In [3]:
```
import pandas as pd
```

In [11]:
```
data = pd.read_excel(r"C:\Users\Ipsita\Desktop\FEV-data-Excel.xlsx")
```

In [16]:
```
data
```

Out[16]:

	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]	Maxim l capa
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	438	...	3130.0	6
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	340	...	3040.0	6
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	364	...	3130.0	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	6
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	6
5	Audi e-tron Sportback S quattro	Audi	e-tron Sportback S quattro	426200	503	973	disc (front + rear)	4WD	95.0	369	...	3130.0	5
6	BMW i3	BMW	i3	169700	170	250	disc (front + rear)	2WD (rear)	42.2	359	...	1730.0	4
7	BMW i3s	BMW	i3s	184200	184	270	disc (front + rear)	2WD (rear)	42.2	345	...	1730.0	4
8	BMW iX3	BMW	iX3	282900	286	400	disc (front + rear)	2WD (rear)	80.0	460	...	2725.0	5
9	Citroën ë-C4	Citroën	ë-C4	125000	136	260	disc (front + rear)	2WD (front)	50.0	350	...	2000.0	4
10	DS DS3 Crossback e-tense	DS	DS3 Crossback e-tense	159900	136	260	disc (front + rear)	2WD (front)	50.0	320	...	1975.0	4
11	Honda e	Honda	e	152900	136	315	disc (front + rear)	2WD (rear)	35.5	222	...	1855.0	3
12	Honda e Advance	Honda	e Advance	165900	154	315	disc (front + rear)	2WD (rear)	35.5	222	...	1870.0	3
13	Hyundai Ioniq electric	Hyundai	Ioniq electric	184500	136	295	disc (front + rear)	2WD (front)	38.3	311	...	1970.0	5
14	Hyundai Kona electric 39.2kWh	Hyundai	Kona electric 39.2kWh	154400	136	395	disc (front + rear)	2WD (front)	39.2	289	...	2020.0	4
15	Hyundai Kona electric 64kWh	Hyundai	Kona electric 64kWh	178400	204	395	disc (front + rear)	2WD (front)	64.0	449	...	2170.0	4
16	Jaguar I-Pace	Jaguar	I-Pace	359500	400	696	disc (front + rear)	4WD	90.0	470	...	2670.0	5
17	Kia e-Niro 39.2kWh	Kia	e-Niro 39.2kWh	146990	136	395	disc (front + rear)	2WD (front)	39.2	289	...	2080.0	4
18	Kia e-Niro 64kWh	Kia	e-Niro 64kWh	167990	204	395	disc (front + rear)	2WD (front)	64.0	455	...	2230.0	4
19	Kia e-Soul 39.2kWh	Kia	e-Soul 39.2kWh	139900	136	395	disc (front + rear)	2WD (front)	39.2	276	...	1682.0	4
20	Kia e-Soul	Kia	e-Soul 64kWh	160990	204	395	disc (front +	2WD	64.0	452	...	1682.0	4

	64kWh						rear)	(front)				
21	Mazda MX-30	Mazda	MX-30	142900	145	270	disc (front + rear)	2WD (front)	35.5	200	...	2119.0	4
22	Mercedes-Benz EQC	Mercedes-Benz	EQC	334700	408	760	disc (front + rear)	4WD	80.0	414	...	2940.0	4
23	Mini Cooper SE	Mini	Cooper SE	139900	184	270	disc (front + rear)	2WD (front)	28.9	234	...	1770.0	4
24	Nissan Leaf	Nissan	Leaf	122900	150	320	disc (front + rear)	2WD (front)	40.0	270	...	1995.0	4
25	Nissan Leaf e+	Nissan	Leaf e+	164000	217	340	disc (front + rear)	2WD (front)	62.0	385	...	2140.0	4
26	Opel Corsa-e	Opel	Corsa-e	128900	136	260	disc (front + rear)	2WD (front)	50.0	337	...	1916.0	3
27	Opel Mokka-e	Opel	Mokka-e	139900	136	260	disc (front + rear)	2WD (front)	50.0	324	...	2015.0	4
28	Peugeot e-208	Peugeot	e-208	124900	136	260	disc (front + rear)	2WD (front)	50.0	340	...	1918.0	4
29	Peugeot e-2008	Peugeot	e-2008	149400	136	260	disc (front + rear)	2WD (front)	50.0	320	...	NaN	N
30	Porsche Taycan 4S (Performance)	Porsche	Taycan 4S (Performance)	457000	435	640	disc (front + rear)	4WD	79.2	407	...	2880.0	7
31	Porsche Taycan 4S (Performance Plus)	Porsche	Taycan 4S (Performance Plus)	482283	490	650	disc (front + rear)	4WD	93.4	463	...	2880.0	6
32	Porsche Taycan Turbo	Porsche	Taycan Turbo	653000	625	850	disc (front + rear)	4WD	93.4	450	...	2880.0	5
33	Porsche Taycan Turbo S	Porsche	Taycan Turbo S	794000	625	1050	disc (front + rear)	4WD	93.4	412	...	2870.0	5
34	Renault Zoe R110	Renault	Zoe R110	135900	108	225	disc (front + rear)	2WD (front)	52.0	395	...	1988.0	4
35	Renault Zoe R135	Renault	Zoe R135	142900	135	245	disc (front + rear)	2WD (front)	52.0	395	...	1988.0	4
36	Skoda Citigo-e iV	Skoda	Citigo-e iV	82050	83	212	disc (front) + drum (rear)	2WD (front)	36.8	260	...	1530.0	3
37	Smart fortwo EQ	Smart	fortwo EQ	96900	82	160	disc (front) + drum (rear)	2WD (rear)	17.6	154	...	1310.0	2
38	Smart forfour EQ	Smart	forfour EQ	98900	82	160	disc (front) + drum (rear)	2WD (rear)	17.6	148	...	1570.0	4
39	Tesla Model 3 Standard Range Plus	Tesla	Model 3 Standard Range Plus	195490	285	450	disc (front + rear)	2WD (rear)	54.0	430	...	NaN	N
40	Tesla Model 3 Long Range	Tesla	Model 3 Long Range	235490	372	510	disc (front + rear)	4WD	75.0	580	...	NaN	N
41	Tesla Model 3 Performance	Tesla	Model 3 Performance	260490	480	639	disc (front + rear)	4WD	75.0	567	...	NaN	N
42	Tesla Model S Long Range Plus	Tesla	Model S Long Range Plus	368990	525	755	disc (front + rear)	4WD	100.0	652	...	NaN	N
43	Tesla Model S Performance	Tesla	Model S Performance	443990	772	1140	disc (front + rear)	4WD	100.0	639	...	NaN	N

44	Tesla Model X Long Range Plus	Tesla	Model X Long Range Plus	407990	525	755	disc (front + rear)	4WD	100.0	561	...	NaN
45	Tesla Model X Performance	Tesla	Model X Performance	482990	772	1140	disc (front + rear)	4WD	100.0	548	...	NaN
46	Volkswagen e-up!	Volkswagen	e-up!	97990	83	210	disc (front) + drum (rear)	2WD (front)	32.3	258	...	1530.0
47	Volkswagen ID.3 Pro Performance	Volkswagen	ID.3 Pro Performance	155890	204	310	disc (front) + drum (rear)	2WD (rear)	58.0	425	...	2270.0
48	Volkswagen ID.3 Pro S	Volkswagen	ID.3 Pro S	179990	204	310	disc (front) + drum (rear)	2WD (rear)	77.0	549	...	2280.0
49	Volkswagen ID.4 1st	Volkswagen	ID.4 1st	202390	204	310	disc (front) + drum (rear)	2WD (rear)	77.0	500	...	2660.0
50	Citroën ë-Spacetourer (M)	Citroën	ë-Spacetourer (M)	215400	136	260	disc (front + rear)	2WD (front)	50.0	230	...	2810.0
51	Mercedes-Benz EQV (long)	Mercedes-Benz	EQV (long)	339480	204	362	NaN	2WD (front)	90.0	356	...	3500.0
52	Nissan e-NV200 evalia	Nissan	e-NV200 evalia	164328	109	254	disc (front + rear)	2WD (front)	40.0	200	...	2250.0

53 rows × 25 columns

In [ ]:
```
1.(a) filter out EVs that meet these criteria
```

In [20]:
```
# Sample list of EVs with their prices and ranges
ev_list = [
    {"make": "Tesla Model 3", "price": 240000, "range_km": 491},
    {"make": "Hyundai Ioniq 5", "price": 265000, "range_km": 507},
    {"make": "BMW i4", "price": 360000, "range_km": 520},
    {"make": "Volkswagen ID.4", "price": 210000, "range_km": 410},
    {"make": "Kia EV6", "price": 280000, "range_km": 528},
    {"make": "Renault Megane E-Tech", "price": 190000, "range_km": 450},
]

# Customer criteria
budget = 350000  # PLN
min_range = 400  # km

# Filter EVs based on budget and minimum range
filtered_evs = [ev for ev in ev_list if ev["price"] <= budget and ev["range_km"] >= min_range]

# Display results
for ev in filtered_evs:
    print(f"{ev['make']}: {ev['price']} PLN, {ev['range_km']} km range")
```

```
Tesla Model 3: 240000 PLN, 491 km range
Hyundai Ioniq 5: 265000 PLN, 507 km range
Volkswagen ID.4: 210000 PLN, 410 km range
Kia EV6: 280000 PLN, 528 km range
Renault Megane E-Tech: 190000 PLN, 450 km range
```

In [ ]:
```
(b)  Group them by the manufacture
```

In [22]:
```
from collections import defaultdict

# Sample list of EVs with their prices and ranges
ev_list = [
    {"make": "Tesla Model 3", "manufacturer": "Tesla", "price": 240000, "range_km": 491},
    {"make": "Tesla Model Y", "manufacturer": "Tesla", "price": 310000, "range_km": 455},
    {"make": "Hyundai Ioniq 5", "manufacturer": "Hyundai", "price": 265000, "range_km": 507},
    {"make": "BMW i4", "manufacturer": "BMW", "price": 360000, "range_km": 520},
    {"make": "Volkswagen ID.4", "manufacturer": "Volkswagen", "price": 210000, "range_km": 410},
    {"make": "Kia EV6", "manufacturer": "Kia", "price": 280000, "range_km": 528},
    {"make": "Renault Megane E-Tech", "manufacturer": "Renault", "price": 190000, "range_km": 450},
]
```

```
# Customer criteria
budget = 350000  # PLN
min_range = 400  # km

# Filter and group
grouped_evs = defaultdict(list)

for ev in ev_list:
    if ev["price"] <= budget and ev["range_km"] >= min_range:
        grouped_evs[ev["manufacturer"]].append(ev)

# Display grouped results
for manufacturer, cars in grouped_evs.items():
    print(f"\n{manufacturer}:")
    for car in cars:
        print(f"  {car['make']} - {car['price']} PLN, {car['range_km']} km")
```

```
Tesla:
  Tesla Model 3 - 240000 PLN, 491 km
  Tesla Model Y - 310000 PLN, 455 km

Hyundai:
  Hyundai Ioniq 5 - 265000 PLN, 507 km

Volkswagen:
  Volkswagen ID.4 - 210000 PLN, 410 km

Kia:
  Kia EV6 - 280000 PLN, 528 km

Renault:
  Renault Megane E-Tech - 190000 PLN, 450 km
```

In [ ]:
```
(c) Calculate the average battery capacity for each manufacturer.
```

In [24]:
```
from collections import defaultdict

# Sample list of EVs with battery capacity added
ev_list = [
    {"make": "Tesla Model 3", "manufacturer": "Tesla", "price": 240000, "range_km": 491, "battery_kWh": 60},
    {"make": "Tesla Model Y", "manufacturer": "Tesla", "price": 310000, "range_km": 455, "battery_kWh": 75},
    {"make": "Hyundai Ioniq 5", "manufacturer": "Hyundai", "price": 265000, "range_km": 507, "battery_kWh": 77}
    {"make": "BMW i4", "manufacturer": "BMW", "price": 360000, "range_km": 520, "battery_kWh": 83},
    {"make": "Volkswagen ID.4", "manufacturer": "Volkswagen", "price": 210000, "range_km": 410, "battery_kWh":
    {"make": "Kia EV6", "manufacturer": "Kia", "price": 280000, "range_km": 528, "battery_kWh": 77},
    {"make": "Renault Megane E-Tech", "manufacturer": "Renault", "price": 190000, "range_km": 450, "battery_kWh
]

# Customer criteria
budget = 350000
min_range = 400

# Group battery capacities by manufacturer
battery_by_manufacturer = defaultdict(list)

for ev in ev_list:
    if ev["price"] <= budget and ev["range_km"] >= min_range:
        battery_by_manufacturer[ev["manufacturer"]].append(ev["battery_kWh"])

# Calculate and display average battery capacity
for manufacturer, batteries in battery_by_manufacturer.items():
    avg_capacity = sum(batteries) / len(batteries)
    print(f"{manufacturer}: {avg_capacity:.1f} kWh average battery capacity")
```

```
Tesla: 67.5 kWh average battery capacity
Hyundai: 77.0 kWh average battery capacity
Volkswagen: 77.0 kWh average battery capacity
Kia: 77.0 kWh average battery capacity
Renault: 60.0 kWh average battery capacity
```

2. You suspect some EVs have unusually high or low energy consumption. Find the

outliers in the mean- Energy consumption [kWh/100 km] column.

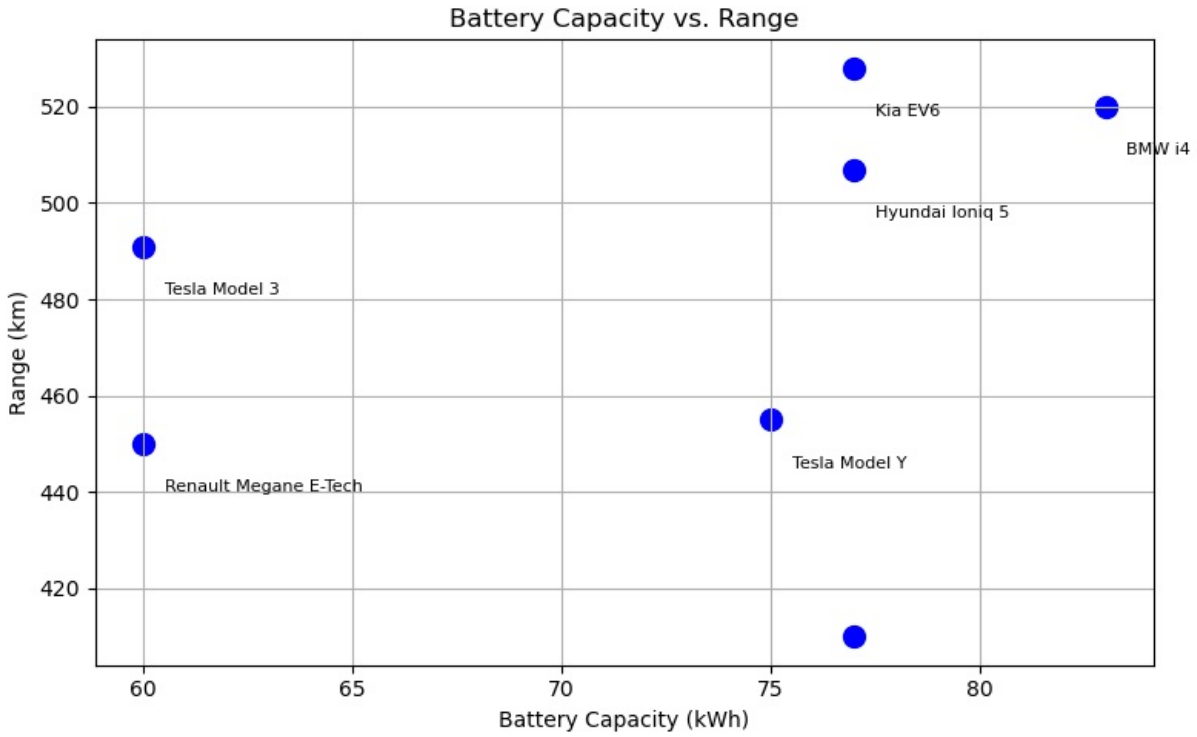
In [26]:
```
import numpy as np

# Sample list of EVs including energy consumption
ev_list = [
    {"make": "Tesla Model 3", "manufacturer": "Tesla", "consumption": 12.2},
    {"make": "Tesla Model Y", "manufacturer": "Tesla", "consumption": 15.5},
    {"make": "Hyundai Ioniq 5", "manufacturer": "Hyundai", "consumption": 17.9},
    {"make": "BMW i4", "manufacturer": "BMW", "consumption": 18.5},
    {"make": "Volkswagen ID.4", "manufacturer": "Volkswagen", "consumption": 19.2},
```

```
    {"make": "Kia EV6", "manufacturer": "Kia", "consumption": 16.8},
    {"make": "Renault Megane E-Tech", "manufacturer": "Renault", "consumption": 15.0},
    {"make": "BYD Tang", "manufacturer": "BYD", "consumption": 22.5},  # Possible high outlier
    {"make": "Mini Electric", "manufacturer": "Mini", "consumption": 11.5},  # Possible low outlier
]

# Extract consumption values
consumptions = [ev["consumption"] for ev in ev_list]

# Compute IQR
q1 = np.percentile(consumptions, 25)
q3 = np.percentile(consumptions, 75)
iqr = q3 - q1

# Define outlier thresholds
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

# Identify outliers
outliers = [ev for ev in ev_list if ev["consumption"] < lower_bound or ev["consumption"] > upper_bound]

# Display results
print("Outliers in energy consumption (kWh/100 km):")
for ev in outliers:
    print(f"{ev['make']} ({ev['consumption']} kWh/100 km)")
```

```
Outliers in energy consumption (kWh/100 km):
```

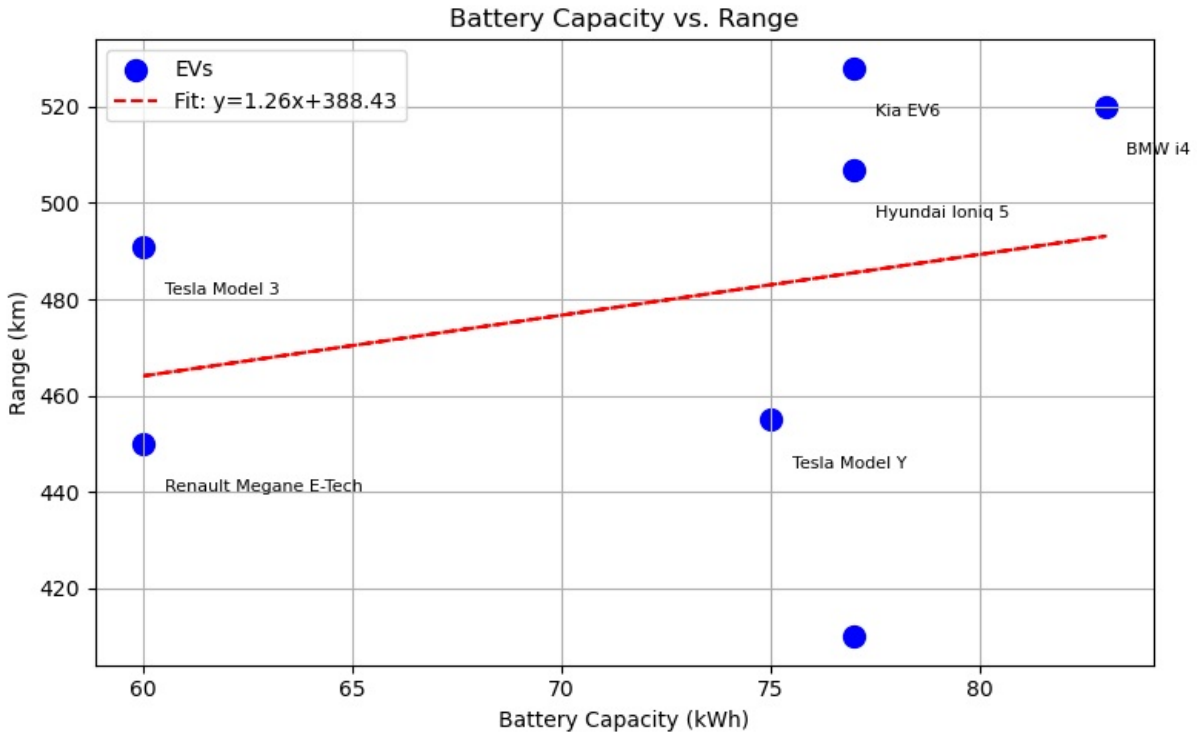
In [ ]:
```
3.(a) Create a suitable plot to visualize
```

In [28]:
```
import matplotlib.pyplot as plt

# Sample EV data
ev_list = [
    {"make": "Tesla Model 3", "battery_kWh": 60, "range_km": 491},
    {"make": "Tesla Model Y", "battery_kWh": 75, "range_km": 455},
    {"make": "Hyundai Ioniq 5", "battery_kWh": 77, "range_km": 507},
    {"make": "BMW i4", "battery_kWh": 83, "range_km": 520},
    {"make": "Volkswagen ID.4", "battery_kWh": 77, "range_km": 410},
    {"make": "Kia EV6", "battery_kWh": 77, "range_km": 528},
    {"make": "Renault Megane E-Tech", "battery_kWh": 60, "range_km": 450},
]

# Extract data for plotting
battery = [ev["battery_kWh"] for ev in ev_list]
range_km = [ev["range_km"] for ev in ev_list]
labels = [ev["make"] for ev in ev_list]

# Create scatter plot
plt.figure(figsize=(8, 5))
plt.scatter(battery, range_km, color='blue', s=100)

# Annotate points with EV names
for i, label in enumerate(labels):
    plt.annotate(label, (battery[i]+0.5, range_km[i]-10), fontsize=8)

plt.title("Battery Capacity vs. Range")
plt.xlabel("Battery Capacity (kWh)")
plt.ylabel("Range (km)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [ ]:
```
(b) Highlight any insights
```

In [30]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import linregress

# Sample EV data
ev_list = [
    {"make": "Tesla Model 3", "battery_kWh": 60, "range_km": 491},
    {"make": "Tesla Model Y", "battery_kWh": 75, "range_km": 455},
    {"make": "Hyundai Ioniq 5", "battery_kWh": 77, "range_km": 507},
    {"make": "BMW i4", "battery_kWh": 83, "range_km": 520},
    {"make": "Volkswagen ID.4", "battery_kWh": 77, "range_km": 410},
    {"make": "Kia EV6", "battery_kWh": 77, "range_km": 528},
    {"make": "Renault Megane E-Tech", "battery_kWh": 60, "range_km": 450},
]

# Extract data
battery = np.array([ev["battery_kWh"] for ev in ev_list])
range_km = np.array([ev["range_km"] for ev in ev_list])
labels = [ev["make"] for ev in ev_list]

# Linear regression
slope, intercept, r_value, p_value, std_err = linregress(battery, range_km)
line = slope * battery + intercept

# Plotting
plt.figure(figsize=(8, 5))
plt.scatter(battery, range_km, color='blue', s=100, label="EVs")
plt.plot(battery, line, color='red', linestyle='--', label=f"Fit: y={slope:.2f}x+{intercept:.2f}")
for i, label in enumerate(labels):
    plt.annotate(label, (battery[i] + 0.5, range_km[i] - 10), fontsize=8)

plt.title("Battery Capacity vs. Range")
plt.xlabel("Battery Capacity (kWh)")
plt.ylabel("Range (km)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Insights
print(f"Pearson correlation coefficient: {r_value:.2f}")
if r_value > 0.7:
    print("Insight: There is a strong positive correlation — larger batteries tend to yield longer range.")
elif r_value > 0.4:
    print("Insight: There is a moderate positive correlation between battery size and range.")
else:
    print("Insight: Weak or no strong correlation — other factors (like efficiency or weight) might play a majo
```




```
Pearson correlation coefficient: 0.26
Insight: Weak or no strong correlation — other factors (like efficiency or weight) might play a major role.
```

4.(a) The class should allow users to input their budget, desired range, and battery capacity.

In [32]:
```python
class EVRecommendation:
    def __init__(self, ev_data):
        self.ev_data = ev_data

    def recommend(self, budget, min_range, min_battery):
        # Filter based on criteria
        recommendations = [
            ev for ev in self.ev_data
            if ev["price"] <= budget and ev["range_km"] >= min_range and ev["battery_kWh"] >= min_battery
        ]
        return recommendations

    def display(self, ev_list):
        if not ev_list:
            print("No matching EVs found.")
            return
        for ev in ev_list:
            print(f"{ev['make']} ({ev['manufacturer']}) - "
                  f"{ev['price']} PLN, {ev['range_km']} km, {ev['battery_kWh']} kWh")

# Example EV dataset
ev_list = [
    {"make": "Tesla Model 3", "manufacturer": "Tesla", "price": 240000, "range_km": 491, "battery_kWh": 60},
    {"make": "Tesla Model Y", "manufacturer": "Tesla", "price": 310000, "range_km": 455, "battery_kWh": 75},
    {"make": "Hyundai Ioniq 5", "manufacturer": "Hyundai", "price": 265000, "range_km": 507, "battery_kWh": 77},
    {"make": "BMW i4", "manufacturer": "BMW", "price": 360000, "range_km": 520, "battery_kWh": 83},
    {"make": "Volkswagen ID.4", "manufacturer": "Volkswagen", "price": 210000, "range_km": 410, "battery_kWh":
    {"make": "Kia EV6", "manufacturer": "Kia", "price": 280000, "range_km": 528, "battery_kWh": 77},
    {"make": "Renault Megane E-Tech", "manufacturer": "Renault", "price": 190000, "range_km": 450, "battery_kWh
]

# Example usage
recommender = EVRecommendation(ev_list)

# User input
user_budget = 300000
user_min_range = 450
user_min_battery = 60

# Recommend and display
matching_evs = recommender.recommend(user_budget, user_min_range, user_min_battery)
recommender.display(matching_evs)
```

```
Tesla Model 3 (Tesla) - 240000 PLN, 491 km, 60 kWh
Hyundai Ioniq 5 (Hyundai) - 265000 PLN, 507 km, 77 kWh
Kia EV6 (Kia) - 280000 PLN, 528 km, 77 kWh
Renault Megane E-Tech (Renault) - 190000 PLN, 450 km, 60 kWh
```

In [ ]:
```
(b) The class should then return the top three EVs
 matching their criteria
```

In [34]:
```
class EVRecommendation:
    def __init__(self, ev_data):
        self.ev_data = ev_data

    def recommend_top_3(self, budget, min_range, min_battery):
        # Filter EVs based on user criteria
        filtered = [
            ev for ev in self.ev_data
            if ev["price"] <= budget and ev["range_km"] >= min_range and ev["battery_kWh"] >= min_battery
        ]

        # Sort by range descending, then price ascending
        sorted_evs = sorted(filtered, key=lambda ev: (-ev["range_km"], ev["price"]))

        # Return top 3
        return sorted_evs[:3]

    def display(self, ev_list):
        if not ev_list:
            print("No matching EVs found.")
            return
        for ev in ev_list:
            print(f"{ev['make']} ({ev['manufacturer']}) - "
                  f"{ev['price']} PLN, {ev['range_km']} km, {ev['battery_kWh']} kWh")


# Sample EV dataset
ev_list = [
    {"make": "Tesla Model 3", "manufacturer": "Tesla", "price": 240000, "range_km": 491, "battery_kWh": 60},
    {"make": "Tesla Model Y", "manufacturer": "Tesla", "price": 310000, "range_km": 455, "battery_kWh": 75},
    {"make": "Hyundai Ioniq 5", "manufacturer": "Hyundai", "price": 265000, "range_km": 507, "battery_kWh": 77}
    {"make": "BMW i4", "manufacturer": "BMW", "price": 360000, "range_km": 520, "battery_kWh": 83},
    {"make": "Volkswagen ID.4", "manufacturer": "Volkswagen", "price": 210000, "range_km": 410, "battery_kWh":
    {"make": "Kia EV6", "manufacturer": "Kia", "price": 280000, "range_km": 528, "battery_kWh": 77},
    {"make": "Renault Megane E-Tech", "manufacturer": "Renault", "price": 190000, "range_km": 450, "battery_kWh
]

# Example usage
recommender = EVRecommendation(ev_list)

# User input
user_budget = 350000
user_min_range = 400
user_min_battery = 60

# Recommend and display top 3 EVs
top_3_evs = recommender.recommend_top_3(user_budget, user_min_range, user_min_battery)
recommender.display(top_3_evs)
```

```
Kia EV6 (Kia) - 280000 PLN, 528 km, 77 kWh
Hyundai Ioniq 5 (Hyundai) - 265000 PLN, 507 km, 77 kWh
Tesla Model 3 (Tesla) - 240000 PLN, 491 km, 60 kWh
```

In [ ]:
```
5. (i)Import libraries & sample dat
```

In [38]:
```
from scipy.stats import ttest_ind
import numpy as np

# Sample engine power data (KM) for Tesla and Audi
tesla_power = np.array([450, 480, 490, 510, 530])
audi_power = np.array([300, 320, 340, 350, 370])
```

In [ ]:
```
(ii) Conduct the two-sample t-test
```

In [40]:
```
# Two-sample t-test (equal variance not assumed)
t_stat, p_value = ttest_ind(tesla_power, audi_power, equal_var=False)

print(f"T-statistic: {t_stat:.2f}")
print(f"P-value: {p_value:.4f}")
```

```
T-statistic: 8.59
P-value: 0.0000
```

In [ ]:
```
(iii) Interpret results
```

In [42]:
```
alpha = 0.05  # significance level

if p_value < alpha:
    print("Result: Significant difference in average engine power between Tesla and Audi.")
else:
    print("Result: No significant difference in average engine power between Tesla and Audi.")
```

Result: Significant difference in average engine power between Tesla and Audi.

In [ ]: