# Mobile Price Range Classification (ML Project)

This project predicts the price range (0 to 3) of mobile phones based on their specifications using machine learning models such as Logistic Regression, SVM, and Random Forest. The goal is to build and compare models, tune them using GridSearchCV, and evaluate their accuracy.

#### **About Dataset**

#### Context

In [1]: # For Data handling

- Bob has started his own mobile company. He wants to give tough fight to big companies
  like Apple, Samsung etc. He does not know how to estimate price of mobiles his company
  creates. In this competitive mobile phone market you cannot simply assume things. To
  solve this problem he collects sales data of mobile phones of various companies. Bob wants
  to find out some relation between features of a mobile phone(eg:- RAM, Internal Memory
  etc) and its selling price. But he is not so good at Machine Learning. So he needs your help
  to solve this problem.
- In this problem you do not have to predict actual price but a price range indicating how high the price is

### Step 0: Import All Required Libraries

```
import pandas as pd
import numpy as np

# For Visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

In [2]: # For Machine Learning Models
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_macimport warnings
warnings.filterwarnings("ignore")
```

Step 1: Load and Understand the Dataset

#### Load the Data

	<pre>Train_df = pd.read_csv("train.csv") Test_df = pd.read_csv("test.csv")</pre>									
[4]:	Train_df.head()									
]:	battery	_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt
	0	842	0	2.2	0	1	0	7	0.6	188
	1	1021	1	0.5	1	0	1	53	0.7	136
	2	563	1	0.5	1	2	1	41	0.9	145
	3	615	1	2.5	0	0	0	10	0.8	131
	4	1821	1	1.2	0	13	1	44	0.6	141
5	5 rows × 21	columns	5							

#### • Understand the Structure

```
In [5]: print("Train Shape:", Train_df.shape)
    print("Test Shape:", Test_df.shape)
```

Train Shape: (2000, 21) Test Shape: (1000, 21)

## • Check Column Info & Missing Data

```
In [6]: Train_df.info()
print("\nMissing values in train:\n", Train_df.isnull().sum())
```

```
<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
       dual_sim 2000 non-null
 3
                                                       int64
      fc 2000 non-null
four_g 2000 non-null
int_memory 2000 non-null
mobile_wt 2000 non-null
n_cores 2000 non-null
pc 2000 non-null
2000 non-null
2000 non-null
2000 non-null
 4
                                                       int64
 5
                                                       int64
 6
                                                       int64
 7
                               2000 non-null float64
 8
                                                       int64
 9
                                                       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
 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: 328.3 KB
```

Missing values in train:

```
battery power
                 0
blue
                 0
clock_speed
                 0
dual sim
                 0
                 0
fc
four q
                 0
int memory
                 0
m_dep
mobile wt
                 0
                 0
n cores
                 0
рс
px height
                 0
                 0
px_width
                 0
ram
sc h
                 0
                 0
SC W
talk time
                 0
three_g
                 0
                 0
touch screen
wifi
                 0
price range
dtype: int64
```

```
In [7]: Test_df.info()
    print("\nMissing values in test:\n", Test_df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
                   Non-Null Count Dtype
    Column
- - -
    -----
                   -----
                                   ----
0
                   1000 non-null
                                   int64
    id
 1
    battery_power 1000 non-null
                                   int64
 2
    blue
                   1000 non-null
                                   int64
 3
                   1000 non-null
                                   float64
    clock_speed
 4
    dual sim
                   1000 non-null
                                   int64
 5
    fc
                   1000 non-null
                                   int64
 6
    four g
                   1000 non-null
                                   int64
 7
    int_memory
                   1000 non-null
                                   int64
 8
    m dep
                   1000 non-null
                                   float64
 9
    mobile wt
                   1000 non-null
                                   int64
 10
                   1000 non-null
    n cores
                                   int64
 11
    рс
                   1000 non-null
                                   int64
 12
    px height
                   1000 non-null
                                   int64
 13
    px width
                   1000 non-null
                                   int64
 14 ram
                   1000 non-null
                                   int64
 15 sc h
                   1000 non-null
                                   int64
 16 sc w
                   1000 non-null
                                   int64
 17 talk_time
                   1000 non-null
                                   int64
 18 three q
                   1000 non-null
                                   int64
 19 touch_screen 1000 non-null
                                   int64
 20 wifi
                   1000 non-null
                                   int64
dtypes: float64(2), int64(19)
memory usage: 164.2 KB
Missing values in test:
id
                 0
battery_power
                0
blue
                0
                0
clock_speed
                0
dual sim
fc
                0
                0
four g
int_memory
                0
                0
m dep
                0
mobile wt
                0
n cores
рс
                0
                0
px height
                0
px width
                0
ram
sc h
                0
SC W
                0
talk_time
                0
                0
three g
touch_screen
                0
wifi
                0
dtype: int64
```

#### • Check the Target Column(Price Range)

```
Out[8]: price_range

1 500

2 500

3 500

0 500

Name: count, dtype: int64
```

Step 2: EDA (Exploratory Data Analysis)

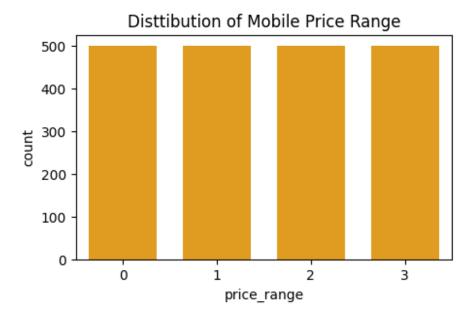
Our goal here: Understand relationships between features and the target (price\_range) using plots & stats.

#### • View Summary Stats

9]:	Train_df.describe()								
:		battery_power	blue	clock_speed	dual_sim	fc	four_g	int_r	
	count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000	2000.000000	2000	
	mean	1238.518500	0.4950	1.522250	0.509500	4.309500	0.521500	32	
	std	439.418206	0.5001	0.816004	0.500035	4.341444	0.499662	18	
	min	501.000000	0.0000	0.500000	0.000000	0.000000	0.000000	2	
	25%	851.750000	0.0000	0.700000	0.000000	1.000000	0.000000	16	
	50%	1226.000000	0.0000	1.500000	1.000000	3.000000	1.000000	32	
	75%	1615.250000	1.0000	2.200000	1.000000	7.000000	1.000000	48	
	max	1998.000000	1.0000	3.000000	1.000000	19.000000	1.000000	64	
	8 rows × 21 columns								

# Visualize Target Distribution

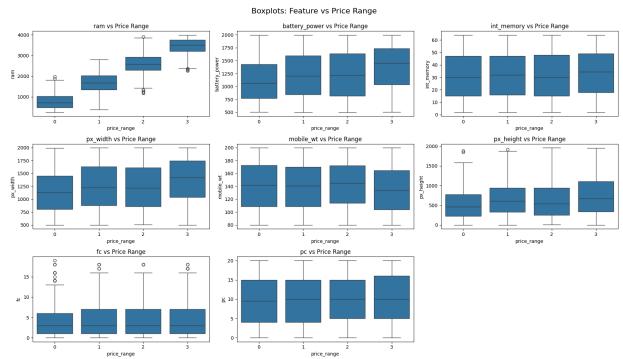
```
In [10]: plt.figure(figsize=(5,3))
    sns.countplot(x="price_range", data=Train_df, color="orange", gap=0.1)
    plt.title("Disttibution of Mobile Price Range")
    plt.show()
```



#### • Feature vs Price Range (Box Plots)

```
In [11]: features = ['ram', 'battery_power', 'int_memory', 'px_width', 'mobile_wt', 'px_
    plt.figure(figsize=(18,10))
    for i,feat in enumerate(features):
        plt.subplot(3, 3 , i+1)
        sns.boxplot(x='price_range', y=feat, data=Train_df)
        plt.title(f"{feat} vs Price Range")
        plt.tight_layout()

plt.suptitle("Boxplots: Feature vs Price Range", fontsize=16, y=1.03)
    plt.show()
```

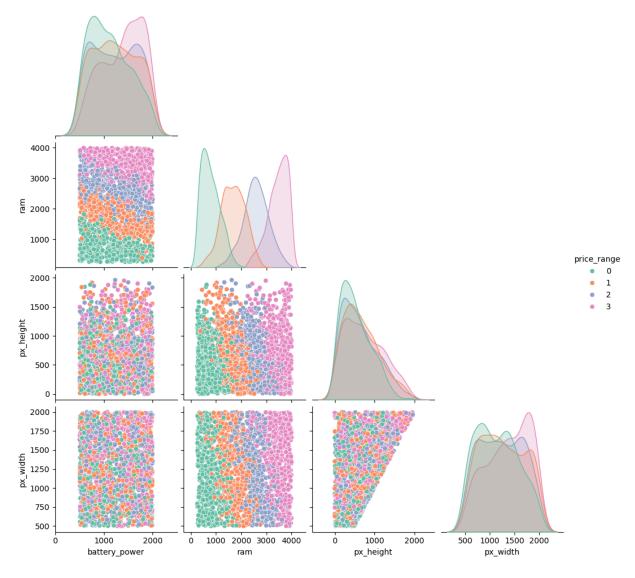


#### Correlation Heatmap

```
In [12]: plt.figure(figsize=(18, 8))
                               sns.heatmap(Train_df.corr(), annot=True, cmap='coolwarm')
                               plt.title("Feature Correlation Matrix")
                               plt.show()
                                                                                                                                     Feature Correlation Matrix
                                                                                                                                                                                                                                                                                          1.0
                                                         0.011 0.011 -0.042 0.033 0.016 -0.004 0.034 0.0018 -0.03 0.031 0.015 -0.00840.00065 -0.03 -0.021 0.053 0.012 -0.011 -0.0083 0.2
                                                                  0.021 0.035 0.0036 0.013 0.041 0.004 0.0086 0.036 -0.01 -0.0069 -0.042 0.026 -0.003 0.00061 0.014 -0.03 0.01 -0.022 0.021
                                                 0.011 0.021 1 -0.00130.00043-0.043 0.0065 -0.014 0.012 -0.0057-0.0052 -0.015 -0.0095 0.0034 -0.029 -0.0074 -0.011 -0.046 0.02 -0.024 -0.006
                            clock speed -
                                 dual_sim - 0.042 0.035 -0.0013 1 0.029 0.0032 0.016 0.022 0.009 0.025 0.017 0.021 0.014 0.041 0.041 0.012 0.017 0.039 0.014 0.017 0.023 0.017
                                                                                                                                                                                                                                                                                          0.8
                                         fc - 0.033 0.0036-0.00043-0.029 1 -0.017 -0.029 0.0018 0.024 -0.013 0.64 -0.01 -0.0052 0.015 -0.011 -0.012 -0.0068 0.0018 -0.015 0.02 0.022
                                    four_g - 0.016 0.013 -0.043 0.0032 -0.017 1 0.0087-0.0018 -0.017 -0.03 -0.0056 -0.019 0.0074 0.0073 0.027 0.037 -0.047 0.58 0.017 -0.018 0.015
                                                 0.004 0.041 0.0065 0.016 0.029 0.0087 1 0.0069 0.034 0.028 0.033 0.01 0.0083 0.033 0.038 0.012 0.0028 0.0094 0.027 0.007 0.044
                               m_dep - 0.034 0.004 0.014 0.022 0.0018 0.0069 1 0.022 0.0035 0.026 0.025 0.024 0.0094 0.025 0.018 0.017 0.012 0.0026 0.028 0.0008 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.001
                                  n_cores - 0.03 0.036 0.0057 0.025 0.013 0.03 0.028 0.0057 0.025 0.013 0.03 0.028 0.0035 0.019 1 0.0012 0.0069 0.024 0.0049 0.00031 0.026 0.013 0.015 0.024 0.014 0.0044
                                         pc - 0.031 -0.01 -0.0052 -0.017 | 0.64 | -0.0056 -0.033 | 0.026 | 0.019 -0.0012 | 1 | -0.018 | 0.0042 | 0.029 | 0.0049 -0.024 | 0.015 -0.0013-0.0087 | 0.0054 | 0.034
                               px_height - 0.015 -0.0069 -0.015 -0.021 -0.01 -0.019 0.01 0.025 0.00094-0.0069 -0.018 1 0.51 -0.02 0.06 0.043 -0.011 -0.031 0.022 0.052 0.15
                                px_width -0.0084 -0.042 -0.0095 0.014 -0.0052 0.0074 -0.0083 0.024 9e-05 0.024 0.0042 0.51 1 0.0041 0.022 0.035 0.0067 0.00035-0.0016 0.03
                                                 .00065 0.026 0.0034 0.041 0.015 0.0073 0.033 -0.0094 0.0026 0.0049 0.029 -0.02 0.0041 1 0.016 0.036 0.011 0.016 -0.03 0.023
                                                                                                                                                                                                0.51 -0.017 0.012 -0.02 0.026
                                                 -0.03 -0.003 -0.029 -0.012 -0.011 0.027 0.038 -0.025 -0.034-0.000310.0049 0.06 0.022 0.016
                                                -0.021 0.00061-0.0074 -0.017 -0.012 0.037 0.012 -0.018 -0.021 0.026 -0.024 0.043 0.035 0.036 0.51 1 -0.023 0.031 0.013 0.035 0.039
                                                                                                                                                                                                                                                                                         0.2
                                talk_time 0.053 0.014 -0.011 -0.039 -0.0068 -0.047 -0.0028 0.017 0.0062 0.013 0.015 -0.011 0.0067 0.011 -0.017 -0.023 1 -0.043 0.017 -0.03 0.022
                                  three_g - 0.012 -0.03 -0.046 -0.014 0.0018 0.58 -0.0094 -0.012 0.0016 -0.015 -0.0013 -0.031 0.00035 0.016 0.012 0.031 -0.043 1 0.014 0.0043 0.024
                           touch_screen --0.011 0.01 0.02 -0.017 -0.015 0.017 -0.027 -0.0026 -0.014 0.024 -0.0087 0.022 -0.0016 -0.03 -0.02 0.013 0.017 0.014 1 0.012 -0.03
                                                 0.0083-0.022-0.024-0.023-0.02-0.018-0.007-0.028-0.00041-0.01-0.0054-0.052-0.03-0.023-0.023-0.025-0.03-0.035-0.03-0.012-1-0.019
                                                0.2  0.021 -0.0066 0.017  0.022  0.015  0.044  0.00085 -0.03  0.0044  0.034  0.15  0.17  0.92  0.023  0.039  0.022  0.024  -0.03  0.019
                             price_range -
```

#### Pairplot

```
In [13]: sns.pairplot(Train_df[['battery_power', 'ram', 'px_height', 'px_width', 'price_
Out[13]: <seaborn.axisgrid.PairGrid at 0x710c8e147410>
```



Step 3: Preprocessing + Feature Scaling

• Split Data into Features & Labels

```
In [14]: x = Train_df.drop("price_range", axis=1)
y = Train_df["price_range"]
```

• Train-Test Split for Validation

```
In [15]: x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.2, random_s
In [16]: x_train.shape
Out[16]: (1600, 20)
In [17]: x_val.shape
Out[17]: (400, 20)
```

 Feature Scaling (Logistic & SVM need it!--> StandardScaler or Min Max Scalling, etc)

```
In [18]: scaler = StandardScaler()
   x_train_scaled = scaler.fit_transform(x_train)
   x_val_scaled = scaler.transform(x_val)
```

Prepare Test.csv for Final Prediction Later

```
In [19]: x_test_final = Test_df.drop("id", axis=1)
    x_test_final = scaler.transform(x_test_final)
```

#### Step 4: Train & Compare Multiple Models

Logistic Regression

```
In [20]: log_model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_i
log_model.fit(x_train_scaled, y_train)
log_preds = log_model.predict(x_val_scaled)
```

Support Vector Machine (SVM)

```
In [21]: svm_model = SVC(kernel="rbf", C=1, gamma='scale')
    svm_model.fit(x_train_scaled, y_train)
    svm_preds = svm_model.predict(x_val_scaled)
```

Random Forest (No Scaling Needed)

```
In [22]: rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_model.fit(x_train, y_train)
    rf_preds = rf_model.predict(x_val)
```

Evaluate All Models

```
In [23]: models = {
    "Logistic Regression": log_preds,
    "Support Vector Machine": svm_preds,
    "Random Forest": rf_preds
}

for name, preds in models.items():
    print(f"----- {name} -----")
    print("Accuracy:", accuracy_score(y_val, preds))
    print(classification_report(y_val, preds))
    print()
```

```
----- Logistic Regression -----
Accuracy: 0.975
            precision recall f1-score support
                          0.96
                                    0.98
                                              105
          0
                 1.00
          1
                 0.94
                          1.00
                                    0.97
                                              91
          2
                 0.99
                          0.95
                                    0.97
                                               92
          3
                 0.97
                          0.99
                                    0.98
                                              112
   accuracy
                                    0.97
                                              400
                                              400
                 0.98
                          0.97
                                    0.97
  macro avg
weighted avg
                 0.98
                          0.97
                                    0.98
                                              400
---- Support Vector Machine -----
Accuracy: 0.8925
            precision recall f1-score support
          0
                 0.95
                          0.93
                                    0.94
                                              105
          1
                 0.80
                          0.89
                                    0.84
                                               91
          2
                 0.84
                          0.82
                                    0.83
                                               92
          3
                 0.96
                          0.92
                                    0.94
                                              112
                                    0.89
                                              400
   accuracy
                 0.89
                          0.89
                                    0.89
                                              400
  macro avg
weighted avg
                 0.90
                          0.89
                                    0.89
                                              400
---- Random Forest ----
Accuracy: 0.8925
            precision recall f1-score support
          0
                 0.95
                          0.96
                                    0.96
                                              105
          1
                 0.89
                          0.87
                                    0.88
                                              91
          2
                 0.78
                          0.87
                                    0.82
                                               92
          3
                 0.94
                          0.87
                                    0.90
                                              112
   accuracy
                                    0.89
                                              400
                 0.89
                          0.89
                                    0.89
                                              400
  macro avg
```

#### Confusion Matrix Heatmap (Optional but Cool)

0.89

0.89

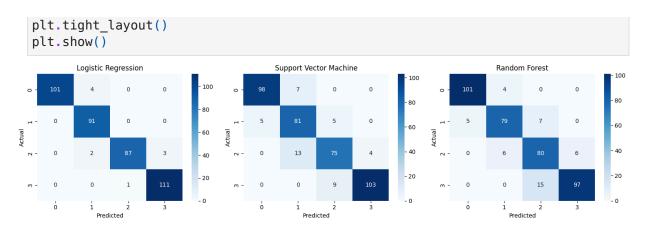
0.90

weighted avg

```
In [24]: plt.figure(figsize=(15, 4))

for i, (name, preds) in enumerate(models.items()):
    plt.subplot(1, 3, i+1)
    cm = confusion_matrix(y_val, preds)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(name)
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
```

400



Step 5: Grid Search for Hyperparameter Tuning

#### • Grid Search for SVM Model

```
In [25]: svm gs = SVC(random state=42)
         param_grid_svm = {
             'C': [0.1, 1, 10],
             'gamma': [0.001, 0.01, 0.1],
             'kernel': ['rbf']
         }
         grid search svm = GridSearchCV(estimator=svm gs,
                                         param grid=param grid svm,
                                         cv=5,
                                         n jobs=-1,
                                         scoring='accuracy',
                                         verbose=2)
         grid search svm.fit(x train scaled, y train)
         print("Best Parameters:", grid_search_svm.best_params_)
         best_svm = grid_search_svm.best_estimator_
         y_pred_best_svm = best_svm.predict(x_val_scaled)
         print("Accuracy of Tuned SVM:", accuracy score(y val, y pred best svm))
         print(classification_report(y_val, y_pred_best_svm))
        Fitting 5 folds for each of 9 candidates, totalling 45 fits
        Best Parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
        Accuracy of Tuned SVM: 0.9275
                      precision
                                    recall f1-score
                                                       support
```

Accuracy of Tuned SVM: 0.9275

precision recall f1-score support

0 0.97 0.94 0.96 105

1 0.85 0.95 0.90 91

2 0.91 0.87 0.89 92

•	0137	0151	0150	103
1	0.85	0.95	0.90	91
2	0.91	0.87	0.89	92
3	0.97	0.95	0.96	112
accuracy			0.93	400
macro avg	0.93	0.93	0.93	400
weighted avg	0.93	0.93	0.93	400

#### Grid Search for Random Forest Model

```
In [26]: rf = RandomForestClassifier(random_state=42)
         param grid = {
             'n estimators': [50, 100],
             'max_depth': [10, 20, None],
             'min samples split': [2, 5],
             'min_samples_leaf': [1, 2]
         grid_search = GridSearchCV(estimator= rf,
                                    param_grid=param_grid,
                                    cv=5,
                                    scoring='accuracy',
                                    n jobs=-1,
                                    verbose=2)
         grid_search.fit(x_train, y_train)
         print("Best Parameters:", grid search.best params_)
         best_rf = grid_search.best_estimator_
         y_pred_best_rf = best_rf.predict(x_val)
         print("Accuracy of Tuned RF:", accuracy score(y val, y pred best rf))
         print(classification_report(y_val, y_pred_best_rf))
        Fitting 5 folds for each of 24 candidates, totalling 120 fits
        Best Parameters: {'max depth': 20, 'min samples leaf': 1, 'min samples split':
        2, 'n estimators': 100}
        Accuracy of Tuned RF: 0.8875
                      precision
                                recall f1-score
                                                      support
                   0
                           0.95
                                     0.96
                                               0.96
                                                          105
                   1
                           0.89
                                     0.86
                                               0.87
                                                           91
                   2
                           0.77
                                   0.87
                                               0.82
                                                           92
                           0.94
                                   0.86
                                               0.90
                                                          112
                                               0.89
                                                          400
            accuracy
                           0.89
                                     0.89
                                               0.89
                                                          400
           macro avg
        weighted avg
                           0.89
                                     0.89
                                               0.89
                                                          400
```

Step 6: Final Prediction on test.csv

Predict on Final Test Data

```
In [27]: final_predictions = log_model.predict(x_test_final)
```

Combine with Test IDs (if needed)

```
In [28]: submission_df = pd.DataFrame({
             'id': Test_df['id'],
             'price_range': final_predictions
         })
In [29]: submission_df.to_csv("final_submission.csv", index=False)
         print("Submission file saved as final submission.csv")
```

Submission file saved as final\_submission.csv



Model	Accuracy
Logistic Regression	97.5%
Tuned SVM	92.75%
Random Forest	89.2%

📌 Conclusion: Logistic Regression performed best on this dataset. This suggests that the features are well-separated and a linear model is sufficient for high accuracy.

# Final Test Predictions

The best model was used to predict values on the unseen test dataset. These predictions can be used for further submission or validation.

# Author:

• Prince Raj