Market Segmentation Analysis of Electric Vehicles Market in India

Contributors

Kanna Lokesh

Neha Dwivedi

Siddhesh Salfale

Amrit Kumar Maurya

Viddyesh Dhobale

Kaveti Venkata Nikhil

Market Segmentation Analysis Of Electric Vehicle Market in India

Problem Statement:

Task is to analyze the Electric Vehicles Market in India using Segmentation analysis and come up with a feasible strategy to enter the market, targeting the segments most likely to use their product in terms of Geographic, Demographic, Psychographic, and Behavioral. In this report we analyze the Electric Vehicles Market in India using segments such as region, price, charging facility, type of vehicles (e.g., 2 wheelers, 3 wheelers, 4 wheelers etc.), retail outlets, manufacturers, body type (e.g., Hatchback, Sedan, SUV, Autorickshaw etc.), safety, plug types and much more.

Fermi Estimation:

Wild Guess: Around 8-10% people will have electric vehicles by the end of 2023 in India.

Educated Guess: Employment rate = it is the ratio of number of available labor force to the population of People in the working age.

We think there are about 1.5 billion Indians in the world. Let's assume the only people over18 and under 60 works, assuming that they account for around 60% of the population then that would make 0.9 billion Indians in the working class.

Out of the 0.9 billion people not all are employed, assuming only 2023 had 45%employment rate that would bring the number around 405 million. Since, not everyone can afford an electric vehicle, let’s assume only people above middle class can afford an electric vehicle, that would be 40 million. Not everyone buysan electric vehicle. Let’s assume out of these 40 million only 10 million are willing to buy an electric vehicle. Variables and Formulas: Let E(x) be the employment rate of the year x (in %).

Let P(x) be the population of the year x.

Let A(x) be the number of available Labor in the year x.

Let r be the ratio of Indians between the age of 18 and 60 to the total population of India.

E(x) = (A(x)*100)/(P(x)*r) This formula will formulate the Employment ratio for the year x.

Gathering More Information: Estimation for the population of the year 2022 can be obtained by the increase in population each year P (2019) = 1.3676 billion P (2020) = 1.3786 billion P (2021) = 1.39199 billion P (2020)-P (2019) = 11million P (2021)-P (2020) = 13.39 million the mean would be 12.195 million thus P (2022) = 1.44185 billion assuming A(x) is constant every year= 471,688,990r=0.6C=0.75 E (2022) = (471,688,990/(1,441,850,000*0.6))*0.75E (2022) = 42% Conclusion: By this analysis, we conclude that by the end of the year 2024 there would a Employment rate of 42%.

That would make 42% of 405 million i.e., 170 million. Out of these 170 million only 10% afford EV'S. So around 17 million people will have EV's by the end of 2024".

Data Pre-processing:

In this step we are going to convert the raw data into a meaningful data which will be used for clustering.

Dataset Origin :

https://www.kaggle.com/datasets/kkhandekar/quickest-electric-cars-ev-database

Initially the dataset looks like,

```
data.head()
```

	Name	Subtitle	Acceleration	TopSpeed	Range	Efficiency	FastChargeSpeed	Drive	NumberofSeats	PriceinGermany	PriceinUK
0	Tesla Roadster	Battery Electric Vehicle | 200 kWh	2.1 sec	410 km/h	970 km	206 Wh/km	920 km/h	All Wheel Drive	4	€215,000	£189,000
1	Tesla Model X Plaid	Battery Electric Vehicle | 90 kWh	2.6 sec	262 km/h	455 km	198 Wh/km	680 km/h	All Wheel Drive	7	€116,990	£110,980
2	Porsche Taycan Turbo S	Battery Electric Vehicle | 83.7 kWh	2.8 sec	260 km/h	390 km	215 Wh/km	860 km/h	All Wheel Drive	4	€186,336	£138,830
3	Porsche Taycan Turbo S Cross Turismo	Battery Electric Vehicle | 83.7 kWh	2.9 sec	250 km/h	380 km	220 Wh/km	790 km/h	All Wheel Drive	4	€187,746	£139,910
4	Tesla Cybertruck Tri Motor	Battery Electric Vehicle | 200 kWh	3.0 sec	210 km/h	750 km	267 Wh/km	710 km/h	All Wheel Drive	7	€75,000	£68,000

The information about the dataset,

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 179 entries, 0 to 178
Data columns (total 11 columns):
Name                179 non-null object
Subtitle            179 non-null object
Acceleration        179 non-null object
TopSpeed            179 non-null object
Range               179 non-null object
Efficiency          179 non-null object
FastChargeSpeed     179 non-null object
Drive               179 non-null object
NumberofSeats       179 non-null int64
PriceinGermany      167 non-null object
PriceinUK           135 non-null object
dtypes: int64(1), object(10)
memory usage: 15.5+ KB
```

The above information seems that even though the data is numerical but the type of the data is object. So we need to pre-process the data.

Required Libraries:

In order to perform EDA and Clustering on collected data, the following libraries are used.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import LabelEncoder
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from bioinfokit.visuz import cluster
from sklearn.cluster import KMeans
from bioinfokit.visuz import cluster
from yellowbrick.cluster import KElbowVisualizer
from collections import Counter
from yellowbrick.cluster import SilhouetteVisualizer
```

We are going to pre-process the data by using the below steps,

```
data['accelaration'] = data['Acceleration'].str.replace("sec"," ")
data['accelaration'] = data['accelaration'].astype('float')
```

```
data['Topspeed'] = data['TopSpeed'].str.replace('km/h',' ')
data['Topspeed'] = data['Topspeed'].astype('float')
```

```
data['range'] = data['Range'].str.replace('km',' ')
data['range'] = data['range'].astype('float')
```

```
data['efficiency'] = data['Efficiency'].str.replace('Wh/km',' ')
data['efficiency'] = data['efficiency'].astype('float')
```

```
data['Fastchargespeed'] = data['FastChargeSpeed'].str.replace('km/h',' ')
```

```
data['Fastchargespeed'] = data['Fastchargespeed'].str.replace('-','440')
```

```
data['Fastchargespeed'] = data['Fastchargespeed'].astype('float')
```

```
data['PriceinGermany'] = data['PriceinGermany'].fillna(method='ffill')
```

```
data['PriceinGermany'] = data['PriceinGermany'].str.replace('€',' ')
```

```
data['PriceinGermany'] = data['PriceinGermany'].str.replace(',','')
```

```
data['PriceinGermany'] = data['PriceinGermany'].astype('float')
```

```
data['PriceinUK'] = data['PriceinUK'].fillna(method='ffill')
```

```
data['PriceinUK'] = data['PriceinUK'].str.replace('£','')
data['PriceinUK'] = data['PriceinUK'].str.replace(',','')
data['PriceinUK'] = data['PriceinUK'].astype('float')
```

The Final Preprocessed Data looks like below as shown in figure

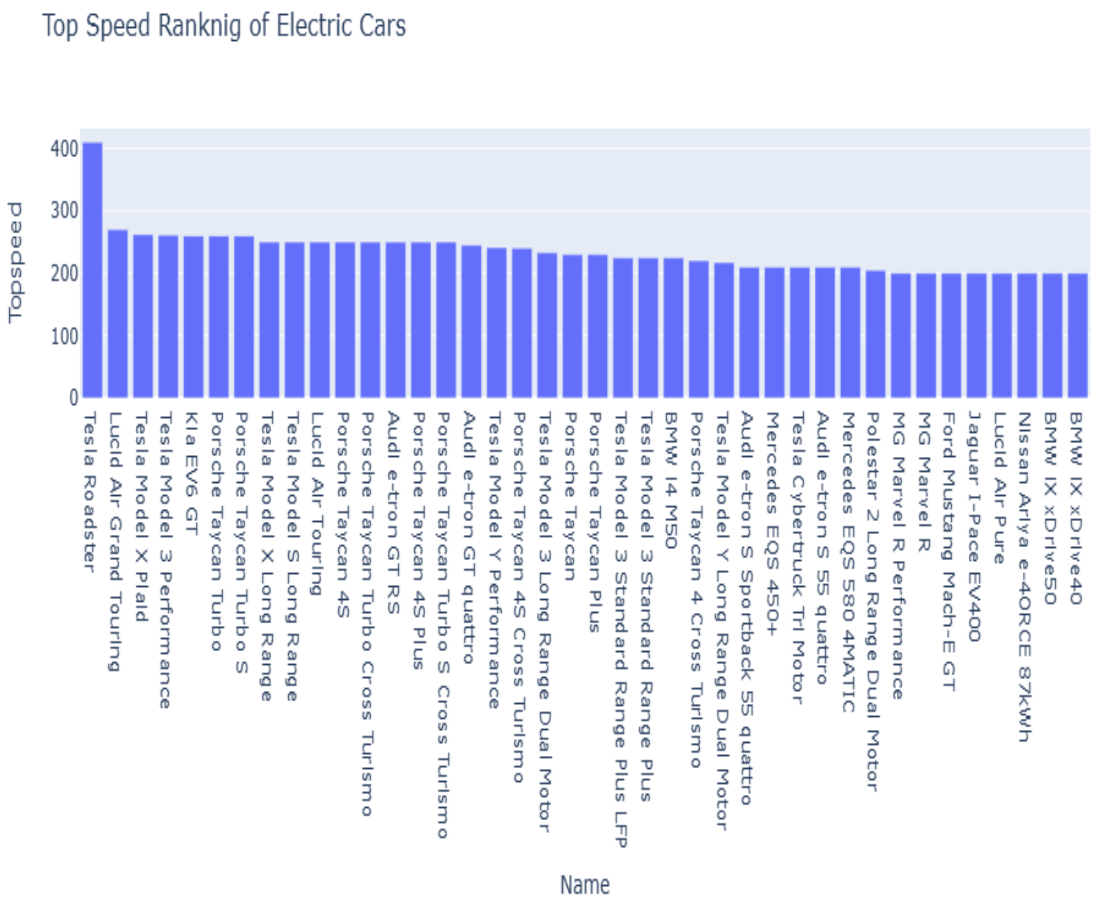
```
data.head()
```

	Name	Drive	NumberofSeats	PriceinGermany	PriceinUK	subtitle	capacity	accelaration	Topspeed	range	efficiency	Fastchargespeed
0	Tesla Roadster	All Wheel Drive	4	215000.0	189000.0	Battery Electric Vehicle	200.0	2.1	410.0	970.0	206.0	920.0
1	Tesla Model X Plaid	All Wheel Drive	7	116990.0	110980.0	Battery Electric Vehicle	90.0	2.6	262.0	455.0	198.0	680.0
2	Porsche Taycan Turbo S	All Wheel Drive	4	186336.0	138830.0	Battery Electric Vehicle	83.7	2.8	260.0	390.0	215.0	860.0
3	Porsche Taycan Turbo S Cross Turismo	All Wheel Drive	4	187746.0	139910.0	Battery Electric Vehicle	83.7	2.9	250.0	380.0	220.0	790.0
4	Tesla Cybertruck Tri Motor	All Wheel Drive	7	75000.0	68000.0	Battery Electric Vehicle	200.0	3.0	210.0	750.0	267.0	710.0

Exploratory Data Analysis:

In data mining, Exploratory Data Analysis (EDA) is an approach to analyzing datasets to summarize their main characteristics, often with visual methods. EDA is used for seeing what the data can tell us before the modelling task.

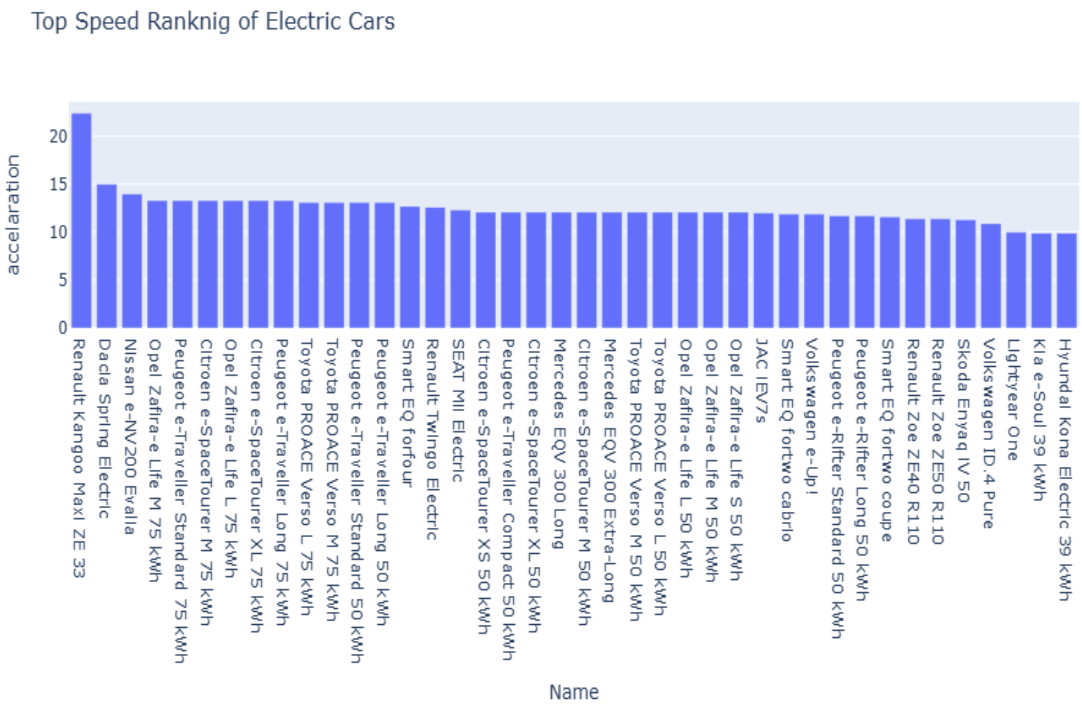
Top Speed Ranking of Electric cars:



Observations from above Graph:

The above graph shows that the highest speed ranking of top 40 electric cars. Among them the top 5 ranked cars are *Tesla Roadster, lucid air grand touring, Tesla Model x Plaid, Tesla Model 3 performance, Kia EV6 GT* .

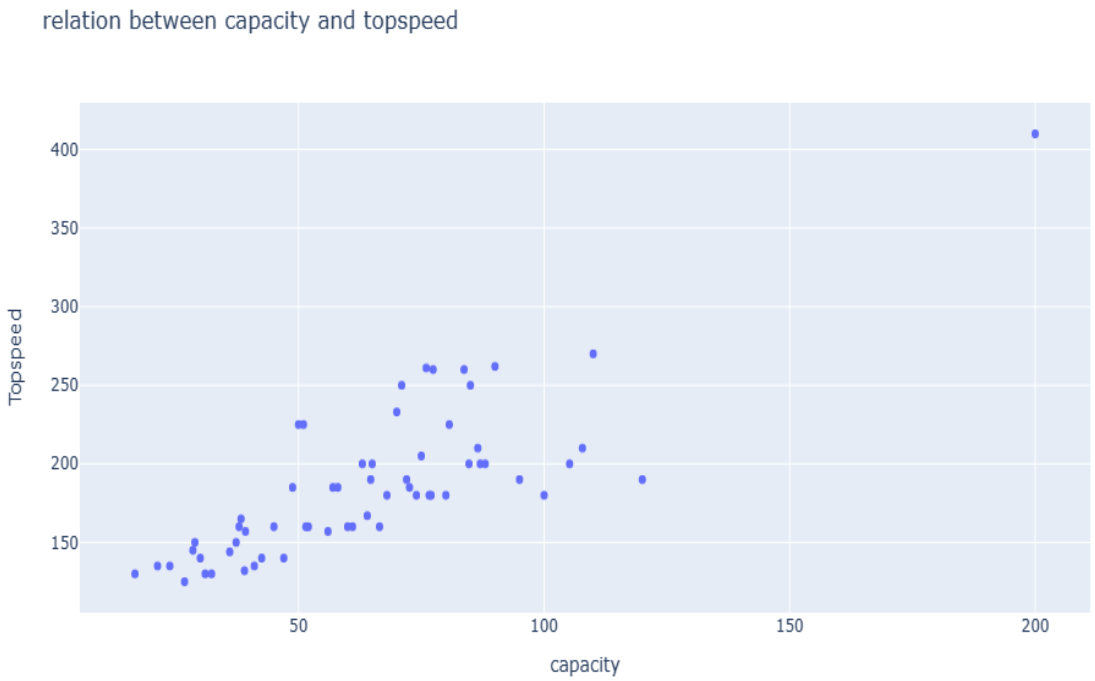
Top Speed Ranking of Electric cars By Acceleration:



Observation from above Graph:

The above graph shows that the top speed ranking of electric vehicles by acceleration. Among them the top 5 electric vehicles are *Renault Kangoo Maxi ZE 33, Dacia Spring Electric, Nissan e-NV200 Evalia, Opel Zafira-e Life M 75 kWh, Peugeot e-Traveller Standard 75 kWh.*

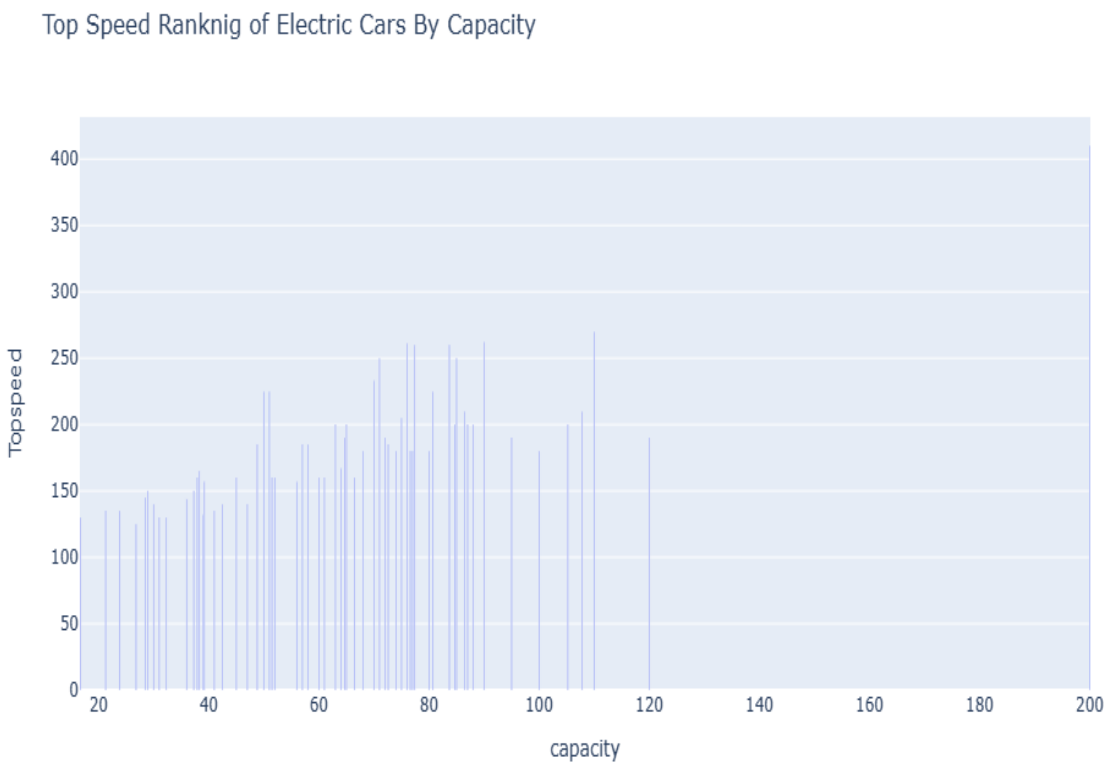
Relation Between Capacity And Top Speed:



Observations from above Graph:

From the above graph there is a linear relation between TopSpeed and Capacity that describes If Capacity increases then the topspeed also increases.

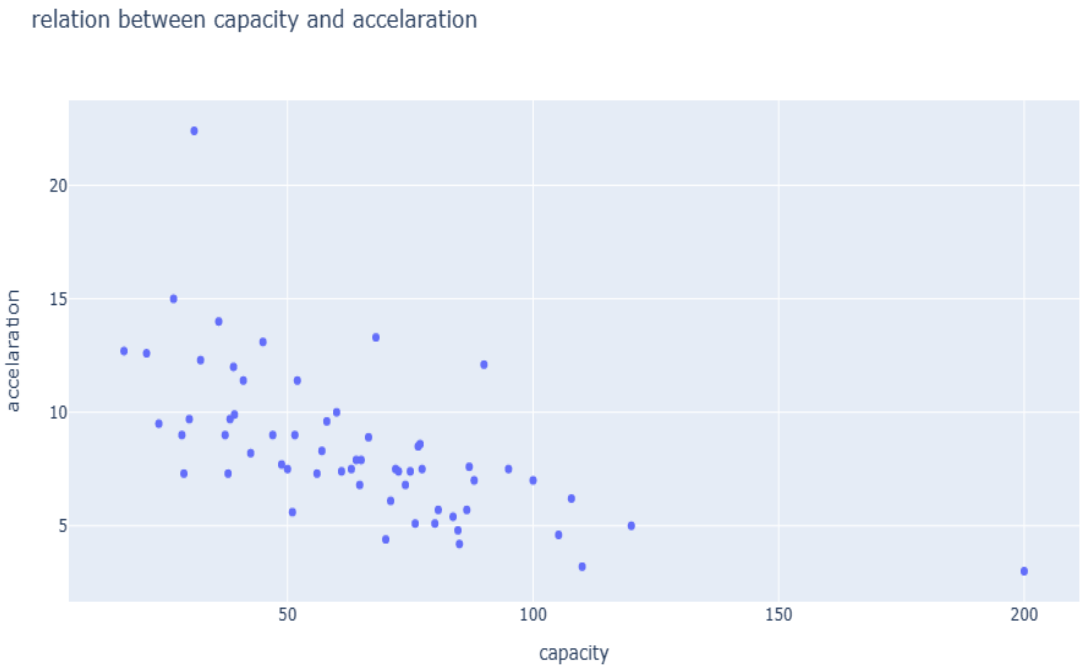
Top Speed Ranking of Electric Cars By Capacity:



Observations from above Graph:

The above graph shows that the Top Speed ranking of Electric Cars By Capacity which indicates that the there is a linear relationship between the speed and capacity and for the capacity of 200 has topspeed 420.

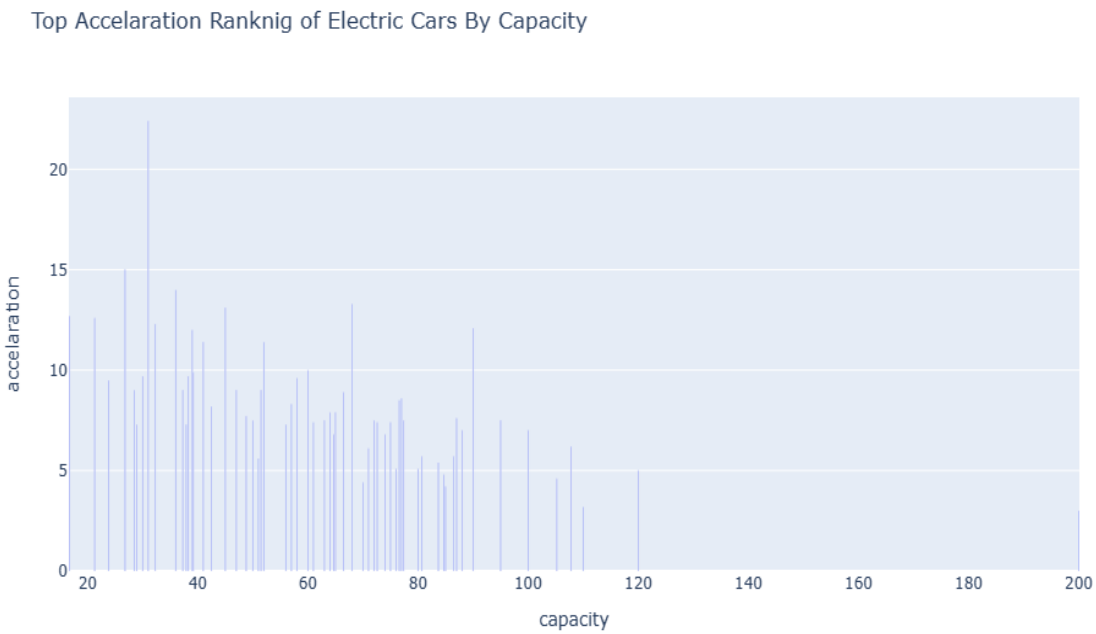
Relation Between Capacity And Acceleration:



Observations from above Graph:

From the above graph there is a linear relation between Acceleration and Capacity that describes If Capacity increases then the topspeed decreases. That it shows a negative correlation between acceleration and capacity.

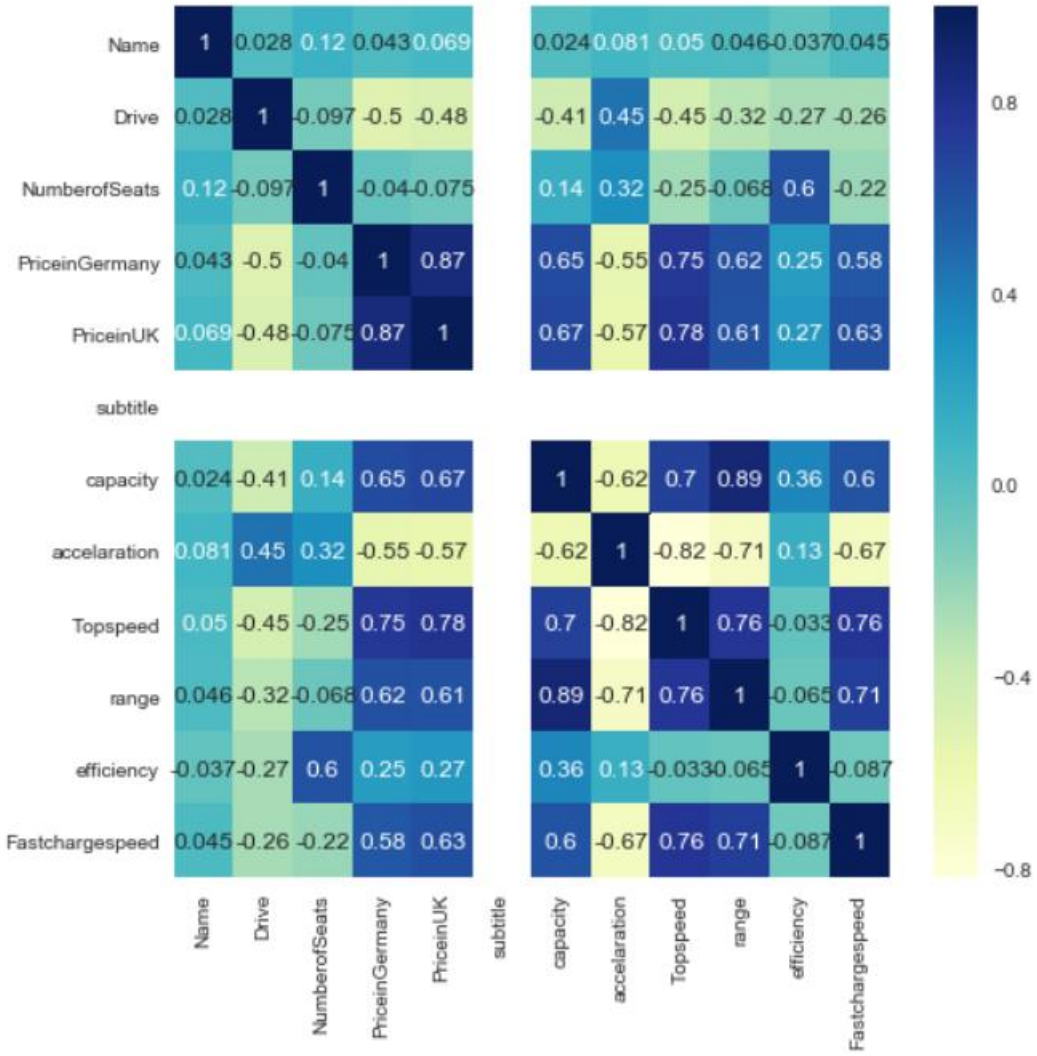
Top Acceleration Ranking of Electric Cars By Capacity:



Observations from above Graph:

The above graph shows that the Top acceleration ranking of Electric Cars By Capacity which indicates that the relation between the acceleration and the capacity and for the capacity of 20 has acceleration 12. There is a negative correlation between acceleration and the capacity.

Correlation Matrix:



The correlation matrix define that the relation among all the variables in the data.

Segmentation Approaches:

Clustering:

Clustering is a type of unsupervised learning method of machine learning. In the unsupervised learning method, the inferences are drawn from the data sets which do not contain labelled output variable. It is an exploratory data analysis technique that allows us to analyze the multivariate data sets.

KMeans Clustering:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

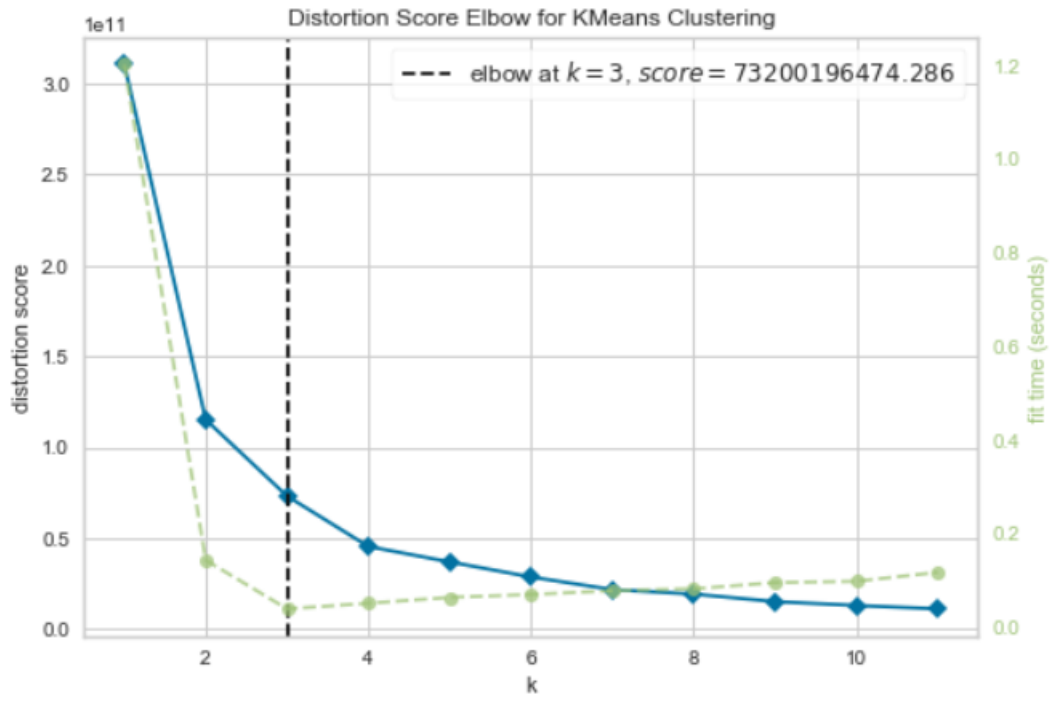
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Elbow Method:

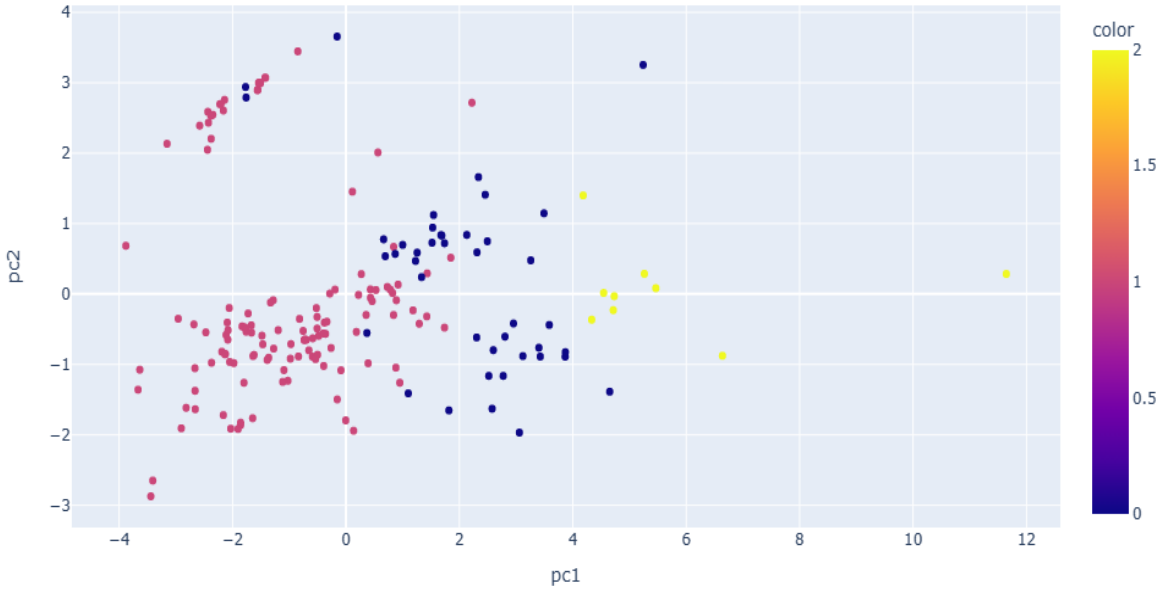
The elbow method is a graphical representation of finding the optimal 'K' in a K-means clustering. It works by finding WCSS (Within-Cluster Sum of Square) i.e. the sum of the square distance between points in a cluster and the cluster centroid.

The optimal clusters for our case study is 3, as shown below:



From the above graph we conclude that the data can be clustered into three clusters.

The clusters can be visualized as shown in below,

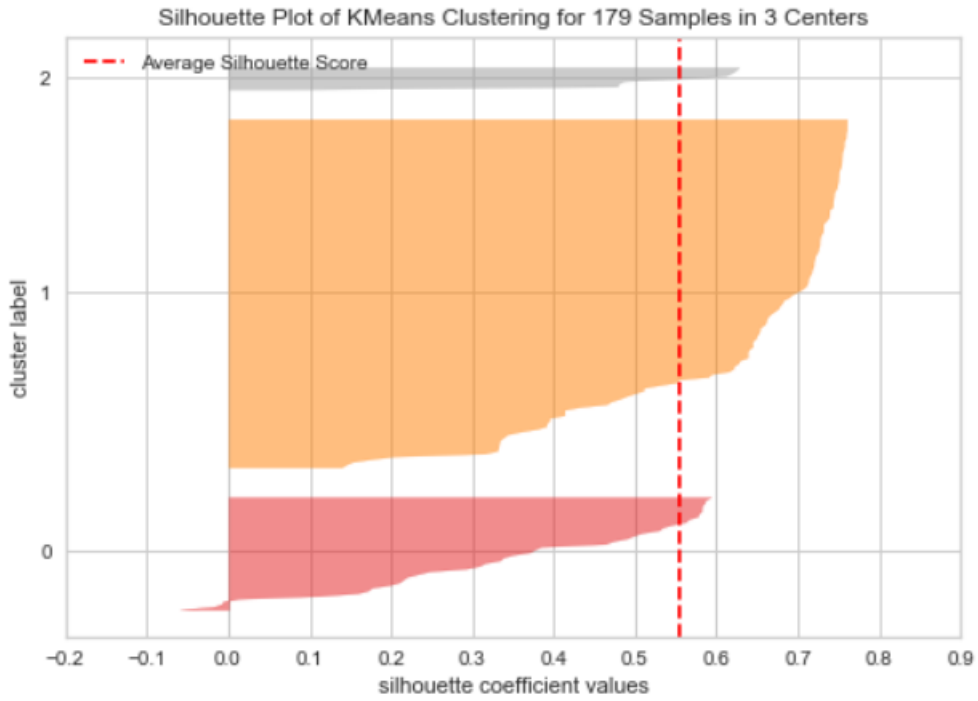


After completion of clustering of data into 3 clusters, that can be visualized as shown.

Obtaining the Stability of the Clusters:

Grouping items into clusters is a complex problem in unsupervised learning with inherent uncertainty. Stability is a measurement that characterizes the strength and reproducibility of a cluster and an items membership to a cluster.

The stability of the clusters is obtained from the below figure,



From the above figure the cluster 1 and 2 are perfectly stabled but not the cluster 0.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Ev2Wheeler.csv')
df
```

```
                  Model  Price  Range        Charging_time  Top_speed
\
0                Ola S1   1.00  128.0                  NaN       95.0

1    TVS iQube Electric   1.60  100.0  2 Hour 50 Min - 80%       78.0

2            Ather 450X   0.98  146.0   5 hours 40 minutes       90.0

3          Bajaj Chetak   1.22   90.0              4 Hours       63.0

4            Ola S1 Pro   1.24  181.0                  NaN      116.0

..                  ...    ...    ...                  ...        ...

69         Cyborg Bob-e   0.95  110.0                  NaN       85.0

70              Hop Oxo   1.64  150.0              5 Hours       95.0

71   Komaki XGT Classic   1.11  105.0                  NaN        NaN

72   PURE EV eTryst 350   1.49  140.0                  NaN       85.0

73  Atumobile AtumVader   1.00  100.0                  NaN       65.0


        Motor_type Battery_type Battery_capacity      Wheels_type
Tyre_type
0    Mid Drive IPM          NaN            3 kWh  Aluminium Alloy
Tubeless
1             BLDC  Lithium Ion         3.04 kwh            Alloy
Tubeless
2             PMSM  Lithium-ion          3.7 kWh            Alloy
Tubeless
3             BLDC  Lithium Ion             3.04            Alloy
Tubeless
4    Mid Drive IPM          NaN            4 kWh  Aluminium Alloy
Tubeless
```

```
..             ...          ...               ...              ...
...
69            BLDC  BMS Battery         2.88 kWH            Alloy
Tubeless
70  BLDC Hub motor  Lithium-ion         3.75 Kwh            Alloy
Tubeless
71            BLDC  Lithium Ion              NaN            Alloy
NaN
72            BLDC  Lithium Ion          3.5 kwh            Alloy
Tubeless
73            BLDC  Lithium Ion          2.3 kwh            Spoke
Tube

[74 rows x 10 columns]
```

```
new_df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/Ev2Wheeler.csv')
```

```
df.head()
```

```
                Model  Price  Range        Charging_time  Top_speed  \
0             Ola S1   1.00  128.0                  NaN       95.0
1  TVS iQube Electric   1.60  100.0  2 Hour 50 Min - 80%       78.0
2          Ather 450X   0.98  146.0   5 hours 40 minutes       90.0
3        Bajaj Chetak   1.22   90.0              4 Hours       63.0
4          Ola S1 Pro   1.24  181.0                  NaN      116.0

      Motor_type Battery_type Battery_capacity      Wheels_type
Tyre_type
0  Mid Drive IPM          NaN            3 kWh  Aluminium Alloy
Tubeless
1           BLDC  Lithium Ion         3.04 kwh            Alloy
Tubeless
2           PMSM  Lithium-ion          3.7 kWh            Alloy
Tubeless
3           BLDC  Lithium Ion             3.04            Alloy
Tubeless
4  Mid Drive IPM          NaN            4 kWh  Aluminium Alloy
Tubeless
```

```
df.describe()
```

```
           Price       Range   Top_speed
count  74.000000   70.000000   59.000000
mean    1.032297  111.871429   66.440678
std     0.574791   49.950097   32.292452
min     0.250000   50.000000   18.000000
25%     0.642500   76.250000   41.000000
50%     0.965000  100.000000   70.000000
75%     1.247500  127.250000   90.000000
max     3.800000  307.000000  152.000000
```

```
#To check if there is any row containing the null value
df.isnull().mean()
```

```
Model               0.000000
Price               0.000000
Range               0.054054
Charging_time       0.824324
Top_speed           0.202703
Motor_type          0.000000
Battery_type        0.108108
Battery_capacity    0.054054
Wheels_type         0.054054
Tyre_type           0.148649
dtype: float64
```

```
#To fill null values present in the cell using forward fill and
backward fill
df = df.fillna(method = 'ffill')
df = df.fillna(method = 'bfill')
```

```
#To check if there is any row containing the null value
df.isnull().mean()
```

```
Model               0.0
Price               0.0
Range               0.0
Charging_time       0.0
Top_speed           0.0
Motor_type          0.0
Battery_type        0.0
Battery_capacity    0.0
Wheels_type         0.0
Tyre_type           0.0
dtype: float64
```

```
df.head()
```

```
                Model  Price  Range        Charging_time  Top_speed  \
0             Ola S1   1.00  128.0  2 Hour 50 Min - 80%       95.0
1  TVS iQube Electric   1.60  100.0  2 Hour 50 Min - 80%       78.0
2          Ather 450X   0.98  146.0   5 hours 40 minutes       90.0
3        Bajaj Chetak   1.22   90.0              4 Hours       63.0
4          Ola S1 Pro   1.24  181.0              4 Hours      116.0

      Motor_type Battery_type Battery_capacity      Wheels_type
Tyre_type
0  Mid Drive IPM  Lithium Ion            3 kWh  Aluminium Alloy
Tubeless
1           BLDC  Lithium Ion         3.04 kwh            Alloy
Tubeless
2           PMSM  Lithium-ion          3.7 kWh            Alloy
```

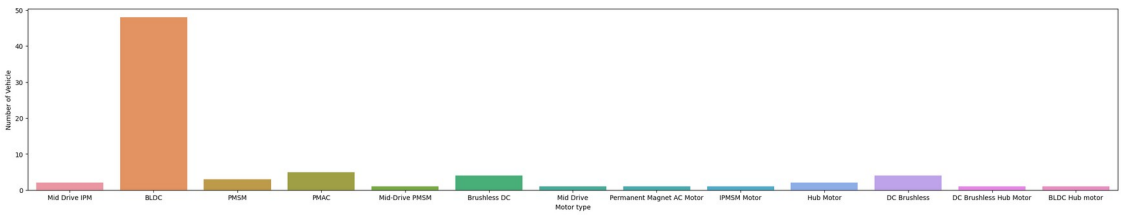
```
Tubeless
3           BLDC  Lithium Ion              3.04            Alloy
Tubeless
4  Mid Drive IPM  Lithium Ion            4 kWh  Aluminium Alloy
Tubeless
```

```
new_df['Motor_type'].value_counts()
```

```
BLDC                          48
PMAC                           5
Brushless DC                   4
DC Brushless                   4
PMSM                           3
Mid Drive IPM                  2
Hub Motor                      2
Mid-Drive PMSM                 1
Mid Drive                      1
Permanent Magnet AC Motor      1
IPMSM Motor                    1
DC Brushless Hub Motor         1
BLDC Hub motor                 1
Name: Motor_type, dtype: int64
```

```
import seaborn as s
import matplotlib.pyplot as plt
plt.figure(figsize=(30,5))
s.countplot(x = new_df['Motor_type'])
plt.xlabel('Motor type')
plt.ylabel('Number of Vehicle')
```

```
Text(0, 0.5, 'Number of Vehicle')
```




```
new_df['Battery_type'].value_counts()
```

```
Lithium Ion                36
Lithium-Ion                 8
VRLA                        6
Lithium-ion                 4
Lithium Ion (Fixed Type)    3
Lead Acid                   2
BMS Battery                 2
LFP                         1
Li-ion                      1
Lithium Ferro Phosphate     1
Lithium ion                 1
```

```
Lithium-Ion (Fixed Type)     1
Name: Battery_type, dtype: int64
```

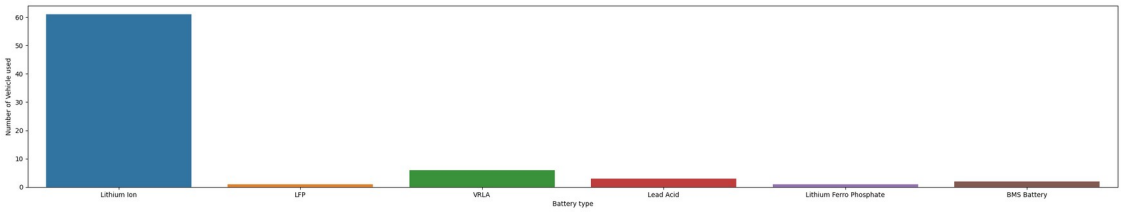
```
#To merge all the battery of same kind

df['Battery_type'] = df['Battery_type'].str.replace('Lithium-Ion',
'Lithium Ion')
df['Battery_type'] = df['Battery_type'].str.replace('Lithium-ion',
'Lithium Ion')
df['Battery_type'] = df['Battery_type'].str.replace('Lithium ion',
'Lithium Ion')
df['Battery_type'] = df['Battery_type'].str.replace('Lithium Ion',
'Lithium Ion')
df['Battery_type'] = df['Battery_type'].str.replace('Li-ion', 'Lithium
Ion')
df.loc[df['Battery_type'].str.contains('Lithium Ion', case=False),
'Battery_type'] = 'Lithium Ion'
```

```
df['Battery_type'].value_counts()
```

```
Lithium Ion                61
VRLA                        6
Lead Acid                   3
BMS Battery                 2
LFP                         1
Lithium Ferro Phosphate     1
Name: Battery_type, dtype: int64
```

```
import seaborn as s
import matplotlib.pyplot as plt
plt.figure(figsize=(30,5))
s.countplot(x = df['Battery_type'])
plt.xlabel('Battery type')
plt.ylabel('Number of Vehicle used')
```

```
Text(0, 0.5, 'Number of Vehicle used')
```




```
df['Wheels_type'] = df['Wheels_type'].str.replace('ALLOY', 'Alloy')
df['Wheels_type'] = df['Wheels_type'].str.replace('Alloy Wheels',
'Alloy')
#df.loc[df['Battery_type'].str.contains('Lithium Ion', case=False),
'Battery_type'] = 'Lithium Ion'
```

```
new_df['Wheels_type'].value_counts()
```

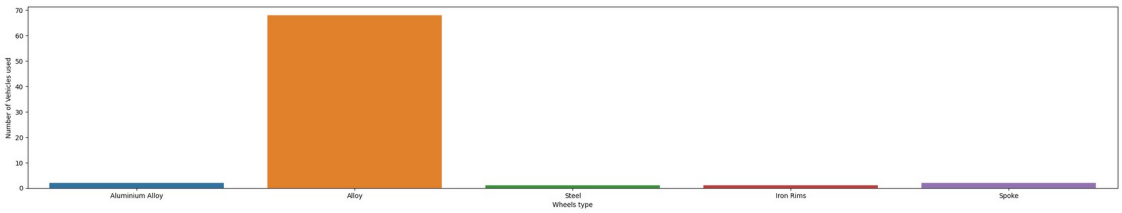
```
Alloy              61
Aluminium Alloy     2
ALLOY               2
Spoke               2
Alloy Wheels        1
Steel               1
Iron Rims           1
Name: Wheels_type, dtype: int64
```

```
df['Wheels_type'].value_counts()
```

```
Alloy              68
Aluminium Alloy     2
Spoke               2
Steel               1
Iron Rims           1
Name: Wheels_type, dtype: int64
```

```
plt.figure(figsize=(30,5))
s.countplot(x = df['Wheels_type'])
plt.xlabel('Wheels type')
plt.ylabel('Number of Vehicles used')
```

```
Text(0, 0.5, 'Number of Vehicles used')
```

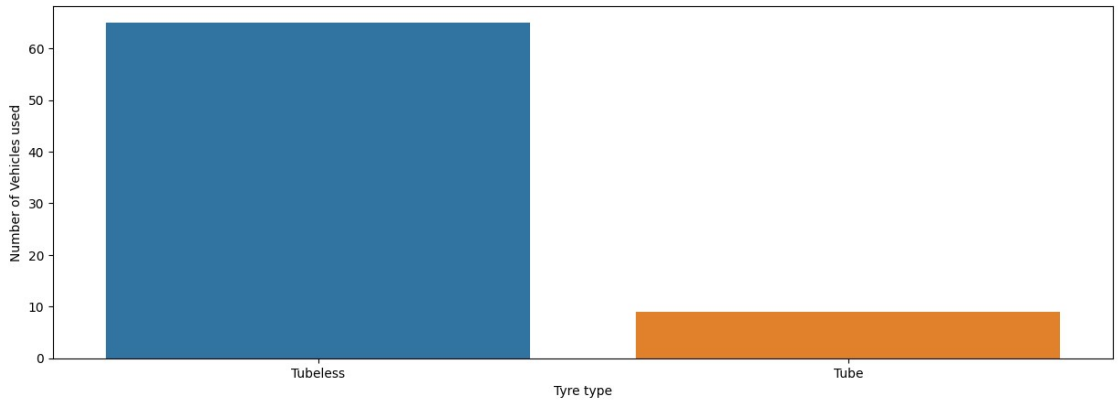



```
df['Tyre_type'].value_counts()
```

```
Tubeless    65
Tube         9
Name: Tyre_type, dtype: int64
```

```
plt.figure(figsize=(15,5))
s.countplot(x = df['Tyre_type'])
plt.xlabel('Tyre type')
plt.ylabel('Number of Vehicles used')
```

```
Text(0, 0.5, 'Number of Vehicles used')
```




```
df_copy = df.copy()
```

```
df_copy
```

	Model	Price	Range	Charging_time	Top_speed
0	Ola S1	1.00	128.0	2 Hour 50 Min - 80%	95.0
1	TVS iQube Electric	1.60	100.0	2 Hour 50 Min - 80%	78.0
2	Ather 450X	0.98	146.0	5 hours 40 minutes	90.0
3	Bajaj Chetak	1.22	90.0	4 Hours	63.0
4	Ola S1 Pro	1.24	181.0	4 Hours	116.0
..	...	...	...	...	...
69	Cyborg Bob-e	0.95	110.0	40 Minutes	85.0
70	Hop Oxo	1.64	150.0	5 Hours	95.0
71	Komaki XGT Classic	1.11	105.0	5 Hours	95.0
72	PURE EV eTryst 350	1.49	140.0	5 Hours	85.0
73	Atumobile AtumVader	1.00	100.0	5 Hours	65.0

	Motor_type	Battery_type	Battery_capacity	Wheels_type	Tyre_type
0	Mid Drive IPM	Lithium Ion	3 kWh	Aluminium Alloy	Tubeless
1	BLDC	Lithium Ion	3.04 kwh	Alloy	Tubeless
2	PMSM	Lithium Ion	3.7 kWh	Alloy	Tubeless

```
3             BLDC  Lithium Ion             3.04            Alloy
Tubeless
4    Mid Drive IPM  Lithium Ion            4 kWh  Aluminium Alloy
Tubeless
..             ...          ...              ...              ...
...
69            BLDC  BMS Battery         2.88 kWH            Alloy
Tubeless
70  BLDC Hub motor  Lithium Ion         3.75 Kwh            Alloy
Tubeless
71            BLDC  Lithium Ion         3.75 Kwh            Alloy
Tubeless
72            BLDC  Lithium Ion          3.5 kwh            Alloy
Tubeless
73            BLDC  Lithium Ion          2.3 kwh            Spoke
Tube

[74 rows x 10 columns]
```

```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
df['Motor_type'] = label.fit_transform(df['Motor_type'])
df['Battery_type'] = label.fit_transform(df['Battery_type'])
df['Wheels_type'] = label.fit_transform(df['Wheels_type'])
df['Tyre_type'] = label.fit_transform(df['Tyre_type'])
```

```
df
```

```
                  Model  Price  Range        Charging_time  Top_speed
\
0                Ola S1   1.00  128.0  2 Hour 50 Min - 80%       95.0

1    TVS iQube Electric   1.60  100.0  2 Hour 50 Min - 80%       78.0

2            Ather 450X   0.98  146.0   5 hours 40 minutes       90.0

3          Bajaj Chetak   1.22   90.0              4 Hours       63.0

4            Ola S1 Pro   1.24  181.0              4 Hours      116.0

..                  ...    ...    ...                  ...        ...

69         Cyborg Bob-e   0.95  110.0           40 Minutes       85.0

70              Hop Oxo   1.64  150.0              5 Hours       95.0

71   Komaki XGT Classic   1.11  105.0              5 Hours       95.0

72   PURE EV eTryst 350   1.49  140.0              5 Hours       85.0
```

```
73  Atumobile AtumVader   1.00  100.0              5 Hours       65.0
```

	Motor_type	Battery_type	Battery_capacity	Wheels_type	Tyre_type
0	8	4	3 kWh	1	1
1	0	4	3.04 kwh	0	1
2	11	4	3.7 kWh	0	1
3	0	4	3.04	0	1
4	8	4	4 kWh	1	1
..	...	...	...	...	...
69	0	0	2.88 kWH	0	1
70	1	4	3.75 Kwh	0	1
71	0	4	3.75 Kwh	0	1
72	0	4	3.5 kwh	0	1
73	0	4	2.3 kwh	3	0

```
[74 rows x 10 columns]
```

```python
import re

def convert_to_minutes(time_str):
    # Check if the time value is empty or NaN
    if pd.isnull(time_str) or time_str.strip() == '':
        return np.nan

    # Convert hours, minutes, and seconds to minutes
    time_str = time_str.lower()
    minutes = 0

    # Check if the time value contains 'hour(s)' or 'hr(s)'
    if re.search(r'\b(hour|hr)s?\b', time_str):
        hours = re.findall(r'\d+', time_str)
        minutes += int(hours[0]) * 60

    # Check if the time value contains 'minute(s)' or 'min(s)'
    if re.search(r'\bminute|min\b', time_str):
        mins = re.findall(r'\d+', time_str)
```

```
        minutes += int(mins[0])

    return minutes

df['Charging_time'] = df['Charging_time'].apply(convert_to_minutes)

df
```

```
                  Model  Price  Range  Charging_time  Top_speed
Motor_type  \
0                Ola S1   1.00  128.0            122       95.0
8
1    TVS iQube Electric   1.60  100.0            122       78.0
0
2            Ather 450X   0.98  146.0            305       90.0
11
3          Bajaj Chetak   1.22   90.0            240       63.0
0
4            Ola S1 Pro   1.24  181.0            240      116.0
8
..                  ...    ...    ...            ...        ...
...
69         Cyborg Bob-e   0.95  110.0             40       85.0
0
70              Hop Oxo   1.64  150.0            300       95.0
1
71   Komaki XGT Classic   1.11  105.0            300       95.0
0
72   PURE EV eTryst 350   1.49  140.0            300       85.0
0
73  Atumobile AtumVader   1.00  100.0            300       65.0
0

    Battery_type Battery_capacity  Wheels_type  Tyre_type
0              4            3 kWh            1          1
1              4         3.04 kwh            0          1
2              4          3.7 kWh            0          1
3              4             3.04            0          1
4              4            4 kWh            1          1
..           ...              ...          ...        ...
69             0         2.88 kWH            0          1
70             4         3.75 Kwh            0          1
71             4         3.75 Kwh            0          1
72             4          3.5 kwh            0          1
73             4          2.3 kwh            3          0

[74 rows x 10 columns]
```

```
def convert_to_kwh(capacity_str):
    # Check if the capacity value is empty or NaN
    if pd.isnull(capacity_str) or capacity_str.strip() == '':
```

```
        return np.nan

    # Extract the numerical value from the capacity string
    capacity_str = capacity_str.lower()
    capacity = re.findall(r'(\d+(?:\.\d+)?)', capacity_str)

    if len(capacity) > 0:
        capacity = float(capacity[0])
    else:
        return np.nan

    # Check if the capacity value contains 'kwh'
    if re.search(r'kwh\b', capacity_str):
        return capacity
    else:
        return capacity

df['Battery_capacity'] = df['Battery_capacity'].apply(convert_to_kwh)

df
```

```
                  Model  Price  Range  Charging_time  Top_speed
Motor_type  \
0                 Ola S1   1.00  128.0            122       95.0
8
1    TVS iQube Electric   1.60  100.0            122       78.0
0
2            Ather 450X   0.98  146.0            305       90.0
11
3          Bajaj Chetak   1.22   90.0            240       63.0
0
4            Ola S1 Pro   1.24  181.0            240      116.0
8
..                  ...    ...    ...            ...        ...
...
69         Cyborg Bob-e   0.95  110.0             40       85.0
0
70              Hop Oxo   1.64  150.0            300       95.0
1
71   Komaki XGT Classic   1.11  105.0            300       95.0
0
72   PURE EV eTryst 350   1.49  140.0            300       85.0
0
73  Atumobile AtumVader   1.00  100.0            300       65.0
0

    Battery_type  Battery_capacity  Wheels_type  Tyre_type
0              4              3.00            1          1
1              4              3.04            0          1
2              4              3.70            0          1
3              4              3.04            0          1
```

```
4              4              4.00            1          1
..           ...               ...         ...        ...
69             0              2.88            0          1
70             4              3.75            0          1
71             4              3.75            0          1
72             4              3.50            0          1
73             4              2.30            3          0

[74 rows x 10 columns]
```

```
selected_column = ["Top_speed", "Motor_type", "Battery_type",
"Battery_capacity", "Wheels_type", "Tyre_type", "Charging_time",
"Range", "Price"]
x = df.loc[:,selected_column]
```

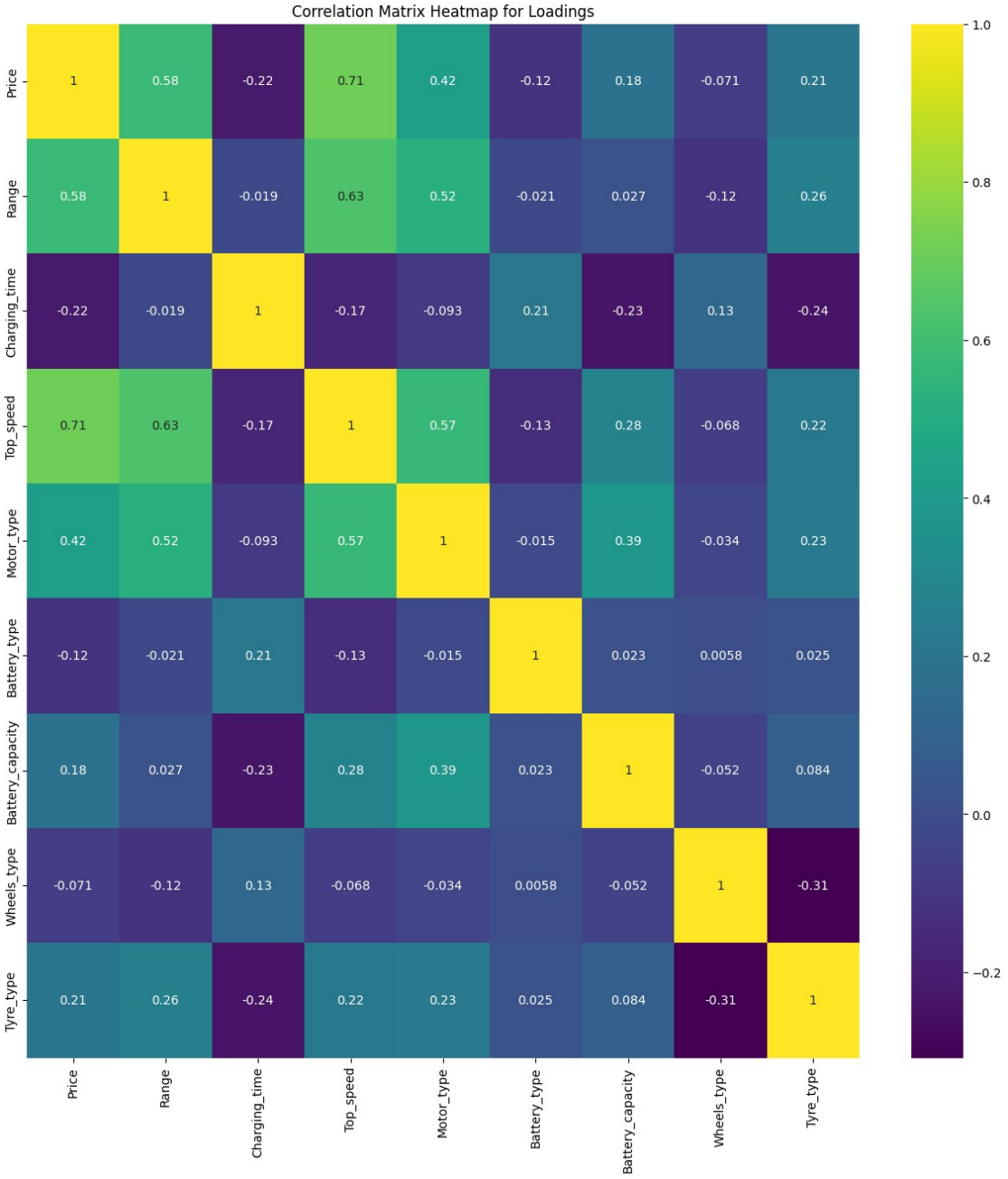
```
x
```

```
    Top_speed  Motor_type  Battery_type  Battery_capacity  Wheels_type
\
0        95.0           8             4              3.00            1

1        78.0           0             4              3.04            0

2        90.0          11             4              3.70            0

3        63.0           0             4              3.04            0

4       116.0           8             4              4.00            1

..        ...         ...           ...               ...          ...

69       85.0           0             0              2.88            0

70       95.0           1             4              3.75            0

71       95.0           0             4              3.75            0

72       85.0           0             4              3.50            0

73       65.0           0             4              2.30            3


    Tyre_type  Charging_time  Range  Price
0           1            122  128.0   1.00
1           1            122  100.0   1.60
2           1            305  146.0   0.98
3           1            240   90.0   1.22
4           1            240  181.0   1.24
..        ...            ...    ...    ...
69          1             40  110.0   0.95
```

```
70          1            300  150.0   1.64
71          1            300  105.0   1.11
72          1            300  140.0   1.49
73          0            300  100.0   1.00

[74 rows x 9 columns]
```

```
# Plot a correlation matrix heatmap for the loadings
plt.rcParams['figure.figsize'] = (15,15)
s.heatmap(df.corr(), annot=True, cmap='viridis')
plt.title('Correlation Matrix Heatmap for Loadings')
plt.show()
```

```
<ipython-input-195-0a40ea0104c7>:3: 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.
  s.heatmap(df.corr(), annot=True, cmap='viridis')
```




```
#Making Principal Component Analysis
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
stx = StandardScaler()
scaled_data = stx.fit_transform(x)
pca = PCA()
scaled_data = pca.fit_transform(scaled_data)
scaled_data
```

```
array([[ 1.24235375e+00,  7.36583797e-01,  4.12868215e-01,
         4.01350173e-01,  8.80517279e-01, -8.42573410e-01,
```

```
  7.50970788e-01, -5.36455402e-01,  6.09000206e-01],
[ 4.88724793e-01, -4.58940303e-01, -9.59539915e-02,
 -3.58395095e-01,  5.14167964e-01,  7.69234014e-01,
 -7.54765651e-01, -3.29149519e-01,  3.15301408e-02],
[ 1.52409702e+00,  1.11096835e+00, -9.14841796e-01,
  4.23429674e-01, -8.74189086e-01, -1.22782852e+00,
  7.81223306e-01, -8.23774199e-01,  2.53074956e-01],
[-3.21279238e-01, -7.85980449e-02, -5.25528432e-01,
 -1.75309125e-01, -6.00516711e-02,  1.23799198e-01,
 -8.89498560e-01, -2.58973325e-01, -5.03886343e-04],
[ 2.02267826e+00,  1.75493370e+00, -1.32957744e-01,
  5.86604908e-02,  4.25942169e-01, -9.14386441e-01,
  2.91639014e-01, -1.99883559e-02,  6.64128713e-01],
[ 3.15797483e+00,  1.60925252e+00, -1.32899216e+00,
 -3.61888779e-01, -5.19273820e-01, -7.97226038e-01,
  1.57311512e+00,  1.41107124e+00,  1.05330957e-01],
[ 2.17355463e+00,  8.91461159e-03, -1.74347508e-01,
  3.11188687e-01,  2.12300725e-01, -4.01613245e-01,
  1.63331265e+00, -5.86243758e-01, -2.14595490e-01],
[-3.41216149e-01, -9.13667759e-01, -2.90036622e-01,
 -2.17599181e-01,  5.93708953e-01,  1.63737562e-01,
  5.89932907e-01,  8.51138341e-01,  2.75776891e-02],
[-2.80892711e-01, -9.52164868e-01, -1.60268794e-01,
 -1.65468522e-01,  6.12189741e-01,  2.97134410e-01,
  3.13229061e-01,  4.88867704e-01,  4.09496848e-02],
[-5.27641314e-01, -1.09677307e+00,  5.44946756e-03,
  1.36798637e-02,  5.45319198e-01,  2.17746438e-01,
  1.64642633e-01,  3.72716578e-02,  7.07421888e-01],
[-3.17087651e-01, -1.03797666e+00,  1.93260976e-02,
 -9.82503964e-02,  6.38562938e-01,  4.35650273e-01,
 -3.53690266e-02, -9.70260673e-02,  1.86452442e-01],
[-6.91097028e-01, -1.12529277e+00, -1.08876723e-01,
 -4.50498791e-02,  6.03470407e-01,  1.94020343e-01,
  3.30641961e-01,  2.62430419e-01, -4.82222994e-02],
[ 1.74162451e+00, -1.01856777e+00,  2.00000552e+00,
 -1.25760494e+00, -1.14006984e+00, -1.86713376e+00,
  1.18580782e+00, -1.09769994e+00, -1.68252679e-01],
[-6.39875069e-01, -1.10582294e+00, -1.01251906e-01,
 -4.61965826e-02,  5.91519845e-01,  1.93659244e-01,
  3.21277199e-01,  2.63158997e-01,  1.01603637e-01],
[ 1.68993659e+00, -2.63743526e-01,  4.14933594e-02,
  3.57682743e-01,  2.65740105e-01, -1.82385077e-01,
  1.04220823e+00, -9.81427631e-01,  2.70495402e-01],
[-1.36798562e+00, -1.43149968e+00,  1.60551066e-02,
  1.93911974e-01,  5.80869398e-01,  4.12919538e-02,
  2.94839533e-01, -1.89032941e-01, -1.64023840e-01],
[ 4.96358348e-01, -6.90799336e-01, -1.36390770e-01,
 -4.21982523e-01,  7.41969049e-01,  7.46623586e-01,
 -8.98775220e-02,  3.97102559e-01, -4.04588524e-01],
[-1.42116453e+00,  2.23284041e-01,  1.14779580e+00,
```

```
  8.35748876e-02, -5.55532289e-01,  1.95448942e+00,
  1.31810303e+00,  2.84580110e-01,  4.05894147e-03],
[-1.57557715e+00,  4.50871050e-01, -1.68589796e+00,
  8.55129304e-01, -2.98430765e-01, -5.26345847e-01,
 -3.53044581e-01, -4.22403800e-01, -3.62999481e-01],
[-1.91150331e+00, -4.66991955e-01,  2.91408760e-01,
 -9.97974759e-01, -1.59551791e+00, -1.85405842e+00,
 -7.88261869e-01, -5.55671816e-03, -1.89126564e-01],
[-1.26108357e+00, -2.84023476e-01,  3.83910100e-01,
 -1.26330764e+00, -1.39876184e+00, -1.30077327e+00,
 -1.37026919e+00, -4.58011444e-01, -9.56786438e-01],
[-8.60453944e-01,  3.65201719e-01, -8.15417371e-01,
 -2.21203786e-02, -6.23898978e-01, -3.37470663e-01,
 -1.38042316e+00, -6.36942064e-01,  4.23178079e-01],
[-2.79875180e+00,  4.51761837e+00,  2.88565505e+00,
  6.59020185e-01,  1.67580184e+00, -5.92763870e-01,
 -6.82384943e-01,  5.81164720e-03, -1.00603076e-01],
[-3.11563048e+00,  1.50204680e+00, -4.07721061e-01,
  9.91765955e-01, -1.43491031e+00,  1.17329792e+00,
  5.21734495e-01, -3.25907258e-01,  1.33343333e-01],
[-1.26251743e+00,  4.70055936e-01, -1.42151962e+00,
 -2.28723102e-01, -6.58524636e-01, -8.93715080e-01,
 -1.39649127e-01,  1.04932799e+00, -4.25499759e-01],
[-3.09159529e+00,  1.50886093e+00, -4.04366276e-01,
  9.81768109e-01, -1.42755763e+00,  1.19379484e+00,
  5.00254152e-01, -3.42749868e-01,  1.04937898e-01],
[-2.88174032e+00,  1.59040722e+00, -4.25621701e-01,
  9.07924875e-01, -1.40018084e+00,  1.28716691e+00,
  4.50936774e-01, -2.69652642e-01, -2.71076917e-02],
[-2.05618643e+00,  2.69648399e-01, -1.68125536e+00,
  7.87654829e-01, -2.49835441e-01, -4.49507823e-01,
 -5.54563171e-01, -2.19917834e-01, -1.05503064e-01],
[ 1.37033691e-01, -1.92123221e-01, -4.42114017e-01,
 -3.05703901e-01,  1.80514085e-01,  3.00664873e-01,
 -5.19720256e-01,  1.83190155e-01,  1.93548564e-01],
[-1.73337669e+00,  1.91154240e-01,  2.20473865e+00,
 -1.03642167e+00, -1.94103129e+00,  6.93379802e-01,
  5.65967804e-01, -8.78250582e-02,  2.25272387e-01],
[-2.61690827e+00,  6.79647075e-01,  1.48283508e-01,
  9.17608614e-01, -6.30559614e-01,  1.78175423e+00,
  1.17002843e+00, -1.23795327e-01, -1.60912478e-01],
[-1.65140369e+00, -8.63924159e-01, -4.02101460e-01,
  1.62597754e-01,  9.33884147e-02, -3.41946416e-01,
 -4.52432630e-02, -2.03654902e-01, -1.19401441e-01],
[-9.80097325e-01, -6.11537786e-01, -4.18105944e-01,
  1.68028928e-02,  1.00337103e-01, -1.37923730e-01,
 -1.89458433e-01,  2.44764343e-03,  1.11947154e-01],
[-1.42180828e+00,  5.00707250e-01,  1.64296021e+00,
 -1.62455312e-01,  1.34737101e+00, -1.50228152e+00,
 -7.32205133e-01,  2.41459825e-02,  8.04528134e-02],
```

```
[-6.28556489e-01, -4.70779573e-01, -4.25972991e-01,
  1.17579793e-01,  3.18307717e-02, -2.89707876e-01,
  7.73363122e-02, -2.05312415e-01, -2.58300212e-02],
[-6.04466870e-01, -4.64002963e-01, -4.22547016e-01,
  1.07752649e-01,  3.91180337e-02, -2.69215833e-01,
  5.57804854e-02, -2.22069997e-01, -5.42618684e-02],
[-1.20693269e+00, -7.28113800e-01, -3.45491872e-01,
  2.79540545e-01,  5.02328047e-02, -3.76718326e-01,
  4.36954081e-02, -5.70463506e-01, -4.40763185e-01],
[-1.52784607e+00, -8.16463683e-01, -4.25133732e-01,
  1.52128450e-01,  8.65152407e-02, -3.21719411e-01,
 -2.90919505e-02, -5.73360660e-02, -1.85350606e-01],
[-1.56799543e+00, -8.27758033e-01, -4.30843691e-01,
  1.68507023e-01,  7.43698040e-02, -3.55872816e-01,
  6.83442745e-03, -2.94067620e-02, -1.37964194e-01],
[-1.04640581e+00, -5.78834861e-01, -5.80785688e-01,
 -7.21358030e-02,  1.34784978e-01, -2.01914810e-01,
  7.29956786e-02,  4.17320017e-01, -4.70848106e-01],
[-1.52784607e+00, -8.16463683e-01, -4.25133732e-01,
  1.52128450e-01,  8.65152407e-02, -3.21719411e-01,
 -2.90919505e-02, -5.73360660e-02, -1.85350606e-01],
[-1.25935179e+00, -7.05169972e-01, -4.67921259e-01,
  4.14933050e-02,  1.27256861e-01, -2.04452946e-01,
 -7.08089274e-02,  6.54287490e-02, -3.94387620e-01],
[-1.01743654e+00, -5.44112907e-01, -6.33925403e-01,
 -9.46574445e-02,  1.20792614e-01, -2.48714322e-01,
  1.80814450e-01,  5.97177550e-01, -4.35091323e-01],
[ 9.15808106e-02, -2.37524993e-01, -2.71915578e-01,
 -1.79664610e-01,  7.18050192e-02,  1.99395542e-01,
 -6.58861880e-01, -2.59887920e-01,  1.58480871e+00],
[-2.32860683e-01, -3.56950875e-01, -2.58893845e-01,
 -3.08204470e-02, -6.04132604e-03, -6.68645732e-03,
 -5.03089992e-01, -2.30517756e-01,  1.89891902e+00],
[-1.17456729e+00, -6.23033006e-01, -5.90514998e-01,
  2.23956137e-03,  9.04501768e-02, -3.09059648e-01,
  1.76768512e-01,  5.15792542e-01, -3.56533294e-01],
[-1.24683615e+00, -6.43362837e-01, -6.00792925e-01,
  3.17209921e-02,  6.85883907e-02, -3.70535776e-01,
  2.41435992e-01,  5.66065289e-01, -2.71237752e-01],
[-1.19062703e+00, -6.27550746e-01, -5.92798982e-01,
  8.79099042e-03,  8.55920021e-02, -3.22721009e-01,
  1.91139063e-01,  5.26964263e-01, -3.37578729e-01],
[ 1.33677241e+00,  6.59161217e-01, -7.17231057e-01,
  2.53539549e-02, -3.56963158e-01, -4.82466074e-01,
  3.19371920e-01, -3.74741154e-01,  6.16161764e-02],
[ 6.30562008e+00,  2.84508136e+00, -1.04475177e+00,
 -1.01143274e+00, -3.27873848e-01,  8.68043514e-01,
 -6.62696608e-01, -3.15299016e-01, -1.48605433e+00],
[ 2.13081531e+00,  1.34248675e+00, -9.89095531e-01,
  9.67564144e-02, -7.61371005e-01, -8.84614875e-01,
```

```
  5.79280339e-01, -3.92755982e-01,  3.24636273e-01],
[ 1.61537598e+00,  1.18756188e+00, -1.12027579e+00,
 -1.97545115e-01, -6.36638365e-01, -6.31616402e-01,
  2.79724640e-01,  5.29496901e-01,  4.82254600e-01],
[ 1.18410610e+00,  9.67963672e-01, -1.10828106e+00,
 -8.28523075e-01, -1.95314540e-01,  3.71574733e-01,
 -9.63382869e-01,  9.63224195e-01, -4.26811486e-01],
[ 6.25245495e-01, -3.88986827e-01, -1.82179907e-01,
 -3.36747237e-01,  4.54836586e-01,  6.66812411e-01,
 -5.30658341e-01,  1.53791192e-01,  1.52792018e-01],
[ 1.77028217e+00,  6.66843561e-02, -9.07472245e-02,
  3.81754656e-01,  2.66796195e-02, -3.47025073e-01,
  8.39714191e-01, -1.36448487e+00,  2.68683922e-01],
[ 1.80822004e+00,  1.76763931e-01, -4.64573859e-01,
 -9.07882090e-01,  5.88348282e-01,  1.02341530e+00,
 -4.67368261e-01,  8.97011947e-01, -1.01208866e-01],
[ 2.00054234e+00,  3.03716720e-01, -5.47040154e-01,
 -3.98435130e-01,  2.80285036e-01,  1.99813200e-01,
  6.52914841e-01,  6.77952447e-01, -1.91839516e-01],
[ 1.70302922e+00,  2.28213692e-01, -5.99186399e-01,
 -3.06968340e-01,  2.02977362e-01, -4.53353295e-02,
  9.23284873e-01,  8.65864060e-01,  1.53436495e-01],
[ 1.57334921e+00, -6.68408737e-01,  1.79610300e-01,
 -5.20743185e-01,  9.19114168e-01,  1.11327512e+00,
 -1.22076824e-01,  4.54051815e-01,  1.17735245e+00],
[-4.01030909e-01, -1.44347268e+00,  2.40297874e-01,
  2.19431191e-01,  8.39973944e-01,  2.42770719e-01,
  7.08857906e-01, -5.88536011e-01, -7.50183138e-01],
[ 8.98705831e-01, -9.52868347e-01,  2.47737503e-01,
 -3.11703143e-01,  9.01267999e-01,  9.34563780e-01,
 -6.28176789e-02,  1.65597539e-01,  9.18851820e-01],
[ 3.10764546e+00, -1.47313351e+00,  2.00879012e+00,
  4.43252416e+00, -1.00815046e+00, -1.65630813e-01,
 -4.86375832e-01,  5.49848956e-01, -2.07856712e-01],
[ 3.62283270e+00, -1.34368072e+00,  2.25882799e+00,
  5.16443253e+00, -1.47079937e+00, -1.56423728e-01,
 -1.17408668e+00,  8.21864190e-01, -1.02713509e-01],
[-1.04812673e+00,  1.02442142e+00,  1.86119308e+00,
  6.03055979e-01,  3.17368756e+00, -1.47396249e+00,
  5.25412128e-01,  3.16700356e-01, -5.48047270e-01],
[-7.21263513e-01, -1.28095726e+00, -4.25193395e-02,
 -9.18911644e-02,  7.77533453e-01,  3.39509586e-01,
  3.81709301e-01,  2.00086875e-01, -5.86688262e-01],
[ 1.91842046e+00, -4.58477239e-01,  3.04507372e-01,
 -3.70843811e-01,  8.85750716e-01,  1.29538774e+00,
 -6.56991084e-01, -9.83355551e-01, -5.73256382e-01],
[ 1.95998428e+00, -4.33602904e-01,  2.97758732e-01,
 -2.67123216e-01,  8.21596383e-01,  1.13186155e+00,
 -4.46904347e-01, -1.05920551e+00, -5.26279926e-01],
[ 1.58473727e+00, -6.26356559e-01,  4.21978179e-01,
```

```
        -2.23443353e-01,  8.61920110e-01,  1.23139260e+00,
        -7.56736796e-01, -1.36453537e+00, -3.93355773e-01],
       [ 1.02307523e+00, -6.76347920e-02,  3.85800840e+00,
        -3.06319637e+00, -1.96517687e+00,  1.21371312e+00,
         2.19721077e-01,  5.21086515e-01, -3.43180972e-01],
       [ 6.53399570e-01, -1.84349168e+00,  2.68407769e+00,
        -2.68087674e+00, -1.04466913e+00, -1.17844938e+00,
        -3.44866445e-01,  4.62072759e-01,  5.02701592e-01],
       [ 1.03401523e+00,  8.28772073e-01, -8.64009133e-01,
        -5.61342382e-01, -2.74335062e-01,  2.77721478e-01,
        -1.13728111e+00,  1.12216442e-01,  1.22077338e-01],
       [ 1.11296493e-01,  4.36444592e-01, -7.24908015e-01,
        -2.57436282e-01, -3.74334756e-01,  1.31768103e-02,
        -1.16072905e+00, -9.90145113e-02,  8.00321264e-01],
       [ 5.60530605e-01,  6.49003220e-01, -8.66073521e-01,
        -5.29715416e-01, -2.58074977e-01,  2.36339018e-01,
        -1.16588411e+00,  2.08114297e-01,  9.20024199e-02],
       [-1.76489536e+00,  3.80720963e+00,  2.34437460e+00,
         2.98637213e-01,  1.08255547e+00,  2.08284686e-01,
        -3.22121818e-01,  3.42173182e-01,  1.66041179e-01]])

pc_df = pd.DataFrame(data = scaled_data,columns=selected_column)
pc_df
```

	Top_speed	Motor_type	Battery_type	Battery_capacity	Wheels_type
0	1.242354	0.736584	0.412868	0.401350	0.880517
1	0.488725	-0.458940	-0.095954	-0.358395	0.514168
2	1.524097	1.110968	-0.914842	0.423430	-0.874189
3	-0.321279	-0.078598	-0.525528	-0.175309	-0.060052
4	2.022678	1.754934	-0.132958	0.058660	0.425942
..	...	...	...	...	...
69	0.653400	-1.843492	2.684078	-2.680877	-1.044669
70	1.034015	0.828772	-0.864009	-0.561342	-0.274335
71	0.111296	0.436445	-0.724908	-0.257436	-0.374335
72	0.560531	0.649003	-0.866074	-0.529715	-0.258075
73	-1.764895	3.807210	2.344375	0.298637	1.082555

```
    Tyre_type  Charging_time     Range     Price
0   -0.842573       0.750971 -0.536455  0.609000
1    0.769234      -0.754766 -0.329150  0.031530
2   -1.227829       0.781223 -0.823774  0.253075
3    0.123799      -0.889499 -0.258973 -0.000504
4   -0.914386       0.291639 -0.019988  0.664129
..        ...            ...       ...       ...
69  -1.178449      -0.344866  0.462073  0.502702
70   0.277721      -1.137281  0.112216  0.122077
71   0.013177      -1.160729 -0.099015  0.800321
72   0.236339      -1.165884  0.208114  0.092002
73   0.208285      -0.322122  0.342173  0.166041

[74 rows x 9 columns]
```

```
std_dev = np.std(scaled_data,axis = 0)
print('Standard Deviation:',std_dev)
```

```
Standard Deviation: [1.74329277 1.14348046 1.07807937 1.03993028
0.85622952 0.80027144
 0.68403288 0.5591151  0.50556759]
```

```
# Get the proportion of variance explained by each principal component
explained_variance_ratio = pca.explained_variance_ratio_

# Print the proportion of variance for each principal component
print("Proportion of Variance : \n", explained_variance_ratio)
```

```
Proportion of Variance :
 [0.33767441 0.14528306 0.12913946 0.12016167 0.08145878 0.07115938
 0.051989   0.03473441 0.02839984]
```

```
# Calculate the cumulative proportion of variance explained by the
principal components
cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# Print the cumulative proportion of variance for each principal
component
print("Cumulative Proportion : \n", cumulative_variance_ratio)
```

```
Cumulative Proportion :
 [0.33767441 0.48295747 0.61209693 0.7322586  0.81371737 0.88487675
 0.93686575 0.97160016 1.        ]
```

```
# Get the principal components and number of features
principal_components = pca.components_
num_features = pca.n_features_

# Create a list of names for the principal components
pc_names = ["PC" + str(i) for i in range(1, num_features + 1)]
```

```
# Create a DataFrame to store the loadings for each principal
component
loadings_df = pd.DataFrame(data=principal_components.T,
columns=pc_names)

# Display the DataFrame
loadings_df
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/
deprecation.py:101: FutureWarning: Attribute `n_features_` was
deprecated in version 1.2 and will be removed in 1.4. Use
`n_features_in_` instead.
  warnings.warn(msg, category=FutureWarning)
```

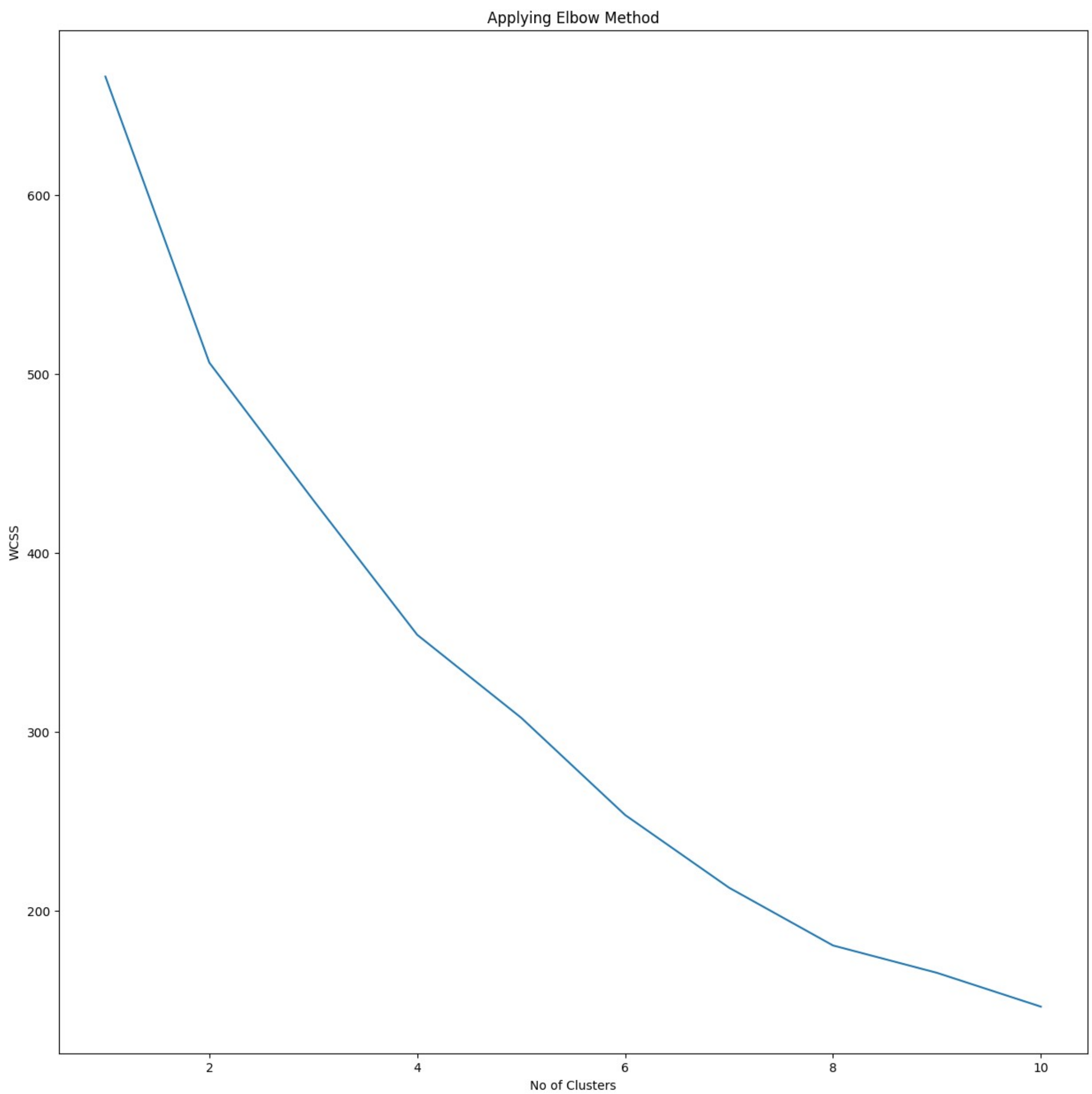
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
0	0.496808	0.173140	0.072899	-0.072391	-0.030762	0.132798	-0.203974	-0.105739	0.800801
1	0.426456	0.169350	0.013222	0.278125	-0.158442	-0.382314	0.544790	-0.472499	-0.143556
2	-0.080498	0.202104	-0.577957	0.549482	0.385860	0.389401	0.084643	-0.061754	0.072182
3	0.229002	-0.157856	0.299482	0.718106	-0.275189	-0.020518	-0.317545	0.357693	-0.111109
4	-0.120248	0.524540	0.453736	0.096318	0.609006	-0.324609	-0.103315	0.090764	-0.008513
5	0.250336	-0.441184	-0.403794	-0.028317	0.372016	-0.575410	-0.330674	-0.000492	-0.000299
6	-0.184622	0.567384	-0.379403	-0.018833	-0.469659	-0.309166	-0.422130	-0.047088	-0.055812
7	0.437073	0.259303	-0.231655	-0.233224	0.015452	0.005210	0.295160	0.722259	-0.156936
8	0.458421	0.128958	0.065196	-0.187009	0.138675	0.389960	-0.410203	-0.318894	-0.541053

```
from sklearn.cluster import KMeans
# Create an empty list to store the within-cluster sum of squares
(WCSS) for each k value
css = []
```

```python
# Calculate the WCSS for k values from 1 to 10 using the k-means
algorithm
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    kmeans.fit(pc_df)
    css.append(kmeans.inertia_)

# Plot the WCSS values against the number of clusters
plt.plot(range(1, 11), css)
plt.title("Applying Elbow Method")
plt.xlabel("No 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(
```



```
# Perform k-means clustering with k=5 as elbow is at k = 5
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=0).fit(x)

# Add a column to the data DataFrame to store the cluster number for
each data point
df['cluster_num'] = kmeans.labels_
new_df['cluster_num'] = kmeans.labels_
# Print the cluster labels, within-cluster sum of squares (inertia),
number of iterations, and cluster centers
print("Cluster labels:", kmeans.labels_)
print("Within-cluster sum of squares (inertia):", kmeans.inertia_)
print("Number of iterations:", kmeans.n_iter_)
print("Cluster centers:", kmeans.cluster_centers_)
```

```
Cluster labels: [3 3 4 0 4 4 3 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
2 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 4 4 4 4 4 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 4 2
4 2]
Within-cluster sum of squares (inertia): 175715.51559564698
Number of iterations: 4
Cluster centers: [[4.17619048e+01 2.85714286e-01 3.90476190e+00
1.74095238e+00
  9.52380952e-02 9.04761905e-01 1.82857143e+02 8.56666667e+01
  6.81428571e-01]
 [6.69565217e+01 2.43478261e+00 3.52173913e+00 1.17448696e+01
  1.30434783e-01 9.13043478e-01 4.50000000e+01 1.07173913e+02
  1.20608696e+00]
 [3.92500000e+01 1.66666667e-01 4.08333333e+00 1.47550000e+00
  5.83333333e-01 5.83333333e-01 3.50000000e+02 7.45833333e+01
  6.55000000e-01]
 [8.66250000e+01 4.87500000e+00 4.00000000e+00 3.70000000e+00
  1.25000000e-01 1.00000000e+00 1.13625000e+02 1.56000000e+02
  1.38125000e+00]
 [1.01300000e+02 6.50000000e+00 4.00000000e+00 4.60100000e+00
  1.00000000e-01 1.00000000e+00 2.82500000e+02 1.95400000e+02
  1.54300000e+00]]

/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(
```

```
from collections import Counter
Counter(kmeans.labels_)
```

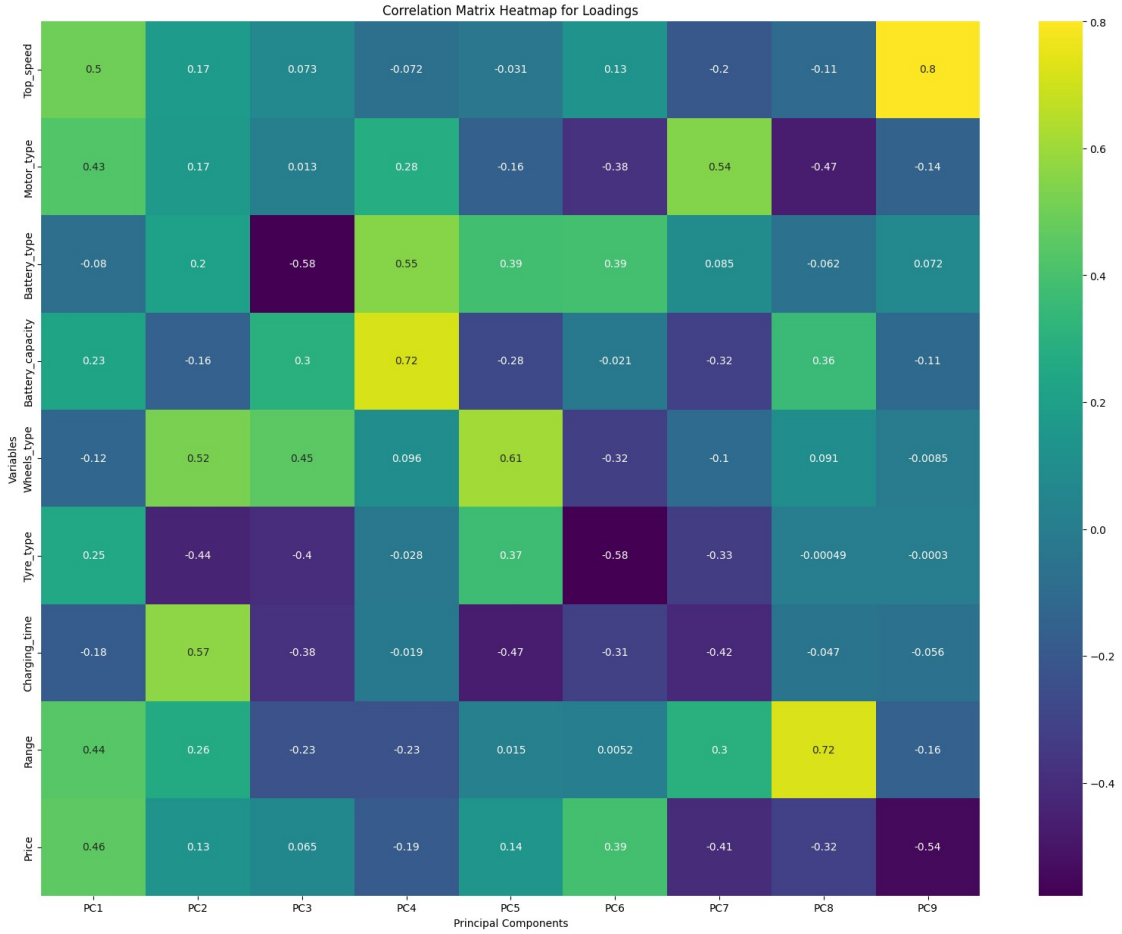
```
Counter({3: 8, 4: 10, 0: 21, 1: 23, 2: 12})
```

```
# Calculate the correlation coefficients between the original
variables and the principal components
loadings = pca.components_
```

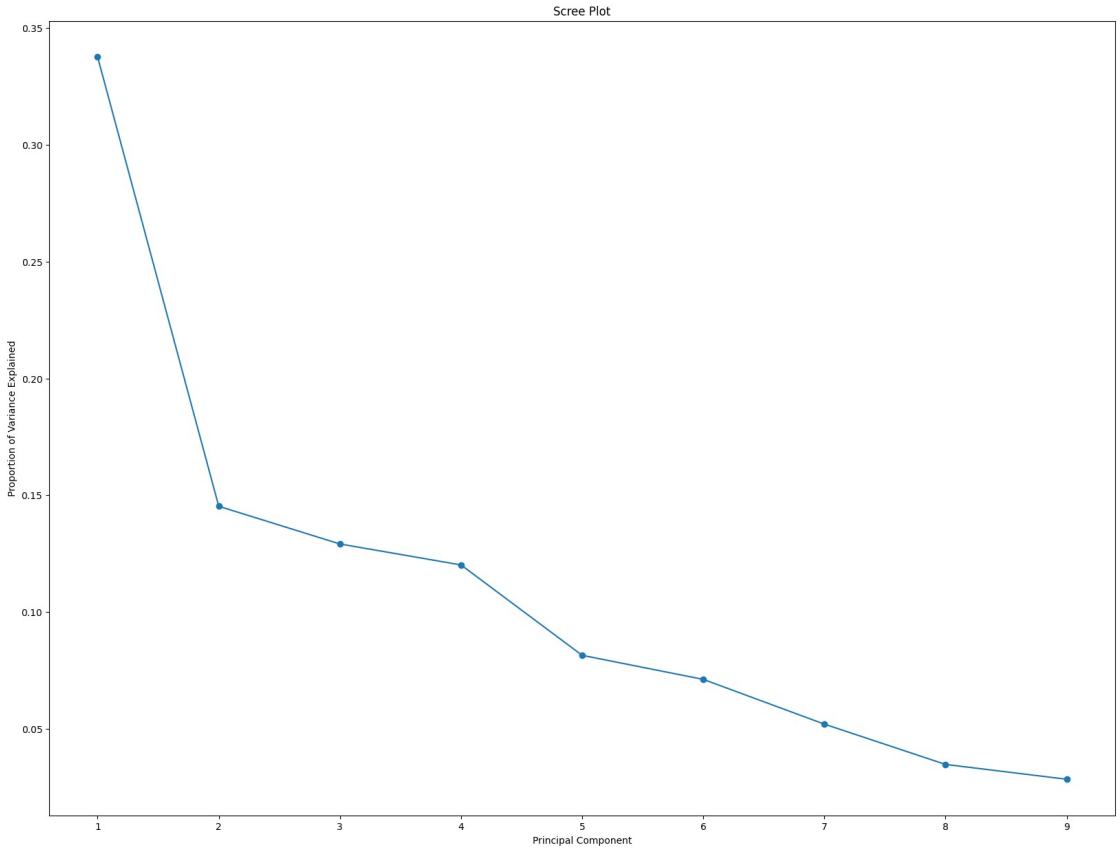
```
num_pc = pca.n_features_
pc_list = ["PC" + str(i) for i in list(range(1, num_pc + 1))]
loadings_df = pd.DataFrame.from_dict(dict(zip(pc_list, loadings)))
loadings_df['variable'] = x.columns.values
loadings_df = loadings_df.set_index('variable')

# Print the resulting DataFrame
print(loadings_df)
```

```
                     PC1       PC2       PC3       PC4       PC5
PC6  \
variable

Top_speed          0.496808  0.173140  0.072899 -0.072391 -0.030762
0.132798
Motor_type         0.426456  0.169350  0.013222  0.278125 -0.158442 -
0.382314
Battery_type      -0.080498  0.202104 -0.577957  0.549482  0.385860
0.389401
Battery_capacity  0.229002 -0.157856  0.299482  0.718106 -0.275189 -
0.020518
Wheels_type       -0.120248  0.524540  0.453736  0.096318  0.609006 -
0.324609
Tyre_type          0.250336 -0.441184 -0.403794 -0.028317  0.372016 -
0.575410
Charging_time     -0.184622  0.567384 -0.379403 -0.018833 -0.469659 -
0.309166
Range              0.437073  0.259303 -0.231655 -0.233224  0.015452
0.005210
Price              0.458421  0.128958  0.065196 -0.187009  0.138675
0.389960

                     PC7       PC8       PC9
variable
Top_speed         -0.203974 -0.105739  0.800801
Motor_type         0.544790 -0.472499 -0.143556
Battery_type       0.084643 -0.061754  0.072182
Battery_capacity -0.317545  0.357693 -0.111109
Wheels_type       -0.103315  0.090764 -0.008513
Tyre_type         -0.330674 -0.000492 -0.000299
Charging_time     -0.422130 -0.047088 -0.055812
Range              0.295160  0.722259 -0.156936
Price             -0.410203 -0.318894 -0.541053

/usr/local/lib/python3.10/dist-packages/sklearn/utils/
deprecation.py:101: FutureWarning: Attribute `n_features_` was
deprecated in version 1.2 and will be removed in 1.4. Use
`n_features_in_` instead.
  warnings.warn(msg, category=FutureWarning)
```

```
# Plot a correlation matrix heatmap for the loadings
plt.rcParams['figure.figsize'] = (20,15)
s.heatmap(loadings_df, annot=True, cmap='viridis')
plt.title('Correlation Matrix Heatmap for Loadings')
plt.xlabel('Principal Components')
plt.ylabel('Variables')
plt.show()
```




```
# Calculate and plot the scree plot for the principal components
plt.plot(range(1, num_pc + 1), pca.explained_variance_ratio_,
marker='o')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.title('Scree Plot')
plt.show()

# Transform the data using PCA and get the principal component scores
pca_scores = pca.transform(x)
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432:
UserWarning: X has feature names, but PCA was fitted without feature
names
  warnings.warn(
```

```
df
```

	Model	Price	Range	Charging_time	Top_speed	Motor_type
0	Ola S1	1.00	128.0	122	95.0	8
1	TVS iQube Electric	1.60	100.0	122	78.0	0
2	Ather 450X	0.98	146.0	305	90.0	11
3	Bajaj Chetak	1.22	90.0	240	63.0	0
4	Ola S1 Pro	1.24	181.0	240	116.0	8
..	...	...	...	...	...	...
69	Cyborg Bob-e	0.95	110.0	40	85.0	0
70	Hop Oxo	1.64	150.0	300	95.0	1

```
71   Komaki XGT Classic   1.11  105.0           300       95.0
0
72   PURE EV eTryst 350   1.49  140.0           300       85.0
0
73  Atumobile AtumVader   1.00  100.0           300       65.0
0

    Battery_type  Battery_capacity  Wheels_type  Tyre_type
cluster_num
0              4              3.00            1          1
3
1              4              3.04            0          1
3
2              4              3.70            0          1
4
3              4              3.04            0          1
0
4              4              4.00            1          1
4
..           ...               ...          ...        ...          ..
.
69             0              2.88            0          1
1
70             4              3.75            0          1
4
71             4              3.75            0          1
2
72             4              3.50            0          1
4
73             4              2.30            3          0
2

[74 rows x 11 columns]
```

```
#One Hot Encoding
df = pd.get_dummies(df, prefix=['cluster'], columns=['cluster_num'])
```

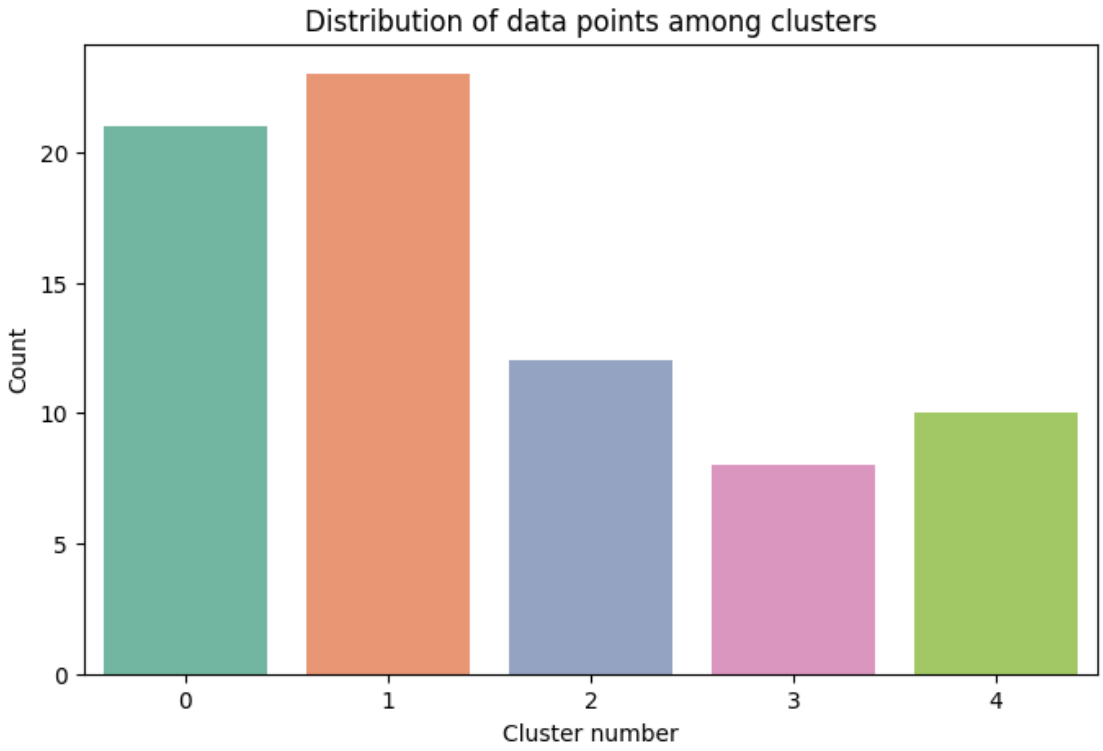
```
df
```

```
                  Model  Price  Range  Charging_time  Top_speed
Motor_type  \
0                Ola S1   1.00  128.0           122       95.0
8
1    TVS iQube Electric   1.60  100.0           122       78.0
0
2            Ather 450X   0.98  146.0           305       90.0
11
3          Bajaj Chetak   1.22   90.0           240       63.0
0
4            Ola S1 Pro   1.24  181.0           240      116.0
8
```

```
..                  ...    ...    ...            ...        ...
...
69         Cyborg Bob-e   0.95  110.0             40       85.0
0
70              Hop Oxo   1.64  150.0            300       95.0
1
71   Komaki XGT Classic   1.11  105.0            300       95.0
0
72   PURE EV eTryst 350   1.49  140.0            300       85.0
0
73  Atumobile AtumVader   1.00  100.0            300       65.0
0

    Battery_type  Battery_capacity  Wheels_type  Tyre_type  cluster_0
\
0              4              3.00            1          1          0

1              4              3.04            0          1          0

2              4              3.70            0          1          0

3              4              3.04            0          1          1

4              4              4.00            1          1          0

..           ...               ...          ...        ...        ...

69             0              2.88            0          1          0

70             4              3.75            0          1          0

71             4              3.75            0          1          0

72             4              3.50            0          1          0

73             4              2.30            3          0          0


    cluster_1  cluster_2  cluster_3  cluster_4
0           0          0          1          0
1           0          0          1          0
2           0          0          0          1
3           0          0          0          0
4           0          0          0          1
..        ...        ...        ...        ...
69          1          0          0          0
70          0          0          0          1
71          0          1          0          0
72          0          0          0          1
```

```
73          0          1          0          0

[74 rows x 15 columns]
```

```
plt.figure(figsize=(8,5))
s.countplot(x = df["cluster_num"], palette = 'Set2')
plt.xlabel("Cluster number")
plt.ylabel("Count")
plt.title("Distribution of data points among clusters")
plt.show()
```



Conclusion

From the above graph i.e. Distribution of data points among clusters vs counts it can be concluded that the Ev vehicle which comes under cluster - 1 are more in demand in the market where as cluster number - 3 vehicles are less demand.

ELECTRONIC VEHICLE MARKET

NEHA DWIVEDI

Abstract

Market segmentation becomes a crucial tool for evolving transportation technology such as electric vehicles (EVs) in emerging markets to explore and implement for extensive adoption. EVs adoption is expected to grow phenomenally in near future as low emission and low operating cost vehicle, and thus, it drives a considerable amount of forthcoming academic research curiosity. The main aim of this study is to explore and identify distinct sets of potential buyer segments for EVs based on psychographic, behavioral, and socio-economic characterization by employing an integrated research framework of 'perceived benefits-attitude-intention'. The study applied robust analytical procedures including cluster analysis, multiple discriminant analysis and Chi-square test to operationalize and validate segments from the data collected of 563 respondents using a cross-sectional online survey. The findings posit that the three distinct sets of young consumer groups have been identified and labelled as 'Conservatives', 'Indifferents', and 'Enthusiasts' which are deemed to be buddying EV buyers The implications are recommended, which may offer some pertinent guidance for scholars and policymakers to encourage EVs adoption in the backdrop of emerging sustainable transport market.

Keywords : Electric vehicles, Market segmentation, Cluster analysis, Attitude towards electric vehicles

INTRODUCTION:

Segmenting for Electric Vehicle Market

The market segmentation approach aims at defining actionable, manageable, homogenous subgroups of individual customers to whom the marketers can target with a similar set of marketing strategies. In practice, there are two ways of segmenting the market-a-priori and post-hoc. An a-priori approach utilizes predefined characteristics such as age, gender, income, education, etc. to predefine the segments followed by profiling based on a host of measured variables (behavioral, psychographic or benefit). In the post-hoc approach to segmentation on other hand, the segments are identified based on the relationship among the multiple measured variables. The commonality between both approaches lies in the fact that the measured variables determine the 'segmentation theme'. The present study utilizes an a-priori approach to segmentation so as to divide the potential EV customers into sub-groups.

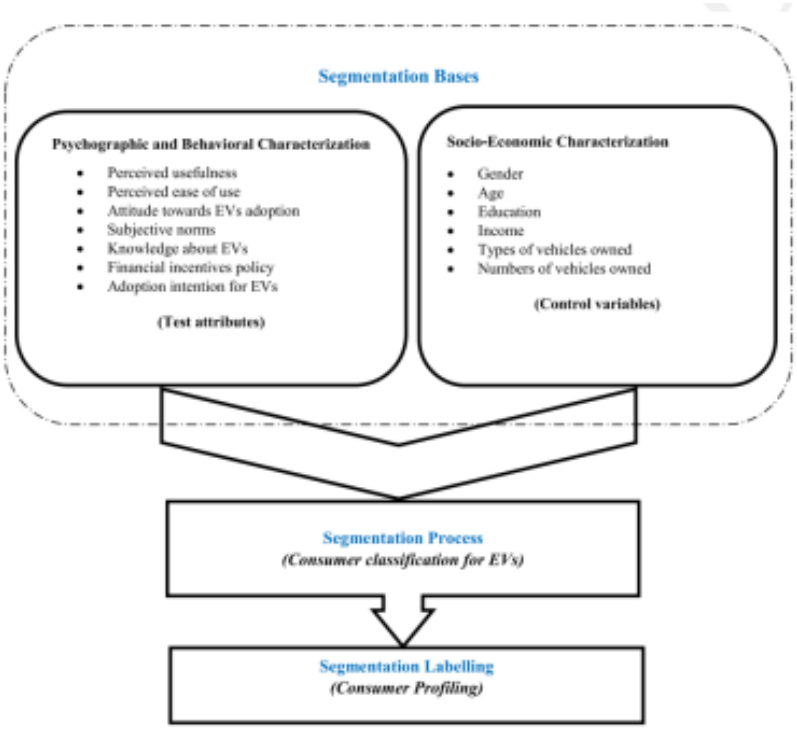


Fig: Market Segmentation Electric Vehicles

It is argued that the blended approach of psychographic and socioeconomic attributes for market segmentation enables the formulation of sub-market strategies which in turn satisfy the specific tastes and preferences of the consumer groups.

Straughan and Roberts presented a comparison between the usefulness of psychographic, demographic, and economic characteristics based on consumer evaluation for eco-friendly products.They pinpointed the perceived superiority of the psychographic characteristics over the socio-demographic and economic ones in explaining the environmentally-conscious consumer behavior and thus, the study recommended the use of psychographic characteristics in profiling the consumer segments in the market for eco-friendly products. The present study adds perceived-benefit characteristics guided by blended psychographic and socio-economic aspects for segmenting the consumer market

PROBLEM STATEMENT:

Electronic Vehical Market

Data Sources:

Importing the Dataset: We will import the dataset that we need to use. So here, we are using the Car Rental Data. It can be imported using the below code
df=pd.read_csv("ElectricCarData_Clean.csv")

Data Pre-processing: (steps and libraries used)

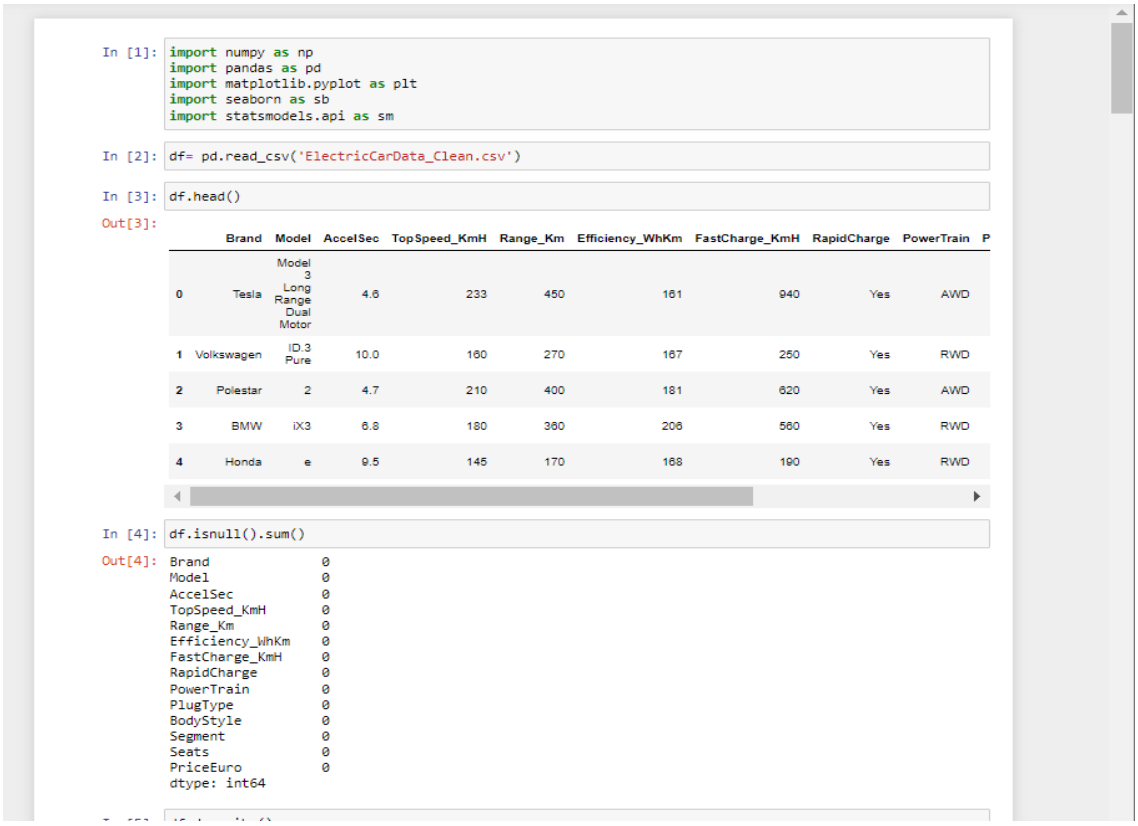
Importing Libraries:
We will import the libraries for our model, which is part of data pre-processing. The code is given below:

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

- Numpy we have imported for the performing mathematics calculation.
- Matplotlib is for plotting the graph, and pandas are for managing the dataset.
- Seaborn is for data visualization library, it is based on matplotlib.

Exploratory Data Analysis:

Exploratory data analysis, or EDA, is a detailed analysis intended to reveal a data set's underlying structure. It is significant for a business because it reveals trends, patterns, and linkages that are not immediately obvious.



```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sb
        import statsmodels.api as sm

In [2]: df= pd.read_csv('ElectricCarData_Clean.csv')

In [3]: df.head()

Out[3]:
```

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	RapidCharge	PowerTrain	P
0	Tesla	Model 3 Long Range Dual Motor	4.6	233	450	161	940	Yes	AWD	
1	Volkswagen	ID.3 Pure	10.0	160	270	167	250	Yes	RWD	
2	Polestar	2	4.7	210	400	181	620	Yes	AWD	
3	BMW	iX3	6.8	180	360	206	560	Yes	RWD	
4	Honda	e	9.5	145	170	168	190	Yes	RWD	

```
In [4]: df.isnull().sum()

Out[4]: 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
```

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

Out[5]:

	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

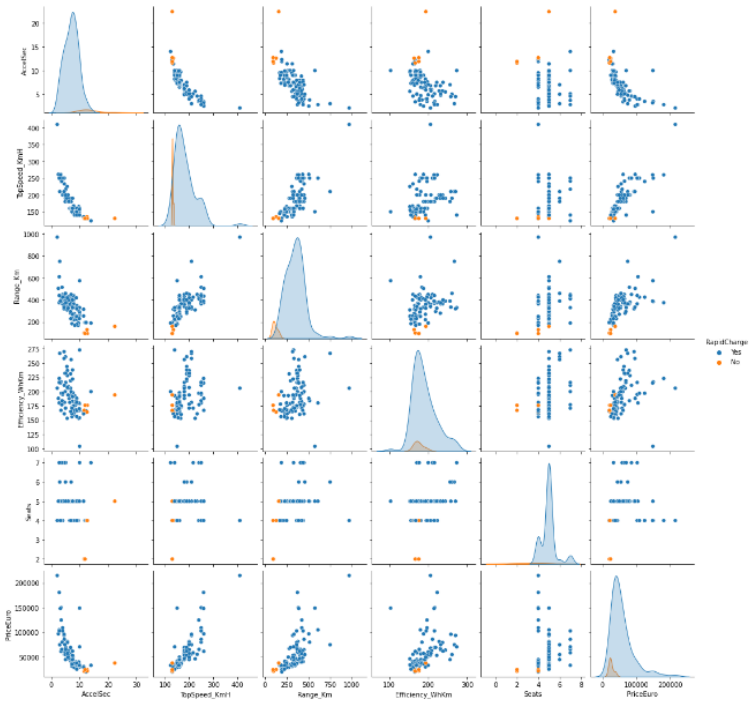
```
In [6]: df.info()
```

```
<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
```

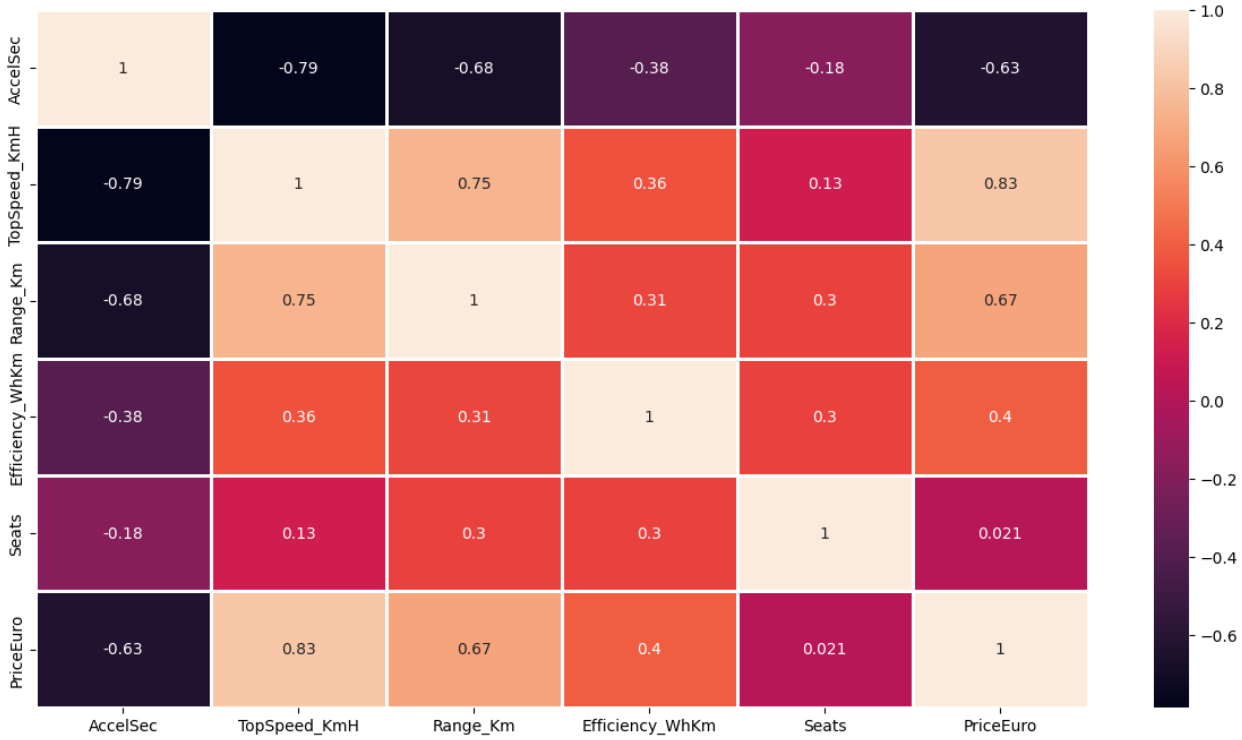
Pairplot of all the columns based on Rapid Charger presence

```
In [7]: sb.pairplot(df,hue='RapidCharge')
```

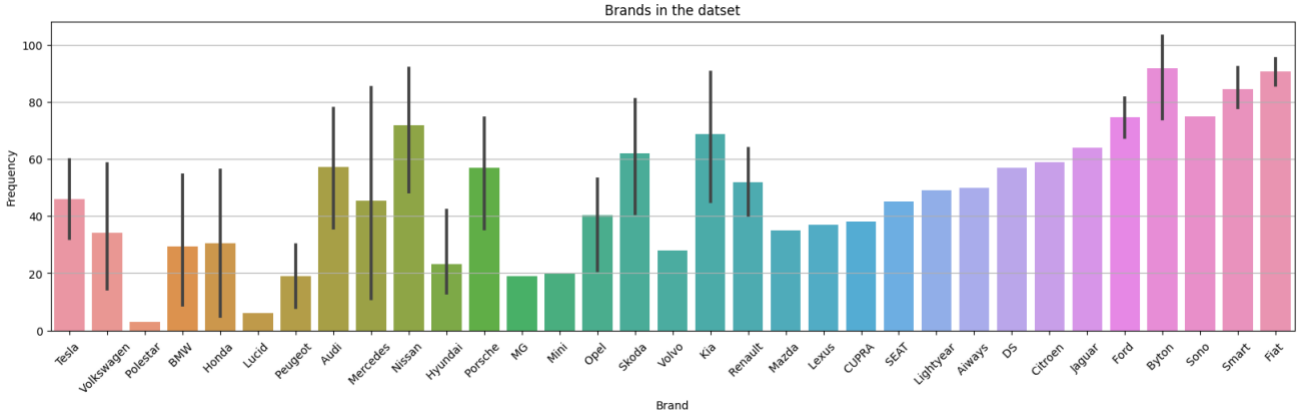
Out[7]: <seaborn.axisgrid.PairGrid at 0x1607f2879a0>



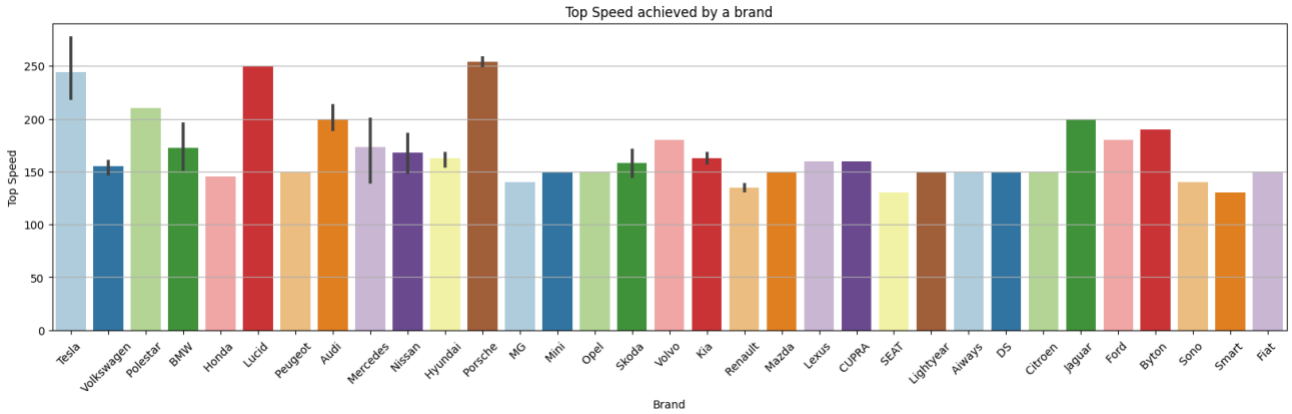
Correlation Matrix: A correlation matrix is simply a table that displays the correlation. It is best used in variables that demonstrate a linear relationship between
each other. Coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values through the heatmap in the below figure. The
relationship between two variables is usually considered strong when their correlation coefficient value is larger than 0.7



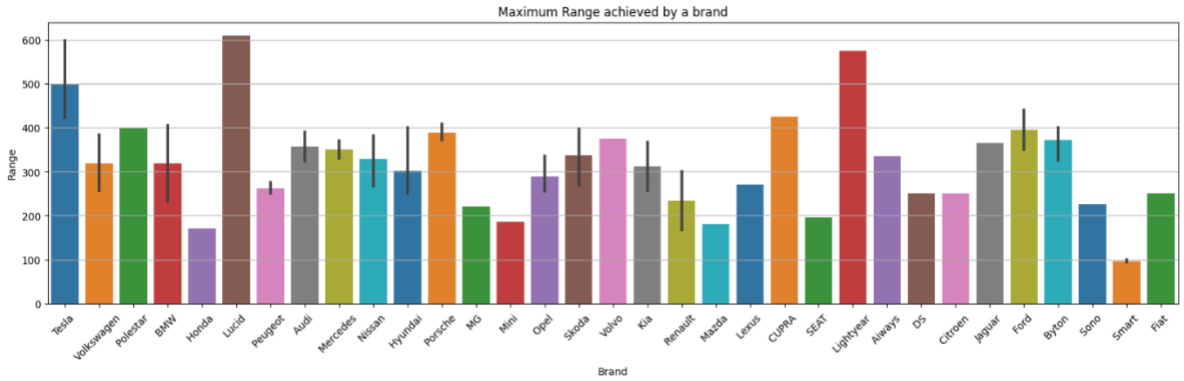
Frequency of the Brands in the dataset



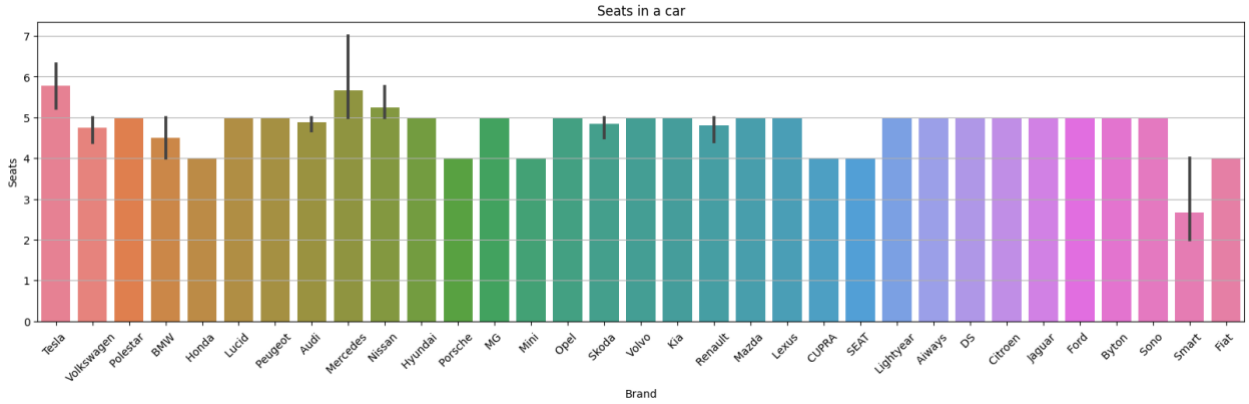
Top speeds achieved by the cars of a brand



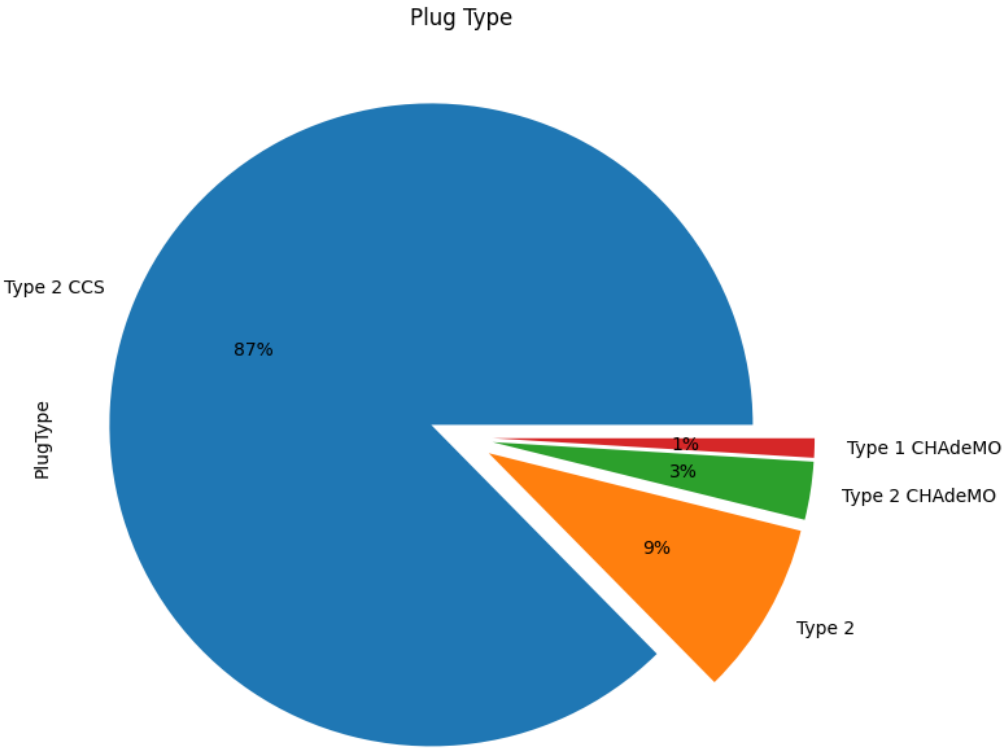
Range a car can achieve



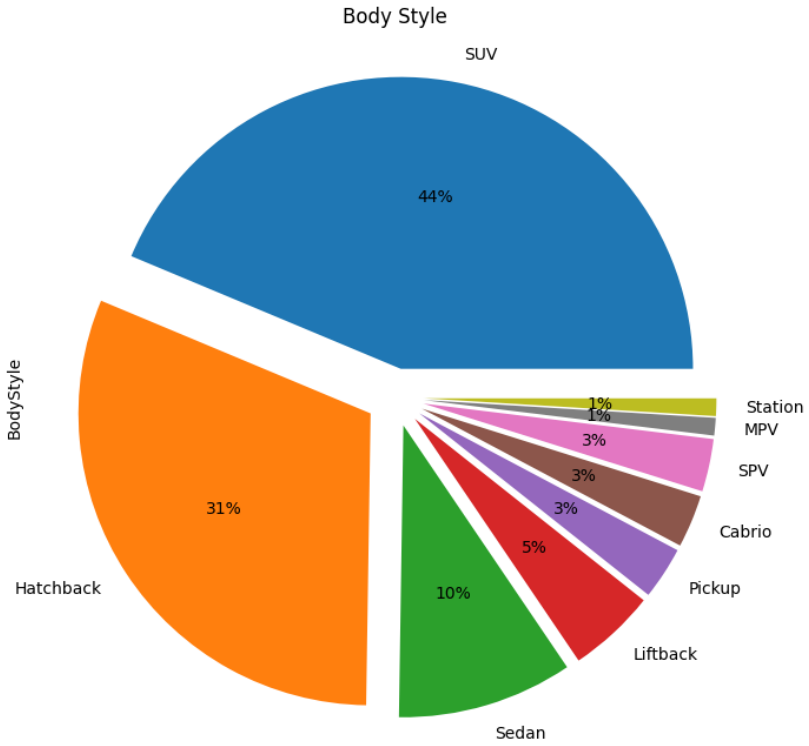
Number of seats in each car



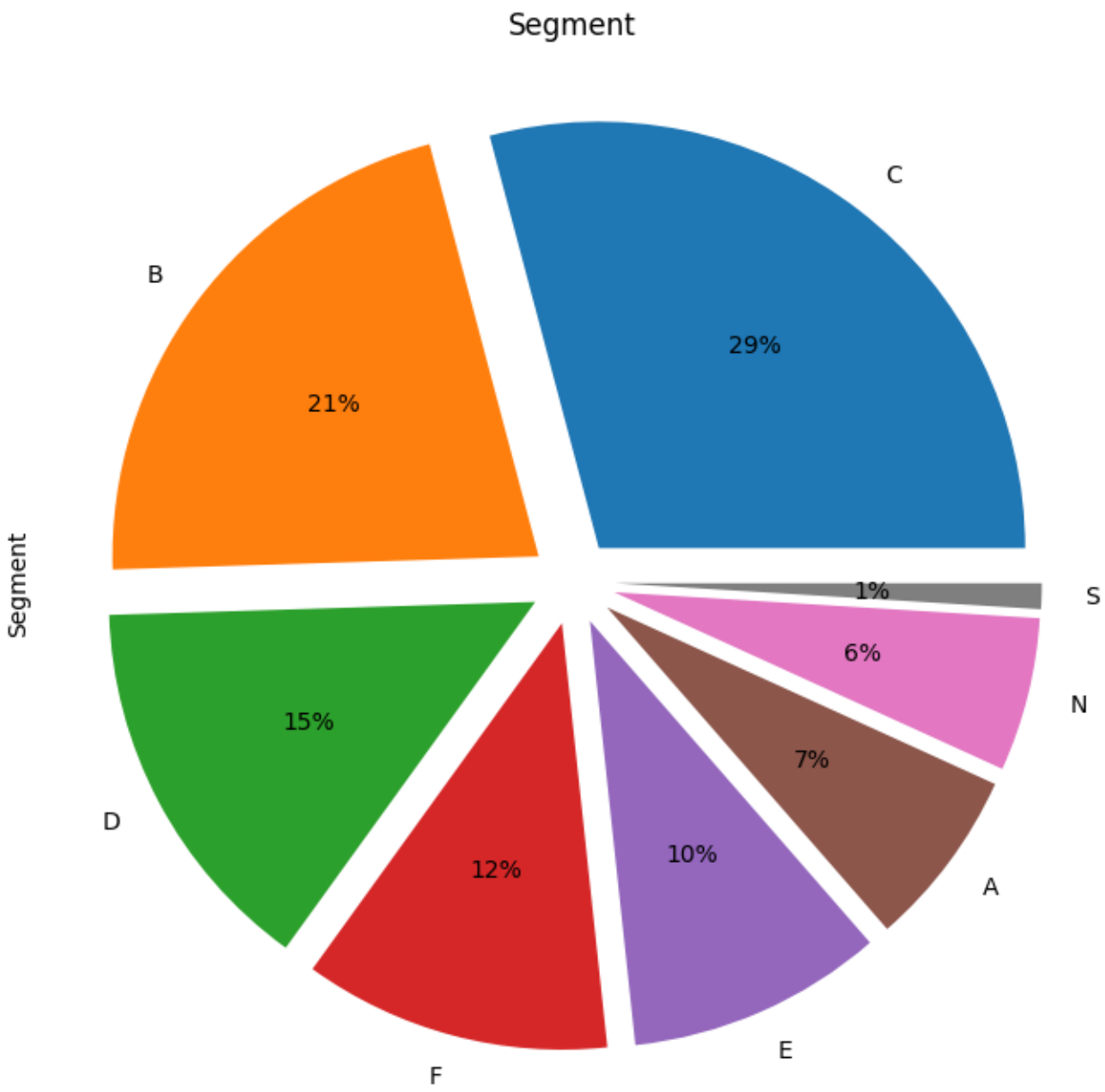
Type of Plug used for charging



Cars and their body style



Segment in which the cars fall under



Regression

In [17]:
```
x=df[['AccelSec','Range_Km','TopSpeed_KmH','Efficiency_WhKm']]
y=df['PriceEuro']
```

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=365)
```

In [20]:
```
lr.fit(X_train, y_train)
pred = lr.predict(X_test)
```

In [21]:
```
#Finding out the R-squared value

from sklearn.metrics import r2_score
r2=(r2_score(y_test,pred))
print(r2*100)
```

```
78.35225979903608
```

In [22]:
```
#Putting Yes value as 1 and No value as 0 for Logistic Regression

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

In [23]:
```
y1=df[['RapidCharge']]
x1=df[['PriceEuro']]
```

In [24]:
```
from sklearn.model_selection import train_test_split
X1_train, X1_test, y1_train, y1_test = train_test_split(x1, y1, test_size=0.2,random_state=365)
```

In [25]:
```
#Importing Logistic Regression

from sklearn.linear_model import LogisticRegression
```

In [26]:
```
log= LogisticRegression()
```

In [27]:
```
log.fit(X1_train, y1_train)
pred1 = log.predict(X1_test)
pred1
```

```
C:\Users\rajen\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[27]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int64)
```

Linear Regression:

The first part of the code selects specific columns from a DataFrame df and assigns them to the variable x. It selects columns 'AccelSec', 'Range_Km', 'TopSpeed_KmH', and 'Efficiency_WhKm'.
The target variable 'PriceEuro' is assigned to the variable y.
The code then imports necessary libraries for linear regression and creates a LinearRegression object named lr.

The data is split into training and testing sets using the train_test_split function from sklearn.
The split is done with a test size of 30% and a random state of 365.
The lr model is fitted using the training data (X_train, y_train).
Predictions are made on the test data (X_test) using the predict method of lr.
The code calculates the R-squared value, which is a measure of how well the linear regression model fits the data. It uses the r2_score function from sklearn.metrics.
Finally, the R-squared value is printed.

Logistic Regression:

The second part of the code focuses on logistic regression.
The column 'RapidCharge' in the DataFrame df is modified to replace 'No' with 0 and 'Yes' with 1, and the changes are made in-place.
The feature variable 'PriceEuro' is assigned to x1, and the target variable 'RapidCharge' is assigned to y1.
The data is split into training and testing sets using the train_test_split function with a test size of 20% and a random state of 365.
Logistic regression is imported from sklearn, and a LogisticRegression object named log is created.
The log model is fitted using the training data (X1_train, y1_train).
Predictions are made on the test data (X1_test) using the predict method of log.
The code prints the predictions (pred1) from logistic regression.

All the elements of the marketing mix influence each other. They make up the business plan for a company and handle it right, and can give it great success. The marketing mix needs a lot of understanding, market research and consultation with several people, from users to trade to manufacturing and several others.

GITHUB:

https://github.com/NehaDwivedi842/Feynnlab_internship

OVERVIEW :

Electric vehicles have come to light as a potential option to lessen carbon emissions and advance sustainability as people's awareness of the environmental impact of transportation grows. The adoption of electric vehicles is still constrained, though, due to a number of issues, such as their high cost, short range, and scarcity of charging stations. Understanding the current situation of the electric car market and identifying prospective areas for development are crucial for promoting the wide adoption of electric vehicles.

Agitate: In order to better understand the electric vehicle market, we analyse a dataset that contains data on electric vehicles in this study. We examine a number of facets of the market for electric cars, such as the distribution of electric vehicles by area, manufacturer, and model, as well as whether electric vehicles qualify for subsidies for clean alternative fuel vehicles. We also look at the distribution of electric cars by price range and the electric utility providers that supply the electricity for the electric vehicles.

Solution: By examining the electric car dataset, we can spot potential flaws and offer suggestions to organizations that support the use of electric vehicles. We discuss the prevalence of electric vehicles in cities and suggest that advocacy organizations concentrate their efforts on advancing the adoption of electric vehicles in metropolitan areas. We also advise promoting the use of renewable energy sources to power electric cars, working with electric utility companies to ensure that the infrastructure is in place to support the widespread adoption of electric vehicles, and pushing for the construction of more charging stations in public areas.

Benefits: This study provides valuable insights into the electric car market, including the distribution of electric cars by location, make, and model, as well as the eligibility of electric cars for clean alternative fuel vehicle incentives. The recommendations we make for electric car advocacy groups can help to promote the adoption of electric cars and support the transition to a more sustainable transportation system. By reducing carbon emissions and promoting sustainability, the widespread adoption of electric cars can have a significant positive impact on the environment and contribute to a more sustainable future.

DATA SOURCES:

- Global EV Outlook 2022

The Global EV Outlook is an annual publication that identifies and discusses recent developments in electric mobility across the globe. It is developed with the support of the members of the Electric Vehicles Initiative (EVI).

Combining historical analysis with projections to 2030, the report examines key areas of interest such as electric vehicle and charging infrastructure deployment, energy use, CO2 emissions, battery demand and related policy developments. The report includes policy recommendations that incorporate lessons learned from leading markets to inform policy makers and stakeholders with regard to policy frameworks and market systems for electric vehicle.

DATA PREPROCESSING:

Handling Missing Values by Dropping Rows

In our dataset, we have observed that only a small number of rows have missing values in certain columns. Given the overall size of the dataset, dropping these rows will likely have a minimal impact on the accuracy and representativeness of our analysis. Furthermore, dropping these rows with missing values is a simple and straightforward approach that allows us to work with a clean dataset without having to make assumptions or impute values that may introduce bias or errors into our analysis.

The following columns have missing values:

- Model (78 missing values)
- Electric Range (1 missing value)
- Base MSRP (1 missing value)
- Legislative District (148 missing values)
- DOL Vehicle ID (1 missing value)
- Vehicle Location (19 missing values)
- Electric Utility (227 missing values)
- 2020 Census Tract (1 missing value)

Considering the relatively small number of missing values in these columns, we can safely drop these rows without significantly affecting the overall dataset. This way, we can proceed with our analysis using a dataset free of missing values.

Feature Engineering: Extracting Latitude and Longitude

In our dataset, the Vehicle Location column contains both latitude and longitude coordinates as a string. To make these coordinates more accessible for further analysis, we will perform the following feature engineering steps:

- Convert the Vehicle Location column to string type.
- Create two new columns in the DataFrame, latitude and longitude.
- Extract latitude and longitude values from the Vehicle Location column and store them in the respective new columns.
- To achieve this, we define a function called extract_coordinates(), which takes two arguments: the input string (containing the coordinates) and the index (0 for latitude and

1 for longitude). The function uses regular expressions to find and return the floating-point numbers representing the coordinates. We then use the apply() function to apply this custom function to each element of the Vehicle Location column and populate the new latitude and longitude columns.

- Finally, we drop any rows with missing values in the latitude and longitude columns to ensure a clean dataset for further analysis.

Feature Engineering: Creating a 'Location' Column
In our dataset, we have three columns representing different levels of geographical divisions: County, City, and State. To create a more informative and combined representation of these geographical attributes, we will create a new column called Location.

- The Location column will be a concatenation of the County, City, and State columns, with each value separated by a comma. For example, if a row has the values "Yakima" for County, "Yakima" for City, and "WA" for State, the corresponding Location value will be "Yakima, Yakima, WA".

Feature Engineering: Creating a Price_Range_Category Column
In our dataset, we have observed an unusual distribution of values in the Base MSRP column, with a large number of vehicles having a value of 0. This could potentially indicate missing or unknown values in the dataset. To account for this uncertainty and still make use of the available data, we have decided to create a new column called Price_Range_Category based on the Base MSRP values.
We have defined four categories for the Price_Range_Category column:

- "Unknown": If the Base MSRP value is 0, we assign this category as it might indicate missing or unknown values.
- "Low": If the Base MSRP value is less than 40,000, we assign this category.
- "Medium": If the Base MSRP value is between 40,000 and 60,000, we assign this category.
- "High": If the Base MSRP value is greater than 60,000, we assign this category.

By creating this new column, we can better understand the distribution of electric vehicle prices in our dataset and account for the potential uncertainty introduced by the large number of 0 values in the Base MSRP column.

Feature Engineering: Creating an 'Electric_Range_Category' Column
In our dataset, we have observed an unusual distribution of values in the 'Electric Range' column, with a large number of vehicles having a value of 0. This could potentially indicate missing or unknown values in the dataset. To account for this uncertainty and still make use of the available data, we have decided to create a new column called 'Electric_Range_Category' based on the 'Electric Range' values.

We have defined four categories for the 'Electric_Range_Category' column:

- "Unknown": If the 'Electric Range' value is 0, we assign this category as it might indicate missing or unknown values.
- "Short": If the 'Electric Range' value is less than 150, we assign this category.
- "Medium": If the 'Electric Range' value is between 150 and 300, we assign this category.
- "Long": If the 'Electric Range' value is greater than 300, we assign this category.

By creating this new column, we can better understand the distribution of electric vehicle ranges in our dataset and account for the potential uncertainty introduced by the large number of 0 values in the 'Electric Range' column.

EXPLORATORY DATA ANALYSIS

The goal of EDA is to gain insights and understanding of the dataset, identify patterns, relationships, and anomalies. Through EDA, we can make informed decisions on how to preprocess and model the data, as well as generate hypotheses for further analysis.

For our electric car dataset, we will focus on the following columns:

- 'Location'
- 'Model Year'
- 'Make'
- 'Model'
- 'Electric Vehicle Type'
- 'Clean Alternative Fuel Vehicle (CAFV) Eligibility'
- 'Electric Range Category'
- 'Price Range Category'
- 'Electric Utility'

We will begin by examining the distribution of data points across various columns, such as the number of electric cars per city or county, the distribution of makes and models, and the distribution of electric vehicle types. We will also visualize the geographical distribution of electric cars using the 'County', 'City', and 'State' columns.

Next, we will explore the relationships between variables, such as the relationship between the electric range and the base MSRP of electric vehicles. We will also analyze the distribution of Clean Alternative Fuel Vehicle (CAFV) eligibility across different makes, models, and electric vehicle types.

Furthermore, we will investigate the trends in the adoption of electric cars over time, focusing on the 'Model Year' column. We will identify patterns and significant changes in the popularity of various makes and models of electric vehicles, as well as any trends related to electric vehicle types and CAFV eligibility.

Finally, we will examine any potential outliers or anomalies in the dataset that may warrant further investigation. Throughout the EDA process, we will visualize our findings using various plotting libraries such as Seaborn and Plotly, to help communicate the insights effectively.

By the end of the EDA, we will have a comprehensive understanding of the electric car dataset, enabling us to make informed decisions on how to proceed with preprocessing, feature engineering, and modeling, as well as guiding our hypotheses for further analysis.

Electric Car Dataset Analysis: Location

The 'Location' column in the Electric Car dataset provides information about the location of the electric cars. The following are the value counts of the 'Location' column:

Location	Count
King, Seattle, WA	21942
King, Bellevue, WA	6476
King, Redmond, WA	4641
Clark, Vancouver, WA	4462
King, Kirkland, WA	3920
King, Sammamish, WA	3650
King, Renton, WA	3112

Snohomish, Bothell, WA	3014
Thurston, Olympia, WA	3014
Pierce, Tacoma, WA	2651

Conclusion

From the above value counts, we can make the following inferences and gain insights:

- The top 10 electric cars in the dataset are primarily located in the state of Washington (WA), with no cars from other states.
- The highest number of electric cars are located in King county, with Seattle having the highest count of 21942 cars. This indicates that the majority of electric cars are concentrated in urban areas, particularly in large cities like Seattle and Bellevue.
- The next most common location is Clark county, with Vancouver having a count of 4462 cars.

Based on the analysis, the following recommendations can be made for electric car advocacies:

1. Focus on urban areas: Since the majority of electric cars are located in urban areas, electric car advocates should focus on promoting the use of electric cars in cities, particularly in large cities like Seattle and Bellevue.
2. Expand to other countries: While King and Clark counties have the highest number of electric cars, there is still potential to expand the use of electric cars in other counties in the state of Washington. Advocates should consider reaching out to these counties and promoting the benefits of electric cars.
3. Target specific cities: Within counties with a high number of electric cars, there may be specific cities or neighborhoods with lower adoption rates. Advocacies should target these areas and promote the use of electric cars through targeted marketing and outreach efforts.

Electric Car Dataset Analysis: CAFV Eligibility

The 'CAFV Eligibility' column in the Electric Car dataset provides information about whether a given electric car is eligible for Clean Alternative Fuel Vehicle (CAFV) incentives based on its battery range. The following are the value counts of the 'CAFV Eligibility' column:

CAFV Eligibility	Count
Clean Alternative Fuel Vehicle Eligible	59092
Eligibility unknown as battery range has not been researched	49346
Not eligible due to low battery range	15600

Electric vehicles (EVs) are gaining traction as a sustainable and environmentally friendly alternative to traditional internal combustion engine vehicles. As the transportation sector is a significant contributor to greenhouse gas emissions, the adoption of electric vehicles can play a critical role in reducing the overall carbon footprint. Analyzing the data available on electric vehicles can provide valuable insights and help inform future strategies to promote their adoption.

Based on the value counts provided for the electric car dataset, we can make a few inferences and provide the following recommendations:

1. **Promote clean alternative fuel vehicles:** A significant portion (about 47.5%) of the vehicles in the dataset are classified as "Clean Alternative Fuel Vehicle Eligible." This indicates that there is a growing interest in and adoption of electric vehicles. Efforts should be made to encourage the use of clean alternative fuel vehicles by offering incentives, improving infrastructure, and educating consumers about the environmental and financial benefits of owning such vehicles.
2. **Research and data collection for battery range:** There is a significant number of vehicles (about 39.8%) with "Eligibility unknown as battery range has not been researched." To improve the understanding of the electric vehicle market, it is essential to collect and analyze data on battery range for all electric vehicles. This will help in making better recommendations for consumers and also help in the development of policies and regulations surrounding electric vehicles.
3. **Improve battery technology:** About 12.7% of the vehicles are "Not eligible due to low battery range." This indicates that there is still room for improvement in battery technology. Investing in research and development to enhance battery performance will not only help increase the range of electric vehicles but also improve their overall efficiency and attractiveness to potential buyers.
4. **Expand charging infrastructure:** A significant barrier to the adoption of electric vehicles is the lack of charging infrastructure. Expanding the availability of charging stations, especially in urban and high-traffic areas, will make electric vehicles more accessible and convenient for users.
5. **Increase public awareness:** It is crucial to create awareness about the benefits of electric vehicles and clean alternative fuel options. Public campaigns, educational programs, and partnerships with local communities can help in promoting the adoption of electric vehicles and in reducing the overall carbon footprint.

Electric Car Dataset Analysis: Make

The 'Make' column in the Electric Car dataset provides information about the **top 10 makers** of electric cars. The following are the value counts of the 'Make' column:

Make	Count
TESLA	56906
NISSAN	12912

CHEVROLET	10797
FORD	6635
BMW	5556
KIA	4831
TOYOTA	4631
VOLKSWAGEN	3356
AUDI	2472
VOLVO	2324

Inference

From the above value counts, we can make the following inferences and gain insights:

- The majority of electric cars in the dataset are from the Tesla make, with 56,906 cars falling into this category.
- The next most common makes are Nissan and Chevrolet, with 12,912 and 10,797 cars respectively.
- The rest of the makes have relatively smaller counts, with Volvo having the lowest count of 2,324 cars.

Based on the analysis, the following recommendations can be made for electric car advocacies:

1. **Promote Tesla electric cars:** Since the majority of electric cars in the dataset are from the Tesla make, electric car advocates should focus on promoting Tesla electric cars to encourage more people to purchase them.
2. **Encourage diversity in make:** While Tesla is the dominant make in the dataset, there are other makes with a significant number of cars. Electric car advocates should encourage diversity in make and promote the benefits of each make to encourage more people to purchase electric cars.
3. **Advocate for increased availability of less common makes:** Electric car advocates should advocate for increased availability of less common makes, such as Volvo and Audi, to increase consumer choice and promote diversity in the electric car market.

We can also make the following inferences and gain insights:

- The most common model of electric car in the dataset is the Tesla Model 3, with 24,300 cars falling into this category.
- The next most common model is the Tesla Model Y, with 20,609 cars.
- The Nissan Leaf is the third most common model, with 12,890 cars.
- The remaining models have relatively smaller counts, with the Niro having the lowest count of 2,286 cars.

Based on the analysis, the following recommendations can be made for electric car advocacies:

1. **Promote popular models:** Since the most common models in the dataset are the Tesla Model 3 and Model Y, electric car advocates should focus on promoting these models to encourage more people to purchase them.
2. **Encourage diversity in model:** While the most common models are dominated by Tesla and Nissan, there are other models with a significant number of cars. Electric car advocates should encourage diversity in model and promote the benefits of each model to encourage more people to purchase electric cars.
3. **Advocate for increased availability of less common models:** Electric car advocates should advocate for increased availability of less common models, such as the VW ID.4 and Kia Niro, to increase consumer choice and promote diversity in the electric car market.

- Battery electric vehicles (BEVs) are the most common type of electric vehicle in the dataset, with 95,753 cars falling into this category.
- The next most common type is plug-in hybrid electric vehicles (PHEVs), with 28,285 cars.
- There are no other types of electric vehicles in the dataset, indicating that the majority of electric cars are either BEVs or PHEVs.

Recommendations for Advocacies:

Based on the analysis, the following recommendations can be made for electric car advocacies:

1. **Promote the benefits of BEVs and PHEVs:** Since BEVs and PHEVs are the most common types of electric vehicles in the dataset, electric car advocacies should focus on promoting the benefits of these types of vehicles to encourage more people to purchase them.
2. **Educate on the differences between BEVs and PHEVs:** While both types of electric vehicles are common, they have different characteristics and are suited for different use cases. Electric car advocacies should educate consumers on the differences between BEVs and PHEVs to help them make informed purchase decisions.
3. **Advocate for increased availability of different types of electric vehicles:** While BEVs and PHEVs are common, there are other types of electric vehicles, such as fuel cell electric vehicles (FCEVs), that are not represented in the dataset. Electric car advocacies should advocate for increased availability of different types of electric vehicles to increase consumer choice and promote diversity in the electric car market.

- The majority of electric cars in the dataset have an unknown price range category, with 120,656 cars falling into this category. This is likely due to a large number of vehicles having a value of 0 in the 'Base MSRP' column.
- Only a small percentage of electric cars have a defined price range category, with 1,653 cars falling into the 'High' category, 971 cars falling into the 'Low' category, and 758 cars falling into the 'Medium' category.
- The price range categories are defined based on the 'Base MSRP' values, with a value of 0 indicating an unknown price range.

Based on the analysis, the following recommendations can be made for electric car advocacies:

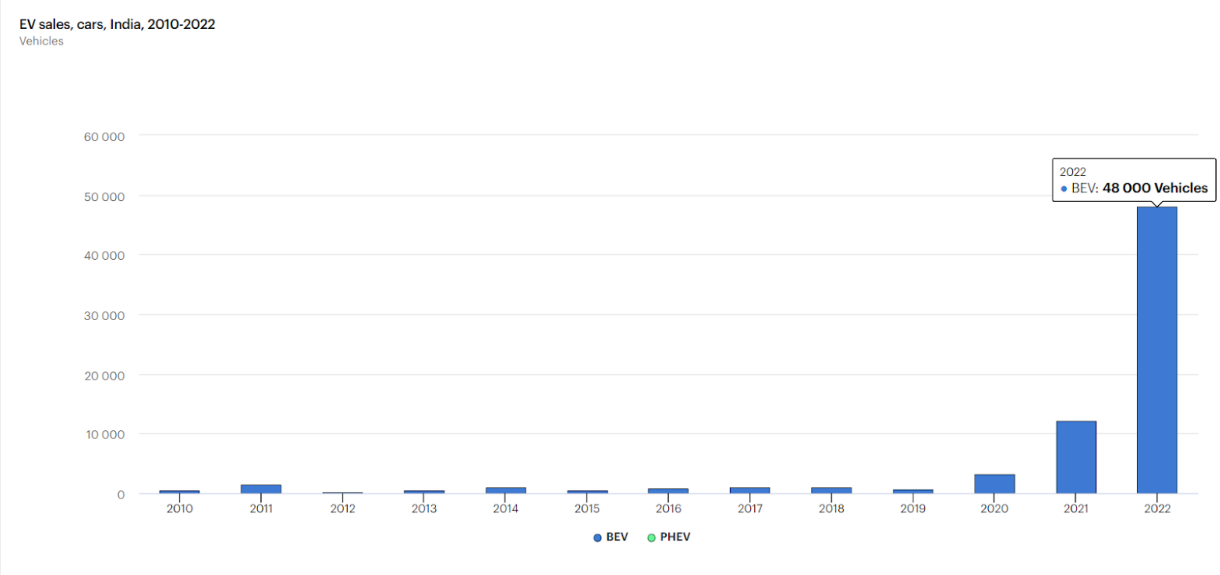
1. **Encourage more transparency in pricing:** Since the majority of electric cars in the dataset have an unknown price range category, electric car advocacies should encourage more transparency in pricing to help consumers make informed purchase decisions.
2. **Focus on affordable options:** Since only a small percentage of electric cars have a defined price range category and fall into the 'Low' and 'Medium' categories, electric car advocacies should focus on promoting affordable options to make electric cars more accessible to a wider range of consumers.
3. **Advocate for more incentives:** Given that electric cars tend to be more expensive than gasoline-powered cars, electric car advocacies should advocate for more incentives and subsidies to make electric cars more affordable and promote their adoption.

- Puget Sound Energy Inc. is the most common electric utility company in the dataset, with 44,926 electric cars being powered by this company.
- The next most common electric utility company is PUGET SOUND ENERGY INC (24,626 cars), followed by CITY OF SEATTLE (23,254 cars).
- There are several other electric utility companies in the dataset, but they have a much lower number of electric cars powered by them.

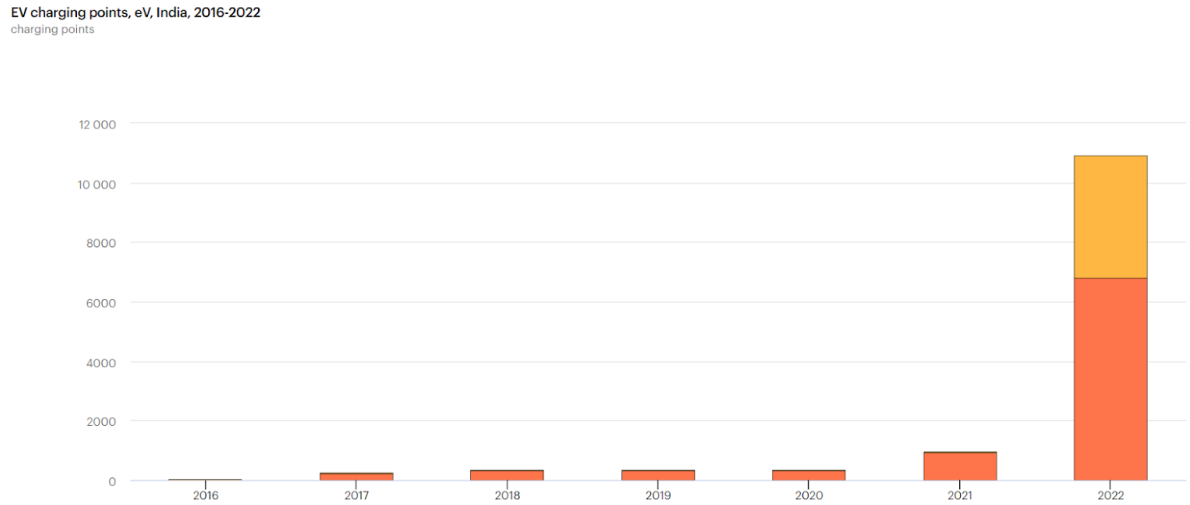
Based on the analysis, the following recommendations can be made for electric car advocacies:

1. **Collaborate with electric utility companies:** Since electric cars require a reliable and efficient electric grid to function, electric car advocacies should collaborate with electric utility companies to ensure that the necessary infrastructure is in place to support the widespread adoption of electric cars.
2. **Promote the use of renewable energy sources:** Many electric utility companies are transitioning to renewable energy sources, such as wind and solar power. Electric car advocacies should promote the use of these renewable energy sources to power electric cars, as they are more sustainable and environmentally friendly.
3. **Advocate for more charging stations:** To support the widespread adoption of electric cars, there need to be more charging stations available for electric car owners to use. Electric car advocacies should advocate for more charging stations to be built in public places, such as parking lots and rest areas, to make it more convenient for electric car owners to charge their cars.

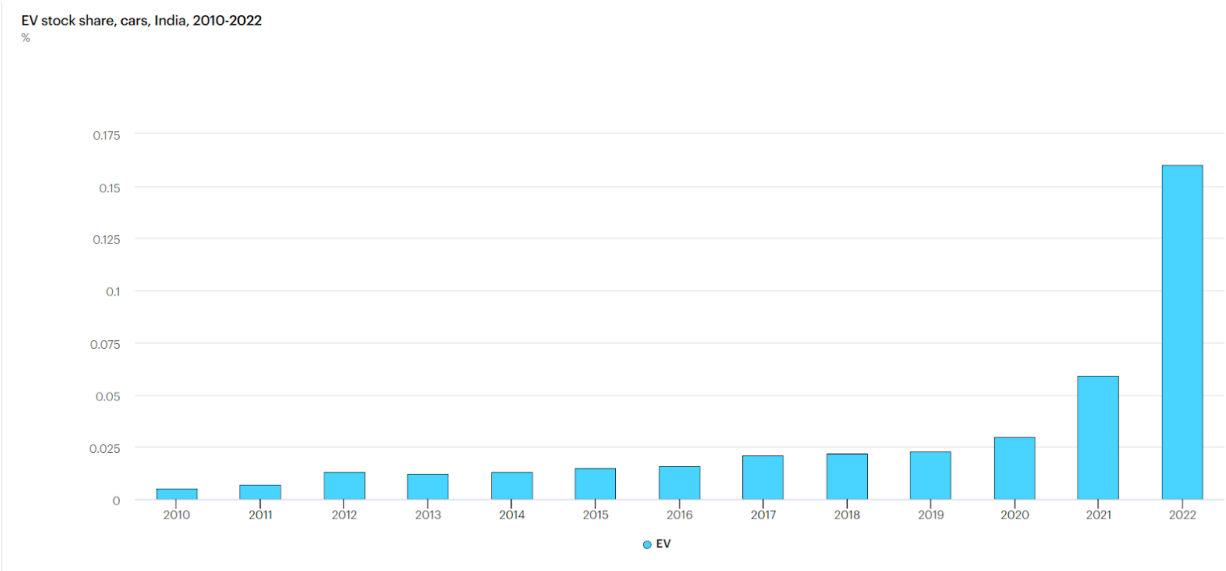
FOR ELECTRIC CARS:
EV SALES:



EV CHARGING POINTS :



EV STOCK SHARE :



In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [5]:
```
#Loading data
df = pd.read_csv("EV_data.csv")
df.head()
```

Out[5]:

	Brand	Model	AccelSec	TopSpeed_KmH	Range_Km	Efficiency_WhKm	FastCharge_KmH	I
0	Tesla	Model 3 Long Range Dual Motor	4.6	233	450	161	940	
1	Volkswagen	ID.3 Pure	10.0	160	270	167	250	
2	Polestar	2	4.7	210	400	181	620	
3	BMW	iX3	6.8	180	360	206	560	
4	Honda	e	9.5	145	170	168	190	

In [6]:
```
df.shape
```

Out[6]:
```
(103, 14)
```

In [7]:
```
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
```

In [8]:
```
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
AccelSec	103.0	7.396117	3.017430	2.1	5.1	7.3	9.0	22.4
TopSpeed_KmH	103.0	179.194175	43.573030	123.0	150.0	160.0	200.0	410.0
Range_Km	103.0	338.786408	126.014444	95.0	250.0	340.0	400.0	970.0
Efficiency_WhKm	103.0	189.165049	29.566839	104.0	168.0	180.0	203.0	273.0
Seats	103.0	4.883495	0.795834	2.0	5.0	5.0	5.0	7.0
PriceEuro	103.0	55811.563107	34134.665280	20129.0	34429.5	45000.0	65000.0	215000.0

In [9]:
```
df.columns
```

Out[9]:
```
Index(['Brand', 'Model', 'AccelSec', 'TopSpeed_KmH', 'Range_Km',
       'Efficiency_WhKm', 'FastCharge_KmH', 'RapidCharge', 'PowerTrain',
       'PlugType', 'BodyStyle', 'Segment', 'Seats', 'PriceEuro'],
      dtype='object')
```

In [10]:
```
#checking Duplicate values
df.duplicated()
```

Out[10]:
```
0      False
1      False
2      False
3      False
4      False
       ...
98     False
99     False
100    False
101    False
102    False
Length: 103, dtype: bool
```

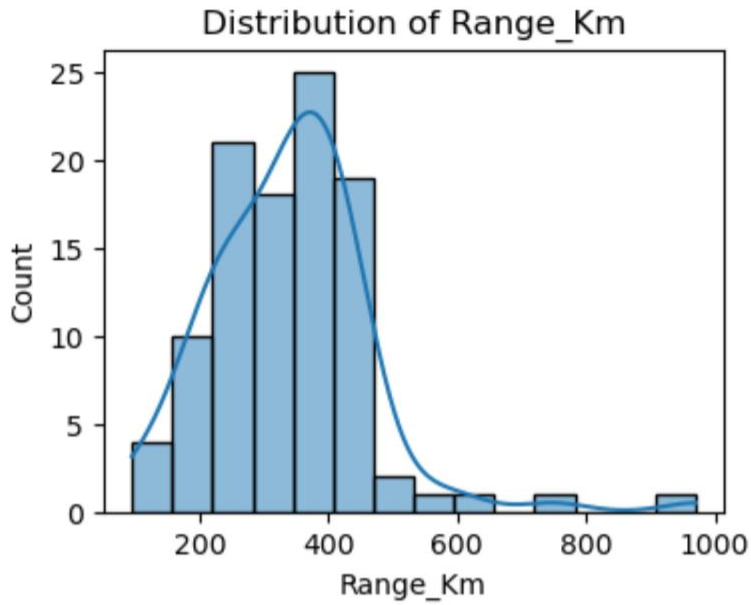
In [11]:
```
df.duplicated().sum()
```

Out[11]:
```
0
```

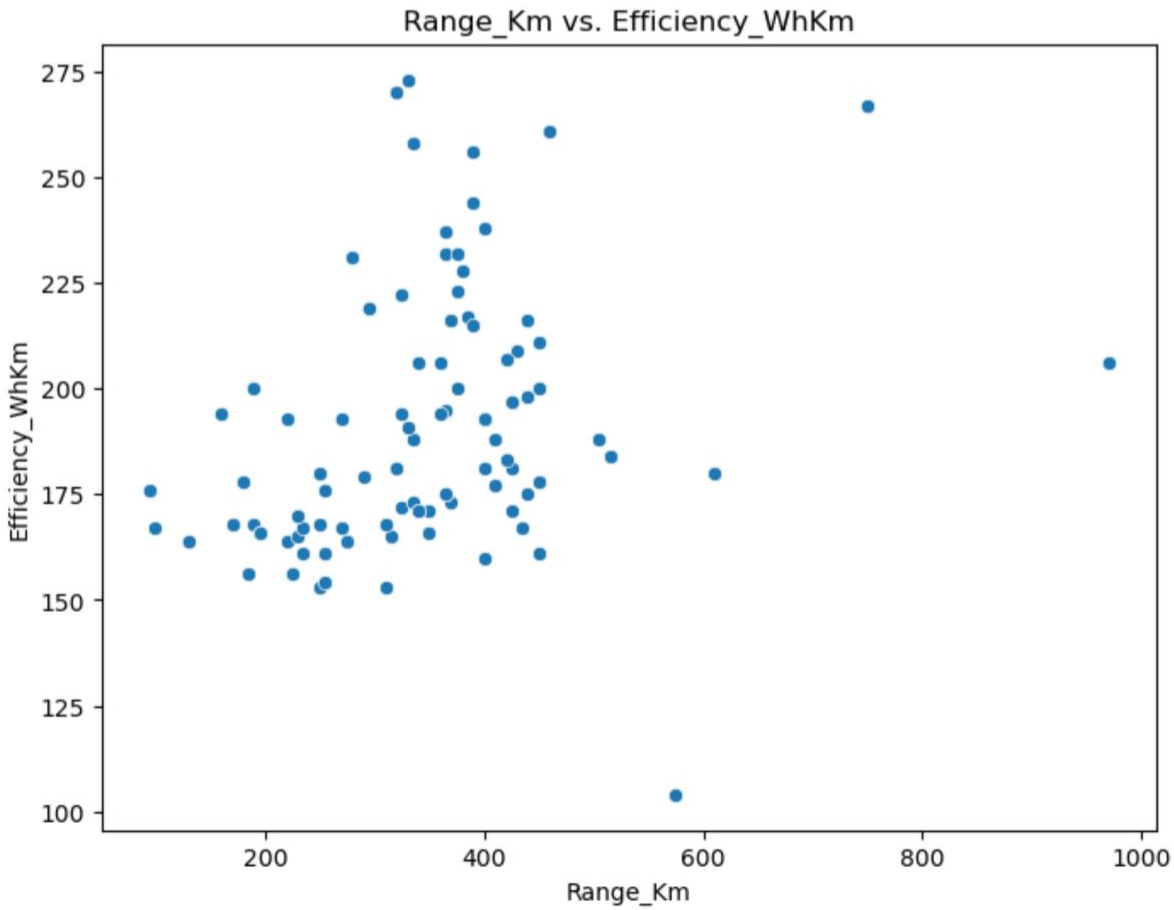
In [12]:
```
#Checking null values
df.isnull().mean()
```

Out[12]:
```
Brand              0.0
Model              0.0
AccelSec           0.0
TopSpeed_KmH       0.0
Range_Km           0.0
Efficiency_WhKm    0.0
FastCharge_KmH     0.0
RapidCharge        0.0
PowerTrain         0.0
PlugType           0.0
BodyStyle          0.0
Segment            0.0
Seats              0.0
PriceEuro          0.0
dtype: float64
```

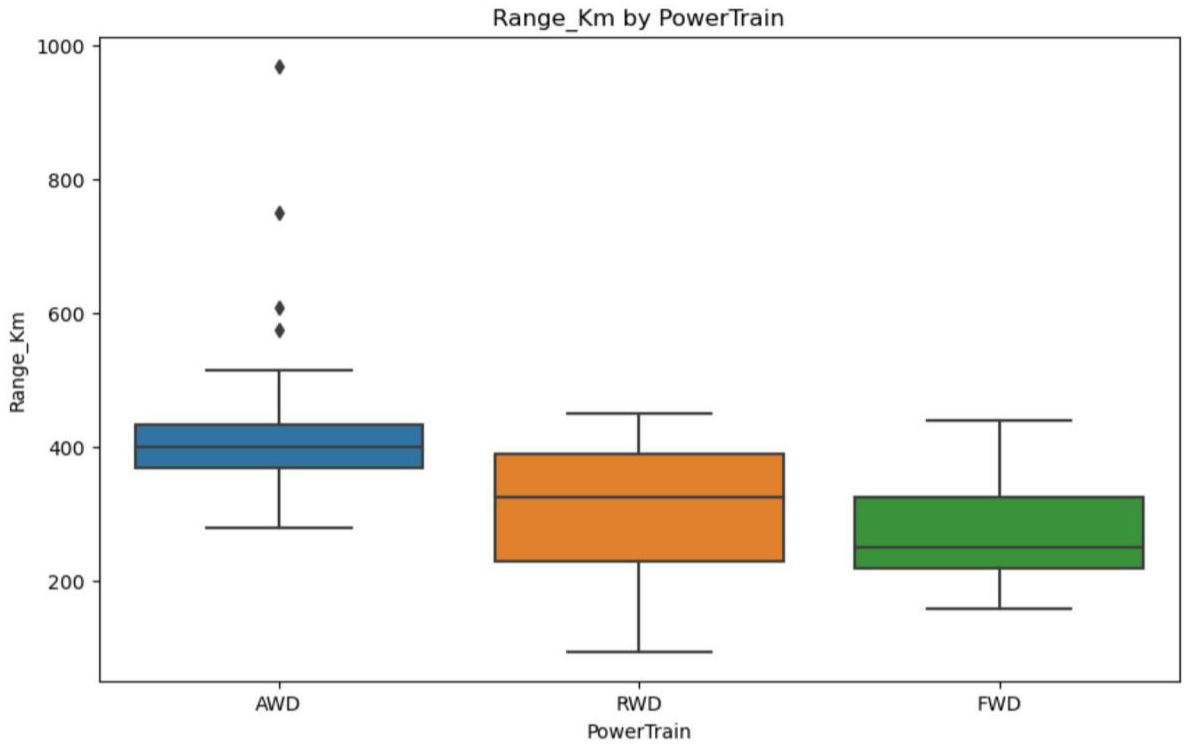
In [13]:
```
plt.figure(figsize=(4, 3))
sns.histplot(df['Range_Km'], kde=True)

plt.title('Distribution of Range_Km')
plt.xlabel('Range_Km')
plt.ylabel('Count')
plt.show()
```



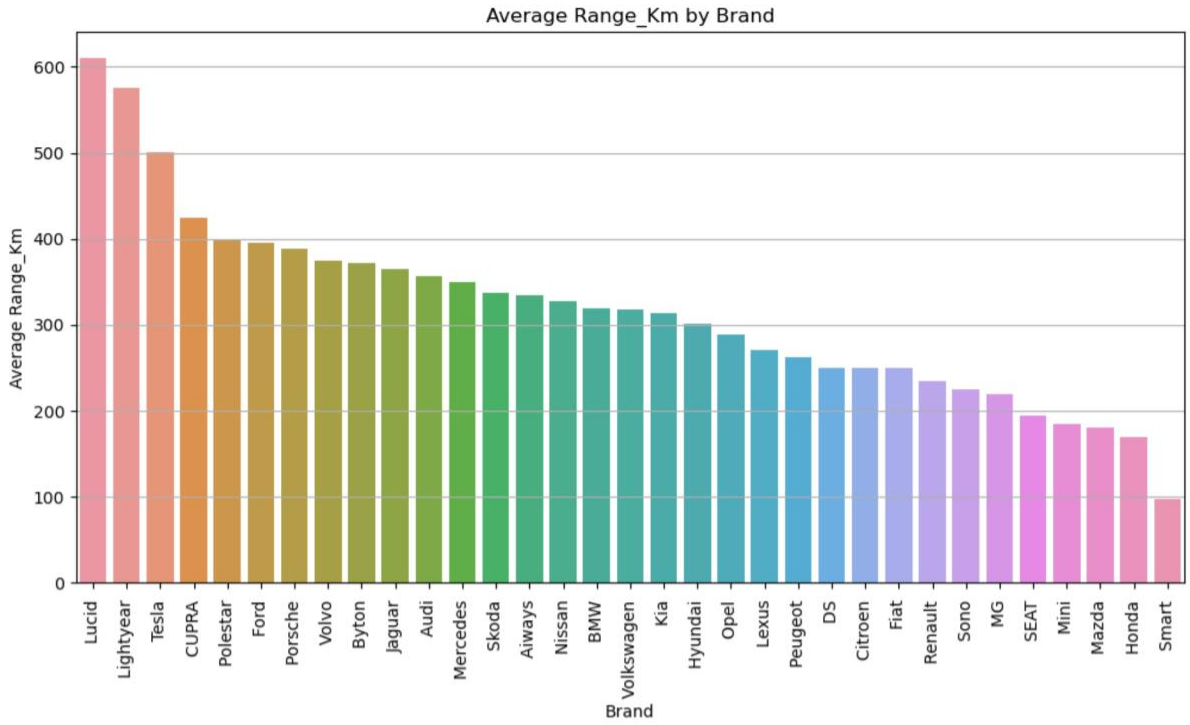
In [11]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='Range_Km', y='Efficiency_WhKm')

plt.title('Range_Km vs. Efficiency_WhKm')
plt.xlabel('Range_Km')
plt.ylabel('Efficiency_WhKm')
plt.show()
```



In [12]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='PowerTrain', y='Range_Km')

plt.title('Range_Km by PowerTrain')
plt.xlabel('PowerTrain')
plt.ylabel('Range_Km')
plt.show()
```

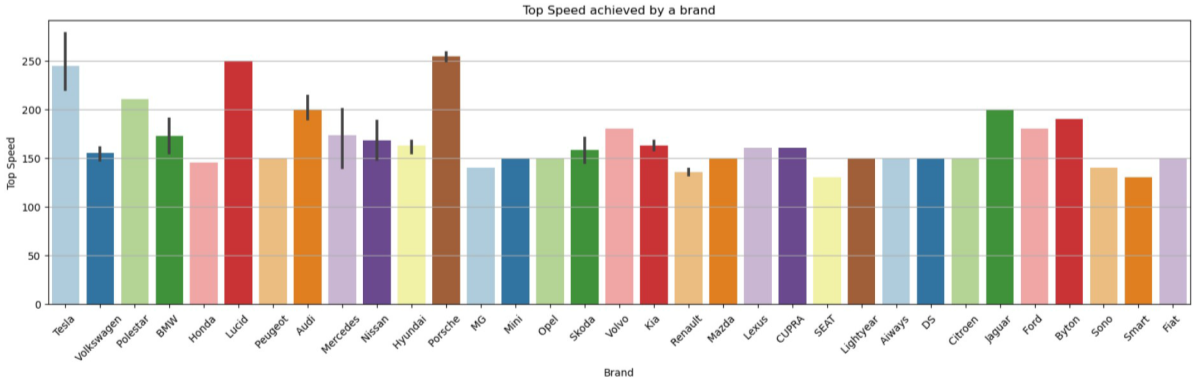


In [13]:
```
plt.figure(figsize=(12, 6))

avg_range_by_brand = df.groupby('Brand')['Range_Km'].mean().sort_values(ascending=F
sns.barplot(x=avg_range_by_brand.index, y=avg_range_by_brand.values)

plt.grid(axis='y')
plt.title('Average Range_Km by Brand')
plt.xlabel('Brand')
plt.ylabel('Average Range_Km')
plt.xticks(rotation=90)
plt.show()
```

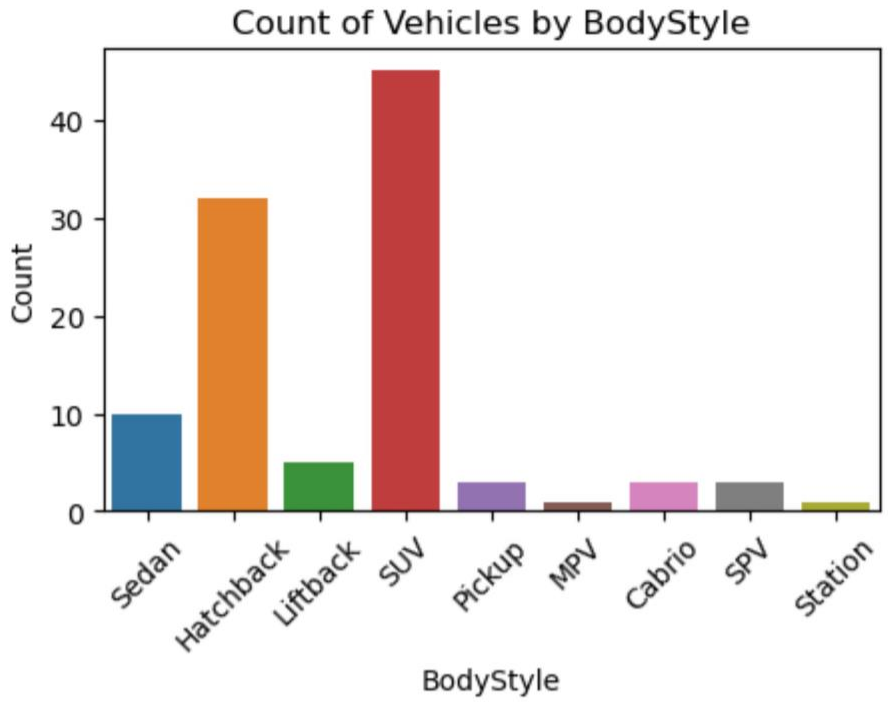


In [14]:
```
ax= plt.figure(figsize=(20,5))
sns.barplot(x='Brand',y='TopSpeed_KmH',data=df,palette='Paired')

plt.grid(axis='y')
plt.title('Top Speed achieved by a brand')
plt.xlabel('Brand')
plt.ylabel('Top Speed')
plt.xticks(rotation=45)
```

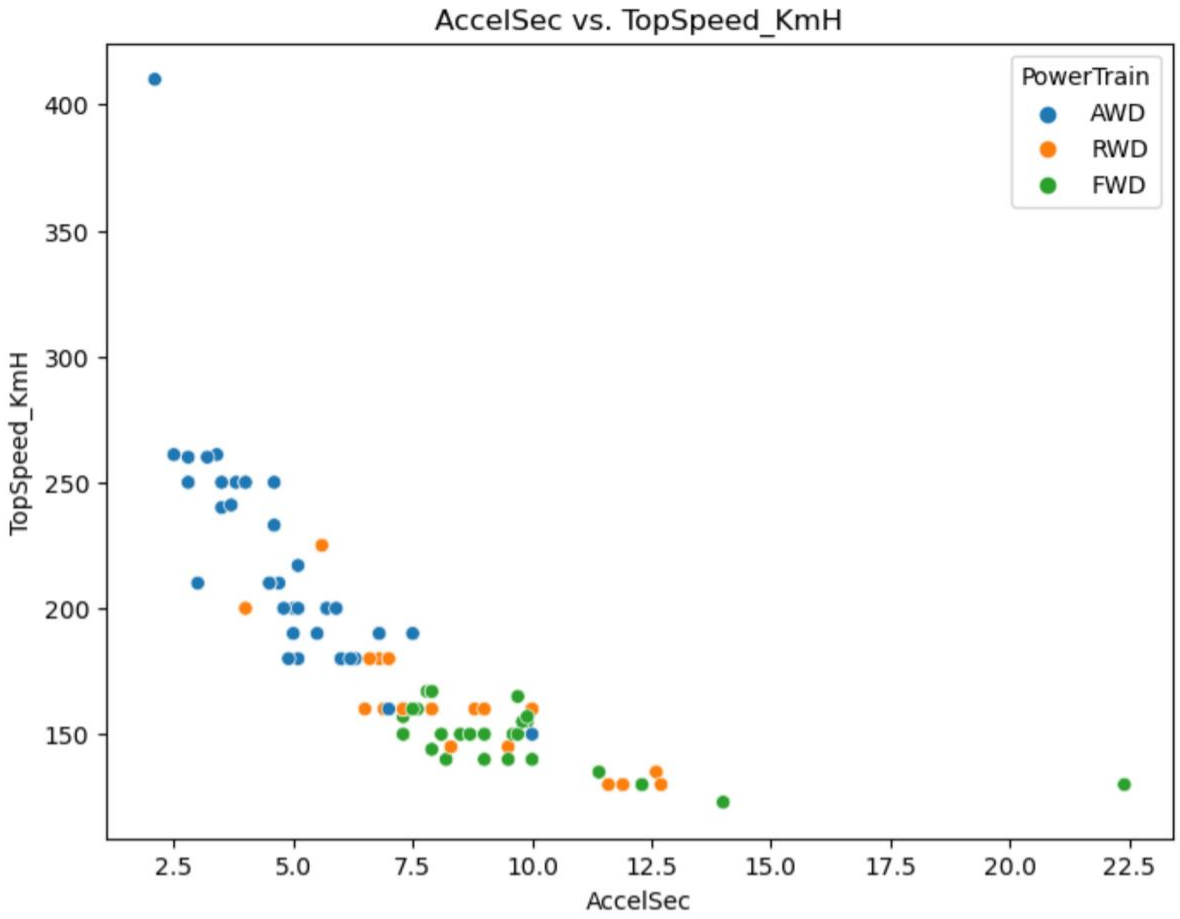
Out[14]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32]),
 [Text(0, 0, 'Tesla '),
  Text(1, 0, 'Volkswagen '),
  Text(2, 0, 'Polestar '),
  Text(3, 0, 'BMW '),
  Text(4, 0, 'Honda '),
  Text(5, 0, 'Lucid '),
  Text(6, 0, 'Peugeot '),
  Text(7, 0, 'Audi '),
  Text(8, 0, 'Mercedes '),
  Text(9, 0, 'Nissan '),
  Text(10, 0, 'Hyundai '),
  Text(11, 0, 'Porsche '),
  Text(12, 0, 'MG '),
  Text(13, 0, 'Mini '),
  Text(14, 0, 'Opel '),
  Text(15, 0, 'Skoda '),
  Text(16, 0, 'Volvo '),
  Text(17, 0, 'Kia '),
  Text(18, 0, 'Renault '),
  Text(19, 0, 'Mazda '),
  Text(20, 0, 'Lexus '),
  Text(21, 0, 'CUPRA '),
  Text(22, 0, 'SEAT '),
  Text(23, 0, 'Lightyear '),
  Text(24, 0, 'Aiways '),
  Text(25, 0, 'DS '),
  Text(26, 0, 'Citroen '),
  Text(27, 0, 'Jaguar '),
  Text(28, 0, 'Ford '),
  Text(29, 0, 'Byton '),
  Text(30, 0, 'Sono '),
  Text(31, 0, 'Smart '),
  Text(32, 0, 'Fiat ')])
```



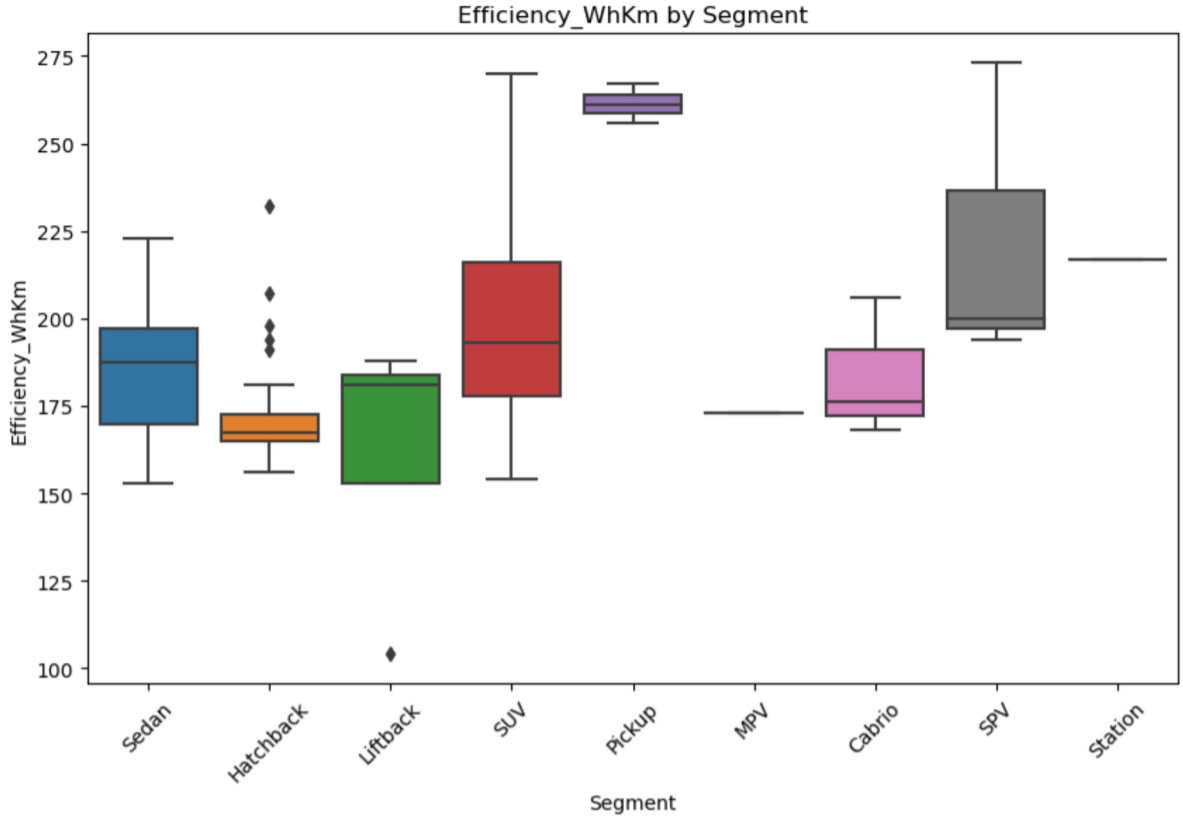
In [15]:
```
plt.figure(figsize=(5, 3))
ax = sns.countplot(data=df, x='BodyStyle')

plt.title('Count of Vehicles by BodyStyle')
plt.xlabel('BodyStyle')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```



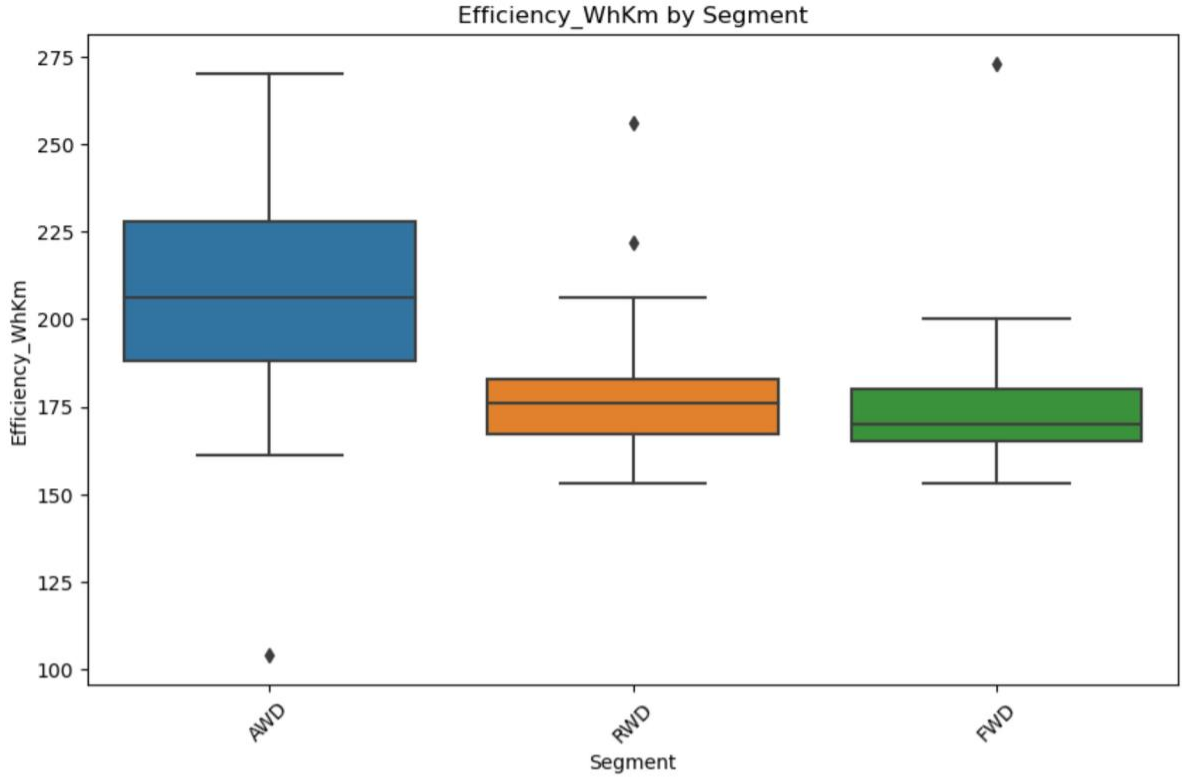
In [16]:
```
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='AccelSec', y='TopSpeed_KmH', hue='PowerTrain')

plt.title('AccelSec vs. TopSpeed_KmH')
plt.xlabel('AccelSec')
plt.ylabel('TopSpeed_KmH')
plt.show()
```



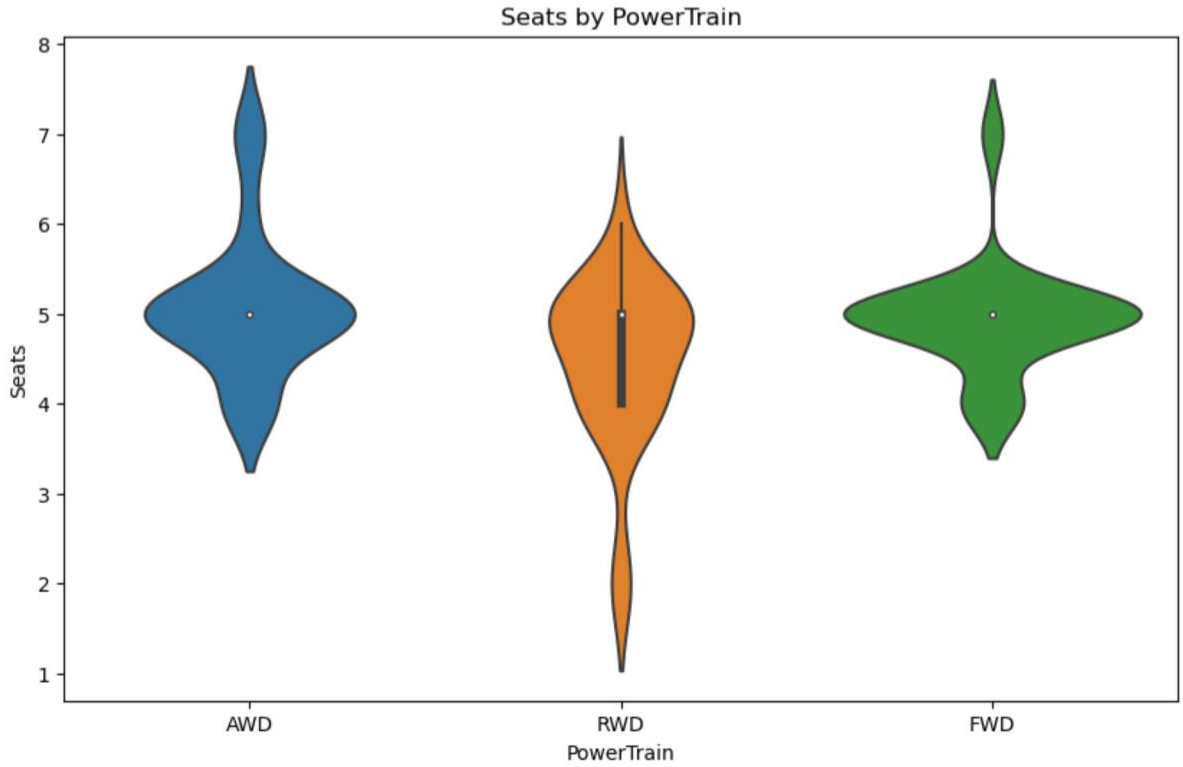
In [17]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='BodyStyle', y='Efficiency_WhKm')

plt.title('Efficiency_WhKm by Segment')
plt.xlabel('Segment')
plt.ylabel('Efficiency_WhKm')
plt.xticks(rotation=45)
plt.show()
```



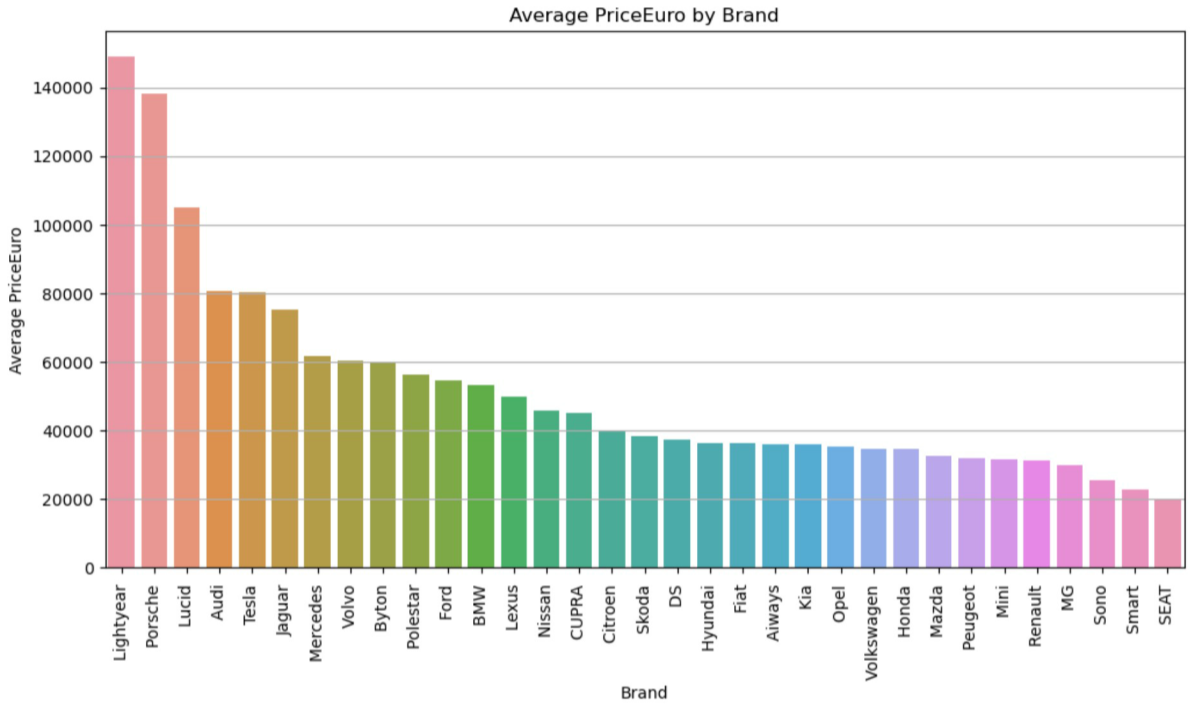
In [18]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='PowerTrain', y='Efficiency_WhKm')

plt.title('Efficiency_WhKm by Segment')
plt.xlabel('Segment')
plt.ylabel('Efficiency_WhKm')
plt.xticks(rotation=45)
plt.show()
```



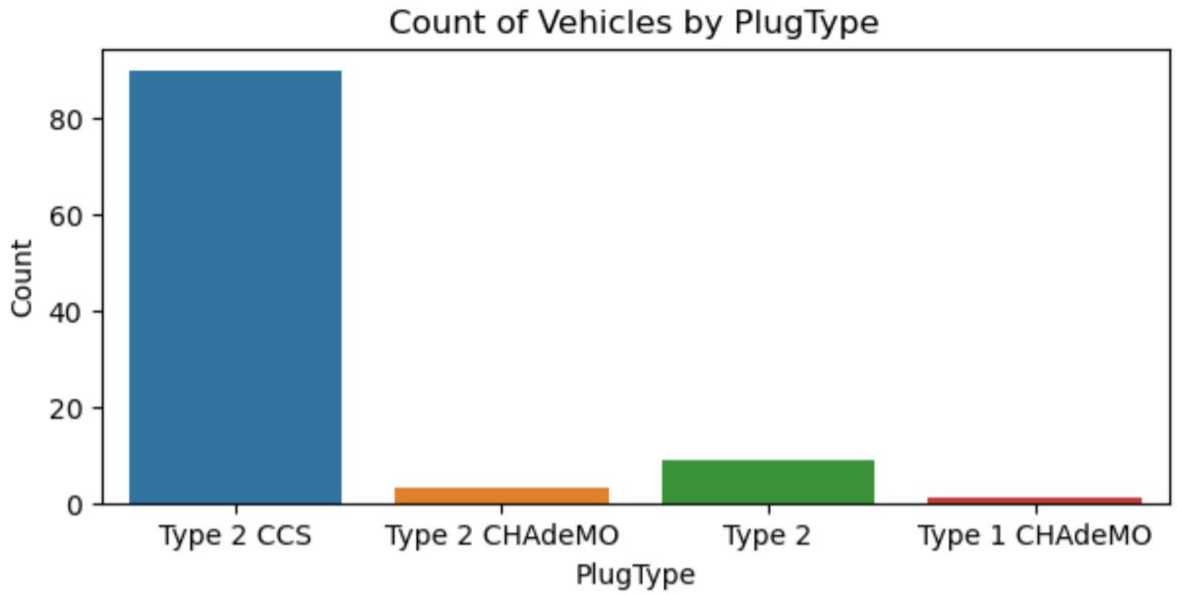
In [19]:
```
plt.figure(figsize=(10, 6))
sns.violinplot(data=df, x='PowerTrain', y='Seats')

plt.title('Seats by PowerTrain')
plt.xlabel('PowerTrain')
plt.ylabel('Seats')
plt.show()
```



In [20]:
```
plt.figure(figsize=(12, 6))

avg_price_by_brand = df.groupby('Brand')['PriceEuro'].mean().sort_values(ascending=
sns.barplot(x=avg_price_by_brand.index, y=avg_price_by_brand.values)

plt.grid(axis='y')
plt.title('Average PriceEuro by Brand')
plt.xlabel('Brand')
plt.ylabel('Average PriceEuro')
plt.xticks(rotation=90)
plt.show()
```



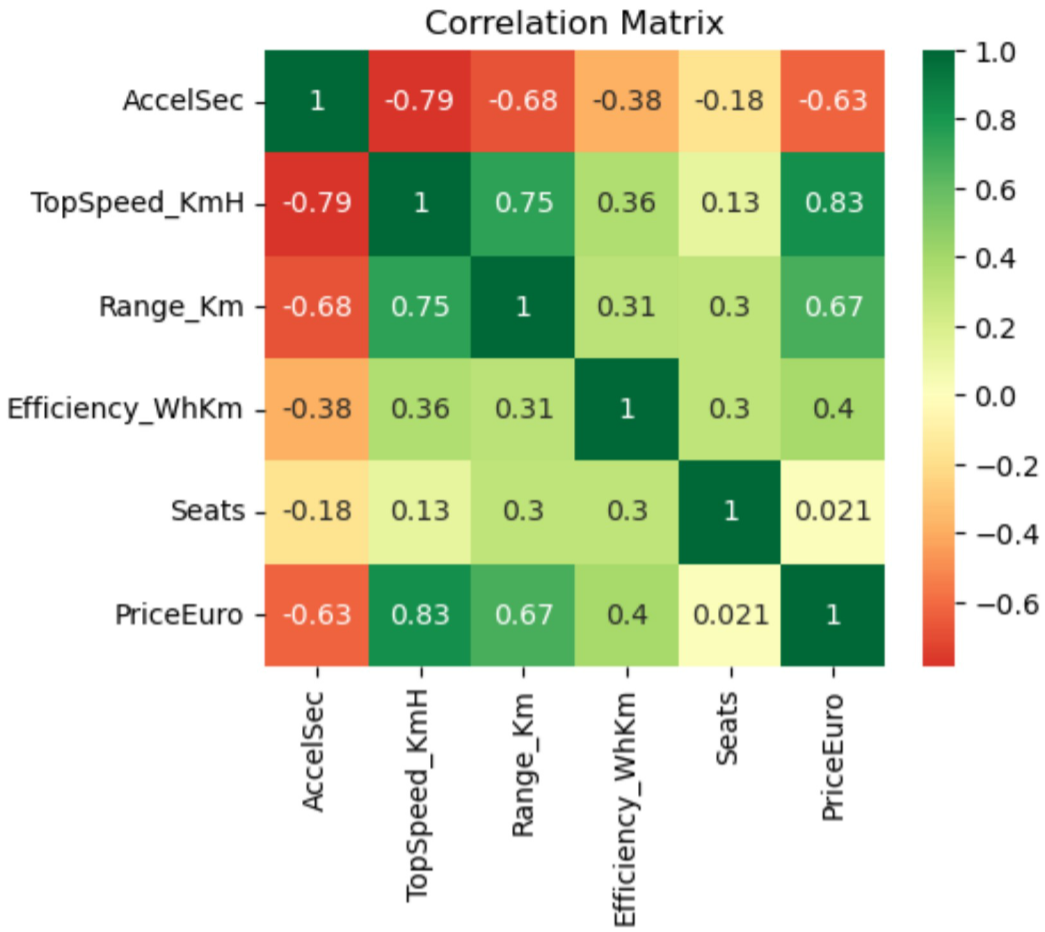
In [21]:
```
plt.figure(figsize=(7,3))
sns.countplot(data=df, x='PlugType')

plt.title('Count of Vehicles by PlugType')
plt.xlabel('PlugType')
plt.ylabel('Count')

plt.show()
```



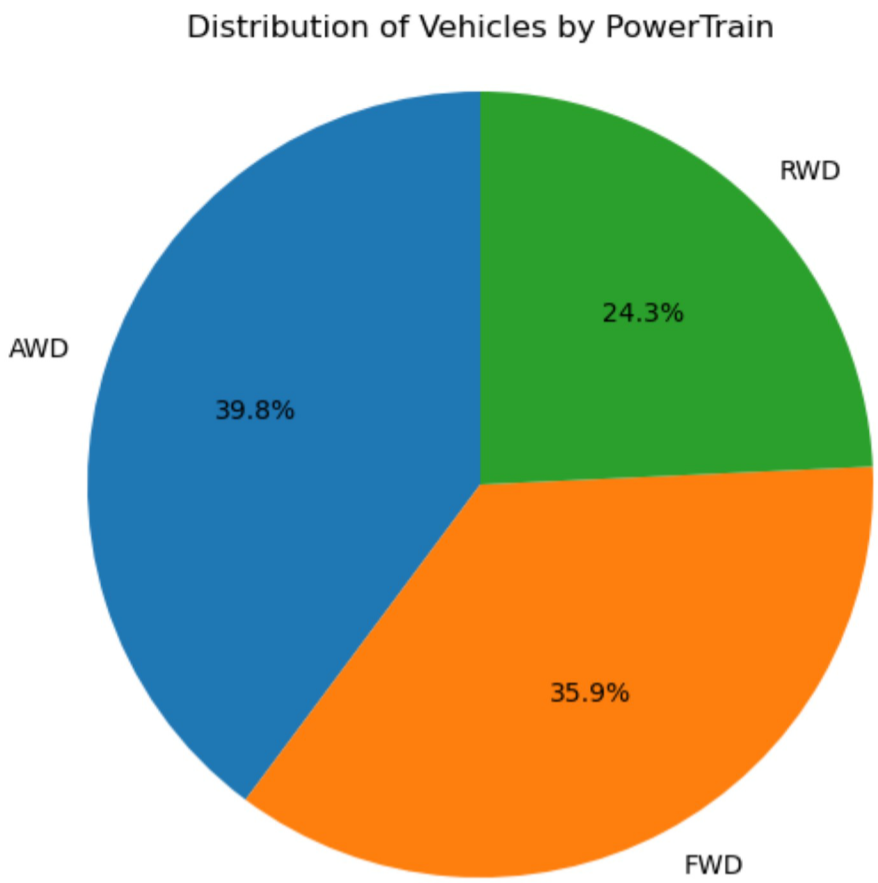
In [22]:
```
correlation_matrix = df.corr()
print(correlation_matrix)
```

```
                AccelSec  TopSpeed_KmH  Range_Km  Efficiency_WhKm     Seats  \
AccelSec         1.000000     -0.786195 -0.677062        -0.382904 -0.175335
TopSpeed_KmH    -0.786195      1.000000  0.746662         0.355675  0.126470
Range_Km        -0.677062      0.746662  1.000000         0.313077  0.300163
Efficiency_WhKm -0.382904      0.355675  0.313077         1.000000  0.301230
Seats           -0.175335      0.126470  0.300163         0.301230  1.000000
PriceEuro       -0.627174      0.829057  0.674844         0.396705  0.020920

                PriceEuro
AccelSec         -0.627174
TopSpeed_KmH      0.829057
Range_Km          0.674844
Efficiency_WhKm   0.396705
Seats             0.020920
PriceEuro         1.000000
C:\Users\viddy\AppData\Local\Temp\ipykernel_4008\2020863294.py:1: FutureWarning: Th
e default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  correlation_matrix = df.corr()
```

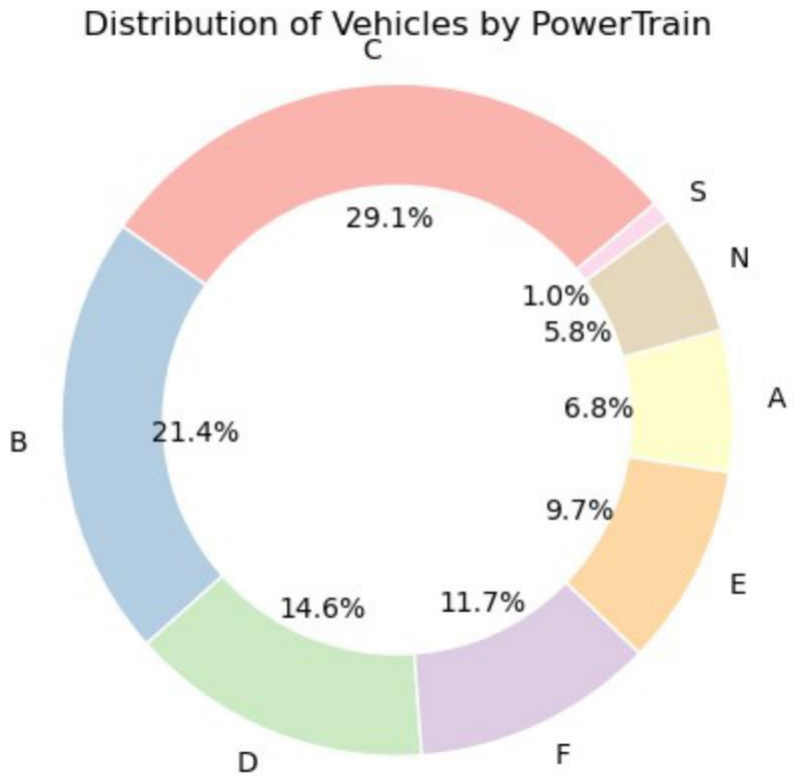
In [23]:
```
plt.figure(figsize=(5, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='RdYlGn', center=0)
plt.title('Correlation Matrix')
plt.show()
```



In [26]:
```
powertrain_counts = df['PowerTrain'].value_counts()

# Create a pie chart
plt.figure(figsize=(8, 6))
plt.pie(powertrain_counts, labels=powertrain_counts.index, autopct='%1.1f%%', start
plt.title('Distribution of Vehicles by PowerTrain')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



In [31]:
```
outer_colors = sns.color_palette('Pastel1')
plt.pie(segment_counts, labels=segment_counts.index, autopct='%1.1f%%', startangle=
        colors=outer_colors, wedgeprops={'width': 0.5, 'edgecolor': 'white'})

# Inner ring (hole in the donut)
inner_radius = 0.7
plt.pie([1], radius=inner_radius, colors='white')

plt.title('Distribution of Vehicles by PowerTrain')
plt.axis('equal')
plt.show()
```



Observations :

- Distribution of Range_km histogram Data follows Positive distribution. It shows most of the ev cars having 200 to 450 km range.
- Range_Km vs Efficiency_WhKm scatterplot shows that As "Range is increases Efficiency also increases".
- Lucid,Lightyear and Tesla brands has "Average Range" above 500 Km, Smart brand has avg range below 100 km.
- Tesla, Lucid and Porsche brand's cars have "Top speed" 250 KmH and above.
- Lightyear and Prorsche "car's price" is above 120000 Euro.
- 80% EV has "Type 2 CCS PlugType".
- In our dataset, we have most of the cars from "SUV" and "Hatchback" segment and we have 40% AWD(All Wheel Drive),36% FWD(Front Wheel Drive) and 24% RWD(Rear Wheel Drive).

GitHub links

Kanna Lokesh	https://github.com/kannalokesh13/Feynn_Labs_Internship_2023/blob/main/Electric_vehicles_in_india.docx
Siddhesh Salfale	https://github.com/Siddhesh6344/Feynn-Labs
Neha Dwivedi	https://github.com/NehaDwivedi842/Feynnlab_internship
Kaveti Venkata Nikhil	https://github.com/Feynnlabs/project-2.2.
Amrit Kumar Maurya	https://github.com/Amrit-2002/EV_Market_Analysis
Viddyesh Dhobale	https://github.com/viddyesh/EV-Data-Analysis-Python