Project Overview

Mobile Phone Price Prediction

Name :Yuvashree M

Internship ID: UMIP19802

Objective: Build a system that predicts mobile phone pricing categories (low, medium, high, very high) based on features like battery power, RAM, processor speed, etc.

Step 1: Import Necessary Libraries Libraries like pandas for data manipulation, sklearn for preprocessing, model building, and evaluation, and matplotlib and seaborn for visualization.

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by yuvashree magesh
20:32 (22 minutes ago)
executed in 4.622 s

Step **ataset**

Load ; structure to understand the features.

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

# Display basic information and summary statistics
print(df.head())
print(df.info())
print(df.describe())

# Check for missing values
print(df.isnull().sum())
```

```
   battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  m_dep  \
0            842     0          2.2         0   1       0           7    0.6
1           1021     1          0.5         1   0       1          53    0.7
2            563     1          0.5         1   2       1          41    0.9
3            615     1          2.5         0   0       0          10    0.8
4           1821     1          1.2         0  13       1          44    0.6

   mobile_wt  n_cores  ...  px_height  px_width   ram  sc_h  sc_w  talk_time  \
0        188        2  ...         20       756  2549     9     7         19
1        136        3  ...        905      1988  2631    17     3          7
2        145        5  ...       1263      1716  2603    11     2          9
3        131        6  ...       1216      1786  2769    16     8         11
4        141        2  ...       1208      1212  1411     8     2         15

   three_g  touch_screen  wifi  price_range
0        0             0     1            1
1        1             1     0            2
2        1             1     0            2
3        1             0     0            2
4        1             1     0            1

[5 rows x 21 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
```

```
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
None
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
```

Step 3: Data Preprocessing Some preprocessing steps may be necessary, like scaling the features or encoding categorical data

```
# Splitting the dataset into features (X) and target (y)
X = df.drop('price_range', axis=1)
y = df['price_range']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Sta                          (scaling)
scale
X_tra                          ansform(X_train)
X_tes                          orm(X_test)
```

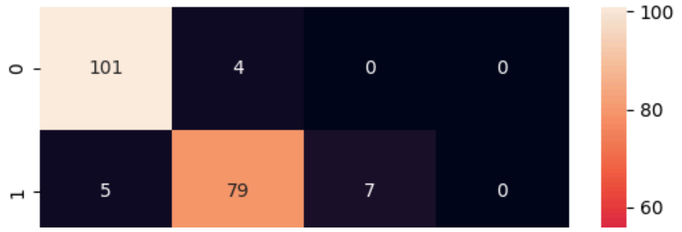
Run cell (Ctrl+Enter)
cell executed since last change

executed by yuvashree magesh
20:32 (22 minutes ago)
executed in 4.622 s

Step 4: Model Building

```
# Initialize and train the Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")

# Display confusion matrix and classification report
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d')
plt.show()

print(classification_report(y_test, y_pred))
```

```
Accuracy: 89.25%
```



Step 5: Hyperparameter Tuning

Further optimize the model using techniques like GridSearchCV for hyperparameter tuning.

```
from sklearn.model_selection import GridSearchCV

# Define a grid of parameters to test
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

# Grid search for the best parameters
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# Best parameters and score
print(f"Best Params: {grid_search.best_params_}")
print                             ·ch.best_score_}")

# Tra
best_                             _estimator_
y_pre                             ict(X_test_scaled)

# Evaluate the tuned model
accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"Tuned Accuracy: {accuracy_best*100:.2f}%")
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by yuvashree magesh
20:32 (22 minutes ago)
executed in 4.622 s

```
Best Params: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 200}
Best Score: 0.878125
Tuned Accuracy: 89.50%
```