Elo Predicting Loyalty Score

1. Description

Imagine being hungry in an unfamiliar part of town and getting restaurant recommendations served up, based on your personal preferences, at just the right moment. The recommendation comes with an attached discount from your credit card provider for a local place around the corner!

Right now, Elo, one of the largest payment brands in Brazil, has built partnerships with merchants in order to offer promotions or discounts to cardholders. But do these promotions work for either the consumer or the merchant? Do customers enjoy their experience? Do merchants see repeat business? Personalization is key.

Elo has built machine learning models to understand the most important aspects and preferences in their customers' lifecycle, from food to shopping. But so far none of them is specifically tailored for an individual or profile. This is where you come in.

In this competition, Kagglers will develop algorithms to identify and serve the most relevant opportunities to individuals, by uncovering signal in customer loyalty. Your input will improve customers' lives and help Elo reduce unwanted campaigns, to create the right experience for customers.

2. Business Problem

2.1 Overview

Nowadays many of us use credit card for transaction. As the name suggests using credit card one can perform transaction even if there is insufficient balance in the Bank Account and the same amount needs to be paid in the next billing cycle by the credit cardholder. Although The basic operation is same for all the credit cards still to retain customers and make use of credit cards more , credit card providers offer various discount coupons and promotions occasionally for the eligible customers. This also helps the merchants to expand the business where the customers can spend the money.

Elo is one of the largest payment brands in Brazil, has built partnerships with merchants in order to offer promotions or discounts to cardholders.Elo wants to build machine learning models to understand the most important aspects and preferences in their customers' lifecycle, from food to shopping so that they can provide more relevant and personalized promotions to the cardholders.

So in summary, our task is to predict the customer loyalty score (a real number value) based on the transaction history. Which will help Elo to personalize the offers and discount coupons to check whether the promotions are working as expected for the consumer or the merchant. or customers are really enjoying their experiences and repeat business is also occurring for the merchants. Research-

2.2 Problem Statement

1. Predict loyalty score for a customer based on his/her purchasing history on credit card
2. Predicting loyalty score would be useful as it will help the credit card provider to offer discount coupons and various promotional offers.
3. This is useful as customer might spend more money to avail them which in returns expands the bisuness

2.3 Dataset source

> credit: Kaggle

https://www.kaggle.com/c/elo-merchant-category-recommendation/data (https://www.kaggle.com/c/elo-merchant-category-recommendation/data)

2.4 Useful links

https://www.researchgate.net/publication/335158533_Predicting_Customer_Loyalty_Using_Various_Regression_M (https://www.researchgate.net/publication/335158533_Predicting_Customer_Loyalty_Using_Various_Regression_

https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/77537 (https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/77537)

https://brendanhasz.github.io/2019/04/04/loyalty-prediction-2-features.html (https://brendanhasz.github.io/2019/04/04/loyalty-prediction-2-features.html)

https://medium.com/@charansai_k/predicting-loyalty-scores-for-elo-merchant-category-using-ml-models-fromscratch-to-deployment-2346b157b837 (https://medium.com/@charansai_k/predicting-loyalty-scores-for-elo-merchant-category-using-ml-models-fromscratch-to-deployment-2346b157b837)

2.5 Business Constraints

- Accuracy should be high , since low or medium accuracy can severely impact the business
- There is no strict low latency requirements , as predicting loyalty score needn't to be immediate and it's acceptable if it takes time
- Even though there is no strict requirement of interpretability still it is partially important , as it will help the card_provider to understand the purchasing behavior of the customers, based on which customers might receive promotional offers.

3. Data

3.1 Data Overview

We have total 5 csv files, train.csv - features are card details of traindataset,first_active_month,card_id, three categorical features - feature_1,feature_2,feature_3 and the target value

test.csv - features are same as train.csv only the target value doesn't present

historical_transactions.csv - It contains up to 3 months' worth of transactions for every card at any of the provided merchant_ids

new_merchant_transactions.csv - The features are same as historical_transactions.csv. However it contains the transactions at new merchants (merchant_ids that this particular card_id has not yet visited) over a period of two months.

and the last one is merchants.csv , It contains aggregate information for each merchant_id represented in the data set.

Along with the above, we have data dictionary also which explains each features.

3.2 Example Data Points

- "first_active_month" , "card_id" , "feature_1" , "feature_2" , "feature_3" , "target"

```
        2017-06 ,     C_ID_92a2005557 ,  5  ,             2 ,            1,
   ,-0.820283
```

- 'authorized_flag', 'card_id', 'city_id', 'category_1', 'installments','category_3', 'merchant_category_id', merchant_id', 'month_lag','purchase_amount', 'purchase_date', 'category_2', 'state_id','subsector_id'

Y ,C_ID_4e6213e9bc, 88, N, 0, A, 80, M_ID_e020e9b302, -8, -0.703331, 2017-06-25 15:33:07, 1.0, 16, 37

- merchant_id, merchant_group_id, merchant_category_id, subsector_id, numerical_1, numerical_2, category_1, most_recent_sales_range, most_recent_purchases_range, avg_sales_lag3, avg_sales_lag6, avg_purchases_lag6, active_months_lag6, avg_sales_lag12, avg_purchases_lag12, active_months_lag12, category_4, city_id, state_id, category_2

0, M_ID_838061e48c, 8353, 792, 9, -0.057471, -0.057471, N, E, E, -0.40, -2.25, 18.666667, 6, -2.32, 13.916667, 12, N, 242, 9, 1.0

4. Machine Learning Problem

4.1 Type

> Clearly this is a regression problem as we need to predict loyalty score which is the real number

4.2 Performance Metric

> we will use Root mean squared error (RMSE) as performance metric

4.3 Train-Test Construction

> We will split the data in 70-30 ratio for evaluating and developing the model before predict on the actual test dataset

5. Exploratory Data Analysis

In [89]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
led
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
ocker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis
t all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that get
s preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

```
/kaggle/input/elo-concat/datas.csv
/kaggle/input/elo-merchant-category-recommendation/new_merchant_transactions.
csv
/kaggle/input/elo-merchant-category-recommendation/merchants.csv
/kaggle/input/elo-merchant-category-recommendation/test.csv
/kaggle/input/elo-merchant-category-recommendation/sample_submission.csv
/kaggle/input/elo-merchant-category-recommendation/historical_transactions.cs
v
/kaggle/input/elo-merchant-category-recommendation/train.csv
/kaggle/input/elo-merchant-category-recommendation/Data_Dictionary.xlsx
/kaggle/input/elo-merchant-category-recommendation/Data Dictionary.xlsx
```

In [90]:
```
#import the required libraries
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [91]:
```
#Load the test dataset
test_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/test.
csv')
test_csv.columns
```

Out[91]:
```
Index(['first_active_month', 'card_id', 'feature_1', 'feature_2', 'feature_
3'], dtype='object')
```

In [92]:
```
#load the train dataset
train_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/train.csv')
train_csv.columns
```

Out[92]:
```
Index(['first_active_month', 'card_id', 'feature_1', 'feature_2', 'feature_3',
       'target'],
      dtype='object')
```

In [93]:
```
print("shape of train:",train_csv.shape)
print("shape of test:",test_csv.shape)
```

```
shape of train: (201917, 6)
shape of test: (123623, 5)
```

In [94]:
```
test_csv.head()
```

Out[94]:

	first_active_month	card_id	feature_1	feature_2	feature_3
0	2017-04	C_ID_0ab67a22ab	3	3	1
1	2017-01	C_ID_130fd0cbdd	2	3	0
2	2017-08	C_ID_b709037bc5	5	1	1
3	2017-12	C_ID_d27d835a9f	2	1	0
4	2015-12	C_ID_2b5e3df5c2	5	1	1

In [95]:
```
train_csv.head()
```

Out[95]:

	first_active_month	card_id	feature_1	feature_2	feature_3	target
0	2017-06	C_ID_92a2005557	5	2	1	-0.820283
1	2017-01	C_ID_3d0044924f	4	1	0	0.392913
2	2016-08	C_ID_d639edf6cd	2	2	0	0.688056
3	2017-09	C_ID_186d6a6901	4	3	0	0.142495
4	2017-11	C_ID_cdbd2c0db2	1	3	0	-0.159749

In [96]:
```
test_csv.describe()
```

Out[96]:

	feature_1	feature_2	feature_3
count	123623.00000	123623.000000	123623.000000
mean	3.10926	1.741796	0.564377
std	1.18911	0.749195	0.495840
min	1.00000	1.000000	0.000000
25%	2.00000	1.000000	0.000000
50%	3.00000	2.000000	1.000000
75%	4.00000	2.000000	1.000000
max	5.00000	3.000000	1.000000

In [97]:
```
train_csv.describe()
```

Out[97]:

	feature_1	feature_2	feature_3	target
count	201917.000000	201917.000000	201917.000000	201917.000000
mean	3.105311	1.745410	0.565569	-0.393636
std	1.186160	0.751362	0.495683	3.850500
min	1.000000	1.000000	0.000000	-33.219281
25%	2.000000	1.000000	0.000000	-0.883110
50%	3.000000	2.000000	1.000000	-0.023437
75%	4.000000	2.000000	1.000000	0.765453
max	5.000000	3.000000	1.000000	17.965068

In [98]:
```
#to check if there is any common card_id exists in both train and test dataset
duplicate_card_id=set(train_csv['card_id']).intersection(set(test_csv['card_i
d']))
print('duplicate_card_id:',duplicate_card_id)
```

```
duplicate_card_id: set()
```

Hence there is no common card_id in train and test dataset

In [99]:
```
#to check if any duplicate card_id present in train & test dataset
duplicate_train = train_csv[train_csv.duplicated(['card_id'])]
duplicate_test = test_csv[test_csv.duplicated(['card_id'])]
print("duplicate card_id in train:",duplicate_train)
print("duplicate in test:",duplicate_test)
```

```
duplicate card_id in train: Empty DataFrame
Columns: [first_active_month, card_id, feature_1, feature_2, feature_3, targe
t]
Index: []
duplicate in test: Empty DataFrame
Columns: [first_active_month, card_id, feature_1, feature_2, feature_3]
Index: []
```

Hence there is no duplicate entry in both train and test dataset based on card_id

In [100]:
```
#check whether NULL value present
print("*********************************************************")
print("For train.csv")
print("*********************************************************")
for i in train_csv.columns:
    if (train_csv[i].isnull().sum() > 0):
        print("Total NULL values in ", i ,":",train_csv[i].isnull().sum())

print("*********************************************************")
print("For test.csv")
print("*********************************************************")
for i in test_csv.columns:
    if (test_csv[i].isnull().sum() > 0):
        print("Total NULL values in ", i ,":",test_csv[i].isnull().sum())
```

```
*********************************************************
For train.csv
*********************************************************
*********************************************************
For test.csv
*********************************************************
Total NULL values in  first_active_month : 1
```

In [101]:
```
test_csv[test_csv['first_active_month'].isnull()]
```

Out[101]:

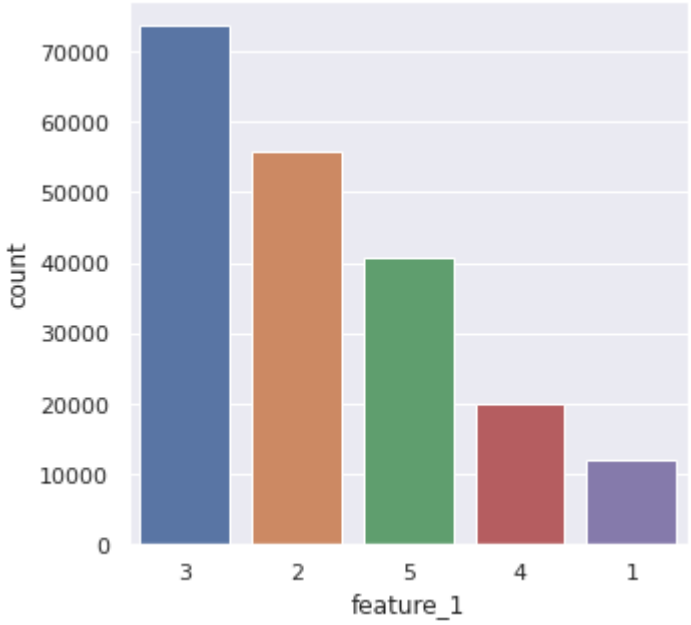
	first_active_month	card_id	feature_1	feature_2	feature_3
11578	NaN	C_ID_c27b4f80f7	5	2	1

Hence there is no Null values in train dataset , however in test dataset one null value exists for the first_active_month for the card_id C_ID_c27b4f80f7 .

we will check later what can be done with this Null value

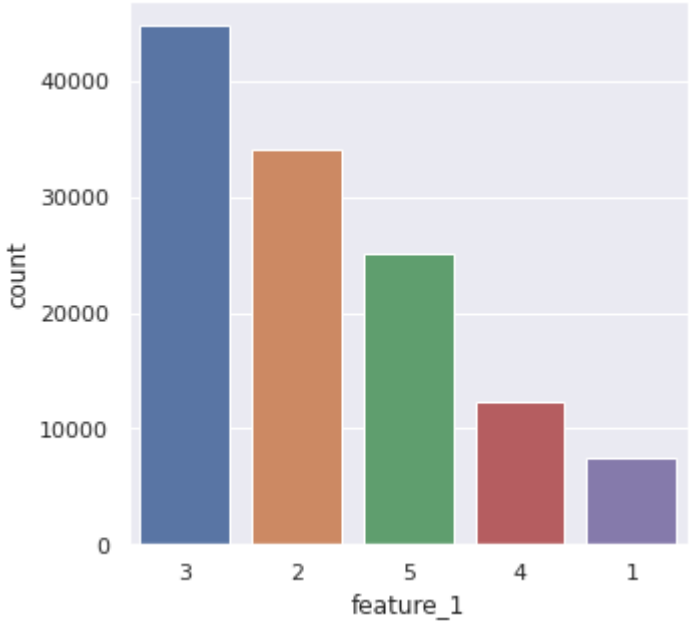
EDA - Plot feature_1 in both train test dataset

In [102]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="feature_1",data=train_csv,order=train_csv['feature_1'].val
ue_counts().index)
```



clearly in feature_1 , category 3 is dominating

In [103]:
```
sns.set_style(style="darkgrid")
#ax=sns.countplot(x="feature_1",data=test_csv)

ax=sns.countplot(x="feature_1",data=test_csv,order=test_csv['feature_1'].value
_counts().index)
#order=df_faculty['PhdDecade'].value_counts().index
```



In [104]:
```
print("Percentage of category_1 in train:",len(train_csv[train_csv['feature_1'] == 1])/len(train_csv['feature_1'])*100)
print("Percentage of category_1 in test:",len(test_csv[test_csv['feature_1'] == 1])/len(test_csv['feature_1'])*100)
print("Percentage of category_2 in train:",len(train_csv[train_csv['feature_1'] == 2])/len(train_csv['feature_1'])*100)
print("Percentage of category_2 in test:",len(test_csv[test_csv['feature_1'] == 2])/len(test_csv['feature_1'])*100)
print("Percentage of category_3 in train:",len(train_csv[train_csv['feature_1'] == 3])/len(train_csv['feature_1'])*100)
print("Percentage of category_3 in test:",len(test_csv[test_csv['feature_1'] == 3])/len(test_csv['feature_1'])*100)
print("Percentage of category_4 in train:",len(train_csv[train_csv['feature_1'] == 4])/len(train_csv['feature_1'])*100)
print("Percentage of category_4 in test:",len(test_csv[test_csv['feature_1'] == 4])/len(test_csv['feature_1'])*100)
print("Percentage of category_5 in train:",len(train_csv[train_csv['feature_1'] == 5])/len(train_csv['feature_1'])*100)
print("Percentage of category_5 in test:",len(test_csv[test_csv['feature_1'] == 5])/len(test_csv['feature_1'])*100)
```

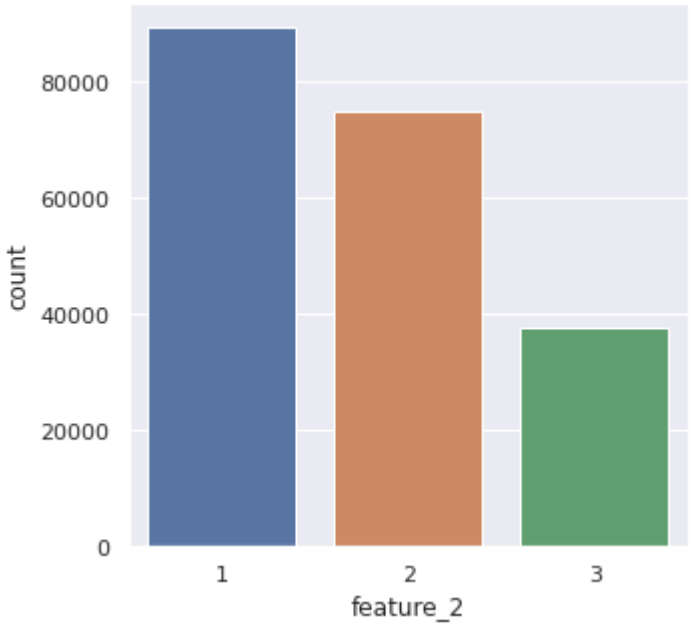
```
Percentage of category_1 in train: 5.961360360940386
Percentage of category_1 in test: 5.990794593239122
Percentage of category_2 in train: 27.633631640723667
Percentage of category_2 in test: 27.59599750855423
Percentage of category_3 in train: 36.43724896863563
Percentage of category_3 in test: 36.17368936201192
Percentage of category_4 in train: 9.84810590490152
Percentage of category_4 in test: 9.975489997815941
Percentage of category_5 in train: 20.119653124798802
Percentage of category_5 in test: 20.26402853837878
```

in test dataset also category-3 appears in most number of cases in feature_1. And the distribution is also almost same for train and test dataset.

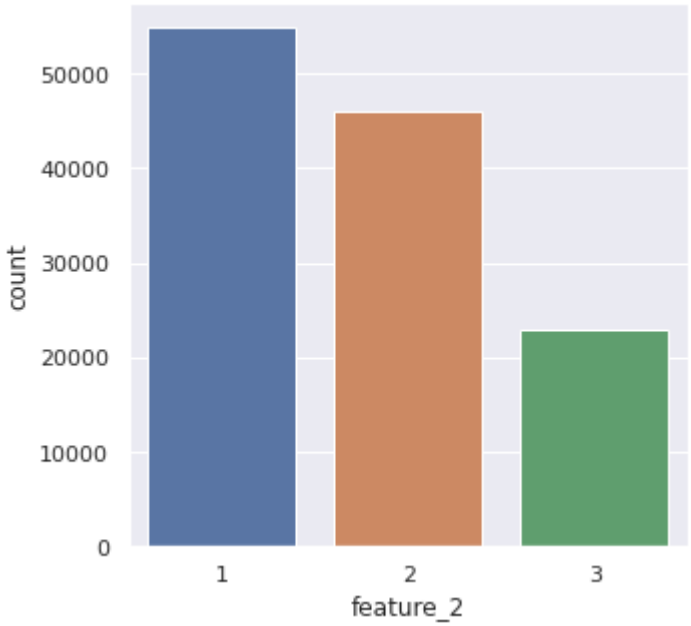
We will check the same counterplot for other two categorical features as well

plot feature_2:

In [105]:
```
sns.set_style(style="darkgrid")
ax=sns.countplot(x="feature_2",data=train_csv)
```



In [106]:
```
sns.set_style(style="darkgrid")
ax=sns.countplot(x="feature_2",data=test_csv)
```

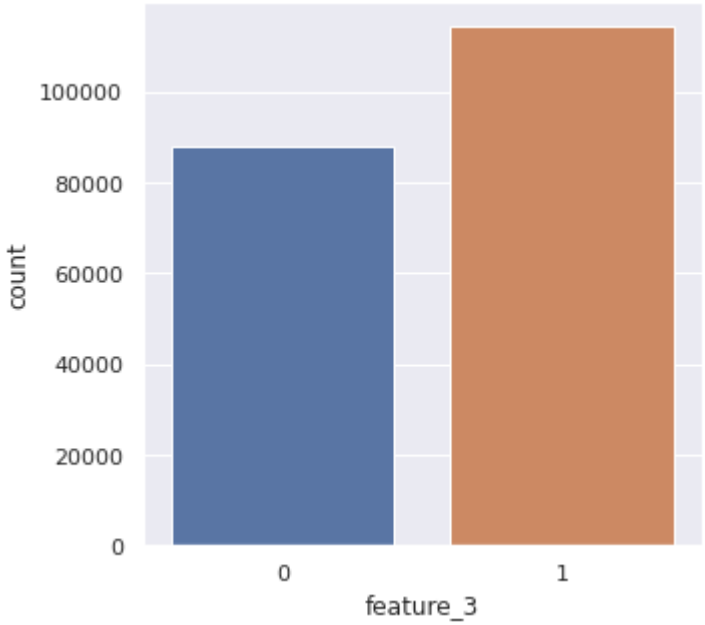


In [107]:
```
print("Percentage of category_1 in train:",len(train_csv[train_csv['feature_2'] == 1])/len(train_csv['feature_2'])*100)
print("Percentage of category_1 in test:",len(test_csv[test_csv['feature_2'] == 1])/len(test_csv['feature_2'])*100)
print("Percentage of category_2 in train:",len(train_csv[train_csv['feature_2'] == 2])/len(train_csv['feature_2'])*100)
print("Percentage of category_2 in test:",len(test_csv[test_csv['feature_2'] == 2])/len(test_csv['feature_2'])*100)
print("Percentage of category_3 in train:",len(train_csv[train_csv['feature_2'] == 3])/len(train_csv['feature_2'])*100)
print("Percentage of category_3 in test:",len(test_csv[test_csv['feature_2'] == 3])/len(test_csv['feature_2'])*100)
```

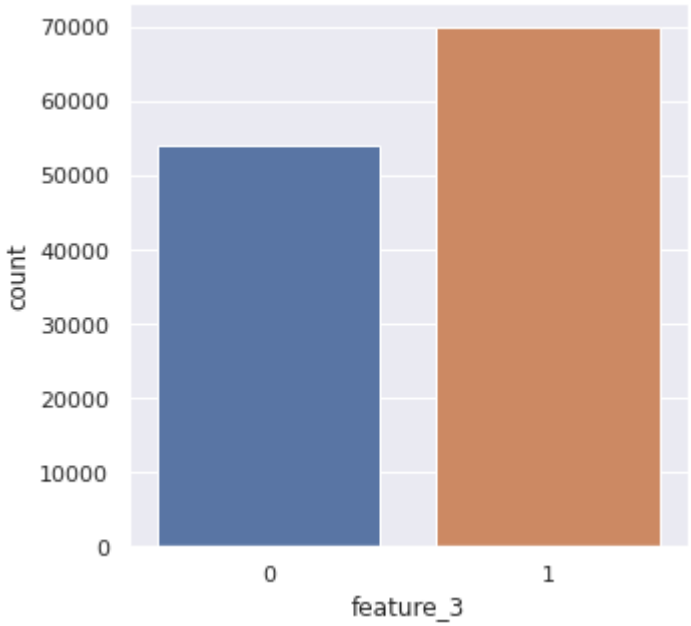
```
Percentage of category_1 in train: 44.19736822555802
Percentage of category_1 in test: 44.3080980076523
Percentage of category_2 in train: 37.06423926662936
Percentage of category_2 in test: 37.20424192909086
Percentage of category_3 in train: 18.738392507812616
Percentage of category_3 in test: 18.487660063256836
```

plot feature_3:

In [108]:
```
sns.set_style(style="darkgrid")
ax=sns.countplot(x="feature_3",data=train_csv)
```



In [109]:
```
sns.set_style(style="darkgrid")
ax=sns.countplot(x="feature_3",data=test_csv)
```



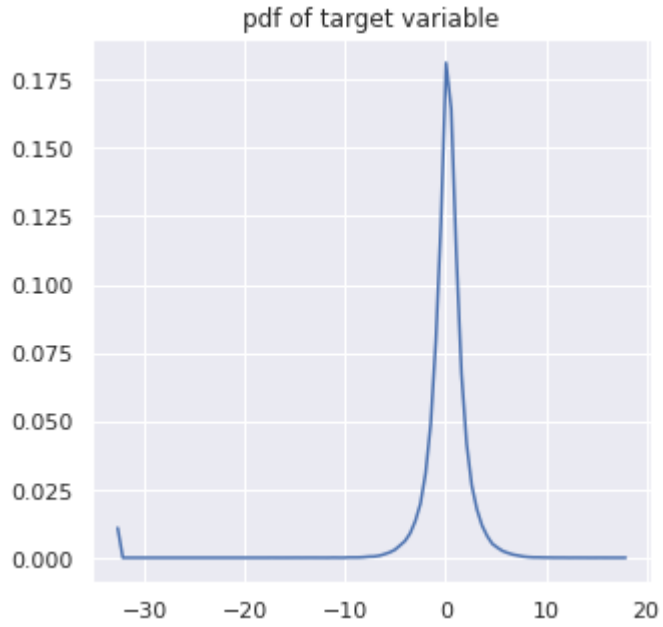
In [110]:
```
print("Percentage of category_0 in train:",len(train_csv[train_csv['feature_3'
] == 0])/len(train_csv['feature_3'])*100)
print("Percentage of category_0 in test:",len(test_csv[test_csv['feature_3'] =
= 0])/len(test_csv['feature_3'])*100)
print("Percentage of category_1 in train:",len(train_csv[train_csv['feature_3'
] == 1])/len(train_csv['feature_3'])*100)
print("Percentage of category_1 in test:",len(test_csv[test_csv['feature_3'] =
= 1])/len(test_csv['feature_3'])*100)
```

```
Percentage of category_0 in train: 43.44309790656557
Percentage of category_0 in test: 43.5622820996093
Percentage of category_1 in train: 56.55690209343444
Percentage of category_1 in test: 56.4377179003907
```

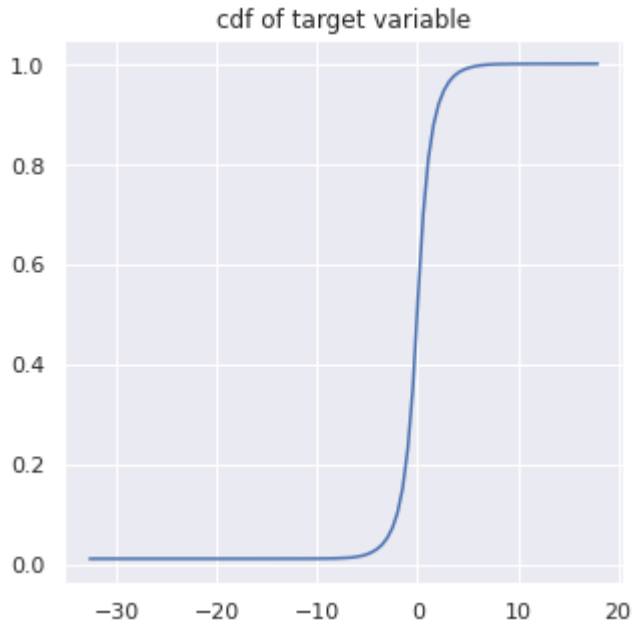
Conclusion: The distribution is almost similar for the three categorical features. We need to convert them using one hot encoding

plot target feature:

In [111]:
```
counts,bin_edges = np.histogram (train_csv['target'],bins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of target variable")
plt.show()
```

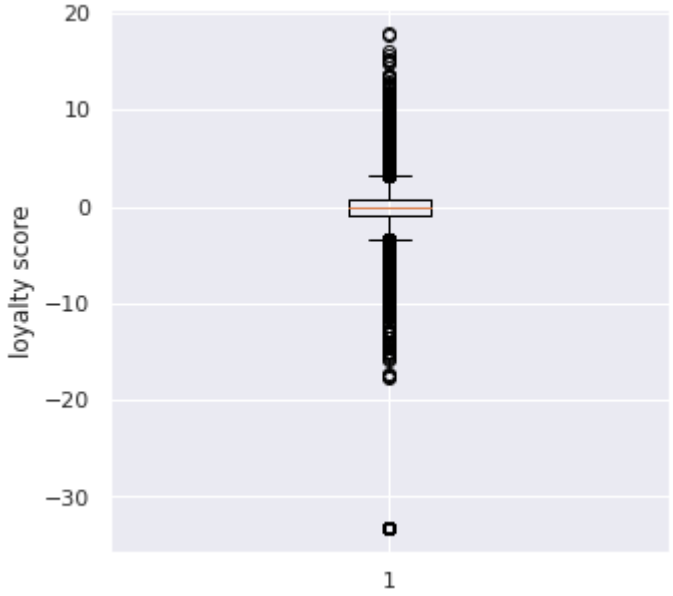


In [112]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of target variable")
plt.show()
```



In [113]:
```
plt.boxplot(train_csv['target'])
plt.ylabel("loyalty score")
```

Out[113]:
```
Text(0, 0.5, 'loyalty score')
```



from the pdf and cdf we could see maximum values present in between (-10,10) . To get more insightful we will look for the percentile data

In [114]:
```
print("The 10th percentile of loyalty score is :", np.percentile(train_csv['target'],[10]))
print("The 20th percentile of loyalty score is :", np.percentile(train_csv['target'],[20]))
print("The 30th percentile of loyalty score is :", np.percentile(train_csv['target'],[30]))
print("The 40th percentile of loyalty score is :", np.percentile(train_csv['target'],[40]))
print("The 50th percentile of loyalty score is :", np.percentile(train_csv['target'],[50]))
print("The 60th percentile of loyalty score is :", np.percentile(train_csv['target'],[60]))
print("The 70th percentile of loyalty score is :", np.percentile(train_csv['target'],[70]))
print("The 80th percentile of loyalty score is :", np.percentile(train_csv['target'],[80]))
print("The 90th percentile of loyalty score is :", np.percentile(train_csv['target'],[90]))
print("The 100th percentile of loyalty score is :", np.percentile(train_csv['target'],[100]))
```

```
The 10th percentile of loyalty score is : [-2.04229431]
The 20th percentile of loyalty score is : [-1.14604072]
The 30th percentile of loyalty score is : [-0.66396523]
The 40th percentile of loyalty score is : [-0.31219633]
The 50th percentile of loyalty score is : [-0.02343689]
The 60th percentile of loyalty score is : [0.23619123]
The 70th percentile of loyalty score is : [0.56450805]
The 80th percentile of loyalty score is : [1.01425502]
The 90th percentile of loyalty score is : [1.83028022]
The 100th percentile of loyalty score is : [17.9650684]
```

let's go for more deep

In [115]:
```
for i in range (1,11):
    print("The" ,i, "th percentile of loyalty score is :", np.percentile(train_csv['target'],[i]))
```

```
The 1 th percentile of loyalty score is : [-33.21928095]
The 2 th percentile of loyalty score is : [-5.01607721]
The 3 th percentile of loyalty score is : [-4.06320537]
The 4 th percentile of loyalty score is : [-3.49365557]
The 5 th percentile of loyalty score is : [-3.10783563]
The 6 th percentile of loyalty score is : [-2.8080848]
The 7 th percentile of loyalty score is : [-2.56256506]
The 8 th percentile of loyalty score is : [-2.36225974]
The 9 th percentile of loyalty score is : [-2.19336072]
The 10 th percentile of loyalty score is : [-2.04229431]
```

if we check for the minimum value , it seems around 1% of data have very less loyalty score -30

In [116]:
```
for i in range (80,101):
    print("The" ,i, "th percentile of loyalty score is :", np.percentile(train_csv['target'],[i]))
```

```
The 80 th percentile of loyalty score is : [1.01425502]
The 81 th percentile of loyalty score is : [1.07173898]
The 82 th percentile of loyalty score is : [1.13227365]
The 83 th percentile of loyalty score is : [1.19754612]
The 84 th percentile of loyalty score is : [1.26898947]
The 85 th percentile of loyalty score is : [1.34520843]
The 86 th percentile of loyalty score is : [1.42507187]
The 87 th percentile of loyalty score is : [1.51228911]
The 88 th percentile of loyalty score is : [1.60639673]
The 89 th percentile of loyalty score is : [1.71256874]
The 90 th percentile of loyalty score is : [1.83028022]
The 91 th percentile of loyalty score is : [1.95925439]
The 92 th percentile of loyalty score is : [2.10680338]
The 93 th percentile of loyalty score is : [2.28343475]
The 94 th percentile of loyalty score is : [2.47487263]
The 95 th percentile of loyalty score is : [2.70268023]
The 96 th percentile of loyalty score is : [2.99319449]
The 97 th percentile of loyalty score is : [3.36769594]
The 98 th percentile of loyalty score is : [3.87841299]
The 99 th percentile of loyalty score is : [4.812662]
The 100 th percentile of loyalty score is : [17.9650684]
```

similarly if we go for the max loyalty score , then around 1% data have extreme high loyalty score <5. If they are outlier then they will create issue if we perform linear regression.

We will check later in feature engineering steps if we need to eliminate them or replace with other values or we can keep them as it is..

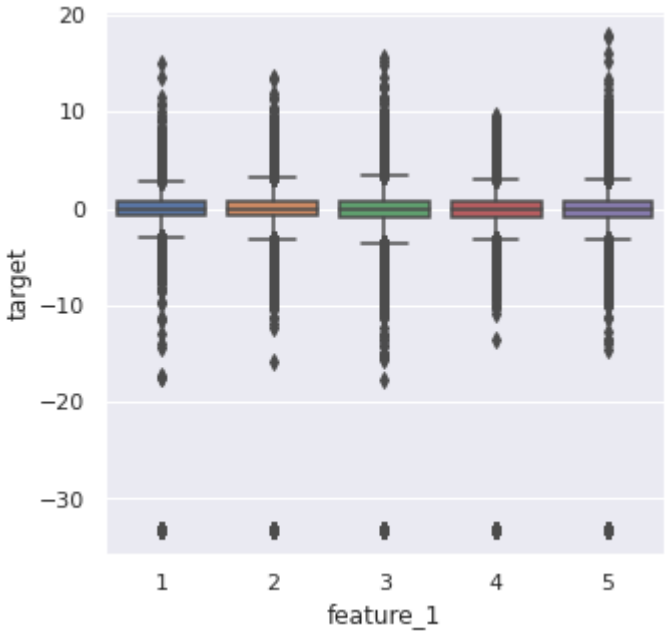
In [117]:
```
min(train_csv['target'])
```

Out[117]: -33.21928095

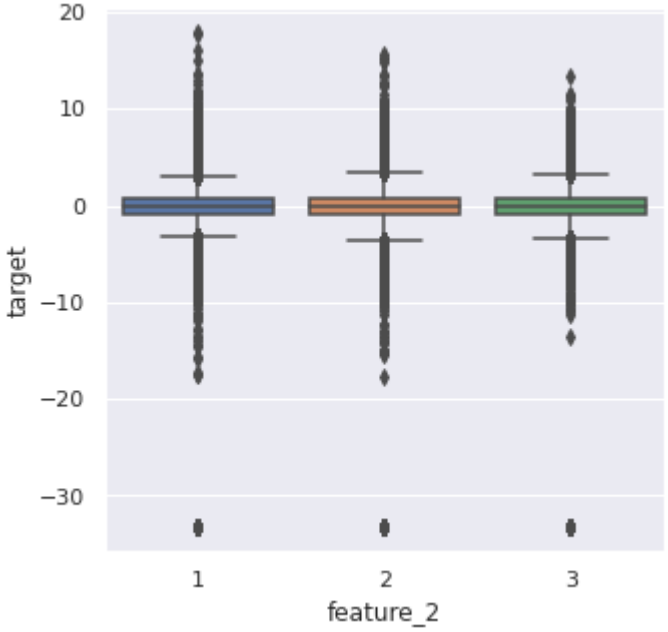
our next target is , we will check if we can plot each features against loyalty score and plot histogram of date

In [118]:
```
import matplotlib as mpl
mpl.rcParams['agg.path.chunksize'] = 10000000
```

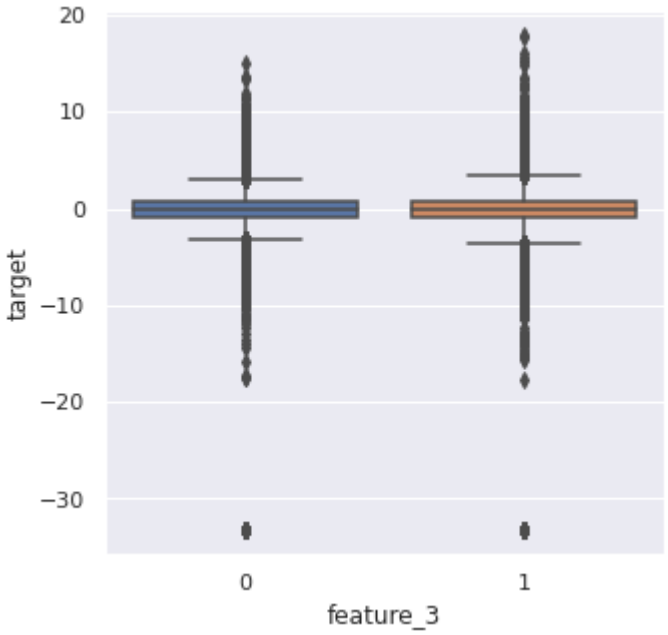
In [119]:
```
ax=sns.boxplot(x="feature_1",y="target",data=train_csv)
```



In [120]:
```
ax=sns.boxplot(x="feature_2",y="target",data=train_csv)
```



In [121]:
```
ax=sns.boxplot(x="feature_3",y="target",data=train_csv)
```



Unfortunately none of the plots are clear, so we can't conclude anything just by using the categorical features (feature_1,feature_2,feature_3) as they are not worthy

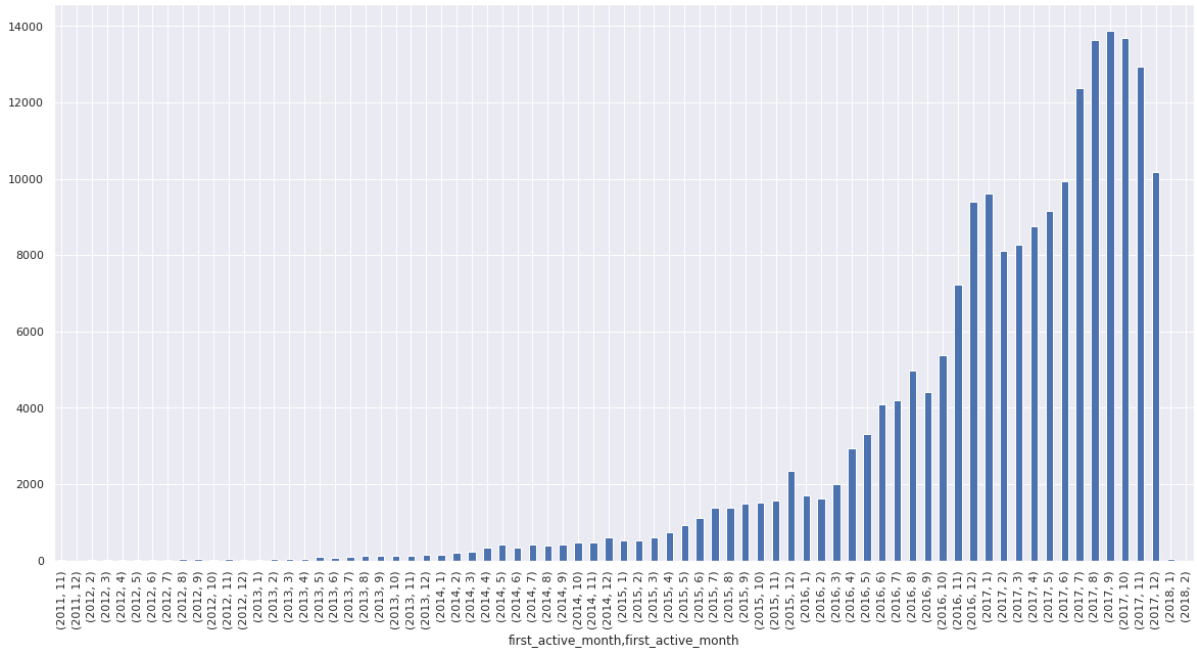
plot first_active_month

As the data type is object , we will convert it to pandas datetime format so that we can perform operations into it

In [122]:
```
# convert the datatype of first_active_month to datetime
test_csv['first_active_month']=pd.to_datetime(train_csv['first_active_month'],
format='%Y-%m')
train_csv['first_active_month']=pd.to_datetime(train_csv['first_active_month'
],format='%Y-%m')
```

In [123]:
```
#ref: https://stackoverflow.com/questions/27365467/can-pandas-plot-a-histogram-of-dates
%matplotlib inline
sns.set(rc={'figure.figsize':(20,10)})
train_csv['first_active_month'].groupby([train_csv['first_active_month'].dt.year, train_csv['first_active_month'].dt.month]).count().plot(kind="bar")
```

Out[123]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5676a9f3d0>
```

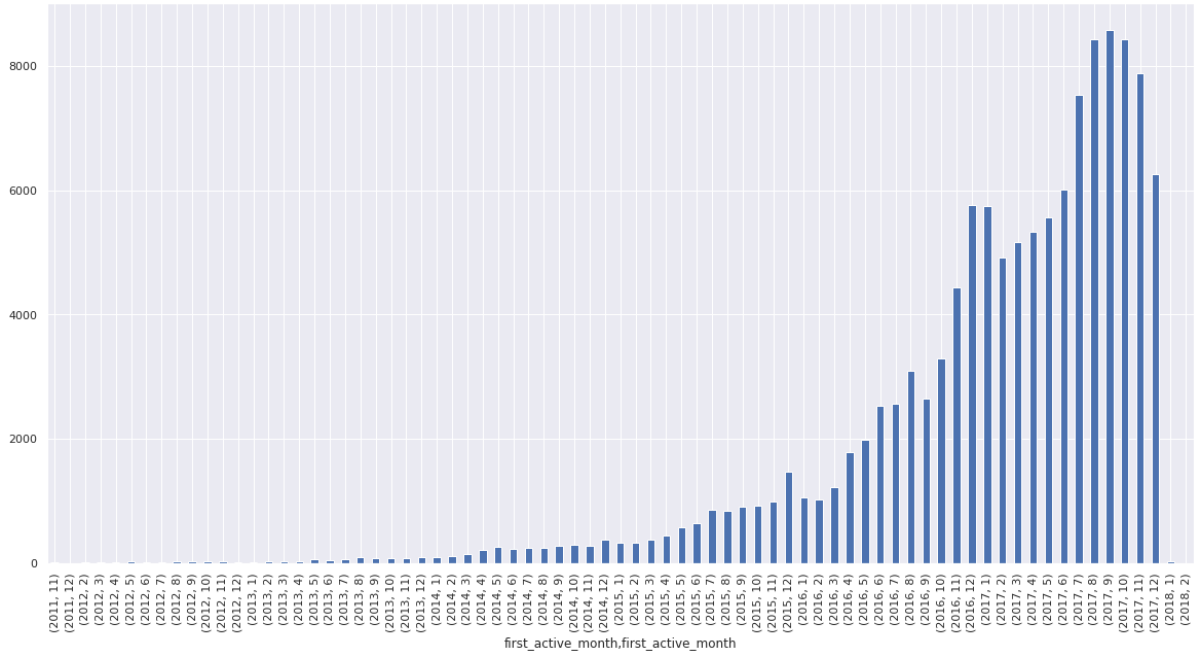


As the time passes by, the count also increases . But there is a sudden drop after Jan'2017. Also one interesting fact is we have data till Jan'2018 only and that number is also very less. And most of the date belongs to 2016 to 2018

let's check for test data

In [124]:
```
test_csv['first_active_month'].groupby([test_csv['first_active_month'].dt.year
, test_csv['first_active_month'].dt.month]).count().plot(kind="bar")
```

Out[124]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f55bc17ddd0>
```



And same pattern continues in test dataset as well. so we have more recent data than historical (past) . It would be difficult for us to understand tif we had less recent data in comparison with past one.

In [125]:
```
#load historical_transactions
historical_transactions=pd.read_csv('/kaggle/input/elo-merchant-category-recom
mendation/historical_transactions.csv')
historical_transactions.columns
```

Out[125]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'category_3', 'merchant_category_id', 'merchant_id', 'month_lag',
       'purchase_amount', 'purchase_date', 'category_2', 'state_id',
       'subsector_id'],
      dtype='object')
```

In [126]:
```
#load new_merchant_transactions
new_merchant_transactions=pd.read_csv('/kaggle/input/elo-merchant-category-rec
ommendation/new_merchant_transactions.csv')
new_merchant_transactions.columns
```

Out[126]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'category_3', 'merchant_category_id', 'merchant_id', 'month_lag',
       'purchase_amount', 'purchase_date', 'category_2', 'state_id',
       'subsector_id'],
      dtype='object')
```

In [127]:
```
print("shape of historical_transaction:",historical_transactions.shape)
print("shape of new_merchant_transactions: ",new_merchant_transactions.shape)
```

```
shape of historical_transaction: (29112361, 14)
shape of new_merchant_transactions:  (1963031, 14)
```

In [128]:
```
historical_transactions.head(10)
```

Out[128]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_cat
0	Y	C_ID_4e6213e9bc	88	N	0	A	
1	Y	C_ID_4e6213e9bc	88	N	0	A	
2	Y	C_ID_4e6213e9bc	88	N	0	A	
3	Y	C_ID_4e6213e9bc	88	N	0	A	
4	Y	C_ID_4e6213e9bc	88	N	0	A	
5	Y	C_ID_4e6213e9bc	333	N	0	A	
6	Y	C_ID_4e6213e9bc	88	N	0	A	
7	Y	C_ID_4e6213e9bc	3	N	0	A	
8	Y	C_ID_4e6213e9bc	88	N	0	A	
9	Y	C_ID_4e6213e9bc	88	N	0	A	

In [129]:
```
new_merchant_transactions.head(10)
```

Out[129]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_cat
0	Y	C_ID_415bb3a509	107	N	1	B	
1	Y	C_ID_415bb3a509	140	N	1	B	
2	Y	C_ID_415bb3a509	330	N	1	B	
3	Y	C_ID_415bb3a509	-1	Y	1	B	
4	Y	C_ID_ef55cf8d4b	-1	Y	1	B	
5	Y	C_ID_ef55cf8d4b	231	N	1	B	
6	Y	C_ID_ef55cf8d4b	69	N	1	B	
7	Y	C_ID_ef55cf8d4b	231	N	1	B	
8	Y	C_ID_ef55cf8d4b	69	N	1	B	
9	Y	C_ID_ef55cf8d4b	69	N	1	B	

In [130]:
```
historical_transactions.describe()
```

Out[130]:

	city_id	installments	merchant_category_id	month_lag	purchase_amount	
count	2.911236e+07	2.911236e+07	2.911236e+07	2.911236e+07	2.911236e+07	2.6
mean	1.293256e+02	6.484954e-01	4.810130e+02	-4.487294e+00	3.640090e-02	2.1
std	1.042563e+02	2.795577e+00	2.493757e+02	3.588800e+00	1.123522e+03	1.5
min	-1.000000e+00	-1.000000e+00	-1.000000e+00	-1.300000e+01	-7.469078e-01	1.0
25%	5.300000e+01	0.000000e+00	3.070000e+02	-7.000000e+00	-7.203559e-01	1.0
50%	9.000000e+01	0.000000e+00	4.540000e+02	-4.000000e+00	-6.883495e-01	1.0
75%	2.120000e+02	1.000000e+00	7.050000e+02	-2.000000e+00	-6.032543e-01	3.0
max	3.470000e+02	9.990000e+02	8.910000e+02	0.000000e+00	6.010604e+06	5.0

In [131]:
```
new_merchant_transactions.describe()
```

Out[131]:

	city_id	installments	merchant_category_id	month_lag	purchase_amount	c
count	1.963031e+06	1.963031e+06	1.963031e+06	1.963031e+06	1.963031e+06	1.8
mean	1.343867e+02	6.829643e-01	4.309701e+02	1.476515e+00	-5.509690e-01	2.1
std	1.015152e+02	1.584069e+00	2.463385e+02	4.994483e-01	6.940043e-01	1.5
min	-1.000000e+00	-1.000000e+00	-1.000000e+00	1.000000e+00	-7.468928e-01	1.0
25%	6.900000e+01	0.000000e+00	2.780000e+02	1.000000e+00	-7.166294e-01	1.0
50%	1.100000e+02	1.000000e+00	3.670000e+02	1.000000e+00	-6.748406e-01	1.0
75%	2.120000e+02	1.000000e+00	6.830000e+02	2.000000e+00	-5.816162e-01	3.0
max	3.470000e+02	9.990000e+02	8.910000e+02	2.000000e+00	2.631575e+02	5.0

However, describe() function is not so useful , as most of them have categorical features instead of numerical features.

we will quickly check for the duplicate & NaN value.

In [132]:
```
print("duplicate card_id in historical_transactions:",historical_transactions.
pivot_table(index=['card_id'], aggfunc='size'))
print("duplicate merchant_id in historical_transactions:",historical_transacti
ons.pivot_table(index=['merchant_id'], aggfunc='size'))
print("duplicate card_id in new_merchant_transactions:",new_merchant_transacti
ons.pivot_table(index=['card_id'], aggfunc='size'))
print("duplicate merchant_id in new_merchant_transactions:",new_merchant_trans
actions.pivot_table(index=['merchant_id'], aggfunc='size'))
```

```
duplicate card_id in historical_transactions: card_id
C_ID_00007093c1    149
C_ID_0001238066    123
C_ID_0001506ef0     66
C_ID_0001793786    216
C_ID_000183fdda    144
                   ...
C_ID_ffff1d9928     12
C_ID_ffff579d3a    114
C_ID_ffff756266     24
C_ID_ffff828181    190
C_ID_fffffd5772     84
Length: 325540, dtype: int64
duplicate merchant_id in historical_transactions: merchant_id
M_ID_000025127f     9
M_ID_0000699140    57
M_ID_00006a5552     1
M_ID_000087311e    22
M_ID_0000ab0b2d    30
                   ..
M_ID_fffeeb852d    21
M_ID_fffef87522     2
M_ID_ffff0af8e7    26
M_ID_ffff655e2c     3
M_ID_ffffc28eaa    17
Length: 326311, dtype: int64
duplicate card_id in new_merchant_transactions: card_id
C_ID_00007093c1     2
C_ID_0001238066    26
C_ID_0001506ef0     2
C_ID_0001793786    31
C_ID_000183fdda    11
                   ..
C_ID_ffff1d9928     4
C_ID_ffff579d3a     1
C_ID_ffff756266     1
C_ID_ffff828181     8
C_ID_fffffd5772     3
Length: 290001, dtype: int64
duplicate merchant_id in new_merchant_transactions: merchant_id
M_ID_000025127f     1
M_ID_0000699140     5
M_ID_000087311e     3
M_ID_0000ab0b2d     3
M_ID_0000edb21f    17
                   ..
M_ID_fffd4df9cd    31
M_ID_fffe6eb555     2
M_ID_fffee4321d     2
M_ID_fffeeb852d     2
M_ID_ffff0af8e7     5
Length: 226129, dtype: int64
```

The result is expected , as same card_id can perform multiple transactions similarly same merchant_id can record multiple transactions.

So we will check for the entire duplicate row.

In [133]:
```
print("duplicate rows in historical_transactions:",historical_transactions[his
torical_transactions.duplicated(subset=None, keep='first')])
```

```
duplicate rows in historical_transactions: Empty DataFrame
Columns: [authorized_flag, card_id, city_id, category_1, installments, catego
ry_3, merchant_category_id, merchant_id, month_lag, purchase_amount, purchase
_date, category_2, state_id, subsector_id]
Index: []
```

In [134]:
```
print("duplicate rows in new_merchant_transactions:",new_merchant_transactions
[new_merchant_transactions.duplicated(subset=None, keep='first')])
```

```
duplicate rows in new_merchant_transactions: Empty DataFrame
Columns: [authorized_flag, card_id, city_id, category_1, installments, catego
ry_3, merchant_category_id, merchant_id, month_lag, purchase_amount, purchase
_date, category_2, state_id, subsector_id]
Index: []
```

So , in both transactions dataset , no duplicate record present

In [135]:
```
# check whether NULL value present

print("*******************************************************")
print("For historical_transactions")
print("*******************************************************")
for i in historical_transactions.columns:
    if(historical_transactions[i].isnull().sum() > 0):
        print("Total NULL values in ", i ,":",historical_transactions[i].isnul
l().sum())
```

```
*******************************************************
For historical_transactions
*******************************************************
Total NULL values in  category_3 : 178159
Total NULL values in  merchant_id : 138481
Total NULL values in  category_2 : 2652864
```

As few NULL values present , we will check the total percentage for each features

In [136]:
```
for i in ['category_3','merchant_id','category_2']:
    print("Percentage of NULL values in ",i,":",historical_transactions[i].isn
ull().sum()/len(historical_transactions[i])*100)
```

```
Percentage of NULL values in  category_3 : 0.6119702898710276
Percentage of NULL values in  merchant_id : 0.4756776683278969
Percentage of NULL values in  category_2 : 9.1125003568072
```

In [137]:
```
print("*****************************************************")
print("For new merchant transactions")
print("*****************************************************")
for i in new_merchant_transactions.columns:
    if(new_merchant_transactions[i].isnull().sum() > 0):
        print("Total NULL values in ", i ,":",new_merchant_transactions[i].isn
ull().sum())
```

```
*****************************************************
For new merchant transactions
*****************************************************
Total NULL values in  category_3 : 55922
Total NULL values in  merchant_id : 26216
Total NULL values in  category_2 : 111745
```

In [138]:
```
for i in ['category_3','merchant_id','category_2']:
    print("Percentage of NULL values in ",i,":",new_merchant_transactions[i].i
snull().sum()/len(new_merchant_transactions[i])*100)
```

```
Percentage of NULL values in  category_3 : 2.8487578647509895
Percentage of NULL values in  merchant_id : 1.3354857870303627
Percentage of NULL values in  category_2 : 5.692472508075522
```

In [139]:
```
# to check if there is any common entry for [city_id  merchant_id]. since as p
er the data definition there shouldn't be any
temp1=historical_transactions
temp2=new_merchant_transactions
temp1.merge(temp2,on=['card_id','merchant_id'])
```

Out[139]:

authorized_flag_x	card_id	city_id_x	category_1_x	installments_x	category_3_x	merchant_cat

0 rows × 26 columns

In [140]:
```
del temp1,temp2
```

In [141]:
```
#check if all the card_ids present in train and test dataset are also present
 in transactions dataset

card_id_train_test=set(train_csv['card_id']).union(set(test_csv['card_id']))
card_id_transactions=set(new_merchant_transactions['card_id']).union(set(histo
rical_transactions['card_id']))
card_id_difference=card_id_train_test.difference(card_id_transactions)
print("card_id_difference:",card_id_difference)
```

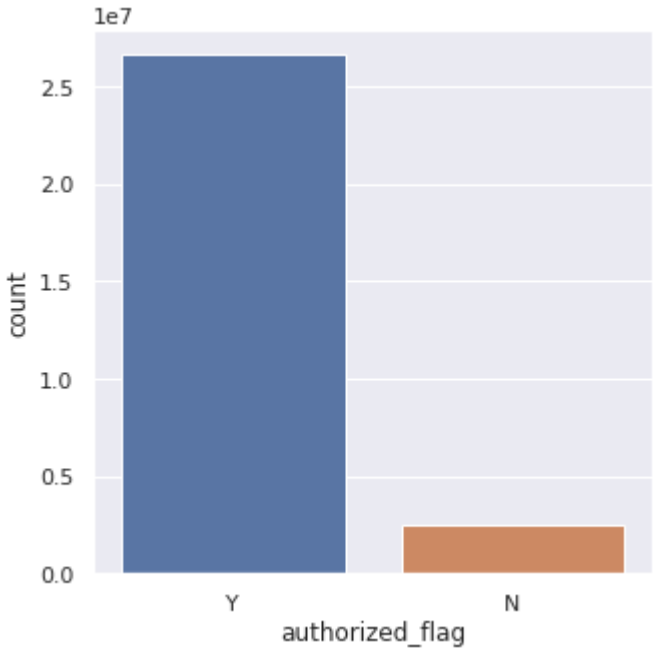
```
card_id_difference: set()
```

As it returns empty set , so there is no such card_id for which we don't have any record in transaction dataset.

Now we can directly perform the univariate analysis

plot authorized_flag

In [142]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="authorized_flag",data=historical_transactions)
```



Looks like most of the entries are Authorized, we will check the percentage

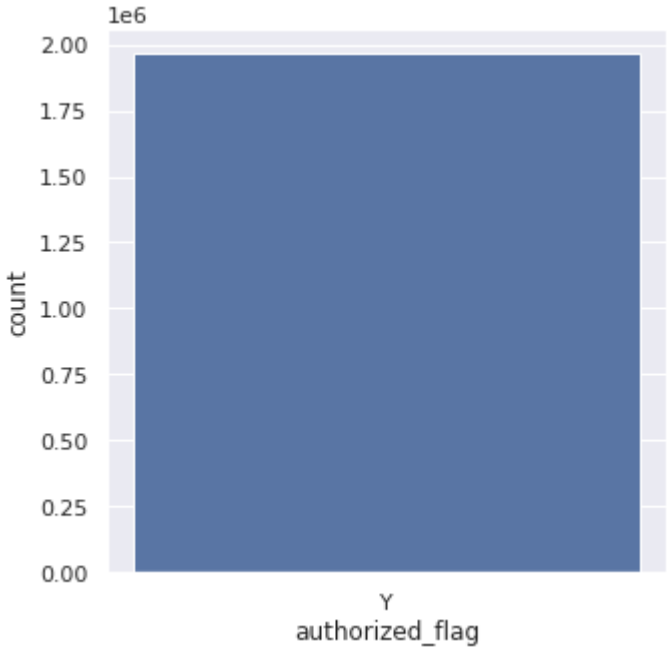
In [143]:
```
result=historical_transactions[historical_transactions['authorized_flag'] ==
'N']
percentage_Y=len(result)/len(historical_transactions['authorized_flag'])
print("Percentage of unauthorized transactions:",percentage_Y*100)
```

```
Percentage of unauthorized transactions: 8.64549941517969
```

So around 9% data are from unauthorized transactions which is very less. We will check in feature Engineering steps if we need to keep them.

Now check the same for new_merchant_transactions

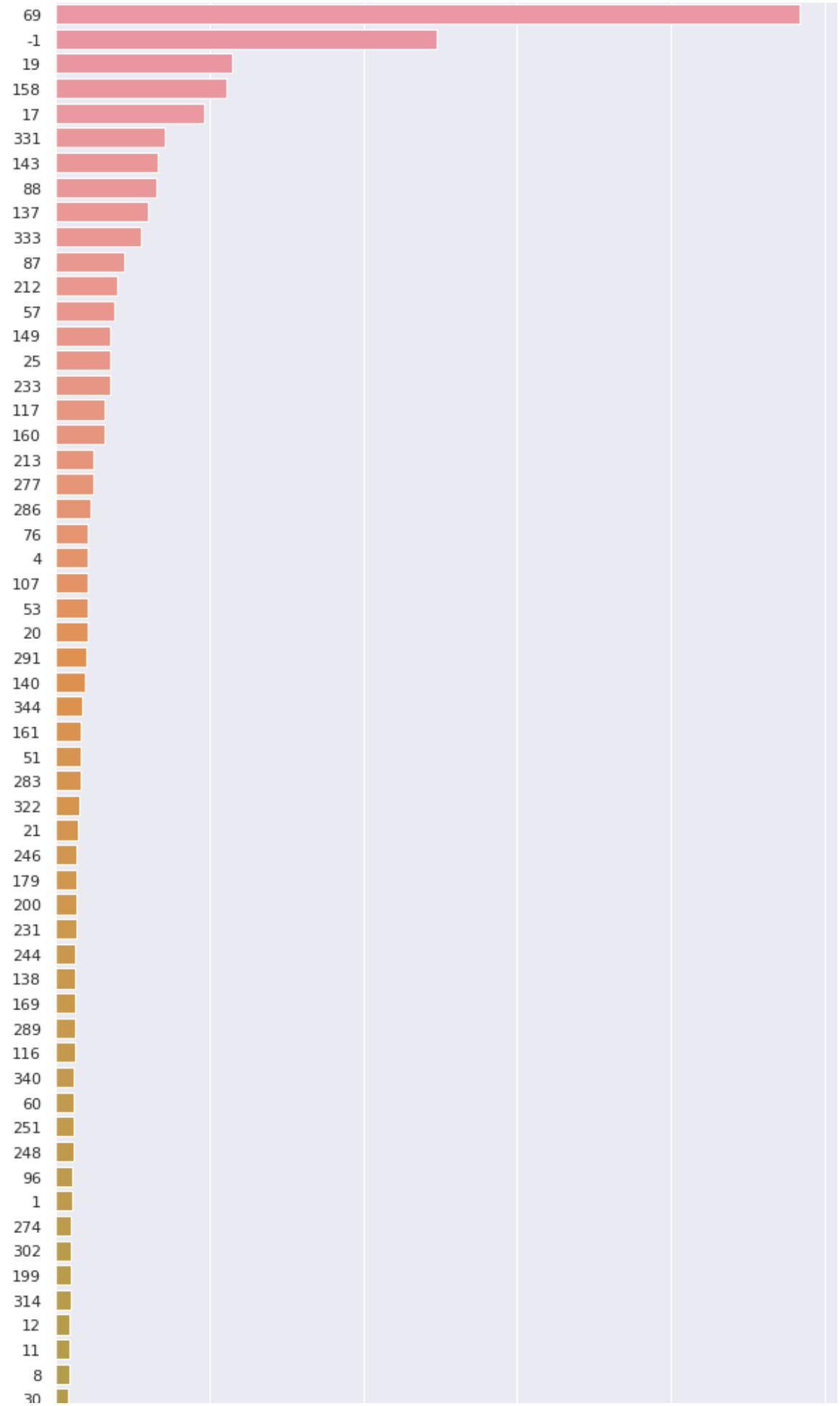
In [144]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="authorized_flag",data=new_merchant_transactions)
```

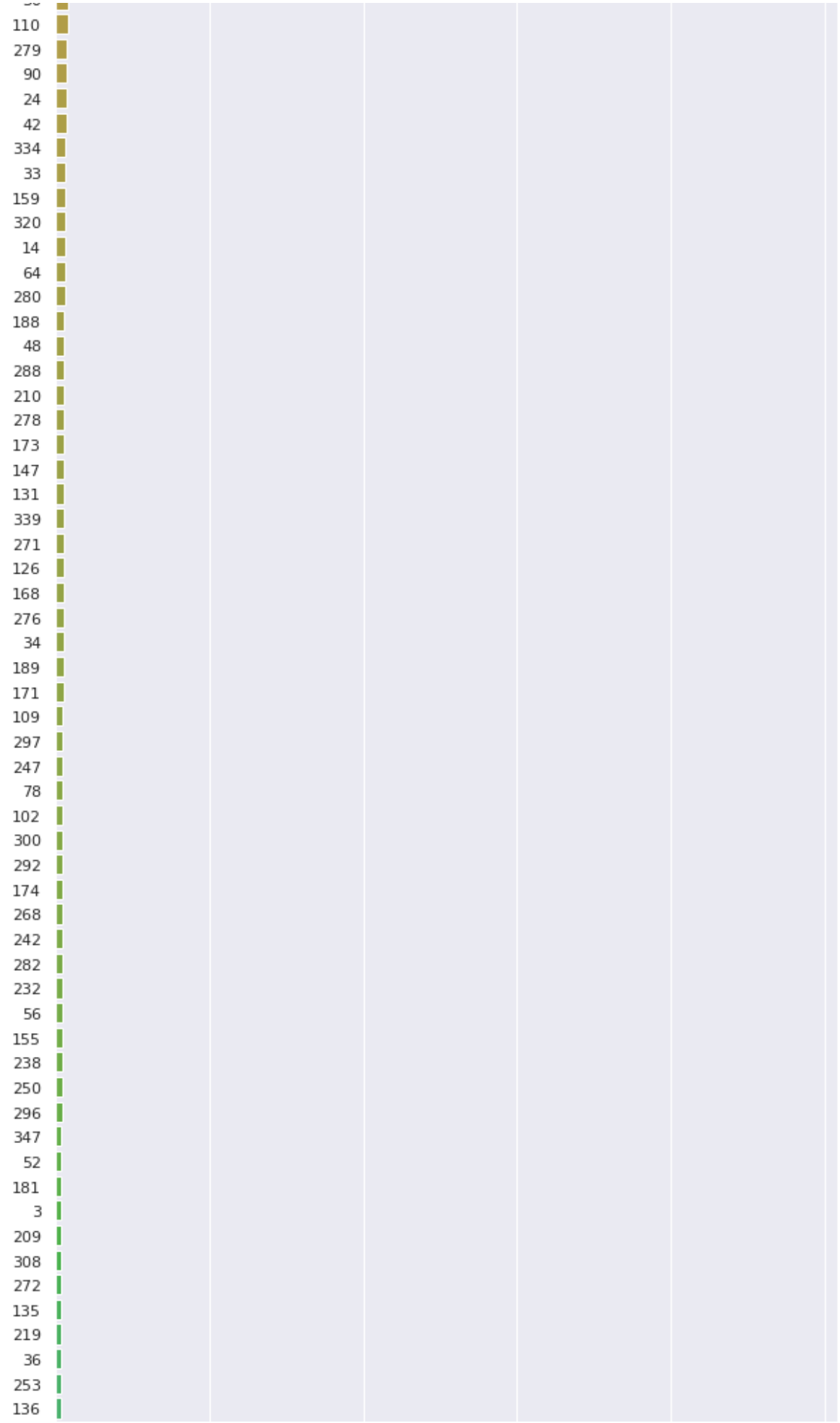


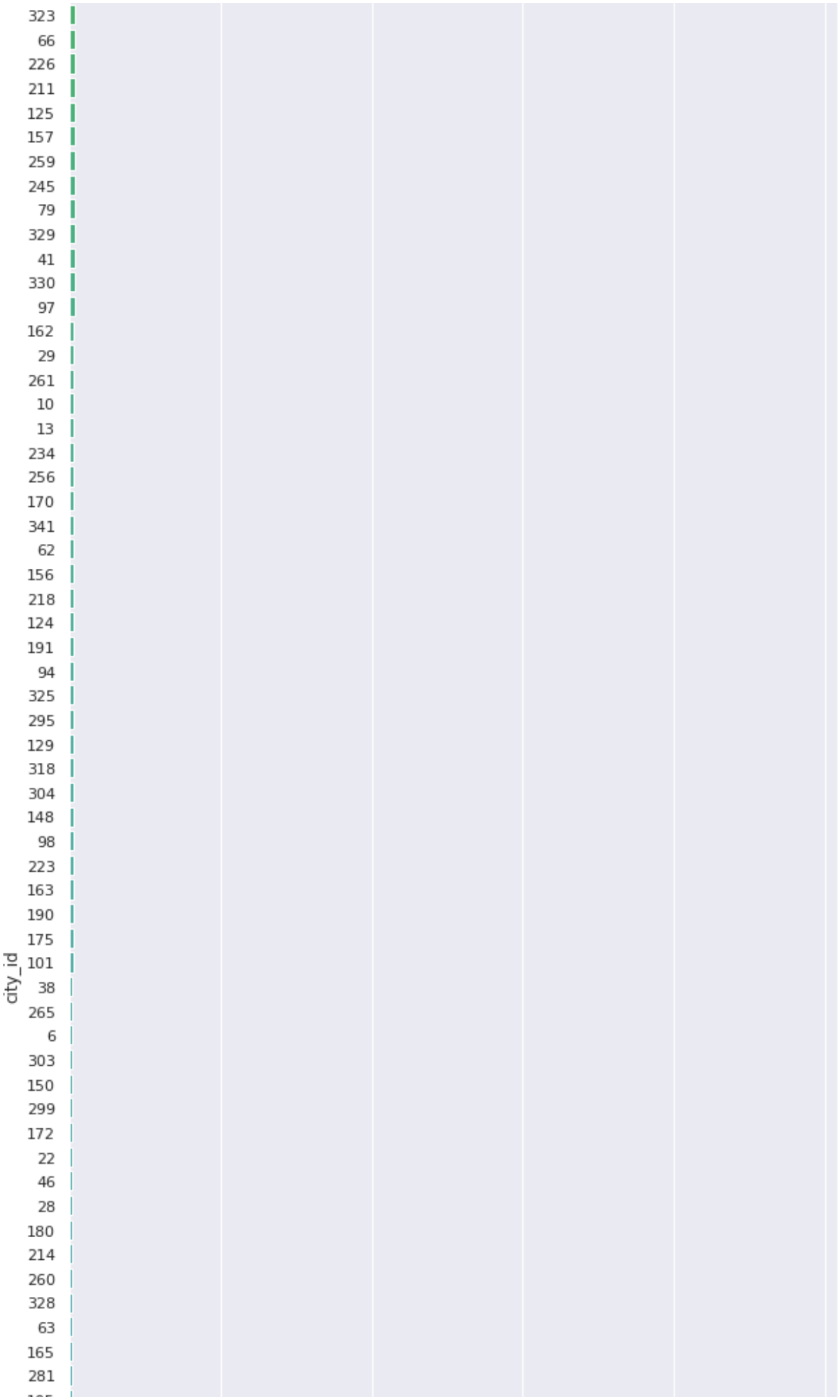
here all of the transactions are authorized

Plot city_id

In [259]:
```
sns.set(rc={'figure.figsize':(10,100)})
sns.set_style(style="darkgrid")
ax=sns.countplot(y="city_id",data=historical_transactions,order=historical_tra
nsactions['city_id'].value_counts().index)
```

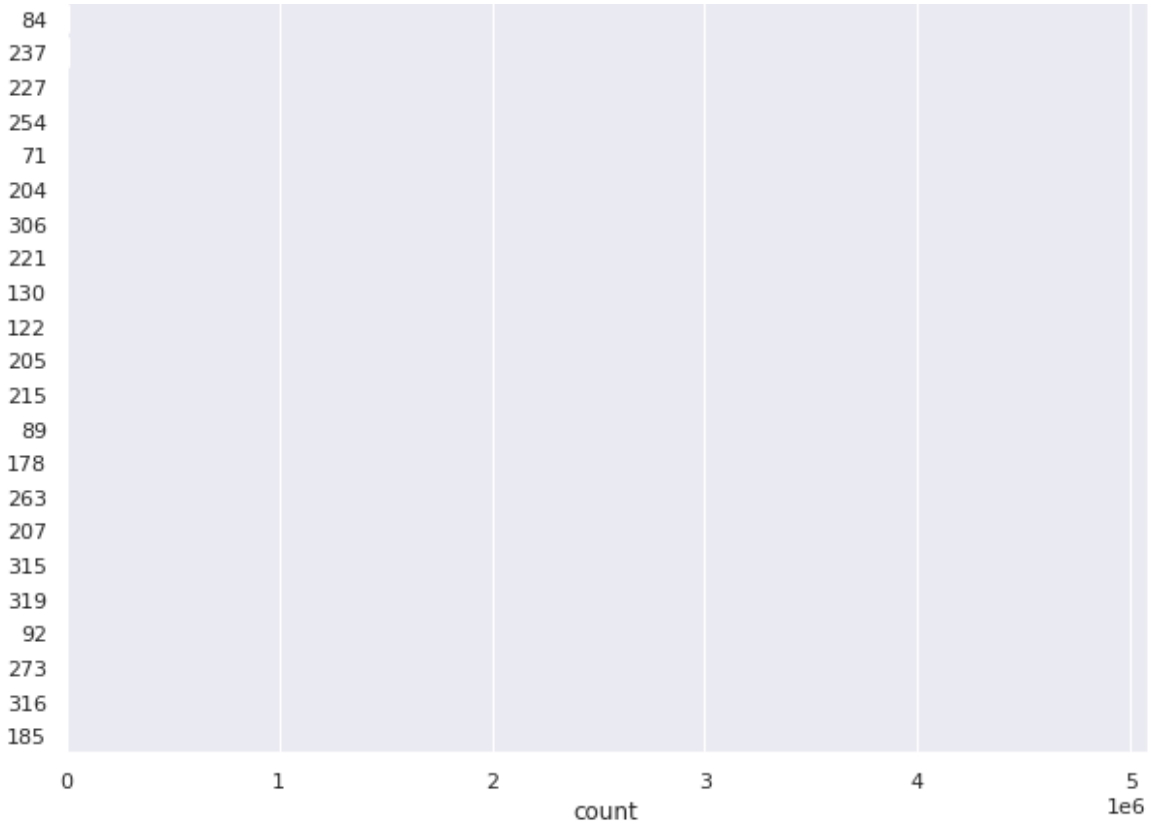

69
-1
19
158
17
331
143
88
137
333
87
212
57
149
25
233
117
160
213
277
286
76
4
107
53
20
291
140
344
161
51
283
322
21
246
179
200
231
244
138
169
289
116
340
60
251
248
96
1
274
302
199
314
12
11
8
30


110
279
90
24
42
334
33
159
320
14
64
280
188
48
288
210
278
173
147
131
339
271
126
168
276
34
189
171
109
297
247
78
102
300
292
174
268
242
282
232
56
155
238
250
296
347
52
181
3
209
308
272
135
219
36
253
136


city_id
323
66
226
211
125
157
259
245
79
329
41
330
97
162
29
261
10
13
234
256
170
341
62
156
218
124
191
94
325
295
129
318
304
148
98
223
163
190
175
101
38
265
6
303
150
299
172
22
46
28
180
214
260
328
63
165
281

105
187
7
23
123
193
301
9
270
186
119
111
229
311
115
82
195
167
100
216
145
255
146
275
75
239
91
337
128
266
153
182
201
74
194
220
342
197
183
198
166
2
58
184
313
68
70
203
49
230
312
294
83
120
144
139
267
290

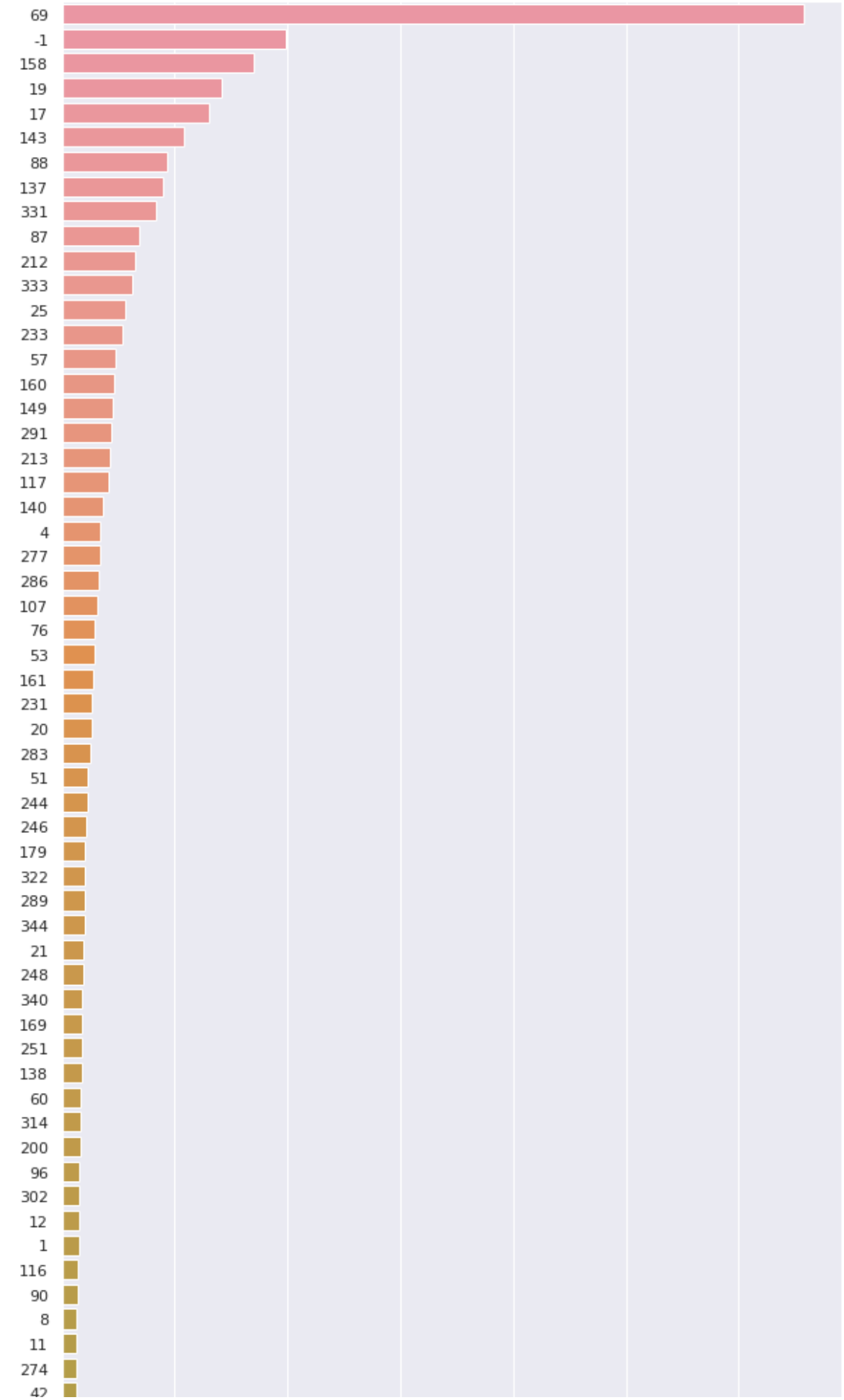
310
85
73
307
47
208
338
39
309
106
44
252
262
80
206
108
264
104
335
287
142
114
343
240
269
228
16
224
285
236
258
40
298
345
103
35
113
151
133
26
321
77
18
61
86
65
336
222
118
32
81
54
235
326
293
327
112

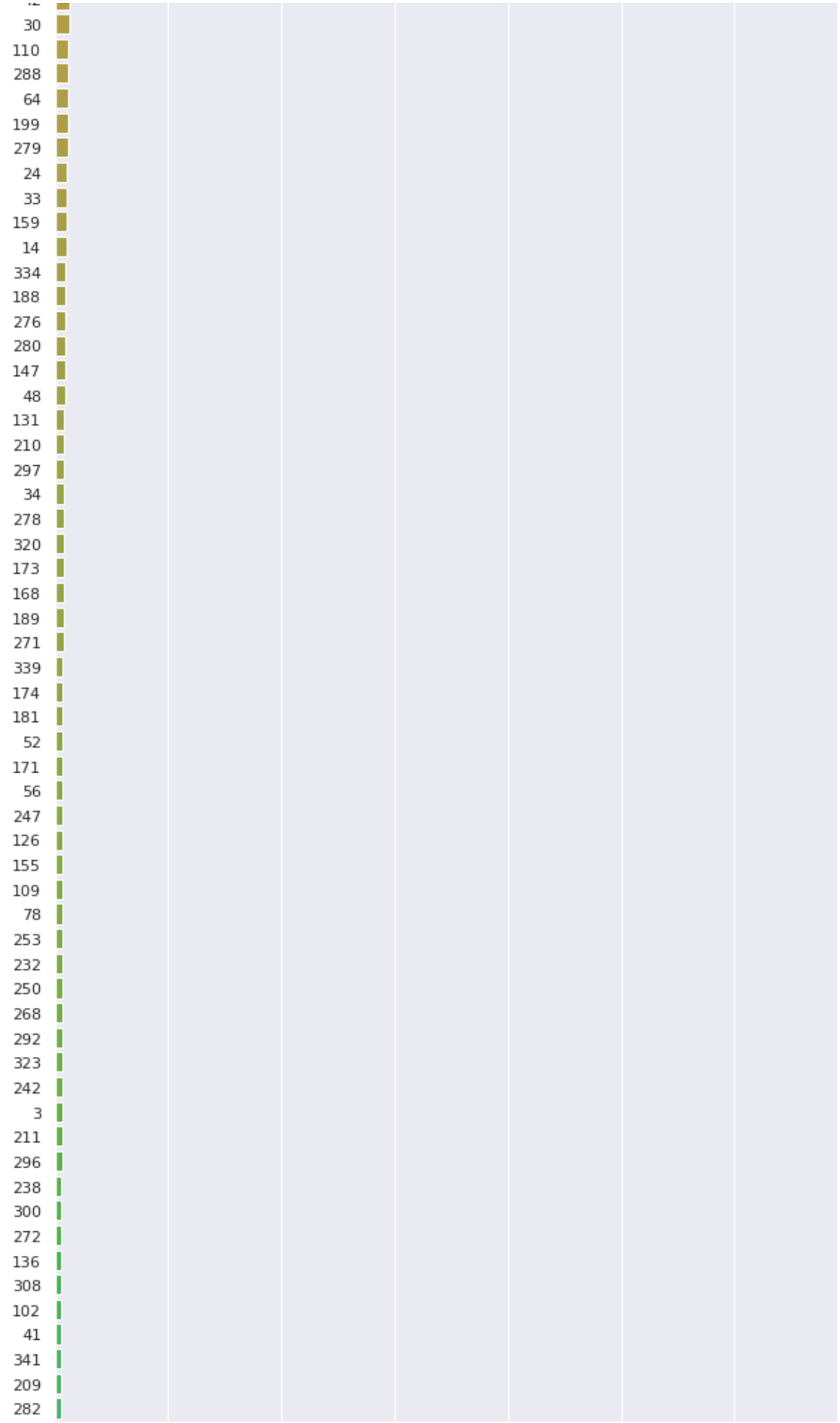


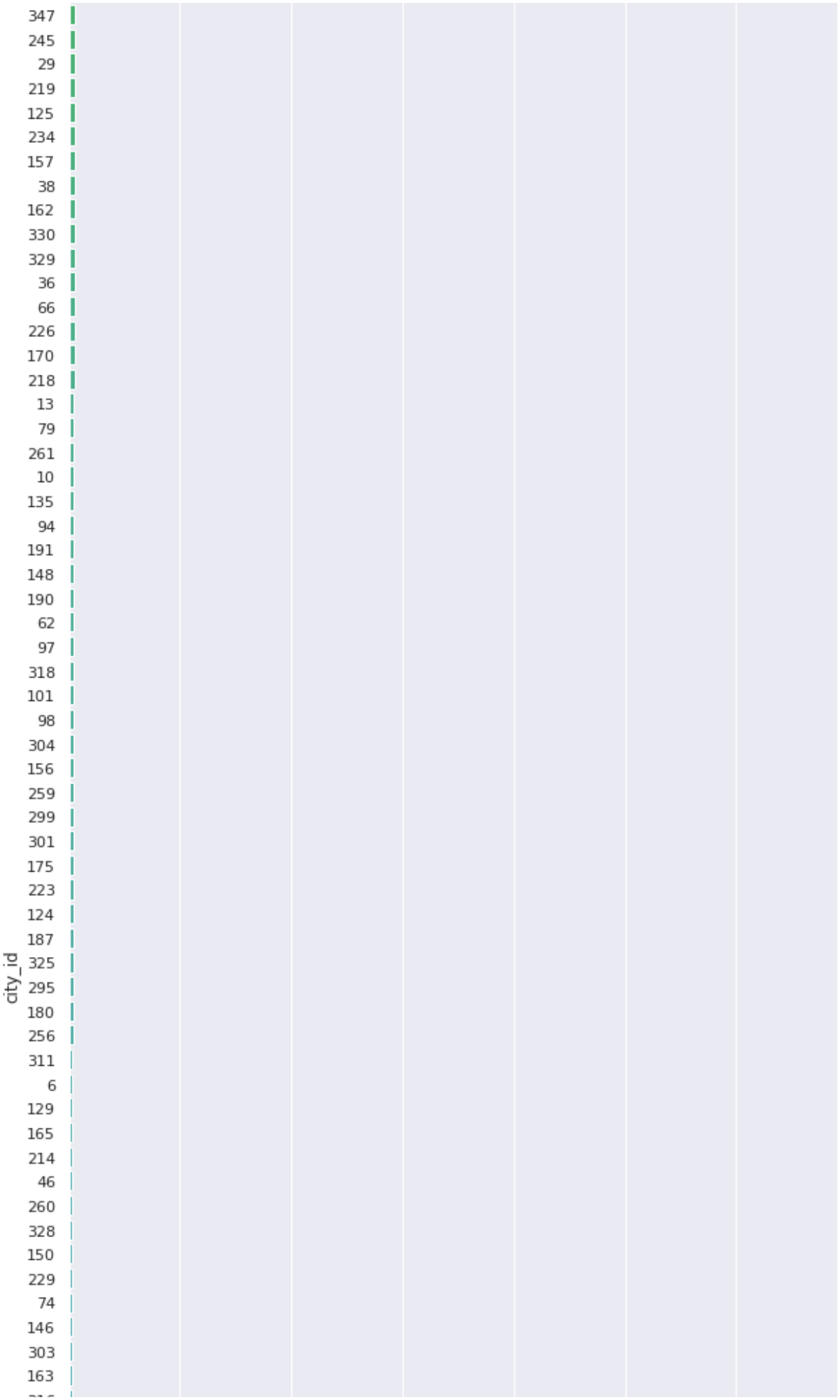
From the above counterplot , it gives us an intuitive idea which city_id recorded max number of transactions. We could also see there is a city_id which is -1. Till now we are not completely sure whether city_id can be negative values.

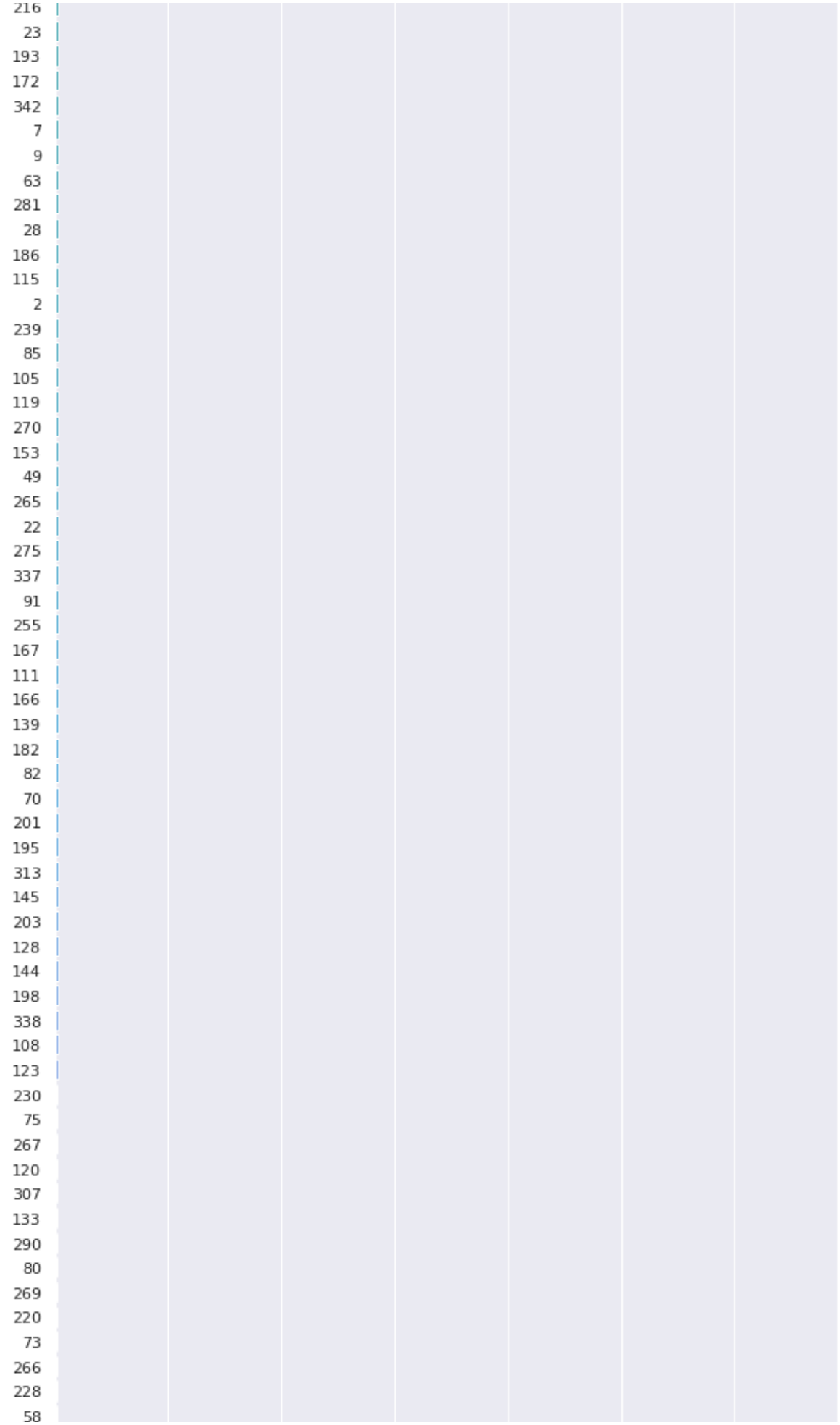
Now let's check for new_merchant_transactions

In [260]:
```
ax=sns.countplot(y="city_id",data=new_merchant_transactions,order=new_merchant
_transactions['city_id'].value_counts().index)
```

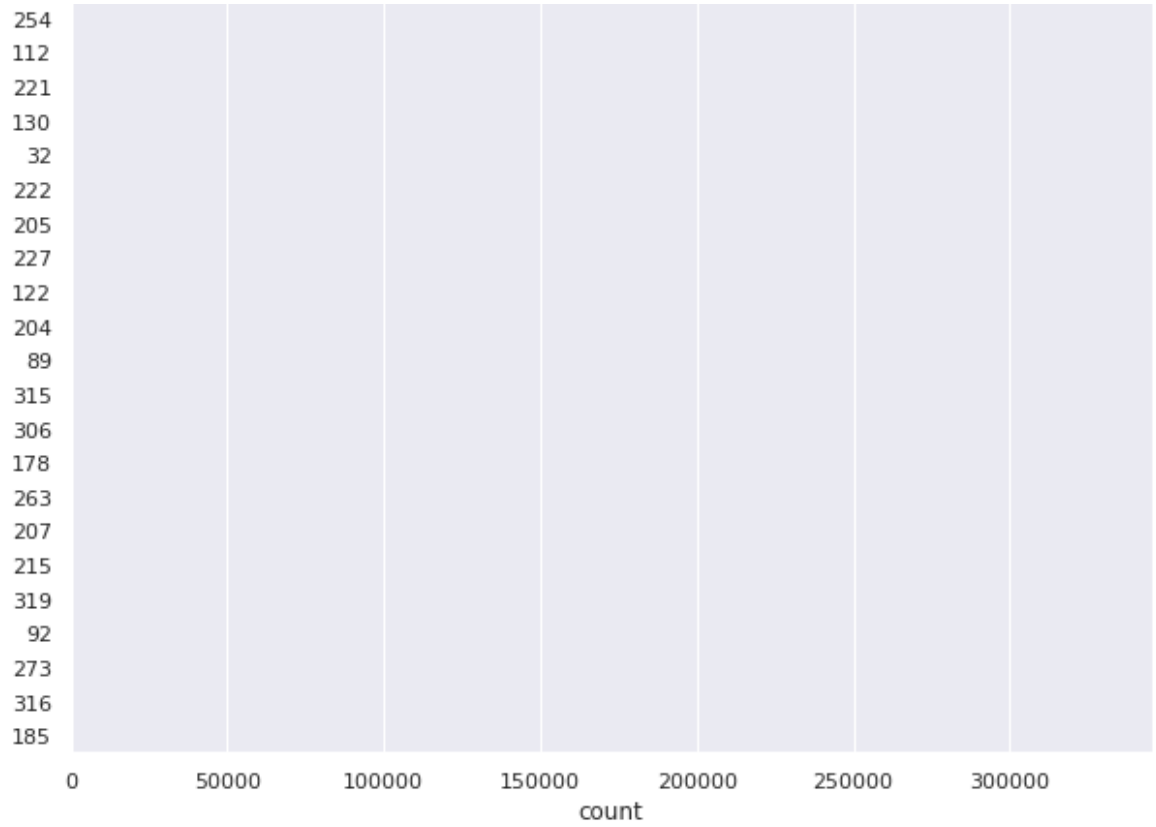

69
-1
158
19
17
143
88
137
331
87
212
333
25
233
57
160
149
291
213
117
140
4
277
286
107
76
53
161
231
20
283
51
244
246
179
322
289
344
21
248
340
169
251
138
60
314
200
96
302
12
1
116
90
8
11
274
42


30
110
288
64
199
279
24
33
159
14
334
188
276
280
147
48
131
210
297
34
278
320
173
168
189
271
339
174
181
52
171
56
247
126
155
109
78
253
232
250
268
292
323
242
3
211
296
238
300
272
136
308
102
41
341
209
282


city_id
347
245
29
219
125
234
157
38
162
330
329
36
66
226
170
218
13
79
261
10
135
94
191
148
190
62
97
318
101
98
304
156
259
299
301
175
223
124
187
325
295
180
256
311
6
129
165
214
46
260
328
150
229
74
146
303
163


216
23
193
172
342
7
9
63
281
28
186
115
2
239
85
105
119
270
153
49
265
22
275
337
91
255
167
111
166
139
182
82
70
201
195
313
145
203
128
144
198
338
108
123
230
75
267
120
307
133
290
80
269
220
73
266
228
58

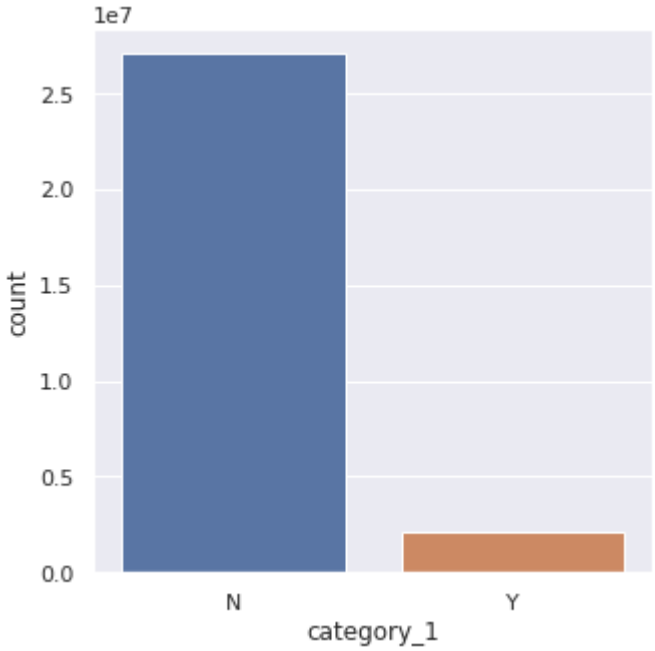
184
83
194
44
252
197
106
258
264
47
183
294
142
310
309
236
40
68
100
16
240
114
104
208
262
285
39
298
312
113
86
345
35
206
287
343
54
81
61
103
65
77
326
151
327
335
118
321
224
336
237
18
235
26
71
293
84



Seems the city_id -1, 69, 158 hold maximum number of transactions . Most of the cities recoreded very less transactions in comparison with them

Plot category_1

In [147]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="category_1",data=historical_transactions)
```

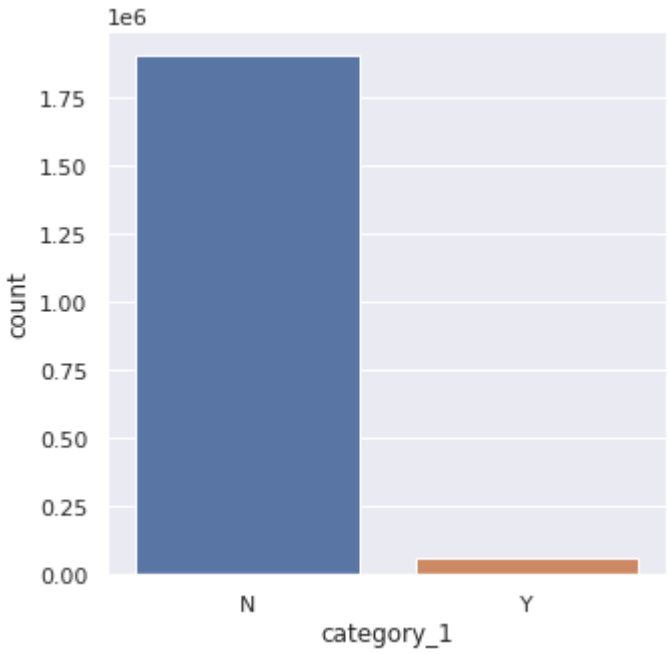


In [148]:
```
result=historical_transactions[historical_transactions['category_1'] == 'Y']
percentage_Y=len(result)/len(historical_transactions['category_1'])
print("Percentage of category_1 with Y:",percentage_Y*100)
```

```
Percentage of category_1 with Y: 7.158570890213954
```

So most of the data comes under 'N' in category_1. Only 7.15% comes under 'Y' category

In [149]:
```
ax=sns.countplot(x="category_1",data=new_merchant_transactions)
```



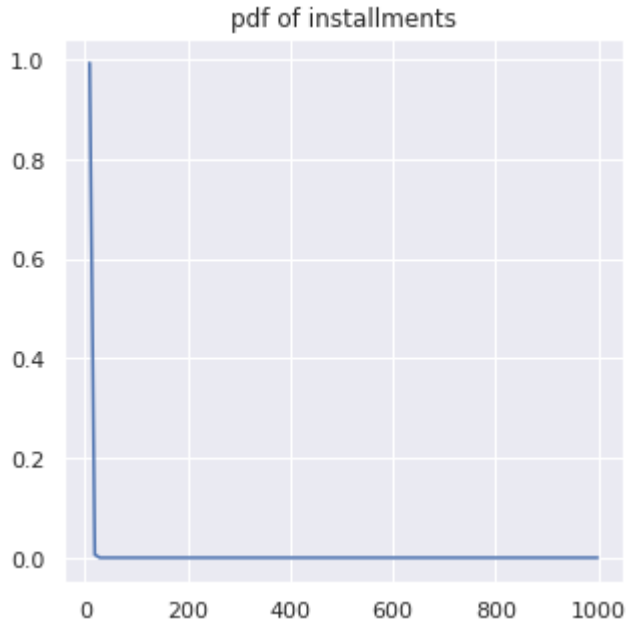
In [150]:
```
result=new_merchant_transactions[new_merchant_transactions['category_1'] ==
'Y']
percentage_Y=len(result)/len(new_merchant_transactions['category_1'])
print("Percentage of category_1 with Y:",percentage_Y*100)
```

```
Percentage of category_1 with Y: 3.2142131224621515
```

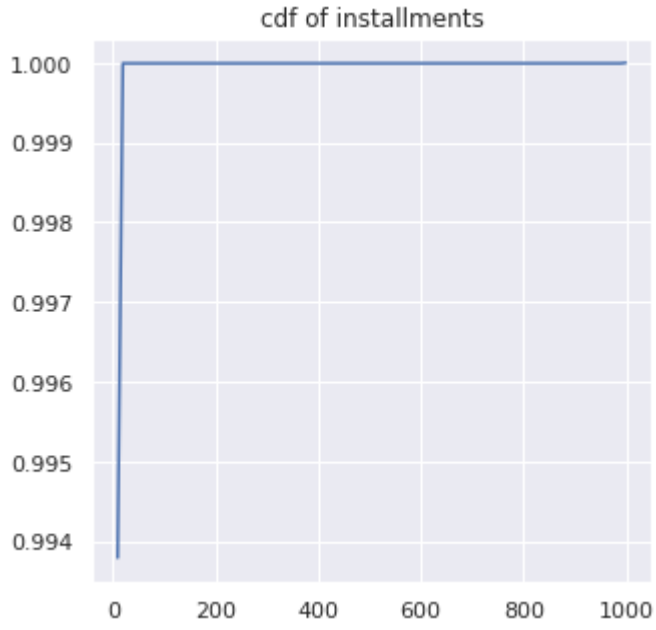
In new transactions also only 3.21% comes under 'Y' category

plot installments

In [151]:
```
sns.set(rc={'figure.figsize':(5,5)})
counts,bin_edges = np.histogram (historical_transactions['installments'],bins=
100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of installments")
plt.show()
```

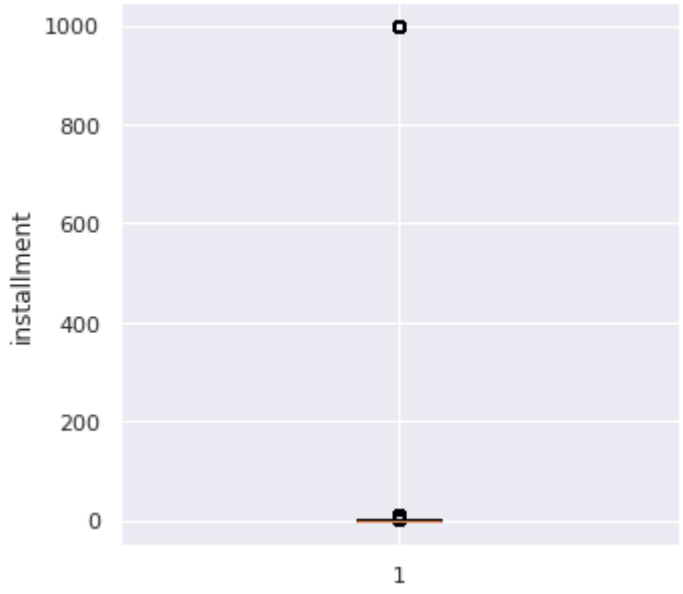


In [152]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of installments")
plt.show()
```



In [153]:
```
plt.boxplot(historical_transactions['installments'])
plt.ylabel("installment")
```

Out[153]: Text(0, 0.5, 'installment')



So from the above plots it is clear mostly are with 0 installments. Now we will check for percentile data

In [154]:
```
for i in [10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile of  :", np.percentile(historical_transactio
ns['installments'],[i]))
```

```
The 10 th percentile of  : [0.]
The 20 th percentile of  : [0.]
The 30 th percentile of  : [0.]
The 40 th percentile of  : [0.]
The 50 th percentile of  : [0.]
The 60 th percentile of  : [1.]
The 70 th percentile of  : [1.]
The 80 th percentile of  : [1.]
The 90 th percentile of  : [1.]
The 100 th percentile of  : [999.]
```

We will check more closely for 0th and 100th percentile

In [155]:
```
for i in range (0,11):
    print("The" ,i, "th percentile of  :", np.percentile(historical_transactio
ns['installments'],[i]))
```

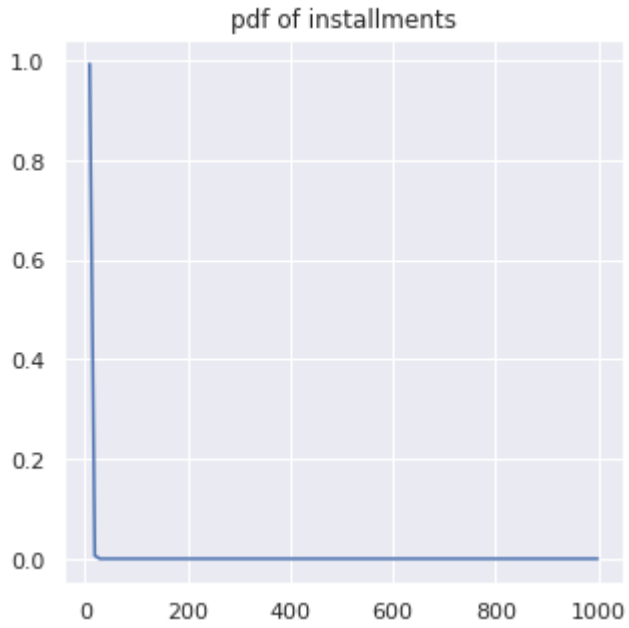
```
The 0 th percentile of  : [-1.]
The 1 th percentile of  : [0.]
The 2 th percentile of  : [0.]
The 3 th percentile of  : [0.]
The 4 th percentile of  : [0.]
The 5 th percentile of  : [0.]
The 6 th percentile of  : [0.]
The 7 th percentile of  : [0.]
The 8 th percentile of  : [0.]
The 9 th percentile of  : [0.]
The 10 th percentile of  : [0.]
```

In [156]:
```
for i in range (90,101):
    print("The" ,i, "th percentile :", np.percentile(historical_transactions[
'installments'],[i]))
```

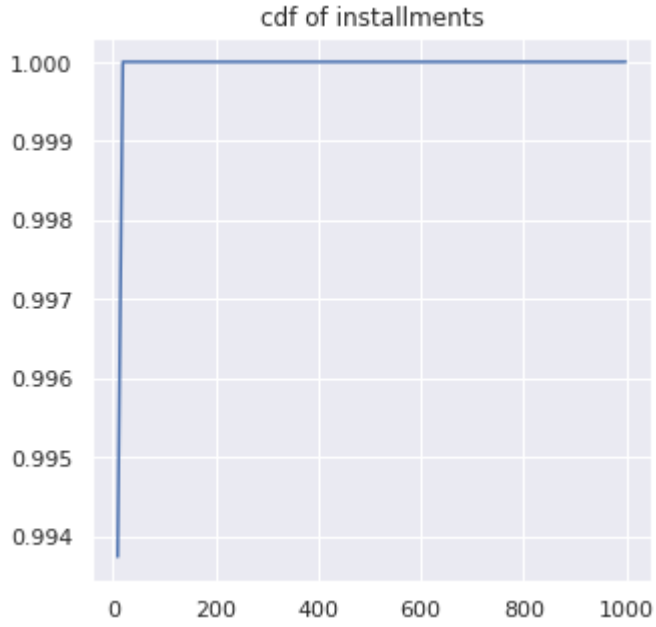
```
The 90 th percentile : [1.]
The 91 th percentile : [1.]
The 92 th percentile : [1.]
The 93 th percentile : [1.]
The 94 th percentile : [2.]
The 95 th percentile : [2.]
The 96 th percentile : [3.]
The 97 th percentile : [3.]
The 98 th percentile : [4.]
The 99 th percentile : [6.]
The 100 th percentile : [999.]
```

Now we have installments with -1 , this should be outlier as installments can't be negative. Alternatively the installments of 999 is massive. There is a high chance that it is also an outlier.

In [157]:
```
sns.set(rc={'figure.figsize':(5,5)})
counts,bin_edges = np.histogram (new_merchant_transactions['installments'],bin
s=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of installments")
plt.show()
```

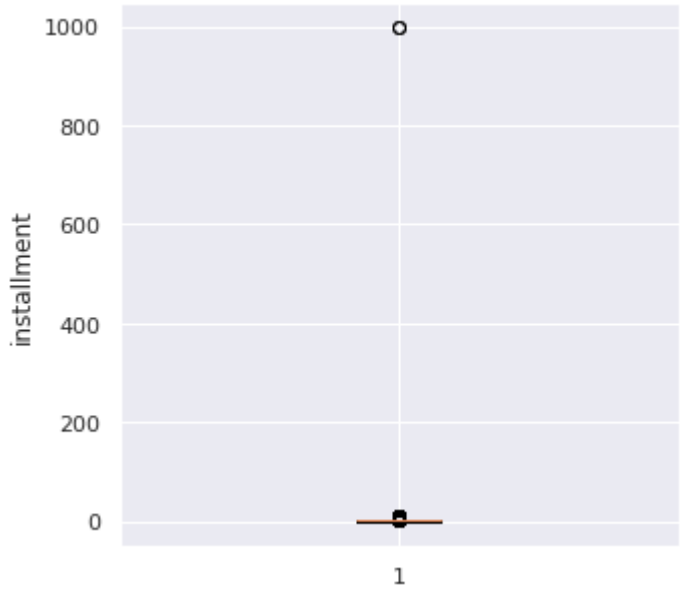


In [158]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of installments")
plt.show()
```



In [159]:
```
plt.boxplot(new_merchant_transactions['installments'])
plt.ylabel("installment")
```

Out[159]: Text(0, 0.5, 'installment')



In [160]:
```
for i in [10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile of  :", np.percentile(new_merchant_transact
ions['installments'],[i]))
```

```
The 10 th percentile of  : [0.]
The 20 th percentile of  : [0.]
The 30 th percentile of  : [0.]
The 40 th percentile of  : [0.]
The 50 th percentile of  : [1.]
The 60 th percentile of  : [1.]
The 70 th percentile of  : [1.]
The 80 th percentile of  : [1.]
The 90 th percentile of  : [1.]
The 100 th percentile of  : [999.]
```

In [161]:
```
for i in range (0,11):
    print("The" ,i, "th percentile of  :", np.percentile(new_merchant_transact
ions['installments'],[i]))
```

```
The 0 th percentile of  : [-1.]
The 1 th percentile of  : [-1.]
The 2 th percentile of  : [-1.]
The 3 th percentile of  : [0.]
The 4 th percentile of  : [0.]
The 5 th percentile of  : [0.]
The 6 th percentile of  : [0.]
The 7 th percentile of  : [0.]
The 8 th percentile of  : [0.]
The 9 th percentile of  : [0.]
The 10 th percentile of  : [0.]
```

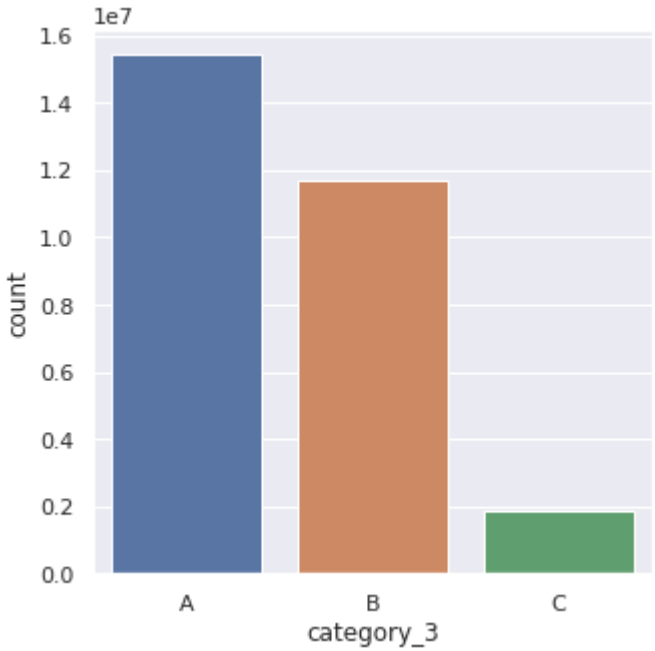
In [162]:
```
for i in range (90,101):
    print("The" ,i, "th percentile :", np.percentile(new_merchant_transactions
['installments'],[i]))
```

```
The 90 th percentile : [1.]
The 91 th percentile : [1.]
The 92 th percentile : [1.]
The 93 th percentile : [2.]
The 94 th percentile : [2.]
The 95 th percentile : [2.]
The 96 th percentile : [3.]
The 97 th percentile : [3.]
The 98 th percentile : [4.]
The 99 th percentile : [6.]
The 100 th percentile : [999.]
```

Same thing is applicable for New transactions as well. In feature engineering steps we need to concentrate on this unexpected behavior if we need to keep them as it is or need to be replaced

Plot category_3

In [163]:
```
ax=sns.countplot(x="category_3",data=historical_transactions)
```

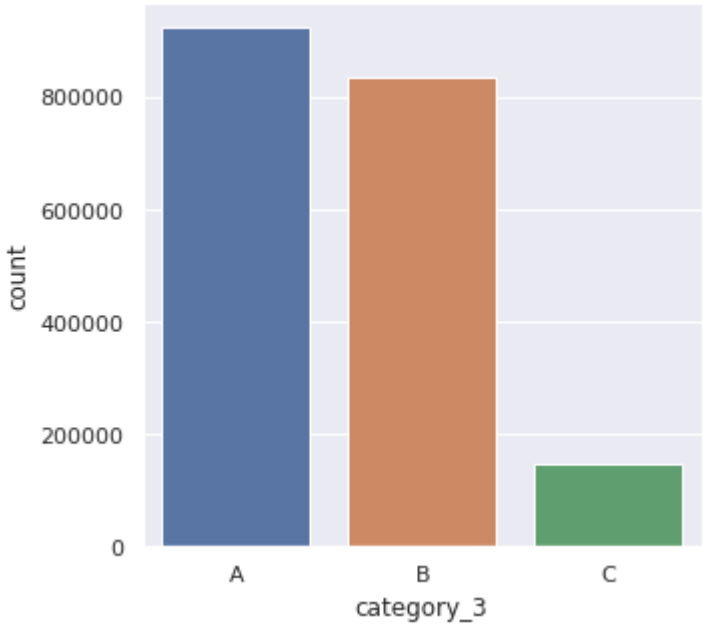


In [164]:
```
result=historical_transactions[historical_transactions['category_3'] == 'A']
percentage_Y=len(result)/len(historical_transactions['category_3'])
print("Percentage of category_3 with A:",percentage_Y*100)
result=historical_transactions[historical_transactions['category_3'] == 'B']
percentage_Y=len(result)/len(historical_transactions['category_3'])
print("Percentage of category_3 with B:",percentage_Y*100)
result=historical_transactions[historical_transactions['category_3'] == 'C']
percentage_Y=len(result)/len(historical_transactions['category_3'])
print("Percentage of category_3 with C:",percentage_Y*100)
```

```
Percentage of category_3 with A: 52.93884271358136
Percentage of category_3 with B: 40.11190298169221
Percentage of category_3 with C: 6.337284014855409
```

So mostly are from category 'A' followed by 'B' and 'C'. let's check for new transactions

In [262]:
```
sns.set(rc={'figure.figsize':(5,5)})
ax=sns.countplot(x="category_3",data=new_merchant_transactions,order=new_merch
ant_transactions['category_3'].value_counts().index)
```



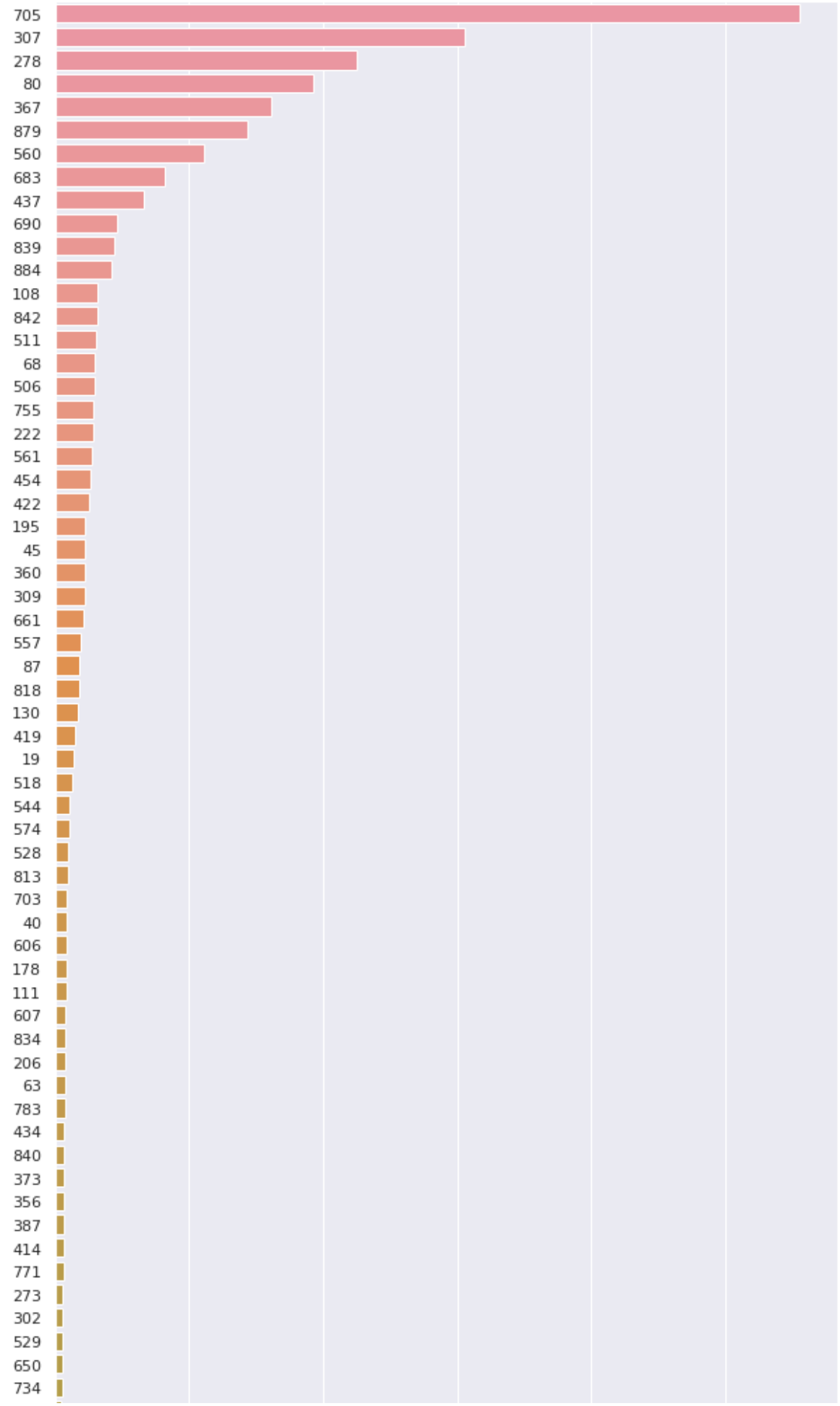
In [166]:
```
result=new_merchant_transactions[new_merchant_transactions['category_3'] ==
'A']
percentage_Y=len(result)/len(new_merchant_transactions['category_3'])
print("Percentage of category_3 with A:",percentage_Y*100)
result=new_merchant_transactions[new_merchant_transactions['category_3'] ==
'B']
percentage_Y=len(result)/len(new_merchant_transactions['category_3'])
print("Percentage of category_3 with B:",percentage_Y*100)
result=new_merchant_transactions[new_merchant_transactions['category_3'] ==
'C']
percentage_Y=len(result)/len(new_merchant_transactions['category_3'])
print("Percentage of category_3 with C:",percentage_Y*100)
```

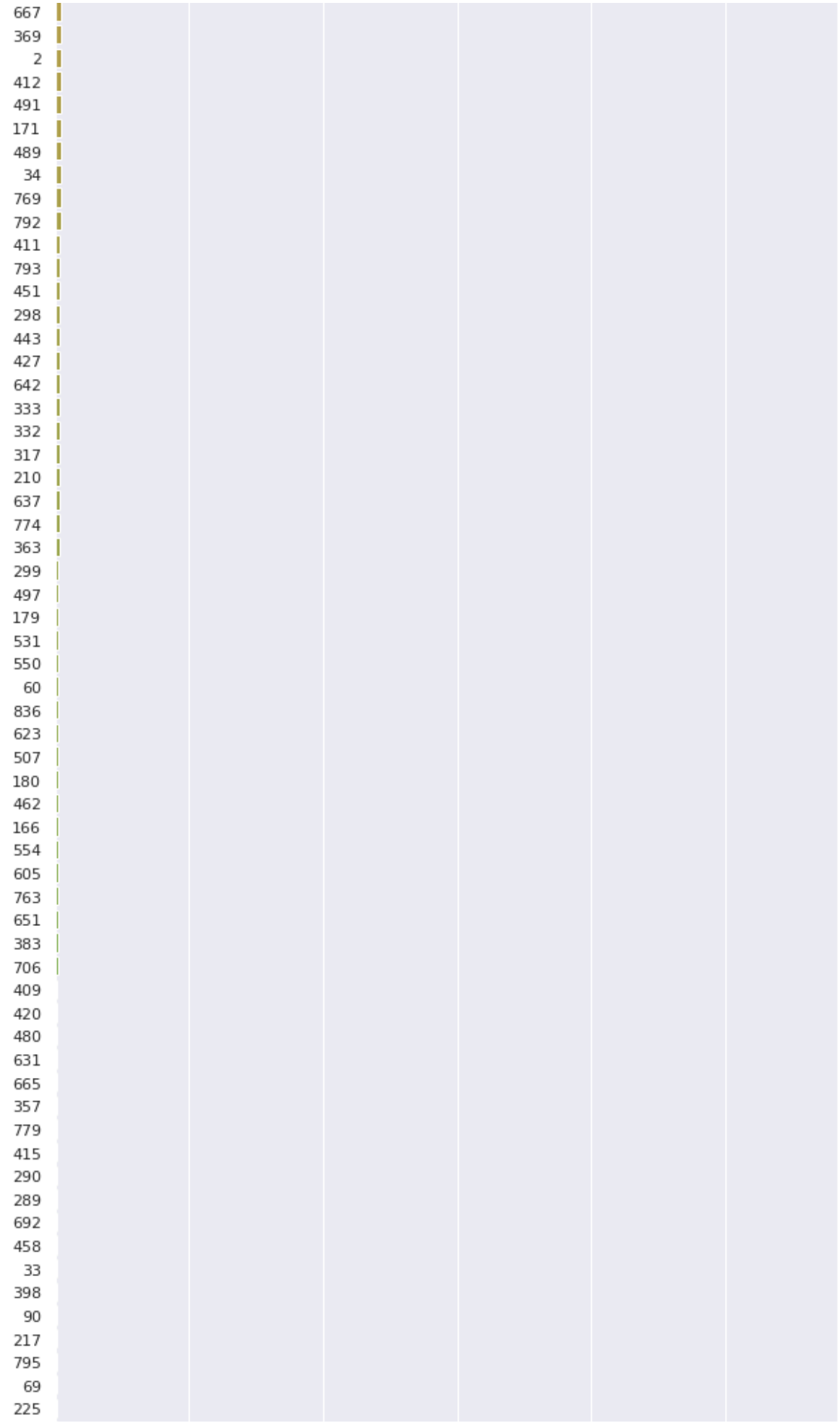
```
Percentage of category_3 with A: 46.98061314365387
Percentage of category_3 with B: 42.596270766992475
Percentage of category_3 with C: 7.574358224602667
```

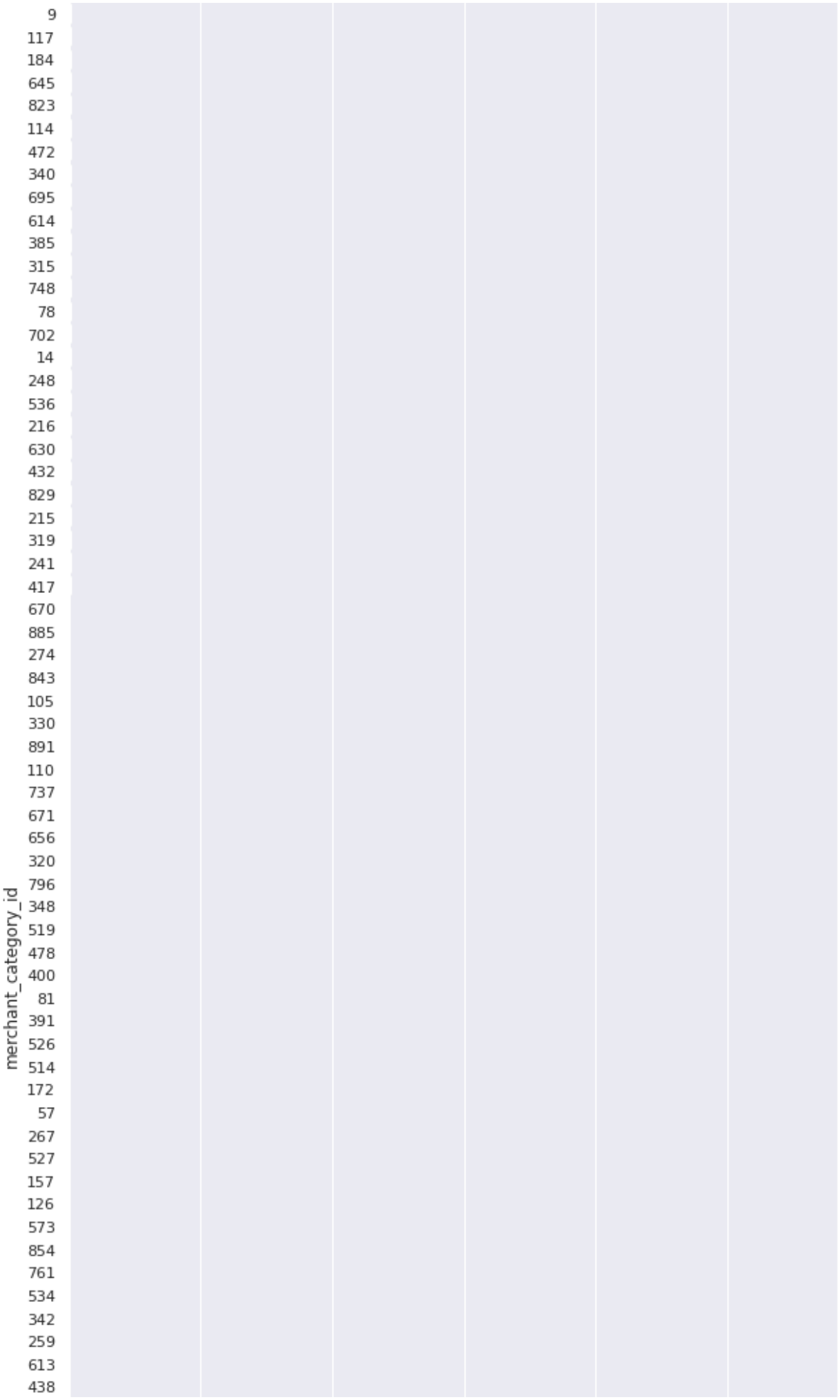
Verified the same in new transactions as well

Plot merchant_category_id

In [263]:
```
sns.set(rc={'figure.figsize':(10,100)})
sns.set_style(style="darkgrid")
ax=sns.countplot(y="merchant_category_id",data=historical_transactions,order=h
istorical_transactions['merchant_category_id'].value_counts().index)
```

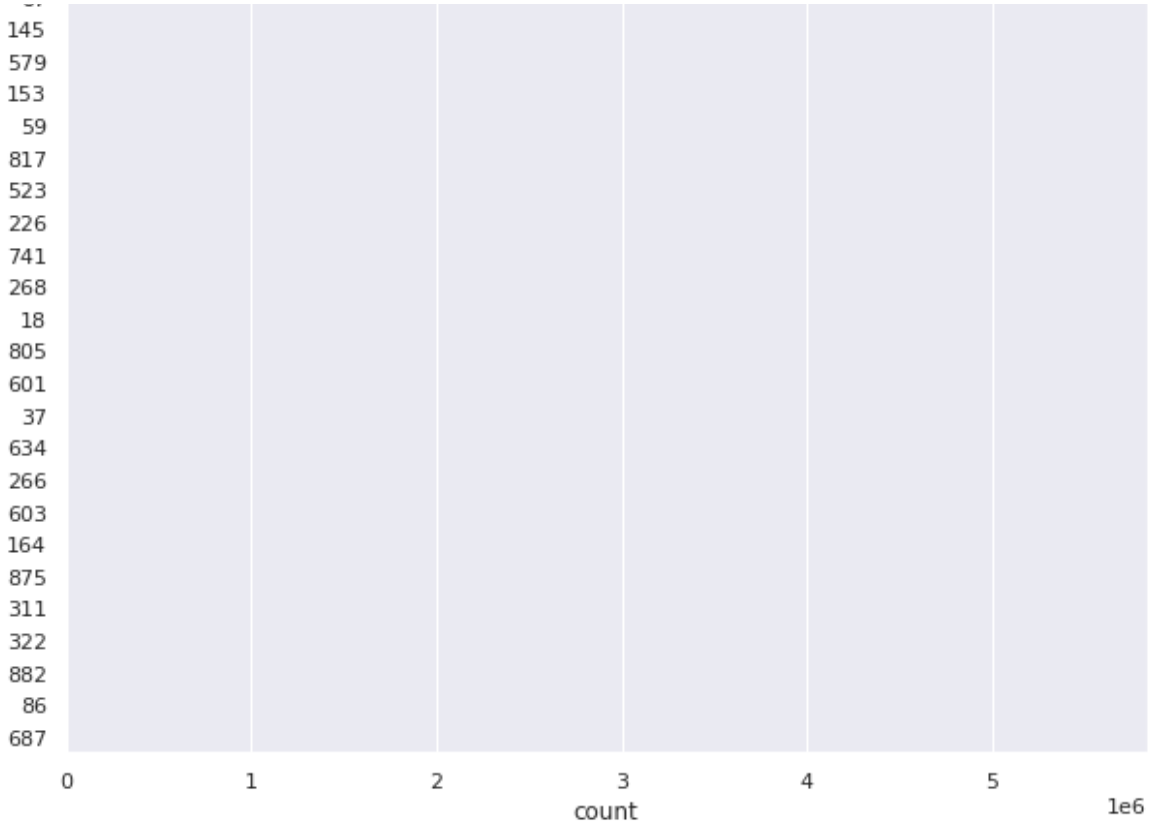

705
307
278
80
367
879
560
683
437
690
839
884
108
842
511
68
506
755
222
561
454
422
195
45
360
309
661
557
87
818
130
419
19
518
544
574
528
813
703
40
606
178
111
607
834
206
63
783
434
840
373
356
387
414
771
273
302
529
650
734


667
369
2
412
491
171
489
34
769
792
411
793
451
298
443
427
642
333
332
317
210
637
774
363
299
497
179
531
550
60
836
623
507
180
462
166
554
605
763
651
383
706
409
420
480
631
665
357
779
415
290
289
692
458
33
398
90
217
795
69
225


merchant_category_id
9
117
184
645
823
114
472
340
695
614
385
315
748
78
702
14
248
536
216
630
432
829
215
319
241
417
670
885
274
843
105
330
891
110
737
671
656
320
796
348
519
478
400
81
391
526
514
172
57
267
527
157
126
573
854
761
534
342
259
613
438

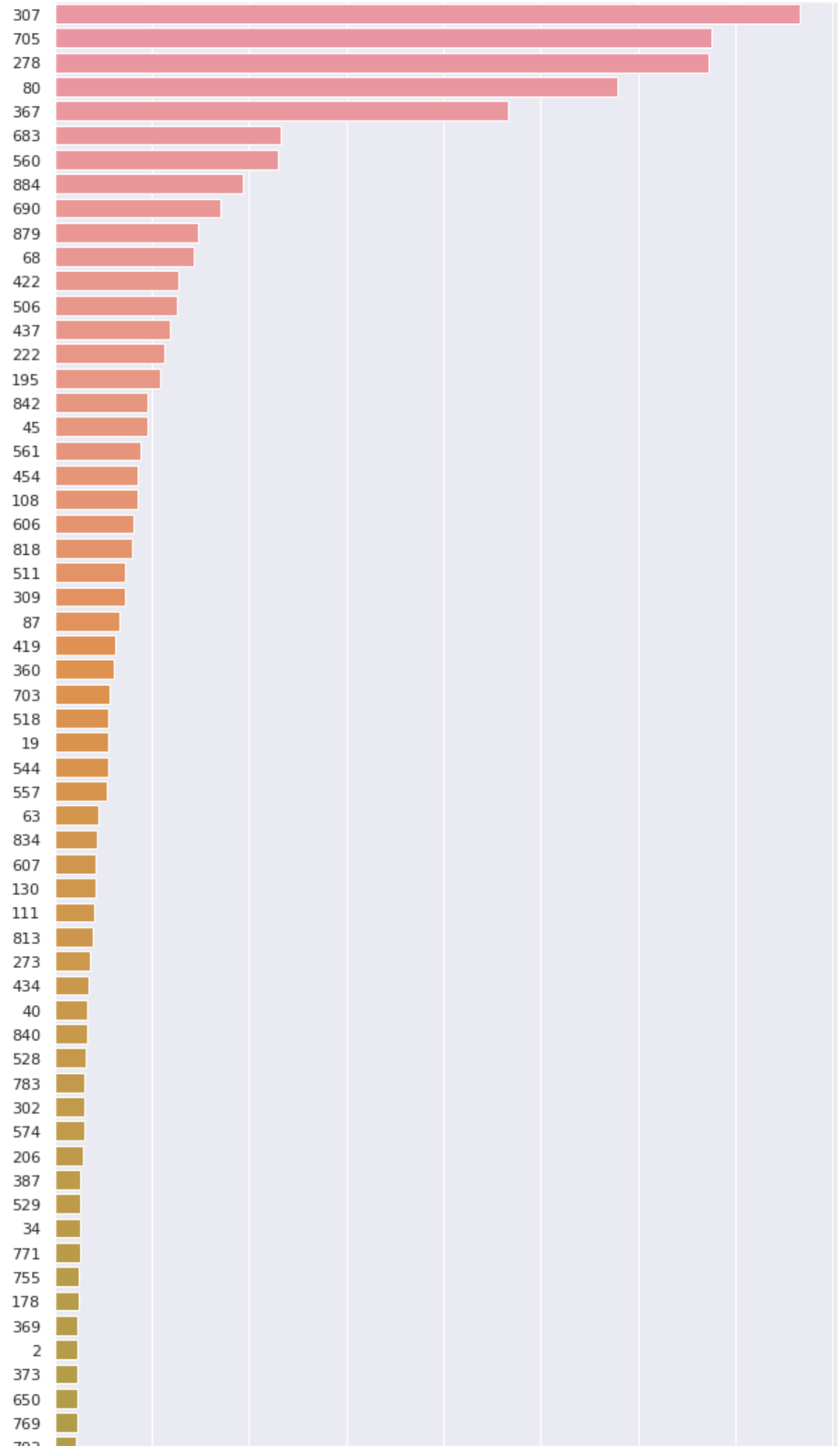
827
593
246
154
416
276
471
386
806
56
36
685
21
751
669
456
384
181
198
53
-1
396
224
498
781
209
873
71
260
312
499
115
265
504
652
38
509
474
556
351
713
343
819
469
374
16
223
653
430
401
591
889
358
101
119
676
551
182
52
482
245

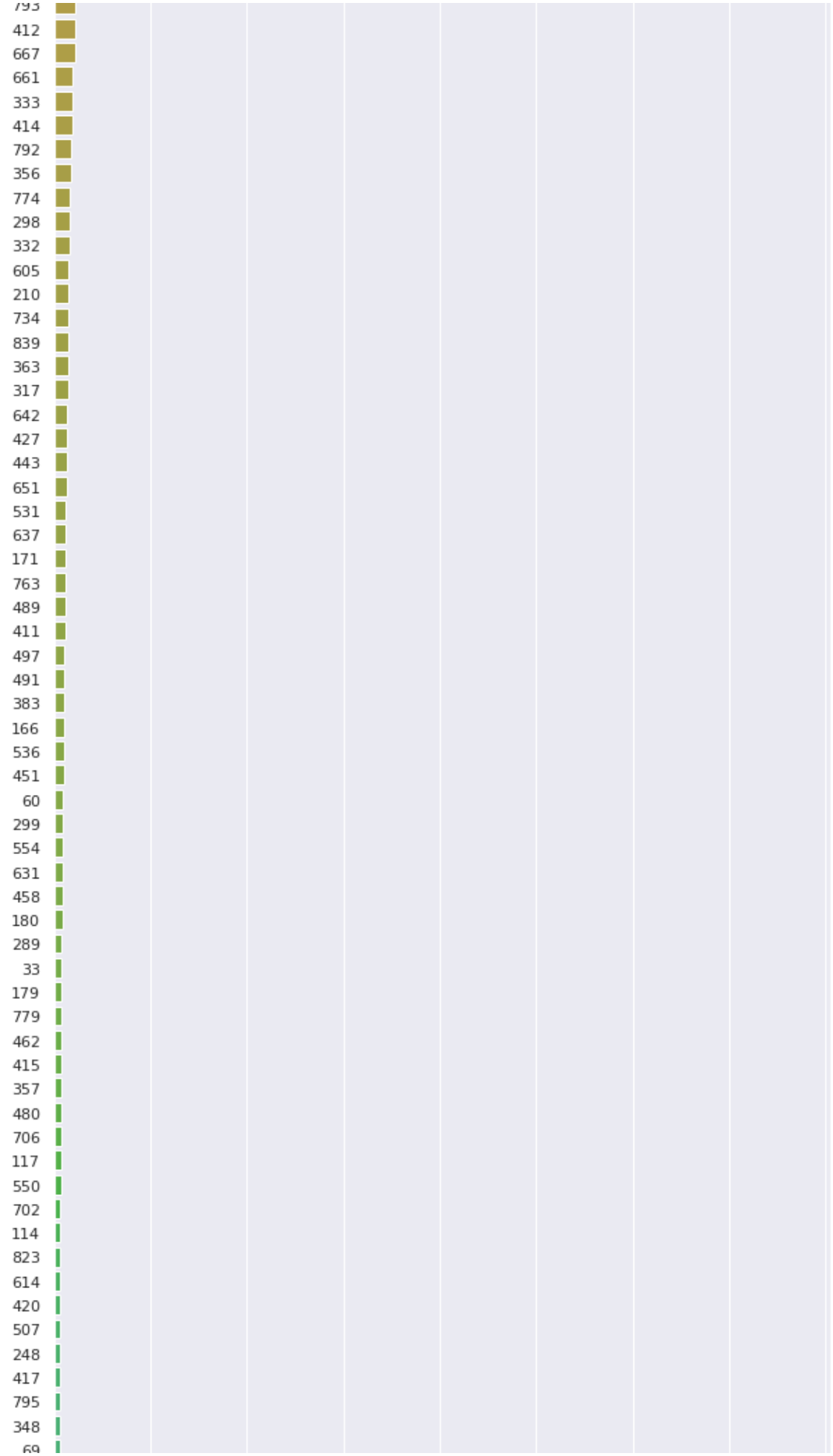
421
790
457
448
847
717
292
696
568
648
261
686
141
814
712
201
247
757
808
140
598
83
668
109
334
381
436
803
190
513
878
187
881
599
546
662
162
583
743
580
31
345
183
393
192
542
249
628
11
780
49
131
355
664
714
587
27
530
464
718
67



There is no NaN values for merchant_category_id. And from the counterplot we could see for the specific merchant_category like 278,307,705 are dominating

In [264]:
```
sns.set(rc={'figure.figsize':(10,100)})
sns.set_style(style="darkgrid")
ax=sns.countplot(y="merchant_category_id",data=new_merchant_transactions,order
=new_merchant_transactions['merchant_category_id'].value_counts().index)
```

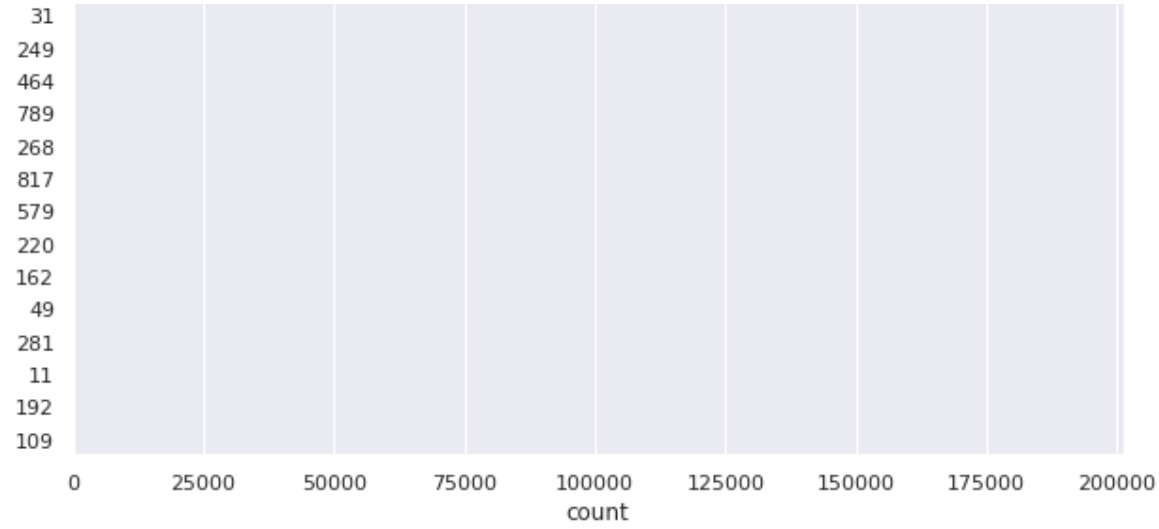

307
705
278
80
367
683
560
884
690
879
68
422
506
437
222
195
842
45
561
454
108
606
818
511
309
87
419
360
703
518
19
544
557
63
834
607
130
111
813
273
434
40
840
528
783
302
574
206
387
529
34
771
755
178
369
2
373
650
769


793
412
667
661
333
414
792
356
774
298
332
605
210
734
839
363
317
642
427
443
651
531
637
171
763
489
411
497
491
383
166
536
451
60
299
554
631
458
180
289
33
179
779
462
415
357
480
706
117
550
702
114
823
614
420
507
248
417
795
348
69


merchant_category_id
645
14
623
829
340
836
748
216
319
695
274
630
290
665
432
225
692
671
9
398
90
184
891
737
409
385
670
215
320
472
885
78
315
391
56
105
81
267
843
126
172
157
808
527
57
796
224
526
400
534
519
593
276
342
259
21
471
110
438
246

330
416
241
573
154
514
669
386
751
36
384
478
396
827
806
854
217
198
761
181
613
498
456
652
312
260
556
504
71
469
223
685
115
819
499
656
873
351
343
265
38
430
16
209
474
696
889
358
119
245
653
713
676
53
591
509
781
401
551
101

182
374
421
292
648
52
717
482
261
686
247
712
140
847
568
-1
381
457
757
345
790
83
201
881
334
814
598
587
513
546
448
599
436
187
878
583
743
393
668
664
190
530
803
67
183
131
542
141
725
27
718
628
662
145
714
153
355
580
523
18

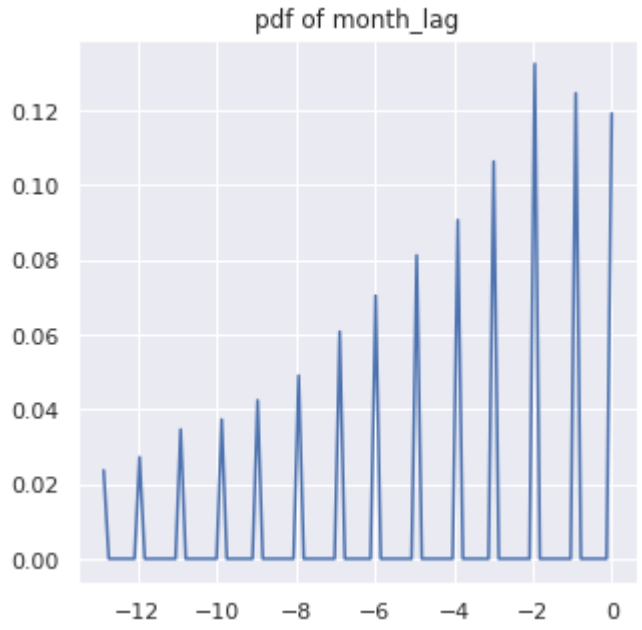


Analyze month_lag

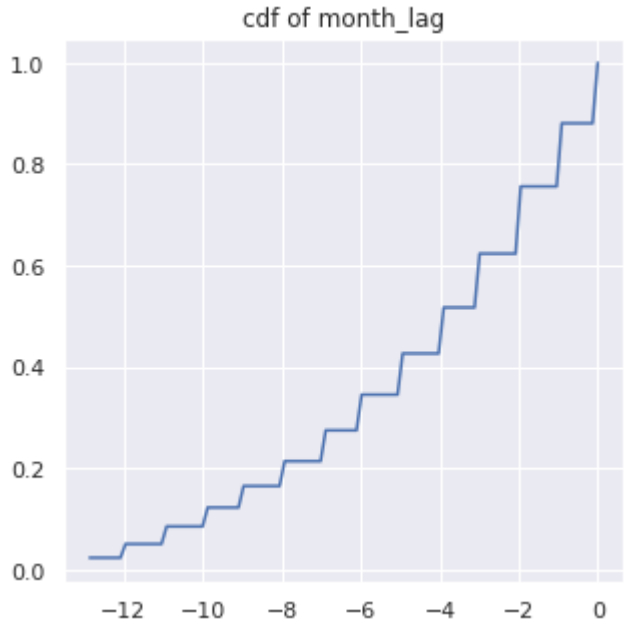
> Ref: https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/77687 (https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/77687)

" If this value is negative month_lag indicated that the transaction was realized before reference date and if this value is positive month_lag is indicated that the transaction was realized after reference date. most likely that reference date is the date where ELO started to recommended this merchant. "

In [169]:
```
sns.set(rc={'figure.figsize':(5,5)})
counts,bin_edges = np.histogram (historical_transactions['month_lag'],bins=100
, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of month_lag")
plt.show()
```

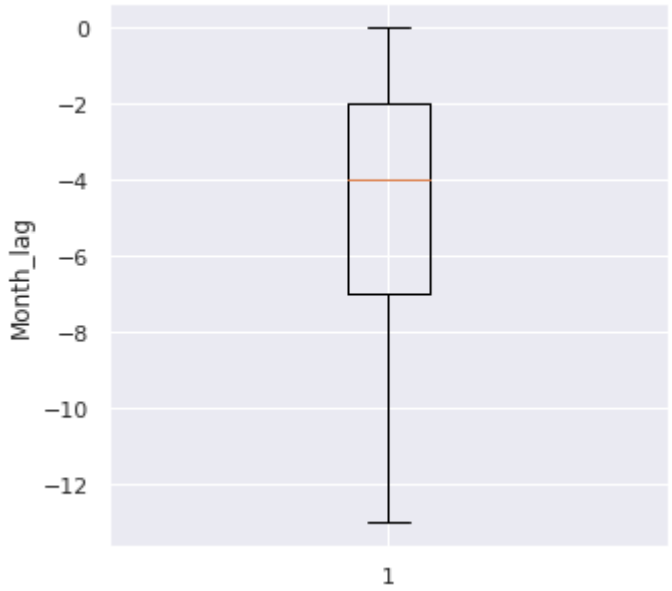


In [170]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of month_lag")
plt.show()
```



In [171]:
```
plt.boxplot(historical_transactions['month_lag'])
plt.ylabel("Month_lag")
```

Out[171]: Text(0, 0.5, 'Month_lag')



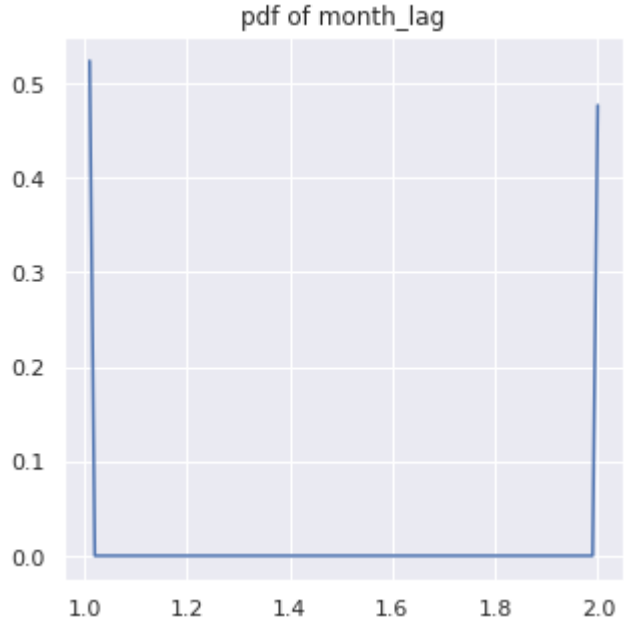
So in between (-10,-2) , maximum data exist for month_lag. As shown below even percentile data displaying the same.

In [172]:
```
for i in [1,10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile of  :", np.percentile(historical_transactio
ns['month_lag'],[i]))
```

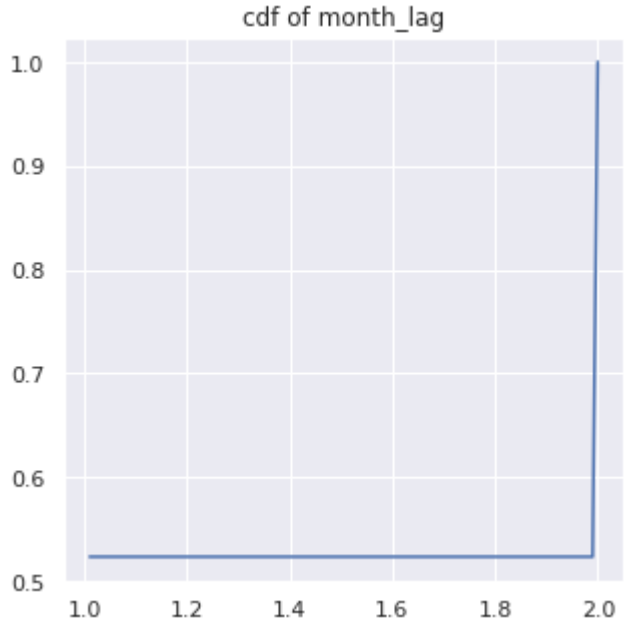
```
The 1 th percentile of  : [-13.]
The 10 th percentile of  : [-10.]
The 20 th percentile of  : [-8.]
The 30 th percentile of  : [-6.]
The 40 th percentile of  : [-5.]
The 50 th percentile of  : [-4.]
The 60 th percentile of  : [-3.]
The 70 th percentile of  : [-2.]
The 80 th percentile of  : [-1.]
The 90 th percentile of  : [0.]
The 100 th percentile of  : [0.]
```

let's check for new transaction

In [173]:
```
counts,bin_edges = np.histogram (new_merchant_transactions['month_lag'],bins=1
00, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of month_lag")
plt.show()
```

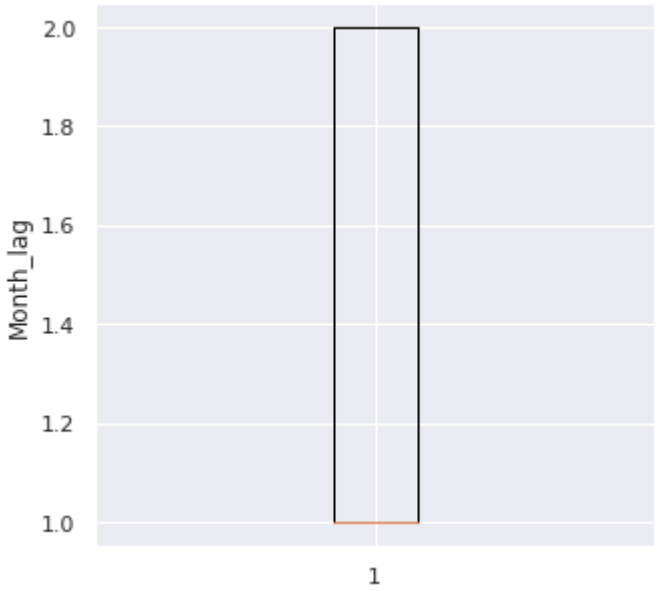


In [174]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of month_lag")
plt.show()
```



In [175]:
```
plt.boxplot(new_merchant_transactions['month_lag'])
plt.ylabel("Month_lag")
```

Out[175]: Text(0, 0.5, 'Month_lag')



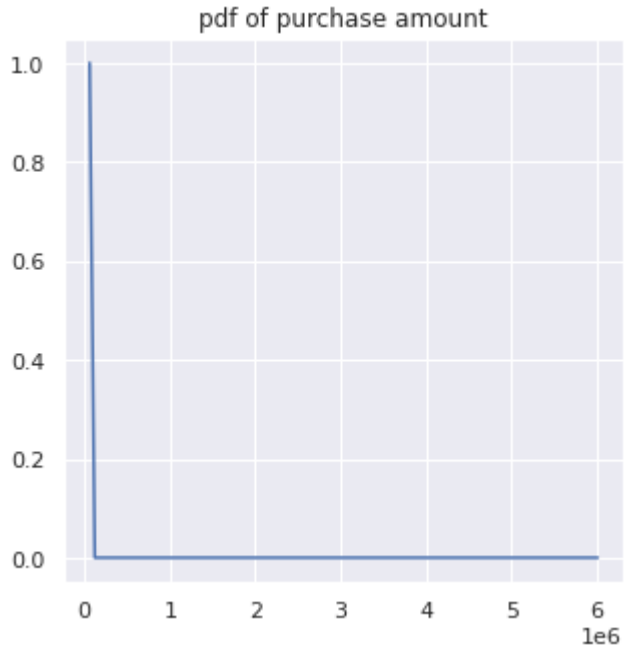
In [176]:
```
for i in [1,10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile of  :", np.percentile(historical_transactio
ns['month_lag'],[i]))
```

```
The 1 th percentile of  : [-13.]
The 10 th percentile of  : [-10.]
The 20 th percentile of  : [-8.]
The 30 th percentile of  : [-6.]
The 40 th percentile of  : [-5.]
The 50 th percentile of  : [-4.]
The 60 th percentile of  : [-3.]
The 70 th percentile of  : [-2.]
The 80 th percentile of  : [-1.]
The 90 th percentile of  : [0.]
The 100 th percentile of  : [0.]
```

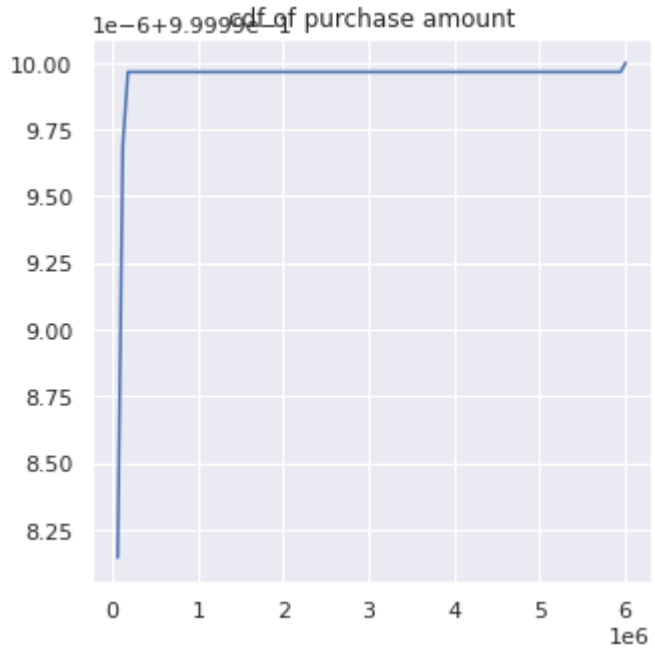
So distribution for month_lag is almost same in new and historical transactions

Analyze Purchase Amount

In [177]:
```
sns.set(rc={'figure.figsize':(5,5)})
counts,bin_edges = np.histogram (historical_transactions['purchase_amount'],bi
ns=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
#plt.xscale('log')
#plt.yscale('log')
plt.title("pdf of purchase amount")
plt.show()
```

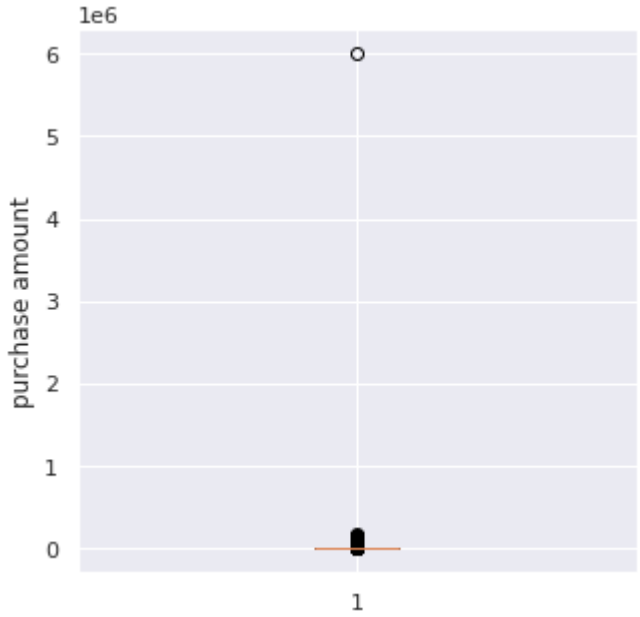


In [178]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of purchase amount")
plt.show()
```



In [179]:
```
plt.boxplot(historical_transactions['purchase_amount'])
plt.ylabel("purchase amount")
```

Out[179]: Text(0, 0.5, 'purchase amount')



In [180]:
```
for i in [1,10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile :", np.percentile(historical_transactions[
'purchase_amount'],[i]))
```

```
The 1 th percentile : [-0.74324133]
The 10 th percentile : [-0.73341399]
The 20 th percentile : [-0.72469861]
The 30 th percentile : [-0.71685477]
The 40 th percentile : [-0.70370657]
The 50 th percentile : [-0.68834948]
The 60 th percentile : [-0.67042285]
The 70 th percentile : [-0.6331571]
The 80 th percentile : [-0.57109761]
The 90 th percentile : [-0.42819548]
The 100 th percentile : [6010603.9717525]
```

In [181]:
```
for i in range (90,101):
    print("The" ,i, "th percentile :", np.percentile(historical_transactions[
'purchase_amount'],[i]))
```

```
The 90 th percentile : [-0.42819548]
The 91 th percentile : [-0.39627917]
The 92 th percentile : [-0.35892326]
The 93 th percentile : [-0.31053789]
The 94 th percentile : [-0.25196455]
The 95 th percentile : [-0.16861249]
The 96 th percentile : [-0.05286327]
The 97 th percentile : [0.13003943]
The 98 th percentile : [0.45521315]
The 99 th percentile : [1.22084097]
The 100 th percentile : [6010603.9717525]
```

It is surprising that purchase_amount is negative . However, According to Data_Dictionary.xlsx purchase_amount is normalized purchase amount. so, it can be negative.

Also the 100th percentile value is significantly larger in comparison with others

let's analyze the same for new transactions

In [182]:
```
for i in [1,10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile :", np.percentile(new_merchant_transactions
['purchase_amount'],[i]))
```

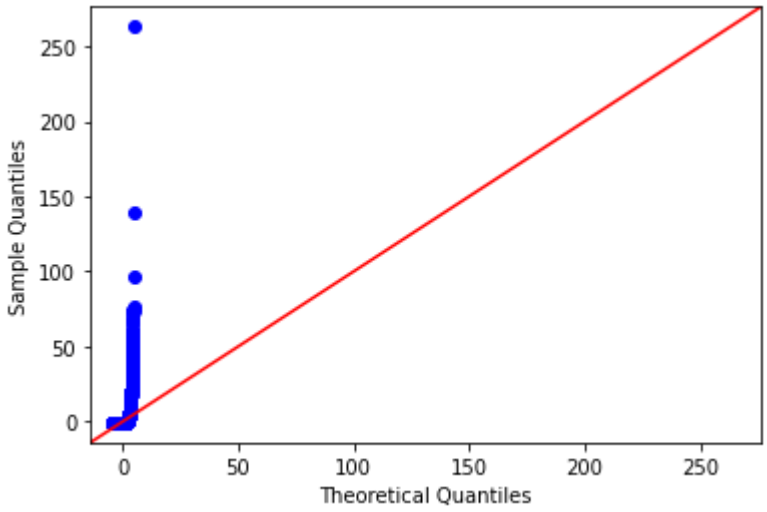
```
The 1 th percentile : [-0.74239984]
The 10 th percentile : [-0.73188128]
The 20 th percentile : [-0.72035595]
The 30 th percentile : [-0.70934152]
The 40 th percentile : [-0.69581766]
The 50 th percentile : [-0.67484064]
The 60 th percentile : [-0.65227082]
The 70 th percentile : [-0.60762706]
The 80 th percentile : [-0.5366869]
The 90 th percentile : [-0.37139527]
The 100 th percentile : [263.15749789]
```

In new transaction also the 100th percentile value is larger than others but not as much as historical transaction is

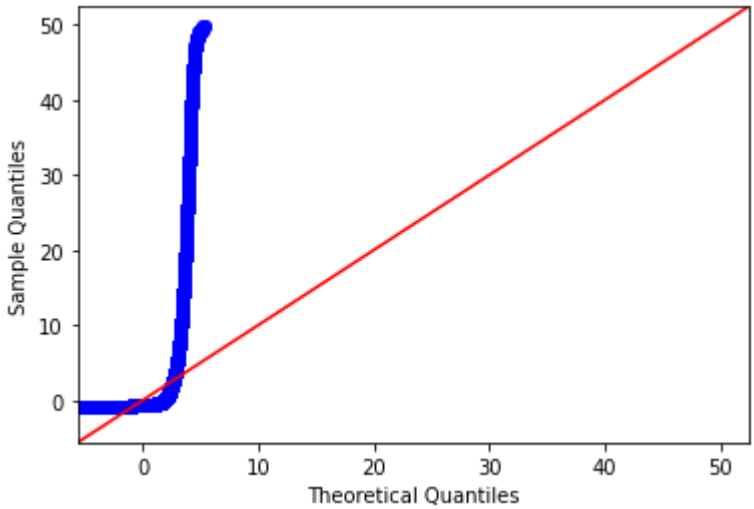
In [ ]:
```
a=new_merchant_transactions['purchase_amount']
a.to_numpy()
```

Out[ ]:
```
array([-0.55757375, -0.56957993, -0.55103721, ..., -0.62103071,
       -0.65674872, -0.73939454])
```

In [ ]:
```
import statsmodels.api as sm
import pylab as py
sm.qqplot(a, line='45')
py.show()
```



In [ ]:
```
qqplot(historical_transactions['purchase_amount'], line='45')
pyplot.show()
```

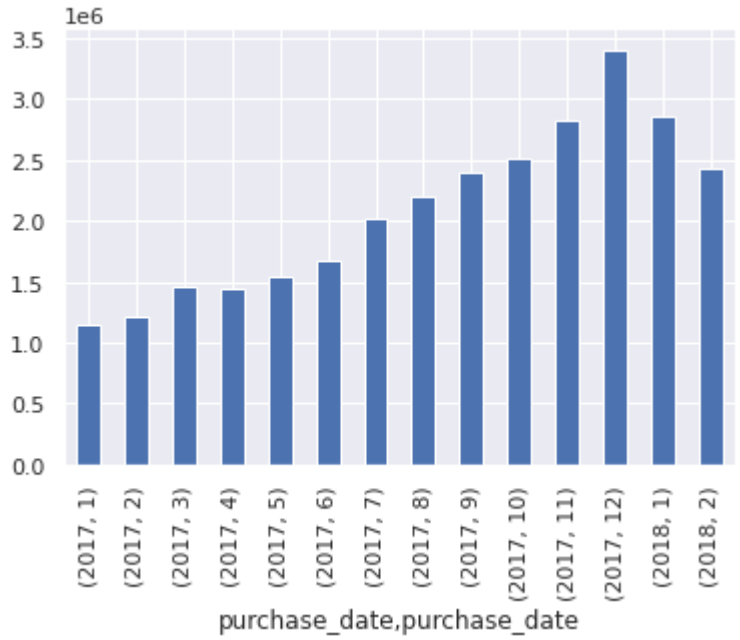


Plot purchase_date

In [183]:
```
# convert the datatype of purchase_date to datetime
historical_transactions['purchase_date']=pd.to_datetime(historical_transaction
s['purchase_date'],format='%Y-%m')
new_merchant_transactions['purchase_date']=pd.to_datetime(new_merchant_transac
tions['purchase_date'],format='%Y-%m')
```

In [184]:
```
#https://stackoverflow.com/questions/27365467/can-pandas-plot-a-histogram-of-d
ates
sns.set(rc={'figure.figsize':(10,10)})
%matplotlib inline
historical_transactions['purchase_date'].groupby([historical_transactions['pur
chase_date'].dt.year, historical_transactions['purchase_date'].dt.month]).coun
t().plot(kind="bar")
```

Out[184]: <matplotlib.axes._subplots.AxesSubplot at 0x7f565bd12810>

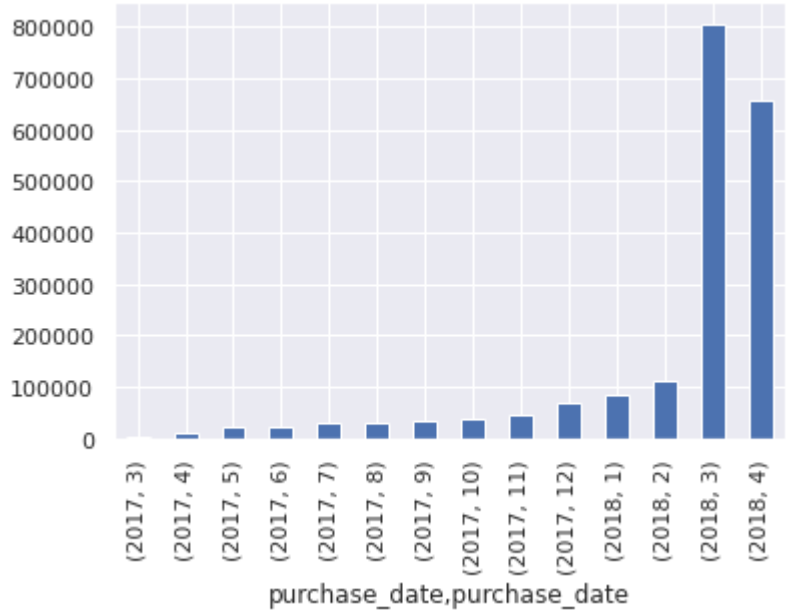


We have the historical transaction data from Jan'2017 to Feb'2018. Even though the transaction kept incresaing still there is a sudden drop after Dec'2017.

We experienced the same behavior in train/test dataset seems in late 2017 the transaction started decresing

In [185]:
```
new_merchant_transactions['purchase_date'].groupby([new_merchant_transactions[
'purchase_date'].dt.year, new_merchant_transactions['purchase_date'].dt.month
]).count().plot(kind="bar")
```

Out[185]: <matplotlib.axes._subplots.AxesSubplot at 0x7f565be3c2d0>

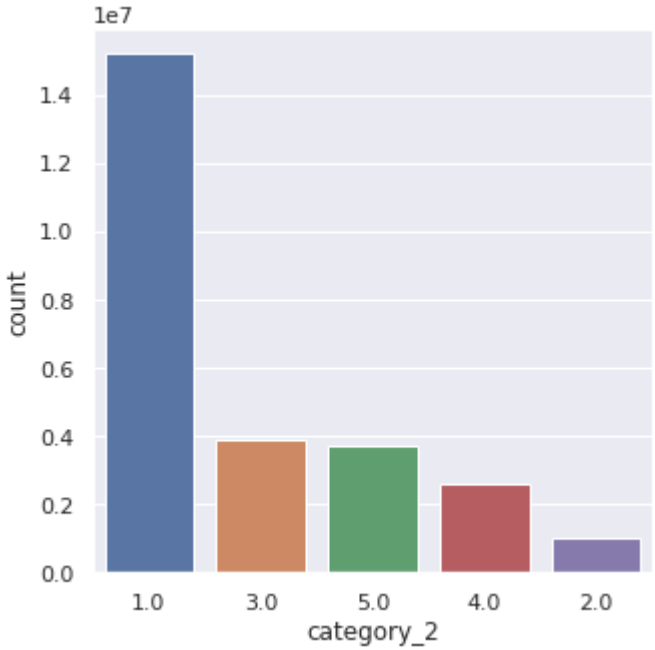


In new transaction details we have vey less amount of transaction till 2017 . As the transaction record are relatively new as per the dataset definition so this behavior is kind of expected

In [186]:
```
import gc
temp=gc.collect()
del temp
```

Plot Category_2

In [266]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="category_2",data=historical_transactions,order=historical_
transactions['category_2'].value_counts().index)
```

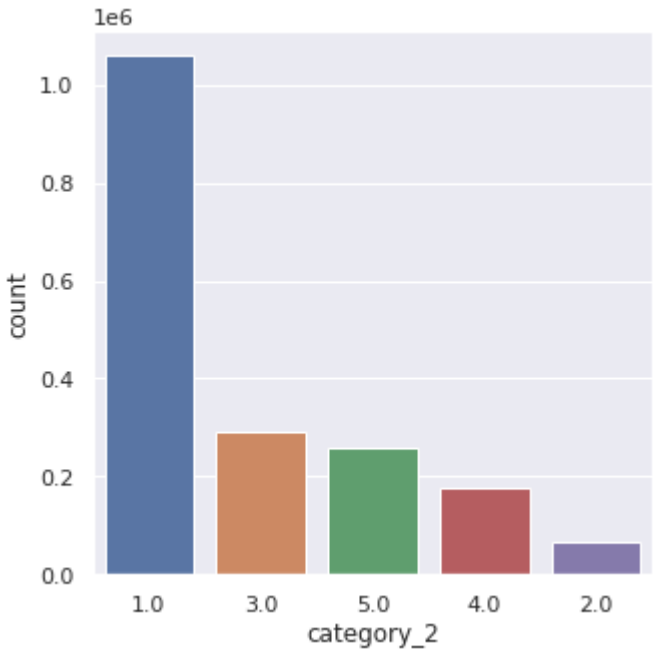


In [188]:
```
for i in [1.0,2.0,3.0,4.0,5.0]:
    result=historical_transactions[historical_transactions['category_2'] == i]
    percentage_Y=len(result)/len(historical_transactions['category_2'])
    print("Percentage of category_2 with",i,":",percentage_Y*100)
```

```
Percentage of category_2 with 1.0 : 52.133178068243936
Percentage of category_2 with 2.0 : 3.526113873072679
Percentage of category_2 with 3.0 : 13.436886826183558
Percentage of category_2 with 4.0 : 8.992925719765568
Percentage of category_2 with 5.0 : 12.798395155927064
```

under category_2 , the value 1.0 is dominating here

In [267]:
```
ax=sns.countplot(x="category_2",data=new_merchant_transactions,order=new_merch
ant_transactions['category_2'].value_counts().index)
```



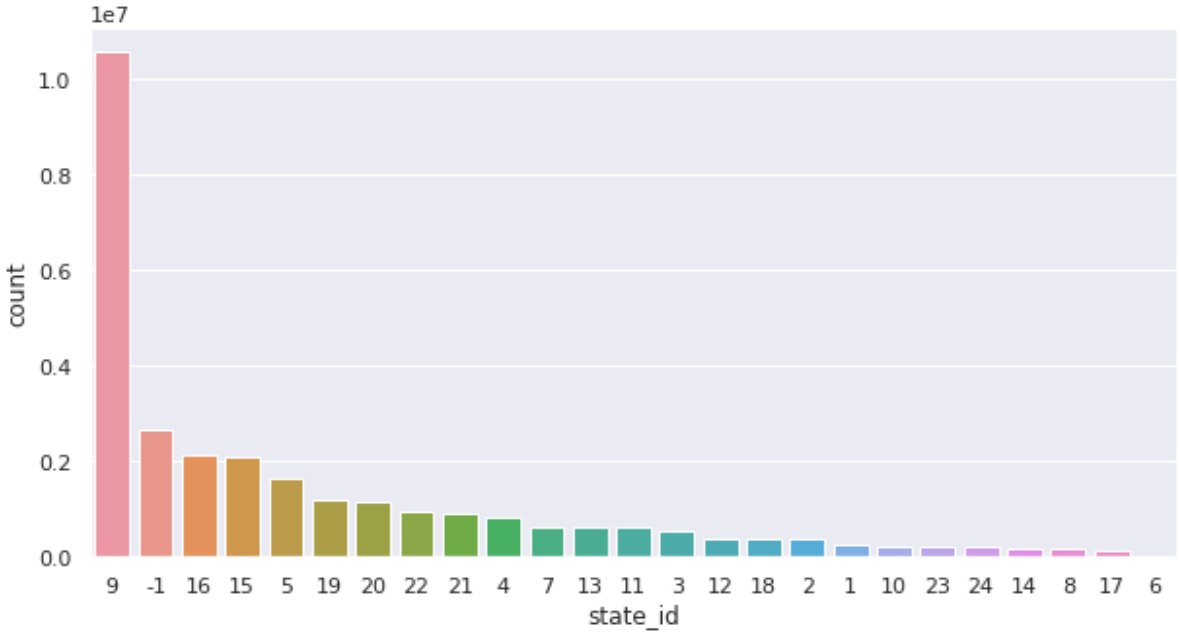
In [190]:
```
for i in [1.0,2.0,3.0,4.0,5.0]:
    result=new_merchant_transactions[new_merchant_transactions['category_2'] =
= i]
    percentage_Y=len(result)/len(new_merchant_transactions['category_2'])
    print("Percentage of category_2 with",i,":",percentage_Y*100)
```

```
Percentage of category_2 with 1.0 : 53.9085730179503
Percentage of category_2 with 2.0 : 3.3449802881360506
Percentage of category_2 with 3.0 : 14.748875590859237
Percentage of category_2 with 4.0 : 9.097665803545638
Percentage of category_2 with 5.0 : 13.207432791433249
```

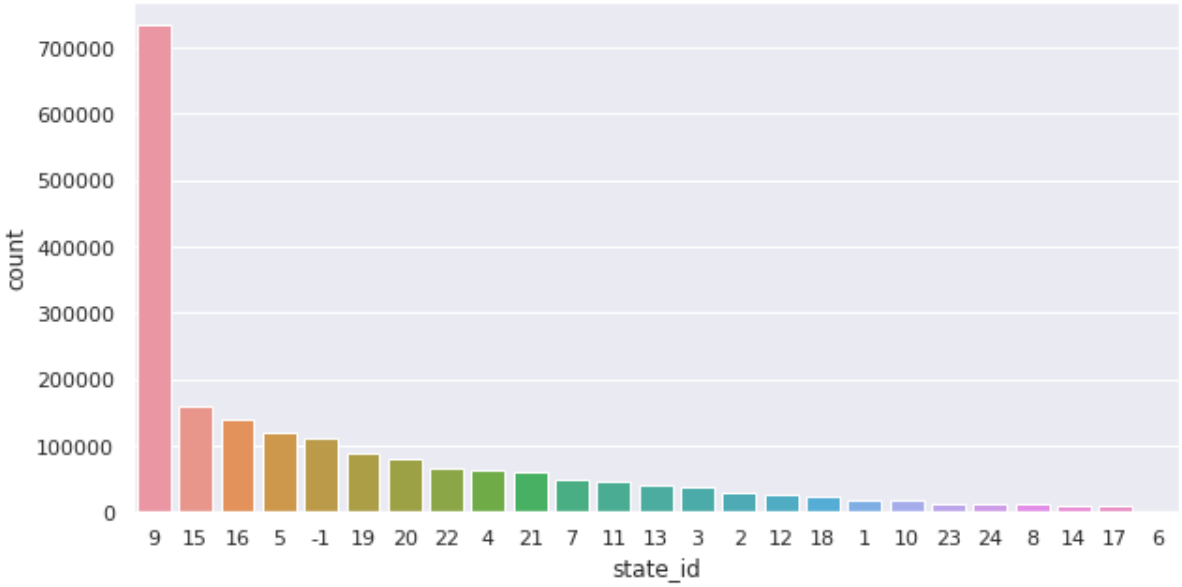
The distribution is almost same in historical and new_merchant transactions

Plot State-ID

In [268]:
```
sns.set(rc={'figure.figsize':(10,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="state_id",data=historical_transactions,order=historical_tr
ansactions['state_id'].value_counts().index)
```



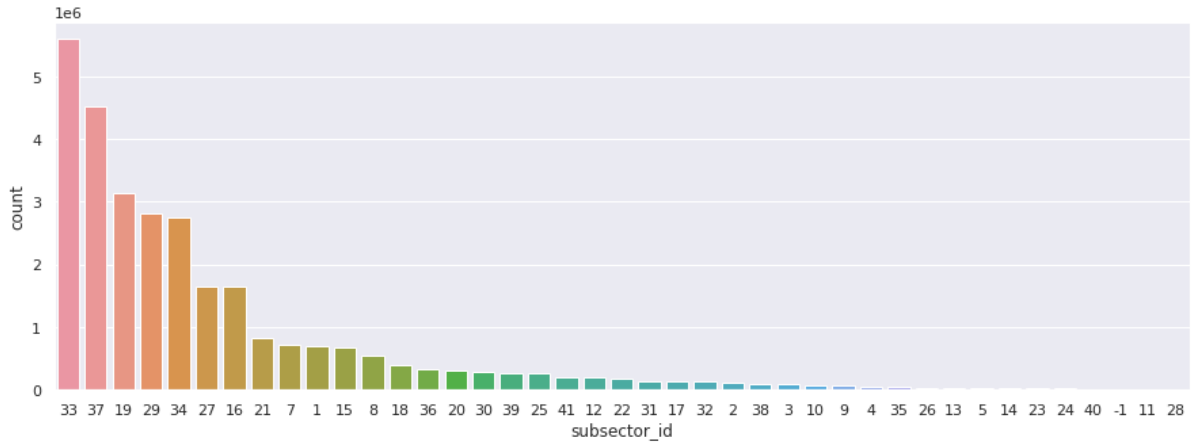
In [269]:
```
ax=sns.countplot(x="state_id",data=new_merchant_transactions,order=new_merchan
t_transactions['state_id'].value_counts().index)
```



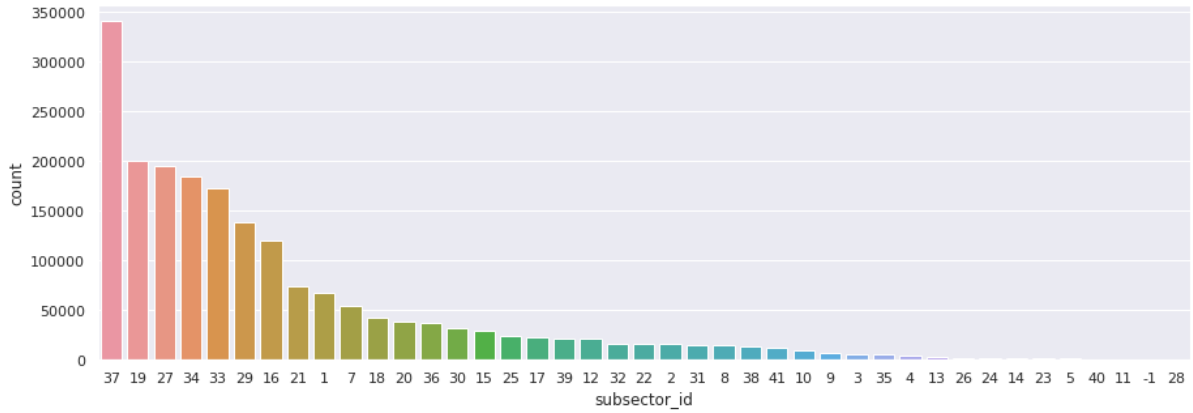
In both cases - the state_id with -1,9,15,16,19,20 are clearly dominating

Plot Subsector_ID

In [270]:
```
sns.set(rc={'figure.figsize':(15,5)})
sns.set_style(style="darkgrid")
ax=sns.countplot(x="subsector_id",data=historical_transactions,order=historica
l_transactions['subsector_id'].value_counts().index)
```



In [272]:
```
ax=sns.countplot(x="subsector_id",data=new_merchant_transactions,order=new_mer
chant_transactions['subsector_id'].value_counts().index)
```



From the plot it is clear the subsector_id with 19,33,34,37 recorded maximum transactions.

In [195]:
```
merchants=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/merc
hants.csv')
merchants.columns
```

Out[195]:
```
Index(['merchant_id', 'merchant_group_id', 'merchant_category_id',
       'subsector_id', 'numerical_1', 'numerical_2', 'category_1',
       'most_recent_sales_range', 'most_recent_purchases_range',
       'avg_sales_lag3', 'avg_purchases_lag3', 'active_months_lag3',
       'avg_sales_lag6', 'avg_purchases_lag6', 'active_months_lag6',
       'avg_sales_lag12', 'avg_purchases_lag12', 'active_months_lag12',
       'category_4', 'city_id', 'state_id', 'category_2'],
      dtype='object')
```

In [196]:
```
print("shape of merchant data:",merchants.shape)
```

```
shape of merchant data: (334696, 22)
```

In [197]:
```
merchants.head(5)
```

Out[197]:

	merchant_id	merchant_group_id	merchant_category_id	subsector_id	numerical_1	nume
0	M_ID_838061e48c	8353	792	9	-0.057471	-0
1	M_ID_9339d880ad	3184	840	20	-0.057471	-0
2	M_ID_e726bbae1e	447	690	1	-0.057471	-0
3	M_ID_a70e9c5f81	5026	792	9	-0.057471	-0
4	M_ID_64456c37ce	2228	222	21	-0.057471	-0

5 rows × 22 columns

In [198]:
```
merchants.describe()
```

Out[198]:

	merchant_group_id	merchant_category_id	subsector_id	numerical_1	numerical_2
count	334696.000000	334696.000000	334696.000000	334696.000000	334696.000000
mean	31028.736143	423.131663	25.116404	0.011476	0.008103
std	31623.043426	252.898046	9.807371	1.098154	1.070497
min	1.000000	-1.000000	-1.000000	-0.057471	-0.057471
25%	3612.000000	222.000000	19.000000	-0.057471	-0.057471
50%	19900.000000	373.000000	27.000000	-0.057471	-0.057471
75%	51707.250000	683.000000	33.000000	-0.047556	-0.047556
max	112586.000000	891.000000	41.000000	183.735111	182.079322

for the purchase features we have few inf values

In [199]:
```
duplicate_merchant_id = merchants[merchants.duplicated(['merchant_id'],keep="first")]
print("duplicate merchant_id:")
duplicate_merchant_id['merchant_id']
```

```
duplicate merchant_id:
```

Out[199]:
```
54        M_ID_c0b712e11a
112       M_ID_0039220eb3
3393      M_ID_bd49e37dda
3394      M_ID_bd49e37dda
3395      M_ID_bd49e37dda
              ...
333905    M_ID_6464db3b45
333906    M_ID_6464db3b45
334071    M_ID_1802942aaf
334072    M_ID_1802942aaf
334073    M_ID_1802942aaf
Name: merchant_id, Length: 63, dtype: object
```

So we have few duplicate merchant_id, we will remove them and keep only the first record in feature engineering steps to avoid further confusion ie which records to be considered for a merchant_id

In [200]:
```
print("duplicate rows in merchants:",merchants[merchants.duplicated(subset=None, keep='first')])
```

```
duplicate rows in merchants: Empty DataFrame
Columns: [merchant_id, merchant_group_id, merchant_category_id, subsector_id, numerical_1, numerical_2, category_1, most_recent_sales_range, most_recent_purchases_range, avg_sales_lag3, avg_purchases_lag3, active_months_lag3, avg_sales_lag6, avg_purchases_lag6, active_months_lag6, avg_sales_lag12, avg_purchases_lag12, active_months_lag12, category_4, city_id, state_id, category_2]
Index: []

[0 rows x 22 columns]
```

Hence, there is no duplicate rows

In [201]:
```
#check for number of unique values
merchants.nunique(axis=0)
```

Out[201]:
```
merchant_id                  334633
merchant_group_id            109391
merchant_category_id            324
subsector_id                     41
numerical_1                     954
numerical_2                     947
category_1                        2
most_recent_sales_range           5
most_recent_purchases_range       5
avg_sales_lag3                 3372
avg_purchases_lag3           100003
active_months_lag3                3
avg_sales_lag6                 4507
avg_purchases_lag6           135202
active_months_lag6                6
avg_sales_lag12                5009
avg_purchases_lag12          172917
active_months_lag12              12
category_4                        2
city_id                         271
state_id                         25
category_2                        5
dtype: int64
```

In [202]:
```
# check whether NULL value present

print("*********************************************************")
print("For merchant data")
print("*********************************************************")
for i in merchants.columns:
    if (merchants[i].isnull().sum() > 0):
        print("Total NULL values in ", i ,":",merchants[i].isnull().sum())
```

```
*********************************************************
For merchant data
*********************************************************
Total NULL values in  avg_sales_lag3 : 13
Total NULL values in  avg_sales_lag6 : 13
Total NULL values in  avg_sales_lag12 : 13
Total NULL values in  category_2 : 11887
```

In the given merchant dataset , we hvae NaN values for the features avg_sales_lag3, avg_sales_lag6, avg_sales_lag12 , category_2. We will need to replace them with proper values.

let's check the common features for transactions and merchant Dataset

In [203]:
```
common_features=set(new_merchant_transactions.columns).intersection(set(mercha
nts.columns))
common_features
```

Out[203]:
```
{'category_1',
 'category_2',
 'city_id',
 'merchant_category_id',
 'merchant_id',
 'state_id',
 'subsector_id'}
```

In [205]:
```
set(merchants.columns).difference(common_features)
```

Out[205]:
```
{'active_months_lag12',
 'active_months_lag3',
 'active_months_lag6',
 'avg_purchases_lag12',
 'avg_purchases_lag3',
 'avg_purchases_lag6',
 'avg_sales_lag12',
 'avg_sales_lag3',
 'avg_sales_lag6',
 'category_4',
 'merchant_group_id',
 'most_recent_purchases_range',
 'most_recent_sales_range',
 'numerical_1',
 'numerical_2'}
```

For the common features we will check more closely if the distribution is same and entries are same or not.

Since we have already observed there are few columns which present both in transactions and merchant dataset so at first we will check whether they're same or not for the common merchant_id.

To verify this we will pick few records from the historical_transaction dataset and verify whether same or different.

In [206]:
```
# we will create one temporary dataset for the historical transaction data  me
rchant and extract only the common features which
#are present both in transaction and merchant dataset.
temp_hist=historical_transactions[['category_1','category_2','city_id','mercha
nt_category_id','merchant_id','state_id','subsector_id']]
temp_merchant=merchants[['category_1','category_2','city_id','merchant_categor
y_id','merchant_id','state_id','subsector_id']]

#consider only top 100000 data
temp_hist=temp_hist.head(100000)

#also extract those rows only from merchant dataset for which the merchant_id
 also present in temp_hist dataframe,
#and construct the new temp1 dataframe
a=temp_hist['merchant_id']
a=a.tolist()
temp1=temp_merchant.loc[temp_merchant['merchant_id'].isin(a)]

# for verification purpose let's delete the duplicate merchant_id as well
temp_hist.drop_duplicates(subset ="merchant_id", keep = False, inplace = True)
temp1.drop_duplicates(subset ="merchant_id", keep = False, inplace = True)

#sort them based on merchant_id
temp1=temp1.sort_values(by="merchant_id")
temp_hist=temp_hist.sort_values(by="merchant_id")

# for simplicity, resetting the index
temp_hist.reset_index()
temp1.reset_index()

# concat them and as defined if there is the row is same for both dataframes t
hen it will not be stored
#ref: https://stackoverflow.com/questions/20225110/comparing-two-dataframes-an
d-getting-the-differences
df_diff = pd.concat([temp_hist,temp1]).drop_duplicates(keep=False)
df_diff
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:17: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
table/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[206]:

	category_1	category_2	city_id	merchant_category_id	merchant_id	state_id	subs
67290	N	3.0	190	705	M_ID_00087160af	7	
84951	N	5.0	268	422	M_ID_00313521c7	21	
23046	N	1.0	331	307	M_ID_003271751d	16	
93155	N	1.0	69	317	M_ID_007e2cbe55	9	
25183	N	3.0	286	454	M_ID_009ae274e9	3	
...	...	...	...	...	...	...	
326882	N	1.0	213	783	M_ID_ffea322b04	9	
331847	N	3.0	25	367	M_ID_ffeb1929c8	7	
129739	N	3.0	-1	80	M_ID_ffec24cacd	8	
304969	N	1.0	233	528	M_ID_ffefc89829	9	
330285	N	4.0	-1	783	M_ID_fff6541e56	4	

17158 rows × 7 columns

As there are many rows present in dt_diff , so it means many of the values are different in transaction and merchant data for a given merchant_id

We will quickly check once for the merchant_id M_ID_00313521c7 to understand better

In [207]:
```
temp_hist[temp_hist['merchant_id'] == 'M_ID_00313521c7']
```

Out[207]:

	category_1	category_2	city_id	merchant_category_id	merchant_id	state_id	subse
84951	N	5.0	268	422	M_ID_00313521c7	21	

In [208]:
```
temp1[temp1['merchant_id'] == 'M_ID_00313521c7']
```

Out[208]:

	category_1	category_2	city_id	merchant_category_id	merchant_id	state_id	subs
220034	N	5.0	-1	422	M_ID_00313521c7	21	

As we could see , the city_id is different in merchant and transaction data. Also , to verify this we have also removed the duplicate merchant_id from the temp_hist dataframe.

So we can conclude , even though there are few common columns in merchant and transaction dataset , still the value could be different for a same merchant_id . Although this is a problem but we will consider only the value present in transaction dataset for the common feature

In [209]:
```
del df_diff,temp1,temp_hist,temp_merchant
```

In [210]:
```
merchant_id_transaction=set(historical_transactions['merchant_id']).union(set(
new_merchant_transactions['merchant_id']))
merchant_id_merchant=set(merchants['merchant_id'])
merchant_id_difference=merchant_id_transaction.difference(merchant_id_merchant
)
merchant_id_difference
```

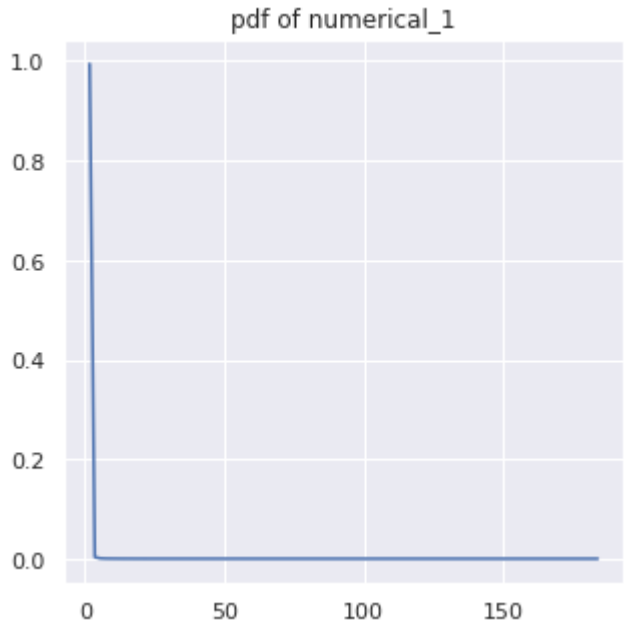
Out[210]:
```
{nan}
```

As the difference returns {NaN} , so we can conclude all the merchant_id which are present in transactions dataset are also present in merchant dataset.

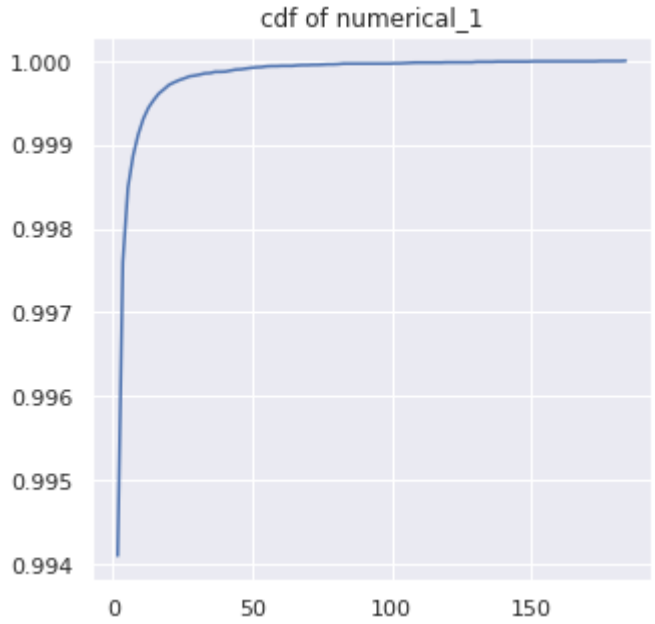
Instead of empty set it returns {NaN} as there are few NULL merchant_id exist in transaction dataset

Plot Numerical_1:

In [211]:
```
sns.set(rc={'figure.figsize':(5,5)})
counts,bin_edges = np.histogram (merchants['numerical_1'],bins=100, density =
True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of numerical_1")
plt.show()
```



In [212]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of numerical_1")
plt.show()
```



In [213]:
```
for i in [10,20,30,40,50,60,70,80,90,100]:
    print("The" ,i, "th percentile :", np.percentile(merchants['numerical_1'],
[i]))
```

```
The 10 th percentile : [-0.05747065]
The 20 th percentile : [-0.05747065]
The 30 th percentile : [-0.05747065]
The 40 th percentile : [-0.05747065]
The 50 th percentile : [-0.05747065]
The 60 th percentile : [-0.05747065]
The 70 th percentile : [-0.04755575]
The 80 th percentile : [-0.04755575]
The 90 th percentile : [-0.00789613]
The 100 th percentile : [183.73511137]
```

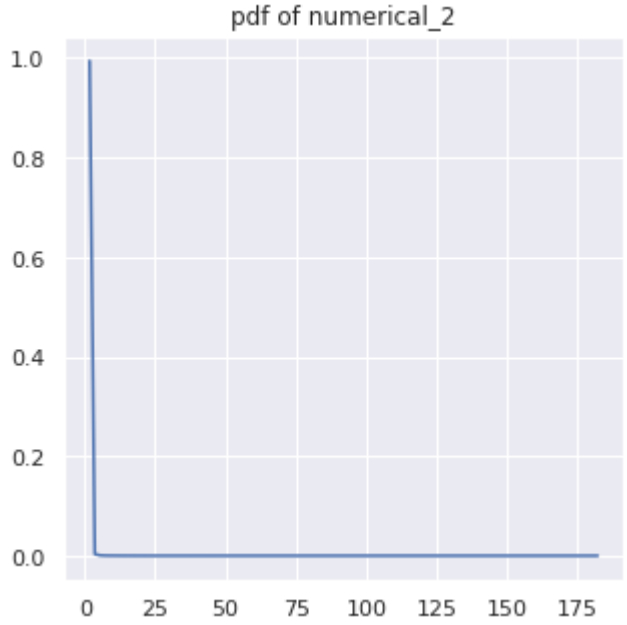
In [214]:
```
for i in range(90,101):
    print("The" ,i, "th percentile :", np.percentile(merchants['numerical_1'],
[i]))
```

```
The 90 th percentile : [-0.00789613]
The 91 th percentile : [0.00201877]
The 92 th percentile : [0.01193368]
The 93 th percentile : [0.03176349]
The 94 th percentile : [0.0615082]
The 95 th percentile : [0.09125291]
The 96 th percentile : [0.15074234]
The 97 th percentile : [0.25980629]
The 98 th percentile : [0.47793418]
The 99 th percentile : [1.06291354]
The 100 th percentile : [183.73511137]
```

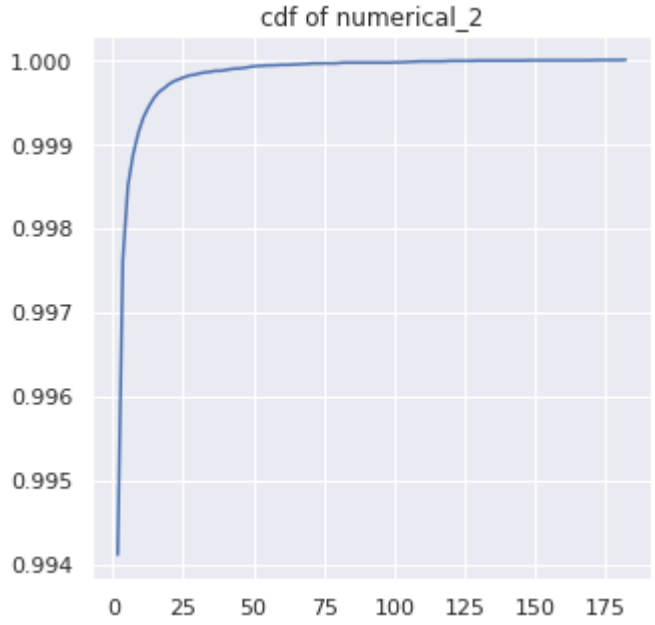
So , around 1% data has higher numerical_1 value

Plot Numerical_2:

In [216]:
```
counts,bin_edges = np.histogram (merchants['numerical_2'],bins=100, density =
True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of numerical_2")
plt.show()
```



In [217]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of numerical_2")
plt.show()
```



In [218]:
```
for i in [10,20,30,40,50,60,70,80,90,100]:
    print("The:", i , "th percentile:" , np.percentile(merchants['numerical_2'
],[i]))
```

```
The: 10 th percentile: [-0.05747065]
The: 20 th percentile: [-0.05747065]
The: 30 th percentile: [-0.05747065]
The: 40 th percentile: [-0.05747065]
The: 50 th percentile: [-0.05747065]
The: 60 th percentile: [-0.05747065]
The: 70 th percentile: [-0.05747065]
The: 80 th percentile: [-0.04755575]
The: 90 th percentile: [-0.01781104]
The: 100 th percentile: [182.07932234]
```

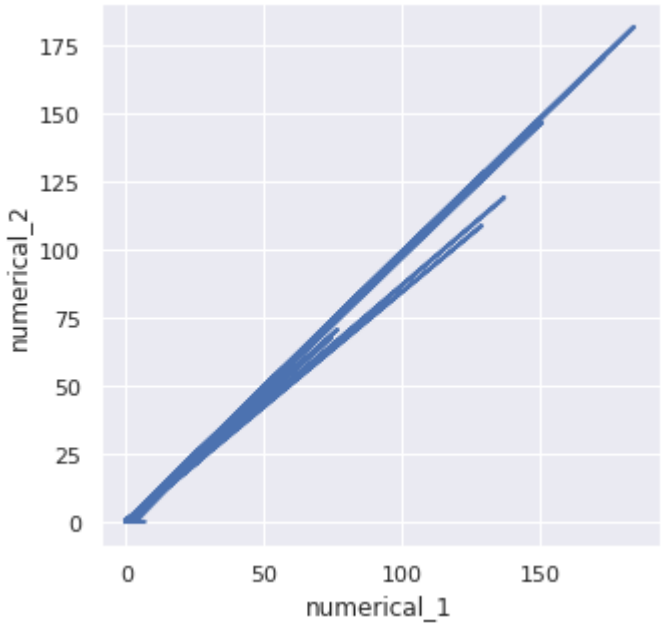
In [219]:
```
for i in range(90,101):
    print("The:", i , "th percentile:" , np.percentile(merchants['numerical_2'
],[i]))
```

```
The: 90 th percentile: [-0.01781104]
The: 91 th percentile: [-0.00789613]
The: 92 th percentile: [0.00201877]
The: 93 th percentile: [0.02184858]
The: 94 th percentile: [0.04167839]
The: 95 th percentile: [0.08133801]
The: 96 th percentile: [0.14082743]
The: 97 th percentile: [0.23997648]
The: 98 th percentile: [0.45810437]
The: 99 th percentile: [1.04308373]
The: 100 th percentile: [182.07932234]
```

The distribution is similar in numerical_1 and numerical_2 feature.

let's plot them together

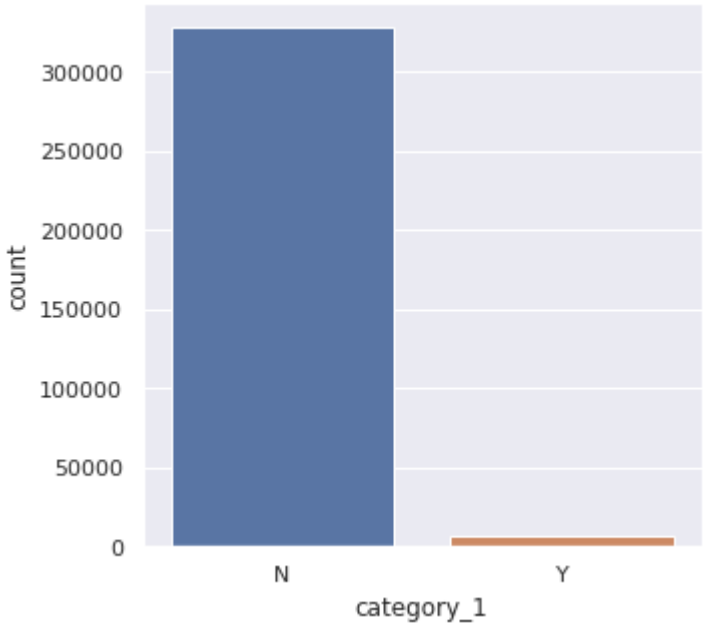
In [220]:
```
plt.plot(merchants['numerical_1'],merchants['numerical_2'],linewidth=2.0)
plt.xlabel('numerical_1')
plt.ylabel('numerical_2')
plt.show()
```



So, both the features - numerical_1 and numerical_2 are highly correlated with each other. In feature engineering steps we will check if we need to use both of them

Plot category_1:

In [221]:
```
sns.set_style(style="darkgrid")
ax=sns.countplot(x="category_1",data=merchants)
```



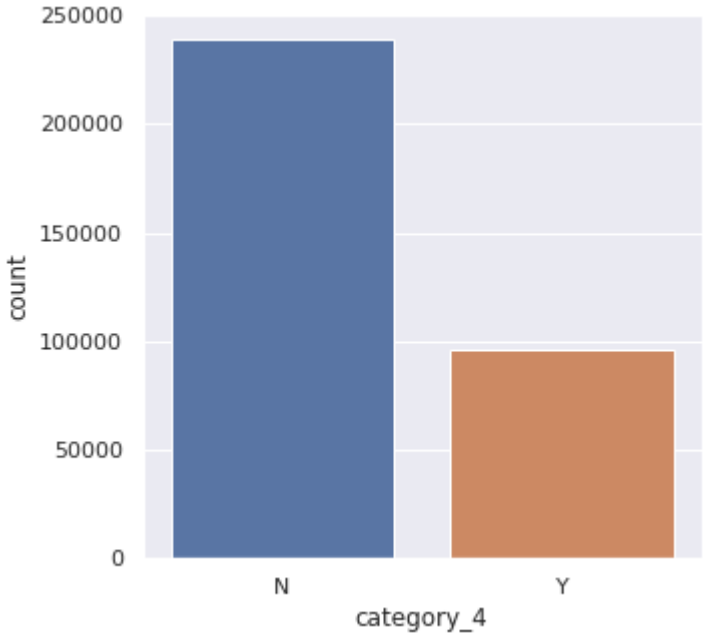
In [222]:
```
result=merchants[merchants['category_1'] == 'Y']
percentage_Y=len(result)/len(merchants['category_1'])
print("Percentage of category_1 with Y:",percentage_Y*100)
```

```
Percentage of category_1 with Y: 2.1031025169108686
```

So only 2.10% data are with 'Y' for category_1 which we also observed in transaction data as well where also most data come under 'N' category

Plot category_4:

In [223]:
```
sns.set_style(style="darkgrid")
ax=sns.countplot(x="category_4",data=merchants)
```



In [224]:
```
result=merchants[merchants['category_4'] == 'Y']
percentage_Y=len(result)/len(merchants['category_4'])
print("Percentage of category_4 with Y:",percentage_Y*100)
```

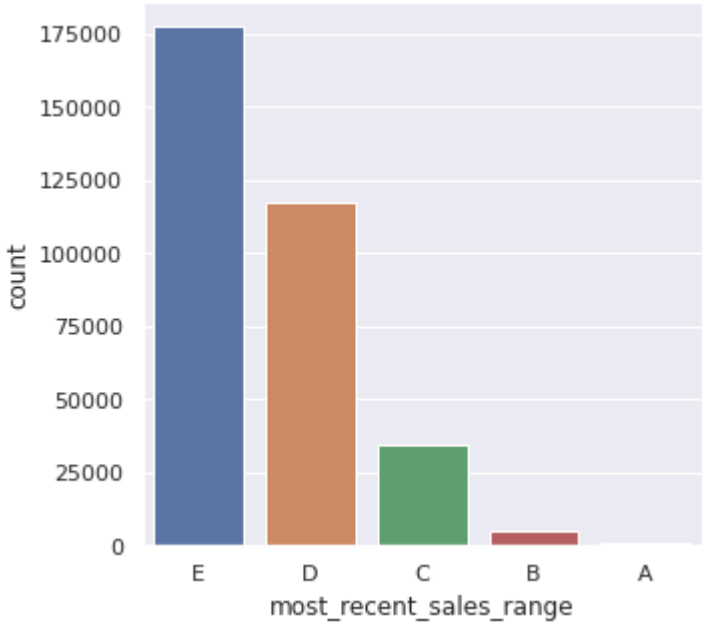
```
Percentage of category_4 with Y: 28.712622798001767
```

More than 70% data are from 'N' category in category_4

Plot Most Recent Sales & Purchase Range

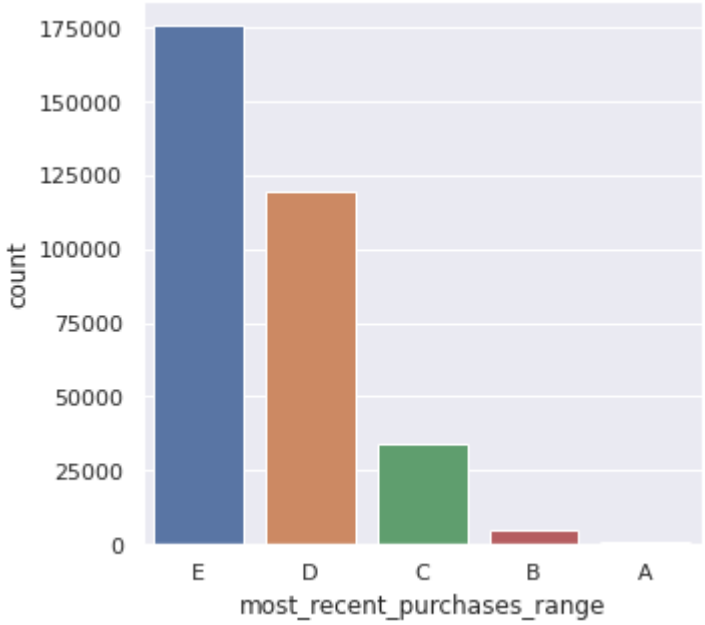
In [274]:
```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style(style="darkgrid")
sns.countplot(x="most_recent_sales_range",data=merchants,order=merchants['most
_recent_sales_range'].value_counts().index)
```

Out[274]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5659ec8f10>



In [275]:
```
sns.countplot(x="most_recent_purchases_range",data=merchants,order=merchants[
'most_recent_purchases_range'].value_counts().index)
```

Out[275]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5659eaa810>



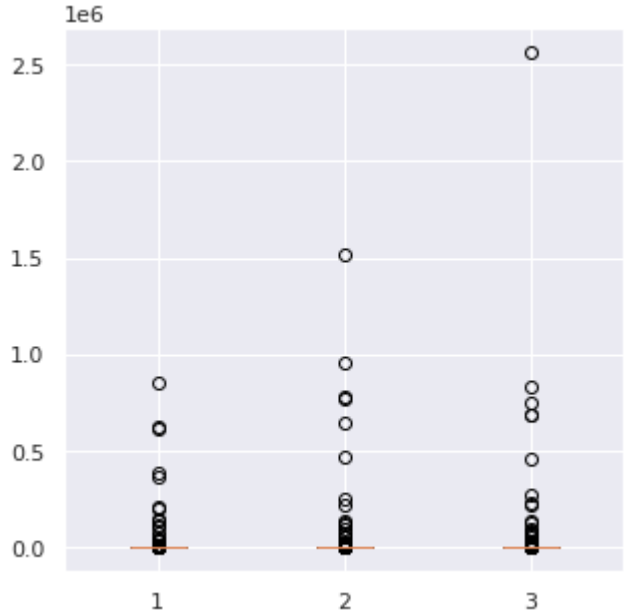
In [227]:
```
for i in ['A','B','C','D','E']:
    result=merchants[merchants['most_recent_sales_range'] == i]
    print("Percentage of:",i, "in most_recent_sales_range:",len(result)/len(me
rchants['most_recent_sales_range'])*100)
    result=merchants[merchants['most_recent_purchases_range'] == i]
    print("Percentage of:",i, "in most_recent_purchases_range:",len(result)/le
n(merchants['most_recent_purchases_range'])*100)

    #result=merchants[merchants['category_4'] != 'N']
```

```
Percentage of: A in most_recent_sales_range: 0.30027248607691753
Percentage of: A in most_recent_purchases_range: 0.3017663790424744
Percentage of: B in most_recent_sales_range: 1.504947773501924
Percentage of: B in most_recent_purchases_range: 1.5076367808399265
Percentage of: C in most_recent_sales_range: 10.180880560269618
Percentage of: C in most_recent_purchases_range: 10.201496283194302
Percentage of: D in most_recent_sales_range: 35.09901522575711
Percentage of: D in most_recent_purchases_range: 35.610524177163754
Percentage of: E in most_recent_sales_range: 52.91488395439443
Percentage of: E in most_recent_purchases_range: 52.378576379759544
```

So it's clear the distribution is almost same for both the features most_recent_sales_range & most_recent_purchases_range

Plot Average Sales

In [228]:
```
data=[merchants['avg_sales_lag3'].dropna().values,merchants['avg_sales_lag6'].
dropna().values,merchants['avg_sales_lag12'].dropna().values]
plt.boxplot(data)
#plt.boxplot(merchants['avg_sales_lag6'].dropna().values)
plt.show()
```



In [229]:
```
result=merchants[merchants['avg_purchases_lag3'] == np.inf]
result
```

Out[229]:

	merchant_id	merchant_group_id	merchant_category_id	subsector_id	numerical_1	nun
10	M_ID_492cfa500c	13462	369	27	-0.057471	-
11	M_ID_73487fed26	17123	427	27	-0.057471	-
12	M_ID_7149162139	2118	63	27	-0.057471	-

3 rows × 22 columns

In [230]:
```
result=merchants[merchants['avg_purchases_lag6'] == np.inf]
result
```

Out[230]:

	merchant_id	merchant_group_id	merchant_category_id	subsector_id	numerical_1	nun
10	M_ID_492cfa500c	13462	369	27	-0.057471	-
11	M_ID_73487fed26	17123	427	27	-0.057471	-
12	M_ID_7149162139	2118	63	27	-0.057471	-

3 rows × 22 columns

In [231]:
```
result=merchants[merchants['avg_purchases_lag12'] == np.inf]
result
```

Out[231]:

	merchant_id	merchant_group_id	merchant_category_id	subsector_id	numerical_1	nun
10	M_ID_492cfa500c	13462	369	27	-0.057471	-
11	M_ID_73487fed26	17123	427	27	-0.057471	-
12	M_ID_7149162139	2118	63	27	-0.057471	-

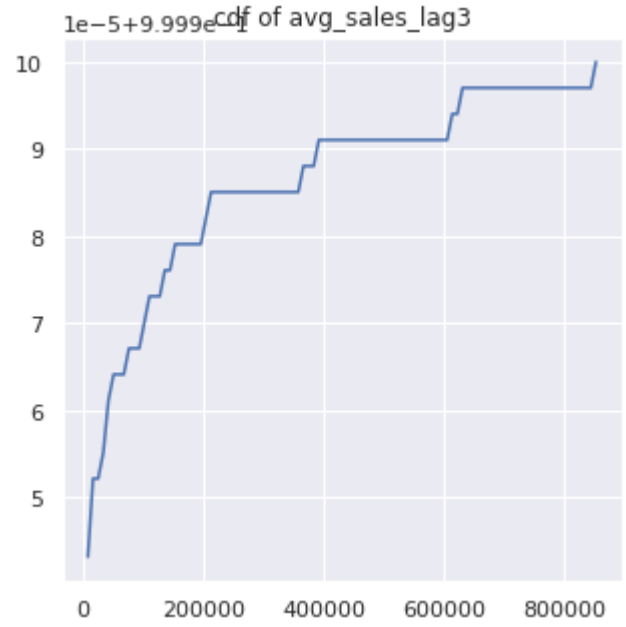
3 rows × 22 columns

in the features - avg_purchases_lag3,avg_purchases_lag6,avg_purchases_lag12 there are few inf values exist. For the time being we are going to set them as NaN.

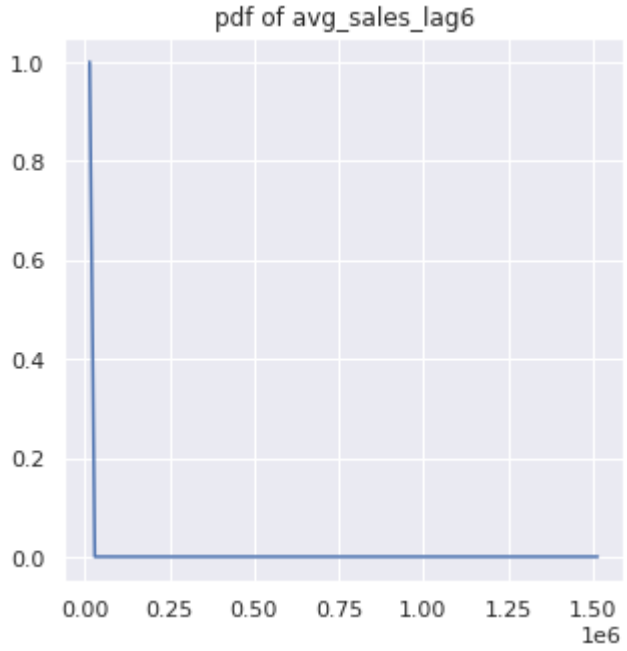
In [232]:
```
# set the inf value to NaN
merchants.replace([np.inf, -np.inf], np.nan,inplace=True)
```

In [233]:
```
counts,bin_edges = np.histogram (merchants['avg_sales_lag3'].dropna().values,b
ins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of avg_sales_lag3")
plt.show()
```

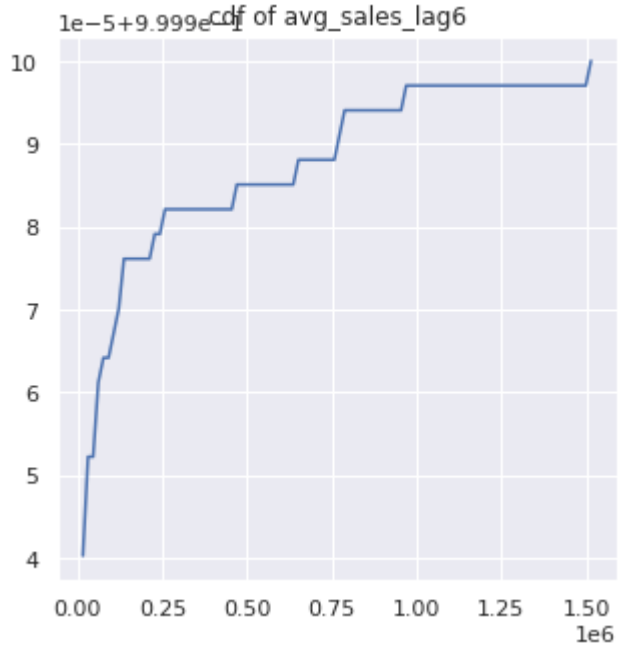
In [234]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of avg_sales_lag3")
plt.show()
```



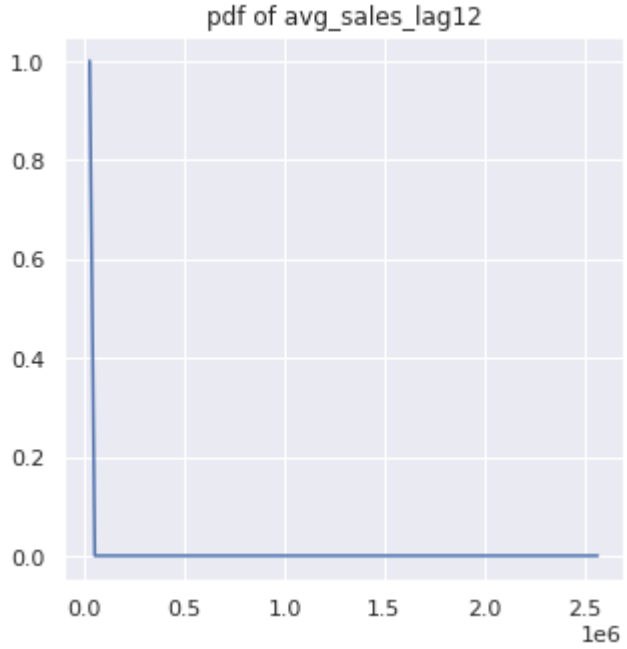
In [235]:
```
counts,bin_edges = np.histogram (merchants['avg_sales_lag6'].dropna().values,b
ins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of avg_sales_lag6")
plt.show()
```



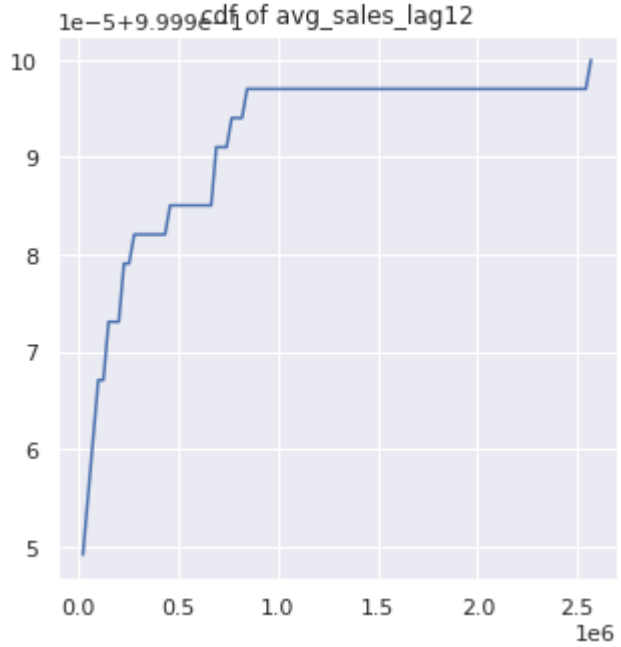
In [236]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of avg_sales_lag6")
plt.show()
```



In [237]:
```
counts,bin_edges = np.histogram (merchants['avg_sales_lag12'].dropna().values,
bins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of avg_sales_lag12")
plt.show()
```



In [238]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of avg_sales_lag12")
plt.show()
```



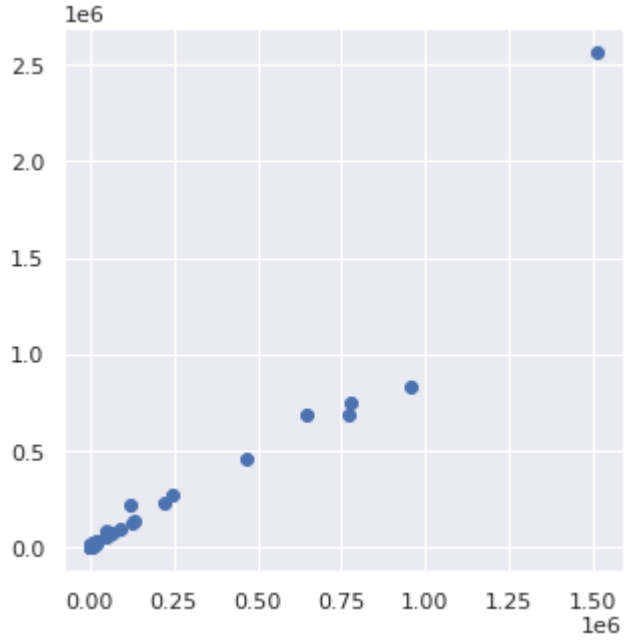
The cdfs are not completely identical, let's dive deep into the percentile data

In [239]:
```
#np.percentile(train_csv['target'],[i]))
print("***********************************************")
for i in [10,20,30,40,50,60,70,80,90,91,92,93,94,95,96,97,98,99,100]:
    print("The:",i,"th percentile of avg_sales_lag3:",np.percentile(merchants[
'avg_sales_lag3'].dropna().values,[i]))
    print("The:",i,"th percentile of avg_sales_lag6:",np.percentile(merchants[
'avg_sales_lag6'].dropna().values,[i]))
    print("The:",i,"th percentile of avg_sales_lag12:",np.percentile(merchants
['avg_sales_lag12'].dropna().values,[i]))
    print("***********************************************")
```

```
**********************************************
The: 10 th percentile of avg_sales_lag3: [0.74]
The: 10 th percentile of avg_sales_lag6: [0.69]
The: 10 th percentile of avg_sales_lag12: [0.67]
**********************************************
The: 20 th percentile of avg_sales_lag3: [0.85]
The: 20 th percentile of avg_sales_lag6: [0.81]
The: 20 th percentile of avg_sales_lag12: [0.8]
**********************************************
The: 30 th percentile of avg_sales_lag3: [0.91]
The: 30 th percentile of avg_sales_lag6: [0.89]
The: 30 th percentile of avg_sales_lag12: [0.89]
**********************************************
The: 40 th percentile of avg_sales_lag3: [0.96]
The: 40 th percentile of avg_sales_lag6: [0.95]
The: 40 th percentile of avg_sales_lag12: [0.96]
**********************************************
The: 50 th percentile of avg_sales_lag3: [1.]
The: 50 th percentile of avg_sales_lag6: [1.01]
The: 50 th percentile of avg_sales_lag12: [1.02]
**********************************************
The: 60 th percentile of avg_sales_lag3: [1.05]
The: 60 th percentile of avg_sales_lag6: [1.07]
The: 60 th percentile of avg_sales_lag12: [1.1]
**********************************************
The: 70 th percentile of avg_sales_lag3: [1.11]
The: 70 th percentile of avg_sales_lag6: [1.16]
The: 70 th percentile of avg_sales_lag12: [1.21]
**********************************************
The: 80 th percentile of avg_sales_lag3: [1.23]
The: 80 th percentile of avg_sales_lag6: [1.33]
The: 80 th percentile of avg_sales_lag12: [1.41]
**********************************************
The: 90 th percentile of avg_sales_lag3: [1.58]
The: 90 th percentile of avg_sales_lag6: [1.87]
The: 90 th percentile of avg_sales_lag12: [2.03]
**********************************************
The: 91 th percentile of avg_sales_lag3: [1.66]
The: 91 th percentile of avg_sales_lag6: [2.]
The: 91 th percentile of avg_sales_lag12: [2.19]
**********************************************
The: 92 th percentile of avg_sales_lag3: [1.77]
The: 92 th percentile of avg_sales_lag6: [2.19]
The: 92 th percentile of avg_sales_lag12: [2.4]
**********************************************
The: 93 th percentile of avg_sales_lag3: [1.91]
The: 93 th percentile of avg_sales_lag6: [2.41]
The: 93 th percentile of avg_sales_lag12: [2.67]
**********************************************
The: 94 th percentile of avg_sales_lag3: [2.11]
The: 94 th percentile of avg_sales_lag6: [2.72]
The: 94 th percentile of avg_sales_lag12: [3.07]
**********************************************
The: 95 th percentile of avg_sales_lag3: [2.39]
The: 95 th percentile of avg_sales_lag6: [3.22]
The: 95 th percentile of avg_sales_lag12: [3.66]
**********************************************
```

```
The: 96 th percentile of avg_sales_lag3: [2.83]
The: 96 th percentile of avg_sales_lag6: [3.97]
The: 96 th percentile of avg_sales_lag12: [4.6]
**********************************************
The: 97 th percentile of avg_sales_lag3: [3.63]
The: 97 th percentile of avg_sales_lag6: [5.33]
The: 97 th percentile of avg_sales_lag12: [6.29]
**********************************************
The: 98 th percentile of avg_sales_lag3: [5.31]
The: 98 th percentile of avg_sales_lag6: [8.31]
The: 98 th percentile of avg_sales_lag12: [10.3]
**********************************************
The: 99 th percentile of avg_sales_lag3: [10.76]
The: 99 th percentile of avg_sales_lag6: [18.7018]
The: 99 th percentile of avg_sales_lag12: [23.9318]
**********************************************
The: 100 th percentile of avg_sales_lag3: [851844.64]
The: 100 th percentile of avg_sales_lag6: [1513959.]
The: 100 th percentile of avg_sales_lag12: [2567408.]
**********************************************
```

In [240]:
```
plt.scatter(merchants['avg_sales_lag6'].dropna().values,merchants['avg_sales_lag12'].dropna().values)
plt.show()
```

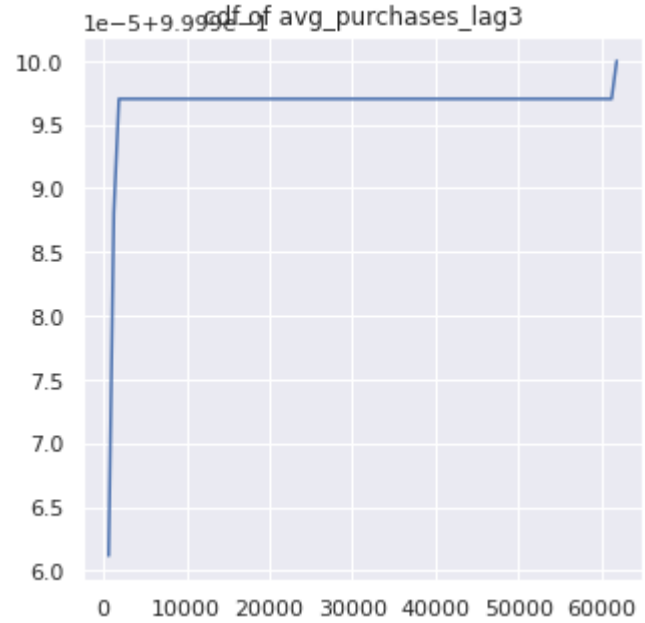


The distributions are not completely identical also the 100th percentile value is quiet large

Plot Average Purchase

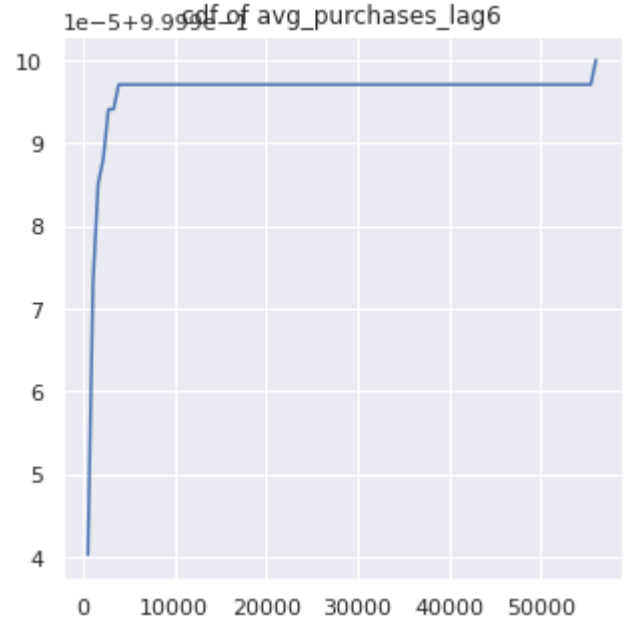
In [241]:
```
counts,bin_edges = np.histogram (merchants['avg_purchases_lag3'].dropna().valu
es,bins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of avg_purchases_lag3")
plt.show()
```

In [242]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of avg_purchases_lag3")
plt.show()
```

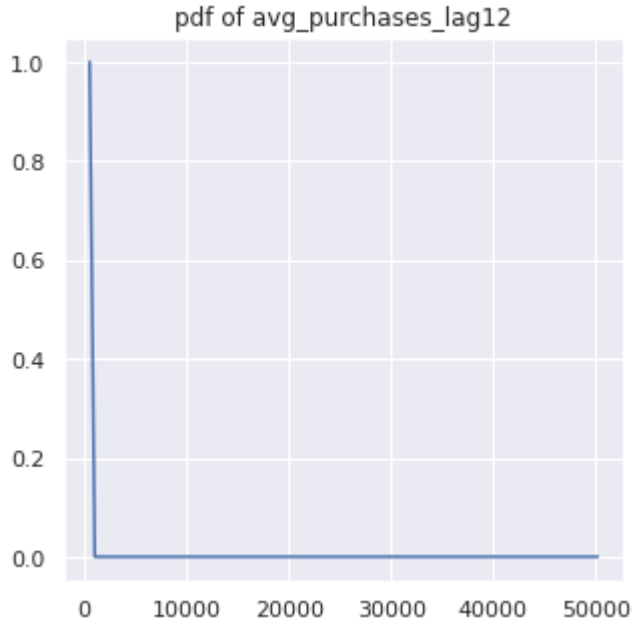


In [243]:
```
counts,bin_edges = np.histogram (merchants['avg_purchases_lag6'].dropna().valu
es,bins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of avg_purchases_lag6")
plt.show()
```

In [244]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of avg_purchases_lag6")
plt.show()
```

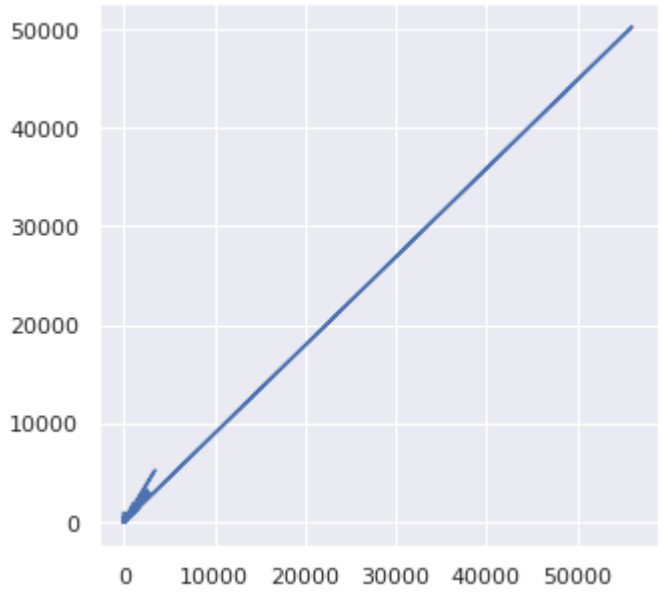


In [245]:
```
counts,bin_edges = np.histogram (merchants['avg_purchases_lag12'].dropna().val
ues,bins=100, density = True)
pdf=counts/sum(counts)
plt.plot(bin_edges[1:],pdf)
plt.title("pdf of avg_purchases_lag12")
plt.show()
```

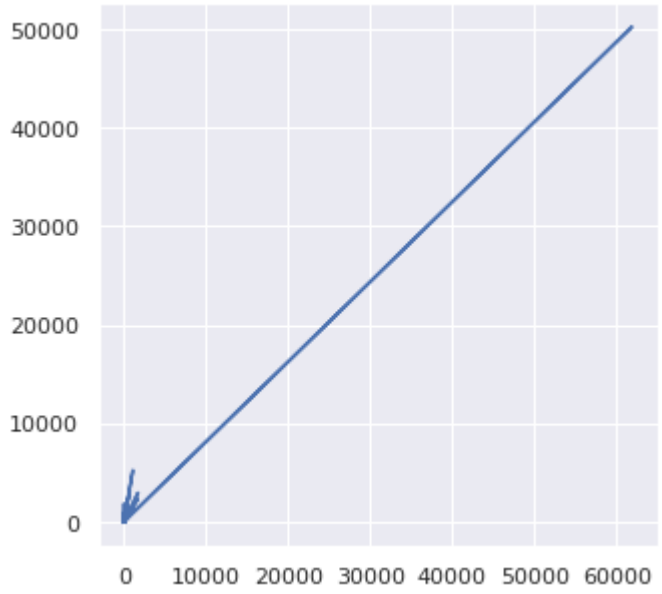


In [246]:
```
cdf=np.cumsum(pdf)
plt.plot(bin_edges[1:],cdf)
plt.title("cdf of avg_purchases_lag12")
plt.show()
```

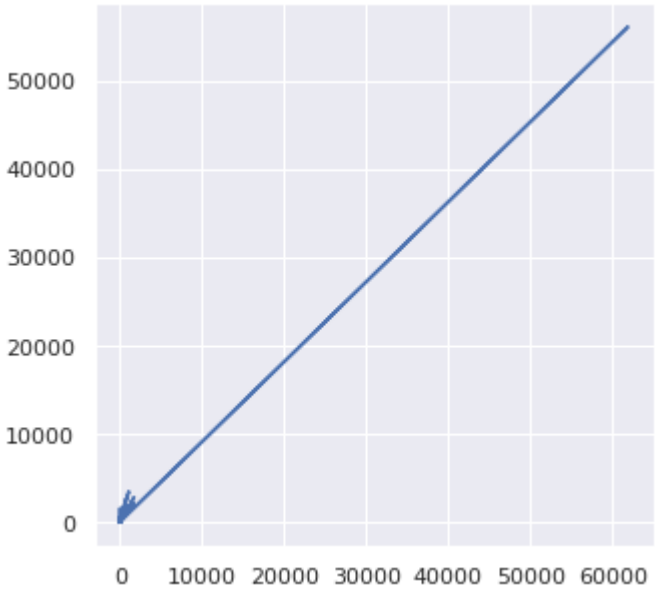
In [247]:
```
plt.plot(merchants['avg_purchases_lag6'],merchants['avg_purchases_lag12'])
plt.show()
```



In [248]:
```
plt.plot(merchants['avg_purchases_lag3'],merchants['avg_purchases_lag12'])
plt.show()
```



In [249]:
```
plt.plot(merchants['avg_purchases_lag3'].dropna().values,merchants['avg_purcha
ses_lag6'].dropna().values)
plt.show()
```



In [250]:
```
print("*********************************************")
for i in [10,20,30,40,50,60,70,80,90,91,92,93,94,95,96,97,98,99,100]:
    print("The:",i,"th percentile of avg_purchases_lag3:",np.percentile(mercha
nts['avg_purchases_lag3'].dropna().values,[i]))
    print("The:",i,"th percentile of avg_purchases_lag6:",np.percentile(mercha
nts['avg_purchases_lag6'].dropna().values,[i]))
    print("The:",i,"th percentile of avg_purchases_lag12:",np.percentile(merch
ants['avg_purchases_lag12'].dropna().values,[i]))
    print("*********************************************")
```

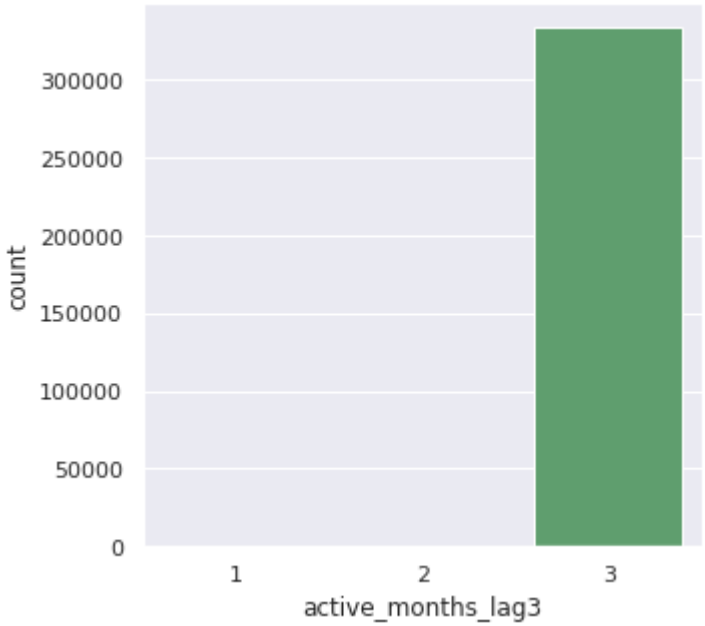
```
**********************************************
The: 10 th percentile of avg_purchases_lag3: [0.81481481]
The: 10 th percentile of avg_purchases_lag6: [0.76666667]
The: 10 th percentile of avg_purchases_lag12: [0.75154513]
**********************************************
The: 20 th percentile of avg_purchases_lag3: [0.8962963]
The: 20 th percentile of avg_purchases_lag6: [0.8681592]
The: 20 th percentile of avg_purchases_lag12: [0.86111111]
**********************************************
The: 30 th percentile of avg_purchases_lag3: [0.94647002]
The: 30 th percentile of avg_purchases_lag6: [0.93074186]
The: 30 th percentile of avg_purchases_lag12: [0.93055556]
**********************************************
The: 40 th percentile of avg_purchases_lag3: [0.98484848]
The: 40 th percentile of avg_purchases_lag6: [0.98044546]
The: 40 th percentile of avg_purchases_lag12: [0.98809386]
**********************************************
The: 50 th percentile of avg_purchases_lag3: [1.01666667]
The: 50 th percentile of avg_purchases_lag6: [1.02696078]
The: 50 th percentile of avg_purchases_lag12: [1.04336043]
**********************************************
The: 60 th percentile of avg_purchases_lag3: [1.05555556]
The: 60 th percentile of avg_purchases_lag6: [1.08333333]
The: 60 th percentile of avg_purchases_lag12: [1.11040877]
**********************************************
The: 70 th percentile of avg_purchases_lag3: [1.10909091]
The: 70 th percentile of avg_purchases_lag6: [1.16086434]
The: 70 th percentile of avg_purchases_lag12: [1.2002924]
**********************************************
The: 80 th percentile of avg_purchases_lag3: [1.2]
The: 80 th percentile of avg_purchases_lag6: [1.29738562]
The: 80 th percentile of avg_purchases_lag12: [1.36111111]
**********************************************
The: 90 th percentile of avg_purchases_lag3: [1.46911395]
The: 90 th percentile of avg_purchases_lag6: [1.71295592]
The: 90 th percentile of avg_purchases_lag12: [1.85321264]
**********************************************
The: 91 th percentile of avg_purchases_lag3: [1.53333333]
The: 91 th percentile of avg_purchases_lag6: [1.8134863]
The: 91 th percentile of avg_purchases_lag12: [1.96738285]
**********************************************
The: 92 th percentile of avg_purchases_lag3: [1.61904762]
The: 92 th percentile of avg_purchases_lag6: [1.93627451]
The: 92 th percentile of avg_purchases_lag12: [2.1185641]
**********************************************
The: 93 th percentile of avg_purchases_lag3: [1.69858364]
The: 93 th percentile of avg_purchases_lag6: [2.10643814]
The: 93 th percentile of avg_purchases_lag12: [2.32321463]
**********************************************
The: 94 th percentile of avg_purchases_lag3: [1.84657005]
The: 94 th percentile of avg_purchases_lag6: [2.33333333]
The: 94 th percentile of avg_purchases_lag12: [2.59655225]
**********************************************
The: 95 th percentile of avg_purchases_lag3: [2.01821822]
The: 95 th percentile of avg_purchases_lag6: [2.66666667]
The: 95 th percentile of avg_purchases_lag12: [3.0027673]
**********************************************
```

```
The: 96 th percentile of avg_purchases_lag3: [2.33333333]
The: 96 th percentile of avg_purchases_lag6: [3.16834576]
The: 96 th percentile of avg_purchases_lag12: [3.69067936]
************************************************
The: 97 th percentile of avg_purchases_lag3: [2.87654321]
The: 97 th percentile of avg_purchases_lag6: [4.10345618]
The: 97 th percentile of avg_purchases_lag12: [4.89731187]
************************************************
The: 98 th percentile of avg_purchases_lag3: [4.]
The: 98 th percentile of avg_purchases_lag6: [6.08333333]
The: 98 th percentile of avg_purchases_lag12: [7.5625]
************************************************
The: 99 th percentile of avg_purchases_lag3: [7.33333333]
The: 99 th percentile of avg_purchases_lag6: [12.62611111]
The: 99 th percentile of avg_purchases_lag12: [16.11733334]
************************************************
The: 100 th percentile of avg_purchases_lag3: [61851.33333333]
The: 100 th percentile of avg_purchases_lag6: [56077.5]
The: 100 th percentile of avg_purchases_lag12: [50215.55555556]
************************************************
```

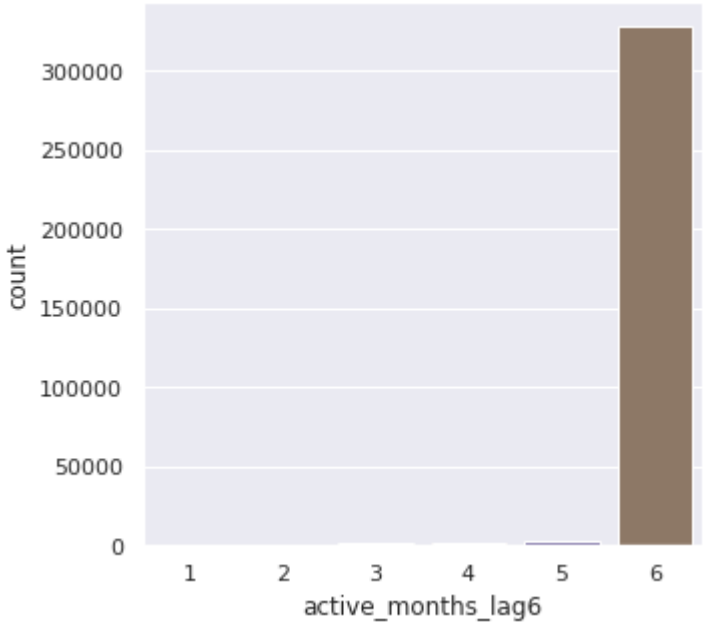
this phenomenon is more or less same as sales . But here the distributions are almost similar

Plot Active Month Lag

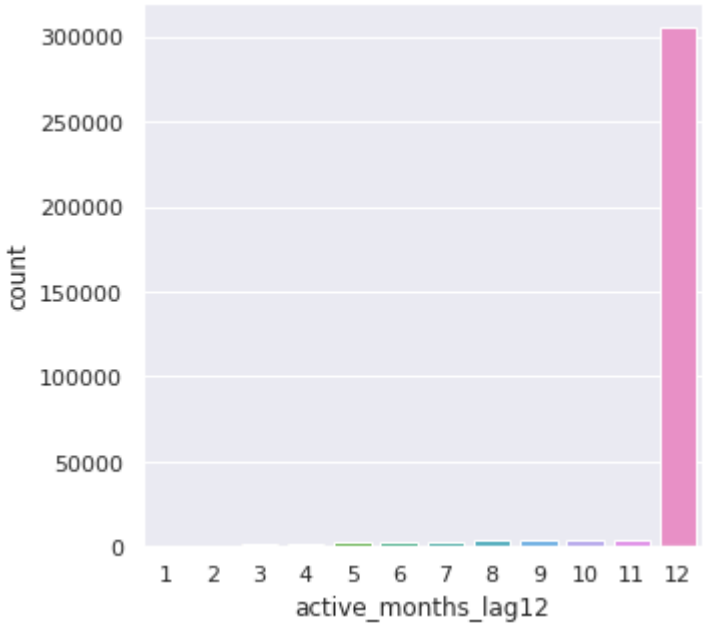
In [251]:
```
ax=sns.countplot(x="active_months_lag3",data=merchants)
```



In [252]:
```
ax=sns.countplot(x="active_months_lag6",data=merchants)
```

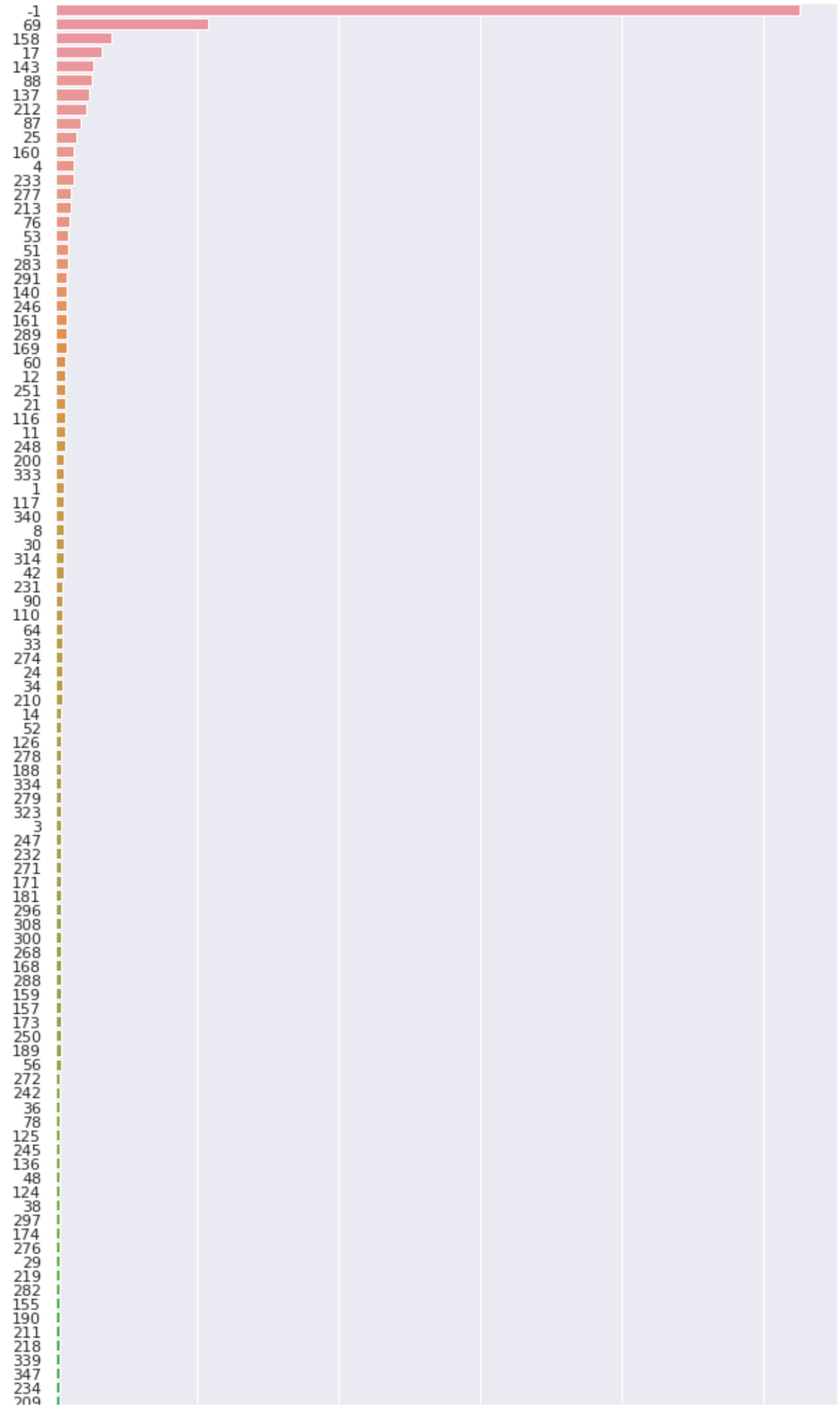


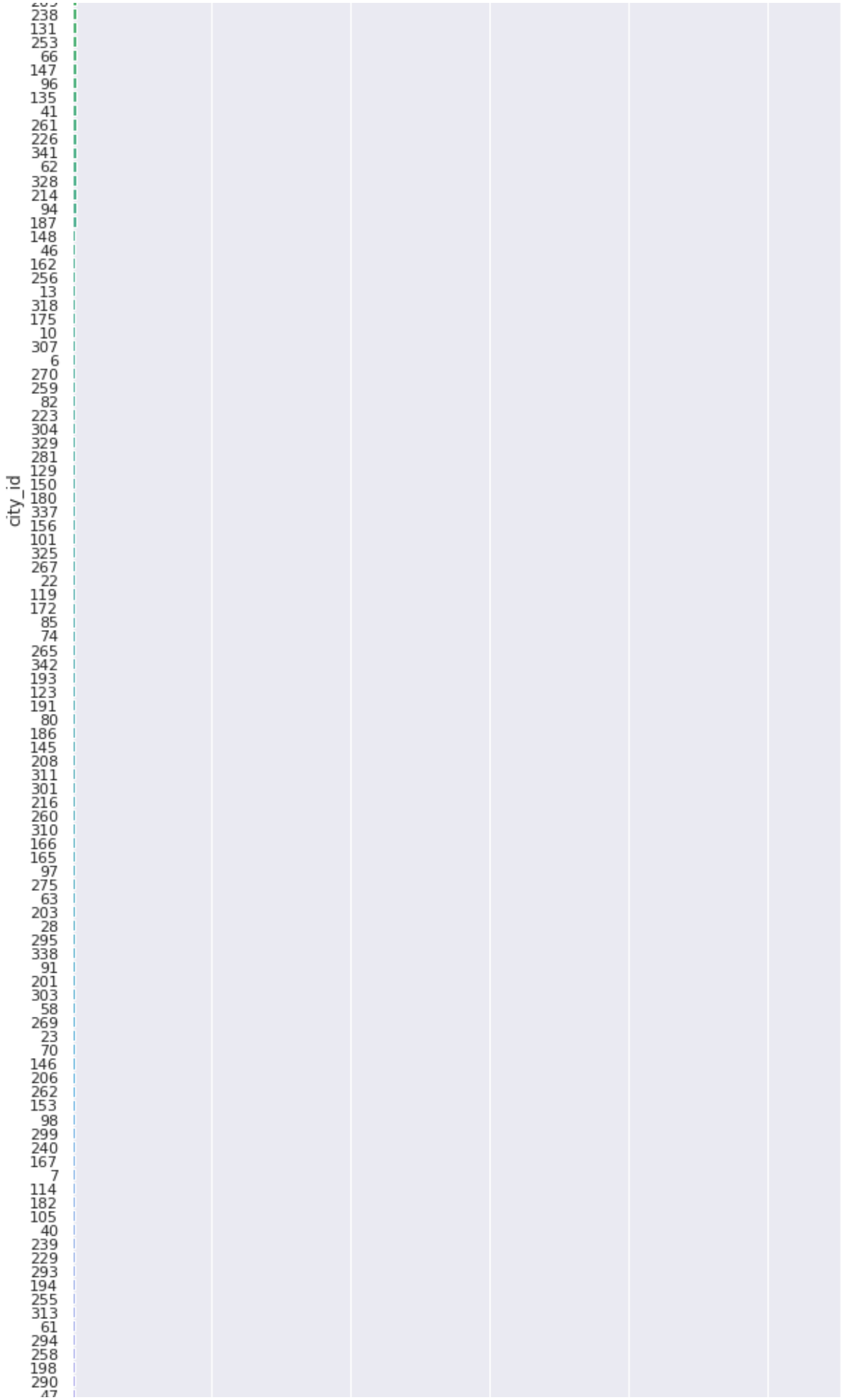
In [253]:
```
ax=sns.countplot(x="active_months_lag12",data=merchants)
```



Plot City ID

In [276]:
```
sns.set(rc={'figure.figsize':(10,50)})
ax=sns.countplot(y="city_id",data=merchants,order=merchants['city_id'].value_c
ounts().index)
```

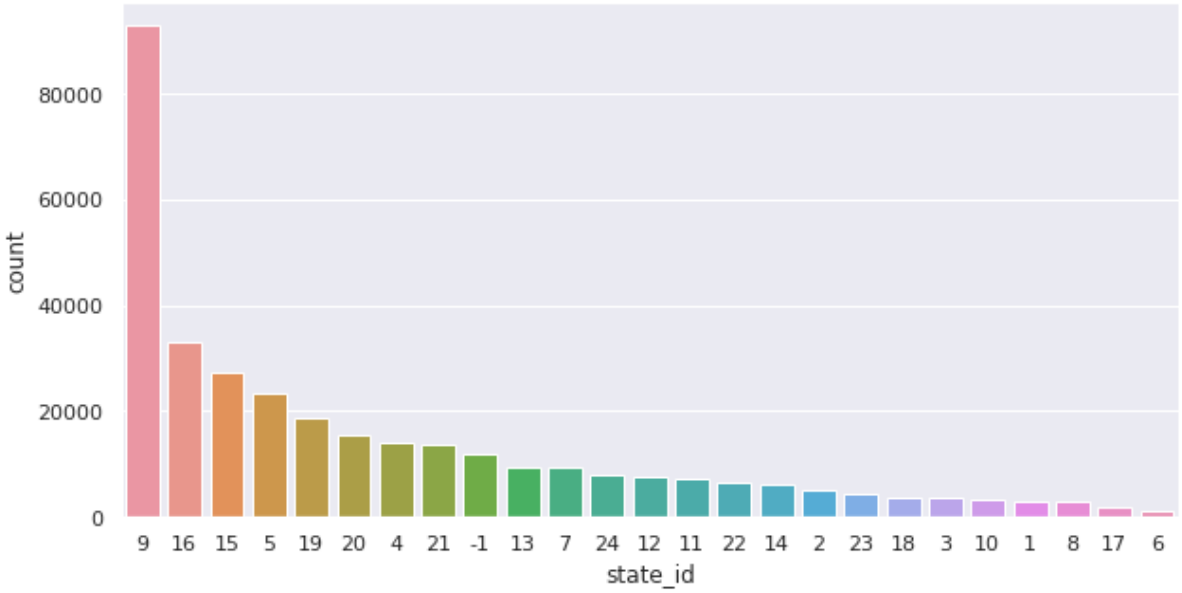

-1
69
158
17
143
88
137
212
87
25
160
4
233
277
213
76
53
51
283
291
140
246
161
289
169
60
12
251
21
116
11
248
200
333
1
117
340
8
30
314
42
231
90
110
64
33
274
24
34
210
14
52
126
278
188
334
279
323
3
247
232
271
171
181
296
308
300
268
168
288
159
157
173
250
189
56
272
242
36
78
125
245
136
48
124
38
297
174
276
29
219
282
155
190
211
218
339
347
234
209


city_id
238
131
253
66
147
96
135
41
261
226
341
62
328
214
94
187
148
46
162
256
13
318
175
10
307
6
270
259
82
223
304
329
281
129
150
180
337
156
101
325
267
22
119
172
85
74
265
342
193
123
191
80
186
145
208
311
301
216
260
310
166
165
97
275
63
203
28
295
338
91
201
303
58
269
23
70
146
206
262
153
98
299
240
167
7
114
182
105
40
239
229
293
194
255
313
61
294
258
198
290

Here, the city_id with -1 having maximum number of merchants. Although in transaction data the city_id with -1,69,158 hold maximum number of transactions.

Plot State ID:

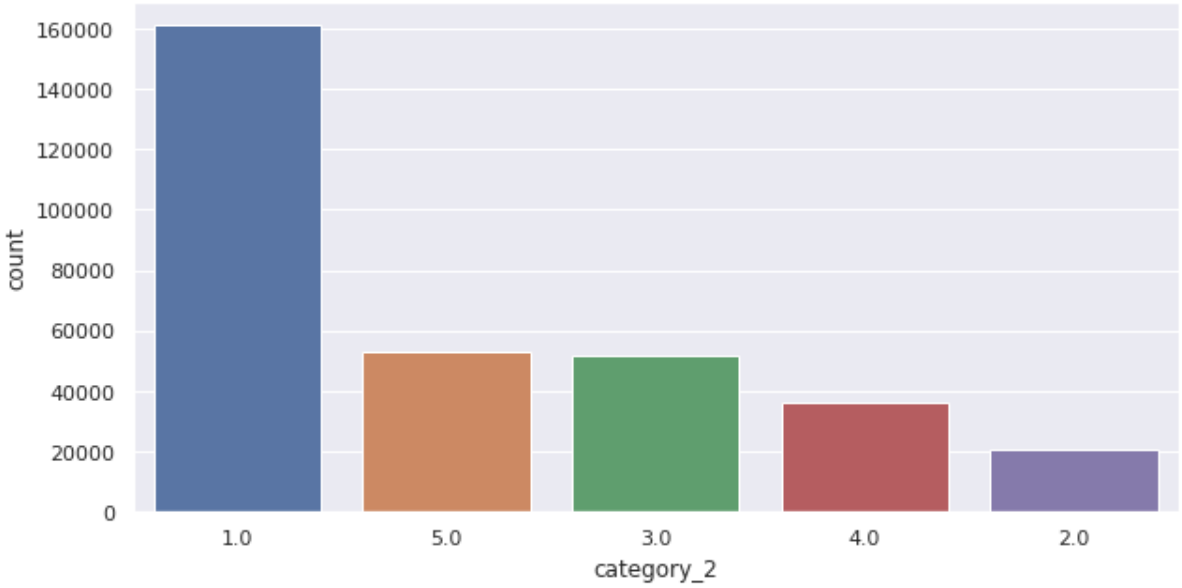
In [277]:
```
sns.set(rc={'figure.figsize':(10,5)})
ax=sns.countplot(x="state_id",data=merchants,order=merchants['state_id'].value
_counts().index)
```



In state_id also id with (-1,5,9,15,16,19) are having maximum record same as transaction data

Plot Category_2:

In [278]:
```
ax=sns.countplot(x="category_2",data=merchants,order=merchants['category_2'].v
alue_counts().index)
```



In [257]:
```
for i in [1.0,2.0,3.0,4.0,5.0]:
    result=merchants[merchants['category_2'] == i]
    percentage_Y=len(result)/len(merchants['category_2'])
    print("Percentage of category_2 with",i,":",percentage_Y*100)
```

```
Percentage of category_2 with 1.0 : 48.06989028850061
Percentage of category_2 with 2.0 : 6.173064512273824
Percentage of category_2 with 3.0 : 15.502724860769176
Percentage of category_2 with 4.0 : 10.890479718909098
Percentage of category_2 with 5.0 : 15.812259483232546
```

The percentage of each category_2 is expected as we observed the same in transaction data also.

Now , let's merge the datasets and start bi-variate analysis

In [ ]:
```
transaction=pd.concat([historical_transactions,new_merchant_transactions])
```

In [ ]:
```
common_features=set(new_merchant_transactions.columns).intersection(set(mercha
nts.columns))
common_features
```

Out[ ]:
```
{'category_1',
 'category_2',
 'city_id',
 'merchant_category_id',
 'merchant_id',
 'state_id',
 'subsector_id'}
```

In [ ]:
```
#we will oad only those features which are not present in transaction dataset
 . As we already observed even though
#there are some mismatches in between transaction data and merchant dataset fo
r the common features, still we will consider
# only the record present in transaction data
set(merchants.columns).difference(common_features)
```

Out[ ]:
```
{'active_months_lag12',
 'active_months_lag3',
 'active_months_lag6',
 'avg_purchases_lag12',
 'avg_purchases_lag3',
 'avg_purchases_lag6',
 'avg_sales_lag12',
 'avg_sales_lag3',
 'avg_sales_lag6',
 'category_4',
 'merchant_group_id',
 'most_recent_purchases_range',
 'most_recent_sales_range',
 'numerical_1',
 'numerical_2'}
```

In [ ]:
```
merchants=merchants[['merchant_id','active_months_lag12','active_months_lag3',
 'active_months_lag6',
 'avg_purchases_lag12',
 'avg_purchases_lag3',
 'avg_purchases_lag6',
 'avg_sales_lag12',
 'avg_sales_lag3',
 'avg_sales_lag6',
 'category_4',
 'merchant_group_id',
 'most_recent_purchases_range',
 'most_recent_sales_range',
 'numerical_1',
 'numerical_2']]
```

In [ ]:
```
data=transaction.merge(merchants,how='outer',on='merchant_id')
```

In [ ]:
```
data.columns
```

Out[ ]:
```
Index(['authorized_flag', 'card_id', 'city_id_x', 'category_1_x',
       'installments', 'category_3', 'merchant_category_id_x', 'merchant_id',
       'month_lag', 'purchase_amount', 'purchase_date', 'category_2_x',
       'state_id_x', 'subsector_id_x', 'merchant_group_id',
       'merchant_category_id_y', 'subsector_id_y', 'numerical_1',
       'numerical_2', 'category_1_y', 'most_recent_sales_range',
       'most_recent_purchases_range', 'avg_sales_lag3', 'avg_purchases_lag3',
       'active_months_lag3', 'avg_sales_lag6', 'avg_purchases_lag6',
       'active_months_lag6', 'avg_sales_lag12', 'avg_purchases_lag12',
       'active_months_lag12', 'category_4', 'city_id_y', 'state_id_y',
       'category_2_y'],
      dtype='object')
```

In [ ]:
```
data.head()
```

Out[ ]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_cat
0	Y	C_ID_4e6213e9bc	88	N	0	A	
1	Y	C_ID_4e6213e9bc	88	N	0	A	
2	Y	C_ID_4e6213e9bc	88	N	0	A	
3	Y	C_ID_4e6213e9bc	88	N	0	A	
4	Y	C_ID_4e6213e9bc	88	N	0	A	

In [ ]:
```
#saving the data for future use
data.to_csv('data.csv')
```

In [ ]:
```
#Since uni-variate analysis is already completed so for rest of the EDA purpos
e we will load only 50 lakhs rows
data=pd.read_csv('datas.csv',nrows=5000000)
```

In [6]:
```
data.columns
```

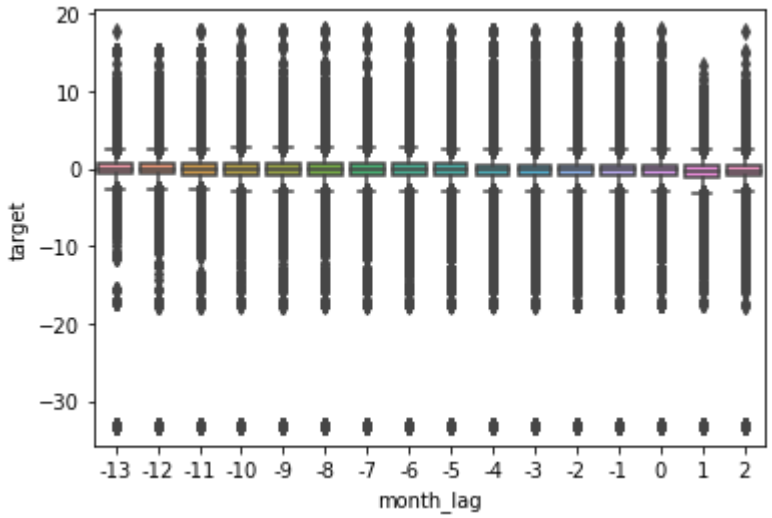
Out[6]:
```
Index(['Unnamed: 0', 'authorized_flag', 'card_id', 'city_id', 'category_1',
       'installments', 'category_3', 'merchant_category_id', 'merchant_id',
       'month_lag', 'purchase_amount', 'purchase_date', 'category_2',
       'state_id', 'subsector_id', 'active_months_lag12', 'active_months_lag
3',
       'active_months_lag6', 'avg_purchases_lag12', 'avg_purchases_lag3',
       'avg_purchases_lag6', 'avg_sales_lag12', 'avg_sales_lag3',
       'avg_sales_lag6', 'category_4', 'merchant_group_id',
       'most_recent_purchases_range', 'most_recent_sales_range', 'numerical_
1',
       'numerical_2'],
      dtype='object')
```

In [8]:
```
#let's drop the Unnamed coulmns and merge it with the train dataset where card
features are present
data.drop(data.filter(regex="Unname"),axis=1, inplace=True)
data=data.merge(train_csv,how='outer',on='card_id')
data.columns
```

Out[8]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'category_3', 'merchant_category_id', 'merchant_id', 'month_lag',
       'purchase_amount', 'purchase_date', 'category_2', 'state_id',
       'subsector_id', 'active_months_lag12', 'active_months_lag3',
       'active_months_lag6', 'avg_purchases_lag12', 'avg_purchases_lag3',
       'avg_purchases_lag6', 'avg_sales_lag12', 'avg_sales_lag3',
       'avg_sales_lag6', 'category_4', 'merchant_group_id',
       'most_recent_purchases_range', 'most_recent_sales_range', 'numerical_
1',
       'numerical_2', 'first_active_month', 'feature_1', 'feature_2',
       'feature_3', 'target'],
      dtype='object')
```

Plot month_lag with target features

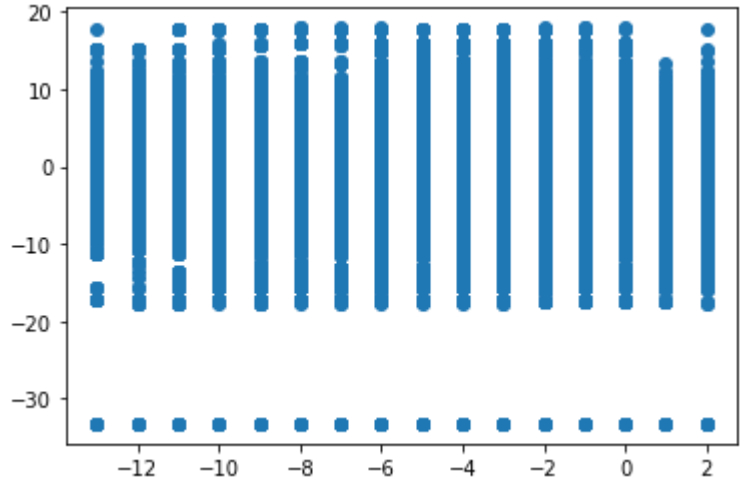
In [ ]:
```
ax=sns.boxplot(x="month_lag",y="target",data=data)
```



In [ ]:
```
```

In [ ]:
```
plt.scatter(data['month_lag'],data['target'])
```

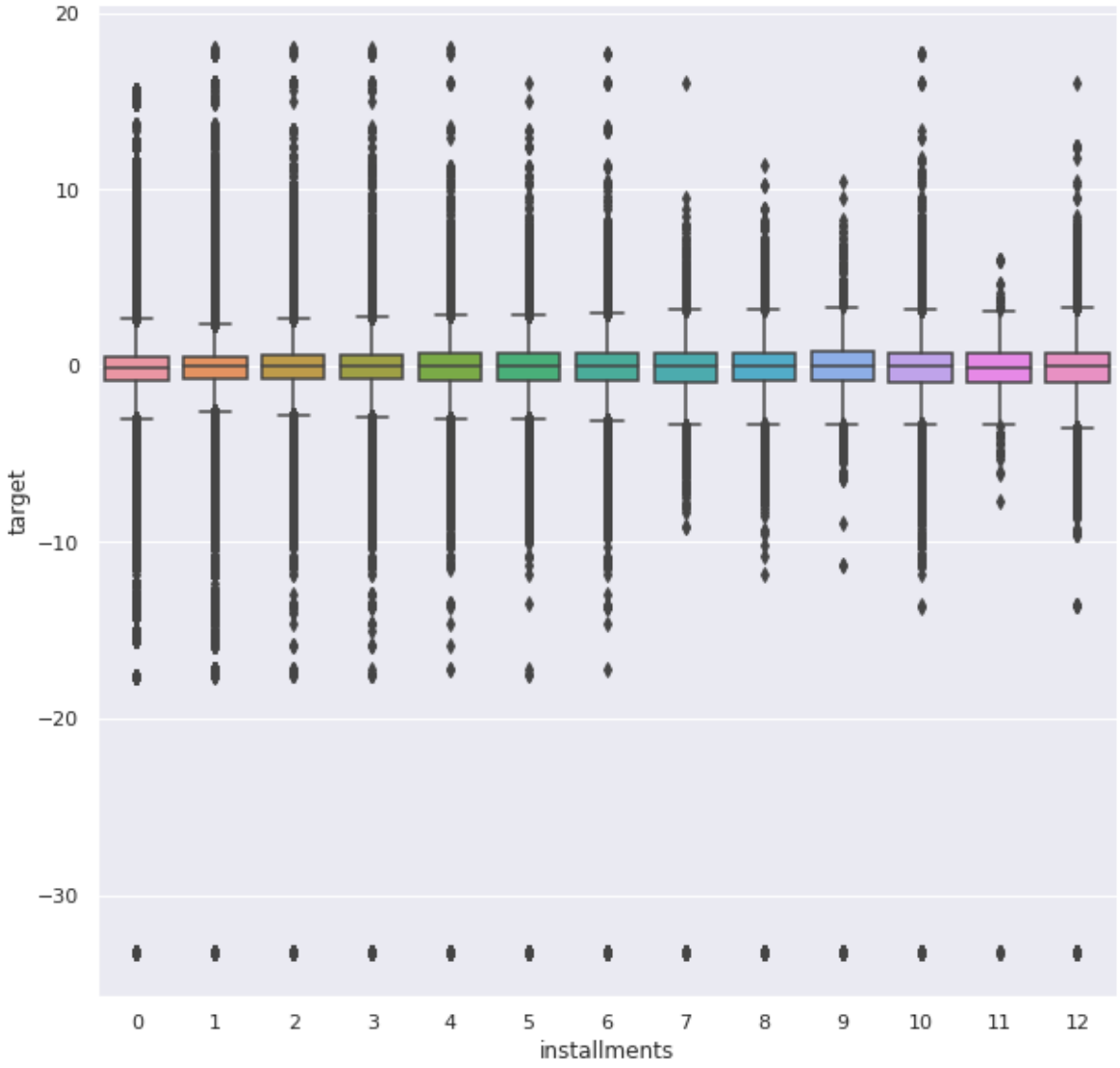
Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7feed14b6a20>
```



So , overlapping happens as shown in the above box plot.

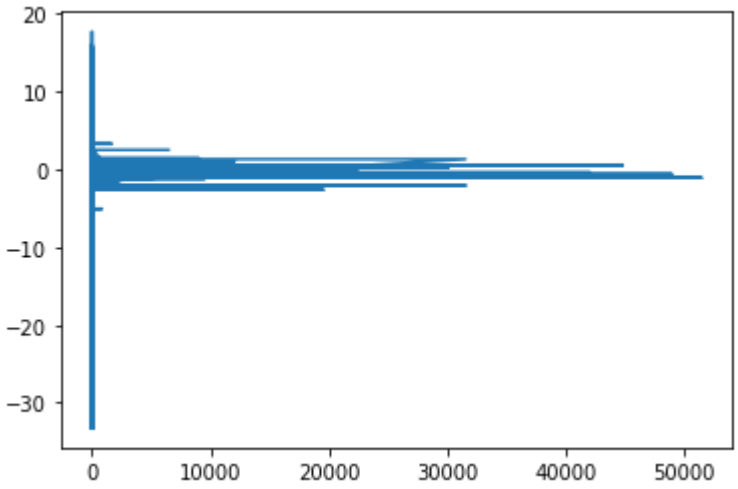
Plot Installments vs target

In [ ]:
```
sns.set(rc={'figure.figsize':(10,10)})
ax=sns.boxplot(x="installments",y="target",data=data)
```



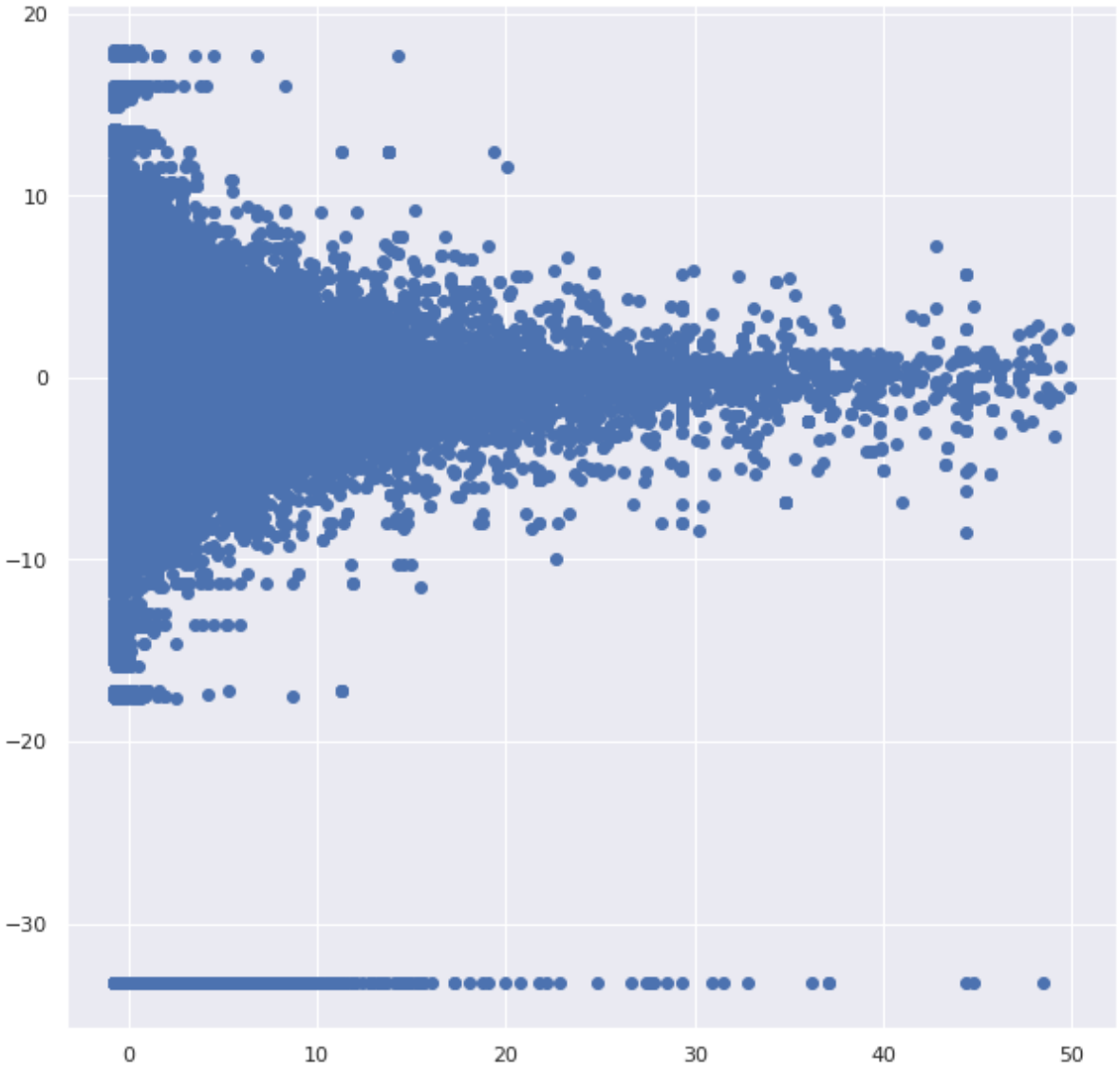
Plot Purchase_amount vs target

In [ ]:
```
plt.plot(data['purchase_amount'],data['target'])
plt.show()
```

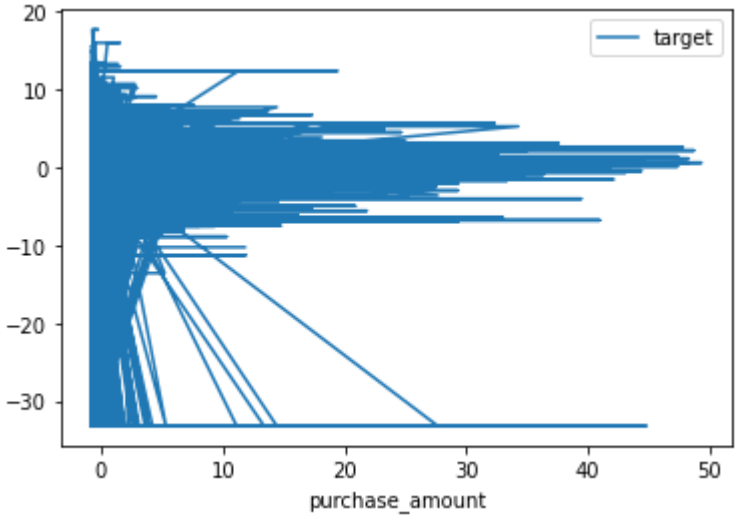


In [ ]:
```
plt.scatter(data['purchase_amount'],data['target'])
```

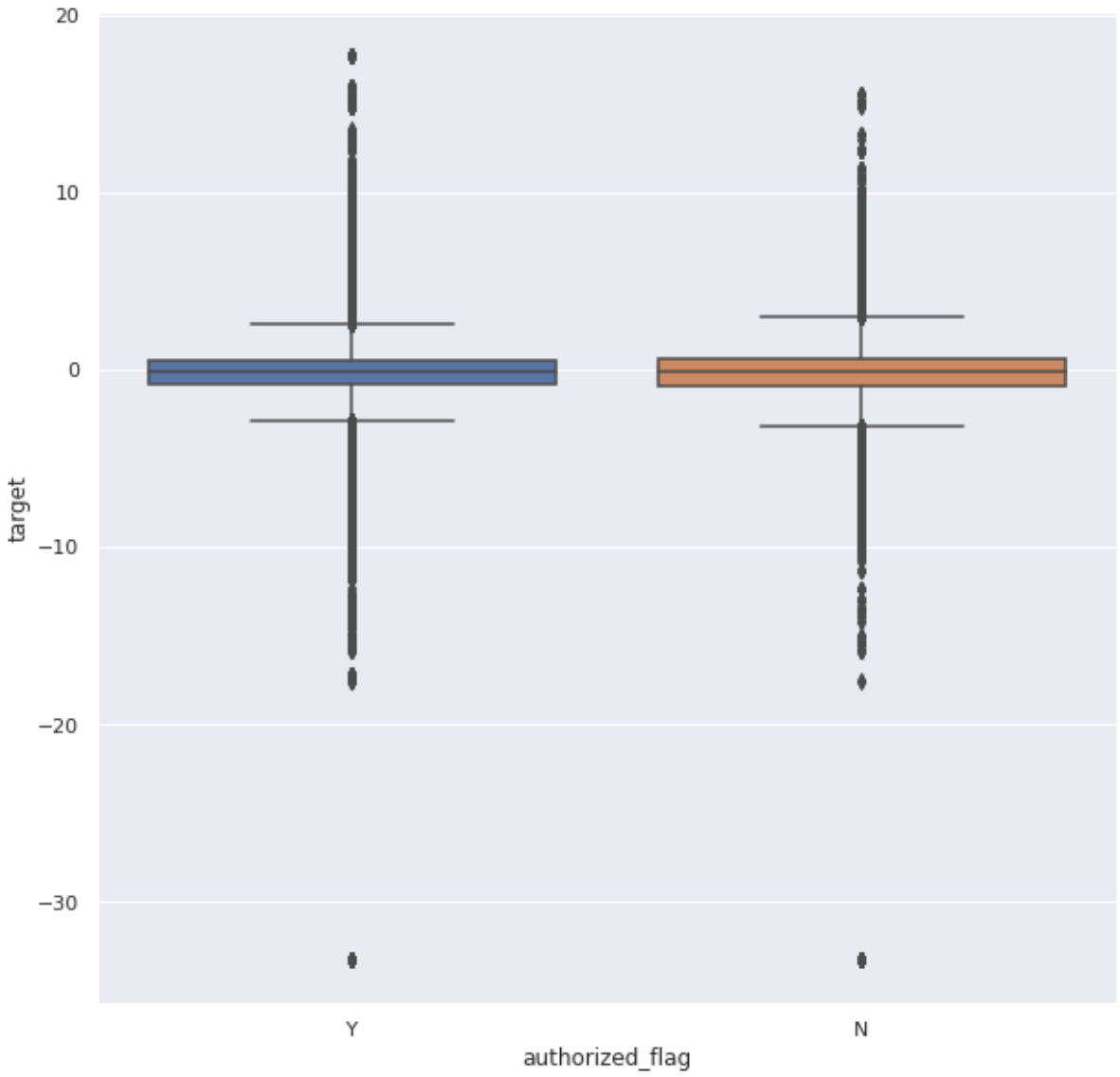
Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f49e0909630>
```



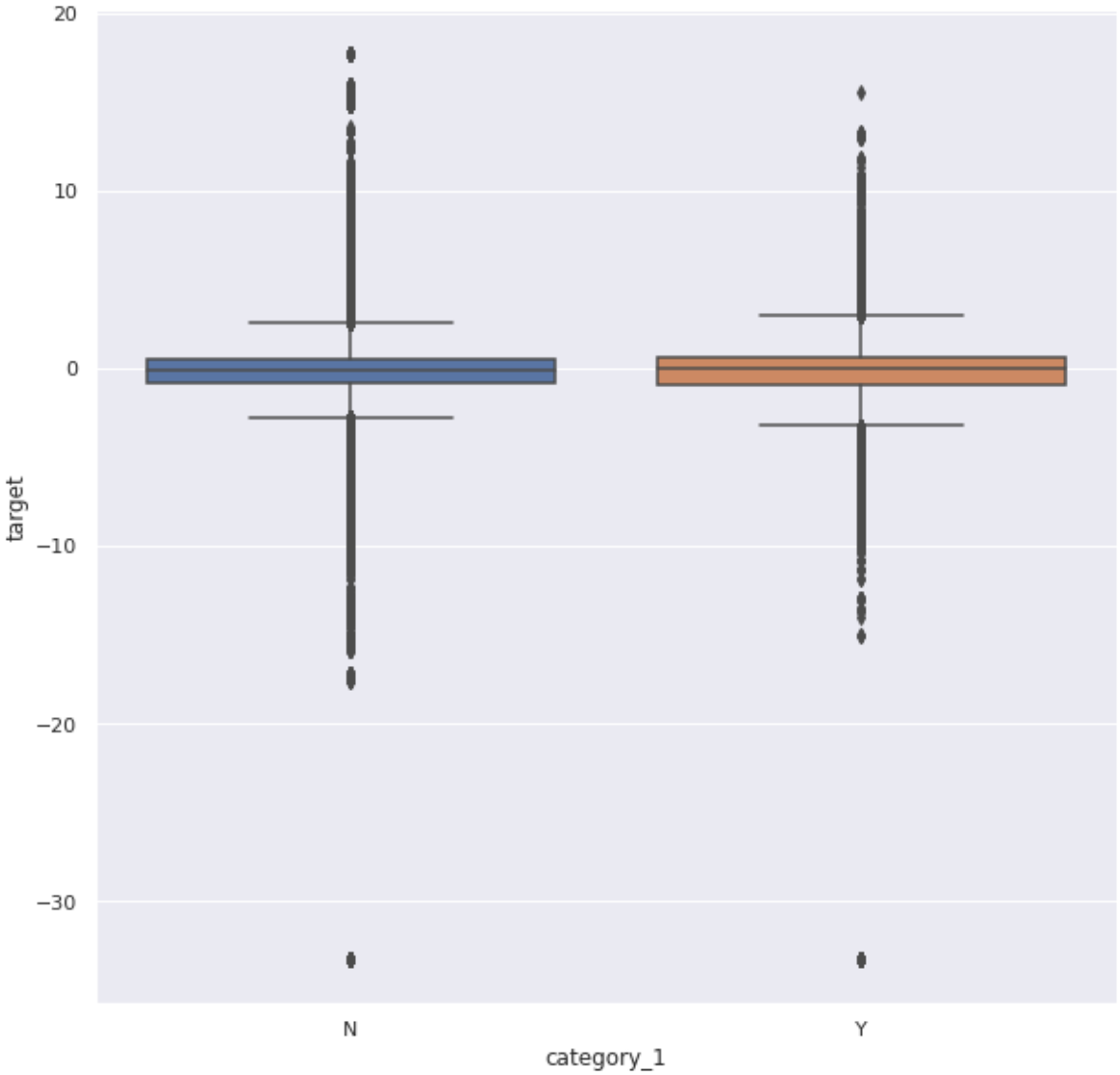
In [ ]:
```
ax = plt.gca()

data.plot(kind='line',x='purchase_amount',y='target',ax=ax)
#df.plot(kind='line',x='name',y='num_pets', color='red', ax=ax)

plt.show()
```



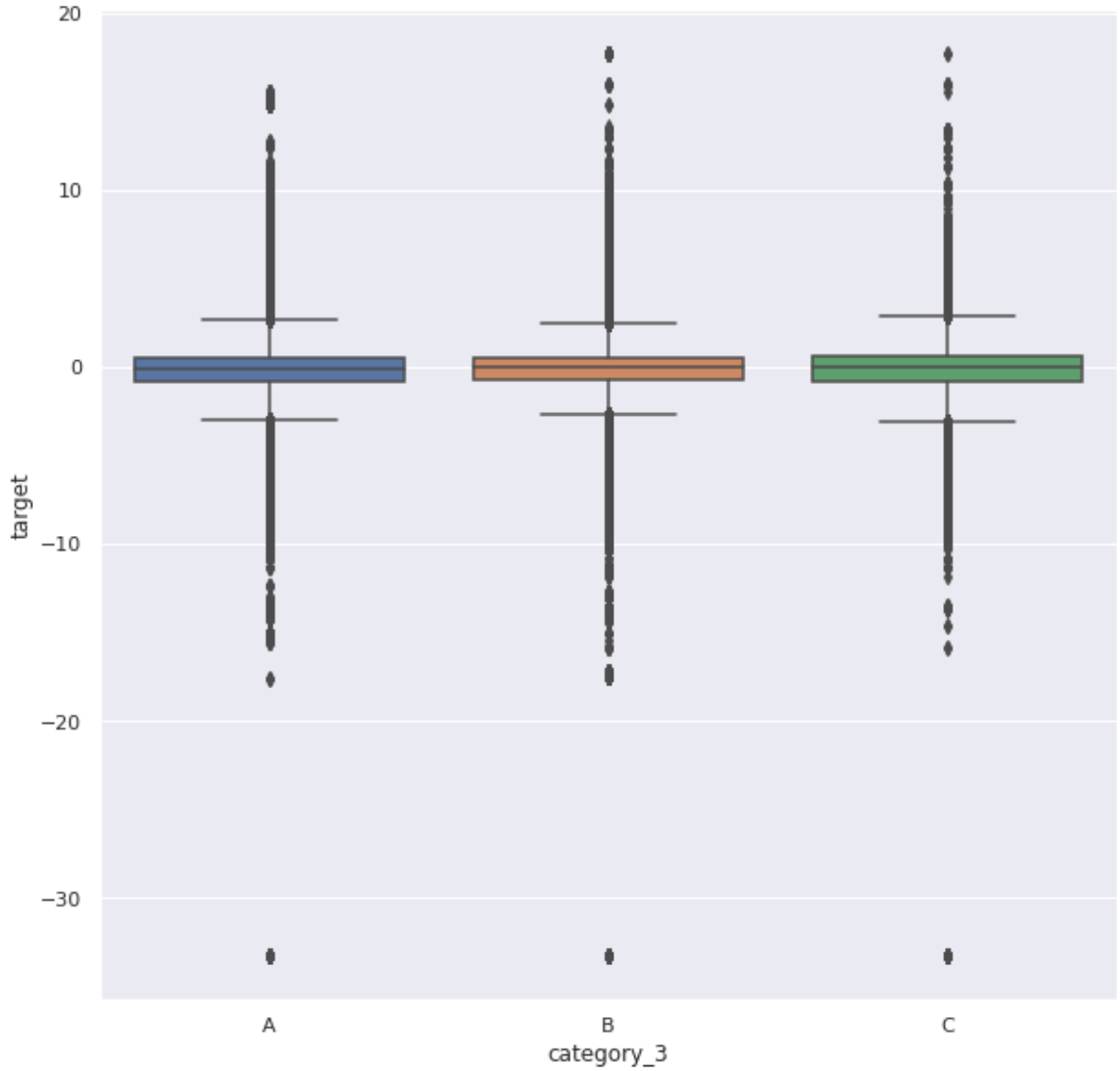
In [ ]:
```
sns.set(rc={'figure.figsize':(10,10)})
ax=sns.boxplot(x="authorized_flag",y="target",data=data)
```



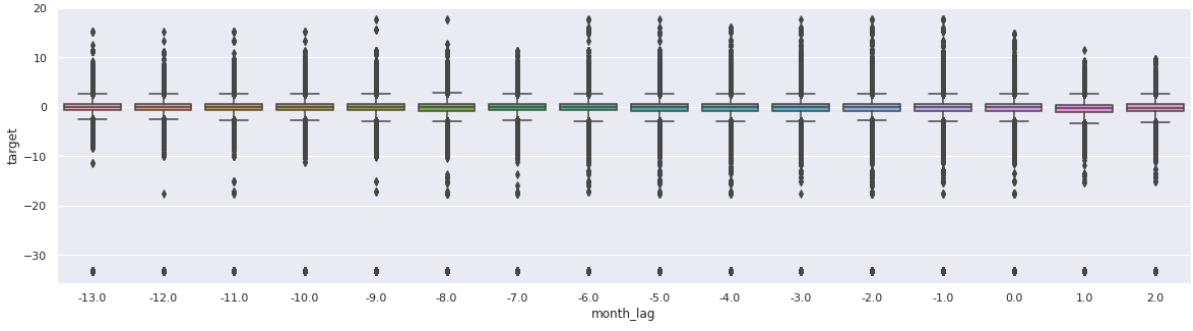
In [ ]:
```
ax=sns.boxplot(x="category_1",y="target",data=data)
```



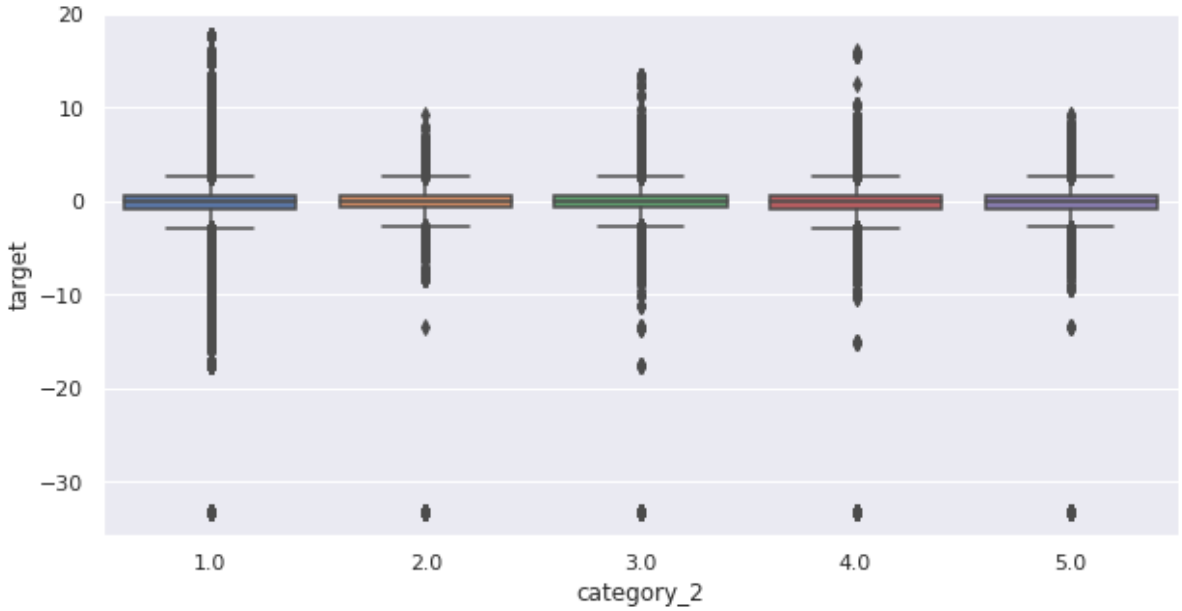
In [ ]:
```
ax=sns.boxplot(x="category_3",y="target",data=data)
```



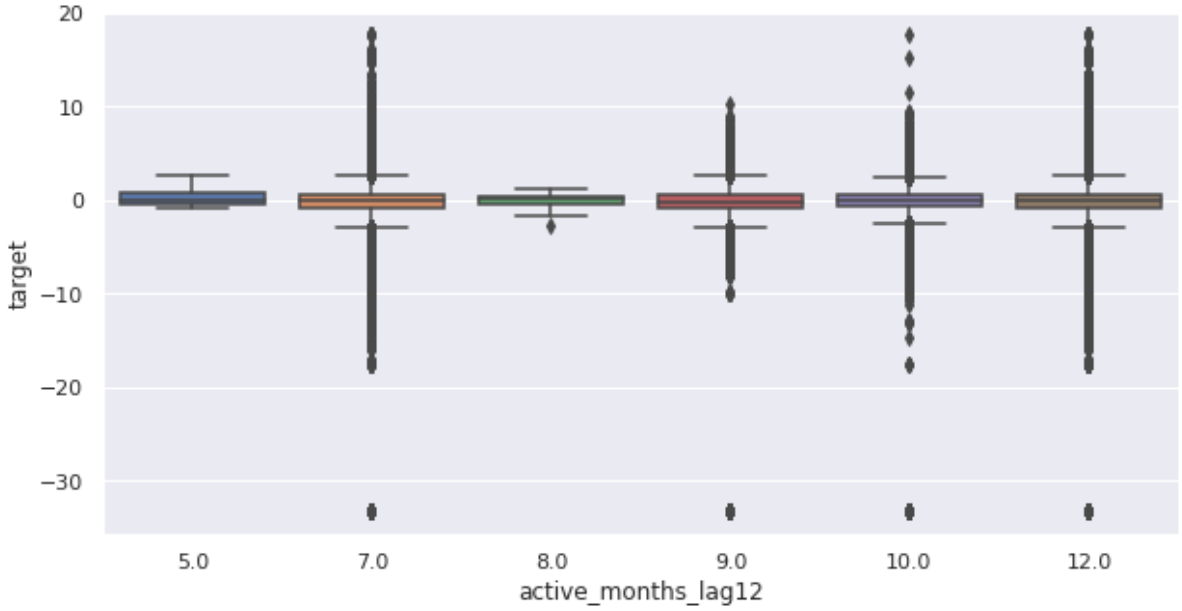
In [ ]:
```
sns.set(rc={'figure.figsize':(20,5)})
ax=sns.boxplot(x="month_lag",y="target",data=data)
```



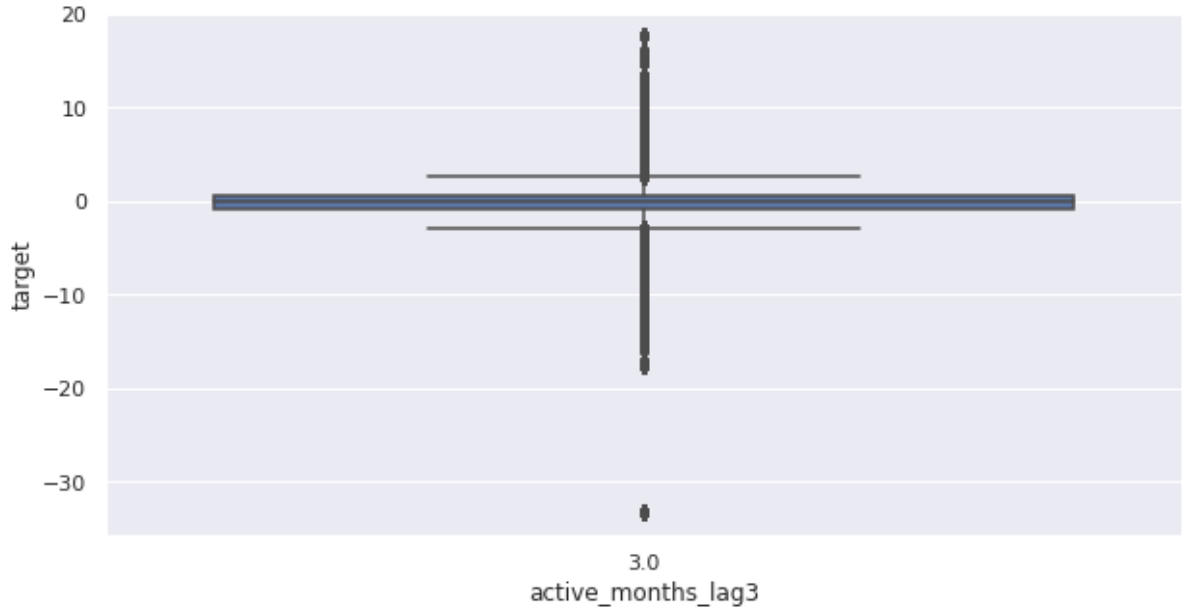
In [ ]:
```
sns.set(rc={'figure.figsize':(10,5)})
ax=sns.boxplot(x="category_2",y="target",data=data)
```



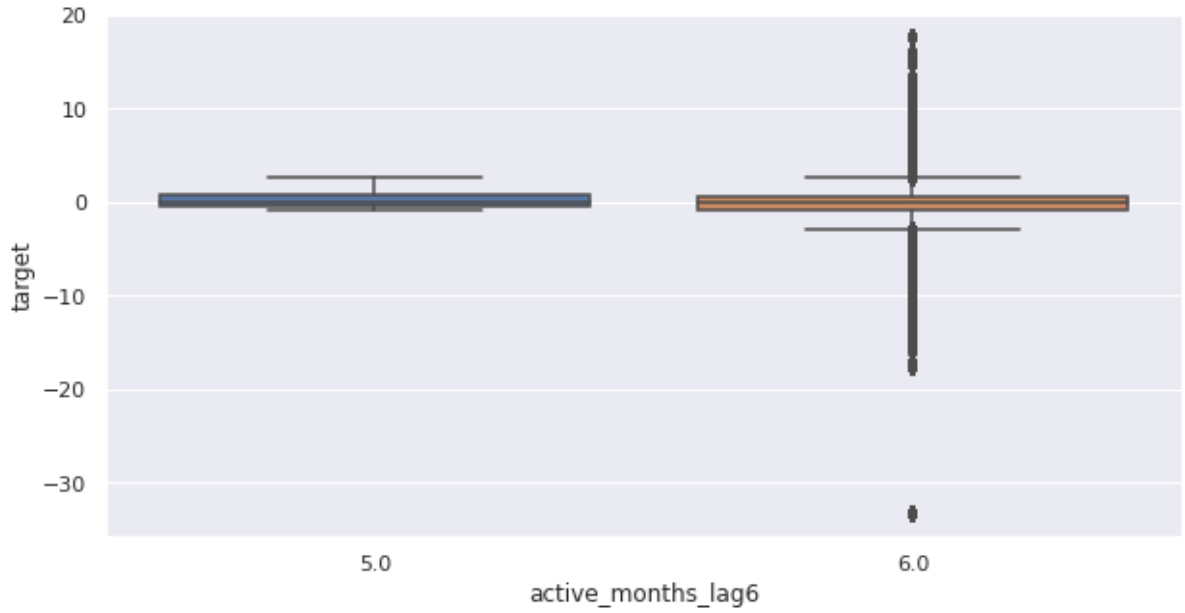
In [ ]:
```
ax=sns.boxplot(x="active_months_lag12",y="target",data=data)
```



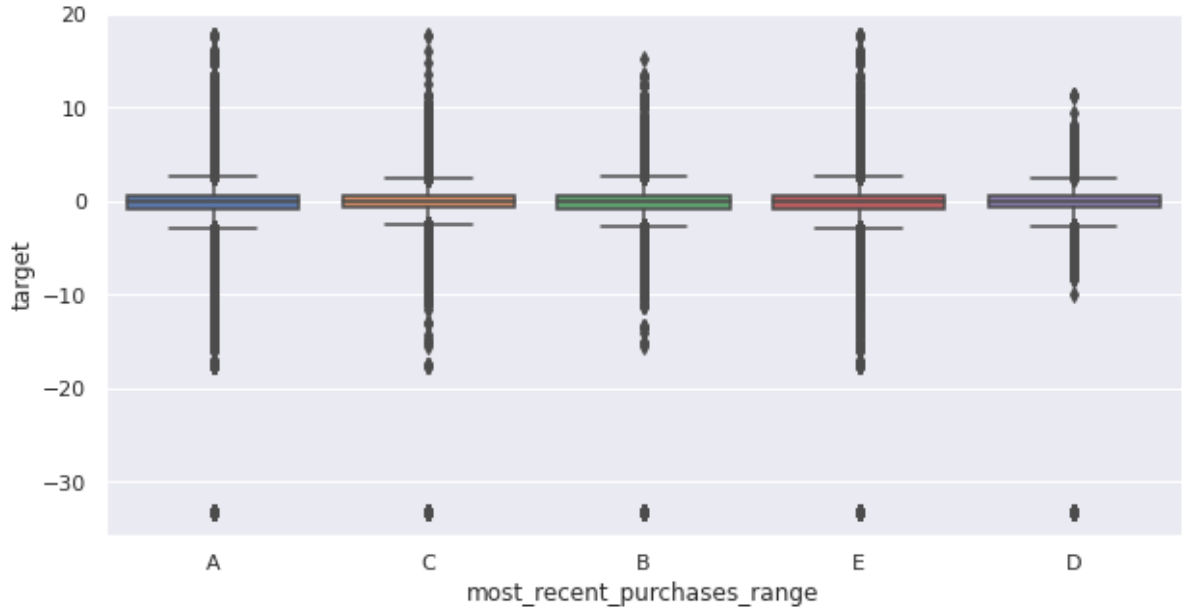
In [ ]:
```
ax=sns.boxplot(x="active_months_lag3",y="target",data=data)
```



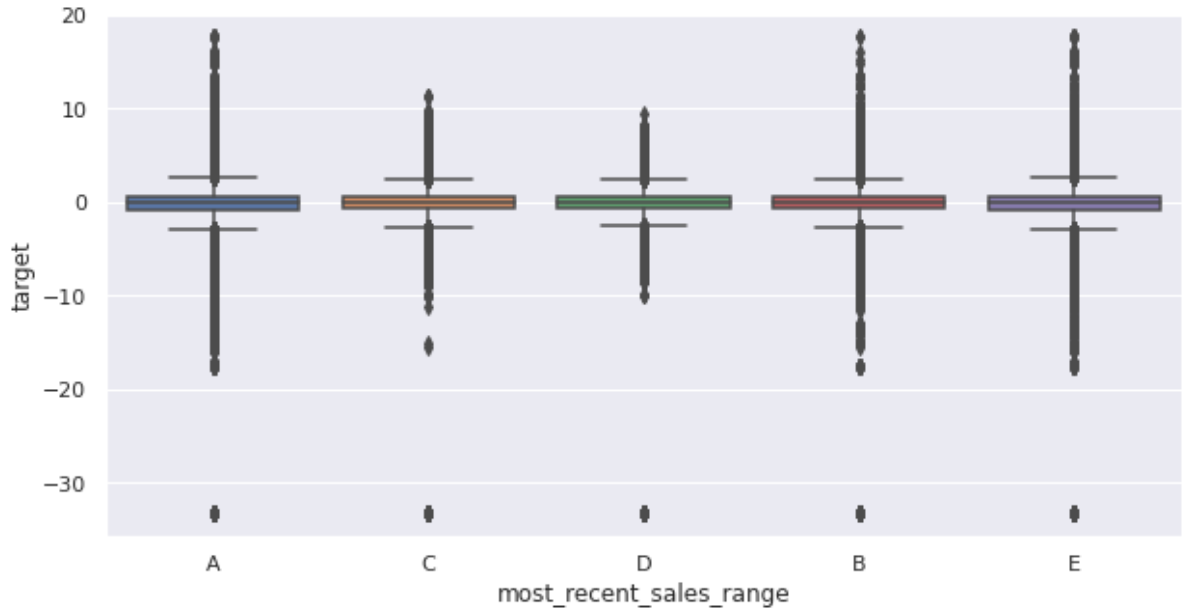
In [ ]:
```
ax=sns.boxplot(x="active_months_lag6",y="target",data=data)
```



In [ ]:
```
ax=sns.boxplot(x="most_recent_purchases_range",y="target",data=data)
```

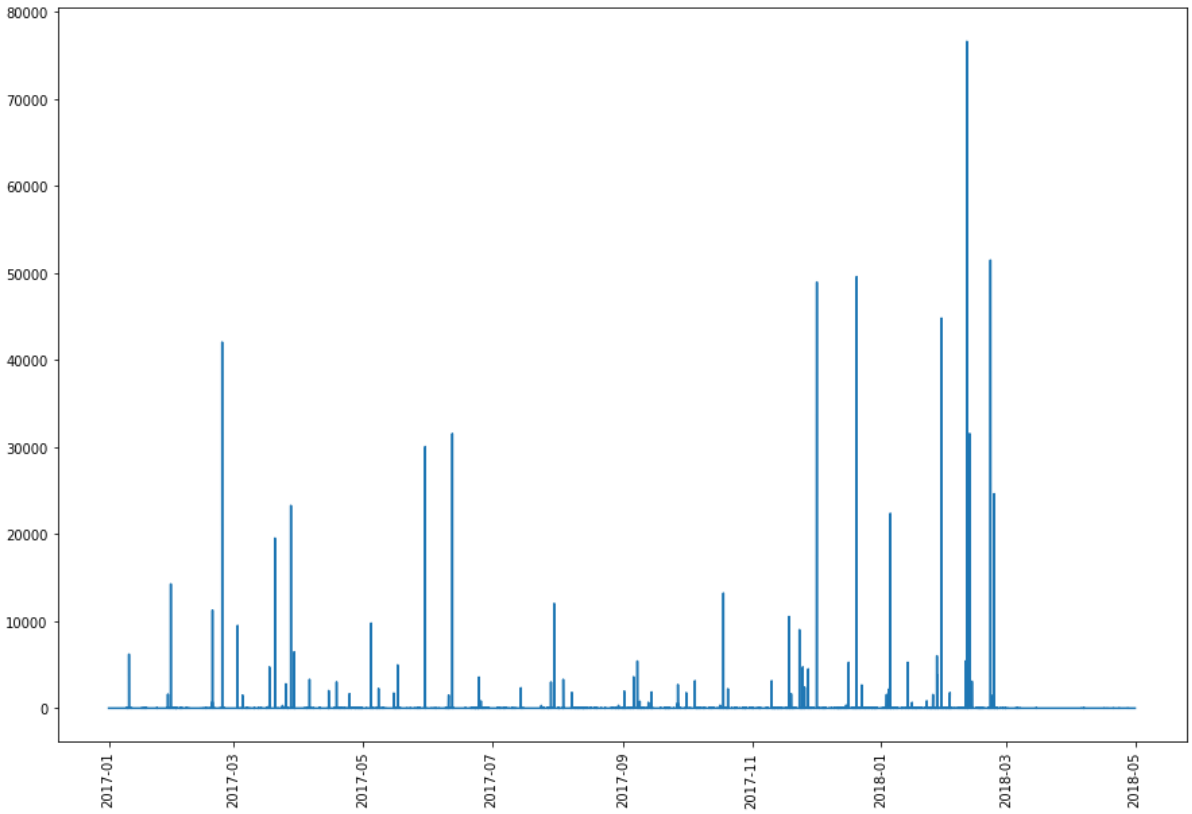


In [ ]:
```
ax=sns.boxplot(x="most_recent_sales_range",y="target",data=data)
```

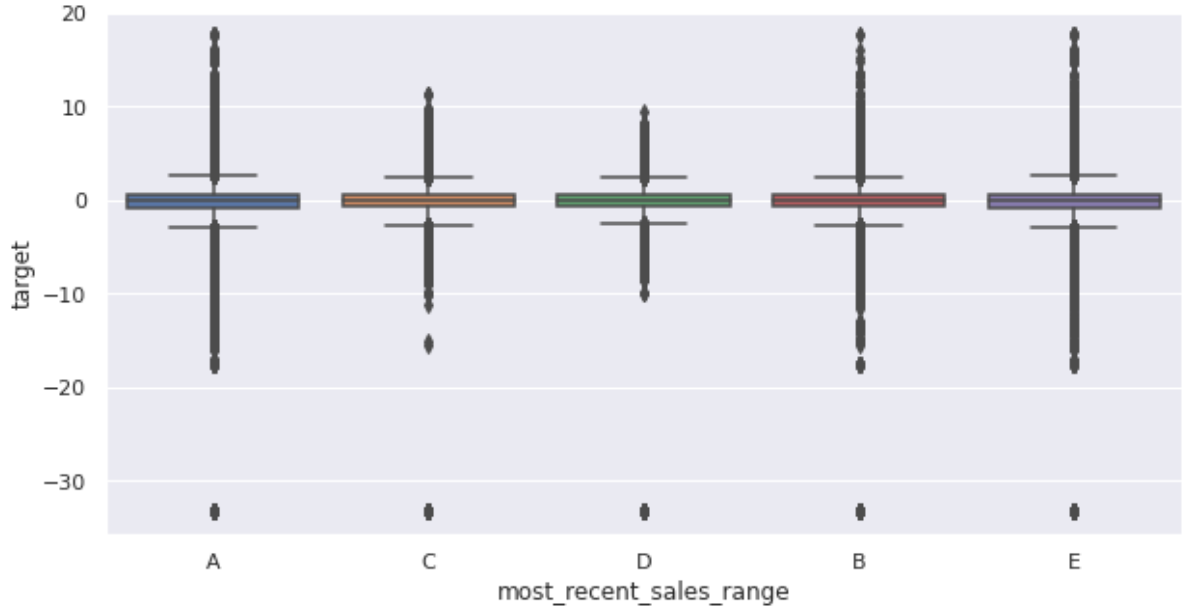


In [10]:
```
data['purchase_date']=pd.to_datetime(data['purchase_date'],format='%Y-%m')
data['first_active_month']=pd.to_datetime(data['first_active_month'],format='%
Y-%m')
```

In [11]:
```
fig, ax = plt.subplots(figsize=(15, 10))
data = data.sort_values('purchase_date', ascending=True)
plt.plot(data['purchase_date'], data['purchase_amount'])
plt.xticks(rotation='vertical')
plt.show()
```



In [ ]:
```
ax=sns.boxplot(x="most_recent_sales_range",y="target",data=data)
```



As we have selected only sample of the actual data , so many of the records are not present . However , it is clear even in Bi-variate analysis no single feture is sufficient to predict the target score. hence, based on the EDA result at first we need to clean the data and then go for feature engineering which can be useful for model creation.

Conclusion:

As the basic EDA is completed and also we get to know about the insight of the data , in the next step we will need to take action based on our observations.

Then we can proceed with the feature engineering steps

In [1]:
```
#print("end")
```

Construct the problem as Classification Problem

As we checked, our primary task is to predict the loyalty score which can be real values. However, instead of thinking the problem statement like that, we can also think to predict the customer whether he is loyal or not . If the loyalty score is 0.0 then he/she should be consider as "Not loyal" and if it's 1.0 then can be considered as "Loyal".

Clearly , if the customer can be considered as "Loyal", then he/she would get maximum offers/discount coupons

Accuracy Metrics

For Binary Classification problems, we can choose roc_auc score. Also True Positive rate is most important

Steps to reconstruct the Regression problem to Binary Classification Problem

We will perform StandardScaler() or standardizee the target feauture. after Standardiing if it's <=0 then to be set as 0 else we will set the target feature as 1

In [28]:
```
target=data['target']
type(target)
```

Out[28]:
```
pandas.core.series.Series
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
#target_std=scaler.fit_transform(target)

data[['target_std']] = scaler.fit_transform(data[['target']])
```

In [32]:
```
data.head()
```

Out[32]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_cat
0	Y	C_ID_f43ae18887	69.0	N	0.0	A	
1	Y	C_ID_f43ae18887	69.0	N	0.0	A	
2	Y	C_ID_90b85e2e11	-1.0	Y	3.0	C	
3	Y	C_ID_3933ea1cb4	331.0	N	0.0	A	
4	N	C_ID_9daf3daa55	-1.0	Y	3.0	C	

In [46]:
```
target_0=data['target_std']<=0
target_1=data['target_std']>0
data.loc[target_0,'target_std']=0
data.loc[target_1,'target_std']=1
data.head()
```

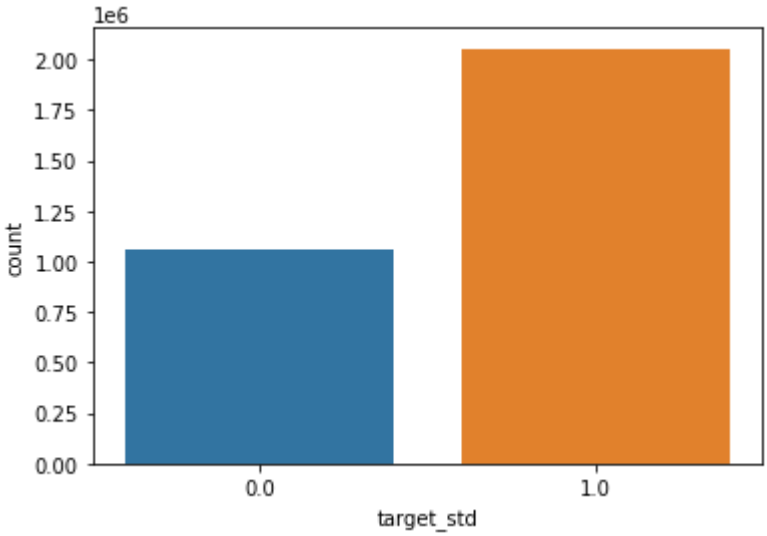
Out[46]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_cat
0	Y	C_ID_f43ae18887	69.0	N	0.0	A	
1	Y	C_ID_f43ae18887	69.0	N	0.0	A	
2	Y	C_ID_90b85e2e11	-1.0	Y	3.0	C	
3	Y	C_ID_3933ea1cb4	331.0	N	0.0	A	
4	N	C_ID_9daf3daa55	-1.0	Y	3.0	C	

As we could see, the new column target_std has been introduced which can hold [0 or 1]. So we will consider the same problem as Regression as well as Binary Classification problem and simulatneously evaluate the accuracy for both of them.

Distribution of Target feature after performing standard scaler

In [17]:
```
ax=sns.countplot(x='target_std',data=data)
```



Even though we have loaded only few percentage of the entire data , still we can conclude now in the target feature only two values present either 0 or 1 and this can be considered as a binary classification problem

Now we will plot against the new target feature and check if it gives us any additional informations

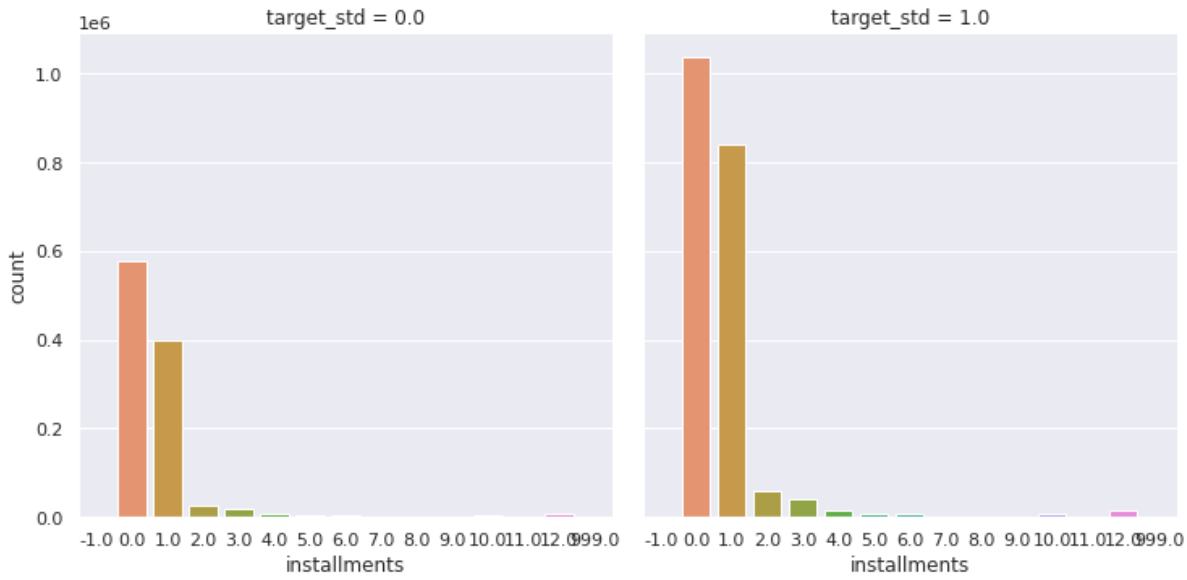
In [36]:
```
#ref: https://www.tutorialspoint.com/seaborn/seaborn_multi_panel_categorical_plots.htm
sns.set(rc={'figure.figsize':(5,5)})
sns.factorplot("month_lag", col = "target_std", col_wrap = 3,data = data[data.month_lag.notnull()],kind = "count")
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

In [41]:
```
sns.factorplot("installments", col = "target_std", col_wrap = 3,data = data[da
ta.target_std.notnull()],kind = "count")
```

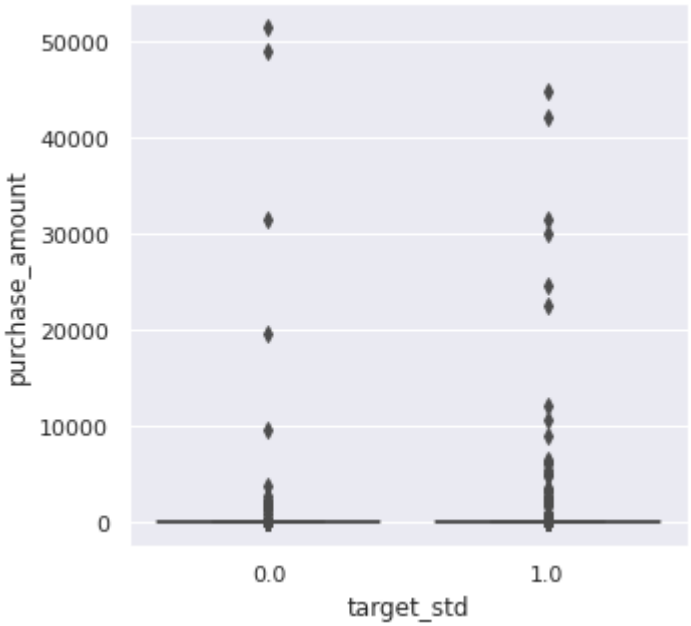
```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[41]: <seaborn.axisgrid.FacetGrid at 0x7fe482998c18>



In [39]:
```
sns.boxplot(x='target_std',y='purchase_amount',data=data)
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe4693a2e10>
```

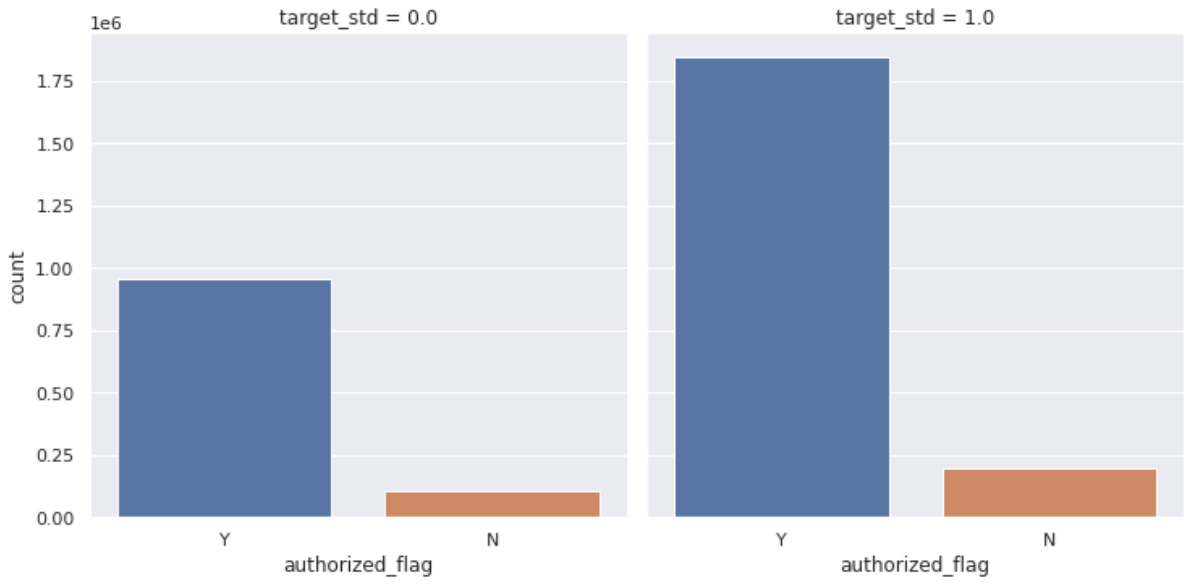


In [ ]:
```

```

In [40]:
```
sns.factorplot("authorized_flag", col = "target_std", col_wrap = 3,data = data
[data.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

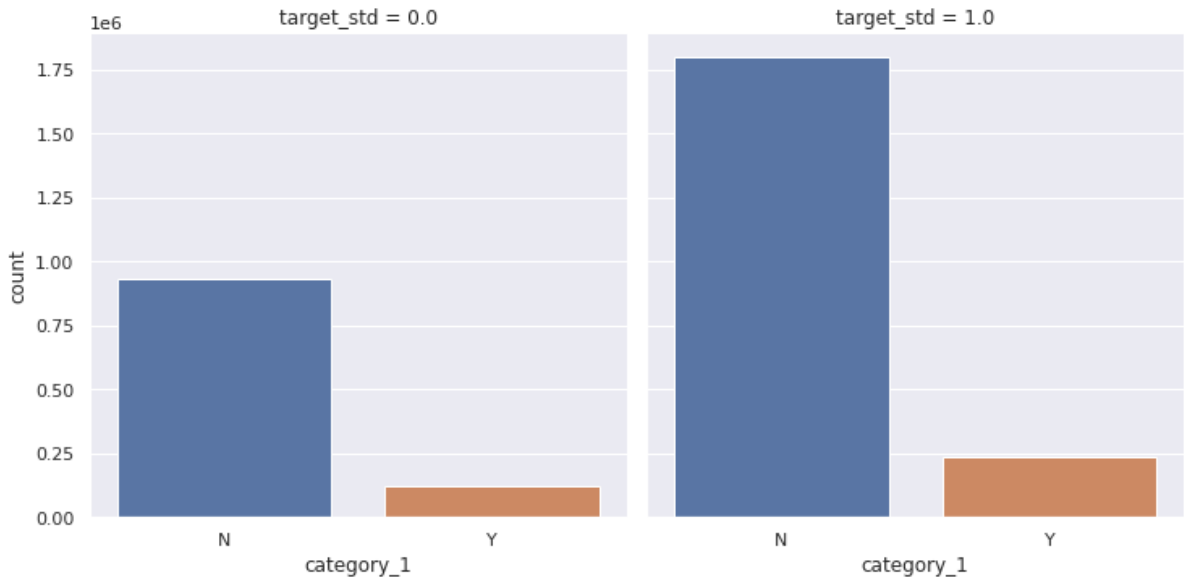
Out[40]: <seaborn.axisgrid.FacetGrid at 0x7fe487a0bf98>



In [42]:
```
sns.factorplot("category_1", col = "target_std", col_wrap = 3,data = data[data
.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

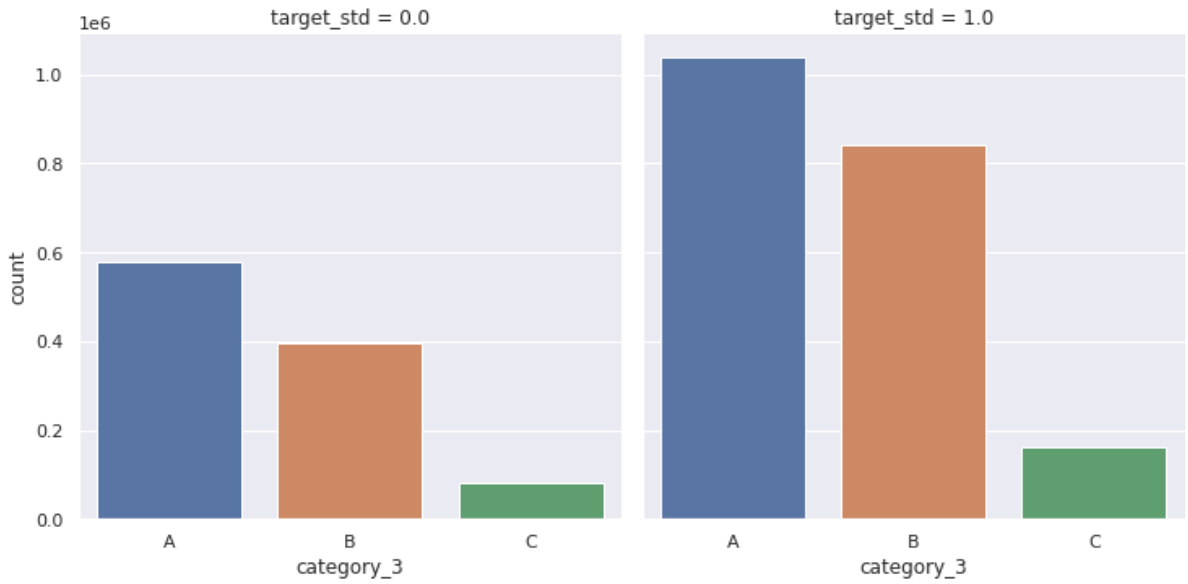
Out[42]: <seaborn.axisgrid.FacetGrid at 0x7fe482c65588>



In [43]:
```
sns.factorplot("category_3", col = "target_std", col_wrap = 3,data = data[data
.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[43]: <seaborn.axisgrid.FacetGrid at 0x7fe4879dfa20>



In [44]:
```
sns.factorplot("category_2", col = "target_std", col_wrap = 3,data = data[data
.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[44]: <seaborn.axisgrid.FacetGrid at 0x7fe47d705748>

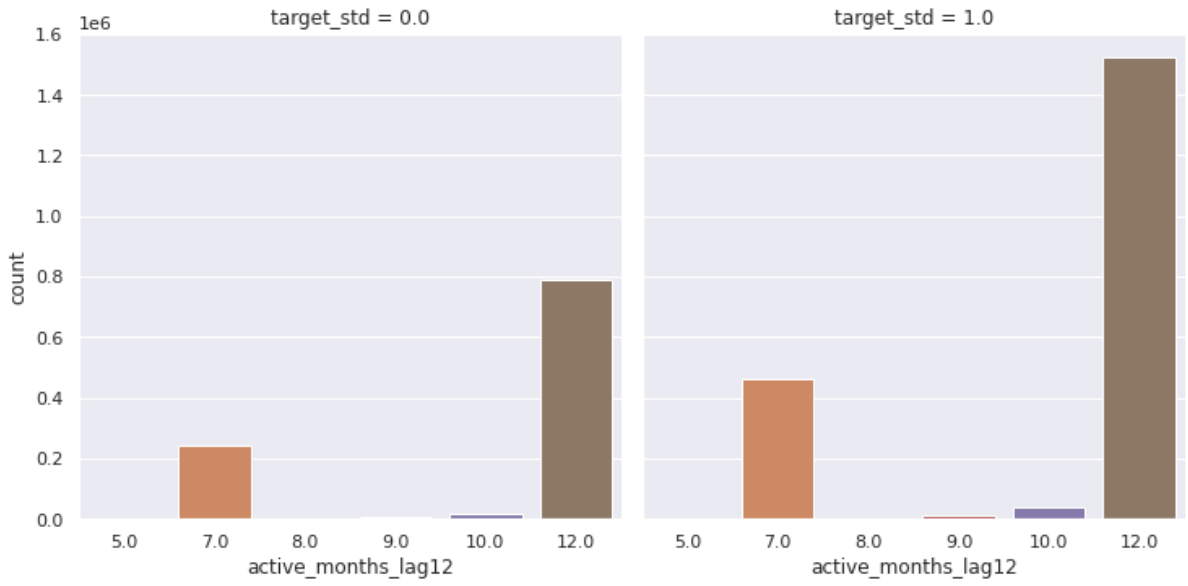
In [46]:
```
data['active_months_lag12'].nunique()
```

Out[46]: 6

In [47]:
```
sns.factorplot("active_months_lag12", col = "target_std", col_wrap = 3,data = data[data.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

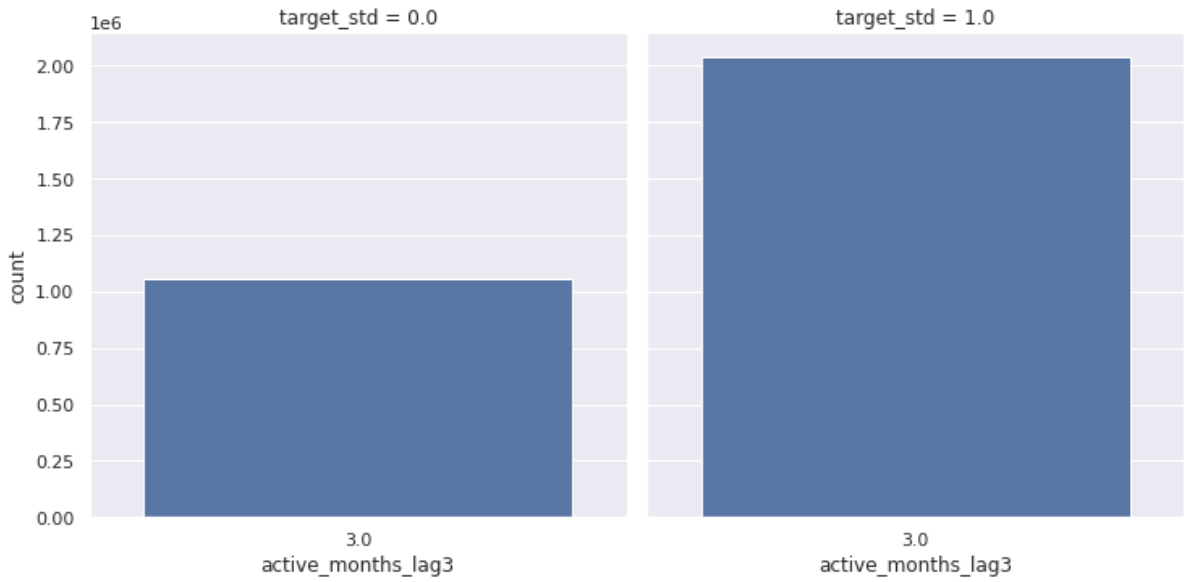
Out[47]:
```
<seaborn.axisgrid.FacetGrid at 0x7fe464ace438>
```



In [48]:
```
sns.factorplot("active_months_lag3", col = "target_std", col_wrap = 3,data = d
ata[data.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

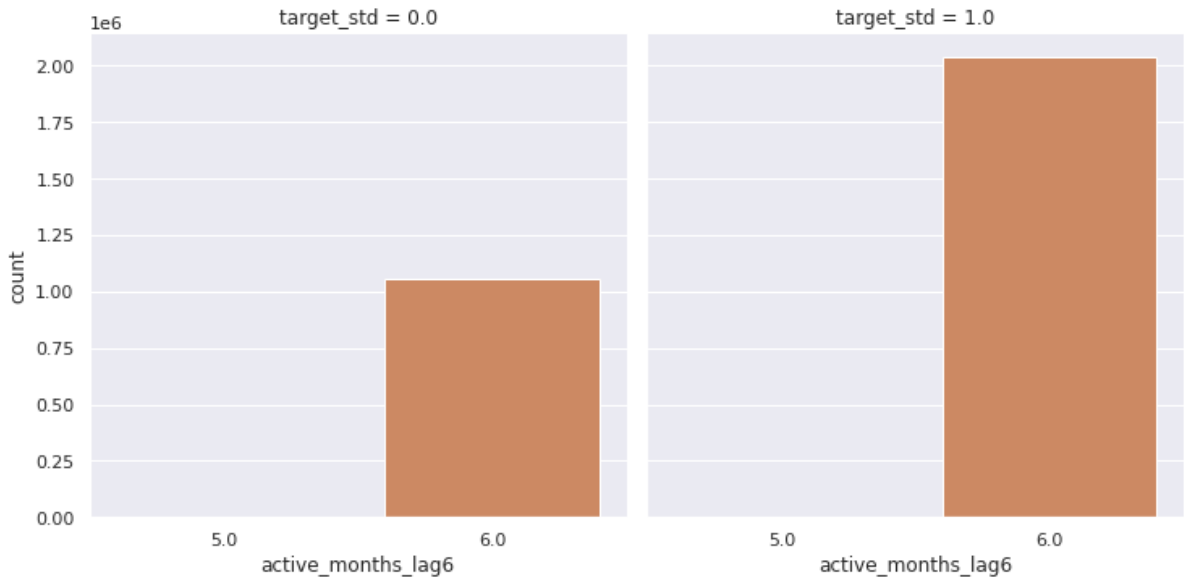
Out[48]: <seaborn.axisgrid.FacetGrid at 0x7fe47b9f6160>



In [49]:
```
sns.factorplot("active_months_lag6", col = "target_std", col_wrap = 3,data = d
ata[data.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

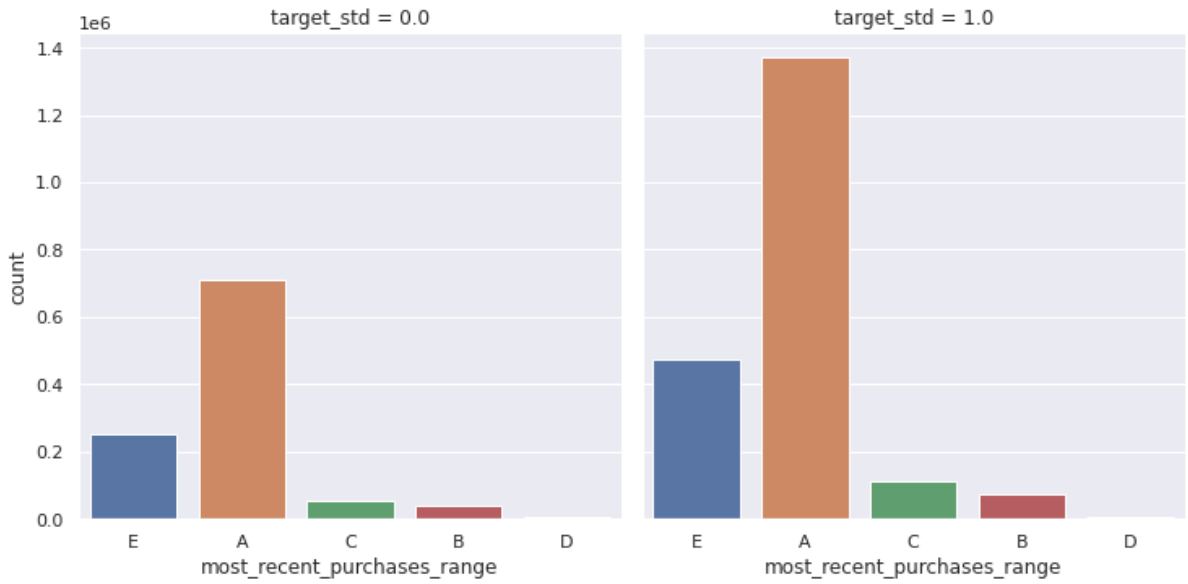
Out[49]: <seaborn.axisgrid.FacetGrid at 0x7fe481921390>



In [50]:
```
sns.factorplot("most_recent_purchases_range", col = "target_std", col_wrap = 3
,data = data[data.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

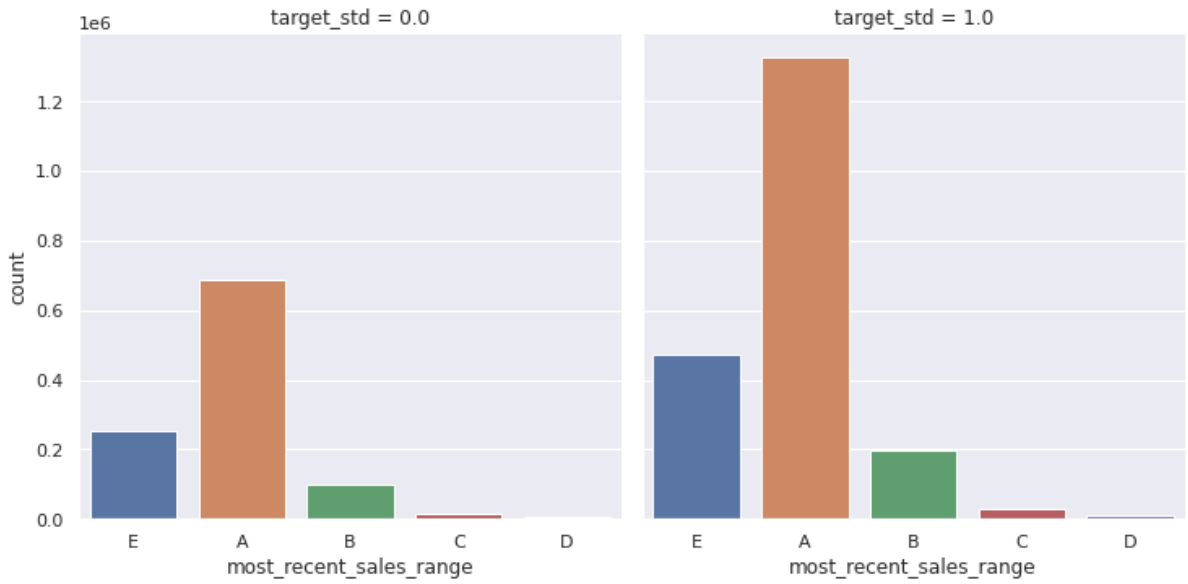
Out[50]: <seaborn.axisgrid.FacetGrid at 0x7fe481e2f6d8>



In [51]:
```
sns.factorplot("most_recent_sales_range", col = "target_std", col_wrap = 3,dat
a = data[data.target_std.notnull()],kind = "count")
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/categorical.py:3704: UserWarni
ng: The `factorplot` function has been renamed to `catplot`. The original nam
e will be removed in a future release. Please update your code. Note that the
default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot
`.
  warnings.warn(msg)
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarni
ng: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[51]: <seaborn.axisgrid.FacetGrid at 0x7fe482cba128>



The dataset which we have loaded has 5M rows which is 1/4 th of the original given dataset but this is clear , there is no direct relationship in between target and remaining features ie no single/bi-features are not sufficient to predict the target feature as the dataset is quiet complex. So we have to perform the feature engineering in next step and remove the features which are not so useful.

Then only we can build the model and check for the accuracy.

Feature Engineering

As the basic EDA is completed, now based on our observation we will perform the feature engineering steps which to be fed to model

In [ ]:
```
#load the train dataset for card details
train_csv=pd.read_csv('/content/train.csv')
```

In [ ]:
```
#load historical transactions
historical_transactions=pd.read_csv('/content/historical_transactions.csv')
historical_transactions.columns
```

Out[ ]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'category_3', 'merchant_category_id', 'merchant_id', 'month_lag',
       'purchase_amount', 'purchase_date', 'category_2', 'state_id',
       'subsector_id'],
      dtype='object')
```

In [ ]:
```
# load the new merchants transactions
new_merchant_transactions=pd.read_csv('/content/new_merchant_transactions.csv'
)
new_merchant_transactions.columns
```

Out[ ]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'category_3', 'merchant_category_id', 'merchant_id', 'month_lag',
       'purchase_amount', 'purchase_date', 'category_2', 'state_id',
       'subsector_id'],
      dtype='object')
```

In [ ]:
```
#load the merchant dataset
merchants=pd.read_csv('/content/merchants.csv')
merchants.columns
```

Out[ ]:
```
Index(['merchant_id', 'merchant_group_id', 'merchant_category_id',
       'subsector_id', 'numerical_1', 'numerical_2', 'category_1',
       'most_recent_sales_range', 'most_recent_purchases_range',
       'avg_sales_lag3', 'avg_purchases_lag3', 'active_months_lag3',
       'avg_sales_lag6', 'avg_purchases_lag6', 'active_months_lag6',
       'avg_sales_lag12', 'avg_purchases_lag12', 'active_months_lag12',
       'category_4', 'city_id', 'state_id', 'category_2'],
      dtype='object')
```

As we checked earlier , we will first remove the duplicate records and replace the NaN values

In [ ]:
```
#remove the duplicate merchant_id
merchants.drop_duplicates(subset ="merchant_id", keep = "first", inplace = Tru
e)
merchants.shape
```

Out[ ]:
```
(334633, 22)
```

In [ ]:
```
merchants['avg_sales_lag3'].median()
```

Out[ ]:
```
1.0
```

In [ ]:
```
merchants['avg_sales_lag6'].median()
```

Out[ ]:
```
1.01
```

In [ ]:
```
merchants['avg_sales_lag12'].median()
```

Out[ ]:
```
1.02
```

In [ ]:
```
merchants['category_2'].mode()
```

Out[ ]:
```
0    1.0
dtype: float64
```

In [ ]:
```
print("avg_purchases_lag3 median:",merchants['avg_purchases_lag3'].median())
print("avg_purchases_lag6 median:",merchants['avg_purchases_lag6'].median())
print("avg_purchases_lag12 median:",merchants['avg_purchases_lag12'].median())
```

```
avg_purchases_lag3 median: 1.01666667
avg_purchases_lag6 median: 1.02693603
avg_purchases_lag12 median: 1.04334433
```

In [ ]:
```
# convert the inf & NaN values
merchants.replace([np.inf, -np.inf], np.nan,inplace=True)
merchants['category_2'].fillna(1.0,inplace=True)
merchants['avg_sales_lag3'].fillna(1.0,inplace=True)
merchants['avg_sales_lag6'].fillna(1.01,inplace=True)
merchants['avg_sales_lag12'].fillna(1.02,inplace=True)
merchants['avg_purchases_lag3'].fillna(1.02,inplace=True)
merchants['avg_purchases_lag6'].fillna(1.03,inplace=True)
merchants['avg_purchases_lag12'].fillna(1.04,inplace=True)
merchants.shape
```

Out[ ]:
```
(334633, 22)
```

Since we have many common features in between transaction and merchant dataset , so we will select only those features which are not present in transaction dataset

In [ ]:
```
historical_transactions=historical_transactions.merge(merchants,how='outer',on
='merchant_id')
historical_transactions.columns

new_merchant_transactions=new_merchant_transactions.merge(merchants,how='oute
r',on='merchant_id')
new_merchant_transactions.columnsmerchants=merchants[['merchant_id','active_mo
nths_lag12','active_months_lag3',
 'active_months_lag6',
 'avg_purchases_lag12',
 'avg_purchases_lag3',
 'avg_purchases_lag6',
 'avg_sales_lag12',
 'avg_sales_lag3',
 'avg_sales_lag6',
 'category_4',
 'merchant_group_id',
 'most_recent_purchases_range',
 'most_recent_sales_range',
 'numerical_1']]
```

In [ ]:
```
historical_transactions['category_3'].mode()
```

Out[ ]:
```
0    A
dtype: object
```

In [ ]:
```
historical_transactions['category_2'].mode()
```

Out[ ]:
```
0    1.0
dtype: float64
```

In [ ]:
```
new_merchant_transactions['category_3'].mode()
```

Out[ ]:
```
0    A
dtype: object
```

In [ ]:
```
new_merchant_transactions['category_2'].mode()
```

Out[ ]:
```
0    1.0
dtype: float64
```

In [ ]:
```
#drop the rows with NaN merchant_id and replace the other NaN values

historical_transactions=historical_transactions.dropna(subset=['merchant_id'])
historical_transactions['category_3'].fillna('A',inplace=True)
historical_transactions['category_2'].fillna(1.0,inplace=True)
new_merchant_transactions=new_merchant_transactions.dropna(subset=['merchant_i
d'])
new_merchant_transactions['category_3'].fillna('A',inplace=True)
new_merchant_transactions['category_2'].fillna(1.0,inplace=True)
```

Remove the installments having less than 0 and with 999

In [ ]:
```
historical_transactions=historical_transactions[historical_transactions['insta
llments'] >=0]
historical_transactions=historical_transactions[historical_transactions['insta
llments'] !=999]
historical_transactions.shape
```

Out[ ]:
```
(28802740, 14)
```

In [ ]:
```
new_merchant_transactions=new_merchant_transactions[new_merchant_transactions[
'installments'] >=0]
new_merchant_transactions=new_merchant_transactions[new_merchant_transactions[
'installments'] !=999]
new_merchant_transactions.shape
```

Out[ ]:
```
(1883073, 14)
```

Since we have few categorical features , let's convert them to numeric form

In [ ]:
```
#https://stackoverflow.com/questions/40901770/is-there-a-simple-way-to-change-a-column-of-yes-no-to-1-0-in-a-pandas-dataframe
historical_transactions['authorized_flag'] = historical_transactions['authorized_flag'].map({'Y': 1, 'N': 0})
historical_transactions['category_1'] = historical_transactions['category_1'].map({'Y': 1, 'N': 0})
historical_transactions['category_3'] = historical_transactions['category_3'].map({'A': 1, 'B': 2, 'C': 3})
```

In [ ]:
```
merchants['category_4'] = merchants['category_4'].map({'Y': 1, 'N': 0})
merchants['most_recent_purchases_range'] = merchants['most_recent_purchases_range'].map({'A': 1, 'B': 2,'C': 3,'D': 4, 'E': 5})
merchants['most_recent_sales_range'] = merchants['most_recent_sales_range'].map({'A': 1, 'B': 2, 'C': 3, 'D': 4, 'E': 5})
```

In [ ]:
```
new_merchant_transactions['authorized_flag'] = new_merchant_transactions['authorized_flag'].map({'Y': 1, 'N': 0})
new_merchant_transactions['category_1'] = new_merchant_transactions['category_1'].map({'Y': 1, 'N': 0})
new_merchant_transactions['category_3'] = new_merchant_transactions['category_3'].map({'A': 1, 'B': 2, 'C': 3})
```

Convert the multiple categorical values in a single features to multiple features

In [ ]:
```
#https://datascience.stackexchange.com/questions/14847/multiple-categorical-values-for-a-single-feature-how-to-convert-them-to-binary-u
historical_transactions = pd.get_dummies(historical_transactions, columns=['category_2', 'category_3'])
historical_transactions.columns
```

Out[ ]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'merchant_category_id', 'merchant_id', 'month_lag', 'purchase_amount',
       'purchase_date', 'state_id', 'subsector_id', 'category_2_1.0',
       'category_2_2.0', 'category_2_3.0', 'category_2_4.0', 'category_2_5.
0',
       'category_3_1', 'category_3_2', 'category_3_3'],
      dtype='object')
```

In [ ]:
```
train_csv = pd.get_dummies(train_csv, columns=['feature_1', 'feature_2'])
train_csv.columns
```

Out[ ]:
```
Index(['first_active_month', 'card_id', 'feature_3', 'target', 'feature_1_1',
       'feature_1_2', 'feature_1_3', 'feature_1_4', 'feature_1_5',
       'feature_2_1', 'feature_2_2', 'feature_2_3'],
      dtype='object')
```

In [ ]:
```

```

Now merge the transactions and merchant dataset based on merchant_id

In [ ]:
```
historical_transactions=historical_transactions.merge(merchants,how='outer',on
='merchant_id')
historical_transactions.columns
```

Out[ ]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'merchant_category_id', 'merchant_id', 'month_lag', 'purchase_amount',
       'purchase_date', 'state_id', 'subsector_id', 'category_2_1.0',
       'category_2_2.0', 'category_2_3.0', 'category_2_4.0', 'category_2_5.
0',
       'category_3_1', 'category_3_2', 'category_3_3', 'active_months_lag12',
       'active_months_lag3', 'active_months_lag6', 'avg_purchases_lag12',
       'avg_purchases_lag3', 'avg_purchases_lag6', 'avg_sales_lag12',
       'avg_sales_lag3', 'avg_sales_lag6', 'category_4', 'merchant_group_id',
       'most_recent_purchases_range', 'most_recent_sales_range',
       'numerical_1'],
      dtype='object')
```

In [ ]:
```
new_merchant_transactions=new_merchant_transactions.merge(merchants,how='oute
r',on='merchant_id')
new_merchant_transactions.columns
```

Out[ ]:
```
Index(['authorized_flag', 'card_id', 'city_id', 'category_1', 'installments',
       'category_3', 'merchant_category_id', 'merchant_id', 'month_lag',
       'purchase_amount', 'purchase_date', 'category_2', 'state_id',
       'subsector_id', 'active_months_lag12', 'active_months_lag3',
       'active_months_lag6', 'avg_purchases_lag12', 'avg_purchases_lag3',
       'avg_purchases_lag6', 'avg_sales_lag12', 'avg_sales_lag3',
       'avg_sales_lag6', 'category_4', 'merchant_group_id',
       'most_recent_purchases_range', 'most_recent_sales_range', 'numerical_
1',
       'numerical_2'],
      dtype='object')
```

In [ ]:
```
new_merchant_transactions = pd.get_dummies(new_merchant_transactions, columns=
['category_2', 'category_3'])
```

Ref:
https://www.researchgate.net/publication/335158533_Predicting_Customer_Loyalty_Using_Various_Regression_M
(https://www.researchgate.net/publication/335158533_Predicting_Customer_Loyalty_Using_Various_Regression_

Now we will perform some aggregate function on our data which are numeric. The features are also mentioned in the above research paper

In [1]:
```
#ref: https://www.geeksforgeeks.org/python-pandas-dataframe-aggregate/
#This function performs the aggregate  operation on the numerical features
def aggregate_func(data,str_data):
    agg_func= {
      'authorized_flag':['mean'],
      'category_1':['mean','sum'],
      'category_2_1.0':['mean'],
      'category_2_2.0':['mean'],
      'category_2_3.0':['mean'],
      'category_2_4.0':['mean'],
      'category_2_5.0':['mean'],
      'category_3_1':['mean'],
      'category_3_2':['mean'],
      'category_3_3':['mean'],
      'city_id':['nunique'],
      'installments':['max','min','mean','std','sum'],
      'merchant_category_id':['nunique'],
      'merchant_id':['nunique'],
      'month_lag':['max','mean','min','std'],
      'purchase_amount':['max','mean','min','std','sum'],
      'state_id':['nunique'],
      'subsector_id':['nunique'],
      'active_months_lag12':['mean'],
      'active_months_lag3':['mean'],
      'active_months_lag6':['mean'],
      'avg_purchases_lag12':['max','mean','min','std','sum'],
      'avg_purchases_lag3':['max','min','mean','std','sum'],
      'avg_purchases_lag6':['max','min','mean','std','sum'],
      'avg_sales_lag12':['max','mean','min','std','sum'],
      'avg_sales_lag3':['max','min','mean','std','sum'],
      'avg_sales_lag6':['max','min','mean','std','sum'],
      'numerical_1':['max','min','mean','std','sum'],
      'category_4':['mean'],
      'merchant_group_id':['nunique'],
      'most_recent_purchases_range':['mean'],
      'most_recent_sales_range':['mean'],


    }
    featured_data=data.groupby(['card_id']).agg(agg_func)
    col_list=[]
    for col in featured_data.columns:
        col_str='_'.join(col)
        col_str=str_data + col_str
        ren_name=col_str.split(",")
        col_list.extend(ren_name)

    col_list.insert(0,'card_id')
    featured_data.reset_index(inplace=True)
    return featured_data,col_list
```

In [ ]:
```
%%time
X_train_history,col_list_hist=aggregate_func(historical_transactions[historica
l_transactions['authorized_flag']==1],'auth_history_')
```

```
CPU times: user 2min 26s, sys: 5.06 s, total: 2min 31s
Wall time: 2min 31s
```

In [ ]:
```
X_train_history.head(5)
```

Out[ ]:

	card_id	auth_history_authorized_flag_mean	auth_history_category_1_mean	auth_hist
0	C_ID_00007093c1	1.0	0.210526	
1	C_ID_0001238066	1.0	0.017094	
2	C_ID_0001506ef0	1.0	0.000000	
3	C_ID_0001793786	1.0	0.000000	
4	C_ID_000183fdda	1.0	0.030075	

In [ ]:
```
X_train_history.columns=col_list_hist
```

In [ ]:
```
X_train_new,col_list_new=aggregate_func(new_merchant_transactions,'new_')
X_train_new.columns=col_list_new
```

In [ ]:
```
%%time
X_train_unauth,col_list_unauth=aggregate_func(historical_transactions[historic
al_transactions['authorized_flag']==0],'unauth_hist_')
X_train_unauth.columns=col_list_unauth
```

```
CPU times: user 15 s, sys: 197 ms, total: 15.2 s
Wall time: 15.2 s
```

Since we have covered the numerical features, now let's concentrate on the date features.

In [ ]:
```
#https://www.kaggle.com/juliaflower/feature-selection-lgbm-with-python
historical_transactions['pur_date'] = pd.DatetimeIndex(historical_transactions
['purchase_date']).date
historical_transactions['pur_date'] = pd.DatetimeIndex(historical_transactions
['pur_date']).astype(np.int64) * 1e-9
```

In [ ]:
```
new_merchant_transactions['pur_date'] = pd.DatetimeIndex(new_merchant_transact
ions['purchase_date']).date
new_merchant_transactions['pur_date'] = pd.DatetimeIndex(new_merchant_transact
ions['pur_date']).astype(np.int64) * 1e-9
```

In [ ]:
```
historical_transactions['purchase_date']=pd.to_datetime(historical_transaction
s['purchase_date'],format='%Y-%m')
new_merchant_transactions['purchase_date']=pd.to_datetime(new_merchant_transac
tions['purchase_date'],format='%Y-%m')
```

In [ ]:
```
#ref: https://stackoverflow.com/questions/42822768/pandas-number-of-months-between-two-dates
import datetime
current_time = datetime.datetime.now()
historical_transactions['months_diff']= (current_time.year - historical_transactions.purchase_date.dt.year) * 12 + (current_time.month - historical_transactions.purchase_date.dt.month)
historical_transactions['purchase_month']=historical_transactions.purchase_date.dt.month
historical_transactions['purchase_year'] = historical_transactions['purchase_date'].dt.year
#data['purchase_month'] = data['purchase_date'].dt.month
historical_transactions['weekofyear'] = historical_transactions['purchase_date'].dt.weekofyear
historical_transactions['dayofweek'] = historical_transactions['purchase_date'].dt.dayofweek
historical_transactions['weekend'] = (historical_transactions.purchase_date.dt.weekday >=5).astype(int)
historical_transactions['hour'] = historical_transactions['purchase_date'].dt.hour
```

In [ ]:
```
import datetime
current_time = datetime.datetime.now()
new_merchant_transactions['months_diff']= (current_time.year - new_merchant_transactions.purchase_date.dt.year) * 12 + (current_time.month - new_merchant_transactions.purchase_date.dt.month)
new_merchant_transactions['purchase_month']=new_merchant_transactions.purchase_date.dt.month
new_merchant_transactions['purchase_year'] = new_merchant_transactions['purchase_date'].dt.year
#data['purchase_month'] = data['purchase_date'].dt.month
new_merchant_transactions['weekofyear'] = new_merchant_transactions['purchase_date'].dt.weekofyear
new_merchant_transactions['dayofweek'] = new_merchant_transactions['purchase_date'].dt.dayofweek
new_merchant_transactions['weekend'] = (new_merchant_transactions.purchase_date.dt.weekday >=5).astype(int)
new_merchant_transactions['hour'] = new_merchant_transactions['purchase_date'].dt.hour
```

Now perform the aggregate operation on the date features

In [ ]:
```
def aggregate_func_date(data,str_data):
    agg_func= {
      'months_diff':['mean','max','min'],
      'purchase_month':['max','min','mean','std'],
      'purchase_year': ['mean', 'max', 'min', 'std','nunique'],
      'weekofyear': ['mean','max','min','nunique'],
      'dayofweek': ['mean'],
      'weekend': ['sum', 'mean'],
      'hour': ['mean','max','min'],
      'pur_date':['max','min',np.ptp]
    }
    featured_data=data.groupby(['card_id']).agg(agg_func)
    col_list=[]
    for col in featured_data.columns:
        col_str='_'.join(col)
        col_str=str_data + col_str
        ren_name=col_str.split(",")
        col_list.extend(ren_name)
    col_list.insert(0,'card_id')
    featured_data.reset_index(inplace=True)
    return featured_data,col_list
```

In [ ]:
```
%%time
X_train_auth_date,col_list_date=aggregate_func_date(historical_transactions[hi
storical_transactions['authorized_flag']==1],'auth_hist_')
X_train_auth_date.columns=col_list_date
```

```
CPU times: user 43.3 s, sys: 8.75 s, total: 52 s
Wall time: 51.8 s
```

In [ ]:
```
%%time
X_train_unauth_date,col_list_date=aggregate_func_date(historical_transactions[
historical_transactions['authorized_flag']==0],'unauth_hist_')
X_train_unauth_date.columns=col_list_date
```

```
CPU times: user 19.3 s, sys: 232 ms, total: 19.6 s
Wall time: 19.3 s
```

In [ ]:
```
%%time
X_train_new_date,col_list_date=aggregate_func_date(new_merchant_transactions,
'new_')
X_train_new_date.columns=col_list_date
```

```
CPU times: user 17.9 s, sys: 104 ms, total: 18 s
Wall time: 17.9 s
```

Merging them altogther

In [ ]:
```
X_train_unauth_merge=X_train_unauth.merge(X_train_unauth_date,on='card_id',how
='left')
```

In [ ]:
```
X_train_unauth_merge.head(5)
```

Out[ ]:

	card_id	unauth_hist_authorized_flag_mean	unauth_hist_category_1_mean	unauth_his
0	C_ID_00007093c1	0.0	0.114286	
1	C_ID_0001238066	0.0	0.000000	
2	C_ID_0001506ef0	0.0	0.000000	
3	C_ID_0001793786	0.0	0.074074	
4	C_ID_000183fdda	0.0	0.000000	

5 rows × 83 columns

In [ ]:
```
X_train_auth_merge=X_train_history.merge(X_train_auth_date,on='card_id',how='left')
```

In [ ]:
```
X_train_auth_merge.head(5)
```

Out[ ]:

	card_id	auth_history_authorized_flag_mean	auth_history_category_1_mean	auth_hist
0	C_ID_00007093c1	1.0	0.210526	
1	C_ID_0001238066	1.0	0.017094	
2	C_ID_0001506ef0	1.0	0.000000	
3	C_ID_0001793786	1.0	0.000000	
4	C_ID_000183fdda	1.0	0.030075	

5 rows × 83 columns

In [ ]:
```
history=X_train_auth_merge.merge(X_train_unauth_merge,on='card_id',how='left')
```

In [ ]:
```
history.shape
```

Out[ ]:
```
(325540, 165)
```

In [ ]:
```
new=X_train_new.merge(X_train_new_date,on='card_id',how='left')
```

In [ ]:
```
new.head(5)
```

Out[ ]:

	card_id	new_authorized_flag_mean	new_category_1_mean	new_category_1_sum	ne
0	C_ID_00007093c1	1	0.000000	0	
1	C_ID_0001238066	1	0.083333	2	
2	C_ID_0001506ef0	1	0.000000	0	
3	C_ID_0001793786	1	0.000000	0	
4	C_ID_000183fdda	1	0.000000	0	

5 rows × 83 columns

In [ ]:
```
train=train_csv.merge(history,on='card_id',how='left')
```

In [ ]:
```
X_train=train.merge(new,on='card_id',how='left')
X_train.shape
```

Out[ ]:
```
(201917, 258)
```

In [ ]:
```
X_train.head(5)
```

Out[ ]:

	first_active_month	card_id	feature_3	target	feature_1_1	feature_1_2	feature_1_
0	2017-06	C_ID_92a2005557	1	-0.820283	0	0	
1	2017-01	C_ID_3d0044924f	0	0.392913	0	0	
2	2016-08	C_ID_d639edf6cd	0	0.688056	0	1	
3	2017-09	C_ID_186d6a6901	0	0.142495	0	0	
4	2017-11	C_ID_cdbd2c0db2	0	-0.159749	1	0	

5 rows × 258 columns

In [ ]:
```
X_train=X_train.sort_values(by=['first_active_month'])
```

In [ ]:
```
Y_train=X_train['target']
```

In [ ]:
```
X_train=X_train.drop(columns=['first_active_month','card_id','target'])
X_train.shape
```

Out[ ]:
```
(201917, 255)
```

For binary classification problem , let's convert the target feature

In [ ]:
```
Y_train_binary=Y_train
Y=Y_train_binary.values.reshape(-1,1)
```

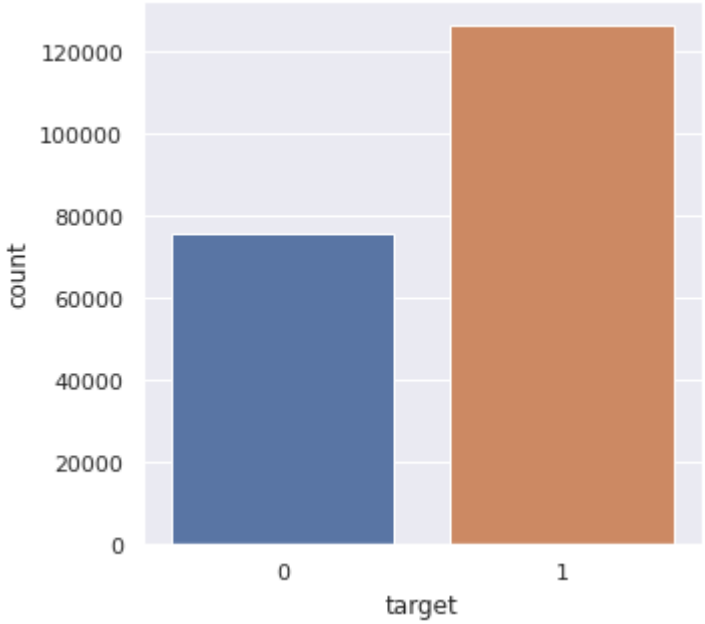
In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
Y_train_binary = scaler.fit_transform(Y)
```

In [ ]:
```
Y_train_binary=pd.DataFrame(Y_train_binary,columns=['target'])
Y_train_binary['target']=Y_train_binary['target'].map(lambda x: 1 if x > 0 els
e 0)
```

To check whether two classes are present or not

In [102]:
```
sns.countplot(x='target',data=Y_train_binary)
```

Out[102]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe488cfe240>
```



So we have both classes - 0 & 1

Save them in a pickle file for future purpose

In [ ]:
```
X_train_history.to_pickle('/content/X_train_history.pkl')
X_train_new.to_pickle('/content/X_train_new.pkl')
X_train_unauth.to_pickle('/content/X_train_unauth.pkl')
X_train_auth_date.to_pickle('/content/drive/My Drive/kaggle/elo/data/X_train_auth_date_v2.pkl')
X_train_unauth_date.to_pickle('/content/drive/My Drive/kaggle/elo/data/X_train_unauth_date_v2.pkl')
X_train_new_date.to_pickle('/content/drive/My Drive/kaggle/elo/data/X_train_new_date_v2.pkl')
#X_train.to_pickle('/content/drive/My Drive/kaggle/elo/data/X_train.pkl')
#Y_train.to_pickle('/content/drive/My Drive/kaggle/elo/data/Y_train.pkl')
#Y_train_binary.to_pickle('/content/drive/My Drive/kaggle/elo/data/Y_train_binary.pkl')
```

Model - Classification Problem

In [97]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
led
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
ocker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis
t all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that get
s preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

```
/kaggle/input/pickledata/data/X_train_new.pkl
/kaggle/input/pickledata/data/X_train_history.pkl
/kaggle/input/pickledata/data/X_train.pkl
/kaggle/input/pickledata/data/X_train_auth_date.pkl
/kaggle/input/pickledata/data/X_train_unauth_date.pkl
/kaggle/input/pickledata/data/X_train_unauth.pkl
/kaggle/input/pickledata/data/train_csv_card.pkl
/kaggle/input/pickledata/data/Y_train.pkl
/kaggle/input/pickledata/data/X_train_new_date.pkl
/kaggle/input/elo-merchant-category-recommendation/test.csv
/kaggle/input/elo-merchant-category-recommendation/sample_submission.csv
/kaggle/input/elo-merchant-category-recommendation/train.csv
/kaggle/input/elo-merchant-category-recommendation/new_merchant_transactions.
csv
/kaggle/input/elo-merchant-category-recommendation/Data Dictionary.xlsx
/kaggle/input/elo-merchant-category-recommendation/merchants.csv
/kaggle/input/elo-merchant-category-recommendation/historical_transactions.cs
v
/kaggle/input/elo-merchant-category-recommendation/Data_Dictionary.xlsx
/kaggle/input/elomodel/X_train_unauth_date_v2.pkl
/kaggle/input/elomodel/X_train_auth_date_v2.pkl
/kaggle/input/elomodel/X_train_new_date_v2.pkl
```

In [100]:
```
train_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/trai
n.csv')
test_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/test.
csv')
test_csv.shape
```

Out[100]: (123623, 5)

In [101]:
```
test_csv = pd.get_dummies(test_csv, columns=['feature_1', 'feature_2'])
train_csv=pd.get_dummies(train_csv, columns=['feature_1', 'feature_2'])
```

In [103]:
```
import datetime
#train
train_csv['first_active_month'] = pd.to_datetime(train_csv['first_active_mont
h'])
train_csv['year'] = train_csv['first_active_month'].dt.year
train_csv['month'] = train_csv['first_active_month'].dt.month
train_csv['howlong'] = (datetime.date(2018,2,1) - train_csv['first_active_mont
h'].dt.date).dt.days
```

In [104]:
```
#test
test_csv['first_active_month'] = pd.to_datetime(test_csv['first_active_month'
])
test_csv['year'] = test_csv['first_active_month'].dt.year
test_csv['month'] = test_csv['first_active_month'].dt.month
test_csv['howlong'] = (datetime.date(2018,2,1) - test_csv['first_active_month'
].dt.date).dt.days
```

In [105]:
```
X_train_unauth=pd.read_pickle('/kaggle/input/pickledata/data/X_train_unauth.pk
l')
X_train_unauth_date=pd.read_pickle('/kaggle/input/elomodel/X_train_unauth_date
_v2.pkl')
X_train_auth_date=pd.read_pickle('/kaggle/input/elomodel/X_train_auth_date_v2.
pkl')
X_train_history=pd.read_pickle('/kaggle/input/pickledata/data/X_train_history.
pkl')
X_train_new=pd.read_pickle('/kaggle/input/pickledata/data/X_train_new.pkl')
X_train_new_date=pd.read_pickle('/kaggle/input/elomodel/X_train_new_date_v2.pk
l')
```

In [106]:
```
X_train_new_date.shape
```

Out[106]:
```
(286913, 26)
```

In [107]:
```
X_train_unauth_date.columns
```

Out[107]:
```
Index(['card_id', 'unauth_hist_months_diff_mean',
       'unauth_hist_months_diff_max', 'unauth_hist_months_diff_min',
       'unauth_hist_purchase_month_max', 'unauth_hist_purchase_month_min',
       'unauth_hist_purchase_month_mean', 'unauth_hist_purchase_month_std',
       'unauth_hist_purchase_year_mean', 'unauth_hist_purchase_year_max',
       'unauth_hist_purchase_year_min', 'unauth_hist_purchase_year_std',
       'unauth_hist_purchase_year_nunique', 'unauth_hist_weekofyear_mean',
       'unauth_hist_weekofyear_max', 'unauth_hist_weekofyear_min',
       'unauth_hist_weekofyear_nunique', 'unauth_hist_dayofweek_mean',
       'unauth_hist_weekend_sum', 'unauth_hist_weekend_mean',
       'unauth_hist_hour_mean', 'unauth_hist_hour_max', 'unauth_hist_hour_mi
n',
       'unauth_hist_pur_date_max', 'unauth_hist_pur_date_min',
       'unauth_hist_pur_date_ptp'],
      dtype='object')
```

In [108]:
```
X_train_auth_date.columns
```

Out[108]:
```
Index(['card_id', 'auth_hist_months_diff_mean', 'auth_hist_months_diff_max',
       'auth_hist_months_diff_min', 'auth_hist_purchase_month_max',
       'auth_hist_purchase_month_min', 'auth_hist_purchase_month_mean',
       'auth_hist_purchase_month_std', 'auth_hist_purchase_year_mean',
       'auth_hist_purchase_year_max', 'auth_hist_purchase_year_min',
       'auth_hist_purchase_year_std', 'auth_hist_purchase_year_nunique',
       'auth_hist_weekofyear_mean', 'auth_hist_weekofyear_max',
       'auth_hist_weekofyear_min', 'auth_hist_weekofyear_nunique',
       'auth_hist_dayofweek_mean', 'auth_hist_weekend_sum',
       'auth_hist_weekend_mean', 'auth_hist_hour_mean', 'auth_hist_hour_max',
       'auth_hist_hour_min', 'auth_hist_pur_date_max',
       'auth_hist_pur_date_min', 'auth_hist_pur_date_ptp'],
      dtype='object')
```

In [109]:
```
X_train_auth_date.shape
```

Out[109]:
```
(325540, 26)
```

In [110]:
```
X_train_unauth_date.shape
```

Out[110]:
```
(274262, 26)
```

In [111]:
```
X_train_unauth_date.head(10)
```

Out[111]:

	card_id	unauth_hist_months_diff_mean	unauth_hist_months_diff_max	unauth_hist_m
0	C_ID_00007093c1	38.028571	43	
1	C_ID_0001238066	33.333333	34	
2	C_ID_0001506ef0	34.500000	40	
3	C_ID_0001793786	39.518519	43	
4	C_ID_000183fdda	37.714286	38	
5	C_ID_00024e244b	36.411765	42	
6	C_ID_0002709b5a	37.166667	45	
7	C_ID_00027503e2	37.444444	40	
8	C_ID_000298032a	39.500000	42	
9	C_ID_0002ba3c2e	38.600000	45	

10 rows × 26 columns

In [112]:
```
X_train_unauth_merge=X_train_unauth.merge(X_train_unauth_date,on='card_id',how
='left')
X_train_auth_merge=X_train_history.merge(X_train_auth_date,on='card_id',how='l
eft')
history=X_train_auth_merge.merge(X_train_unauth_merge,on='card_id',how='left')
new=X_train_new.merge(X_train_new_date,on='card_id',how='left')
test=test_csv.merge(history,on='card_id',how='left')
```

In [117]:
```
X_test=test.merge(new,on='card_id',how='left')
X_test.shape
```

Out[117]: (123623, 305)

In [119]:
```
train=train_csv.merge(history,on='card_id',how='left')
X_train=train.merge(new,on='card_id',how='left')
X_train.shape
```

Out[119]: (201917, 306)

In [120]:
```
X_train=X_train.sort_values(by=['first_active_month'])
```

In [122]:
```
X_test=X_test.drop(columns=['first_active_month','card_id'])
X_test.shape
```

Out[122]: (123623, 303)

In [123]:
```
Y_train=X_train['target']
```

In [124]:
```
X_train=X_train.drop(columns=['first_active_month','card_id','target'])
X_train.shape
```

Out[124]: (201917, 303)

In [125]:
```
type(Y_train)
```

Out[125]: pandas.core.series.Series

In [126]:
```
Y_train.head(5)
```

Out[126]:
```
69678     0.645766
145874   -0.069560
9322     -2.544361
33398    -3.573636
152463   -0.693447
Name: target, dtype: float64
```

In [127]:
```
#converting the target variable from continuous to 0  1
Y_train_binary=Y_train
Y=Y_train_binary.values.reshape(-1,1)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
Y_train_binary = scaler.fit_transform(Y)
Y_train_binary=pd.DataFrame(Y_train_binary,columns=['target'])
Y_train_binary['target']=Y_train_binary['target'].map(lambda x: 1 if x > 0 els
e 0)
Y_train_binary
```

Out[127]:

	target
0	1
1	1
2	0
3	0
4	0
...	...
201912	1
201913	1
201914	1
201915	1
201916	1

201917 rows × 1 columns

LGBM Classifier

In [129]:
```
random_grid={'boosting_type':['gbdt'],
             'objective':['binary'],
             'num_leaves':[15],
             'learning_rate': [0.01,0.001,0.05,0.02,0.03],
              'n_estimators':[2011],
              'max_depth':[3,5,10,15],
              'metric':['none'],
             'min_child_samples':[84],
             'bagging_freq':[2],
             'bagging_seed':[5,10,12,15],
             'lambda_l1':[2.786857535211943e-08],
             'lambda_l2':[1.283262179401335],
              'bagging_fraction':[0.9959913528376535],
              'feature_fraction':[0.6110722387385512],
               'max_bin':[10,30,50,70,80],
             'min_data_in_leaf':[10,20,30,40],
              'reg_lambda':[0.9,1.0,0.5,1.5,0.1]
            }
```

In [134]:
```
X_train_init=X_train
Y_train_init=Y_train_binary
```

In [135]:
```
X_train=X_train_init[:170000]
Y_train=Y_train_init[:170000]
X_CV=X_train_init[170000:]
Y_CV=Y_train_init[170000:]
```

In [136]:
```
X_train.shape
```

Out[136]:
```
(170000, 303)
```

In [137]:
```
from sklearn.metrics import roc_auc_score
import lightgbm as lgb
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
```

In [139]:
```
model=lgb.LGBMClassifier()
lgb_randoms = RandomizedSearchCV(estimator = model, param_distributions = rand
om_grid, n_iter = 10,
                                 cv = 5, verbose=2, random_state=42,scoring='r
oc_auc')

lgb_randoms.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=15, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  2.3min remaining:    0.
0s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=15, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=15, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=15, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=15, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=3, max_bin=10,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=5,
bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=5,
bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=5,
bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=5,
bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=10,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=5,
bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=10, min_child_samples=84, metric=none, max_depth=15, max_bin=1
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=5, max_bin=50,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=10,
learning_rate=0.02, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=80,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=8
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=80,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=8
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=80,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=8
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=80,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=8
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=80,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=1.0, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=15, max_bin=8
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.0min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.0min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.0min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.0min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=3, max_bin=50,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.0min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=70,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=7
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=10, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=70,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=7
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=10, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=70,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=7
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=10, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=70,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=7
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=10, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=70,
learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
0, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.5, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=7
0, learning_rate=0.05, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=10, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=30,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=3
0, learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=30,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=3
0, learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.2min
[CV] reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=30,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=3
0, learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=30,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=3
0, learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=30,
learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
2, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.1, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=30, min_child_samples=84, metric=none, max_depth=10, max_bin=3
0, learning_rate=0.03, lambda_l2=1.283262179401335, lambda_l1=2.7868575352119
43e-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed
=12, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.1min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.3min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.4min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.4min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.4min
[CV] reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min_
data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  reg_lambda=0.9, objective=binary, num_leaves=15, n_estimators=2011, min
_data_in_leaf=20, min_child_samples=84, metric=none, max_depth=5, max_bin=70,
learning_rate=0.01, lambda_l2=1.283262179401335, lambda_l1=2.786857535211943e
-08, feature_fraction=0.6110722387385512, boosting_type=gbdt, bagging_seed=1
5, bagging_freq=2, bagging_fraction=0.9959913528376535, total= 2.4min

[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed: 108.5min finished
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

Out[139]:
```
RandomizedSearchCV(cv=5, estimator=LGBMClassifier(),
                   param_distributions={'bagging_fraction': [0.99599135283765
35],
                                        'bagging_freq': [2],
                                        'bagging_seed': [5, 10, 12, 15],
                                        'boosting_type': ['gbdt'],
                                        'feature_fraction': [0.61107223873855
12],
                                        'lambda_l1': [2.786857535211943e-08],
                                        'lambda_l2': [1.283262179401335],
                                        'learning_rate': [0.01, 0.001, 0.05,
                                                          0.02, 0.03],
                                        'max_bin': [10, 30, 50, 70, 80],
                                        'max_depth': [3, 5, 10, 15],
                                        'metric': ['none'],
                                        'min_child_samples': [84],
                                        'min_data_in_leaf': [10, 20, 30, 40],
                                        'n_estimators': [2011],
                                        'num_leaves': [15],
                                        'objective': ['binary'],
                                        'reg_lambda': [0.9, 1.0, 0.5, 1.5,
                                                       0.1]},
                   random_state=42, scoring='roc_auc', verbose=2)
```

In [140]:
```
Y_pred=lgb_randoms.predict(X_CV)
roc_auc_score(Y_pred, Y_CV)
```

Out[140]:
```
0.648474182589488
```

In [142]:
```
Y_pred=lgb_randoms.predict(X_train)
roc_auc_score(Y_pred, Y_train)
```

Out[142]:
```
0.688158183160029
```

In [144]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_pred, Y_train)
```

Out[144]:
```
0.6948
```

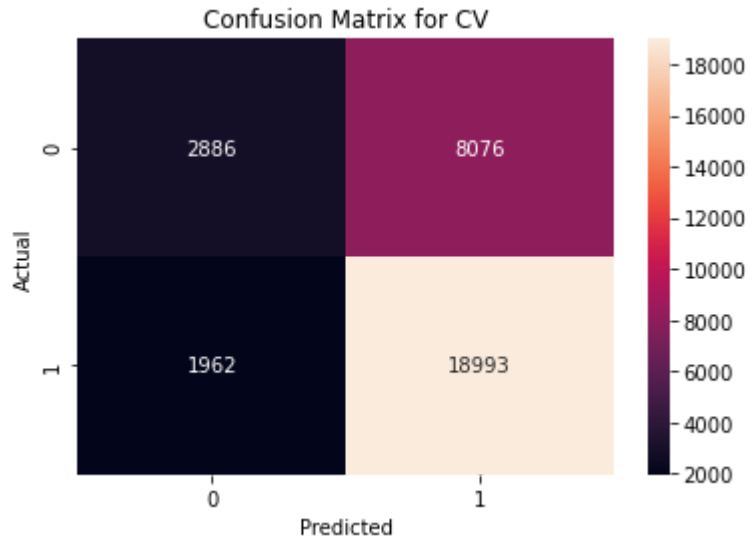
In [145]:
```
Y_pred=lgb_randoms.predict(X_CV)
accuracy_score(Y_pred, Y_CV)
```

Out[145]:
```
0.6854967572140239
```

In [147]:
```
lgb_randoms.best_params_
```

Out[147]:
```
{'reg_lambda': 0.9,
 'objective': 'binary',
 'num_leaves': 15,
 'n_estimators': 2011,
 'min_data_in_leaf': 30,
 'min_child_samples': 84,
 'metric': 'none',
 'max_depth': 15,
 'max_bin': 10,
 'learning_rate': 0.02,
 'lambda_l2': 1.283262179401335,
 'lambda_l1': 2.786857535211943e-08,
 'feature_fraction': 0.6110722387385512,
 'boosting_type': 'gbdt',
 'bagging_seed': 15,
 'bagging_freq': 2,
 'bagging_fraction': 0.9959913528376535}
```

In [132]:
```
conf_matrix = confusion_matrix(Y_CV, Y_pred, labels)
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(conf_matrix, index=class_label, columns=class_la
bel)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix for CV")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [148]:
```
import pickle
filename = 'lgbm_classifier_model.sav'
pickle.dump(lgb_randoms, open(filename, 'wb'))
```

XGBOOST Classifier

In [116]:
```
random_grid = {'learning_rate'    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 'max_depth'        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 'n_estimators'     : [100,200,300,500],
 'subsample'        : [0.5,0.8,1.0],
 'min_child_weight' : [ 1, 3, 5, 7 ],
 'reg_alpha'        : [100,200,300],
 'reg_lambda'       : [100,200,300],
 'gamma'            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 'colsample_bytree' : [ 0.3, 0.4, 0.5 , 0.7 ]
}
print(random_grid)
```

```
{'learning_rate': [0.05, 0.1, 0.15, 0.2, 0.25, 0.3], 'max_depth': [3, 4, 5,
6, 8, 10, 12, 15], 'n_estimators': [100, 200, 300, 500], 'subsample': [0.5,
0.8, 1.0], 'min_child_weight': [1, 3, 5, 7], 'reg_alpha': [100, 200, 300], 'r
eg_lambda': [100, 200, 300], 'gamma': [0.0, 0.1, 0.2, 0.3, 0.4], 'colsample_b
ytree': [0.3, 0.4, 0.5, 0.7]}
```

In [117]:
```
# lgb_randoms = RandomizedSearchCV(estimator = model, param_distributions = ra
ndom_grid, n_iter = 10,
#                                  cv = 5, verbose=2, random_state=42,scoring
='roc_auc')

# lgb_randoms.fit(X_train,Y_train)
```

In [118]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
x_model = xgb. XGBClassifier()
xgb_random = RandomizedSearchCV(estimator = x_model, param_distributions = ran
dom_grid, n_iter = 10,
                                cv = 5, verbose=2, random_state=42,scoring='r
oc_auc')
xgb_random.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chil
d_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worke
rs.
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chi
ld_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4,
total= 1.6min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chil
d_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.6min remaining:    0.
0s
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chi
ld_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4,
total= 1.6min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chil
d_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chi
ld_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4,
total= 1.5min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chil
d_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chi
ld_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4,
total= 1.6min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chil
d_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=100, min_chi
ld_weight=3, max_depth=5, learning_rate=0.3, gamma=0.0, colsample_bytree=0.4,
total= 1.5min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chil
d_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chi
ld_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.
4, total= 9.8min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chil
d_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chi
ld_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.
4, total= 9.8min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chil
d_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chi
ld_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.
4, total= 9.7min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chil
d_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chi
ld_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.
4, total= 9.6min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chil
d_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=200, reg_alpha=200, n_estimators=500, min_chi
ld_weight=7, max_depth=6, learning_rate=0.05, gamma=0.2, colsample_bytree=0.
4, total= 8.8min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chil
d_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chi
ld_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.
4, total= 5.0min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chil
d_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chi
ld_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.
4, total= 4.8min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chil
d_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chi
ld_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.
4, total= 5.0min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chil
d_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chi
ld_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.
4, total= 5.0min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chil
d_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.8, reg_lambda=100, reg_alpha=200, n_estimators=300, min_chi
ld_weight=3, max_depth=4, learning_rate=0.15, gamma=0.1, colsample_bytree=0.
4, total= 4.8min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chil
d_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.7

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chi
ld_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
7, total=11.5min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chil
d_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.7

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chi
ld_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
7, total=12.2min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chil
d_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.7

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chi
ld_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
7, total=11.8min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chil
d_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.7

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chi
ld_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
7, total=11.8min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chil
d_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.7

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=200, min_chi
ld_weight=7, max_depth=12, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
7, total=10.9min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chil
d_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.5
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chi
ld_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.
5, total=15.3min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chil
d_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.5
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chi
ld_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.
5, total=19.2min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chil
d_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.5
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chi
ld_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.
5, total=17.8min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chil
d_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.5
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chi
ld_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.
5, total=17.2min
[CV] subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chil
d_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.5
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=200, reg_alpha=100, n_estimators=500, min_chi
ld_weight=7, max_depth=15, learning_rate=0.15, gamma=0.2, colsample_bytree=0.
5, total=17.5min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chil
d_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chi
ld_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.
4, total= 4.9min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chil
d_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chi
ld_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.
4, total= 4.9min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chil
d_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chi
ld_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.
4, total= 4.9min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chil
d_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chi
ld_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.
4, total= 4.7min
[CV] subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chil
d_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=1.0, reg_lambda=300, reg_alpha=300, n_estimators=500, min_chi
ld_weight=3, max_depth=10, learning_rate=0.25, gamma=0.0, colsample_bytree=0.
4, total= 4.5min
[CV] subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chil
d_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chi
ld_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.
4, total= 5.3min
[CV] subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chil
d_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chi
ld_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.
4, total= 5.3min
[CV] subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chil
d_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chi
ld_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.
4, total= 5.3min
[CV] subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chil
d_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.4
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chi
ld_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.
4, total= 5.3min
[CV] subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chil
d_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=100, reg_alpha=100, n_estimators=100, min_chi
ld_weight=3, max_depth=15, learning_rate=0.1, gamma=0.0, colsample_bytree=0.
4, total= 5.2min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chil
d_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.5

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chi
ld_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.
5, total= 9.7min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chil
d_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.5

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chi
ld_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.
5, total= 9.4min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chil
d_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.5

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chi
ld_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.
5, total= 9.6min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chil
d_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.5

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chi
ld_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.
5, total= 9.4min
[CV] subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chil
d_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.5

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=100, reg_alpha=300, n_estimators=500, min_chi
ld_weight=1, max_depth=4, learning_rate=0.05, gamma=0.0, colsample_bytree=0.
5, total= 9.1min
[CV] subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chil
d_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.3

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chi
ld_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
3, total= 3.7min
[CV] subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chil
d_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.3

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chi
ld_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
3, total= 3.5min
[CV] subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chil
d_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.3

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chi
ld_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
3, total= 3.6min
[CV] subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chil
d_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.3

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chi
ld_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
3, total= 3.6min
[CV] subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chil
d_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.3

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.8, reg_lambda=200, reg_alpha=300, n_estimators=100, min_chi
ld_weight=1, max_depth=15, learning_rate=0.2, gamma=0.2, colsample_bytree=0.
3, total= 3.6min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chil
d_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chi
ld_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.
4, total= 7.8min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chil
d_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chi
ld_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.
4, total= 7.8min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chil
d_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chi
ld_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.
4, total= 7.9min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chil
d_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

```
[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chi
ld_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.
4, total= 8.0min
[CV] subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chil
d_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.4

/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)

[CV]  subsample=0.5, reg_lambda=300, reg_alpha=200, n_estimators=300, min_chi
ld_weight=7, max_depth=10, learning_rate=0.2, gamma=0.1, colsample_bytree=0.
4, total= 7.5min

[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed: 379.6min finished
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

Out[118]:

```
RandomizedSearchCV(cv=5,
                   estimator=XGBClassifier(base_score=None, booster=None,
                                           colsample_bylevel=None,
                                           colsample_bynode=None,
                                           colsample_bytree=None, gamma=None,
                                           gpu_id=None, importance_type='gai
n',
                                           interaction_constraints=None,
                                           learning_rate=None,
                                           max_delta_step=None, max_depth=Non
e,
                                           min_child_weight=None, missing=na
n,
                                           monotone_constraints=None,
                                           n_estimators=100,...
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.
2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7],
                                        'n_estimators': [100, 200, 300, 500],
                                        'reg_alpha': [100, 200, 300],
                                        'reg_lambda': [100, 200, 300],
                                        'subsample': [0.5, 0.8, 1.0]},
                   random_state=42, scoring='roc_auc', verbose=2)
```

In [119]:
```
xgb_random.best_params_
```

Out[119]:
```
{'subsample': 0.8,
 'reg_lambda': 100,
 'reg_alpha': 300,
 'n_estimators': 500,
 'min_child_weight': 1,
 'max_depth': 4,
 'learning_rate': 0.05,
 'gamma': 0.0,
 'colsample_bytree': 0.5}
```

In [120]:
```
Y_pred=xgb_random.predict(X_CV)
roc_auc_score(Y_pred, Y_CV)
```

Out[120]:
```
0.6375713948329914
```

In [121]:
```
Y_pred=xgb_random.predict(X_train)
roc_auc_score(Y_pred, Y_train)
```

Out[121]:
```
0.6531587813076793
```

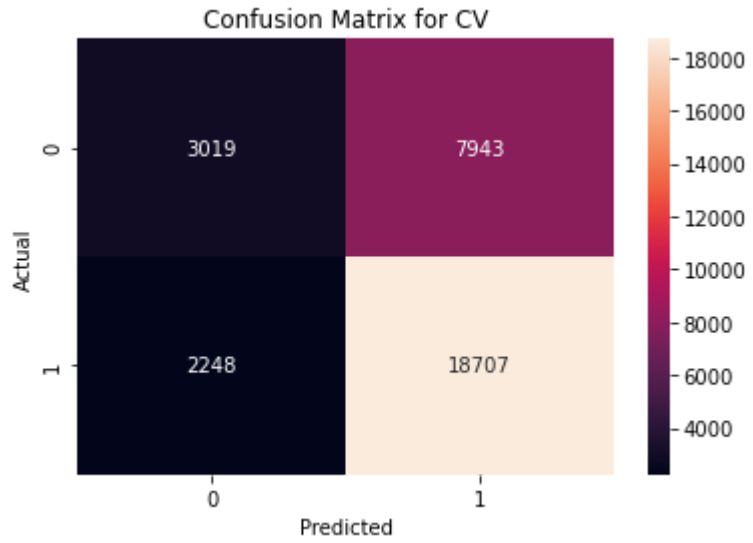
In [122]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_pred, Y_train)
```

Out[122]:
```
0.6713117647058824
```

In [123]:
```
Y_pred=xgb_random.predict(X_CV)
accuracy_score(Y_pred, Y_CV)
```

Out[123]:
```
0.6807030735971425
```

In [128]:
```
conf_matrix = confusion_matrix(Y_CV, Y_pred, labels)
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(conf_matrix, index=class_label, columns=class_la
bel)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix for CV")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [124]:
```
import pickle
filename = 'xgb_classifier_model.sav'
pickle.dump(xgb_random, open(filename, 'wb'))
```

In [1]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Validation ROC_AUC Score", "Validation Accuracy"]
x.add_row(["LGBM Classifier", 0.65, 0.69])
x.add_row(["XGBOOST Classifier", 0.64, 0.68])
print(x)
```

```
+-------------------+--------------------------+---------------------+
|       Model       | Validation ROC_AUC Score | Validation Accuracy |
+-------------------+--------------------------+---------------------+
|  LGBM Classifier  |           0.65           |         0.69        |
| XGBOOST Classifier |           0.64           |         0.68        |
+-------------------+--------------------------+---------------------+
```

Regression with LGBM regressor

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
led
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
ocker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis
t all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that get
s preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

```
/kaggle/input/pickledatav2/X_train_new_date_v2.pkl
/kaggle/input/pickledatav2/X_train_unauth_date_v2.pkl
/kaggle/input/pickledatav2/X_train_auth_date_v2.pkl
/kaggle/input/pickledata/data/X_train_new_date.pkl
/kaggle/input/pickledata/data/Y_train.pkl
/kaggle/input/pickledata/data/X_train.pkl
/kaggle/input/pickledata/data/X_train_unauth_date.pkl
/kaggle/input/pickledata/data/train_csv_card.pkl
/kaggle/input/pickledata/data/X_train_auth_date.pkl
/kaggle/input/pickledata/data/X_train_new.pkl
/kaggle/input/pickledata/data/X_train_unauth.pkl
/kaggle/input/pickledata/data/X_train_history.pkl
/kaggle/input/elo-merchant-category-recommendation/new_merchant_transactions.
csv
/kaggle/input/elo-merchant-category-recommendation/merchants.csv
/kaggle/input/elo-merchant-category-recommendation/test.csv
/kaggle/input/elo-merchant-category-recommendation/sample_submission.csv
/kaggle/input/elo-merchant-category-recommendation/historical_transactions.cs
v
/kaggle/input/elo-merchant-category-recommendation/train.csv
/kaggle/input/elo-merchant-category-recommendation/Data_Dictionary.xlsx
/kaggle/input/elo-merchant-category-recommendation/Data Dictionary.xlsx
```

In [2]:
```
#import the required libraries
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [3]:
```
train_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/trai
n.csv')
```

In [4]:
```
test_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/test.csv')
test_csv.shape
```

Out[4]:
```
(123623, 5)
```

In [5]:
```
test_csv = pd.get_dummies(test_csv, columns=['feature_1', 'feature_2'])
```

In [6]:
```
train_csv=pd.get_dummies(train_csv, columns=['feature_1', 'feature_2'])
```

In [7]:
```
import datetime
#train
train_csv['first_active_month'] = pd.to_datetime(train_csv['first_active_month'])
train_csv['year'] = train_csv['first_active_month'].dt.year
train_csv['month'] = train_csv['first_active_month'].dt.month
train_csv['howlong'] = (datetime.date(2018,2,1) - train_csv['first_active_month'].dt.date).dt.days
```

In [8]:
```
#test
test_csv['first_active_month'] = pd.to_datetime(test_csv['first_active_month'])
test_csv['year'] = test_csv['first_active_month'].dt.year
test_csv['month'] = test_csv['first_active_month'].dt.month
test_csv['howlong'] = (datetime.date(2018,2,1) - test_csv['first_active_month'].dt.date).dt.days
```

In [9]:
```
X_train_unauth=pd.read_pickle('/kaggle/input/pickledata/data/X_train_unauth.pkl')
X_train_unauth_date=pd.read_pickle('/kaggle/input/pickledatav2/X_train_unauth_date_v2.pkl')
X_train_auth_date=pd.read_pickle('/kaggle/input/pickledatav2/X_train_auth_date_v2.pkl')
X_train_history=pd.read_pickle('/kaggle/input/pickledata/data/X_train_history.pkl')
X_train_new=pd.read_pickle('/kaggle/input/pickledata/data/X_train_new.pkl')
X_train_new_date=pd.read_pickle('/kaggle/input/pickledatav2/X_train_new_date_v2.pkl')
```

In [10]:
```
X_train_new_date.shape
```

Out[10]:
```
(286913, 26)
```

In [11]:
```
X_train_unauth_date.columns
```

Out[11]:
```
Index(['card_id', 'unauth_hist_months_diff_mean',
       'unauth_hist_months_diff_max', 'unauth_hist_months_diff_min',
       'unauth_hist_purchase_month_max', 'unauth_hist_purchase_month_min',
       'unauth_hist_purchase_month_mean', 'unauth_hist_purchase_month_std',
       'unauth_hist_purchase_year_mean', 'unauth_hist_purchase_year_max',
       'unauth_hist_purchase_year_min', 'unauth_hist_purchase_year_std',
       'unauth_hist_purchase_year_nunique', 'unauth_hist_weekofyear_mean',
       'unauth_hist_weekofyear_max', 'unauth_hist_weekofyear_min',
       'unauth_hist_weekofyear_nunique', 'unauth_hist_dayofweek_mean',
       'unauth_hist_weekend_sum', 'unauth_hist_weekend_mean',
       'unauth_hist_hour_mean', 'unauth_hist_hour_max', 'unauth_hist_hour_mi
n',
       'unauth_hist_pur_date_max', 'unauth_hist_pur_date_min',
       'unauth_hist_pur_date_ptp'],
      dtype='object')
```

In [12]:
```
X_train_auth_date.columns
```

Out[12]:
```
Index(['card_id', 'auth_hist_months_diff_mean', 'auth_hist_months_diff_max',
       'auth_hist_months_diff_min', 'auth_hist_purchase_month_max',
       'auth_hist_purchase_month_min', 'auth_hist_purchase_month_mean',
       'auth_hist_purchase_month_std', 'auth_hist_purchase_year_mean',
       'auth_hist_purchase_year_max', 'auth_hist_purchase_year_min',
       'auth_hist_purchase_year_std', 'auth_hist_purchase_year_nunique',
       'auth_hist_weekofyear_mean', 'auth_hist_weekofyear_max',
       'auth_hist_weekofyear_min', 'auth_hist_weekofyear_nunique',
       'auth_hist_dayofweek_mean', 'auth_hist_weekend_sum',
       'auth_hist_weekend_mean', 'auth_hist_hour_mean', 'auth_hist_hour_max',
       'auth_hist_hour_min', 'auth_hist_pur_date_max',
       'auth_hist_pur_date_min', 'auth_hist_pur_date_ptp'],
      dtype='object')
```

In [13]:
```
X_train_auth_date.shape
```

Out[13]:
```
(325540, 26)
```

In [14]:
```
X_train_unauth_date.shape
```

Out[14]:
```
(274262, 26)
```

In [15]:
```
X_train_unauth_date.head(10)
```

Out[15]:

	card_id	unauth_hist_months_diff_mean	unauth_hist_months_diff_max	unauth_hist_m
0	C_ID_00007093c1	38.028571	43	
1	C_ID_0001238066	33.333333	34	
2	C_ID_0001506ef0	34.500000	40	
3	C_ID_0001793786	39.518519	43	
4	C_ID_000183fdda	37.714286	38	
5	C_ID_00024e244b	36.411765	42	
6	C_ID_0002709b5a	37.166667	45	
7	C_ID_00027503e2	37.444444	40	
8	C_ID_000298032a	39.500000	42	
9	C_ID_0002ba3c2e	38.600000	45	

10 rows × 26 columns

In [16]:
```
X_train_unauth_merge=X_train_unauth.merge(X_train_unauth_date,on='card_id',how='left')
```

In [17]:
```
X_train_auth_merge=X_train_history.merge(X_train_auth_date,on='card_id',how='left')
```

In [18]:
```
history=X_train_auth_merge.merge(X_train_unauth_merge,on='card_id',how='left')
```

In [19]:
```
new=X_train_new.merge(X_train_new_date,on='card_id',how='left')
```

In [20]:
```
test=test_csv.merge(history,on='card_id',how='left')
```

In [21]:
```
X_test=test.merge(new,on='card_id',how='left')
X_test.shape
```

Out[21]:
```
(123623, 305)
```

In [22]:
```
train=train_csv.merge(history,on='card_id',how='left')
```

In [23]:
```
X_train=train.merge(new,on='card_id',how='left')
X_train.shape
```

Out[23]:
```
(201917, 306)
```

In [24]:
```
X_train=X_train.sort_values(by=['first_active_month'])
```

In [25]:
```
# X_test=X_test.sort_values(by=['first_active_month'])
# X_test_card=X_test['card_id']
```

In [26]:
```
X_test=X_test.drop(columns=['first_active_month','card_id'])
X_test.shape
```

Out[26]:
```
(123623, 303)
```

In [ ]:
```

```

In [27]:
```
Y_train=X_train['target']
```

In [28]:
```
X_train=X_train.drop(columns=['first_active_month','card_id','target'])
X_train.shape
```

Out[28]:
```
(201917, 303)
```

In [34]:
```
def root_mean_squared_error(y_true, y_pred):
    """Root mean squared error regression loss"""
    return np.sqrt(np.mean(np.square(y_true-y_pred)))
```

In [35]:
```
X_train_init=X_train
Y_train_init=Y_train
```

In [ ]:
```
#mean model to check the RMSE and cosider it as baseline model
root_mean_squared_error(np.mean(Y_train), Y_train)
```

Out[ ]:
```
3.85049046061597
```

In [ ]:
```

```

In [36]:
```
X_train=X_train_init[:170000]
Y_train=Y_train_init[:170000]
X_CV=X_train_init[170000:]
Y_CV=Y_train_init[170000:]
```

In [78]:
```
#https://github.com/optuna/optuna/blob/master/examples/lightgbm_tuner_simple.py

import optuna.integration.lightgbm as lgb
```

In [79]:
```
#https://www.kaggle.com/aleksandrovich/smart-hyperparameter-tuning-with-optuna
import lightgbm as lgbm
from optuna.samplers import TPESampler
import optuna

sampler = TPESampler(seed=10) # for reproducibility
def objective(trial):

    # dtrain = lgb.Dataset(X_train, label=y_train)

    param = {
        'objective': 'regression',
        'metric': 'rmse',
        'verbosity': -1,
        'boosting_type': 'gbdt',
        # 'early_stopping_rounds': trial.suggest_int('early_stopping_rounds',
 10, 2000),
        'lambda_l1': trial.suggest_loguniform('lambda_l1', 1e-8, 10.0),
        'lambda_l2': trial.suggest_loguniform('lambda_l2', 1e-8, 10.0),
        'num_leaves': trial.suggest_int('num_leaves', 2, 512),
        # 'learning_rate': trial.suggest_loguniform('learning_rate', 1e-8, 1.
0),
        'learning_rate': 0.01,
        'n_estimators': trial.suggest_int('n_estimators', 700, 3000),
        'feature_fraction': trial.suggest_uniform('feature_fraction', 0.4, 1.0
),
        'bagging_fraction': trial.suggest_uniform('bagging_fraction', 0.4, 1.0
),
        'bagging_freq': trial.suggest_int('bagging_freq', 1, 7),
        'min_child_samples': trial.suggest_int('min_child_samples', 5, 100),
    }
    lgbm_regr = lgbm.LGBMRegressor(**param)
    gbm_2 = lgbm_regr.fit(X_train, Y_train, eval_set=[(X_CV, Y_CV)], verbose=F
alse)
    return root_mean_squared_error(Y_CV,gbm_2.predict(X_CV))

study = optuna.create_study(direction='minimize', sampler=sampler)
study.optimize(objective, n_trials=100)
```

```
[I 2020-10-31 20:02:04,907] A new study created in memory with name: no-name-
fe0979e4-a86f-4b50-80f8-eb9f05412ce9
[I 2020-10-31 20:12:37,048] Trial 0 finished with value: 1.4609999950803207 a
nd parameters: {'lambda_l1': 0.08747537025773001, 'lambda_l2': 1.537331564587
801e-08, 'num_leaves': 322, 'n_estimators': 1880, 'feature_fraction': 0.69910
42073815543, 'bagging_fraction': 0.5348779873185086, 'bagging_freq': 2, 'min_
child_samples': 13}. Best is trial 0 with value: 1.4609999950803207.
[I 2020-10-31 20:36:25,856] Trial 1 finished with value: 1.4668211769548478 a
nd parameters: {'lambda_l1': 2.4552467279949516e-06, 'lambda_l2': 2.357358394
2260753e-06, 'num_leaves': 504, 'n_estimators': 2252, 'feature_fraction': 0.8
899611011324, 'bagging_fraction': 0.6016429506787311, 'bagging_freq': 1, 'min
_child_samples': 67}. Best is trial 0 with value: 1.4609999950803207.
[I 2020-10-31 20:38:22,543] Trial 2 finished with value: 1.4319445797047732 a
nd parameters: {'lambda_l1': 1.8860355948710983e-08, 'lambda_l2': 0.791652265
7454635, 'num_leaves': 179, 'n_estimators': 777, 'feature_fraction': 0.478874
89144151374, 'bagging_fraction': 0.6482004246510716, 'bagging_freq': 7, 'min_
child_samples': 18}. Best is trial 2 with value: 1.4319445797047732.
[I 2020-10-31 20:41:31,773] Trial 3 finished with value: 1.443380601215659 an
d parameters: {'lambda_l1': 0.001799336090092687, 'lambda_l2': 4.402333369795
2007e-07, 'num_leaves': 91, 'n_estimators': 2528, 'feature_fraction': 0.43933
7959812867, 'bagging_fraction': 0.4338665124820424, 'bagging_freq': 7, 'min_c
hild_samples': 28}. Best is trial 2 with value: 1.4319445797047732.
[I 2020-10-31 20:45:16,068] Trial 4 finished with value: 1.4335915365782628 a
nd parameters: {'lambda_l1': 1.2767100129345944e-08, 'lambda_l2': 0.003217288
9695974236, 'num_leaves': 76, 'n_estimators': 1940, 'feature_fraction': 0.735
7890192737848, 'bagging_fraction': 0.601299787911922, 'bagging_freq': 7, 'min
_child_samples': 76}. Best is trial 2 with value: 1.4319445797047732.
[I 2020-10-31 20:51:09,643] Trial 5 finished with value: 1.430911628646777 an
d parameters: {'lambda_l1': 4.9169921588150913e-08, 'lambda_l2': 0.4757588906
948674, 'num_leaves': 137, 'n_estimators': 1496, 'feature_fraction': 0.728551
6935515461, 'bagging_fraction': 0.8915721974020412, 'bagging_freq': 1, 'min_c
hild_samples': 49}. Best is trial 5 with value: 1.430911628646777.
[I 2020-10-31 21:06:08,445] Trial 6 finished with value: 1.4505865424200604 a
nd parameters: {'lambda_l1': 0.5148168134404182, 'lambda_l2': 1.4617521957829
217e-05, 'num_leaves': 346, 'n_estimators': 2542, 'feature_fraction': 0.57757
70241278072, 'bagging_fraction': 0.9303618877367118, 'bagging_freq': 6, 'min_
child_samples': 93}. Best is trial 5 with value: 1.430911628646777.
[I 2020-10-31 21:11:36,608] Trial 7 finished with value: 1.4348601351194656 a
nd parameters: {'lambda_l1': 0.7615535600340646, 'lambda_l2': 4.6197480389018
12e-06, 'num_leaves': 87, 'n_estimators': 2247, 'feature_fraction': 0.7954217
823629728, 'bagging_fraction': 0.7572637629997463, 'bagging_freq': 7, 'min_ch
ild_samples': 62}. Best is trial 5 with value: 1.430911628646777.
[I 2020-10-31 21:14:38,071] Trial 8 finished with value: 1.4286956477606225 a
nd parameters: {'lambda_l1': 0.00012786183233975197, 'lambda_l2': 0.272336338
75662455, 'num_leaves': 44, 'n_estimators': 1447, 'feature_fraction': 0.94169
90561989764, 'bagging_fraction': 0.720734769281089, 'bagging_freq': 2, 'min_c
hild_samples': 78}. Best is trial 8 with value: 1.4286956477606225.
[I 2020-10-31 21:15:55,604] Trial 9 finished with value: 1.438286075019143 an
d parameters: {'lambda_l1': 2.2570225857344917e-08, 'lambda_l2': 1.6392146534
756105e-05, 'num_leaves': 5, 'n_estimators': 2062, 'feature_fraction': 0.7959
789683244987, 'bagging_fraction': 0.5289719151256645, 'bagging_freq': 4, 'min
_child_samples': 22}. Best is trial 8 with value: 1.4286956477606225.
[I 2020-10-31 21:31:36,726] Trial 10 finished with value: 1.446427251353658 a
nd parameters: {'lambda_l1': 1.823973518707282e-05, 'lambda_l2': 0.0145073628
83197464, 'num_leaves': 500, 'n_estimators': 1332, 'feature_fraction': 0.9612
434276883192, 'bagging_fraction': 0.7797298435189901, 'bagging_freq': 3, 'min
_child_samples': 84}. Best is trial 8 with value: 1.4286956477606225.
```

```
[I 2020-10-31 21:40:05,398] Trial 11 finished with value: 1.4312530284152587
and parameters: {'lambda_l1': 1.6899283783615504e-06, 'lambda_l2': 4.08484497
3005144, 'num_leaves': 186, 'n_estimators': 1327, 'feature_fraction': 0.99901
42538299961, 'bagging_fraction': 0.9416027539631071, 'bagging_freq': 1, 'min_
child_samples': 43}. Best is trial 8 with value: 1.4286956477606225.
[I 2020-10-31 21:41:54,296] Trial 12 finished with value: 1.427578843376055 a
nd parameters: {'lambda_l1': 0.0029369836308719127, 'lambda_l2': 0.1585745661
5116594, 'num_leaves': 34, 'n_estimators': 1292, 'feature_fraction': 0.607969
9494711887, 'bagging_fraction': 0.8339694119646435, 'bagging_freq': 3, 'min_c
hild_samples': 44}. Best is trial 12 with value: 1.427578843376055.
[I 2020-10-31 21:42:47,529] Trial 13 finished with value: 1.4314038204120672
and parameters: {'lambda_l1': 0.002796951752477185, 'lambda_l2': 0.0278984520
59306605, 'num_leaves': 14, 'n_estimators': 820, 'feature_fraction': 0.592728
7660530283, 'bagging_fraction': 0.828057396147716, 'bagging_freq': 4, 'min_ch
ild_samples': 100}. Best is trial 12 with value: 1.427578843376055.
[I 2020-10-31 21:43:29,958] Trial 14 finished with value: 1.4762006918903885
and parameters: {'lambda_l1': 6.638682857212565e-05, 'lambda_l2': 8.875366611
828468, 'num_leaves': 3, 'n_estimators': 1113, 'feature_fraction': 0.58558528
26878553, 'bagging_fraction': 0.7076478431960427, 'bagging_freq': 3, 'min_chi
ld_samples': 41}. Best is trial 12 with value: 1.427578843376055.
[I 2020-10-31 21:51:58,903] Trial 15 finished with value: 1.439077847055157 a
nd parameters: {'lambda_l1': 0.024543561340938446, 'lambda_l2': 0.00095232877
61254093, 'num_leaves': 280, 'n_estimators': 1616, 'feature_fraction': 0.6397
498331963148, 'bagging_fraction': 0.8129066850754364, 'bagging_freq': 3, 'min
_child_samples': 61}. Best is trial 12 with value: 1.427578843376055.
[I 2020-10-31 21:57:50,564] Trial 16 finished with value: 1.4377364587780654
and parameters: {'lambda_l1': 0.0007258550867812199, 'lambda_l2': 0.095026615
29081215, 'num_leaves': 422, 'n_estimators': 979, 'feature_fraction': 0.51003
25211544446, 'bagging_fraction': 0.8685778465093082, 'bagging_freq': 5, 'min_
child_samples': 33}. Best is trial 12 with value: 1.427578843376055.
[I 2020-10-31 22:13:57,562] Trial 17 finished with value: 1.4522295413557804
and parameters: {'lambda_l1': 0.017089902770790116, 'lambda_l2': 0.0001748991
9848570772, 'num_leaves': 213, 'n_estimators': 2966, 'feature_fraction': 0.86
90458430044257, 'bagging_fraction': 0.722794743324794, 'bagging_freq': 2, 'mi
n_child_samples': 76}. Best is trial 12 with value: 1.427578843376055.
[I 2020-10-31 22:17:17,168] Trial 18 finished with value: 1.427275300469121 a
nd parameters: {'lambda_l1': 9.72580679060692, 'lambda_l2': 0.392557520279669
17, 'num_leaves': 54, 'n_estimators': 1634, 'feature_fraction': 0.67335951352
45701, 'bagging_fraction': 0.9759761415288419, 'bagging_freq': 2, 'min_child_
samples': 6}. Best is trial 18 with value: 1.427275300469121.
[I 2020-10-31 22:22:06,637] Trial 19 finished with value: 1.4268345373077145
and parameters: {'lambda_l1': 9.571083770665211, 'lambda_l2': 4.3832012284972
11, 'num_leaves': 135, 'n_estimators': 1675, 'feature_fraction': 0.6552518087
761084, 'bagging_fraction': 0.96452674680318, 'bagging_freq': 5, 'min_child_s
amples': 9}. Best is trial 19 with value: 1.4268345373077145.
[I 2020-10-31 22:27:00,510] Trial 20 finished with value: 1.4242982892598202
and parameters: {'lambda_l1': 5.7449956757263925, 'lambda_l2': 9.578554064243
74, 'num_leaves': 135, 'n_estimators': 1657, 'feature_fraction': 0.6718134395
051014, 'bagging_fraction': 0.9525697517954351, 'bagging_freq': 5, 'min_child
_samples': 11}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 22:32:03,974] Trial 21 finished with value: 1.4262132505400045
and parameters: {'lambda_l1': 9.12410096532177, 'lambda_l2': 9.09284645839454
5, 'num_leaves': 140, 'n_estimators': 1674, 'feature_fraction': 0.67581043638
39874, 'bagging_fraction': 0.9967568316720953, 'bagging_freq': 5, 'min_child_
samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 22:36:35,834] Trial 22 finished with value: 1.425744091633663 a
nd parameters: {'lambda_l1': 8.970221537055883, 'lambda_l2': 6.96070735174820
```

```
5, 'num_leaves': 149, 'n_estimators': 1725, 'feature_fraction': 0.52995876900
00075, 'bagging_fraction': 0.9868054171864169, 'bagging_freq': 5, 'min_child_
samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 22:43:04,044] Trial 23 finished with value: 1.42953756651354 an
d parameters: {'lambda_l1': 1.8754954193896396, 'lambda_l2': 6.21901868431819
5, 'num_leaves': 239, 'n_estimators': 1737, 'feature_fraction': 0.53651212681
10027, 'bagging_fraction': 0.9964338719090418, 'bagging_freq': 5, 'min_child_
samples': 8}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 22:47:11,342] Trial 24 finished with value: 1.429818461623861 a
nd parameters: {'lambda_l1': 3.999629586678349, 'lambda_l2': 2.10370917362232
17, 'num_leaves': 134, 'n_estimators': 2079, 'feature_fraction': 0.4152131569
407965, 'bagging_fraction': 0.9994275274979981, 'bagging_freq': 6, 'min_child
_samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 22:54:19,461] Trial 25 finished with value: 1.4392001096079963
and parameters: {'lambda_l1': 0.21900766462516852, 'lambda_l2': 0.03597985266
241246, 'num_leaves': 166, 'n_estimators': 1813, 'feature_fraction': 0.778629
8162503951, 'bagging_fraction': 0.9086117402269681, 'bagging_freq': 6, 'min_c
hild_samples': 18}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 22:59:03,034] Trial 26 finished with value: 1.4273153483158383
and parameters: {'lambda_l1': 9.511322303164134, 'lambda_l2': 8.4498201266647
74, 'num_leaves': 259, 'n_estimators': 1161, 'feature_fraction': 0.5226290161
29251, 'bagging_fraction': 0.9929822155818331, 'bagging_freq': 5, 'min_child_
samples': 28}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:03:17,478] Trial 27 finished with value: 1.4285231320981946
and parameters: {'lambda_l1': 1.9967701022267215, 'lambda_l2': 1.973394895955
9906, 'num_leaves': 114, 'n_estimators': 1497, 'feature_fraction': 0.73983316
22266132, 'bagging_fraction': 0.864378936739222, 'bagging_freq': 6, 'min_chil
d_samples': 16}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:11:26,808] Trial 28 finished with value: 1.4384200610447688
and parameters: {'lambda_l1': 0.1585506276860628, 'lambda_l2': 1.199990803530
5484, 'num_leaves': 208, 'n_estimators': 2012, 'feature_fraction': 0.62723714
2903408, 'bagging_fraction': 0.9529616320391399, 'bagging_freq': 4, 'min_chil
d_samples': 24}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:22:07,238] Trial 29 finished with value: 1.454505052468954 a
nd parameters: {'lambda_l1': 0.039746171158817824, 'lambda_l2': 0.00445845374
4852102, 'num_leaves': 313, 'n_estimators': 2222, 'feature_fraction': 0.55927
73779876152, 'bagging_fraction': 0.9119739597271728, 'bagging_freq': 5, 'min_
child_samples': 12}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:28:20,731] Trial 30 finished with value: 1.4348977864766768
and parameters: {'lambda_l1': 0.6891884616603827, 'lambda_l2': 2.026369004812
8137e-07, 'num_leaves': 145, 'n_estimators': 1863, 'feature_fraction': 0.6956
399670546441, 'bagging_fraction': 0.8736437597146818, 'bagging_freq': 4, 'min
_child_samples': 34}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:32:40,792] Trial 31 finished with value: 1.4285962297821755
and parameters: {'lambda_l1': 6.579557999486956, 'lambda_l2': 1.8741483250309
274, 'num_leaves': 112, 'n_estimators': 1646, 'feature_fraction': 0.667960503
6267398, 'bagging_fraction': 0.9625008403285374, 'bagging_freq': 5, 'min_chil
d_samples': 11}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:38:16,746] Trial 32 finished with value: 1.4256576686263507
and parameters: {'lambda_l1': 8.968770820759108, 'lambda_l2': 9.3483236803642
3, 'num_leaves': 159, 'n_estimators': 1740, 'feature_fraction': 0.65870702853
9404, 'bagging_fraction': 0.9826230884048148, 'bagging_freq': 5, 'min_child_s
amples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:43:32,971] Trial 33 finished with value: 1.4256383370667483
and parameters: {'lambda_l1': 2.0027448510896173, 'lambda_l2': 8.177422734550
026, 'num_leaves': 201, 'n_estimators': 1851, 'feature_fraction': 0.467595548
3551797, 'bagging_fraction': 0.9930403982516227, 'bagging_freq': 6, 'min_chil
```

```
d_samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:49:23,581] Trial 34 finished with value: 1.4379478523163893
and parameters: {'lambda_l1': 1.683160160985869, 'lambda_l2': 0.8708938717173
37, 'num_leaves': 212, 'n_estimators': 1935, 'feature_fraction': 0.4555673619
6562336, 'bagging_fraction': 0.9383100361034222, 'bagging_freq': 6, 'min_chil
d_samples': 14}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-10-31 23:57:56,528] Trial 35 finished with value: 1.44460110615473 an
d parameters: {'lambda_l1': 0.08870418658021584, 'lambda_l2': 1.0485177933674
37e-08, 'num_leaves': 236, 'n_estimators': 2393, 'feature_fraction': 0.498293
2154633976, 'bagging_fraction': 0.9964901044669476, 'bagging_freq': 6, 'min_c
hild_samples': 20}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:02:03,760] Trial 36 finished with value: 1.4465897604258071
and parameters: {'lambda_l1': 0.33054509218803013, 'lambda_l2': 0.07827004329
076502, 'num_leaves': 170, 'n_estimators': 1780, 'feature_fraction': 0.467288
7268204935, 'bagging_fraction': 0.6565792760860447, 'bagging_freq': 6, 'min_c
hild_samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:05:41,073] Trial 37 finished with value: 1.425788067660745 a
nd parameters: {'lambda_l1': 3.217021091531724, 'lambda_l2': 9.26253164349907
5, 'num_leaves': 185, 'n_estimators': 1508, 'feature_fraction': 0.40765811814
947894, 'bagging_fraction': 0.9119064386314312, 'bagging_freq': 7, 'min_child
_samples': 15}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:08:31,772] Trial 38 finished with value: 1.4385610502911785
and parameters: {'lambda_l1': 0.7859656071029163, 'lambda_l2': 0.610014548151
4251, 'num_leaves': 72, 'n_estimators': 2133, 'feature_fraction': 0.557607487
3753112, 'bagging_fraction': 0.4296971750823183, 'bagging_freq': 5, 'min_chil
d_samples': 26}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:11:30,429] Trial 39 finished with value: 1.435628690687871 a
nd parameters: {'lambda_l1': 2.3575927230135851e-07, 'lambda_l2': 6.130958297
907373e-08, 'num_leaves': 105, 'n_estimators': 1910, 'feature_fraction': 0.43
74100470218472, 'bagging_fraction': 0.5693596880980438, 'bagging_freq': 4, 'm
in_child_samples': 32}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:25:10,337] Trial 40 finished with value: 1.446555200796948 a
nd parameters: {'lambda_l1': 0.01277770607079087, 'lambda_l2': 2.882134887894
467, 'num_leaves': 292, 'n_estimators': 2338, 'feature_fraction': 0.757049781
7887915, 'bagging_fraction': 0.8910853012777168, 'bagging_freq': 6, 'min_chil
d_samples': 11}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:29:34,797] Trial 41 finished with value: 1.42633544868111 an
d parameters: {'lambda_l1': 2.209349854613159, 'lambda_l2': 9.64453221140380
6, 'num_leaves': 190, 'n_estimators': 1516, 'feature_fraction': 0.49323358346
27426, 'bagging_fraction': 0.9212956589884319, 'bagging_freq': 7, 'min_child_
samples': 15}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:32:56,317] Trial 42 finished with value: 1.4304923573466142
and parameters: {'lambda_l1': 3.162379699610409, 'lambda_l2': 0.9439242705944
418, 'num_leaves': 176, 'n_estimators': 1422, 'feature_fraction': 0.409018582
7605527, 'bagging_fraction': 0.9669974788162495, 'bagging_freq': 7, 'min_chil
d_samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:37:51,198] Trial 43 finished with value: 1.4275138594415224
and parameters: {'lambda_l1': 0.8287983001153612, 'lambda_l2': 9.378207894364
005, 'num_leaves': 236, 'n_estimators': 1586, 'feature_fraction': 0.435806479
42310297, 'bagging_fraction': 0.9449142401590439, 'bagging_freq': 7, 'min_chi
ld_samples': 19}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:44:46,688] Trial 44 finished with value: 1.4357161631685642
and parameters: {'lambda_l1': 5.351360110792861, 'lambda_l2': 0.3531495299158
31, 'num_leaves': 153, 'n_estimators': 1772, 'feature_fraction': 0.8350309327
322465, 'bagging_fraction': 0.8992910519917587, 'bagging_freq': 5, 'min_child
_samples': 9}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:50:54,374] Trial 45 finished with value: 1.431636189827708 a
```

```
nd parameters: {'lambda_l1': 0.13920054980746532, 'lambda_l2': 3.666866084216
3865, 'num_leaves': 200, 'n_estimators': 1405, 'feature_fraction': 0.71492831
75992132, 'bagging_fraction': 0.9762072063196733, 'bagging_freq': 6, 'min_chi
ld_samples': 16}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:52:53,959] Trial 46 finished with value: 1.425866186370225 a
nd parameters: {'lambda_l1': 1.1149721611025456, 'lambda_l2': 9.8985479202647
73, 'num_leaves': 103, 'n_estimators': 1210, 'feature_fraction': 0.4054361966
9287606, 'bagging_fraction': 0.8494855364671939, 'bagging_freq': 7, 'min_chil
d_samples': 23}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 00:56:31,522] Trial 47 finished with value: 1.433436334857982 a
nd parameters: {'lambda_l1': 0.33244650329566433, 'lambda_l2': 0.145814119864
9376, 'num_leaves': 81, 'n_estimators': 1958, 'feature_fraction': 0.615667131
4895805, 'bagging_fraction': 0.7925734500872375, 'bagging_freq': 5, 'min_chil
d_samples': 13}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:00:27,920] Trial 48 finished with value: 1.4330043867577387
and parameters: {'lambda_l1': 0.05152008565492703, 'lambda_l2': 1.16042449315
86403, 'num_leaves': 162, 'n_estimators': 1557, 'feature_fraction': 0.4785729
3281494284, 'bagging_fraction': 0.9319679720929385, 'bagging_freq': 6, 'min_c
hild_samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:13:55,692] Trial 49 finished with value: 1.44501777365303 an
d parameters: {'lambda_l1': 4.176178931812134, 'lambda_l2': 5.258497922542522
e-05, 'num_leaves': 371, 'n_estimators': 1857, 'feature_fraction': 0.70005191
53393486, 'bagging_fraction': 0.8878231513873821, 'bagging_freq': 4, 'min_chi
ld_samples': 52}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:16:30,806] Trial 50 finished with value: 1.4285502785205202
and parameters: {'lambda_l1': 0.0076995835093218385, 'lambda_l2': 0.313516988
57926935, 'num_leaves': 123, 'n_estimators': 1051, 'feature_fraction': 0.5376
345219771018, 'bagging_fraction': 0.926656328669047, 'bagging_freq': 6, 'min_
child_samples': 22}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:18:34,493] Trial 51 finished with value: 1.4265182407354735
and parameters: {'lambda_l1': 1.2687018711998526, 'lambda_l2': 3.915658575736
0873, 'num_leaves': 94, 'n_estimators': 1331, 'feature_fraction': 0.400767889
0177631, 'bagging_fraction': 0.8445883357203423, 'bagging_freq': 7, 'min_chil
d_samples': 28}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:19:53,378] Trial 52 finished with value: 1.428382598005009 a
nd parameters: {'lambda_l1': 3.5126644402085794, 'lambda_l2': 9.4766960033696
18, 'num_leaves': 66, 'n_estimators': 1203, 'feature_fraction': 0.42863808872
148174, 'bagging_fraction': 0.4779565557771237, 'bagging_freq': 7, 'min_child
_samples': 9}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:20:49,325] Trial 53 finished with value: 1.428248750777495 a
nd parameters: {'lambda_l1': 0.5285552231615744, 'lambda_l2': 3.3730970484730
09, 'num_leaves': 32, 'n_estimators': 914, 'feature_fraction': 0.451543270481
1118, 'bagging_fraction': 0.7532967455523223, 'bagging_freq': 7, 'min_child_s
amples': 39}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:24:58,782] Trial 54 finished with value: 1.4257157966216445
and parameters: {'lambda_l1': 8.310517957124743, 'lambda_l2': 8.8745420886595
38, 'num_leaves': 179, 'n_estimators': 1735, 'feature_fraction': 0.4038886487
6683217, 'bagging_fraction': 0.981183786450221, 'bagging_freq': 7, 'min_child
_samples': 23}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:31:04,022] Trial 55 finished with value: 1.4312248678036052
and parameters: {'lambda_l1': 9.865656752481904, 'lambda_l2': 1.6205798438109
156, 'num_leaves': 191, 'n_estimators': 1709, 'feature_fraction': 0.593372578
1576219, 'bagging_fraction': 0.9738539362287023, 'bagging_freq': 5, 'min_chil
d_samples': 17}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:38:58,688] Trial 56 finished with value: 1.4304605921564306
and parameters: {'lambda_l1': 5.115571653582676, 'lambda_l2': 5.5702071533468
08, 'num_leaves': 218, 'n_estimators': 2009, 'feature_fraction': 0.6340737354
```

```
904822, 'bagging_fraction': 0.9870212214211326, 'bagging_freq': 6, 'min_child
_samples': 8}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:45:07,748] Trial 57 finished with value: 1.4324580365383657
and parameters: {'lambda_l1': 5.95352103154981e-06, 'lambda_l2': 9.8521833840
39302, 'num_leaves': 251, 'n_estimators': 1581, 'feature_fraction': 0.4808042
241423719, 'bagging_fraction': 0.95033954855184, 'bagging_freq': 5, 'min_chil
d_samples': 60}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:49:25,653] Trial 58 finished with value: 1.4284834733254421
and parameters: {'lambda_l1': 9.060287651343252, 'lambda_l2': 0.5887963708803
396, 'num_leaves': 157, 'n_estimators': 1454, 'feature_fraction': 0.559585120
9725895, 'bagging_fraction': 0.9937690057665673, 'bagging_freq': 7, 'min_chil
d_samples': 21}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 01:53:30,766] Trial 59 finished with value: 1.4321961571307746
and parameters: {'lambda_l1': 2.629109329943141, 'lambda_l2': 2.4951936272784
01e-06, 'num_leaves': 128, 'n_estimators': 1720, 'feature_fraction': 0.509323
3150964822, 'bagging_fraction': 0.9577854298415345, 'bagging_freq': 6, 'min_c
hild_samples': 12}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:02:31,192] Trial 60 finished with value: 1.4494753654687276
and parameters: {'lambda_l1': 0.40311145131063936, 'lambda_l2': 0.01206973804
50903, 'num_leaves': 275, 'n_estimators': 1808, 'feature_fraction': 0.6483616
724805534, 'bagging_fraction': 0.9195602811659588, 'bagging_freq': 5, 'min_ch
ild_samples': 7}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:05:39,182] Trial 61 finished with value: 1.4291505876385064
and parameters: {'lambda_l1': 1.3180834167465079, 'lambda_l2': 5.372071509455
767, 'num_leaves': 179, 'n_estimators': 1273, 'feature_fraction': 0.419455797
1826436, 'bagging_fraction': 0.852901546889485, 'bagging_freq': 7, 'min_child
_samples': 31}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:08:31,464] Trial 62 finished with value: 1.4296451935601624
and parameters: {'lambda_l1': 1.0958399089467694, 'lambda_l2': 2.180472077503
434, 'num_leaves': 147, 'n_estimators': 1376, 'feature_fraction': 0.401367453
8839285, 'bagging_fraction': 0.8815143209793763, 'bagging_freq': 7, 'min_chil
d_samples': 25}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:11:08,560] Trial 63 finished with value: 1.4266660333734513
and parameters: {'lambda_l1': 9.795850242893072, 'lambda_l2': 9.8361282308800
34, 'num_leaves': 98, 'n_estimators': 1495, 'feature_fraction': 0.45967822089
90302, 'bagging_fraction': 0.8134078866275495, 'bagging_freq': 7, 'min_child_
samples': 38}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:14:56,455] Trial 64 finished with value: 1.4272738733898653
and parameters: {'lambda_l1': 4.84010466053813, 'lambda_l2': 5.36509661856398
2, 'num_leaves': 120, 'n_estimators': 2133, 'feature_fraction': 0.40374945619
437264, 'bagging_fraction': 0.9052794630581681, 'bagging_freq': 7, 'min_child
_samples': 23}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:19:59,364] Trial 65 finished with value: 1.4324347708050362
and parameters: {'lambda_l1': 2.341400773122735, 'lambda_l2': 2.5936957638433
5, 'num_leaves': 218, 'n_estimators': 1698, 'feature_fraction': 0.43831565754
09628, 'bagging_fraction': 0.9431990266911658, 'bagging_freq': 6, 'min_child_
samples': 18}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:25:35,596] Trial 66 finished with value: 1.4324411461286068
and parameters: {'lambda_l1': 9.96853160247204, 'lambda_l2': 0.17827514577835
002, 'num_leaves': 191, 'n_estimators': 1623, 'feature_fraction': 0.528239007
3119388, 'bagging_fraction': 0.9998571638971633, 'bagging_freq': 4, 'min_chil
d_samples': 47}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:30:27,595] Trial 67 finished with value: 1.4333348687324896
and parameters: {'lambda_l1': 0.00037426351084976505, 'lambda_l2': 1.24401717
05262784, 'num_leaves': 166, 'n_estimators': 1823, 'feature_fraction': 0.4794
22515427525, 'bagging_fraction': 0.9722529601104782, 'bagging_freq': 6, 'min_
child_samples': 14}. Best is trial 20 with value: 1.4242982892598202.
```

```
[I 2020-11-01 02:34:23,087] Trial 68 finished with value: 1.434142530915861 a
nd parameters: {'lambda_l1': 1.2007879494168543, 'lambda_l2': 0.5971600043473
61, 'num_leaves': 135, 'n_estimators': 1891, 'feature_fraction': 0.4199545788
390761, 'bagging_fraction': 0.9821018784430606, 'bagging_freq': 5, 'min_child
_samples': 10}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:40:09,374] Trial 69 finished with value: 1.4292517118080519
and parameters: {'lambda_l1': 0.15576086691157373, 'lambda_l2': 9.49345263792
8144, 'num_leaves': 222, 'n_estimators': 1248, 'feature_fraction': 0.69504845
08852001, 'bagging_fraction': 0.8625443458839017, 'bagging_freq': 4, 'min_chi
ld_samples': 36}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:43:00,754] Trial 70 finished with value: 1.4296393378714236
and parameters: {'lambda_l1': 0.5893667297161673, 'lambda_l2': 5.451066706915
07, 'num_leaves': 57, 'n_estimators': 1998, 'feature_fraction': 0.60785313531
67408, 'bagging_fraction': 0.6499382010961963, 'bagging_freq': 7, 'min_child_
samples': 28}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:45:26,778] Trial 71 finished with value: 1.42515000710314 an
d parameters: {'lambda_l1': 6.22076389892138, 'lambda_l2': 9.877786690112442,
'num_leaves': 157, 'n_estimators': 725, 'feature_fraction': 0.660774919228552
8, 'bagging_fraction': 0.949616131835897, 'bagging_freq': 5, 'min_child_sampl
es': 6}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:47:49,289] Trial 72 finished with value: 1.4257498644861524
and parameters: {'lambda_l1': 6.483354128680242, 'lambda_l2': 2.2736618280458
75, 'num_leaves': 144, 'n_estimators': 754, 'feature_fraction': 0.66071045684
90126, 'bagging_fraction': 0.9560828438430979, 'bagging_freq': 5, 'min_child_
samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:50:54,035] Trial 73 finished with value: 1.4275392499399628
and parameters: {'lambda_l1': 6.145169223922325, 'lambda_l2': 2.1740762123819
377, 'num_leaves': 200, 'n_estimators': 758, 'feature_fraction': 0.6661737796
451404, 'bagging_fraction': 0.9523503733232168, 'bagging_freq': 5, 'min_child
_samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 02:53:45,676] Trial 74 finished with value: 1.4266834534586146
and parameters: {'lambda_l1': 2.62028520047614, 'lambda_l2': 4.02574497446202
9, 'num_leaves': 178, 'n_estimators': 706, 'feature_fraction': 0.726381082898
4998, 'bagging_fraction': 0.9996934118086286, 'bagging_freq': 5, 'min_child_s
amples': 7}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:04:20,357] Trial 75 finished with value: 1.4442464311547132
and parameters: {'lambda_l1': 6.241774749607406, 'lambda_l2': 0.0005109959331
639752, 'num_leaves': 155, 'n_estimators': 2951, 'feature_fraction': 0.760656
4816462964, 'bagging_fraction': 0.9295298967890552, 'bagging_freq': 5, 'min_c
hild_samples': 12}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:07:24,052] Trial 76 finished with value: 1.4275784051879388
and parameters: {'lambda_l1': 3.5067127301863152, 'lambda_l2': 1.389492308663
3382, 'num_leaves': 139, 'n_estimators': 922, 'feature_fraction': 0.676826654
762493, 'bagging_fraction': 0.962551185275405, 'bagging_freq': 4, 'min_child_
samples': 15}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:17:20,358] Trial 77 finished with value: 1.437052916954404 a
nd parameters: {'lambda_l1': 8.51512661636134, 'lambda_l2': 0.812592363963354
2, 'num_leaves': 169, 'n_estimators': 2706, 'feature_fraction': 0.63684925453
78607, 'bagging_fraction': 0.9786933810560163, 'bagging_freq': 5, 'min_child_
samples': 89}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:20:21,405] Trial 78 finished with value: 1.4256568448745173
and parameters: {'lambda_l1': 1.870827725588051, 'lambda_l2': 6.2765166019310
95, 'num_leaves': 201, 'n_estimators': 858, 'feature_fraction': 0.57928063020
24711, 'bagging_fraction': 0.9033146172724266, 'bagging_freq': 3, 'min_child_
samples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:23:28,790] Trial 79 finished with value: 1.427419988578502 a
nd parameters: {'lambda_l1': 1.615849166865128, 'lambda_l2': 3.40053630657060
```

```
9, 'num_leaves': 201, 'n_estimators': 844, 'feature_fraction': 0.578991633722
1451, 'bagging_fraction': 0.9415909334305266, 'bagging_freq': 3, 'min_child_s
amples': 5}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:25:32,916] Trial 80 finished with value: 1.4256211348872292
and parameters: {'lambda_l1': 4.721522142283085e-05, 'lambda_l2': 5.686301809
106458, 'num_leaves': 113, 'n_estimators': 760, 'feature_fraction': 0.6107708
810072745, 'bagging_fraction': 0.9599398120477202, 'bagging_freq': 3, 'min_ch
ild_samples': 10}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:27:37,060] Trial 81 finished with value: 1.4255420643050003
and parameters: {'lambda_l1': 1.416126943935801e-05, 'lambda_l2': 5.763527763
183969, 'num_leaves': 118, 'n_estimators': 730, 'feature_fraction': 0.6165825
163550365, 'bagging_fraction': 0.9997997754499219, 'bagging_freq': 3, 'min_ch
ild_samples': 9}. Best is trial 20 with value: 1.4242982892598202.
[I 2020-11-01 03:29:51,398] Trial 82 finished with value: 1.424188560336277 a
nd parameters: {'lambda_l1': 0.00011695397311687352, 'lambda_l2': 6.215236198
348551, 'num_leaves': 114, 'n_estimators': 827, 'feature_fraction': 0.6016720
073147184, 'bagging_fraction': 0.984776963426634, 'bagging_freq': 3, 'min_chi
ld_samples': 10}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:31:52,444] Trial 83 finished with value: 1.4251386042002476
and parameters: {'lambda_l1': 5.836288655882512e-05, 'lambda_l2': 6.078652791
789714, 'num_leaves': 85, 'n_estimators': 830, 'feature_fraction': 0.61990601
04863096, 'bagging_fraction': 0.968697998567494, 'bagging_freq': 3, 'min_chil
d_samples': 10}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:33:45,357] Trial 84 finished with value: 1.4266121105916452
and parameters: {'lambda_l1': 5.661436222745518e-05, 'lambda_l2': 5.865685424
873214, 'num_leaves': 109, 'n_estimators': 702, 'feature_fraction': 0.6288505
873782567, 'bagging_fraction': 0.9320980923034771, 'bagging_freq': 3, 'min_ch
ild_samples': 10}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:35:41,005] Trial 85 finished with value: 1.426334203094542 a
nd parameters: {'lambda_l1': 2.375547037219001e-05, 'lambda_l2': 1.4228819155
506252, 'num_leaves': 84, 'n_estimators': 829, 'feature_fraction': 0.60805288
24462659, 'bagging_fraction': 0.9685333950311643, 'bagging_freq': 3, 'min_chi
ld_samples': 9}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:37:21,111] Trial 86 finished with value: 1.4263047648765843
and parameters: {'lambda_l1': 0.00013008771379737443, 'lambda_l2': 3.17735927
8602988, 'num_leaves': 36, 'n_estimators': 1036, 'feature_fraction': 0.592770
521630029, 'bagging_fraction': 0.900741131809545, 'bagging_freq': 2, 'min_chi
ld_samples': 13}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:39:56,364] Trial 87 finished with value: 1.425133430859495 a
nd parameters: {'lambda_l1': 4.103206455838544e-06, 'lambda_l2': 5.9000314593
96768, 'num_leaves': 129, 'n_estimators': 886, 'feature_fraction': 0.64884825
57082215, 'bagging_fraction': 0.9184345755937553, 'bagging_freq': 3, 'min_chi
ld_samples': 8}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:42:31,610] Trial 88 finished with value: 1.4300392349818187
and parameters: {'lambda_l1': 3.5457850351712316e-06, 'lambda_l2': 0.38369759
764584926, 'num_leaves': 118, 'n_estimators': 907, 'feature_fraction': 0.6211
518039890161, 'bagging_fraction': 0.9105271749246644, 'bagging_freq': 3, 'min
_child_samples': 8}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:44:49,498] Trial 89 finished with value: 1.4275526352406587
and parameters: {'lambda_l1': 7.85262211446973e-07, 'lambda_l2': 0.9473716266
113896, 'num_leaves': 93, 'n_estimators': 1015, 'feature_fraction': 0.5648704
938617085, 'bagging_fraction': 0.9213393131632344, 'bagging_freq': 3, 'min_ch
ild_samples': 18}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:47:40,391] Trial 90 finished with value: 1.4286365029880004
and parameters: {'lambda_l1': 2.018101384762861e-05, 'lambda_l2': 5.921819220
051915, 'num_leaves': 128, 'n_estimators': 867, 'feature_fraction': 0.6483298
517259302, 'bagging_fraction': 0.8732336209953923, 'bagging_freq': 2, 'min_ch
```

```
ild_samples': 70}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:49:27,571] Trial 91 finished with value: 1.4268896237049356
and parameters: {'lambda_l1': 3.81141260489144e-05, 'lambda_l2': 3.5972342287
517796, 'num_leaves': 62, 'n_estimators': 798, 'feature_fraction': 0.68713306
76222557, 'bagging_fraction': 0.9879682833635975, 'bagging_freq': 3, 'min_chi
ld_samples': 11}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:51:29,739] Trial 92 finished with value: 1.4250663016955327
and parameters: {'lambda_l1': 0.00015239218596527058, 'lambda_l2': 6.70502469
7403156, 'num_leaves': 78, 'n_estimators': 967, 'feature_fraction': 0.6029103
405379808, 'bagging_fraction': 0.9413777133249535, 'bagging_freq': 3, 'min_ch
ild_samples': 7}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:53:07,678] Trial 93 finished with value: 1.426940508612177 a
nd parameters: {'lambda_l1': 8.752703940182468e-06, 'lambda_l2': 1.9327836951
479878, 'num_leaves': 78, 'n_estimators': 709, 'feature_fraction': 0.60532357
92769729, 'bagging_fraction': 0.9418904492202795, 'bagging_freq': 3, 'min_chi
ld_samples': 7}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:54:57,231] Trial 94 finished with value: 1.4324621534950952
and parameters: {'lambda_l1': 0.00021987451494715832, 'lambda_l2': 6.65261465
9668428, 'num_leaves': 104, 'n_estimators': 1100, 'feature_fraction': 0.54749
82046382516, 'bagging_fraction': 0.4043311337892712, 'bagging_freq': 3, 'min_
child_samples': 13}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:56:48,902] Trial 95 finished with value: 1.4291857181488103
and parameters: {'lambda_l1': 8.771904883762342e-05, 'lambda_l2': 0.228698637
00710183, 'num_leaves': 89, 'n_estimators': 968, 'feature_fraction': 0.586591
927044404, 'bagging_fraction': 0.6708219109985938, 'bagging_freq': 3, 'min_ch
ild_samples': 10}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 03:58:12,385] Trial 96 finished with value: 1.4284748709013013
and parameters: {'lambda_l1': 0.0010060327305353527, 'lambda_l2': 4.180014335
598369, 'num_leaves': 21, 'n_estimators': 971, 'feature_fraction': 0.56978429
30795599, 'bagging_fraction': 0.9593571837078487, 'bagging_freq': 2, 'min_chi
ld_samples': 16}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 04:00:32,961] Trial 97 finished with value: 1.4276655962111122
and parameters: {'lambda_l1': 1.240898859002182e-06, 'lambda_l2': 2.644463745
7462794, 'num_leaves': 113, 'n_estimators': 770, 'feature_fraction': 0.713921
7472860938, 'bagging_fraction': 0.8936626030954395, 'bagging_freq': 3, 'min_c
hild_samples': 20}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 04:02:24,101] Trial 98 finished with value: 1.4262981901324454
and parameters: {'lambda_l1': 9.681232479072655e-06, 'lambda_l2': 7.242607829
233308, 'num_leaves': 48, 'n_estimators': 899, 'feature_fraction': 0.61943406
19016283, 'bagging_fraction': 0.9995504796848216, 'bagging_freq': 2, 'min_chi
ld_samples': 8}. Best is trial 82 with value: 1.424188560336277.
[I 2020-11-01 04:04:43,455] Trial 99 finished with value: 1.4270404300358885
and parameters: {'lambda_l1': 0.00037634052928480123, 'lambda_l2': 0.74076377
62691383, 'num_leaves': 68, 'n_estimators': 1129, 'feature_fraction': 0.64693
62244867513, 'bagging_fraction': 0.9196039229239518, 'bagging_freq': 3, 'min_
child_samples': 11}. Best is trial 82 with value: 1.424188560336277.
```

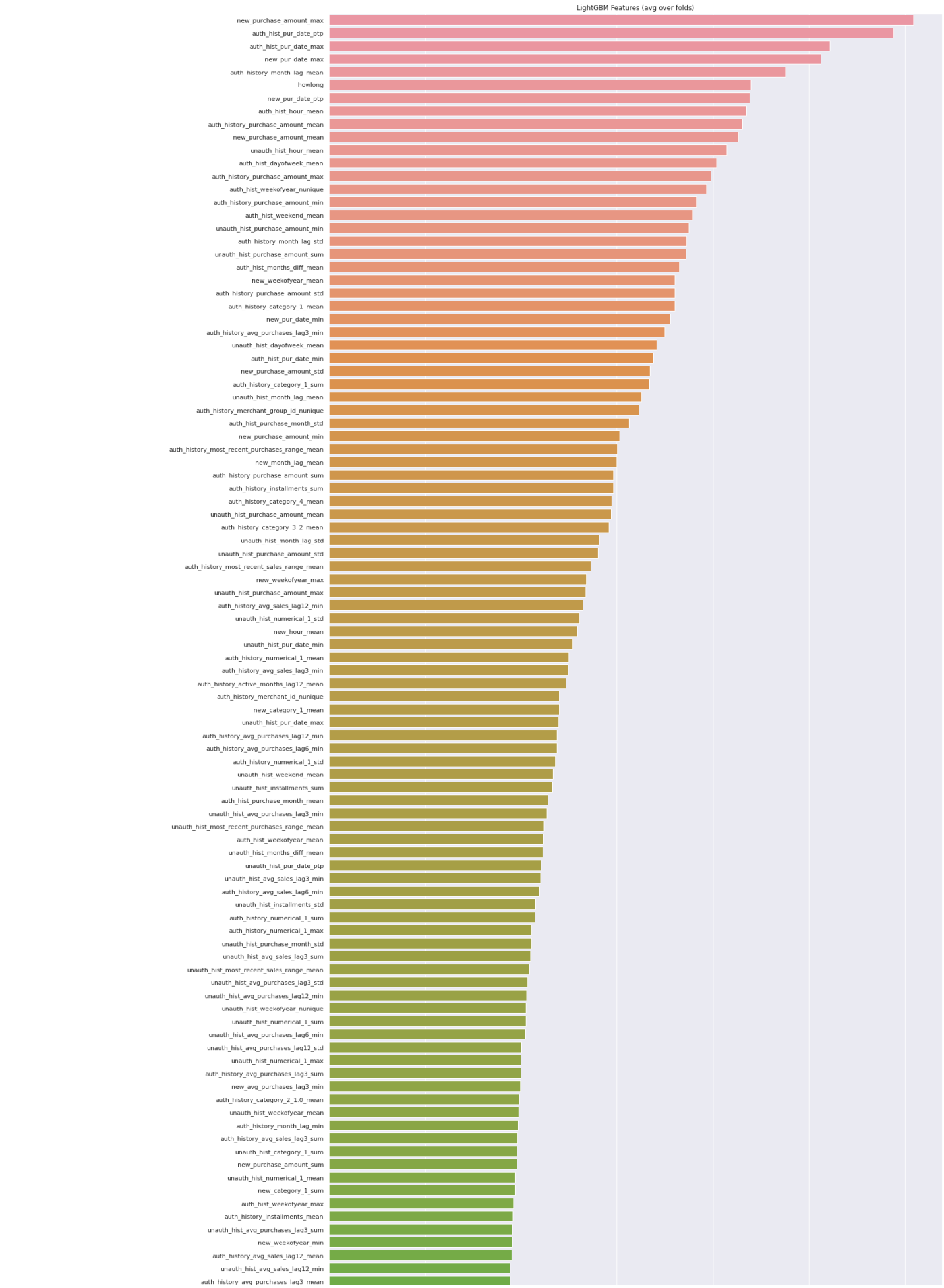
Now Train the model with the parameters

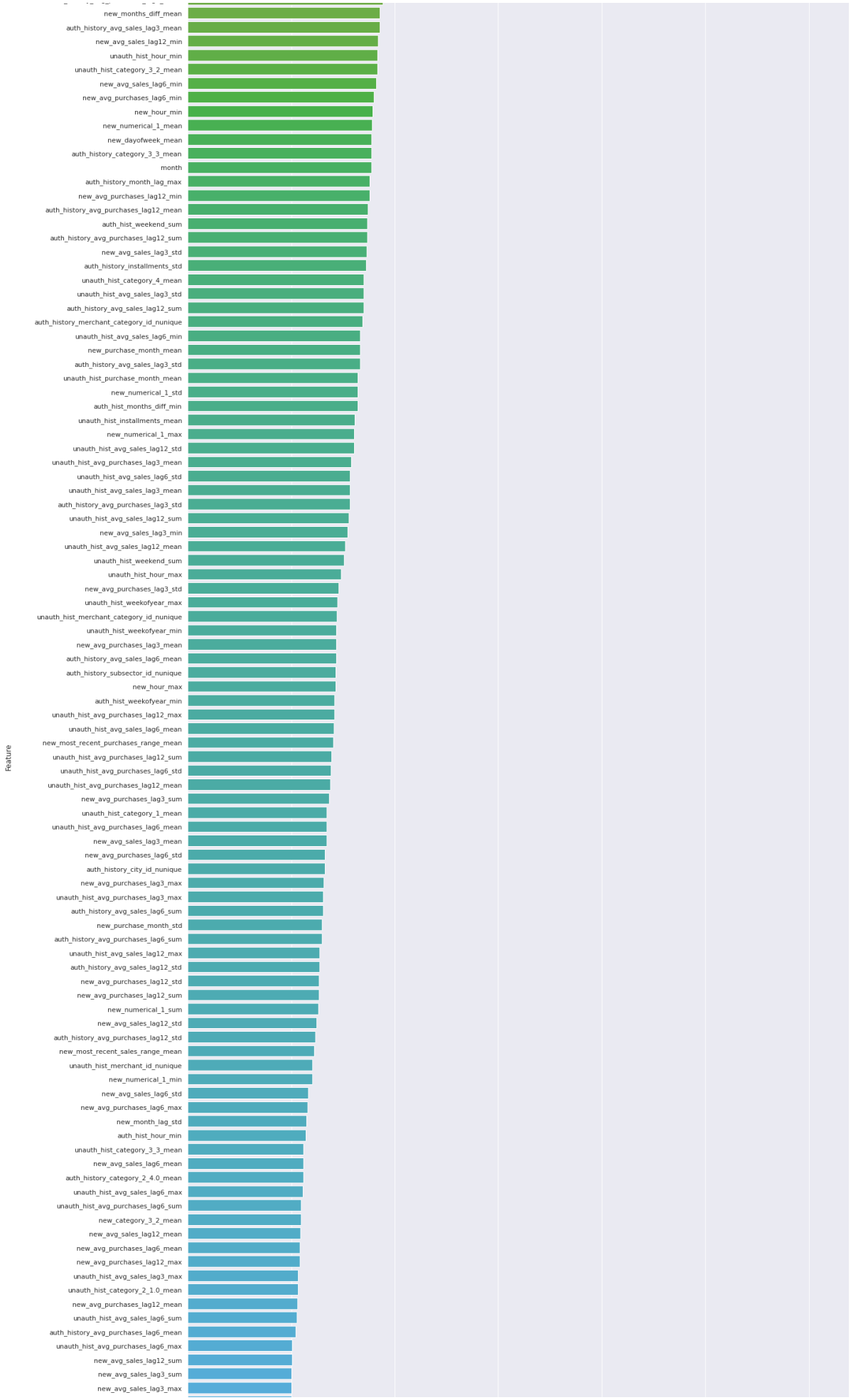
In [40]:
```
import lightgbm as lgbm

model = lgbm.LGBMRegressor(objective='regression',metric='rmse',
                           learning_rate=0.01,boosting_type='gbdt',lambda_l1=
0.00011695397311687352,
                           lambda_l2=6.215236198348551, num_leaves=114,n_estim
ators=827,
                           feature_fraction=0.6016720073147184,
                           bagging_fraction=0.984776963426634, bagging_freq=3,
min_child_samples=10

                          )
model.fit(X_train,Y_train)
```

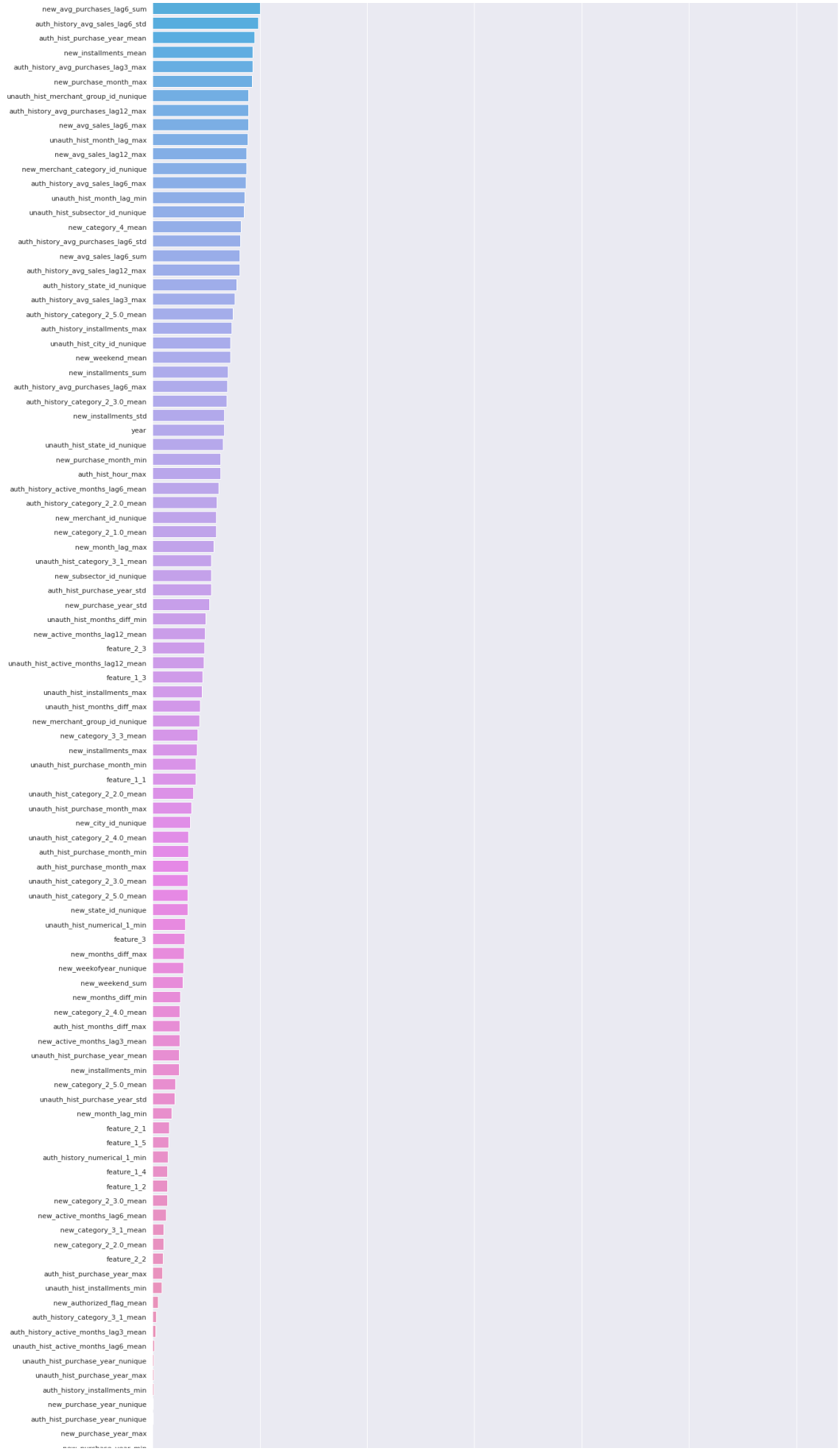
Out[40]:
```
LGBMRegressor(bagging_fraction=0.984776963426634, bagging_freq=3,
              feature_fraction=0.6016720073147184,
              lambda_l1=0.00011695397311687352, lambda_l2=6.215236198348551,
              learning_rate=0.01, metric='rmse', min_child_samples=10,
              n_estimators=827, num_leaves=114, objective='regression')
```

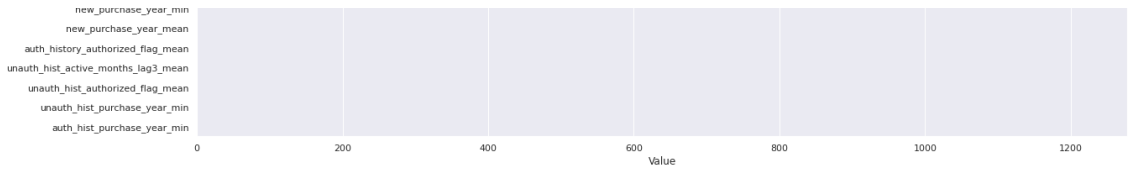
Feature Importance

In [41]:
```
#https://www.kaggle.com/ashishpatel26/feature-importance-of-lightgbm
sns.set(rc={'figure.figsize':(20,100)})
feature_imp = pd.DataFrame(sorted(zip(model.feature_importances_,X_train.colum
ns)), columns=['Value','Feature'])

#plt.figure(figsize=(20, 10))
sns.barplot(x="Value", y="Feature", data=feature_imp.sort_values(by="Value", a
scending=False))
plt.title('LightGBM Features (avg over folds)')
plt.tight_layout()
plt.show()
```


LightGBM Features (avg over folds)
new_purchase_amount_max
auth_hist_pur_date_ptp
auth_hist_pur_date_max
new_pur_date_max
auth_history_month_lag_mean
howlong
new_pur_date_ptp
auth_hist_hour_mean
auth_history_purchase_amount_mean
new_purchase_amount_mean
unauth_hist_hour_mean
auth_hist_dayofweek_mean
auth_history_purchase_amount_max
auth_hist_weekofyear_nunique
auth_history_purchase_amount_min
auth_hist_weekend_mean
unauth_hist_purchase_amount_min
auth_history_month_lag_std
unauth_hist_purchase_amount_sum
auth_hist_months_diff_mean
new_weekofyear_mean
auth_history_purchase_amount_std
auth_history_category_1_mean
new_pur_date_min
auth_history_avg_purchases_lag3_min
unauth_hist_dayofweek_mean
auth_hist_pur_date_min
new_purchase_amount_std
auth_history_category_1_sum
unauth_hist_month_lag_mean
auth_history_merchant_group_id_nunique
auth_hist_purchase_month_std
new_purchase_amount_min
auth_history_most_recent_purchases_range_mean
new_month_lag_mean
auth_history_purchase_amount_sum
auth_history_installments_sum
auth_history_category_4_mean
unauth_hist_purchase_amount_mean
auth_history_category_3_2_mean
unauth_hist_month_lag_std
unauth_hist_purchase_amount_std
auth_history_most_recent_sales_range_mean
new_weekofyear_max
unauth_hist_purchase_amount_max
auth_history_avg_sales_lag12_min
unauth_hist_numerical_1_std
new_hour_mean
unauth_hist_pur_date_min
auth_history_numerical_1_mean
auth_history_avg_sales_lag3_min
auth_history_active_months_lag12_mean
auth_history_merchant_id_nunique
new_category_1_mean
unauth_hist_pur_date_max
auth_history_avg_purchases_lag12_min
auth_history_avg_purchases_lag6_min
auth_history_numerical_1_std
unauth_hist_weekend_mean
unauth_hist_installments_sum
auth_hist_purchase_month_mean
unauth_hist_avg_purchases_lag3_min
unauth_hist_most_recent_purchases_range_mean
auth_hist_weekofyear_mean
unauth_hist_months_diff_mean
unauth_hist_pur_date_ptp
unauth_hist_avg_sales_lag3_min
auth_history_avg_sales_lag6_min
unauth_hist_installments_std
auth_history_numerical_1_sum
auth_history_numerical_1_max
unauth_hist_purchase_month_std
unauth_hist_avg_sales_lag3_sum
unauth_hist_most_recent_sales_range_mean
unauth_hist_avg_purchases_lag3_std
unauth_hist_avg_purchases_lag12_min
unauth_hist_weekofyear_nunique
unauth_hist_numerical_1_sum
unauth_hist_avg_purchases_lag6_min
unauth_hist_avg_purchases_lag12_std
unauth_hist_numerical_1_max
auth_history_avg_purchases_lag3_sum
new_avg_purchases_lag3_min
auth_history_category_2_1.0_mean
unauth_hist_weekofyear_mean
auth_history_month_lag_min
auth_history_avg_sales_lag3_sum
unauth_hist_category_1_sum
new_purchase_amount_sum
unauth_hist_numerical_1_mean
new_category_1_sum
auth_hist_weekofyear_max
auth_history_installments_mean
unauth_hist_avg_purchases_lag3_sum
new_weekofyear_min
auth_history_avg_sales_lag12_mean
unauth_hist_avg_sales_lag12_min
auth_history_avg_purchases_lag3_mean


Feature
new_months_diff_mean
auth_history_avg_sales_lag3_mean
new_avg_sales_lag12_min
unauth_hist_hour_min
unauth_hist_category_3_2_mean
new_avg_sales_lag6_min
new_avg_purchases_lag6_min
new_hour_min
new_numerical_1_mean
new_dayofweek_mean
auth_history_category_3_3_mean
month
auth_history_month_lag_max
new_avg_purchases_lag12_min
auth_history_avg_purchases_lag12_mean
auth_hist_weekend_sum
auth_history_avg_purchases_lag12_sum
new_avg_sales_lag3_std
auth_history_installments_std
unauth_hist_category_4_mean
unauth_hist_avg_sales_lag3_std
auth_history_avg_sales_lag12_sum
auth_history_merchant_category_id_nunique
unauth_hist_avg_sales_lag6_min
new_purchase_month_mean
auth_history_avg_sales_lag3_std
unauth_hist_purchase_month_mean
new_numerical_1_std
auth_hist_months_diff_min
unauth_hist_installments_mean
new_numerical_1_max
unauth_hist_avg_sales_lag12_std
unauth_hist_avg_purchases_lag3_mean
unauth_hist_avg_sales_lag6_std
unauth_hist_avg_sales_lag3_mean
auth_history_avg_purchases_lag3_std
unauth_hist_avg_sales_lag12_sum
new_avg_sales_lag3_min
unauth_hist_avg_sales_lag12_mean
unauth_hist_weekend_sum
unauth_hist_hour_max
new_avg_purchases_lag3_std
unauth_hist_weekofyear_max
unauth_hist_merchant_category_id_nunique
unauth_hist_weekofyear_min
new_avg_purchases_lag3_mean
auth_history_avg_sales_lag6_mean
auth_history_subsector_id_nunique
new_hour_max
auth_hist_weekofyear_min
unauth_hist_avg_purchases_lag12_max
unauth_hist_avg_sales_lag6_mean
new_most_recent_purchases_range_mean
unauth_hist_avg_purchases_lag12_sum
unauth_hist_avg_purchases_lag6_std
unauth_hist_avg_purchases_lag12_mean
new_avg_purchases_lag3_sum
unauth_hist_category_1_mean
unauth_hist_avg_purchases_lag6_mean
new_avg_sales_lag3_mean
new_avg_purchases_lag6_std
auth_history_city_id_nunique
new_avg_purchases_lag3_max
unauth_hist_avg_purchases_lag3_max
auth_history_avg_sales_lag6_sum
new_purchase_month_std
auth_history_avg_purchases_lag6_sum
unauth_hist_avg_sales_lag12_max
auth_history_avg_sales_lag12_std
new_avg_purchases_lag12_std
new_avg_purchases_lag12_sum
new_numerical_1_sum
new_avg_sales_lag12_std
auth_history_avg_purchases_lag12_std
new_most_recent_sales_range_mean
unauth_hist_merchant_id_nunique
new_numerical_1_min
new_avg_sales_lag6_std
new_avg_purchases_lag6_max
new_month_lag_std
auth_hist_hour_min
unauth_hist_category_3_3_mean
new_avg_sales_lag6_mean
auth_history_category_2_4.0_mean
unauth_hist_avg_sales_lag6_max
unauth_hist_avg_purchases_lag6_sum
new_category_3_2_mean
new_avg_sales_lag12_mean
new_avg_purchases_lag6_mean
new_avg_purchases_lag12_max
unauth_hist_avg_sales_lag3_max
unauth_hist_category_2_1.0_mean
new_avg_purchases_lag12_mean
unauth_hist_avg_sales_lag6_sum
auth_history_avg_purchases_lag6_mean
unauth_hist_avg_purchases_lag6_max
new_avg_sales_lag12_sum
new_avg_sales_lag3_sum
new_avg_sales_lag3_max


new_avg_purchases_lag6_sum
auth_history_avg_sales_lag6_std
auth_hist_purchase_year_mean
new_installments_mean
auth_history_avg_purchases_lag3_max
new_purchase_month_max
unauth_hist_merchant_group_id_nunique
auth_history_avg_purchases_lag12_max
new_avg_sales_lag6_max
unauth_hist_month_lag_max
new_avg_sales_lag12_max
new_merchant_category_id_nunique
auth_history_avg_sales_lag6_max
unauth_hist_month_lag_min
unauth_hist_subsector_id_nunique
new_category_4_mean
auth_history_avg_purchases_lag6_std
new_avg_sales_lag6_sum
auth_history_avg_sales_lag12_max
auth_history_state_id_nunique
auth_history_avg_sales_lag3_max
auth_history_category_2_5.0_mean
auth_history_installments_max
unauth_hist_city_id_nunique
new_weekend_mean
new_installments_sum
auth_history_avg_purchases_lag6_max
auth_history_category_2_3.0_mean
new_installments_std
year
unauth_hist_state_id_nunique
new_purchase_month_min
auth_hist_hour_max
auth_history_active_months_lag6_mean
auth_history_category_2_2.0_mean
new_merchant_id_nunique
new_category_2_1.0_mean
new_month_lag_max
unauth_hist_category_3_1_mean
new_subsector_id_nunique
auth_hist_purchase_year_std
new_purchase_year_std
unauth_hist_months_diff_min
new_active_months_lag12_mean
feature_2_3
unauth_hist_active_months_lag12_mean
feature_1_3
unauth_hist_installments_max
unauth_hist_months_diff_max
new_merchant_group_id_nunique
new_category_3_3_mean
new_installments_max
unauth_hist_purchase_month_min
feature_1_1
unauth_hist_category_2_2.0_mean
unauth_hist_purchase_month_max
new_city_id_nunique
unauth_hist_category_2_4.0_mean
auth_hist_purchase_month_min
auth_hist_purchase_month_max
unauth_hist_category_2_3.0_mean
unauth_hist_category_2_5.0_mean
new_state_id_nunique
unauth_hist_numerical_1_min
feature_3
new_months_diff_max
new_weekofyear_nunique
new_weekend_sum
new_months_diff_min
new_category_2_4.0_mean
auth_hist_months_diff_max
new_active_months_lag3_mean
unauth_hist_purchase_year_mean
new_installments_min
new_category_2_5.0_mean
unauth_hist_purchase_year_std
new_month_lag_min
feature_2_1
feature_1_5
auth_history_numerical_1_min
feature_1_4
feature_1_2
new_category_2_3.0_mean
new_active_months_lag6_mean
new_category_3_1_mean
new_category_2_2.0_mean
feature_2_2
auth_hist_purchase_year_max
unauth_hist_installments_min
new_authorized_flag_mean
auth_history_category_3_1_mean
auth_history_active_months_lag3_mean
unauth_hist_active_months_lag6_mean
unauth_hist_purchase_year_nunique
unauth_hist_purchase_year_max
auth_history_installments_min
new_purchase_year_nunique
auth_hist_purchase_year_nunique
new_purchase_year_max



Prepare for submission

In [ ]:
```
Y_predict=model.predict(X_test)
column_values=['target']
df = pd.DataFrame(data = Y_predict,columns = column_values)
df.shape
X_test_card.head()
test_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/test.csv')
df['card_id']=test_csv['card_id']
submission=pd.DataFrame()
submission['card_id']=df['card_id']
submission['target']=df['target']
submission.shape
submission.to_csv('mysubmission.csv',index=False)
```

In [1]:
```
print("After submitting in Kaggle:")
```

```
After submitting in Kaggle:
```

Submission and Description	Private Score	Public Score	Use for Final Score
lgbm-submission.csv just now by RaisulAlam add submission details	3.63770	3.71609	☐

Neural Network

In [54]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.models import Model
from keras.layers import Input
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras import regularizers
from keras.regularizers import l2
from keras.layers import Flatten
from keras.layers import Dense, Input , Dropout
from keras.layers import concatenate
from keras.layers.normalization import BatchNormalization
```

In [48]:
```
X_train.shape
```

Out[48]:
```
(170000, 303)
```

In [58]:
```
model = Sequential()
model.add(Dense(2 ** 10, input_dim = 303, kernel_initializer='random_uniform',
activation='relu'))
model.add(Dropout(0.25))
model.add(BatchNormalization())
model.add(Dense(2 ** 9, kernel_initializer='random_uniform', activation='relu'
))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Dense(2 ** 5, kernel_initializer='random_uniform', activation='relu'
))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Dense(1))
model.compile(loss=root_mean_squared_error, optimizer='adam')
```

In [59]:
```
history = model.fit(X_train, Y_train, batch_size=32, epochs=30, verbose=1, val
idation_data=(X_CV, Y_CV))
```

```
Epoch 1/30
5313/5313 [==============================] - 24s 4ms/step - loss: 3.3931 - va
l_loss: 1.4300
Epoch 2/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3898 - va
l_loss: 1.4288
Epoch 3/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3814 - va
l_loss: 1.4341
Epoch 4/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3964 - va
l_loss: 1.4315
Epoch 5/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3894 - va
l_loss: 1.4327
Epoch 6/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3822 - va
l_loss: 1.4338
Epoch 7/30
5313/5313 [==============================] - 24s 4ms/step - loss: 3.3836 - va
l_loss: 1.4309
Epoch 8/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3832 - va
l_loss: 1.4325
Epoch 9/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3814 - va
l_loss: 1.4334
Epoch 10/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3839 - va
l_loss: 1.4348
Epoch 11/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3944 - va
l_loss: 1.4345
Epoch 12/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3898 - va
l_loss: 1.4368
Epoch 13/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3951 - va
l_loss: 1.4349
Epoch 14/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3947 - va
l_loss: 1.4315
Epoch 15/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3897 - va
l_loss: 1.4326
Epoch 16/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3894 - va
l_loss: 1.4342
Epoch 17/30
5313/5313 [==============================] - 24s 4ms/step - loss: 3.3839 - va
l_loss: 1.4337
Epoch 18/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3912 - va
l_loss: 1.4356
Epoch 19/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3892 - va
l_loss: 1.4316
```

```
Epoch 20/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3815 - va
l_loss: 1.4388
Epoch 21/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3757 - va
l_loss: 1.4329
Epoch 22/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3954 - va
l_loss: 1.4381
Epoch 23/30
5313/5313 [==============================] - 24s 4ms/step - loss: 3.4059 - va
l_loss: 1.4356
Epoch 24/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3948 - va
l_loss: 1.4336
Epoch 25/30
5313/5313 [==============================] - 23s 4ms/step - loss: 3.3778 - va
l_loss: 1.4337
Epoch 26/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3845 - va
l_loss: 1.4324
Epoch 27/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3896 - va
l_loss: 1.4344
Epoch 28/30
5313/5313 [==============================] - 24s 4ms/step - loss: 3.3886 - va
l_loss: 1.4330
Epoch 29/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3877 - va
l_loss: 1.4346
Epoch 30/30
5313/5313 [==============================] - 22s 4ms/step - loss: 3.3941 - va
l_loss: 1.4350
```

In [ ]:
```
Y_predict=model.predict(X_test)
#This is for test submission
column_values=['target']
df = pd.DataFrame(data = Y_predict,columns = column_values)
df.shape
#X_test_card.head()
test_csv=pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/test.
csv')
df['card_id']=test_csv['card_id']
submission=pd.DataFrame()
submission['card_id']=df['card_id']
submission['target']=df['target']
submission.shape
submission.to_csv('mysubmission_mlp.csv',index=False)
```

But after submitting the rmse(private score) is increased to 3.8

Submission and Description	Private Score	Public Score	Use for Final Score
mysubmission_mlp.csv a few seconds ago by RaisulAlam add submission details	3.82018	3.93716	☐

XGBoost Regressor

In [37]:
```
import xgboost as xgb
model_xgb =  xgb.XGBRegressor(max_depth=2,
                              colsample_bytree=0.7,
                              n_estimators=20000,
                              scale_pos_weight = 9,
                              learning_rate=0.02,
                              min_child_weight=1.5,
                              #max_depth=3,
                              reg_alpha=0.75,
                              reg_lambda=0.45,
                              verbosity =1,
                              eval_metric  = 'rmse',
                              tree_method='gpu_hist',
                              n_jobs=-1)
```

In [44]:
```
from sklearn.model_selection import StratifiedKFold

from sklearn.model_selection import RepeatedKFold

import lightgbm as lgbm

import gc
from sklearn.metrics import mean_squared_error

oof = np.zeros(len(train))
predictions = np.zeros(len(test))

#folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=1234)
folds = RepeatedKFold(n_splits=5, n_repeats=2, random_state=4950)

for fold_index, (train_index,val_index) in enumerate(folds.split(X_train.value
s,Y_train.values)):
    print("fold:",fold_index,"started")
    gc.collect()
    bst = model_xgb.fit(X_train.iloc[train_index],Y_train.iloc[train_index],
              eval_set = [(X_train.iloc[val_index],Y_train.iloc[val_index])],
              early_stopping_rounds=200,
              verbose= 200,
              eval_metric ='rmse'
              )
    #val[val_index] = model_xgb.predict_proba(train[val_index])[:,1]
    #print('auc of this val set is {}'.format(roc_auc_score(y[val_index],val[v
al_index])))
    #pred += model_xgb.predict(test.values)[:,1]/folds.n_splits

    oof[val_index] = bst.predict(X_train.iloc[val_index])
    predictions += bst.predict(X_test) / (5*2)

np.sqrt(mean_squared_error(oof, Y_train))
```

```
fold: 0 started
[0]     validation_0-rmse:3.95585
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.74718
[400]   validation_0-rmse:3.73106
[600]   validation_0-rmse:3.72228
[800]   validation_0-rmse:3.71696
[1000]  validation_0-rmse:3.71361
[1200]  validation_0-rmse:3.71155
[1400]  validation_0-rmse:3.70988
[1600]  validation_0-rmse:3.70868
[1800]  validation_0-rmse:3.70792
[2000]  validation_0-rmse:3.70700
[2200]  validation_0-rmse:3.70681
[2400]  validation_0-rmse:3.70653
[2600]  validation_0-rmse:3.70633
[2800]  validation_0-rmse:3.70596
[3000]  validation_0-rmse:3.70548
[3200]  validation_0-rmse:3.70507
[3400]  validation_0-rmse:3.70477
[3600]  validation_0-rmse:3.70471
[3800]  validation_0-rmse:3.70446
[4000]  validation_0-rmse:3.70406
[4200]  validation_0-rmse:3.70401
Stopping. Best iteration:
[4176]  validation_0-rmse:3.70395

fold: 1 started
[0]     validation_0-rmse:4.03851
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.83709
[400]   validation_0-rmse:3.82077
[600]   validation_0-rmse:3.81234
[800]   validation_0-rmse:3.80768
[1000]  validation_0-rmse:3.80455
[1200]  validation_0-rmse:3.80215
[1400]  validation_0-rmse:3.80009
[1600]  validation_0-rmse:3.79844
[1800]  validation_0-rmse:3.79727
[2000]  validation_0-rmse:3.79629
[2200]  validation_0-rmse:3.79547
[2400]  validation_0-rmse:3.79494
[2600]  validation_0-rmse:3.79432
[2800]  validation_0-rmse:3.79420
[3000]  validation_0-rmse:3.79403
[3200]  validation_0-rmse:3.79375
Stopping. Best iteration:
[3132]  validation_0-rmse:3.79365

fold: 2 started
[0]     validation_0-rmse:3.95593
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.76144
[400]   validation_0-rmse:3.74762
[600]   validation_0-rmse:3.74071
[800]   validation_0-rmse:3.73584
[1000]  validation_0-rmse:3.73300
```

```
[1200]  validation_0-rmse:3.73110
[1400]  validation_0-rmse:3.72918
[1600]  validation_0-rmse:3.72789
[1800]  validation_0-rmse:3.72675
[2000]  validation_0-rmse:3.72543
[2200]  validation_0-rmse:3.72425
[2400]  validation_0-rmse:3.72331
[2600]  validation_0-rmse:3.72218
[2800]  validation_0-rmse:3.72150
[3000]  validation_0-rmse:3.72120
[3200]  validation_0-rmse:3.72070
[3400]  validation_0-rmse:3.72041
[3600]  validation_0-rmse:3.71987
[3800]  validation_0-rmse:3.71977
[4000]  validation_0-rmse:3.71936
[4200]  validation_0-rmse:3.71931
[4400]  validation_0-rmse:3.71912
Stopping. Best iteration:
[4333]  validation_0-rmse:3.71904

fold: 3 started
[0]     validation_0-rmse:3.90255
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.70815
[400]   validation_0-rmse:3.69277
[600]   validation_0-rmse:3.68467
[800]   validation_0-rmse:3.68056
[1000]  validation_0-rmse:3.67745
[1200]  validation_0-rmse:3.67546
[1400]  validation_0-rmse:3.67408
[1600]  validation_0-rmse:3.67240
[1800]  validation_0-rmse:3.67118
[2000]  validation_0-rmse:3.67033
[2200]  validation_0-rmse:3.66954
[2400]  validation_0-rmse:3.66898
[2600]  validation_0-rmse:3.66852
[2800]  validation_0-rmse:3.66817
[3000]  validation_0-rmse:3.66777
[3200]  validation_0-rmse:3.66772
[3400]  validation_0-rmse:3.66765
[3600]  validation_0-rmse:3.66748
[3800]  validation_0-rmse:3.66728
[4000]  validation_0-rmse:3.66691
[4200]  validation_0-rmse:3.66664
[4400]  validation_0-rmse:3.66624
[4600]  validation_0-rmse:3.66581
[4800]  validation_0-rmse:3.66559
[5000]  validation_0-rmse:3.66532
Stopping. Best iteration:
[4988]  validation_0-rmse:3.66528

fold: 4 started
[0]     validation_0-rmse:3.87928
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.68764
[400]   validation_0-rmse:3.67357
[600]   validation_0-rmse:3.66586
```

```
[800]   validation_0-rmse:3.66151
[1000]  validation_0-rmse:3.65885
[1200]  validation_0-rmse:3.65682
[1400]  validation_0-rmse:3.65560
[1600]  validation_0-rmse:3.65409
[1800]  validation_0-rmse:3.65274
[2000]  validation_0-rmse:3.65161
[2200]  validation_0-rmse:3.65066
[2400]  validation_0-rmse:3.64986
[2600]  validation_0-rmse:3.64913
[2800]  validation_0-rmse:3.64841
[3000]  validation_0-rmse:3.64784
[3200]  validation_0-rmse:3.64734
[3400]  validation_0-rmse:3.64690
[3600]  validation_0-rmse:3.64648
[3800]  validation_0-rmse:3.64613
[4000]  validation_0-rmse:3.64574
[4200]  validation_0-rmse:3.64543
[4400]  validation_0-rmse:3.64500
[4600]  validation_0-rmse:3.64497
[4800]  validation_0-rmse:3.64474
[5000]  validation_0-rmse:3.64455
[5200]  validation_0-rmse:3.64434
[5400]  validation_0-rmse:3.64436
[5600]  validation_0-rmse:3.64418
[5800]  validation_0-rmse:3.64415
[6000]  validation_0-rmse:3.64404
Stopping. Best iteration:
[5973]  validation_0-rmse:3.64392

fold: 5 started
[0]     validation_0-rmse:3.97712
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.77312
[400]   validation_0-rmse:3.75526
[600]   validation_0-rmse:3.74633
[800]   validation_0-rmse:3.74197
[1000]  validation_0-rmse:3.73859
[1200]  validation_0-rmse:3.73541
[1400]  validation_0-rmse:3.73300
[1600]  validation_0-rmse:3.73154
[1800]  validation_0-rmse:3.73031
[2000]  validation_0-rmse:3.72924
[2200]  validation_0-rmse:3.72840
[2400]  validation_0-rmse:3.72768
[2600]  validation_0-rmse:3.72695
[2800]  validation_0-rmse:3.72642
[3000]  validation_0-rmse:3.72588
[3200]  validation_0-rmse:3.72513
[3400]  validation_0-rmse:3.72474
[3600]  validation_0-rmse:3.72436
[3800]  validation_0-rmse:3.72425
[4000]  validation_0-rmse:3.72395
[4200]  validation_0-rmse:3.72361
[4400]  validation_0-rmse:3.72352
[4600]  validation_0-rmse:3.72309
[4800]  validation_0-rmse:3.72282
```

```
[5000]  validation_0-rmse:3.72266
[5200]  validation_0-rmse:3.72255
[5400]  validation_0-rmse:3.72241
[5600]  validation_0-rmse:3.72221
[5800]  validation_0-rmse:3.72190
[6000]  validation_0-rmse:3.72168
Stopping. Best iteration:
[5999]  validation_0-rmse:3.72167

fold: 6 started
[0]     validation_0-rmse:3.95081
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.75417
[400]   validation_0-rmse:3.73934
[600]   validation_0-rmse:3.73203
[800]   validation_0-rmse:3.72707
[1000]  validation_0-rmse:3.72444
[1200]  validation_0-rmse:3.72218
[1400]  validation_0-rmse:3.72037
[1600]  validation_0-rmse:3.71885
[1800]  validation_0-rmse:3.71779
[2000]  validation_0-rmse:3.71723
[2200]  validation_0-rmse:3.71639
[2400]  validation_0-rmse:3.71573
[2600]  validation_0-rmse:3.71527
[2800]  validation_0-rmse:3.71475
[3000]  validation_0-rmse:3.71450
[3200]  validation_0-rmse:3.71418
[3400]  validation_0-rmse:3.71377
[3600]  validation_0-rmse:3.71361
[3800]  validation_0-rmse:3.71332
[4000]  validation_0-rmse:3.71321
[4200]  validation_0-rmse:3.71276
[4400]  validation_0-rmse:3.71272
[4600]  validation_0-rmse:3.71264
[4800]  validation_0-rmse:3.71231
[5000]  validation_0-rmse:3.71203
[5200]  validation_0-rmse:3.71193
[5400]  validation_0-rmse:3.71189
[5600]  validation_0-rmse:3.71198
Stopping. Best iteration:
[5420]  validation_0-rmse:3.71187

fold: 7 started
[0]     validation_0-rmse:3.94624
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.74711
[400]   validation_0-rmse:3.73085
[600]   validation_0-rmse:3.72158
[800]   validation_0-rmse:3.71563
[1000]  validation_0-rmse:3.71126
[1200]  validation_0-rmse:3.70828
[1400]  validation_0-rmse:3.70579
[1600]  validation_0-rmse:3.70398
[1800]  validation_0-rmse:3.70232
[2000]  validation_0-rmse:3.70135
[2200]  validation_0-rmse:3.70028
```

```
[2400]  validation_0-rmse:3.69964
[2600]  validation_0-rmse:3.69869
[2800]  validation_0-rmse:3.69835
[3000]  validation_0-rmse:3.69774
[3200]  validation_0-rmse:3.69710
[3400]  validation_0-rmse:3.69696
[3600]  validation_0-rmse:3.69648
[3800]  validation_0-rmse:3.69624
[4000]  validation_0-rmse:3.69624
Stopping. Best iteration:
[3807]  validation_0-rmse:3.69617

fold: 8 started
[0]     validation_0-rmse:3.90513
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.71510
[400]   validation_0-rmse:3.70198
[600]   validation_0-rmse:3.69464
[800]   validation_0-rmse:3.69052
[1000]  validation_0-rmse:3.68754
[1200]  validation_0-rmse:3.68576
[1400]  validation_0-rmse:3.68430
[1600]  validation_0-rmse:3.68277
[1800]  validation_0-rmse:3.68158
[2000]  validation_0-rmse:3.68046
[2200]  validation_0-rmse:3.67989
[2400]  validation_0-rmse:3.67929
[2600]  validation_0-rmse:3.67896
[2800]  validation_0-rmse:3.67863
[3000]  validation_0-rmse:3.67826
[3200]  validation_0-rmse:3.67815
[3400]  validation_0-rmse:3.67786
[3600]  validation_0-rmse:3.67743
[3800]  validation_0-rmse:3.67714
[4000]  validation_0-rmse:3.67699
[4200]  validation_0-rmse:3.67657
[4400]  validation_0-rmse:3.67663
[4600]  validation_0-rmse:3.67655
Stopping. Best iteration:
[4582]  validation_0-rmse:3.67647

fold: 9 started
[0]     validation_0-rmse:3.95438
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.75386
[400]   validation_0-rmse:3.73850
[600]   validation_0-rmse:3.73214
[800]   validation_0-rmse:3.72878
[1000]  validation_0-rmse:3.72657
[1200]  validation_0-rmse:3.72484
[1400]  validation_0-rmse:3.72369
[1600]  validation_0-rmse:3.72309
[1800]  validation_0-rmse:3.72231
[2000]  validation_0-rmse:3.72126
[2200]  validation_0-rmse:3.72002
[2400]  validation_0-rmse:3.71944
[2600]  validation_0-rmse:3.71894
```

```
[2800]  validation_0-rmse:3.71851
[3000]  validation_0-rmse:3.71838
[3200]  validation_0-rmse:3.71821
[3400]  validation_0-rmse:3.71801
[3600]  validation_0-rmse:3.71808
[3800]  validation_0-rmse:3.71785
Stopping. Best iteration:
[3795]  validation_0-rmse:3.71781
```

Out[44]: 3.704836623559138

In [43]:
```
np.sqrt(mean_squared_error(oof, Y_train))
```

Out[43]: 3.704836623559138

In [45]:
```
sample_submission = pd.read_csv('../input/elo-merchant-category-recommendation/sample_submission.csv')
sample_submission['target'] = predictions
sample_submission.to_csv('xgb_submission.csv', index=False)
```

xgb_submission.csv an hour ago by RaisulAlam add submission details	3.66345	3.76282

In [2]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model_Description", "Public Score", "Private Score"]
x.add_row(["LGBM Regressor", 3.71609, 3.63770])
x.add_row(["XGBOOST Regressor", 3.76282, 3.66345])
x.add_row(["MLP Architecture",3.93716, 3.82018])
print(x)
```

```
+-------------------+--------------+---------------+
| Model_Description | Public Score | Private Score |
+-------------------+--------------+---------------+
|   LGBM Regressor  |   3.71609    |     3.6377    |
| XGBOOST Regressor |   3.76282    |    3.66345    |
|  MLP Architecture |   3.93716    |    3.82018    |
+-------------------+--------------+---------------+
```

To improve the RMSE for the test submission , we will take the reference to other kernels and perform some advanced feature engineering with the Kfold modelling technic

This kernel is extensively inspired from the below reference kernel

ref: https://www.kaggle.com/mfjwr1/simple-lightgbm-without-blending/output (https://www.kaggle.com/mfjwr1/simple-lightgbm-without-blending/output)

In [3]:
```
# Import the necessary libraries
import numpy as np
import pandas as pd
import os
import time
import warnings
import gc
gc.collect()
import os
from six.moves import urllib
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
warnings.filterwarnings('ignore')
%matplotlib inline
plt.style.use('seaborn')
from scipy.stats import norm, skew
from sklearn.preprocessing import StandardScaler
```

In [4]:
```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_log_error,mean_squared_error, r2_scor
e,mean_absolute_error
from sklearn.model_selection import StratifiedKFold, RepeatedKFold
from sklearn.model_selection import KFold #for K-fold cross validation
from scipy import sparse
```

In [ ]:
```
train_csv = pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/tr
ain.csv',parse_dates=["first_active_month"])
test_csv = pd.read_csv('/kaggle/input/elo-merchant-category-recommendation/tes
t.csv',parse_dates=["first_active_month"])
```

In [6]:
```
# Remove the Outliers if any
train_csv['outliers'] = 0
train_csv.loc[train_csv['target'] < -30, 'outliers'] = 1
train_csv['outliers'].value_counts()

for features in ['feature_1','feature_2','feature_3']:
    order_label = train_csv.groupby([features])['outliers'].mean()
    #print(order_label)
    train_csv[features] = train_csv[features].map(order_label)
    test_csv[features] =  test_csv[features].map(order_label)
```

In [7]:
```
#train
train_csv['day'] = (datetime.date(2018, 2, 1) - train_csv['first_active_month'
].dt.date).dt.days
train_csv['quarter'] = train_csv['first_active_month'].dt.quarter

for feature in ['feature_1','feature_2','feature_3']:
  column=feature+'_day'
  train_csv[column] = train_csv['day'] * train_csv[feature]
  column=feature+'_day_ratio'
  train_csv[column] = train_csv[feature] / train_csv['day']
```

In [8]:
```
#test
test_csv['day'] = (datetime.date(2018, 2, 1) - test_csv['first_active_month'].
dt.date).dt.days
test_csv['quarter'] = test_csv['first_active_month'].dt.quarter

for feature in ['feature_1','feature_2','feature_3']:
  column=feature+'_day'
  test_csv[column] = test_csv['day'] * test_csv[feature]
  column=feature+'_day_ratio'
  test_csv[column] = test_csv[feature] / test_csv['day']


gc.collect()
```

Out[8]:
```
30
```

In [1]:
```
historical_transactions = pd.read_csv('/kaggle/input/elo-merchant-category-rec
ommendation/historical_transactions.csv')
new_merchant_transactions = pd.read_csv('/kaggle/input/elo-merchant-category-r
ecommendation/new_merchant_transactions.csv')
gc.collect()
```

In [10]:
```
#history
historical_transactions['category_3']=historical_transactions['category_3'].fi
llna('A',inplace=True)
historical_transactions['category_2']=historical_transactions['category_2'].fi
llna(1.0,inplace=True)
historical_transactions['merchant_id']=historical_transactions['merchant_id'].
fillna('M_ID_00a6ca8a8a',inplace=True)

#new
new_merchant_transactions['category_2'] = new_merchant_transactions['category_
2'].fillna(1.0,inplace=True)
new_merchant_transactions['category_3'] = new_merchant_transactions['category_
3'].fillna('A',inplace=True)
new_merchant_transactions['merchant_id'] = new_merchant_transactions['merchant
_id'].fillna('M_ID_00a6ca8a8a',inplace=True)
```

In [11]:
```
#history
historical_transactions['installments'].replace(-1, np.nan,inplace=True)
historical_transactions['installments'].replace(999, np.nan,inplace=True)

#new
new_merchant_transactions['installments'].replace(-1, np.nan,inplace=True)
new_merchant_transactions['installments'].replace(999, np.nan,inplace=True)
```

In [12]:
```
#history
historical_transactions['purchase_amount'] = historical_transactions['purchase
_amount'].apply(lambda x: min(x, 0.8))

#new
new_merchant_transactions['purchase_amount'] = new_merchant_transactions['purc
hase_amount'].apply(lambda x: min(x, 0.8))
```

In [24]:
```
#https://stackoverflow.com/questions/40901770/is-there-a-simple-way-to-change-a-column-of-yes-no-to-1-0-in-a-pandas-dataframe
#history
historical_transactions['authorized_flag'] = historical_transactions['authorized_flag'].map({'Y': 1, 'N': 0})
historical_transactions['category_1'] = historical_transactions['category_1'].map({'Y': 1, 'N': 0})
historical_transactions['category_3'] = historical_transactions['category_3'].map({'A': 1, 'B': 2, 'C': 3})

#new
new_merchant_transactions['authorized_flag'] = new_merchant_transactions['authorized_flag'].map({'Y': 1, 'N': 0})
new_merchant_transactions['category_1'] = new_merchant_transactions['category_1'].map({'Y': 1, 'N': 0})
new_merchant_transactions['category_3'] = new_merchant_transactions['category_3'].map({'A':0, 'B':1, 'C':2})
```

In [13]:
```
#https://www.kaggle.com/juliaflower/feature-selection-lgbm-with-python

#history
historical_transactions['pur_date'] = pd.DatetimeIndex(historical_transactions['purchase_date']).date
historical_transactions['pur_date'] = pd.DatetimeIndex(historical_transactions['pur_date']).astype(np.int64) * 1e-9

#new
new_merchant_transactions['pur_date'] = pd.DatetimeIndex(new_merchant_transactions['purchase_date']).date
new_merchant_transactions['pur_date'] = pd.DatetimeIndex(new_merchant_transactions['pur_date']).astype(np.int64) * 1e-9
```

In [14]:
```
#history
historical_transactions['purchase_date']=pd.to_datetime(historical_transactions['purchase_date'],format='%Y-%m')

#new
new_merchant_transactions['purchase_date']=pd.to_datetime(new_merchant_transactions['purchase_date'],format='%Y-%m')
```

In [15]:
```
#ref: https://stackoverflow.com/questions/42822768/pandas-number-of-months-between-two-dates

def date_features(data):
  import datetime
  current_time = datetime.datetime.now()
  data['months_diff']= (current_time.year - data.purchase_date.dt.year) * 12 + (current_time.month - data.purchase_date.dt.month)
  data['months_diff'] = data['months_diff'] + data['month_lag']
  data['purchase_month']=data.purchase_date.dt.month
  data['purchase_day']=data['purchase_date'].dt.day
  data['weekday']=data['purchase_date'].dt.weekday
  data['purchase_year'] = data['purchase_date'].dt.year
  #data['purchase_month'] = data['purchase_date'].dt.month
  data['weekofyear'] = data['purchase_date'].dt.weekofyear
  data['dayofweek'] = data['purchase_date'].dt.dayofweek
  data['weekend'] = (data.purchase_date.dt.weekday >=5).astype(int)
  data['hour'] = data['purchase_date'].dt.hour

  return data
```

In [16]:
```
historical_transactions = date_features(historical_transactions)
new_merchant_transactions = date_features(new_merchant_transactions)
```

In [17]:
```
#other features:
historical_transactions['duration'] = historical_transactions['purchase_amount'] * historical_transactions['months_diff']
historical_transactions['amount_month_ratio'] = historical_transactions['purchase_amount'] / historical_transactions['months_diff']
historical_transactions['price'] = historical_transactions['purchase_amount'] / historical_transactions['installments']

new_merchant_transactions['duration'] = new_merchant_transactions['purchase_amount'] * new_merchant_transactions['months_diff']
new_merchant_transactions['amount_month_ratio'] = new_merchant_transactions['purchase_amount'] / new_merchant_transactions['months_diff']
new_merchant_transactions['price'] = new_merchant_transactions['purchase_amount'] / new_merchant_transactions['installments']
```

In [18]:
```
for i in ['category_2','category_3']:
  historical_transactions[i + '_mean']=historical_transactions['purchase_amoun
t'].groupby(historical_transactions[i]).agg('mean')
  historical_transactions[i + '_min']=historical_transactions['purchase_amoun
t'].groupby(historical_transactions[i]).agg('min')
  historical_transactions[i + '_max']=historical_transactions['purchase_amoun
t'].groupby(historical_transactions[i]).agg('max')
  historical_transactions[i + '_sum']=historical_transactions['purchase_amoun
t'].groupby(historical_transactions[i]).agg('sum')
  historical_transactions[i + '_var']=historical_transactions['purchase_amoun
t'].groupby(historical_transactions[i]).agg('var')

  new_merchant_transactions[i + '_mean']=new_merchant_transactions['purchase_a
mount'].groupby(new_merchant_transactions[i]).agg('mean')
  new_merchant_transactions[i + '_min']=new_merchant_transactions['purchase_am
ount'].groupby(new_merchant_transactions[i]).agg('min')
  new_merchant_transactions[i + '_max']=new_merchant_transactions['purchase_am
ount'].groupby(new_merchant_transactions[i]).agg('max')
  new_merchant_transactions[i + '_sum']=new_merchant_transactions['purchase_am
ount'].groupby(new_merchant_transactions[i]).agg('sum')
  new_merchant_transactions[i + '_var']=new_merchant_transactions['purchase_am
ount'].groupby(new_merchant_transactions[i]).agg('var')
```

In [26]:
```
#ref: https://www.geeksforgeeks.org/python-pandas-dataframe-aggregate/
#This function performs the aggregate  operation on the numerical features
def aggregate_func(data,str_data):
    agg_func= {
      'authorized_flag':['sum','mean'],
      'card_id':['size','count'],
      'category_1':['mean','sum','max','min'],
      'installments':['max','var','mean','skew','sum'],
      'merchant_category_id':['nunique'],
      'month_lag':['max','mean','min','var','skew'],
      'purchase_amount':['max','mean','min','var','sum','skew'],
      'subsector_id':['nunique'],
      'months_diff':['mean','max','min','var','skew'],
      'purchase_month':['max','min','mean','nunique'],
      'weekofyear': ['mean','max','min','nunique'],
      'weekend': ['sum', 'mean'],
      'weekday':['sum','mean'],
      'hour': ['mean','max','min','nunique'],
      'purchase_day':['nunique','max','min','mean'],
      'pur_date':['max','min'],
      'price' :['sum','mean','max','min','var'],
      'duration' : ['mean','min','max','var','skew'],
      'amount_month_ratio':['mean','min','max','var','skew'],

    }
    #print("11")
    featured_data=data.groupby(['card_id']).agg(agg_func)
    #print(featured_data)
    #print("ssss")
    col_list=[]
    for col in featured_data.columns:
        col_str='_'.join(col)
        col_str=str_data + col_str
        ren_name=col_str.split(",")
        col_list.extend(ren_name)

    col_list.insert(0,'card_id')
    featured_data.reset_index(inplace=True)
    return featured_data,col_list
```

In [20]:
```
#historical_transactions.info
```

In [27]:
```
hist_trans,col_list=aggregate_func(historical_transactions,'hist_')
hist_trans.columns=col_list
```

In [28]:
```
new_trans,col_list=aggregate_func(new_merchant_transactions,'new_')
new_trans.columns=col_list
```

In [30]:
```
train_data=pd.merge(train_csv,hist_trans,on='card_id',how='left')
test_data=pd.merge(test_csv,hist_trans,on='card_id',how='left')

train_data=pd.merge(train_data,new_trans,on='card_id',how='left')
test_data=pd.merge(test_data,new_trans,on='card_id',how='left')
```

In [31]:
```
gc.collect()
```

Out[31]: 42

In [32]:
```
def additional_feature(data,str):
  data[str +'_purchase_date_max'] = pd.to_datetime(data[str + '_pur_date_max'
])
  #print("1")
  data[str + '_purchase_date_min'] = pd.to_datetime(data[str + '_pur_date_min'
])
  #print("2")
  data[str + '_purchase_date_diff'] = (data[str + '_purchase_date_max'] - data
[str + '_purchase_date_min']).dt.days
  data[str + '_purchase_date_average'] = data[str + '_purchase_date_diff']/dat
a[str + '_card_id_size']
  #print("3")
  data[str + '_purchase_date_uptonow'] = (datetime.datetime.today() - data[str
+ '_purchase_date_max']).dt.days
  #print("4")
  data[str + '_purchase_date_uptomin'] = (datetime.datetime.today() - data[str
+ '_purchase_date_min']).dt.days
  #print("5")
  data[str + '_first_buy'] = (data[str + '_purchase_date_min'] - data['first_a
ctive_month']).dt.days
  data[str + '_last_buy'] = (data[str + '_purchase_date_max'] - data['first_ac
tive_month']).dt.days

  if (str=='hist'):
    for feature in [str + '_purchase_date_max', str + '_purchase_date_min']:
      data[feature] = data[feature].astype(np.int64) * 1e-9

  if (str=='new'):
    for feature in ['new_purchase_date_max','new_purchase_date_min']:
      data[feature] = pd.DatetimeIndex(data[feature]).astype(np.int64) * 1e-9

  return data
```

In [33]:
```
train_data=additional_feature(train_data,'hist')
train_data=additional_feature(train_data,'new')

test_data=additional_feature(test_data,'hist')
test_data=additional_feature(test_data,'new')
```

In [34]:
```
def combined_feature(data):
  data['card_id_total'] = data['new_card_id_size'] + data['hist_card_id_size']
  data['card_id_cnt_total'] = data['new_card_id_count'] + data['hist_card_id_c
ount']
  data['card_id_cnt_ratio'] = data['new_card_id_count'] / data['hist_card_id_c
ount']
  data['purchase_amount_total'] = data['new_purchase_amount_sum'] + data['hist
_purchase_amount_sum']
  data['purchase_amount_mean'] = data['new_purchase_amount_mean'] + data['hist
_purchase_amount_mean']
  data['purchase_amount_max'] = data['new_purchase_amount_max'] + data['hist_p
urchase_amount_max']
  data['purchase_amount_min'] = data['new_purchase_amount_min']+ data['hist_pu
rchase_amount_min']
  data['purchase_amount_ratio'] = data['new_purchase_amount_sum'] / data['hist
_purchase_amount_sum']
  data['month_diff_mean'] = data['new_months_diff_mean'] + data['hist_months_d
iff_mean']
  data['month_diff_ratio'] = data['new_months_diff_mean'] / data['hist_months_
diff_mean']
  data['month_lag_mean'] = data['new_month_lag_mean'] + data['hist_month_lag_m
ean']
  data['month_lag_max'] = data['new_month_lag_max'] + data['hist_month_lag_ma
x']
  data['month_lag_min'] = data['new_month_lag_min']+ data['hist_month_lag_min'
]
  data['category_1_mean'] = data['new_category_1_mean'] + data['hist_category_
1_mean']
  data['installments_total'] = data['new_installments_sum'] + data['hist_insta
llments_sum']
  data['installments_mean'] = data['new_installments_mean'] + data['hist_insta
llments_mean']
  data['installments_max'] = data['new_installments_max'] + data['hist_install
ments_max']
  data['installments_ratio'] = data['new_installments_sum'] / data['hist_insta
llments_sum']
  data['price_total'] = data['purchase_amount_total'] / data['installments_tot
al']
  data['price_mean'] = data['purchase_amount_mean'] / data['installments_mean'
]
  data['price_max'] = data['purchase_amount_max'] / data['installments_max']
  data['duration_mean'] = data['new_duration_mean'] + data['hist_duration_mea
n']
  data['duration_min'] = data['new_duration_min'] + data['hist_duration_min']
  data['duration_max'] = data['new_duration_max'] + data['hist_duration_max']
  data['amount_month_ratio_mean']= data['new_amount_month_ratio_mean'] + data[
'hist_amount_month_ratio_mean']
  data['amount_month_ratio_min'] = data['new_amount_month_ratio_min'] + data[
'hist_amount_month_ratio_min']
  data['amount_month_ratio_max']= data['new_amount_month_ratio_max'] + data['h
ist_amount_month_ratio_max']
  data['new_CLV'] = data['new_card_id_count'] * data['new_purchase_amount_sum'
] / data['new_months_diff_mean']
  data['hist_CLV'] = data['hist_card_id_count'] * data['hist_purchase_amount_s
um'] / data['hist_months_diff_mean']
  data['CLV_ratio'] = data['new_CLV'] / data['hist_CLV']
```

```
    return data
```

In [35]:
```
train=combined_feature(train_data)
test=combined_feature(test_data)
```

In [36]:
```
train.to_pickle('train.pkl')
test.to_pickle('test.pkl')
```

In [4]:
```
import pickle
test=pd.read_pickle('/kaggle/input/todaydata/test.pkl')
train=pd.read_pickle('/kaggle/input/todaydata/train.pkl')
```

In [ ]:
```
```

In [5]:
```
feature = [c for c in train.columns if c not in ['target', 'outliers']]
#target = train['target']
cols=train[feature].columns
```

In [6]:
```
# https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/77537
from tqdm import tqdm
from scipy.stats import ks_2samp
list_p_value =[]

for i in tqdm(cols):
    list_p_value.append(ks_2samp(test[i] , train[i])[1])

Se = pd.Series(list_p_value, index = cols).sort_values()
list_discarded = list(Se[Se < .1].index)
```

```
100%|██████████| 195/195 [00:10<00:00, 18.93it/s]
```

In [7]:
```
discard_cols=list_discarded + ['target', 'outliers']
feature = [c for c in train.columns if c not in discard_cols]
```

In [8]:
```
X_Train=train[feature]
Y_Train=train['target']
```

In [9]:
```
X_Train = X_Train.drop(['card_id', 'first_active_month'], axis = 1)
#test = test.drop(['card_id', 'first_active_month'], axis = 1)
```

In [10]:
```
def root_mean_squared_error(y_true, y_pred):
    """Root mean squared error regression loss"""
    return np.sqrt(np.mean(np.square(y_true-y_pred)))
```

In [12]:
```
#https://www.kaggle.com/aleksandrovich/smart-hyperparameter-tuning-with-optuna
import lightgbm as lgbm
from optuna.samplers import TPESampler
import optuna

sampler = TPESampler(seed=10) # for reproducibility
def objective(trial):

    # dtrain = lgb.Dataset(X_train, label=y_train)

    param = {
                'objective': 'regression',
                'metric': 'rmse',
                'verbosity': -1,
                'learning_rate': 0.01,
                'device': 'gpu',
                'seed': trial.suggest_int('seed',1,50),
                'boosting_type': 'gbdt',
                'n_jobs': -1,
                'num_leaves': trial.suggest_int('num_leaves', 16, 64),
                'colsample_bytree': trial.suggest_uniform('colsample_bytree', 0.
001, 1),
                'subsample': trial.suggest_uniform('subsample', 0.001, 1),
                'top_rate': trial.suggest_uniform('top_rate', 0.001, 1),
                'other_rate': trial.suggest_uniform('other_rate', 0.001, 1),
                'max_depth': trial.suggest_int('max_depth', 1, 12),
                'reg_alpha': trial.suggest_uniform('reg_alpha', 0, 10),
                'reg_lambda': trial.suggest_uniform('reg_lambda', 0, 10),
                'min_split_gain': trial.suggest_uniform('min_split_gain', 0, 10
),
                'min_child_weight': trial.suggest_uniform('min_child_weight', 0,
45),
                'min_data_in_leaf': trial.suggest_int('min_data_in_leaf', 16, 64
)
                }


    lgbm_regr = lgbm.LGBMRegressor(**param)
    gbm_2 = lgbm_regr.fit(X_Train, Y_Train, eval_set=[(X_Train, Y_Train)], ver
bose=False)
    return root_mean_squared_error(Y_Train,gbm_2.predict(X_Train))

study = optuna.create_study(direction='minimize', sampler=sampler)
study.optimize(objective, n_trials=200)
```

```
[I 2020-11-20 11:25:37,613] A new study created in memory with name: no-name-
6578a692-c698-4d28-a3cc-ad371a31a62e
[I 2020-11-20 11:25:51,310] Trial 0 finished with value: 3.7713163522359276 a
nd parameters: {'seed': 10, 'num_leaves': 52, 'colsample_bytree': 0.495095339
60383073, 'subsample': 0.44357193274091977, 'top_rate': 0.8320794476122297,
 'other_rate': 0.5837384152007985, 'max_depth': 2, 'reg_alpha': 7.09208009843
012, 'reg_lambda': 2.65566126772097, 'min_split_gain': 2.6360284602895234, 'm
in_child_weight': 6.7670040267794676, 'min_data_in_leaf': 32}. Best is trial
 0 with value: 3.7713163522359276.
[I 2020-11-20 11:26:00,717] Trial 1 finished with value: 3.7092360587328757 a
nd parameters: {'seed': 37, 'num_leaves': 63, 'colsample_bytree': 0.336735512
88008735, 'subsample': 0.8909257141486431, 'top_rate': 0.1989236912199373, 'o
ther_rate': 0.031586037218820716, 'max_depth': 5, 'reg_alpha': 2.918760681706
3316, 'reg_lambda': 9.177741225129434, 'min_split_gain': 7.145757833976906,
 'min_child_weight': 24.414496560506755, 'min_data_in_leaf': 41}. Best is tri
al 1 with value: 3.7092360587328757.
[I 2020-11-20 11:26:13,015] Trial 2 finished with value: 3.699860668185463 an
d parameters: {'seed': 14, 'num_leaves': 44, 'colsample_bytree': 0.6744594814
51279, 'subsample': 0.4423913412485731, 'top_rate': 0.4345799793399604, 'othe
r_rate': 0.6181492114908479, 'max_depth': 5, 'reg_alpha': 0.6556326635477827,
'reg_lambda': 0.5644418747007063, 'min_split_gain': 7.654558182151854, 'min_c
hild_weight': 0.5304613223233917, 'min_data_in_leaf': 27}. Best is trial 2 wi
th value: 3.699860668185463.
[I 2020-11-20 11:26:22,670] Trial 3 finished with value: 3.69778034251213 and
parameters: {'seed': 29, 'num_leaves': 26, 'colsample_bytree': 0.319916852899
55674, 'subsample': 0.09136888992163647, 'top_rate': 0.3013993565795672, 'oth
er_rate': 0.11487037750168622, 'max_depth': 12, 'reg_alpha': 8.53042990318658
3, 'reg_lambda': 4.399874555042919, 'min_split_gain': 1.2195414712716746, 'mi
n_child_weight': 32.9280581091934, 'min_data_in_leaf': 48}. Best is trial 3 w
ith value: 3.69778034251213.
[I 2020-11-20 11:26:30,873] Trial 4 finished with value: 3.744075063281806 an
d parameters: {'seed': 25, 'num_leaves': 60, 'colsample_bytree': 0.8569934521
552754, 'subsample': 0.3523009867926558, 'top_rate': 0.7548930438383273, 'oth
er_rate': 0.29666574517279903, 'max_depth': 3, 'reg_alpha': 9.83020866852340
4, 'reg_lambda': 4.674032789842478, 'min_split_gain': 8.757444947401376, 'min
_child_weight': 13.32309144888607, 'min_data_in_leaf': 37}. Best is trial 3 w
ith value: 3.69778034251213.
[I 2020-11-20 11:26:38,753] Trial 5 finished with value: 3.7841392269946157 a
nd parameters: {'seed': 43, 'num_leaves': 38, 'colsample_bytree': 0.152000867
6229213, 'subsample': 0.3847303342435074, 'top_rate': 0.9443164515265623, 'ot
her_rate': 0.9876378494269704, 'max_depth': 2, 'reg_alpha': 5.87130925253948
8, 'reg_lambda': 1.4947133698593174, 'min_split_gain': 1.712385982079181, 'mi
n_child_weight': 17.872403529490743, 'min_data_in_leaf': 61}. Best is trial 3
with value: 3.69778034251213.
[I 2020-11-20 11:26:47,493] Trial 6 finished with value: 3.8152993594712115 a
nd parameters: {'seed': 41, 'num_leaves': 41, 'colsample_bytree': 0.003404354
5815532947, 'subsample': 0.5492675393881244, 'top_rate': 0.12784486961017158,
'other_rate': 0.08071288795139236, 'max_depth': 5, 'reg_alpha': 3.30719311982
132, 'reg_lambda': 7.7383029621059585, 'min_split_gain': 0.39959208689977266,
'min_child_weight': 19.327148029423725, 'min_data_in_leaf': 41}. Best is tria
l 3 with value: 3.69778034251213.
[I 2020-11-20 11:26:55,291] Trial 7 finished with value: 3.748562936171755 an
d parameters: {'seed': 47, 'num_leaves': 64, 'colsample_bytree': 0.3470008029
29953, 'subsample': 0.04405425884878945, 'top_rate': 0.8800352593433981, 'oth
er_rate': 0.7634773465565373, 'max_depth': 3, 'reg_alpha': 4.17509143839267,
 'reg_lambda': 6.055775643937568, 'min_split_gain': 5.134666274082884, 'min_c
hild_weight': 26.902649158333816, 'min_data_in_leaf': 25}. Best is trial 3 wi
```

```
th value: 3.69778034251213.
[I 2020-11-20 11:27:05,462] Trial 8 finished with value: 3.7576497707422636 a
nd parameters: {'seed': 1, 'num_leaves': 54, 'colsample_bytree': 0.0394219678
6224467, 'subsample': 0.3331116518291369, 'top_rate': 0.0012722589598640657,
 'other_rate': 0.6649908106546216, 'max_depth': 12, 'reg_alpha': 4.7513224741
50506, 'reg_lambda': 2.9279797628950908, 'min_split_gain': 0.642510606948244
5, 'min_child_weight': 44.04686155909392, 'min_data_in_leaf': 45}. Best is tr
ial 3 with value: 3.69778034251213.
[I 2020-11-20 11:27:16,786] Trial 9 finished with value: 3.69196994641622 and
parameters: {'seed': 46, 'num_leaves': 27, 'colsample_bytree': 0.487067115136
26305, 'subsample': 0.10682712550709428, 'top_rate': 0.34069725136424994, 'ot
her_rate': 0.8357352229526429, 'max_depth': 12, 'reg_alpha': 5.78136429882467
5, 'reg_lambda': 8.539337505004864, 'min_split_gain': 0.6809727353795003, 'mi
n_child_weight': 20.903886350069964, 'min_data_in_leaf': 42}. Best is trial 9
with value: 3.69196994641622.
[I 2020-11-20 11:27:27,050] Trial 10 finished with value: 3.709569641424341 a
nd parameters: {'seed': 31, 'num_leaves': 16, 'colsample_bytree': 0.976127004
5310055, 'subsample': 0.7353435413307334, 'top_rate': 0.6293747327237661, 'ot
her_rate': 0.9700437098278902, 'max_depth': 9, 'reg_alpha': 0.944003874737810
6, 'reg_lambda': 9.628714332904462, 'min_split_gain': 3.9890776570065984, 'mi
n_child_weight': 37.52497452728797, 'min_data_in_leaf': 16}. Best is trial 9
 with value: 3.69196994641622.
[I 2020-11-20 11:27:36,901] Trial 11 finished with value: 3.6977619675022 and
parameters: {'seed': 50, 'num_leaves': 22, 'colsample_bytree': 0.548178519850
8585, 'subsample': 0.002426473496185355, 'top_rate': 0.3903863359481885, 'oth
er_rate': 0.3380816686917125, 'max_depth': 12, 'reg_alpha': 8.80889172121587,
'reg_lambda': 5.4016109862899455, 'min_split_gain': 0.32918255094238213, 'min
_child_weight': 33.11315105548441, 'min_data_in_leaf': 53}. Best is trial 9 w
ith value: 3.69196994641622.
[I 2020-11-20 11:27:47,783] Trial 12 finished with value: 3.6946300144326254
 and parameters: {'seed': 48, 'num_leaves': 24, 'colsample_bytree': 0.6249254
169082468, 'subsample': 0.18944456803045107, 'top_rate': 0.444151871630025,
 'other_rate': 0.3615164623387812, 'max_depth': 10, 'reg_alpha': 9.9998891250
0448, 'reg_lambda': 6.9055679168476995, 'min_split_gain': 0.1020544404993941
6, 'min_child_weight': 33.279697115053494, 'min_data_in_leaf': 56}. Best is t
rial 9 with value: 3.69196994641622.
[I 2020-11-20 11:27:59,088] Trial 13 finished with value: 3.6857870658713163
 and parameters: {'seed': 50, 'num_leaves': 30, 'colsample_bytree': 0.7288008
035838035, 'subsample': 0.19790598129465373, 'top_rate': 0.5763351304173416,
 'other_rate': 0.42670529469616686, 'max_depth': 9, 'reg_alpha': 7.0928910757
87034, 'reg_lambda': 8.033882894796466, 'min_split_gain': 3.0586570960855397,
'min_child_weight': 41.08122816206095, 'min_data_in_leaf': 63}. Best is trial
13 with value: 3.6857870658713163.
[I 2020-11-20 11:28:12,526] Trial 14 finished with value: 3.681143105743361 a
nd parameters: {'seed': 50, 'num_leaves': 33, 'colsample_bytree': 0.785464063
4129069, 'subsample': 0.17193605158434555, 'top_rate': 0.6109379921891891, 'o
ther_rate': 0.8421653921234666, 'max_depth': 9, 'reg_alpha': 6.63354018678304
1, 'reg_lambda': 8.282880066499233, 'min_split_gain': 3.6584868724650175, 'mi
n_child_weight': 42.42286065594607, 'min_data_in_leaf': 64}. Best is trial 14
with value: 3.681143105743361.
[I 2020-11-20 11:28:24,296] Trial 15 finished with value: 3.6806536506518874
 and parameters: {'seed': 36, 'num_leaves': 34, 'colsample_bytree': 0.8060683
503188445, 'subsample': 0.2435389489612071, 'top_rate': 0.6062204717032859,
 'other_rate': 0.488002188240867, 'max_depth': 8, 'reg_alpha': 6.982033706399
089, 'reg_lambda': 7.586397003088477, 'min_split_gain': 4.22523997165895, 'mi
n_child_weight': 44.25872760136531, 'min_data_in_leaf': 62}. Best is trial 15
with value: 3.6806536506518874.
```

```
[I 2020-11-20 11:28:36,532] Trial 16 finished with value: 3.6817500207773466
 and parameters: {'seed': 35, 'num_leaves': 36, 'colsample_bytree': 0.9602388
464154403, 'subsample': 0.23843303665684346, 'top_rate': 0.716011128104549,
 'other_rate': 0.4991309625623855, 'max_depth': 7, 'reg_alpha': 7.37771585057
1061, 'reg_lambda': 9.801055460696915, 'min_split_gain': 5.4825560368411494,
 'min_child_weight': 44.913513211574745, 'min_data_in_leaf': 64}. Best is tri
al 15 with value: 3.6806536506518874.
[I 2020-11-20 11:28:50,424] Trial 17 finished with value: 3.6729058368113527
 and parameters: {'seed': 22, 'num_leaves': 46, 'colsample_bytree': 0.8087455
511299725, 'subsample': 0.6143457700247706, 'top_rate': 0.5694611938855659,
 'other_rate': 0.8499753783843708, 'max_depth': 7, 'reg_alpha': 6.00314817126
977, 'reg_lambda': 7.179586632557911, 'min_split_gain': 4.164204095326569, 'm
in_child_weight': 39.85601279044802, 'min_data_in_leaf': 56}. Best is trial 1
7 with value: 3.6729058368113527.
[I 2020-11-20 11:29:03,748] Trial 18 finished with value: 3.6714670795245503
 and parameters: {'seed': 21, 'num_leaves': 47, 'colsample_bytree': 0.8623889
929720487, 'subsample': 0.6065908342282336, 'top_rate': 0.5065688874512148,
 'other_rate': 0.2562843638855871, 'max_depth': 7, 'reg_alpha': 8.41922178090
5684, 'reg_lambda': 6.694139013716882, 'min_split_gain': 6.2091721123666765,
 'min_child_weight': 37.551505013927745, 'min_data_in_leaf': 57}. Best is tri
al 18 with value: 3.6714670795245503.
[I 2020-11-20 11:29:18,601] Trial 19 finished with value: 3.682152965349539 a
nd parameters: {'seed': 19, 'num_leaves': 47, 'colsample_bytree': 0.911573148
3192646, 'subsample': 0.623229918836214, 'top_rate': 0.5243145003049676, 'oth
er_rate': 0.21760584063071908, 'max_depth': 6, 'reg_alpha': 8.59640771516221
8, 'reg_lambda': 6.272620615801136, 'min_split_gain': 6.191822805124719, 'min
_child_weight': 29.09435931214169, 'min_data_in_leaf': 55}. Best is trial 18
 with value: 3.6714670795245503.
[I 2020-11-20 11:29:32,483] Trial 20 finished with value: 3.666469015226232 a
nd parameters: {'seed': 21, 'num_leaves': 51, 'colsample_bytree': 0.866322334
1707414, 'subsample': 0.823657802284142, 'top_rate': 0.2516374772723754, 'oth
er_rate': 0.2761375369222218, 'max_depth': 7, 'reg_alpha': 7.988290176922034,
'reg_lambda': 3.6575278742496993, 'min_split_gain': 9.467857346079505, 'min_c
hild_weight': 38.85598005725836, 'min_data_in_leaf': 49}. Best is trial 20 wi
th value: 3.666469015226232.
[I 2020-11-20 11:29:46,218] Trial 21 finished with value: 3.667192810647348 a
nd parameters: {'seed': 21, 'num_leaves': 50, 'colsample_bytree': 0.863076445
5483789, 'subsample': 0.8011694221981986, 'top_rate': 0.2034524619981104, 'ot
her_rate': 0.22038869211454745, 'max_depth': 7, 'reg_alpha': 8.2286200634801
8, 'reg_lambda': 3.7079627038644785, 'min_split_gain': 9.6643490990673, 'min_
child_weight': 38.57864485862147, 'min_data_in_leaf': 51}. Best is trial 20 w
ith value: 3.666469015226232.
[I 2020-11-20 11:30:00,107] Trial 22 finished with value: 3.6799880825190776
 and parameters: {'seed': 16, 'num_leaves': 52, 'colsample_bytree': 0.8957857
44601716, 'subsample': 0.9613438997245743, 'top_rate': 0.20877934713286575,
 'other_rate': 0.19289991580096344, 'max_depth': 6, 'reg_alpha': 7.9945829795
36571, 'reg_lambda': 3.645742803186239, 'min_split_gain': 9.865376432524428,
 'min_child_weight': 36.68333733388213, 'min_data_in_leaf': 50}. Best is tria
l 20 with value: 3.666469015226232.
[I 2020-11-20 11:30:16,068] Trial 23 finished with value: 3.6540188780868528
 and parameters: {'seed': 8, 'num_leaves': 57, 'colsample_bytree': 0.98986059
43405381, 'subsample': 0.7907703660245433, 'top_rate': 0.1897264030708881, 'o
ther_rate': 0.21412727841781137, 'max_depth': 8, 'reg_alpha': 9.4735055986869
68, 'reg_lambda': 2.253139884257207, 'min_split_gain': 9.433448981062583, 'mi
n_child_weight': 36.723024724217424, 'min_data_in_leaf': 50}. Best is trial 2
3 with value: 3.6540188780868528.
[I 2020-11-20 11:30:31,535] Trial 24 finished with value: 3.6440090304597543
```

```
 and parameters: {'seed': 5, 'num_leaves': 57, 'colsample_bytree': 0.99268020
83478886, 'subsample': 0.8038526107708127, 'top_rate': 0.017857544841497985,
 'other_rate': 0.15175531390201652, 'max_depth': 10, 'reg_alpha': 9.592353098
569978, 'reg_lambda': 1.7885853542335513, 'min_split_gain': 9.61808281289620
5, 'min_child_weight': 30.2045002481213, 'min_data_in_leaf': 49}. Best is tri
al 24 with value: 3.6440090304597543.
[I 2020-11-20 11:30:46,392] Trial 25 finished with value: 3.6425669285324056
 and parameters: {'seed': 2, 'num_leaves': 58, 'colsample_bytree': 0.98208945
19402135, 'subsample': 0.7599894062605592, 'top_rate': 0.005086955423018014,
 'other_rate': 0.005943928754293337, 'max_depth': 10, 'reg_alpha': 9.83502359
1682052, 'reg_lambda': 1.487303757197943, 'min_split_gain': 9.01970445958202
7, 'min_child_weight': 27.184504944630056, 'min_data_in_leaf': 48}. Best is t
rial 25 with value: 3.6425669285324056.
[I 2020-11-20 11:31:02,385] Trial 26 finished with value: 3.640570703812481 a
nd parameters: {'seed': 2, 'num_leaves': 58, 'colsample_bytree': 0.9903037331
191186, 'subsample': 0.7327076396444363, 'top_rate': 0.0111154212716808, 'oth
er_rate': 0.007973880329531224, 'max_depth': 10, 'reg_alpha': 9.5249089803819
27, 'reg_lambda': 0.5760830253357776, 'min_split_gain': 8.397197043177453, 'm
in_child_weight': 29.073188400514645, 'min_data_in_leaf': 46}. Best is trial
 26 with value: 3.640570703812481.
[I 2020-11-20 11:31:18,591] Trial 27 finished with value: 3.634108613418497 a
nd parameters: {'seed': 1, 'num_leaves': 58, 'colsample_bytree': 0.9950982531
819598, 'subsample': 0.704062823685543, 'top_rate': 0.012032904666657428, 'ot
her_rate': 0.009565002396675234, 'max_depth': 11, 'reg_alpha': 9.475942171523
938, 'reg_lambda': 0.027186554206700464, 'min_split_gain': 8.428396363077752,
'min_child_weight': 28.81733447182579, 'min_data_in_leaf': 37}. Best is trial
27 with value: 3.634108613418497.
[I 2020-11-20 11:31:34,510] Trial 28 finished with value: 3.632379796048591 a
nd parameters: {'seed': 1, 'num_leaves': 60, 'colsample_bytree': 0.9905104942
105738, 'subsample': 0.6906008010176942, 'top_rate': 0.08772395981855174, 'ot
her_rate': 0.0016069506749218365, 'max_depth': 11, 'reg_alpha': 9.10197108119
9247, 'reg_lambda': 0.38326017714827854, 'min_split_gain': 8.249135427511467,
'min_child_weight': 24.763171196568884, 'min_data_in_leaf': 36}. Best is tria
l 28 with value: 3.632379796048591.
[I 2020-11-20 11:31:48,326] Trial 29 finished with value: 3.6298425675817296
 and parameters: {'seed': 10, 'num_leaves': 62, 'colsample_bytree': 0.7351884
552844894, 'subsample': 0.6913761958159236, 'top_rate': 0.0699707176704157,
 'other_rate': 0.013513379531850543, 'max_depth': 11, 'reg_alpha': 9.13246271
570577, 'reg_lambda': 0.31083850390436973, 'min_split_gain': 8.01441102466791
7, 'min_child_weight': 24.97420429214658, 'min_data_in_leaf': 33}. Best is tr
ial 29 with value: 3.6298425675817296.
[I 2020-11-20 11:32:03,015] Trial 30 finished with value: 3.630804061140695 a
nd parameters: {'seed': 9, 'num_leaves': 62, 'colsample_bytree': 0.7182677828
087032, 'subsample': 0.6625897554155801, 'top_rate': 0.10179310062119044, 'ot
her_rate': 0.06496582363626219, 'max_depth': 11, 'reg_alpha': 9.1061437922016
07, 'reg_lambda': 0.1276398864038244, 'min_split_gain': 7.839930019085669, 'm
in_child_weight': 15.078789644547708, 'min_data_in_leaf': 35}. Best is trial
 29 with value: 3.6298425675817296.
[I 2020-11-20 11:32:17,296] Trial 31 finished with value: 3.6350587370620304
 and parameters: {'seed': 10, 'num_leaves': 62, 'colsample_bytree': 0.5749156
282342996, 'subsample': 0.6718861527334776, 'top_rate': 0.09820811991895723,
 'other_rate': 0.0675210607326205, 'max_depth': 11, 'reg_alpha': 9.1503531740
6206, 'reg_lambda': 0.017718973574645747, 'min_split_gain': 8.05979432225074
2, 'min_child_weight': 13.11881128486775, 'min_data_in_leaf': 34}. Best is tr
ial 29 with value: 3.6298425675817296.
[I 2020-11-20 11:32:32,235] Trial 32 finished with value: 3.627143515325044 a
nd parameters: {'seed': 7, 'num_leaves': 64, 'colsample_bytree': 0.7176145668
```

```
409653, 'subsample': 0.5340114025606384, 'top_rate': 0.09020752779863597, 'ot
her_rate': 0.004991590569186219, 'max_depth': 11, 'reg_alpha': 9.063020008603
182, 'reg_lambda': 0.11651923269351183, 'min_split_gain': 7.1078403256274205,
'min_child_weight': 22.891686608620002, 'min_data_in_leaf': 29}. Best is tria
l 32 with value: 3.627143515325044.
[I 2020-11-20 11:32:46,054] Trial 33 finished with value: 3.6273419275980023
 and parameters: {'seed': 12, 'num_leaves': 63, 'colsample_bytree': 0.6823452
497740918, 'subsample': 0.48077329054245255, 'top_rate': 0.10993517405019995,
'other_rate': 0.11074978126356291, 'max_depth': 11, 'reg_alpha': 7.5863357499
638555, 'reg_lambda': 0.7423472752816497, 'min_split_gain': 7.06226414853113
3, 'min_child_weight': 23.6060011373924, 'min_data_in_leaf': 26}. Best is tri
al 32 with value: 3.627143515325044.
[I 2020-11-20 11:33:00,111] Trial 34 finished with value: 3.6271760425974624
 and parameters: {'seed': 12, 'num_leaves': 64, 'colsample_bytree': 0.7049107
669756071, 'subsample': 0.47401459413212854, 'top_rate': 0.13814807794093606,
'other_rate': 0.12322680565224914, 'max_depth': 11, 'reg_alpha': 7.6552739950
62406, 'reg_lambda': 0.984344316822891, 'min_split_gain': 7.235660039005512,
 'min_child_weight': 15.642491739652353, 'min_data_in_leaf': 28}. Best is tri
al 32 with value: 3.627143515325044.
[I 2020-11-20 11:33:15,309] Trial 35 finished with value: 3.629166665438734 a
nd parameters: {'seed': 13, 'num_leaves': 63, 'colsample_bytree': 0.651800312
0961448, 'subsample': 0.49878624435762714, 'top_rate': 0.15008806309959935,
 'other_rate': 0.13565047166642372, 'max_depth': 11, 'reg_alpha': 7.558897532
602143, 'reg_lambda': 0.9843830891565734, 'min_split_gain': 6.90293601133107,
'min_child_weight': 9.178148630012245, 'min_data_in_leaf': 29}. Best is trial
32 with value: 3.627143515325044.
[I 2020-11-20 11:33:27,804] Trial 36 finished with value: 3.6322544726741355
 and parameters: {'seed': 14, 'num_leaves': 64, 'colsample_bytree': 0.4549262
500775113, 'subsample': 0.4819235505238544, 'top_rate': 0.15530088340842413,
 'other_rate': 0.14396053029235584, 'max_depth': 12, 'reg_alpha': 7.815969819
991466, 'reg_lambda': 0.8551670660488218, 'min_split_gain': 7.00045946221484
1, 'min_child_weight': 6.723848166036664, 'min_data_in_leaf': 28}. Best is tr
ial 32 with value: 3.627143515325044.
[I 2020-11-20 11:33:41,438] Trial 37 finished with value: 3.639915745824197 a
nd parameters: {'seed': 6, 'num_leaves': 54, 'colsample_bytree': 0.6266396283
372906, 'subsample': 0.5336136868597666, 'top_rate': 0.2855146774550579, 'oth
er_rate': 0.10868752748243164, 'max_depth': 10, 'reg_alpha': 6.3942952927203
1, 'reg_lambda': 1.2446750846771253, 'min_split_gain': 7.124305090489827, 'mi
n_child_weight': 7.834443652609171, 'min_data_in_leaf': 20}. Best is trial 32
with value: 3.627143515325044.
[I 2020-11-20 11:33:54,454] Trial 38 finished with value: 3.6465435251530116
 and parameters: {'seed': 14, 'num_leaves': 55, 'colsample_bytree': 0.6707603
354482934, 'subsample': 0.4333174498744989, 'top_rate': 0.2585435418106172,
 'other_rate': 0.16001497613865692, 'max_depth': 9, 'reg_alpha': 7.6873483132
3767, 'reg_lambda': 2.297630916327523, 'min_split_gain': 6.578272953987756,
 'min_child_weight': 9.261301980679733, 'min_data_in_leaf': 29}. Best is tria
l 32 with value: 3.627143515325044.
[I 2020-11-20 11:34:10,280] Trial 39 finished with value: 3.6229343561195724
 and parameters: {'seed': 17, 'num_leaves': 64, 'colsample_bytree': 0.6561116
953377482, 'subsample': 0.3971951112479438, 'top_rate': 0.15415928687614514,
 'other_rate': 0.08437144494031187, 'max_depth': 12, 'reg_alpha': 5.563065186
197658, 'reg_lambda': 0.9701901377706659, 'min_split_gain': 5.70610820202523
8, 'min_child_weight': 0.4403459386629187, 'min_data_in_leaf': 23}. Best is t
rial 39 with value: 3.6229343561195724.
[I 2020-11-20 11:34:23,279] Trial 40 finished with value: 3.6281708628885956
 and parameters: {'seed': 17, 'num_leaves': 64, 'colsample_bytree': 0.5412398
473692728, 'subsample': 0.39714899114170416, 'top_rate': 0.37275683314763863,
```

```
'other_rate': 0.06504295027292556, 'max_depth': 12, 'reg_alpha': 4.4387218124
27616, 'reg_lambda': 1.9501537886981204, 'min_split_gain': 5.505837751278469,
'min_child_weight': 0.778749642005609, 'min_data_in_leaf': 22}. Best is trial
39 with value: 3.6229343561195724.
[I 2020-11-20 11:34:35,568] Trial 41 finished with value: 3.632726626508676 a
nd parameters: {'seed': 17, 'num_leaves': 64, 'colsample_bytree': 0.422689974
23138854, 'subsample': 0.404804269455371, 'top_rate': 0.33515549246673215, 'o
ther_rate': 0.06288890733077418, 'max_depth': 12, 'reg_alpha': 3.415909915814
2725, 'reg_lambda': 1.9066266674656664, 'min_split_gain': 5.645667523254449,
 'min_child_weight': 3.1444992339758433, 'min_data_in_leaf': 23}. Best is tri
al 39 with value: 3.6229343561195724.
[I 2020-11-20 11:34:48,546] Trial 42 finished with value: 3.633104329907944 a
nd parameters: {'seed': 25, 'num_leaves': 60, 'colsample_bytree': 0.567761885
4129899, 'subsample': 0.30807017062649533, 'top_rate': 0.05567590360181312,
 'other_rate': 0.09767953333284546, 'max_depth': 12, 'reg_alpha': 5.149184212
91551, 'reg_lambda': 2.7935926303559055, 'min_split_gain': 4.805424996566071,
'min_child_weight': 3.0711683097200364, 'min_data_in_leaf': 20}. Best is tria
l 39 with value: 3.6229343561195724.
[I 2020-11-20 11:35:02,284] Trial 43 finished with value: 3.627425552735517 a
nd parameters: {'seed': 13, 'num_leaves': 60, 'colsample_bytree': 0.701951754
3812079, 'subsample': 0.4599005465666995, 'top_rate': 0.15503627953912572, 'o
ther_rate': 0.046904012691762684, 'max_depth': 11, 'reg_alpha': 4.16428011667
6383, 'reg_lambda': 0.8966945226470264, 'min_split_gain': 6.035234429236106,
 'min_child_weight': 22.598544828072892, 'min_data_in_leaf': 16}. Best is tri
al 39 with value: 3.6229343561195724.
[I 2020-11-20 11:35:18,440] Trial 44 finished with value: 3.6275685796911885
 and parameters: {'seed': 12, 'num_leaves': 60, 'colsample_bytree': 0.7561919
321066815, 'subsample': 0.5600747716334687, 'top_rate': 0.1484028925906528,
 'other_rate': 0.3258986586208776, 'max_depth': 11, 'reg_alpha': 2.0854102588
14491, 'reg_lambda': 1.2077033225397678, 'min_split_gain': 7.368173992825456,
'min_child_weight': 22.40705945897773, 'min_data_in_leaf': 16}. Best is trial
39 with value: 3.6229343561195724.
[I 2020-11-20 11:35:25,511] Trial 45 finished with value: 3.811180040856963 a
nd parameters: {'seed': 5, 'num_leaves': 61, 'colsample_bytree': 0.6913350007
59789, 'subsample': 0.46603048199757, 'top_rate': 0.2320930549447111, 'other_
rate': 0.17872509705375683, 'max_depth': 1, 'reg_alpha': 5.371141788713099,
 'reg_lambda': 0.7263813194812376, 'min_split_gain': 6.30617106894394, 'min_c
hild_weight': 17.54064950544075, 'min_data_in_leaf': 26}. Best is trial 39 wi
th value: 3.6229343561195724.
[I 2020-11-20 11:35:37,368] Trial 46 finished with value: 3.6586444939786857
 and parameters: {'seed': 12, 'num_leaves': 41, 'colsample_bytree': 0.6079561
898646484, 'subsample': 0.31723704945903636, 'top_rate': 0.16699400032298967,
'other_rate': 0.05344159286163289, 'max_depth': 10, 'reg_alpha': 3.8666090109
908793, 'reg_lambda': 1.391680198846303, 'min_split_gain': 4.830612990018135,
'min_child_weight': 19.68939847691697, 'min_data_in_leaf': 18}. Best is trial
39 with value: 3.6229343561195724.
[I 2020-11-20 11:35:51,957] Trial 47 finished with value: 3.6283776075865157
 and parameters: {'seed': 7, 'num_leaves': 64, 'colsample_bytree': 0.76945056
04264644, 'subsample': 0.36578235045631197, 'top_rate': 0.11968283042133088,
 'other_rate': 0.3842422398264953, 'max_depth': 12, 'reg_alpha': 2.4059686737
037076, 'reg_lambda': 2.9706538854947127, 'min_split_gain': 7.43971863840333
1, 'min_child_weight': 22.615545964124106, 'min_data_in_leaf': 31}. Best is t
rial 39 with value: 3.6229343561195724.
[I 2020-11-20 11:36:05,542] Trial 48 finished with value: 3.6412701639247733
 and parameters: {'seed': 11, 'num_leaves': 55, 'colsample_bytree': 0.7099886
066769878, 'subsample': 0.5262950382392881, 'top_rate': 0.054701432286679175,
'other_rate': 0.1141125234472912, 'max_depth': 9, 'reg_alpha': 4.841407526244
```

```
773, 'reg_lambda': 0.7326566118231139, 'min_split_gain': 6.6042185210699955,
 'min_child_weight': 18.17525319429816, 'min_data_in_leaf': 24}. Best is tria
l 39 with value: 3.6229343561195724.
[I 2020-11-20 11:36:16,082] Trial 49 finished with value: 3.7218047984175837
 and parameters: {'seed': 28, 'num_leaves': 64, 'colsample_bytree': 0.6656874
152799125, 'subsample': 0.4477717752228675, 'top_rate': 0.3001648431910522,
 'other_rate': 0.5947325644127941, 'max_depth': 4, 'reg_alpha': 5.56913600814
7627, 'reg_lambda': 2.388227200572396, 'min_split_gain': 5.931486770174293,
 'min_child_weight': 22.615225191400732, 'min_data_in_leaf': 31}. Best is tri
al 39 with value: 3.6229343561195724.
[I 2020-11-20 11:36:30,821] Trial 50 finished with value: 3.630458389573519 a
nd parameters: {'seed': 16, 'num_leaves': 60, 'colsample_bytree': 0.816730618
7358591, 'subsample': 0.35685948375722654, 'top_rate': 0.0451693181491582, 'o
ther_rate': 0.24186459873163596, 'max_depth': 11, 'reg_alpha': 6.416684815095
646, 'reg_lambda': 3.219605973044445, 'min_split_gain': 7.601173523684881, 'm
in_child_weight': 16.23261872648766, 'min_data_in_leaf': 21}. Best is trial 3
9 with value: 3.6229343561195724.
[I 2020-11-20 11:36:44,941] Trial 51 finished with value: 3.626009955032339 a
nd parameters: {'seed': 4, 'num_leaves': 60, 'colsample_bytree': 0.7604512312
425419, 'subsample': 0.5653980038240898, 'top_rate': 0.17073563037999442, 'ot
her_rate': 0.31853751101266436, 'max_depth': 11, 'reg_alpha': 2.0608122244973
144, 'reg_lambda': 1.1211561631126672, 'min_split_gain': 7.344791094211679,
 'min_child_weight': 20.77131110752529, 'min_data_in_leaf': 18}. Best is tria
l 39 with value: 3.6229343561195724.
[I 2020-11-20 11:36:58,188] Trial 52 finished with value: 3.6323225350305615
 and parameters: {'seed': 4, 'num_leaves': 59, 'colsample_bytree': 0.59367168
99861301, 'subsample': 0.5550671060008332, 'top_rate': 0.12412816160732637,
 'other_rate': 0.4430277745794373, 'max_depth': 10, 'reg_alpha': 1.4998406752
006908, 'reg_lambda': 1.0379593989146965, 'min_split_gain': 6.57764215724384
6, 'min_child_weight': 20.590987424579364, 'min_data_in_leaf': 17}. Best is t
rial 39 with value: 3.6229343561195724.
[I 2020-11-20 11:37:12,448] Trial 53 finished with value: 3.63001134080151 an
d parameters: {'seed': 8, 'num_leaves': 62, 'colsample_bytree': 0.63984351565
47977, 'subsample': 0.43926742391550466, 'top_rate': 0.186401305716407, 'othe
r_rate': 0.5409500678950061, 'max_depth': 11, 'reg_alpha': 0.734885005327206
5, 'reg_lambda': 1.6669966267390581, 'min_split_gain': 5.137642305749175, 'mi
n_child_weight': 12.682720184508325, 'min_data_in_leaf': 26}. Best is trial 3
9 with value: 3.6229343561195724.
[I 2020-11-20 11:37:25,714] Trial 54 finished with value: 3.6355609838223777
 and parameters: {'seed': 15, 'num_leaves': 56, 'colsample_bytree': 0.5061225
183616036, 'subsample': 0.5867925261079099, 'top_rate': 0.23528264320100448,
 'other_rate': 0.2808249821295142, 'max_depth': 12, 'reg_alpha': 3.1196042847
502095, 'reg_lambda': 0.38719229494277785, 'min_split_gain': 8.8769064489042
1, 'min_child_weight': 23.959250918723328, 'min_data_in_leaf': 19}. Best is t
rial 39 with value: 3.6229343561195724.
[I 2020-11-20 11:37:40,611] Trial 55 finished with value: 3.6195138490491945
 and parameters: {'seed': 18, 'num_leaves': 64, 'colsample_bytree': 0.8219509
788750268, 'subsample': 0.504198597165328, 'top_rate': 0.9938401082420221, 'o
ther_rate': 0.10893596003865413, 'max_depth': 11, 'reg_alpha': 7.062651770877
939, 'reg_lambda': 0.12739938648121496, 'min_split_gain': 6.802708693703791,
 'min_child_weight': 21.622850861308486, 'min_data_in_leaf': 24}. Best is tri
al 55 with value: 3.6195138490491945.
[I 2020-11-20 11:37:56,059] Trial 56 finished with value: 3.6391041594449756
 and parameters: {'seed': 19, 'num_leaves': 64, 'colsample_bytree': 0.8089728
290704129, 'subsample': 0.5047370473489566, 'top_rate': 0.9133842094684428,
 'other_rate': 0.6711798490379235, 'max_depth': 8, 'reg_alpha': 6.93775620717
8094, 'reg_lambda': 0.13346196230732044, 'min_split_gain': 6.894073491098104,
```

```
'min_child_weight': 26.268976811661474, 'min_data_in_leaf': 24}. Best is tria
l 55 with value: 3.6195138490491945.
[I 2020-11-20 11:38:07,354] Trial 57 finished with value: 3.699734567119386 a
nd parameters: {'seed': 18, 'num_leaves': 20, 'colsample_bytree': 0.926578373
9515085, 'subsample': 0.2880128673446375, 'top_rate': 0.7031484576984727, 'ot
her_rate': 0.18862280253759234, 'max_depth': 10, 'reg_alpha': 7.3667671032380
04, 'reg_lambda': 5.069139738568365, 'min_split_gain': 7.358917970571061, 'mi
n_child_weight': 31.23250065436053, 'min_data_in_leaf': 27}. Best is trial 55
with value: 3.6195138490491945.
[I 2020-11-20 11:38:23,626] Trial 58 finished with value: 3.6203633601664476
 and parameters: {'seed': 4, 'num_leaves': 62, 'colsample_bytree': 0.83250126
20511022, 'subsample': 0.6354463257382033, 'top_rate': 0.8442653080892735, 'o
ther_rate': 0.32772852232549127, 'max_depth': 12, 'reg_alpha': 6.050031507160
472, 'reg_lambda': 0.5099311045289081, 'min_split_gain': 7.6845595778412115,
 'min_child_weight': 15.039875365616282, 'min_data_in_leaf': 22}. Best is tri
al 55 with value: 3.6195138490491945.
[I 2020-11-20 11:38:37,136] Trial 59 finished with value: 3.6354897553538486
 and parameters: {'seed': 3, 'num_leaves': 53, 'colsample_bytree': 0.76096117
54850207, 'subsample': 0.6400049558041699, 'top_rate': 0.980373495285269, 'ot
her_rate': 0.387508726627361, 'max_depth': 12, 'reg_alpha': 5.98813651970493
6, 'reg_lambda': 1.2819051143328595, 'min_split_gain': 7.694745680451717, 'mi
n_child_weight': 15.094936426773673, 'min_data_in_leaf': 22}. Best is trial 5
5 with value: 3.6195138490491945.
[I 2020-11-20 11:38:50,354] Trial 60 finished with value: 3.640167588181225 a
nd parameters: {'seed': 23, 'num_leaves': 49, 'colsample_bytree': 0.830211663
6221767, 'subsample': 0.5680814629628842, 'top_rate': 0.7873129836975569, 'ot
her_rate': 0.31183914768132415, 'max_depth': 12, 'reg_alpha': 0.2469599654299
6634, 'reg_lambda': 0.4077224021055016, 'min_split_gain': 8.513951264706314,
 'min_child_weight': 11.061391878751676, 'min_data_in_leaf': 24}. Best is tri
al 55 with value: 3.6195138490491945.
[I 2020-11-20 11:39:05,932] Trial 61 finished with value: 3.625201609473311 a
nd parameters: {'seed': 8, 'num_leaves': 62, 'colsample_bytree': 0.7885289474
517541, 'subsample': 0.5987979561233305, 'top_rate': 0.8687844787630764, 'oth
er_rate': 0.4517197159892741, 'max_depth': 11, 'reg_alpha': 6.67486366059637
7, 'reg_lambda': 0.029569026278754906, 'min_split_gain': 6.472898257769903,
 'min_child_weight': 20.85599702176622, 'min_data_in_leaf': 26}. Best is tria
l 55 with value: 3.6195138490491945.
[I 2020-11-20 11:39:21,537] Trial 62 finished with value: 3.6284489015542793
 and parameters: {'seed': 7, 'num_leaves': 62, 'colsample_bytree': 0.89705263
43834619, 'subsample': 0.6112478559497211, 'top_rate': 0.8460935608537764, 'o
ther_rate': 0.4460305690598173, 'max_depth': 10, 'reg_alpha': 6.5841944554016
33, 'reg_lambda': 0.06284241077744022, 'min_split_gain': 5.849341523104192,
 'min_child_weight': 19.171980073963862, 'min_data_in_leaf': 30}. Best is tri
al 55 with value: 3.6195138490491945.
[I 2020-11-20 11:39:36,439] Trial 63 finished with value: 3.634835704305556 a
nd parameters: {'seed': 4, 'num_leaves': 57, 'colsample_bytree': 0.8446574244
043452, 'subsample': 0.5193255669751626, 'top_rate': 0.971565684956657, 'othe
r_rate': 0.34793881884613986, 'max_depth': 12, 'reg_alpha': 6.235105730373736
5, 'reg_lambda': 0.5580796549592061, 'min_split_gain': 6.6423431809347555, 'm
in_child_weight': 20.55897558194963, 'min_data_in_leaf': 39}. Best is trial 5
5 with value: 3.6195138490491945.
[I 2020-11-20 11:39:50,805] Trial 64 finished with value: 3.62098965985463 an
d parameters: {'seed': 9, 'num_leaves': 61, 'colsample_bytree': 0.77655294908
97751, 'subsample': 0.5801040811463957, 'top_rate': 0.917280371961255, 'other
_rate': 0.5302259944995763, 'max_depth': 11, 'reg_alpha': 6.790837103703697,
 'reg_lambda': 0.029820959516332846, 'min_split_gain': 6.243437571599659, 'mi
n_child_weight': 17.364749411172898, 'min_data_in_leaf': 18}. Best is trial 5
```

```
5 with value: 3.6195138490491945.
[I 2020-11-20 11:40:05,950] Trial 65 finished with value: 3.6315325359798583
 and parameters: {'seed': 7, 'num_leaves': 59, 'colsample_bytree': 0.79238967
03322203, 'subsample': 0.5833892700231454, 'top_rate': 0.9044926182644935, 'o
ther_rate': 0.5181611293668158, 'max_depth': 9, 'reg_alpha': 5.6063718100566
3, 'reg_lambda': 0.12327807949198913, 'min_split_gain': 6.321577918046062, 'm
in_child_weight': 17.288803073395876, 'min_data_in_leaf': 18}. Best is trial
 55 with value: 3.6195138490491945.
[I 2020-11-20 11:40:20,674] Trial 66 finished with value: 3.625349967883777 a
nd parameters: {'seed': 10, 'num_leaves': 61, 'colsample_bytree': 0.745604326
6417999, 'subsample': 0.6317475809330632, 'top_rate': 0.854106440767179, 'oth
er_rate': 0.4709885940231707, 'max_depth': 12, 'reg_alpha': 6.83030698642024
1, 'reg_lambda': 0.013861138916663407, 'min_split_gain': 4.478151955847325,
 'min_child_weight': 25.938475957449054, 'min_data_in_leaf': 21}. Best is tri
al 55 with value: 3.6195138490491945.
[I 2020-11-20 11:40:36,661] Trial 67 finished with value: 3.6308372273777514
 and parameters: {'seed': 10, 'num_leaves': 58, 'colsample_bytree': 0.9445961
124758158, 'subsample': 0.6617796546377464, 'top_rate': 0.8724242060843471,
 'other_rate': 0.45939766403081483, 'max_depth': 12, 'reg_alpha': 6.860169104
347325, 'reg_lambda': 1.6092358641660223, 'min_split_gain': 4.51071910799878
9, 'min_child_weight': 26.595587369535462, 'min_data_in_leaf': 21}. Best is t
rial 55 with value: 3.6195138490491945.
[I 2020-11-20 11:40:51,563] Trial 68 finished with value: 3.621310175260278 a
nd parameters: {'seed': 9, 'num_leaves': 61, 'colsample_bytree': 0.8762366734
920328, 'subsample': 0.7283334244162555, 'top_rate': 0.801458294650323, 'othe
r_rate': 0.5682585307333665, 'max_depth': 12, 'reg_alpha': 6.177979074164358,
'reg_lambda': 0.4803272803082624, 'min_split_gain': 3.898826596292195, 'min_c
hild_weight': 21.623444220862023, 'min_data_in_leaf': 19}. Best is trial 55 w
ith value: 3.6195138490491945.
[I 2020-11-20 11:41:07,779] Trial 69 finished with value: 3.630057282600825 a
nd parameters: {'seed': 9, 'num_leaves': 56, 'colsample_bytree': 0.8724081233
599977, 'subsample': 0.73277002118864, 'top_rate': 0.7997733267098696, 'other
_rate': 0.7023763287564968, 'max_depth': 12, 'reg_alpha': 7.173144581929018,
 'reg_lambda': 0.0007987585935302337, 'min_split_gain': 3.120375531117632, 'm
in_child_weight': 25.62382122647794, 'min_data_in_leaf': 20}. Best is trial 5
5 with value: 3.6195138490491945.
[I 2020-11-20 11:41:18,665] Trial 70 finished with value: 3.67724787194721 an
d parameters: {'seed': 9, 'num_leaves': 44, 'colsample_bytree': 0.21784548975
29159, 'subsample': 0.7550322448664745, 'top_rate': 0.7207096951558762, 'othe
r_rate': 0.554510721515787, 'max_depth': 12, 'reg_alpha': 5.955631248646701,
 'reg_lambda': 0.43239729070038196, 'min_split_gain': 3.6987086529336053, 'mi
n_child_weight': 27.56728354341712, 'min_data_in_leaf': 23}. Best is trial 55
with value: 3.6195138490491945.
[I 2020-11-20 11:41:32,973] Trial 71 finished with value: 3.624107578009555 a
nd parameters: {'seed': 5, 'num_leaves': 61, 'colsample_bytree': 0.7503490451
823055, 'subsample': 0.8511110387564969, 'top_rate': 0.9211898149655332, 'oth
er_rate': 0.48849722649407945, 'max_depth': 11, 'reg_alpha': 6.73861944934832
2, 'reg_lambda': 0.6155015874446392, 'min_split_gain': 5.093365464043544, 'mi
n_child_weight': 20.940632368383383, 'min_data_in_leaf': 18}. Best is trial 5
5 with value: 3.6195138490491945.
[I 2020-11-20 11:41:48,271] Trial 72 finished with value: 3.622324658016556 a
nd parameters: {'seed': 6, 'num_leaves': 62, 'colsample_bytree': 0.8381537414
152733, 'subsample': 0.9617046806897531, 'top_rate': 0.9391098840513497, 'oth
er_rate': 0.4829116507773254, 'max_depth': 11, 'reg_alpha': 6.88206583968090
9, 'reg_lambda': 0.5033680890770729, 'min_split_gain': 5.250999915872439, 'mi
n_child_weight': 21.780471692198457, 'min_data_in_leaf': 16}. Best is trial 5
5 with value: 3.6195138490491945.
```

```
[I 2020-11-20 11:42:03,504] Trial 73 finished with value: 3.6244120249904284
 and parameters: {'seed': 2, 'num_leaves': 62, 'colsample_bytree': 0.88029231
04115535, 'subsample': 0.9102029036414085, 'top_rate': 0.9327445603334446, 'o
ther_rate': 0.6245118388934138, 'max_depth': 10, 'reg_alpha': 6.1079535131208
66, 'reg_lambda': 0.6671421752108799, 'min_split_gain': 5.081858866072238, 'm
in_child_weight': 18.925656058387226, 'min_data_in_leaf': 19}. Best is trial
 55 with value: 3.6195138490491945.
[I 2020-11-20 11:42:20,126] Trial 74 finished with value: 3.6257171555473735
 and parameters: {'seed': 2, 'num_leaves': 61, 'colsample_bytree': 0.87207608
20214047, 'subsample': 0.9058279028772659, 'top_rate': 0.943499784737238, 'ot
her_rate': 0.6394350332768288, 'max_depth': 10, 'reg_alpha': 6.12136987040553
9, 'reg_lambda': 0.676424694773112, 'min_split_gain': 5.211803809924793, 'min
_child_weight': 18.858095801554395, 'min_data_in_leaf': 17}. Best is trial 55
with value: 3.6195138490491945.
[I 2020-11-20 11:42:35,622] Trial 75 finished with value: 3.6219369642926513
 and parameters: {'seed': 5, 'num_leaves': 59, 'colsample_bytree': 0.92739639
15835642, 'subsample': 0.9921614709337113, 'top_rate': 0.9420013330258302, 'o
ther_rate': 0.5652052029784073, 'max_depth': 11, 'reg_alpha': 5.7539783434442
6, 'reg_lambda': 0.44671484471148043, 'min_split_gain': 5.345059633676241, 'm
in_child_weight': 16.625391549253425, 'min_data_in_leaf': 16}. Best is trial
 55 with value: 3.6195138490491945.
[I 2020-11-20 11:42:51,080] Trial 76 finished with value: 3.6280224823857337
 and parameters: {'seed': 5, 'num_leaves': 58, 'colsample_bytree': 0.93511105
9656972, 'subsample': 0.8602342652419516, 'top_rate': 0.8181072909461198, 'ot
her_rate': 0.5706019366357417, 'max_depth': 11, 'reg_alpha': 5.07308457057751
5, 'reg_lambda': 2.074803723090782, 'min_split_gain': 5.405292017535612, 'min
_child_weight': 16.70313777931354, 'min_data_in_leaf': 16}. Best is trial 55
 with value: 3.6195138490491945.
[I 2020-11-20 11:43:06,117] Trial 77 finished with value: 3.6422852942907062
 and parameters: {'seed': 6, 'num_leaves': 59, 'colsample_bytree': 0.84349370
5583673, 'subsample': 0.9967391707823792, 'top_rate': 0.9685436308086766, 'ot
her_rate': 0.5174228328771912, 'max_depth': 11, 'reg_alpha': 5.6704066290912
2, 'reg_lambda': 8.881335787663785, 'min_split_gain': 5.720138556528263, 'min
_child_weight': 14.240631856467925, 'min_data_in_leaf': 19}. Best is trial 55
with value: 3.6195138490491945.
[I 2020-11-20 11:43:22,908] Trial 78 finished with value: 3.6182323466733446
 and parameters: {'seed': 27, 'num_leaves': 63, 'colsample_bytree': 0.9070872
176478442, 'subsample': 0.9656742882033594, 'top_rate': 0.8994492484045095,
 'other_rate': 0.730513268425651, 'max_depth': 12, 'reg_alpha': 7.30918044122
20185, 'reg_lambda': 1.4237717975525102, 'min_split_gain': 3.932633816976652
6, 'min_child_weight': 14.096079905251392, 'min_data_in_leaf': 17}. Best is t
rial 78 with value: 3.6182323466733446.
[I 2020-11-20 11:43:35,695] Trial 79 finished with value: 3.6602582991133015
 and parameters: {'seed': 28, 'num_leaves': 38, 'colsample_bytree': 0.9165691
825411275, 'subsample': 0.9442310918641139, 'top_rate': 0.897027170254475, 'o
ther_rate': 0.7877946879060685, 'max_depth': 12, 'reg_alpha': 7.1994158498885
18, 'reg_lambda': 1.46875711477046, 'min_split_gain': 2.9795722025955724, 'mi
n_child_weight': 12.02870184895352, 'min_data_in_leaf': 22}. Best is trial 78
with value: 3.6182323466733446.
[I 2020-11-20 11:43:51,940] Trial 80 finished with value: 3.6132012392791917
 and parameters: {'seed': 33, 'num_leaves': 63, 'colsample_bytree': 0.9017624
455273957, 'subsample': 0.9779142337410566, 'top_rate': 0.986337948472175, 'o
ther_rate': 0.7359180530844636, 'max_depth': 12, 'reg_alpha': 5.3749193878239
945, 'reg_lambda': 0.3490931567891934, 'min_split_gain': 3.8506781909794854,
 'min_child_weight': 14.082970526240134, 'min_data_in_leaf': 16}. Best is tri
al 80 with value: 3.6132012392791917.
[I 2020-11-20 11:44:07,799] Trial 81 finished with value: 3.6126987619350968
```

```
 and parameters: {'seed': 31, 'num_leaves': 63, 'colsample_bytree': 0.8979338
827329957, 'subsample': 0.9927456340489518, 'top_rate': 0.9905699796938134,
 'other_rate': 0.7364300446206179, 'max_depth': 12, 'reg_alpha': 4.5978144541
9756, 'reg_lambda': 0.3174774152633517, 'min_split_gain': 2.386613012021091,
 'min_child_weight': 13.975952911919718, 'min_data_in_leaf': 16}. Best is tri
al 81 with value: 3.6126987619350968.
[I 2020-11-20 11:44:24,651] Trial 82 finished with value: 3.612623939159734 a
nd parameters: {'seed': 33, 'num_leaves': 63, 'colsample_bytree': 0.969460295
4519327, 'subsample': 0.9958304427619159, 'top_rate': 0.9981677295120532, 'ot
her_rate': 0.8920045745035899, 'max_depth': 12, 'reg_alpha': 4.45228701098612
5, 'reg_lambda': 0.38202930570836197, 'min_split_gain': 2.214703421137724, 'm
in_child_weight': 10.904913337301425, 'min_data_in_leaf': 16}. Best is trial
 82 with value: 3.612623939159734.
[I 2020-11-20 11:44:40,145] Trial 83 finished with value: 3.613211421650535 a
nd parameters: {'seed': 34, 'num_leaves': 63, 'colsample_bytree': 0.967269981
2691256, 'subsample': 0.9923140036840906, 'top_rate': 0.9958448104417866, 'ot
her_rate': 0.9155942331375487, 'max_depth': 12, 'reg_alpha': 4.395792646955,
 'reg_lambda': 0.17015397972281573, 'min_split_gain': 1.8640750114648865, 'mi
n_child_weight': 13.960896445813404, 'min_data_in_leaf': 17}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:44:56,567] Trial 84 finished with value: 3.6254503224559573
 and parameters: {'seed': 33, 'num_leaves': 63, 'colsample_bytree': 0.9612974
432083191, 'subsample': 0.936126402860377, 'top_rate': 0.9944274866641242, 'o
ther_rate': 0.9344276391419485, 'max_depth': 12, 'reg_alpha': 4.4689556510209
33, 'reg_lambda': 4.354652392885198, 'min_split_gain': 1.8627836472462858, 'm
in_child_weight': 14.194524116442842, 'min_data_in_leaf': 17}. Best is trial
 82 with value: 3.612623939159734.
[I 2020-11-20 11:45:13,257] Trial 85 finished with value: 3.616418399034635 a
nd parameters: {'seed': 40, 'num_leaves': 63, 'colsample_bytree': 0.960251656
1599238, 'subsample': 0.9786743175822278, 'top_rate': 0.989416779846733, 'oth
er_rate': 0.9581389431940791, 'max_depth': 12, 'reg_alpha': 3.649650606585273
4, 'reg_lambda': 0.29318723896449583, 'min_split_gain': 1.8902661559210427,
 'min_child_weight': 10.933232357901417, 'min_data_in_leaf': 20}. Best is tri
al 82 with value: 3.612623939159734.
[I 2020-11-20 11:45:29,698] Trial 86 finished with value: 3.615681063110139 a
nd parameters: {'seed': 39, 'num_leaves': 63, 'colsample_bytree': 0.966791091
4163165, 'subsample': 0.9935454374070328, 'top_rate': 0.9979159445631516, 'ot
her_rate': 0.921418186599476, 'max_depth': 12, 'reg_alpha': 3.565204811188864
6, 'reg_lambda': 0.2166886639510294, 'min_split_gain': 1.9292416743322716, 'm
in_child_weight': 11.310190960114713, 'min_data_in_leaf': 20}. Best is trial
 82 with value: 3.612623939159734.
[I 2020-11-20 11:45:45,144] Trial 87 finished with value: 3.6157396307727336
 and parameters: {'seed': 40, 'num_leaves': 63, 'colsample_bytree': 0.9525776
971728748, 'subsample': 0.9766790382403323, 'top_rate': 0.985646678874527, 'o
ther_rate': 0.916490843541382, 'max_depth': 12, 'reg_alpha': 4.0116676508353
9, 'reg_lambda': 0.267332465801294, 'min_split_gain': 1.634956966472712, 'min
_child_weight': 10.262813081235294, 'min_data_in_leaf': 20}. Best is trial 82
with value: 3.612623939159734.
[I 2020-11-20 11:46:00,712] Trial 88 finished with value: 3.6158234487184457
 and parameters: {'seed': 39, 'num_leaves': 63, 'colsample_bytree': 0.9047037
385257021, 'subsample': 0.979213401048222, 'top_rate': 0.9946597414621514, 'o
ther_rate': 0.9010262057405918, 'max_depth': 12, 'reg_alpha': 3.5438408483484
194, 'reg_lambda': 0.27118659465634415, 'min_split_gain': 1.8131066523557966,
'min_child_weight': 10.659183271285197, 'min_data_in_leaf': 20}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:46:17,744] Trial 89 finished with value: 3.6165136297562865
 and parameters: {'seed': 40, 'num_leaves': 63, 'colsample_bytree': 0.9612698
```

```
922075772, 'subsample': 0.9758945392162641, 'top_rate': 0.9990195630538599,
 'other_rate': 0.9197405437952885, 'max_depth': 12, 'reg_alpha': 3.6468762044
054266, 'reg_lambda': 0.2849511471623439, 'min_split_gain': 1.803207415003207
8, 'min_child_weight': 10.900219774529312, 'min_data_in_leaf': 20}. Best is t
rial 82 with value: 3.612623939159734.
[I 2020-11-20 11:46:33,827] Trial 90 finished with value: 3.615049189877039 a
nd parameters: {'seed': 39, 'num_leaves': 63, 'colsample_bytree': 0.962264741
9735086, 'subsample': 0.9867525819064862, 'top_rate': 0.9980060647934621, 'ot
her_rate': 0.8954951977383979, 'max_depth': 12, 'reg_alpha': 3.79119753391757
7, 'reg_lambda': 0.26584805520287963, 'min_split_gain': 1.7258127579171993,
 'min_child_weight': 10.119482618875686, 'min_data_in_leaf': 20}. Best is tri
al 82 with value: 3.612623939159734.
[I 2020-11-20 11:46:49,445] Trial 91 finished with value: 3.6167699274084026
 and parameters: {'seed': 40, 'num_leaves': 63, 'colsample_bytree': 0.9617631
927327234, 'subsample': 0.9876562981960278, 'top_rate': 0.9979422056906377,
 'other_rate': 0.9186716823956459, 'max_depth': 12, 'reg_alpha': 3.6528934328
33728, 'reg_lambda': 0.27693790107622357, 'min_split_gain': 1.745425706049804
4, 'min_child_weight': 11.00879802771322, 'min_data_in_leaf': 20}. Best is tr
ial 82 with value: 3.612623939159734.
[I 2020-11-20 11:47:06,447] Trial 92 finished with value: 3.618065456491181 a
nd parameters: {'seed': 39, 'num_leaves': 63, 'colsample_bytree': 0.973230369
1897539, 'subsample': 0.934831936356825, 'top_rate': 0.9591717908915587, 'oth
er_rate': 0.8882402190441431, 'max_depth': 12, 'reg_alpha': 4.13034095255685
9, 'reg_lambda': 0.9274989670965752, 'min_split_gain': 1.3624853458469381, 'm
in_child_weight': 8.227925571088683, 'min_data_in_leaf': 21}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:47:22,652] Trial 93 finished with value: 3.61408933356355 an
d parameters: {'seed': 44, 'num_leaves': 64, 'colsample_bytree': 0.9430514799
701014, 'subsample': 0.8797690129131424, 'top_rate': 0.9971666328559987, 'oth
er_rate': 0.9898560436098859, 'max_depth': 12, 'reg_alpha': 2.707901821190752
4, 'reg_lambda': 0.2654856634679407, 'min_split_gain': 2.2074117829343742, 'm
in_child_weight': 6.14780261987084, 'min_data_in_leaf': 20}. Best is trial 82
with value: 3.612623939159734.
[I 2020-11-20 11:47:38,541] Trial 94 finished with value: 3.6148319523165178
 and parameters: {'seed': 44, 'num_leaves': 64, 'colsample_bytree': 0.9459700
121206858, 'subsample': 0.8809923273254849, 'top_rate': 0.9605066491480524,
 'other_rate': 0.9999450272845329, 'max_depth': 12, 'reg_alpha': 2.7591574410
009962, 'reg_lambda': 0.8315764585805521, 'min_split_gain': 2.15275385065642
6, 'min_child_weight': 5.940061215804692, 'min_data_in_leaf': 19}. Best is tr
ial 82 with value: 3.612623939159734.
[I 2020-11-20 11:47:54,164] Trial 95 finished with value: 3.612821424292281 a
nd parameters: {'seed': 46, 'num_leaves': 64, 'colsample_bytree': 0.939078316
6898488, 'subsample': 0.8829548132514197, 'top_rate': 0.959910422162962, 'oth
er_rate': 0.994030818787934, 'max_depth': 12, 'reg_alpha': 2.754235327423680
6, 'reg_lambda': 0.8373562706977726, 'min_split_gain': 2.1697679650770203, 'm
in_child_weight': 5.132164762514672, 'min_data_in_leaf': 16}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:48:11,721] Trial 96 finished with value: 3.6154292293891963
 and parameters: {'seed': 45, 'num_leaves': 64, 'colsample_bytree': 0.9379823
55406032, 'subsample': 0.8801810456201284, 'top_rate': 0.9536299481428516, 'o
ther_rate': 0.9902478489790922, 'max_depth': 12, 'reg_alpha': 2.8680016122604
53, 'reg_lambda': 1.1710269901592667, 'min_split_gain': 2.2365120501831974,
 'min_child_weight': 5.310692599703917, 'min_data_in_leaf': 17}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:48:27,790] Trial 97 finished with value: 3.6142644036385354
 and parameters: {'seed': 45, 'num_leaves': 64, 'colsample_bytree': 0.9985531
422312904, 'subsample': 0.8665576097089235, 'top_rate': 0.9580547214947748,
```

'other_rate': 0.9847257136231088, 'max_depth': 12, 'reg_alpha': 2.6538725495159126, 'reg_lambda': 1.2197296146331942, 'min_split_gain': 2.20212275780889, 'min_child_weight': 5.742979402019947, 'min_data_in_leaf': 16}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:48:43,940] Trial 98 finished with value: 3.614496762652692 and parameters: {'seed': 44, 'num_leaves': 64, 'colsample_bytree': 0.9994830106688533, 'subsample': 0.8735965916087377, 'top_rate': 0.9659943802913925, 'other_rate': 0.9999006029963408, 'max_depth': 12, 'reg_alpha': 2.7765345982074243, 'reg_lambda': 1.2207073055814583, 'min_split_gain': 2.4208729528139297, 'min_child_weight': 5.24861721606623, 'min_data_in_leaf': 17}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:48:55,611] Trial 99 finished with value: 3.6809770244716438 and parameters: {'seed': 43, 'num_leaves': 27, 'colsample_bytree': 0.9941385270096811, 'subsample': 0.8236590264183906, 'top_rate': 0.9606464157083472, 'other_rate': 0.8211739789035568, 'max_depth': 11, 'reg_alpha': 2.6651093509029993, 'reg_lambda': 0.8741856217605675, 'min_split_gain': 2.4002649785484835, 'min_child_weight': 5.4945021388494295, 'min_data_in_leaf': 16}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:49:13,119] Trial 100 finished with value: 3.62406627625804 and parameters: {'seed': 48, 'num_leaves': 59, 'colsample_bytree': 0.9963285932786345, 'subsample': 0.8898653495652055, 'top_rate': 0.8775593353292599, 'other_rate': 0.9640736113813748, 'max_depth': 12, 'reg_alpha': 3.1873361415553263, 'reg_lambda': 1.6310417270916726, 'min_split_gain': 2.761319859062182, 'min_child_weight': 3.632599569140643, 'min_data_in_leaf': 18}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:49:28,409] Trial 101 finished with value: 3.6143769111269317 and parameters: {'seed': 44, 'num_leaves': 64, 'colsample_bytree': 0.9409489399651397, 'subsample': 0.8559254344277458, 'top_rate': 0.9560460270974321, 'other_rate': 0.9813451389397747, 'max_depth': 12, 'reg_alpha': 2.830283781970446, 'reg_lambda': 1.1594460927454238, 'min_split_gain': 2.5216599945376363, 'min_child_weight': 5.608290832374698, 'min_data_in_leaf': 17}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:49:45,226] Trial 102 finished with value: 3.615904636548525 and parameters: {'seed': 31, 'num_leaves': 64, 'colsample_bytree': 0.9985175991502192, 'subsample': 0.8514085885002998, 'top_rate': 0.9688239016875781, 'other_rate': 0.8669169922157687, 'max_depth': 12, 'reg_alpha': 2.380046786769245, 'reg_lambda': 1.8121248213445134, 'min_split_gain': 0.873086498811597, 'min_child_weight': 1.4645123211073363, 'min_data_in_leaf': 16}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:49:59,898] Trial 103 finished with value: 3.6221533118294977 and parameters: {'seed': 43, 'num_leaves': 60, 'colsample_bytree': 0.9236285779017107, 'subsample': 0.8265492101998217, 'top_rate': 0.9259456817627697, 'other_rate': 0.9928955779101518, 'max_depth': 11, 'reg_alpha': 1.6378596517715962, 'reg_lambda': 1.103857798095123, 'min_split_gain': 2.2786445847535917, 'min_child_weight': 6.719745897172037, 'min_data_in_leaf': 17}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:50:16,510] Trial 104 finished with value: 3.6185562791828603 and parameters: {'seed': 36, 'num_leaves': 61, 'colsample_bytree': 0.8885088674431401, 'subsample': 0.912376587512356, 'top_rate': 0.8878300287238973, 'other_rate': 0.9429563581748166, 'max_depth': 12, 'reg_alpha': 4.412839465443759, 'reg_lambda': 0.8041058319407242, 'min_split_gain': 3.404902565285889, 'min_child_weight': 4.936772096995694, 'min_data_in_leaf': 18}. Best is trial 82 with value: 3.612623939159734.

[I 2020-11-20 11:50:32,026] Trial 105 finished with value: 3.6148929235703506 and parameters: {'seed': 46, 'num_leaves': 64, 'colsample_bytree': 0.9821726807950002, 'subsample': 0.8769627336385746, 'top_rate': 0.9501252268339526, 'other_rate': 0.9775693765616791, 'max_depth': 12, 'reg_alpha': 2.9778973037800

```
29, 'reg_lambda': 1.201951438268584, 'min_split_gain': 2.556244863589086, 'mi
n_child_weight': 8.189324193990725, 'min_data_in_leaf': 16}. Best is trial 82
with value: 3.612623939159734.
[I 2020-11-20 11:50:48,814] Trial 106 finished with value: 3.617348798710878
 and parameters: {'seed': 45, 'num_leaves': 64, 'colsample_bytree': 0.9864704
61918146, 'subsample': 0.8806742526759423, 'top_rate': 0.9430933480296185, 'o
ther_rate': 0.9727294961265585, 'max_depth': 12, 'reg_alpha': 2.7777951675834
496, 'reg_lambda': 2.168193001206789, 'min_split_gain': 2.649190226977911, 'm
in_child_weight': 7.306938465262275, 'min_data_in_leaf': 16}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:50:59,784] Trial 107 finished with value: 3.7007507675458116
and parameters: {'seed': 47, 'num_leaves': 62, 'colsample_bytree': 0.93816102
3103557, 'subsample': 0.783072780755175, 'top_rate': 0.9729448531503966, 'oth
er_rate': 0.9788557757784905, 'max_depth': 5, 'reg_alpha': 2.479978747024888
6, 'reg_lambda': 1.343753859832976, 'min_split_gain': 1.1464855453591736, 'mi
n_child_weight': 8.87874505584855, 'min_data_in_leaf': 59}. Best is trial 82
 with value: 3.612623939159734.
[I 2020-11-20 11:51:16,682] Trial 108 finished with value: 3.6214392412265926
and parameters: {'seed': 33, 'num_leaves': 60, 'colsample_bytree': 0.90443285
8210021, 'subsample': 0.8667803494712009, 'top_rate': 0.9172952874642359, 'ot
her_rate': 0.9970818617153181, 'max_depth': 11, 'reg_alpha': 3.00676597423066
2, 'reg_lambda': 1.0671419049748327, 'min_split_gain': 2.1026268166809974, 'm
in_child_weight': 1.973340017247827, 'min_data_in_leaf': 17}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:51:32,216] Trial 109 finished with value: 3.617239804840687
 and parameters: {'seed': 44, 'num_leaves': 64, 'colsample_bytree': 0.9785074
088129508, 'subsample': 0.9254460028253216, 'top_rate': 0.9603302455394925,
 'other_rate': 0.8638980542417647, 'max_depth': 12, 'reg_alpha': 2.1142850761
563547, 'reg_lambda': 1.6887588619408267, 'min_split_gain': 2.42279608135401
5, 'min_child_weight': 4.468343680172936, 'min_data_in_leaf': 19}. Best is tr
ial 82 with value: 3.612623939159734.
[I 2020-11-20 11:51:47,757] Trial 110 finished with value: 3.624638107818292
 and parameters: {'seed': 49, 'num_leaves': 61, 'colsample_bytree': 0.8534640
509932795, 'subsample': 0.9450258742751878, 'top_rate': 0.4380797175375649,
 'other_rate': 0.9537061739676865, 'max_depth': 11, 'reg_alpha': 1.9022059675
793472, 'reg_lambda': 2.6079770669978117, 'min_split_gain': 1.535537185478622
4, 'min_child_weight': 6.215944281295295, 'min_data_in_leaf': 16}. Best is tr
ial 82 with value: 3.612623939159734.
[I 2020-11-20 11:51:59,922] Trial 111 finished with value: 3.6725705483343356
and parameters: {'seed': 42, 'num_leaves': 31, 'colsample_bytree': 0.94612742
77145953, 'subsample': 0.8358532259282494, 'top_rate': 0.9999995223112184, 'o
ther_rate': 0.8107483999802998, 'max_depth': 12, 'reg_alpha': 3.2636551931844
47, 'reg_lambda': 0.7979898966068306, 'min_split_gain': 2.629285971394063, 'm
in_child_weight': 9.929882610175605, 'min_data_in_leaf': 18}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:52:16,748] Trial 112 finished with value: 3.6202383170890498
and parameters: {'seed': 46, 'num_leaves': 62, 'colsample_bytree': 0.99959598
67225803, 'subsample': 0.8049612906800561, 'top_rate': 0.9774483840686669, 'o
ther_rate': 0.8913565653292163, 'max_depth': 12, 'reg_alpha': 4.7840329532958
86, 'reg_lambda': 1.2389619645733683, 'min_split_gain': 2.0780927880569076,
 'min_child_weight': 7.932088508924313, 'min_data_in_leaf': 19}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:52:32,261] Trial 113 finished with value: 3.6287761328237296
and parameters: {'seed': 33, 'num_leaves': 64, 'colsample_bytree': 0.92003912
44286289, 'subsample': 0.9053226172358054, 'top_rate': 0.9515290961806357, 'o
ther_rate': 0.9987646305949229, 'max_depth': 12, 'reg_alpha': 2.5861760643573
657, 'reg_lambda': 0.6718328646179854, 'min_split_gain': 2.8925703998562566,
```

```
 'min_child_weight': 3.958652913982534, 'min_data_in_leaf': 44}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:52:47,605] Trial 114 finished with value: 3.617167119921279
 and parameters: {'seed': 31, 'num_leaves': 64, 'colsample_bytree': 0.8945171
744452063, 'subsample': 0.8774277469635865, 'top_rate': 0.9282470408055954,
 'other_rate': 0.944005195557062, 'max_depth': 12, 'reg_alpha': 5.31842465817
8107, 'reg_lambda': 1.9552133470683708, 'min_split_gain': 3.2354793350195945,
'min_child_weight': 6.077528149669005, 'min_data_in_leaf': 17}. Best is trial
82 with value: 3.612623939159734.
[I 2020-11-20 11:53:03,421] Trial 115 finished with value: 3.618374107959887
 and parameters: {'seed': 37, 'num_leaves': 62, 'colsample_bytree': 0.9467803
679698564, 'subsample': 0.9528232199008222, 'top_rate': 0.903139625304865, 'o
ther_rate': 0.8525045440839885, 'max_depth': 12, 'reg_alpha': 4.5585981015542
44, 'reg_lambda': 1.0132152995777919, 'min_split_gain': 1.4346976679494827,
 'min_child_weight': 8.757253561095705, 'min_data_in_leaf': 18}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:53:14,074] Trial 116 finished with value: 3.7209386864651073
and parameters: {'seed': 34, 'num_leaves': 60, 'colsample_bytree': 0.98878685
52692962, 'subsample': 0.7760114282609384, 'top_rate': 0.980185024102597, 'ot
her_rate': 0.7476883487651429, 'max_depth': 4, 'reg_alpha': 2.95448444398439
6, 'reg_lambda': 1.478618079272121, 'min_split_gain': 1.0301915571750477, 'mi
n_child_weight': 2.5455352332968073, 'min_data_in_leaf': 16}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:53:29,687] Trial 117 finished with value: 3.628924332302232
 and parameters: {'seed': 47, 'num_leaves': 57, 'colsample_bytree': 0.9708089
854147425, 'subsample': 0.8417384601840867, 'top_rate': 0.8592973006640774,
 'other_rate': 0.9792986209772877, 'max_depth': 11, 'reg_alpha': 3.8968212428
72415, 'reg_lambda': 0.8143139563015696, 'min_split_gain': 2.468095592301786,
'min_child_weight': 12.080383742555092, 'min_data_in_leaf': 21}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:53:45,160] Trial 118 finished with value: 3.620474420648953
 and parameters: {'seed': 42, 'num_leaves': 61, 'colsample_bytree': 0.9221828
385374315, 'subsample': 0.9238197009550456, 'top_rate': 0.9480777413564356,
 'other_rate': 0.8735437279993268, 'max_depth': 12, 'reg_alpha': 3.3553516754
023316, 'reg_lambda': 1.24524597395948, 'min_split_gain': 2.119653320003674,
 'min_child_weight': 9.791670271139287, 'min_data_in_leaf': 19}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:53:56,170] Trial 119 finished with value: 3.711848942574058
 and parameters: {'seed': 45, 'num_leaves': 64, 'colsample_bytree': 0.0705630
5589363976, 'subsample': 0.9081985460041081, 'top_rate': 0.8816330871680961,
 'other_rate': 0.9992600938602654, 'max_depth': 11, 'reg_alpha': 2.3409768725
949567, 'reg_lambda': 5.707386813506387, 'min_split_gain': 3.460634604930369
6, 'min_child_weight': 6.823536283839758, 'min_data_in_leaf': 22}. Best is tr
ial 82 with value: 3.612623939159734.
[I 2020-11-20 11:54:12,968] Trial 120 finished with value: 3.6157780944687348
and parameters: {'seed': 30, 'num_leaves': 62, 'colsample_bytree': 0.99888929
87411759, 'subsample': 0.810661316335203, 'top_rate': 0.9288587350587949, 'ot
her_rate': 0.9634573308953194, 'max_depth': 12, 'reg_alpha': 4.94212448965478
4, 'reg_lambda': 0.6426084651600336, 'min_split_gain': 2.779230886712017, 'mi
n_child_weight': 7.743432951853616, 'min_data_in_leaf': 16}. Best is trial 82
with value: 3.612623939159734.
[I 2020-11-20 11:54:29,244] Trial 121 finished with value: 3.6148388035434262
and parameters: {'seed': 45, 'num_leaves': 64, 'colsample_bytree': 0.93871238
14115081, 'subsample': 0.8879063194458789, 'top_rate': 0.9587654234296769, 'o
ther_rate': 0.9354429640519656, 'max_depth': 12, 'reg_alpha': 2.8491691484272
7, 'reg_lambda': 1.1244256079733987, 'min_split_gain': 2.1969298478974033, 'm
in_child_weight': 5.178717994192861, 'min_data_in_leaf': 17}. Best is trial 8
```

```
2 with value: 3.612623939159734.
[I 2020-11-20 11:54:44,264] Trial 122 finished with value: 3.6156565706065975
and parameters: {'seed': 44, 'num_leaves': 63, 'colsample_bytree': 0.88812651
1213568, 'subsample': 0.8675157127385843, 'top_rate': 0.9760174442113496, 'ot
her_rate': 0.9084076240162894, 'max_depth': 12, 'reg_alpha': 2.74519947439858
44, 'reg_lambda': 0.9750194008214266, 'min_split_gain': 2.6132277009617964,
 'min_child_weight': 4.456974843626666, 'min_data_in_leaf': 17}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:55:00,585] Trial 123 finished with value: 3.6140890965490957
and parameters: {'seed': 46, 'num_leaves': 64, 'colsample_bytree': 0.97546115
59584635, 'subsample': 0.9558874272433187, 'top_rate': 0.9971811038816554, 'o
ther_rate': 0.9339569694054914, 'max_depth': 12, 'reg_alpha': 3.058842651635
8, 'reg_lambda': 0.5247479502160209, 'min_split_gain': 0.5170054300765383, 'm
in_child_weight': 6.019006937882953, 'min_data_in_leaf': 18}. Best is trial 8
2 with value: 3.612623939159734.
[I 2020-11-20 11:55:17,230] Trial 124 finished with value: 3.6131027094286403
and parameters: {'seed': 49, 'num_leaves': 64, 'colsample_bytree': 0.93963896
24648225, 'subsample': 0.8920029032817035, 'top_rate': 0.9077781706184428, 'o
ther_rate': 0.9386214428193671, 'max_depth': 12, 'reg_alpha': 2.2660630275450
986, 'reg_lambda': 0.5379326183764594, 'min_split_gain': 0.22029972392547403,
'min_child_weight': 5.936895103509199, 'min_data_in_leaf': 18}. Best is trial
82 with value: 3.612623939159734.
[I 2020-11-20 11:55:33,173] Trial 125 finished with value: 3.6195469541926926
and parameters: {'seed': 50, 'num_leaves': 61, 'colsample_bytree': 0.94377402
78628369, 'subsample': 0.9003997941957751, 'top_rate': 0.9067155957406532, 'o
ther_rate': 0.9345060080569142, 'max_depth': 11, 'reg_alpha': 1.3435606897540
18, 'reg_lambda': 0.4925440417371296, 'min_split_gain': 0.2156592197763878,
 'min_child_weight': 2.853877122858788, 'min_data_in_leaf': 18}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:55:48,187] Trial 126 finished with value: 3.6156045655103175
and parameters: {'seed': 48, 'num_leaves': 64, 'colsample_bytree': 0.86243816
12368414, 'subsample': 0.9522673142303922, 'top_rate': 0.9702184599238696, 'o
ther_rate': 0.8326878778365578, 'max_depth': 12, 'reg_alpha': 1.8416533204001
68, 'reg_lambda': 0.5721913786408179, 'min_split_gain': 0.0967065548175121,
 'min_child_weight': 5.710271518819688, 'min_data_in_leaf': 19}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:56:03,835] Trial 127 finished with value: 3.616173865663859
 and parameters: {'seed': 49, 'num_leaves': 62, 'colsample_bytree': 0.9101208
065927966, 'subsample': 0.8932565128123672, 'top_rate': 0.9968165477892411,
 'other_rate': 0.9421823669283103, 'max_depth': 12, 'reg_alpha': 2.3167410791
891108, 'reg_lambda': 0.8367120625183725, 'min_split_gain': 0.501626527139251
7, 'min_child_weight': 4.087470160401158, 'min_data_in_leaf': 17}. Best is tr
ial 82 with value: 3.612623939159734.
[I 2020-11-20 11:56:20,158] Trial 128 finished with value: 3.625298502903591
 and parameters: {'seed': 44, 'num_leaves': 59, 'colsample_bytree': 0.9348664
98058224, 'subsample': 0.9176213923918642, 'top_rate': 0.9299737708477629, 'o
ther_rate': 0.9549816298320131, 'max_depth': 11, 'reg_alpha': 3.1272960683756
743, 'reg_lambda': 1.4058113585252467, 'min_split_gain': 0.6778668303572684,
 'min_child_weight': 7.038871495207201, 'min_data_in_leaf': 18}. Best is tria
l 82 with value: 3.612623939159734.
[I 2020-11-20 11:56:36,399] Trial 129 finished with value: 3.6104015176319613
and parameters: {'seed': 42, 'num_leaves': 64, 'colsample_bytree': 0.88871532
73254533, 'subsample': 0.9329118114749473, 'top_rate': 0.9535768954929632, 'o
ther_rate': 0.8838751719744662, 'max_depth': 12, 'reg_alpha': 2.6027678289155
98, 'reg_lambda': 0.01154310213896581, 'min_split_gain': 2.302694653018929,
 'min_child_weight': 0.6963898150017283, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
```

```
[I 2020-11-20 11:56:51,528] Trial 130 finished with value: 3.612830645112456
 and parameters: {'seed': 42, 'num_leaves': 63, 'colsample_bytree': 0.8781863
890632547, 'subsample': 0.960039520945442, 'top_rate': 0.8919127467698481, 'o
ther_rate': 0.7976806553985105, 'max_depth': 12, 'reg_alpha': 2.2162558550586
926, 'reg_lambda': 0.0784000866896114, 'min_split_gain': 1.2525901944213138,
 'min_child_weight': 0.19255198000709406, 'min_data_in_leaf': 16}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:57:07,961] Trial 131 finished with value: 3.61132396625989 a
nd parameters: {'seed': 37, 'num_leaves': 63, 'colsample_bytree': 0.888513764
0493378, 'subsample': 0.9355689157165301, 'top_rate': 0.8958364945540985, 'ot
her_rate': 0.7823206921286947, 'max_depth': 12, 'reg_alpha': 2.13748360394923
86, 'reg_lambda': 0.008170899085845, 'min_split_gain': 0.87920511244243, 'min
_child_weight': 0.1918336366178366, 'min_data_in_leaf': 16}. Best is trial 12
9 with value: 3.6104015176319613.
[I 2020-11-20 11:57:23,429] Trial 132 finished with value: 3.612407373147257
 and parameters: {'seed': 36, 'num_leaves': 63, 'colsample_bytree': 0.8865457
838201279, 'subsample': 0.9539934174738819, 'top_rate': 0.8349176323797047,
 'other_rate': 0.7646410965618019, 'max_depth': 12, 'reg_alpha': 2.1808317340
048875, 'reg_lambda': 0.03685555256181945, 'min_split_gain': 0.91412645439158
24, 'min_child_weight': 0.31915528024778705, 'min_data_in_leaf': 16}. Best is
trial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:57:38,854] Trial 133 finished with value: 3.6131568569504546
and parameters: {'seed': 36, 'num_leaves': 62, 'colsample_bytree': 0.86494892
13246212, 'subsample': 0.9635847675460688, 'top_rate': 0.8280187730976601, 'o
ther_rate': 0.7690764500344093, 'max_depth': 12, 'reg_alpha': 2.1737658559527
88, 'reg_lambda': 0.0742723964260768, 'min_split_gain': 0.93170698703412, 'mi
n_child_weight': 0.6428335655354936, 'min_data_in_leaf': 16}. Best is trial 1
29 with value: 3.6104015176319613.
[I 2020-11-20 11:57:53,574] Trial 134 finished with value: 3.6172771984280048
and parameters: {'seed': 35, 'num_leaves': 60, 'colsample_bytree': 0.85540335
95544414, 'subsample': 0.998613470474787, 'top_rate': 0.7696218025648808, 'ot
her_rate': 0.7661436008225974, 'max_depth': 12, 'reg_alpha': 1.18013945937983
9, 'reg_lambda': 0.21687855182336643, 'min_split_gain': 0.9401831699424015,
 'min_child_weight': 1.2482721814066047, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 11:58:08,741] Trial 135 finished with value: 3.61706646643502 a
nd parameters: {'seed': 37, 'num_leaves': 61, 'colsample_bytree': 0.893140193
2182055, 'subsample': 0.9556292417371929, 'top_rate': 0.831849205843529, 'oth
er_rate': 0.7048397318864887, 'max_depth': 11, 'reg_alpha': 2.212908260389837
3, 'reg_lambda': 0.06005784418687252, 'min_split_gain': 0.3602307385402243,
 'min_child_weight': 0.5837583945549533, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 11:58:24,702] Trial 136 finished with value: 3.614041703015134
 and parameters: {'seed': 35, 'num_leaves': 62, 'colsample_bytree': 0.8735034
827430878, 'subsample': 0.9390895094963592, 'top_rate': 0.8354964560518594,
 'other_rate': 0.7969954911792689, 'max_depth': 12, 'reg_alpha': 1.8797028509
251736, 'reg_lambda': 0.08262249577656444, 'min_split_gain': 1.21062281521844
26, 'min_child_weight': 0.5355987906081805, 'min_data_in_leaf': 16}. Best is
 trial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:58:32,002] Trial 137 finished with value: 3.811172383107802
 and parameters: {'seed': 35, 'num_leaves': 16, 'colsample_bytree': 0.8046024
494041275, 'subsample': 0.9362625224655551, 'top_rate': 0.8147157540741676,
 'other_rate': 0.8014947787327968, 'max_depth': 1, 'reg_alpha': 1.83597492706
10575, 'reg_lambda': 0.0174087400527416, 'min_split_gain': 1.254853789460939
7, 'min_child_weight': 0.1647133375417832, 'min_data_in_leaf': 18}. Best is t
rial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:58:47,625] Trial 138 finished with value: 3.613574521751529
```

```
 and parameters: {'seed': 32, 'num_leaves': 62, 'colsample_bytree': 0.8632966
277854366, 'subsample': 0.9685986154432414, 'top_rate': 0.7413705287444343,
 'other_rate': 0.7785816277095177, 'max_depth': 12, 'reg_alpha': 2.0009945513
073233, 'reg_lambda': 0.006333573301061782, 'min_split_gain': 0.5287709701091
603, 'min_child_weight': 0.07191402787324924, 'min_data_in_leaf': 16}. Best i
s trial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:59:01,904] Trial 139 finished with value: 3.619782212103831
 and parameters: {'seed': 34, 'num_leaves': 58, 'colsample_bytree': 0.8675584
395361401, 'subsample': 0.9677474028663333, 'top_rate': 0.7631316551891661,
 'other_rate': 0.7247553300316509, 'max_depth': 12, 'reg_alpha': 1.6145538175
451286, 'reg_lambda': 0.005352323432429225, 'min_split_gain': 0.8385881638902
535, 'min_child_weight': 2.0359085417085967, 'min_data_in_leaf': 16}. Best is
trial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:59:18,570] Trial 140 finished with value: 3.6165358980280713
and parameters: {'seed': 32, 'num_leaves': 62, 'colsample_bytree': 0.82599459
05552961, 'subsample': 0.9617887490209677, 'top_rate': 0.7209659286653561, 'o
ther_rate': 0.7817533894540937, 'max_depth': 11, 'reg_alpha': 1.9832231910622
886, 'reg_lambda': 0.0028608502261758227, 'min_split_gain': 0.50227399307252
4, 'min_child_weight': 0.21000912439112557, 'min_data_in_leaf': 16}. Best is
 trial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:59:33,481] Trial 141 finished with value: 3.615055628512639
 and parameters: {'seed': 36, 'num_leaves': 62, 'colsample_bytree': 0.8803432
52848042, 'subsample': 0.998808195348353, 'top_rate': 0.8341251069511054, 'ot
her_rate': 0.7615488321351971, 'max_depth': 12, 'reg_alpha': 2.10379342946743
36, 'reg_lambda': 0.37373277101206637, 'min_split_gain': 1.1652854681189786,
 'min_child_weight': 0.017452310032755852, 'min_data_in_leaf': 18}. Best is t
rial 129 with value: 3.6104015176319613.
[I 2020-11-20 11:59:49,018] Trial 142 finished with value: 3.6159296898233473
and parameters: {'seed': 37, 'num_leaves': 63, 'colsample_bytree': 0.84126666
15259728, 'subsample': 0.9356298777187079, 'top_rate': 0.8504858443555571, 'o
ther_rate': 0.7853426692975412, 'max_depth': 12, 'reg_alpha': 1.6792574444109
16, 'reg_lambda': 0.40845786249141985, 'min_split_gain': 0.7416401956969572,
 'min_child_weight': 1.4998914645596901, 'min_data_in_leaf': 19}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:00:03,906] Trial 143 finished with value: 3.6176722422408263
and parameters: {'seed': 34, 'num_leaves': 60, 'colsample_bytree': 0.91151329
32799214, 'subsample': 0.99748189403819, 'top_rate': 0.6755748557535618, 'oth
er_rate': 0.677978465147469, 'max_depth': 12, 'reg_alpha': 1.226295589116026
5, 'reg_lambda': 0.16290083410347803, 'min_split_gain': 0.006140218276495912,
'min_child_weight': 3.2699689488742223, 'min_data_in_leaf': 17}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:00:20,578] Trial 144 finished with value: 3.616993568401538
 and parameters: {'seed': 32, 'num_leaves': 61, 'colsample_bytree': 0.8605269
281629861, 'subsample': 0.966147462879262, 'top_rate': 0.7898182561751322, 'o
ther_rate': 0.8359195364491663, 'max_depth': 12, 'reg_alpha': 2.4514264922926
223, 'reg_lambda': 0.4957857636222229, 'min_split_gain': 1.4571096096479184,
 'min_child_weight': 1.0153202293933878, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:00:35,909] Trial 145 finished with value: 3.6146370339732545
and parameters: {'seed': 30, 'num_leaves': 63, 'colsample_bytree': 0.90078820
40634339, 'subsample': 0.9376850831876922, 'top_rate': 0.8252181444177819, 'o
ther_rate': 0.7384370410036573, 'max_depth': 12, 'reg_alpha': 2.2806671486517
867, 'reg_lambda': 0.1368037680340819, 'min_split_gain': 0.618839702947817,
 'min_child_weight': 2.0517831142142997, 'min_data_in_leaf': 18}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:00:50,793] Trial 146 finished with value: 3.621706796734681
 and parameters: {'seed': 35, 'num_leaves': 59, 'colsample_bytree': 0.8814234
```

```
420900635, 'subsample': 0.9774123778019065, 'top_rate': 0.8683705837295276,
 'other_rate': 0.7045929795337755, 'max_depth': 12, 'reg_alpha': 4.6246232363
28236, 'reg_lambda': 0.33101072853821334, 'min_split_gain': 1.056237898466474
6, 'min_child_weight': 0.14172610883638243, 'min_data_in_leaf': 19}. Best is
 trial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:01:06,593] Trial 147 finished with value: 3.6164291055312807
and parameters: {'seed': 32, 'num_leaves': 62, 'colsample_bytree': 0.80425176
69439966, 'subsample': 0.9275312582754157, 'top_rate': 0.8913003787684194, 'o
ther_rate': 0.7634429334207378, 'max_depth': 11, 'reg_alpha': 1.4292362469107
107, 'reg_lambda': 0.023884374550668358, 'min_split_gain': 1.333078085250934
7, 'min_child_weight': 13.623273622704, 'min_data_in_leaf': 17}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:01:22,211] Trial 148 finished with value: 3.6134871555650148
and parameters: {'seed': 36, 'num_leaves': 63, 'colsample_bytree': 0.91697220
72967341, 'subsample': 0.9489027193195598, 'top_rate': 0.5351006180949387, 'o
ther_rate': 0.8243011403204569, 'max_depth': 12, 'reg_alpha': 1.7992368048238
356, 'reg_lambda': 0.6179476939589257, 'min_split_gain': 0.24048997397810024,
'min_child_weight': 13.084610995075003, 'min_data_in_leaf': 16}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:01:37,525] Trial 149 finished with value: 3.6152486744337318
and parameters: {'seed': 38, 'num_leaves': 63, 'colsample_bytree': 0.83342375
5995344, 'subsample': 0.9986400858150061, 'top_rate': 0.5728572574594523, 'ot
her_rate': 0.8113750556428005, 'max_depth': 12, 'reg_alpha': 1.80079089115446
28, 'reg_lambda': 0.6250185229395395, 'min_split_gain': 0.2940942186215808,
 'min_child_weight': 12.694452719223667, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:01:52,774] Trial 150 finished with value: 3.6184406114358016
and parameters: {'seed': 38, 'num_leaves': 61, 'colsample_bytree': 0.91605914
69847956, 'subsample': 0.9493435221349145, 'top_rate': 0.502220038461867, 'ot
her_rate': 0.7957848432250373, 'max_depth': 11, 'reg_alpha': 4.31158331264641
5, 'reg_lambda': 0.5019511343037706, 'min_split_gain': 0.02737492677271236,
 'min_child_weight': 13.231420441984985, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:02:08,758] Trial 151 finished with value: 3.613415211212329
 and parameters: {'seed': 35, 'num_leaves': 63, 'colsample_bytree': 0.8726338
227329593, 'subsample': 0.9193042555136632, 'top_rate': 0.6276266000356618,
 'other_rate': 0.8363013039070314, 'max_depth': 12, 'reg_alpha': 2.0369987306
446666, 'reg_lambda': 0.21088753505842414, 'min_split_gain': 0.54028138283105
34, 'min_child_weight': 2.6518182478813843, 'min_data_in_leaf': 17}. Best is
 trial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:02:24,544] Trial 152 finished with value: 3.6133970203682737
and parameters: {'seed': 36, 'num_leaves': 63, 'colsample_bytree': 0.87262549
31026006, 'subsample': 0.9199707139132391, 'top_rate': 0.680498611066918, 'ot
her_rate': 0.8497848213038135, 'max_depth': 12, 'reg_alpha': 2.01190851346058
96, 'reg_lambda': 0.05252210679420332, 'min_split_gain': 0.5864530503830254,
 'min_child_weight': 2.6590636917686155, 'min_data_in_leaf': 18}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:02:39,903] Trial 153 finished with value: 3.613751106068311
 and parameters: {'seed': 36, 'num_leaves': 62, 'colsample_bytree': 0.8704160
381337351, 'subsample': 0.9277042936320409, 'top_rate': 0.6413818831939989,
 'other_rate': 0.8500915874353315, 'max_depth': 12, 'reg_alpha': 2.0146851923
53252, 'reg_lambda': 0.04268255205233032, 'min_split_gain': 0.777016722395517
3, 'min_child_weight': 2.614272723331003, 'min_data_in_leaf': 17}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:02:54,836] Trial 154 finished with value: 3.6128049464793954
and parameters: {'seed': 36, 'num_leaves': 63, 'colsample_bytree': 0.84805622
87044012, 'subsample': 0.9168502230923833, 'top_rate': 0.6387669313300137, 'o
```

```
ther_rate': 0.848434739690289, 'max_depth': 12, 'reg_alpha': 2.10247917931855
13, 'reg_lambda': 0.032598945696257535, 'min_split_gain': 0.7490462895308746,
'min_child_weight': 2.6096340745254074, 'min_data_in_leaf': 17}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:03:10,924] Trial 155 finished with value: 3.6137932711154908
and parameters: {'seed': 38, 'num_leaves': 63, 'colsample_bytree': 0.84178221
73386234, 'subsample': 0.9108720054394238, 'top_rate': 0.6033563435813953, 'o
ther_rate': 0.8228590347454251, 'max_depth': 12, 'reg_alpha': 1.0971318741468
947, 'reg_lambda': 0.2754653669686956, 'min_split_gain': 0.2411098599334044,
 'min_child_weight': 3.3643059540535836, 'min_data_in_leaf': 17}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:03:25,744] Trial 156 finished with value: 3.6193910644164102
and parameters: {'seed': 34, 'num_leaves': 60, 'colsample_bytree': 0.89994396
35403406, 'subsample': 0.9732245293340454, 'top_rate': 0.6231607304838084, 'o
ther_rate': 0.8849419393056646, 'max_depth': 12, 'reg_alpha': 1.5035793974527
418, 'reg_lambda': 0.35441367402685464, 'min_split_gain': 0.9630856579185612,
'min_child_weight': 15.622580510742301, 'min_data_in_leaf': 18}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:03:41,629] Trial 157 finished with value: 3.6144718557584175
and parameters: {'seed': 37, 'num_leaves': 63, 'colsample_bytree': 0.85822711
22300911, 'subsample': 0.9793795952068115, 'top_rate': 0.46590474434405943,
 'other_rate': 0.8383489070613684, 'max_depth': 12, 'reg_alpha': 2.2216002479
844406, 'reg_lambda': 0.04277319291969822, 'min_split_gain': 0.43621826604218
783, 'min_child_weight': 1.5509758635186839, 'min_data_in_leaf': 19}. Best is
trial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:03:56,536] Trial 158 finished with value: 3.620199086727749
 and parameters: {'seed': 33, 'num_leaves': 61, 'colsample_bytree': 0.8902249
93569276, 'subsample': 0.9167328759219585, 'top_rate': 0.6903488584084533, 'o
ther_rate': 0.7748990576753214, 'max_depth': 12, 'reg_alpha': 2.5229781632382
82, 'reg_lambda': 0.6831380925100838, 'min_split_gain': 0.6252920369613589,
 'min_child_weight': 12.198852431092321, 'min_data_in_leaf': 21}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:04:13,041] Trial 159 finished with value: 3.6155506522176055
and parameters: {'seed': 36, 'num_leaves': 63, 'colsample_bytree': 0.81295833
88585566, 'subsample': 0.9959659666876951, 'top_rate': 0.5436875099587721, 'o
ther_rate': 0.8695965447460783, 'max_depth': 11, 'reg_alpha': 0.7305730786753
677, 'reg_lambda': 0.19650099430315718, 'min_split_gain': 1.6420266843744165,
'min_child_weight': 2.3153118109949062, 'min_data_in_leaf': 16}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:04:27,942] Trial 160 finished with value: 3.616587990338638
 and parameters: {'seed': 30, 'num_leaves': 60, 'colsample_bytree': 0.9200057
428957898, 'subsample': 0.9999567928623562, 'top_rate': 0.6637116395710606,
 'other_rate': 0.8532376509748268, 'max_depth': 12, 'reg_alpha': 1.7298696774
621072, 'reg_lambda': 0.011051207657323772, 'min_split_gain': 0.7863091651722
991, 'min_child_weight': 1.3093925926207461, 'min_data_in_leaf': 17}. Best is
trial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:04:43,647] Trial 161 finished with value: 3.6144579650228774
and parameters: {'seed': 36, 'num_leaves': 62, 'colsample_bytree': 0.87297337
1149258, 'subsample': 0.9551379565573158, 'top_rate': 0.6366468100420121, 'ot
her_rate': 0.7477539334764123, 'max_depth': 12, 'reg_alpha': 2.21069988914919
82, 'reg_lambda': 0.016605724427417187, 'min_split_gain': 0.9214391491749055,
'min_child_weight': 2.69789748996606, 'min_data_in_leaf': 17}. Best is trial
 129 with value: 3.6104015176319613.
[I 2020-11-20 12:04:58,672] Trial 162 finished with value: 3.6139167296847754
and parameters: {'seed': 34, 'num_leaves': 62, 'colsample_bytree': 0.85890022
98204519, 'subsample': 0.9227826142175527, 'top_rate': 0.594776835696775, 'ot
her_rate': 0.8464560883588611, 'max_depth': 12, 'reg_alpha': 2.03898327139056
```

```
82, 'reg_lambda': 0.422684332413797, 'min_split_gain': 0.7071881254782968, 'm
in_child_weight': 3.5929896878622336, 'min_data_in_leaf': 16}. Best is trial
 129 with value: 3.6104015176319613.
[I 2020-11-20 12:05:15,688] Trial 163 finished with value: 3.614100090195996
 and parameters: {'seed': 36, 'num_leaves': 63, 'colsample_bytree': 0.8873426
913097038, 'subsample': 0.9284750969728975, 'top_rate': 0.6488658328058524,
 'other_rate': 0.8187205687964876, 'max_depth': 12, 'reg_alpha': 5.2711811470
28322, 'reg_lambda': 0.20447652029246172, 'min_split_gain': 0.299691213722590
33, 'min_child_weight': 14.730724522833265, 'min_data_in_leaf': 18}. Best is
 trial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:05:30,280] Trial 164 finished with value: 3.6171732191409824
and parameters: {'seed': 41, 'num_leaves': 61, 'colsample_bytree': 0.82573848
89913653, 'subsample': 0.9035364620024752, 'top_rate': 0.6672431922672587, 'o
ther_rate': 0.8796550191098036, 'max_depth': 12, 'reg_alpha': 1.9936758390653
087, 'reg_lambda': 0.0050934996114958, 'min_split_gain': 0.00495977419367466,
'min_child_weight': 0.005001592049972192, 'min_data_in_leaf': 17}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:05:45,759] Trial 165 finished with value: 3.616218812712992
 and parameters: {'seed': 32, 'num_leaves': 63, 'colsample_bytree': 0.8454866
217863092, 'subsample': 0.9697724644129341, 'top_rate': 0.7039667255331643,
 'other_rate': 0.8564029619465265, 'max_depth': 12, 'reg_alpha': 1.5395599012
272254, 'reg_lambda': 0.3927798191870095, 'min_split_gain': 1.152308856436747
4, 'min_child_weight': 2.446167626570525, 'min_data_in_leaf': 19}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:06:01,149] Trial 166 finished with value: 3.6123863522460202
and parameters: {'seed': 38, 'num_leaves': 64, 'colsample_bytree': 0.90674254
7882392, 'subsample': 0.9456620934677201, 'top_rate': 0.7463213556165356, 'ot
her_rate': 0.8073722501910648, 'max_depth': 12, 'reg_alpha': 2.52088012143333
7, 'reg_lambda': 0.6918765247529624, 'min_split_gain': 0.5119654665706241, 'm
in_child_weight': 0.9405520589809906, 'min_data_in_leaf': 16}. Best is trial
 129 with value: 3.6104015176319613.
[I 2020-11-20 12:06:18,149] Trial 167 finished with value: 3.613088191460911
 and parameters: {'seed': 38, 'num_leaves': 64, 'colsample_bytree': 0.9103906
675676358, 'subsample': 0.9485153033833166, 'top_rate': 0.7420306417820878,
 'other_rate': 0.7227308311721973, 'max_depth': 12, 'reg_alpha': 2.4078720883
508495, 'reg_lambda': 0.6720709139652945, 'min_split_gain': 0.456319820528510
3, 'min_child_weight': 0.9668502146689677, 'min_data_in_leaf': 16}. Best is t
rial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:06:33,251] Trial 168 finished with value: 3.633985176657116
 and parameters: {'seed': 38, 'num_leaves': 64, 'colsample_bytree': 0.9204743
319702298, 'subsample': 0.948969412535304, 'top_rate': 0.5888288836024864, 'o
ther_rate': 0.7222833345020367, 'max_depth': 11, 'reg_alpha': 2.5668466240851
49, 'reg_lambda': 0.5897034911251691, 'min_split_gain': 0.15075760434335245,
 'min_child_weight': 1.0554881963868326, 'min_data_in_leaf': 53}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:06:49,335] Trial 169 finished with value: 3.6137387965802406
and parameters: {'seed': 37, 'num_leaves': 64, 'colsample_bytree': 0.90415668
41149513, 'subsample': 0.8962338786911471, 'top_rate': 0.5513173228661967, 'o
ther_rate': 0.7565316710401244, 'max_depth': 12, 'reg_alpha': 2.4105802796949
205, 'reg_lambda': 0.7014268202005232, 'min_split_gain': 1.9764389665125435,
 'min_child_weight': 16.111748194485845, 'min_data_in_leaf': 18}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:06:56,980] Trial 170 finished with value: 3.7690708869778637
and parameters: {'seed': 41, 'num_leaves': 64, 'colsample_bytree': 0.92937909
85799272, 'subsample': 0.945379485748277, 'top_rate': 0.74138956348002, 'othe
r_rate': 0.802969048453741, 'max_depth': 2, 'reg_alpha': 2.389948913073979,
 'reg_lambda': 0.92572594263414, 'min_split_gain': 1.5619864684404887, 'min_c
```

```
hild_weight': 3.73354165986539, 'min_data_in_leaf': 16}. Best is trial 129 wi
th value: 3.6104015176319613.
[I 2020-11-20 12:07:13,450] Trial 171 finished with value: 3.6127236043637936
and parameters: {'seed': 35, 'num_leaves': 63, 'colsample_bytree': 0.89797882
1426806, 'subsample': 0.9801615970469659, 'top_rate': 0.6880740929890838, 'ot
her_rate': 0.7868349915577134, 'max_depth': 12, 'reg_alpha': 2.22046193346678
86, 'reg_lambda': 0.29986969366469496, 'min_split_gain': 0.467800057820007,
 'min_child_weight': 0.07076036223844273, 'min_data_in_leaf': 16}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:07:29,103] Trial 172 finished with value: 3.6139454972960756
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.89621764
83760886, 'subsample': 0.12227600789726567, 'top_rate': 0.686428774375178, 'o
ther_rate': 0.6836189465003767, 'max_depth': 12, 'reg_alpha': 2.2223528405259
847, 'reg_lambda': 0.7013725424294965, 'min_split_gain': 0.4358445763309601,
 'min_child_weight': 0.9272132399164966, 'min_data_in_leaf': 17}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:07:40,768] Trial 173 finished with value: 3.6312794131501396
and parameters: {'seed': 39, 'num_leaves': 63, 'colsample_bytree': 0.33906759
661401525, 'subsample': 0.9802920367167187, 'top_rate': 0.6152867681479552,
 'other_rate': 0.8237332582821743, 'max_depth': 12, 'reg_alpha': 1.7811912557
31489, 'reg_lambda': 0.28436686292276764, 'min_split_gain': 0.970862269909491
2, 'min_child_weight': 1.5465275131758722, 'min_data_in_leaf': 16}. Best is t
rial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:07:56,834] Trial 174 finished with value: 3.6118906977951464
and parameters: {'seed': 34, 'num_leaves': 64, 'colsample_bytree': 0.90962405
81238758, 'subsample': 0.9235878958238919, 'top_rate': 0.7447582736630566, 'o
ther_rate': 0.7380641889469328, 'max_depth': 12, 'reg_alpha': 2.5829335553671
99, 'reg_lambda': 0.48293939538847197, 'min_split_gain': 0.33201287972182836,
'min_child_weight': 0.03590308969000411, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:08:13,495] Trial 175 finished with value: 3.6126735442091564
and parameters: {'seed': 34, 'num_leaves': 64, 'colsample_bytree': 0.95616514
44256208, 'subsample': 0.9152676592716534, 'top_rate': 0.7279691170204116, 'o
ther_rate': 0.7193726144960985, 'max_depth': 12, 'reg_alpha': 2.5347814829693
09, 'reg_lambda': 0.412905184113752, 'min_split_gain': 0.6179329685811128, 'm
in_child_weight': 0.053939151371523136, 'min_data_in_leaf': 18}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:08:29,883] Trial 176 finished with value: 3.613781571249283
 and parameters: {'seed': 33, 'num_leaves': 64, 'colsample_bytree': 0.9565943
70291178, 'subsample': 0.9031570701252698, 'top_rate': 0.7447860510131934, 'o
ther_rate': 0.7149069276928545, 'max_depth': 12, 'reg_alpha': 2.4835337880772
62, 'reg_lambda': 0.4617252238997023, 'min_split_gain': 1.0784345457952211,
 'min_child_weight': 0.00500708559046581, 'min_data_in_leaf': 19}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:08:45,392] Trial 177 finished with value: 3.6167498450941236
and parameters: {'seed': 34, 'num_leaves': 64, 'colsample_bytree': 0.95765251
97726952, 'subsample': 0.997174527751748, 'top_rate': 0.7549215862447407, 'ot
her_rate': 0.6497879420487834, 'max_depth': 11, 'reg_alpha': 2.60477635030226
43, 'reg_lambda': 0.8999161367662389, 'min_split_gain': 0.804683158399957, 'm
in_child_weight': 0.034726078627362704, 'min_data_in_leaf': 18}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:09:01,107] Trial 178 finished with value: 3.619836462023509
 and parameters: {'seed': 38, 'num_leaves': 61, 'colsample_bytree': 0.9024999
030922741, 'subsample': 0.9612110540588498, 'top_rate': 0.7264648511705383,
 'other_rate': 0.7404152300890563, 'max_depth': 12, 'reg_alpha': 4.7700097628
33925, 'reg_lambda': 0.40868045026231486, 'min_split_gain': 0.395334432458837
5, 'min_child_weight': 1.0325753242154696, 'min_data_in_leaf': 20}. Best is t
```

```
rial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:09:17,248] Trial 179 finished with value: 3.615656436397712
 and parameters: {'seed': 37, 'num_leaves': 62, 'colsample_bytree': 0.9293251
600070305, 'subsample': 0.893956058997655, 'top_rate': 0.7826366980685437, 'o
ther_rate': 0.6913313172660649, 'max_depth': 12, 'reg_alpha': 5.0281923008638
78, 'reg_lambda': 0.755496455498952, 'min_split_gain': 1.3159278130590066, 'm
in_child_weight': 1.7680149510580923, 'min_data_in_leaf': 16}. Best is trial
 129 with value: 3.6104015176319613.
[I 2020-11-20 12:09:33,195] Trial 180 finished with value: 3.6125090465179373
and parameters: {'seed': 34, 'num_leaves': 64, 'colsample_bytree': 0.88559191
25766111, 'subsample': 0.9329300262301758, 'top_rate': 0.7055744246666165, 'o
ther_rate': 0.7374304716719917, 'max_depth': 12, 'reg_alpha': 2.2607810925721
092, 'reg_lambda': 0.22036529764930657, 'min_split_gain': 0.6778222403541206,
'min_child_weight': 0.11525923916248204, 'min_data_in_leaf': 18}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:09:48,214] Trial 181 finished with value: 3.6121839974923846
and parameters: {'seed': 34, 'num_leaves': 64, 'colsample_bytree': 0.89203009
08362695, 'subsample': 0.9343771907576768, 'top_rate': 0.6998381806143353, 'o
ther_rate': 0.7376112491952327, 'max_depth': 12, 'reg_alpha': 2.2403071967420
805, 'reg_lambda': 0.2444447170071627, 'min_split_gain': 0.6868410946746857,
 'min_child_weight': 0.00892630879194567, 'min_data_in_leaf': 18}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:10:04,168] Trial 182 finished with value: 3.611913383851322
 and parameters: {'seed': 33, 'num_leaves': 64, 'colsample_bytree': 0.8910097
740526204, 'subsample': 0.9388213136520465, 'top_rate': 0.7078808284510361,
 'other_rate': 0.7338367980214169, 'max_depth': 12, 'reg_alpha': 2.2732419256
698466, 'reg_lambda': 0.24195568096125697, 'min_split_gain': 0.00461605504993
4453, 'min_child_weight': 0.004946957487200199, 'min_data_in_leaf': 17}. Best
is trial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:10:16,890] Trial 183 finished with value: 3.6288591595888575
and parameters: {'seed': 33, 'num_leaves': 64, 'colsample_bytree': 0.38449800
95901576, 'subsample': 0.9387836571763578, 'top_rate': 0.7104216527088177, 'o
ther_rate': 0.7393666598490808, 'max_depth': 12, 'reg_alpha': 2.3586370163138
87, 'reg_lambda': 0.5046181988223291, 'min_split_gain': 0.02662016479125485,
 'min_child_weight': 0.12815596981210287, 'min_data_in_leaf': 19}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:10:32,270] Trial 184 finished with value: 3.610756993028083
 and parameters: {'seed': 31, 'num_leaves': 64, 'colsample_bytree': 0.8914280
19394793, 'subsample': 0.9622072717256234, 'top_rate': 0.7718657068232073, 'o
ther_rate': 0.7212798577699672, 'max_depth': 12, 'reg_alpha': 2.6124323892361
32, 'reg_lambda': 0.29374227267841385, 'min_split_gain': 0.6954352879155458,
 'min_child_weight': 0.059635473174676434, 'min_data_in_leaf': 16}. Best is t
rial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:10:43,664] Trial 185 finished with value: 3.6378381497584362
and parameters: {'seed': 31, 'num_leaves': 64, 'colsample_bytree': 0.27372794
36792267, 'subsample': 0.9343286235679635, 'top_rate': 0.7740778733725366, 'o
ther_rate': 0.7172774532195758, 'max_depth': 12, 'reg_alpha': 2.5651015691771
977, 'reg_lambda': 0.27385695452629366, 'min_split_gain': 0.6808403692861131,
'min_child_weight': 0.6790720415689553, 'min_data_in_leaf': 17}. Best is tria
l 129 with value: 3.6104015176319613.
[I 2020-11-20 12:10:58,735] Trial 186 finished with value: 3.6138568480062303
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.88675477
17687431, 'subsample': 0.9573287207088215, 'top_rate': 0.744066105463183, 'ot
her_rate': 0.6628108172343599, 'max_depth': 12, 'reg_alpha': 2.26322285335837
4, 'reg_lambda': 0.581411067596658, 'min_split_gain': 0.19370157129408688, 'm
in_child_weight': 0.11335355391438062, 'min_data_in_leaf': 18}. Best is trial
129 with value: 3.6104015176319613.
```

```
[I 2020-11-20 12:11:15,605] Trial 187 finished with value: 3.6144319832602205
and parameters: {'seed': 31, 'num_leaves': 62, 'colsample_bytree': 0.91868871
7471884, 'subsample': 0.9137419366316921, 'top_rate': 0.6968055311744182, 'ot
her_rate': 0.7648545448464216, 'max_depth': 12, 'reg_alpha': 2.62077898168724
13, 'reg_lambda': 0.24915285749499133, 'min_split_gain': 0.888221514941482,
 'min_child_weight': 1.0093276597022067, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:11:30,384] Trial 188 finished with value: 3.6167879188008536
and parameters: {'seed': 29, 'num_leaves': 64, 'colsample_bytree': 0.88768894
28021697, 'subsample': 0.9406545032535684, 'top_rate': 0.8031736267959797, 'o
ther_rate': 0.6972621677808799, 'max_depth': 11, 'reg_alpha': 2.8256692010110
034, 'reg_lambda': 0.7939628578006523, 'min_split_gain': 0.3692215640689614,
 'min_child_weight': 1.8052654564648971, 'min_data_in_leaf': 18}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:11:41,219] Trial 189 finished with value: 3.7012011375983653
and parameters: {'seed': 34, 'num_leaves': 18, 'colsample_bytree': 0.92978981
29820105, 'subsample': 0.893822089293031, 'top_rate': 0.7287881284692834, 'ot
her_rate': 0.7824948602475839, 'max_depth': 12, 'reg_alpha': 2.16786805537149
75, 'reg_lambda': 0.015884651936995203, 'min_split_gain': 0.700862616848975,
 'min_child_weight': 0.1949666948370814, 'min_data_in_leaf': 16}. Best is tri
al 129 with value: 3.6104015176319613.
[I 2020-11-20 12:11:56,036] Trial 190 finished with value: 3.6174603116502038
and parameters: {'seed': 33, 'num_leaves': 62, 'colsample_bytree': 0.85124869
40340299, 'subsample': 0.9773073053551776, 'top_rate': 0.7123943343492481, 'o
ther_rate': 0.7494367282991532, 'max_depth': 12, 'reg_alpha': 2.4294843633484
886, 'reg_lambda': 0.4470652566113876, 'min_split_gain': 0.4114581171397209,
 'min_child_weight': 0.06593829410806074, 'min_data_in_leaf': 20}. Best is tr
ial 129 with value: 3.6104015176319613.
[I 2020-11-20 12:12:13,187] Trial 191 finished with value: 3.609960170625465
 and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.9052035
604121135, 'subsample': 0.9604180879173653, 'top_rate': 0.763101374445726, 'o
ther_rate': 0.7253051136445952, 'max_depth': 12, 'reg_alpha': 2.2165085238993
47, 'reg_lambda': 0.24313348715238994, 'min_split_gain': 0.990691442681803,
 'min_child_weight': 1.0183266781956646, 'min_data_in_leaf': 16}. Best is tri
al 191 with value: 3.609960170625465.
[I 2020-11-20 12:12:28,633] Trial 192 finished with value: 3.6107973527710375
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.90630407
28442225, 'subsample': 0.9656529335816049, 'top_rate': 0.7785157402120647, 'o
ther_rate': 0.7225801514792585, 'max_depth': 12, 'reg_alpha': 2.2264906673375
83, 'reg_lambda': 0.18830700768251324, 'min_split_gain': 1.0340966564051395,
 'min_child_weight': 1.2843038429151434, 'min_data_in_leaf': 17}. Best is tri
al 191 with value: 3.609960170625465.
[I 2020-11-20 12:12:44,716] Trial 193 finished with value: 3.6122092516686894
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.91274617
10761061, 'subsample': 0.9323664771280245, 'top_rate': 0.7599430776614492, 'o
ther_rate': 0.7206744086017621, 'max_depth': 12, 'reg_alpha': 2.6357092920941
803, 'reg_lambda': 0.23056557477674652, 'min_split_gain': 0.6543133838750079,
'min_child_weight': 1.5459509898722508, 'min_data_in_leaf': 17}. Best is tria
l 191 with value: 3.609960170625465.
[I 2020-11-20 12:13:00,044] Trial 194 finished with value: 3.6118818937287993
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.90471396
75129374, 'subsample': 0.928493460479701, 'top_rate': 0.7671631089678848, 'ot
her_rate': 0.7220991685031591, 'max_depth': 12, 'reg_alpha': 2.99830346986673
4, 'reg_lambda': 0.2510312333881063, 'min_split_gain': 1.070009169950223, 'mi
n_child_weight': 1.2897472343611502, 'min_data_in_leaf': 17}. Best is trial 1
91 with value: 3.609960170625465.
[I 2020-11-20 12:13:17,024] Trial 195 finished with value: 3.6120435981259513
```

```
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.89465143
12513936, 'subsample': 0.9286833796646018, 'top_rate': 0.7658582692123225, 'o
ther_rate': 0.7071318156736909, 'max_depth': 12, 'reg_alpha': 2.6998461767113
016, 'reg_lambda': 0.22243597060154327, 'min_split_gain': 1.143907236735684,
 'min_child_weight': 1.7492072735072253, 'min_data_in_leaf': 17}. Best is tri
al 191 with value: 3.609960170625465.
[I 2020-11-20 12:13:31,872] Trial 196 finished with value: 3.6121450411815306
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.90121082
56296126, 'subsample': 0.9217790833396841, 'top_rate': 0.7676758415081935, 'o
ther_rate': 0.7013316325161532, 'max_depth': 12, 'reg_alpha': 2.9770289609695
144, 'reg_lambda': 0.29236882553346377, 'min_split_gain': 1.0471514211058544,
'min_child_weight': 1.8388610636003182, 'min_data_in_leaf': 17}. Best is tria
l 191 with value: 3.609960170625465.
[I 2020-11-20 12:13:47,868] Trial 197 finished with value: 3.6137900743502196
and parameters: {'seed': 35, 'num_leaves': 64, 'colsample_bytree': 0.89926085
43883316, 'subsample': 0.92049295547667, 'top_rate': 0.7674026188194607, 'oth
er_rate': 0.6694007851443294, 'max_depth': 12, 'reg_alpha': 3.04823422258294
5, 'reg_lambda': 0.22067355555914897, 'min_split_gain': 1.0664422767209205,
 'min_child_weight': 1.9546141081699824, 'min_data_in_leaf': 19}. Best is tri
al 191 with value: 3.609960170625465.
[I 2020-11-20 12:14:03,044] Trial 198 finished with value: 3.6132289624436966
and parameters: {'seed': 34, 'num_leaves': 63, 'colsample_bytree': 0.90624623
30688337, 'subsample': 0.9249633963591581, 'top_rate': 0.7842998971175419, 'o
ther_rate': 0.6990510993264599, 'max_depth': 12, 'reg_alpha': 2.6326817669852
09, 'reg_lambda': 0.33903482808784147, 'min_split_gain': 1.3849140074548973,
 'min_child_weight': 1.7128644588517, 'min_data_in_leaf': 17}. Best is trial
 191 with value: 3.609960170625465.
[I 2020-11-20 12:14:20,364] Trial 199 finished with value: 3.613973449397736
 and parameters: {'seed': 32, 'num_leaves': 64, 'colsample_bytree': 0.8886650
998978671, 'subsample': 0.9995542968608077, 'top_rate': 0.8028113023231666,
 'other_rate': 0.7301572238331935, 'max_depth': 11, 'reg_alpha': 3.4153971349
129124, 'reg_lambda': 0.2606239396753332, 'min_split_gain': 0.81665374988644
1, 'min_child_weight': 3.042027060316875, 'min_data_in_leaf': 17}. Best is tr
ial 191 with value: 3.609960170625465.
```

In [3]:
```
target=train['target']
train = train.drop(['card_id', 'first_active_month'], axis = 1)
test = test.drop(['card_id', 'first_active_month'], axis = 1)
```

In [24]:
```
test.shape
```

Out[24]:
```
(123623, 193)
```

In [28]:
```
feature.remove('first_active_month')
feature.remove('card_id')
```

In [ ]:
```
feature = [c for c in train.columns if c not in ['first_active_month', 'targe
t', 'card_id', 'outliers',
                    'hist_purchase_date_max', 'hist_purchase_date_min', 'hist_ca
rd_id_size',
                    'new_purchase_date_max', 'new_purchase_date_min', 'new_card_
id_size']]
target = train['target']
```

LGBM Regressor

In [21]:
```
from sklearn.model_selection import StratifiedKFold
import lightgbm as lgbm

from sklearn.metrics import mean_squared_error

oof = np.zeros(len(train))
predictions = np.zeros(len(test))

skf = StratifiedKFold(n_splits=5,shuffle = True, random_state=4950)

fold=0
for train_index, test_index in skf.split(train.values, train['outliers'].value
s):
    X_train, Y_train = train.iloc[train_index][feature], target.iloc[train_ind
ex]
    X_test, Y_test = train.iloc[test_index][feature], target.iloc[test_index]


    data_train=lgbm.Dataset(X_train,label=Y_train)
    data_test=lgbm.Dataset(X_test,label=Y_test)

  #after tweaking the few parameter ,obtained using optuna
    param ={
                'task': 'train',
                'boosting': 'goss',
                'objective': 'regression',
                'metric': 'rmse',
                'learning_rate': 0.01,
                'subsample': 0.9855232997390695,
                'max_depth': 7,
                'top_rate': 0.9064148448434349,
                'num_leaves': 63,
                'min_child_weight': 41.9612869171337,
                'other_rate': 0.0721768246018207,
                'reg_alpha': 9.677537745007898,
                'colsample_bytree': 0.5665320670155495,
                'min_split_gain': 9.820197773625843,
                'reg_lambda': 8.2532317400459,
                'min_data_in_leaf': 21,
                'verbose': -1,
                'seed':int(2**fold),
                'bagging_seed':int(2**fold),
                'drop_seed':int(2**fold)
                }

    num_round = 10000

    model = lgbm.train(param, data_train, num_round, valid_sets = [data_train,
data_test],
                       verbose_eval=-1, early_stopping_rounds = 200)

    oof[test_index] = model.predict(train.iloc[test_index][feature], num_itera
tion=model.best_iteration)


    predictions += model.predict(test[feature], num_iteration=model.best_itera
```

```
tion) / skf.n_splits

    fold=fold+1

np.sqrt(mean_squared_error(oof, target))
```

```
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1037]  training's rmse: 3.3741 valid_1's rmse: 3.65358
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1397]  training's rmse: 3.32412        valid_1's rmse: 3.65022
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[841]   training's rmse: 3.4003 valid_1's rmse: 3.66828
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1185]  training's rmse: 3.35631        valid_1's rmse: 3.6489
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[873]   training's rmse: 3.4009 valid_1's rmse: 3.65938
```

Out[21]: 3.6560778694866767

In [22]:
```
from sklearn.model_selection import RepeatedKFold
import lightgbm as lgbm

from sklearn.metrics import mean_squared_error

oof_rkfold = np.zeros(len(train))
predictions_rkfold = np.zeros(len(test))

rkf = RepeatedKFold(n_splits=5, n_repeats=2, random_state=4950)

fold=0
for train_index, test_index in rkf.split(train.values, target.values):
    X_train, Y_train = train.iloc[train_index][feature], target.iloc[train_ind
ex]
    X_test, Y_test = train.iloc[test_index][feature], target.iloc[test_index]


    data_train=lgbm.Dataset(X_train,label=Y_train)
    data_test=lgbm.Dataset(X_test,label=Y_test)

    print("fold:",fold)

    param ={
                'task': 'train',
                'boosting': 'goss',
                'objective': 'regression',
                'metric': 'rmse',
                'learning_rate': 0.01,
                'subsample': 0.9855232997390695,
                'max_depth': 7,
                'top_rate': 0.9064148448434349,
                'num_leaves': 63,
                'min_child_weight': 41.9612869171337,
                'other_rate': 0.0721768246018207,
                'reg_alpha': 9.677537745007898,
                'colsample_bytree': 0.5665320670155495,
                'min_split_gain': 9.820197773625843,
                'reg_lambda': 8.2532317400459,
                'min_data_in_leaf': 21,
                'verbose': -1,
                'seed':int(2**fold),
                'bagging_seed':int(2**fold),
                'drop_seed':int(2**fold)
                }

    num_round = 10000

    models = lgbm.train(param, data_train, num_round, valid_sets = [data_train
, data_test],
                    verbose_eval=-1, early_stopping_rounds = 200)

    oof_rkfold[test_index] = models.predict(train.iloc[test_index][feature], n
um_iteration=models.best_iteration)

    #feature=[c for c in train.columns if c not in ['outliers']]
    predictions_rkfold += models.predict(test[feature], num_iteration=models.b
```

```
est_iteration) / (5*2)

    fold=fold+1

np.sqrt(mean_squared_error(oof_rkfold, target))
```

```
fold: 0
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1017]  training's rmse: 3.37683        valid_1's rmse: 3.667
fold: 1
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1025]  training's rmse: 3.34465        valid_1's rmse: 3.80252
fold: 2
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[702]   training's rmse: 3.45443        valid_1's rmse: 3.56279
fold: 3
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1082]  training's rmse: 3.36701        valid_1's rmse: 3.61899
fold: 4
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[992]   training's rmse: 3.38719        valid_1's rmse: 3.61939
fold: 5
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[869]   training's rmse: 3.38518        valid_1's rmse: 3.73264
fold: 6
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[933]   training's rmse: 3.39116        valid_1's rmse: 3.63362
fold: 7
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[803]   training's rmse: 3.42276        valid_1's rmse: 3.63999
fold: 8
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1034]  training's rmse: 3.36849        valid_1's rmse: 3.6683
fold: 9
Training until validation scores don't improve for 200 rounds
Early stopping, best iteration is:
[1062]  training's rmse: 3.38022        valid_1's rmse: 3.59812
```

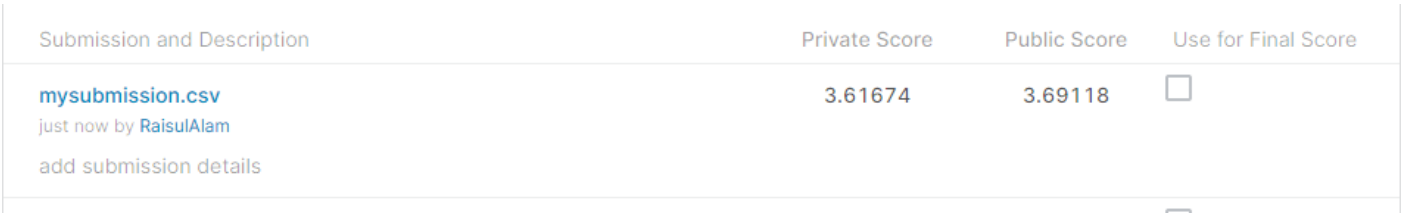
Out[22]: 3.6548108906553756

In [23]:
```
from sklearn.linear_model import BayesianRidge

train_stack = np.vstack([oof,oof_rkfold]).transpose()
test_stack = np.vstack([predictions, predictions_rkfold]).transpose()

folds_stack = RepeatedKFold(n_splits=5, n_repeats=1, random_state=4590)
oof_stack = np.zeros(train_stack.shape[0])
predictions_3 = np.zeros(test_stack.shape[0])

for fold_, (trn_idx, val_idx) in enumerate(folds_stack.split(train_stack,targe
t)):
    print("fold {}".format(fold_))
    trn_data, trn_y = train_stack[trn_idx], target.iloc[trn_idx].values
    val_data, val_y = train_stack[val_idx], target.iloc[val_idx].values

    model = BayesianRidge()
    model.fit(trn_data, trn_y)

    oof_stack[val_idx] = model.predict(val_data)
    predictions_3 += model.predict(test_stack) / 5

np.sqrt(mean_squared_error(target.values, oof_stack))
```

```
fold 0
fold 1
fold 2
fold 3
fold 4
```

Out[23]:
```
3.6530326031203604
```

In [25]:
```
sample_submission = pd.read_csv('../input/elo-merchant-category-recommendatio
n/sample_submission.csv')
sample_submission['target'] = predictions_3
sample_submission.to_csv('mysubmission.csv', index=False)
```



XGBOOST Regressor

In [19]:
```
import xgboost as xgb
```

In [20]:
```
model_xgb =  xgb.XGBRegressor(max_depth=2,
                              colsample_bytree=0.7,
                              n_estimators=20000,
                              scale_pos_weight = 9,
                              learning_rate=0.02,
                              min_child_weight=1.5,
                              #max_depth=3,
                              reg_alpha=0.75,
                              reg_lambda=0.45,
                              verbosity =1,
                              eval_metric  = 'rmse',
                              tree_method='gpu_hist',
                              n_jobs=-1)
```

In [22]:
```
target
```

Out[22]:
```
0         -0.820312
1          0.392822
2          0.687988
3          0.142456
4         -0.159790
              ...
201912    -2.740234
201913     0.312988
201914     0.093506
201915    -4.675781
201916    -1.859375
Name: target, Length: 201917, dtype: float16
```

In [32]:
```
from sklearn.model_selection import StratifiedKFold
import lightgbm as lgbm

import gc
from sklearn.metrics import mean_squared_error

oof = np.zeros(len(train))
predictions = np.zeros(len(test))

folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=1234)
for fold_index, (train_index,val_index) in enumerate(folds.split(train.values,
train['outliers'].values)):
    print("fold:",fold_index,"started")
    gc.collect()
    bst = model_xgb.fit(train.iloc[train_index][feature],target.iloc[train_ind
ex],
                eval_set = [(train.iloc[val_index][feature],target.iloc[val_inde
x])],
                early_stopping_rounds=200,
                verbose= 200,
                eval_metric ='rmse'
                )
    #val[val_index] = model_xgb.predict_proba(train[val_index])[:,1]
    #print('auc of this val set is {}'.format(roc_auc_score(y[val_index],val[v
al_index])))
    #pred += model_xgb.predict(test.values)[:,1]/folds.n_splits

    oof[val_index] = bst.predict(train.iloc[val_index][feature])
    predictions += bst.predict(test[feature]) / folds.n_splits

np.sqrt(mean_squared_error(oof, target))
```

```
fold: 0 started
[0]     validation_0-rmse:3.94124
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.72281
[400]   validation_0-rmse:3.70677
[600]   validation_0-rmse:3.69799
[800]   validation_0-rmse:3.69340
[1000]  validation_0-rmse:3.69035
[1200]  validation_0-rmse:3.68792
[1400]  validation_0-rmse:3.68549
[1600]  validation_0-rmse:3.68345
[1800]  validation_0-rmse:3.68251
[2000]  validation_0-rmse:3.68143
[2200]  validation_0-rmse:3.68019
[2400]  validation_0-rmse:3.67942
[2600]  validation_0-rmse:3.67889
[2800]  validation_0-rmse:3.67861
[3000]  validation_0-rmse:3.67821
[3200]  validation_0-rmse:3.67798
[3400]  validation_0-rmse:3.67793
[3600]  validation_0-rmse:3.67780
Stopping. Best iteration:
[3451]  validation_0-rmse:3.67760

fold: 1 started
[0]     validation_0-rmse:3.93904
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.72049
[400]   validation_0-rmse:3.70606
[600]   validation_0-rmse:3.69851
[800]   validation_0-rmse:3.69425
[1000]  validation_0-rmse:3.69140
[1200]  validation_0-rmse:3.68895
[1400]  validation_0-rmse:3.68694
[1600]  validation_0-rmse:3.68562
[1800]  validation_0-rmse:3.68429
[2000]  validation_0-rmse:3.68323
[2200]  validation_0-rmse:3.68188
[2400]  validation_0-rmse:3.68101
[2600]  validation_0-rmse:3.68029
[2800]  validation_0-rmse:3.67963
[3000]  validation_0-rmse:3.67928
[3200]  validation_0-rmse:3.67890
[3400]  validation_0-rmse:3.67866
Stopping. Best iteration:
[3281]  validation_0-rmse:3.67854

fold: 2 started
[0]     validation_0-rmse:3.94855
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.72815
[400]   validation_0-rmse:3.71303
[600]   validation_0-rmse:3.70578
[800]   validation_0-rmse:3.70169
[1000]  validation_0-rmse:3.69858
[1200]  validation_0-rmse:3.69630
[1400]  validation_0-rmse:3.69438
```

```
[1600]  validation_0-rmse:3.69288
[1800]  validation_0-rmse:3.69174
[2000]  validation_0-rmse:3.69092
[2200]  validation_0-rmse:3.69003
[2400]  validation_0-rmse:3.68962
Stopping. Best iteration:
[2396]  validation_0-rmse:3.68944

fold: 3 started
[0]     validation_0-rmse:3.95664
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.73892
[400]   validation_0-rmse:3.72441
[600]   validation_0-rmse:3.73701
Stopping. Best iteration:
[513]   validation_0-rmse:3.71976

fold: 4 started
[0]     validation_0-rmse:3.94394
Will train until validation_0-rmse hasn't improved in 200 rounds.
[200]   validation_0-rmse:3.74272
[400]   validation_0-rmse:3.72894
[600]   validation_0-rmse:3.72323
[800]   validation_0-rmse:3.71997
[1000]  validation_0-rmse:3.71748
[1200]  validation_0-rmse:3.71560
[1400]  validation_0-rmse:3.71329
[1600]  validation_0-rmse:3.71148
[1800]  validation_0-rmse:3.70996
[2000]  validation_0-rmse:3.70888
[2200]  validation_0-rmse:3.70837
[2400]  validation_0-rmse:3.70814
[2600]  validation_0-rmse:3.70767
[2800]  validation_0-rmse:3.70744
[3000]  validation_0-rmse:3.70710
[3200]  validation_0-rmse:3.70701
[3400]  validation_0-rmse:3.70697
[3600]  validation_0-rmse:3.70672
[3800]  validation_0-rmse:3.70647
[4000]  validation_0-rmse:3.70638
[4200]  validation_0-rmse:3.70634
[4400]  validation_0-rmse:3.70607
[4600]  validation_0-rmse:3.70594
[4800]  validation_0-rmse:3.70569
Stopping. Best iteration:
[4787]  validation_0-rmse:3.70566
```

Out[32]: 3.694238229091928

In [33]:
```
sample_submission = pd.read_csv('../input/elo-merchant-category-recommendation/sample_submission.csv')
sample_submission['target'] = predictions
sample_submission.to_csv('xgb_mysubmission.csv', index=False)
```

xgb_mysubmission.csv 8 hours ago by RaisulAlam add submission details	3.64933	3.75041	☐

MLP Architecture

In [36]:
```
from sklearn.model_selection import train_test_split

X_Train, X_Test, Y_Train, Y_Test = train_test_split(train[feature], target, te
st_size=0.30, random_state=42)
```

In [54]:
```
from keras import backend as K
def root_mean_squared_error(y_true, y_pred):
    """Root mean squared error regression loss"""
    return K.sqrt(K.mean(K.square(y_true-y_pred)))
```

In [34]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.models import Model
from keras.layers import Input
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras import regularizers
from keras.regularizers import l2
from keras.layers import Flatten
from keras.layers import Dense, Input , Dropout
from keras.layers import concatenate
from keras.layers.normalization import BatchNormalization
```

In [ ]:
```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(train_df[train_cols].shape
[1],)))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(16, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(8, activation='relu'))
model.add(Dense(1))
model.compile(loss=root_mean_squared_error, optimizer="adam")
```

In [66]:
```
model = Sequential()
model.add(Dense(2 ** 10, input_dim = 187, kernel_initializer='random_uniform',
activation='relu'))
model.add(Dropout(0.25))
model.add(BatchNormalization())
model.add(Dense(2 ** 9, kernel_initializer='random_uniform', activation='relu'
))
model.add(BatchNormalization())
model.add(Dropout(0.1))
model.add(Dense(2 ** 5, kernel_initializer='random_uniform', activation='relu'
))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss=keras_root_mean_squared_error, optimizer='adam')
```

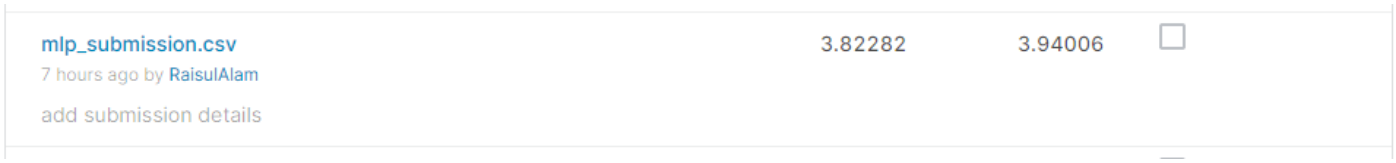
In [67]:
```
history = model.fit(X_Train, Y_Train, batch_size=32, epochs=10, verbose=1, val
idation_data=(X_Test, Y_Test))
```

```
Epoch 1/10
4417/4417 [==============================] - 19s 4ms/step - loss: 3.1019 - va
l_loss: 3.2950
Epoch 2/10
4417/4417 [==============================] - 18s 4ms/step - loss: 3.1114 - va
l_loss: 3.1176
Epoch 3/10
4417/4417 [==============================] - 20s 4ms/step - loss: 3.0987 - va
l_loss: 3.1225
Epoch 4/10
4417/4417 [==============================] - 18s 4ms/step - loss: 3.1088 - va
l_loss: 3.1173
Epoch 5/10
4417/4417 [==============================] - 19s 4ms/step - loss: 3.1089 - va
l_loss: 3.1174
Epoch 6/10
4417/4417 [==============================] - 19s 4ms/step - loss: 3.1051 - va
l_loss: 3.1174
Epoch 7/10
4417/4417 [==============================] - 18s 4ms/step - loss: 3.1000 - va
l_loss: 3.1173
Epoch 8/10
4417/4417 [==============================] - 19s 4ms/step - loss: 3.1113 - va
l_loss: 3.1173
Epoch 9/10
4417/4417 [==============================] - 19s 4ms/step - loss: 3.1059 - va
l_loss: 3.1173
Epoch 10/10
4417/4417 [==============================] - 19s 4ms/step - loss: 3.1011 - va
l_loss: 3.1173
```

In [68]:
```
prediction=model.predict(test[feature])
```

In [69]:
```
!rm /kaggle/working/*.csv
```

In [70]:
```
sample_submission = pd.read_csv('../input/elo-merchant-category-recommendation/sample_submission.csv')
sample_submission['target'] = prediction
sample_submission.to_csv('mlp_submission.csv', index=False)
```



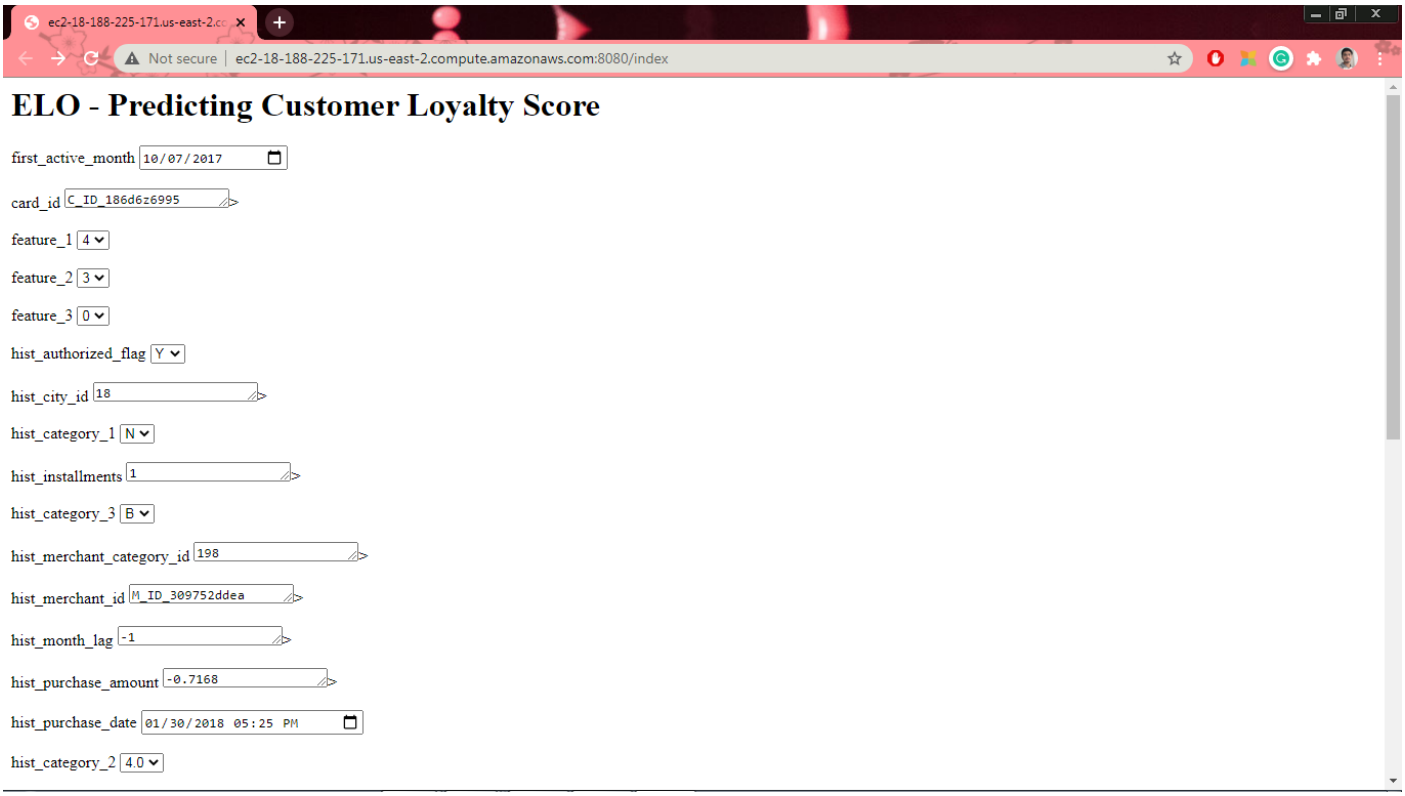
In [1]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model_Description", "Public Score", "Private Score"]
x.add_row(["LGBM Regressor with kfold", 3.69118, 3.61674])
x.add_row(["XGBOOST Regressor with k fold", 3.75041, 3.64933])
x.add_row(["MLP Architecture",3.94006,3.82282 ])
print(x)
```

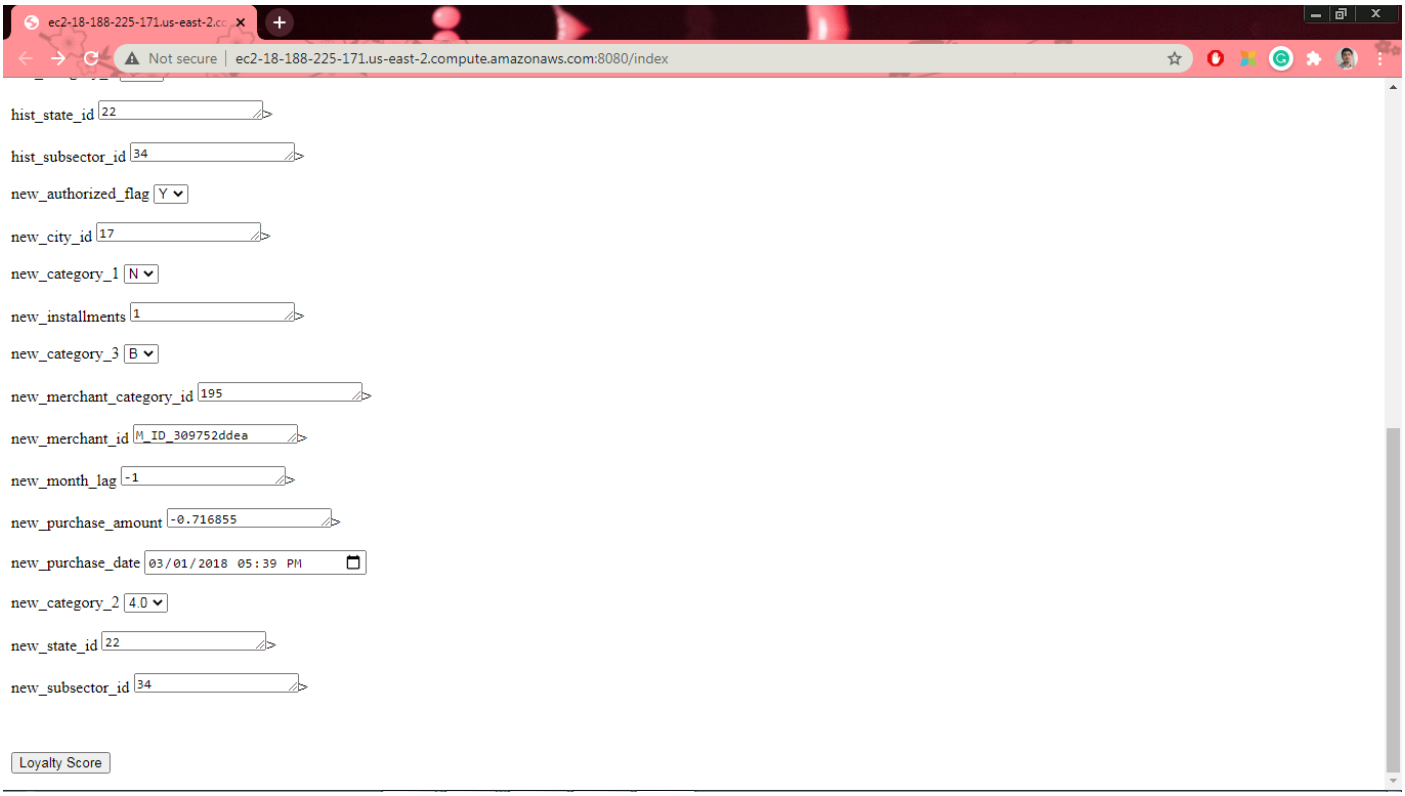
```
+------------------------------+--------------+---------------+
|       Model_Description      | Public Score | Private Score |
+------------------------------+--------------+---------------+
|   LGBM Regressor with kfold  |   3.69118    |    3.61674    |
| XGBOOST Regressor with k fold |   3.75041    |    3.64933    |
|        MLP Architecture      |   3.94006    |    3.82282    |
+------------------------------+--------------+---------------+
```

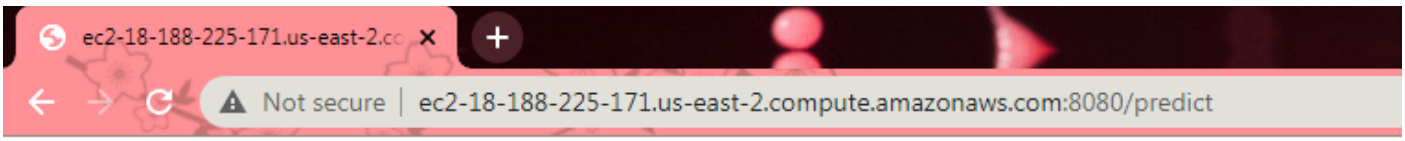
Deployment

Currently the site is live at http://ec2-18-188-225-171.us-east-2.compute.amazonaws.com:8080/index (http://ec2-18-188-225-171.us-east-2.compute.amazonaws.com:8080/index) . This is done using Flask framework and it is deployed in AWS .

There are few limitations exist in frontend perspective which can be improved further ie, i) Entering the NULL/inproper value will hamper the loyalty score also Run time exceptions may occur which can be prevented further to make it real time compliance. ii) UI can be improved significantly and make it responsive.


ec2-18-188-225-171.us-east-2.c
Not secure | ec2-18-188-225-171.us-east-2.compute.amazonaws.com:8080/index
ELO - Predicting Customer Loyalty Score
first_active_month 10/07/2017
card_id C_ID_186d6z6995
feature_1 4
feature_2 3
feature_3 0
hist_authorized_flag Y
hist_city_id 18
hist_category_1 N
hist_installments 1
hist_category_3 B
hist_merchant_category_id 198
hist_merchant_id M_ID_309752ddea
hist_month_lag -1
hist_purchase_amount -0.7168
hist_purchase_date 01/30/2018 05:25 PM
hist_category_2 4.0






```
{
  "Loyalty Score": 0.0076363761497151034
}
```


In [ ]: