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

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

Vectorized Computation

Executing mathematical operations across entire numeric arrays or data columns simultaneously in compiled C memory, replacing slow Python row-by-row loops.

CONCEPT 02 — Core Keyword

Pandas Series

A one-dimensional labeled array in Pandas capable of holding any data type, representing a single column in an enterprise tabular dataset.

CONCEPT 03 — Core Keyword

Pandas DataFrame

A two-dimensional, size-mutable tabular data structure with labeled axes (rows and columns), serving as the primary analytical workhorse for commercial data.

CONCEPT 04 — Core Keyword

Tabular CSV Ingestion

Reading structured comma-separated corporate ledgers directly into memory using `pd.read_csv()`, parsing data types and column headers automatically.

Computational Foundations & Business Operations

PILLAR 01 Architecture

**Core Programming & Computational Foundations

**

PILLAR 02 Architecture

Vectorized Array Math vs. Row-by-Row Iteration

While Python `for` loops are intuitive, calculating metrics across 500,000 retail records using a loop is computationally slow. NumPy and Pandas use vectorized execution, operating on entire memory blocks in parallel, reducing execution time from seconds to milliseconds.

PILLAR 03 Architecture

The Two-Dimensional DataFrame Structure

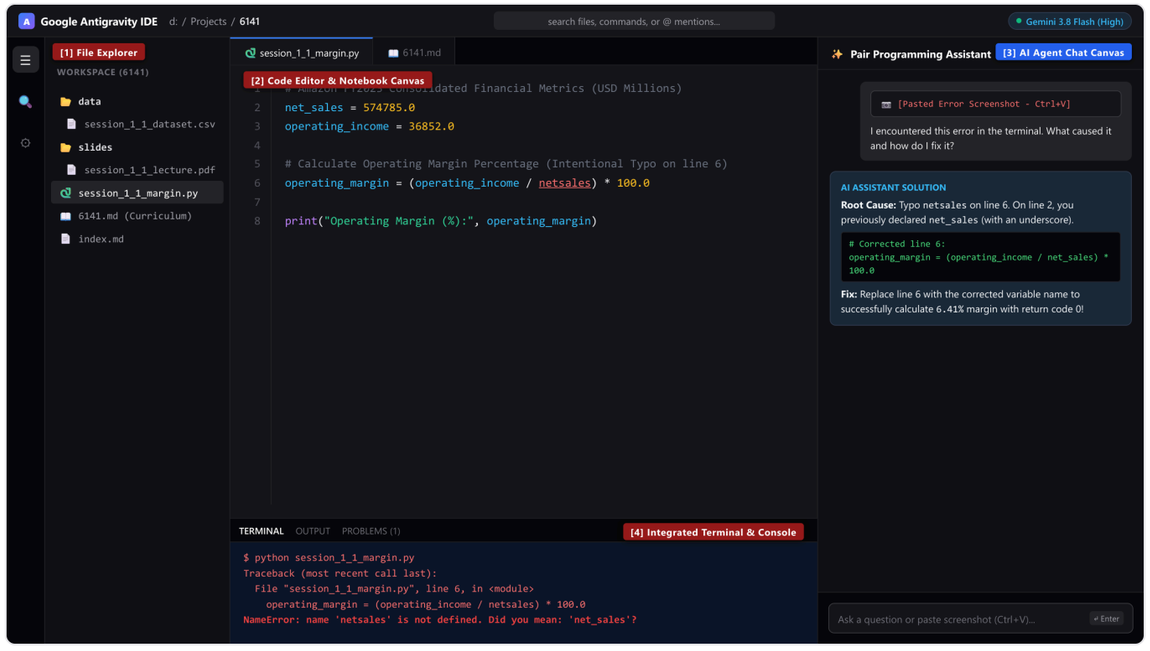
Bridging spreadsheet grids into structured programmatic objects. A Pandas DataFrame consists of an Index (row identifiers), Columns (attribute headers), and Values (underlying two-dimensional NumPy array).

PILLAR 04 Architecture

Explicit Data Types and Memory Management

Unlike spreadsheets where a cell can silently mix text and numbers, Pandas enforces explicit column data types (`float64`, `int64`, `object`, `datetime64`). This prevents silent financial corruption during aggregation.

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_3_demo_1.py — Python 3.10

```python
import pandas as pd

    df = pd.read_csv("data/session_1_3_dataset.csv")
    print("Dataset Dimensions (Rows, Columns):", df.shape)
    print("Column Data Types:\\n", df.dtypes)
    print(df.head(3))
```

```
$ python session_1_3_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_3_demo_2.py — Python 3.10

```
df["Operating_Profit_USD"] = df["Monthly_Sales_USD"] - df["Operating_Cost_USD"]
    df["Operating_Margin_Pct"] = (df["Operating_Profit_USD"] / df["Monthly_Sales
    "fn">print(df[["Store_ID", "Monthly_Sales_USD", "Operating_Margin_Pct"]]."fr
```

```
$ python session_1_3_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_3_demo_3.py — Python 3.10

```
# Select high-revenue stores "kw">class="kw">in the West region
    west_high_sales = df.loc[(df["Region_Name"] == "West") & (df["Monthly_Sales_

    # Impute missing satisfaction scores "kw">with median value
    median_score = df["Customer_Satisfaction_Score"].median()
    df["Customer_Satisfaction_Score"] = df["Customer_Satisfaction_Score"].fillna
    "fn">print("Median Satisfaction Score Imputed:", median_score)
```

```
$ python session_1_3_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_3_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.

Interactive Lab Protocol & Delivery Rhythm

Hands-On Code Execution

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.