Capstone Project (Healthcare) By Mehul Parmar

December 15, 2022

0.1 Week 1

(1) Read data and perform descriptive analysis

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="white", color_codes=True)
sns.set(font_scale=1.2)
from imblearn.over_sampling import SMOTE
```

[2]:
```
data = pd.read_csv('C:/Users/Lenovo/Project 2 (Healthcare - Diabetes)/
↪Healthcare - Diabetes/health care diabetes.csv')
```

[3]:
```
data.head()
```

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

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[4]:
```
cols_with_null_as_zero = ['Glucose', 'BloodPressure', 'SkinThickness',␣
↪'Insulin', 'BMI']
data[cols_with_null_as_zero] = data[cols_with_null_as_zero].replace(0, np.NaN)
```

[5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   763 non-null    float64
 2   BloodPressure             733 non-null    float64
 3   SkinThickness             541 non-null    float64
 4   Insulin                   394 non-null    float64
 5   BMI                       757 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 54.1 KB
```

[6]:
```
data.isnull().sum()
```

[6]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

[7]:
```
data.describe()
```

[7]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  763.000000     733.000000     541.000000  394.000000   
mean      3.845052  121.686763      72.405184      29.153420  155.548223   
std       3.369578   30.535641      12.382158      10.476982  118.775855   
min       0.000000   44.000000      24.000000       7.000000   14.000000   
25%       1.000000   99.000000      64.000000      22.000000   76.250000   
50%       3.000000  117.000000      72.000000      29.000000  125.000000   
75%       6.000000  141.000000      80.000000      36.000000  190.000000   
max      17.000000  199.000000     122.000000      99.000000  846.000000   

              BMI  DiabetesPedigreeFunction         Age     Outcome  
count  757.000000                768.000000  768.000000  768.000000  
mean    32.457464                  0.471876   33.240885    0.348958  
std      6.924988                  0.331329   11.760232    0.476951  
min     18.200000                  0.078000   21.000000    0.000000  
```

```
25%      27.500000                    0.243750   24.000000    0.000000
50%      32.300000                    0.372500   29.000000    0.000000
75%      36.600000                    0.626250   41.000000    1.000000
max      67.100000                    2.420000   81.000000    1.000000
```

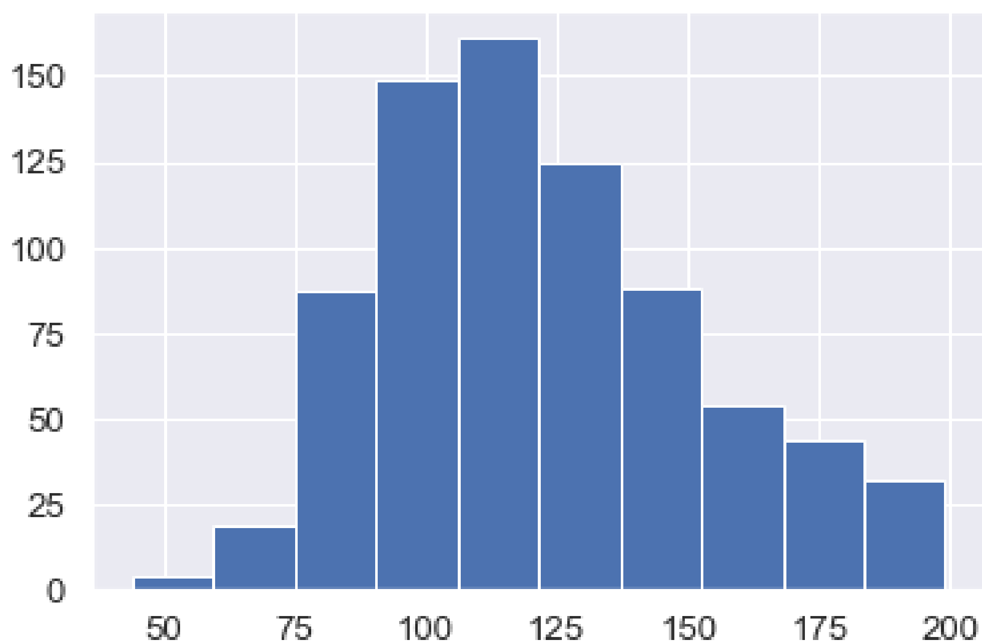
(2) Visually explore these variables using histogram and treat the missing values accordingly:

[8]:
```
data['Glucose'].value_counts().head(7)
```

[8]:
```
99.0     17
100.0    17
111.0    14
129.0    14
125.0    14
106.0    14
112.0    13
Name: Glucose, dtype: int64
```

[9]:
```
plt.hist(data['Glucose'])
```

[9]:
```
(array([  4.,  19.,  87., 149., 161., 125.,  88.,  54.,  44.,  32.]),
 array([ 44. ,  59.5,  75. ,  90.5, 106. , 121.5, 137. , 152.5, 168. ,
        183.5, 199. ]),
 <BarContainer object of 10 artists>)
```

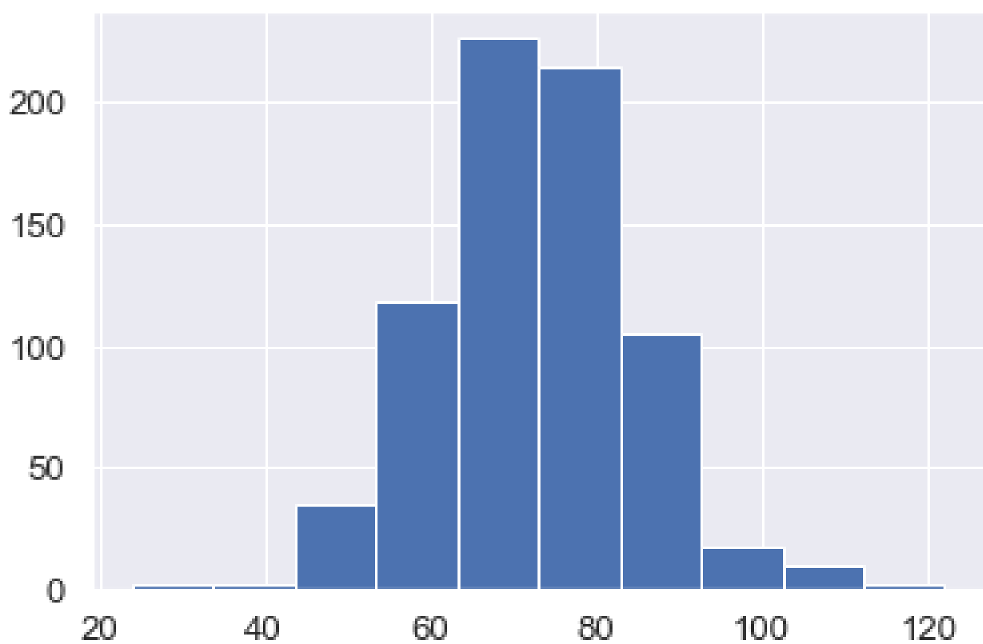


[10]:
```
data['BloodPressure'].value_counts().head(7)
```

[10]:
```
70.0    57
74.0    52
78.0    45
68.0    45
72.0    44
64.0    43
80.0    40
Name: BloodPressure, dtype: int64
```

[11]:
```
plt.hist(data['BloodPressure'])
```

[11]:
```
(array([  3.,   2.,  35., 118., 226., 214., 105.,  18.,  10.,   2.]),
 array([ 24. ,  33.8,  43.6,  53.4,  63.2,  73. ,  82.8,  92.6, 102.4,
        112.2, 122. ]),
 <BarContainer object of 10 artists>)
```



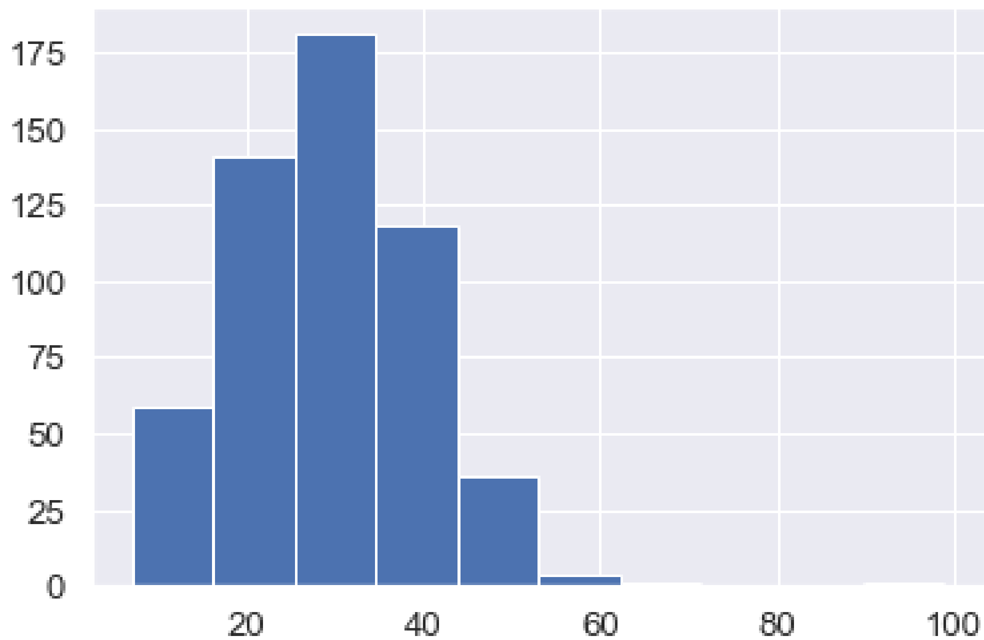
[12]:
```
data['SkinThickness'].value_counts().head(7)
```

[12]:
```
32.0    31
30.0    27
27.0    23
23.0    22
28.0    20
```

```
33.0    20
18.0    20
Name: SkinThickness, dtype: int64
```

[13]:
```
plt.hist(data['SkinThickness'])
```

[13]:
```
(array([ 59., 141., 181., 118.,  36.,   4.,   1.,   0.,   0.,   1.]),
 array([ 7. , 16.2, 25.4, 34.6, 43.8, 53. , 62.2, 71.4, 80.6, 89.8, 99. ]),
 <BarContainer object of 10 artists>)
```



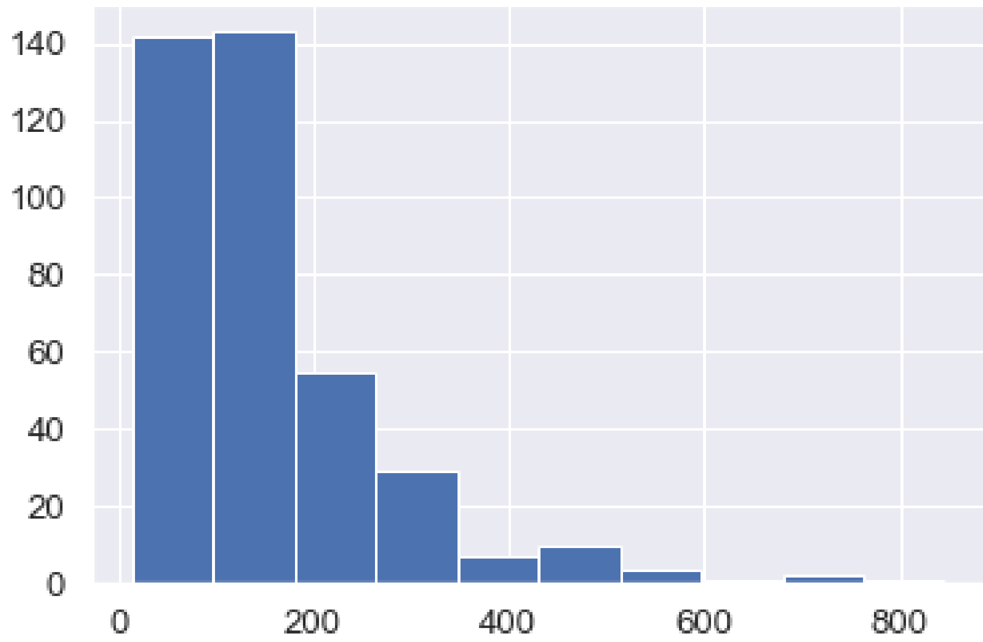
[14]:
```
data['Insulin'].value_counts().head(7)
```

[14]:
```
105.0    11
130.0     9
140.0     9
120.0     8
94.0      7
180.0     7
100.0     7
Name: Insulin, dtype: int64
```

[15]:
```
plt.hist(data['Insulin'])
```

[15]:
```
(array([142., 143.,  55.,  29.,   7.,  10.,   4.,   1.,   2.,   1.]),
 array([ 14. ,  97.2, 180.4, 263.6, 346.8, 430. , 513.2, 596.4, 679.6,
```

```
        762.8, 846. ]),
 <BarContainer object of 10 artists>)
```

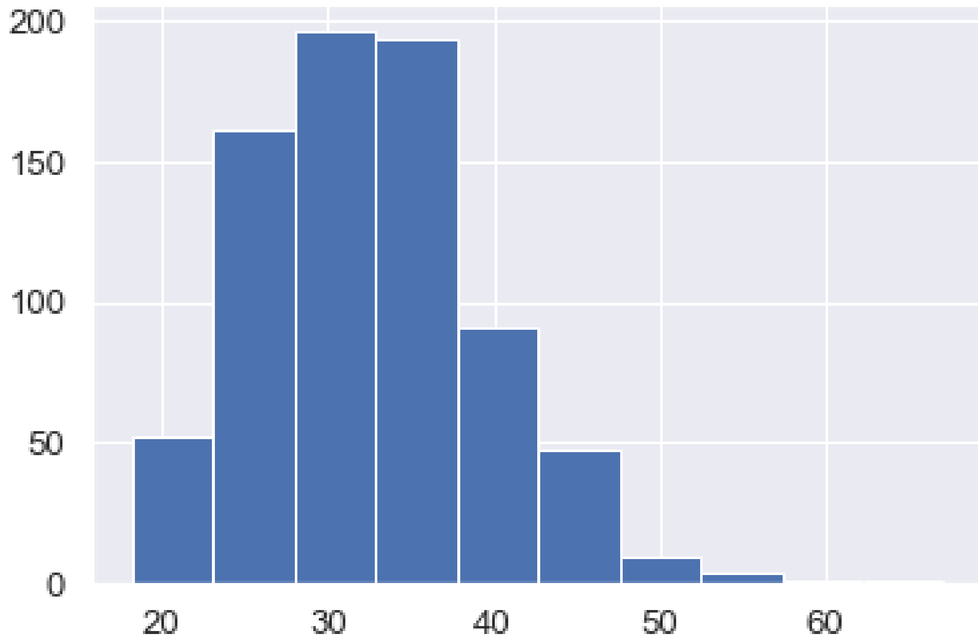


[16]:
```
data['BMI'].value_counts().head(7)
```

[16]:
```
32.0    13
31.6    12
31.2    12
32.4    10
33.3    10
32.9     9
32.8     9
Name: BMI, dtype: int64
```

[17]:
```
plt.hist(data['BMI'])
```

[17]:
```
(array([ 52., 161., 196., 193.,  91.,  48.,  10.,   4.,   1.,   1.]),
 array([18.2 , 23.09, 27.98, 32.87, 37.76, 42.65, 47.54, 52.43, 57.32,
        62.21, 67.1 ]),
 <BarContainer object of 10 artists>)
```



From above histograms, it is clear that **Insulin** has highly skewed data distribution and remaining 4 variables have relatively balanced data distribution therefore we will treat missing values in these 5 variables as below:-

- Glucose - replace missing values with mean of values.
- BloodPressure - replace missing values with mean of values.
- SkinThickness - replace missing values with mean of values.
- Insulin - replace missing values with median of values.
- BMI - replace missing values with mean of values.

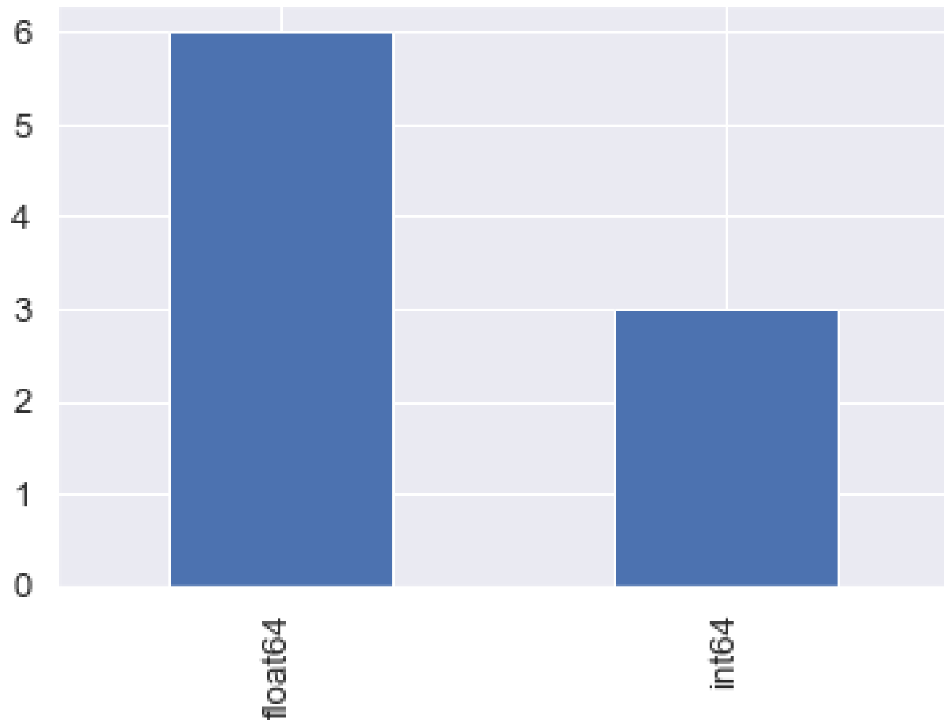
[18]:
```
data['Insulin'] = data['Insulin'].fillna(data['Insulin'].median())
```

[19]:
```
cols_mean_for_null = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI']
data[cols_mean_for_null] = data[cols_mean_for_null].
 ↪fillna(data[cols_mean_for_null].mean())
```

(3) Create a count (frequency) plot describing the data types and count of variables:

[20]:
```
data.dtypes.value_counts().plot(kind='bar')
```

[20]: <AxesSubplot:>

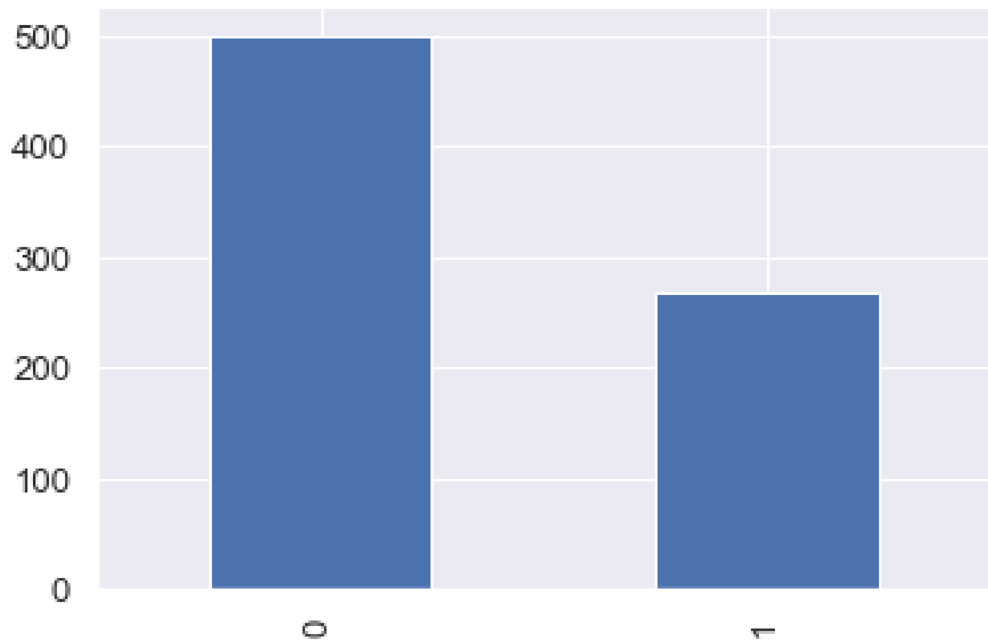


0.1.1 Data Exploration:

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

[21]:
```
data['Outcome'].value_counts().plot(kind='bar')
data['Outcome'].value_counts()
```

[21]:
```
0    500
1    268
Name: Outcome, dtype: int64
```



Since classes in **Outcome** is little skewed so we will generate new samples using **SMOTE (Synthetic Minority Oversampling Technique)** for the class '1' which is under-represented in our data. We will use SMOTE out of many other techniques available since:

- It generates new samples by interpolation.
- It doesn't duplicate data.

[22]:
```
data_x = data.drop('Outcome', axis=1)
data_y = data['Outcome']
print(data_x.shape, data_y.shape)
```

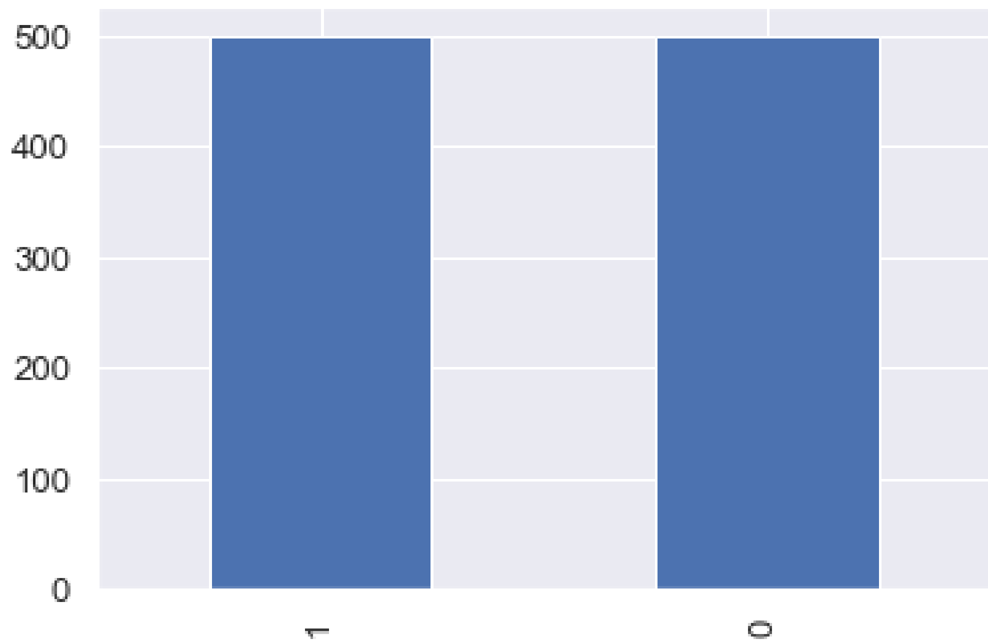
```
(768, 8) (768,)
```

[23]:
```
data_x_resampled, data_y_resampled = SMOTE(random_state=108).
 ↪fit_resample(data_x, data_y)
print(data_x_resampled.shape, data_y_resampled.shape)
```

```
(1000, 8) (1000,)
```

[24]:
```
data_y_resampled.value_counts().plot(kind='bar')
data_y_resampled.value_counts()
```

[24]:
```
1    500
0    500
Name: Outcome, dtype: int64
```



(5) Create scatter plots between the pair of variables to understand the relationships. Describe findigs:

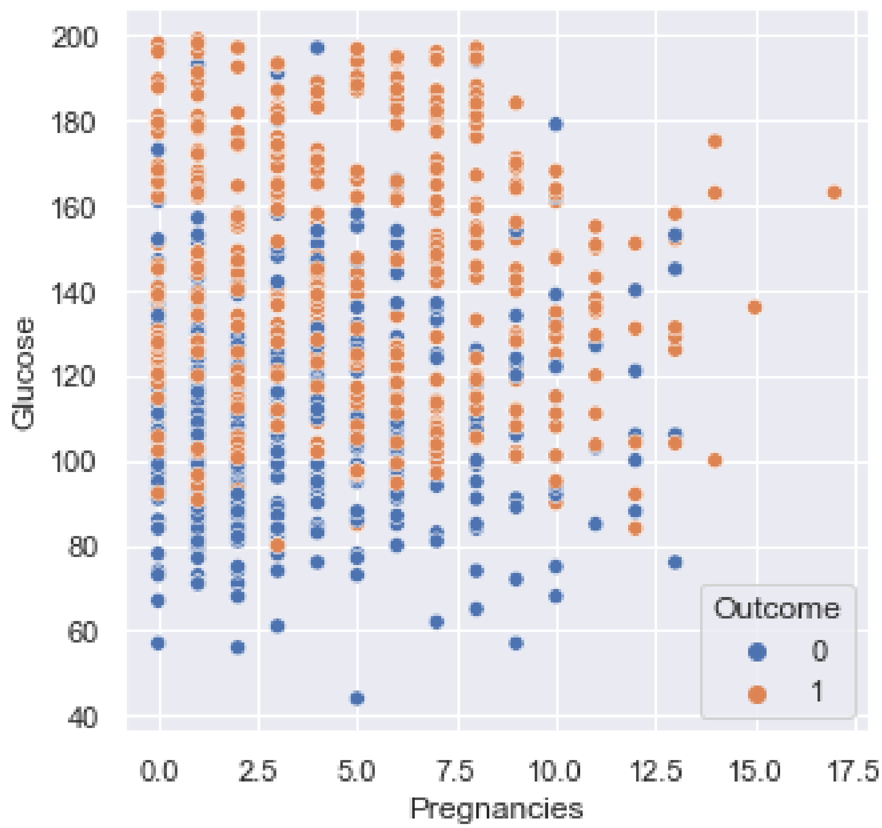
[25]:
```
data_resampled = pd.concat([data_x_resampled, data_y_resampled], axis=1)
data_resampled
```

[25]:
```
     Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
0              6  148.000000      72.000000      35.000000  125.000000
1              1   85.000000      66.000000      29.000000  125.000000
2              8  183.000000      64.000000      29.153420  125.000000
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      29.153420  125.000000
996            0  138.913540      69.022720      27.713033  127.283849
997           10  131.497740      66.331574      33.149837  125.000000
998            0  105.571347      83.238205      29.153420  125.000000
999            0  127.727025     108.908879      44.468195  129.545366

           BMI  DiabetesPedigreeFunction  Age  Outcome
0    33.600000                  0.627000   50        1
1    26.600000                  0.351000   31        0
2    23.300000                  0.672000   32        1
3    28.100000                  0.167000   21        0
4    43.100000                  2.288000   33        1
..         …                       …   …      …
```

```
995  42.767110                   0.726091   29         1
996  39.177649                   0.703702   24         1
997  45.820819                   0.498032   38         1
998  27.728596                   0.649204   60         1
999  65.808840                   0.308998   26         1

[1000 rows x 9 columns]
```

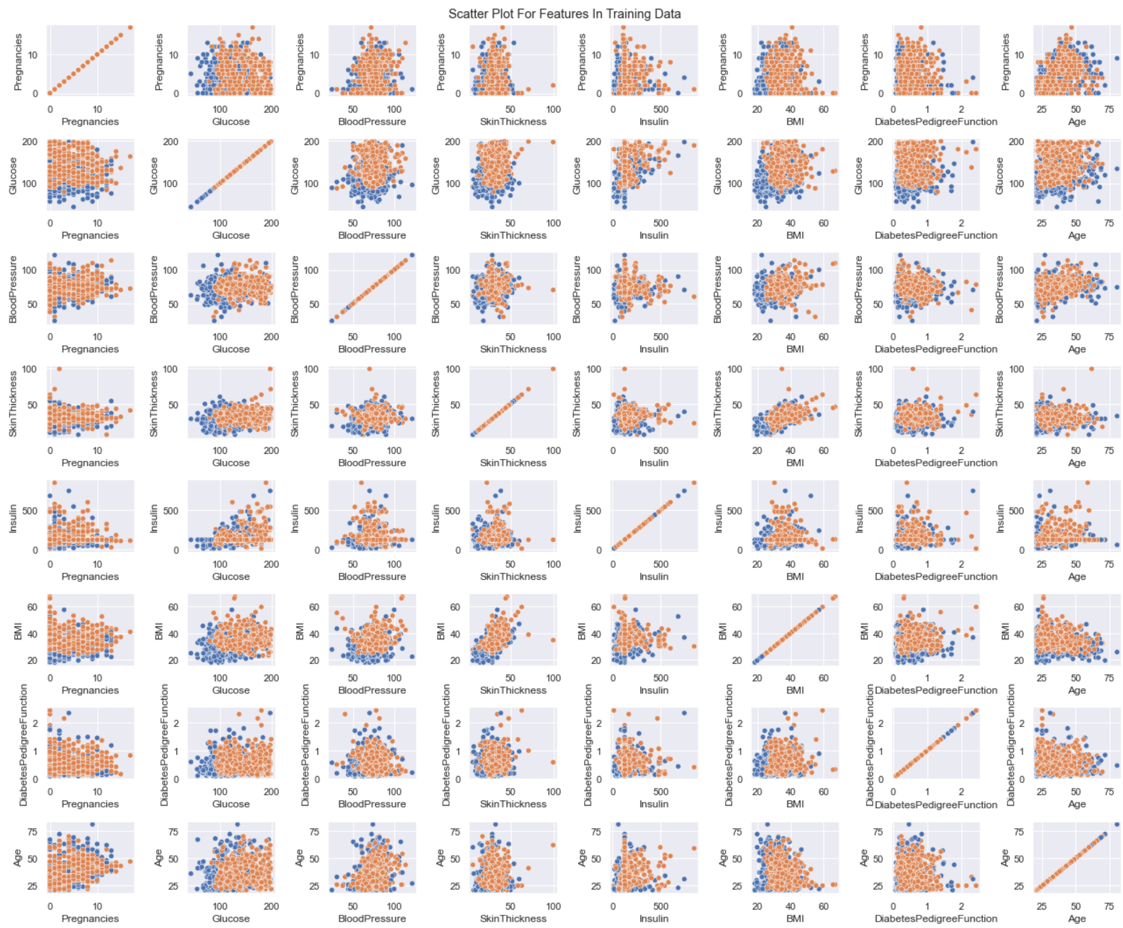
[26]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.scatterplot(x="Pregnancies", y="Glucose", data=data_resampled,␣
↪hue="Outcome")
```

[26]: <AxesSubplot:xlabel='Pregnancies', ylabel='Glucose'>



[27]:
```
fig, axes = plt.subplots(8,8, figsize=(18,15))
fig.suptitle('Scatter Plot For Features In Training Data')

for i, col_y in enumerate(data_x_resampled.columns):
    for j, col_x in enumerate(data_x_resampled.columns):
```

```
        sns.scatterplot(ax=axes[i, j], x=col_x, y=col_y, data=data_resampled,␣
↪hue="Outcome", legend = False)

plt.tight_layout()
```



We have some intresting observation from above scatter plot of pair of features: - **Glucose** alone is impressively good to distiguish between the **Outcome** classes. - **Age** alone is also able to distinguish between classes to some extent. - It seems none of pairs in the dataset is able to clearly distinguish between the **Outcome** classes. - We need to use combination of features to build model for prediction of classes in **Outcome**.

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

[28]:
```
data_x_resampled.corr()
```

[28]:

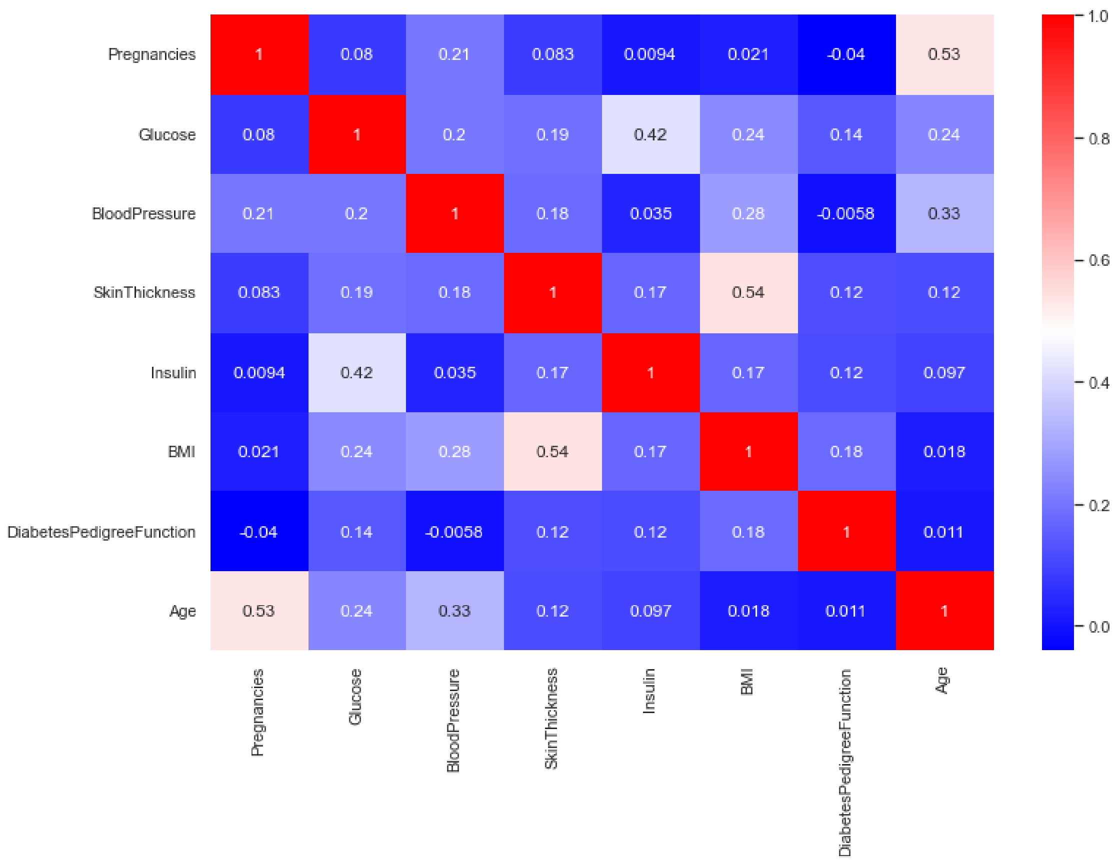
	Pregnancies	Glucose	BloodPressure	SkinThickness \
Pregnancies	1.000000	0.079953	0.205232	0.082752
Glucose	0.079953	1.000000	0.200717	0.189776
BloodPressure	0.205232	0.200717	1.000000	0.176496

```
SkinThickness               0.082752  0.189776       0.176496       1.000000
Insulin                     0.009365  0.418830       0.034861       0.170719
BMI                         0.021006  0.242501       0.277565       0.538207
DiabetesPedigreeFunction   -0.040210  0.138945      -0.005850       0.120799
Age                         0.532660  0.235522       0.332015       0.117644

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               0.009365  0.021006                 -0.040210
Glucose                   0.418830  0.242501                  0.138945
BloodPressure             0.034861  0.277565                 -0.005850
SkinThickness             0.170719  0.538207                  0.120799
Insulin                   1.000000  0.168702                  0.115187
BMI                       0.168702  1.000000                  0.177915
DiabetesPedigreeFunction  0.115187  0.177915                  1.000000
Age                       0.096940  0.017529                  0.010532

                               Age
Pregnancies               0.532660
Glucose                   0.235522
BloodPressure             0.332015
SkinThickness             0.117644
Insulin                   0.096940
BMI                       0.017529
DiabetesPedigreeFunction  0.010532
Age                       1.000000
```

[29]:

```
plt.figure(figsize=(12,8))
sns.heatmap(data_x_resampled.corr(), cmap='bwr', annot=True)
```

[29]: <AxesSubplot:>



It appears from correlation matrix and heatmap that the there exists significant correlation between some pairs such as - - Age-Pregnancies - BMI-SkinThickness

Also we can see that no pair of variables have negative correlation.

0.2 Week 2:

0.2.1 Data Modeling:

(1) Device strategies for model building. It is important to decide the right validation framework. Express your thought process:

Answer: Since this is a classification problem, we will be building all popular classification models for our training data and then compare performance of each model on test data to accurately predict target variable (Outcome).

1) Logistic Regression
2) Decison Tree
3) RandomForest Classifier
4) K-Nearest Neighbour (KNN)
5) Support Vector Machine (SVM)

6) Naive Bayes
7) Ensemble Learning -> Boosting -> Adaptive Boosting
8) Ensemble Learning -> Boosting -> Gradient Boosting (XGBClassifier)

We will use **GridSearchCV** with Cross Validation (CV) = 5 for training and testing model wich will give us insight about model performance on versatile data. It helps to loop through predefined hyper parameters and fit model on training set. GridSearchCV performs hyper parameter tunning which will give us optimal hyper parameters for each of the model. We will again train model with these optimized hyper parameters and then predict test data to get metrics for comparing all models.

Performing Train - Test split on input data (To train and test model without Cross Validation and Hyper Parameter Tunning):

[30]:
```
from sklearn.model_selection import train_test_split, KFold, RandomizedSearchCV
from sklearn.metrics import accuracy_score, average_precision_score, f1_score,␣
↪confusion_matrix, classification_report, auc, roc_curve, roc_auc_score
```

[31]:
```
x_train, x_test, y_train, y_test = train_test_split(data_x_resampled,␣
↪data_y_resampled, test_size=0.15, random_state=10)
```

[32]:
```
x_train.shape, x_test.shape
```

[32]:
```
((850, 8), (150, 8))
```

Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

[33]:
```
models = []
model_accuracy = []
model_f1 = []
model_auc = []
```

1) Logistic Regression:

[34]:
```
from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression(max_iter=300)
```

[35]:
```
lr1.fit(x_train, y_train)
```

[35]:
```
LogisticRegression(max_iter=300)
```

[37]:
```
lr1.score(x_train, y_train)
```

[37]:
```
0.7294117647058823
```

[38]:
```
lr1.score(x_test, y_test)
```

[38]:
```
0.76
```

Performance evaluation and optimizing parameters using GridSearchCV: Logistic regression does not really have any critical hyperparameters to tune. However we will try to optimize one of its parameters 'C' with the help of GridSearchCV. So we have set this parameter as a list of values form which GridSearchCV will select the best value of parameter.

[39]:
```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

[40]:
```
parameters = {'C':np.logspace(-5, 5, 50)}
```

[41]:
```
gs_lr = GridSearchCV(lr1, param_grid = parameters, cv=5, verbose=0)
gs_lr.fit(data_x_resampled, data_y_resampled)
```

[41]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=300),
             param_grid={'C': array([1.00000000e-05, 1.59985872e-05,
2.55954792e-05, 4.09491506e-05,
       6.55128557e-05, 1.04811313e-04, 1.67683294e-04, 2.68269580e-04,
       4.29193426e-04, 6.86648845e-04, 1.09854114e-03, 1.75751062e-03,
       2.81176870e-03, 4.49843267e-03, 7.19685673e-03, 1.15139540e-02,
       1.84206997e-02, 2.94705170e…
       7.90604321e-01, 1.26485522e+00, 2.02358965e+00, 3.23745754e+00,
       5.17947468e+00, 8.28642773e+00, 1.32571137e+01, 2.12095089e+01,
       3.39322177e+01, 5.42867544e+01, 8.68511374e+01, 1.38949549e+02,
       2.22299648e+02, 3.55648031e+02, 5.68986603e+02, 9.10298178e+02,
       1.45634848e+03, 2.32995181e+03, 3.72759372e+03, 5.96362332e+03,
       9.54095476e+03, 1.52641797e+04, 2.44205309e+04, 3.90693994e+04,
       6.25055193e+04, 1.00000000e+05])})
```

[42]:
```
gs_lr.best_params_
```

[42]:
```
{'C': 13.257113655901108}
```

[43]:
```
gs_lr.best_score_
```

[43]:
```
0.738
```

[44]:
```
lr2 = LogisticRegression(C=13.257113655901108, max_iter=300)
```

[45]:
```
lr2.fit(x_train, y_train)
```

[45]:
```
LogisticRegression(C=13.257113655901108, max_iter=300)
```

[46]:
```
lr2.score(x_train, y_train)
```

[46]:
```
0.731764705882353
```

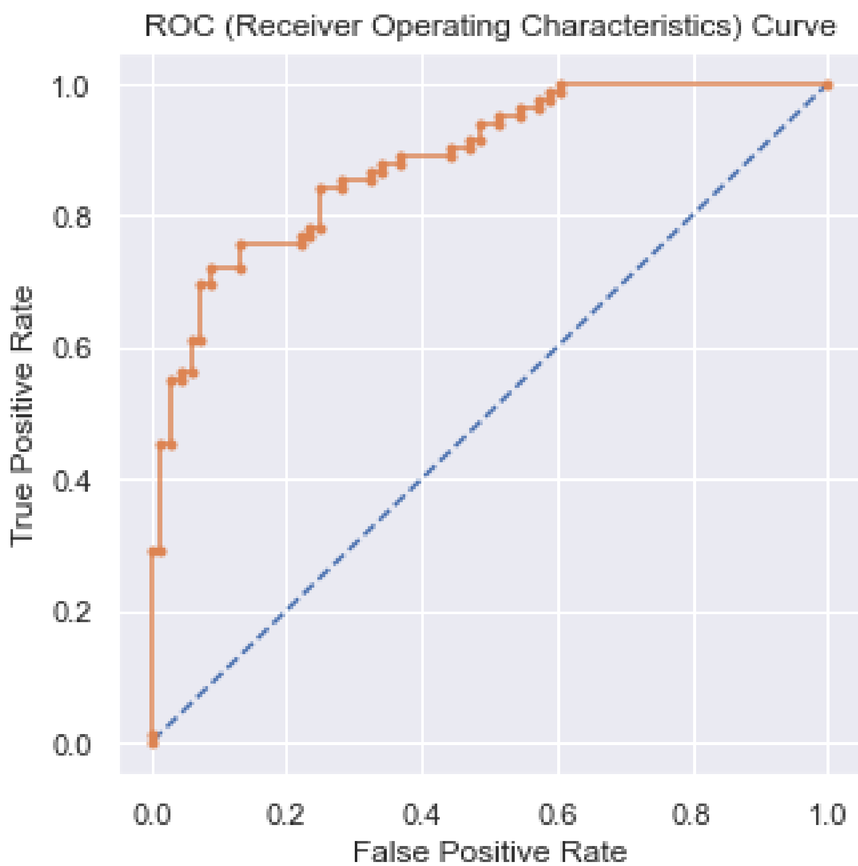
[47]:
```
lr2.score(x_test, y_test)
```

[47]:
```
0.7733333333333333
```

[48]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curves)

probs = lr2.predict_proba(x_test)                   # Predict probabilities
probs = probs[:,1]                                  # keep probabilities for the␣
 ↪positive outcome only

auc_lr = roc_auc_score(y_test, probs)               # calculate AUC
print('AUC: %.3f' %auc_lr)
fpr, tpr, thresholds = roc_curve(y_test, probs)     # calculate roc curve
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")
```

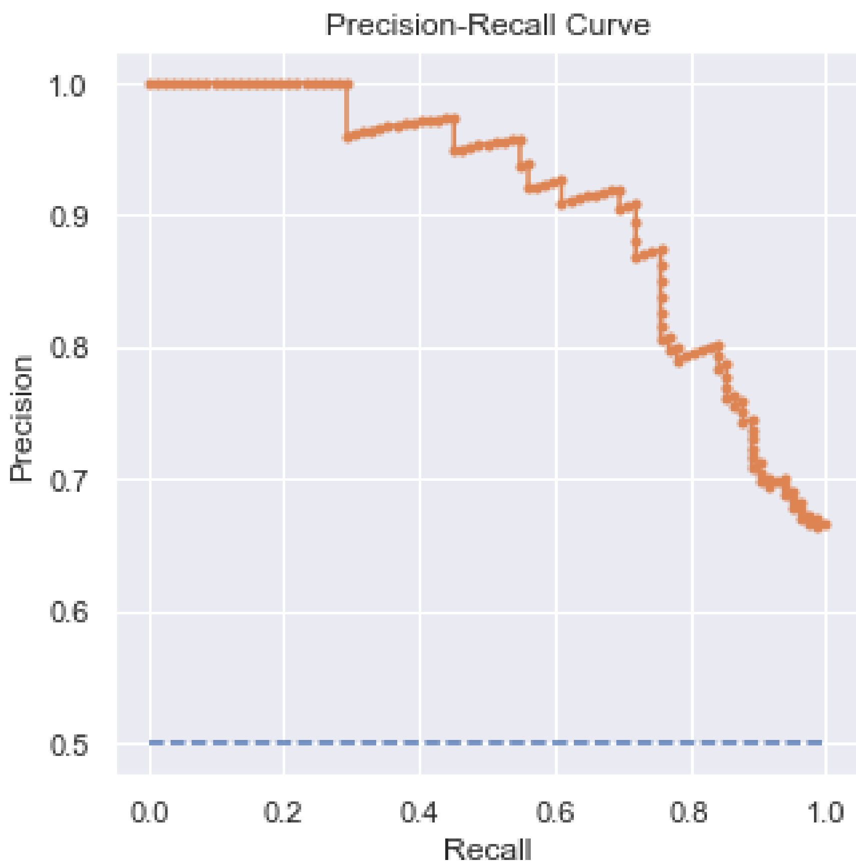
```
AUC: 0.884
```

[48]:
```
Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')
```



[50]:
```
# Precision Recall Curve
from sklearn.metrics import precision_recall_curve
pred_y_test = lr2.predict(x_test)                                         # predict␣
 ↪class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                        #␣
 ↪calculate F1 score
auc_lr_pr = auc(recall, precision)                                        #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                               #␣
 ↪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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.790 auc_pr=0.908 ap=0.909
```



[52]:
```
models.append('LR')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_lr)
```

2) Decision Tree:

[53]:
```
from sklearn.tree import DecisionTreeClassifier
dt1 = DecisionTreeClassifier(random_state=0)
```

[54]:
```
dt1.fit(x_train, y_train)
```

[54]: DecisionTreeClassifier(random_state=0)

[55]:
```
dt1.score(x_train, y_train)
```

[55]: 1.0

[56]:
```
dt1.score(x_test, y_test)
```

[56]: 0.7733333333333333

Performance evaluation and optimizing parameters using GridSearchCV:

[57]:
```
parameters = {
    'max_depth':[1,2,3,4,5,None]
}
```

[59]:
```
gs_dt = GridSearchCV(dt1, param_grid = parameters, cv=5, verbose=0)
gs_dt.fit(data_x_resampled, data_y_resampled)
```

[59]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [1, 2, 3, 4, 5, None]})
```

[60]:
```
gs_dt.best_params_
```

[60]:
```
{'max_depth': 4}
```

[61]:
```
gs_dt.best_score_
```

[61]: 0.76

[62]:
```
dt1.feature_importances_
```

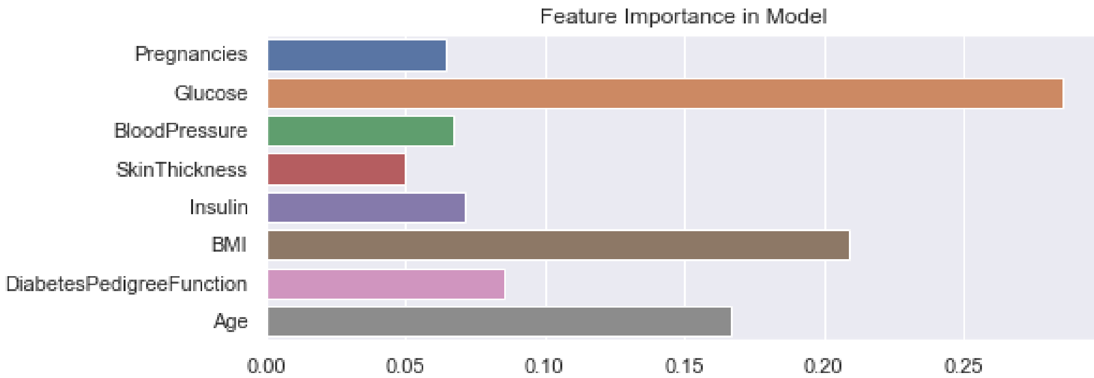
[62]:
```
array([0.06452226, 0.28556999, 0.06715314, 0.04979714, 0.07150365,
       0.20905992, 0.08573109, 0.16666279])
```

[63]:
```
x_train.columns
```

[63]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

[64]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=x_train.columns, x=dt1.feature_importances_)
plt.title("Feature Importance in Model")
```

[64]:
```
Text(0.5, 1.0, 'Feature Importance in Model')
```



[65]:
```
dt2 = DecisionTreeClassifier(max_depth=4)
```

[66]:
```
dt2.fit(x_train, y_train)
```

[66]: DecisionTreeClassifier(max_depth=4)

[67]:
```
dt2.score(x_train, y_train)
```

[67]: 0.8070588235294117

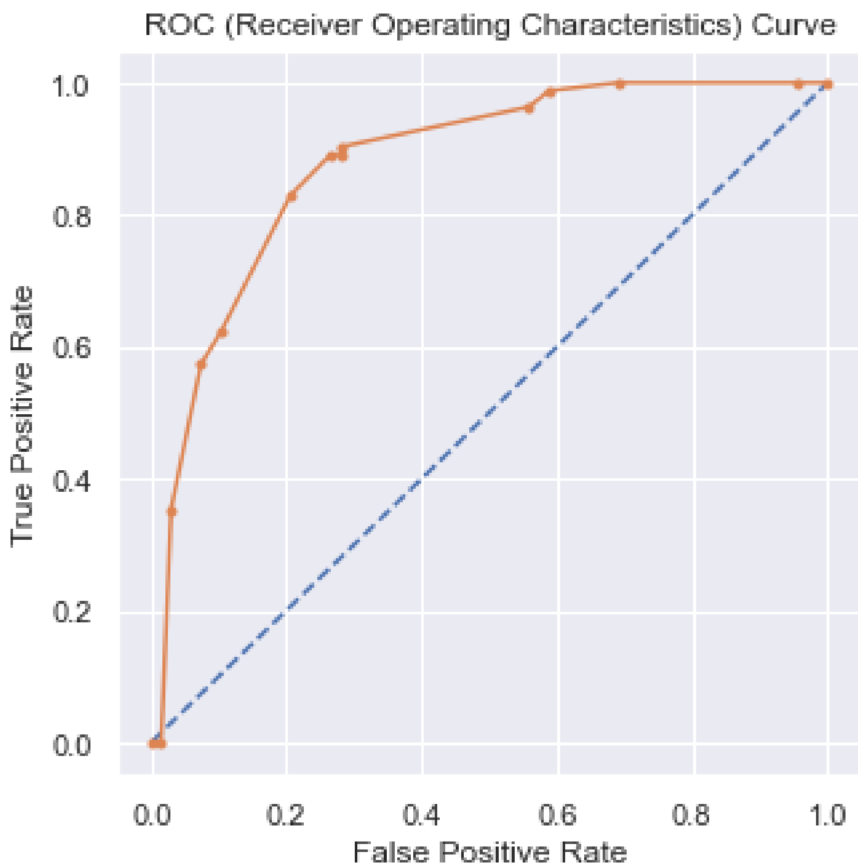
[68]:
```
dt2.score(x_test, y_test)
```

[68]: 0.82

[70]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curves)

probs = dt2.predict_proba(x_test)                  # Predict probabilities
probs = probs[:,1]                                 # keep probabilities for the␣
 ↪positive outcome only

auc_dt = roc_auc_score(y_test, probs)              # calculate AUC
print('AUC: %.3f' %auc_dt)
fpr, tpr, thresholds = roc_curve(y_test, probs)    # calculate roc curve
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")
```

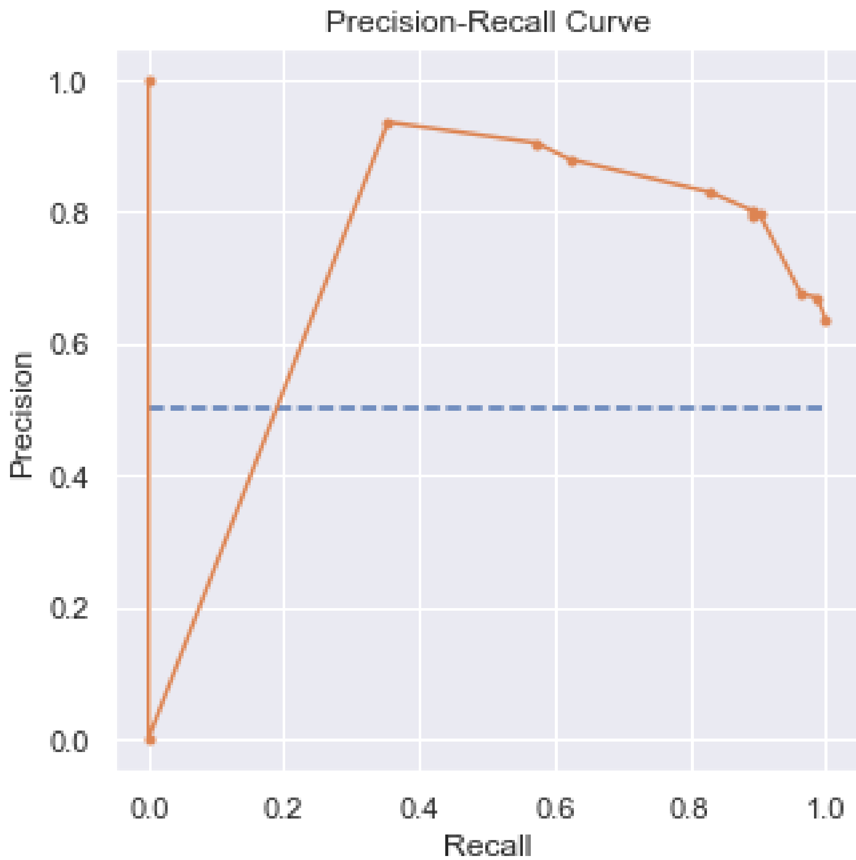
AUC: 0.879

[70]: Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')



[71]:
```
# Precision Recall Curve
pred_y_test = dt2.predict(x_test)                                        # predict␣
 ↪class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                       #␣
 ↪calculate F1 score
auc_dt_pr = auc(recall, precision)                                       #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                              #␣
 ↪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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.844 auc_pr=0.908 ap=0.868
```



[72]:
```
models.append('DT')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_dt)
```

3) RandomForest Classifier:

[73]:
```
from sklearn.ensemble import RandomForestClassifier
rf1 = RandomForestClassifier()
```

[74]:
```
rf1 = RandomForestClassifier(random_state=0)
```

[75]:
```
rf1.fit(x_train, y_train)
```

[75]:
```
RandomForestClassifier(random_state=0)
```

[76]:
```
rf1.score(x_test, y_test)
```

[76]: 0.8466666666666667

Performance evaluation and optimizing parameters using GridsearchCV:

[77]:
```
parameters = {
    'n_estimators':[50,10,150],
    'max_depth':[None,1,3,5,7],
    'min_samples_leaf':[1,3,5]
}
```

[78]:
```
gs_dt = GridSearchCV(estimator=rf1, param_grid=parameters, cv=5, verbose=0)
gs_dt.fit(data_x_resampled, data_y_resampled)
```

[78]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=0),
             param_grid={'max_depth': [None, 1, 3, 5, 7],
                         'min_samples_leaf': [1, 3, 5],
                         'n_estimators': [50, 10, 150]})
```

[79]:
```
gs_dt.best_params_
```

[79]:
```
{'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 150}
```

[80]:
```
gs_dt.best_score_
```

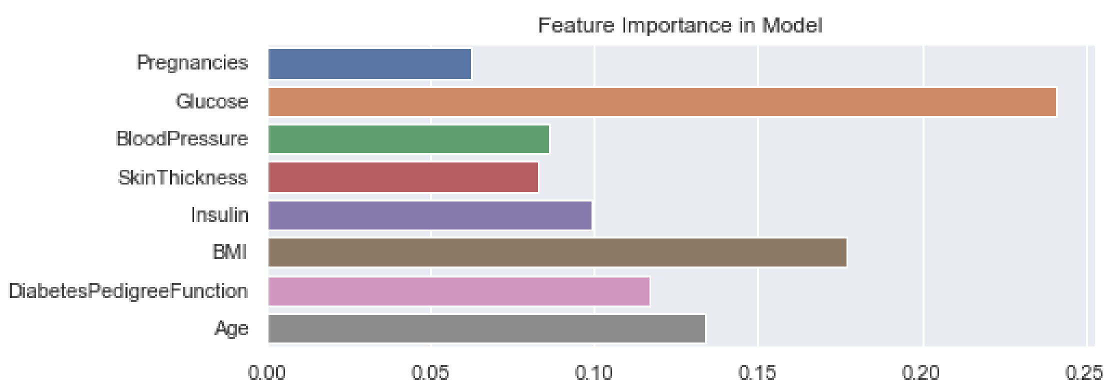
[80]: 0.813

[81]:
```
rf1.feature_importances_
```

[81]:
```
array([0.06264995, 0.24106573, 0.08653626, 0.08301549, 0.09945063,
       0.17678287, 0.11685244, 0.13364664])
```

[82]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=x_train.columns, x=rf1.feature_importances_)
plt.title("Feature Importance in Model")
```

[82]:
```
Text(0.5, 1.0, 'Feature Importance in Model')
```



[83]:
```
rf2 = RandomForestClassifier(max_depth=None, min_samples_leaf=1,␣
↪n_estimators=100)
```

[85]:
```
rf2.fit(x_train, y_train)
```

[85]:
```
RandomForestClassifier()
```

[86]:
```
rf2.score(x_train, y_train)
```

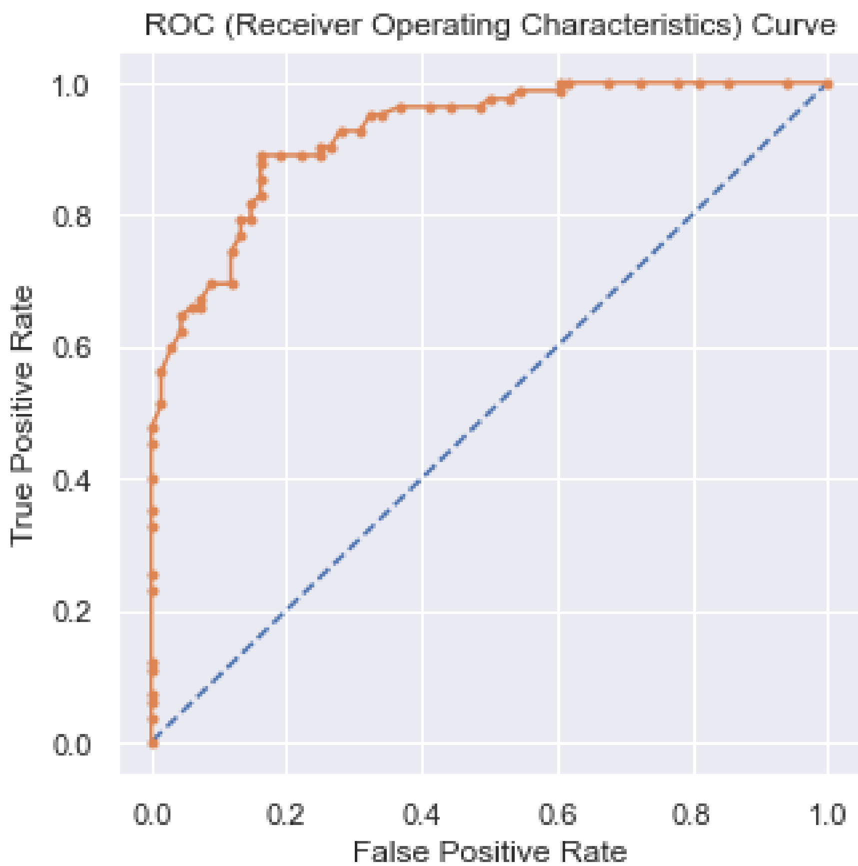
[86]:
```
1.0
```

[87]:
```
rf2.score(x_test, y_test)
```

[87]:
```
0.8466666666666667
```

[88]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)

probs = rf2.predict_proba(x_test)                    # predict probabilities
probs = probs[:, 1]                                  # keep probabilities for the␣
↪positive outcome only

auc_rf = roc_auc_score(y_test, probs)                # calculate AUC
print('AUC: %.3f' %auc_rf)
fpr, tpr, thresholds = roc_curve(y_test, probs)  # calculate roc curve
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: 0.924
```

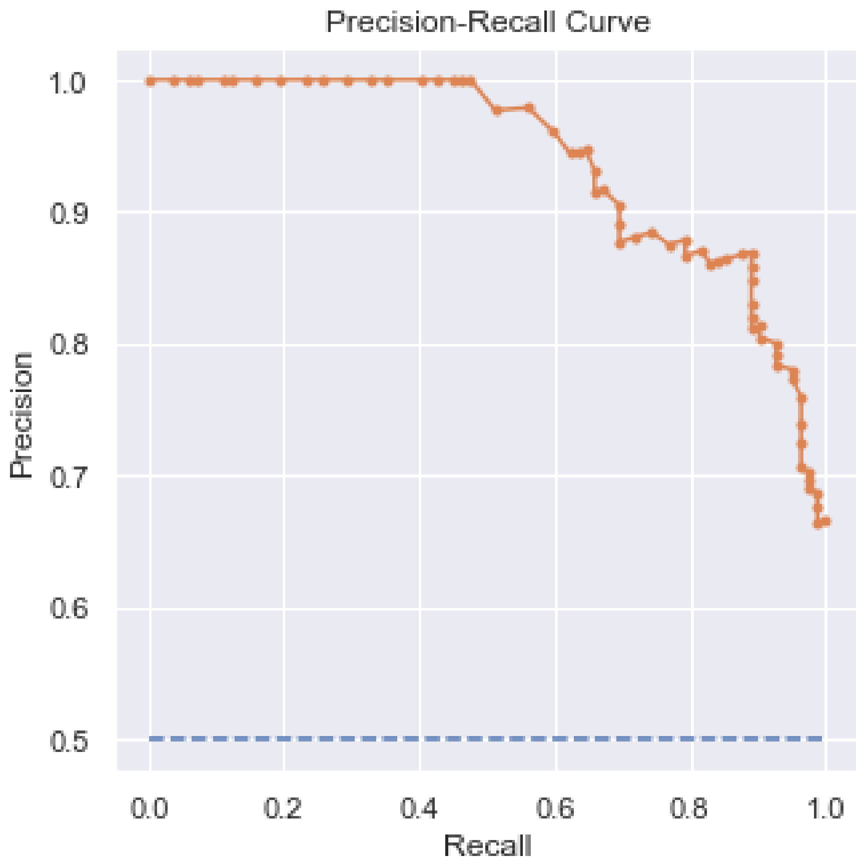


[89]:

```
# Precision Recall Curve

pred_y_test = rf2.predict(x_test)                                            # predict␣
 ↪class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                            #␣
 ↪calculate F1 score
auc_rf_pr = auc(recall, precision)                                            #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                                   #␣
 ↪calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_rf_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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision-Recall Curve");
```

```
f1=0.859 auc_pr=0.939 ap=0.938
```



[90]:
```
models.append('RF')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_dt)
```

4) K-Nearest Neighbour (KNN) Classification:

[91]:
```
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=3)
```

[92]:
```
knn1.fit(x_train, y_train)
```

[92]:
```
KNeighborsClassifier(n_neighbors=3)
```

[93]:
```
knn1.score(x_train, y_train)
```

[93]: 0.8835294117647059

[94]:
```
knn1.score(x_test, y_test)
```

[94]: 0.7866666666666666

Performance evaluation and optimizing parameters using GridSearchCV:

[97]:
```
knn_neighbors = [ i for i in range(2,16)]
parameters = {
    'n_neighbors':knn_neighbors
}
```

[98]:
```
gs_knn = GridSearchCV(estimator=knn1, param_grid=parameters, cv=5, verbose=0)
gs_knn.fit(data_x_resampled, data_y_resampled)
```

[98]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(n_neighbors=3),
             param_grid={'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
                                         14, 15]})
```

[99]:
```
gs_knn.best_params_
```

[99]: {'n_neighbors': 3}

[100]:
```
gs_knn.best_score_
```

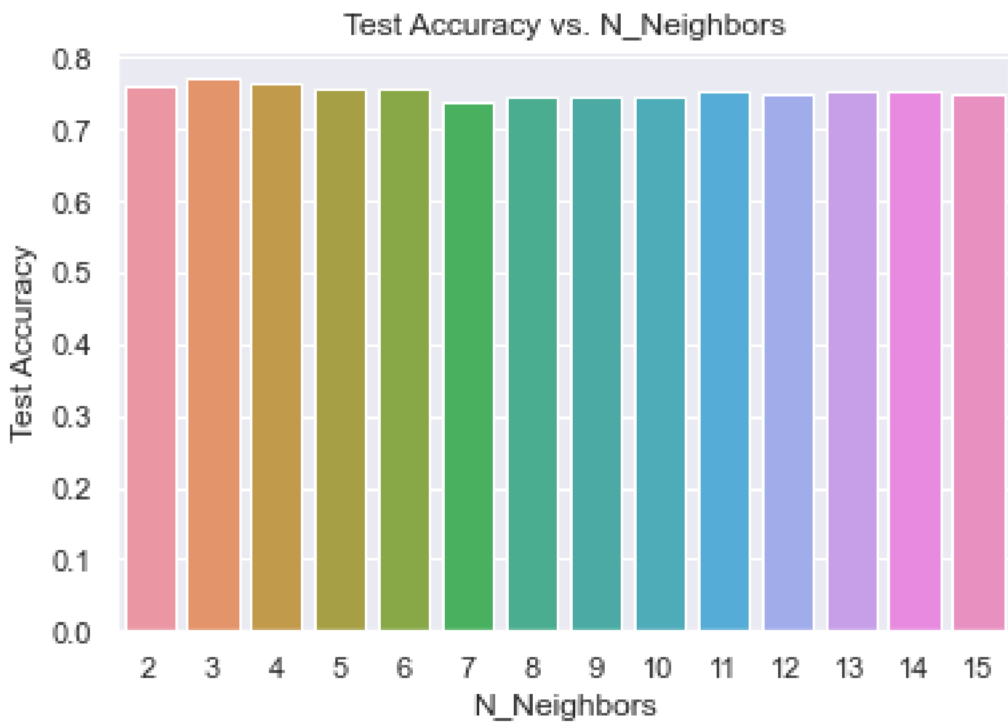
[100]: 0.771

[101]:
```
gs_knn.cv_results_['mean_test_score']
```

[101]:
```
array([0.76 , 0.771, 0.765, 0.757, 0.757, 0.739, 0.744, 0.746, 0.744,
       0.755, 0.751, 0.755, 0.754, 0.749])
```

[102]:
```
plt.figure(figsize=(6,4))
sns.barplot(x=knn_neighbors, y=gs_knn.cv_results_['mean_test_score'])
plt.xlabel("N_Neighbors")
plt.ylabel("Test Accuracy")
plt.title("Test Accuracy vs. N_Neighbors")
```

[102]: Text(0.5, 1.0, 'Test Accuracy vs. N_Neighbors')



[103]:
```
knn2 = KNeighborsClassifier(n_neighbors=3)
```

[104]:
```
knn2.fit(x_train, y_train)
```

[104]:
```
KNeighborsClassifier(n_neighbors=3)
```

[105]:
```
knn2.score(x_train, y_train)
```

[105]:
```
0.8835294117647059
```

[106]:
```
knn2.score(x_test, y_test)
```

[106]:
```
0.7866666666666666
```

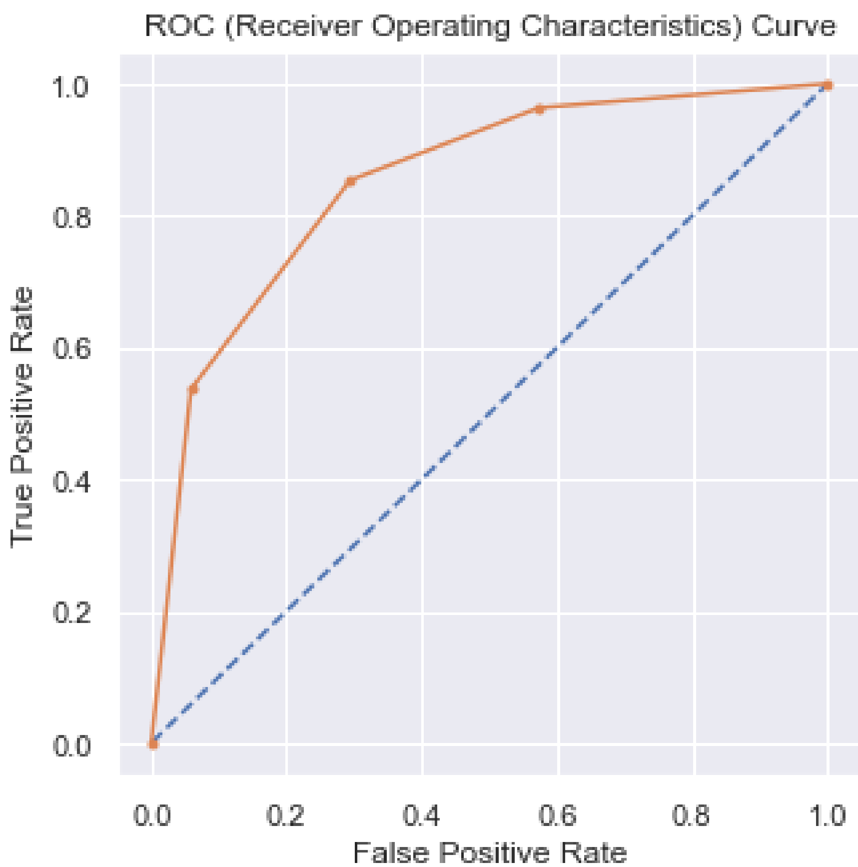
[107]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)

probs = knn2.predict_proba(x_test)              # predict probabilities
probs = probs[:, 1]                             # keep probabilities for the␣
↪positive outcome only

auc_knn = roc_auc_score(y_test, probs)          # calculate AUC
print('AUC: %.3f' %auc_knn)
fpr, tpr, thresholds = roc_curve(y_test, probs)  # calculate roc curve
```

```
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: 0.852
```

[107]:

```
Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')
```

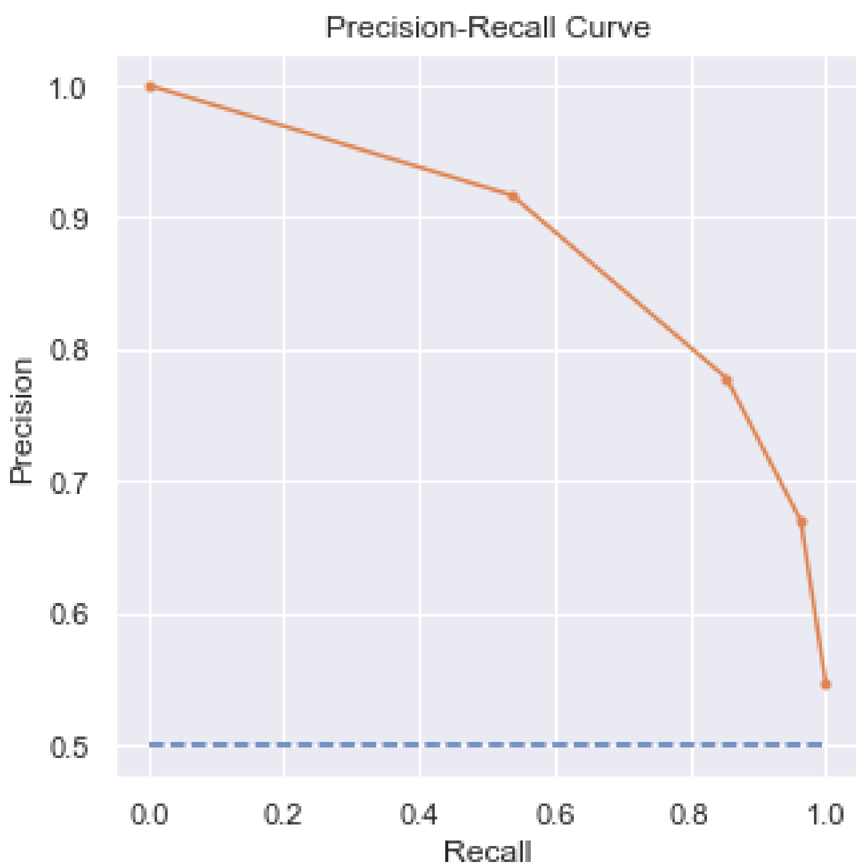


[108]:

```
# Precision Recall Curve

pred_y_test = knn2.predict(x_test)                                        #␣
 ↪predict class values
precision, recall, thresholds = precision_recall_curve(y_test, probs)  #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                      #␣
 ↪calculate F1 score
```

```
auc_knn_pr = auc(recall, precision)                                      #␣
↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                             #␣
↪calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_knn_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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.814 auc_pr=0.885 ap=0.832
```



[109]:
```
models.append('KNN')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_knn)
```

5) Support Vector Machine (SVM) Algorithm:

[110]:
```
from sklearn.svm import SVC
svm1 = SVC(kernel='rbf')
```

[112]:
```
svm1.fit(x_train, y_train)
```

[112]:
```
SVC()
```

[113]:
```
svm1.score(x_train, y_train)
```

[113]:
```
0.7282352941176471
```

[114]:
```
svm1.score(x_test, y_test)
```

[114]:
```
0.78
```

Performance evaluation and optimizing parameters using GridSearchCV:

[115]:
```
parameters = {
    'C':[1, 5, 10, 15, 20, 25],
    'gamma':[0.001, 0.005, 0.0001, 0.00001]
}
```

[116]:
```
gs_svm = GridSearchCV(estimator=svm1, param_grid=parameters, cv=5, verbose=0)
gs_svm.fit(data_x_resampled, data_y_resampled)
```

[116]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [1, 5, 10, 15, 20, 25],
                         'gamma': [0.001, 0.005, 0.0001, 1e-05]})
```

[117]:
```
gs_svm.best_params_
```

[117]:
```
{'C': 20, 'gamma': 0.005}
```

[118]:
```
gs_svm.best_score_
```

[118]:
```
0.8089999999999999
```

[119]:
```
svm2 = SVC(kernel='rbf', C=20, gamma=0.005, probability=True)
```

[120]:
```
svm2.fit(x_train, y_train)
```

[120]:
```
SVC(C=20, gamma=0.005, probability=True)
```

[121]:
```
svm2.score(x_train, y_train)
```

[121]:
```
0.9941176470588236
```

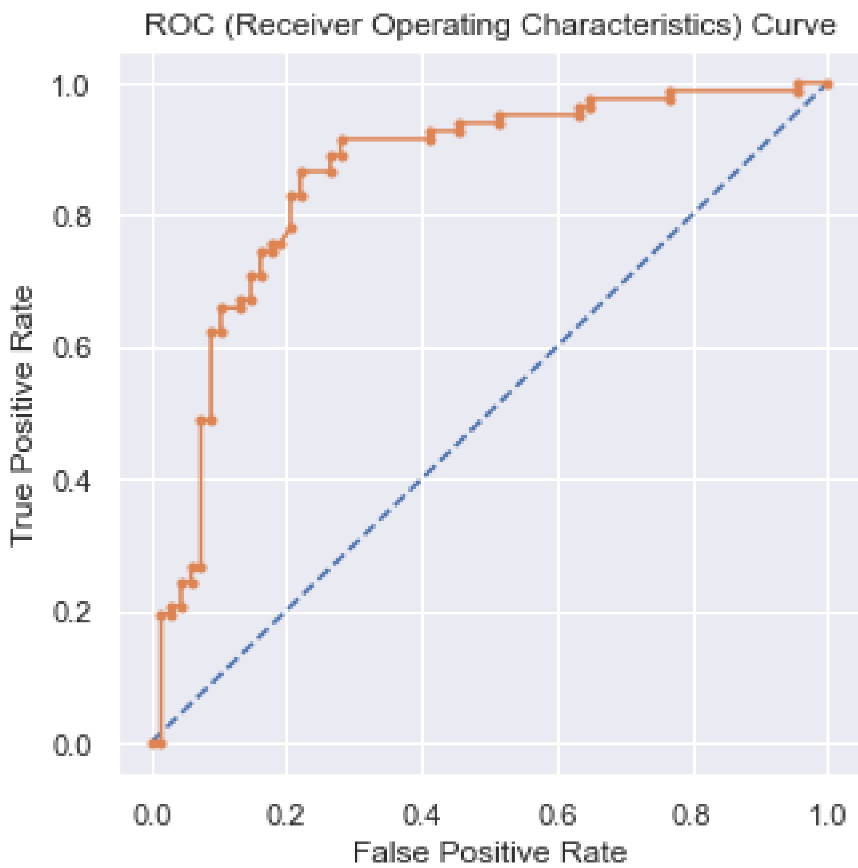
[122]:
```
svm2.score(x_test, y_test)
```

[122]: 0.8133333333333334

[123]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)

probs = svm2.predict_proba(x_test)               # predict probabilities
probs = probs[:, 1]                              # keep probabilities for the␣
 ↪positive outcome only

auc_svm = roc_auc_score(y_test, probs)           # calculate AUC
print('AUC: %.3f' %auc_svm)
fpr, tpr, thresholds = roc_curve(y_test, probs)  # calculate roc curve
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: 0.857

[123]: Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')

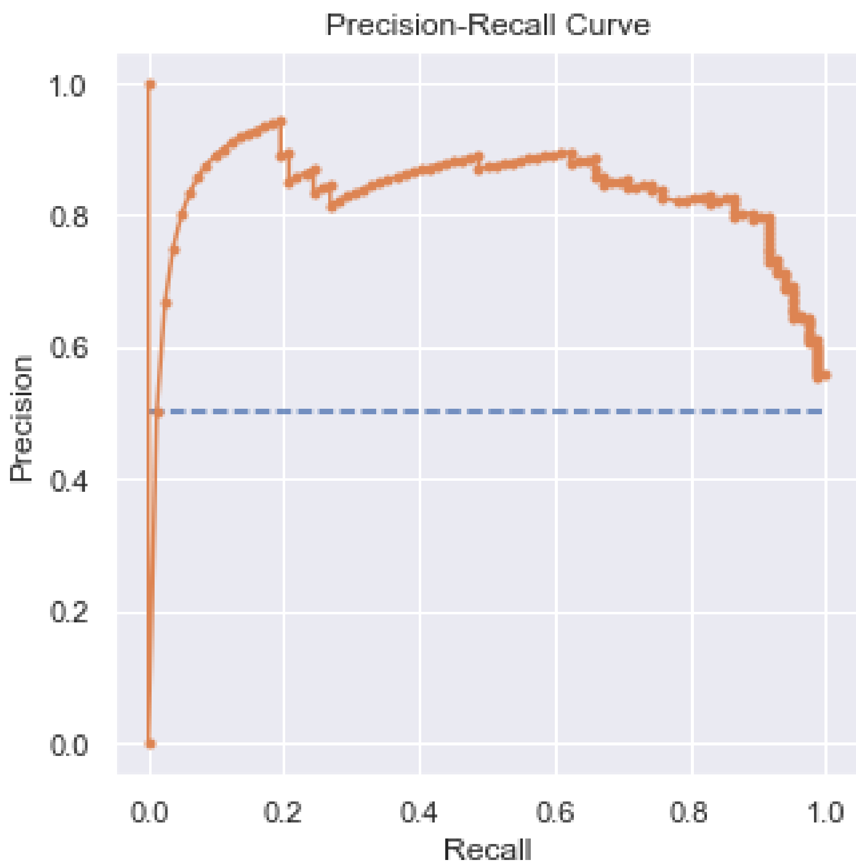


[124]:
```
# Precision Recall Curve

pred_y_test = svm2.predict(x_test)                                              # predict␣
 ↪class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                              #␣
 ↪calculate F1 score
auc_svm_pr = auc(recall, precision)                                             #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                                     #␣
 ↪calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_svm_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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision-Recall Curve")
```

```
f1=0.829 auc_pr=0.829 ap=0.836
```

[124]:
```
Text(0.5, 1.0, 'Precision-Recall Curve')
```



[125]:
```
models.append('SVM')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_svm)
```

6) Naive Bayes Algorithm:

[126]:
```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
gnb = GaussianNB()
```

[127]:
```
gnb.fit(x_train, y_train)
```

[127]:
```
GaussianNB()
```

[128]:
```
gnb.score(x_train, y_train)
```

[128]: 0.7294117647058823

[129]:
```
gnb.score(x_test, y_test)
```

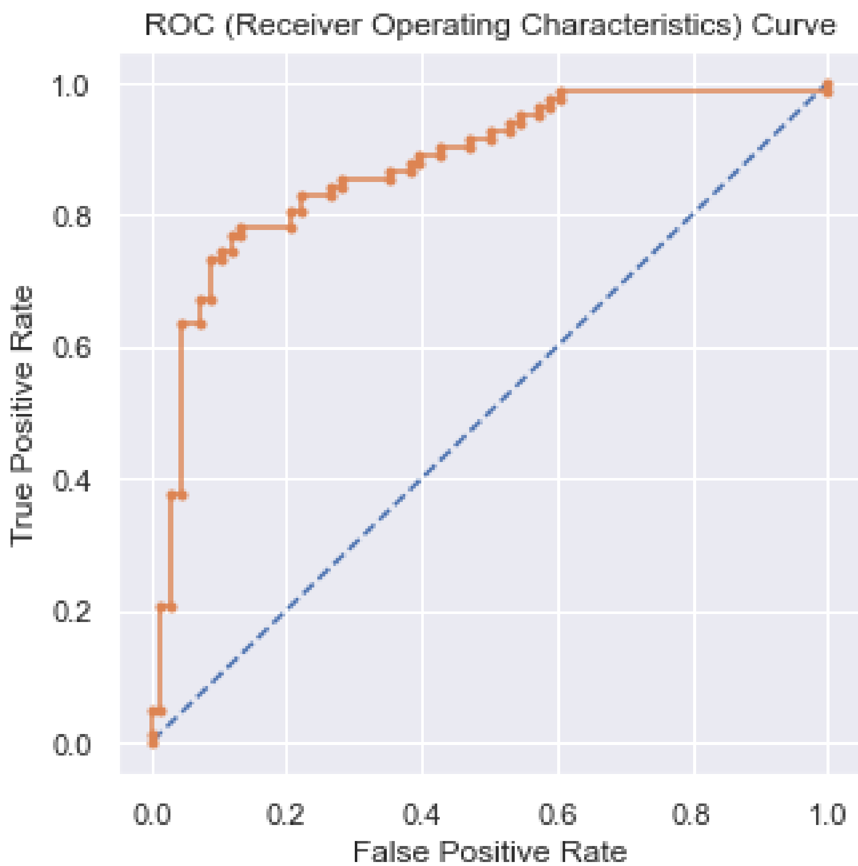
[129]: 0.8

Naive Bayes has almost no hyperparameters to tune, so it usually generalizes well.

[130]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)

probs = gnb.predict_proba(x_test)                # predict probabilities
probs = probs[:, 1]                              # keep probabilities for the␣
 ↪positive outcome only

auc_gnb = roc_auc_score(y_test, probs)            # calculate AUC
print('AUC: %.3f' %auc_gnb)
fpr, tpr, thresholds = roc_curve(y_test, probs)  # calculate roc curve
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: 0.873
```

[130]: Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')

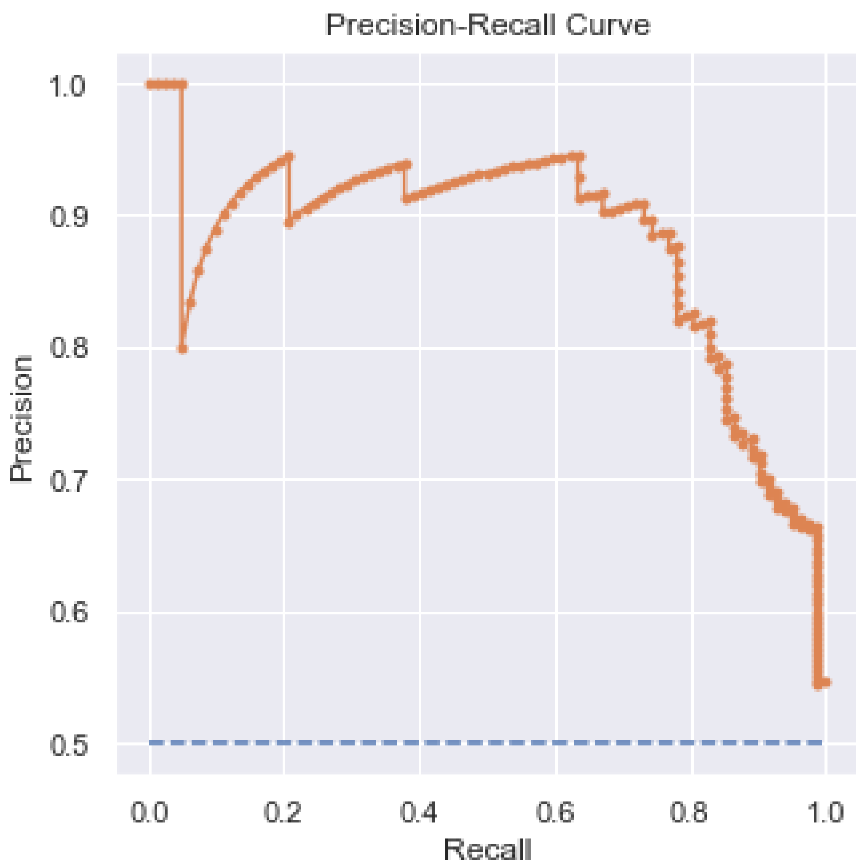


[131]:
```
# Precision Recall Curve

pred_y_test = gnb.predict(x_test)                                         # predict␣
 ↪class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                    #␣
 ↪calculate F1 score
auc_gnb_pr = auc(recall, precision)                                    #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                           #␣
 ↪calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_gnb_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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision-Recall Curve")
```

f1=0.819 auc_pr=0.879 ap=0.880

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



[132]:
```
models.append('GNB')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_gnb)
```

7) Ensemble Learning --> Boosting --> Adaptive Boosting:

[133]:
```
from sklearn.ensemble import AdaBoostClassifier
ada1 = AdaBoostClassifier(n_estimators=100)
```

[134]:
```
ada1.fit(x_train,y_train)
```

[134]: AdaBoostClassifier(n_estimators=100)

[135]:
```
ada1.score(x_train,y_train)
```

[135]: 0.8564705882352941

[136]:
```
ada1.score(x_test,y_test)
```

[136]: 0.7666666666666667

Performance evaluation and optimizing parameters using cross_val_score:

[137]:
```
parameters = {'n_estimators': [100,200,300,400,500,700,1000]}
```

[139]:
```
gs_ada = GridSearchCV(ada1, param_grid = parameters, cv=5, verbose=0)
gs_ada.fit(data_x_resampled, data_y_resampled)
```

[139]:
```
GridSearchCV(cv=5, estimator=AdaBoostClassifier(n_estimators=100),
             param_grid={'n_estimators': [100, 200, 300, 400, 500, 700, 1000]})
```

[140]:
```
gs_ada.best_params_
```

[140]: {'n_estimators': 500}

[141]:
```
gs_ada.best_score_
```

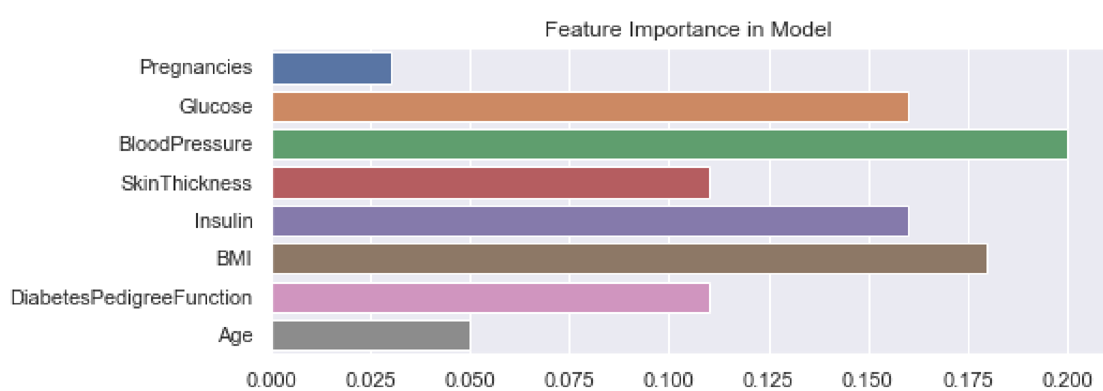
[141]: 0.785

[142]:
```
ada1.feature_importances_
```

[142]: array([0.03, 0.16, 0.2 , 0.11, 0.16, 0.18, 0.11, 0.05])

[143]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=x_train.columns, x=ada1.feature_importances_)
plt.title("Feature Importance in Model")
```

[143]: Text(0.5, 1.0, 'Feature Importance in Model')



[144]:

```
ada2 = AdaBoostClassifier(n_estimators=500)
```

[145]:

```
ada2.fit(x_train,y_train)
```

[145]: AdaBoostClassifier(n_estimators=500)

[146]:

```
ada2.score(x_train,y_train)
```

[146]: 0.9247058823529412

[147]:

```
ada2.score(x_test,y_test)
```

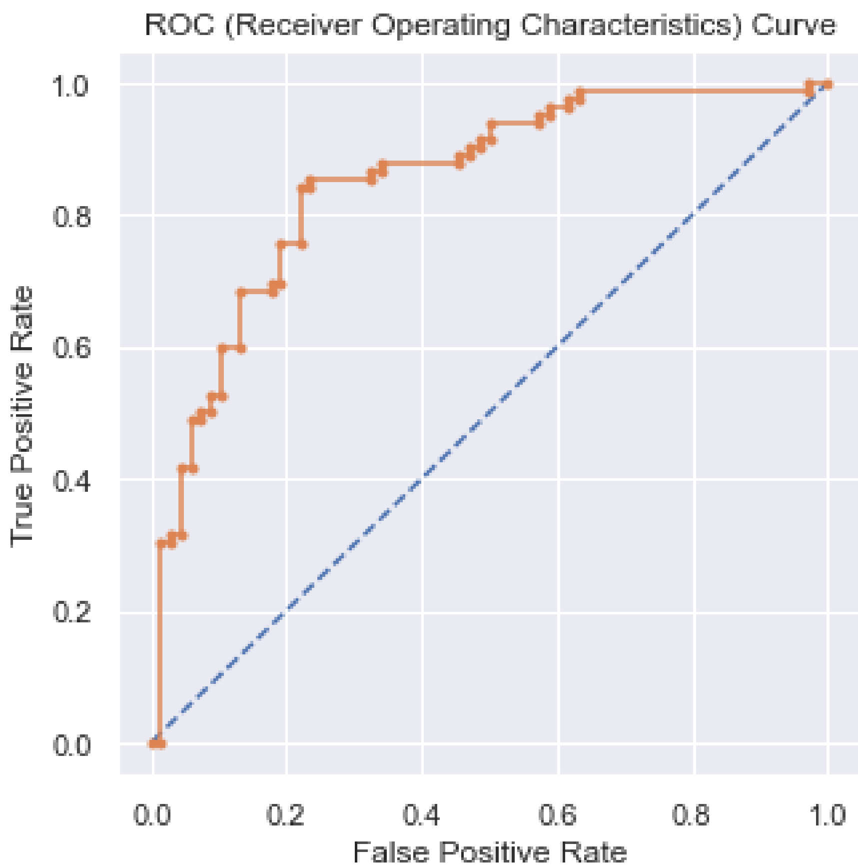
[147]: 0.7733333333333333

[148]:

```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)

probs = ada2.predict_proba(x_test)                  # predict probabilities
probs = probs[:, 1]                                 # keep probabilities for the␣
 ↪positive outcome only

auc_ada = roc_auc_score(y_test, probs)              # calculate AUC
print('AUC: %.3f' %auc_ada)
fpr, tpr, thresholds = roc_curve(y_test, probs)     # calculate roc curve
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: 0.850
```

[148]: Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')

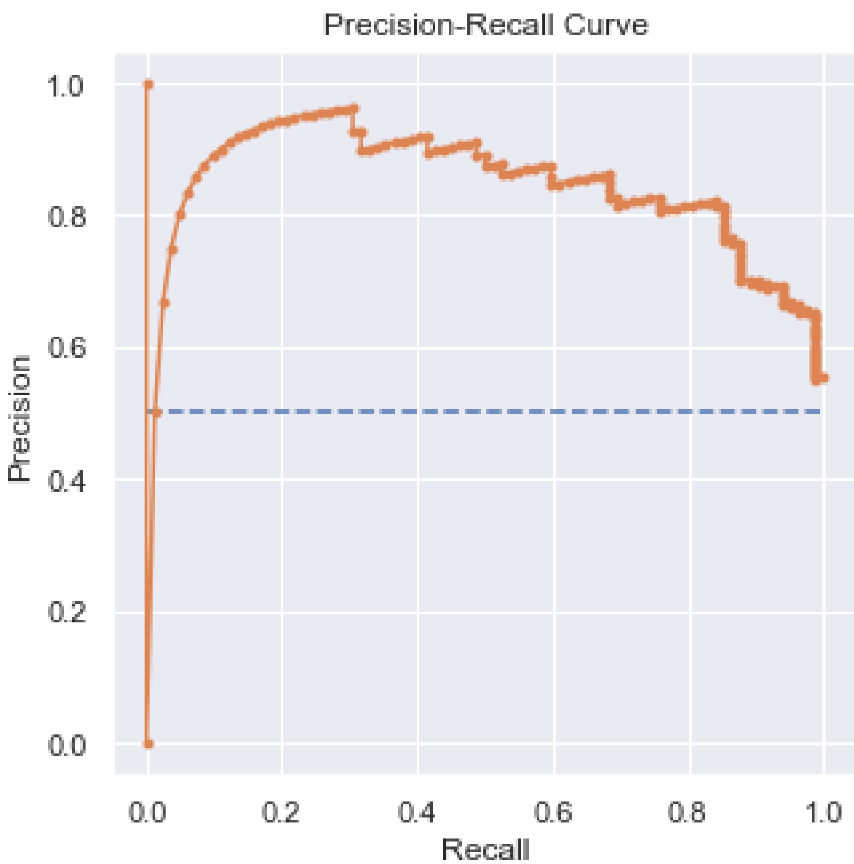


[149]:
```
# Precision Recall Curve

pred_y_test = ada2.predict(x_test)                                              # predict␣
 ↪class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                              #␣
 ↪calculate F1 score
auc_ada_pr = auc(recall, precision)                                             #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                                     #␣
 ↪calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_ada_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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision-Recall Curve")
```

```
f1=0.785 auc_pr=0.838 ap=0.845
```

[149]:
```
Text(0.5, 1.0, 'Precision-Recall Curve')
```



[150]:
```
models.append('ADA')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_ada)
```

8) Ensemble Learning --> Boosting --> Gradient Boosting (XGBClassifier):

[151]:
```
from xgboost import XGBClassifier
xgb1 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic',␣
↪nthread=4, seed=10)
```

[152]:
```
xgb1.fit(x_train, y_train)
```

[152]: XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=4, nthread=4, num_parallel_tree=1, predictor='auto',
              random_state=10, reg_alpha=0, …)

[153]:
```
xgb1.score(x_train, y_train)
```

[153]: 1.0

[154]:
```
xgb1.score(x_test, y_test)
```

[154]: 0.8266666666666667

Performance evaluation and optimizing parameters using GridSearchCV:

[155]:
```
parameters = {
    'max_depth': range (2, 10, 1),
    'n_estimators': range(60, 220, 40),
    'learning_rate': [0.1, 0.01, 0.05]
}
```

[156]:
```
gs_xgb = GridSearchCV(xgb1, param_grid = parameters, scoring = 'roc_auc',␣
 ↪n_jobs = 10, cv=5, verbose=0)
gs_xgb.fit(data_x_resampled, data_y_resampled)
```

[156]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     callbacks=None, colsample_bylevel=1,
                                     colsample_bynode=1, colsample_bytree=1,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None,
                                     gamma=0, gpu_id=-1,
                                     grow_policy='depthwise',
                                     importance_type=None,
                                     interaction_constraints='',
                                     learning_rate=0.300000012, max_bin=256,
                                     max_cat_to_onehot=4, max_delta_step=0,
                                     max_depth=6, max_leaves=0,
                                     min_child_weight=1, missing=nan,
                                     monotone_constraints='()',
                                     n_estimators=100, n_jobs=4, nthread=4,
```

```
                                         num_parallel_tree=1, predictor='auto',
                                         random_state=10, reg_alpha=0, …),
                     n_jobs=10,
                     param_grid={'learning_rate': [0.1, 0.01, 0.05],
                                 'max_depth': range(2, 10),
                                 'n_estimators': range(60, 220, 40)},
                     scoring='roc_auc')
```

[157]:
```
gs_xgb.best_params_
```

[157]:
```
{'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 180}
```

[158]:
```
gs_xgb.best_score_
```

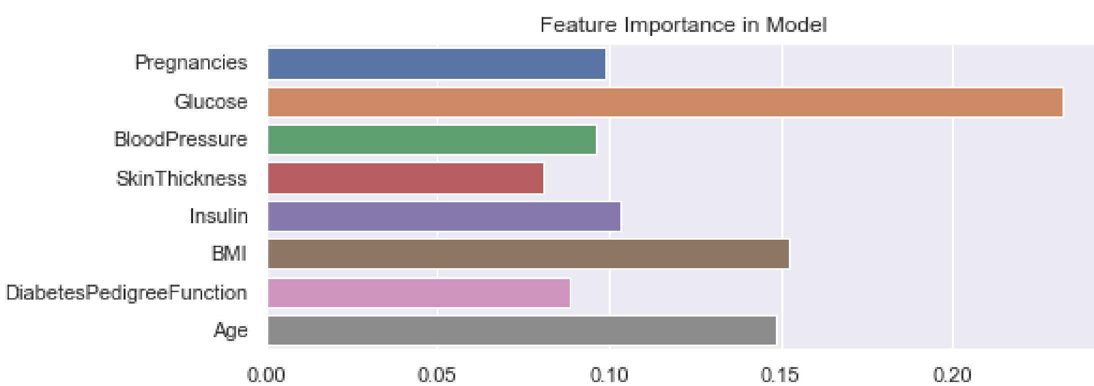
[158]:
```
0.88522
```

[159]:
```
xgb1.feature_importances_
```

[159]:
```
array([0.09883171, 0.23199296, 0.09590795, 0.08073226, 0.10332598,
       0.15247224, 0.08829137, 0.14844562], dtype=float32)
```

[160]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=x_train.columns, x=xgb1.feature_importances_)
plt.title("Feature Importance in Model")
```

[160]:
```
Text(0.5, 1.0, 'Feature Importance in Model')
```



[166]:
```
xgb2 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic',␣
↪nthread=4, seed=10, learning_rate= 0.05, max_depth= 7, n_estimators= 180)
```

[168]:
```
xgb2.fit(x_train, y_train)
```

[168]: XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.05, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=7, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=180,
              n_jobs=4, nthread=4, num_parallel_tree=1, predictor='auto',
              random_state=10, reg_alpha=0, …)

[169]:
```
xgb2.score(x_train, y_train)
```

[169]: 0.9976470588235294

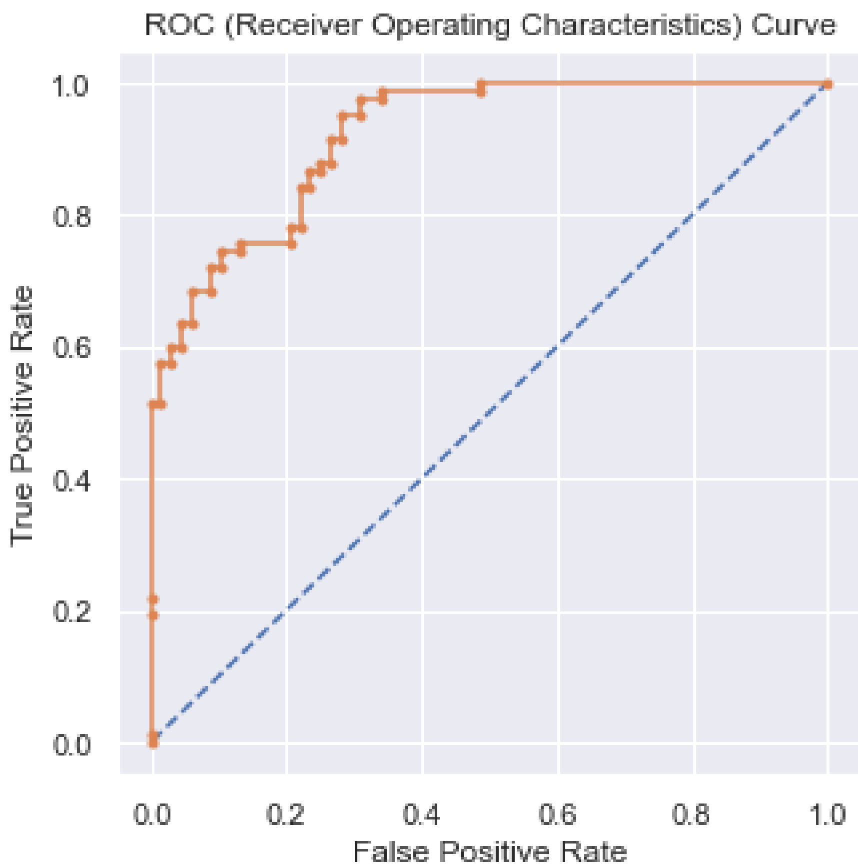
[170]:
```
xgb2.score(x_test, y_test)
```

[170]: 0.8066666666666666

[171]:
```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)

probs = xgb2.predict_proba(x_test)                 # predict probabilities
probs = probs[:, 1]                               # keep probabilities for the␣
 ↪positive outcome only

auc_xgb = roc_auc_score(y_test, probs)             # calculate AUC
print('AUC: %.3f' %auc_xgb)
fpr, tpr, thresholds = roc_curve(y_test, probs)   # calculate roc curve
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: 0.922

[171]: Text(0.5, 1.0, 'ROC (Receiver Operating Characteristics) Curve')

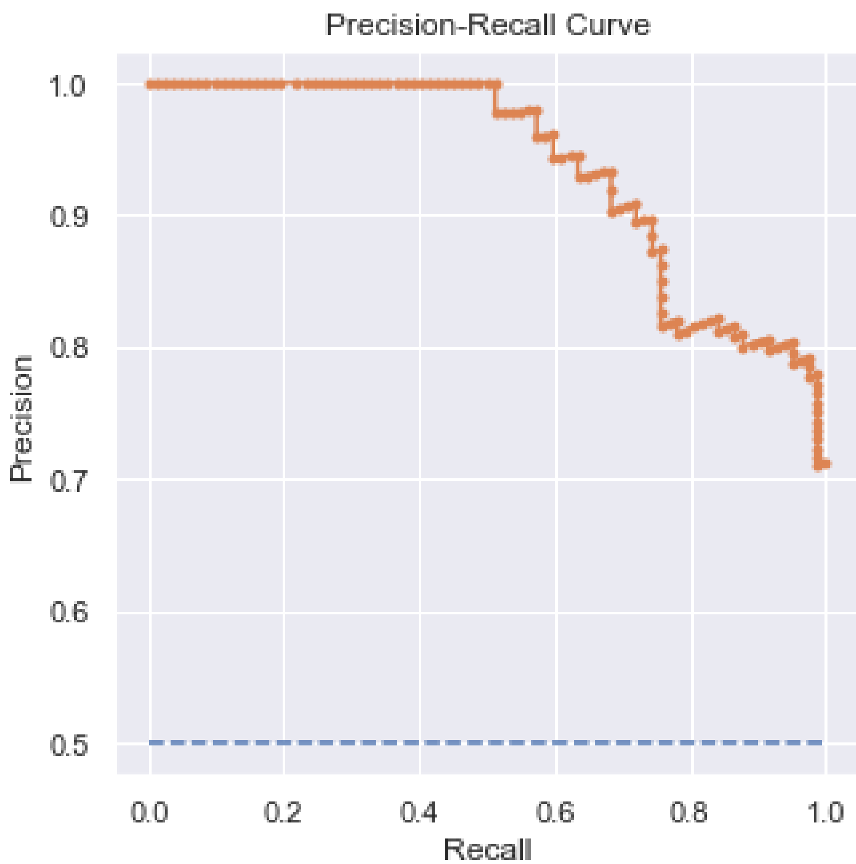


[172]:
```
# Precision Recall Curve

pred_y_test = xgb2.predict(x_test)                                         #␣
 ↪predict class values
precision, recall, thresholds = precision_recall_curve(y_test, probs) #␣
 ↪calculate precision-recall curve
f1 = f1_score(y_test, pred_y_test)                                    #␣
 ↪calculate F1 score
auc_xgb_pr = auc(recall, precision)                                    #␣
 ↪calculate precision-recall AUC
ap = average_precision_score(y_test, probs)                           #␣
 ↪calculate average precision score
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_xgb_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 model
plt.xlabel("Recall")
plt.ylabel("Precision")
```

```
plt.title("Precision-Recall Curve")
```

f1=0.824 auc_pr=0.936 ap=0.937

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

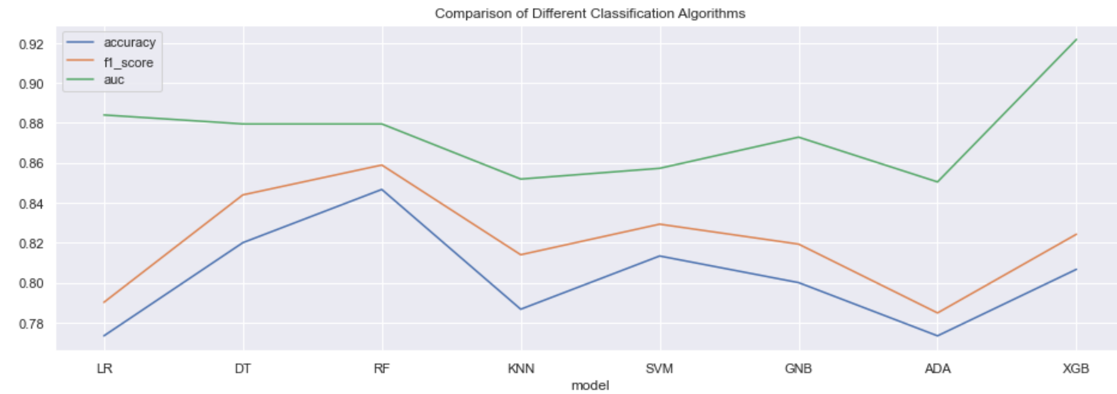


[173]:
```
models.append('XGB')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_xgb)
```

[174]:
```
model_summary = pd.DataFrame(zip(models,model_accuracy,model_f1,model_auc),
 ↪columns = ['model','accuracy','f1_score','auc'])
model_summary = model_summary.set_index('model')
```

[175]:
```
model_summary.plot(figsize=(16,5))
plt.title("Comparison of Different Classification Algorithms")
```

[175]: Text(0.5, 1.0, 'Comparison of Different Classification Algorithms')



[176]:
```
model_summary
```

[176]:

model	accuracy	f1_score	auc
LR	0.773333	0.790123	0.883967
DT	0.820000	0.843931	0.879484
RF	0.846667	0.858896	0.879484
KNN	0.786667	0.813953	0.851865
SVM	0.813333	0.829268	0.857245
GNB	0.800000	0.819277	0.872848
ADA	0.773333	0.784810	0.850430
XGB	0.806667	0.824242	0.921808

Among all models, RandomForest has given best accuracy and f1_score. Therefore we will build final model using RandomForest.

0.3 FINAL CLASSIFIER:

[177]:
```
final_model = rf2
```

0.4 Data Modeling:

(1) Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used:

[178]:
```
cr = classification_report(y_test, final_model.predict(x_test))
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.83      0.84      0.83        68
```

```
           1       0.86      0.85      0.86        82

    accuracy                           0.85       150
   macro avg       0.85      0.85      0.85       150
weighted avg       0.85      0.85      0.85       150
```

[179]:
```
confusion = confusion_matrix(y_test, final_model.predict(x_test))
print("Confusion Matrix:\n", confusion)
```

```
Confusion Matrix:
 [[57 11]
 [12 70]]
```

[182]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives

Accuracy = (TP+TN)/(TP+TN+FP+FN)
Precision = TP/(TP+FP)
Sensitivity = TP/(TP+FN)                    # also called recall
Specificity = TN/(TN+FP)
```

[183]:
```
print("Accuracy: %.3f"%Accuracy)
print("Precision: %.3f"%Precision)
print("Sensitivity: %.3f"%Sensitivity)
print("Specificity: %.3f"%Specificity)
print("AUC: %.3f"%auc_rf)
```

```
Accuracy: 0.847
Precision: 0.864
Sensitivity: 0.854
Specificity: 0.838
AUC: 0.924
```

Sensitivity and Specificity: By changing the threshold, target classification will be changed hence the sensitivity and specificity will also be changed. Which one of these two we should maximize? What should be ideal threshold?

Ideally we want to maximize both Sensitivity & Specificity. But this is not possible always. There is always a trade-off. Sometimes we want to be 100% sure on Predicted negatives, sometimes we want to be 100% sure on Predicted positives. Sometimes we simply don't want to compromise on sensitivity sometimes we don't want to compromise on specificity.

The threshold is set based on business problem. There are some cases where Sensitivity is important and need to be near to 1. There are business cases where Specificity is important and need to be near to 1. We need to understand the business problem and decide the importance of Sensitivity and Specificity.

0.5 Data Reporting:

2. 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 or non-diabetic population
   b. Scatter charts between relevant variables to analyze the relationships
   c. Histogram or frequency charts to analyze the distribution of the data
   d. Heatmap of correlation analysis among the relevant variables
   e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart. ### **PLEASE REFER TABLEAU FILE FOR DASHBOARD AND VISUALIZATION CREATED FOR DATA REPORTING.**

[ ]:

```

```