HEALTHCARE INDUSTRY

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Problem Statement:

Build a model to accurately predict whether the patients in the dataset have diabetes or not?

Dataset Description:

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skin fold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age (years)

Outcome: Class variable (0 or 1) 268 of 768 are 1, the others are 0

Approach:

Following pointers will be helpful to structure your findings.

1. Perform descriptive analysis. It is very important to understand the variables and corresponding values. We need to think through - Can minimum value of below listed columns be zero (0)? On these columns, a value of zero does not make sense and thus indicates missing value.

    - Glucose
    - BloodPressure
    - SkinThickness
    - Insulin
    - BMI

In [5]:
```
dataset.describe()
```
Out[5]:

	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 [8]:
```
dataset.isna().sum()
```
Out[8]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [11]:
```
dataset['Glucose']=dataset[['Glucose']].replace(0,dataset['Glucose'].mean())
```

In [12]:
```
dataset['BloodPressure']=dataset[['BloodPressure']].replace(0,dataset['BloodPressure'].mean())
dataset['SkinThickness']=dataset[['SkinThickness']].replace(0,dataset['SkinThickness'].mean())
```

In [13]:
```
dataset['Insulin']=dataset[['Insulin']].replace(0,dataset['Insulin'].mean())
```

In [14]:
```
dataset['BMI']=dataset[['BMI']].replace(0,dataset['BMI'].mean())
```

In [15]:
```
dataset[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]
```
Out[15]:

	Glucose	BloodPressure	SkinThickness	Insulin	BMI
0	148.0	72.0	35.000000	79.799479	33.6
1	85.0	66.0	29.000000	79.799479	26.6
2	183.0	64.0	20.536458	79.799479	23.3
3	89.0	66.0	23.000000	94.000000	28.1
4	137.0	40.0	35.000000	168.000000	43.1
...	...	...	...	...	...
763	101.0	76.0	48.000000	180.000000	32.9
764	122.0	70.0	27.000000	79.799479	36.8
765	121.0	72.0	23.000000	112.000000	26.2
766	126.0	60.0	20.536458	79.799479	30.1
767	93.0	70.0	31.000000	79.799479	30.4

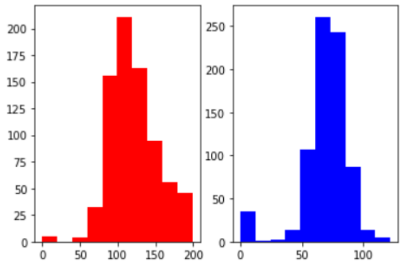
How will you treat these values?

1. Visually explore these variable, you may need to look for the distribution of these variables using histograms. Treat the missing values accordingly.

2. We observe integer as well as float data-type of variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

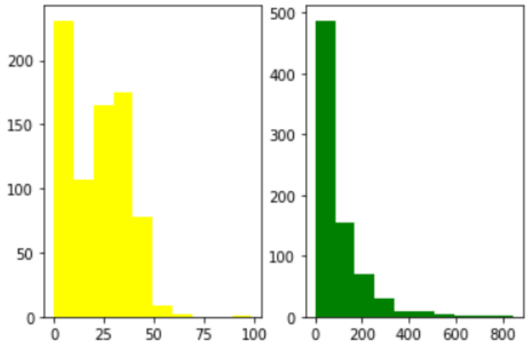
```
In [9]: plt.subplot(1,2,1)
        plt.hist(dataset['Glucose'],color='red')
        plt.subplot(1,2,2)
        plt.hist(dataset['BloodPressure'],color='blue')
```

```
Out[9]: (array([ 35.,   1.,   2.,  13., 107., 261., 243.,  87.,  14.,   5.]),
         array([  0. ,  12.2,  24.4,  36.6,  48.8,  61. ,  73.2,  85.4,  97.6,
                109.8, 122. ]),
         <BarContainer object of 10 artists>)
```



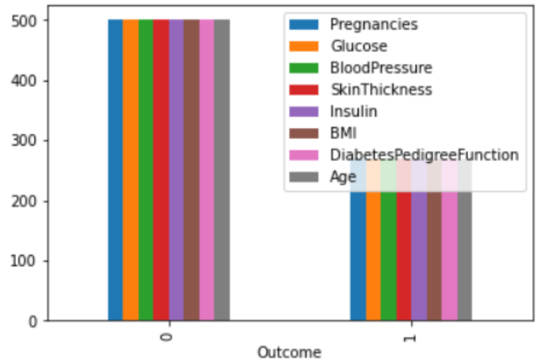
```
In [10]: plt.subplot(1,2,1)
         plt.hist(dataset['SkinThickness'],color='yellow')
         plt.subplot(1,2,2)
         plt.hist(dataset['Insulin'],color='green')
```

```
Out[10]: (array([487., 155.,  70.,  30.,   8.,   9.,   5.,   1.,   2.,   1.]),
          array([  0. ,  84.6, 169.2, 253.8, 338.4, 423. , 507.6, 592.2, 676.8,
                 761.4, 846. ]),
          <BarContainer object of 10 artists>)
```



```
In [20]: dataset.groupby('Outcome').count().plot.bar()
```

Out[20]: <AxesSubplot:xlabel='Outcome'>

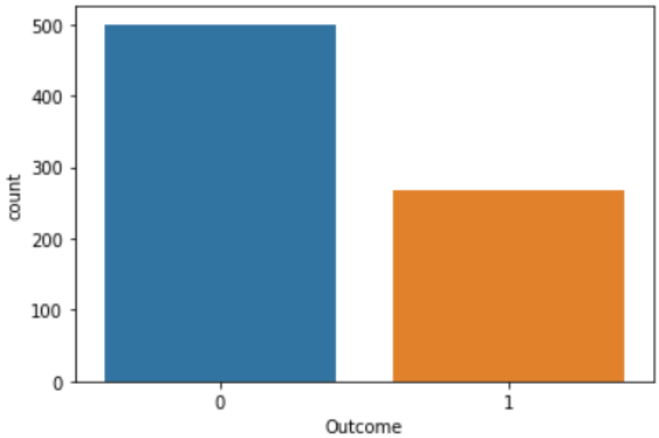


4. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of actions.

```
In [22]: import seaborn as sns
sns.countplot(dataset['Outcome'])
```

```
C:\Users\MONIKA\AppData\Roaming\Python\Python38\site-packages\seaborn\_decorato
able as a keyword arg: x. From version 0.12, the only valid positional argument
hout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[22]: <AxesSubplot:xlabel='Outcome', ylabel='count'>



From the graph we can say data is skewed towards 0 than 1 so model will mostly be trained with 0 than 1 this is problem of imbalance i
to apply sampling techniques.

In [23]:
```
X=dataset.drop('Outcome',axis=1)
Y=dataset['Outcome']
```

Either we can do upsampling or downsampling so we use SMOTE to sample the data

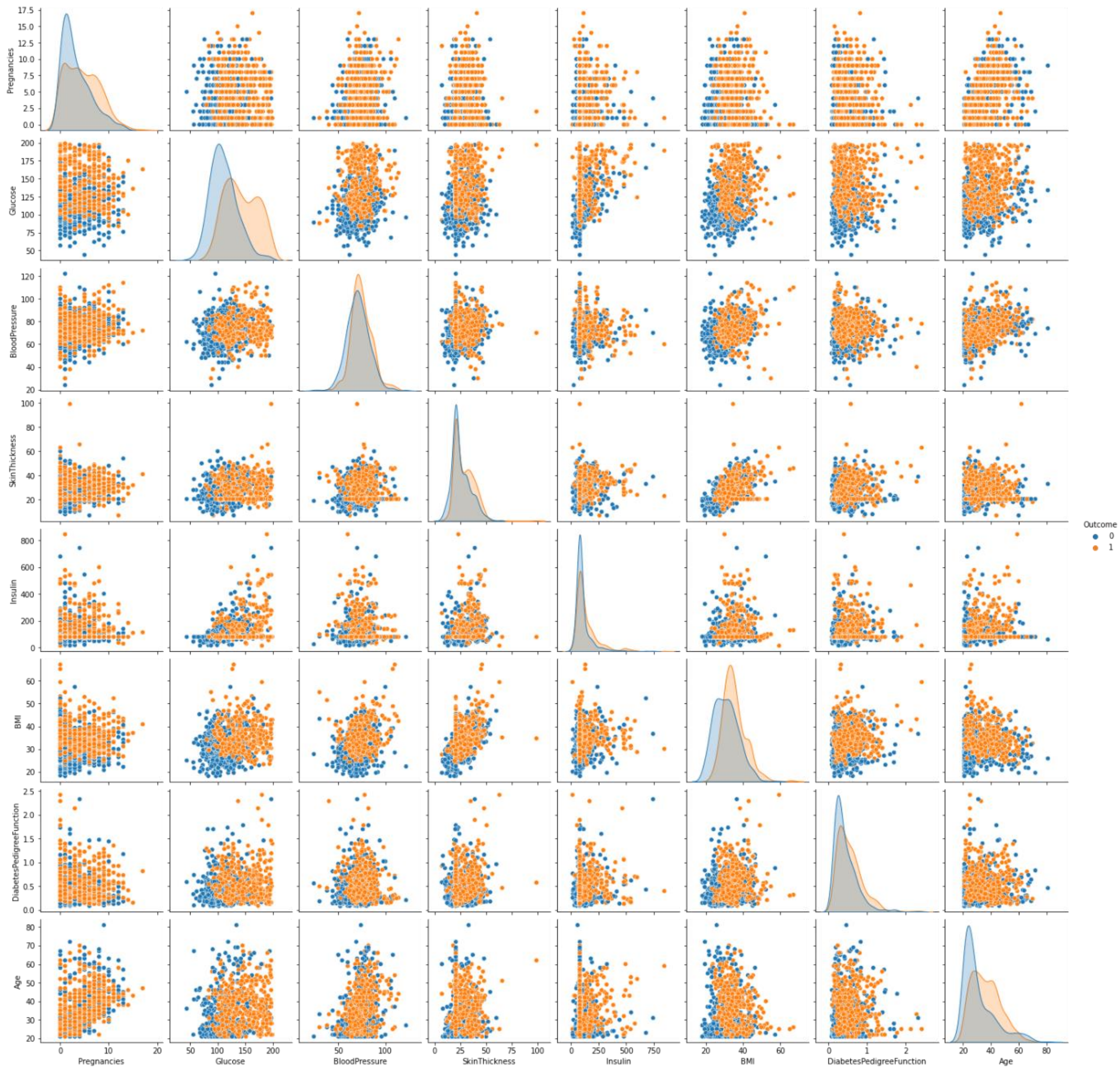
In [24]:
```
from imblearn.over_sampling import SMOTE
X_sample,Y_sample=SMOTE(random_state=108).fit_resample(X,Y)
X_sample,Y_sample
```

Out[24]:
```
(     Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
0              6  148.000000      72.000000      35.000000   79.799479
1              1   85.000000      66.000000      29.000000   79.799479
2              8  183.000000      64.000000      20.536458   79.799479
3              1   89.000000      66.000000      23.000000   94.000000
4              0  137.000000      40.000000      35.000000  168.000000
..           ...         ...            ...            ...         ...
995            3  164.686765      74.249021      20.536458   79.799479
996            0  144.889072      66.629691      27.629691  128.370309
997            4  140.837100      69.105469      20.536458   79.799479
998            0  105.571347      83.238205      20.536458   79.799479
999            0  126.544977     107.817758      45.090732  129.090732

          BMI  DiabetesPedigreeFunction  Age
0   33.600000                  0.627000   50
1   26.600000                  0.351000   31
```

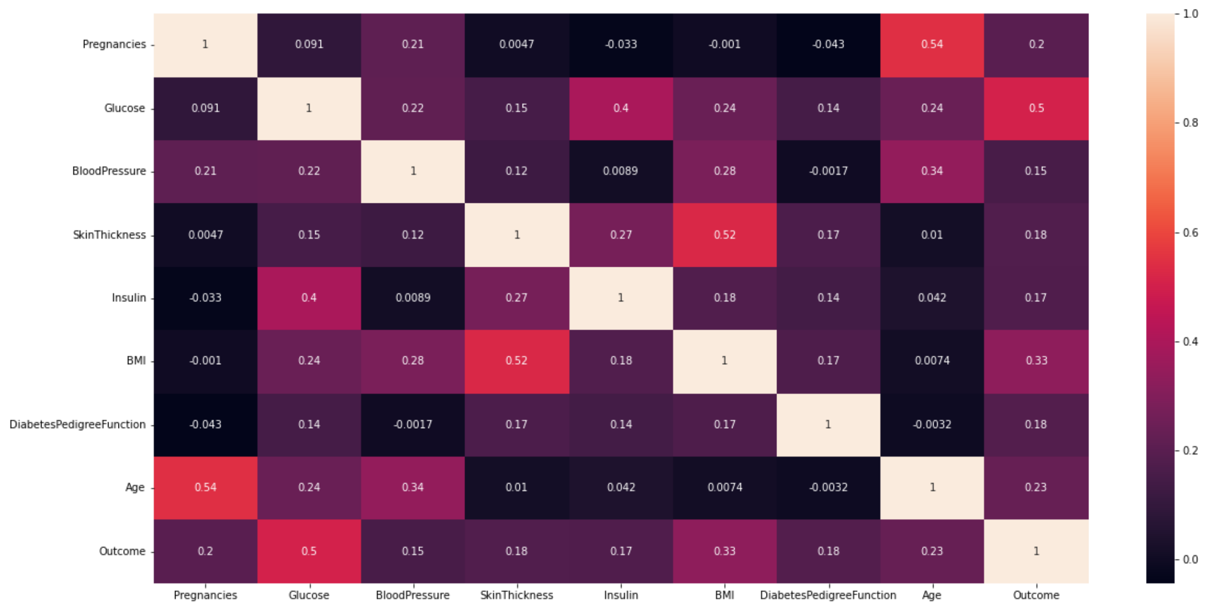
In [25]:
```
Y_sample.value_counts()
```

Out[25]:
```
1    500
0    500
Name: Outcome, dtype: int64
```

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



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



From the heatmap we can conclude that some of the variables have better correlation

Age and Pregnancies

Outcome and Glucose

SkinThickness and BMI

7. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process. Would Cross validation be useful in this scenario? .Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN. Create a classification report by analysing sensitivity, specificity, AUC(ROC curve) etc. Please try to be as descriptive as possible to explain what values of these parameter you settled for? any why?

Since the given problem is classifcation problem will work on some algorithms as mentioned here:

- Logistic Regression
- Random Forest
- NaiveBayes
- Support Vector Machine and compare it with KNN

For this problem cross validation is useful since we can decide which parameter values of a particular algorithm gives better result.

1. Logistic Regression.

```
In [33]: X1=data1.iloc[:,[0,1,2,3,4,5,6]]
         Y1=data1.iloc[:,7]

In [34]: type(X1)
         type(Y1)

Out[34]: pandas.core.series.Series

In [37]: from sklearn.model_selection import train_test_split
         X_train,X_test,Y_train,Y_test=train_test_split(X_sample,Y_sample,random_state=10,test_size=0.5)
         X_train.shape,Y_train.shape

Out[37]: ((500, 8), (500,))

In [38]: from sklearn.linear_model import LogisticRegression
         l1=LogisticRegression()

In [39]: l1.fit(X_train,Y_train)

B:\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (s
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Out[39]: ▾ LogisticRegression
```

```
       1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0], dtype=int64)
```

[41]:

```
from sklearn.metrics import mean_absolute_error,mean_squared_error
print('MAE:',mean_absolute_error(Y_test,pred))
print('MSE:',mean_squared_error(Y_test,pred))
print('RMSE:',np.sqrt(mean_squared_error(Y_test,pred)))
print('Train Acu',l1.score(X_train,Y_train))
print('Test Acu',l1.score(X_test,Y_test))
```

```
MAE: 0.268
MSE: 0.268
RMSE: 0.5176871642217914
Train Acu 0.736
Test Acu 0.732
```

In [42]:

```
from sklearn.model_selection import GridSearchCV
parameters={'C':np.logspace(-5,5,50)}
cv1=GridSearchCV(l1,param_grid=parameters,cv=5)
cv1.fit(X_train,Y_train)
```

In [73]:
```
l2.fit(X_train,Y_train)
```

Out[73]:
```
▾ LogisticRegression
LogisticRegression(C=0.04, max_iter=200)
```

In [74]:
```
pred2=l2.predict(X_test)
```

In [75]:
```
print('Accuracy Score:',l2.score(X_train,Y_train))
print('Test Accuracy:',l2.score(X_test,Y_test))
```

```
Accuracy Score: 0.726
Test Accuracy: 0.754
```

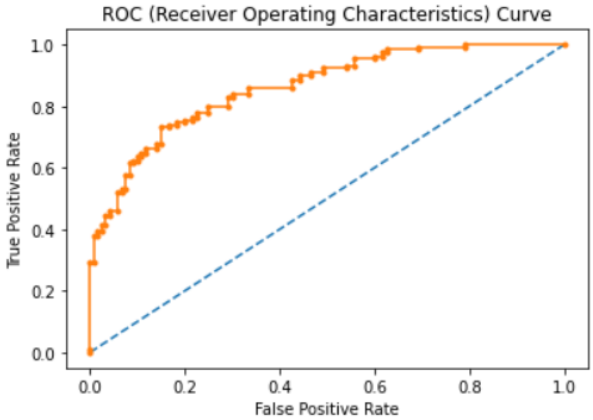
In [46]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve

print((l1.score(X_train,Y_train)))
print((l1.score(X_test,Y_test)))
```

```
0.752
0.776
```

In [47]:
```
p1=l1.predict_proba(X_test)
p2=p1[:,1]
fpr,tpr,thresh=roc_curve(Y_test,p2,pos_label=1)
p2
```

In [48]:
```
from sklearn.metrics import roc_auc_score,roc_curve
p4=l2.predict_proba(X_test)
p5=p4[:,1]
auc=roc_auc_score(Y_test,p5)
fpr,tpr,thresh=roc_curve(Y_test,p5,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')          # plot no skill
plt.plot(fpr, tpr, marker='.')                    # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
auc
```

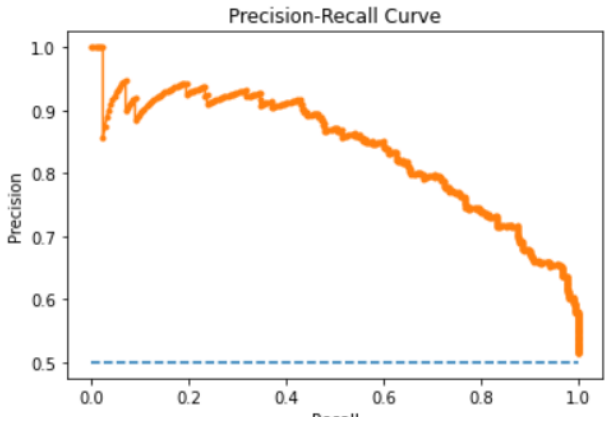
Out[48]:
```
0.8616666666666667
```



```
In [80]: from sklearn.metrics import precision_recall_curve,f1_score,auc,average_precision_scor
         pred_y_test = l1.predict(X_test)                                    # predict class v
         precision, recall, thresholds = precision_recall_curve(Y_test, p5) # calculate precisi
         f1 = f1_score(Y_test, pred_y_test)                                   # calculate F1 s
         auc_lr_pr = auc(recall, precision)                                   # calculate prec
         ap = average_precision_score(Y_test, p5)                           # calculate average
         print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_lr_pr, ap))
         plt.plot([0, 1], [0.5, 0.5], linestyle='--')                         # plot no skill
         plt.plot(recall, precision, marker='.')                              # plot the preci
         plt.xlabel("Recall")
         plt.ylabel("Precision")
         plt.title("Precision-Recall Curve")
```

```
f1=0.729 auc_pr=0.839 ap=0.839
```

```
Out[80]: Text(0.5, 1.0, 'Precision-Recall Curve')
```



Random Forest Classifier

```
In [81]: from sklearn.ensemble import RandomForestClassifier
         model1=RandomForestClassifier(n_estimators=100,criterion='entropy')
```

```
In [82]: model1.fit(X_train,Y_train)
```

```
Out[82]: RandomForestClassifier
         RandomForestClassifier(criterion='entropy')
```

```
In [83]: pred1=model1.predict(X_test)
```

```
In [84]: pred1
```

```
Out[84]: array([0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
                0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
                0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
                0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
                1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0,
                0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0,
                1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
                0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1,
                0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
                0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
                1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
                0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
                0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
                0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
```

In [109]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_test,pred1))
cm=confusion_matrix(Y_test,pred1)
print(cm)
print('Accuracy RF1',model1.score(X_train,Y_train)*100)
print('Accuracy RF1',model1.score(X_test,Y_test)*100)
print('Sensitivity',cm[0][0]/(cm[0][0]+cm[1][0]))
print('Specificity',cm[1][1]/(cm[0][1]+cm[1][1]))
```

```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77       243
           1       0.78      0.80      0.79       257

    accuracy                           0.78       500
   macro avg       0.78      0.78      0.78       500
weighted avg       0.78      0.78      0.78       500

[[185  58]
 [ 52 205]]
Accuracy RF1 100.0
Accuracy RF1 78.0
Sensitivity 0.7805907172995781
Specificity 0.779467680608365
```

In [112]:
```
parameters={
    'n_estimators':[20,40,80,100],
    'max_depth':[2,6,3],
    'criterion':['gini','entropy']
}

gs_rf=GridSearchCV(model1,param_grid=parameters,cv=5)
gs_rf.fit(X_train,Y_train)
```

Out[112]:
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [113]:
```
gs_rf.best_params_
```

Out[113]:
```
{'criterion': 'gini', 'max_depth': 6, 'n_estimators': 100}
```

In [114]:
```
model2=RandomForestClassifier(max_depth=6,n_estimators=100,criterion='gini')
model2.fit(X_train,Y_train)
```

Out[114]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=6)
```

In [119]:
```
pred5=model2.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
c2=confusion_matrix(Y_test,pred5)

print(classification_report(Y_test,pred5))

print('Train accuracy',model2.score(X_train,Y_train)*100)
print('Test accuracy',model2.score(X_test,Y_test)*100)
print('Sensitivity',c2[0][0]/(c2[0][0]+c2[1][0]))
print('Specificity',c2[1][1]/(c2[1][1]+c2[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.80      0.72      0.76       243
           1       0.76      0.82      0.79       257

    accuracy                           0.78       500
   macro avg       0.78      0.77      0.77       500
weighted avg       0.78      0.78      0.78       500

Train accuracy 91.60000000000001
Test accuracy 77.60000000000001
Sensitivity 0.7963800904977375
Specificity 0.7598566308243727
```

In [121]:
```
parameters={
    'n_estimators':[20,60,80],
    'max_depth':[1,4,3],
    'criterion':['gini','entropy']
}

gs_rf1=GridSearchCV(model2,param_grid=parameters,cv=4)
gs_rf1.fit(X_train,Y_train)
gs_rf1.best_params_
```

Out[121]:
```
{'criterion': 'entropy', 'max_depth': 4, 'n_estimators': 20}
```

In [122]:
```
model3=RandomForestClassifier(max_depth=4,n_estimators=20,criterion='entropy')
model3.fit(X_train,Y_train)
```

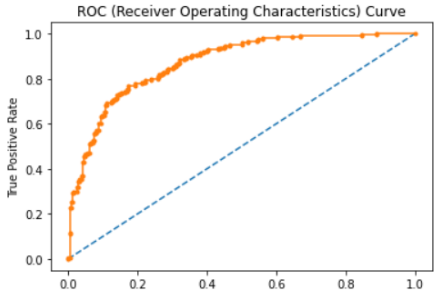
Out[122]:
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=4, n_estimators=20)
```

In [123]:
```
pred8=model3.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
c3=confusion_matrix(Y_test,pred8)

print(classification_report(Y_test,pred8))

print('Train accuracy',model3.score(X_train,Y_train)*100)
print('Test accuracy',model3.score(X_test,Y_test)*100)
print('Sensitivity',c3[0][0]/(c3[0][0]+c3[1][0]))
print('Specificity',c3[1][1]/(c3[1][1]+c3[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.78      0.74      0.76       243
           1       0.77      0.80      0.78       257

    accuracy                           0.77       500
   macro avg       0.77      0.77      0.77       500
weighted avg       0.77      0.77      0.77       500

Train accuracy 84.6
Test accuracy 77.4
Sensitivity 0.7801724137931034
Specificity 0.7686567164179104
```

In [124]:
```
rf=model2.predict_proba(X_test)
r1=rf[:,1]
auc=roc_auc_score(Y_test,r1)
fpr_rf,tpr_rf,thresh_rf=roc_curve(Y_test,r1,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')        # plot no skill
plt.plot(fpr_rf, tpr_rf, marker='.')             # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
auc
```

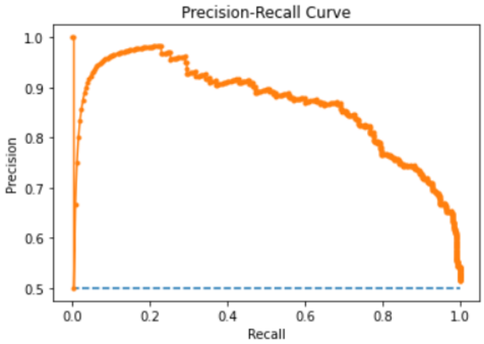
Out[124]:
```
0.8735488623080495
```



In [125]:
```
from sklearn import metrics
pred_y_test = model2.predict(X_test)                                  # predict class values
precision, recall, thresholds = precision_recall_curve(Y_test, r1) # calculate precision-recall curve
f1 = f1_score(Y_test, pred_y_test)                                 # calculate F1 score
auc_lr_pr = metrics.auc(recall, precision)                                 # calculate precision-recall AUC
ap = average_precision_score(Y_test, r1)                           # calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_lr_pr, ap))
plt.plot([0, 1], [0.5, 0.5], linestyle='--')                       # plot no skill
plt.plot(recall, precision, marker='.')                            # plot the precision-recall curve for the
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
```

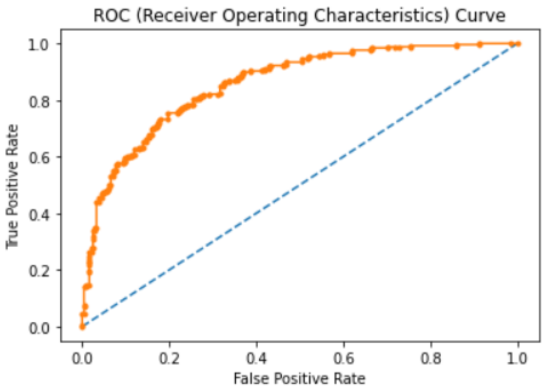
```
f1=0.791 auc_pr=0.866 ap=0.867
```

Out[125]:
```
Text(0.5, 1.0, 'Precision-Recall Curve')
```



In [126]:
```
rf2=model3.predict_proba(X_test)
r6=rf2[:,1]
auc_new=roc_auc_score(Y_test,r6,multi_class='ovo')
fpr_rf1,tpr_rf1,thresh_rf1=roc_curve(Y_test,r6,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')          # plot no skill
plt.plot(fpr_rf1, tpr_rf1, marker='.')                # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
auc_new
```

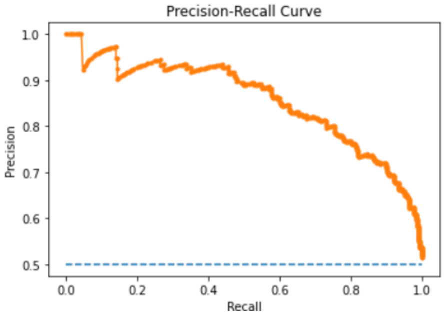
Out[126]:
```
0.8581127604041568
```



In [128]:
```python
from sklearn import metrics
pred_rf1 = model3.predict(X_test)                                         # predict class values
precision_rf2, recall_rf2, thresholds_rf2 = precision_recall_curve(Y_test, r6) # calculate precision-recall curve
f1_rf2 = f1_score(Y_test, pred_rf1)                                        # calculate F1 score
auc_lr_pr_rf2 = metrics.auc(recall_rf2, precision_rf2)                                       # calculate precision-recall AUC
ap_rf2 = average_precision_score(Y_test, r6)                               # calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1_rf2, auc_lr_pr_rf2, ap_rf2))
plt.plot([0, 1], [0.5, 0.5], linestyle='--')                              # plot no skill
plt.plot(recall_rf2, precision_rf2, marker='.')                                   # plot the precision-recall curve for the mode
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
```

```
f1=0.785 auc_pr=0.857 ap=0.858
```

Out[128]: Text(0.5, 1.0, 'Precision-Recall Curve')



Naïve Bayes

In [129]:
```python
from sklearn.naive_bayes import GaussianNB
model2=GaussianNB()
```

In [130]:
```python
model2.fit(X_train,Y_train)
```

Out[130]:
```
▾ GaussianNB
GaussianNB()
```

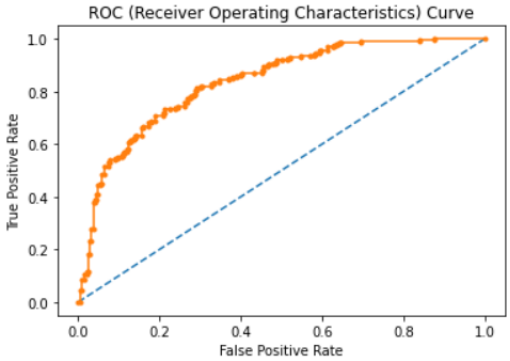
In [131]:
```python
pred3=model2.predict(X_test)
```

In [135]:
```python
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_test,pred3))
c4=confusion_matrix(Y_test,pred3)
print('Accuracy',model2.score(X_train,Y_train))
print('Accuracy2',model2.score(X_test,Y_test))
print('Sensitivity',c4[0][0]/(c4[0][0]+c4[1][0]))
print('Specificity',c4[1][1]/(c4[1][1]+c4[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.72      0.79      0.76       243
           1       0.78      0.72      0.75       257

    accuracy                           0.75       500
   macro avg       0.75      0.75      0.75       500
weighted avg       0.75      0.75      0.75       500

Accuracy 0.722
Accuracy2 0.752
Sensitivity 0.7245283018867924
Specificity 0.7829787234042553
```

n [137]:
```
nb=model2.predict_proba(X_test)
n1=nb[:,1]
auc=roc_auc_score(Y_test,n1)
fpr1,tpr1,thresh1=roc_curve(Y_test,n1,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')          # plot no skill
plt.plot(fpr1, tpr1, marker='.')                    # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
auc
```

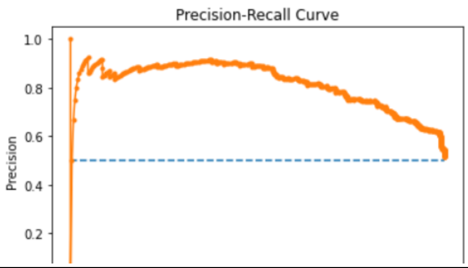
ut[137]: 0.8368160637940145



In [146]:
```
from sklearn import metrics
pred_new = model2_nb.predict(X_test)                                    # predict class values
precision1, recall1, thresholds1 = precision_recall_curve(Y_test, n1) # calculate precision-recall curve
f11 = f1_score(Y_test, pred_y_test)                                   # calculate F1 score
auc_lr_pr1 = metrics.auc(recall1, precision1)                               # calculate precision-recall AUC
ap1 = average_precision_score(Y_test, n1)                           # calculate average precision score
print('f1=%.3f,auc_lr_pr1=%3.f,ap=%.3f' % (f11,auc_lr_pr1,ap1))
plt.plot([0, 1], [0.5, 0.5], linestyle='--')                         # plot no skill
plt.plot(recall1, precision1, marker='.')                              # plot the precision-recall curve for the mod
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
```

```
f1=0.791,auc_lr_pr1=  1,ap=0.819
```

Out[146]: Text(0.5, 1.0, 'Precision-Recall Curve')



Support Vector Machine:

In [147]:
```python
from sklearn import svm
s1=svm.SVC(kernel='rbf',probability=True)
```

In [148]:
```python
s1.fit(X_train,Y_train)
```

Out[148]:
```
SVC
SVC(probability=True)
```

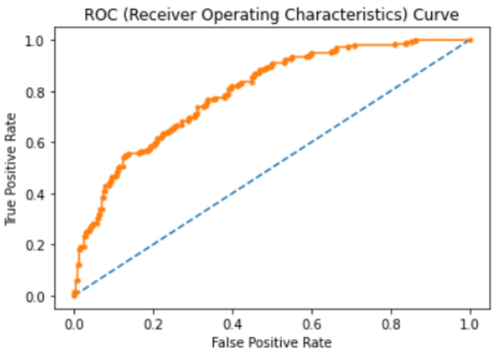
In [149]:
```python
pred1=s1.predict(X_test)
pred1
```

Out[149]:
```
array([0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0
```

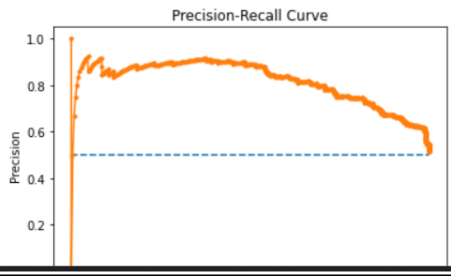
In [157]:
```python
from sklearn.metrics import classification_report,confusion_matrix
c5=confusion_matrix(Y_test,pred1_s)
print(classification_report(Y_test,pred1_s))
print("Ac1",s1.score(X_train,Y_train)*100)
print("Ac2",s1.score(X_test,Y_test)*100)
print("Sensitivity",c5[0][0]/(c5[0][0]+c5[1][0]))
print("Specificity",c5[1][1]/(c5[1][1]+c5[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.68      0.73      0.70       243
           1       0.73      0.67      0.70       257

    accuracy                           0.70       500
   macro avg       0.70      0.70      0.70       500
weighted avg       0.70      0.70      0.70       500

Ac1 73.4
Ac2 70.0
Sensitivity 0.6768060836501901
Specificity 0.7257383966244726
```

```
In [159]: sp=s1.predict_proba(X_test)
s2=sp[:,1]
from sklearn.metrics import roc_auc_score,roc_curve
auc1=roc_auc_score(Y_test,s2,multi_class='ovo')
fpr_s,tpr_s,thresh_s=roc_curve(Y_test,s2,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')          # plot no skill
plt.plot(fpr_s, tpr_s,marker='.')                   # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
auc1

Out[159]: 0.795968038942531
```



```
In [162]: from sklearn import metrics
#pred_ss = s1.predict(X_test)                                  # predict class values
precision_s, recall_s, thresholds_ss = precision_recall_curve(Y_test, s2) # calculate precision-recall curve
f1s = f1_score(Y_test, pred1_s)                                 # calculate F1 score
auc_lr_prs = metrics.auc(recall_s, precision_s)                            # calculate precision-recall AUC
ap1s = average_precision_score(Y_test, s2)                         # calculate average precision score
print('f1=%.3f,auc_lr_pr1=%3.f,ap=%.3f' % (f1s,auc_lr_prs,ap1s))
plt.plot([0, 1], [0.5, 0.5], linestyle='--')                        # plot no skill
plt.plot(recall1, precision1, marker='.')                             # plot the precision-recall curve for the model
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")

f1=0.696,auc_lr_pr1=  1,ap=0.793

Out[162]: Text(0.5, 1.0, 'Precision-Recall Curve')
```



K-Nearest Neighbor:

In [163]:
```python
from sklearn.neighbors import KNeighborsClassifier
kn1=KNeighborsClassifier(n_neighbors=6,weights='uniform')
```

In [164]:
```python
kn1.fit(X_train,Y_train)
```

Out[164]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=6)
```

In [166]:
```python
pred_k=kn1.predict(X_test)
```

In [168]:
```python
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_test,pred_k))
c6=confusion_matrix(Y_test,pred_k)
print("Train",kn1.score(X_train,Y_train))
print("Test",kn1.score(X_test,Y_test))
print("Specificity",c6[1][1]/(c6[1][1]+c6[0][1]))
print("Sensitivity",c6[0][0]/(c6[0][0]+c6[1][0]))
```

```
              precision    recall  f1-score   support

           0       0.69      0.71      0.70       243
           1       0.72      0.70      0.71       257

    accuracy                           0.71       500
   macro avg       0.71      0.71      0.71       500
weighted avg       0.71      0.71      0.71       500
```

```
Train 0.832
Test 0.708
Specificity 0.7211155378486056
Sensitivity 0.6947791164658634
```

n [169]:
```python
print(confusion_matrix(Y_test,pred_k))
```

```
[[173  70]
 [ 76 181]]
```

In [157]:
```python
parameters={
    'n_neighbors':[12,15,20,25],
    'weights':['uniform','distance'],
    'algorithm':['auto','kd_tree','brute'],
    'leaf_size':[30,60,80]
}

gs_knn=GridSearchCV(kn1,param_grid=parameters,cv=6)
gs_knn.fit(X_train,Y_train)
gs_knn.best_params_
```

Out[157]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'n_neighbors': 12,
 'weights': 'distance'}
```

In [158]:
```python
from sklearn.neighbors import KNeighborsClassifier
kn7=KNeighborsClassifier(n_neighbors=12,weights='distance',leaf_size=30,algorithm='auto')
```

In [159]:
```python
kn7.fit(X_train,Y_train)
```

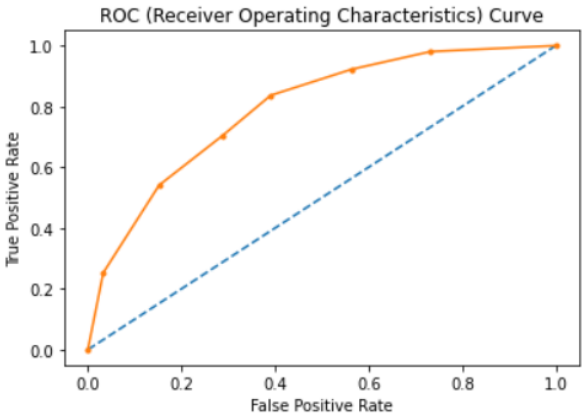
Out[159]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=12, weights='distance')
```

In [160]:
```python
predk7=kn7.predict(X_test)
```

In [176]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_test,predk7))
print("Train",kn7.score(X_train,Y_train)*100)
print("Test",kn7.score(X_test,Y_test)*100)
```

```
              precision    recall  f1-score   support

           0       0.75      0.65      0.70       243
           1       0.71      0.79      0.75       257

    accuracy                           0.73       500
   macro avg       0.73      0.72      0.72       500
weighted avg       0.73      0.73      0.72       500

Train 100.0
Test 72.6
```

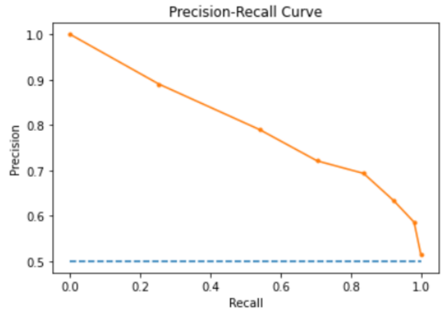
In [177]:
```
kn3=kn1.predict_proba(X_test)[:,1]
from sklearn import metrics
auc_k=metrics.roc_auc_score(Y_test,kn3,multi_class='ovo')
auc_k
fpr_k,tpr_k,thresh_k=roc_curve(Y_test,kn3,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')         # plot no skill
plt.plot(fpr_k, tpr_k,marker='.')                   # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
```



In [179]:
```
from sklearn import metrics
pred_kk = kn1.predict(X_test)                                    # predict class values
precision_k, recall_k, thresholds_kk = precision_recall_curve(Y_test, kn3) # calculate precision-recall curve
f1k = f1_score(Y_test, pred_kk)                                   # calculate F1 score
auc_lr_prk = metrics.auc(recall_k, precision_k)                                 # calculate precision-recall AUC
ap1k = average_precision_score(Y_test, kn3)                          # calculate average precision score
print('f1=%.3f,auc_lr_pr1=%3.f,ap=%.3f' % (f1k,auc_lr_prk,ap1k))
plt.plot([0, 1], [0.5, 0.5], linestyle='--')                        # plot no skill
plt.plot(recall_k, precision_k, marker='.')                            # plot the precision-recall curve for the model
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
```

```
f1=0.713,auc_lr_pr1=  1,ap=0.761
```

Out[179]:
```
Text(0.5, 1.0, 'Precision-Recall Curve')
```



XGBoost

In [180]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in b:\anaconda3\envs\nenvs\lib\site-packages (1.7.4)
Requirement already satisfied: numpy in b:\anaconda3\envs\nenvs\lib\site-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in b:\anaconda3\envs\nenvs\lib\site-packages (from xgboost) (1.10.0)
```

In [181]:
```
import xgboost
from xgboost import XGBClassifier
xg1=XGBClassifier(n_estimators=50,max_depth=100)
```

In [182]:
```
xg1.fit(X_train,Y_train)
```

Out[182]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=100, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=50, n_jobs=None, num_parallel_tree=None,
```

In [202]:
```
pred_xg=xg1.predict(X_test)
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_test,pred_xg))
c7=confusion_matrix(Y_test,pred_xg)
print('Train Accuracy',xg1.score(X_train,Y_train)*100)
print('Test Accuracy',xg1.score(X_test,Y_test)*100)
print('Sensitivity',c7[0][0]/(c7[0][0]+c7[1][0]))
print('Specificity',c7[1][1]/(c7[1][1]+c7[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.77      0.79      0.78       243
           1       0.80      0.77      0.78       257

    accuracy                           0.78       500
   macro avg       0.78      0.78      0.78       500
weighted avg       0.78      0.78      0.78       500

Train Accuracy 100.0
Test Accuracy 78.2
Sensitivity 0.7658730158730159
Specificity 0.7983870967741935
```

In [195]:
```
parameters={
    'n_estimators':[5,10,15,5],
    'max_depth':[50,100,150,200],
    'max_leaves':[3,5,6,10,15]
}

gs_xg=GridSearchCV(xg1,param_grid=parameters,cv=6)
gs_xg.fit(X_train,Y_train)
```

Out[195]:
```
GridSearchCV
estimator: XGBClassifier
XGBClassifier
```

In [196]:
```
gs_xg.best_params_
```

Out[196]:
```
{'max_depth': 50, 'max_leaves': 3, 'n_estimators': 15}
```

In [197]:
```
xg2=XGBClassifier(max_depth=50,max_leaves=3,n_estimators=15)
```

In [198]:
```
xg2.fit(X_train,Y_train)
```

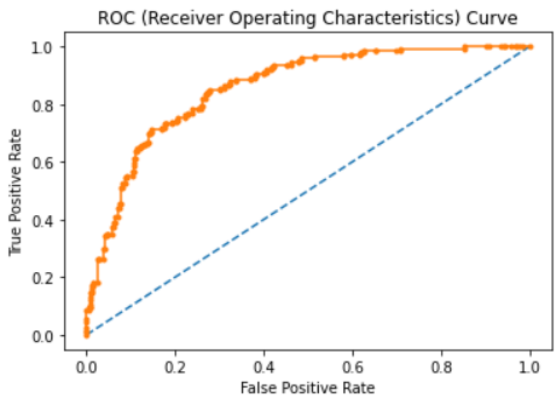
Out[198]:
```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
```

In [209]:
```
print(classification_report(Y_test,pred_xg2))
print("Train XG",xg2.score(X_train,Y_train)*100)
print("Test XG",xg2.score(X_test,Y_test)*100)
c9=confusion_matrix(Y_test,pred_xg2)
print("Sensitivity",c9[0][0]/(c9[0][0]+c9[1][0]))
print("Specificity",c9[1][1]/(c9[1][1]+c9[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.76      0.77      0.76       243
           1       0.78      0.77      0.77       257

    accuracy                           0.77       500
   macro avg       0.77      0.77      0.77       500
weighted avg       0.77      0.77      0.77       500

Train XG 99.8
Test XG 76.6
Sensitivity 0.7560975609756098
Specificity 0.7755905511811023
```

In [210]:
```
xgp=xg2.predict_proba(X_test)
xg1t=xgp[:,1]
auc_score_xg=roc_auc_score(Y_test,xg1t)
fpr_xg,tpr_xg,threshold_xg=roc_curve(Y_test,xg1t,pos_label=1)
plt.plot([0, 1], [0, 1], linestyle='--')          # plot no skill
plt.plot(fpr_xg, tpr_xg,marker='.')                # plot the roc curve for the model
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC (Receiver Operating Characteristics) Curve");
auc_score_xg
```

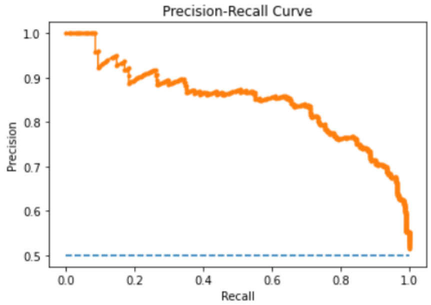
Out[210]: 0.8578725720965237



In [211]:
```
from sklearn import metrics
pred_xg2 = xg2.predict(X_test)                                    # predict class values
precision_xg2, recall_xg2, thresholds_xg2 = precision_recall_curve(Y_test, xg1t) # calculate precision-recall curve
f1xg = f1_score(Y_test, pred_xg2)                                  # calculate F1 score
auc_lr_prxg = metrics.auc(recall_xg2, precision_xg2)                            # calculate precision-recall AUC
ap1xg = average_precision_score(Y_test, xg1t)                        # calculate average precision score
print('f1=%.3f,auc_lr_pr1=%3.f,ap=%.3f' % (f1k,auc_lr_prxg,ap1xg))
plt.plot([0, 1], [0.5, 0.5], linestyle='--')                       # plot no skill
plt.plot(recall_xg2, precision_xg2, marker='.')                          # plot the precision-recall curve for the model
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
```

```
f1=0.713,auc_lr_pr1=  1,ap=0.848
```

Out[211]: Text(0.5, 1.0, 'Precision-Recall Curve')



Decision Tree:

In [212]:
```
from sklearn.tree import DecisionTreeClassifier
dt_ht=DecisionTreeClassifier(criterion='entropy',splitter='best',max_depth=100)
```

In [213]:
```
dt_ht.fit(X_train,Y_train)
```

Out[213]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=100)
```

In [214]:
```
pred_dt=dt_ht.predict(X_test)
```

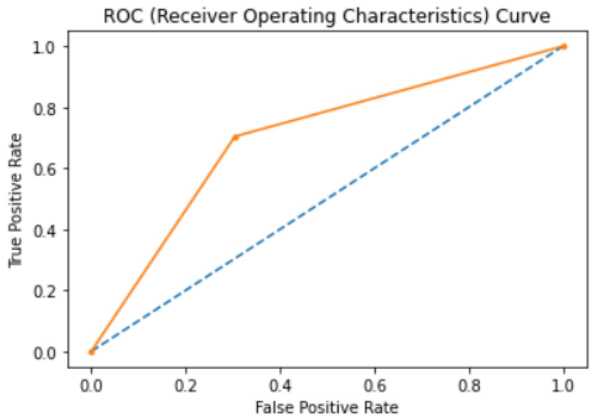
In [223]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(Y_test,pred_dt))
cd=confusion_matrix(Y_test,pred_dt)
print("DT Accuracy Train",dt_ht.score(X_train,Y_train)*100)
print("DT Accuracy Test",dt_ht.score(X_test,Y_test)*100)
print("Sensitivity",cd[0][0]/(cd[0][0]+cd[1][0]))
print("Specificity",cd[1][1]/(cd[1][1]+cd[0][1]))
```

```
              precision    recall  f1-score   support

           0       0.69      0.70      0.69       243
           1       0.71      0.70      0.71       257

    accuracy                           0.70       500
   macro avg       0.70      0.70      0.70       500
weighted avg       0.70      0.70      0.70       500

DT Accuracy Train 100.0
DT Accuracy Test 70.0
Sensitivity 0.689795918367347
Specificity 0.7098039215686275
```

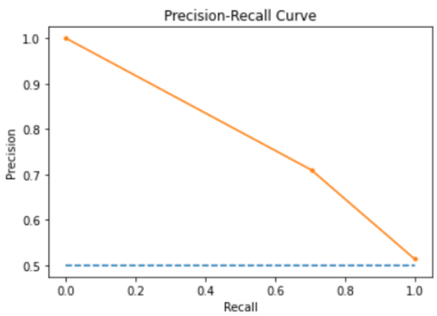
In [239]:
```
p1={
     'max_depth':[50,100,200],
     'criterion':['gini','entropy']
}
gs_dt=GridSearchCV(dt_ht,param_grid=p1,cv=5)
gs_dt.fit(X_train,Y_train)
gs_dt.best_params_
gs_dt.best_score_
```

Out[239]: 0.722

```
In [243]: d12=dt_ht.predict_proba(X_test)
          d6=d12[:,1]
          auc_new=roc_auc_score(Y_test,d6,multi_class='ovo')
          fpr_d1,tpr_d1,thresh_d1=roc_curve(Y_test,d6,pos_label=1)
          plt.plot([0, 1], [0, 1], linestyle='--')          # plot no skill
          plt.plot(fpr_d1, tpr_d1, marker='.')                 # plot the roc curve for the model
          plt.xlabel("False Positive Rate")
          plt.ylabel("True Positive Rate")
          plt.title("ROC (Receiver Operating Characteristics) Curve");
          auc_new

Out[243]: 0.6998767033354151
```




```
In [245]: from sklearn import metrics
          pred_d1 = dt_ht.predict(X_test)                                   # predict class values
          precision_dt, recall_dt, thresholds_dt = precision_recall_curve(Y_test, d6) # calculate precision-recall curve
          f1_d2 = f1_score(Y_test, pred_d1)                                 # calculate F1 score
          auc_lr_pr_d2 = metrics.auc(recall_dt, precision_dt)                         # calculate precision-recall AUC
          ap_d2 = average_precision_score(Y_test, r6)                        # calculate average precision score
          print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1_d2, auc_lr_pr_d2, ap_d2))
          plt.plot([0, 1], [0.5, 0.5], linestyle='--')                       # plot no skill
          plt.plot(recall_dt, precision_dt, marker='.')                             # plot the precision-recall curve for the model
          plt.xlabel("Recall")
          plt.ylabel("Precision")
          plt.title("Precision-Recall Curve")

          f1=0.707 auc_pr=0.783 ap=0.858

Out[245]: Text(0.5, 1.0, 'Precision-Recall Curve')
```



10. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a) Pie chart to describe the diabetic/non-diabetic population

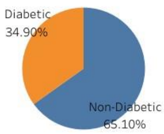
b) Scatter charts between relevant variables to analyse the relationships

c) Histogram/frequency charts to analyse the distribution of the data

d) Heatmap of correlation analysis among the relevant variables

e) Create bins of Age values – 20-25, 25-30, 30-35 etc. and analyse different variables for these age brackets using a bubble chart.

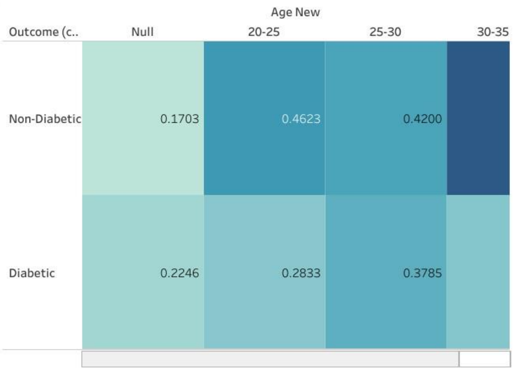
Pie Chart for Diabetic vs Non Diabetic



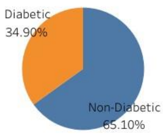
Frequency Plot'

Scatterplot

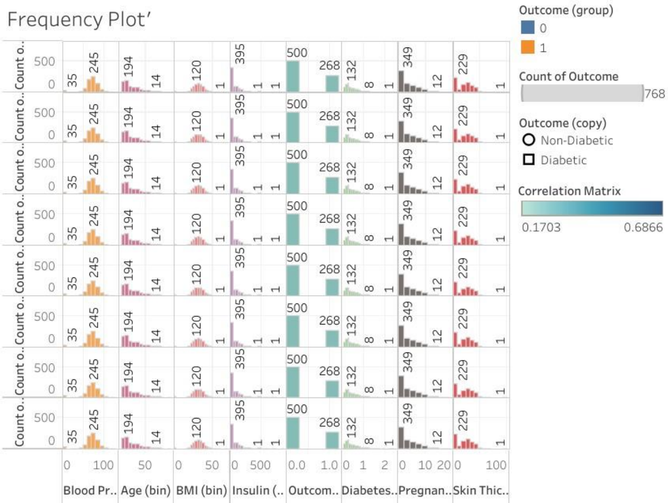
Correlation Heatmap



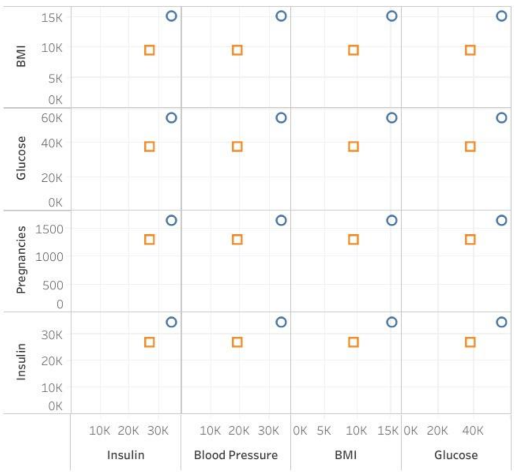
Pie Chart for Diabetic vs Non Diabetic



Frequency Plot'



Scatterplot



Correlation Heatmap