For columns:

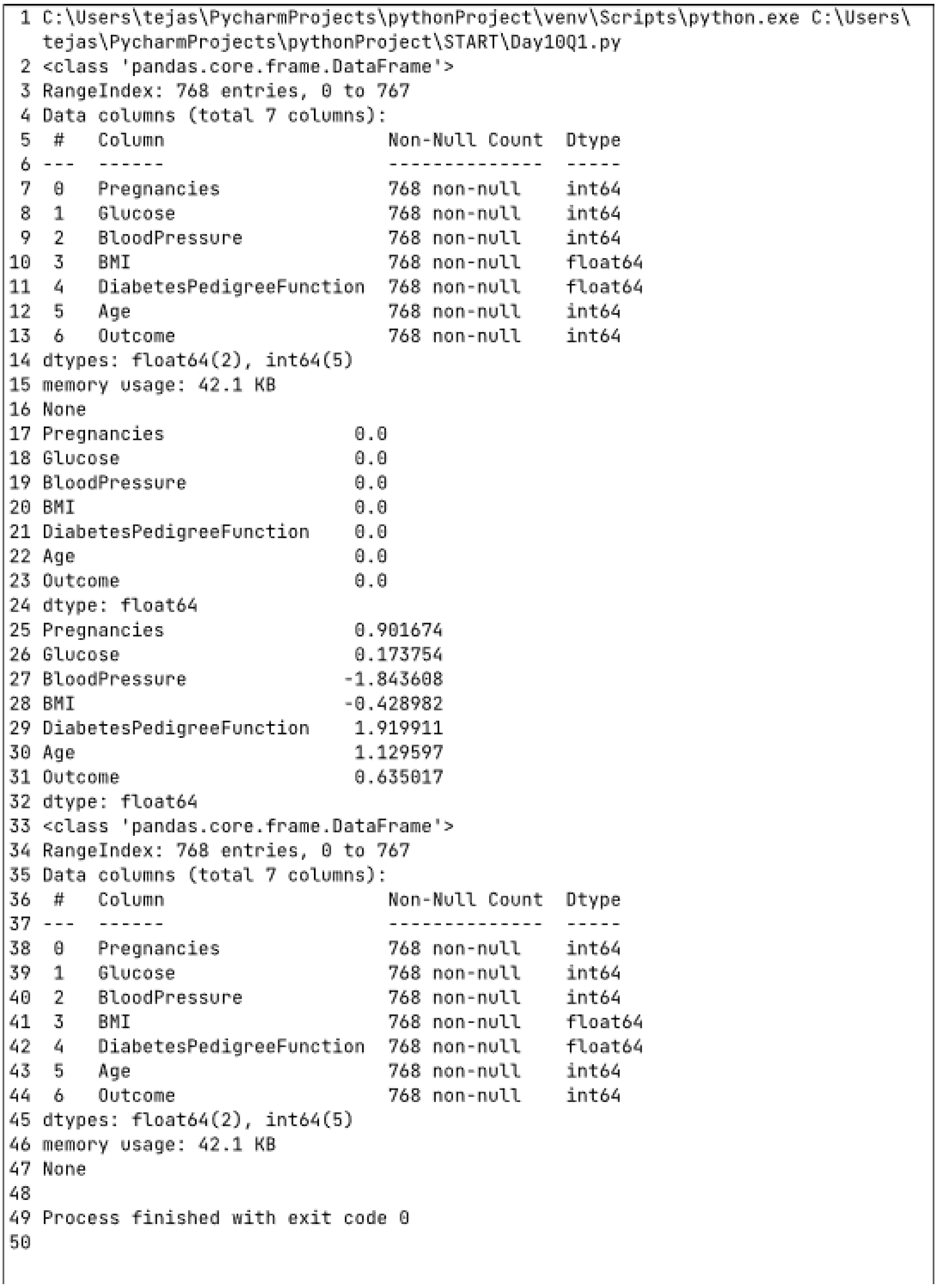
Glucose,
BloodPressure,
BMI,
DiabetesPedigreeFunction

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"].
mean(), inplace=True)
df["BMI"].fillna(df["BMI"].median(),
inplace=True)
df["Outcome"].fillna(df["Outcome"].mean(),
inplace=True)
print(df.info())
```

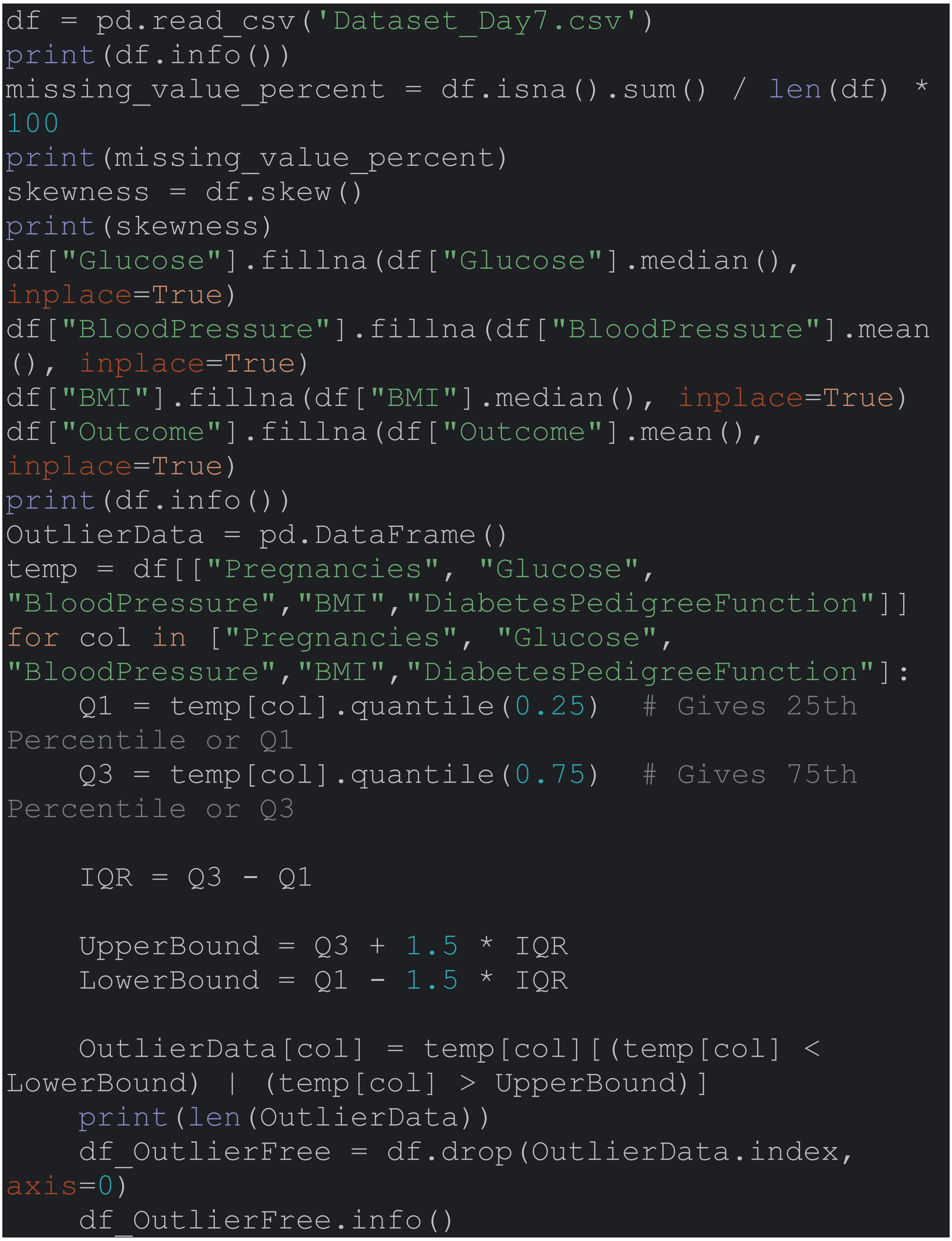
File - Day10Q1



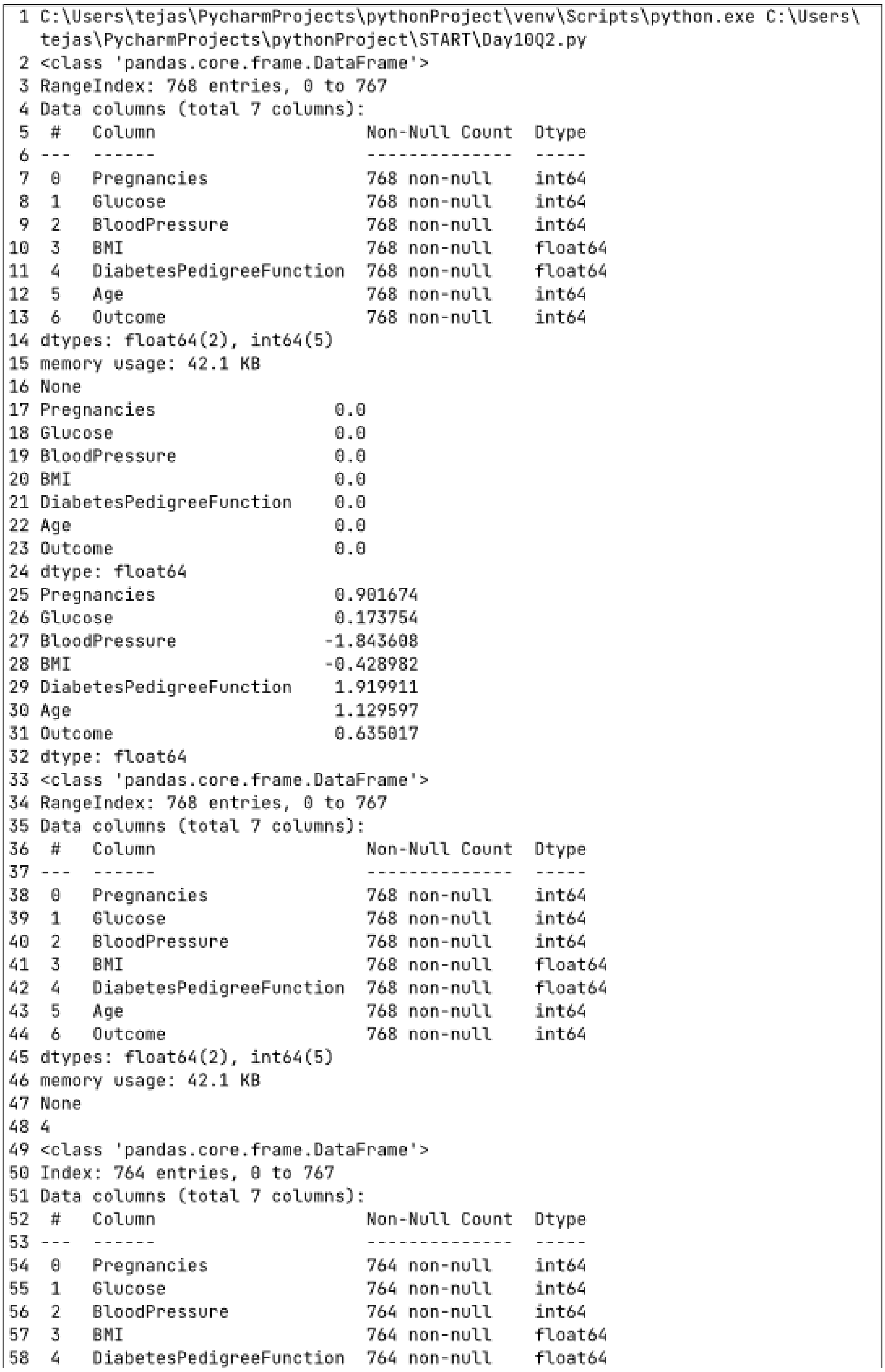
```
C:\Users\tejas\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\tejas\PycharmProjects\pythonProject\START\Day10Q1.py
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 7 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 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.

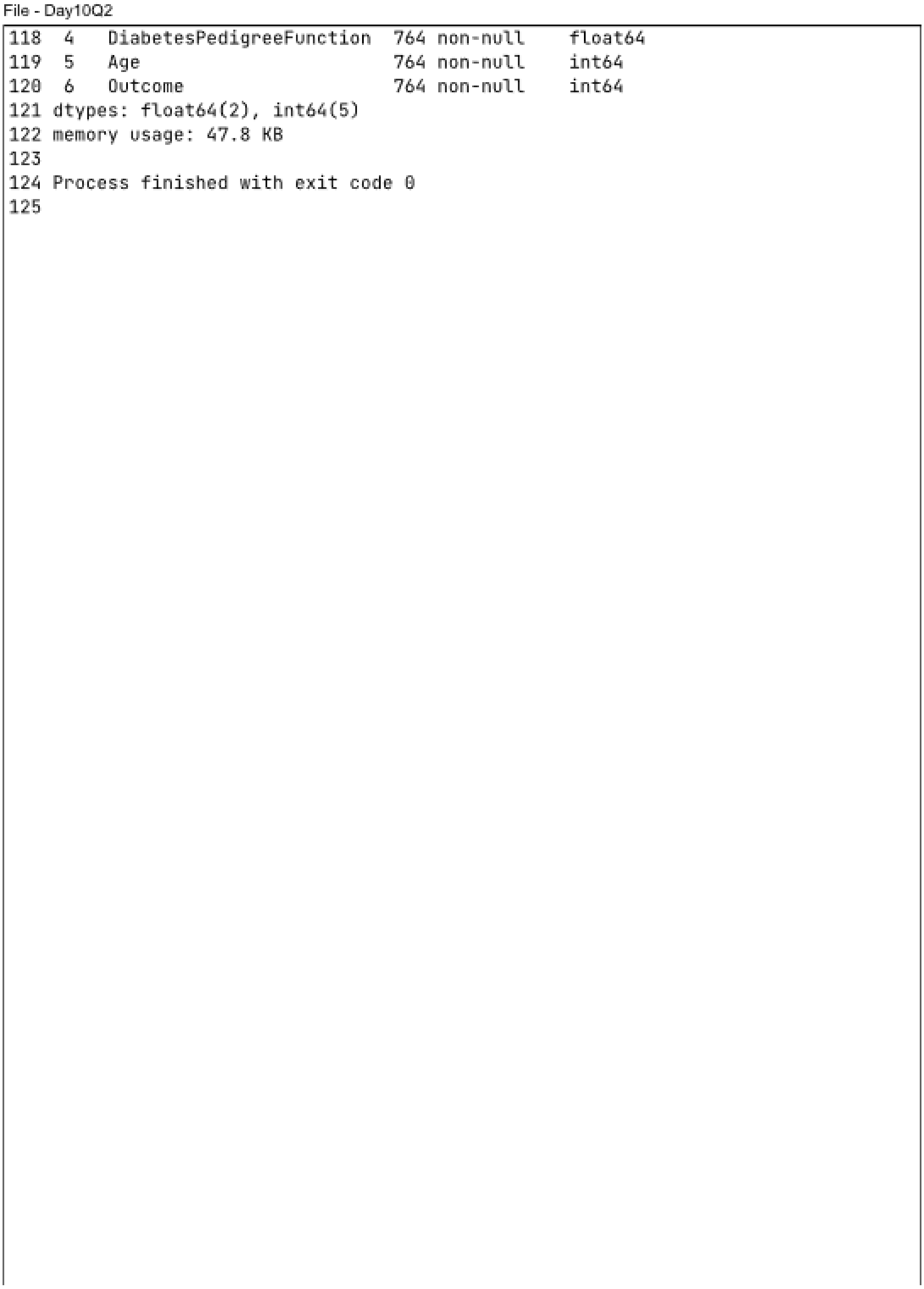
```
import numpy as np
import pandas as pd
```



```
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"].mean
(), 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()
```



```
C:\Users\tejas\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\tejas\PycharmProjects\pythonProject\START\Day10Q2.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 - Day10Q2

```
 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

Process finished with exit code 0

```

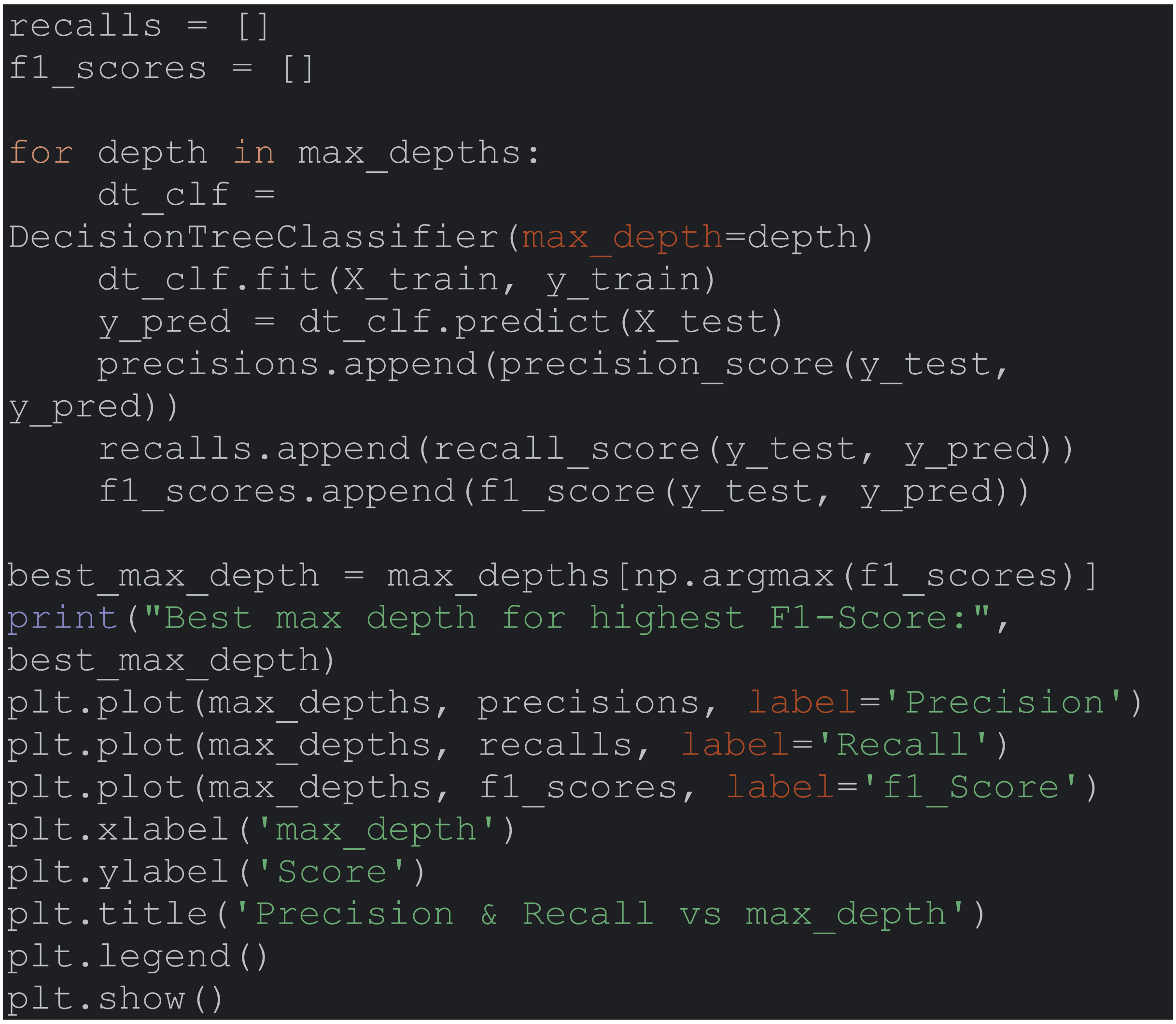
3.Split the data into 80% training and 20% testing data. Then, use a Decision Tree classifier algorithm with target variable as ‘Outcome’.

a. Print the default model performance metrics: Accuracy, Precision, Recall, F1Score

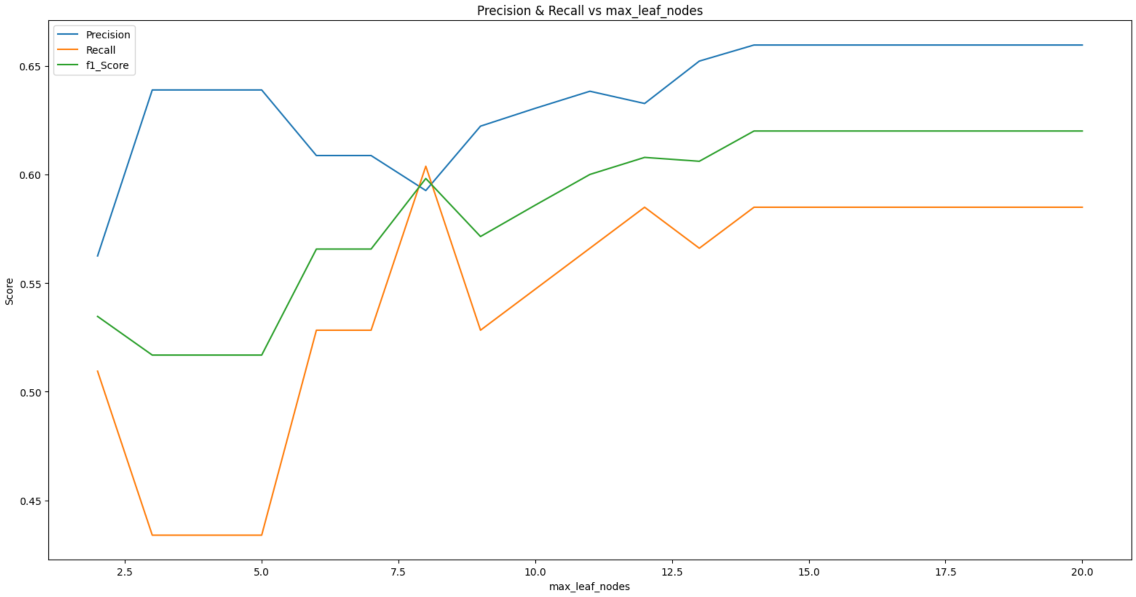
b. Plot a Precision & Recall vs max_leaf_nodes (**consider a range of numbers**) curve (both Prec and Rec on the same graph). Find the kernel type for which F1-score is the highest.

c. Plot a Precision & Recall vs max_depth (**consider a range of numbers**) curve (both Prec and Rec on the same graph). Find the kernel type for which F1-score is the highest.

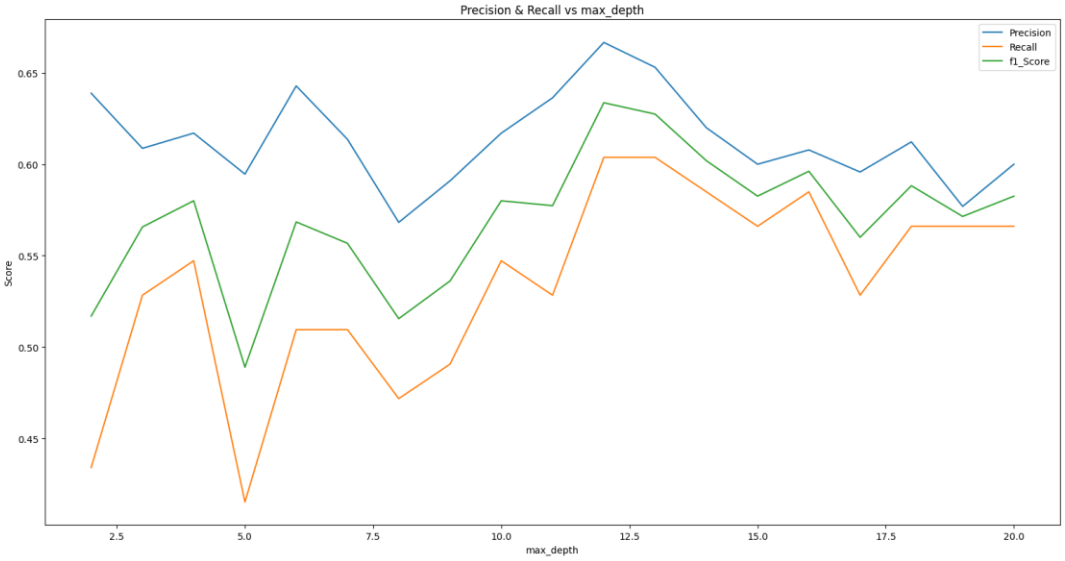
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import
train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
precision_score, recall_score, f1_score

# 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"].mean
(), 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)  # Features
y = df['Outcome']  # Target variable

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=50)

# Part (a) - Default model performance metrics
dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train, y_train)
y_pred = dt_clf.predict(X_test)

print("Model Performance metrics (Default
Parameters):\n")
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))

# Part (b) - Plot Precision & Recall vs
max_leaf_nodes
max_leaf_nodes = np.arange(2, 21)
precisions = []
recalls = []
f1_scores = []

for nodes in max_leaf_nodes:
    dt_clf =
DecisionTreeClassifier(max_leaf_nodes=nodes)
    dt_clf.fit(X_train, y_train)
    y_pred = dt_clf.predict(X_test)
    precisions.append(precision_score(y_test,
y_pred))
    recalls.append(recall_score(y_test, y_pred))
    f1_scores.append(f1_score(y_test, y_pred))

best_max_leaf_node =
max_leaf_nodes[np.argmax(f1_scores)]
print("Best max leaf nodes for highest F1-Score:",
best_max_leaf_node)
plt.plot(max_leaf_nodes, precisions,
label='Precision')
plt.plot(max_leaf_nodes, recalls, label='Recall')
plt.plot(max_leaf_nodes, f1_scores,
label='f1_Score')
plt.xlabel('max_leaf_nodes')
plt.ylabel('Score')
plt.title('Precision & Recall vs max_leaf_nodes')
plt.legend()
plt.show()

# Part (c) - Plot Precision & Recall vs max_depth
max_depths = np.arange(2, 21)
precisions = []
```



```
recalls = []
f1_scores = []

for depth in max_depths:
    dt_clf =
DecisionTreeClassifier(max_depth=depth)
    dt_clf.fit(X_train, y_train)
    y_pred = dt_clf.predict(X_test)
    precisions.append(precision_score(y_test,
y_pred))
    recalls.append(recall_score(y_test, y_pred))
    f1_scores.append(f1_score(y_test, y_pred))

best_max_depth = max_depths[np.argmax(f1_scores)]
print("Best max depth for highest F1-Score:",
best_max_depth)
plt.plot(max_depths, precisions, label='Precision')
plt.plot(max_depths, recalls, label='Recall')
plt.plot(max_depths, f1_scores, label='f1_Score')
plt.xlabel('max_depth')
plt.ylabel('Score')
plt.title('Precision & Recall vs max_depth')
plt.legend()
plt.show()
```



Precision & Recall vs max_leaf_nodes
Precision
Recall
f1_Score
Score
0.65
0.60
0.55
0.50
0.45
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_leaf_nodes



Precision & Recall vs max_depth
Precision
Recall
f1_Score
Score
0.65
0.60
0.55
0.50
0.45
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
max_depth

```
C:\Users\tejas\PycharmProjects\pythonProject\venv\Scripts\python.exe C:\Users\tejas\PycharmProjects\pythonProject\START\Day10Q3.py
Model Performance metrics (Default Parameters):

Accuracy:  0.7272727272727273
Precision:  0.6122448979591837
Recall:  0.5660377358490566
F1-Score:  0.588235294117647
Best max leaf nodes for highest F1-Score: 14
Best max depth for highest F1-Score: 12

Process finished with exit code 0

```