

Prompting AI chatbots for help

Large Language Models (LLMs) can be useful tools in data science. In this document we will:

- Find solutions by generating code for your needs.
- Explain code to boost your understanding and confidence.
- Fix bugs in your code.

By the end, you'll have a toolkit to supercharge your projects with LLMs.

1. Using LLMs to find solutions

Need to whip up some Python code fast? LLMs can generate snippets for common tasks.

How to do it:

- Be specific: tell the LLM exactly what you need (e.g., **column names**, conditions).
- Pro Tip: include **the first few rows** of your dataframes.

Example: TSV file

Prompt: "How do I load a TSV file named 'sales_data.tsv' into a Pandas DataFrame."

Why good: Specific with exact filename.

LLM response:

To load a TSV (Tab-Separated Values) file named `sales_data.tsv` into a Pandas DataFrame, you can use the `pd.read_csv()` function with the `sep='\t'` parameter, which specifies that the values are separated by tabs. Here's the code to load the file:

```
import pandas as pd

# Load the TSV file into a DataFrame
df = pd.read_csv('sales_data.tsv', sep='\t')

# Display the first few rows of the DataFrame
print(df.head())
```

Example: filter rows

Prompt: "In Pandas, how can I filter rows where 'sales' is above 500 and below or equal to 1000? Use query()."

Why good: Specific with exact **column** name. Suggests a preferred method.

LLM Response:

You can use the `query()` method in Pandas to filter rows based on conditions. Here's how you can filter rows where the 'sales' column is above 500 and less than or equal to 1000:

```
df.query('sales > 500 and sales <= 1000')
```

Example: weak prompt

You have a dataframe shown below. You want to create a line graph, with 1 line for each category.

```
import pandas as pd
df = pd.read_csv('global-energy-use.csv')
df
```



	year	category	energy_use
0	1980	Bioenergy	29.940440
1	1980	Biofuels	0.118030
2	1980	Coal	75.087394
3	1980	Gas	51.253059
4	1980	Hydro	18.433686
...
425	2022	Nuclear	24.128422
426	2022	Oil	190.690521
427	2022	Other	8.689707
428	2022	Solar	12.413654
429	2022	Wind	19.755361

430 rows × 3 columns



Weak prompt: "I have data showing energy_use for each category by year. How do I create a line plot using pandas and matplotlib?"

Why weak: Not specific enough.

LLM response:

You can create a line plot using `pandas` and `matplotlib` by following these steps:

1. Load Your Data

Make sure your dataset is in a pandas DataFrame. Your data might look something like this:

Year	Residential	Commercial	Industrial	Transportation
2000	100	200	300	400
2001	110	210	290	420
2002	120	215	280	430

2. Plot the Data

Here's how you can create a line plot:

```
import pandas as pd
import matplotlib.pyplot as plt
```

This prompt did not work. Your dataframe has a category column, not separate columns for each category!

Example: better prompt

Let's write a better prompt. We highly recommend that you **include a few rows from your dataframes** in prompts.

Better prompt:

"I have data showing energy_use for each category by year. How do I create a line plot using pandas and matplotlib.

Here is the dataframe:

year	category	energy_use
0	1980 Bioenergy	29.940440
1	1980 Biofuels	0.118030
2	1980 Coal	75.087394"

Why better: Copying in the **first few lines** of the **dataframe** gives the LLM more context for a better response.

LLM Response:

You can create a line plot using `pandas` and `matplotlib` by following these steps:

Steps:

1. Pivot the dataframe so that each category becomes a separate column.
2. Use `matplotlib` to plot the data.

```
import pandas as pd
import matplotlib.pyplot as plt

# Pivot the dataframe
df_pivot = df.pivot(index="year", columns="category", values="energy_use")

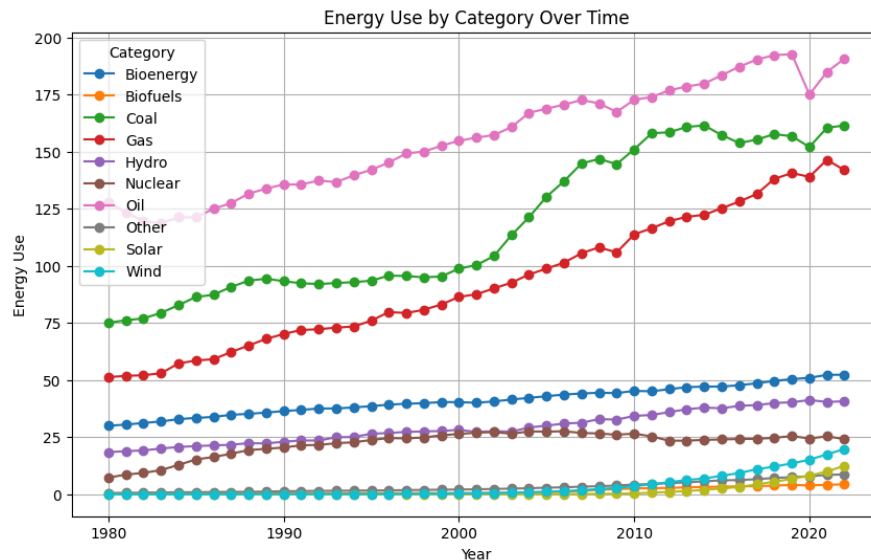
# Plot
df_pivot.plot(kind="line", marker="o", figsize=(10, 6))

# Customize plot
plt.xlabel("Year")
plt.ylabel("Energy Use")
plt.title("Energy Use by Category Over Time")
plt.legend(title="Category")
plt.grid(True)

# Show plot
plt.show()
```

This seems to be a better response. Let's try the code!

Below is the graph using the LLM generated code. We live in exciting times!



2. Using LLMs to explain code

Ever stared at a line of code and thought, what is this doing? LLMs can explain code to you step by step—helping you level up your coding skills.

How to do it:

- Ask for specifics.
- Pro Tip: paste in **code snippets**.

Example: `df.pivot()`

From the earlier example, the LLM used a method you may not be familiar with, `df.pivot()`.

Prompt:

```
"df_pivot = df.pivot(index='year', columns='category', values='energy_use')
```

Here is an example dataframe:

	year	category	energy_use
0	1980	Bioenergy	29.940440
1	1980	Biofuels	0.118030
2	1980	Coal	75.087394

I do not understand what `df.pivot()` does. Please explain."

Why good: Asks a **specific** question, provides a **code snippet**, and includes the **first few rows** of the dataframe. Specifically asks about `df.pivot()`.

LLM Response:

The `df.pivot` method in pandas reshapes your DataFrame by turning rows into columns based on specified keys.

In your code: `python df_pivot = df.pivot(index='year', columns='category', values='energy_use')`

- `index='year'` : Keeps 'year' as the row index (e.g., 1980 stays as a row label).
- `columns='category'` : Takes unique values from the 'category' column (Bioenergy, Biofuels, Coal) and makes them new column headers.
- `values='energy_use'` : Fills the table with the 'energy_use' values corresponding to each year-category pair.

From your sample DataFrame:

	year	category	energy_use
0	1980	Bioenergy	29.940440
1	1980	Biofuels	0.118030
2	1980	Coal	75.087394

After pivoting, it becomes:

year	Bioenergy	Biofuels	Coal
1980	29.940440	0.118030	75.087394

Each category gets its own column, with `energy_use` values aligned under them for each year. It's a quick way to reorganize data for analysis or visualization.

In summary, using LLMs to **deepen your understanding** is a winning strategy!

3. Using LLMs to fix bugs

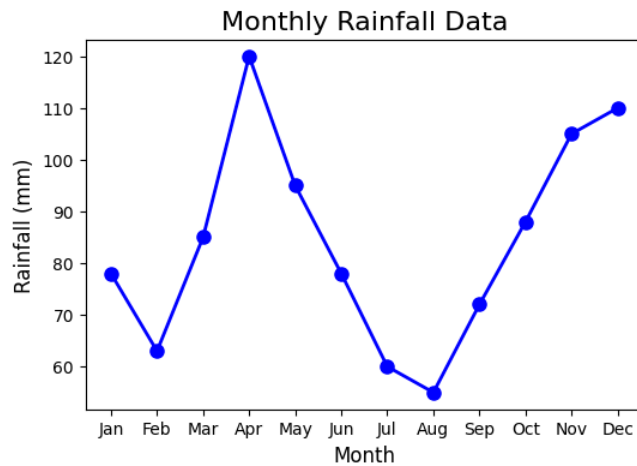
Bugs happen. While error messages can initially look intimidating, LLMs can often diagnose and fix issues if you provide it with enough context.

How to do it:

- Share the error: Copy-paste the **error message** or describe what's breaking.
- Add context: Provide **code snippets** that are not working.

Example: remove spines

You have a graph as shown:



You want to remove the top and right spines. You run the following code:

```
plt.spines(['top', 'right']).isFalse()  
plt.show()
```

You get a scary looking error!

```
-----  
AttributeError                                Traceback (most recent call last)  
<ipython-input-2-ca3b8e279b90> in <cell line: 0>()  
    18  
    19  
>>> 20 plt.spines(['top', 'right']).isFalse()  
    21 plt.show()  
  
AttributeError: module 'matplotlib.pyplot' has no attribute 'spines'
```

Here is how you can use LLMs to help you:

Prompt:

```
"plt.spines(['top', 'right']).isFalse()
```

This code did not work.

Here is the error message: `AttributeError: module 'matplotlib.pyplot' has no attribute 'spines'`

What is wrong with this code?"

Why good: Provides **code snippet** and **error message** to provide enough context to the LLM

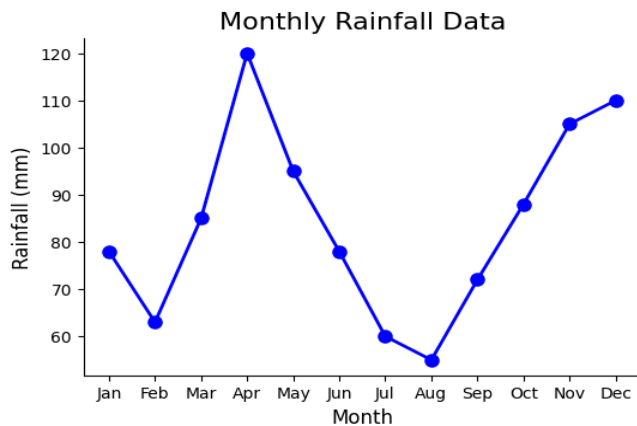
LLM response:

Your code is incorrect because `plt.spines` does not exist. Instead, access the spines through the axes object:

```
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```

This correctly hides the top and right spines.

We now change the code and get the following graph. It works!



Note that there are different ways to solve this problem. Another option is:

```
ax = plt.gca()
ax.spines[['top', 'right']].set_visible(False)
```

Example: remove column

Sometimes code does not do what you want even if it runs without errors. For example, the code below does **not** drop the column 'element'.

```
df.drop(columns=['element'])
df.head(3)
```

	place	symbol	element	percent_mass
0	atmosphere	O	oxygen	78.084
1	atmosphere	N	nitrogen	20.946
2	atmosphere	Ar	argon	0.934

Prompt: "In the code below, why does it not remove the column 'element'?"

```
df.drop(columns=['element'])
df.head(3)
"
```

Why good: Specific and provides a **code snippet** to help guide the LLM.

LLM Response:

The issue is that `df.drop(columns=['element'])` **does not modify df in place** unless explicitly specified. By default, `.drop()` returns a new DataFrame with the column removed, but it does not change `df` itself.

Possible Solutions:

1. **Modify df in place** using `inplace=True`:

```
df.drop(columns=['element'], inplace=True)
```

2. **Reassign** the returned DataFrame:

```
df = df.drop(columns=['element'])
```

Explanation:

- `df.drop(columns=['element'])` **returns a new DataFrame** with the column removed.
- Since you didn't store this new DataFrame in `df`, the original `df` remains unchanged.
- Using `inplace=True` modifies `df` directly.

Both those solutions work!

```
df = df.drop(columns=['element'])
df.head(3)
```

	place	symbol	percent_mass
0	atmosphere	O	78.084
1	atmosphere	N	20.946
2	atmosphere	Ar	0.934

4. Conclusion

LLMs are powerful tools to help you progress on your data science journey.

If their response is **incorrect** or not helpful the first time, modify the prompt or ask follow up questions. Use phrases such as "that did not work", providing what you want instead.

Keep in mind that LLM solutions do not always provide the ideal solution or follow best practice, even if they work.

Finally, using LLMs to **deepen your own understanding** is a **winning** strategy.