



Using LLMs as Coding Assistants

Practical Guidance for Python, Data Analysis, and Machine Learning

For use in AI subjects.
Written by **Vitali Alexeev**
UTS Business School
vitali.alexeev@uts.edu.au

Purpose

This guide shows how to use ChatGPT (or any other LLM) effectively as a *coding assistant* to help you write, debug, and understand Python. It complements lectures and labs by providing prompt templates, good practice, and examples. The aim is not only to complete tasks but to **deepen your understanding** of programming, data analysis, and machine learning.

1 Ground Rules & Academic Integrity

- **Learning first.** Use ChatGPT to *understand* code, not to bypass learning. You remain responsible for the work you submit.
- **Prompt Log.** Keep a simple log of prompts and note which ones worked best.
- **Privacy & data.** Do not paste confidential data, API keys, or personally identifying information.
- **Verification.** Always run and test generated code. ChatGPT can be wrong or invent non-existent functions/packages.

2 How to Ask for High-Quality Help

Prompt Checklist

Context (what you're doing) | **Goal** (what output you want) | **Constraints** (package versions, forbidden libraries) | **Minimal Code** (smallest snippet that fails) | **Error Message & Traceback** | **Environment** (Python version, OS, Colab/Jupyter) | **Output Format** (e.g., function, explanation, tests).

Prompt Template

You: "I am working on Jupyter on Windows with Python [3.13] and **pandas** [2.3]. I want a clean function to compute rolling volatility with a 90-day window and daily step. Please: (1) write a function with docstring and type hints, (2) include a simple test, and (3) explain complexity and edge cases. Here is my minimal input series and my attempt (which fails with a broadcasting error): `<paste short code>`. Error: `<paste traceback>`."

[**optional**] info in this prompt (such as exact Python or package versions) is not necessary at the beginner stage. However, once you progress further and start encountering issues related to package updates or version conflicts, including this info becomes very helpful for diagnosing problems quickly and accurately.

3 Core Use-Cases with Examples

Below are common ways ChatGPT can help, each with suggested prompts you can adapt quickly.

3.1 Generate Data for early experimentation

- Real-world data from financial datasets, news articles, or blogs is ideal, but during early experimentation you can boost your productivity with ChatGPT-generated data. This lets you dive into coding straight away, avoiding the delays of data cleaning and preprocessing that typically come with real data. The more precise you are about the data you ask for—its structure, fields, and value ranges—the better. Aim to make it mirror the real datasets you'll eventually work with.

Reminder
Synthetic data is only for practice. Don't use it for analysis, results, or conclusions — it won't reflect real market dynamics.

Example Prompt 1 (text data)

Generate 120 realistic sentences about *artificial intelligence in sustainable finance* across sub-themes (ESG scoring, climate risk, greenwashing, carbon markets). Output as a Python list of strings named **docs**. Avoid duplicates; vary sentence length.

Example Prompt 2 (numeric data)

Generate a synthetic dataset of 250 daily observations of financial data with the following columns: **'date'** (business days in 2022), **'stock_price'** (float, starting near 100, random walk with volatility 1–3%), **'trading_volume'** (integer, 1,000–50,000), **'esg_score'** (float, range 0–100). Output as a Pandas DataFrame named **'df'**. Ensure numerical realism (e.g., prices cannot go negative, ESG scores between 0 and 100).

Tip
Be precise in your request: define columns, ranges, and rules. The clearer the spec, the better the data.

Why functions? Functions are like reusable “mini-programs” inside your code. Instead of writing the same set of instructions again and again, you wrap them in a function and give that function a name. Later, whenever you need that behaviour, you just “call” the function. This makes your code shorter, easier to read, and much easier to fix if something goes wrong. Think of it as writing a recipe once, and then simply reusing it whenever you want to cook the same dish — you don’t need to reinvent the steps each time.

3.2 Generate Code & Functions

- One of the most powerful uses of ChatGPT is asking it to write code for you. By describing what you want in plain language, you can quickly get working examples without starting from scratch. The clearer and more specific your instructions are, the more useful the generated code will be. You can then copy, test, and adapt the output to suit your task. This is especially helpful when you are still learning, because you see not only the solution but also the structure and style of good code.

Listing 1: Anatomy of a Python function

```

1 # Function definition always starts with 'def' and ends with ':'
2
3 # (minimal example)
4 def myfunction(someinput):
5     message = "Hello, " + someinput + ". How are you?"
6     return message
7
8
9 # (with type hints and docstring)
10 def myfunction(someinput: str) -> str:
11     """
12     This is the docstring (documentation string).
13     It explains what the function does, its inputs and outputs.
14
15     someinput: str This specifies what type of input should the function expect.
16     -> str This specifies what type of output should be expect from the function.
17     """
18
19     # Function body (the code that runs when called)
20     message = "Hello, " + someinput + ". How are you?"
21
22     # The 'return' keyword sends back a value to the caller
23     return message
24
25
26 # --- Usage Example ---
27 print(myfunction("Vitali")) # Output: Hello, Vitali. How are you?

```

- Function templates.

Example Prompt 1 (plain and simple)

“Write a Python function called `split_into_sentences()` that takes a long piece of text and splits it into separate sentences. Show how to use the function with an example.”

This kind of prompt leaves room for interpretation, so the output might vary depending on how the model “feels” about the request. If you want more predictable results, it helps to be much more specific:

Example Prompt 2 (precise and technical)

“Write a Python function `split_into_sentences(text: str) → list[str]` that tokenizes a block of text into individual sentences using `spaCy` (small English model). If `spaCy` is unavailable, fall back to a simpler sentence splitter, ideally from `nltk`. Ensure that both implementations return consistent results (list of strings). Handle edge cases: (a) empty input should return an empty list, (b) text without punctuation should be treated as a single sentence. Include a usage example.”

Reminder

If you use `spaCy` or `nltk`, you may need to install them first with `!pip install spacy nltk` and download language models.

3.3 Debugging (🔧)

- **Error-to-fix workflow.** When you encounter an error in your code, you can paste the error message into ChatGPT, and it will guide you through understanding the issue and suggest solutions.

Don't panic when you see a red error block—Python is just telling you what went wrong. Focus on the last line of the error message.

Example Prompt

"When I run the code below, I get `ValueError: operands could not be broadcast together`. I'm trying to align two series by date before computing a ratio. Here is the *minimal* code and the full traceback. Please: (1) explain the cause, (2) fix the code, (3) show a small test verifying the fix." : `<paste code>`

- **Not-doing-what-I-expect.** If your code isn't doing what you expect, explain what the code is supposed to do, and ChatGPT can help you debug it line by line.

Example Prompt

"I'm building a bar chart of sentiment scores, but nothing displays in Jupyter. Here's my minimal example. Please debug line-by-line and explain inline where the display is suppressed or overwritten." : `<paste code>`

3.4 Code Explanations & Learning New Libraries

Ask ChatGPT to explain any part of the code you don't understand. Understanding how certain code blocks or algorithms work is essential to improving your coding skills. ChatGPT can explain code in simple terms and provide examples.

- **Break down unfamiliar code.**

Example Prompt (plain and simple)

"Explain what this piece of Python code does, line by line, in simple terms. Then show me a shorter or clearer way to write it if possible." : `<paste code>`

Example Prompt (precise and technical)

"Analyse the following Python code snippet in detail: 1. Provide a line-by-line explanation, including variable scope and data types. 2. Evaluate its time and space complexity. 3. Identify any anti-patterns or inefficiencies (e.g., repeated computations, unnecessary loops). 4. Refactor the code into a more Pythonic version using **list comprehensions** or **pandas** vectorisation where appropriate. 5. Include a unit test that validates the correctness of the refactored code." : `<paste code>`

List comprehension
A compact syntax for creating lists, for example,
`[x**2 for x in range(10)]`
which will result in output:
`[0,1,4,9,16,25,36,49,64,81]`.

- **Targeted library guidance.** If you're unfamiliar with a Python library (e.g., `matplotlib`, `seaborn`, etc), you can ask ChatGPT to generate concise, working examples. A vague or broad prompt might return something useful but incomplete, while a precise and technical prompt is more likely to produce exactly what you need.

Example Prompt (plain and simple)

"How do I use `pyLDAvis` to make an interactive topic model visualisation? Show me a simple example."

Example Prompt (precise and technical)

"Show a minimal, reproducible example of using `pyLDAvis` to visualise topics from an LDA model trained with `scikit-learn`. Include synthetic data, model training, and the visualisation call. Make sure the example runs end-to-end without needing any external dataset."

3.5 Enhancing Visualisations

■ Aesthetics & readability.

Example Prompt

"Given the seaborn bar plot below, help me: (a) increase font sizes, (b) add value labels to each bar, (c) rotate x-tick labels by 30°, (d) ensure colour-blind friendly palette, (e) add a footnote with data source."

■ Advanced features.

Example Prompt

"Convert this static matplotlib chart to Plotly with hover tooltips showing ticker, date, and z-scored return. Add a dropdown to filter by sector."

3.6 Optimising & Refactoring Code

■ Cleaner, faster, safer.

Example Prompt

"Refactor this function for clarity and performance. Add type hints, a doctest, and handle edge cases (empty input, NaNs, non-monotonic index). Provide Big-O complexity and a quick benchmark stub."

■ Best practices.

Example Prompt

"Suggest best practices for handling large text datasets (memory, streaming, chunking, caching). Provide short code patterns for each."

4 If you're feeling lazy

Too tired to think of a prompt? Let ChatGPT do the work for you. Just ask:

💡 A Prompt for a prompt

"Generate a prompt I can use to plot my data. Ask me any questions you need first."

ChatGPT will then follow up with a few clarifying questions:

💡 Follow-up Questions

To generate a good plotting prompt for you, I need a bit of context first. Could you tell me:

1. **Data format** – Is your data in a CSV, Excel, Pandas DataFrame, SQL table, or something else?
2. **Type of plot** – Do you want a time-series plot, bar chart, scatter plot, histogram, heatmap, or something else?
3. **Variables** – Which variables (columns) should be on the x-axis, y-axis, and (if applicable) used for grouping, colouring, or sizing?
4. **Goal** – Are you aiming to explore patterns, compare groups, show trends over time, or present results in a polished figure?
5. **Preferred library** – Do you want to use **Matplotlib**, **Seaborn**, **Plotly**, or another plotting library?
6. **Extra features** – Do you want labels, titles, legends, annotations, interactive features, or custom styling?

Once you give me these details, I'll create a precise, ready-to-use prompt that you can paste into ChatGPT (or another AI assistant) to generate the plotting code.

Do you want me to assume a finance/econometrics example dataset (like stock prices or ESG scores), or is this for your own dataset?

Yes—sometimes you can even outsource your laziness.

5 Limitations & Safe Use

⚠ Mind the Gaps

ChatGPT can:

- Misname functions or use older/newer APIs.
- Provide code that runs but is inefficient or numerically unstable.
- Sound confident when incorrect.

Mitigation: run, test, and compare with official docs; ask for citations and version-specific examples; use an alternative LLM with the same prompt(s).

6 Quick-Start Prompt Library

Scaffold a Starter Notebook

"Create a minimal Jupyter notebook outline for sentiment analysis: (1) environment checks, (2) data ingestion, (3) cleaning, (4) baseline (BoW/TF-IDF), (5) transformer baseline, (6) visualisations, (7) evaluation metrics, (8) discussion & next steps. Use placeholders for my data."

Explain This Code Block

"Explain this function step-by-step, then provide a vectorised **pandas** version with equal results. Discuss trade-offs and memory use."

Enhance My Plot

"Given this matplotlib figure, increase readability for print: (1) larger fonts, (2) thicker spines, (3) value labels, (4) colour-blind friendly palette, (5) add source footnote."

Refactor for Clarity

"Refactor this script into functions with clear responsibilities, type hints, and docstrings. Add a **main()** guard and simple CLI."

Library Mini-Tutorial

"Give a 15-line, version-pinned example of **scikit-learn TfidfVectorizer** on 5 short sentences. Show how to get the vocabulary and top weighted terms per document."

7 Compact Prompt Snippets (Copy & Adapt)

Corpus Generation

"Generate 100 diverse sentences about ESG risk, greenwashing, and climate disclosures suitable for topic modelling. Output as `docs = [...]` with no quotes inside sentences and no duplicates."

Debug Template

"My goal: <goal>. Expected result: <expected>. Actual result: <actual>. Environment: Python <ver>, **pandas** <ver>, platform <OS>. Here is a minimal reproducible example and the full traceback. Please identify the root cause and propose a fix with a short test."

Explain + Refactor

"Explain the following code line-by-line for a beginner. Then refactor it into a pure function with type hints, docstring, and tests. Discuss algorithmic complexity and memory behaviour."

Visualisation Upgrade

"Upgrade this seaborn figure for publication quality: font sizes, labelling, annotation of key points, consistent colour palette, and a note on data source. Show the exact code diff from my version."

8 Final Advice

⚠ Running Generated Code

Generated code may assume that required libraries are already available. Installing and re-running is part of the workflow.

- If you get a **ModuleNotFoundError**, install the missing package first (e.g., `!pip install plotly` in a notebook cell).
- **Restart the kernel** if imports behave oddly.

Be Specific, Iterative, and Verifiable

Provide context, set constraints, share *minimal* code, and ask for tests. Iterate: refine your prompt based on partial results. Always run the code and verify outputs.

Common Pitfalls

Over-long prompts without code; no error messages; copying large proprietary data; trusting output without checking; skipping documentation or tests.

A **kernel** is the engine that runs your code. Every time you execute a cell, the kernel processes it and keeps track of all the variables, functions, and imports in memory.

When you restart (reset) the kernel, it's like switching off the engine and starting it fresh: The memory is completely cleared. All variables, functions, imports, and intermediate results are forgotten.

Any code you wrote before still exists in the notebook, but the kernel no longer remembers that it was run. To continue after a restart, you need to re-run the cells that import libraries, define functions, and load data—essentially rebuilding the environment step by step.

Glossary of Coding Terms

API (Application Programming Interface)

A set of functions, methods, or endpoints that let different pieces of software communicate (e.g., OpenAI API, pandas API).

Broadcasting error

A NumPy or pandas error that occurs when arrays or series of incompatible shapes cannot be automatically aligned for operations.

Class / Object

A `class` defines a blueprint for creating objects (instances), which group related data and methods together.

Decorator

A special Python function (often starting with `@`) that modifies or extends the behaviour of another function or class.

Dependency / Environment

External packages or libraries that your code needs to run. Managing them often involves virtual environments and files like `requirements.txt`.

Docstring

A string literal placed at the start of a function, class, or module that explains what it does. Used by tools and IDEs for documentation.

Edge cases

Unusual or extreme inputs that test the limits of code logic (e.g., empty lists, division by zero, negative numbers).

Exception handling

Using `try ... except` blocks to catch and respond to errors gracefully.

Keyword argument (kwargs)

A function argument specified by name rather than position, e.g., `plt.plot(x, y, color="red")`.

List comprehension

A compact syntax for creating lists, e.g., `[x**2 for x in range(10)]`.

Minimal reproducible example

The smallest possible snippet of code that still produces the error or behaviour in question.

Package / Module

A module is a single `.py` file containing Python code; a package is a collection of modules with an `__init__.py`.

Parsing Breaking down text or data into a structured format (e.g., parsing JSON, parsing dates).

Pipeline A sequence of processing steps chained together, common in machine learning and data workflows.

Refactoring

Restructuring working code to make it clearer, more efficient, or more maintainable without changing its output.

Requirements file

A plain text file (usually `requirements.txt`) listing Python packages and their versions so an environment can be reproduced.

Syntax error

An error caused by invalid code structure, caught before execution (e.g., missing colon).

Traceback

The error report printed when Python raises an exception; shows where and why the program failed.

Type hints

Annotations in Python that specify the expected types of variables, function parameters, or return values (e.g., `def f(x: int) → str:`).

Vectorisation

Rewriting loops into array operations (e.g., with NumPy/pandas) to improve performance and readability.

Virtual environment (venv, conda env)

An isolated Python setup to manage dependencies without interfering with global installations.