# A PROJECT REPORT ON

# Customer churn Prediction

A project report submitted in fulfillment for the Diploma Degree in AI & ML Under

Applied Roots with University of Hyderabad


Project submitted by

**Roshan R B**

Under the Guidance of

Mentor: Harichandhana K


**University of Hyderabad**

# ***Declaration of Authorship***

We hereby declare that this thesis titled "Predict if any subscriber will churn or not" and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

**Name: Roshan R B**

**Thesis Title:** Predict if any subscriber will churn or not

**CERTIFICATE OF RECOMMENDATION**

We hereby recommend that the thesis entitled “Predict if any subscriber will churn or not” prepared under my supervision and guidance by Roshan R B be accepted in fulfilment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with University of Hyderabad. The project, in our opinion, is worthy for its acceptance.

_____

Mentor: Harichandhana K

**Under Applied roots with**

**University of Hyderabad**

## ACKNOWLEDGEMENT

Every project big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success. I, Roshan R B student of applied roots, is extremely grateful to mentors for the confidence bestowed in me and entrusting my project entitled “Predict if any subscriber will churn or not” with special reference.

At this juncture, I express my sincere thanks to Mentor: Harichandhana K of applied roots for making the resources available at right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

**Name: Roshan R B**

## Contents

# Predict if any subscriber will churn or not

**Abstract:**

This is a project wherein we have to predict whether a user will churn after his/her subscription expires. A forecast has to be made whether a user makes a new service subscription transaction within 30 days after the current membership expiration date. The subscription is provided by the company KKBOX, Asia's leading music streaming service, holding the world's most comprehensive Asia Pop-Music library with over 30 million tracks, to its users.

## 1. Business Problem:

Customers are very important for any company. And every company wants to provide good services to its customers and earn their trust. Moreover, if the customer is not satisfied, they tend to leave. The growth of the company decreases if the number of unhappy customers increases. Therefore a company can address this issue by using **Churn Prediction**. If the company manages to keep a low churning rate, then the company's revenue and profits will definitely increase.

For a subscription business model, predicting customer churn accurately is critical to long-term success. Even minor variations in churn can drastically affect profits.

As the title of the problem states, churn prediction is typically a binary classification problem to determine whether a customer will churn or not churn. So in order to address this problem, we have to first understand "What is Churn?" ***Churn*** *is basically used to represent the number of the paid customer or subscribers who terminated the use of the product or services of a company in a particular time period.*

Now, let us understand, "Why is this churn rate useful?" *As we know, a high churn rate leads to decline in the growth rate of the company. And according to many types of research which claim that "if any company retains just 5% of its existing customers, then it would result in 25% to 95% increase of profit" .*

Now, we are able to understand the basics of churn prediction, let us further understand the business problem. "Based on the data available, the model must predict whether the paid subscriber will Churn after his/her subscription expires or not". The criteria of Churn is, if there is no new valid subscription within 30 days after the current membership/subscription expires.

## 2. Business Objectives and Constraints:

- Computation time is not a considered factor, therefore any number of features can be created from the base dataset
- Need to assign a penalty for each misclassification.
- Interpret the outcome or derived results, so that the company can proceed further for improvisation.

## 3. Data:

KKBox has provided 2 versions of data. The version 1 contains the data till February 2017, version 2 data, which contains the data till March 2017. So, the given task is to predict customer churn for the month of April 2017.

For predicting customer churn in the month of april, we need to use the version 2 data. There are 4 major comma separated files(csv's) given.

- **train_v2.csv —** This file contains 970960 number of rows and just 2 features, msno - (unique id) and is_churn - (class labels).
- **transactions_v2.csv —** The file contains the transactions data for all the users. This file is containing 1431009 number of rows and 9 features, msno(unique id), payment_method_id, payment_plan_days, plan_list_price, actual_amount_paid, is_auto_renew, transaction_date, membership_expire_date and is_cancel.
- **user_logs_v2.csv —** The file contains the day to day user data(user activity) for all the users. This file is containing 18396362 number of rows and 9 features, msno(unique id), date, num_25, num_50, num_75, num_985, num_100, num_unq and total_secs.
- **members_v3.csv —** The file contains basic customer data. This file is containing 6769473 number of rows and 6 features, msno(unique id), city, bd(age), gender, registered_via and registration_init_time.

This Data is downloaded from this link: https://www.kaggle.com/c/kkbox-churn-prediction-challenge/data

**4. Evaluation Metric:**

After analysing the data, further, we have to determine Performance Metric which is Log-loss. Since the problem is a binary classification, we can use Binary log-loss. Also, since one of our key constraints for this problem is “to assign penalty for every misclassification”, so log-loss is preferred in this situation. In order to represent binary log-loss mathematically, we can use the below eqaution

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))$$

binary log-loss

In the above mathematical formulation, yi’s are the actual class labels(ground truth) and pi’s are the predicted probabilities. The log-loss tends to lie in between 0 and positive infinity. But the value close to 0 is consider as a good value. So with the log-loss as performance metric our main goal is to keep it as close as 0.

**Overall goal >>**
Use historical subscriber data from KKbox music subscription service to predict future customer churn in a given test dataset.

**Target >>**
Likelihood of the customer churn in the upcoming month.Defined as true if the customer does not renew subscription within the prediction month( the month following the current month) Therefore members whose subscription ending in the current month only were considered.

**Evaluation >>**
Log loss

**Dataset >>**
KKBox subscription data stored in 3 datasets

**Transaction:** Historical payment and subscription info, including renewal and cancellation date and pricing data.

**Userlogs:** Daily subscriber activity such as the number of unique songs played each day.

**Member data:** Demographic information about each subscriber such as birthdate, gender and member join date.

## Exploratory Data Analysis

Importing the required libraries to perform EDA:

```
import pandas as pd
import os
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

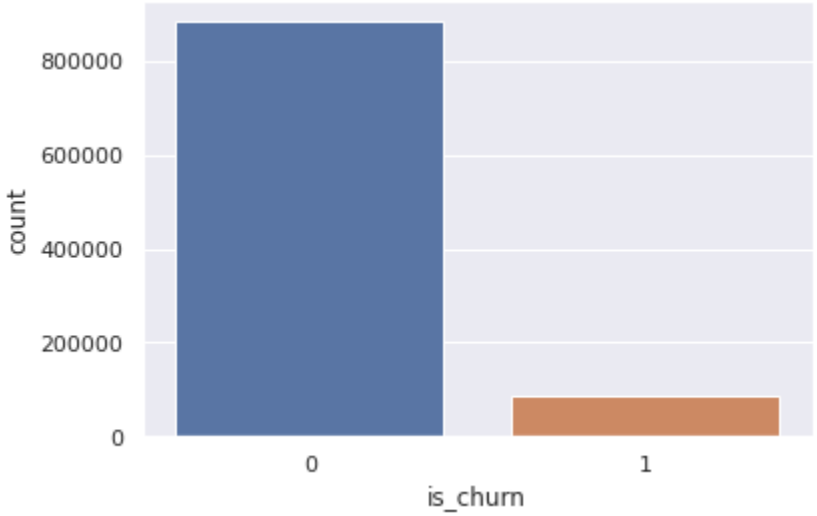
1. First we will read the **train_v2.csv** file and check what fields are present

```
#Read train data
train_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/train_v2.csv')
```

| | msno | is_churn |
|---|---|---|
| 0 | ugx0CjOMzazClkFzU2xasmDZaoIqOUAZPsH1q0teWCg= | 1 |
| 1 | f/NmvEzHfhINFEYZTR05prUdr+E+3+oewvweYz9cCQE= | 1 |
| 2 | zLo9f73nGGT1p21ltZC3ChiRnAVvgibMyazbCxvWPcg= | 1 |
| 3 | 8iF/+8HY8lJKFrTc7iR9ZYGCG2Ecrogbc2Vy5YhsfhQ= | 1 |
| 4 | K6fja4+jmoZ5xG6BypqX80Uw/XKpMgrEMdG2edFOxnA= | 1 |

> The given dataset has customer id and the target variable

**# To check the distribution of all members who churned and who did not churn, lets plot total count vs is_churn**




Though it is good to have positive points(members who did not churn) for any business, given data is imbalanced not all performance metrics works well in this situation.

2. Next we shall read the **members** dataset to know the details about the users

```
#Read members data
members = pd.read_csv('/content/drive/MyDrive/KKbox_data/members_v3.csv',parse_dates=['registration_init_time'])
```

> Using parse_dates to read dates in the csv file.

| | msno | city | bd | gender | registered_via | registration_init_time |
|---|---|---|---|---|---|---|
| **0** | Rb9UwLQTrxzBVwCB6+bCcSQWZ9JiNLC9dXtM1oEsZA8= | 1 | 0 | NaN | 11 | 2011-09-11 |
| **1** | +tJonkh+O1CA796Fm5X60UMOtB6POHAwPjbTRVl/EuU= | 1 | 0 | NaN | 7 | 2011-09-14 |
| **2** | cV358ssn7a0f7jZOwGNWS07wCKVqxyiImJUX6xcIwKw= | 1 | 0 | NaN | 11 | 2011-09-15 |
| **3** | 9bzDeJP6sQodK73K5CBlJ6fgIQzPeLnRl0p5B77XP+g= | 1 | 0 | NaN | 11 | 2011-09-15 |
| **4** | WFLY3s7z4EZsieHCt63XrsdtfTEmJ+2PnnKLH5GY4Tk= | 6 | 32 | female | 9 | 2011-09-15 |

- There are NaN Values in gender
- Gender is in string format, need to be encoded

Lets further take a look at the description of each field

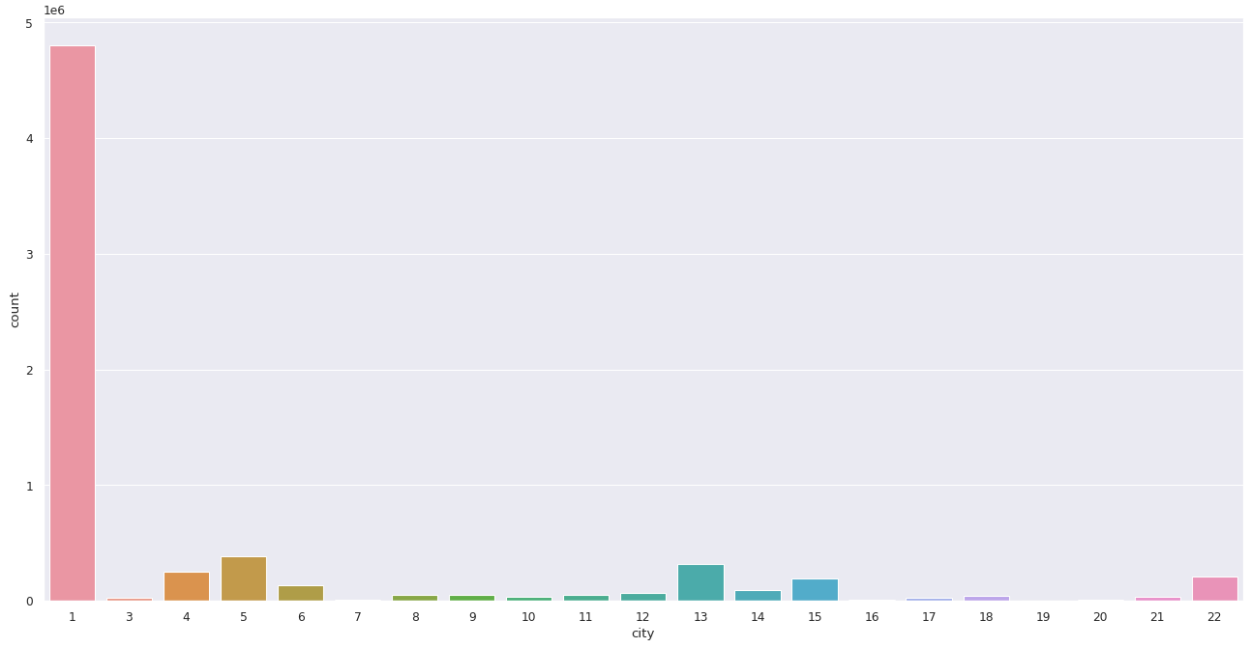
| | city | bd | registered_via |
|---|---|---|---|
| **count** | 6769473.000000 | 6769473.000000 | 6769473.000000 |
| **mean** | 3.847358 | 9.795794 | 5.253069 |
| **std** | 5.478359 | 17.925900 | 2.361398 |
| **min** | 1.000000 | -7168.000000 | -1.000000 |
| **25%** | 1.000000 | 0.000000 | 4.000000 |
| **50%** | 1.000000 | 0.000000 | 4.000000 |
| **75%** | 4.000000 | 21.000000 | 7.000000 |
| **max** | 22.000000 | 2016.000000 | 19.000000 |

*bd implies age, which clearly shows that some values are negative and some values are very high which means outliers are present

*>>> Now, lets explore the members dataset in depth*

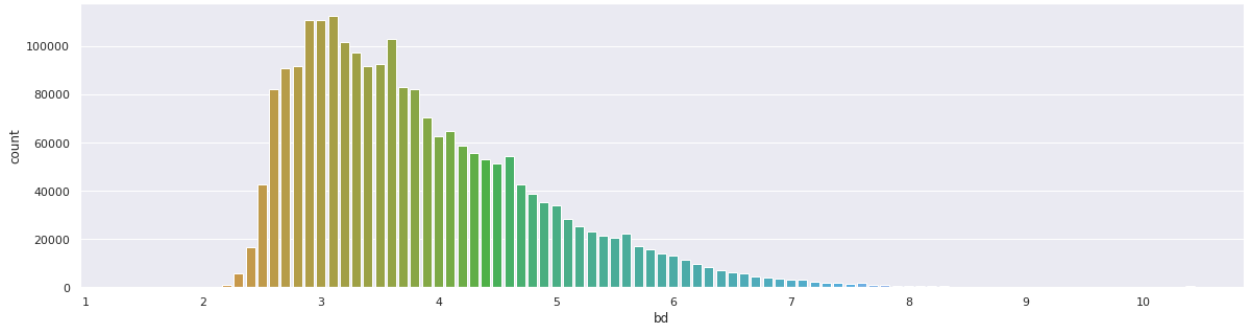
There are total 21 distinct cities in member dataset

**# To check the distribution of users from different cities**



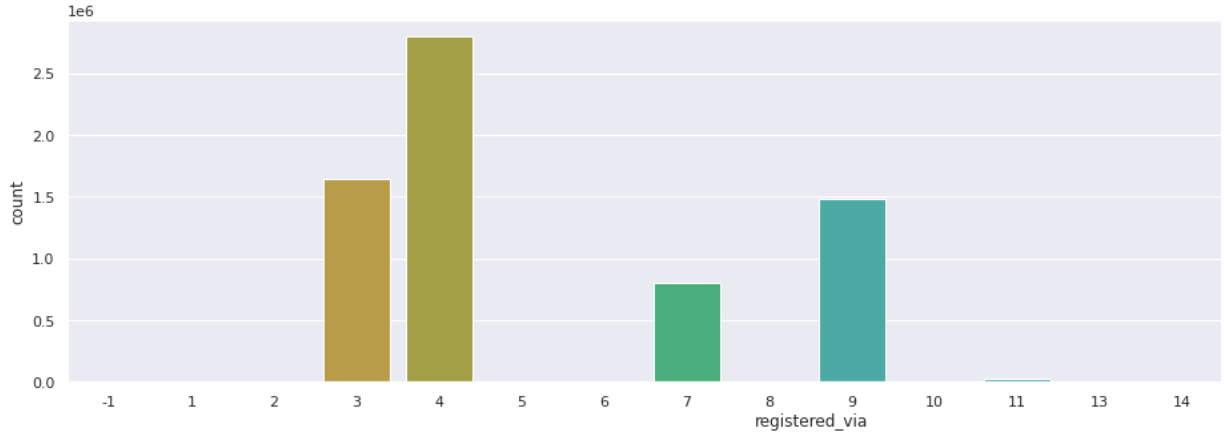

Clearly, we can see city 1 is dominant here!

**# To check the distribution of age groups among the users**




To understand this plot we need to decode the age groups, 0 represent(0–10), 1 represent(10–20), 2 represent(20–30), 3 represent(30–40), 4 represent(40–50), 5 represent(50–60), 6 represent(60–70) and 7 represent(70–80).
Clearly, we can see that most of the users belong to age group between 20-70, youngsters are high in number

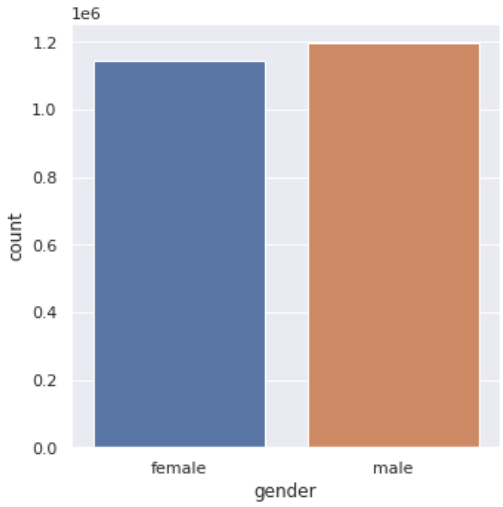
**# To check the different methods opted by users for registration**




Total number of distinct registration methods 18

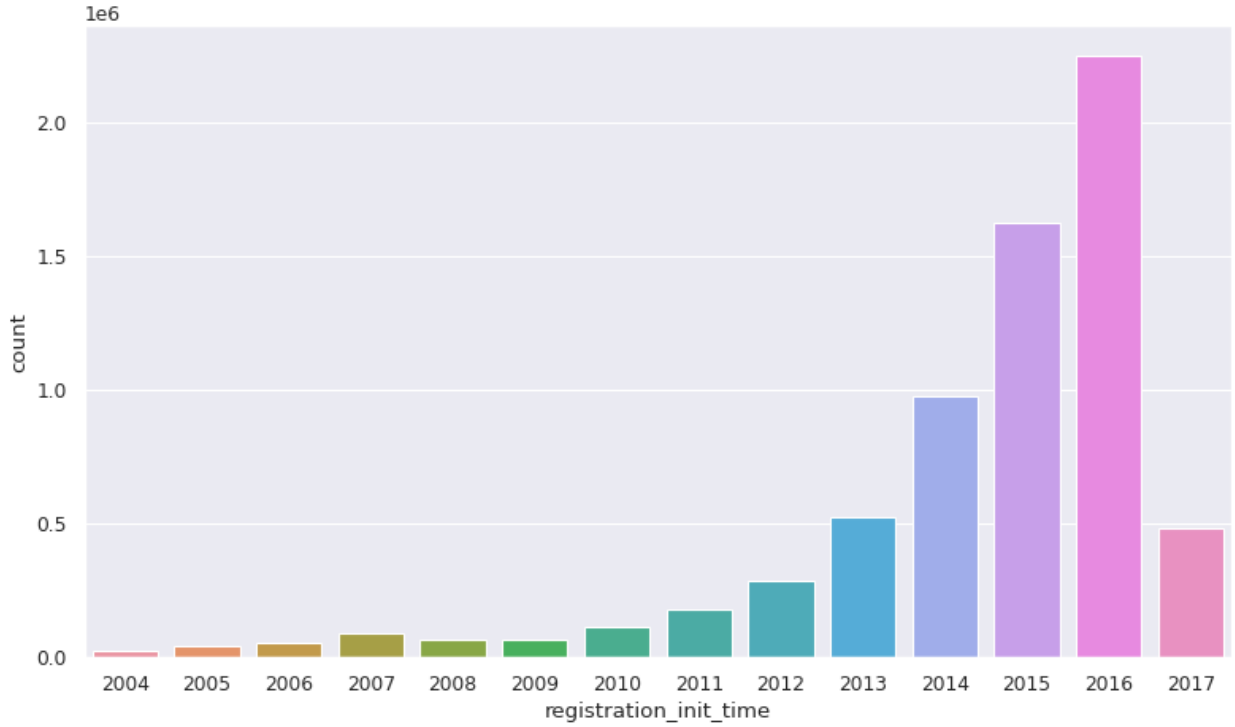
Here, methods 3,4,7,9 are mostly preferred by the users

**# To check gender distribution among users**



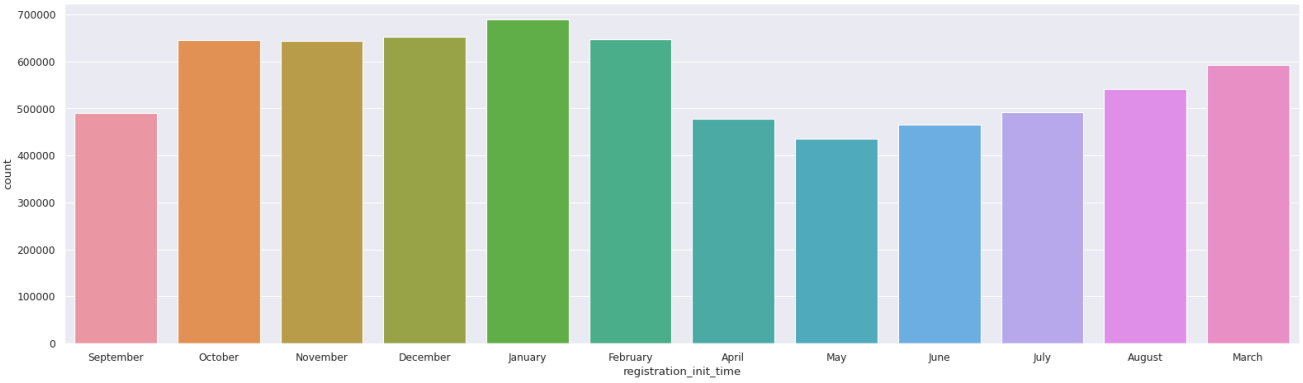

Gender distribution is almost same for both men and women

**# To see how the new registrations have increased down the years**




2017 users are less, because we have data only for 3 months

**# To check the user registrations in different months of the year**




More Users tend to make registration either in ending and starting of year

***# Inferences drawn from members data***

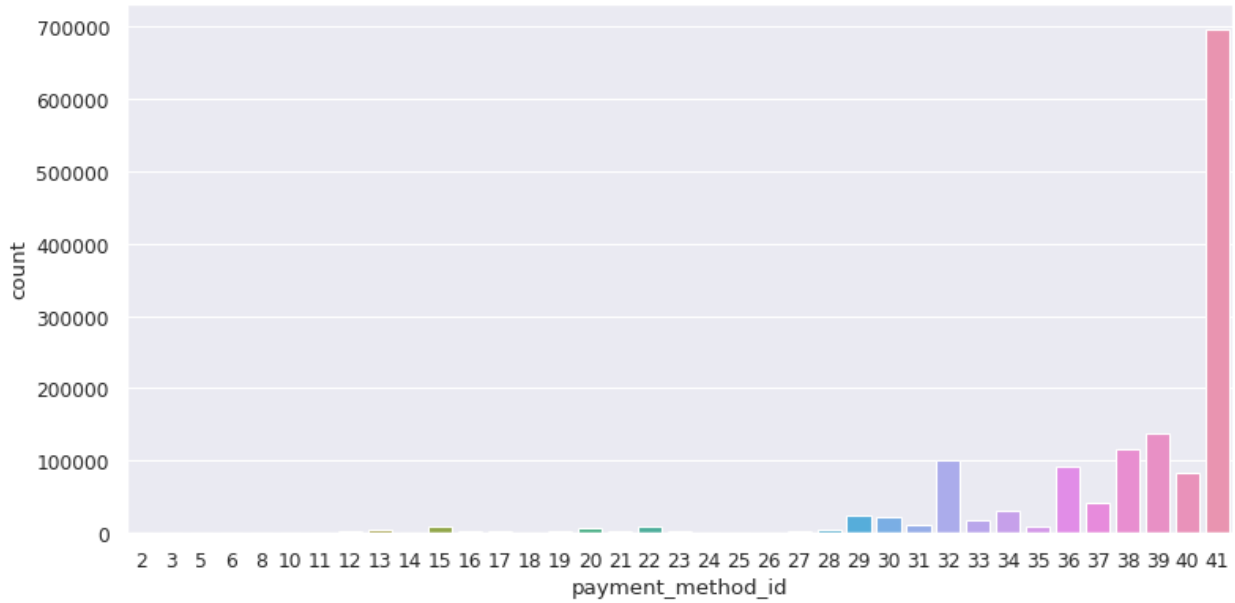
- members data contains 6 columns with shape (6769473,6)
- There are overall 21 distinct cities and city 1 having highest number of customers
- Age columns clearly contains outliers, therfore it needs to be rectified
- Null values are present in gender coulmn, datatype of gender data is string
- There is 18 distinct registration methods
- Mostly youngsters are more in number

## ***3.*** Transaction data

```
#Read transaction data
transaction_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/transactions_v2.csv',
```

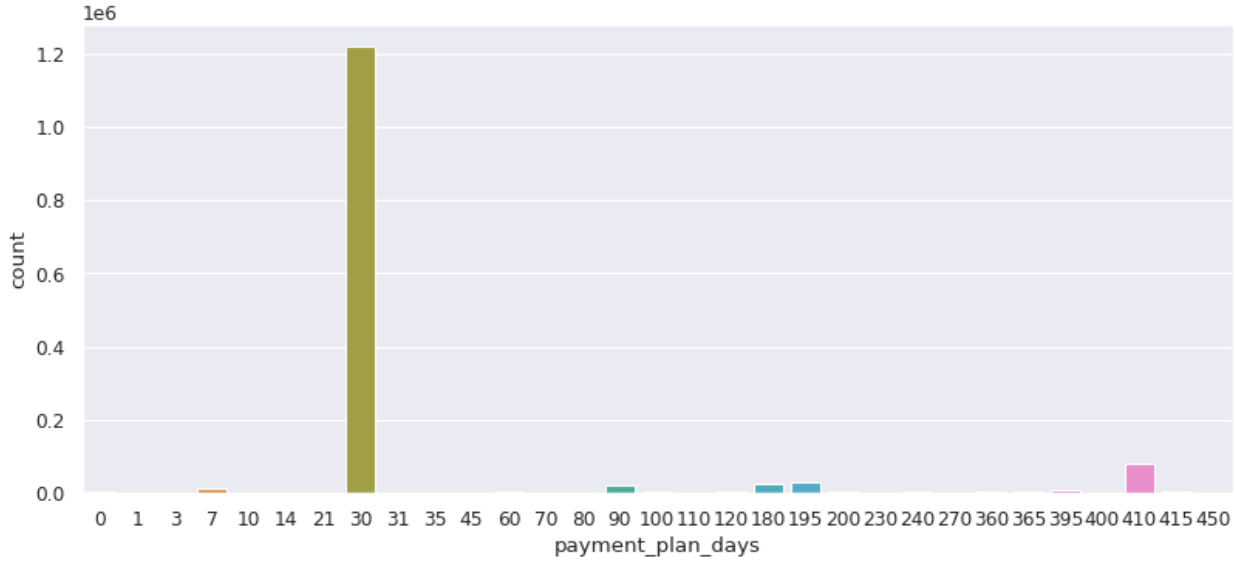
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1431009 entries, 0 to 1431008
Data columns (total 9 columns):
 #   Column                  Non-Null Count    Dtype         
---  ------                  --------------    -----         
 0   msno                    1431009 non-null  object        
 1   payment_method_id       1431009 non-null  int64         
 2   payment_plan_days       1431009 non-null  int64         
 3   plan_list_price         1431009 non-null  int64         
 4   actual_amount_paid      1431009 non-null  int64         
 5   is_auto_renew           1431009 non-null  int64         
 6   transaction_date        1431009 non-null  datetime64[ns]
 7   membership_expire_date  1431009 non-null  datetime64[ns]
 8   is_cancel               1431009 non-null  int64         
dtypes: datetime64[ns](2), int64(6), object(1)
```

**# To check the distribution of payment method, which users have opted**



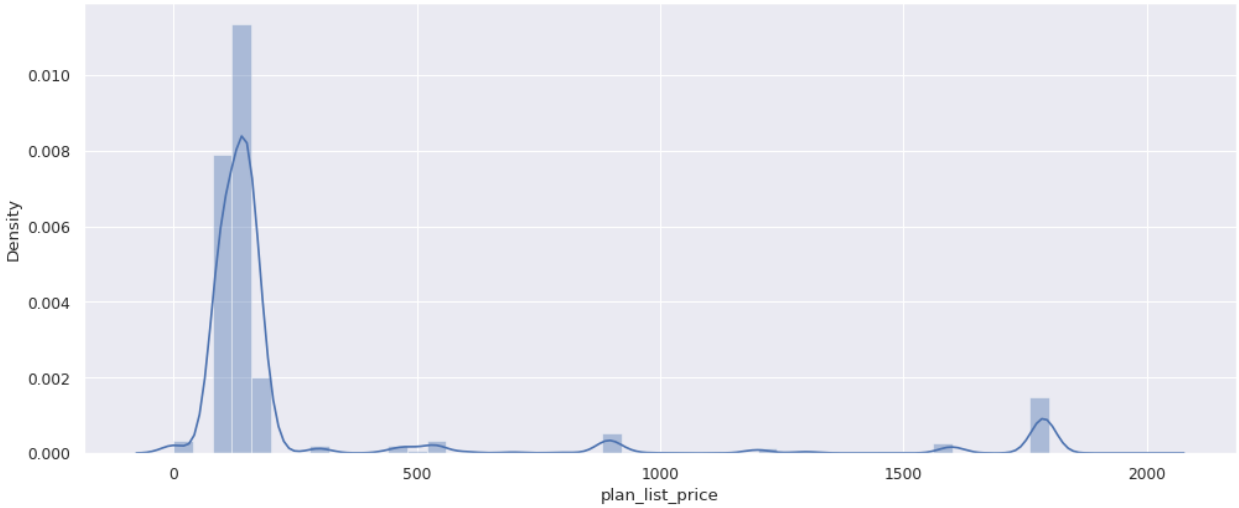

41 payment method is more used

**# To  check the distribution of subscription plan which users have opted**



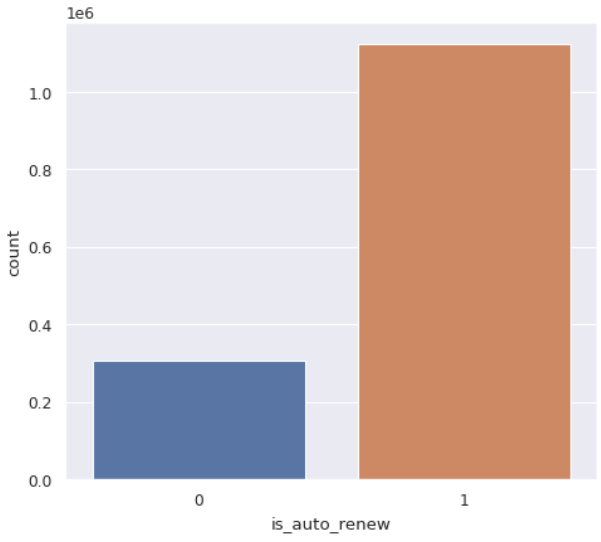

30 days plan seems to highly popular!
And beside 30 days plan, the users who purchased another plan have very high tendency to leave the service.

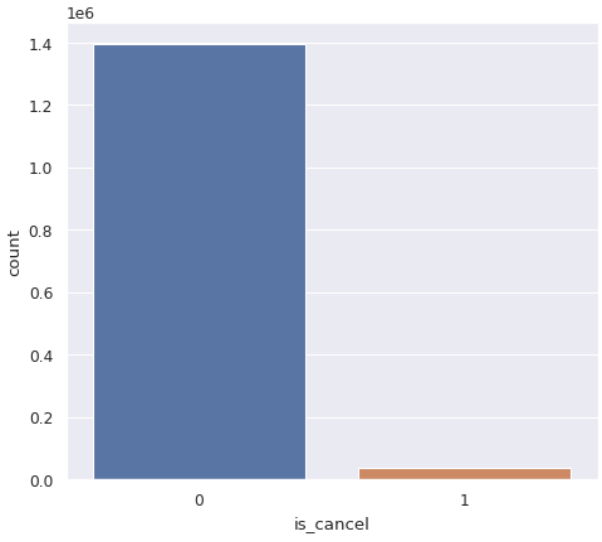
Many users have their plan price, like 99, 100, 129, 149, 180 NTD. Beside these five values if a user purchased any other plan then there is very high tendency of churning.




Many people tend to choose plan price around 99-200, however there are also some people who choose high price plans between 1500-2000! Since, actual amount paid and plan list price are almost same, the plot are similar

**# To check the relation between auto_renew and is_cancel against target variable**

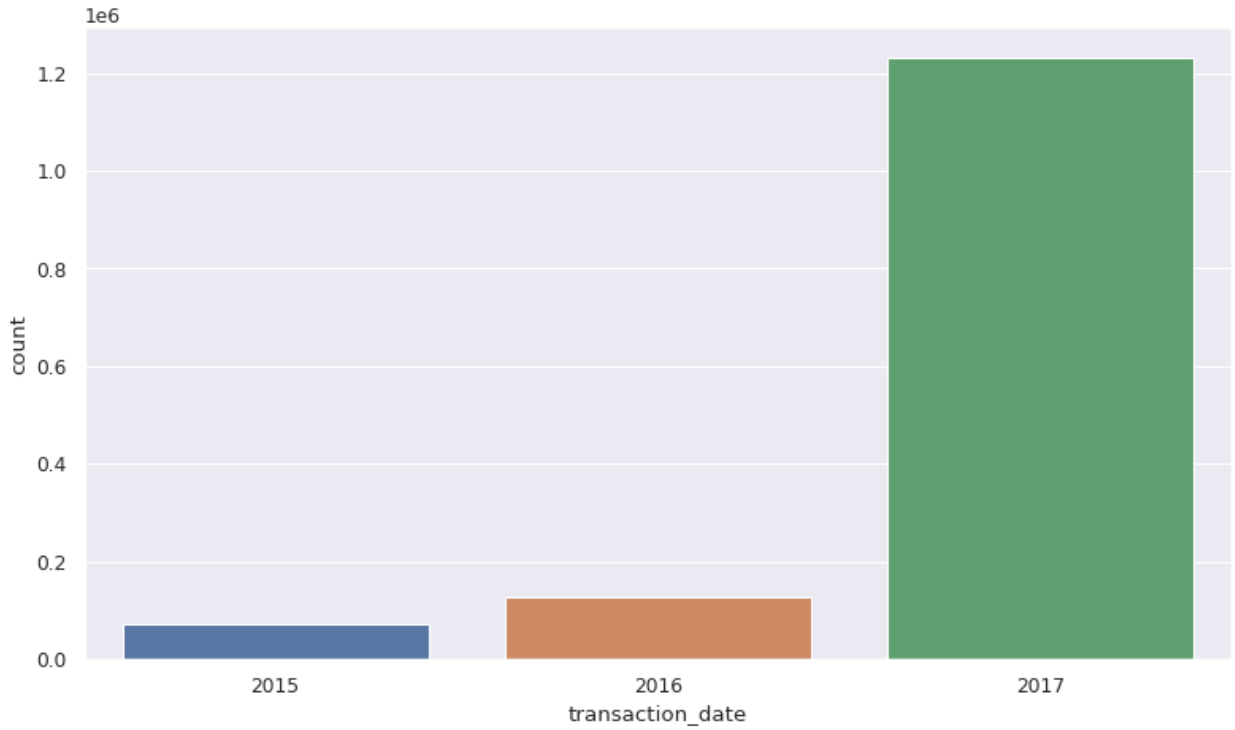



We can observe that, many users who had auto renew have not churned.

There are a lot of users who didn't cancel their subscription, and also among all those users churned. But the users who cancel their subscription, have very high tendency of churning.

**# User transactions year wise**




We have seen previously that most of the users opted for 30 days plan, and since the given data is mostly in 2017, in the above graph we can observe this pattern

```
April        1037972
May           142927
March          57106
June           39078
July           27782
August         26938
September      25030
October        18034
November       14913
January        14268
February       13532
December       13429
Name: membership_expire_date, dtype: int64
```

The given transaction data is of march month. And since many people have 30 Days plan, we can observe that they have expiry in the month of April

## *4.* User Logs dataset

```
User_logs = pd.read_csv('/content/drive/MyDrive/KKbox_data/user_logs_v2.csv', parse_dates=['date'])
```

| | msno | date | num_25 | num_50 | num_75 | num_985 | num_100 | num_unq | total_secs |
|---|---|---|---|---|---|---|---|---|---|
| 0 | u9E91QDTvHLq6NXjEaWv8u4QIqhrHk72kE+w31Gnhdg= | 2017-03-31 | 8 | 4 | 0 | 1 | 21 | 18 | 6309.273 |
| 1 | nTeWW/eOZA/UHKdD5L7DEqKKFTjaAj3ALLPoAWsU8n0= | 2017-03-30 | 2 | 2 | 1 | 0 | 9 | 11 | 2390.699 |
| 2 | 2UqkWXwZbIjs03dHLU9KHJNNEvEkZVzm69f3jCS+uLI= | 2017-03-31 | 52 | 3 | 5 | 3 | 84 | 110 | 23203.337 |
| 3 | ycwLc+m2O0a85jSLALtr941AaZt9ai8Qwlg9n0NqI5U= | 2017-03-31 | 176 | 4 | 2 | 2 | 19 | 191 | 7100.454 |
| 4 | EGcbTofOSOkMmQyN1NMLxHEXJ1yV3t/JdhGwQ9wXjnI= | 2017-03-31 | 2 | 1 | 0 | 1 | 112 | 93 | 28401.558 |

Basically,the data says

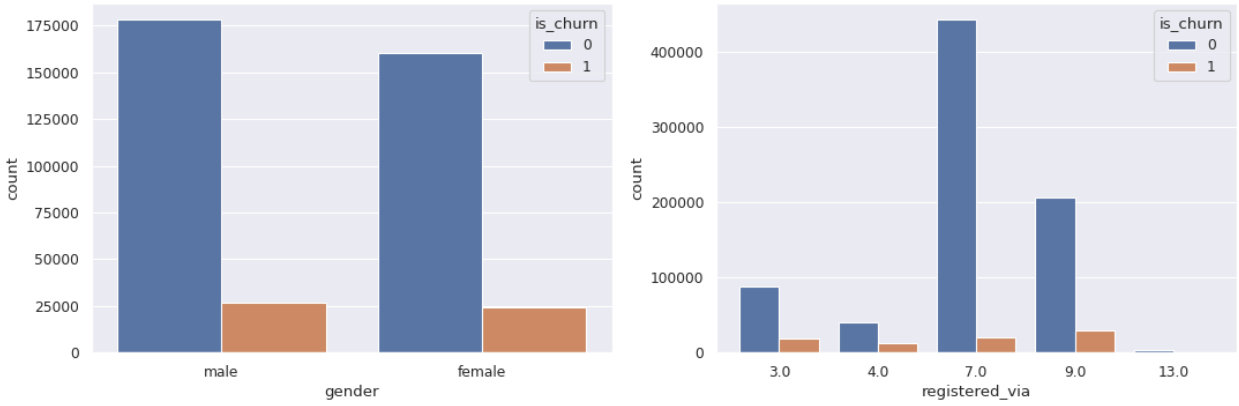
- msno: user id
- Date: format %Y%m%d
- num_25: Number of songs played less than 25% of the song length
- It can be seen that for most of the users, num25 and num100 is high and num50 and num75 is mostly low
- num_50: Number of songs played between 25% to 50% of the song length
- num_75: Number of songs played between 50% to 75% of of the song length
- num_985: Number of songs played between 75% to 98.5% of the song length
- num_100: Number of songs played over 98.5% of the song length
- num_unq: Number of unique songs played
- total_secs: total seconds played

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18396362 entries, 0 to 18396361
Data columns (total 9 columns):
 #   Column      Dtype
---  ------      -----
 0   msno        object
 1   date        datetime64[ns]
 2   num_25      int64
 3   num_50      int64
 4   num_75      int64
 5   num_985     int64
 6   num_100     int64
 7   num_unq     int64
 8   total_secs  float64
dtypes: datetime64[ns](1), float64(1), int64(6), object(1)
memory usage: 1.2+ GB
```

## *5.* Data Analysis against Target Variable

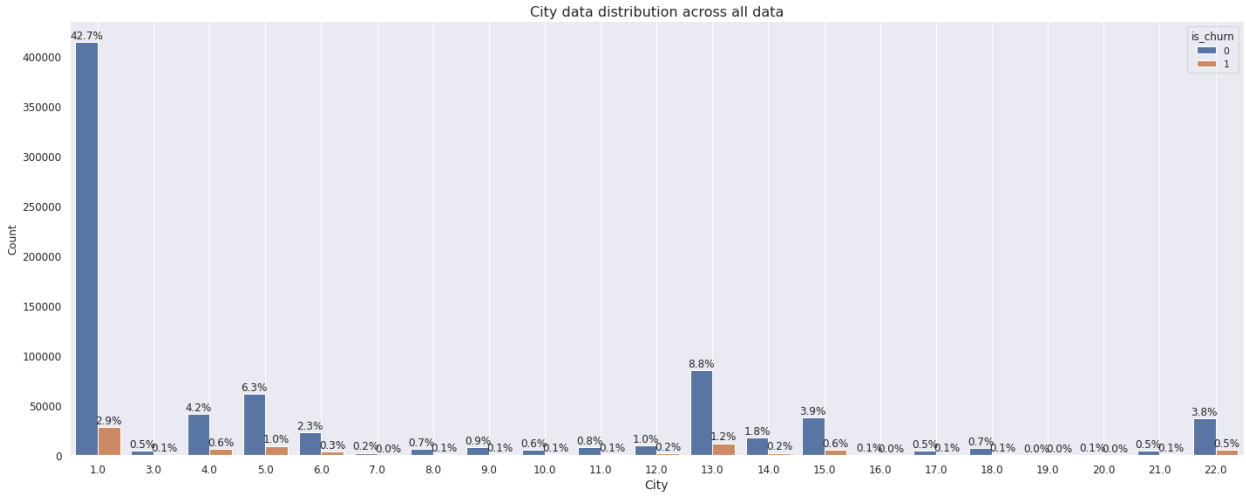
Now, lets merge all the datasets to analyse different fields against User churn data

```
#merging members data set with train data set
train_member=pd.merge(train_data,members,how='left',on='msno')
```




Here we can notice that, gender does not affect much on customer churn, however we can see that users who registered via method 9 have higher churn stats, we can mark that.

### # City data distribution




Observations: -

- There are a lot of users(almost 45%) from the city 1, but in terms of average churned users for this city is less, as compare to other cities.

- City 21 contains the highest churning rate as compare to other cities, which is 14.71%
- Average churning rate lie in between 10% to 14.7% except for the city 1, which is having average churning rate of 6.4%

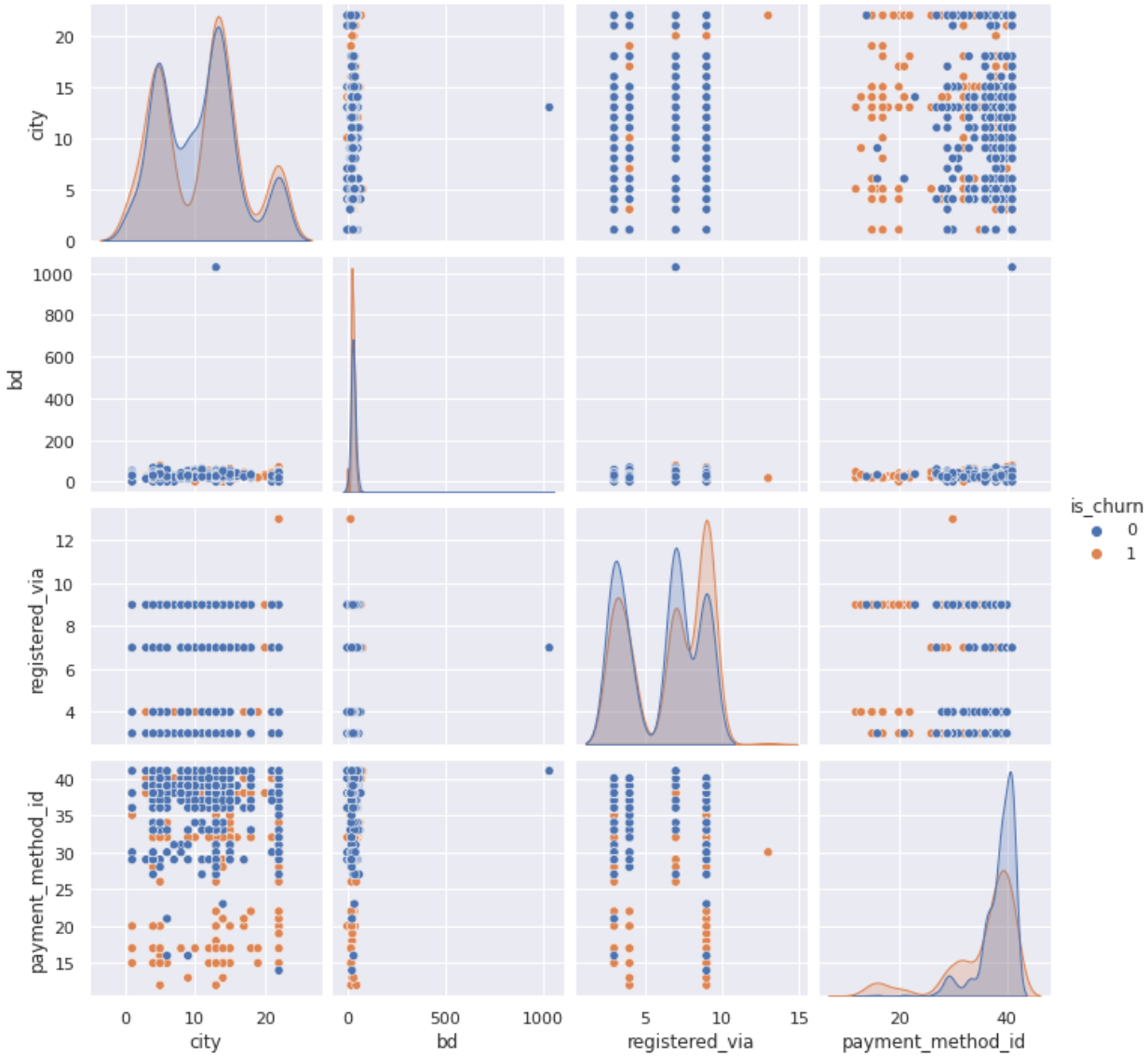
Merging with member dataset

```
# merging all 3 data i.e train member and transaction
train_member_transaction=pd.merge(train_member,transaction_data,how='left',on='msno')
```

| | %MissingValues |
|---|---|
| msno | 0.00 |
| is_churn | 0.00 |
| city | 9.99 |
| bd | 9.99 |
| gender | 57.16 |
| registered_via | 9.99 |
| registration_init_time | 9.99 |
| payment_method_id | 3.20 |
| payment_plan_days | 3.20 |
| plan_list_price | 3.20 |
| actual_amount_paid | 3.20 |
| is_auto_renew | 3.20 |
| transaction_date | 3.20 |
| membership_expire_date | 3.20 |

One of the key takeaway is the combined dataset is containing a lot of null values and outliers. So it's necessary to preprocess the data first.

**# Bivariate Analysis:**




Since most of the features overlap, its very difficult to differentiate and draw inferences on our target variable. So, using these features, we need to perform **feature engineering** in further steps.

# Data Preprocessing

In the previous phase, we analyzed all the datasets and derived conclusions. We have observed various patterns in the dataset through visualization. So, accordingly we need to now proceed towards refining the given datasets and create new features from the existing ones using statistical methods.

**# Lets import all the libraries to prepare the data**

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import warnings
import dask.dataframe as dd
from matplotlib.pyplot import figure
sns.set()
warnings.filterwarnings("ignore")
```

We also need to run an additional function to optimize the datasets, since they are consuming high memory.
This function takes DataFrame and adjusts the datatypes of the columns depending upon their Maximum and Minimum values.

**# Data preprocessing and featuring engg of members_dataset**

Reading the dataset from source

```
member_data=pd.read_csv('/content/drive/MyDrive/KKbox_data/members_v3.csv',parse_dates=['registration_init_time'])
member_data.head()
```

1. We have seen that in **gender** - feature, the data was equally distributed. So it does not provide much inferences towards the target variable. We drop it.
2. Replacing the negative values in **regsitered_via** feature with NaN and impute zero for NaN.
3. Setting upper limit and lower limit for age feature as 80 and 10. Replace the outliers with NaN values and impute a median age that is 28.
4. From EDA, we know city 1 and 13 have more user churn, so we create a new feature and impute 1 if its equal to 1 and 13 else impute 0
5. We drop the '**city**' column
6. Registration methods 7 and 9 show more user churn, hence we create a new feature and impute 1 and 0 accordingly
7. Now, we check for any duplicates or NaN values and store the dataset as a new csv file to the drive.

**# Data preprocessing and feature engg of transaction_data**

Reading the dataset from source

```
transaction_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/transactions_v2.csv',parse_dates=['transaction_date','membership_expire_date'])
transaction_data.head()
```

1. Imputing 0 in place of nan value in **payment_method_id**
2. From EDA we know that **payment_plan** 30 days has more influence on the target variable. So we need to remove the outliers greater than 30 and impute NaN values.
3. Similarly, for **plan_list_price** and **actual_plan_price**, we replace the plan prices above 180 with NaN and impute the median value that is 149
4. Replace the NaN values in the **is_auto_renew** feature
5. Transaction date contains NaN values, So we impute the median value that is 2017-03-16 inplace
6. Similarly, we impute median value in place of NaN values in **membership_expire_date** feature
7. Now, we need to drop all the duplicate values in the dataframe and add some new features
8. **is_auto_renew_change -** Captures if the user changed its auto_renew state
9. **is_cancel_change_feature -** Captures if the user changed its is_cancel state
10. **Average_plan** - Average plan of customer in days
11. **Average_amount_paid** - average amount paid by the user
12. **Average_amount_charged** - average amount charged for a user
13. **Change_in_plan -** captures how many times plan is changed

Now, we save the dataset to an updated csv file in the drive.

**# Data preprocessing and feature engg of user_logs data**

Reading the dataset from source

```
user_logs=pd.read_csv("/content/drive/MyDrive/KKbox_data/user_logs_v2.csv",parse_dates=['date'])
user_logs.head()
```

1. Checking for any null values and duplicate values in the dataframe.
2. Replace the NaN values in the **Date** column with the median date value that is 2017-03-16
3. In the **num_25** feature, from EDA we can see that 90 % of the values lie between 0 to 15. So, we can set all values above 15 as outliers.
4. Similarly, for **num_50, num_75, num_100, num_985**, we can remove the outliers and impute the median value
5. For the **num_unq** feature, 0 to 68 looks to be a good range. So, accordingly, we can remove the outliers and imute the median value
6. For, **total_secs** as well, 0 to 19167 looks to be a good range, So we can remove the outliers and impute 3880 as median value.
7. Further, we need to check for any duplicate values in the dataset and optimize the data types to save some space.

Now, we save the dataset to an updated csv file in the drive.

# Modeling

### # Importing the libraries for modeling

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import warnings
import dask.dataframe as dd
from matplotlib.pyplot import figure
sns.set()
warnings.filterwarnings("ignore")
```

We also need to run an additional function to optimize the datasets, since they are consuming high memory.
This function takes DataFrame and adjusts the datatypes of the columns depending upon their Maximum and Minimum values.

### # Importing the preprocessed datasets

```
[ ]  member_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/New_preprocessed_Data/members_data_updated')
```

```
[ ]  transaction_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/New_preprocessed_Data/transaction_data_updated')
```

```
[ ]  user_logs_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/New_preprocessed_Data/user_logs_updated')
```

### # Merging all the datasets

```
combined=pd.merge(member_data,transaction_data,on='msno',how='left')
combined=pd.merge(combined,user_logs_data,how='left',on='msno')
combined.head()
```

**# Importing the train dataset and merging with combined data**

```
train_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/train_v2.csv')
```

```
train_data=pd.merge(train_data,combined,on='msno',how='left')
train_data.fillna(0,inplace=True)
```

```
train_data.shape
```

```
(970960, 31)
```

**# Preparing data for balancing**

```
churned_user=train_data[train_data['is_churn']==1]
not_churned_user=train_data[train_data['is_churn']==0]
not_churned_user_sampled=not_churned_user.sample(frac=0.7)
new_sampled_data=not_churned_user_sampled.append(churned_user,ignore_index=True)
new_sampled_data.head()
```

```
X_train=new_sampled_data[train_features]
y_train=new_sampled_data['is_churn']
```

**# Balancing data Using SMOTE**

```
#using smote to balance the imbalance in the dataset
smote=SMOTE(random_state=110,n_jobs=-1)
X_bal,y_bal=smote.fit_resample(X_train,y_train)
```

**# loading test data**

```
test_data=pd.read_csv('/content/drive/MyDrive/KKbox_data/sample_submission_v2.csv')
test_data=pd.merge(test_data,combined,on='msno',how='left')
```

```
X_test=test_data[train_features]
X_test=sc.transform(X_test)
```

**# Splitting the data**

```
from sklearn.model_selection import train_test_split
X_train,X_cv,y_train,y_cv = train_test_split(X_bal,y_bal,test_size=0.3,random_state=110,stratify=y_bal)
```

# Beginning with some helper function to select the best hyperparameter with least log loss

As a baseline model, we can initially begin with logistic regression.

## # Logistic regression with l1

```
sample_points_train=int(0.4*X_train.shape[0])
sample_points_cv=int(0.4*X_cv.shape[0])

from sklearn.linear_model import LogisticRegression
feature_importance_lr=[]
feature_importance_values_lr=[]
reg=[]
loss_all=[]
```

# Hyperparameter tuning

```
#hyperparameter tuning

for i in [10**i for i in range(-3,3)]:
    model=LogisticRegression(C=i,n_jobs=-1,random_state=110,penalty='l1',solver='liblinear')
    print("Model is training")
    model.fit(X_train[:sample_points_train],y_train[:sample_points_train])
    print("Done")
    y_pred_proba=model.predict_proba(X_cv[:sample_points_cv])
    y_pred=model.predict(X_cv[:sample_points_cv])
    loss=result(y_pred,y_pred_proba,y_cv[:sample_points_cv])
    reg.append(i)
    loss_all.append(loss)
hyperparameter=lowest(reg,loss_all)
```

# training model with best hyperparameter

```
print("Training model with best hyperparameter")
model=LogisticRegression(C=hyperparameter,n_jobs=-1,random_state=110,penalty='l1',solver='liblinear')
print("Model is training")
model.fit(X_train,y_train)
print("Done")
y_pred_proba=model.predict_proba(X_cv)
y_pred=model.predict(X_cv)
loss=result(y_pred,y_pred_proba,y_cv,confusion=1)
```

# The output gives the results in log loss  and F1 score for different hyperparameters

```
Model is training
Done
log loss is  0.33512849077200774
F1 score is 0.8481155431335966
Model is training
Done
log loss is  0.32892273221928137
F1 score is 0.8540978981520666
Model is training
Done
log loss is  0.3277699514067516
F1 score is 0.8545145668708543
Model is training
Done
log loss is  0.32696738992966945
F1 score is 0.8546356729020707
Model is training
Done
log loss is  0.3268674454617268
F1 score is 0.8546841081446817
Model is training
Done
log loss is  0.32679698257722506
F1 score is 0.8546639884413434
The best hyperparameter is  100 and the lowest loss associaed with it  0.32679698257722506
```

# Training the model with the best hyperparameter

```
Training model with best hyperparameter
Model is training
Done
log loss is  0.3262687108457936
F1 score is 0.8542288149251959
```

Similarly,

## # Logistic regression with l2

```
Model is training
Done
log loss is  0.33379494908522156
F1 score is 0.8489587760305992
Model is training
Done
log loss is  0.3308268206320705
F1 score is 0.8527435926385909
Model is training
Done
log loss is  0.330222027630256
F1 score is 0.8536277558041977
Model is training
Done
log loss is  0.33009616636851297
F1 score is 0.8543846668780608
Model is training
Done
log loss is  0.3298305199481093
F1 score is 0.8537001522928536
Model is training
Done
log loss is  0.32988761596392524
F1 score is 0.8532050558627694
The best hyperparameter is  10 and the lowest loss associaed with it  0.3298305199481093
```

```
Training model with best hyperparameter
Model is training
Done
log loss is  0.3268573849501192
F1 score is 0.8542112352428493
```

## # Random Forest classifier

```
from sklearn.ensemble import RandomForestClassifier
feature_importance_rf=[]
feature_importance_values_rf=[]
parameter=[]
loss_all=[]

#hyperparameter tuning
for i in [100,200,500]:
    for j in [2,5,10,50]:
        model=RandomForestClassifier(n_estimators=i,random_state=110,n_jobs=-1,min_samples_split=j)
        print("Model is training")
        model.fit(X_train[:sample_points_train],y_train[:sample_points_train])
        print("Done")
        y_pred_proba=model.predict_proba(X_cv[:sample_points_cv])
        y_pred=model.predict(X_cv[:sample_points_cv])
        loss=result(y_pred,y_pred_proba,y_cv[:sample_points_cv])
        parameter.append((i,j))
        loss_all.append(loss)
```

```
Training model with best hyperparameter
Model is training
Done
log loss is  0.10422030414729824
F1 score is 0.950132603896194
```

## # XGB Classifier

```
from xgboost import XGBClassifier
hyp=[]
feature_importance_xgb=[]
feature_importance_values_xgb=[]
loss_all=[]


#hyperparameter tuning

for i in [100,200,500,1000]:
    for j in [0.5,0.6,0.8]:
        model=XGBClassifier(n_estimators=i,verbosity=1,n_jobs=-1,scale_pos_weight=j,random_state=110)
        print("Model is training")
        model.fit(X_train[:sample_points_train],y_train[:sample_points_train])
        print("Done")
        y_pred_proba=model.predict_proba(X_cv[:sample_points_cv])
        y_pred=model.predict(X_cv[:sample_points_cv])
        loss=result(y_pred,y_pred_proba,y_cv[:sample_points_cv],silent=1)
        hyp.append((i,j))
        loss_all.append(loss)
```

```
Training model with best hyperparameter
Model is training
Done
log loss is  0.11284224573004356
F1 score is 0.944588247107521
```

Among all the models, Random forest classifier performed the best and gave the best result with the least log loss of 0.10422

Further steps would be Advanced modeling where we need to create more features and try more efficient algorithms to find the best results.

# Advanced Modeling

In the previous phase, we had worked on data preprocessing, and created new features. Later, we built ML models like Logistic regression, Random forest and XGB classifier.
In this phase, we shall analyse the datasets further and create more features based on previous results and further build deep - learning models to see if we can obtains better results.

# Importing the libraries

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import warnings
from matplotlib.pyplot import figure
sns.set()
warnings.filterwarnings("ignore")
```

# Reading the datasets

```
member_data=pd.read_csv('/content/drive/MyDrive/KKbox_data/New_preprocessed_Data/members_data_updated',parse_dates = ['registration_init_time'])
member_data.head()

transaction_data =
pd.read_csv('/content/drive/MyDrive/KKbox_data/New_preprocessed_Data/transaction_data_updated')

userlogs_data =
pd.read_csv('/content/drive/MyDrive/KKbox_data/New_preprocessed_Data/user_logs_updated')
```

**# Adding new features to members_data**

```
member_data["year"]=member_data['registration_init_time'].dt.year
member_data['month']=member_data['registration_init_time'].dt.month
member_data['day']=member_data['registration_init_time'].dt.day
member_data['weekday']=member_data['registration_init_time'].dt.weekday
member_data.drop('registration_init_time',axis=1,inplace=True)
```

**By extracting the month data, we can create 'Month_Feature'**

```
Member_data['month_feature'] = member_data['month'].apply(lambda x:0 if
(x==4 or x==5 or x==6) else 1 )
```

**By extracting the year data, we can create 'year_Feature'**

```
Member_data['year_feature'] = member_data['year'].apply(lambda x:0 if
x>2014 else 1 )
```

**Along, with the previously added features, we have cleaned and prepared the members_dataset for modeling**

```
member_data.shape
```

```
(6769473, 8)
```

| | | |
|---|---|---|
| 0 | msno | object |
| 1 | bd | float32 |
| 2 | registered_via | float32 |
| 3 | city_feature | int8 |
| 4 | registration_method_f | int8 |
| 5 | year | int16 |
| 6 | month_feature | int8 |
| 7 | year_feature | int8 |

Saving the dataset to drive

```
member_data.to_csv('/content/drive/MyDrive/KKbox_data/AM_Datasets/members_
AM.csv')
```

**# Adding new features to transaction_dataset**

**We have added most of the new features to the transaction dataset in the previous phase itself. So, we can just cleanup the dataset again and optimize it for memory consumption**

```
transaction_data.shape
```

```
(1197050, 17)
```

```
 8   payment_method_id             float32
 9   payment_plan_days             float32
 10  plan_list_price               float32
 11  actual_amount_paid            float32
 12  is_auto_renew                 float32
 13  transaction_date              object
 14  membership_expire_date        object
 15  is_cancel                     float32
 16  duration                      float32
 17  is_auto_renew_change_feature  float32
 18  is_cancel_change_feature      float32
 19  average_plan                  float32
 20  average_amount_paid           float32
 21  average_amount_charged        float32
 22  change_in_plan                float32
```

**# Adding new features to Userlogs_dataset**

Along with previously added features, we are adding more features using the date feature

```
[ ]  temp_df = userlogs_data.groupby('msno').agg(active_days=('date', 'nunique'),
                                                 date_min=('date', 'min'),
                                                 date_max=('date', 'max'))
```

```
[ ]  userlogs_data = pd.merge(userlogs_data, temp_df, on='msno', how='left')
```

```
[ ]  import datetime as dt
```

```
     # Feature (activity period)

     userlogs_data['date_max'] = pd.to_datetime(userlogs_data['date_max'], errors='coerce')

     userlogs_data['date_min'] = pd.to_datetime(userlogs_data['date_min'], errors='coerce')
```

**'Active_days' feature and 'Inactive_days' feature**

```
[ ]  userlogs_data['active_days'] = (userlogs_data['date_max'] - userlogs_data['date_min']).dt.days + 1
```

```
[ ]  userlogs_data['inactive_days'] = userlogs_data['date'].nunique() - userlogs_data['active_days']
```

```
[ ]  userlogs_data = userlogs_data.drop_duplicates('msno', keep='first', inplace=False)
```

```
[ ]  userlogs_data.drop(['Unnamed: 0','date_min','date_max'],axis=1,inplace=True)
```

```
[ ]  userlogs_data.fillna(0,inplace = True)
```

**Further, we cleanup the dataset and optimize it for memory consumption**

```
date             1103894 non-null  object
num_25           1103894 non-null  float64
num_50           1103894 non-null  float64
num_75           1103894 non-null  float64
num_985          1103894 non-null  float64
num_100          1103894 non-null  float64
num_unq          1103894 non-null  float64
total_secs       1103894 non-null  float64
total_secs_mean  1103894 non-null  float64
active_days      1103894 non-null  int64
inactive_days    1103894 non-null  int64
```

**Note: Due to Shortage of resources, we are not adding more possible features like sum, mean, and Std Deviation.**

**# Combining the Datasets**

```
[ ]  combined=pd.merge(members_data,transaction_data,on='msno',how='left')
```

```
[ ]  adjust_datatype(combined)
```

```
[ ]  combined  = combined.drop_duplicates('msno', keep='first', inplace=False)
```

```
[ ]  combined.fillna(0,inplace=True)
```

```
[ ]  adjust_datatype(combined)
```

```
[ ]  combined=pd.merge(combined,userlogs_data,how='left',on='msno')
```

```
[ ]  combined  = combined.drop_duplicates('msno', keep='first', inplace=False)
```

```
[ ]  combined.fillna(0,inplace=True)
```

**# Saving the dataset to drive**

```
combined.to_csv('/content/drive/MyDrive/KKbox_data/AM_Datasets/combined_AM_3.csv',index=False)
```

**# Importing the Libraries**

```
pip install "dask[dataframe]" --upgrade

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gc
import warnings
import dask.dataframe as dd
from matplotlib.pyplot import figure
sns.set()
warnings.filterwarnings("ignore")

#Reading the combined dataset
combined =
pd.read_csv('/content/drive/MyDrive/KKbox_data/AM_Datasets/combined_AM_3.csv')
```

**# Preparing the combined dataset for modeling**

We also need to run an additional function to optimize the datasets, since they are consuming high memory.
This function takes DataFrame and adjusts the datatypes of the columns depending upon their Maximum and Minimum values

**# Importing the train dataset and merging with combined data**

```
train_data = pd.read_csv('/content/drive/MyDrive/KKbox_data/train_v2.csv')
```

```
train_data=pd.merge(train_data,combined,on='msno',how='left')
train_data.fillna(0,inplace=True)
```

```
train_features = train_data.columns
train_features = list(train_features)
train_features.remove('is_churn')
train_features.remove('msno')
train_data.head()
```

## # Preparing data for balancing

```
churned_user=train_data[train_data['is_churn']==1]
not_churned_user=train_data[train_data['is_churn']==0]
not_churned_user_sampled=not_churned_user.sample(frac=0.7)
new_sampled_data=not_churned_user_sampled.append(churned_user,ignore_index=True)
new_sampled_data.head()
```

```
X_train=new_sampled_data[train_features]
y_train=new_sampled_data['is_churn']
```

## # Balancing data Using SMOTE

```
#using smote to balance the imbalance in the dataset
smote=SMOTE(random_state=110,n_jobs=-1)
X_bal,y_bal=smote.fit_resample(X_train,y_train)
```

## # loading test data

```
test_data=pd.read_csv('/content/drive/MyDrive/KKbox_data/sample_submission_v2.csv')
test_data=pd.merge(test_data,combined,on='msno',how='left')
```

```
X_test=test_data[train_features]
X_test=sc.transform(X_test)
```

## # Splitting the data

```
from sklearn.model_selection import train_test_split
X_train,X_cv,y_train,y_cv = train_test_split(X_bal,y_bal,test_size=0.3,random_state=110,stratify=y_bal)
```

## # Advanced Modeling - Deep learning model -MLP

```
# loading library
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from prettytable import PrettyTable

from sklearn.metrics import log_loss

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, LSTM, Flatten, Concatenate,
Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model
```

### # Defining callback function

**Lets define a custom callback function, to get the F1 score, and use early stopping if the model performance does not improve on consecutive epocs, and therefore we have to save our models at every checkpoint and finally get the best model.**

**Note: Hyperparameter tuning is done separately because of lack of resources**

```
import tensorflow as tf

# Lets us define some custom callbacks
# setting F1 score as the Metric
# stopping if accuracy does not increase
class CustomCallback(tf.keras.callbacks.Callback):

    def __init__(self,X_cv,y_cv):
        self.x = X_cv
        self.y = y_cv
    def on_train_begin(self, logs={}):
        self.history={'loss': [],'F1_score':[]}

    def on_epoch_end(self, epoch, logs={}):
        self.history['loss'].append(logs.get('loss'))
        y_pred = (self.model.predict(self.x) > 0.5).astype("int32")
        F1_score=f1_score(self.y, y_pred, average='micro')
        self.history['F1_score'].append(F1_score)
        print("F1 Score  {}".format(self.history['F1_score'][0]))

history_own = CustomCallback(X_cv,y_cv)

from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(monitor='val_loss', patience=3,verbose=1)

callback_list = [history_own,early_stop]
```

```
# to store the best model
filepath = "/content/drive/MyDrive/KKbox_data/Models/best_model_MLP.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
```

```
callback_list = [history_own, early_stop, checkpoint]
```

**# Setting up different layers for the model**
**Using 'relu' and 'sigmoid' as activation functions**

```
# After hyperparameter tuning, we get 512,32 as best output
# Note: Hyperparameter tuning and selection is done separatly to avoid memory outage in Colab
model = Sequential()

# Dense hidden layer 1
model.add(Dense(512, input_dim=int(X_train.shape[1]),activation='relu'))

# Batch Normalization layer 1
model.add(BatchNormalization())

# Dropout layer 1
model.add(Dropout(rate=0.25))

# Dense hidden layer 2
model.add(Dense(32, activation='relu'))

# Batch Normalization layer 2
model.add(BatchNormalization())

# Dropout layer 2
model.add(Dropout(rate=0.25))

# Dense hidden layer 3
model.add(Dense(32, activation='relu'))

model.add(Dropout(rate=0.1))
model.add(Dense(1, activation='sigmoid'))

# summary
model.summary()
```

**# Compiling the model**

```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

**# Training the model**

```
# Fit the model
model.fit(X_train, y_train, epochs=50, batch_size=2**13,validation_data=(X_cv,y_cv), verbose=1,callbacks=callback_list)
```

```
Epoch 45: val_loss improved from 0.12455 to 0.12333, saving model to
/content/drive/MyDrive/KKbox_data/Models/best_model_MLP.h5
106/106 [==============================] - 23s 215ms/step - loss: 0.1334 -
accuracy: 0.9423 - val_loss: 0.1233 - val_accuracy: 0.9450
```

## # Advanced Modeling - Deep learning model - CNN

```
X_train=np.reshape(X_train,(X_train.shape[0],X_train.shape[1],1))
X_cv=np.reshape(X_cv,(X_cv.shape[0],X_cv.shape[1],1))
```

## # Similarly, we define custom callbacks for the CNN model

### # Defining the layers for the model and using'relu' and 'sigmoid' as activation function

### Note: After, hyperparamter tuning, we get the best results from 32,32 filter

```
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1:])))
model.add(Conv1D(filters=32, kernel_size=3, activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPool1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Model summary
model.summary()
```

```
# to store the best model
filepath = "/content/drive/MyDrive/KKbox_data/Models/best_model_CNN.h5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
```

```
callback_list = [history_own, early_stop, checkpoint]
```

**# Training the model**

```
# Fit the model
model.fit(X_train,
y_train,epochs=50,batch_size=2**16,validation_data=(X_cv,y_cv),
verbose=1,callbacks=callback_list)

Epoch 32: val_loss improved from 0.17053 to 0.16677, saving model to
/content/drive/MyDrive/KKbox_data/Models/best_model_CNN.h5
14/14 [==============================] - 18s 1s/step - loss: 0.1746 -
accuracy: 0.9287 - val_loss: 0.1668 - val_accuracy: 0.9340
```

**# Here, we can clearly see that, the MLP model has comparatively performed better.**

**Deployment**: The final phase of this project is Deployment, here we are deploying the whole machine learning pipeline into a simple Webapp.
In this final stage we need to deploy this machine learning pipeline for research available to end users for use. The model will be deployed in streamlit webapp where it takes input from customer data and predict the output. Logistic Regression, Random Forest, and XGB Classifier were trained using the trained dataset, it was found that , Random Forest gives the best performance.

**Conclusion: Among all the models, Random forest classifier performed the best and gave the best result with the least log loss of 0.10422**

**Referencs:**


1. https://www.kaggle.com/c/kkbox-churn-prediction-challenge
2. https://blog.hubspot.com/service/how-to-reduce-customer-churn
3. http://cs229.stanford.edu/proj2017/final-posters/5147439.pdf
4. https://www.geeksforgeeks.org/python-working-with-date-and-time-using-pandas/
5. https://www.kaggle.com/c/kkbox-churn-prediction-challenge/discussion/46078
6. https://www.kaggle.com/jeru666/did-you-think-of-these-features
7. https://blogs.oracle.com/datascience/introduction-to-churn-prediction-in-python
8. https://arxiv.org/pdf/1802.03396
9. https://medium.com/analytics-vidhya/kaggle-top-4-solution-wsdm-kkboxs-churn-prediction-fc49104568d6