For columns:

Glucose,
BloodPressure,
BMI,
DiabetesPedigreeFunction

If the column value is 0, then they should be considered as **missing data.**

1. Firstly, replace all Missing values with relevant figures.

```
import numpy as np
import pandas as pd
df = pd.read_csv('Dataset_Day7.csv')
print(df.info())
missing_value_percent = df.isna().sum() / len(df)
* 100
print(missing_value_percent)
skewness = df.skew()
print(skewness)
df["Glucose"].fillna(df["Glucose"].median(),
inplace=True)
df["BloodPressure"].fillna(df["BloodPressure"].me
an(), inplace=True)
df["BMI"].fillna(df["BMI"].median(),
inplace=True)
df["Outcome"].fillna(df["Outcome"].mean(),
inplace=True)
print(df.info())
```

File - Day11Q1

```
C:\Users\tejas\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\
tejas\PycharmProjects\pythonProject\START\Day11Q1.py
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 7 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   BMI                       768 non-null    float64
 4   DiabetesPedigreeFunction  768 non-null    float64
 5   Age                       768 non-null    int64
 6   Outcome                   768 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 42.1 KB
None
Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64
Pregnancies                 0.901674
Glucose                     0.173754
BloodPressure              -1.843608
BMI                        -0.428982
DiabetesPedigreeFunction    1.919911
Age                         1.129597
Outcome                     0.635017
dtype: float64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 7 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   BMI                       768 non-null    float64
 4   DiabetesPedigreeFunction  768 non-null    float64
 5   Age                       768 non-null    int64
 6   Outcome                   768 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 42.1 KB
None

Process finished with exit code 0

```

2. Then remove all existing outliers and get the final data for classification.

```
import numpy as np
import pandas as pd
df = pd.read_csv('Dataset_Day7.csv')
print(df.info())
missing_value_percent = df.isna().sum() / len(df)
```

```
* 100
print(missing_value_percent)
skewness = df.skew()
print(skewness)
df["Glucose"].fillna(df["Glucose"].median(),
inplace=True)
df["BloodPressure"].fillna(df["BloodPressure"].me
an(), inplace=True)
df["BMI"].fillna(df["BMI"].median(),
inplace=True)
df["Outcome"].fillna(df["Outcome"].mean(),
inplace=True)
print(df.info())
OutlierData = pd.DataFrame()
temp = df[["Pregnancies", "Glucose",
"BloodPressure","BMI","DiabetesPedigreeFunction"]
]
for col in ["Pregnancies", "Glucose",
"BloodPressure","BMI","DiabetesPedigreeFunction"]
:
    Q1 = temp[col].quantile(0.25)  # Gives 25th
Percentile or Q1
    Q3 = temp[col].quantile(0.75)  # Gives 75th
Percentile or Q3

    IQR = Q3 - Q1

    UpperBound = Q3 + 1.5 * IQR
    LowerBound = Q1 - 1.5 * IQR

    OutlierData[col] = temp[col][(temp[col] <
LowerBound) | (temp[col] > UpperBound)]
    print(len(OutlierData))
    df_OutlierFree = df.drop(OutlierData.index,
axis=0)
    df_OutlierFree.info()
```

File - Day11Q2

```
C:\Users\tejas\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\
tejas\PycharmProjects\pythonProject\START\Day11Q2.py
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 7 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   BMI                       768 non-null    float64
 4   DiabetesPedigreeFunction  768 non-null    float64
 5   Age                       768 non-null    int64
 6   Outcome                   768 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 42.1 KB
None
Pregnancies                 0.0
Glucose                     0.0
BloodPressure               0.0
BMI                         0.0
DiabetesPedigreeFunction    0.0
Age                         0.0
Outcome                     0.0
dtype: float64
Pregnancies                 0.901674
Glucose                     0.173754
BloodPressure              -1.843608
BMI                        -0.428982
DiabetesPedigreeFunction    1.919911
Age                         1.129597
Outcome                     0.635017
dtype: float64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 7 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   BMI                       768 non-null    float64
 4   DiabetesPedigreeFunction  768 non-null    float64
 5   Age                       768 non-null    int64
 6   Outcome                   768 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 42.1 KB
None
4
<class 'pandas.core.frame.DataFrame'>
Index: 764 entries, 0 to 767
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               764 non-null    int64
 1   Glucose                   764 non-null    int64
 2   BloodPressure             764 non-null    int64
 3   BMI                       764 non-null    float64
 4   DiabetesPedigreeFunction  764 non-null    float64
```

```
 5   Age                       764 non-null    int64
 6   Outcome                   764 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 47.8 KB
4
<class 'pandas.core.frame.DataFrame'>
Index: 764 entries, 0 to 767
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               764 non-null    int64
 1   Glucose                   764 non-null    int64
 2   BloodPressure             764 non-null    int64
 3   BMI                       764 non-null    float64
 4   DiabetesPedigreeFunction  764 non-null    float64
 5   Age                       764 non-null    int64
 6   Outcome                   764 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 47.8 KB
4
<class 'pandas.core.frame.DataFrame'>
Index: 764 entries, 0 to 767
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               764 non-null    int64
 1   Glucose                   764 non-null    int64
 2   BloodPressure             764 non-null    int64
 3   BMI                       764 non-null    float64
 4   DiabetesPedigreeFunction  764 non-null    float64
 5   Age                       764 non-null    int64
 6   Outcome                   764 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 47.8 KB
4
<class 'pandas.core.frame.DataFrame'>
Index: 764 entries, 0 to 767
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               764 non-null    int64
 1   Glucose                   764 non-null    int64
 2   BloodPressure             764 non-null    int64
 3   BMI                       764 non-null    float64
 4   DiabetesPedigreeFunction  764 non-null    float64
 5   Age                       764 non-null    int64
 6   Outcome                   764 non-null    int64
dtypes: float64(2), int64(5)
memory usage: 47.8 KB
4
<class 'pandas.core.frame.DataFrame'>
Index: 764 entries, 0 to 767
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               764 non-null    int64
 1   Glucose                   764 non-null    int64
 2   BloodPressure             764 non-null    int64
 3   BMI                       764 non-null    float64
```

File - Day11Q2

```
118  4   DiabetesPedigreeFunction  764 non-null    float64
119  5   Age                       764 non-null    int64
120  6   Outcome                   764 non-null    int64
121 dtypes: float64(2), int64(5)
122 memory usage: 47.8 KB
123
124 Process finished with exit code 0
125
```


3. Split the data into 80% training and 20% testing data. Use target variable as ‘Outcome’.

    a. Use Bagging algorithm on Decision trees to classify *Outcome* and print the default model performance metrics: Accuracy, Precision, Recall, F1Score. Plot

F1Score & accuracy against parameter: *n_estimators* with range of values from 2 to 25.

b. Use Random Forest algorithm to classify *Outcome* and print the default model performance metrics: Accuracy, Precision, Recall, F1Score. Plot F1Score * Accuracy against parameter: *n_estimators* with range of values from 2 to 25.

c. Use Adaboost algorithm on Decision trees to classify *Outcome* and print the default model performance metrics: Accuracy, Precision, Recall, F1Score.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier,
RandomForestClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score

# Load the dataset
df = pd.read_csv('Dataset_Day7.csv')

# Handle missing values
df["Glucose"].fillna(df["Glucose"].median(),
inplace=True)
df["BloodPressure"].fillna(df["BloodPressure"].me
an(), inplace=True)
df["BMI"].fillna(df["BMI"].median(),
inplace=True)
df["Outcome"].fillna(df["Outcome"].mean(),
inplace=True)

# Split the data into training and testing sets
X = df.drop('Outcome', axis=1)
y = df['Outcome']  # Target variable

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=99)

# Part (a) - Bagging on Decision Trees
# Note: Parameter - n_estimators stands for how
many tree we want to grow
```

```
n_estimators = np.arange(2, 26)
f1_scores = []
accuracy = []

for n in n_estimators:
    bagging_clf =
BaggingClassifier(estimator=DecisionTreeClassifie
r(), n_estimators=n)
    bagging_clf.fit(X_train, y_train)
    y_pred = bagging_clf.predict(X_test)
    f1_scores.append(f1_score(y_test, y_pred))
    accuracy.append(accuracy_score(y_test,
y_pred))

print("Bagging (Decision Trees):")
print("Accuracy: ", accuracy)
print("Precision: ", precision_score(y_test,
y_pred))
print("Recall: ", recall_score(y_test, y_pred))
print("F1-Score: ", f1_scores)
best_n_estimators_bagging =
n_estimators[np.argmax(f1_scores)]
print("Best n_estimators for Bagging: ",
best_n_estimators_bagging)

plt.plot(n_estimators, f1_scores, label='F1-
Score')
plt.plot(n_estimators, accuracy,
label='Accuracy')
plt.xlabel('n_estimators')
plt.ylabel('Score')
plt.title('F1-Score & Accuracy vs n_estimators
(Bagging)')
plt.legend()
plt.show()

# Part (b) - Random Forest
f1_times_accuracy = []
for n in n_estimators:
    rf_clf =
RandomForestClassifier(n_estimators=n)
    rf_clf.fit(X_train, y_train)
```

```
    y_pred = rf_clf.predict(X_test)
    f1_times_accuracy.append(f1_score(y_test,
y_pred) * accuracy_score(y_test, y_pred))

print("Random Forest:")
print("Accuracy: ", accuracy_score(y_test,
y_pred))
print("Precision: ", precision_score(y_test,
y_pred))
print("Recall: ", recall_score(y_test, y_pred))
print("F1-Score: ", f1_score(y_test, y_pred))
best_n_estimators_random_forest =
n_estimators[np.argmax(f1_scores)]
print("Best n_estimators for Random Forest: ",
best_n_estimators_random_forest)
plt.plot(n_estimators, f1_times_accuracy)
plt.xlabel('n_estimators')
plt.ylabel('F1-Score * Accuracy')
plt.title('F1-Score * Accuracy vs n_estimators
(Random Forest)')
plt.show()

# Part (c) - Adaboost on Decision Trees
adaboost_clf =
AdaBoostClassifier(estimator=DecisionTreeClassifi
er())
adaboost_clf.fit(X_train, y_train)
y_pred = adaboost_clf.predict(X_test)

print("Adaboost (Decision Trees):")
print("Accuracy: ", accuracy_score(y_test,
y_pred))
print("Precision: ", precision_score(y_test,
y_pred))
print("Recall: ", recall_score(y_test, y_pred))
print("F1-Score: ", f1_score(y_test, y_pred))
best_n_estimators_adaboost =
adaboost_clf.n_estimators
print("Best n_estimators for AdaBoost: ",
best_n_estimators_adaboost)
```

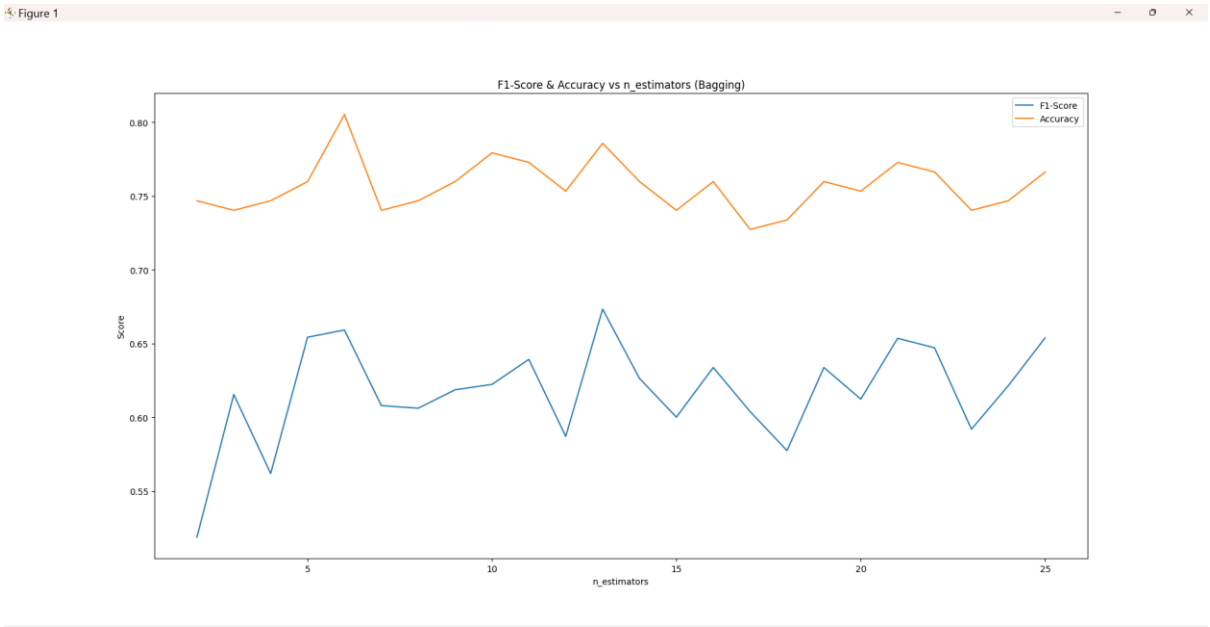
Figure 1
F1-Score & Accuracy vs n_estimators (Bagging)
F1-Score
Accuracy
Score
n_estimators
0.80
0.75
0.70
0.65
0.60
0.55
5
10
15
20
25

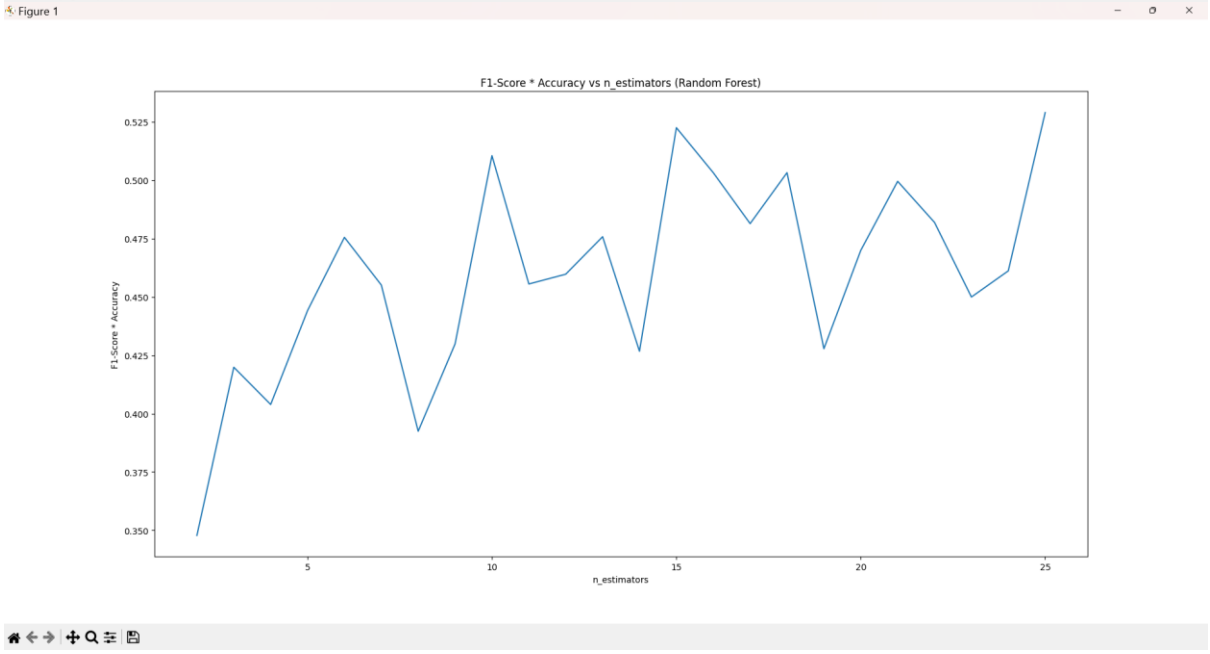
Figure 1
F1-Score * Accuracy vs n_estimators (Random Forest)
F1-Score * Accuracy
n_estimators
0.525
0.500
0.475
0.450
0.425
0.400
0.375
0.350
5
10
15
20
25

File - Day11Q3

```
C:\Users\tejas\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\tejas\PycharmProjects\pythonProject\START\Day11Q3.py
Bagging (Decision Trees):
Accuracy:  [0.7467532467532467, 0.7402597402597403, 0.7467532467532467, 0.7597402597402597, 0.8051948051948052, 0.7402597402597403, 0.7467532467532467, 0.7597402597402597, 0.7792207792207793, 0.7727272727272727, 0.7532467532467533, 0.7857142857142857, 0.7597402597402597, 0.7402597402597403, 0.7597402597402597, 0.7272727272727273, 0.7337662337662337, 0.7597402597402597, 0.7532467532467533, 0.7727272727272727, 0.7662337662337663, 0.7402597402597403, 0.7467532467532467, 0.7662337662337663]
Precision:  0.6181818181818182
Recall:  0.6938775510204082
F1-Score:  [0.5185185185185185, 0.6153846153846153, 0.5617977528089888, 0.6542056074766354, 0.6590909090909091, 0.6078431372549019, 0.6060606060606062, 0.6185567010309279, 0.6222222222222223, 0.6391752577319587, 0.5869565217391305, 0.6732673267326732, 0.6262626262626263, 0.6000000000000001, 0.6336633663366337, 0.6037735849056605, 0.577319587628866, 0.6336633663366337, 0.6122448979591837, 0.6534653465346534, 0.6470588235294118, 0.5918367346938775, 0.6213592233009709, 0.6538461538461539]
Best n_estimators for Bagging:  13
Random Forest:
Accuracy:  0.7857142857142857
Precision:  0.6538461538461539
Recall:  0.6938775510204082
F1-Score:  0.6732673267326732
Best n_estimators for Random Forest:  13
Adaboost (Decision Trees):
Accuracy:  0.7402597402597403
Precision:  0.5882352941176471
Recall:  0.6122448979591837
F1-Score:  0.6000000000000001
Best n_estimators for AdaBoost:  50

Process finished with exit code 0

```