Video Explanation

Link: https://drive.google.com/file/d/1W6RGrFA-_7g6kdzGtRL-MXs_kz2DMY5K/view?usp=sharing

Task 1: Cleaning the Data

In [7]:
```
# importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [9]:
```
# loading the data
data = pd.read_csv("ev.csv")
data.head()
```

Out[9]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Number of seats	Num do
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	438	...	3130.0	640.0	5	
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	340	...	3040.0	670.0	5	
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	364	...	3130.0	565.0	5	
3	Audi e-tron Sportback 50 quattro	Audi	e-tron Sportback 50 quattro	319700	313	540	disc (front + rear)	4WD	71.0	346	...	3040.0	640.0	5	
4	Audi e-tron Sportback 55 quattro	Audi	e-tron Sportback 55 quattro	357000	360	664	disc (front + rear)	4WD	95.0	447	...	3130.0	670.0	5	

5 rows × 25 columns

In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53 entries, 0 to 52
Data columns (total 25 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Car full name                         53 non-null     object 
 1   Make                                  53 non-null     object 
 2   Model                                 53 non-null     object 
 3   Minimal price (gross) [PLN]           53 non-null     int64  
 4   Engine power [KM]                     53 non-null     int64  
 5   Maximum torque [Nm]                   53 non-null     int64  
 6   Type of brakes                        52 non-null     object 
 7   Drive type                            53 non-null     object 
 8   Battery capacity [kWh]                53 non-null     float64
 9   Range (WLTP) [km]                     53 non-null     int64  
 10  Wheelbase [cm]                        53 non-null     float64
 11  Length [cm]                           53 non-null     float64
 12  Width [cm]                            53 non-null     float64
 13  Height [cm]                           53 non-null     float64
 14  Minimal empty weight [kg]             53 non-null     int64  
 15  Permissable gross weight [kg]         45 non-null     float64
 16  Maximum load capacity [kg]            45 non-null     float64
 17  Number of seats                       53 non-null     int64  
 18  Number of doors                       53 non-null     int64  
 19  Tire size [in]                        53 non-null     int64  
 20  Maximum speed [kph]                   53 non-null     int64  
 21  Boot capacity (VDA) [l]               52 non-null     float64
 22  Acceleration 0-100 kph [s]            50 non-null     float64
 23  Maximum DC charging power [kW]        53 non-null     int64  
 24  mean - Energy consumption [kWh/100 km]  44 non-null     float64
dtypes: float64(10), int64(10), object(5)
memory usage: 10.5+ KB
```

In [13]:
```
data.isnull().sum() #for checking the missing values.
```

Out[13]:
```
Car full name                            0
Make                                     0
Model                                    0
Minimal price (gross) [PLN]              0
Engine power [KM]                        0
Maximum torque [Nm]                      0
Type of brakes                           1
Drive type                               0
Battery capacity [kWh]                   0
Range (WLTP) [km]                        0
Wheelbase [cm]                           0
Length [cm]                              0
Width [cm]                               0
Height [cm]                              0
Minimal empty weight [kg]                0
Permissable gross weight [kg]            8
Maximum load capacity [kg]               8
Number of seats                          0
Number of doors                          0
Tire size [in]                           0
Maximum speed [kph]                      0
Boot capacity (VDA) [l]                  1
Acceleration 0-100 kph [s]               3
Maximum DC charging power [kW]           0
mean - Energy consumption [kWh/100 km]   9
dtype: int64
```

In [15]:
```
data['Type of brakes'].value_counts() #to find mode. To replace missing value.
```

Out[15]:
```
Type of brakes
disc (front + rear)          45
disc (front) + drum (rear)    7
Name: count, dtype: int64
```

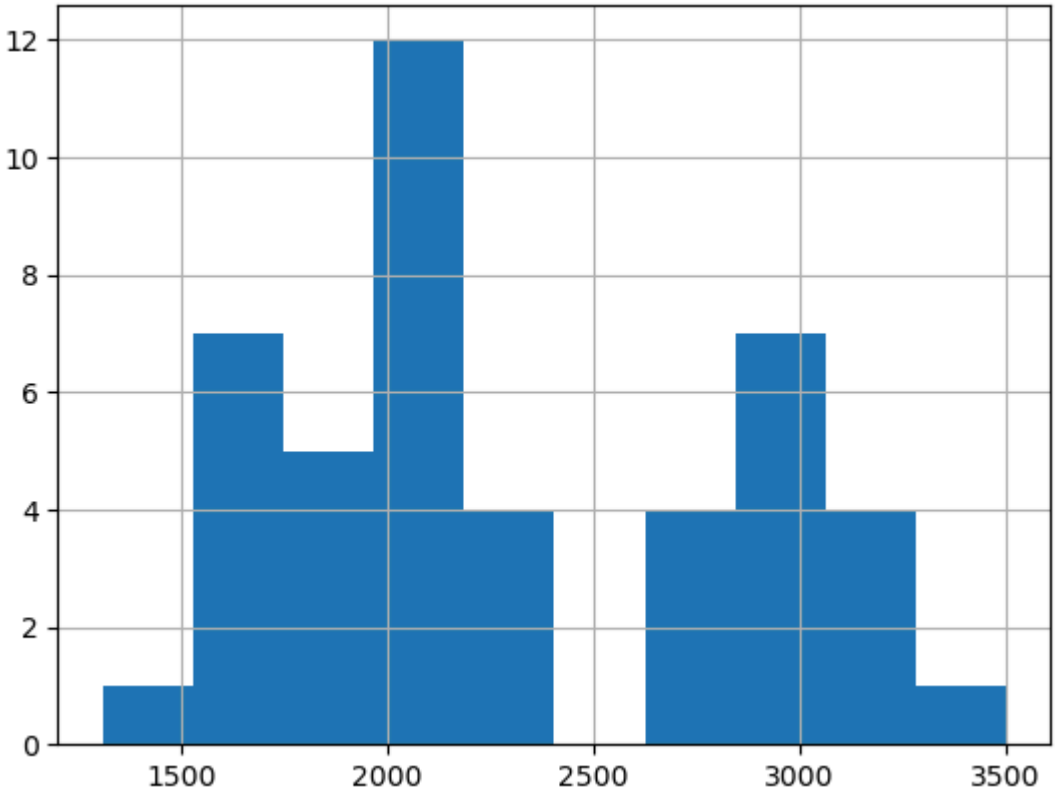
In [17]:
```
data['Type of brakes'].fillna(data['Type of brakes'].mode()[0],inplace=True) #replacing missing value with modal 'disc (front
data['Type of brakes'].value_counts()
```

```
C:\Windows\Temp\ipykernel_2860\3783079006.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series t
hrough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  data['Type of brakes'].fillna(data['Type of brakes'].mode()[0],inplace=True) #replacing missing value with modal 'disc (front
+ rear)'
```

Out[17]:
```
Type of brakes
disc (front + rear)          46
disc (front) + drum (rear)    7
Name: count, dtype: int64
```

In [19]:
```
# to check distribution of values in continuous type of variables - we plot histograms.
data['Permissable gross weight [kg]'].hist(bins = 10)
```

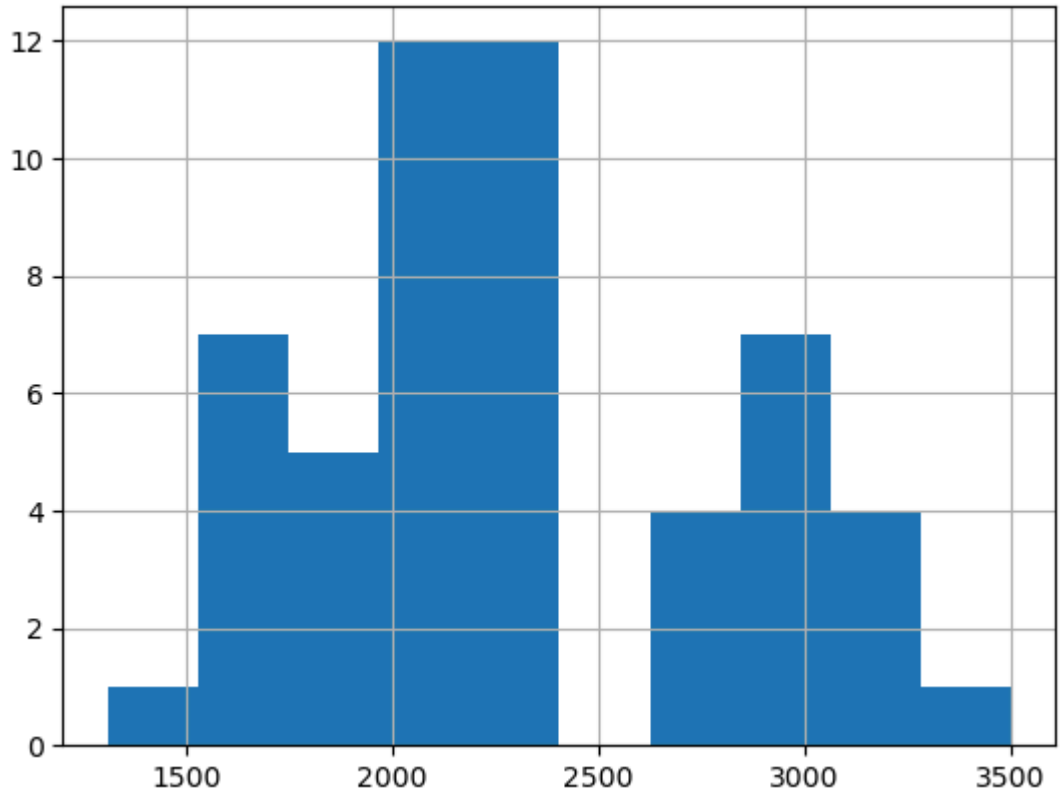
Out[19]:
```
<Axes: >
```



In [21]:
```
# from the distribution of 'Permissable gross weight [kg]' we can replace the missing values by the mean values
data['Permissable gross weight [kg]'].fillna(data['Permissable gross weight [kg]'].mean(),inplace=True)
data['Permissable gross weight [kg]'].hist(bins = 10)
```

```
C:\Windows\Temp\ipykernel_2860\2254656642.py:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series t
hrough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  data['Permissable gross weight [kg]'].fillna(data['Permissable gross weight [kg]'].mean(),inplace=True)
```

Out[21]: <Axes: >

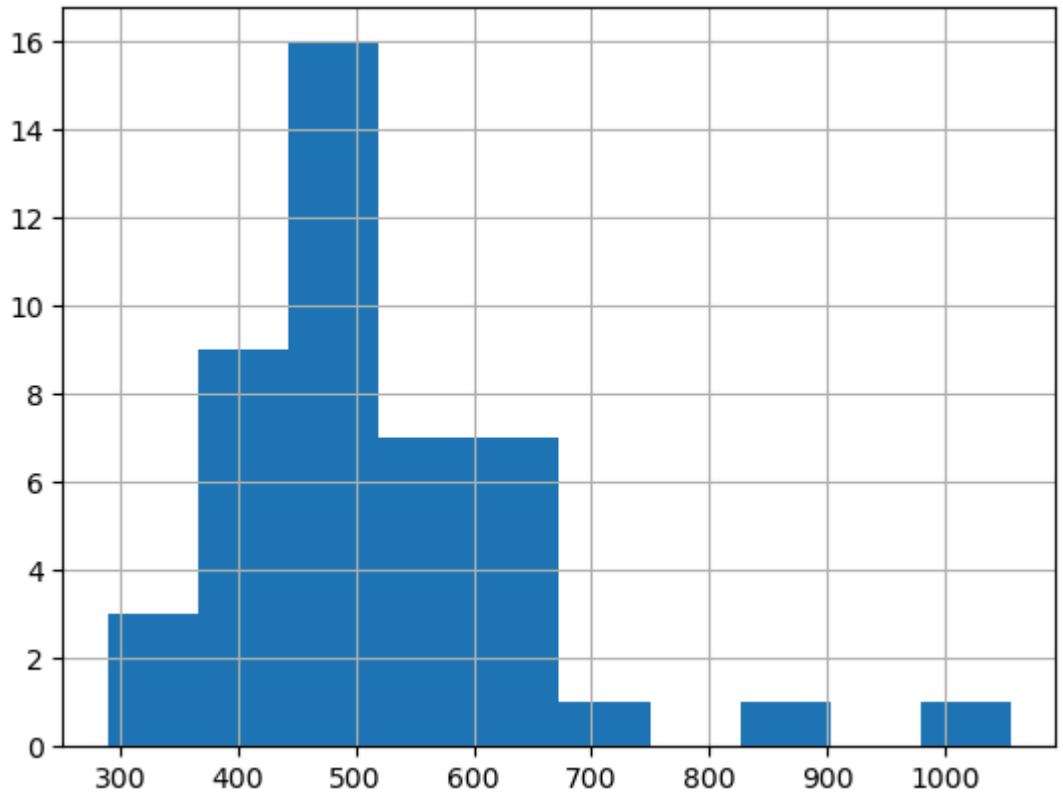


In [23]:
```
#to check remaining missing value columns
data.isnull().sum()
```

Out[23]:
```
Car full name                          0
Make                                   0
Model                                  0
Minimal price (gross) [PLN]            0
Engine power [KM]                      0
Maximum torque [Nm]                    0
Type of brakes                         0
Drive type                             0
Battery capacity [kWh]                 0
Range (WLTP) [km]                      0
Wheelbase [cm]                         0
Length [cm]                            0
Width [cm]                             0
Height [cm]                            0
Minimal empty weight [kg]              0
Permissable gross weight [kg]          0
Maximum load capacity [kg]             8
Number of seats                        0
Number of doors                        0
Tire size [in]                         0
Maximum speed [kph]                    0
Boot capacity (VDA) [l]                1
Acceleration 0-100 kph [s]             3
Maximum DC charging power [kW]         0
mean - Energy consumption [kWh/100 km]    9
dtype: int64
```

In [25]:
```
data['Maximum load capacity [kg]'].hist(bins = 10)
```

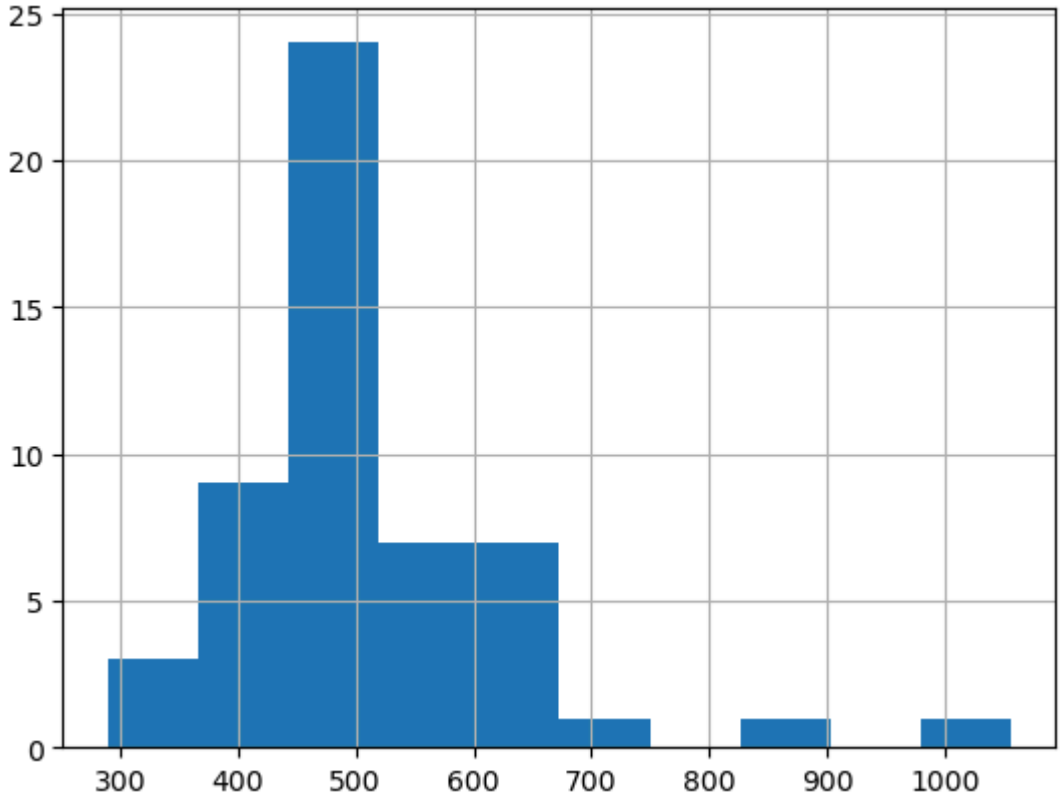
Out[25]:
```
<Axes: >
```



In [27]:
```
data['Maximum load capacity [kg]'].fillna(data['Maximum load capacity [kg]'].median(),inplace=True)
data['Maximum load capacity [kg]'].hist(bins = 10)
```

```
C:\Windows\Temp\ipykernel_2860\998857968.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series th
rough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  data['Maximum load capacity [kg]'].fillna(data['Maximum load capacity [kg]'].median(),inplace=True)
```

Out[27]: <Axes: >

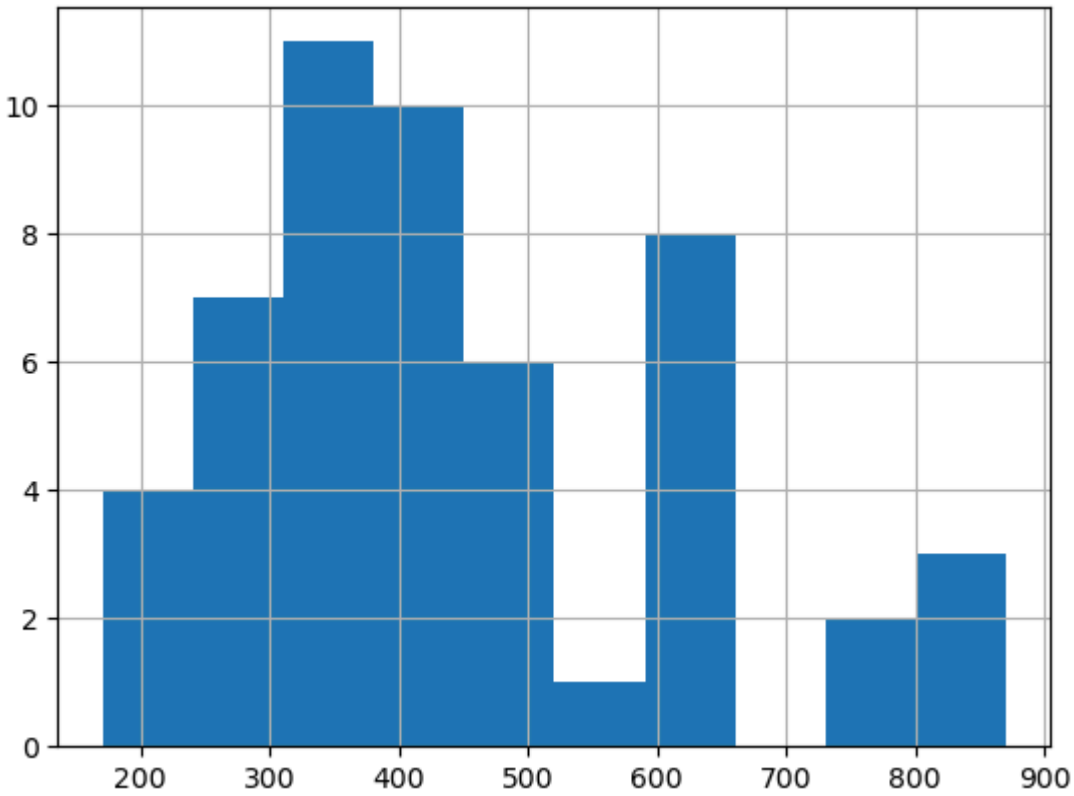


In [29]:
```
data.isnull().sum()
```

Out[29]:
```
Car full name                          0
Make                                   0
Model                                  0
Minimal price (gross) [PLN]            0
Engine power [KM]                      0
Maximum torque [Nm]                    0
Type of brakes                         0
Drive type                             0
Battery capacity [kWh]                 0
Range (WLTP) [km]                      0
Wheelbase [cm]                         0
Length [cm]                            0
Width [cm]                             0
Height [cm]                            0
Minimal empty weight [kg]              0
Permissable gross weight [kg]          0
Maximum load capacity [kg]             0
Number of seats                        0
Number of doors                        0
Tire size [in]                         0
Maximum speed [kph]                    0
Boot capacity (VDA) [l]                1
Acceleration 0-100 kph [s]             3
Maximum DC charging power [kW]         0
mean - Energy consumption [kWh/100 km]    9
dtype: int64
```

In [31]:
```
data['Boot capacity (VDA) [l]'].hist(bins = 10)
```

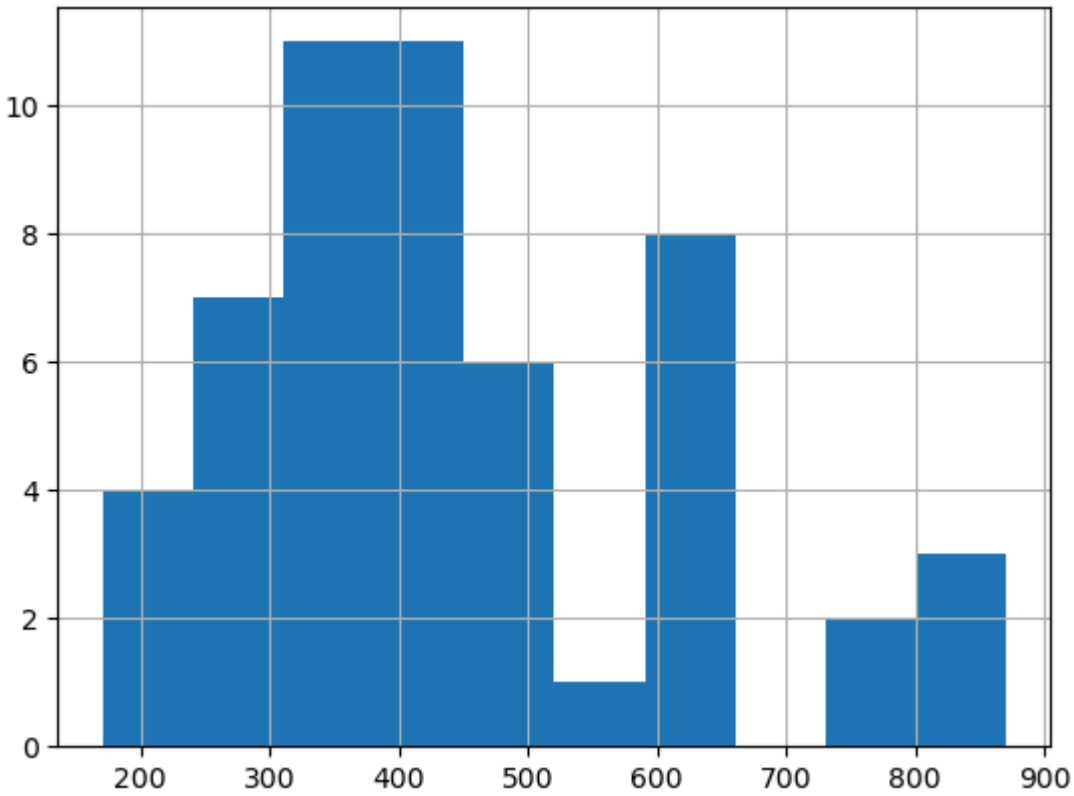
Out[31]:
```
<Axes: >
```



In [33]:
```
data['Boot capacity (VDA) [l]'].fillna(data['Boot capacity (VDA) [l]'].mean(),inplace = True)
data['Boot capacity (VDA) [l]'].hist(bins = 10)
```

```
C:\Windows\Temp\ipykernel_2860\2052638756.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series t
hrough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  data['Boot capacity (VDA) [l]'].fillna(data['Boot capacity (VDA) [l]'].mean(),inplace = True)
```

Out[33]: <Axes: >

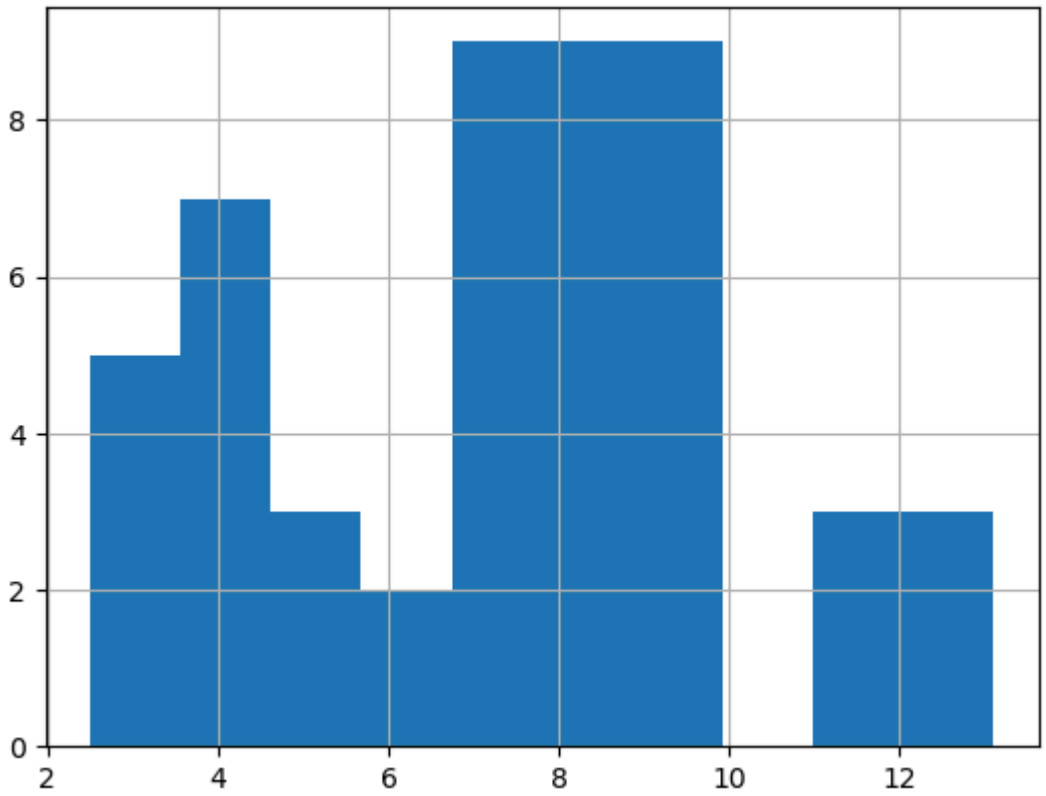


In [35]:
```
data.isnull().sum()
```

Out[35]:
```
Car full name                          0
Make                                   0
Model                                  0
Minimal price (gross) [PLN]            0
Engine power [KM]                      0
Maximum torque [Nm]                    0
Type of brakes                         0
Drive type                             0
Battery capacity [kWh]                 0
Range (WLTP) [km]                      0
Wheelbase [cm]                         0
Length [cm]                            0
Width [cm]                             0
Height [cm]                            0
Minimal empty weight [kg]              0
Permissable gross weight [kg]          0
Maximum load capacity [kg]             0
Number of seats                        0
Number of doors                        0
Tire size [in]                         0
Maximum speed [kph]                    0
Boot capacity (VDA) [l]                0
Acceleration 0-100 kph [s]             3
Maximum DC charging power [kW]         0
mean - Energy consumption [kWh/100 km]    9
dtype: int64
```

In [37]:
```
data['Acceleration 0-100 kph [s]'].hist(bins = 10)
```

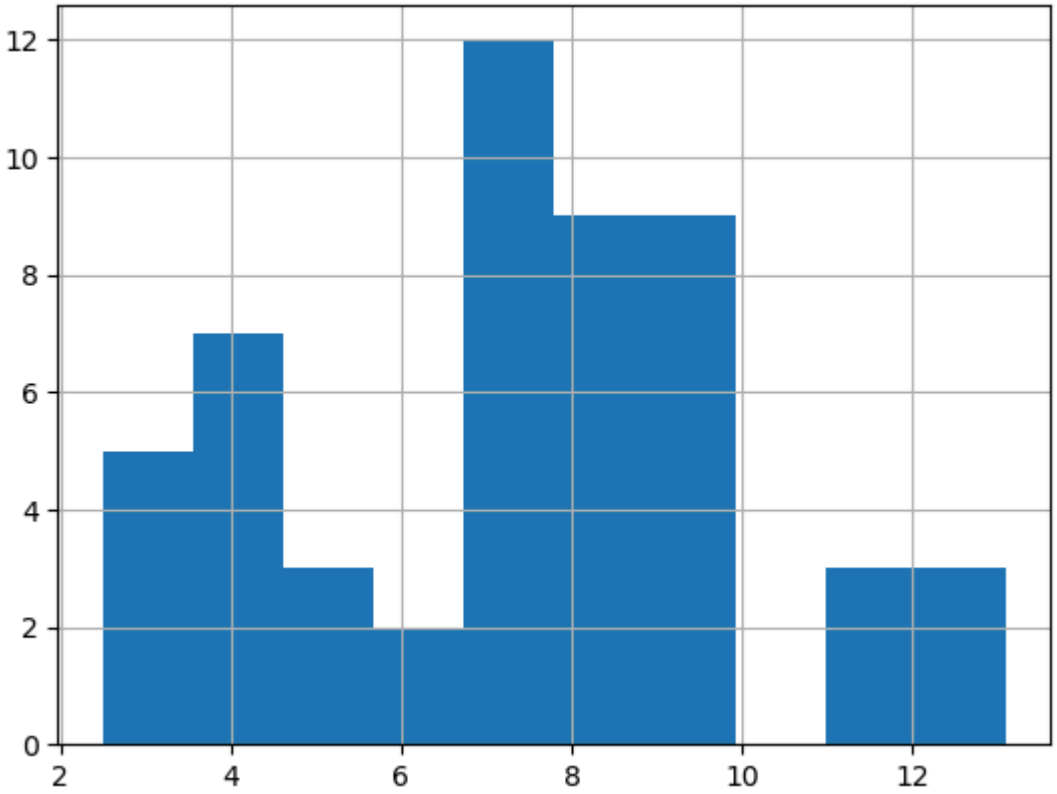
Out[37]:
```
<Axes: >
```



In [39]:
```
data['Acceleration 0-100 kph [s]'].fillna(data['Acceleration 0-100 kph [s]'].mean(),inplace = True)
data['Acceleration 0-100 kph [s]'].hist(bins = 10)
```

```
C:\Windows\Temp\ipykernel_2860\2368333599.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series t
hrough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  data['Acceleration 0-100 kph [s]'].fillna(data['Acceleration 0-100 kph [s]'].mean(),inplace = True)
```

Out[39]: <Axes: >

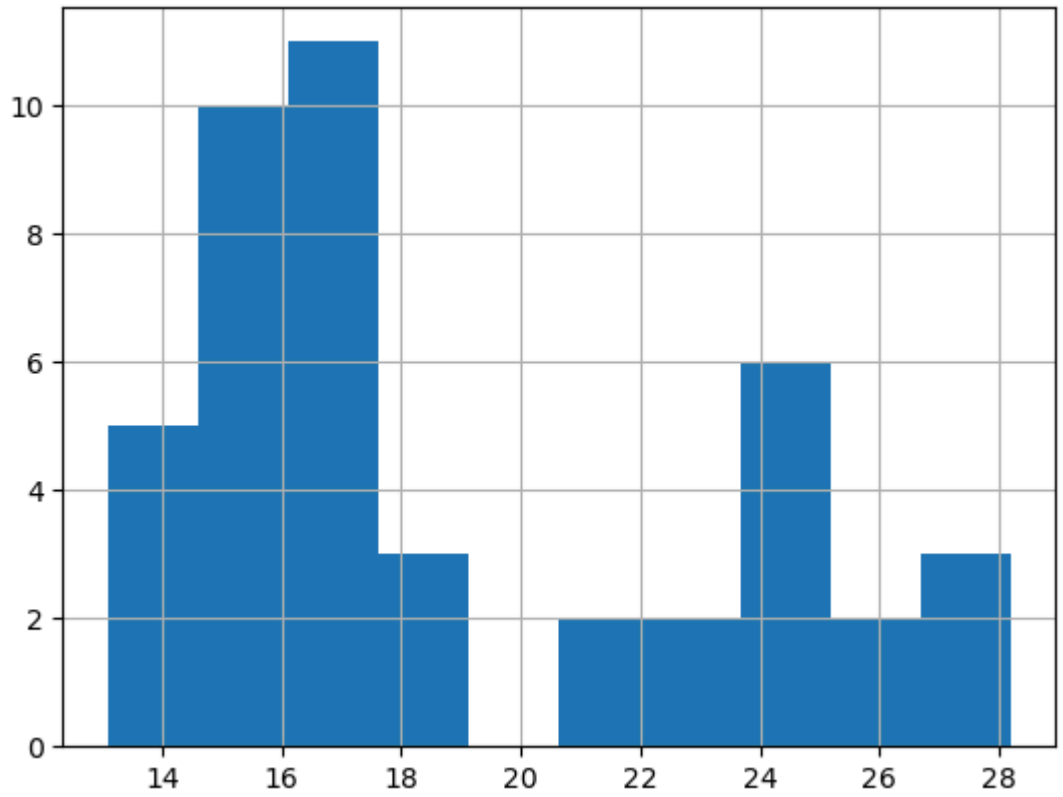


In [41]:
```
data.isnull().sum()
```

Out[41]:
```
Car full name                            0
Make                                     0
Model                                    0
Minimal price (gross) [PLN]              0
Engine power [KM]                        0
Maximum torque [Nm]                      0
Type of brakes                           0
Drive type                               0
Battery capacity [kWh]                   0
Range (WLTP) [km]                        0
Wheelbase [cm]                           0
Length [cm]                              0
Width [cm]                               0
Height [cm]                              0
Minimal empty weight [kg]                0
Permissable gross weight [kg]            0
Maximum load capacity [kg]               0
Number of seats                          0
Number of doors                          0
Tire size [in]                           0
Maximum speed [kph]                      0
Boot capacity (VDA) [l]                  0
Acceleration 0-100 kph [s]               0
Maximum DC charging power [kW]           0
mean - Energy consumption [kWh/100 km]   9
dtype: int64
```

In [43]:
```
data['mean - Energy consumption [kWh/100 km]'].hist(bins = 10)
```

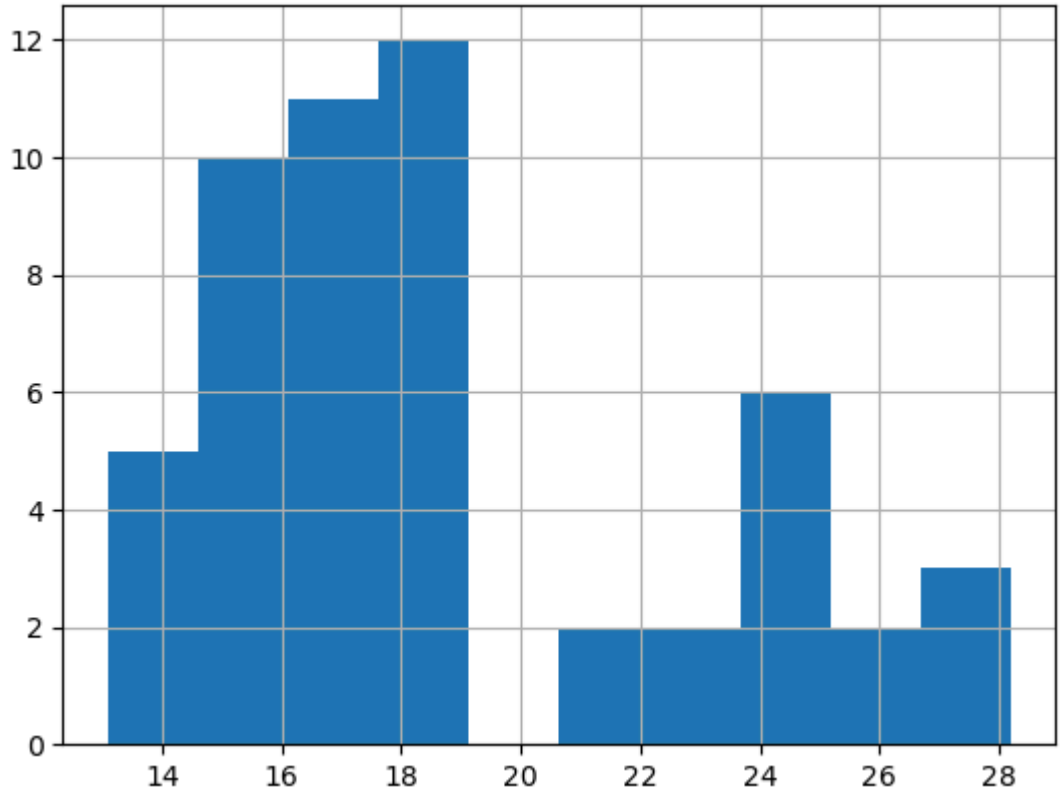
Out[43]:
```
<Axes: >
```



In [45]:
```
data['mean - Energy consumption [kWh/100 km]'].fillna(data['mean - Energy consumption [kWh/100 km]'].mean(),inplace = True)
data['mean - Energy consumption [kWh/100 km]'].hist(bins = 10)
```

```
C:\Windows\Temp\ipykernel_2860\958342638.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series th
rough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  data['mean - Energy consumption [kWh/100 km]'].fillna(data['mean - Energy consumption [kWh/100 km]'].mean(),inplace = True)
```

Out[45]: <Axes: >



In [47]:
```
data.isnull().sum()
```

Out[47]:
```
Car full name                            0
Make                                     0
Model                                    0
Minimal price (gross) [PLN]              0
Engine power [KM]                        0
Maximum torque [Nm]                      0
Type of brakes                           0
Drive type                               0
Battery capacity [kWh]                   0
Range (WLTP) [km]                        0
Wheelbase [cm]                           0
Length [cm]                              0
Width [cm]                               0
Height [cm]                              0
Minimal empty weight [kg]                0
Permissable gross weight [kg]            0
Maximum load capacity [kg]               0
Number of seats                          0
Number of doors                          0
Tire size [in]                           0
Maximum speed [kph]                      0
Boot capacity (VDA) [l]                  0
Acceleration 0-100 kph [s]               0
Maximum DC charging power [kW]           0
mean - Energy consumption [kWh/100 km]   0
dtype: int64
```

Now we have checked for the errors, missing values & outliers. Replaced the missing ones. Our data is clean for further processing.

Task 2: EV Growth

In [49]:
```
data.head()
```

Out[49]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Number of seats	Num do
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	438	...	3130.0	640.0	5	
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	340	...	3040.0	670.0	5	
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	364	...	3130.0	565.0	5	
3	Audi e-tron Sportback 50 quattro	Audi	e-tron Sportback 50 quattro	319700	313	540	disc (front + rear)	4WD	71.0	346	...	3040.0	640.0	5	
4	Audi e-tron Sportback 55 quattro	Audi	e-tron Sportback 55 quattro	357000	360	664	disc (front + rear)	4WD	95.0	447	...	3130.0	670.0	5	

5 rows × 25 columns

In [51]:
```
data['Make'].value_counts()
```

Out[51]:
```
Make
Tesla            7
Audi             6
Kia              4
Porsche          4
Volkswagen       4
Hyundai          3
BMW              3
Nissan           3
Honda            2
Mercedes-Benz    2
Opel             2
Peugeot          2
Renault          2
Smart            2
Citroën          2
Jaguar           1
Mazda            1
DS               1
Skoda            1
Mini             1
Name: count, dtype: int64
```

In [53]:
```
data['Make'].value_counts().sum()
```

Out[53]:
```
53
```

In [55]:
```
data.describe() #Overview of the data
```

Out[55]:

	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Battery capacity [kWh]	Range (WLTP) [km]	Wheelbase [cm]	Length [cm]	Width [cm]	Height [cm]	Minimal empty weight [kg]	Perm wei
count	53.000000	53.000000	53.000000	53.000000	53.000000	53.000000	53.000000	53.000000	53.000000	53.000000	53
mean	246158.509434	269.773585	460.037736	62.366038	376.905660	273.581132	442.509434	186.241509	155.422642	1868.452830	2288
std	149187.485190	181.298589	261.647000	24.170913	118.817938	22.740518	48.863280	14.280641	11.275358	470.880867	513
min	82050.000000	82.000000	160.000000	17.600000	148.000000	187.300000	269.500000	164.500000	137.800000	1035.000000	1310
25%	142900.000000	136.000000	260.000000	40.000000	289.000000	258.800000	411.800000	178.800000	148.100000	1530.000000	1970
50%	178400.000000	204.000000	362.000000	58.000000	364.000000	270.000000	447.000000	180.900000	155.600000	1685.000000	2250
75%	339480.000000	372.000000	640.000000	80.000000	450.000000	290.000000	490.100000	193.500000	161.500000	2370.000000	2725
max	794000.000000	772.000000	1140.000000	100.000000	652.000000	327.500000	514.000000	255.800000	191.000000	2710.000000	3500

Data Tells About the growth of EV Industry:

Price: The EV market caters to a wide range of customers, from budget buyers to luxury segment enthusiasts.

Range and Battery Capacity: Battery capacities align with range, showcasing how technology is tailored to optimize distance per charge. Average range (~377 km) indicates a focus on meeting daily driving needs while moving towards longer ranges.

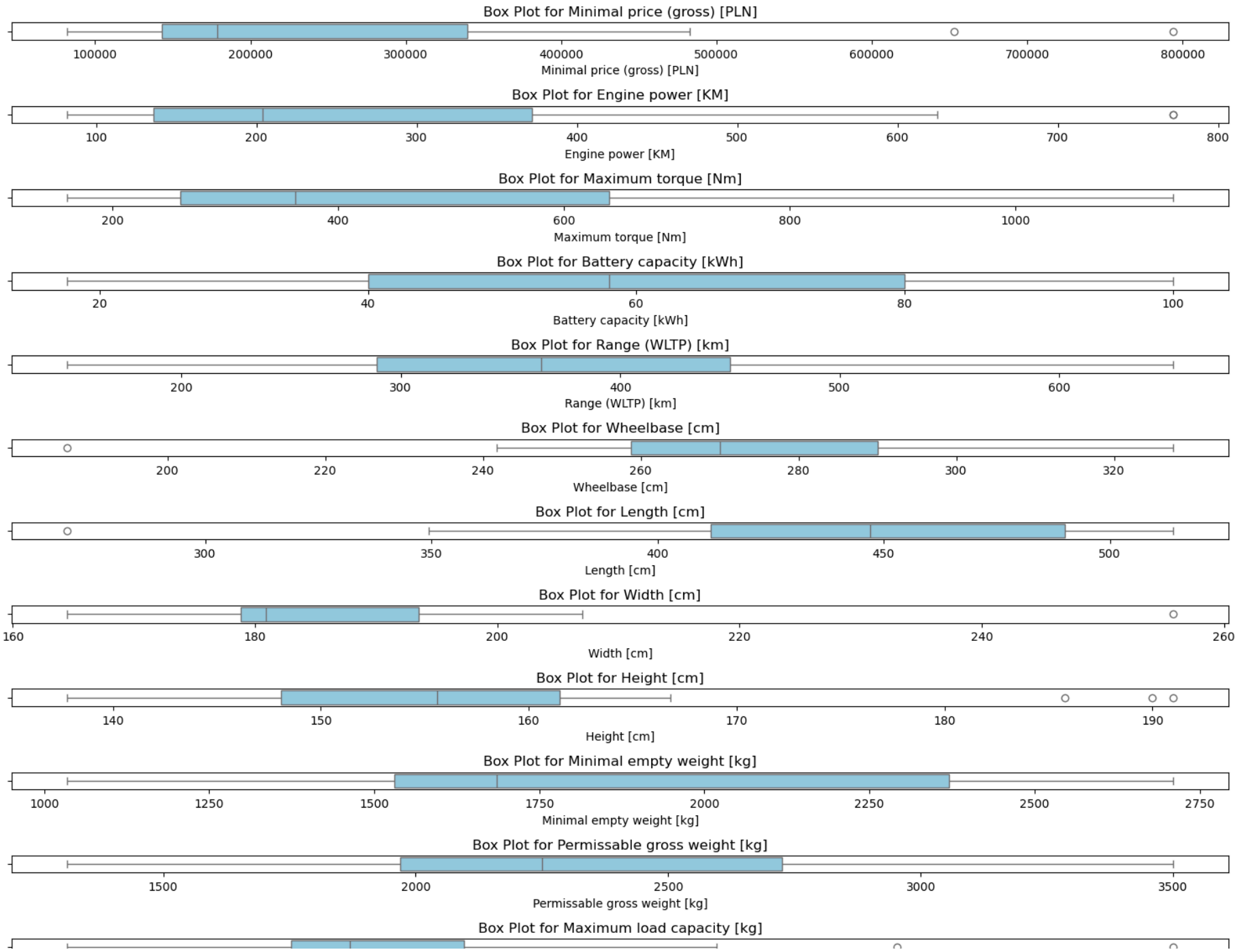
Performance: EVs are increasingly competitive in terms of speed and acceleration. Models show performance suitable for both urban and highway driving.

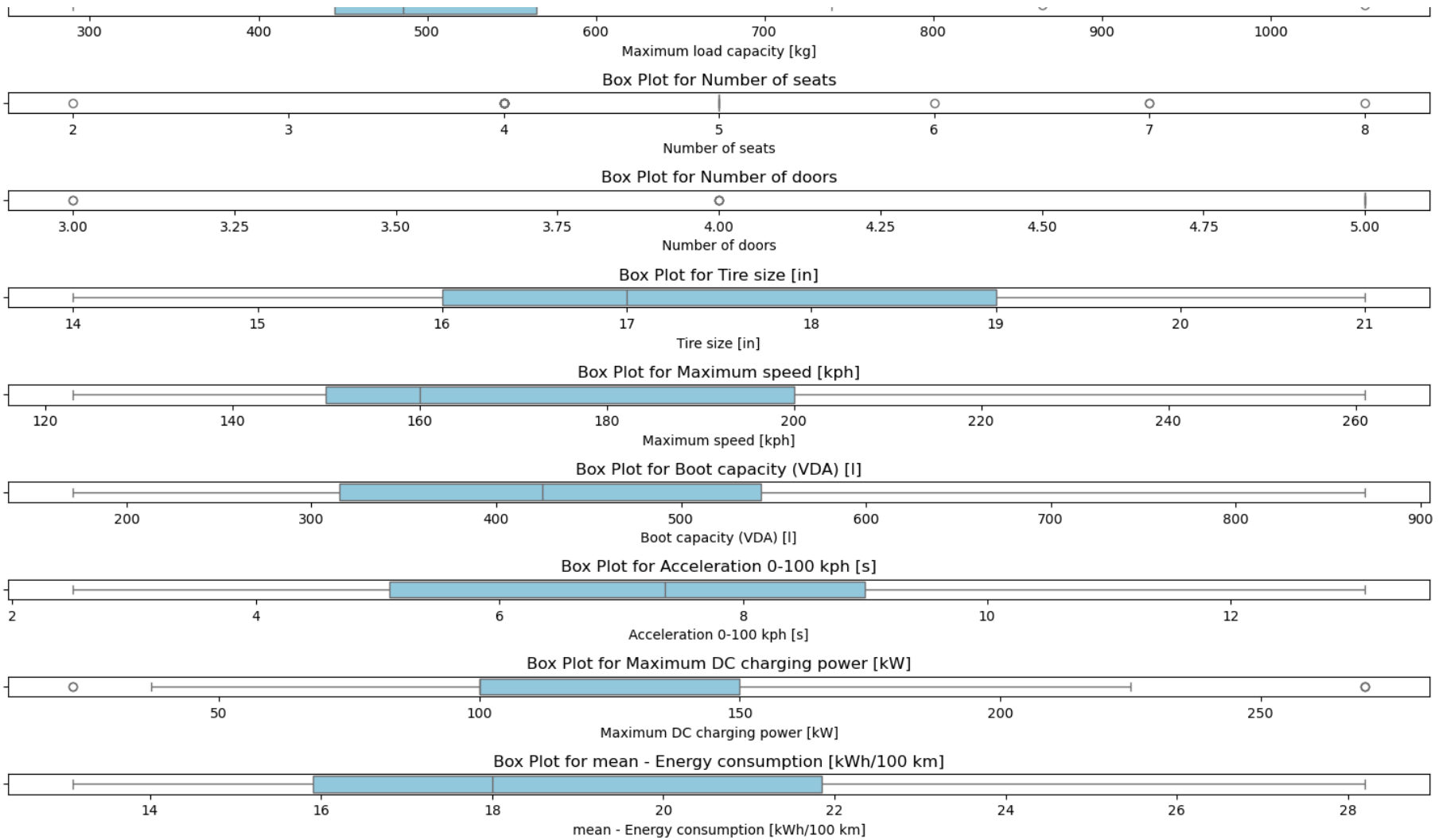
Diversity in Offerings: Variety in car size, seating capacity, and torque illustrates the versatility and growth of the industry.

Charging and Efficiency: DC charging power varies widely, reflecting innovations in quick charging. Energy consumption indicates efforts to balance power and efficiency.

Task 3: Outliers

In [57]:
```
continuous_columns = data.select_dtypes(include=['float64', 'int64']).columns

# Plot box plots for each continuous variable
plt.figure(figsize=(15, 20))
for i, column in enumerate(continuous_columns, 1):
    plt.subplot(len(continuous_columns), 1, i)
    sns.boxplot(data= data, x=column, color='skyblue')
    plt.title(f'Box Plot for {column}')
    plt.xlabel(column)
    plt.tight_layout()
```


Box Plot for Minimal price (gross) [PLN]
100000 200000 300000 400000 500000 600000 700000 800000
Minimal price (gross) [PLN]
Box Plot for Engine power [KM]
100 200 300 400 500 600 700 800
Engine power [KM]
Box Plot for Maximum torque [Nm]
200 400 600 800 1000
Maximum torque [Nm]
Box Plot for Battery capacity [kWh]
20 40 60 80 100
Battery capacity [kWh]
Box Plot for Range (WLTP) [km]
200 300 400 500 600
Range (WLTP) [km]
Box Plot for Wheelbase [cm]
200 220 240 260 280 300 320
Wheelbase [cm]
Box Plot for Length [cm]
300 350 400 450 500
Length [cm]
Box Plot for Width [cm]
160 180 200 220 240 260
Width [cm]
Box Plot for Height [cm]
140 150 160 170 180 190
Height [cm]
Box Plot for Minimal empty weight [kg]
1000 1250 1500 1750 2000 2250 2500 2750
Minimal empty weight [kg]
Box Plot for Permissable gross weight [kg]
1500 2000 2500 3000 3500
Permissable gross weight [kg]
Box Plot for Maximum load capacity [kg]



There are:

2 outliers for Price.

1 outlier for Engine Power.

1 outlier for Width.

3 outliers for Height.

2 outliers for Load Capacity.

1 outlier for DC charging power.

Wheelbase,Height, & DC charging power 1 outlier each on the lower end.

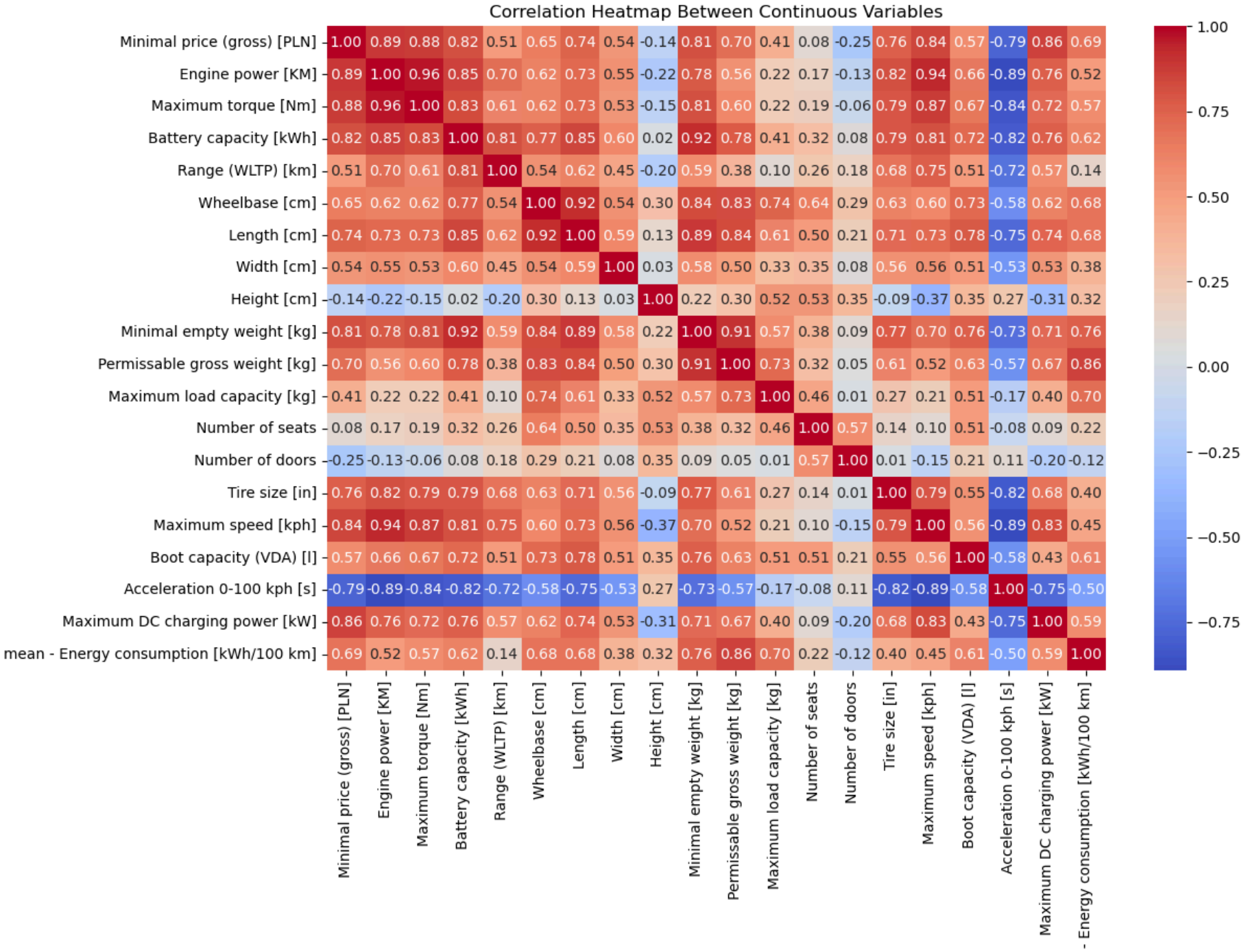
Task 4: Correlation

In [59]:
```
continuous_columns = data.select_dtypes(include=['float64', 'int64'])

# correlation matrix
correlation_matrix = continuous_columns.corr()

# Plotting the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
plt.title('Correlation Heatmap Between Continuous Variables')

#Corr Values
correlation_matrix
```

Out[59]:

	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Battery capacity [kWh]	Range (WLTP) [km]	Wheelbase [cm]	Length [cm]	Width [cm]	Height [cm]	Minimal empty weight [kg]	Permissable gross weight [kg]	M
Minimal price (gross) [PLN]	1.000000	0.887146	0.878673	0.816513	0.506311	0.645141	0.742424	0.538703	-0.143217	0.810863	0.702922	
Engine power [KM]	0.887146	1.000000	0.963558	0.852380	0.704177	0.622910	0.732844	0.547958	-0.222466	0.782882	0.558353	
Maximum torque [Nm]	0.878673	0.963558	1.000000	0.833661	0.608532	0.617950	0.731517	0.532670	-0.146665	0.809609	0.597192	
Battery capacity [kWh]	0.816513	0.852380	0.833661	1.000000	0.810439	0.768190	0.847859	0.596496	0.023974	0.922944	0.775988	
Range (WLTP) [km]	0.506311	0.704177	0.608532	0.810439	1.000000	0.543831	0.622771	0.449905	-0.204751	0.585427	0.377780	
Wheelbase [cm]	0.645141	0.622910	0.617950	0.768190	0.543831	1.000000	0.923959	0.542909	0.300418	0.835694	0.828054	
Length [cm]	0.742424	0.732844	0.731517	0.847859	0.622771	0.923959	1.000000	0.593569	0.134934	0.885366	0.836820	
Width [cm]	0.538703	0.547958	0.532670	0.596496	0.449905	0.542909	0.593569	1.000000	0.030915	0.575602	0.495926	
Height [cm]	-0.143217	-0.222466	-0.146665	0.023974	-0.204751	0.300418	0.134934	0.030915	1.000000	0.221711	0.295200	
Minimal empty weight [kg]	0.810863	0.782882	0.809609	0.922944	0.585427	0.835694	0.885366	0.575602	0.221711	1.000000	0.908366	
Permissable gross weight [kg]	0.702922	0.558353	0.597192	0.775988	0.377780	0.828054	0.836820	0.495926	0.295200	0.908366	1.000000	
Maximum load capacity [kg]	0.408814	0.219997	0.223430	0.414868	0.100778	0.744437	0.612319	0.329434	0.520849	0.565555	0.732673	

	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Battery capacity [kWh]	Range (WLTP) [km]	Wheelbase [cm]	Length [cm]	Width [cm]	Height [cm]	Minimal empty weight [kg]	Permissable gross weight [kg]	M
Number of seats	0.084391	0.170583	0.192065	0.322401	0.260799	0.643433	0.503590	0.347542	0.525857	0.380914	0.323294	
Number of doors	-0.251997	-0.127781	-0.059644	0.081257	0.180208	0.287811	0.213272	0.075470	0.352966	0.091225	0.052715	
Tire size [in]	0.762241	0.816400	0.788039	0.794093	0.680959	0.625286	0.713818	0.562873	-0.088007	0.773997	0.613579	
Maximum speed [kph]	0.839414	0.936912	0.869748	0.808666	0.749304	0.600772	0.728818	0.556110	-0.366791	0.700600	0.515988	
Boot capacity (VDA) [l]	0.569046	0.657612	0.670302	0.721615	0.512180	0.728138	0.778569	0.506408	0.352072	0.759029	0.629816	
Acceleration 0-100 kph [s]	-0.788205	-0.886769	-0.838427	-0.818494	-0.719866	-0.583901	-0.753557	-0.531822	0.268082	-0.728117	-0.574700	
Maximum DC charging power [kW]	0.855938	0.756852	0.718691	0.761795	0.566520	0.617206	0.736709	0.530594	-0.305874	0.712539	0.674793	
mean - Energy consumption [kWh/100 km]	0.685051	0.518251	0.568060	0.623438	0.140780	0.683567	0.677684	0.380051	0.321489	0.763811	0.856715	


Correlation Heatmap Between Continuous Variables
Minimal price (gross) [PLN] 1.00 0.89 0.88 0.82 0.51 0.65 0.74 0.54 -0.14 0.81 0.70 0.41 0.08 -0.25 0.76 0.84 0.57 -0.79 0.86 0.69
Engine power [KM] 0.89 1.00 0.96 0.85 0.70 0.62 0.73 0.55 -0.22 0.78 0.56 0.22 0.17 -0.13 0.82 0.94 0.66 -0.89 0.76 0.52
Maximum torque [Nm] 0.88 0.96 1.00 0.83 0.61 0.62 0.73 0.53 -0.15 0.81 0.60 0.22 0.19 -0.06 0.79 0.87 0.67 -0.84 0.72 0.57
Battery capacity [kWh] 0.82 0.85 0.83 1.00 0.81 0.77 0.85 0.60 0.02 0.92 0.78 0.41 0.32 0.08 0.79 0.81 0.72 -0.82 0.76 0.62
Range (WLTP) [km] 0.51 0.70 0.61 0.81 1.00 0.54 0.62 0.45 -0.20 0.59 0.38 0.10 0.26 0.18 0.68 0.75 0.51 -0.72 0.57 0.14
Wheelbase [cm] 0.65 0.62 0.62 0.77 0.54 1.00 0.92 0.54 0.30 0.84 0.83 0.74 0.64 0.29 0.63 0.60 0.73 -0.58 0.62 0.68
Length [cm] 0.74 0.73 0.73 0.85 0.62 0.92 1.00 0.59 0.13 0.89 0.84 0.61 0.50 0.21 0.71 0.73 0.78 -0.75 0.74 0.68
Width [cm] 0.54 0.55 0.53 0.60 0.45 0.54 0.59 1.00 0.03 0.58 0.50 0.33 0.35 0.08 0.56 0.56 0.51 -0.53 0.53 0.38
Height [cm] -0.14 -0.22 -0.15 0.02 -0.20 0.30 0.13 0.03 1.00 0.22 0.30 0.52 0.53 0.35 -0.09 -0.37 0.35 0.27 -0.31 0.32
Minimal empty weight [kg] 0.81 0.78 0.81 0.92 0.59 0.84 0.89 0.58 0.22 1.00 0.91 0.57 0.38 0.09 0.77 0.70 0.76 -0.73 0.71 0.76
Permissable gross weight [kg] 0.70 0.56 0.60 0.78 0.38 0.83 0.84 0.50 0.30 0.91 1.00 0.73 0.32 0.05 0.61 0.52 0.63 -0.57 0.67 0.86
Maximum load capacity [kg] 0.41 0.22 0.22 0.41 0.10 0.74 0.61 0.33 0.52 0.57 0.73 1.00 0.46 0.01 0.27 0.21 0.51 -0.17 0.40 0.70
Number of seats 0.08 0.17 0.19 0.32 0.26 0.64 0.50 0.35 0.53 0.38 0.32 0.46 1.00 0.57 0.14 0.10 0.51 -0.08 0.09 0.22
Number of doors -0.25 -0.13 -0.06 0.08 0.18 0.29 0.21 0.08 0.35 0.09 0.05 0.01 0.57 1.00 0.01 -0.15 0.21 0.11 -0.20 -0.12
Tire size [in] 0.76 0.82 0.79 0.79 0.68 0.63 0.71 0.56 -0.09 0.77 0.61 0.27 0.14 0.01 1.00 0.79 0.55 -0.82 0.68 0.40
Maximum speed [kph] 0.84 0.94 0.87 0.81 0.75 0.60 0.73 0.56 -0.37 0.70 0.52 0.21 0.10 -0.15 0.79 1.00 0.56 -0.89 0.83 0.45
Boot capacity (VDA) [l] 0.57 0.66 0.67 0.72 0.51 0.73 0.78 0.51 0.35 0.76 0.63 0.51 0.51 0.21 0.55 0.56 1.00 -0.58 0.43 0.61
Acceleration 0-100 kph [s] -0.79 -0.89 -0.84 -0.82 -0.72 -0.58 -0.75 -0.53 0.27 -0.73 -0.57 -0.17 -0.08 0.11 -0.82 -0.89 -0.58 1.00 -0.75 -0.50
Maximum DC charging power [kW] 0.86 0.76 0.72 0.76 0.57 0.62 0.74 0.53 -0.31 0.71 0.67 0.40 0.09 -0.20 0.68 0.83 0.43 -0.75 1.00 0.59
mean - Energy consumption [kWh/100 km] 0.69 0.52 0.57 0.62 0.14 0.68 0.68 0.38 0.32 0.76 0.86 0.70 0.22 -0.12 0.40 0.45 0.61 -0.50 0.59 1.00
Minimal price (gross) [PLN]
Engine power [KM]
Maximum torque [Nm]
Battery capacity [kWh]
Range (WLTP) [km]
Wheelbase [cm]
Length [cm]
Width [cm]
Height [cm]
Minimal empty weight [kg]
Permissable gross weight [kg]
Maximum load capacity [kg]
Number of seats
Number of doors
Tire size [in]
Maximum speed [kph]
Boot capacity (VDA) [l]
Acceleration 0-100 kph [s]
Maximum DC charging power [kW]
- Energy consumption [kWh/100 km]
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

mean

We can find very high Correlation between

1. Price & Engine Power (Positive)

2. Price & Torque (Positive)

3. Power & Torque (Positive)

4. Price & Empty Weight (Positive)

5. Price & Max Speed (Positive)

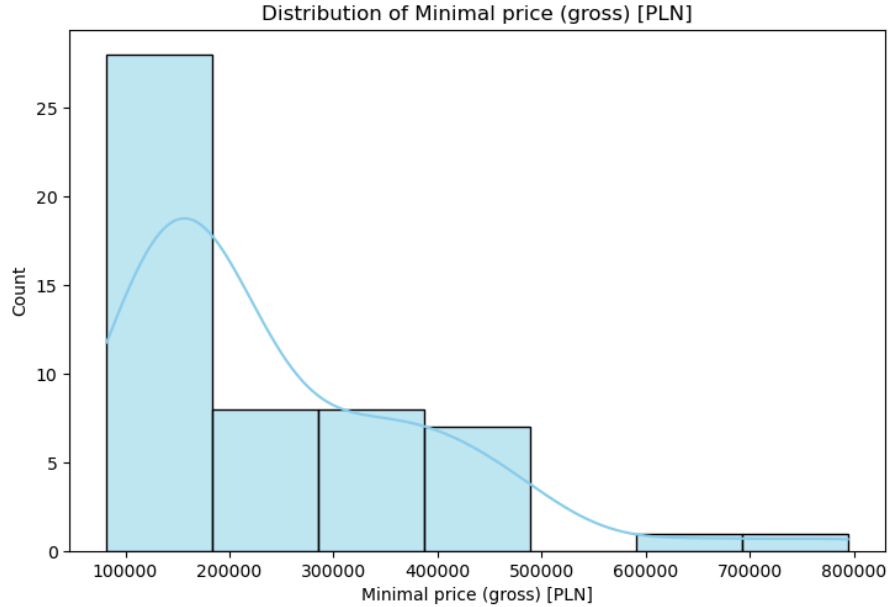
6. Price & Accelaration (Negative)

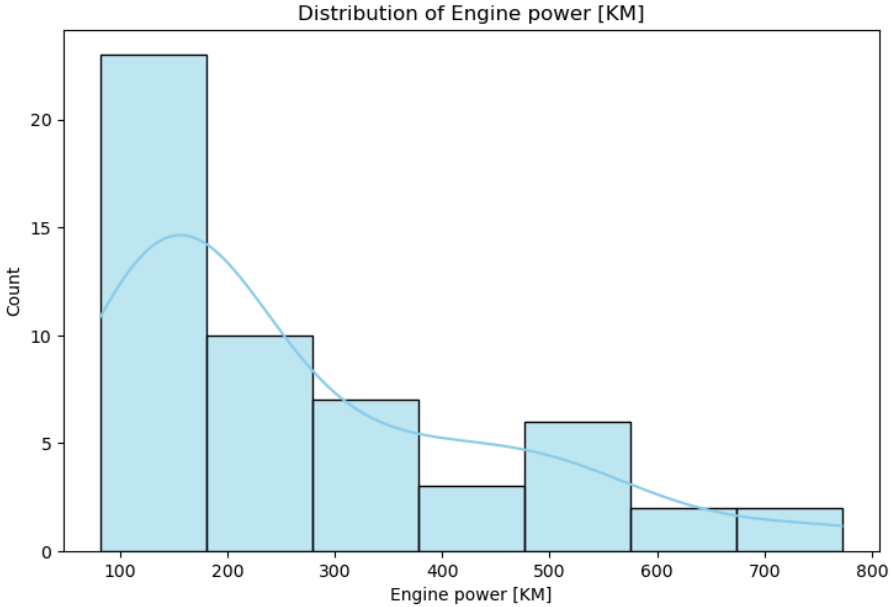
7. Price & DC charging power (Positive)

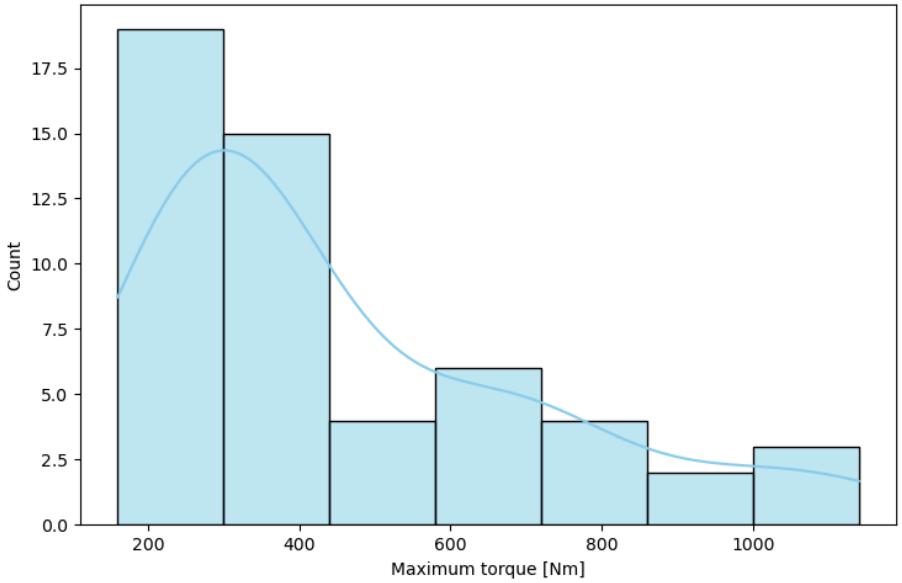
Similarly we can explore other correlations

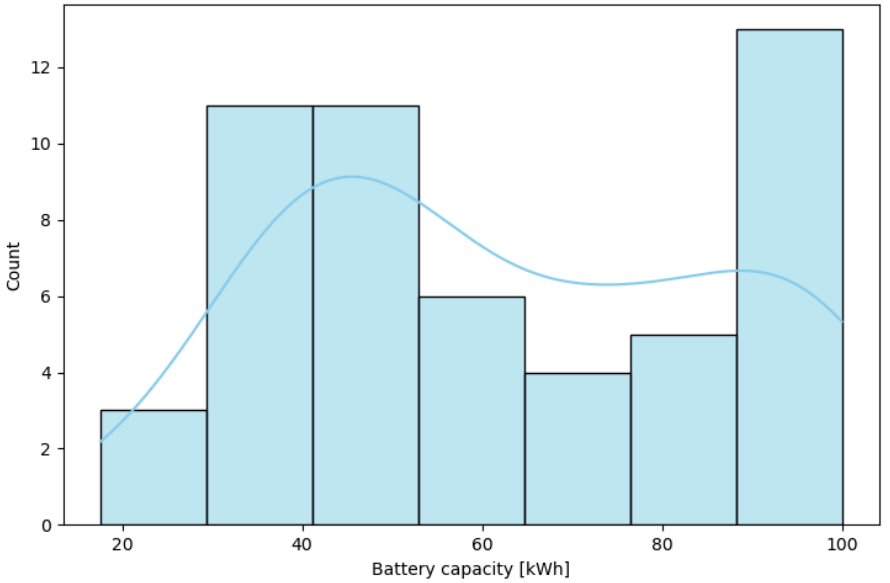
Task 5: Trend in data

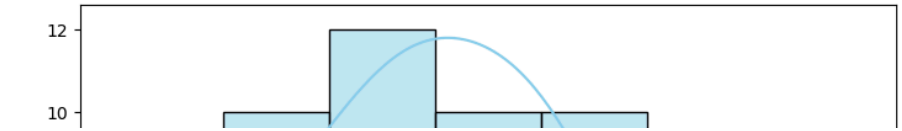
In [61]:
```
categorical_columns = data.select_dtypes(include=['object']).columns
continuous_columns = data.select_dtypes(include=['float64', 'int64']).columns

# Plotting Distribution for Continuous Variables
plt.figure(figsize=(15, 100))
for i, column in enumerate(continuous_columns, 1):
    plt.subplot(len(continuous_columns), 2, i)
    sns.histplot(data[column], kde=True, color='skyblue')
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
plt.tight_layout()
```


Distribution of Minimal price (gross) [PLN]
Count
Minimal price (gross) [PLN]
0
5
10
15
20
25
100000
200000
300000
400000
500000
600000
700000
800000

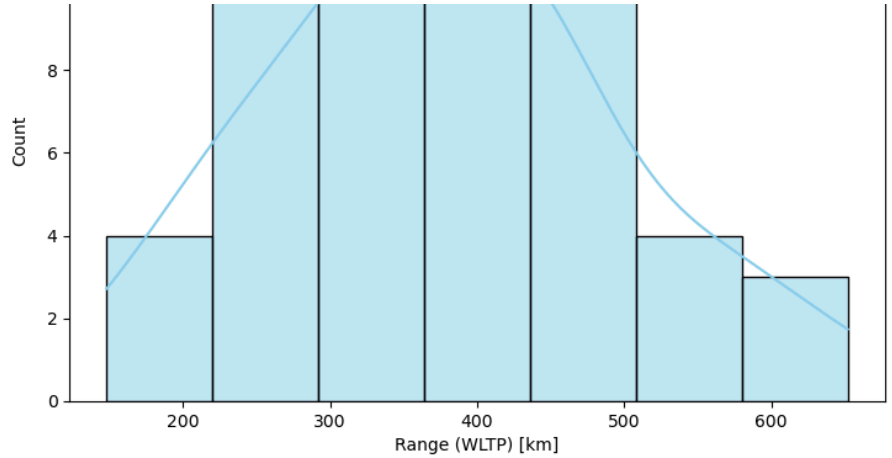

Distribution of Engine power [KM]
Count
Engine power [KM]
0
5
10
15
20
100
200
300
400
500
600
700
800

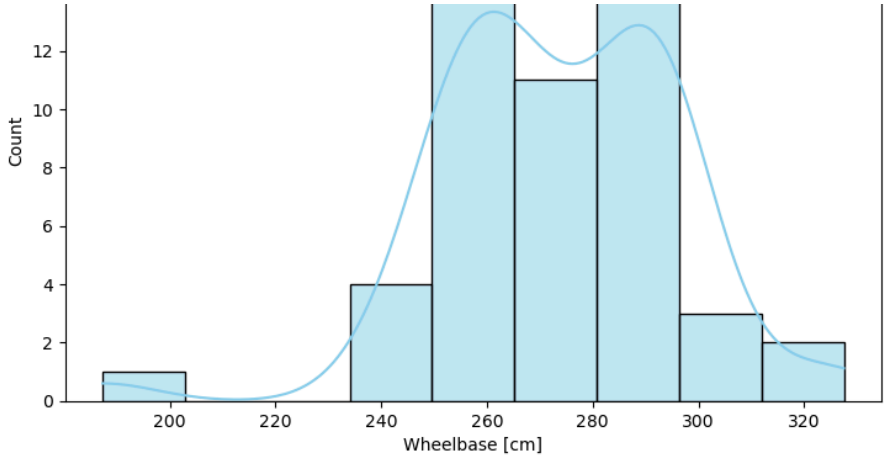

Distribution of Maximum torque [Nm]
Count
Maximum torque [Nm]
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
200
400
600
800
1000

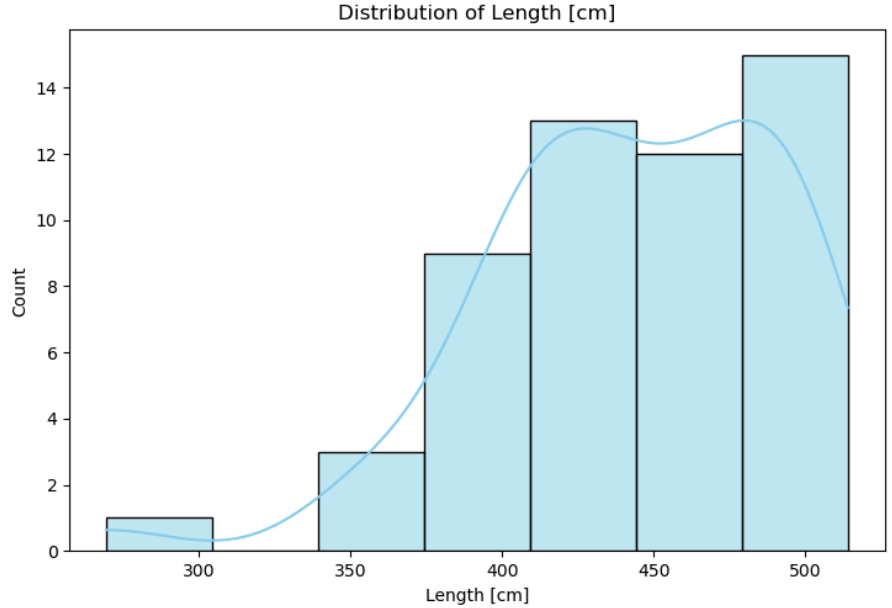

Distribution of Battery capacity [kWh]
Count
Battery capacity [kWh]
0
2
4
6
8
10
12
20
40
60
80
100

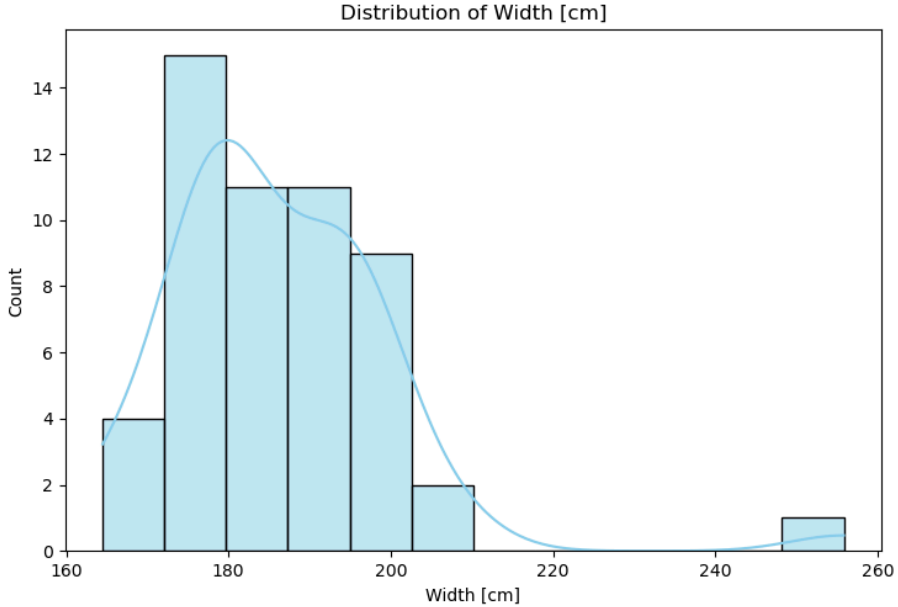

Distribution of Range (WLTP) [km]
10
12

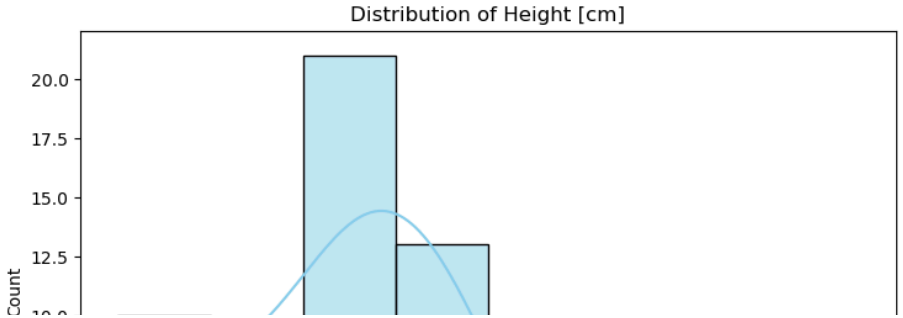

Distribution of Wheelbase [cm]
14
16

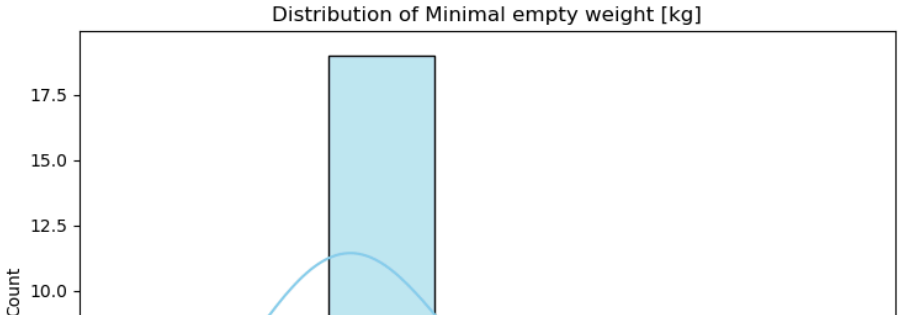

Count
Range (WLTP) [km]

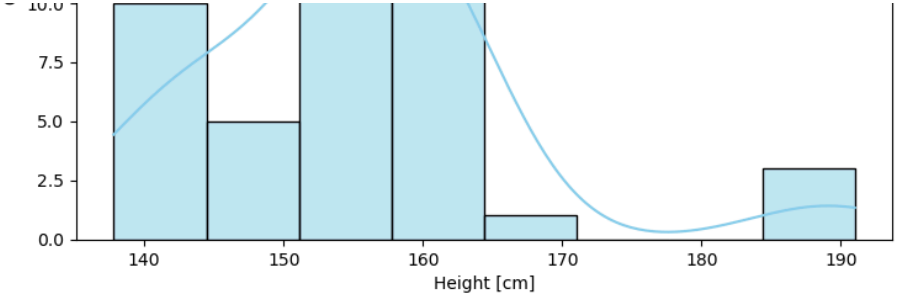

Count
Wheelbase [cm]

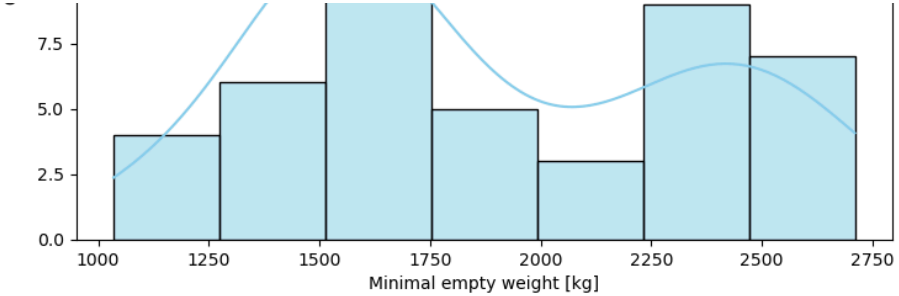

Distribution of Length [cm]
Count
Length [cm]


Distribution of Width [cm]
Count
Width [cm]

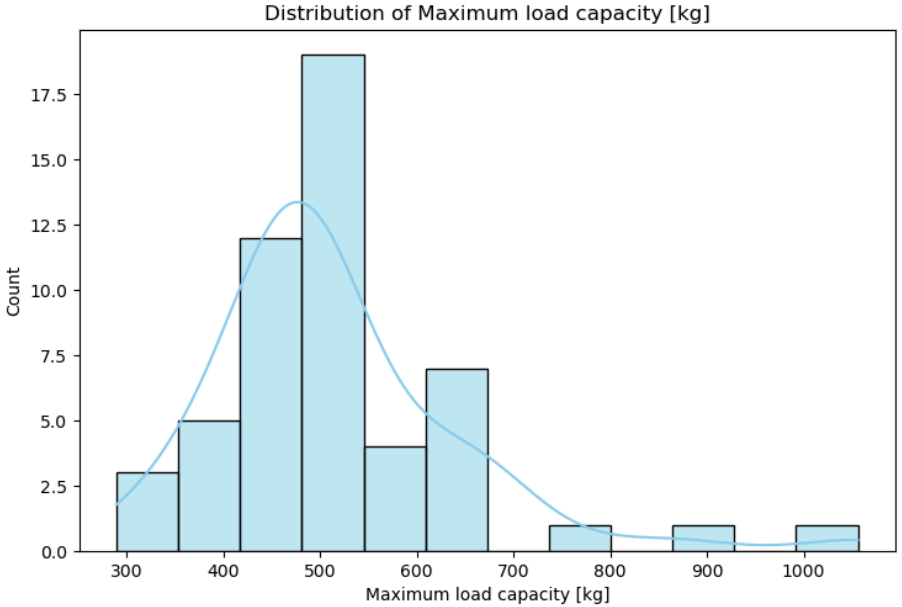

Distribution of Height [cm]
Count

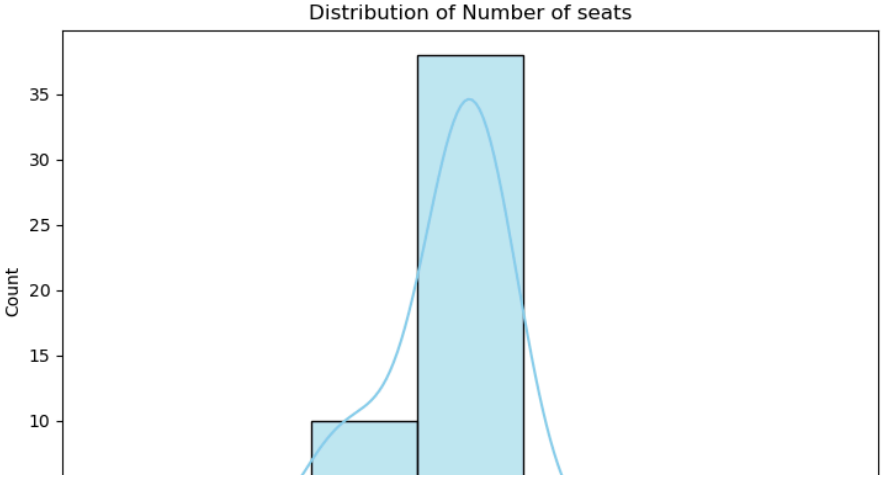

Distribution of Minimal empty weight [kg]
Count

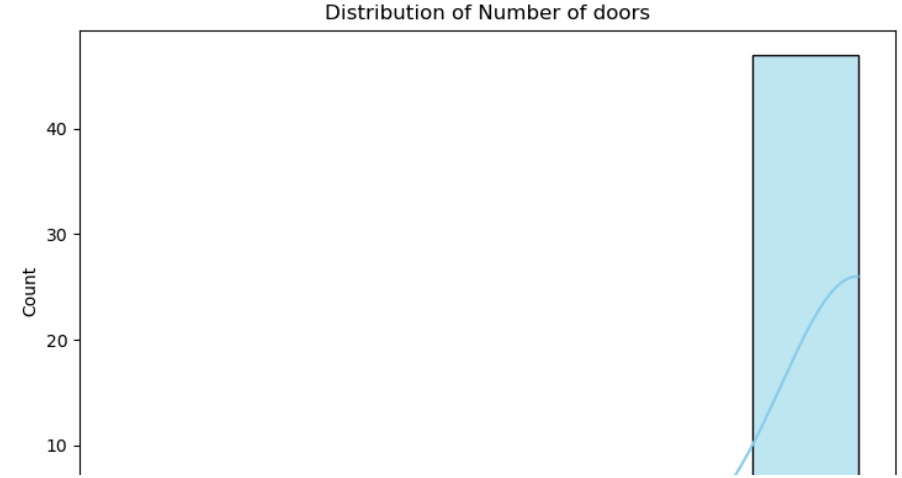

Height [cm]

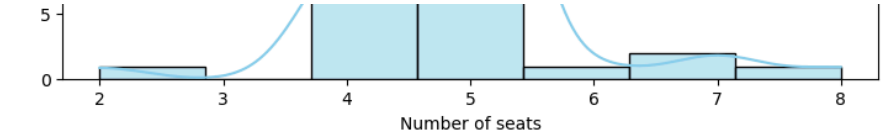

Minimal empty weight [kg]

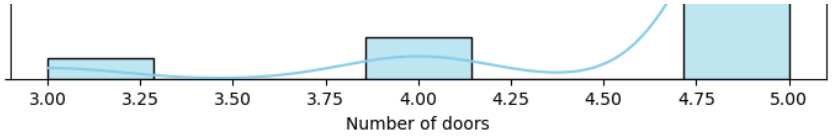

Distribution of Permissable gross weight [kg]
Count
Permissable gross weight [kg]

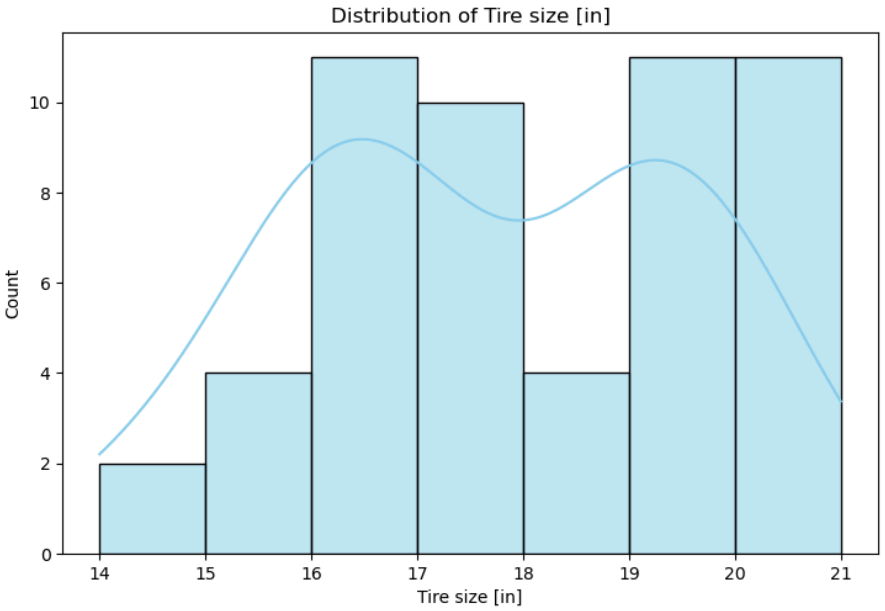

Distribution of Maximum load capacity [kg]
Count
Maximum load capacity [kg]

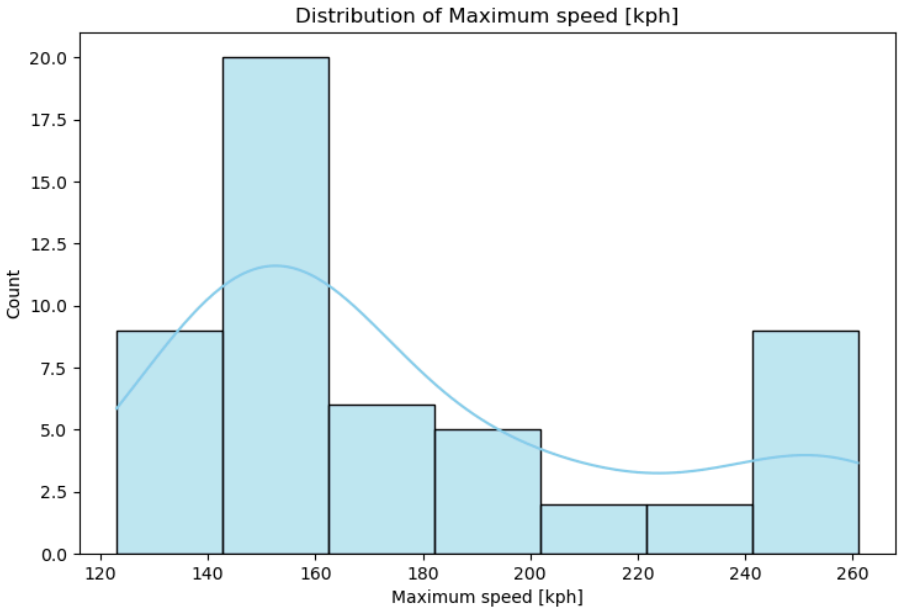

Distribution of Number of seats
Count

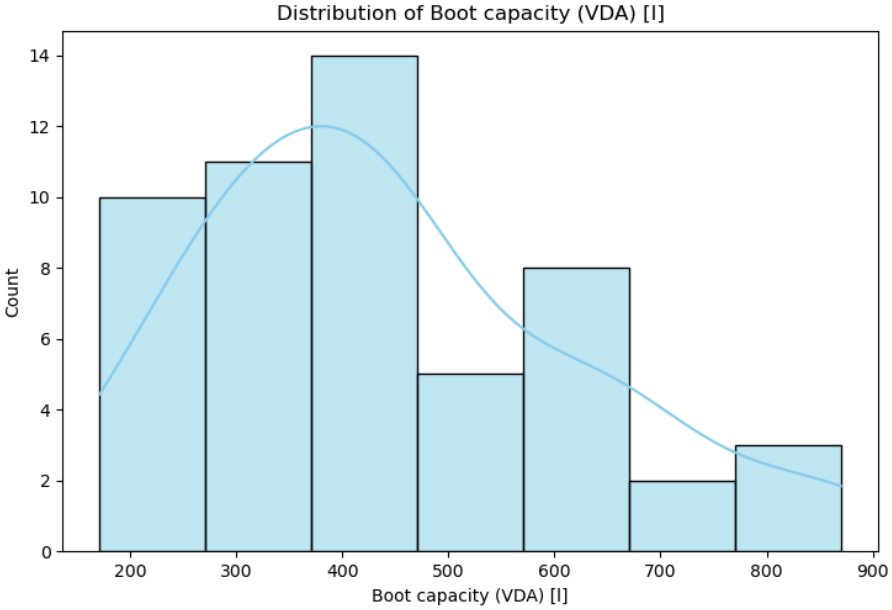

Distribution of Number of doors
Count

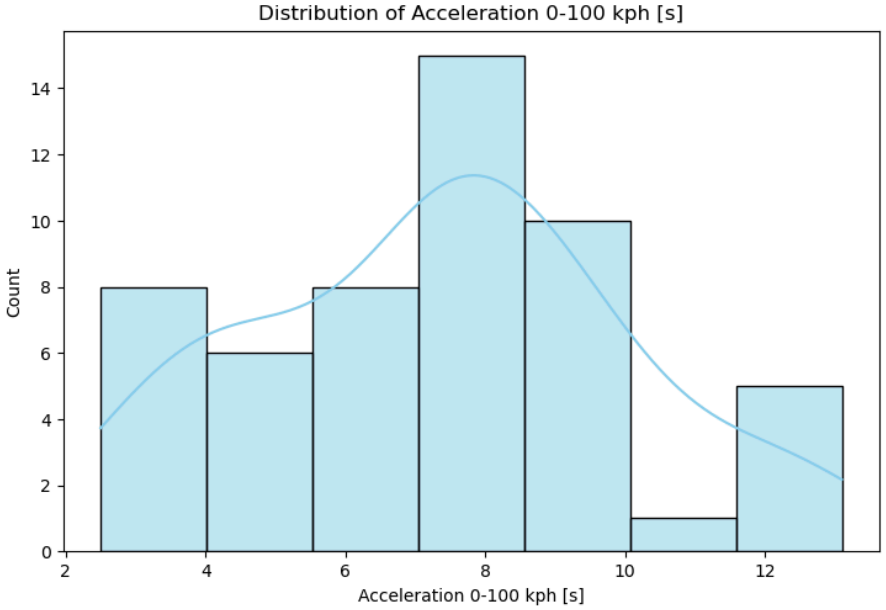

Number of seats

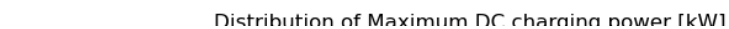

Number of doors


Distribution of Tire size [in]
Count
Tire size [in]

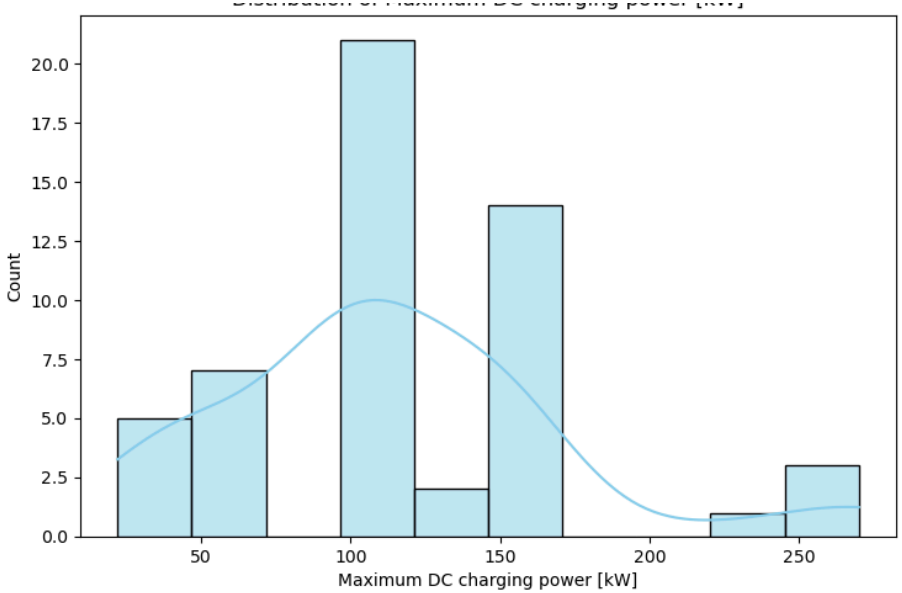

Distribution of Maximum speed [kph]
Count
Maximum speed [kph]

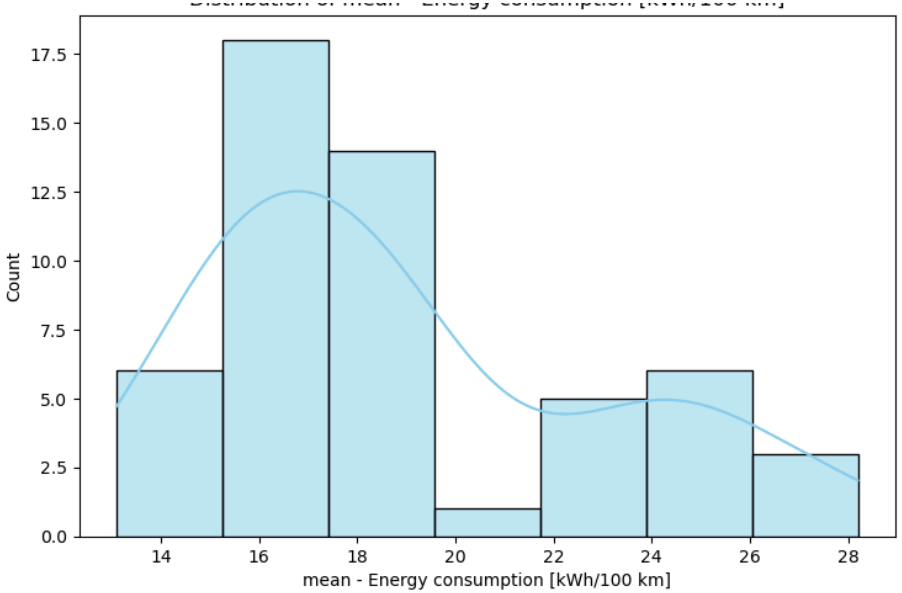

Distribution of Boot capacity (VDA) [l]
Count
Boot capacity (VDA) [l]


Distribution of Acceleration 0-100 kph [s]
Count
Acceleration 0-100 kph [s]


Distribution of Maximum DC charging power [kW]


Distribution of mean - Energy consumption [kWh/100 km]





Trend in few important features in the EVs.

1. Price (PLN): Right-skewed, indicating a higher concentration of lower-priced EVs with some expensive outliers.

2. Engine Power [KM]: Bimodal distribution, showing two common ranges of power likely catering to different vehicle segments.

3. Battery Capacity [kWh]: Left-skewed, with most vehicles clustering around mid to high battery capacities.

4. Range (WLTP) [km]: Slightly right-skewed, reflecting more EVs with shorter to moderate ranges.

5. Acceleration 0-100 kph [s]: Left-skewed, as most EVs are designed for quick acceleration within a few seconds.

6. DC charging power [kW]: Uniformly distributed, indicating a balanced range of fast-charging capabilities across models.

7. mean - Energy consumption [kWh/100 km]: Normal distribution, showing average energy efficiency for most vehicles.

8. Torque [Nm]: Right-skewed, suggesting most EVs have moderate torque, with fewer high-performance models.

9. Boot capacity (VDA) [l]: Normal distribution, indicating standard luggage capacities.

10. Permissible gross weight [kg]: Narrow range with a slight right skew, reflecting vehicle weight regulations.

11. Maximum load capacity [kg]: Right-skewed, suggesting most EVs have moderate load capabilities.

For relation between the variables we can find the correlation coefficient in task 4.

Task 6: Manufacturer Analysis

There is no relevant data for analysis of performance of manufaturers.

In [63]:
```
data.head()
```

Out[63]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Number of seats	Num do
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	438	...	3130.0	640.0	5	
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	340	...	3040.0	670.0	5	
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	364	...	3130.0	565.0	5	
3	Audi e-tron Sportback 50 quattro	Audi	e-tron Sportback 50 quattro	319700	313	540	disc (front + rear)	4WD	71.0	346	...	3040.0	640.0	5	
4	Audi e-tron Sportback 55 quattro	Audi	e-tron Sportback 55 quattro	357000	360	664	disc (front + rear)	4WD	95.0	447	...	3130.0	670.0	5	

5 rows × 25 columns

In [65]:
```
data['Make'].value_counts().count()
```

Out[65]: 20

In [67]:
```
data['Make'].value_counts()
```

Out[67]:
```
Make
Tesla            7
Audi             6
Kia              4
Porsche          4
Volkswagen       4
Hyundai          3
BMW              3
Nissan           3
Honda            2
Mercedes-Benz    2
Opel             2
Peugeot          2
Renault          2
Smart            2
Citroën          2
Jaguar           1
Mazda            1
DS               1
Skoda            1
Mini             1
Name: count, dtype: int64
```

From above summary it can be found out that there are 20 manufacturers. In total there are 53 models.

Most models are with Tesla 7 then Audi 6.

There are 5 makers with 1 model each.

Task 7: Top EVs by Range, Accelaration, Max Speed.

In [71]:
```
data.head()
```

Out[71]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Number of seats	Num do
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	438	...	3130.0	640.0	5	
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	340	...	3040.0	670.0	5	
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	364	...	3130.0	565.0	5	
3	Audi e-tron Sportback 50 quattro	Audi	e-tron Sportback 50 quattro	319700	313	540	disc (front + rear)	4WD	71.0	346	...	3040.0	640.0	5	
4	Audi e-tron Sportback 55 quattro	Audi	e-tron Sportback 55 quattro	357000	360	664	disc (front + rear)	4WD	95.0	447	...	3130.0	670.0	5	

5 rows × 25 columns

In [73]:
```
#Top 5 EVs by Range
data.sort_values(by='Range (WLTP) [km]', ascending=False).head(5)
```

Out[73]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Number of seats
42	Tesla Model S Long Range Plus	Tesla	Model S Long Range Plus	368990	525	755	disc (front + rear)	4WD	100.0	652	...	2288.844444	486.0	5
43	Tesla Model S Performance	Tesla	Model S Performance	443990	772	1140	disc (front + rear)	4WD	100.0	639	...	2288.844444	486.0	5
40	Tesla Model 3 Long Range	Tesla	Model 3 Long Range	235490	372	510	disc (front + rear)	4WD	75.0	580	...	2288.844444	486.0	5
41	Tesla Model 3 Performance	Tesla	Model 3 Performance	260490	480	639	disc (front + rear)	4WD	75.0	567	...	2288.844444	486.0	5
44	Tesla Model X Long Range Plus	Tesla	Model X Long Range Plus	407990	525	755	disc (front + rear)	4WD	100.0	561	...	2288.844444	486.0	7

5 rows × 25 columns

Top 5 Ranges are between 652 & 561.

In [76]:
```
# Top 5 models by Accelaration (lowest first)
data.sort_values(by='Acceleration 0-100 kph [s]', ascending=True).head(5)
```

Out[76]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Numbe of seat
43	Tesla Model S Performance	Tesla	Model S Performance	443990	772	1140	disc (front + rear)	4WD	100.0	639	...	2288.844444	486.0	5
33	Porsche Taycan Turbo S	Porsche	Taycan Turbo S	794000	625	1050	disc (front + rear)	4WD	93.4	412	...	2870.000000	575.0	4
45	Tesla Model X Performance	Tesla	Model X Performance	482990	772	1140	disc (front + rear)	4WD	100.0	548	...	2288.844444	486.0	7
32	Porsche Taycan Turbo	Porsche	Taycan Turbo	653000	625	850	disc (front + rear)	4WD	93.4	450	...	2880.000000	575.0	4
41	Tesla Model 3 Performance	Tesla	Model 3 Performance	260490	480	639	disc (front + rear)	4WD	75.0	567	...	2288.844444	486.0	5

5 rows × 25 columns

Top 5 Models by Accelaration i.e. low seconds first lie between 2.5 seconds to 3.3 seconds.

In [79]:
```
# Top 5 models by Max Speed
data.sort_values(by='Maximum speed [kph]', ascending=False).head(5)
```

Out[79]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Numbe of seat
41	Tesla Model 3 Performance	Tesla	Model 3 Performance	260490	480	639	disc (front + rear)	4WD	75.0	567	...	2288.844444	486.0	5
43	Tesla Model S Performance	Tesla	Model S Performance	443990	772	1140	disc (front + rear)	4WD	100.0	639	...	2288.844444	486.0	5
45	Tesla Model X Performance	Tesla	Model X Performance	482990	772	1140	disc (front + rear)	4WD	100.0	548	...	2288.844444	486.0	7
33	Porsche Taycan Turbo S	Porsche	Taycan Turbo S	794000	625	1050	disc (front + rear)	4WD	93.4	412	...	2870.000000	575.0	4
32	Porsche Taycan Turbo	Porsche	Taycan Turbo	653000	625	850	disc (front + rear)	4WD	93.4	450	...	2880.000000	575.0	4

5 rows × 25 columns

Top 5 Max speeds are 261 & 260. (3 Are by Tesla & 2 by Porsche)

Task 8: Hypothesis - Similarity in 'Engine Power' between Tesla & Audi

In [83]:
```
data.head()
```

Out[83]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	Range (WLTP) [km]	...	Permissable gross weight [kg]	Maximum load capacity [kg]	Number of seats	Num do
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	438	...	3130.0	640.0	5	
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	340	...	3040.0	670.0	5	
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	364	...	3130.0	565.0	5	
3	Audi e-tron Sportback 50 quattro	Audi	e-tron Sportback 50 quattro	319700	313	540	disc (front + rear)	4WD	71.0	346	...	3040.0	640.0	5	
4	Audi e-tron Sportback 55 quattro	Audi	e-tron Sportback 55 quattro	357000	360	664	disc (front + rear)	4WD	95.0	447	...	3130.0	670.0	5	

5 rows × 25 columns

In [85]:
```
# 2 Sample t test.
from scipy.stats import ttest_ind

tesla_data = data[data['Make'] == 'Tesla']['Engine power [KM]']
audi_data = data[data['Make'] == 'Audi']['Engine power [KM]']

# Perform the 2-sample t-test
```

```
t_stat, p_value = ttest_ind(tesla_data, audi_data, equal_var=False)

print("2-Sample T-Test Results:")
print(f"T-Statistic: {t_stat}")
print(f"P-Value: {p_value}")

alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The average engine power of Tesla and Audi is significantly different.")
else:
    print("Fail to reject the null hypothesis: No significant difference in the average engine power of Tesla and Audi.")
```

```
2-Sample T-Test Results:
T-Statistic: 1.7939951827297178
P-Value: 0.10684105068839565
Fail to reject the null hypothesis: No significant difference in the average engine power of Tesla and Audi.
```

Task 9: Future Prediction of EVs

Based on the data summary, potential future trends and predictions for the EV industry: 1. Increasing Range and Battery Efficiency Prediction: As battery technology advances, the average range of EVs (~377 km) will steadily increase, with more models exceeding 500 km on a single charge. 2. Broader Market Segmentation Prediction: The market will see further segmentation with an increasing number of models catering to diverse needs, from ultra-budget EVs to luxury and high-performance vehicles. 3. Dominance of Performance-Oriented EVs Prediction: EVs with quick acceleration (<3 seconds) and high torque will dominate the high-end segment as manufacturers compete with traditional sports and luxury vehicles. 4. Innovations in Charging Infrastructure Prediction: Uniform distribution of DC charging power suggests future improvements in charging speed and accessibility, with innovations reducing charging times to minutes. 5. Standardization of Energy Efficiency Prediction: Energy consumption will converge toward a global standard, ensuring most vehicles operate efficiently while maintaining performance. 6. Increased Focus on Compact and Utility Models Prediction: Smaller, urban-focused EVs and versatile utility EVs (e.g., SUVs and crossovers) will dominate production, reflecting shifting consumer preferences. 7. Enhanced Brand Competition Prediction: Tesla will maintain its leadership, but competitors like Audi, Porsche, and emerging brands will capture more market share by focusing on niche innovations and competitive pricing. 8. Growth in Global EV Adoption Prediction: Adoption of EVs will accelerate globally, driven by falling costs, government incentives, and increased awareness of environmental benefits.

Task 10: Recommendations & Conclusion

1. Price and Market Segmentation: The EV industry should continue offering a broad range of price options to cater to diverse customer segments, from budget-conscious buyers to luxury enthusiasts. Addressing the price outliers with advanced features can strengthen the appeal of high-end models, but affordability should remain a priority for market penetration. 2. Range and Battery Capacity: The strong correlation between battery capacity and range indicates that investments in battery technology can enhance range performance. While the average range (~377 km) meets daily needs, manufacturers should focus on extending ranges to make EVs more viable for long-distance travel. 3. Performance and Innovation: Quick acceleration and higher torque are attractive to performance-oriented buyers. Maintaining the trend of left-skewed acceleration times and competitive torque can help attract more customers. With bimodal distributions in engine power, manufacturers should maintain offerings for both economy and premium performance segments. 4. Charging and Efficiency Uniform distribution of DC charging power suggests balanced fast-charging capabilities. Manufacturers should focus on innovations that improve charging speed and convenience. Energy consumption trends suggest good efficiency levels, but continued optimization will be essential for reducing costs and improving sustainability. 5. Addressing Outliers Price, engine power, width, height, and other

features with outliers indicate room for improvement in design standardization and targeting specific consumer needs.For example, outliers in load capacity and charging power could represent niche models. Efforts to diversify while managing costs can cater to these specialized demands. 6. Correlation Insights Strong positive correlations between price and features like engine power, torque, and max speed suggest a direct relationship between premium features and cost. Conversely, the negative correlation between price and acceleration highlights a potential area to market high-performance EVs as premium vehicles. 7. Trends in Important Features Right-skewed features like price, range, and torque indicate a focus on economy to mid-tier offerings, with high-end outliers catering to niche markets. Normal distributions for energy consumption and boot capacity reflect standardization in efficiency and utility, which can be marketed as a strong selling point. 8. Market Presence Tesla leads the market with 7 models, indicating its strong presence. Other manufacturers like Audi and Porsche are also competing well. Smaller manufacturers with one model each should consider expanding their portfolio to strengthen their market presence. 9. Performance Leaders Models with the highest range, fastest acceleration, and top speeds (largely from Tesla and Porsche) show the industry's ability to deliver innovation. Highlighting these leaders can boost overall market credibility. 10. Tesla vs. Audi Engine Power Comparison The 2-sample t-test results indicate no significant difference in average engine power between Tesla and Audi. Both brands compete closely, showing parity in this feature.

In [ ]: