

Code Structure

The repository of the project can be found here:

<https://github.com/hakim-l/customer-purchase-forecast.git>

The code repository is structured with the following way:

```
|— LICENSE
|— Makefile          <- Makefile with commands like `make data` or `make train`
|— README.md         <- The top-level README for developers using this project.
|— data
|   |— interim       <- Intermediate data that has been transformed.
|   |— processed      <- The final, canonical data sets for modeling.
|   └─ raw           <- The original, immutable data dump.
|
|— models            <- Trained and serialized models, model predictions, or model summaries
|
|— notebooks         <- Jupyter notebooks. Naming convention is a number (for ordering),
|                       the creator's initials, and a short `~` delimited description, e.g.
|                       `1.0-jqp-initial-data-exploration`.
|
|— references        <- Data dictionaries, manuals, and all other explanatory materials.
|
|— reports           <- Generated analysis as HTML, PDF, LaTeX, etc.
|   └─ figures       <- Generated graphics and figures to be used in reporting
|
|— requirements.txt  <- The requirements file for reproducing the analysis environment, e.g
|                       generated with `pip freeze > requirements.txt`
|
|— deployment        <- Web app deployment with django platform
└─ src              <- Source code for use in this project.
    |— __init__.py   <- Makes src a Python module
    |
    |— data           <- Scripts to download or generate data
    |   |— preprocess_data.py
    |   └─ synthesize_data.py
    |
    |— features       <- Scripts to turn raw data into features for modeling
    |   └─ generate_features.py
    |
    └─ models         <- Scripts to train models and then use trained models to make
        |               predictions
        └─ mlp
            |— data_generator.py
            |— evaluation.py
            |— model.py
            └─ train.py
```

Approach

About

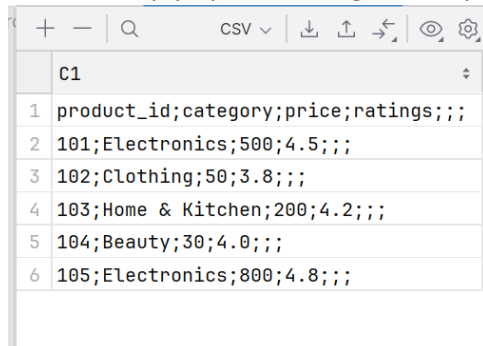
In this project, I try to make a deep learning that can forecast what product will be bought by customer on their next purchase.

Dataset preprocessing

- customer_interactions.csv
- product_details.csv
- purchase_history.csv

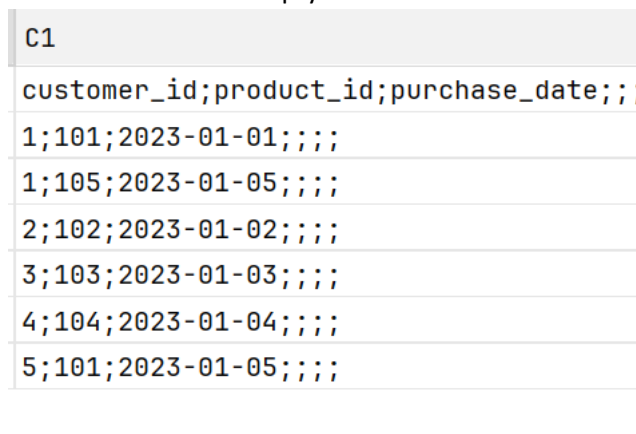
This assignment uses 3 datasets, customer_interactions, product_details, and purchase_history. These datasets have the following issues:

- product_details.csv is not using ',' as its character separator. This is a minor issue since it can be fixed simply by mentioning what separator to be used in pandas library.



| | C1 |
|---|--------------------------------------|
| 1 | product_id;category;price;ratings;;; |
| 2 | 101;Electronics;500;4.5;;; |
| 3 | 102;Clothing;50;3.8;;; |
| 4 | 103;Home & Kitchen;200;4.2;;; |
| 5 | 104;Beauty;30;4.0;;; |
| 6 | 105;Electronics;800;4.8;;; |

- Empty column
There are files with empty columns in them. To fix this I simply deleted those columns.



| | C1 |
|---|---|
| | customer_id;product_id;purchase_date;;; |
| 1 | 1;101;2023-01-01;;; |
| 1 | 1;105;2023-01-05;;; |
| 2 | 2;102;2023-01-02;;; |
| 3 | 3;103;2023-01-03;;; |
| 4 | 4;104;2023-01-04;;; |
| 5 | 5;101;2023-01-05;;; |

- Number of data
There are a limited number of data records.
 - 5 data records on customer_interactions
 - 5 data records on product_details
 - 6 data records on purchase_history

Referring to the assignment description that said I can modify the number of records, I decided to do data synthesis to generate new data.

Dataset Synthesis

The HMASynthesizer is a powerful tool within the Synthetic Data Vault (SDV), designed to create synthetic data by learning from real data and generating high-quality synthetic samples. I choose the algorithm with following reasons.

- The HMASynthesizer utilizes a hierarchical machine learning (ML) algorithm to learn from actual data and produce synthetic data.
- It focuses on modeling both individual tables and the connections between them.
- The algorithm leverages classical statistics to ensure accurate representation.
- Suitability and Capabilities:
 - The HMASynthesizer is well-suited for datasets with the following characteristics:
 - Around 5 tables in the dataset.
 - level of depth, such as a parent table and its child table.

The detailed informations of algorithm can be found here:

[HMASynthesizer - Synthetic Data Vault \(sdv.dev\)](https://sdv.dev/HMASynthesizer)

Feature engineering

To complete the task I did some feature engineering to generate new features.

| Generated variables | Roles | How this computed |
|---------------------|---|--|
| target_array | Target variable | <ul style="list-style-type: none">- Doing operation on purchase_history based on grouping by customer_id:<ul style="list-style-type: none">- Order the records based on order_date- Shift the product_id with -1 step to represent what product will be bought on the next purchase. This process give the target variable- Do One Hot Encoding towards target variable, giving target array of next purchase product_id |
| Beauty | Predictor Represent total bought beauty product by each customer | <ul style="list-style-type: none">- Join purchase_history and product_details- Pivot table of purchase_history (customer_id as index, product_category as columns) |
| Clothing | Predictor Represent total bought clothing product by each customer | <ul style="list-style-type: none">- Join purchase_history and product_details- Pivot table of purchase_history (customer_id as index, product_category as columns) |
| Electronics | Predictor Represent total bought electronic | <ul style="list-style-type: none">- Join purchase_history and product_details |

| Generated variables | Roles | How this computed |
|---------------------|---|--|
| | product by each customer | - Pivot table of purchase_history (customer_id as index, product_category as columns) |
| Home & Kitchen | Predictor Represent total bought Home & Kitchen product by each customer | - Join purchase_history and product_details - Pivot table of purchase_history (customer_id as index, product_category as columns) |
| Price | Average money spent on each purchase | - Pivot table of purchase_history (customer_id as index) |
| ratings | Average ratings of product bought by customer | - Join purchase_history and product_details - Pivot table of purchase_history (customer_id as index) |
| spent_per_view | Average time spent per views per customer | - Using customer_interactions - Dividing time_spent and page_views |

Modelling

Model: Multilayer Perceptron

Number of layers: 3

Consideration in using this method:

- Number of unique value of target variable (product_id) after data synthesis are 500 product. This number is quite much for classical machine learning use.
- Most predictor are numerical. MLP can utilize these variables more than CART algorithm.
- Using MLP make it easier to do incremental learning.

Deployment

- Used Cloud Service:
 - GCP
- Frameworks:
 - Django
 - Plotly
 - Dash
 - Redis
 - nohup
- Database: sqlite