Lab Worksheet: AI-Driven Blind Data Exploration

Anaconda Workspace Assignment • Metadata Prompting & Feature Relation Scans

Participant Name: _____ **Date:** _____

Section 1: The Core Methodology of Blind Exploration

Blind exploration involves a high-leverage data analysis paradigm: feeding high-level metadata—specifically column names and structured data types—into an AI model to gauge initial feature potential and draft processing paths before performing heavy computational analysis on the underlying records.

By abstracting the schema shape early, an analyst can use Large Language Models (LLMs) to identify hidden relationships between disparate fields and formulate concrete hypothesis frameworks without exposing sensitive row-level client information.

Section 2: The Step-by-Step Prompting Framework

To perform an auditing pass inside your Jupyter environment, follow this structured procedural pipeline:

Step 1: Extract Schema & Sample Values

Isolate structural properties and pull brief non-sensitive samples out of your active DataFrame using programmatic methods like `df.info()` or `df.head(2)`.

Step 2: Declare Domain Context

Provide specific real-world domain clarity to guide the AI's semantic mapping engine. Example: *"This is a retail dataset reflecting omni-channel purchase funnels."*

Step 3: Execute Target Story Querying

Issue a specific open-ended exploration prompt to target structural anomalies: *"What are the three most interesting stories or trend vectors hidden in this data schema?"*

Section 3: Guided Code Environment

Run this setup code in your Jupyter Notebook cell block to isolate and package structural data features into text format:

```
import pandas as pd

# Creating a high-dimensional retail transactional summary matrix
retail_data = {
    'Invoice_No': [536365, 536366],
    'Stock_Code': ['85123A', '71053'],
    'Store_Region': ['UK_Online', 'EMEA_Retail'],
```

```
    'Customer_Account_Age_Days': [412, 18],
    'Gross_Basket_Value_USD': [154.20, 22.10],
    'Discount_Applied_Flag': [True, False]
}

df = pd.DataFrame(retail_data)

# Extract technical data schema components to a clean markdown or string format
import io
buffer = io.StringIO()
df.info(buf=buffer)
schema_string = buffer.getvalue()

print("--- Markdown Schema to Copy into Your AI Prompt ---")
print(schema_string)
```

Section 4: Verification & Analysis Exercises

Execute the code framework in Section 3, then evaluate the resulting schema attributes below.

Question 1: Feature Type Analysis

Identify the data classifications present in the output. Which data type did Pandas automatically assign to the `'Discount_Applied_Flag'` column, and which type handles the raw geographic categories in `'Store_Region'`?

Question 2: Bridging Disparate Fields

Review the printed column catalog visually. Identify at least two fields that appear structural or separate on the surface but can be linked together logically to uncover a consumer trend story (e.g., matching promotional actions against user loyalty tracking metrics):

Question 3: Security & Performance Trade-offs

Why is it highly advantageous to perform a "blind exploration" pass on column configurations and schemas before feeding thousands of raw CSV data rows directly into an AI prompt assistant? Discuss security and memory efficiency constraints:

Section 5: Practical Prompt Construction Sandbox

🚀 **Project Task:** Using the structural guidelines from Section 2 and your schema output from Section 3, write a complete, raw data audit prompt inside the box below.
Your constructed text must combine **dataset schema metrics**, clear **retail domain context**, and an explicit instruction to find **hidden relationships**.

Write Your Composite Exploration Prompt Below:

```
# Draft your system prompt text here to paste directly into your AI modeling platform:
```

Interactive Help Reference

- `df.info()` → Prints a concise technical summary of a DataFrame, including index data type, column types, non-null values, and memory usage metrics.
- `df.columns.tolist()` → Quickly returns a clean, low-overhead string list of all category names for fast prompt engineering actions.