This Case Study is about solving a business problem of Elo using the methods of machine learning which is described briefly along with the various steps of building a machine learning model.

**Intro:**

Elo, one of the largest payment brands in Brazil, has built partnerships with merchants in order to offer promotions or discounts to its cardholders. Which is a part of marketing campaign, in this campaign their main objective is to keep their customers happy, by this sort of customized and specific perks. So, In order to know whom to target, they want to assign a loyalty score to each of the customers which is the main objective of this case study also. So, in this case study we will be building one machine learning model to predict loyalty score for a customer given its card-id and purchase history.

**Business problem:**

By doing this Elo will make sure that only its loyal customers get the best experiences and its partner merchants get repeated business. This will also save Elo from doing unwanted campaigns.

The reason of focusing loyalty is that, it has been seen that loyal customers convert friends and family into customers just through word of mouth, and they contribute higher revenues for brands they're loyal to. This isn’t merely speculation. Research by Bain and Company, cites that in almost no instance can a retailer break even on one-time shoppers due to the high cost of customer acquisition.

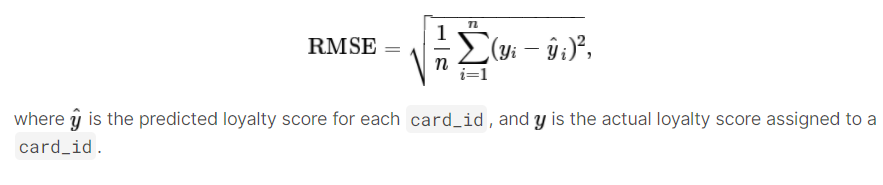
Evangelistic customers are much more effective in boosting the top line of a brand more than marketing any campaign can. This has led to the rise and popularity of loyalty programs.

**ML formulation of business problem:**

The ML problem that we want solve here is to predict the loyalty scores based on the available purchase data of cardholders and merchants. Loyalty score is a real number hence we are to use regression models here.

**Performance metric:**

Also, the metric provided is RMSE to evaluate this model.

**RMSE is defined as:**

**EDA ON THE DATA SETS –**

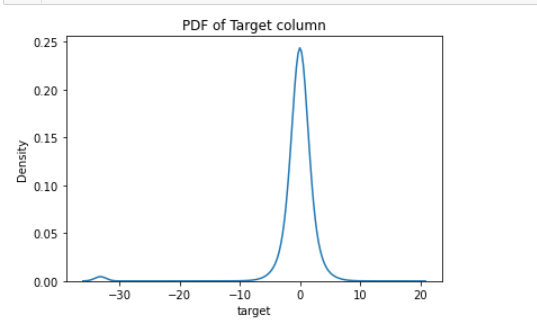
* 1. **Train.csv and test.csv** - This file consists of card id , various categorical features whose meaning are not explicit and also the loyalty score which we have to predict.

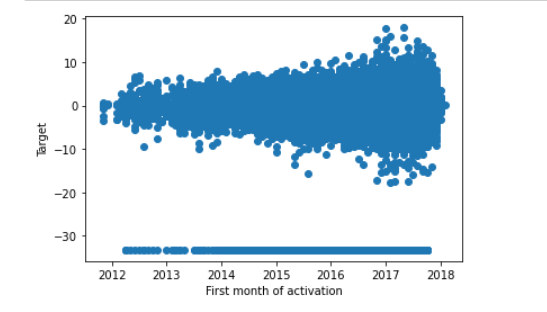
Train dataset columns detail

|  |  |
| --- | --- |
| **Columns** | **Description** |
| card\_id | Unique card identifier |
| first\_active\_month | 'YYYY-MM', month of first purchase |
| feature\_1 | Anonymized card categorical feature |
| feature\_2 | Anonymized card categorical feature |
| feature\_3 | Anonymized card categorical feature |
| target | Loyalty numerical score calculated 2 months after historical and evaluation period |

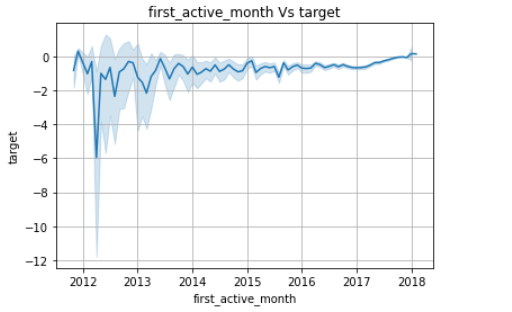
**\*Test data set has all the similar features except the target column which is to be predicted**

**TARGET VS FIRST MONTH OF ACTIVATION – SCATTER PLOT**

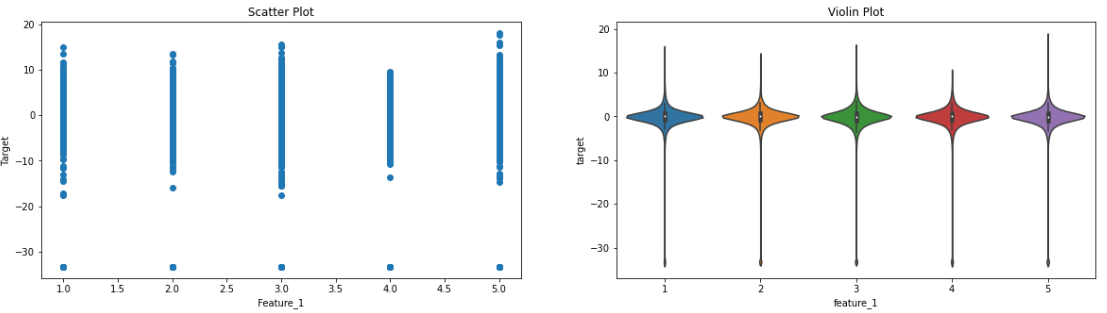
****

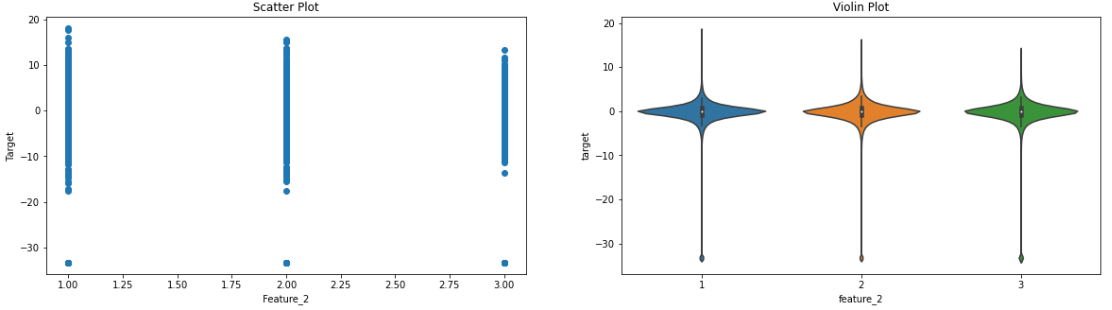
****

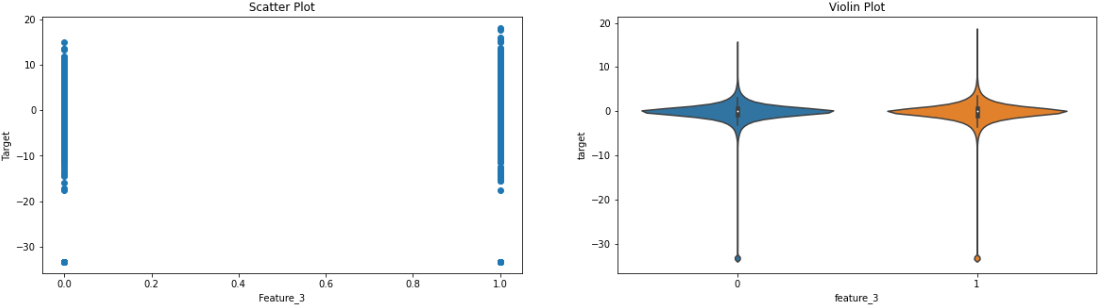
|  |  |
| --- | --- |
| We can see most of the data is distributed between -10 to +10 though some distribution can be noticed below -30 as well.  Apart from this we can see most of the values are tend have 0 loyalty scores.  This could mean data has been standardized at first and then this values below -30 has been added | It's clear from above plot that with time number of users have increased. Even though with time the range of loyalty score have increased but the polarity is almost equal on both sides. |



Though from this line plot it seems like after 2016 loyalty scores are getting more stable and tend to have positive values.

# Distribution of Feature 1,2 & 3





from above observation of feature 1,2 & 3 I can conclude that

#1. All these features are categorical

#2. There's no such distinguishable relationship that can be found between loyalty scores and these features

#3. Some data imbalanced can be noticed of these features

#5. most of the data of these features are contributing to the loyalty scores between the range of -10,10

#6. Data overlap is too much here

* 1. **Transactions Files-**
     1. **historical\_transactions.csv** - up to 3 months' worth of historical transactions for each card\_id
     2. **new\_merchant\_transactions.csv** - two months' worth of data for each card\_id containing ALL purchases that card\_id made at merchant\_ids that were not visited in the historical data

Columns Detail

|  |  |
| --- | --- |
| **Columns** | **Description** |
| card\_id | Card identifier |
| month\_lag | month lag to reference date |
| purchase\_date | Purchase date |
| authorized\_flag | Y' if approved, 'N' if denied |
| category\_3 | anonymized category |
| installments | number of installments of purchase |
| category\_1 | anonymized category |
| merchant\_category\_id | Merchant category identifier (anonymized ) |
| subsector\_id | Merchant category group identifier (anonymized ) |
| merchant\_id | Merchant identifier (anonymized) |
| purchase\_amount | Normalized purchase amount |
| city\_id | City identifier (anonymized ) |
| state\_id | State identifier (anonymized ) |
| category\_2 | anonymized category |

Some basic stats for numerical features of Transaction tables

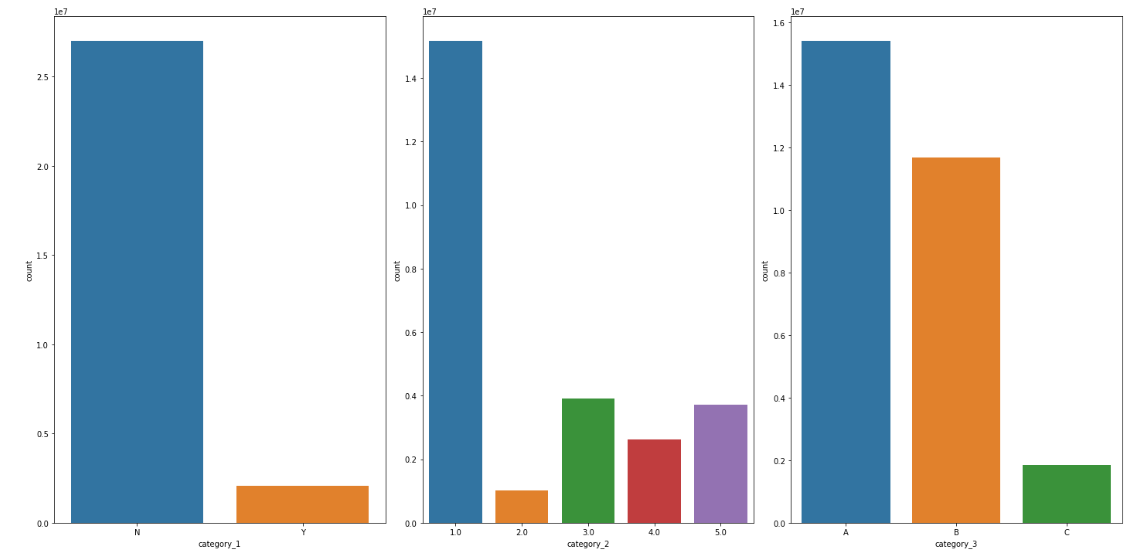
|  |  |
| --- | --- |
| **Historical Transaction** | **new\_merchant\_transactions** |
|  |  |
|  |  |
|  |  |
|  |  |

From the above I would like to say that only a few card holders have high installment and purchase amount its so rare that we can treat these high values as outliers.

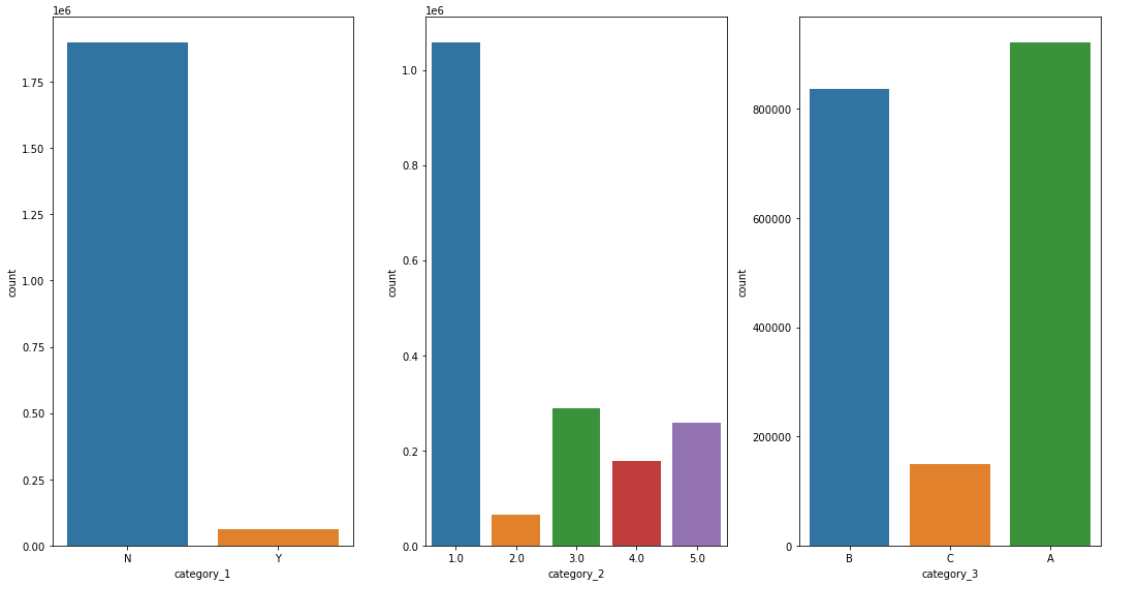
Also purchase\_amount and installment doesn't seems to be proportional

**Plots for categorical features**

1.1 Bar PLots for category 1, 2, 3 of historical transaction table



1.2 Bar PLots for category 1, 2, 3 of new\_merchant\_transactions



|  |  |
| --- | --- |
| **Historical Transaction** | **new\_merchant\_transactions** |
|  |  |
|  |  |
|  |  |
|  |  |

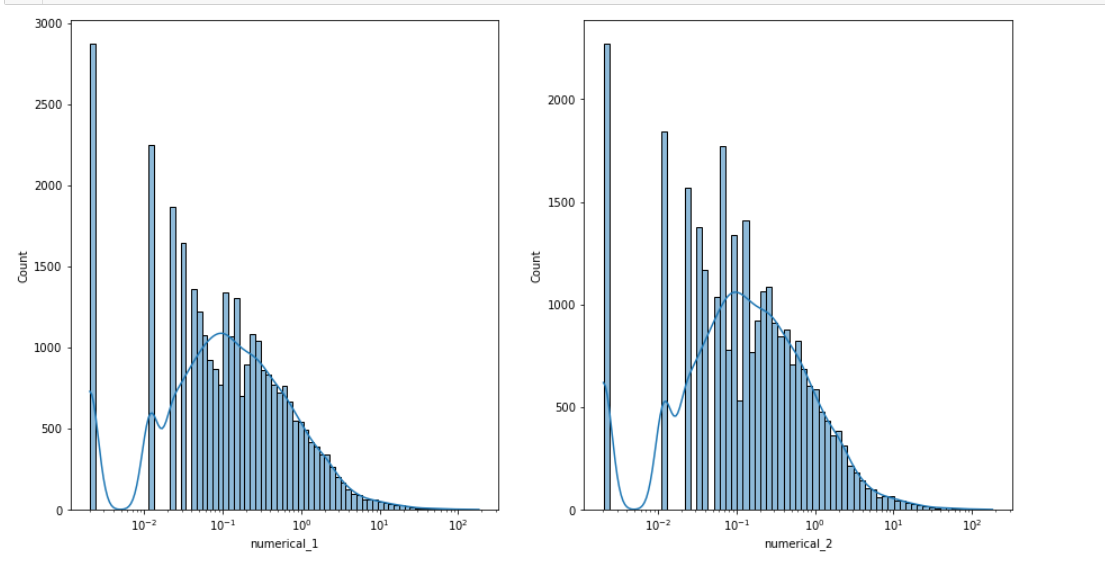
* 1. Merchants.csv - This file consists of merchants details.

Columns Detail

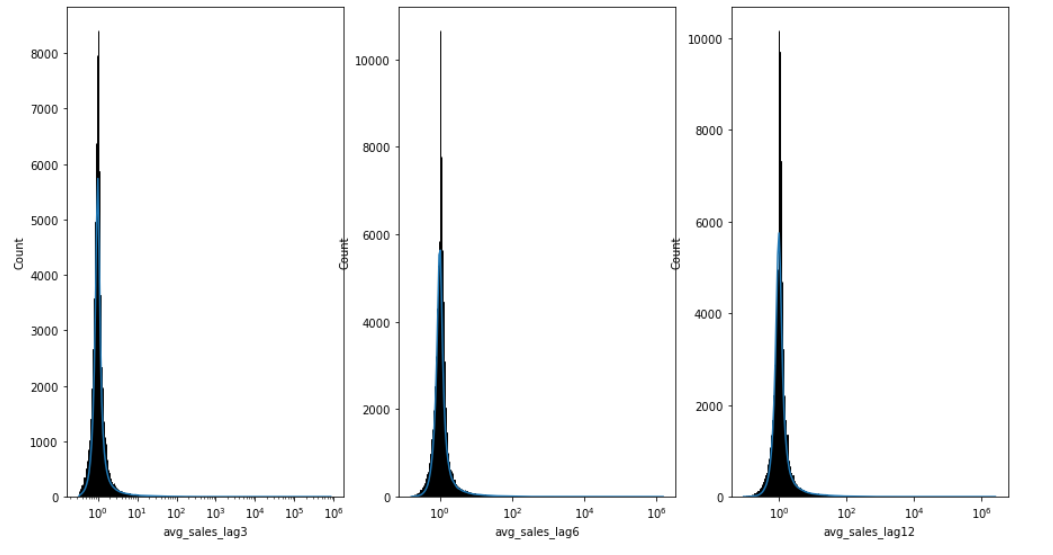
|  |  |
| --- | --- |
| **Columns** | **Description** |
| merchant\_id | Unique merchant identifier |
| merchant\_group\_id | Merchant group (anonymized ) |
| merchant\_category\_id | Unique identifier for merchant category (anonymized ) |
| subsector\_id | Merchant category group (anonymized ) |
| numerical\_1 | anonymized measure |
| numerical\_2 | anonymized measure |
| category\_1 | anonymized category |
| most\_recent\_sales\_range | Range of revenue (monetary units) in last active month --> A > B > C > D > E |
| most\_recent\_purchases\_range | Range of quantity of transactions in last active month --> A > B > C > D > E |
| avg\_sales\_lag3 | Monthly average of revenue in last 3 months divided by revenue in last active month |
| avg\_purchases\_lag3 | Monthly average of transactions in last 3 months divided by transactions in last active month |
| active\_months\_lag3 | Quantity of active months within last 3 months |
| avg\_sales\_lag6 | Monthly average of revenue in last 6 months divided by revenue in last active month |
| avg\_purchases\_lag6 | Monthly average of transactions in last 6 months divided by transactions in last active month |
| active\_months\_lag6 | Quantity of active months within last 6 months |
| avg\_sales\_lag12 | Monthly average of revenue in last 12 months divided by revenue in last active month |
| avg\_purchases\_lag12 | Monthly average of transactions in last 12 months divided by transactions in last active month |
| active\_months\_lag12 | Quantity of active months within last 12 months |
| category\_4 | anonymized category |
| city\_id | City identifier (anonymized ) |
| state\_id | State identifier (anonymized ) |
| category\_2 | anonymized category |

Distribution plots of numerical feature columns

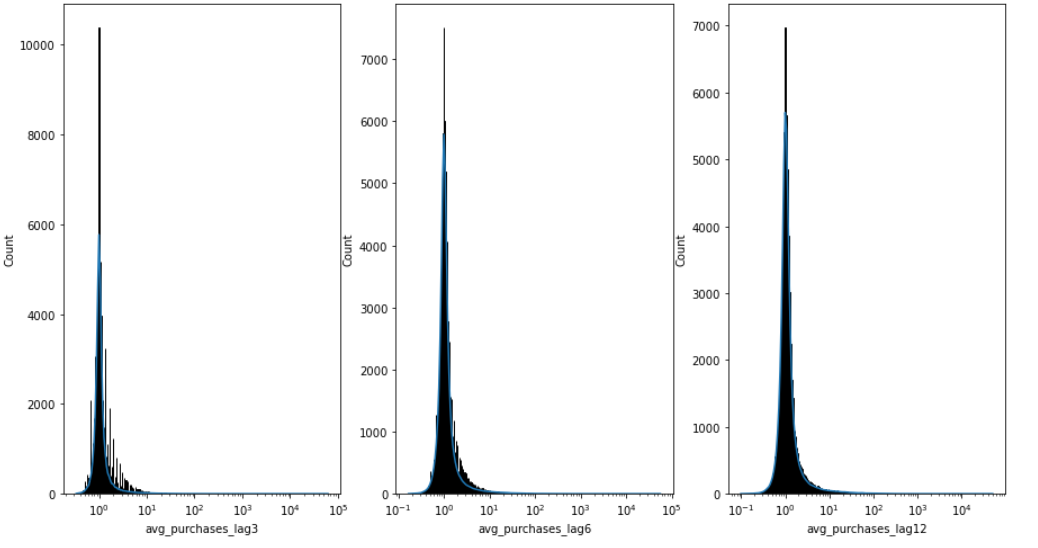
* For numerical 1& 2



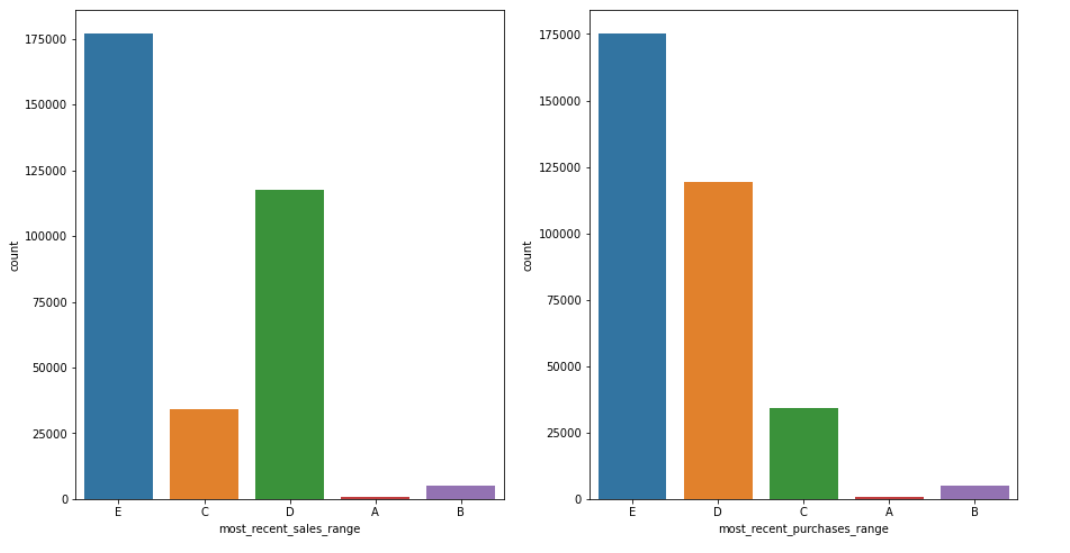
* For average sales

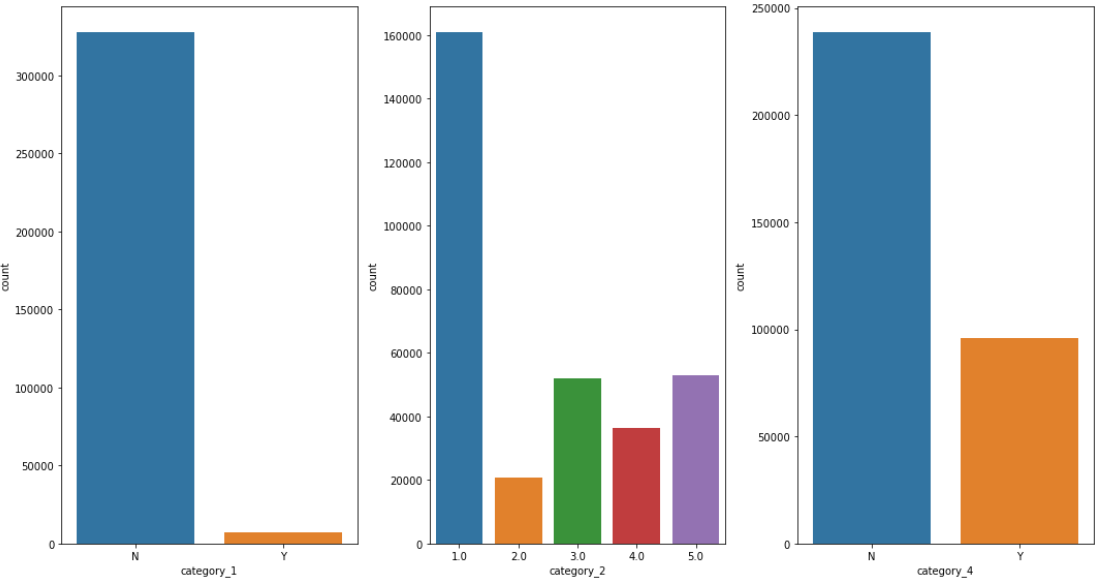


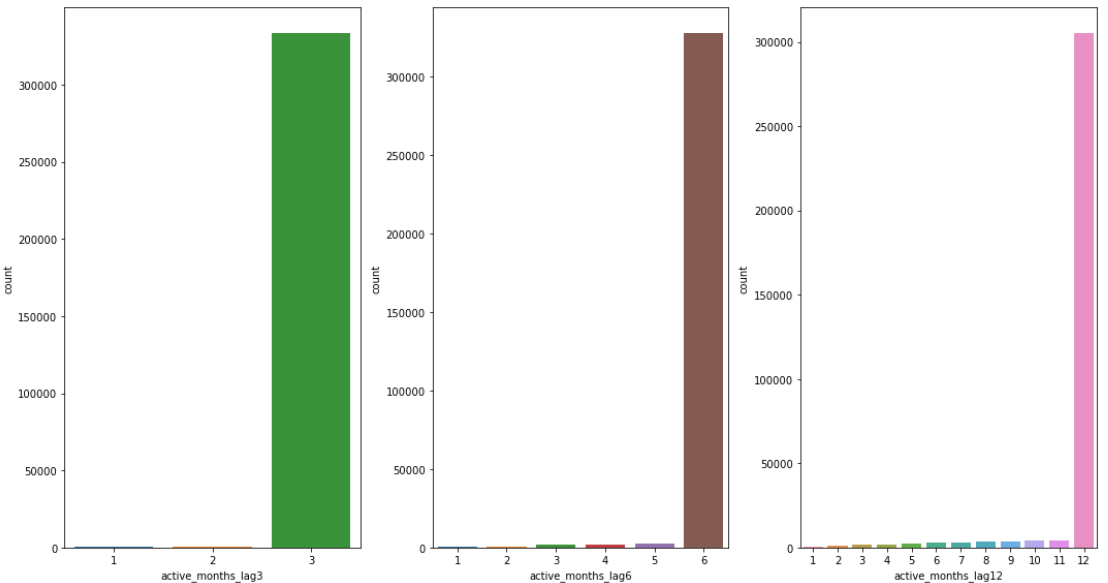
* For average purchase:



Bar Plots for categorical features:

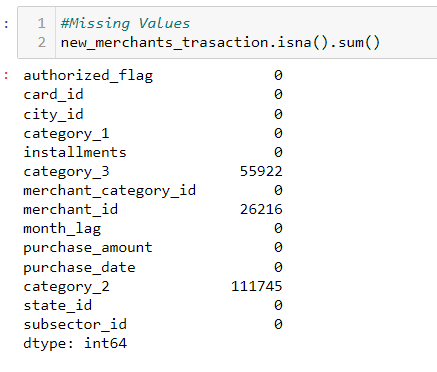
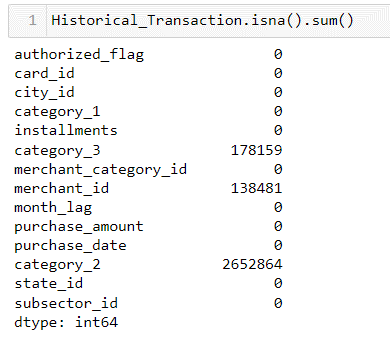




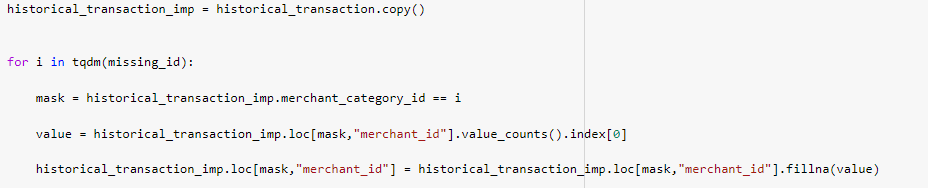


**Techniques adopted for missing value imputation and data cleaning:**

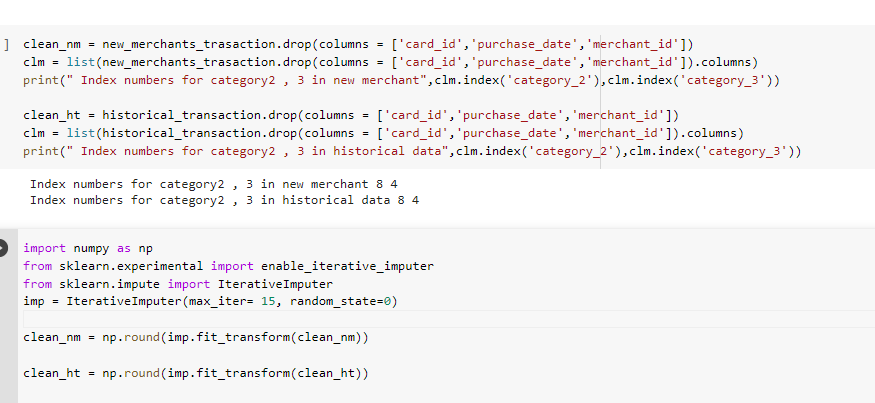
* + - 1. Train table didn’t have any missing value so it left as it is.

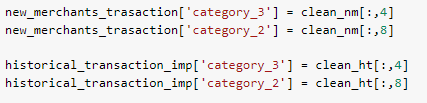
1. Test.csv had one missing card id which has been imputed with most frequent card\_id.
2. Now, transactions files had few categorical columns which had large number of missing values as shown below, that we simply can’t impute with most frequent values else it’d create a biased dataset, 

So, at first to impute merchant ids, rather taking most frequent merchant id in the entire dataset I took most frequent merchant ids belonging to each of the merchant categorie ids.

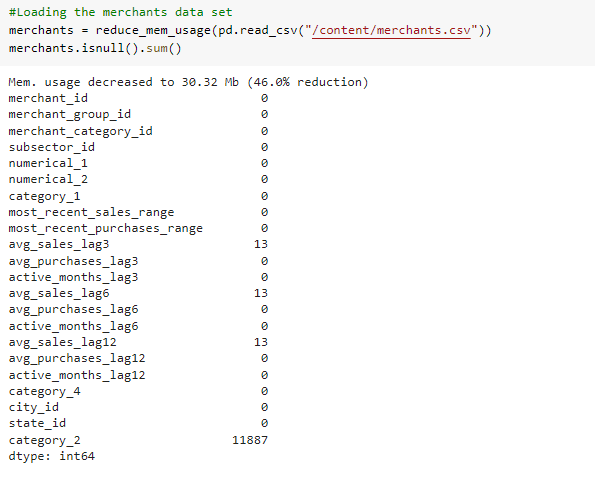
 

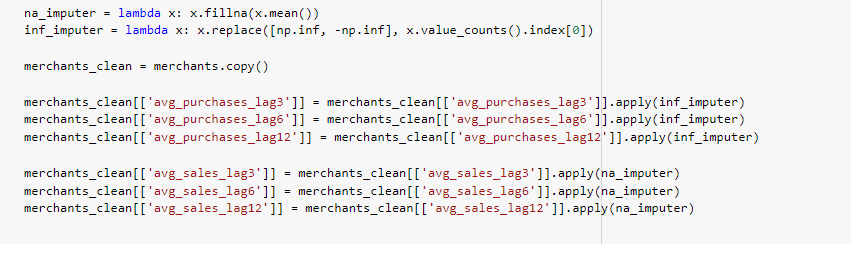
For other 2 columns I used iterative imputation technique





1. Merchant table has few missing values and inf values that has been taken care as followed





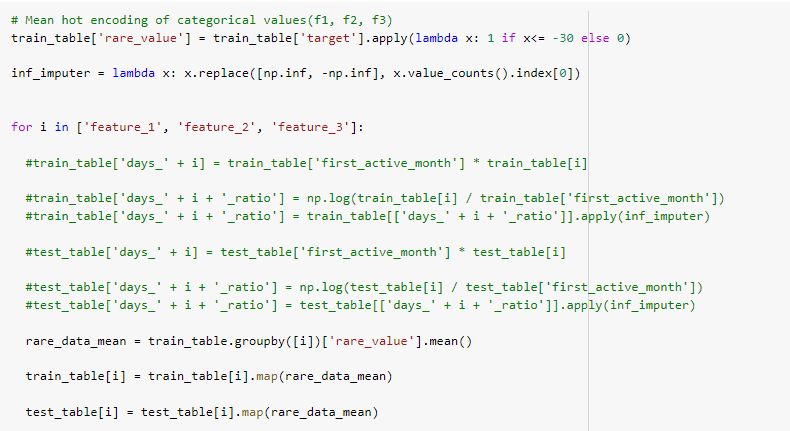
**Feature engineering**

In order to extract features from all the data sets techniques like **one** **hot encoding, mean hot encoding, cross feature extraction and aggregation methods** have been used along with time series data.

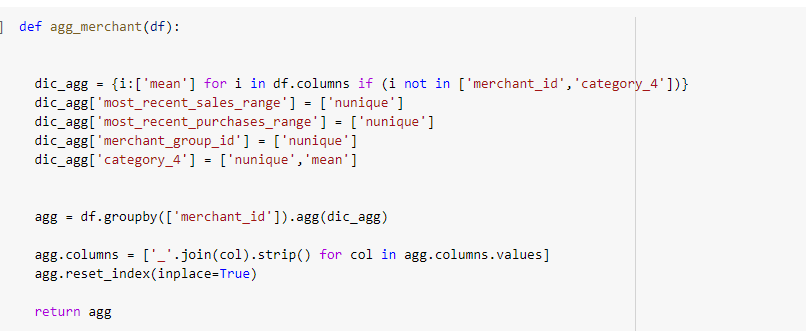
I am discussing in brief each of the kernels used for each of the tables.

* + - 1. For train and test table at the below kernel is used to get date time features, then feature 1, 2 & 3 has been mean hot encoded with the target values.



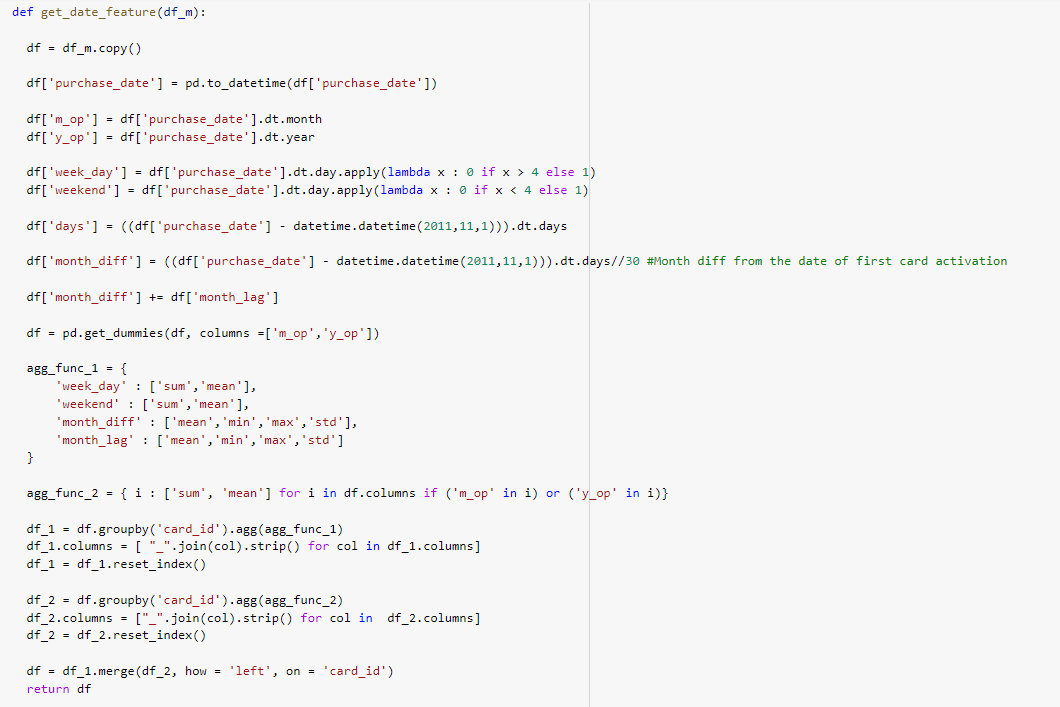
.

* + - 1. To get features from merchant data set below kernel is used which mainly gets aggregates over merchant ids to get various feature’s mean value and unique value counts

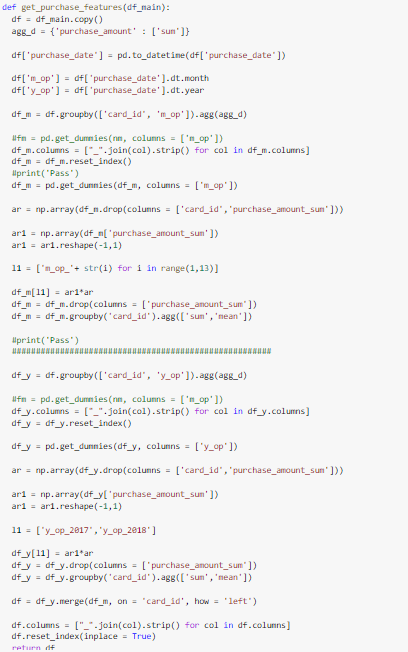


* + - 1. For transaction files at first I have merged two transaction files then at first I used two kernels, as mentioned below.

One is get\_date\_feature kernel which is used to get the detail about how frequently one user is purchasing each year, each month, weekends and week days.



And another kernel is get\_purchase\_feature, which I used to get an idea about how much each card user have spent on each month, each year etc.



I also used two more kernals that is used after merging transaction dataframe with featurized merchant dateframe and these kernals are mention bleow:

aggregate\_transactions in this kernel I’m aggregating all the columns after grouping them card ids

|  |
| --- |
| def aggregate\_transactions(df):      agg\_func = {      'category\_1': ['sum', 'mean',mode],      'category\_3\_0.0': ['mean','sum'],      'category\_3\_1.0': ['mean','sum'],      'category\_3\_2.0': ['mean','sum'],      'category\_2\_1.0': ['mean','sum'],      'category\_2\_2.0': ['mean','sum'],      'category\_2\_3.0': ['mean','sum'],      'category\_2\_4.0': ['mean','sum'],      'category\_2\_5.0': ['mean','sum'],      'authorized\_flag\_0' :['mean','sum'],      'authorized\_flag\_1' :['mean','sum'],      'merchant\_id': ['nunique'],      'merchant\_category\_id': ['nunique',mode],      'state\_id': ['nunique',mode],      'city\_id': ['nunique',mode],      'subsector\_id': ['nunique',mode],      'purchase\_amount': ['sum', 'mean', 'max', 'min', std],      'installments': ['sum', 'mean', 'max', 'min', std],        'numerical\_1\_mean' : ['sum', 'mean', 'max', 'min',std],      'numerical\_2\_mean' : ['sum', 'mean', 'max', 'min',std],      'most\_recent\_sales\_range\_mode\_c\_A' : ['mean','sum'],      'most\_recent\_sales\_range\_mode\_c\_B' : ['mean','sum'],      'most\_recent\_sales\_range\_mode\_c\_C' : ['mean','sum'],      'most\_recent\_sales\_range\_mode\_c\_D' : ['mean','sum'],      'most\_recent\_sales\_range\_mode\_c\_E' : ['mean','sum'],      'most\_recent\_purchases\_range\_mode\_c\_A' : ['mean','sum'],      'most\_recent\_purchases\_range\_mode\_c\_B' : ['mean','sum'],      'most\_recent\_purchases\_range\_mode\_c\_C' : ['mean','sum'],      'most\_recent\_purchases\_range\_mode\_c\_D' : ['mean','sum'],      'most\_recent\_purchases\_range\_mode\_c\_E' : ['mean','sum'],      'avg\_sales\_lag3\_mean' : ['sum', 'mean', 'max', 'min',std],      'avg\_purchases\_lag3\_mean':  ['sum', 'mean', 'max', 'min',std],      'active\_months\_lag3\_mean':['sum', 'mean', 'max', 'min',std],      'avg\_sales\_lag6\_mean':['sum', 'mean', 'max', 'min',std],      'avg\_purchases\_lag6\_mean':['sum', 'mean', 'max', 'min',std],      'active\_months\_lag6\_mean': ['sum', 'mean', 'max', 'min',std],      'avg\_sales\_lag12\_mean':['sum', 'mean', 'max', 'min',std],      'avg\_purchases\_lag12\_mean': ['sum', 'mean', 'max', 'min',std],      'active\_months\_lag12\_mean': ['sum', 'mean', 'max', 'min',std],      'category\_4\_mode\_c\_N' : ['mean','sum'],      'category\_4\_mode\_c\_Y' : ['mean','sum']         }        df\_n = df.groupby('card\_id').agg(agg\_func)        df\_tr = (df.groupby('card\_id')\              .size()\              .reset\_index(name='transactions\_count'))      df\_n.columns = ['\_'.join(col).strip() for col in df\_n.columns.values]      df\_n = df\_n.merge(df\_tr, how = 'left', on = 'card\_id')        return df\_n |

And another feature kernel on this merged transaction data frame **aggregate\_per\_month,** which is basically for grouped on card id and month lag then aggerated on installment and purchase amount.

|  |
| --- |
| def aggregate\_per\_month(df):      grouped = df.groupby(['card\_id', 'month\_lag'])      agg\_func = {              'purchase\_amount': ['count', 'sum', 'mean', 'min', 'max',std],              'installments': ['count', 'sum', 'mean', 'min', 'max',std],              }      intermediate\_group = grouped.agg(agg\_func)      intermediate\_group.columns = ['\_'.join(col).strip() for col in intermediate\_group.columns.values]      intermediate\_group.reset\_index(inplace=True)      final\_group = intermediate\_group.groupby('card\_id').agg(['mean', 'std'])      final\_group.columns = ['\_'.join(col).strip() for col in final\_group.columns.values]      final\_group.reset\_index(inplace=True)        return final\_group |

Modeling:

From the research done with the already available solutions and kernels, the simple model like SVM regression, KNN doesn’t work better for the prediction of the target value. So, I jumped into the complicated model like GBDT.

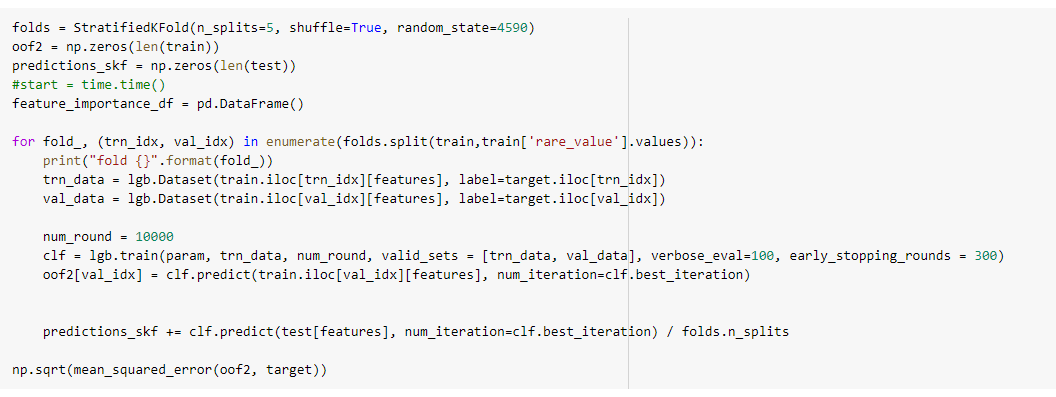
At first I used models like random forest and LGBM model among these two I got better result with LGBM model so I went with this base model for my further model building process.

In later stages I used k-fold and stratified k fold to get oof1 and oof2 features then using these two features I built one staked model .

k-fold model



Stratified k-fold model

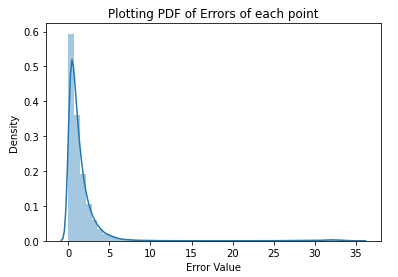




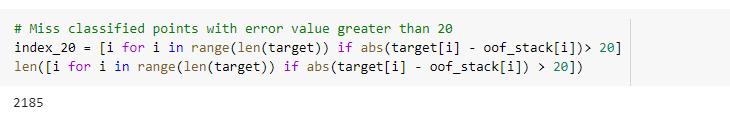
|  |  |  |
| --- | --- | --- |
| MODEL | PRIVATE SCORE | PUBLIC SCORE |
| RANDOM FOREST | 3.69694 | 3.798669 |
| LGBM | 3.66822 | 3.7625 |
| LGBM with k-fold | 3.65965 | 3.75837 |
| LGBM with stratified k-fold | 3.65922 | 3.75827 |
| STACKED model with Ridge regression | 3.66479 | 3.75600 |

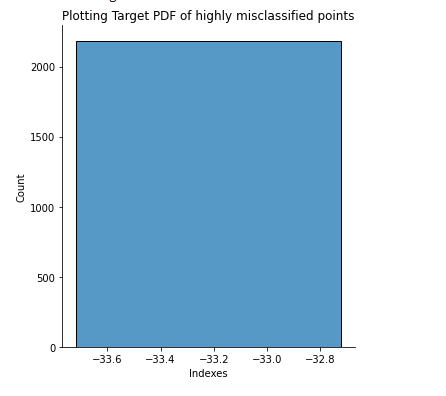
POST TRAINING ERROR ANALYSIS:

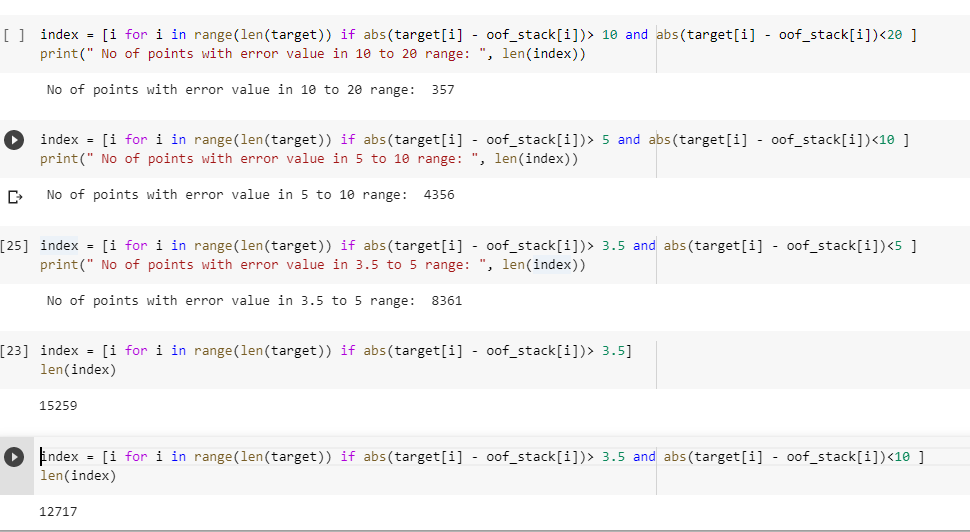
Below we can see a pdf of error values for each points.

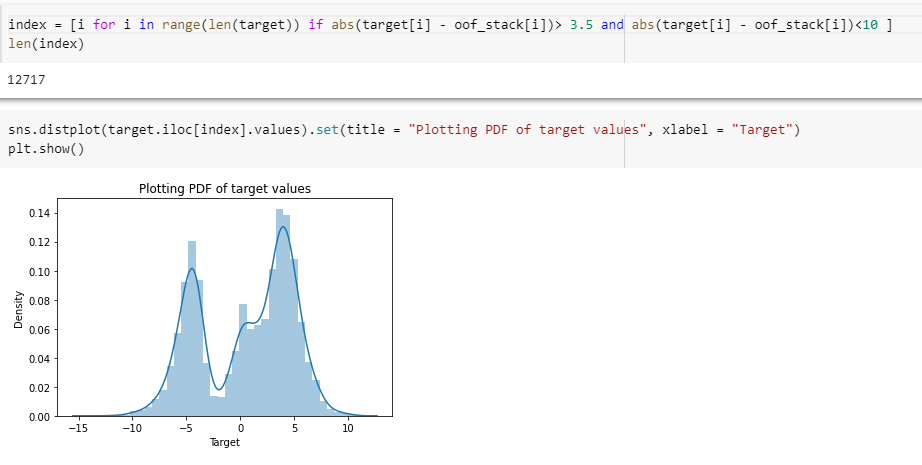


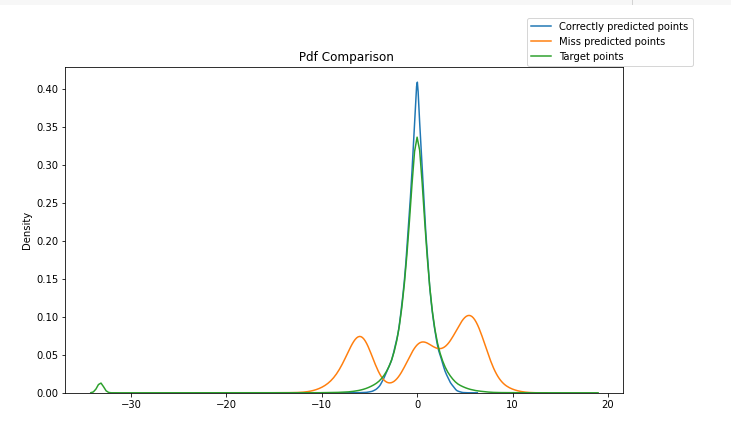
Though from above pdf we can say most of points have error values which is between 0-5 , and I did some analysis to get number points that falls into different range.











Conclusion:

So far the best model is the stacked model with LGBM models trained on stratified and non-stratified data with ridge meta learner, gave the Kaggle score 3.66479.

The reason model scores are poor its due to the fact that data over lap is too much and it’s very much difficult to separate rare data points from normal data points. Which is affecting rmse scores very much.

Also, from this assignment we can understand how much feature engineering is important in machine learning.

From the above analysis we can see all the highly miss-classified points are in the range below -30.

Future Work:

Since, we can see my model isn’t able distinguish rare points from the normal points very well, thus I would like to train one classification model before using any regression model. That will classify the rare points first then train 2 different regression models, One for normal points and another for rare points.