Okay, so first of all, I have created this to document, what I have done, so I don’t forget what and how I have done.

When I opened the get files link for github files, it took me to the github space, where I have copied the code URL to create a clone of it, so the clone folder is named - complete-guide-to-python-data-analysis-4571000.

After this I have chosen to create the codespace for this course, there was an option to have a codespace, so I chose yes. I have a codespace now, like a virtual computer with all the set up and pre-installed environment to work on the projects.

 It's a **ready-to-go computer** in the cloud where you can write and test your code.

 ️ It has all your **coding tools and settings pre-installed**, like a superhero's utility belt.

 You can access it **from any computer**, and it looks just like your regular coding app (like VS Code).

 ⏱️ It saves time because you don’t have to install stuff on your computer. You just **click a button, and boom — your code space is ready!**

Then I have created a fork of the main project from their github (from github profile). So I have this new project created, and then I chose to open it with Visual studio code, it asked for my login credentials and also I had to give 2 permissions from my personal GitHub account. I also clicked yes to download the extension to work with github codespace.

Comprehensions:

List: ­{<element> for <variable> in <iterable> [if clause]}

Dict: {key:value for <variable> in <iterable> [if clause]}

Set: {<element> for <variable> in <iterable> [if clause]}

Tuple: tuple (<element> for <variable> in <iterable> [if clause])

Omit wrappers to get a generator expression

Reading Unicode:

Common Unicode encodings for open (filename, encoding=’…’):

‘UTF-8’ (default): variable length (one, two, or four bytes), generalized ASCII

‘UTF-16’ : variable length (two or four bytes), non-Latin scripts, native in Java, JavaScript, Windows

‘ISO-8859-1’ (or latin1): one byte, western European Languages, compatible with first 256 Unicode characters

‘CP1252’:one byte, western European Languages, common in Windows

‘ascii’: not Unicode; seven-bit only, useful for legacy applications.

Numpy:

* The fundamental package for numerical computing with Python,
* Fast, memory efficient N- dimensional arrays
* Excellent choice for large, homogeneous datasets
* A foundation for many mathematical packages and to integrate Python with C/Fortran

NumPy is a fundamental part of the Python ecosystem, and it provides the foundation for many data analysis and numerical libraries in applications, including SciPy for mathematics, Matplotlib for plotting, pandas and statsmodels for statistics, scikit-learn for machine learning, and scikit-image for image processing. NumPy is also crucial in interfacing with compiled code in C, C++, or Fortran. In addition, if you learn NumPy, you will be able to use deep learning frameworks such as PyTorch and JAX, which share the same array interface, as well as specialized array libraries that are interoperable with NumPy. For instance, CuPy to work with arrays on fast GPUs, Dask to spread arrays across computers, Xarray for arrays with labels, and by PyData/Sparse for sparse arrays with many zeros and efficient memory layout. So let's talk about how NumPy arrays are different from Python containers. Python variables are often described as labels. They are not little copies in computer memory ready to receive a value. Rather, the values are independent objects with their own space and memory, and Python variables are just names associated with the values. So you can have more than one variable referring to the same object. This mechanism is very flexible, and it makes it possible to have lists and dictionaries with heterogeneous elements. In fact, you can think of a list as a numbered sequence of labels. However, this scheme is not very efficient when we need to deal with many values of the same type. In that case, you want to reserve space in memory and store all the values side by side, and that's exactly what a NumPy array is. Organizing data in this way is both faster and more memory efficient, and it's also necessary to interface Python with other languages such as C or Fortran, which count on data being laid out in memory in this fashion.