Modeling customer churn for transactions between 04-01-19 and 10-31-19

- Use the Lifetimes Python package to predict non-contractual churn risk or CLV
  - The lifetimes package was implemented by Cameron Davidson Pilon at Shopify
- It's more cost effective to retain existing customers than to acquire new ones, which is why it's important to track customers at high risk of turnover (churn) and target them with retention strategies.
- Build a customer churn model based off of data from dw_bi_vw.F_POS_TXN_DTL
  - Identify high risk customers and inform retention strategies and marketing experiments.
- While it's straightforward to measure churn for a contractual (subscription-based) business.
- Churns aren't explicitly observed in non-contractual businesses.
  - The probabilistic BG/NBD(Beta Geometric Negative Binomial Distribution) model from the Lifetimes package will be used for for estimating time of customer death/churn.

working history

- module name: C:\Users\syue003\CLV\churn_lifetimes_apr19_oct19.ipynb
- Author: Sophia Yue
- Date: Nov, 2019 ## Ref
- Modeling customer churn for an e-commerce company with Python by https://github.com/collindching/Olist-Customer-Churn/blob/master/Olist%20churn%20risk%20model.ipynb (https://github.com/collindching/Olist-Customer-Churn/blob/master/Olist%20churn%20risk%20model.ipynb) https://towardsdatascience.com/modeling-customer-churn-for-an-e-commerce-business-with-python-874315e688bf (https://towardsdatascience.com/modeling-customer-churn-for-an-e-commerce-business-with-python-874315e688bf)
- https://lifetimes.readthedocs.io/en/master/Quickstart.html (https://lifetimes.readthedocs.io/en/master/Quickstart.html)
  - lifetimes document https://readthedocs.org/projects/lifetimes/downloads/pdf/latest/ (https://readthedocs.org/projects/lifetimes/downloads/pdf/latest/)

input files

- Summary table: temp_tables.chn_txn_0419_1019_sumy with total count = 30,547,466
  - Count of summary table "temp_tables.chn_txn_0419_1019_sumy with frequency > 0 " = 25211251
    - Only includes customers with repeat purchases
  - Summarize the transactional data '2019-04-01' and '2019-10-30' from dw_bi_vw.F_POS_TXN_DTL
    - group by hh_sk to create the columns hh_sk, freqnency, recency, monetary_value, T (Age)
  - SQL to create the summary table

    create table temp_tables.chn_txn_0419_1019_sumy as (SELECT hh_sk, COUNT(distinct txn_dt) - 1 as frequency, MAX(txn_dt) - MIN(txn_dt) as recency, AVG(EXT_PRC_AMT) as monetary_value, cast('2019-10-30' as date) - MIN(txn_dt) as T FROM dw_bi_vw.F_POS_TXN_DTL where txn_dt between '2019-04-01' and '2019-10-30' and prod_sk > 0 and hh_sk > 0 GROUP BY hh_sk) with data
- Divided summary table into temp_tables.chn_txn_0419_1019_sumy_1, ~ temp_tables.chn_txn_0419_1019_sumy_4
  - It took forever to read the complete summary table from Python
  - Only include frequency > 0
  - SQL to create temp_tables.chn_txn_0419_1019_sumy_1 (count = 6,302,813)
    - create table temp_tables.chn_txn_0419_1019_sumy_1 as sel * from temp_tables.chn_txn_0419_1019_sumy where frequency > 0 and hh_sk < 14881620) with data
  - SQL to create temp_tables.chn_txn_0419_1019_sumy_2 (count = 6,302,812
    - create table temp_tables.chn_txn_0419_1019_sumy_2 as (sel * from temp_tables.chn_txn_0419_1019_sumy where frequency > 0 and hh_sk between 14881620 and 30675905) with data
  - SQL to create temp_tables.chn_txn_0419_1019_sumy_3 (count = 6,302,813)
    - create table temp_tables.chn_txn_0419_1019_sumy_3 as (sel * from temp_tables.chn_txn_0419_1019_sumy where frequency > 0 and hh_sk between 30675906 and 64065711) with data
  - SQL to create temp_tables.chn_txn_0419_1019_sumy_4 (count = 6,302,813)
    - create table temp_tables.chn_txn_0419_1019_sumy_4 as (sel * from temp_tables.chn_txn_0419_1019_sumy where frequency > 0 and hh_sk > 64065711) with data

Load the data

Notes

- Install 'lifetimes' package
  - Use the following command to install the Python package 'lifetimes'
    - pip install lifetimes

- Assumption of BG/NBD model
  - A customer's relationship has two phases: "alive" for an unobserved period of time, then "dead"
  - While alive, the number of transactions made by a customer follows a Poisson distribution with transaction rate lambda
    - Poisson distribution
      - The population is large and the probability is small
  - Heterogeneity in lambda follows a gamma distribution
  - After any transaction, a customer dies with probability p
    - The probability that a customer dies after a number of transactions follows a geometric distribution
  - p follows a beta distribution
  - Lambda and p vary independently across customers
- RFM (Recency-Frequency-Monetary)
  - The Lifetimes package relies on (RFM) analysis to model churn and customer lifetime value (CLV).
    - To make our models, we'll need a a dataframe that consists of recency, frequency, and monetary columns.
      - Recency: time between initial purchase and most recent (last) purchase
      - Frequency: number of repeat purchases made by a customer (total purchases - 1)
      - Monetary: total spent on purchases

Main process starts here

Load function

In [2]:
```
def load_dataset(filename, **kwargs):
    '''
    Load a dataset from lifetimes.datasets

    Parameters:
    filename : for example "larynx.csv"
    usecols : list of columns in file to use

    Returns : Pandas dataframe
    '''
    return pd.read_csv(resource_filename('lifetimes', 'datasets/' + filename), **kwargs)
```

Initialization

- Python compile() function is used to compile the source into code object or AST module object.
- The returned code object can be executed using exec()
- Use compile function to execute the following codes which might be used by other modules
  - c_import.py : Import packages/libraries
  - c_setup_dbs_con.py : Set up Teradata connection
  - c_time_dte.py

In [4]:
```
prg_name = ""
path_code = "C:\\Users\\syue003\\wip_RecSys\\"
c_import  = path_code + "c_import.py"
c_setup_dbs_con = path_code + "c_setup_dbs_con.py"
c_timedte = path_code + "c_time_dte.py"

exec(compile(open(c_import, 'rb').read(), c_import,  'exec'))
exec(compile(open(c_setup_dbs_con, 'rb').read(),c_setup_dbs_con, 'exec'))
exec(compile(open(c_timedte, 'rb').read(),c_timedte, 'exec'))
session, td_enginex = cf_setup_dbs_con(userName = 'syue003', passWord = 'newpassword')
```

```
t_engine teradata://syue003:newpassword@tqdpr02/temp_tables
```

In [5]:
```
from lifetimes.utils import *
from lifetimes import BetaGeoFitter,GammaGammaFitter
from lifetimes.plotting import plot_probability_alive_matrix, plot_frequency_recency_matrix, plot_pe
riod_transactions
from lifetimes.plotting import plot_cumulative_transactions,plot_incremental_transactions
from lifetimes.generate_data import beta_geometric_nbd_model
from lifetimes.plotting import plot_calibration_purchases_vs_holdout_purchases, plot_period_transact
ions
from lifetimes.plotting import plot_history_alive

from lifetimes.datasets import load_transaction_data
from lifetimes.utils import summary_data_from_transaction_data
from pkg_resources import resource_filename
```

Load lifetimes dataset

- The csv file is a summary file to aggragate from transaction table with the structure of hh_sk, freqnency, recency, monetary_value, T
  - One hh_sk would have one row only
- Invoke the function 'load_dataset' to convert the csv file to lifetimes dataset
  - Use hh_sk as an index
    - index_col = [0]

In [8]:
```
summary = load_dataset('C:\SYUE\RecSys\data\chn_txn_0419_1019_sumy_comb.csv', index_col = [0])
```

In [10]:
```
summary.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 5 columns):
hh_sk             1048575 non-null int64
frequency         1048575 non-null int64
recency           1048575 non-null int64
monetary_value    1048575 non-null float64
T                 1048575 non-null int64
dtypes: float64(1), int64(4)
memory usage: 40.0 MB
```

In [7]:
```
start_time = time.time()
summary = load_dataset('C:\SYUE\RecSys\data\chn_txn_0419_1019_sumy_comb.csv', index_col = [0])
summary.describe()
end_time = time.time()
cf_elapse_time (  start_time, end_time, "Modul build completed.")
```

```
 Modul build completed. It took 62.809000 seconds - 0hh:1mm:2ss.
 start time: Dec 05 2019 11:40:50  end time:  Dec 05 2019 11:41:53
```

Part 1: Predict Churn - case1

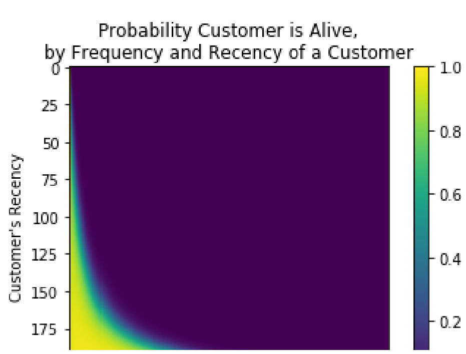
- Case1
  - Not remove 'monetary_value' < = 0 from the dataset summary
  - penalizer_coef=0.0 for modeling
    - Penalty the likelihood
      - The coefficient applied to an l2 norm on the parameters ????
  - t = 1 in expected_number_of_purchases_up_to_time
    - one unit of time
    - times to calculate the expectation for

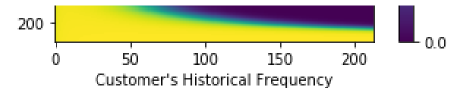
In [9]:
```
start_time = time.time()

bgf = BetaGeoFitter(penalizer_coef=0.0)
bgf.fit(summary['frequency'], summary['recency'], summary['T'])
end_time = time.time()
cf_elapse_time (  start_time, end_time, "Modul build completed.")
print(bgf)
```

```
 Modul build completed. It took 701.523000 seconds - 0hh:11mm:41ss.
 start time: Dec 05 2019 11:46:58  end time:  Dec 05 2019 11:58:39
<lifetimes.BetaGeoFitter: fitted with 25211251 subjects, a: 0.06, alpha: 9.30, b: 1.44, r: 1.12>
```

In [10]:
```
from lifetimes.plotting import plot_probability_alive_matrix
plot_probability_alive_matrix(bgf)
```

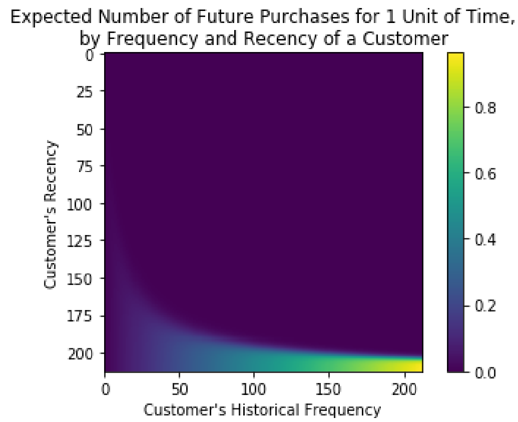
Out[10]: `<matplotlib.axes._subplots.AxesSubplot at 0xec1d9b0>`





In [11]:
```
from lifetimes.plotting import plot_frequency_recency_matrix

plot_frequency_recency_matrix(bgf)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0xf77d898>

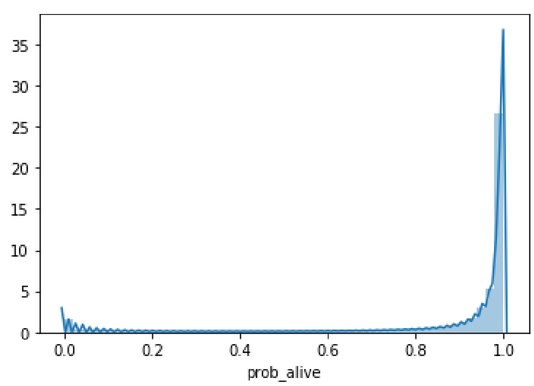


In [12]:
```
t = 1
summary['predicted_purchases'] = bgf.conditional_expected_number_of_purchases_up_to_time(t, summary[
'frequency'], summary['recency'], summary['T'])
summary.sort_values(by='predicted_purchases').tail(5)
```

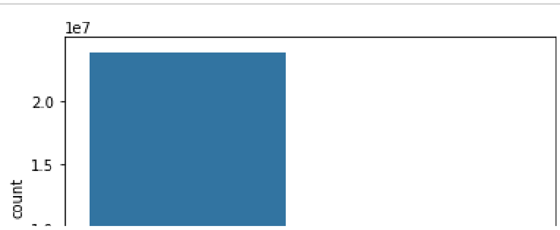
Out[12]:

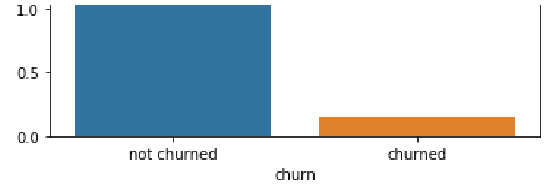
	frequency	recency	monetary_value	T	predicted_purchases
HH_SK					
19025325.0	212.0	212.0	3.493262	212.0	0.962623
29906616.0	212.0	212.0	3.200693	212.0	0.962623
19532370.0	212.0	212.0	4.350386	212.0	0.962623
6204593.0	212.0	212.0	3.393995	212.0	0.962623
49184280.0	212.0	212.0	3.188452	212.0	0.962623

In [13]:
```
summary['prob_alive'] = bgf.conditional_probability_alive(summary['frequency'],summary['recency'],su
mmary['T'])
sns.distplot(summary['prob_alive']);
```



In [15]:
```
summary['churn'] = ['churned' if p < .1 else 'not churned' for p in summary['prob_alive']]
sns.countplot(summary['churn']);
```





In [16]:
```
summary['churn'][(summary['prob_alive']>=.1) & (summary['prob_alive']<.2)] = "high risk"
summary['churn'].value_counts()
```

Out[16]:
```
not churned    23308618
churned         1416882
high risk        485751
Name: churn, dtype: int64
```

In [17]:
```
summary.head()
```

Out[17]:

	frequency	recency	monetary_value	T	predicted_purchases	prob_alive	churn
HH_SK							
1741572.0	10.0	185.0	2.680000	204.0	5.129661e-02	9.840199e-01	not churned
2912171.0	22.0	144.0	3.861483	209.0	1.010236e-02	9.539740e-02	churned
11106364.0	92.0	210.0	3.276888	210.0	4.242879e-01	9.993506e-01	not churned
3609983.0	3.0	4.0	2.619706	176.0	2.455762e-05	1.104431e-03	churned
11061359.0	129.0	159.0	5.138111	212.0	4.285068e-13	7.288822e-13	churned

In [18]:
```
summary.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Float64Index: 25211251 entries, 1741572.0 to 68066956.0
Data columns (total 7 columns):
frequency              float64
recency                float64
monetary_value         float64
T                      float64
predicted_purchases    float64
prob_alive             float64
churn                  object
dtypes: float64(6), object(1)
memory usage: 2.1+ GB
```

In [20]:
```
#Correlation
df_churn = pd.get_dummies(summary)
corr_matrix = df_churn.corr()
corr_matrix['prob_alive
            '].sort_values(ascending=False)
```

Out[20]:
```
prob_alive             1.000000
churn_not churned      0.832049
recency                0.660016
predicted_purchases    0.381457
frequency              0.248130
T                      0.088744
monetary_value        -0.007901
churn_high risk       -0.362479
churn_churned         -0.737941
Name: prob_alive, dtype: float64
```

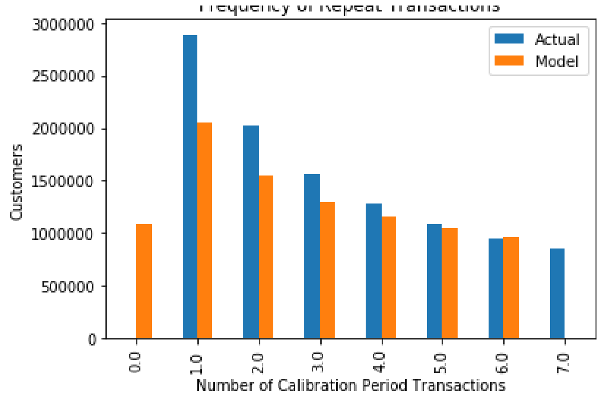
In [21]:
```
summary.to_excel('C:\SYUE\RecSys\data\chn_txn_0419_1019_churn.xlsx', index = True)
```

Assess model fit

In [22]:
```
start_time = time.time()
plot_period_transactions(bgf)

end_time = time.time()
cf_elapse_time (  start_time, end_time, "Validation")
```

```
 Validation It took 17876.992329 seconds - 4hh:57mm:56ss.
 start time: Dec 05 2019 13:04:55  end time:  Dec 05 2019 18:02:52
```

Frequency of Repeat Transactions



Part 1: Predict Churn - case2

- Case2
  - Remove 'monetary_value' < = 0 from the dataset summary
  - penalizer_coef=0.05 for modeling
    - Penalty the likelihood
      - The coefficient applied to an l2 norm on the parameters ????
  - t = 1 in expected_number_of_purchases_up_to_time
    - one unit of time
    - times to calculate the expectation for

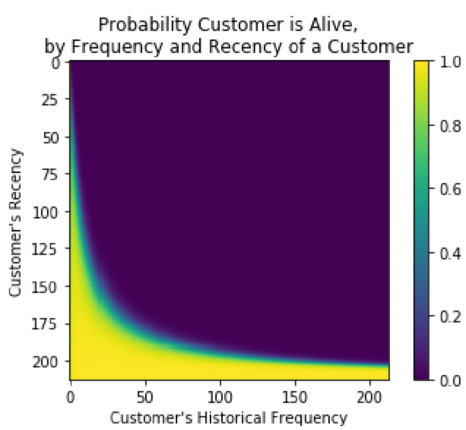
In [23]:
```
sumy_mnt = summary [  summary['monetary_value'] > 0 ]
```

In [25]:
```
start_time = time.time()

bgf_c = BetaGeoFitter(penalizer_coef=0.05)
bgf_c.fit(sumy_mnt['frequency'], sumy_mnt ['recency'], sumy_mnt['T'])
end_time = time.time()
cf_elapse_time (  start_time, end_time, "Modul build bgf_c.BetaGeoFitter(penalizer_coef=0.05) comple
ted.")
print(bgf_c)
```

```
 Modul build bgf_c.BetaGeoFitter(penalizer_coef=0.05) completed. It took 731.661500 seconds - 0hh:12
mm:11ss.
 start time: Dec 06 2019 11:53:57  end time:  Dec 06 2019 12:06:09
<lifetimes.BetaGeoFitter: fitted with 25172740 subjects, a: 0.03, alpha: 7.76, b: 0.43, r: 0.94>
```

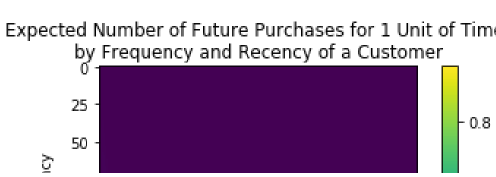
In [26]:
```
plot_probability_alive_matrix(bgf_c)
```

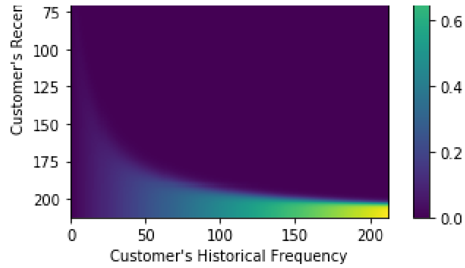
Out[26]: `<matplotlib.axes._subplots.AxesSubplot at 0x105ed630>`



In [27]:
```
plot_frequency_recency_matrix(bgf_c)
```

Out[27]: `<matplotlib.axes._subplots.AxesSubplot at 0x102d6978>`



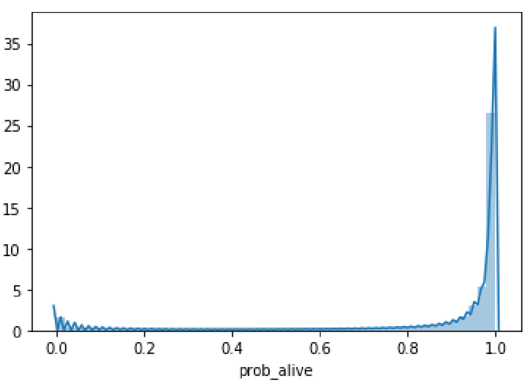


In [28]:
```
t = 1
sumy_mnt['predicted_purchases'] = bgf_c.conditional_expected_number_of_purchases_up_to_time(t,sumy_m
nt['frequency'], sumy_mnt['recency'],sumy_mnt['T'])
sumy_mnt.sort_values(by='predicted_purchases').tail(5)
```

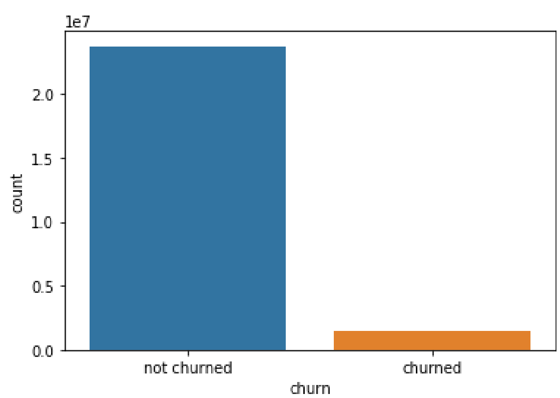
Out[28]:

	frequency	recency	monetary_value	T	predicted_purchases	prob_alive	churn
HH_SK							
20779400.0	212.0	212.0	6.319027	212.0	0.968744	0.999717	not churned
1753527.0	212.0	212.0	3.667429	212.0	0.968744	0.999717	not churned
75234978.0	212.0	212.0	1.489168	212.0	0.968744	0.999717	not churned
19586023.0	212.0	212.0	5.178110	212.0	0.968744	0.999717	not churned
7251912.0	212.0	212.0	2.297757	212.0	0.968744	0.999717	not churned

In [29]:
```
sumy_mnt['prob_alive'] = bgf.conditional_probability_alive(sumy_mnt['frequency'],sumy_mnt['recency'
],sumy_mnt['T'])
sns.distplot(sumy_mnt['prob_alive']);
```



In [30]:
```
sumy_mnt['churn'] = ['churned' if p < .1 else 'not churned' for p in sumy_mnt['prob_alive']]
sns.countplot(sumy_mnt['churn']);
```



In [31]:
```
sumy_mnt['churn'][(summary['prob_alive']>=.1) & (sumy_mnt['prob_alive']<.2)] = "high risk"
sumy_mnt['churn'].value_counts()
```

Out[31]:
```
not churned    23273971
churned         1414382
high risk        484387
Name: churn, dtype: int64
```

Check the correclation

In [32]:
```
sumy_mnt[['monetary_value', 'frequency']].corr()
```

Out[32]:

	monetary_value	frequency
monetary_value	1.000000	-0.047503
frequency	-0.047503	1.000000

part2 - Predict customer_lifetime_value

- customer_lifetime_value(transaction_prediction*model, frequency, recency, T, monetary* value, time=12, discount_rate=0.01, freq='D')
  - Computes the average lifetime value for a group of one or more customers
    - Return customer lifetime value
- Parameters
  - time: The lifetime expected for the user in months. Default: 12
  - discount_rate: The monthly adjusted discount rate. Default:0.01
  - freq: Unit of time to measure in. {"D", "H", "M", "W"} for day, hour, month, week. Default = "D"

In [34]:
```
ggf =GammaGammaFitter(penalizer_coef=0.05)
ggf.fit(sumy_mnt['frequency'], sumy_mnt ['monetary_value'])
```

Out[34]: <lifetimes.GammaGammaFitter: fitted with 25172740 subjects, p: 1.81, q: 1.05, v: 1.69>

In [35]:
```
bgf.fit(sumy_mnt['frequency'], sumy_mnt['recency'],
        sumy_mnt['T'])
ser_ggf = ggf.customer_lifetime_value(
bgf, #the model to use to predict the number of future transactions
sumy_mnt['frequency'],
sumy_mnt['recency'],
sumy_mnt['T'],
sumy_mnt['monetary_value'],
time=12,
discount_rate = 0.01)
```

In [36]:
```
def cf_order_cluster(cluster_field_name, target_field_name,df,ascending):
    """
     Module name: cf_order_cluster
     Purpose    : Function to base on the mean of cluster to order the cluster
     Parameters:
        cluster_field_name: Field name to be clustered
         e.g.  RecencyCluster
        target_field_name : Variable name of cluster
         e.g.  Recency
        df: dataframe to be clustered
        ascending: The sequence to sort the mean to order the cluster
          True: Ascending
          False : Decending
      Return
        A dataframe with ordered cluster
    """
    new_cluster_field_name = 'new_' + cluster_field_name
    df_new = df.groupby(cluster_field_name)[target_field_name].mean().reset_index()
    df_new = df_new.sort_values(by=target_field_name,ascending=ascending).reset_index(drop=True)
    df_new['index'] = df_new.index
    df_final = pd.merge(df,df_new[[cluster_field_name,'index']], on=cluster_field_name)
    df_final = df_final.drop([cluster_field_name],axis=1)
    df_final = df_final.rename(columns={"index":cluster_field_name})
    return df_final
```

In [37]:
```
df_ggf_y =  ser_ggf.to_frame()
df_ggf_y.reset_index( level = 0, inplace = True)  # index will become a column
df_ggf_y.head()
df_ggf_y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25172740 entries, 0 to 25172739
Data columns (total 2 columns):
HH_SK    float64
clv      float64
dtypes: float64(2)
```

```
dtypes: float64(2)
memory usage: 384.1 MB
```

In [38]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(df_ggf_y[['clv']])
df_ggf_y['CLVCluster'] = kmeans.predict(df_ggf_y[['clv']])
df_ggf_z = cf_order_cluster('CLVCluster', 'clv',df_ggf_y,True)
```

In [39]:
```
df_ggf_z.CLVCluster.value_counts()
```

Out[39]:
```
0    18550330
1     5433295
2     1189115
Name: CLVCluster, dtype: int64
```

In [41]:
```
18550330/25172740     # CLVCluster = 0
```

Out[41]:
```
0.7369213681148735
```

In [42]:
```
5433295/25172740        # CLVCluster = 1
```

Out[42]:
```
0.21584042897197525
```