KNN AND NAVIE BAIYES

2148059

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Import the Dataset

In [3]:
```
#READING DATASET
df=pd.read_csv("/Users/persie/Downloads/diabetes.csv")
```

In [5]:
```
df.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFun
0	6	148	72	35	0	33.6	(
1	1	85	66	29	0	26.6	(
2	8	183	64	0	0	23.3	(
3	1	89	66	23	94	28.1	(
4	0	137	40	35	168	43.1	:

Exploratory Data Analysis

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [7]:
```
df.describe()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Di
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [8]:
```
df.isna().sum()
```

Out[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

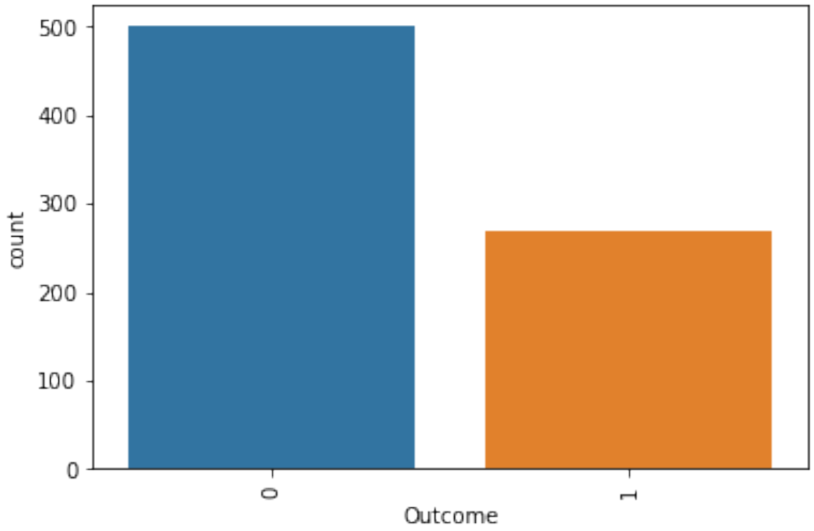
No Null Values

In [9]:
```
df['Outcome'].value_counts()
```

Out[9]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [10]:
```
sns.countplot(x = df['Outcome'], data = df)
plt.xticks(rotation = 90)
```

Out[10]:
```
(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])
```



The target is imbalanced

In [11]:
```
# separate x and y from the dataset
X = df.drop("Outcome", axis = 1)
y = df.Outcome.values
```

In [12]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=2)
X_new, y_new = sm.fit_resample(X, y.ravel())
```

In [13]:
```
print(len(y_new))
print(len(X_new))
```

```
1000
1000
```

In [14]:
```
unique, counts = np.unique(y_new, return_counts=True)
dict(zip(unique, counts))
```

Out[14]:
```
{0: 500, 1: 500}
```

The class is balanced now

StandardScaler

In [15]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# transform data
X_sc = scaler.fit_transform(X_new)
print(X_sc)
```

```
[[ 0.60196414  0.70922357  0.14749408 ...  0.1281228   0.41056721
   1.43824553]
 [-0.89248506 -1.23707102 -0.15925271 ... -0.81868657 -0.41664108
  -0.2520288 ]
 [ 1.19974381  1.79049834 -0.26150164 ... -1.26503956  0.54543813
  -0.16306699]
 ...
 [ 1.19974381  0.74011713  0.65873874 ...  0.51699547  1.44720258
   1.0823983 ]
 [-0.29470538 -0.31026407  0.50536534 ... -0.21763017 -0.23789621
   2.06097817]
 [-1.1913749  -0.86634824  0.55648981 ...  0.26228036 -0.47817362
  -0.96372325]]
```

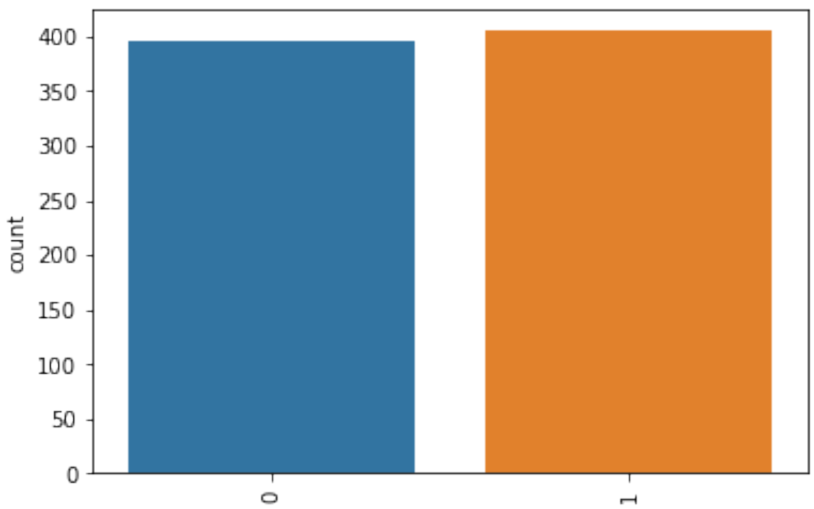
Train Test Split

In [17]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_sc, y_new, te
```

In [18]:
```
sns.countplot(y_train, data = df)
plt.xticks(rotation = 90)
```

```
/Users/persie/opt/anaconda3/lib/python3.8/site-packages/seaborn/_d
ecorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argume
nt will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[18]:
```
(array([0, 1]), [Text(0, 0, '0'), Text(1, 0, '1')])
```



KNN

In [19]:
```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
clf_knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p
clf_knn.fit(X_train, y_train)
```

Out[19]:
```
KNeighborsClassifier()
```

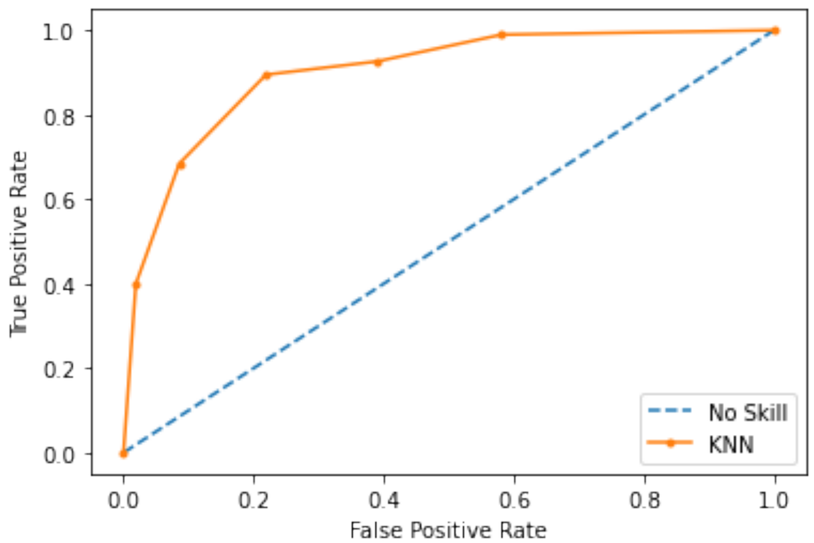
In [21]:
```
#Predicting the test set result
y_pred = clf_knn.predict(X_test)
```

In [22]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)

print('Accuracy -> '+ str(accuracy_score(y_test,y_pred)))
```

```
[[82 23]
 [10 85]]
Accuracy -> 0.835
```

In [23]:
```python
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]

# predict probabilities
lr_probs = clf_knn.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]

# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)

# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('KNN: ROC AUC=%.3f' % (lr_auc))

# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='KNN')

# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

# show the legend
plt.legend()
# show the plot
plt.show()
```

```
No Skill: ROC AUC=0.500
KNN: ROC AUC=0.901
```



NAIVE BAYES

In [24]:
```
from sklearn.naive_bayes import GaussianNB
clf_nb = GaussianNB()
clf_nb.fit(X_train,y_train)
ypred = clf_nb.predict(X_test)
```

In [25]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, class
cm = confusion_matrix(y_pred,y_test)
print(cm)
```

```
[[82 10]
 [23 85]]
```

In [26]:
```
acc = accuracy_score(y_pred,y_test)
print(acc)
```

```
0.835
```

In [27]:
```
cr = classification_report(y_pred,y_test)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.78      0.89      0.83        92
           1       0.89      0.79      0.84       108

    accuracy                           0.83       200
   macro avg       0.84      0.84      0.83       200
weighted avg       0.84      0.83      0.84       200
```

In [28]:
```python
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(y_test))]

# predict probabilities
lr_probs = clf_nb.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]

# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)

# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('KNN: ROC AUC=%.3f' % (lr_auc))

# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)

# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Naive Bayes')

# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')

# show the legend
plt.legend()
# show the plot
plt.show()
```

```
No Skill: ROC AUC=0.500
KNN: ROC AUC=0.854
```