Electric Vehicle Data Analysis Project

Project Overview

In this project, we 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. our 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.

Dataset:

FEV-data-Excel.xlsx

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Dataset loading
df = pd.read_excel("FEV-data-Excel.xlsx")
```

In [3]:
```
#Display the few rows of data
df.head(5)
```

Out[3]:

	Car full name	Make	Model	Minimal price (gross) [PLN]	Engine power [KM]	Maximum torque [Nm]	Type of brakes	Drive type	Battery capacity [kWh]	('
0	Audi e-tron 55 quattro	Audi	e-tron 55 quattro	345700	360	664	disc (front + rear)	4WD	95.0	
1	Audi e-tron 50 quattro	Audi	e-tron 50 quattro	308400	313	540	disc (front + rear)	4WD	71.0	
2	Audi e-tron S quattro	Audi	e-tron S quattro	414900	503	973	disc (front + rear)	4WD	95.0	
3	Audi e-tron Sportback 50 quattro	Audi	e-tron Sportback 50 quattro	319700	313	540	disc (front + rear)	4WD	71.0	
4	Audi e-tron Sportback 55 quattro	Audi	e-tron Sportback 55 quattro	357000	360	664	disc (front + rear)	4WD	95.0	

5 rows × 25 columns

In [4]:

```
#check the size of dataset
df.shape
```

Out[4]: (53, 25)

In [5]:

```
#Dataframe structure
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
```

In [7]:
```
#Seperate numeric and categorical columns
num_col = df.select_dtypes(include='number').columns
cat_col = df.select_dtypes(include='object').columns

print("Numeric Columns:")
print("*"*50)
print(num_col)
print("\nCategorical columns:")
print("*"*50)
print(cat_col)
```

```
Numeric Columns:
**************************************************
Index(['Minimal price (gross) [PLN]', 'Engine power [KM]',
       'Maximum torque [Nm]', 'Battery capacity [kWh]', 'Range (WLTP) [km]',
       'Wheelbase [cm]', 'Length [cm]', 'Width [cm]', 'Height [cm]',
       '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]',
       'Acceleration 0-100 kph [s]', 'Maximum DC charging power [kW]',
       'mean - Energy consumption [kWh/100 km]'],
      dtype='object')

Categorical columns:
**************************************************
Index(['Car full name', 'Make', 'Model', 'Type of brakes', 'Drive type'], dtype
='object')
```

In [8]:
```python
# check for dupliate and missing values
print("Duplicate Values:", df.duplicated().sum())
print("Missing Values:")
print(df.isna().sum())
```

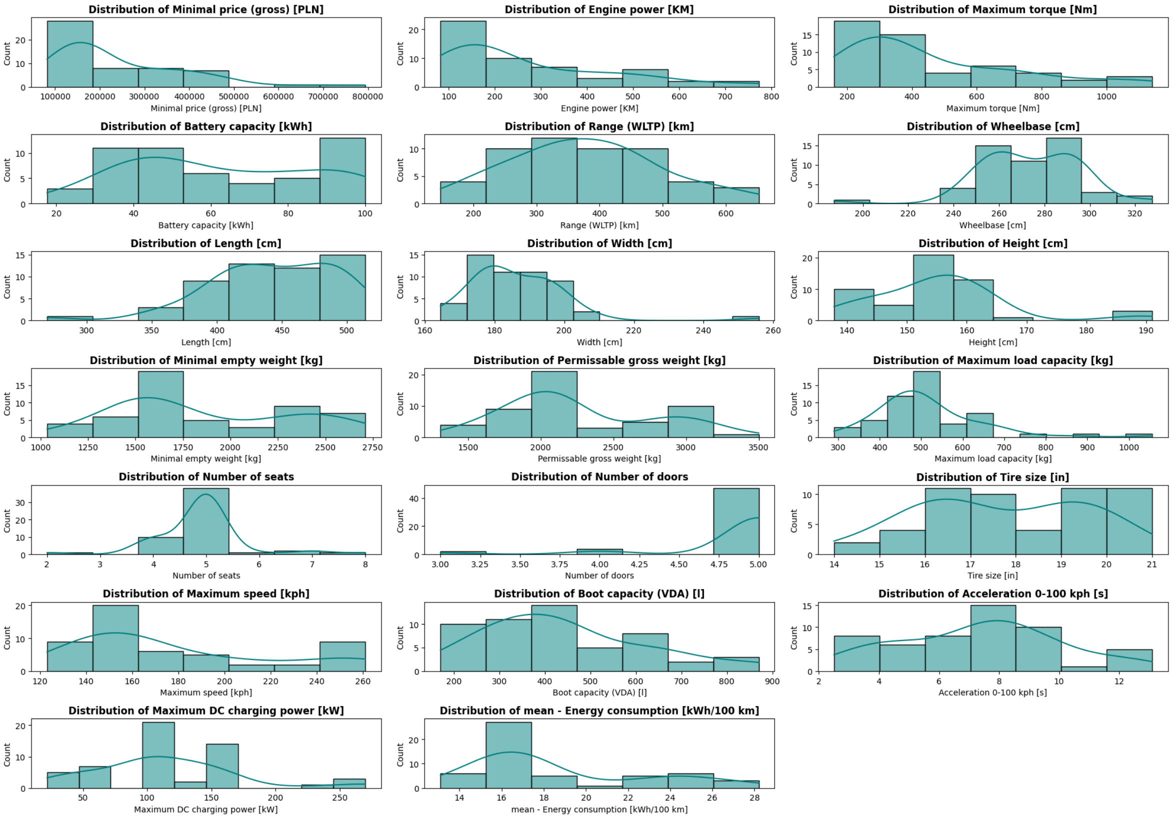
```
Duplicate Values: 0
Missing Values:
Car full name                              0
Make                                       0
Model                                      0
Minimal price (gross) [PLN]                0
Engine power [KM]                          0
Maximum torque [Nm]                        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
```

In [19]:
```python
# filling missing values for numeric column using median
for col in num_col:
    if df[col].isna().sum() > 0:
        df[col].fillna(df[col].median(), inplace=True)

#fill missing value of categorical column
for col in cat_col:
    if df[col].isna().sum() > 0:
        df[col].fillna(df[col].mode()[0], inplace = True)

#check for missing values
df.isna().sum()
```

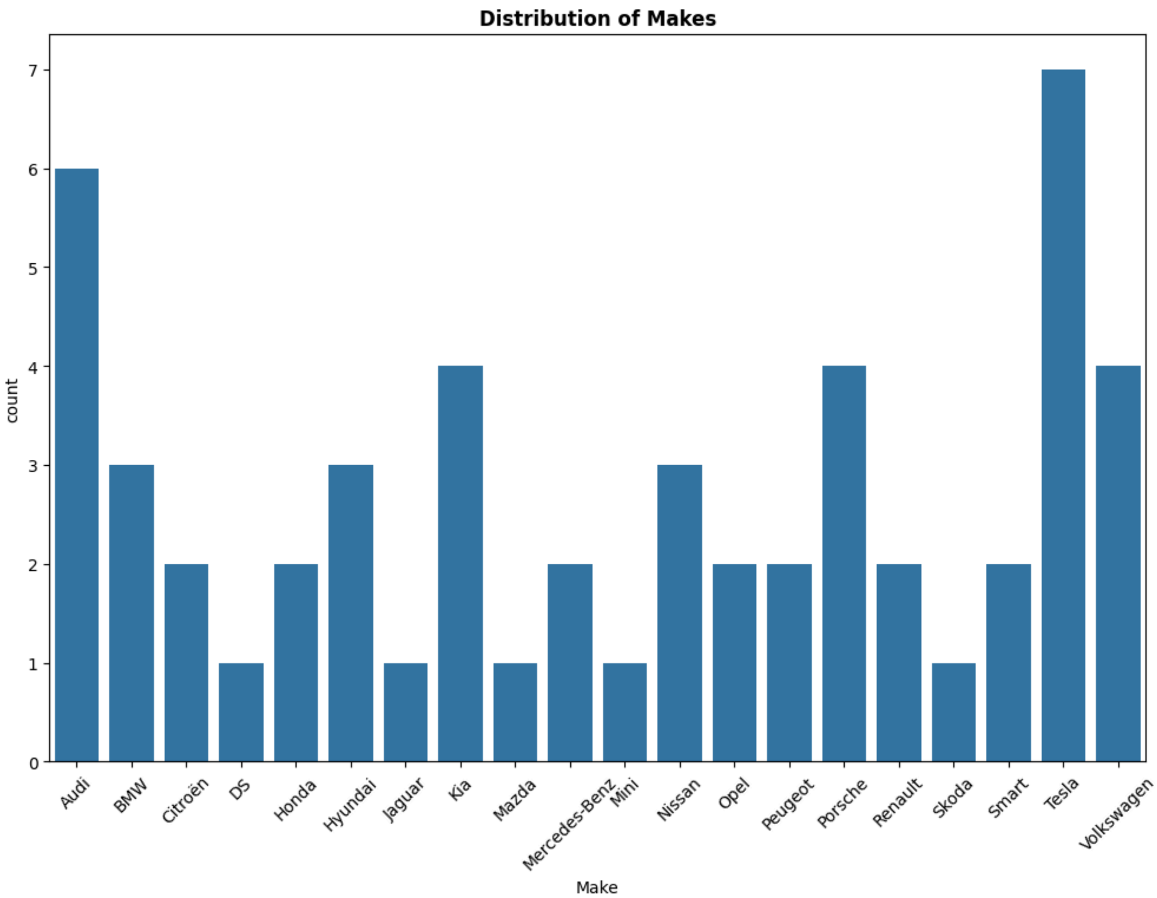
```
Out[19]: Car full name                           0
         Make                                    0
         Model                                   0
         Minimal price (gross) [PLN]             0
         Engine power [KM]                       0
         Maximum torque [Nm]                     0
         Type of brakes                          0
         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]           0
         Maximum load capacity [kg]              0
         Number of seats                         0
         Number of doors                         0
         Tire size [in]                          0
         Maximum speed [kph]                     0
         Boot capacity (VDA) [l]                 0
         Acceleration 0-100 kph [s]              0
         Maximum DC charging power [kW]          0
         mean - Energy consumption [kWh/100 km]  0
         dtype: int64
```

In [20]:

```
#plotting the histogram for numeric column
plt.figure(figsize=(20,14))

for i, col in enumerate(num_col, 1):
    plt.subplot((len(num_col) + 2) // 3, 3, i) #create enough row based on total
    sns.histplot(df[col], kde = True, color='teal')
    plt.title(f"Distribution of {col}", fontweight='bold')

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



In [18]:
```
#plotting for make columns
plt.figure(figsize=(12,8))
sns.countplot(x='Make', data=df)
plt.xticks(rotation=45)
plt.title('Distribution of Makes', fontweight = 'bold')
plt.show()
```



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

```
a) Your task is to filter out EVs that meet these criteria.
b) Group them by the manufacturer (Make).
c) Calculate the average battery capacity for each manufacturer.
```

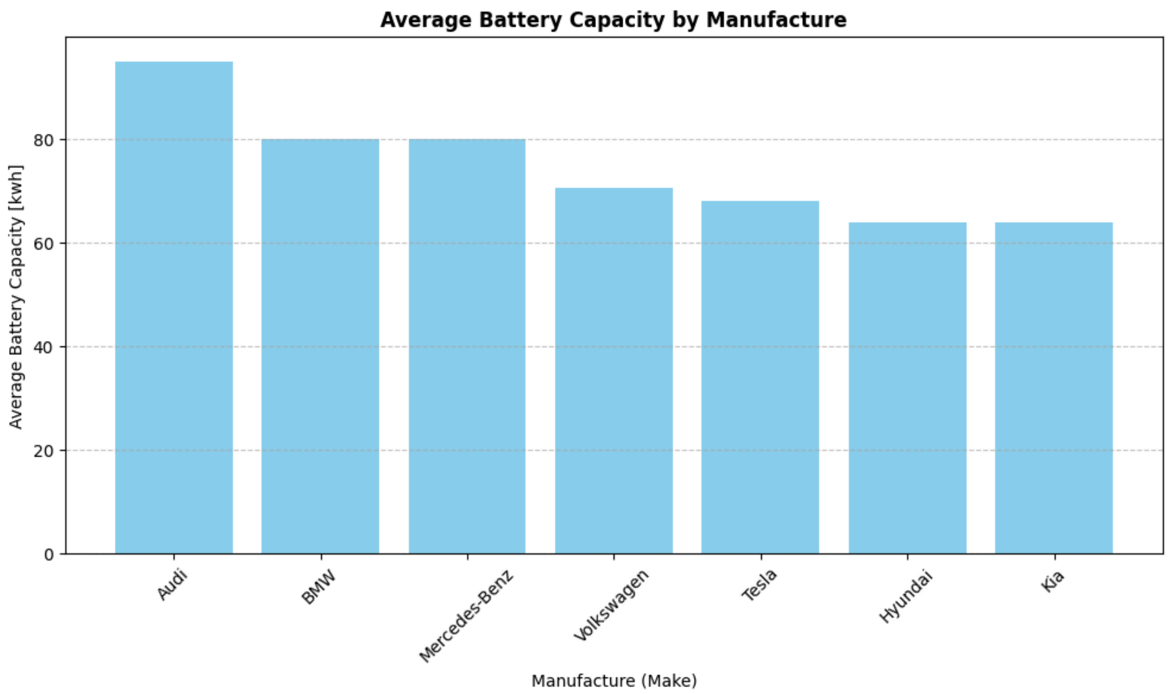
In [12]:

```python
# filter EVs based on given criteria
filter_df = df[(df['Minimal price (gross) [PLN]'] <= 350000) & (df['Range (WLTP)

# Group by make and calculate the average battery capacity
result = filter_df.groupby('Make')['Battery capacity [kWh]'].mean().reset_index(

#Display the result
print(result)
```

```
            Make  Battery capacity [kWh]
0           Audi               95.000000
1            BMW               80.000000
2        Hyundai               64.000000
3            Kia               64.000000
4  Mercedes-Benz               80.000000
5          Tesla               68.000000
6     Volkswagen               70.666667
```

In [13]:

```python
#sorting them by battery capacity
result =result.sort_values('Battery capacity [kWh]', ascending=False)

#plotting a bar chart
plt.figure(figsize=(10,6))
plt.bar(x = result['Make'], height = result['Battery capacity [kWh]'], color='sk
plt.xlabel('Manufacture (Make)')
plt.ylabel('Average Battery Capacity [kwh]')
plt.title('Average Battery Capacity by Manufacture', fontweight='bold')
plt.grid(axis='y',linestyle='--', alpha=0.7)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Methodology:

Filtering:

We filtered the dataset using the following conditions: Minimal price (gross) [PLN] ≤ 350,000 Range (WLTP) [km] ≥ 400

Grouping:

After filtering, we grouped the remaining EVs by their manufacturer (Make).

Aggregation:

For each manufacturer group, we calculated the average battery capacity using the column Battery capacity [kWh].

Visualization:

A bar chart was created to visualize the average battery capacity per manufacturer.

Graph

The bar chart shows that Audi leads with the highest average battery capacity.

Conclusion:

Manufacturers differ in how they balance cost, battery capacity, and range. This analysis helps customers identify which brands provide the best value within a defined budget, particularly if battery capacity is a priority.

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.

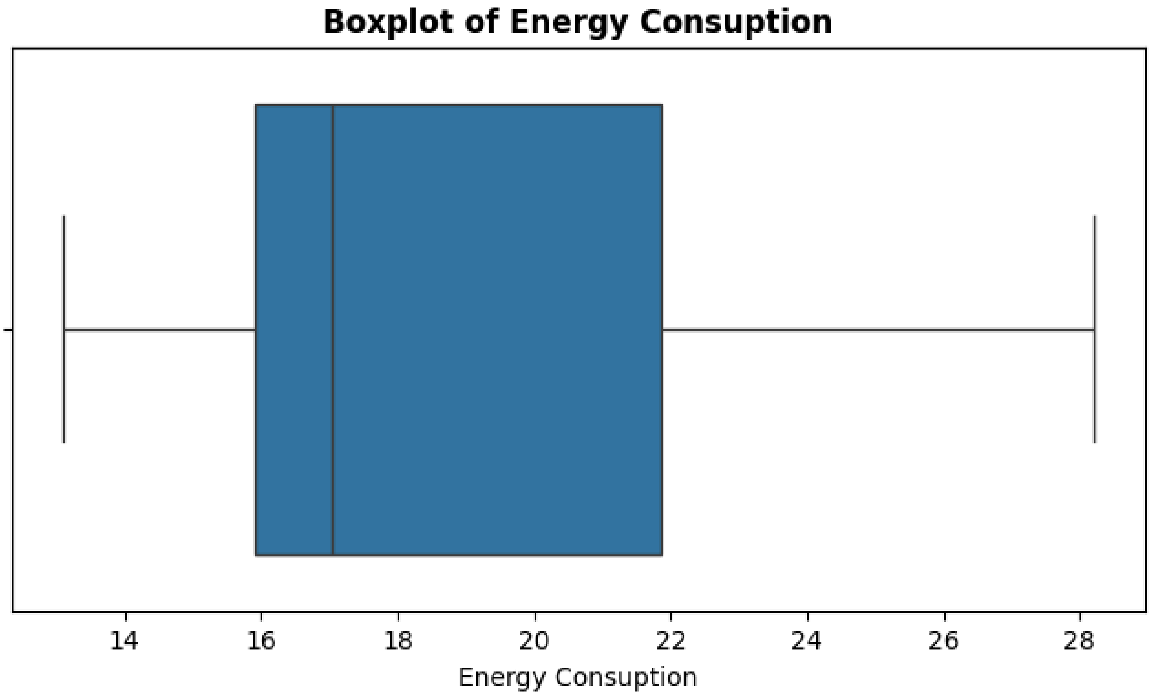
In [14]:
```
# Rename the columns for easier reference
df = df.rename(columns = {'mean - Energy consumption [kWh/100 km]' : 'Energy Con

# Calculate Q1 and Q3
Q1 = df['Energy Consuption'].quantile(0.25)
Q3 = df['Energy Consuption'].quantile(0.75)

# Calculate IQR (Interquartile Range)
IQR = Q3 - Q1

# Determine outliers
lower_bond = Q1 - 1.5 * IQR
upper_bond = Q3 + 1.5 * IQR

# Find Outliers
outliers = df[(df['Energy Consuption'] < lower_bond) | (df['Energy Consuption']

#Display the outliers
print(outliers[['Car full name', 'Energy Consuption']])
```

```
Empty DataFrame
Columns: [Car full name, Energy Consuption]
Index: []
```

In [15]:
```
# Displaying IQR and bonds
print(f"Q1: {Q1}")
print(f"Q3: {Q3}")
print(f"IQR: {IQR}")
print(f"Lower Bond: {lower_bond}")
print(f"Upper Bond: {upper_bond}")
```

```
Q1: 15.9
Q3: 21.85
IQR: 5.950000000000001
Lower Bond: 6.975
Upper Bond: 30.775000000000002
```

In [16]:
```
#plotting box plot
plt.figure(figsize=(8,4))
sns.boxplot(data=df, x='Energy Consuption')
plt.title('Boxplot of Energy Consuption', fontweight='bold')
plt.xlabel('Energy Consuption')
plt.show()
```



Methodology:

For easier readability and coding, the original column name was renamed to Energy Consumption.

To detect outliers, we used the Interquartile Range (IQR) method: Calculated the first (Q1) and third quartiles (Q3) of the Energy Consumption column.

Computed the IQR as Q3 - Q1.

Defined outlier bounds using the standard formula:

Lower bound = Q1 - 1.5 × IQR Upper bound = Q3 + 1.5 × IQR

EVs with energy consumption values outside these bounds were flagged as outliers. After applying the IQR method:

No statistical outliers were detected in the dataset. All EVs had energy consumption values within the range of approximately 6.97 to 30.78 kWh/100 km, which falls within the expected range for modern EVs.

Conclusion:

The dataset appears to be well-curated with no extreme anomalies in energy usage. This suggests:

Consistent vehicle design across models,

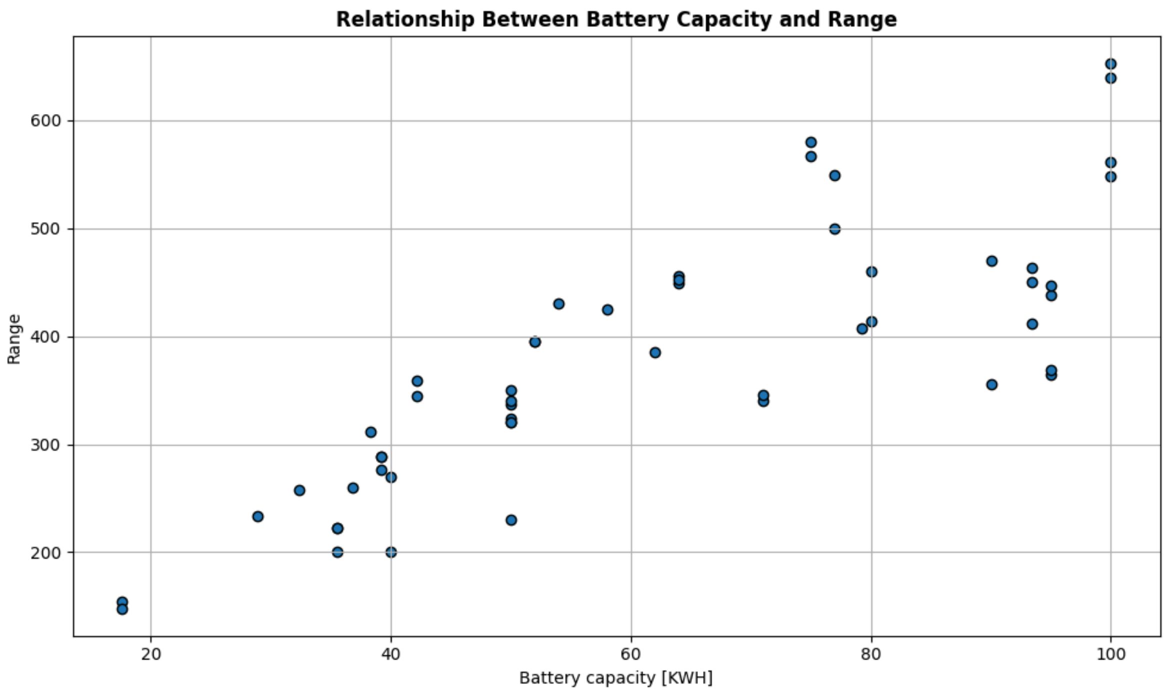
No major data entry errors in this column,

And that most EVs operate within a reasonable efficiency range.

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

```
a) Create a suitable plot to visualize.
b) Highlight any insights.
```

In [17]:
```python
#Display the scatterplot
plt.figure(figsize=(10,6))
plt.scatter(x = df['Battery capacity [kWh]'], y = df['Range (WLTP) [km]'], edgec
#Add levels and titles

plt.title('Relationship Between Battery Capacity and Range', fontweight='bold')
plt.xlabel('Battery capacity [KWH]')
plt.ylabel('Range')
plt.grid(True)
plt.tight_layout()
plt.show()
```



a). Visualization:

A scatter plot was used to visualize the relationship between battery capacity and range. Each point on the graph represents an Electric Vehicle, with:

```
X-axis: Battery capacity [kWh]
Y-axis: Range (WLTP) [km]
```

b). Highlight & insights.

1. Positive Trend: The plot shows a general upward trend, indicating that as battery capacity increases, the range tends to increase as well. This suggests a positive

correlation between the two variables.

2. Strength of the Relationship: --> While the relationship is generally positive, it's not perfectly linear—some EVs with similar battery capacities have noticeably different ranges. --> This variance might be influenced by other factors like energy efficiency, vehicle weight, Engine Power, Driving ondition.

3. Outliers: Some cars in the graph are far away from most of the other points. This means they behave differently from the rest. For example, a few cars with around 100 kWh battery capacity can go much farther than others with the same battery size.

This might be because they use energy more efficiently, or they're built in a way that helps them go farther on the same amount of power.

Conclusion:

There is a strong positive relationship between battery capacity and range, which aligns with expectations—larger batteries generally store more energy and enable longer driving distances

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.

In [18]:
```python
class EVselector:
    def __init__(self,ev_data):
        #Store the ev dataset
        self.data = ev_data

    def recommended(self):
        try:
            budget = float(input("Enter Your Budget: "))
            desired_range = float(input("Enter Your Range (km):"))
            battery_capacity = float(input("Enter the Battery Capacity (kWh):"))
        except ValueError:
            print("Invalid input. Please Enter Number")
            return

        # Filter EVs that meet the user's criteria
        filters = self.data[
        (self.data['Minimal price (gross) [PLN]'] <= budget) &
        (self.data['Range (WLTP) [km]'] >= desired_range) &
        (self.data['Battery capacity [kWh]'] >= battery_capacity)
        ]

        if filters.empty:
            print("No EVs match your criteria. Try adjusting your inputs.")
            return

        top_ev = filters.sort_values(by = 'Range (WLTP) [km]', ascending = False
```

```
        print("\n Top 3 Recommended EV:")
        print(top_ev[['Car full name', 'Minimal price (gross) [PLN]', 'Range (WL

# Run the class
selector = EVselector(df)
selector.recommended()
```

```
Top 3 Recommended EV:
               Car full name  Minimal price (gross) [PLN]  Range (WLTP) [km]  \
40   Tesla Model 3 Long Range                       235490                580
41  Tesla Model 3 Performance                       260490                567
48       Volkswagen ID.3 Pro S                       179990                549

   Battery capacity [kWh]
40                   75.0
41                   75.0
48                   77.0
```

In this task, I developed an EV recommendation system using a Python class named EVselector. The purpose of this class is to help users find the most suitable electric vehicles (EVs) based on their personal preferences.

How It Works:

```
The program prompts the user to enter their budget, desired
range (km), and minimum battery capacity (kWh).
It filters the EV dataset to find vehicles that match all three
criteria.
If any matching EVs are found, it returns the top three based on
highest driving range.
If no matches are found, the system notifies the user to adjust
their inputs.
```

Task 5: Inferential Statistics – Hypothesis Testing:

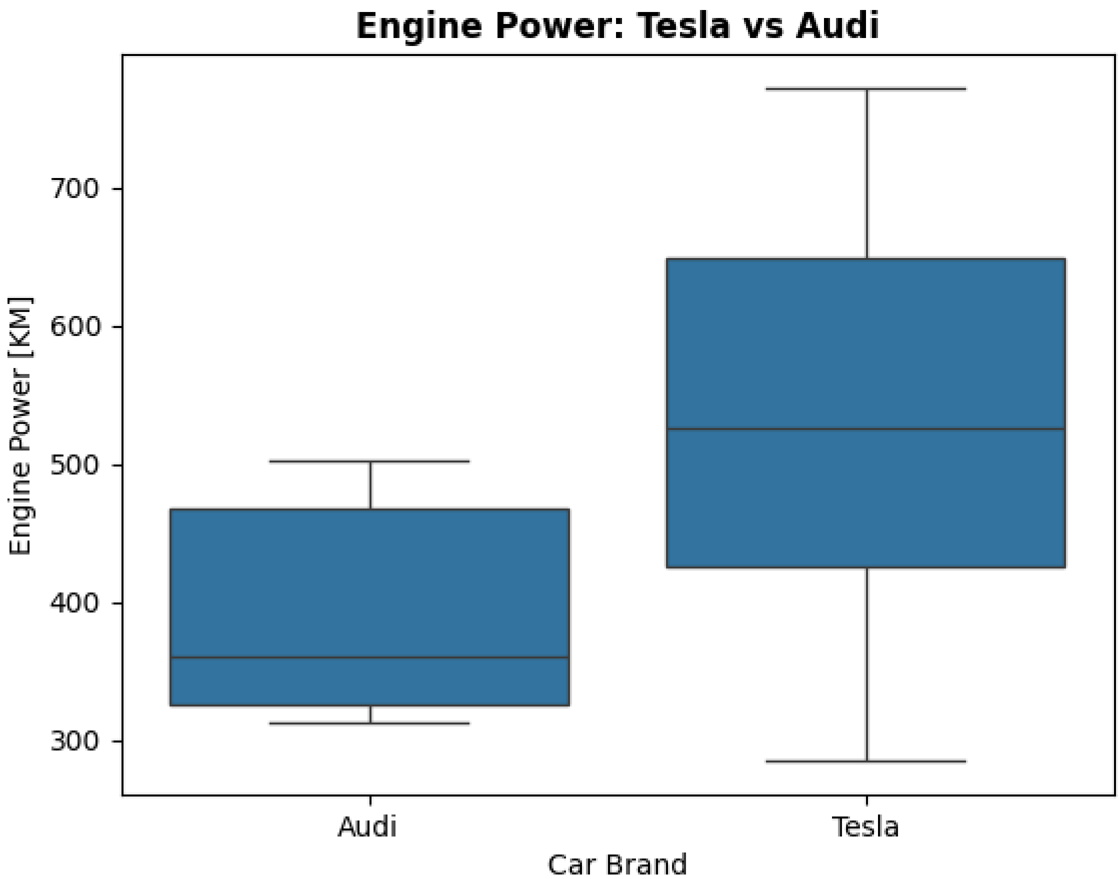
Test whether there is a significant difference in the average Engine power [KM] of vehicles manufactured by two leading manufacturers i.e. Tesla and Audi. What insights can you draw from the test results? Recommendations and Conclusion: Provide actionable insights based on your analysis. (Conduct a two sample t-test using ttest_ind from scipy.stats module)

In [27]:
```
# filter the data from tesla and audi
tesla_power = df[df['Make'] == 'Tesla']['Engine power [KM]']
audi_power = df[df['Make'] == 'Audi']['Engine power [KM]']

# perform two sample t-test
t_state, p_value = ttest_ind(tesla_power, audi_power, equal_var = False)  #Welch

#Print the result
print("T-statistic:", round(t_state,3))
print("P-value:", round(p_value,3))

# report format
alpha = 0.05 # significance level
```

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

```
T-statistic: 1.794
P-value: 0.107
No significant difference in engine power between Tesla and Audi.
```

In [21]:
```python
#filter only tesla and audi rows
subset = df[df['Make'].isin(['Tesla','Audi'])]

#plotting the box plot
sns.boxplot(x='Make',y='Engine power [KM]', data=subset)
plt.title('Engine Power: Tesla vs Audi', fontweight='bold')
plt.xlabel('Car Brand')
plt.ylabel('Engine Power [KM]')
plt.show()
```



Method:

We conducted a two-sample independent t-test using ttest_ind from scipy.stats. The comparison was made using the Engine power [KM] column for both Tesla and Audi EVs.

Hypotheses:

Null Hypothesis (H_0): Tesla and Audi EVs have similar average engine power. Alternative Hypothesis (H_1): Tesla and Audi EVs have different average engine power.

Results:

T-statistic: 1.794 P-value: 0.107 Significance level (α): 0.05

Since the p-value (0.107) is greater than 0.05, we fail to reject the null hypothesis. This means there is no statistically significant difference in average engine power between Tesla and Audi EVs based on the data.

Conclusion:

While the data sample graph shows Tesla vehicles appear to have higher engine power than Audi, the statistical test ($p = 0.107$) shows that this difference is not statistically significant. Therefore, we cannot confidently say there is a real difference in average engine power based on this data.

Video link

Electric_Vehicle