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

PEDAGOGICAL AGENDA

1:2 Delivery Rhythm

Session Learning Architecture

Structured 3-hour applied computing session designed for business students with zero prior coding experience.

- **Phase 1 (60 Min Lecture):** Computational foundations, business vocabulary, and syntactic mechanics.
- **Phase 2 (120 Min Lab):** Guided hands-on implementation, error diagnosis, and AI co-pilot resolution.
- **Zero Fluff Mandate:** Focus directly on executable Python code, tabular data wrangling, and commercial intelligence.

STRATEGIC OUTCOME

Operational Autonomy

Business Analytical Competency

Transforming passive spreadsheet users into autonomous programmatic decision makers.

- **Speed & Scale:** Processing datasets exceeding spreadsheet limits with instantaneous vectorized execution.
- **Repeatable Pipelines:** Replacing manual copy-paste workflows with fully automated Python scripts.
- **AI Debugging Literacy:** Leveraging modern IDEs to diagnose and resolve errors independently.

Core Computational Keywords & Business Meaning

CONCEPT 01 — Core Keyword

Boolean Masking

Filtering tabular rows using boolean condition vectors combined with bitwise logical operators (`&` for AND, `|` for OR, `~` for NOT).

CONCEPT 02 — Core Keyword

Split-Apply-Combine Pattern

The computational workflow of splitting data into operational groups, applying analytical functions, and combining results into an aggregated table.

CONCEPT 03 — Core Keyword

Multi-Metric Aggregation (.agg())

Computing different statistical metrics simultaneously across multiple columns (e.g., total expenses, average variance, maximum transaction size).

CONCEPT 04 — Core Keyword

Relational Merging

Joining two DataFrames based on common business keys (such as `Cost_Center_Code`), replacing error-prone spreadsheet `VLOOKUP` and `XLOOKUP` functions.

Computational Foundations & Business Operations

PILLAR 01 Architecture

**Core Programming & Computational Foundations

**

PILLAR 02 Architecture

Boolean Indexing at Enterprise Scale

In business reporting, filtering data across multiple conditions requires bitwise operators (`&`, `|`) because standard Python `and`/`or` operate on single scalar booleans, whereas Pandas evaluates element-wise boolean arrays.

PILLAR 03 Architecture

The Split-Apply-Combine Paradigm

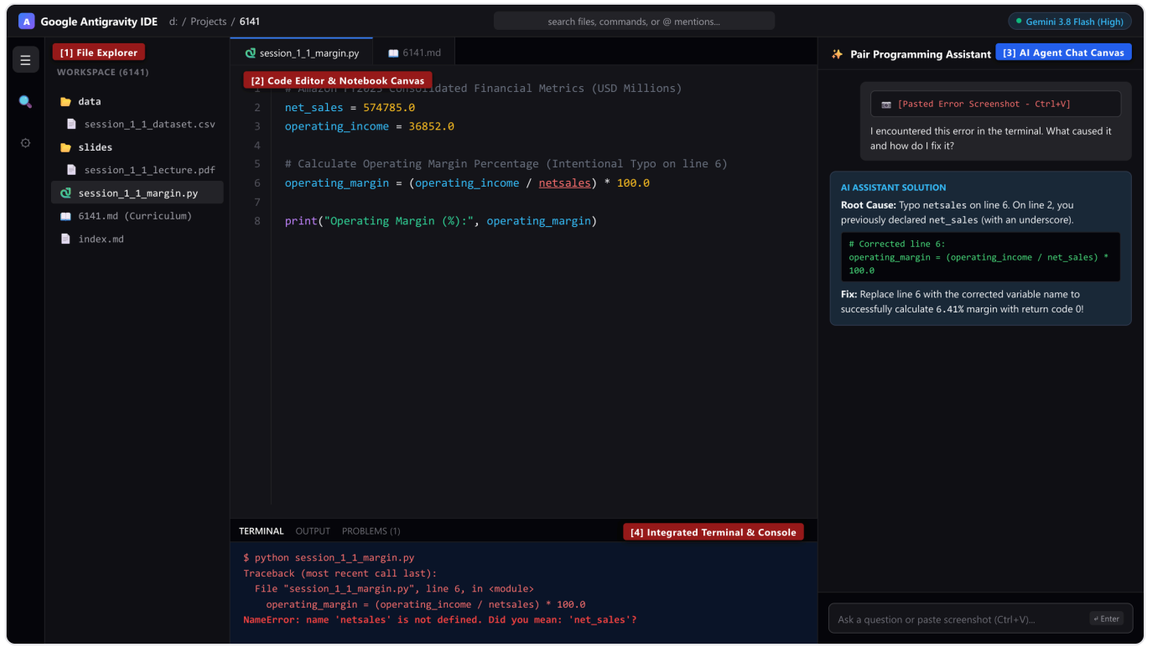
Grouping data by corporate dimensions (e.g., `Division_Name`, `Fiscal_Quarter`) to calculate sub-totals and unit economics without manually building pivot tables.

PILLAR 04 Architecture

Relational Joins vs. Fragile Spreadsheet Lookups

Spreadsheet lookup formulas fail when columns are rearranged or when duplicate keys exist. A Pandas merge (`how="inner"`, `how="left"`) explicitly specifies match criteria and logs unmapped rows automatically.

Google Antigravity IDE Architecture & AI Debugging Protocol



[1] File Explorer

Project Hierarchy

Manage datasets in `data/`, scripts in root, and exported artifacts.

[2] Code Editor

Syntax Workspace

Modern syntax highlighting, lint diagnostics, and line-by-line script authoring.

[3] AI Agent Chat

AI Pair Programmer

Paste error screenshots directly to inspect root causes and get verified fixes.

[4] Terminal

Execution Engine

Run Python scripts (`python script.py`) and view console logs and tracebacks.

Zero-Friction AI Error Resolution Protocol

1. **Capture:** Press `Win + Shift + S` to clip terminal error traceback.
2. **Paste:** Press `Ctrl + V` into AI Chat [3] with "Explain root cause & fix".
3. **Apply:** Review the explanation and apply verified fix into Editor [2].

Script Demonstration 1: Script Demonstration 1

session_1_4_demo_1.py — Python 3.10

```python
import pandas as pd

    df = pd.read_csv("data/session_1_4_dataset.csv")
    mask = (df["Division_Name"] == "Consumer") & (df["Actual_Expense_USD"] > df
    overrun_txns = df[mask]
    print(f"Consumer Overrun Transactions Count: {len(overrun_txns)}")
```

```
$ python session_1_4_demo_1.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Script Demonstration 2: Script Demonstration 2

session_1_4_demo_2.py — Python 3.10

```
div_summary = df."fn">groupby("Division_Name")."fn">agg(
        Total_Budget_USD=("Budget_Expense_USD", "sum"),
        Total_Actual_USD=("Actual_Expense_USD", "sum"),
        Transaction_Count=("Transaction_ID", "count")
    )
    div_summary["Variance_USD"] = div_summary["Total_Actual_USD"] - div_summary
    div_summary["Variance_Pct"] = (div_summary["Variance_USD"] / div_summary["T
    "fn">print(div_summary[["Total_Actual_USD", "Variance_USD", "Variance_Pct"]
```

```
$ python session_1_4_demo_2.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Script Demonstration 3: Script Demonstration 3

session_1_4_demo_3.py — Python 3.10

```
dept_info = pd.DataFrame({
        "Cost_Center_Code": ["CC-101", "CC-202", "CC-303", "CC-404"],
        "VP_Lead": ["Sarah Chen", "Marcus Vance", "Elena Rostova", "David Kim"]
    })
    merged_ledger = pd.merge(df, dept_info, on="Cost_Center_Code", how="left")
    "fn">print(merged_ledger[["Transaction_ID", "Division_Name", "VP_Lead", "Ac
```

```
$ python session_1_4_demo_3.py [Exit 0]
```

1 Syntactic Structure

Importing production libraries and defining clean analytical data structures.

2 Vectorized Transformation

Applying deterministic operations across the full dataset without fragile cell references.

3 Console Verification

Emitting formatted business KPIs to the terminal for auditability and compliance.

Interactive Two-Tier Lab Protocol & AI Diagnosis

EXERCISE 1

Guided Lab

Foundational Implementation

Step-by-step guided practice establishing baseline syntax mastery in Antigravity IDE.

- **Step 1:** Ingest dataset from `data/session_1_4_dataset.csv`.
- **Step 2:** Implement core analytical functions and compute primary business metrics.
- **Step 3:** Format console outputs and verify calculations against baseline ledger totals.
- **Step 4:** Run in Terminal [4] and confirm clean execution with exit code 0.

EXERCISE 2+

Coding Challenge

Advanced Scripting & AI Debugging

Applied programming challenge expanding pipeline logic and resolving intentional exceptions.

- **Task 1 (Pipeline Expansion):** Add secondary business unit logic and multi-tier calculations.
- **Task 2 (Deliberate Error & AI Capture):** Introduce intentional syntax or type error, capture traceback with `Win + Shift + S`, and diagnose via AI Chat [3].
- **Task 3 (Verified Production Run):** Apply verified fix, export audit output, and confirm zero errors with return code 0.

The Pedagogical Contract

Computational mastery develops through active problem-solving. By creating intentional errors and using AI co-pilots to resolve them, students build lifelong debugging autonomy.