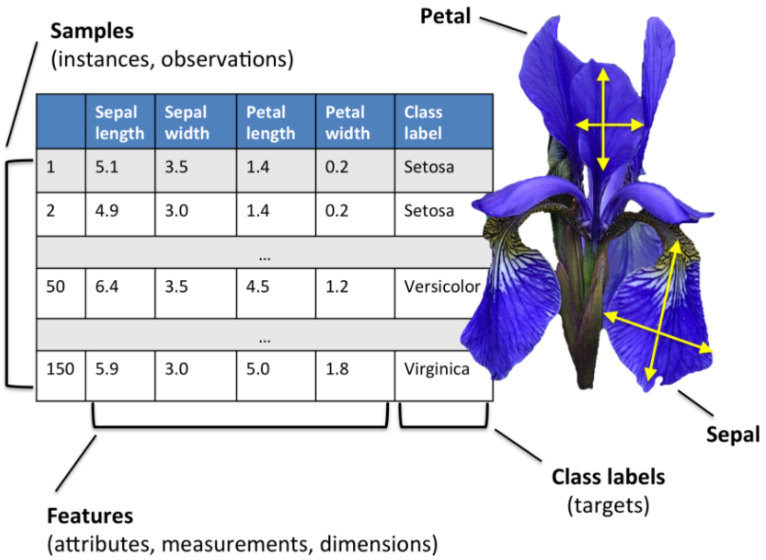
```
X = boston.drop('PRICE', axis = 1)
y = boston['PRICE']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)

lr_all = LinearRegression()
lr_all.fit(X_train, y_train)

# model evaluation for training set

y_train_predict = lr_all.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = round(lr_all.score(X_train, y_train),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 4.6520331848801675
R2 score is 0.75
```

Classification problem

Decision tree

- ใช้ตัวอย่างของดอกไม้ Iris dataset



- Loading iris dataset

```
from sklearn.datasets import load_iris
from sklearn import tree

#x, y = load_iris(return_X_y=True)
iris = load_iris()
```

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
```

- แปลงข้อมูลให้อยู่ในรูปแบบของ pandas DataFrame

```
1 import pandas as pd
2
3 iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)
4 iris_data
```

	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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9

- แสดงประเภทของดอกไม้

```
1 iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

- สร้างโมเดลของ Decision tree

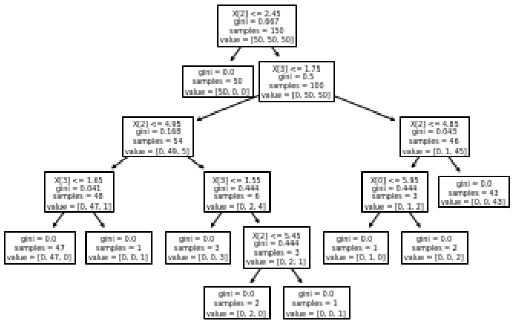
```
1 X, y = load_iris(return_X_y=True)
2
3 clf = tree.DecisionTreeClassifier()
4 clf.fit(X,y)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

- ดูโครงสร้างของต้นไม้

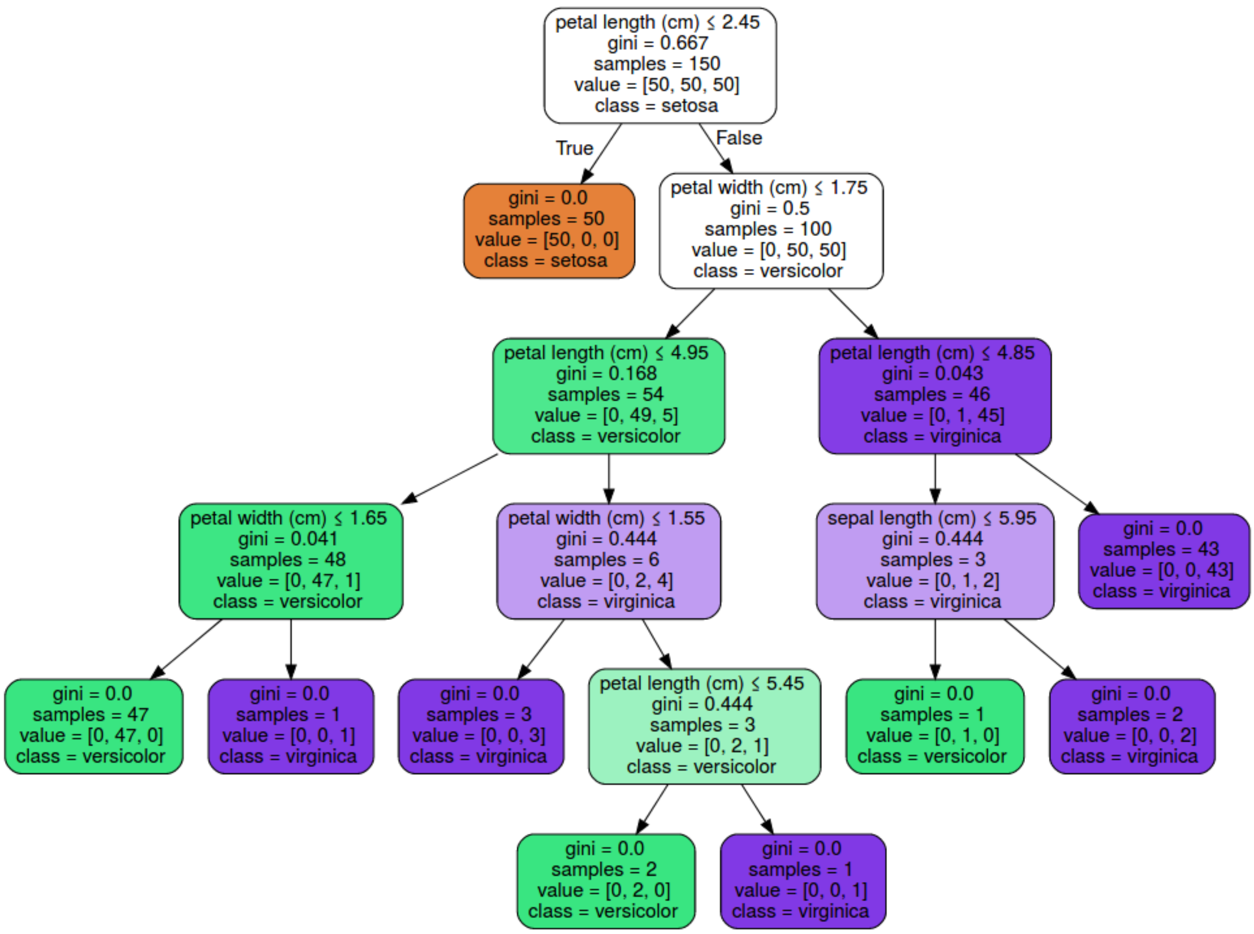
```
tree.plot_tree(clf)
```

```
[Text(167.4, 199.32, 'X[2] <= 2.45\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 50]')
 Text(141.64615384615385, 163.07999999999998, 'gini = 0.0\nsamples = 50\nvalue = [50, 0
 Text(193.15384615384616, 163.07999999999998, 'X[3] <= 1.75\ngini = 0.5\nsamples = 100\
 Text(103.01538461538462, 126.83999999999999, 'X[2] <= 4.95\ngini = 0.168\nsamples = 54
 Text(51.50769230769231, 90.6, 'X[3] <= 1.65\ngini = 0.041\nsamples = 48\nvalue = [0, 4
 Text(25.753846153846155, 54.359999999999985, 'gini = 0.0\nsamples = 47\nvalue = [0, 47
 Text(77.26153846153846, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1
 Text(154.52307692307693, 90.6, 'X[3] <= 1.55\ngini = 0.444\nsamples = 6\nvalue = [0, 2
 Text(128.76923076923077, 54.359999999999985, 'gini = 0.0\nsamples = 3\nvalue = [0, 0,
 Text(180.27692307692308, 54.359999999999985, 'X[2] <= 5.45\ngini = 0.444\nsamples = 3\
 Text(154.52307692307693, 18.119999999999976, 'gini = 0.0\nsamples = 2\nvalue = [0, 2,
 Text(206.03076923076924, 18.119999999999976, 'gini = 0.0\nsamples = 1\nvalue = [0, 0,
 Text(283.2923076923077, 126.83999999999999, 'X[2] <= 4.85\ngini = 0.043\nsamples = 46\
 Text(257.53846153846155, 90.6, 'X[0] <= 5.95\ngini = 0.444\nsamples = 3\nvalue = [0, 1
 Text(231.7846153846154, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0
 Text(283.2923076923077, 54.359999999999985, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2
 Text(309.04615384615386, 90.6, 'gini = 0.0\nsamples = 43\nvalue = [0, 0, 43]')]
```



- visualize ต้นไม้

```
import graphviz

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
#graph.render('iris') # generate pdf file
graph
```



- predict ข้อมูล และแสดงผลลัพธ์

```
cols = [iris.feature_names, 'predicted']
```

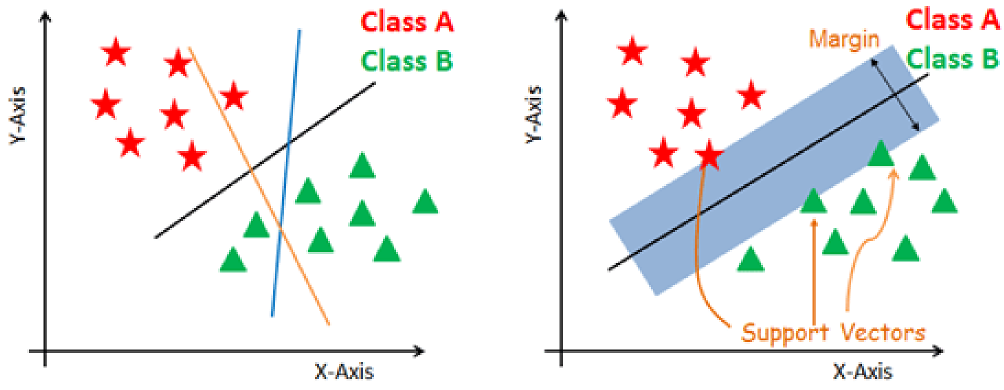
```
import numpy as np
import random

i = random.randrange(0,150)
print('index number', i)
out = clf.predict([X[i]])
data_i = np.append(X[i], y[i])
data_i = np.append(data_i, out[0])
df = pd.DataFrame([data_i], columns=iris.feature_names+['actual', 'predicted'])
print('output =', out[0], iris.target_names[out])
df
```

```
index number 35
output = 0 ['setosa']
```

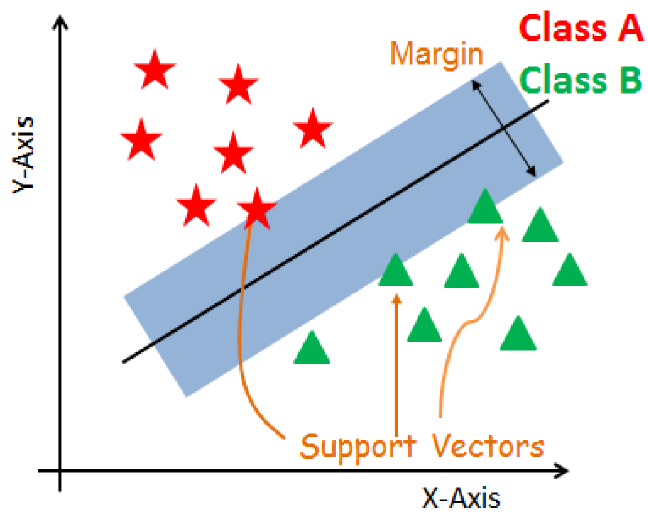
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	actual	predicted
0	5.0	3.2	1.2	0.2	0.0	0.0

Support vector machine (SVM)

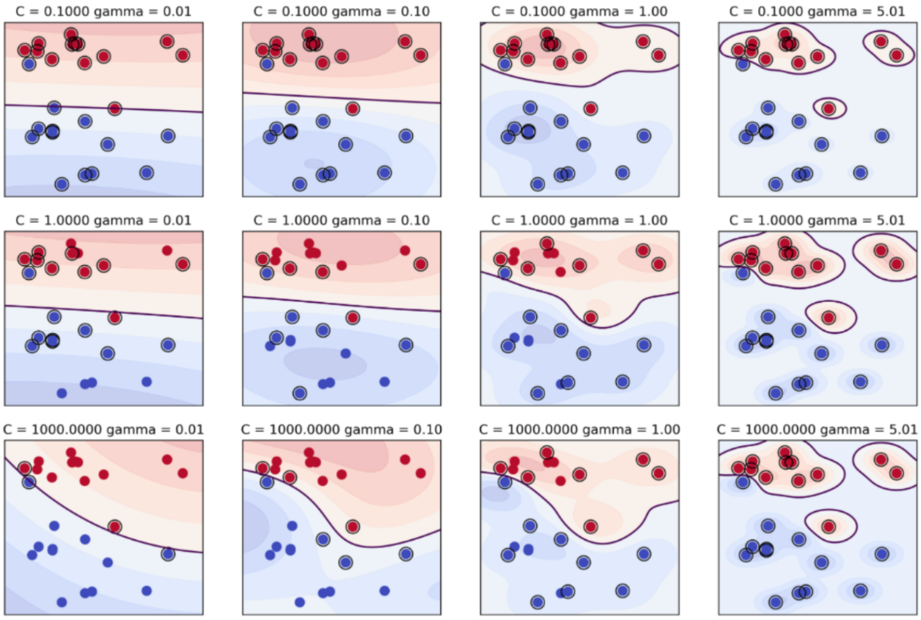
Hyperplane



Linear Kernel



Radial basis function (RBF) Kernel



Loading data

```
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
cancer = datasets.load_breast_cancer()
```

- Exploring data

```
# print the names of the 13 features
print('Features: ', cancer.feature_names)

# print the label type of cancer('malignant' 'benign')
print('Labels: ', cancer.target_names)
```

```
Features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
```

```
# print data(feature)shape
print(cancer.data.shape)
```

```
(569, 30)
```

```
# print the cancer data features (top 5 records)
print(cancer.data[0:5])
```

```
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
```

```
# print the cancer labels (0:malignant, 1:benign)
print(cancer.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1
```

-

Splitting data

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target,
    test_size=0.3,random_state=109) # 70% training and 30% test
```

Generating model

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Predicting data

```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Report

- Accuracy score

```
from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)
```

```
0.9649122807017544
```

- Classification report

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.97      0.95        63
           1       0.98      0.96      0.97       108

    accuracy                           0.96       171
   macro avg       0.96      0.97      0.96       171
weighted avg       0.97      0.96      0.97       171
```

- Confusion matrix

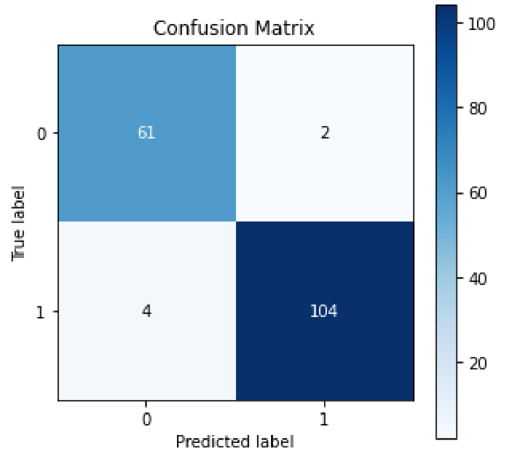
```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(y_test, y_pred))
```

```
[[ 61   2]
 [  4 104]]
```

```
[ ] !pip install -q scikit-plot
```

```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    y_test,
    y_pred,
    figsize=(5,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7e4ce5c2e8>
```



Searching best parameter using grid search

```
[ ]
from sklearn.model_selection import GridSearchCV

# Set the parameters by cross-validation
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

gs_clf_svm = GridSearchCV(clf, tuned_parameters, n_jobs=-1)
gs_clf_svm = gs_clf_svm.fit(X_train, y_train)
```

```
[ ]
print(gs_clf_svm.best_score_)
print(gs_clf_svm.best_params_)
```

```
0.9497784810126582
{'C': 100, 'kernel': 'linear'}
```

```
[ ]
# predict with the best parameters
gs_predicted = gs_clf_svm.predict(X_test)
accuracy_score(y_test, gs_predicted)
```

```
0.9707602339181286
```

- comparing SVM results

```
print('SVM without tuning parameter', accuracy_score(y_test, y_pred))
print('SVM with tuning parameter', accuracy_score(y_test, gs_predicted))
```

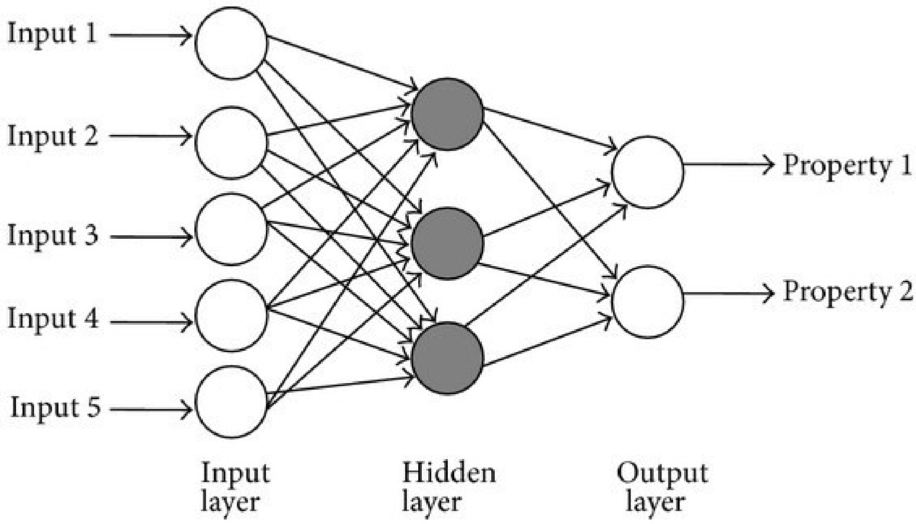
```
SVM without tuning parameter 0.9649122807017544
SVM with tuning parameter 0.9707602339181286
```

- Training the new model using best parameters

```
# Linear Kernel
gs_clf = svm.SVC(kernel='linear', C=100)

#Train the model using the training sets
gs_clf.fit(X_train, y_train)

y_pred_gs = gs_clf.predict(X_test)
accuracy_score(y_test, y_pred_gs)
```

```
0.9707602339181286
```

Multilayer perceptron - MLP



- loading data

```python
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
cancer = datasets.load_breast_cancer()
```

```python
print(cancer.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1
```

- splitting data

```python
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data,
                                                    cancer.target,
                                                    test_size=0.3,
                                                    random_state=109)
```

parameter ของ MLP

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(30, 30, 30), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

Training MLP model

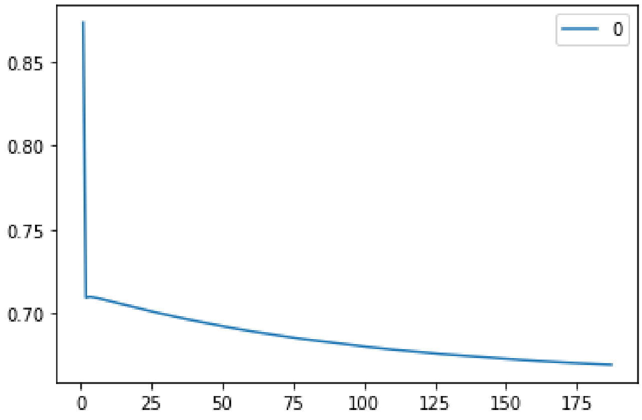
```python
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(10), random_state=1, verbose=True)
clf.fit(X_train, y_train)

neural_output = clf.predict(X_test)
print('sgd')
print(accuracy_score(y_test, neural_output))
```

```
Iteration 1, loss = inf
Iteration 2, loss = 0.87378527
Iteration 3, loss = 0.70910193
Iteration 4, loss = 0.70969458
Iteration 5, loss = 0.70960079
Iteration 6, loss = 0.70931514
Iteration 7, loss = 0.70897305
Iteration 8, loss = 0.70855273
Iteration 9, loss = 0.70815045
Iteration 10, loss = 0.70775779
Iteration 11, loss = 0.70733859
```

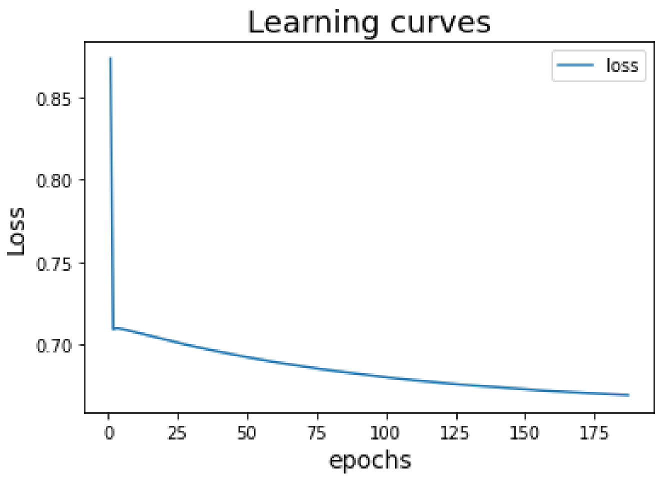
Learning curve

```
import pandas as pd

pd.DataFrame(clf.loss_curve_).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58b0c085f8>
```




```
import matplotlib.pyplot as plt

plt.title('Learning curves', fontsize = 18)
plt.legend()
#plt.grid()
plt.plot(clf.loss_curve_, label = 'loss')
plt.ylabel('Loss', fontsize = 14)
plt.xlabel('epochs', fontsize = 14)
plt.show()
```



ทดสอบการปรับค่าพารามิเตอร์

Hidden layer และ Solver

```
for i in [10, 200, 300]:
  clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(i), random_state=1)
  clf.fit(X_train, y_train)

  neural_output = clf.predict(X_test)
  print('\nsgd,', 'hidden', i)
  print(accuracy_score(y_test, neural_output))
```

```
sgd, hidden 10
0.631578947368421

sgd, hidden 200
0.9122807017543859

sgd, hidden 300
0.9415204678362573
```

```
for i in [10, 200, 300]:
  clf = MLPClassifier(solver='adam', hidden_layer_sizes=(i), random_state=1)
  clf.fit(X_train, y_train)

  neural_output = clf.predict(X_test)
  print('\nsgd,', 'hidden', i)
  print(accuracy_score(y_test, neural_output))
```

```
sgd, hidden 10
0.631578947368421

sgd, hidden 200
0.9824561403508771

sgd, hidden 300
0.9649122807017544
```

แสดงผล predict ด้วยค่าความน่าจะเป็น (Probability)

```
proba_output = clf.predict_proba(X_test)
print(proba_output[0:5])
```

```
[[1.08722789e-01 8.91277211e-01]
 [1.99169405e-02 9.80083060e-01]
 [9.99999997e-01 3.15383864e-09]
 [1.00000000e+00 2.82881002e-12]
 [4.88133441e-01 5.11866559e-01]]
```

report

```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test,neural_output))
print('Confusion matrix')
print(confusion_matrix(y_test,neural_output))
```

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95        63
           1       0.96      0.98      0.97       108

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

Confusion matrix
[[ 59   4]
 [  2 106]]
```

สร้างโมเดลของ MLP ด้วย Keras library

```
from keras.models import Sequential
from keras.utils import np_utils
from keras.layers.core import Dense, Activation, Dropout

import pandas as pd
import numpy as np
```

- เปลี่ยน label ให้อยู่ในรูป class matrix

```
# convert list of labels to binary class matrix
y_train_cat = np_utils.to_categorical(y_train)
y_test_cat = np_utils.to_categorical(y_test)
```

```
print(y_train[0:10])
print(y_train_cat[0:10])
```

```
[0 1 1 1 1 1 1 0 0 1]
[[1. 0.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]]
```

- Normalize ข้อมูล

```
# pre-processing: divide by max and substract mean
scale = np.max(X_train)
X_train_scale = X_train / scale
X_test_scale = X_test / scale
```

```
[ ]
print(X_train[0])
print(X_train_scale[0])
```

```
[1.422e+01 2.312e+01 9.437e+01 6.099e+02 1.075e-01 2.413e-01 1.981
 6.618e-02 2.384e-01 7.542e-02 2.860e-01 2.110e+00 2.112e+00 3.172
 7.970e-03 1.354e-01 1.166e-01 1.666e-02 5.113e-02 1.172e-02 1.574
 3.718e+01 1.064e+02 7.624e+02 1.533e-01 9.327e-01 8.488e-01 1.772
 5.166e-01 1.446e-01]
[3.34273625e-03 5.43488481e-03 2.21838270e-02 1.43370945e-01
 2.52703338e-05 5.67230842e-05 4.65679361e-05 1.55571227e-05
 5.60413728e-05 1.77291961e-05 6.72308416e-05 4.96003761e-04
 4.96473907e-04 7.45651152e-03 1.87353079e-06 3.18288669e-05
```

- แสดงจำนวนของ feature และ class

```
input_dim = X_train.shape[1]
nb_classes = y_train.max()+1

print('feature:', input_dim)
print('class:', nb_classes)
```

```
feature: 30
class: 2
```

- สร้าง function ของ mlp

```
def mlp_clf(input_dim, nb_classes):
  model = Sequential()
  model.add(Dense(128, input_dim=input_dim))
  model.add(Activation('relu'))
  model.add(Dropout(0.15))
  model.add(Dense(128))
  model.add(Activation('relu'))
  model.add(Dropout(0.15))
  model.add(Dense(nb_classes))
  model.add(Activation('softmax'))

  # we'll use categorical xent for the loss, and RMSprop as the optimizer
  #model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
  model.compile(loss='binary_crossentropy', optimizer='rmsprop')

  return model
```

Training mlp

```
mlp_model = mlp_clf(input_dim, nb_classes)

print('Training...')
mlp_model.fit(X_train_scale, y_train_cat, epochs=10,
              batch_size=16, validation_split=0.1, verbose=1)

print('Generating test predictions...')
preds = mlp_model.predict_classes(X_test_scale, verbose=1)
```

```
Training...
Train on 358 samples, validate on 40 samples
Epoch 1/10
358/358 [==============================] - 0s 492us/step - loss: 0.6606 - val_loss: 0.6161
Epoch 2/10
358/358 [==============================] - 0s 120us/step - loss: 0.5648 - val_loss: 0.4832
Epoch 3/10
358/358 [==============================] - 0s 108us/step - loss: 0.4585 - val_loss: 0.3690
Epoch 4/10
358/358 [==============================] - 0s 114us/step - loss: 0.3711 - val_loss: 0.3443
Epoch 5/10
358/358 [==============================] - 0s 113us/step - loss: 0.3110 - val_loss: 0.2610
Epoch 6/10
358/358 [==============================] - 0s 105us/step - loss: 0.2849 - val_loss: 0.2359
Epoch 7/10
```

- report

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, preds))
```

```
0.9473684210526315
```

```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test,preds))
print('Confusion matrix')
print(confusion_matrix(y_test,preds))
```

```
              precision    recall  f1-score   support

           0       0.95      0.90      0.93        63
           1       0.95      0.97      0.96       108

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171

Confusion matrix
[[ 57   6]
 [  3 105]]
```

Feature engineering and Evaluation method

Feature selection

Pearson correlation

- loading data

```python
from sklearn.datasets import load_breast_cancer
import pandas as pd

cancer_data = load_breast_cancer()
df = pd.DataFrame(cancer_data.data, columns = cancer_data.feature_names)
df['diagnosis'] = cancer_data.target

X = df.drop('diagnosis', 1)
y = df['diagnosis']
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	me symmet
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.24
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.18
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.20

```python
# print label
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: diagnosis, Length: 569, dtype: int64
```

- plot correlation

```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15,15))
cor = df.corr(method ='pearson')
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```

- แสดงค่า correlation

```python
#Correlation with output variable
cor_target = abs(cor['diagnosis']) #Selecting highly correlated features
relevant_features = cor_target[cor_target>0.5].sort_values(ascending=False)
relevant_features[1:-1]
```

```
worst concave points    0.793566
worst perimeter         0.782914
mean concave points     0.776614
worst radius            0.776454
mean perimeter          0.742636
worst area              0.733825
mean radius             0.730029
mean area               0.708984
mean concavity          0.696360
worst concavity         0.659610
mean compactness        0.596534
worst compactness       0.590998
radius error            0.567134
perimeter error         0.556141
Name: diagnosis, dtype: float64
```

- เลือกข้อมูลที่มี correlation สูงที่สุด

```python
# select correlation data
cor_data = pd.DataFrame()

for i in range(1, relevant_features.index[1:-1].shape[0]+1):
  tmp_data = df[relevant_features.index[i:i+1][0]]
  column_name = relevant_features.index[i:i+1][0]
  cor_data[column_name] = tmp_data

cor_data
```

	worst concave points	worst perimeter	mean concave points	worst radius	mean perimeter	worst area	mean radius	mean area	me concav
0	0.2654	184.60	0.14710	25.380	122.80	2019.0	17.99	1001.0	0.30
1	0.1860	158.80	0.07017	24.990	132.90	1956.0	20.57	1326.0	0.08
2	0.2430	152.50	0.12790	23.570	130.00	1709.0	19.69	1203.0	0.19
3	0.2575	98.87	0.10520	14.910	77.58	567.7	11.42	386.1	0.24

- Normalization data

```python
from sklearn.preprocessing import Normalizer

transformer = Normalizer().fit(cor_data)
cor_data_norm = transformer.transform(cor_data)

cor_data_norm = pd.DataFrame(data=cor_data_norm, columns=cor_data.columns)
cor_data_norm
```

	worst concave points	worst perimeter	mean concave points	worst radius	mean perimeter	worst area	mean radius	mean area	mean concavity	wor concavi
0	0.000117	0.081514	0.000065	0.011207	0.054225	0.891535	0.007944	0.442014	0.000133	0.000
1	0.000078	0.066937	0.000030	0.010534	0.056020	0.824491	0.008671	0.558934	0.000037	0.000
2	0.000116	0.072627	0.000061	0.011225	0.061912	0.813901	0.009377	0.572921	0.000094	0.000
3	0.000369	0.141602	0.000151	0.021354	0.111110	0.813063	0.016356	0.552974	0.000346	0.000
4	0.000079	0.074220	0.000051	0.010992	0.065881	0.768046	0.009894	0.632480	0.000097	0.000
...	...	...	...	...	...	...	...	...	...	
564	0.000088	0.065940	0.000055	0.010103	0.056373	0.804701	0.008559	0.587150	0.000097	0.000
565	0.000076	0.072045	0.000046	0.011011	0.060982	0.804576	0.009357	0.586118	0.000067	0.000

- splitting data

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(cor_data_norm, y,
                                                    test_size=0.2, random_state=75)

print('x_train shape is: ', x_train.shape)
print('y_train shape is: ', y_train.shape)
print('x_test shape is: ', x_test.shape)
print('y_test shape is: ', y_test.shape)
```

```
x_train shape is:  (455, 14)
y_train shape is:  (455,)
x_test shape is:  (114, 14)
y_test shape is:  (114,)
```

- สร้างโมเดลด้วย KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

clf = KNeighborsClassifier(n_neighbors=7)
clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)
accuracy_score(y_test, y_pred)
```

```
0.9385964912280702
```

Recursive feature elimination (RFE)

- สร้าง RFE ด้วยวิธี SVM

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split

X, y = make_friedman1(n_features=10, random_state=0)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=75)

svm_clf = SVR(kernel="linear")
rfe_selector = RFE(svm_clf, n_features_to_select=5, step=1)
rfe_selector.fit(x_train, y_train)

print(selector.support_)
print(selector.ranking_)
```

```
[ True  True False  True  True False False  True False False]
[1 1 6 1 1 5 3 1 4 2]
```

สร้าง RFE data

```
# create RFE data
x_train_rfe = rfe_selector.transform(x_train)
x_test_rfe = rfe_selector.transform(x_test)

print(x_train_rfe.shape)
print(x_test_rfe.shape)
```

```
(80, 5)
(20, 5)
```

- สร้าง RFE ด้วยวิธี linear

```
from sklearn.datasets import make_friedman1
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

X, y = make_friedman1(n_features=10, random_state=0)
x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=75)

lr_clf = LinearRegression() #Initializing RFE model
rfe_selector = RFE(lr_clf, n_features_to_select=5, step=1)
rfe_selector.fit(x_train, y_train)

print(selector.support_)
print(selector.ranking_)
```

```
[ True  True False  True  True False False  True False False]
[1 1 6 1 1 5 3 1 4 2]
```

```
# create RFE data
x_train_rfe = rfe_selector.transform(x_train)
x_test_rfe = rfe_selector.transform(x_test)

print(x_train_rfe.shape)
print(x_test_rfe.shape)
```

```
(80, 5)
(20, 5)
```

Principal component analysis (pca)

```
from sklearn.datasets import make_friedman1
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

X, y = make_friedman1(n_features=10, random_state=0)
x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=75)

pca_clf = PCA()
x_train_pca = pca_clf.fit_transform(x_train)
x_test_pca = pca_clf.fit_transform(x_test)

print(x_train_pca.shape)
print(x_test_pca.shape)

```

```
(80, 10)
(20, 10)
```

เปรียบเทียบข้อมูล

```
print(x_train[0])
print(x_train_pca[0])
```

```
[0.86385561 0.11753186 0.51737911 0.13206811 0.71685968 0.3960597
 0.56542131 0.18327984 0.14484776 0.48805628]
[-0.53920864 -0.21534482 -0.3465056  -0.20667164  0.05223516 -0.08325658
  0.11006185  0.18758579 -0.31364582 -0.10428217]
```

- เลือกจำนวน component ที่เหมาะสม

```
var_ratio = pca_clf.explained_variance_ratio_
print("Explained variance ratio:",var_ratio, sep='\n')
```

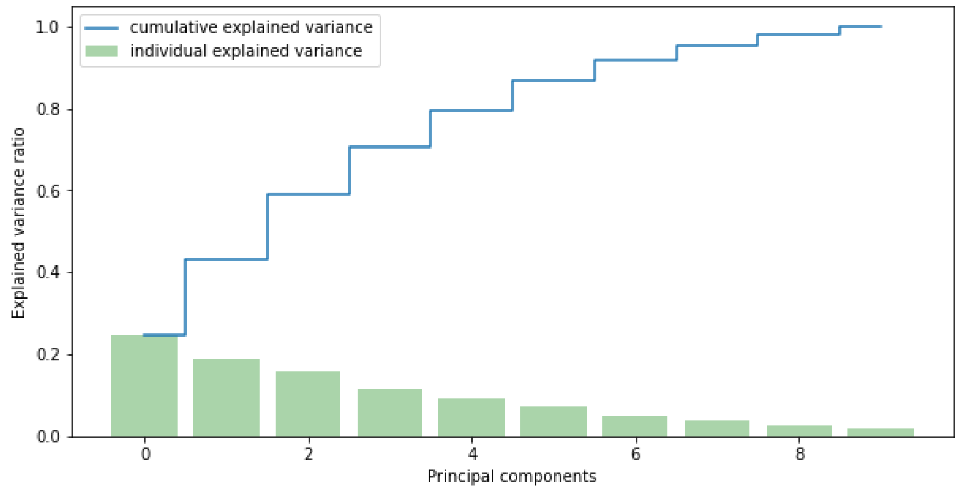
```
Explained variance ratio:
[0.24648355 0.1869188  0.15737862 0.11449547 0.09005619 0.072385
 0.05078349 0.03626017 0.02507092 0.02016778]
```

สร้างฟังก์ชั่นแสดงค่า cumulative variance

```
import matplotlib.pyplot as plt
import numpy as np

def plot_pca_cumulative_variance(var_ratio):
  cum_var_ratio = np.cumsum(var_ratio)
  plt.figure(figsize=(10, 5))
  plt.bar(range(len(var_ratio)),
          var_ratio,
          alpha=0.3333,
          align='center',
          label='individual explained variance',
          color = 'g')
  plt.step(range(len(cum_var_ratio)),
           cum_var_ratio,
           where='mid',
           label='cumulative explained variance')
  plt.ylabel('Explained variance ratio')
  plt.xlabel('Principal components')
  plt.legend(loc='best')
  plt.show()

  print("Explained variance ratio:",var_ratio, sep='\n')
  print("Cumulative explained ratio:",cum_var_ratio, sep='\n')
```

พลอตค่า cumulative variance

```python
plot_pca_cumulative_variance(var_ratio)
```

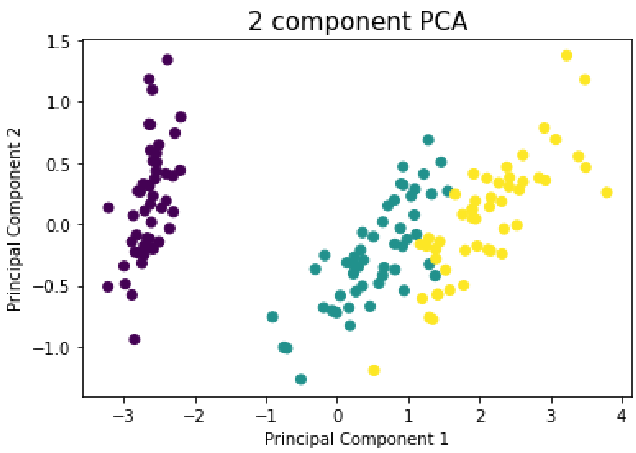



```
Explained variance ratio:
[0.24648355 0.1869188  0.15737862 0.11449547 0.09005619 0.072385
 0.05078349 0.03626017 0.02507092 0.02016778]
Cumulative explained ratio:
[0.24648355 0.43340235 0.59078097 0.70527644 0.79533262 0.86771763
 0.91850112 0.95476129 0.97983222 1.        ]
```

ทดสอบกับ iris dataset

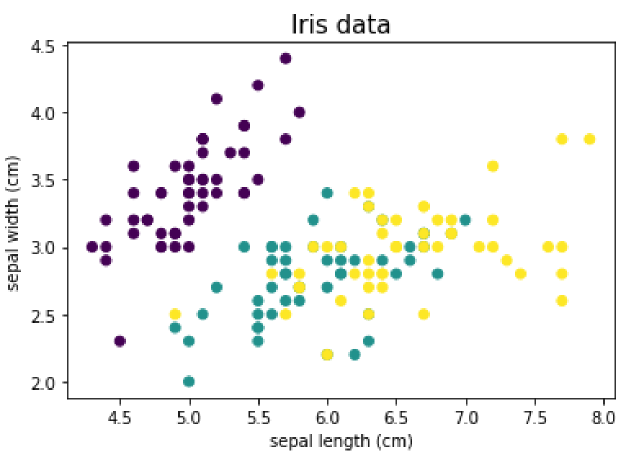
```python
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split


iris = datasets.load_iris()
X = iris.data
y = iris.target

pca_clf = PCA()
x_pca = pca_clf.fit_transform(X)
```

```
import matplotlib.pyplot as plt

plt.scatter(x_pca[:,0], x_pca[:,1], c=iris.target)
plt.title('2 component PCA', fontsize = 15)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



ข้อมูลที่ไม่ผ่าน pca

```
import matplotlib.pyplot as plt

fe = [0,1]
plt.scatter(X[:,fe[0]], X[:,fe[1]], c=iris.target)
plt.title('Iris data', fontsize = 15)
plt.xlabel(iris.feature_names[fe[0]])
plt.ylabel(iris.feature_names[fe[1]])
plt.show()
```



ทดสอบกับชุดข้อมูล MNIST

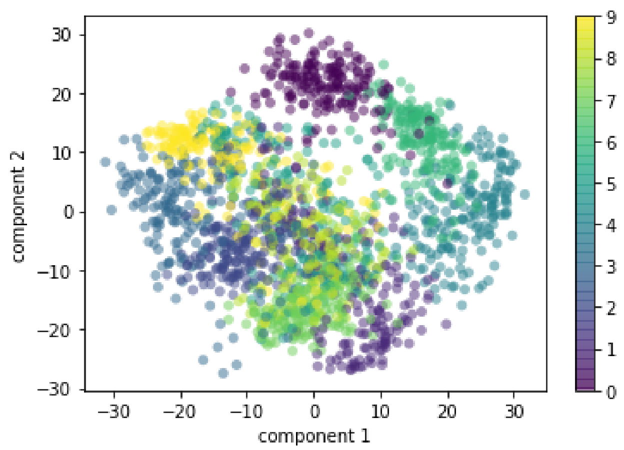
```python
from sklearn.datasets import load_digits

digits = load_digits()
digits.data.shape
```

```
(1797, 64)
```

```python
from sklearn.decomposition import PCA

pca = PCA(2)  # project from 64 to 2 dimensions
projected = pca.fit_transform(digits.data)

print(digits.data.shape)
print(projected.shape)
```

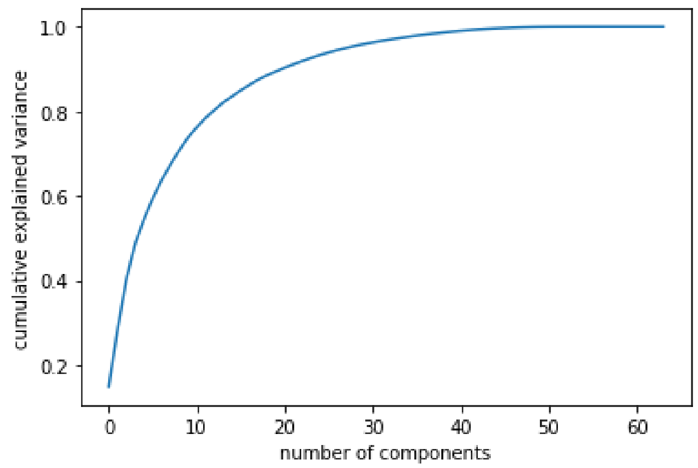
```
(1797, 64)
(1797, 2)
```

```python
import matplotlib.pyplot as plt

plt.scatter(projected[:, 0], projected[:, 1],
            c=digits.target, edgecolor='none', alpha=0.5)
            #cmap=plt.cm.get_cmap('spectral', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar()
plt.show()
```



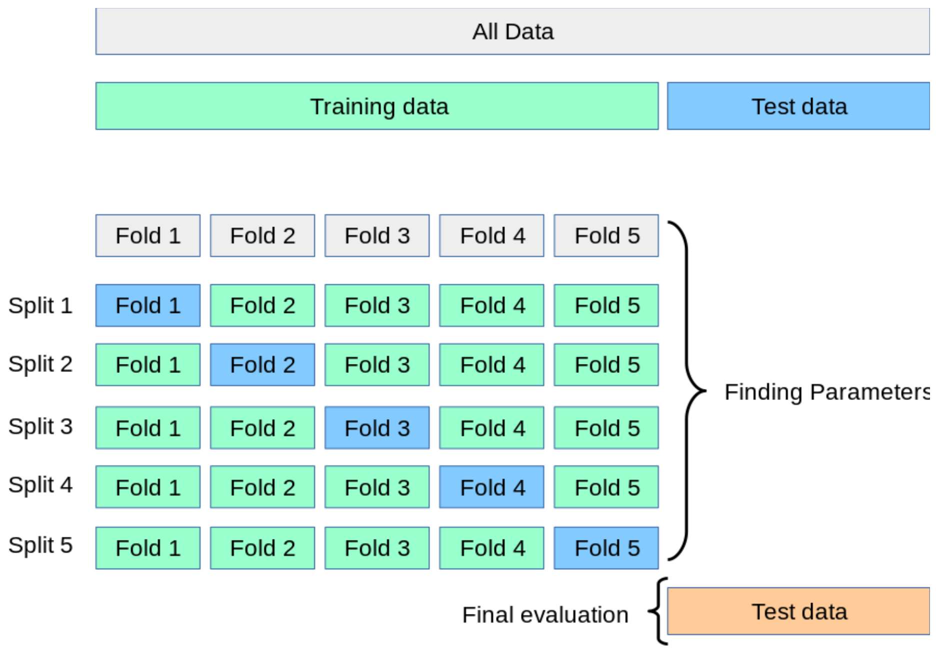
- เลือกจำนวน component ที่เหมาะสม

```
pca = PCA().fit(digits.data)

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```

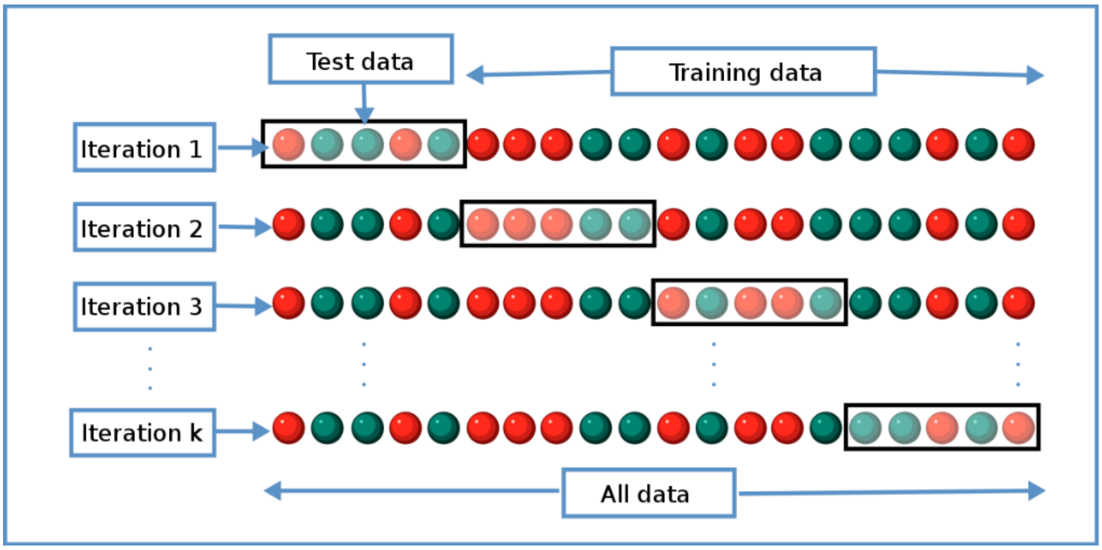


Cross-validation

Method 1



Method 2



- loading data

```
[ ]
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn import svm

X, y = datasets.load_iris(return_X_y=True)
X.shape, y.shape
```

```
((150, 4), (150,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.4,
                                                    random_state=0)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(90, 4) (90,)
(60, 4) (60,)
```

- cross-validation

```
from sklearn.model_selection import cross_val_score

clf = svm.SVC(kernel='linear', C=1)

scores = cross_val_score(clf, X, y, cv=5)
print('score', scores)

# mean score
print('Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))
```

```
score [0.96666667 1.         0.96666667 0.96666667 1.        ]
Accuracy: 0.98 (+/- 0.03)
```

- ShuffleSplit

```
[ ]
from sklearn.model_selection import ShuffleSplit

cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
scores = cross_val_score(clf, X, y, cv=cv)
print('score', scores)

# mean score
print('Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))
```

```
score [0.97777778 0.97777778 1.         0.95555556 1.        ]
Accuracy: 0.98 (+/- 0.03)
```

- ทดสอบกับชุดข้อมูล diabetes

```
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_validate

diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
```

```
[ ]
print(X.shape)
print(X)
```

```
(150, 10)
[[ 0.03807591  0.05068012  0.06169621 ... -0.00259226  0.01990842
  -0.01764613]
 [-0.00188202 -0.04464164 -0.05147406 ... -0.03949338 -0.06832974
  -0.09220405]
 [ 0.08529891  0.05068012  0.04445121 ... -0.00259226  0.00286377
  -0.02593034]
```

label มีลักษณะเป็น discrete

```
print(y.shape)
print(y)
```

```
(150,)
[151.  75. 141. 206. 135.  97. 138.  63. 110. 310. 101.  69. 179. 185.
 118. 171. 166. 144.  97. 168.  68.  49.  68. 245. 184. 202. 137.  85.
 131. 283. 129.  59. 341.  87.  65. 102. 265. 276. 252.  90. 100.  55.
  61.  92. 259.  53. 190. 142.  75. 142. 155. 225.  59. 104. 182. 128.
  52.  37. 170. 170.  61. 144.  52. 128.  71. 163. 150.  97. 160. 178.
  48. 270. 202. 111.  85.  42. 170. 200. 252. 113. 143.  51.  52. 210.
  65. 141.  55. 134.  42. 111.  98. 164.  48.  96.  90. 162. 150. 279.
  92.  83. 128. 102. 302. 198.  95.  53. 134. 144. 232.  81. 104.  59.
 246. 297. 258. 229. 275. 281. 179. 200. 200. 173. 180.  84. 121. 161.
  99. 109. 115. 268. 274. 158. 107.  83. 103. 272.  85. 280. 336. 281.
 118. 317. 235.  60. 174. 259. 178. 128.  96. 126.]
```

ทดสอบด้วย linear model

```
from sklearn import linear_model

lasso = linear_model.Lasso()

# single metric evaluation using cross_validate
cv_results = cross_validate(lasso, X, y, cv=3)
print(cv_results['test_score'])

# mean score
print('Accuracy: %0.2f (+/- %0.2f)' % (cv_results['test_score'].mean(),
                                       cv_results['test_score'].std() * 2))
```

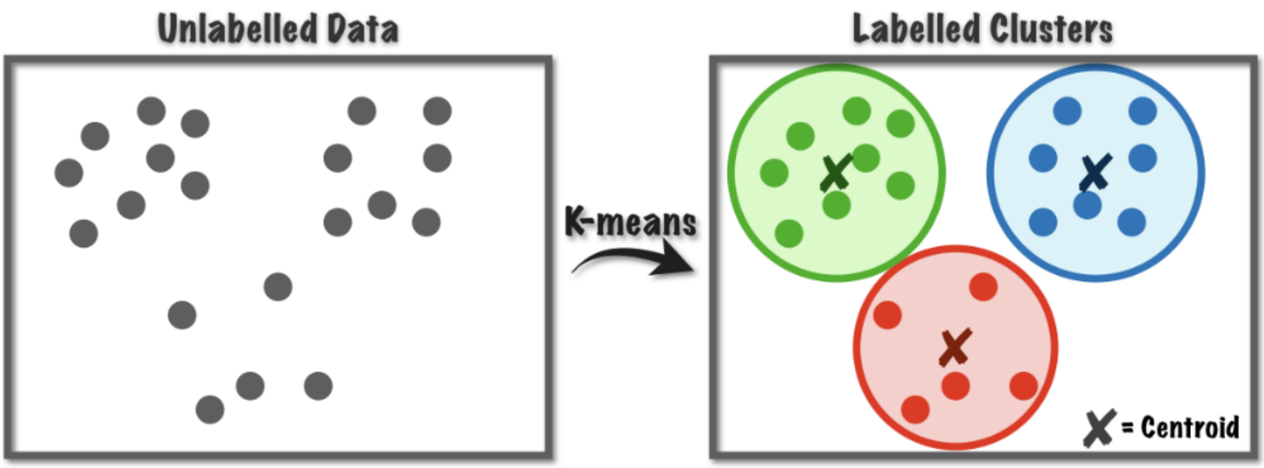
```
[0.33150734 0.08022311 0.03531764]
Accuracy: 0.15 (+/- 0.26)
```

```python
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_validate

cv = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
scores = cross_val_score(lasso, X, y, cv=cv)
print('score', scores)

# mean score
print('Accuracy: %0.2f (+/- %0.2f)' % (scores.mean(), scores.std() * 2))
```

```
score [0.30553672 0.22064577 0.17962466 0.25542505 0.33841705]
Accuracy: 0.26 (+/- 0.11)
```

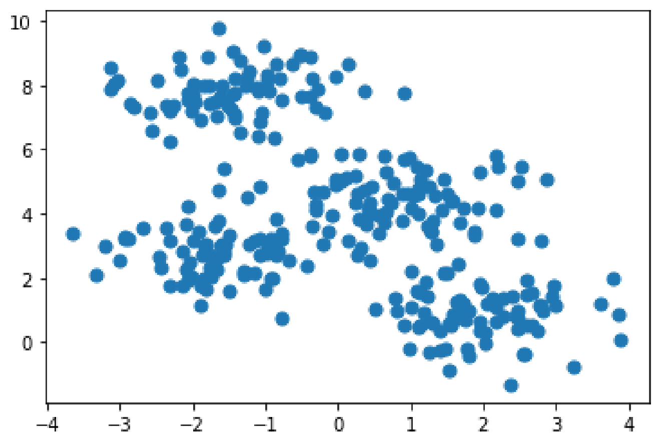
Clustering problem

K-Means clustering



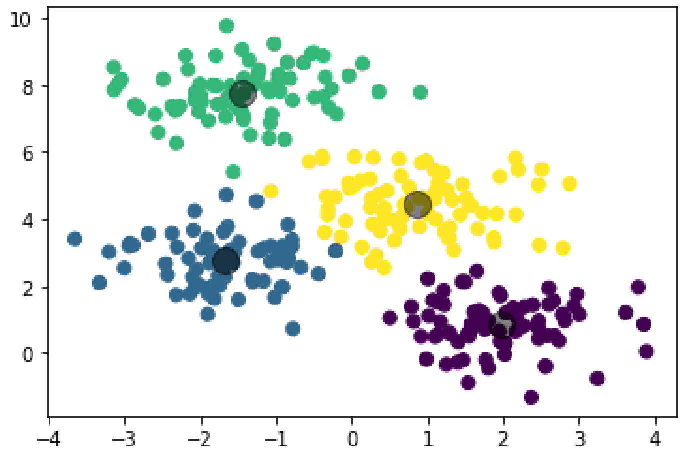
- สร้างข้อมูลเพื่อทดสอบ

```
from sklearn.datasets.samples_generator import make_blobs
import matplotlib.pyplot as plt

X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.80, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);11
plt.show()
```



สร้างโมเดล k-means โดยกำหนดให้มี 4 cluster

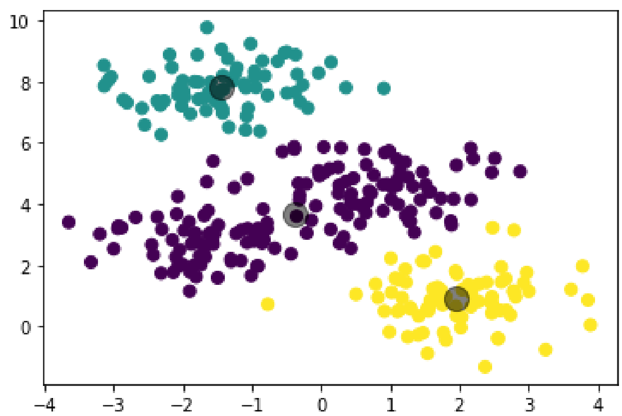
```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
plt.show()
print('Centroids', centers, sep='\n')
```




```
Centroids
[[ 1.97899828  0.83487115]
 [-1.65917487  2.7607673 ]
 [-1.44074146  7.78059306]
 [ 0.85491787  4.44098171]]
```

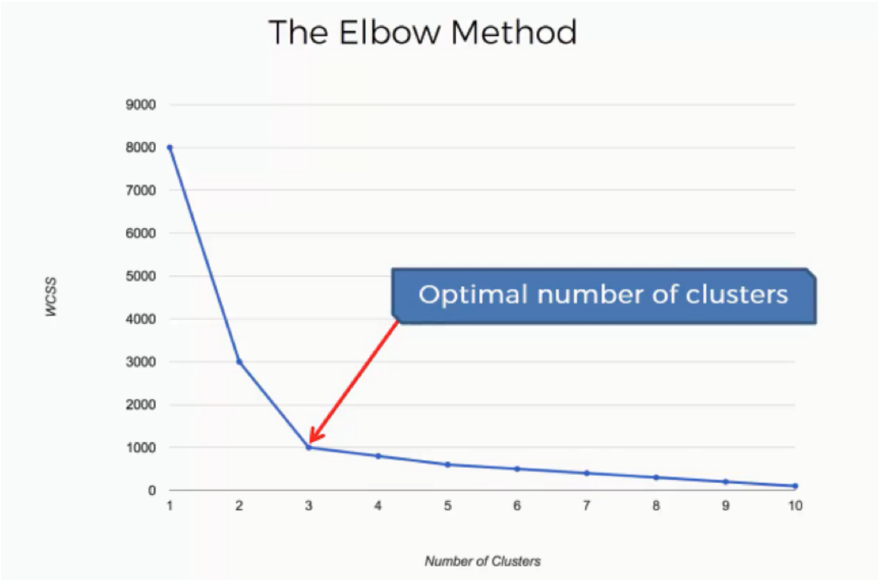
สร้างโมเดล k-means โดยกำหนดให้มี 3 cluster

```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
plt.show()
print('Centroids', centers, sep='\n')
```



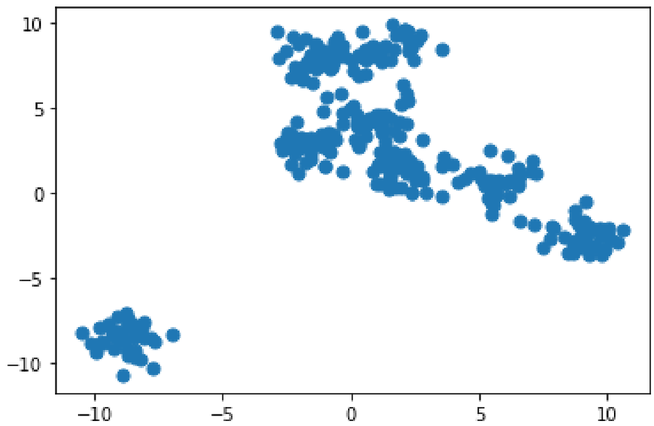

```
Centroids
[[-0.37269409  3.68742579]
 [-1.439055    7.81367705]
 [ 1.96036715  0.8944478 ]]
```

การเลือก cluster ที่เหมาะสมกับข้อมูลด้วย Elbow method และ Silhouette score



- สร้างข้อมูล

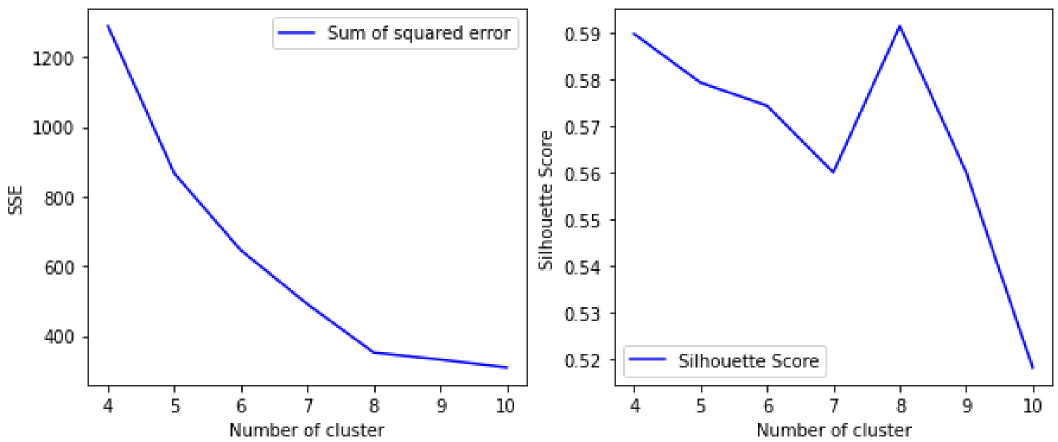
```
from sklearn.datasets.samples_generator import make_blobs
import matplotlib.pyplot as plt

X, y_true = make_blobs(n_samples=300, centers=8,
                       cluster_std=0.80, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);11
plt.show()
```



- คำนวณค่า Sum of squared error และ Silhouette score

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

range_n_clusters = [4, 5, 6, 7 ,8, 9, 10]
elbow = []
ss = []
for n_clusters in range_n_clusters:
   #iterating through cluster sizes
   clusterer = KMeans(n_clusters = n_clusters, random_state=42)
   cluster_labels = clusterer.fit_predict(X)
   #Finding the average silhouette score
   silhouette_avg = silhouette_score(X , cluster_labels)
   ss.append(silhouette_avg)
   print("For n_clusters =", n_clusters,"The average silhouette_score is :", silhouette_avg)
   #Finding the average SSE"
   elbow.append(clusterer.inertia_) # Inertia: Sum of distances of samples to their closest cluster center

fig = plt.figure(figsize=(10,4))
fig.add_subplot(121)
plt.plot(range_n_clusters, elbow,'b-',label='Sum of squared error')
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.legend()
fig.add_subplot(122)
plt.plot(range_n_clusters, ss,'b-',label='Silhouette Score')
plt.xlabel("Number of cluster")
plt.ylabel("Silhouette Score")
plt.legend()
plt.show()

print('Best n_clusters = ', range_n_clusters[ss.index(max(ss))], 'with Silhouette score is : ', max(ss))
```

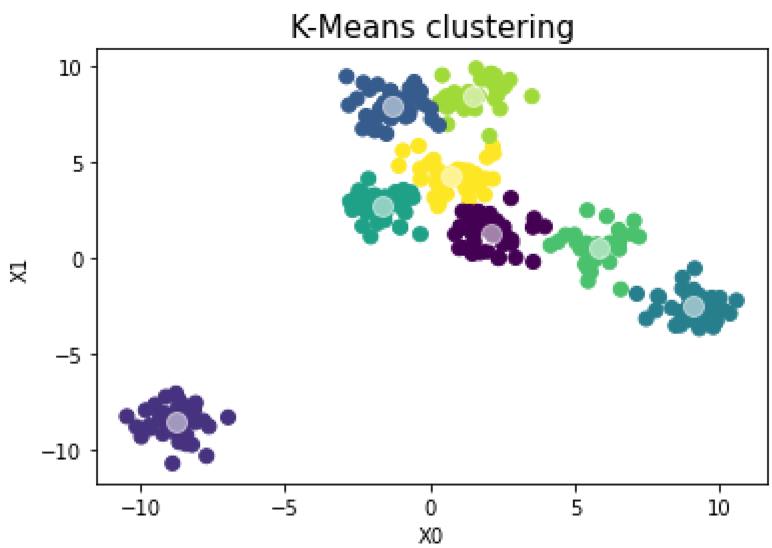
```
For n_clusters = 4 The average silhouette_score is : 0.589738519529238
For n_clusters = 5 The average silhouette_score is : 0.5793149220932733
For n_clusters = 6 The average silhouette_score is : 0.5743851347631377
For n_clusters = 7 The average silhouette_score is : 0.5600911929009983
For n_clusters = 8 The average silhouette_score is : 0.5914087441834297
For n_clusters = 9 The average silhouette_score is : 0.5601200018172094
For n_clusters = 10 The average silhouette_score is : 0.5183564079697408
```



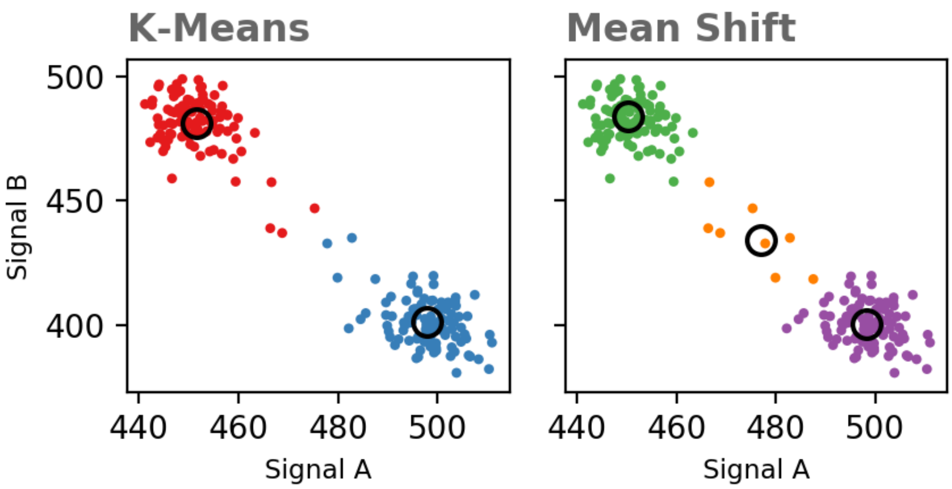

```
Best n_clusters =  8 with Silhouette score is :  0.5914087441834297
```

- สร้างโมเดล k-means โดยกำหนดให้มีจำนวน 8 cluster ตาม elbow method

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

kmeans = KMeans(n_clusters = 8)
cluster_labels = kmeans.fit_predict(X)
centers = kmeans.cluster_centers_

plt.scatter(X[:, 0], X[:, 1], s=50, c=cluster_labels)
plt.scatter(centers[:, 0], centers[:, 1], c='white', s=100, alpha=0.5)

plt.title('K-Means clustering', fontsize = 15)
plt.xlabel('X0')
plt.ylabel('X1')
plt.show()
```

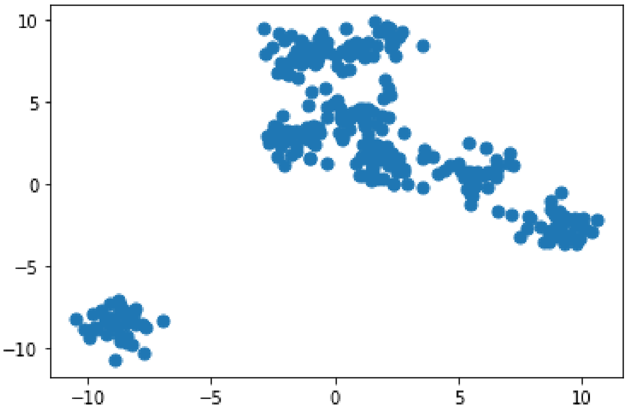


Mean-shift clustering



- สร้างข้อมูล

```
from sklearn.datasets.samples_generator import make_blobs
import matplotlib.pyplot as plt

X, y_true = make_blobs(n_samples=300, centers=8,
                       cluster_std=0.80, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);11
plt.show()
```



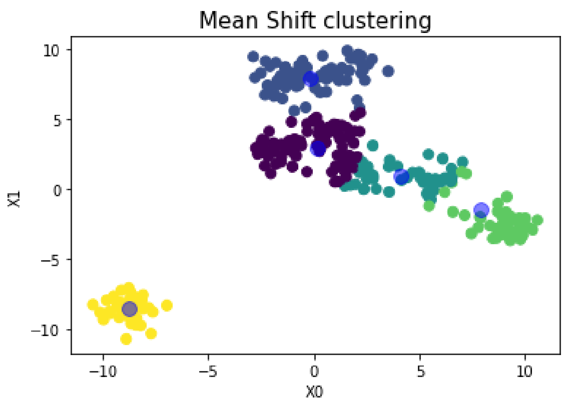
- สร้างโมเดลด้วย MeanShift

```python
import numpy as np
from sklearn.cluster import MeanShift, estimate_bandwidth

# Compute clustering with MeanShift

# The following bandwidth can be automatically detected using
bandwidth = estimate_bandwidth(X, quantile=0.2, n_samples=50)

ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
ms.fit(X)
labels = ms.labels_
cluster_centers = ms.cluster_centers_

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)

print("number of estimated clusters : %d" % n_clusters_)
```

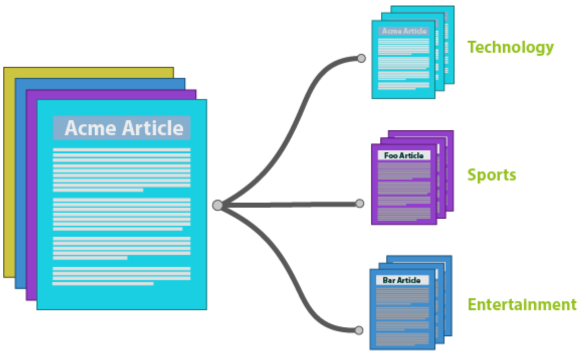
```
number of estimated clusters : 5
```

จากตัวอย่างจำนวน cluster ที่เหมาะสมคือ 5 cluster

```python
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1], s=50, c=labels)
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], c='blue', s=100, alpha=0.5)

plt.title('Mean Shift clustering', fontsize = 15)
plt.xlabel('X0')
plt.ylabel('X1')
plt.show()
```



Workshop – Text classification



The 20 newsgroups text dataset

- Loading data

```
from sklearn.datasets import fetch_20newsgroups

# loading train data
newsgroups_train = fetch_20newsgroups(subset='train')
#newsgroups_train = fetch_20newsgroups(subset='train', shuffle=True)

# loading test data
newsgroups_test = fetch_20newsgroups(subset='test')
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
```

- Exploring data

```
from pprint import pprint

pprint(newsgroups_train.target_names)
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
```

```
1 newsgroups_train.data[0:10]
```

```
["From: lerxst@wam.umd.edu (where's my thing)\nSubject: WHAT car is this
 "From: guykuo@carson.u.washington.edu (Guy Kuo)\nSubject: SI Clock Poll
 'From: twillis@ec.ecn.purdue.edu (Thomas E Willis)\nSubject: PB questio
 'From: jgreen@amber (Joe Green)\nSubject: Re: Weitek P9000 ?\nOrganizat
 'From: jcm@head-cfa.harvard.edu (Jonathan McDowell)\nSubject: Re: Shutt
 'From: dfo@vttoulu.tko.vtt.fi (Foxvog Douglas)\nSubject: Re: Rewording t
 'From: bmdelane@quads.uchicago.edu (brian manning delaney)\nSubject: Bra
 'From: bgrubb@dante.nmsu.edu (GRUBB)\nSubject: Re: IDE vs SCSI\nOrganiza
 'From: holmes7000@iscsvax.uni.edu\nSubject: WIn 3.0 ICON HELP PLEASE!\n
 "From: kerr@ux1.cso.uiuc.edu (Stan Kerr)\nSubject: Re: Sigma Designs Do
```

```
1 print("\n".join(newsgroups_train.data[0].split("\n")[:3])) #prints first line of the first data file
```

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
```

```
1 for i in range(0,5):
2   print("\n".join(newsgroups_train.data[i].split("\n")[:3]))
3   print('\n')
```

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu


From: guykuo@carson.u.washington.edu (Guy Kuo)
Subject: SI Clock Poll - Final Call
Summary: Final call for SI clock reports


From: twillis@ec.ecn.purdue.edu (Thomas E Willis)
Subject: PB questions...
Organization: Purdue University Engineering Computer Network


From: jgreen@amber (Joe Green)
```

```python
print('Number of train instances', newsgroups_train.filenames.shape)
print('Number of test instances', newsgroups_test.filenames.shape)
```

```
Number of train instances (11314,)
Number of test instances (7532,)
```

```python
print(newsgroups_train.target.shape)
print(newsgroups_test.target.shape)
```

```
(11314,)
(7532,)
```

```python
newsgroups_train.target[0:10]
```

```
array([ 7,  4,  4,  1, 14, 16, 13,  3,  2,  4])
```

```python
print('min class', newsgroups_train.target.min())
print('max class', newsgroups_train.target.max())
```

```
min class 0
max class 19
```

Feature engineering – feature extraction

- ตัวอย่างการสร้าง feature

```python
# dictionary
measurements = [
                    {'city': 'Dubai', 'temperature': 33.0},
                    {'city': 'London', 'temperature': 12.},
                    {'city': 'San Francisco', 'temperature': 18.3},
                    ]
```

```python
from sklearn.feature_extraction import DictVectorizer

vec = DictVectorizer()
fe = vec.fit_transform(measurements)

fe
```

```
<3x4 sparse matrix of type '<class 'numpy.float64'>'
	with 6 stored elements in Compressed Sparse Row format>
```

```
print(fe)
```

```
(0, 0)        1.0
(0, 3)        33.0
(1, 1)        1.0
(1, 3)        12.0
(2, 2)        1.0
(2, 3)        18.3
```

```
fe = fe.toarray()
print(fe)
```

```
[[ 1.   0.   0.  33. ]
 [ 0.   1.   0.  12. ]
 [ 0.   0.   1.  18.3]]
```

```
vec.get_feature_names()
```

```
['city=Dubai', 'city=London', 'city=San Francisco', 'temperature']
```

```
import pandas as pd

measure_data = pd.DataFrame(data = fe)
measure_data.columns = vec.get_feature_names()
```

```
measure_data
```

	city=Dubai	city=London	city=San Francisco	temperature
0	1.0	0.0	0.0	33.0
1	0.0	1.0	0.0	12.0
2	0.0	0.0	1.0	18.3

Text feature extraction – bag of words

```
[ ]
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
```

```
corpus = [
          'This is the first document.',
          'This is the second second document.',
          'And the third one.',
          'Is this the first document?',
          ]
```

```
[ ]
# create feature vector from corpus
X = vectorizer.fit_transform(corpus)
X
```

```
<4x9 sparse matrix of type '<class 'numpy.int64'>'
        with 19 stored elements in Compressed Sparse Row format>
```

- แสดงตัวอย่างข้อมูล

```
print(X)
```

```
(0, 8)        1
(0, 3)        1
(0, 6)        1
(0, 2)        1
(0, 1)        1
(1, 8)        1
(1, 3)        1
(1, 6)        1
(1, 1)        1
(1, 5)        2
(2, 6)        1
(2, 0)        1
(2, 7)        1
(2, 4)        1
(3, 8)        1
(3, 3)        1
(3, 6)        1
(3, 2)        1
(3, 1)        1
```

- สร้าง feature vector

```
# feature vector
print(X.toarray())
```

```
[[0 1 1 1 0 0 1 0 1]
 [0 1 0 1 0 2 1 0 1]
 [1 0 0 0 1 0 1 1 0]
 [0 1 1 1 0 0 1 0 1]]
```

- feature names

```
# label - feature names
vectorizer.get_feature_names()
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

- create feature vector from new text

```
vectorizer.transform(['Something completely new.']).toarray()
```

```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
[ ]
vectorizer.transform(['This is the second second document.']).toarray()
```

```
array([[0, 1, 0, 1, 0, 2, 1, 0, 1]])
```

N-gram - bigram

```
bigram_vectorizer = CountVectorizer(ngram_range=(1, 2),
                                    token_pattern=r'\b\w+\b', min_df=1)
analyze = bigram_vectorizer.build_analyzer()
```

```
[ ]
analyze('Bi-grams are cool!')
```

```
['bi', 'grams', 'are', 'cool', 'bi grams', 'grams are', 'are cool']
```

```
#bigram - the vocabulary extracted
X_2 = bigram_vectorizer.fit_transform(corpus).toarray()
X_2
```

```
array([[0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0],
       [0, 0, 1, 0, 0, 1, 1, 0, 0, 2, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0],
       [0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1]])
```

Tf-idf term weighting

```
corpus = [
          'This is the first document.',
          'This is the second second document.',
          'And the third one.',
          'Is this the first document?',
          ]
```

```
[ ]
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X_counts = count_vect.fit_transform(corpus)
X_counts.shape
```

```
(4, 9)
```

```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()
X_tfidf = tfidf_transformer.fit_transform(X_counts)
```

```
[ ]
X_tfidf = X_tfidf.toarray()
print(X_tfidf)
```

```
[[0.         0.43877674 0.54197657 0.43877674 0.         0.
  0.35872874 0.         0.43877674]
 [0.         0.27230147 0.         0.27230147 0.         0.85322574
  0.22262429 0.         0.27230147]
 [0.55280532 0.         0.         0.         0.55280532 0.
  0.28847675 0.55280532 0.        ]
 [0.         0.43877674 0.54197657 0.43877674 0.         0.
  0.35872874 0.         0.43877674]]
```

Machine learning

Create feature vector using Tf-idf

```
from sklearn.feature_extraction.text import CountVectorizer

count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(newsgroups_train.data)

X_test_counts = count_vect.transform(newsgroups_test.data)
```

```
[ ]
print(X_train_counts.shape)
print(X_test_counts.shape)
```

```
(11314, 130107)
(7532, 130107)
```

```
# TF-IDF
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

X_test_tfidf = tfidf_transformer.transform(X_test_counts)
```

```
[ ]
print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
```

```
(11314, 130107)
(7532, 130107)
```

Training data using Naive Bayes

```
# check size of train and test data
print(X_train_tfidf.shape)
print(newsgroups_train.target.shape)

print(X_test_tfidf.shape)
print(newsgroups_test.target.shape)
```

```
(11314, 130107)
(11314,)
(7532, 130107)
(7532,)
```

```
from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB()
clf = clf.fit(X_train_tfidf, newsgroups_train.target)
```

Predicting test data

```
# Performance of NB Classifier
import numpy as np

predicted = clf.predict(X_test_tfidf)
np.mean(predicted == newsgroups_test.target)
```

```
0.7738980350504514
```

Accuracy score

```
from sklearn.metrics import accuracy_score

accuracy_score(newsgroups_test.target, predicted)
```

```
0.7738980350504514
```

Classification report

```
from sklearn.metrics import classification_report

print(classification_report(newsgroups_test.target, predicted))
```

```
              precision    recall  f1-score   support

           0       0.80      0.52      0.63       319
           1       0.81      0.65      0.72       389
           2       0.82      0.65      0.73       394
           3       0.67      0.78      0.72       392
           4       0.86      0.77      0.81       385
           5       0.89      0.75      0.82       395
           6       0.93      0.69      0.80       390
           7       0.85      0.92      0.88       396
           8       0.94      0.93      0.93       398
           9       0.92      0.90      0.91       397
          10       0.89      0.97      0.93       399
          11       0.59      0.97      0.74       396
          12       0.84      0.60      0.70       393
          13       0.92      0.74      0.82       396
          14       0.84      0.89      0.87       394
          15       0.44      0.98      0.61       398
          16       0.64      0.94      0.76       364
          17       0.93      0.91      0.92       376
          18       0.96      0.42      0.58       310
          19       0.97      0.14      0.24       251

    accuracy                           0.77      7532
   macro avg       0.83      0.76      0.76      7532
weighted avg       0.82      0.77      0.77      7532
```

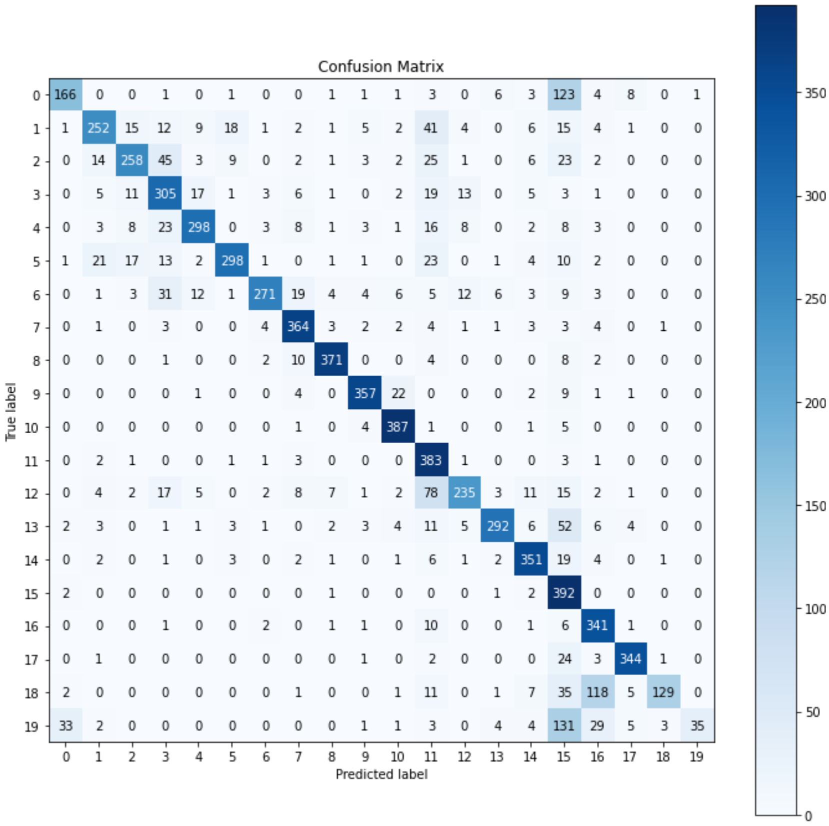
Confusion matrix

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(newsgroups_test.target, predicted))
```

```
[[166   0   0   1   0   1   0   0   1   1   1   3   0   6   3 123   4   8
    0   1]
 [  1 252  15  12   9  18   1   2   1   5   2  41   4   0   6  15   4   1
    0   0]
 [  0  14 258  45   3   9   0   2   1   3   2  25   1   0   6  23   2   0
    0   0]
 [  0   5  11 305  17   1   3   6   1   0   2  19  13   0   5   3   1   0
    0   0]
 [  0   3   8  23 298   0   3   8   1   3   1  16   8   0   2   8   3   0
    0   0]
 [  1  21  17  13   2 298   1   0   1   1   0  23   0   1   4  10   2   0
    0   0]
```

```
# install python package
!pip install -q scikit-plot
```

```
import scikitplot as skplt

skplt.metrics.plot_confusion_matrix(
    newsgroups_test.target,
    predicted,
    figsize=(12,12))
```



Cross-validation

```
from sklearn.model_selection import cross_val_score
import numpy as np

cross_score = cross_val_score(clf, X_train_tfidf, newsgroups_train.target, cv=5, scoring='recall_macro')
print(cross_score)
print('Average:', np.average(cross_score))
```

```
[0.83263535 0.82639709 0.82621982 0.82267228 0.82857016]
Average: 0.8272989381933493
```

Building pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

#training
text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                     ('clf', MultinomialNB())])
text_clf = text_clf.fit(newsgroups_train.data, newsgroups_train.target)

#predict
predicted = text_clf.predict(newsgroups_test.data)
accuracy_score(newsgroups_test.target, predicted)
```

```
0.7738980350504514
```

```
# Training Support Vector Machines - SVM and calculating its performance

from sklearn.linear_model import SGDClassifier
text_clf_svm = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                         ('clf-svm', SGDClassifier())])

text_clf_svm = text_clf_svm.fit(newsgroups_train.data, newsgroups_train.target)
predicted_svm = text_clf_svm.predict(newsgroups_test.data)
accuracy_score(newsgroups_test.target, predicted)
```

```
0.7738980350504514
```