```
In [3]: #Step 1: Setup and Data Loading
        !pip install pandas
        !pip install matplotlib
        !pip install seaborn
        !pip install scipy
```

```
Requirement already satisfied: pandas in c:\users\mohini\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\mohini\anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\mohini\anaconda3\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\mohini\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\mohini\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\mohini\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: matplotlib in c:\users\mohini\anaconda3\lib\site-packages (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: numpy>=1.23 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in c:\users\mohini\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: seaborn in c:\users\mohini\anaconda3\lib\site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\mohini\anaconda3\lib\site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in c:\users\mohini\anaconda3\lib\site-packages (from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\mohini\anaconda3\lib\site-packages (from seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\mohini\anaconda3\lib\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\mohini\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\mohini\anaconda3\lib\site-packages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\mohini\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Requirement already satisfied: scipy in c:\users\mohini\anaconda3\lib\site-packages (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in c:\users\mohini\anaconda3\lib\site-packages (from scipy) (1.26.4)
```

```
In [4]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        from scipy.stats import ttest_ind
        from scipy.stats import pearsonr
```

```
In [5]: dataset = pd.read_csv("C:/Users/MOHINI/Downloads/FEV-data-Excel.xlsx - Auta elektryczne.csv")
        dataset.head()
        dataset.info()
        dataset.describe()
```

```
<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
```

Out[5]:

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

	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]	Accele 0-10
count	53.000000	45.000000	45.000000	53.000000	53.000000	53.000000	53.000000	52.000000	50
mean	1868.452830	2288.844444	520.466667	4.905660	4.849057	17.679245	178.169811	445.096154	7
std	470.880867	557.796026	140.682848	0.838133	0.455573	1.868500	43.056196	180.178480	2
min	1035.000000	1310.000000	290.000000	2.000000	3.000000	14.000000	123.000000	171.000000	2
25%	1530.000000	1916.000000	440.000000	5.000000	5.000000	16.000000	150.000000	315.000000	4
50%	1685.000000	2119.000000	486.000000	5.000000	5.000000	17.000000	160.000000	425.000000	7
75%	2370.000000	2870.000000	575.000000	5.000000	5.000000	19.000000	200.000000	558.000000	9
max	2710.000000	3500.000000	1056.000000	8.000000	5.000000	21.000000	261.000000	870.000000	13

```
In [6]: #Step 2: Data Cleaning and Preprocessing
        dataset.isnull().sum()  # Check for missing values
        dataset.dropna(inplace=True)  # Remove rows with missing values
```

```
In [7]: dataset['Minimal price (gross) [PLN]'] = dataset['Minimal price (gross) [PLN]'].astype(float)
```

```
In [8]: dataset.rename(columns={'Minimal price (gross) [PLN]': 'Price', 'Range (WLTP) [km]': 'Range'}, inplace=True)
```

```
In [9]: #Step 3:
        #Task 1
        #part a
        #Filtering

        filtered_dataset = dataset[(dataset['Price'] <= 350000) & (dataset['Range'] >= 400)]
        print(filtered_dataset[['Make', 'Model', 'Price', 'Range']])
```

```
             Make                 Model     Price  Range
0            Audi     e-tron 55 quattro  345700.0    438
8             BMW                   iX3  282900.0    460
15        Hyundai   Kona electric 64kWh  178400.0    449
18            Kia          e-Niro 64kWh  167990.0    455
20            Kia          e-Soul 64kWh  160990.0    452
22  Mercedes-Benz                   EQC  334700.0    414
47     Volkswagen  ID.3 Pro Performance  155890.0    425
48     Volkswagen            ID.3 Pro S  179990.0    549
49     Volkswagen              ID.4 1st  202390.0    500
```

```
In [10]: #part b
         #Grouping
         grouped = filtered_dataset.groupby('Make').size()
         print(grouped)
```

```
Make
Audi             1
BMW              1
Hyundai          1
Kia              2
Mercedes-Benz    1
Volkswagen       3
dtype: int64
```

```
In [11]: #part c
         #calculating avarage
         battery_avg = filtered_dataset.groupby('Make')['Battery capacity [kWh]'].mean()
         print(battery_avg)
```

```
Make
Audi             95.000000
BMW              80.000000
Hyundai          64.000000
Kia              64.000000
Mercedes-Benz    80.000000
Volkswagen       70.666667
Name: Battery capacity [kWh], dtype: float64
```

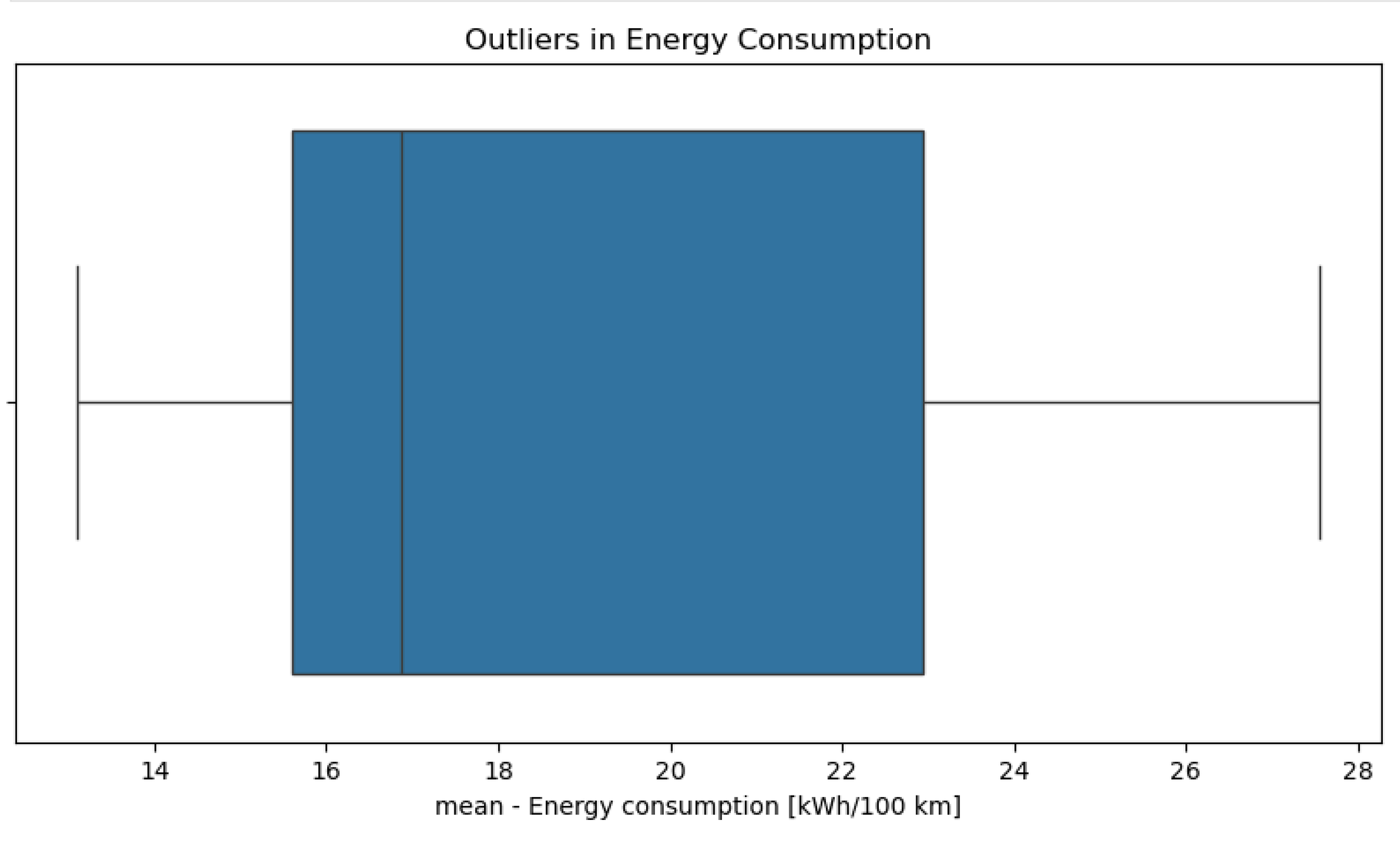
```
In [12]: import numpy as np

         # Task 2: Detecting Outliers in Energy Consumption
         def detect_outliers():
             column_name = "mean - Energy consumption [kWh/100 km]"

             # Calculate Q1 (25th percentile) and Q3 (75th percentile)
             q1 = dataset[column_name].quantile(0.25)
             q3 = dataset[column_name].quantile(0.75)

             # Calculate Interquartile Range (IQR)
             iqr = q3 - q1

             # Define lower and upper bounds for outliers
             lower_bound = q1 - 1.5 * iqr
             upper_bound = q3 + 1.5 * iqr

             # Find outliers: values outside the lower and upper bounds
             outliers = dataset[(dataset[column_name] < lower_bound) | (dataset[column_name] > upper_bound)]

             # Display results
             print(f"Lower Bound: {lower_bound:.2f}, Upper Bound: {upper_bound:.2f}")
             print(f"Number of Outliers: {len(outliers)}")
             print("Outliers in Energy Consumption:\n", outliers[['Car full name', 'Make', column_name]])

             return outliers

         # Run Task 2
         outliers = detect_outliers()
```

```
Lower Bound: 4.59, Upper Bound: 33.94
Number of Outliers: 0
Outliers in Energy Consumption:
 Empty DataFrame
Columns: [Car full name, Make, mean - Energy consumption [kWh/100 km]]
Index: []
```

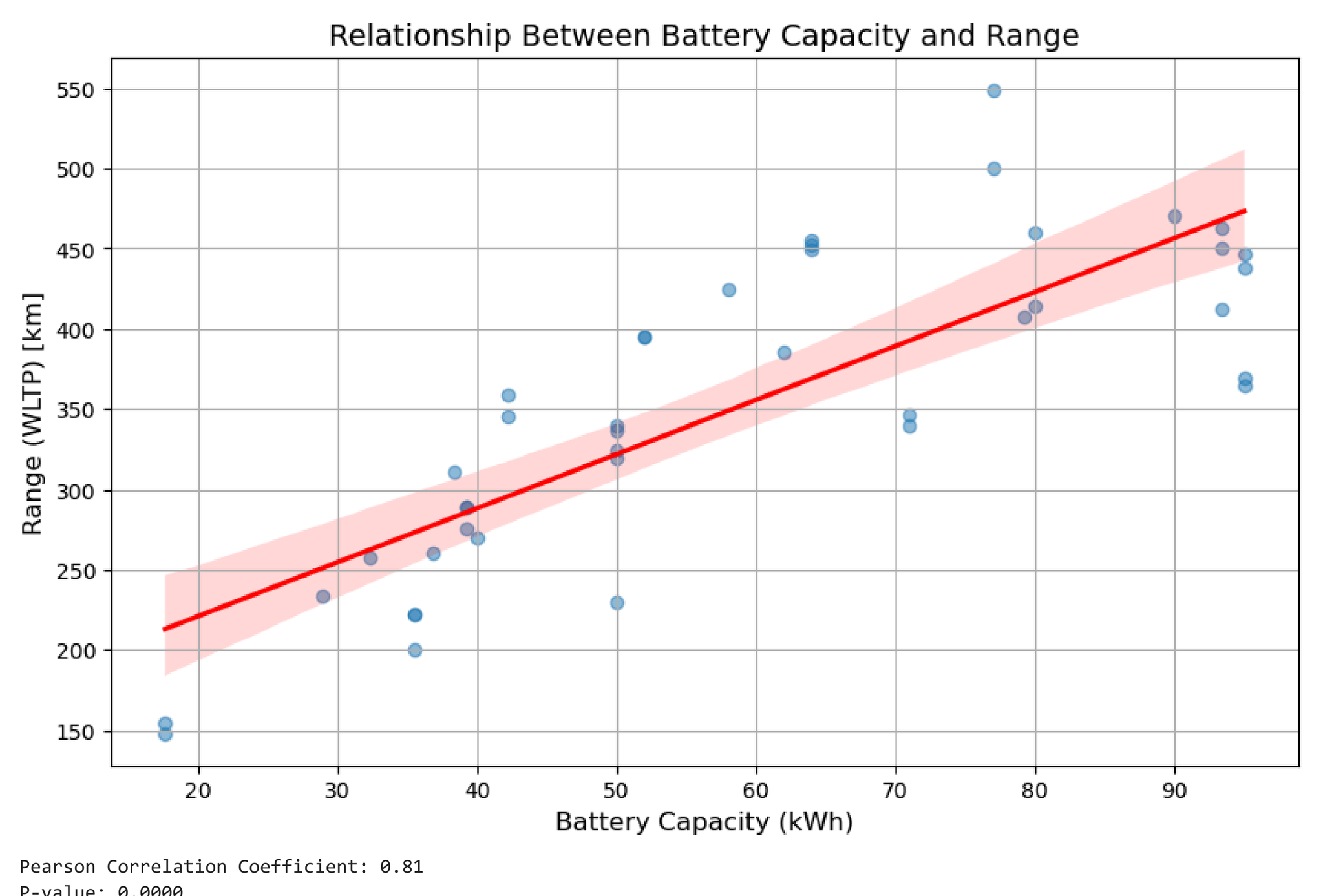
```
In [13]: #alternate method of task-2
         #note-The question says "You suspect some EVs have unusually high or low energy consumption. Find the outliers in the mean - Energy consumption [kWh/100 km] column".
         # so i did the top 5% and bottom 5% approach
         def detect_outliers():
             column_name = "mean - Energy consumption [kWh/100 km]"

             # Define the bottom 5% and top 5% thresholds
             low_threshold = dataset[column_name].quantile(0.05)  # Bottom 5%
             high_threshold = dataset[column_name].quantile(0.95)  # Top 5%

             # Find EVs that consume much less or much more energy than usual
             outliers = dataset[(dataset[column_name] < low_threshold) | (dataset[column_name] > high_threshold)]

             # Display results
             print(f"Bottom 5% Threshold: {low_threshold:.2f}, Top 5% Threshold: {high_threshold:.2f}")
             print(f"Number of Outliers: {len(outliers)}")
             print("Outliers in Energy Consumption:\n", outliers[['Car full name', 'Make', column_name]])

             return outliers

         # Run Task 2 with percentiles
         outliers = detect_outliers()
```

```
Bottom 5% Threshold: 14.02, Top 5% Threshold: 25.20
Number of Outliers: 6
Outliers in Energy Consumption:
                  Car full name        Make  \
2            Audi e-tron S quattro        Audi
5   Audi e-tron Sportback S quattro        Audi
6                           BMW i3         BMW
13          Hyundai Ioniq electric     Hyundai
46              Volkswagen e-up!  Volkswagen
50       Citroën ë-Spacetourer (M)     Citroën

    mean - Energy consumption [kWh/100 km]
2                                    27.55
5                                    27.20
6                                    13.10
13                                   13.80
46                                   14.00
50                                   25.20
```

```
In [14]: #ploting outliers
         plt.figure(figsize=(10,5))
         sns.boxplot(x=dataset['mean - Energy consumption [kWh/100 km]'])
         plt.title("Outliers in Energy Consumption")
         plt.show()
```




```
In [15]: #Task 3 - Relationship Between Battery Capacity and Range
         #part a
         # Scatter plot with regression line
         plt.figure(figsize=(10, 6))
         sns.regplot(x=dataset["Battery capacity [kWh]"], y=dataset["Range"],
                     scatter_kws={'alpha': 0.5}, line_kws={"color": "red"})

         # Labels and title
         plt.xlabel("Battery Capacity (kWh)", fontsize=12)
         plt.ylabel("Range (WLTP) [km]", fontsize=12)
         plt.title("Relationship Between Battery Capacity and Range", fontsize=14)
         plt.grid(True)

         # Show plot
         plt.show()

         #part b
         # Calculate Pearson correlation coefficient
         correlation, p_value = pearsonr(dataset["Battery capacity [kWh]"], dataset["Range"])

         # Print correlation results
         print(f"Pearson Correlation Coefficient: {correlation:.2f}")
         print(f"P-value: {p_value:.4f}")

         # Determine the strength of the relationship
         if abs(correlation) >= 0.8:
             print("The relationship between Battery Capacity and Range is **strong**.")
         elif abs(correlation) >= 0.5:
             print("The relationship between Battery Capacity and Range is **moderate**.")
         else:
             print("The relationship between Battery Capacity and Range is **weak**.")
```




```
Pearson Correlation Coefficient: 0.81
P-value: 0.0000
The relationship between Battery Capacity and Range is **strong**.
```

```
In [16]: #Task 4 - EV Recommendation System
         # Load dataset
         import pandas as pd
         dataset = pd.read_csv("C:/Users/MOHINI/Downloads/FEV-data-Excel.xlsx - Auta elektryczne.csv")

         # Strip spaces from column names (Prevents KeyErrors due to extra spaces)
         dataset.columns = dataset.columns.str.strip()
         class EVRecommender:
             def __init__(self, data):
                 self.data = data

             def recommend(self, budget, min_range, min_battery):
                 filtered = self.data[(self.data['Minimal price (gross) [PLN]'] <= budget) &
                                      (self.data['Range (WLTP) [km]'] >= min_range) &
                                      (self.data['Battery capacity [kWh]'] >= min_battery)]
                 results = filtered[['Make', 'Model', 'Minimal price (gross) [PLN]', 'Range (WLTP) [km]', 'Battery capacity [kWh]']].nsmallest(3, 'Minimal price (gross) [PLN]')

                 # Display results in column format
                 return results.to_string(index=False)

         # Example usage:
         recommender = EVRecommender(dataset)
         print(recommender.recommend(350000, 400, 60))  # Recommend EVs based on criteria
```

```
   Make               Model  Minimal price (gross) [PLN]  Range (WLTP) [km]  Battery capacity [kWh]
    Kia        e-Soul 64kWh                       160990                452                    64.0
    Kia        e-Niro 64kWh                       167990                455                    64.0
Hyundai Kona electric 64kWh                       178400                449                    64.0
```

```
In [17]: #Task 5 - Hypothesis Testing (Tesla vs Audi)
         from scipy.stats import ttest_ind

         tesla_power = dataset[dataset['Make'] == 'Tesla']['Engine power [KM]']
         audi_power = dataset[dataset['Make'] == 'Audi']['Engine power [KM]']

         t_stat, p_value = ttest_ind(tesla_power, audi_power, equal_var=False)  # Welch's t-test

         print("\n🔹 Tesla vs Audi Engine Power Analysis:")
         print(f"T-statistic: {t_stat:.2f}")
         print(f"P-value: {p_value:.4f}")

         if p_value < 0.05:
             print("There is a **significant difference** in engine power between Tesla and Audi.")
         else:
             print("There is **no significant difference** in engine power between Tesla and Audi.")

         ### 🔥 Final Recommendations ###
         print("\n📌 **Final Recommendations:**")
         print("1️⃣ Customers looking for **long-range EVs within budget** should consider:")


         print("\n2️⃣ **Manufacturers** should focus on optimizing **energy efficiency**, as some models have unusually high/low consumption.")

         if p_value < 0.05:
             print("\n3️⃣ Tesla and Audi buyers should **consider performance differences**, as their engine power differs significantly.")
         else:
             print("\n3️⃣ Tesla and Audi buyers can focus on other factors like **price, range, and charging speed**, since engine power is similar.")
```

```
🔹 Tesla vs Audi Engine Power Analysis:
T-statistic: 1.79
P-value: 0.1068
There is **no significant difference** in engine power between Tesla and Audi.

📌 **Final Recommendations:**
1️⃣ Customers looking for **long-range EVs within budget** should consider:

2️⃣ **Manufacturers** should focus on optimizing **energy efficiency**, as some models have unusually high/low consumption.

3️⃣ Tesla and Audi buyers can focus on other factors like **price, range, and charging speed**, since engine power is similar.
```