```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

os.chdir('C:\\Users\\utpala mohapatra\\Documents\\python_folder\\python datasets')
```

Load the dataset

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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

Task 1 - Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value.

```
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
```

check for null values

```
diabetes_df.isna().sum()
```

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

No null values present in the database.

Descriptive Analysis

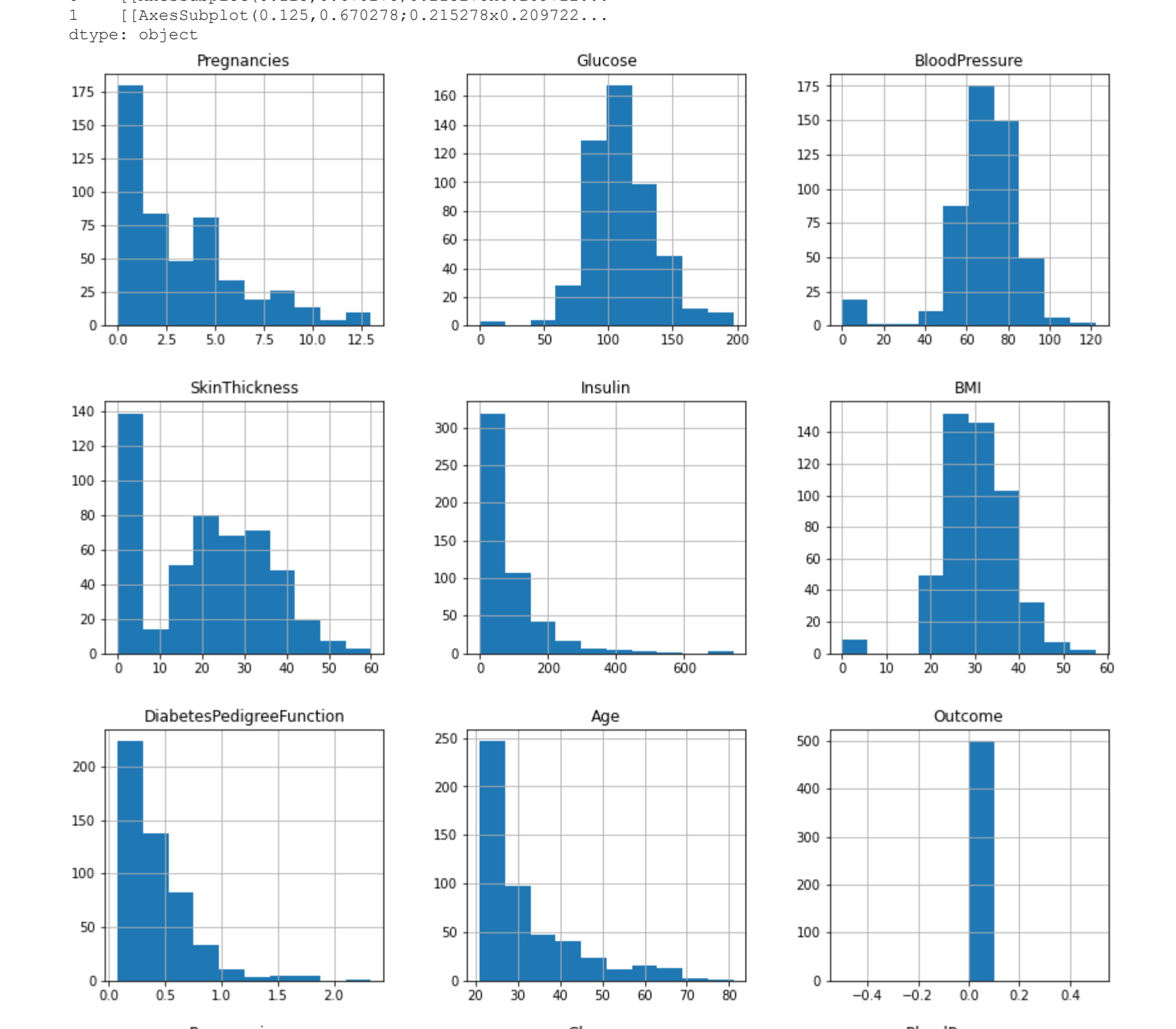
```
diabetes_df.describe()
```

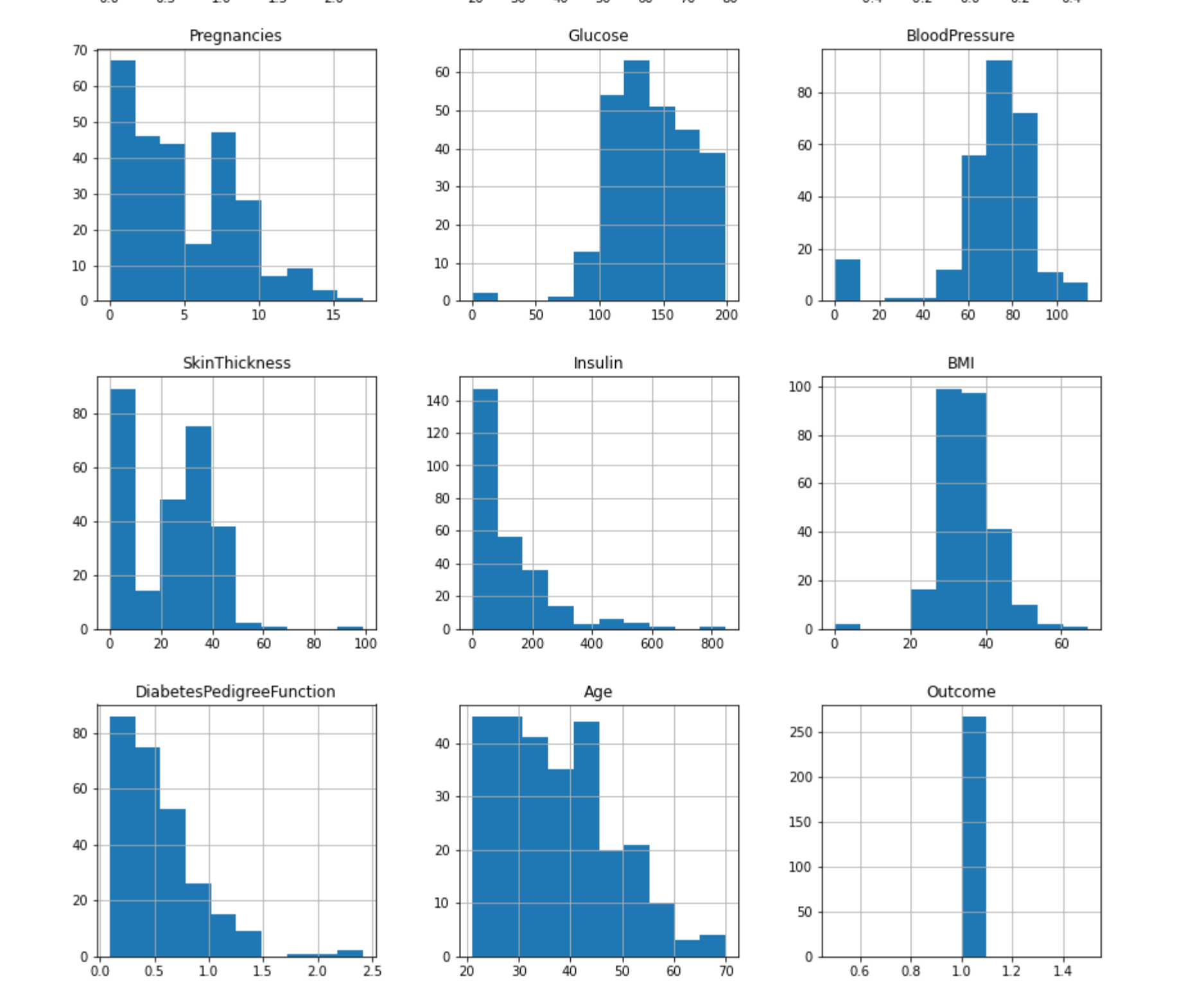
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Task 2 - Visually explore these variables using histograms. Treat the missing values accordingly.

```
diabetes_df.groupby('Outcome').hist(figsize = (14,13))
```

```
Outcome
0    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
1    [[AxesSubplot(0.125,0.670278;0.215278x0.209722...
dtype: object
```





Find out which columns have 0 values as 0 values means missing values and then Impute the missing Values.

```
(diabetes_df==0).sum()
```

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

```
# convert the zero values to missing values
cols = ['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI']
diabetes_df[cols]= diabetes_df[cols].replace(0,np.nan)
diabetes_df.isna().sum()
```

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

```
# Impute the missing values with the help of SimpleImputer
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
diabetes_df[cols]=imputer.fit_transform(diabetes_df[cols])
diabetes_df.isna().sum()
```

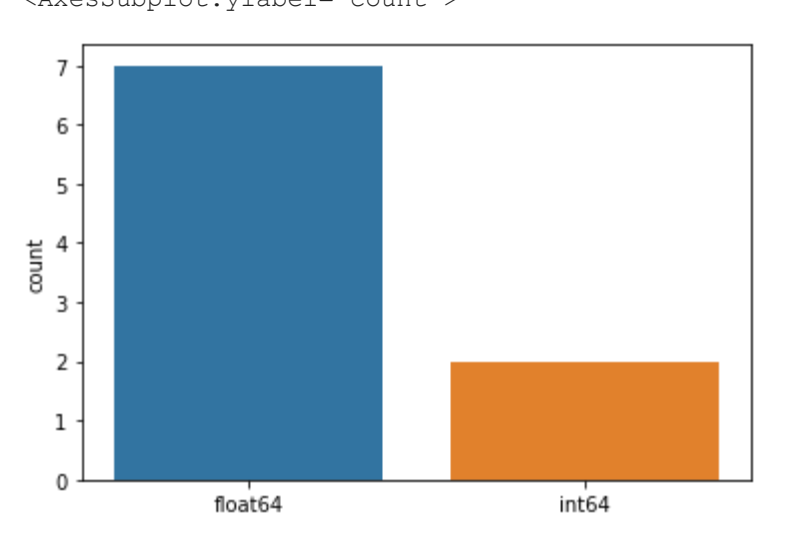
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

outcome is our target variable so no need to remove zeros

Task 3 - There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

```
sns.countplot(diabetes_df.dtypes.map(str))
```

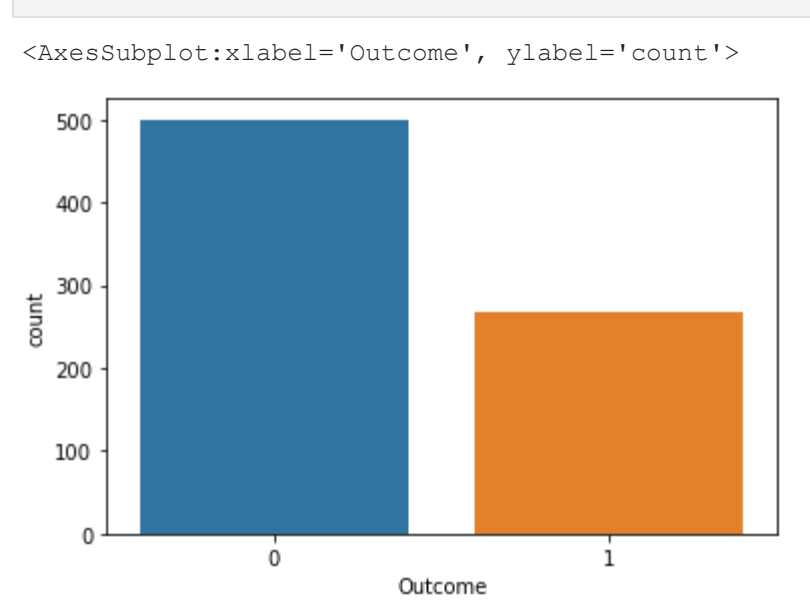
```
<AxesSubplot:ylabel='count'>
```



Task 4 - Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

```
sns.countplot(x='Outcome',data=diabetes_df)
```

```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```

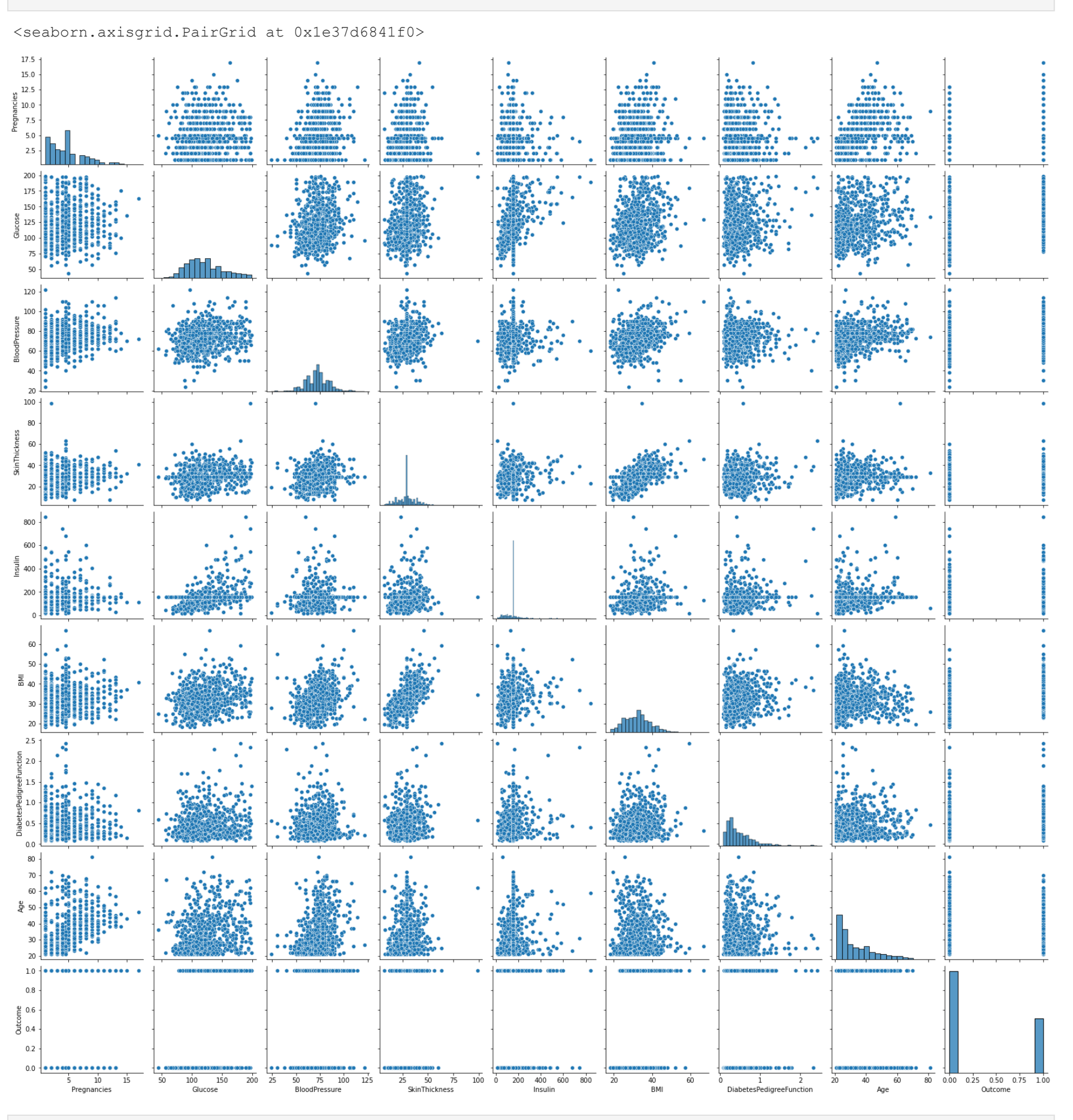


From the bar plot above we can see as number of Zero outcomes is more so as compared to diabetic people nondiabetic people are more.

Task 5 - Create scatter charts between the pair of variables to understand the relationships. Describe your findings.

```
sns.pairplot(diabetes_df)
```

```
<seaborn.axisgrid.PairGrid at 0x1e37d6841f0>
```



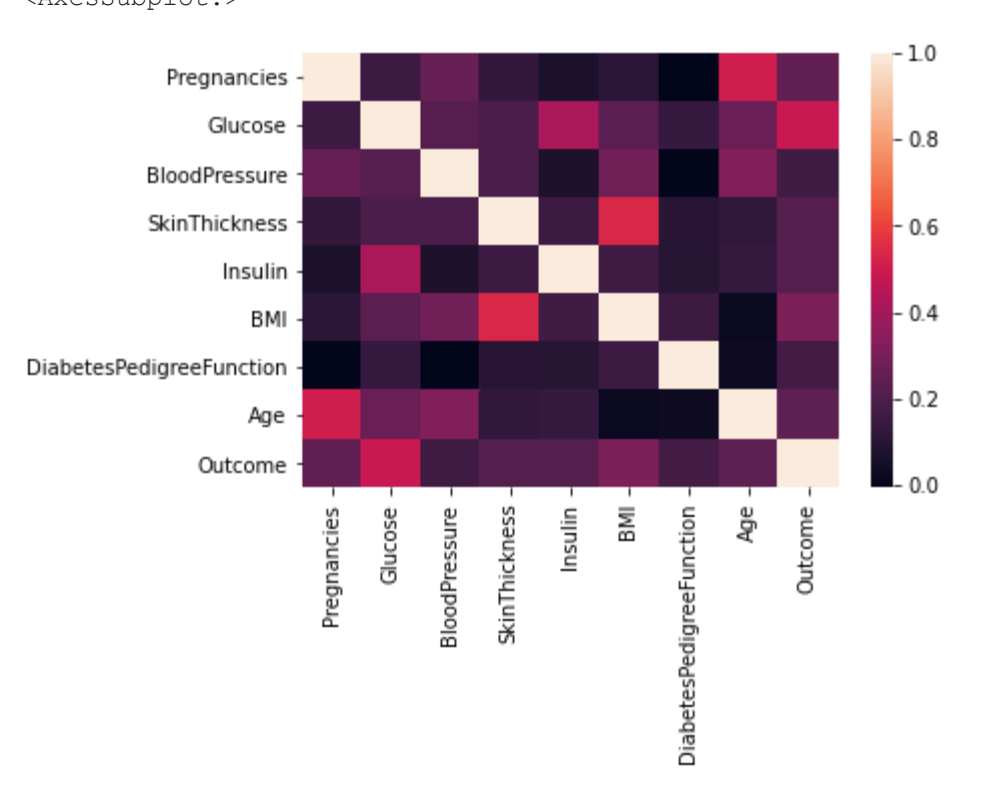
```
diabetes_df.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcom
Pregnancies	1.000000	0.154290	0.259117	0.131819	0.068077	0.110590	-0.005658	0.511662	0.24826
Glucose	0.154290	1.000000	0.218367	0.192991	0.420157	0.230941	0.137060	0.266534	0.49292
BloodPressure	0.259117	0.218367	1.000000	0.192816	0.072517	0.281268	-0.002763	0.324595	0.16607
SkinThickness	0.131819	0.192991	0.192816	1.000000	0.158139	0.542398	0.100966	0.127872	0.21529
Insulin	0.068077	0.420157	0.072517	0.158139	1.000000	0.166586	0.098634	0.136734	0.21441
BMI	0.110590	0.230941	0.281268	0.542398	0.166586	1.000000	0.153400	0.025519	0.31192
DiabetesPedigreeFunction	-0.005658	0.137060	-0.002763	0.100966	0.098634	0.153400	1.000000	0.033561	0.17384
Age	0.511662	0.266534	0.324595	0.127872	0.136734	0.025519	0.033561	1.000000	0.23835
Outcome	0.248263	0.492928	0.166074	0.215299	0.214411	0.311924	0.173844	0.238356	1.00000

Task 6 - Perform correlation analysis. Visually explore it using a heat map.

```
sns.heatmap(diabetes_df.corr())
```

```
<AxesSubplot:>
```



From the scatter plot and the heatmap we can visualize that outcome of Diabetic data is more correlated with the Glucose level.

Task 7 - Apply an appropriate classification algorithm to build a model.

Model 1 - As the target variable that is Outcome values are bianary,so the best possible algorithm should be Logistic Regression.

```
# Convert the data into feature and target variables.
X = diabetes_df.drop(['Outcome'],axis=1)
y = diabetes_df['Outcome']
```

```
# Convert the data to train and test data
from sklearn.model_selection import train_test_split,cross_val_score,KFold
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix,roc_auc_score,roc_curve
xtrain,xtest,ytrain,ytest = train_test_split(X,y,test_size=0.3,random_state=23)
print(xtrain.shape,ytrain.shape,xtest.shape,ytest.shape)
```

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

```
from sklearn.linear_model import LogisticRegression
lrg = LogisticRegression()
lrg.fit(xtrain,ytrain)
```

```
LogisticRegression()
```

```
lrg_predict = lrg.predict(xtest)
accuracy_score(ytest,lrg_predict)
```

```
0.7792207792207793
```

The accuracy score in Logistic Regression is 78%.

Task 8 - Compare various models with the results from KNN algorithm.

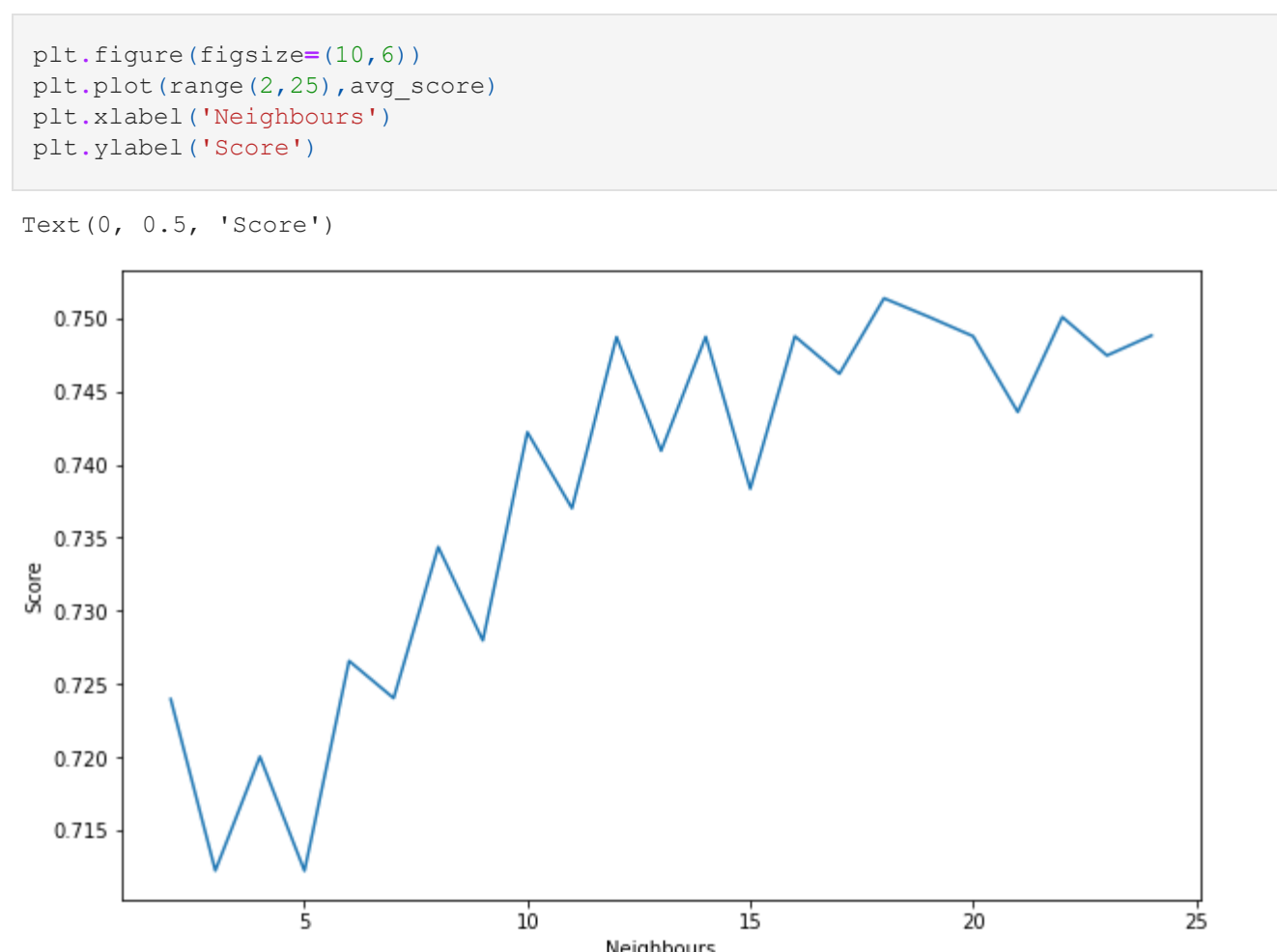
Model 2 - Lets try KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier

kfold = KFold(n_splits=10)
avg_score = []
for k in range(2,25):
    knn = KNeighborsClassifier(n_neighbors=k,n_jobs=-1)
    score = cross_val_score(knn,X,y,cv=kfold,scoring='accuracy')
    avg_score.append(score.mean())
```

```
plt.figure(figsize=(10,6))
plt.plot(range(2,25),avg_score)
plt.xlabel('Neighbours')
plt.ylabel('Score')
```

```
Text(0, 0.5, 'Score')
```



We can see in th graph The KNN model gives the best score with the neighbours=17

```
knn_17 = KNeighborsClassifier(n_neighbors=17,n_jobs=-1)
knn_17.fit(xtrain,ytrain)
knn_pred = knn_17.predict(xtest)
print(accuracy_score(ytest,knn_pred))
```

```
0.7402597402597403
```

The accuracy score in the knn model is 74%

Model 3 - Lets try with Decision Tree Classifier.

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(xtrain,ytrain)
tree_pred = tree.predict(xtest)
print(accuracy_score(ytest,tree_pred))
```

```
0.6753246753246753
```

The accuracy score in DecisionTree model is just 67%.

Model 4 - Lets try RandomforestClassifier

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(xtrain,ytrain)
rfc_pred = rfc.predict(xtest)
print(accuracy_score(ytest,rfc_pred))
```

```
0.7575757575757576
```

The accuracy Score in Randomforest Classifier is 75%.

Lets try AdaBoost Classifier.

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100)
ada.fit(xtrain,ytrain)
ada_pred = ada.predict(xtest)
accuracy_score(ytest,ada_pred)
```

```
0.7316017316017316
```

In AdaBoost model also the Accracy score is 73%

From all these models we can see RandomforestClassifier is giving better Accuracy than KNN model.

Task 9 - Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc.

```
conf_matx =(confusion_matrix(ytest,lrg_predict))
print(conf_matx)
```

```
[[130  18]
 [ 33  50]]
```

```
TP = conf_matx[1][1]
TN = conf_matx[0][0]
FP = conf_matx[0][1]
FN = conf_matx[1][0]
```

```
sensitivity = TP/(TP+FN)
specificity = TN/(TN+FP)
print('Sensitivity: ',sensitivity,'\n','Specificity: ',specificity)
```

```
Sensitivity:  0.6024096385542169
 Specificity:  0.8783783783783784
```

The sencitivity is 60% and the specificity is 88%.

```
print(classification_report(ytest,lrg_predict))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       148
           1       0.74      0.60      0.66        83

    accuracy                           0.78       231
   macro avg       0.77      0.74      0.75       231
weighted avg       0.78      0.78      0.77       231
```

```
pred_proba = lrg.predict_proba(xtest)[:,1]
```

```
false_positive_rate,true_positive_rate,threshold = roc_curve(ytest,pred_proba)
print('roc_auc_score for logistic regression: ',roc_auc_score(ytest,pred_proba))
```

```
roc_auc_score for logistic regression:  0.8371865841745358
```

```
plt.figure(figsize=(10,6))
plt.title('ROC Curve for  Logistic regression')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".5"), plt.plot([1, 1] , c=".5")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```