

APPLIED MACHINE LEARNING

Submitted In partial fulfilment of requirements for the degree of

Master of Technology

(AI & DS)

SUBMITTED BY:

SANUJA CHAKRABORTY

24167016

UNDER THE GUIDENCE OF

Bapuji Kanaparthi

Industry Instructor



SCHOOL OF COMPUTER ENGINEERING

Kalinga Institute of Industrial Technolgy

Deemed to be University

Bhubaneswar-751024

2024

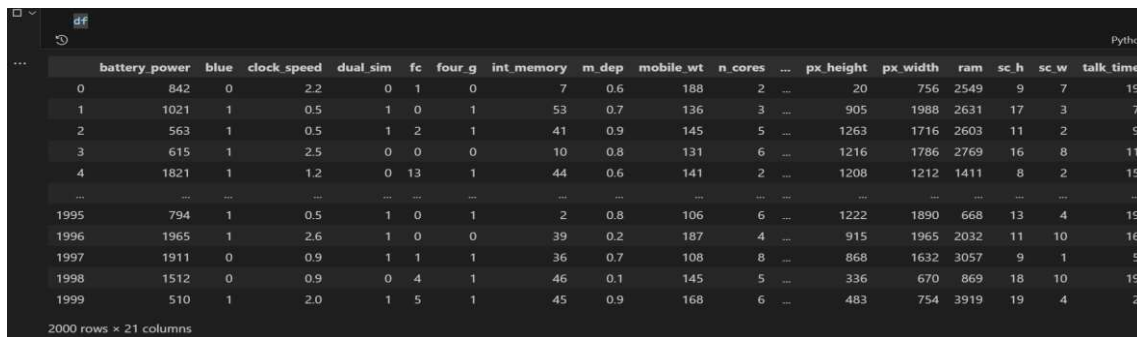
Experiment No	Experiment Details	Page No.
Experiment 1	Understanding "Mobile Price" dataset by doing feature analysis. Data is available at: https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification/data	3-6
Experiment 2	Execute data preprocessing step on the above dataset: perform outlier and missing data analysis towards building a refined dataset	7-8
Experiment 3	Build machine learning model/s to predict the actual price of the new mobile based on other given features like RAM, Internal Memory etc	9-11
Experiment 4	Calculate the prediction accuracy of the models used in Experiment 3 and do comparative analysis among them to identify the best technique.	12-13
Experiment 5	Understanding "Second Hand Car Prediction Price" dataset by doing feature analysis. Data is available at: https://www.kaggle.com/datasets/sujithmandala/second-hand-car-priceprediction	14-16
Experiment 6	Perform data preprocessing step on the above dataset: perform outlier and missing data analysis towards building a refined dataset	17-18
Experiment 7	Perform Feature Engineering towards building new feature which is more impactful. Build machine learning model/s to predict the price of the car based on other given features like Brand, Model, Year, Fuel Type etc	19
Experiment 8	Calculate the prediction accuracy of the models used in Experiment 7 and do comparative analysis among them to identify the best technique.	20
Experiment 9	Plot the features (actual price and predicted price) in scatter plot to understand the variation.	21-24

Experiment 1

Objective: Understanding "Mobile Price" dataset by doing feature analysis.
Data is available at: <https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification/data>

First, we must extract the dataset using Pandas library.

The dataset we downloaded is shown:



	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	0	2.2	0	1	0	7	0.6	188	2	20	756	2549	9	7	19
1	1021	1	0.5	1	0	1	53	0.7	136	3	905	1988	2631	17	3	7
2	563	1	0.5	1	2	1	41	0.9	145	5	1263	1716	2603	11	2	9
3	615	1	2.5	0	0	0	10	0.8	131	6	1216	1786	2769	16	8	11
4	1821	1	1.2	0	13	1	44	0.6	141	2	1208	1212	1411	8	2	15
...
1995	794	1	0.5	1	0	1	2	0.8	106	6	1222	1890	668	13	4	19
1996	1965	1	2.6	1	0	0	39	0.2	187	4	915	1965	2032	11	10	16
1997	1911	0	0.9	1	1	1	36	0.7	108	8	868	1632	3057	9	1	5
1998	1512	0	0.9	0	4	1	46	0.1	145	5	336	670	869	18	10	19
1999	510	1	2.0	1	5	1	45	0.9	168	6	483	754	3919	19	4	2

Now we will start EDA (Exploratory Data Analysis):

The describe() function in Python's Pandas library provides a concise summary of the numerical data in a DataFrame or Series.

```
print(df.describe())
```

	battery_power	blue	clock_speed	dual_sim	fc
count	2000.000000	2000.0000	2000.000000	2000.000000	2000.000000
mean	1238.518500	0.4950	1.522250	0.509500	4.309500
std	439.418206	0.5001	0.816004	0.500035	4.341444
min	501.000000	0.0000	0.500000	0.000000	0.000000
25%	851.750000	0.0000	0.700000	0.000000	1.000000
50%	1226.000000	0.0000	1.500000	1.000000	3.000000
75%	1615.250000	1.0000	2.200000	1.000000	7.000000
max	1998.000000	1.0000	3.000000	1.000000	19.000000

	four_g	int_memory	m_dep	mobile_wt	n_cores
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	0.521500	32.046500	0.501750	140.249000	4.520500
std	0.499662	18.145715	0.288416	35.399655	2.287837
min	0.000000	2.000000	0.100000	80.000000	1.000000
25%	0.000000	16.000000	0.200000	109.000000	3.000000
50%	1.000000	32.000000	0.500000	141.000000	4.000000
75%	1.000000	48.000000	0.800000	170.000000	7.000000
max	1.000000	64.000000	1.000000	200.000000	8.000000

In Python's Pandas library, the combination of `isnull()` and `sum()` is a powerful tool for identifying and quantifying missing values within a DataFrame.

Breakdown:

1. `isnull()`:

- This method checks each element in a data for missing values (typically represented as NaN or None).
- It returns a data of the same size, with True where values are missing and False otherwise.

2. `sum()`:

- This method calculates the sum of values in a Series or DataFrame.
- When applied to a Boolean DataFrame (like the output of `isnull()`), it counts the True values, effectively tallying the missing values.

```
[7] df.isnull().sum()
...
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
```

The `dtypes` is used to see the type of the feature in the data frame.

```
[8] # Display data types of the columns
print(df.dtypes)
...
battery_power    int64
blue             int64
clock_speed      float64
dual_sim         int64
fc              int64
four_g          int64
int_memory       int64
m_dep           float64
mobile_wt        int64
n_cores          int64
pc              int64
px_height        int64
px_width         int64
ram             int64
sc_h            int64
sc_w            int64
talk_time        int64
three_g          int64
touch_screen     int64
wifi            int64
price_range      int64
dtype: object
```

The `nunique()` is used to see the unique values in every column:

```

# Get unique value counts for each column
unique_counts = df.nunique()
print(unique_counts)

```

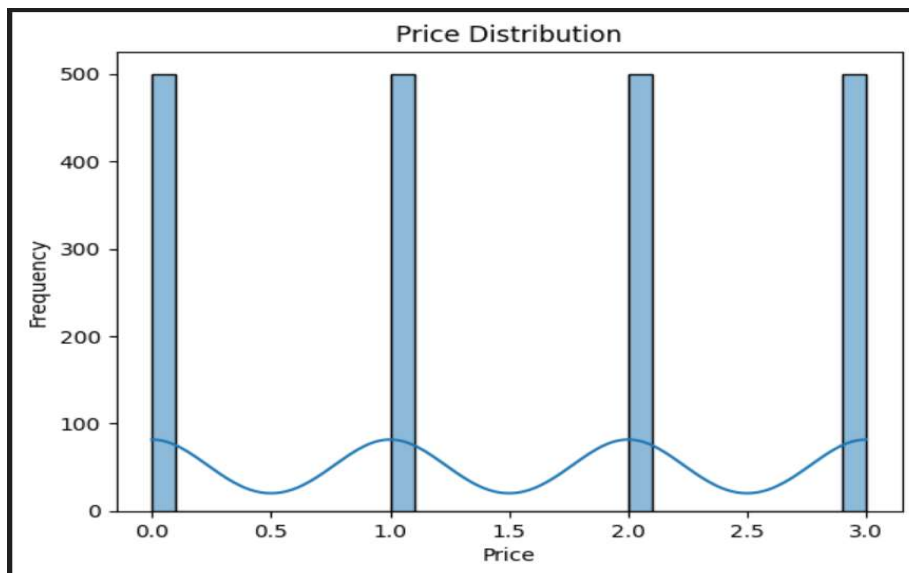
[9]

```

... battery_power    1094
    blue             2
    clock_speed      26
    dual_sim         2
    fc               20
    four_g           2
    int_memory       63
    m_dep            10
    mobile_wt        121
    n_cores          8
    pc               21
    px_height        1137
    px_width         1109
    ram              1562
    sc_h             15
    sc_w             19
    talk_time        19
    three_g          2
    touch_screen     2
    wifi             2
    price_range      4
    dtype: int64

```

Here the target column is price. We will see the distribution of the prices.

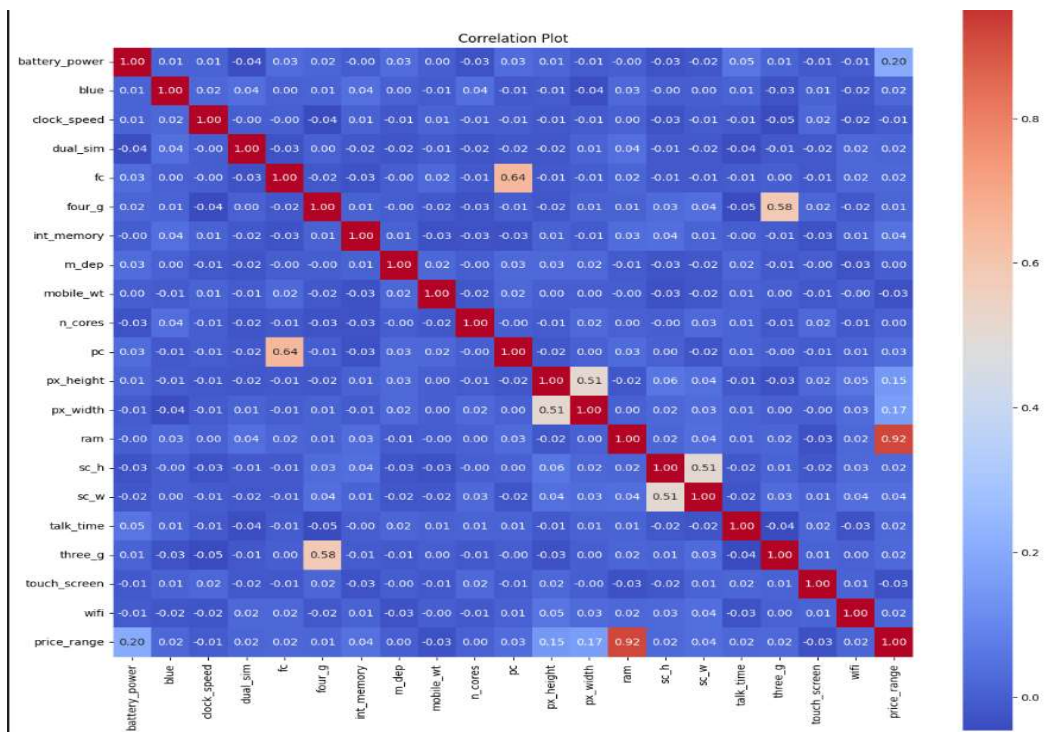


A **correlation matrix** is a table that shows the correlation coefficients between different variables in a dataset. These coefficients measure the strength and direction of the linear relationship between two variables.

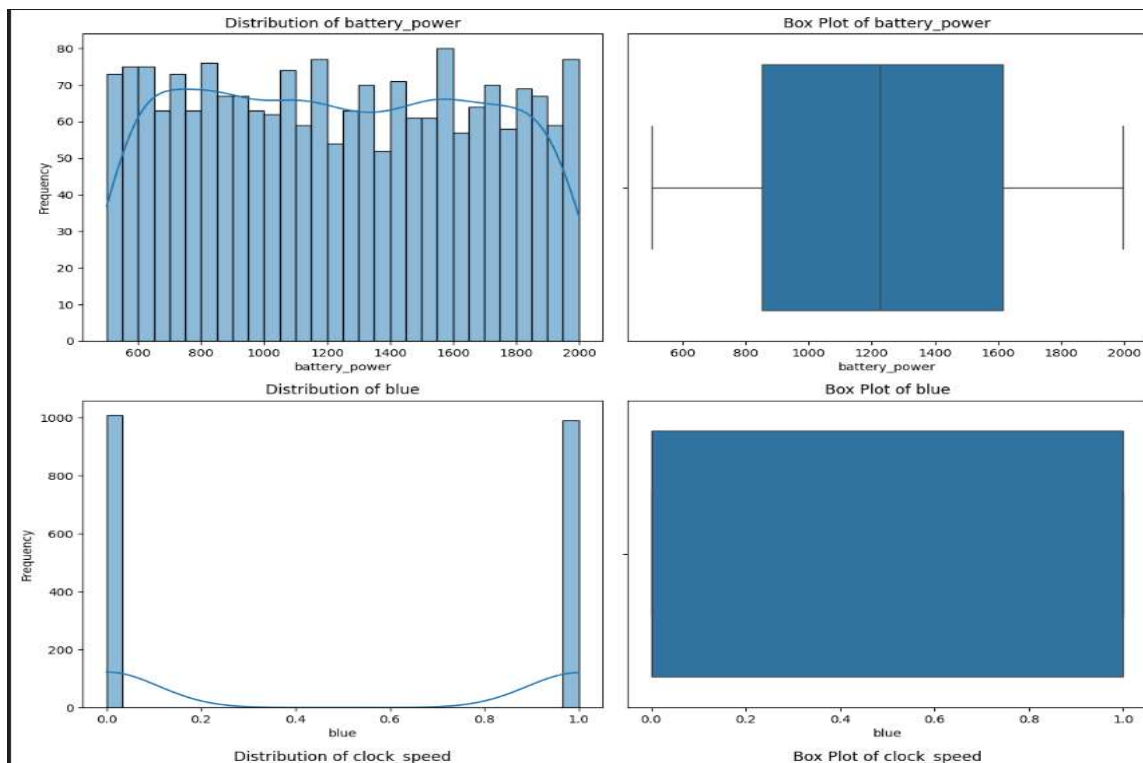
```

import matplotlib.pyplot as plt
import seaborn as sns
correlation_matrix=df.corr()
print(correlation_matrix)
plt.figure(figsize=(15, 15))
sns.heatmap(correlation_matrix,annot=True,cmap="coolwarm",fmt=".2f",square=True)
plt.title("Correlation Plot")
plt.show()

```



Then we will see the distribution of every feature using box plot and histogram.



This concludes the Experiment 1.

Experiment 2

Objective: Execute data preprocessing step on the above dataset: perform outlier and missing data analysis towards building a refined dataset

This experiment is the continuation of the above experiment i.e. experiment 1, all the steps that were done in the experiment 1 will be done again and then the objective for this experiment will be started.

So here according to the objective we will see the outlier and the missing data analysis and then build a refined dataset.

As it is already shown that there are no null or missing values in the dataset, we will directly move to check the presence of outliers in the data.

Here we have use two methods for checking the outliers. They are:

Z-Score and IQR (Interquartile Range).

Z-Score: A Z-score, also known as a standard score, measures how many standard deviations a data point is from the mean of a data set. It standardizes data, allowing comparison of values from different distributions.

$$\text{Z-Score} = (X - \mu) / \sigma$$

```
from scipy import stats

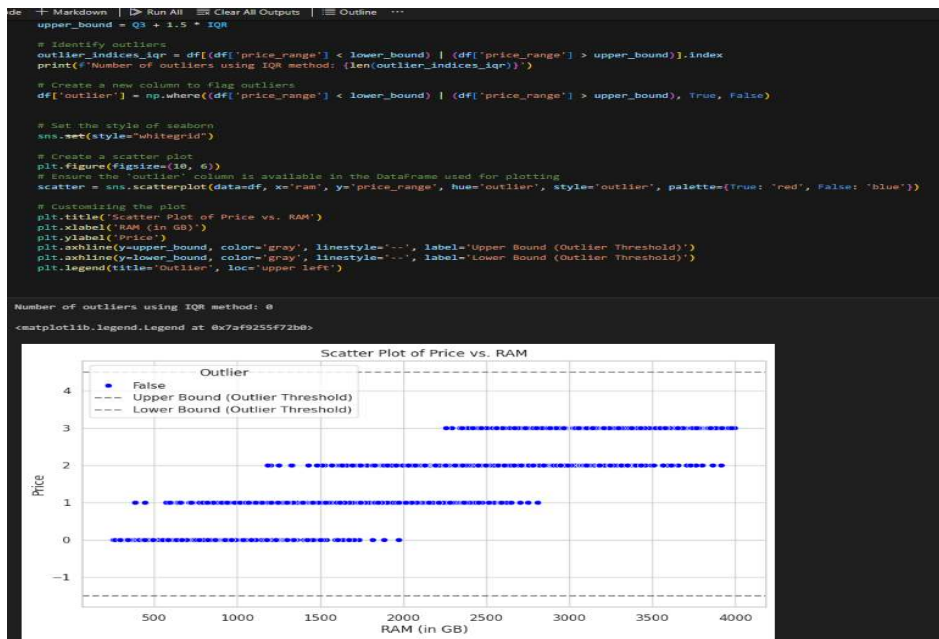
# Calculate Z-scores
z_scores = stats.zscore(df.select_dtypes(include=['float64', 'int64']))
abs_z_scores = abs(z_scores)
threshold = 3
outliers = (abs_z_scores > threshold).any(axis=1)

# Get the indices of outliers
outlier_indices = df[outliers].index
print(f'Number of outliers: {len(outlier_indices)}')
```

Number of outliers: 12

IQR: The IQR is a measure of variability, especially useful when data is skewed or contains outliers. It represents the range of the middle 50% of the data.

$$\text{IQR} = Q3 - Q1$$



The choice between Z-scores and IQR depends on the specific characteristics of your data and the goals of your analysis. If you have a normally distributed dataset and need to compare values across different distributions, Z-scores are a powerful tool. If your data is skewed or contains significant outliers, IQR may be a more appropriate choice.

Now after finding outliers, we will remove it and we will prepare a refined dataset.

```

# Refine the dataset by removing outliers
df_refined = df[(df['price_range'] >= lower_bound) & (df['price_range'] <= upper_bound)]

# Save the refined DataFrame to a new CSV file
df_refined.to_csv('refined_mobile_prices.csv', index=False)

```

With these we have completed the objective of Experiment 2.

Experiment 3

Objective: Build machine learning model/s to predict the actual price of the new mobile based on other given features like RAM, Internal Memory etc

This experiment is also a continuation of the experiment 1 and 2 where we have already done the feature analysis and removing outliers and build a refined dataset on which experiment 3 will be performed. Here we have performed total five classifiers in this experiment and we have checked their accuracy score and prediction score for each of them.

We have split the dataset in training and testing dataset.

The machine learning model we have used here:

1. KNN Classifier:

```
# Initialize classifiers
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC()
}

# Loop through the classifiers, fit them, make predictions, and calculate precision
for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate precision
    precision = precision_score(y_test, y_pred, average='weighted') # Use 'weighted' for multi-class
    print(f'{name} Precision: {precision:.2f}')

48]

** KNN Precision: 0.94
    Decision Tree Precision: 0.83
    Random Forest Precision: 0.89
    Naive Bayes Precision: 0.81
    SVM Precision: 0.97
```

2. Decision Tree:

```
# Initialize classifiers
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC()
}

# Loop through the classifiers, fit them, make predictions, and calculate precision
for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate precision
    precision = precision_score(y_test, y_pred, average='weighted') # Use 'weighted' for multi-class
    print(f'{name} Precision: {precision:.2f}')

48]

** KNN Precision: 0.94
    Decision Tree Precision: 0.83
    Random Forest Precision: 0.89
    Naive Bayes Precision: 0.81
    SVM Precision: 0.97
```

3. Random Forest Classifier:

```
# Initialize classifiers
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC()
}

# Loop through the classifiers, fit them, make predictions, and calculate precision
for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate precision
    precision = precision_score(y_test, y_pred, average='weighted') # Use 'weighted' for multi-class
    print(f'{name} Precision: {precision:.2f}')

48]

** KNN Precision: 0.94
    Decision Tree Precision: 0.83
    Random Forest Precision: 0.89
    Naive Bayes Precision: 0.81
    SVM Precision: 0.97
```

4. Naive Bayes Classifier:

```

# Initialize classifiers
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC()
}

# Loop through the classifiers, fit them, make predictions, and calculate precision
for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate precision
    precision = precision_score(y_test, y_pred, average='weighted') # Use 'weighted' for multi-class
    print(f'{name} Precision: {precision:.2f}')

```

48]

```

.. KNN Precision: 0.94
   Decision Tree Precision: 0.83
   Random Forest Precision: 0.89
   Naive Bayes Precision: 0.81
   SVM Precision: 0.97

```

5. SVM Classifier:

```

# Initialize classifiers
classifiers = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC()
}

# Loop through the classifiers, fit them, make predictions, and calculate precision
for name, clf in classifiers.items():
    # Train the classifier
    clf.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = clf.predict(X_test)

    # Calculate precision
    precision = precision_score(y_test, y_pred, average='weighted') # Use 'weighted' for multi-class
    print(f'{name} Precision: {precision:.2f}')

```

48]

```

.. KNN Precision: 0.94
   Decision Tree Precision: 0.83
   Random Forest Precision: 0.89
   Naive Bayes Precision: 0.81
   SVM Precision: 0.97

```

Now getting all the classifier together we came to conclusion is that SVM Classifier is giving better accuracy than others.

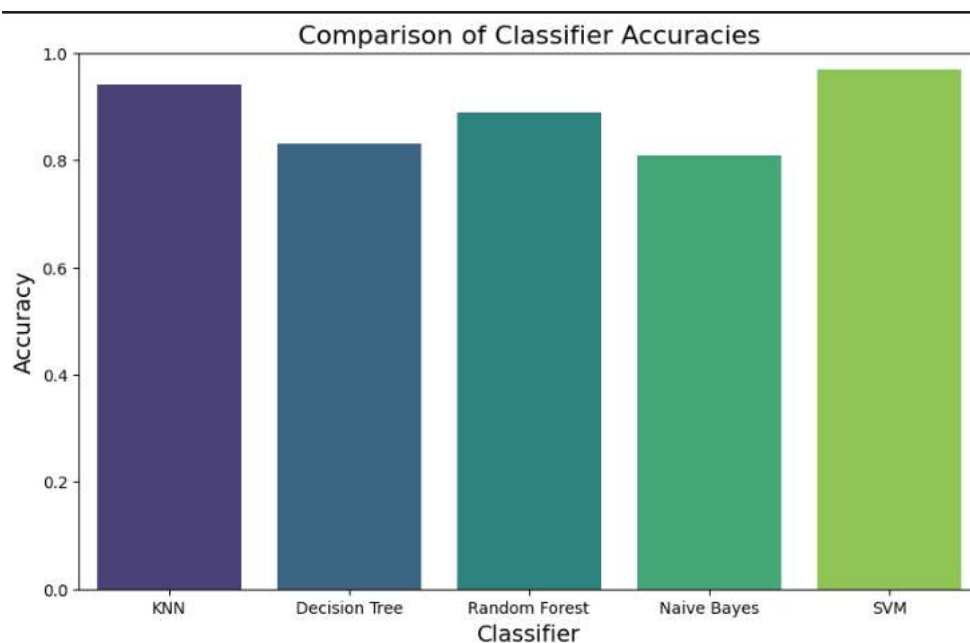
With these we conclude the Experiment 3.

Experiment 4

Objective: Calculate the prediction accuracy of the models used in Experiment 3 and do comparative analysis among them to identify the best technique.

This is also the continuation of the experiment 1, 2, 3.

We will do competitive analysis of the of the experiment 3.



According to this visualization we displayed here, this shows that the SVM is giving better accuracy than other while comparing them to each other

Then we will create the confusion matrix of these:

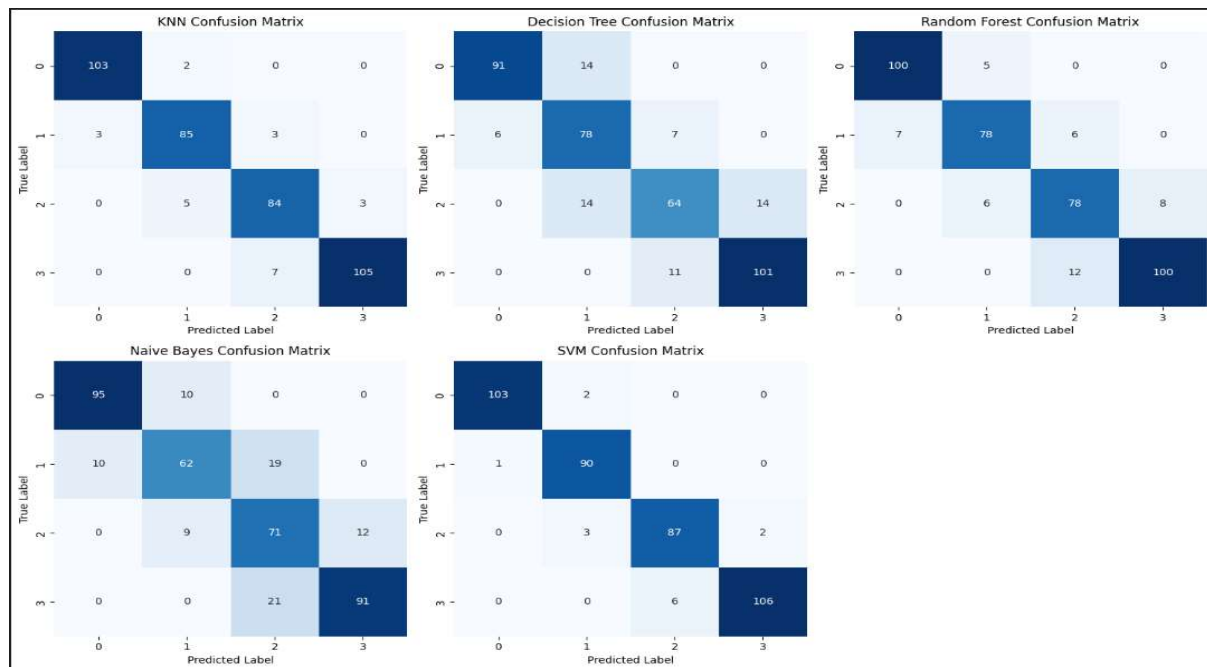
What is a Confusion Matrix?

A confusion matrix is a table that summarizes the performance of a classification model on a set of test data. It's a visual way to understand how well your model is distinguishing between different classes.

Key Components:

1. **True Positive (TP):** The model correctly predicted a positive class.
2. **True Negative (TN):** The model correctly predicted a negative class.

3. **False Positive (FP):** The model incorrectly predicted a positive class (Type I error).
4. **False Negative (FN):** The model incorrectly predicted a negative class (Type II error).



Conclusion:

- **Class Imbalance:** It appears that the dataset is imbalanced, with a significantly higher number of instances in class 3 compared to the other classes. This can affect the performance of some models.
- **Model Performance:** The models seem to perform fairly well overall, with most of the predictions concentrated along the diagonal of the confusion matrices. However, there are some misclassifications, especially for classes 1 and 2.

Specific Model Analysis:

- **KNN:** KNN seems to struggle with class 1, misclassifying several instances as class 2. It also has some difficulty distinguishing between classes 2 and 3.
- **Decision Tree:** The decision tree has a similar issue with class 1 as KNN. It also misclassifies some instances of class 2 as class 3.
- **Random Forest:** Random forest appears to be the best-performing model among these, with fewer misclassifications overall. However, it still has some difficulties with class 1.
- **Naive Bayes:** Naive Bayes struggles with class 1, misclassifying many instances as class 2. It also has some issues with class 2.
- **SVM:** SVM performs reasonably well, but it still has some misclassifications, particularly with class 1.

Experiment 5

Objective: Understanding "Second Hand Car Prediction Price" dataset by doing feature analysis. Data is available at:

<https://www.kaggle.com/datasets/sujithmandala/second-hand-car-priceprediction>

First, we must extract the dataset using Pandas library.

The dataset we downloaded is shown:

data													
	Car_ID	Brand	Model	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price
0	1	Toyota	Corolla	2018	50000	Petrol	Manual	First	15	1498	108	5	800000
1	2	Honda	Civic	2019	40000	Petrol	Automatic	Second	17	1597	140	5	1000000
2	3	Ford	Mustang	2017	20000	Petrol	Automatic	First	10	4951	395	4	2500000
3	4	Maruti	Swift	2020	30000	Diesel	Manual	Third	23	1248	74	5	600000
4	5	Hyundai	Sonata	2016	60000	Diesel	Automatic	Second	18	1999	194	5	850000
...
95	96	Mercedes	C-Class	2019	22000	Diesel	Automatic	First	16	1950	191	5	2900000
96	97	Toyota	Innova Crysta	2017	38000	Diesel	Manual	Second	13	2755	171	7	1400000
97	98	Ford	EcoSport	2018	26000	Petrol	Manual	Third	18	1497	121	5	750000
98	99	Hyundai	Verna	2019	24000	Petrol	Automatic	Second	17	1497	113	5	850000
99	100	Tata	Altroz	2020	18000	Petrol	Manual	First	20	1199	85	5	600000

100 rows × 13 columns

In next step we will do feature engineering:

Feature Engineering: Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning. ¹ It's a crucial step in machine learning pipelines, as it directly impacts the performance of your models.

```
# Feature engineering
from datetime import date
data['Car_age'] = date.today().year - data['Year']
```

So, the new dataset that is formed after adding the column formed through previous process:


```
# Statistical Summary
data.describe(include='all') #for all columns
```

	Car_ID	Brand	Model	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	Car_age
count	100.000000	100	100	100.000000	100.000000	100	100	100	100.000000	100.000000	100.000000	100.000000	1.000000e+02	100.000000
unique	NaN	11	58	NaN	NaN	2	2	3	NaN	NaN	NaN	NaN	NaN	NaN
top	NaN	Ford	Mustang	NaN	NaN	Petrol	Automatic	First	NaN	NaN	NaN	NaN	NaN	NaN
freq	NaN	11	3	NaN	NaN	52	57	44	NaN	NaN	NaN	NaN	NaN	NaN
mean	50.500000	NaN	NaN	2018.390000	28150.000000	NaN	NaN	NaN	17.210000	1855.230000	158.130000	5.230000	1.574000e+06	5.610000
std	29.011492	NaN	NaN	1.171116	9121.375716	NaN	NaN	NaN	3.309902	631.311475	76.968137	0.750151	1.000265e+06	1.171116
min	1.000000	NaN	NaN	2016.000000	10000.000000	NaN	NaN	NaN	10.000000	999.000000	68.000000	4.000000	4.500000e+05	3.000000
25%	25.750000	NaN	NaN	2017.750000	22000.000000	NaN	NaN	NaN	15.000000	1462.000000	103.000000	5.000000	7.000000e+05	5.000000
50%	50.500000	NaN	NaN	2018.000000	27000.000000	NaN	NaN	NaN	17.000000	1774.000000	148.000000	5.000000	1.300000e+06	6.000000
75%	75.250000	NaN	NaN	2019.000000	32000.000000	NaN	NaN	NaN	19.000000	2143.000000	187.000000	5.000000	2.500000e+06	6.250000
max	100.000000	NaN	NaN	2021.000000	60000.000000	NaN	NaN	NaN	25.000000	4951.000000	396.000000	7.000000	4.000000e+06	8.000000

After this we distribute the categorical column and numerical column:

```
# seperate numerical and categorical variables
cat_cols=data.select_dtypes(include=['object']).columns

num_cols = data.select_dtypes(include=np.number).columns.tolist()

print("Categorical columns:",cat_cols)

... Categorical columns: Index(['Brand', 'Model', 'Fuel_Type', 'Transmission', 'Owner_Type'], dtype='object')

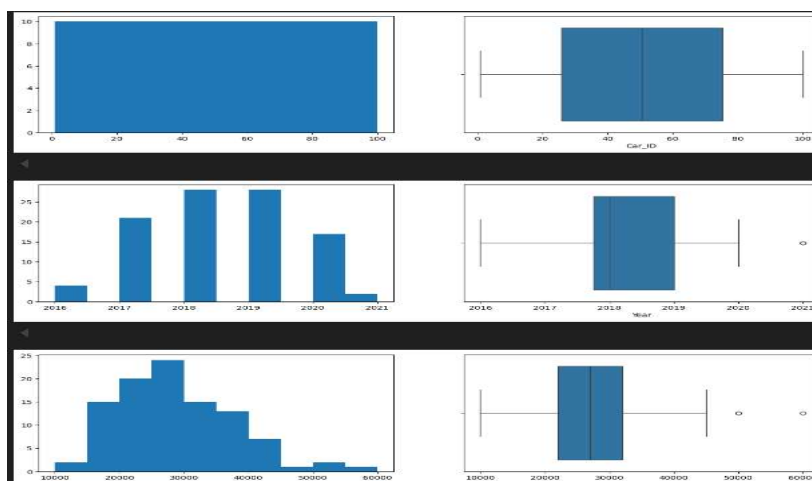
print("Numerical columns:",num_cols)

... Numerical columns: ['Car_ID', 'Year', 'Kilometers_Driven', 'Mileage', 'Engine', 'Power', 'Seats', 'Price', 'Car_age']
```

Here we are doing univariate analysis:

Univariate Analysis: Univariate analysis is a statistical method used to describe a single variable. In the context of machine learning, it's a fundamental technique for understanding the distribution, central tendency, and dispersion of a single feature within a dataset.

Then we have plotted all the features to see their distribution.



Then we will be performing the bivariate analysis.

Bivariate Analysis: Bivariate analysis is a statistical method used to understand the relationship between two variables. It helps to identify patterns, trends, and correlations between these two variables.

So, we have plotted all the features and here is the output.



Here we conclude experiment 5.

EXPERIMENT – 6

Objective: Perform data preprocessing step on the above dataset: perform outlier and missing data analysis towards building a refined dataset.

This experiment is the continuation of the above experiment i.e. experiment 5, all the steps that were done in the experiment 5 will be done again and then the objective for this experiment will be started.

So here according to the objective we will see the outlier and the missing data analysis and then build a refined dataset.

As it is already shown that there are no null or missing values in the dataset, we will directly move to check the presence of outliers in the data.

So first we are applying Label Encoder.

Label Encoder: Label encoding is a technique used to convert categorical data into numerical format. This is crucial in machine learning as many algorithms can only process numerical data.

	Car_ID	Brand	Model	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price
0	1	9	15	2018	50000	1	1	0	15	1498	108	5	800000
1	2	3	14	2019	40000	1	0	1	17	1597	140	5	1000000
2	3	2	30	2017	20000	1	0	0	10	4951	395	4	2500000
3	4	6	42	2020	30000	0	1	2	23	1248	74	5	600000
4	5	4	41	2016	60000	0	0	1	18	1999	194	5	850000
...
95	96	7	11	2019	22000	0	0	0	16	1950	191	5	2900000
96	97	9	29	2017	38000	0	1	1	13	2755	171	7	1400000
97	98	2	18	2018	26000	1	1	2	18	1497	121	5	750000
98	99	4	50	2019	24000	1	0	1	17	1497	113	5	850000
99	100	8	7	2020	18000	1	1	0	20	1199	85	5	600000

100 rows x 13 columns

Here we are also performing correlation matrix.

Correlation Matrix: A correlation matrix is a table that shows the correlation coefficients between a set of variables. It's a powerful tool for understanding the relationships between variables in a dataset.

	Car_ID	Brand	Model	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats
Car_ID	1.000000	0.015205	0.053171	0.059904	-0.227442	0.088063	0.087118	0.049568	0.026140	-0.027965	0.027637	-0.021582
Brand	0.015205	1.000000	0.194702	0.110875	0.082947	0.063037	0.187123	-0.079183	0.086441	-0.166709	-0.335425	0.114417
Model	0.053171	0.194702	1.000000	0.090754	0.011237	-0.059048	0.263721	0.088050	0.130721	-0.074587	-0.153203	-0.063274
Year	0.059904	0.110875	0.090754	1.000000	-0.741176	0.252843	0.211993	-0.036276	0.213177	-0.355122	-0.249446	-0.252598
Kilometers_Driven	-0.227442	0.082947	0.011237	-0.741176	1.000000	-0.323762	0.030157	-0.000560	-0.104437	0.112340	-0.026732	0.396443
Fuel_Type	0.088063	0.063037	-0.059048	0.252843	-0.323762	1.000000	0.066306	0.323325	-0.060292	-0.258043	-0.008562	-0.347549
Transmission	0.087118	0.187123	0.263721	0.211993	0.030157	0.066306	1.000000	0.097708	0.331015	-0.421374	-0.574349	0.111225
Owner_Type	0.049568	-0.079183	0.088050	-0.036276	-0.000560	0.323325	0.097708	1.000000	0.072823	-0.248205	-0.237857	-0.075318
Mileage	0.026140	0.086441	0.130721	0.213177	-0.104437	-0.060292	0.331015	0.072823	1.000000	-0.680949	-0.648894	-0.194581
Engine	-0.027965	-0.166709	-0.074587	-0.355122	0.112340	-0.258043	-0.421374	-0.248205	-0.680949	1.000000	0.805709	0.179179
Power	0.027637	-0.335425	-0.153203	-0.249446	-0.026732	-0.008562	-0.574349	-0.237857	-0.648894	0.805709	1.000000	-0.102867
Seats	-0.021582	0.114417	-0.063274	-0.252598	0.396443	-0.347549	0.111225	-0.075318	-0.194581	0.179179	-0.102867	1.000000
Price	0.037105	-0.354543	-0.292434	-0.232687	-0.051104	-0.180964	-0.676484	-0.301488	-0.595252	0.714465	0.856620	-0.000027

Here we have use two methods for checking the outliers. They are:

Z-Score and IQR (Interquartile Range).

Z-Score: A Z-score, also known as a standard score, measures how many standard deviations a data point is from the mean of a data set. It standardizes data, allowing comparison of values from different distributions.

$$\text{Z-Score} = (X - \mu) / \sigma$$

```
from scipy import stats

# Calculate Z-scores
z_scores = stats.zscore(df.select_dtypes(include=['float64', 'int64']))
abs_z_scores = abs(z_scores)
threshold = 3
outliers = (abs_z_scores > threshold).any(axis=1)

# Get the indices of outliers
outlier_indices = df[outliers].index
print(f'Number of outliers: {len(outlier_indices)}')
```

Number of outliers: 12

QR: The IQR is a measure of variability, especially useful when data is skewed or contains outliers. It represents the range of the middle 50% of the data.

$$\text{IQR} = Q3 - Q1$$

```
# Identify outliers
upper_bound = Q3 + 1.5 * IQR
outlier_indices_iqr = df[(df['price_range'] < lower_bound) | (df['price_range'] > upper_bound)].index
print(f'Number of outliers using IQR method: {len(outlier_indices_iqr)}')

# Create a new column to flag outliers
df['outlier'] = np.where((df['price_range'] < lower_bound) | (df['price_range'] > upper_bound), True, False)

# Set the style of seaborn
sns.set(style='whitegrid')

# Create a scatter plot
plt.figure(figsize=(10, 6))
# Group the outlier column is available in the DataFrame used for plotting
scatter = sns.scatterplot(data=df, x='price', y='price_range', hue='outlier', style='outlier', palette={True: 'red', False: 'blue'})

# Customizing the plot
plt.title('Scatter Plot of Price vs. Range')
plt.xlabel('Price (in $B)')
plt.ylabel('Price')
plt.axhline(y=upper_bound, color='gray', linestyle='--', label='Upper Bound (Outlier Threshold)')
plt.axhline(y=lower_bound, color='gray', linestyle='--', label='Lower Bound (Outlier Threshold)')
plt.legend(title='Outlier', loc='upper left')
```

Number of outliers using IQR method: 0

matplotlib.legend.legend at #a7a92554722ab

The choice between Z-scores and IQR depends on the specific characteristics of your data and the goals of your analysis. If you have a normally distributed dataset and need to compare values across different distributions, Z-scores are a powerful tool. If your data is skewed or contains significant outliers, IQR may be a more appropriate choice.

Now after finding outliers, we will remove it and we will prepare a refined dataset.

```
# Refine the dataset by removing outliers
df_refined = df[(df['price_range'] >= lower_bound) & (df['price_range'] <= upper_bound)]

# Save the refined DataFrame to a new CSV file
df_refined.to_csv('refined_mobile_prices.csv', index=False)
```

With these we have completed the objective of Experiment 6.

EXPERIMENT – 7

Objective: Perform Feature Engineering towards building new feature which is more impactful. Build machine learning model/s to predict the price of the car based on other given features like Brand, Model, Year, Fuel Type etc

This experiment is the continuation of the above experiment i.e. experiment 5 and 6, all the steps that were done in the experiment 5 and 6 will be done again and then the objective for this experiment will be started.

So here according to the objective we will see how to perform feature engineering and have to build a machine learning model

Feature Engineering: It is the process of selecting and transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy¹ on unseen data. It's a crucial step in the machine learning pipeline, often requiring domain knowledge and creativity.

```
# Feature Engineering:

# 1. Interaction terms:
df['Brand_Model_Interaction'] = df['Brand'] * df['Model']
df['Age_Kilometers_Interaction'] = df['Car_age'] * df['Kilometers_Driven_log']
```

We are using three machine learning model here:

Regression: Regression is a statistical method used to model the relationship between a dependent variable (target variable) and one or more independent variables (predictor variables). It's a fundamental technique in machine learning and statistics, used to make predictions and understand the impact of independent variables on the Here we are seeing the performance of the models with all features and the performance of the model with selected important features to see the result of the metrics we got.

- Here **Linear Regression, Random Forest Regression, Decision Tree Regression** have used.

```
Linear Regression:
Mean Squared Error: 187353440551.8432
R-squared: 0.7709195567012983

Decision Tree Regressor:
Mean Squared Error: 198375000000.0
R-squared: 0.7574432964480039

Random Forest Regressor:
Mean Squared Error: 135053200000.0
R-squared: 0.8348680075808522
```

EXPERIMENT – 8

Objective: Calculate the prediction accuracy of the models used in Experiment 7 and do comparative analysis among them to identify the best technique.

This experiment is the continuation of the above experiment i.e. experiment 5 and 6 and 7, all the steps that were done in the experiment 5 and 6 and 7 will be done again and then the objective for this experiment will be started.

So here according to the objective we will see how to calculate prediction and accuracy and have to perform competitive analysis among them.

We will do prediction and accuracy determination analysis among all features and important features.

```
target_correlation = correlation_matrix['Price'].abs().sort_values(ascending=False)
selected_features = target_correlation[target_correlation>0.5].index.tolist()
print(f"Selected features high;y with price: {selected_features}")

.. Selected features high;y with price: ['Price', 'Price_log', 'Power', 'Engine', 'Transmission', 'Mileage']
```

Here is the selected feature is mentioned above. We will do competitive analysis among all the features and the different important features.

```
Random Forest Regressor (All Features):
Mean Squared Error: 135053200000.0
R-squared: 0.8348680075808522

Random Forest Regressor (Important Features):
Mean Squared Error: 3143987500.0
R-squared: 0.9961557895702146

Comparison:
All Features - MSE: 135053200000.0, R-squared: 0.8348680075808522
Important Features - MSE: 3143987500.0, R-squared: 0.9961557895702146
```

Conclusion:

- **Feature Importance:** Using only the important features significantly improves the model's performance. This is evident from the drastic reduction in MSE and the substantial increase in R-squared.
- **Model Complexity:** The model with all features is likely more complex and prone to overfitting. This is supported by the higher MSE and lower R-squared.
- **Feature Selection:** Identifying and using only the most relevant features is crucial for building accurate and efficient models.

EXPERIMENT – 9

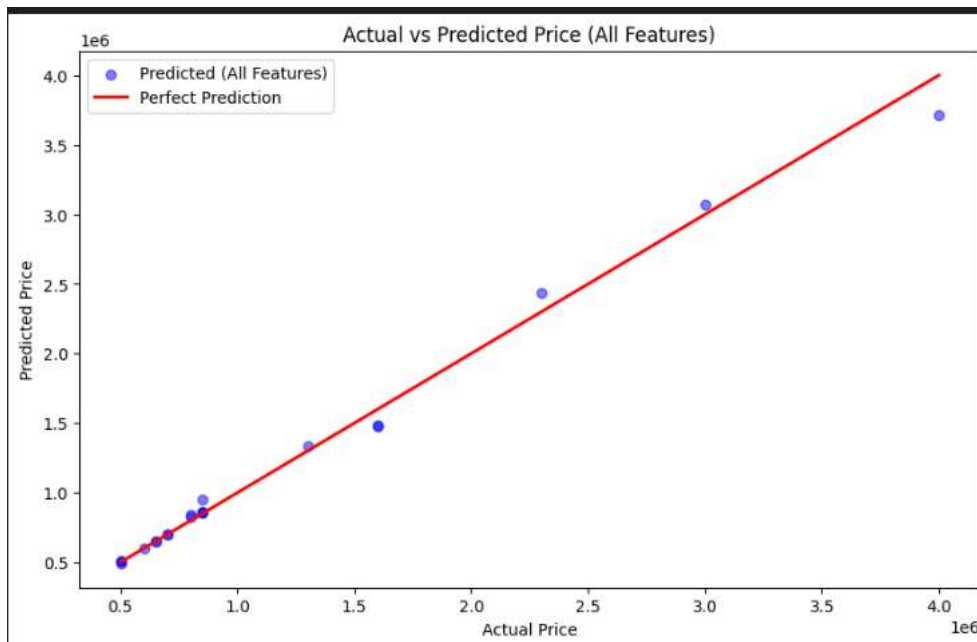
Objective: Plot the features (actual price and predicted price) in scatter plot to understand the variation

This experiment is the continuation of the above experiment i.e. experiment 5 and 6 and 7, all the steps that were done in the experiment 5 and 6 and 7 will be done again and then the objective for this experiment will be started.

So here according to the objective we will plot the features (actual price and predicted price) in scatter plot to understand the variation.

Scatter Plot: Scatter plots are a powerful visualization tool used to explore the relationship between two numerical variables. They plot individual data points on a two-dimensional graph, with one variable on the x-axis and the other on the y-axis.

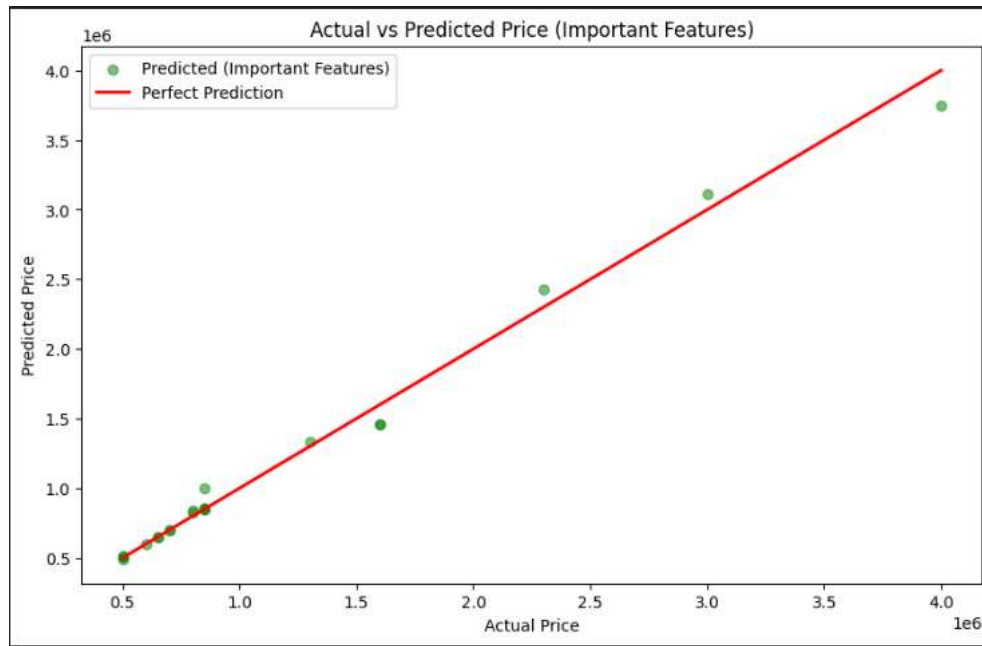
This scatter plot is for all features to see the difference between the actual price and predicted price.



The conclusion we get from this:

- The red line represents the perfect prediction line. The closer the blue dots are to this line, the better the model's performance.
- In this case, the model's predictions are quite close to the perfect line, suggesting good accuracy.

This next scatter plot is to see the relation between the actual and predicted price using only the important features:



Based on the scatter plot, we can draw the following conclusions:

1. **The model has a strong predictive power.** It accurately predicts the house prices.
2. **The model might be slightly underestimating higher-priced houses.** This is indicated by the slight underestimation of the actual price for the higher-priced houses.
3. **Further analysis is needed to understand the outliers.** These outliers could be due to factors like unique features, location, or other factors not captured by the model.

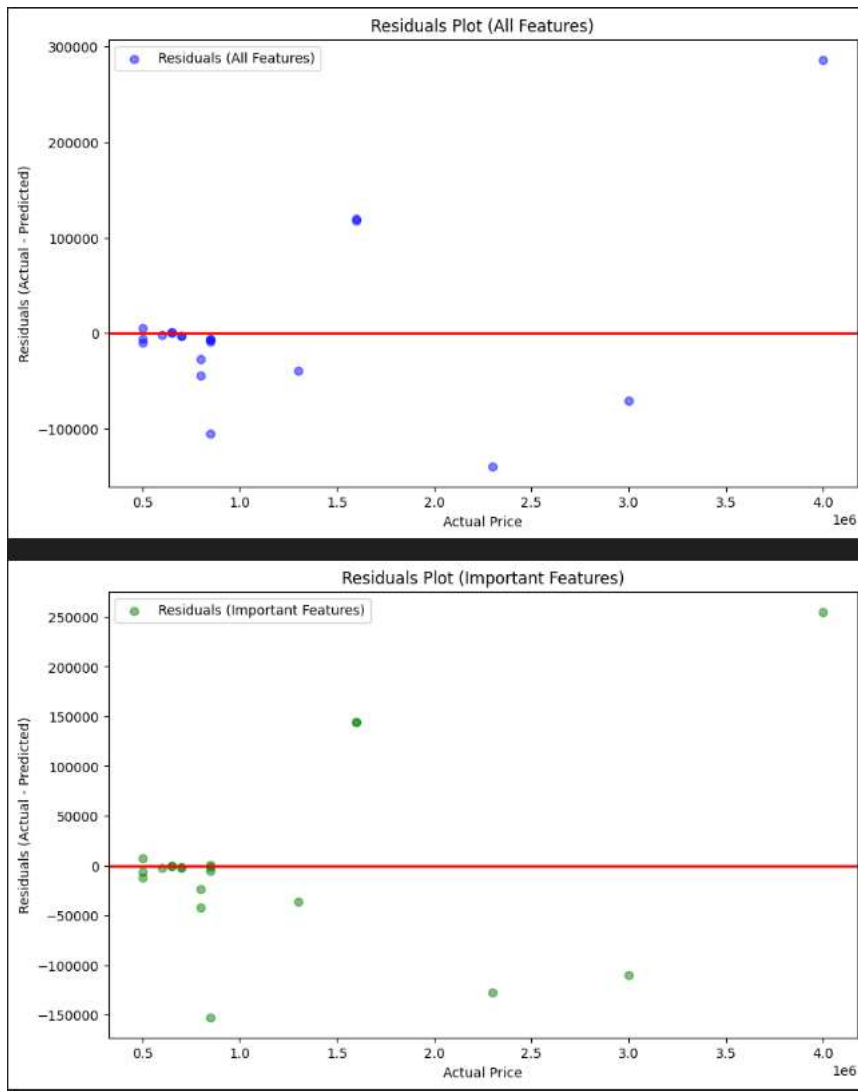
Again, we have residual plot.

Residual plots are a valuable tool in assessing the quality of a regression model. They help visualize the difference between the predicted values and the actual values, providing insights into the model's assumptions and potential issues.

What is a Residual?

A residual is the difference between the observed value of a dependent variable (y) and the predicted value (\hat{y}) from a statistical model.

The first plot for all features and second for important features.



Residual Plot (All Features)

- **Randomness:** The residuals appear to be randomly scattered around the horizontal line at zero, which is a good sign. This suggests that the model is capturing the underlying patterns in the data.
- **Constant Variance:** There seems to be a slight increase in the spread of residuals as the actual price increases. This could indicate potential heteroscedasticity, where the variability of the residuals changes with the level of the independent variable.

Residual Plot (Important Features)

- **Randomness:** The residuals are more tightly clustered around the horizontal line compared to the "All Features" plot. This indicates a better fit and less variability in the predictions.

- **Constant Variance:** The spread of residuals appears more consistent across the range of actual prices, suggesting less heteroscedasticity.

Conclusions:

1. **Feature Importance:** The model with important features performs better than the model with all features. This is evident from the tighter clustering of residuals and the more consistent variance in the "Important Features" plot.
2. **Model Fit:** Both models appear to be reasonably good fits for the data, as the residuals are randomly scattered around zero.
3. **Heteroscedasticity:** The "All Features" model might benefit from techniques like log transformation or weighted least squares to address the potential heteroscedasticity.
4. **Outliers:** There are a few outliers in both plots, which could be due to data quality issues or unusual data points. These outliers might be influencing the model's performance and could be investigated further.