DATA SCIENCE

MACHINE LEARNING PROJECT

CAR PRICE PREDICTION

NAME: SAWAIRA IQBAL

ID: sawaira.iqbal.ds@gmail.com

Project Presentation

DATA SCIENCE COMPLETE PROJECT

CAR PRICE PREDICTION

NEED YOUR ATTENTION

Process Flow of Project

1. Understanding Problem Statement
2. Getting System Ready
3. Data Collection
4. Understanding the Data-Data Eyeballing & Data Description
5. Data Cleaning & Preprocessing I
6. Exploratory Data Analysis (EDA)
    - Univariate Analysis
    - Bivariate Analysis
    - Multivariate Analysis
7. Data Cleaning & Preprocessing II
8. Insights from Data Visualization
9. Feature Engineerig
10. Model Buidling & Evaluation
11. Selection of Best Model & Hyperparameter Tuninng
12. Generating Pickle file

PROBLEM IDENTIFICATION AND CLARIFICATION

The Problem:

- Many things affect prices, like car model, mileage, condition, and market trends.
- Sellers don’t have enough tools or data to set the right price.
- Buyers often feel prices are unfair, making it hard to complete sales.

Key Question?

"What factors decide the price of a used car, and how can knowing these factors help sellers set better prices?"

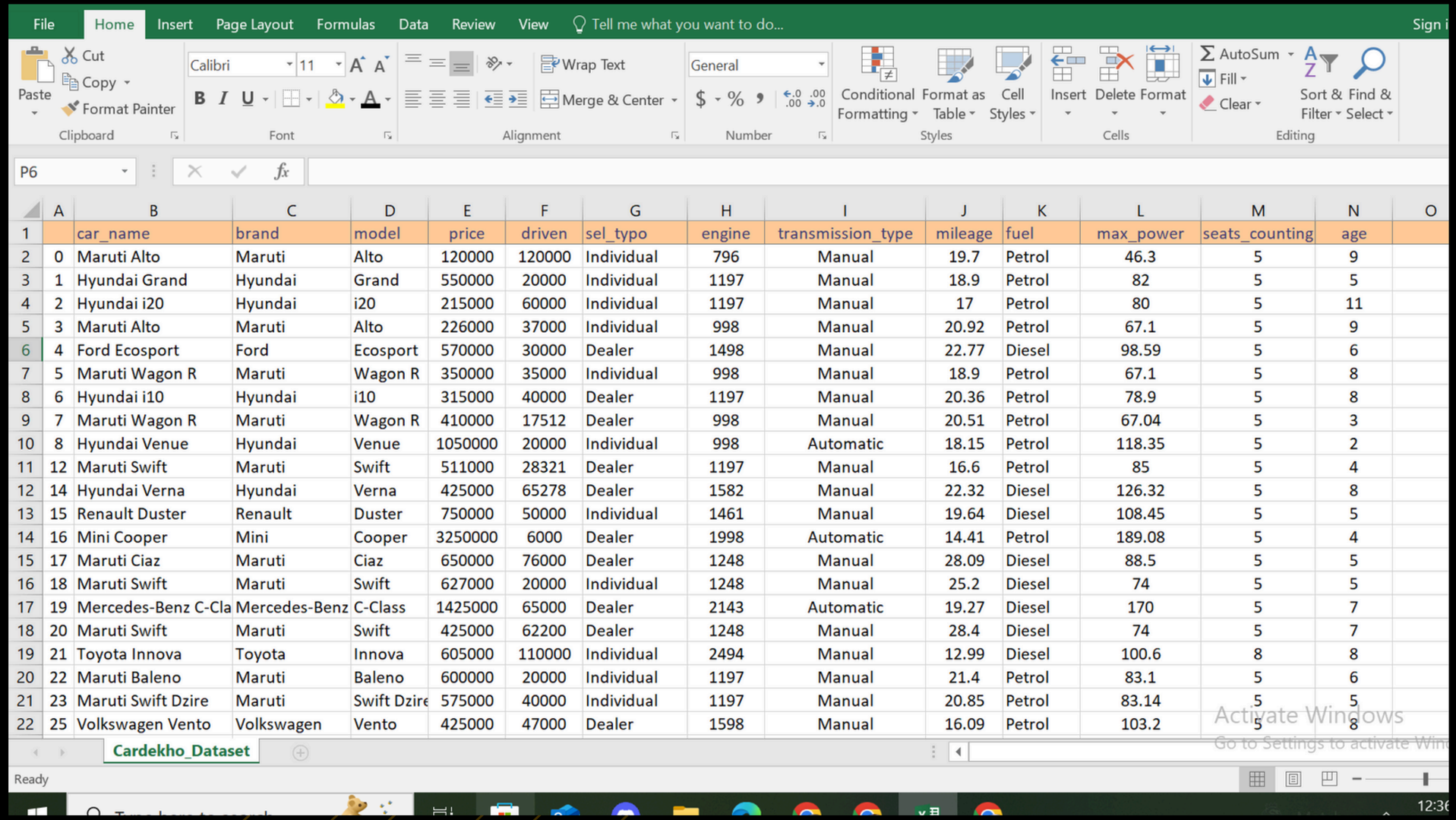
Goal:

To find the main factors and create a simple system to help sellers price cars fairly and accurately.

TOOL USING

- Google Colab is a cloud-based Python development environment that enables you to write and execute Python code in your browser, providing free access to GPUs and TPUs for intensive computations.
- It facilitates seamless integration with Google Drive for data access and collaboration, making it highly suitable for machine learning projects that require robust computational resources without any local setup.

DATASET



	A	car_name	brand	model	price	driven	sel_typo	engine	transmission_type	mileage	fuel	max_power	seats_counting	age
2	0	Maruti Alto	Maruti	Alto	120000	120000	Individual	796	Manual	19.7	Petrol	46.3	5	9
3	1	Hyundai Grand	Hyundai	Grand	550000	20000	Individual	1197	Manual	18.9	Petrol	82	5	5
4	2	Hyundai i20	Hyundai	i20	215000	60000	Individual	1197	Manual	17	Petrol	80	5	11
5	3	Maruti Alto	Maruti	Alto	226000	37000	Individual	998	Manual	20.92	Petrol	67.1	5	9
6	4	Ford Ecosport	Ford	Ecosport	570000	30000	Dealer	1498	Manual	22.77	Diesel	98.59	5	6
7	5	Maruti Wagon R	Maruti	Wagon R	350000	35000	Individual	998	Manual	18.9	Petrol	67.1	5	8
8	6	Hyundai i10	Hyundai	i10	315000	40000	Dealer	1197	Manual	20.36	Petrol	78.9	5	8
9	7	Maruti Wagon R	Maruti	Wagon R	410000	17512	Dealer	998	Manual	20.51	Petrol	67.04	5	3
10	8	Hyundai Venue	Hyundai	Venue	1050000	20000	Individual	998	Automatic	18.15	Petrol	118.35	5	2
11	12	Maruti Swift	Maruti	Swift	511000	28321	Dealer	1197	Manual	16.6	Petrol	85	5	4
12	14	Hyundai Verna	Hyundai	Verna	425000	65278	Dealer	1582	Manual	22.32	Diesel	126.32	5	8
13	15	Renault Duster	Renault	Duster	750000	50000	Individual	1461	Manual	19.64	Diesel	108.45	5	5
14	16	Mini Cooper	Mini	Cooper	3250000	6000	Dealer	1998	Automatic	14.41	Petrol	189.08	5	4
15	17	Maruti Ciaz	Maruti	Ciaz	650000	76000	Dealer	1248	Manual	28.09	Diesel	88.5	5	5
16	18	Maruti Swift	Maruti	Swift	627000	20000	Individual	1248	Manual	25.2	Diesel	74	5	5
17	19	Mercedes-Benz C-Cla	Mercedes-Benz	C-Class	1425000	65000	Dealer	2143	Automatic	19.27	Diesel	170	5	7
18	20	Maruti Swift	Maruti	Swift	425000	62200	Dealer	1248	Manual	28.4	Diesel	74	5	7
19	21	Toyota Innova	Toyota	Innova	605000	110000	Individual	2494	Manual	12.99	Diesel	100.6	8	8
20	22	Maruti Baleno	Maruti	Baleno	600000	20000	Individual	1197	Manual	21.4	Petrol	83.1	5	6
21	23	Maruti Swift Dzire	Maruti	Swift Dzire	575000	40000	Individual	1197	Manual	20.85	Petrol	83.14	5	5
22	25	Volkswagen Vento	Volkswagen	Vento	425000	47000	Dealer	1598	Manual	16.09	Petrol	103.2	5	8

DATASET

Used Car Dataset from CarDekho website available on kaggle:

Data Description (Feature Information)

- **car_name**: Full car name (brand + model).
- **brand:** Car brand name.
- **model:** Specific model name.
- **seller_type:** Type of seller (individual/dealer).
- **fuel_type:** Type of fuel (petrol/diesel/CNG).
- **transmission_type:** Transmission type (manual/automatic).
- **vehicle_age:** Years since purchase.
- **mileage:** Kilometers per liter (km/l).
- **engine:** Engine capacity (cc).
- **max_power:** Maximum power (BHP).
- **seats:** Number of seats.
- **selling_price:** Listed selling price.

UNDERSTANDING DATA

Understanding the Data

```
[ ] df.head()
```

Show hidden output

```
df.info(7)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15411 entries, 0 to 19543
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   car_name           15411 non-null  object
 1   brand              15411 non-null  object
 2   model              15411 non-null  object
 3   vehicle_age        15411 non-null  int64
 4   km_driven          15411 non-null  int64
 5   seller_type        15411 non-null  object
 6   fuel_type          15411 non-null  object
 7   transmission_type  15411 non-null  object
 8   mileage            15411 non-null  float64
 9   engine             15411 non-null  int64
 10  max_power          15411 non-null  float64
 11  seats              15411 non-null  int64
 12  selling_price      15411 non-null  int64
dtypes: float64(2), int64(5), object(6)
memory usage: 1.6+ MB
```

```
[ ] print('The size of Dataframe is: ', df.shape)
    print('-'*100)
    print('The Column Name, Record Count and Data Types are as follows: ')
    df.info()
    print('-'*100)
```

```
The size of Dataframe is:  (15411, 13)
----------------------------------------------------------------------------------------------------
```

Numerical and categorical columns are

```
[ ] # Defining numerical & categorical columns
    numeric_features = [feature for feature in df.columns if df[feature].dtype != 'O']
    categorical_features = [feature for feature in df.columns if df[feature].dtype == 'O']

    # print columns
    print('We have {} numerical features : {}'.format(len(numeric_features), numeric_features))
    print('\nWe have {} categorical features : {}'.format(len(categorical_features), categorical_features))
```

```
We have 7 numerical features : ['vehicle_age', 'km_driven', 'mileage', 'engine', 'max_power', 'seats', 'selling_price']

We have 6 categorical features : ['car_name', 'brand', 'model', 'seller_type', 'fuel_type', 'transmission_type']
```

```
print('Missing Value Presence in different columns of DataFrame are as follows : ')
print('-'*100)
total=df.isnull().sum().sort_values(ascending=False)
percent=(df.isnull().sum()/df.isnull().count()*100).sort_values(ascending=False)
pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
```

```
Missing Value Presence in different columns of DataFrame are as follows :
----------------------------------------------------------------------------------------------------
```

	Total	Percent
car_name	0	0.0
brand	0	0.0
model	0	0.0
vehicle_age	0	0.0
km_driven	0	0.0
seller_type	0	0.0
fuel_type	0	0.0
transmission_type	0	0.0
mileage	0	0.0
engine	0	0.0
max_power	0	0.0
seats	0	0.0
selling_price	0	0.0

STATISTICAL EDA

Overview of Dataset:

- **Number of records:** 15,411
- **Features:** 13 (7 numerical, 6 categorical)
- ***Examples*:**
  - **Numerical:** Vehicle Age, Mileage, Engine, Selling Price
  - **Categorical:** Brand, Fuel Type, Transmission Type

Summary Statistics:

- ***Average vehicle age*:** 6.03 years
- ***Average selling price*:** ₹7.75 lakhs
- ***Mileage range*:** 4.0 to 33.5 km
- ***Outliers detected in*:** km_driven, engine, selling_price, max_power

STATISTICAL EDA

Summary Statics of Numerical Features

```
print('Summary Statistics of numerical features for DataFrame are as follows:')
print('-'*100)
df.describe()
```

```
Summary Statistics of numerical features for DataFrame are as follows:
----------------------------------------------------------------------------------------------------
```

	vehicle_age	km_driven	mileage	engine	max_power	seats	selling_price
count	15411.000000	1.541100e+04	15411.000000	15411.000000	15411.000000	15411.000000	1.541100e+04
mean	6.036338	5.561648e+04	19.701151	1486.057751	100.588254	5.325482	7.749711e+05
std	3.013291	5.161855e+04	4.171265	521.106696	42.972979	0.807628	8.941284e+05
min	0.000000	1.000000e+02	4.000000	793.000000	38.400000	0.000000	4.000000e+04
25%	4.000000	3.000000e+04	17.000000	1197.000000	74.000000	5.000000	3.850000e+05
50%	6.000000	5.000000e+04	19.670000	1248.000000	88.500000	5.000000	5.560000e+05
75%	8.000000	7.000000e+04	22.700000	1582.000000	117.300000	5.000000	8.250000e+05
max	29.000000	3.800000e+06	33.540000	6592.000000	626.000000	9.000000	3.950000e+07

Summary statics for categorical features

```
[11] print('Summary Statistics of categorical features for DataFrame are as follows:')
     print('-'*100)
     df.describe(include= 'object')
```

```
Summary Statistics of categorical features for DataFrame are as follows:
----------------------------------------------------------------------------------------------------
```

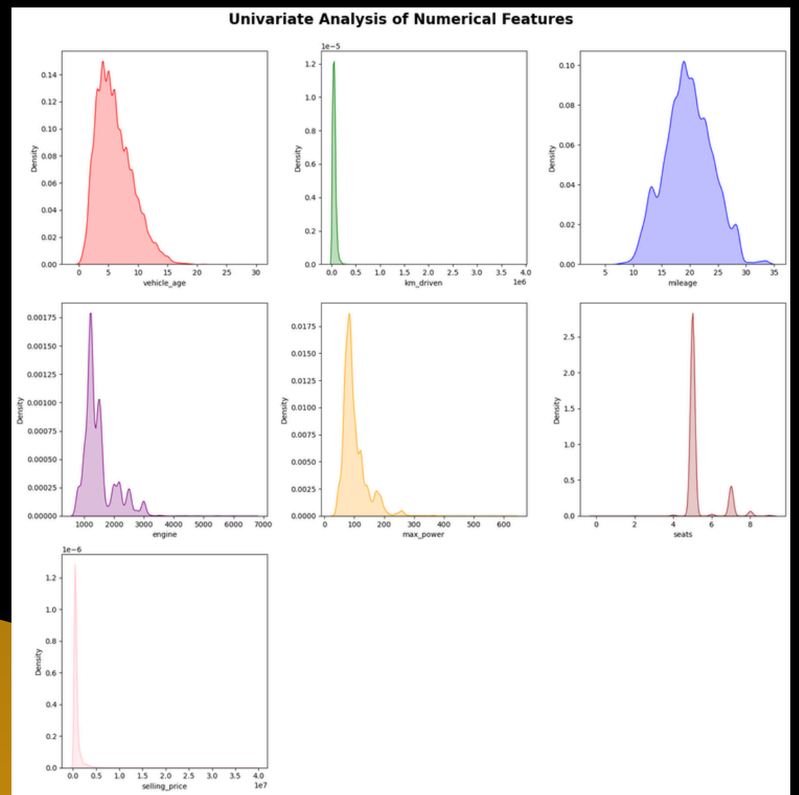
	car_name	brand	model	seller_type	fuel_type	transmission_type
count	15411	15411	15411	15411	15411	15411
unique	121	32	120	3	5	2
top	Hyundai i20	Maruti	i20	Dealer	Petrol	Manual
freq	906	4992	906	9539	7643	12225

GRAPHICAL EDA

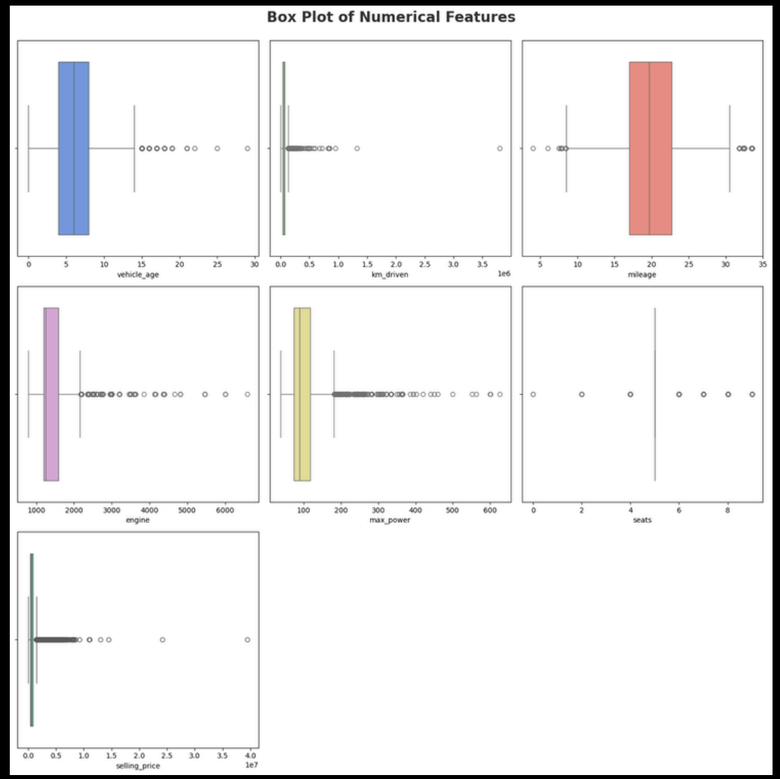
UNIVARIATE ANALYSIS OF NUMERICAL FEATURES

Numerical Features

```
[ ] # Settings for the figure
    plt.figure(figsize=(15, 15))
    plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight='bold', y=1.0)

    # Colors for each plot
    colors = ['red', 'green', 'blue', 'purple', 'orange', 'brown', 'pink', 'gray', 'olive']

    # Loop to create a KDE plot for each feature
    for i, feature in enumerate(numeric_features):
        plt.subplot(3, 3, i+1)
        sns.kdeplot(x=df[feature], shade=True, color=colors[i % len(colors)])
        plt.xlabel(feature)
        plt.tight_layout()
```




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

  # Setting up the figure
  plt.figure(figsize=(15, 15))
  plt.suptitle('Box Plot of Numerical Features', fontsize=20, fontweight='bold', alpha=0.8, y=1.0)

  # List of medium-light colors for the plots
  medium_light_colors = ['cornflowerblue', 'lightgreen', 'salmon', 'plum', 'khaki', 'lightsteelblue', 'mediumaquamarine', 'thistle', 'lightsalmon']

  for i in range(len(numeric_features)):
      plt.subplot(3, 3, i+1)
      # Apply a medium-light color from the list
      sns.boxplot(x=df[numeric_features[i]], color=medium_light_colors[i % len(medium_light_colors)])
      plt.xlabel(numeric_features[i])
      plt.tight_layout()
```



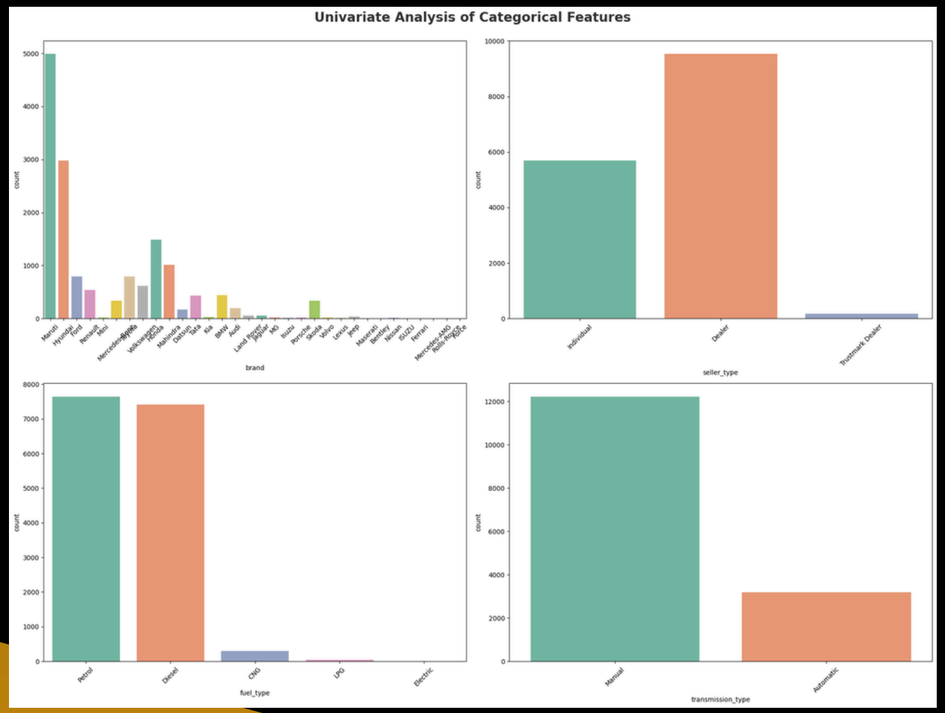
Report

- Km_driven, max_power, selling_price, and engine are right skewed and postively skewed.
- Almost every feature has outliers, which could be actual extreme values or potential errors in data entry.
- Several features (km_driven, engine, max power, selling price) are right-skewed, which might necessitate transformations (like logarithmic) before conducting advanced statistical analyses

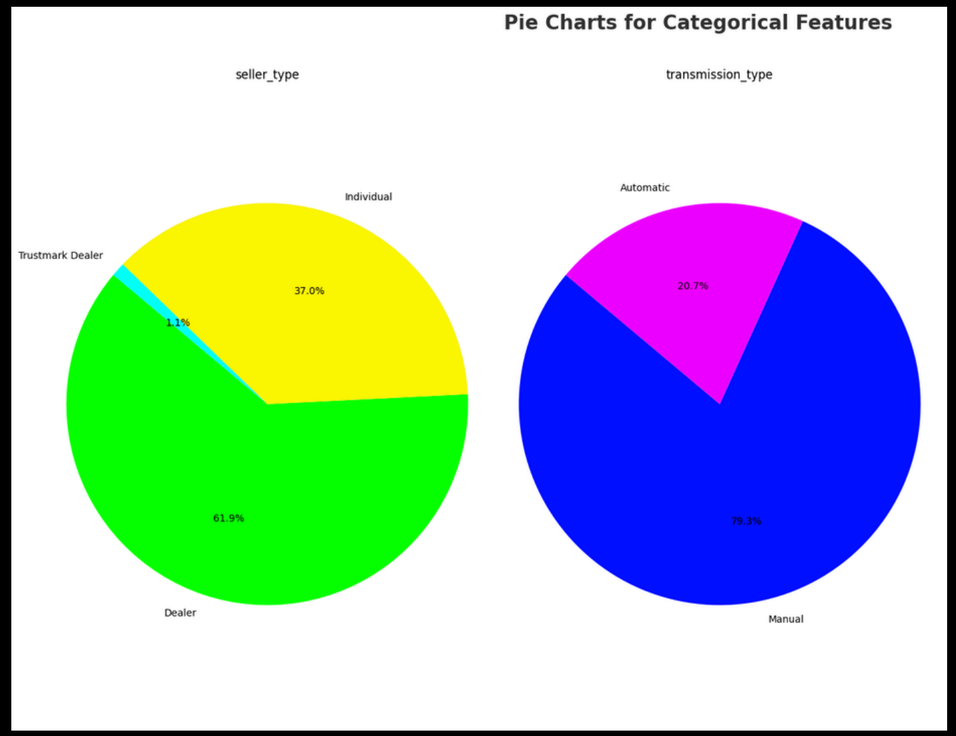
UNIVARIATE ANALYSIS OF CATEGORICAL FEATURES

Univariate analysis of categorical features

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

# Setting up the figure
plt.figure(figsize=(20, 15))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fontweight='bold', alpha=0.8, y=1.0)

# List of categorical features
cat1 = ['brand', 'seller_type', 'fuel_type', 'transmission_type']

# Loop through each categorical feature and plot
for i, feature in enumerate(cat1):
    plt.subplot(2, 2, i+1)
    # Use a palette that provides a unique color for each category
    sns.countplot(x=df[feature], palette='Set2')  # 'Set2' is an example of a qualitative color palette
    plt.xlabel(feature)
    plt.xticks(rotation=45)
    plt.tight_layout()
```



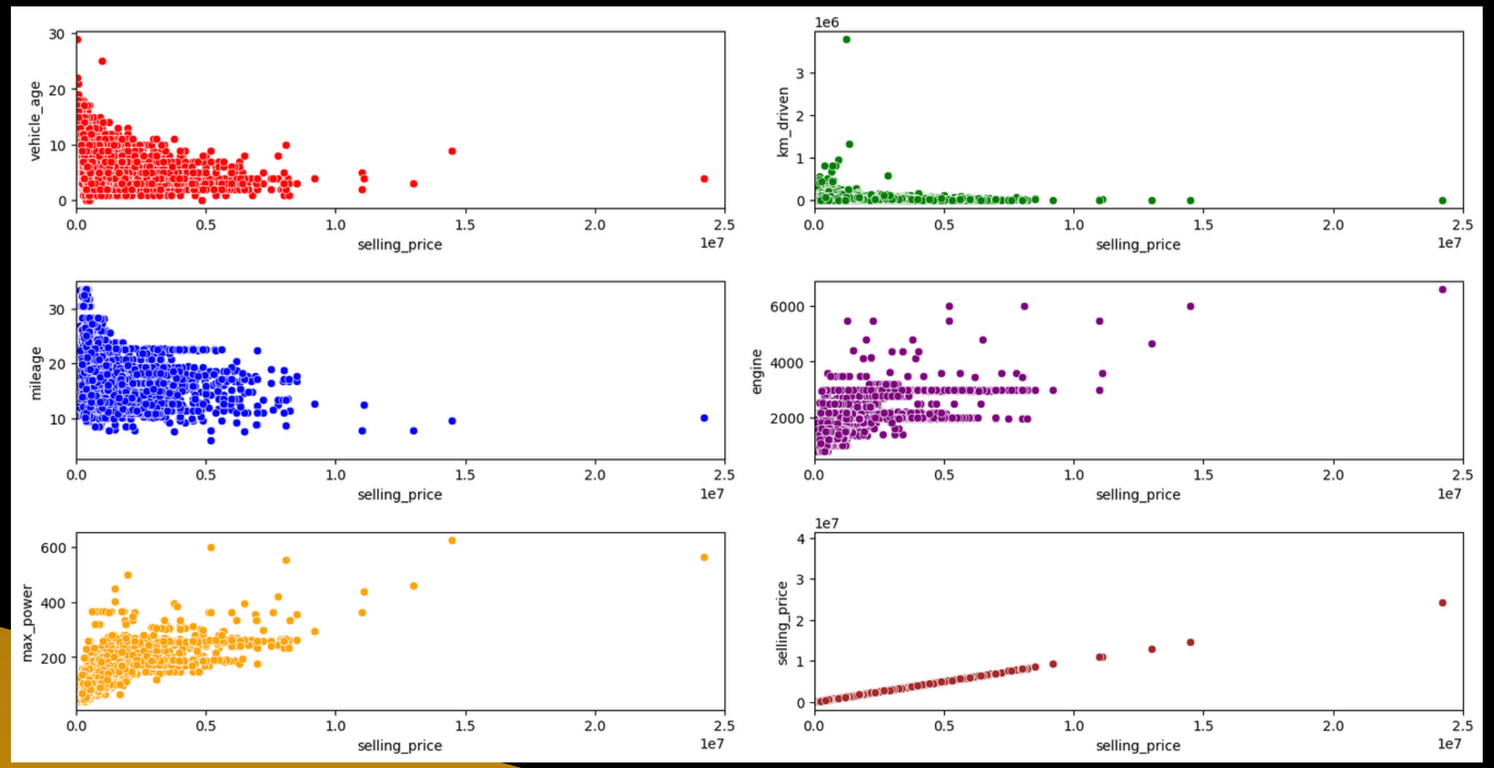

```
# Setting up the figure
plt.figure(figsize=(20, 10))  # Adjusted size based on fewer plots
plt.suptitle('Pie Charts for Categorical Features', fontsize=20, fontweight='bold', alpha=0.8, y=1.02)

# List of categorical features, excluding 'brand'
cat1 = ['seller_type', 'transmission_type']

# Generating a large palette with unique colors
palette = sns.color_palette("hsv", n_colors=df[cat1].nunique().sum())

# Accumulating unique colors for each category in a dictionary
color_dict = {}
idx = 0
for feature in cat1:
    unique_vals = df[feature].unique()
    for val in unique_vals:
        if val not in color_dict:
            color_dict[val] = palette[idx]
            idx += 1  # Move to the next color in the palette

# Loop through each categorical feature and plot a pie chart
for i, feature in enumerate(cat1):
    plt.subplot(1, 3, i+1)  # Adjusted subplot layout for three plots
    # Getting the count of each category
    count_data = df[feature].value_counts()
    colors = [color_dict[val] for val in count_data.index]  # Color for each category
    plt.pie(count_data, labels=count_data.index, autopct='%1.1f%%', startangle=140, colors=colors)
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
    plt.title(feature)

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



BIVARIATE ANALYSIS OF NUMERICAL FEATURES

5.2 Bivariate Analysis

```
[ ] continuous_features=[feature for feature in numeric_features if len(df[feature].unique())>=10]
    print('Num of continuos features:',continuous_features)

Num of continuos features: ['vehicle_age', 'km_driven', 'mileage', 'engine', 'max_power', 'selling_price']

[ ] # Assuming 'df' is your DataFrame and 'continuous_features' are defined
    fig = plt.figure(figsize=(15, 20))

    # List of colors for the plots
    colors = ['red', 'green', 'blue', 'purple', 'orange', 'brown', 'pink', 'gray', 'olive', 'cyan', 'magenta', 'yellow', 'black', 'lime', 'teal']

    for i in range(len(continuous_features)):
        ax = plt.subplot(8, 2, i+1)

        # Apply a different color from the list for each plot
        sns.scatterplot(data=df, x='selling_price', y=continuous_features[i], color=colors[i % len(colors)])
        plt.xlim(0, 25000000)  # Limit to 25 lakhs Rupees to view clean
        plt.tight_layout()

    plt.show()
```



Insights

- **Vehicle Age vs. Selling Price**:
  - *Negative correlation*: Older vehicles generally have lower selling prices.
- **Mileage vs. Selling Price**:
  - *No clear trend*: Vehicle prices do not significantly vary with changes in mileage.
- **Kilometers Driven vs. Selling Price**:
  - *No strong impact*: Less-used vehicles show no consistent pricing pattern.
- **Engine Size vs. Selling Price**:
  - *Moderate positive correlation*: Larger engine sizes often correlate with higher selling prices.
- **Max Power vs. Selling Price**:
  - *Positive correlation*: Higher maximum power typically leads to higher selling prices.

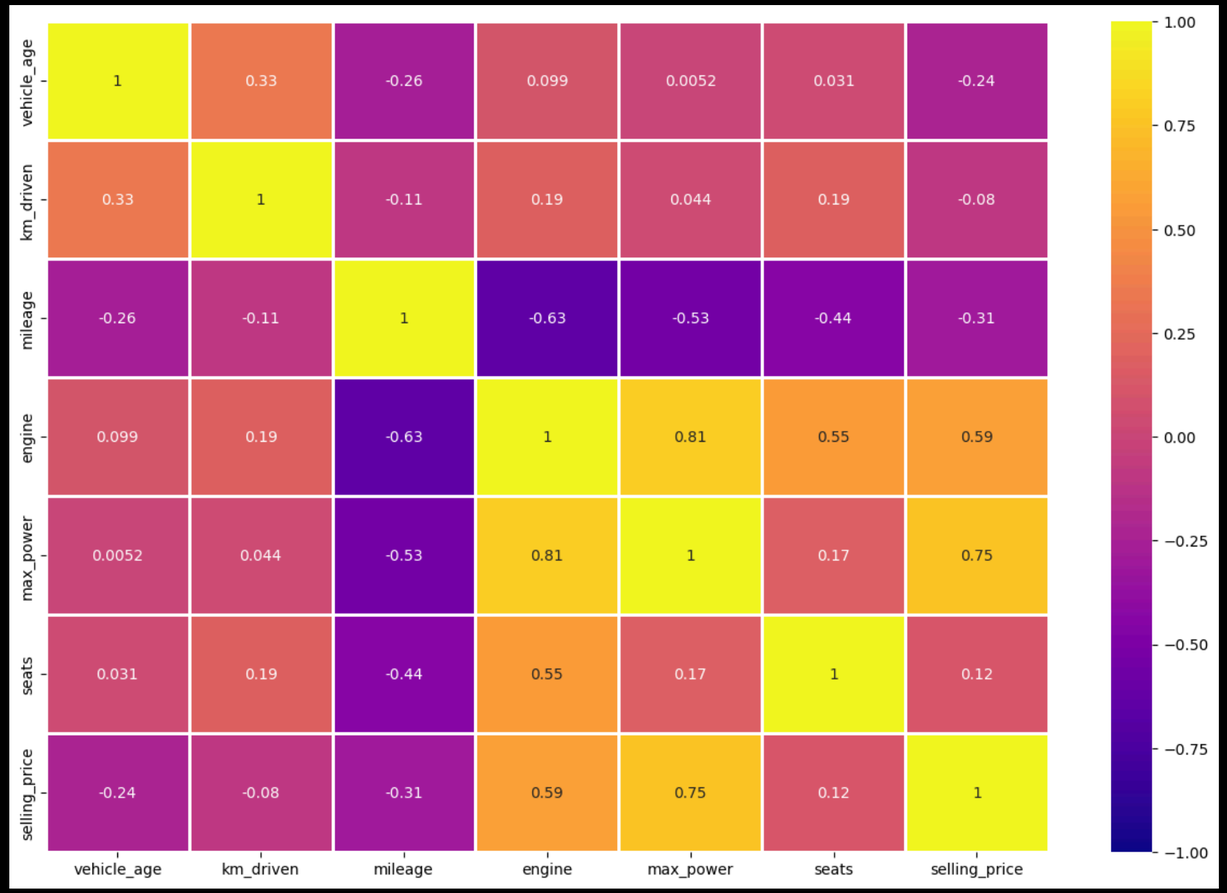
MULTIVARIATE ANALYSIS OF NUMERICAL FEATURES

Check Multicollinearity in Numerical features

```
[ ] df[numeric_features].corr()
```

	vehicle_age	km_driven	mileage	engine	max_power	seats	selling_price
vehicle_age	1.000000	0.333891	-0.257394	0.098965	0.005208	0.030791	-0.241851
km_driven	0.333891	1.000000	-0.105239	0.192885	0.044421	0.192830	-0.080030
mileage	-0.257394	-0.105239	1.000000	-0.632987	-0.533128	-0.440280	-0.305549
engine	0.098965	0.192885	-0.632987	1.000000	0.807368	0.551236	0.585844
max_power	0.005208	0.044421	-0.533128	0.807368	1.000000	0.172257	0.750236
seats	0.030791	0.192830	-0.440280	0.551236	0.172257	1.000000	0.115033
selling_price	-0.241851	-0.080030	-0.305549	0.585844	0.750236	0.115033	1.000000

```
[ ] plt.figure(figsize=(15,10))
    sns.heatmap(data = df[numeric_features].corr(), annot= True, cmap= 'plasma', vmin= -1 , vmax= 1, linecolor='white', linewidths=2)
    plt.show()
```



Insights

1. *Max Power and Engine are highly correlated*
2. *Max Power and Selling Price are highly correlated.*

MULTIVARIATE ANALYSIS OF CATEGORCAL FEATURES

Check Multicollinearity for Categorical features

- A chi-squared test (also chi-square or χ2 test) is a statistical hypothesis test that is valid to perform when the test statistic is chi-squared distributed under the null hypothesis, specifically Pearson's chi-squared test
- A chi-square statistic is one way to show a relationship between two categorical variables.
- Here we test correlation of Categorical columns with Target column i.e Selling Price

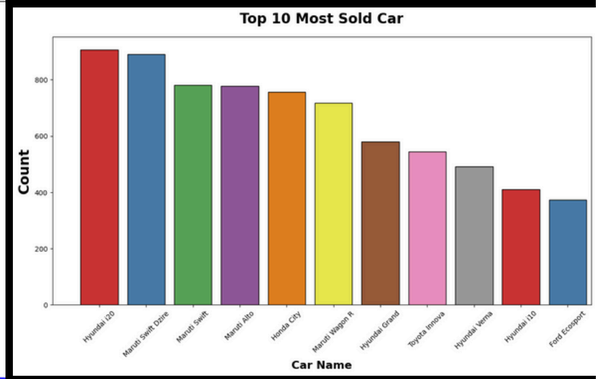
```
[ ] from scipy.stats import chi2_contingency
    chi2_test = []

    for feature in categorical_features:
        if chi2_contingency(pd.crosstab(df['selling_price'], df[feature]))[1] < 0.05:
            chi2_test.append('Reject Null Hypothesis')
        else:
            chi2_test.append('Fail to Reject Null Hypothesis')
    test_result = pd.DataFrame(data=[categorical_features, chi2_test]).T
    test_result.columns = ['Categorical Features', 'Hypothesis Result']

    print('-'*100)
    print('Chi-Squarred Test (Checking Multi-collinearity for  Categorical features) results are as follows :')
    print('-'*100)

    test_result
```

```
----------------------------------------------------------------------------------------------------
Chi-Squarred Test (Checking Multi-collinearity for  Categorical features) results are as follows :
----------------------------------------------------------------------------------------------------
```

	Categorical Features	Hypothesis Result
0	car_name	Reject Null Hypothesis
1	brand	Reject Null Hypothesis
2	model	Reject Null Hypothesis
3	seller_type	Reject Null Hypothesis
4	fuel_type	Reject Null Hypothesis
5	transmission_type	Reject Null Hypothesis

For all categorical features, the null hypothesis was rejected, meaning these features significantly affect the selling price. They should be included in further analysis and modeling.

GRAPHICAL EDA:



- From the chart it is clear that the Target Variable Skewed

- Check mean price of Hyundai i20 which is most sold

```
[ ] i20 = df[df['car_name'] == 'Hyundai i20']['selling_price'].mean()
    print(f'The mean price of Hyundai i20 is {i20:.2f} Rupees')
```

```
The mean price of Hyundai i20 is 543603.75 Rupees
```

Insights

- As per the Chart these are top 10 most selling cars in used car website.
- Of the total cars sold Hyundai i20 shares 5.8% of total ads posted and followed by Maruti Swift Dzire.
- Mean Price of Most Sold Car is 5.4 lakhs.
- This Feature has impact on the Target Variable.

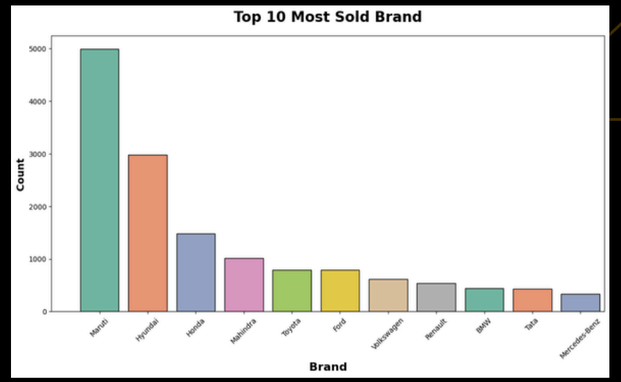
Most Sold Cars on CarDekho Website

```
[ ] print('-'*50)
    print('Top 10 Sold Cars on CarDekho Website')
    print('-'*50)

    df.car_name.value_counts()[0:10]
```

```
--------------------------------------------------
Top 10 Sold Cars on CarDekho Website
--------------------------------------------------
```

car_name	count
Hyundai i20	906
Maruti Swift Dzire	890
Maruti Swift	781
Maruti Alto	778
Honda City	757
Maruti Wagon R	717
Hyundai Grand	580
Toyota Innova	545
Hyundai Verna	492
Hyundai i10	410

dtype: int64



Most Sold Car Brand on CarDekho Website

```
[ ] print('-'*50)
    print('Top 10 Most Sold Car Brand')
    print('-'*50)

    df.brand.value_counts()[0:10]
```

```
--------------------------------------------------
Top 10 Most Sold Car Brand
--------------------------------------------------
```

brand	count
Maruti	4992
Hyundai	2982
Honda	1485
Mahindra	1011
Toyota	793
Ford	790
Volkswagen	620
Renault	536
BMW	439
Tata	430

dtype: int64

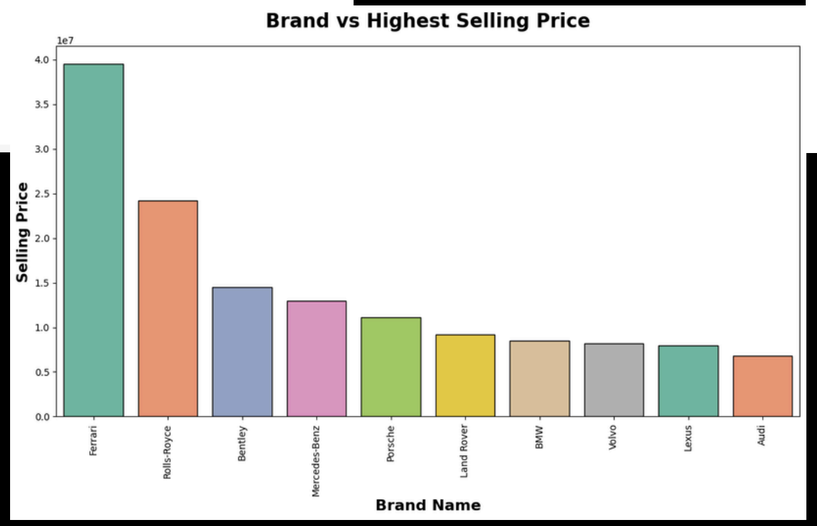
GRAPHICAL EDA:

Costlier Brand on CarDekho Website

```
[ ] brand = df.groupby('brand').selling_price.max()
    brand = brand.to_frame().sort_values('selling_price',ascending=False)[0:10]

    print('-'*50)
    print('Top 10 Costlier Brands on CarDekho Website')
    print('-'*50)

    brand
```

```
--------------------------------------------------
Top 10 Costlier Brands on CarDekho Website
--------------------------------------------------
```

brand	selling_price
Ferrari	39500000
Rolls-Royce	24200000
Bentley	14500000
Mercedes-Benz	13000000
Porsche	11100000
Land Rover	9200000
BMW	8500000
Volvo	8195000
Lexus	8000000
Audi	6800000

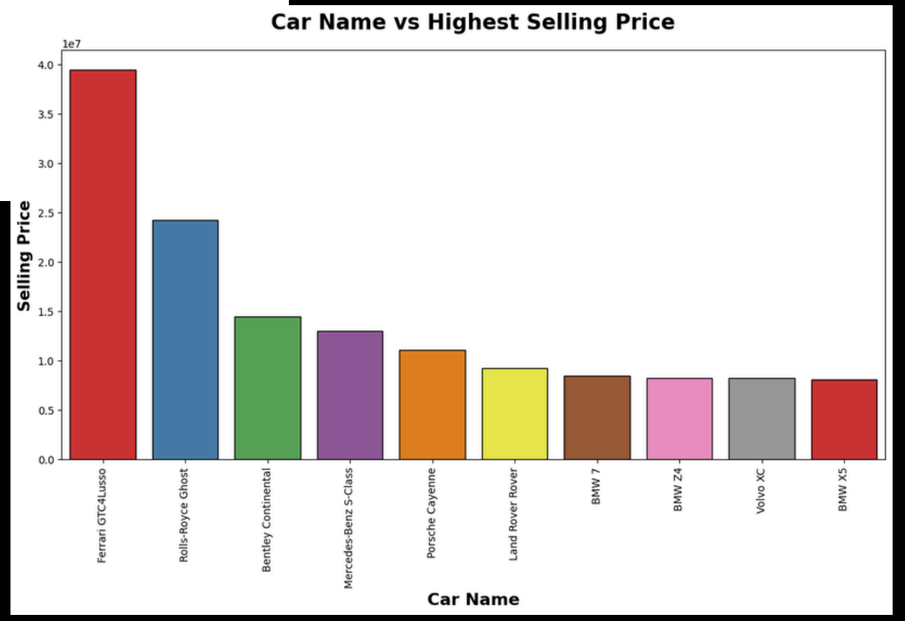


Costlier Car on CarDekho Website

```
[ ] car= df.groupby('car_name').selling_price.max()
    car =car.to_frame().sort_values('selling_price',ascending=False)[0:10]

    print('-'*50)
    print('Top 10 Costlier Cars on CarDekho Website')
    print('-'*50)

    car
```

```
--------------------------------------------------
Top 10 Costlier Cars on CarDekho Website
--------------------------------------------------
```

car_name	selling_price
Ferrari GTC4Lusso	39500000
Rolls-Royce Ghost	24200000
Bentley Continental	14500000
Mercedes-Benz S-Class	13000000
Porsche Cayenne	11100000
Land Rover Rover	9200000
BMW 7	8500000
BMW Z4	8250000
Volvo XC	8195000
BMW X5	8100000



GRAPHICAL EDA:

Most Mileage Car Brand on CarDekho Website

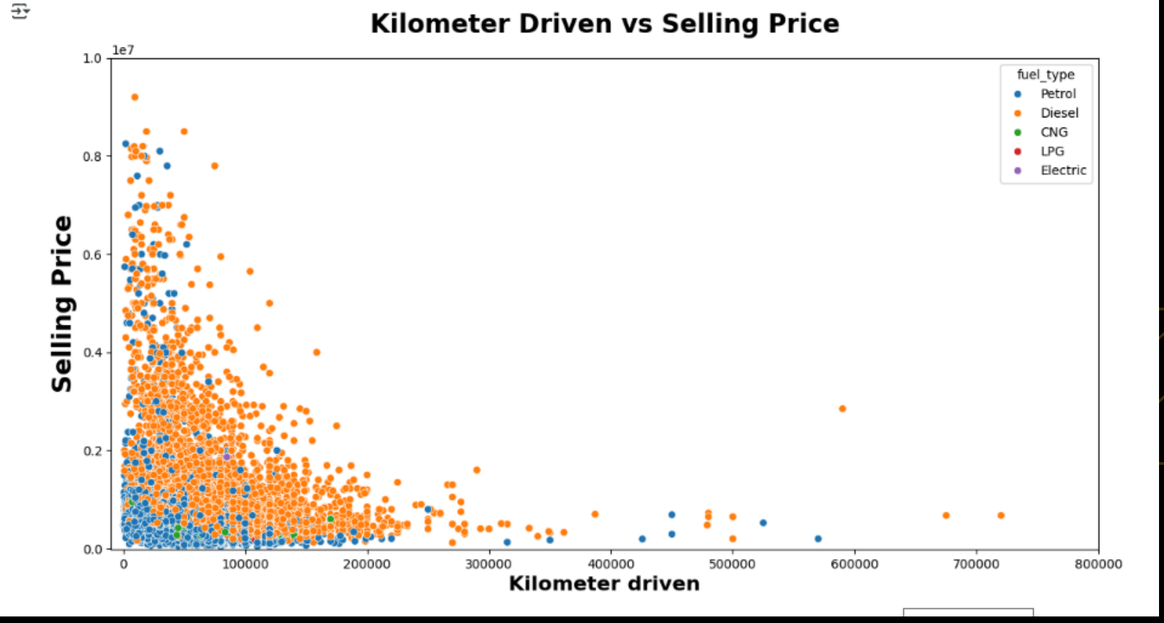
```
[ ] mileage= df.groupby('brand')['mileage'].mean().sort_values(ascending=False).head(15)

    print('-'*50)
    print('Most Mileage Car Brand on CarDekho Website')
    print('-'*50)

    mileage.to_frame()
```

```
--------------------------------------------------
Most Mileage Car Brand on CarDekho Website
--------------------------------------------------
```

brand	mileage
Maruti	22.430980
Renault	22.099142
Datsun	21.215647
Lexus	20.846000
Ford	19.922620
Honda	19.908795
Maserati	19.820000
Tata	19.755279
Hyundai	19.588776
Volkswagen	18.689774
Mini	18.287647
Skoda	17.667006
BMW	17.440182
Kia	17.323125
Force	17.000000

Kilometers Driven Vs Selling Price

```
[ ] plt.subplots(figsize=(14,7))
    sns.scatterplot(x="km_driven", y='selling_price', data=df,ec = "white",color='b', hue='fuel_type')
    plt.title("Kilometer Driven vs Selling Price", weight="bold",fontsize=20, pad=20)
    plt.ylabel("Selling Price", weight="bold", fontsize=20)
    plt.xlim(-10000,800000) #used limit for better visualization
    plt.ylim(-10000,10000000)
    plt.xlabel("Kilometer driven", weight="bold", fontsize=16)
    plt.show()
```



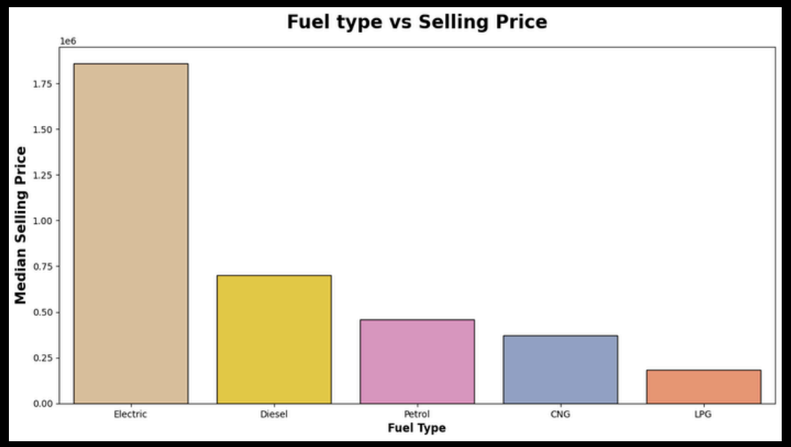
GRAPHICAL EDA:

Fuel Type Vs Selling Price

```
[ ] fuel = df.groupby('fuel_type')['selling_price'].median().sort_values(ascending=False)
    fuel.to_frame()
```

fuel_type	selling_price
Electric	1857500.0
Diesel	700000.0
Petrol	460000.0
CNG	370000.0
LPG	182500.0

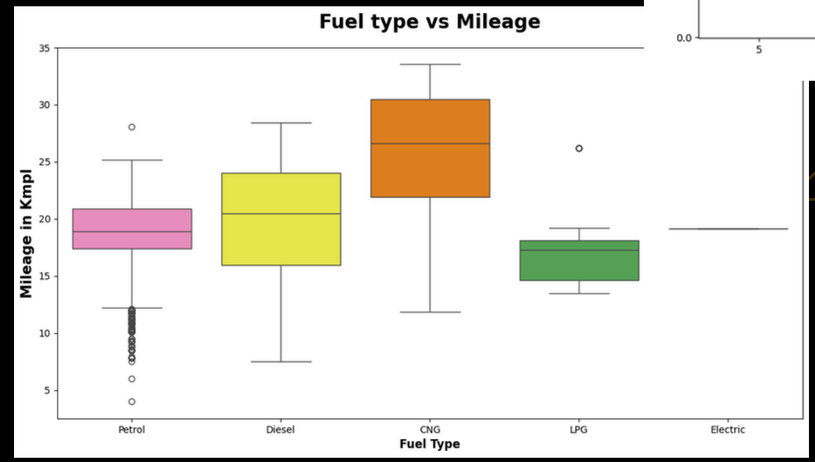
```
[ ] plt.subplots(figsize=(14,7))
    sns.barplot(x=fuel.index, y=fuel.values, ec = "black", palette="Set2_r")
    plt.title("Fuel type vs Selling Price", weight="bold",fontsize=20, pad=20)
    plt.ylabel("Median Selling Price", weight="bold", fontsize=15)
    plt.xlabel("Fuel Type", weight="bold", fontsize=12)
    plt.show()
```

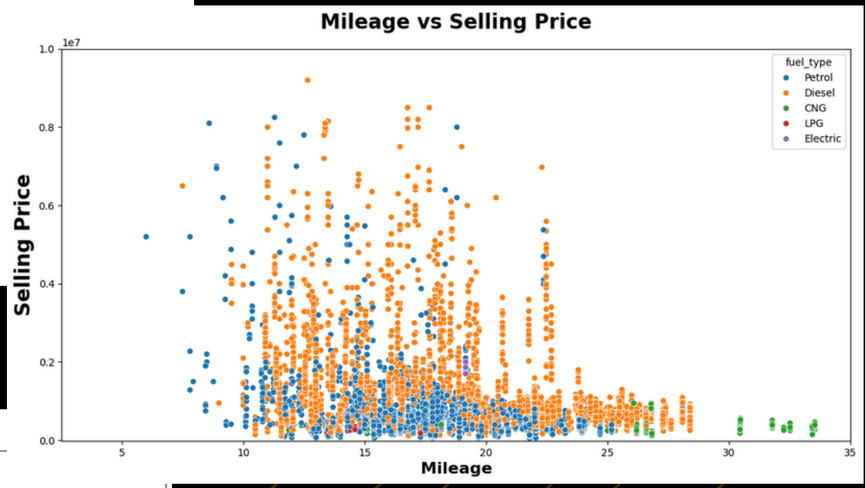


Fuel Type Vs Mileage

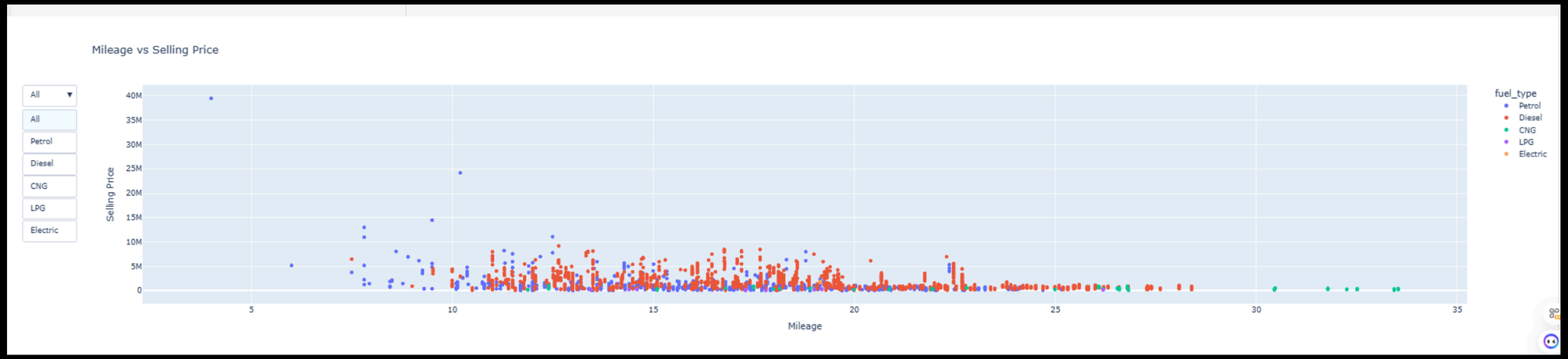
```
] fuel_mileage = df.groupby('fuel_type')['mileage'].mean().sort_values(ascending=False)
  fuel_mileage.to_frame()
```

fuel_type	mileage
CNG	25.814651
Diesel	20.060030
Electric	19.160000
Petrol	19.123045
LPG	17.836364



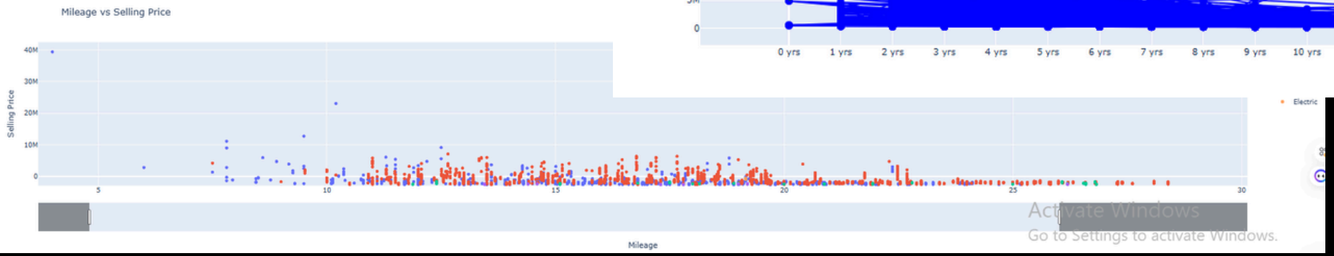


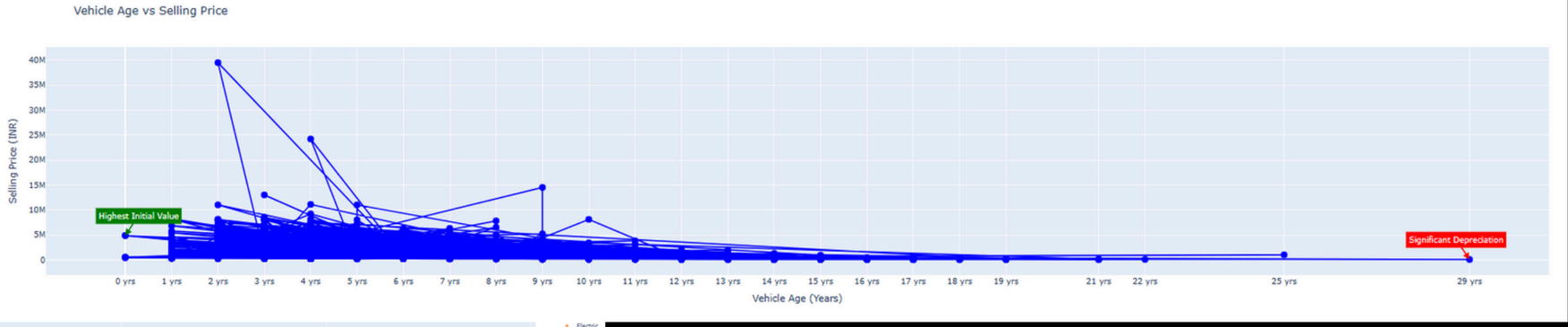
INTERATIVE EDA:



Adding a Range Slider for Mileage

```
# Adding range slider for 'mileage'
fig = px.scatter(
    df,
    x='mileage',
    y='selling_price',
    color='fuel_type',
    labels={'mileage': 'Mileage', 'selling_price': 'Selling Price'},
    title="Mileage vs Selling Price"
)

# Add range slider for the x-axis (mileage)
fig.update_layout(
    xaxis=dict(
        rangeslider=dict(visible=True),
        title="Mileage"
    )
)
fig.show()
```





GRAPHICAL EDA:

```
[ ] oldest = df.groupby('car_name')['vehicle_age'].max().sort_values(ascending=False).head(10)
    oldest.to_frame()
```

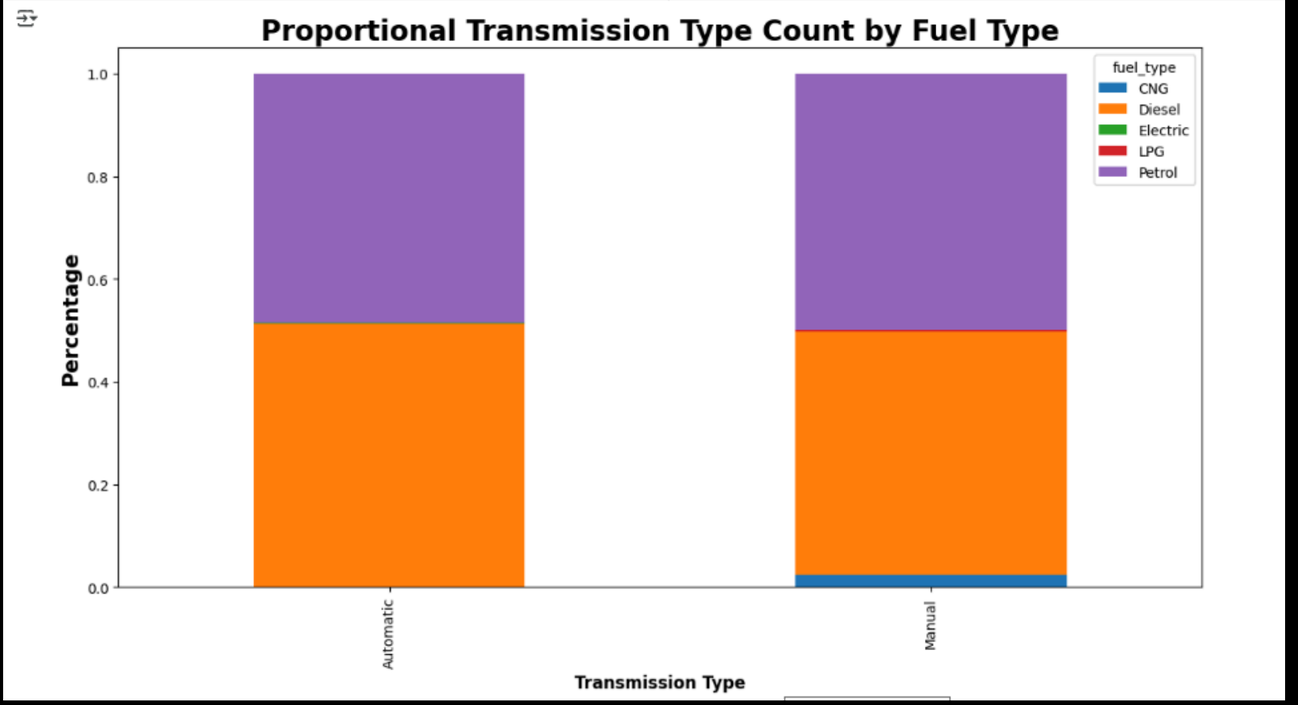
car_name	vehicle_age
Maruti Alto	29
BMW 3	25
Honda City	22
Maruti Wagon R	21
Mahindra Bolero	18
Mahindra Scorpio	18
Skoda Octavia	18
Honda CR-V	17
Mercedes-Benz E-Class	17
Honda Civic	15

Insight

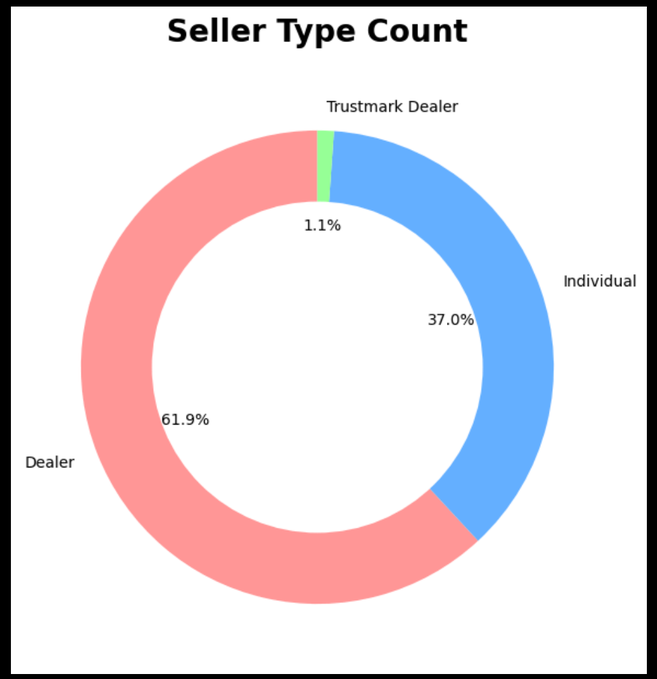
- Maruti Alto is the Oldest car available 29 years old in the used car website followed by BMW 3 for 25 years old.

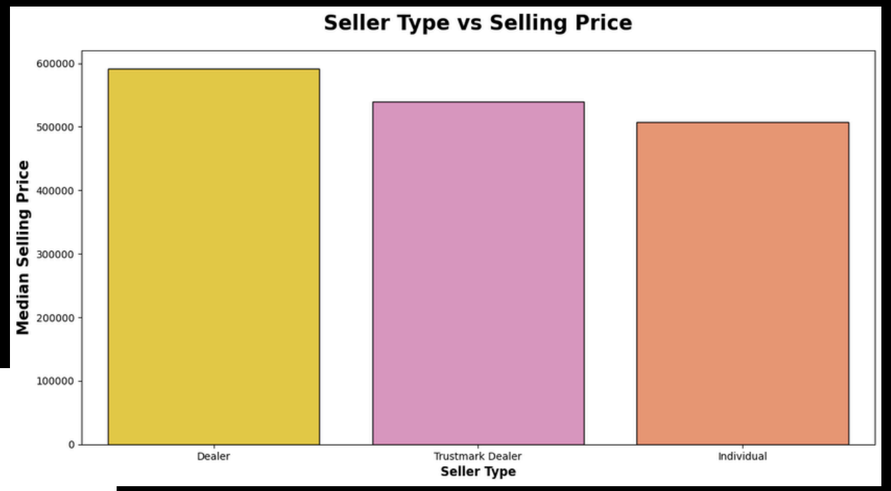
Transmission Type Vs Count

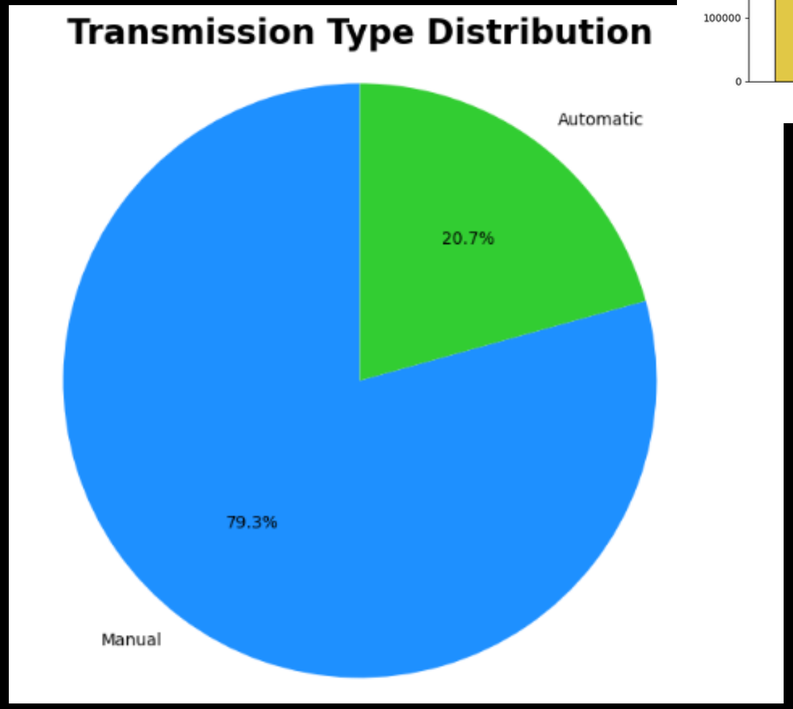
```
[ ] df_grouped = df.groupby(['transmission_type', 'fuel_type']).size().unstack().fillna(0)
    df_percent = df_grouped.div(df_grouped.sum(axis=1), axis=0)
    df_percent.plot(kind='bar', stacked=True, figsize=(14, 7))
    plt.title("Proportional Transmission Type Count by Fuel Type", weight="bold", fontsize=20)
    plt.xlabel("Transmission Type", weight="bold", fontsize=12)
    plt.ylabel("Percentage", weight="bold", fontsize=15)
    plt.show()
```



GRAPHICAL EDA:







Final Report

- The datatypes and Column names were right and there was 15411 rows and 13 columns
- The `selling_price` column is the target to predict. i.e Regression Problem.
- There are outliers in the `km_driven`, `enginer`, `selling_price`, and `max power`.
- Dealers are the highest sellers of the used cars.
- Skewness is found in few of the columns will check it after handling outliers.
- Vehicle age has negative impact on the price.
- Manual cars are mostly sold and automatic has higher selling average than manual cars.
- Petrol is the most preffered choice of fuel in used car website, followed by diesel and LPG.
- We just need less data cleaning for this dataset.

FEATURE ENGINEERING

Dropping Unnessary Columns

Dropping car_name, brand, and model removes features that introduce noise and are not directly correlated with price. This ensures a more accurate and reliable model by focusing on meaningful attributes.

Removing unnecessary features

Dropping `car_name` ,`brand` and `model`

These features are not directly correlated with the price of car and they can actually introduce noise into the model. For example, two cars with the same features but different brands may have different prices. This is because brand reputation and perceived quality can play a role in determining the price of a car. By dropping the `car_name` ,`brand` and `model`, we can create a model that is more accurate and reliable.

```
[ ] df_model.drop(labels=['car_name','brand','model'],axis=1,inplace=True)

    df_model
```

	vehicle_age	km_driven	seller_type	fuel_type	transmission_type	mileage	engine	max_power	seats	selling_price
0	9	120000	Individual	Petrol	Manual	19.70	796	46.30	5	120000
1	5	20000	Individual	Petrol	Manual	18.90	1197	82.00	5	550000
2	11	60000	Individual	Petrol	Manual	17.00	1197	80.00	5	215000
3	9	37000	Individual	Petrol	Manual	20.92	998	67.10	5	226000
4	6	30000	Dealer	Diesel	Manual	22.77	1498	98.59	5	570000
...	...	...	...	...	...	...	...	...	...	...
19537	9	10723	Dealer	Petrol	Manual	19.81	1086	68.05	5	250000
19540	2	18000	Dealer	Petrol	Manual	17.50	1373	91.10	7	925000
19541	6	67000	Dealer	Diesel	Manual	21.14	1498	103.52	5	425000
19542	5	3800000	Dealer	Diesel	Manual	16.00	2179	140.00	7	1225000
19543	2	13000	Dealer	Petrol	Automatic	18.00	1497	117.60	5	1200000

15411 rows × 10 columns

FEATURE ENGINEERING

Converting Categorical Columns into numerical

Using One Hot Encoding (via pd.get_dummies()), categorical variables are converted into numerical format by creating binary columns for each category. This allows models to process categorical data effectively without introducing ordinal bias.

Converting Categorical Columns into numerical

Using `One Hot Encoding (get_dummies)` to convert categorical variables to numerical

```
[ ] df_model=pd.get_dummies(df_model,dtype=float)
    df_model
```

	vehicle_age	km_driven	mileage	engine	max_power	seats	selling_price	seller_type_Dealer	seller_type_Individual	seller_type_Trustmark Dealer	fuel_type_CNG	fuel_type_Diesel	fuel_type_Electric	fuel_type_LPG	fuel_type_Petrol	transmission_type_Automatic	transmission_type_Manual
0	9	120000	19.70	796	46.30	5	120000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
1	5	20000	18.90	1197	82.00	5	550000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
2	11	60000	17.00	1197	80.00	5	215000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
3	9	37000	20.92	998	67.10	5	226000	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
4	6	30000	22.77	1498	98.59	5	570000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
19537	9	10723	19.81	1086	68.05	5	250000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
19540	2	18000	17.50	1373	91.10	7	925000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0
19541	6	67000	21.14	1498	103.52	5	425000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
19542	5	3800000	16.00	2179	140.00	7	1225000	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
19543	2	13000	18.00	1497	117.60	5	1200000	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0

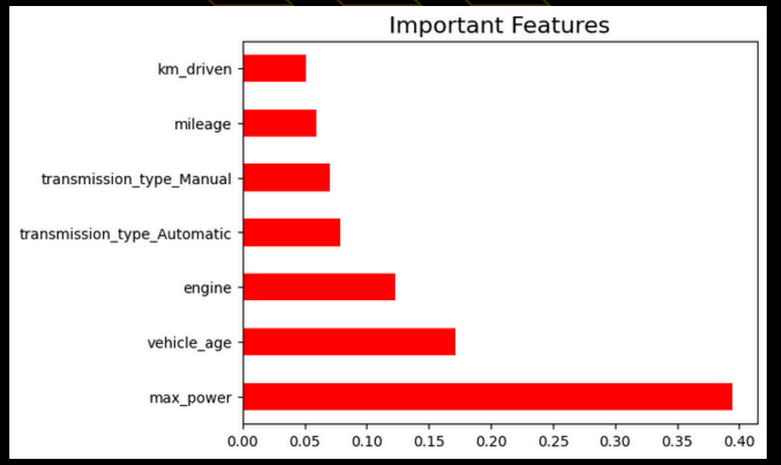
15411 rows × 17 columns

FEATURE ENGINEERING

Checking For important features

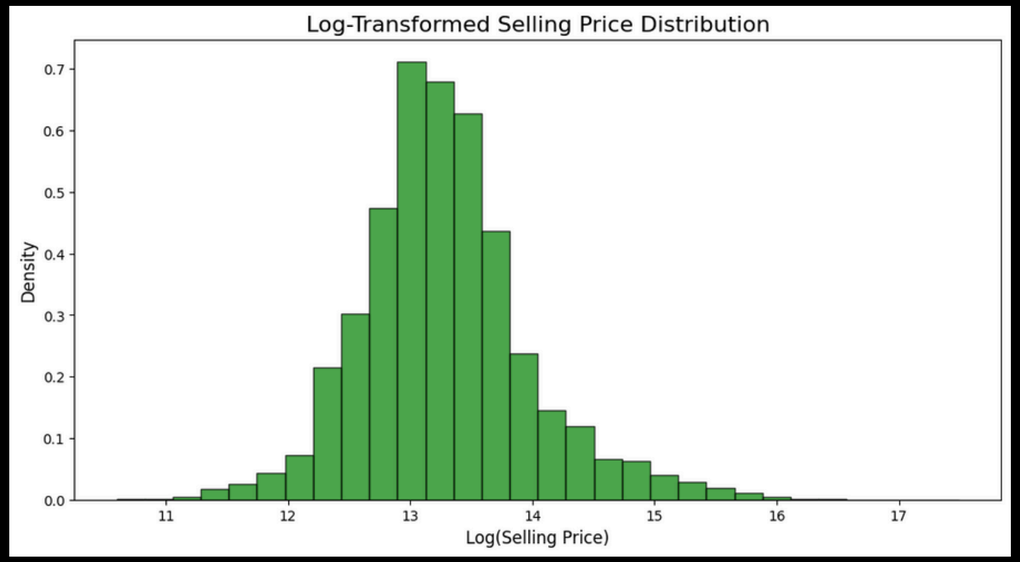
Checking for important features

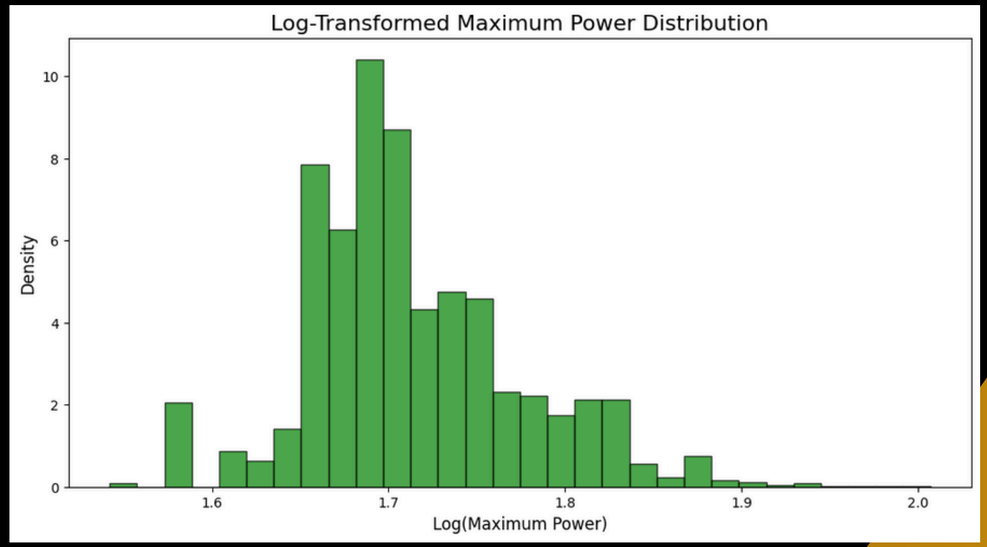
```
[ ] from sklearn.ensemble import ExtraTreesRegressor

    model = ExtraTreesRegressor()

    print(model.fit(X,y))

    ExtraTreesRegressor()

[ ] print('-'*50)
    print('Checking for feature importance')
    print('-'*50)

    print(model.feature_importances_)

    --------------------------------------------------
    Checking for feature importance
    --------------------------------------------------
    [1.71574048e-01 5.10113088e-02 5.94218027e-02 1.22909710e-01
     3.94665693e-01 1.44914647e-02 4.38232069e-03 3.42544344e-03
     2.31306147e-05 5.07937085e-05 7.25130717e-03 1.82935738e-05
     2.16639564e-06 2.15768617e-02 7.89396379e-02 7.02560177e-02]
```



However, we will use all features for prediction

Transformation to handle outliers





MODELING

MODEL BUILDING & EVALUARION

Objective:

The goal of car price prediction is to develop a machine learning model that accurately forecasts the resale value of vehicles based on key attributes like make, model, mileage, and age. This helps ensure transparent and fair pricing, assists buyers and sellers in making informed decisions, and enhances market efficiency.

IMPORT RELEVANT LIBRARIES

```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import ExtraTreesRegressor, RandomForestRegressor, VotingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, explained_variance_score, r2_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import KFold
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn import metrics
from xgboost import XGBRegressor
```

TRAIN TEST SPLIT

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

INITIALIZE AND FIT THE MODEL

```
random_forest_model = RandomForestRegressor(random_state=42)
random_forest_model.fit(X_train, y_train)
```

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

```
[ ]  SVR = SVR()
     SVR.fit(X_train, y_train)
```

```
SVR
SVR()
```

```
[ ]  DecisionTreeRegressor = DecisionTreeRegressor(random_state=1)
     DecisionTreeRegressor.fit(X_train, y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=1)
```

RANDOM FOREST

The RandomForestRegressor is a versatile machine learning model that uses multiple decision trees to predict outcomes more accurately than a single tree could. In the context of car price prediction, it helps in understanding complex relationships between various features like make, model, age, mileage, and condition, providing robust and reliable pricing estimations based on historical data.

SVR

The Support Vector Regressor (SVR) applies the principles of Support Vector Machines to regression problems, creating a model that aims to fit the error within a certain threshold. For car price prediction, SVR is particularly effective when dealing with non-linear relationships and high-dimensional spaces, ensuring that the model can generalize well from the training data to unseen data.

DECISION TREE

The Decision Tree Regressor constructs a model in the form of a tree structure, dividing the dataset into smaller subsets based on descriptive features, which can include aspects like car make, year, and mileage. This approach is intuitive and easy to visualize, making it useful for car price prediction as it clearly shows the decision paths and how different features directly affect the vehicle's estimated value.

INITIALIZE AND FIT THE MODEL

```
[ ] XGBRegressor = XGBRegressor()
    XGBRegressor.fit(X_train, y_train)
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

XGBOOST

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting that is designed to be highly efficient, flexible, and portable. It uses decision trees as base learners and is highly favored for its performance in regression tasks, including car price prediction. XGBoost is particularly valued for its ability to handle large datasets efficiently, manage missing data, and improve prediction accuracy through its robust handling of a variety of data types and relationships.

EVALUATION MATRICS

Mean Squared Error (MSE):

- Measures the average of the squares of errors.
- Lower MSE indicates a better model fit.

Root Mean Squared Error (RMSE):

- Square root of MSE.
- Provides error in the same units as the target variable.
- Lower RMSE means more accurate predictions.

Explained Variance Score:

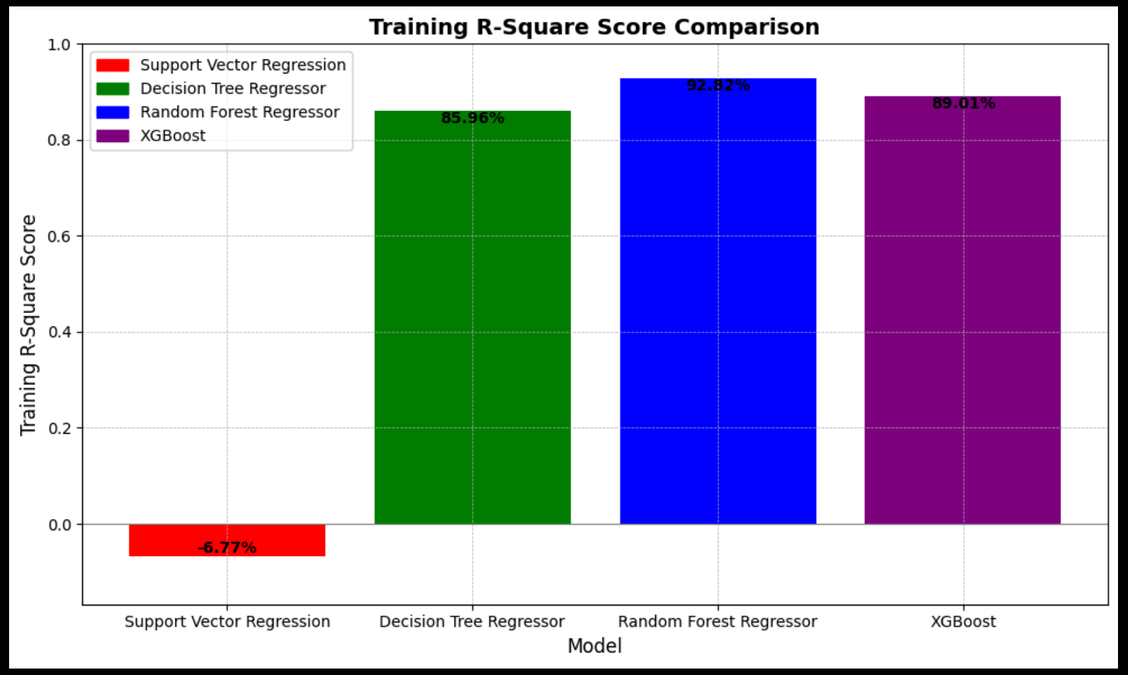
- Indicates the proportion of target variable's variance that the model explains.
- A score of 1 means perfect prediction; closer to 0 suggests poor model performance.

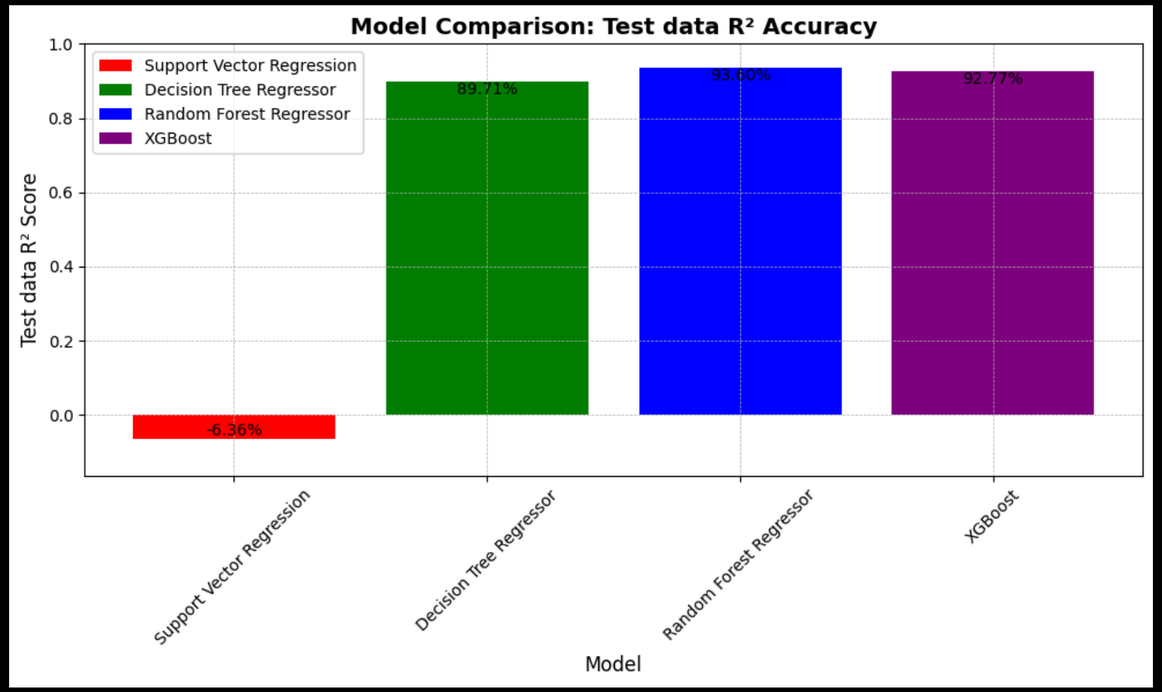
R-Square Score (R^2):

- Shows the percentage of response variable variance explained by the model.
- Ranges from 0 to 1, with 1 being perfect prediction accuracy.

Model	Mean Squarred Error	Root Mean Squarred Error	Explained Variance Score	R-Sqaure Score(Accuracy)
Linear Regression	2.385772e+11	488443.669214	0.682814	0.682809
Support Vector Rregression	8.001844e+11	894530.277795	0.000096	-0.063855
Decision Tree Regressor	1.027511e+11	320548.196667	0.863434	0.863391
Random Forest Regressor	5.580073e+10	236221.780729	0.925818	0.925812

Model	Mean Squared Error	Root Mean Squared Error	Explained Variance Score	R-Square Score
Support Vector Regression	685,334,492,915.19	827,849.32	0.0001	-0.0636
Decision Tree Regressor	67,601,425,532.94	260,002.74	0.8951	0.8951
Random Forest Regressor	40,719,465,572.93	201,790.65	0.9368	0.9368
XGBoost	46,582,432,470.31	215,829.64	0.9277	0.9277

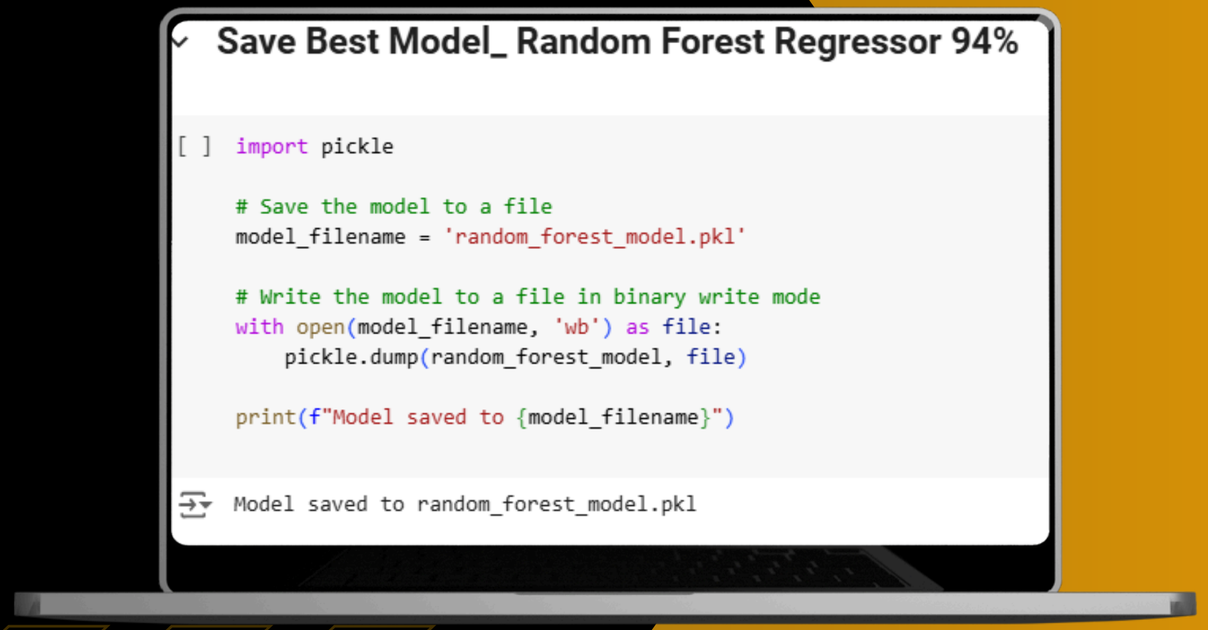

Training R-Square Score Comparison
Support Vector Regression
Decision Tree Regressor
Random Forest Regressor
XGBoost
1.0
0.8
0.6
0.4
0.2
0.0
Training R-Square Score
-6.77%
85.96%
92.82%
89.01%
Support Vector Regression
Decision Tree Regressor
Random Forest Regressor
XGBoost
Model


Model Comparison: Test data R² Accuracy
Support Vector Regression
Decision Tree Regressor
Random Forest Regressor
XGBoost
1.0
0.8
0.6
0.4
0.2
0.0
Test data R² Score
-6.36%
89.71%
93.60%
92.77%
Support Vector Regression
Decision Tree Regressor
Random Forest Regressor
XGBoost
Model

SELECTION OF BEST MODEL

RANDOM FOREST REGRESSOR > DECISION TREE REGRESSOR > XGBOOST > SVR

- **Random Forest Regressor:** Excels with the highest accuracy, boasting an R^2 of 0.9413 and the lowest error metrics, ideal for complex prediction tasks.
- **Decision Tree Regressor:** Offers good interpretability and solid performance with an R^2 of 0.8926, suitable for applications where model understanding is important.
- **XGBoost:** Highly effective, with an R^2 of 0.9277, providing a strong balance between precision and handling diverse data features.
- **Support Vector Regression (SVR):** Performs poorly on this dataset with a negative R^2, indicating it might require different settings or is unsuitable for the data type.



Prediction on train and test data

```
[ ] train_predictions = loaded_model.predict(X_train)
    train_predictions
```

```
array([1402716.66666667,  482450.        ,  769850.        , ...,
        372855.        ,  318297.61904762,  501630.        ])
```

```
[ ] # Make predictions
    test_predictions = loaded_model.predict(X_test)
    test_predictions
```

```
array([ 632928.45238095,  312615.23809524,  380030.        , ...,
        334020.        , 2504400.        , 6684150.        ])
```

```
] import pandas as pd

  input_data1 = pd.DataFrame(
      [
          [4, 20000, 18, 1490, 117.0, 5, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0]
      ],
      columns=[
          'vehicle_age', 'km_driven', 'mileage', 'engine', 'max_power', 'seats',
          'seller_type_Dealer', 'seller_type_Individual', 'seller_type_Trustmark Dealer',
          'fuel_type_CNG', 'fuel_type_Diesel', 'fuel_type_Electric', 'fuel_type_LPG', 'fuel_type_Petrol',
          'transmission_type_Automatic', 'transmission_type_Manual'
      ]
  )
```

```
] price = model.predict(input_data1)
  print("Predicted Price of used Car is: ", price)
```

```
Predicted Price of used Car is:  [1149860.]
```

THANK YOU

FOR YOUR ATTENTION