Pandas Series & DataFrames

🔹 Reminder

Yesterday we learned **NumPy arrays** → efficient numerical data storage and manipulation.

👉 Pandas **builds on NumPy** to handle **tabular/labeled data** (like Excel tables).

🔹 Introduction to Pandas

🐼 How Pandas is Related to NumPy

1. Pandas is Built on Top of NumPy

- Under the hood, Pandas **stores data in NumPy arrays**.
- Every `Series(1D labeled array (like an Excel column))` in Pandas is basically a **wrapper around a NumPy ndarray** with labels.
- A `DataFrame` is just a **collection of Series objects**, which means → lots of NumPy arrays working together.

---

2. Shared Operations

- Both Pandas and NumPy support **vectorized operations** (fast, element-wise math without loops).
- Many Pandas methods actually **call NumPy functions internally**.
  Example: `df.mean()` → uses `numpy.mean()`.

---

3. Differences

Feature	NumPy	Pandas
Main Object	`ndarray` (n-dimensional array)	`Series` (1D) & `DataFrame` (2D)
Data Types	Homogeneous (all elements must be same type)	Heterogeneous (different columns can have different types)
Labels	Only integer indices	Row & column labels (names)
Use Case	Numerical computation	Data analysis / tabular data

Use in Data Science

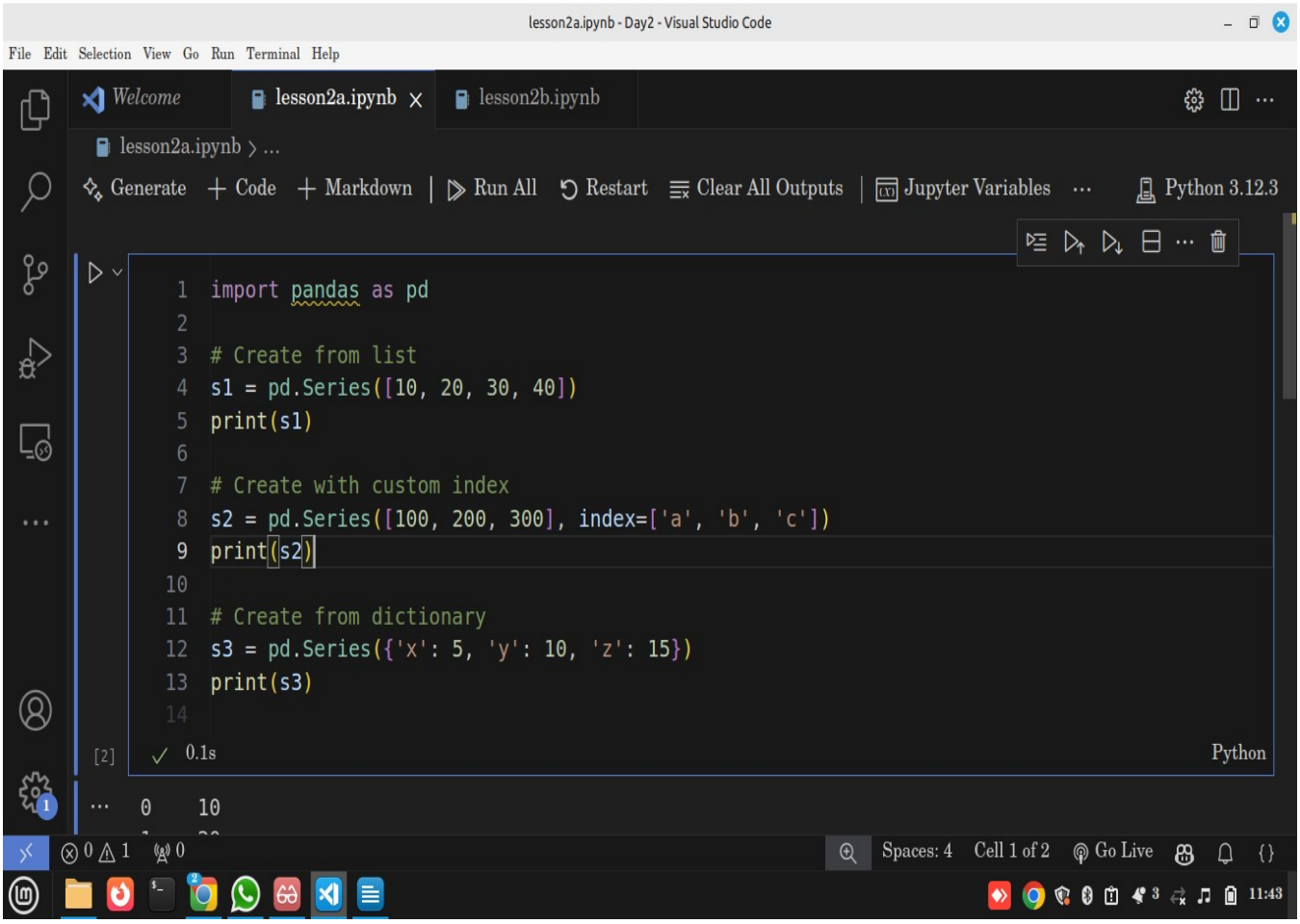
Almost every dataset in data science is loaded into a **DataFrame** before analysis.

🔹 Pandas Series

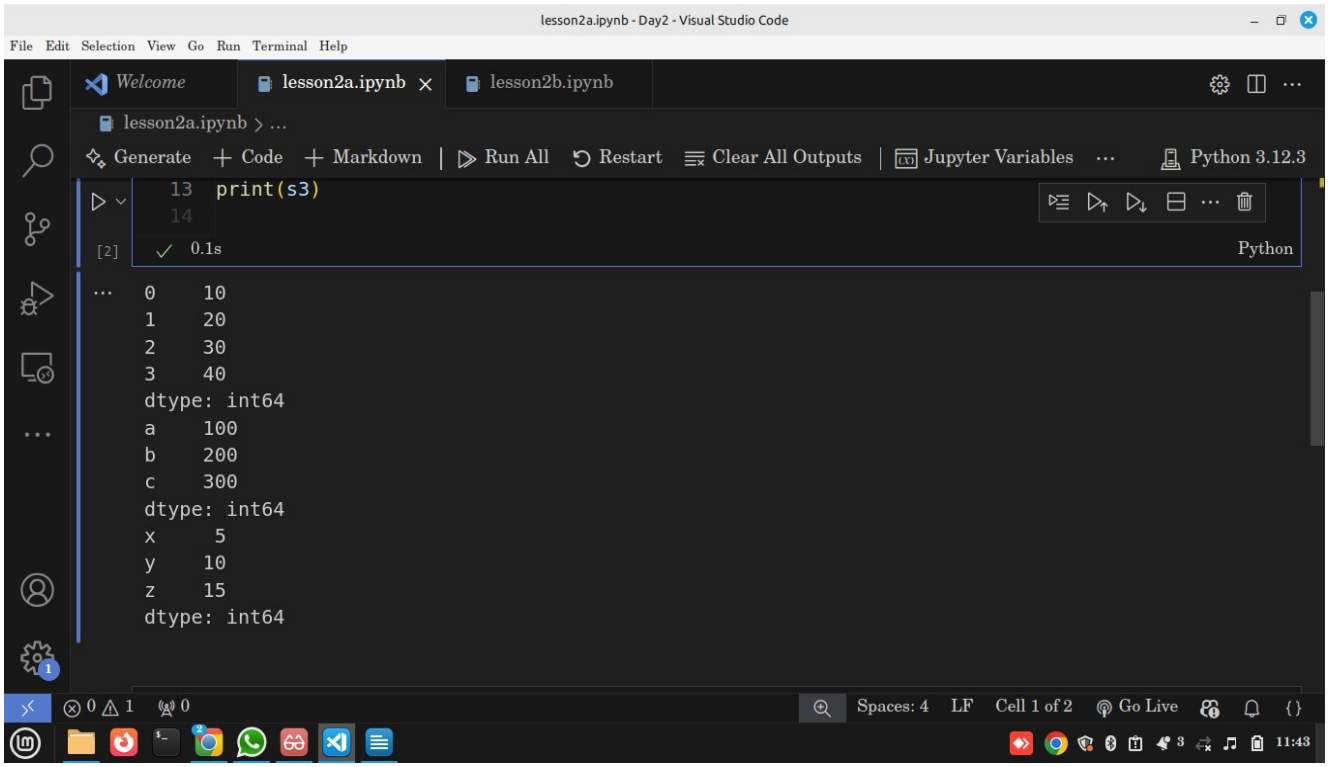
Notes

- Series = **one column** of data, with an **index**.
- Can be created from lists, NumPy arrays, or dictionaries.
- Supports slicing, indexing, and operations like NumPy.

Practical 1 – Creating a Series



Output



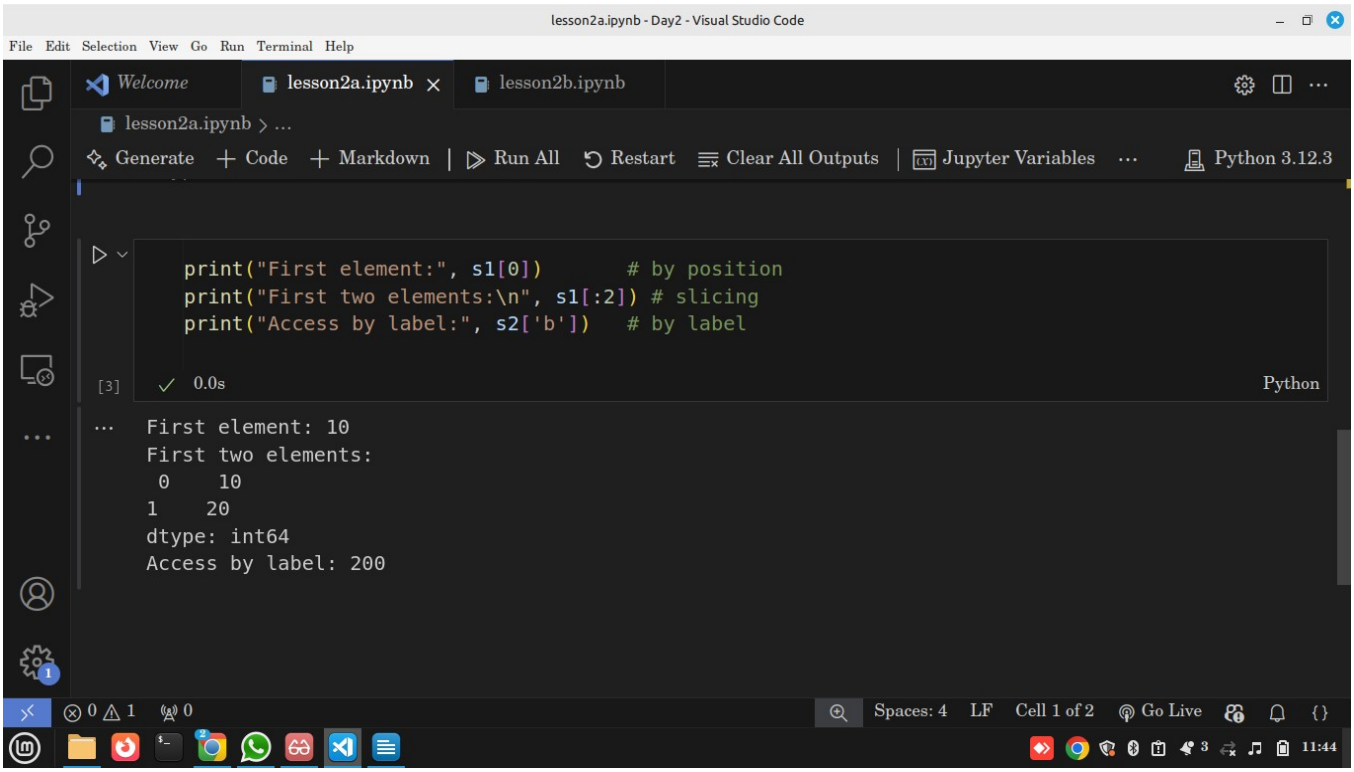
Explanation

- `pd.Series(list)` → creates a Series with default integer index (0,1,2...).
- `index=` lets us name rows (like Excel row labels).
- A dictionary automatically maps **keys → index**.

Use in Data Science

Series often represent **a single variable** (like "Age" column in Titanic dataset).

Practical 2 – Indexing and Slicing



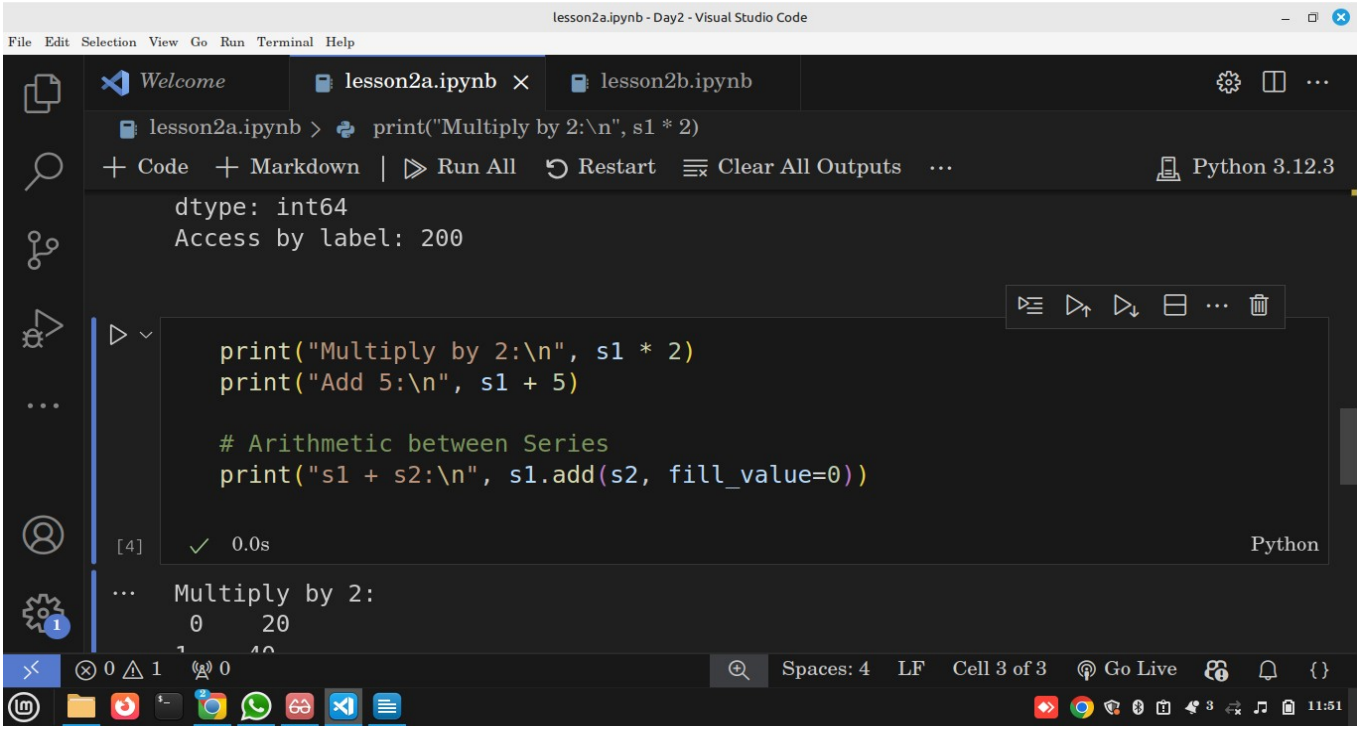
Explanation

- Access by **position** (like NumPy arrays).
- Access by **label** if index is custom.
- Slicing works like Python lists.

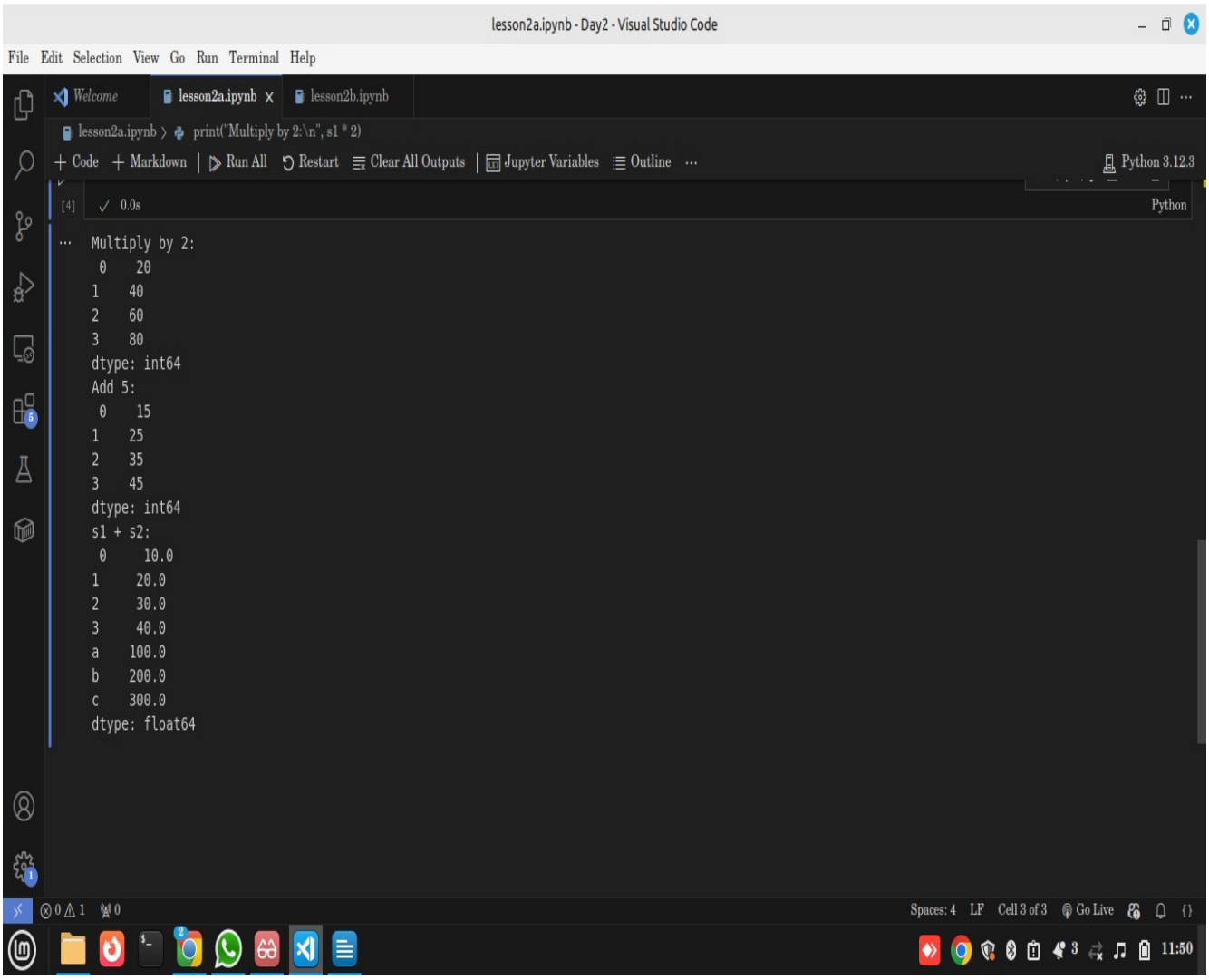
Use in Data Science

Lets us extract **specific rows** or **ranges** for analysis.

Practical 3 – Operations



Output



Explanation

- Operations apply element-wise.
- When adding Series with different indexes, Pandas aligns by index.
- `fill_value=0` fills missing values before arithmetic.

Use in Data Science

This makes data transformation **vectorized & efficient** compared to Python loops.

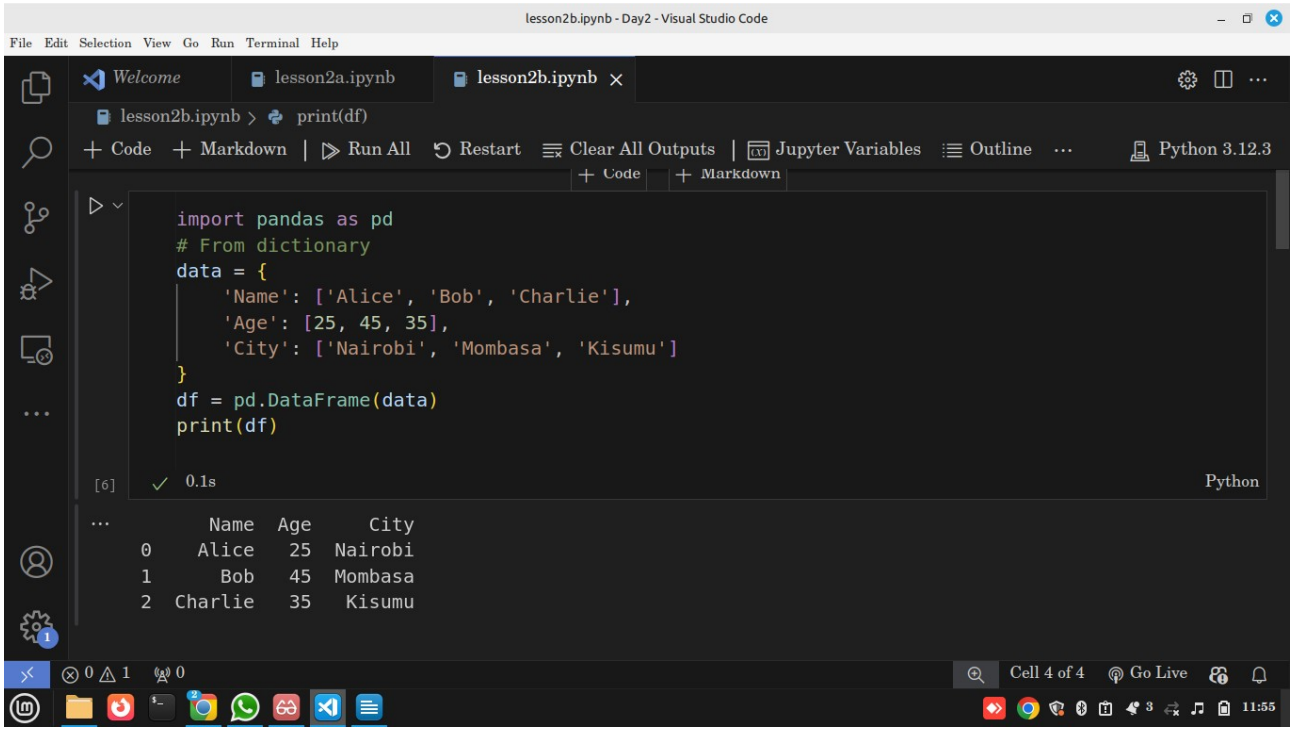
🔹 Pandas DataFrames

Notes

- A DataFrame is **a collection of Series** (columns).
- Think of it as a **table**: rows + columns.

- Can be created from dictionaries, NumPy arrays, or files (CSV, Excel, etc.).

Practical 1 – Creating a DataFrame



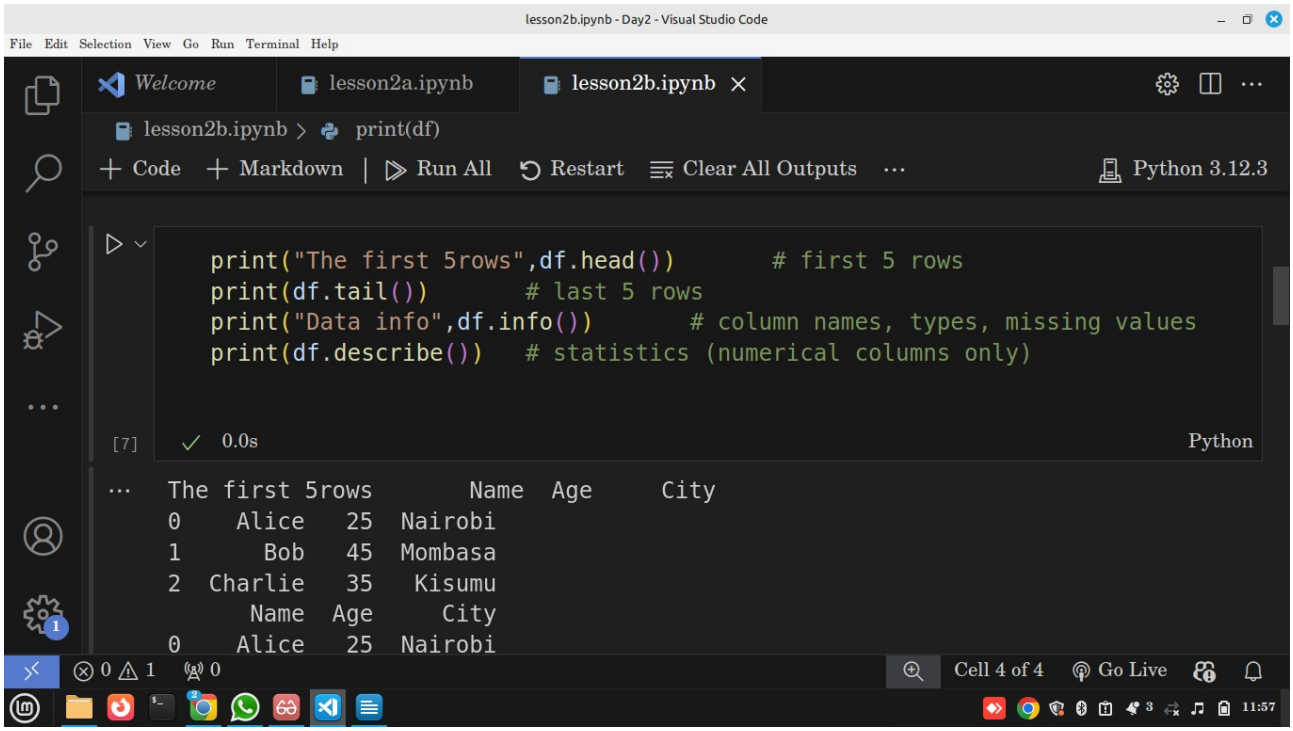
Explanation

- Each key in dict → column name.
- Each list → column values.
- Output is tabular, with row and column labels.

Use in Data Science

DataFrames are the **standard way** to store datasets for analysis.

Practical 2 – Inspecting Data



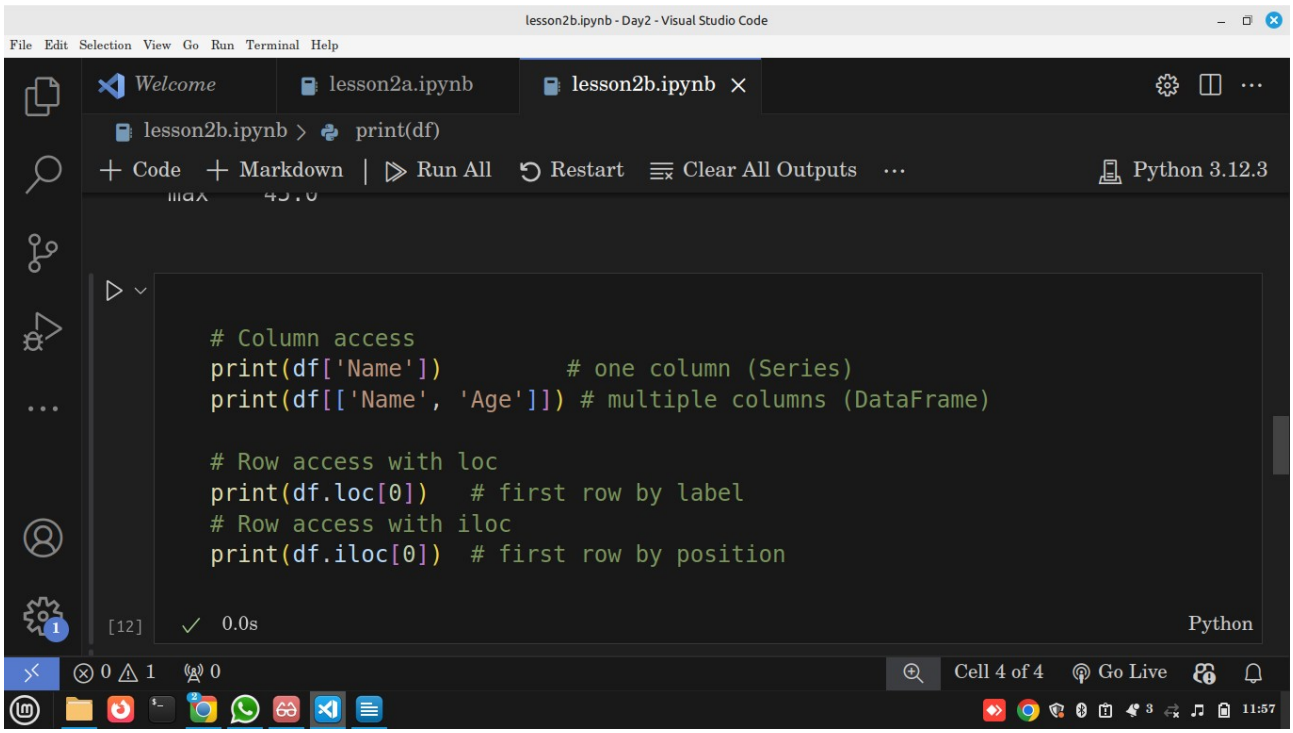
Explanation

- `.head()` and `.tail()` → quick dataset previews.
- `.info()` → useful for checking **data types & null values**.
- `.describe()` → summary statistics.

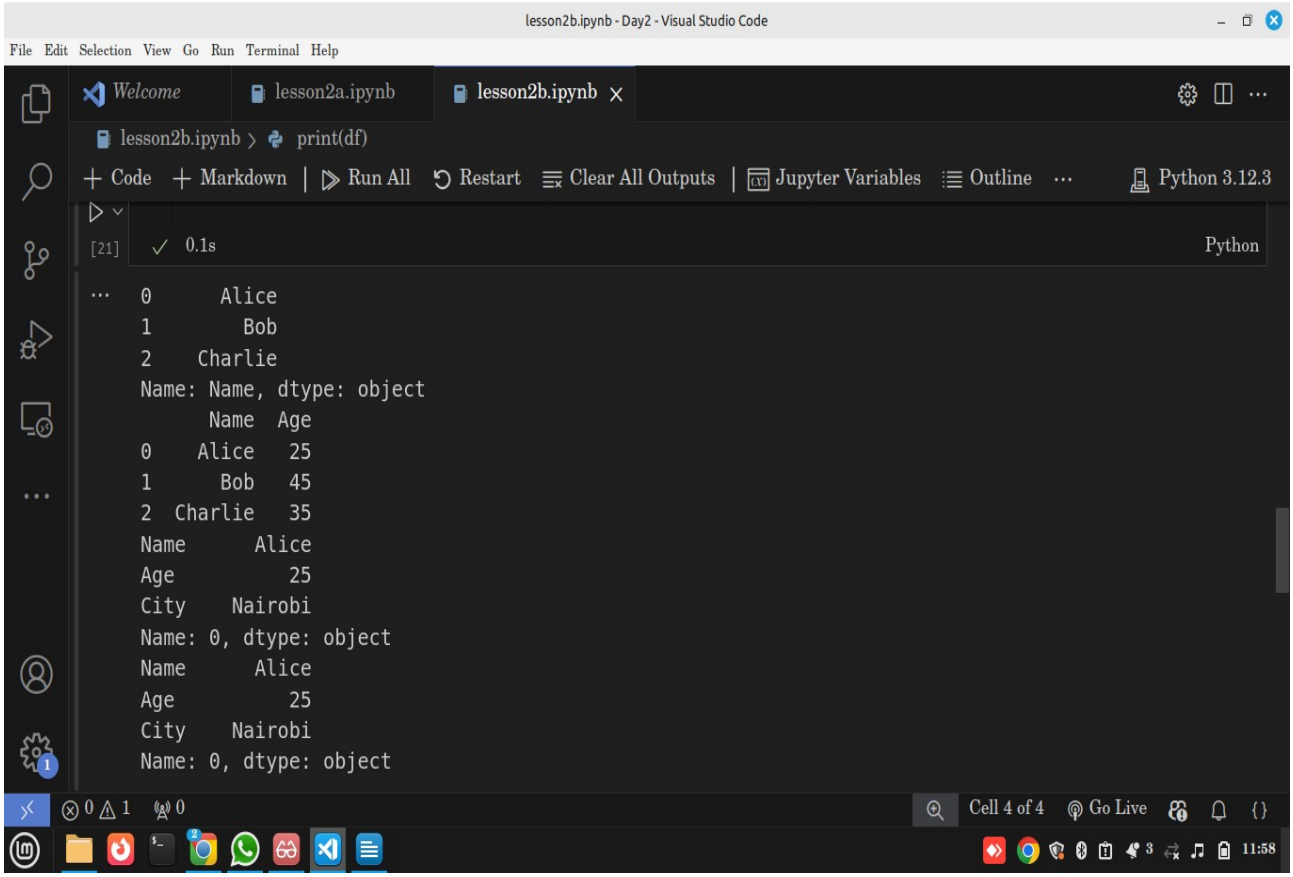
Use in Data Science

These are the **first commands** analysts run after loading a dataset.

Practical 3 – Accessing Columns & Rows



Output



Explanation

- Single column = Series.
- Multiple columns = DataFrame.
- `.loc` → label-based indexing.
- `.iloc` → position-based indexing.

Use in Data Science

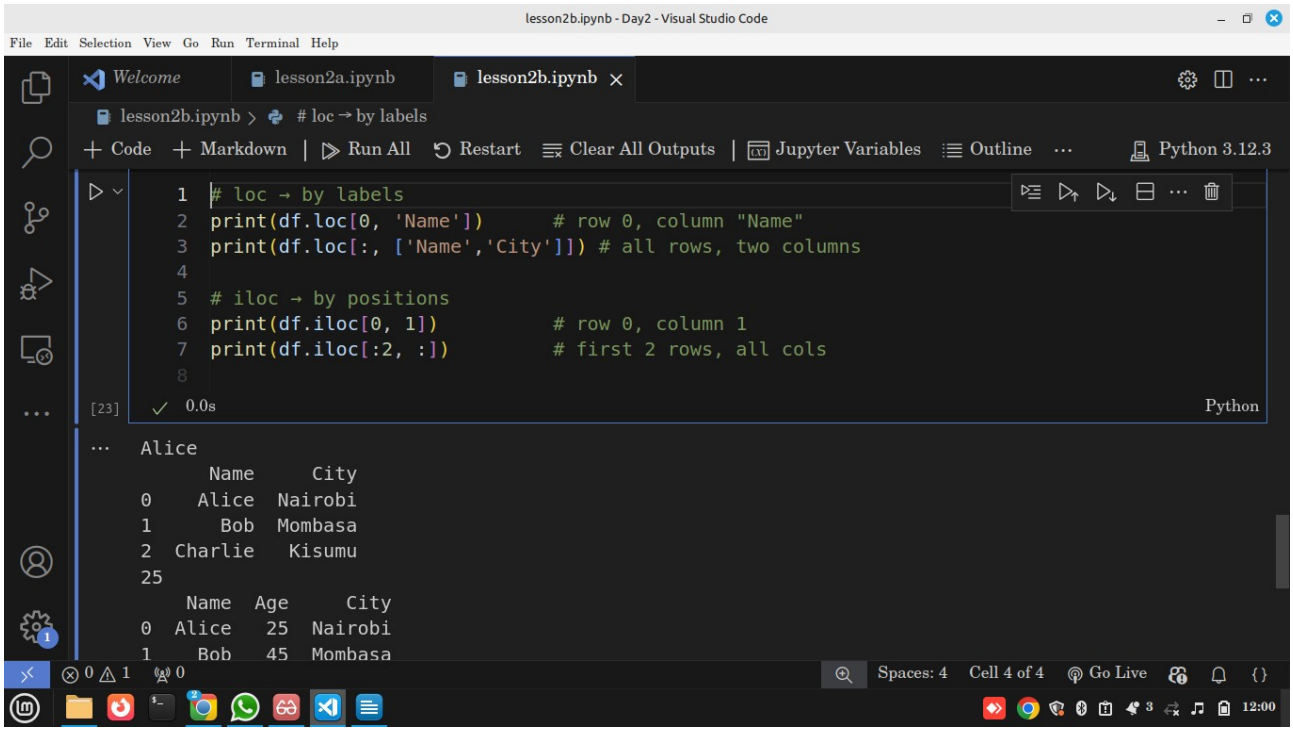
Lets us **select subsets** for analysis (e.g., only numeric columns for statistics).

🔹 Indexing: loc vs iloc

Notes

- `.loc[]` → label-based (row/column names).
- `.iloc[]` → integer-based (row/column positions).

Practical



Explanation

- `.loc[0, 'Name']` → fetches "Alice".
- `.iloc[0, 1]` → fetches first row, second column (Age = 25).

Use in Data Science

Used heavily in **subsetting datasets** for training/testing ML models.

🔍 Reflection

Why are DataFrames more powerful than NumPy arrays?