Nama: Raina Imtiyaz

NIM: 2502010976

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time

sns.set()
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier, AdaBoostClassifier, RandomForestClassifier, BaggingClassifier


from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline


df = pd.read_csv("diabetes.csv", sep=',')
```

1. EDA

```
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

```
df.tail()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

```
#iInformation about the dataset
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                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
```

```
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
#descriptive statistic
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

```
#check null values
df.isnull().sum()
```

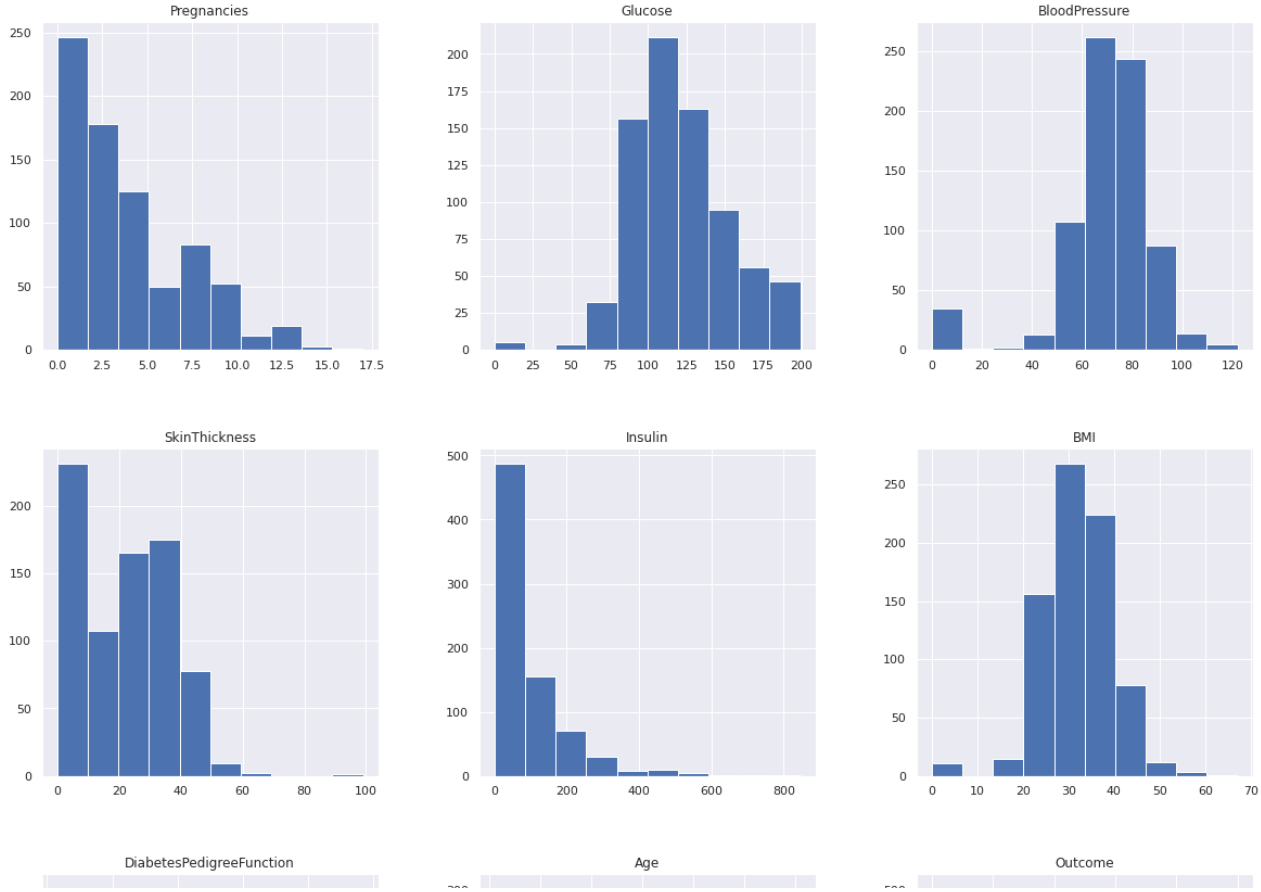
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

```
df_copy = df.copy(deep = True)
df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI'
```

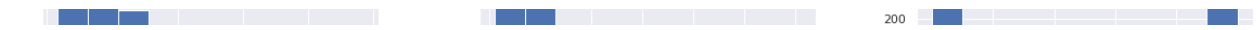
```
# Showing the Count of NANs
print(df_copy.isnull().sum())
```

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

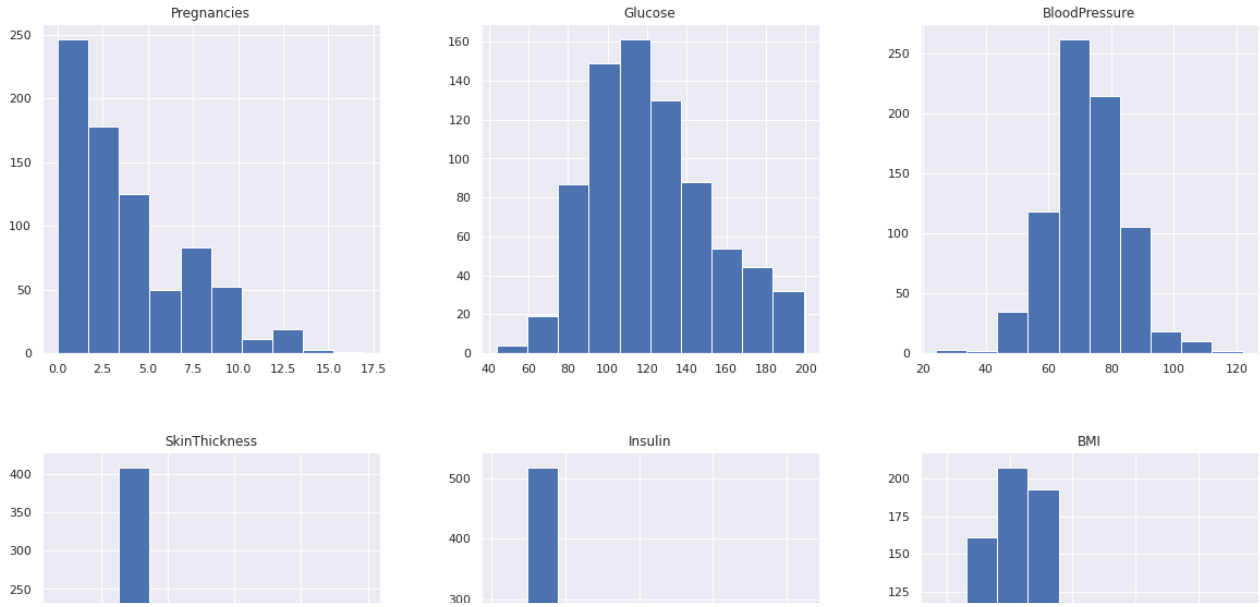
```
# Plotting the data distribution plots before removing null values
p = df.hist(figsize = (20,20))
```




```
# Now we will be imputing the mean value of the column to each missing value of that particular column
df_copy['Glucose'].fillna(df_copy['Glucose'].mean(), inplace = True)
df_copy['BloodPressure'].fillna(df_copy['BloodPressure'].mean(), inplace = True)
df_copy['SkinThickness'].fillna(df_copy['SkinThickness'].median(), inplace = True)
df_copy['Insulin'].fillna(df_copy['Insulin'].median(), inplace = True)
df_copy['BMI'].fillna(df_copy['BMI'].median(), inplace = True)
```

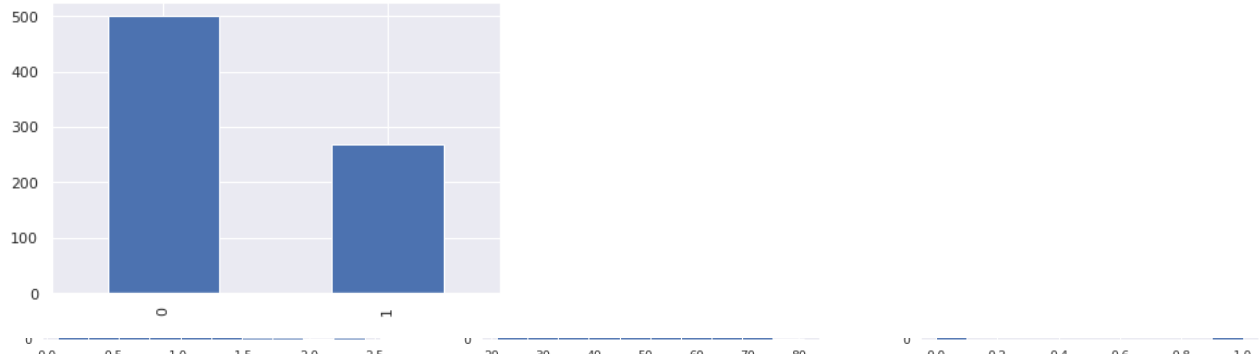


```
# Plotting the distributions after removing the NAN values
p = df_copy.hist(figsize = (20,20))
```

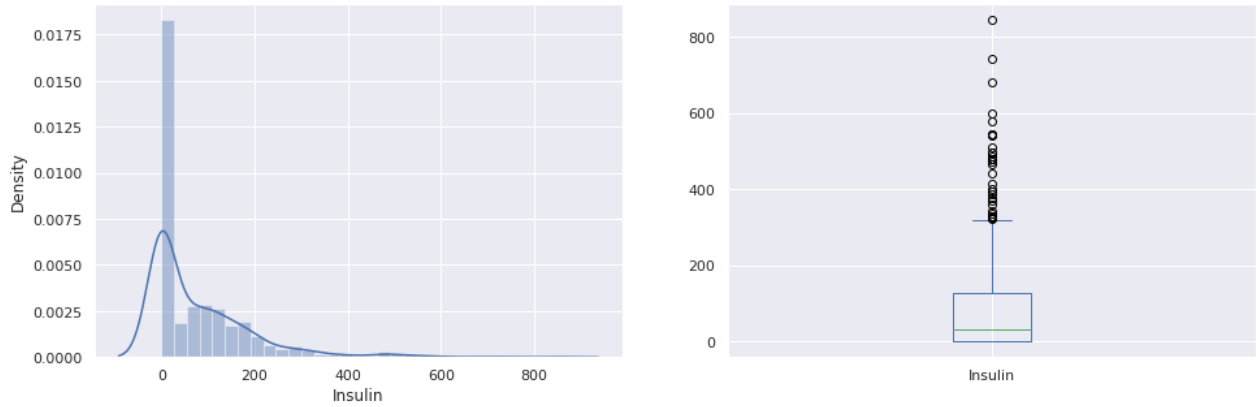



```
# Now, let's check that how well our outcome column is balanced
color_wheel = {1: "#0392cf", 2: "#7bc043"}
colors = df["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(df.Outcome.value_counts())
p= df.Outcome.value_counts().plot(kind="bar")
```

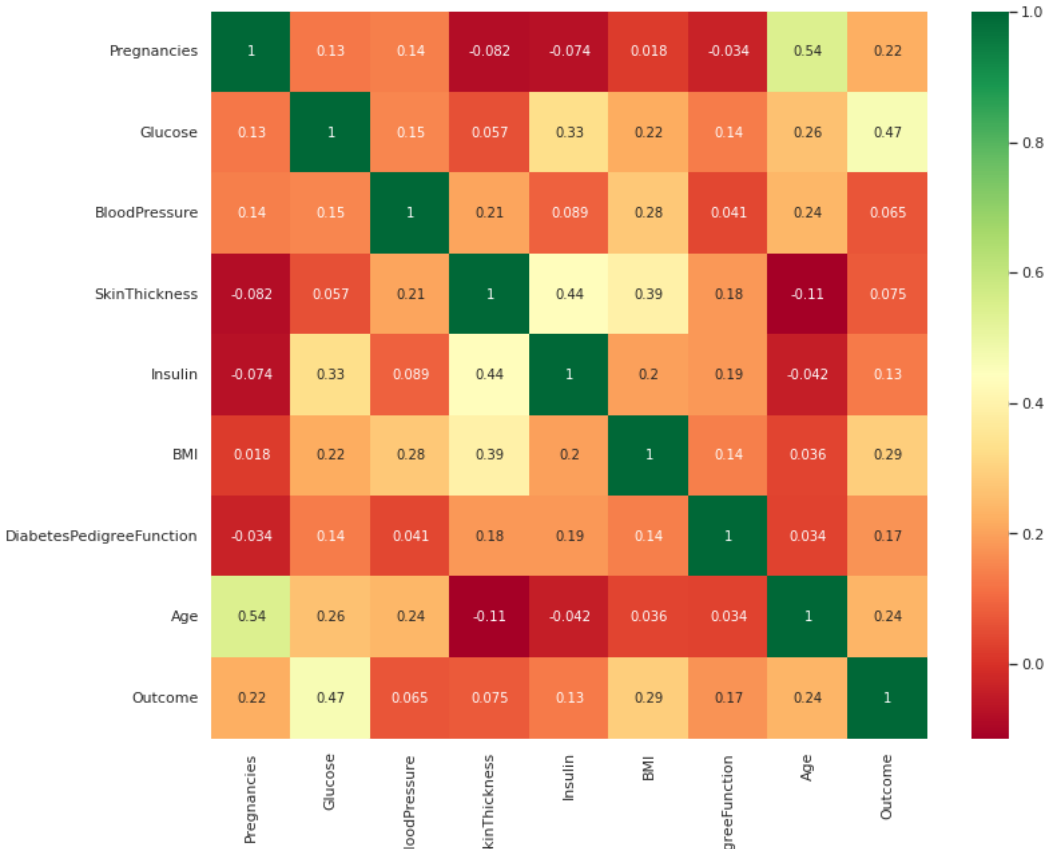
```
0    500
1    268
Name: Outcome, dtype: int64
```



```
plt.subplot(121), sns.distplot(df['Insulin'])
plt.subplot(122), df['Insulin'].plot.box(figsize=(16,5))
plt.show()
```




```
#Correlation
plt.figure(figsize=(12,10))
p = sns.heatmap(df.corr(), annot=True,cmap ='RdYlGn')  # seaborn has an easy method to showcase heatmap
```




```
df_copy.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	125.0	33.6	0.627	50	1
1	1	85.0	66.0	29.0	125.0	26.6	0.351	31	0
2	8	183.0	64.0	29.0	125.0	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

```
sc_X = StandardScaler() #scaling hanya untuk dependant variable
X =  pd.DataFrame(sc_X.fit_transform(df_copy.drop(["Outcome"],axis = 1),), columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThick
X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639947	0.865108	-0.033518	0.670643	-0.181541	0.166619	0.468492	1.425995
1	-0.844885	-1.206162	-0.529859	-0.012301	-0.181541	-0.852200	-0.365061	-0.190672
2	1.233880	2.015813	-0.695306	-0.012301	-0.181541	-1.332500	0.604397	-0.105584
3	-0.844885	-1.074652	-0.529859	-0.695245	-0.540642	-0.633881	-0.920763	-1.041549
4	-1.141852	0.503458	-2.680669	0.670643	0.316566	1.549303	5.484909	-0.020496

```
y = df_copy.Outcome
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
#Splitting the dataset
```

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
```

```
#SVC Classifier

svc_model = SVC()
svc_model.fit(X_train, y_train)
```

```
SVC()
```

```
svc_pred = svc_model.predict(X_test)
print(confusion_matrix(y_test, svc_pred))
print(classification_report(y_test, svc_pred))
```

```
[[87 12]
 [24 31]]
              precision    recall  f1-score   support

           0       0.78      0.88      0.83        99
           1       0.72      0.56      0.63        55

    accuracy                           0.77       154
   macro avg       0.75      0.72      0.73       154
weighted avg       0.76      0.77      0.76       154
```

```
print("Accuracy Score =", format(metrics.accuracy_score(y_test, svc_pred)))
```

```
Accuracy Score = 0.7662337662337663
```

```
#decision tree
dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
predictions = dtree.predict(X_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
```

```
[[80 19]
 [16 39]]
              precision    recall  f1-score   support

           0       0.83      0.81      0.82        99
           1       0.67      0.71      0.69        55

    accuracy                           0.77       154
   macro avg       0.75      0.76      0.76       154
weighted avg       0.78      0.77      0.77       154
```

Grid Search

```
param_grid = {'C' : [0.1, 1], 'gamma':[1, 0.1], 'kernel':['rbf']}
```

```
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=1)
```

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

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1], 'gamma': [1, 0.1], 'kernel': ['rbf']},
             verbose=1)
```

```
grid.best_params_
```

```
{'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

```
grid.best_estimator_
```

```
SVC(C=0.1, gamma=1)
```

```
grid_pred = grid.predict(X_test)
```

```
print(confusion_matrix(y_test, grid_pred))
print(classification_report(y_test, grid_pred))
```

```
print(classification_report(y_test, grid_pred))
```

```
[[99  0]
 [55  0]]
              precision    recall  f1-score   support

           0       0.64      1.00      0.78        99
           1       0.00      0.00      0.00        55

    accuracy                           0.64       154
   macro avg       0.32      0.50      0.39       154
weighted avg       0.41      0.64      0.50       154
```

Model dari Best Learner memiliki akurasi 0.64 (64%). Precision yang menunjukkan ketepatan pengklasifikasian menunjukkan bahwa ketepatan pengklasifikasian orang yang tidak terkena diabetes mencapai 64%. Recall yang menunjukkan kelengkapan pengklasifikasian juga menujukkan bahwa kelengkapan dari klasifikasi orang yang tidak terkena diabetes lebih tinggi daripada yang terkena diabetes, yaitu 100%.

KNN

```
knn = KNeighborsClassifier()
knnModel = knn.fit(X_train, y_train)
knnPred = knn.predict(X_test)
print(classification_report(y_test, knnPred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.71      0.73        99
           1       0.52      0.58      0.55        55

    accuracy                           0.66       154
   macro avg       0.64      0.64      0.64       154
weighted avg       0.67      0.66      0.67       154
```

Model dari KNN memiliki akurasi 0.66 (66%). Precision yang menunjukkan ketepatan pengklasifikasian menunjukkan bahwa ketepatan pengklasifikasian orang yang tidak terkena diabetes mencapai 75%. Recall yang menunjukkan kelengkapan pengklasifikasian juga menujukkan bahwa kelengkapan dari klasifikasi orang yang tidak terkena diabetes lebih tinggi daripada yang terkena diabetes, yaitu 71%.

Voting Classifier

```
estimators=[('svm', svc_model),('KNN', knn),('DecisionTree', dtree)]
ensemble = VotingClassifier(estimators, voting='hard')

ensembleModel = ensemble.fit(X_train, y_train)
ensemblePred = ensemble.predict(X_test)
print(classification_report(y_test, ensemblePred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.79        99
           1       0.62      0.64      0.63        55

    accuracy                           0.73       154
   macro avg       0.71      0.71      0.71       154
weighted avg       0.73      0.73      0.73       154
```

Model dari Voting Classifier memiliki akurasi 0.73 (73%). Precision yang menunjukkan ketepatan pengklasifikasian menunjukkan bahwa ketepatan pengklasifikasian orang yang tidak terkena diabetes mencapai 80%. Recall yang menunjukkan kelengkapan pengklasifikasian juga menujukkan bahwa kelengkapan dari klasifikasi orang yang tidak terkena diabetes lebih tinggi daripada yang terkena diabetes, yaitu 79%.

Bagging Classifier (Random Forest)

```
param_dist = {'max_depth' : [2, 3, 4],
              'bootstrap' : [True, False],
              'max_features' : ['auto', 'sqrt', 'log2', None],
              'criterion' : ['gini', 'entropy']}
```

```
rfPred = cv_rf.predict(X_test)
print(classification_report(y_test, rfPred))
```

```
                  precision    recall  f1-score   support

               0       0.80      0.88      0.84        99
               1       0.73      0.60      0.66        55

        accuracy                           0.78       154
       macro avg       0.77      0.74      0.75       154
    weighted avg       0.78      0.78      0.77       154
```

Model dari Bagging Classifier memiliki akurasi 0.78 (78%). Precision yang menunjukkan ketepatan pengklasifikasian menunjukkan bahwa ketepatan pengklasifikasian orang yang tidak terkena diabetes mencapai 80%. Recall yang menunjukkan kelengkapan pengklasifikasian juga menujukkan bahwa kelengkapan dari klasifikasi orang yang tidak terkena diabetes lebih tinggi daripada yang terkena diabetes, yaitu 88%.

Boosting Classifier

```
adaBoost = AdaBoostClassifier()
abModel = adaBoost.fit(X_train, y_train)
abPred = adaBoost.predict(X_test)
print(classification_report(y_test, abPred))
```

```
                  precision    recall  f1-score   support

               0       0.80      0.79      0.79        99
               1       0.62      0.64      0.63        55

        accuracy                           0.73       154
       macro avg       0.71      0.71      0.71       154
    weighted avg       0.73      0.73      0.73       154
```

Model dari Boosting Classifier memiliki akurasi 0.73 (73%). Precision yang menunjukkan ketepatan pengklasifikasian menunjukkan bahwa ketepatan pengklasifikasian orang yang tidak terkena diabetes mencapai 80%. Recall yang menunjukkan kelengkapan pengklasifikasian juga menujukkan bahwa kelengkapan dari klasifikasi orang yang tidak terkena diabetes lebih tinggi daripada yang terkena diabetes, yaitu 79%.

CONCLUSION

Berdasarkan akurasinya, di antara model dari best learner, voting classifier, bagging classifier, dan boosting classifier, model terbaik adalah model bagging classifier dengan accuracy 78%.