Midterm Presentation - Click Through Rate

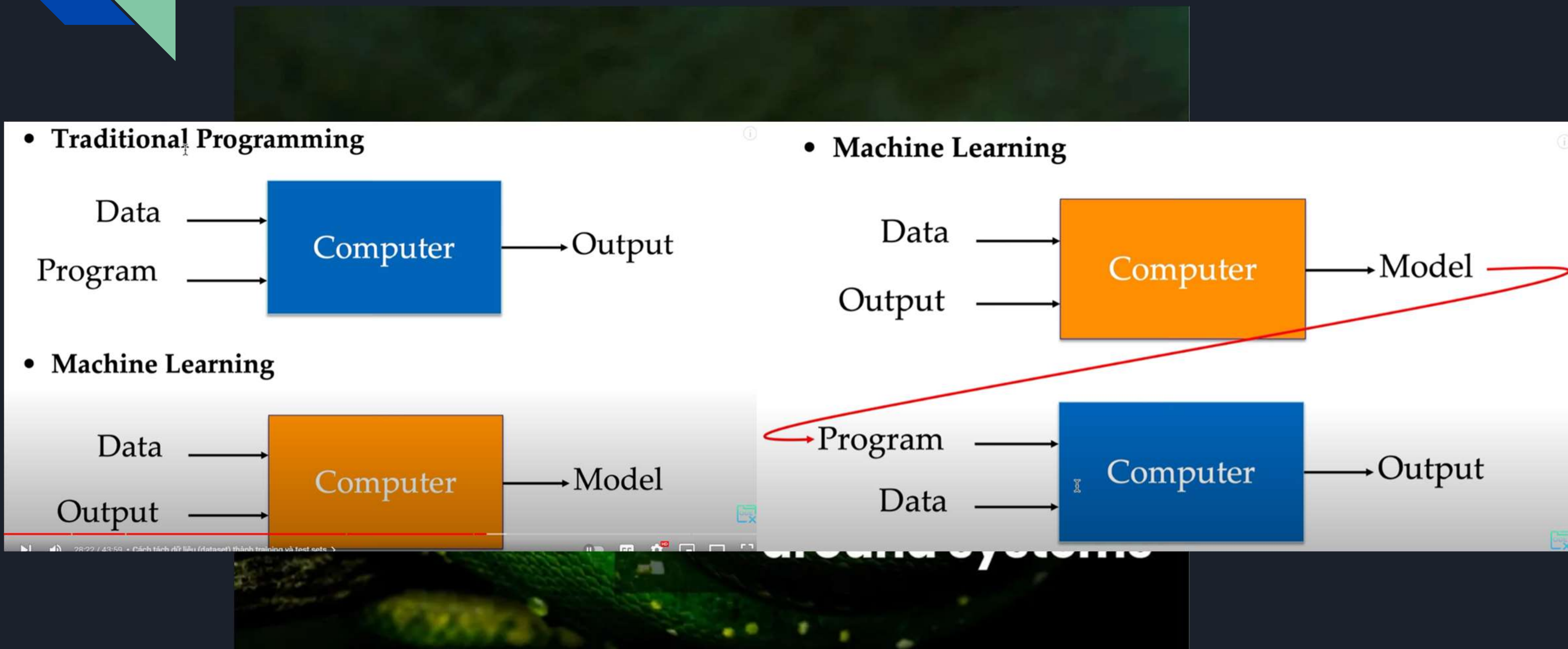
Vivian Kuang
Malik Aqib Rehman
Dzung Do

July 14th, 2023

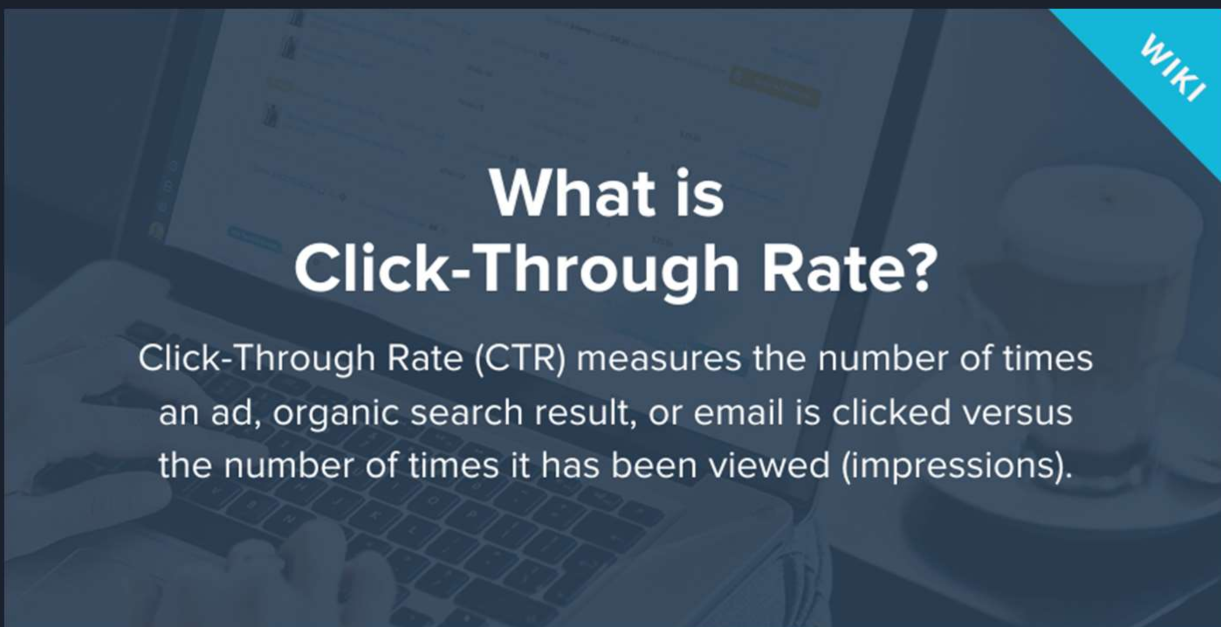
Content

1. Introduction
2. Objective
3. Data
4. Modelling
5. Challenges
6. Conclusion
7. Next steps

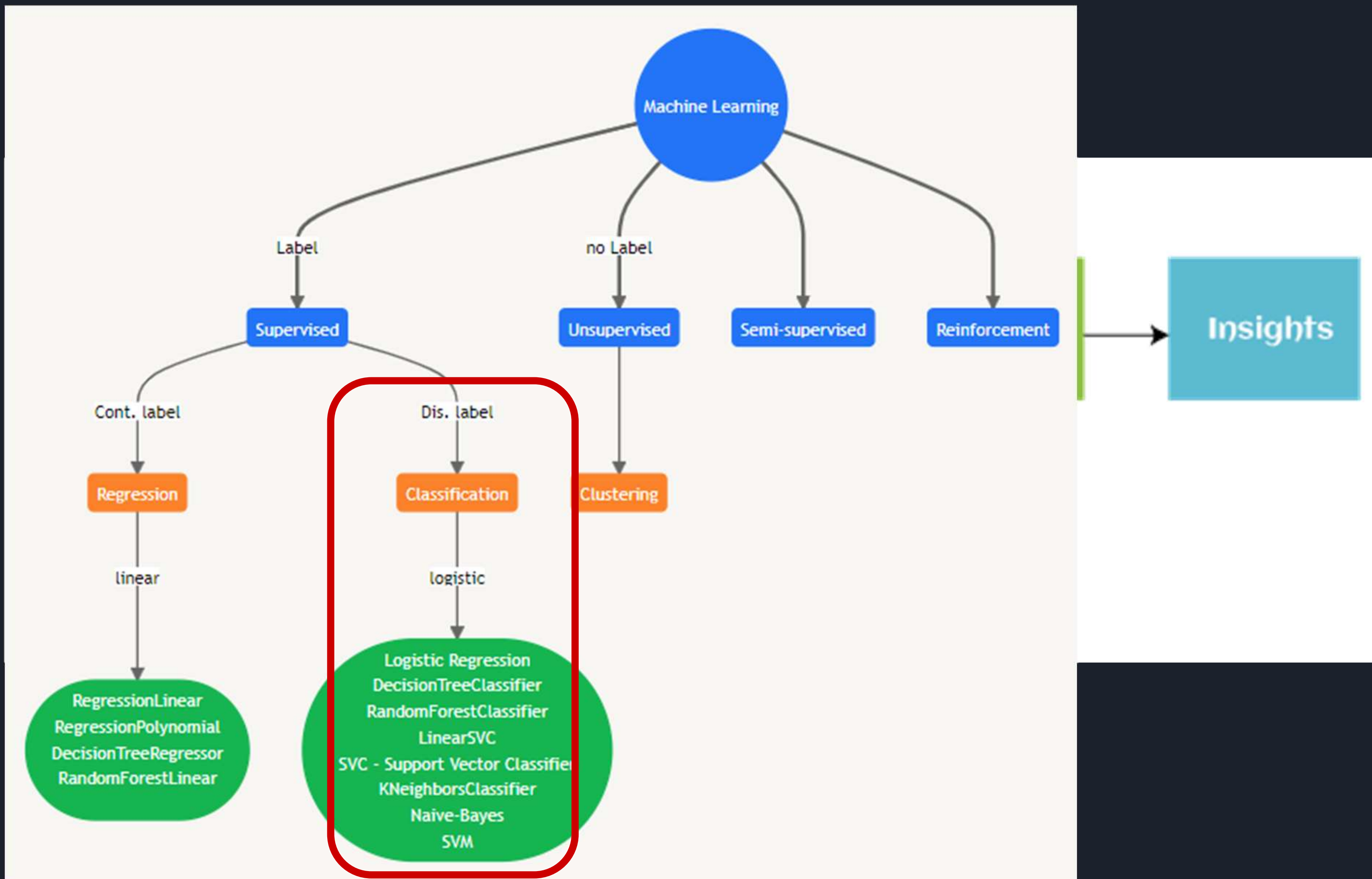
1 Introduction



2 Objective - clicked or not



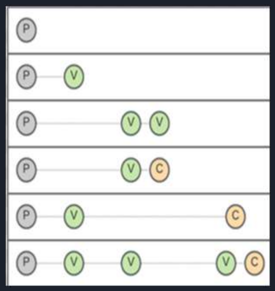
$$CTR = \frac{\text{Clicks}}{\text{Impressions}} \times 100$$


Machine Learning
Label
no Label
Supervised
Unsupervised
Semi-supervised
Reinforcement
Cont. label
Dis. label
Regression
Classification
Clustering
linear
logistic
RegressionLinear
RegressionPolynomial
DecisionTreeRegressor
RandomForestLinear
Logistic Regression
DecisionTreeClassifier
RandomForestClassifier
LinearSVC
SVC - Support Vector Classifie
KNeighborsClassifier
Naive-Bayes
SVM
Insights

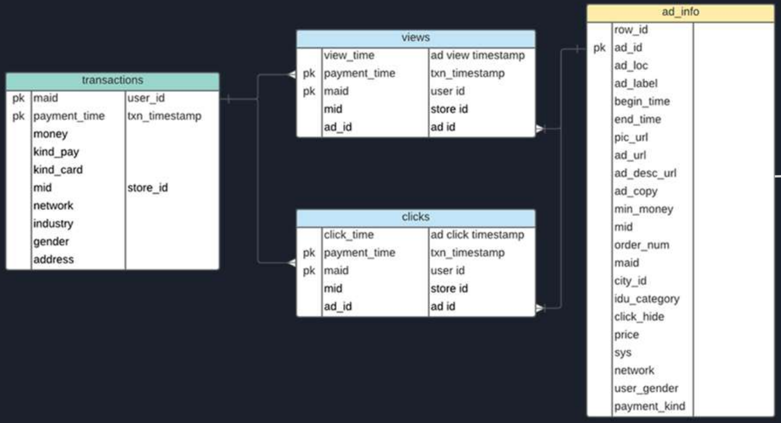
3 Data - Analyzing Data Files & Understanding data entities

- Analyze Datasets
  - Data Files: bulk data with imbalanced data
    - About 8G in total,
    - Tens of millions of transactions, millions of views but a couple of millions of clicks
    - High duplicated clicks
  - transactions, views, clicks, ad_info combined to => analysis_data

Scenarios for transaction-review-click relations



Data Model - Mobile Ad CTR Database



Final table: analysis_data
Transactions inner join on views, left join on clicks and adinfo

3 Data - Loading data & Insights of the Data

- Label: clicked VS Dependant Variables: more than 20 features,
  - No PK: load fast but need to process duplicate rows
  - PK: take time to process the bulk data, (split files to make it work)
  - Consider other choices to improve loading efficiency:
    - ignoring duplicates;
    - remove duplicates before loading

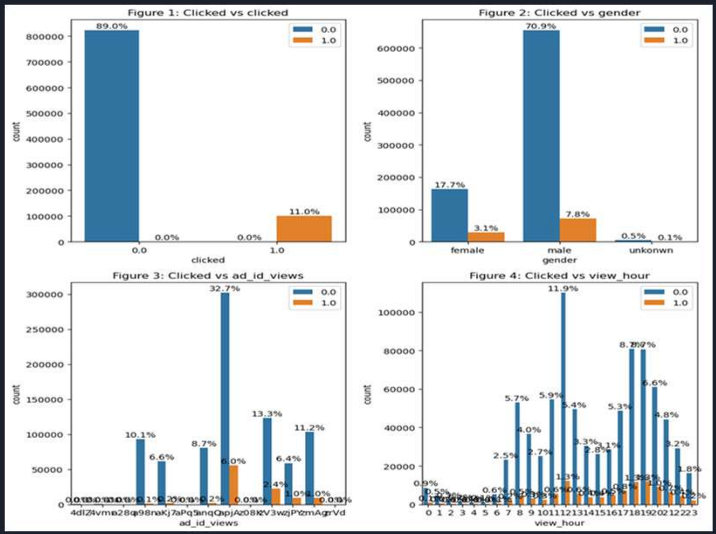
- **Considering business scenarios when building tables**
  - ad_info: could be important but provides few info, View+ad_info -> clicked
    - Interesting features: ad_loc, ad_label, add to analysis_data joining view
    - Missing data, city_id, industry, etc.
  - views: View-Click relation study
    - contain more info as it is the important step to click,
    - contain info of ad_info to find out the relation between view and clicked

3 Data - Cleansing, Visualization and Transform with pandas

- Get some general ideas: df.head(), df.shape, df.info(), df.isnum().sum(),df.describe()
- **numeric** features VS **category** features
  - corr(), heatmap for numeric features
  - .value_count, barplot to study the distribution of number of each category
  - Crosstab, Countplot, groupby to study the importance of category
- Add features:
  - view_time -> view_hour
  - address can also be considered mapping with chinese cities and districts
- Transform feature: genders, combine “” to “unknown”
- Drop features: payment_time, maid, mid, etc
- Features remained: ['money', 'kind_Pay', 'kind_Card', 'network', 'industry', 'gender',
- 'ad_id_views', 'clicked', 'view_hour']

```
df.corr(numeric_only=True)
```

	money	clicked
money	1.000000	0.004474
clicked	0.004474	1.000000



4.1 Modelling (Test/Train Split & Encoding Variables)

```python
# Perform train-test split
train_df, test_df = train_test_split(df, test_size=0.3, random_state=42)
```

```python
# Calculate IQR and determine upper threshold
Q3 = train_df['money'].quantile(0.75)
IQR = Q3 - train_df['money'].quantile(0.25)
upper_threshold = Q3 + 1.5 * IQR

# Apply upper threshold capping to both train and test data
train_df['money_capped'] = np.clip(train_df['money'], a_min=None, a_max=upper_threshold)
test_df['money_capped'] = np.clip(test_df['money'], a_min=None, a_max=upper_threshold)
```

```python
# Frequency encoding for ad_id on the training data
ad_id_counts = train_df['ad_id'].value_counts()
train_df['ad_id_freq'] = train_df['ad_id'].map(ad_id_counts)
# Mapping user_id frequencies from training to test data
test_df['ad_id_freq'] = test_df['ad_id'].map(ad_id_counts)
```

```python
# Frequency encoding for store_id on the training data
store_id_counts = train_df['store_id'].value_counts()
train_df['store_id_freq'] = train_df['store_id'].map(store_id_counts)
# Mapping user_id frequencies from training to test data
test_df['store_id_freq'] = test_df['store_id'].map(store_id_counts)
```

```python
# Frequency encoding for industry id n the training data
industry_counts = train_df['industry'].value_counts()
train_df['industry'] = train_df['industry'].map(industry_counts)
# Mapping user_id frequencies from training to test data
test_df['industry'] = test_df['industry'].map(industry_counts)
```

```python
features = [ 'money_capped', 'ad_id_freq', 'store_id_freq'
            , 'kind_Pay',  'kind_Card', 'network',
            'gender', 'industry','viewing_hour']

train_encoded = pd.get_dummies(train_df[features].fillna(0))
test_encoded = pd.get_dummies(test_df[features].fillna(0))
```

4.2 Modelling (Baseline - Decision Tree)

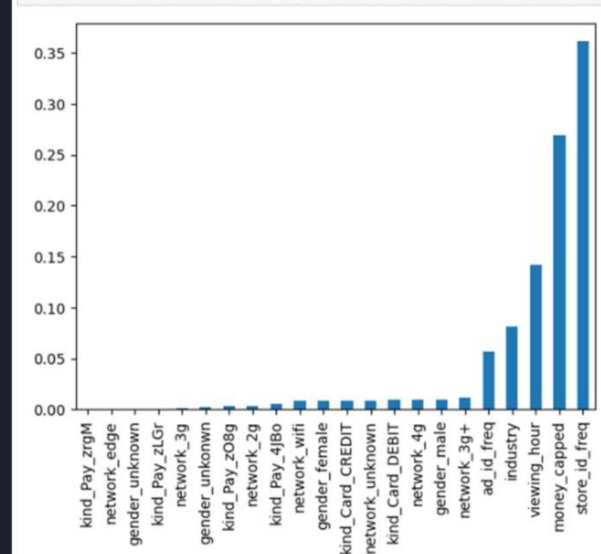
Separating feature matrix X_and target variable y

- X_train = train_encoded
- y_train = train_df['Clicked']
- X_test = test_encoded
- y_test = test_df['Clicked']

```python
# Create and fit the decision tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)

# Choosing DT
# Perform predictions on the test set
y_pred = dt_classifier.predict(X_test)
```

```python
pd.Series(dt_classifier.feature_importances_, index=X_train.columns).sort_values().plot(kind='bar');
```




```
Decision Tree Training Set Metrics:
Training Set AUC: 0.9945145855957995
Training Set Confusion Matrix:
[[751822   2242]
 [ 23894  70045]]
Training Set Accuracy: 0.969179354318322
Training Set Precision: 0.9691739222279934
Training Set Recall: 0.969179354318322
Training Set F1 Score: 0.9673901737854119
```

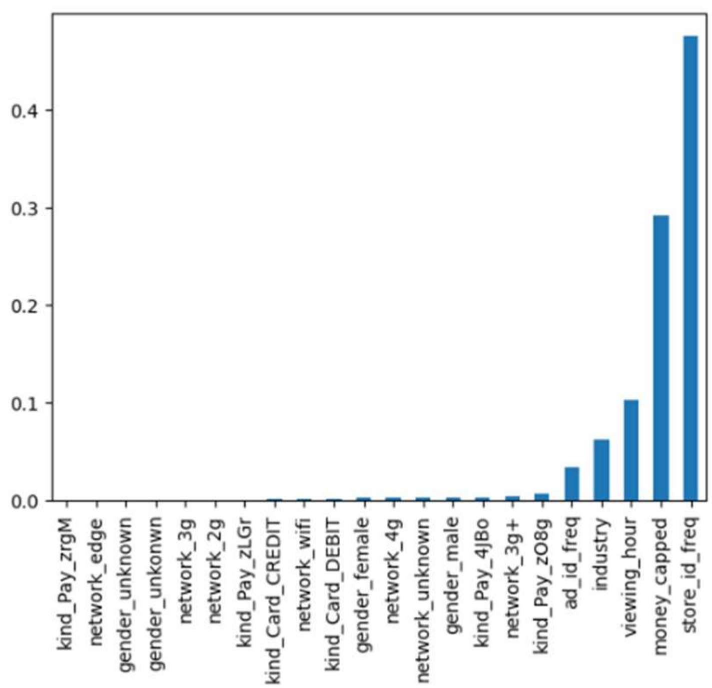
```
Decision Tree Test Set Metrics:
Test Set AUC: 0.6070529279318795
Test Set Confusion Matrix:
[[293165  30011]
 [ 29627  10627]]
Test Set Accuracy: 0.8359023745975841
Test Set Precision: 0.8365856907008711
Test Set Recall: 0.8359023745975841
Test Set F1 Score: 0.8362430916029198
```

4.3 Modelling (Baseline - Random Forest)

```
# Random Forest classifier
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)

y_pred = rf_classifier.predict(X_test)
```

```
pd.Series(rf_classifier.feature_importances_, index=X_train.columns).sort_values().plot(kind='bar');
```




```
Random Forest Training Set Metrics:
Training Set AUC: 0.9929335248231822
Training Set Confusion Matrix:
[[749355   4709]
 [ 21469  72470]]
Training Set Accuracy: 0.9691298261916527
Training Set Precision: 0.9684744098654253
Training Set Recall: 0.9691298261916527
Training Set F1 Score: 0.9677876969561521
```

```
Random Forest Test Set Metrics:
Test Set AUC: 0.6808666974376176
Test Set Confusion Matrix:
[[310580  12596]
 [ 32228   8026]]
Test Set Accuracy: 0.8766640068238726
Test Set Precision: 0.8487475876440665
Test Set Recall: 0.8766640068238726
Test Set F1 Score: 0.8585944552121542
```

4.4 Modelling (Model Validation)

```
# Create an instance of DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier()

# Create an instance of StratifiedKFold
stratified_kfold = StratifiedKFold(n_splits=20)

# Initialize lists to store the scores and metrics
scores = []
```

```
# Perform cross-validation
for k, (train, test) in enumerate(stratified_kfold.split(X_train, y_train)):
    X_train_fold = X_train.iloc[train, :]
    y_train_fold = y_train.iloc[train]
    X_test_fold = X_train.iloc[test, :]
    y_test_fold = y_train.iloc[test]

    # Fit the model on the training fold
    dt_classifier.fit(X_train_fold, y_train_fold)

    # Predict probabilities on the test fold
    y_pred_prob = dt_classifier.predict_proba(X_test_fold)[:, 1]

    # Calculate predicted labels on the test fold
    y_pred = dt_classifier.predict(X_test_fold)
```

```
Cross-Validation Accuracy: 0.835 +/- 0.002
```

```
# Create an instance of RandomForestClassifier
dt_classifier = RandomForestClassifier(n_estimators=100, max_depth=4)

# Create an instance of StratifiedKFold
stratified_kfold = StratifiedKFold(n_splits=5)

# Initialize lists to store the scores and metrics
scores = []
```

```
Cross-Validation Accuracy: 0.889 +/- 0.000
```

4.5 Modelling (GridSearch Hyperparameters)

```
from sklearn.model_selection import GridSearchCV

parameters = {
    'max_depth': [1, 2, 5, 10, 20,30,40],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2,5,10]
}

model = DecisionTreeClassifier()

gs = GridSearchCV(model, parameters, cv=5, scoring=['f1', 'accuracy'], verbose=2, n_jobs=-1, refit='f1')

gs.fit(X_train, y_train)
```

```
gs.best_params_
```

```
{'criterion': 'gini', 'max_depth': 40, 'min_samples_split': 2}
```

```
# Get the best estimator from the grid search
best_dt_classifier =  gs.best_estimator_

# Train the best estimator on the entire training data
best_dt_classifier.fit(X_train, y_train)

# Predict on the training set
dt_predicted_train = best_dt_classifier.predict(X_train)
```

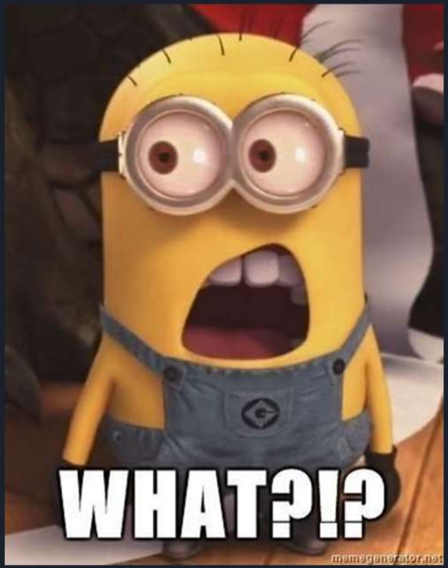
4.5+ Modelling (GridSearch Hyperparameters+)

```
Decision Tree Classifier (Training Set) Confusion Matrix:
[[751704   2360]
 [ 24994  68945]]
Decision Tree Classifier (Training Set) Accuracy: 0.9677430386449104
Decision Tree Classifier (Training Set) ROC AUC: 0.8654020075623309
Decision Tree Classifier (Training Set) Precision: 0.9677185512893708
Decision Tree Classifier (Training Set) Recall: 0.9677430386449104
Decision Tree Classifier (Training Set) F1 Score: 0.9657723463108007
```

```
Decision Tree Classifier (Test Set) Confusion Matrix:
[[293675  29501]
 [ 29783  10471]]
Decision Tree Classifier (Test Set) Accuracy: 0.8368764273725339
Decision Tree Classifier (Test Set) ROC AUC: 0.5844192962362588
Decision Tree Classifier (Test Set) Precision: 0.8363751972532089
Decision Tree Classifier (Test Set) Recall: 0.8368764273725339
Decision Tree Classifier (Test Set) F1 Score: 0.836625301452619
```

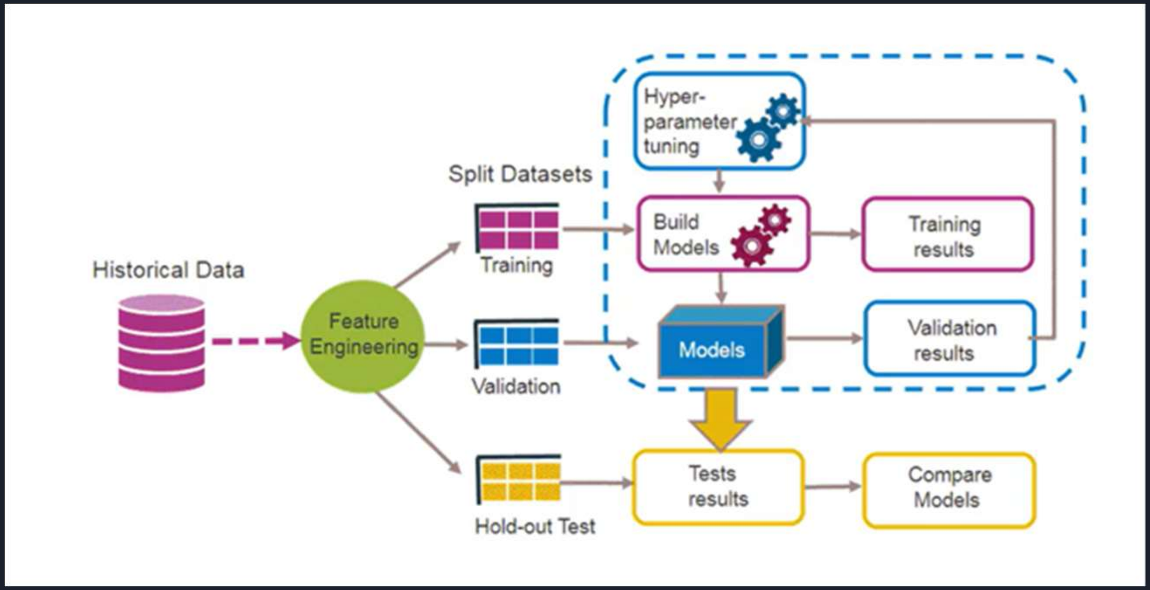
5 Challenges

- Many definitions, libraries, models … -> like ‘Confusion matrix’
- Take long time to run the large data and transfer from sql to Python
- Deal with missing values & datetime (bining)
- Super imbalance dataset (90% vs 10%)
- Not get the high precision & accuracy scores
- White spaces using sqlalchemy -> use regex command (as Malik case)
- … to be continued …



6 Conclusion

- Practical assignment
- Go through whole process: Raw data -> SQL -> Python -> ML
- Apply , Mix & Match: SQLAlchemy, Pandas, Numpy, Libraries & Models in ScikitLearn/ imblearn
- Validating how the best performing model



7 Next steps

- Manage & estimate how long to make the large data over 1 million rows
- Study more and apply Pipeline for filling missing values & building combinations of models
- Using GridSearch for generating parameter of other models
- Study more some efficient way to tune parameters to get the best performing model

Thank you!