Session 1.3: Tabular Data Wrangling Part 1: NumPy Arrays, Pandas DataFrames, and CSV Ingestion

THE BUSINESS SITUATION

12 Stores, One Monthly Report

- Business systems share tables as Comma-Separated Values (CSV) files
- Our case: a retail chain exports monthly results for its 12 stores as a CSV file
- Each row: sales, costs, visitors, and rating for one store
- Your manager asks: which stores are most profitable, and which meet the target?

Today, Python answers these questions for every store at once.

Today's flow: 60 min lecture, then 120 min hands-on lab

EXPECTED OUTCOMES

By the end of today, you will be able to:

1. ***"Did the store data load correctly?"***
   Read and inspect a `DataFrame`
   `DataFrame`: a table in Python, like one Excel sheet
2. ***"What is each store's profit margin?"***
   Create new columns with vectorized math
   Vectorized: one formula runs on every row at once
3. ***"Which stores meet our target?"***
   Filter rows with `.loc[]` and fix a `KeyError`
   `.loc[]`: keep rows that match a condition
   `KeyError`: the column name you typed does not exist

Session Map: Follow the Numbers

Today's question: *Which stores are most profitable, and which meet the target?*

1 Why it matters
Lecture, part 1

1. **Session Purpose**
   answer today's question
2. **Learning Outcomes**
   3 skills you will show today
3. **Business Context**
   Why CSV?
4. **Data Basics**
   Rows and Columns

→

2 Learn the tools
Lecture, part 2

5. **Lab Dataset**
   1 CSV
6. **Keywords**
   6 terms
7. **Core Foundations**
   5 ideas, one at a time
8. **IDE Setup**
   name files, run in the Terminal
9. **Code Examples**
   3 demos + Quick Checks
10. **References**
    3 official docs

→

3 Practice
Lab

11. **Exercise 1: Guided**
    Retail Performance Wrangling
12. **Exercise 2: Applied**
    Find the Top Stores
13. **Exercise 3: Advanced**
    Store Scorecard

Why CSV? The Everyday Format of Business Data

WHERE DATA COMES FROM

Sales registers | Accounting system | Customer database | Excel reports

→ export

ONE SHARED FILE

`stores.xlsx` or `stores.csv`

`.xlsx` Excel file

Packed format: needs a spreadsheet program such as Excel

Excel	Notepad	Any text tool
✓	✗ unreadable symbols	✗ cannot read it

```
PK□□□□ ▒[Content_Types].xml
□▒□▒▒□▒□□▒□▒▒
```

Same data opened in Notepad: unreadable

vs

`.csv` CSV file

Plain text: opens almost anywhere

Notepad	Excel	Google Sheets	Python `pandas`	AI assistants
✓	✓	✓	✓	✓

```
Store_ID,Region_Name,Monthly_Sales_USD
STR-1000,East,142000.0
```

Same file opened in Notepad: still readable

Trade-off: CSV keeps only values, with no formulas and no formatting

What Are Rows and Columns?

COLUMN ↓

	index	Store_ID	Region_Name	Monthly_Sales_USD	+3 more
	0	STR-1000	East	142000.0	…
ROW →	1	STR-1001	West	**185000.0**	…
	2	STR-1002	North	98000.0	…
	3	STR-1003	South	115000.0	…
	4	STR-1004	East	210000.0	…
	5	STR-1005	West	175000.0	…
	… 6 more rows				

ROW: READ ACROSS →

One record

All the facts about one item: here, one store

Row `1` = everything about `STR-1001`

COLUMN: READ DOWN ↓

One type of information

The same fact for every store

In pandas, one column is a `Series`

CELL

One value

Where a row and a column meet: `185000.0`

Size of a table: `df.shape` returns `(12, 6)` = 12 rows, 6 columns. Row labels (the `index`) start at `0`, not 1.

One File, 12 Stores

THE FILE

`session_01_03_dataset.csv`

- In the `data/` folder
- 12 rows: one row per store
- 6 columns, listed on the right
- Used in: all demos and Exercises 1-3

THE COLUMNS

`Store_ID`	Store code, such as `STR-1001`
`Region_Name`	East, West, North, or South
`Monthly_Sales_USD`	Revenue for the month, in U.S. Dollars
`Operating_Cost_USD`	Monthly cost to run the store
`Customer_Footfall_Count`	Number of visitors in the month
`Customer_Satisfaction_Score`	Customer rating, 1.0 to 5.0

The store data is fictional teaching data, checked with a script before class.

Six Terms You Need Today

TOOLS

`pandas` and `NumPy`

What: free Python libraries for tables (`pandas`) and fast number arrays (`NumPy`)

Why: they sort, filter, and calculate tables of any size

INPUT

CSV file and `pd.read_csv()`

What: a plain-text table where commas separate values; `pd.read_csv()` loads it

Why: most business systems export data as CSV

TABLE

`DataFrame`

What: a whole table in Python, with row labels and column names

Why: it is the main object you clean, calculate, and filter

COLUMN

`Series`

What: one column of a DataFrame, such as `df["Monthly_Sales_USD"]`

Why: most calculations work column by column

SPEED

Vectorized Computation

What: one formula applied to every row at once, with no loop

Why: fast, and nothing to drag down row by row

SELECTION

`.loc[]` vs `.iloc[]`

What: `.loc[]` selects by label or condition; `.iloc[]` by position number

Why: pull out exactly the rows a question needs

Example: `df.iloc[0:2, 0:3]` = first 2 rows, first 3 columns

How a DataFrame Works

7.1

Columns, Not Cells

Excel: 12 copies of a formula, one per row

Remember: write the rule once; pandas applies it to every row

7.2

Index, Columns, Values

Index: row labels on the left, from `0`

Remember: counting starts at `0`

7.3

One Type per Column

`float64`, `int64`: numbers; `str`: text

Remember: check `df.dtypes` first

7.4

Check Before You Calculate

`.shape`, `.head()`, `.info()`, `.describe()`

Remember: look first, calculate second

7.5

`.loc[]` vs `.iloc[]`

`.loc[]`: label or condition

Remember: misspelled column name = `KeyError`

A DataFrame, Seen Up Close

1. COLUMNS, NOT CELLS

Excel: a copy in every row

	E (Margin %)
2	=(C2-D2)/C2*100
3	=(C3-D3)/C3*100
4	=(C4-D4)/C4*100
5	30

pandas: one formula

Operating_Margin_Pct
31.0
34.1
27.6
...

Red: `E5` was typed over, and the mistake is easy to miss

Green: one line of code fills every row (values rounded)

2. TABLE ANATOMY

	Store_ID	Region_Name	Monthly_Sales_USD
0	STR-1000	East	142000.0
1	STR-1001	West	185000.0
2	STR-1002	North	98000.0

Index: row labels **Columns**: names **Values**: the data

3. DATA TYPES (`dtype`)

`Store_ID`	`str` text
`Monthly_Sales_USD`	`float64` decimal number
`Customer_Footfall_Count`	`int64` whole number

Excel: `"1,200"` saved as text is quietly skipped by `SUM`

pandas: that column turns into text, and `.dtypes` shows it

Older pandas versions show `object` instead of `str`

4. QUICK INSPECTION: REAL RESULTS

`df.shape`	`(12, 6)`: 12 rows, 6 columns
`df.head(2)`	first 2 rows: `STR-1000`, `STR-1001`
`df.info()`	`12 non-null` in every column: no gaps
`df.describe()`	Sales: mean `152416.67`, min `89000.0`, max `225000.0`

Load a CSV File and Inspect It

YourName_S13_demo_1.py — Input → Process → Output

```
# 1. Input: load the tool and the data
import pandas as pd
df = pd.read_csv("data/session_01_03_dataset.csv")

# 2. Process: measure the table and take a preview
dimensions = df.shape
column_types = df.dtypes
preview = df.head(3)

# 3. Output: show the results
print("Dataset Dimensions (Rows, Columns):", dimensions)
print("Column Data Types:\n", column_types)
print(preview)
```

1. INPUT

`import` and `pd.read_csv()`

`import pandas as pd` loads pandas as `pd`

`pd.read_csv()` reads the file into a DataFrame `df`

2. PROCESS

`.shape`, `.dtypes`, `.head(3)`

Size, column types, and the first 3 rows

Why: confirm the data loaded correctly

3. OUTPUT

`print()`

Shows each result in the Terminal

Why: without `print()`, a script shows nothing

See the Last Rows with `.tail()`

NEW TOOL

`.tail(n)`

What: shows the last `n` rows of a DataFrame

Why: new records are often added at the bottom, so this is where you check them

	Store_ID	
0	STR-1000	
1	STR-1001	← `.head(3)`: first 3
2	STR-1002	
⋮	*6 rows*	
9	STR-1009	
10	STR-1010	← `.tail(3)`: last 3
11	STR-1011	

YOUR TASK

Change One Line

In `YourName_S13_demo_1.py`, part **2. Process**: show the **last** 3 rows instead of the first 3.

Now

```
preview = df.head(3)
```

Your change: fill in the yellow blank

```
preview = df._____(3)
```

Check: run the script; the preview shows index `9`, `10`, `11` (`STR-1009`, `STR-1010`, `STR-1011`).

Calculate New Columns Without a Loop

YourName_S13_demo_2.py — Input → Process → Output

```
# 1. Input: load the store data
import pandas as pd
df = pd.read_csv("data/session_01_03_dataset.csv")

# 2. Process: profit and margin for every store at once
df["Operating_Profit_USD"] = (
    df["Monthly_Sales_USD"] - df["Operating_Cost_USD"]
)
df["Operating_Margin_Pct"] = (
    df["Operating_Profit_USD"] / df["Monthly_Sales_USD"] * 100.0
)

# 3. Output: show the first three stores
print(df[["Store_ID", "Monthly_Sales_USD",
          "Operating_Margin_Pct"]].head(3))
```

1. INPUT

Same load step as Demo 1

Why: every script loads its own data before it can calculate

2. PROCESS

`df["Column_Name"]` with `-` `/` `*`

Picks a column (a `Series`); the math runs on all 12 stores at once

Why: no loop and no dragging a formula down

3. OUTPUT

`df[[ ... ]]` double brackets

A list of column names returns a smaller table

Why: show only what your reader needs

Round Numbers with `.round()`

NEW TOOL

`.round(n)`

What: rounds every number in a column to `n` decimal places

Why: long decimals are hard to read in a report

`30.985915` → `.round(1)` → **`31.0`**

YOUR TASK

Add One Line

In `YourName_S13_demo_2.py`, add one line at the end of **2. Process** so the margin has 1 decimal place.

Now

```
# (new line goes here)
```

Your change: fill in the yellow blank

```
df["Operating_Margin_Pct"] = (
    df["Operating_Margin_Pct"].______(1)
)
```

Check: the margins print as `31.0`, `34.1`, and `27.6`.

Filter Rows and Fill Missing Values

YourName_S13_demo_3.py — Input → Process → Output

```
# 1. Input: load the store data
import pandas as pd
df = pd.read_csv("data/session_01_03_dataset.csv")

# 2. Process: keep West stores with sales of $150,000+
west_high_sales = df.loc[
    (df["Region_Name"] == "West")
    & (df["Monthly_Sales_USD"] >= 150000.0)
]
#    then fill missing satisfaction scores with the median
median_score = df["Customer_Satisfaction_Score"].median()
df["Customer_Satisfaction_Score"] = (
    df["Customer_Satisfaction_Score"].fillna(median_score)
)

# 3. Output: show the filtered stores and the median
print(west_high_sales[["Store_ID", "Monthly_Sales_USD"]])
print("Median Satisfaction Score Imputed:", median_score)
```

1. INPUT

Same load step as Demo 1

Why: every script loads its own data first

2. PROCESS

`.loc[]`, `&`, `.fillna()`

- `df.loc[condition]` keeps rows where the condition is `True`
- `&` means "and"; wrap each condition in `( )`
- `.fillna()` fills empty cells (`NaN`) with the middle value (today's dataset has none)

3. OUTPUT

`print()` the answer

Shows the filtered stores and the median score

Count Rows with `len()`

NEW TOOL

`len()`

What: counts items; for a DataFrame, it returns the number of rows

Why: it answers "how many?" questions in one step

`df` → `len(df)` → `12`

The full table has 12 rows

YOUR TASK

Add One Line

In `YourName_S13_demo_3.py`, part **3. Output**: print how many stores passed the filter.

Now

```
# (new line goes here)
```

Your change: fill in the yellow blank

```
print("Stores found:", ____(west_high_sales))
```

Check: the Terminal shows `Stores found: 3`.

Where to Look Things Up

PANDAS

10 Minutes to pandas

A short tour: loading data, selecting columns, and simple math. pandas.pydata.org/docs/user_guide/10min.html

PANDAS

Indexing and Selecting Data

All the details of `.loc[]` and `.iloc[]`. pandas.pydata.org/docs/user_guide/indexing.html

NUMPY

NumPy Quickstart

The arrays that run under every pandas column. numpy.org/doc/stable/user/quickstart.html

Three Exercises

EXERCISE 1: GUIDED

Retail Performance Wrangling

Goal: Add an operating margin column and summarize stores by region

File:
`YourName_S13_wrangling.py`

Check: `count` 12; margins from `23.6`% to `36.9`%

EXERCISE 2: APPLIED

Find the Top Stores

Goal: Keep stores with margin `>= 30` and satisfaction `>= 4.5`

File:
`YourName_S13_practice.py`

Check: 5 stores, led by `STR-1011` (`36.89`%)

EXERCISE 3: ADVANCED

Store Scorecard

Goal: Sort all 12 stores into Gold, Watch, and Standard tiers

File:
`YourName_S13_challenge.py`

Check: Gold `4`, Watch `5`, Standard `3`

Full steps for all three exercises: sections 11-13 of the course page.