Session 1.4: Tabular Data Wrangling Part 2: Advanced Filtering, Multi-Segment GroupBy, and Financial Auditing

THE BUSINESS SITUATION

The Quarterly Budget Review

- Each division gets a spending budget every quarter
- Finance compares the budget with what was actually spent
- Our case: 4 divisions × 4 quarters = 16 rows in one CSV file
- The finance director asks: who overspent, by how much, and who is responsible?

Today, Python answers this for the whole year 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. ***"Which quarters went over budget?"***
   Filter rows with a Boolean mask and `&`
   Boolean mask: a `True`/`False` value for every row
2. ***"How much did each division overspend this year?"***
   Summarize groups with `groupby()` and `.agg()`
   `groupby()`: split rows into groups, like an Excel pivot table (a summary with one row per group)
3. ***"Who is responsible?"***
   Join tables with `pd.merge()` and fix a `KeyError`
   `pd.merge()`: join two tables on a shared column, like Excel's `VLOOKUP` lookup formula

Session Map: Follow the Numbers

Today's question: *Who overspent, by how much, and who is responsible?*

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**
   Budget vs Actual
4. **Data Basics**
   From Excel Tools You Know to pandas

→

2 Learn the tools
Lecture, part 2

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

→

3 Practice
Lab

11. **Exercise 1: Guided**
    Quarterly Division Audit
12. **Exercise 2: Applied**
    Annual Cost Center Overruns
13. **Exercise 3: Advanced**
    VP Exception Report

Budget vs Actual: How Finance Spots Overspending

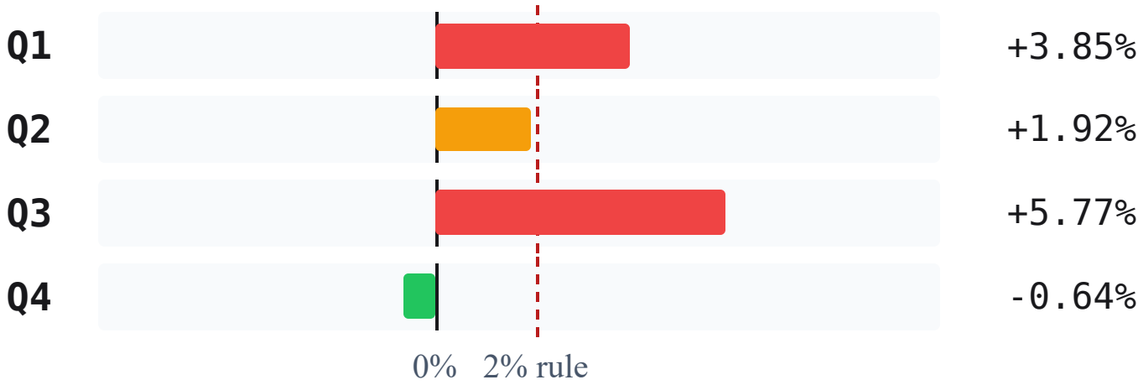
FOUR TERMS, ONE EXAMPLE

Consumer Division, Q1

Budget	planned spending	`780,000`
Actual	real spending	`810,000`
Variance	Actual - Budget	`+30,000`
Variance %	Variance / Budget × 100	`+3.85%`

- Positive variance = **overspent**; negative = under budget
- Cost center: the accounting code spending is charged to; each has a VP (Vice President)
- Company rule in our data: flag a quarter when variance is **more than 2%** (column `Audit_Status`)

CONSUMER DIVISION: VARIANCE % BY QUARTER



Red: flagged (over 2%) **Amber**: over, within rule

Green: under budget

Today's job: find these overruns for all 4 divisions, add up the year, and name the person responsible.

From Excel Tools You Know to pandas

EXCEL

Filter

Click the column arrow, tick the boxes

→

PANDAS

Boolean mask: `df[mask]`

The rule is written in code, so it repeats exactly next quarter

EXCEL

Pivot Table

A summary by group: `Division_Name` in Rows, spending in Values = 4 rows instead of 16

→

PANDAS

`df.groupby().agg()`

One line; run it again when new data arrives

EXCEL

VLOOKUP

A formula that looks up a code in another table and returns column **number** N, such as the 2nd

→

PANDAS

`pd.merge()`

Matches rows by column **name**, such as `Cost_Center_Code`

One File and One Small Table

THE FILE: `session_01_04_dataset.csv` (16 ROWS)

`Transaction_ID`	Record code, such as `TXN-5001`
`Division_Name`	Cloud, Consumer, Enterprise, Logistics
`Cost_Center_Code`	Accounting code, such as `CC-202`
`Fiscal_Quarter`	Q1 to Q4
`Budget_Expense_USD`	Planned spending
`Actual_Expense_USD`	Real spending
`Audit_Status`	`Variance_Flag` if more than 2% over budget

THE SMALL TABLE: `vp_info`

- Built in code in Demo 3 with `pd.DataFrame({...})`
- 4 rows: one VP (Vice President) per cost center
- VP names are fictional
- Used in: Demo 3, Exercises 2-3

Each CSV row is one division's spending in one quarter: 4 divisions × 4 quarters = 16 rows.

Six Terms You Need Today

FILTER

Boolean Mask

What: one `True` or `False` per row

Why: picks exactly the rows that meet a rule

COMBINE RULES

`&` `|` `~`

What: "and", "or", "not" for masks

Why: Python's `and` / `or` fail on whole columns

PATTERN

Split-Apply-Combine

What: split rows into groups, calculate each group, combine the results

Why: the idea behind every pivot table

SUMMARY

`groupby()` + `.agg()`

What: `groupby()` splits; `.agg()` sums, counts, or averages each group

Why: yearly totals per division in one step

JOIN

`pd.merge()`

What: joins two tables row by row

Why: adds facts from another table, such as the VP's name, with no copy-paste

MATCH

Key Column

What: a column both tables share, such as `Cost_Center_Code`

Why: tells pandas which rows belong together

How Filtering, Grouping, and Joining Work

7.1

`&`, Not `and`

A rule gives one `True` / `False` per row

Remember: combine with `&` `|` `~`, each rule in `( )`

7.2

Split, Apply, Combine

Consumer: 4 quarters, summed to `3205000.0`

Remember: `groupby()` splits, `.agg()` applies

7.3

Match by Name, Not Position

`VLOOKUP` returns "column 2": breaks if a column is inserted

Remember: the key needs the same name in both tables

7.4

`how="left"` vs `"inner"`

`left`: an unmatched row stays, with `NaN`

Remember: use `left`, then look for `NaN`

Filtering, Grouping, and Joining, Seen Up Close

1. `and` FAILS, `&` WORKS

```
is_consumer = df["Division_Name"] == "Consumer"
is_over = df["Actual_Expense_USD"] > df["Budget_Expense_USD"]
```

```
is_consumer and is_over
```

`ValueError: The truth value of a Series is ambiguous`

```
is_consumer & is_over
```

One `True` / `False` per row

2. APPLY: CONSUMER ACTUAL SPENDING

Actual_Expense_USD
810000.0
795000.0
825000.0
775000.0

→ `"sum"`

Total
3205000.0

Same step for all 4 divisions

3. VLOOKUP BREAKS WHEN A COLUMN IS INSERTED

Before

Code	VP
CC-202	Marcus Vance

After inserting Region

Code	Region	VP
CC-202	North	Marcus Vance

"2nd column" now returns "North": wrong, and easy to miss

4. `how="left"` KEEPS UNMATCHED ROWS

Cost_Center_Code	VP_Lead
CC-202	Marcus Vance
CC-404	David Kim
CC-505	*NaN*

A code with no VP (example `CC-505`) stays, with `NaN`

Find Overspending with a Boolean Mask

YourName_S14_demo_1.py — Input → Process → Output

```
# 1. Input: load the quarterly budget data
import pandas as pd
df = pd.read_csv("data/session_01_04_dataset.csv")

# 2. Process: Consumer quarters that spent more than budget
mask = (
    (df["Division_Name"] == "Consumer")
    & (df["Actual_Expense_USD"] > df["Budget_Expense_USD"])
)
overruns = df[mask]

# 3. Output: count and list them
print("Consumer overrun quarters:", len(overruns))
print(overruns[["Fiscal_Quarter", "Budget_Expense_USD",
                "Actual_Expense_USD"]])
```

1. INPUT

Same load step as Session 1.3

`pd.read_csv()` reads the 16 rows into `df`

2. PROCESS

`mask` and `df[mask]`

- `mask`: one `True` / `False` per row
- `&`: both rules must be `True`
- `df[mask]` keeps only the `True` rows

3. OUTPUT

`len()` and a short table

The count, then the matching quarters

Flip a Rule with `~`

NEW TOOL

`~` (not)

What: turns every `True` into `False`, and every `False` into `True`

Why: find the opposite group, such as quarters that stayed within budget

Fiscal_Quarter	Over budget?	After `~`
Q1	True	False
Q2	True	False
Q3	True	False
Q4	False	True

Consumer division; blue = `True` (kept)

YOUR TASK

Change One Line

In `YourName_S14_demo_1.py`, part **2. Process**: find Consumer quarters that did **not** go over budget.

Now

```
    & (df["Actual_Expense_USD"] > df["Budget_Expense_USD"])
```

Your change: fill in the yellow blank

```
    & ___(df["Actual_Expense_USD"] > df["Budget_Expense_USD"])
```

Check: the script prints `Consumer overrun quarters: 1`, and the table shows `Q4` (budget `780000.0`, actual `775000.0`). The label still says "overrun", since only the rule changed.

Total Each Division with `groupby()`

YourName_S14_demo_2.py — Input → Process → Output

```python
# 1. Input: load the quarterly budget data
import pandas as pd
df = pd.read_csv("data/session_01_04_dataset.csv")

# 2. Process: yearly totals for each division
div_summary = df.groupby("Division_Name").agg(
    Total_Budget_USD=("Budget_Expense_USD", "sum"),
    Total_Actual_USD=("Actual_Expense_USD", "sum"),
    Quarters=("Transaction_ID", "count"),
)
div_summary["Variance_USD"] = (
    div_summary["Total_Actual_USD"]
    - div_summary["Total_Budget_USD"]
)

# 3. Output: one row per division
print(div_summary[["Total_Actual_USD", "Variance_USD"]])
```

1. INPUT

Same load step

Every script loads its own data

2. PROCESS

`groupby()` + `.agg()`

- `groupby("Division_Name")`: 4 groups
- Inside `.agg()`: `New_Name=("Column", "sum")`
- Then one subtraction per division

3. OUTPUT

One row per division

4 rows instead of 16

Rank Results with `.sort_values()`

NEW TOOL

`.sort_values()`

What: reorders rows by one column; `ascending=False` puts the largest first

Why: the director reads the biggest overspender first

Division_Name		Division_Name
Cloud		?
Consumer	→	?
Enterprise		?
Logistics		?

Now: A to Z. After sorting: largest variance on top

YOUR TASK

Add One Line

In `YourName_S14_demo_2.py`, add one line at the end of **2. Process**: largest `Variance_USD` first.

Now

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

Your change: fill in the yellow blank

```
div_summary = div_summary.____________(
    "Variance_USD", ascending=False
)
```

Check: the first row is `Consumer` (`85000.0`), then `Logistics` (`28000.0`).

Add VP Names with `pd.merge()`

YourName_S14_demo_3.py — Input → Process → Output

```
# 1. Input: budget data and a VP list (fictional names)
import pandas as pd
df = pd.read_csv("data/session_01_04_dataset.csv")
vp_info = pd.DataFrame({
    "Cost_Center_Code": ["CC-101", "CC-202", "CC-303", "CC-404"],
    "VP_Lead": ["Sarah Chen", "Marcus Vance",
                "Elena Rostova", "David Kim"],
})

# 2. Process: add VP names, then flag spending over budget
ledger = pd.merge(df, vp_info, on="Cost_Center_Code", how="left")
ledger["Over_Budget"] = (
    ledger["Actual_Expense_USD"] > ledger["Budget_Expense_USD"]
)

# 3. Output: show the first three rows
print(ledger[["Transaction_ID", "VP_Lead", "Over_Budget"]].head(3))
```

1. INPUT

Two tables

- The CSV file
- `pd.DataFrame({...})` builds a small table in code: column name, then its list of values

2. PROCESS

`pd.merge()` + a flag

- `on=` names the key column
- `how="left"` keeps all 16 rows
- `>` makes a `True` / `False` column

3. OUTPUT

`.head(3)`

The first rows, now with `VP_Lead`

Count Categories with `.value_counts()`

NEW TOOL

`.value_counts()`

What: counts how often each value appears in a column

Why: answers "how many quarters were over budget?" in one line

Over_Budget
False
True
False
... 16 rows

→

Value	Count
False	?
True	?

One count per distinct value

YOUR TASK

Add One Line

In `YourName_S14_demo_3.py`, add one line to **3. Output**: count the `Over_Budget` values.

Now

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

Your change: fill in the yellow blank

```
print(ledger["Over_Budget"].[redacted]())
```

Check: the Terminal shows `False 8` and `True 8`: half of the 16 quarters went over budget.

Where to Look Things Up

PANDAS

GroupBy: Split-Apply-Combine

Everything `groupby()` and `.agg()` can do. pandas.pydata.org/docs/user_guide/groupby.html

PANDAS

Merge, Join, Concatenate and Compare

All the options of `pd.merge()`, including `how=`. pandas.pydata.org/docs/user_guide/merging.html

PYTHON STYLE GUIDE

PEP 8: Line Breaks Around Operators

Why we start a new line with `&` or `-` in long rules. peps.python.org/pep-0008

Three Exercises

EXERCISE 1: GUIDED

Quarterly Division Audit

Goal: Total each division's budget and actual spend by quarter

File: `YourName_S14_audit.py`

Check: 16 rows; 8 with a positive `Total_Variance`

EXERCISE 2: APPLIED

Annual Cost Center Overruns

Goal: Find the cost centers that overspent for the whole year

File: `YourName_S14_practice.py`

Check: `CC-202` (`85000.0`) and `CC-404` (`28000.0`)

EXERCISE 3: ADVANCED

VP Exception Report

Goal: Tell the finance director which VPs must explain their spending

File: `YourName_S14_challenge.py`

Check: `0` disagreements; 6 of 16 quarters flagged

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