In [1]:
```python
# import the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:
```python
df = pd.read_csv('health care diabetes.csv')
```

In [4]:
```python
df.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [6]:
```python
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)
```

In [7]:
```python
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
```

In [8]:
```python
df.isnull().sum()
```

Out[8]:
```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

In [9]:
```python
df.describe()
```

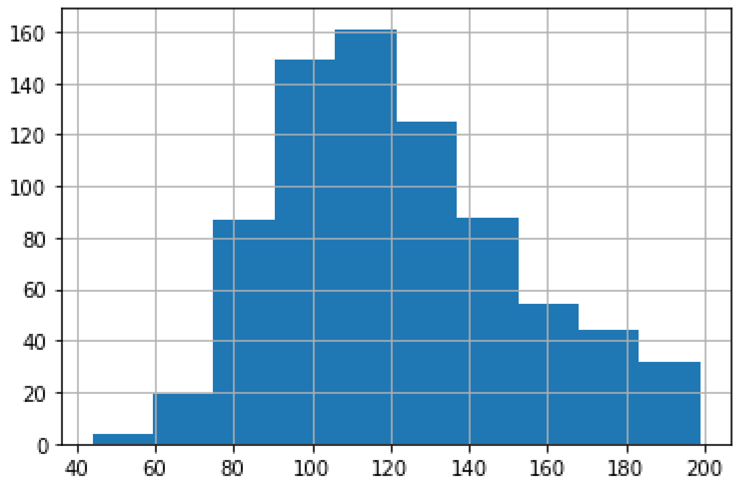
Out[9]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
count	768.000000	763.000000	733.000000	541.000000	394.000000	757.000000	768.000000

mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876
std	3.369578	30.535641	12.382158	10.476982	118.775855	6.924988	0.331329
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.500000	0.243750
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.300000	0.372500
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.600000	0.626250
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000

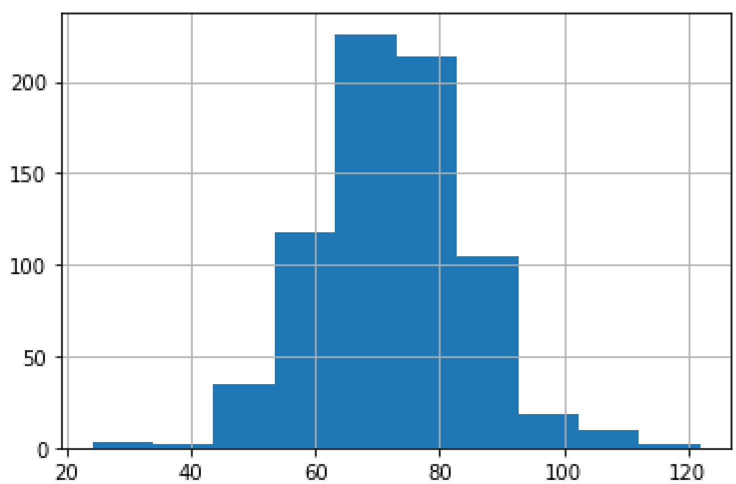
In [10]:

```
# Visually explore these variables using histograms. Treat the missing values accordingl
df['Glucose'].hist();
```



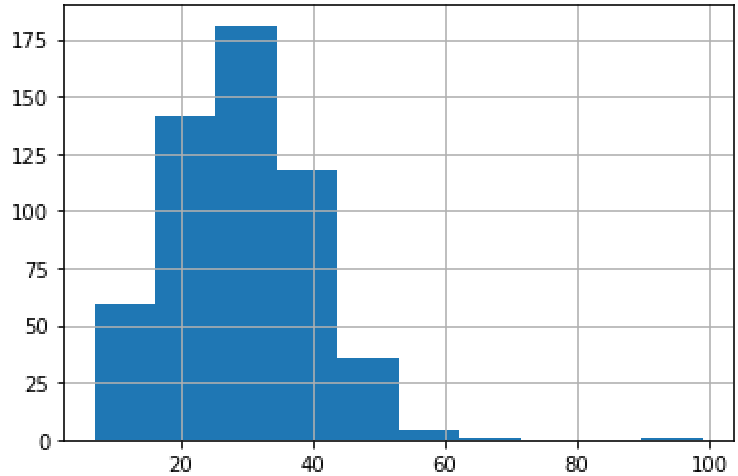
In [11]:

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

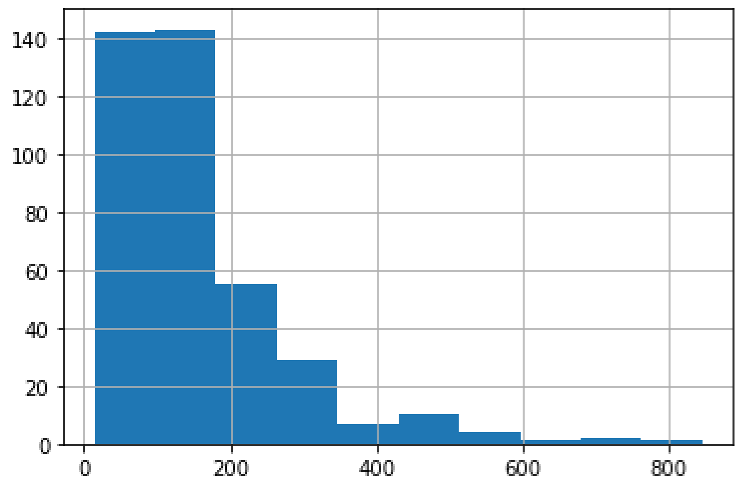


In [12]:

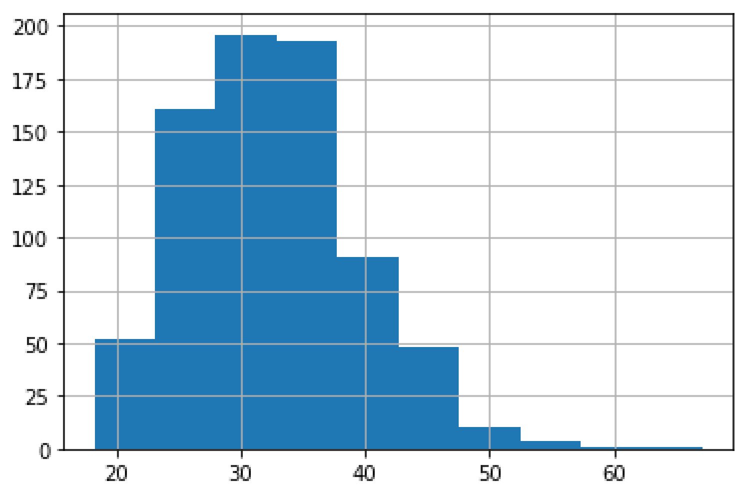
```
df['SkinThickness'].hist();
```



In [14]:
```
df['Insulin'].hist();
```



In [15]:
```
df['BMI'].hist();
```



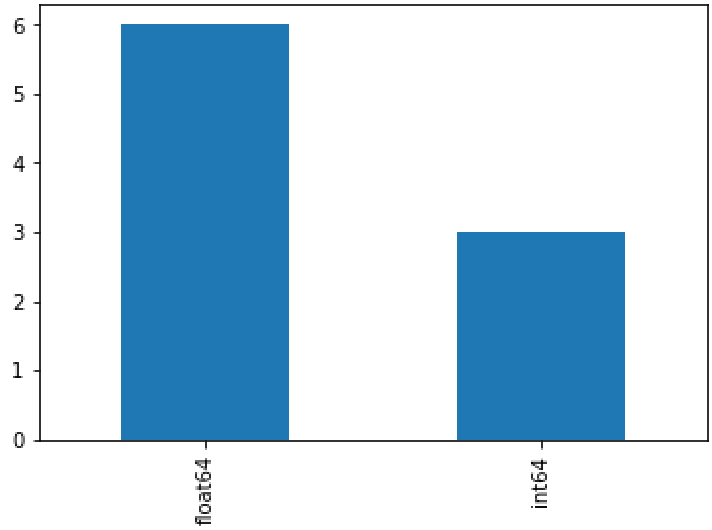
In [16]:
```
df['Insulin'] = df['Insulin'].fillna(df['Insulin'].median())
```

In [18]:
```
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())
```

There are integer and float data type variables in this dataset. Create a count (frequency) plot

describing the data types and the count of variables.

In [20]:
```
df.dtypes.value_counts().plot(kind='bar');
```

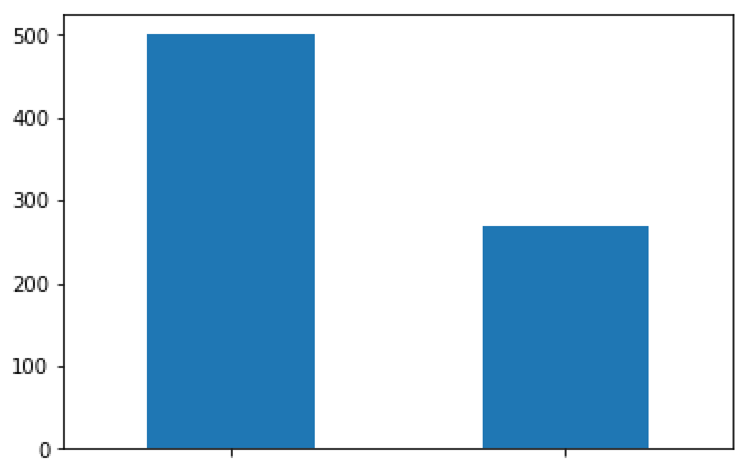


Data Exploration:

1. Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

In [22]:
```
df['Outcome'].value_counts().plot(kind='bar')
df['Outcome'].value_counts()
```

Out[22]:
```
0    500
1    268
Name: Outcome, dtype: int64
```



In [23]:
```
df_X = df.drop('Outcome', axis=1)
df_y = df['Outcome']
print(df_X.shape, df_y.shape)
```

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

In [27]:
```
!pip install imblearn
```

```
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)
Requirement already satisfied: numpy>=1.17.3 in c:\users\91992.laptop-e0op569h\anaconda3
\lib\site-packages (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: scipy>=1.3.2 in c:\users\91992.laptop-e0op569h\anaconda3
\lib\site-packages (from imbalanced-learn->imblearn) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\91992.laptop-e0op569h\an
aconda3\lib\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\91992.laptop-e0op569h\ana
conda3\lib\site-packages (from imbalanced-learn->imblearn) (1.0.2)
Collecting joblib>=1.1.1
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
Installing collected packages: joblib, imbalanced-learn, imblearn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
Successfully installed imbalanced-learn-0.10.1 imblearn-0.0 joblib-1.2.0
```

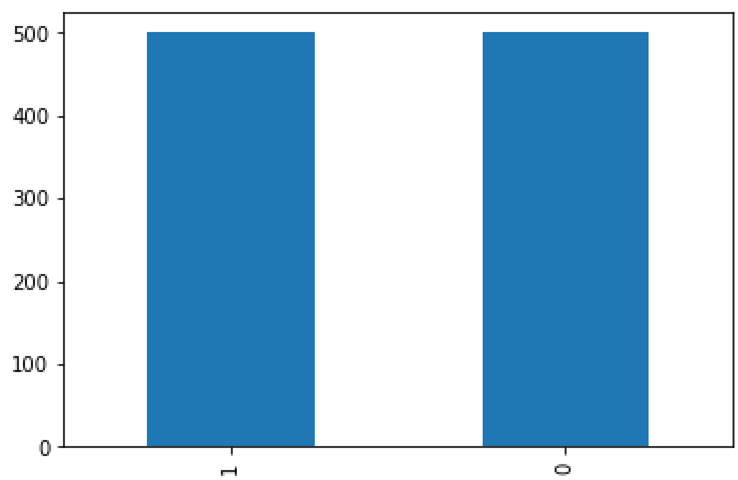
In [28]:
```
from imblearn.over_sampling import SMOTE
```

In [29]:
```
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,)
```

In [30]:
```
df_y_resampled.value_counts().plot(kind='bar')
df_y_resampled.value_counts()
```

Out[30]:
```
1    500
0    500
Name: Outcome, dtype: int64
```



In [32]:
```
df_resampled = pd.concat([df_X_resampled, df_y_resampled],axis=1)
df_resampled
```

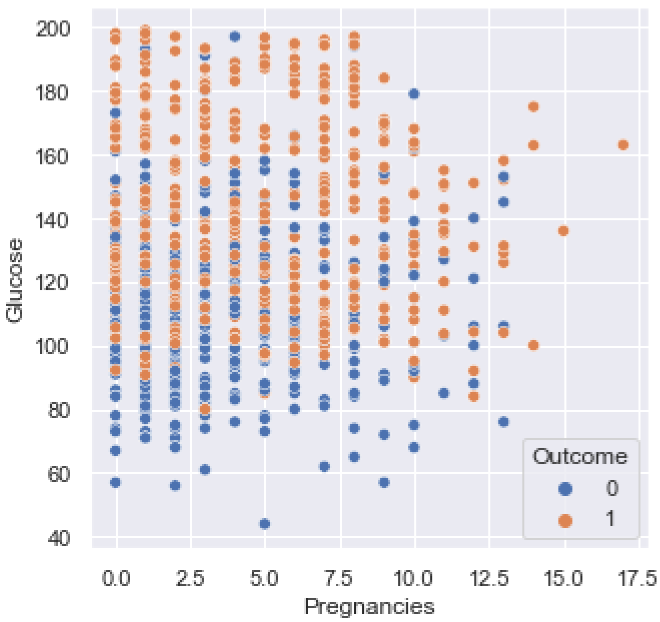
Out[32]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	6	148.000000	72.000000	35.000000	125.000000	33.600000	0.627000	5
1	1	85.000000	66.000000	29.000000	125.000000	26.600000	0.351000	3
2	8	183.000000	64.000000	29.153420	125.000000	23.300000	0.672000	3
3	1	89.000000	66.000000	23.000000	94.000000	28.100000	0.167000	2

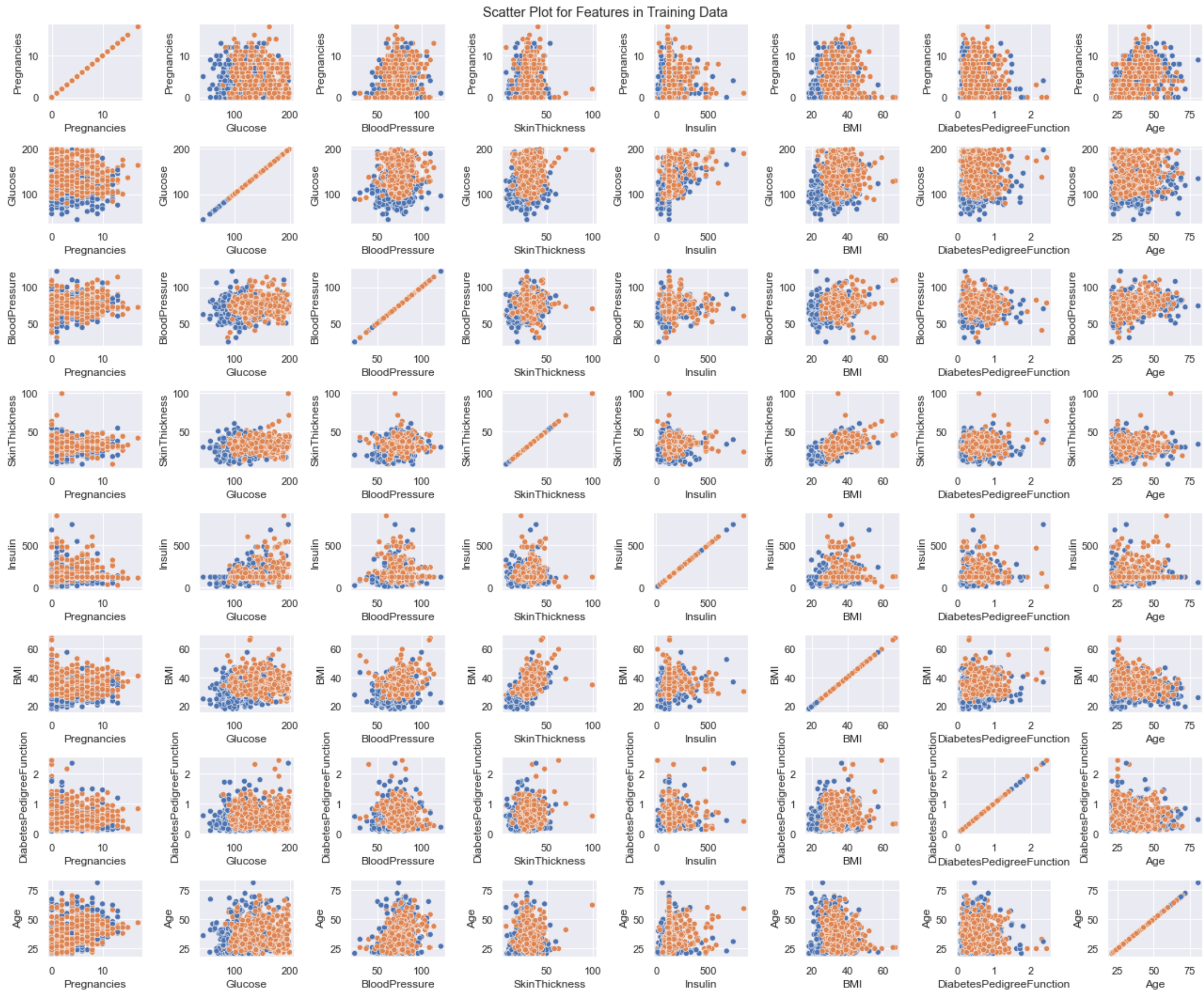
4	0	137.000000	40.000000	35.000000	168.000000	43.100000	2.288000	3
...	...	...	...	...	...	...	...	.
995	3	164.686765	74.249021	29.153420	125.000000	42.767110	0.726091	2
996	0	138.913540	69.022720	27.713033	127.283849	39.177649	0.703702	2
997	10	131.497740	66.331574	33.149837	125.000000	45.820819	0.498032	3
998	0	105.571347	83.238205	29.153420	125.000000	27.728596	0.649204	6
999	0	127.727025	108.908879	44.468195	129.545366	65.808840	0.308998	2

1000 rows × 9 columns

In [33]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.scatterplot(x="Pregnancies", y="Glucose", data=df_resampled, hue="Outcome");
```



In [34]:
```
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

plt.tight_layout()
```



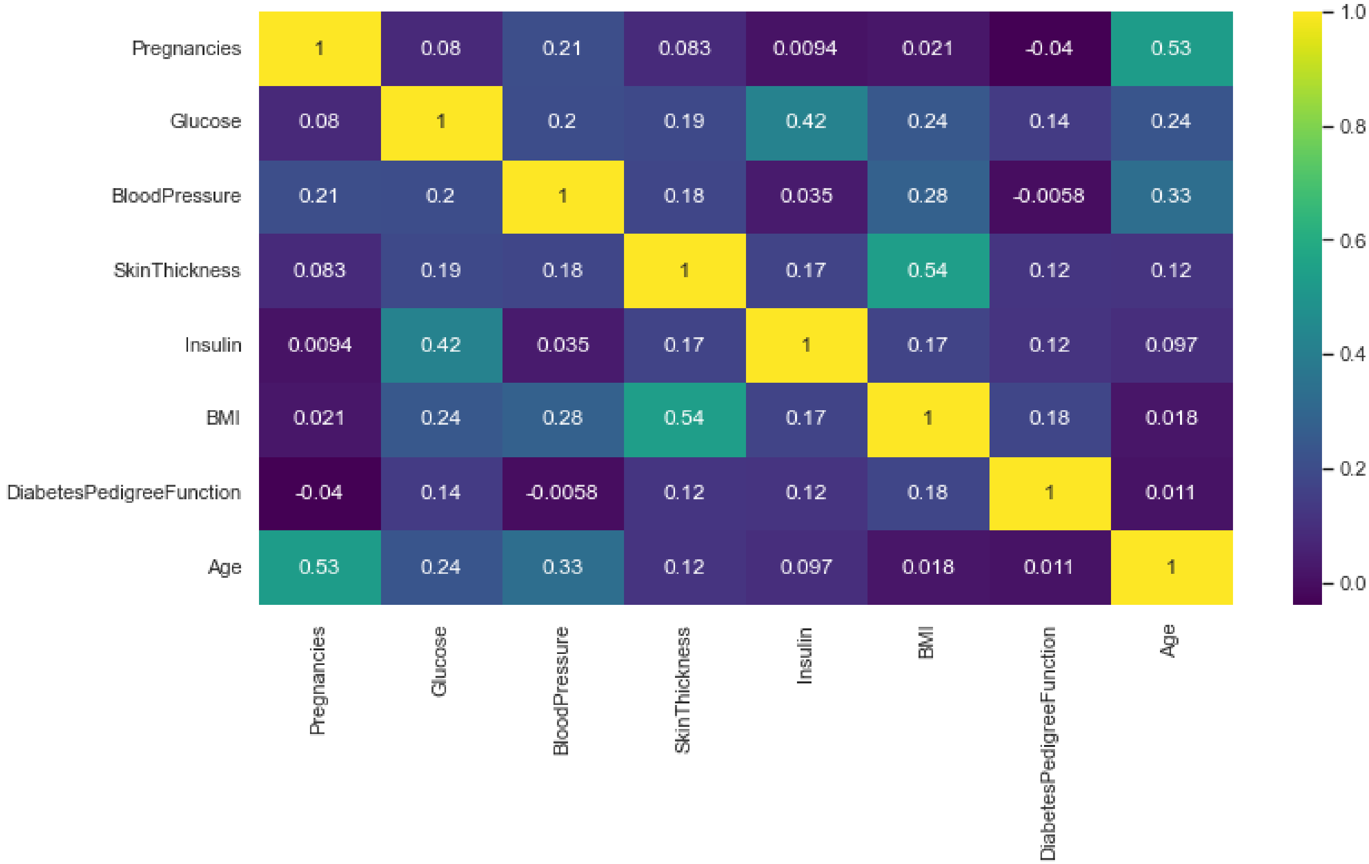
In [35]:
```python
df_X_resampled.corr()
```

Out[35]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
Pregnancies	1.000000	0.079953	0.205232	0.082752	0.009365	0.021006	
Glucose	0.079953	1.000000	0.200717	0.189776	0.418830	0.242501	
BloodPressure	0.205232	0.200717	1.000000	0.176496	0.034861	0.277565	
SkinThickness	0.082752	0.189776	0.176496	1.000000	0.170719	0.538207	
Insulin	0.009365	0.418830	0.034861	0.170719	1.000000	0.168702	
BMI	0.021006	0.242501	0.277565	0.538207	0.168702	1.000000	
DiabetesPedigreeFunction	-0.040210	0.138945	-0.005850	0.120799	0.115187	0.177915	
Age	0.532660	0.235522	0.332015	0.117644	0.096940	0.017529	

In [38]:
```python
plt.figure(figsize=(12,6))
sns.heatmap(df_X_resampled.corr(),cmap='viridis',annot=True)
```

Out[38]: <AxesSubplot:>



In [42]:
```
from sklearn.model_selection import train_test_split, KFold, RandomizedSearchCV
from sklearn.metrics import accuracy_score, average_precision_score, f1_score, confusion
```

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(df_X_resampled, df_y_resampled, test
```

In [44]:
```
X_train.shape, X_test.shape
```

Out[44]:
```
((850, 8), (150, 8))
```

In [45]:
```
models = []
model_accuracy = []
model_f1 = []
model_auc = []
```

In [46]:
```
from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression(max_iter=300)
```

In [47]:
```
lr1.fit(X_train,y_train)
```

Out[47]:
```
LogisticRegression(max_iter=300)
```

In [48]:
```
lr1.score(X_train,y_train)
```

Out[48]:
```
0.7294117647058823
```

In [49]:
```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

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

In [51]:
```
gs_lr = GridSearchCV(lr1, param_grid = parameters, cv=5, verbose=0)
gs_lr.fit(df_X_resampled, df_y_resampled)
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=300),
```

Out[51]:

```
             param_grid={'C': array([1.00000000e-05, 1.59985872e-05, 2.55954792e-05, 4.0
9491506e-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])})
```

In [52]:

```
gs_lr.best_params_
```

Out[52]:

```
{'C': 13.257113655901108}
```

In [53]:

```
gs_lr.best_score_
```

Out[53]:

```
0.738
```

In [54]:

```
lr2 = LogisticRegression(C=13.257113655901108, max_iter=300)
```

In [55]:

```
lr2.fit(X_train,y_train)
```

Out[55]:

```
LogisticRegression(C=13.257113655901108, max_iter=300)
```

In [56]:

```
lr2.score(X_train,y_train)
```

Out[56]:

```
0.7305882352941176
```

In [57]:

```
lr2.score(X_test, y_test)
```

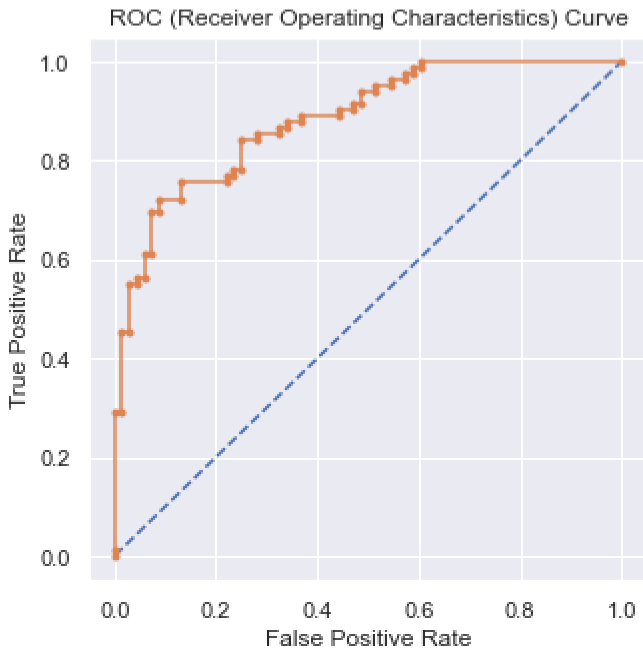
Out[57]:

```
0.7733333333333333
```

In [58]:

```
probs = lr2.predict_proba(X_test)                    # predict probabilities
probs = probs[:, 1]                                  # keep probabilities for the positive o

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
```

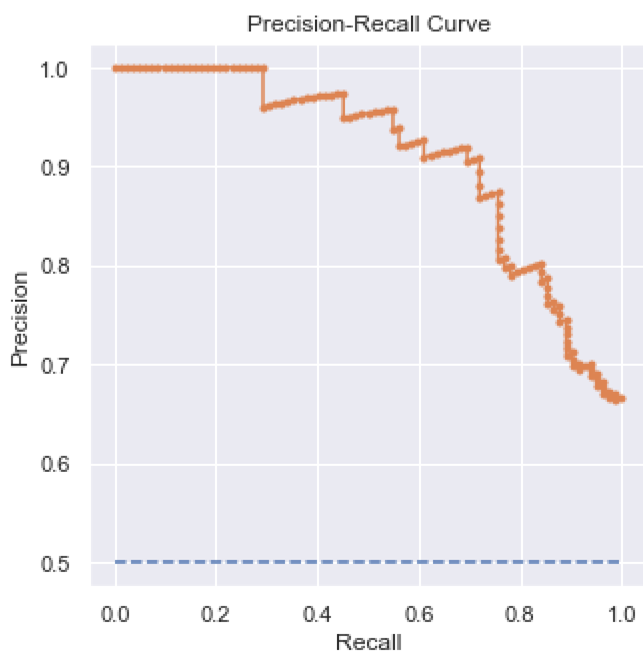
```
AUC: 0.884
```



In [61]:
```
from sklearn.metrics import precision_recall_curve
```

In [62]:
```
pred_y_test = lr2.predict(X_test)                                           # predict class va
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                         # calculate F1 sco
auc_lr_pr = auc(recall, precision)                                         # calculate precis
ap = average_precision_score(y_test, probs)                                # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

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



In [63]:
```
models.append('LR')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_lr)
```

In [64]:
```
from sklearn.tree import DecisionTreeClassifier
```

```python
dt1 = DecisionTreeClassifier(random_state=0)
```

In [65]:
```python
dt1.fit(X_train,y_train)
```
Out[65]:
```
DecisionTreeClassifier(random_state=0)
```

In [66]:
```python
dt1.score(X_train,y_train)
```
Out[66]:
```
1.0
```

In [67]:
```python
dt1.score(X_test, y_test)
```
Out[67]:
```
0.7733333333333333
```

In [68]:
```python
parameters = {
    'max_depth':[1,2,3,4,5,None]
}
```

In [69]:
```python
gs_dt = GridSearchCV(dt1, param_grid = parameters, cv=5, verbose=0)
gs_dt.fit(df_X_resampled, df_y_resampled)
```
Out[69]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'max_depth': [1, 2, 3, 4, 5, None]})
```

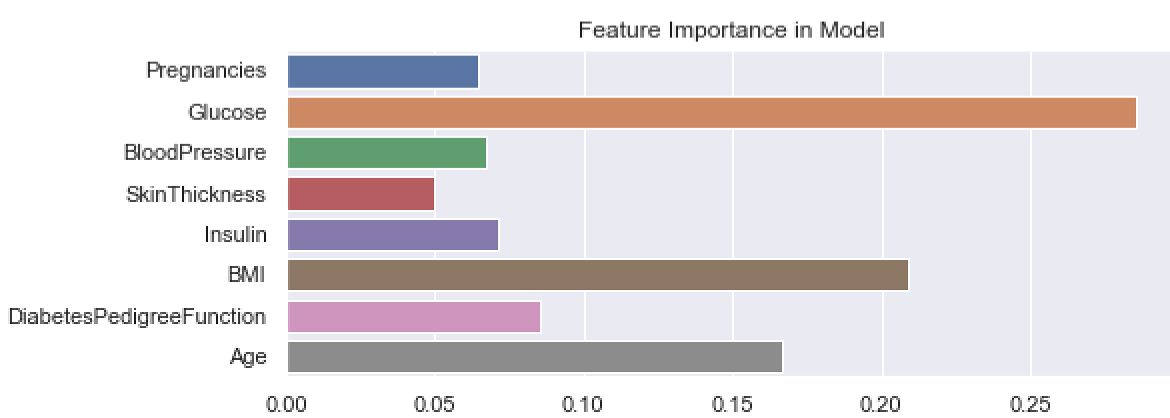
In [70]:
```python
gs_dt.best_params_
```
Out[70]:
```
{'max_depth': 4}
```

In [71]:
```python
gs_dt.best_score_
```
Out[71]:
```
0.76
```

In [72]:
```python
dt1.feature_importances_
```
Out[72]:
```
array([0.06452226, 0.28556999, 0.06715314, 0.04979714, 0.07150365,
       0.20905992, 0.08573109, 0.16666279])
```

In [73]:
```python
X_train.columns
```
Out[73]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

In [75]:
```python
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");
```



In [79]:
```
dt2 = DecisionTreeClassifier(max_depth=4)
```

In [80]:
```
dt2.fit(X_train,y_train)
```

Out[80]:
```
DecisionTreeClassifier(max_depth=4)
```

In [81]:
```
dt2.score(X_train,y_train)
```

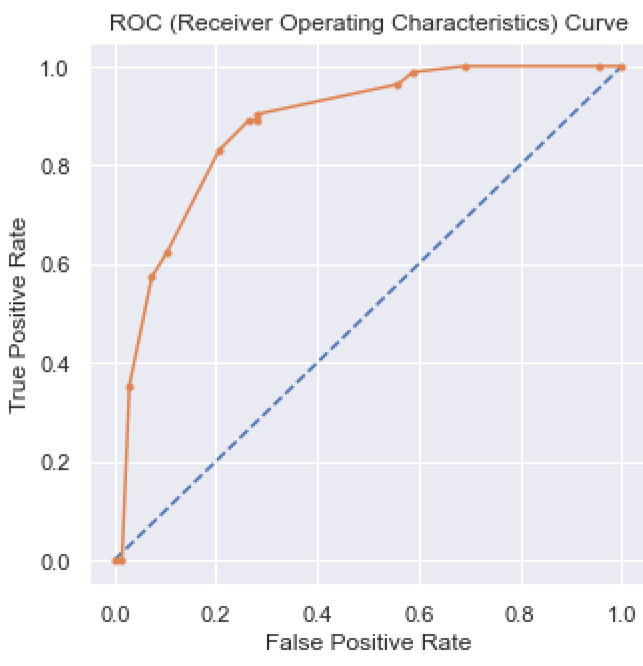
Out[81]:
```
0.8070588235294117
```

In [82]:
```
dt2.score(X_test, y_test)
```

Out[82]:
```
0.82
```

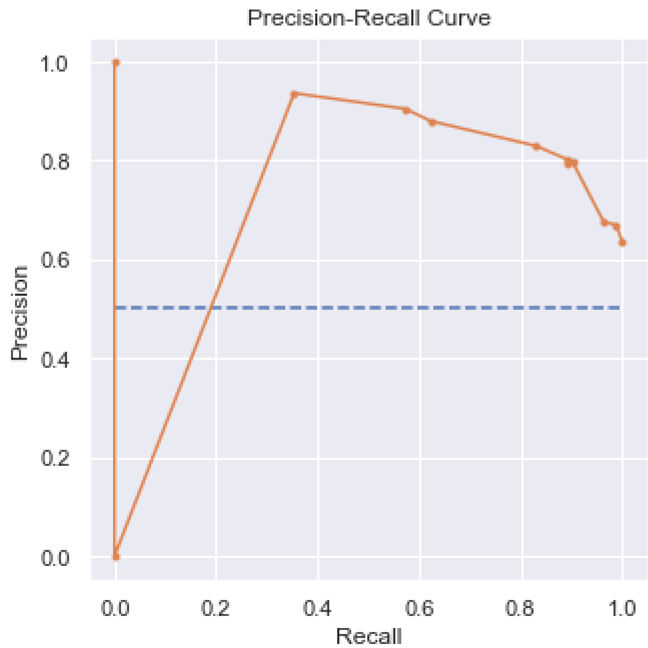
In [83]:
```
probs = dt2.predict_proba(X_test)                      # predict probabilities
probs = probs[:, 1]                                    # keep probabilities for the positive o

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
```



In [84]:
```
pred_y_test = dt2.predict(X_test)                                    # predict class va
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                   # calculate F1 sco
auc_dt_pr = auc(recall, precision)                                   # calculate precis
ap = average_precision_score(y_test, probs)                          # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.844 auc_pr=0.717 ap=0.868
```



In [85]:
```
models.append('DT')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_dt)
```

In [86]:
```
from sklearn.ensemble import RandomForestClassifier
rf1 = RandomForestClassifier()
```

In [87]:
```
rf1 = RandomForestClassifier(random_state=0)
```

In [88]:
```
rf1.fit(X_train, y_train)
```

Out[88]:
```
RandomForestClassifier(random_state=0)
```

In [89]:
```
rf1.score(X_train, y_train)
```

Out[89]:
```
1.0
```

In [90]:
```
rf1.score(X_test, y_test)
```

Out[90]:
```
0.8466666666666667
```

In [91]:
```
parameters = {
    'n_estimators': [50,100,150],
    'max_depth': [None,1,3,5,7],
    'min_samples_leaf': [1,3,5]
}
```

In [92]:
```
gs_dt = GridSearchCV(estimator=rf1, param_grid=parameters, cv=5, verbose=0)
gs_dt.fit(df_X_resampled, df_y_resampled)
```

Out[92]:
```
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]})
```

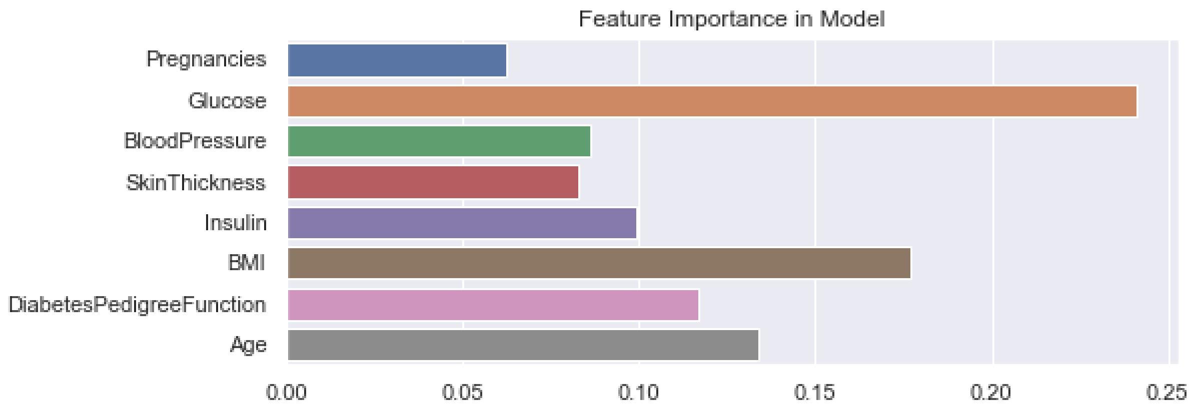
In [93]:
```
gs_dt.best_params_
```

Out[93]:
```
{'max_depth': None, 'min_samples_leaf': 1, 'n_estimators': 100}
```

In [94]:
```
gs_dt.best_score_
```
Out[94]:
```
0.813
```

In [95]:
```
rf1.feature_importances_
```
Out[95]:
```
array([0.06264995, 0.24106573, 0.08653626, 0.08301549, 0.09945063,
       0.17678287, 0.11685244, 0.13364664])
```

In [96]:
```
plt.figure(figsize=(8,3))
sns.barplot(y=X_train.columns, x=rf1.feature_importances_);
plt.title("Feature Importance in Model");
```



In [97]:
```
rf2 = RandomForestClassifier(max_depth=None, min_samples_leaf=1, n_estimators=100)
```

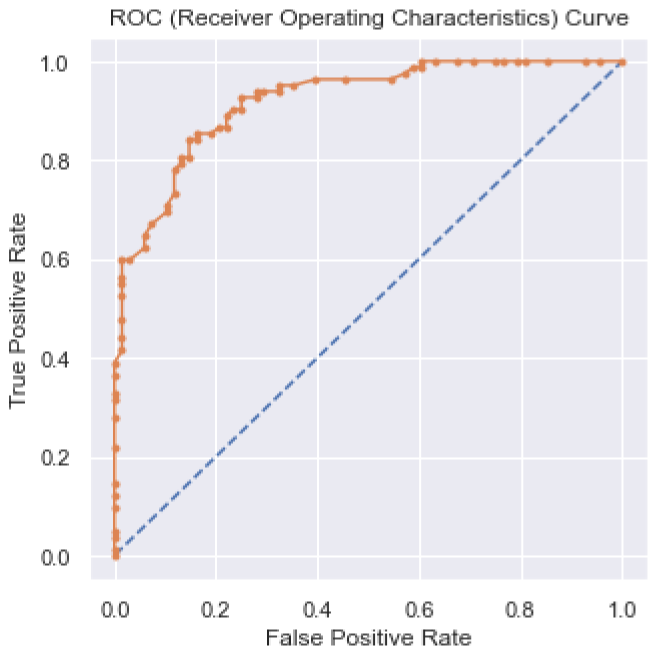
In [98]:
```
rf2.fit(X_train,y_train)
```
Out[98]:
```
RandomForestClassifier()
```

In [99]:
```
rf2.score(X_train,y_train)
```
Out[99]:
```
1.0
```

In [100…
```
rf2.score(X_test, y_test)
```
Out[100]:
```
0.8466666666666667
```

In [101…
```
probs = rf2.predict_proba(X_test)                   # predict probabilities
probs = probs[:, 1]                                 # keep probabilities for the positive o

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");
```

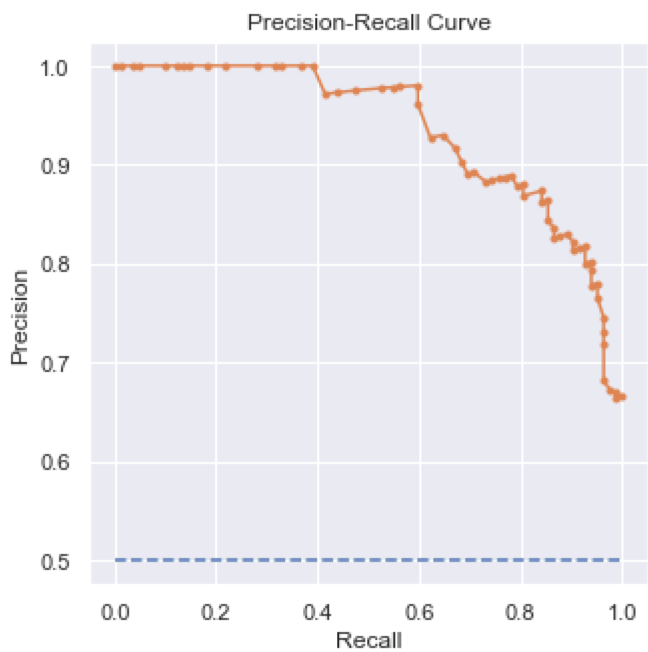
```
AUC: 0.921
```



In [102…

```
pred_y_test = rf2.predict(X_test)                                    # predict class va
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                   # calculate F1 sco
auc_rf_pr = auc(recall, precision)                                   # calculate precis
ap = average_precision_score(y_test, probs)                          # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.859 auc_pr=0.936 ap=0.935
```

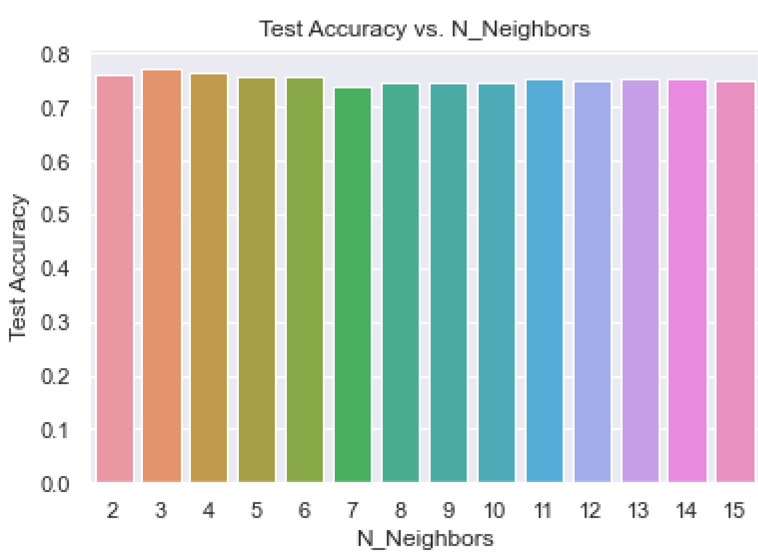


In [103…

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

In [104…

```
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=3)
```

In [105...
```
knn1.fit(X_train, y_train)
```
Out[105]:
```
KNeighborsClassifier(n_neighbors=3)
```
In [106...
```
knn1.score(X_train,y_train)
```
Out[106]:
```
0.8835294117647059
```
In [107...
```
knn1.score(X_test,y_test)
```
Out[107]:
```
0.7866666666666666
```
In [108...
```
knn_neighbors = [i for i in range(2,16)]
parameters = {
    'n_neighbors': knn_neighbors
}
```
In [109...
```
gs_knn = GridSearchCV(estimator=knn1, param_grid=parameters, cv=5, verbose=0)
gs_knn.fit(df_X_resampled, df_y_resampled)
```
Out[109]:
```
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]})
```
In [110...
```
gs_knn.best_params_
```
Out[110]:
```
{'n_neighbors': 3}
```
In [111...
```
gs_knn.best_score_
```
Out[111]:
```
0.771
```
In [112...
```
gs_knn.cv_results_['mean_test_score']
```
Out[112]:
```
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])
```
In [113...
```
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");
```


In [114...
```
knn2 = KNeighborsClassifier(n_neighbors=3)
```

In [115…

```
knn2.fit(X_train, y_train)
```

Out[115]:

```
KNeighborsClassifier(n_neighbors=3)
```

In [116…

```
knn2.score(X_train,y_train)
```

Out[116]:

```
0.8835294117647059
```

In [117…

```
knn2.score(X_test,y_test)
```

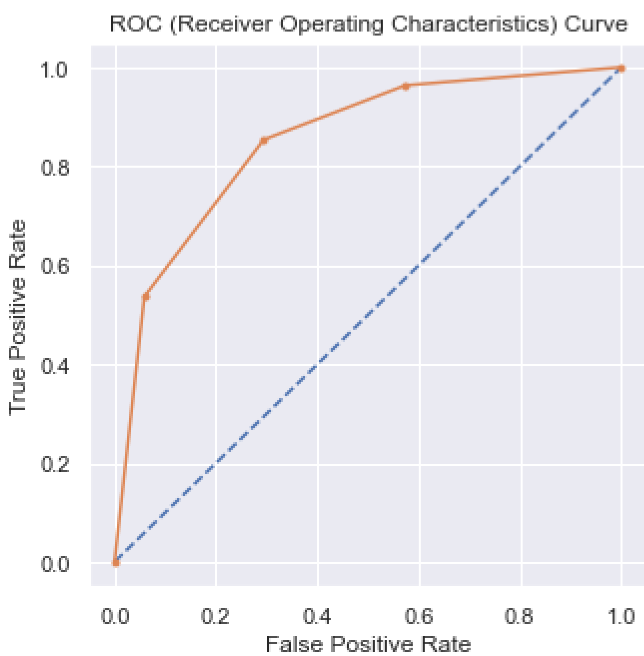
Out[117]:

```
0.7866666666666666
```

In [118…

```
probs = knn2.predict_proba(X_test)                    # predict probabilities
probs = probs[:, 1]                                   # keep probabilities for the positive o

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");
```

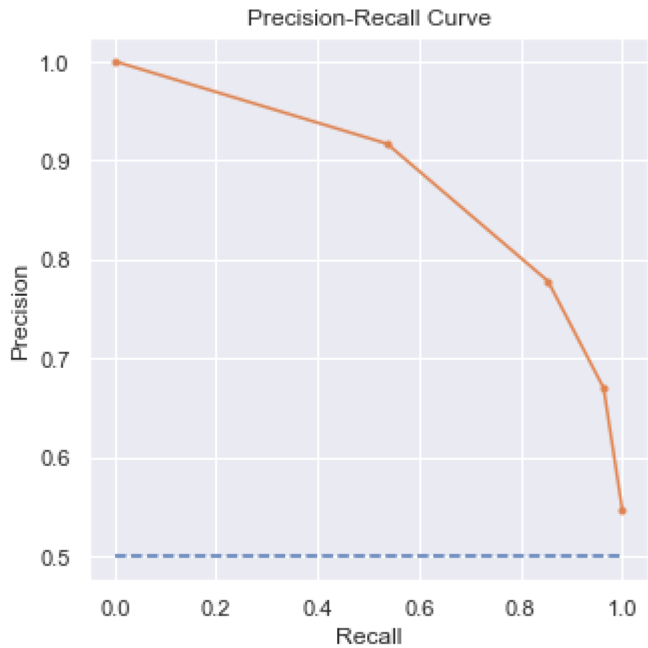
```
AUC: 0.852
```



In [119…

```
pred_y_test = knn2.predict(X_test)                                        # predict class v
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                      # calculate F1 sco
auc_knn_pr = auc(recall, precision)                                     # calculate preci
ap = average_precision_score(y_test, probs)                             # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

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



In [120…
```
models.append('KNN')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_knn)
```

In [121…
```
from sklearn.svm import SVC
svm1 = SVC(kernel='rbf')
```

In [122…
```
svm1.fit(X_train, y_train)
```

Out[122]:
```
SVC()
```

In [123…
```
svm1.score(X_train, y_train)
```

Out[123]:
```
0.7282352941176471
```

In [124…
```
svm1.score(X_test, y_test)
```

Out[124]:
```
0.78
```

In [125…
```
parameters = {
    'C':[1, 5, 10, 15, 20, 25],
    'gamma':[0.001, 0.005, 0.0001, 0.00001]
}
```

In [126…
```
gs_svm = GridSearchCV(estimator=svm1, param_grid=parameters, cv=5, verbose=0)
gs_svm.fit(df_X_resampled, df_y_resampled)
```

Out[126]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [1, 5, 10, 15, 20, 25],
                         'gamma': [0.001, 0.005, 0.0001, 1e-05]})
```

In [127…
```
gs_svm.best_params_
```

Out[127]:
```
{'C': 20, 'gamma': 0.005}
```

In [128…
```
gs_svm.best_score_
```

Out[128]:
```
0.8089999999999999
```

In [129…
```
svm2 = SVC(kernel='rbf', C=20, gamma=0.005, probability=True)
```

In [130…
```
svm2.fit(X_train, y_train)
```

Out[130]:
```
SVC(C=20, gamma=0.005, probability=True)
```

In [131…
```
svm2.score(X_train, y_train)
```

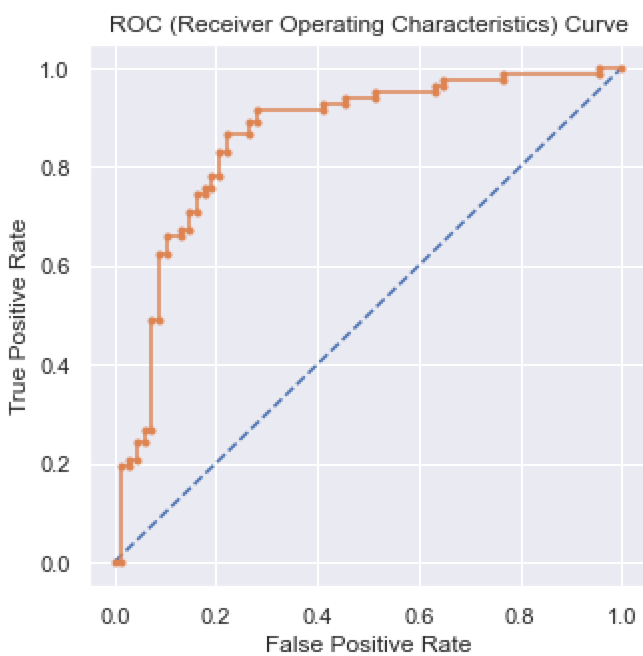
Out[131]:
```
0.9941176470588236
```

In [132…
```
svm2.score(X_test, y_test)
```

Out[132]:
```
0.8133333333333334
```

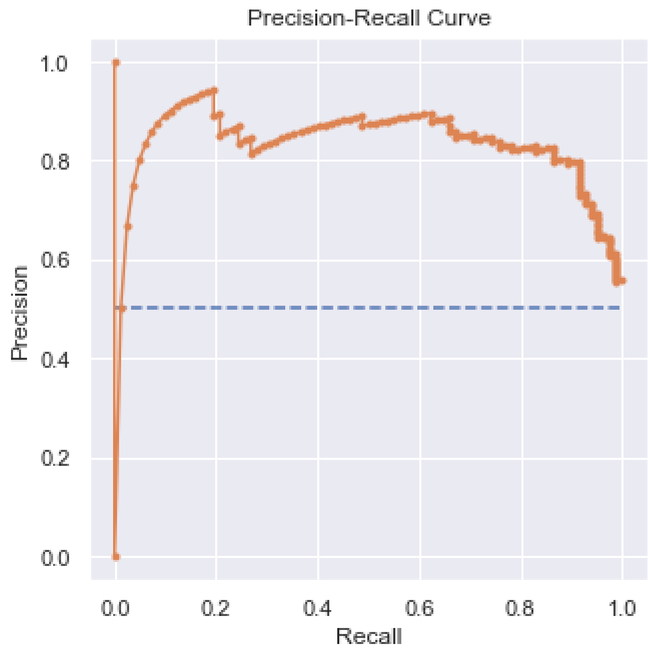
In [133…
```
probs = svm2.predict_proba(X_test)                    # predict probabilities
probs = probs[:, 1]                                   # keep probabilities for the positive o

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
```



In [134…
```
pred_y_test = svm2.predict(X_test)                                    # predict class va
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                    # calculate F1 sco
auc_svm_pr = auc(recall, precision)                                   # calculate precis
ap = average_precision_score(y_test, probs)                           # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.829 auc_pr=0.830 ap=0.837
```



In [135…

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

In [136…

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
gnb = GaussianNB()
```

In [137…

```
gnb.fit(X_train, y_train)
```

Out[137]:

```
GaussianNB()
```

In [138…

```
gnb.score(X_train, y_train)
```

Out[138]:

```
0.7294117647058823
```

In [139…

```
gnb.score(X_test, y_test)
```

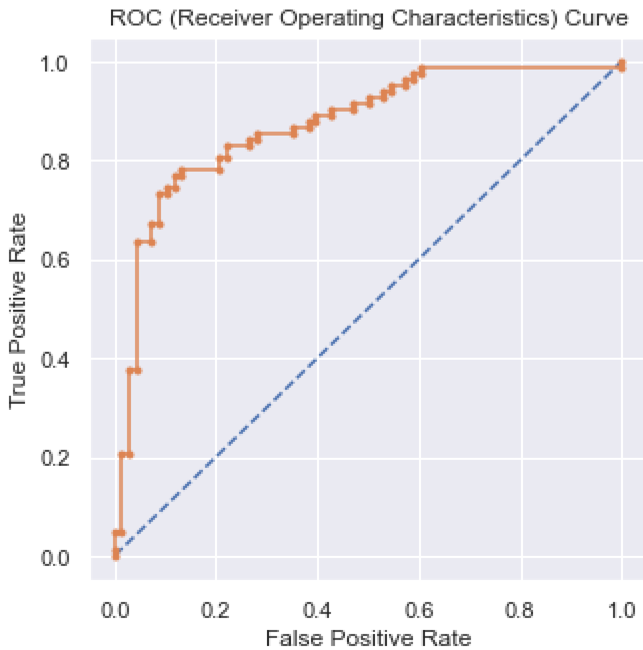
Out[139]:

```
0.8
```

In [140…

```
probs = gnb.predict_proba(X_test)                   # predict probabilities
probs = probs[:, 1]                                 # keep probabilities for the positive o

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");
```

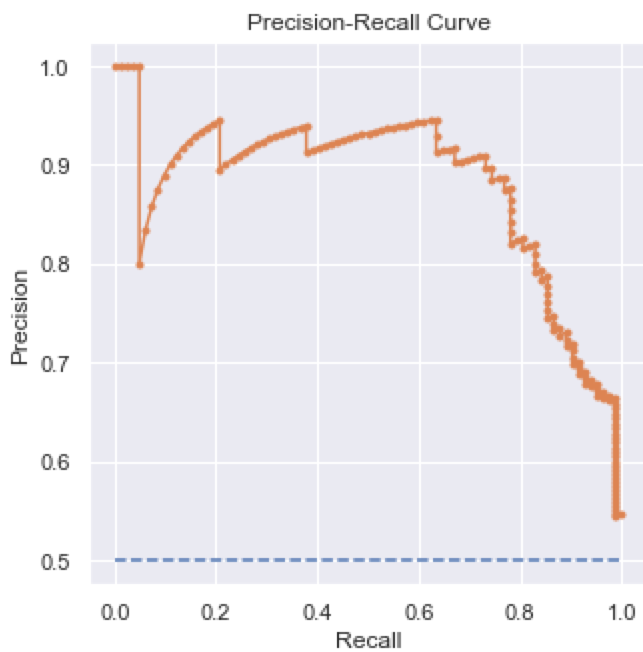
```
AUC: 0.873
```



In [141…

```
pred_y_test = gnb.predict(X_test)                                           # predict class va
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                          # calculate F1 sco
auc_gnb_pr = auc(recall, precision)                                          # calculate preci
ap = average_precision_score(y_test, probs)                                 # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

```
f1=0.819 auc_pr=0.879 ap=0.880
```



In [142…

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

In [143…

```
from xgboost import XGBClassifier
xgb1 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic', nthread=4,
```

```
C:\Users\91992.LAPTOP-E0OP569H\AppData\Roaming\Python\Python39\site-packages\xgboost\skl
earn.py:1421: UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

In [144…

```
xgb1.fit(X_train, y_train)
```

Out[144]:

```
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, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, 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', ...)
```

In [145…

```
xgb1.score(X_train, y_train)
```

Out[145]:

```
1.0
```

In [146…

```
xgb1.score(X_test, y_test)
```

Out[146]:

```
0.8266666666666667
```

In [147…

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

In [148…

```
gs_xgb = GridSearchCV(xgb1, param_grid = parameters, scoring = 'roc_auc', n_jobs = 10, c
gs_xgb.fit(df_X_resampled, df_y_resampled)
```

```
C:\Users\91992.LAPTOP-E0OP569H\AppData\Roaming\Python\Python39\site-packages\xgboost\skl
earn.py:1421: UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
C:\Users\91992.LAPTOP-E0OP569H\AppData\Roaming\Python\Python39\site-packages\xgboost\skl
earn.py:1421: UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

Out[148]:

```
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,
                                     feature_types=None, gamma=0, gpu_id=-1,
                                     grow_policy='depthwise',
                                     importance_type=None,
                                     interaction_constraints='',
                                     learning_rate=0.300000012...56,
                                     max_cat_threshold=64, 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', ...),
             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')
```

In [149…

```
gs_xgb.best_params_
```

Out[149]: {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 180}

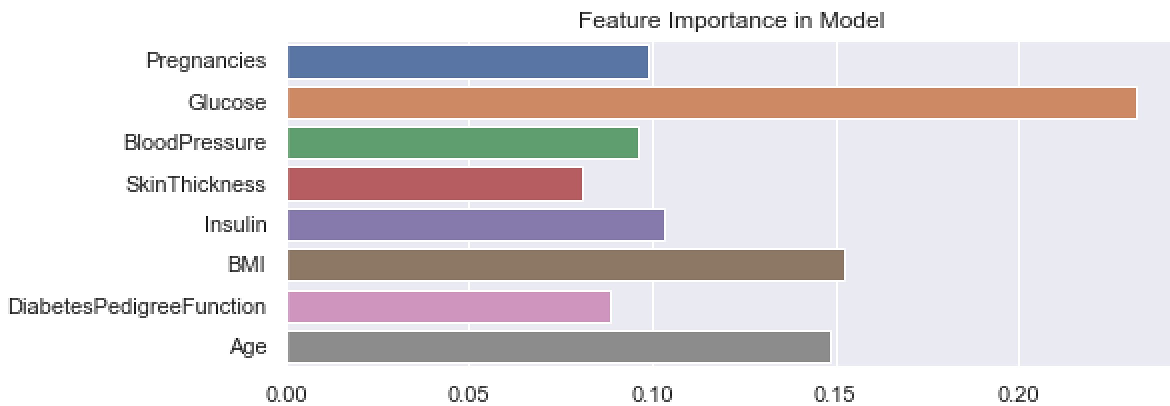
In [150…
```
gs_xgb.best_score_
```

Out[150]: 0.88522

In [151…
```
xgb1.feature_importances_
```

Out[151]:
```
array([0.09883169, 0.23199295, 0.09590794, 0.08073225, 0.10332596,
       0.15247223, 0.08829136, 0.1484456 ], dtype=float32)
```

In [152…
```
plt.figure(figsize=(8,3))
sns.barplot(y=X_train.columns, x=xgb1.feature_importances_)
plt.title("Feature Importance in Model");
```



In [153…
```
xgb2 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic',
                     nthread=4, seed=10, learning_rate= 0.05, max_depth= 7, n_estimators=
```

```
C:\Users\91992.LAPTOP-E0OP569H\AppData\Roaming\Python\Python39\site-packages\xgboost\skl
earn.py:1421: UserWarning: `use_label_encoder` is deprecated in 1.7.0.
  warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

In [154…
```
xgb2.fit(X_train,y_train)
```

Out[154]:
```
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, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.05, max_bin=256,
              max_cat_threshold=64, 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', ...)
```

In [155…
```
xgb2.score(X_train,y_train)
```

Out[155]: 0.9976470588235294

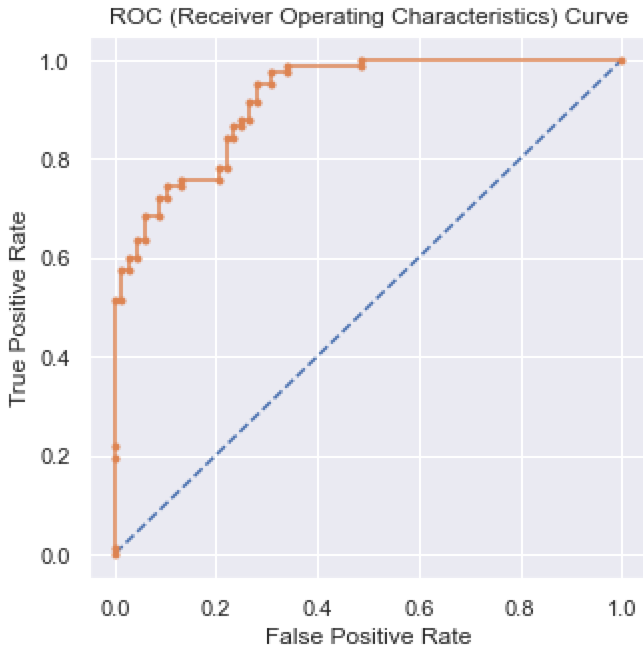
In [156…
```
xgb2.score(X_test, y_test)
```

Out[156]: 0.8066666666666666

In [157…
```
probs = xgb2.predict_proba(X_test)                    # predict probabilities
probs = probs[:, 1]                                  # keep probabilities for the positive o

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");
```

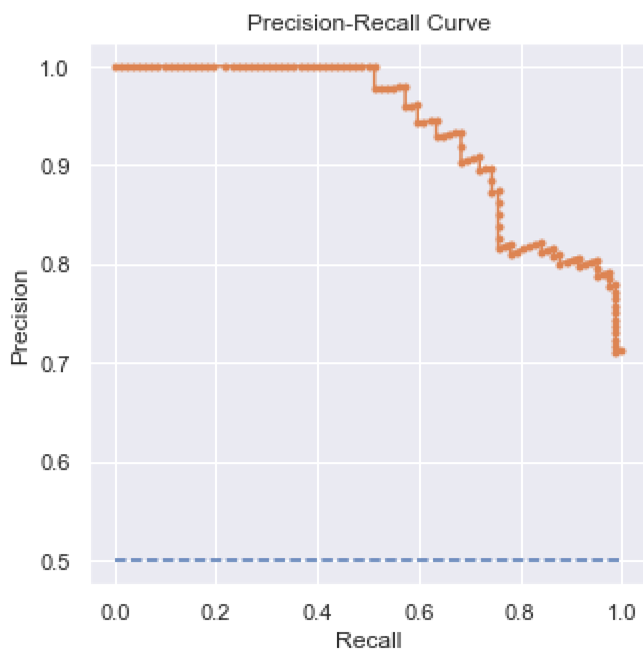
```
AUC: 0.922
```



In [158…

```
pred_y_test = xgb2.predict(X_test)                                          # predict class v
precision, recall, thresholds = precision_recall_curve(y_test, probs) # calculate precis
f1 = f1_score(y_test, pred_y_test)                                       # calculate F1 sco
auc_xgb_pr = auc(recall, precision)                                       # calculate preci
ap = average_precision_score(y_test, probs)                              # calculate averag
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 precisi
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve");
```

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



In [159…

```
models.append('XGB')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
```

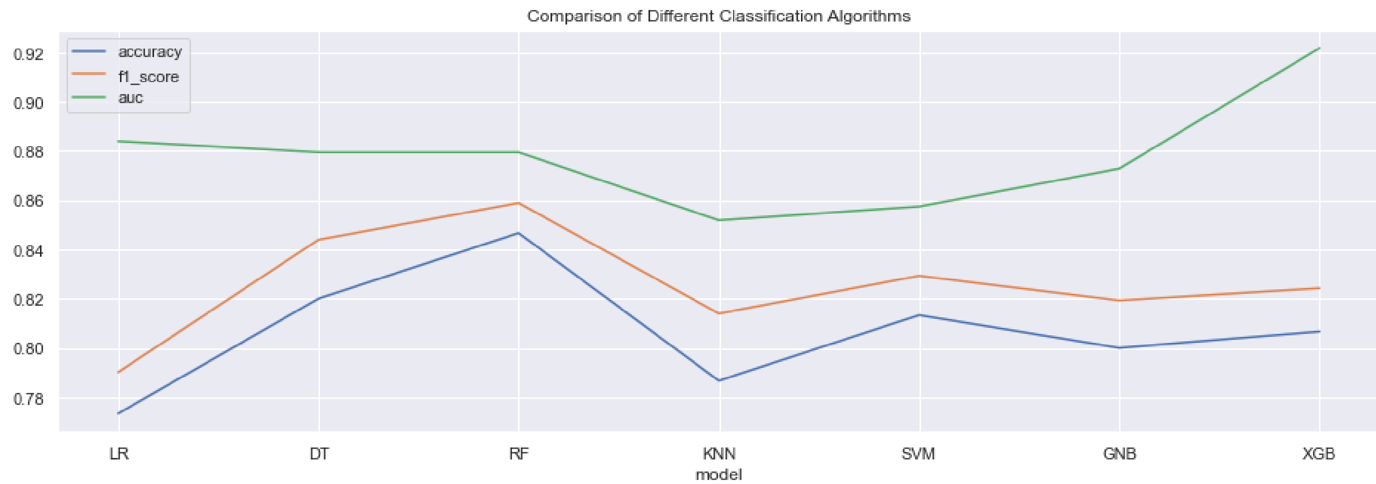
```
model_f1.append(f1)
model_auc.append(auc_xgb)
```

In [160…

```
model_summary = pd.DataFrame(zip(models,model_accuracy,model_f1,model_auc), columns = ['
model_summary = model_summary.set_index('model')
```

In [161…

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



In [162…

```
model_summary
```

Out[162]:

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	0.857425
GNB	0.800000	0.819277	0.872848
XGB	0.806667	0.824242	0.921808

In [163…

```
final_model = rf2
```

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.

In [164…

```
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
```

In [165…

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

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

In [166…

```
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)
```

In [167…

```
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: 0.921
```

Data Reporting:

1. 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

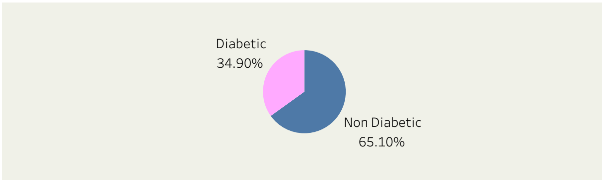
d. Heatmap of correlation analysis among the relevant variables

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.

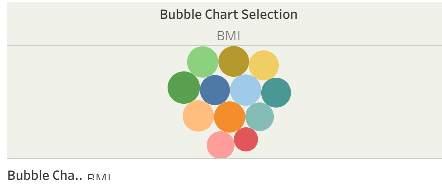
In [ ]:

HealthCare project

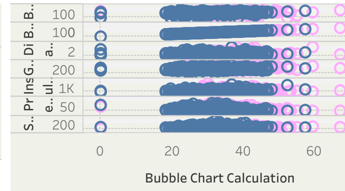
Diabetic And Non-Diabetic Population



Bubble Chart



Scatter Plot

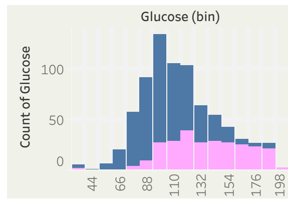


Heat Map

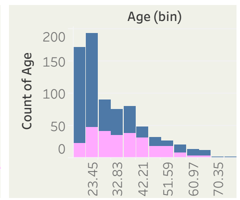
Age BIns Calculated Field

	20-25	25-30	30-35	35-40	40-45	45-50	50-55	55-60	60-65	65-70	70-75	80-85
Avg. Age	22.8	27.8	32.6	37.8	42.7	47.7	52.7	58.1	62.9	67.4	72.0	81.0
Avg. BMI	30.4	33.0	32.8	33.0	35.3	32.9	31.8	30.2	29.9	27.5	19.6	25.9
Avg. Blood ..	63.8	68.0	68.8	71.8	73.3	77.9	81.9	77.5	76.0	80.7	0.0	74.0
Avg. Bubble..	30.4	33.0	32.8	33.0	35.3	32.9	31.8	30.2	29.9	27.5	19.6	25.9
Avg. Diabet..	0.5	0.4	0.6	0.5	0.5	0.5	0.5	0.6	0.4	0.5	0.8	0.5
Avg. Glucose	110.7	120.3	124.2	128.3	125.1	124.5	143.2	138.3	136.4	139.0	119.0	134.0
Avg. Insulin	84.3	84.3	92.8	59.5	56.7	67.6	109.9	149.5	26.4	0.0	0.0	60.0
Avg. Numb..	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Avg. Pregn..	1.5	2.8	4.5	6.1	7.1	7.3	6.4	6.7	4.7	4.9	2.0	9.0
Avg. Skin T..	22.0	21.3	20.1	21.0	18.9	20.4	16.3	18.7	20.0	1.6	0.0	33.0

Glucose Distribution



Age Distribution



Blood Pressure Distribution