Feature Extraction and Price Prediction for Mobile Phones

BY:-PRIYANSHU KUMAR

DATE:- MARCH 2025

Agenda

- **PROJECT OVERVIEW**
- **DATA ANALYSIS**
- **VISUALIZATIONS**
- **FEATURE ENGINEERING**
- **MODEL BUILDING**
- **MODEL COMPARISON**
- **MOBILE PRICE PREDICTION MODEL**

Project Overview

This project, will work with a dataset that contains detailed information about various mobile phones , including their model, color, memory, RAM, battery capacity, rear camera specifications, front camera specifications, presence of AI lens, mobile height, processor, and, most importantly, the price. Your primary goal is to develop a predictive model for mobile phones prices.

Data Analysis

Data Overview

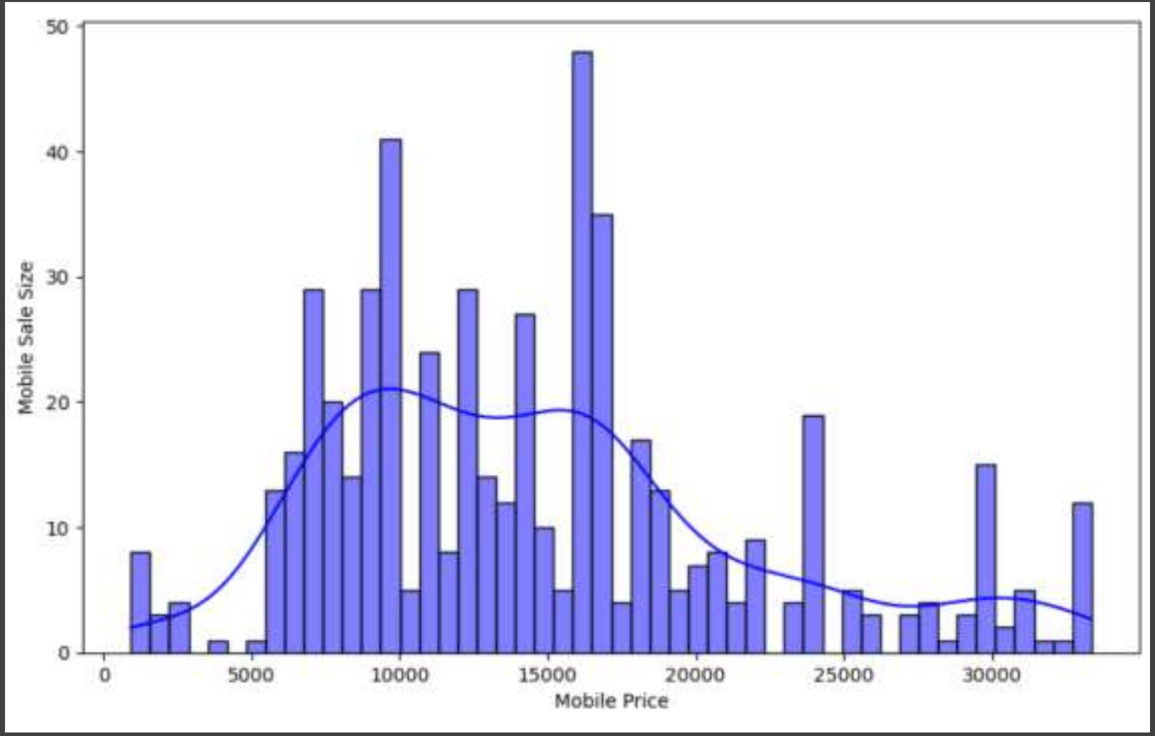
- **Total Entries:** 541
- **Total Features:** 12
- **Data Types:**
  - **Numerical Features:** 6
  - **Categorical Features:** 5
- **Target Variable:** Mobile SalesPrice
- **Key Objective:** Predict mobile prices based on Their Model , Colour , RAM, Battery size , Processor , Camera etc.

Data Exploration

Missing Value /NAN Value Analysis

- **No columns have missing/nan values.**
- **Unnamed: 0**$\rightarrow$ Dropped due to the unused column

Distribution Of Sales Price



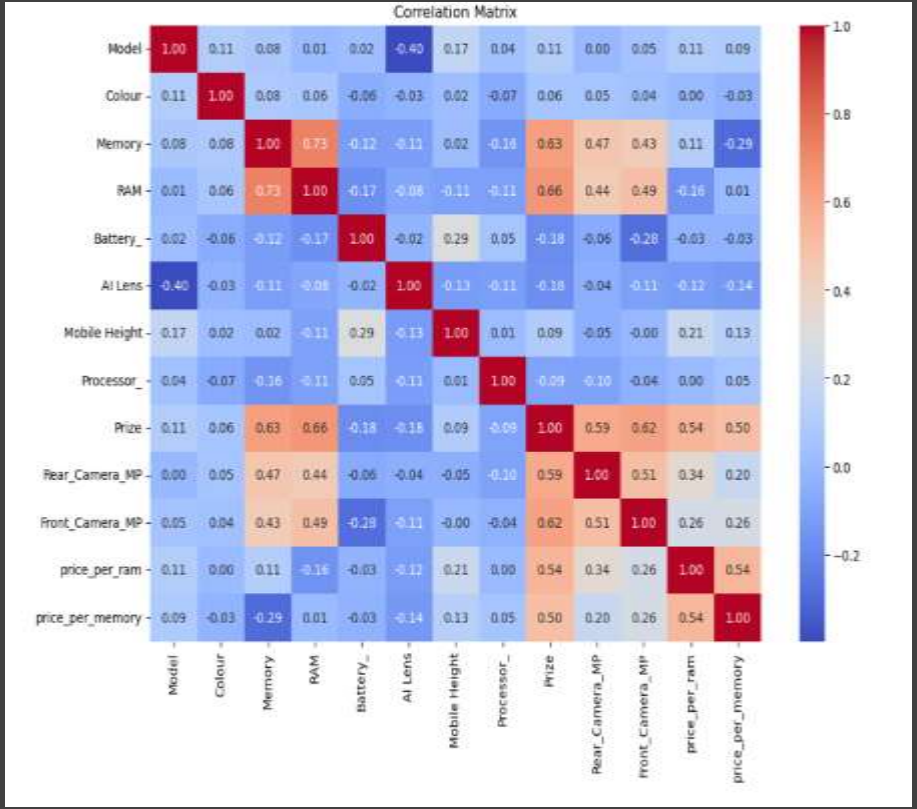
Distribution Shape:

- The distribution appears **right-skewed** (positively skewed).
- This suggests that while most Mobile have relatively higher prices.

Peaks & Mode:

- The distribution has multiple peaks, indicating that sale prices are **not normally distributed**.
- The most frequent sale price range appears to be around **15000-16000**, where the highest bar is observed.

Strong correlations with sale price



	Model	Colour	Memory	RAM	Battery_	AI Lens	Mobile Height	Processor_	Prize	Rear_Camera_MP	Front_Camera_MP	price_per_ram	price_per_memory
Model	1.00	0.11	0.08	0.01	0.02	-0.40	0.17	0.04	0.11	0.00	0.05	0.11	0.09
Colour	0.11	1.00	0.08	0.06	-0.06	-0.03	0.02	-0.07	0.06	0.05	0.04	0.00	-0.03
Memory	0.08	0.08	1.00	0.73	-0.12	-0.11	0.02	-0.16	0.63	0.47	0.43	0.11	-0.29
RAM	0.01	0.06	0.73	1.00	-0.17	-0.08	-0.11	-0.11	0.66	0.44	0.49	-0.16	0.01
Battery_	0.02	-0.06	-0.12	-0.17	1.00	-0.02	0.29	0.05	-0.18	-0.06	-0.28	-0.03	-0.03
AI Lens	-0.40	-0.03	-0.11	-0.08	-0.02	1.00	-0.13	-0.11	-0.18	-0.04	-0.11	-0.12	-0.14
Mobile Height	0.17	0.02	0.02	-0.11	0.29	-0.13	1.00	0.01	0.09	-0.05	-0.00	0.21	0.13
Processor_	0.04	-0.07	-0.16	-0.11	0.05	-0.11	0.01	1.00	-0.09	-0.10	-0.04	0.00	0.05
Prize	0.11	0.06	0.63	0.66	-0.18	-0.18	0.09	-0.09	1.00	0.59	0.62	0.54	0.50
Rear_Camera_MP	0.00	0.05	0.47	0.44	-0.06	-0.04	-0.05	-0.10	0.59	1.00	0.51	0.34	0.20
Front_Camera_MP	0.05	0.04	0.43	0.49	-0.28	-0.11	-0.00	-0.04	0.62	0.51	1.00	0.26	0.26
price_per_ram	0.11	0.00	0.11	-0.16	-0.03	-0.12	0.21	0.00	0.54	0.34	0.26	1.00	0.54
price_per_memory	0.09	-0.03	-0.29	0.01	-0.03	-0.14	0.13	0.05	0.50	0.20	0.26	0.54	1.00

Key Insight Of Heat Map

RAM (≈ 0.73): Higher RAM is associated with higher mobile prices.

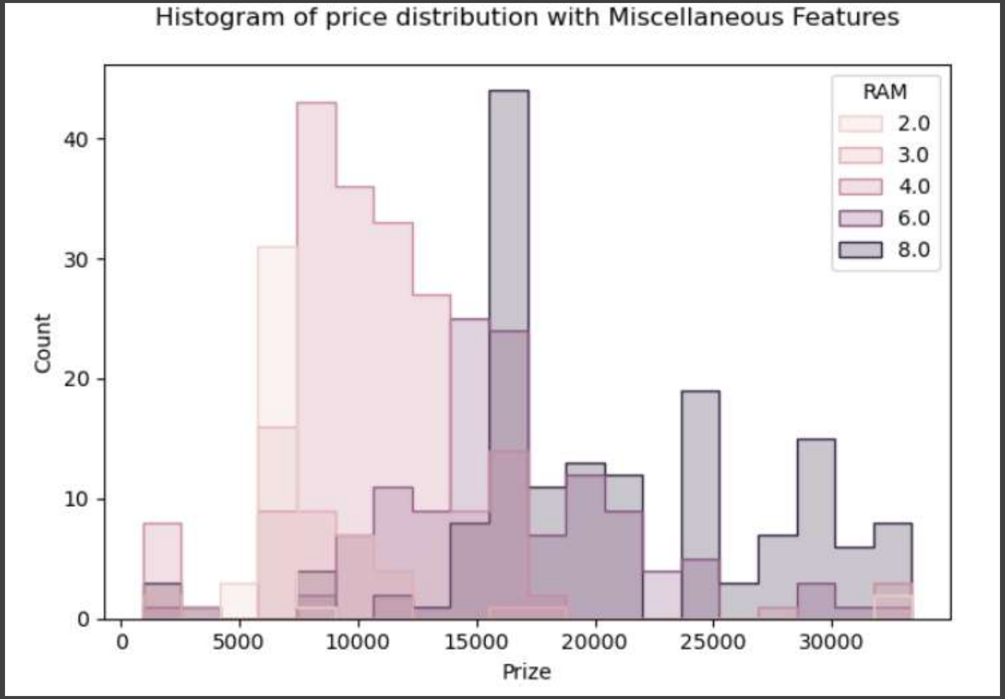
Memory (≈ 0.63): More storage leads to a higher price.

Processor (≈ 0.50): The processor type has a moderate impact on price.

Rear Camera MP & Front Camera MP (≈ 0.20 - 0.26): Slightly influences price but not as strongly as RAM and storage**.**

RAM & Memory (≈ 0.73): Higher RAM is often found in phones with higher storage.

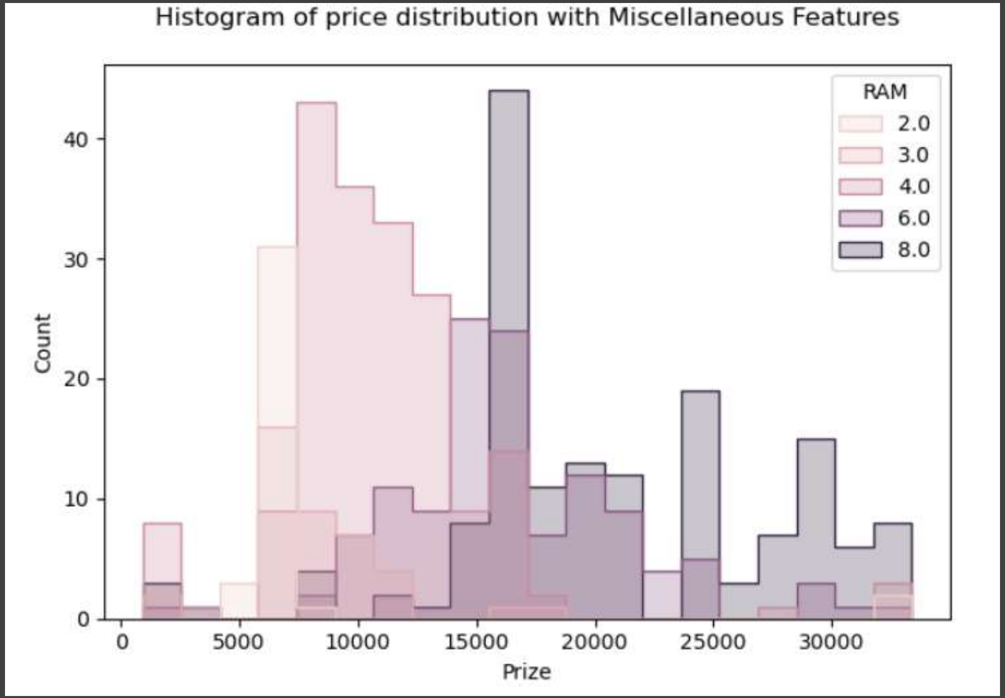
Price distribution with miscellaneous feature



Key Insight Of Graph

- The majority of mobile phones are priced between ₹5,000 to ₹20,000, with fewer models beyond ₹25,000.
- Lower RAM models (2GB, 3GB, 4GB) are more frequent in the lower price range (₹5,000 - ₹15,000).
- Higher RAM models (6GB, 8GB) are more concentrated in the premium price segment (₹15,000 - ₹30,000+).

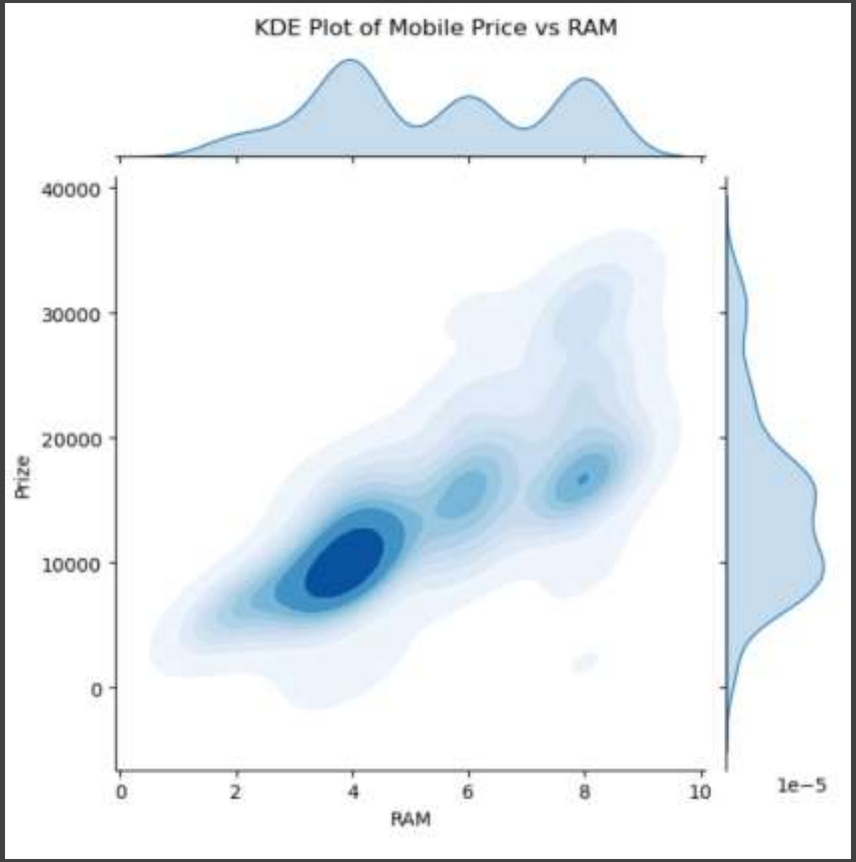
Price distribution with miscellaneous feature



Key Insight Of Graph

- The majority of mobile phones are priced between ₹5,000 to ₹20,000, with fewer models beyond ₹25,000.
- Lower RAM models (2GB, 3GB, 4GB) are more frequent in the lower price range (₹5,000 - ₹15,000).
- Higher RAM models (6GB, 8GB) are more concentrated in the premium price segment (₹15,000 - ₹30,000+).

KDE Plot (RAM vs Mobile Price)



Key Insights from the KDE Plot (RAM vs Mobile Price)

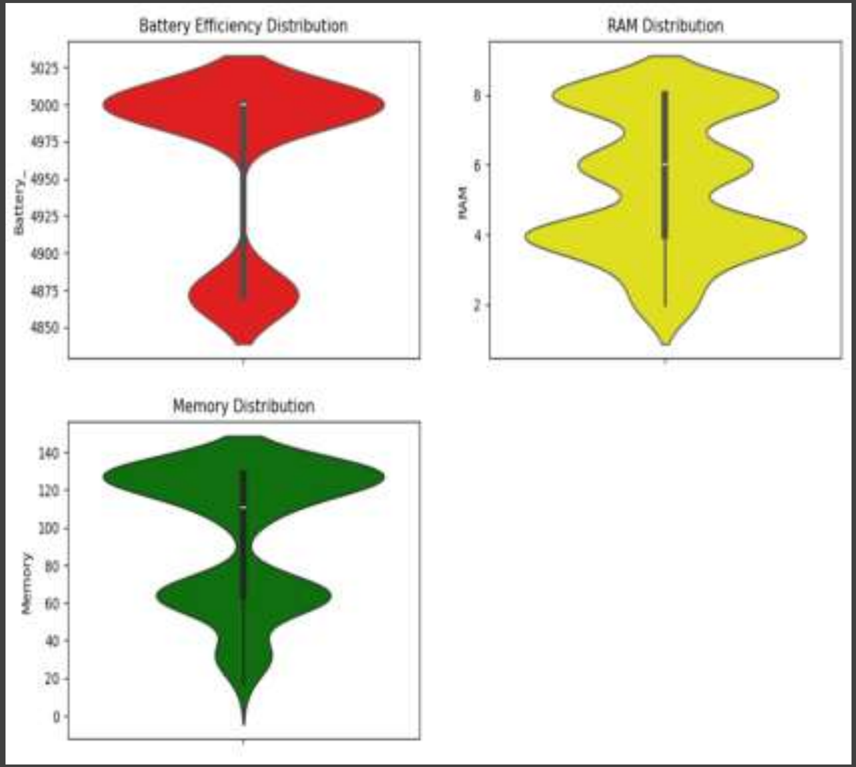
1.Positive Correlation Between RAM and Price
The KDE plot shows that as RAM increases, the mobile price also tends to increase.
This suggests that higher RAM models are generally priced higher.

2.High-Density Region
The darkest region (highest density) is around 2GB to 4GB RAM and price between 5,000 to 15,000.
This indicates that most mobile phones fall within this range, making it the most common segment in the dataset.

3.Less Density in Higher RAM & Price
As RAM increases beyond 6GB-8GB, the distribution spreads out, meaning fewer phones have very high RAM and those that do are priced higher.However, the density is lower for very high-end devices, suggesting that flagship models (high RAM & price) are less frequent.

4.Price Variability at Lower RAM
At lower RAM values (0GB - 2GB), the price distribution spreads widely, meaning there are both budget and slightly expensive models in this category.This suggests that other features (brand, processor, etc.) might also play a role in determining price at lower RAM levels.

Violin Plots(Battery , RAM , Memory)



Key Insights from the Violin Plots

1. Battery Efficiency Distribution (Red)

The distribution appears bimodal (two peaks), indicating two common ranges of battery efficiency.
The middle part is relatively narrow, meaning most values cluster around a certain range.
There are some outliers at the lower end, suggesting a few mobile phones with significantly lower battery efficiency.

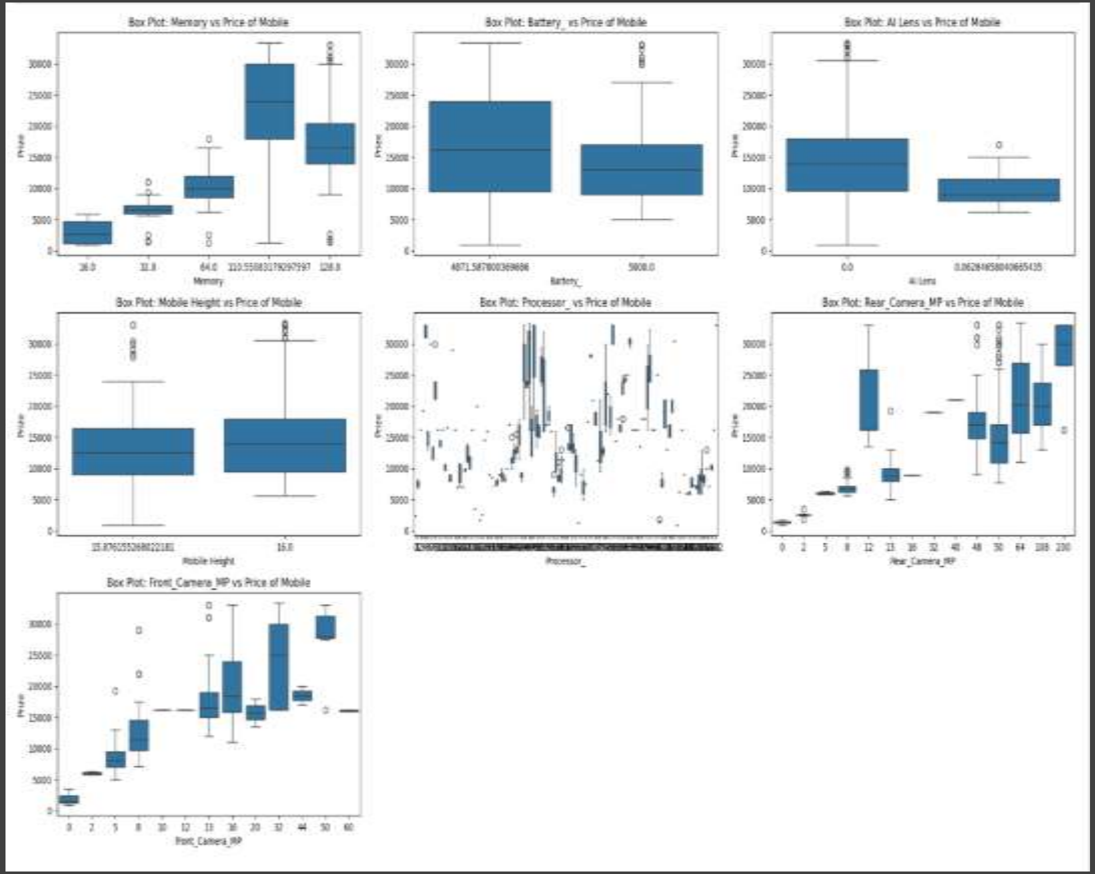
2. RAM Distribution (Yellow)

The plot is multi-modal, meaning there are distinct groups of RAM capacities.
The wider sections indicate more common RAM values (e.g., 4GB, 6GB, 8GB).
The lower part of the violin is narrow, meaning fewer low-RAM devices in the dataset.

3. Memory Distribution (Green)

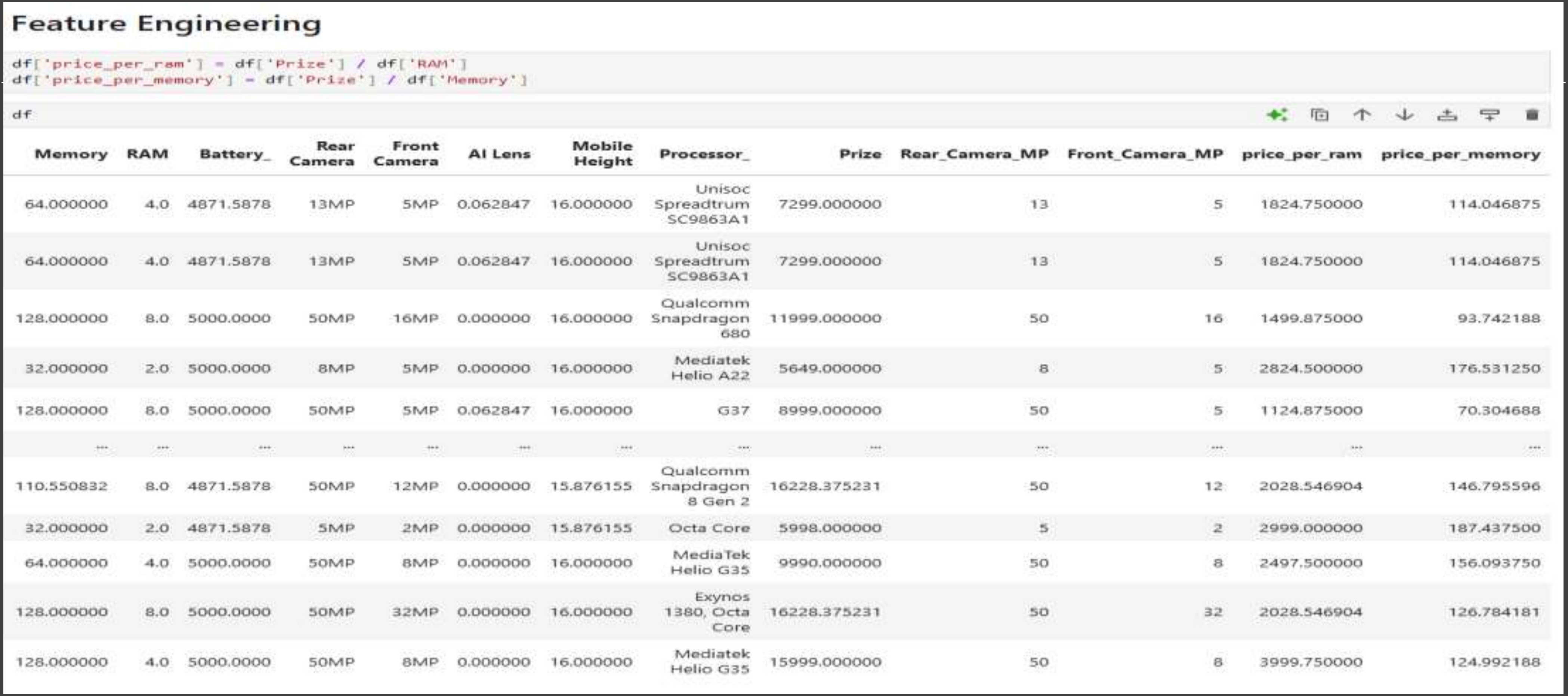
The plot has two distinct peaks, indicating that mobile phones tend to have two common memory capacity ranges (possibly lower-end and higher-end models).
This suggests that most phones either have low or high memory, with fewer in the middle range.

Mobile Price Vs Features



- Limited data points for AI Lens.
- No clear pattern in pricing based on AI Lens presence.
- This feature may not significantly impact mobile pricing.
- Higher variance in price for different processors.
- Some specific processors are associated with high-end models.
- Premium phones tend to use flagship processors.
- Taller phones might be premium models, but not a decisive factor.

Feature Engineering (New Columns)



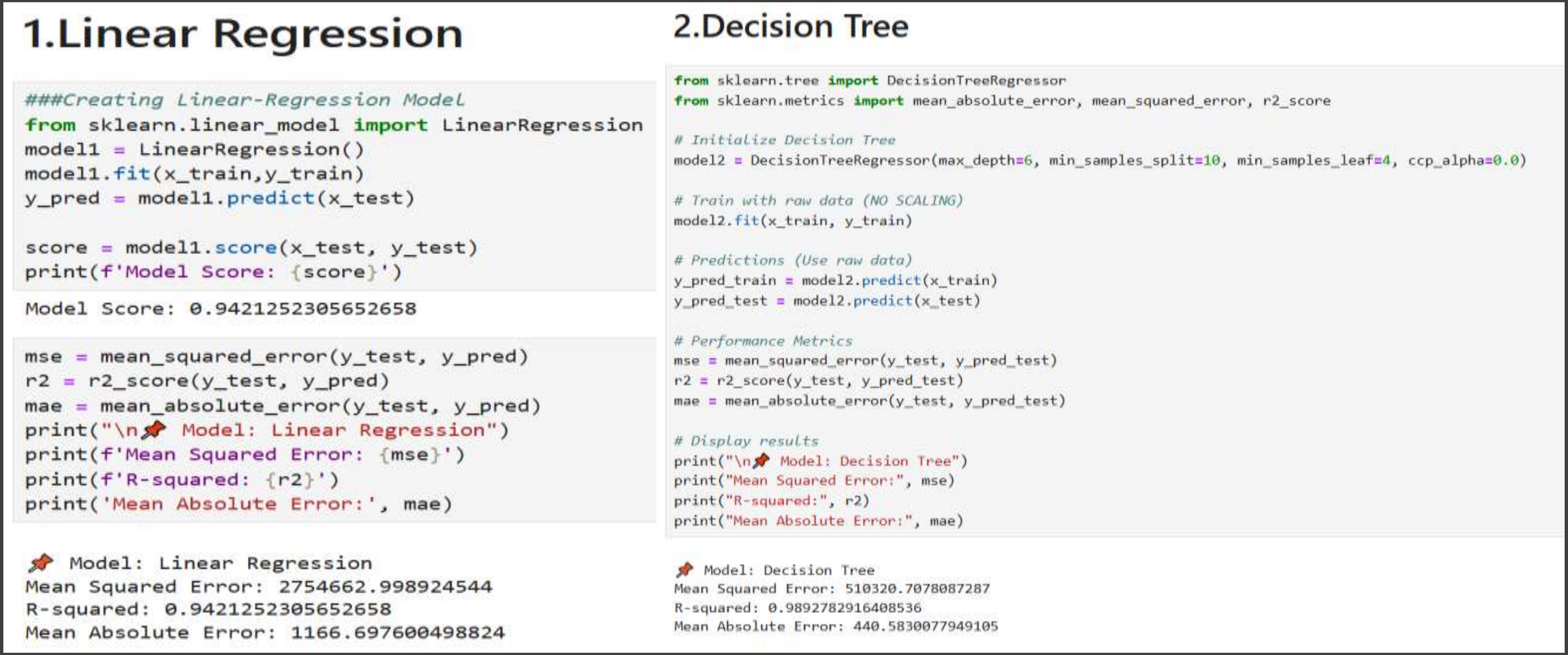
Feature Engineering

```
df['price_per_ram'] = df['Prize'] / df['RAM']
df['price_per_memory'] = df['Prize'] / df['Memory']
```

```
df
```

Memory	RAM	Battery_	Rear Camera	Front Camera	AI Lens	Mobile Height	Processor_	Prize	Rear_Camera_MP	Front_Camera_MP	price_per_ram	price_per_memory
64.000000	4.0	4871.5878	13MP	5MP	0.062847	16.000000	Unisoc Spreadtrum SC9863A1	7299.000000	13	5	1824.750000	114.046875
64.000000	4.0	4871.5878	13MP	5MP	0.062847	16.000000	Unisoc Spreadtrum SC9863A1	7299.000000	13	5	1824.750000	114.046875
128.000000	8.0	5000.0000	50MP	16MP	0.000000	16.000000	Qualcomm Snapdragon 680	11999.000000	50	16	1499.875000	93.742188
32.000000	2.0	5000.0000	8MP	5MP	0.000000	16.000000	Mediatek Helio A22	5649.000000	8	5	2824.500000	176.531250
128.000000	8.0	5000.0000	50MP	5MP	0.062847	16.000000	G37	8999.000000	50	5	1124.875000	70.304688
...	...	...	...	...	...	...	...	...	...	...	...	...
110.550832	8.0	4871.5878	50MP	12MP	0.000000	15.876155	Qualcomm Snapdragon 8 Gen 2	16228.375231	50	12	2028.546904	146.795596
32.000000	2.0	4871.5878	5MP	2MP	0.000000	15.876155	Octa Core	5998.000000	5	2	2999.000000	187.437500
64.000000	4.0	5000.0000	50MP	8MP	0.000000	16.000000	MediaTek Helio G35	9990.000000	50	8	2497.500000	156.093750
128.000000	8.0	5000.0000	50MP	32MP	0.000000	16.000000	Exynos 1380, Octa Core	16228.375231	50	32	2028.546904	126.784181
128.000000	4.0	5000.0000	50MP	8MP	0.000000	16.000000	Mediatek Helio G35	15999.000000	50	8	3999.750000	124.992188

Model Building



1.Linear Regression

```
###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)
print(f'Model Score: {score}')
```

```
Model Score: 0.9421252305652658
```

```
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)
```

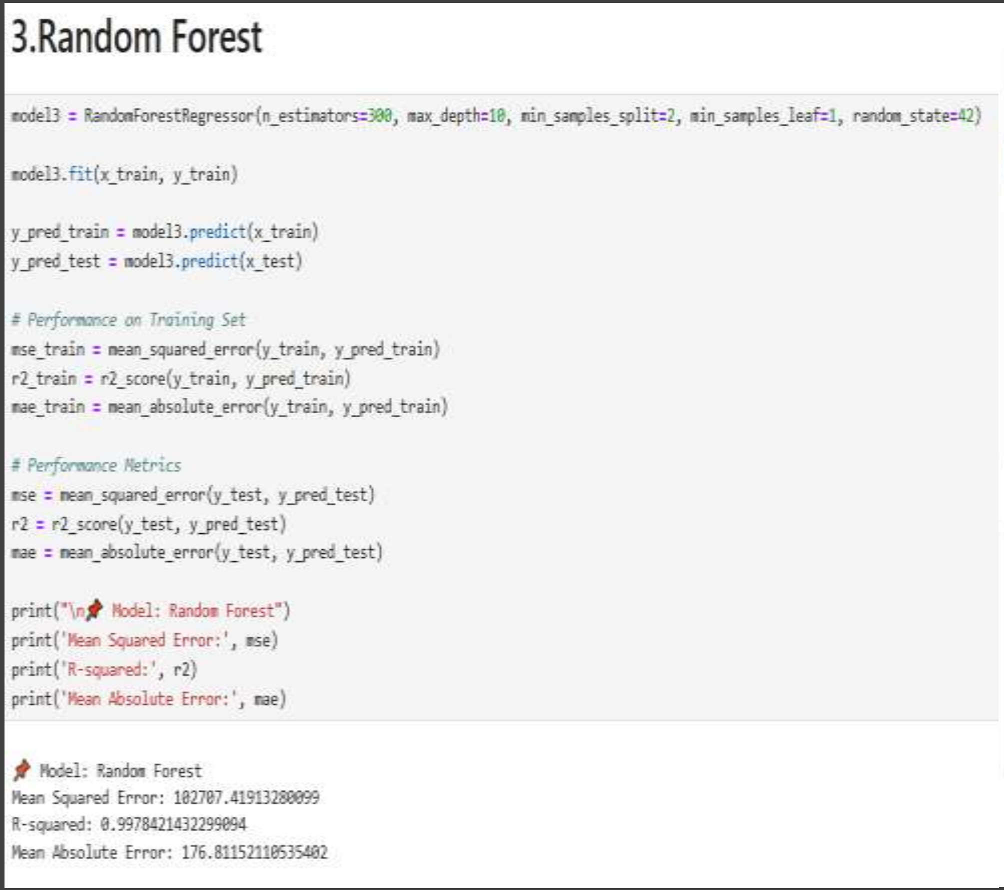
```
📌 Model: Linear Regression
Mean Squared Error: 2754662.998924544
R-squared: 0.9421252305652658
Mean Absolute Error: 1166.697600498824
```

2.Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Initialize Decision Tree
model2 = DecisionTreeRegressor(max_depth=6, min_samples_split=10, min_samples_leaf=4, ccp_alpha=0.0)

# 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: 510320.7078087287
R-squared: 0.9892782916408536
Mean Absolute Error: 440.5830077949105
```

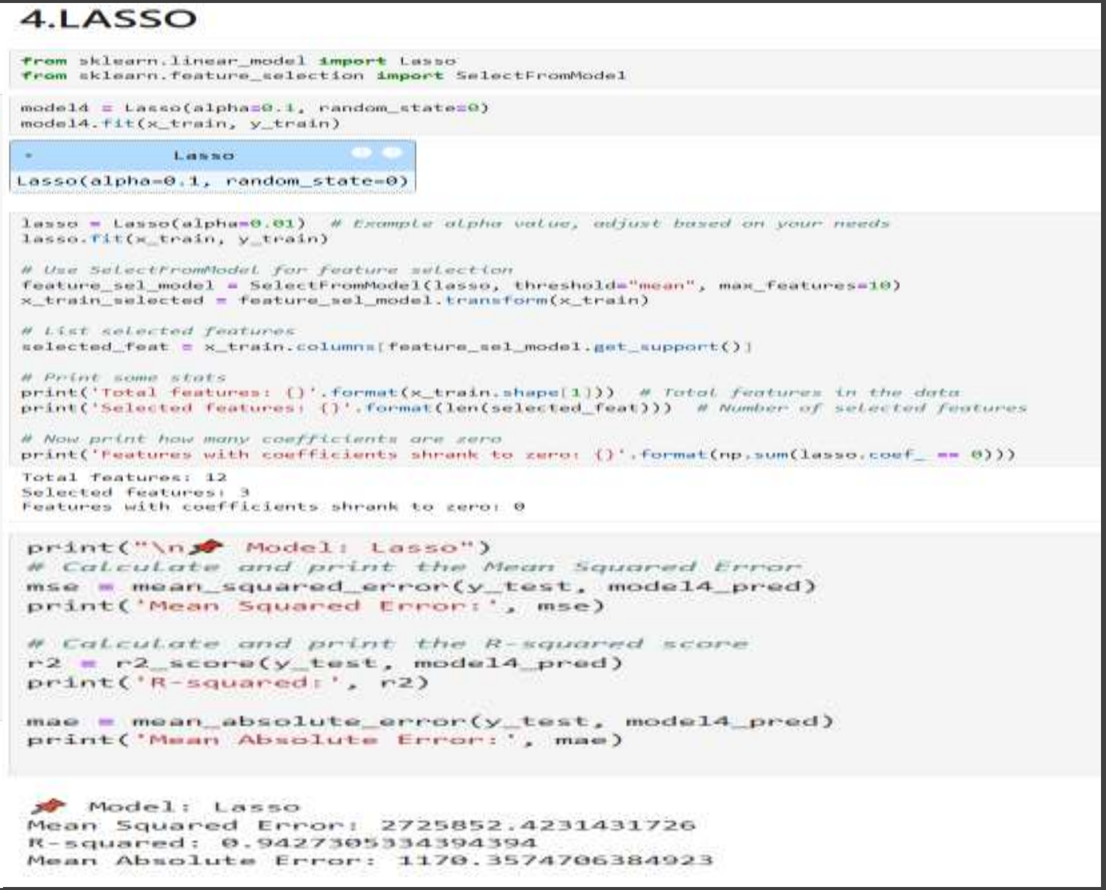
Model Building



3.Random Forest

```
model3 = RandomForestRegressor(n_estimators=300, max_depth=10, min_samples_split=2, min_samples_leaf=1, random_state=42)

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: 102707.41913280099
R-squared: 0.9978421432299094
Mean Absolute Error: 176.81152110535402
```



4.LASSO

```
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel
```

```
model4 = Lasso(alpha=0.1, random_state=0)
model4.fit(x_train, y_train)
```

```
Lasso
Lasso(alpha=0.1, random_state=0)
```

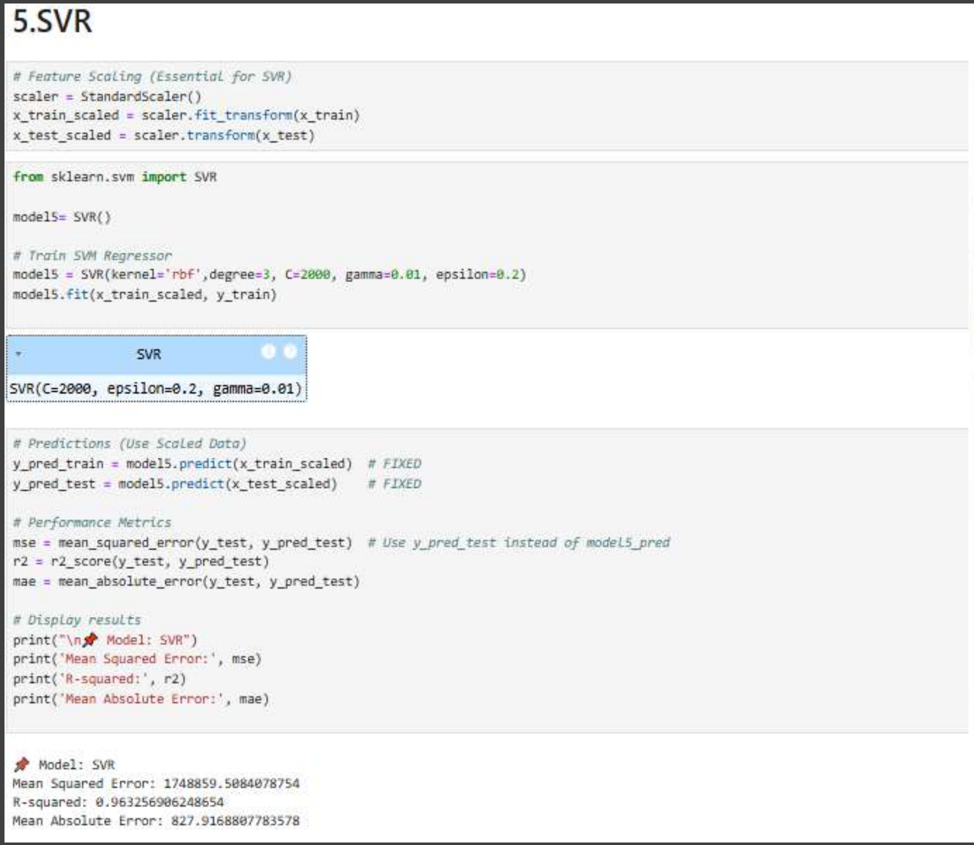
```
lasso = Lasso(alpha=0.01)  # Example alpha value, adjust based on your needs
lasso.fit(x_train, y_train)

# Use SelectFromModel for feature selection
feature_sel_model = SelectFromModel(lasso, threshold="mean", max_features=10)
x_train_selected = feature_sel_model.transform(x_train)

# List selected features
selected_feat = x_train.columns[feature_sel_model.get_support()]

# Print some stats
print('Total features: {}'.format(x_train.shape[1]))  # Total features in the data
print('Selected features: {}'.format(len(selected_feat)))  # Number of selected features

# Now print how many coefficients are zero
print('Features with coefficients shrank to zero: {}'.format(np.sum(lasso.coef_ == 0)))
```

```
Total features: 12
Selected features: 3
Features with coefficients shrank to zero: 0
```

```
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: 2725852.4231431726
R-squared: 0.9427305334394394
Mean Absolute Error: 1170.3574706384923
```

Model Building



5.SVR

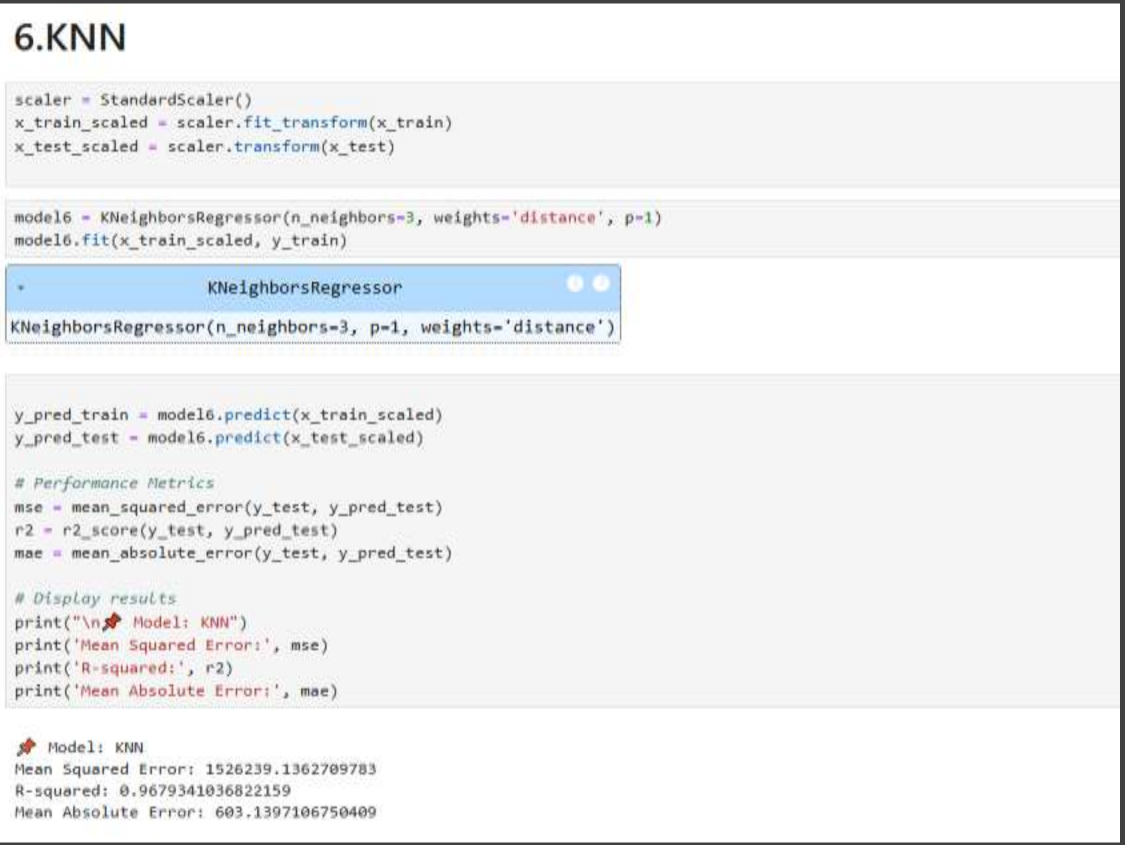
```
# Feature Scaling (Essential for SVR)
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
from sklearn.svm import SVR

model5= SVR()

# Train SVM Regressor
model5 = SVR(kernel='rbf',degree=3, C=2000, gamma=0.01, epsilon=0.2)
model5.fit(x_train_scaled, y_train)
```

```
SVR(C=2000, epsilon=0.2, gamma=0.01)
```

```
# Predictions (Use Scaled Data)
y_pred_train = model5.predict(x_train_scaled)  # FIXED
y_pred_test = model5.predict(x_test_scaled)    # FIXED

# Performance Metrics
mse = mean_squared_error(y_test, y_pred_test)  # Use y_pred_test instead of model5_pred
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: 1748859.5084078754
R-squared: 0.963256906248654
Mean Absolute Error: 827.9168807783578
```



6.KNN

```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

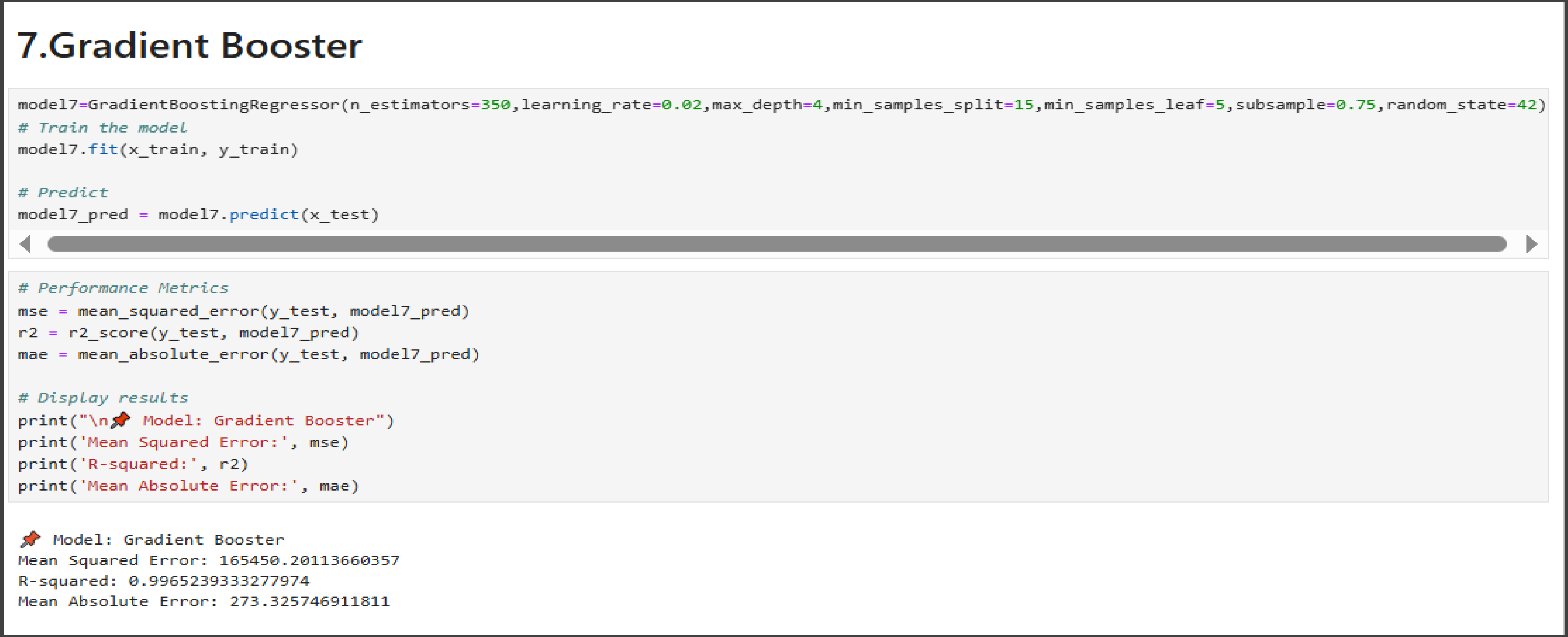
```
model6 = KNeighborsRegressor(n_neighbors=3, weights='distance', p=1)
model6.fit(x_train_scaled, y_train)
```

```
KNeighborsRegressor(n_neighbors=3, p=1, weights='distance')
```

```
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: 1526239.1362709783
R-squared: 0.9679341036822159
Mean Absolute Error: 603.1397106750409
```

Model Building



7.Gradient Booster

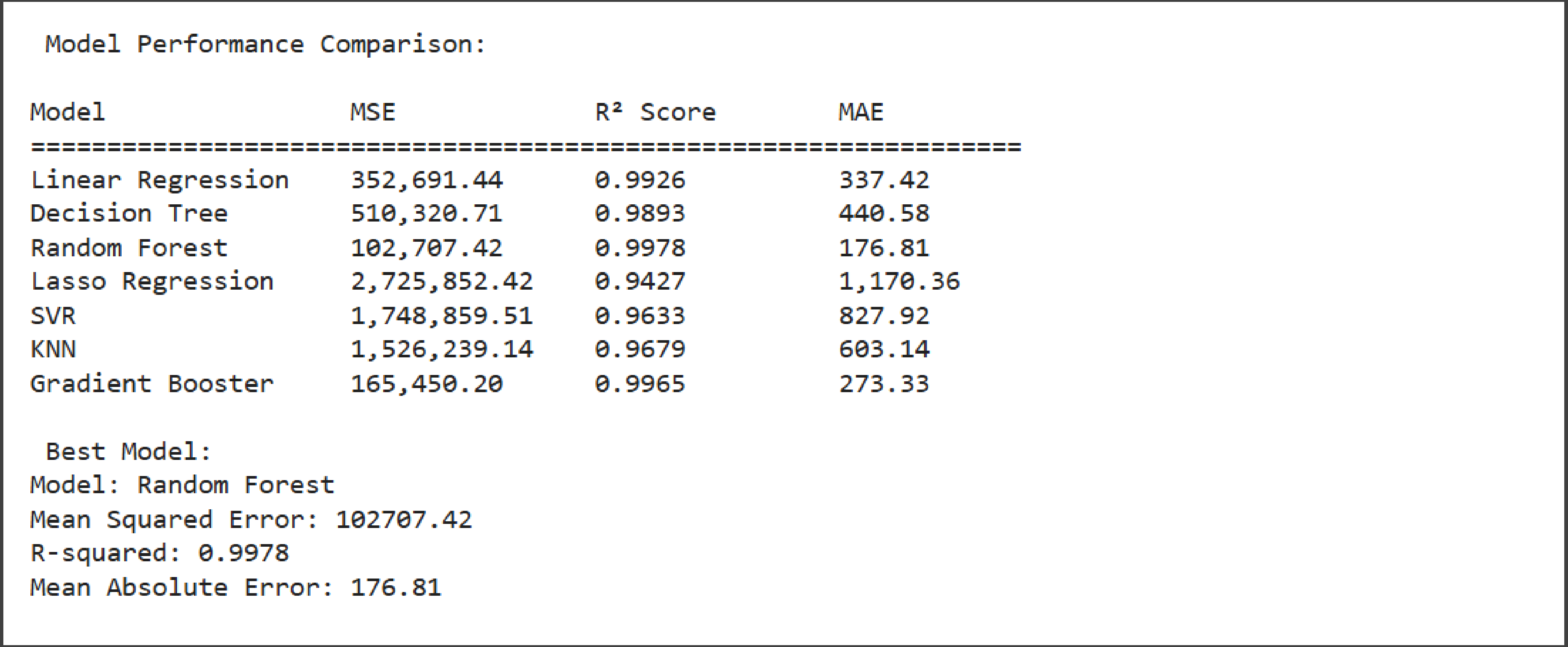
```
model7=GradientBoostingRegressor(n_estimators=350,learning_rate=0.02,max_depth=4,min_samples_split=15,min_samples_leaf=5,subsample=0.75,random_state=42)
# Train the model
model7.fit(x_train, y_train)

# 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 Booster")
print('Mean Squared Error:', mse)
print('R-squared:', r2)
print('Mean Absolute Error:', mae)
```

```
📌 Model: Gradient Booster
Mean Squared Error: 165450.20113660357
R-squared: 0.9965239333277974
Mean Absolute Error: 273.325746911811
```

MODEL COMPARISON

Model comparison Table



```
 Model Performance Comparison:

Model                    MSE              R² Score        MAE
=================================================================
Linear Regression        352,691.44       0.9926          337.42
Decision Tree            510,320.71       0.9893          440.58
Random Forest            102,707.42       0.9978          176.81
Lasso Regression         2,725,852.42     0.9427          1,170.36
SVR                      1,748,859.51     0.9633          827.92
KNN                      1,526,239.14     0.9679          603.14
Gradient Booster         165,450.20       0.9965          273.33

 Best Model:
Model: Random Forest
Mean Squared Error: 102707.42
R-squared: 0.9978
Mean Absolute Error: 176.81
```

Model Comparison

Key Insights

Random Forest - The Top Performer

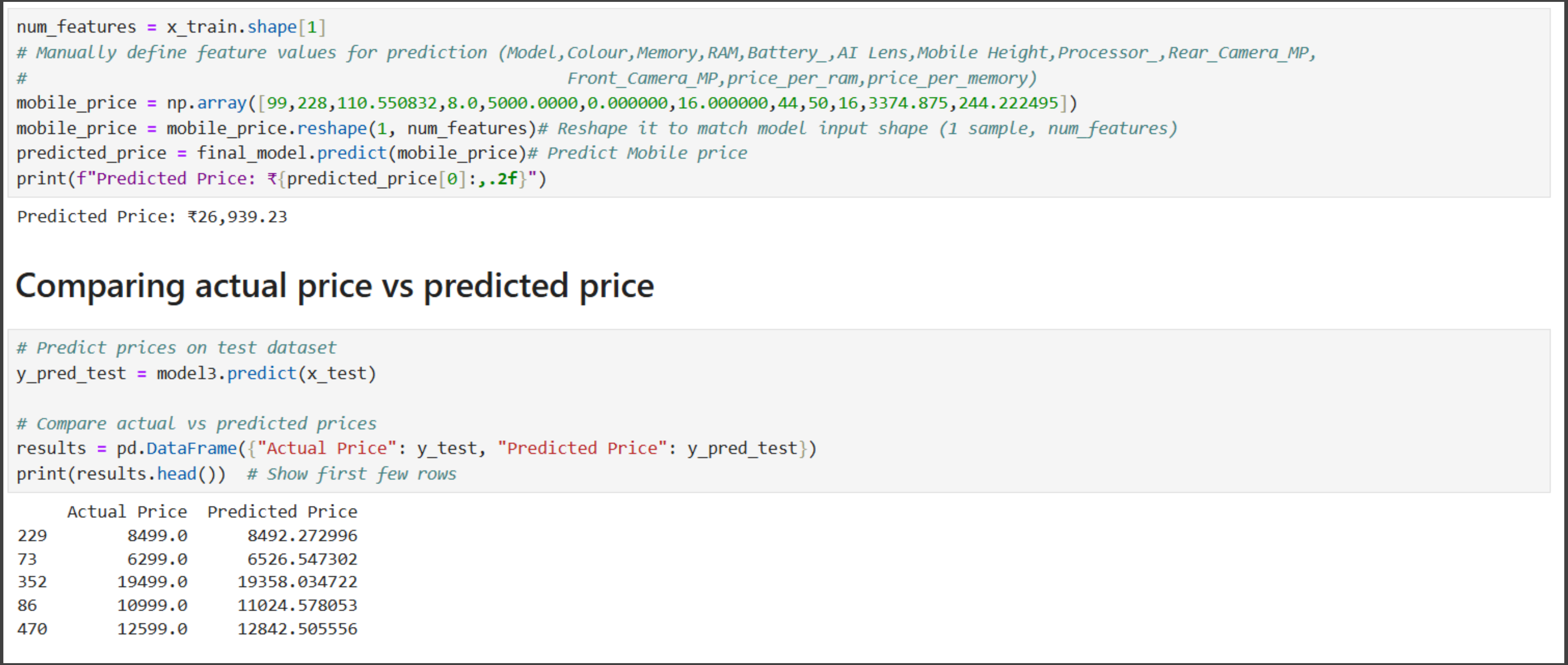
- Mean squared error =102707.42
- R-Squared = 0.9978
- Mean absolute error = 176.81

Insight:

•**Random Forest** stands out as the best model based on **MAE**, **RMSE**, and **MSE**. It minimizes errors effectively, showing good generalization and stability.

•The low **MSE** indicates minimal prediction variance, and the low **MAE** suggests that the model makes predictions that are close to the actual values on average.

House Price Prediction Model And Comparing Actual vs Predicted Price



```
num_features = x_train.shape[1]
# Manually define feature values for prediction (Model,Colour,Memory,RAM,Battery_,AI Lens,Mobile Height,Processor_,Rear_Camera_MP,
#                                                Front_Camera_MP,price_per_ram,price_per_memory)
mobile_price = np.array([99,228,110.550832,8.0,5000.0000,0.000000,16.000000,44,50,16,3374.875,244.222495])
mobile_price = mobile_price.reshape(1, num_features)# Reshape it to match model input shape (1 sample, num_features)
predicted_price = final_model.predict(mobile_price)# Predict Mobile price
print(f"Predicted Price: ₹{predicted_price[0]:,.2f}")
```

```
Predicted Price: ₹26,939.23
```

Comparing actual price vs predicted price

```
# 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": y_pred_test})
print(results.head())  # Show first few rows
```

```
     Actual Price  Predicted Price
229        8499.0      8492.272996
73         6299.0      6526.547302
352       19499.0     19358.034722
86        10999.0     11024.578053
470       12599.0     12842.505556
```

Thank You

THANK YOU