Mobile Phone Pricing Model

In [34]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("dataset.csv")  # Load your dataset
```

In [31]:
```
## Summary of the dataset
df.info()
```

```
<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: 328.3 KB
```

In [ ]:
```

```

In [14]:
```
df.describe()
```

Out[14]:

	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

In [15]:
```
df.columns
```

Out[15]:
```
Index(['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g',
       'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height',
       'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g',
       'touch_screen', 'wifi', 'price_range'],
      dtype='object')
```

In [16]:
```
df.shape
```

Out[16]:
```
(2000, 21)
```

In [17]:
```
df['price_range'].unique()
```

Out[17]:
```
array([1, 2, 3, 0])
```

In [18]:
```
## To check if we have missing value in the dataset
df.isnull().sum()
```

Out[18]:
```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

In [19]:
```
## To check the duplicate records
df[df.duplicated()]         ## Output : 0 rows x 21 columns
```

Out[19]:

battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt

0 rows × 21 columns

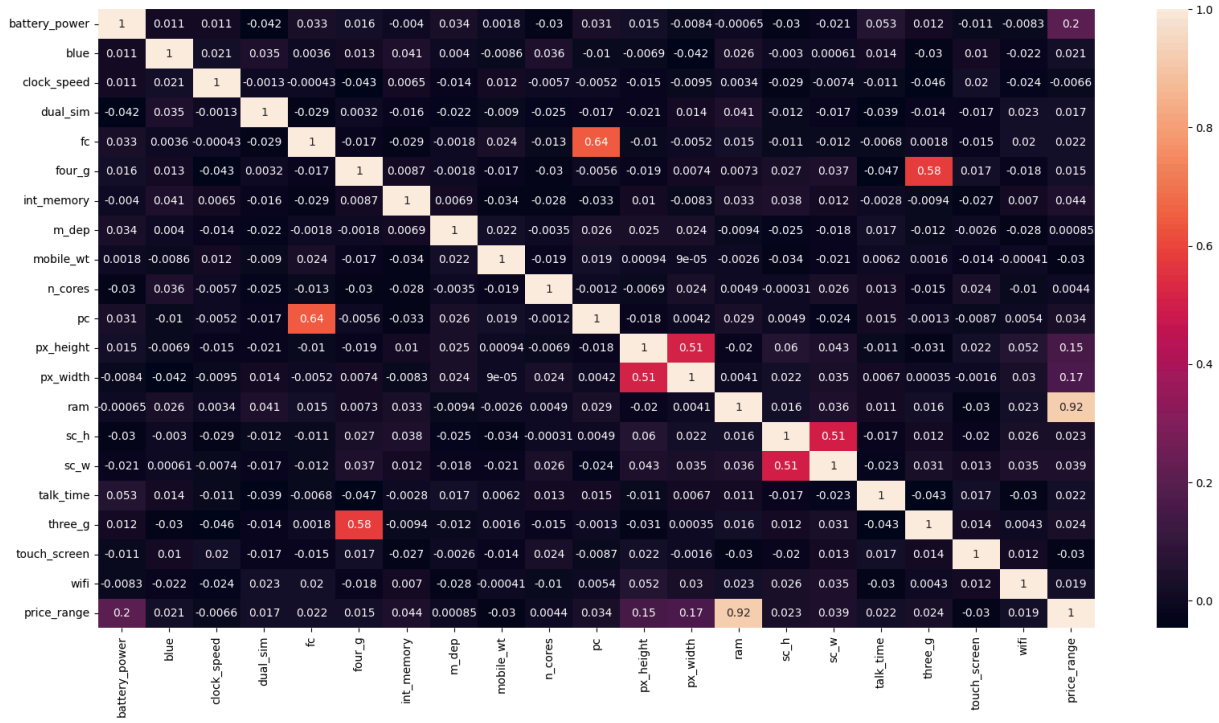
In [20]:
```
df.corr()
```

Out[20]:

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_
battery_power	1.000000	0.011252	0.011482	-0.041847	0.033334	0.015665	-
blue	0.011252	1.000000	0.021419	0.035198	0.003593	0.013443	
clock_speed	0.011482	0.021419	1.000000	-0.001315	-0.000434	-0.043073	
dual_sim	-0.041847	0.035198	-0.001315	1.000000	-0.029123	0.003187	-
fc	0.033334	0.003593	-0.000434	-0.029123	1.000000	-0.016560	-
four_g	0.015665	0.013443	-0.043073	0.003187	-0.016560	1.000000	
int_memory	-0.004004	0.041177	0.006545	-0.015679	-0.029133	0.008690	
m_dep	0.034085	0.004049	-0.014364	-0.022142	-0.001791	-0.001823	
mobile_wt	0.001844	-0.008605	0.012350	-0.008979	0.023618	-0.016537	-
n_cores	-0.029727	0.036161	-0.005724	-0.024658	-0.013356	-0.029706	-
pc	0.031441	-0.009952	-0.005245	-0.017143	0.644595	-0.005598	-
px_height	0.014901	-0.006872	-0.014523	-0.020875	-0.009990	-0.019236	
px_width	-0.008402	-0.041533	-0.009476	0.014291	-0.005176	0.007448	-
ram	-0.000653	0.026351	0.003443	0.041072	0.015099	0.007313	
sc_h	-0.029959	-0.002952	-0.029078	-0.011949	-0.011014	0.027166	
sc_w	-0.021421	0.000613	-0.007378	-0.016666	-0.012373	0.037005	
talk_time	0.052510	0.013934	-0.011432	-0.039404	-0.006829	-0.046628	-
three_g	0.011522	-0.030236	-0.046433	-0.014008	0.001793	0.584246	-
touch_screen	-0.010516	0.010061	0.019756	-0.017117	-0.014828	0.016758	-
wifi	-0.008343	-0.021863	-0.024471	0.022740	0.020085	-0.017620	
price_range	0.200723	0.020573	-0.006606	0.017444	0.021998	0.014772	

21 rows × 21 columns

In [35]:
```
## Visualising the correlation in the dataset
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot=True)
plt.savefig("correlation_heatmap.png", dpi=300, bbox_inches='tight')
```



In [ ]:
```
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Separate features and target variable
X = df.drop(columns=["price_range"])
y = df["price_range"]

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st

# Initialize and train the XGBoost model
model = xgb.XGBClassifier(objective="multi:softmax", num_class=4, eval_metric="mlog
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
```

```
c:\Users\amanv\AppData\Local\Programs\Python\Python312\Lib\site-packages\xgboost\cor
e.py:158: UserWarning: [18:49:13] WARNING: C:\buildkite-agent\builds\buildkite-windo
ws-cpu-autoscaling-group-i-08cbc0333d8d4aae1-1\xgboost\xgboost-ci-windows\src\learne
r.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Model Accuracy: 0.9240
```

In [23]:
```
# Function to predict price range for new input
def predict_price(new_data):
```

```
    new_df = pd.DataFrame([new_data], columns=X.columns)
    prediction = model.predict(new_df)[0]
    price_categories = {0: "Low Cost", 1: "Medium Cost", 2: "High Cost", 3: "Very H
    return price_categories[prediction]

# Example usage
new_mobile = {"battery_power": 1500, "blue": 1, "clock_speed": 2.0, "dual_sim": 1,
              "int_memory": 32, "m_dep": 0.5, "mobile_wt": 140, "n_cores": 4, "pc":
              "px_width": 1200, "ram": 3000, "sc_h": 14, "sc_w": 7, "talk_time": 12
              "touch_screen": 1, "wifi": 1}

predicted_category = predict_price(new_mobile)
print(f"Predicted Price Category: {predicted_category}")
```

```
Predicted Price Category: Very High Cost
```

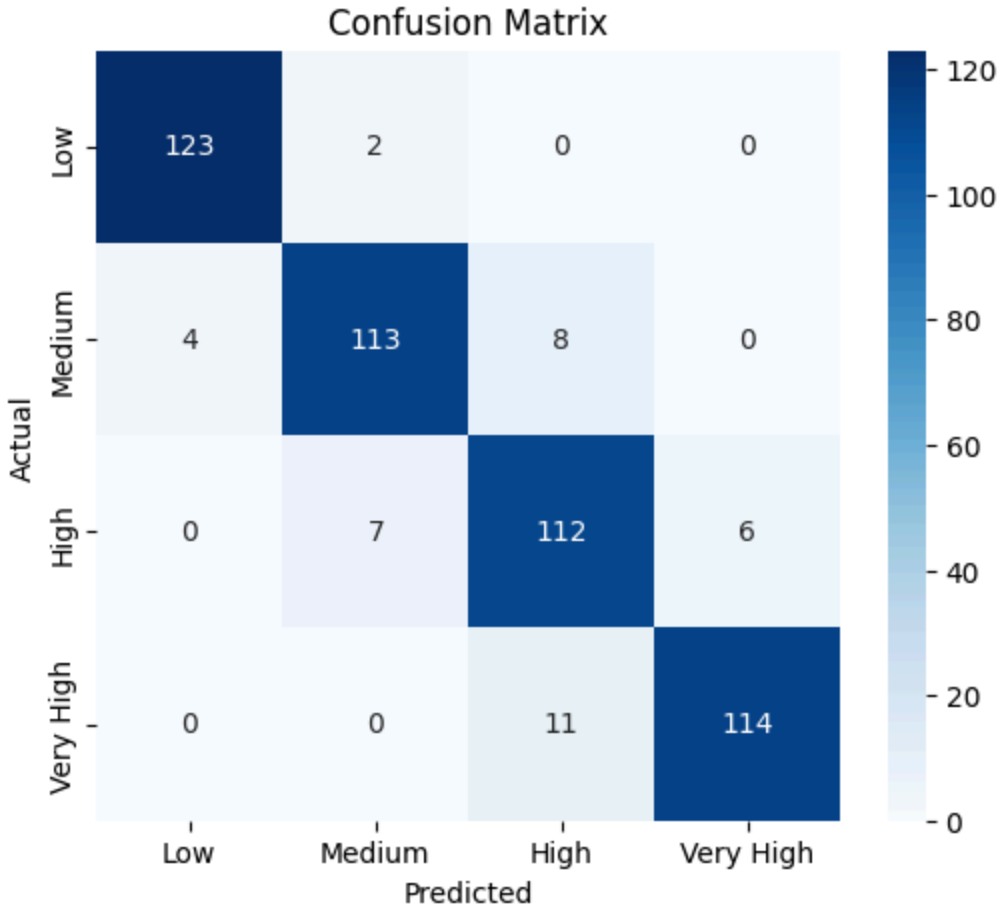
In [25]:
```
new_mobile1={
     "battery_power": 4500,"blue": 1,"clock_speed": 3.0,"dual_sim": 1,"fc": 32,"fou
    "int_memory": 32,"m_dep": 0.5,"mobile_wt": 145,"n_cores": 4,"pc": 10,"px_height
    "px_width": 1080,"ram": 12,"sc_h": 12,"sc_w": 6,"talk_time": 14,"three_g": 1,
    "touch_screen": 1,"wifi": 1
}
predicted_category = predict_price(new_mobile1)
print(f"Predicted Price Category: {predicted_category}")
```

```
Predicted Price Category: Low Cost
```

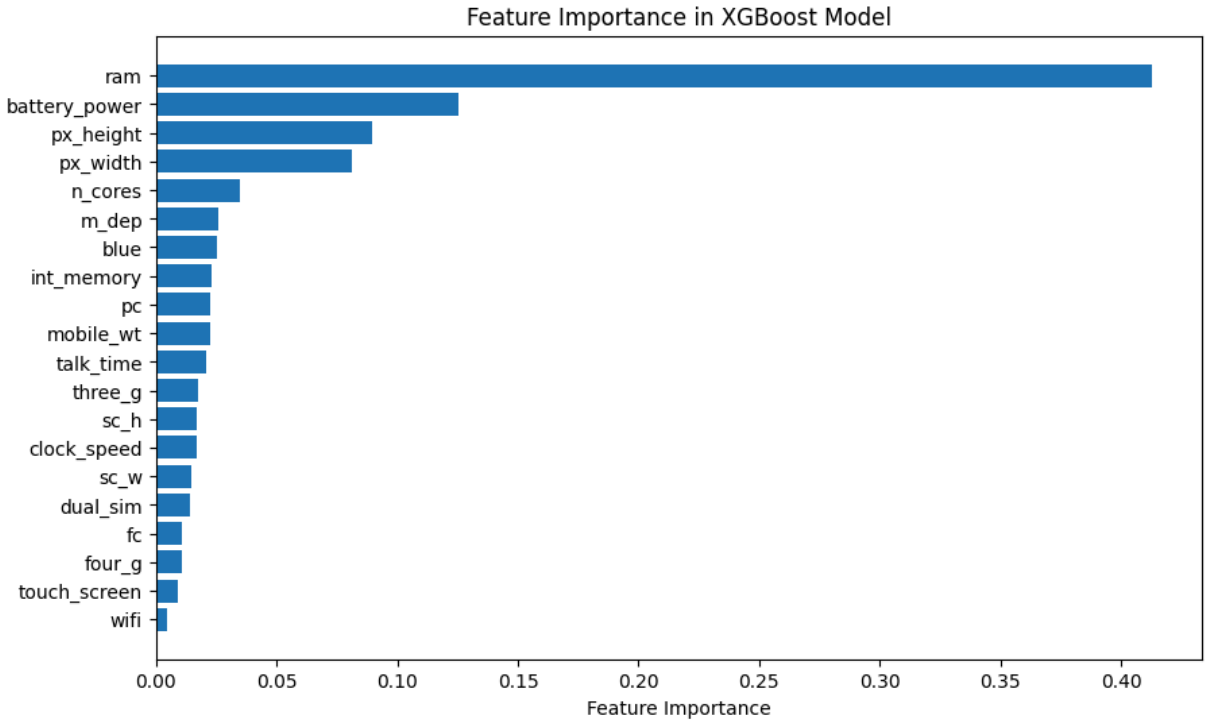
In [26]:
```
new_mobile_low = {
    "battery_power": 800, "blue": 0, "clock_speed": 1.0, "dual_sim": 0, "fc": 2, "f
    "int_memory": 8, "m_dep": 0.3, "mobile_wt": 180, "n_cores": 2, "pc": 5,
    "px_area": 500 * 800,  # Using px_area instead of px_height & px_width
    "ram": 512, "sc_h": 10, "sc_w": 4, "talk_time": 7, "three_g": 0,
    "touch_screen": 0, "wifi": 0
}

predicted_category = predict_price(new_mobile_low)
print(f"Predicted Price Category: {predicted_category}")
```

```
Predicted Price Category: Low Cost
```

In [37]:
```
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot heatmap
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Low", "Medium", "H
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



In [28]:
```
importances = model.feature_importances_
feature_names = X.columns

# Sort and plot
sorted_idx = importances.argsort()
plt.figure(figsize=(10,6))
plt.barh([feature_names[i] for i in sorted_idx], importances[sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Feature Importance in XGBoost Model")
plt.show()
plt.savefig("imp_features.png", dpi=200, bbox_inches='tight')
```




```
<Figure size 640x480 with 0 Axes>
```

In [38]:
```python
from sklearn.metrics import classification_report

# Generate classification report
report = classification_report(y_test, y_pred)

# Print report
print(report)

# Save classification report to a text file
with open("classification_report.txt", "w") as f:
    f.write(report)

print("Classification report saved as 'classification_report.txt'")
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       125
           1       0.93      0.90      0.91       125
           2       0.85      0.90      0.88       125
           3       0.95      0.91      0.93       125

    accuracy                           0.92       500
   macro avg       0.92      0.92      0.92       500
weighted avg       0.92      0.92      0.92       500

Classification report saved as 'classification_report.txt'
```