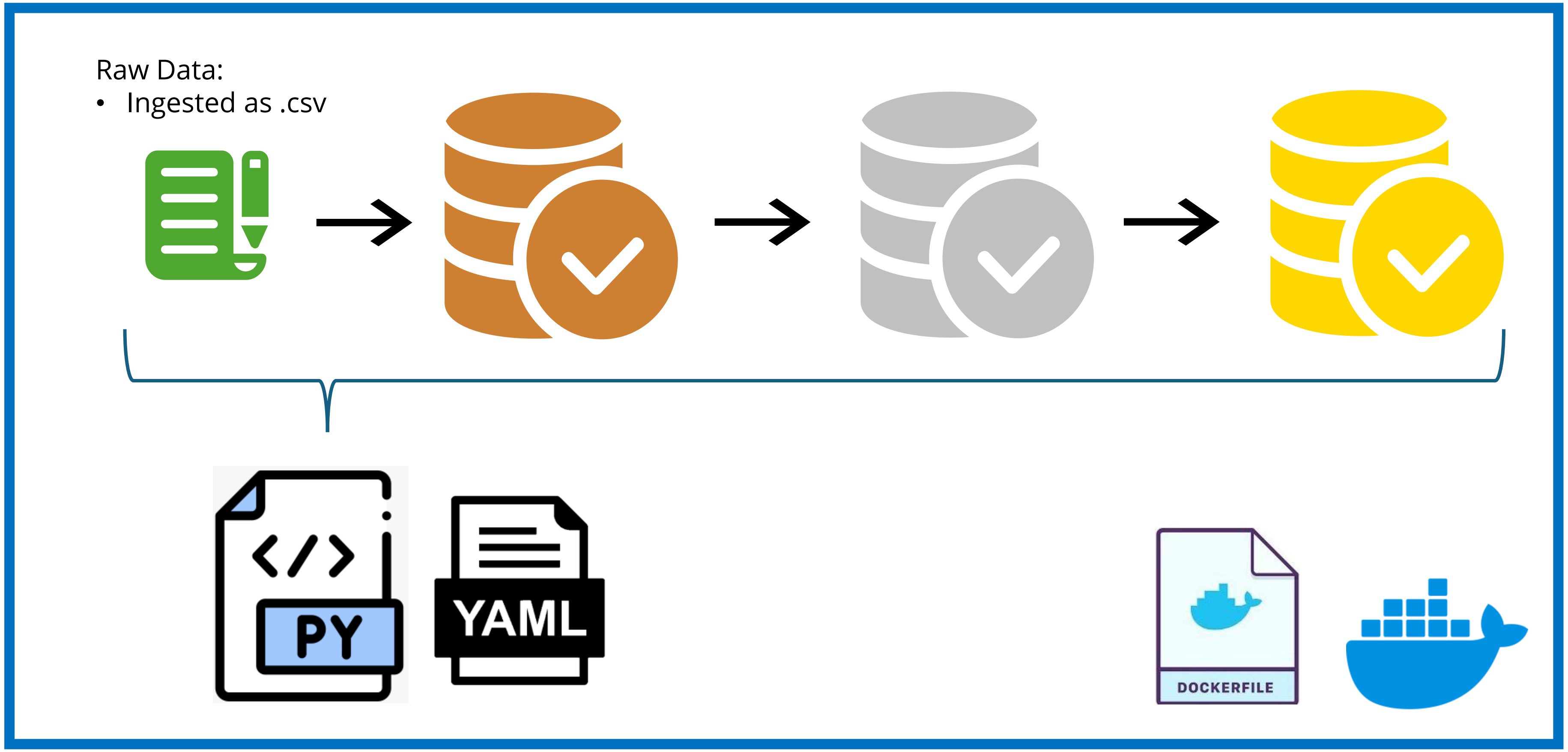
Data Processing Pipelines

CS611 – Machine Learning Engineering
Assignment 1
Yue Jun Yuan

Proposed Pipeline – Medallion Architecture



Single python script orchestrates entire preprocessing pipeline

- Logic transforming from each stage captured in individual python scripts
- Configuration and parameter details managed on separate YAML files
- Pipeline fully containerized to ensure reproducibility and facilitate production deployment
- Logs are created with each run to detail results of data validation and dataset shape details

Bronze:

- Partitioned by month and saved as parquet
- Raw data dump
- No processing

Silver:

- Partitioned by month and saved as parquet
- Data Type enforcement
- Data Quality Handling
- Synthesize additional columns where relevant

Gold:

- Separate feature and label store
- Each store subdivided into train / val / test / OOT
- Identifiers retained (to be dropped during ML pipeline)
- Missing value handling and encoding rules derived from train and applied to other splits

Overall Pipelining Considerations

Storing Configuration / Parameters in YAML

- Separation of concerns – separation of core pipeline logic from config and parameters
  - Update of parameters follows different (simpler) approval process as compared to code change
  - Cleaner and easier to read code logic

- Scalability – different YAML files can be used for development / production
  - Easy switching of file paths, or API end points without modifying code

- Version control – changes to parameters can be archived and versioned
  - Avoids multiple versions of scripts or commented code

- User friendly – allows non-programmers to easily understand and edit
  - Business users or new project members can easily make simple changes
  - Reduced hardcoding which results in more maintainable and less error-prone codes

Pipeline logging system

- Pipeline logging captures script execution, completions and errors in dedicated log files
  - New time-stamped log file created with each run of pipeline, enabling traceability and easier debugging

Silver Tables Processing

Silver Tables Processing Steps:

1. Handle **Data Quality issues** (detailed missing and invalid handling available in following slides – treatment to be agreed with business)

2. Create or transform additional columns to **improve data usability**
   - ❖ Create integer based “Credit_History_Age_Months” by transforming text description in “Credit_History_Age”
   - ❖ Create “Type_of_Loan_list” of Array(String) type by parsing “Type_of_Loan”
   - ❖ Full list available in following slides

3. **Data type enforcement** – full list of variables with expected data types defined in **silver_config.yaml** and enforced during processing
   - ❖ Parquet format preserves data type through defined schema

Objective:

Silver tables (separate for each dataset) is processed only **minimally** to allow for general view and analytics consumption. Use case specific transformation are done in gold layer.

Silver Tables Processing – Data Quality Handling

Dataset	Data Quality Issue	Handling
features_attributes.csv	"SSN" contain values "#F%$D@*&8" not following convention	Replace with np.nan
features_attributes.csv	"Occupation" column contain invalid "_______" values	Replace with np.nan
features_attributes.csv	"Age" containing numeric values suffixed with '_' "Age" column contains large invalid values "Age" column containing negative values	Remove '_' suffix Age<0 or Age>120 replace with np.nan; set "Age_valid' binary flag to 0 else 1
features_financials.csv	"Annual_Income" containing numeric values suffixed with '_' "Annual_Income" contain inconsistent rounding	Remove '_' suffix Standardize rounding to 2d.p.
features_financials.csv	"Num_Bank_Accounts" containing invalid negative values "Num_Bank_Accounts" containing large invalid values	Replace with np.nan Cap values at 11*
features_financials.csv	"Num_Credit_Card" containing invalid large values	Cap values at 11*
features_financials.csv	"Interest_Rate" containing invalid large values	Cap values at 34*
features_financials.csv	**"Type_of_Loan"** contains multiple categorical values, including "Not Specified", and missing values	**Create "Type_of_Loan_list" as array of string representation**
features_financials.csv	"Num_of_Loan" contain invalid numeric values suffixed with '_' "Num_of_Loan" contain negative values "Num_of_Loan" contain invalid large values	**Derive values for "Num_of_Loan" using "Type_of_Loan_list"**

* Invalid values handling derived based on data exploration, business to confirm correct maximum values

Silver Tables Processing – Data Quality Handling

Dataset	Data Quality Issue	Handling
features_financials.csv	“Num_of_Delayed_Payment” contain numeric suffixed with ‘_’ “Num_of_Delayed_Payment” contain invalid large values “Num_of_Delayed_Payment” contain negative values	Remove ‘_’ suffix Cap values at 28* No handling business to advice
features_financials.csv	“Changed_Credit_Limit” contain invalid values ‘_’ “Changed_Credit_Limit” contain inconsistent rounding “Changed_Credit_Limit” contain negative values	Replace with np.nan Standardize rounding to 2d.p. No handling business to advice
features_financials.csv	“Outstanding_Debt” containing numeric values suffixed with ‘_’	Remove ‘_’ suffix
features_financials.csv	“Credit_History_Age” in format of “X Years and Y Months”	Convert to int no. of months
features_financials.csv	“Payment_of_Min_Amount” contains {Yes, No, NM}	Business to advice ‘NM’ validity
features_financials.csv	“Amount_invested_monthly” containing invalid ‘__10000__’ values	Remove both prefix and suffix
features_financials.csv	“Payment_Behaviour” containing invalid ‘!@9#%8’ values	Replace with np.nan
features_financials.csv	“Monthly_Balance” contain invalid ‘__-333333333333333333333333333__’ value	Replace with np.nan
features_financials.csv	“Delay_from_due_date” contains negative values	No handling business to advice
features_financials.csv	“Credit_Mix” contain invalid “_” values	Replace with np.nan

* Invalid values handling derived based on data exploration, business to confirm correct maximum values

Gold Tables Processing – Pipelining Considerations

1. The set of concatenated values (**Customer_ID + snapshot_date**) is equivalent between feature_financials and features_attributes and also equivalent with **(Customer_ID + loan_start_date**) on lms_loan_daily
   - ❖ **Align feature store (financials / attributes) to label store (lms_loan_daily)** using above join keys

```
set1 = set(financials_df['Customer_ID_Snapshot'])
set2 = set(attributes_df['Customer_ID_Snapshot'])
set3 = set(lms_loan_daily_df['Customer_ID_start_date'])
set1 == set2 == set3
```

```
True
```

2. Clickstream data is missing **3,526 (approximately 28% of)** customers
   - ❖ **Option 1**: **Drop records** from feature and label stores where Customer_ID are not found in the set of Customer_ID in clickstream data
     - ➢ **Impact**: Lose **more than 1 quarter** of dataset, significant reduction in training data impact model ability to generalize ✗
   - ❖ **Option 2**: Attempt to **extrapolate** clickstream data for missing customers
     - ➢ **Impact**: Introduce **noise** and bias, **limited information** on nature of features (fe1-fe20) preventing accurate extrapolation ✗
   - ❖ **Option 3**: Assume clickstream data to reflect some form of **digital activity (to verify with business)** – therefore lack of data reflects absence of digital activity → **impute 0**
     - ➢ **Impact**: **Absence of activity is a valid parameter** for model learning, avoid injecting false assumptions, simple to implement ✓

3. **Backward looking aggregation** of temporal data (features_clickstream) results in **misalignment of datapoints** available – lack of datapoints to compute **LM / L3M** aggregates for loans incepted in earlier months of dataset (2023-01-01to 2023-03-01)
   - ❖ **Option 1:** Introduce **"months_available"** flag while truncating aggregated data for loans incepted on or before 2023-03-01
     - ❖ **Impact**: Retains full dataset but may **introduces more complexities than value**, model **may require even more datapoints** to learn nuanced differences between how such a flag interacts differently with SUM and AVG aggregates ✗
   - ❖ **Option 2**: **Drop records** pertaining to loans incepted **on or before 2023-03-01**
     - ➢ **Impact**: Lose **1,537 (12.3%)** of customers available, **prevents increase of feature space**, more practical approach considering 'missing' datapoints are **more likely due to data sourcing limitations** rather than data quality issues ✓

Gold Tables Processing

Gold Tables Processing Steps:

1. Filter away loans on or before 2023-03-1 (**prevent truncation of aggregated features**)

2. Create Label Store – Label definition: default_flag=1 if customer has any overdue_amt > 0 over the life of the loan
   - **No label leakage** as every Customer_ID only has 1 loan, and any overdue_amt relates to future outcome relative to loan_start_date

3. Create aggregated features (AVG_L3M / SUM_L3M / LM) from temporal clickstream data
   - Aggregation done for each customer using **snapshot_date immediately before loan_start_date**.

Loan_start_date	LM	L3M
2025-01-01	2024-12-01	2024-10-01 to 2024-12-01

4. Create Feature Store – "Customer_ID" and "loan_start_date" from **Label store used as anchor keys**
   - Left join static features (attributes and financials) followed by aggregated clickstream features
   - Ensures no data leakage – static features joined on static_feats.snapshot_date, temporal features joined on click_aggs.loan_start_date (already aggregated per logic above)

5. Perform date-based splits on both feature and label stores using – **gold_config.yaml defines split windows**

6. Model specific handling – apply missing value treatment and encoding based on **train set data or gold_config.yaml**
   - Details on encoding strategy and missing value handling can be found in following slides

Objective:

Gold tables are split into Feature and Label Stores, each store further split into train / val / test / OOT sets. Gold layer built to be **ML ready** (identifiers to be dropped during ML pipeline) and designed based on requirements for a tree based model – (no scaling applied). Encoding format is designed to be compatible with most scikit-learn models rather than SparkML models.

Gold Tables Processing – Pipelining Considerations

Encoding Decisioning

Create and maintain a **gold_config.yaml** file for features which are expected to contain only **business approved values**

- Processing pipeline verifies that columns like "Types_of_Loan" contain only approved list of values
  - ➢ Adopt **lenient policy –** log warning and ignore unapproved values
  - ➢ Ensures **stability of gold table schema** (after multi-hot-encoding)
  - ➢ **Early detection** of data drift or data quality issues if unexpected values appear

- Multi-hot encoding relies on expected set of values from gold_config.yaml to apply encoding to all splits (train / validation / test / OOT)

loan_type
Auto Loan
Credit-Builder Loan
Debt Consolidation Loan
Home Equity Loan
Mortgage Loan
Not Specified
Payday Loan
Personal Loan
Student Loan

Encoding pipeline for other categorical variables **fits only on train dataset** and **transforms other splits** (test / validation / OOT) using the train fitted pipeline

➢ **Guarantees no data leakage** from unseen datasets into training

➢ All splits result in the same number of columns

Gold Tables Processing – ML specific handling

Feature	Data Type	Missing - Reason	Fill / Impute Strategy	Missing Frequency	
				Number	(%)
Age	Integer	Unrealistic values (negative or values above reasonable cap 120) replaced with NaN on silver tables	Median	104	0.83%
Occupation	String	“_______” values were replace with NaN on silver tables	“Unknown” Flag	880	7.04%
Num_Bank_Accounts	Integer	Negative values treated as invalid and replace with NaN on silver tables	Median	4	0.03%
Type_of_Loan_list	Array(String)	Missing values indicate no loans (Num_of_Loans == 0)	“No_Loan” flag	1,426	11.41%
Changed_Credit_Limit	Float	Invalid “_” values were replaced with NaN on silver tables	Impute 0	254	2.03%
Credit_Mix	String	Invalid “_” values were replaced with NaN on silver tables	“Unknown” flag	2,611	20.89%
Payment_Behvaiour	String	Invalid ‘!@9#%8’ values were replaced with NaN on silver tables	“Unknown” flag	998	7.98%
Monthly_Balance	Float	Invalid ‘__-333333333333333333333333333__’ values were replaced with NaN on silver tables	Impute 0	1	0.01%
fe_1 – fe_20	Float	Absence of digital activity / profile – to confirm with business	Impute 0	3,526	28.21%

Split	Period	Duration	Purpose
Train	2023-04 → 2024-06	15 months	Used to fit models and transformations (StringIndexer, OneHotEncoder, etc.)
Validation	2024-07 → 2024-09	3 months	Used for hyperparameter tuning / early stopping
Test	2024-10 → 2024-12	3 months	Final performance evaluation before deployment
OOT (Out of Time)	2025-01	1 month	Simulates unseen future data for production generalization check