GRECO SOLUTION USING PYTHON

PROBLEM STATEMENT DESCRIPTION:

A pharmacy analytics team needs to build a **dimensional table** that tracks each patient's longitudinal engagement with the pharmacy based on their purchase patterns. This table will be used for downstream patient-journey dashboards and re-engagement analysis.

You are required to build a **slowly changing, rule-driven dimensional table** that determines a patient's **daily status**, **recent status classification**, **transition dates**, and **previous store number**, based on purchase behavior.

Input Source data

- **sale_table** (Contains patient purchases across multiple stores with at least:)
  - entrp_ptnt_id
  - store_nbr
  - sale_dt

Output Source data

- A output table with the following columns of data:
  Build a eff_dt grained dimensional table with the following columns:

Column Name	**Description**
entrp_ptnt_id	Patient ID
eff_dt	Effective date (one row per patient per effective change)
status	Daily status: Active, Inactive, Lapsed, or Lost
recent_status	Sub-classification for Active: Recently New, Recently Reactivated, Active Continuing or for Inactive: Inactive Continuing
transition_dt	Date when the recent_status changed (equals eff_dt only on change)
prev_store_nbr	Store number of the last purchase before the current eff_dt

TASKS REQUIRED TO BE DONE:

1. Process data using Python(pandas)
2. Prepare dashboard using Python.

TECH USED:

1. **Pandas –** for data processing and creating the output file (dim_patient_status) table.
2. **Flask –** backend framework in python used to develop a server.
3. **HTML, CSS, JS(plotly) –** frontend has been developed using HTML, CSS, and the charts have been implemented using the plotly library in js.

APPROACH USED:

1. **Dataset Generation**
   Generated a dummy dataset(sale_table) using the attributes provided in the document.
   The dataset has the 3 columns and 38 rows.

2. **Loading the Data within a Dataframe**
   Loaded the dataset into a dataframe using pandas within a ipynb file(greco2.ipynb)

3. **Grouping and Ordering data**
   Grouped the data with respect to the 'entrp_ptnt_id' and ordered it in ascending order using the 'sale_dt'

4. **Initialization of pilot dummy row**
   Determined the first sale_dt per entrp_ptnt_id by determining the minimum date. Then Initialized a dummy row before each patient's first purchase having the sale_dt which is 3 years older than the first purchase.
   Initialized the other columns like Status, Recent_Status as 'Unknown' and prev_store_nbr = NA and NaN for the other rows.

5. **Logic for 'prev_sale_dt' column**

   Created a column as 'prev_sale_dt and determined the previous purchase date by grouping the entrp_ptnt_id and shifting it by one position.
   Created a column as 'days_since_previous_purchase'.
   Calculated the difference between the current sale_dt and prev_sale_dt.

6. **Logic for 'status' column**

   Created a new column called status.
   Created a function called assign_status and passed the 'days_since_previous_purchase' as an argument and created the following loop (pseudocode provided below) for determining the status:

```
if pd.isna(days)            then "Unknown"
#first purchase only
if days >= 1095             then "Active"
#other subsequent purchases
if days <= 110              then "Active"
elif days <= 365            then "Inactive"
elif days < 1095            then "Lapsed"
else:                       then "Lost"
```

7. **Logic for 'recent_status' column**

   Now for the recent_status column the following pseudocode logic was used:

```
if the 'days_since_prev_purchase' (d) is >= 1095        then "Recently New"
if d >= 366 and d < 1095:                               then "Recently Reactivated"
if d <= 110:                                            then "Active Continuing"
if d >= 111 and d <= 365:                               then "Inactive Continuing"
```

8. **Logic for 'eff_dt' col**

   Created eff_dt column
   Created a prev_recent_status column and prev_store_nbr and filled it by shifting the data rows by 1 position.
   then the recent_status is compared with prev_recent_status
   and store_nbr is compared with prev_store_nbr if either one is different then the effective date (eff_dt) is updated with the current sale date.
   If the data is not available then they are labelled as NA after converting it to

string format.

9. **Logic for ‘previous_store_nbr’ column**
   The previous_store_nbr column is created.
   A function called prev_store_nbr is created and first convert sale_dt and eff_dt into datetime formats for accurate comparison.
   For each patient (entrp_ptnt_id), process their purchase history independently.
   Iterate through each row in chronological order for that patient.
   Read the row’s effective date (eff_dt_dt).
   If eff_dt_dt is missing (NaT), assign "NULL" as prev_store_nbr.
   Otherwise, identify all earlier purchases where sale_dt is strictly before eff_dt_dt.
   If no earlier purchases exist, assign "NULL".
   If earlier purchases exist, pick the **last** store from those rows.
   If that previous store is missing or "Unknown", return "NULL"; otherwise return that store number.
   Remove the helper datetime column and return the updated dataframe.

10. **Filtering data into Final Output format**
    Create sale_date (date part) and find index of the last sale_dt per (entrp_ptnt_id, sale_date).
    Build df_last_of_day containing only those last-of-day rows, sorted by patient and datetime.
    Compute shifted previous values per patient: prev_store_nbr_shift and prev_recent_status_shift.
    Create masks: first_row_mask, status_changed_mask, and store_changed_mask.
    Combine masks to keep_mask and keep rows where any condition is true → df_kept.
    Convert eff_dt to date-only if present.
    Normalize prev_store_nbr (replace "Unknown", convert to numeric, cast to nullable Int64).
    Make a copy df_kept_copy and identify dummy rows: first row per patient with status == 'Unknown' and recent_status == 'Unknown'.
    Remove those dummy rows → df_kept_filtered.
    Create final output DataFrame with the selected columns ['entrp_ptnt_id','eff_dt','status','recent_status','transition_dt','prev_store_nbr'].

11. The Resultant Output dataframe is saved as an Excel file ‘dim_ptnt_status.xlsx’ to be used to visualize in the Web Dashboard.
    The processed dataframe is saved as ‘processed_table.xlsx’ format.