GROUP PROJECT
Home Credit Default Risk

Aoran LI, Tianying ZHOU, Langting WENG, Yijia MA

START

Topic

1 Exploratory Data Analysis

2 Feature Engineering

3 Model Construction

4 Conclusion

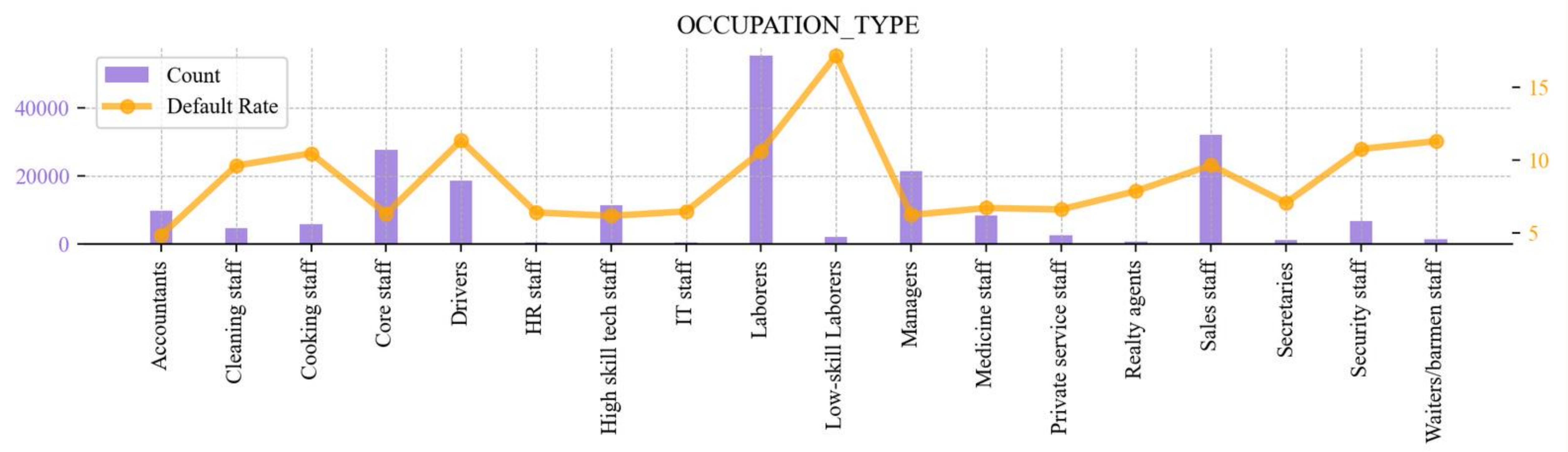
1

Exploratory Data Analysis

Categorical Variable

1.

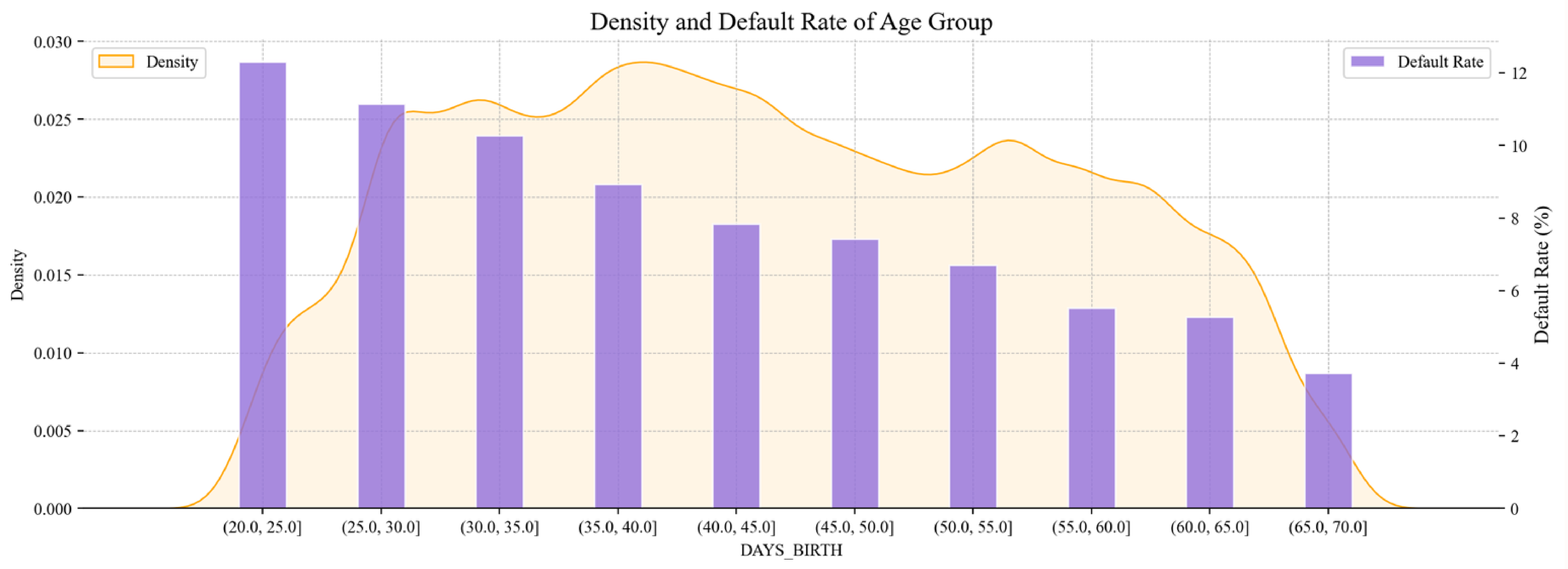
Chi-square Test



Numerical Variable

2.

Point-Biserial Correlation



2

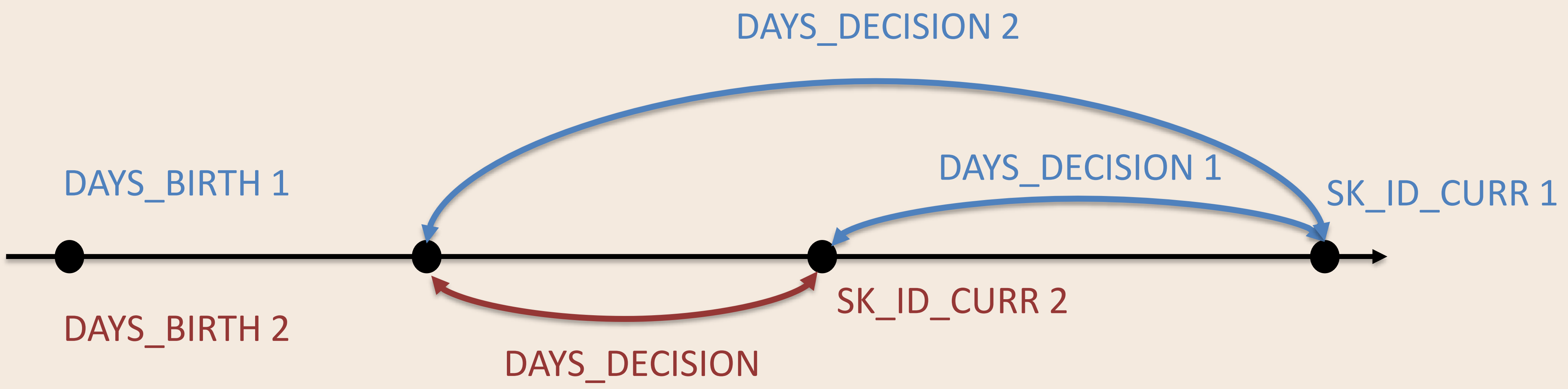
Feature Engineering

Dataset

application_{train|test}.csv
previous_application.csv
installments_payments.csv
bureau.csv
bureau_balance.csv

Feature Engineering Pt.1

User Identify – previous default



1. Both applicants have the same gender.
2. Difference between two DAYS_DECISION of 1 = DAYS_DECISION of 2
3. DAYS_BIRTH of 1 - DAYS_DECISION 1 of 1 = DAYS_BIRTH of 2

Feature Engineering Pt.2

Installments payments analysis

- Original features;
- New features: Constructed by simple operations, such as calculate the total default amount for each SK_ID_CIRR.

Previous application analysis

- Define functions that do one-hot encoding on a dataframe;
- New features: Conducted by performing a calculation on the grouped data ,and match the aggregated statistics to the appropriate client.

Feature Engineering Pt.3

"Bureau" & "Bureau balance" analysis

- Original features do not have significant performance; ✗
- New features: Extracted some significant features via some statistical analysis, like calculating the minimum, maximum, summation, and so on. √

3

Model Construction

Model Construction

To show the performance of different models on different feature sets, we selected **logistic regression**, **random forest**, and **lightgbm** as comparison models and applied them to predict on different feature sets (one generated features using all data, and the other generated features using only part of the data).

In 2

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score, confusion_matrix, accuracy_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import lightgbm as lgb

import pandas as pd
import numpy as np
import gc
import os
```

Code

We have developed a function that performs the entire process from model training to inference.

The function takes in the training dataset and test dataset data, and selects the corresponding model based on the input model name for k-fold cross-validation inference.

In simpler terms, this function allows you to input different sets of features and choose different training models for each call. This makes it easy to compare the effectiveness of different models and data when it comes to training.

This section of code is responsible for performing basic data cleaning and splitting the input data into x and y for the model.

In 1

```python
def model_construct(train_data, test_data, model, n_folds = 5):
    """
    :param model: name of model: 'logistic', 'RamdomForest', 'lgbm'
    :param train_data:  training data df, include ids, features and label
    :param test_data:  testing data df
    :param n_folds: number of folds to use for cross validation, default 5
    :return:
    """
    # Get Data
    train_ids = train_data['SK_ID_CURR']
    test_ids = test_data['SK_ID_CURR']

    x_train = train_data.drop(['TARGET', 'SK_ID_CURR'], axis=1)
    x_test = test_data.drop('SK_ID_CURR', axis=1)
    y_train = train_data['TARGET']

    # Extract feature names
    feature_names = list(x_train.columns)

    # Convert to np arrays
    x_train = np.array(x_train)
    x_test = np.array(x_test)

    # Preprocessing
    imputer = SimpleImputer(strategy='most_frequent')
    scaler = MinMaxScaler(feature_range=(0, 1))
    # Imputer
    x_train = imputer.fit_transform(x_train)
    x_test = imputer.transform(x_test)
    # Scalar
    x_train = scaler.fit_transform(x_train)
    x_test = scaler.transform(x_test)

    print('Training Data Shape: ', x_train.shape)
    print('Testing Data Shape: ', x_test.shape)
```

Logistic model

Before creating each model, we first create a k-fold classification object, which would divide the dataset into n groups for n-fold training task. Meanwhile, we will generate a series of empty lists and zero np array to store the future data.

In each function call, we train the same model **n-fold** times. Each training session records the model's predictions and performance on the training set, valid set and testing set. Finally, we take the average of the **n-fold** training model on the test set set as the final output of the model.

```python
# Create the kfold object
k_fold = KFold(n_splits = n_folds, shuffle = True, random_state = 9170)

# Empty array for feature importances
feature_importance_values = np.zeros(len(feature_names))

# Empty array for test predictions
test_predictions = np.zeros(x_test.shape[0])

# Empty array for out of fold validation predictions
out_of_fold = np.zeros(x_train.shape[0])

# Lists for recording validation and training scores
valid_scores = []
train_scores = []
```

```python
if model == 'logistic':
    # K fold iteration
    for train_index, valid_index in k_fold.split(x_train):
        # train set
        x_tr, y_tr = x_train[train_index], y_train[train_index]
        # valid set
        x_va, y_va = x_train[valid_index], y_train[valid_index]

        lr_model =  LogisticRegression(C = 0.0001, random_state=9171)
        lr_model.fit(x_tr, y_tr)

        # feature importance
        feature_importance_values += np.abs(lr_model.coef_[0]) / k_fold.n_splits
        # test prediction
        test_predictions += lr_model.predict_proba(x_test)[:, 1] / k_fold.n_splits  # only need the second column
        # valid predition
        out_of_fold[valid_index] = lr_model.predict_proba(x_va)[:, 1]

        # train score/ valid score
        train_scores.append(roc_auc_score(y_tr, lr_model.predict_proba(x_tr)[:, 1]))
        valid_scores.append(roc_auc_score(y_va, lr_model.predict_proba(x_va)[:, 1]))

        gc.enable()
        del lr_model, x_tr, x_va, y_tr, y_va
        gc.collect()
```

Logistic regression model's code

RandomForest

Random Forest regression tree model's code →

In each model's construction part, we first spilt the train and valid data set. Then create an model object and fit it with train data set. After that, we record the model's feature importance, model's metrics on valid and testing data, and the prediction on both valid and testing data.

To ensure sufficient computing memory, we perform variable deletion and memory clearing after each training session.

```python
elif model == 'RandomForest':
    # K fold iteration
    for train_index, valid_index in k_fold.split(x_train):
        # train set
        x_tr, y_tr = x_train[train_index], y_train[train_index]
        # valid set
        x_va, y_va = x_train[valid_index], y_train[valid_index]

        # model
        rf_model = RandomForestClassifier(n_estimators = 100, random_state = 9172, verbose = 1, n_jobs = -1)
        rf_model.fit(x_tr, y_tr)

        # feature importance
        feature_importance_values += rf_model.feature_importances_ / k_fold.n_splits
        # test prediction
        test_predictions += rf_model.predict_proba(x_test)[:, 1] / k_fold.n_splits
        # valid prediction
        out_of_fold[valid_index] = rf_model.predict_proba(x_va)[:, 1]

        # train / valid score
        train_scores.append(roc_auc_score(y_tr, rf_model.predict_proba(x_tr)[:, 1]))
        valid_scores.append(roc_auc_score(y_va, rf_model.predict_proba(x_va)[:, 1]))

        gc.enable()
        del rf_model, x_tr, x_va, y_tr, y_va
        gc.collect()
```

Lightgbm

Light-gbm regression tree model's code →

```
elif model == 'lgbm':
    # K fold iteration
    for train_index, valid_index in k_fold.split(x_train):
        # train set
        x_tr, y_tr = x_train[train_index], y_train[train_index]
        # valid set
        x_va, y_va = x_train[valid_index], y_train[valid_index]

        # model
        lgb_model = lgb.LGBMClassifier(n_estimators=1000,
                                       objective = 'binary',
                                       learning_rate = 0.05,
                                       reg_alpha = 0.3,
                                       reg_lambda = 0.5,
                                       n_jobs = -1,
                                       random_state = 9173)
        lgb_model.fit(x_tr, y_tr,
                      eval_metric = 'auc',
                      eval_set= [(x_va, y_va), (x_tr, y_tr)],
                      eval_names=['valid', 'train'],
                      # early_stopping_rounds = 100,
                      callbacks=[lgb.log_evaluation(period=200), lgb.early_stopping(stopping_rounds=100)]
                      )
        best_iteration = lgb_model.best_iteration_

        # feature importances
        feature_importance_values += lgb_model.feature_importances_ / k_fold.n_splits
        # test prediction
        test_predictions += lgb_model.predict_proba(x_test, num_iteration=best_iteration)[:, 1] / k_fold.n_splits
        # valid prediction
        out_of_fold[valid_index] = lgb_model.predict_proba(x_va, num_iteration=best_iteration)[:, 1]

        # train / valid scores
        train_scores.append(lgb_model.best_score_['train']['auc'])
        valid_scores.append(lgb_model.best_score_['valid']['auc'])
```

Same as coding construction as above.

Due to time constraints, we were unable to optimize the hyperparameters of the model. Therefore, we randomly fixed some hyperparameters for training.

Output

Function's ending code →

At the end of the function, it will integrate different fold model's result and out put the submission data-frame and model's performance.

And you can upload the submission csv to Kaggle platform to check the public scores.

```
        gc.enable()
        del lgb_model, x_tr, x_va, y_tr, y_va
        gc.collect()

else:
    raise ValueError('Only support logistic, randomforest, lgbm model')


submission = pd.DataFrame({'SK_ID_CURR': test_ids, 'TARGET': test_predictions})
feature_importance = pd.DataFrame({'feature': feature_names, 'importance': feature_importance_values})

valid_auc = roc_auc_score(y_train, out_of_fold)
valid_scores.append(valid_auc)
train_scores.append(np.mean(train_scores))
fold_names = list(range(n_folds))
fold_names.append('overall')

# Dataframe of validation scores
metrics = pd.DataFrame({'fold': fold_names,
                        'train': train_scores,
                        'valid': valid_scores})

return submission, feature_importance, metrics, out_of_fold
```

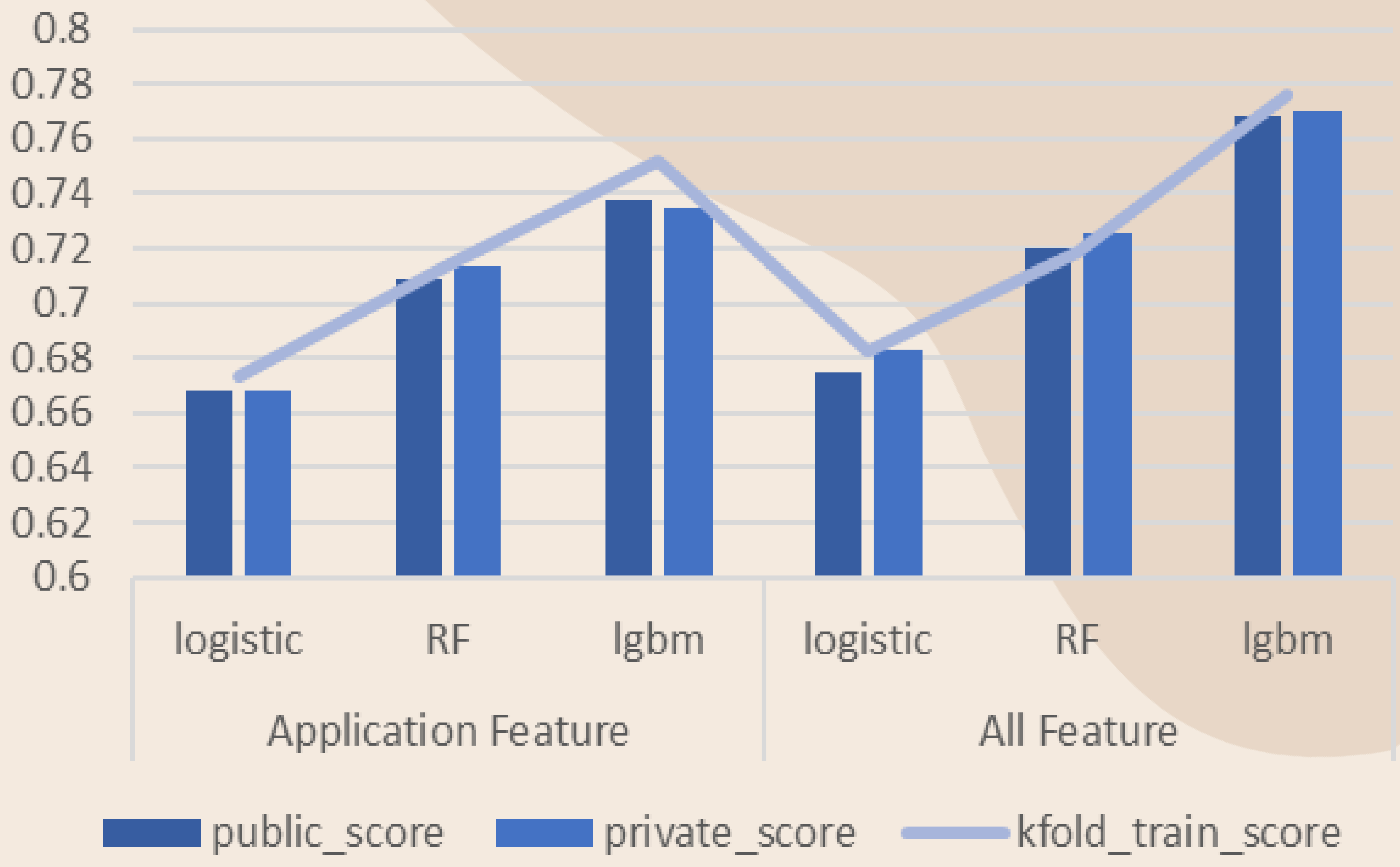
Model Performance

In this section, we upload the test data prediction result on kaggle and compare different model's public score and private score.

It is evident that the **lgbm model** outperforms the other two models.

We also tested the predictive ability of the models under different feature sets, and the results showed a significant improvement in performance for all three models when more features from different tables were added.

However, when we applied some processing techniques to the original features (such as interpolation, mean aggregation, etc.), there was no significant improvement in the model's prediction results.



Model performance with all feature

	public_score	private_score	kfold_train_score
logistic	0.67483	0.68265	0.682569
RandomForest	0.71975	0.72521	0.718389
lgbm	0.76849	0.77002	0.775649

Model performance with processing feature

	public_score	private_score	kfold_train_score
logistic	0.67783	0.68420	0.683688
RandomForest	0.71375	0.72173	0.718786
lgbm	0.76720	0.77044	0.776433

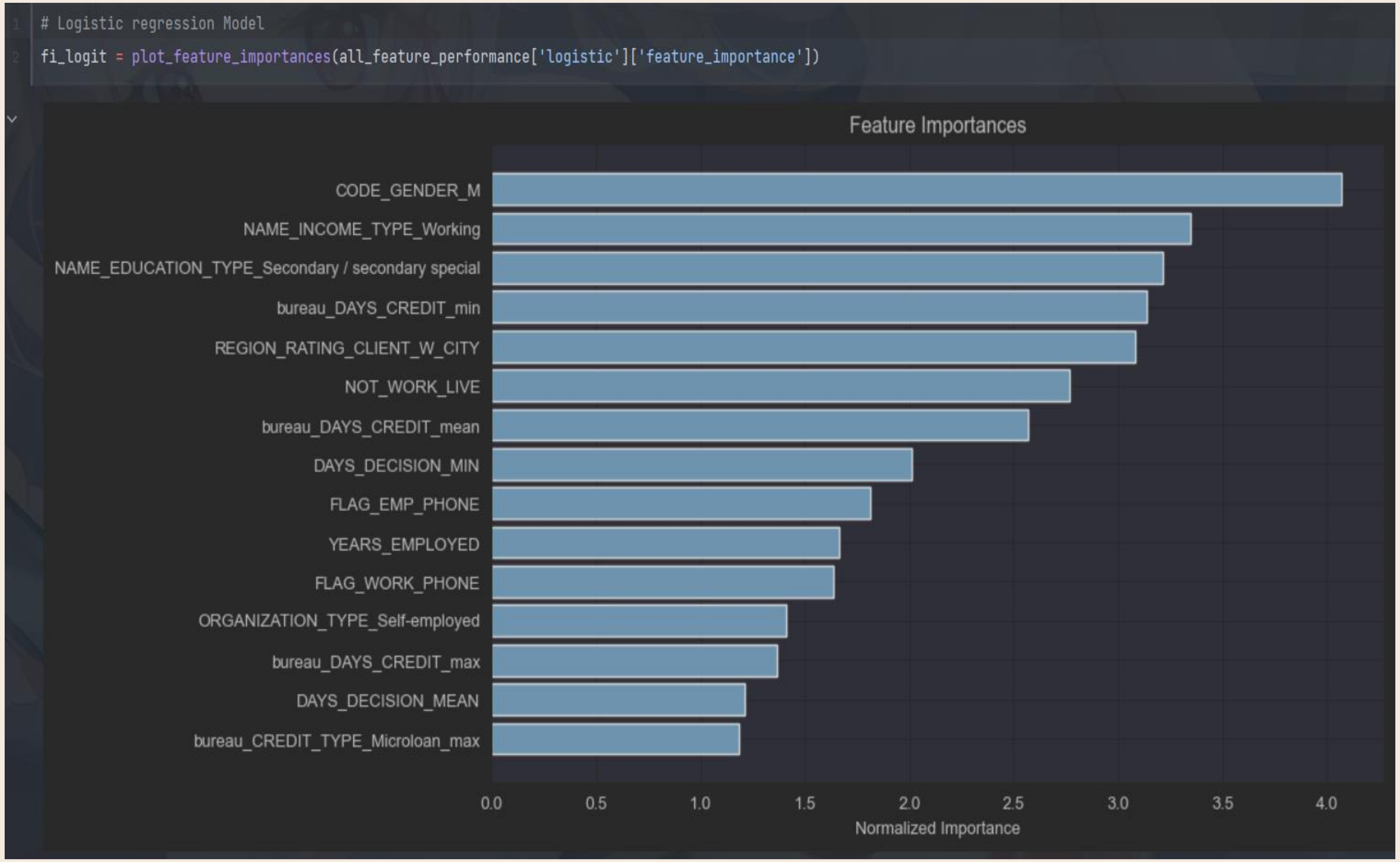
Feature Importance

In this section, we analyze the importance of all feature derived from the bureau, previous_application and application table. We use the best performance model: the **all_feature_trained model** to check the feature importance.

We output the **top 15** important factors according to the model trained above, and finally take the intersection to obtain the factors that are considered impotant by all three models.

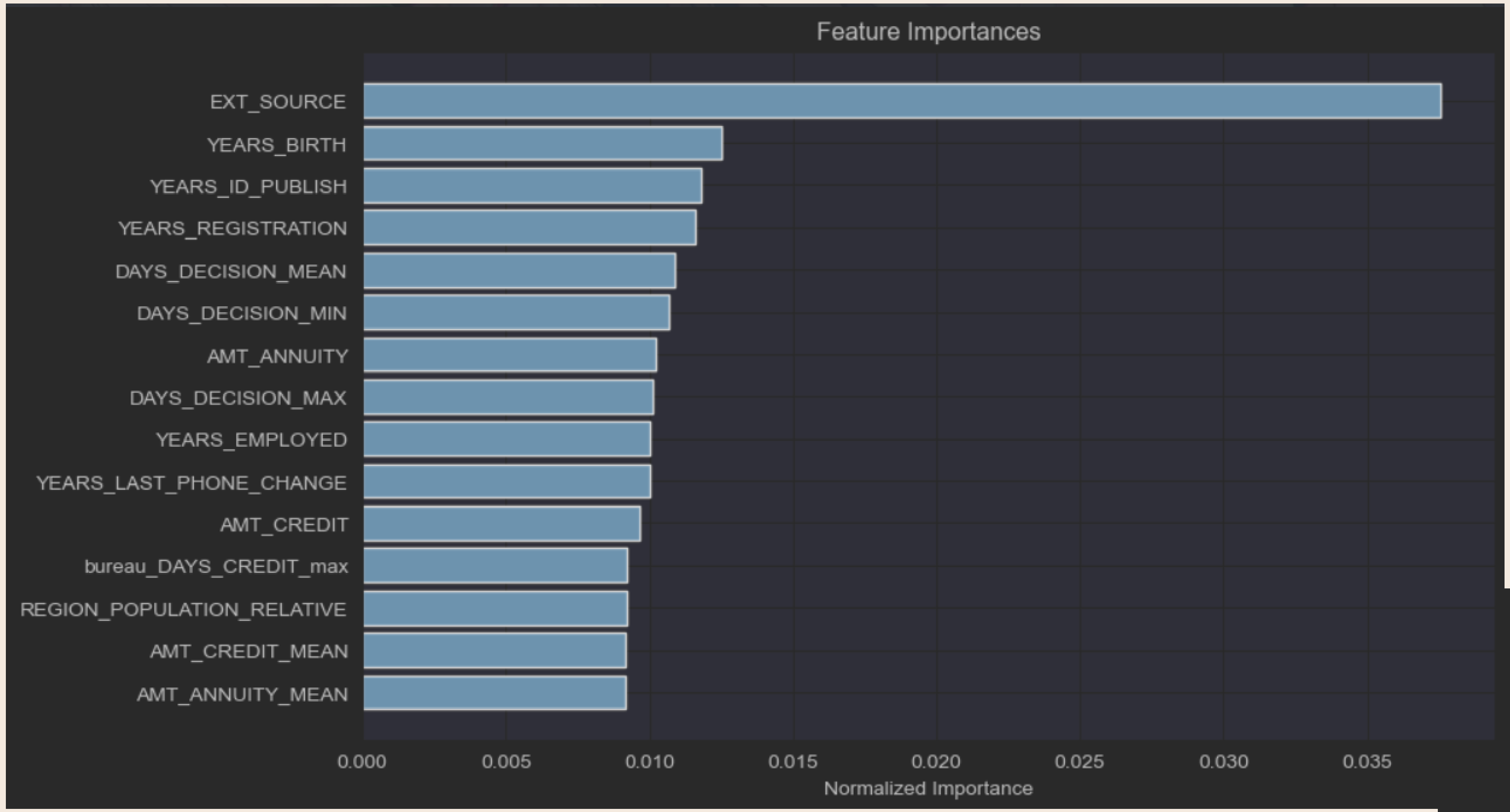
Both **lgbm and random forest** model give high score to EXT_SOURCE、YEAR_BIRTH、YEAR_PUBLISH, etc. On the contrary**, logistic regression** in the regression model has a significant divergence from the tree model in terms of variable importance.

Logistic regression model

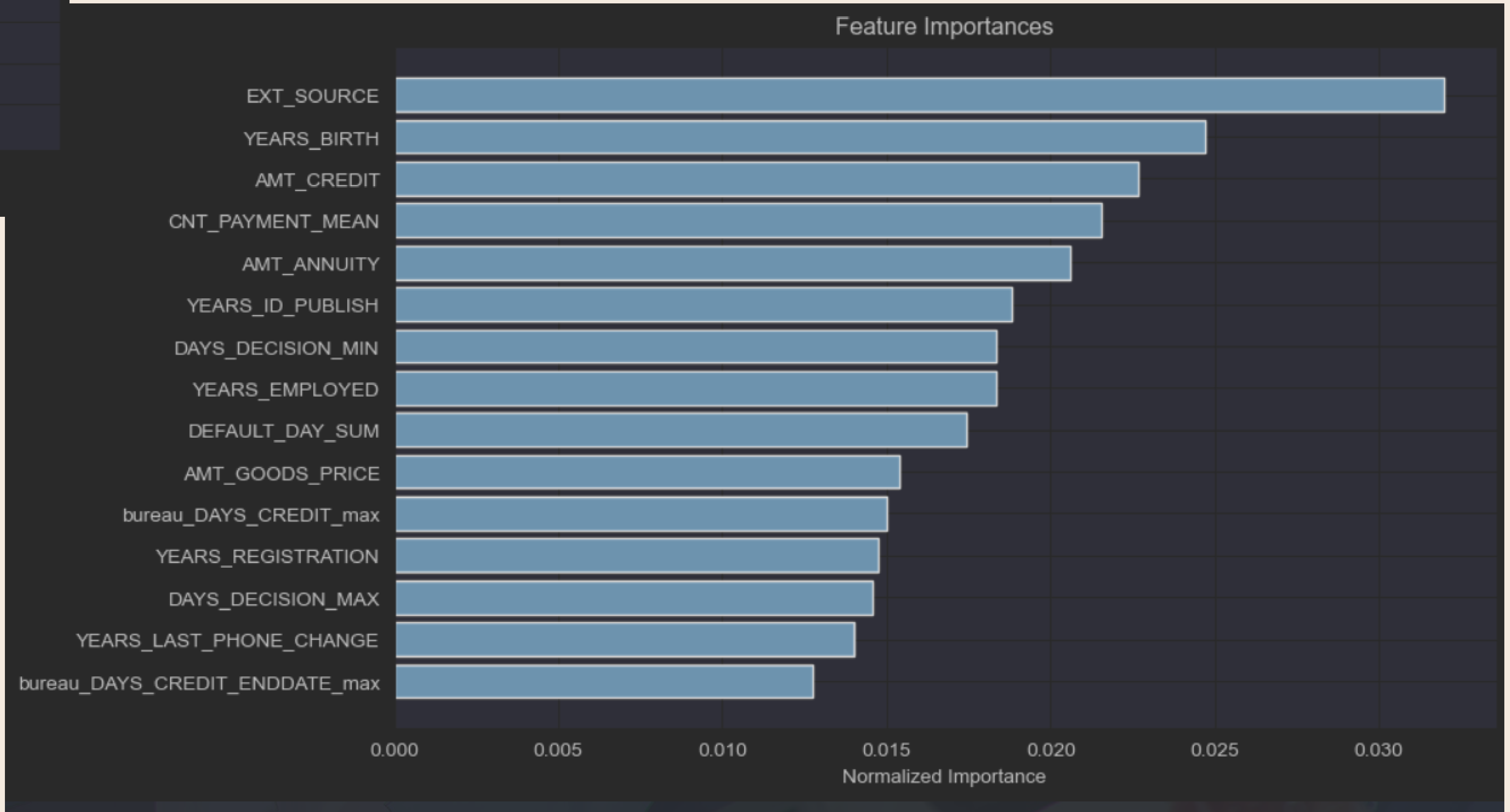


Feature Importance

Random Forest model



LGBM model



4

Conclusion

Conclusion & Future Work

Conclusion:

- Feature engineering is of great help in improving the prediction performance of models, as new features often bring new information.

- Not all feature engineering can improve the model performance. Only when **new features** can provide new information to the model can the performance of the model be improved.

- Similar models have similarities in variable selection. Tree model and linear model capture different patterns of features.

Future Work:

- model hyperparameter tunning

- adding more domain relative feature

Reference:

- Will koehrsen. (n.d.). *Introduction to Manual Feature Engineering*. Kaggle.

Question Time

THANK YOU SO MUCH!