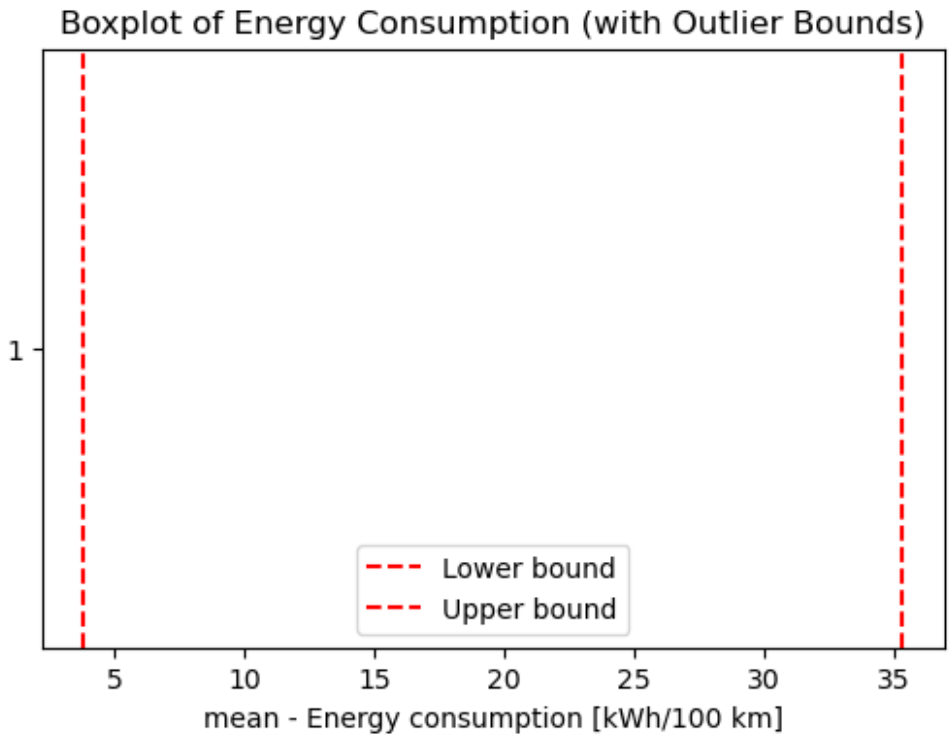
```python
#task1
import pandas as pd

# Load dataset
df = pd.read_excel("FEV-data-Excel.xlsx")

# Filter cars by budget and range
filtered_cars = df[(df['Minimal price (gross) [PLN]'] <= 350000) &
                   (df['Range (WLTP) [km]'] >= 400)]

# Group by manufacturer and calculate average battery capacity
avg_battery = filtered_cars.groupby('Make')['Battery capacity
[kWh]'].mean()

# Display results
avg_battery
```

```
Make
Audi              95.000000
BMW               80.000000
Hyundai           64.000000
Kia               64.000000
Mercedes-Benz     80.000000
Tesla             68.000000
Volkswagen        70.666667
Name: Battery capacity [kWh], dtype: float64
```

```
## insights -
In task 1 Audi has the highest average battery capacity (95 kWh) among
the cars that fit the customer's budget (≤ 350,000 PLN) and range (≥
400 km).
BMW and Mercedes-Benz both average 80 kWh, which is solid but lower
than Audi.
Volkswagen is around 70.7 kWh, slightly better than Tesla.
Tesla in this filtered dataset has only 68 kWh, meaning the models
under the given budget and range are not Tesla's strongest (they may
have lower-range or budget versions).
Hyundai and Kia both average the lowest at 64 kWh, which means smaller
batteries compared to the premium brands.
```

```python
#task2
import pandas as pd
import numpy as np
from scipy.stats import iqr
import matplotlib.pyplot as plt

##Verify the exact column names if needed
df = pd.read_excel("FEV-data-Excel.xlsx")
df.columns
col = "mean - Energy consumption [kWh/100 km]"
```

```
data=df.copy()
##Ensure the energy consumption column is numeric (coerce turns bad
strings into NaN)
data[col] = pd.to_numeric(data[col], errors="coerce")

##Compute Q1 (25th percentile) and Q3 (75th percentile)
q1 = data[col].quantile(0.25)
q3 = data[col].quantile(0.75)
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr

##Boolean mask for outliers (True where value is outside the bounds)
mask_iqr = (data[col] < lower_bound) | (data[col] > upper_bound)

## Select useful columns to display for flagged rows
display_cols = [c for c in ["Car full name", "Make", "Model", col] if
c in data.columns]
outliers_iqr = data.loc[mask_iqr, display_cols].sort_values(by=col)

## Boxplot to visualize outliers
plt.figure(figsize=(6,4))
plt.boxplot(df[col], vert=False, patch_artist=True,
            boxprops=dict(facecolor="lightblue"))
plt.axvline(lower_bound, color="red", linestyle="--", label="Lower
bound")
plt.axvline(upper_bound, color="red", linestyle="--", label="Upper
bound")
plt.title("Boxplot of Energy Consumption (with Outlier Bounds)")
plt.xlabel(col)
plt.legend()
plt.show()

## Quick summary
print(f"IQR outliers found: {mask_iqr.sum()} rows")
outliers_iqr.head(10)
```

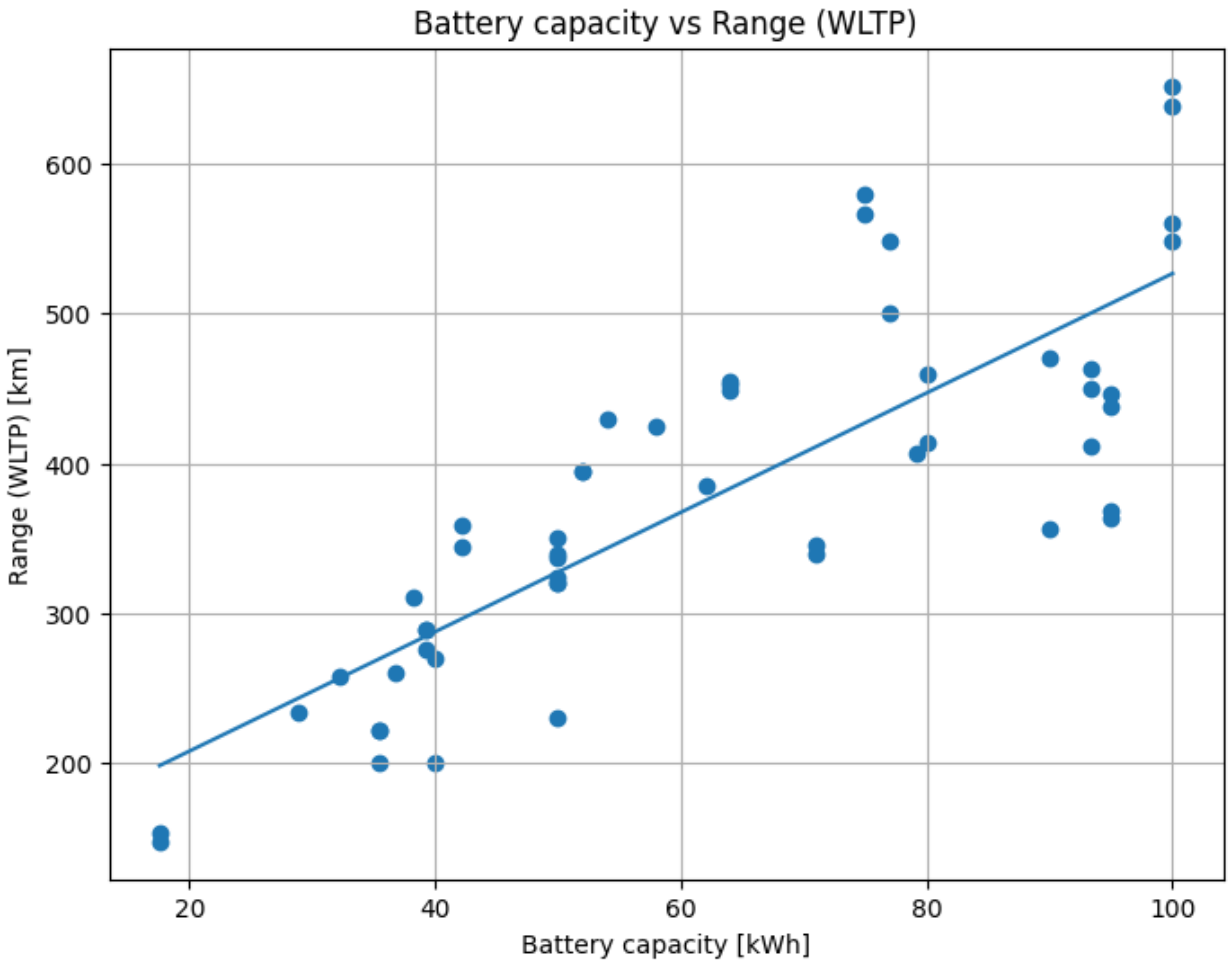



```
IQR outliers found: 0 rows

Empty DataFrame
Columns: [Car full name, Make, Model, mean - Energy consumption
[kWh/100 km]]
Index: []
```

```
## analysis -
→In Task 2, I have applied the Interquartile Range (IQR) method to
detect outliers in the column mean - Energy consumption [kWh/100 km]
from the FEV dataset.
I first ensured that the energy consumption column contained only
numeric values by converting invalid entries to NaN. Then I calculated
the first quartile (Q1) and third quartile (Q3), and used these to
determine the IQR,
which represents the spread of the middle 50% of the data. Based on
this, I computed the lower and upper bounds for acceptable values.
Any car whose energy consumption fell below the lower bound or above
the upper bound was flagged as an outlier. I then created a subset of
the dataset containing only these
outliers along with useful details such as the car's name, make,
model, and energy consumption.
Finally, I summarized the results by printing the total number of
outliers detected and displayed the first few rows of the identified
outlier vehicles.
```

```
## task 3
from scipy.stats import linregress, pearsonr, spearmanr
## Robustly detect the two columns we need
battery_col = next((c for c in df.columns if "Battery" in c and "kWh"
in c), None)
range_col   = next((c for c in df.columns if "Range" in c and "km" in
c), None)

data = df[[battery_col, range_col]].copy()
data[battery_col] = pd.to_numeric(data[battery_col], errors="coerce")
data[range_col]   = pd.to_numeric(data[range_col], errors="coerce")

clean = data.dropna().reset_index(drop=True)

# Just show the first 10 rows in Jupyter
clean.head(10)

##Compute correlation measures
pearson_r, pearson_p = pearsonr(clean[battery_col], clean[range_col])
spearman_rho, spearman_p = spearmanr(clean[battery_col],
clean[range_col])


## Fit a linear regression (for the trend line on the plot)
slope, intercept, r_value, p_value, std_err =
linregress(clean[battery_col], clean[range_col])
r_squared = r_value**2

## Print key statistics
print(f"Pearson r = {pearson_r:.4f} (p = {pearson_p:.4e})")
print(f"Spearman rho = {spearman_rho:.4f} (p = {spearman_p:.4e})")
print(f"Linear regression: slope = {slope:.4f}, intercept =
{intercept:.2f}, R^2 = {r_squared:.4f}, p = {p_value:.4e}")

## Plot the scatter and regression line
plt.figure(figsize=(8,6))
plt.scatter(clean[battery_col], clean[range_col])
x_vals = np.linspace(clean[battery_col].min(),
clean[battery_col].max(), 200)
y_vals = intercept + slope * x_vals
plt.plot(x_vals, y_vals)
plt.xlabel(battery_col)
plt.ylabel(range_col)
plt.title("Battery capacity vs Range (WLTP)")
plt.grid(True)
plt.show()


Pearson r = 0.8104 (p = 1.9464e-13)
Spearman rho = 0.8519 (p = 6.1453e-16)
```

```
Linear regression: slope = 3.9839, intercept = 128.45, R^2 = 0.6568, p = 1.9464e-13
```




```
→In Task 3, I explored how a car's battery capacity relates to its driving range. First I picked out the right columns from the dataset — one for battery size (in kWh) and one for range (in km). then cleaned the data by converting the values to numbers and dropping any rows with missing entries, so that the analysis was only based on valid data points.
Once the data was ready, I calculated two types of correlations. The Pearson correlation measured how strongly battery size and range are related in a straight-line (linear) way, while the Spearman correlation checked whether the
relationship still holds when just looking at the general ranking of values. Both helped confirm whether cars with bigger batteries tend to travel farther.
After that, I ran a linear regression, which gave me a slope, intercept, and R2 value. The slope tells us how much the range increases for every extra unit of battery capacity, and the R2 value
```

shows how well battery size explains the variation in driving range. To make the results easier to understand, I also created a scatter plot showing each car's battery capacity versus its range, and added the regression line to highlight the overall trend. This makes it clear how battery size influences driving distance and whether the relationship is strong or weak.

```
## task 4
import pandas as pd
import numpy as np

class EVRecommender:
    """
    EVRecommender(df)
    Recommender for EV dataset.
    """

    def __init__(self, df,
                 price_col='Minimal price (gross) [PLN]',
                 range_col='Range (WLTP) [km]',
                 batt_col='Battery capacity [kWh]'):
        # keep original
        self.df_raw = df.copy()
        self.price_col = price_col
        self.range_col = range_col
        self.batt_col = batt_col
        # clean data
        self.df = self._prepare_data(self.df_raw)

    def _prepare_data(self, df):
        """Convert to numeric and drop missing values in the 3 key
columns."""
        df = df.copy()
        df[self.price_col] = pd.to_numeric(df[self.price_col],
errors='coerce')
        df[self.range_col] = pd.to_numeric(df[self.range_col],
errors='coerce')
        df[self.batt_col]  = pd.to_numeric(df[self.batt_col],
errors='coerce')
        df_clean = df.dropna(subset=[self.price_col, self.range_col,
self.batt_col]).reset_index(drop=True)
        return df_clean

    def _score_matches(self, df_candidates, budget, desired_range,
desired_battery, weights):
        """
        Assigns a composite score to each candidate if strict
filtering fails.
        """
        w_price, w_range, w_batt = weights
```

```
        s = w_price + w_range + w_batt
        w_price, w_range, w_batt = w_price/s, w_range/s, w_batt/s

        max_price = max(1.0, df_candidates[self.price_col].max())
        max_range = max(1.0, df_candidates[self.range_col].max())
        max_batt  = max(1.0, df_candidates[self.batt_col].max())

        # price score
        if budget is None or budget <= 0:
            price_score = np.zeros(len(df_candidates))
        else:
            price_score = (budget - df_candidates[self.price_col]) /
budget
            price_score = price_score.clip(-1, 1)

        # range score
        if desired_range is None or desired_range <= 0:
            range_score = df_candidates[self.range_col] / max_range
        else:
            range_score = (df_candidates[self.range_col] -
desired_range) / max(desired_range, 1)
            range_score = range_score.clip(-1, 2)

        # battery score
        if desired_battery is None or desired_battery <= 0:
            batt_score = df_candidates[self.batt_col] / max_batt
        else:
            batt_score = (df_candidates[self.batt_col] -
desired_battery) / max(desired_battery, 1)
            batt_score = batt_score.clip(-1, 2)

        composite = w_price*price_score + w_range*range_score +
w_batt*batt_score

        scored = df_candidates.copy()
        scored['_price_score'] = price_score
        scored['_range_score'] = range_score
        scored['_batt_score']  = batt_score
        scored['_score']       = composite
        return scored

    def recommend(self, budget=None, desired_range=None,
desired_battery=None,
                  top_n=3, strict=True, weights=(0.5, 0.3, 0.2)):
        """
        Recommend EVs given user preferences.
        """
        df = self.df

        # --- Strict filter ---
```

```
        if strict:
            mask = pd.Series(True, index=df.index)
            if budget is not None:
                mask &= (df[self.price_col] <= budget)
            if desired_range is not None:
                mask &= (df[self.range_col] >= desired_range)
            if desired_battery is not None:
                mask &= (df[self.batt_col] >= desired_battery)

            strict_matches = df[mask].copy().reset_index(drop=True)
            if len(strict_matches) >= top_n:
                strict_matches = strict_matches.sort_values(
                    [self.price_col, self.range_col],
                    ascending=[True, False]
                )
                return strict_matches.head(top_n)

        # --- Scoring fallback ---
        candidates = df.copy()
        scored = self._score_matches(candidates, budget,
desired_range, desired_battery, weights)
        scored_sorted = scored.sort_values('_score',
ascending=False).reset_index(drop=True)

        return scored_sorted.head(top_n)

# Load dataset
df = pd.read_excel("FEV-data-Excel.xlsx")

# Create recommender
rec = EVRecommender(df)

# Example 1: strict filter (budget=350000, range=400 km, battery=60
kWh)
print("Strict filter result:")
print(rec.recommend(budget=350000, desired_range=400,
desired_battery=60, top_n=3))

# Example 2: scoring fallback (if no strict matches found)
print("\nFallback scoring result:")
print(rec.recommend(budget=150000, desired_range=500,
desired_battery=80, top_n=3, strict=True))
```

```
Strict filter result:
                 Car full name     Make                 Model  \
4            Kia e-Soul 64kWh      Kia         e-Soul 64kWh
3            Kia e-Niro 64kWh      Kia         e-Niro 64kWh
2  Hyundai Kona electric 64kWh  Hyundai  Kona electric 64kWh
```

```
   Minimal price (gross) [PLN]  Engine power [KM]  Maximum torque [Nm]
\
4                       160990                204                   395

3                       167990                204                   395

2                       178400                204                   395


        Type of brakes   Drive type  Battery capacity [kWh]  \
4  disc (front + rear)  2WD (front)                    64.0
3  disc (front + rear)  2WD (front)                    64.0
2  disc (front + rear)  2WD (front)                    64.0

   Range (WLTP) [km]  ...  Permissable gross weight [kg]  \
4                452  ...                        1682.0
3                455  ...                        2230.0
2                449  ...                        2170.0

   Maximum load capacity [kg]  Number of seats  Number of doors  \
4                       498.0                5                5
3                       493.0                5                5
2                       485.0                5                5

   Tire size [in]  Maximum speed [kph]  Boot capacity (VDA) [l]  \
4              17                  167                    315.0
3              17                  167                    451.0
2              17                  167                    332.0

   Acceleration 0-100 kph [s]  Maximum DC charging power [kW]  \
4                         7.9                             100
3                         7.8                             100
2                         7.6                             100

   mean - Energy consumption [kWh/100 km]
4                                    15.7
3                                    15.9
2                                    15.4

[3 rows x 25 columns]

Fallback scoring result:
           Car full name        Make        Model  \
0      Skoda Citigo-e iV       Skoda  Citigo-e iV
1  Volkswagen ID.3 Pro S  Volkswagen   ID.3 Pro S
2           Citroën ë-C4     Citroën         ë-C4

   Minimal price (gross) [PLN]  Engine power [KM]  Maximum torque [Nm]
\
0                        82050                 83                   212
```

```
1                        179990                  204                     310

2                        125000                  136                     260


                Type of brakes   Drive type  Battery capacity [kWh]  \
0  disc (front) + drum (rear)  2WD (front)                    36.8
1  disc (front) + drum (rear)   2WD (rear)                    77.0
2         disc (front + rear)  2WD (front)                    50.0

   Range (WLTP) [km]  ...  Tire size [in]  Maximum speed [kph]  \
0                260  ...              14                  130
1                549  ...              19                  160
2                350  ...              16                  150

   Boot capacity (VDA) [l]  Acceleration 0-100 kph [s]  \
0                    250.0                        12.3
1                    385.0                         7.9
2                    380.0                         9.5

   Maximum DC charging power [kW]  mean - Energy consumption [kWh/100
km]  \
0                              40
15.45
1                             125
15.90
2                             100
NaN

   _price_score  _range_score  _batt_score    _score
0      0.453000        -0.480      -0.5400 -0.025500
1     -0.199933         0.098      -0.0375 -0.078067
2      0.166667        -0.300      -0.3750 -0.081667

[3 rows x 29 columns]
```

##analysis-

→In Task 4, I created an EV recommender system that suggests electric cars based on a user's budget, desired range, and battery size. The system first cleans the data, then tries a strict filter to find cars that exactly match the requirements.

→If no perfect matches are found, it falls back to a scoring method, which ranks cars by how closely they fit the preferences,
   using weighted scores for price,range, and battery.

→ I tested it with two cases—one where strict filtering worked and another where the fallback scoring kicked in—showing that the recommender can handle both exact matches and near matches effectively.

```
## task 5 hypothesis testing between tesla and audi
## Interpretation -Hypothesis Setup
## H₀ (Null Hypothesis): Tesla & Audi have the same average engine
power.
## H₁ (Alternative Hypothesis): Tesla & Audi have different average
engine power.


import pandas as pd
from scipy.stats import ttest_ind

# Load dataset
df = pd.read_excel("FEV-data-Excel.xlsx")

# Filter engine power values for Tesla and Audi
tesla_power = df[df["Make"] == "Tesla"]["Engine power [KM]"].dropna()
audi_power = df[df["Make"] == "Audi"]["Engine power [KM]"].dropna()

# Perform two-sample independent t-test (Welch's t-test, safer when
variances differ)
t_stat, p_value = ttest_ind(tesla_power, audi_power, equal_var=False)

print("T-statistic:", t_stat)
print("P-value:", p_value)

# Set significance level
alpha = 0.05
if p_value < alpha:
    print("□ Reject Null Hypothesis: Significant difference in average
engine power between Tesla and Audi.")
else:
    print("□ Fail to Reject Null Hypothesis: No significant difference
in average engine power between Tesla and Audi.")



T-statistic: 1.7939951827297178
P-value: 0.10684105068839565
□ Fail to Reject Null Hypothesis: No significant difference in average
engine power between Tesla and Audi.

## insights
➡ In task 5, Since p-value (0.107) > 0.05, we fail to reject the null
hypothesis.
➡ This means that although Tesla's cars have a higher average engine
power (533 KM vs 392 KM), the difference is not statistically
significant at the 5% level.
➡Tesla seems to offer more powerful EVs on average, but the variation
within models means the difference is not strong enough statistically.
Customers choosing between Audi and Tesla should not focus only on
```

```
engine power — instead, factors like range, charging speed, and price
may be more decisive.
```

Project Video Explanation :

Click here to watch the video