وزارة الاتصالات وتكنولوجيا المعلومات
رواد مصر الرقمية

DEPI PROJECT

IBM Data Science Track

Under the supervision to DR. Eman Raslan

SPECIAL THANKS

وزارة الاتصالات وتكنولوجيا المعلومات
رواد مصر الرقمية

We would like to extend our deepest gratitude to **DR. Eman Raslan** for her tireless dedication, expertise, and persistence.

Her outstanding explanations and invaluable mentorship have profoundly shaped our learning journey.

In particular, her guidance and advice during the **Depi Ibm Data Science Initiative** have been instrumental in deepening our understanding of complex concepts and advancing our skills. Her unwavering support has been key to our growth and success.

No words can truly describe her kindness and the support she has shown throughout this journey, for which we are endlessly grateful.

Team Members

وزارة الاتصالات وتكنولوجيا المعلومات
رواد مصر الرقمية

- Salma Ibrahim Nasri
- Ahmed Ezzat Rafallah Ibrahim
- Ahmed Mohamed Mohamed Elzaghel
- Ahmed Taha Mohamed
- Kerolos Ashraf Adeb Goda
- Mohamed Mohsen Abdeltawab

Machine Learning Life Cycle

1. Problem Definition

- Clearly define the problem you want to solve with machine learning.
- Identify the objective, whether it's classification, regression, clustering, etc.
- Determine the key performance metrics (e.g., accuracy, RMSE, precision) to evaluate success.

2. Data Collection

- Gather relevant data from various sources (databases, APIs, CSV files, web scraping).
- Ensure that the collected data contains the necessary features for solving the problem.
- In the case of a supervised learning task, make sure the target variable (label) is included.

3. Data Exploration & Analysis

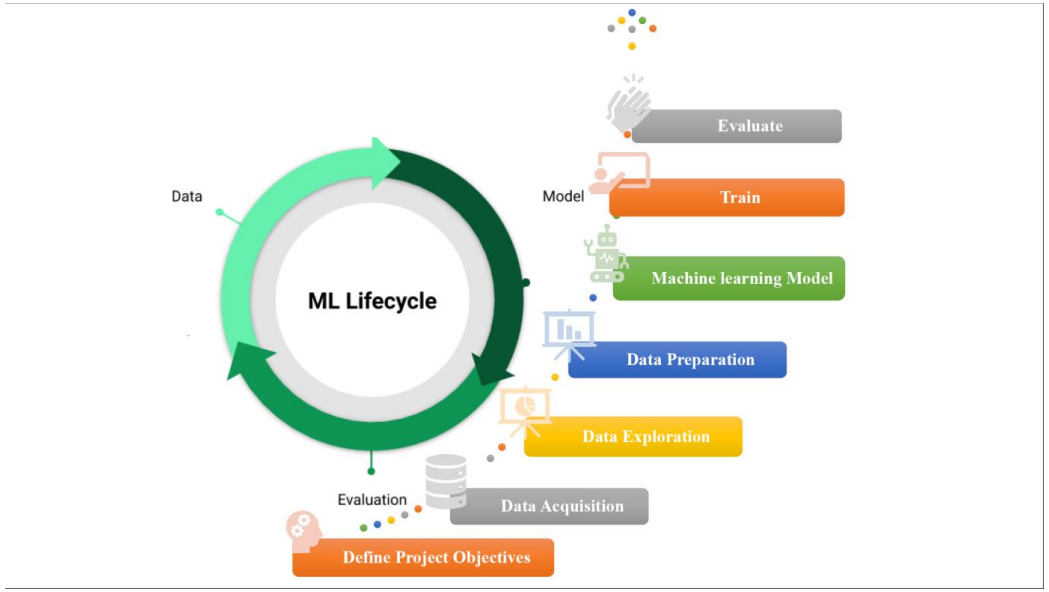
- Perform **Exploratory Data Analysis (EDA)** to understand the structure, patterns, and distributions in the data.
- Visualize relationships between features, outliers, and any potential correlations.
- Identify missing or erroneous data points.

4. Data Preprocessing

- **Data Cleaning**: Handle missing values, remove or impute outliers, correct errors.
- **Feature Engineering**: Create new features from existing ones (e.g., calculating the age of a car based on its year of manufacture).
- **Encoding**: Convert categorical data into numerical representations (e.g., one-hot encoding or label encoding).
- **Scaling**: Normalize or standardize numerical features to bring them into a comparable range.
- **Train-Test Split**: Split the dataset into training and test sets (e.g., 80% training, 20% testing) to evaluate model performance later.

5. Model Selection

- Choose an appropriate machine learning algorithm based on the problem type (e.g., Linear Regression, Decision Trees, Random Forest, XGBoost for regression tasks).
- Consider models that suit the dataset size, complexity, and structure.



Machine Learning Life Cycle

وزارة الاتصـــــالات
وتكنولوجيا المعلومات
رواد مصر الرقمية

6. Model Training

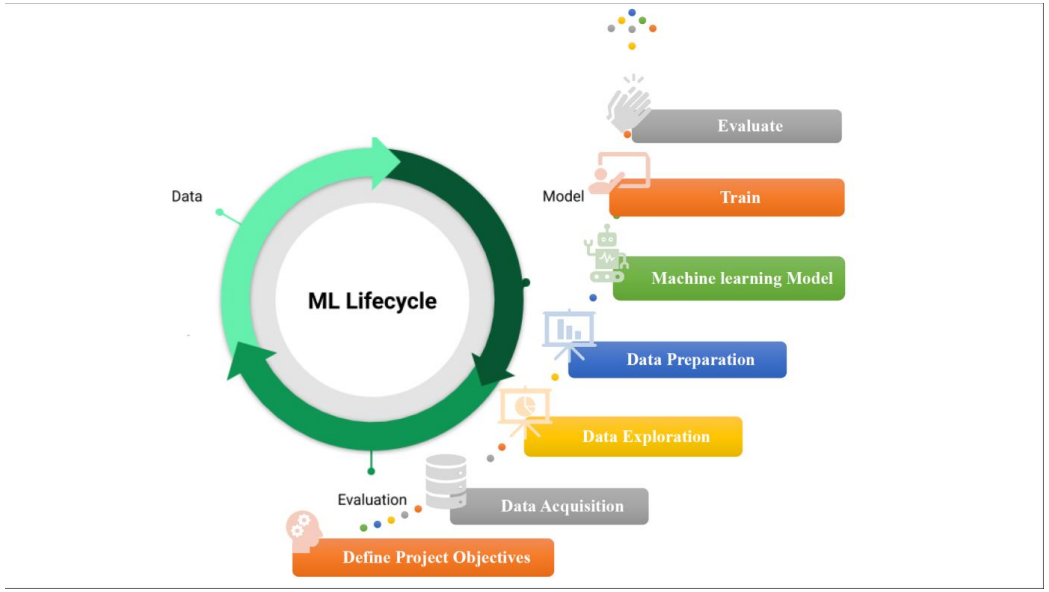
- Train the model using the training dataset.
- Use cross-validation techniques to ensure that the model generalizes well to unseen data.
- Tune hyperparameters during the training phase to optimize model performance.

7. Model Evaluation

- Evaluate the model's performance on the test dataset using performance metrics (e.g., RMSE for regression, accuracy for classification).
- Compare the model's predicted results against actual results.
- Generate evaluation reports including metrics such as precision, recall, F1 score, or confusion matrix (for classification) or mean squared error, R^2 (for regression).

8. Model Deployment

- Once the model performs well, deploy it into production.
- This could involve integrating the model into an application or exposing it via an API.
- Monitor the model's performance over time to ensure it continues to perform well in real-world scenarios.



Dataset Description

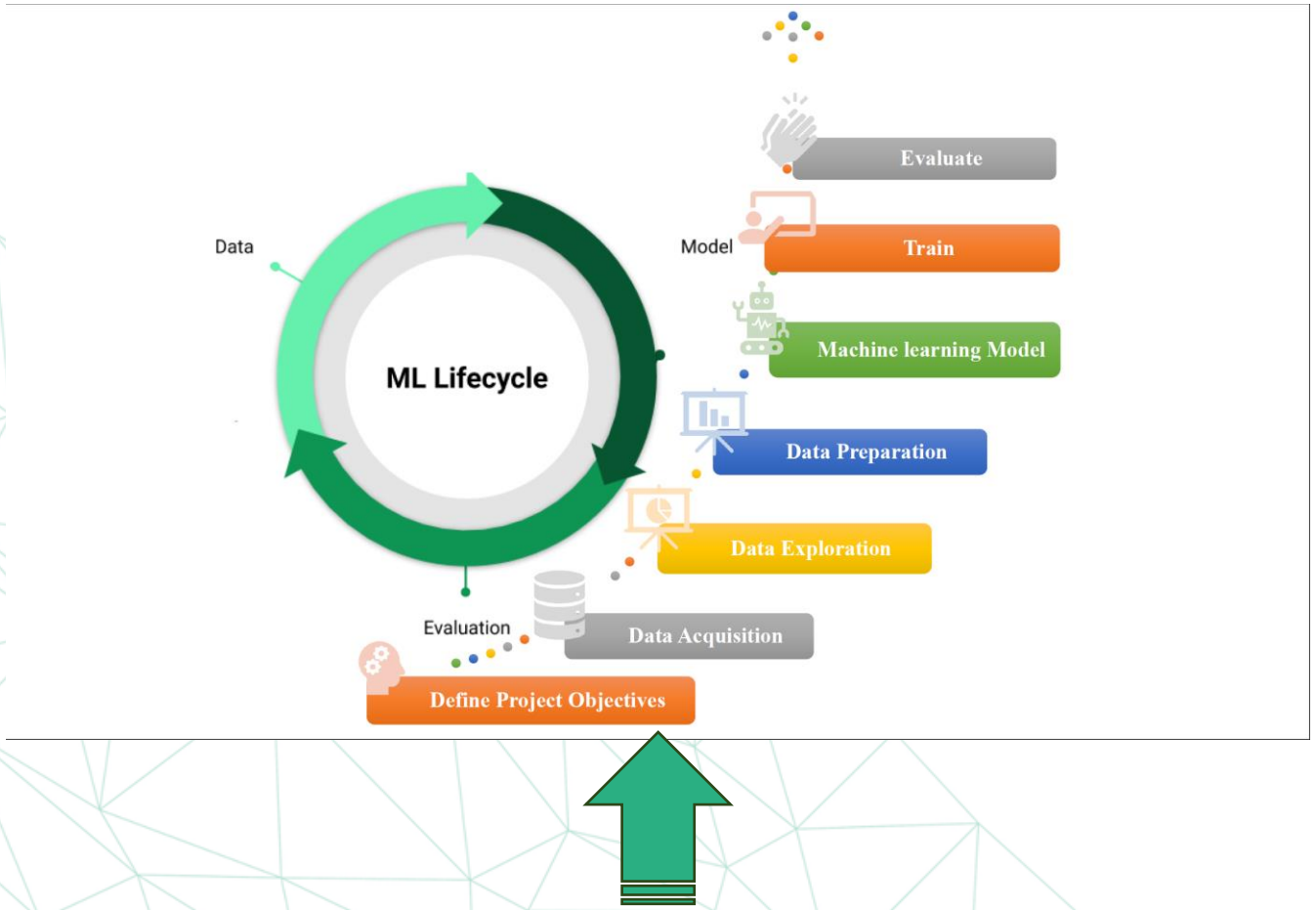
This dataset contains detailed information on 6,019 used cars, providing insights into various features such as model specifications, location, ownership history, and pricing.

It can be used to analyze factors influencing the prices of second-hand cars in different cities across India. Here's a breakdown of the dataset:

- **Total Rows:** 6,019
- **Total Columns:** 14

Column Descriptions:

1. **Unnamed: 0** - Index column (irrelevant for analysis).
2. **Name** - The name of the car, including make, model, and variant.
3. **Location** - The city where the car is being sold [Example: *Mumbai*, *Pune*, *Chennai*.]
4. **Year** - The manufacturing year of the car.
5. **Kilometers_Driven** - The total distance the car has been driven (in kilometers).
6. **Fuel_Type** - The type of fuel the car uses. [*CNG, Diesel, Petrol*.]
7. **Transmission** - The type of transmission in the car. [*Manual, Automatic*.]
8. **Owner_Type** - The number of previous owners of the car. [*First, Second,…]*
9. **Mileage** - The fuel efficiency of the car (in kmpl or km/kg for CNG vehicles).
10. **Engine** - The engine capacity of the car, measured in cubic centimeters (CC).
11. **Power** - The power output of the car, measured in brake horsepower (bhp).
12. **Seats** - The number of seats in the car.
13. **New_Price** - The price of the car when it was new (in INR).
14. **Price** - The current selling price of the car (in INR lakhs).



1

THE PROJECT OBJECTIVE

1. The Project Objective

In this project, we aim to build a machine learning model that uses historical data of used cars, including technical specifications and market trends, to predict the resale price.

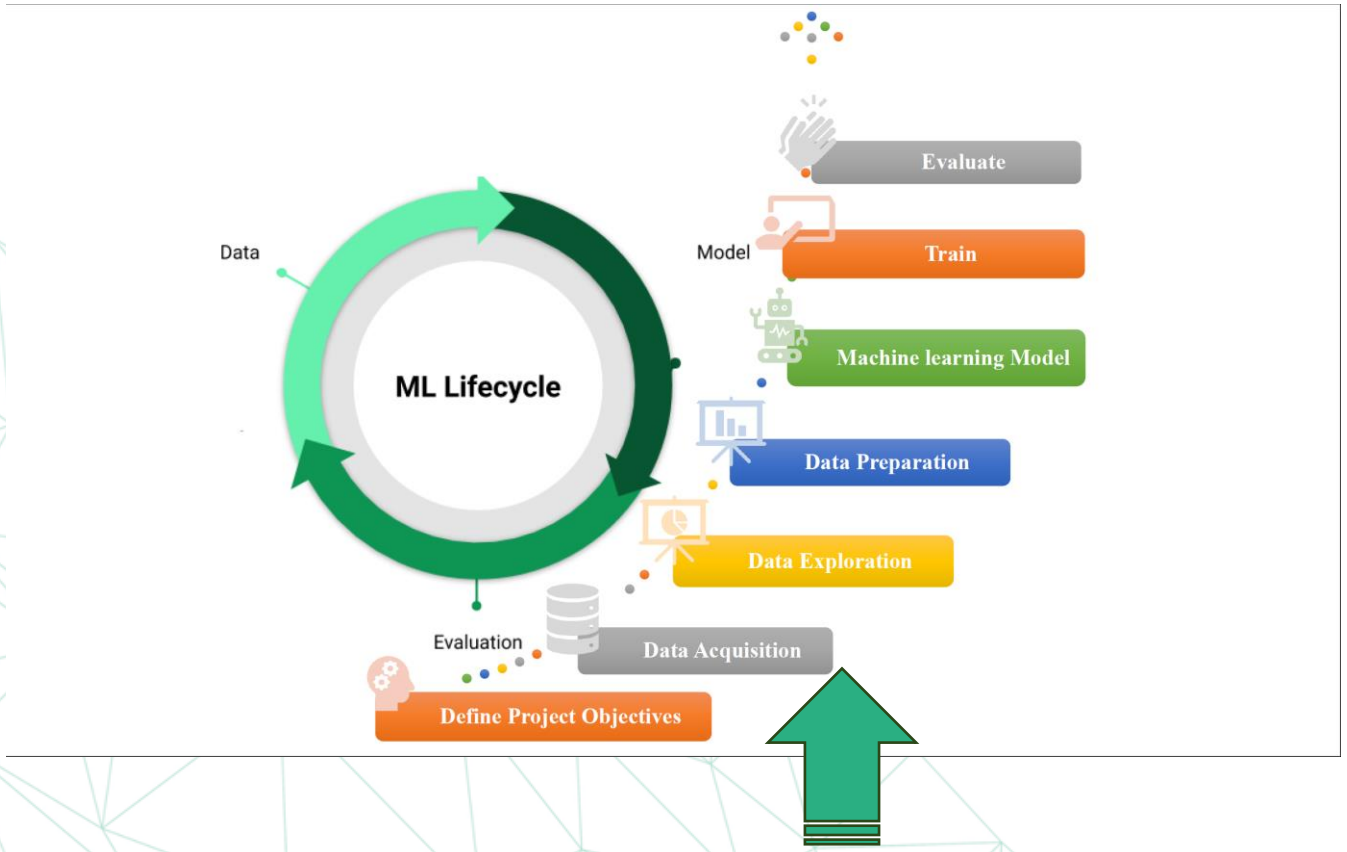
In the rapidly growing second-hand car market, accurately predicting the resale price of used vehicles is crucial for both sellers and buyers.

Various factors such as the car's age, kilometers driven, fuel type, transmission, engine capacity, and ownership history play a significant role in determining its value.

By leveraging these features, we can develop a robust predictive model to estimate car prices, helping individuals make more informed decisions.

This model will not only provide insights into key factors influencing pricing but also assist stakeholders in setting competitive prices or finding the best deals in the market.

Through feature analysis, preprocessing, and the application of regression techniques, we aim to create an efficient and accurate pricing tool for the second-hand car industry.

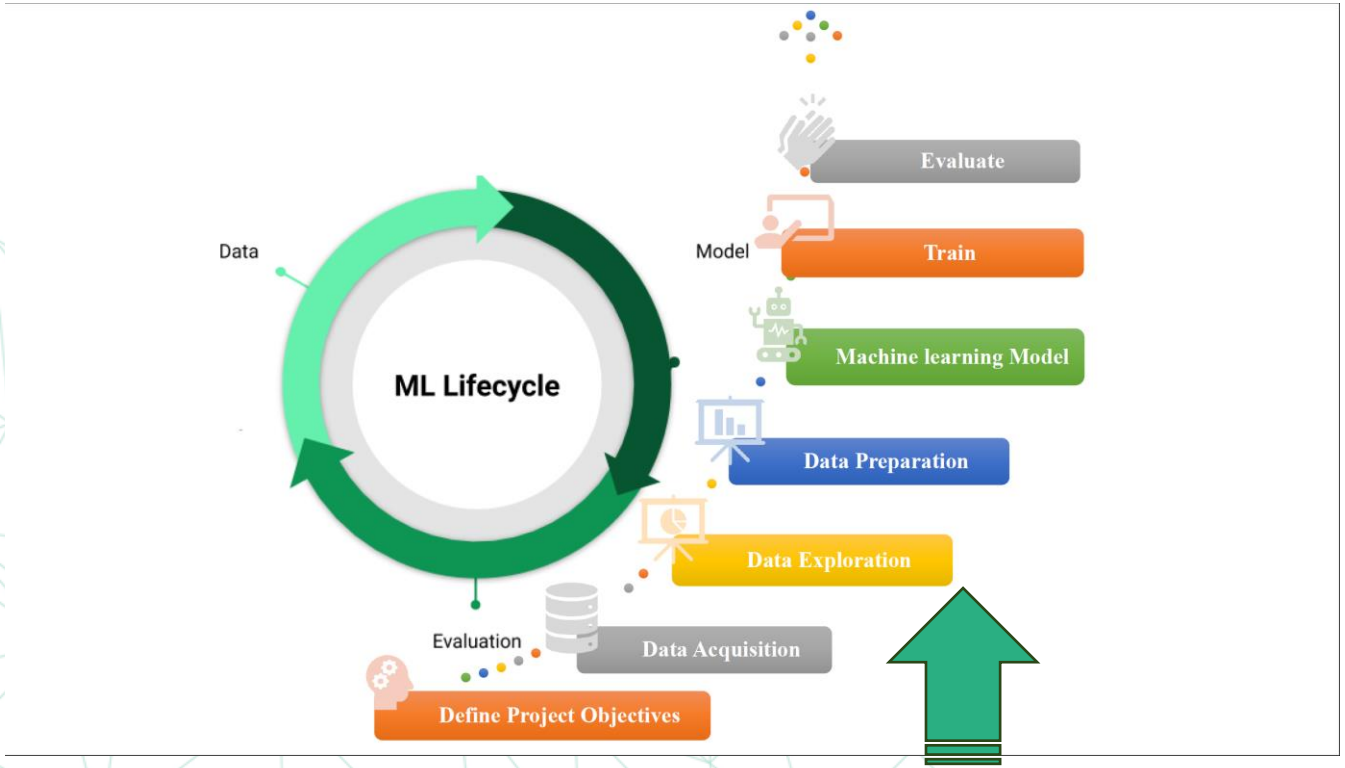


2

Data Collection

2. Data Collection

- The data set are given and if there more features must be collect will be collected



3

Data Exploration and Cleaning

A) Check For Data Quality

B) feature selection for exploration

C) Data Exploration ,visualization ,insights

3. Data Exploration and Cleaning

وزارة الاتصـــــالات وتكنولوجيا المعلومات
رواد مصر الرقمية

```
#import the important libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

✓ 0.0s

```
#load the data set
original_data=pd.read_csv("used-car-dataset.csv")
```

✓ 0.0s

```
#explore the data
original_data
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6014	6014	Maruti Swift VDI	Delhi	2014	27365	Diesel	Manual	First	28.4 kmpl	1248 CC	74 bhp	5.0	7.88 Lakh	4.75
6015	6015	Hyundai Xcent 1.1 CRDi S	Jaipur	2015	100000	Diesel	Manual	First	24.4 kmpl	1120 CC	71 bhp	5.0	NaN	4.00
6016	6016	Mahindra Xylo D4 BSIV	Jaipur	2012	55000	Diesel	Manual	Second	14.0 kmpl	2498 CC	112 bhp	8.0	NaN	2.90
6017	6017	Maruti Wagon R VXI	Kolkata	2013	46000	Petrol	Manual	First	18.9 kmpl	998 CC	67.1 bhp	5.0	NaN	2.65
6018	6018	Chevrolet Beat Diesel	Hyderabad	2011	47000	Diesel	Manual	First	25.44 kmpl	936 CC	57.6 bhp	5.0	NaN	2.50

6019 rows × 14 columns

A) Check For Data Quality

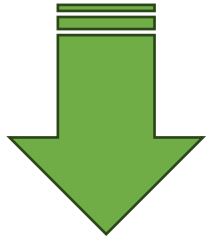
1) Check For Completeness : This checks whether there are any missing values in the dataset.

```
#check for completness
for i in original_data.columns:
    null_percent=original_data[i].isnull().sum()/len(original_data[i])
    print(f"{i} has {null_percent} null info")
    print("-------------")
```

```
Unnamed: 0 has 0.0 null info
-------------
Name has 0.0 null info
-------------
Location has 0.0 null info
-------------
Year has 0.0 null info
-------------
Kilometers_Driven has 0.0 null info
-------------
Fuel_Type has 0.0 null info
-------------
Transmission has 0.0 null info
-------------
Owner_Type has 0.0 null info
-------------
Mileage has 0.0003322811098189068 null info
-------------
Engine has 0.005981059976740323 null info
-------------
Power has 0.005981059976740323 null info
-------------
Seats has 0.006977903306197043 null info
-------------
New_Price has 0.8631001827546104 null info
-------------
Price has 0.0 null info
-------------
```

Comment

New_Price has 0.8631001827546104 null info so it must be eliminated an Unnamed columns has no meaning



Solution

```
# drop the nan values beacuse it can't be imputated and it will  be misleading
original_data.drop(["Unnamed: 0","New_Price"],axis=1,inplace=True)
original_data.dropna(inplace=True)
```

2. Consistency : This checks whether data follows a consistent format across the dataset.

```
#explore the data
original_data
```

	Unnamed: 0	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	New_Price	Price
0	0	Maruti Wagon R LXI CNG	Mumbai	2010	72000	CNG	Manual	First	26.6 km/kg	998 CC	58.16 bhp	5.0	NaN	1.75
1	1	Hyundai Creta 1.6 CRDi SX Option	Pune	2015	41000	Diesel	Manual	First	19.67 kmpl	1582 CC	126.2 bhp	5.0	NaN	12.50
2	2	Honda Jazz V	Chennai	2011	46000	Petrol	Manual	First	18.2 kmpl	1199 CC	88.7 bhp	5.0	8.61 Lakh	4.50
3	3	Maruti Ertiga VDI	Chennai	2012	87000	Diesel	Manual	First	20.77 kmpl	1248 CC	88.76 bhp	7.0	NaN	6.00
4	4	Audi A4 New 2.0 TDI Multitronic	Coimbatore	2013	40670	Diesel	Automatic	Second	15.2 kmpl	1968 CC	140.8 bhp	5.0	NaN	17.74
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6014	6014	Maruti Swift VDI	Delhi	2014	27365	Diesel	Manual	First	28.4 kmpl	1248 CC	74 bhp	5.0	7.88 Lakh	4.75
6015	6015	Hyundai Xcent 1.1 CRDi S	Jaipur	2015	100000	Diesel	Manual	First	24.4 kmpl	1120 CC	71 bhp	5.0	NaN	4.00
6016	6016	Mahindra Xylo D4 BSIV	Jaipur	2012	55000	Diesel	Manual	Second	14.0 kmpl	2498 CC	112 bhp	8.0	NaN	2.90
6017	6017	Maruti Wagon R VXI	Kolkata	2013	46000	Petrol	Manual	First	18.9 kmpl	998 CC	67.1 bhp	5.0	NaN	2.65
6018	6018	Chevrolet Beat Diesel	Hyderabad	2011	47000	Diesel	Manual	First	25.44 kmpl	936 CC	57.6 bhp	5.0	NaN	2.50

6019 rows × 14 columns

Comment

- Mileage has 2 units for fuel consumption according to the vehicle From domain knowledge km/kg is for CNG fuel type so must be converted to kmpl / Mileage (kmpl)=Mileage (km/kg)×1.39
- All object values must be uniform on lower case letter like [Name,Location,Fuel_Type]
- Categorical columns and must be changed to numerical columns [Owner_Type,Mileage,Engine,Power]

2. Consistency : This checks whether data follows a consistent format across the dataset.

Solution

```
# From domain knowledge km/kg is for cng fuel type so must be converted to kmpl / Mileage (kmpl)=Mileage (km/kg)×1.39**
def kmpl_converter(x):
    if "km/kg" in x:
        number=float(x.split()[0])*1.39
        return number
    elif "kmpl" in x :
        number=float(x.split()[0])
        return number
```

✓ 0.0s

```
original_data["kmpl"]=original_data["Mileage"].apply(kmpl_converter)
```

✓ 0.0s

```
# Check for consistency for other columns
original_data.info()
```

✓ 0.1s

```
<class 'pandas.core.frame.DataFrame'>
Index: 5975 entries, 0 to 6018
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               5975 non-null   object
 1   Location           5975 non-null   object
 2   Year               5975 non-null   int64
 3   Kilometers_Driven  5975 non-null   int64
 4   Fuel_Type          5975 non-null   object
 5   Transmission       5975 non-null   object
 6   Owner_Type         5975 non-null   object
 7   Mileage            5975 non-null   object
 8   Engine             5975 non-null   object
 9   Power              5975 non-null   object
 10  Seats              5975 non-null   float64
 11  Price              5975 non-null   float64
 12  kmpl               5975 non-null   float64
dtypes: float64(3), int64(2), object(8)
```

Solution

```
# Convert every word to lower levels to make it easy to manapulate
original_data["Price_Lakh"]=original_data["Price"]
original_data["Name"]=original_data["Name"].str.lower()
original_data["Location"]=original_data["Location"].str.lower()
original_data["Fuel_Type"]=original_data["Fuel_Type"].str.lower()
original_data["Transmission"]=original_data["Transmission"].str.lower()
original_data["Owner_Type"]=original_data["Owner_Type"].str.lower()
```

✓ 0.0s

2. Consistency : This checks whether data follows a consistent format across the dataset.
>> Feature Engineering (Domain Knowledge)

```
original_data
```

✓ 0.0s

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	kmpl	Price_Lakh	Brand
0	maruti wagon r lxi cng	mumbai	2010	72000	cng	manual	first	26.6 km/kg	998 CC	58.16 bhp	5.0	1.75	36.974	1.75	maruti
1	hyundai creta 1.6 crdi sx option	pune	2015	41000	diesel	manual	first	19.67 kmpl	1582 CC	126.2 bhp	5.0	12.50	19.670	12.50	hyundai
2	honda jazz v	chennai	2011	46000	petrol	manual	first	18.2 kmpl	1199 CC	88.7 bhp	5.0	4.50	18.200	4.50	honda
3	maruti ertiga vdi	chennai	2012	87000	diesel	manual	first	20.77 kmpl	1248 CC	88.76 bhp	7.0	6.00	20.770	6.00	maruti
4	audi a4 new 2.0 tdi multitronic	coimbatore	2013	40670	diesel	automatic	second	15.2 kmpl	1968 CC	140.8 bhp	5.0	17.74	15.200	17.74	audi
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6014	maruti swift vdi	delhi	2014	27365	diesel	manual	first	28.4 kmpl	1248 CC	74 bhp	5.0	4.75	28.400	4.75	maruti
6015	hyundai xcent 1.1 crdi s	jaipur	2015	100000	diesel	manual	first	24.4 kmpl	1120 CC	71 bhp	5.0	4.00	24.400	4.00	hyundai
6016	mahindra xylo d4 bsiv	jaipur	2012	55000	diesel	manual	second	14.0 kmpl	2498 CC	112 bhp	8.0	2.90	14.000	2.90	mahindra
6017	maruti wagon r vxi	kolkata	2013	46000	petrol	manual	first	18.9 kmpl	998 CC	67.1 bhp	5.0	2.65	18.900	2.65	maruti
6018	chevrolet beat diesel	hyderabad	2011	47000	diesel	manual	first	25.44 kmpl	936 CC	57.6 bhp	5.0	2.50	25.440	2.50	chevrolet

5975 rows × 15 columns

Comment

Name Column has the brand name and the model and the level of the vehicle

2. Consistency : This checks whether data follows a consistent format across the dataset.
>> Feature Engineering (Domain Knowledge)

```
original_data["Name"].unique()
```
✓ 0.0s

```
array(['maruti wagon r lxi cng', 'hyundai creta 1.6 crdi sx option',
       'honda jazz v', ..., 'volkswagen polo ipl ii 1.2 petrol highline',
       'tata bolt revotron xt', 'mahindra xylo d4 bsiv'], dtype=object)
```

Comment

Hyundai>> is the brand name
Creta>> is the model name
1.6>> is considering unique selling point
Sx>> name of the level
Option>> general word

Solution

Feature Engineering according to the brand knowledge > the model information from the name column

```
#identify the length of words in each line
original_data["lenght_of_names"]=original_data["Name"].apply(lambda x: len(x.split()))
```
✓ 0.0s

```
def extract_brand_model(car_name):
    # Split the string and take the most words that can describe the model of the vehicle
    words = car_name.split()
    if len(words) == 3:
        return words[1]
    elif len(words) > 3:
        return ' '.join(words[1:3])
    else:
        return car_name
```
✓ 0.0s

```
# Feature enginnering the Brand name and the model and unique selling point
original_data["Model"]=original_data["Name"].apply(extract_brand_model)
original_data["Brand"]=original_data["Name"].str.split(expand=True)[0].str.lower()
```
✓ 0.0s

	Name
0	maruti wagon r lxi cng
1	hyundai creta 1.6 crdi sx option
2	honda jazz v
3	maruti ertiga vdi
4	audi a4 new 2.0 tdi multitro
...	...
6014	maruti swift vdi
6015	hyundai xcent 1.1 crdi s
6016	mahindra xylo d4 bsiv
6017	maruti wagon r vxi
6018	chevrolet beat diesel

5975 rows × 17 columns

lenght_of_names	Model	Brand
5	wagon r	maruti
6	creta 1.6	hyundai
3	jazz	honda
3	ertiga	maruti
6	a4 new	audi
...	...	...
3	swift	maruti
5	xcent 1.1	hyundai
4	xylo d4	mahindra
4	wagon r	maruti
3	beat	chevrolet

2. Consistency : This checks whether data follows a consistent format across the dataset.
>> Feature Engineering (Domain Knowledge)

```python
# check for constency in other columns
for i in original_data.select_dtypes("object"):
    print(i)
    print(original_data[i].unique())
    print("------------------------------------")
```

✓ 0.0s

```
Name
['maruti wagon r lxi cng' 'hyundai creta 1.6 crdi sx option'
 'honda jazz v' ... 'volkswagen polo ipl ii 1.2 petrol highline'
 'tata bolt revotron xt' 'mahindra xylo d4 bsiv']
------------------------------------
Location
['mumbai' 'pune' 'chennai' 'coimbatore' 'hyderabad' 'jaipur' 'kochi'
 'kolkata' 'delhi' 'bangalore' 'ahmedabad']
------------------------------------
Fuel_Type
['cng' 'diesel' 'petrol' 'lpg']
------------------------------------
Transmission
['manual' 'automatic']
------------------------------------
Owner_Type
['first' 'second' 'fourth & above' 'third']
------------------------------------
```

```
------------------------------------
Brand
['maruti' 'hyundai' 'honda' 'audi' 'nissan' 'toyota' 'volkswagen' 'tata'
 'land' 'mitsubishi' 'renault' 'mercedes-benz' 'bmw' 'mahindra' 'ford'
 'porsche' 'datsun' 'jaguar' 'volvo' 'chevrolet' 'skoda' 'mini' 'fiat'
 'jeep' 'smart' 'ambassador' 'isuzu' 'force' 'bentley' 'lamborghini']
------------------------------------
```

```
Model
['wagon r' 'creta 1.6' 'jazz' 'ertiga' 'a4 new' 'eon lpg' 'micra diesel'
 'innova crysta' 'vento diesel' 'indica vista' 'ciaz' 'city 1.5'
 'swift vdi' 'rover range' 'rover freelander' 'pajero sport' 'amaze s'
 'swift ddis' 'duster 85ps' 'new c-class' '3 series' 's cross'
 'a6 2011-2015' 'i20 1.2' 'vento petrol' 'city corporate' 'alto k10'
 'wrv i-vtec' 'innova 2.5' 'duster 110ps' 'corolla altis'
 'ssangyong rexton' 'a6 2.7' 'vento 1.6' 'vitara brezza' 'kuv 100'
 'm-class ml' 'polo diesel' 'alto' 'nano lx' 'i20 magna' 'elantra 2.0'
 'elantra' 'xcent 1.1' 'thar crde' 'a4 2.0' 'swift' 'swift ldi'
 'grand i10' 'vento 2013-2015' 'kwid' 'i10' 'x-trail slx' 'zen estilo'
 'figo diesel' 'indica v2' 'city zx' 'c-class progressive' 'ertiga shvs'
 'creta 1.4' 'cayenne 2009-2014' 'xuv500 w8' 'i10 sportz' 'terrano xv'
```

2. Consistency : This checks whether data follows a consistent format across the dataset.
>> Feature Engineering (Domain Knowledge)

Power

```
['58.16 bhp' '126.2 bhp' '88.7 bhp' '88.76 bhp' '140.8 bhp' '55.2 bhp'
 '63.1 bhp' '171.5 bhp' '103.6 bhp' '74 bhp' '103.25 bhp' '116.3 bhp'
 '187.7 bhp' '115 bhp' '175.56 bhp' '98.6 bhp' '83.8 bhp' '167.62 bhp'
 '190 bhp' '88.5 bhp' '177.01 bhp' '80 bhp' '67.1 bhp' '102 bhp'
 '108.45 bhp' '138.1 bhp' '184 bhp' '179.5 bhp' '103.5 bhp' '64 bhp'
 '82 bhp' '254.8 bhp' '73.9 bhp' '46.3 bhp' '37.5 bhp' '77 bhp' '82.9 bhp'
 '149.92 bhp' '138.03 bhp' '112.2 bhp' '163.7 bhp' '71 bhp' '105 bhp'
 '174.33 bhp' '75 bhp' '103.2 bhp' '53.3 bhp' '78.9 bhp' '147.6 bhp'
 '147.8 bhp' '68 bhp' '186 bhp' '170 bhp' '69 bhp' '140 bhp' '78 bhp'
 '194 bhp' '500 bhp' '108.5 bhp' '86.8 bhp' '187.74 bhp' 'null bhp'
 '132 bhp' '86.7 bhp' '73.94 bhp' '117.3 bhp' '218 bhp' '168.5 bhp'
 '89.84 bhp' '110 bhp' '90 bhp' '82.85 bhp' '67 bhp' '241.4 bhp' '35 bhp'
 '270.9 bhp' '126.32 bhp' '73 bhp' '130 bhp' '100.6 bhp' '150 bhp'
 '75.94 bhp' '215 bhp' '107.3 bhp' '37.48 bhp' '120 bhp' '178 bhp'
 '152 bhp' '91.1 bhp' '85.80 bhp' '362.07 bhp' '121.3 bhp' '143 bhp'
 '81.80 bhp' '171 bhp' '76.8 bhp' '103.52 bhp' '444 bhp' '362.9 bhp'
 '67.06 bhp' '120.7 bhp' '258 bhp' '81.86 bhp' '112 bhp' '88.73 bhp'
 '57.6 bhp' '157.75 bhp' '102.5 bhp' '201.1 bhp' '83.1 bhp' '68.05 bhp'
 '88.50 bhp' '106 bhp' '100 bhp' '81.83 bhp' '85 bhp' '64.1 bhp'
 '177.5 bhp' '246.7 bhp' '177.46 bhp' '65 bhp' '67.04 bhp' '189.08 bhp'
 '53.5 bhp' '194.3 bhp' '70 bhp' '183 bhp' '254.79 bhp' '66.1 bhp'
 '76 bhp' '60 bhp' '123.24 bhp' '47.3 bhp' '118 bhp' '88.8 bhp' '177 bhp'
 '136 bhp' '201.15 bhp' '93.7 bhp' '177.6 bhp' '313 bhp' '245 bhp'
 '125 bhp' '141 bhp' '227 bhp' '62 bhp' '141.1 bhp' '83.14 bhp' '192 bhp'
 '67.05 bhp' '47 bhp' '235 bhp' '37 bhp' '87.2 bhp' '203 bhp' '204 bhp'
 '246.74 bhp' '122 bhp' '282 bhp' '181 bhp' '224 bhp' '94 bhp' '367 bhp'
```

Engine

```
['998 CC' '1582 CC' '1199 CC' '1248 CC' '1968 CC' '814 CC' '1461 CC'
 '2755 CC' '1598 CC' '1462 CC' '1497 CC' '2179 CC' '2477 CC' '1498 CC'
 '2143 CC' '1995 CC' '1984 CC' '1197 CC' '2494 CC' '1798 CC' '2696 CC'
 '2698 CC' '1061 CC' '1198 CC' '2987 CC' '796 CC' '624 CC' '1999 CC'
 '1991 CC' '2694 CC' '1120 CC' '2498 CC' '799 CC' '2393 CC' '1399 CC'
 '1796 CC' '2148 CC' '1396 CC' '1950 CC' '4806 CC' '1998 CC' '1086 CC'
 '1193 CC' '2982 CC' '1493 CC' '2967 CC' '2993 CC' '1196 CC' '1799 CC'
 '2497 CC' '2354 CC' '1373 CC' '2996 CC' '1591 CC' '2894 CC' '5461 CC'
 '1595 CC' '936 CC' '1997 CC' '1896 CC' '1390 CC' '1364 CC' '2199 CC'
 '993 CC' '999 CC' '1405 CC' '2956 CC' '1794 CC' '995 CC' '2496 CC'
 '1599 CC' '2400 CC' '1495 CC' '2523 CC' '793 CC' '4134 CC' '1596 CC'
 '1395 CC' '2953 CC' '1586 CC' '2362 CC' '1496 CC' '1368 CC' '1298 CC'
 '1956 CC' '1299 CC' '3498 CC' '2835 CC' '1150 CC' '3198 CC' '1343 CC'
 '1499 CC' '1186 CC' '1590 CC' '2609 CC' '2499 CC' '2446 CC' '1978 CC'
 '2360 CC' '3436 CC' '2198 CC' '4367 CC' '2706 CC' '1422 CC' '2979 CC'
 '1969 CC' '1489 CC' '2489 CC' '1242 CC' '1388 CC' '1172 CC' '2495 CC'
 '1194 CC' '3200 CC' '1781 CC' '1341 CC' '2773 CC' '3597 CC' '1985 CC'
 '2147 CC' '1047 CC' '2999 CC' '2995 CC' '2997 CC' '1948 CC' '2359 CC'
 '4395 CC' '2349 CC' '2720 CC' '1468 CC' '3197 CC' '2487 CC' '1597 CC'
 '2771 CC' '4951 CC' '970 CC' '2925 CC' '2200 CC' '5000 CC' '2149 CC'
 '5998 CC' '2092 CC' '5204 CC' '2112 CC' '1797 CC']
```

Comment

- **Power column have 2 problem one is must be converted to float and have a null values**
- **Engine column must be in float format and remove CC**

2. Consistency : This checks whether data follows a consistent format across the dataset.
>> Feature Engineering (Domain Knowledge)

Solution

```
#transform all the numerical numbers to their suitable format
original_data["CC"]=original_data["Engine"].str.split(expand=True)[0].astype("float")
original_data["Horse_Power"]=original_data["Power"].str.split(expand=True)[0]
original_data["Horse_Power"]=original_data["Horse_Power"].replace("null",np.nan)
original_data["Horse_Power"]=original_data["Horse_Power"].astype("float")
```

✓ 0.0s

```
# the horspower feature has a more than 100 null values so it should be imputated with domain knowledge to imputate horsepower of
# of engine you must know the cc and fuel type and transmission
original_data["Horse_Power"]=original_data["Horse_Power"].fillna(original_data["Horse_Power"].median())
```

✓ 0.0s

Comment

- **All columns are clean and in their right format**

```
original_data.info()
```

[98] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 5975 entries, 0 to 6018
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               5975 non-null   object
 1   Location           5975 non-null   object
 2   Year               5975 non-null   int64
 3   Kilometers_Driven  5975 non-null   int64
 4   Fuel_Type          5975 non-null   object
 5   Transmission       5975 non-null   object
 6   Owner_Type         5975 non-null   object
 7   Mileage            5975 non-null   object
 8   Engine             5975 non-null   object
 9   Power              5975 non-null   object
 10  Seats              5975 non-null   float64
 11  Price              5975 non-null   float64
 12  kmpl               5975 non-null   float64
 13  Price_Lakh         5975 non-null   float64
 14  lenght_of_names    5975 non-null   int64
 15  Model              5975 non-null   object
 16  Brand              5975 non-null   object
 17  CC                 5975 non-null   float64
 18  Horse_Power        5975 non-null   float64
dtypes: float64(6), int64(3), object(10)
memory usage: 933.6+ KB
```

3. Accuracy: This metric checks if the data reflects the real-world values it is supposed to

4. Validity: This checks whether the data falls within accepted ranges or conforms to defined rules.

```
# Data validation
original_data.describe(include="all")
```

✓ 0.1s

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	kmpl	Price_Lakh	lenght_of_names	Model	Brand	CC	Horse_Power
count	5975	5975	5975.000000	5.975000e+03	5975	5975	5975	5975	5975	5975	5975.000000	5975.000000	5975.000000	5975.000000	5975.000000	5975	5975	5975.000000	5975.000000
unique	1831	11	NaN	NaN	4	2	4	442	145	371	NaN	NaN	NaN	NaN	NaN	646	30	NaN	NaN
top	maruti swift dzire vdi	mumbai	NaN	NaN	diesel	manual	first	18.9 kmpl	1197 CC	74 bhp	NaN	NaN	NaN	NaN	NaN	wagon r	maruti	NaN	NaN
freq	50	784	NaN	NaN	3195	4266	4903	172	606	235	NaN	NaN	NaN	NaN	NaN	153	1197	NaN	NaN
mean	NaN	NaN	2013.386778	5.867431e+04	NaN	NaN	NaN	NaN	NaN	NaN	5.278828	9.501647	18.284969	9.501647	4.840167	NaN	NaN	1621.606695	113.008372
std	NaN	NaN	3.247238	9.155851e+04	NaN	NaN	NaN	NaN	NaN	NaN	0.808959	11.205736	4.813770	11.205736	1.248659	NaN	NaN	601.036987	53.453844
min	NaN	NaN	1998.000000	1.710000e+02	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.440000	0.000000	0.440000	3.000000	NaN	NaN	624.000000	34.200000
25%	NaN	NaN	2012.000000	3.390800e+04	NaN	NaN	NaN	NaN	NaN	NaN	5.000000	3.500000	15.260000	3.500000	4.000000	NaN	NaN	1198.000000	77.000000
50%	NaN	NaN	2014.000000	5.300000e+04	NaN	NaN	NaN	NaN	NaN	NaN	5.000000	5.650000	18.200000	5.650000	5.000000	NaN	NaN	1493.000000	97.700000
75%	NaN	NaN	2016.000000	7.300000e+04	NaN	NaN	NaN	NaN	NaN	NaN	5.000000	9.950000	21.100000	9.950000	6.000000	NaN	NaN	1984.000000	138.100000
max	NaN	NaN	2019.000000	6.500000e+06	NaN	NaN	NaN	NaN	NaN	NaN	10.000000	160.000000	46.620600	160.000000	10.000000	NaN	NaN	5998.000000	560.000000

5. Timeliness: This metric evaluates whether the data is up-to-date or still relevant to the current context

6. Uniqueness: This checks for duplicate entries, ensuring each observation is unique.

```
# check for duplication
original_data[original_data.duplicated()]
```

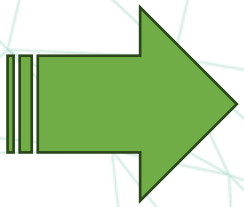
✓ 0.0s Python

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	kmpl	Price_Lakh	lenght_of_names	Model	Brand	CC	Horse_Power
4781	maruti alto lxi	hyderabad	2007	52195	petrol	manual	first	19.7 kmpl	796 CC	46.3 bhp	5.0	1.75	19.7	1.75	3	alto	maruti	796.0	46.3

Solution

```
original_data.drop_duplicates(inplace=True)
```

✓ 0.0s



```
original_data
```

✓ 0.1s Python

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Seats	Price	kmpl	Price_Lakh	lenght_of_names	Model	Brand	CC	Horse_Power
0	maruti wagon r lxi cng	mumbai	2010	72000	cng	manual	first	26.6 km/kg	998 CC	58.16 bhp	5.0	1.75	36.974	1.75	5	wagon r	maruti	998.0	58.16
1	hyundai creta 1.6 crdi sx option	pune	2015	41000	diesel	manual	first	19.67 kmpl	1582 CC	126.2 bhp	5.0	12.50	19.670	12.50	6	creta 1.6	hyundai	1582.0	126.20
2	honda jazz v	chennai	2011	46000	petrol	manual	first	18.2 kmpl	1199 CC	88.7 bhp	5.0	4.50	18.200	4.50	3	jazz	honda	1199.0	88.70
3	maruti ertiga vdi	chennai	2012	87000	diesel	manual	first	20.77 kmpl	1248 CC	88.76 bhp	7.0	6.00	20.770	6.00	3	ertiga	maruti	1248.0	88.76
4	audi a4 new 2.0 tdi multitronic	coimbatore	2013	40670	diesel	automatic	second	15.2 kmpl	1968 CC	140.8 bhp	5.0	17.74	15.200	17.74	6	a4 new	audi	1968.0	140.80
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6014	maruti swift vdi	delhi	2014	27365	diesel	manual	first	28.4 kmpl	1248 CC	74 bhp	5.0	4.75	28.400	4.75	3	swift	maruti	1248.0	74.00
6015	hyundai xcent 1.1 crdi s	jaipur	2015	100000	diesel	manual	first	24.4 kmpl	1120 CC	71 bhp	5.0	4.00	24.400	4.00	5	xcent 1.1	hyundai	1120.0	71.00
6016	mahindra xylo d4 bsiv	jaipur	2012	55000	diesel	manual	second	14.0 kmpl	2498 CC	112 bhp	8.0	2.90	14.000	2.90	4	xylo d4	mahindra	2498.0	112.00
6017	maruti wagon r vxi	kolkata	2013	46000	petrol	manual	first	18.9 kmpl	998 CC	67.1 bhp	5.0	2.65	18.900	2.65	4	wagon r	maruti	998.0	67.10
6018	chevrolet beat diesel	hyderabad	2011	47000	diesel	manual	first	25.44 kmpl	936 CC	57.6 bhp	5.0	2.50	25.440	2.50	3	beat	chevrolet	936.0	57.60

5974 rows × 19 columns

B) feature selection for exploration

```
original_data.columns
```

✓ 0.0s

```
Index(['Name', 'Location', 'Year', 'Kilometers_Driven', 'Fuel_Type',
       'Transmission', 'Owner_Type', 'Mileage', 'Engine', 'Power', 'Seats',
       'Price', 'kmpl', 'Price_Lakh', 'lenght_of_names', 'Model', 'Brand',
       'CC', 'Horse_Power'],
      dtype='object')
```

+ Code + Markdown

```
data=original_data[['Price_Lakh','Brand','Model','Kilometers_Driven', 'kmpl','CC',
                    'Horse_Power', 'Year' , 'Seats', 'Location', 'Fuel_Type','Transmission', 'Owner_Type']]
```

✓ 0.0s

```
data
```

✓ 0.0s

	Price_Lakh	Brand	Model	Kilometers_Driven	kmpl	CC	Horse_Power	Year	Seats	Location	Fuel_Type	Transmission	Owner_Type
0	1.75	maruti	wagon r	72000	36.974	998.0	58.16	2010	5.0	mumbai	cng	manual	first
1	12.50	hyundai	creta 1.6	41000	19.670	1582.0	126.20	2015	5.0	pune	diesel	manual	first
2	4.50	honda	jazz	46000	18.200	1199.0	88.70	2011	5.0	chennai	petrol	manual	first
3	6.00	maruti	ertiga	87000	20.770	1248.0	88.76	2012	7.0	chennai	diesel	manual	first
4	17.74	audi	a4 new	40670	15.200	1968.0	140.80	2013	5.0	coimbatore	diesel	automatic	second
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6014	4.75	maruti	swift	27365	28.400	1248.0	74.00	2014	5.0	delhi	diesel	manual	first
6015	4.00	hyundai	xcent 1.1	100000	24.400	1120.0	71.00	2015	5.0	jaipur	diesel	manual	first
6016	2.90	mahindra	xylo d4	55000	14.000	2498.0	112.00	2012	8.0	jaipur	diesel	manual	second
6017	2.65	maruti	wagon r	46000	18.900	998.0	67.10	2013	5.0	kolkata	petrol	manual	first
6018	2.50	chevrolet	beat	47000	25.440	936.0	57.60	2011	5.0	hyderabad	diesel	manual	first

```
data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 5974 entries, 0 to 6018
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Price_Lakh         5974 non-null   float64
 1   Brand              5974 non-null   object
 2   Model              5974 non-null   object
 3   Kilometers_Driven  5974 non-null   int64
 4   kmpl               5974 non-null   float64
 5   CC                 5974 non-null   float64
 6   Horse_Power        5974 non-null   float64
 7   Year               5974 non-null   int64
 8   Seats              5974 non-null   float64
 9   Location           5974 non-null   object
 10  Fuel_Type          5974 non-null   object
 11  Transmission       5974 non-null   object
 12  Owner_Type         5974 non-null   object
dtypes: float64(5), int64(2), object(6)
memory usage: 653.4+ KB
```

C) Data Exploration ,visualization ,insights

- A) Univariate Questions and Answers
- B) Bivariate Analysis Questions
- C) Multivariate Analysis Questions

A) Univariate Questions and Answers:

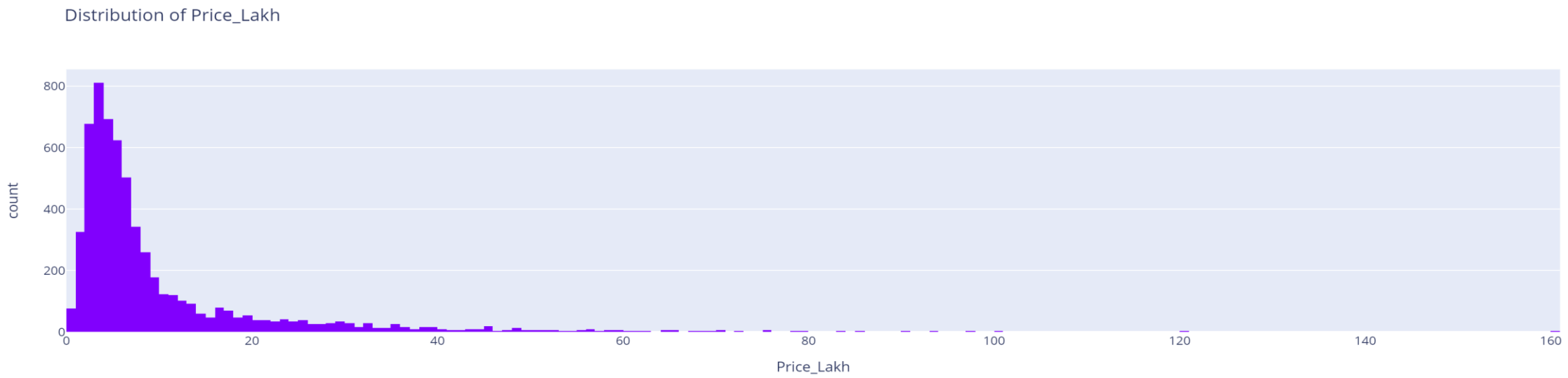
```
# see distribution for numeric varaibles
for i in data.select_dtypes("number").columns:
    fig=px.histogram(data,x=i,title=f"Distribution of {i}")
    fig.show()
```

✓ 0.4s

```
# visualize categorical data
for i in data.select_dtypes("object").columns:
    fig=px.histogram(data,x=i,title=i,text_auto=True,barmode="group").update_xaxes(categoryorder="total descending")
    fig.show()
```

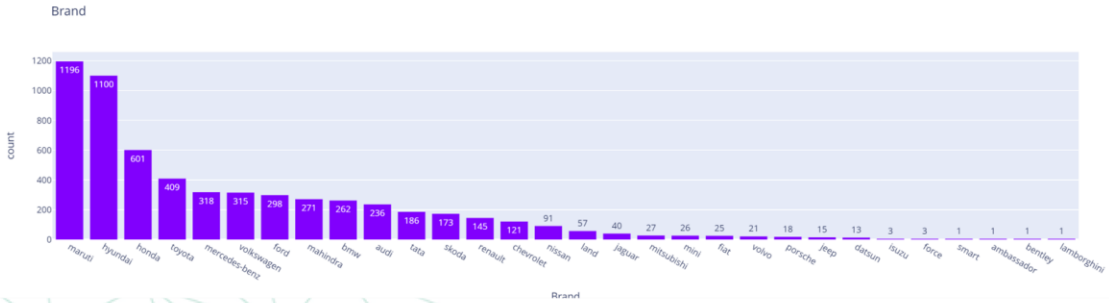
13] ✓ 0.4s

What is the distribution of car prices across the

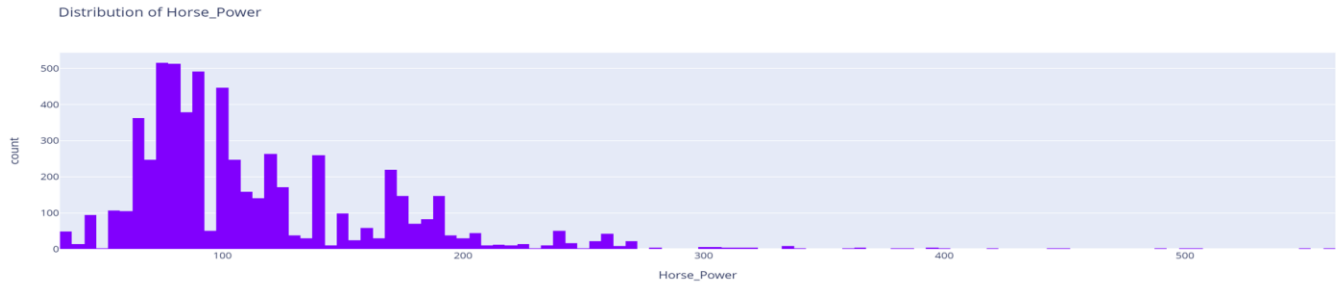


A) Univariate Questions and Answers:

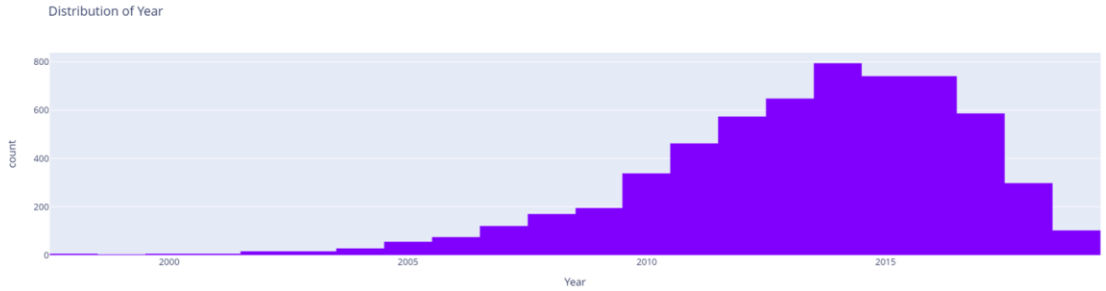
What is the most frequent car brand in the dataset?



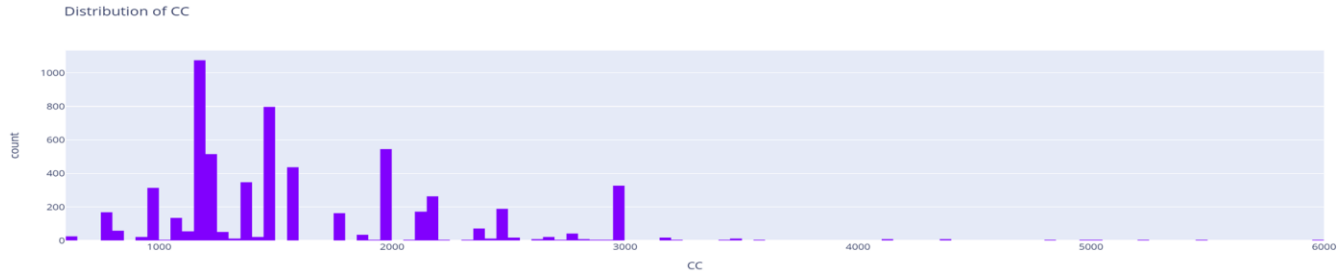
What is the distribution of vehicle Horse Power ?



How is the distribution of car manufacturing years spread across the dataset?

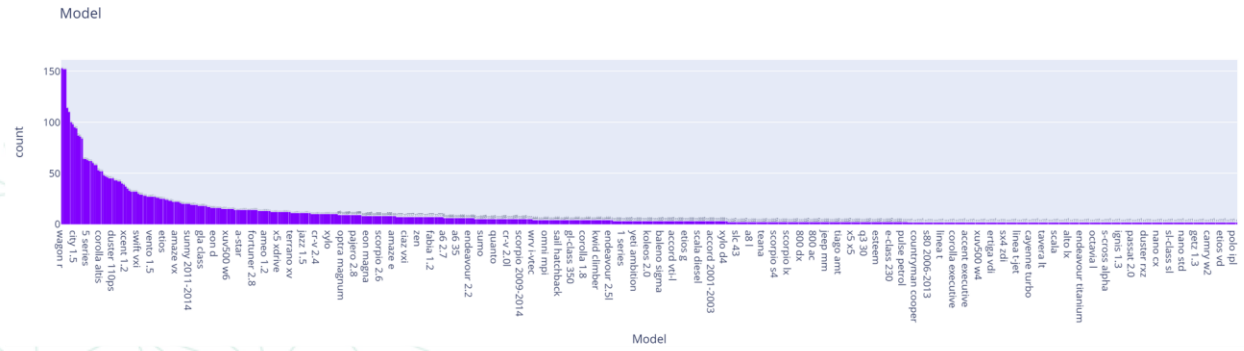


What is the distribution of vehicle cc ?

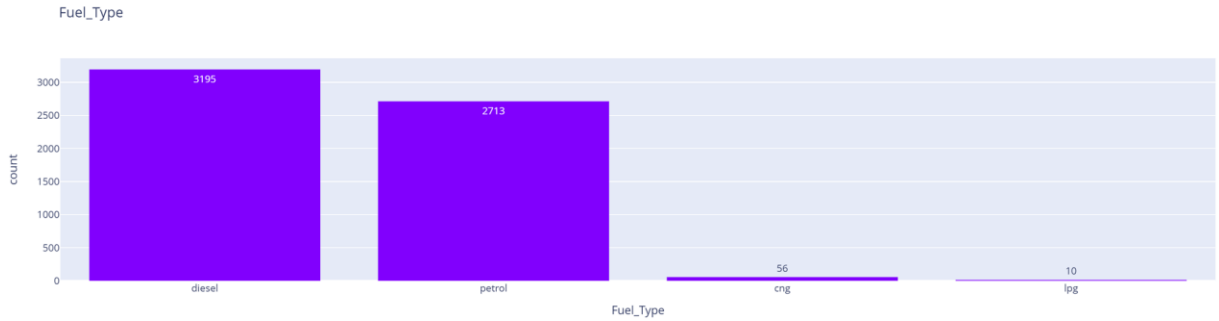


A) Univariate Questions and Answers:

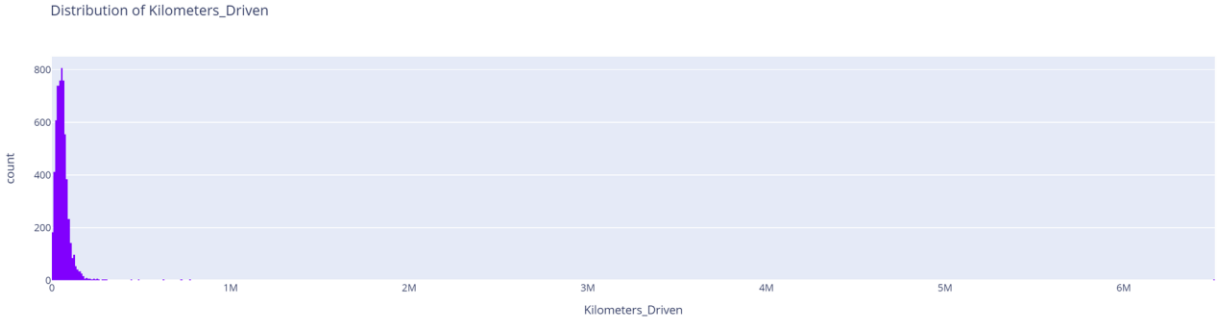
What is the most frequent car model in the dataset



What is the most common fuel type among used cars?



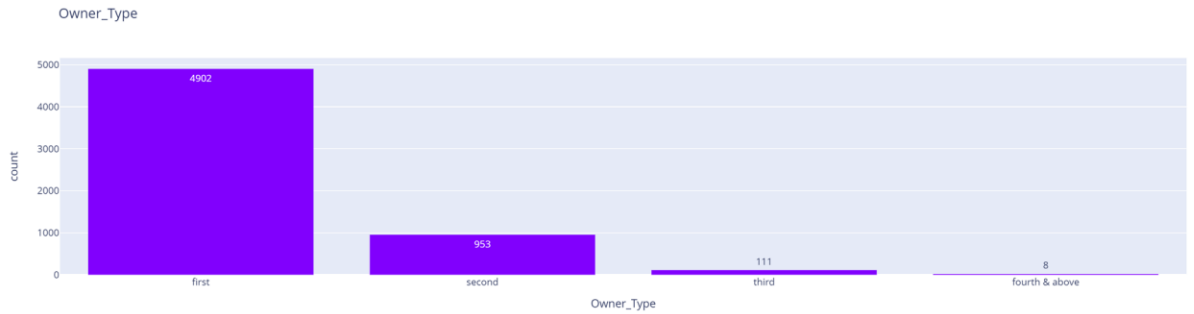
What is the distribution of Kilometer driven across the dataset?



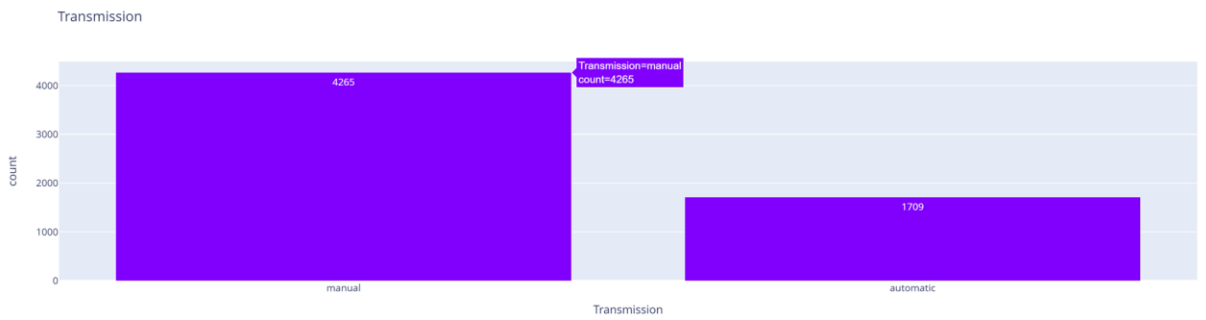
Which location have the most selling cars ?

A) Univariate Questions and Answers:

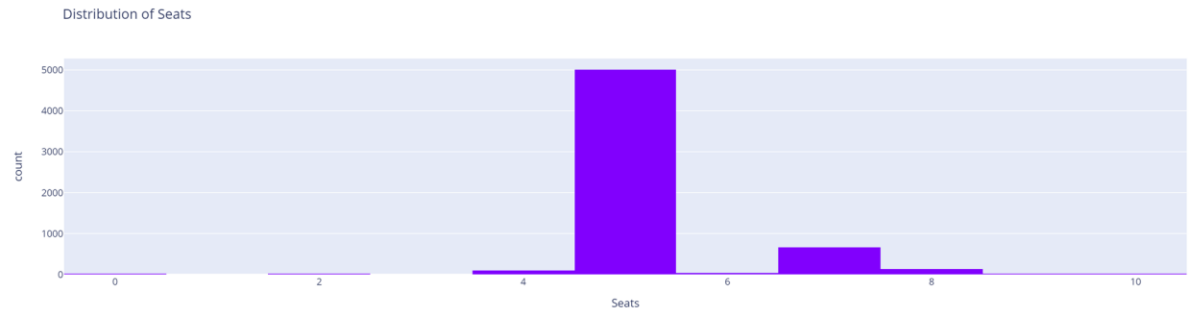
What is the most common ownership when selling the car ?



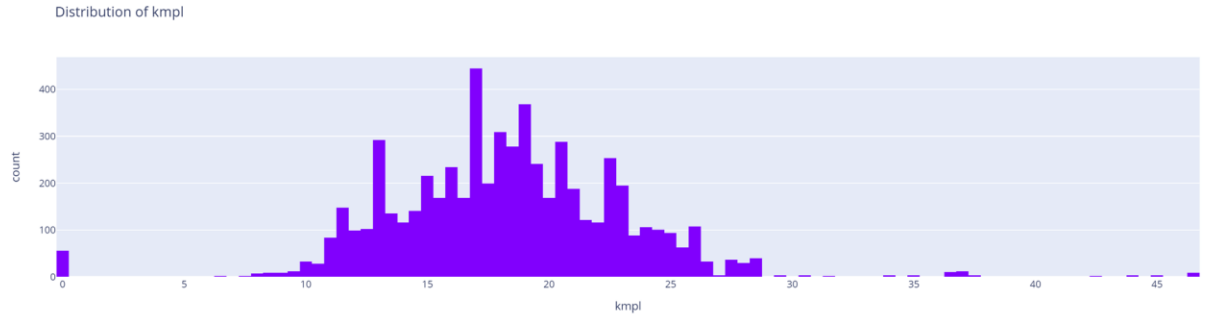
How many cars in the dataset have automatic transmission vs. manual transmission?



What is the most common number of seats type among used cars?



What is the distribution of vehicle fuel consumption?

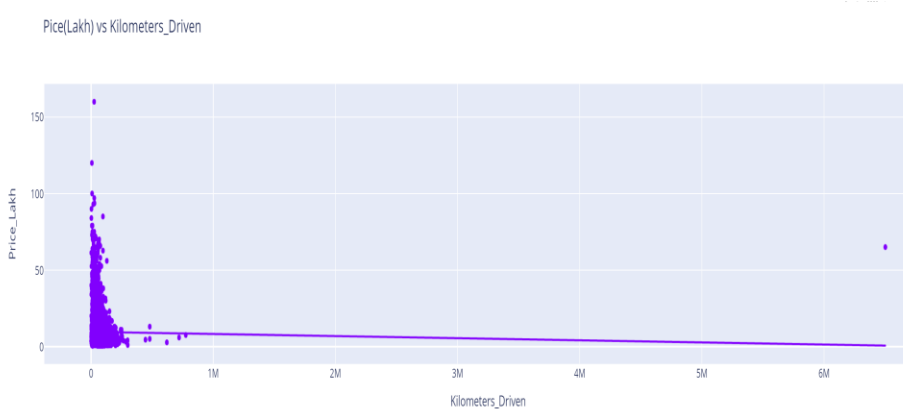


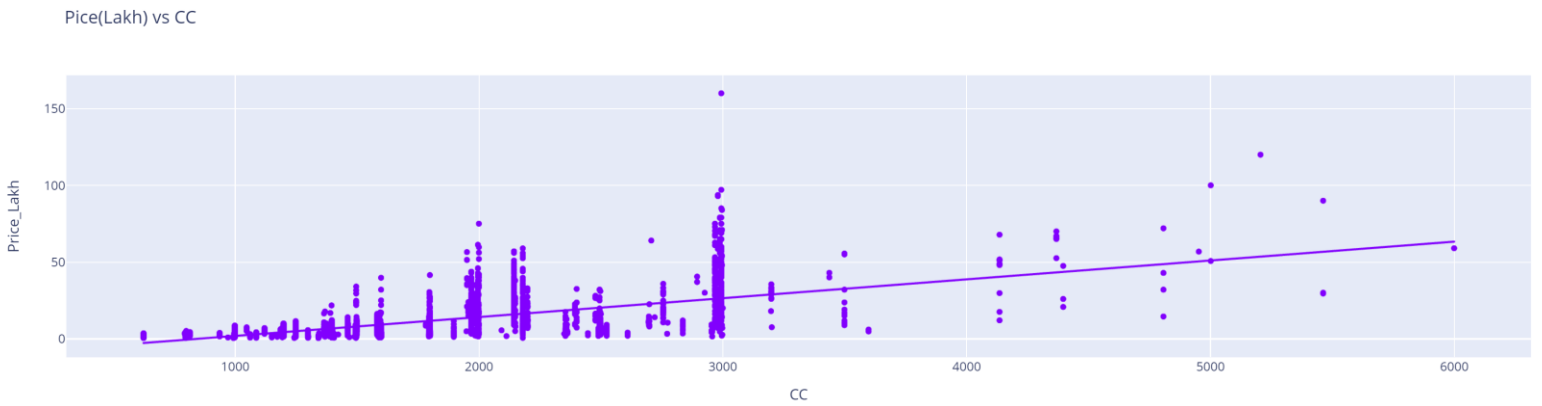
A) Univariate Insights:

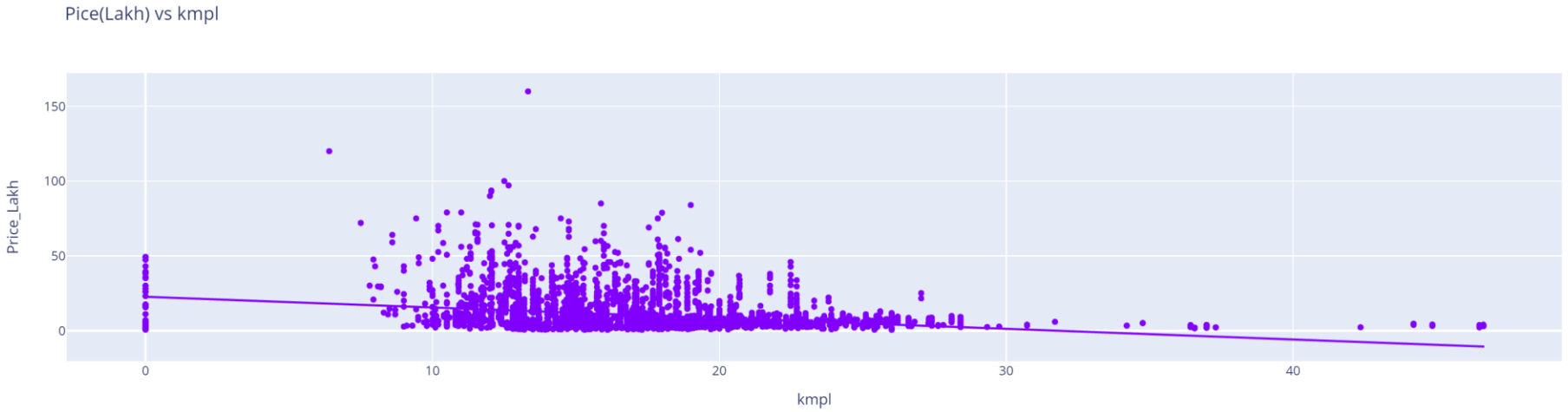
- In the price feature there are outliers and the range of prices from 0 to 160
- In the kilometer driven feature there are outliers and the range of prices from 0 to 6m
- In the kmpl feature the most widely fuel consumption is from 10 to 25 kmpl
- In the cc feature the most widely engine size from 1000 to 2000 cc
- In the Horse power feature the most widely engine horse power from 50 to 200 bhp
- In the year feature the most widely vehicle are from 2010 to 2014
- In the seats feature the most widely vehicle have 5 seats
- Maruti , Hyundai and Honda are the most 3 common brand
- Wagon r is the most selling vehicle
- Mumbai ,Hyderbad and Kochi are the most 3 locations that have car park
- Diesel and petrol are more common fuel types
- Manual transmission is preferable than automatic transmission
- The most selling vehicles are first owned

B) Bivariate Analysis Questions:

How does every numeric feature impact the target?

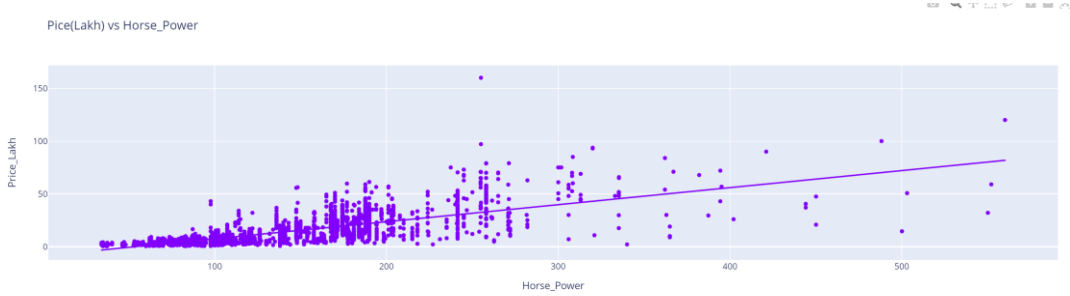


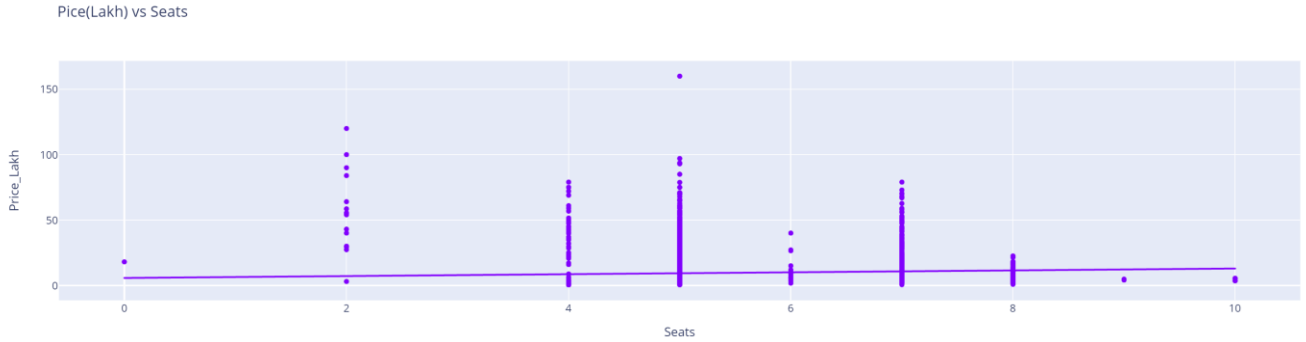


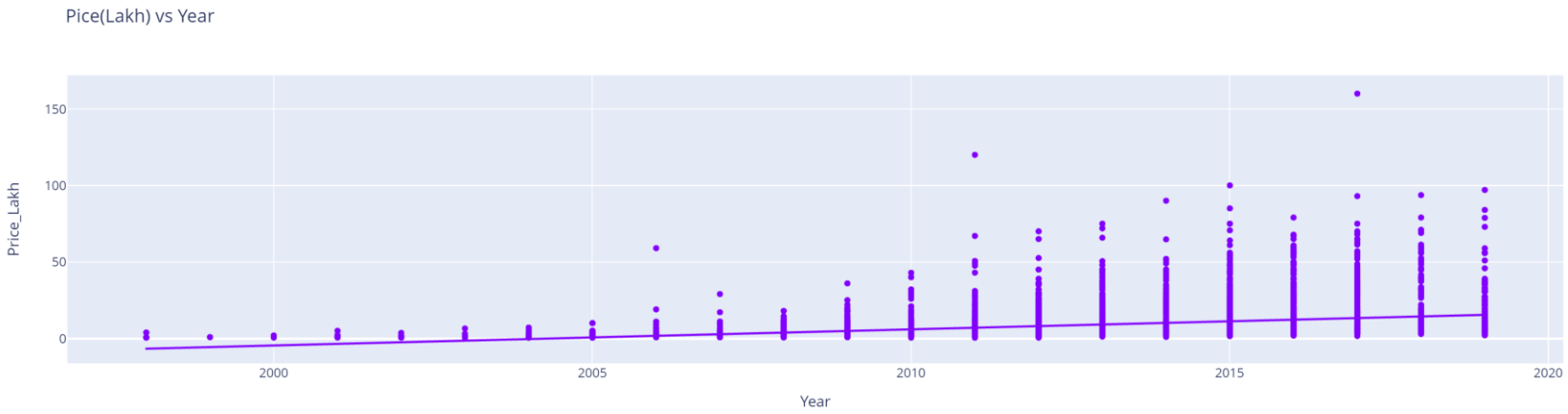


B) Bivariate Analysis Questions:

How does every numeric feature impact the target?







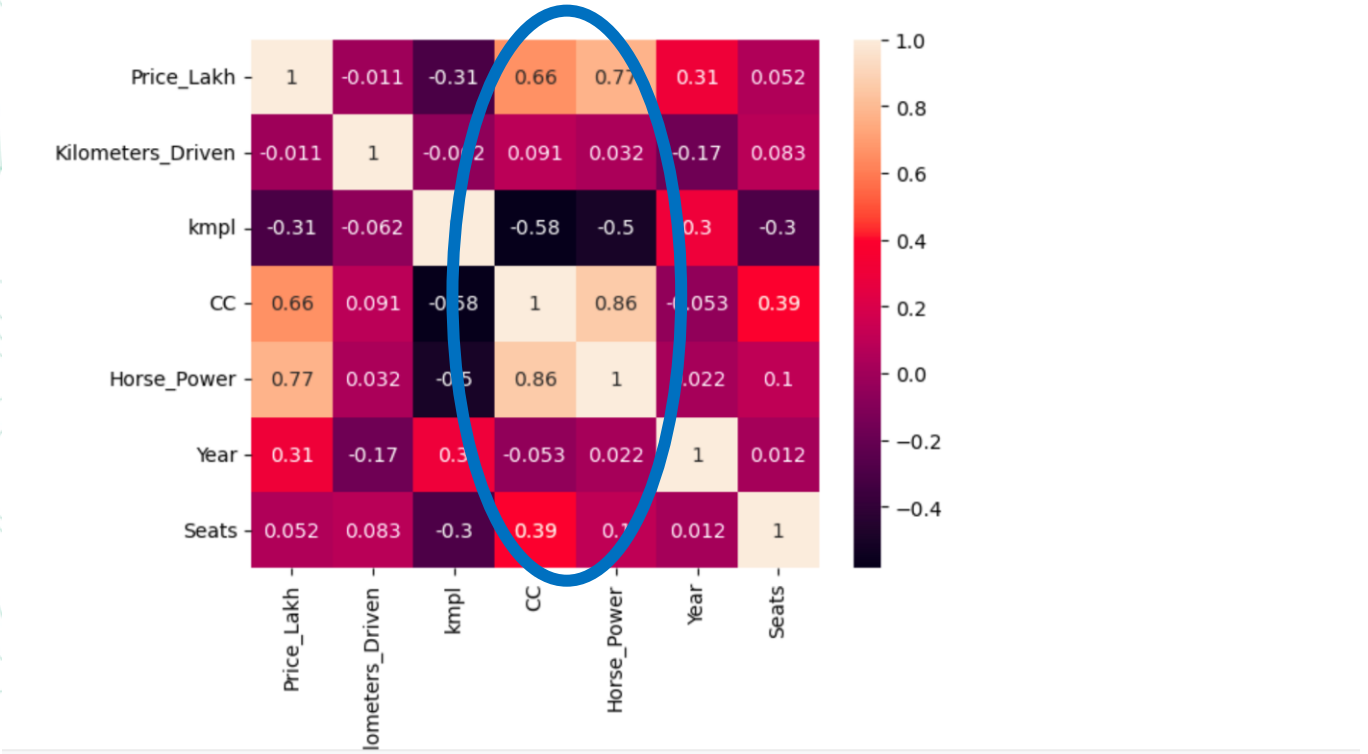
B) Bivariate Analysis Questions:

How does every numeric feature impact the target?

```
#Check for impact of numeric values
corr=data.select_dtypes("number").corr()
sns.heatmap(corr,annot=True)
```

✓ 0.5s

<Axes: >



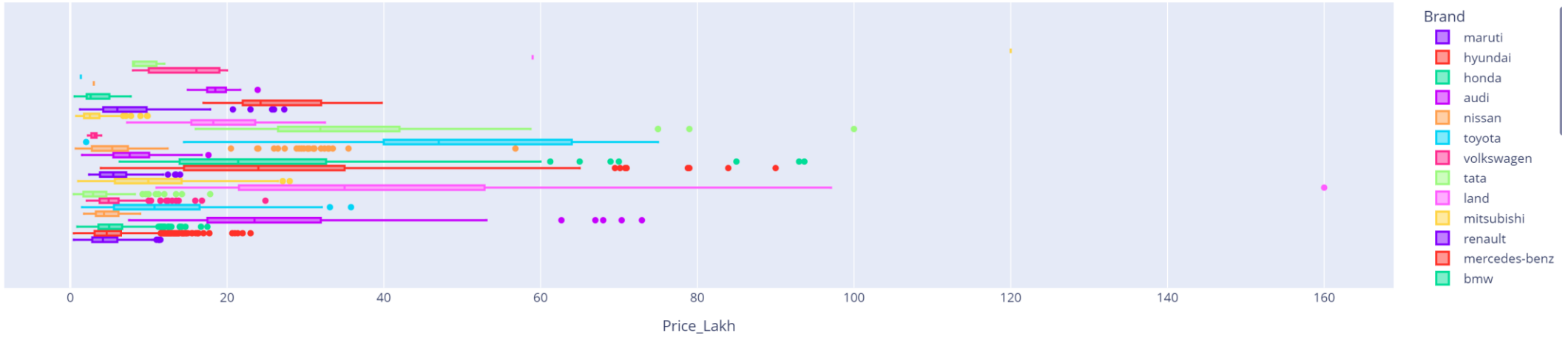
B) Bivariate Analysis Questions:

How does every categoric data feature impact the target?

```
# how does every categorical  feature impact the target
for i in data.select_dtypes("object").columns:
    if i=="Model":
        continue
    else:
        fig=px.box(data,x="Price_Lakh",color=i,title=f"Pice(Lakh) vs {i}")
        fig.show()
```

✓ 0.4s

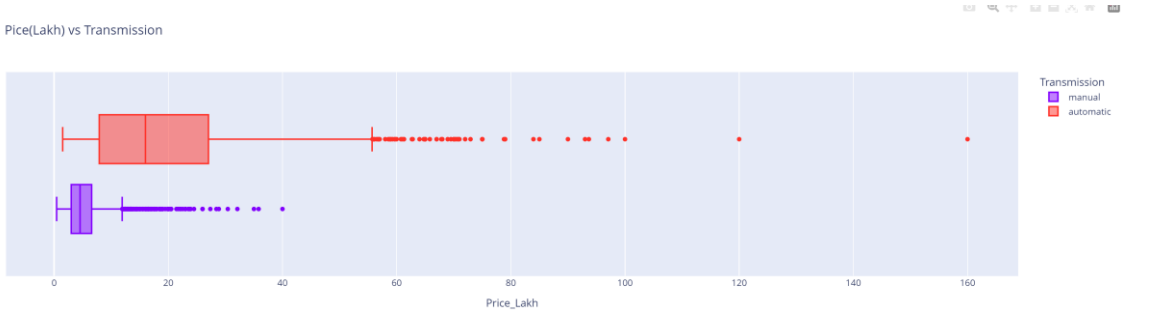
Pice(Lakh) vs Brand



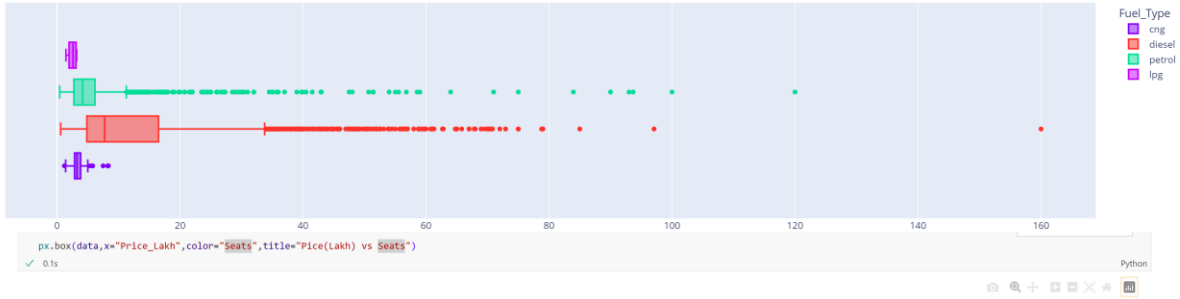
B) Bivariate Analysis Questions:

Pice(Lakh) vs Location

Pice(Lakh) vs Transmission

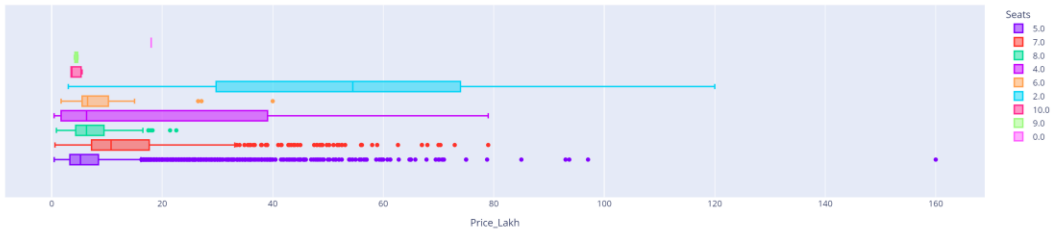


Pice(Lakh) vs Fuel_Type



Pice(Lakh) vs Owner_Type

Pice(Lakh) vs Seats



B) Bivariate Insights:

- The horse power and the cc are the most 2 numeric factors impact the price
- Price and brand are identified the price ladder for direct competitors such as Mercedes and BMW are neck to neck in the price interval from 15 to 50 lakh
- Diesel engine are the more expensive than others
- Automatic cars are expensive than other
- First owner vehicle are the most expensive
- The 2 seaters vehicles are expensive

C) Multivariate Analysis Questions:

how does CC impact Price for different fuel type and locations ?

```
fig = px.scatter(data, x='CC', y='Price_Lakh', color='Fuel_Type', facet_col='Location', title='CC vs Price for different fuel type and locations')
fig.show()
```

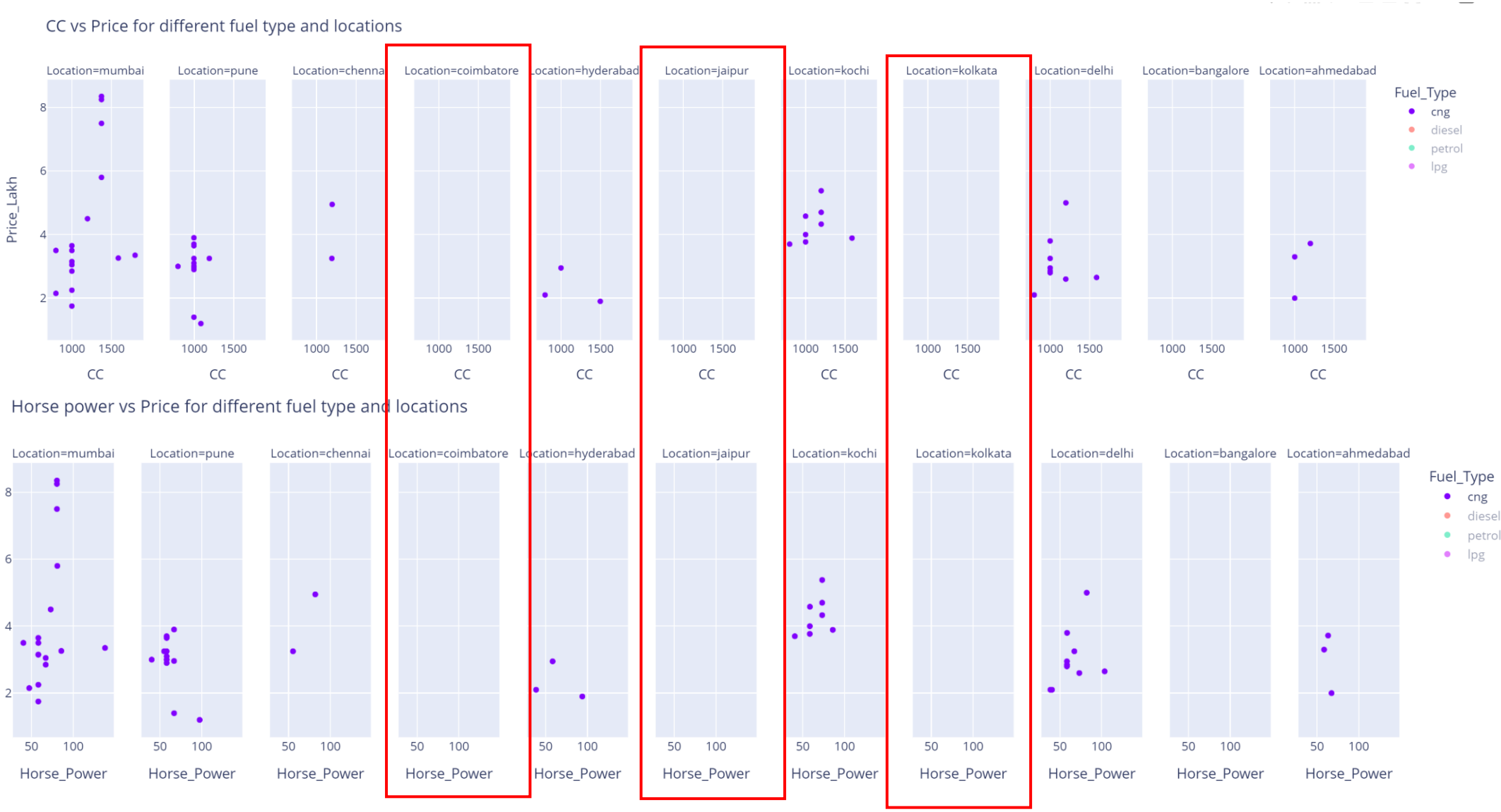
✓ 0.2s

how does Horse power impact Price for different fuel type and locations ?

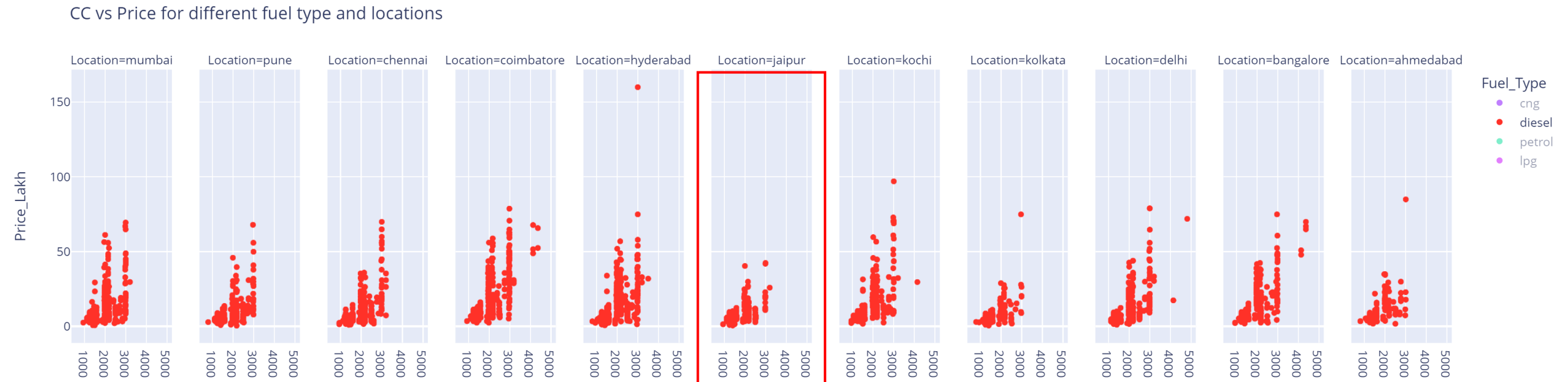
```
fig = px.scatter(data, x='Horse_Power', y='Price_Lakh', color='Fuel_Type', facet_col='Location', title='Horse power vs Price for different fuel type and locations')
fig.show()
```

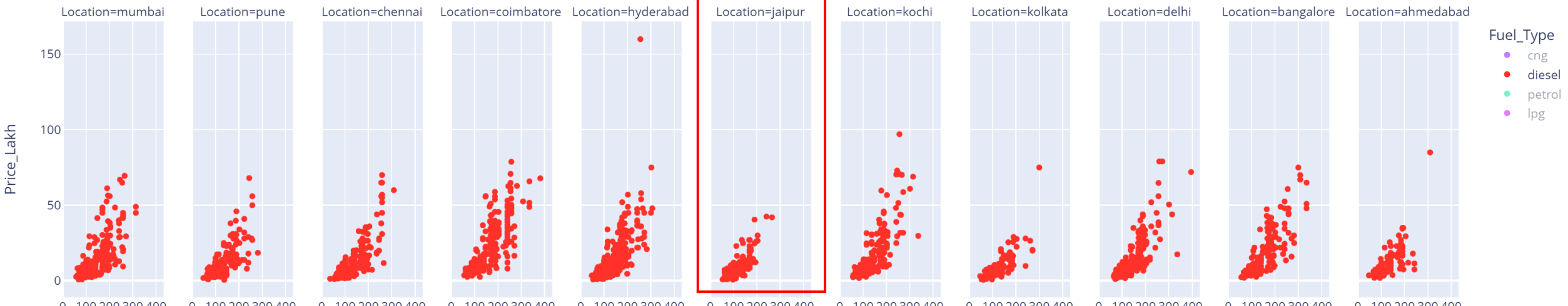
✓ 0.2s

C) Multivariate Analysis Questions:

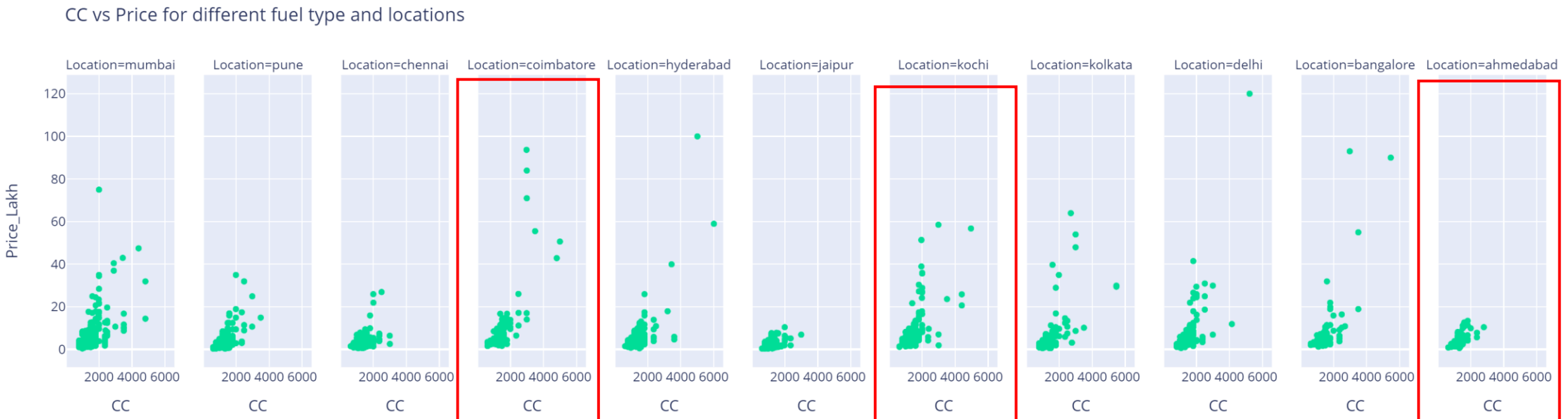


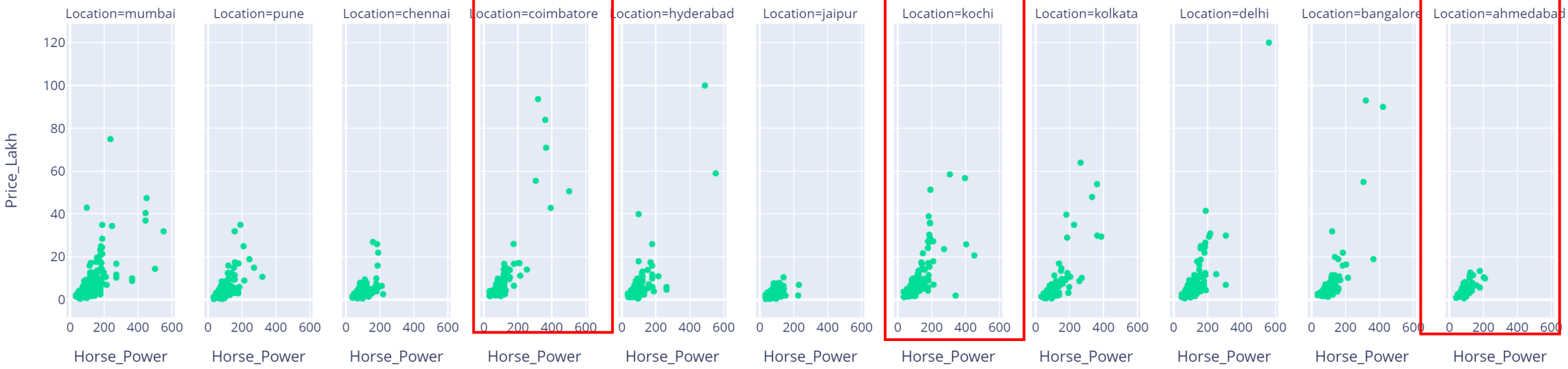
C) Multivariate Analysis Questions:





C) Multivariate Analysis Questions:

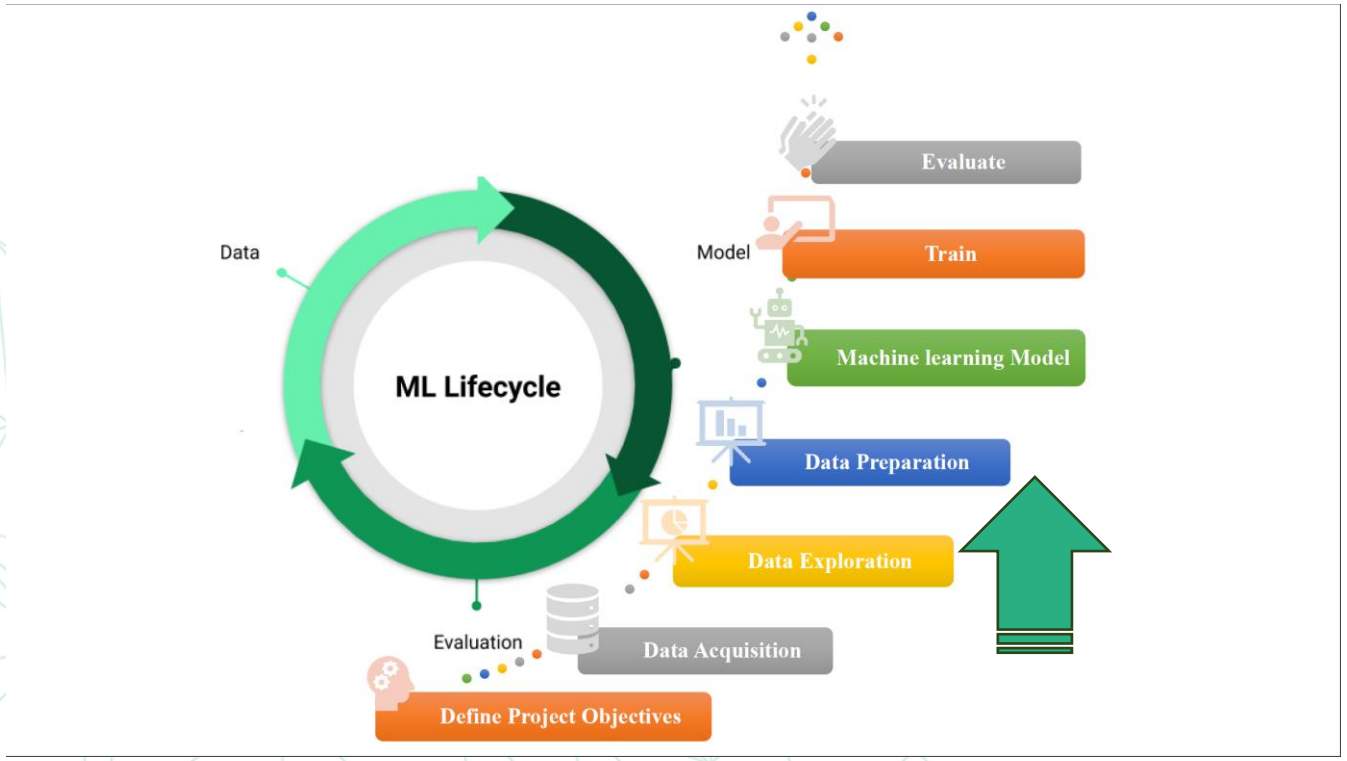





CC vs Price for different fuel type and locations
Location=mumbai
Location=pune
Location=chennai
Location=coimbatore
Location=hyderabad
Location=jaipur
Location=kochi
Location=kolkata
Location=delhi
Location=bangalore
Location=ahmedabad
Price_Lakh
Fuel_Type
cng
diesel
petrol
lpg
Horse power vs Price for different fuel type and locations
Horse_Power

B) Multivariate Insights:

- Jaipur, Kolkata, Coimbatore don’t like the CNG cars but Mumbai and pune like it
- Japur like diesel engines car with lower horse power , lower cc and lower cost
- ahmedbad like petrol engines cars with lower horse power , lower cc and lower cost
- Kochi have varieties of petrol and diesel engines with different horse power and cc and prices
- Lpg fuel type vehicle are rarely use and available only in hydepad and jaypur



4

Data Preparation Preprocessing

4. Data Preparation Preprocessing

```
data
```
✓ 0.0s

	Price_Lakh	Brand	Model	Kilometers_Driven	kmpl	CC	Horse_Power	Year	Seats	Location	Fuel_Type	Transmission	Owner_Type
0	1.75	maruti	wagon r	72000	36.974	998.0	58.16	2010	5.0	mumbai	cng	manual	first
1	12.50	hyundai	creta 1.6	41000	19.670	1582.0	126.20	2015	5.0	pune	diesel	manual	first
2	4.50	honda	jazz	46000	18.200	1199.0	88.70	2011	5.0	chennai	petrol	manual	first
3	6.00	maruti	ertiga	87000	20.770	1248.0	88.76	2012	7.0	chennai	diesel	manual	first
4	17.74	audi	a4 new	40670	15.200	1968.0	140.80	2013	5.0	coimbatore	diesel	automatic	second
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6014	4.75	maruti	swift	27365	28.400	1248.0	74.00	2014	5.0	delhi	diesel	manual	first
6015	4.00	hyundai	xcent 1.1	100000	24.400	1120.0	71.00	2015	5.0	jaipur	diesel	manual	first
6016	2.90	mahindra	xylo d4	55000	14.000	2498.0	112.00	2012	8.0	jaipur	diesel	manual	second
6017	2.65	maruti	wagon r	46000	18.900	998.0	67.10	2013	5.0	kolkata	petrol	manual	first
6018	2.50	chevrolet	beat	47000	25.440	936.0	57.60	2011	5.0	hyderabad	diesel	manual	first

5974 rows × 13 columns

```
data.info()
```
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 5974 entries, 0 to 6018
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Price_Lakh         5974 non-null   float64
 1   Brand              5974 non-null   object
 2   Model              5974 non-null   object
 3   Kilometers_Driven  5974 non-null   int64
 4   kmpl               5974 non-null   float64
 5   CC                 5974 non-null   float64
 6   Horse_Power        5974 non-null   float64
 7   Year               5974 non-null   int64
 8   Seats              5974 non-null   float64
 9   Location           5974 non-null   object
 10  Fuel_Type          5974 non-null   object
 11  Transmission       5974 non-null   object
 12  Owner_Type         5974 non-null   object
dtypes: float64(5), int64(2), object(6)
memory usage: 653.4+ KB
```

Comment

- **There are 5974 value and 12 feature and one target**
- **The data types and format are clear and there is no missing values**

Feature Scaling

Feature Scaling

```
#import preprocessing libraries
from sklearn.preprocessing import RobustScaler,StandardScaler
```
1] ✓ 0.2s

```
# identify target
target=data[["Price_Lakh"]]
```
2] ✓ 0.0s

```
# identify numeric and object features
x_numeric=data.select_dtypes("number")
x_object=data.select_dtypes("object")
```
4] ✓ 0.0s

Selecting best scaler for numeric data

```
# create robust scaler and standard scaler objects
robus_scaler=RobustScaler()
stand_scaler=StandardScaler()
```
] ✓ 0.0s

```
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.preprocessing import StandardScaler, RobustScaler
import pandas as pd
```
] ✓ 0.1s

Feature Scaling numeric values

```python
# Create a subplot with the number of rows equal to the number of columns in x_numeric
rows = len(x_numeric.columns)

# Create a subplot figure, with 3 columns per row (original, standard scaled, and robust scaled)
fig = make_subplots(rows=rows, cols=3, subplot_titles=["Original", "Standard Scaled", "Robust Scaled"])

for idx, col in enumerate(x_numeric.columns):
    # Original data
    fig.add_trace(
        go.Histogram(x=x_numeric[col], name=f"Original {col}"),
        row=idx+1, col=1
    )

    # Standard Scaled data
    standard_scaled = StandardScaler().fit_transform(x_numeric[[col]])
    fig.add_trace(
        go.Histogram(x=standard_scaled[:, 0], name=f"Standard Scaled {col}"),
        row=idx+1, col=2
    )

    # Robust Scaled data
    robust_scaled = RobustScaler().fit_transform(x_numeric[[col]])
    fig.add_trace(
        go.Histogram(x=robust_scaled[:, 0], name=f"Robust Scaled {col}"),
        row=idx+1, col=3
    )

# Update layout with titles and axis labels
fig.update_layout(height=300*rows, width=900, title_text="Original vs Scaled Data Histograms")
fig.show()
```

✓ 0.4s

Feature Scaling numeric values

Comment

- **Standard scaler can handle the target columns**
- **Robust scaler can handle the other features**

Feature Scaling numeric values

Solution

```
# transform x numeric or features
x_numeric_scaled=robus_scaler.fit_transform(x_numeric)
```

✓ 0.0s

```
# transform target feature
target_scaled=stand_scaler.fit_transform(target)
```

✓ 0.0s

Feature Scaling categorical values

Selecting best encoder for categorical data

```
#use label and one hot encoder
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
```

1] ✓ 0.0s

```
for i in x_object.columns:
    print(i)
    print(f"{len(x_object[i].unique())} unique value")
    print("-----------")
```

2] ✓ 0.0s

```
Brand
30 unique value
-----------
Model
646 unique value
-----------
Location
11 unique value
-----------
Fuel_Type
4 unique value
-----------
Transmission
2 unique value
-----------
Owner_Type
4 unique value
-----------
```

Comment

To avoid high dimensionality we will use label encoding for Brand , Model and Owner Type and one hot encoding for location , fuel type , transmission

Feature Scaling categorical values

Solution

```
# define feature scaling parameter for categorical data
one_hot=OneHotEncoder(drop="first")
label_encoder=LabelEncoder()
```

✓ 0.0s

```
x_object_one_hot=x_object[[ 'Location', 'Fuel_Type', 'Transmission']]
```

✓ 0.0s

```
x_object_label=x_object[['Brand', 'Model','Owner_Type']]
```

✓ 0.0s

```
x_object_one_hot_scaled=one_hot.fit_transform(x_object_one_hot).toarray()
```

✓ 0.0s

Feature scaling all data

create a data frame containing encoded categorical features

```
# create a data frame containing encoded categoricl features
cat_df=pd.DataFrame(x_object_one_hot_scaled,columns=one_hot.get_feature_names_out())
```

✓ 0.0s

```
for i in x_object_label.columns:
    cat_df[f'{i}_labeld']=label_encoder.fit_transform(x_object_label[i])
```

✓ 0.0s

```
cat_df
```

✓ 0.0s

create a data frame containing encoded categorical features

```
# create a data frame containg all numeric features after scaling
numeric_df=pd.DataFrame(x_numeric_scaled,columns=robus_scaler.get_feature_names_out())
```

✓ 0.0s

Feature scaling all data

concatenate all features together numerical and categorical

```
# concatenate all features together numerical and categorical
features=pd.concat([cat_df,numeric_df],axis=1)
```

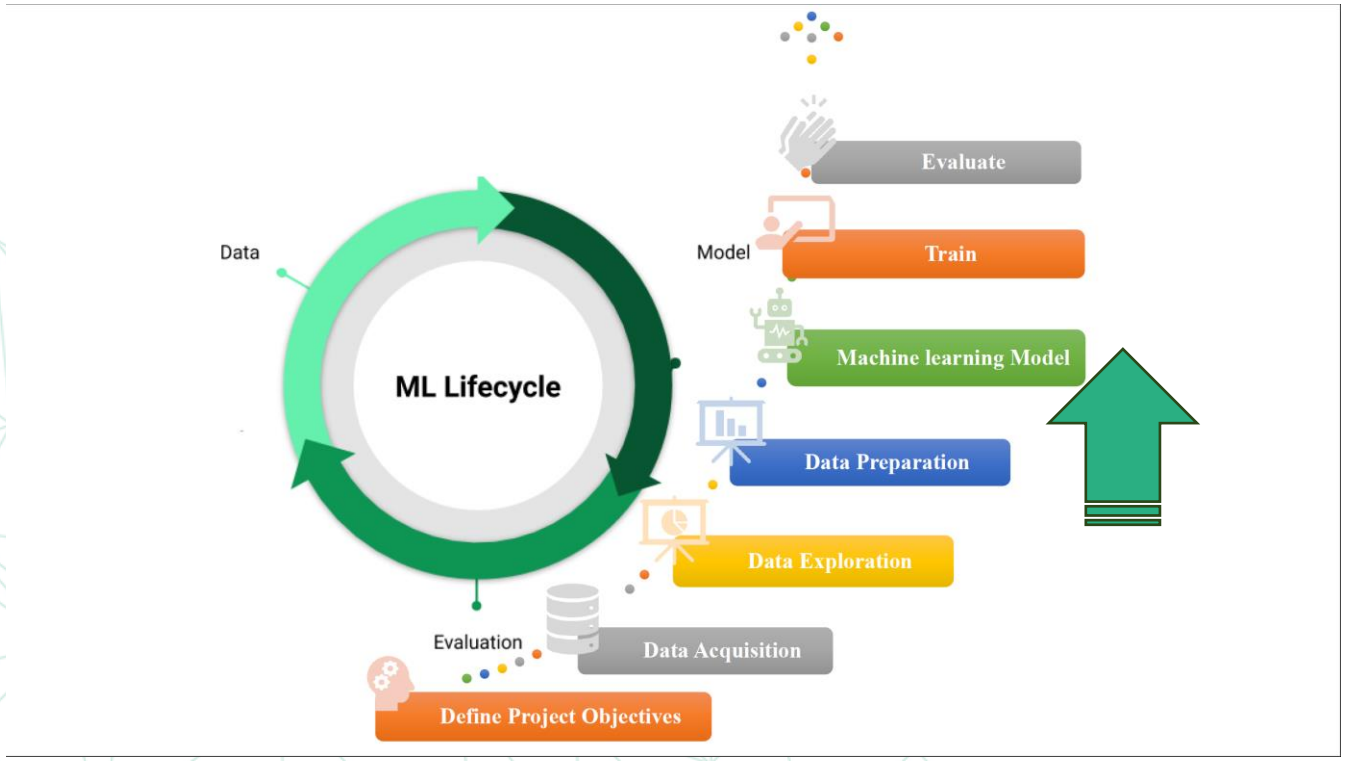
✓ 0.0s

```
features.columns
```

✓ 0.0s

```
Index(['Location_bangalore', 'Location_chennai', 'Location_coimbatore',
       'Location_delhi', 'Location_hyderabad', 'Location_jaipur',
       'Location_kochi', 'Location_kolkata', 'Location_mumbai',
       'Location_pune', 'Fuel_Type_diesel', 'Fuel_Type_lpg',
       'Fuel_Type_petrol', 'Transmission_manual', 'Brand_labeld',
       'Model_labeld', 'Owner_Type_labeld', 'Price_Lakh', 'Kilometers_Driven',
       'kmpl', 'CC', 'Horse_Power', 'Year', 'Seats'],
      dtype='object')
```

```
# create a data freame to contain target values
target_scaled=pd.DataFrame(target_scaled,columns=stand_scaler.get_feature_names_out())
```



5

Machine Learning Model + Evaluation

5.Machine Learning Model + Evaluation

Import Libraries for regression model

```
# Import  libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,Ridge,Lasso
from sklearn.metrics import  r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor
```

✓ 0.0s

Split data

```
# split data
X_train, X_test, y_train, y_test=train_test_split(features,target_scaled, test_size=0.2, random_state=42)
```

✓ 0.1s

Model Selection

Create a function to give an evaluation for the regression model used

```
# create a function to give an evaluation for the regression model used
def regression_model(model_name,xtr=X_train,ytr=y_train,xte=X_test,yte=y_test):
    reg_model=model_name
    reg_model.fit(xtr,ytr)
    y_hat_test=reg_model.predict(xte)
    r2_test= r2_score(yte,y_hat_test)
    r2_train=r2_score(ytr,(reg_model.predict(xtr)))
    conclusion=r2_train - r2_test
    if conclusion <= 0.05 :
        fitting = "underfit"
    elif conclusion > 0.1:
        fitting = "overfit"
    else:
        fitting = "good fitting"

    print (f'''
          \n The model used is {model_name} model and the training data r_squared = {r2_train}
          \n and test data r_squared={r2_test}
          \n and the differenc between them = {conclusion} and
          \n the model is {fitting}''')
```

✓ 0.0s

Model Selection

Using liner regression model

```
# using linear regression model
regression_model(LinearRegression())
```
✓ 0.0s

the model used is LinearRegression() and training data r_squared = 1.0

and test data r_squared=1.0

and the differenc between them = 0.0 and

the model is underfit

Using Lasso regression model

```
# using Lasso regression model
regression_model(Lasso())
```
✓ 0.1s

the model used is Lasso() and training data r_squared = 0.6539088982033916

and test data r_squared=0.6530571996199737

and the differenc between them = 0.0008516985834179369 and

the model is underfit

Using Ridge regression model

```
# using Ridge regression model
regression_model(Ridge())
```
✓ 0.1s

the model used is Ridge() and training data r_squared = 0.9999999821858558

and test data r_squared=0.999999965761808

and the differenc between them = 1.6424047788454743e-08 and

the model is underfit

Using Random Forest regression model

```
# using randome forest  regression model
regression_model (RandomForestRegressor(random_state=1))
```
✓ 2.8s

c:\Users\DELL\anaconda3\Lib\site-packages\sklearn\base.py:1474: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().

the model used is RandomForestRegressor(random_state=1) and training data r_squared = 0.9984852272563436

and test data r_squared=0.9992050197523394

and the differenc between them = -0.0007197924959958524 and

the model is underfit

Model Selection

Using XGBRegressor Forest regression model

```
regression_model(XGBRegressor(n_estimators=300, learning_rate=0.5, max_depth=1, min_child_weight=2, n_jobs=4 ,reg_alpha=0.1, reg_lambda=0.1))
```

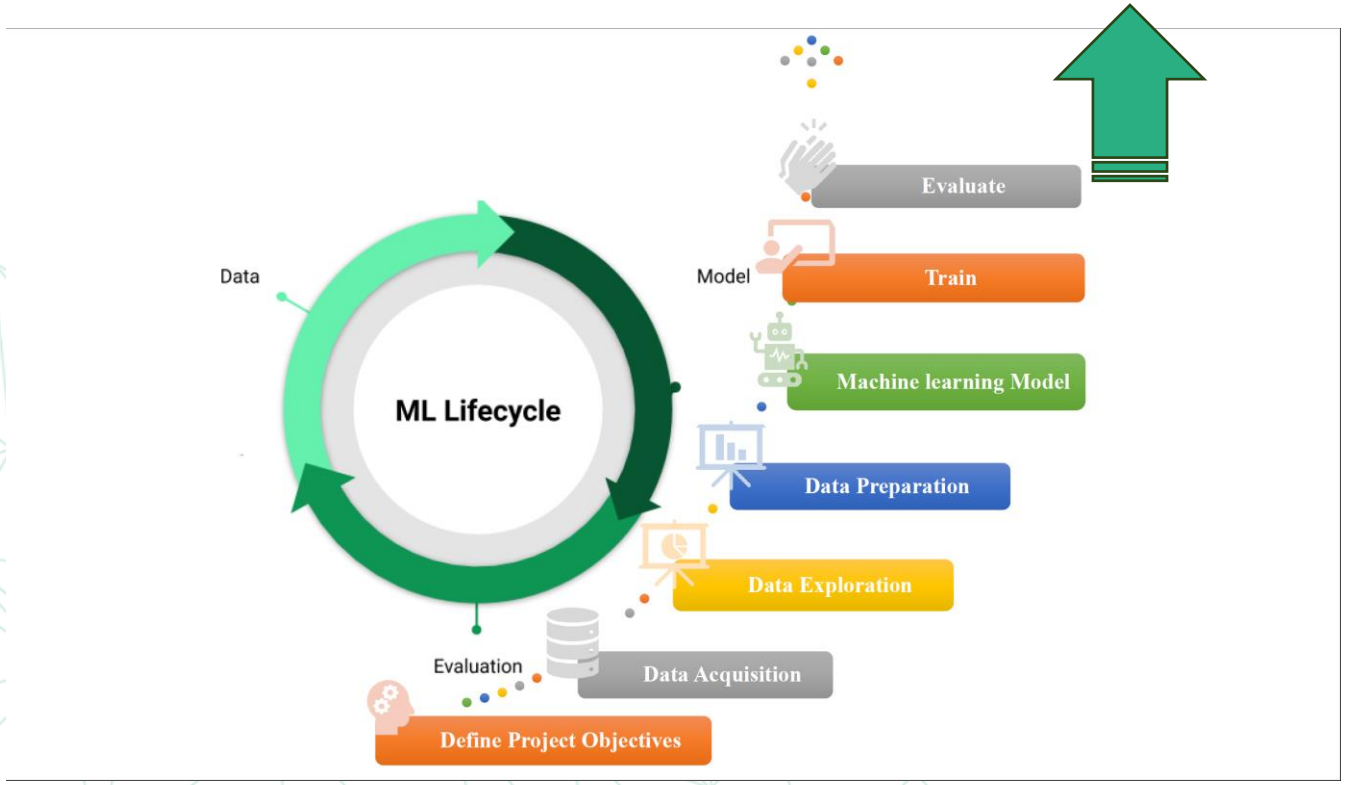
✓ 0.3s Python

```
the model used is 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=0.5, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=1, max_leaves=None,
             min_child_weight=2, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=300, n_jobs=4,
             num_parallel_tree=None, random_state=None, ...) and training data r_squared = 0.8384986519813538  and test data r_squared=0.745514452457428and the differenc betwe
```

training data r_squared = 0.8384986519813538 and test data r_squared=0.745514452457428and the difference between them = 0.09298419952392578 and the model is good fitt

Comment

XGBRegressor is the only model that give good fit all other models give under fit



6

Deployment

6.Deploy the model using dash

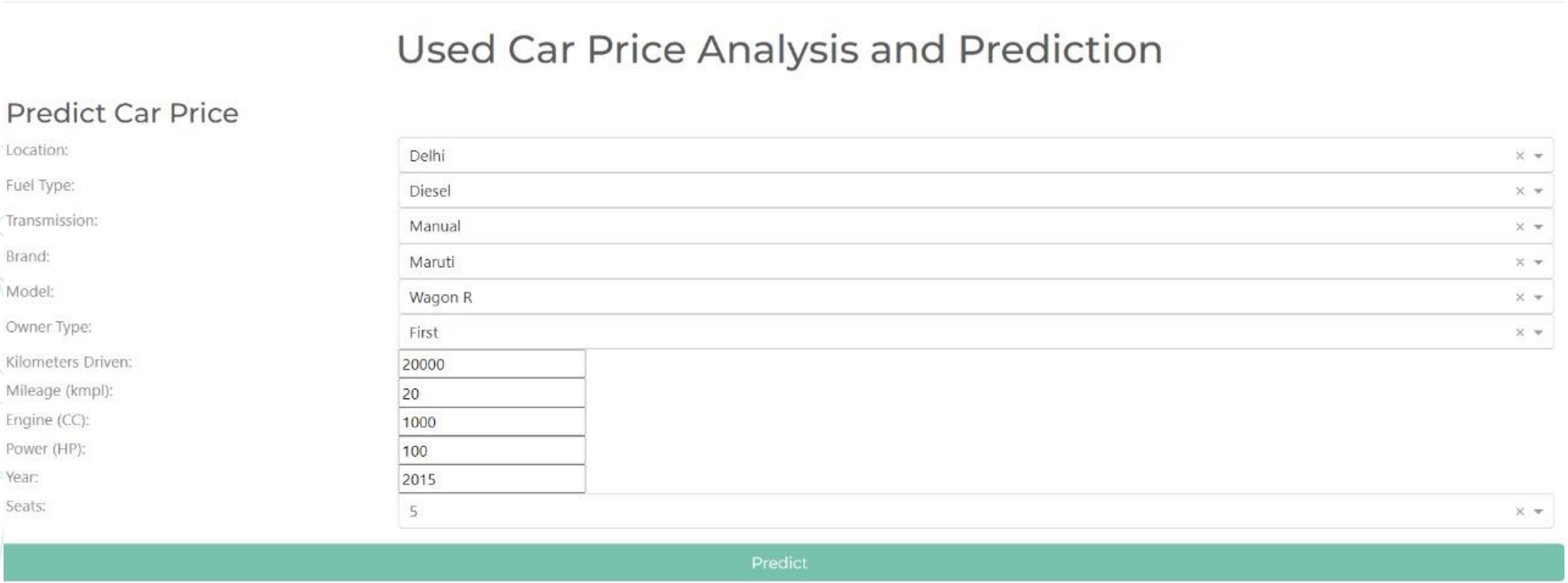

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import dash_bootstrap_components as dbc
import joblib
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv("car.csv")

# Initialize the Dash app with Bootstrap
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.MINTY])

# Load the dataset
df = pd.read_csv("car.csv")

# Initialize the Dash app with Bootstrap
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.MINTY])

app.layout = dbc.Container(
    [
        html.H1("Used Car Price Analysis and Prediction", className="text-center my-4"),

        html.H3("Predict Car Price"),
        dbc.Row([
            dbc.Col(html.Label('Location:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-loc',
                                 options=[{'label': i, 'value': i} for i in df['Location'].unique()],
                                 value='Delhi'), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Fuel Type:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-fuel',
                                 options=[{'label': i, 'value': i} for i in df['Fuel_Type'].unique()],
                                 value='Petrol'), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Transmission:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-transmission',
                                 options=[{'label': i, 'value': i} for i in df['Transmission'].unique()],
                                 value='Manual'), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Brand:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-brand',
                                 options=[{'label': i, 'value': i} for i in df['Brand'].unique()],
                                 value='Maruti'), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Model:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-model',
                                 options=[{'label': i, 'value': i} for i in df['Model'].unique()],
                                 value='Wagon R'), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Owner Type:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-owner',
                                 options=[{'label': i, 'value': i} for i in df['Owner_Type'].unique()],
                                 value='First'), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Kilometers Driven:'), width=3),
            dbc.Col(dcc.Input(id='input-kil', type='number', value=20000), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Mileage (kmpl):'), width=3),
            dbc.Col(dcc.Input(id='input-mil', type='number', value=20), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Engine (CC):'), width=3),
            dbc.Col(dcc.Input(id='input-eng', type='number', value=1000), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Power (HP):'), width=3),
            dbc.Col(dcc.Input(id='input-pow', type='number', value=100), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Year:'), width=3),
            dbc.Col(dcc.Input(id='input-year', type='number', value=2015), width=9)
        ]),
        dbc.Row([
            dbc.Col(html.Label('Seats:'), width=3),
            dbc.Col(dcc.Dropdown(id='input-seats',
                                 options=[{'label': i, 'value': i} for i in np.arange(1, 11)],
                                 value=5), width=9)
        ]),
```

6.Deploy the model using dash

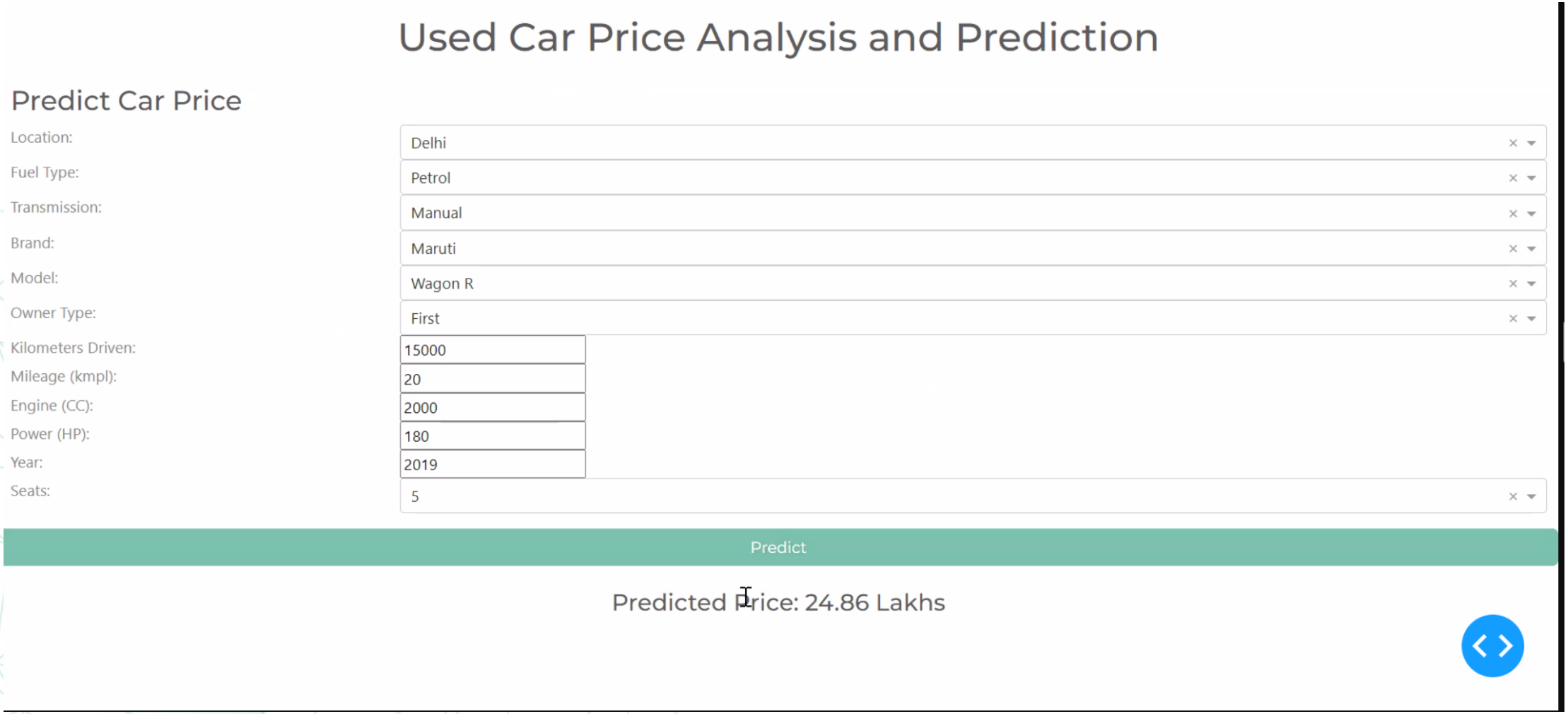
وزارة الاتصـــــالات
وتكنولوجيا المعلومات
رواد مصر الرقمية

```
        ]),

        dbc.Row(
            dbc.Button('Predict', id='predict-button', color='primary', className="mt-3"),
            justify="center"
        ),
        html.H4(id='prediction-result', children='Predicted Price: ', className="mt-4 text-center")
    ],
    fluid=True
)

@app.callback(
    Output('prediction-result', 'children'),
    [Input('predict-button', 'n_clicks')],
    [Input('input-loc', 'value'),
     Input('input-fuel', 'value'),
     Input('input-transmission', 'value'),
     Input('input-brand', 'value'),
     Input('input-model', 'value'),
     Input('input-owner', 'value'),
     Input('input-kil', 'value'),
     Input('input-mil', 'value'),
     Input('input-eng', 'value'),
     Input('input-pow', 'value'),
     Input('input-year', 'value'),
     Input('input-seats', 'value')]
)
def predict_price(n_clicks, loc, fuel, transmission, brand, model, owner, kil, mil, eng, pow, year, seats):
    if n_clicks:
        xgb = joblib.load('car_price_model.pkl')
        stand_scaler = joblib.load('stand_scaler.pkl')
        robust_scaler = joblib.load('robust_scaler.pkl')
        brand_encoder = joblib.load('brand_encoder.pkl')
        model_encoder = joblib.load('model_encoder.pkl')
        owner_encoder = joblib.load('owner_encoder.pkl')
        one_hot = joblib.load('one_hot.pkl')

        input_data = pd.DataFrame([[loc.lower(), fuel.lower(), transmission.lower(), brand.lower(), model.lower(), owner.lower(), kil, mil, eng, pow, year, seats]],
                                  columns=['Location', 'Fuel_Type', 'Transmission', 'Brand', 'Model', 'Owner_Type', 'Kilometers_Driven', 'kmpl', 'CC', 'Horse_Power','Year', 'Seats'])

        input_num = input_data[['Kilometers_Driven', 'kmpl', 'CC', 'Horse_Power','Year', 'Seats']]
        input_num = robust_scaler.transform(input_num)
        input_num = pd.DataFrame(input_num, columns=robust_scaler.get_feature_names_out())

        input_label = input_data[["Brand", "Model", "Owner_Type"]]
        input_label['Brand'] = brand_encoder.transform(input_label['Brand'])
        input_label['Model'] = model_encoder.transform(input_label['Model'])
        input_label['Owner_Type'] = owner_encoder.transform(input_label['Owner_Type'])

        input_one_hot = input_data[['Location', 'Fuel_Type', 'Transmission']]
        input_one_hot = one_hot.transform(input_one_hot).toarray()
        input_one_hot = pd.DataFrame(input_one_hot, columns=one_hot.get_feature_names_out())

        cat = pd.concat([input_one_hot, input_label], axis=1)
        df = pd.concat([cat, input_num], axis=1)

        input = df.iloc[0, 0:]
        input = pd.DataFrame([input.values], columns=xgb.get_booster().feature_names)

        predicted_price = xgb.predict(input)
        predicted_price = stand_scaler.inverse_transform(predicted_price.reshape(-1, 1))

if __name__ == '__main__':
    app.run_server(debug=True)
```

6.Deploy the model using dash

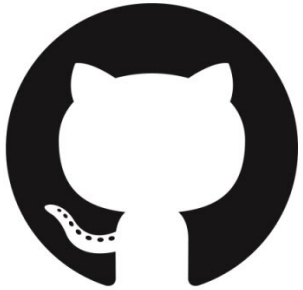


6.Deploy the model using dash



THANK YOU

وزارة الاتصالات وتكنولوجيا المعلومات
رواد مصر الرقمية



GitHub