```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
urvishahir_electric_vehicle_specifications_dataset_2025_path = kagglehub.dataset_download('urvishahir/electric-vehicle-specifications-dataset-2025'

print('Data source import complete.')
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run A
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/electric-vehicle-specifications-dataset-2025/electric_vehicles_spec_2025.csv.csv
```

Double-click (or enter) to edit

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
df = pd.read_csv("/kaggle/input/electric-vehicle-specifications-dataset-2025/electric_vehicles_spec_2025.csv.csv")
df
```

```
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
```

	brand	model	top_speed_kmh	battery_capacity_kWh	battery_type	number_of_cells	torque_nm	efficiency_wh_per_km	range_km	acceler
0	Abarth	500e Convertible	155	37.8	Lithium-ion	192.0	235.0	156	225	
1	Abarth	500e Hatchback	155	37.8	Lithium-ion	192.0	235.0	149	225	
2	Abarth	600e Scorpionissima	200	50.8	Lithium-ion	102.0	345.0	158	280	
3	Abarth	600e Turismo	200	50.8	Lithium-ion	102.0	345.0	158	280	
4	Aiways	U5	150	60.0	Lithium-ion	NaN	310.0	156	315	
...	...	...	...	...	...	...	...	...	...	
473	Zeekr	7X Premium RWD	210	71.0	Lithium-ion	NaN	440.0	148	365	
474	Zeekr	X Core RWD (MY25)	190	49.0	Lithium-ion	NaN	343.0	148	265	
475	Zeekr	X Long Range RWD (MY25)	190	65.0	Lithium-ion	NaN	343.0	146	360	
476	Zeekr	X Privilege AWD (MY25)	190	65.0	Lithium-ion	NaN	543.0	153	350	
477	firefly	NaN	150	41.2	Lithium-ion	112.0	200.0	125	250	

478 rows × 22 columns

Data exploration

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 478 entries, 0 to 477
Data columns (total 22 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   brand                      478 non-null    object
 1   model                      477 non-null    object
 2   top_speed_kmh              478 non-null    int64
 3   battery_capacity_kWh       478 non-null    float64
 4   battery_type               478 non-null    object
 5   number_of_cells            276 non-null    float64
 6   torque_nm                  471 non-null    float64
 7   efficiency_wh_per_km       478 non-null    int64
 8   range_km                   478 non-null    int64
 9   acceleration_0_100_s       478 non-null    float64
 10  fast_charging_power_kw_dc  477 non-null    float64
 11  fast_charge_port           477 non-null    object
 12  towing_capacity_kg         452 non-null    float64
 13  cargo_volume_l             477 non-null    object
 14  seats                      478 non-null    int64
 15  drivetrain                 478 non-null    object
 16  segment                    478 non-null    object
 17  length_mm                  478 non-null    int64
 18  width_mm                   478 non-null    int64
 19  height_mm                  478 non-null    int64
 20  car_body_type              478 non-null    object
 21  source_url                 478 non-null    object
dtypes: float64(6), int64(7), object(9)
memory usage: 82.3+ KB
```

```
df.describe()
```

	top_speed_kmh	battery_capacity_kWh	number_of_cells	torque_nm	efficiency_wh_per_km	range_km	acceleration_0_100_s	fast_charging_pow
count	478.000000	478.000000	276.000000	471.000000	478.000000	478.000000	478.000000	4
mean	185.487448	74.043724	485.293478	498.012739	162.903766	393.179916	6.882636	1
std	34.252773	20.331058	1210.819733	241.461128	34.317532	103.287335	2.730696	
min	125.000000	21.300000	72.000000	113.000000	109.000000	135.000000	2.200000	
25%	160.000000	60.000000	150.000000	305.000000	143.000000	320.000000	4.800000	
50%	180.000000	76.150000	216.000000	430.000000	155.000000	397.500000	6.600000	1
75%	201.000000	90.600000	324.000000	679.000000	177.750000	470.000000	8.200000	1
max	325.000000	118.000000	7920.000000	1350.000000	370.000000	685.000000	19.100000	2

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

```
brand                       0
model                       1
top_speed_kmh               0
battery_capacity_kWh        0
battery_type                0
number_of_cells           202
torque_nm                   7
efficiency_wh_per_km        0
range_km                    0
acceleration_0_100_s        0
fast_charging_power_kw_dc   1
fast_charge_port            1
towing_capacity_kg         26
cargo_volume_l              1
seats                       0
drivetrain                  0
segment                     0
length_mm                   0
width_mm                    0
height_mm                   0
car_body_type               0
source_url                  0
dtype: int64
```

Lis of brands

```
# Count brands
df['brand'].unique()
```

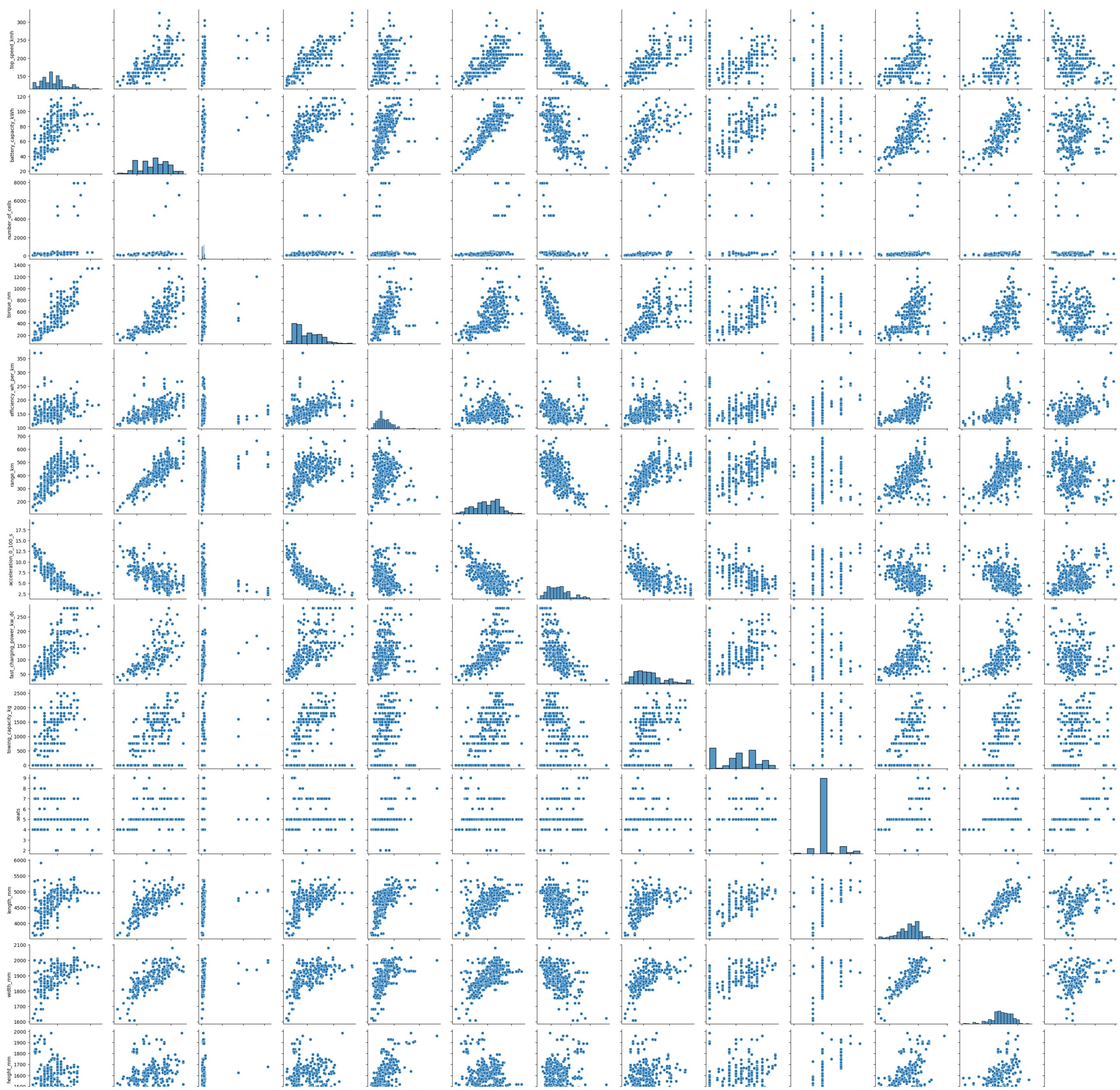
```
array(['Abarth', 'Aiways', 'Alfa', 'Alpine', 'Audi', 'BMW', 'BYD',
       'CUPRA', 'Cadillac', 'Citroen', 'DS', 'Dacia', 'Dongfeng',
       'Elaris', 'Fiat', 'Ford', 'GWM', 'Genesis', 'Honda', 'Hongqi',
       'Hyundai', 'Jaguar', 'Jeep', 'KGM', 'Kia', 'Lancia', 'Leapmotor',
       'Lexus', 'Lotus', 'Lucid', 'Lynk&Co', 'MG', 'Maserati', 'Maxus',
       'Mazda', 'Mercedes-Benz', 'Mini', 'NIO', 'Nissan', 'Omoda', 'Opel',
       'Peugeot', 'Polestar', 'Porsche', 'Renault', 'Rolls-Royce',
       'Skoda', 'Skywell', 'Smart', 'Subaru', 'Tesla', 'Toyota',
       'VinFast', 'Volkswagen', 'Volvo', 'Voyah', 'XPENG', 'Zeekr',
       'firefly'], dtype=object)
```

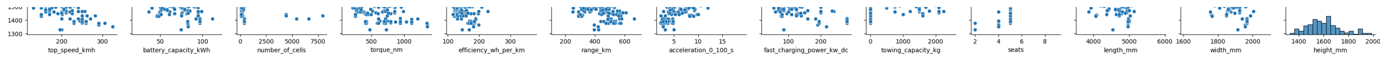
```
numerical_cols = df.select_dtypes(include='number').columns.tolist()
print("numerical_cols: ", numerical_cols)
```

```
sns.pairplot(df,
             vars=numerical_cols)
plt.suptitle("Pairplot of Selected Features", y=1.02)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Pairplot of Selected Features

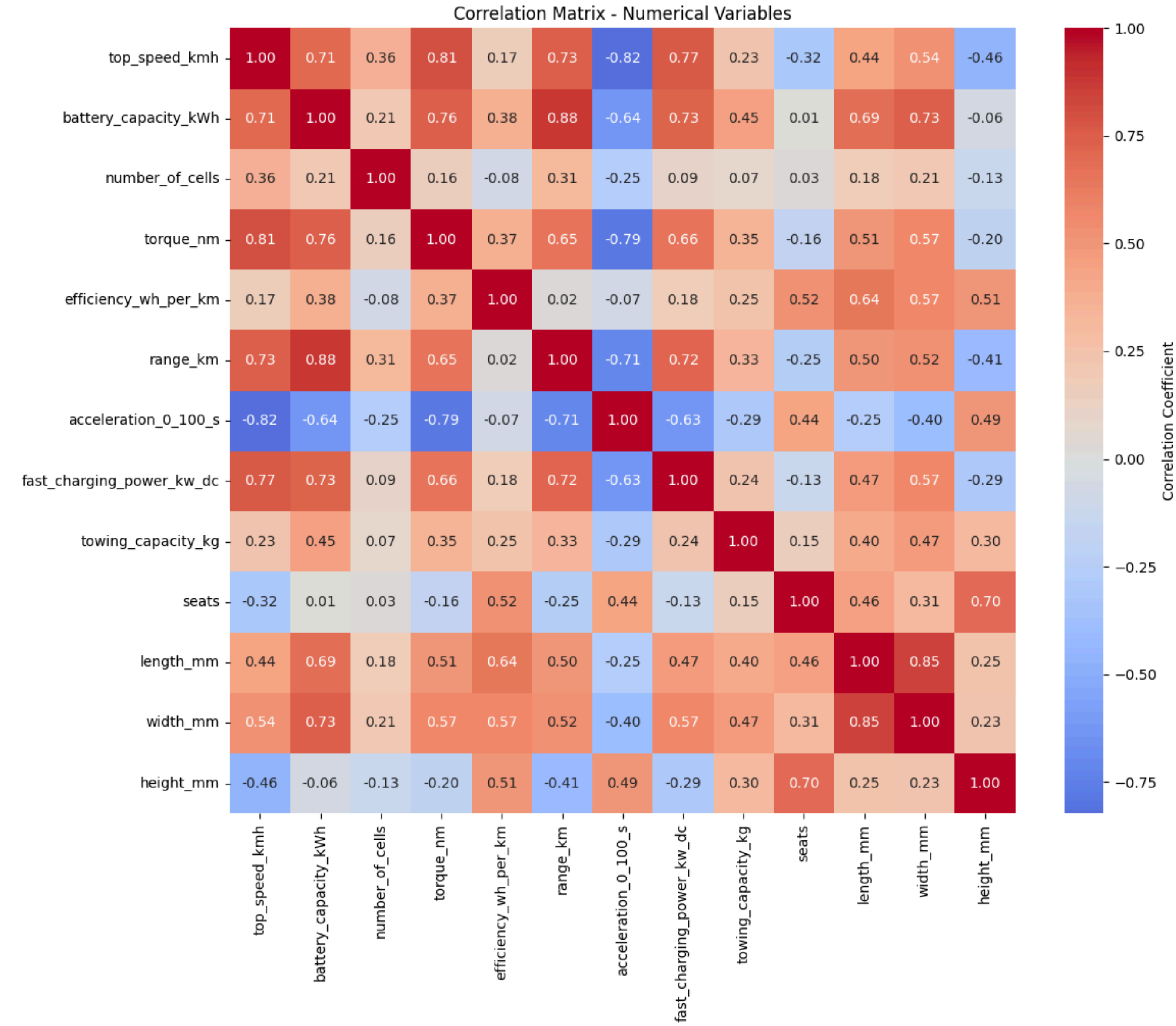




Correlation

```
correlation_matrix = df[numerical_cols].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,
            square=True, fmt='.2f', cbar_kws={'label': 'Correlation Coefficient'})
plt.title('Correlation Matrix - Numerical Variables')
plt.tight_layout()
plt.show()

# Find highly correlated pairs
print("\nHighly correlated variable pairs (|correlation| > 0.7):")
high_corr_pairs = []
for i in range(len(correlation_matrix.columns)):
    for j in range(i+1, len(correlation_matrix.columns)):
        corr_val = correlation_matrix.iloc[i, j]
        if abs(corr_val) > 0.7:
            high_corr_pairs.append((
                correlation_matrix.columns[i],
                correlation_matrix.columns[j],
                corr_val
            ))

if high_corr_pairs:
    for var1, var2, corr in high_corr_pairs:
        print(f"  {var1} <-> {var2}: {corr:.3f}")
```

```
numerical_cols:  ['top_speed_kmh', 'battery_capacity_kWh', 'number_of_cells', 'torque_nm', 'efficiency_wh_per_km', 'range_km', 'acceleration_
/usr/local/lib/python3.11/dist-packages/matplotlib/colors.py:721: RuntimeWarning: invalid value encountered in less
  xa[xa < 0] = -1
```




```
Highly correlated variable pairs (|correlation| > 0.7):
  top_speed_kmh <-> battery_capacity_kWh: 0.708
  top_speed_kmh <-> torque_nm: 0.806
  top_speed_kmh <-> range_km: 0.732
  top_speed_kmh <-> acceleration_0_100_s: -0.823
  top_speed_kmh <-> fast_charging_power_kw_dc: 0.772
  battery_capacity_kWh <-> torque_nm: 0.757
  battery_capacity_kWh <-> range_km: 0.880
  battery_capacity_kWh <-> fast_charging_power_kw_dc: 0.728
  battery_capacity_kWh <-> width_mm: 0.731
  torque_nm <-> acceleration_0_100_s: -0.788
  range_km <-> acceleration_0_100_s: -0.712
  range_km <-> fast_charging_power_kw_dc: 0.721
  length_mm <-> width_mm: 0.850
```

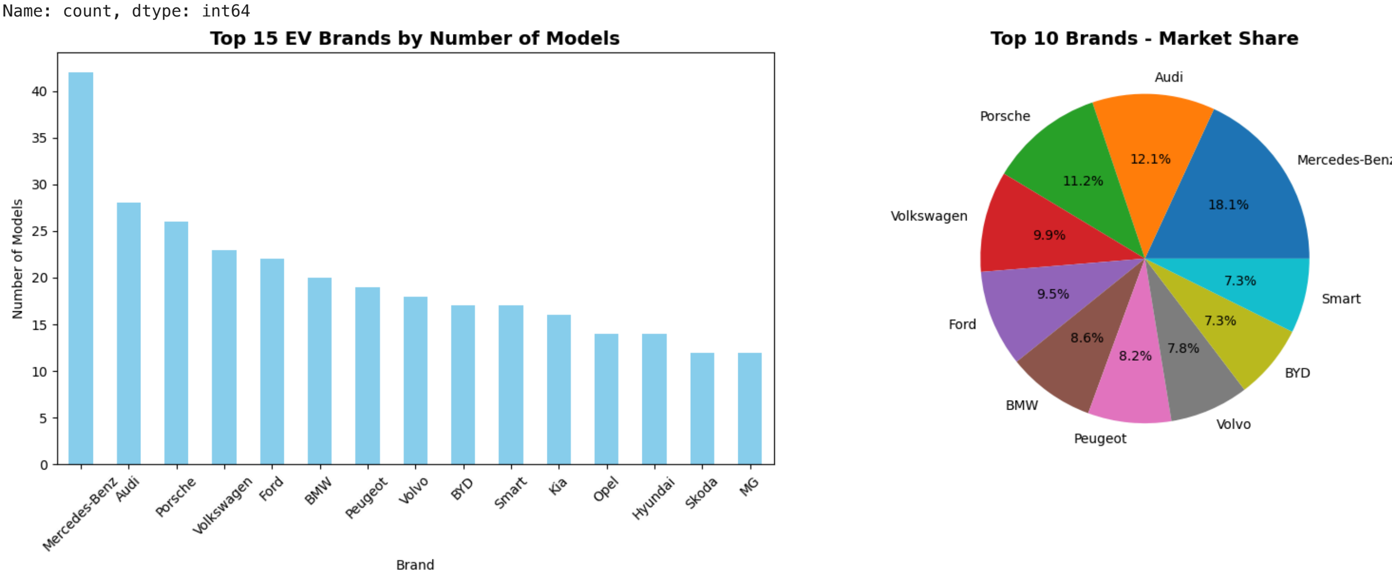
Data visualization

Distribution of Electric Vehicles by Brand

```
brand_counts = df['brand'].value_counts().head(15)
print(f"Top 15 brands by number of models:")
print(brand_counts)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))
```

```
# Bar plot
brand_counts.plot(kind='bar', ax=ax1, color='skyblue')
ax1.set_title('Top 15 EV Brands by Number of Models', fontsize=14, fontweight='bold')
ax1.set_xlabel('Brand')
ax1.set_ylabel('Number of Models')
ax1.tick_params(axis='x', rotation=45)

# Pie chart for top 10
brand_counts.head(10).plot(kind='pie', ax=ax2, autopct='%1.1f%%')
ax2.set_title('Top 10 Brands - Market Share', fontsize=14, fontweight='bold')
ax2.set_ylabel('')

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

```
Top 15 brands by number of models:
brand
Mercedes-Benz    42
Audi             28
Porsche          26
Volkswagen       23
Ford             22
BMW              20
Peugeot          19
Volvo            18
BYD              17
Smart            17
Kia              16
Opel             14
Hyundai          14
Skoda            12
MG               12
Name: count, dtype: int64
```

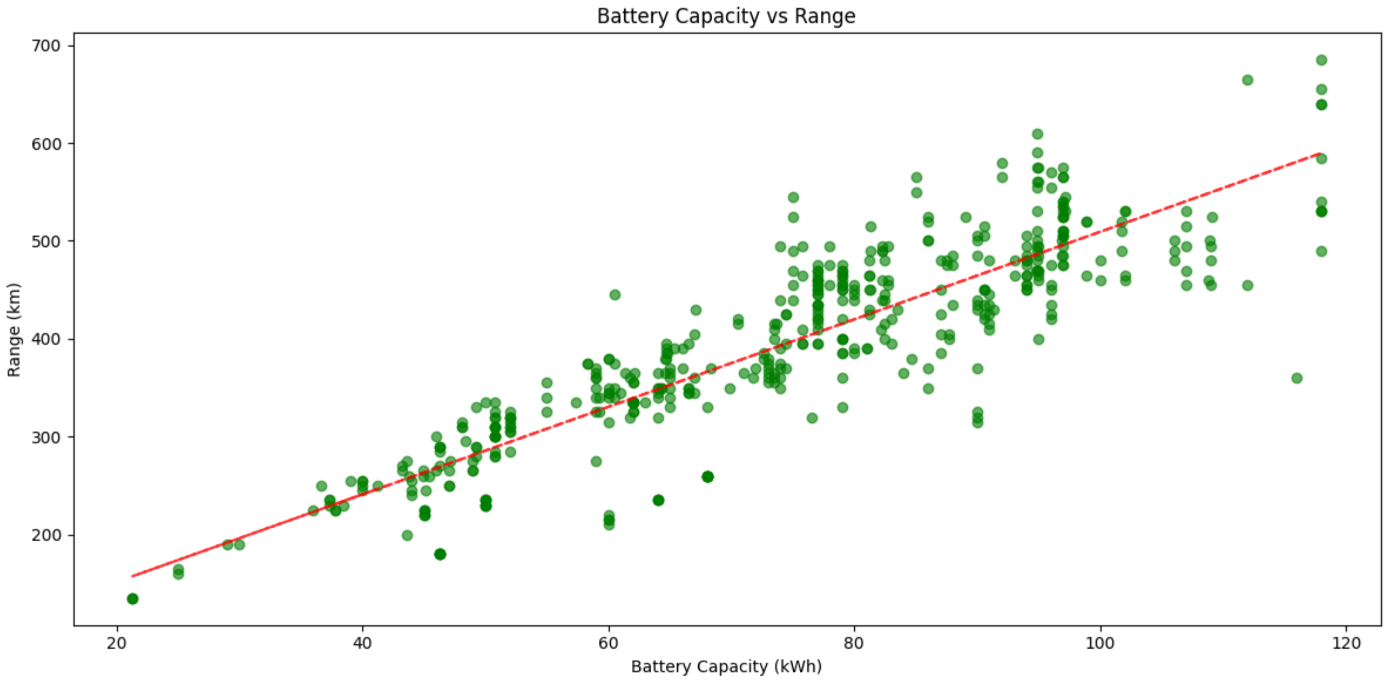


Battery Capacity vs Driving Range Correlation

```
# Remove outliers for better visualization
df_clean = df.dropna(subset=['battery_capacity_kWh', 'range_km'])
df_clean = df_clean[(df_clean['battery_capacity_kWh'] < 200) & (df_clean['range_km'] < 1000)]

correlation = df_clean['battery_capacity_kWh'].corr(df_clean['range_km'])
print(f"Correlation coefficient: {correlation:.3f}")

plt.figure(figsize=(12, 6))

plt.scatter(df_clean['battery_capacity_kWh'], df_clean['range_km'], alpha=0.6, color='green')
plt.xlabel('Battery Capacity (kWh)')
plt.ylabel('Range (km)')
plt.title('Battery Capacity vs Range')
z = np.polyfit(df_clean['battery_capacity_kWh'], df_clean['range_km'], 1)
p = np.poly1d(z)
plt.plot(df_clean['battery_capacity_kWh'], p(df_clean['battery_capacity_kWh']), "r--", alpha=0.8)

plt.tight_layout()
```

```
plt.show()
```

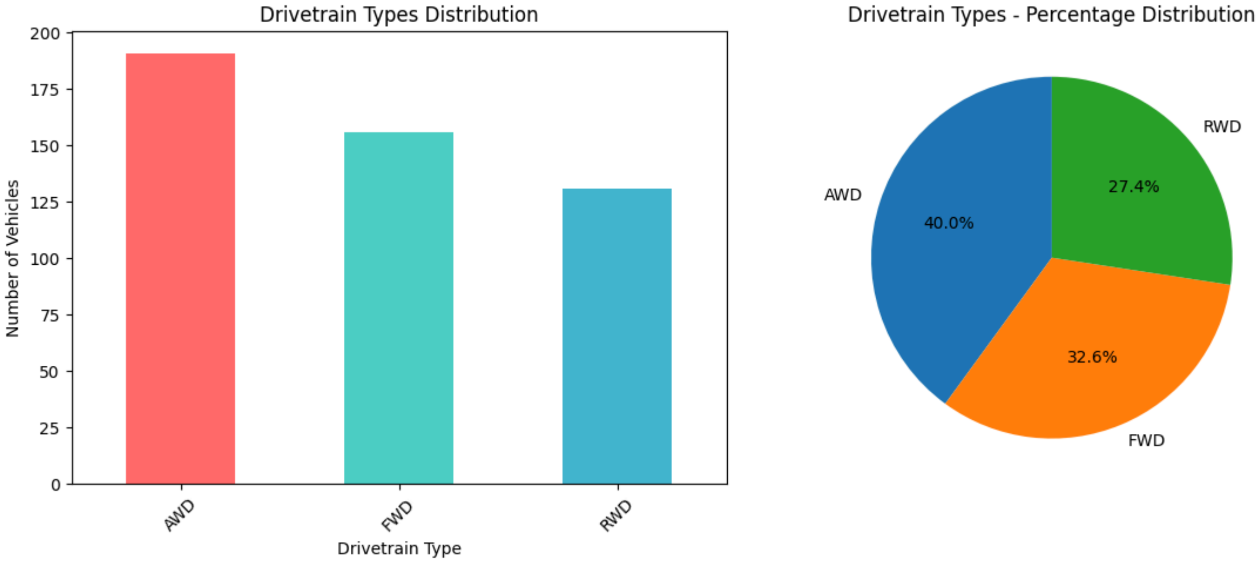
```
Correlation coefficient: 0.880
```



What are the most common drivetrain types?

```
drivetrain_counts = df['drivetrain'].value_counts()
print("Drivetrain distribution:")
print(drivetrain_counts)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
drivetrain_counts.plot(kind='bar', color=['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4'])
plt.title('Drivetrain Types Distribution')
plt.xlabel('Drivetrain Type')
plt.ylabel('Number of Vehicles')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
plt.pie(drivetrain_counts.values, labels=drivetrain_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Drivetrain Types - Percentage Distribution')

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

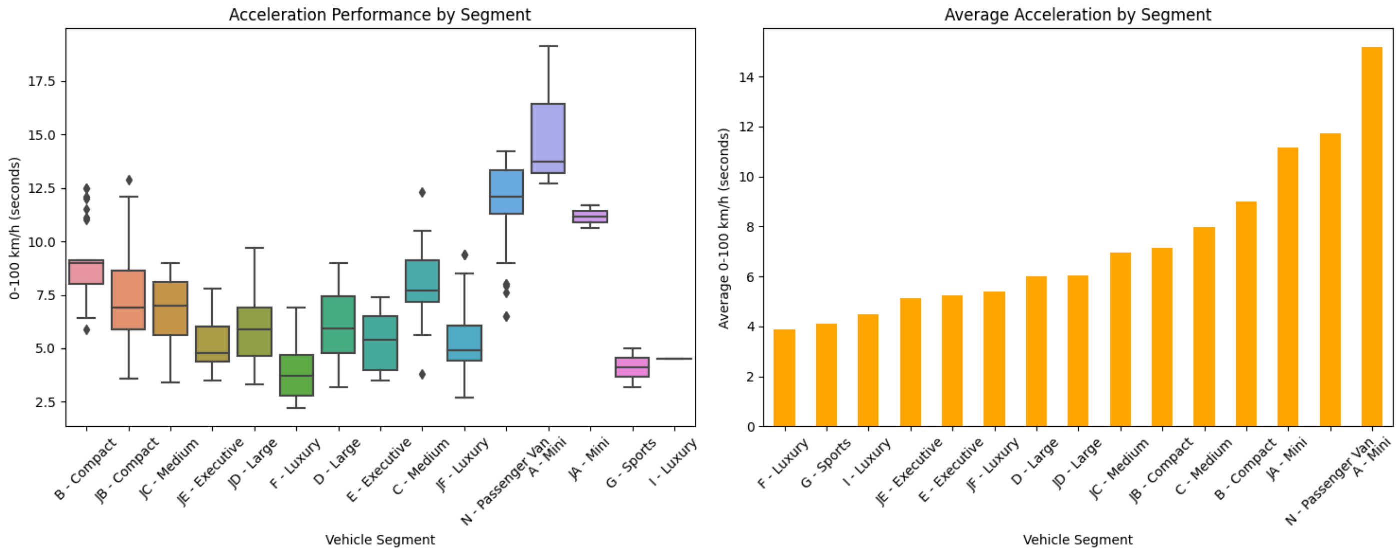
```
Drivetrain distribution:
drivetrain
AWD    191
FWD    156
RWD    131
Name: count, dtype: int64
```



How does acceleration performance vary by segment?

```
df_accel = df.dropna(subset=['acceleration_0_100_s', 'segment'])
df_accel = df_accel[df_accel['acceleration_0_100_s'] < 20]  # Remove outliers

print("Average acceleration by segment:")
segment_accel = df_accel.groupby('segment')['acceleration_0_100_s'].agg(['mean', 'count']).round(2)
print(segment_accel)

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df_accel, x='segment', y='acceleration_0_100_s')
plt.title('Acceleration Performance by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('0-100 km/h (seconds)')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
segment_means = df_accel.groupby('segment')['acceleration_0_100_s'].mean().sort_values()
segment_means.plot(kind='bar', color='orange')
plt.title('Average Acceleration by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('Average 0-100 km/h (seconds)')
plt.xticks(rotation=45)

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

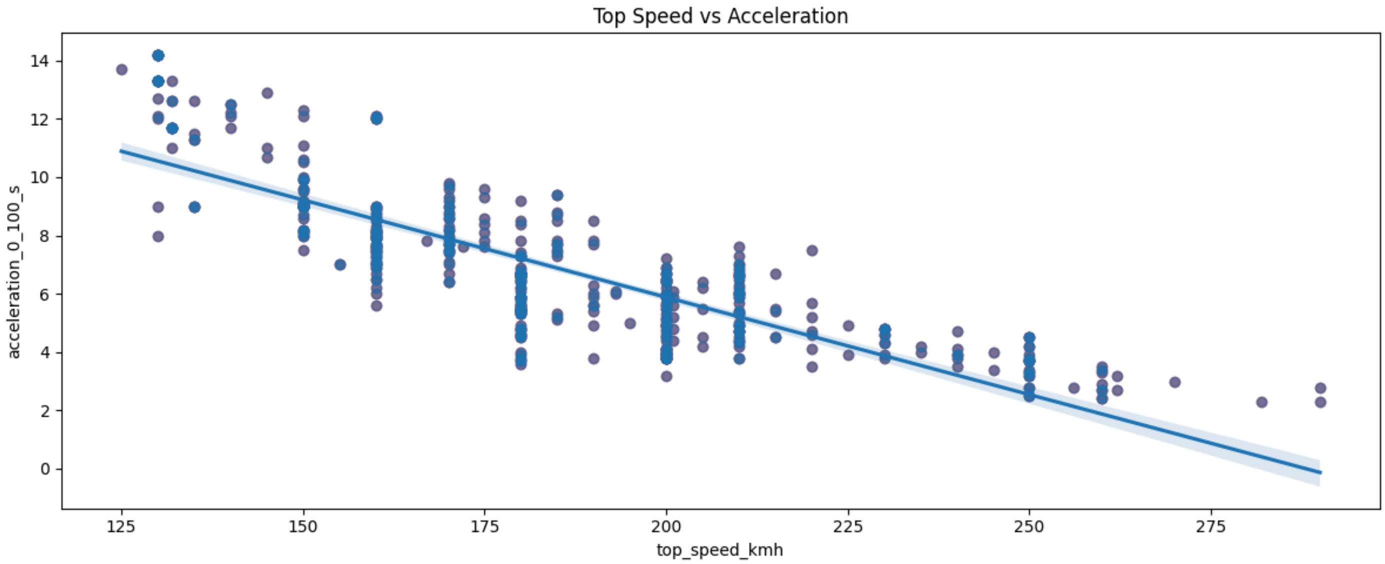
```
Average acceleration by segment:
                     mean  count
segment
A – Mini            15.17      3
B – Compact          8.99     29
C – Medium           7.98     34
D – Large            6.00     28
E – Executive        5.25     30
F – Luxury           3.89     51
G – Sports           4.10      2
I – Luxury           4.50      1
JA – Mini           11.15      2
JB – Compact         7.16     44
JC – Medium          6.94     91
JD – Large           6.03     58
JE – Executive       5.13     28
JF – Luxury          5.41     30
N – Passenger Van   11.74     47
```



What is the relationship between top speed and acceleration?

```
df_perf = df.dropna(subset=['top_speed_kmh', 'acceleration_0_100_s'])
df_perf = df_perf[(df_perf['top_speed_kmh'] < 300) & (df_perf['acceleration_0_100_s'] < 15)]

correlation = df_perf['top_speed_kmh'].corr(df_perf['acceleration_0_100_s'])
print(f"Correlation between top speed and acceleration: {correlation:.3f}")

plt.figure(figsize=(12, 5))

plt.scatter(df_perf['top_speed_kmh'], df_perf['acceleration_0_100_s'], alpha=0.6, color='red')
plt.xlabel('Top Speed (km/h)')
plt.ylabel('0-100 km/h (seconds)')
plt.title('Top Speed vs Acceleration')

sns.regplot(data=df_perf, x='top_speed_kmh', y='acceleration_0_100_s', scatter_kws={'alpha':0.6})

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

```
Correlation between top speed and acceleration: -0.831
```



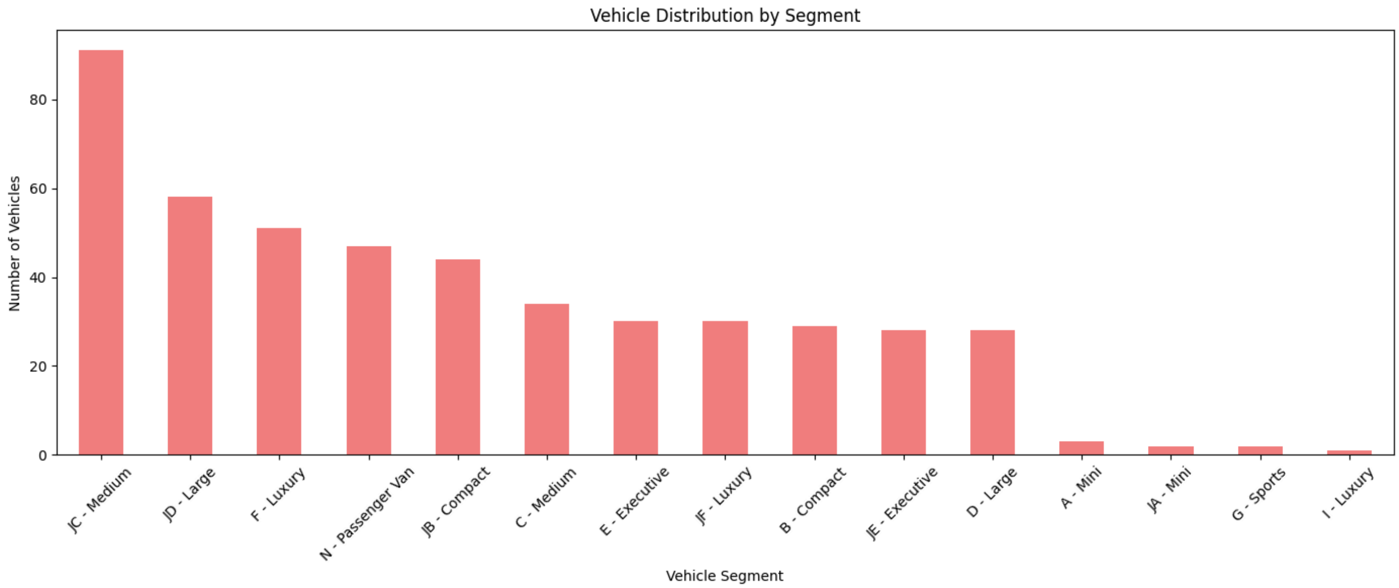
Vehicle Distribution Across Segments

```
segment_counts
```

```
segment
JC - Medium          91
JD - Large           58
F - Luxury           51
N - Passenger Van    47
JB - Compact         44
C - Medium           34
E - Executive        30
JF - Luxury          30
B - Compact          29
JE - Executive       28
D - Large            28
A - Mini              3
JA - Mini             2
G - Sports            2
I - Luxury            1
Name: count, dtype: int64
```

```
segment_counts = df['segment'].value_counts()
print("Segment distribution:")
print(segment_counts)

plt.figure(figsize=(14, 6))

segment_counts.plot(kind='bar', color='lightcoral')
plt.title('Vehicle Distribution by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('Number of Vehicles')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

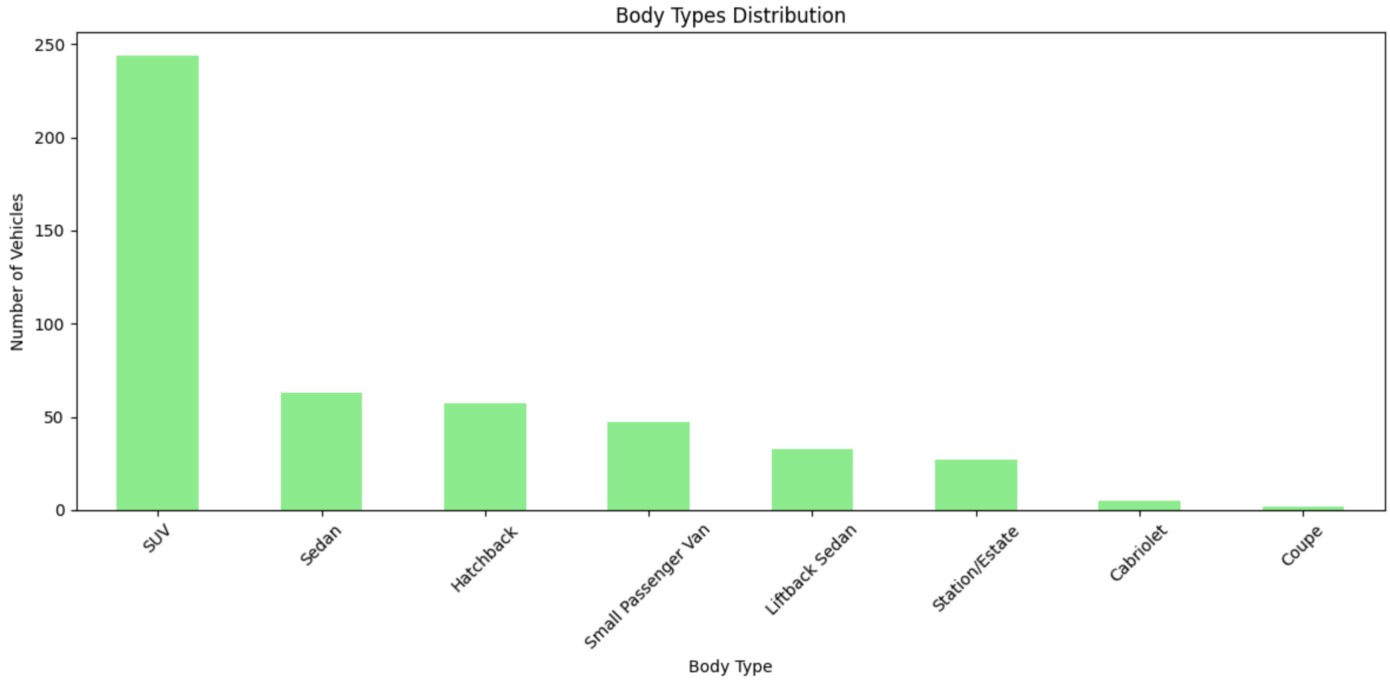
```
Segment distribution:
segment
JC - Medium          91
JD - Large           58
F - Luxury           51
N - Passenger Van    47
JB - Compact         44
C - Medium           34
E - Executive        30
JF - Luxury          30
B - Compact          29
JE - Executive       28
D - Large            28
A - Mini              3
JA - Mini             2
G - Sports            2
I - Luxury            1
Name: count, dtype: int64
```



What body types are most common in the EV market?

```
body_counts = df['car_body_type'].value_counts()
print("Body type distribution:")
print(body_counts)

plt.figure(figsize=(12, 6))

body_counts.plot(kind='bar', color='lightgreen')
plt.title('Body Types Distribution')
plt.xlabel('Body Type')
plt.ylabel('Number of Vehicles')
plt.xticks(rotation=45)


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

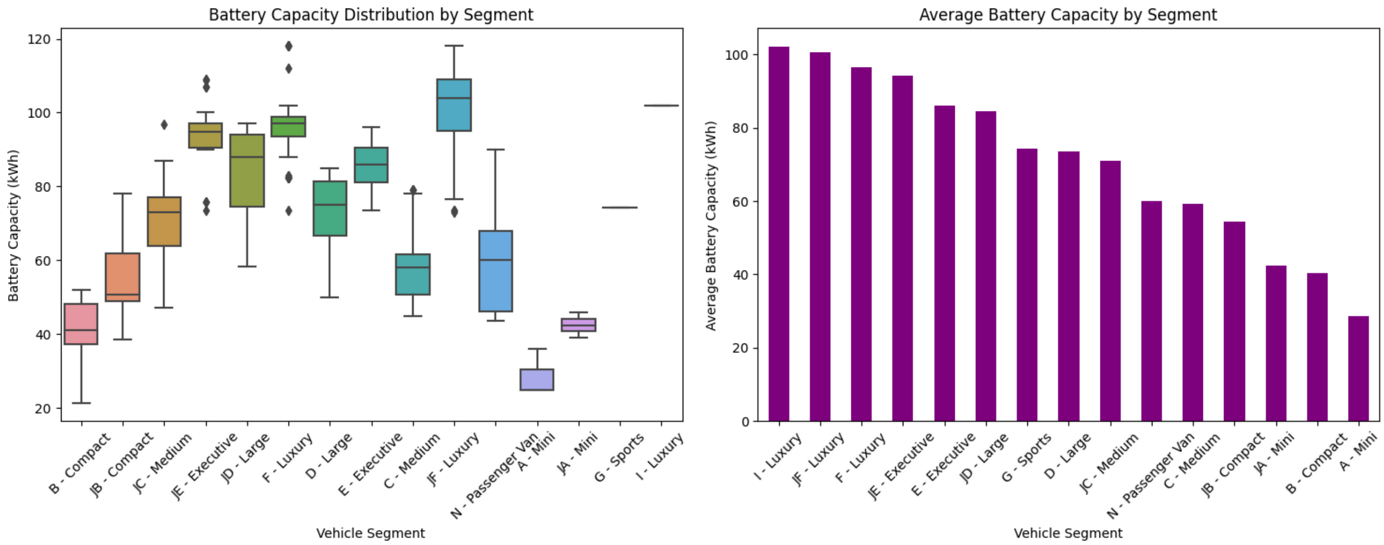
```
Body type distribution:
car_body_type
SUV                    244
Sedan                   63
Hatchback               57
Small Passenger Van     47
Liftback Sedan          33
Station/Estate          27
Cabriolet                5
Coupe                    2
Name: count, dtype: int64
```



How does battery capacity vary by vehicle segment?

```
df_battery = df.dropna(subset=['battery_capacity_kWh', 'segment'])
df_battery = df_battery[df_battery['battery_capacity_kWh'] < 200]  # Remove outliers

print("Battery capacity statistics by segment:")
battery_stats = df_battery.groupby('segment')['battery_capacity_kWh'].agg(['mean', 'median', 'std', 'count']).round(2)
print(battery_stats)

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df_battery, x='segment', y='battery_capacity_kWh')
plt.title('Battery Capacity Distribution by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('Battery Capacity (kWh)')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
segment_battery_mean = df_battery.groupby('segment')['battery_capacity_kWh'].mean().sort_values(ascending=False)
segment_battery_mean.plot(kind='bar', color='purple')
plt.title('Average Battery Capacity by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('Average Battery Capacity (kWh)')
plt.xticks(rotation=45)

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

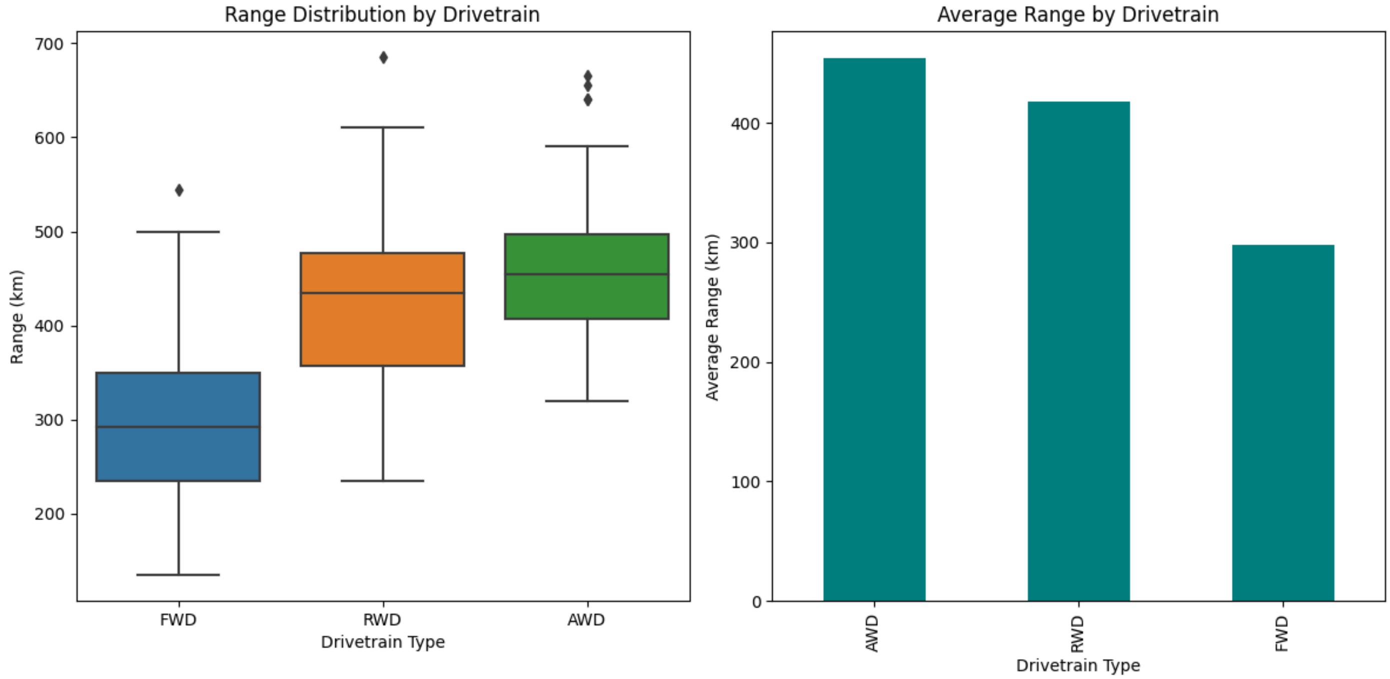
```
Battery capacity statistics by segment:
                    mean  median    std  count
segment
A - Mini           28.67   25.00   6.35      3
B - Compact        40.47   41.20   8.94     29
C - Medium         59.14   58.20  11.62     34
D - Large          73.43   75.00   9.60     28
E - Executive      85.92   86.00   6.41     30
F - Luxury         96.59   97.00   9.97     51
G - Sports         74.40   74.40   0.00      2
I - Luxury        102.00  102.00    NaN      1
JA - Mini          42.50   42.50   4.95      2
JB - Compact       54.43   50.80   8.84     44
JC - Medium        70.85   73.00   9.56     91
JD - Large         84.58   87.85  10.81     58
JE - Executive     94.23   94.90  10.01     28
JF - Luxury       100.65  104.00  13.85     30
N - Passenger Van  60.01   60.00  14.49     47
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1458: RuntimeWarning: invalid value encountered in greater
  has_large_values = (abs_vals > 1e6).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning: invalid value encountered in less
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
/usr/local/lib/python3.11/dist-packages/pandas/io/formats/format.py:1459: RuntimeWarning: invalid value encountered in greater
  has_small_values = ((abs_vals < 10 ** (-self.digits)) & (abs_vals > 0)).any()
```



What is the average range by drivetrain type?

```
df_range = df.dropna(subset=['range_km', 'drivetrain'])
df_range = df_range[df_range['range_km'] < 1000]  # Remove outliers

print("Range statistics by drivetrain:")
range_stats = df_range.groupby('drivetrain')['range_km'].agg(['mean', 'median', 'std', 'count']).round(2)
print(range_stats)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df_range, x='drivetrain', y='range_km')
plt.title('Range Distribution by Drivetrain')
plt.xlabel('Drivetrain Type')
plt.ylabel('Range (km)')

plt.subplot(1, 2, 2)
drivetrain_range_mean = df_range.groupby('drivetrain')['range_km'].mean().sort_values(ascending=False)
drivetrain_range_mean.plot(kind='bar', color='teal')
plt.title('Average Range by Drivetrain')
plt.xlabel('Drivetrain Type')
plt.ylabel('Average Range (km)')

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

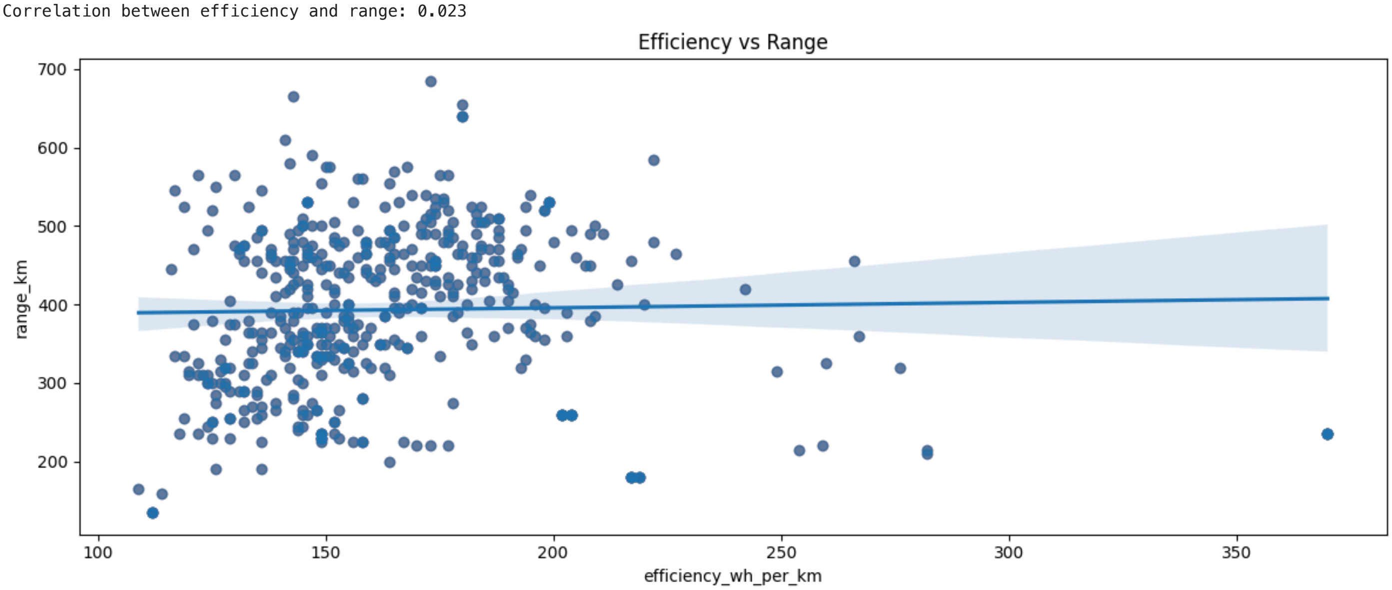
```
Range statistics by drivetrain:
              mean  median    std  count
drivetrain
AWD         453.77   455.0  68.33    191
FWD         298.01   292.5  78.79    156
RWD         418.17   435.0  89.60    131
```



How does vehicle efficiency correlate with range?

```
df_eff = df.dropna(subset=['efficiency_wh_per_km', 'range_km'])
df_eff = df_eff[(df_eff['efficiency_wh_per_km'] < 500) & (df_eff['range_km'] < 1000)]

correlation = df_eff['efficiency_wh_per_km'].corr(df_eff['range_km'])
print(f"Correlation between efficiency and range: {correlation:.3f}")

plt.figure(figsize=(12, 5))

plt.scatter(df_eff['efficiency_wh_per_km'], df_eff['range_km'], alpha=0.6, color='brown')
plt.xlabel('Efficiency (Wh/km)')
plt.ylabel('Range (km)')
plt.title('Efficiency vs Range')

sns.regplot(data=df_eff, x='efficiency_wh_per_km', y='range_km', scatter_kws={'alpha':0.6})

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

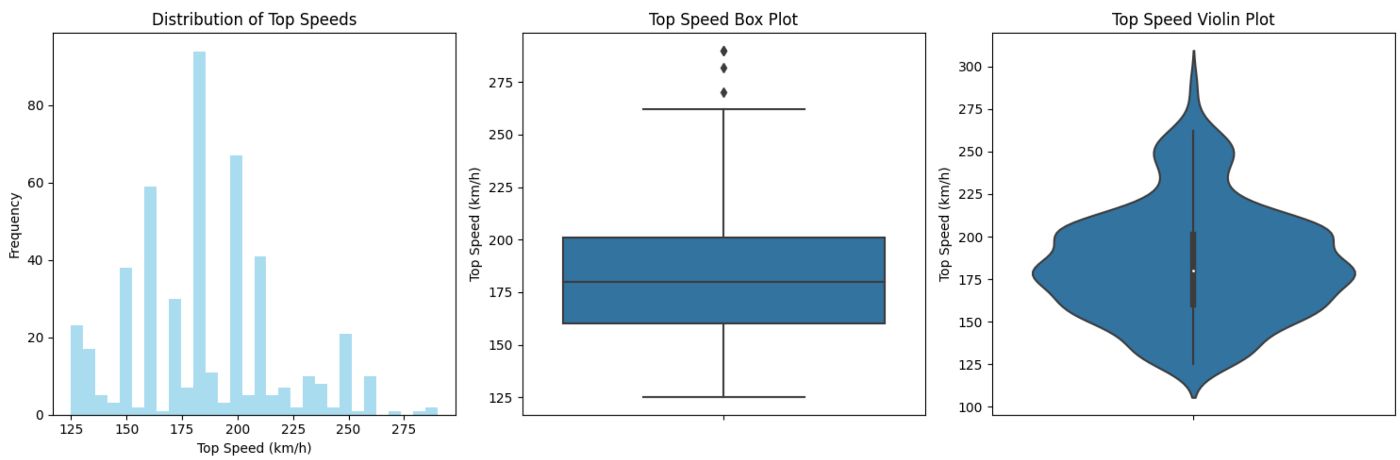
```
Correlation between efficiency and range: 0.023
```



What is the distribution of top speeds?

```
df_speed = df.dropna(subset=['top_speed_kmh'])
df_speed = df_speed[df_speed['top_speed_kmh'] < 300]  # Remove outliers

print(f"Top speed statistics:")
print(f"Mean: {df_speed['top_speed_kmh'].mean():.1f} km/h")
print(f"Median: {df_speed['top_speed_kmh'].median():.1f} km/h")
print(f"Standard deviation: {df_speed['top_speed_kmh'].std():.1f} km/h")

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.hist(df_speed['top_speed_kmh'], bins=30, color='skyblue', alpha=0.7)
plt.xlabel('Top Speed (km/h)')
plt.ylabel('Frequency')
plt.title('Distribution of Top Speeds')

plt.subplot(1, 3, 2)
sns.boxplot(y=df_speed['top_speed_kmh'])
plt.ylabel('Top Speed (km/h)')
plt.title('Top Speed Box Plot')

plt.subplot(1, 3, 3)
sns.violinplot(y=df_speed['top_speed_kmh'])
plt.ylabel('Top Speed (km/h)')
plt.title('Top Speed Violin Plot')

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

```
Top speed statistics:
Mean: 184.9 km/h
Median: 180.0 km/h
Standard deviation: 33.3 km/h
```

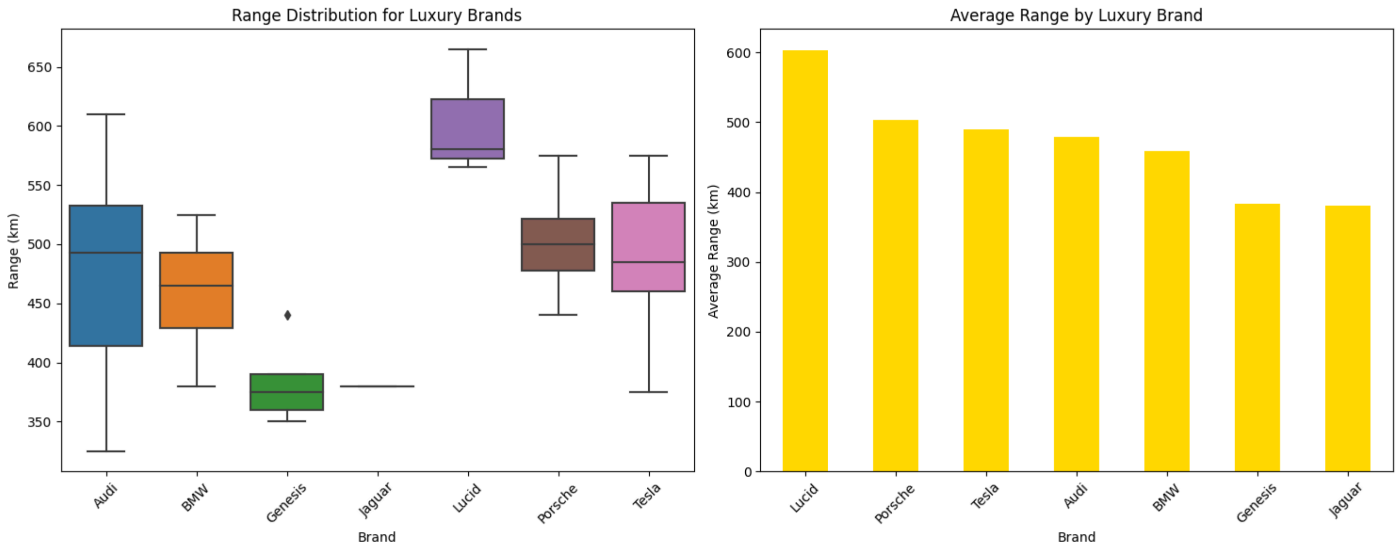


How do luxury brands compare in terms of range?

```
luxury_brands = ['Tesla', 'Mercedes', 'BMW', 'Audi', 'Porsche', 'Jaguar', 'Lucid', 'Rivian', 'Genesis']
df_luxury = df[df['brand'].isin(luxury_brands) & df['range_km'].notna()]
df_luxury = df_luxury[df_luxury['range_km'] < 1000]

print("Range statistics for luxury brands:")
luxury_range = df_luxury.groupby('brand')['range_km'].agg(['mean', 'median', 'count']).round(2)
print(luxury_range)

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df_luxury, x='brand', y='range_km')
plt.title('Range Distribution for Luxury Brands')
plt.xlabel('Brand')
plt.ylabel('Range (km)')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
luxury_range_mean = df_luxury.groupby('brand')['range_km'].mean().sort_values(ascending=False)
```

```
luxury_range_mean.plot(kind='bar', color='gold')
plt.title('Average Range by Luxury Brand')
plt.xlabel('Brand')
plt.ylabel('Average Range (km)')
plt.xticks(rotation=45)

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

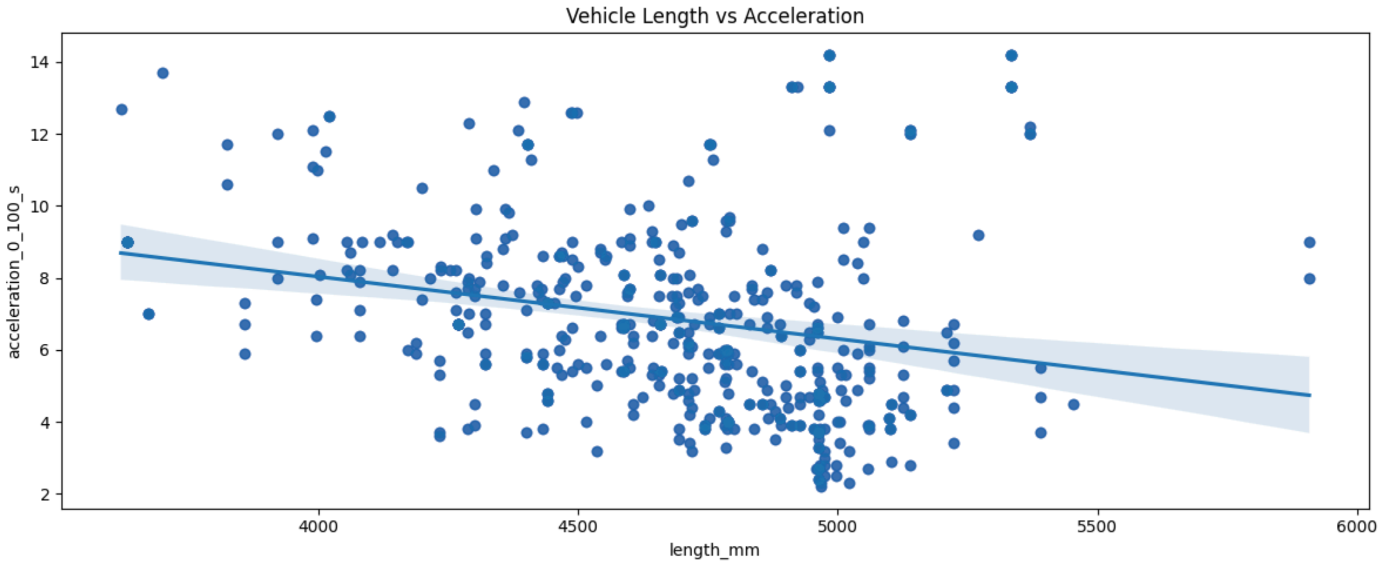
```
Range statistics for luxury brands:
           mean  median  count
brand
Audi     478.39   492.5     28
BMW      458.25   465.0     20
Genesis  383.00   375.0      5
Jaguar   380.00   380.0      1
Lucid    603.33   580.0      3
Porsche  502.88   500.0     26
Tesla    490.00   485.0     11
```



Relationship between vehicle dimensions and acceleration

```
df_dim = df.dropna(subset=['length_mm', 'acceleration_0_100_s'])
df_dim = df_dim[df_dim['acceleration_0_100_s'] < 15]

correlation = df_dim['length_mm'].corr(df_dim['acceleration_0_100_s'])
print(f"Correlation between length and acceleration: {correlation:.3f}")

plt.figure(figsize=(12, 5))

plt.scatter(df_dim['length_mm'], df_dim['acceleration_0_100_s'], alpha=0.6, color='navy')
plt.xlabel('Vehicle Length (mm)')
plt.ylabel('0-100 km/h (seconds)')
plt.title('Vehicle Length vs Acceleration')

sns.regplot(data=df_dim, x='length_mm', y='acceleration_0_100_s', scatter_kws={'alpha':0.6})

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

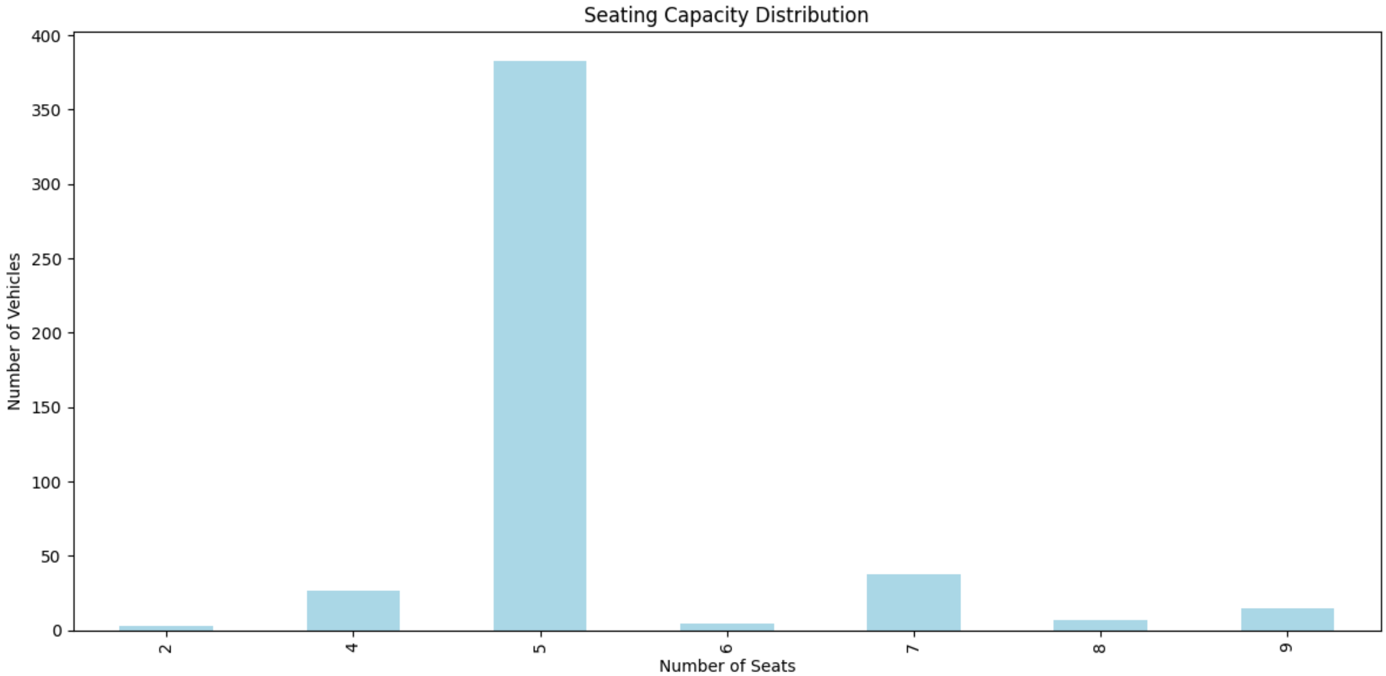
Correlation between length and acceleration: -0.237



Seating capacity distribution

```
df_seats = df.dropna(subset=['seats'])
df_seats = df_seats[df_seats['seats'] <= 10]  # Remove outliers

seat_counts = df_seats['seats'].value_counts().sort_index()
print("Seating capacity distribution:")
print(seat_counts)

plt.figure(figsize=(12, 6))

seat_counts.plot(kind='bar', color='lightblue')
plt.title('Seating Capacity Distribution')
plt.xlabel('Number of Seats')
plt.ylabel('Number of Vehicles')

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

```
Seating capacity distribution:
seats
2      3
4     27
5    383
6      5
7     38
8      7
9     15
Name: count, dtype: int64
```



Towing capacity by segment

```
df_tow = df.dropna(subset=['towing_capacity_kg', 'segment'])
df_tow = df_tow[df_tow['towing_capacity_kg'] > 0]  # Only vehicles with towing capacity

print("Towing capacity statistics by segment:")
towing_stats = df_tow.groupby('segment')['towing_capacity_kg'].agg(['mean', 'median', 'count']).round(2)
print(towing_stats)

plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.boxplot(data=df_tow, x='segment', y='towing_capacity_kg')
plt.title('Towing Capacity by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('Towing Capacity (kg)')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
towing_mean = df_tow.groupby('segment')['towing_capacity_kg'].mean().sort_values(ascending=False)
towing_mean.plot(kind='bar', color='red')
plt.title('Average Towing Capacity by Segment')
plt.xlabel('Vehicle Segment')
plt.ylabel('Average Towing Capacity (kg)')
plt.xticks(rotation=45)

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