Shopee’s User Churn Prediction Model

User churn, also known as attrition, occurs when a user stops using our product, in this case it is **Shopee** **(A giant E-commerce players in Southeast-Asia, direct competitor of TikTok Shop**) mobile version browser or more precisely the Android version. Understanding and detecting churn is the first step to retaining these users and improving DAU, MAU as well other key metrics.

A. Define Problem

1. How do we define a churned user in our business context?

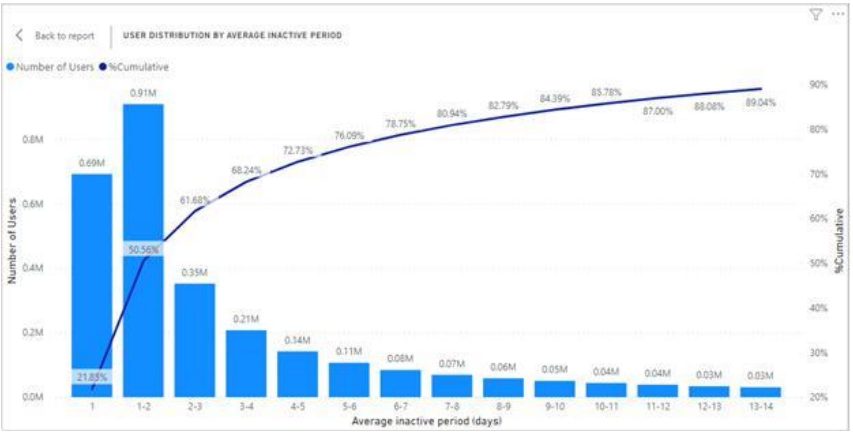
Let’s say we have a user who hasn’t been active in the last X days, how likely is it for him/her to churn?

To answer the first and foremost question, we need to find X that maximizes the likelihood of churning. In other words, we need to find X such that users who hasn’t been active in the last X days are most likely to churn (predicted to churn).

There are so many ways and methods that could help us find such X. In this paper, I implemented 3 solutions and picked the most suitable out of them.

Solution 1: Average inactive period in users' lifetime

Using dataset of all active users who have at least 2 active days in this current year 2022, I calculated average inactive period for each user to see their frequency of using our app and have a look at its distribution.

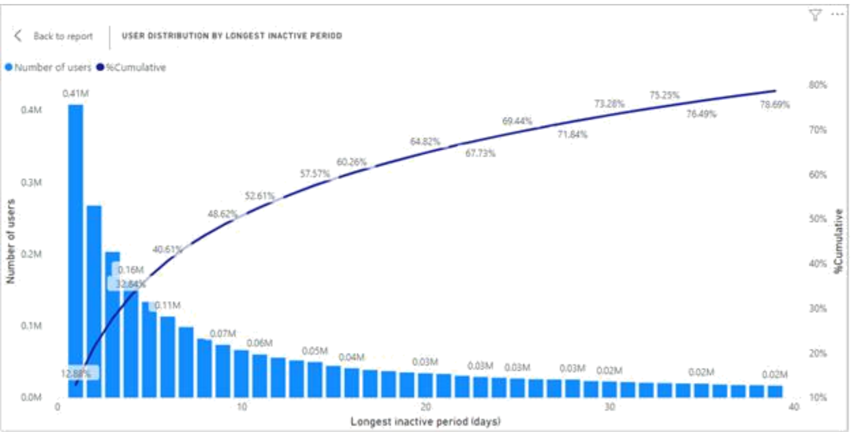


A.1.1. User distribution by average inactive period

We can easily notice that ~90% of our users have their average inactive period less than or equal to 14 days. Therefore if a user is inactive for 14 consecutive days there is high chance that this user already churned.

Solution 2: Longest inactive period in users' lifetime

I also used the same dataset as former solution, however in this case instead of average inactive period our target is the longest inactive period in order to know precisely churn probability of a user with X inactive days till observed date.



A.1.2. User distribution by longest inactive period

From above distribution, it is noticeable that 58% and 72% users have their longest inactive period less than or equal to 14 and 28 days respectively. We can also say a user who has been inactive for the last 14 and 28 days has 58% and 72% chances respectively to actually churn.

Solution 3: Actual likelihood of churning for users who has been inactive for X days from their last active dates

At each data point (or observed date), I find all users who were active at that date but inactive in the next X days (weeks). I mark these users as churn in X days (weeks) - churn_x then compare with their actual last active date till current. If their actual last active date are the same as observed date they are also marked as churn till current - churn_current. The rate churn_current / churn_x is the actual churn likelihood of x day (week) inactive users.

Due to time limitation and massive daily data, I only pick sample of 28 observed dates from 2022-07-18 to 2022-08-14 and calculate average churn probability in this period for 2 groups: 14 and 28 day inactive users. The result is shown below.

Churn_current / Churn_14d	**Churn_current / Churn_28d**
51.4%	75.1%

Pretty consistent to previous solution, right????

Conclusion

From above results, in the scope of this paper I define a user at high risk to churn such that he/she has been inactive for the last 28 days

2. What does the churn model look like?

Typically, using historical data, we know which clients eventually left and which did not. However, we need to be careful about how far in advance we want to estimate the propensity to leave. If that time is too short, we won't have much time to make any kind of response. If, on the other hand, it is too long, the model will be less accurate and up to date.

From the previous part, we already determined that users with 28 days of inactivation are at high risk to churn. I have two model ideas:

- Idea 1: For all active users on a specific date (observed date), we look back 28 days into their data and try to predict if they will be active/inactive in the next 28 days.
    - Why do we look at the past 28 days?
- Idea 2: For all users active in the last 28 days from a specific date (observed date), we look at their data in this range and predict if they will be active/inactive in the next 14/28 days.
    - Why is 14/28 days in advance we want to predict? After trying to implement this idea I find that the average inactive period in the last 28 days from the observed date of actual churned users in the next 14 days is ~12.5 days. Since our target is to determine users with 28-day inactivation, the next 14 days is the target point. However, to have a more general view, I will also implement a model to predict 28 days in advance and compare the two models together

In my point of view, idea 1 seems to be better for our business since if it goes well we can detect a potential churn user right at their last active date. However, since we have no data about their inactive period from the last active date to ref. date, which is an important feature affecting the likelihood of churning, thus leading to a bad model.

I will start with idea 2 first since it is easier to implement and may have more positive outcomes. I leave idea 1 as what we can do afterwards for improvement.

3. What do we need for the churn model?

Like any supervised machine learning model, a churn model needs training data with output (target) and input (explanatory variables). Based on this training data, the model learns to best capture the relationship between input and output.

Output

As discussed above, the model will look at our D28 user pool at the observed date and then try to predict the status of each user in our pool in the next 14/28 days. Therefore, the defined target in this case is whether a user is active (churn_xd = 0) or inactive (churn_xd = 1) in the next 14/28days (x days).

Input

Our input needs to include any data that can help identify clients who churn. Intuitively, I break it down into two variable categories: Engagement and Features.

The first category consists of variables describing users' engagement with our browser. Some can be mentioned such as how many active days, how many sessions per day, how frequently any feature is used, etc… The list of all these metrics used in this paper is below in the summary part (probably not enough, or some variables maybe not be appropriate, please review and feedback).

The latter category helps determine the characteristics and features of users who are likely to churn. For example: how long have they been using Shopee browser (their seniority), whether they use any feature such as download, cinema mode, search, etc… All metrics used can also be found below.

Summary

- ***Target:***

  churn_xd (x = 14/28): *is this user active (churn_xd = 1) or inactive (churn_xd = 0) in the next x days from observed date*

- ***Explanatory variables:***

  Engagement:

  active_days_last_4w: *number of active days in the past 4 weeks from the observed date*

  inactive_days_till_ref: *number of inactive days from the last active date to the observed date*

  total_sessions_last_4w: *number of sessions browsed in the past 4 weeks from the observed date*

  total_dowload_events_last_4w: *number of download events in the past 4 weeks from the observed date*

  total_cinema_events_last_4w: *number of cinema events in the past 4 weeks from the observed date*

  total_pin_events_last_4w: *number of pin events in the past 4 weeks from the observed date*

  total_headset_events_last_4w: *number of headset events in the past 4 weeks from the observed date*

  total_queries_last_4w: *number of queries searched in the past 4 weeks from the observed date*

  total_clicks_last_4w: *number of clicks at newsfeed page in the past 4 weeks from the observed date*

  seniority_till_last_active: *number of days from first to last active date*

  Features:

  is_dowload_user: *have any dowload event in the past 4 weeks from observed date (yes = 1, no = 0)*

  is_cinema_user: *have any cinema event in the past 4 weeks from observed date (yes = 1, no = 0)*

  is_pin_user: *have any pin event in the past 4 weeks from observed date (yes = 1, no = 0)*

  is_headset_user: *have any headset event in the past 4 weeks from observed date (yes = 1, no = 0)*

  is_search_user: *have any search query in the past 4 weeks from observed date (yes = 1, no = 0)*

  is_nf_user: *is impressed or click at newsfeed page (yes = 1, no = 0)*

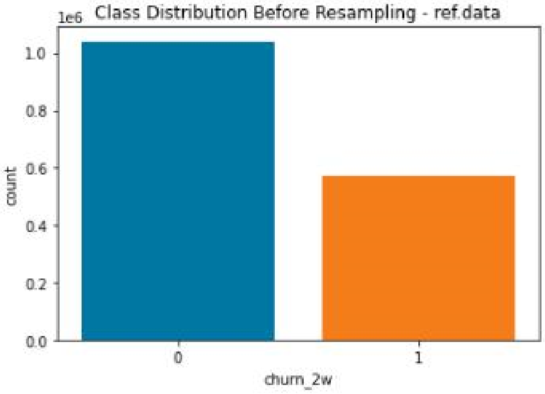
4. How do we collect data for the churn model?

At each observed date (ref_date), we roll back 28 days to access our D28 user pool and collect all explanatory variables (input) as discussed above. Then we go beyond X days from the observed date to collect the target variable (output). In this paper scope, I picked 8 observed dates with a 28-day gap between each corresponding with 8 D28 user pools from the beginning of 2022 as follows: 30jan, 27feb 27mar, 24apr, 22may, 19jun, 17jul, 14aug.

B. Exploratory Data Analysis

1. Target variable

The dataset we are using is pre-cleaned so we can jump to explore it right away. Let's start with the target variable (churn_2w):



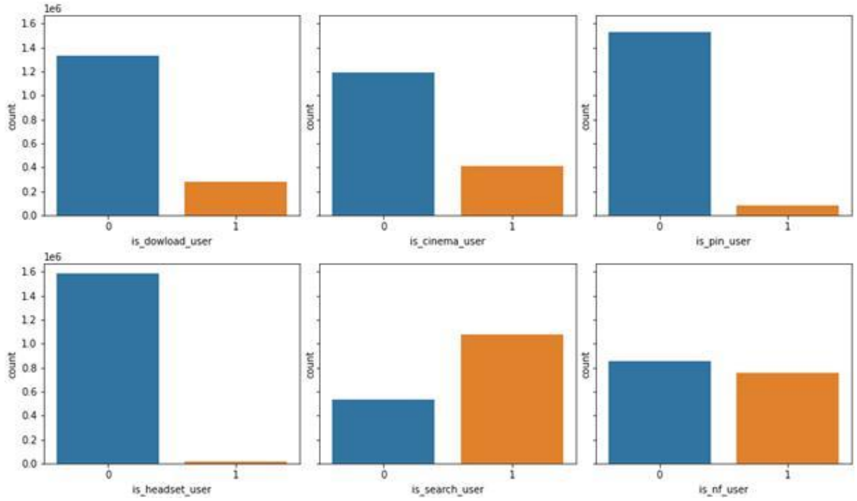
B.1.1. Target variable - Class distribution

The target variable has imbalanced class distribution. Positive class (churn_2w=1) is much less than negative class (churn_2w=0). Imbalanced class distributions influence the performance of a machine learning model negatively. We will use upsampling or downsampling method to overcome this issue.

2. Explanatory variables

We then explore the independent variables which were already divided into two categories: Feature and Engagement. We first discover the former variables which all are dummy.

Feature variables

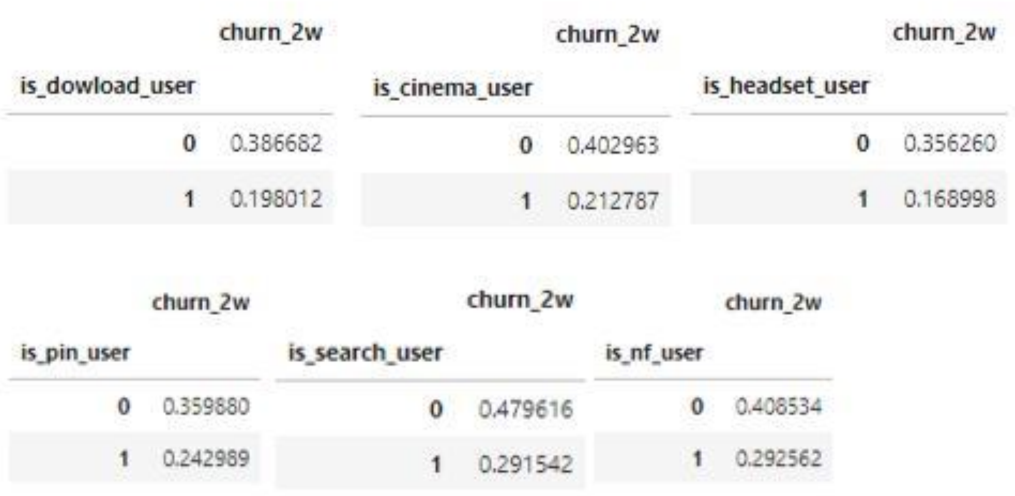


B.2.1. Feature variables - Class distribution

There is a high imbalance in is_dowload_user, is_pin_user and is_headset_user variables. It is quite intuitive that mobile users don't use these features much since there are so many other apps serving specifically for these purposes on mobile.

We can also notice that most users use our browser for searching and reading news on the newsfeed, and around 25% of users watch movies on their phones.

It is better to check how the target variable (churn) changes according to the dummy features. Churn rate for each categorical feature is shown below:



	churn_2w
is_dowload_user	
0	0.386682
1	0.198012

	churn_2w
is_cinema_user	
0	0.402963
1	0.212787

	churn_2w
is_headset_user	
0	0.356260
1	0.168998

	churn_2w
is_pin_user	
0	0.359880
1	0.242989

	churn_2w
is_search_user	
0	0.479616
1	0.291542

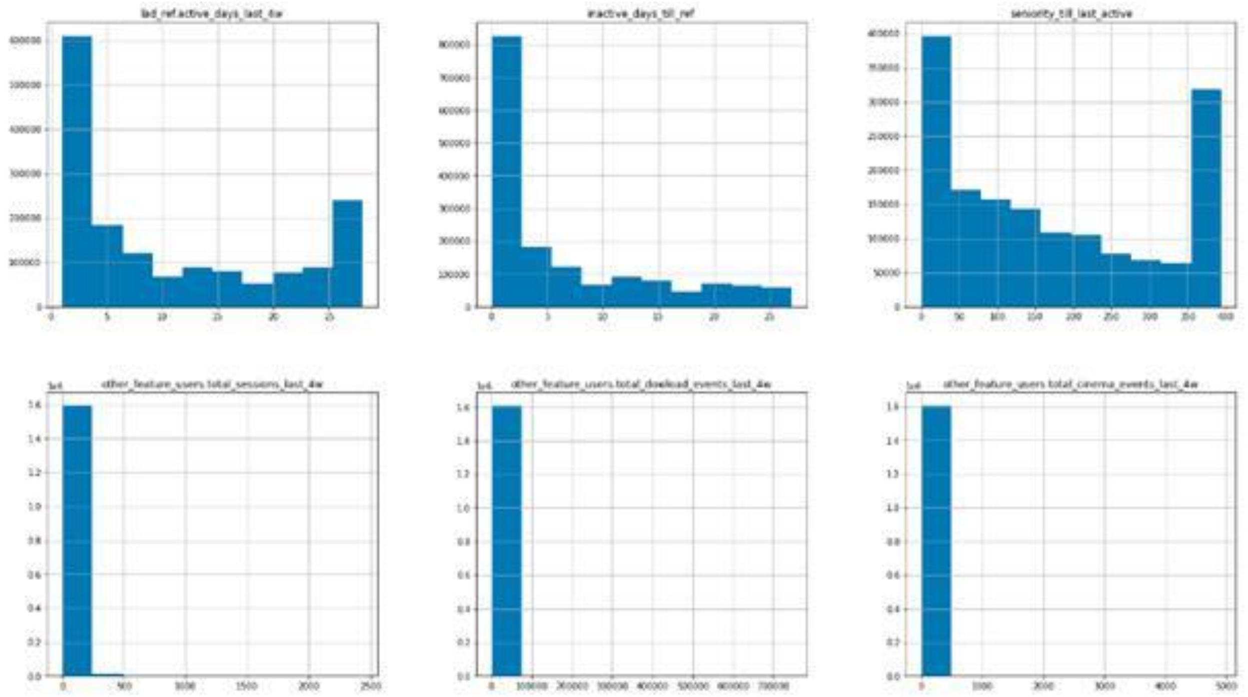
	churn_2w
is_nf_user	
0	0.408534
1	0.292562

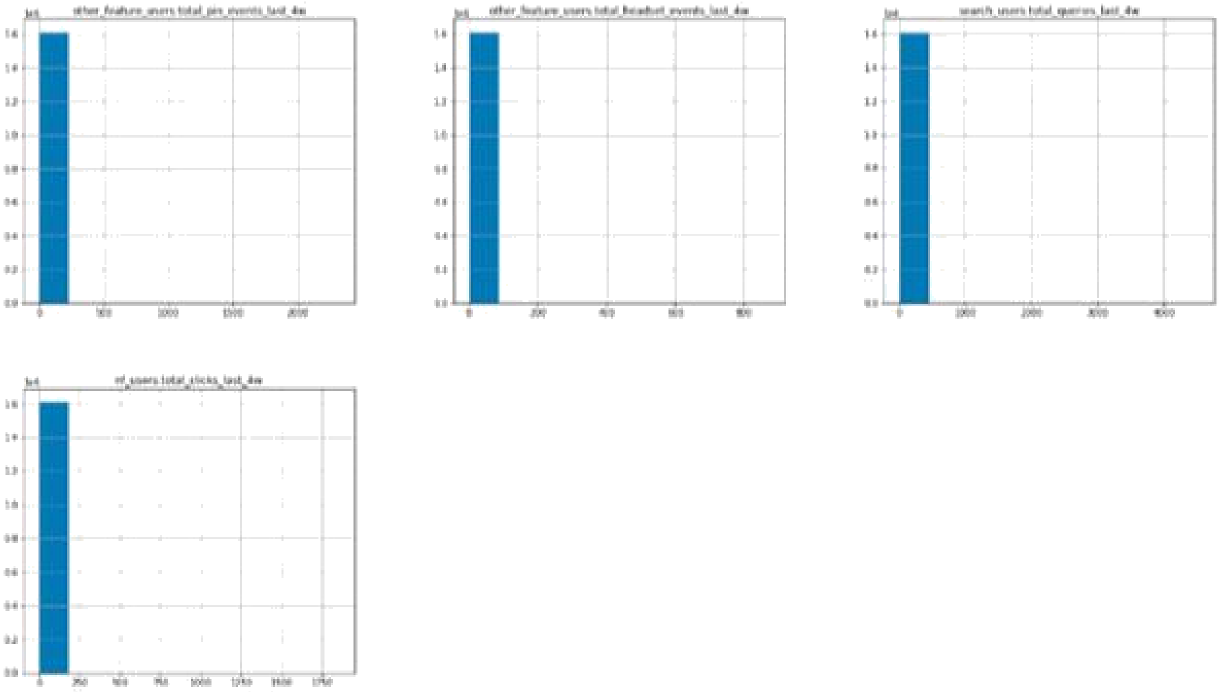
B.2.2. Churn rate by Categorical Feature

We can easily notice that these features have substantial effects on target variable. However, the effects of variables with high imbalance may not be represented correctly.

Engagement variables

We next explore the variables representing level of engagement between users and our browser. All of them are continuous variables which we want to have a look at their distribution.





B.2.3. Engagement variables - Distribution

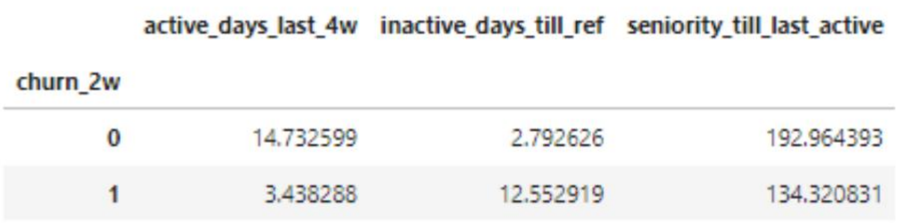
According to the distribution of the active_days_last_4w variable, most users are active for less than 7 days in total of 28 days (< 25%). There is also a substantial group of users who are active for more than 25 days in total of 28 days (almost every day).

Moving on, we can see that most users have been inactive for less than 7 days till ref_date, which is quite intuitive since we have metrics showing that average number of D7 users ~ 2/3 number of D28 users.

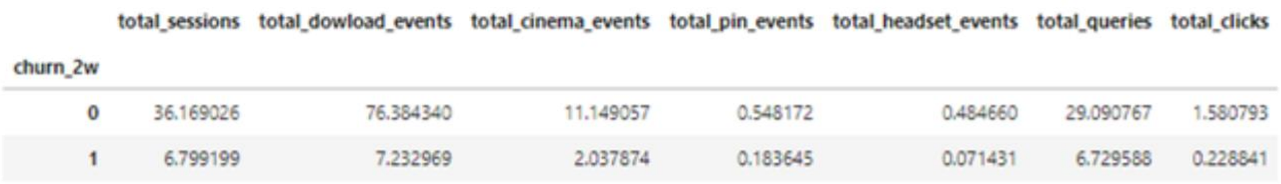
Furthermore, most of the users are either pretty new or have stayed for a long time with the company (distribution of seniority_till_last_active variable). Our goal should be to find a way to keep those customers with seniority of up to a few months.

For other variables, we do not capture any clear pattern or trend from their distributions. However, I have decided to keep these variables in the model as a measure of how frequent the browser (features of the browser) is used.

Next, let’s check how churn rate changes according to engagement features:



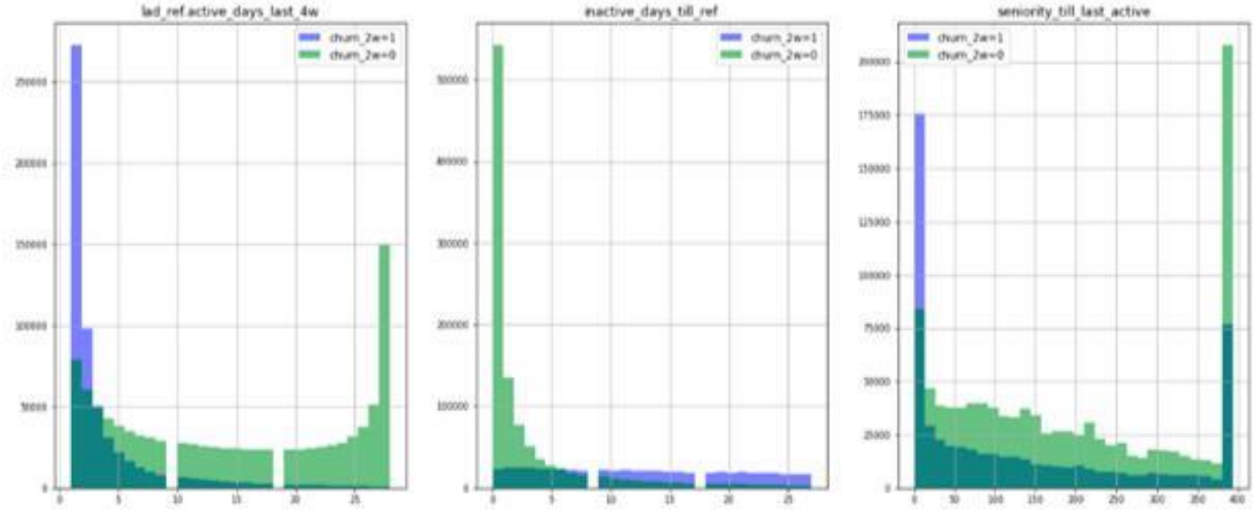
	active_days_last_4w	inactive_days_till_ref	seniority_till_last_active
churn_2w			
0	14.732599	2.792626	192.964393
1	3.438288	12.552919	134.320831



	total_sessions	total_dowload_events	total_cinema_events	total_pin_events	total_headset_events	total_queries	total_clicks
churn_2w							
0	36.169026	76.384340	11.149057	0.548172	0.484660	29.090767	1.580793
1	6.799199	7.232969	2.037874	0.183645	0.071431	6.729588	0.228841

It is clear that people who have been more engaged in the recent period and have not been inactive for too long tend to stay. The average active days last 28 days for people who left is over 11 days less (40% of total) than the average for people who stay. Moreover, the frequency of use of some features also has a huge difference between the two groups. On the other hand, churned users have much longer inactive days from their last active to observed date than the others (12.5 compared with 2.79).

We also can notice that people who have been active for a longer time (longer lifetime) are more likely to stay, however, it is not as clear as other features so I want to dive further into this variable to see the distribution of two groups.



B.2.4. Class distribution by feature variable

We can notice the pattern pretty clear for the first two distributions. For the last one, we can see significant proportion of new users who actually churned compared with the group of users who stayed (rate ~ 2 times => new user retention rate ~ 33%). Besides, there is a substantial amount of senior users (have been active for long time) who left. I suspect this group as occasional users. I leave the deep-dive into users' behaviour as room for improvement.

C. Model creation and evaluation

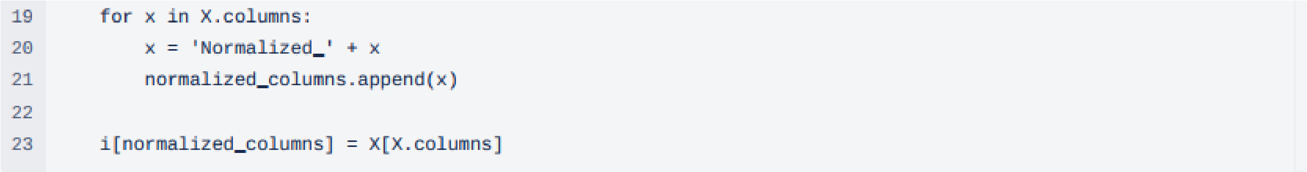
In the scope of this paper, I decided to use two of the most popular and basic classification algorithms: Logistic Regression and XGBoost Classifier (decision-tree-based algorithm).

1. Data preprocessing

To apply Logistic Regression, we first need to scale continuous variables. Otherwise, variables with higher values will be given more importance which affects the accuracy of the model. It is not necessary to rescale for the XGBoost model as it is a decision-tree-based algorithm.

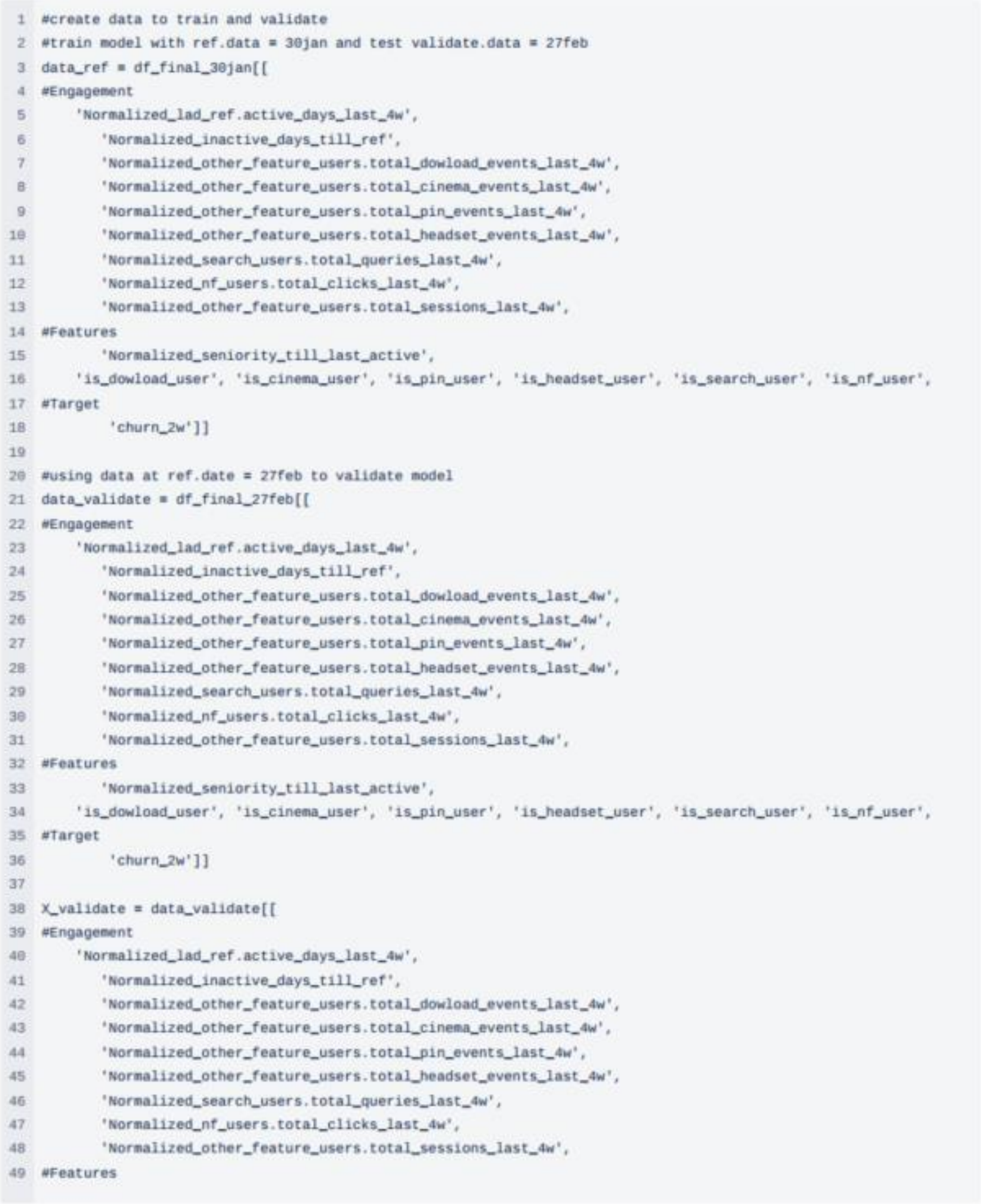
I used MinMaxScaler to rescale all continuous variables within range 0 and 1.

```
df_list = [df_final_30jan, df_final_27feb, df_final_27mar, df_final_24apr, df_final_22may, df_final_19jun, df_fi
#normalize dataset
for i in df_list:
    #Scaling continuous variables:
    sc = MinMaxScaler()
    a = sc.fit_transform(i[[
        'lad_ref.active_days_last_4w', 'inactive_days_till_ref', 'other_feature_users.total_dowload_events_last_
        'other_feature_users.total_cinema_events_last_4w', 'other_feature_users.total_pin_events_last_4w',
        'other_feature_users.total_headset_events_last_4w', 'search_users.total_queries_last_4w',
        'nf_users.total_clicks_last_4w', 'other_feature_users.total_sessions_last_4w', 'seniority_till_last_acti

    X = pd.DataFrame(a, columns=[
        'lad_ref.active_days_last_4w', 'inactive_days_till_ref', 'other_feature_users.total_dowload_events_last_
        'other_feature_users.total_cinema_events_last_4w', 'other_feature_users.total_pin_events_last_4w',
        'other_feature_users.total_headset_events_last_4w', 'search_users.total_queries_last_4w',
        'nf_users.total_clicks_last_4w', 'other_feature_users.total_sessions_last_4w', 'seniority_till_last_acti

    normalized_columns = []
```



```
    for x in X.columns:
        x = 'Normalized_' + x
        normalized_columns.append(x)

    i[normalized_columns] = X[X.columns]
```

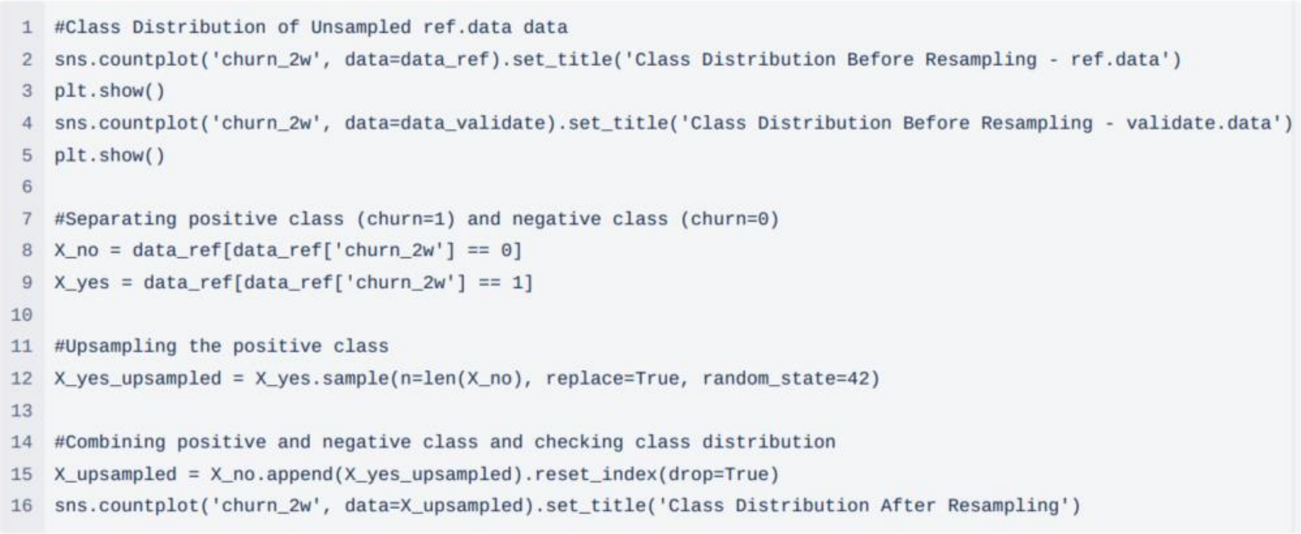
I used data at ref.date = 30jan to train and test the model. To evaluate the performance of our model I used data at ref.date = 27feb. We can try with any data set at whenever ref.date at our preferences.



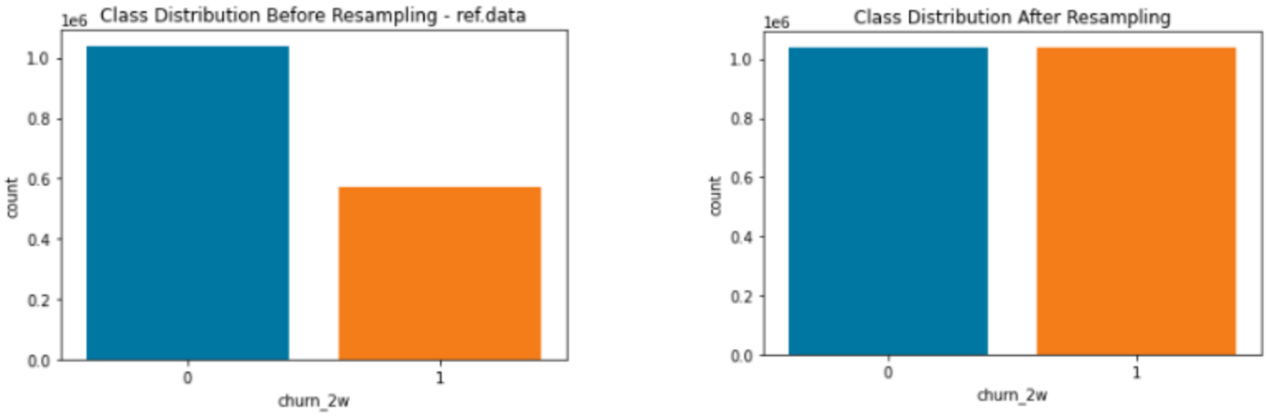
```
#create data to train and validate
#train model with ref.data = 30jan and test validate.data = 27feb
data_ref = df_final_30jan[[
#Engagement
    'Normalized_lad_ref.active_days_last_4w',
       'Normalized_inactive_days_till_ref',
       'Normalized_other_feature_users.total_dowload_events_last_4w',
       'Normalized_other_feature_users.total_cinema_events_last_4w',
       'Normalized_other_feature_users.total_pin_events_last_4w',
       'Normalized_other_feature_users.total_headset_events_last_4w',
       'Normalized_search_users.total_queries_last_4w',
       'Normalized_nf_users.total_clicks_last_4w',
       'Normalized_other_feature_users.total_sessions_last_4w',
#Features
       'Normalized_seniority_till_last_active',
    'is_dowload_user', 'is_cinema_user', 'is_pin_user', 'is_headset_user', 'is_search_user', 'is_nf_user',
#Target
        'churn_2w']]

#using data at ref.date = 27feb to validate model
data_validate = df_final_27feb[[
#Engagement
    'Normalized_lad_ref.active_days_last_4w',
       'Normalized_inactive_days_till_ref',
       'Normalized_other_feature_users.total_dowload_events_last_4w',
       'Normalized_other_feature_users.total_cinema_events_last_4w',
       'Normalized_other_feature_users.total_pin_events_last_4w',
       'Normalized_other_feature_users.total_headset_events_last_4w',
       'Normalized_search_users.total_queries_last_4w',
       'Normalized_nf_users.total_clicks_last_4w',
       'Normalized_other_feature_users.total_sessions_last_4w',
#Features
       'Normalized_seniority_till_last_active',
    'is_dowload_user', 'is_cinema_user', 'is_pin_user', 'is_headset_user', 'is_search_user', 'is_nf_user',
#Target
        'churn_2w']]

X_validate = data_validate[[
#Engagement
    'Normalized_lad_ref.active_days_last_4w',
       'Normalized_inactive_days_till_ref',
       'Normalized_other_feature_users.total_dowload_events_last_4w',
       'Normalized_other_feature_users.total_cinema_events_last_4w',
       'Normalized_other_feature_users.total_pin_events_last_4w',
       'Normalized_other_feature_users.total_headset_events_last_4w',
       'Normalized_search_users.total_queries_last_4w',
       'Normalized_nf_users.total_clicks_last_4w',
       'Normalized_other_feature_users.total_sessions_last_4w',
#Features
```

```
        'Normalized_seniority_till_last_active',
     'is_dowload_user', 'is_cinema_user', 'is_pin_user', 'is_headset_user', 'is_search_user', 'is_nf_user'
]]

Y_validate = data_validate[['churn_2w']]
```

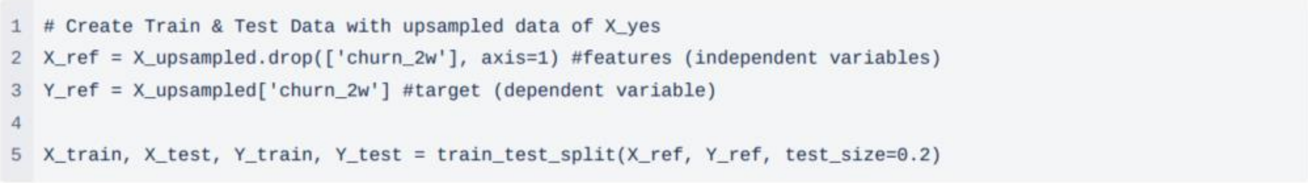
The next step is to resample the dataset since target variables with imbalanced class distribution are not desired for machine learning models. I will use upsampling which means increasing the number of samples of the class with less samples by randomly selecting rows from it.



```
#Class Distribution of Unsampled ref.data data
sns.countplot('churn_2w', data=data_ref).set_title('Class Distribution Before Resampling - ref.data')
plt.show()
sns.countplot('churn_2w', data=data_validate).set_title('Class Distribution Before Resampling - validate.data')
plt.show()

#Separating positive class (churn=1) and negative class (churn=0)
X_no = data_ref[data_ref['churn_2w'] == 0]
X_yes = data_ref[data_ref['churn_2w'] == 1]

#Upsampling the positive class
X_yes_upsampled = X_yes.sample(n=len(X_no), replace=True, random_state=42)

#Combining positive and negative class and checking class distribution
X_upsampled = X_no.append(X_yes_upsampled).reset_index(drop=True)
sns.countplot('churn_2w', data=X_upsampled).set_title('Class Distribution After Resampling')
```



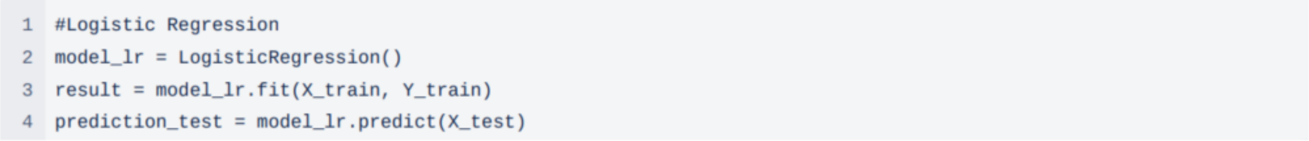
We then need to divide the dataset into training and test subsets so that we are able to measure the performance of our model on new, previously unseen examples.



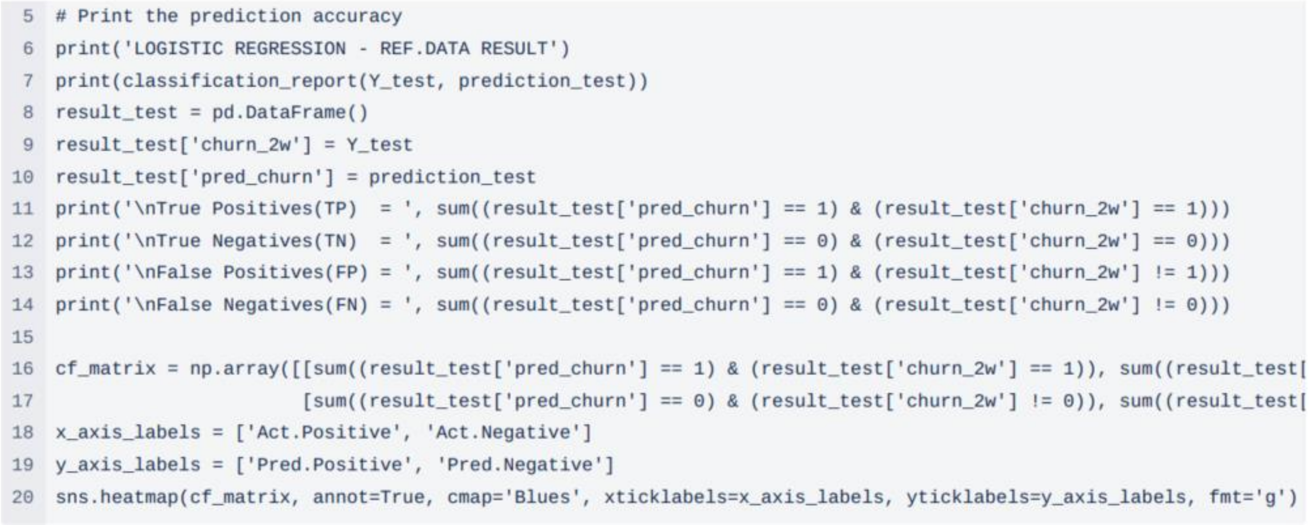
```
# Create Train & Test Data with upsampled data of X_yes
X_ref = X_upsampled.drop(['churn_2w'], axis=1) #features (independent variables)
Y_ref = X_upsampled['churn_2w'] #target (dependent variable)

X_train, X_test, Y_train, Y_test = train_test_split(X_ref, Y_ref, test_size=0.2)
```

1. Apply model and evaluation

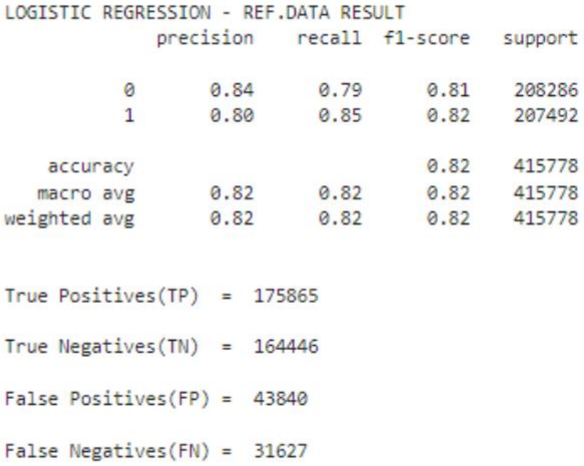
- **Logistic Regression**



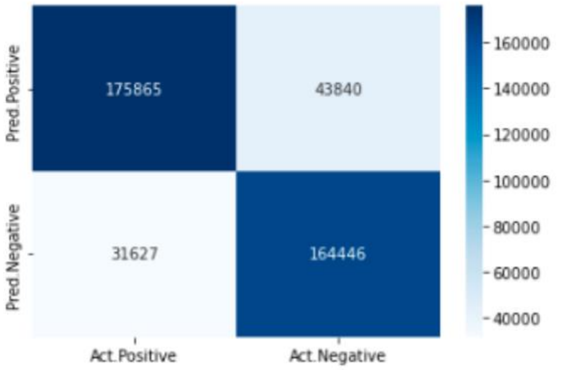
```
#Logistic Regression
model_lr = LogisticRegression()
result = model_lr.fit(X_train, Y_train)
prediction_test = model_lr.predict(X_test)
```



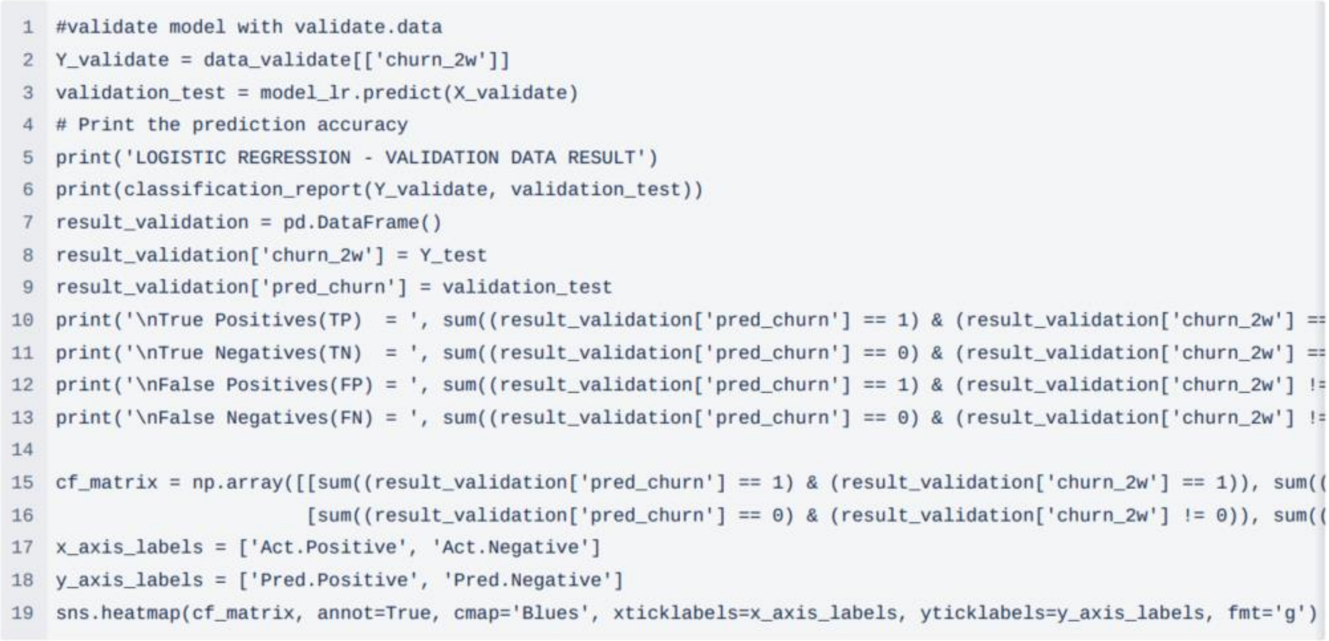
```
# Print the prediction accuracy
print('LOGISTIC REGRESSION - REF.DATA RESULT')
print(classification_report(Y_test, prediction_test))
result_test = pd.DataFrame()
result_test['churn_2w'] = Y_test
result_test['pred_churn'] = prediction_test
print('\nTrue Positives(TP)  = ', sum((result_test['pred_churn'] == 1) & (result_test['churn_2w'] == 1)))
print('\nTrue Negatives(TN)  = ', sum((result_test['pred_churn'] == 0) & (result_test['churn_2w'] == 0)))
print('\nFalse Positives(FP) = ', sum((result_test['pred_churn'] == 1) & (result_test['churn_2w'] != 1)))
print('\nFalse Negatives(FN) = ', sum((result_test['pred_churn'] == 0) & (result_test['churn_2w'] != 0)))

cf_matrix = np.array([[sum((result_test['pred_churn'] == 1) & (result_test['churn_2w'] == 1)), sum((result_test[
                      [sum((result_test['pred_churn'] == 0) & (result_test['churn_2w'] != 0)), sum((result_test[
x_axis_labels = ['Act.Positive', 'Act.Negative']
y_axis_labels = ['Pred.Positive', 'Pred.Negative']
sns.heatmap(cf_matrix, annot=True, cmap='Blues', xticklabels=x_axis_labels, yticklabels=y_axis_labels, fmt='g')
```



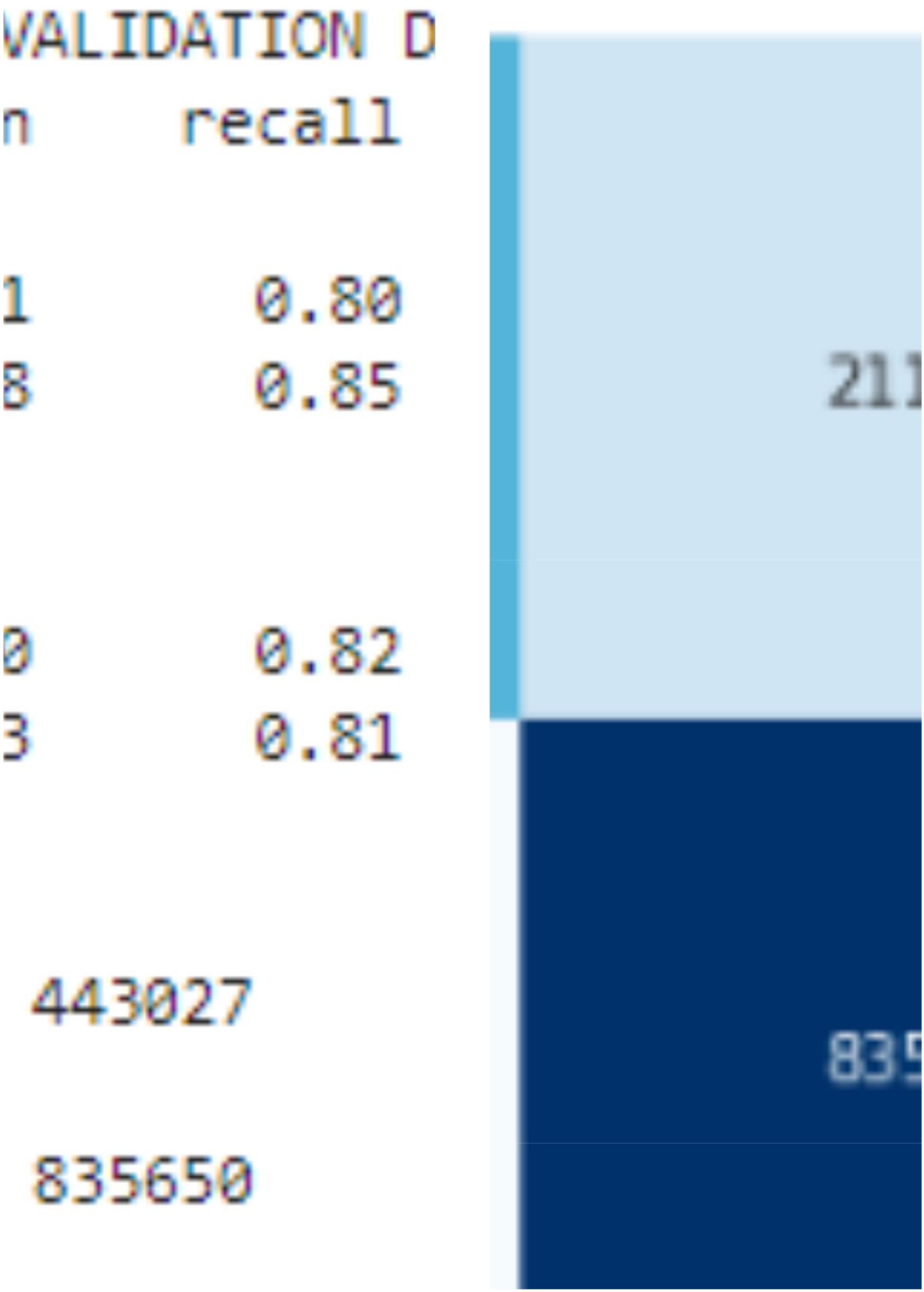
```
LOGISTIC REGRESSION - REF.DATA RESULT
              precision    recall  f1-score   support

           0       0.84      0.79      0.81    208286
           1       0.80      0.85      0.82    207492

    accuracy                           0.82    415778
   macro avg       0.82      0.82      0.82    415778
weighted avg       0.82      0.82      0.82    415778


True Positives(TP)  =  175865

True Negatives(TN)  =  164446

False Positives(FP) =  43840

False Negatives(FN) =  31627
```



The model achieved 82% accuracy on test set. The result is quite optimistic since both precision and recall are over 80%. However, this is just the result of resampled data, we should evaluate this model with the actual dataset to have close look at its performance.



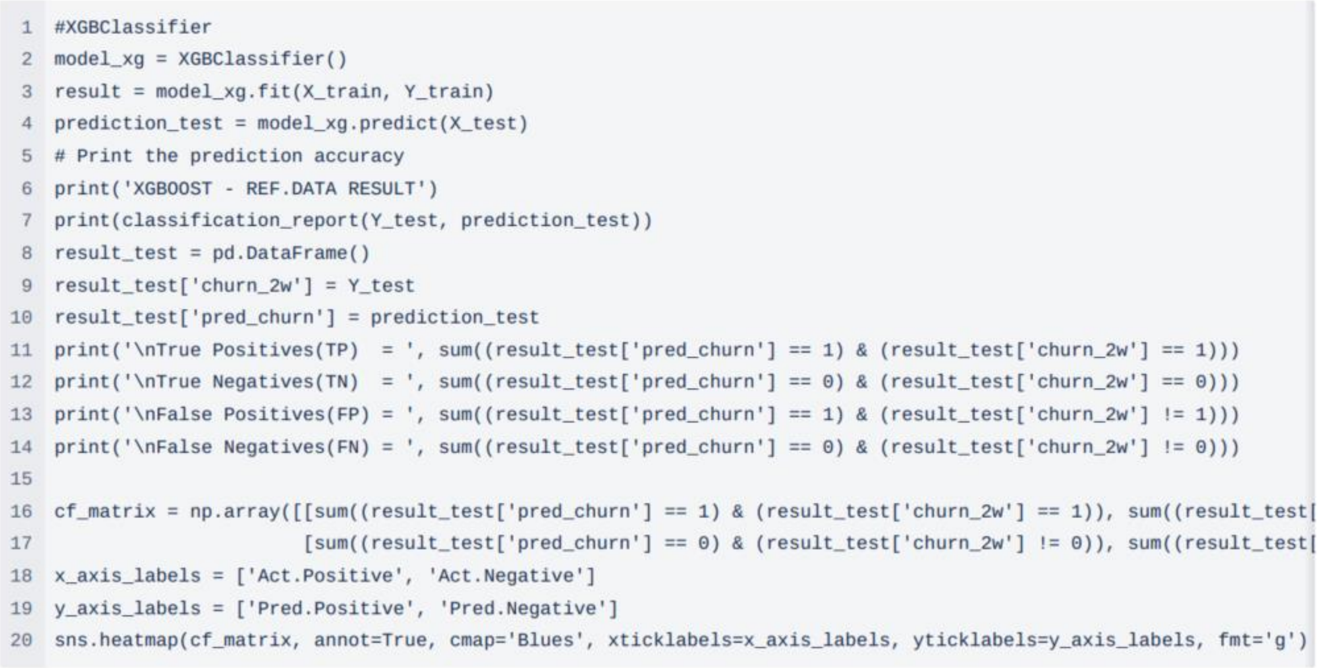
```
#validate model with validate.data
Y_validate = data_validate[['churn_2w']]
validation_test = model_lr.predict(X_validate)
# Print the prediction accuracy
print('LOGISTIC REGRESSION - VALIDATION DATA RESULT')
print(classification_report(Y_validate, validation_test))
result_validation = pd.DataFrame()
result_validation['churn_2w'] = Y_test
result_validation['pred_churn'] = validation_test
print('\nTrue Positives(TP)  = ', sum((result_validation['pred_churn'] == 1) & (result_validation['churn_2w'] ==
print('\nTrue Negatives(TN)  = ', sum((result_validation['pred_churn'] == 0) & (result_validation['churn_2w'] ==
print('\nFalse Positives(FP) = ', sum((result_validation['pred_churn'] == 1) & (result_validation['churn_2w'] !=
print('\nFalse Negatives(FN) = ', sum((result_validation['pred_churn'] == 0) & (result_validation['churn_2w'] !=

cf_matrix = np.array([[sum((result_validation['pred_churn'] == 1) & (result_validation['churn_2w'] == 1)), sum((
                      [sum((result_validation['pred_churn'] == 0) & (result_validation['churn_2w'] != 0)), sum((
x_axis_labels = ['Act.Positive', 'Act.Negative']
y_axis_labels = ['Pred.Positive', 'Pred.Negative']
sns.heatmap(cf_matrix, annot=True, cmap='Blues', xticklabels=x_axis_labels, yticklabels=y_axis_labels, fmt='g')
```



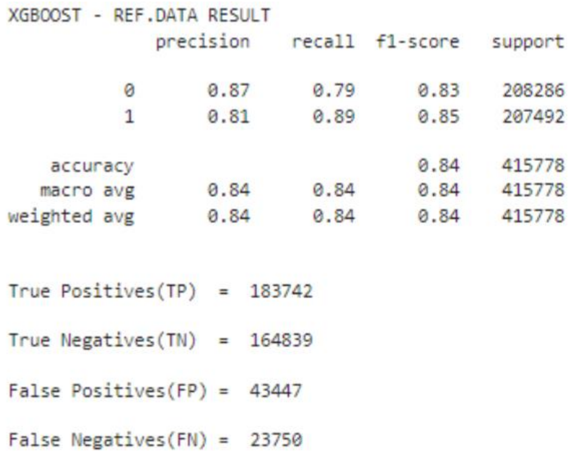
```
VALIDATION D
n      recall

1        0.80
8        0.85


0        0.82
3        0.81



 443027


 835650
```

On the validation set, the result is slightly worse than the test set due to its imbalance. Although recall metric is still good (~85%), precision is quite low (~68%) due to high False Positive prediction. We'll later focus on it to find reasons why our model fails to predict this group.

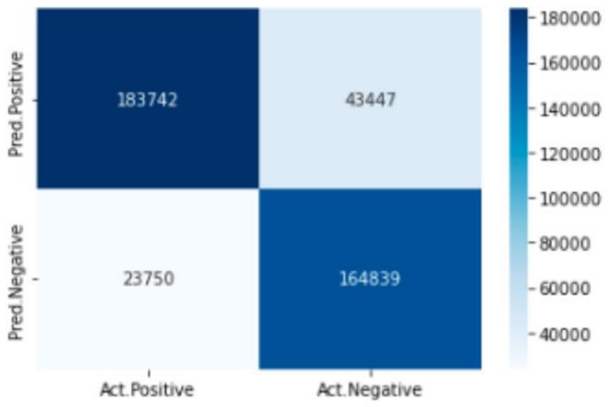
XGBoost Classifier



```
#XGBClassifier
model_xg = XGBClassifier()
result = model_xg.fit(X_train, Y_train)
prediction_test = model_xg.predict(X_test)
# Print the prediction accuracy
print('XGBOOST - REF.DATA RESULT')
print(classification_report(Y_test, prediction_test))
result_test = pd.DataFrame()
result_test['churn_2w'] = Y_test
result_test['pred_churn'] = prediction_test
print('\nTrue Positives(TP)  = ', sum((result_test['pred_churn'] == 1) & (result_test['churn_2w'] == 1)))
print('\nTrue Negatives(TN)  = ', sum((result_test['pred_churn'] == 0) & (result_test['churn_2w'] == 0)))
print('\nFalse Positives(FP) = ', sum((result_test['pred_churn'] == 1) & (result_test['churn_2w'] != 1)))
print('\nFalse Negatives(FN) = ', sum((result_test['pred_churn'] == 0) & (result_test['churn_2w'] != 0)))

cf_matrix = np.array([[sum((result_test['pred_churn'] == 1) & (result_test['churn_2w'] == 1)), sum((result_test[
                      [sum((result_test['pred_churn'] == 0) & (result_test['churn_2w'] != 0)), sum((result_test[
x_axis_labels = ['Act.Positive', 'Act.Negative']
y_axis_labels = ['Pred.Positive', 'Pred.Negative']
sns.heatmap(cf_matrix, annot=True, cmap='Blues', xticklabels=x_axis_labels, yticklabels=y_axis_labels, fmt='g')
```

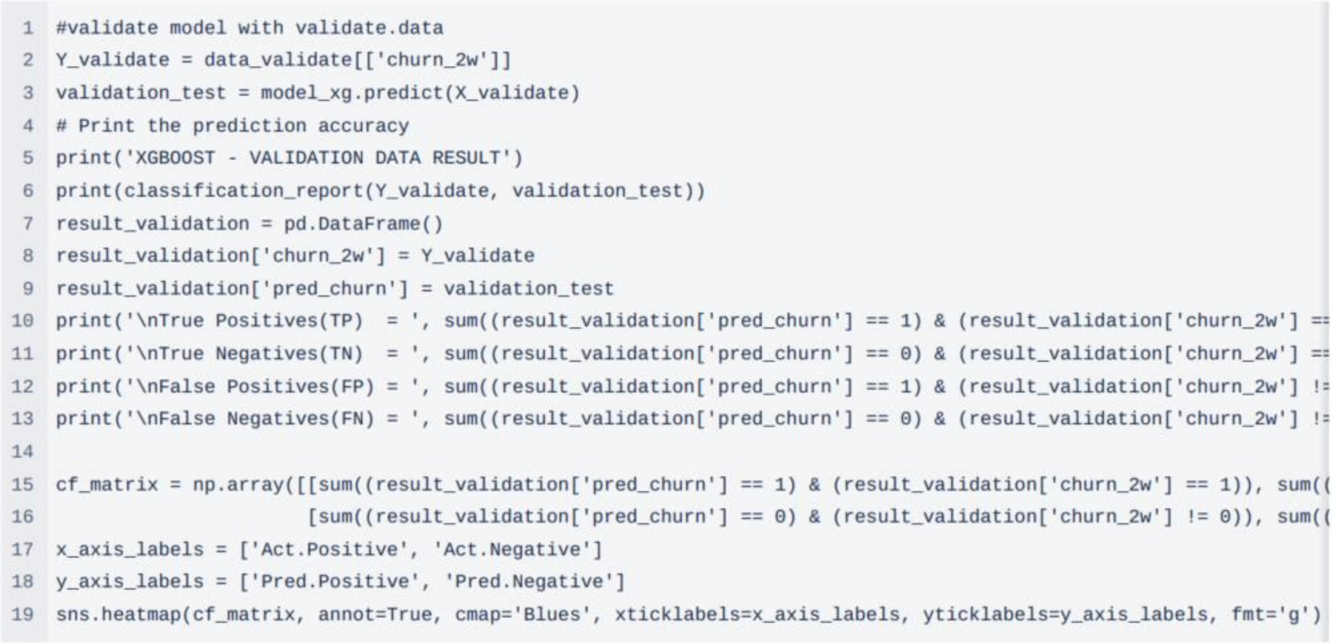


```
XGBOOST - REF.DATA RESULT
              precision    recall  f1-score   support

           0       0.87      0.79      0.83    208286
           1       0.81      0.89      0.85    207492

    accuracy                           0.84    415778
   macro avg       0.84      0.84      0.84    415778
weighted avg       0.84      0.84      0.84    415778


True Positives(TP)  =  183742

True Negatives(TN)  =  164839

False Positives(FP) =  43447

False Negatives(FN) =  23750
```

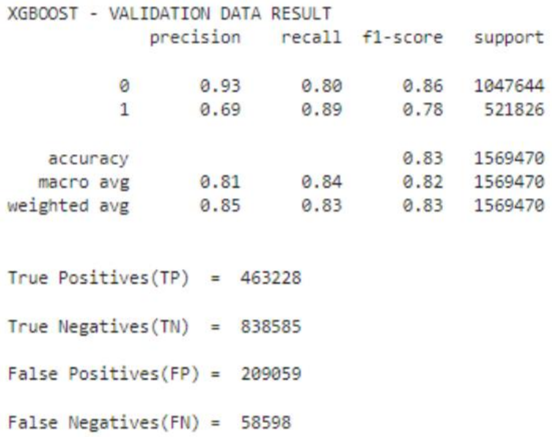


This model achieved better results than the former on test dataset. Accuracy is improved from 82% to 84%. Both precision and recall are improved more or less.

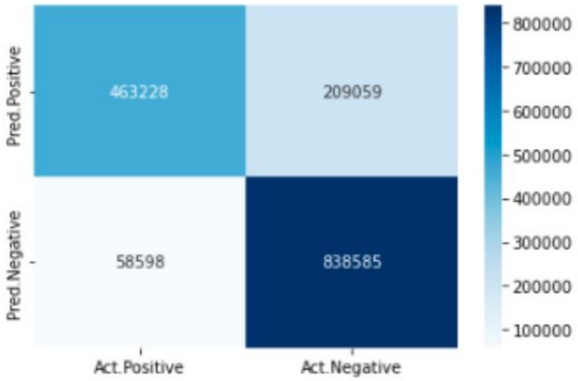
Then we again evaluate this model with the validation dataset.



```
#validate model with validate.data
Y_validate = data_validate[['churn_2w']]
validation_test = model_xg.predict(X_validate)
# Print the prediction accuracy
print('XGBOOST - VALIDATION DATA RESULT')
print(classification_report(Y_validate, validation_test))
result_validation = pd.DataFrame()
result_validation['churn_2w'] = Y_validate
result_validation['pred_churn'] = validation_test
print('\nTrue Positives(TP)  = ', sum((result_validation['pred_churn'] == 1) & (result_validation['churn_2w'] ==
print('\nTrue Negatives(TN)  = ', sum((result_validation['pred_churn'] == 0) & (result_validation['churn_2w'] ==
print('\nFalse Positives(FP) = ', sum((result_validation['pred_churn'] == 1) & (result_validation['churn_2w'] !=
print('\nFalse Negatives(FN) = ', sum((result_validation['pred_churn'] == 0) & (result_validation['churn_2w'] !=

cf_matrix = np.array([[sum((result_validation['pred_churn'] == 1) & (result_validation['churn_2w'] == 1)), sum((
                      [sum((result_validation['pred_churn'] == 0) & (result_validation['churn_2w'] != 0)), sum((
x_axis_labels = ['Act.Positive', 'Act.Negative']
y_axis_labels = ['Pred.Positive', 'Pred.Negative']
sns.heatmap(cf_matrix, annot=True, cmap='Blues', xticklabels=x_axis_labels, yticklabels=y_axis_labels, fmt='g')
```



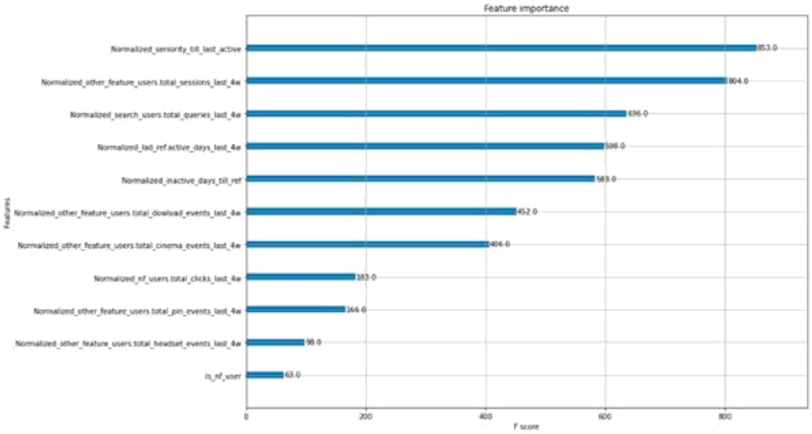
```
XGBOOST - VALIDATION DATA RESULT
              precision    recall  f1-score   support

           0       0.93      0.80      0.86   1047644
           1       0.69      0.89      0.78    521826

    accuracy                           0.83   1569470
   macro avg       0.81      0.84      0.82   1569470
weighted avg       0.85      0.83      0.83   1569470


True Positives(TP)  =  463228

True Negatives(TN)  =  838585

False Positives(FP) =  209059

False Negatives(FN) =  58598
```



We again encounter the same issue as the former model due to imbalanced dataset. The result is quite good though as we can detect 89% of total actual churned users.

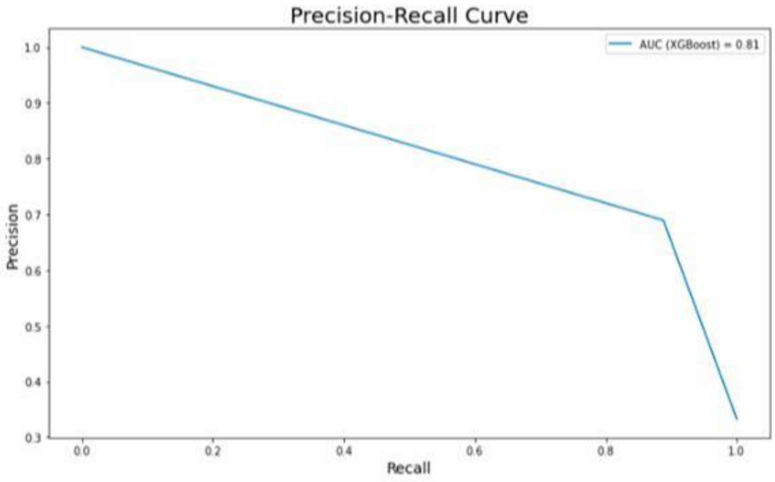
3. Result Analysis

Since XGBoost did a better job, we will use it as the base model and dive deeper into the result. We can see below top 5 important features including variables: seniority_till_last_active, total_sessions_last_4w, total_queries_last_4w, active_days_last_4w and inactive_days_till_ref.



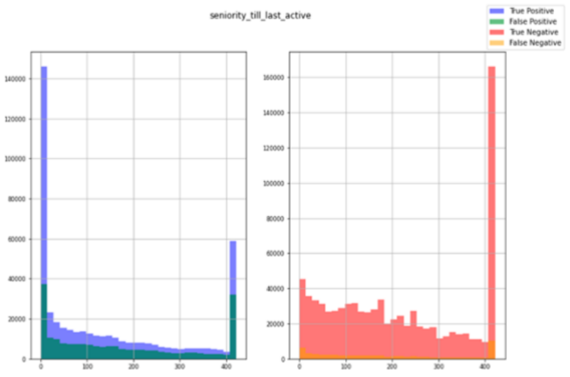
C.3.1. Feature Importance

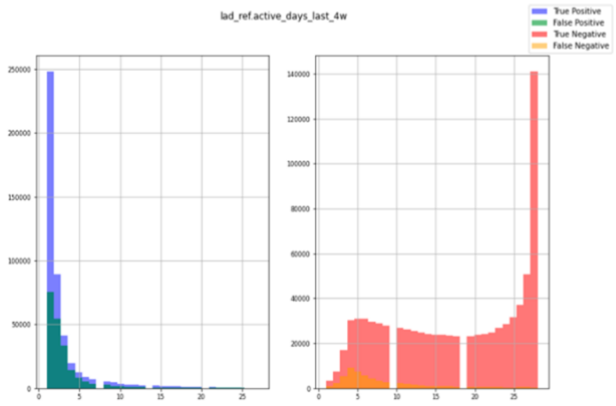
Below is the precision-recall curve which shows the trade-off between precision and recall. We can see there is a sharp decline at the end which indicate the model finds it difficult to classify a group of churned users correctly



C.3.2. Precision-Recall trade off

We can see most the False Positive predictions are active for less than 7 days in the last 28 days from ref.date. However, not all of them are new users but also senior users. I suspect this group as occasional users and the model find it hard to classify them since occasional users have quite similar behavior as a potential churning user.





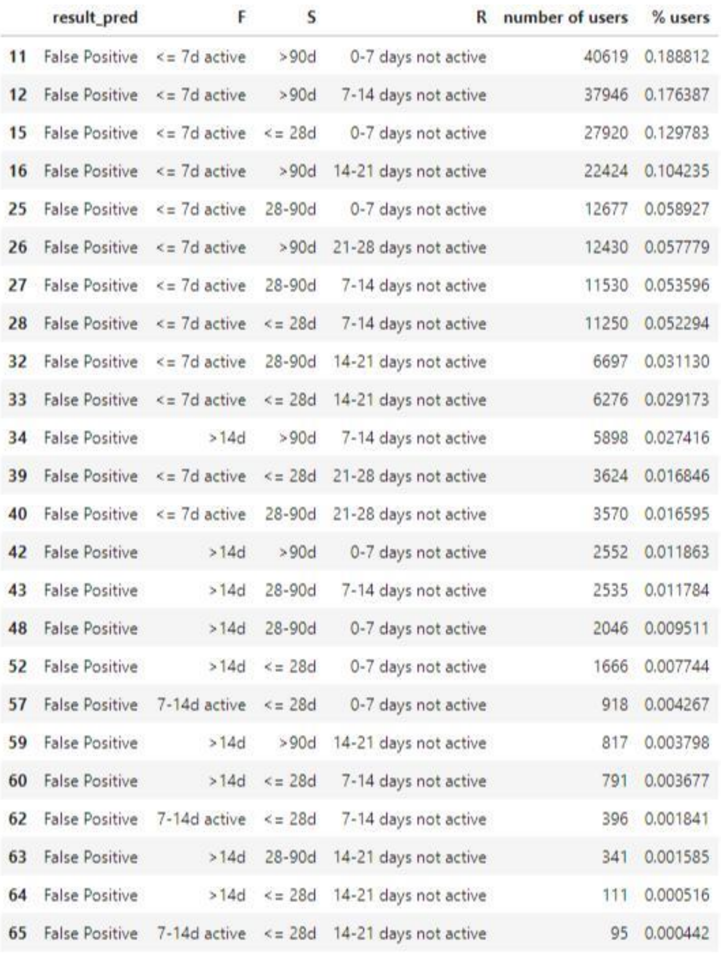
3. User segmentation and result analysis

In this section, we will try to categorize our users using RFS analysis - my modified version of RFM and Kmeans clustering.

RFS stands for Recency, Frequency, and Seniority value, each corresponding to some key customer trait. These RFS metrics are important indicators to detect a churned user because frequency and seniority value directly affects a user’s lifetime, and recency affects retention, a measure of engagement.

I classify each user into each recency (inactive 0-7 days, 7-14 days, 14-21 days and 21-28 days), frequency (active 0-7 days, 7-14 days and >14 days in total 28 days) and seniority group (first active 0-28 days, 28-90 days, >90 days)

We have the analysis group by result predicted from model as below (I only focus on False Positive results as that what we want to improve):



	result_pred	F	S	R	number of users	% users
11	False Positive	<= 7d active	>90d	0-7 days not active	40619	0.188812
12	False Positive	<= 7d active	>90d	7-14 days not active	37946	0.176387
15	False Positive	<= 7d active	<= 28d	0-7 days not active	27920	0.129783
16	False Positive	<= 7d active	>90d	14-21 days not active	22424	0.104235
25	False Positive	<= 7d active	28-90d	0-7 days not active	12677	0.058927
26	False Positive	<= 7d active	>90d	21-28 days not active	12430	0.057779
27	False Positive	<= 7d active	28-90d	7-14 days not active	11530	0.053596
28	False Positive	<= 7d active	<= 28d	7-14 days not active	11250	0.052294
32	False Positive	<= 7d active	28-90d	14-21 days not active	6697	0.031130
33	False Positive	<= 7d active	<= 28d	14-21 days not active	6276	0.029173
34	False Positive	>14d	>90d	7-14 days not active	5898	0.027416
39	False Positive	<= 7d active	<= 28d	21-28 days not active	3624	0.016846
40	False Positive	<= 7d active	28-90d	21-28 days not active	3570	0.016595
42	False Positive	>14d	>90d	0-7 days not active	2552	0.011863
43	False Positive	>14d	28-90d	7-14 days not active	2535	0.011784
48	False Positive	>14d	28-90d	0-7 days not active	2046	0.009511
52	False Positive	>14d	<= 28d	0-7 days not active	1666	0.007744
57	False Positive	7-14d active	<= 28d	0-7 days not active	918	0.004267
59	False Positive	>14d	>90d	14-21 days not active	817	0.003798
60	False Positive	>14d	<= 28d	7-14 days not active	791	0.003677
62	False Positive	7-14d active	<= 28d	7-14 days not active	396	0.001841
63	False Positive	>14d	28-90d	14-21 days not active	341	0.001585
64	False Positive	>14d	<= 28d	14-21 days not active	111	0.000516
65	False Positive	7-14d active	<= 28d	14-21 days not active	95	0.000442

It is easily noticed that most of FP predictions are for group of users who have frequency value <= 7 (which means they are active less than 7 days in total 28 days). Moreover most of them are quite senior users (first active date is more than 90 days). This result is exactly what we expect that our model failed to predict occasional users who have similar behaviors to potential churning users.

Besides, around 13% of our FP predictions are new users (first active date less than 28 days from observed point). These new users might be occasional users (which again our model failed to predict) or not (which needs further analysis).

To sum up, I think we can still count on predictions from our model. Excluding FP results the model did pretty good job. For the false positive results, I think it doesn’t affect much as we can also have same actions as reactivating churned users to convert these occasional users into loyal ones.