


My 3rd Project (Data Science Capstone)

```
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)
```

 C:\Users\snayi\AppData\Local\Temp\ipykernel_18460\3245060676.py:1: DeprecationWarning: Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0), (to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries) but was not found to be installed on your system. If this would cause problems for you, please provide us feedback at <https://github.com/pandas-dev/pandas/issues/54466>

```
import pandas as pd
```

```
df = pd.read_csv('health care diabetes.csv')
df.head()
```

	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

According to problem statement, a value of zero in the following columns indicates missing value:

- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI

We will replace zeros in these columns with null values.

```
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)
```

```
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
```

```
df.isnull().sum()
```

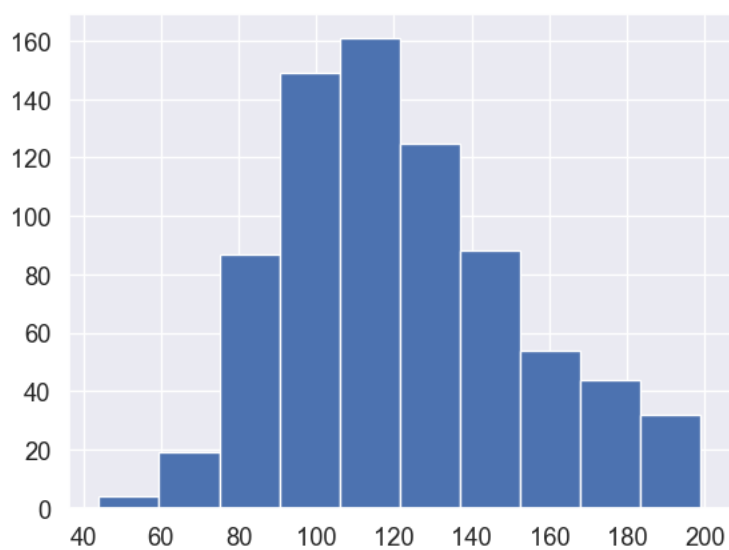
```
Pregnancies      0
Glucose           5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI              11
DiabetesPedigreeFunction 0
Age              0
Outcome          0
dtype: int64
```

```
df.describe()
```

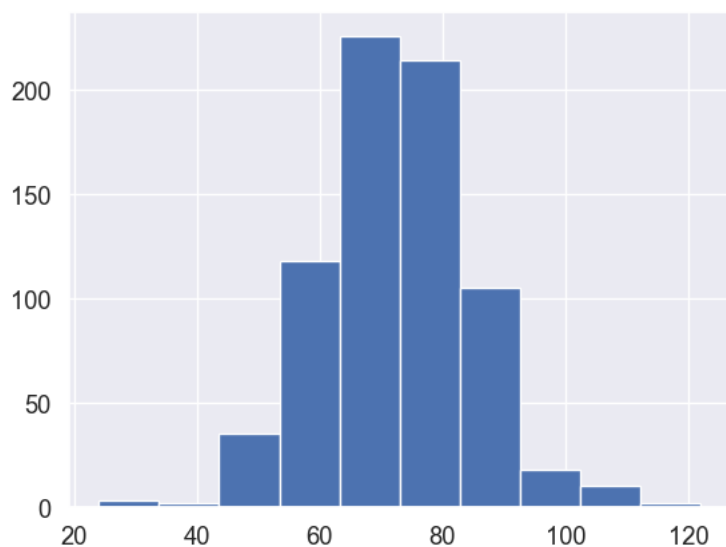
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	763.000000	733.000000	541.000000	394.000000	757.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.348958
std	3.369578	30.535641	12.382158	10.476982	118.775855	6.924988	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	64.000000	22.000000	76.250000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	29.000000	125.000000	32.300000	0.372500	29.000000	0.000000
75%	6.000000	141.000000	80.000000	36.000000	190.000000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

✓ Visually explore these variables using histograms and deal with missing values

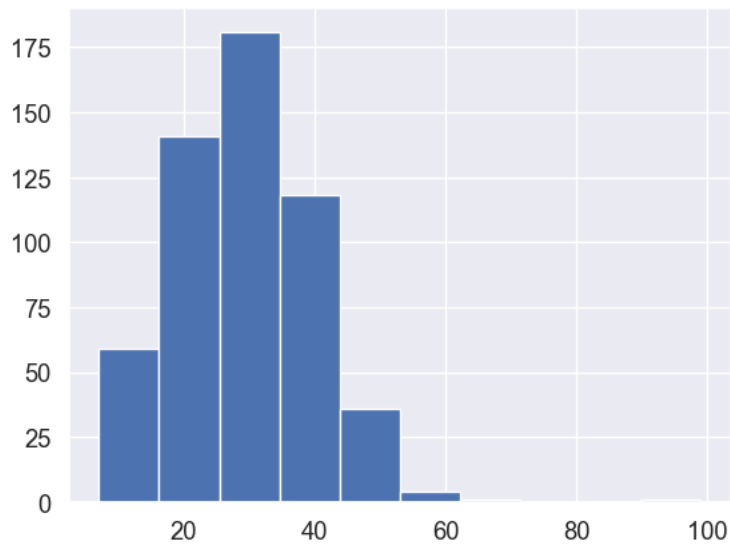
```
df['Glucose'].hist();
```



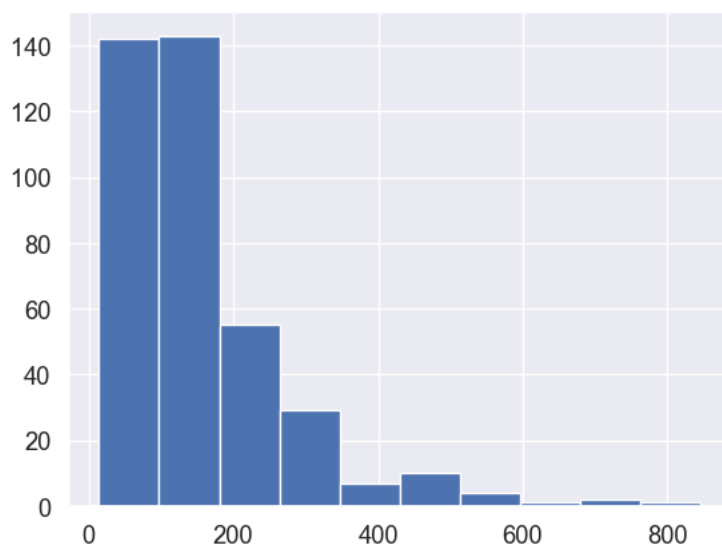
```
df['BloodPressure'].hist();
```



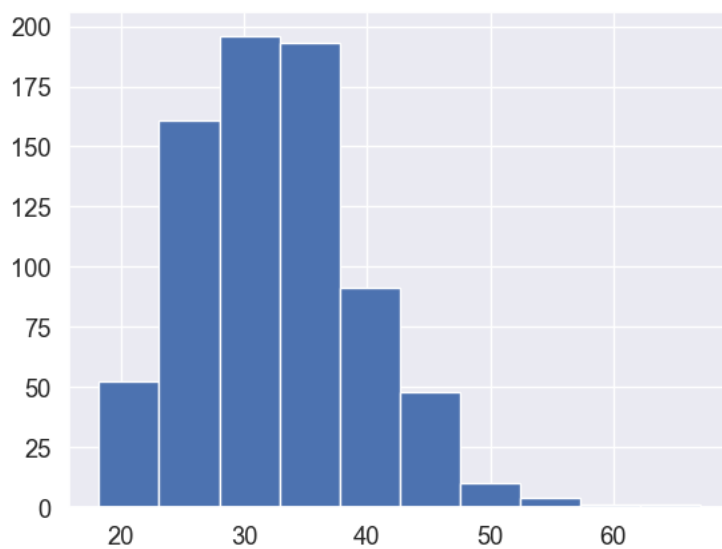
```
df['SkinThickness'].hist();
```



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



```
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 in these 5 variables as below:-

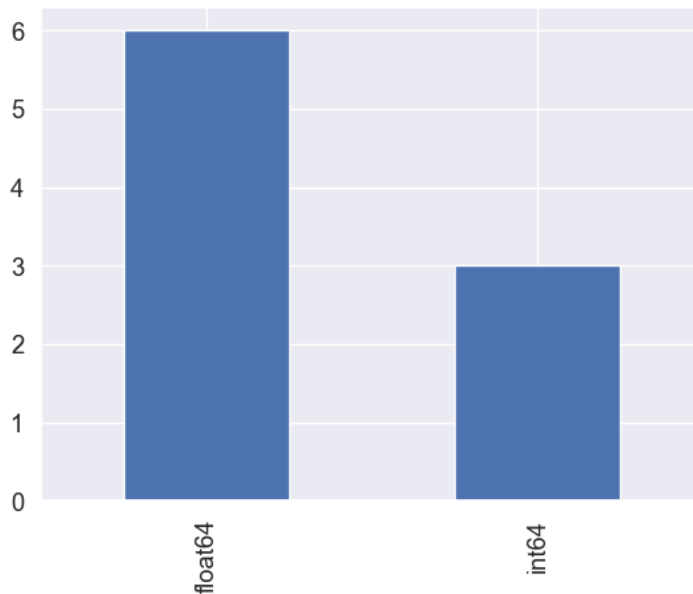
- Glucose - replace missing values with mean of values.
- BloodPressure - replace missing values with mean of values.
- SkinThickness - replace missing values with mean of values.

- Insulin - replace missing values with median of values.
- BMI - replace missing values with mean of values.

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

```
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())
```

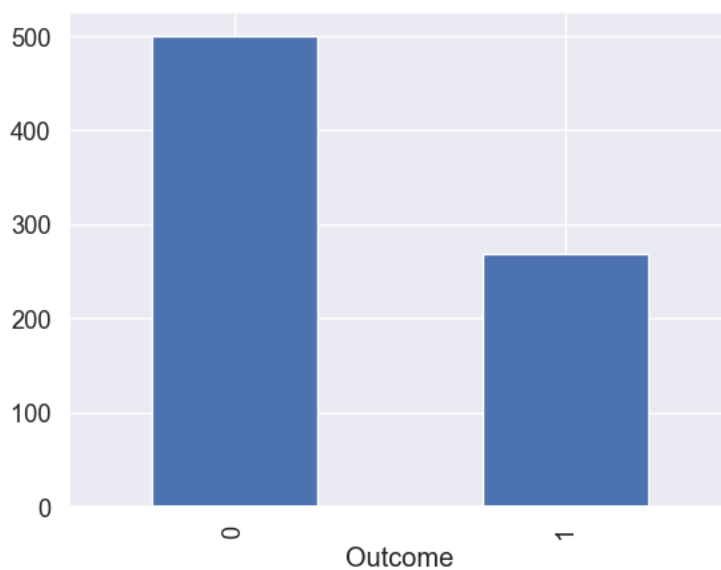
```
df.dtypes.value_counts().plot(kind='bar');
```



▼ Data Exploration

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

```
Outcome  
0    500  
1    268  
Name: count, 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.

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

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

```
import sys
print(sys.version)
```

```
3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
```

```
pip install ipython
```

```
Requirement already satisfied: ipython in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (8.20.0)
Requirement already satisfied: decorator in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipython)
Requirement already satisfied: jedi>=0.16 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipython)
Requirement already satisfied: matplotlib-inline in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipyt
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-pack
Requirement already satisfied: pygments>=2.4.0 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipy
Requirement already satisfied: stack-data in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipython)
Requirement already satisfied: traitlets>=5 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipythc
Requirement already satisfied: exceptiongroup in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipyt
Requirement already satisfied: colorama in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from ipython)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from
Requirement already satisfied: wcwidth in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from prompt-tool
Requirement already satisfied: executing>=1.2.0 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from st
Requirement already satisfied: asttokens>=2.1.0 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from st
Requirement already satisfied: pure-eval in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from stack-dat
Requirement already satisfied: six>=1.12.0 in c:\users\snyai\appdata\local\programs\python\python310\lib\site-packages (from asttoko
Note: you may need to restart the kernel to use updated packages.
```

```
import sys
print(sys.version)
```

```
3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
```

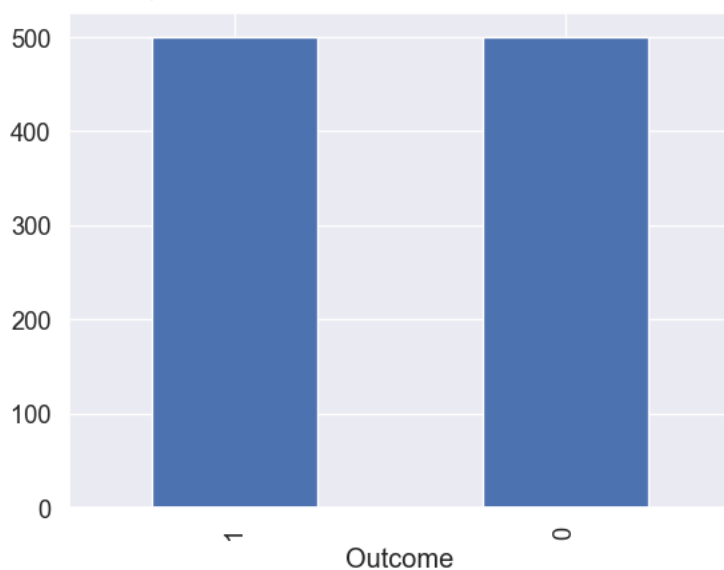
```
from imblearn.over_sampling import SMOTE
```

```
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,)
```

```
df_y_resampled.value_counts().plot(kind='bar')
df_y_resampled.value_counts()
```

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

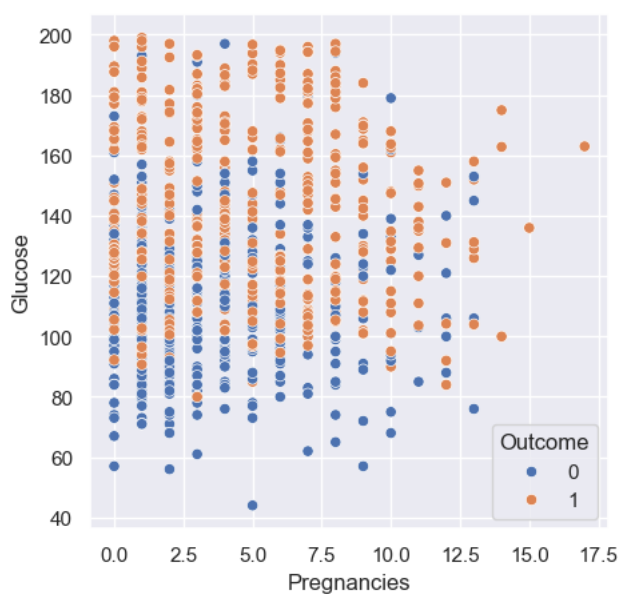


```
df_resampled = pd.concat([df_X_resampled, df_y_resampled], axis=1)
df_resampled
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.000000	72.000000	35.000000	125.000000	33.600000	0.627000	50	1
1	1	85.000000	66.000000	29.000000	125.000000	26.600000	0.351000	31	0
2	8	183.000000	64.000000	29.153420	125.000000	23.300000	0.672000	32	1
3	1	89.000000	66.000000	23.000000	94.000000	28.100000	0.167000	21	0
4	0	137.000000	40.000000	35.000000	168.000000	43.100000	2.288000	33	1
...
995	3	164.686765	74.249021	29.153420	125.000000	42.767110	0.726091	29	1
996	0	138.913540	69.022720	27.713033	127.283849	39.177649	0.703702	24	1
997	10	131.497740	66.331574	33.149837	125.000000	45.820819	0.498032	38	1
998	0	105.571347	83.238205	29.153420	125.000000	27.728596	0.649204	60	1
999	0	127.727025	108.908879	44.468195	129.545366	65.808840	0.308998	26	1

1000 rows × 9 columns

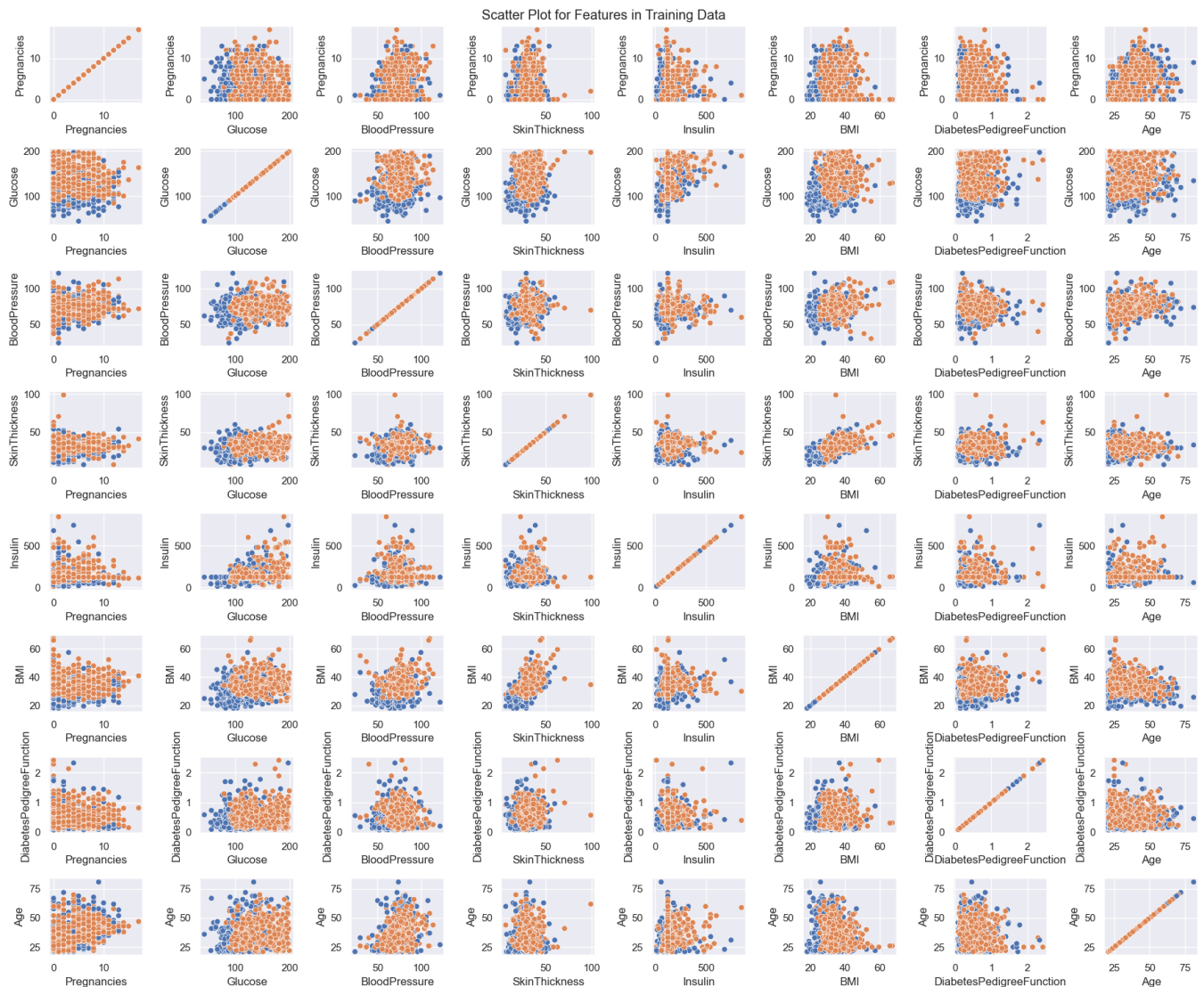
```
sns.set(rc={'figure.figsize':(5,5)})
sns.scatterplot(x="Pregnancies", y="Glucose", data=df_resampled, hue="Outcome");
```



```
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()
```



We have some observations from above scatter plot of pairs of features:

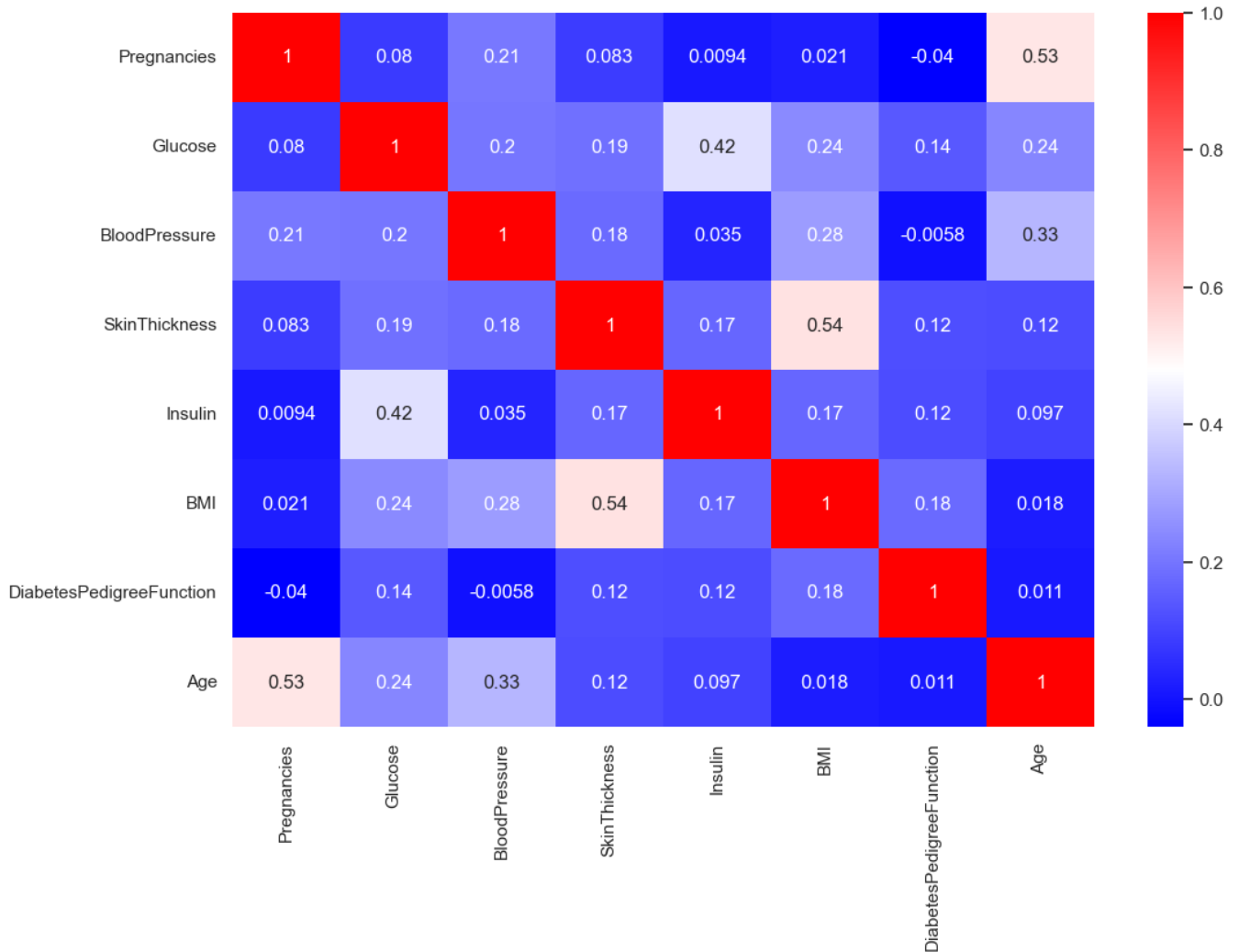
- **Glucose** alone is impressively good to distinguish between the **Outcome** classes.
- **Age** alone is also able to distinguish between classes to some extent.
- It seems none of pairs in the dataset is able to clearly distinguish between the **Outcome** classes.
- We need to use combination of features to build model for prediction of classes in **Outcome**.

✓ Perform correlation analysis using a heat map

```
df_X_resampled.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
Pregnancies	1.000000	0.079953	0.205232	0.082752	0.009365	0.021006	-0.040210	0.532660
Glucose	0.079953	1.000000	0.200717	0.189776	0.418830	0.242501	0.138945	0.235522
BloodPressure	0.205232	0.200717	1.000000	0.176496	0.034861	0.277565	-0.005850	0.332015
SkinThickness	0.082752	0.189776	0.176496	1.000000	0.170719	0.538207	0.120799	0.117644
Insulin	0.009365	0.418830	0.034861	0.170719	1.000000	0.168702	0.115187	0.096940
BMI	0.021006	0.242501	0.277565	0.538207	0.168702	1.000000	0.177915	0.017529
DiabetesPedigreeFunction	-0.040210	0.138945	-0.005850	0.120799	0.115187	0.177915	1.000000	0.010532
Age	0.532660	0.235522	0.332015	0.117644	0.096940	0.017529	0.010532	1.000000

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



It appears from correlation matrix and heatmap that 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.

▼ Data Modeling

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) Decision 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 which will give us insight about model performance on versatile data. It helps to loop through predefined hyperparameters and fit model on training set. GridSearchCV performs hyper parameter tuning 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 Tuning):

```
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,

X_train, X_test, y_train, y_test = train_test_split(df_X_resampled, df_y_resampled, test_size=0.15, random_state =10)

X_train.shape, X_test.shape

((850, 8), (150, 8))
```

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

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

✓ 1) Logistic Regression:

```
from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression(max_iter=300)
```

```
lr1.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=300)
```

```
lr1.score(X_train,y_train)
```

```
0.7294117647058823
```

```
lr1.score(X_test, y_test)
```

```
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 from which GridSearchCV will select the best value of parameter.

```
from sklearn.model_selection import GridSearchCV, cross_val_score
```

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

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

```

GridSearchCV
  estimator: LogisticRegression
    LogisticRegression

```

```
gs_lr.best_params_
```

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

```
gs_lr.best_score_
```

```
0.738
```

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

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

```

LogisticRegression
LogisticRegression(C=13.257113655901108, max_iter=300)

```

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

```
0.731764705882353
```

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

```
0.7733333333333333
```

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

```

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

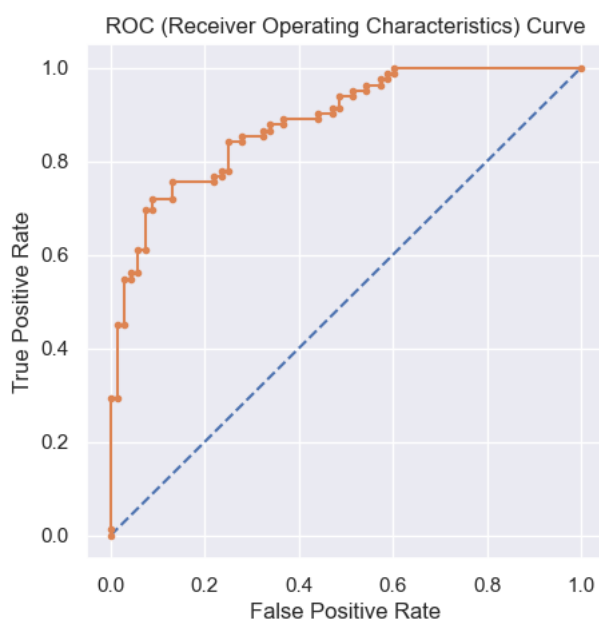
```

```

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");

```

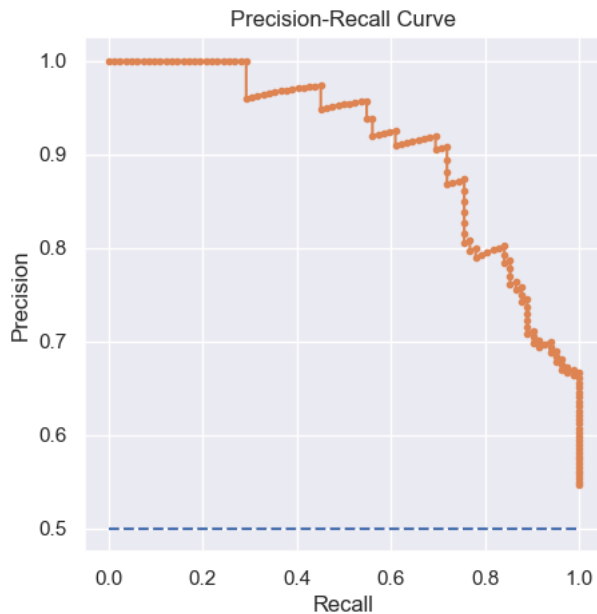
```
AUC: 0.884
```



```
# 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



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

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
dt1 = DecisionTreeClassifier(random_state=0)
```

```
dt1.fit(X_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
dt1.score(X_train,y_train) # Decision Tree always 100% accuracy over train data

1.0
```

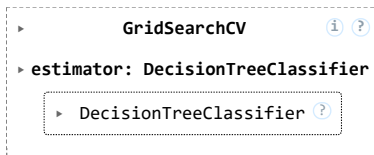
```
dt1.score(X_test, y_test)

0.7733333333333333
```

Performance evaluation and optimizing parameters using GridSearchCV:

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

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



```
gs_dt.best_params_
```

```
{'max_depth': 4}
```

```
gs_dt.best_score_
```

```
0.76
```

```
dt1.feature_importances_
```

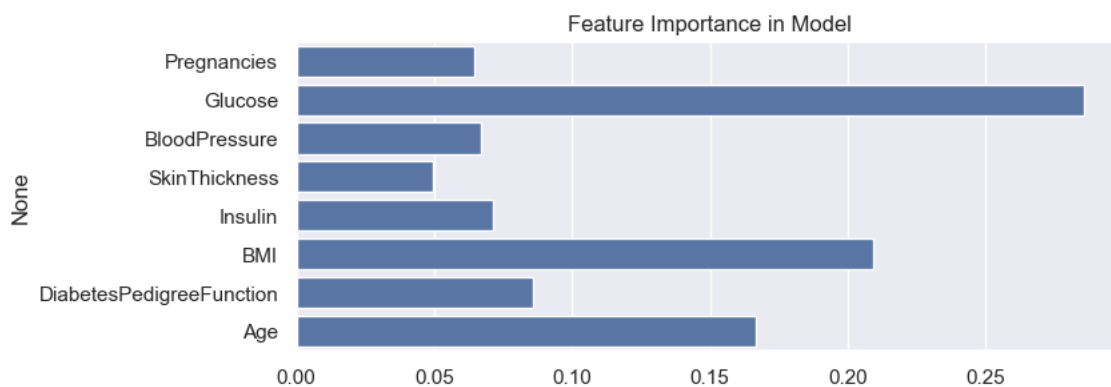
```
array([0.06452226, 0.28556999, 0.06715314, 0.04979714, 0.07150365,
       0.20905992, 0.08573109, 0.16666279])
```

```
X_train.columns
```

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

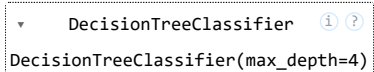
```
import seaborn as sns
import matplotlib.pyplot as plt
```

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



```
dt2 = DecisionTreeClassifier(max_depth=4)
```

```
dt2.fit(X_train,y_train)
```



```
dt2.score(X_train,y_train)
```

```
0.8070588235294117
```

```
dt2.score(X_test, y_test)
```

```
0.82
```

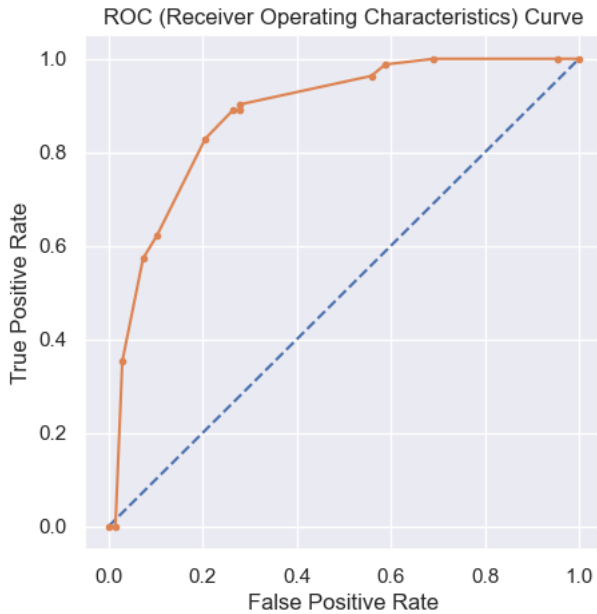
```
# Preparing ROC Curve (Receiver Operating Characteristics Curve)
```

```
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

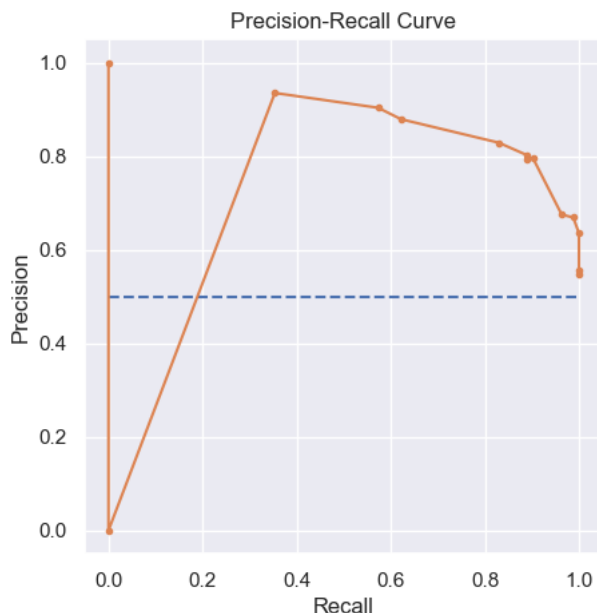


```
# Precision Recall Curve
```

```
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_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=0.844 auc_pr=0.717 ap=0.868



```
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

```
from sklearn.ensemble import RandomForestClassifier
rf1 = RandomForestClassifier()
```

```
rf1 = RandomForestClassifier(random_state=0)
```

```
rf1.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=0)
```

```
rf1.score(X_train, y_train)          # Random Forest also 100% accuracy over train data always

1.0
```

```
rf1.score(X_test, y_test)

0.8466666666666667
```

Performance evaluation and optimizing parameters using GridSearchCV:

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

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

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
gs_dt.best_params_

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

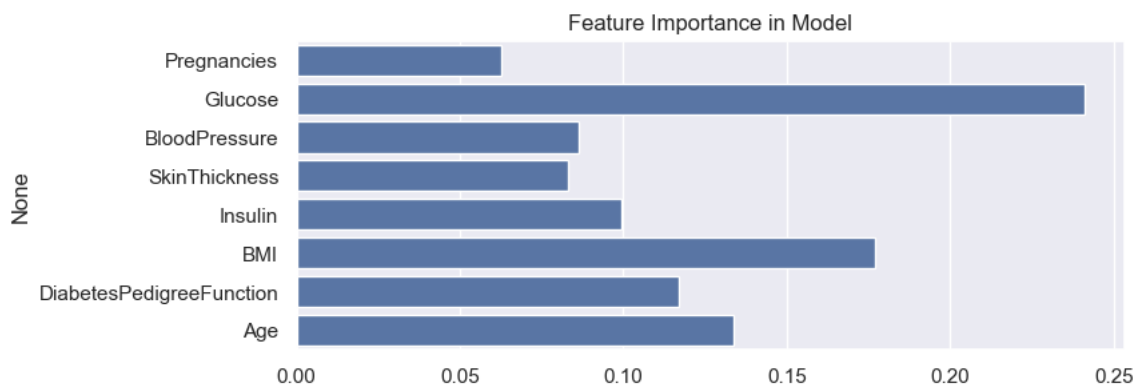
```
gs_dt.best_score_

0.813
```

```
rf1.feature_importances_

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

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



```
rf2 = RandomForestClassifier(max_depth=None, min_samples_leaf=1, n_estimators=100)
```

```
rf2.fit(X_train,y_train)
```

```
RandomForestClassifier
```

```
rf2.score(X_train,y_train)
```

```
1.0
```

```
rf2.score(X_test, y_test)
```

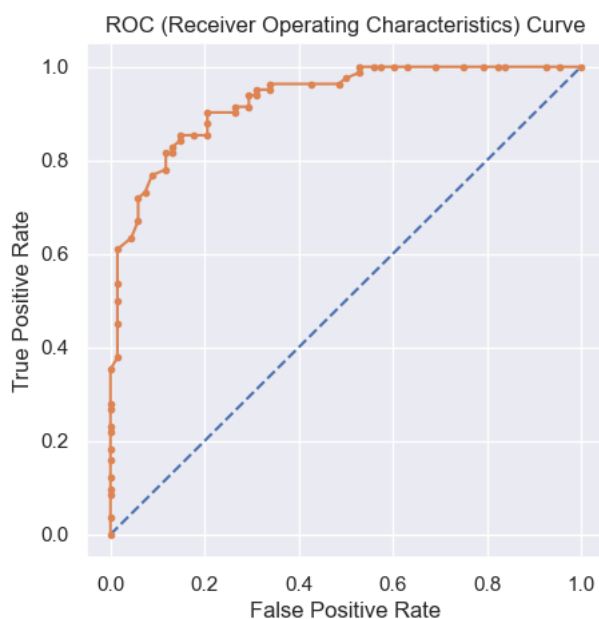
```
0.8466666666666667
```

```
# 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");
```

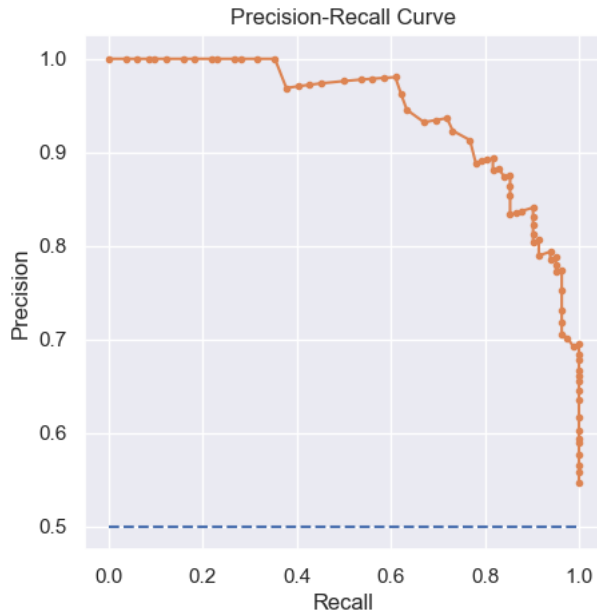
```
AUC: 0.929
```



```
# 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=0.942 ap=0.941
```



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

✓ K-Nearest Neighbour (KNN) Classification

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

```
knn1.fit(X_train, y_train)
```

```
▼ KNeighborsClassifier ⓘ ?
KNeighborsClassifier(n_neighbors=3)
```

```
knn1.score(X_train, y_train)
```

```
0.8835294117647059
```

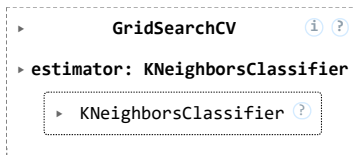
```
knn1.score(X_test, y_test)
```

```
0.7866666666666666
```

Performance evaluation and optimizing parameters using GridSearchCV:

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

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

```
gs_knn.best_params_
```

```
{'n_neighbors': 3}
```

```
gs_knn.best_score_
```

```
0.771
```

```
# gs_knn.cv_results_
```

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

```
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])
```

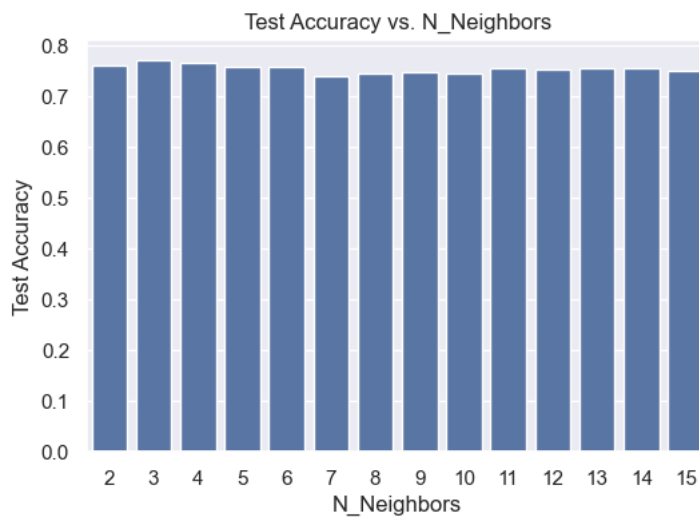
```
plt.figure(figsize=(6,4))
```

```
sns.barplot(x=knn_neighbors, y=gs_knn.cv_results_['mean_test_score'])
```

```
plt.xlabel("N_Neighbors")
```

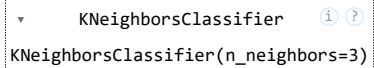
```
plt.ylabel("Test Accuracy")
```

```
plt.title("Test Accuracy vs. N_Neighbors");
```



```
knn2 = KNeighborsClassifier(n_neighbors=3)
```

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



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

```
0.8835294117647059
```

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

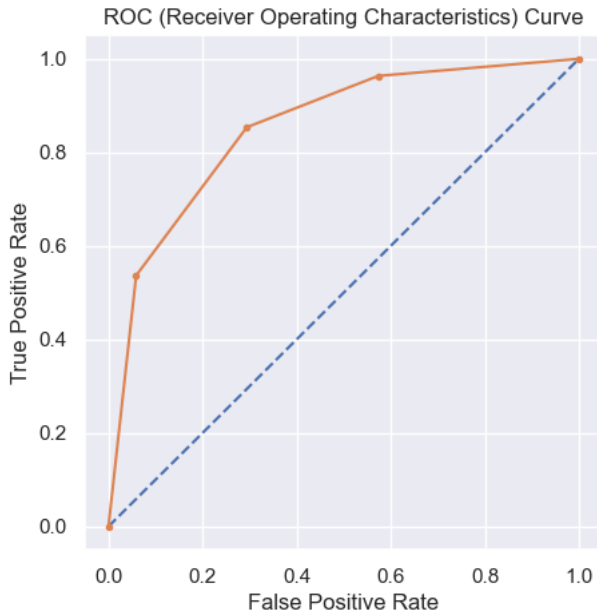
```
0.7866666666666666
```

```
# 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");
```

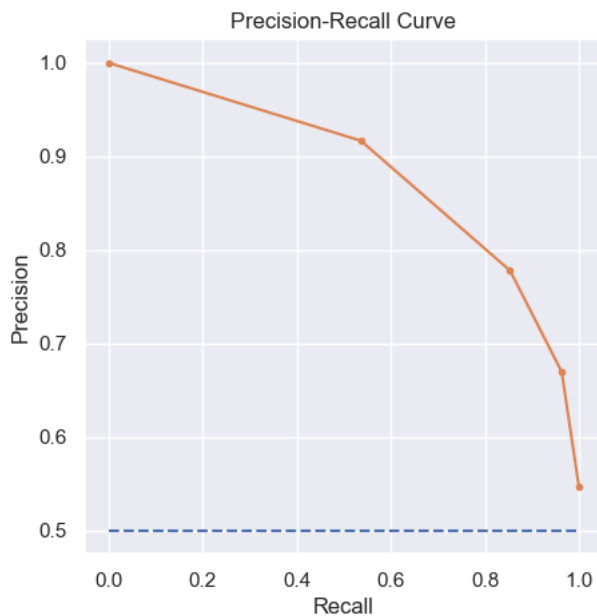
AUC: 0.852



```
# 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



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

✓ Support Vector Machine (SVM) Algorithm

```
from sklearn.svm import SVC
svm1 = SVC(kernel='rbf')
```

```
svm1.fit(X_train, y_train)
```

▼ SVC ⓘ ?
SVC()

```
svm1.score(X_train, y_train)
```

```
0.7282352941176471
```

```
svm1.score(X_test, y_test)
```

```
0.78
```

Performance evaluation and optimizing parameters using GridSearchCV:

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

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

► GridSearchCV ⓘ ?
 ► estimator: SVC
 ► SVC ?

```
gs_svm.best_params_
```

```
{'C': 20, 'gamma': 0.005}
```

```
gs_svm.best_score_
```

```
0.8089999999999999
```

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

```
svm2.fit(X_train, y_train)
```

▼ SVC ⓘ ?
SVC(C=20, gamma=0.005, probability=True)

```
svm2.score(X_train, y_train)
```

```
0.9941176470588236
```

```
svm2.score(X_test, y_test)
```

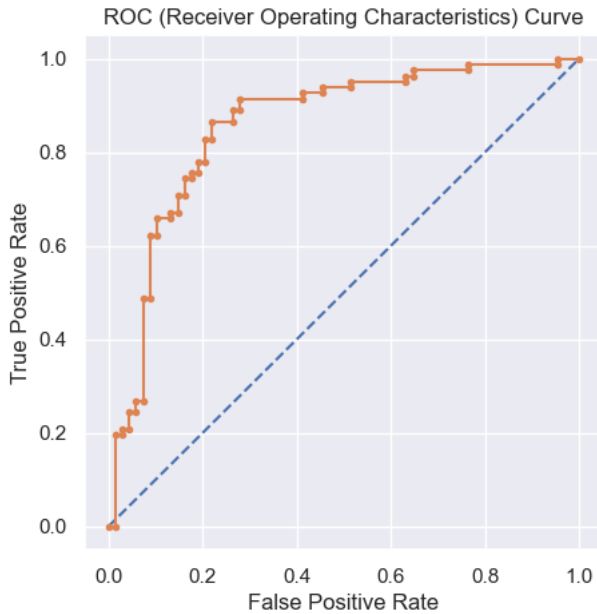
```
0.8133333333333334
```

```
# 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");
```

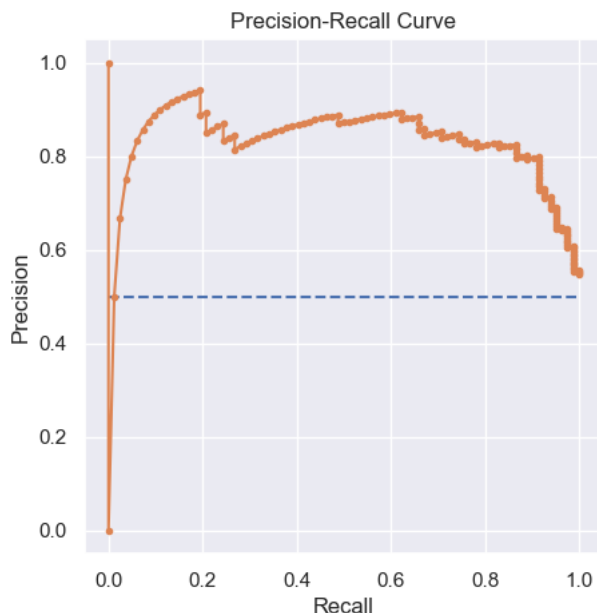
AUC: 0.857



```
# 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=0.830 ap=0.837



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

✓ Naive Bayes Algorithm

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

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

```
▼ GaussianNB ⓘ (?)
GaussianNB()
```

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

```
0.7294117647058823
```

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

```
0.8
```

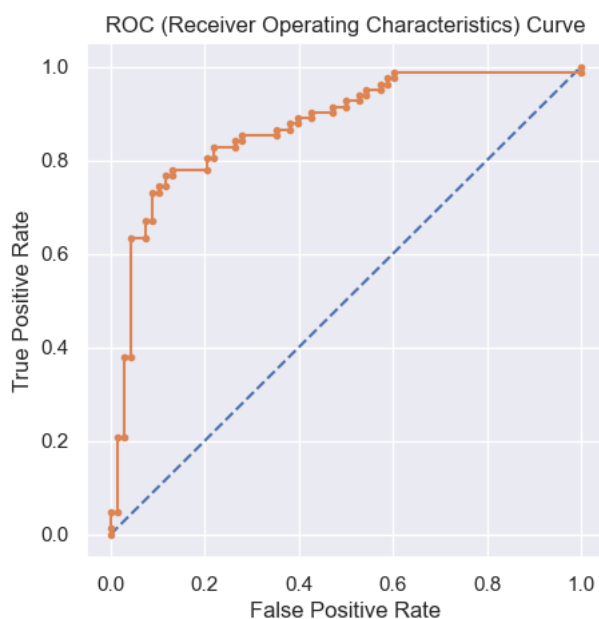
Naive Bayes has almost no hyperparameters to tune, so it usually generalizes well.

```
# 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");
```

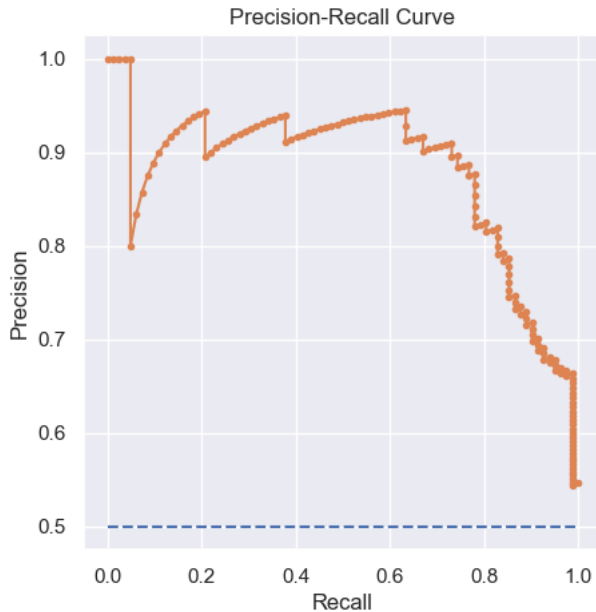
```
AUC: 0.873
```



```
# 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
```



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

✓ Ensemble Learning --> Boosting --> Adaptive Boosting

```
from sklearn.ensemble import AdaBoostClassifier
ada1 = AdaBoostClassifier(n_estimators=100)
```

```
ada1.fit(X_train,y_train)
```

```
C:\Users\sna\i\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The
warnings.warn()
```

```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=100)
```

```
ada1.score(X_train,y_train)
```

```
0.8564705882352941
```

```
ada1.score(X_test, y_test)
```

```
0.7666666666666667
```

Performance evaluation and optimizing parameters using cross_val_score:

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

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

```

GridSearchCV
  estimator: AdaBoostClassifier
    AdaBoostClassifier

```

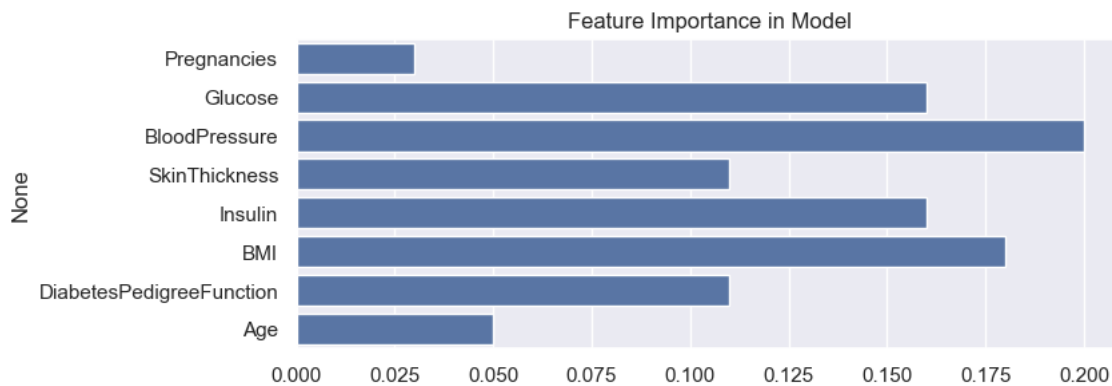
```
gs_ada.best_params_  
    {'n_estimators': 500}  
gs_ada.best_score
```

0.785

ada1.feature_importances_

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

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



ada2 = AdaBoostClassifier(n_estimators=500)

ada2.fit(X_train,y_train)

```
C:\Users\snayi\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: Th
```

```
warnings.warn(
AdaBoostClassifier
AdaBoostClassifier(n_estimators=500)
```

ada2.score(X_train,y_train)

0.9247058823529412

ada2.score(X_test, y_test)

0.7733333333333333

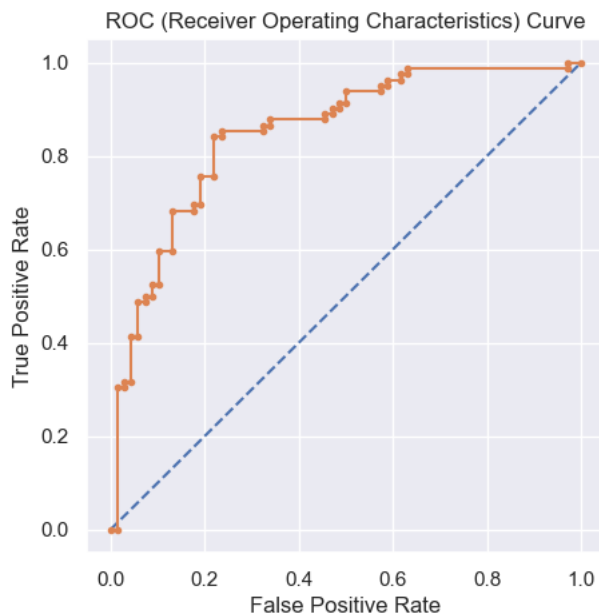
Preparing ROC Curve (Receiver Operating Characteristics Curve)

```
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



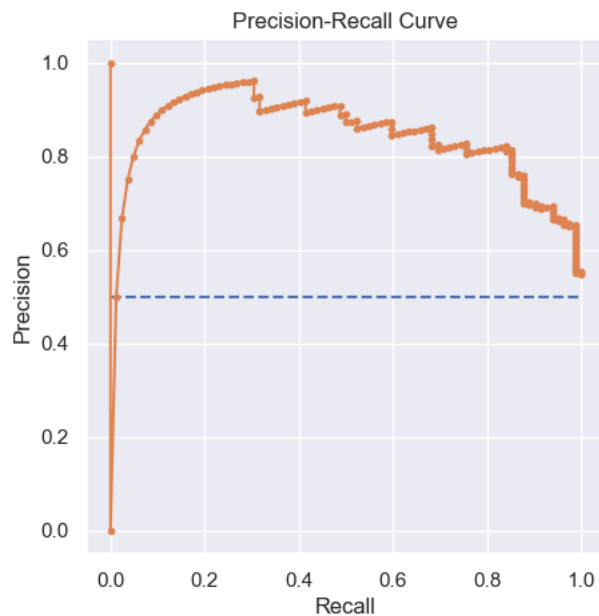
Precision Recall Curve

```

pred_y_test = ada2.predict(X_test)
precision, recall, thresholds = precision_recall_curve(y_test, probs) # predict class values
f1 = f1_score(y_test, pred_y_test) # calculate precision-recall curve
auc_ada_pr = auc(recall, precision) # calculate F1 score
ap = average_precision_score(y_test, probs) # calculate precision-recall AUC
print('f1=%.3f auc_pr=%.3f ap=%.3f' % (f1, auc_ada_pr, ap)) # calculate average precision score
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



```

models.append('ADA')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_ada)

```

✓ Ensemble Learning --> Boosting --> Gradient Boosting (XGBClassifier)

```

from xgboost import XGBClassifier
xgb1 = XGBClassifier(use_label_encoder=False, objective = 'binary:logistic', nthread=4, seed=10)

```

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

```

XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               multi_strategy=None, n_estimators=None, n_jobs=None, nthread=4,
               num_parallel_tree=None, ...)

```

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

```
1.0
```

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

```
print(acc*100)
```

```
80.66666666666666
```

Performance evaluation and optimizing parameters using GridSearchCV:

```

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

```

```

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)

```

```

GridSearchCV
  estimator: XGBClassifier
    XGBClassifier

```

```
gs_xgb.best_params_
```

```
{'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 180}
```

```
gs_xgb.best_score_
```

```
0.8826799999999999
```

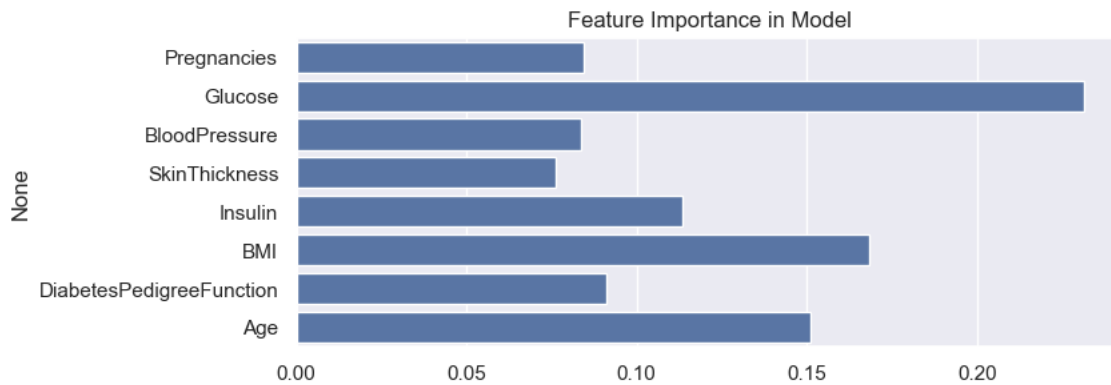
```
xgb1.feature_importances_
```

```
array([0.08452258, 0.23145914, 0.08379252, 0.07622254, 0.11353929,
       0.16838206, 0.09102635, 0.15105554], dtype=float32)
```

```

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

```



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

```
xgb2.fit(X_train,y_train)
```

```

XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,
               gamma=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=0.05, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=7, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               multi_strategy=None, n_estimators=180, n_jobs=None, nthread=4,
               num_parallel_tree=None, ...)

```

```
xgb2.score(X_train,y_train)
```

```
0.9929411764705882
```

```
xgb2.score(X_test, y_test)
```

```
0.82
```

```
# 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: 0.916

ROC (Receiver Operating Characteristics) Curve



Precision Recall Curve

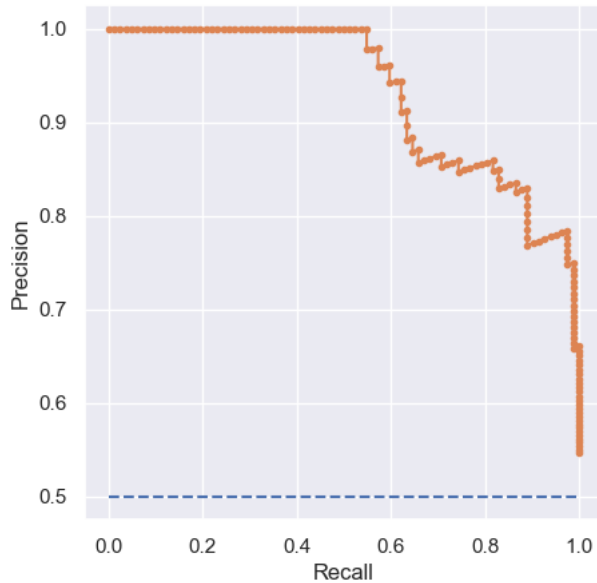
```

pred_y_test = xgb2.predict(X_test)
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=0.836 auc_pr=0.932 ap=0.932

Precision-Recall Curve



```

models.append('XGB')
model_accuracy.append(accuracy_score(y_test, pred_y_test))
model_f1.append(f1)
model_auc.append(auc_xgb)

```

```

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')

```

```

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

```

Comparison of Different Classification Algorithms

