Market Segment Analysis of EV Vehicles

Team Members:
Taksh Prajpati
Durga Sanagiri
Shruthi Ganesh
Sanket Bangar
Varun Kumar

Market Segment Analysis of EV Vehicles
by
Prajapati Taksh Rajeshkumar

Dataset used:
https://drive.google.com/file/d/1yeTKNvAxCALz4QIKluGZqDFc6GbHt9dV/view?usp=sharing

Project link:
https://github.com/TakshPrajapati/Intern_Feynnlabs

1. Data Pre-Processing:

Data preprocessing is a crucial step in preparing raw data to make it suitable for machine learning models. The process involves cleaning the data, removing any errors or inconsistencies, and transforming it into a format that can be easily analyzed. It is essential to preprocess the data before performing any segmentation analysis.

To preprocess data, the first step is to import the raw data in a suitable format and create a data frame for further analysis. The next step is to identify any null values in the dataset and remove them to avoid any data inconsistencies.

CAR DETAILS V3

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import re
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df=pd.read_csv("car details v3.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
0	Maruti Swift Dzire VDI	2014	450000	145500	Diesel	Individual	Manual	First Owner	23.4 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	5.0
1	Skoda Rapid 1.5 TDI Ambition	2014	370000	120000	Diesel	Individual	Manual	Second Owner	21.14 kmpl	1498 CC	103.52 bhp	250Nm@ 1500-2500rpm	5.0
2	Honda City 2017-2020 EXi	2006	158000	140000	Petrol	Individual	Manual	Third Owner	17.7 kmpl	1497 CC	78 bhp	12.7@ 2,700(kgm@ rpm)	5.0
3	Hyundai i20 Sportz Diesel	2010	225000	127000	Diesel	Individual	Manual	First Owner	23.0 kmpl	1396 CC	90 bhp	22.4 kgm at 1750-2750rpm	5.0
4	Maruti Swift VXI BSIII	2007	130000	120000	Petrol	Individual	Manual	First Owner	16.1 kmpl	1298 CC	88.2 bhp	11.5@ 4,500(kgm@ rpm)	5.0

In [7]:
```
df.describe(include="all")
```

Out[7]:

	name	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	torque	seats
count	8128	8128.000000	8.128000e+03	8.128000e+03	8128	8128	8128	8128	7907	7907	7913	7906	7907.000000
unique	2058	NaN	NaN	NaN	4	3	2	5	393	121	322	441	NaN
top	Maruti Swift Dzire VDI	NaN	NaN	NaN	Diesel	Individual	Manual	First Owner	18.9 kmpl	1248 CC	74 bhp	190Nm@ 2000rpm	NaN
freq	129	NaN	NaN	NaN	4402	6766	7078	5289	225	1017	377	530	NaN
mean	NaN	2013.804011	6.382718e+05	6.981951e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.416719
std	NaN	4.044249	8.062534e+05	5.655055e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.959588
min	NaN	1983.000000	2.999900e+04	1.000000e+00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.000000
25%	NaN	2011.000000	2.549990e+05	3.500000e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.000000
50%	NaN	2015.000000	4.500000e+05	6.000000e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.000000
75%	NaN	2017.000000	6.750000e+05	9.800000e+04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.000000
max	NaN	2020.000000	1.000000e+07	2.360457e+06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	14.000000

In [8]:
```
df.isna().sum()
```

Out[8]:
```
name              0
year              0
selling_price     0
km_driven         0
fuel              0
seller_type       0
transmission      0
owner             0
mileage         221
engine          221
max_power       215
torque          222
seats           221
dtype: int64
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   name           8128 non-null   object
 1   year           8128 non-null   int64
 2   selling_price  8128 non-null   int64
 3   km_driven      8128 non-null   int64
 4   fuel           8128 non-null   object
 5   seller_type    8128 non-null   object
 6   transmission   8128 non-null   object
 7   owner          8128 non-null   object
 8   mileage        7907 non-null   object
 9   engine         7907 non-null   object
 10  max_power      7913 non-null   object
 11  torque         7906 non-null   object
 12  seats          7907 non-null   float64
dtypes: float64(1), int64(3), object(9)
memory usage: 825.6+ KB
```

In [5]:
```
df.describe()
```

Out[5]:

	year	selling_price	km_driven	seats
count	8128.000000	8.128000e+03	8.128000e+03	7907.000000
mean	2013.804011	6.382718e+05	6.981951e+04	5.416719
std	4.044249	8.062534e+05	5.655055e+04	0.959588
min	1983.000000	2.999900e+04	1.000000e+00	2.000000
25%	2011.000000	2.549990e+05	3.500000e+04	5.000000
50%	2015.000000	4.500000e+05	6.000000e+04	5.000000
75%	2017.000000	6.750000e+05	9.800000e+04	5.000000
max	2020.000000	1.000000e+07	2.360457e+06	14.000000

To make the attributes of data easier to understand we make changes to it known as Label encoding which is a technique used to represent categorical variables as numerical variables so that machine learning models can use them as inputs.

Feature Engineering

In [16]:
```
from sklearn.preprocessing import LabelEncoder
labelEncoder = LabelEncoder()
df['fuel'] = labelEncoder.fit_transform(df['fuel'])
df['transmission'] = labelEncoder.fit_transform(df['transmission'])
df['owner'] = labelEncoder.fit_transform(df['owner'])
df['seller_type'] = labelEncoder.fit_transform(df['seller_type'])
```

In [17]:
```
df.dropna(inplace = True)
df.reset_index(inplace = True, drop = True)
df.drop(['name', 'torque'], inplace = True, axis = 1)
```

In [18]:
```
lst, lst1, lst2 = [], [], []
for i in range(0, 7906):
    lst.append(re.sub('[^0-9.]', '', str(df['mileage'][i])))
    lst1.append(re.sub('[^0-9.]', '', str(df['engine'][i])))
    lst2.append(re.sub('[^0-9.]', '', str(df['max_power'][i])))
new_lst = list(map(float, lst))
new_lst1 = list(map(float, lst1))
new_lst2 = list(map(float, lst2))
df['mileage'] = new_lst
df['engine'] = new_lst1
df['max_power'] = new_lst2
df.head()
```

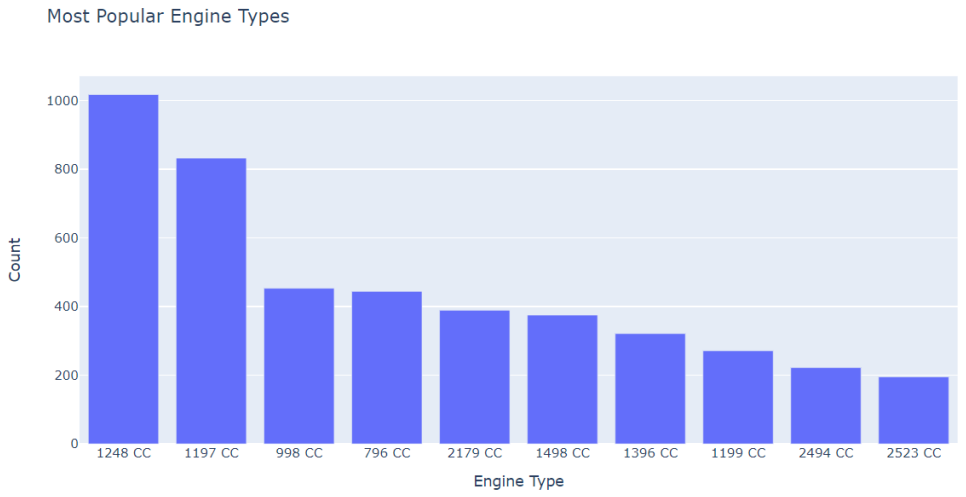
Out[18]:

	year	selling_price	km_driven	fuel	seller_type	transmission	owner	mileage	engine	max_power	seats
0	2014	450000	145500	1	1	1	0	23.40	1248.0	74.00	5.0
1	2014	370000	120000	1	1	1	2	21.14	1498.0	103.52	5.0
2	2006	158000	140000	3	1	1	4	17.70	1497.0	78.00	5.0
3	2010	225000	127000	1	1	1	0	23.00	1396.0	90.00	5.0
4	2007	130000	120000	3	1	1	0	16.10	1298.0	88.20	5.0

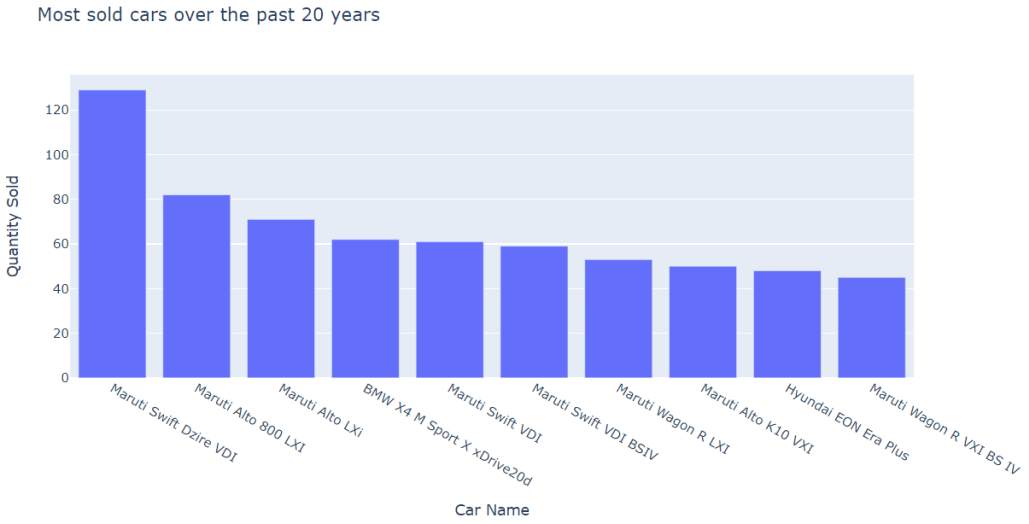
2. Visualization

Data visualization is used to make complex data easier to understand, identify relationships and correlations, and communicate insights and findings to others. It also makes data more engaging, which can encourage people to explore it further. Finally, data visualization supports decision-making by providing a clear, visual representation of the data that can help identify trends and patterns that might be missed in other forms of analysis.

In [12]:
```
engine = df.engine.value_counts()[:10]
px.bar(x = engine.index, y = engine, labels = {'x': 'Engine Type', 'y': 'Count'}, title = 'Most Popular Engine Types')
```



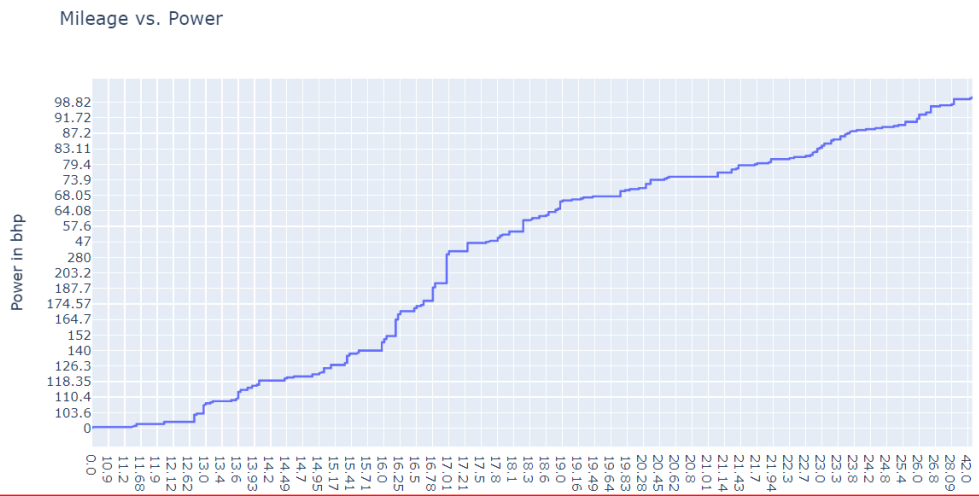
```
In [11]:
most_sold = df.name.value_counts()[:10]
px.bar(data_frame = most_sold, x = most_sold.index, y = most_sold, labels= {'index':'Car Name', 'y': 'Quantity Sold'},
       title = 'Most sold cars over the past 20 years')
```



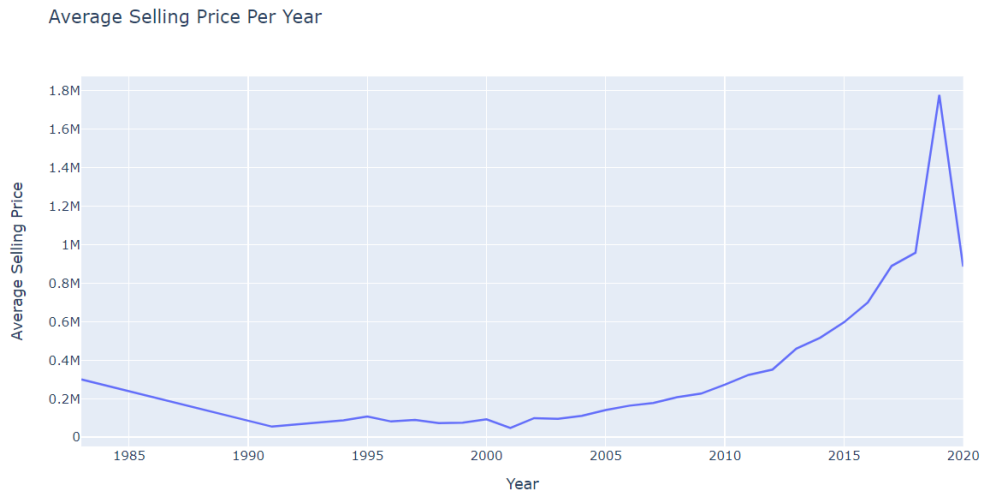
3. Geometric Analysis

Geometric analysis is used to study geometric objects and their properties such as shape, size, and position. It is used to provide a rigorous mathematical foundation for various areas such as physics, engineering, and computer science. Geometric analysis enables the development of powerful tools to solve complex problems in these fields.

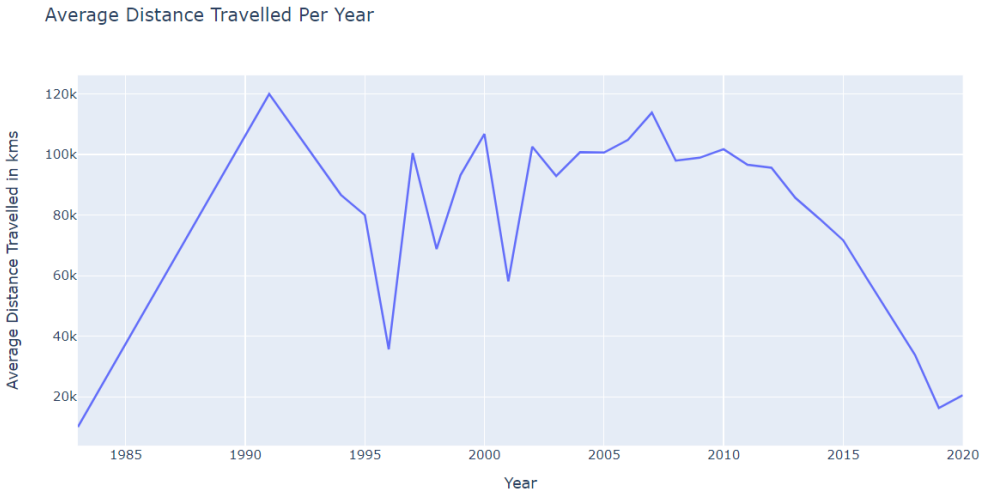
```
for i in range(0, 8128):
    temp = str(df['max_power'][i])
    temp = re.sub('[^0-9.]', '', temp)
    power.append(temp)
while('' in power) :
    power.remove('')
    power.sort()

power = power[:len(power)-5]
px.line(x = mileage, y = power, title = "Mileage vs. Power", labels = {'x': 'Mileage in kmpl', 'y': 'Power in bhp'})
```



In [14]:
```
data = df.groupby(['year']).mean()
px.line(data_frame = data, x = data.index, y = 'selling_price', labels = {'year': 'Year', 'selling_price': 'Average Selling
        title = 'Average Selling Price Per Year')
```



In [15]:
```
px.line(data_frame = data, x = data.index, y = 'km_driven', labels = {'year': 'Year', 'km_driven': 'Average Distance Travell
        title = 'Average Distance Travelled Per Year')
```

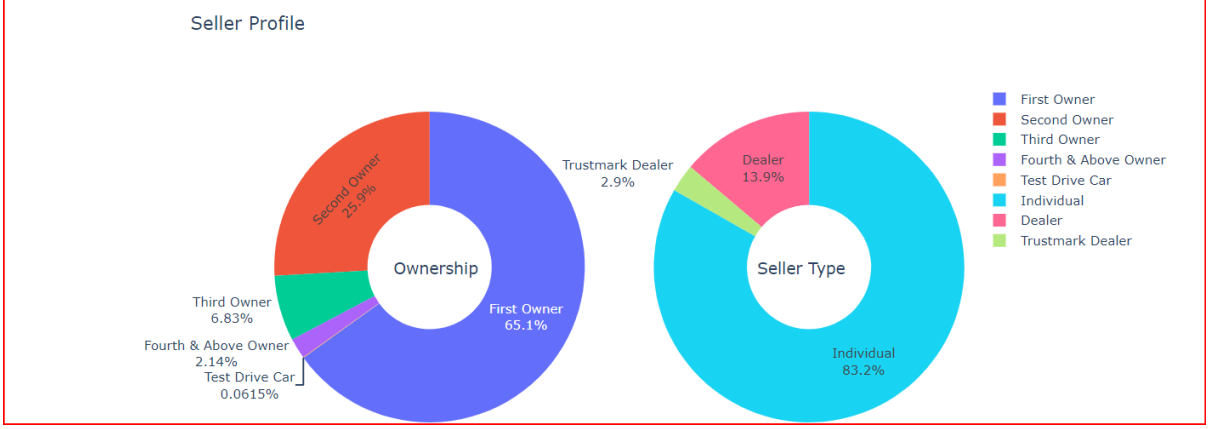


4. Psychographic Analysis

Psychographics helps in understanding consumer behaviour by analyzing their personality, values, interests, and lifestyle. It provides insights into the motivations and attitudes of the target audience, which can help marketers create more effective marketing strategies. By understanding the psychographics of their target audience, businesses can tailor their products and services to better meet customer needs and preferences.

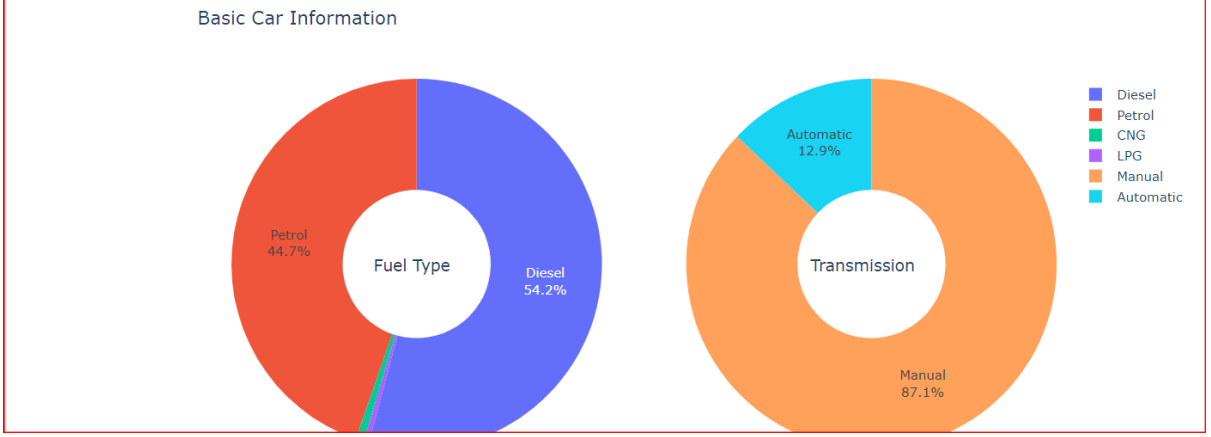
In [9]:

```
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
fig.add_trace(go.Pie(labels=df['owner'], name="Ownership", textinfo='label+percent'),
              1, 1)
fig.add_trace(go.Pie(labels=df['seller_type'], name="Seller Type",textinfo='label+percent'),
              1, 2)

fig.update_traces(hole=.4, hoverinfo="label+percent+name")

fig.update_layout(
    title_text="Seller Profile",
    annotations=[dict(text='Ownership', x=0.17, y=0.5, font_size=15, showarrow=False),
                 dict(text='Seller Type', x=0.83, y=0.5, font_size=15, showarrow=False)])
fig.show()
```



In [10]:

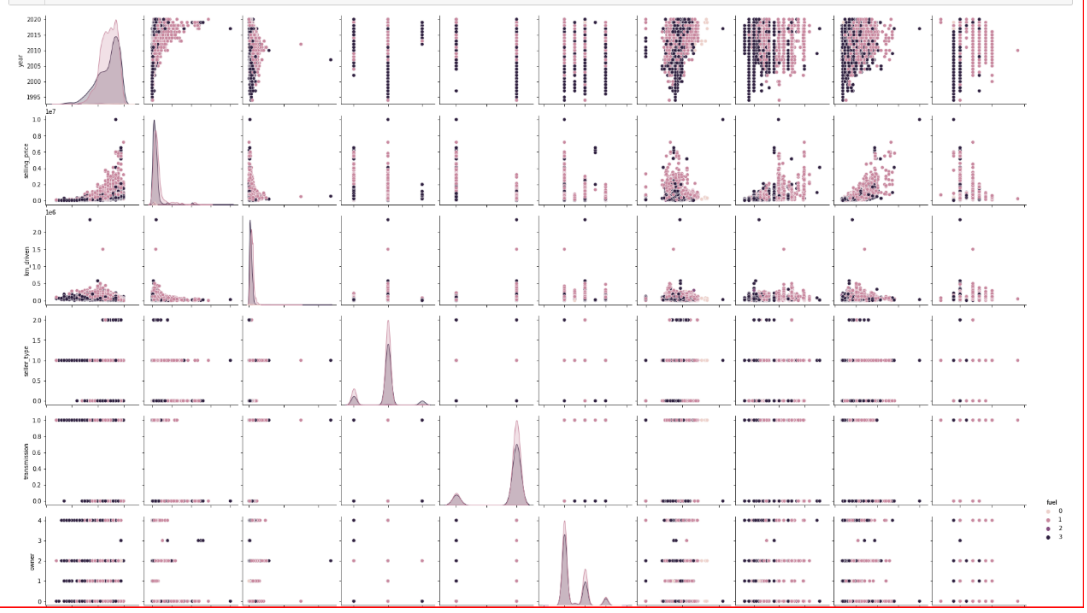
```
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
fig.add_trace(go.Pie(labels=df['fuel'], name="Fuel Type",textinfo='label+percent'),
              1, 1)
fig.add_trace(go.Pie(labels=df['transmission'], name="Transmission", textinfo='label+percent'),
              1, 2)

fig.update_traces(hole=.4, hoverinfo="label+percent+name")

fig.update_layout(
    title_text="Basic Car Information",

    annotations=[dict(text='Fuel Type', x=0.17, y=0.5, font_size=15, showarrow=False),
                 dict(text='Transmission', x=0.83, y=0.5, font_size=15, showarrow=False)])
fig.show()
```



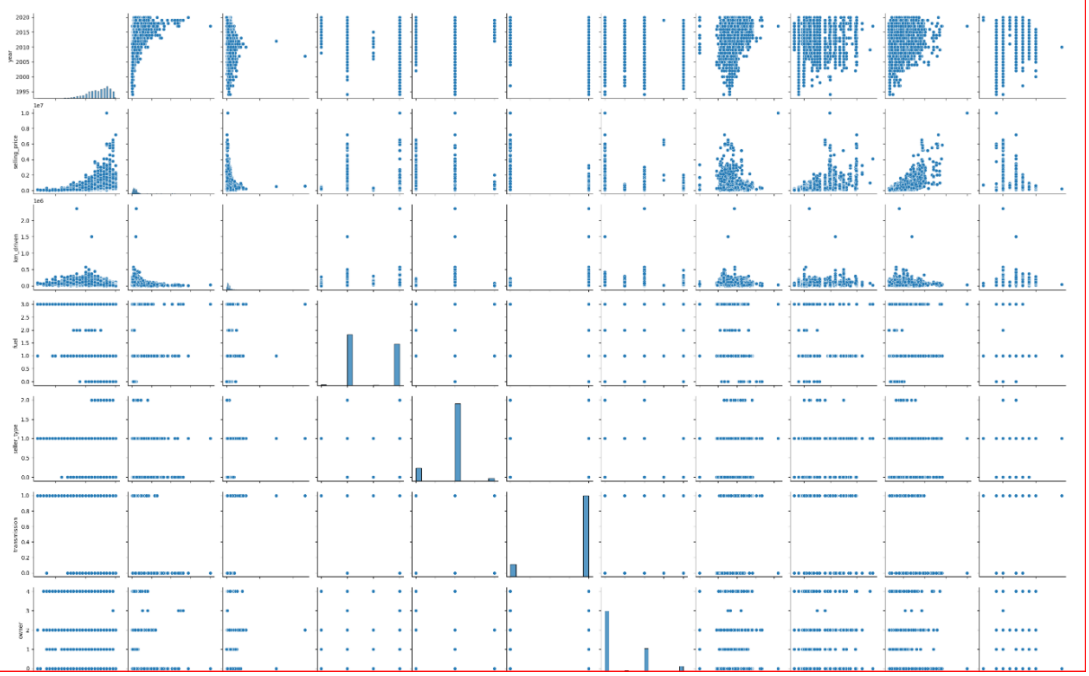
5. Demographic Analysis:

Demographic analysis helps in understanding the characteristics of a population, such as age, gender, income, and education. It provides insights into the preferences and behaviors of a particular group, which can help in developing effective marketing strategies. By understanding the demographic makeup of their target audience, businesses can tailor their products and services to better meet customer needs and preferences.

```
In [19]:
1 sns.pairplot(df, hue = 'fuel')
2 plt.show()
```



```
In [20]:
1 sns.pairplot(df)

Out[20]: <seaborn.axisgrid.PairGrid at 0x2493c063fa0>
```

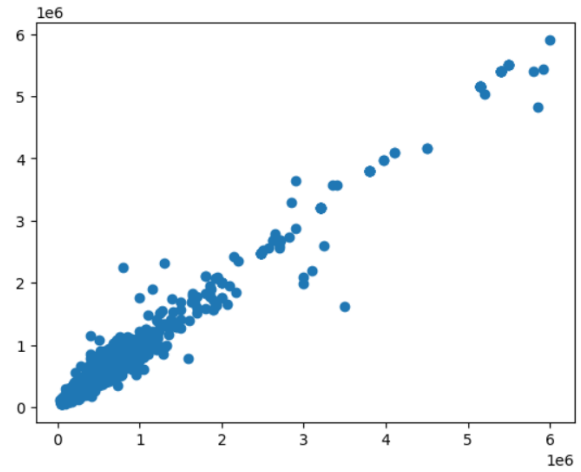


6. Behaviour Analysis:

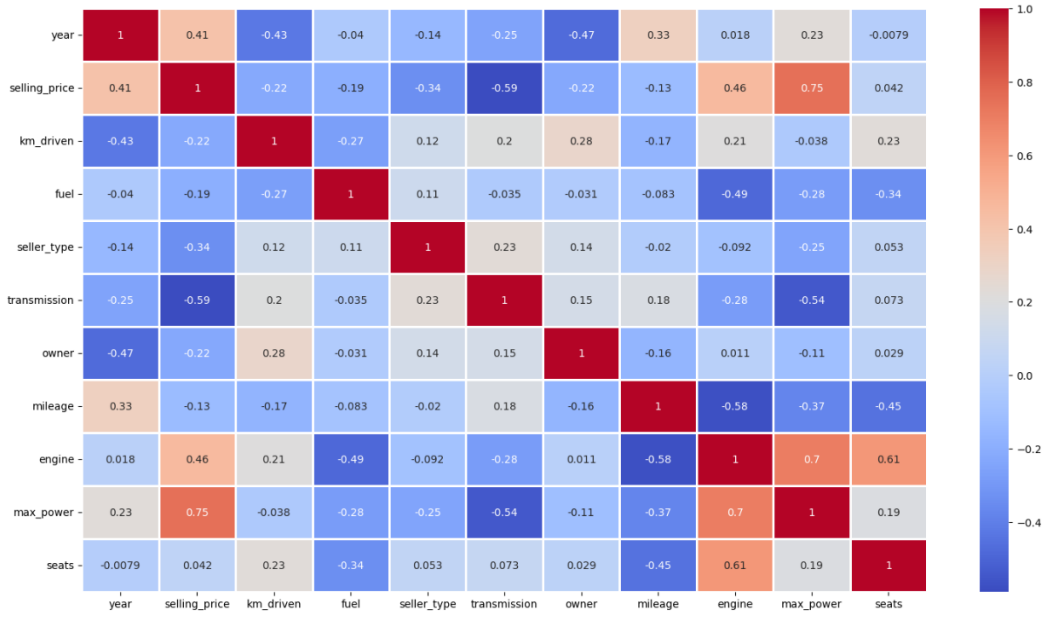
Behaviour analysis helps in understanding the actions and choices made by individuals, providing insights into their preferences and motivations. It helps businesses identify the factors that influence consumer behaviour and develop effective marketing strategies. By understanding consumer behaviour, businesses can improve their products and services, enhance customer satisfaction, and increase profitability.

In [37]:
```
plt.scatter(y_test,predictions)
```

Out[37]: <matplotlib.collections.PathCollection at 0x24946ade220>

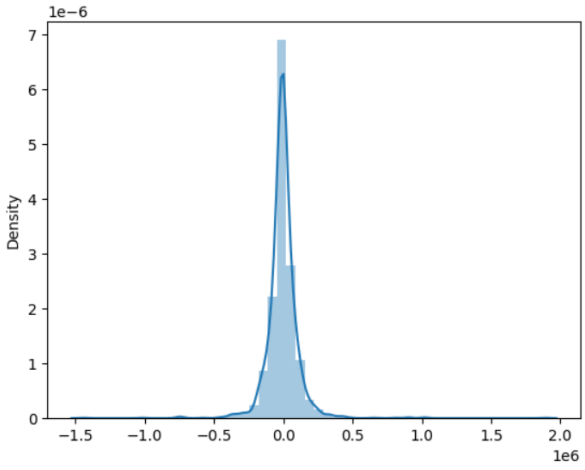


In [21]:
```
plt.figure(figsize=(18, 10))
sns.heatmap(df.corr(), linecolor = 'white', linewidths = 1, cmap = 'coolwarm', annot=True)
plt.show()
```



In [36]:
```
sns.distplot(y_test-predictions)
```

Out[36]: <Axes: ylabel='Density'>



Market Segmentation Analysis
On
EV Vehicles

By Sanagiri Durga Bhavani

Githublink : https://github.com/Durga-s-02/EV

Market Segmentation Analysis:

Market segmentation analysis is the study of customers, divided into smaller groups, to understand their specific characteristics like behavior, age, income, and personality. It's easier for companies to advertise when they're marketing a smaller segment of customers; this way, each campaign can be highly targeted and precise to the characteristics of each group.

EV Vehicles:

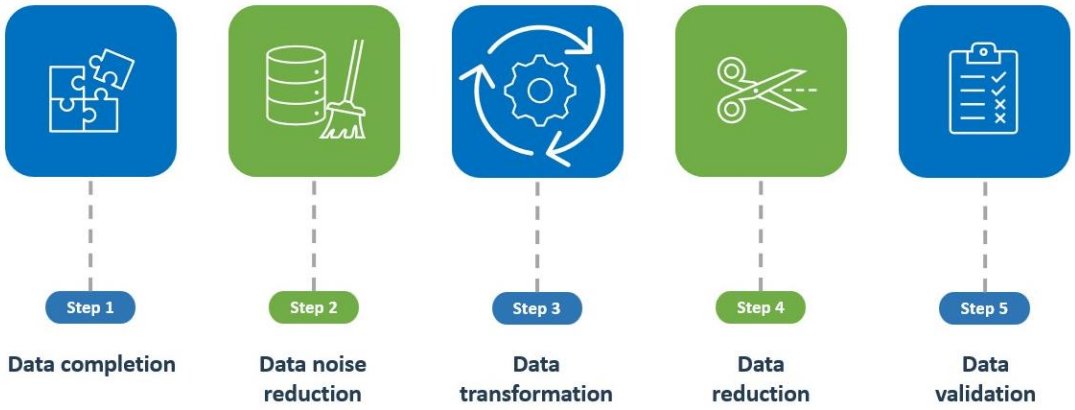
An electric vehicle is a vehicle that uses one or more electric motors for propulsion. It can be powered by a collector system, with electricity from extravehicular sources, or it can be powered autonomously by a battery.

Buying an electric car in India comes with several benefits, including lower running costs, environmental benefits, government subsidies, electric vehicle insurance benefits, and reduced dependence on foreign oil.

Data Collection – Data Preprocessing

Data preprocessing transforms the data into a format that is more easily and effectively processed in data mining, machine learning and other data science tasks. The techniques are generally used at the earliest stages of the machine learning and AI development pipeline to ensure accurate results.

Steps for data preprocessing




```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

+ Code + Text

```
df = pd.read_csv("/content/ElectricCarData_clean.csv")
df
```

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	RapidCharge	PowerTrain	PlugType	BodyStyle	Segment	Seats	PriceEuro
0	Tesla	Model 3 Long Range Dual Motor	4.6	233	450	161	940	Yes	AWD	Type 2 CCS	Sedan	D	5	55480
1	Volkswagen	ID.3 Pure	10.0	160	270	167	250	Yes	RWD	Type 2 CCS	Hatchback	C	5	30000
2	Polestar	2	4.7	210	400	181	620	Yes	AWD	Type 2 CCS	Liftback	D	5	56440
3	BMW	iX3	6.8	180	360	206	560	Yes	RWD	Type 2 CCS	SUV	D	5	68040
4	Honda	e	9.5	145	170	168	190	Yes	RWD	Type 2 CCS	Hatchback	B	4	32997
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
98	Nissan	Ariya 63kWh	7.5	160	330	191	440	Yes	FWD	Type 2 CCS	Hatchback	C	5	45000
99	Audi	e-tron S Sportback 55 quattro	4.5	210	335	258	540	Yes	AWD	Type 2 CCS	SUV	E	5	96050

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Brand            103 non-null    object
 1   Model            103 non-null    object
 2   AccelSec         103 non-null    float64
 3   TopSpeed_KmH     103 non-null    int64
 4   Range_Km         103 non-null    int64
 5   Efficiency_WhKm  103 non-null    int64
 6   FastCharge_KmH   103 non-null    object
 7   RapidCharge      103 non-null    object
 8   PowerTrain       103 non-null    object
 9   PlugType         103 non-null    object
 10  BodyStyle        103 non-null    object
 11  Segment          103 non-null    object
 12  Seats            103 non-null    int64
 13  PriceEuro        103 non-null    int64
dtypes: float64(1), int64(5), object(8)
memory usage: 11.4+ KB
```

```
df.describe()
```

	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	Seats	PriceEuro
count	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000
mean	7.396117	179.194175	338.786408	189.165049	4.883495	55811.563107
std	3.017430	43.573030	126.014444	29.566839	0.795834	34134.665280
min	2.100000	123.000000	95.000000	104.000000	2.000000	20129.000000
25%	5.100000	150.000000	250.000000	168.000000	5.000000	34429.500000
50%	7.300000	160.000000	340.000000	180.000000	5.000000	45000.000000
75%	9.000000	200.000000	400.000000	203.000000	5.000000	65000.000000
max	22.400000	410.000000	970.000000	273.000000	7.000000	215000.000000

```
df.isnull().sum()
```

```
Brand              0
Model              0
AccelSec           0
TopSpeed_KmH       0
Range_Km           0
Efficiency_WhKm    0
FastCharge_KmH     0
RapidCharge        0
PowerTrain         0
PlugType           0
BodyStyle          0
Segment            0
Seats              0
PriceEuro          0
dtype: int64
```

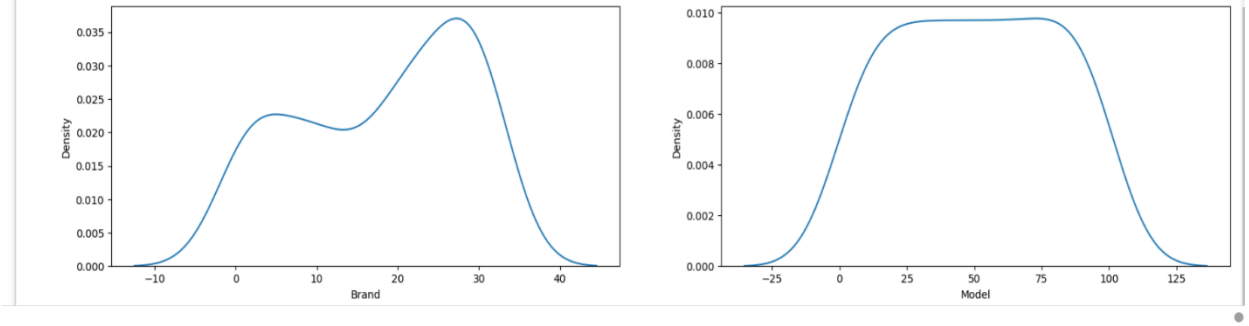
```
#cheking for duplicate rows in the dataset
df.duplicated().sum()
```

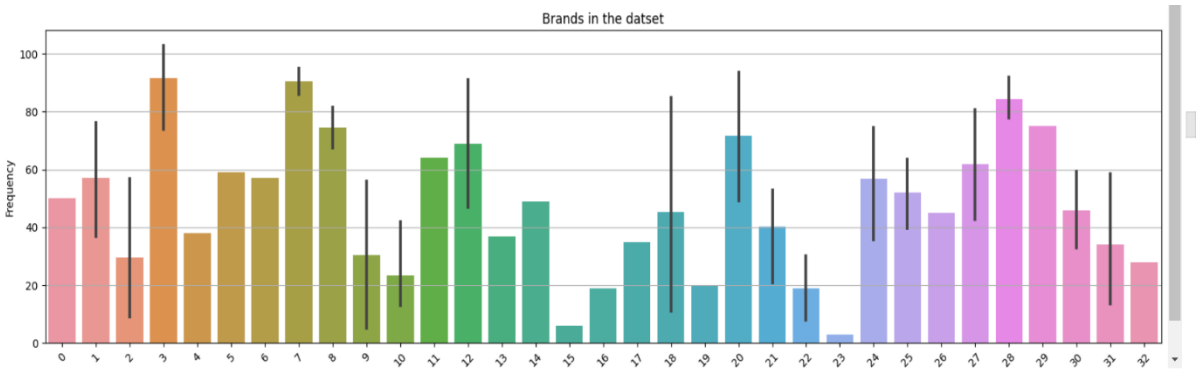
```
0
```

Data Visualisation

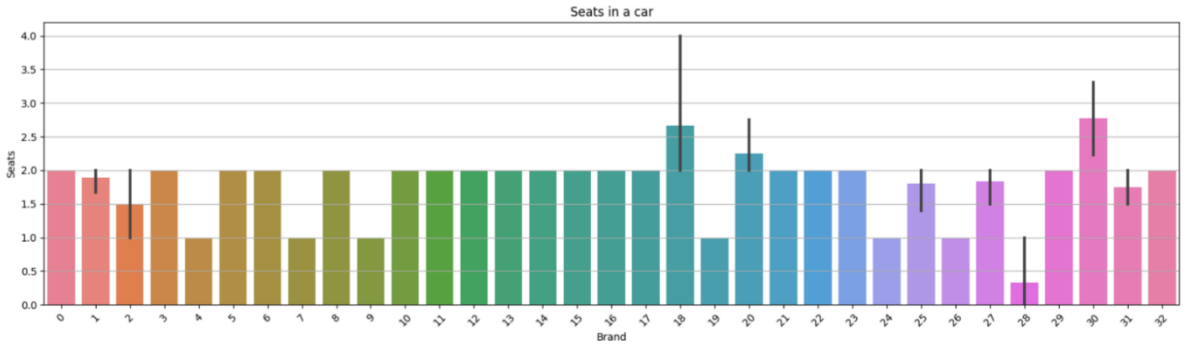
Data visualization is the representation of data through use of common graphics, such as charts, plots, infographics, and even animations. These visual displays of information communicate complex data relationships and data-driven insights in a way that is easy to understand.

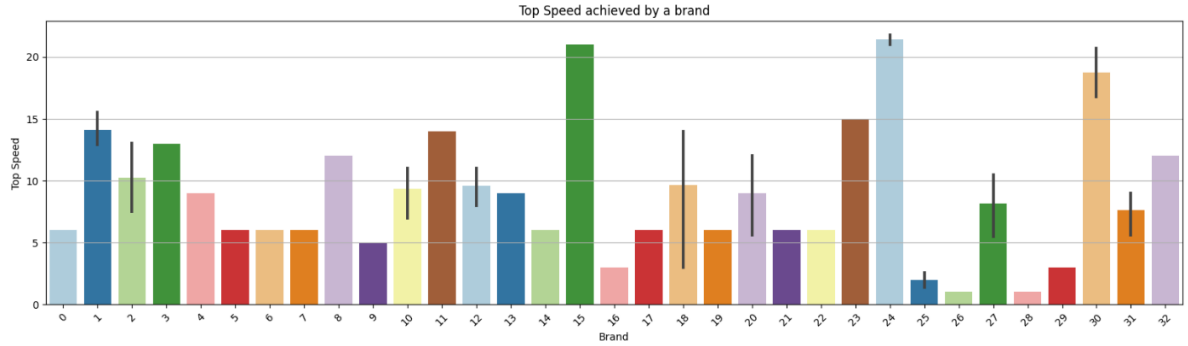
```
#Data Visualization
plt.figure(figsize=(20,35))
for i, col in enumerate (df.columns):
  if df[col].dtype != 'object':
    ax = plt.subplot(7, 2, i+1)
    sns.kdeplot(df[col], ax=ax)
    plt.xlabel(col)

plt.show()
```





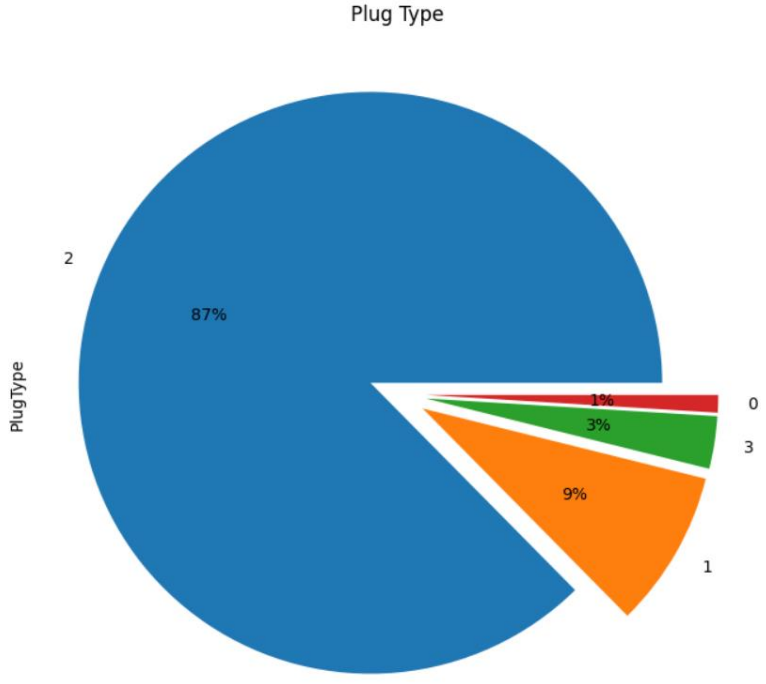

```
#Number of seats in each car
```





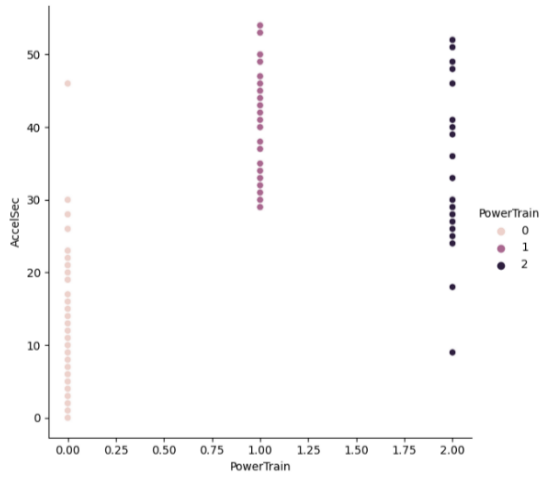

```
Text(0.5, 1.0, 'Plug Type')
```

[ ]



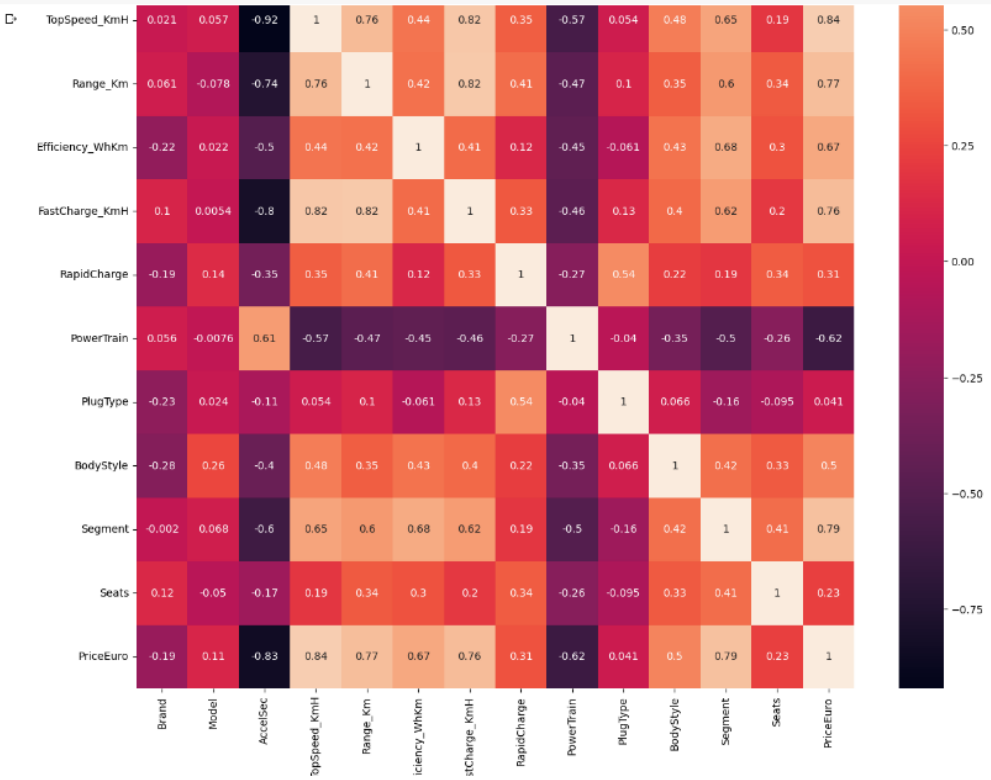
```
[ ] sns.relplot(x="PowerTrain", y="AccelSec", height=6,hue="PowerTrain",data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7c390f46e740>
```



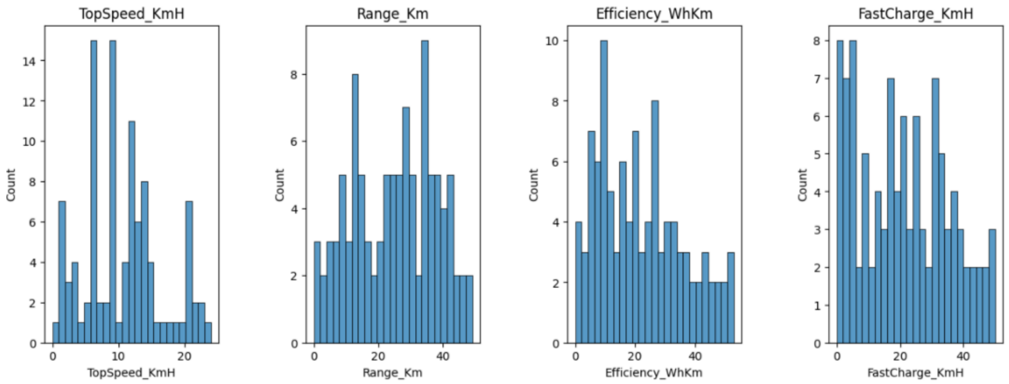

```
df.corr()
```

```
plt.figure(figsize=(15,15))
sns.heatmap(df.corr(), annot=True)
plt.show()
```




```
[ ] plt.figure(1, figsize=(15,5))
```

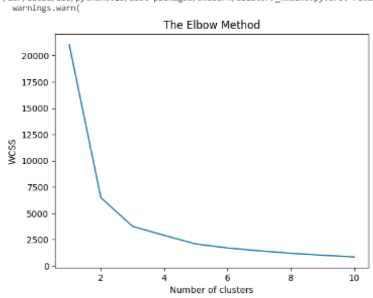
```
[ ] plt.figure(1, figsize=(15,5))
    n=0

    for x in ['TopSpeed_KmH', 'Range_Km' ,'Efficiency_WhKm'  ,'FastCharge_KmH']:
      n += 1
      plt.subplot(1,4,n)
      plt.subplots_adjust(hspace=0.5, wspace=0.5)
      sns.histplot(df[x], bins= 25)
      plt.title(f'{x}')
    plt.show()
```



Segment Extraction

In the Elbow method, we are actually varying the number of clusters (K) from 1 – 10. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of the squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow. As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph, we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph moves almost parallel to the X-axis. The K value corresponding to this point is the optimal value of K or an optimal number of clusters.

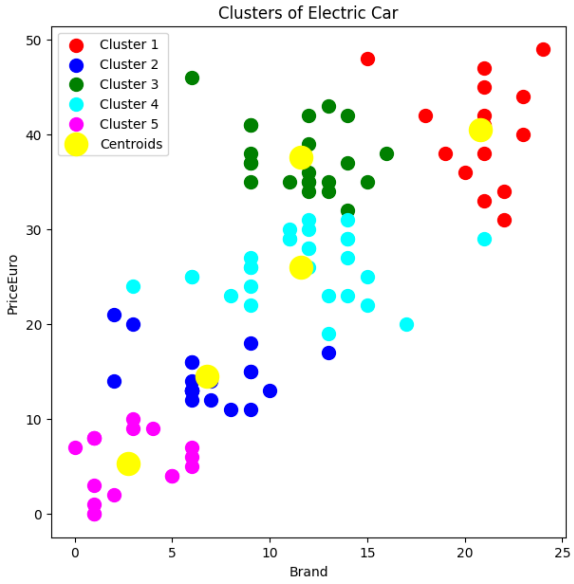
```
[ ] #K-Means Clustering
    # Using the elbow method to find the optimal number of clusters
    X = df.iloc[:, [3, 4]].values
    from sklearn.cluster import KMeans
    wcss = []
    for i in range(1, 11):
        kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
        kmeans.fit(X)
        wcss.append(kmeans.inertia_)
    plt.plot(range(1, 11), wcss)
    plt.title('The Elbow Method')
    plt.xlabel('Number of clusters')
    plt.ylabel('WCSS')
    plt.show()

    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
    /usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
      warnings.warn(
```



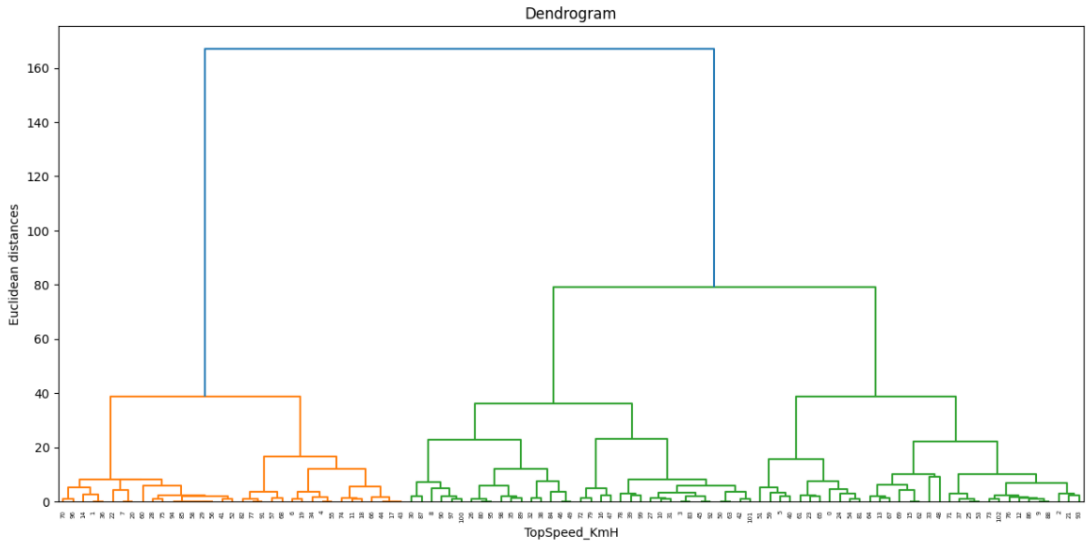

```
# Fitting K-Means to the dataset
```

```
[ ]  plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
     plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids')
     plt.title('Clusters of Electric Car')
     plt.xlabel('Brand')
     plt.ylabel('PriceEuro')
     plt.legend()
     plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will char
  warnings.warn(
```




```
#Hierarchical Clustering
# Using the dendrogram to find the optimal number of clusters for TopSpeed of Electric Car
plt.figure(figsize=(15,7))
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('TopSpeed_KmH')
plt.ylabel('Euclidean distances')
plt.show()
```

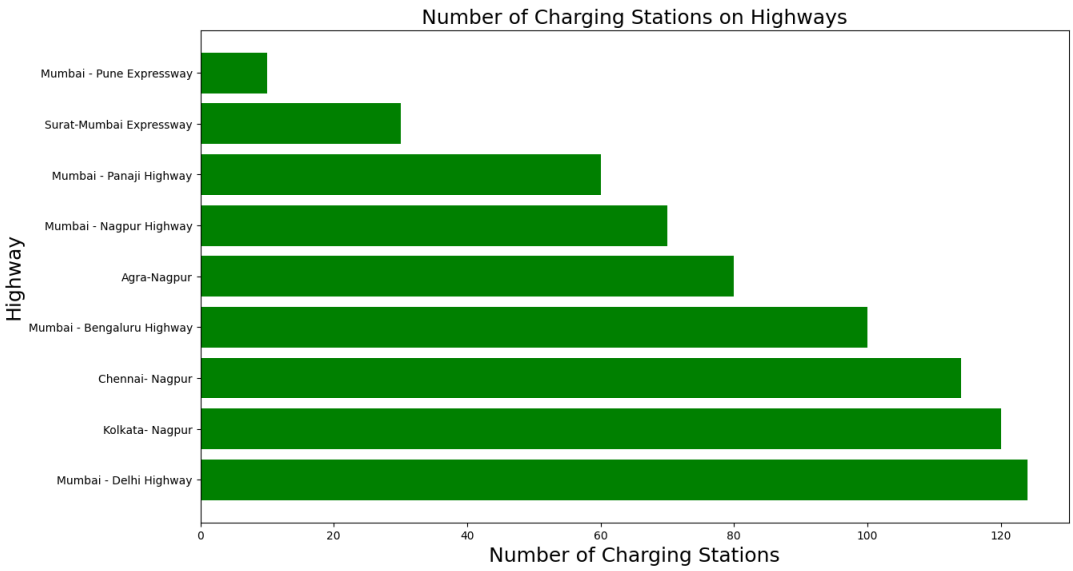


Geographic Segmentation

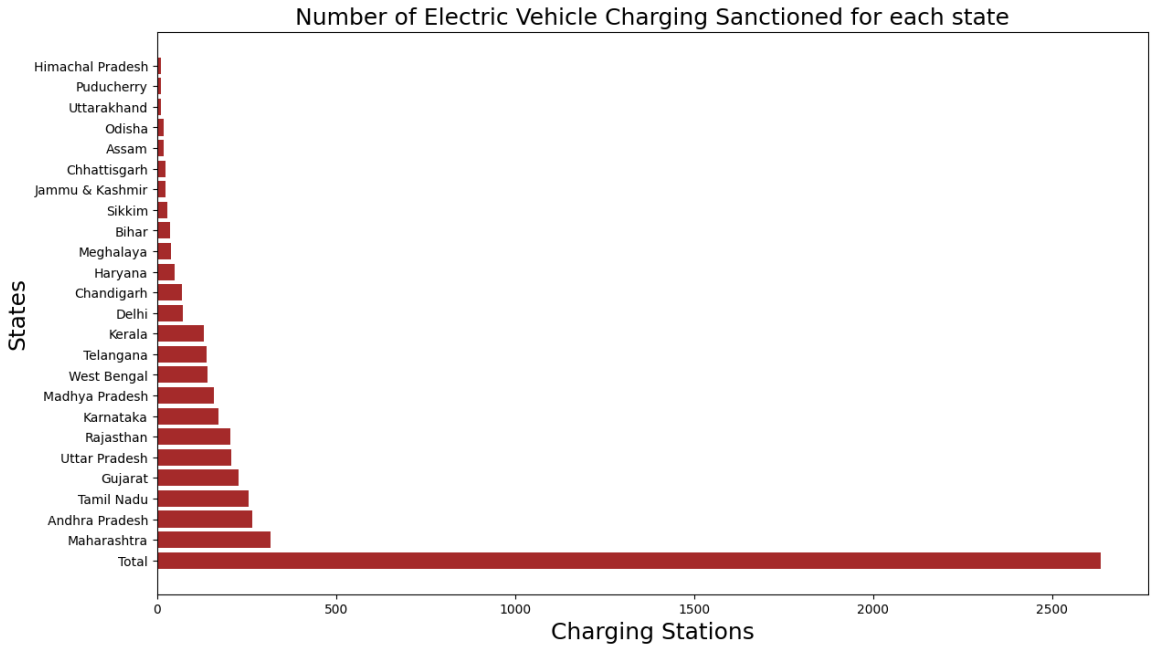
Geographic segmentation is a marketing strategy used to target products or services at people who live in, or shop at, a particular location. It works on the principle that people in that location have similar needs, wants, and cultural considerations.

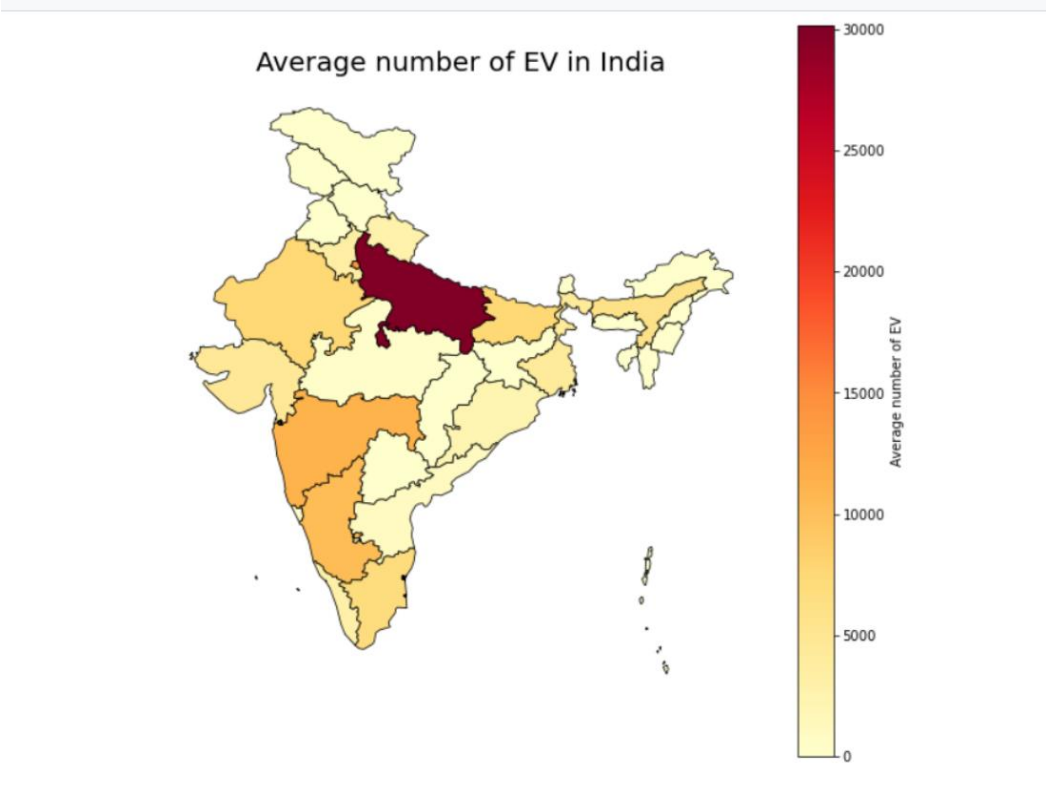
Geographic segmentation organizes your audience into groups based on their physical location, such as country or postal code. This type of segmentation can provide valuable insights into the buying trends and preferences of different regions, allowing for more targeted and effective marketing efforts.

```python
#Charging stations on highway
def plot_frequency(data,feature1,feature2,x_lbl,y_lbl,text,color):
    new_df = data
    new = new_df.sort_values(by=[feature2],ascending=False) # sorting in descending order
    plt.figure(figsize=(14, 8))
    x_state = list(new[feature1].values) # defining x axis
    y_state = list(new[feature2].values) # defining y axis (count of numericl entity)
    plt.barh(x_state,y_state, color = color)
    plt.xlabel(x_lbl, fontsize=18) # xlabel
    plt.ylabel(y_lbl, fontsize=18) # ylabel
    plt.title(text, fontsize = 18)
    return plt.show()
plot_frequency(cs_highway,"Highways/Expressways","Charging Stations", "Number of Charging Stations",
               "Highway","Number of Charging Stations on Highways",'green')
```



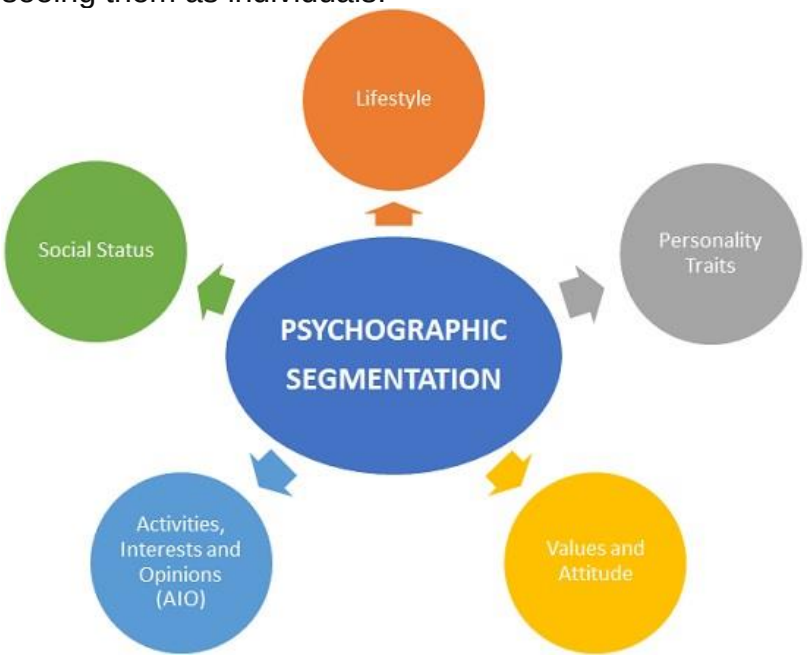

```python
#Sanctioned Charging Stations
plot_frequency(cs_sanctioned,'State/UT-wise','Number of Electric Vehicle Charging Sanctioned','Charging Stations','States',
               'Number of Electric Vehicle Charging Sanctioned for each state','brown')
```





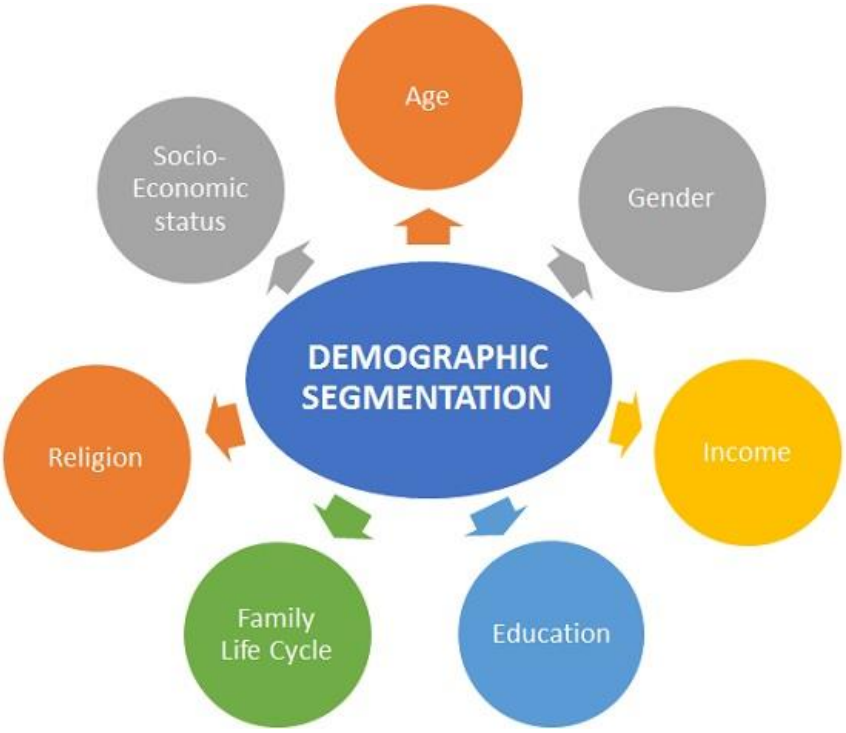
Psychographic Segmentation

Psychographic segmentation divides buyers into different segments based on internal characteristics—personality, values, beliefs, lifestyle, attitudes, interests, and social class—so you can market accordingly. It requires looking beyond customers as they pertain to your brand and seeing them as individuals.



Demographic Segmentation

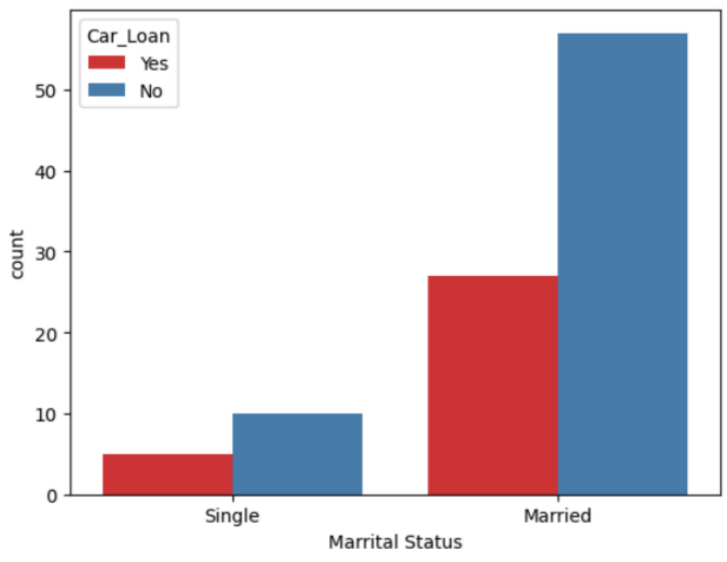
Demographic segmentation is a precise form of audience identification based on data points like age, gender, marital status, family size, income, education, race, occupation, nationality, and/or religion.



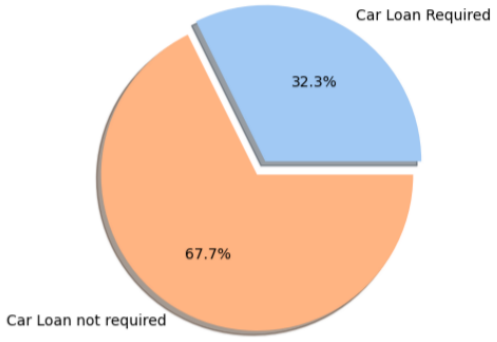
Behavioural Segmentation

Behavioral segmentation is the process of grouping customers according to their behavior when making purchasing decisions. Market researchers observe aspects such as readiness to buy, i.e., the knowledge they have about the product, level of loyalty, interactions with your brand or product usage experience, etc.

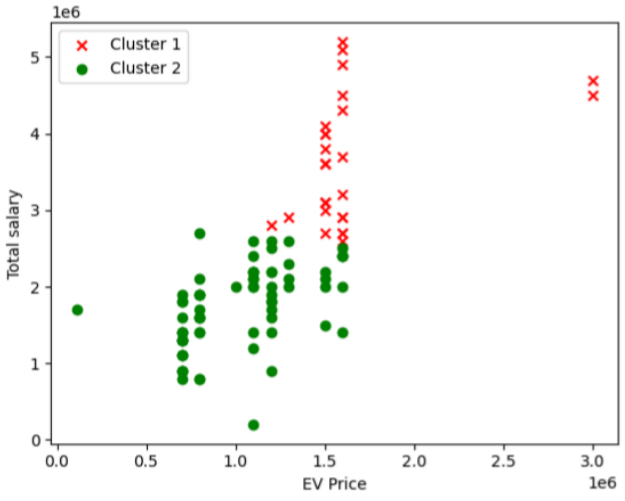
```
sns.countplot(x ='Marrital Status', hue = 'Car_Loan',  data = data, palette = 'Set1')
plt.show()
```



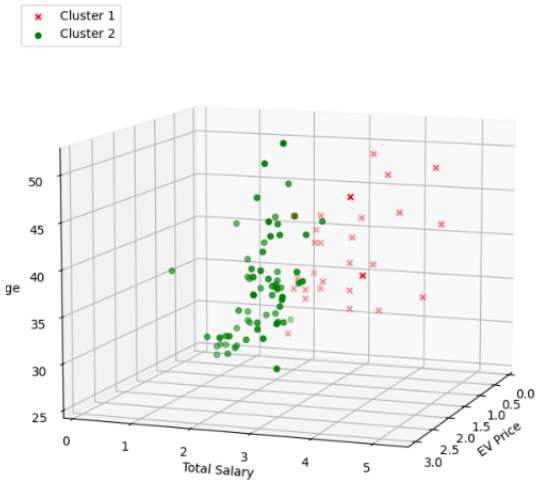
```
#Getting labels and data
labels = ['Car Loan Required','Car Loan not required']
Loan_status = [data.query('Car_Loan == "Yes"').Car_Loan.count(),data.query('Car_Loan == "No"').Car_Loan.count()]

# declaring exploding pie
explode = [0.1, 0]
# define Seaborn color palette to use
palette_color = sns.color_palette('pastel')

# plotting data on chart
plt.pie(Loan_status, labels=labels, colors=palette_color, shadow = "True",
        explode=explode, autopct='%1.1f%%')

# displaying chart
plt.show()
```



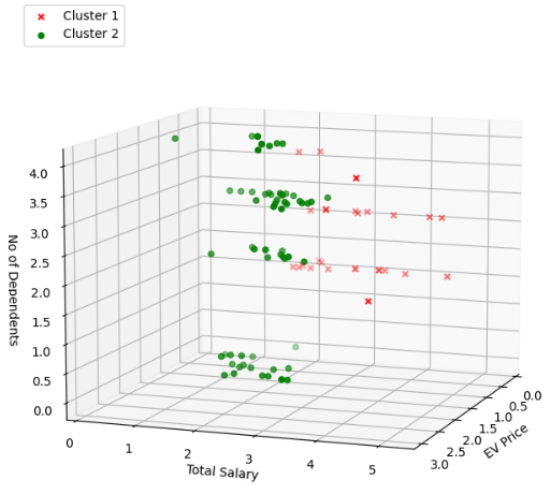

```
[24] # plotting the effct of salary and ev price on cluster data

     plt.scatter(Cluster_0.EV_Price, Cluster_0['Total Salary'],color='red', marker = 'x', label = 'Cluster 1')
     plt.scatter(Cluster_1.EV_Price, Cluster_1['Total Salary'],color='green', label = 'Cluster 2')
     plt.legend(loc="upper left")

     plt.xlabel('EV Price')
     plt.ylabel('Total salary')
     plt.show()

     # there is a clear difference in segments when comparing salary and the price of EV purchased
```



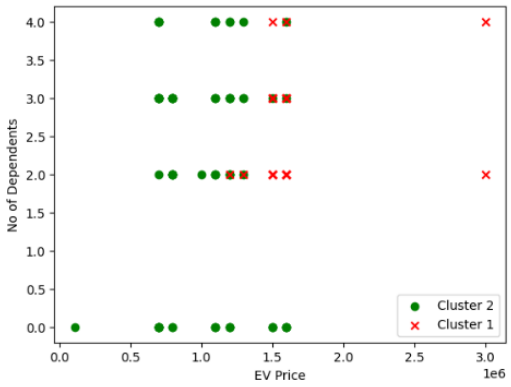
```
from mpl_toolkits.mplot3d import Axes3D
# plotting influence of age

fig = plt.figure(figsize=(8,8))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(Cluster_0.EV_Price, Cluster_0['Total Salary'], Cluster_0['Age'], color='red', marker = 'x', label = 'Cluster 1')
ax.scatter(Cluster_1.EV_Price, Cluster_1['Total Salary'],Cluster_1['Age'], color='green', label = 'Cluster 2')
plt.legend(loc = 'upper left')

ax.view_init(10, 20)

plt.xlabel("EV Price")
plt.ylabel("Total Salary")
ax.set_zlabel('Age')
plt.show()
```




```
[27] # plotting influence of No of Dependents

    fig = plt.figure(figsize=(8,8))

    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(Cluster_0.EV_Price, Cluster_0['Total Salary'], Cluster_0['No of Dependents'], color='red', marker = 'x', label = 'Cluster 1')
    ax.scatter(Cluster_1.EV_Price, Cluster_1['Total Salary'],Cluster_1['No of Dependents'], color='green', label = 'Cluster 2')
    plt.legend(loc = 'upper left')
    ax.view_init(10,20)

    plt.xlabel("EV Price")
    plt.ylabel("Total Salary")
    ax.set_zlabel('No of Dependents')
    plt.show()
```



```
[29] # plotting the effct of no of dependents and ev price on cluster data


     plt.scatter(Cluster_1.EV_Price, Cluster_1['No of Dependents'],color='green', label = 'Cluster 2')
     plt.scatter(Cluster_0.EV_Price, Cluster_0['No of Dependents'],color='red', marker = 'x', label = 'Cluster 1')
     plt.legend(loc="lower right")

     plt.xlabel('EV Price')
     plt.ylabel('No of Dependents')
     plt.show()

     # there is a clear difference in segments when comparing salary and the price of EV purchased
```



MARKET SEGMENTATION ANALYSIS OF EV

- Shruthi Nanditha Ganesh

Dataset-
https://github.com/shruthi1210/marketanalysisofEV/blob/main/3_ev_market_india_dataset.xlsx

Project-

https://github.com/shruthi1210/marketanalysisof

EV-Data Pre processing

Data preprocessing, a component of data preparation, describes any type of

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.decomposition import PCA
        from sklearn.cluster import KMeans
```

```
In [15]: df = pd.read_excel(r"C:\Users\shruthi\Desktop\fenny labs\3_ev_market_india_dataset.xlsx")
         df.head()
```

Out[15]:

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	RapidCharge	PowerTrain	PlugType	BodyStyle	Segment	S
0	Tesla	Model 3 Long Range Dual Motor	4.6	233	450	161	940	Yes	AWD	Type 2 CCS	Sedan	D	
1	Volkswagen	ID.3 Pure	10.0	160	270	167	250	No	RWD	Type 2 CCS	Hatchback	C	
2	Polestar	2	4.7	210	400	181	620	Yes	AWD	Type 2 CCS	Liftback	D	
3	BMW	iX3	6.8	180	360	206	560	Yes	RWD	Type 2 CCS	SUV	D	
4	Honda	e	9.5	145	170	168	190	Yes	RWD	Type 2 CCS	Hatchback	B	

processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process.

```
In [16]: d= df.describe(include="all")
         display(d)
```

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	RapidCharge	PowerTrain	PlugType	BodyStyle	Segment
count	103	103	103.000000	103.000000	103.000000	103.000000	103.000000	103	103	103	103	103
unique	33	102	NaN	NaN	NaN	NaN	NaN	2	3	4	9	8
top	Tesla	e-Soul 64 kWh	NaN	NaN	NaN	NaN	NaN	Yes	AWD	Type 2 CCS	SUV	C
freq	13	2	NaN	NaN	NaN	NaN	NaN	77	41	90	45	30
mean	NaN	NaN	7.396117	179.194175	338.786408	189.165049	444.271845	NaN	NaN	NaN	NaN	NaN
std	NaN	NaN	3.017430	43.573030	126.014444	29.566839	203.949253	NaN	NaN	NaN	NaN	NaN
min	NaN	NaN	2.100000	123.000000	95.000000	104.000000	170.000000	NaN	NaN	NaN	NaN	NaN
25%	NaN	NaN	5.100000	150.000000	250.000000	168.000000	260.000000	NaN	NaN	NaN	NaN	NaN
50%	NaN	NaN	7.300000	160.000000	340.000000	180.000000	440.000000	NaN	NaN	NaN	NaN	NaN
75%	NaN	NaN	9.000000	200.000000	400.000000	203.000000	555.000000	NaN	NaN	NaN	NaN	NaN
max	NaN	NaN	22.400000	410.000000	970.000000	273.000000	940.000000	NaN	NaN	NaN	NaN	NaN

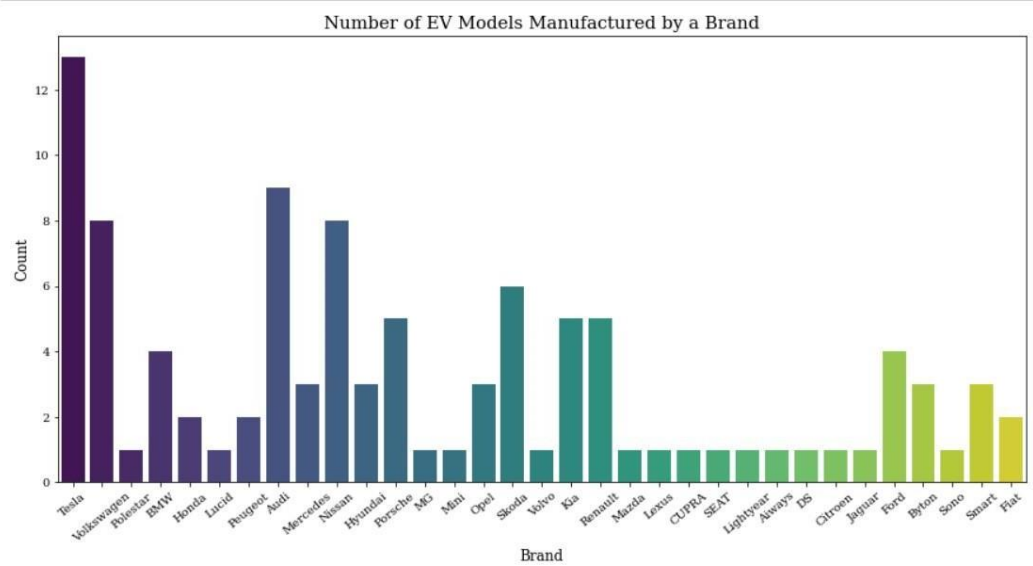
```
In [17]: print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Brand            103 non-null    object
 1   Model            103 non-null    object
 2   AccelSec         103 non-null    float64
 3   TopSpeed_KmH     103 non-null    int64
 4   Range_Km         103 non-null    int64
 5   Efficiency_WhKm  103 non-null    int64
 6   FastCharge_KmH   103 non-null    int64
 7   RapidCharge      103 non-null    object
 8   PowerTrain       103 non-null    object
 9   PlugType         103 non-null    object
 10  BodyStyle        103 non-null    object
 11  Segment          103 non-null    object
 12  Seats            103 non-null    int64
 13  PriceEuro        103 non-null    int64
dtypes: float64(1), int64(6), object(7)
memory usage: 11.4+ KB
None
```

Exploratory Data Analysis

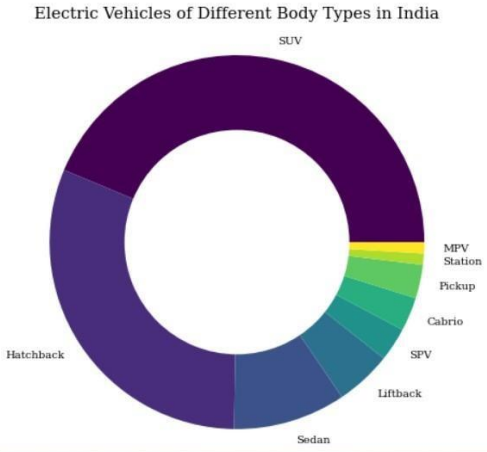
Exploratory Data Analysis, popularly abbreviated as EDA, is one of the most important steps in the data science pipeline. It is the process of gaining the information present inside the data with the help of summary statistics and visual representations. Keys features of this technique are presented in the below image.

```
In [19]: # brand-wise count of EV models
sns.catplot(data=df, x='Brand', kind='count', palette='viridis', height=6, aspect=2)
sns.despine(right=False, top=False)
plt.tick_params(axis='x', rotation=40)
plt.xlabel('Brand',family='serif', size=12)
plt.ylabel('Count', family='serif', size=12)
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.title('Number of EV Models Manufactured by a Brand', family='serif', size=15)
plt.show()
```



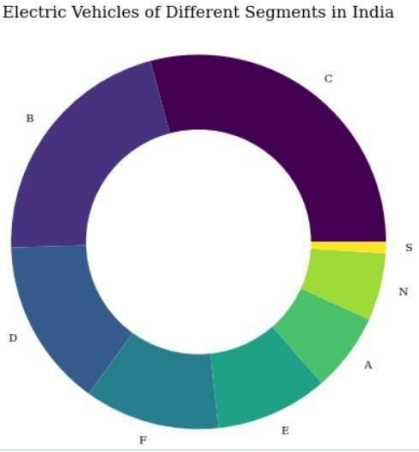
Anaysis of different body types of EVs
Observation: SUV and Hatchback body types form the majority while Station and MPV the minority

In [20]:
```
x = df['BodyStyle'].value_counts().plot.pie(radius=2, cmap='viridis', startangle=0, textprops=dict(family='serif'))
plt.pie(x=[1], radius=1.2, colors='white')
plt.title(label='Electric Vehicles of Different Body Types in India', family='serif', size=15, pad=100)
plt.ylabel('')
plt.show()
```



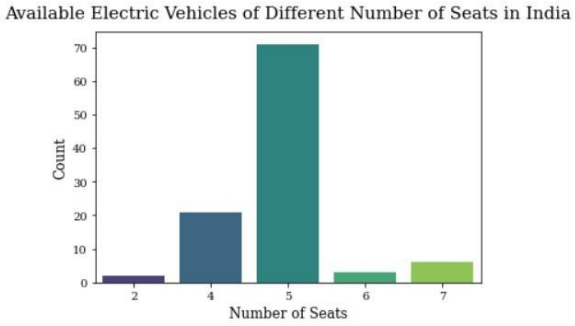
Analysis of different segments of EVs
Observation: B and C body segments form the majority while S and A the minority.

In [21]:
```
x = df['Segment'].value_counts().plot.pie(radius=2, cmap='viridis', startangle=0, textprops=dict(family='serif'), pctdistance=.5)
plt.pie(x=[1], radius=1.2, colors='white')
plt.title(label='Electric Vehicles of Different Segments in India', family='serif', size=15, pad=100)
plt.ylabel('')
plt.show()
```



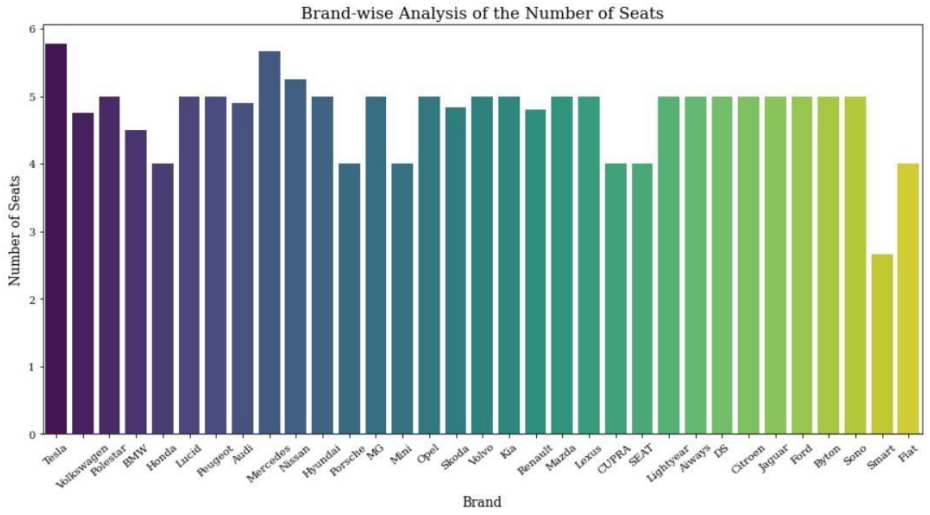
In [ ]:
```
#Analysis of EVs of different number of seats
#Observation: EVs with 5 sitters dominate the market while EVs with 2 sitters are less in number.
```

In [23]:
```
sns.countplot(data=df, x='Seats', palette='viridis')
plt.xlabel('Number of Seats', family='serif', size=12)
plt.ylabel('Count', family='serif', size=12)
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.title(label='Available Electric Vehicles of Different Number of Seats in India', family='serif', size=15, pad=12)
plt.show()
```



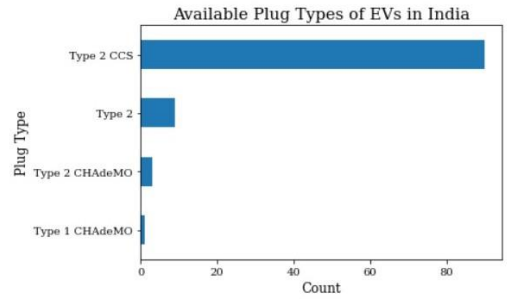
In [ ]:
```
#Analysis of the number of seats by each brand
#Observation: Based on the number of seats, Tesla, Mercedes and Nissan have the maximum number of seats and Smart the minimum.
```

In [24]:
```
sns.catplot(kind='bar', data=df, x='Brand', y='Seats', palette='viridis', ci=None, height=6, aspect=2)
sns.despine(right=False, top=False)
plt.tick_params(axis='x', rotation=40)
plt.xlabel('Brand',family='serif', size=12)
plt.ylabel('Number of Seats', family='serif', size=12)
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.title('Brand-wise Analysis of the Number of Seats', family='serif', size=15);
```

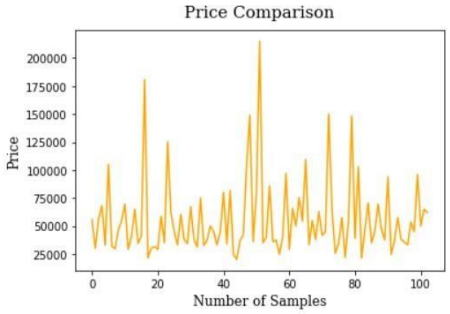


In [ ]:
```
#Analysis of different plug types
#Observation: EVs with plus type of 'Type 2 CCS' seem to dominate the market.
```

In [25]:
```
df['PlugType'].value_counts().sort_values(ascending=True).plot.barh()
plt.xlabel('Count', family='serif', size=12)
plt.ylabel('Plug Type', family='serif', size=12)
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.title('Available Plug Types of EVs in India', family='serif', size=15)
plt.show()
```

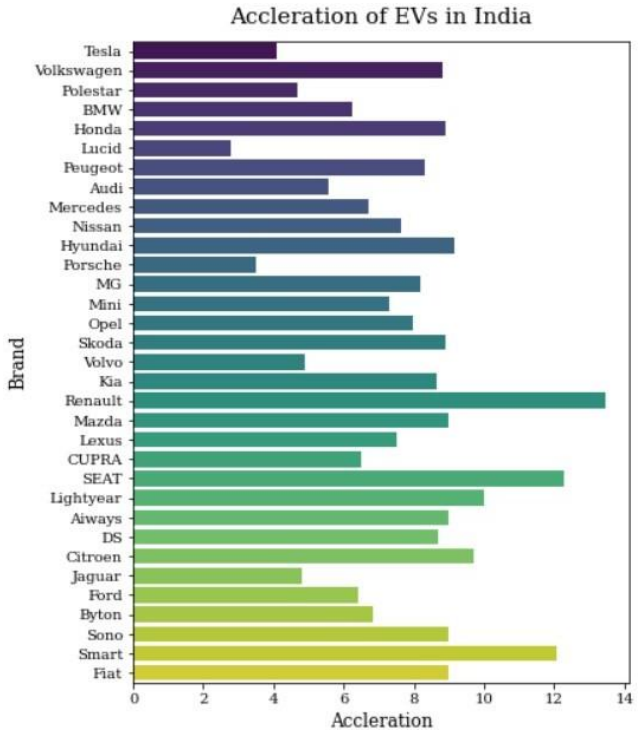


In [26]:
```
plt.plot(df['PriceEuro'], color='orange')
plt.xlabel('Number of Samples', family='serif', size=12)
plt.ylabel('Price', family='serif', size=12)
plt.title('Price Comparison', family='serif', size=15, pad=12);
```



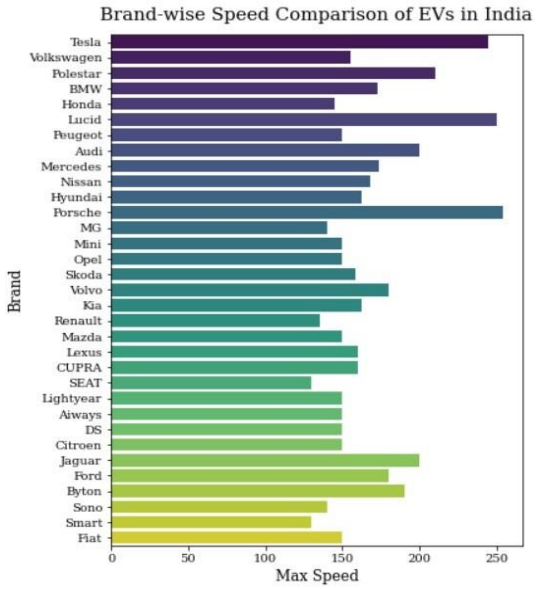

```
#Analysis of EVs based on accleration
#Observation: Based on accleration, EVs from Renault, Seat and Smart are the top performers while Tesla, Lucid and Porsche dont
make it to the same.
```

In [27]:
```
plt.figure(figsize=(6, 8))
sns.barplot(data=df, y='Brand', x='AccelSec', ci=None, palette='viridis')
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.xlabel('Accleration', family='serif', size=12)
plt.ylabel('Brand', family='serif', size=12)
plt.title(label='Accleration of EVs in India', family='serif', size=15, pad=12)
plt.show()
```

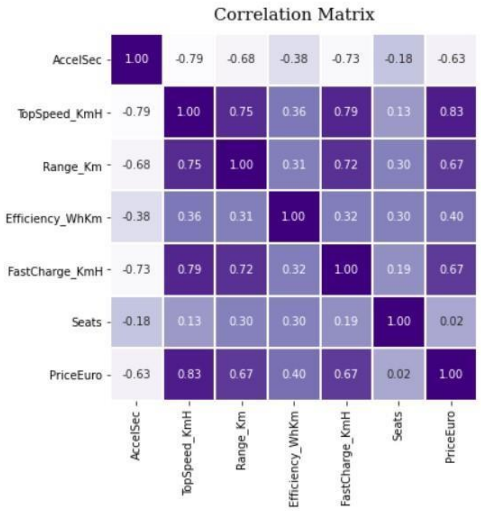


#Analysis of EVs based on speed
#Observation: Based on speed parameter, EVs from Tesla, Lucid and Porsche are the top performers while Renault, Smart and SEAT dont make it to the same.

In [28]:
```
plt.figure(figsize=(6, 8))
sns.barplot(data=df, x='TopSpeed_KmH', y='Brand', ci=None, palette='viridis')
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.xlabel('Max Speed', family='serif', size=12)
plt.ylabel('Brand', family='serif', size=12)
plt.title(label='Brand-wise Speed Comparison of EVs in India', family='serif', size=15, pad=12)
plt.show()
```

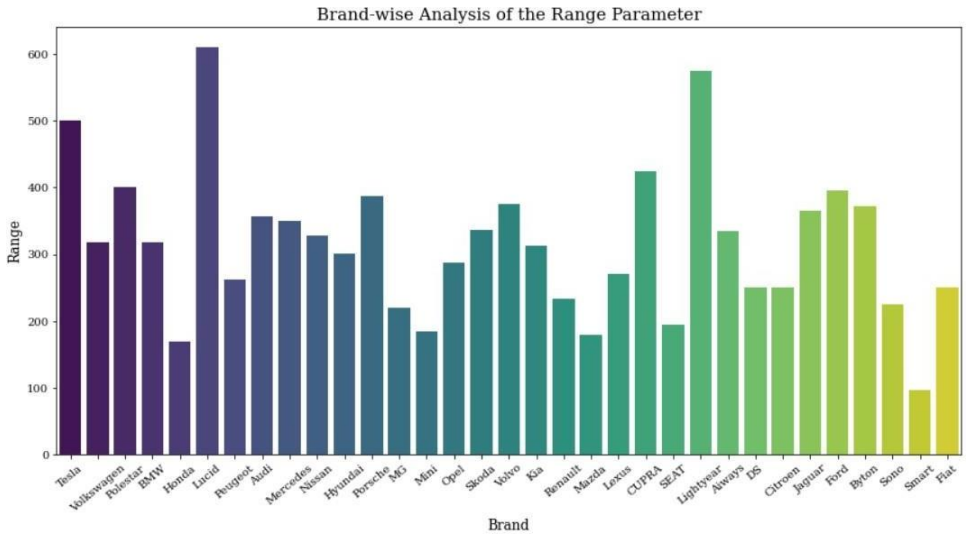


In [29]:
```
plt.figure(figsize=(6,6))
sns.heatmap(data=df.corr(), annot=True, cmap='Purples', cbar=False, square=True, fmt='.2f', linewidths=.3)
plt.title('Correlation Matrix', family='serif', size=15, pad=12);
```



In [ ]:
```
#Analysis of EVs based on the range parameter
#Observation: Based on range (Km), Lucid, Lightyear and Tesla have the highest range and Smart the lowest.
```

In [30]:
```
sns.catplot(kind='bar', data=df, x='Brand', y='Range_Km', palette='viridis', ci=None, height=6, aspect=2)
sns.despine(right=False, top=False)
plt.tick_params(axis='x', rotation=40)
plt.xlabel('Brand',family='serif', size=12)
plt.ylabel('Range', family='serif', size=12)
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.title('Brand-wise Analysis of the Range Parameter', family='serif', size=15);
```



Segmentation

Approaches

Clustering

Clustering is an unsupervised machine learning technique of grouping similar data points into clusters. The sole objective of this technique is to segregate datapoints with similar traits and place them into different clusters. There are several algorithms to perform clustering on data such as k- means clustering, hierarchical clustering, density-based clustering etc.

K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm whose job is to group the unlabelled dataset into different clusters where each datapoint belongs to only one cluster. Here, K is the number of clusters that need to be created in the process. The algorithm finds its applicability into a variety of use cases including market segmentation, image segmentation, image compression, document clustering etc. The below image is the results of clustering on one of our datasets.

Principle Component Analysis

Principal component analysis (PCA) is a linear dimensionality-reduction technique that is used to reduce the dimensionality of large data sets by transforming a large set of variables into a smallerone while preserving most of the information present in the large set.

Elbow Method

The Elbow method is a way of determining the optimal number of clusters (k) in K-Means Clustering. It is based on calculating the Within Cluster Sum of Squared Errors (WCSS) for a different number of clusters (k) and selecting the k for which change in WCSS first starts to diminish. When you plot its graph, at one point the line starts to run parallel to the X-axis and that point, known as the Elbow Point, is considered as the best value for the k .

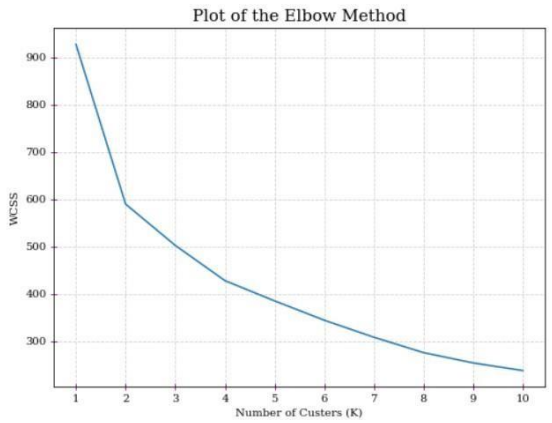
In [ ]:
```
#Model Building Using K-Means Clusteing
```

In [31]:
```
# encoding the categorical features

# PowerTrain feature
df['PowerTrain'].replace(to_replace=['RWD','FWD','AWD'],value=[0, 1, 2],inplace=True)

# RapidCharge feature
df['RapidCharge'].replace(to_replace=['No','Yes'],value=[0, 1],inplace=True)


# selecting features for building a model
X = df[['AccelSec','TopSpeed_KmH','Efficiency_WhKm','FastCharge_KmH', 'Range_Km', 'RapidCharge', 'Seats', 'PriceEuro','PowerTrain


# feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)


# applying Principle Component Analysis (PCA)
pca = PCA(n_components=9)
X_pca = pca.fit_transform(X_scaled)
df_pca = pd.DataFrame(X_pca, columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9'])
df_pca.head()
```

Out[31]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	2.429225	-0.554599	-1.147772	-0.882791	0.839988	-0.959297	0.998880	0.711148	-0.396662
1	-2.322483	-0.345449	0.896473	-1.305529	0.079598	0.235116	-0.213678	-0.544135	-0.181867
2	1.587851	0.008899	-0.650523	0.041024	0.593537	-0.698248	0.058718	0.248837	-0.202775
3	0.291018	-0.000150	-0.307702	-0.514196	-1.608861	0.291624	0.364999	-0.235543	0.261663
4	-2.602679	-0.626489	-0.888088	0.585294	-0.802108	0.027387	-0.084955	-0.507790	-0.049904

In [32]:
```
# plotting the results of Elbow

wcss = []

for i in range(1, 11):
  kmean = KMeans(n_clusters=i, init='k-means++', random_state=90)
  kmean.fit(X_pca)
  wcss.append(kmean.inertia_)

plt.figure(figsize=(8,6))
plt.title('Plot of the Elbow Method', size=15, family='serif')
plt.plot(range(1, 11), wcss)
plt.xticks(range(1, 11), family='serif')
plt.yticks(family='serif')
plt.xlabel('Number of Custers (K)', family='serif')
plt.ylabel('WCSS', family='serif')
plt.grid()
plt.tick_params(axis='both', direction='inout', length=6, color='purple', grid_color='lightgray', grid_linestyle='--')
plt.show()
```



In [33]:
```
# training the model using k=4 as rendered by the above plot
kmean = KMeans(n_clusters=4, init='k-means++', random_state=90)
kmean.fit(X_pca)
```

Out[33]:

```
KMeans(n_clusters=4, random_state=90)
```

In [34]:

```
# check the labels assigned to each data point
print(kmean.labels_)
```

```
[0 3 2 1 1 0 3 3 1 2 2 1 1 2 3 1 0 1 3 1 1 2 1 0 0 1 1 2 3 3 2 1 1 2 1 1 1
 3 3 2 0 1 2 1 1 1 1 0 0 3 2 0 1 1 2 1 1 3 1 0 3 2 2 2 3 0 1 2 3 2 1 2 0 2
 1 1 2 3 2 0 1 2 3 1 2 1 2 2 2 1 2 3 3 2 1 1 1 3 1 2 2 2 2]
```

In [35]:

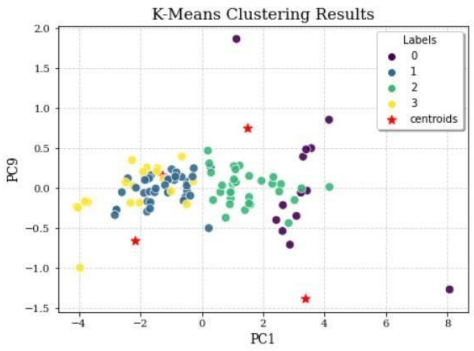
```
# check the size of clusters
pd.Series(kmean.labels_).value_counts()
```

Out[35]:

```
1    39
2    32
3    19
0    13
dtype: int64
```

In [36]:

```
df['clusters'] = kmean.labels_
# visualizing clusters
plt.figure(figsize=(7,5))
sns.scatterplot(data=df_pca, x='PC1', y='PC9', s=70, hue=kmean.labels_, palette='viridis', zorder=2, alpha=.9)
plt.scatter(x=kmean.cluster_centers_[:,0], y=kmean.cluster_centers_[:,1], marker="*", c="r", s=80, label="centroids")
plt.xlabel('PC1', family='serif', size=12)
plt.ylabel('PC9', family='serif', size=12)
plt.xticks(family='serif')
plt.yticks(family='serif')
plt.grid()
plt.tick_params(grid_color='lightgray', grid_linestyle='--', zorder=1)
plt.legend(title='Labels', fancybox=True, shadow=True)
plt.title('K-Means Clustering Results', family='serif', size=15)
plt.show()
```



MARKET SEGMENTATION ON ELECTRIC VEHICLES

Sanket Bangar

GitHub Link: https://github.com/SanketBangar/Market-Segmentation-for-Electric-Vehicle/tree/main

PROBLEM STATEMENT:

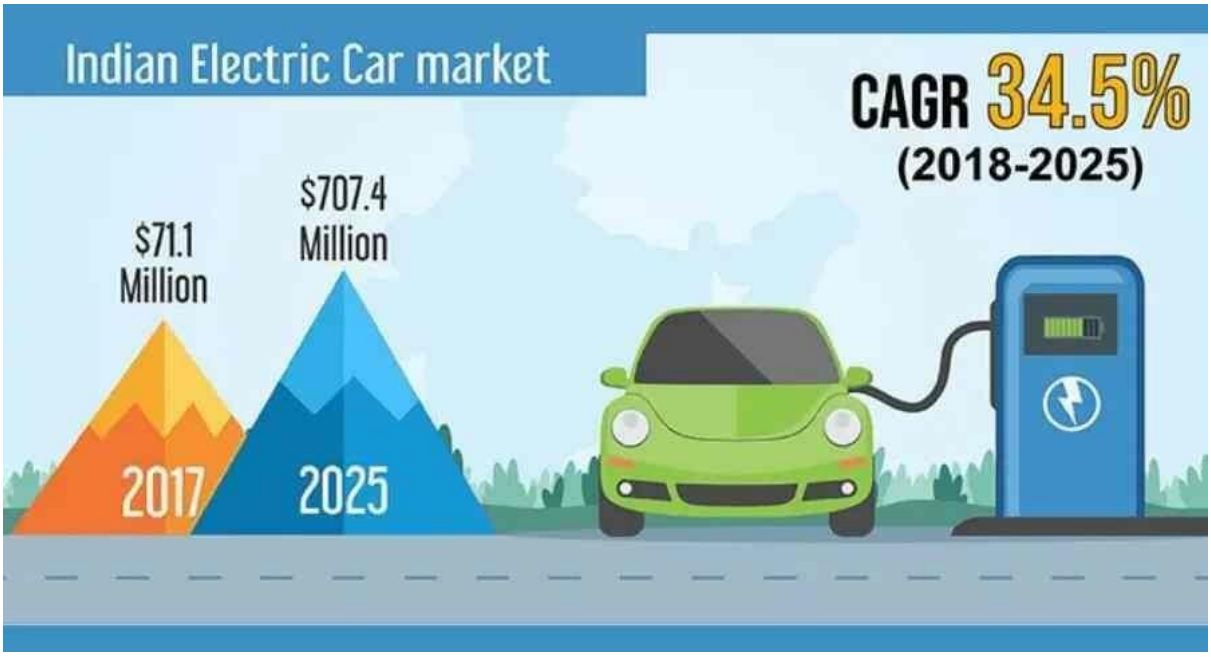
You are a team working under an Electric Vehicle Startup. The Startup is still deciding in which vehicle/customer space it will be develop its EVs. You have to analyze the Electric Vehicle market in India using Segmentation analysis and come up with a feasible strategy to enter the market, targeting the segments most likely to use Electric vehicles.

Overview:

What is Electric Vehicle?

The electric vehicle is a vehicle that runs on electricity alone. Such a vehicle does not contain an internal combustion engine like the other conventional vehicles. Instead, it employs an electric motor to run the wheels. These vehiclesare becoming very popular nowadays. They are considered to be a promising solution for the future transportation. The most common example is Tesla.

Business Opportunity:



GLOBAL ELECTRIC CAR MARKET

Electric Car Market by Technology by Segment, by Geography, Global Market Size, Share, Development, Growth, and Demand Forecast, 2013–2023

PRESCIENT & STRATEGIC INTELLIGENCE
Where knowledge inspires strategy

Global electric car market is projected to attain a size of 52,72,312 units by 2023

European electric car market is the second largest in the world.

Drivers

1. Stringent emission norms
2. Declining battery costs
3. New electric car variants

Restraints

1. Lack of adequate charging infrastructure and value chain ecosystem
2. Mass adoption of electric cars reducing the subsidies

By Technology

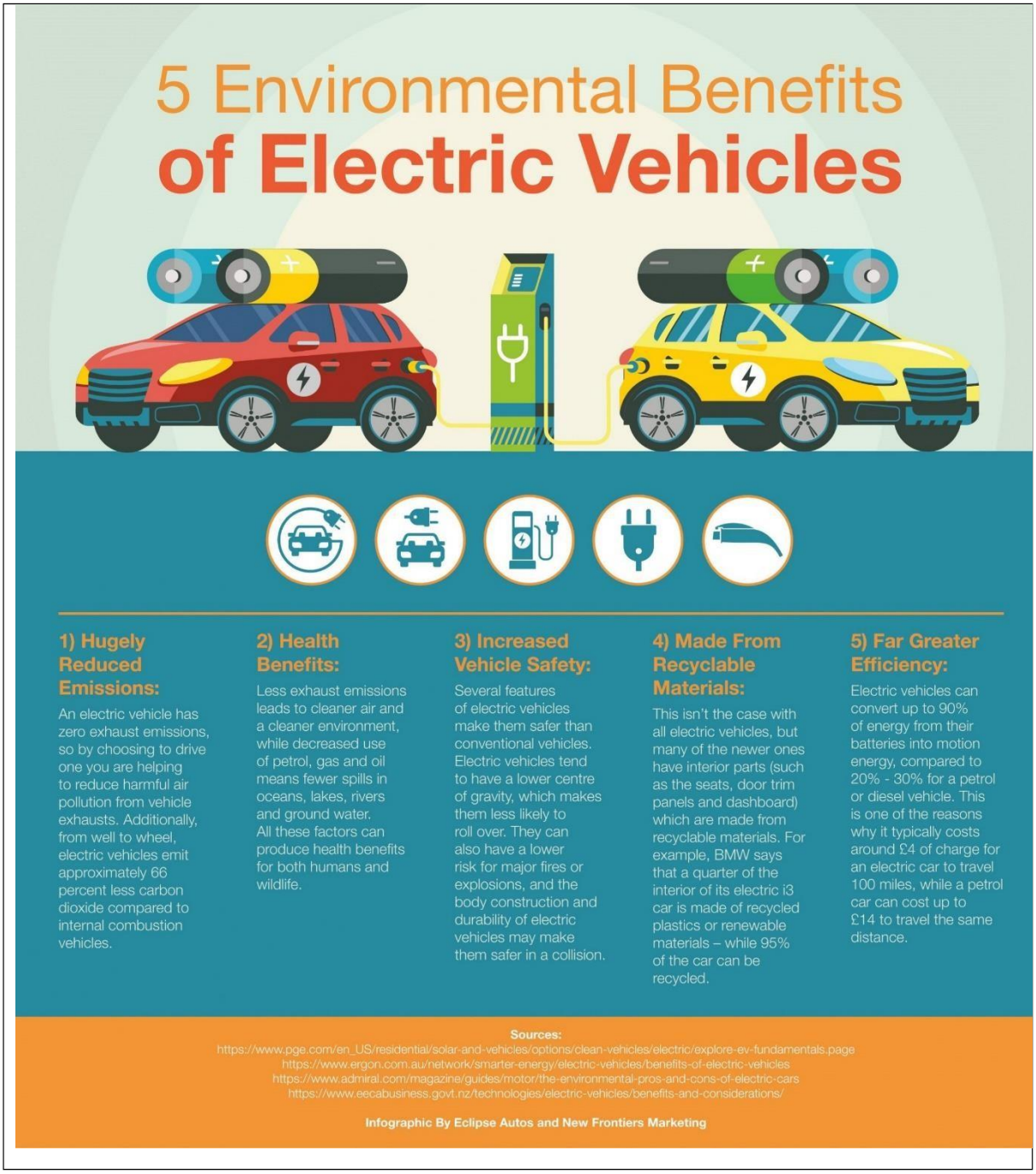
Battery Electric Vehicles

Plug-in hybrid electric vehicles

Major Players

Tesla, GM, BYD, Renault Nissan Mitsubishi, Ford, Volkswagen, Hyundai, BAIC Group, Daimler, Geely

Benefits of Electric Vehicles:



Data Sources:

Data was taken from the Kaggle website

[Indian Consumers Cars purchasing behaviour | Kaggle]

Market Segmentation:

1. Behavioural Segmentation:

Behavioural segmentation is a form of marketing segmentation that divides people into different groups who have a specific behavioural pattern in common. Users may share the same lifecycle stage, previously purchase particular products, or have similar reactions to your messages.

Benefits of Behavioural Segmentation

- Improves targeting accuracy
- Helps provide better-personalized experience
- Sifts engaged users from uninterested
- Saves money
- Makes it easier to track success
- Helps build loyalty to your brand

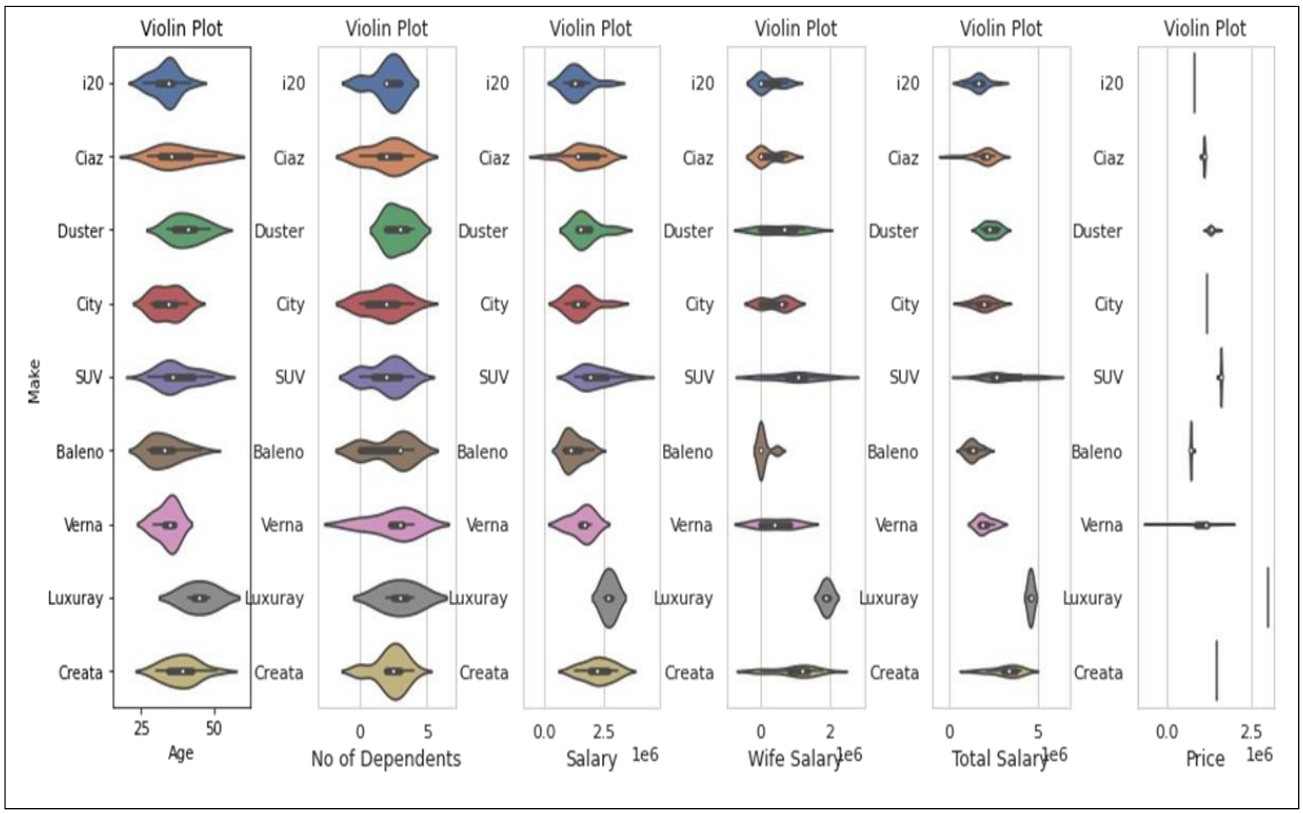
2. Psychographic Segmentation:

Psychographic segmentation's emphasis on characteristics like personality and values differs from demographic segmentation, which uses a specific trait (like gender, age, income, etc.) to categorize potential audiences.

Market researchers use psychographic characteristics to help develop and position their products and marketing messages for different target groups

Marketers use both demographics and psychographics in their market researchto create their marketing strategy. So we will combine these both categories aswell.

The violin plot below gives us some insight on the relation between the segmentation and descriptive variables in our data.



Observations:

Age: Younger consumers purchase less expensive vehicles. This canbe explained simply as they have lesser dependents, lesser income and are single, and so they don't have both the option and the need to buy more expensive vehicles.

Number of Dependents: Greater number of dependents makes theconsumer buy a vehicle with more seats and so they tend to prefer SUVs.

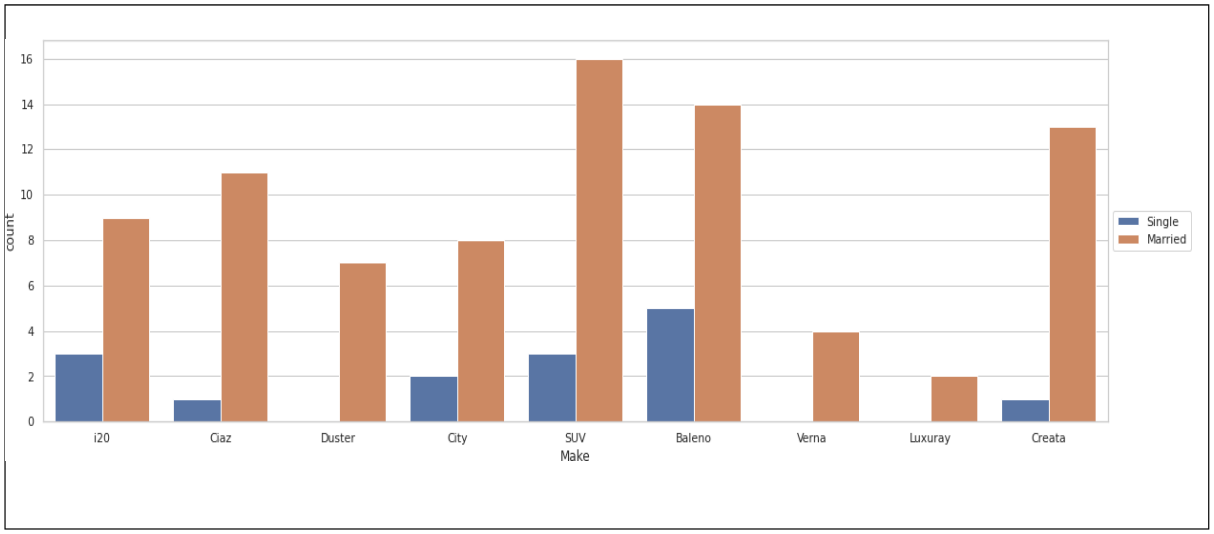
Salary: If you overlap the normalised salary plots with price plot, you would observe the median of salary violin plot matches that ofthe price of the vehicle indicating a very direct relationship, whichmakes sense

asmost people would buy vehicles they can afford.

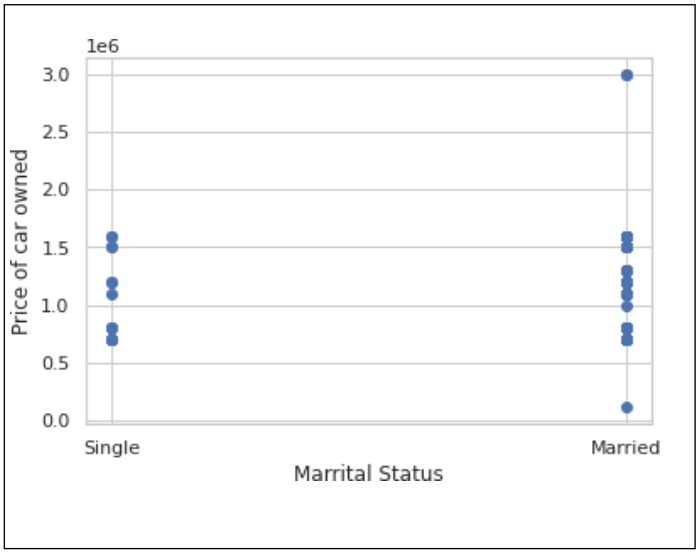
Dependency of make and price of vehicles on other descriptor variables

1) Marital Status:

Make of vehicle they tend to purchase

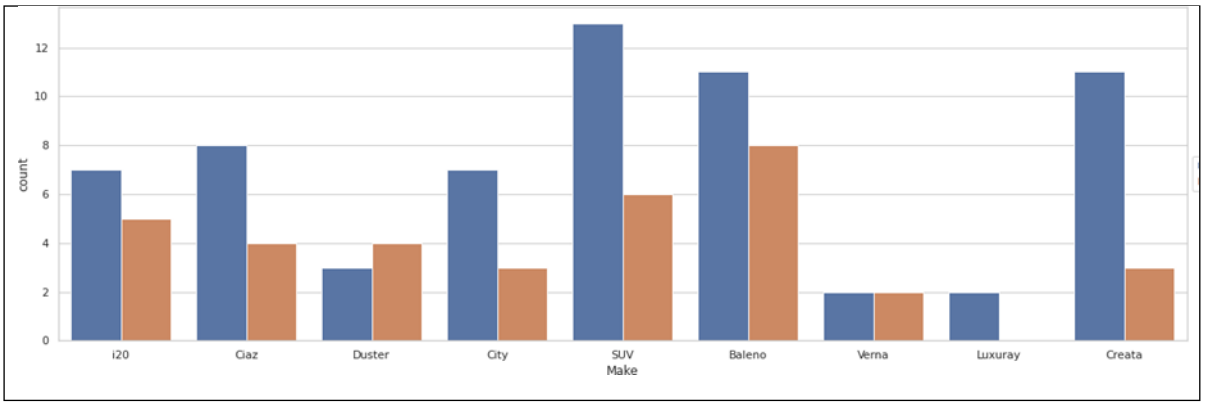


Price of vehicle they owned:



2) Profession:

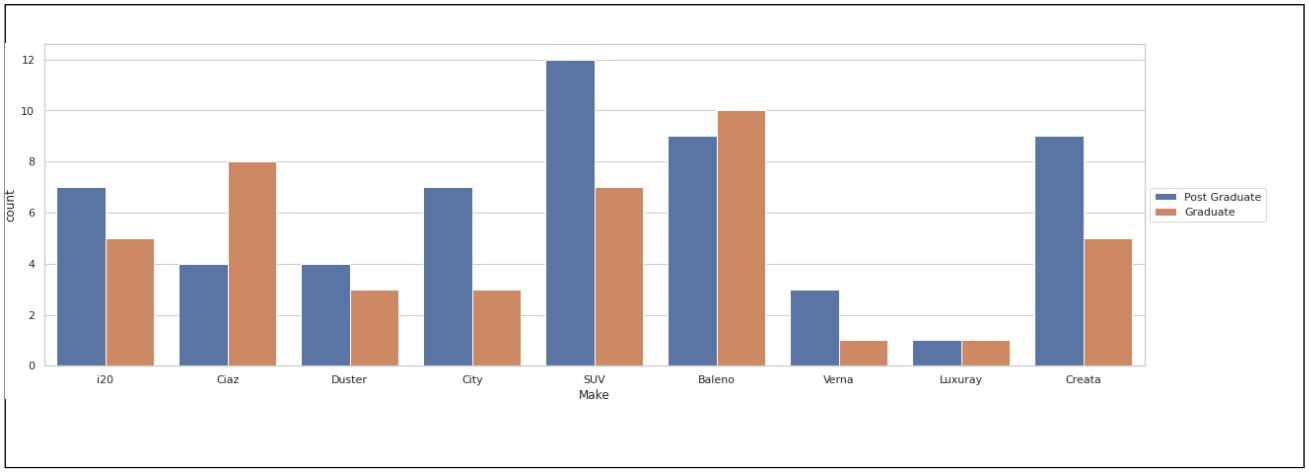
Make of vehicle they tend to purchase:



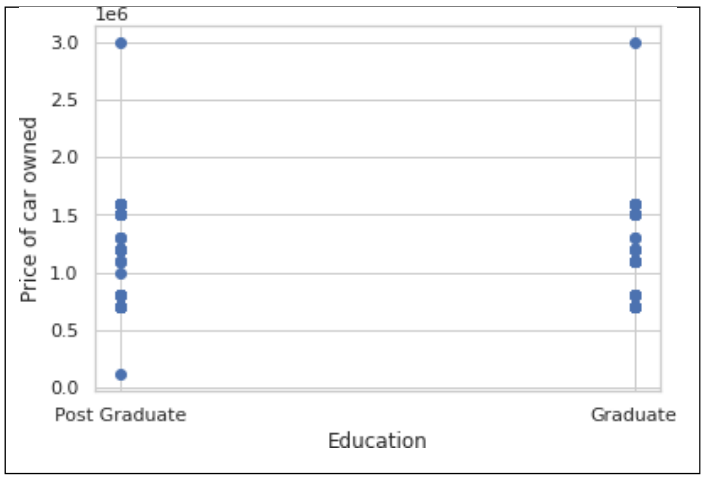
Price of vehicle owned:

3) Education:

Make of vehicle they tend to purchase:

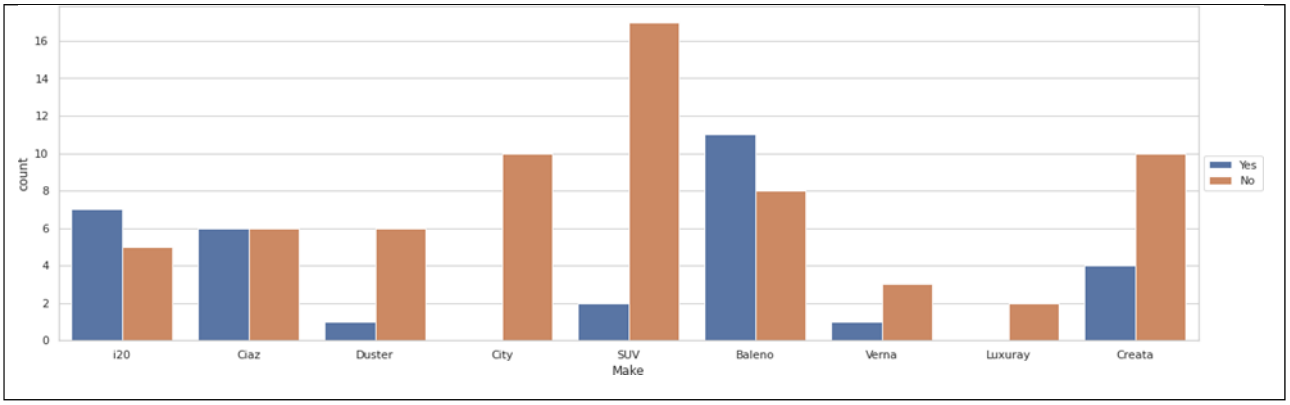


Price of vehicle owned:

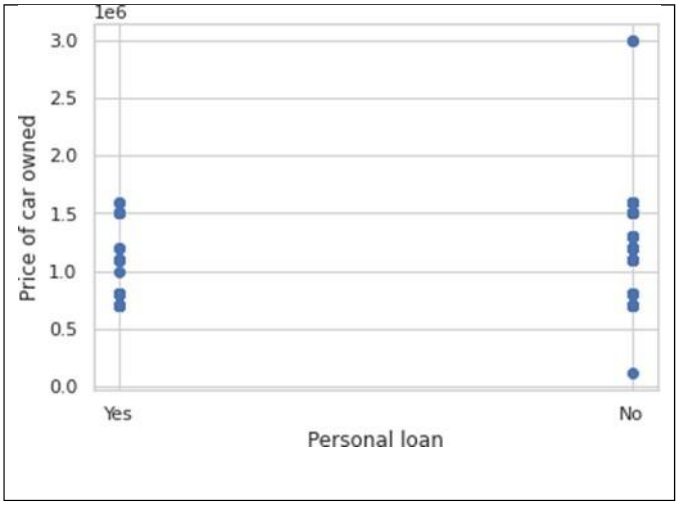


4) Personal Loan:

Make of vehicle they owned:



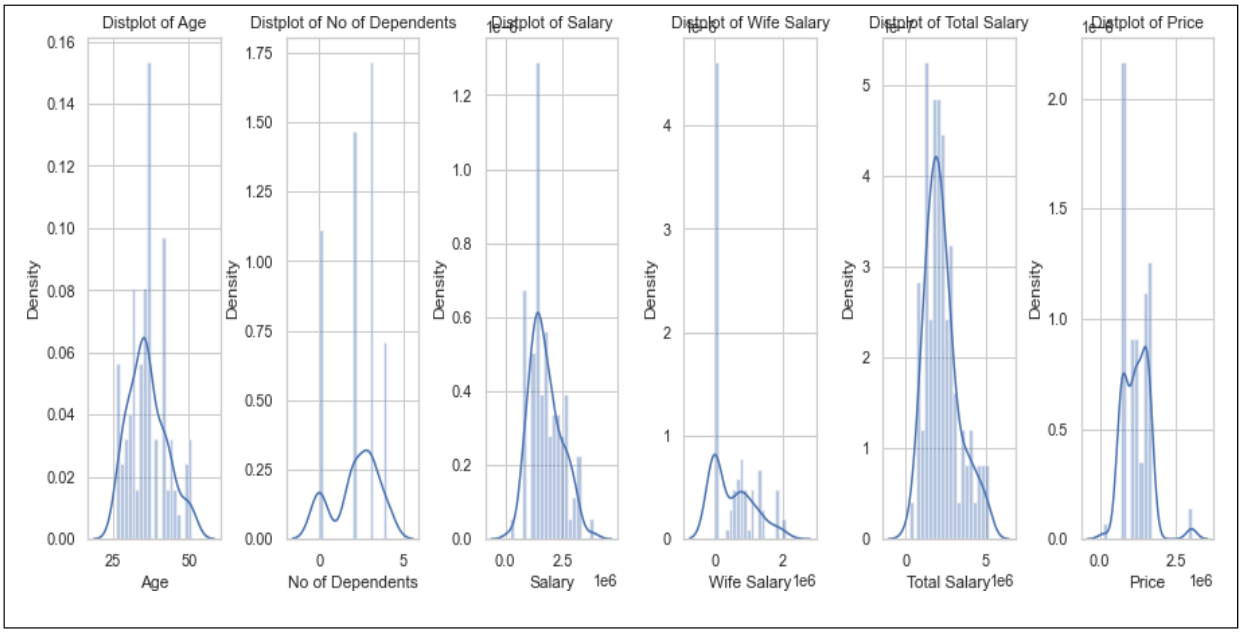
Price of Vehicle owned:



Demographic Segmentation:

Demographic segmentation is a market segmentation technique where an organization's target market is segmented based on demographic variables such as age, gender, education, income, etc. It helps organizations understand who their customers are so that their needs can be addressed more effectively. When an organization looks at the demographic segmentation, it focuses on the peoplewho are most likely to buy a product. This helps in identifying the target market.

We have used the same dataset we used for behavioral and psychographic analysis and the following plots help us understand the socio-demographic structure of the market:



Geographic Segmentation:

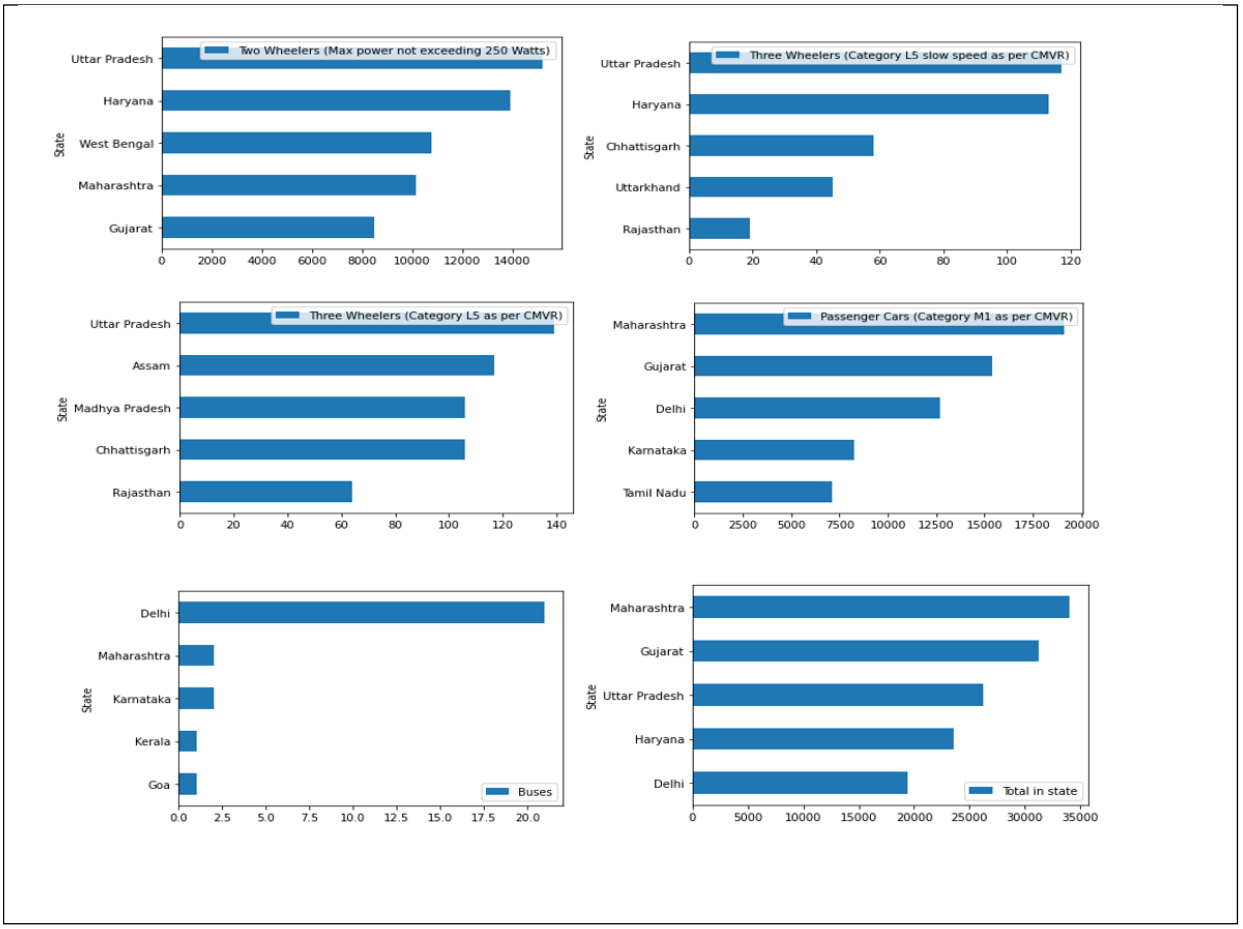
Geographic segmentation is the process of dividing people into groups based on location, such as city, country, state, region, and even continent. It can help youtailor your approach during seasons customers may need your product.

For example, a fisherman in Alaska may only buy more equipment leading upto the salmon season. Whereas a fisherman in Orange Beach, Alabama, might purchase equipment all year round.

In contrast to other types of segmentation — demographic, psychographic, and behavioral — location-based segmentation analysis is easier to see results from.It doesn't take a lot of research to identify someone's location and the characteristics of a certain area, versus figuring out potential customers purchasing behaviors and psychographics. Here we have made divisions in terms of states and union territories in India.

For geographic analysis we used state-wise sales of different types of Electric Vehicles dataset which would help us understand our target region. Based on the type of electric vehicle, states with higher numbers of electric vehicles can be targeted as people in these states are more likely to purchase them. Given below

are bar charts showing the top 5 states in sales of a particular EV type:



Depending on the type of Electric Vehicle the start-up comes with, it can target that particular state. What is important to consider is that for most of these electric vehicles that market would be a fairly developed city in that state, because consumers should be willing to purchase the electric vehicle and factorslike cost versus average consumer income and the resources to charge the EV (e.g., Charging Stations) and being able to maintain it are important

Model fitting:

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solvethe clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the

Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-

defined clusters that need to be created in the process, as if K=2, there will betwo clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similarproperties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without theneed for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid.The main aim oif this algorithm is to minimize the sum of distances between thedata point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K centre points or centroids by an iterativeprocess.
- Assigns each data point to its closest k-centre. Those data points whichare near to the particular k-centre, create a cluster.

Data pre-processing:

The libraries used are:

- Numpy – For Computations
- Pandas – Manipulating the datasets
- Scikit-learn – For ML-based applications

In [1]:
```
#Importing required libraries
import numpy as np
import pandas as pd
```

```
In [8]: #Label Encoding Categorical Variables
        from sklearn.preprocessing import LabelEncoder
```

With the help of this dataset, we can implement Behavioural, Psychographicand Demographic Segmentation of Indian Automobile Market. This helps us with understanding the various attributes leading to the consumer buying behaviour.

```
In [2]: #Loading the dataset
        df = pd.read_csv("Indian automoble buying behavour study 1.0.csv")
```

After uploading the 'Indian automobile buying behaviour study 1.0' dataset, weinspect the dataset. It has 99 rows and 13 columns.

```
In [4]: #Size of the dataset
        df.shape

Out[4]: (99, 13)
```

The various features include Age, Profession, Education, Number of dependentsof the person buying the Vehicle and if the buyer is married or not, spouse is earning or not to ascertain the net income of the family. It further includes the Maker and the Price of the Vehicle of the buyer.

```
In [3]: #Inspecting the dataset
        df.head()
```

Out[3]:

	Age	Profession	Marrital Status	Education	No of Dependents	Personal loan	House Loan	Wife Working	Salary	Wife Salary	Total Salary	Make	Price
0	27	Salaried	Single	Post Graduate	0	Yes	No	No	800000	0	800000	i20	800000
1	35	Salaried	Married	Post Graduate	2	Yes	Yes	Yes	1400000	600000	2000000	Ciaz	1000000
2	45	Business	Married	Graduate	4	Yes	Yes	No	1800000	0	1800000	Duster	1200000
3	41	Business	Married	Post Graduate	3	No	No	Yes	1600000	600000	2200000	City	1200000
4	31	Salaried	Married	Post Graduate	2	Yes	No	Yes	1800000	800000	2600000	SUV	1600000

We then use describe() function to gain in-depth insight about the mean, medianand other statistical figures about the attributes of the dataset. This gives us an idea about the buyers' essential ranges.

```
In [5]: #Statistics for Numerical Variables
        df.describe()
```

Out[5]:

	Age	No of Dependents	Salary	Wife Salary	Total Salary	Price
count	99.000000	99.000000	9.900000e+01	9.900000e+01	9.900000e+01	9.900000e+01
mean	36.313131	2.181818	1.736364e+06	5.343434e+05	2.270707e+06	1.194040e+06
std	6.246054	1.335265	6.736217e+05	6.054450e+05	1.050777e+06	4.376955e+05
min	26.000000	0.000000	2.000000e+05	0.000000e+00	2.000000e+05	1.100000e+05
25%	31.000000	2.000000	1.300000e+06	0.000000e+00	1.550000e+06	8.000000e+05
50%	36.000000	2.000000	1.600000e+06	5.000000e+05	2.100000e+06	1.200000e+06
75%	41.000000	3.000000	2.200000e+06	9.000000e+05	2.700000e+06	1.500000e+06
max	51.000000	4.000000	3.800000e+06	2.100000e+06	5.200000e+06	3.000000e+06

There aren't any missing values or irrelevant attributes and thus needs no data handling.

In most of the algorithms, categorical values cannot be handled. Thus, there is aneed to convert these categorical values to numerical levels. For this, we make use of the Label Encoder function.

In [8]:
```
#Label Encoding Categorical Variables
from sklearn.preprocessing import LabelEncoder
labelen = LabelEncoder()
df['Profession'] = labelen.fit_transform(df['Profession'])
df['Marrital Status'] = labelen.fit_transform(df['Marrital Status'])
df['Education'] = labelen.fit_transform(df['Education'])
df['Personal loan'] = labelen.fit_transform(df['Personal loan'])
df['House Loan'] = labelen.fit_transform(df['House Loan'])
df['Wife Working'] = labelen.fit_transform(df['Wife Working'])
```

In [9]:
```
#Encoded dataset
df.head()
```

Out[9]:

	Age	Profession	Marrital Status	Education	No of Dependents	Personal loan	House Loan	Wife Working	Salary	Wife Salary	Total Salary	Make	Price
0	27	1	1	1	0	1	0	0	800000	0	800000	i20	800000
1	35	1	0	1	2	1	1	1	1400000	600000	2000000	Ciaz	1000000
2	45	0	0	0	4	1	1	0	1800000	0	1800000	Duster	1200000
3	41	0	0	1	3	0	0	1	1600000	600000	2200000	City	1200000
4	31	1	0	1	2	1	0	1	1800000	800000	2600000	SUV	1600000

ML algorithms work better when feature values are on relatively on a similar scale and close to normalized distribution. Using StandardScaler(), we scale the entire numerical portion of the dataset for an enhanced productive result.

In [10]:
```
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
scaled_df=df
scaled_df[["Age","Salary","Wife Salary","Total Salary","Price"]]=StandardScaler().fit_transform(df[["Age","Salary","Wife Salary",
scaled_df
```

Out[10]:

	Age	Profession	Marrital Status	Education	No of Dependents	Personal loan	House Loan	Wife Working	Salary	Wife Salary	Total Salary	Make	Price
0	-1.498630	1	1	1	0	1	0	0	-1.397118	-0.887055	-1.406760	i20	-0.904843
1	-0.211304	1	0	1	2	1	1	1	-0.501877	0.108995	-0.258937	Ciaz	-0.445579
2	1.397855	0	0	0	4	1	1	0	0.094950	-0.887055	-0.450240	Duster	0.013685
3	0.754191	0	0	1	3	0	0	1	-0.203464	0.108995	-0.067633	City	0.013685
4	-0.854967	1	0	1	2	1	0	1	0.094950	0.441012	0.314975	SUV	0.932213
...	...	...	...	...	...	...	...	...	...	...	...	...	...
94	-1.498630	0	1	0	0	0	0	0	0.990190	-0.887055	0.123671	SUV	0.932213
95	2.202434	1	0	1	3	0	0	1	3.079085	1.271054	2.706274	SUV	0.932213
96	2.363350	0	0	0	2	1	1	0	0.691777	-0.887055	-0.067633	Ciaz	-0.215947
97	2.363350	1	0	1	2	0	0	1	1.437811	1.271054	1.654102	Creata	0.702581
98	2.363350	1	0	1	2	1	1	0	0.691777	-0.887055	-0.067633	Ciaz	-0.215947

99 rows × 13 columns

Libraries used for the algorithm:

K - Means Algorithm

In [37]:
```
# Importing Important Libraries
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

In [38]:
```
X_scaled = StandardScaler().fit_transform(obj_df)
X_scaled = pd.DataFrame(X_scaled,columns=['Age', 'Profession', 'Marrital Status', 'Education', 'No of Dependents',
                                          'Personal loan', 'House Loan', 'Wife Working', 'Salary', 'Wife Salary',
                                          'Total Salary','Price'])
x = X_scaled.to_numpy()
X_scaled
```

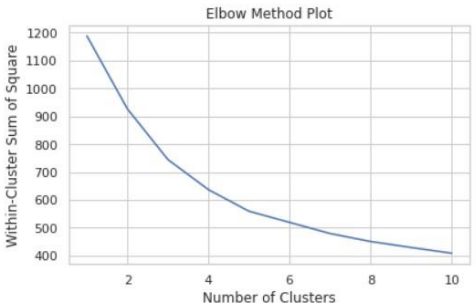
Out[38]:

	Age	Profession	Marrital Status	Education	No of Dependents	Personal loan	House Loan	Wife Working	Salary	Wife Salary	Total Salary	Price
0	-1.498630	-0.739510	-2.366432	0.876275	-1.642313	1.446980	-0.772512	-1.051847	-1.397118	-0.887055	-1.406760	-0.904843
1	-0.211304	-0.739510	0.422577	0.876275	-0.136859	1.446980	1.294479	0.950708	-0.501877	0.108995	-0.258937	-0.445579
2	1.397855	1.352247	0.422577	-1.141195	1.368594	1.446980	1.294479	-1.051847	0.094950	-0.887055	-0.450240	0.013685
3	0.754191	1.352247	0.422577	0.876275	0.615867	-0.691095	-0.772512	0.950708	-0.203464	0.108995	-0.067633	0.013685
4	-0.854967	-0.739510	0.422577	0.876275	-0.136859	1.446980	-0.772512	0.950708	0.094950	0.441012	0.314975	0.932213
...	...	...	...	...	...	...	...	...	...	...	...	...
94	-1.498630	1.352247	-2.366432	-1.141195	-1.642313	-0.691095	-0.772512	-1.051847	0.990190	-0.887055	0.123671	0.932213
95	2.202434	-0.739510	0.422577	0.876275	0.615867	-0.691095	-0.772512	0.950708	3.079085	1.271054	2.706274	0.932213
96	2.363350	1.352247	0.422577	-1.141195	-0.136859	1.446980	1.294479	-1.051847	0.691777	-0.887055	-0.067633	-0.215947
97	2.363350	-0.739510	0.422577	0.876275	-0.136859	-0.691095	-0.772512	0.950708	1.437811	1.271054	1.654102	0.702581
98	2.363350	-0.739510	0.422577	0.876275	-0.136859	1.446980	1.294479	-1.051847	0.691777	-0.887055	-0.067633	-0.215947

99 rows × 12 columns

In [39]:
```
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
                    max_iter = 300, n_init = 10,random_state = 0)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

In [40]:
```
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method Plot')
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Square') # Within cluster sum of squares
plt.tight_layout()
plt.show()
```



Either take K=3 or K= 5

1) when k= 3

k = 3

In [43]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++',
                max_iter = 300, n_init = 10,random_state = 42)
kmeans.fit(X_scaled)
```

```
C:\Users\dhruv\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak
on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OM
P_NUM_THREADS=1.
  warnings.warn(
```

Out[43]:
```
KMeans(n_clusters=3, random_state=42)
```

In [44]:
```
y = kmeans.predict(X_scaled)
y_df = pd.DataFrame(y,columns=['Class'])
```

In [45]:
```
final_data = pd.concat([df,y_df],axis=1)
final_data
```

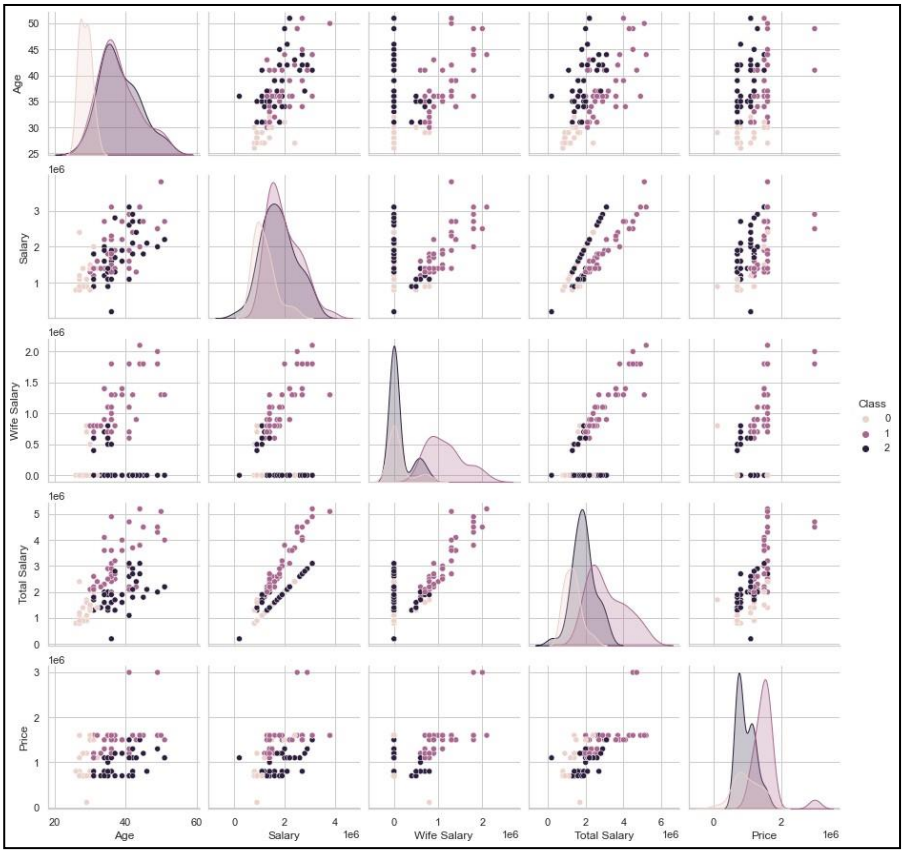
Out[45]:

	Age	Profession	Marrital Status	Education	No of Dependents	Personal loan	House Loan	Wife Working	Salary	Wife Salary	Total Salary	Make	Price	Class
0	27	Salaried	Single	Post Graduate	0	Yes	No	No	800000	0	800000	i20	800000	0
1	35	Salaried	Married	Post Graduate	2	Yes	Yes	Yes	1400000	600000	2000000	Ciaz	1000000	2
2	45	Business	Married	Graduate	4	Yes	Yes	No	1800000	0	1800000	Duster	1200000	2
3	41	Business	Married	Post Graduate	3	No	No	Yes	1600000	600000	2200000	City	1200000	1

Class 0: total salary equals to husband salary Class

1: total salary is greater than husband salary

Class 2: total salary is nearly equal and greater than husband salary



2) when k= 5

k = 5

In [49]:
```
kmeans1 = KMeans(n_clusters = 5, init = 'k-means++',
                 max_iter = 300, n_init = 10,random_state = 42)
kmeans1.fit(X_scaled)
```

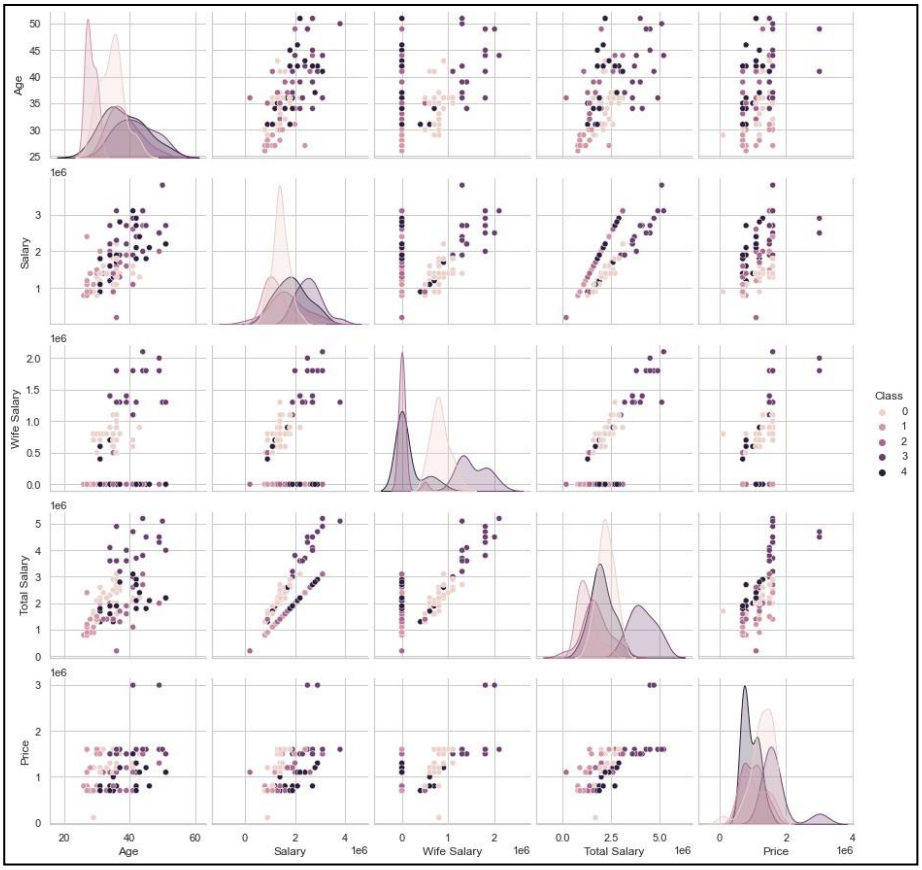
Out[49]:
```
KMeans(n_clusters=5, n_init=10, random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [50]:
```
y1 = kmeans1.predict(X_scaled)
y1_df = pd.DataFrame(y1,columns=['Class'])
```

In [51]:
```
final_data1 = pd.concat([df,y1_df],axis=1)
final_data1
```

Out[51]:

	Age	Profession	Marrital Status	Education	No of Dependents	Personal loan	House Loan	Wife Working	Salary	Wife Salary	Total Salary	Make	Price	Class
0	27	Salaried	Single	Post Graduate	0	Yes	No	No	800000	0	800000	i20	800000	1
1	35	Salaried	Married	Post Graduate	2	Yes	Yes	Yes	1400000	600000	2000000	Ciaz	1000000	4
2	45	Business	Married	Graduate	4	Yes	Yes	No	1800000	0	1800000	Duster	1200000	4
3	41	Business	Married	Post Graduate	3	No	No	Yes	1600000	600000	2200000	City	1200000	2



To reduce the homogeneity, we chose k= 3

Potential Customer Base * Your Target Price Range = Potential Profit

For Electric 2-Wheeler

The per unit average price will be 1 lakh and the number of units sold will be around 5,00,000.

Potential Profit = 500000 *100000 = 50 billionFor

Electric 3-Wheeler

The per unit Avg. price will be 2 lakhs and the number of units sold will be around 90000

Potential Profit = 90000*200000 = 18 billion

Optimal market segments to open in the market

From the above report, we conclude that to create an Electric Vehicle startupin India, the most optimal market segment for us will be based on Geographicand Demographic segments which would be the most amount of EVS sold in particular states and the type of electric vehicle respectively.

After analysing the EV market using Market Segmentation Analysis, the feasible strategy which we have come up with is that we will be focussing onthe states that have more demand for EVs like Maharashtra and Gujrat. Also,one more reason to set the startup in these 2 states is that the infrastructure required for the EVS including the charging station is available which wouldease our setting up of the startup process.

Marketing Mix with regard to EV:

PRICE

Affordability is the number one issue for any vehicle, more so in the case of EVs. The more cost efficient a product is the more it's sale. We can see fromthe above analysis that the product's price should ideally range between 10 to20 lakh, as most people would make a purchase in this range.

PRODUCT

Product totally depends on the start-up, it's design, it's mechanics. Having saidthat, in general, if an EV start-up has to get successful in India, it's key would be to get into 2-wheeler EV business.

Another type of product EV Start up can look into is public transport vehicles,because the current government policies are supportive for revamping public transport to electric-based engines

PLACE

Major cities of the country (Especially metropolitan cities) should be targeted asthese are the places where infrastructure would support. Another reason for targeting urban cities is that here it is more likely to have an aware population willing to buy Electric Vehicles.

For different types of vehicles, the list of top states which will promise a goodmarket have been given in our geographical analysis.

PROMOTION

In EV business, awareness is the key, with its edge over fuel using vehicles,more and more people should be made aware of its advantages.

https://github.com/Varunkumarjetty/EV-segmentation

July 16, 2023

1 EV Market Segmenation

Name - Varun Kumar Jetty

Importing libraries

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

[2]:
```
df = pd.read_csv(r"C:\Users\admin\Downloads\CAR DETAILS FROM CAR DEKHO.csv")
```

[3]:
```
df.sample(10)
```

[3]:

	name	year	selling_price	km_driven	\
678	Honda Jazz VX	2018	600000	20000	
1161	Maruti Swift ZDI Plus	2015	484999	73000	
2398	Maruti Alto LXi	2007	90000	100000	
4052	Hyundai Creta 1.6 CRDi SX	2016	1151000	85000	
1265	Chevrolet Beat Diesel LS	2013	190000	70000	
1371	Maruti Swift Dzire ZDI	2019	475000	120000	
801	Maruti Wagon R LXI	2009	180000	30375	
1176	Renault KWID 1.0 RXT Optional	2017	350000	10000	
1268	Toyota Corolla Executive (HE)	2009	300000	23262	
127	Toyota Innova 2.5 GX (Diesel) 8 Seater	2012	500000	100000	

	fuel	seller_type	transmission		owner
678	Petrol	Individual	Manual	First	Owner
1161	Diesel	Dealer	Manual	Second	Owner
2398	Petrol	Individual	Manual	Second	Owner
4052	Diesel	Individual	Manual	First	Owner
1265	Diesel	Individual	Manual	First	Owner
1371	Diesel	Individual	Manual	Second	Owner
801	Petrol	Dealer	Manual	First	Owner

1176	Petrol	Individual	Manual	First Owner
1268	Petrol	Dealer	Manual	First Owner
127	Diesel	Individual	Manual	First Owner

2 Exploratory Data Analysis

An Exploratory Data Analysis, or EDA is a thorough examination meant to uncover the underlying structure of a data set and is important for a company because it exposes trends, patterns, and relationships that are not readily apparent.

[4]:
```
df.shape
```

[4]: (4340, 8)

[5]:
```
df.describe()
```

[5]:

	year	selling_price	km_driven
count	4340.000000	4.340000e+03	4340.000000
mean	2013.090783	5.041273e+05	66215.777419
std	4.215344	5.785487e+05	46644.102194
min	1992.000000	2.000000e+04	1.000000
25%	2011.000000	2.087498e+05	35000.000000
50%	2014.000000	3.500000e+05	60000.000000
75%	2016.000000	6.000000e+05	90000.000000
max	2020.000000	8.900000e+06	806599.000000

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4340 entries, 0 to 4339
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   name           4340 non-null   object
 1   year           4340 non-null   int64 
 2   selling_price  4340 non-null   int64 
 3   km_driven      4340 non-null   int64 
 4   fuel           4340 non-null   object
 5   seller_type    4340 non-null   object
 6   transmission   4340 non-null   object
 7   owner          4340 non-null   object
dtypes: int64(3), object(5)
memory usage: 271.4+ KB
```

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

[7]:
```
name             0
year             0
selling_price    0
km_driven        0
fuel             0
seller_type      0
transmission     0
owner            0
dtype: int64
```

[8]:
```
df.fuel.unique()
```

[8]:
```
array(['Petrol', 'Diesel', 'CNG', 'LPG', 'Electric'], dtype=object)
```

[9]:
```
df.fuel.value_counts()
```

[9]:
```
Diesel      2153
Petrol      2123
CNG           40
LPG           23
Electric       1
Name: fuel, dtype: int64
```

[10]:
```
electric_cars = df[df['fuel'] == 'Electric']
print(electric_cars)
```

```
                        name  year  selling_price  km_driven      fuel  \
4145  Toyota Camry Hybrid   2006         310000      62000  Electric

     seller_type transmission           owner
4145      Dealer    Automatic  Second Owner
```

[11]:
```
df.seller_type.unique()
```

[11]:
```
array(['Individual', 'Dealer', 'Trustmark Dealer'], dtype=object)
```

[12]:
```
df.transmission.unique()
```

[12]:
```
array(['Manual', 'Automatic'], dtype=object)
```

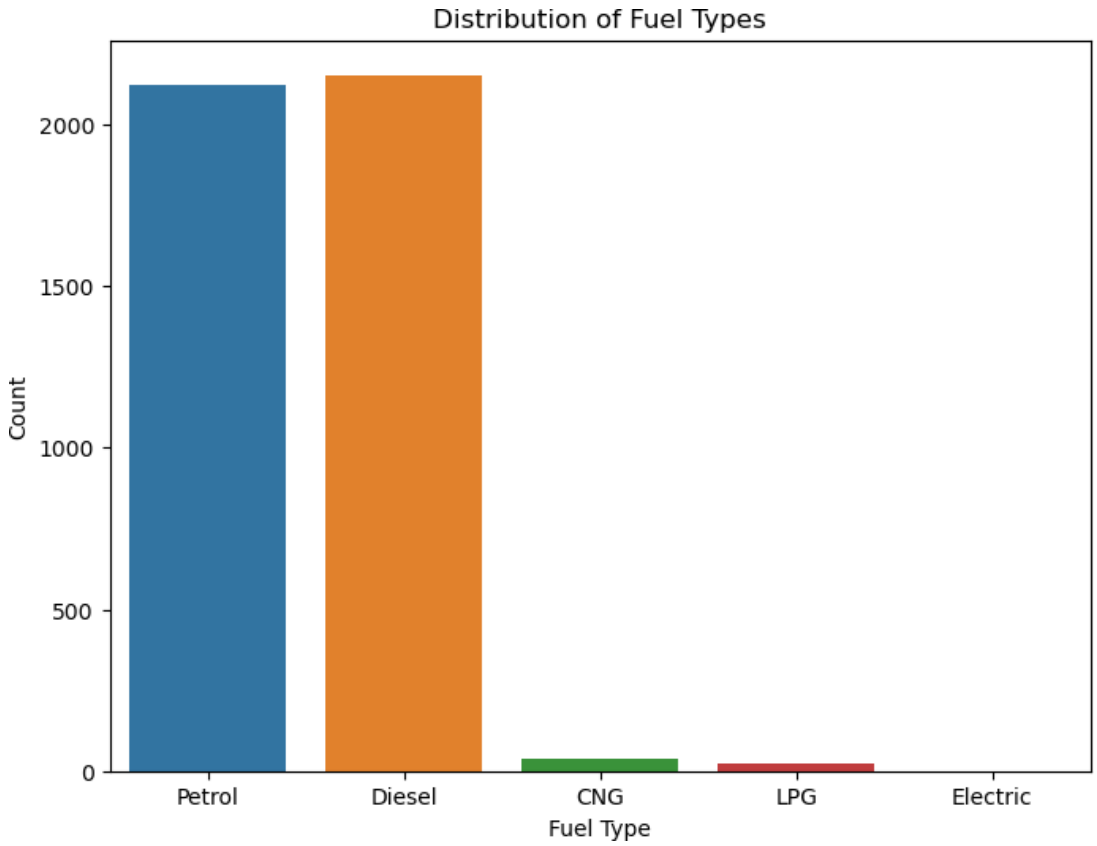
[13]:
```
df.owner.unique()
```

[13]:
```
array(['First Owner', 'Second Owner', 'Fourth & Above Owner',
       'Third Owner', 'Test Drive Car'], dtype=object)
```

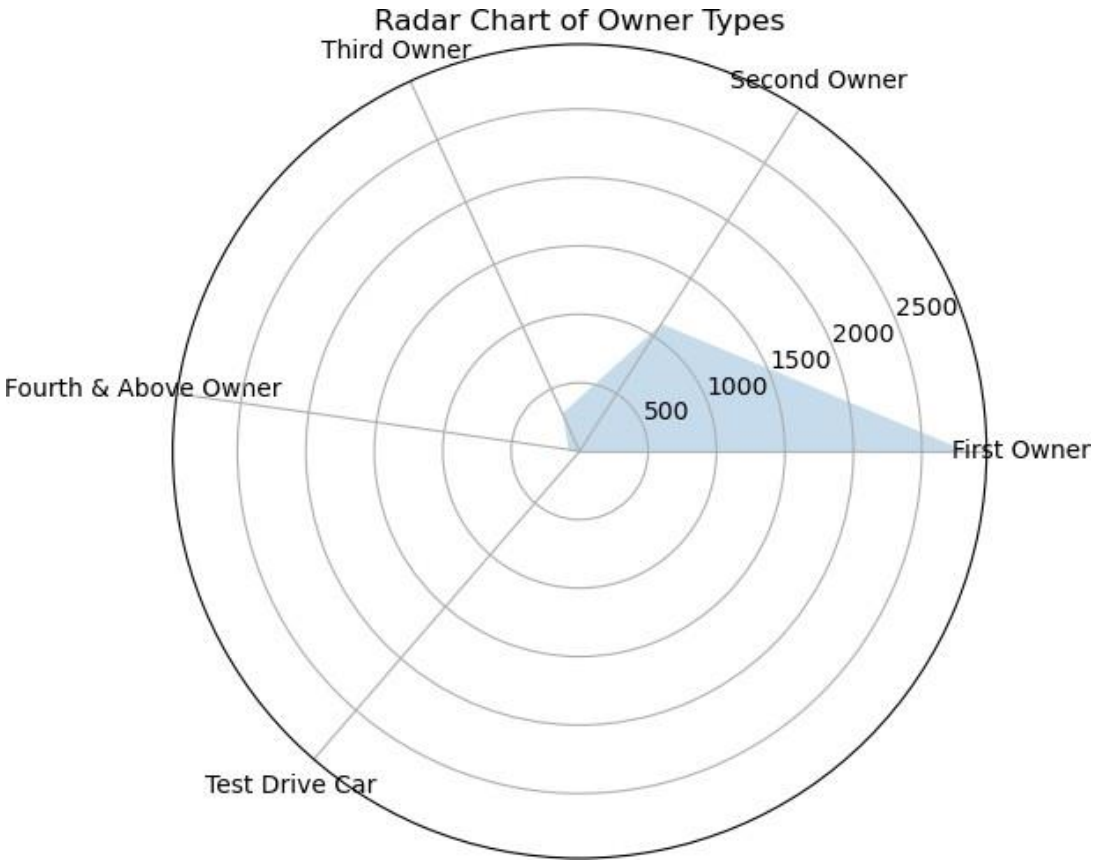
[14]:
```
df.name.unique
```

[14] : <bound method Series.unique of 0 Maruti 800 AC
1 Maruti Wagon R LXI Minor
2 Hyundai Verna 1.6 SX
3 Datsun RediGO T Option
4 Honda Amaze VX i-DTEC
...
4335 Hyundai i20 Magna 1.4 CRDi (Diesel)
4336 Hyundai i20 Magna 1.4 CRDi
4337 Maruti 800 AC BSIII
4338 Hyundai Creta 1.6 CRDi SX Option
4339 Renault KWID RXT
Name: name, Length: 4340, dtype: object>

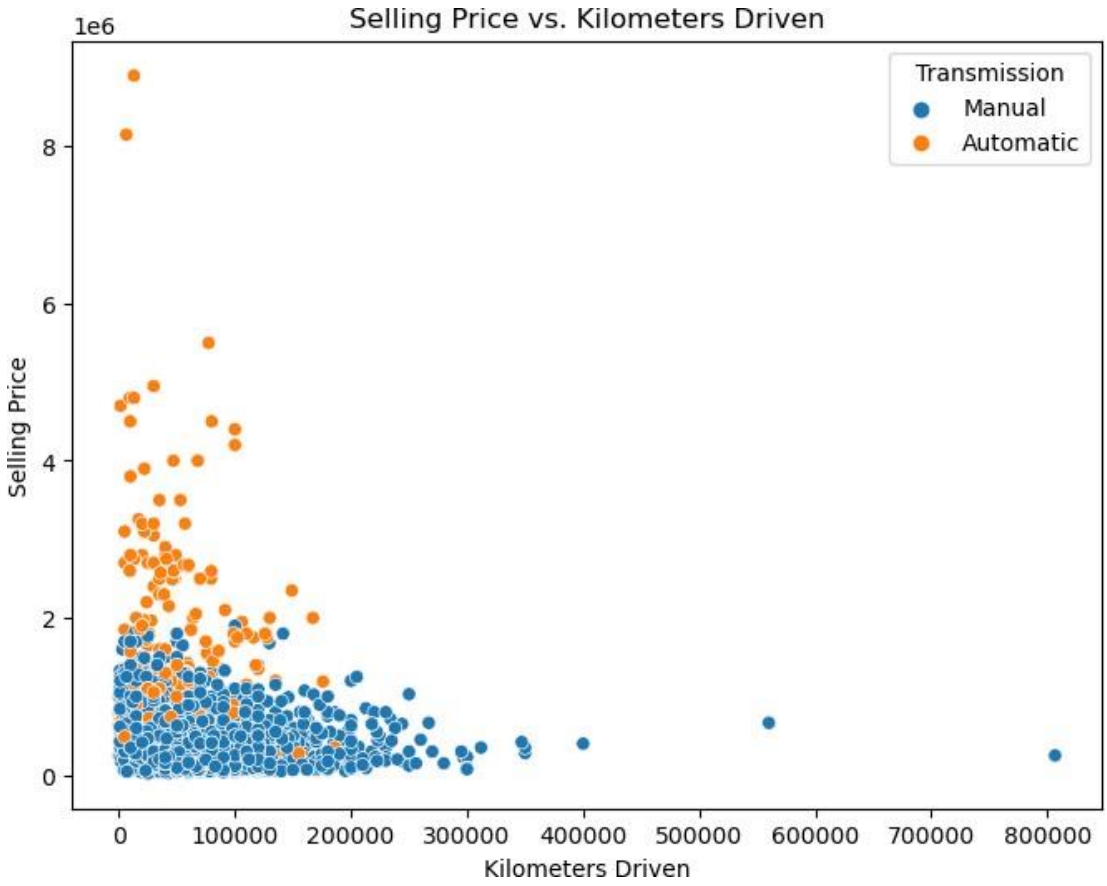
[15]:
```
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='fuel')
plt.title('Distribution of Fuel Types')
plt.xlabel('Fuel Type')
plt.ylabel('Count')
plt.show()
```



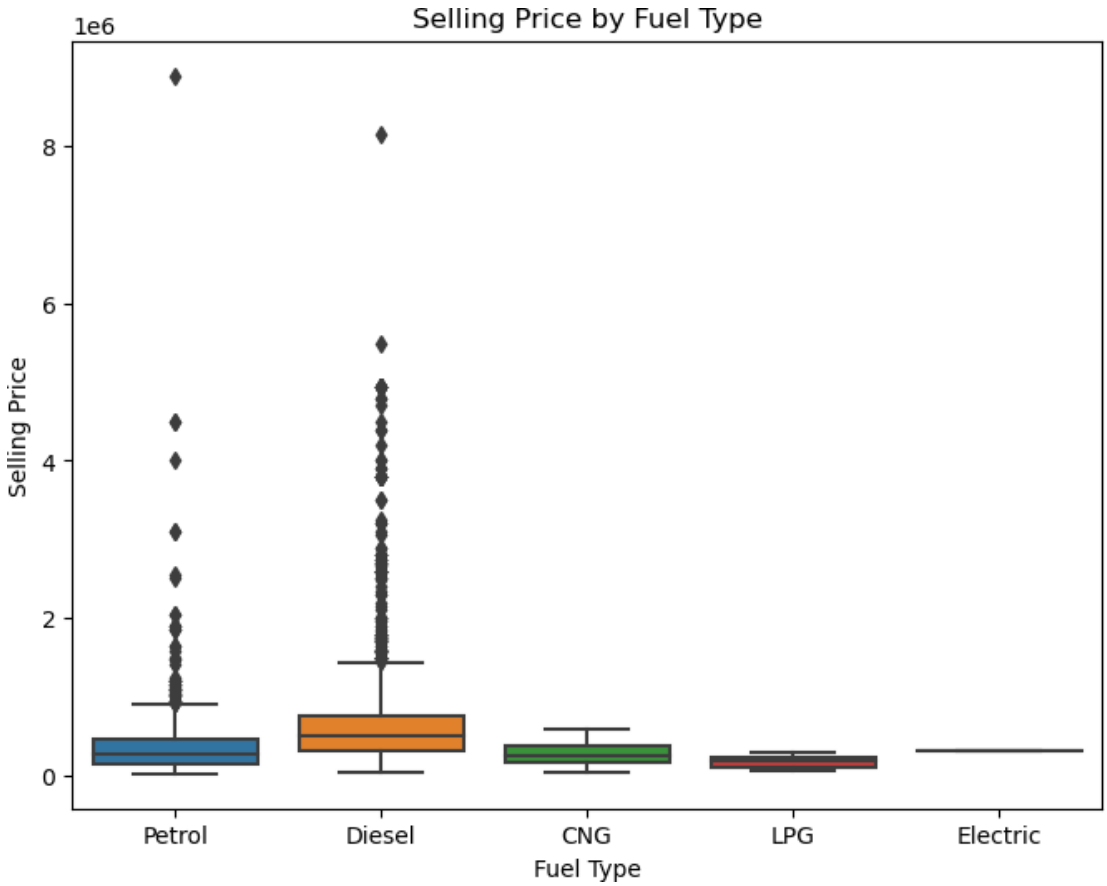
[16]:
```
plt.figure(figsize=(8, 6))
owner_counts = df['owner'].value_counts().reset_index()
owner_counts.columns = ['Owner', 'Count']
categories = owner_counts['Owner']
values = owner_counts['Count']
plt.polar()
plt.fill(categories.tolist() + [categories[0]], values.tolist() + [values[0]],␣
↪alpha=0.25)
plt.xticks(ticks=range(len(categories)), labels=categories)
plt.title('Radar Chart of Owner Types')
plt.show()
```



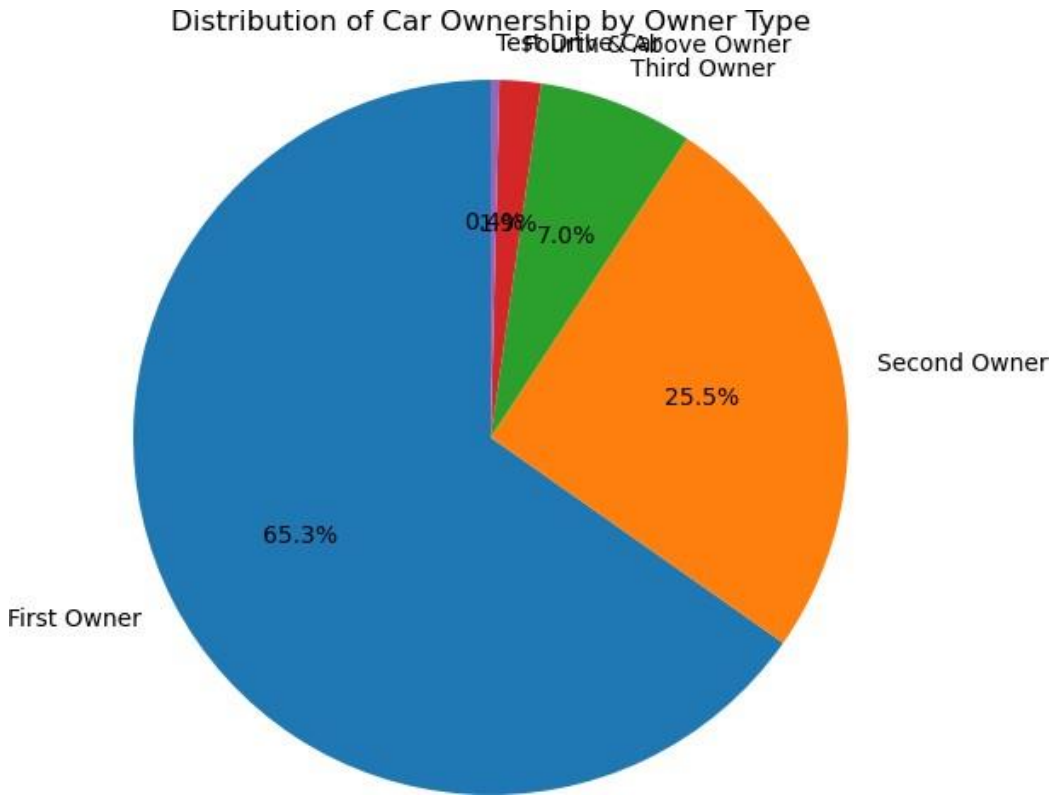
[17]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='km_driven', y='selling_price', hue='transmission')
plt.title('Selling Price vs. Kilometers Driven')
plt.xlabel('Kilometers Driven')
plt.ylabel('Selling Price')
plt.legend(title='Transmission')
plt.show()
```



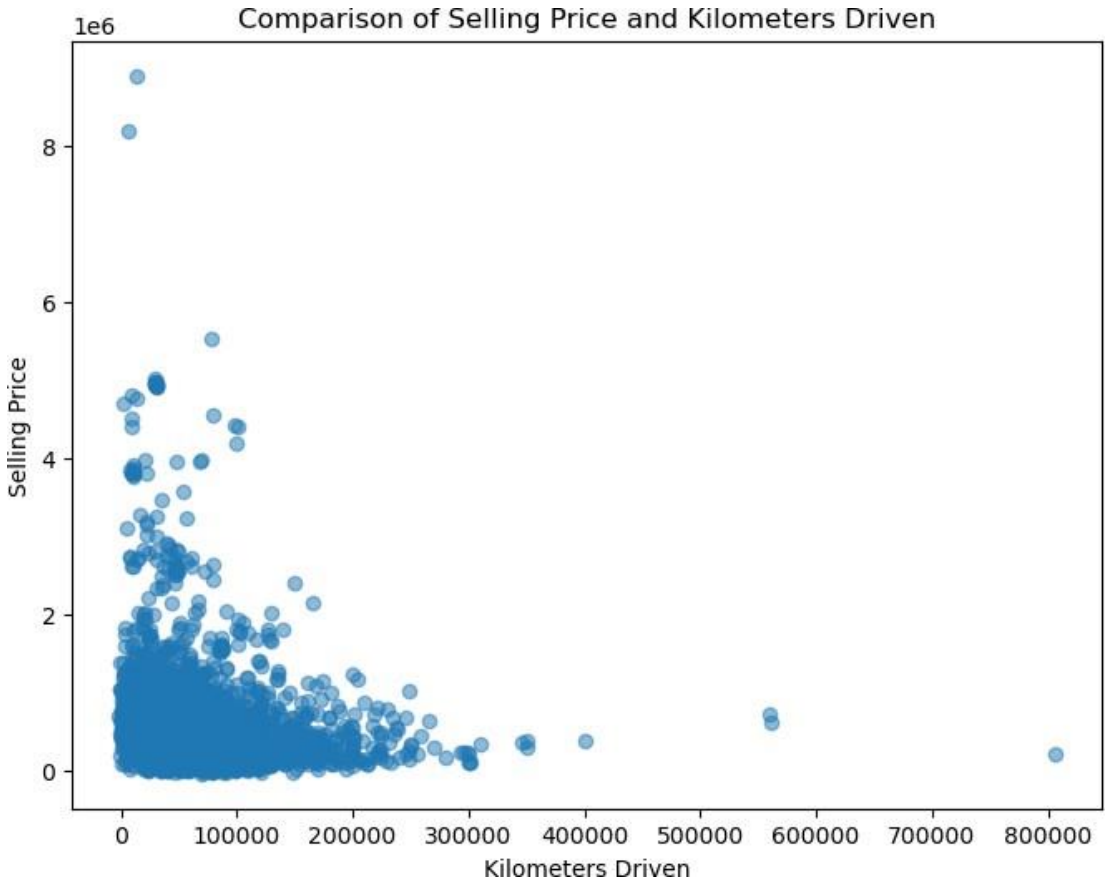
[18]:
```
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x="fuel", y="selling_price")
plt.title("Selling Price by Fuel Type")
plt.xlabel("Fuel Type")
plt.ylabel("Selling Price")
plt.show()
```



[19]:
```
plt.figure(figsize=(8, 6))
owner_counts = df['owner'].value_counts()
plt.pie(owner_counts, labels=owner_counts.index, autopct='%1.1f%%',␣
↪startangle=90)
plt.title('Distribution of Car Ownership by Owner Type')
plt.axis('equal')
plt.show()
```

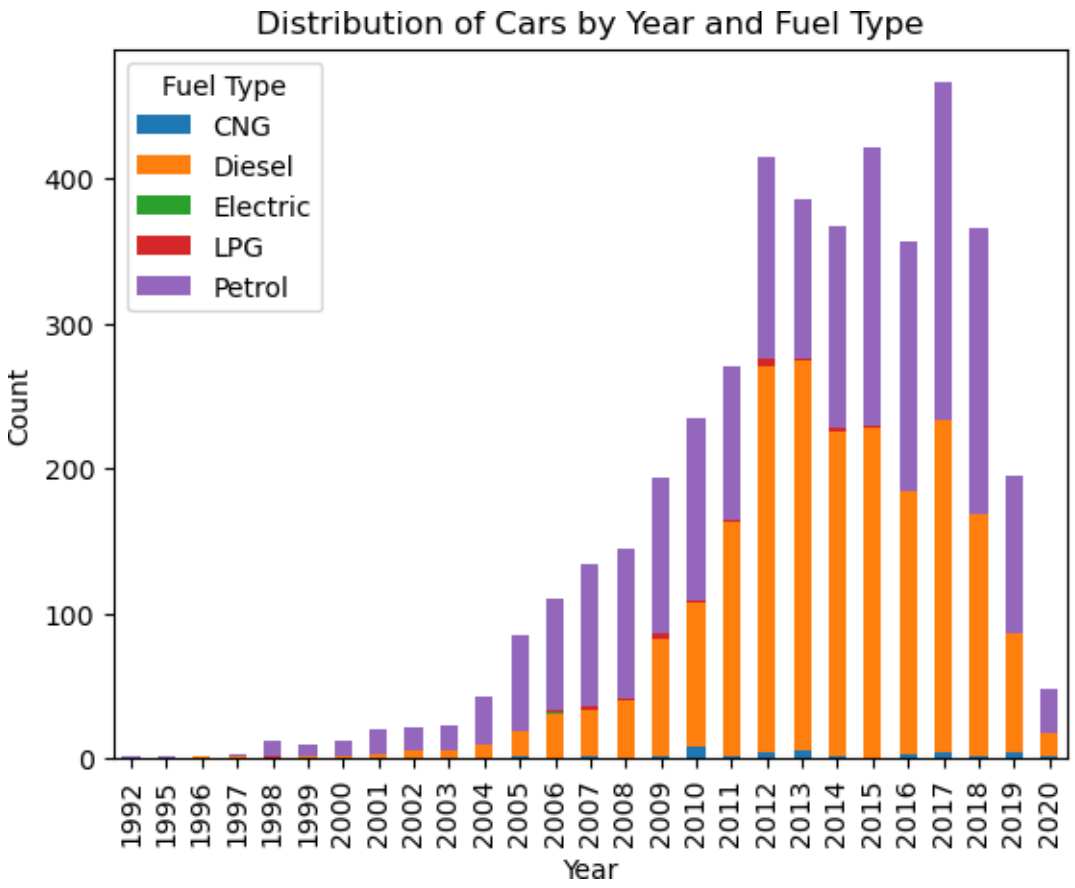


[20]:
```
jittered_km_driven = df["km_driven"] + np.random.normal(0, 1000, len(df))
jittered_selling_price = df["selling_price"] + np.random.normal(0, 50000,␣
↪len(df))

plt.figure(figsize=(8, 6))
plt.scatter(jittered_km_driven, jittered_selling_price, alpha=0.5)
plt.title("Comparison of Selling Price and Kilometers Driven")
plt.xlabel("Kilometers Driven")
plt.ylabel("Selling Price")
plt.show()
```



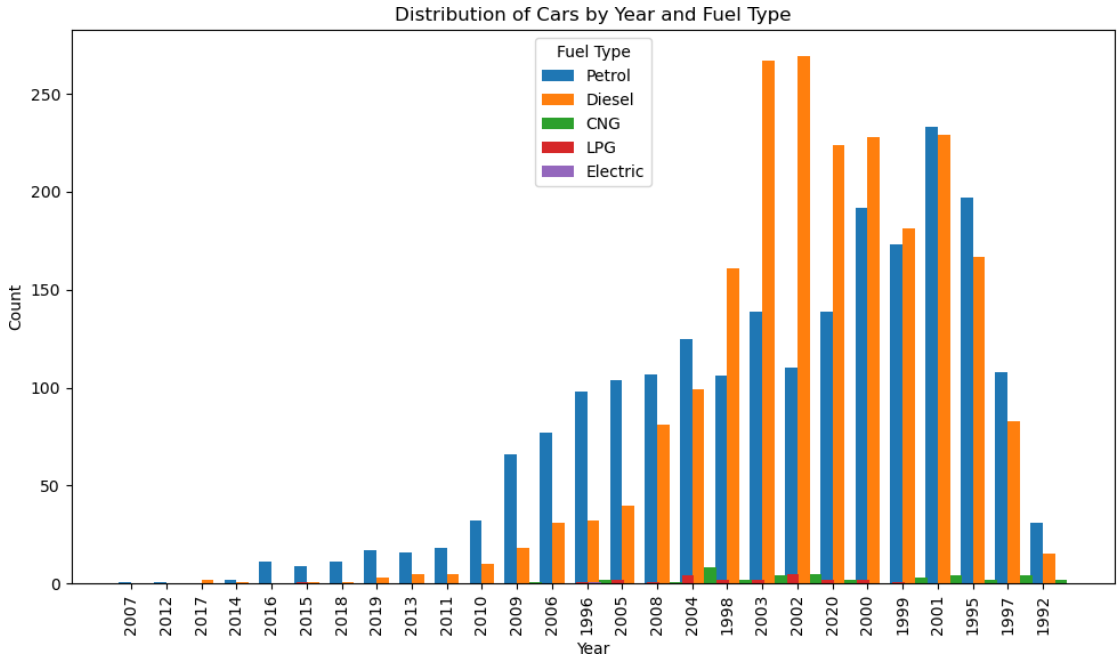
[21]:
```
grouped_data = df.groupby(['year', 'fuel']).size().unstack()

plt.figure(figsize=(10, 6))
grouped_data.plot(kind='bar', stacked=True)
plt.title('Distribution of Cars by Year and Fuel Type')
plt.xlabel('Year')
plt.ylabel('Count')
plt.legend(title='Fuel Type')
plt.show()
```

<Figure size 1000x600 with 0 Axes>

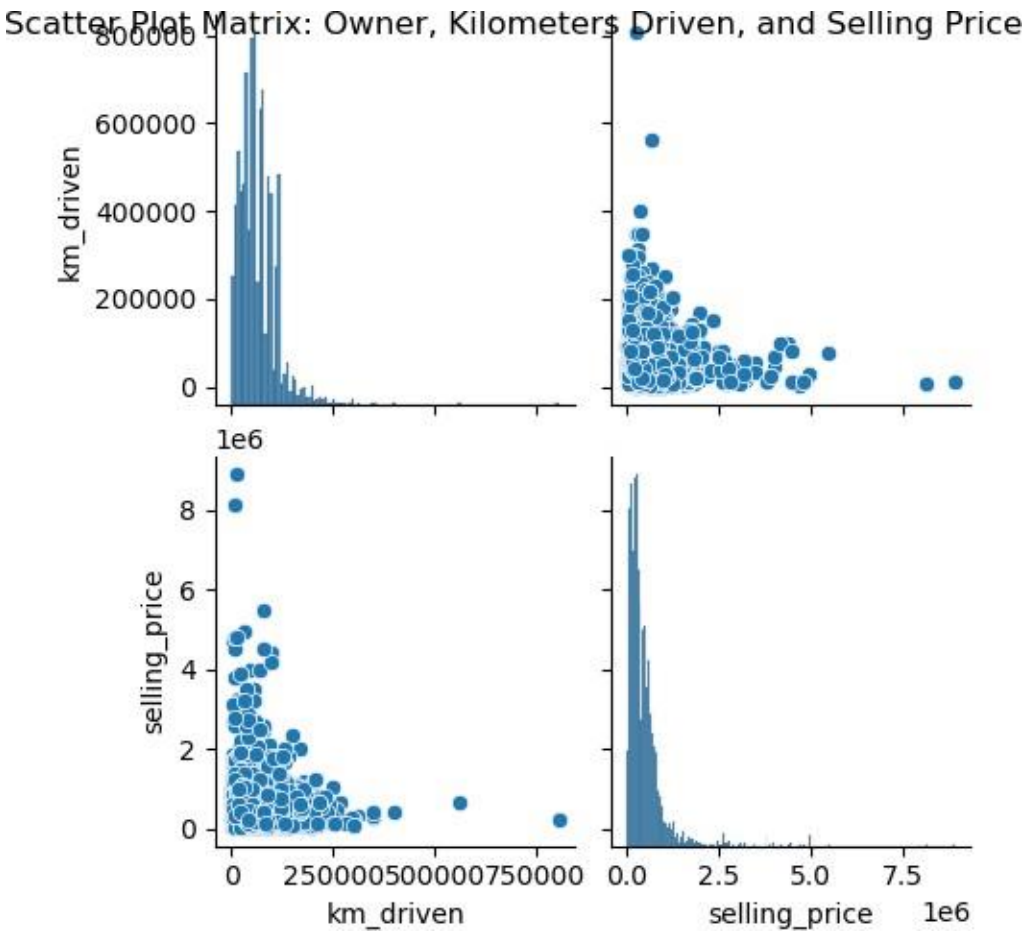


[22]:
```
grouped_data = df.groupby(['year', 'fuel']).size().unstack()

fuel_types = df['fuel'].unique()
years = df['year'].unique()

bar_width = 0.35

index = np.arange(len(years))

plt.figure(figsize=(10, 6))
for i, fuel_type in enumerate(fuel_types):
    plt.bar(index + (i * bar_width), grouped_data[fuel_type], bar_width,␣
 ↪label=fuel_type)

plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Distribution of Cars by Year and Fuel Type')
plt.xticks(index + bar_width / 2, years, rotation='vertical')
plt.legend(title='Fuel Type')
```

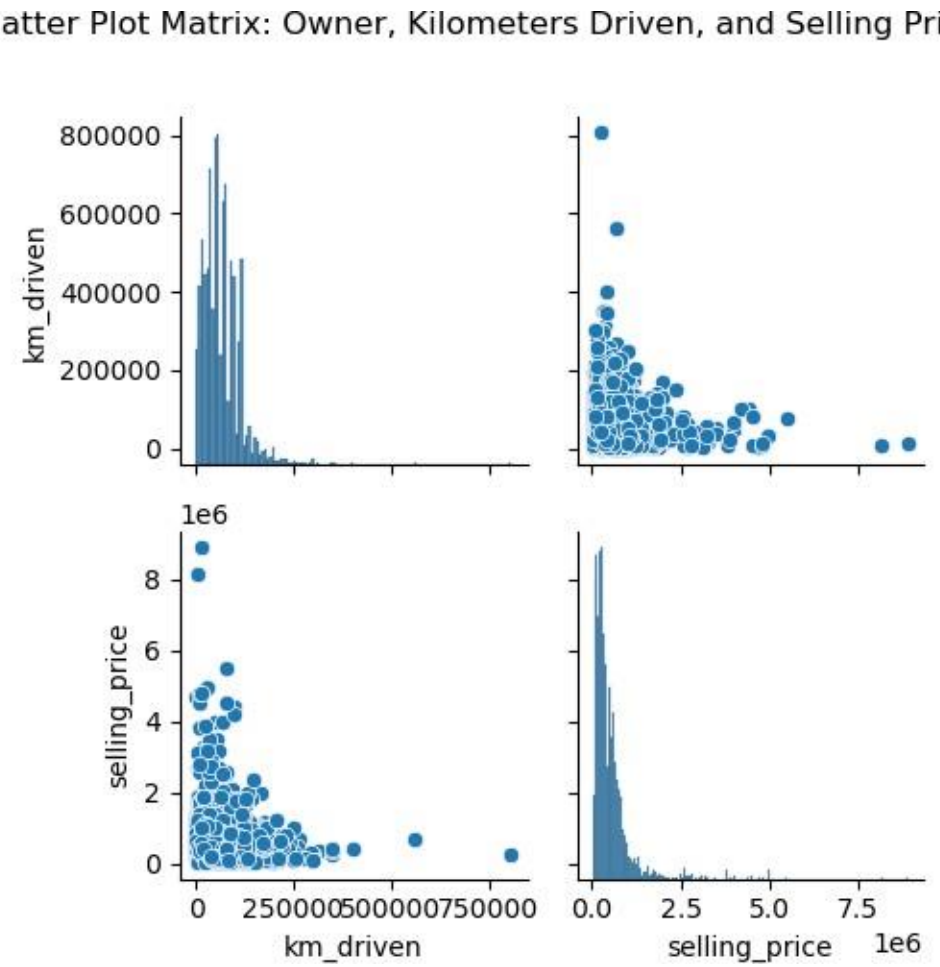
```
plt.tight_layout()    # Adjust the layout to prevent label cutoff
plt.show()
```



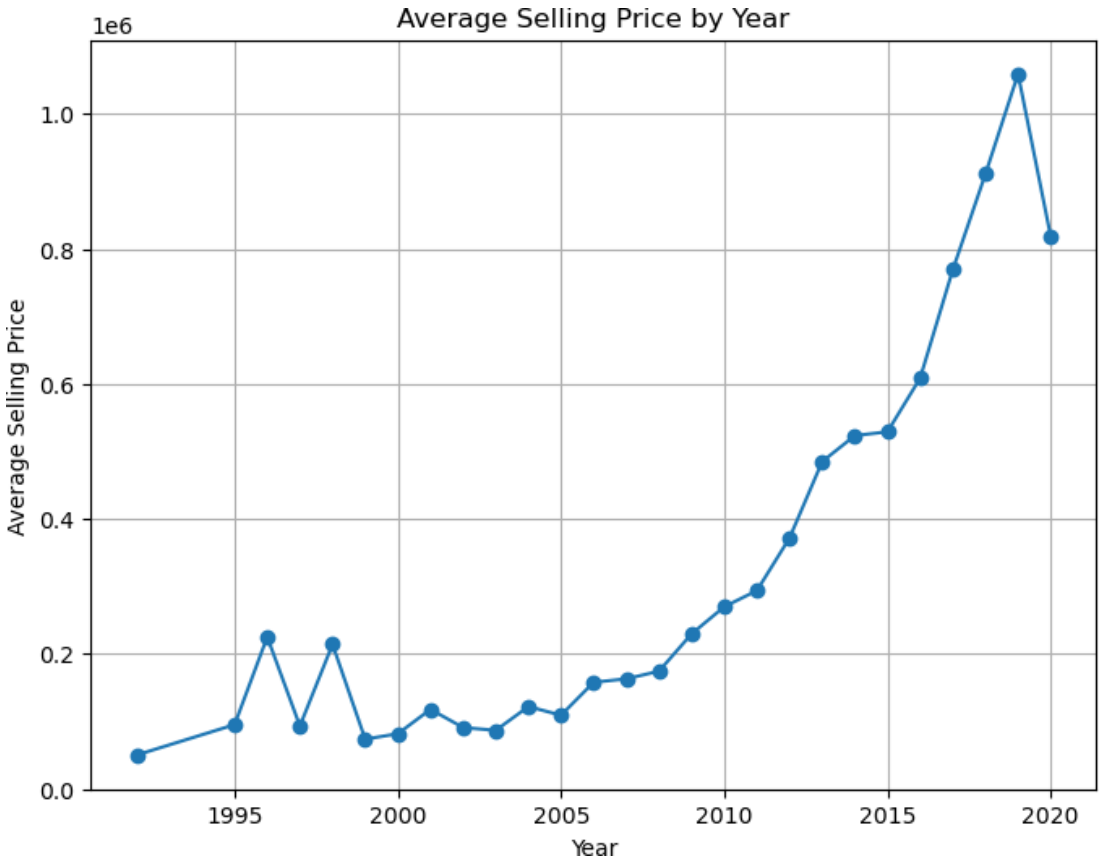
[23]:
```
cols_of_interest = ['owner', 'km_driven', 'selling_price']

sns.pairplot(df[cols_of_interest])
plt.suptitle('Scatter Plot Matrix: Owner, Kilometers Driven, and Selling Price')
plt.show()
```



[24]:
```
cols_of_interest = ['owner', 'km_driven', 'selling_price']

sns.pairplot(df[cols_of_interest])
plt.suptitle('Scatter Plot Matrix: Owner, Kilometers Driven, and Selling␣
↪Price', y=1.02)
plt.tight_layout()
plt.show()
```

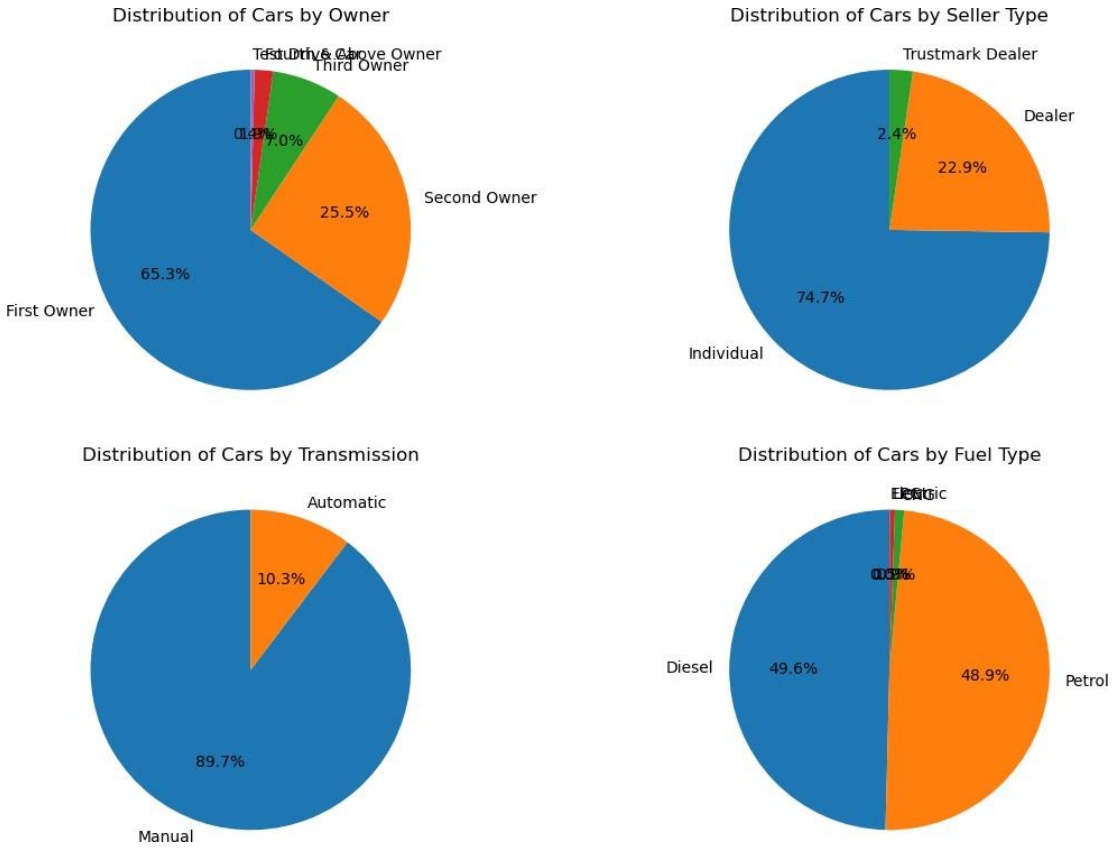


[25]:
```
average_price_by_year = df.groupby('year')['selling_price'].mean()

plt.figure(figsize=(8, 6))
plt.plot(average_price_by_year.index, average_price_by_year.values, marker='o')
plt.title('Average Selling Price by Year')
plt.xlabel('Year')
plt.ylabel('Average Selling Price')
plt.grid(True)
plt.show()
```



[26]:
```
owner_counts = df['owner'].value_counts()
seller_type_counts = df['seller_type'].value_counts()
transmission_counts = df['transmission'].value_counts()
fuel_type_counts = df['fuel'].value_counts()

fig, axes = plt.subplots(2, 2, figsize=(12, 8))

axes[0, 0].pie(owner_counts, labels=owner_counts.index, autopct='%1.1f%%',␣
↪startangle=90)
axes[0, 0].set_title('Distribution of Cars by Owner')

axes[0, 1].pie(seller_type_counts, labels=seller_type_counts.index, autopct='%1.
↪1f%%', startangle=90)
axes[0, 1].set_title('Distribution of Cars by Seller Type')

axes[1, 0].pie(transmission_counts, labels=transmission_counts.index,␣
↪autopct='%1.1f%%', startangle=90)
axes[1, 0].set_title('Distribution of Cars by Transmission')
```

```
axes[1, 1].pie(fuel_type_counts, labels=fuel_type_counts.index, autopct='%1.
↪1f%%', startangle=90)
axes[1, 1].set_title('Distribution of Cars by Fuel Type')

plt.tight_layout()

plt.show()
```



[27]:
```
label_encoder = LabelEncoder()
```

[28]:
```
df['fuel'] = label_encoder.fit_transform(df['fuel'])
df['seller_type'] = label_encoder.fit_transform(df['seller_type'])
df['transmission'] = label_encoder.fit_transform(df['transmission'])
df['owner'] = label_encoder.fit_transform(df['owner'])
```

[29]:
```
df.head()
```

[29]:

	name	year	selling_price	km_driven	fuel	\
0	Maruti 800 AC	2007	60000	70000	4	

```
1   Maruti Wagon R LXI Minor  2007     135000          50000    4
2        Hyundai Verna 1.6 SX  2012     600000         100000    1
3     Datsun RediGO T Option  2017     250000          46000    4
4      Honda Amaze VX i-DTEC  2014     450000         141000    1

   seller_type  transmission  owner
0            1             1      0
1            1             1      0
2            1             1      0
3            1             1      0
4            1             1      2
```

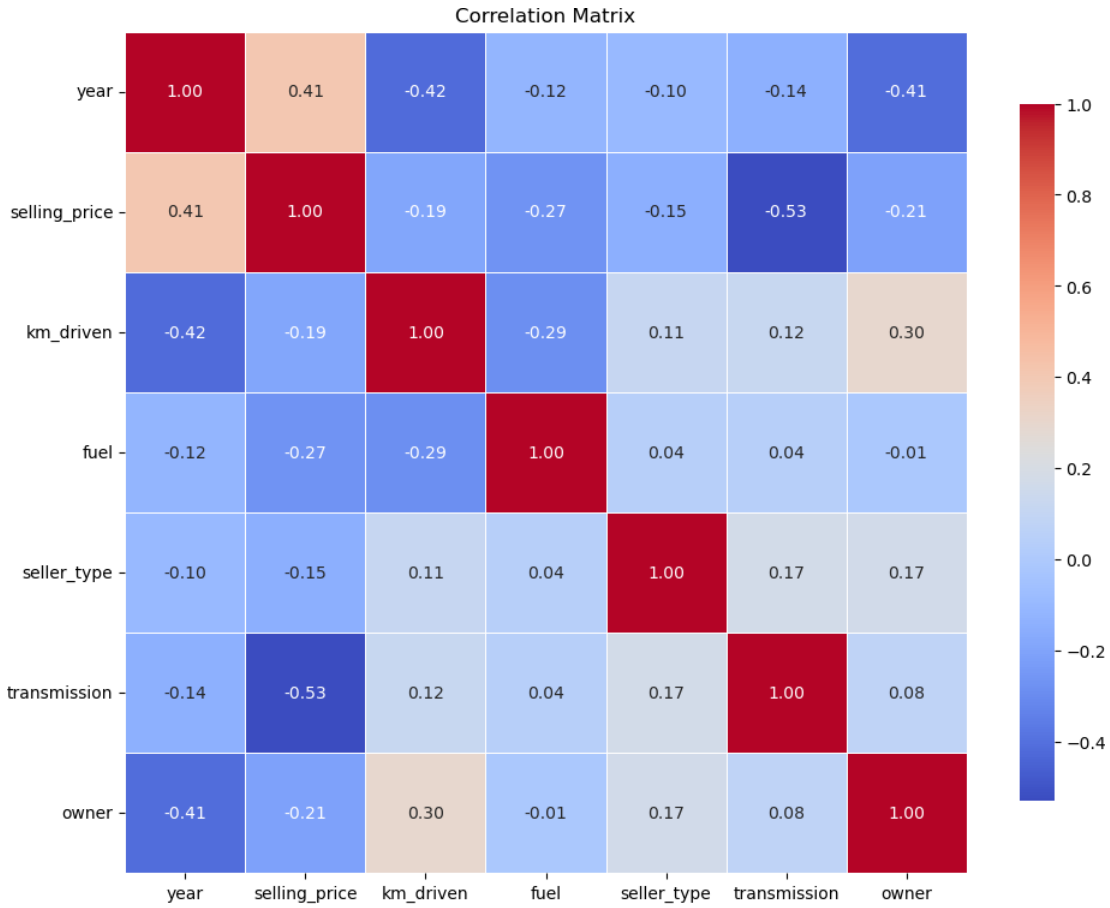
[30]:
```
df.describe()
```

[30]:
```
              year  selling_price  km_driven         fuel  seller_type  \
count  4340.000000   4.340000e+03  4340.000000  4340.000000  4340.000000
mean   2013.090783   5.041273e+05  66215.777419     2.469124     0.794470
std       4.215344   5.785487e+05  46644.102194     1.508435     0.458629
min    1992.000000   2.000000e+04      1.000000     0.000000     0.000000
25%    2011.000000   2.087498e+05  35000.000000     1.000000     1.000000
50%    2014.000000   3.500000e+05  60000.000000     1.000000     1.000000
75%    2016.000000   6.000000e+05  90000.000000     4.000000     1.000000
max    2020.000000   8.900000e+06 806599.000000     4.000000     2.000000

       transmission         owner
count   4340.000000   4340.000000
mean       0.896774      0.820276
std        0.304289      1.233494
min        0.000000      0.000000
25%        1.000000      0.000000
50%        1.000000      0.000000
75%        1.000000      2.000000
max        1.000000      4.000000
```

[31]:
```
correlation_matrix = df.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True, fmt='.
↪2f',
            linewidths=0.5, cbar_kws={'shrink': 0.8})
plt.title('Correlation Matrix')

plt.yticks(rotation=0)

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

C:\Users\admin\AppData\Local\Temp\ipykernel_10480\460784327.py:1: FutureWarning:The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.

```
  correlation_matrix = df.corr()
```



3 Market Segmentation

Segment Extraction

K means is one of the most popular Unsupervised Machine Learning Algorithms Used for Solving Classification Problems. K Means segregates the unlabeled data into various groups, called clusters, based on having similar features, common patterns. Suppose we have N number of Unlabeled Multivariate Datasets of various features like water-availability, price, city etc. from our dataset. The technique to segregate Datasets into various groups, on the basis of having similar features and characteristics, is called Clustering. The groups being Formed are known as Clusters. Clustering is being used in Unsupervised Learning Algorithms in Machine Learning as it can segregate multivariate data into various groups, without any supervisor, on the basis of a common pattern hidden inside the datasets. In the Elbow method, we are actually varying the number of clusters

(K) from 1 – 10. For each value of K, we are calculating WCSS (Within-Cluster Sum of Square). WCSS is the sum of squared distance between each point and the centroid in a cluster. When we plot the WCSS with the K value, the plot looks like an Elbow.

As the number of clusters increases, the WCSS value will start to decrease. WCSS value is largest when K = 1. When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape. From this point, the graph starts to move almost parallel to the X-axis. The K value corresponding to this point is the optimal K value or an optimal number of clusters.

K-Means

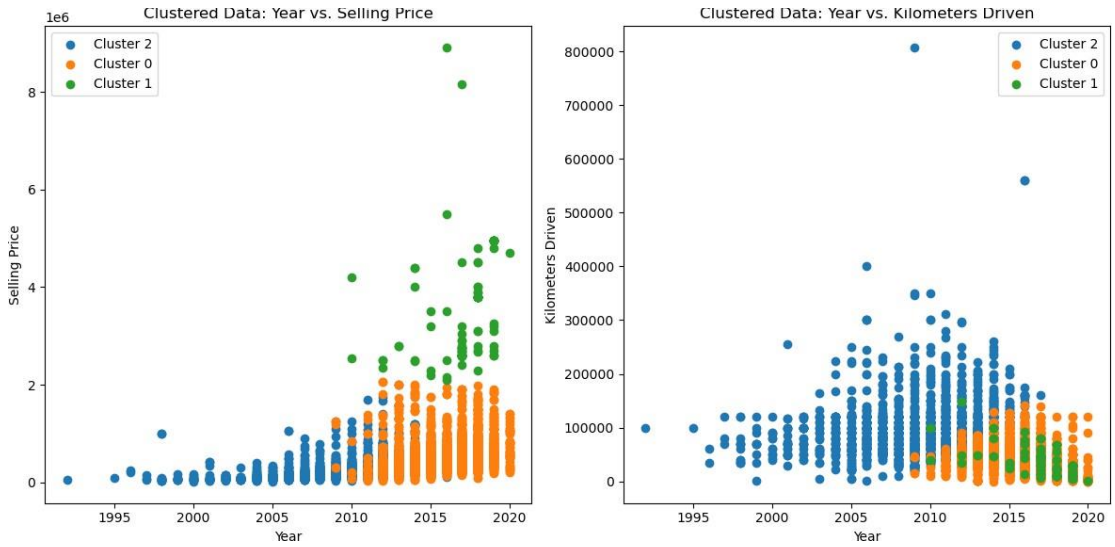
[32]:
```
features = ['year', 'selling_price', 'km_driven']

subset_df = df[features]

scaler = StandardScaler()
scaled_features = scaler.fit_transform(subset_df)

num_clusters = 3

kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(scaled_features)

cluster_labels = kmeans.labels_

df['cluster'] = cluster_labels

cluster_centers = scaler.inverse_transform(kmeans.cluster_centers_)
for i, center in enumerate(cluster_centers):
    print(f"Cluster {i+1} Center: {center}")

cluster_counts = df['cluster'].value_counts()
print("\nCluster Counts:")
print(cluster_counts)
```

```
C:\Users\admin\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(

Cluster 1 Center: [   2015.78224456 579554.47822446   42665.40075377]
Cluster 2 Center: [2.01698947e+03 3.48434737e+06 3.66715684e+04]
Cluster 3 Center: [   2009.43026387 254670.6725902    98011.68443726]

Cluster Counts:
0    2388
2    1857
1      95
```

Name: cluster, dtype: int64

[33]:
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
for cluster_label in df['cluster'].unique():
    cluster_data = df[df['cluster'] == cluster_label]
    plt.scatter(cluster_data['year'], cluster_data['selling_price'],␣
 ↪label=f'Cluster {cluster_label}', cmap='viridis')
plt.xlabel('Year')
plt.ylabel('Selling Price')
plt.title('Clustered Data: Year vs. Selling Price')
plt.legend()

plt.subplot(1, 2, 2)
for cluster_label in df['cluster'].unique():
    cluster_data = df[df['cluster'] == cluster_label]
    plt.scatter(cluster_data['year'], cluster_data['km_driven'],␣
 ↪label=f'Cluster {cluster_label}', cmap='viridis')
plt.xlabel('Year')
plt.ylabel('Kilometers Driven')
plt.title('Clustered Data: Year vs. Kilometers Driven')
plt.legend()

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

C:\Users\admin\AppData\Local\Temp\ipykernel_10480\4293902767.py:6: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(cluster_data['year'], cluster_data['selling_price'],
label=f'Cluster {cluster_label}', cmap='viridis')
C:\Users\admin\AppData\Local\Temp\ipykernel_10480\4293902767.py:15: UserWarning:
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(cluster_data['year'], cluster_data['km_driven'], label=f'Cluster
{cluster_label}', cmap='viridis')



Based on the K-means clustering results, the dataset has been divided into three clusters (Cluster 1, Cluster 2, and Cluster 3).

Cluster 1:

Center: [2015, 579,554.48, 42,665.40] Cluster Count: 2,388 In Cluster 1, the average values for the features are as follows:

Year: Around 2015 Selling Price: Around 579,554.48 Kilometers Driven: Around 42,665.40 This cluster contains the highest number of data points (2,388). Vehicles in this cluster tend to be relatively newer (around 2015), have a moderately high selling price (around 579,554.48), and have lower kilometers driven (around 42,665.40).

Cluster 2:

Center: [2016, 3,484,347.37, 36,671.57] Cluster Count: 95 In Cluster 2, the average values for the features are as follows:

Year: Around 2016 Selling Price: Around 3,484,347.37 Kilometers Driven: Around 36,671.57 This cluster contains a relatively small number of data points (95). Vehicles in this cluster are typically newer (around 2016), have a significantly higher selling price (around 3,484,347.37), and have relatively lower kilometers driven (around 36,671.57).

Cluster 3:

Center: [2009, 254,670.67, 98,011.68] Cluster Count: 1,857 In Cluster 3, the average values for the features are as follows:

Year: Around 2009 Selling Price: Around 254,670.67 Kilometers Driven: Around 98,011.68 This cluster contains a moderate number of data points (1,857). Vehicles in this cluster tend to be older (around 2009), have a lower selling price (around 254,670.67), and have relatively higher kilometers driven (around 98,011.68).

4 Summary

Based on the data and information analyzed, the market segmentation of Electric Vehicles (EVs) can be summarized as follows:

Fuel Type Distribution: Among the available fuel types, the dataset shows that EVs represent a very small portion, with only one record labeled as “Electric.” EVs constitute a niche segment within the dataset.

Cluster Analysis: Through K-means clustering, the dataset was divided into three clusters:

Cluster 1: This cluster comprises the majority of data points (2,388) and represents vehicles that are relatively newer (around 2015), have a moderately high selling price (around 579,554.48), and lower kilometers driven (around 42,665.40).

Cluster 2: This cluster contains a small number of data points (95) and represents vehicles that are even newer (around 2016.99), have significantly higher selling prices (around 3,484,347.37), and relatively lower kilometers driven (around 36,671.57).

Cluster 3: This cluster includes a moderate number of data points (1,857) and represents vehicles that are older (around 2009.43), have lower selling prices (around 254,670.67), and relatively higher kilometers driven (around 98,011.68).

Market Share: Based on the clustering results, it appears that EVs are not dominant in the overall market segment. However, it’s important to note that the dataset might not fully represent the EV market, as there is only one record labeled as “Electric.” Therefore, further analysis with a larger and more representative dataset is recommended to gain a comprehensive understanding of the EV market segmentation.

We should bear in mind that this summary is based on the provided dataset and the clustering results. The actual market segmentation and trends may vary based on additional factors, such as geographic location, specific market dynamics, and the availability and adoption of EVs in different regions.