Session 2.4: Midterm Analytics Pipeline Defense Part 2: Exploratory Findings and Executive Review

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

Reproducible Data Pipeline

A self-contained, end-to-end Python script execution sequence that reliably transforms raw data into verified executive reports with zero manual intervention.

CONCEPT 02 — Core Keyword

Executive Code Defense

Articulating the business rationale, methodology, and computational assumptions behind an analytics codebase to executive decision-makers.

CONCEPT 03 — Core Keyword

Exploratory Data Findings

Structuring quantitative insights into concise, actionable findings linked directly to operating cash flows and margin expansion.

CONCEPT 04 — Core Keyword

Cross-Functional Technical Translation

The ability to explain computational trade-offs, data limitations, and algorithmic results to non-technical stakeholders.

Computational Foundations & Business Operations

PILLAR 01 Architecture

**Core Programming & Computational Foundations

**

PILLAR 02 Architecture

Code Auditability in High-Stakes Environments

In corporate finance and strategic planning, unverified analytics pipelines create catastrophic financial risks. A formal code defense requires students to demonstrate that every calculation traces back to audited primary records.

PILLAR 03 Architecture

Communicating Analytics to Non-Technical Leadership

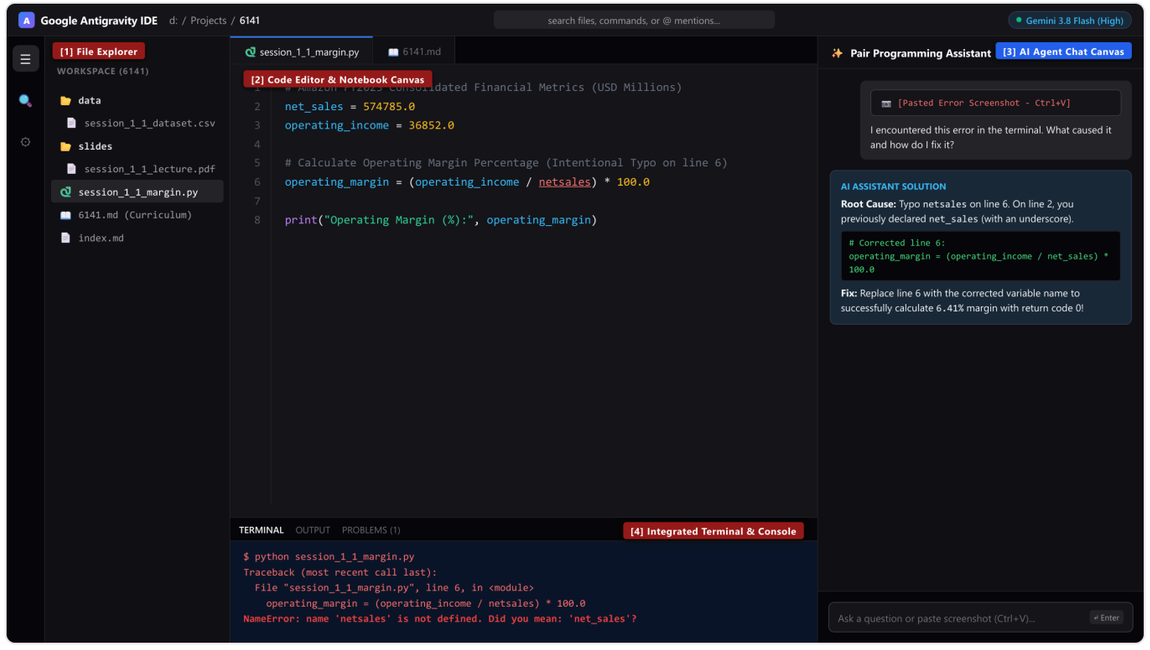
Senior leadership rarely reads raw Python code. Analysts must present clear visual exhibits, executive summary metrics, and explicit strategic trade-offs, maintaining code as the verifiable engine underneath.

PILLAR 04 Architecture

Peer Review and Code Auditing

Evaluating external codebases builds critical debugging and quality assurance skills. Analysts inspect code for hard-coded assumptions, fragile cell dependencies, and unhandled exception paths.

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_2_4_demo_1.py — Python 3.10

```
"kw">class="kw">import time
    "kw">class="kw">import pandas "kw">class="kw">as pd

    start_time = time.time()
    df = pd."fn">read_csv("data/session_2_4_dataset.csv")

    # Compute platform take rate and cash flows
    df["Recognized_Revenue_USD_M"] = df["Gross_Merchandise_Value_USD_M"] * (df
    df["Cash_Flow_Conversion_Pct"] = (df["Operating_Cash_Flow_USD_M"] / df["Re

    elapsed_time = time.time() - start_time
    "fn">print(f"Pipeline Executed ">class="kw">in {elapsed_time:.4f} seconds
```

```
$ python session_2_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_2_4_demo_2.py — Python 3.10

```python
segment_audit = df."fn">groupby("Market_Segment")."fn">agg(
        Total_GMV_USD_M=("Gross_Merchandise_Value_USD_M", "sum"),
        Total_Revenue_USD_M=("Recognized_Revenue_USD_M", "sum"),
        Total_Operating_Cash_Flow_USD_M=("Operating_Cash_Flow_USD_M", "sum"),
        Mean_Take_Rate=("Platform_Take_Rate_Pct", "mean")
    )
    "fn">print("Geographic Cash Flow Conversion Audit:\\n", segment_audit)
```

```
$ python session_2_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_2_4_demo_3.py — Python 3.10

```
"kw">class="kw">import hashlib

    summary_str = segment_audit.to_string()
    audit_hash = hashlib.sha256(summary_str.encode("utf-8")).hexdigest()[:12]
    segment_audit.to_csv("data/midterm_defense_summary.csv")
    "fn">print(f"Audited Executive Summary Exported. Verification Hash: {audit_
```

```
$ python session_2_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_2_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.

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.