Assignment 2 - Introduction To Data Science

Group - 2

Team Members

- Gaurav Singhal - 2023mt03187,
- Deepak Khandelwal - 2023mt03164,
- Debleena Goswami - 2023mt03099
- Palan Bhavin Bharatbhai - 2023mt03146

Problem Statement

Device crashes can lead to significant downtime and loss of productivity, especially in environments where the device plays a critical role in daily operations. Predicting Risk of Device Crash, based on various operational and environmental factors, can help mitigate these issues by enabling preemptive measures.

In assignment 2 - To classify the data into Crash Risk Category, We have used Decision Tree Classifier, Evaluated Model and Fine Tuned based on various Hyper Paarameters.

Device Crash Scenario - Choice of Metric

- Precision - Precision is important as false positives are costly. In our case, if the model falsely predicts a crash when there is no crash, it might lead to unnecessary device shutdowns, which can be disruptive or costly.
- Recall - Recall is crucial as false negatives are costly. In our case, if the model fails to predict a crash (false negative), the device could crash unexpectedly, leading to potentially more severe consequences like data loss or system failure. Thus, you'd want to minimize false negatives.

As both Precision and Recall are important - F1 Score is a suitable metric to calculate the performance

The Mobile Crash Data

The data has been used from Kaggle (link Below). As a team we discussed the varios factors in a real world scenario which can lead to low performance of a mobile device. The below step enlists the columns and values of the factors like CPU Usage, APP Name (Running in the device), Memory Usage, Battery Level , Temperature , Disk Space and more. Data Contains different type of attributes ranging from Numerical, Binary to Categorical.

https://www.kaggle.com/datasets/dkkh8788/dataset-crash-multiclass

Python Notebook Github Link

https://github.com/dkkh8788/Classification-Decision-tree/blob/main/Group-2.ipynb

Below Code cell contains the code from Assignment 1 on same data set that does, pre-processing and EDA on the Device Crash Data.

```
# Importing the required python Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import matplotlib.pyplot as plt

#Load the data set
df = pd.read_csv('mobile_crash_data_1115_v1.csv')

#Summary Statistics will do find mean and Quantiles finding central
tendencies and dispersion
print("\nSummary Statistics:")
display(df.describe(include='all'))

# Find records greater than 100 in 'Battery Level' which is beyond
conformity
outliers = df[df['Battery_Level'] > 100]
print("\nRecords greater than 100:")
print(outliers['Battery_Level'])

# Remove records greater than 100
cleaned_data = df[df['Battery_Level'] <= 100]

outliers = cleaned_data[cleaned_data['Battery_Level'] > 100]
print("\nRecords greater than 100:")
display(outliers['Battery_Level'])

# Identify outliers in all numeric coloumns
new_df = cleaned_data.copy()
numeric_cols = cleaned_data.select_dtypes(include=np.number).columns
non_binary_non_discrete_numerical_columns = [col for col in
numeric_cols if cleaned_data[col].nunique() > 10]

print(f"cleaned_data summary")
print(f"=====================================================================")
print(cleaned_data.describe())
for col in non_binary_non_discrete_numerical_columns:
    Q1 = cleaned_data[col].quantile(0.25)
    Q3 = cleaned_data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
```

```
    upper_bound = Q3 + 1.5 * IQR
    print(f"\n{col}, Outliers Range[lb, ub] => [{lower_bound},
{upper_bound}] ")

print(f"=================================================================
=======")
    filtered_df = cleaned_data[(cleaned_data[col] < lower_bound) |
(cleaned_data[col] > upper_bound)]
    if not filtered_df.empty:
        print(filtered_df.head(4))
        print(f"\n")
    # removing outliers in this coloumn
    temp_df = cleaned_data[(cleaned_data[col] >= lower_bound) &
(cleaned_data[col] <= upper_bound)]
    new_df[col] = temp_df[col]

non_binary_non_discrete_numerical_columns = [col for col in
numeric_cols if new_df[col].nunique() > 10]
non_binary_non_discrete_numerical_columns_df =
new_df[non_binary_non_discrete_numerical_columns]

fig, ax = plt.subplots(figsize=(12, 8))
# Create box plots
non_binary_non_discrete_numerical_columns_df.boxplot(ax=ax)

plt.title("Box Plots After Outliers removed")
plt.ylabel('Values')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

cleaned_data = new_df

# Missing values
print("\nMissing Values Before Imputation:")
print(cleaned_data.isnull().sum())

cleaned_data['CPU_Usage']=cleaned_data['CPU_Usage'].fillna(cleaned_dat
a['CPU_Usage'].mean())

cleaned_data['Memory_Usage']=cleaned_data['Memory_Usage'].fillna(clean
ed_data['Memory_Usage'].median())

cleaned_data['Temperature']=cleaned_data['Temperature'].ffill()
cleaned_data['Session_Time']=cleaned_data['Session_Time'].fillna(clean
ed_data['Session_Time'].median())


print("\nMissing Values after imputation:")
print(cleaned_data.isnull().sum())
```

```
## round to 2 decimal digit
cleaned_data = cleaned_data.round(2)
display(cleaned_data.head())

######## Data Preprocessing ##########


#Binning involves grouping continuous data into discrete categories or
bins.
min_value = cleaned_data['Session_Time'].min()
max_value = cleaned_data['Session_Time'].max()

# Suppose the bin size is 4
# Returns num evenly spaced samples, calculated over the interval
[start, stop].
bins = np.linspace(min_value,max_value,4)
display(bins)


#Labeling and grouping into fixed categories
labels = ['Low', 'Medium', 'High'];

# We need to specify the bins and the labels.
cleaned_data['Session_Time'] = pd.cut(cleaned_data['Session_Time'],
bins=bins, labels=labels, include_lowest=True)
display(cleaned_data['Session_Time'])

#Use Label Encoder to make all categorical attributes as numerical
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
cleaned_data['App_Usage_Level']=le.fit_transform(cleaned_data['App_Usa
ge_Level'])

cleaned_data['Device_Model']=le.fit_transform(cleaned_data['Device_Mod
el'])

cleaned_data['App_Name']=le.fit_transform(cleaned_data['App_Name'])

cleaned_data['Session_Time']=le.fit_transform(cleaned_data['Session_Ti
me'])

cleaned_data['Crash_Label']=le.fit_transform(cleaned_data['Crash_Label
'])


# Validate if now all attributes are numerical
print("\nData Types:")
display(cleaned_data.dtypes)

display(cleaned_data.head())
```

```
#########

 ## Objective 4 - Feature Selection

#Using Pearson Correlation to find correlation between different
attributes as well as with label attribute to do Fetaure Selection.
from tkinter import TRUE
# Correlation Matrix - Internally uses Pearson Correlation
cor = cleaned_data.corr()

# Plotting Heatmap
plt.figure(figsize = (10,6))
#sns.heatmap(cor, annot=True)
sns.heatmap(
    cor,
    annot=True,               # Annotate cells with their values
    fmt=".2f",               # Format for annotation
    cmap='coolwarm',         # Color map
    cbar=True,               # Show color bar
    linewidths=.5,           # Lines between cells
    linecolor='black',        # Color of the lines
    square=False,             # Make cells square-shaped
    xticklabels=True,        # Show x-tick labels
    yticklabels=True         # Show y-tick labels
)
plt.show()

# Removing Duration_Since_Last_charge as it is redundent

data_cleaned =
cleaned_data.drop(columns=['Duration_Since_Last_Charge'])
display(data_cleaned.dtypes)

#Finding top 5 features that contribute the most to the classification
task.

from sklearn.ensemble import RandomForestRegressor
# define the model
model = RandomForestRegressor()

X = data_cleaned.iloc[:,[0,1,2,3,4,5,6,7,8,9,10,11,12]]
Y = data_cleaned.iloc[:,[13]]

# fit the model
model.fit(X,Y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
```

```
    #print('Feature: %0d, Score: %.5f' % (i,v))
    print(f'Feature: {X.columns[i]}, Score: {v:.5f}')
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.title("Feature Importance")
plt.ylabel('Importance towards Crash_Label')
plt.xlabel('Features')
plt.show()
```

```
Summary Statistics:

       App_Name     CPU_Usage  Memory_Usage  Battery_Level
Temperature  \
count     11560  11329.000000  11329.000000   11560.000000
11329.000000
unique       15           NaN           NaN            NaN
NaN
top     Discord           NaN           NaN            NaN
NaN
freq        807           NaN           NaN            NaN
NaN
mean        NaN     62.397586     56.959348      50.304638
36.832064
std         NaN     16.272409      9.880617      28.954965
9.403873
min         NaN      4.310223     24.027532       0.000000
10.115146
25%         NaN     51.389702     50.215708      25.270028
31.205169
50%         NaN     62.486420     56.980808      50.390288
37.597585
75%         NaN     73.819859     63.572382      74.859107
43.533911
max         NaN     99.952131     94.278519     149.286188
55.000000

          Disk_Space  Network_Signal   App_Version    Error_Logs
Device_Model  \
count   11560.000000    11560.000000  11560.000000  11560.000000
11560
unique           NaN             NaN           NaN           NaN
3
top              NaN             NaN           NaN           NaN
Model_B
freq             NaN             NaN           NaN           NaN
3893
mean      257.773411       50.307640      1.327820      0.505709
NaN
std       147.875113       28.929610      0.398261      0.499989
```

```
NaN
min          0.047753         0.000930      1.000000      0.000000
NaN
25%        129.936641        25.440228      1.000000      0.000000
NaN
50%        257.998770        50.380077      1.200000      1.000000
NaN
75%        386.445334        75.435823      2.000000      1.000000
NaN
max        511.929107        99.997387      2.000000      1.000000
NaN

        Session_Time  Num_App_Crashes  Duration_Since_Last_Charge  \
count   11560.000000     11560.000000                11560.000000
unique           NaN              NaN                         NaN
top              NaN              NaN                         NaN
freq             NaN              NaN                         NaN
mean      170.070402         1.462976                   23.875100
std        11.999830         0.908865                   13.837832
min       127.948335         0.000000                    0.004914
25%       161.946831         1.000000                   12.097158
50%       170.231241         1.000000                   23.827640
75%       178.066092         2.000000                   35.843880
max       215.188104         4.000000                   47.995185

       App_Usage_Level Crash_Label
count            11560       11560
unique               3           4
top             Medium         Hig
freq              5672        4427
mean               NaN         NaN
std                NaN         NaN
min                NaN         NaN
25%                NaN         NaN
50%                NaN         NaN
75%                NaN         NaN
max                NaN         NaN


Records greater than 100:
11548    149.286188
11549    146.987775
11550    138.980624
11551    141.555246
11552    139.083107
11553    142.135888
11554    112.648205
11555    122.810369
11556    110.043715
11557    129.408817
```

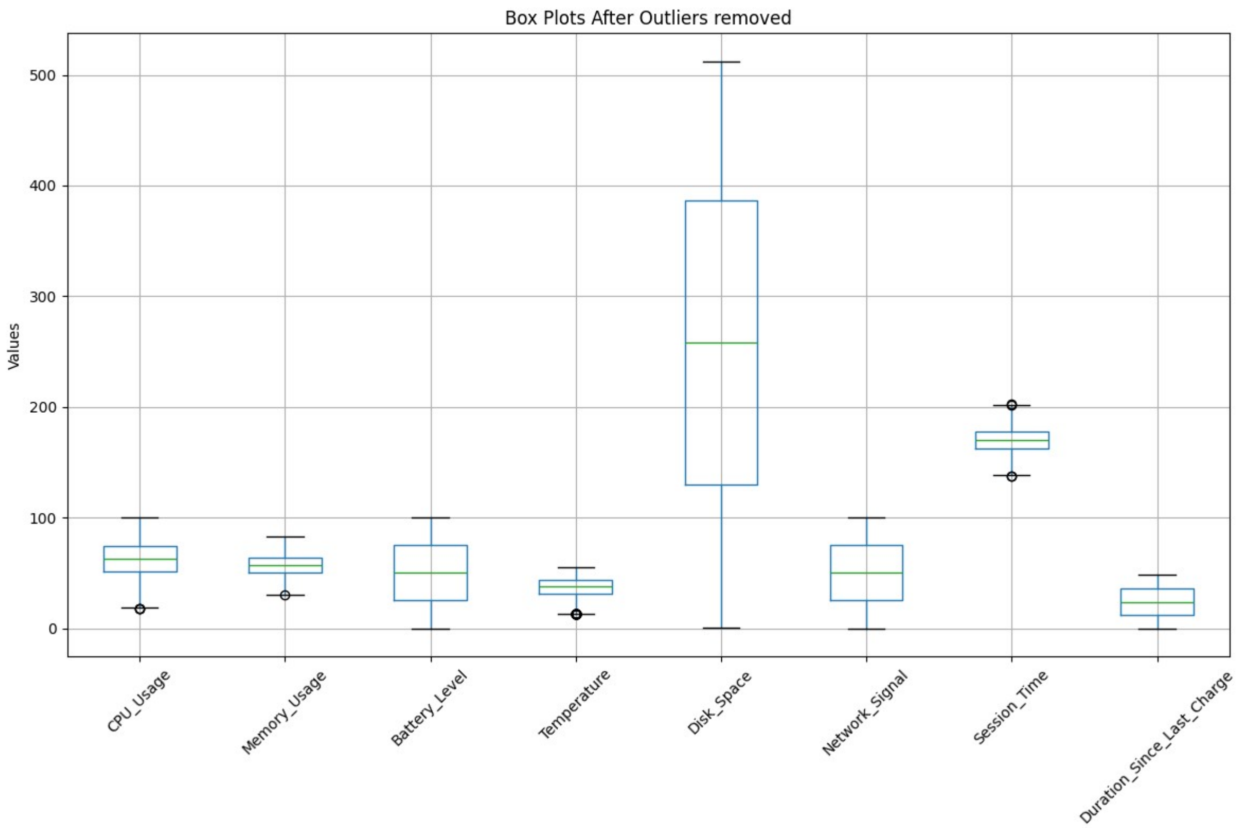
```
11558    115.882346
11559    136.678449
Name: Battery_Level, dtype: float64

Records greater than 100:

Series([], Name: Battery_Level, dtype: float64)

cleaned_data summary
======================================================================
=
          CPU_Usage  Memory_Usage  Battery_Level   Temperature
Disk_Space  \
count  11317.000000  11317.000000   11548.000000  11317.000000
11548.000000
mean      62.396201     56.960900      50.219615     36.838100
257.815966
std       16.275145      9.876400      28.846484      9.401749
147.880769
min        4.310223     24.027532       0.000000     10.115146
0.047753
25%       51.401378     50.218679      25.264509     31.214783
129.893457
50%       62.487586     56.980808      50.375528     37.598273
258.065780
75%       73.819859     63.572382      74.780431     43.535505
386.511185
max       99.952131     94.278519     100.000000     55.000000
511.929107

       Network_Signal   App_Version    Error_Logs  Session_Time  \
count    11548.000000  11548.000000  11548.000000  11548.000000
mean        50.303771      1.327684      0.505715    170.068714
std         28.935591      0.398166      0.499989     12.002211
min          0.000930      1.000000      0.000000    127.948335
25%         25.414422      1.000000      0.000000    161.942817
50%         50.377723      1.200000      1.000000    170.231241
75%         75.445145      2.000000      1.000000    178.066092
max         99.997387      2.000000      1.000000    215.188104

       Num_App_Crashes  Duration_Since_Last_Charge
count     11548.000000                11548.000000
mean          1.462158                   23.876836
std           0.907923                   13.838301
min           0.000000                    0.004914
25%           1.000000                   12.097158
50%           1.000000                   23.830493
75%           2.000000                   35.858172
max           4.000000                   47.995185
```

```
CPU_Usage, Outliers Range[lb, ub] => [17.7736578296853,
107.44757949232508]
======================================================================
        App_Name  CPU_Usage  Memory_Usage  Battery_Level  Temperature
\
50   Google Maps  13.487756     66.965693       0.000000    16.936001

74         Slack  13.233145     71.657289      18.576445    15.132392

173     Snapchat  17.140654     51.426192      24.038154    17.627353

271     Facebook   9.957077     43.584716      42.352317    15.326085


     Disk_Space  Network_Signal  App_Version  Error_Logs Device_Model
\
50   489.068751       95.152984          2.0           0      Model_C

74   213.984642       85.232868          1.2           0      Model_A

173  147.963900        8.763582          1.1           0      Model_C

271  170.922055       32.082426          1.2           0      Model_C


     Session_Time  Num_App_Crashes  Duration_Since_Last_Charge  \
50     152.550148                0                   47.338311
74     166.592642                1                   39.453726
173    179.585676                0                   37.013300
271    174.567159                1                   28.686906

    App_Usage_Level Crash_Label
50           Medium         Cri
74              Low         Med
173             Low         Low
271             Low         Low



Memory_Usage, Outliers Range[lb, ub] => [30.188124412659327,
83.60293689764828]
======================================================================
        App_Name  CPU_Usage  Memory_Usage  Battery_Level  Temperature
\
43   Google Maps  66.247185     83.964010      46.810409    40.684433

57       Netflix  39.176034     30.081611      53.696679          NaN

85      Facebook  85.637648     89.330365      56.455036    47.670644

226     Telegram  49.279034     28.404739      53.441764    32.973051
```

```
     Disk_Space  Network_Signal  App_Version  Error_Logs Device_Model  \
43   153.904385       76.333964          1.0           1      Model_B

57   259.715384       23.098987          1.0           1      Model_B

85   452.013762       63.113690          2.0           0      Model_C

226  417.732369        9.801543          1.2           0      Model_C


     Session_Time  Num_App_Crashes  Duration_Since_Last_Charge  \
43     167.548806                2                   26.020113
57     154.954805                1                   24.329426
85     169.420467                1                   20.788190
226    181.089893                1                   22.345511

    App_Usage_Level Crash_Label
43           Medium         Hig
57           Medium         Low
85             High         Hig
226             Low         Hig



Battery_Level, Outliers Range[lb, ub] => [-49.009374399917434,
149.05431390446722]
=====================================================================

Temperature, Outliers Range[lb, ub] => [12.733699528834766,
62.016587989330006]
=====================================================================
         App_Name  CPU_Usage  Memory_Usage  Battery_Level  Temperature  \
4148       Amazon   4.310223     58.549497      98.969306    10.150442

7138     Telegram  10.057908     47.754230      57.432725    10.115146

9036     Snapchat  64.082543     58.335432       6.313506    10.835434

9037  Google Maps  47.291453     58.120915      53.102186    10.808375


      Disk_Space  Network_Signal  App_Version  Error_Logs Device_Model  \
4148  135.604815       91.801156          2.0           0      Model_C

7138  269.413785       95.679137          2.0           1      Model_C
```

```
9036  331.972823       52.546160         2.0            0      Model_A

9037  415.965688       84.823668         1.2            1      Model_C


      Session_Time  Num_App_Crashes  Duration_Since_Last_Charge  \
4148    162.349520                1                    2.508105
7138    152.936058                0                   20.177895
9036    184.990744                2                   43.529176
9037    180.474822                1                   21.314676

     App_Usage_Level Crash_Label
4148          Medium         Low
7138          Medium         Low
9036            High         Hig
9037             Low         Hig



Disk_Space, Outliers Range[lb, ub] => [-255.03313367640143,
771.4377756086253]
=====================================================================

Network_Signal, Outliers Range[lb, ub] => [-49.63166187618599,
150.491229174725]
=====================================================================

Session_Time, Outliers Range[lb, ub] => [137.75790491931173,
202.25100467731284]
=====================================================================
      App_Name  CPU_Usage  Memory_Usage  Battery_Level  Temperature  \
145  Instagram  84.041416     64.248936      81.220841    47.308205
247    Spotify  70.130939     59.759087      70.877142    40.594072
301   WhatsApp  85.513879     74.602058      17.455935    48.827514
313       Uber  86.511620     54.403387      34.126206    46.589666

     Disk_Space  Network_Signal  App_Version  Error_Logs Device_Model
\
145  494.234964       85.421674         2.0            0      Model_B

247  381.154328       49.177406         2.0            1      Model_A

301  202.226877       29.556774         2.0            0      Model_A

313  375.794314       62.483719         2.0            0      Model_C


     Session_Time  Num_App_Crashes  Duration_Since_Last_Charge  \
145    208.358642                1                    8.260518
247    205.002132                2                   14.892533
301    132.716447                1                   38.993907
```

```
313    202.831170               2                   30.609609

    App_Usage_Level Crash_Label
145          Medium         Hig
247             Low         Med
301            High         Hig
313          Medium         Hig



Duration_Since_Last_Charge, Outliers Range[lb, ub] => [-
23.544361956656303,  71.49969233999829]
=====================================================================
```

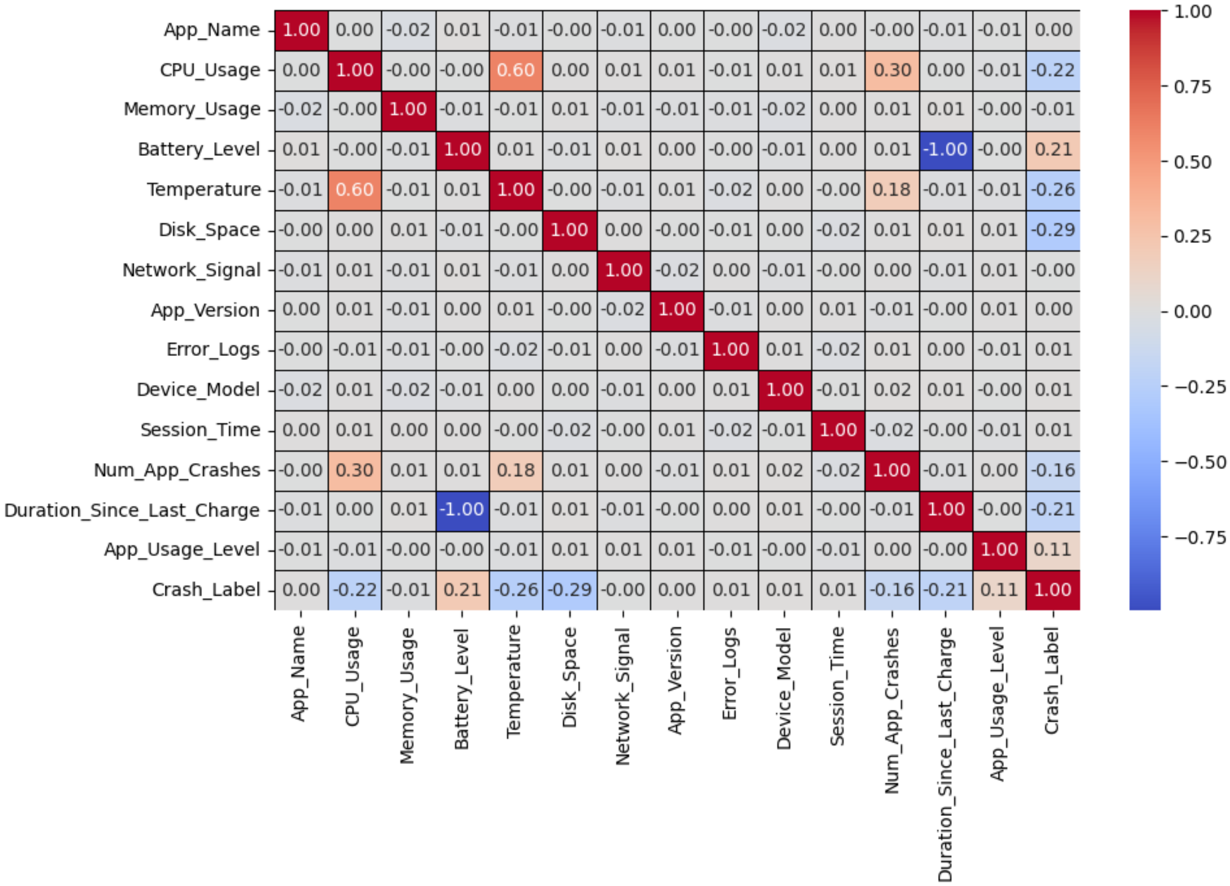



```
Missing Values Before Imputation:
App_Name                          0
CPU_Usage                       281
Memory_Usage                    308
Battery_Level                     0
Temperature                     390
Disk_Space                        0
Network_Signal                    0
App_Version                       0
```

```
Error_Logs                    0
Device_Model                  0
Session_Time                 97
Num_App_Crashes               0
Duration_Since_Last_Charge    0
App_Usage_Level               0
Crash_Label                   0
dtype: int64

Missing Values after imputation:
App_Name                    0
CPU_Usage                   0
Memory_Usage                0
Battery_Level               0
Temperature                 0
Disk_Space                  0
Network_Signal              0
App_Version                 0
Error_Logs                  0
Device_Model                0
Session_Time                0
Num_App_Crashes             0
Duration_Since_Last_Charge  0
App_Usage_Level             0
Crash_Label                 0
dtype: int64

  App_Name  CPU_Usage  Memory_Usage  Battery_Level  Temperature
Disk_Space  \
0  Netflix      35.47         52.05          30.17        22.55
484.09
1  Twitter      98.13         35.68          42.99        50.85
315.26
2  Discord      61.32         60.21          33.43        42.39
13.54
3    Slack      86.21         52.39          65.45        48.86
333.17
4   Amazon      77.27         57.53           3.69        47.11
483.71

   Network_Signal  App_Version  Error_Logs Device_Model  Session_Time
\
0           77.38          1.1           1      Model_C        183.23

1           88.07          1.2           1      Model_C        171.33

2           26.11          1.2           0      Model_B        165.10

3            9.51          1.2           0      Model_A        157.10
```

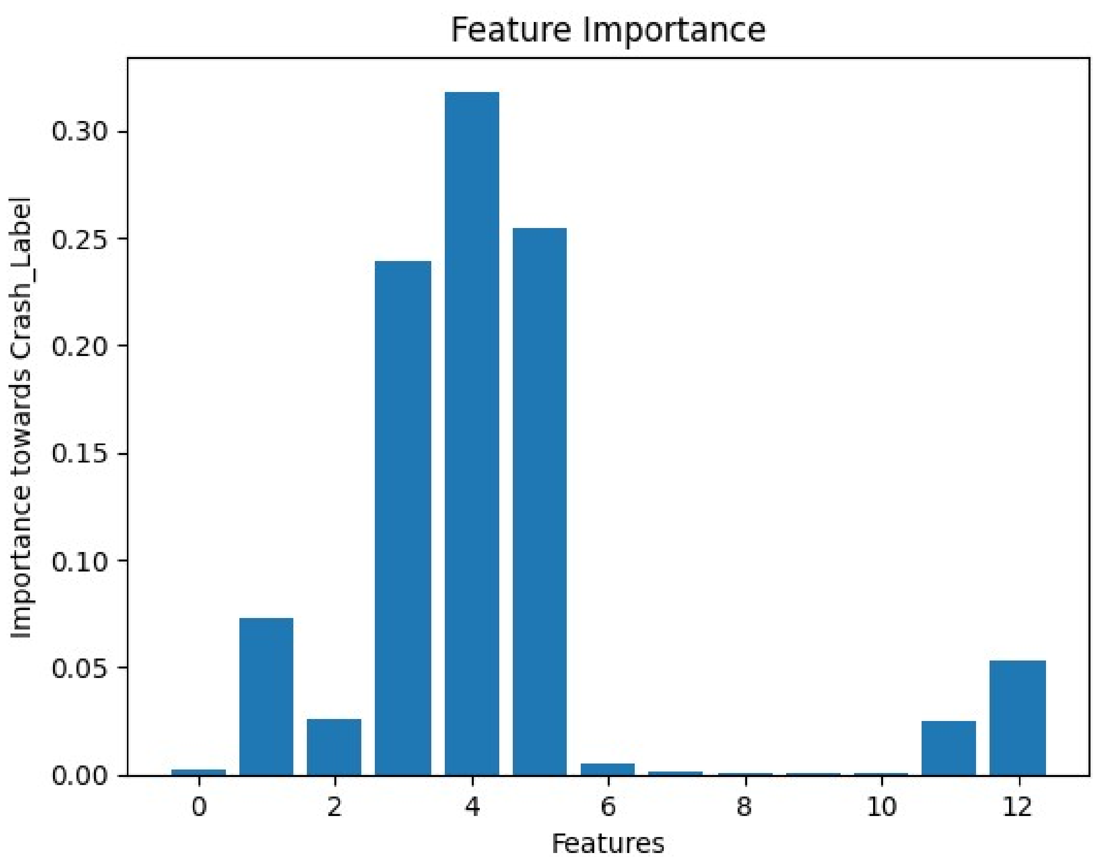
```
4           13.69          2.0           1      Model_B        151.32


   Num_App_Crashes  Duration_Since_Last_Charge App_Usage_Level
Crash_Label
0                1                       34.10          Medium
Hig
1                2                       25.20          Medium
Cri
2                1                       32.87             Low
Med
3                1                       16.24             Low
Hig
4                2                       44.96             Low
Hig

array([137.96      , 159.36333333, 180.76666667, 202.17      ])

0          High
1        Medium
2        Medium
3           Low
4           Low
         ...
11543    Medium
11544    Medium
11545    Medium
11546    Medium
11547    Medium
Name: Session_Time, Length: 11548, dtype: category
Categories (3, object): ['Low' < 'Medium' < 'High']


Data Types:

App_Name                          int64
CPU_Usage                       float64
Memory_Usage                    float64
Battery_Level                   float64
Temperature                     float64
Disk_Space                      float64
Network_Signal                  float64
App_Version                     float64
Error_Logs                        int64
Device_Model                      int64
Session_Time                      int64
Num_App_Crashes                   int64
Duration_Since_Last_Charge      float64
App_Usage_Level                   int64
```

```
Crash_Label                      int64
dtype: object

   App_Name  CPU_Usage  Memory_Usage  Battery_Level  Temperature
Disk_Space  \
0         6      35.47         52.05          30.17        22.55
484.09
1        11      98.13         35.68          42.99        50.85
315.26
2         1      61.32         60.21          33.43        42.39
13.54
3         7      86.21         52.39          65.45        48.86
333.17
4         0      77.27         57.53           3.69        47.11
483.71

   Network_Signal  App_Version  Error_Logs  Device_Model  Session_Time
\
0           77.38          1.1           1             2             0

1           88.07          1.2           1             2             2

2           26.11          1.2           0             1             2

3            9.51          1.2           0             0             1

4           13.69          2.0           1             1             1


   Num_App_Crashes  Duration_Since_Last_Charge  App_Usage_Level
Crash_Label
0                1                       34.10                2
1
1                2                       25.20                2
0
2                1                       32.87                1
3
3                1                       16.24                1
1
4                2                       44.96                1
1
```




```
App_Name                  int64
CPU_Usage               float64
Memory_Usage            float64
Battery_Level           float64
Temperature             float64
Disk_Space              float64
Network_Signal          float64
App_Version             float64
Error_Logs                int64
Device_Model              int64
Session_Time              int64
Num_App_Crashes           int64
App_Usage_Level           int64
Crash_Label               int64
dtype: object

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
site-packages/sklearn/base.py:1473: DataConversionWarning: A column-
vector y was passed when a 1d array was expected. Please change the
shape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

```
Feature: App_Name, Score: 0.00290
Feature: CPU_Usage, Score: 0.07249
Feature: Memory_Usage, Score: 0.02607
Feature: Battery_Level, Score: 0.23914
Feature: Temperature, Score: 0.31792
Feature: Disk_Space, Score: 0.25481
Feature: Network_Signal, Score: 0.00510
Feature: App_Version, Score: 0.00140
Feature: Error_Logs, Score: 0.00047
Feature: Device_Model, Score: 0.00093
Feature: Session_Time, Score: 0.00075
Feature: Num_App_Crashes, Score: 0.02466
Feature: App_Usage_Level, Score: 0.05335
```



Assignment 2 Scope

```
# Drop redundenent, or irrelevent features

data_cleaned = data_cleaned.drop(columns=['App_Version'])
data_cleaned = data_cleaned.drop(columns=['App_Name'])
data_cleaned = data_cleaned.drop(columns=['Error_Logs'])
```

```
data_cleaned = data_cleaned.drop(columns=['Device_Model'])
data_cleaned = data_cleaned.drop(columns=['Session_Time'])
data_cleaned = data_cleaned.drop(columns=['Network_Signal'])

display(data_cleaned.dtypes)

CPU_Usage            float64
Memory_Usage         float64
Battery_Level        float64
Temperature          float64
Disk_Space           float64
Num_App_Crashes        int64
App_Usage_Level        int64
Crash_Label            int64
dtype: object
```

1. Decision Tree Implementation

1.1 Split the dataset into training and testing sets

```
#1. Split the Cleaned Dataframe into Train and Test Dataset

from sklearn.model_selection import train_test_split

# Split the data into features (X) and target (y)
X = data_cleaned.drop('Crash_Label', axis=1)  # Features (independent
variables)
y = data_cleaned['Crash_Label']               # Target (dependent
variable)

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Show the split data
print("Training Features:\n", X_train)
print("Test Features:\n", X_test)
print("Training Labels:\n", y_train)
print("Test Labels:\n", y_test)

Training Features:
         CPU_Usage  Memory_Usage  Battery_Level  Temperature
Disk_Space  \
7460       62.61         67.65          83.87        45.24       11.55

5969       83.39         56.19           9.39        45.54      267.54

8902       71.30         80.43          27.70        41.75       13.93

7952       75.96         47.94          89.18        45.53      499.90
```

```
6482        36.76         75.60          53.72        29.45      426.24

...           ...           ...            ...          ...         ...

11284       51.36         76.06           8.31        19.41      278.57

5191        36.18         52.27          15.96        41.58      417.33

5390        47.20         46.48          10.51        33.75       44.81

860         32.91         47.70          89.25        28.82      436.78

7270        36.89         70.08          26.74        28.89       62.50


       Num_App_Crashes  App_Usage_Level
7460                 1                2
5969                 2                0
8902                 1                1
7952                 1                1
6482                 1                1
...                ...              ...
11284                4                2
5191                 1                2
5390                 1                2
860                  0                1
7270                 0                2

[9238 rows x 7 columns]
Test Features:
        CPU_Usage  Memory_Usage  Battery_Level  Temperature
Disk_Space  \
10028       55.11         54.82          15.02        37.32      474.36

7541        46.90         49.80          74.82        32.99      440.97

8525        69.81         60.04          28.59        40.25      136.59

2701        71.51         51.06           9.88        41.70      231.64

3315        53.55         71.93          53.57        30.42      295.60

...           ...           ...            ...          ...         ...

6213        64.87         63.62          56.71        39.27      257.76

7885        81.98         75.03          41.09        42.10      177.57

3984        56.96         60.50          74.62        34.38      475.08
```

```
3995       74.33         59.76          27.92        42.20      307.89

10395      78.76         47.91           9.95        54.58      386.51


       Num_App_Crashes  App_Usage_Level
10028                3                1
7541                 1                2
8525                 2                2
2701                 2                1
3315                 2                1
...                ...              ...
6213                 1                0
7885                 2                1
3984                 2                0
3995                 1                1
10395                4                0

[2310 rows x 7 columns]
Training Labels:
 7460     1
5969     1
8902     1
7952     1
6482     1
        ..
11284    1
5191     1
5390     1
860      1
7270     2
Name: Crash_Label, Length: 9238, dtype: int64
Test Labels:
 10028    1
7541     1
8525     3
2701     1
3315     2
        ..
6213     2
7885     1
3984     1
3995     3
10395    0
Name: Crash_Label, Length: 2310, dtype: int64
```

1.2 Implement a decision tree classifier using scikit-learn's DecisionTreeClassifier class.

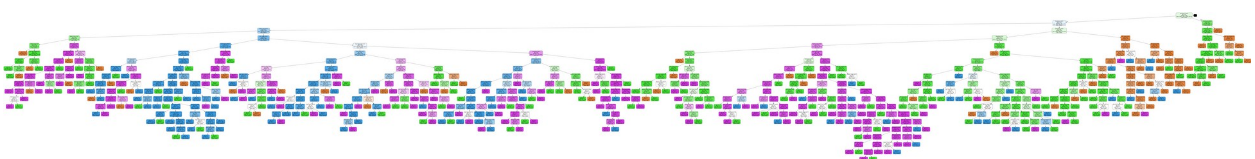
1.3 Train the decision tree classifier on the training dataset.

```
##2. Implement a decision tree classifier using scikit-learn's
DecisionTreeClassifier class.
##3. Train the decision tree classifier on the training dataset.

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Create the DecisionTreeClassifier model
untuned_model = DecisionTreeClassifier(random_state=42)

# Train the model using the training data
untuned_model.fit(X_train, y_train)

print("Decision Tree Model Classified")
```

```
Decision Tree Model Classified
```

```
import os

# import the necessary libraries
from io import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

# Method to plot the graph
def plot_Tree_graphviz(data_cleaned, model):

    # Feature names (replace with your own names)

    feature_names = ['CPU_Usage', 'Memory_Usage', 'Battery_Level',
'Temperature','Disk_Space','Num_App_Crashes','App_Usage_Level']

    # Class names (replace with your own class labels)
    target_names = ['Crash_Label']

    # extact the class names
    class_int = data_cleaned['Crash_Label'].unique().tolist()

    print(class_int)
    # 1, 0 , 3, 2
    class_names = ['High', 'Critical', 'Medium', 'Low']

    # # map the class names to the class number as specified in the
dataset
```

```
    # dictionary = dict(zip(class_names, class_int))

    # print(dictionary)

    # Vizualize the tree

    dot_data = StringIO()

    export_graphviz(model, out_file=dot_data,
                    filled=True, rounded=True,
                    feature_names = feature_names,
                    class_names = class_names,
                    proportion = False, precision = 2,
                    special_characters=True)

    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph

graph= plot_Tree_graphviz(data_cleaned, untuned_model)
Image(graph.create_png())

[1, 0, 3, 2]
```



Observation From Graph Above

- a. Root Node:
  - Top most node used for Split is Disk_Space. It shows the most important feature is Disk_Space.
  - Split is based on Disk_space <= 400.01.
  - Gini Index of 0.72 at root node means it is having high Impurity meaning unevenly distributed.
  - Here the Class = Critical, it means it has the Majority.
- a. Internal/Leaf Nodes:
  - On left Side, if condition is True, we find the next important feature and have a split.
  - On right Side, if condition is False, we find the next split for the same feature and repeat the process.
  - Some of the internal nodes with gini close to 0 means it is towards more purity
  - App_Usage_Level is a feature deep inside the tree, representing very localized influence.
  - gini=0 at the leaf noe for num_app_crashes shows perfect separation.

Image Link - https://drive.google.com/file/d/13-LSYEu2aE1ZHMXzC8ZbM7Pcjx0O7kPv/view?usp=share_link

2. Model Evaluation

- Evaluate the performance of the trained decision tree classifier using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrix. Interpret the findings from each of the evaluation metrics.
- Visualize the decision tree using graphviz or any other suitable visualization tool/library.##

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz,
plot_tree
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
from IPython.display import display, HTML

def generate_metrics(model,X_test,y_test):
    # Predict the target values using the test data
    y_pred = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy: {accuracy:.2f}')

    # Detailed classification report
    print("Classification Report:\n", classification_report(y_test,
y_pred))

    # Confusion Matrix
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    return accuracy

generate_metrics(untuned_model,X_test,y_test)
```

```
Accuracy: 0.96
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.96      0.96       345
           1       0.97      0.96      0.96       877
           2       0.96      0.97      0.97       635
           3       0.92      0.94      0.93       453

    accuracy                           0.96      2310
   macro avg       0.95      0.96      0.96      2310
weighted avg       0.96      0.96      0.96      2310

Confusion Matrix:
 [[331   7   4   3]
 [  8 839   7  23]
```

```
 [   1   10 614   10]
 [   3   11   12 427]]

0.9571428571428572
```

Observation -

1. Overall accuracy of the Model is 96% which is good.
2. High Precision & Recall with overall high F1-Score.
3. Class 1 (Crash Level High) has the highest Performance with 97 Precision and 96 Recall.
4. Class Imbalance - Since Class 1(Crash Level **High**) has higher Support of 877 compare to Class 0 (**Crit**) (345) & Class 3 (**Med**)(453) but close to Class 2(**Low**) (635)

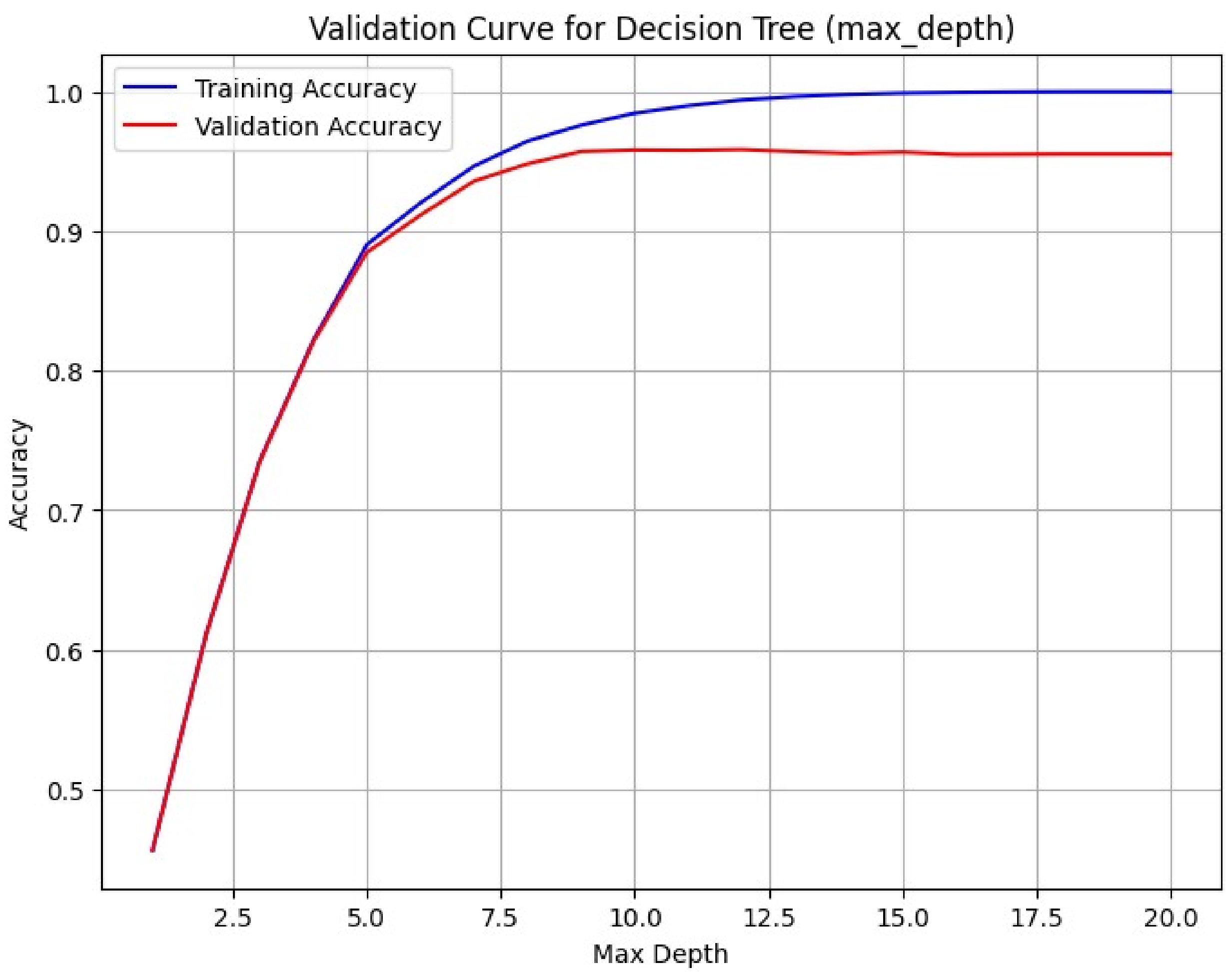
3. Hyperparameter Tuning

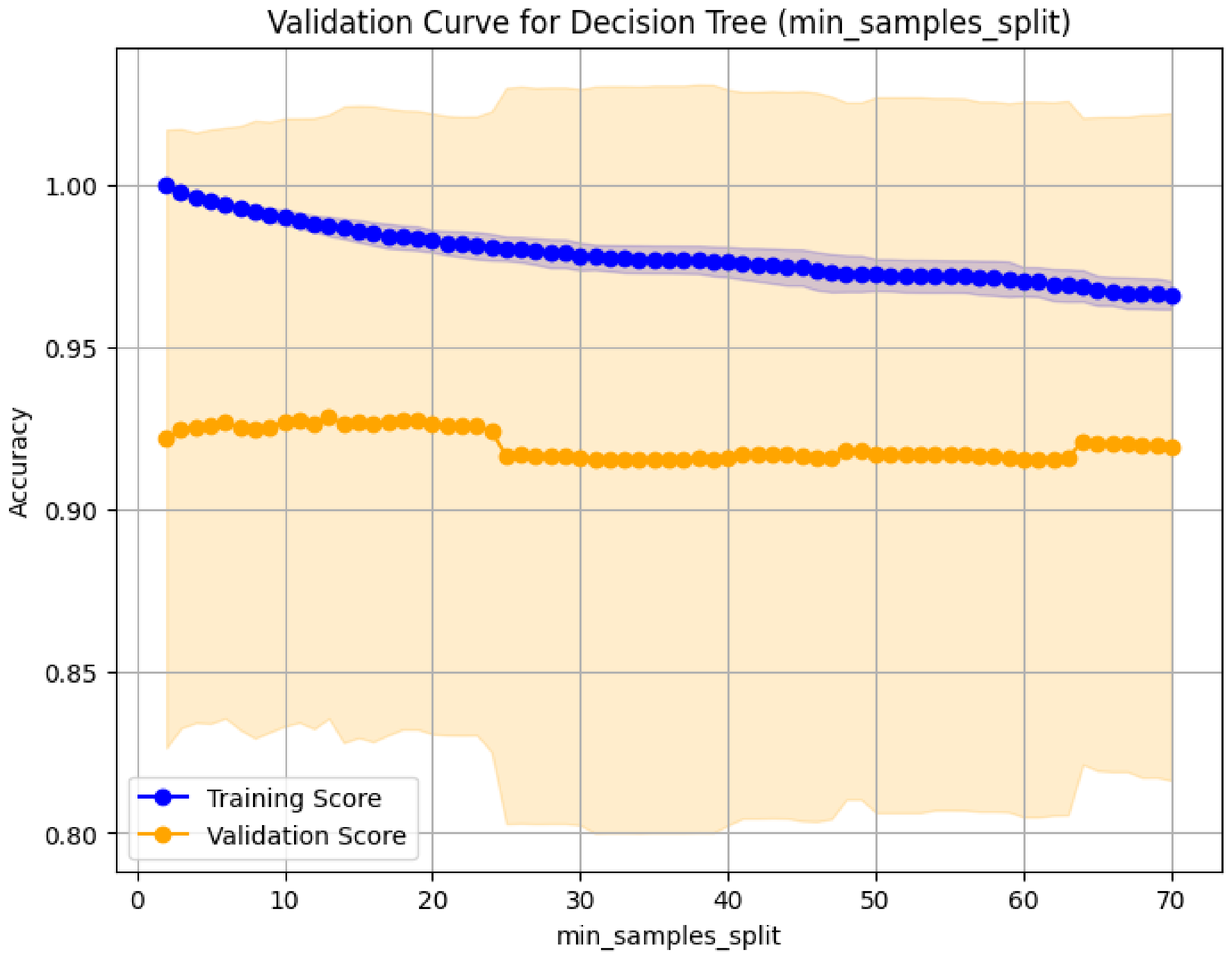
- Explore different hyperparameters of the decision tree classifier (e.g., max_depth, min_samples_split, min_samples_leaf, etc.).
- Evaluate the performance of the tuned model and compare it with the untuned model.

```
#### Validation Curve - Validation curves help visualize the model's
performance as you vary a particular hyperparameter
#### (e.g., max_depth, min_samples_split, etc.).
#### This can help you understand how sensitive the model is to
changes in hyperparameters and whether you are overfitting or
underfitting.

#### MAX Depth Plot
from sklearn.model_selection import validation_curve

param_range = np.arange(1, 21)
train_scores, test_scores = validation_curve(untuned_model,
                                             X_train, y_train,
param_name="max_depth",
                                             param_range=param_range,
cv=5, scoring="accuracy")

plt.figure(figsize=(8, 6))
plt.plot(param_range, np.mean(train_scores, axis=1), label="Training
Accuracy", color='blue')
plt.plot(param_range, np.mean(test_scores, axis=1), label="Validation
Accuracy", color='red')
plt.title("Validation Curve for Decision Tree (max_depth)")
plt.xlabel("Max Depth")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)
plt.show()

#### Min Sample Split Plot

param_range = np.arange(2, 71, 1)
```

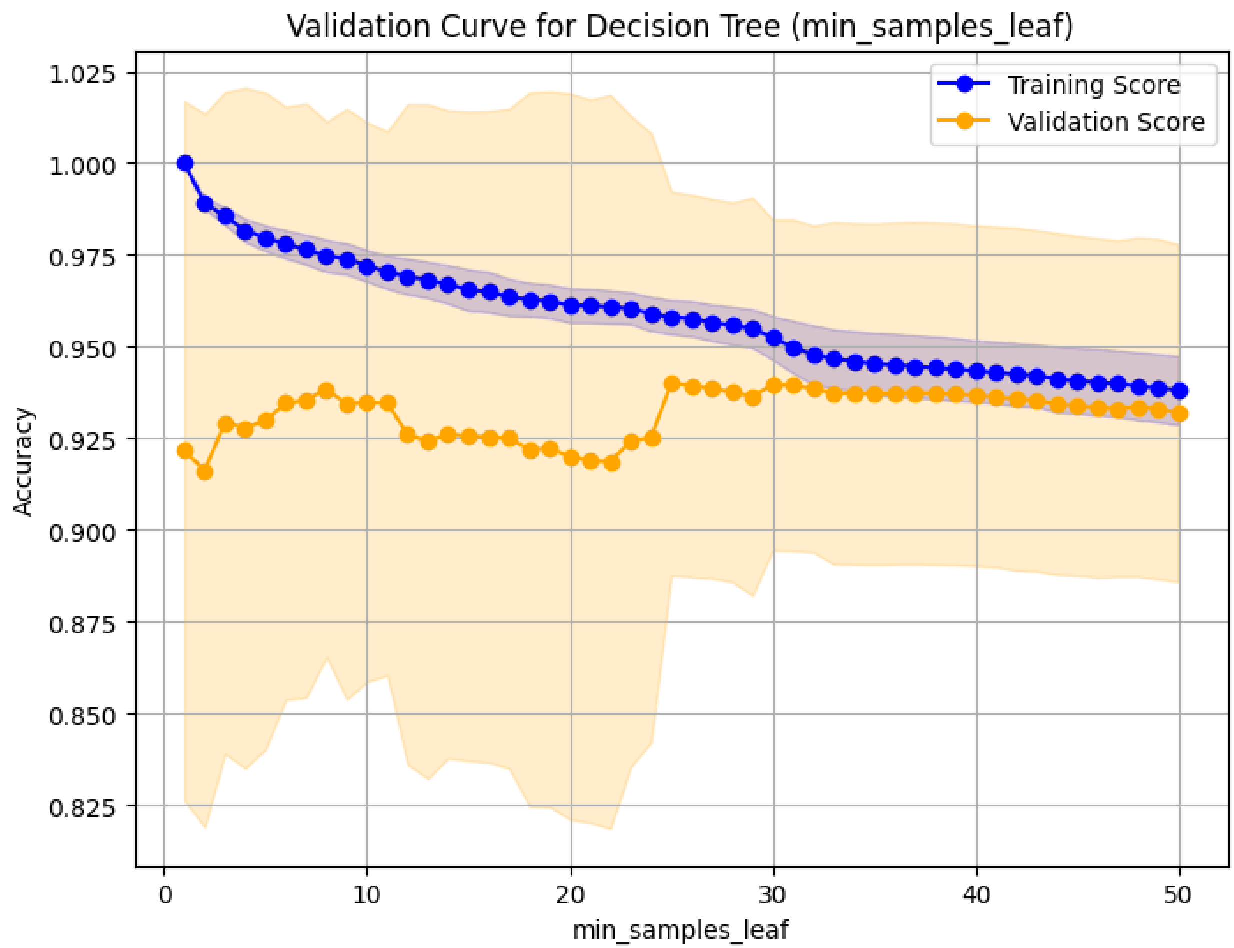
```
train_scores, val_scores = validation_curve(
    untuned_model, X, y,
    param_name="min_samples_split",
    param_range=param_range,
    cv=5,  # 5-fold cross-validation
    scoring="accuracy"  # Metric to evaluate
)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
val_mean = np.mean(val_scores, axis=1)
val_std = np.std(val_scores, axis=1)

plt.figure(figsize=(8, 6))
plt.plot(param_range, train_mean, label="Training Score",
color="blue", marker='o')
plt.fill_between(param_range, train_mean - train_std, train_mean +
train_std, color="blue", alpha=0.2)

plt.plot(param_range, val_mean, label="Validation Score",
color="orange", marker='o')
plt.fill_between(param_range, val_mean - val_std, val_mean + val_std,
color="orange", alpha=0.2)

plt.title("Validation Curve for Decision Tree (min_samples_split)")
plt.xlabel("min_samples_split")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid()
plt.show()

#### Min Sample leaf Plot


param_range = np.arange(1, 51)  # Evaluate min_samples_leaf from 1 to
20
train_scores, val_scores = validation_curve(
    untuned_model, X, y,
    param_name="min_samples_leaf",
    param_range=param_range,
    cv=5,  # 5-fold cross-validation
    scoring="accuracy"
)

train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
val_mean = np.mean(val_scores, axis=1)
val_std = np.std(val_scores, axis=1)
```

```
plt.figure(figsize=(8, 6))
plt.plot(param_range, train_mean, label="Training Score",
color="blue", marker='o')
plt.fill_between(param_range, train_mean - train_std, train_mean +
train_std, color="blue", alpha=0.2)

plt.plot(param_range, val_mean, label="Validation Score",
color="orange", marker='o')
plt.fill_between(param_range, val_mean - val_std, val_mean + val_std,
color="orange", alpha=0.2)

plt.title("Validation Curve for Decision Tree (min_samples_leaf)")
plt.xlabel("min_samples_leaf")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid()
plt.show()
```





Validation Curve for Decision Tree (min_samples_split)
Accuracy
1.00
0.95
0.90
0.85
0.80
0
10
20
30
40
50
60
70
min_samples_split
Training Score
Validation Score



Observation

Max Depth

- max_depth * parameter controls the maximum depth of tree. By limiting the depth, we constrain how complex the tree can become
- for low values, the tree is shallow, capturing only high-level patterns in the data. Underfitting, as the model cannot learn enough about the data. that;s why accuracy is low.
- for higher values, the tree becomes deep, Overfitting, as the tree memorizes the training data instead of generalizing. that;s why accuracy do not improve or little bit drop can be seen as well.
- optimal value around 8-10 where validation accuracy is highest and gap between training and validation score is low. Represents a balanced tree, capturing meaningful patterns without overfitting.

Min Sample Split

- min_samples_split * parameter controls the minimum number of samples required to split an internal node.
- for low values both training and validation accuracy are good. so it is not underfitting
- for higher values of parameters accuracy do not drop. so it is not overfitting as well.

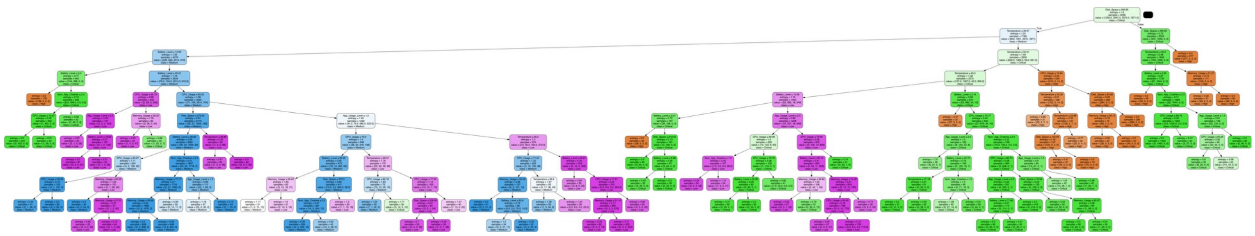
- optimal value around 15-20 where validation accuracy is highest and gap between training and validation score is low.

Min Sample Leaf

- min_samples_leaf * parameter specifies the minimum number of samples required to be in a leaf node
- lower values depth increase. indicate overfitting, training accuracy good but validation accuracy low.
- higher values indicate underfitting. both trainig and validation accuracies are dropping.
- optimal value around 30 when validation accuracy is good and gap between training and validation score is low.

```
### Changing Max Depth to 10 to limit the Tree Depth,
min_samples_leaf=30, min_samples_split=20  and test against test_set

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Create the DecisionTreeClassifier model
tuned_model = DecisionTreeClassifier(max_depth=10,min_samples_leaf=30,
min_samples_split=20, criterion= 'entropy', random_state=42)

# Train the model using the training data
tuned_model.fit(X_train, y_train)

# Predict the target values using the test data
y_pred = tuned_model.predict(X_test)


generate_metrics(tuned_model,X_test,y_test)

graph=plot_Tree_graphviz(data_cleaned,tuned_model)
Image(graph.create_png())


Accuracy: 0.94
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.85      0.91       345
           1       0.95      0.96      0.96       877
           2       0.93      0.97      0.95       635
           3       0.92      0.94      0.93       453

    accuracy                           0.94      2310
   macro avg       0.94      0.93      0.93      2310
weighted avg       0.94      0.94      0.94      2310

Confusion Matrix:
 [[293  29  15   8]
```

```
 [  6 845   6  20]
 [  2   8 614  11]
 [  1   5  23 424]]
[1, 0, 3, 2]
```



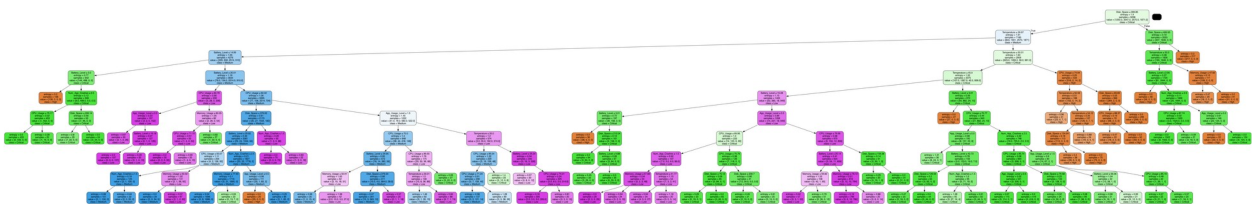
Observation : With this tuning Tree is now least complex but Accuracy has reduced to .94 from .96 and recall for Class 0 (Critical) got a big impact and come down to .85 from .96, F1Score is also reduced by .05 for class 0.

Image Link - https://drive.google.com/file/d/13-LSYEu2aE1ZHMXzC8ZbM7Pcjx0O7kPv/view?usp=share_link

Further Tuning

```
### Further Tuning with min_samples_leaf=20, min_samples_split=20  and test against test_set

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Create the DecisionTreeClassifier model
tuned_model_final =
DecisionTreeClassifier(max_depth=9,min_samples_leaf=20,
min_samples_split=20, criterion= 'entropy', random_state=42)

# Train the model using the training data
tuned_model_final.fit(X_train, y_train)

# Predict the target values using the test data
y_pred = tuned_model_final.predict(X_test)


generate_metrics(tuned_model_final,X_test,y_test)

graph=plot_Tree_graphviz(data_cleaned,tuned_model_final)
Image(graph.create_png())

Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.90      0.93       345
```

```
           1       0.96      0.97      0.96       877
           2       0.95      0.97      0.96       635
           3       0.93      0.93      0.93       453

    accuracy                           0.95      2310
   macro avg       0.95      0.94      0.95      2310
weighted avg       0.95      0.95      0.95      2310

Confusion Matrix:
 [[310  19  10   6]
 [  6 851   2  18]
 [  2   9 616   8]
 [  1   8  23 421]]
[1, 0, 3, 2]
```

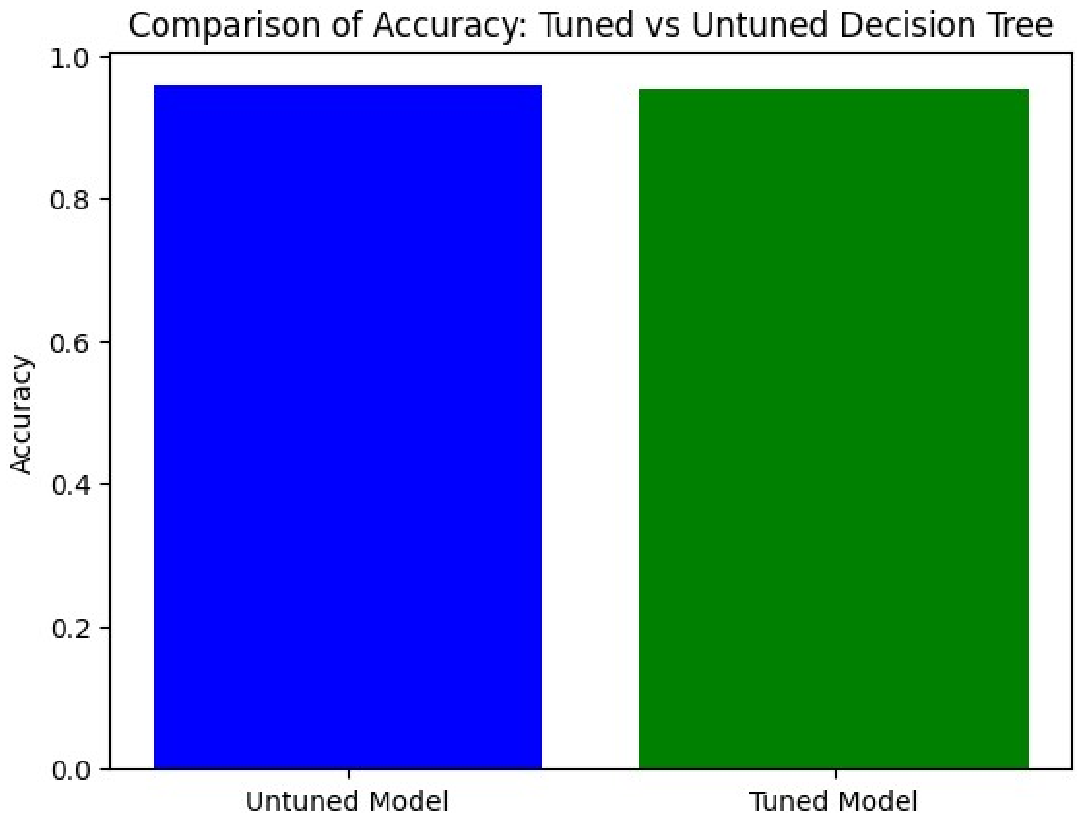


Observation : With this tuning tree is now less complex but Accuracy has increased to .95 from .94 and recall for Class 0 (Critical) improved from .85 to .90 and F1 Score is improved to .93.

Image Link - #### Image Link - https://drive.google.com/file/d/13-LSYEu2aE1ZHMXzC8ZbM7Pcjx0O7kPv/view?usp=share_link

```
#### Evaluate the performance of the tuned model and compare it with
the untuned model.

accuracy_untuned=generate_metrics(untuned_model,X_test,y_test)

print(accuracy_untuned)

accuracy_tuned=generate_metrics(tuned_model_final,X_test,y_test)

print(accuracy_tuned)

# 5. Plot the comparison of accuracy
models = ['Untuned Model', 'Tuned Model']
accuracies = [accuracy_untuned, accuracy_tuned]

plt.bar(models, accuracies, color=['blue', 'green'])
plt.ylabel('Accuracy')
plt.title('Comparison of Accuracy: Tuned vs Untuned Decision Tree')
plt.show()

Accuracy: 0.96
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       345
           1       0.97      0.96      0.96       877
           2       0.96      0.97      0.97       635
           3       0.92      0.94      0.93       453

    accuracy                           0.96      2310
   macro avg       0.95      0.96      0.96      2310
weighted avg       0.96      0.96      0.96      2310

Confusion Matrix:
 [[331   7   4   3]
 [  8 839   7  23]
 [  1  10 614  10]
 [  3  11  12 427]]
0.9571428571428572
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.90      0.93       345
           1       0.96      0.97      0.96       877
           2       0.95      0.97      0.96       635
           3       0.93      0.93      0.93       453

    accuracy                           0.95      2310
   macro avg       0.95      0.94      0.95      2310
weighted avg       0.95      0.95      0.95      2310

Confusion Matrix:
 [[310  19  10   6]
 [  6 851   2  18]
 [  2   9 616   8]
 [  1   8  23 421]]
0.9515151515151515
```



Observation : With only .05 reduction in accuracy, we could make the tree simpler.

######Tuned Model Parameters :

- max_depth=9
- min_samples_leaf=20
- min_samples_split=20
- criterion= 'entropy'
- random_state=42