Machine Learning 🖥️

Naive Bayes

𝓑𝔂 ⟹ 𝓟𝓡𝓘𝓝𝓒𝓔 👑❤️

Mounting Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Bayes' Theorem ⟶

- ***Bayes' Theorem is a simple mathematical formula used for calculating conditional probabilites.***
- ***Bayes' Theorem provides a way to calculate the probability of a hypothesis based on its prior probability.***

Formula ⟶

P (A | B) = [P (B | A) * P (A) / P (B)]

where,

A,B ⟶ Events

P (A | B) ⟶ Prpbability of event A given event B has already happened.

- **Probablity of A given event B is true.**

P (B | A) ⟶ Probablity of event B given event A has already happened.

- **Probablity of B given event A is true.**

P (A) ⟶ The independent probablity of A.

P (B) ⟶ The independent probablity of B.

▶ P (B | A) ⟹ Conditional Probablity

Question :

There is a cricket match tomorrow and weather man predicted there are 5% chance that will be rain. The weather man correctly predicts 90% of the time. It rained 5 day in the previous year. What is the probability that will be rain on the given day of cricket match.

▶ Solution :

P(A) = 5 / 365 = 0.0136

P(Ā) = 1 - P(A) = 1 - 0.0136 = 0.9864

P(B | A) = 90% = 0.9

P(B | Ā) = 1 - 0.9 = 0.1

P(B) = ?

P(B) = P(B and A) + P(B and Ā)

P(B) = P(B | A) * P(A) + P(B | Ā) * P(Ā)

P(B) = 0.9 * 0.0136 + 0.1 * 0.9864

P(B) = 0.11 = 11%

So, there is 11% chance that will be rain on the given day of cricket match.

Python Implementation for Naive Bayes

on Pima Diabetes Dataset

```
# Python Implementation for Naive Bayes
# Importing pandas and numpy
import numpy as np
import pandas as pd
# Uploading and Reading 'Pima Diabetes.csv' dataset.
df = pd.read_csv('/content/drive/MyDrive/Dataset/pima_diabetes..csv')
```

```
df.head()
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.tail()
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
df.shape
```

(768, 9)

```
df.describe().T
```

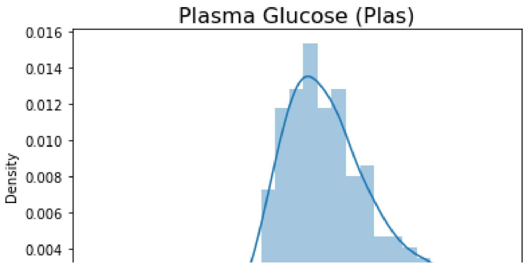
	count	mean	std	min	25%	50%	75%	max
Preg	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Plas	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
Pres	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
skin	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
test	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
mass	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
pedi	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

Data Visualization

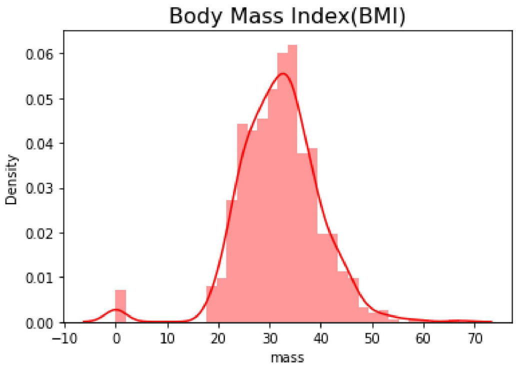
```
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
import matplotlib.pyplot as plt
```

Distplot

```
sns.distplot(df['Plas'])
plt.title('Plasma Glucose (Plas)', fontsize = 16)
plt.show()
```

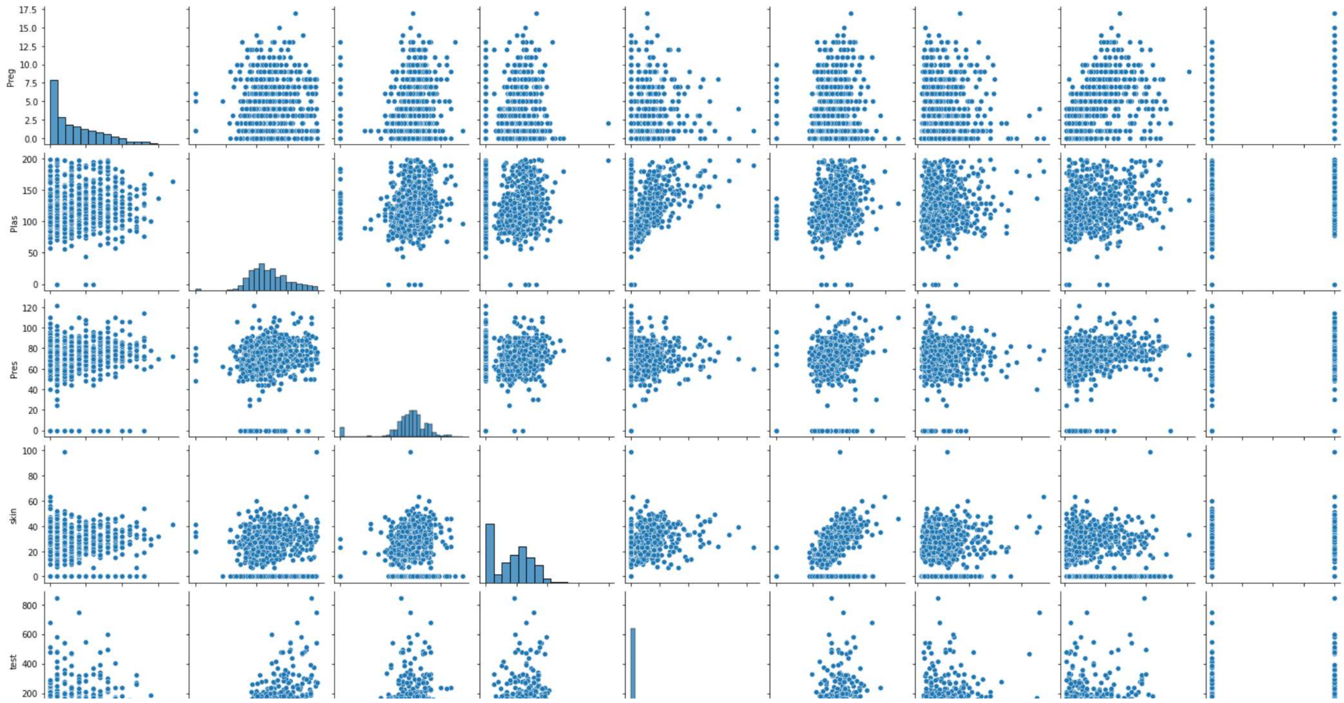



```
sns.distplot(df['mass'], color = 'r')
plt.title('Body Mass Index(BMI)', fontsize = 16)
plt.show()
```

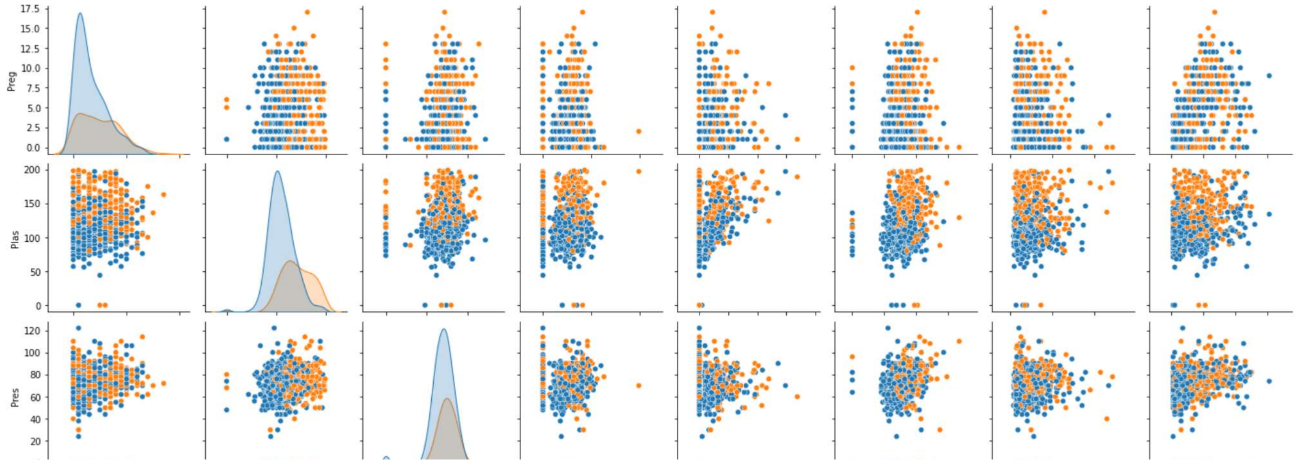


Pairplot

```
sns.pairplot(df)
plt.show()
```

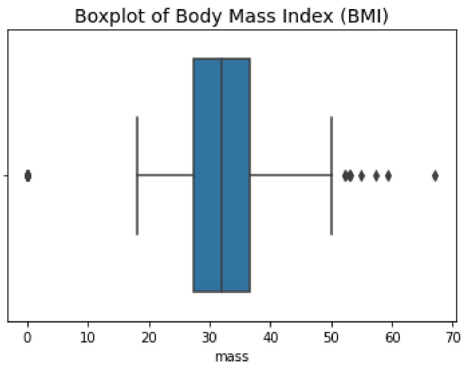



```
sns.pairplot(df, hue = 'class')
plt.show()
```

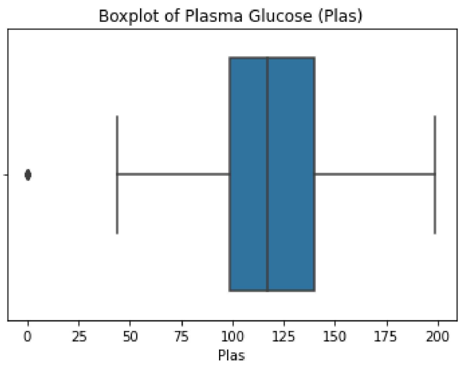


Boxplot

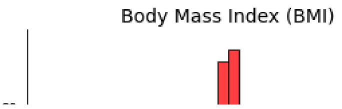
```
sns.boxplot(df['mass'])
plt.title('Boxplot of Body Mass Index (BMI)', fontsize = 14)
plt.show()
```



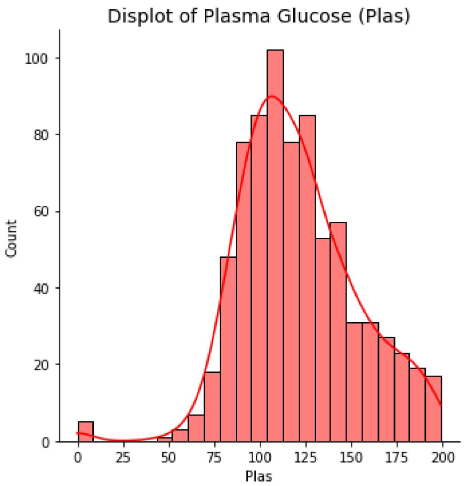

```
sns.boxplot(df['Plas'])
plt.title('Boxplot of Plasma Glucose (Plas)')
plt.show()
```




```
sns.displot(df['mass'], color = 'r')
plt.title('Body Mass Index (BMI)', fontsize = 14)
plt.show()
```




```
sns.displot(df['Plas'], color = 'r', kde = True)
plt.title('Displot of Plasma Glucose (Plas)', fontsize = 14)
plt.show()
```

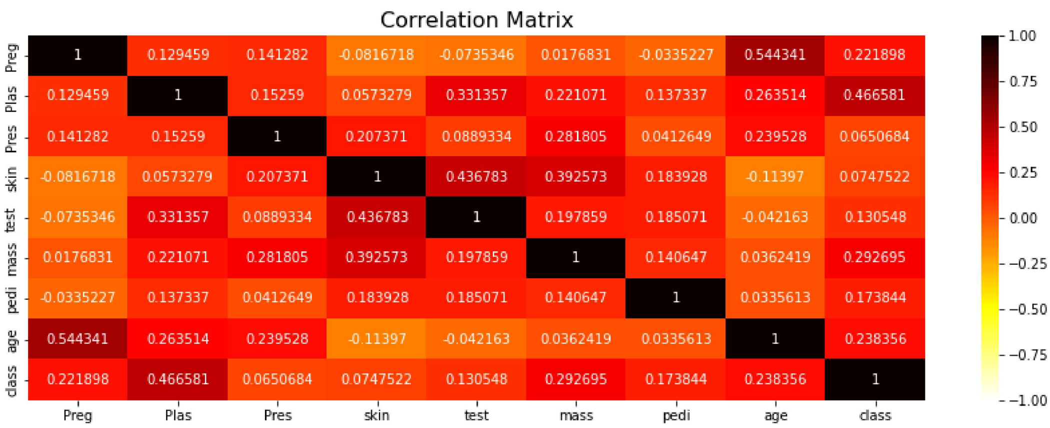


Correlation

```
corr = df.corr()
corr
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
Preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Plas	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
Pres	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
test	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
mass	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
pedi	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

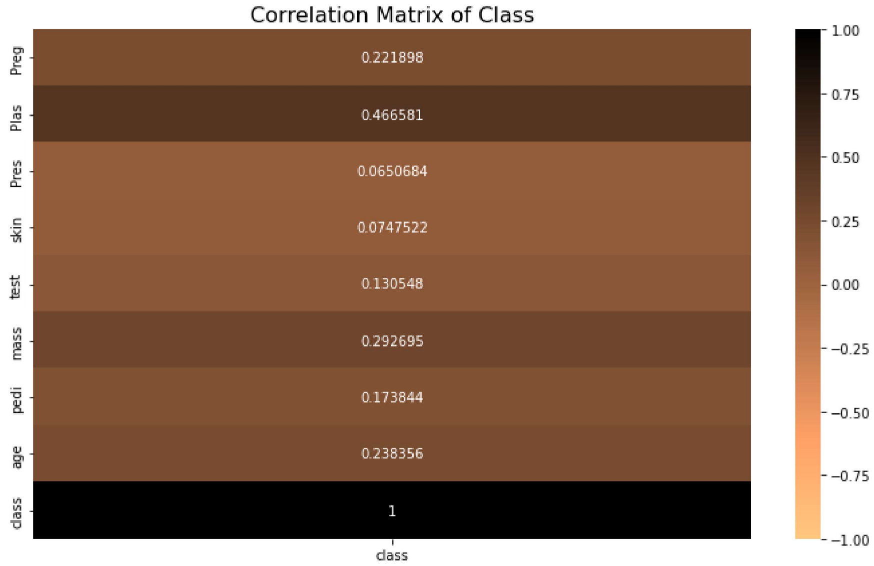
```
plt.figure(figsize=(15,5))
plt.title('Correlation Matrix', fontsize = 16)
sns.heatmap(corr, annot = True, fmt = 'g', vmax = 1.0, vmin = -1.0, cmap = 'hot_r')
plt.show()
```




```
corr[['class']]
```

	class
Preg	0.221898
Plas	0.466581
Pres	0.065068
skin	0.074752
test	0.130548
mass	0.292695
pedi	0.173844
age	0.238356

```
plt.figure(figsize=(12,7))
plt.title('Correlation Matrix of Class', fontsize = 16)      # Class = Outcome
sns.heatmap(corr[['class']], annot = True, cmap = 'copper_r', fmt = 'g', vmax = 1.0, vmin = -1.0)
plt.show()
```



Checking Missing Values

```
df.isna().apply(pd.value_counts).T
```

	False
Preg	768
Plas	768
Pres	768
skin	768
test	768
mass	768
pedi	768
age	768
class	768

```
df.isna().sum()
```

```
Preg     0
Plas     0
Pres     0
skin     0
test     0
mass     0
pedi     0
age      0
class    0
dtype: int64
```

Prepare Dataset

```
x = df.drop(['class'], axis = 1)
y = df[['class']]
```

```
x.head()
```

	Preg	Plas	Pres	skin	test	mass	pedi	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

```
y.head()
```

	class
0	1
1	0
2	1
3	0
4	1

Train Test Split

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.3, random_state = 1)
```

```
xtrain.shape, xtest.shape
```

```
((537, 8), (231, 8))
```

```
ytrain.shape, ytest.shape
```

```
((537, 1), (231, 1))
```

Fourth Machine Learning Model

```
from sklearn.naive_bayes import GaussianNB
# Define our Fourth ML Model
model = GaussianNB()
```

```
model.fit(xtrain, ytrain)
```

```
GaussianNB()
```

Prediction

```
# Predict
ypred = model.predict(xtest)
ypred
```

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

Accuracy, Confusion Matrix, Classification Report

Accuracy Score

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
accuracy_score(ytest, ypred)
```

```
0.7835497835497836
```

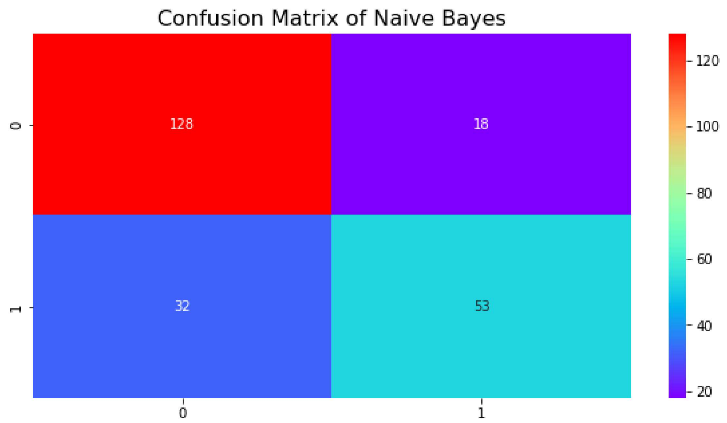
```
# 0.78354 is a good accuracy score because it is higher than 0.7
```

Confusion Matrix

```
cf = confusion_matrix(ytest, ypred)
cf
```

```
array([[128,  18],
       [ 32,  53]])
```

```
plt.figure(figsize=(10,5))
sns.heatmap(cf, cmap = 'rainbow', annot = True, fmt = 'g')
plt.title('Confusion Matrix of Naive Bayes', fontsize = 16 )
plt.show()
```



Classification Report

```
# Classification Report
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       146
           1       0.75      0.62      0.68        85

    accuracy                           0.78       231
   macro avg       0.77      0.75      0.76       231
weighted avg       0.78      0.78      0.78       231
```

Compare with Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

```
lr.fit(xtrain, ytrain)
```

```
LogisticRegression()
```

```
lr_pred = lr.predict(xtest)
lr_pred
```

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

```
accuracy_score(ytest, lr_pred)
```

```
0.7835497835497836
```

```
# Nave Bayes and Logistic Regression both have same accuracy score.
# It is Rare.
```

Naive Bayes and Logistic Regression have same accuracy score. It is rare.

Compare with K-Nearest Neighbor Model

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(xtrain, ytrain)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
knn_pred = knn.predict(xtest)
knn_pred
```

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

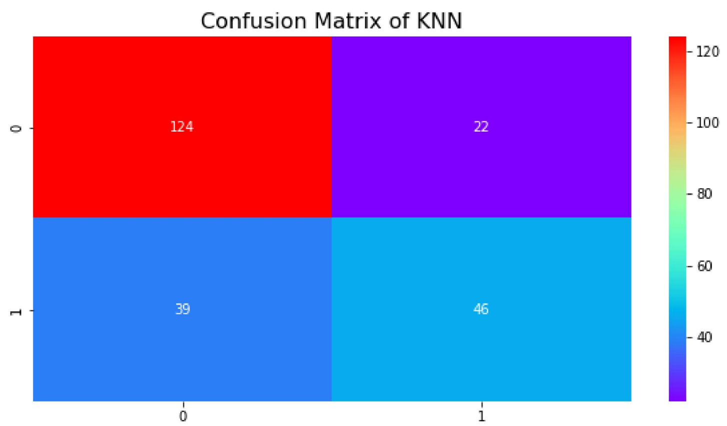
```
accuracy_score(ytest, knn_pred)
```

```
0.7359307359307359
```

```
# Confusion Matrix of KNN
knn_cf = confusion_matrix(ytest, knn_pred)
knn_cf
```

```
array([[124,  22],
       [ 39,  46]])
```

```
plt.figure(figsize=(10,5))
sns.heatmap(knn_cf, cmap = 'rainbow', annot = True, fmt = 'g')
plt.title('Confusion Matrix of KNN', fontsize = 16 )
plt.show()
```



```
# Classification Report of KNN
print(classification_report(ytest, knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       146
           1       0.68      0.54      0.60        85

    accuracy                           0.74       231
   macro avg       0.72      0.70      0.70       231
weighted avg       0.73      0.74      0.73       231
```

✓ 0s completed at 3:45 PM