In [95]:
```
# importing libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [96]:
```
# importing the data

diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df.head()
```

Out[96]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	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

In [97]:
```
# Exploratory Data Analysis (EDA)
```

In [98]:
```
# Total number of columns in the dataset
diabetes_df.columns
```

Out[98]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [99]:
```
# Information about the dataset
diabetes_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 [100…

```
diabetes_df.describe()
```

Out[100…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedig
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 [101…

```
# Transposing the dataset
diabetes_df.describe().T
```

Out[101…

	count	mean	std	min	25%	50%	75%	m
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.

In [102…

```
# Checking for null values
diabetes_df.isnull()
```

Out[102…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	F

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
2	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	
763	False	False	False	False	False	False	False	F
764	False	False	False	False	False	False	False	F
765	False	False	False	False	False	False	False	F
766	False	False	False	False	False	False	False	F
767	False	False	False	False	False	False	False	F

768 rows × 9 columns

In [103...]

```
# Checking for null values
diabetes_df.isnull().sum()
```

Out[103...]

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [104...]

```
# value of zero indicates missing value.

# replacing zeros with NAN
diabetes_df_copy = diabetes_df.copy(deep = True)
diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diab

# Showing the Count of NANs
print(diabetes_df_copy.isnull().sum())
```

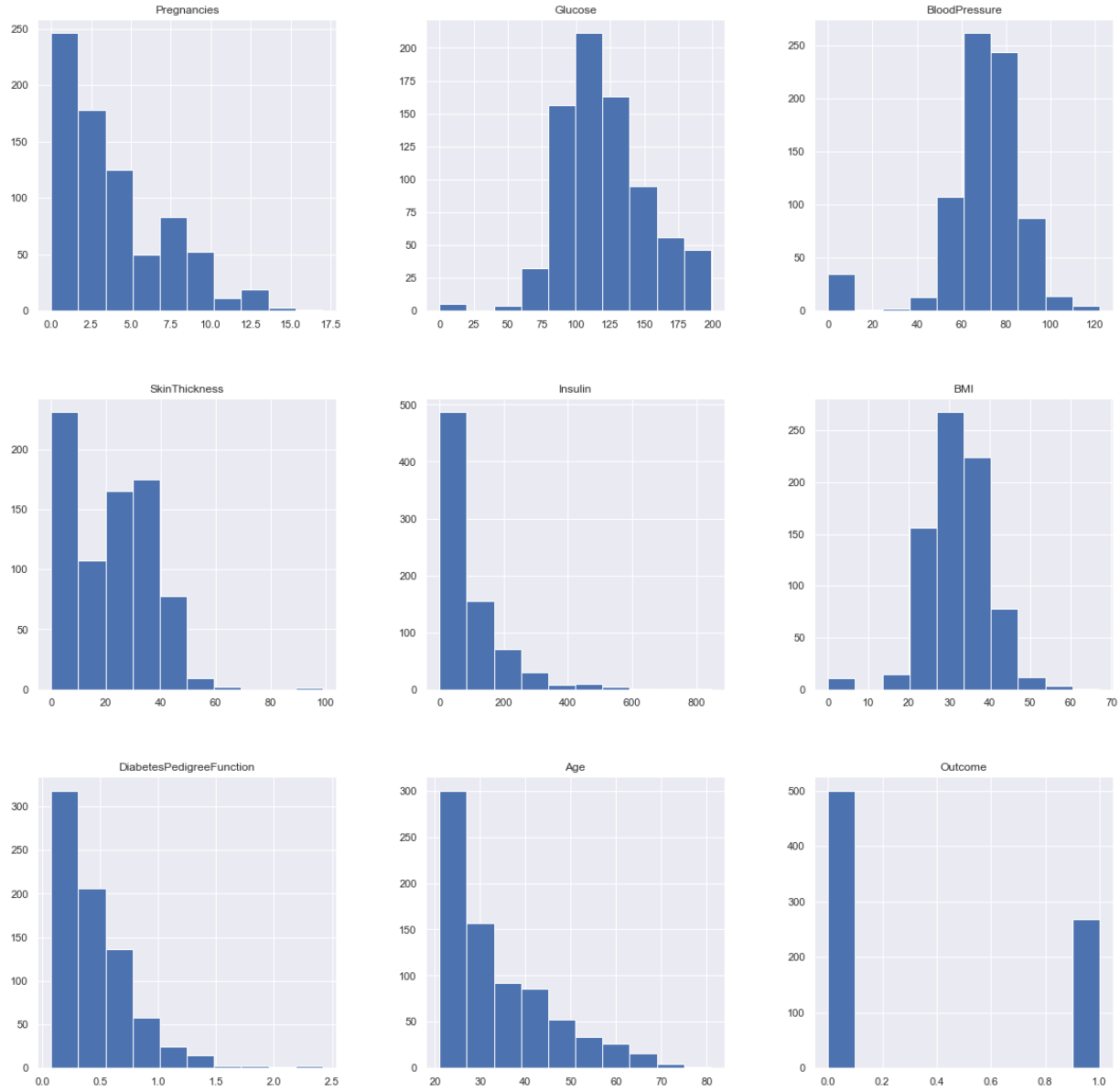
```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [105...]

```
# Data visualization
```

In [106...]

```
# Plotting the data distribution plots
p = diabetes_df.hist(figsize = (20,20))
```

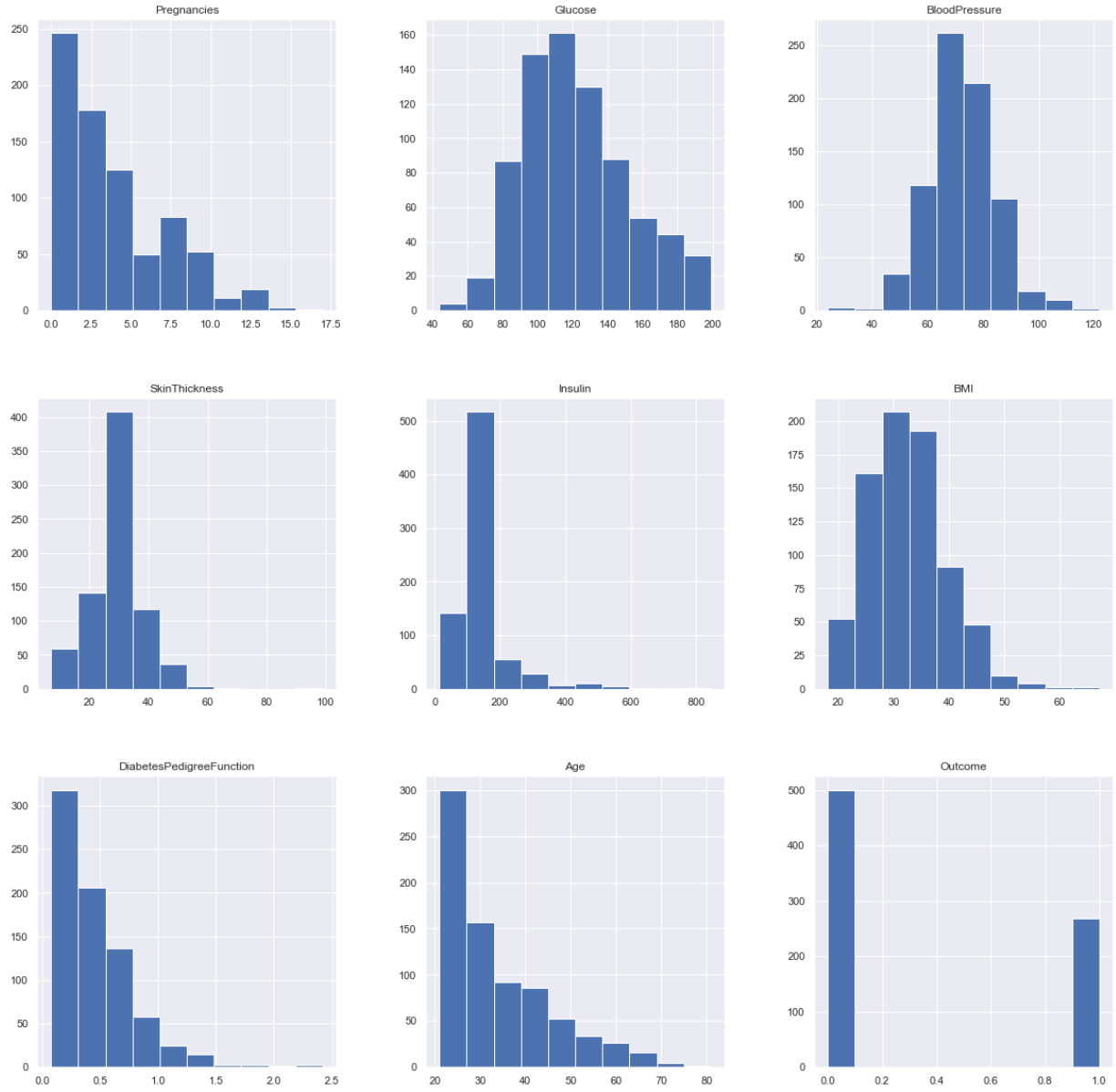


In [107…

```
# imputing NAN values for the columns in accordance with their distribution
diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glucose'].mean(), inplace = Tru
diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['BloodPressure'].mean(), i
diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['SkinThickness'].median(),
diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].median(), inplace = T
diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(), inplace = True)
```

In [108…

```
# Plotting the distributions after removing the NAN values
p = diabetes_df_copy.hist(figsize = (20,20))
```



In [ ]:
```
# Checking the balance of the data by plotting the count of outcomes by their value
color_wheel = {1: "#0392cf", 2: "#7bc043"}
colors = diabetes_df["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_df.Outcome.value_counts())
p=diabetes_df.Outcome.value_counts().plot(kind="bar")
```

In [ ]:
```
# Plotting a Scatter matrix of uncleaned data
p=scatter_matrix(diabetes_df,figsize=(20,20))
```

In [ ]:
```
# Plotting the Pair Plots for the data
p=sns.pairplot(diabetes_df_copy, hue = 'Outcome')
```

In [ ]:
```
# Correlation between all the features
```

In [ ]:
```
# Correlstion between all the features before cleaning
plt.figure(figsize=(12,10))
p = sns.heatmap(diabetes_df.corr(), annot=True,cmap ='RdYlGn')
```

In [ ]:
```
# Correlstion between all the features after cleaning
plt.figure(figsize=(12,10))
p=sns.heatmap(diabetes_df_copy.corr(), annot=True,cmap ='RdYlGn')  # seaborn has ver
```

In [110…
```
# Scaling the data
```

In [127…
```
diabetes_df_copy.head()
```

Out[127…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33

In [128…
```
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetes_df_copy.drop(["Outcome"],axis = 1),),
X.head()
```

Out[128…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484

In [129…
```
y = diabetes_df_copy.Outcome
y
```

Out[129…
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

In [130…
```
# Splitting the data into Train and Test

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/3,random_state=42,
```

In [131…

```
# Model Building - K-Nearest Neighbor (KNN)
test_scores = []
train_scores = []

for i in range(1,15):

    knn = KNeighborsClassifier(i)
    knn.fit(X_train,y_train)

    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(X_test,y_test))
```

In [132…

```
max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score]
print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambda x
```

```
Max train score 100.0 % and k = [1]
```

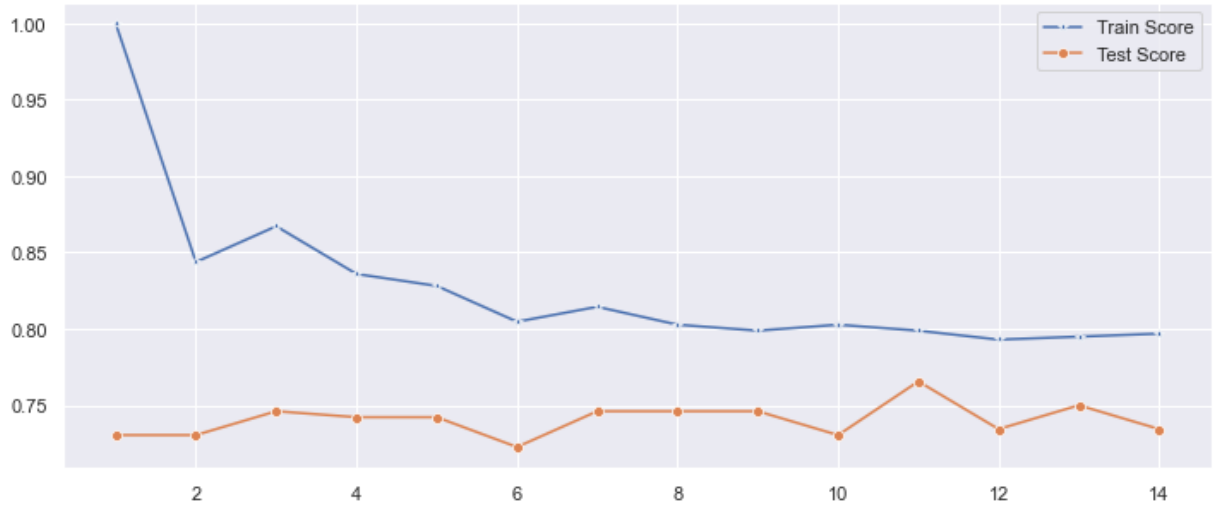
In [133…

```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x:
```

```
Max test score 76.5625 % and k = [11]
```

In [134…

```
plt.figure(figsize=(12,5))
p = sns.lineplot(range(1,15),train_scores,marker='*',label='Train Score')
p = sns.lineplot(range(1,15),test_scores,marker='o',label='Test Score')
```



In [135…

```
# The best result is captured at k = 11 hence 11 is used for the final model

knn = KNeighborsClassifier(11)
knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

Out[135…

```
0.765625
```

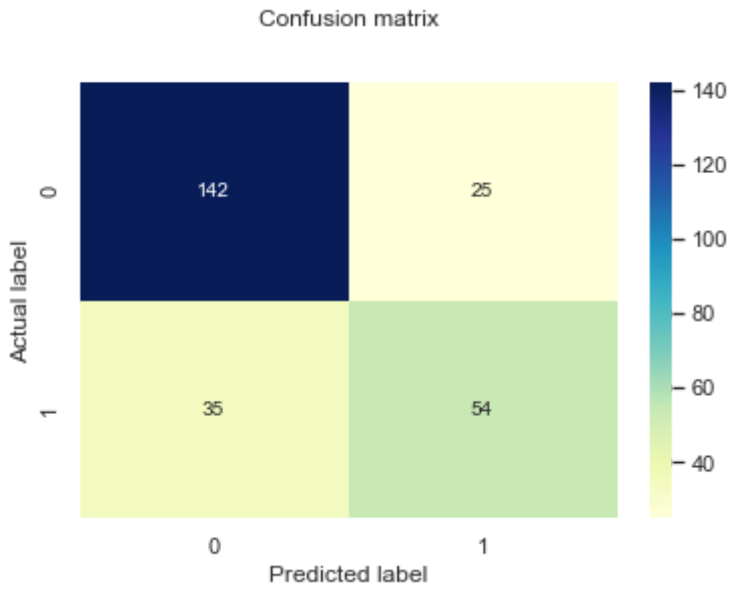
In [137…

```
# Comfusion matrix

y_pred = knn.predict(X_test)
```

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[137…

```
Text(0.5, 12.5, 'Predicted label')
```



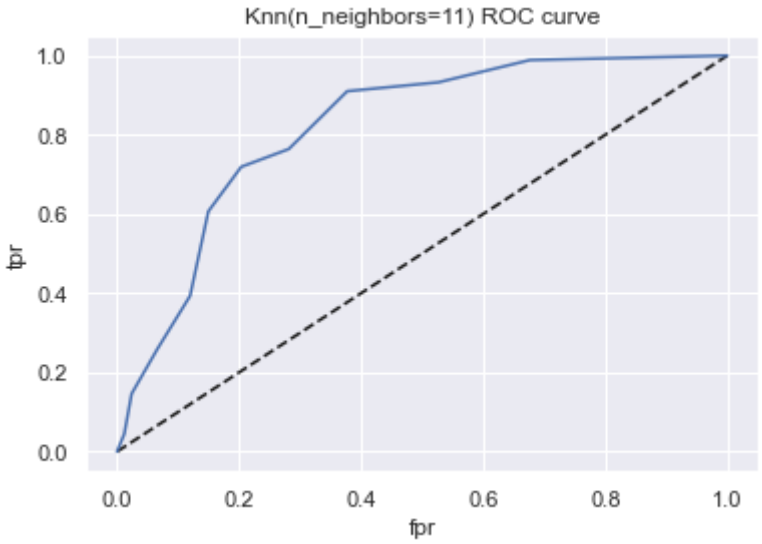
In [138…

```
# Classification Reports
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.83       167
           1       0.68      0.61      0.64        89

    accuracy                           0.77       256
   macro avg       0.74      0.73      0.73       256
weighted avg       0.76      0.77      0.76       256
```

In [139…

```
# ROC-AUC Curve

y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=11) ROC curve')
plt.show()
plt.savefig('Prediction of diabetes using KNN.png')
```




```
<Figure size 432x288 with 0 Axes>
```

In [124…

```
roc_auc_score(y_test,y_pred_proba)
```

Out[124…

```
0.8193500639171096
```

In [125…

```
# Implementing GridSearchCV

param_grid = {'n_neighbors':np.arange(1,50)}
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X,y)

print("Best Score:" + str(knn_cv.best_score_))
print("Best Parameters: " + str(knn_cv.best_params_))
```

```
Best Score:0.7721840251252015
Best Parameters: {'n_neighbors': 25}
```

In [ ]:

In [ ]: