Data Cleansing: Finding Cross-Column Duplicates

Anaconda Platform Training Guide & Student Lab Worksheet

Participant Name: _____ **Date:** _____

Section 1: Conceptual Architecture

When working with consolidated customer ecosystems, records are often merged from multiple source tools. Consequently, a unique enterprise identity may accidentally possess overlapping system IDs across separate tracker fields (e.g., matching codes sitting concurrently inside `Legacy_ID`, `CRM_ID`, and `Billing_ID`).

Row-Level (Internal) vs. Global (Cross-Row) Matching

To resolve data pollution effectively, analysts must distinguish between two primary matching topologies:

Topology A: Row-Level Mirroring

```
Name: Bob
ID_1: B456 ←── [Identical Match]
ID_2: B456 ←── [Same Record]
ID_3: W555
```

Topology B: Global Cross-Row Pool

```
Row 1 (Alice): ID_1 = A123 ←─+
Row 2 (Chris): ID_2 = X999 |
Row 3 (Epsilon): ID_3 = A123 ←+ [Global Match]
```

Section 2: The Core Pandas Toolbox

The following table defines the specific toolkit used within the Anaconda Environment to parse multi-dimensional duplicate layers:

Method	Functional Execution Purpose within This Workflow
`df[['Col1', 'Col2']]`	Isolates specified identifier paths to avoid processing noisy fields like names or values.
`.stack()`	Compresses multi-column structural planes into a single vertical multi-indexed Vector.
`.duplicated()`	Scans downstream values sequentially, evaluating repeated instances as boolean `True`.
`.any(axis=1)`	Performs horizontal bitwise checks; flags a row if at least one target column resolves to true.

Section 3: Code Implementation Guide

Execute the following programmatic building blocks within your Jupyter Notebook workspace to systematically map out overlapping instances.

Step 3.1: Environment Initialization

```
import pandas as pd

# Simulating merged corporate tracking profiles
data = {
    'Customer_Name': ['Alpha Corp', 'Beta Inc', 'Gamma LLC', 'Delta Co', 'Epsilon Ltd'],
    'Legacy_ID':     ['A123', 'B456', 'C789', 'D012', 'E345'],
    'CRM_ID':        ['X999', 'B456', 'Y888', 'D012', 'Z777'],
    'Billing_ID':    ['A123', 'W555', 'Y888', 'V444', 'E345']
}

df = pd.DataFrame(data)
print(df)
```

Step 3.2: Row-Level Evaluation

```
# Identify entities whose tracking codes are instantly mirrored in the same record row
row_level_matches = df[df['Legacy_ID'] == df['CRM_ID']]
print(row_level_matches)
```

Step 3.3: Flattening and Isolating the Global Pool

```
id_cols = ['Legacy_ID', 'CRM_ID', 'Billing_ID']

# Flatten matrix arrays into 1D space
stacked = df[id_cols].stack()

# Extract uniquely corrupted identification tags
corrupted_ids = stacked[stacked.duplicated()].unique()
print(f"Flagged IDs: {corrupted_ids}")
```

Section 4: Practice Lab & Assessment Worksheet

Complete the steps outlined in Section 3 inside your local Jupyter Notebook instance, then answer the verification items below.

Question 1: Visual Inspection

Based on the initial printed representation of the `df` DataFrame, write down at least two identifier codes that appear in more than one system tracking column across the entire dataset:

Question 2: Runtime Code Analysis

When you executed Step 3.2, which customer profile entity was printed out as containing matching `Legacy_ID` and `CRM_ID` elements within their single unified profile row?

Question 3: Mechanics of the Stack Method

Explain briefly what happens to the multi-column dimensionality of `df[id_cols]` after running the `.stack()` function expression:

> **🚀 Section 5: Hands-On Independent Challenge**
>
> Using your local runtime instance, write a filtering script to extract the fully populated original customer records belonging to any account flagged with elements inside `corrupted_ids`.
>
> **Expected Output:** The execution filter must return 4 out of the 5 original corporate listings.

Write Your Solution Script Below:

```
# Provide your lookup filter solution code here:
```

Keyboard Reference Shortcuts

- `Shift + Enter` → Execute the currently highlighted cell block and advance the focus block downstream.
- `Esc → M` → Convert an active code block box immediately into structured descriptive Markdown text.
- `Esc → A / B` → Programmatically insert a blank development execution block Above / Below.