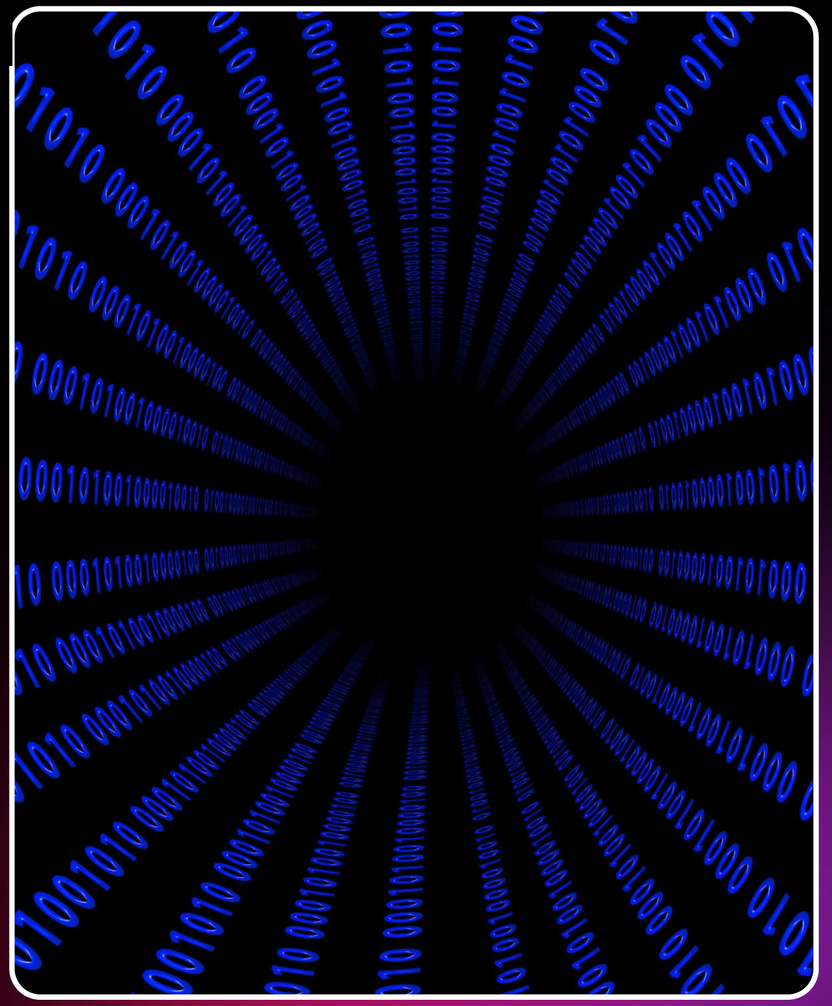
Exploratory Data Analysis (EDA) and Machine Learning Approaches for Mobile Specifications Dataset



PRESENTED BY

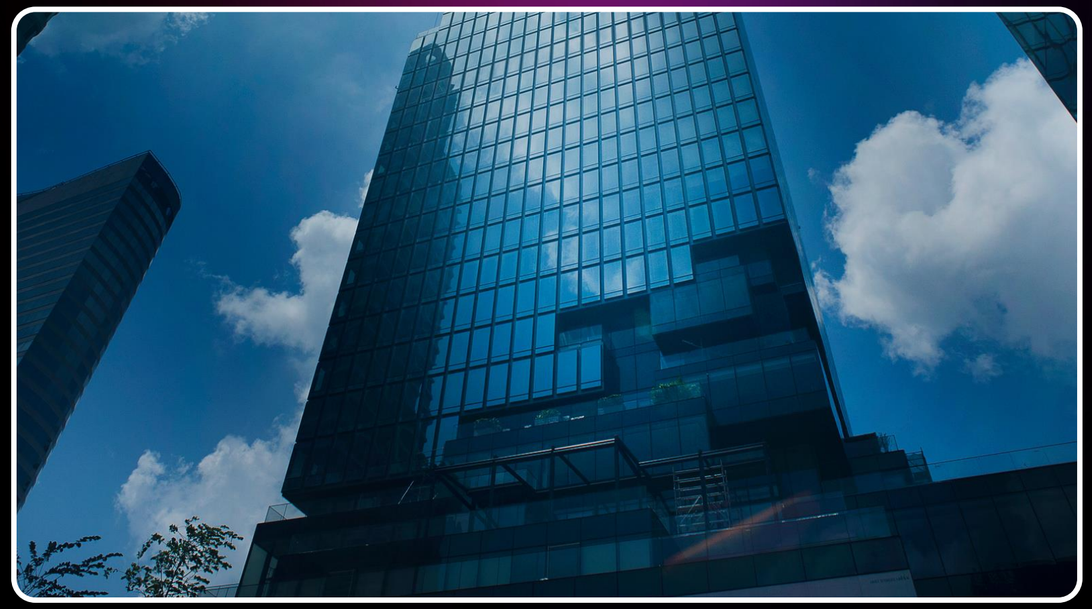
KALAVATHI ALEGAPALLI

Introduction

This project focuses on analyzing a dataset containing detailed specifications of various mobile phones, including model, color, memory, RAM, battery capacity, camera features, AI lens presence, mobile height, processor, and price. The primary objective is to develop a predictive model that accurately estimates mobile phone prices based on these features.

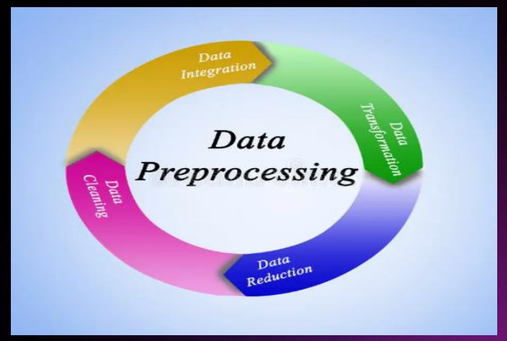
Introduction

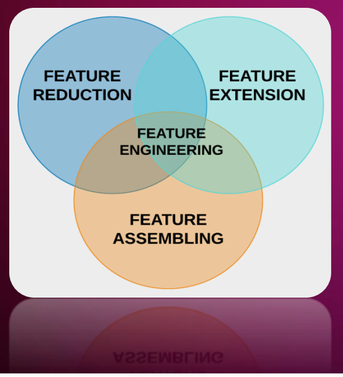
Project Overview

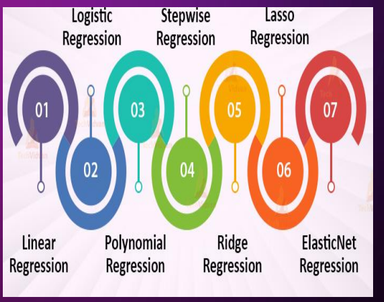


1. Data Cleaning & Pre-processing
2. Exploratory Data Analysis (EDA)
3. Feature Engineering & Selection
4. Model Training, Evaluation & Interpretation

Project Tasks:







1. Data Cleaning & Pre-processing

- Handling Missing Values & Duplicates
- Data Preprocessing (Encoding, Scaling, etc.)
- Outlier Detection & Treatment

2. Exploratory Data Analysis(EDA)

- Univariate Analysis
- Bivariate Analysis
- Multivariate Analysis

3. Feature Engineering& Selection

- Creating & Transforming Features
- Feature Importance Analysis

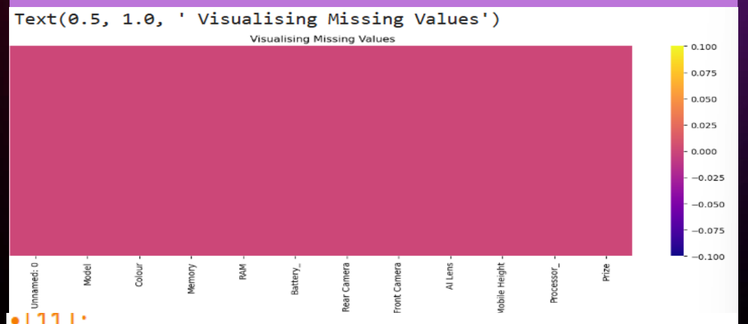
4. Model Training, Evaluation & Interpretation

- Model Training & Selection
- Model Evaluation & Accuracy Assessment

MAIN AGENDA

To ensure data consistency, we will check for missing values, verify data types, and identify duplicate entries and outliers. If any issues are found, appropriate preprocessing steps will be applied; otherwise, the dataset will be considered clean for further analysis.

POINT 01




```
[11]:
print(f' We have {mobile_data.duplicated().sum()}
duplicate values in dataset.')

 We have 0 duplicate values in dataset.
```

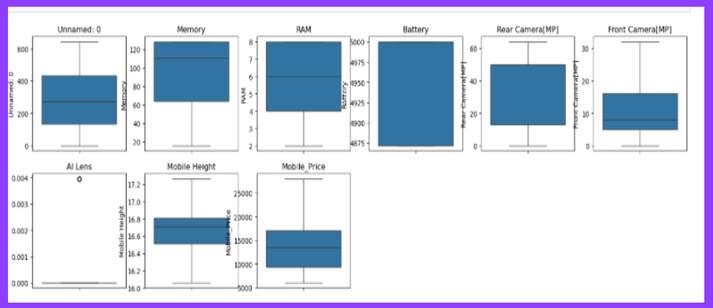
- The dataset contains 0 duplicate records, ensuring data integrity.
- - The dataset contains 0 missing records, ensuring data integrity.

POINT 02

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541 entries, 0 to 540
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         541 non-null    int64
 1   Brand              541 non-null    object
 2   Series             541 non-null    object
 3   Mobile_colour      541 non-null    object
 4   Model              541 non-null    object
 5   Colour             541 non-null    object
 6   Memory             541 non-null    int64
 7   RAM                541 non-null    int64
 8   Battery            541 non-null    int64
 9   Rear Camera[MP]    541 non-null    Int64
 10  Front Camera[MP]   541 non-null    Int64
 11  AI Lens            541 non-null    int64
 12  Mobile Height      541 non-null    float64
 13  Processor          541 non-null    object
 14  Mobile_Price       541 non-null    int64
dtypes: Int64(2), float64(1), int64(6), object(6)
memory usage: 64.6+ KB
```

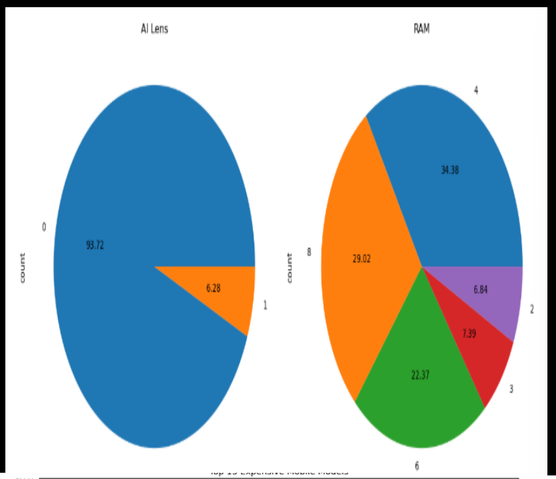
The dataset contains 541 entries and 15 columns with int64, float64, and object data types.
All columns have complete data with no missing values.
The dataset occupies 64.6 KB of memory.

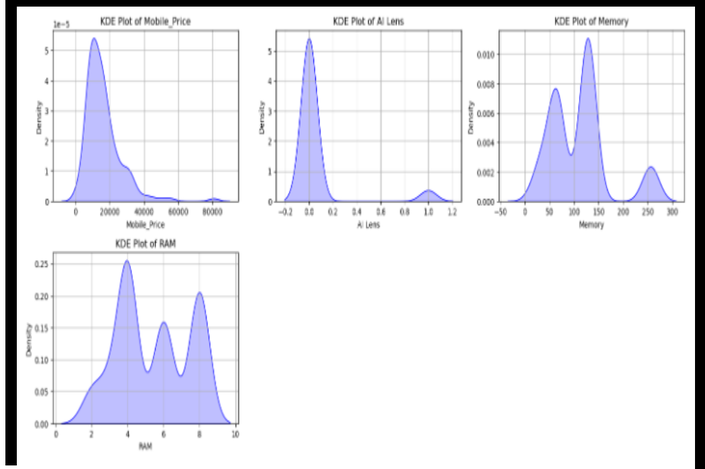
POINT 03

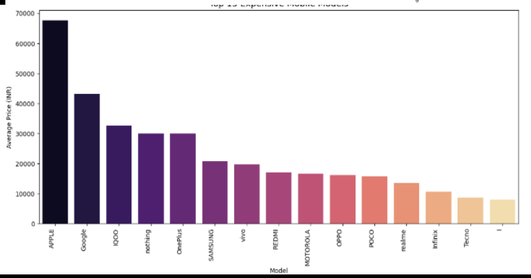


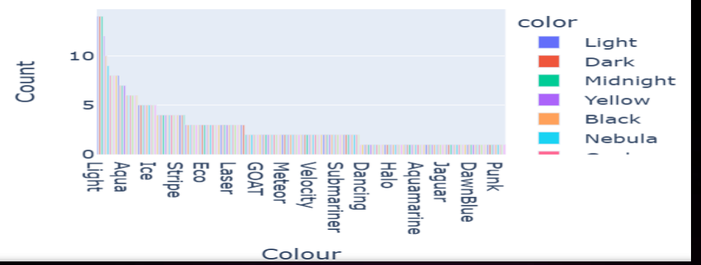
-Boxplots initially revealed outliers in "AI Lens" and "Mobile Height," which have now been removed.
-Skewness in numerical features has been adjusted while maintaining data integrity.
-The "Battery" feature remains uniformly distributed with no visible outliers.

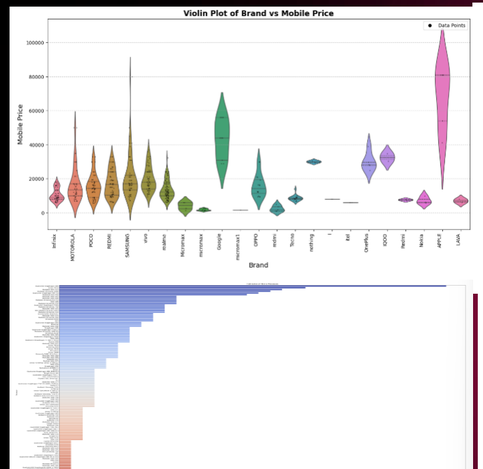
UNI-VARIANT AND BIVARIANT ANLAYSIS:

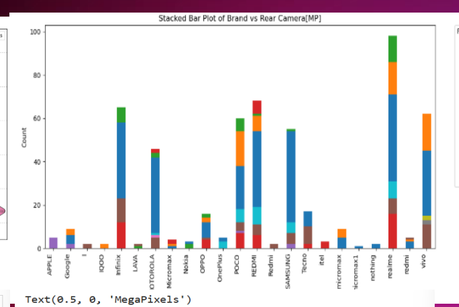


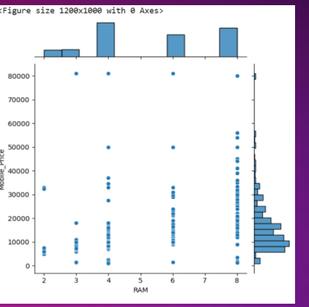


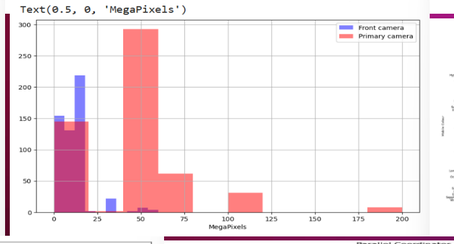


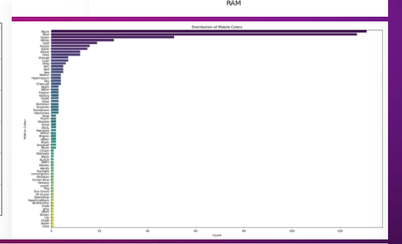


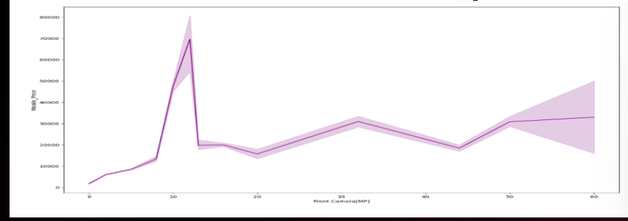


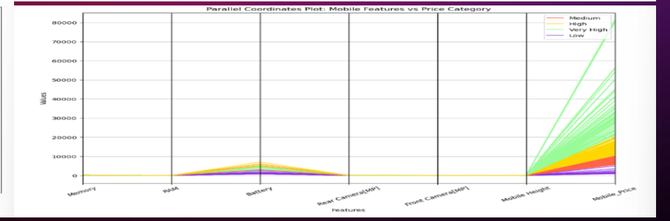










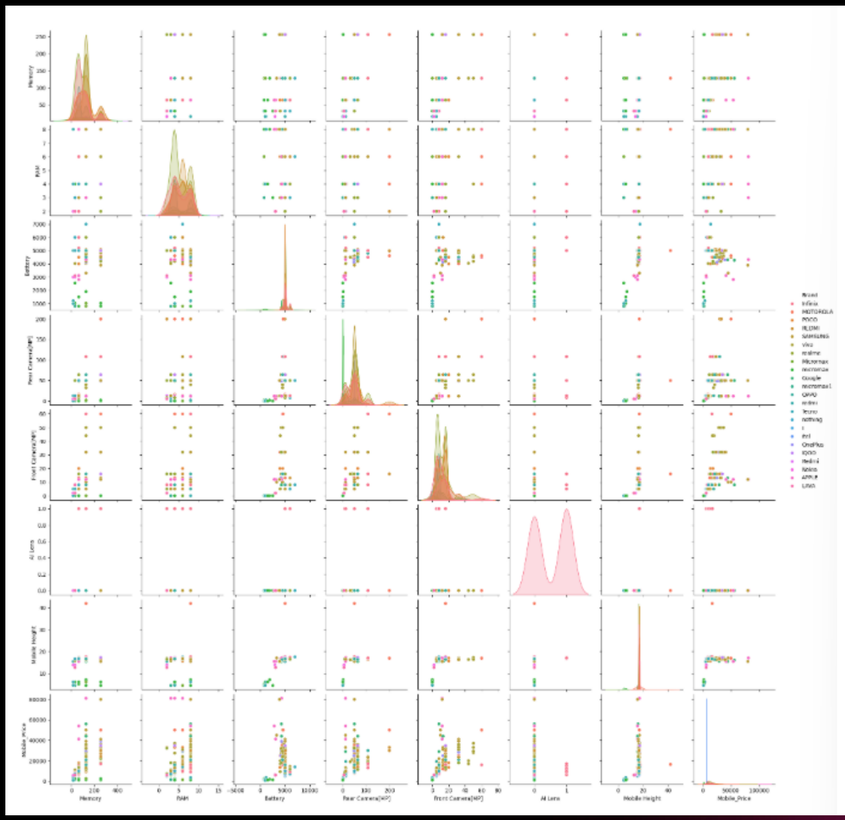


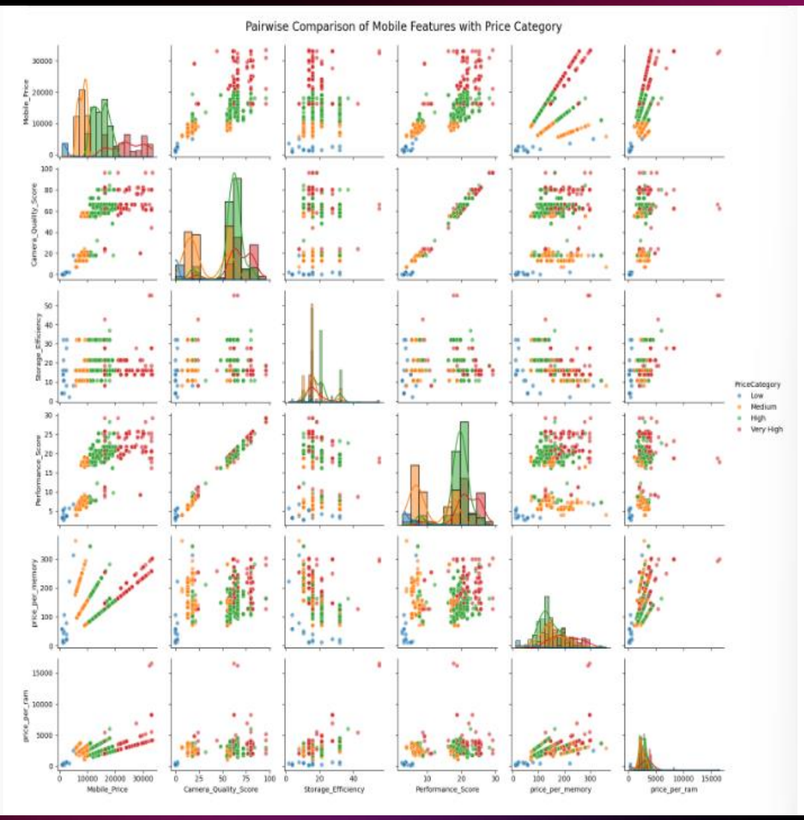
The univariate analysis includes pie charts, bar charts, and density plots to explore the distribution of categorical and numerical features such as AI Lens, RAM, Memory, Battery, and Color.
These visualizations help identify feature distributions, dominant categories, and potential data imbalances for further analysis.

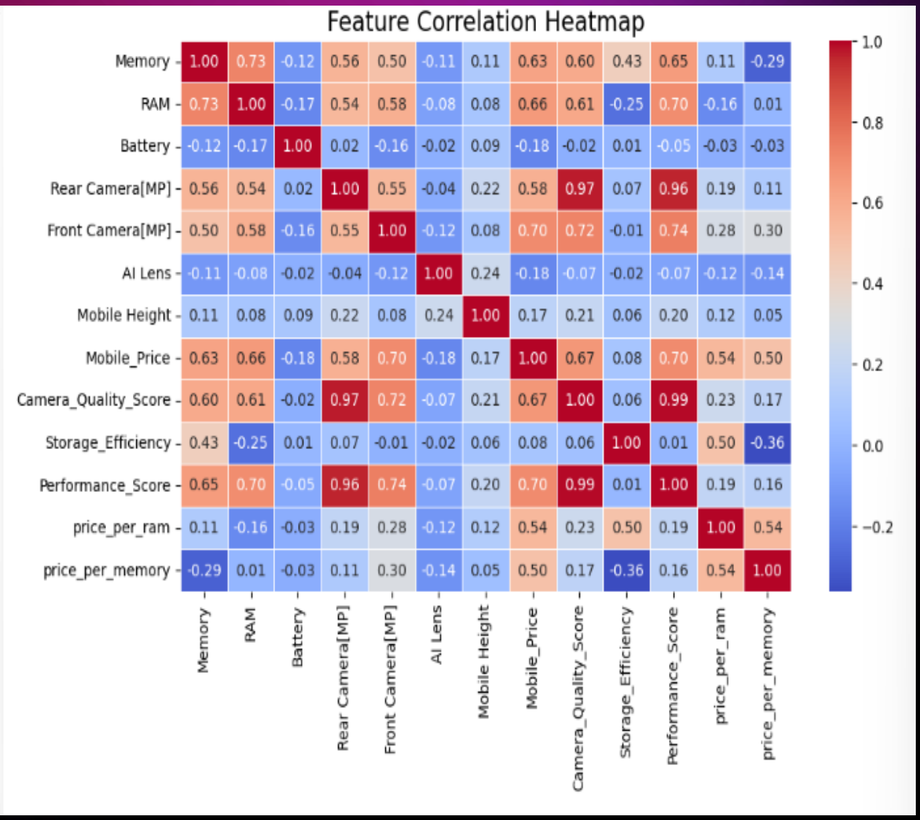
For a detailed univariate analysis, refer to **Univariate Analysis** **HYPERLINK.**

-The bivariate analysis explores relationships between mobile phone features and price using violin plots, bar charts, scatter plots, and line graphs.
-These visualizations help identify trends, correlations, and category-wise price variations for better model insights.
-The remaining plots are available in the **IPYNB file** for further reference.

MULTI-VARIANT ANALYSIS







Multivariate analysis includes pair plots and correlation heatmaps to examine relationships between multiple features, revealing strong positive correlations between **RAM, Battery, and Mobile Price**.

Scatter plots highlight patterns in feature interactions, with memory and camera specifications showing a significant impact on price categories.

The correlation heatmap indicates that **RAM and Performance Score** have the highest influence on price, while some features show weak or no correlation, which can aid in feature selection.

Feature Engineering

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541 entries, 0 to 540
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Unnamed: 0     541 non-null    int64
 1   Model          541 non-null    object
 2   Colour         541 non-null    object
 3   Memory         541 non-null    int64
 4   RAM            541 non-null    int64
 5   Battery_       541 non-null    int64
 6   Rear Camera    541 non-null    object
 7   Front Camera   541 non-null    object
 8   AI Lens        541 non-null    int64
 9   Mobile Height  541 non-null    float64
 10  Processor_     541 non-null    object
 11  Prize          541 non-null    int64
dtypes: float64(1), int64(6), object(5)
memory usage: 50.8+ KB
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541 entries, 0 to 540
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Brand                 541 non-null    object
 1   Series                541 non-null    object
 2   Mobile_colour         541 non-null    object
 3   Memory                541 non-null    float64
 4   RAM                   541 non-null    float64
 5   Battery               541 non-null    float64
 6   Rear Camera[MP]       541 non-null    float64
 7   Front Camera[MP]      541 non-null    float64
 8   AI Lens               541 non-null    float64
 9   Mobile Height         541 non-null    float64
 10  Processor             541 non-null    object
 11  Mobile_Price          541 non-null    float64
 12  Camera_Quality_Score  541 non-null    float64
 13  Storage_Efficiency    541 non-null    float64
 14  Screen_Type           541 non-null    object
 15  Performance_Score     541 non-null    float64
 16  price_per_ram         541 non-null    float64
 17  price_per_memory      541 non-null    float64
dtypes: float64(13), object(5)
memory usage: 76.2+ KB
```

- **Feature Engineering Impact**: The original dataset contained **12 features**, while the engineered dataset expanded to **18 features**, incorporating new variables like **Camera Quality Score, Performance Score, and Price per Memory** to enhance predictive power.
- **Data Type Adjustments**: Additional features were converted to **float64**, ensuring numerical consistency for model training.
- **Memory Usage Increase**: The dataset size grew from **50.8 KB to 76.2 KB**, reflecting the inclusion of more informative variables for better model accuracy.

Newly Engineered Features & Their Impact on Mobile Pricing:

The dataset introduces seven derived features to improve prediction accuracy:

Camera_Quality_Score (0.67) = Rear Camera (MP) + Front Camera (MP) → Better cameras lead to higher costs.
Performance_Score (0.70) = Processor + RAM → Higher performance increases phone prices.
Storage_Efficiency (0.06) = Memory / Mobile_Price → Storage cost-effectiveness has minimal impact on pricing.
Screen_Type (0.31) = Extracted from Processor & Model → Display technology influences phone prices.
Battery_Efficiency (-0.15) = Battery / Mobile_Height → Efficient batteries may reduce cost.
Price_per_Memory (0.54) = Mobile_Price / Memory → Storage pricing affects overall cost.
Price_per_RAM (0.54) = Mobile_Price / RAM → RAM pricing impacts phone cost.
These engineered features enhance interpretability and improve mobile price predictions.

Performing Model evaluation and accuracy assessment in machine learning

Convert categorical columns to numeric using Label Encoding

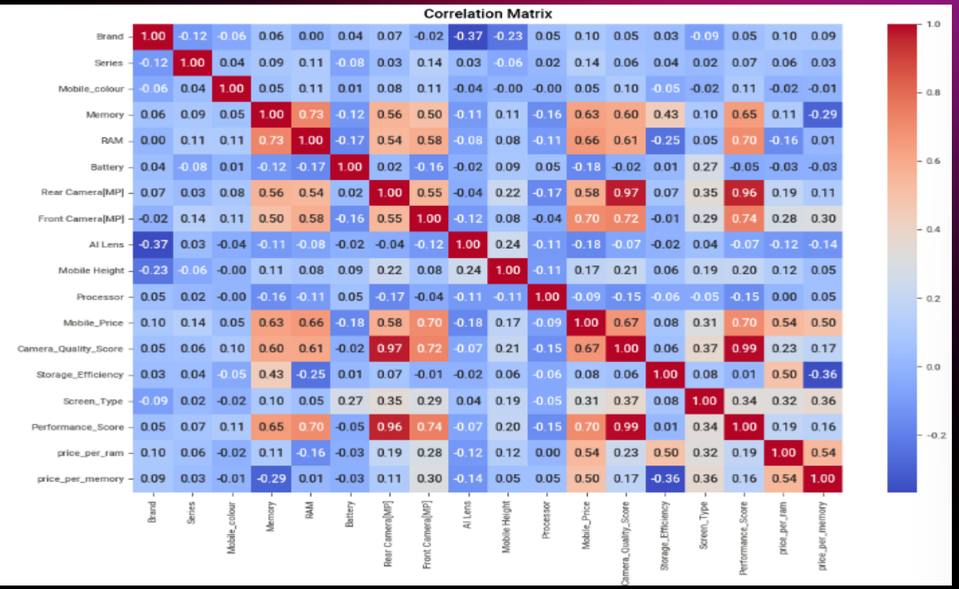
```
from sklearn.preprocessing import LabelEncoder
categorical_cols = ['Brand','Series', 'Mobile_colour','Processor','Screen_Type']

# Apply Label Encoding
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    mobile_data[col] = le.fit_transform(mobile_data[col])
    label_encoders[col] = le
```

```
x = mobile_data.drop("Mobile_Price", axis=1)  # Features
y = mobile_data["Mobile_Price"]  # Target variable
```

Splitting Data for model performance

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
Brand                    int32
Series                   int32
Mobile_colour            int32
Memory                 float64
RAM                    float64
Battery                float64
Rear Camera[MP]        float64
Front Camera[MP]       float64
AI Lens                float64
Mobile Height          float64
Processor                int32
Mobile_Price           float64
Camera_Quality_Score   float64
Storage_Efficiency     float64
Screen_Type              int32
Performance_Score      float64
price_per_ram          float64
price_per_memory       float64
dtype: object
```



Insights from the Image:

-Categorical Data Handling: The dataset contains categorical features like 'Brand', 'Series', and 'Mobile_colour', which are converted into numerical values using **Label Encoding** to make them suitable for machine learning models.

-Feature & Target Selection: The independent variables (**features**) are stored in X, while 'Mobile_Price' is set as the **target variable** (y), indicating the model aims to predict mobile prices.

-Train-Test Split: The dataset is split into training and test sets using an **80-20 ratio**, ensuring a controlled evaluation with random_state=42 for reproducibility.

-Correlation Heatmap Analysis: The heatmap provides a **color-coded view of feature relationships**, helping identify **highly correlated variables** that may influence model performance. Strong correlations suggest key predictors for 'Mobile_Price', aiding in feature selection.

Comparison of Different Machine Learning Models

1.Linear Regression

[96]:

```
###Creating Linear-Regression Model
from sklearn.linear_model import LinearRegression
model1 = LinearRegression()
model1.fit(x_train,y_train)
y_pred = model1.predict(x_test)
score = model1.score(x_test, y_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("\n Model: Linear Regression")
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
print('Mean Absolute Error:', mae)
```

```
 Model: Linear Regression
Mean Squared Error: 2519738.6969304397
R-squared: 0.9470609304377484
Mean Absolute Error: 1149.7786245533682
```

```
# Initialize Decision Tree
model2 = DecisionTreeRegressor(max_depth=6, min_samples_split=10

# Train with raw data (NO SCALING)
model2.fit(x_train, y_train)

# Predictions (Use raw data)
y_pred_train = model2.predict(x_train)
y_pred_test = model2.predict(x_test)

# Performance Metrics
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)

# Display results
print("\n Model: Decision Tree")
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Mean Absolute Error:", mae)
```

```
 Model: Decision Tree
Mean Squared Error: 798767.078970043
R-squared: 0.9832181066992602
Mean Absolute Error: 657.482699394115
```

```
model3 = RandomForestRegressor(n_estimators=300, max_depth=10, m

model3.fit(x_train, y_train)

y_pred_train = model3.predict(x_train)
y_pred_test = model3.predict(x_test)

# Performance on Training Set
mse_train = mean_squared_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)
mae_train = mean_absolute_error(y_train, y_pred_train)

# Performance Metrics
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)

print("\n Model: Random Forest")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
 Model: Random Forest
Mean Squared Error: 133545.45795727882
R-squared: 0.9971942438725351
Mean Absolute Error: 211.4801688702163
```

```
print("\n Model: Lasso")
# Calculate and print the Mean Squared Error
mse = mean_squared_error(y_test, model4_pred)
print('Mean Squared Error:', mse)

# Calculate and print the R-squared score
r2 = r2_score(y_test, model4_pred)
print('R-squared:', r2)

mae = mean_absolute_error(y_test, model4_pred)
print('Mean Absolute Error:', mae)
```

```
 Model: Lasso
Mean Squared Error: 2495370.687394698
R-squared: 0.9475728961243006
Mean Absolute Error: 1142.5900130464054
```

```
# Predictions
y_pred_train = model5.predict(x_train_scaled)
y_pred_test = model5.predict(x_test_scaled)

# Performance Metrics
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)

# Display results
print("\n Model: SVR")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
 Model: SVR
Mean Squared Error: 1571474.600020255
R-squared: 0.9669837180866694
Mean Absolute Error: 814.391861812528
```

```
y_pred_train = model6.predict(x_train_scaled)
y_pred_test = model6.predict(x_test_scaled)

# Performance Metrics
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
mae = mean_absolute_error(y_test, y_pred_test)

# Display results
print("\n Model: KNN")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
 Model: KNN
Mean Squared Error: 1691939.605427172
R-squared: 0.9644527789425341
Mean Absolute Error: 699.0266835619348
```

```
model7=GradientBoostingRegressor(n_estimators=350,learning_rate=
# Train the model
model7.fit(x_train, y_train)

# Predict
model7_pred = model7.predict(x_test)
# Performance Metrics
mse = mean_squared_error(y_test, model7_pred)
r2 = r2_score(y_test, model7_pred)
mae = mean_absolute_error(y_test, model7_pred)

# Display results
print("\n Model: Gradient Boosting")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
 Model: Gradient Boosting
Mean Squared Error: 193953.24747023522
R-squared: 0.995250915691538
Mean Absolute Error: 300.92310645358026
```

```
estimator = DecisionTreeRegressor(max_depth=4, min_samples_split
# Define AdaBoostRegressor
model8 = AdaBoostRegressor(
    estimator=estimator,  # Use 'estimator' instead of 'base_est
    n_estimators=350,
    learning_rate=0.02,
    random_state=42
)
# Train the model
model8.fit(x_train, y_train)
# Predict
model8_pred = model8.predict(x_test)
# Performance Metrics
mse = mean_squared_error(y_test, model8_pred)
r2 = r2_score(y_test, model8_pred)
mae = mean_absolute_error(y_test, model8_pred)
# Display results
print("\n Model: AdaBoost Regressor")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
 Model: AdaBoost Regressor
Mean Squared Error: 970171.374151618
R-squared: 0.9796169460245681
Mean Absolute Error: 794.9338476300087
```

```
# Define ElasticNet model
model9 = ElasticNet(
    alpha=0.1,
    l1_ratio=0.5,
    random_state=42
)
# Train the model
model9.fit(x_train, y_train)
# Predict
model9_pred = model9.predict(x_test)
# Performance Metrics
mse = mean_squared_error(y_test, model9_pred)
r2 = r2_score(y_test, model9_pred)
mae = mean_absolute_error(y_test, model9_pred)

# Display results
print("\n Model: ElasticNet Regression")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
 Model: ElasticNet Regression
Mean Squared Error: 2512766.7817931324
R-squared: 0.947207408602681
Mean Absolute Error: 1134.9103387791718
```

Comparing MAE, R2 and MSE of all the models For the final selection of Model

Model Performance Comparison:

Model	MSE	R^2 Score	MAE
Linear Regression	2,519,738.70	0.9471	1,149.78
Decision Tree	798,767.08	0.9832	657.48
Random Forest	133,545.46	0.9972	211.48
Lasso Regression	2,495,370.69	0.9476	1,142.59
SVR	1,571,474.60	0.9670	814.39
KNN	1,691,939.61	0.9645	699.03
Gradient Boosting	193,953.25	0.9959	300.92
AdaBoost Regressor	970,171.37	0.9796	794.93
ElasticNet Regression	2,512,766.78	0.9472	1,134.91

Best Model:
Model: Random Forest
Mean Squared Error: 133545.46
R-squared: 0.9972
Mean Absolute Error: 211.48

- **Random Forest** outperforms all models with the lowest **MSE (133,545.46)**, lowest **MAE (211.48)**, and highest **R^2 score (0.9972)**, indicating superior predictive accuracy.
- **Gradient Boosting** also performs well (**$R^2 = 0.9959$**) but has a slightly higher error.
- Traditional regression models like **Linear and Lasso Regression** show significantly higher errors and lower performance.
- Thus, **Random Forest is the most suitable model for this dataset**.

Mobile Price Prediction

```
final_model = RandomForestRegressor(n_estimators=100, random_state
final_model.fit(x_train, y_train)
```

[124]:

RandomForestRegressor

RandomForestRegressor(random_state=42)

[125]:

```
# Predict on test data
y_pred = final_model.predict(x_test)
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
Mean Absolute Error (MAE): 232.82
Mean Squared Error (MSE): 157558.90
Root Mean Squared Error (RMSE): 396.94
R² Score: 0.9967
```

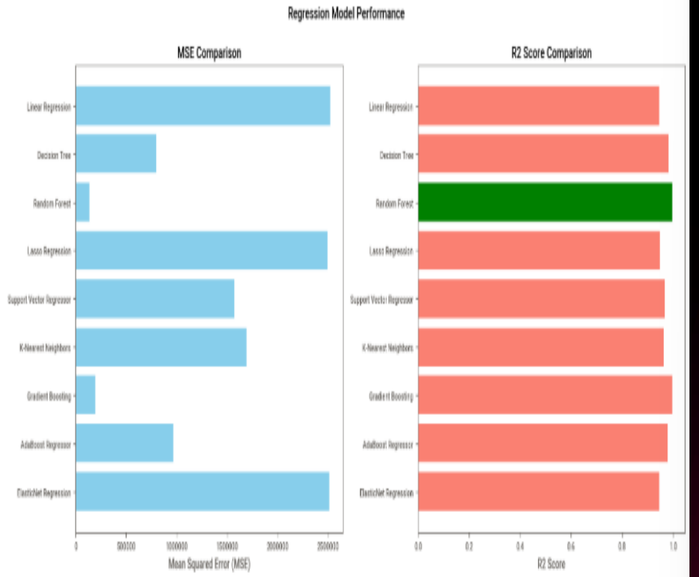
```
import numpy as np
num_features = x_train.shape[1]  # Expected feature count
# Define mobile price input with features in the correct order
mobile_price = np.array([
    21.000000,    # Brand
    172.000000,   # Series
    32.000000,    # Mobile_colour
    16.000000,    # Memory
    8.000000,     # RAM
    4871.587800,  # Battery
    2.000000,     # Rear Camera[MP]
    0.000000,     # Front Camera[MP]
    0.000000,     # AI Lens
    16.431201,    # Mobile Height
    24.000000,    # Processor
    2.000000,     # Camera_Quality_Score
    2.000000,     # Storage_Efficiency
    1.000000,     # Screen_Type
    5.717897,     # Performance_Score
    443.625000,   # price_per_ram
    221.812500    # price_per_memory
])
# Reshape it to match model input shape (1 sample, num_features)
mobile_price = mobile_price.reshape(1, num_features)
# Predict Mobile price
predicted_price = final_model.predict(mobile_price)
print(f"Predicted Price: ₹{predicted_price[0]:,.2f}")
```

```
Predicted Price: ₹3,578.45
```

```
# Predict prices on test dataset
y_pred_test = model3.predict(x_test)

# Compare actual vs predicted prices
results = pd.DataFrame({"Actual Price": y_test, "Predicted Price":
print(results.head())  # Show first few rows
```

	Actual Price	Predicted Price
229	8499.0	8498.92
73	6299.0	6814.73
352	19499.0	19280.25
86	10999.0	11344.91
470	12599.0	12834.70

Best Model: Random Forest with R2 Score: 0.9972



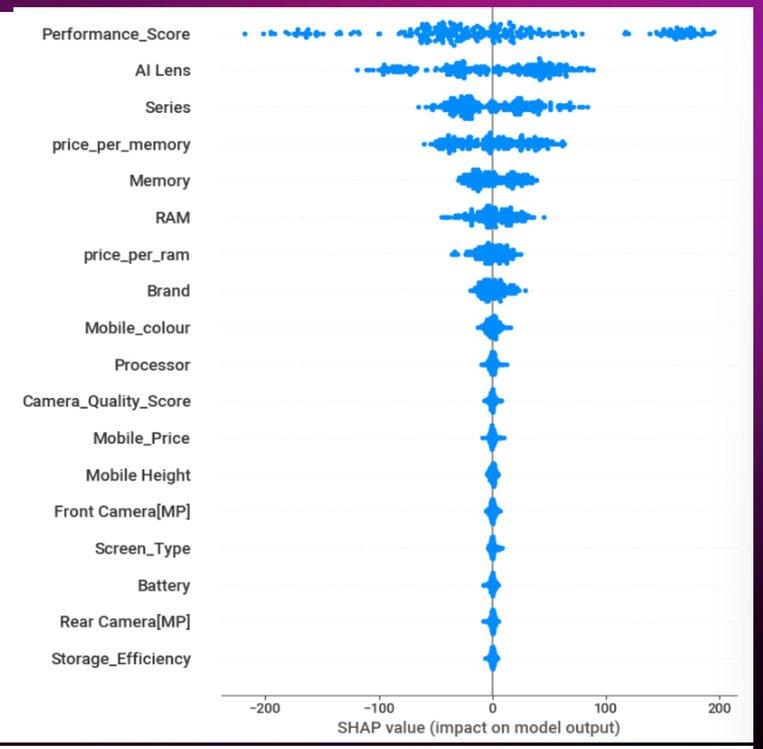
Best performing model: Random Forest with R2 Score: 0.9972

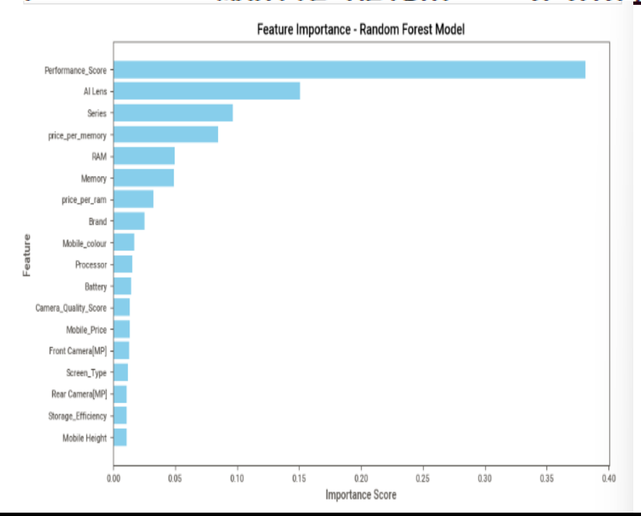
- **Random Forest Regressor** is the best-performing model with an **R^2 score of 0.9972**, indicating high accuracy in predicting mobile prices.
-The model successfully predicts prices, with minimal error between actual and predicted values, as shown in the comparison table and performance charts.
-A sample prediction using the trained model results in a **predicted price of $3,578.45**, demonstrating its effectiveness in real-world applications.

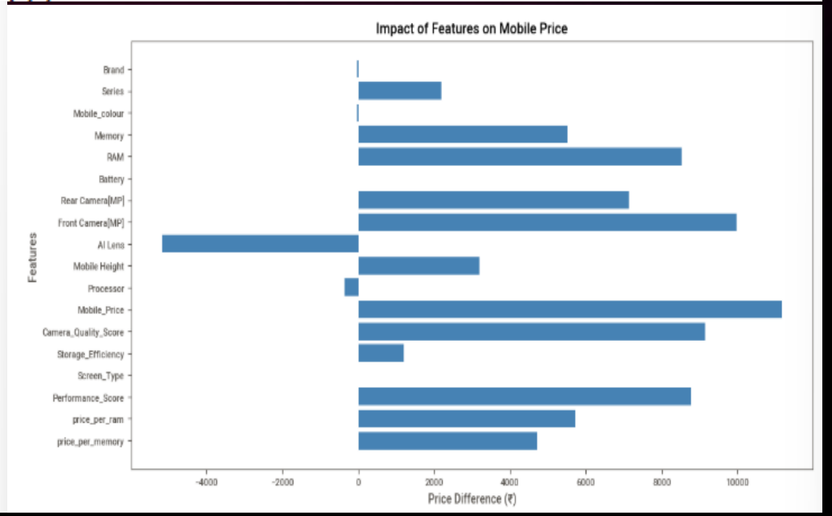
Feature Importance Analysis

Random Forest Feature Importance:

	Feature	Importance
15	Performance_Score	0.381363
8	AI Lens	0.150683
1	Series	0.096546
17	price_per_memory	0.084517
4	RAM	0.049548
3	Memory	0.048955
16	price_per_ram	0.032647
0	Brand	0.025322
2	Mobile_colour	0.016856
10	Processor	0.015484
5	Battery	0.014474
12	Camera_Quality_Score	0.013417
11	Mobile_Price	0.013182
7	Front Camera[MP]	0.012909
14	Screen_Type	0.012000
6	Rear Camera[MP]	0.010768
13	Storage_Efficiency	0.010728
9	Mobile Height	0.010599







-Performance Score is the most influential feature in predicting mobile prices, followed by **AI Lens, Series, and price per memory**, as shown by the feature importance values.

-The **SHAP plot** confirms that **Performance Score** has the highest impact on model predictions, indicating its strong correlation with mobile price variations.

-Other features like **RAM, Brand, and Battery** also contribute to price predictions but with relatively lower importance compared to top-ranking features.

Conclusion and Recommendations for Increasing Mobile Sales

Targeted Pricing Strategy:

-Focus on the **₹9,999 - ₹16,999** range, as it has the highest demand. Offer **discounts/trade-in offers** for premium models and introduce **budget-friendly options** to attract entry-level customers.

-Product Enhancement & Differentiation: Improve **AI camera features, battery life, and storage options (128GB standard for mid-range)** to meet consumer preferences. Highlight **Snapdragon-powered models** for performance-conscious buyers.

-Marketing & Brand Positioning: Aggressively promote **Vivo and Realme** due to their market dominance, leverage **influencers & tech reviews**, and introduce **limited-edition colors and stylish designs** for premium appeal.

-Feature Optimization Based on Segments:

Budget Phones: Focus on **battery life & affordability**.

Mid-Range: Enhance **camera quality & display**.

Premium: Emphasize **high-end processors, build quality, and software experience**.

References:

https://www.geeksforgeeks.org/data-analysis-visualization-python/?ref=gcse_ind
http://pandas.pydata.org/pandas-docs/stable/tutorials.html
https://www.datacamp.com
https://www.geeksforgeeks.org/data-wrangling-in-python/?ref=gcse_ind
https://www.youtube.com/watch?v=cN3i8ktEg54&list=PLq7bAUXvGrZX8OEg3UBXKT3Fr10I1vp9P&index=5
https://seaborn.pydata.org/index.html
https://www.youtube.com/watch?v=SM95HJ1CCdM&list=PLP_4EPVEox9-Vc0pXKnnso3SOIpP_6SLQ
https://builtin.com/articles/feature-engineering
https://www.geeksforgeeks.org/what-is-feature-engineering/

THANK YOU!