Feature Analysis & Price Prediction for Handsets

Understanding What Affects Mobile Prices and Predicting Them Accurately

In this project, I followed a step-by-step process to analyze and predict mobile prices. I began with data cleaning, where I resolved errors, handled missing values, and standardized the dataset. Then, I explored the data visually using **Python libraries** like **Matplotlib** and **Seaborn** to identify patterns and relationships. Key features like **RAM**, **Processor**, and **Battery** capacity were highlighted as significant factors affecting prices.

Next, I engineered new features to make the data more predictive and selected the most important ones for building models. Using **Scikit-learn**, I developed two models: a **Linear Regression model** as a baseline and a **Random Forest Regressor** to handle complex relationships. To improve **model accuracy**, I used **GridSearchCV** for hyperparameter tuning.

Finally, I evaluated the models using metrics like **R-squared** and **Mean Absolute Error (MAE)**, identifying the **Random Forest mod**el as the most accurate. This systematic approach integrated **Data Cleaning**, **Visualization**, **Feature Selection**, **Model Building**, and **Tuning** to understand mobile pricing dynamics and predict prices effectively.


by Debasis Baidya

Importing of the necessary Libraries & Modules

Numpy

It helped with mathematical operations, allowing me to compute statistics and handle missing values efficiently.

Pandas

To load & organize the datasets, making it easy to explore the data structure and perform operations like merging and filtering.

Matplotlib

Used it to create visualizations, such as bar charts and histograms, to better understand the data distributions and trends.

Seaborn

For generating more advanced visualizations, helping me check for skewness & visualize correlations between different variables.

Scikit-learn (Sklearn)

For preprocessing the data, specifically using the **SimpleImputer** to fill in missing values and to identify and manage outliers effectively.

Mobile Phone Price Prediction: A Comprehensive Analysis

1 Data Exploration

1. Dataset Size: 541 rows and 12 columns
Missing Values: None
2. Numerical Features: Memory, RAM, Battery_, Prize
Categorical Features: Model, Brand, Colour Family, Rear Camera, Front Camera, Processor_

2 Data Preprocessing

1. Identified **187 distinct models** and **275 colours.**
2. Conducted a **Univariate Analysis** of memory, RAM, battery, height, and price.
3. Performed a **Multivariate Analysis** to explore correlations between features such as memory and RAM, battery and mobile height, and AI lens and price.

3 Feature Analysis and Extraction

1.Unique Brands: Extracted distinct brands from the models.
2. Colour Normalization: Extracted & standardized colour names into categories.
3.Feature Identification: Extracted categorical and numerical features.
4.Outlier Detection and Handling: Extracted outliers using the IQR method and capped them to improve quality.
5.Data Encoding: Extracted and applied One-Hot and Label Encoding for analysis.

4 Model Training, Feature Scaling | PCA

1. Data Preparation: Extracted features (X) and target (y); split data into 80/20 training/testing sets.
2.Feature Scaling: Standardized features using **StandardScaler.**
3.Dimensionality Reduction: Reduced to 10 components with **PCA**.
4.Model Training: Trained models: Linear Regression, Decision Tree, Random Forest.
5.Performance Evaluation: Metrics: **MAE, RMSE, R^2 Score.**

5 Model Performance, Predict | Compare

1. Model Saving: Saved Random Forest as **`handset_price_model.pkl`.**
2. Performance: Evaluated models **(MAE, RMSE, R^2).**
3. Loading: Loaded model using **joblib.**
4. Data Prep: Selected 5 entries, scaled, and transformed.
5. Predictions: Predicted target variable (Prize).
6. Comparison: Actual vs predicted values; calculated differences.
7. Visualization: Scatter plot with perfect prediction line.

6 Improving Predictions &Tuning with XGBoost

1.Model Usage: The XGBoost Regressor was used for effective value prediction.
2.Parameter Tuning: Optimized the model's settings for better performance.
3.Optimization Process: GridSearchCV helped us find the best hyperparameter combinations.
4.Model Training: The model learned from the data to improve its predictions.
5.Optimal Parameters: Key settings included a learning rate of 0.1, max depth of 5, 200 trees, and a subsample rate of 1.0.
6. Prediction Results: It achieved a Mean Absolute Error (MAE) of 36.78, indicating strong accuracy.
7.New Data Testing: The model also performed well on new data, showing it can generalize effectively.

7 Feature Importance Analysis

1. Feature Importance Analysis: Feature importance extraction was used to identify key predictors.
2. Model Usage: The best predictive model was utilized for feature importance analysis.
3. Parameter Analysis: Feature importance were extracted and sorted by importance score.
4. Key Findings: "Prize" of product emerged as the most important feature, followed by front and rear camera specs followed by front and rear camera specs
5. Feature Ranking: Top 10 most important features were identified, providing insights into key prediction drivers.

Data Exploration Summary

Loading the Data

Importing of Libraries & Loading the Data

Libraries Used: Utilized Pandas and NumPy for data manipulation.

Data Source: Loaded the dataset from an Excel file located on the local drive.

Inspecting Dataset

Inspecting Dataset & Statistics Summary

First Few Rows: Displayed the initial rows using data.head()

DataFrame Info: Used data.info() to check the structure and data types.

Summary Statistics: Generated descriptive statistics with data.describe() to analyze distributions.

Missing Values

Missing Values & Dataset Size

Check for Missing Values: Used data.isnull().sum() to inspect for any missing entries in the dataset.

Dataset Size: 541 rows and 12 columns

Missing Values: None

Categorical Columns

Identifying Categorical Columns

Identified Columns: Extracted categorical columns using data.select_dtypes(include =['object'])

Categorical Features Found: Model, Brand, Colour Family, Rear Camera, Front Camera, Processor_

Numerical Columns

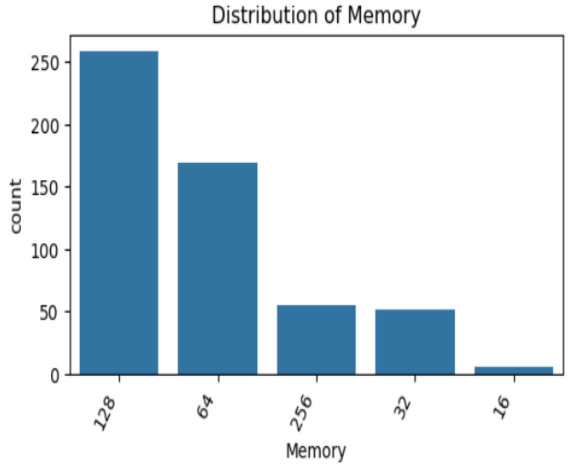
Identifying Categorical Columns

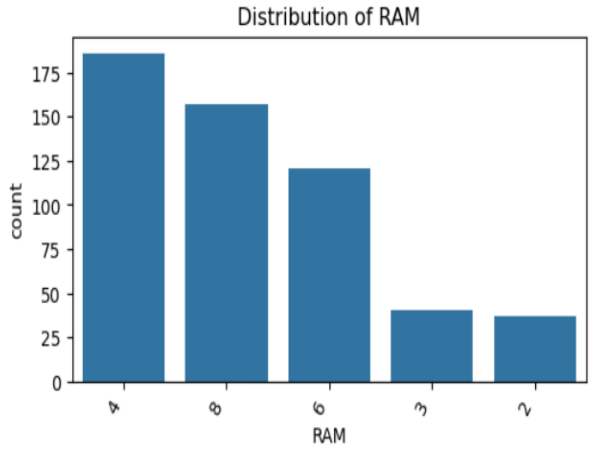
Identified Columns: Extracted numerical columns with data.select_dtypes(include =['int64', 'float64'])

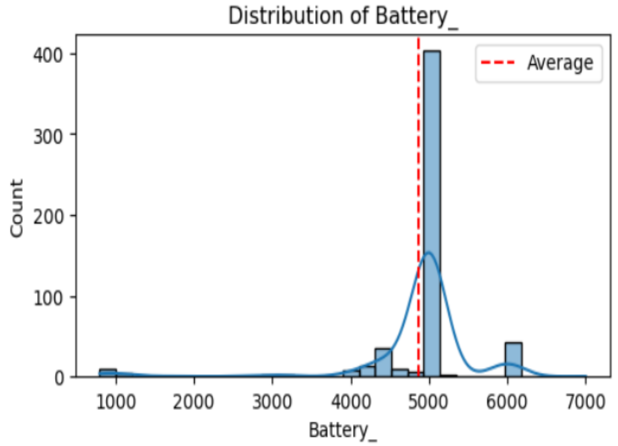
Numerical Features Found: Memory, RAM, Battery_, Prize

Data Preprocessing

Univariate Analysis







1 Memory and RAM

Most devices have 128GB or 64GB of memory and 4GB or 8GB of RAM.

2 Battery Capacity

The majority of phones have batteries around 5000mAh.

3 Mobile Height

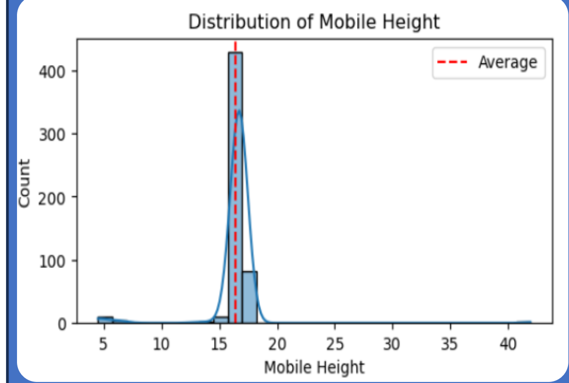
Most devices fall within the 15-17 units Height Range.

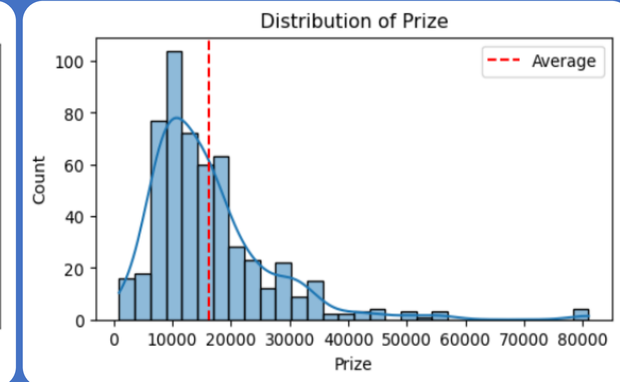
4 Price Distribution

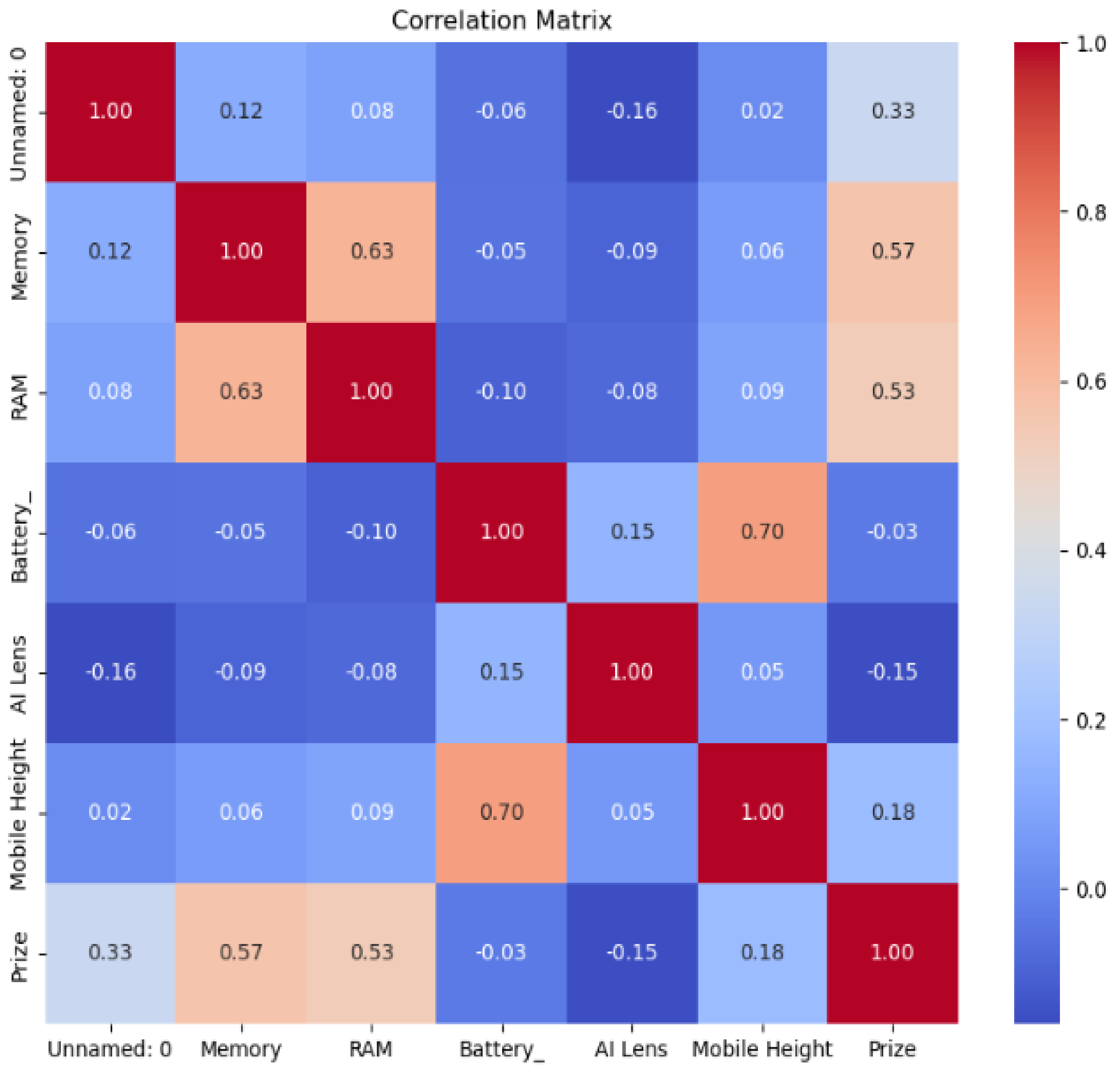
Most phones cost less than or equal to ₹20,000.00.

Univariate Analysis

```python
# Visualize feature distributions
import matplotlib.gridspec as gridspec
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

# Set up the figure
grid = gridspec.GridSpec(3, 2, hspace=0.5, wspace=0.4)
numeric_features = ['Memory', 'RAM', 'Battery_', 'Mobile Height', 'Prize']
fig = plt.figure(figsize=(14, 12))

# Plot distributions
for idx, feature in enumerate(numeric_features):
    ax = fig.add_subplot(grid[idx])

    if feature in ['Mobile Height', 'Prize', 'Battery_']:
        sns.histplot(data[feature], kde=True, bins=30, ax=ax)
        avg_value = data[feature].mean()
        ax.axvline(avg_value, color='red', linestyle='--', label='Average')
        ax.legend()
    else:
        sns.countplot(data=data, x=feature, order=data[feature].value_counts().index, ax=ax)
        ax.set_xticklabels(ax.get_xticklabels(), rotation=60, horizontalalignment='right')

    ax.set_title(f'Distribution of {feature}')

plt.tight_layout()
plt.show()
```







Data Preprocessing

Multivariate Analysis

Multivariate Analysis

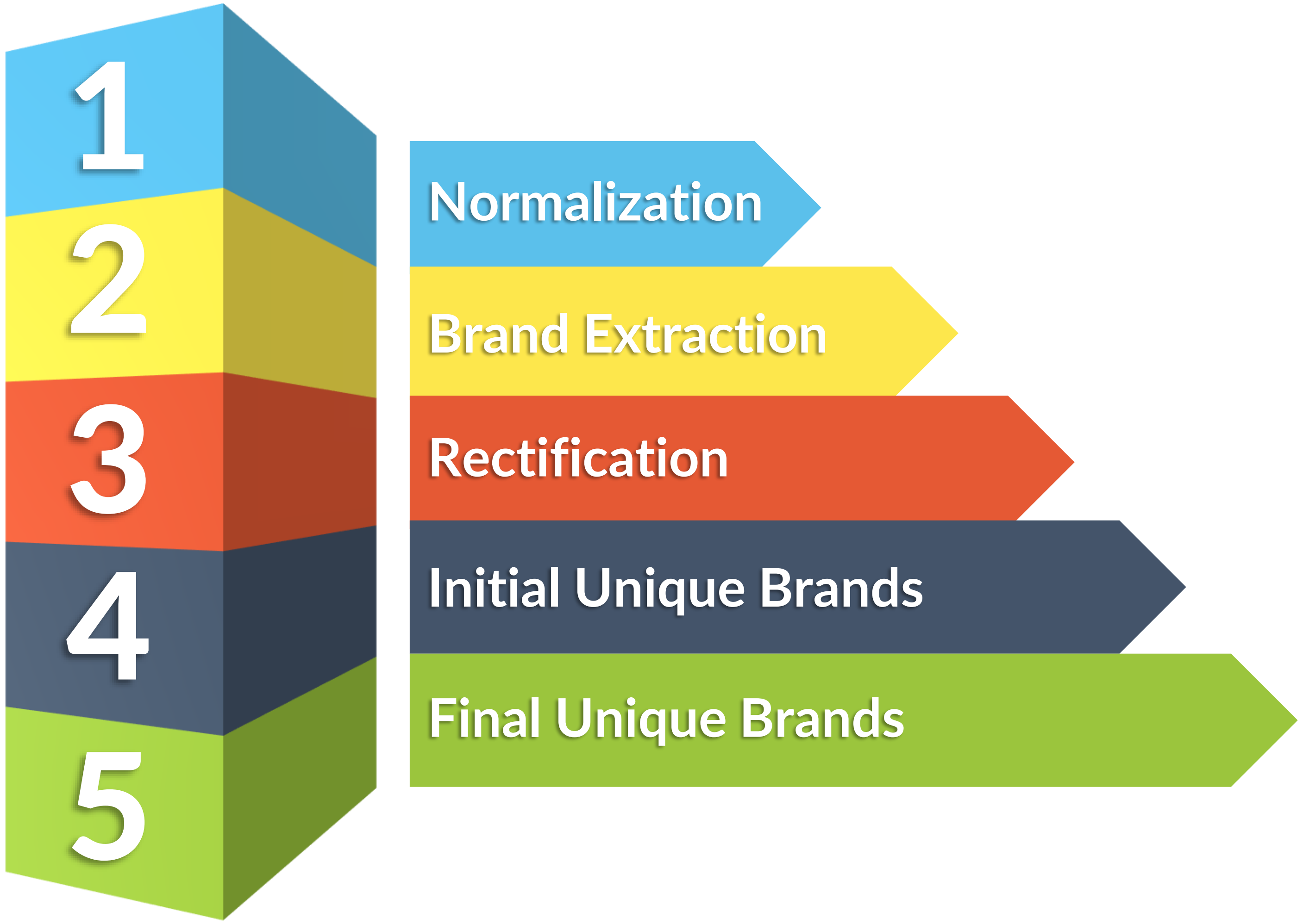
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Correlation analysis
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()

# Pairplot analysis for relationships
selected_features = ['Memory', 'RAM', 'Battery_', 'Mobile Height', 'Prize']
sns.pairplot(data[selected_features], diag_kind='kde', corner=True, plot_kws={'alpha': 0.7})
plt.suptitle('Pairplot Analysis of Selected Features', y=1.02, fontsize=16, fontweight='bold')
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()
```

Key Findings

1. **Memory-RAM**: Moderate positive correlation (0.63) - as Memory increases, RAM tends to increase.
2. **Battery-Mobile Height**: Moderately strong positive correlation (0.70) - higher Battery capacity is associated with increased Mobile Height.
3. **AI Lens-Prize**: Weak negative correlation (-0.15) - higher AI Lens feature is linked to slightly lower device Prices.
4. **Other Correlations**: Ranging from weak to moderate, both positive and negative, for features like Memory-Price, RAM-Price, etc.

Feature Analysis and Extraction

Feature Extraction: Brand Extraction from Models



1 | Normalization

Converted model names to lowercase for case-insensitive comparison.

2 | Brand Extraction

Extracted brand names by taking the first word from the normalized model names.

3 | Rectification

Corrected improperly extracted brands by replacing:
- 'I' with 'I Kall'
- 'Micromax1' with 'Micromax'

4 | Initial Unique Brands

A preliminary list of unique brands was generated:

['Infinix' 'Motorola' 'Poco' 'Redmi' 'Samsung' 'Vivo' 'Realme' 'Micromax'
'Google' 'Micromax1' 'Oppo' 'Tecno' 'Nothing' 'I' 'Itel' 'Oneplus' 'Iqoo'
'Nokia' 'Apple' 'Lava']

5 | Final Unique Brands

After rectification, the unique brands identified include:

['Infinix' 'Motorola' 'Poco' 'Redmi' 'Samsung' 'Vivo' 'Realme' 'Micromax'
'Google' 'Oppo' 'Tecno' 'Nothing' 'I Kall' 'Itel' 'Oneplus' 'Iqoo'
'Nokia' 'Apple' 'Lava']

Feature Analysis and Extraction

Feature Extraction: Colour Families Extraction from Colours

```python
# Normalize the colors in the DataFrame to sentence case
data['Colour'] = data['Colour'].str.title()

# Function to find the colour family using if statements
def find_colour_family(Colour):
    if 'Black' in Colour:
        return 'Black'
    elif 'Blue' in Colour:
        return 'Blue'
    elif 'Gray' in Colour or 'Grey' in Colour:
        return 'Gray'
    elif 'White' in Colour:
        return 'White'
    elif 'Marigold' in Colour or 'Yellow' in Colour:
        return 'Yellow'
    elif 'Charcoal' in Colour:
        return 'Black'
    elif 'Purple' in Colour:
        return 'Purple'
    elif 'Silver' in Colour:
        return 'Gray'
    elif 'Orange' in Colour:
        return 'Orange'
    elif 'Copper' in Colour:
        return 'Brown'
    elif 'Velocity Wave' in Colour:
        return 'Blue'
    elif 'Nitro Blaze' in Colour:
        return 'Red'
    elif 'Gold' in Colour:
        return 'Gold'
    elif 'Sea' in Colour:
        return 'Blue'
    elif 'Dark' in Colour:
        return 'Black'
    elif 'Chalk' in Colour:
        return 'White'
    elif 'Brown' in Colour:
        return 'Brown'
    else:
        return 'Other'  # Return 'Other' if no match is found

# Update the DataFrame with Colour Family
data['Colour Family'] = data['Colour'].apply(find_colour_family)

# Display the unique Colour Families
unique_colour_families = data['Colour Family'].unique()

print("Unique Colour Families:")
print(unique_colour_families)
```

1

Normalization

Converted colours to sentence case using the str.title() method

2

Color Family Classification

Defined a function to categorize colours based on keywords

3

DataFrame Update

Applied the function to create a new column for Colour Families

4

Unique Color Families Extraction

Extracted unique colour families from the updated DataFrame: Unique Colour Families:
['Black' ,'Blue', 'Gray', 'Yellow', 'White', 'Other', 'Gold', 'Purple', 'Orange', 'Brown', 'Red']

Data Preprocessing

Things I did before Outlier Handling

```python
# Now as I have now created the "Colour Family", Dropping the 'Colour', 'Normalized Model' Column
data = data.drop(columns=['Colour', 'Normalized Model'])

# Display the updated DataFrame
print("Updated DataFrame after dropping specified columns:")
data.head()
```

```python
# Reorder the columns in the DataFrame
columns_order = ['Model', 'Brand', 'Colour Family', 'Memory', 'RAM', 'Battery_',
                 'Rear Camera', 'Front Camera', 'AI Lens', 'Mobile Height',
                 'Processor_', 'Prize']

# Reassign the DataFrame with the new column order
data = data[columns_order]

# Display the DataFrame to confirm the change
data.head()
```

```python
# Identifying categorical and numerical columns
categorical_cols = data.select_dtypes(include=['object']).columns.tolist()
numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical Columns:")
print(categorical_cols)

print("\nNumerical Columns:")
print(numerical_cols)
```

```
Categorical Columns:
['Model', 'Brand', 'Colour Family', 'Rear Camera', 'Front Camera', 'Processor_']

Numerical Columns:
['Memory', 'RAM', 'Battery_', 'AI Lens', 'Mobile Height', 'Prize']
```

1

Reordered Columns

Specified the new column order and reassigned the DataFrame to this new order

2

Dropped Unnecessary Columns

Removed the 'Colour' and 'Normalized Model' columns from the DataFrame as we now have 'Brand' & 'Colour Family'.

The DataFrame now contains the columns in the desired order, excluding 'Colour' and 'Normalized Model'

3

Re-Identified Categorical & Numerical Columns

Categorical Columns:

['Model', 'Brand', 'Colour Family', 'Rear Camera', 'Front Camera', 'Processor_']

Numerical Columns:

['Memory', 'RAM', 'Battery_', 'AI Lens', 'Mobile Height', 'Prize']

Outliers Detection | Handling | One Hot & Label Encoding

Outlier Identification Function
Defined a function identify_outliers_iqr to calculate outliers based on the Interquartile Range (IQR)

IQR Calculation
Calculated the first quartile (Q1) and third quartile (Q3) for the specified column to determine the IQR

Outlier Bounds
Established lower and upper bounds:
- Lower bound = Q1 - 1.5 * IQR
- Upper bound = Q3 + 1.5 * IQR

Count of Outliers
Generated a count of outliers for each numerical feature using the defined function

Capping Outliers
Used the clip method to cap values outside the bounds for each numerical feature

One-Hot Encoding
Applied One-Hot Encoding to nominal categorical columns (*Model, **Brand, **Processor_, **Colour Family*) using pd.get_dummies(), excluding the first category to avoid dummy variable trap.

Label Encoding
Used LabelEncoder to encode ordinal categorical columns (*Rear Camera, **Front Camera*) into numerical values

Model Building: Model Training, Feature Scaling | PCA

```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Define features (X) and target (y)
X = data_encoded
y = data['Prize']

# Split 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)

# Feature scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# PCA for dimensionality reduction
from sklearn.decomposition import PCA

pca = PCA(n_components=10)  # Adjust number of components as needed
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

```python
# Model training
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(random_state=42, n_estimators=100)
}

performance = {}

for model_name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)

# Metrics
    mae = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    r2 = model.score(X_test_pca, y_test)

    performance[model_name] = {'MAE': mae, 'RMSE': rmse, 'R²': r2}
```

1

Data Preparation

- Defined features (X) and target variable (y) from the dataset.
- Split the data into training and testing sets with an 80/20 ratio, using 80% for training and 20% for testing.

2

Feature Scaling

- Applied *StandardScaler* to standardize the features, ensuring they have a mean of 0 and a standard deviation of 1.

3

Dimensionality Reduction

- Utilized Principal Component Analysis (PCA) to reduce the feature set to 10 components, retaining the most significant variance in the Data.

4

Model Training

- Trained three regression models:
  - Linear Regression
  - Decision Tree Regressor
  - Random Forest Regressor with 100 estimators.

5

Performance Evaluation

- Evaluated each model's performance using:
  - Mean Absolute Error (MAE)
  - Root Mean Squared Error (RMSE)
  - R^2 Score

Model Evaluation: Model Performance, Best Model Selection & Saving the Model

1

Model Saving

The best-performing model, identified as the Random Forest Regressor, was saved using the joblib library. The model is stored as “handset_price_model.pkl” for future use.

2

Model Performance

Displayed the performance metrics for each model:
1. Linear Regression | 2. Decision Tree | 3. Random Forest.

3

Outcome

The Random Forest Regressor emerged as the best model based on the performance metrics, particularly excelling in R^2 score, indicating it explains a significant portion of the variance in the target variable (Prize).

4

Model Loading

The saved model (handset_price_model.pkl) was loaded using the joblib library.

5

Data Preparation for Predictions

Selected the first 5 entries from the test dataset (X_test) for making predictions.
Scaled the new data using the previously fitted *StandardScaler*.
Transformed the scaled data using *PCA* to match the model's input requirements.

6

Predictions

The loaded model was used to predict the target variable (Prize) for the prepared new data. Predictions were printed, providing the estimated values for the selected entries.

7

Actual vs Predicted Values

A DataFrame was created to compare the actual values from the test set with the predicted values obtained from the loaded model for the first 5 entries.

8

Calculating Differences

A new column was added to the DataFrame to calculate the difference between the actual and predicted values.

```
# Display performance
print("Model Performance:")
for model, metrics in performance.items():
    print(f"{model}: MAE={metrics['MAE']:.2f}, RMSE={metrics['RMSE']:.2f}, R²={metrics['R²']:.2%}")
```

```
Model Performance:
Linear Regression: MAE=2221.80, RMSE=3104.58, R²=83.49%
Decision Tree: MAE=1831.15, RMSE=2984.08, R²=84.74%
Random Forest: MAE=1340.23, RMSE=1998.53, R²=93.16%
```

```
# Select the best model
from sklearn.ensemble import RandomForestRegressor

best_model = RandomForestRegressor(random_state=42, n_estimators=100)
best_model.fit(X_train_pca, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
# Save the model
import joblib

joblib.dump(best_model, 'handset_price_model.pkl')
```

```
['handset_price_model.pkl']
```

```
loaded_model = joblib.load('handset_price_model.pkl')

new_data = X_test.iloc[:5]
new_data_scaled = scaler.transform(new_data)
new_data_pca = pca.transform(new_data_scaled)

predictions = loaded_model.predict(new_data_pca)
print("Predictions for new data:")
print(predictions)
```

```
Predictions for new data:
[ 8430.21  7372.49 19963.82 10946.01 13123.05]
```

Checking the difference of Actual & Predicted Values

```
import pandas as pd

actual_values = y_test.iloc[:5]

# DataFrame to display actual vs predicted values
results_df = pd.DataFrame({
    'Actual Values': actual_values,
    'Predicted Values': predictions
})

# Display the results
print("Comparison of Actual vs Predicted Values for New Data:")
print(results_df)

actual_values = y_test.iloc[:5]

# Calculating the difference
results_df['Difference'] = results_df['Actual Values'] - results_df['Predicted Values']

# Display the updated DataFrame with differences
print("\nComparison with Differences:")
print(results_df)
```

```python
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error

# Defining the model
model = xgb.XGBRegressor()

# Setting up the parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0],
}

# Setting up the grid search
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5, scoring='neg_mean_absolute_error')

# Fitting the model
grid_search.fit(X_train, y_train)

# Best parameters
print("Best parameters found: ", grid_search.best_params_)

# Predictions with the best model
best_model = grid_search.best_estimator_
predictions = best_model.predict(X_test)

# Evaluation
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae}')
```

```python
# New DataFrame to display actual vs predicted values
results_df_new = pd.DataFrame({
    'Actual Values': actual_values,
    'Predicted Values': new_predictions
})

# Calculate the difference
results_df_new['Difference'] = results_df_new['Actual Values'] - results_df_new['Predicted Values']

# Display the updated DataFrame with differences
print("\nComparison with Differences:")
print(results_df_new)
```

```python
import matplotlib.pyplot as plt

# Visualize predictions vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(actual_values, new_predictions, color='blue', label='Predicted Values')
plt.plot([actual_values.min(), actual_values.max()],
         [actual_values.min(), actual_values.max()],
         'r--', label='Perfect Prediction')  # Diagonal line
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Predicted vs Actual Values for New Data')
plt.legend()
plt.grid()
plt.show()
```

Improving Predictions with Hyperparameter Tuning in XGBoost

Feature Important Analysis

6. Feature Importance Analysis:

```
# Extract feature importances
feature_importances = best_model.feature_importances_
feature_names = X_train.columns

# Create a DataFrame to display importances
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances}
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Display feature importances
print("Feature Importances:\n")
print(importance_df.head(10))
```

```
Feature Importances:

                   Feature  Importance
7                    Prize    0.884032
4             Front Camera    0.114773
3              Rear Camera    0.001152
0                   Memory    0.000006
1                      RAM    0.000005
87     Model_POCO X4 Pro 5G   0.000003
235       Processor__MT6260A  0.000002
132    Model_Tecno Spark 9T   0.000002
59         Model_Micromax 3   0.000002
204          Brand_Oneplus    0.000002
```

```
# Visualize feature importances using a bar chart
plt.figure(figsize=(10, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df.head(5), palette = "viridis")
plt.title('Top 10 Feature Importances', fontsize = 14, fontweight ='bold')
plt.xlabel('Importance Score', fontsize = 12, fontweight ='bold')
plt.ylabel('Features', fontsize = 12, fontweight ='bold')

plt.xticks(fontsize =12)
plt.yticks(fontsize = 12)
plt.tight_layout()
plt.show()
```

Most Important Feature

The "Prize" of the product is identified as the most important feature, exhibiting a high importance score that significantly influences model predictions.

Relatively Important Features

The specifications of the front and rear cameras are also important, contributing meaningfully to the model's accuracy and predictions.

Less Important Features

Other features in the dataset have much lower importance scores, indicating they contribute less to the overall predictions of the model.

Feature Extraction

The feature importances were derived from the best predictive model utilized in the analysis.

Organization

The features along with their importance scores were compiled into a table and sorted from highest to lowest importance for clarity.

Summary

This project developed a mobile handset price prediction model using data exploration, feature extraction (Brand & Colour), preprocessing (One-Hot & Label Encoding, IQR Handling), & Model Building with Random Forest and XGBoost. Key predictors included Memory, RAM, Battery, and Brand. Challenges with high cardinality features and outliers were addressed, resulting in a strong predictive model.

Future Enhancements

To improve the model by adding more features. Tune the settings for better accuracy. Using multiple models can enhance predictions. Checking performance will ensure reliability. Tools can help explain predictions. A real-time system would be useful. Gathering feedback will aid in improvements.

Q&A

The presentation is open for any questions & suggestions from the reviewer.