Lesson 1 – NumPy Refresher
Topic: Arrays, creation, indexing, slicing, reshaping, operations

---

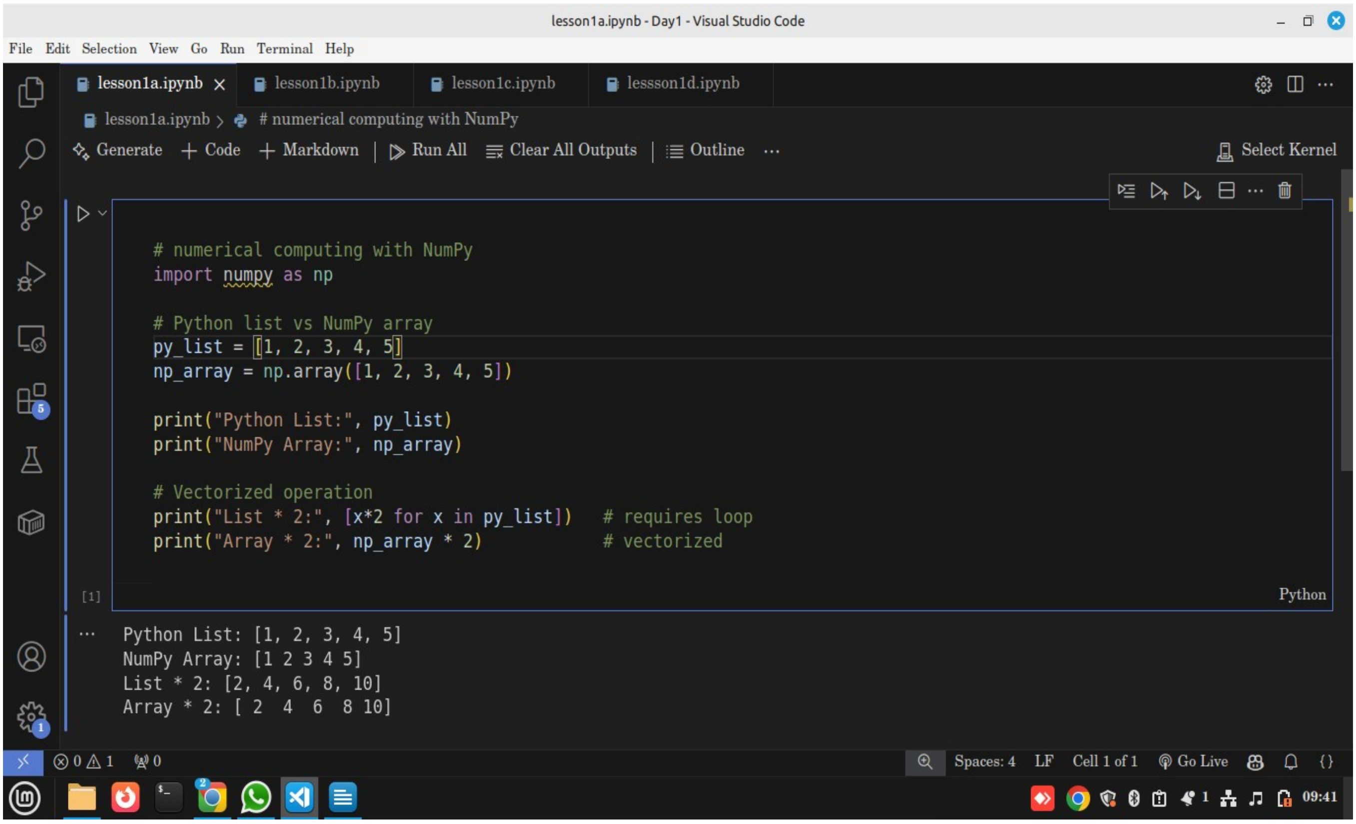
🔹 NumPy & Arrays

Theory Notes

- NumPy = *Numerical Python*, a library for numerical computing.
- Main object: **ndarray** (n-dimensional array).
- Advantages: **fast**, **efficient**, **vectorized operations**.

Python lists vs NumPy Array

Practical



Explanation

- We imported NumPy as `np` (common alias).
- `py_list` is a normal Python list.
- `np_array` is a NumPy array created from the same numbers.
- When multiplying a Python list by 2 → it duplicates the list `[1,2,3,4,5,1,2,3,4,5]`.
- When multiplying a NumPy array by 2 → it multiplies **each element** (`[2,4,6,8,10]`).
- This is the **power of vectorization**.

Note:

- A Python list requires a loop to multiply each element.
- A NumPy array applies the operation automatically to all elements.
- This makes NumPy much faster when working with large datasets.

Use in Data Science

NumPy arrays are the foundation of **pandas DataFrames, scikit-learn, and deep learning libraries**. Almost every dataset you load is converted internally into arrays for speed.

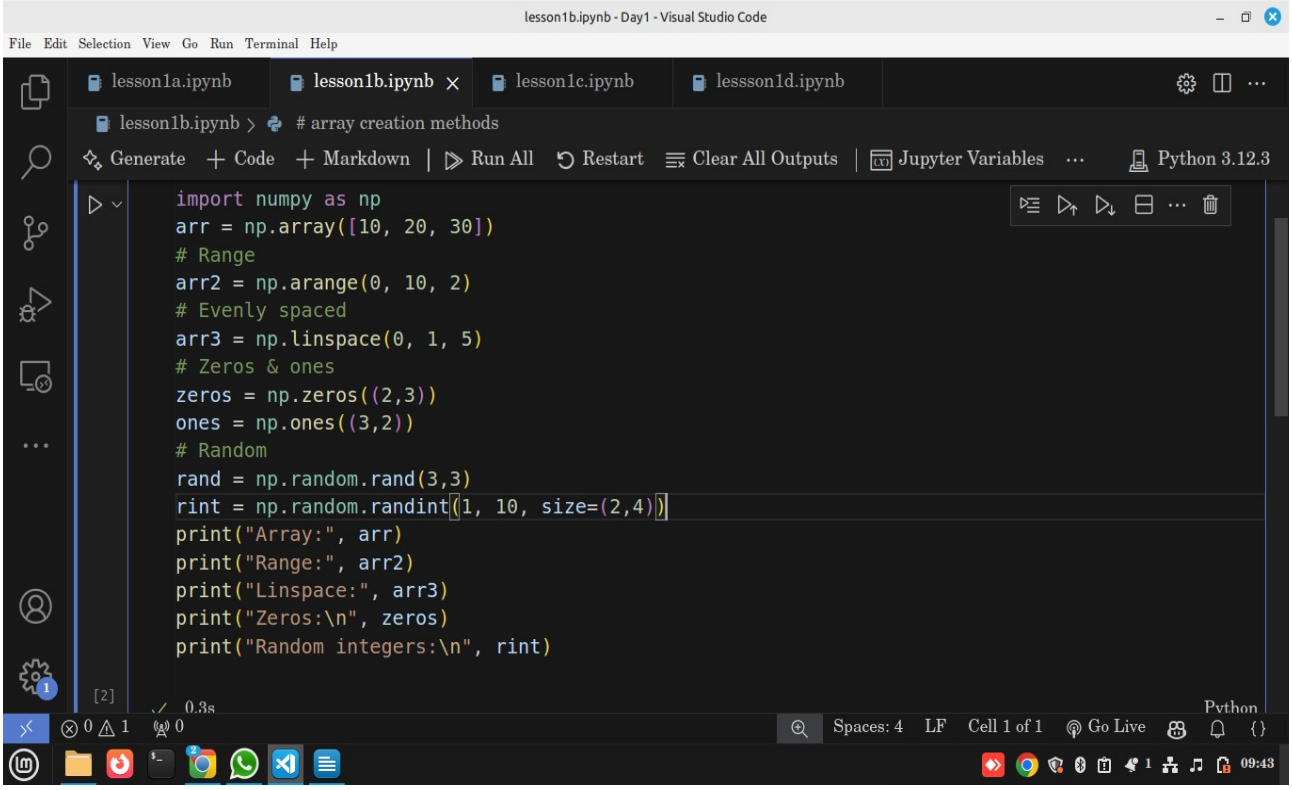
🔹 Array Creation Methods

Theory Notes

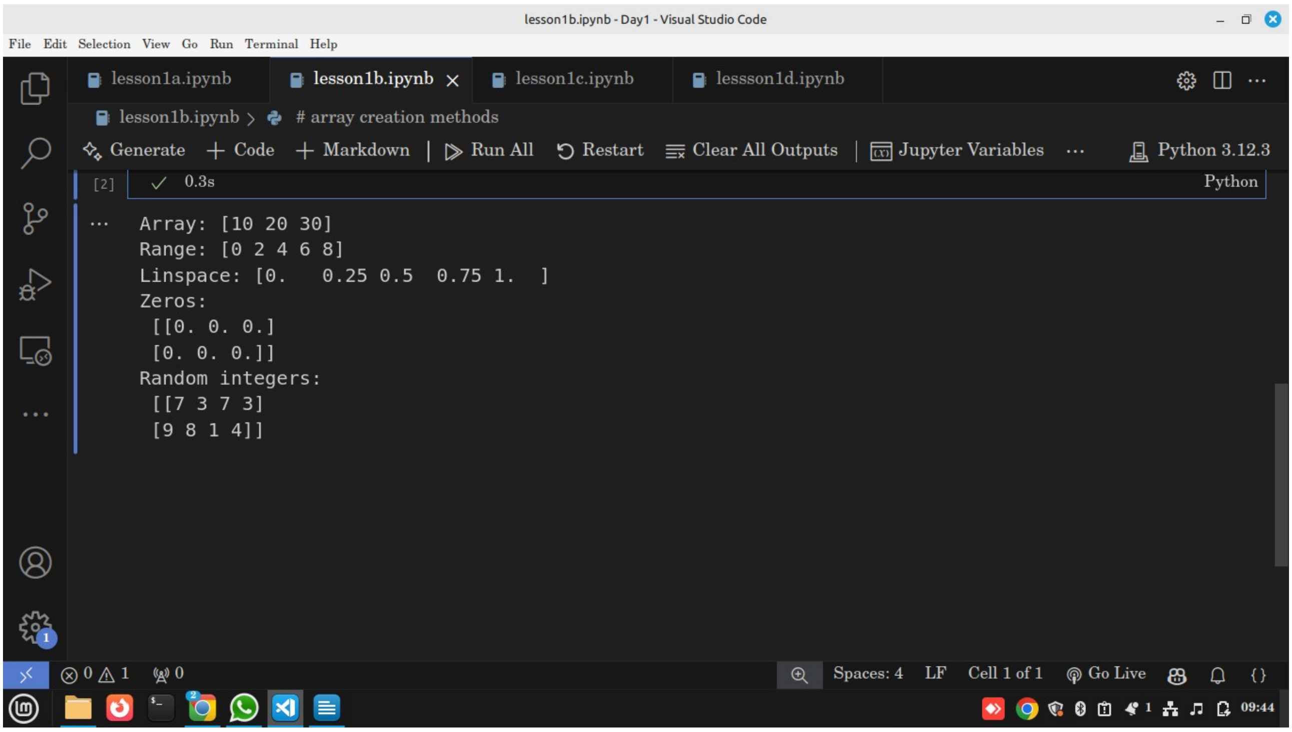
Ways to create arrays in NumPy:

- From Python lists → `np.array()`
- Number ranges → `np.arange()`
- Even spacing → `np.linspace()`
- Special arrays → `np.zeros()`, `np.ones()`, `np.eye()`
- Random arrays → `np.random.rand()`, `np.random.randint()`

Practical



Output



Explanation

- `np.array([10,20,30])` → creates an array from a list.
- `np.arange(0,10,2)` → creates `[0,2,4,6,8]`.
- `np.linspace(0,1,5)` → creates 5 numbers evenly spaced between 0 and 1.
- `np.zeros((2,3))` → 2 rows, 3 columns, all zeros.
- `np.ones((3,2))` → 3 rows, 2 columns, all ones.
- `np.random.rand(3,3)` → 3×3 matrix of random floats between 0–1.
- `np.random.randint(1,10,size=(2,4))` → random integers 1–9 in a 2×4 grid.

Use in Data Science

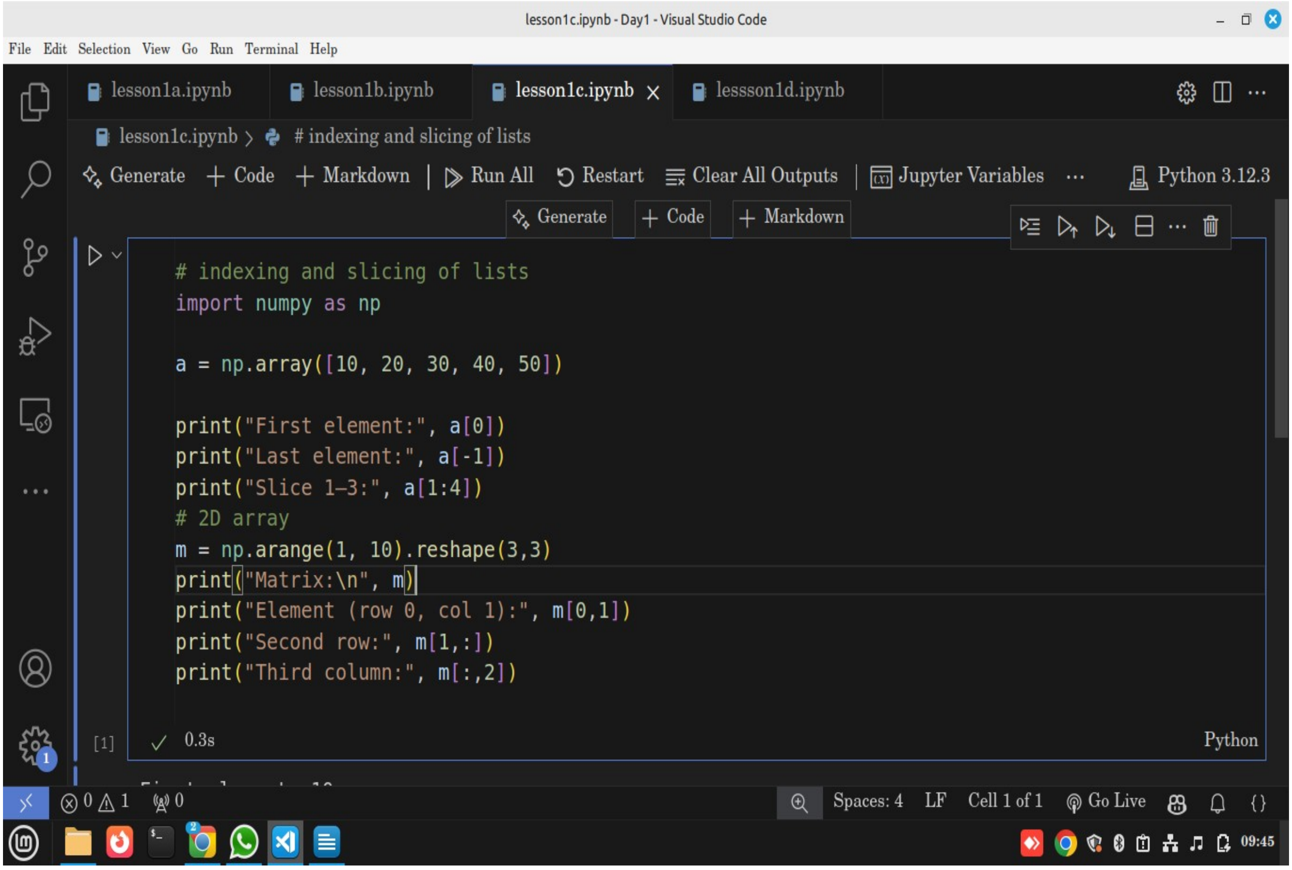
- `np.arange` and `np.linspace` are used to generate sequences of features (like time steps).
- Random arrays are essential in **machine learning** for initializing model weights, or for creating synthetic datasets.

🔹 Indexing & Slicing

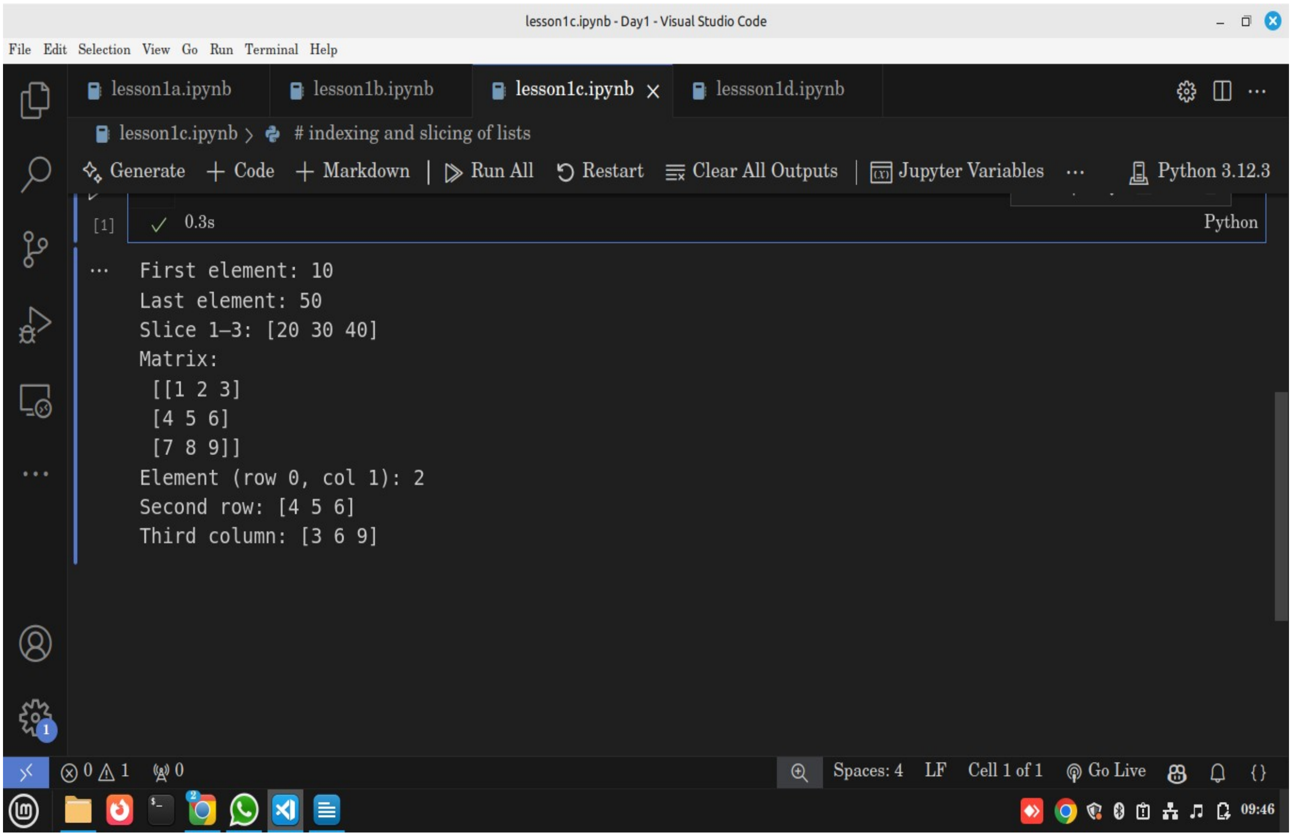
Theory Notes

- Indexing = accessing elements.
- Works in 1D and 2D arrays.
- Uses **row, column** notation for 2D.

Practical



Output



Explanation

- `a[0]` → gives first element `10`.
- `a[-1]` → gives last element `50`.
- `a[1:4]` → extracts values from index 1 to 3 → `[20,30,40]`.
- `m = np.arange(1,10).reshape(3,3)` → makes a 3×3 matrix from numbers 1–9.
- `m[0,1]` → element in **row 0, col 1** (value = 2).
- `m[1,:]` → entire second row `[4,5,6]`.
- `m[:,2]` → entire third column `[3,6,9]`.

Use in Data Science

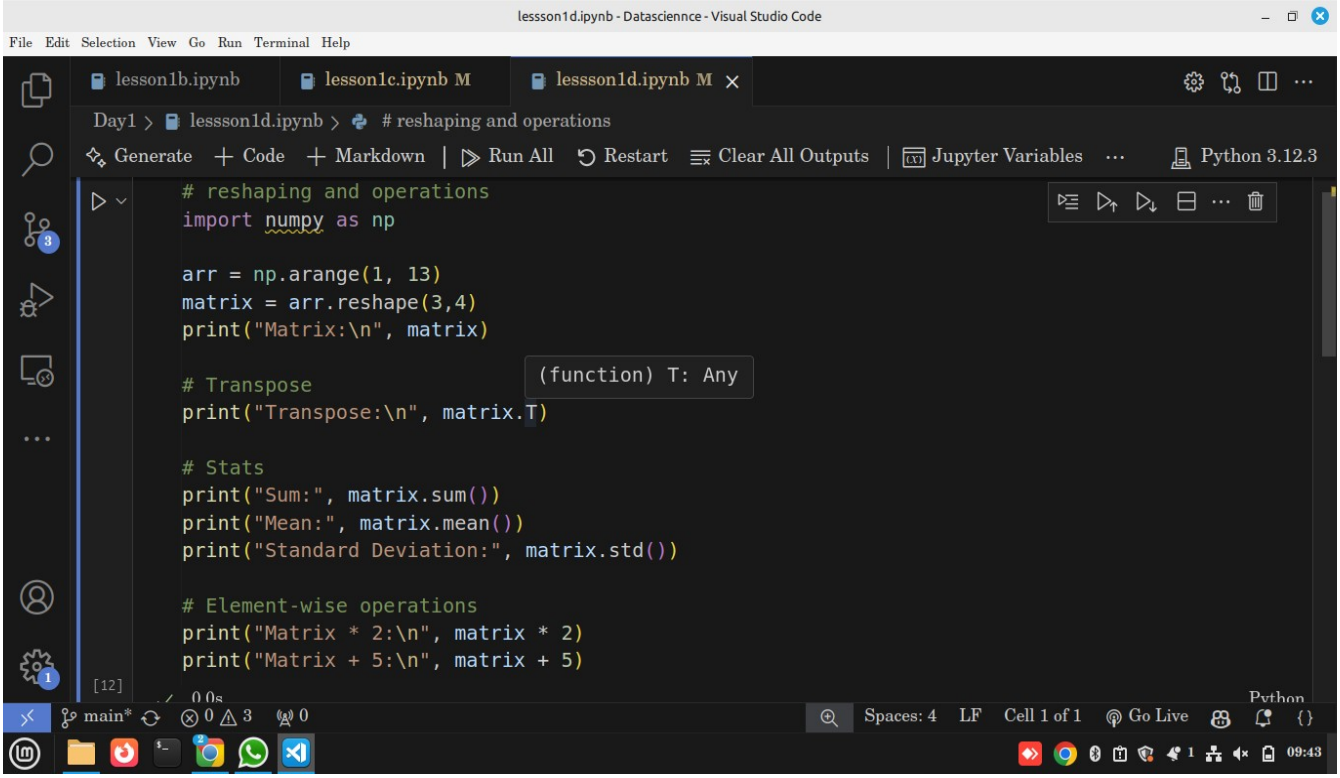
Indexing and slicing are key in **data wrangling**. For example, selecting a column of features (like "age") from a dataset or filtering specific rows (like "patients over 50").

🔹 Reshaping & Operations

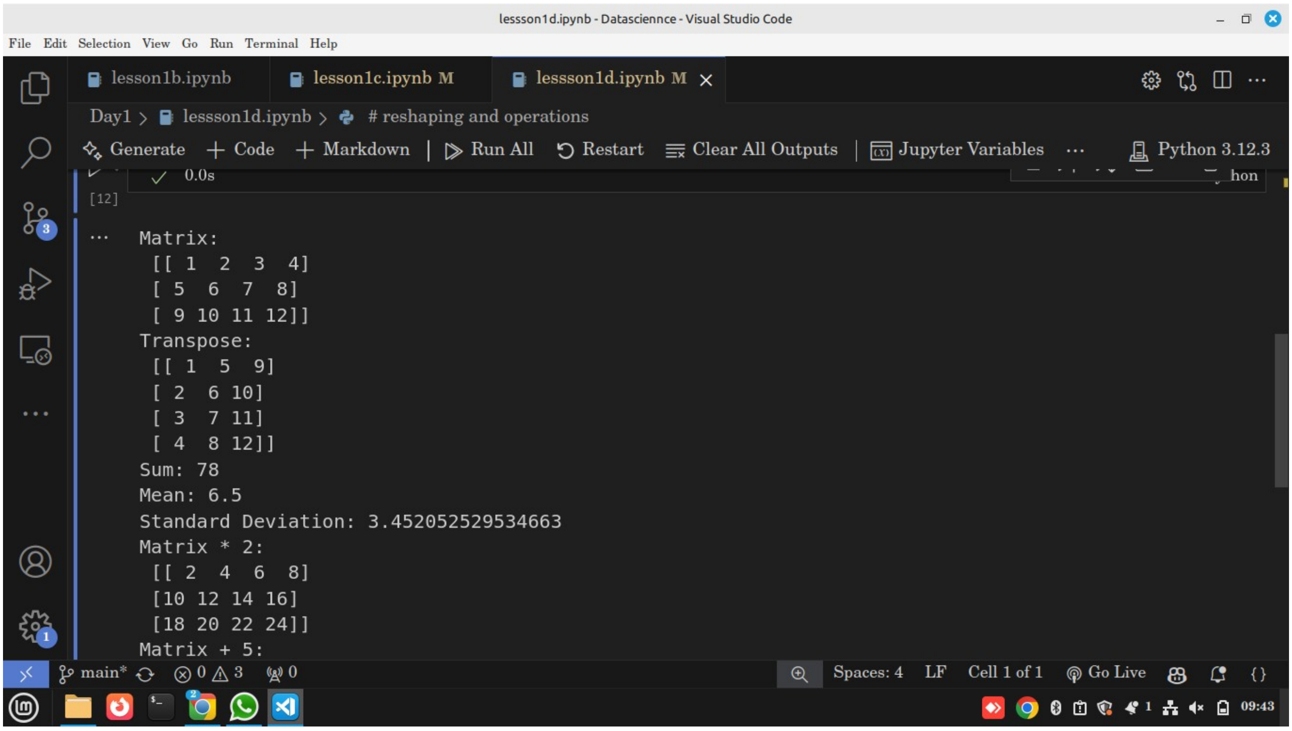
Theory Notes

- `.reshape()` changes array shape without changing data.
- `.T` transposes rows ↔ columns.
- Math/stat functions: `.sum()`, `.mean()`, `.std()`, `.max()`, `.min()`.
- Arithmetic is element-wise.

Practical



Sample Output



Explanation

- `np.arange(1,13)` → creates numbers 1–12.
- `.reshape(3,4)` → reshapes into 3 rows × 4 columns.
- `.T` → flips rows & columns.
- `.sum()` → adds all numbers.
- `.mean()` → computes average.
- `.std()` → measures spread (standard deviation).
- Multiplication (`*2`) and addition (`+5`) apply to every element.

Use in Data Science

Reshaping is crucial when preparing data for machine learning models, where inputs must have a fixed size (e.g., reshape images into arrays). Stats functions help quickly summarize datasets (e.g., average income, standard deviation of ages).

📝 Assignment

🔍 Reflection

Discussion on **why NumPy is faster** and **real-world use cases**.