INTRODUCTION

We have a data which classified if patients have heart disease or not according to features in it. We will try to use this data to create a model which tries predict if a patient has this disease or not. We will use logistic regression (classification) algorithm.

1

```
# This Python 3 environment comes with many helpful analytics libraries
installed
# It is defined by the kaggle/python docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np
import pandas as pd          # for data manipulation and analysis
import matplotlib.pyplot as plt
import seaborn as sns        # a Python data visualization library based on
matplotlib
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will
list the files in the input directory

import os
print(os.listdir("E:/input"))

# Any results you write to the current directory are saved as output.
['bike-sharing-dataset', 'heart.csv', 'pima-diabetes']
```

Read Data

2

```
# We are reading our data
df = pd.read_csv("E:/input/heart.csv")
```

No output

3

```
# First 5 rows of our data
```

```
df.head()
```

3

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Data contains;

- age - age in years
- sex - (1 = male; 0 = female)
- cp - chest pain type
  #胸痛型
- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
  #血清胆汁淤积
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

- restecg - resting electrocardiographic results
  #静息心电图结果
- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
  #运动性心绞痛
- oldpeak - ST depression induced by exercise relative to rest
- slope - the slope of the peak exercise ST segment
- ca - number of major vessels (0-3) colored by flourosopy
- thal - 3 = normal; 6 = fixed defect; 7 = reversable defect
- target - have disease or not (1=yes, 0=no)

Data Exploration

4

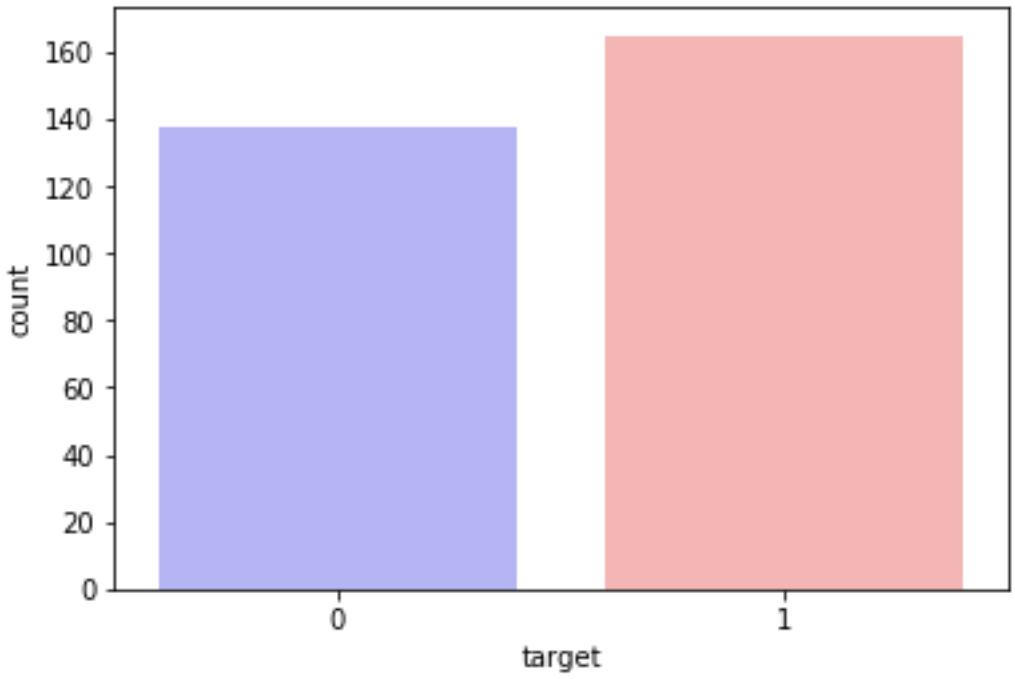
```
df.target.value_counts()
```

4

```
1    165
0    138
Name: target, dtype: int64
```

5

```
sns.countplot(x="target", data=df,palette="bwr")     #palette 颜色  #Show the counts of observations in each categorical bin using bars.
plt.show()
```

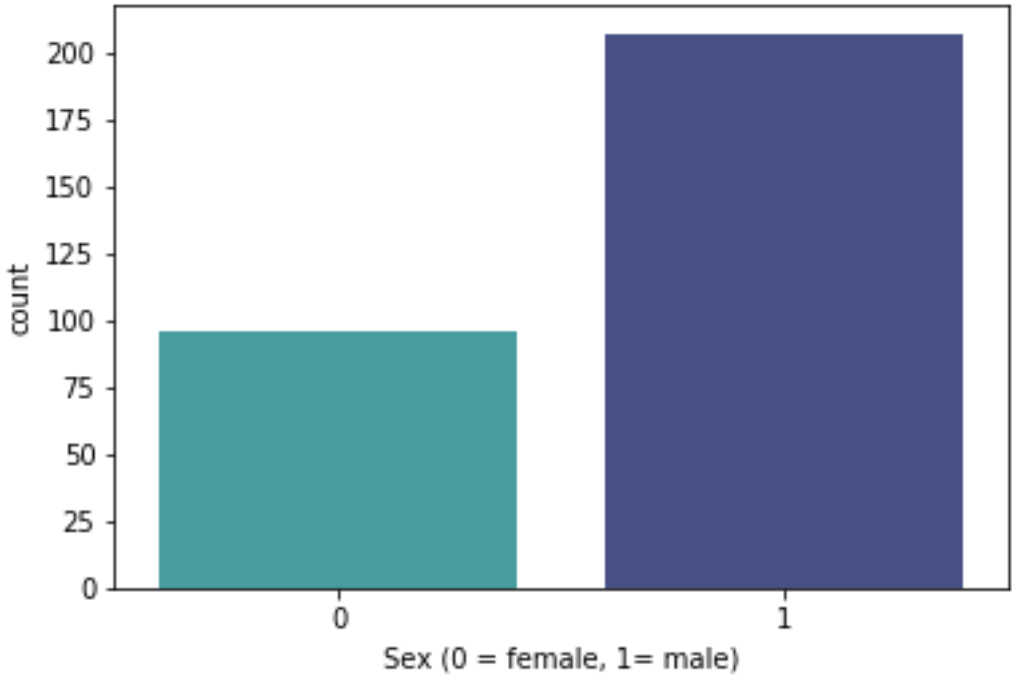


6

```
countNoDisease = len(df[df.target == 0])
countHaveDisease = len(df[df.target == 1])
print("Percentage of Patients Haven't Heart Disease:
{:.2f}%".format((countNoDisease / (len(df.target))*100)))
print("Percentage of Patients Have Heart Disease:
{:.2f}%".format((countHaveDisease / (len(df.target))*100)))
Percentage of Patients Haven't Heart Disease: 45.54%
Percentage of Patients Have Heart Disease: 54.46%
```

7

```
sns.countplot(x='sex', data=df, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```



8

```
countFemale = len(df[df.sex == 0])
countMale = len(df[df.sex == 1])
print("Percentage of Female Patients: {:.2f}%".format((countFemale /
(len(df.sex))*100)))
print("Percentage of Male Patients: {:.2f}%".format((countMale /
(len(df.sex))*100)))
Percentage of Female Patients: 31.68%
Percentage of Male Patients: 68.32%
```

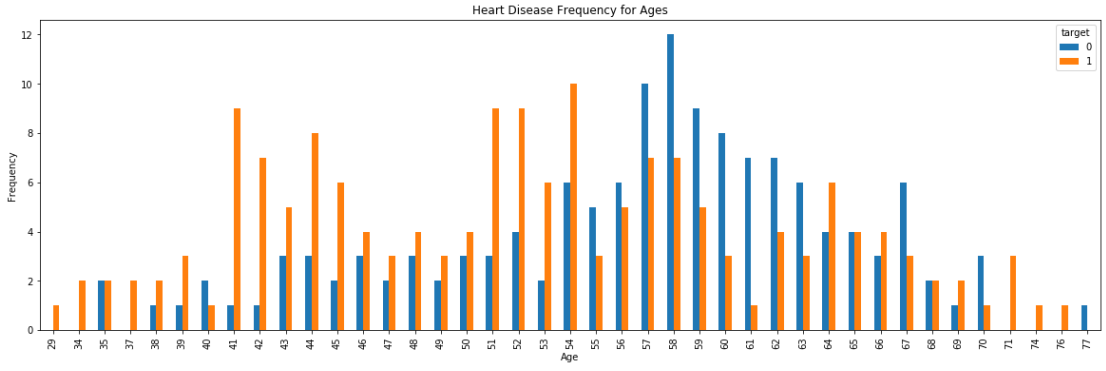
9

```
df.groupby('target').mean()          #分组看均值
```

9

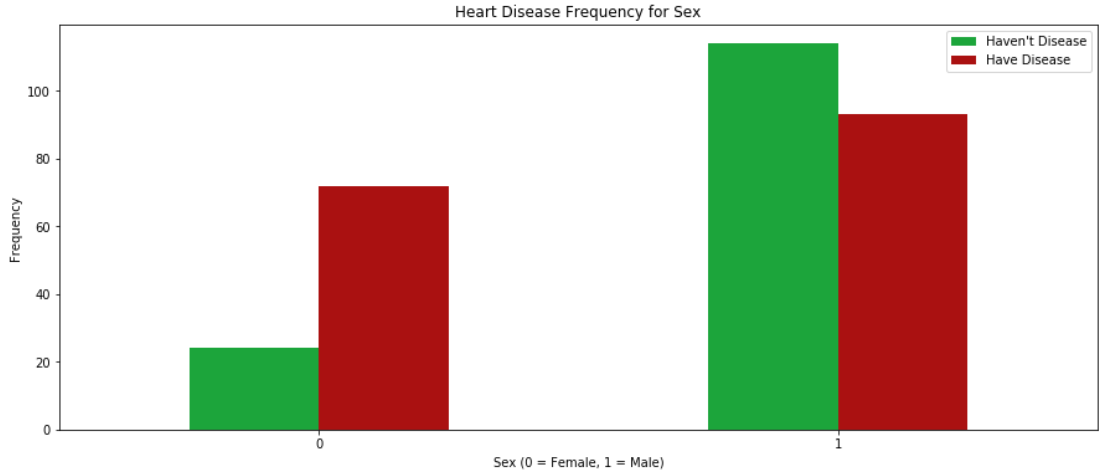
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
target													
0	56.601449	0.8260087	0.4782611	134.3985 51	251.0869 57	0.1594 20	0.4492 75	139.1014 49	0.5507 25	1.5855 07	1.1666 67	1.1666 67	2.5434 78
1	52.496970	0.5636 36	1.3757 58	129.3030 30	242.2303 03	0.1393 94	0.5939 39	158.4666 67	0.1393 94	0.5830 30	1.5939 39	0.3636 36	2.1212 12


```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6))   #A crosstab
is a table showing the relationship between two or more variables
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.savefig('heartDiseaseAndAges.png')
plt.show()
```



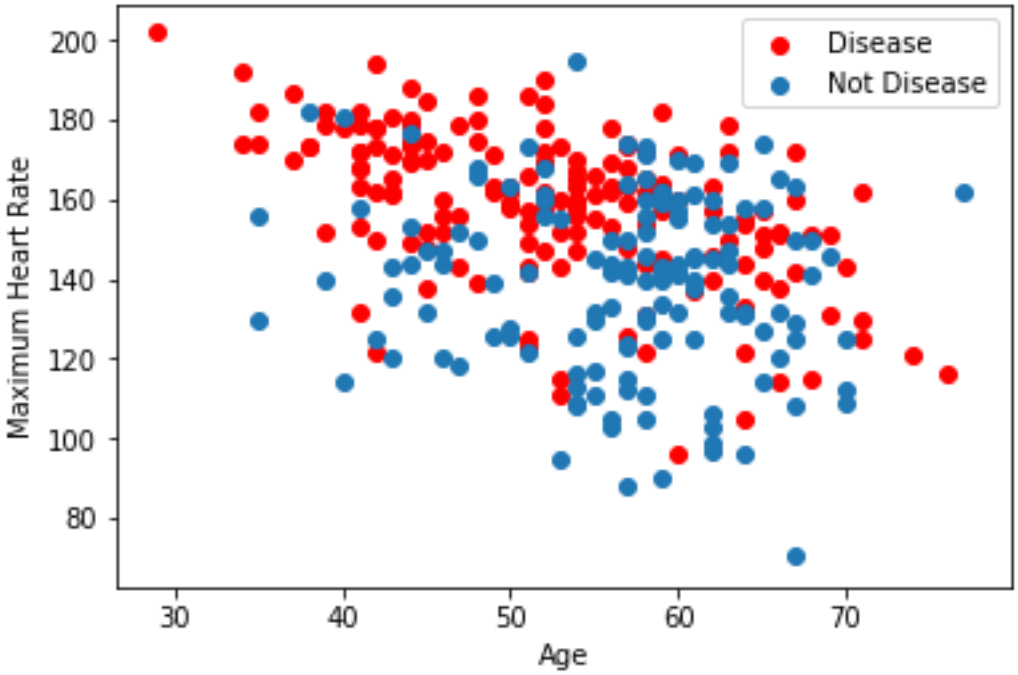
11

```
pd.crosstab(df.sex,df.target).plot(kind="bar",figsize=(15,6),color=['#1CA53B
','#AA1111' ])
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex (0 = Female, 1 = Male)')
plt.xticks(rotation=0)
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency')
plt.show()
```



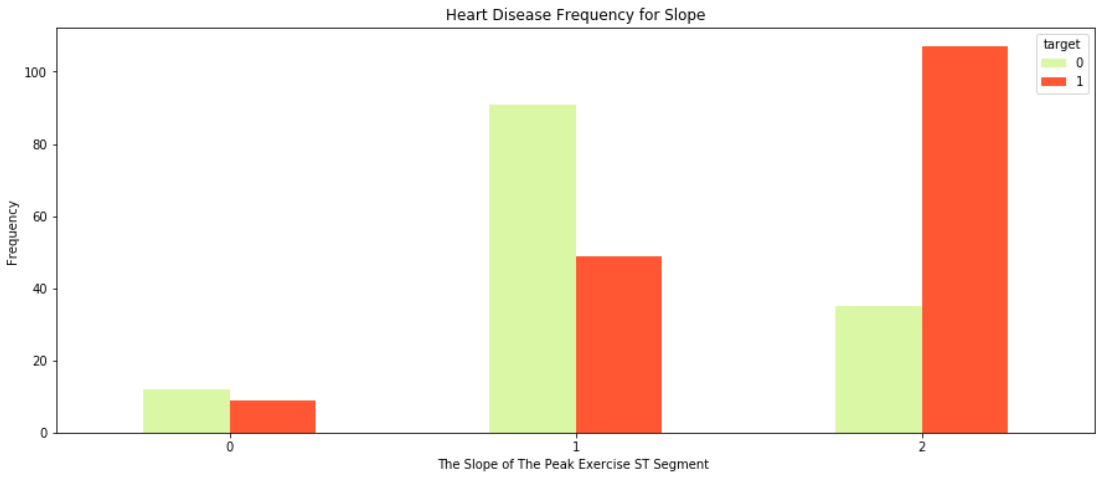
12

```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



13

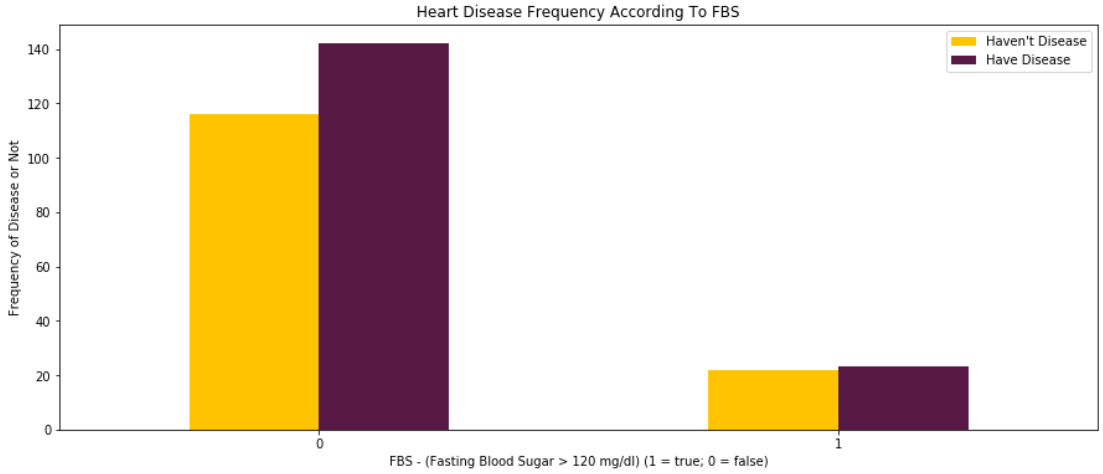
```
pd.crosstab(df.slope,df.target).plot(kind="bar",figsize=(15,6),color=['#DAF7
A6','#FF5733' ])
plt.title('Heart Disease Frequency for Slope')
plt.xlabel('The Slope of The Peak Exercise ST Segment ')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')
plt.show()
```



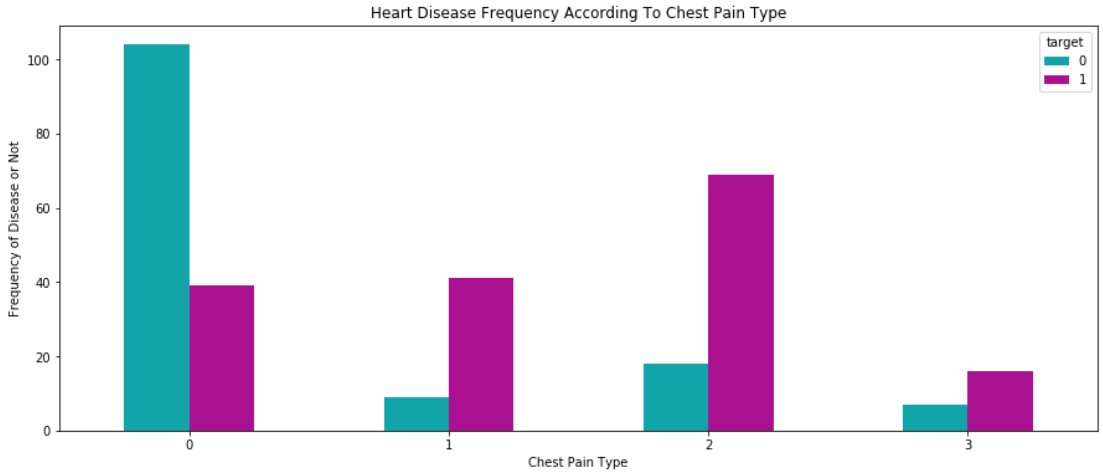
14

```
pd.crosstab(df.fbs,df.target).plot(kind="bar",figsize=(15,6),color=['#FFC300
','#581845' ])
plt.title('Heart Disease Frequency According To FBS')
plt.xlabel('FBS - (Fasting Blood Sugar > 120 mg/dl) (1 = true; 0 = false)')
plt.xticks(rotation = 0)
```

```
plt.legend(["Haven't Disease", "Have Disease"])
plt.ylabel('Frequency of Disease or Not')
plt.show()
```




```
pd.crosstab(df.cp,df.target).plot(kind="bar",figsize=(15,6),color=['#11A5AA'
,'#AA1190' ])
plt.title('Heart Disease Frequency According To Chest Pain Type')
plt.xlabel('Chest Pain Type')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Disease or Not')
plt.show()
```



Creating Dummy Variables

Since 'cp', 'thal' and 'slope' are categorical variables we'll turn them into dummy variables.


```
a = pd.get_dummies(df['cp'], prefix = "cp")
```

```
b = pd.get_dummies(df['thal'], prefix = "thal")
c = pd.get_dummies(df['slope'], prefix = "slope")
```

No output

17

```
frames = [df, a, b, c]
df = pd.concat(frames, axis = 1)
df.head()
```

17

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	...	cp_1	cp_2	cp_3	thal_0	thal_1	thal_2	thal_3	slope_0	slope_1	slope_2
0	63	1	3	145	233	1	0	150	0	2.3	...	0	0	1	0	1	0	0	1	0	0
1	37	1	2	130	250	0	1	187	0	3.5	...	0	1	0	0	0	1	0	1	0	0
2	41	0	1	130	204	0	0	172	0	1.4	...	1	0	0	0	0	1	0	0	0	1
3	56	1	1	120	236	0	1	178	0	0.8	...	1	0	0	0	0	1	0	0	0	1
4	57	0	0	120	354	0	1	163	1	0.6	...	0	0	0	0	0	1	0	0	0	1

5 rows × 25 columns

18

```
df = df.drop(columns = ['cp', 'thal', 'slope'])
```

```
df.head()
```


	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	...	cp_1	cp_2	cp_3	thal_0	thal_1	thal_2	thal_3	slope_0	slope_1	slope_2
0	63	1	145	233	1	0	150	0	2.3	0	...	0	0	1	0	1	0	0	1	0	0
1	37	1	130	250	0	1	187	0	3.5	0	...	0	1	0	0	0	1	0	1	0	0
2	41	0	130	204	0	0	172	0	1.4	0	...	1	0	0	0	0	1	0	0	0	1
3	56	1	120	236	0	1	178	0	0.8	0	...	1	0	0	0	0	1	0	0	0	1
4	57	0	120	354	0	1	163	1	0.6	0	...	0	0	0	0	0	1	0	0	0	1

5 rows × 22 columns

Creating Model for Logistic Regression

We can use sklearn library or we can write functions ourselves. Let's them both. Firstly we will write our functions after that we'll use sklearn library to calculate score.

```
y = df.target.values
x_data = df.drop(['target'], axis = 1)
```

No output

Normalize Data

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

20

```
# Normalize
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values
```

No output

We will split our data. 80% of our data will be train data and 20% of it will be test data.

21

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size =
0.2,random_state=0)
```

No output

Sklearn Logistic Regression

22

```
accuracies = {}

lr = LogisticRegression()
lr.fit(x_train,y_train)
acc = lr.score(x_test,y_test)*100

accuracies['Logistic Regression'] = acc
print("Test Accuracy {:.2f}%".format(acc))
D:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this
warning.
  FutureWarning)
```

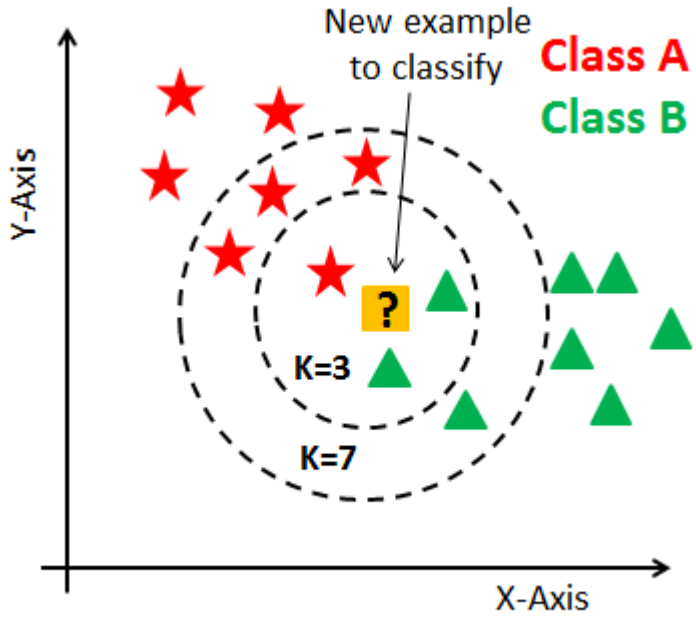
```
Test Accuracy 86.89%
```

1. **Our model works with 86.89%accuracy.**

K-Nearest Neighbour (KNN) Classification

Let's see what will be score if we use KNN algorithm.

KNN Algorithm



23

```
# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(x_train, y_train)
prediction = knn.predict(x_test)

print("{} NN Score: {:.2f}%".format(2, knn.score(x_test, y_test)*100))
```

```
2 NN Score: 77.05%
```

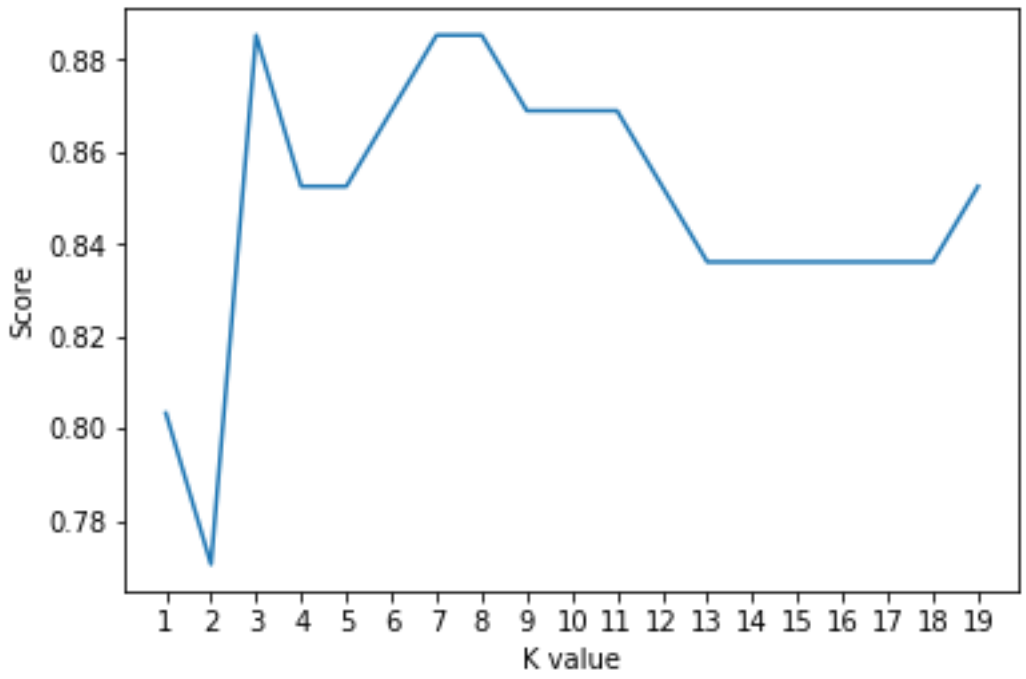
24

```
# try ro find best k value
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
```

```
    knn2.fit(x_train, y_train)
    scoreList.append(knn2.score(x_test, y_test))

plt.plot(range(1,20), scoreList)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()

acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))
```




```
Maximum KNN Score is 88.52%
```

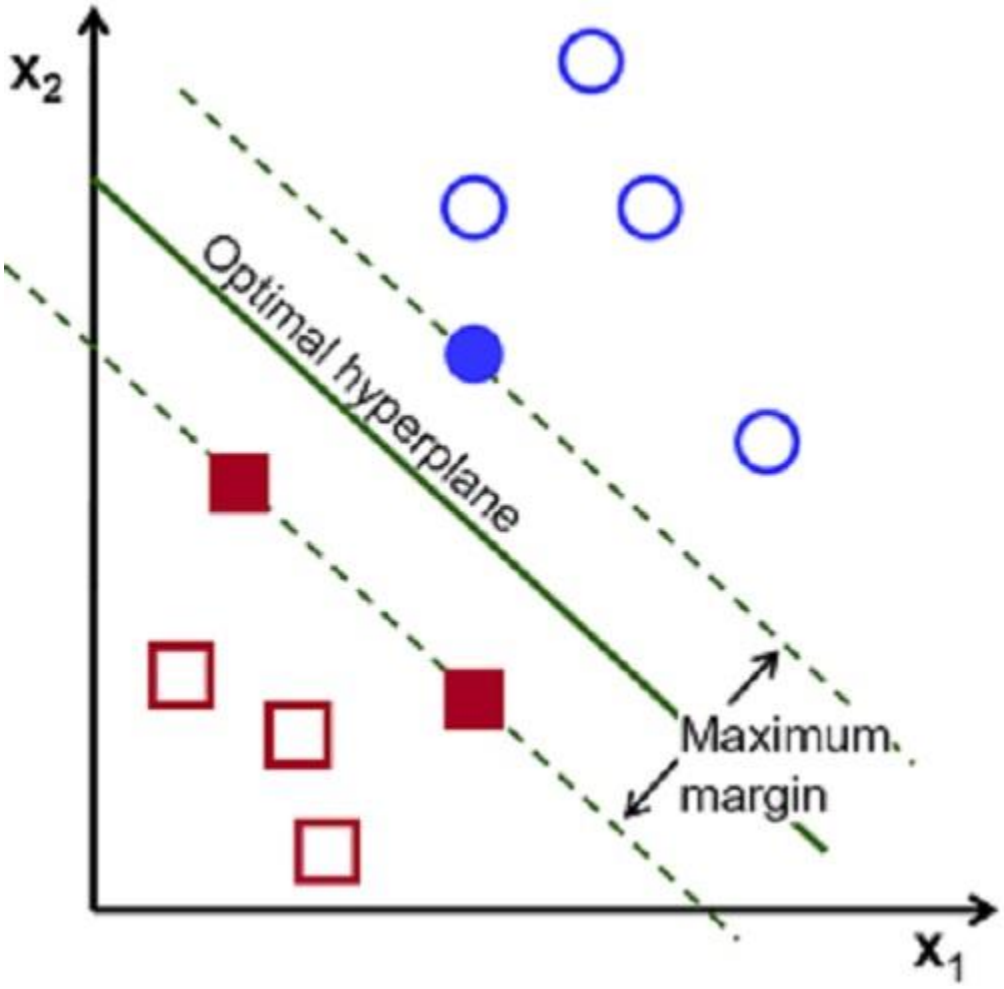
As you can see above if we define k as 3-7-8 we will reach maximum score.

KNN Model's Accuracy is 88.52%

Support Vector Machine (SVM) Algorithm

Now we will use SVM algorithm.

Support Vector Machine Algorithm



25

```
from sklearn.svm import SVC
```

No output

26

```
svm = SVC(random_state = 1)
svm.fit(x_train, y_train)

acc = svm.score(x_test,y_test)*100
accuracies['SVM'] = acc
print("Test Accuracy of SVM Algorithm: {:.2f}%".format(acc))
```

```
Test Accuracy of SVM Algorithm: 86.89%
D:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:193:
FutureWarning: The default value of gamma will change from 'auto' to 'scale'
in version 0.22 to account better for unscaled features. Set gamma
explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Test Accuracy of SVM Algorithm is 86.89%

Naive Bayes Algorithm

Naive Bayes Algorithm

27

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)

acc = nb.score(x_test,y_test)*100
accuracies['Naive Bayes'] = acc
print("Accuracy of Naive Bayes: {:.2f}%".format(acc))
Accuracy of Naive Bayes: 86.89%
```

Accuracy of Naive Bayes: 86.89%

Decision Tree Algorithm

Decision Tree Algorithm

28

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)

acc = dtc.score(x_test, y_test)*100
accuracies['Decision Tree'] = acc
print("Decision Tree Test Accuracy {:.2f}%".format(acc))
Decision Tree Test Accuracy 78.69%
```

Test Accuracy of Decision Tree Algorithm: 78.69%

Random Forest Classification

29

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(x_train, y_train)

acc = rf.score(x_test,y_test)*100
accuracies['Random Forest'] = acc
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
Random Forest Algorithm Accuracy Score : 88.52%
```

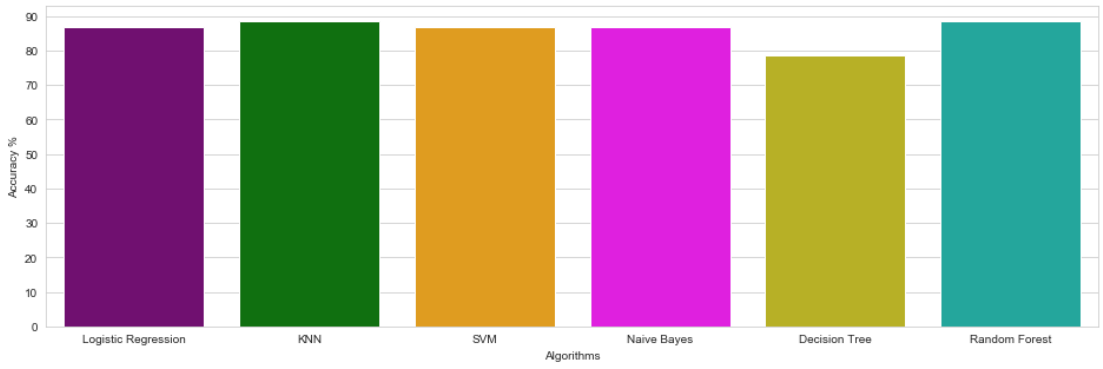
Test Accuracy of Random Forest: 88.52%

Comparing Models

30

```
colors = ["purple", "green", "orange", "magenta","#CFC60E","#0FBBAE"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))          #y 轴以 10 为单位
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()),
palette=colors)
plt.show()
```



V