Electric Vehicle Data Analysis Project

Project Overview

In this project, you will analyze a dataset related to electric vehicles (EVs). The dataset contains various features such as electric range, energy consumption, price, and other relevant attributes. Your goal is to conduct a thorough analysis to uncover meaningful insights, tell a compelling story, conduct hypothesis testing and provide actionable recommendations based on the data.

Video Explanation Link: https://drive.google.com/file/d/1Je3O4O_rmvQOylWnSZjRSqwqH72PQFfn/view?usp=sharing

Import Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load, Inspect & Cleaning Data

Before starting any analysis, it's essential to load the dataset and understand its structure. Using df.info(), df.head(), and df.isnull().sum() allows us to quickly see the data types, the number of rows, and where missing values exist. This step ensures we know exactly what data we are working with and whether cleaning is required.

```
df = pd.read_excel('/content/drive/MyDrive/Python Project/FEV-data-Excel.xlsx')
df.head()
```

	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	N
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	
	Audi e-tron		e-tron				disc								

df.head() - Visual confirmation of column names & sample values.

```
df.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
```

df.info() - Identifies data types and non-null counts.

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

	0
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

df.isnull().sum() - Shows which columns require cleaning.

```
df['Type of brakes'] = df['Type of brakes'].fillna(df['Type of brakes'].mode()[0])
print("Cloumn Type of brakes contain nullvalues is: ",df['Type of brakes'].isnull().sum())
```

```
Cloumn Type of brakes contain nullvalues is:  0
```

For categorical columns like "Type of brakes", filling with the mode makes sense because categories must remain consistent and mean/median do not apply.

```
df[["Boot capacity (VDA) [l]","Acceleration 0-100 kph [s]","Permissable gross weight [kg]","Maximum load capacity [kg]","me
```

	Boot capacity (VDA) [l]	Acceleration 0-100 kph [s]	Permissable gross weight [kg]	Maximum load capacity [kg]	mean - Energy consumption [kWh/100 km]
count	52.000000	50.00000	45.000000	45.000000	44.000000
mean	445.096154	7.36000	2288.844444	520.466667	18.994318
std	180.178480	2.78663	557.796026	140.682848	4.418253
min	171.000000	2.50000	1310.000000	290.000000	13.100000
25%	315.000000	4.87500	1916.000000	440.000000	15.600000
50%	425.000000	7.70000	2119.000000	486.000000	17.050000
75%	558.000000	9.37500	2870.000000	575.000000	23.500000

```
num_cols = [
    'Boot capacity (VDA) [l]',
    'Acceleration 0-100 kph [s]',
    'Permissable gross weight [kg]',
    'Maximum load capacity [kg]',
    'mean - Energy consumption [kWh/100 km]'
]
for col in num_cols:
  df[col] = df[col].fillna(df[col].mean())

for col in num_cols:
  print(f"Cloumn {col} contain nullvalues is: {df[col].isnull().sum()}")
```

```
Cloumn Boot capacity (VDA) [l] contain nullvalues is: 0
Cloumn Acceleration 0-100 kph [s] contain nullvalues is: 0
Cloumn Permissable gross weight [kg] contain nullvalues is: 0
Cloumn Maximum load capacity [kg] contain nullvalues is: 0
Cloumn mean - Energy consumption [kWh/100 km] contain nullvalues is: 0
```

For numeric columns (e.g, Boot Capacity, Energy Consumption), filling missing values with the mean is reasonable because these features are continuous and the dataset is not extremely large.

```
df.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                        53 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]         53 non-null     float64
 16  Maximum load capacity [kg]            53 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]               53 non-null     float64
 22  Acceleration 0-100 kph [s]            53 non-null     float64
 23  Maximum DC charging power [kW]        53 non-null     int64
 24  mean - Energy consumption [kWh/100 km]  53 non-null     float64
dtypes: float64(10), int64(10), object(5)
memory usage: 10.5+ KB
```

Task 1: A customer has a budget of 350,000 PLN and wants an EV with a minimum range of 400 km.

1. Your task is to filter out EVs that meet these criteria.

```
df_filtered = df[
    (df['Minimal price (gross) [PLN]'] <= 350000) &
    (df['Range (WLTP) [km]'] >= 400)
]
df_filtered.head()
```

	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	N
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	
8	BMW iX3	BMW	iX3	282900	286	400	disc (front + rear)	2WD (rear)	80.0	460	...	2725.0	540.0	5	
15	Hyundai Kona electric 64kWh	Hyundai	Kona electric 64kWh	178400	204	395	disc (front + rear)	2WD (front)	64.0	449	...	2170.0	485.0	5	
18	Kia e-Niro 64kWh	Kia	e-Niro 64kWh	167990	204	395	disc (front + rear)	2WD (front)	64.0	455	...	2230.0	493.0	5	
20	Kia e-Soul 64kWh	Kia	e-Soul 64kWh	160990	204	395	disc (front + rear)	2WD (front)	64.0	452	...	1682.0	498.0	5	

We used a conditional filter because we needed to select EVs that satisfy two numerical constraints simultaneously:

- Price less than or equal to 350,000 PLN
- Range greater than or equal to 400 km

Using Pandas boolean filtering is the most efficient way to retrieve rows matching these criteria.

2. Group them by the manufacturer (Make).

```
grouped = df_filtered.groupby('Make')["Model"].count()
grouped
```

Make	Model
Audi	1
BMW	1
Hyundai	1
Kia	2
Mercedes-Benz	1
Tesla	3
Volkswagen	3

dtype: int64

Grouping by "Make" allows us to understand which brands produce EVs that meet customer requirements. This is important for market insights, helping us see which manufacturers offer long-range EVs at a reasonable price.

3. Calculate the average battery capacity for each manufacturer.

```
avg_battery = df_filtered.groupby("Make")['Battery capacity [kWh]'].mean().round(2)
avg_battery
```

Make	Battery capacity [kWh]
Audi	95.00
BMW	80.00
Hyundai	64.00
Kia	64.00
Mercedes-Benz	80.00
Tesla	68.00
Volkswagen	70.67

dtype: float64

Average battery capacity by manufacturer helps identify:

- Which brands offer powerful battery systems
- Whether higher-range vehicles from each brand rely on larger batteries

Task 2: You suspect some EVs have unusually high or low energy consumption. Find the outliers in the mean - Energy consumption [kWh/100 km] column.

```
Q1 = df['mean - Energy consumption [kWh/100 km]'].quantile(0.25)
Q3 = df['mean - Energy consumption [kWh/100 km]'].quantile(0.75)
print("Q1 values is:", Q1)
print("Q3 values is:", Q3)
IQR = Q3 - Q1
print("IQR values is:", IQR.round(2))
```

```
Q1 values is: 15.9
Q3 values is: 21.85
IQR values is: 5.95
```

The Interquartile Range (IQR) is a standard, widely accepted statistical technique for detecting outliers.

Logic behind IQR calculation:

- Q1 = 25th percentile
- Q3 = 75th percentile
- IQR = Q3 − Q1
- Outliers = values outside Q1 - 1.5*ISR or Q3 + 1.5*IQR This ensures we detect vehicles that consume far more or far less energy than typical EVs.

```
lower_level = Q1 - 1.5 * IQR
upper_level = Q3 + 1.5 * IQR
print("Lower lever is:", lower_level.round(2))
print("Upper lever is:", upper_level.round(2))
```

```
Lower lever is: 6.98
Upper lever is: 30.78
```

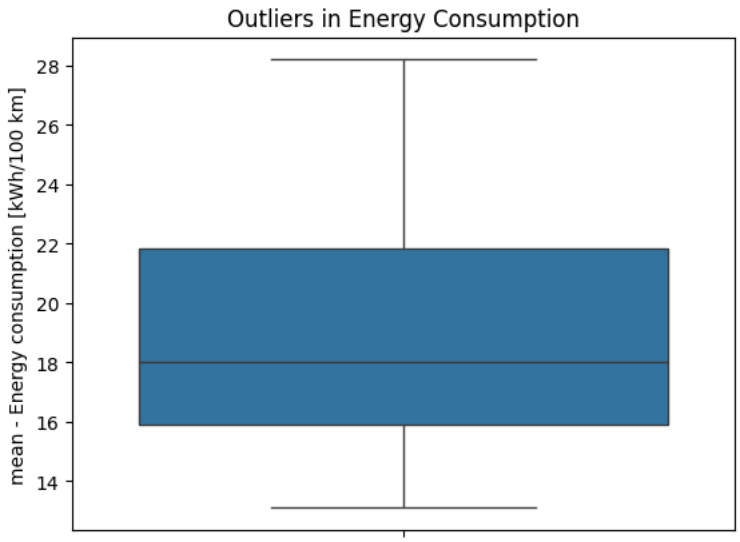
```
df[(df['mean - Energy consumption [kWh/100 km]'] < lower_level) |
              (df['mean - Energy consumption [kWh/100 km]'] > upper_level)]
```

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	Number of doors	Ti si [i

In dataset, no outliers were detected. This suggests:

- All EVs fall within a reasonable energy consumption range.
- The dataset is clean and consistent.
- There are no extreme or unusual models in terms of consumption efficiency.

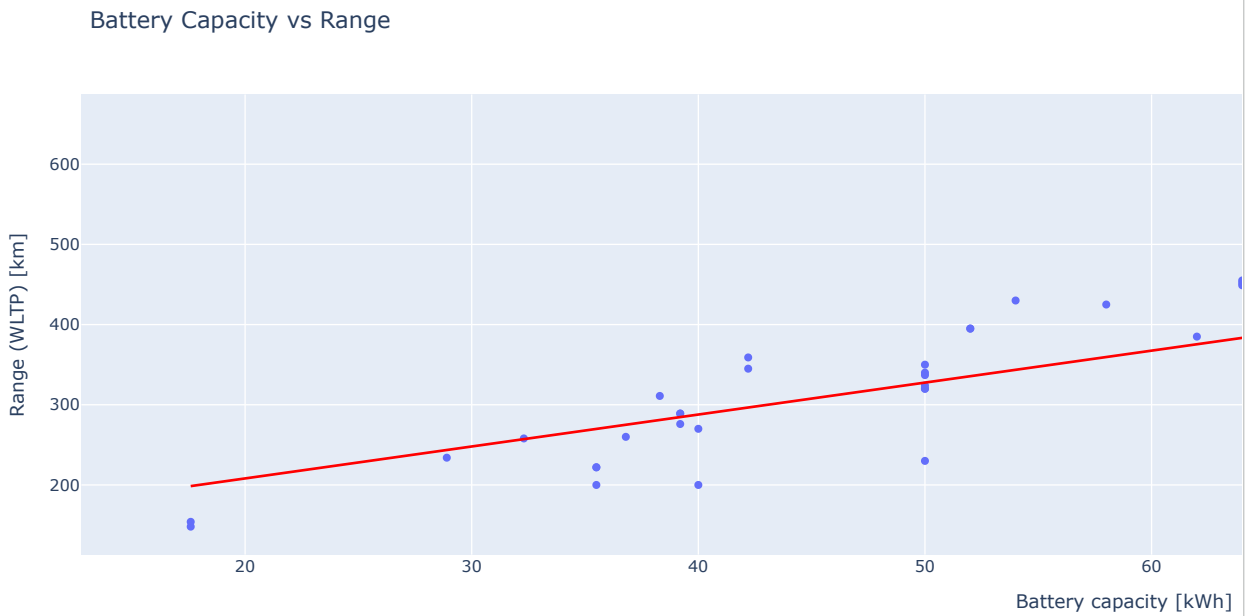
```
sns.boxplot(df['mean - Energy consumption [kWh/100 km]'])
plt.title('Outliers in Energy Consumption')
plt.show()
```



Task 3: Your manager wants to know if there's a strong relationship between battery capacity and range.

1. Create a suitable plot to visualize

```
import plotly.express as px
fig = px.scatter(df, x = df['Battery capacity [kWh]'], y = df['Range (WLTP) [km]'], title="Battery Capacity vs Range",
                 trendline="ols",  trendline_color_override="red" )
fig.show()
```



A scatter plot is the most suitable way to visualize the relationship between two continuous numeric variables.

Here, we want to observe how battery capacity influences driving range.

```
correlation = df[['Battery capacity [kWh]', 'Range (WLTP) [km]']].corr()
correlation
```

	Battery capacity [kWh]	Range (WLTP) [km]
Battery capacity [kWh]	1.000000	0.810439
Range (WLTP) [km]	0.810439	1.000000

After visual inspection, calculating the Pearson correlation coefficient quantifies the strength and direction of the relationship:

- A correlation of 0.81 indicates a strong positive relationship.
- This means EVs with higher battery capacity generally achieve higher range.

2. Highlight any insights

- Scatter Plot Confirms the Trend
  - The scatter plot visually shows an upward-sloping pattern.
  - As battery capacity increases, range tends to increase as well.
- Strong Positive Correlation
  - The correlation coefficient between Battery Capacity (kWh) and Range (WLTP) is 0.81.
  - A value above 0.8 indicates a strong positive linear relationship.
  - This suggests that EVs with larger battery packs generally achieve longer driving ranges.

Task 4: Build an EV recommendation class. The class should allow users to input their budget, desired range, and battery capacity. The class should then return the top three EVs matching their criteria.

```
class ev_recommender:
  def __init__(self, dataframe):
    self.df = dataframe

  def recommend(self):
    print("Please enter your preferences for an EV:")
    budget = int(input("Enter your budget in PLN (e.g. 350000): "))
    desired_range = int(input("Enter your desired minimum range in km (e.g. 400): "))
    battery_capacity = int(input("Enter your desired minimum battery capacity in kWh (e.g. 60): "))

    filtered_evs = self.df[
        (self.df['Minimal price (gross) [PLN]'] <= budget) &
        (self.df['Range (WLTP) [km]'] >= desired_range) &
        (self.df['Battery capacity [kWh]'] >= battery_capacity)
    ]
    return filtered_evs.sort_values(by='Range (WLTP) [km]', ascending=False).head(3)
```

Using an object-oriented approach allows you to package the recommendation logic in a clean, reusable structure.

The recommend() method filters EVs based on three user-provided preferences:

- Budget
- Minimum range
- Minimum battery capacity

These are the most common decision factors for EV buyers.

```
recommender = ev_recommender(df)
recommender.recommend()
```

Show hidden output

Customers usually want a shortlist rather than a long list.

Sorting by range ensures the best-performing EVs appear first.

Task 5: Inferential Statistics – Hypothesis Testing: Test whether there is a significant difference in the average Engine power [KM] of vehicles manufactured by two leading manufacturers i.e. Tesla and Audi. What insights can you draw from the test results?

Recommendations and Conclusion: Provide actionable insights based on your analysis.

```
tesla = df[df['Make'] == 'Tesla']['Engine power [KM]']
audi = df[df['Make'] == 'Audi']['Engine power [KM]']
tesla.head()
```

	Engine power [KM]
39	285
40	372
41	480
42	525
43	772

dtype: int64

```
audi.head()
```

	Engine power [KM]
0	360
1	313
2	503
3	313
4	360

dtype: int64

1. Null Hypothesis (H_0):

There is no significant difference in the average Engine power [KM] between Tesla and Audi electric vehicles.

---

2. Alternative Hypothesis (H_1):

There is a significant difference in the average Engine power [KM] between Tesla and Audi electric vehicles.

A two-sample t-test (specifically Welch's t-test, because equal_var=False) is used to compare the means of two independent groups — here:

- Tesla engine power
- Audi engine power

This statistical test tells us whether the difference in mean power is significant or simply due to random variation.

```
from scipy.stats import ttest_ind

t_stat, p_value = ttest_ind(tesla, audi, equal_var=False)
print(f"T-statistic: {t_stat.round(2)}, P-value: {p_value.round(2)}")
```

```
T-statistic: 1.79, P-value: 0.11
```

- Hypothesis Test Conclusion

  The independent two-sample t-test was used to compare the average Engine power (KM) of Tesla and Audi EVs.

  - T-statistic: 1.79
  - P-value: 0.11

  Since the p-value (0.11) is greater than the significance level of 0.05, we fail to reject the null hypothesis. This means there is no statistically significant difference in the average engine power between Tesla and Audi vehicles in this dataset.

  Although Tesla shows slightly higher mean engine power (as suggested by the positive t-statistic), the difference is not strong enough to be considered statistically meaningful.

- Recommendations

  Since engine power is similar between the two brands, customers should compare Tesla and Audi based on other factors like range, charging speed, price, and features. Both brands offer competitive performance, so the final choice should depend on the buyer's preferences rather than engine power alone.