Introduction to Python with AI

Mike McRae

February 14, 2025

Overview

- GPT and Python
- What is Python?
- Miniconda, Spyder, Environments
- Prompting GPT
- Running Python code in Spyder and the Command Line

What you will/will not learn?

Will learn:

- How to install Python, create environments and execute code.
- How to effectively prompt GPT/Deepseek to solve nearly any problem.
- Basic structure of Python code and Spyder interface.

Will not learn:

- How to code in Python (not directly anyway).

GPT and Python

- GPT (Large Language Model - LLM) is brilliant at predicting output in a language.
- Python is an open source programming language, so there is a lot of training data.
- Is that cheating? It depends what you use it for.
- Can it do everything? Not yet.
  - But it can probably tell you how.
- Should it do everything? Probably not.
  - Ethically, policy relevant results need to be checked.
  - Knowing what your code does is important.

My personal example uses

- Writing lecture slides (in LaTeX).
- Writing code for:
  - Web scraping
  - Solving Captcha/Cloudflare
  - Interacting with APIs (incl. GPT itself)
  - Topic Analysis (LDA, BERT, novel approaches)
  - Machine learning approaches (RF, CF, neural networks, fine-tuning transformers)
- Idea & hypothesis generation QJE paper here
- Using Docker containers.

What is Python?

- Python is a high-level, interpreted programming language.
- Used for data science, AI, web development, automation, etc.
- Readable, easy to learn, and has a large community.

Why Use an LLM for Coding?

- **Low Investment:** No need for deep prior coding knowledge to start solving problems.
- **Error Reduction:** Can assist in debugging and improving code quality.
- **Learning Tool:** Provides explanations and alternative solutions to improve understanding.
- **Scalability:** Helps tackle both simple and complex programming tasks efficiently.

Prompting

The Iterative Process with Chat-GPT

- Writing effective prompts is key to getting useful code.
- Start with a clear and detailed prompt specifying the task.
- Paste in data, photos, correspondence, anything which will give context and detail.
- If the output isn't what you expected, refine the prompt or ask for corrections.
- If errors occur, copy the error message and provide it back to Chat-GPT.
- Iterate: keep refining until you get working, understandable code.

Example: Refining a Chat-GPT Prompt

Unclear Prompt:

- `Write Python code to generate a bar chart from a CSV file.`

Refined Prompt:

- `Here are the first 3 rows of my csv stored at path/to/file.csv. Write me a Python script to generate a bar chart the value column by category column and save the output graph in path/to/output.png`

Using Chat-GPT for Step-by-Step Guidance

- GPT can be your coder and/or your teacher.
- Chat-GPT can break down problems into smaller steps.
- Instead of asking for a full solution, ask:
  - `How do I load a CSV file in Pandas?`
  - `How do I filter rows where 'Age' > 30?`
  - `How do I create a scatter plot from two columns?`
- This helps in understanding and learning each part of the code.

CLI and IDE

What is the Command Line vs. IDE?

- **Terminal / Command Prompt**: A command-line interface (CLI) for (among other things) running Python scripts directly.
- **IDE**: An Integrated Development Environment (IDE) has a built-in editor and interactive console.
- Terminal is useful for quick script execution, while IDE provides a more user-friendly interface.
- Windows: Press `Win + R`, type `cmd`, and press Enter.
- Mac: Open Spotlight (`Cmd + Space`), type `Terminal`, and press Enter.

Why Miniconda and Spyder?

We will be using Miniconda to manage Python and Spyder to write code.

- **Miniconda:** A lightweight package manager for Python.
- **Spyder:** A beginner-friendly Integrated Development Environment (IDE) for Python.
- Allows for easy package management and an interactive coding environment.

Task 1 - Installing and running Miniconda and Spyder

- Open GPT.
- Prompt:

```
I am new to using Python.  I am told I should
install Miniconda and Spyder and start a new
script in Spyder.  Explain to me what to do and
why I am doing it.  I am using Windows/Mac/Linux.
Assume I have no experience.
```

Task 1 - Installing and running Miniconda and Spyder

- Now follow the tasks
- If things do not work out as you expected, then explain what is happening to GPT and it will guide you.
- This can be as simple as, "I don't know where the file downloaded, explain what to do next."
- The following slides guide you through the process also.

Installing Miniconda (Windows)

1. Go here and put in your email.
2. On the next page scroll down to the Miniconda installers (not Anaconda).
3. Select the appropriate installer based on your system:
   - **Windows 64-bit (Recommended)**: `Miniconda3 Windows 64-bit .exe`
   - If unsure, check by right-clicking "This PC" → "Properties" → Look for "System type".
4. Run the downloaded installer and follow the on-screen instructions.
5. **Important:** Select **"Add Miniconda to PATH"** when prompted.
6. Click **Next** and complete the installation.
7. Restart your computer to apply changes.

Verifying Miniconda Installation

After installation, verify that Miniconda is correctly installed:

1. Open the Command Prompt:
   - Press **Win + R**, type `cmd`, and press **Enter**.
   - Alternatively, search for **Command Prompt** in the Start menu.
2. Type the following command and press **Enter**:
   - `conda --version`
3. If installed correctly, it should display something like:
   - `conda 23.1.0` (version number may vary)
4. If you get an error: `"Command not found"` or similar, Conda is not in your system's PATH.

Fixing PATH Issues: Finding Miniconda

If `conda --version` does not work, manually add Miniconda to your system's PATH.

1. **Find your Miniconda installation folder:**
   - Open **File Explorer**.
   - Navigate to the default location:
     - `C:\Users\YourUsername\Miniconda3`
   - If installed elsewhere, search for `conda.exe` inside the Miniconda folder.
2. **Confirm the folder contains:**
   - `conda.exe` (in the main Miniconda directory).
   - `Scripts` folder (contains utilities like `conda` and `python`).
3. Once located, you will add this path manually to system environment variables.

Fixing PATH Issues: Adding Miniconda to System Variables

To add Miniconda to your system PATH manually:

1. **Open Environment Variables:**
   - Click **Start** and search for **Environment Variables**.
   - Select **"Edit the system environment variables"**.
   - Under the **Advanced** tab, click **Environment Variables**.
2. **Modify the System PATH:**
   - In the "System variables" section, find and select **Path**, then click **Edit**.
   - Click **New** and add the following paths:
     - `C:\Users\YourUsername\Miniconda3`
     - `C:\Users\YourUsername\Miniconda3\Scripts`
   - Click **OK** and close all windows.
   - Restart your computer to apply changes.
3. **Verify the Fix:**
   - Open a new Command Prompt and type: `conda --version`
     - Press **Win + R**, type `cmd`, and press **Enter**.
     - Alternatively, search for **Command Prompt** in the Start menu.
   - If set correctly, it should display a version number.

Installing Miniconda (Mac)

1. Go here and put in your email.
2. On the next page scroll down to the Miniconda installers (not Anaconda).
3. Select the appropriate installer based on your system:
   - **Mac 64-bit (Apple Silicon) Graphical Installer**
   - If unsure, check by clicking the apple in top left corner and **About This Mac**.
4. Run the downloaded installer and follow the on-screen instructions.
5. Open a terminal: Open Spotlight (`Cmd + Space`), type `Terminal`, and press Enter.
6. Type: `conda --version` to check installation.
7. If anything other than Conda version - relay back to GPT.

Setting Up Spyder

- Install Spyder:
  - Open Command Line:
    - Windows: Press `Win + R`, type `cmd`, and press Enter.
    - Mac: Open Spotlight (`Cmd + Space`), type `Terminal`, and press Enter.
  - Type: `conda install spyder`
  - Run Spyder: Type `spyder`
- Note: This Command Line terminal is now just for running Spyder. Do not use it for anything else. If you need another command line, then open a new one.

Python & Spyder

What is a Python Environment?

- **Definition:** A Python environment is an isolated workspace where specific versions of Python and its packages are installed.
- **Workflow for Using a Python Environment:**
  1. **Create an environment:** `conda create --name myenv python=3.10`
  2. **Activate the environment:** `conda activate myenv`
  3. **Install a package:** `pip install pandas` or `conda install pandas`
  4. **Import a package in Python:** `import pandas as pd`
- **Why Use Environments?**
  - Avoid conflicts between different projects.
  - Keep dependencies organized.
  - Ensure reproducibility of code.

Setting Up a Python Environment

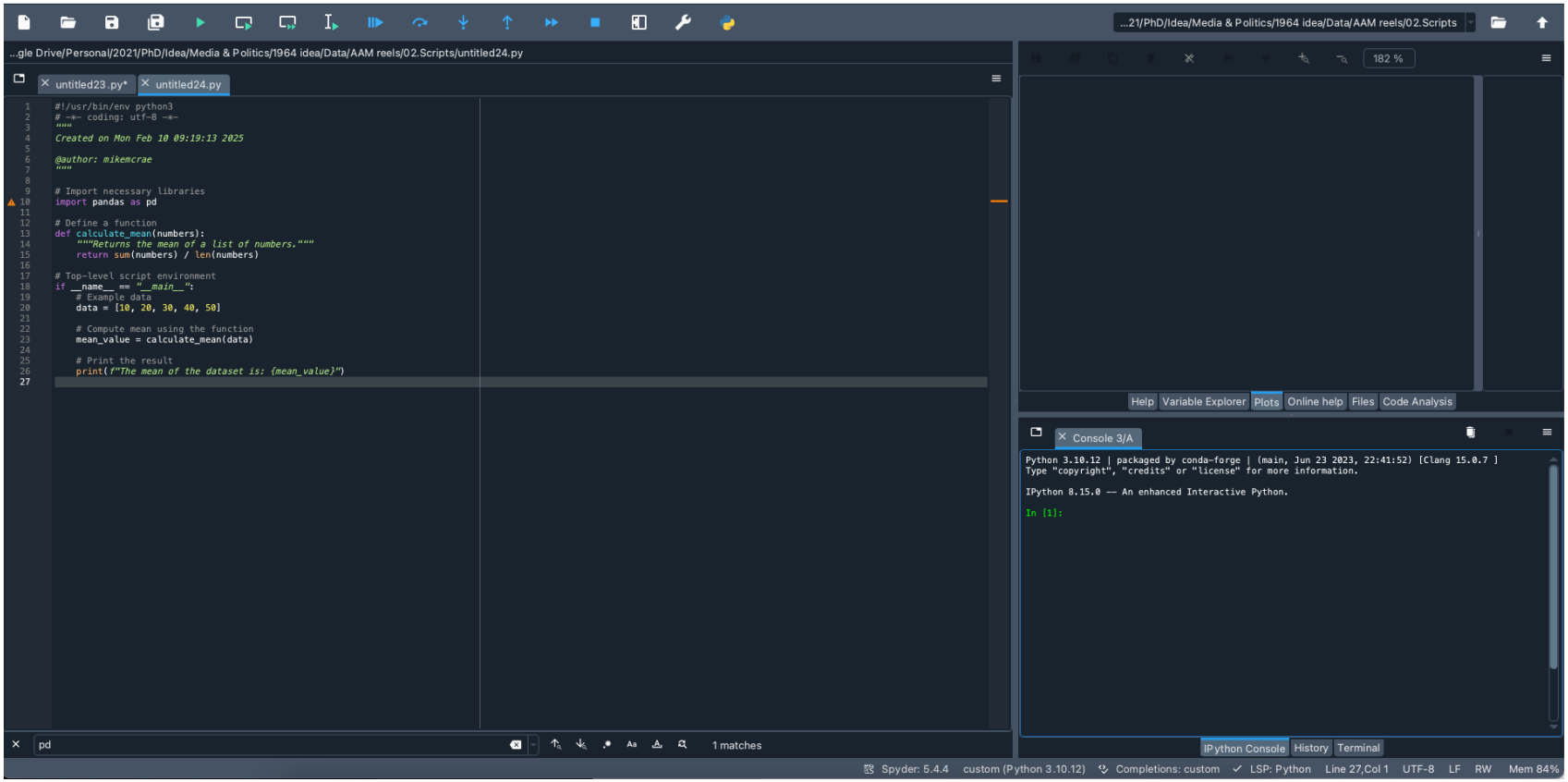
1. Create a folder called `python_class` on your desktop or documents.
2. Open a terminal (Mac) or Command Prompt (Windows):
   - Windows: Press `Win + R`, type `cmd`, and press Enter.
   - Mac: Open Spotlight (`Cmd + Space`), type `Terminal`, and press Enter.
3. Navigate to your folder: `cd path/to/python_class`
4. Create a new Conda environment: `conda create --name myenv python=3.10`
5. Activate the environment:
   - Windows: `conda activate myenv`
   - Mac: `conda activate myenv`
6. Install new packages: `pip install seaborn`

Understanding the Spyder Interface

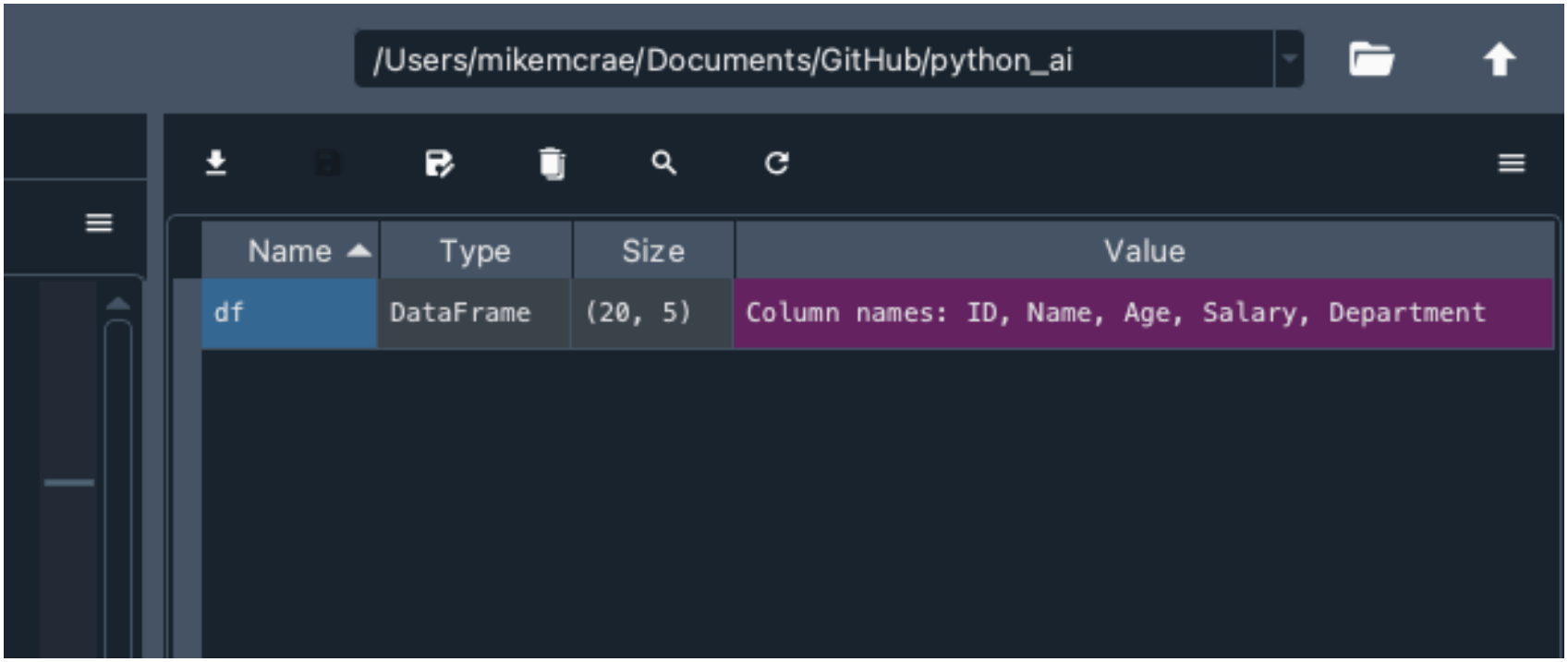
Spyder consists of three main sections:

- **Editor (Script Editor):**
  - Where you write and edit Python code.
  - Supports multiple script tabs for working on different files.
- **Console (CLI - Command Line Interface):**
  - Runs Python code interactively.
  - Displays output, errors, and debug messages.
  - Links directly to the environment (So, you can install packages here).
- **Variable Explorer:**
  - Shows stored variables and their values.
  - Useful for inspecting data structures and debugging.

Spyder Interface



Set working directory



Basic Parts of a Python Script

- **Importing Dependencies:**
  - Code reuse and functionality extension.
  - Example: `import pandas as pd`
- **Defining Functions:**
  - Encapsulates reusable logic.
  - Example: `def calculate_mean(numbers):`
- **Top-Level Script Environment:**
  - Ensures that code runs only when the script is executed directly.
  - Uses: `if __name__ == "__main__":`
  - Allows importing the script as a module without unintended execution.

Python Script

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 10 09:19:13 2025

@author: mikemcrae
"""

# Import necessary libraries
import pandas as pd

# Define a function
def calculate_mean(numbers):
    """Returns the mean of a list of numbers."""
    return sum(numbers) / len(numbers)

# Top-level script environment
if __name__ == "__main__":
    # Example data
    data = [10, 20, 30, 40, 50]

    # Compute mean using the function
    mean_value = calculate_mean(data)

    # Print the result
    print(f"The mean of the dataset is: {mean_value}")
```

Key Concepts in Python

- **Variables and Data Types:**
  - Variables store data: `x = 10`, `name = "Alice"`
  - Common data types: `int`, `float`, `str`, `bool`, `list`, `dict`, `tuple`, `set`
- **Data Structures:**
  - **Lists:** Ordered, mutable collection: `[1, 2, 3]`
  - **Dictionaries:** Key-value pairs: `{"name": "Alice", "age": 25}`
  - **Tuples:** Immutable sequences: `(1, 2, 3)`
  - **Sets:** Unordered, unique elements: `{1, 2, 3}`
- **Control Flow (Loops and Conditionals):**
  - `if-elif-else` for decision-making.
  - `for` and `while` loops for iteration.
- **List Comprehensions:**
  - Concise way to create lists: `squares = [x**2 for x in range(10)]`
- **Error Handling:**
  - Use `try-except` to catch and handle errors.
  - Example: `try: x = int("abc") except ValueError: print("Invalid input")`

Basic Concepts

```python
# --- Variables and Data Types ---
x = 10  # Integer
name = "Alice"  # String
pi = 3.14  # Float
is_python_fun = True  # Boolean

print(f"Integer: {x}, String: {name}, Float: {pi}, Boolean: {is_python_fun}")

# --- Data Structures ---
# List (Ordered, Mutable)
my_list = [1, 2, 3]
my_list.append(4)
print(f"List: {my_list}")

# Dictionary (Key-Value Pairs)
my_dict = {"name": "Alice", "age": 25}
print(f"Dictionary: {my_dict}")

# Tuple (Immutable Sequence)
my_tuple = (1, 2, 3)
print(f"Tuple: {my_tuple}")

# Set (Unordered, Unique Elements)
my_set = {1, 2, 3, 1, 2}  # Duplicates are removed
print(f"Set: {my_set}")

# --- Control Flow ---
# If-elif-else
num = 10
if num > 10:
    print("Greater than 10")
elif num == 10:
    print("Exactly 10")
else:
    print("Less than 10")

# Loops
print("For loop output:")
for i in range(3):
    print(i)

print("While loop output:")
count = 0
while count < 3:
    print(count)
    count += 1

# --- List Comprehensions ---
squares = [x**2 for x in range(10)]
print(f"Squares using list comprehension: {squares}")

# --- Error Handling ---
try:
```

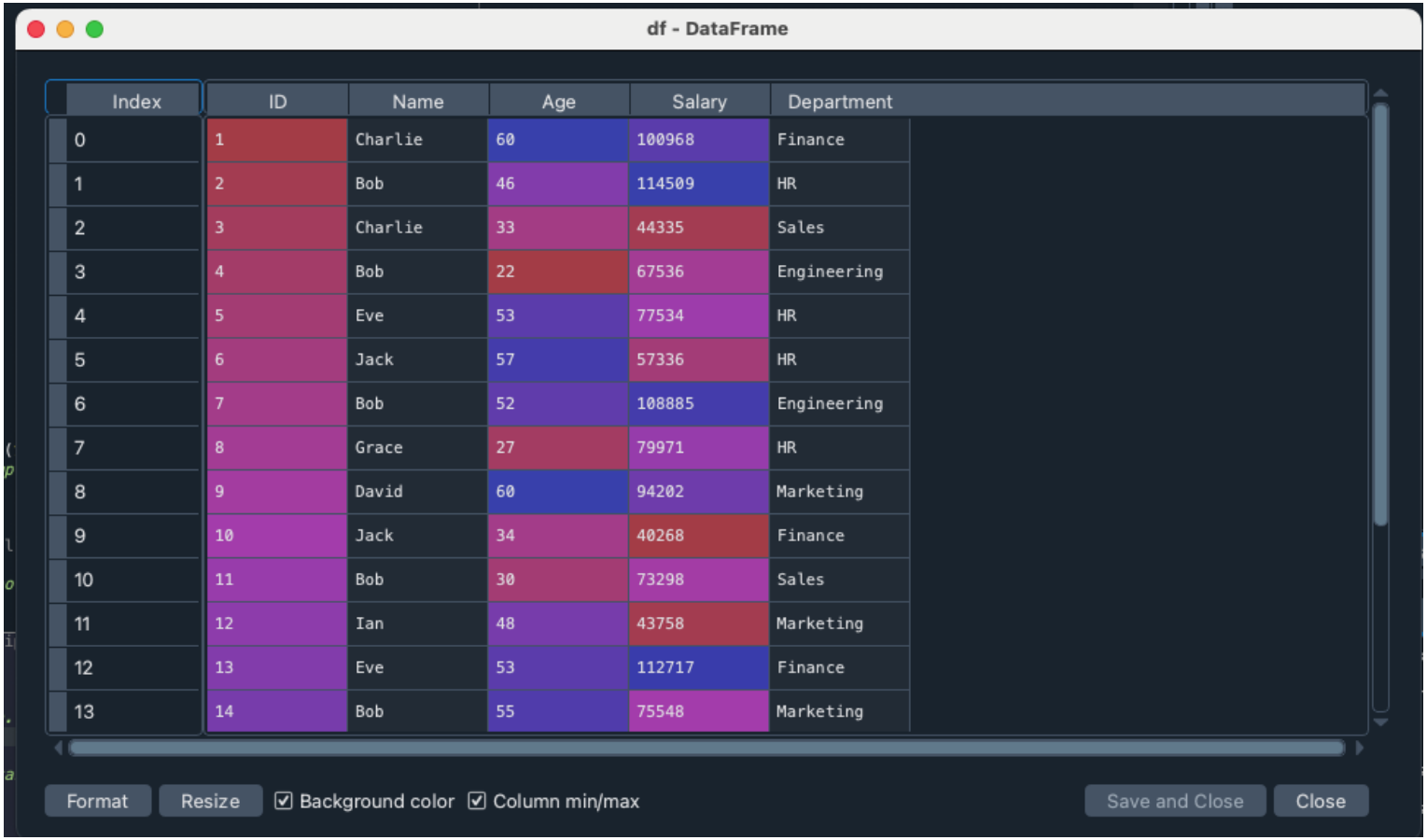
Working with DataFrames in Python

- **What is a DataFrame?**
  - A 2D table-like structure used for data manipulation.
  - Provided by the `pandas` library.
  - Similar to Excel spreadsheets but optimized for Python.
- **Loading Data:**
  - Read from a CSV file: `df = pd.read_csv("data.csv")`
  - Read from an Excel file: `df = pd.read_excel("data.xlsx")`
- **Basic Operations:**
  - View first 5 rows: `df.head()`
  - Get column names: `df.columns`
  - Filter rows: `df[df["Age"] > 25]`
  - Select a column: `df["Name"]`
- **Modifying DataFrames:**
  - Add a new column: `df["Salary"] = [50000, 60000]`
  - Drop a column: `df.drop("Age", axis=1, inplace=True)`
  - Sort values: `df.sort_values("Age")`

Dataframes

```python
#%% Importing dataframe and mnipulating
import pandas as pd

# Load the CSV file
df = pd.read_csv("student_data.csv")

# Display first few rows
print("First 5 rows of the dataset:")
print(df.head())

# Basic statistics
print("\nSummary Statistics:")
print(df.describe())

# Group by Department and calculate the average salary
avg_salary = df.groupby("Department")["Salary"].mean()
print("\nAverage Salary by Department:")
print(avg_salary)

# Count number of students per department
dept_counts = df["Department"].value_counts()
print("\nNumber of Students per Department:")
print(dept_counts)

# Filter students older than 40
older_students = df[df["Age"] > 40]
print("\nStudents older than 40:")
print(older_students)
```

Variable viewer



Index	ID	Name	Age	Salary	Department
0	1	Charlie	60	100968	Finance
1	2	Bob	46	114509	HR
2	3	Charlie	33	44335	Sales
3	4	Bob	22	67536	Engineering
4	5	Eve	53	77534	HR
5	6	Jack	57	57336	HR
6	7	Bob	52	108885	Engineering
7	8	Grace	27	79971	HR
8	9	David	60	94202	Marketing
9	10	Jack	34	40268	Finance
10	11	Bob	30	73298	Sales
11	12	Ian	48	43758	Marketing
12	13	Eve	53	112717	Finance
13	14	Bob	55	75548	Marketing

Task 2

Task 2: Data Manipulation and Visualization

Task: Generate a dataset, manipulate it, and visualize the results.

1. Prompt Chat-GPT to generate a dataset and save it.
2. Load the dataset into Python.
3. Perform basic data manipulation (e.g., filtering, grouping).
4. Create a visualization using Matplotlib or Seaborn.

Task 2: Prompting Chat-GPT to generate data

- Open Chat-GPT and enter the prompt:

  ```
  Generate a CSV file with 100 rows containing
  columns:  ID, Name, Age, Salary, Department,
  which can be downloaded for use.
  ```

- Save it in the `python_class` folder as `data.csv`.

Task 2: Prompting Chat-GPT to generate Python script

- Now prompt:

```
Write me a python script which creates a bar
chart of the salary by department using
/path/to/python_class/data.csv.  Save the graph
at /path/to/python_class/output.png
```

Task 2: Executing the script in Spyder

- Open Spyder and create a new Python script.
- Copy the generated code into a new Python script.
- Run (Shift+Enter).

Task 2: Executing the script in Terminal/Command ilne

- Save the Python script in /path/to/python_class/script1.py
- Open a new terminal/prompt
- Type:
  - `conda activate myenv`
  - `cd /path/to/python_class`
  - `python3 script1.py`

Task 3: Convert HEIC images into pdf and collate into one

1. **Download the Task 3 Folder**
2. **Ask GPT to Write the Code** Prompt GPT:

   ```
   Write Python code to convert these 4 HEIC images
   to a single ordered PDF. The files are: {insert
   file location}. Save pdf at {insert where you
   wan the output}
   ```

3. **Do as instructed, iterate, fix, etc**
4. **Run the Final Script and Verify Output**

HW

- Complete some tasks using GPT and Python
- Send me some tasks

Next week

- Interacting with the ChatGPT API
- Web Scraping
- Your suggested tasks

Q&A

Questions?