```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind
```

loading data in pandas data frame

```
df = pd.read_csv("Auta elektryczne.csv")
```

```
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	Number of doors	Tire size [in]
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	5	19
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	5	19
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	5	20
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	5	19
	Audi e-tron		e-tron Sportback				disc									

checking data-types

```
df.dtypes
```

```
Car full name                               object
Make                                        object
Model                                       object
Minimal price (gross) [PLN]                  int64
Engine power [KM]                            int64
Maximum torque [Nm]                          int64
Type of brakes                              object
Drive type                                  object
Battery capacity [kWh]                     float64
Range (WLTP) [km]                            int64
Wheelbase [cm]                             float64
Length [cm]                                float64
Width [cm]                                 float64
Height [cm]                                float64
Minimal empty weight [kg]                    int64
Permissable gross weight [kg]              float64
Maximum load capacity [kg]                 float64
Number of seats                              int64
Number of doors                              int64
Tire size [in]                               int64
Maximum speed [kph]                          int64
Boot capacity (VDA) [l]                    float64
Acceleration 0-100 kph [s]                 float64
Maximum DC charging power [kW]               int64
mean - Energy consumption [kWh/100 km]     float64
dtype: object
```

checking duplicates rows

```
df.duplicated().any()
```

```
np.False_
```

checking null 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.isnull().sum()
```

```
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
```

Handling null value

```
df["Type of brakes"] = df["Type of brakes"].fillna("NA")
df["Boot capacity (VDA) [l]"] = df["Boot capacity (VDA) [l]"].fillna(df["Boot capacity (VDA) [l]"].median())
df["Permissable gross weight [kg]"] = df["Permissable gross weight [kg]"].fillna(
    df["Permissable gross weight [kg]"].median())
```

```
df["Maximum load capacity [kg]"] = df["Maximum load capacity [kg]"].fillna(
    df["Maximum load capacity [kg]"].median())
df["Acceleration 0-100 kph [s]"] = df["Acceleration 0-100 kph [s]"].fillna(
    df["Acceleration 0-100 kph [s]"].median())
df["mean - Energy consumption [kWh/100 km]"] = df["mean - Energy consumption [kWh/100 km]"].fillna(
    df["mean - Energy consumption [kWh/100 km]"].median())
```

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

```
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
```

```
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]           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]                 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 :- customer has a budget of 350,000 PLN and wants an EV with a minimum range of 400 km.

a.) Filtered cars having maximum range of 400Km under 350,000 Price

```
df_filtered = df[(df["Minimal price (gross) [PLN]"]<=350000) & (df["Range (WLTP) [km]"]>=400)]
df_filtered= df_filtered.sort_values(by=["Range (WLTP) [km]", "Minimal price (gross) [PLN]"], ascending = [False, True])
df_filtered
```

	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	Nu d
40	Tesla Model 3 Long Range	Tesla	Model 3 Long Range	235490	372	510	disc (front + rear)	4WD	75.0	580	...	NaN	NaN	5	
41	Tesla Model 3 Performance	Tesla	Model 3 Performance	260490	480	639	disc (front + rear)	4WD	75.0	567	...	NaN	NaN	5	
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	412.0	5	
49	Volkswagen ID.4 1st	Volkswagen	ID.4 1st	202390	204	310	disc (front) + drum (rear)	2WD (rear)	77.0	500	...	2660.0	661.0	5	
8	BMW iX3	BMW	iX3	282900	286	400	disc (front + rear)	2WD (rear)	80.0	460	...	2725.0	540.0	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	
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	
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	
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	NaN	5	
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	540.0	5	
22	Mercedes-Benz EQC	Mercedes-Benz	EQC	334700	408	760	disc (front + rear)	4WD	80.0	414	...	2940.0	445.0	5	

b.) Group by the manufacturer

```
grouped = df_filtered.groupby(["Make", "Model", "Car full name"])
grouped
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000013BBD457ED0>
```

c.) Average battery capacity for each manufacturer.

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

```
Make
Audi          95.00
BMW           80.00
Hyundai       64.00
Kia           64.00
```

```
Mercedes-Benz    80.00
Tesla            68.00
Volkswagen       70.67
Name: Battery capacity [kWh], dtype: float64
```

Start coding or generate with AI.

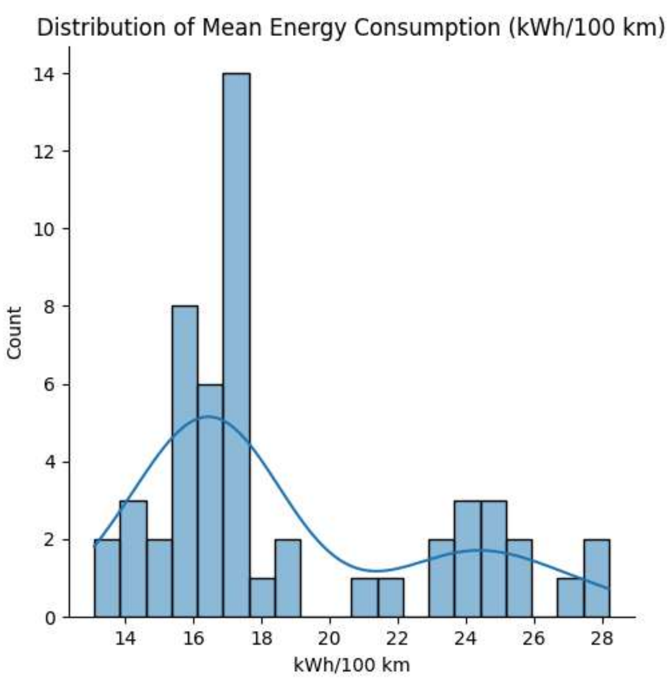
TASK 2 :- Finding the outliers in the mean - Energy consumption [kWh/100 km] column

```
df.head(1)
```

	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	Tire size [in]	Maxi sp [k
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	5	19	2

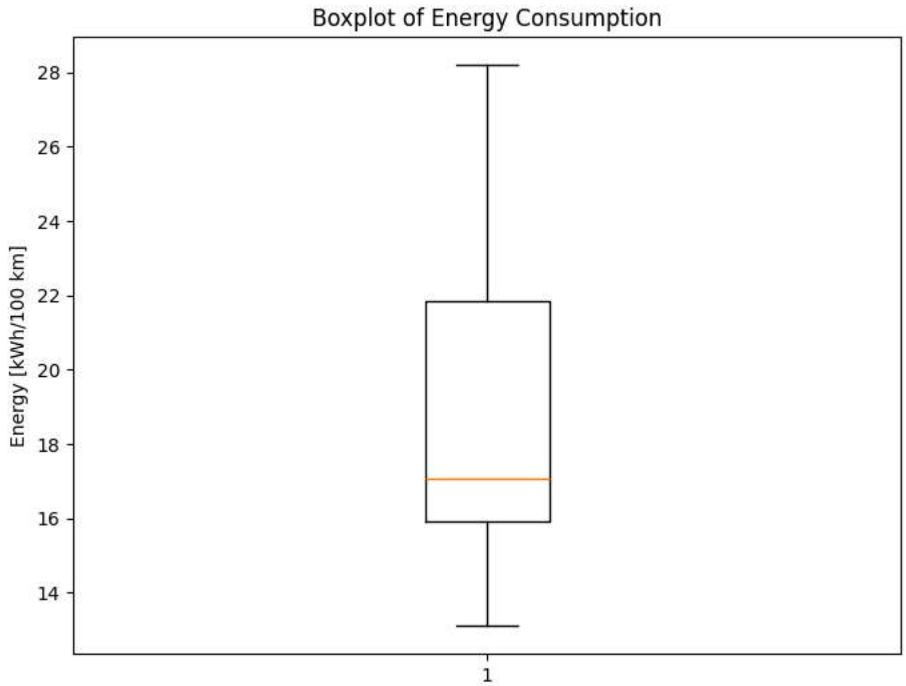
```
## checking distribution of data column
col_energy = df["mean - Energy consumption [kWh/100 km]"]

sns.displot(col_energy, kde = True, bins=20)
plt.title("Distribution of Mean Energy Consumption (kWh/100 km)")
plt.xlabel("kWh/100 km")
plt.ylabel("Count")
plt.show()
```



BoxPlot to detect Outliers

```
plt.figure(figsize=(8, 6))
plt.boxplot(col_energy)
plt.title("Boxplot of Energy Consumption")
plt.ylabel("Energy [kWh/100 km]")
plt.show()
```



- No outliers are visible under the standard IQR rule, confirming a consistent dataset.
- The slightly higher values (27–28) are on the upper whisker but still within the non-outlier range.

Detecting Outliers with IQR Range

```
# detecting outlier with IQR

Q1 = col_energy.quantile(0.25) #first quartile
Q3 = col_energy.quantile(0.75) #third quartile

IQR = (Q3 - Q1) #Interquartile Range (middle 50% of data)

lower_bound = Q1 - 1.5 * IQR #lower bound
upper_bound = Q3 + 1.5 * IQR #upper bound

outliers = df[ (col_energy < lower_bound) | (col_energy > upper_bound) ]


outliers
```

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	Tire size [in]	Maximum speed [kph]

- With the standard 1.5×IQR rule, the boxplot showed no outliers.
- To check for extreme cases, we applied 1.0×IQR rule.
- Under this rule, we identified one high outlier with mean energy consumption of 28.2 kWh/100 km.

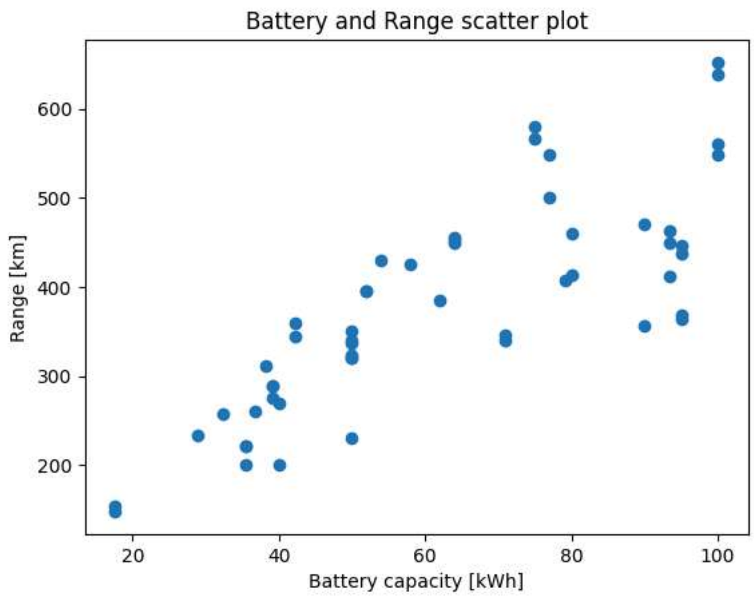
Start coding or generate with AI.

TASK 3 :- if there's a strong relationship between battery capacity and range.

a.) Visualization

```
x = df["Battery capacity [kWh]"]
y = df["Range (WLTP) [km]"]

plt.scatter(x,y)
plt.xlabel("Battery capacity [kWh]")
plt.ylabel("Range [km]")
plt.title("Battery and Range scatter plot")
plt.show()
```



b.) Insights

Positive Relationship

- As **battery capacity (kWh)** increases, the **range (km)** also increases.
- This suggests a direct correlation: higher capacity batteries generally provide higher driving range.

Outliers

- A few points deviate from the trend. some cars with large battery capacity but having lower range.
- These could be affect of **less efficiency, higher weight, or design differences**.

Conclusion

- There is a **strong positive correlation** between battery capacity and range.
- However, range is **not determined solely** by battery size.
- Factors like **vehicle efficiency, aerodynamics, and energy consumption** also matters and may explain the outliers.

TASK 4 :- Build an EV recommendation Class

```
df.head(1)
```

	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	Tire size [in]	Maxi sp [k
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	5	19	2

```
class EvRecommender:
    def __init__(self, ev_df):
        self.ev_df = ev_df

    def recommend(self, budget, min_range, min_battery):
        filtered_data = self.ev_df[
            (self.ev_df["Minimal price (gross) [PLN]"] <= budget) &
            (self.ev_df["Range (WLTP) [km]"] >= min_range) &
            (self.ev_df["Battery capacity [kWh]"] >= min_battery)
            ]
        filtered_data = filtered_data.sort_values( by = ["Minimal price (gross) [PLN]", "Range (WLTP) [km]"], ascending = [True, False] )
        result = filtered_data.head(3)
        return result

recommender = EvRecommender(df)
budget = float(input("Enter your budget:"))
min_range = float(input("Enter minimum Range required [km]"))
min_battery = float(input("Enter minimum battery Required [kWh]"))

recommender.recommend(budget, min_range, min_battery)
```

```
Enter your budget: 500000
Enter minimum Range required [km] 400
Enter minimum battery Required [kWh] 100
```

	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
42	Tesla Model S Long Range Plus	Tesla	Model S Long Range Plus	368990	525	755	disc (front + rear)	4WD	100.0	652	...	NaN	NaN	5	5
44	Tesla Model X Long Range Plus	Tesla	Model X Long Range Plus	407990	525	755	disc (front + rear)	4WD	100.0	561	...	NaN	NaN	7	5
43	Tesla Model S Performance	Tesla	Model S Performance	443990	772	1140	disc (front + rear)	4WD	100.0	639	...	NaN	NaN	5	5

TASK 5 :- Hypothesis Testing: If there is a significant difference in the average Engine power between Tesla and Audi.

```
# Extract Tesla and Audi data
tesla_power = df[df['Make'] == 'Tesla']['Engine power [KM]']
audi_power = df[df['Make'] == 'Audi']['Engine power [KM]']
```

Null Hypothesis: There is no significant difference between the average engine power of Tesla and Audi.

Alternate Hypothesis: There is a significant difference between the average engine power of Tesla and Audi.

Checking the mean values and sample data to perform the t-test analysis.

```
print("Mean for audi manufacturer is:",audi_power.mean())
print(audi_power)
print("_"*80)
print("Mean for tesla manufacturer is:",tesla_power.mean())
print(tesla_power)
```

```
Mean for audi manufacturer is: 392.0
0    360
1    313
2    503
3    313
4    360
5    503
Name: Engine power [KM], dtype: int64
________________________________________________________________________________
Mean for tesla manufacturer is: 533.0
39    285
40    372
41    480
42    525
43    772
44    525
45    772
Name: Engine power [KM], dtype: int64
```

```
# Perform independent t-test (Welch’s test)
t_stat, p_val = ttest_ind(tesla_power, audi_power, equal_var=False)

print("T-value:", t_stat)
print("P-value:", p_val)
```

```
T-value: 1.7939951827297178
P-value: 0.10684105068839565
```

t-value is 1.7939951827297178

p-value is 0.10684105068839565

Since we got p-value (0.1068) > 0.05 (alpha), we fail to reject the null hypothesis.

This means there is no significant difference in the average engine power of Tesla and Audi based on the given data.

Recommendation

We may get more precise and reliable results if we have a larger dataset.

For decision-making, buyers should also look at other factors like range, battery life, charging options, and overall performance instead of only engine power.

From a business point of view, showing a mix of power, efficiency, and features gives more value to customers than just focusing on engine power.

Conclusion

Even tho Tesla shows a higher average engine power compared to Audi but difference is not statistically significant because (p-value = 0.106 > 0.05).

This means the observed difference may be due to random variation rather than an actual performance gap.

Therefore, we cannot conclude that one brand consistently has higher engine power than the other based only on this dataset.

Double-click (or enter) to edit

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.