health care

October 3, 2024

[2]:
```
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)
```

[3]:
```
df = pd.read_csv('health care diabetes.csv')
df.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']
df[cols_with_null_as_zero] = df[cols_with_null_as_zero].replace(0, np.NaN)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   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
```

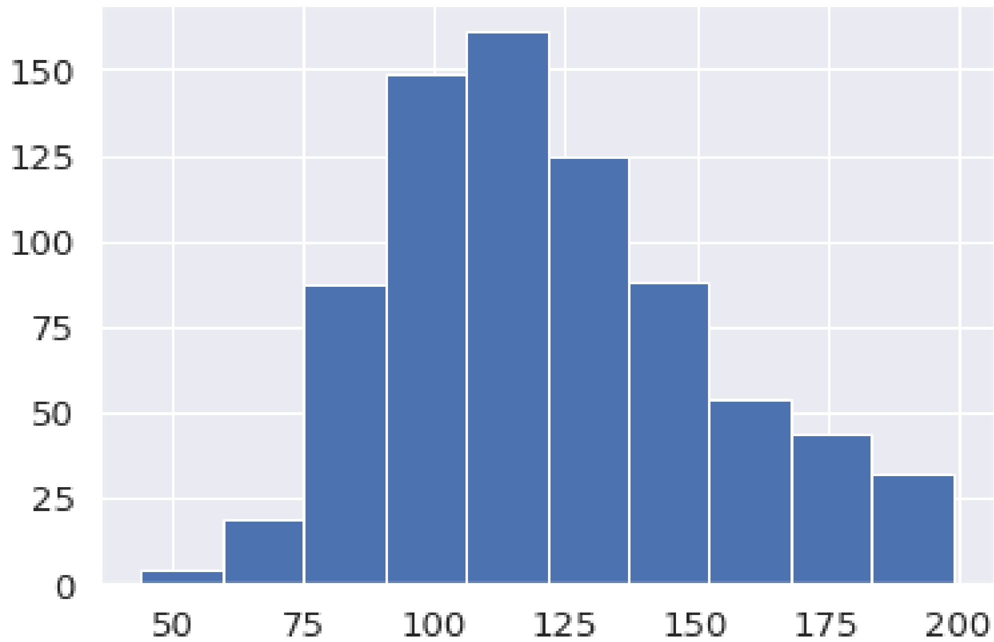
[6]:
```python
df.isnull().sum()
```

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

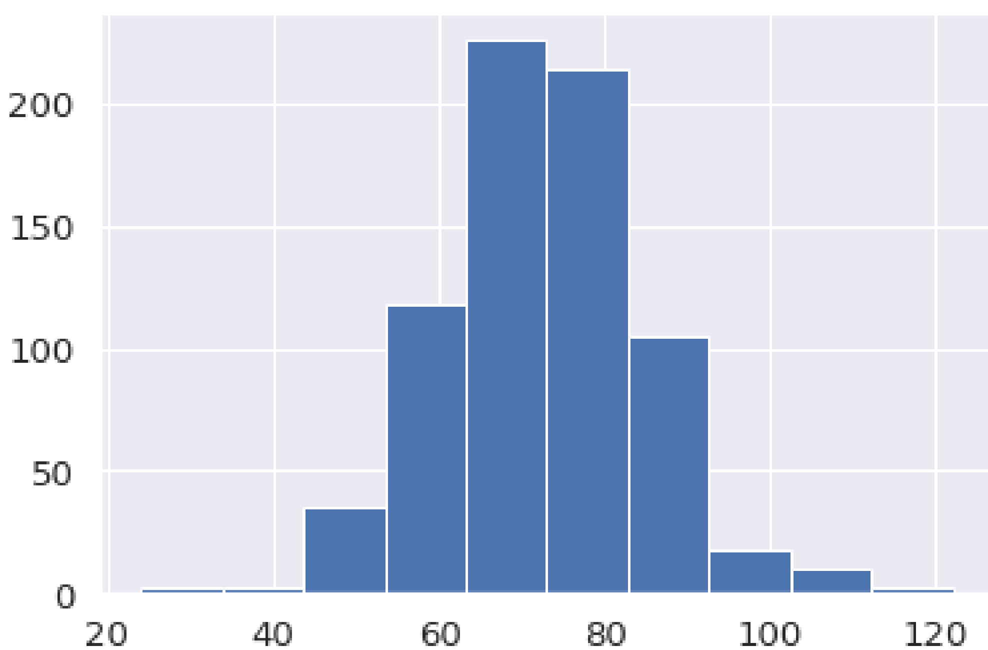
[7]:
```python
df.describe()
```

[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
```

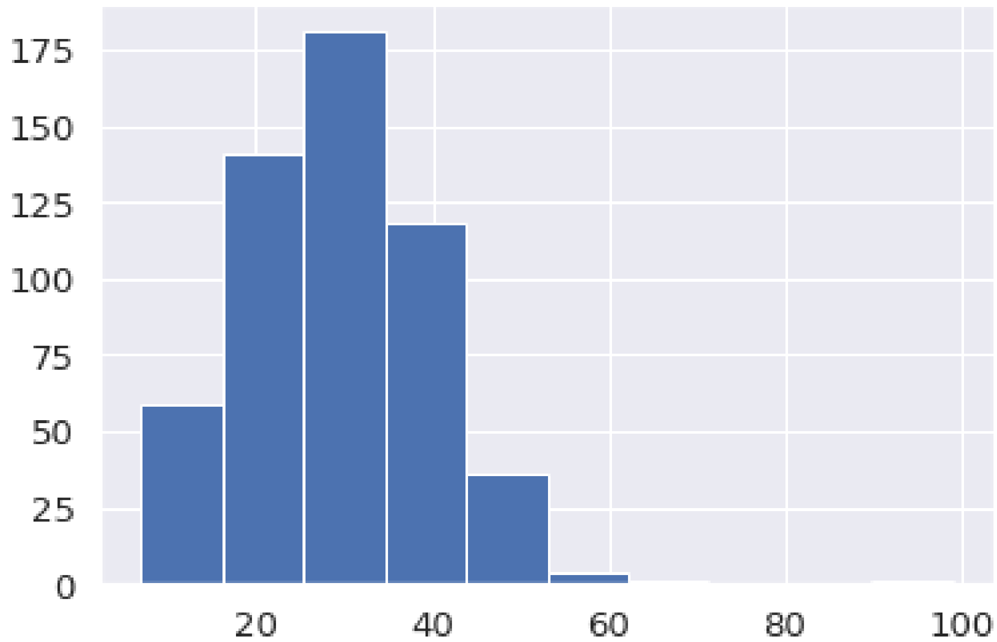
[8]:
```python
df['Glucose'].hist();
```



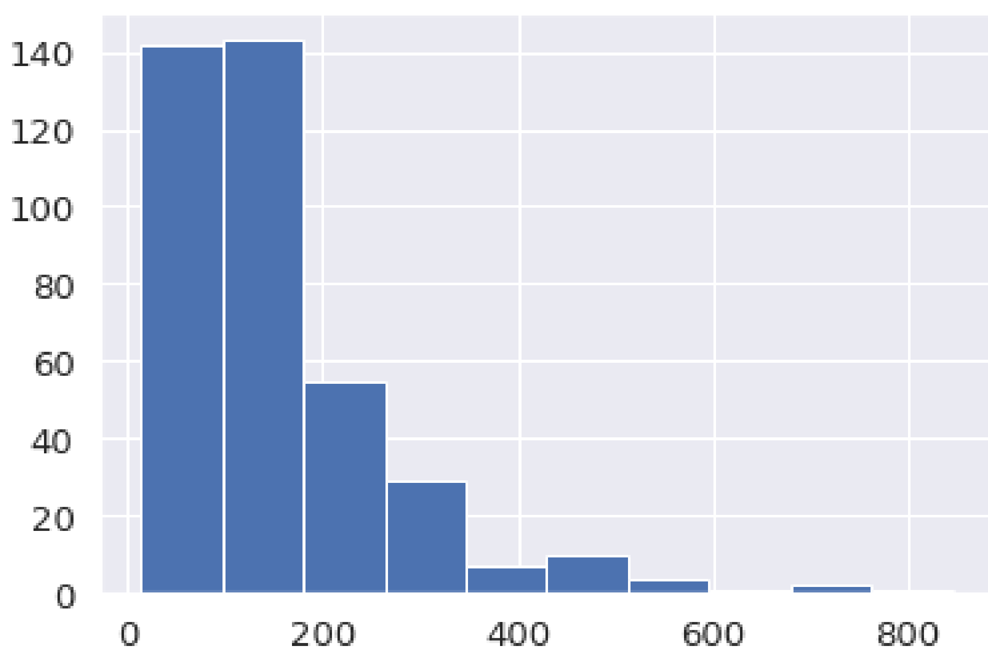
[9]:
```
df['BloodPressure'].hist();
```



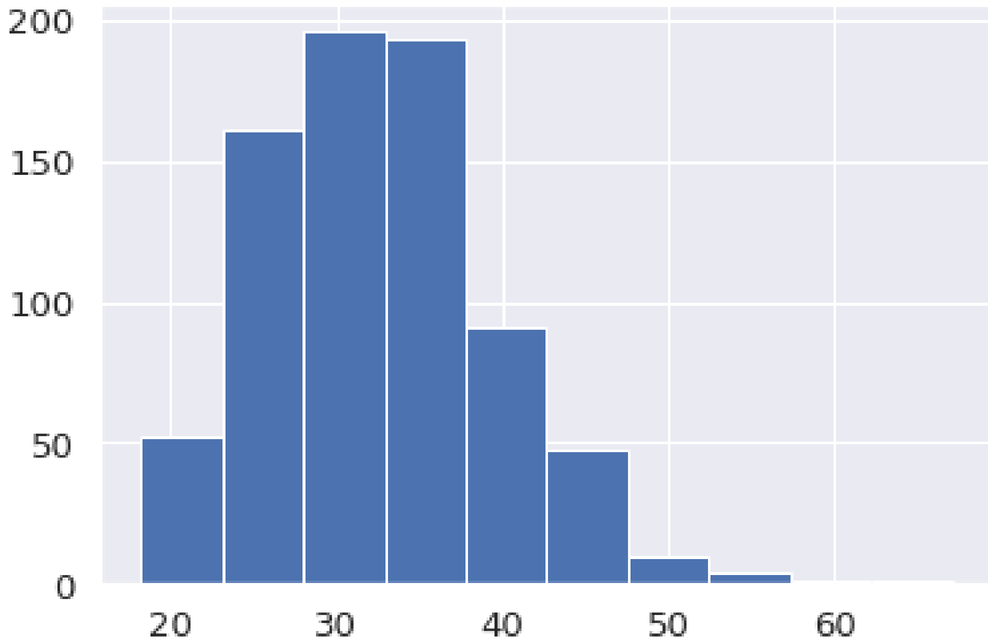
[10]:
```
df['SkinThickness'].hist();
```




```
[11]: df['Insulin'].hist();
```




```
[12]: df['BMI'].hist();
```

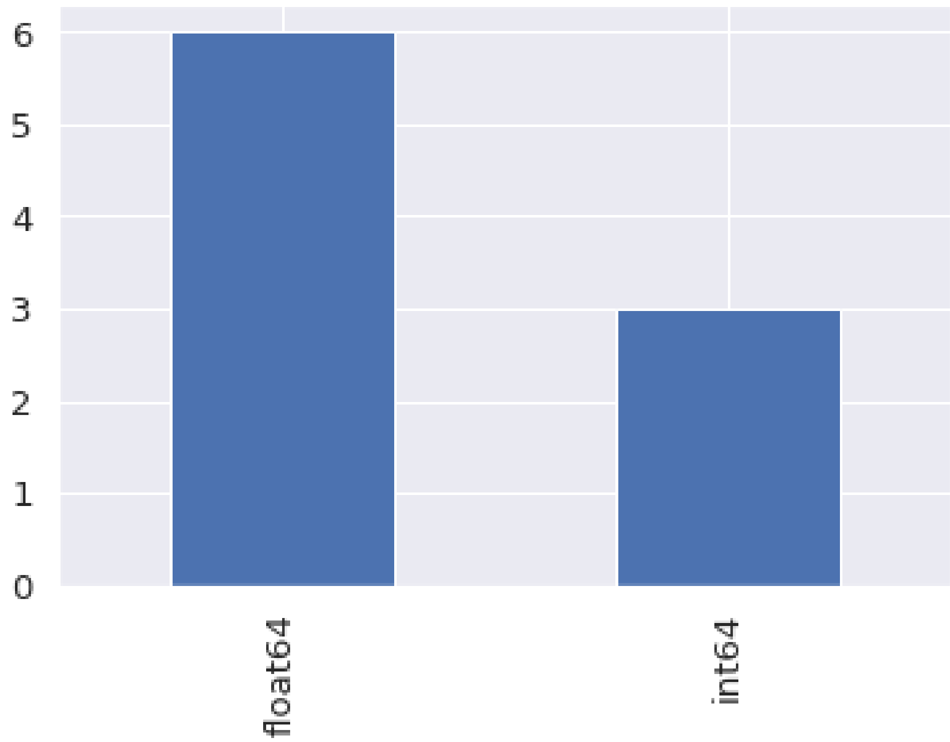


From above histograms, it is clear that Insulin has highly skewed data distribution and remaining 4 variables have relatively balanced data distribution therefore we will treat missing values

[13]:
```
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].median())
```

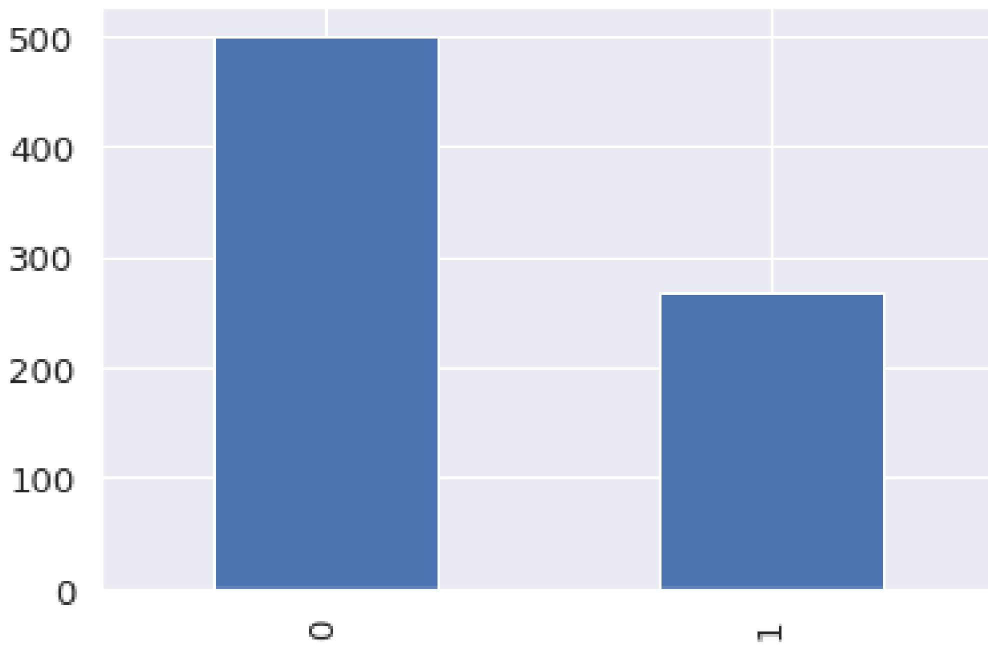
[14]:
```
cols_mean_for_null = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI']
df[cols_mean_for_null] = df[cols_mean_for_null].fillna(df[cols_mean_for_null].
 ↪mean())
```

[15]:
```
df.dtypes.value_counts().plot(kind='bar');
```



[16]:
```
df['Outcome'].value_counts().plot(kind='bar')
df['Outcome'].value_counts()
```

[16]:
```
0    500
1    268
Name: Outcome, dtype: int64
```



[17]:
```
df_X = df.drop('Outcome', axis=1)
df_y = df['Outcome']
print(df_X.shape, df_y.shape)
```

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

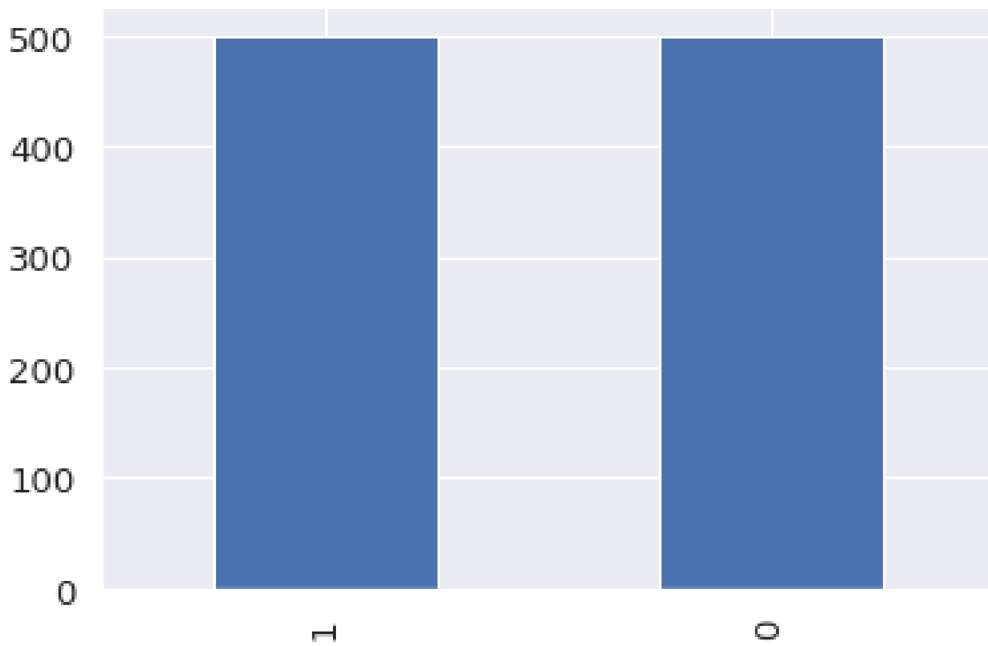
[18]:
```
from imblearn.over_sampling import SMOTE
```

[19]:
```
df_X_resampled, df_y_resampled = SMOTE(random_state=108).fit_resample(df_X,␣
↪df_y)
print(df_X_resampled.shape, df_y_resampled.shape)
```

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

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

[20]:
```
1    500
0    500
Name: Outcome, dtype: int64
```



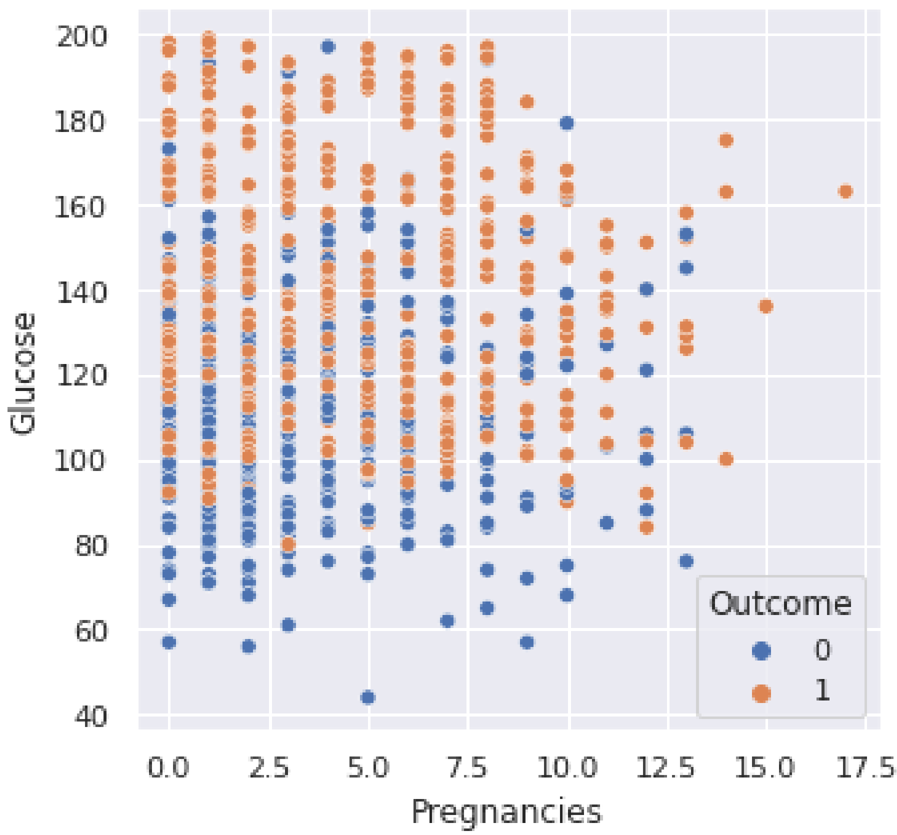
5. Create scatter charts between the pair of variables to understand the relationships. Describe your findings:

[21]:
```
df_resampled = pd.concat([df_X_resampled, df_y_resampled], axis=1)
df_resampled
```

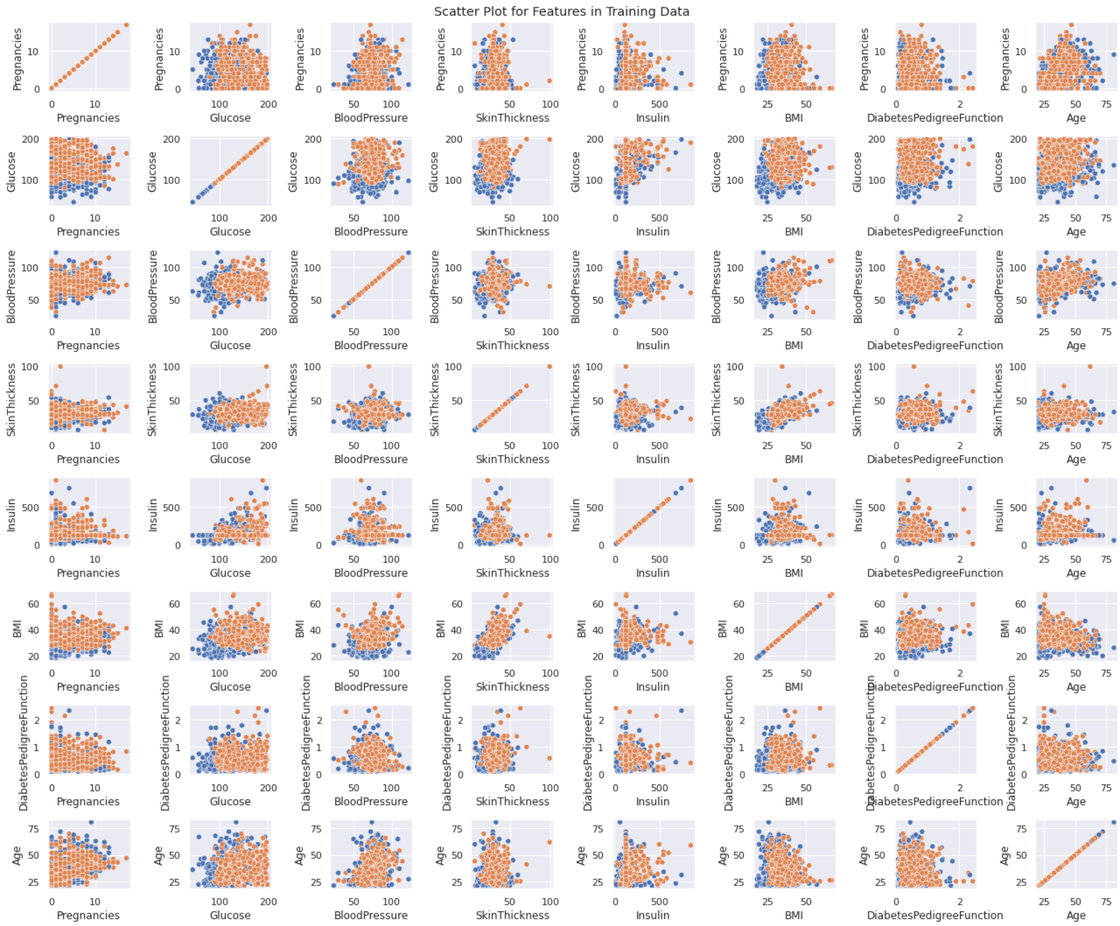
[21]:
```
     Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
0              6  148.000000      72.000000      35.000000  125.000000
1              1   85.000000      66.000000      29.000000  125.000000
2              8  183.000000      64.000000      29.153420  125.000000
3              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]
```

[22]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.scatterplot(x="Pregnancies", y="Glucose", data=df_resampled, hue="Outcome");
```



[23]:
```
fig, axes = plt.subplots(8, 8, figsize=(18, 15))
fig.suptitle('Scatter Plot for Features in Training Data')

for i, col_y in enumerate(df_X_resampled.columns):
    for j, col_x in enumerate(df_X_resampled.columns):
        sns.scatterplot(ax=axes[i, j], x=col_x, y=col_y, data=df_resampled,␣
 ↪hue="Outcome", legend = False)

plt.tight_layout()
```



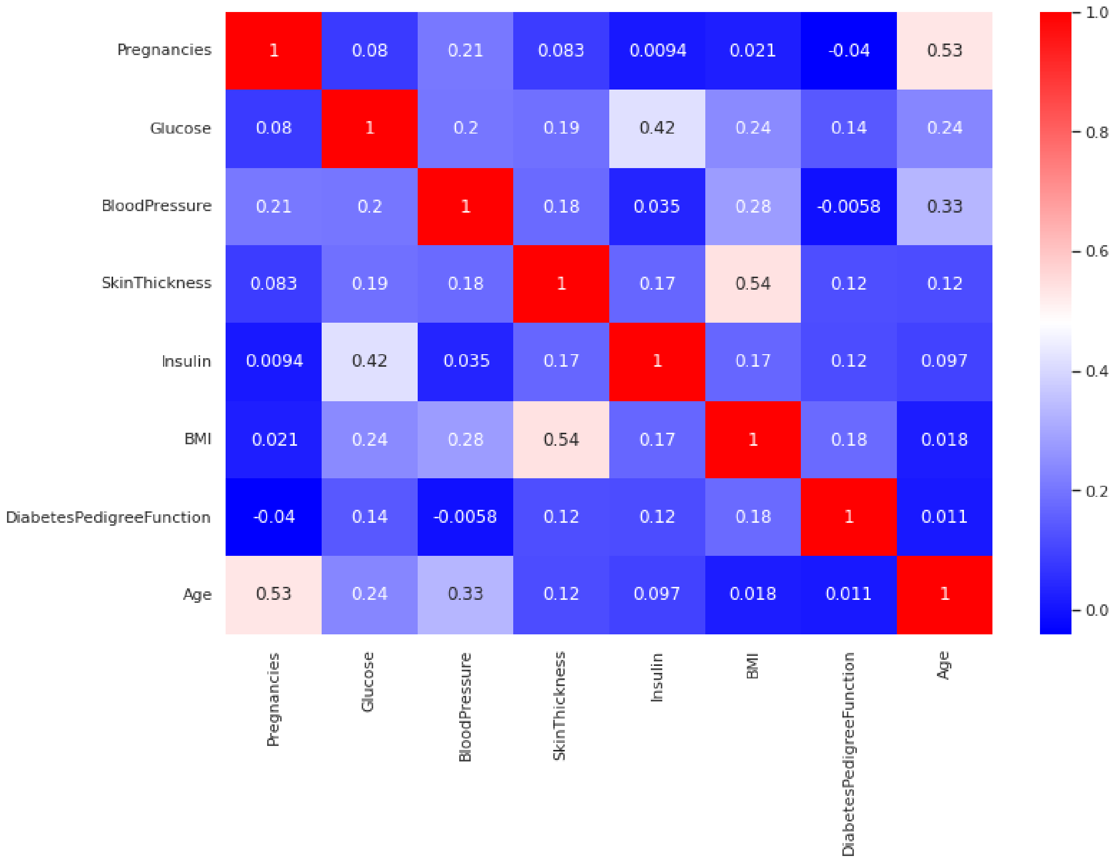
6. Perform correlation analysis. Visually explore it using a heat map:

[24]:
```
df_X_resampled.corr()
```

[24]:
```
                          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
```

[25]:
```python
plt.figure(figsize=(12,8))
sns.heatmap(df_X_resampled.corr(), cmap='bwr', annot=True);
```



Week 2: Data Modeling: (1) Devise strategies for model building. It is important to decide the right validation framework. Express your thought process:

[26]:
```
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,␣
↪precision_recall_curve
```

[27]:
```
X_train, X_test, y_train, y_test = train_test_split(df_X_resampled,␣
↪df_y_resampled, test_size=0.15, random_state =10)
```

[28]:
```
X_train.shape, X_test.shape
```

[28]:
```
((850, 8), (150, 8))
```

2. Apply an appropriate classification algorithm to build a model. 3. Compare various models with the results from KNN algorithm.

[29]:
```
models = []
model_accuracy = []
model_f1 = []
model_auc = []
```

[30]:
```
from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression(max_iter=300)
```

[31]:
```
lr1.fit(X_train,y_train)
```

[31]:
```
LogisticRegression(max_iter=300)
```

[32]:
```
lr1.score(X_train,y_train)
```

[32]:
```
0.7294117647058823
```

[33]:
```
lr1.score(X_test, y_test)
```

[33]:
```
0.76
```

[34]:
```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

[35]:
```
parameters = {'C':np.logspace(-5, 5, 50)}
```

[36]:
```
gs_lr = GridSearchCV(lr1, param_grid = parameters, cv=5, verbose=0)
gs_lr.fit(df_X_resampled, df_y_resampled)
```

[36]:
```
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])})
```

[37]:
```
gs_lr.best_params_
```

[37]:
```
{'C': 13.257113655901108}
```

[38]:
```
gs_lr.best_score_
```

[38]:
```
0.738
```

[39]:
```
lr2 = LogisticRegression(C=13.257113655901108, max_iter=300)
```

[40]:
```
lr2.fit(X_train,y_train)
```

[40]:
```
LogisticRegression(C=13.257113655901108, max_iter=300)
```

[41]:
```
lr2.score(X_train,y_train)
```

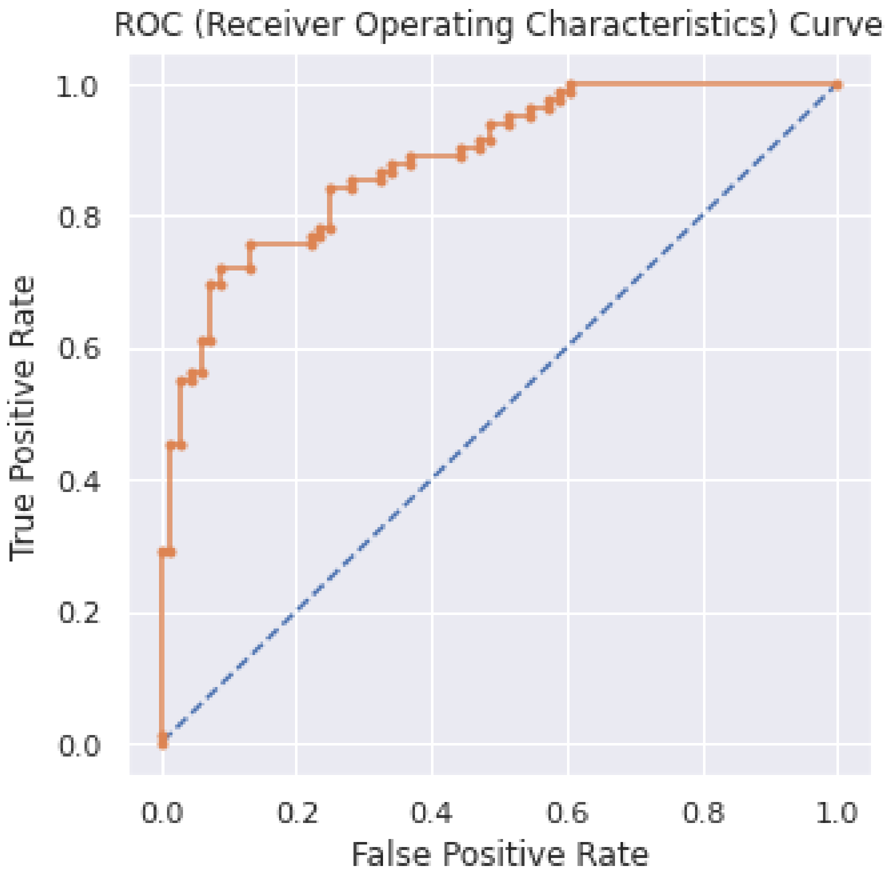
[41]:
```
0.7305882352941176
```

[42]:
```
lr2.score(X_test, y_test)
```

[42]:
```
0.7733333333333333
```

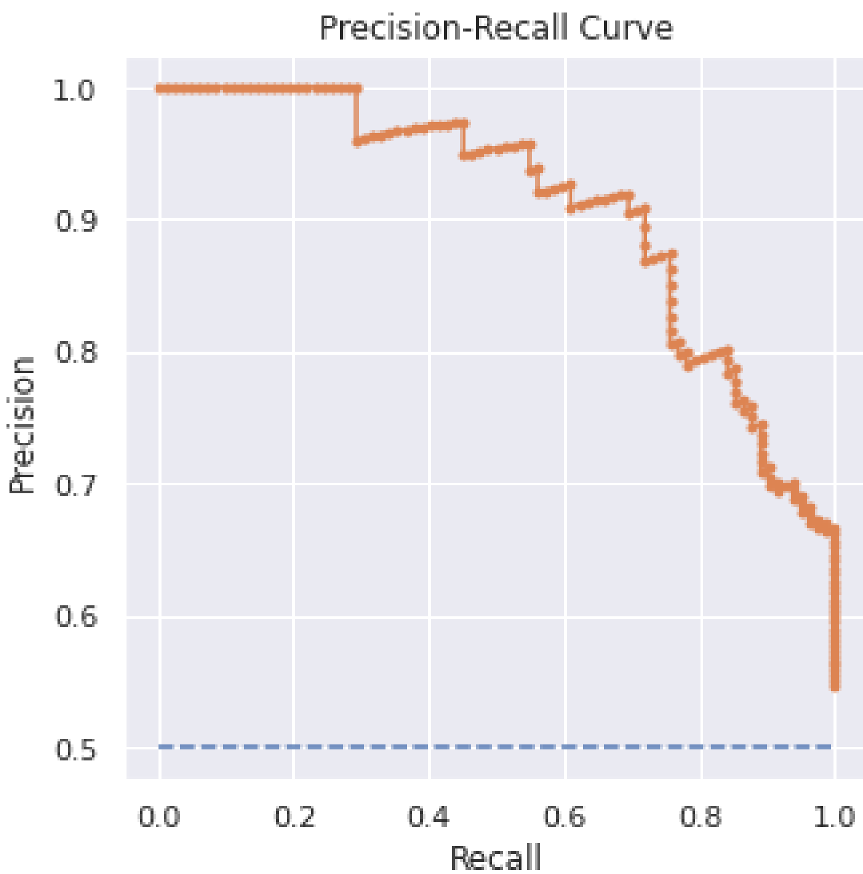
[43]:
```
probs = lr2.predict_proba(X_test)                    # predict probabilities
probs = probs[:, 1]                                  # keep probabilities for the␣
 ↪positive outcome only

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
```



[44]:
```
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
```



[45]:
```python
models.append('LR')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_lr)
```

[46]:
```python
from sklearn.tree import DecisionTreeClassifier
dt1 = DecisionTreeClassifier(random_state=0)
```

[47]:
```python
dt1.fit(X_train,y_train)
```

[47]:
```
DecisionTreeClassifier(random_state=0)
```

[48]:
```python
dt1.score(X_train,y_train)
```

[48]:
```
1.0
```

[49]:
```python
dt1.score(X_test, y_test)
```

[49]: 0.7733333333333333

[50]:
```
parameters = {
    'max_depth':[1,2,3,4,5,None]
}
```

[51]:
```
gs_dt = GridSearchCV(dt1, param_grid = parameters, cv=5, verbose=0)
gs_dt.fit(df_X_resampled, df_y_resampled)
```

[51]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [1, 2, 3, 4, 5, None]})
```

[52]:
```
gs_dt.best_params_
```

[52]: {'max_depth': 4}

[53]:
```
gs_dt.best_score_
```

[53]: 0.76

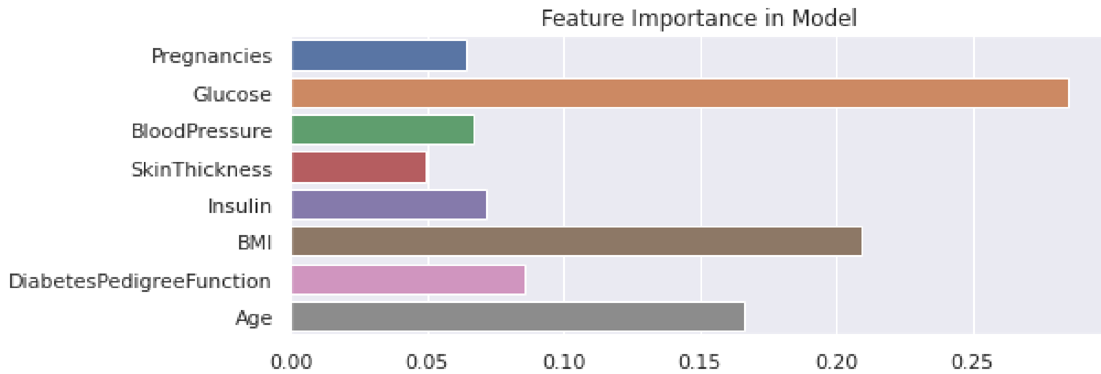
[54]:
```
dt1.feature_importances_
```

[54]:
```
array([0.06452226, 0.28556999, 0.06715314, 0.04979714, 0.07150365,
       0.20905992, 0.08573109, 0.16666279])
```

[55]:
```
X_train.columns
```

[55]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

[56]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8,3))
sns.barplot(y=X_train.columns, x=dt1.feature_importances_)
plt.title("Feature Importance in Model");
```



[57]:
```
dt2 = DecisionTreeClassifier(max_depth=4)
```

[58]:
```
dt2.fit(X_train,y_train)
```

[58]: DecisionTreeClassifier(max_depth=4)

[59]:
```
dt2.score(X_train,y_train)
```

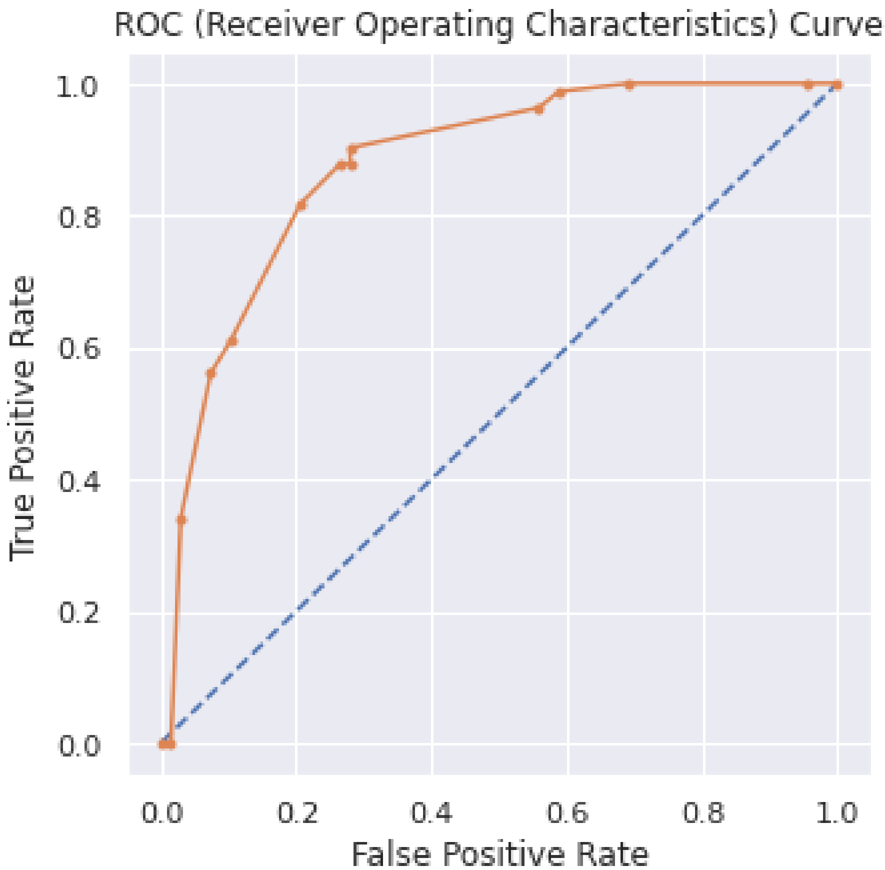
[59]: 0.8070588235294117

[60]:
```
dt2.score(X_test, y_test)
```

[60]: 0.8133333333333334

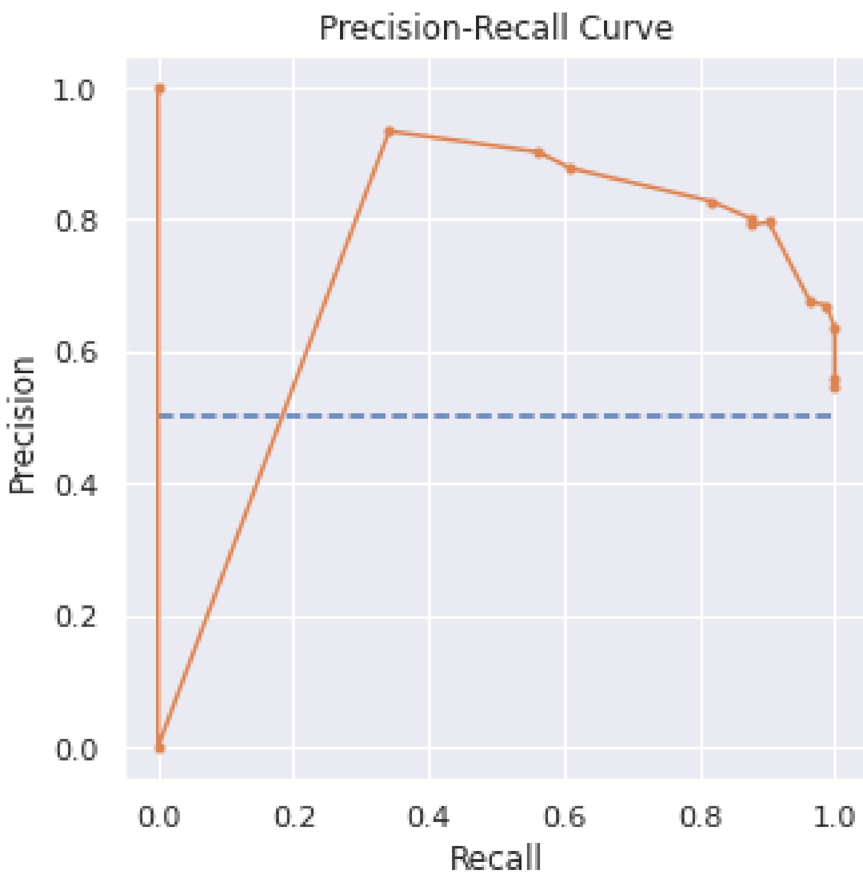
[61]:
```
probs = dt2.predict_proba(X_test)                # predict probabilities
probs = probs[:, 1]                              # keep probabilities for the␣
 ↪positive outcome only

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: 0.876



[62]:
```
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_dt_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=0.837 auc_pr=0.719 ap=0.864
```



[63]:
```
models.append('DT')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_dt)
```

[64]:
```
from sklearn.ensemble import RandomForestClassifier
rf1 = RandomForestClassifier()
```

[65]:
```
rf1 = RandomForestClassifier(random_state=0)
```

[66]:
```
rf1.fit(X_train, y_train)
```

[66]:
```
RandomForestClassifier(random_state=0)
```

[67]:
```
rf1.score(X_train, y_train)
```

[67]:
```
1.0
```

[68]:
```
rf1.score(X_test, y_test)
```

[68]:
```
0.8466666666666667
```

[69]:
```
parameters = {
    'n_estimators': [50,100,150],
    'max_depth': [None,1,3,5,7],
    'min_samples_leaf': [1,3,5]
}
```

[70]:
```
gs_dt = GridSearchCV(estimator=rf1, param_grid=parameters, cv=5, verbose=0)
gs_dt.fit(df_X_resampled, df_y_resampled)
```

[70]:
```
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, 100, 150]})
```

[71]:
```
gs_dt.best_params_
```

[71]:
```
{'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 100}
```

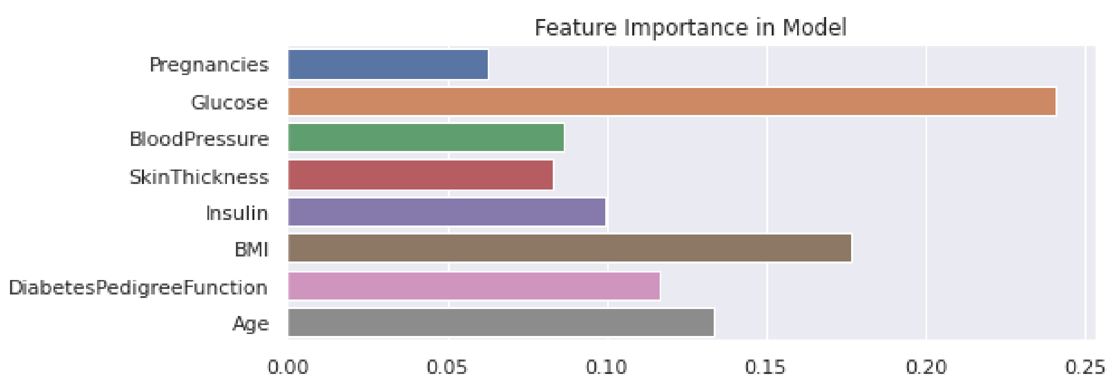
[72]:
```
gs_dt.best_score_
```

[72]:
```
0.813
```

[73]:
```
rf1.feature_importances_
```

[73]:
```
array([0.06264995, 0.24106573, 0.08653626, 0.08301549, 0.09945063,
       0.17678287, 0.11685244, 0.13364664])
```

[74]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=X_train.columns, x=rf1.feature_importances_);
plt.title("Feature Importance in Model");
```



[75]:
```
rf2 = RandomForestClassifier(max_depth=None, min_samples_leaf=1,␣
 ↪n_estimators=100)
```

[76]:
```
rf2.fit(X_train,y_train)
```

[76]: RandomForestClassifier()

[77]:
```
rf2.score(X_train,y_train)
```

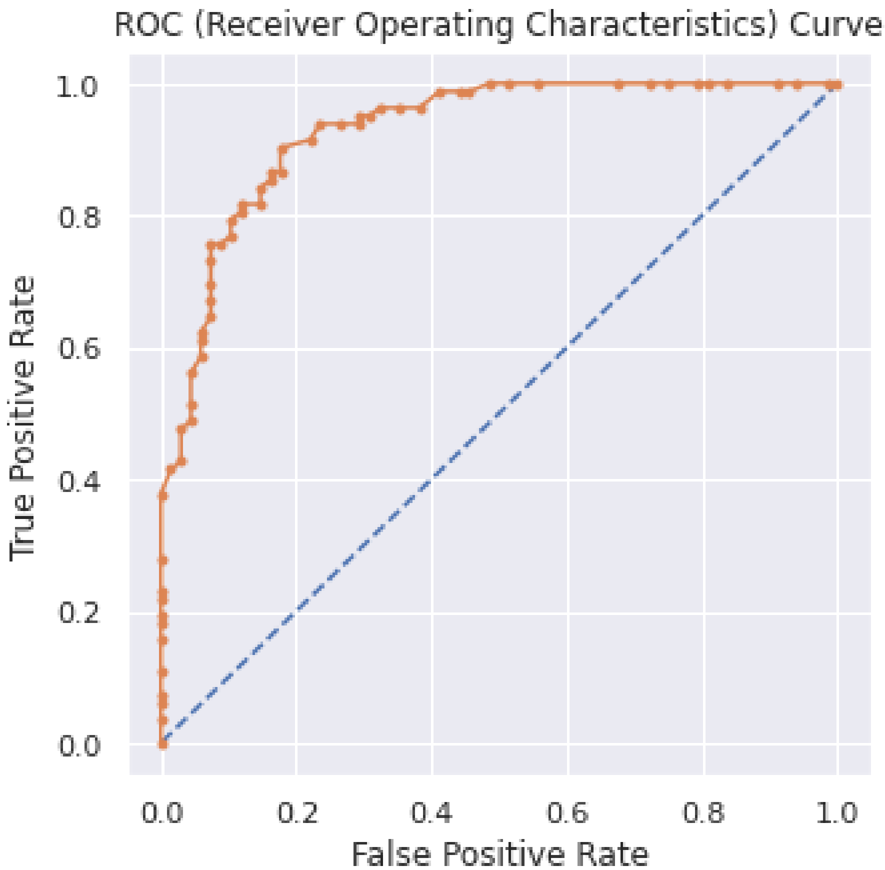
[77]: 1.0

[78]:
```
rf2.score(X_test, y_test)
```

[78]: 0.8466666666666667

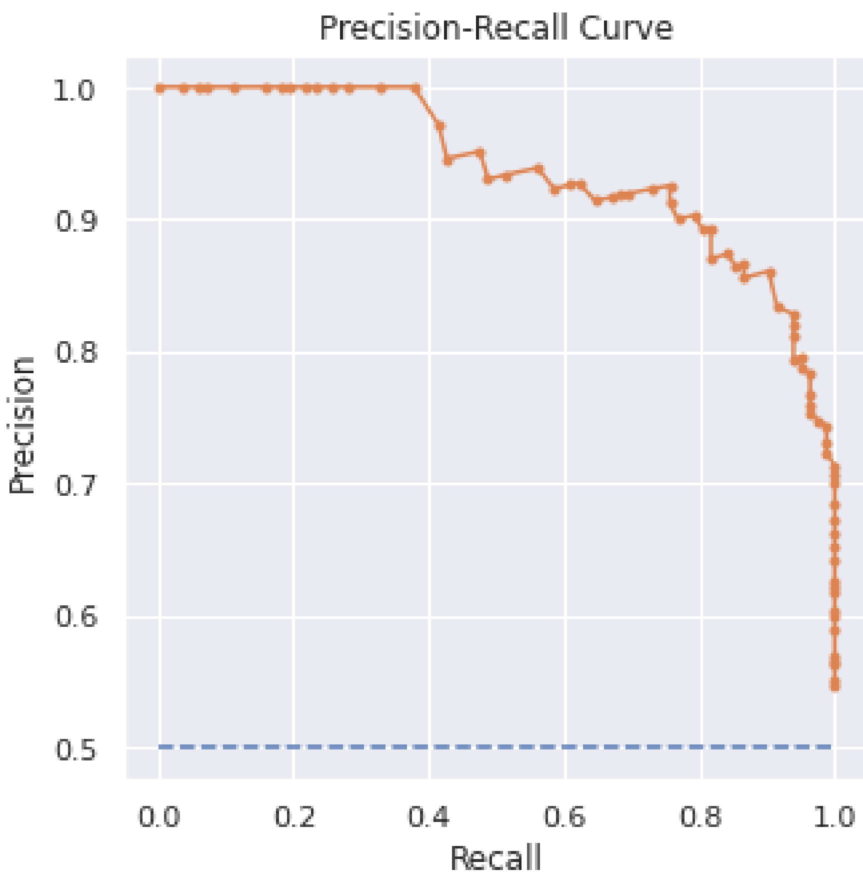
[79]:
```
probs = rf2.predict_proba(X_test)                # predict probabilities
probs = probs[:, 1]                              # keep probabilities for the␣
 ↪positive outcome only

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: 0.930



[80]:
```
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=0.861 auc_pr=0.937 ap=0.936
```



[81]:
```
models.append('RF')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_dt)
```

[82]:
```
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=3)
```

[83]:
```
knn1.fit(X_train, y_train)
```

[83]:
```
KNeighborsClassifier(n_neighbors=3)
```

[84]:
```
knn1.score(X_train,y_train)
```

[84]:
```
0.8835294117647059
```

[85]:
```
knn1.score(X_test,y_test)
```

[85]: 0.7866666666666666

[86]:
```
knn_neighbors = [i for i in range(2,16)]
parameters = {
    'n_neighbors': knn_neighbors
}
```

[87]:
```
gs_knn = GridSearchCV(estimator=knn1, param_grid=parameters, cv=5, verbose=0)
gs_knn.fit(df_X_resampled, df_y_resampled)
```

[87]:
```
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]})
```

[88]:
```
gs_knn.best_params_
```

[88]: {'n_neighbors': 3}

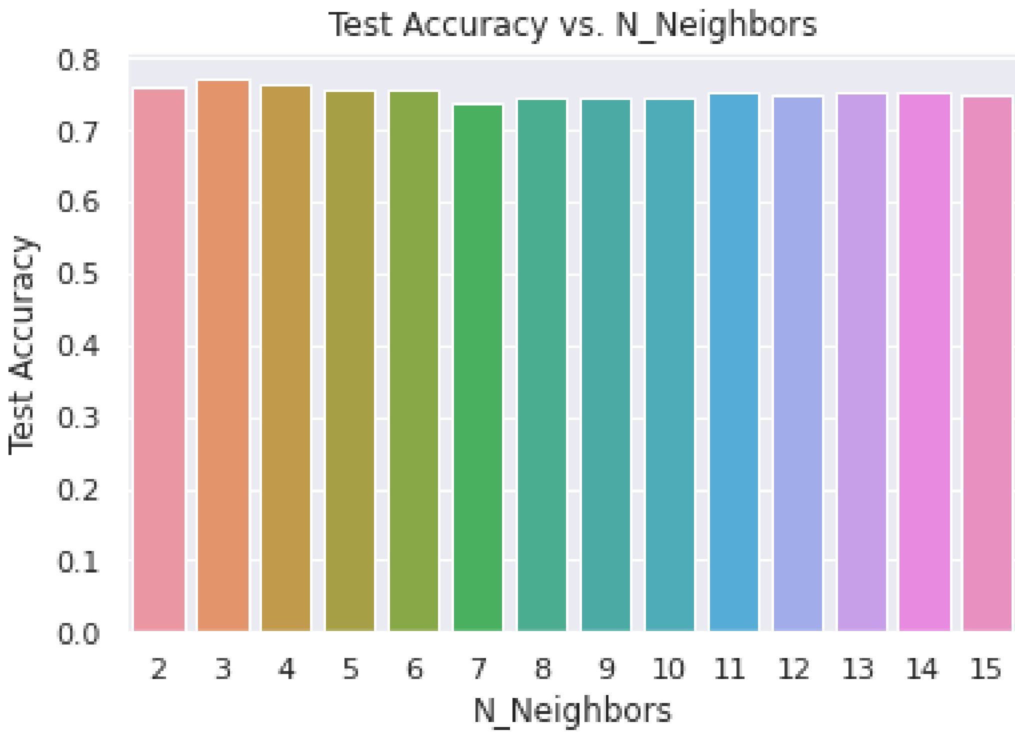
[89]:
```
gs_knn.best_score_
```

[89]: 0.771

[90]:
```
gs_knn.cv_results_['mean_test_score']
```

[90]:
```
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])
```

[91]:
```
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");
```



[92]:
```
knn2 = KNeighborsClassifier(n_neighbors=3)
```

[93]:
```
knn2.fit(X_train, y_train)
```

[93]: KNeighborsClassifier(n_neighbors=3)

[94]:
```
knn2.score(X_train,y_train)
```

[94]: 0.8835294117647059

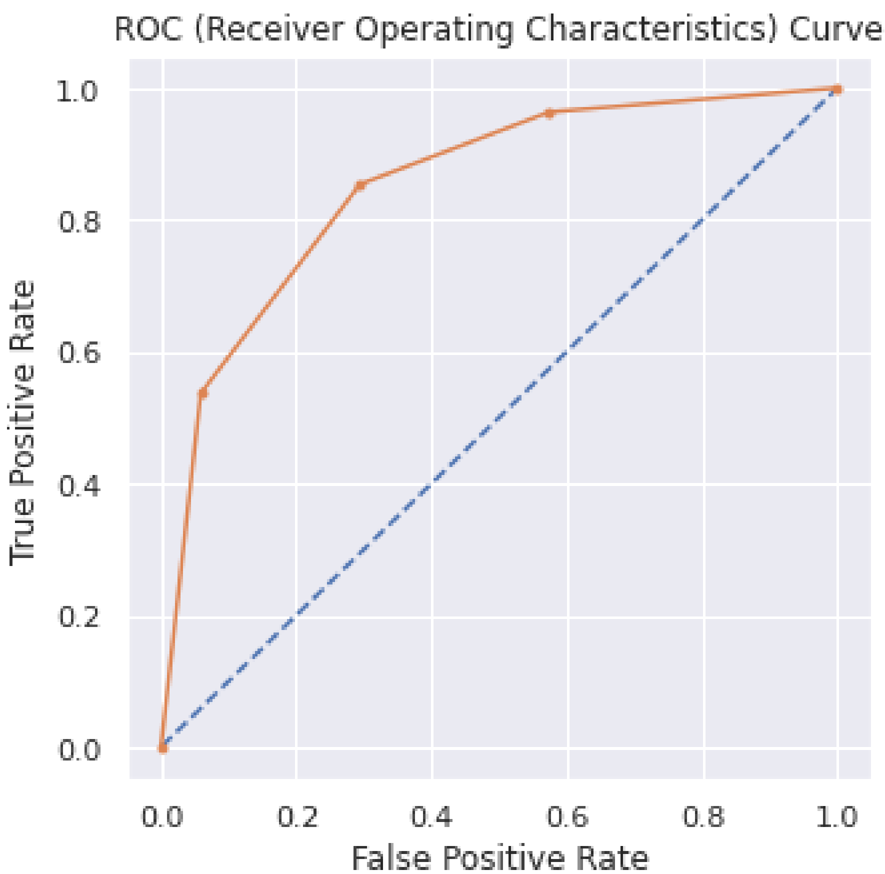
[95]:
```
knn2.score(X_test,y_test)
```

[95]: 0.7866666666666666

[96]:
```
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");
```

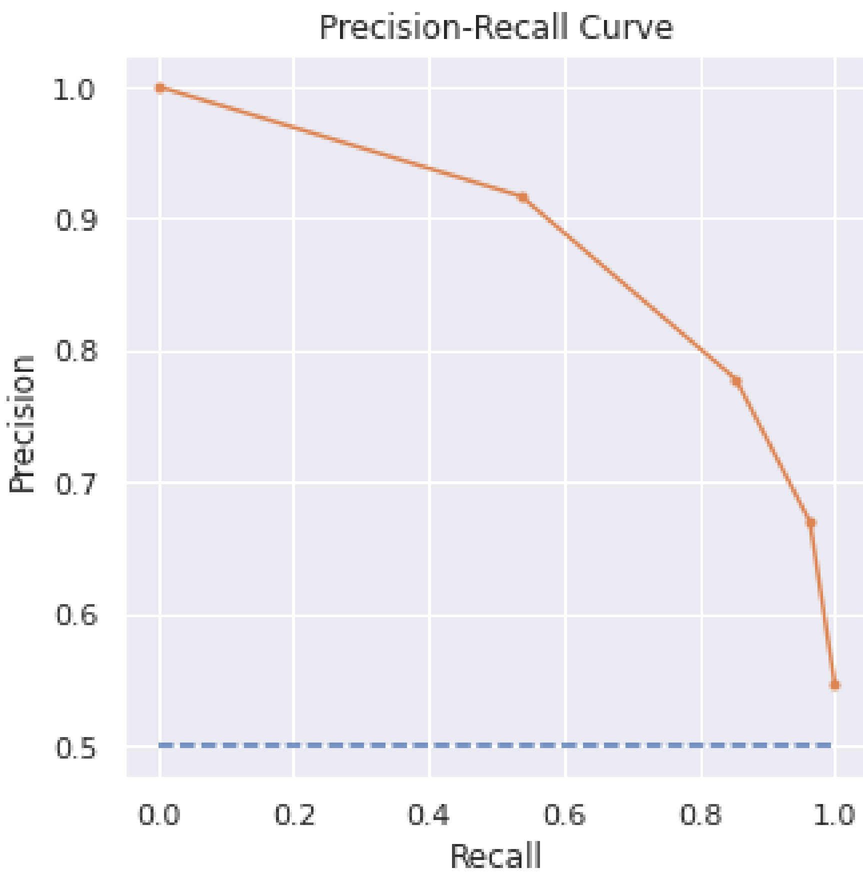
AUC: 0.852



[97]:
```
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
```



[98]:
```
models.append('KNN')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_knn)
```

[99]:
```
from sklearn.svm import SVC
svm1 = SVC(kernel='rbf')
```

[100]:
```
svm1.fit(X_train, y_train)
```

[100]: SVC()

[101]:
```
svm1.score(X_train, y_train)
```

[101]: 0.7282352941176471

[102]:
```
svm1.score(X_test, y_test)
```

[102]: 0.78

[103]:
```
parameters = {
    'C':[1, 5, 10, 15, 20, 25],
    'gamma':[0.001, 0.005, 0.0001, 0.00001]
}
```

[104]:
```
gs_svm = GridSearchCV(estimator=svm1, param_grid=parameters, cv=5, verbose=0)
gs_svm.fit(df_X_resampled, df_y_resampled)
```

[104]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [1, 5, 10, 15, 20, 25],
                         'gamma': [0.001, 0.005, 0.0001, 1e-05]})
```

[105]:
```
gs_svm.best_params_
```

[105]: {'C': 20, 'gamma': 0.005}

[106]:
```
gs_svm.best_score_
```

[106]: 0.8089999999999999

[107]:
```
svm2 = SVC(kernel='rbf', C=20, gamma=0.005, probability=True)
```

[108]:
```
svm2.fit(X_train, y_train)
```

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

[109]:
```
svm2.score(X_train, y_train)
```

[109]: 0.9941176470588236

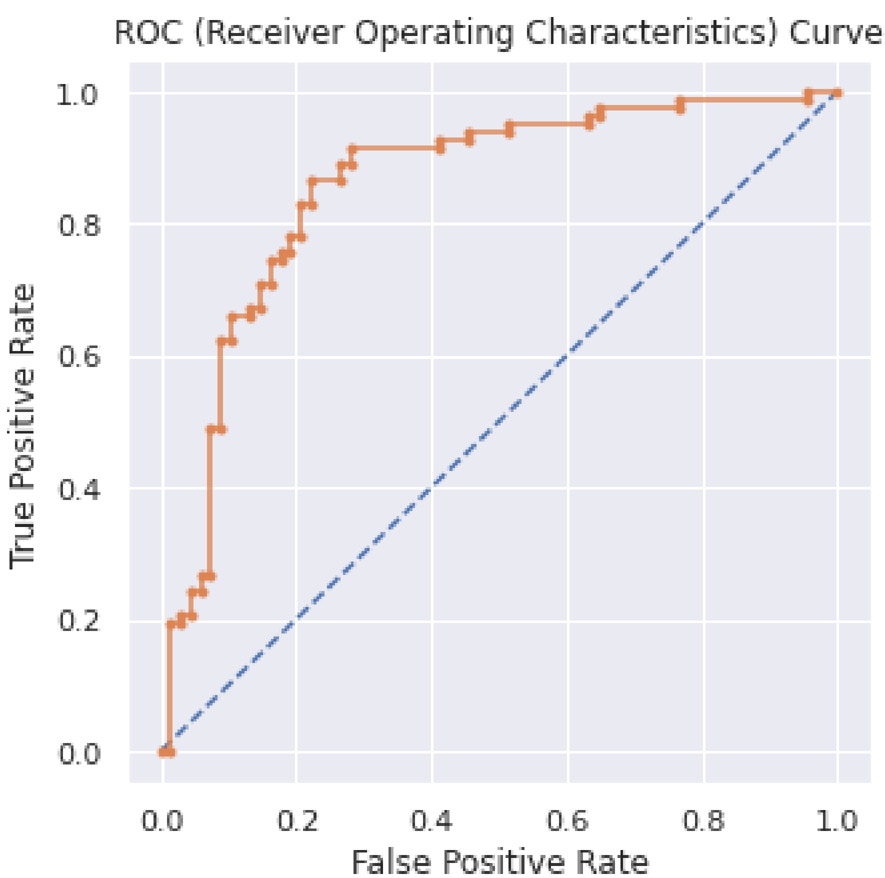
[110]:
```
svm2.score(X_test, y_test)
```

[110]: 0.8133333333333334

[111]:
```
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");
```

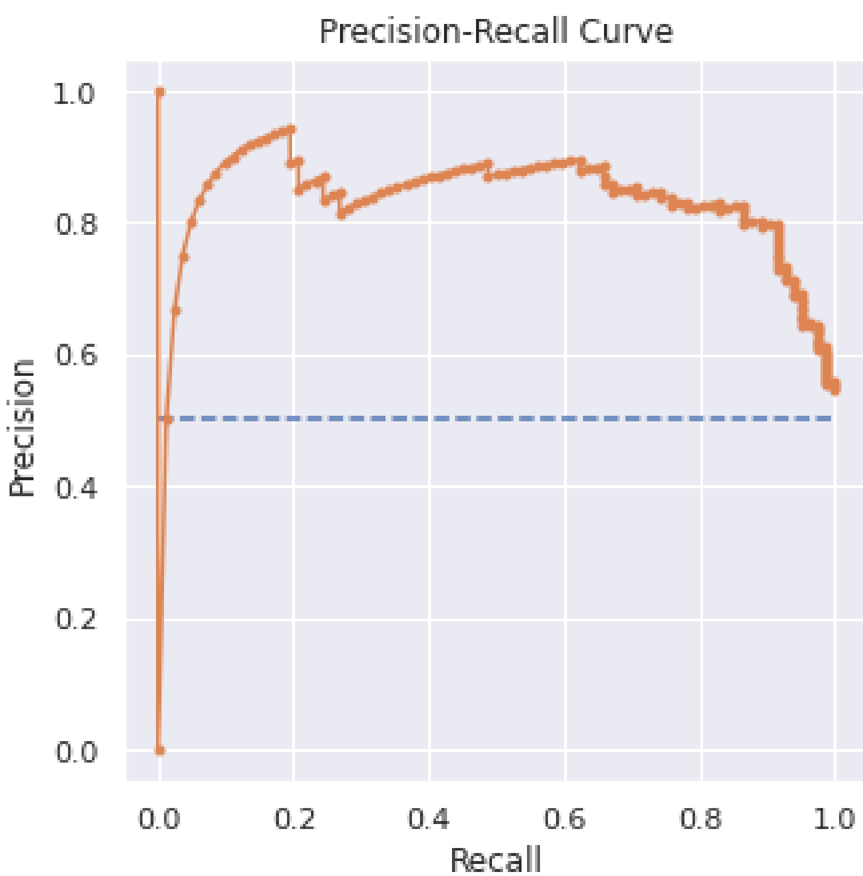
```
AUC: 0.857
```



[112]:
```
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=0.830 ap=0.837
```



[113]:
```
models.append('SVM')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_svm)
```

[114]:
```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
gnb = GaussianNB()
```

[115]:
```
gnb.fit(X_train, y_train)
```

[115]: GaussianNB()

[116]:
```
gnb.score(X_train, y_train)
```

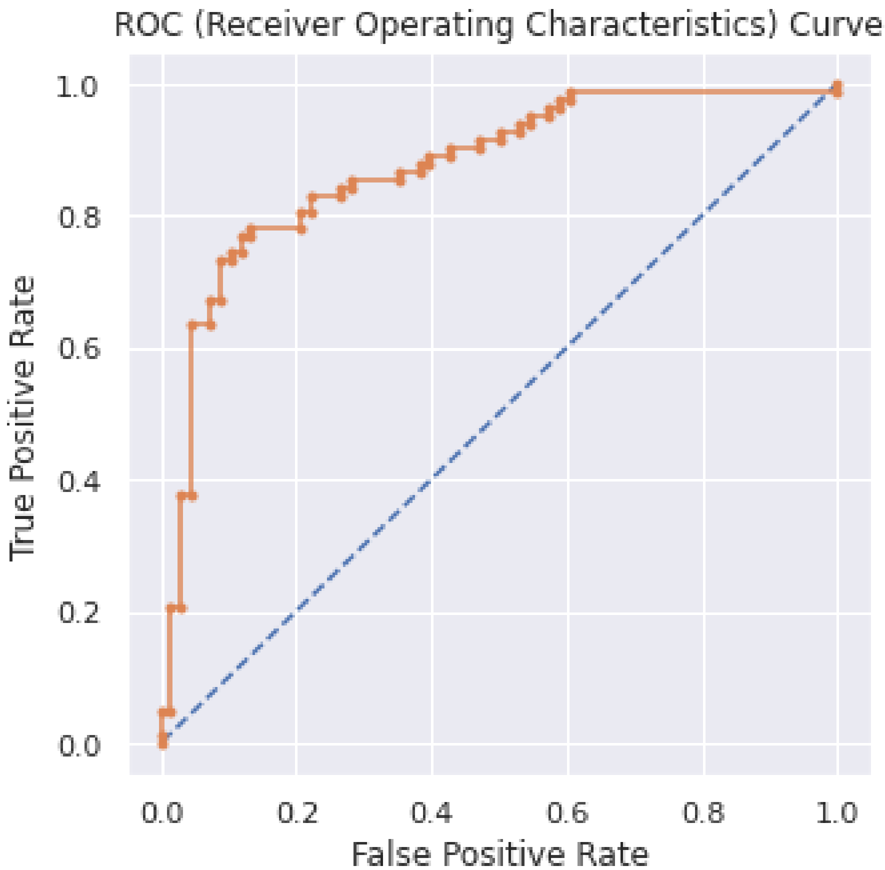
[116]: 0.7294117647058823

[117]:
```
gnb.score(X_test, y_test)
```

[117]: 0.8

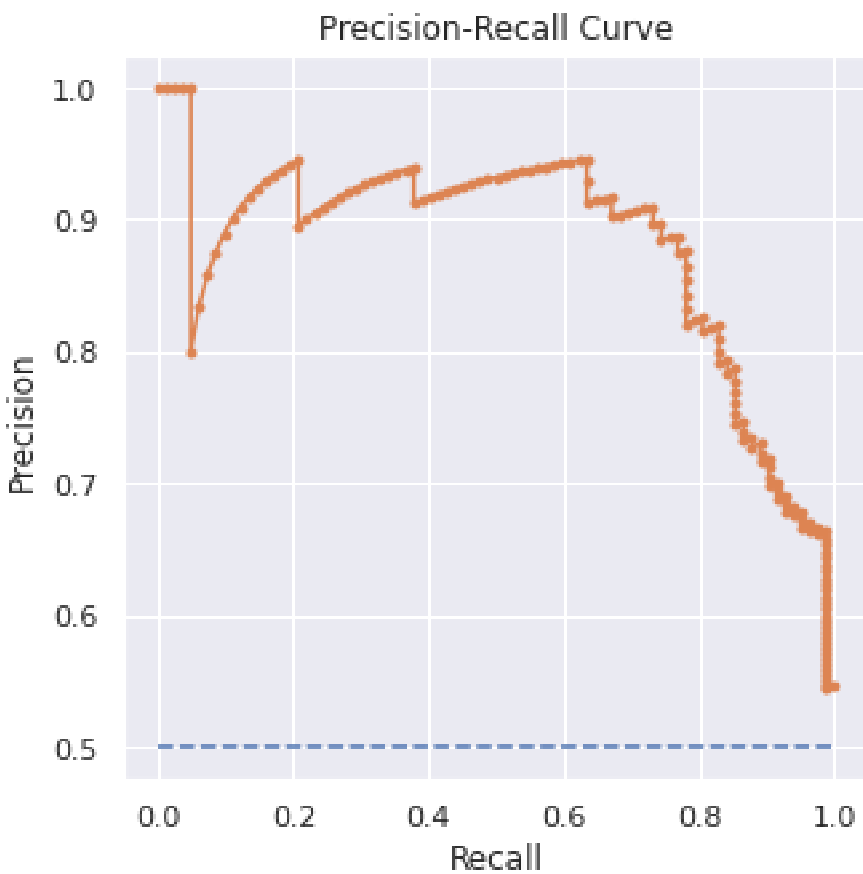
[118]:
```
probs = gnb.predict_proba(X_test)                # predict probabilities
probs = probs[:, 1]                              # keep probabilities for the␣
 ↪positive outcome only

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
```



[119]:
```
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
```



[120]:
```
models.append('GNB')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_gnb)
```

[121]:
```
from sklearn.ensemble import AdaBoostClassifier
ada1 = AdaBoostClassifier(n_estimators=100)
```

[122]:
```
ada1.fit(X_train,y_train)
```

[122]:
```
AdaBoostClassifier(n_estimators=100)
```

[123]:
```
ada1.score(X_train,y_train)
```

[123]:
```
0.8564705882352941
```

[124]:
```
ada1.score(X_test, y_test)
```

[124]: 0.7666666666666667

[125]:
```
parameters = {'n_estimators': [100,200,300,400,500,700,1000]}
```

[126]:
```
gs_ada = GridSearchCV(ada1, param_grid = parameters, cv=5, verbose=0)
gs_ada.fit(df_X_resampled, df_y_resampled)
```

[126]:
```
GridSearchCV(cv=5, estimator=AdaBoostClassifier(n_estimators=100),
             param_grid={'n_estimators': [100, 200, 300, 400, 500, 700, 1000]})
```

[127]:
```
gs_ada.best_params_
```

[127]: {'n_estimators': 500}

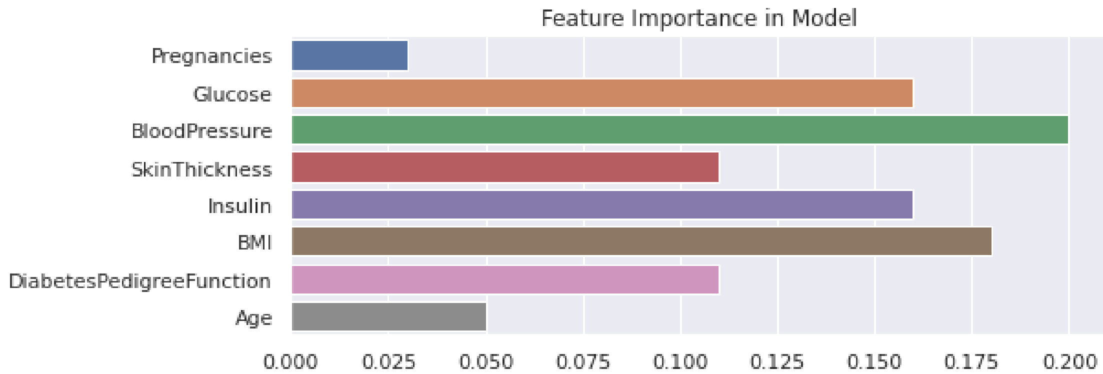
[128]:
```
gs_ada.best_score_
```

[128]: 0.785

[129]:
```
ada1.feature_importances_
```

[129]:
```
array([0.03, 0.16, 0.2 , 0.11, 0.16, 0.18, 0.11, 0.05])
```

[130]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=X_train.columns, x=ada1.feature_importances_)
plt.title("Feature Importance in Model");
```



[131]:
```
ada2 = AdaBoostClassifier(n_estimators=500)
```

[132]:
```
ada2.fit(X_train,y_train)
```

[132]: AdaBoostClassifier(n_estimators=500)

[133]:
```
ada2.score(X_train,y_train)
```

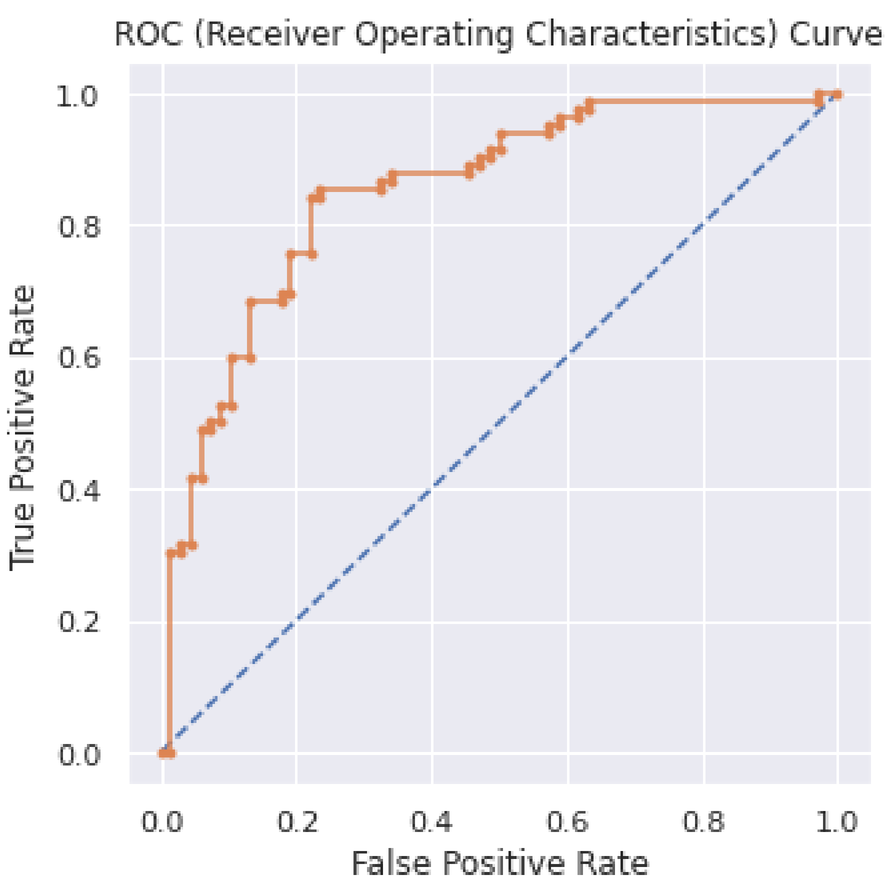
[133]: 0.9247058823529412

[134]:
```python
ada2.score(X_test, y_test)
```

[134]: 0.7733333333333333

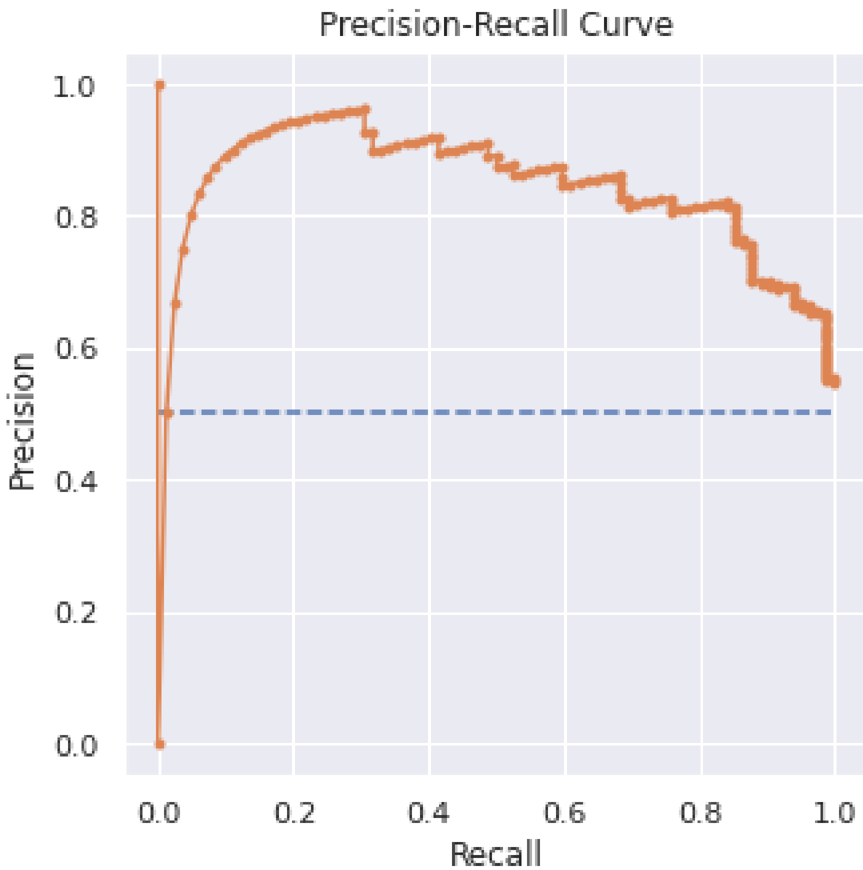
[135]:
```python
probs = ada2.predict_proba(X_test)                  # predict probabilities
probs = probs[:, 1]                                 # keep probabilities for the␣
 ↪positive outcome only

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



[136]:
```python
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
```



[137]:
```
models.append('ADA')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_ada)
```

[138]:
```
from xgboost import XGBClassifier
xgb1 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic',␣
↪nthread=4, seed=10)
```

[139]:
```
xgb1.fit(X_train, y_train)
```

[139]:
```
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, …)
```

[140]:
```
xgb1.score(X_train, y_train)
```

[140]:
```
1.0
```

[141]:
```
xgb1.score(X_test, y_test)
```

[141]:
```
0.8266666666666667
```

[142]:
```
parameters = {
    'max_depth': range (2, 10, 1),
    'n_estimators': range(60, 220, 40),
    'learning_rate': [0.1, 0.01, 0.05]
}
```

[143]:
```
gs_xgb = GridSearchCV(xgb1, param_grid = parameters, scoring = 'roc_auc',␣
 ↪n_jobs = 10, cv=5, verbose=0)
gs_xgb.fit(df_X_resampled, df_y_resampled)
```

[143]:
```
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')
```

[144]:
```
gs_xgb.best_params_
```

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

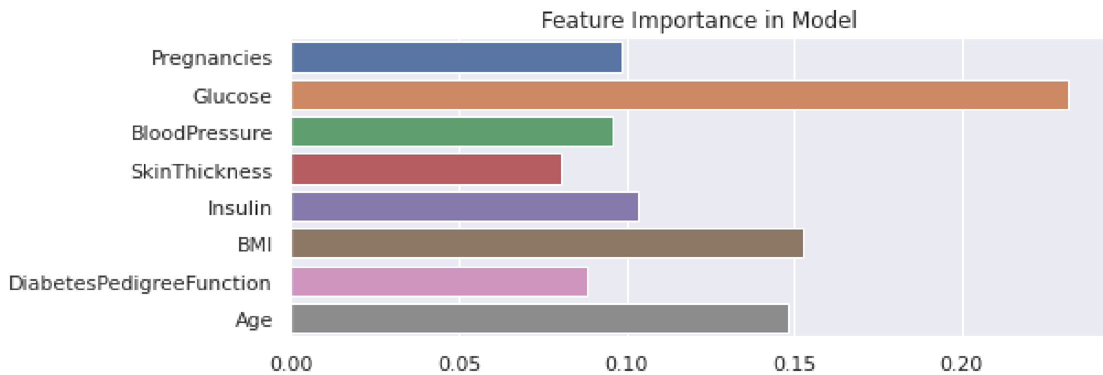
[145]:
```
gs_xgb.best_score_
```

[145]: 0.88522

[146]:
```
xgb1.feature_importances_
```

[146]:
```
array([0.09883171, 0.23199296, 0.09590795, 0.08073226, 0.10332598,
       0.15247224, 0.08829137, 0.14844562], dtype=float32)
```

[147]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=X_train.columns, x=xgb1.feature_importances_)
plt.title("Feature Importance in Model");
```



[148]:
```
xgb2 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic',
                     nthread=4, seed=10, learning_rate= 0.05, max_depth= 7,␣
 ↪n_estimators= 180)
```

[149]:
```
xgb2.fit(X_train,y_train)
```

[149]:
```
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, …)
```

[150]:
```
xgb2.score(X_train,y_train)
```

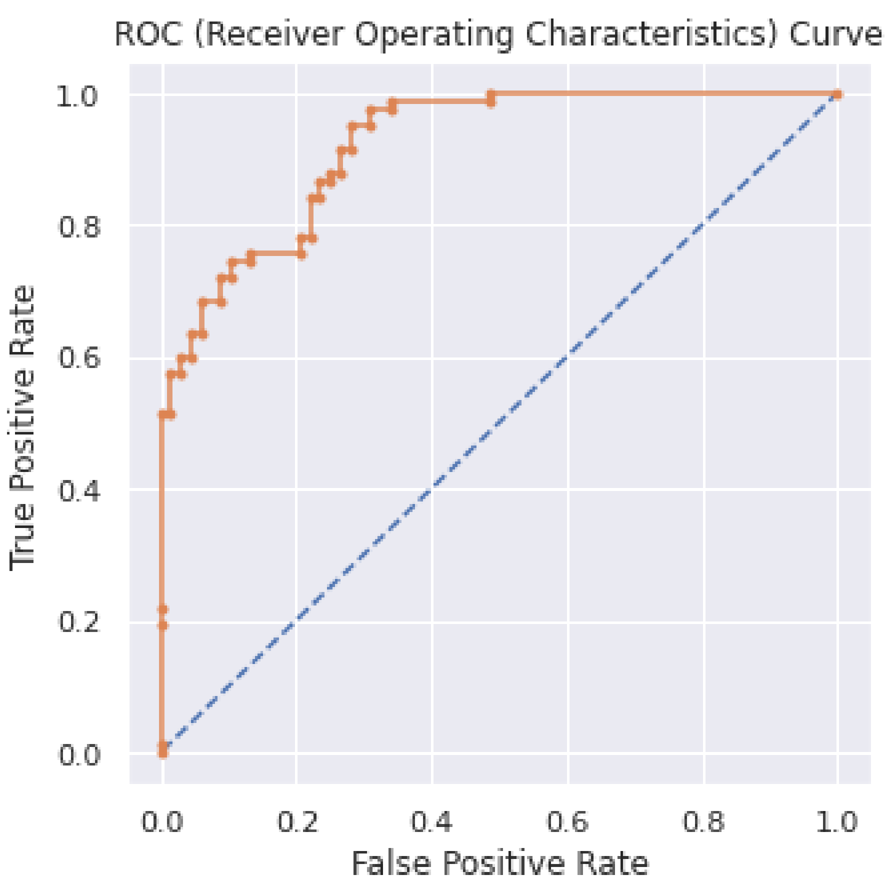
[150]: 0.9976470588235294

[151]:
```
xgb2.score(X_test, y_test)
```

[151]: 0.8066666666666666

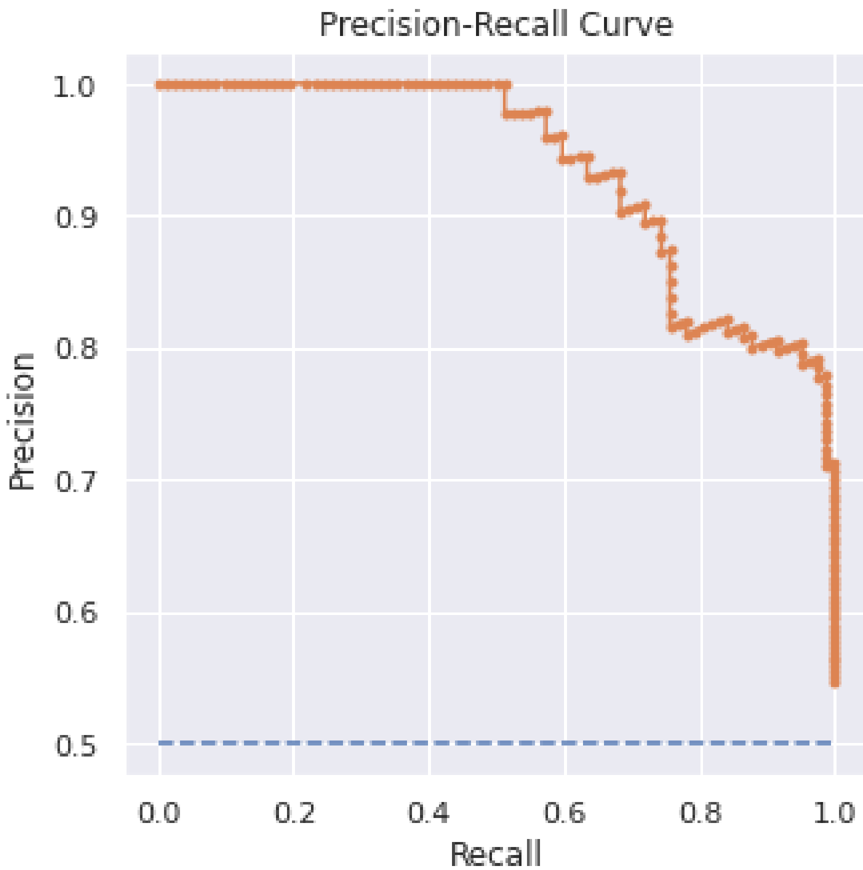
[152]:
```
probs = xgb2.predict_proba(X_test)                    # predict probabilities
probs = probs[:, 1]                                   # keep probabilities for the␣
 ↪positive outcome only

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



[153]:
```
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");
```

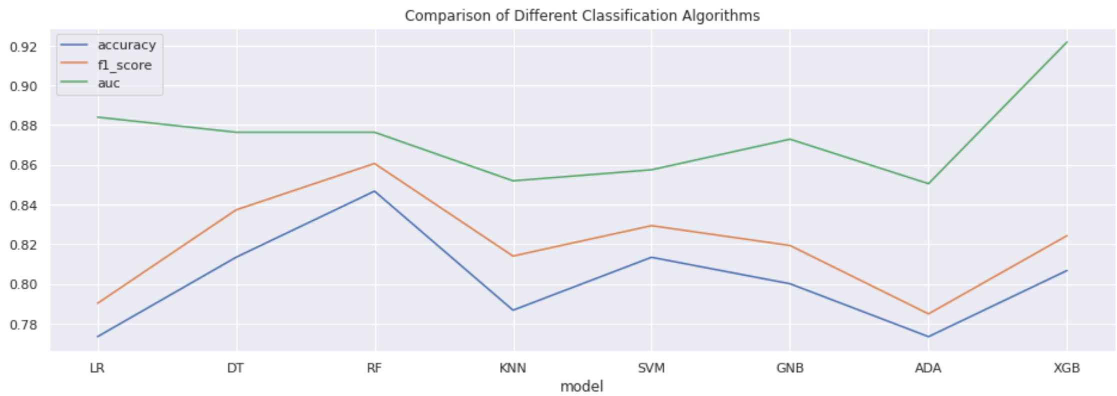
```
f1=0.824 auc_pr=0.936 ap=0.937
```



[154]:
```
models.append('XGB')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_xgb)
```

[155]:
```
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')
```

[156]:
```
model_summary.plot(figsize=(16,5))
plt.title("Comparison of Different Classification Algorithms");
```



[157]:
```
model_summary
```

[157]:
```
       accuracy  f1_score       auc
model
LR     0.773333  0.790123  0.883967
DT     0.813333  0.837209  0.876345
RF     0.846667  0.860606  0.876345
KNN    0.786667  0.813953  0.851865
SVM    0.813333  0.829268  0.857425
GNB    0.800000  0.819277  0.872848
ADA    0.773333  0.784810  0.850430
XGB    0.806667  0.824242  0.921808
```

[158]:
```
final_model = rf2
```

[159]:
```
cr = classification_report(y_test, final_model.predict(X_test))
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        68
           1       0.86      0.87      0.86        82

    accuracy                           0.85       150
   macro avg       0.85      0.84      0.85       150
weighted avg       0.85      0.85      0.85       150
```

[160]:
```
confusion = confusion_matrix(y_test, final_model.predict(X_test))
print("Confusion Matrix:\n", confusion)
```

```
Confusion Matrix:
 [[56 12]
```

```
 [11 71]]
```

[161]:
```
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)
```

[162]:
```
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: 0.855
Sensitivity: 0.866
Specificity: 0.824
AUC: 0.930
```

[ ]:

[ ]: