Bank Customer Segmentation

Vinu Karthek
29 March 2024

Problem Statement

- In the competitive banking industry, understanding the diverse customer base & their financial behaviors is critical for providing tailored services and improving customer satisfaction.
- However, with the vast amounts of customer data generated daily, manual segmentation is not feasible

Objective

- The objective is to use machine learning to segment customers to Affluent & Normal
- This can be done based on their relevant banking data attributes such as account balances, transaction volumes, sources of funds, investment portfolios, credit ratings, and more

Motivation & Business Relevance

The success full classification of the customer segments allows for,

- Personalized Marketing
- Improved Customer Service
- Product Development

Agenda

Data Analysis

- Exploratory Analysis
- Statistical Analysis
- Data Cleaning
- Feature Engineering

Model Development

- Model Architecture
  - Decision Tree
  - Random Forest/xGBoost
  - RUSBoost
- Data Flow
- Performance
- Business Summary

Deployment

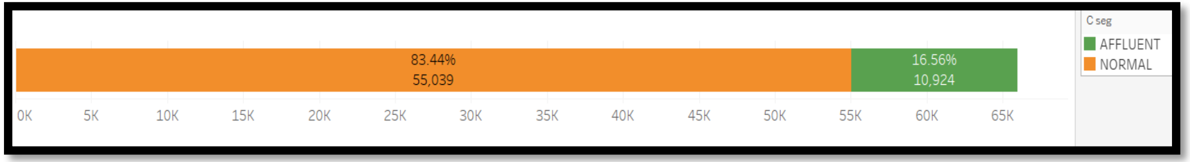
- Solution Architecture
- Serve Model

Data Analysis

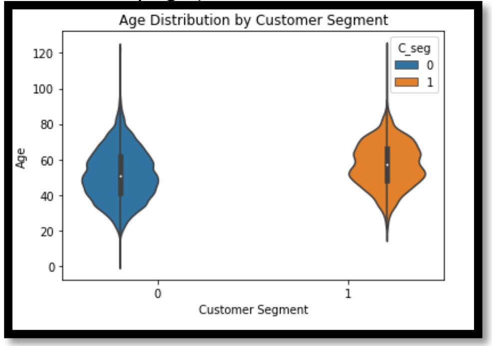
NOTE: Plotted using Tableau & some using Seaborn

Data Demographics

Normal/Affluent Ratio → ~4:1

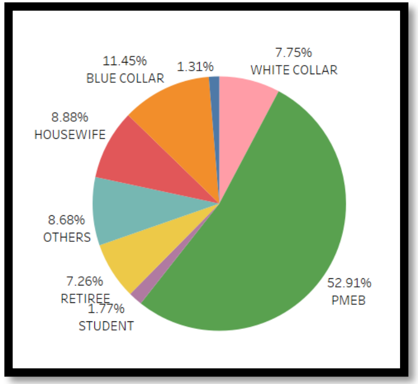


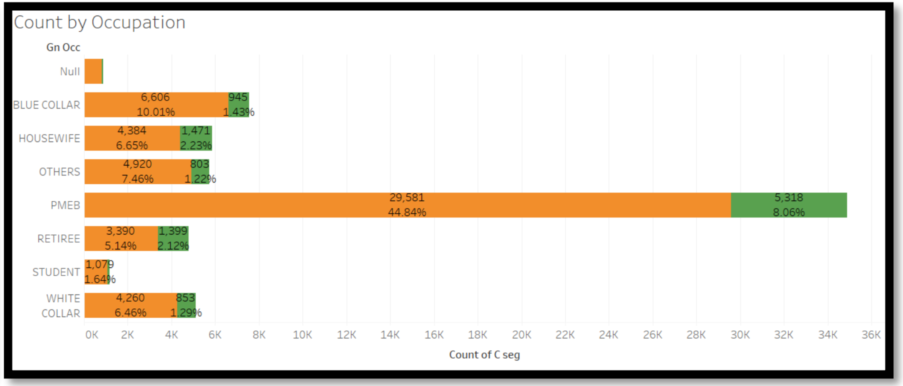
Distribution by Age (Affluent Median > Normal Median)



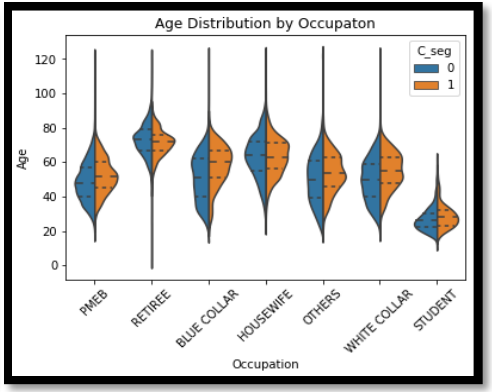
Normal/Affluent Ratio by Occupation

- The Normal/Affluent Ratio across occupation is ~15%, while the majority of occupation is PMEB
- And outliers Age >100 exists for all occupation (which can be removed as they are > 3 Sigma)

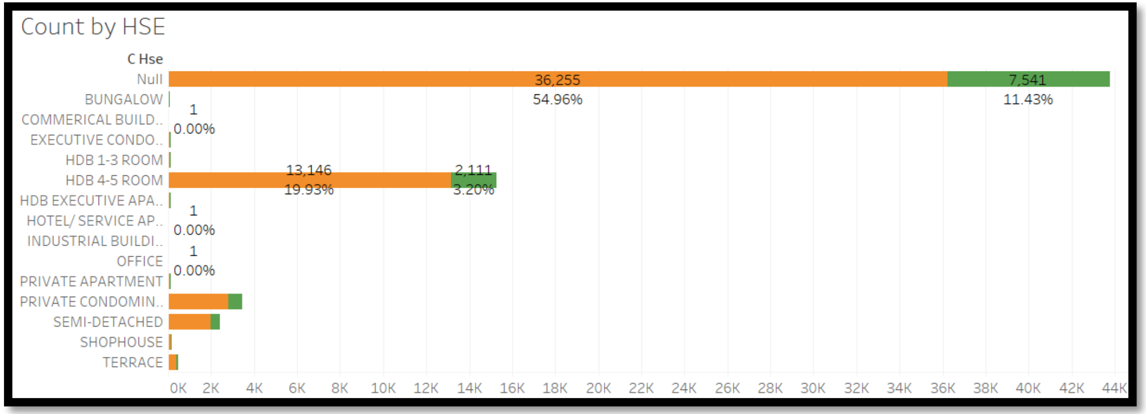


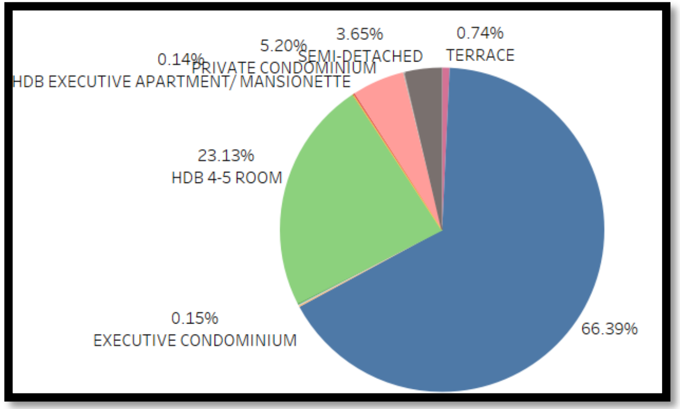


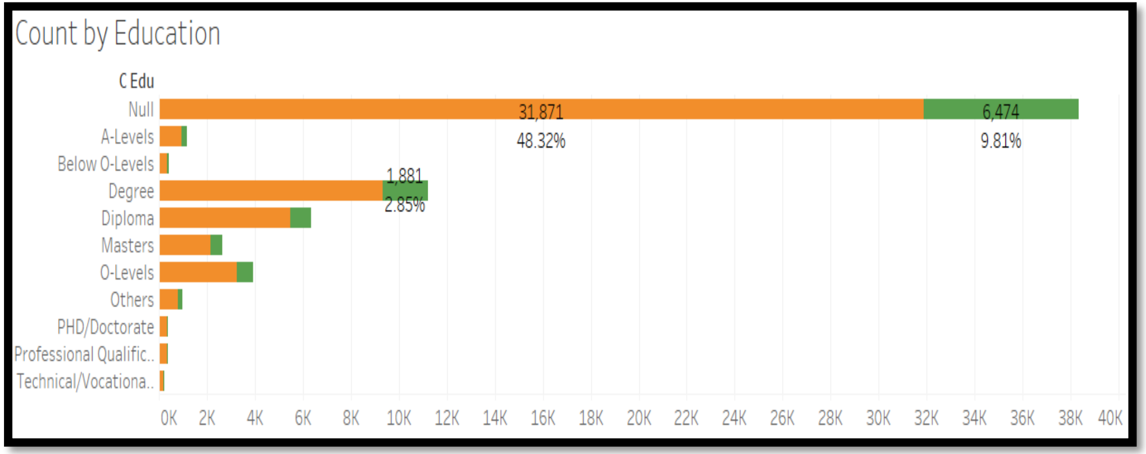
Distribution by Age

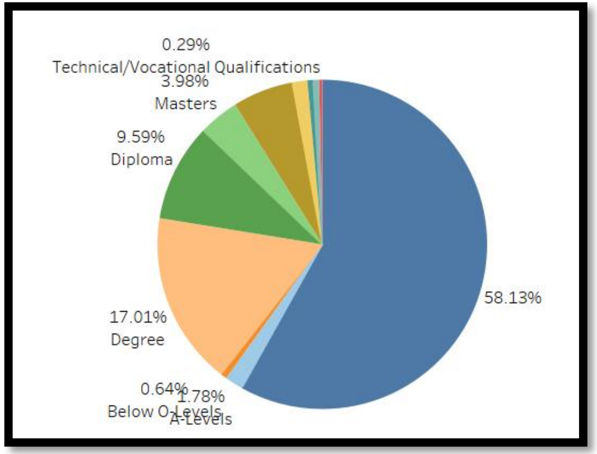


Data Demographics



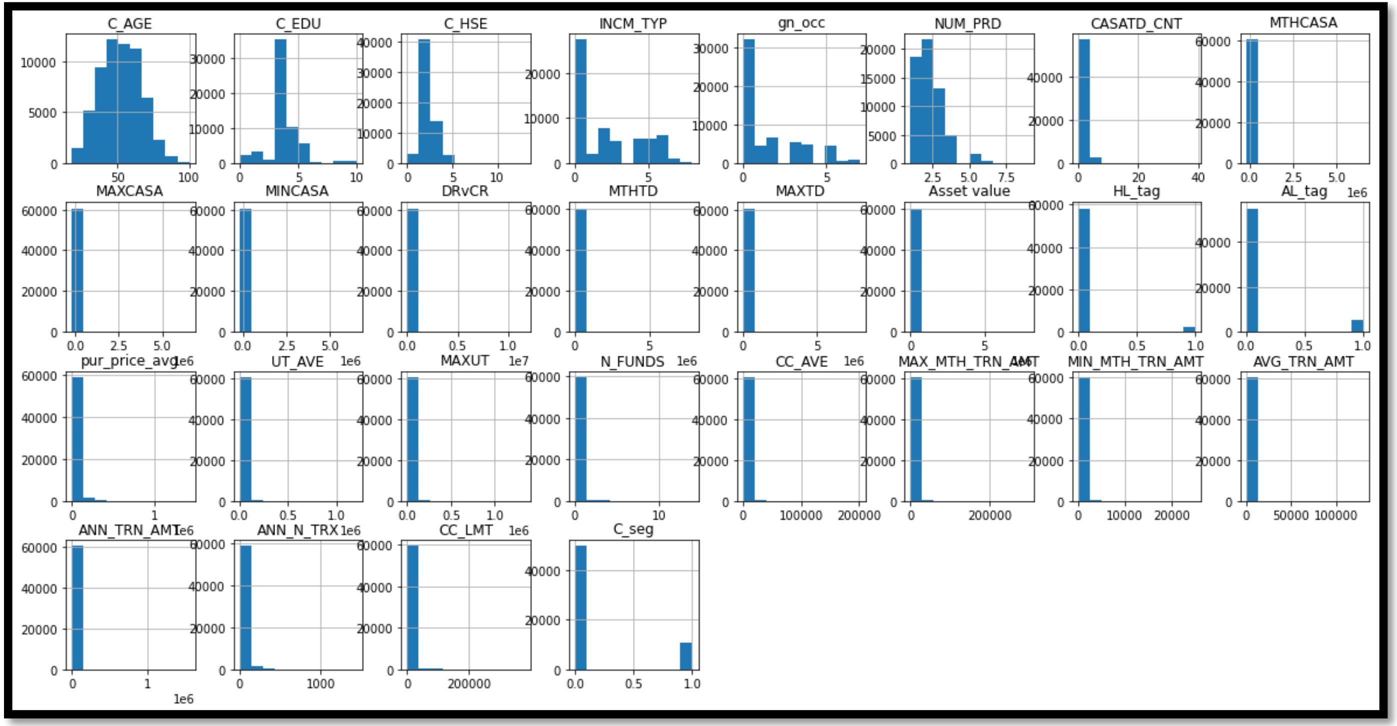






Data Distribution (Histogram)

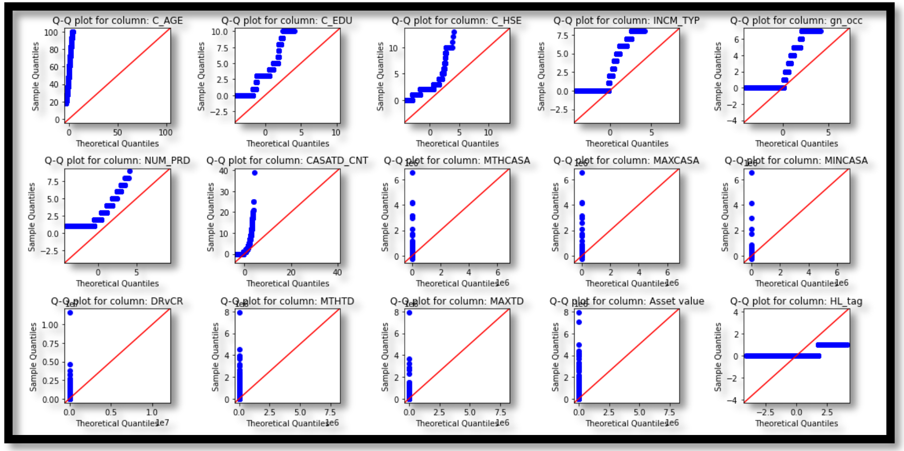
- Every numerical features except for the 1st six in the plot below are heavily skewed as show by the histogram
- This could mean that most of the other numerical features are empty for majority of the rows

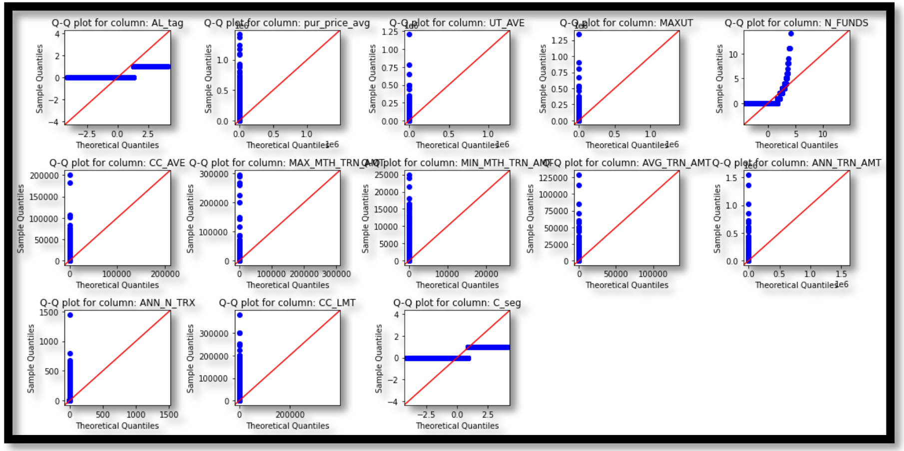


NOTE: All the categorical features are factorized to numbers

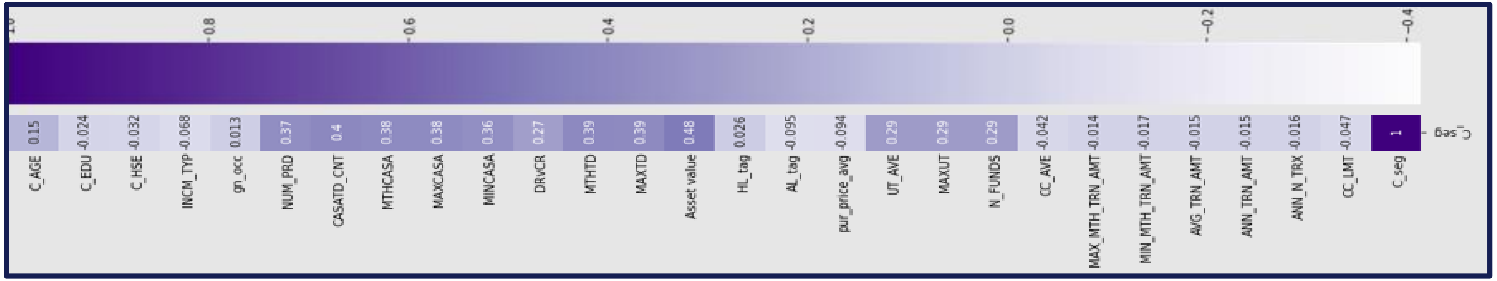
Statistical Analysis

- **Q-Q Plot** shows that none of the features have Normal Distribution
- Pearson's Linear Correlation cannot be performed





- **Spearman's Correlation** shows very low correlation for the features with unscrewed histogram (from prev. slide)
- **Kendal Tau's** correlation (less sensitive to outliers) also yields similar results



Data Cleaning & Engineering

Cleaning

- As discussed earlier Ages >100 were removed to remove outliers
- Filled empty values with 0 (for numerical) & NA (for categorical)
- Removed ID (as its irrelevant for modelling)

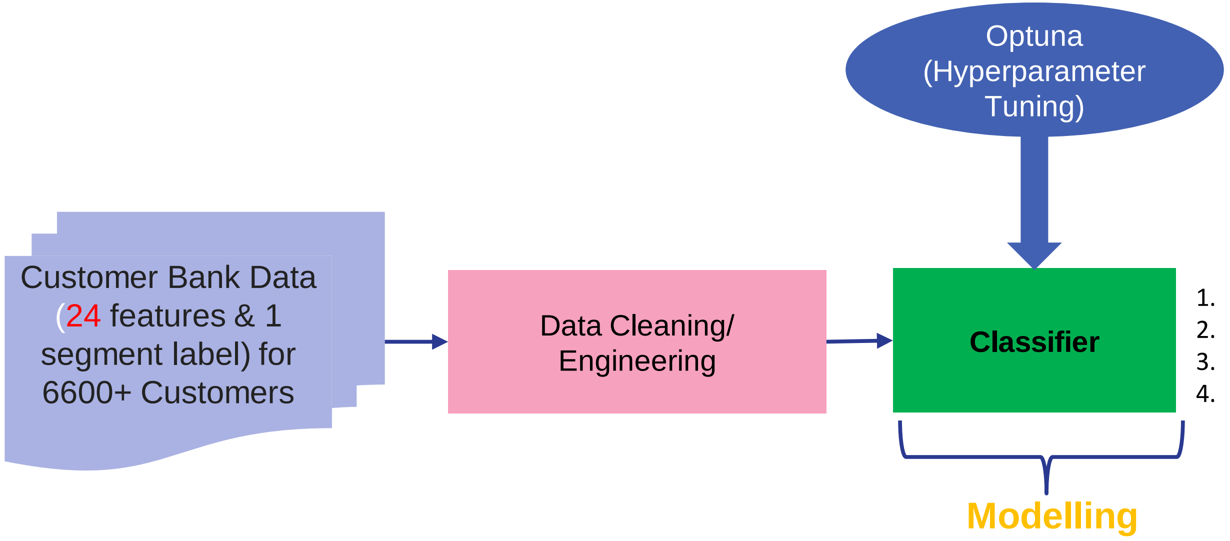
Feature Engineering

- Added two now columns Has_TD & Has_CC based on the existing categorical columns
- If CC/TD numerical Columns are empty, then Has_TD & Has_CC are 0 respectively
- Convert Categorical to numeric by factorizing them

Model Development

Model Architecture

- The following models where chosen since the data is tabular & has numerical/categorical features
- Once we attain the optimum model we can go with the hyper parameter tuning (for optimized performance)

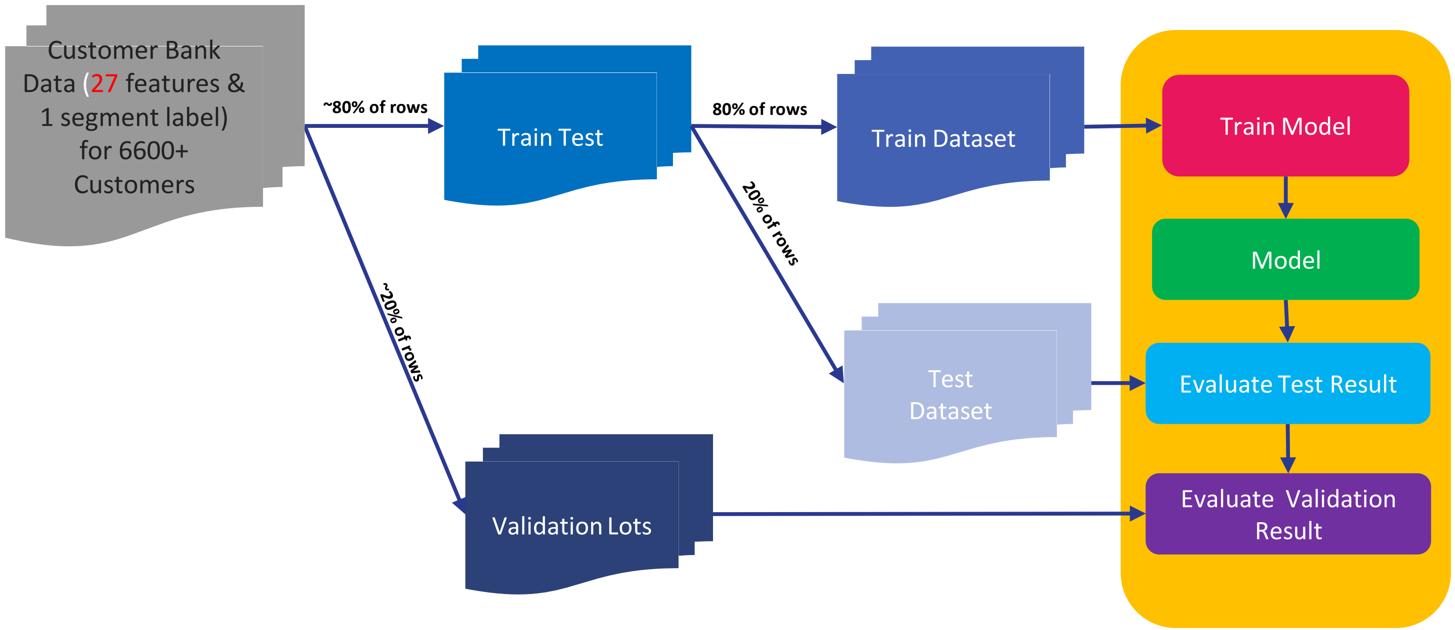


1. Decision Tree
2. Random Forest
3. xGBoost (Gradient Boosting)
4. RUSBoost (Random Under Sample Boosting)

NOTE: The same architecture to be used for several models

Model Development (Data flow)

- Data flow during model training process, the split ratio can be 20% or 30% depending on the dataset
- Once the model is trained on train dataset & evaluated on test dataset, a k-fold cross validation is done to check the robustness of the model



NOTE: The ratio of Normal to Affluent was maintained across all splits (Stratify = True)

Business Analysis Definition

	Predicted Normal (0’)	Predicted Affluent (1’)
Normal (0)	9119 (TN)	1889 (FP)
Affluent (1)	1307 (FN)	1756 (TP)

Precision = $\frac{TP}{TP+FP} = \frac{1756}{1756+1889}$ = 0.48

- Overkill Rate → The percentage of normal predicted as affluent(~50% in this example)

Recall = $\frac{TP}{TP+FN} = \frac{1756}{1756+1307}$ = 0.83

- Escapee Rate→ The percentage of affluent customer that is predicted correctly.

Model Performance & Business Summary

- From the performance chart we can see that all models have **very low precision for Class = 'Affluent'**. This could mean any of the following

Class Imbalance:

- **Cause**: If class 0 instances significantly outnumber class 1 instances, machine learning algorithms may become biased towards the majority class and may not perform well on the minority class.
- **Observation**: In this situation, **under sampling & class weighting was tried and the performance could not be improved**

Noise & Outliers

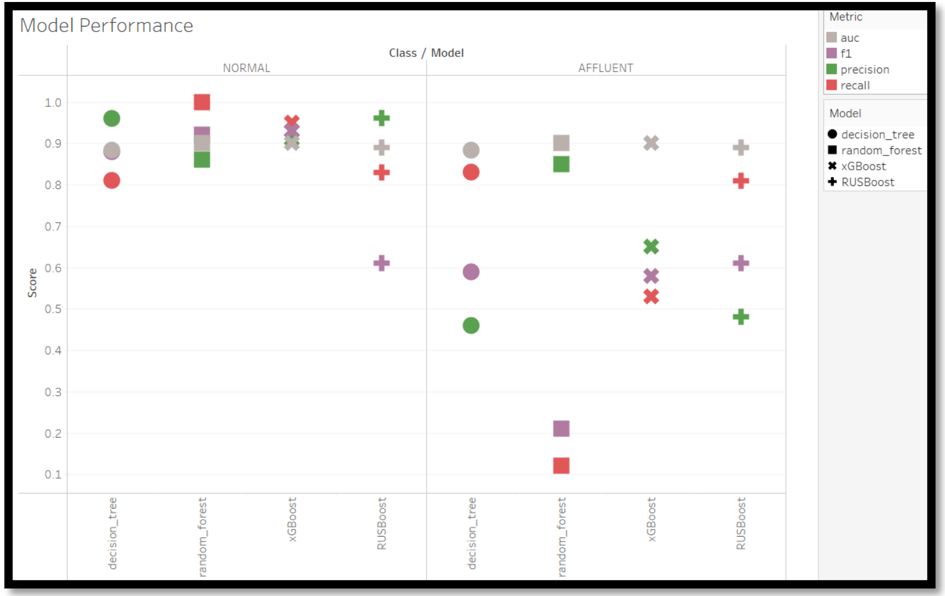
- **Cause**: Noise & outliers in the class 1 instances can also lead to poor precision as the model struggles to correctly classify these instances
- **Observation**: In this case, removing the outlier did not improve the model performance significantly
- **Suggestion**: Understand the False Negative cases (Affluent predicted as Normal), to understand what is causing the noise

Feature Selection/Engineering:

- **Cause**: The features used to train the model might not include enough information to correctly classify class 1 instances.
- **Suggestion**: With the help of Domain Expert, explore the data further & see if new features can be engineered that could help improve the classification performance.

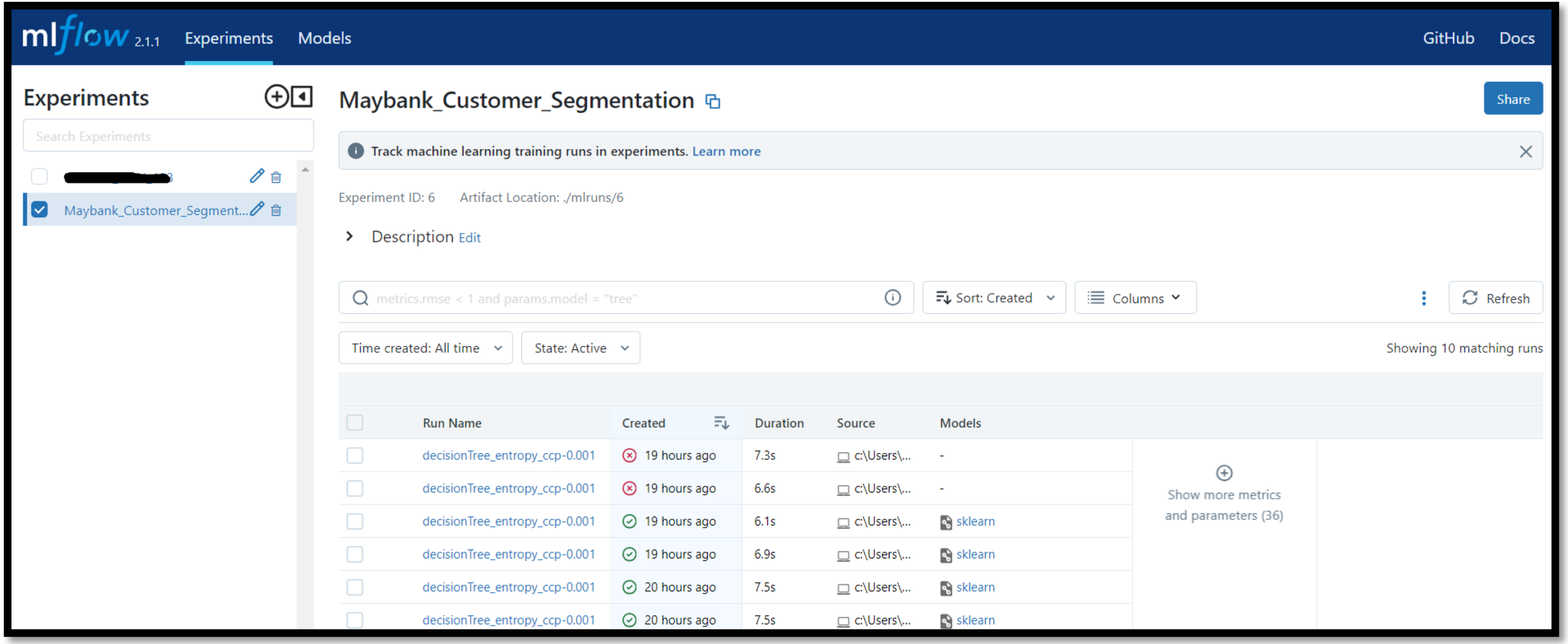
Prediction Threshold:

- **Observation**: All the analysis was done with threshold 50%, changing this might slight improve the precision at the cost of recall
- **Suggestion**: With the help of Domain Expert, determine the optimum threshold for which a prediction can be classified as affluent



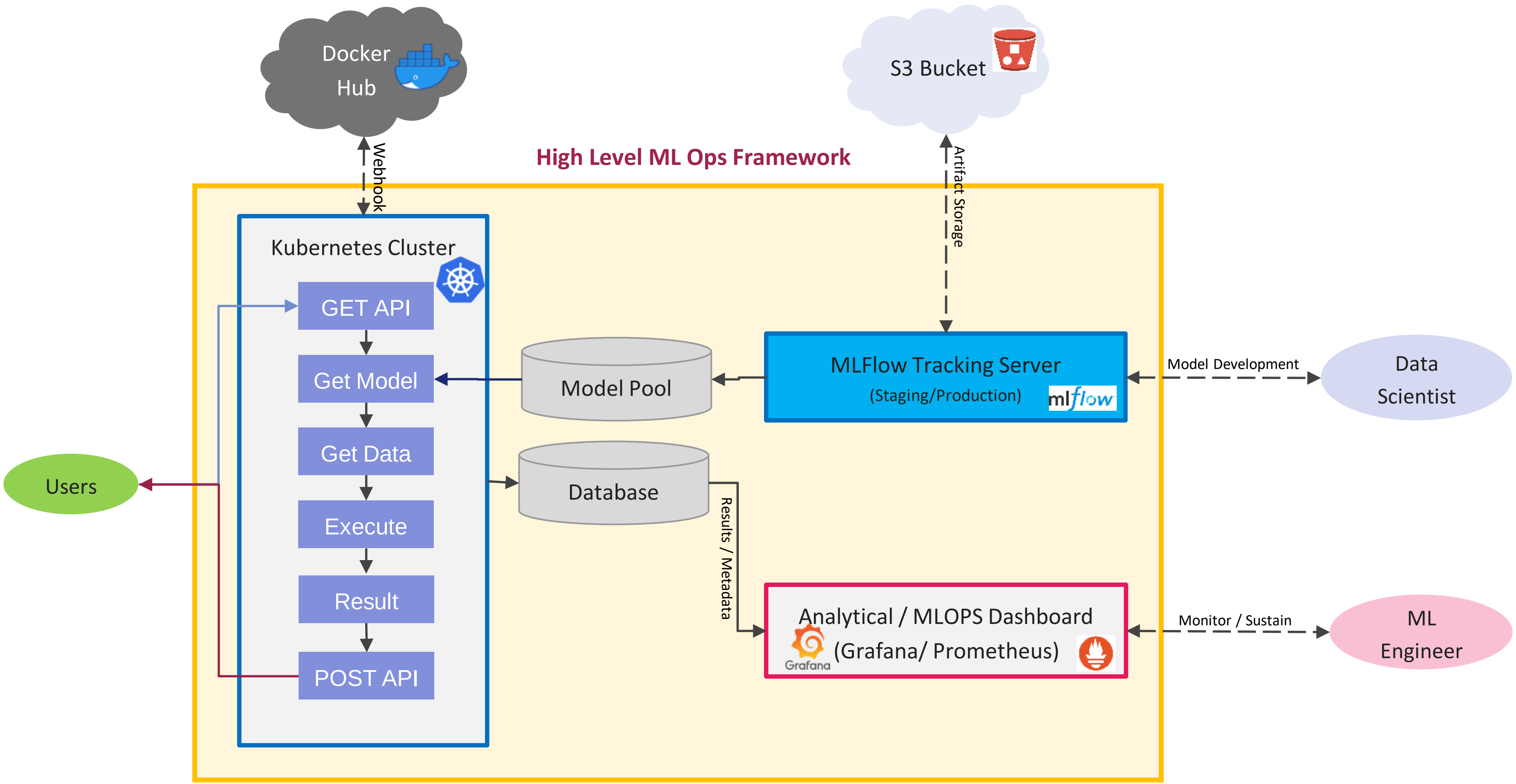
MLFlow Tracking Server

- Experiments & Runs logged on MLFlow use case using Decision Tree Classifier



Deployment

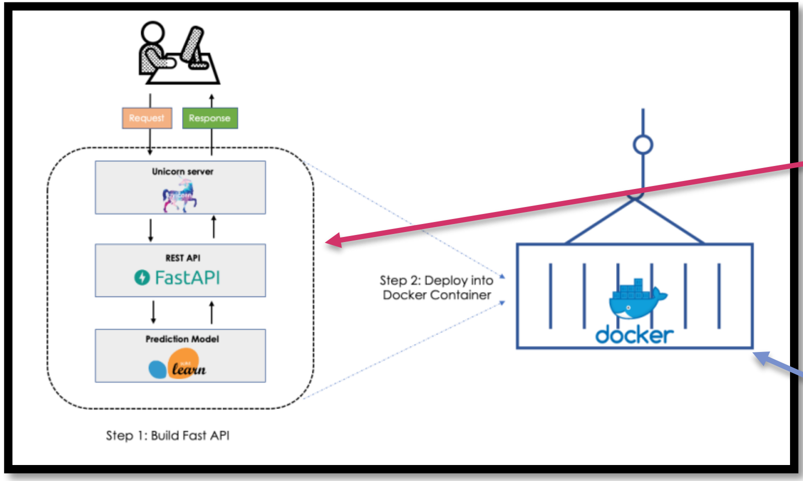
Solution Architecture



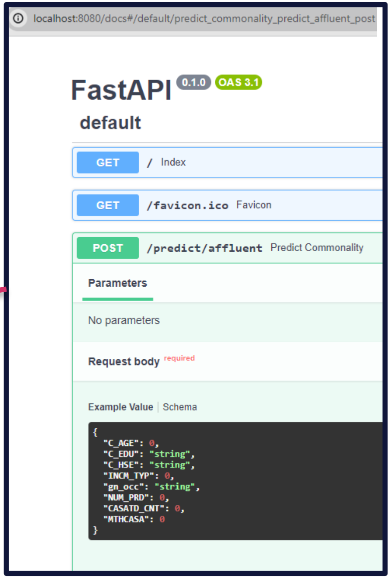
NOTE: Refer Slide Notes (V)

Model Serving (Fast API)

- The Model Can be served using several methods like Seldon Core/Bento ML/Streamlit/Django/FastAPI. For this use case FastAPI was chosen
- The Code can be containerized in to a docker image & uploaded to a docker hub
- This can then be used in any Kubernetes Cluster or just as a container in any Docker Engine



STEP1



STEP2

git > python > projects > Machine-Learning > bank_affluent_classif > deployement > Dockerfile > ...

```dockerfile
# Use Python 3.9 base image
FROM python:3.9

# Set working directory in the container
WORKDIR /app

# Copy requirements file and install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy your Streamlit Python file
COPY main.py .

# Define an environment variable for the start command (default is streamlit command)
ENV SERVER_PORT = "8080"
ENV START_COMMAND="uvicorn main:app --host localhost --port $SERVER_PORT --reload"

# Set the command to start the Streamlit app using the environment variable
CMD ["/bin/sh", "-c", "$START_COMMAND"]
```

Thanks for the Opportunity ☺