In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
```

In [ ]:

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving SP_Preprocessed.csv to SP_Preprocessed (1).csv
```

In [ ]:

```
import pandas as pd
import io

df = pd.read_csv(io.BytesIO(uploaded['SP_Preprocessed.csv']))
df.set_index('Model', inplace=True)
```

In [ ]:

```
df.shape
```

Out[ ]:

```
(1167, 50)
```

In [ ]:

```
df1 = df.drop(['Operating System', 'Brands', 'Processor', 'Display Type', 'Battery_Type'
#df1 = df.drop(['Operating System', 'Brands', 'Processor', 'Display Type', 'Battery_Type
```

In [ ]:

```
df['Processor'] = df['Processor'].fillna(df['Processor'].value_counts().index[0])
df['Battery_Type'] = df['Battery_Type'].fillna(df['Battery_Type'].value_counts().index[0
```

In [ ]:

```
def price_range(value):
    if value < 10000:
        return 0
    if 10000 <= value < 20000:
        return 1
    elif 20000 <= value < 30000:
        return 2
    elif value >= 30000:
        return 3
df1['Price Range'] = df1['Price'].map(price_range)
```

In [ ]:

```
def price_range(value):
    if value < 10000:
        return "<10K"
    if 10000 <= value < 20000:
        return "10K-20K"
    elif 20000 <= value < 30000:
        return "20K-30K"
    elif value >= 30000:
        return ">30K"
df1['Price Range'] = df1['Price'].map(price_range)
```

In [ ]:

```
X = df1.drop(['Price', 'Price Range'],axis=1)
y = df1['Price Range']
#X = df_scaled.drop(['Price', 'Price Range'],axis=1)
#y = df_scaled['Price Range']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=32
```

Loading [MathJax]/extensions/Safe.js

Logistic Regression

In [ ]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold, GridSearchCV, StratifiedKFold
import warnings
warnings.filterwarnings('ignore')
log = LogisticRegression()
log.get_params()
```

Out[ ]:

```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [ ]:

```
#C = [1,2,4]
#fit_intercept = [True, False]
max_iter = [100, 150, 200, 250]
n_jobs = [-1]
#solver = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
param_grid = {'n_jobs':n_jobs
             }
grid = GridSearchCV(estimator=log,
                    param_grid=param_grid,
                    cv = 10)
grid.fit(X_train, y_train)
from sklearn.metrics import accuracy_score, f1_score
print("Best Parameters", grid.best_params_)
print("Best Parameters", grid.best_score_)
```

```
Best Parameters {'n_jobs': -1}
Best Parameters 0.706280027453672
```

In [ ]:

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
pred = grid.predict(X_test)
accuracy = accuracy_score(y_test, pred)
f1 = f1_score(y_test, pred, average='weighted')
precision = precision_score(y_test, pred, average='weighted')
recall = recall_score(y_test, pred, average='weighted')
print('Accuracy: ', accuracy)
print('F1 Score: ', f1)
print('Precision: ', precision)
print('Recall: ', recall)
```

```
Accuracy:  0.7094017094017094
F1 Score:  0.6557369705928464
Precision:  0.6189668174962293
Recall:  0.7094017094017094
```

RANDOM FOREST CLASSIFICATION

Loading [MathJax]/extensions/Safe.js

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
RFC = RandomForestClassifier()
RFC.get_params()
```

Out[ ]:

```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [ ]:

```
param_grid = {'criterion':['gini', 'entropy', 'log_loss'],
              'max_depth':[2, 5, 10, 15, 17, 19, 20, 25],
              'n_jobs':[-1]
             }
grid = GridSearchCV(estimator=RFC,
                    param_grid=param_grid,
                    cv = 10,
                   scoring='accuracy')
grid.fit(X_train, y_train)
print("Best Parameters", grid.best_params_)
print("Best Parameters", grid.best_score_)
```

```
Best Parameters {'criterion': 'gini', 'max_depth': 25, 'n_jobs': -1}
Best Parameters 0.8788606726149621
```

In [ ]:

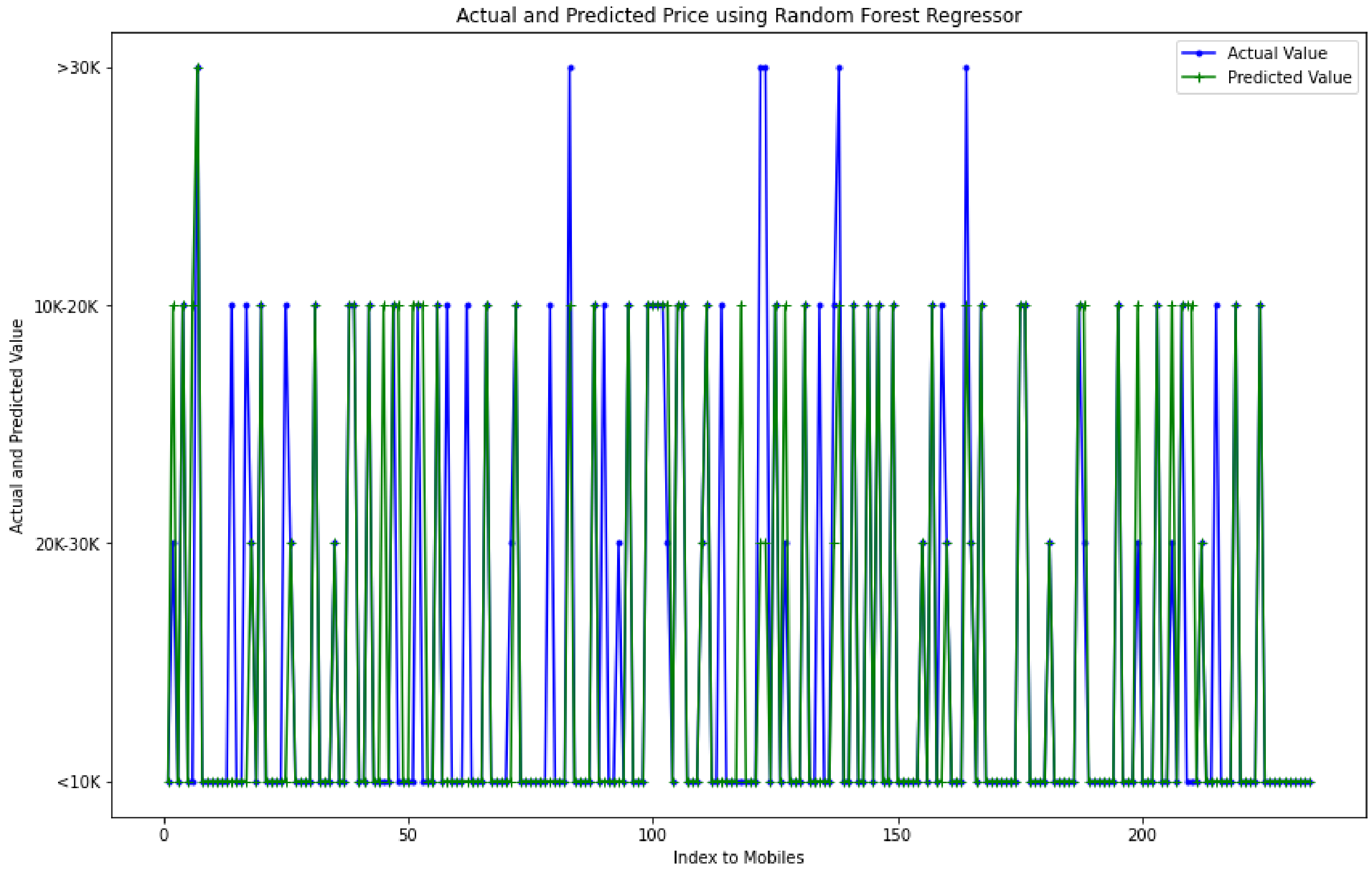
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
pred = grid.predict(X_test)
accuracy = accuracy_score(y_test, pred)
f1 = f1_score(y_test, pred, average='weighted')
precision = precision_score(y_test, pred, average='weighted')
recall = recall_score(y_test, pred, average='weighted')
print('Accuracy: ', accuracy)
print('F1 Score: ', f1)
print('Precision: ', precision)
print('Recall: ', recall)
```

```
Accuracy:  0.8589743589743589
F1 Score:  0.8510360765226423
Precision:  0.8621396510547454
Recall:  0.8589743589743589
```

In [ ]:

```
# multiple lines with legend
plt.figure(figsize=(14, 9))
plt.plot(np.arange(1,235),y_test,marker='.', color='b', label= 'Actual Value')
plt.plot(np.arange(1,235),pred, marker = '+', color = 'g',label = 'Predicted Value')
plt.xlabel("Index to Mobiles")
plt.ylabel("Actual and Predicted Value")
plt.title("Actual and Predicted Price using Random Forest Regressor")
plt.legend();
plt.savefig('Actual and predicted.png')
```

Loading [MathJax]/extensions/Safe.js



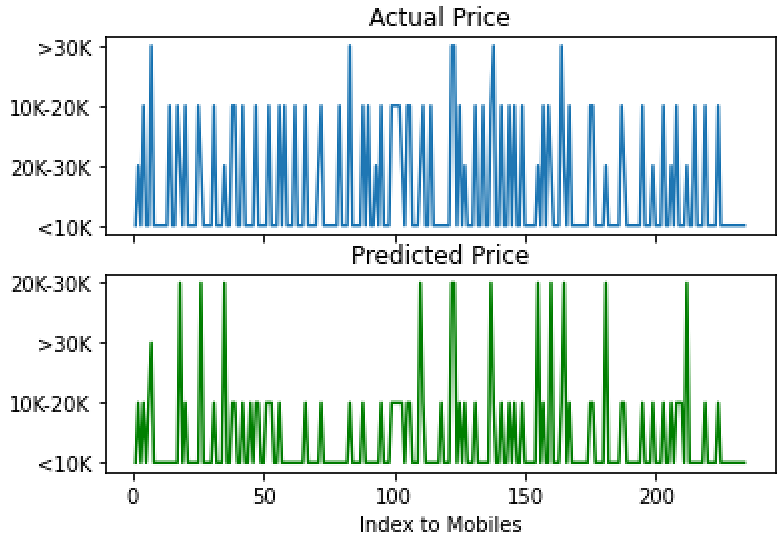
In [ ]:

```
fig = plt.figure()
plt.figure(figsize=(20,20))
fig, (ax1, ax2) = plt.subplots(2, sharex=True)
ax1.plot(np.arange(1,235), y_test)
ax2.plot(np.arange(1,235), pred, 'g')
ax1.set_title("Actual Price")
ax2.set_title("Predicted Price")
plt.xlabel("Index to Mobiles")
```

Out[ ]:

```
Text(0.5, 0, 'Index to Mobiles')

<Figure size 432x288 with 0 Axes>
<Figure size 1440x1440 with 0 Axes>
```



In [ ]:

```
a, b = np.polyfit(y_test, pred, 1)
plt.figure(figsize=(14, 9))
plt.plot(y_test, pred,marker='.', color='g')
plt.plot(pred, a*pred+b, 'r')
plt.xlabel("Actual Price")
```

Loading [MathJax]/extensions/Safe.js

```
plt.ylabel("Predicted Price")
plt.title("Actual and Predicted Price")
plt.savefig('Predicted bs actual.png')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-6b8f21ecdf89> in <module>
----> 1 a, b = np.polyfit(y_test, pred, 1)
      2 plt.figure(figsize=(14, 9))
      3 plt.plot(y_test, pred,marker='.', color='g')
      4 plt.plot(pred, a*pred+b, 'r')
      5 plt.xlabel("Actual Price")

<__array_function__ internals> in polyfit(*args, **kwargs)

/usr/local/lib/python3.8/dist-packages/numpy/lib/polynomial.py in polyfit(x, y, deg, rco
nd, full, w, cov)
    619     """
    620     order = int(deg) + 1
--> 621     x = NX.asarray(x) + 0.0
    622     y = NX.asarray(y) + 0.0
    623

TypeError: can only concatenate str (not "float") to str
```

Best Estimator

In [ ]:

```
print("Best Estimator: ", grid.best_estimator_.base_estimator_)
print("Estimators:")
print("")
grid.best_estimator_.estimators_
```

```
Best Estimator:  DecisionTreeClassifier()
Estimators:
```

```
Out[ ]: [DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=781229225),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=193507108),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1391260674),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1044610643),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1526106243),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=2120830365),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1636038560),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1369262850),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=285681324),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=104060200),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=652240142),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1078131820),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1180522936),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1772002472),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1253924327),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1871244038),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=141732984),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1310771820),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=2059421358),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1149097427),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=650448607),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=940800616),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1788593757),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=925886605),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1982642785),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1478999362),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1704304610),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=431581412),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=949359604),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=1064191589),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=214785013),
         DecisionTreeClassifier(max_depth=25, max_features='auto',
                                random_state=872648769),
```

Loading [MathJax]/extensions/Safe.js

```
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=695050107),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=134461120),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1926584493),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=917399301),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1513691207),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1605276715),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=841852114),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1138780953),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1655109054),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=2009102849),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=515017013),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=834421824),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1981001608),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1135876080),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1454074373),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1259665054),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1364978834),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1243671705),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=392974495),
              DecisionTreeClassifier(max_depth=25, max_features='auto', random_state=88970119),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=597377627),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=677654255),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1361611634),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=775952768),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=807513257),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1364733167),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1193804674),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1571136673),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1292581171),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1303163210),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1129521724),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=2125143552),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1153775969),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1720356973),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1678751668),
              DecisionTreeClassifier(max_depth=25, max_features='auto', random_state=23013585),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1748914576),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=356434378),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1531695621),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=731331753),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=2118924672),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=341773723),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1064931382),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=462025643),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1309877019),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1414733353),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1542345691),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=713736900),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=2012107845),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=988972859),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=2011281716),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1581710790),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=334675004),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=2011824824),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1774446292),
              DecisionTreeClassifier(max_depth=25, max_features='auto', random_state=2215940),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1940866739),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1930550996),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=235531711),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1774087228),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1679799625),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1968756924),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1618554876),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=474487460),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
                                     random_state=1210676748),
              DecisionTreeClassifier(max_depth=25, max_features='auto',
```

Loading [MathJax]/extensions/Safe.js

```
                               random_state=416375206),
 DecisionTreeClassifier(max_depth=25, max_features='auto',
                               random_state=629593924),
 DecisionTreeClassifier(max_depth=25, max_features='auto',
                               random_state=1656305038)]
```

In [ ]:

```
grid.best_estimator_.n_features_in_
```

Out[ ]:

```
44
```

In [ ]:

```
importance_gb = grid.best_estimator_.feature_importances_
importance_gb
columns = X_train.columns
#Combine columns with feature importances
gbGraph = pd.Series(importance_gb, columns)
gbGraph
```

Out[ ]:

```
Number of Sim Slots                    8.379921e-03
Ratings                                4.994477e-02
Number Of Ratings                      5.928969e-02
Reviews                                5.590301e-02
RAM                                    9.411634e-02
Storage                                1.162341e-01
Expandable Storage                     3.690192e-02
Expandable or Not                      1.393956e-02
Warranty                               3.853411e-03
Front Camera1                          5.662943e-02
Front Camera2                          3.436790e-03
Number of Front Cameras                5.552423e-03
Display_Size                           7.762168e-02
Battery_(mAh)                          6.714140e-02
Rear Camera1                           1.181309e-01
Rear Camera2                           3.809798e-02
Rear Camera3                           4.692729e-02
Rear Camera4                           1.008701e-02
Number Of Rear Cameras                 5.413631e-02
FWVGA Display                          1.133143e-03
WVGA Display                           3.890960e-04
HVGA Display                           1.432494e-04
Normal Display                         7.139797e-03
HD Display                             6.000171e-03
Full HD Display                        7.217336e-03
quarter HD Display                     1.351060e-03
HD+ Display                            1.062501e-02
NA Display                             4.618961e-04
FWVGA+ Display                         0.000000e+00
Full HD+ Display                       1.616870e-02
QVGA Display                           1.011328e-08
Quad HD+ Display                       1.518436e-05
Full HD+ AMOLED Display                2.351927e-03
Full HD+ Super AMOLED Display          2.208817e-04
Retina Display                         5.202772e-04
Retina HD Display                      1.336118e-03
Super Retina XDR Display               3.183374e-03
Full HD+ E3 Super AMOLED Display       3.203777e-04
Liquid Retina HD Display               1.859402e-04
Quad HD Display                        2.016570e-03
5G                                     6.488055e-03
Android                                7.526936e-03
Windows                                9.359888e-04
iOS                                    7.944880e-03
dtype: float64
```

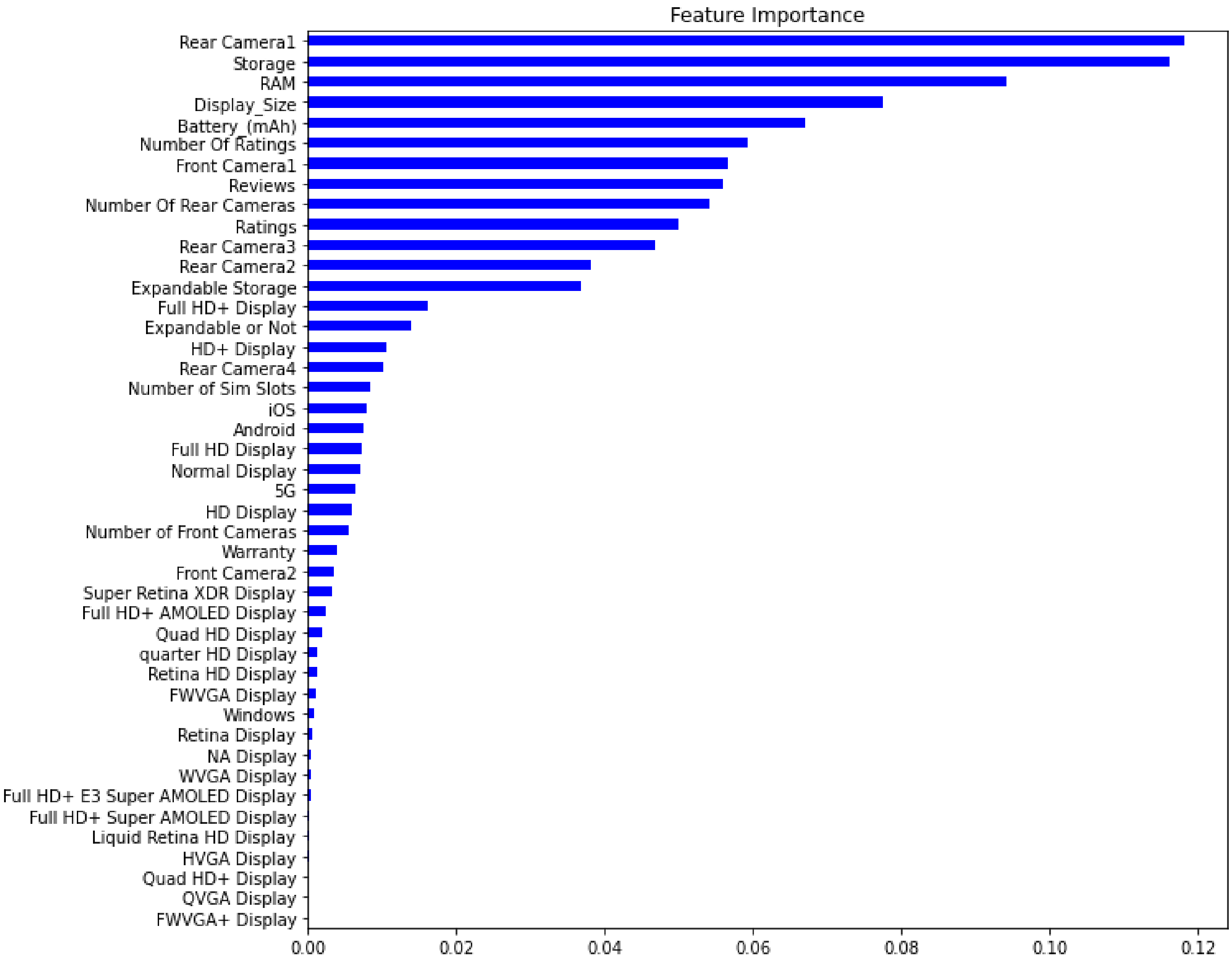
In [ ]:

```
# Visualizing importance from our model

from matplotlib.pyplot import figure
figure(figsize = (10,10))
gbGraph.sort_values().plot.barh(color='b')
plt.title('Feature Importance')
```

Out[ ]:

```
Text(0.5, 1.0, 'Feature Importance')
```



In [ ]:

```
len(grid.best_estimator_.estimators_)
```

Out[ ]:

```
100
```

In [ ]:

```
# first decision tree of the random forest
grid.best_estimator_.estimators_[0]
```

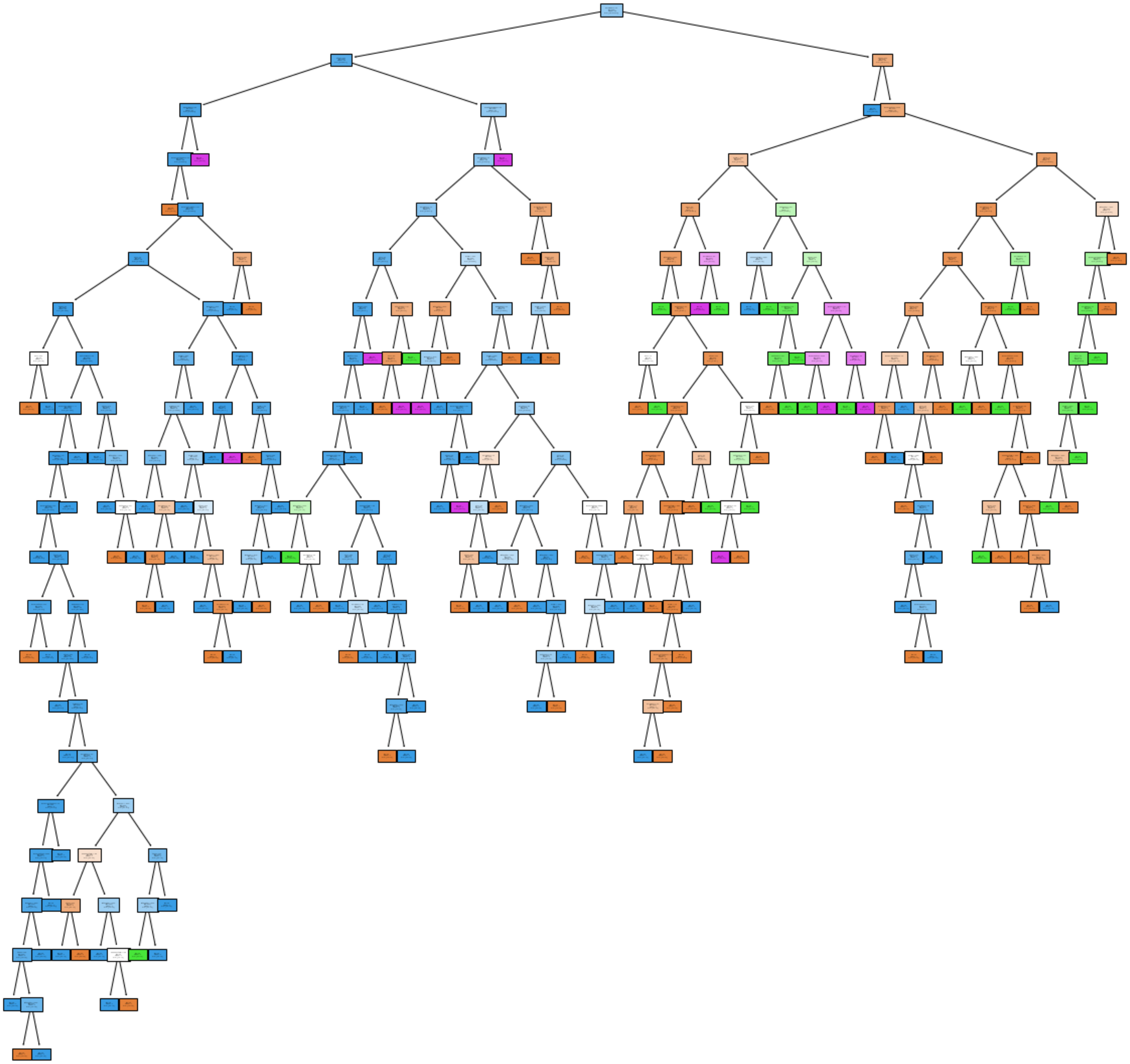
Out[ ]:

```
DecisionTreeClassifier(max_depth=25, max_features='auto',
                       random_state=781229225)
```

In [ ]:

```
#We can plot a first Decision Tree from the Random Forest
from sklearn import tree
plt.figure(figsize=(20,20))
_ = tree.plot_tree(grid.best_estimator_.estimators_[0], feature_names=X_train.columns, f
plt.savefig('First Decision Tree.png')
```

Loading [MathJax]/extensions/Safe.js



In [ ]:

```
# Visualize Decision Tree
from sklearn.tree import export_graphviz

# Creates dot file named tree.dot
export_graphviz(
            tree,
            out_file =  "myTreeName.dot",
            feature_names = list(X.columns),
            class_names = iris.target_names,
            filled = True,
            rounded = True)
```

Loading [MathJax]/extensions/Safe.js