In [390]:
```
'''
DESCRIPTION

Problem Statement
* NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments
for the most chronic, costly, and consequential diseases.
* The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes,
based on certain diagnostic measurements included in the dataset.
* Build a model to accurately predict whether the patients in the dataset have diabetes or not.
'''
```

Out[390]:
```
'\nDESCRIPTION\n\nProblem Statement\n* NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatment
s \nfor the most chronic, costly, and consequential diseases.\n* The dataset used in this project is originally from NIDDK. The objective is to predict whethe
r or not a patient has diabetes, \nbased on certain diagnostic measurements included in the dataset.\n* Build a model to accurately predict whether the patien
ts in the dataset have diabetes or not.\n'
```

In [391]:
```python
#Importing required libraries and dataset
import pandas as pd
diabetes_df = pd.read_csv("E:/Education/PGP Simplilearn-Purdue/PGP in Data Science\Data Science Capstone/Data-Science-Capstone-Projects-master/Project 2/Health
diabetes_df
```

Out[391]:

	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
...	...	...	...	...	...	...	...	...	...
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

768 rows × 9 columns

In [392]:
```
'''
Data Exploration:

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:

• Glucose
• BloodPressure
• SkinThickness
• Insulin
• BMI

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

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.
'''
```

Out[392]:
```
'\nData Exploration:\n\n1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, \na value of zero does not make sense and thus indicates missing value:\n\n• Glucose\n• BloodPressure\n• SkinThickness\n• Insulin\n• BMI\n\n2. Visually explore these variables using histograms. Treat the missing values accordingly.\n\n3. There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the \ndata types and the count of variables. \n'
```

In [393]:
```
#Data Preprocessing
#Exploring the dataset
diabetes_df.info()
#768 rows x 9 columns
```

```
<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 [394]:
```
#Checking for NULL values
diabetes_df.isna().sum()
# Observations:
# No NULL values are present in the dataset
```

Out[394]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [395]:
```
#Checking for duplicates
diabetes_df[diabetes_df.duplicated()]
# Observations:
# No duplicate records are present in the dataset
```

Out[395]:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome

In [396]:
```
#Statistical Overview
diabetes_df.describe()
```

Out[396]:

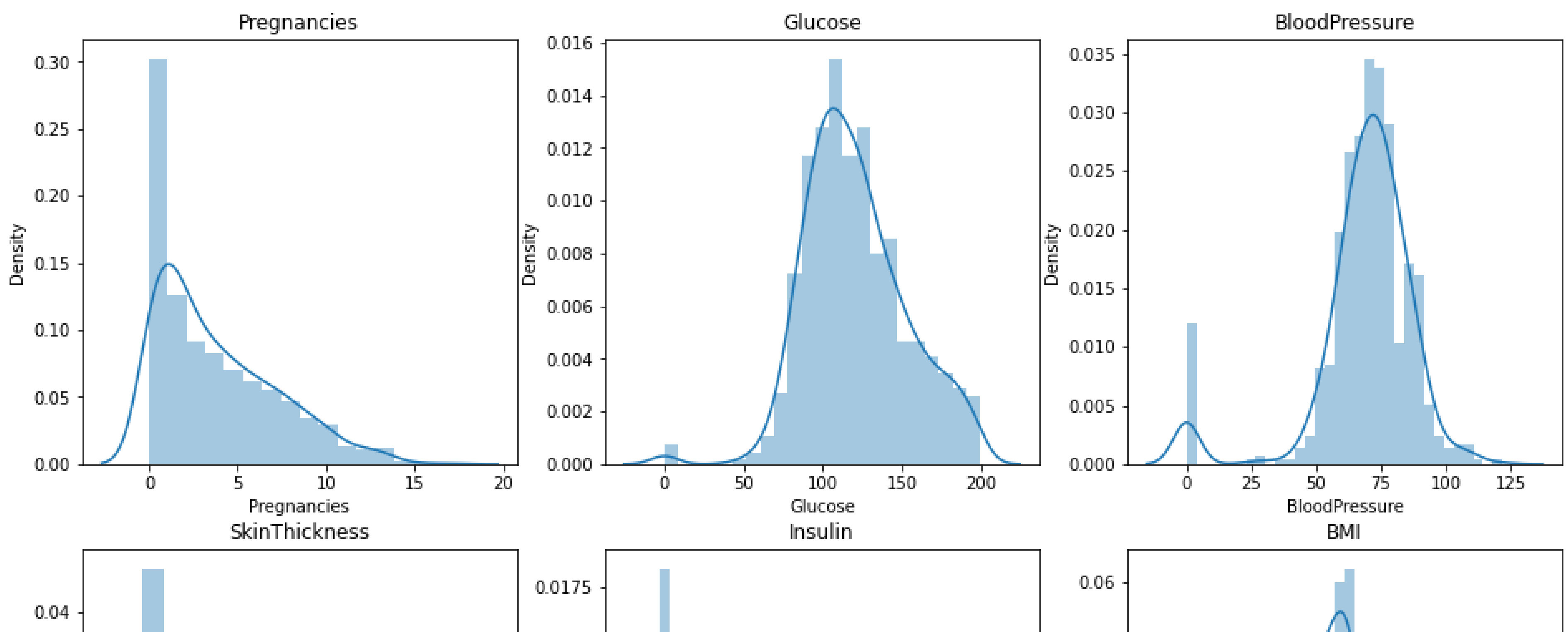
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	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

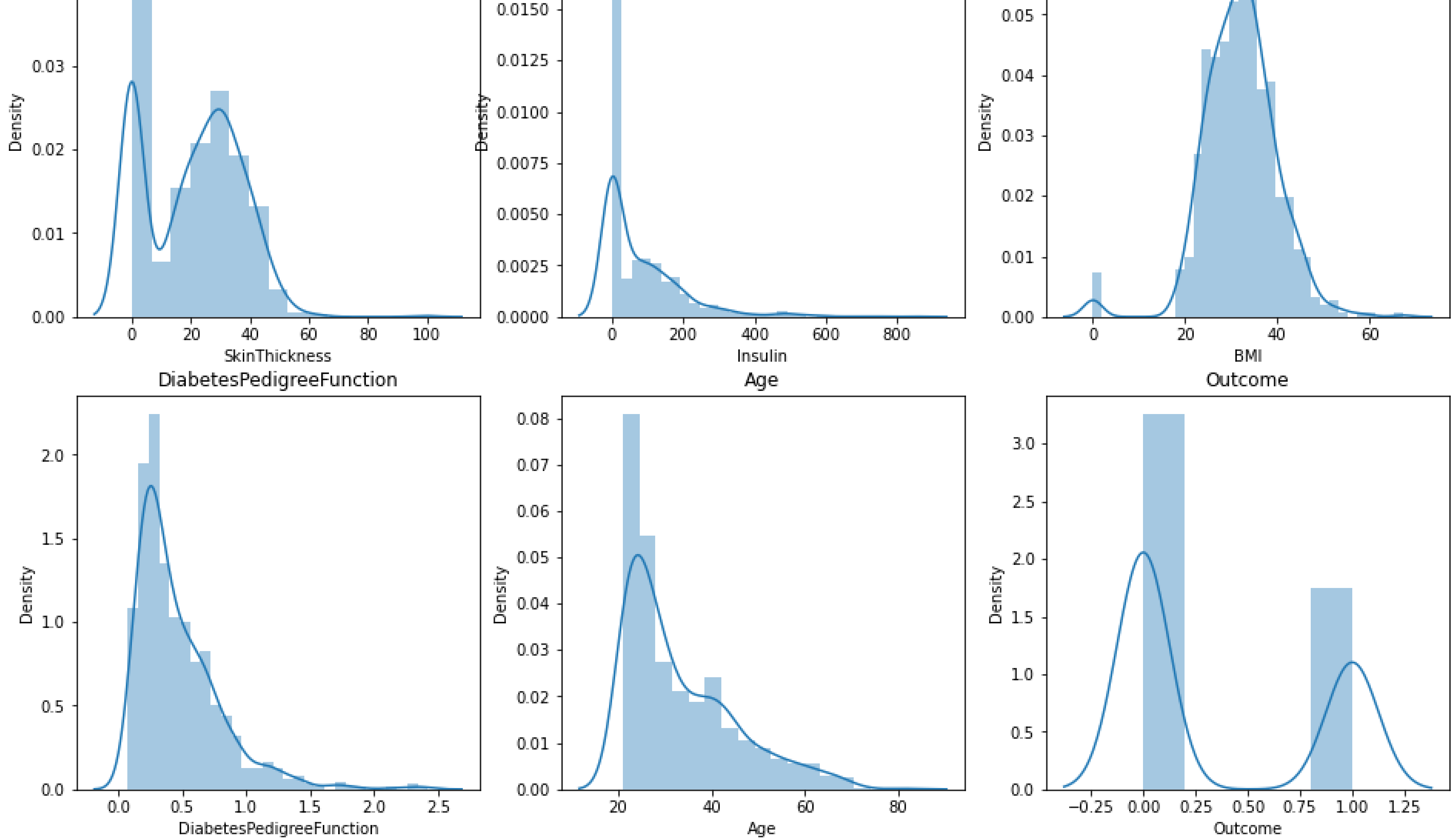
In [397]:
```python
'''
1. Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below,
a value of zero does not make sense and thus indicates missing value:

• Glucose
• BloodPressure
• SkinThickness
• Insulin
• BMI
'''
#Checking for zero values in above specified columns
(diabetes_df==0).sum()
# Observations:
# Zero values in the above mentioned columns are similar to NULL values whereas zeros in other columns make sense
```

Out[397]:
```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

In [398]:

```python
# 2. Visually explore these variables using histograms. Treat the missing values accordingly.
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")

#Histograms
fig = plt.figure(figsize=(16,16))
for i in range(len(diabetes_df.columns)):
    plt.subplot(3,3,i+1)
    sns.distplot(diabetes_df.iloc[:,i],kde=True)
    plt.title(diabetes_df.columns[i])
plt.show()

# Observations:
# 1. Pregnancies, Insulin, Diabetes pedigree function and Age have right-skewed distributions
# 2. Glucose, Blood Pressure, Skin Thickness and BMI are approximately normally distributed (ignoring zero values)
# 3. We can either remove missing values or impute the missing values with any measure of central tendency (mean , median or mode)
# depending on the datatype
# 4. Since we have a small dataset, It doesn't make sense to remove rows with zero values
# 5. Also, We can impute with either mean or median as we have numerical columns
```




Density
SkinThickness
Density
Insulin
Density
BMI
DiabetesPedigreeFunction
Density
DiabetesPedigreeFunction
Age
Density
Age
Outcome
Density
Outcome

In [399]:
```
#Splitting the dataset into train and test sets before imputing to prevent data leakage
#Splitting the data into x and y variables
x = diabetes_df.drop(['Outcome'],axis=1)
y = diabetes_df['Outcome']

print(diabetes_df.shape)
print(x.shape)
print(y.shape)

#Splitting into train and test sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25,random_state=10)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(768, 9)
(768, 8)
(768,)
(576, 8)
(192, 8)
(576,)
(192,)
```

In [400]:
```
#Imputing missing values with median
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=0,strategy='median')
x_train2 = imputer.fit_transform(x_train.iloc[:,1:6])
x_test2 = imputer.transform(x_test.iloc[:,1:6])
```

In [401]:
```
#Converting arrays into data frames
x_train2 = pd.DataFrame(x_train2)
x_test2 = pd.DataFrame(x_test2)

print(x_train2)
print(x_test2)
```

```
         0      1     2      3     4
0    171.0   72.0  29.0  122.0  43.6
1    108.0   44.0  20.0  130.0  24.0
2    196.0   90.0  29.0  122.0  39.8
3    134.0   70.0  29.0  122.0  28.9
4    117.0   96.0  29.0  122.0  28.7
..     ...    ...   ...    ...   ...
571  133.0  102.0  28.0  140.0  32.8
572  129.0   60.0  12.0  231.0  27.5
573  116.0   74.0  15.0  105.0  26.3
574   88.0   30.0  42.0   99.0  55.0
575   96.0   74.0  18.0   67.0  33.6

[576 rows x 5 columns]
         0     1     2      3     4
0    154.0  72.0  29.0  126.0  31.3
1    112.0  86.0  42.0  160.0  38.4
2    135.0  54.0  29.0  122.0  26.7
3    107.0  62.0  13.0   48.0  22.9
4    102.0  74.0  29.0  122.0  29.5
..     ...   ...   ...    ...   ...
187  105.0  80.0  28.0  122.0  32.5
188   87.0  68.0  34.0   77.0  37.6
189  103.0  66.0  29.0  122.0  24.3
190  143.0  74.0  22.0   61.0  26.2
191  173.0  78.0  32.0  265.0  46.5

[192 rows x 5 columns]
```

In [402]:
```python
#Replacing with imputed values in the original data
for i in range(5):
    x_train.iloc[:,i+1] = x_train2.iloc[:,i].values
    x_test.iloc[:,i+1] = x_test2.iloc[:,i].values

print(x_train)
print(x_test)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
235            4    171.0           72.0           29.0    122.0  43.6
576            6    108.0           44.0           20.0    130.0  24.0
22             7    196.0           90.0           29.0    122.0  39.8
451            2    134.0           70.0           29.0    122.0  28.9
616            6    117.0           96.0           29.0    122.0  28.7
..           ...      ...            ...            ...      ...   ...
369            1    133.0          102.0           28.0    140.0  32.8
320            4    129.0           60.0           12.0    231.0  27.5
527            3    116.0           74.0           15.0    105.0  26.3
125            1     88.0           30.0           42.0     99.0  55.0
265            5     96.0           74.0           18.0     67.0  33.6

     DiabetesPedigreeFunction  Age
235                     0.479   26
576                     0.813   35
22                      0.451   41
451                     0.542   23
616                     0.157   30
..                        ...  ...
369                     0.234   45
320                     0.527   31
527                     0.107   24
125                     0.496   26
265                     0.997   43

[576 rows x 8 columns]
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
568            4    154.0           72.0           29.0    126.0  31.3
620            2    112.0           86.0           42.0    160.0  38.4
456            1    135.0           54.0           29.0    122.0  26.7
197            3    107.0           62.0           13.0     48.0  22.9
714            3    102.0           74.0           29.0    122.0  29.5
..           ...      ...            ...            ...      ...   ...
613            6    105.0           80.0           28.0    122.0  32.5
562            1     87.0           68.0           34.0     77.0  37.6
587            6    103.0           66.0           29.0    122.0  24.3
413            1    143.0           74.0           22.0     61.0  26.2
487            0    173.0           78.0           32.0    265.0  46.5
```

```
     DiabetesPedigreeFunction  Age
568                     0.338   37
620                     0.246   28
456                     0.687   62
197                     0.678   23
714                     0.121   32
..                        ...  ...
613                     0.878   26
562                     0.401   24
587                     0.249   29
413                     0.256   21
487                     1.159   58

[192 rows x 8 columns]
```

In [403]:

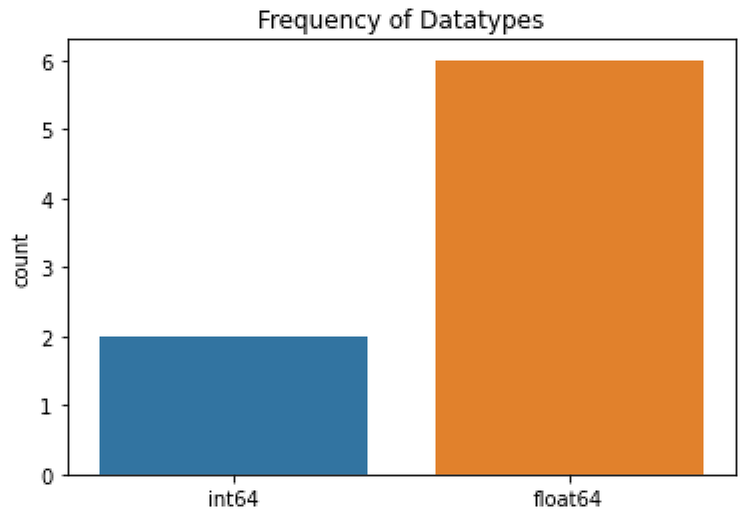
```
#Checking for NULL values
print(x_train.isna().sum())
print(x_test.isna().sum())
# Observation:
# The original dataset has zero NULL values after imputing
```

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

In [404]:
```
# 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.
print(x_train.dtypes.value_counts())
sns.countplot(x_train.dtypes.map(str))
plt.title("Frequency of Datatypes")
```

```
float64    6
int64      2
dtype: int64
```

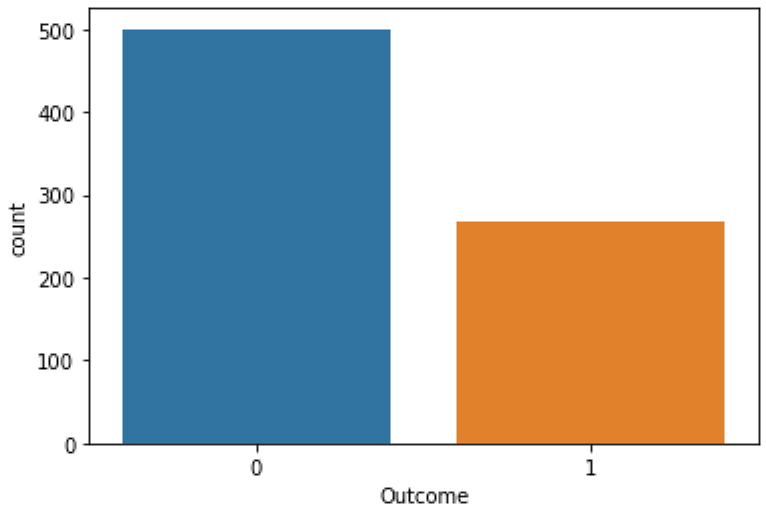
Out[404]:
```
Text(0.5, 1.0, 'Frequency of Datatypes')
```



In [405]:
```
# Check the balance of the data by plotting the count of outcomes by their value.
# Describe your findings and plan future course of action.
print(diabetes_df['Outcome'].value_counts())
sns.countplot(diabetes_df['Outcome'])
# Observations:
# We can clearly see that we have an unbalanced dataset which might bias the predictions towards the 'No diabetes' Category
# We can use techniques like undersampling the majority, oversampling the minority classes or change class weights according
# to the category
#Since, we have decent amount (~35%) of 'Diabetes' category, We can proceed to model building
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

Out[405]:
```
<AxesSubplot:xlabel='Outcome', ylabel='count'>
```

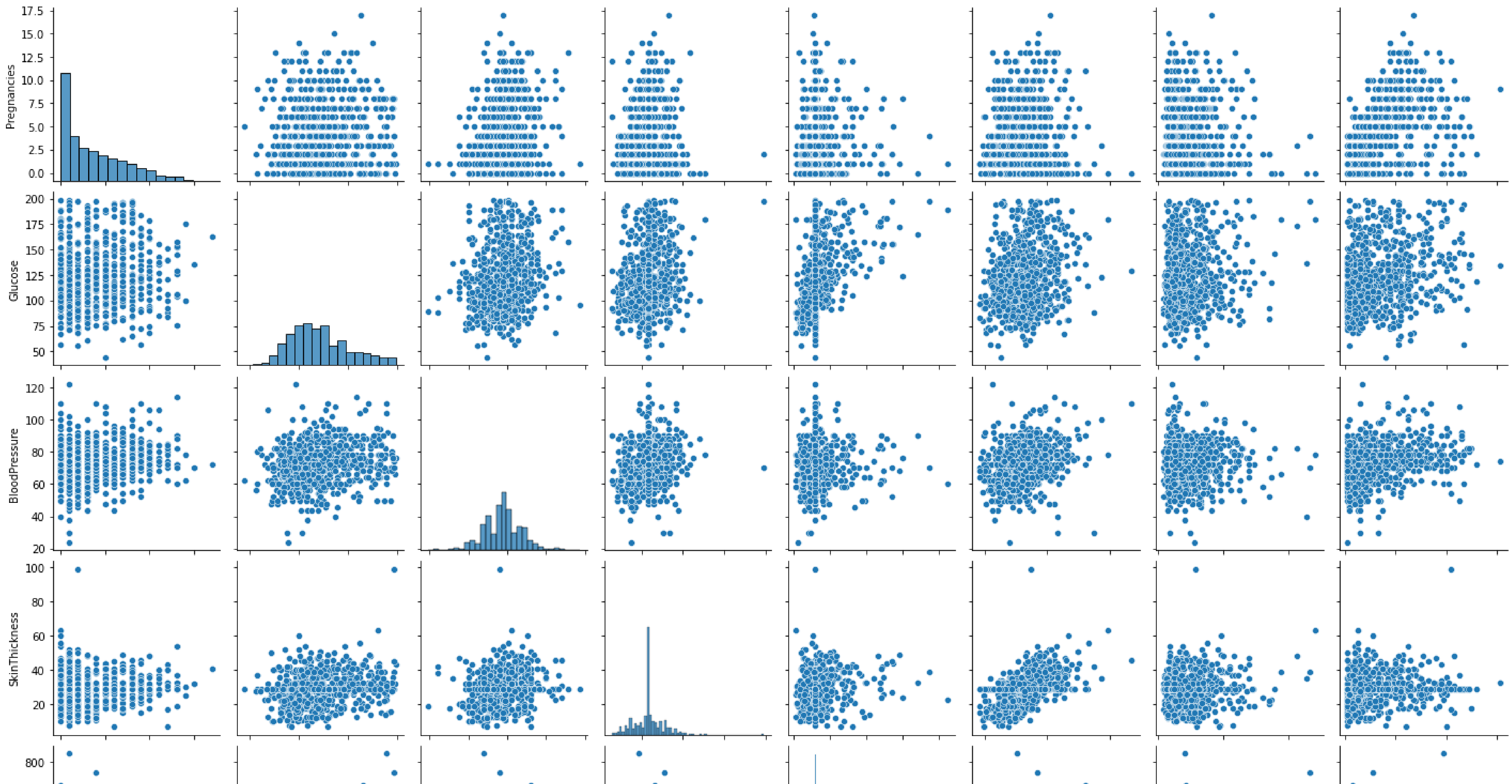


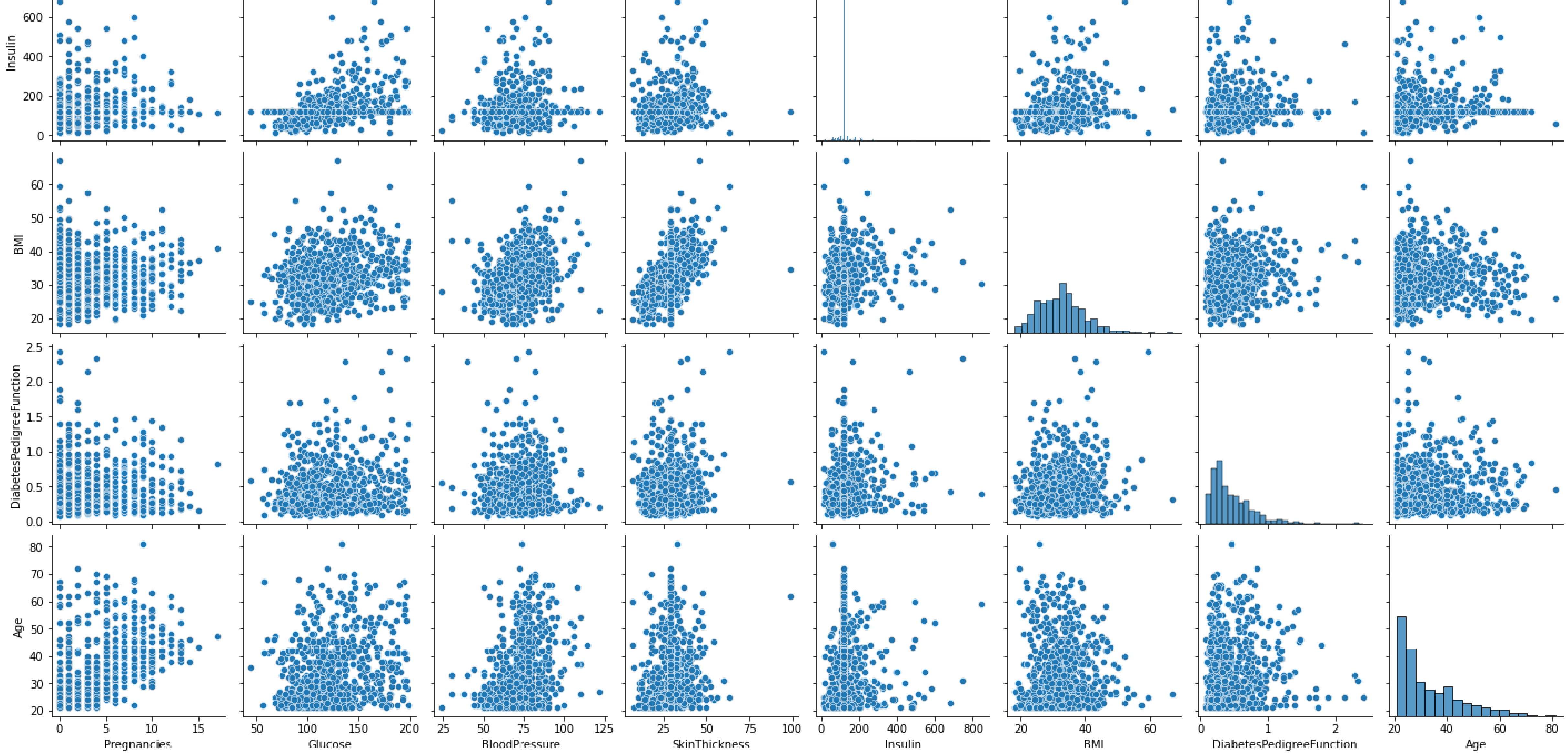
In [406]:

```
# Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
sns.pairplot(x_train.append(x_test))
# Observations:
# Number of Pregnancies is positively correlated with Age and no clear relationship can be seen with any other variable
# Glucose is positively coreelated with Insulin levels, BMI and Age
# Blood Pressure is slightly positively correlated with BMI and Age
# Skin Thickness is positively correlated with BMI
```

Out[406]:

```
<seaborn.axisgrid.PairGrid at 0x241db9a0e48>
```

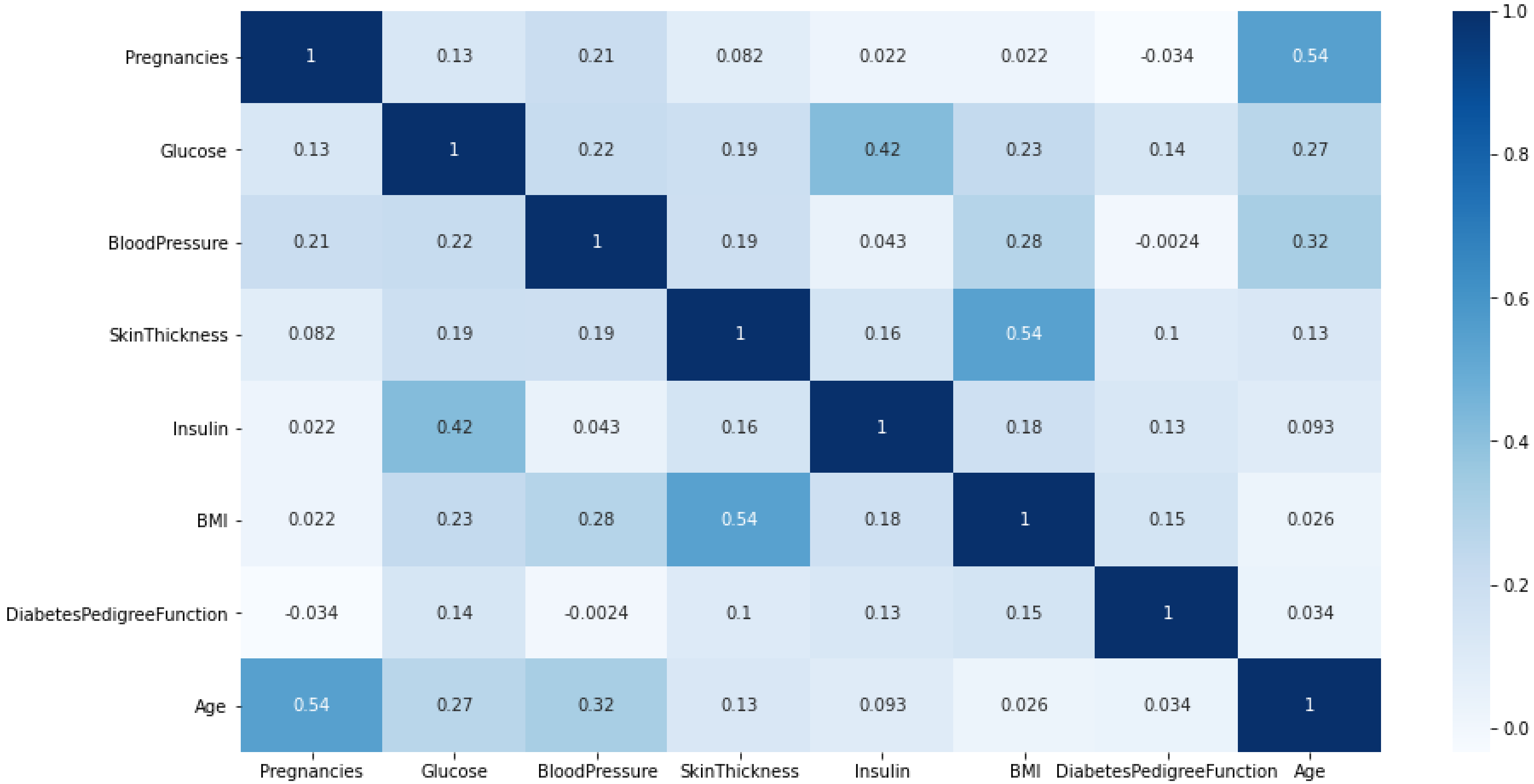



Insulin
BMI
DiabetesPedigreeFunction
Age
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age

In [407]:

```
#Perform correlation analysis. Visually explore it using a heat map.
plt.figure(figsize=(15,8))
sns.heatmap(x_train.append(x_test).corr(),annot=True,cmap='Blues')
# Observations:
# We can use correlation heatmap to provide affirmation to the relationships found using scatterplots
```

Out[407]: <AxesSubplot:>



In [408]:
```
# Data Modeling:

#1. Devise strategies for model building. It is important to decide the right validation framework.
# Express your thought process.

# Observations:
# As the objective is to predict whether a person has diabetes or not, we will choose classification techniques
# Since it is a binary classification problem, we can start with logistic regression and proceed with Naive bayes, KNN, SVM,
# decision trees and random forest
# We can validate the model using performance metrics like accuracy, precision, recall and f1-score
```

```
In [409]: #2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.
# Classification Model 1 ---Logistic Regression
from sklearn.linear_model import LogisticRegression

#Initiating the model
lr = LogisticRegression()

#Training the data
diabetes_logreg_model = lr.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_logreg_model.predict(x_test)
y_score = diabetes_logreg_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

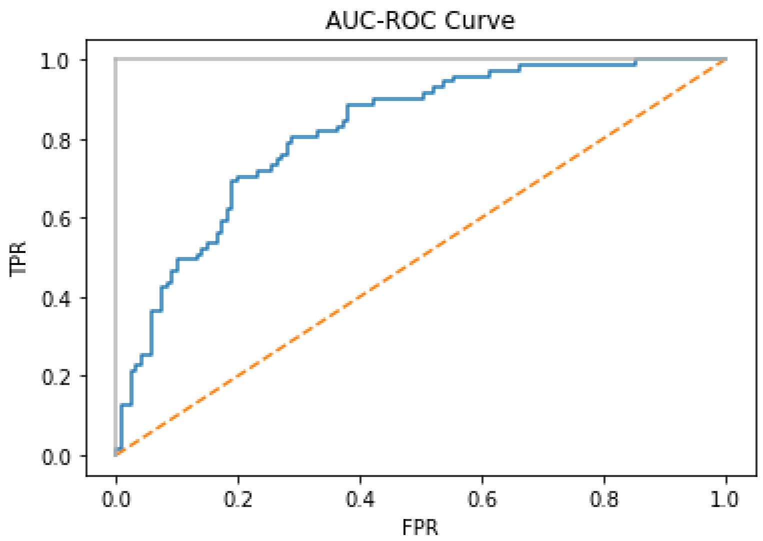
```
Predictions:
 [1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 1
 0 0 1 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 1 0 1 0
 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes:
 [0.54570864 0.18602329 0.56242091 0.10742602 0.08027523 0.04482417
 0.0834603  0.29423032 0.04409881 0.29290412 0.06147831 0.19366369
 0.91955662 0.27378981 0.09495772 0.66833483 0.76979316 0.04407987
 0.07405271 0.97583381 0.3236701  0.47340994 0.34970138 0.24921917
 0.13159764 0.68082063 0.26052793 0.38209257 0.46198958 0.84192141
 0.26835354 0.03587535 0.06455571 0.16371296 0.44188361 0.1061591
 0.38165122 0.22239545 0.75208911 0.04284856 0.3774296  0.29150902
 0.40725902 0.38320321 0.54170171 0.08362314 0.46954966 0.09775233
 0.48284205 0.32473913 0.07074289 0.90824224 0.19952115 0.34262681
 0.34050614 0.08107533 0.06868358 0.32742238 0.20840146 0.0364433
 0.13567119 0.09671573 0.03400335 0.43447926 0.20169735 0.51166791
 0.65398735 0.28222864 0.19468394 0.71959184 0.67748521 0.20619735
 0.37054698 0.62932693 0.19445892 0.1089558  0.66147042 0.77559054
 0.05061469 0.22615128 0.06769278 0.79295234 0.05075846 0.21749243
 0.29294788 0.76946905 0.50543046 0.36062173 0.87993113 0.28093858
 0.15191488 0.32273901 0.0403771  0.1345637  0.07309844 0.13132839
 0.86894455 0.08813901 0.09694291 0.29491311 0.73057071 0.30310616
 0.67243897 0.32584763 0.09726503 0.4913228  0.87898413 0.78806
 0.18479866 0.83678292 0.24912153 0.55928406 0.31621548 0.22498557
 0.37074461 0.07606599 0.02470633 0.46997015 0.13911628 0.88475715
 0.15028633 0.05752294 0.19210938 0.58418115 0.4573124  0.57802031
 0.20004619 0.18480833 0.46074005 0.08145138 0.79602347 0.40094351
 0.72838924 0.08839686 0.07064504 0.22839577 0.50134943 0.3882422
```

```
 0.71137804 0.68175731 0.21084411 0.05709248 0.38516209 0.20482728
 0.8146144  0.32282688 0.6625573  0.70261498 0.79731714 0.27301263
 0.26007837 0.06745565 0.65334093 0.05951761 0.91549742 0.07957326
 0.11428361 0.27006077 0.15612103 0.04552297 0.35264049 0.82545816
 0.06062423 0.14490473 0.39274206 0.37556037 0.27815403 0.54676724
 0.02686596 0.42807903 0.72828283 0.04760008 0.0291012  0.536198
 0.73290379 0.33007947 0.27227694 0.70479111 0.05926836 0.24347415
 0.27744446 0.02774268 0.08461068 0.17674637 0.09009181 0.17625555
 0.15900097 0.25237118 0.10321739 0.08810721 0.2005946  0.95868483]
```

In [410]:
```python
#Model Evaluation
from sklearn import metrics
def evaluate(y_test,y_pred,y_score):
    fpr, tpr, th = metrics.roc_curve(y_test,y_score)

    #Computing Metric Values
    cm = metrics.confusion_matrix(y_test,y_pred)
    print("Accuracy: ",metrics.accuracy_score(y_test,y_pred))
    print("\n Classification Report: \n",metrics.classification_report(y_test,y_pred))
    print("Sensitivity: \n",cm[1,1]/(cm[1,1]+cm[1,0]))
    print("\n Specificity: \n",cm[0,0]/(cm[0,0]+cm[0,1]))
    print("\n AUC Score: \n",metrics.roc_auc_score(y_test,y_score))
    print("\n Confusion Matrix: \n",metrics.confusion_matrix(y_test,y_pred))

    #Plotting AUC-ROC Curve
    plt.plot(fpr,tpr)
    plt.plot([0,1],ls="--")
    plt.plot([0,0],[1,0],c='0.7'),plt.plot([1,1],c='0.7')
    plt.title("AUC-ROC Curve")
    plt.xlabel("FPR")
    plt.ylabel("TPR")

evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.734375

 Classification Report: 
               precision    recall  f1-score   support

           0       0.75      0.88      0.81       121
           1       0.70      0.49      0.58        71

    accuracy                           0.73       192
   macro avg       0.72      0.68      0.69       192
weighted avg       0.73      0.73      0.72       192

Sensitivity: 
 0.49295774647887325

 Specificity: 
 0.8760330578512396

 AUC Score: 
 0.8181818181818183

 Confusion Matrix: 
```

```
[[106  15]
 [ 36  35]]
```



In [411]:
```
#Creating Variables to store metrics
acc = []
prec = []
rec = []
f1 = []
auc = []
sen = []
spec = []
```

In [412]:
```
#Passing metric values into list
def results(y_test,y_pred,y_score):
    acc.append(metrics.accuracy_score(y_test,y_pred))
    prec.append(metrics.precision_score(y_test,y_pred))
    rec.append(metrics.recall_score(y_test,y_pred))
    f1.append(metrics.f1_score(y_test,y_pred))
    auc.append(metrics.roc_auc_score(y_test,y_score))
    cm = metrics.confusion_matrix(y_test,y_pred)
    spec.append(cm[0,0]/(cm[0,0]+cm[0,1]))
    sen.append(cm[1,1]/(cm[1,1]+cm[1,0]))

results(y_test,y_pred,y_score)
```

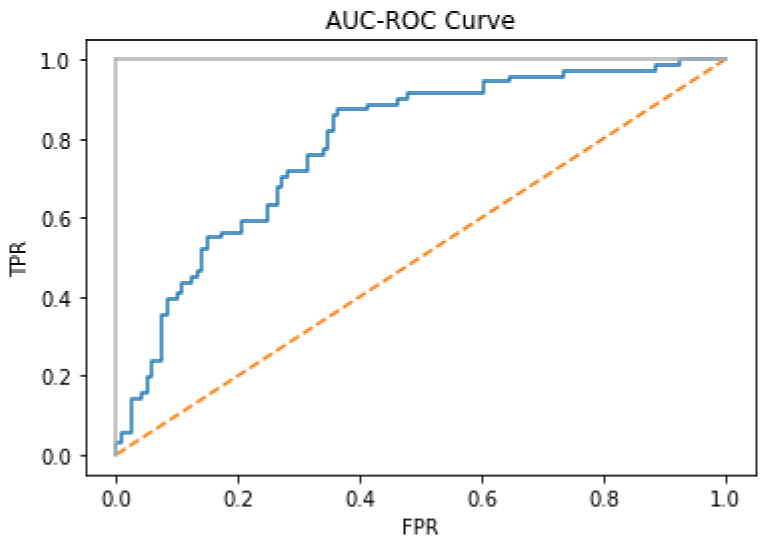
In [413]:
```
# Classification Model 2 ---Naive Bayes
from sklearn.naive_bayes import GaussianNB

#Initiating the model
nb = GaussianNB()

#Training the data
diabetes_nb_model = nb.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_nb_model.predict(x_test)
y_score = diabetes_nb_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions:
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0
 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1
 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 1 1 0 1 0
 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 1 1
 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes:
 [0.38358245 0.24167194 0.25814408 0.00579071 0.03112576 0.00363374
 0.00922275 0.15717169 0.01845631 0.35089651 0.00591287 0.22304191
 0.9982309  0.34846645 0.02411314 0.92666107 0.96163962 0.04657914
 0.10992066 0.9993804  0.13684936 0.58475875 0.55551833 0.48421721
 0.33012784 0.88048094 0.44360731 0.31134409 0.32181527 0.89339423
 0.03905201 0.0069864  0.00880741 0.0602197  0.59306965 0.01944284
 0.17861264 0.08964734 0.96014809 0.00589882 0.29884142 0.03419019
 0.57138847 0.36900901 0.80656367 0.11562879 0.59519033 0.02094675
 0.19233178 0.09556876 0.01082964 0.98128424 0.14129153 0.60947699
 0.32517772 0.0317174  0.00788904 0.21571981 0.0588123  0.0040487
 0.02581932 0.01850234 0.00448508 0.4972774  0.33763228 0.21580393
 0.80342681 0.18677102 0.07511768 0.85540102 0.76651482 0.24673592
 0.67547348 0.52165931 0.09766566 0.16978538 0.53076481 0.77167538
 0.00611235 0.15562922 0.0080066  0.9800525  0.00611525 0.0683074
 0.28058151 0.97014297 0.411921   0.87063422 0.95845832 0.07003373
 0.0367954  0.72109877 0.00274772 0.03217126 0.0181503  0.00938237
 0.97722918 0.01633351 0.00683814 0.29868349 0.96560549 0.17580712
 0.52242854 0.11624551 0.06141011 0.50092752 0.91358053 0.91842257
 0.0660736  0.98393727 0.19508318 0.82905559 0.05922102 0.09932764
 0.2426187  0.00238627 0.00131292 0.26129585 0.01521046 0.99690363
 0.01177803 0.00850701 0.22092202 0.40135213 0.85903404 0.36176074
 0.07546355 0.01727428 0.99343841 0.00832781 0.85658294 0.14435739
 0.45751944 0.00832624 0.01762085 0.07383963 0.11152504 0.20969097
```

```
 0.78274449 0.5252083  0.03649274 0.00640895 0.20655899 0.10017698
 0.67286717 0.86168832 0.74375284 0.799867   0.99917524 0.07622224
 0.23942664 0.0182769  0.6243779  0.00375286 0.98224215 0.01443875
 0.01979615 0.29517166 0.03497182 0.00967746 0.38624324 0.89050087
 0.03931342 0.36713633 0.77254552 0.40565372 0.0358466  0.27688654
 0.0210646  0.19335236 0.74837989 0.0112207  0.00505347 0.78483845
 0.880978   0.18455723 0.13472368 0.74860204 0.00667607 0.20095635
 0.20051178 0.00448849 0.05029653 0.09417955 0.01821686 0.10448969
 0.01129022 0.12700957 0.02837507 0.02109449 0.04575998 0.99845627]
```

In [414]:
```
#Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.7291666666666666

 Classification Report:
               precision    recall  f1-score   support

           0       0.75      0.85      0.80       121
           1       0.67      0.52      0.59        71

    accuracy                           0.73       192
   macro avg       0.71      0.69      0.69       192
weighted avg       0.72      0.73      0.72       192

Sensitivity:
 0.5211267605633803

 Specificity:
 0.8512396694214877

 AUC Score:
 0.7851239669421487

 Confusion Matrix:
 [[103  18]
 [ 34  37]]
```



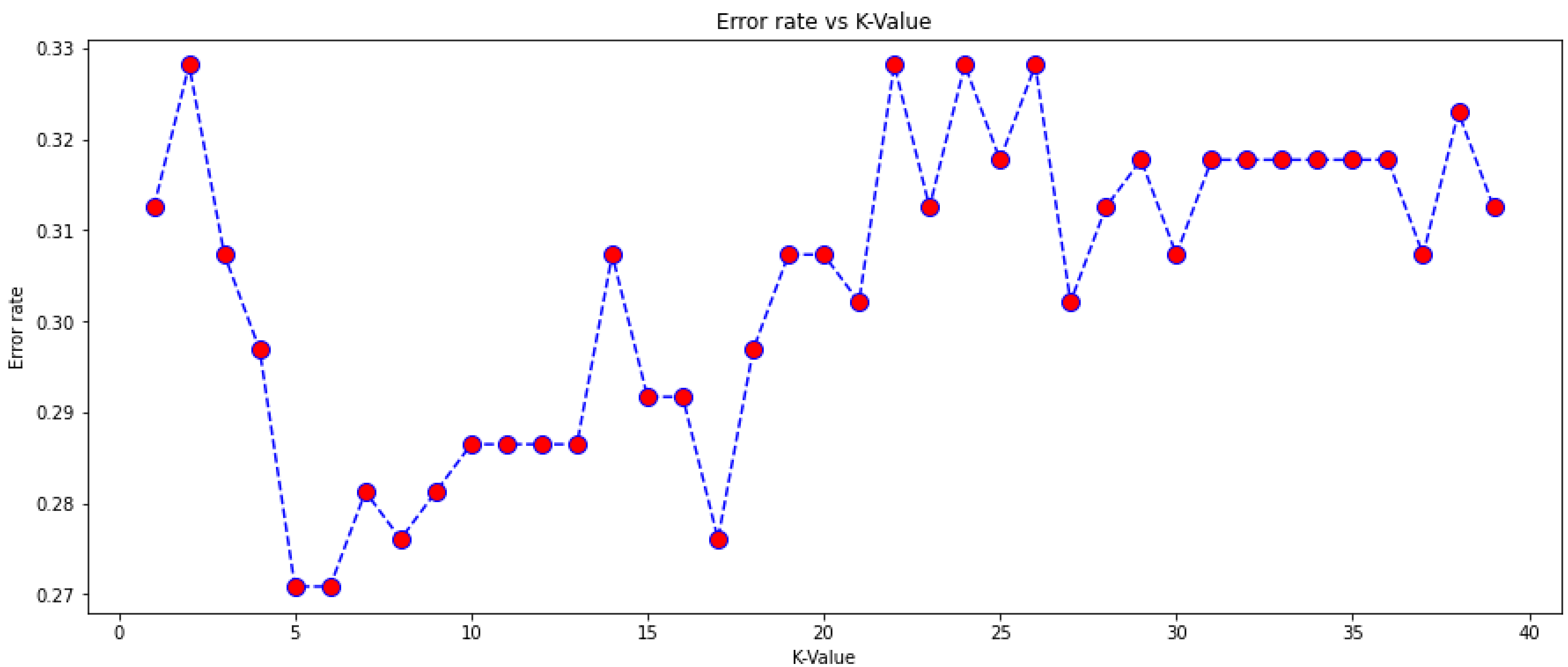
In [415]:
```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

In [416]:
```
# Classification Model 3 ---KNN
from sklearn.neighbors import KNeighborsClassifier

#Initiating the model
knn = KNeighborsClassifier(n_neighbors=5)

#Training the data
diabetes_knn_model = knn.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_knn_model.predict(x_test)
y_score = diabetes_knn_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions: 
 [0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
 0 1 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1
 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0
 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 0 0 1
 1 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes: 
 [0.4 0.4 0.8 0.  0.6 0.  0.  0.8 0.  0.4 0.  0.  0.8 0.2 0.2 1.  0.6 0.
 0.  1.  0.6 0.4 0.4 0.  0.  0.4 0.  0.4 0.8 1.  0.4 0.  0.  0.  0.4 0.4
 0.4 0.4 0.6 0.  0.6 0.2 0.4 0.2 0.6 0.  0.6 0.4 0.8 0.6 0.2 0.8 0.2 0.4
 0.4 0.  0.  0.4 0.6 0.  0.2 0.2 0.  0.2 0.  0.6 0.4 0.2 0.  0.4 0.8 0.2
 0.  0.6 0.2 0.  0.8 1.  0.4 0.2 0.  1.  0.  0.  1.  0.6 0.8 0.4 0.4 0.4
 0.4 0.2 0.  0.  0.4 0.  0.8 0.  0.2 0.8 0.2 0.4 0.4 0.6 0.2 0.2 1.  0.
 0.6 0.4 0.  0.8 0.2 0.  0.6 0.  0.  0.6 0.  0.4 0.2 0.  0.  0.4 0.4 0.8
 0.2 0.  0.8 0.  0.6 0.6 0.2 0.  0.  0.  0.8 0.4 0.8 0.8 0.4 0.  0.2 0.6
 0.8 0.4 0.4 1.  0.8 0.4 0.6 0.  0.2 0.  0.8 0.  0.  0.4 0.2 0.  0.2 0.6
 0.4 0.  0.2 0.8 0.  0.4 0.2 0.  0.6 0.  0.  0.6 0.8 0.6 0.4 0.8 0.  0.6
 0.2 0.  0.4 0.2 0.  0.  0.2 0.4 0.  0.2 0.  0.8]
```

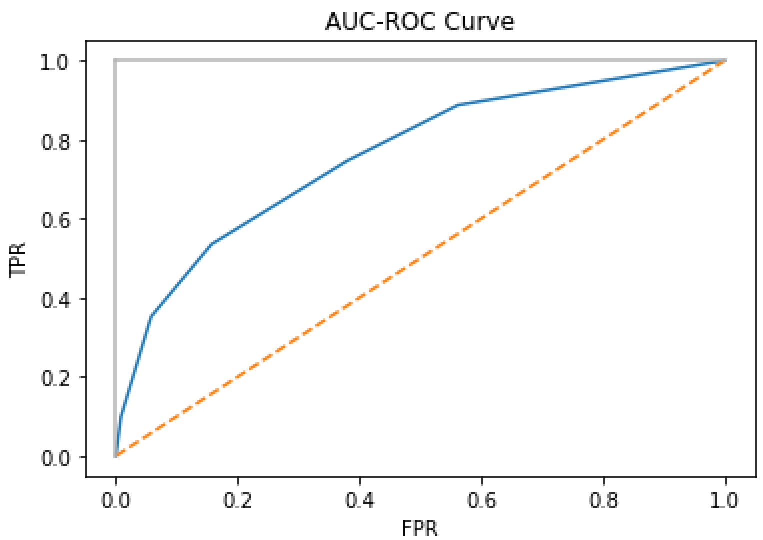
In [417]:
```python
#Finding optimal number for 'K' (Number of Neighbors)
import numpy as np
error_rate = []

for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    pred_i = knn.predict(x_test)
    error_rate.append(np.mean(pred_i!=y_test))

#Visualizing error rate
plt.figure(figsize=(15,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o',markerfacecolor='red',markersize=10)
plt.title("Error rate vs K-Value")
plt.xlabel("K-Value")
plt.ylabel("Error rate")
# Observations:
# Error rate is minimal for k=5,6 => Optimal K-Value=5 or 6
```

Out[417]:
```
Text(0, 0.5, 'Error rate')
```



```
In [418]: #Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.7291666666666666

 Classification Report:
               precision    recall  f1-score   support

           0       0.76      0.84      0.80       121
           1       0.67      0.54      0.59        71

    accuracy                           0.73       192
   macro avg       0.71      0.69      0.70       192
weighted avg       0.72      0.73      0.72       192

Sensitivity:
 0.5352112676056338

 Specificity:
 0.8429752066115702

 AUC Score:
 0.7604469793970434

 Confusion Matrix:
 [[102  19]
 [ 33  38]]
```



In [419]:

```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

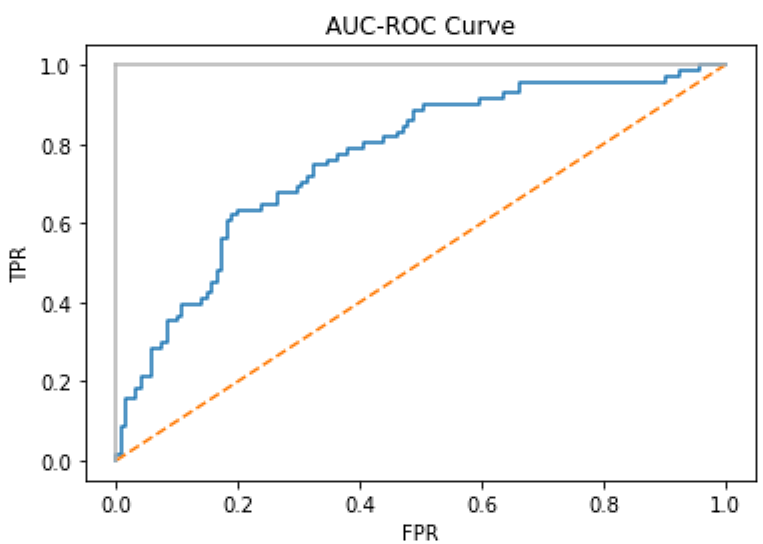
In [420]:
```
# Classification Model 4 ---SVM
from sklearn.svm import SVC

#Initiating the model
svc = SVC(probability=True)

#Training the data
diabetes_svc_model = svc.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_svc_model.predict(x_test)
y_score = diabetes_svc_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions:
 [1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1
 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 1 0
 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1
 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes:
 [0.64773692 0.24837852 0.47293221 0.11723825 0.143487   0.0856918
 0.1230792  0.25239868 0.08645449 0.5        0.09096335 0.28107278
 0.8606862  0.20664901 0.12288065 0.77088639 0.90331082 0.09936851
 0.11346125 0.86932196 0.24370468 0.31638004 0.19920493 0.08480476
 0.09853526 0.72538172 0.14373208 0.3094995  0.5196593  0.90648346
 0.27926711 0.06618877 0.14313153 0.1203312  0.36212492 0.22605962
 0.26751948 0.20459183 0.7235242  0.08817088 0.35307431 0.36586434
 0.33920927 0.24732906 0.44955837 0.10279763 0.42047562 0.22915438
 0.40360468 0.25534161 0.12933946 0.89783824 0.3579319  0.5
 0.40325623 0.10447378 0.08958479 0.26016805 0.18143549 0.10147632
 0.16422301 0.10200563 0.10719672 0.42293398 0.08788638 0.32961417
 0.59883806 0.42509397 0.11626899 0.41869142 0.83365177 0.17686349
 0.3417633  0.53381135 0.19409465 0.1025012  0.68686202 0.73731263
 0.12114507 0.18177056 0.09301964 0.82169912 0.07993098 0.10540153
 0.21470631 0.34879694 0.47759704 0.19683486 0.58254543 0.30907308
 0.22450128 0.27602454 0.09236943 0.1158775  0.14100169 0.08452508
 0.74911386 0.11318549 0.14580967 0.27483584 0.35744217 0.33301621
 0.29427352 0.29686152 0.14971511 0.57827991 0.75252037 0.68134326
 0.18623492 0.53744762 0.07661118 0.58554852 0.27733576 0.18081069
 0.18345793 0.14300906 0.08373071 0.54407879 0.1135195  0.71739059
 0.13685945 0.10373132 0.10531029 0.72289989 0.23894667 0.37538876
 0.13456743 0.20034006 0.76439812 0.11320772 0.69584301 0.42893625
 0.5966383  0.11634927 0.17030368 0.27790746 0.33414569 0.40057133
 0.78079309 0.58588801 0.27182014 0.12177334 0.26506376 0.21577686
```

```
 0.85938855 0.30383991 0.56597476 0.69287275 0.82501423 0.29342596
 0.22937979 0.09433458 0.42376368 0.10329154 0.78336262 0.08328003
 0.31600411 0.24130238 0.14086988 0.08410558 0.29032733 0.73959807
 0.23656125 0.11667697 0.24234003 0.40006252 0.2047724  0.3788733
 0.09232092 0.3185276  0.6586991  0.08222522 0.06567343 0.242052
 0.5813645  0.27301378 0.43139687 0.86389896 0.08265758 0.18698866
 0.26336353 0.09129814 0.28157713 0.24279449 0.10526975 0.08813145
 0.1426235  0.1592083  0.10781184 0.13083554 0.31336932 0.85910068]
```

In [421]:
```
#Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.703125

 Classification Report:
               precision    recall  f1-score   support

           0       0.71      0.89      0.79       121
           1       0.68      0.38      0.49        71

    accuracy                           0.70       192
   macro avg       0.69      0.64      0.64       192
weighted avg       0.70      0.70      0.68       192

Sensitivity:
 0.38028169014084506

 Specificity:
 0.8925619834710744

 AUC Score:
 0.7649866138982656

 Confusion Matrix:
 [[108  13]
 [ 44  27]]
```



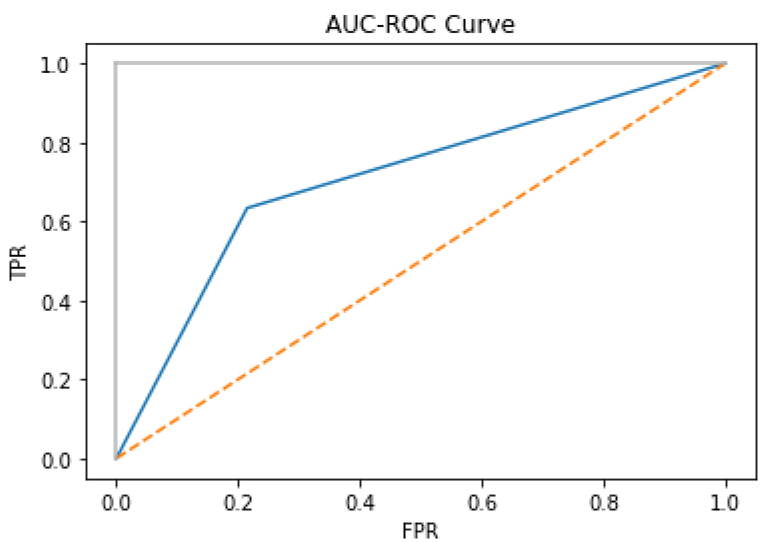
In [422]:
```python
#Passing metric values into list
results(y_test,y_pred,y_score)
```

In [423]:
```python
# Classification Model 5 ---Decision Trees
from sklearn.tree import DecisionTreeClassifier

#Initiating the model
dt = DecisionTreeClassifier()

#Training the data
diabetes_dt_model = dt.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_dt_model.predict(x_test)
y_score = diabetes_dt_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions: 
 [1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0
 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0
 0 0 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 1 0 1 0
 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 0 1 1
 1 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0
 0 0 1 0 0 0 1]
Predicted Probabilities for diabetes: 
 [1. 0. 1. 0. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 1. 0.
 0. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 0. 0. 1.
 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 1. 1. 0.
 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1. 0. 1.
 0. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 0. 1. 1. 1. 0. 1. 0. 1. 0.
 1. 0. 1. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1.
 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1.]
```

In [424]:
```
#Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.7291666666666666

 Classification Report:
               precision    recall  f1-score   support

           0       0.79      0.79      0.79       121
           1       0.63      0.63      0.63        71

    accuracy                           0.73       192
   macro avg       0.71      0.71      0.71       192
weighted avg       0.73      0.73      0.73       192

Sensitivity:
 0.6338028169014085

 Specificity:
 0.7851239669421488

 AUC Score:
 0.7094633919217787

 Confusion Matrix:
 [[95 26]
 [26 45]]
```



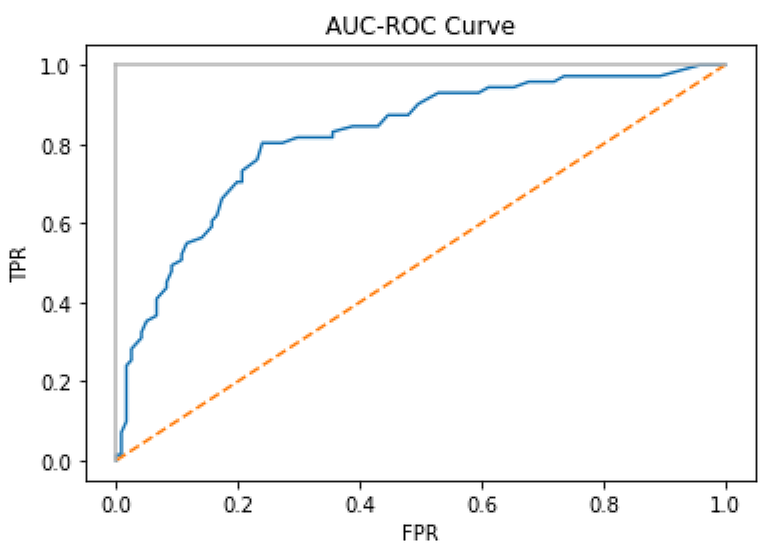
In [425]:
```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

In [426]:
```
# Classification Model 6 ---Random Forest
from sklearn.ensemble import RandomForestClassifier

#Initiating the model
rfc = RandomForestClassifier()

#Training the data
diabetes_rfc_model = rfc.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_rfc_model.predict(x_test)
y_score = diabetes_rfc_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions:
 [1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 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 0 0 0 0 0 0 0 1 1 0 0 1 1 1 0 0
 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0
 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0 1 0 1 0 1 1
 1 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes:
 [0.59 0.25 0.44 0.01 0.26 0.   0.01 0.56 0.09 0.42 0.04 0.33 0.95 0.21
 0.1  0.7  0.63 0.09 0.11 0.78 0.53 0.35 0.33 0.39 0.21 0.52 0.16 0.37
 0.54 0.85 0.1  0.06 0.03 0.17 0.42 0.11 0.26 0.24 0.82 0.02 0.47 0.07
 0.44 0.22 0.63 0.26 0.37 0.11 0.44 0.46 0.05 0.93 0.32 0.51 0.5  0.17
 0.07 0.13 0.37 0.01 0.12 0.06 0.03 0.18 0.11 0.63 0.75 0.42 0.2  0.66
 0.82 0.51 0.26 0.49 0.18 0.28 0.76 0.7  0.   0.12 0.   0.77 0.03 0.35
 0.36 0.72 0.49 0.36 0.72 0.14 0.24 0.37 0.   0.17 0.08 0.08 0.79 0.06
 0.18 0.23 0.42 0.22 0.59 0.56 0.1  0.43 0.8  0.83 0.5  0.53 0.05 0.59
 0.32 0.22 0.6  0.04 0.04 0.58 0.1  0.81 0.03 0.04 0.26 0.36 0.49 0.5
 0.14 0.17 0.33 0.   0.78 0.46 0.59 0.01 0.03 0.18 0.53 0.62 0.87 0.69
 0.35 0.02 0.51 0.25 0.79 0.37 0.66 0.82 0.76 0.4  0.61 0.13 0.49 0.01
 0.79 0.14 0.13 0.5  0.17 0.12 0.31 0.75 0.15 0.22 0.34 0.43 0.19 0.33
 0.03 0.41 0.69 0.02 0.04 0.56 0.66 0.36 0.29 0.67 0.02 0.44 0.11 0.02
 0.17 0.31 0.07 0.27 0.04 0.32 0.04 0.06 0.09 0.94]
```

In [427]:
```
#Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.7604166666666666

 Classification Report:
               precision    recall  f1-score   support

           0       0.77      0.88      0.82       121
           1       0.74      0.55      0.63        71

    accuracy                           0.76       192
   macro avg       0.75      0.72      0.73       192
weighted avg       0.76      0.76      0.75       192

Sensitivity:
 0.5492957746478874

 Specificity:
 0.8842975206611571

 AUC Score:
 0.8200442323361657

 Confusion Matrix:
 [[107  14]
 [ 32  39]]
```



In [428]:
```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

In [429]:

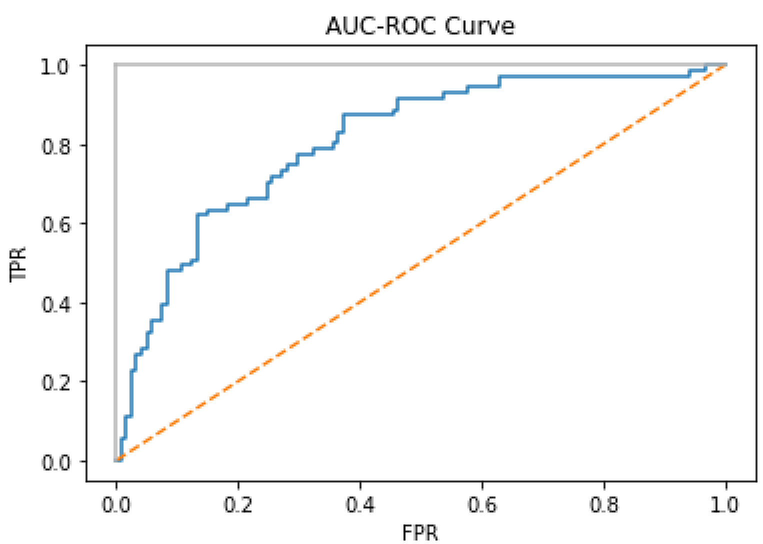
```
# Classification Model 7 ---XGBoost (Extreme Gradient Boost)
from xgboost import XGBClassifier

#Initiating the model
xgb = XGBClassifier(max_depth=5,n_estimators=100,learning_rate=0.05)
#xgb = XGBRFClassifier(max_depth=10,n_estimators=100,learning_rate=0.05)

#Training the data
diabetes_xgb_model = xgb.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_xgb_model.predict(x_test)
y_score = diabetes_xgb_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions: 
 [1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0
 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 1 0 0
 1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1
 1 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes: 
 [0.72407854 0.29476905 0.5800569  0.00980023 0.15538847 0.00858412
 0.05162851 0.5542416  0.05733493 0.41707045 0.01202211 0.13382986
 0.93788725 0.11831927 0.06448663 0.7565831  0.88275707 0.02881015
 0.04577414 0.79510134 0.48697668 0.39593467 0.25513202 0.18546177
 0.0440326  0.63608134 0.04306978 0.34740248 0.4086774  0.9226656
 0.10276293 0.03444543 0.01399162 0.12522642 0.13994414 0.1225279
 0.17310512 0.13392022 0.91641027 0.03234716 0.48695424 0.0518192
 0.52191114 0.18700972 0.5999961  0.13323708 0.58334064 0.03551537
 0.5799211  0.47886732 0.01263662 0.91513205 0.06461935 0.3463889
 0.4351886  0.10498597 0.04707078 0.10800327 0.28497717 0.00695379
 0.05681442 0.02230578 0.01236253 0.16862755 0.03675661 0.5585841
 0.73635656 0.41248178 0.07075122 0.59647125 0.804225   0.42323887
 0.3071385  0.34684858 0.08555964 0.08704429 0.82716435 0.7617669
 0.01613104 0.14991897 0.01294351 0.907928   0.01292854 0.3654208
 0.2886357  0.890019   0.1900371  0.26253998 0.8520289  0.12021236
 0.12974177 0.23675275 0.00992975 0.05359731 0.03288604 0.02524251
 0.8009173  0.07787064 0.12704074 0.4083957  0.7342442  0.36037564
 0.8096774  0.5961189  0.10145985 0.4319091  0.85015035 0.53027683
 0.56362057 0.42201006 0.06055201 0.68900555 0.2815981  0.1745073
 0.61508834 0.0094812  0.02639418 0.6387148  0.03109731 0.76955765
 0.02857239 0.01750568 0.12039563 0.42734465 0.8039021  0.3354279
 0.07348058 0.11992171 0.12591143 0.04135425 0.7632124  0.27381444
 0.64729095 0.05728618 0.02488619 0.114804   0.46568605 0.5034267
```

```
 0.933231   0.63884205 0.261563   0.02006053 0.5149661  0.30068022
 0.9076006  0.2326734  0.7021115  0.83461493 0.8497573  0.4997614
 0.6503149  0.02543828 0.5229806  0.01017851 0.9393277  0.07304693
 0.02269696 0.5479484  0.07655291 0.03823455 0.09748767 0.7762895
 0.02866547 0.05291839 0.23951416 0.29986748 0.23668659 0.13672163
 0.02014844 0.42252234 0.76387244 0.0234545  0.0118791  0.7574174
 0.7833119  0.45465863 0.36236182 0.83116144 0.01197286 0.6784343
 0.06542146 0.00900627 0.03126395 0.11029539 0.02718564 0.11477447
 0.02298712 0.43634486 0.0220334  0.06012082 0.02749418 0.9440062 ]
```

In [430]:
```
#Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.7604166666666666

 Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.87      0.82       121
           1       0.72      0.58      0.64        71

    accuracy                           0.76       192
   macro avg       0.75      0.72      0.73       192
weighted avg       0.76      0.76      0.75       192

Sensitivity:
 0.5774647887323944

 Specificity:
 0.8677685950413223

 AUC Score:
 0.8086369456407869

 Confusion Matrix:
 [[105  16]
 [ 30  41]]
```



In [431]:

```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

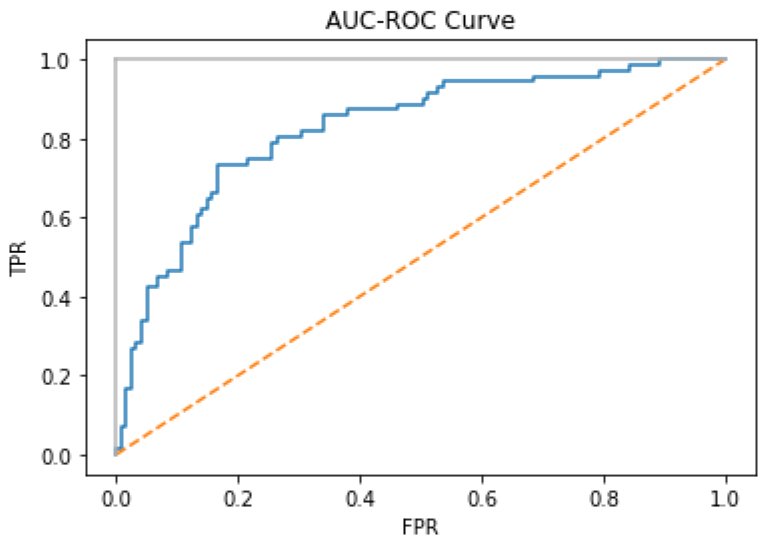
In [432]:
```python
# Classification Model 8 ---XGBoost with Random Forest (Extreme Gradient Boost with Random Forest)
from xgboost import XGBRFClassifier

#Initiating the model
xgbrf = XGBRFClassifier(max_depth=10,n_estimators=100,learning_rate=0.05)

#Training the data
diabetes_xgbrf_model = xgbrf.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_xgbrf_model.predict(x_test)
y_score = diabetes_xgbrf_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions: 
 [1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0
 0 1 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0
 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 1 0
 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1
 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0 1 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes: 
 [0.50765467 0.4825804  0.5011224  0.47536993 0.49225974 0.4750208
 0.47512057 0.50586504 0.47883058 0.49540338 0.4756756  0.4827393
 0.51862824 0.48508456 0.47760314 0.51271266 0.51637626 0.47984907
 0.48195103 0.5169981  0.5003002  0.4940526  0.48439202 0.48757285
 0.47539493 0.50342435 0.48376763 0.498778   0.5021733  0.5196018
 0.4821005  0.47615665 0.47543642 0.48183662 0.49105248 0.48510286
 0.481892   0.48401436 0.5187162  0.4750208  0.506485   0.4814225
 0.49410504 0.47706735 0.50824165 0.4827569  0.49614358 0.48200306
 0.50280714 0.5037372  0.4763629  0.5186996  0.484687   0.49105218
 0.4926007  0.47888285 0.47640088 0.47766253 0.4846478  0.47546974
 0.47870958 0.47708753 0.4750208  0.48252448 0.47728    0.5033161
 0.5074911  0.49141365 0.48980504 0.50551826 0.52015215 0.49491036
 0.48612973 0.49270892 0.47991517 0.47631067 0.51509595 0.5023456
 0.4750208  0.48168695 0.4758272  0.5210209  0.4751455  0.493911
 0.49466532 0.5129741  0.4883569  0.4913759  0.5157972  0.48160604
 0.48328602 0.4858781  0.4750208  0.47773498 0.47596613 0.47794294
 0.51544374 0.47654346 0.48047885 0.49295637 0.49568126 0.49122787
 0.50659513 0.50630337 0.48216686 0.4975649  0.516733   0.5074637
 0.49802172 0.5054857  0.47913718 0.49646288 0.49142087 0.48653546
 0.5088535  0.47516334 0.47587353 0.5073411  0.48214525 0.51600885
 0.47544122 0.47622392 0.48393887 0.5028333  0.50018215 0.4976275
 0.4784907  0.48418188 0.48696607 0.47568586 0.51590437 0.4951878
 0.50089574 0.4767026  0.47547805 0.4854304  0.49334484 0.50272554
 0.52138925 0.5086591  0.4932325  0.47503507 0.5000777  0.48297727
```

```
 0.5203815  0.4882074  0.51119286 0.5123778  0.51706356 0.49522334
 0.5006119  0.4751206  0.50045455 0.4750208  0.518161   0.47703767
 0.4786404  0.49314997 0.485764   0.477141   0.48292756 0.50609916
 0.4817974  0.47620374 0.4898487  0.49697983 0.48321888 0.48985025
 0.4775021  0.49177042 0.50334823 0.47502086 0.47541273 0.5076239
 0.5184817  0.5021878  0.4895611  0.513254   0.475501   0.5081056
 0.4791287  0.47527018 0.48186752 0.47982216 0.4765636  0.48185566
 0.47772074 0.48623568 0.47773543 0.4793594  0.47984675 0.52121574]
```

```
In [433]: #Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.765625

 Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.87      0.82       121
           1       0.72      0.59      0.65        71

    accuracy                           0.77       192
   macro avg       0.75      0.73      0.74       192
weighted avg       0.76      0.77      0.76       192

Sensitivity:
 0.5915492957746479

 Specificity:
 0.8677685950413223

 AUC Score:
 0.826213479222442

 Confusion Matrix:
 [[105  16]
 [ 29  42]]
```



In [434]:

```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

In [435]:

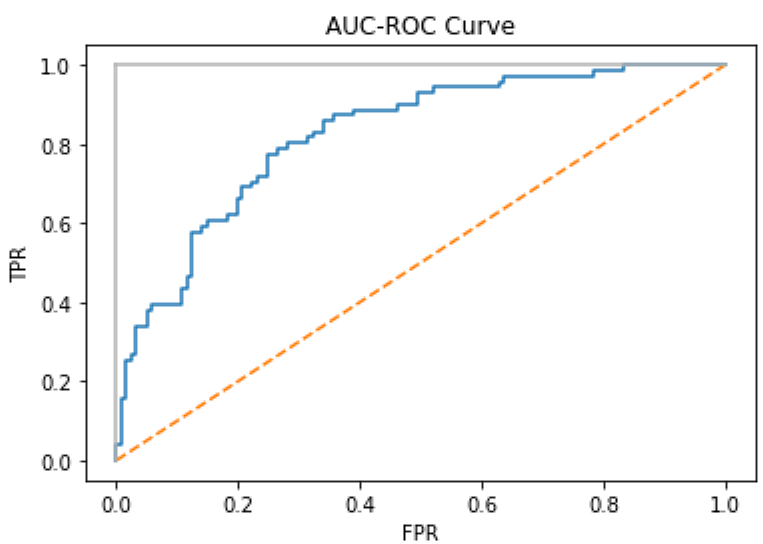
```python
# Classification Model 9 ---lightgbm (Light Gradient Boosting Mechanism)
import lightgbm as lgb

#Initiating the model
lgbm = lgb.LGBMClassifier(max_depth=7,n_estimators=100,num_leaves=100,boosting_type='dart',learning_rate=0.05)

#Training the data
diabetes_lgbm_model = lgbm.fit(x_train,y_train)

#Predictions on test data
y_pred = diabetes_lgbm_model.predict(x_test)
y_score = diabetes_lgbm_model.predict_proba(x_test)[:,1]
print("Predictions: \n",y_pred)
print("Predicted Probabilities for diabetes: \n",y_score)
```

```
Predictions:
 [1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0
 0 1 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0
 0 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 0 1 0
 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1
 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 0 0
 0 0 0 0 0 0 1]
Predicted Probabilities for diabetes:
 [0.70925338 0.30099592 0.51371888 0.06866427 0.21874506 0.05302457
 0.10355582 0.52898484 0.10495372 0.39597173 0.0486     0.19709245
 0.8248774  0.31375206 0.1033305  0.66296024 0.75609165 0.06477603
 0.08906904 0.86934485 0.5224748  0.50620544 0.29766873 0.3384804
 0.10790025 0.64708565 0.12682798 0.38117819 0.51402945 0.8503603
 0.12825682 0.08028097 0.05690168 0.24350058 0.33775291 0.22631289
 0.31325127 0.23160414 0.64986147 0.06354676 0.50080093 0.13976778
 0.24270409 0.10722339 0.54499936 0.16328548 0.47429529 0.11824514
 0.5556382  0.57398195 0.0567011  0.82592962 0.21588604 0.3442821
 0.47522318 0.16318697 0.10571371 0.08848548 0.28550896 0.05676714
 0.07024187 0.07765827 0.05230549 0.25708361 0.06911191 0.49357
 0.74427771 0.31480121 0.30871698 0.56535341 0.75306994 0.52179812
 0.46172819 0.47781844 0.17455803 0.10515604 0.77829083 0.56429675
 0.06108993 0.17635881 0.05748924 0.79860526 0.05480786 0.3075166
 0.46113215 0.7495498  0.26039399 0.31966546 0.7702167  0.10996952
 0.09106465 0.32827579 0.05458154 0.13032345 0.05664515 0.08387244
 0.83633643 0.13886853 0.19357979 0.32256634 0.57371056 0.28512378
 0.6733317  0.60195751 0.20370054 0.44336561 0.78222437 0.61978631
 0.43665422 0.54637995 0.10339214 0.4089654  0.35454112 0.24723645
 0.52282584 0.06517984 0.06440341 0.68340141 0.12044484 0.74815836
 0.06821351 0.05264529 0.25250739 0.60211993 0.57542491 0.34990211
 0.08211923 0.29508962 0.26233367 0.07091273 0.6975856  0.38873345
 0.62296943 0.09809686 0.06098327 0.26283161 0.46780899 0.5586751
 0.88646396 0.7160588  0.28048728 0.05478334 0.54418397 0.3240032
```

```
0.83080804 0.25442399 0.70458348 0.69884802 0.7006571  0.43147
0.52350627 0.06170612 0.39456056 0.05933677 0.74981807 0.09321151
0.09117329 0.55916606 0.12683546 0.07552362 0.22861579 0.60037806
0.0999441  0.101571   0.38633294 0.33116111 0.2227392  0.26285521
0.06782082 0.38231325 0.61258233 0.06094712 0.05268667 0.7085209
0.73538904 0.41652959 0.28111637 0.62781375 0.06415985 0.62824879
0.16291671 0.05825539 0.11206454 0.09998347 0.06982237 0.14091557
0.06864807 0.34240397 0.10186663 0.07447069 0.10853384 0.84863907]
```

In [436]:
```
#Model Evaluation
evaluate(y_test,y_pred,y_score)
```

```
Accuracy:  0.7604166666666666

 Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.86      0.82       121
           1       0.71      0.59      0.65        71

    accuracy                           0.76       192
   macro avg       0.75      0.73      0.73       192
weighted avg       0.76      0.76      0.76       192

Sensitivity:
 0.5915492957746479

 Specificity:
 0.859504132231405

 AUC Score:
 0.8237690606448609

 Confusion Matrix:
 [[104  17]
 [ 29  42]]
```



In [437]:
```
#Passing metric values into list
results(y_test,y_pred,y_score)
```

In [438]:
```
#Summary Table with Metrics
cm = ['Logistic regression','Naive Bayes','KNN','SVM','Decision Tree','Random Forest','XGBoost','XGBoost with RF','Light GBM']
pd.DataFrame({'Classification Model':cm,'Accuracy':acc,'Precision':prec,'Recall':rec,'F1-Score':f1,
              'Sensitivity':sen,'Specificity':spec,'AUC-ROC Score':auc})
# Observations:
# XGBoost with Random Forest Classification models yields highest accuracy of 76.56% and AUC-ROC score of 0.8262
# Hence, XGBoost with Random Forest is the best Classification Model
```

Out[438]:

	Classification Model	Accuracy	Precision	Recall	F1-Score	Sensitivity	Specificity	AUC-ROC Score
0	Logistic regression	0.734375	0.700000	0.492958	0.578512	0.492958	0.876033	0.818182
1	Naive Bayes	0.729167	0.672727	0.521127	0.587302	0.521127	0.851240	0.785124
2	KNN	0.729167	0.666667	0.535211	0.593750	0.535211	0.842975	0.760447
3	SVM	0.703125	0.675000	0.380282	0.486486	0.380282	0.892562	0.764987
4	Decision Tree	0.729167	0.633803	0.633803	0.633803	0.633803	0.785124	0.709463
5	Random Forest	0.760417	0.735849	0.549296	0.629032	0.549296	0.884298	0.820044
6	XGBoost	0.760417	0.719298	0.577465	0.640625	0.577465	0.867769	0.808637
7	XGBoost with RF	0.765625	0.724138	0.591549	0.651163	0.591549	0.867769	0.826213
8	Light GBM	0.760417	0.711864	0.591549	0.646154	0.591549	0.859504	0.823769