Predicting Next Purchase Date

Machine Learning model to predict when the customers will make their next purchase

In this article, we will be using 18 month transactional dataset of 600000 customers with over 3 crore transactions and follow the steps below:

- Data Wrangling (creating previous/next datasets and calculate purchase day differences)
- Feature Engineering
- Selecting a Machine Learning Model
- Multi-Classification Model
- Hyperparameter Tuning

Data Wrangling

Let's start with importing libraries and reading our dataset

In [1]:

```
#import libraries
from datetime import datetime, timedelta,date
import pandas as pd
%matplotlib inline
from sklearn.metrics import classification_report,confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from __future__ import division
from sklearn.cluster import KMeans
```

In [2]:

```
#do not show warnings
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
#import plotly for visualization
import chart_studio.plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go
```

Height: 768

In [4]:

```
#import machine learning related libraries
from sklearn.svm import SVC
from sklearn.multioutput import MultiOutputClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split
```

In [5]:

```
#initiate plotly
pyoff.init_notebook_mode()
```

In [6]:

```
#import the csv
tx_data = pd.read_csv('18monthtxns.csv')
```

In [7]:

```
tx_data.shape
```

Out[7]:

```
(10389324, 4)
```

In [8]:

```
#print first 10 rows
tx_data.head(2)
```

Out[8]:

	bank	id	transdate	transamount
0	SBI	253987	2018-11-04 00:00:00	499.0
1	SBI	127456	2018-11-05 00:00:00	13700.0
2	SBI	513643	2018-11-21 00:00:00	460.0
3	SBI	3773	2018-11-25 00:00:00	158.0
4	SBI	346051	2018-11-21 00:00:00	3040.0
5	SBI	59058	2018-11-11 00:00:00	1000.0
6	SBI	78780	2018-11-13 00:00:00	200.0
7	SBI	592574	2018-11-03 00:00:00	170.0
8	SBI	50710	2018-11-06 00:00:00	310.0
9	SBI	229798	2018-11-15 00:00:00	150.0

Height: 768

In [9]:

```
#convert date field from string to datetime
tx_data['transdate'] = pd.to_datetime(tx_data['transdate'])
```

In [10]:

```
tx_data['transdate'].describe()
```

Out[10]:

```
count                10389324
unique                    546
top       2018-11-05 00:00:00
freq                    27724
first     2018-01-01 00:00:00
last      2019-06-30 00:00:00
Name: transdate, dtype: object
```

To build our model, we should split our data into two parts:

- We use fifteen months of behavioral data to predict customers' first purchase date in the next three months. If there is no purchase, we will predict that too. Let's assume our cut off date is Mar 30th '19 and split the data.

In [11]:

```
tx_m = tx_data[(tx_data.transdate < '2019-03-30') & (tx_data.transdate >= '2018-01-01')].re
tx_next = tx_data[(tx_data.transdate >= '2019-03-30') & (tx_data.transdate <= '2019-06-30')
```

tx_m represents the fifteen months performance whereas we will use tx_next to find out the days between the last purchase date in tx_m and the first one in tx_next.

In [12]:

```
tx_next['transdate'].describe()
```

Out[12]:

```
count                 2152864
unique                     93
top       2019-05-04 00:00:00
freq                    26913
first     2019-03-30 00:00:00
last      2019-06-30 00:00:00
Name: transdate, dtype: object
```

Also, we will create a dataframe called tx_user to possess a user-level feature set for the prediction model:

In [13]:

```
tx_user = pd.DataFrame(tx_m['id'].unique())
tx_user.columns = ['CustomerID']
```

Adding Label

Height: 768

By using the data in tx_next, we need the calculate our label (days between last purchase before cut off date and first purchase after that):

In [14]:

```
#create a dataframe with customer id and first purchase date in tx_next
tx_next_first_purchase = tx_next.groupby('id').transdate.min().reset_index()
```

In [15]:

```
tx_next_first_purchase.columns = ['CustomerID','MinPurchaseDate']
```

In [16]:

```
tx_next_first_purchase.head()
```

Out[16]:

	CustomerID	MinPurchaseDate
0	1	2019-04-10
1	3	2019-04-02
2	7	2019-04-30
3	8	2019-04-09
4	10	2019-04-23

In [17]:

```
#create a dataframe with customer id and last purchase date in tx_m
tx_last_purchase = tx_m.groupby('id').transdate.max().reset_index()
```

In [18]:

```
tx_last_purchase.columns = ['CustomerID','MaxPurchaseDate']
```

In [19]:

```
#merge two dataframes
tx_purchase_dates = pd.merge(tx_last_purchase,tx_next_first_purchase,on='CustomerID',how='l
```

In [20]:

```
#calculate the time difference in days
tx_purchase_dates['NextPurchaseDay'] = (tx_purchase_dates['MinPurchaseDate'] - tx_purchase_
```

Height: 768

In [21]:

```
tx_purchase_dates.head()
```

Out[21]:

	CustomerID	MaxPurchaseDate	MinPurchaseDate	NextPurchaseDay
0	1	2019-03-23	2019-04-10	18.0
1	2	2018-08-03	NaT	NaN
2	3	2019-03-26	2019-04-02	7.0
3	4	2018-11-18	NaT	NaN
4	5	2018-07-21	NaT	NaN

In [22]:

```
#merge with tx_user
tx_user = pd.merge(tx_user, tx_purchase_dates[['CustomerID','NextPurchaseDay']],on='Custome
```

In [23]:

```
tx_user.head()
```

Out[23]:

	CustomerID	NextPurchaseDay
0	253987	152.0
1	127456	NaN
2	513643	2.0
3	3773	5.0
4	346051	NaN

In [24]:

```
tx_user.shape
```

Out[24]:

```
(600000, 2)
```

As you can easily notice, we have NaN values because those customers haven't made any purchase yet. We fill NaN with 999 to quickly identify them later.

In [25]:

```
#fill NA values with 999
tx_user = tx_user.fillna(999)
```

Feature Engineering

For this project, I have selected the feature candidates like below:

- RFM scores & clusters
- Days between the last three purchases

Height: 768

- Mean & standard deviation of the difference between purchases in days

Recency

In [26]:

```
#get max purchase date for Recency and create a dataframe
tx_max_purchase = tx_m.groupby('id').transdate.max().reset_index()
```

In [27]:

```
tx_max_purchase.columns = ['CustomerID','MaxPurchaseDate']
```

In [28]:

```
#find the recency in days and add it to tx_user
tx_max_purchase['Recency'] = (tx_max_purchase['MaxPurchaseDate'].max() - tx_max_purchase['M
```

In [29]:

```
tx_user = pd.merge(tx_user, tx_max_purchase[['CustomerID','Recency']], on='CustomerID')
```

In [30]:

```
tx_user.head()
```

Out[30]:

	CustomerID	NextPurchaseDay	Recency
0	253987	152.0	145
1	127456	999.0	144
2	513643	2.0	1
3	3773	5.0	3
4	346051	999.0	85

In [31]:

```
tx_user.Recency.describe()
```

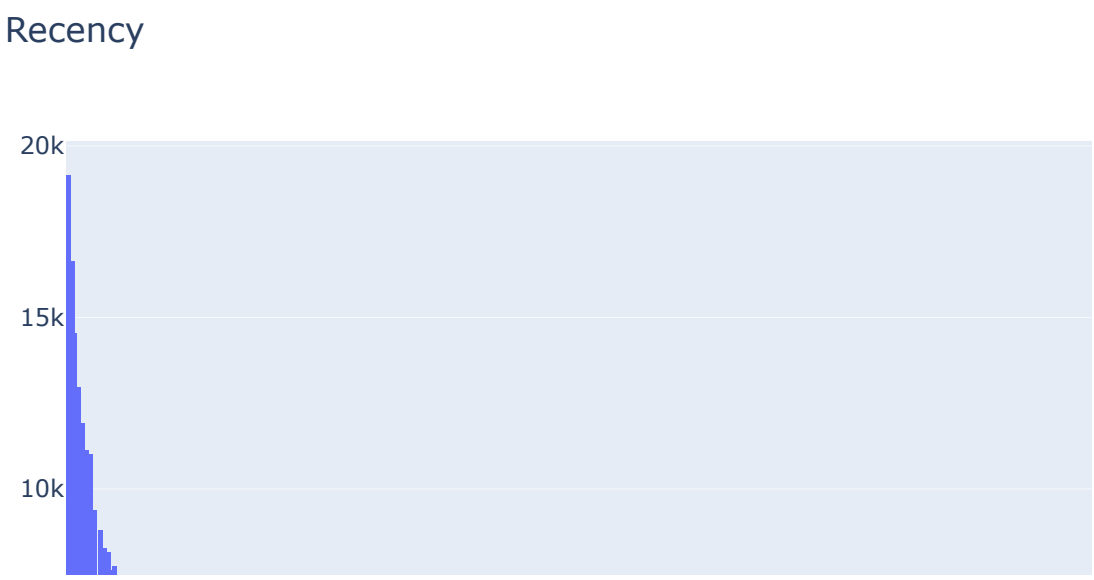
Out[31]:

```
count    600000.000000
mean         92.140965
std         102.055801
min           0.000000
25%          13.000000
50%          51.000000
75%         142.000000
max         452.000000
Name: Recency, dtype: float64
```

Height: 768

In [32]:

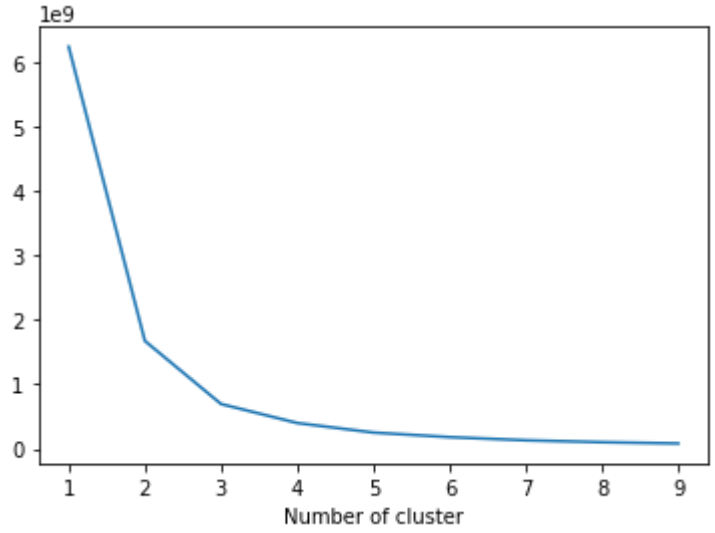
```
#plot recency
plot_data = [
    go.Histogram(
        x=tx_user['Recency']
    )
]

plot_layout = go.Layout(
        title='Recency'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```



Height: 768

In [33]:

```
sse={}
tx_recency = tx_user[['Recency']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_recency)
    tx_recency["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



In [34]:

```
#clustering for Recency
kmeans = KMeans(n_clusters=4)
kmeans.fit(tx_user[['Recency']])
tx_user['RecencyCluster'] = kmeans.predict(tx_user[['Recency']])
```

Height: 768

In [35]:

```
#order cluster method
def order_cluster(cluster_field_name, target_field_name,df,ascending):
    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=
    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 [36]:

```
#order recency clusters
tx_user = order_cluster('RecencyCluster', 'Recency',tx_user,False)
```

In [37]:

```
#print cluster characteristics
tx_user.groupby('RecencyCluster')['Recency'].describe()
```

Out[37]:

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	48282.0	342.131125	45.548672	276.0	303.0	336.0	376.0	452.0
1	82420.0	207.556807	34.035510	158.0	177.0	203.0	236.0	275.0
2	141486.0	106.666525	26.985892	64.0	83.0	104.0	130.0	157.0
3	327812.0	20.033287	17.547502	0.0	5.0	15.0	31.0	63.0

Frequency

In [38]:

```
#get total purchases for frequency scores
tx_frequency = tx_m.groupby('id').transdate.count().reset_index()
```

In [39]:

```
tx_frequency.columns = ['CustomerID','Frequency']
```

Height: 768

In [40]:

```
tx_frequency.head()
```

Out[40]:

	CustomerID	Frequency
0	1	33
1	2	7
2	3	4
3	4	4
4	5	3

In [41]:

```
#add frequency column to tx_user
tx_user = pd.merge(tx_user, tx_frequency, on='CustomerID')
```

In [42]:

```
tx_user.head()
```

Out[42]:

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency
0	253987	152.0	145	2	6
1	127456	999.0	144	2	6
2	346051	999.0	85	2	7
3	174775	221.0	130	2	6
4	518115	999.0	145	2	4

In [43]:

```
tx_user.Frequency.describe()
```

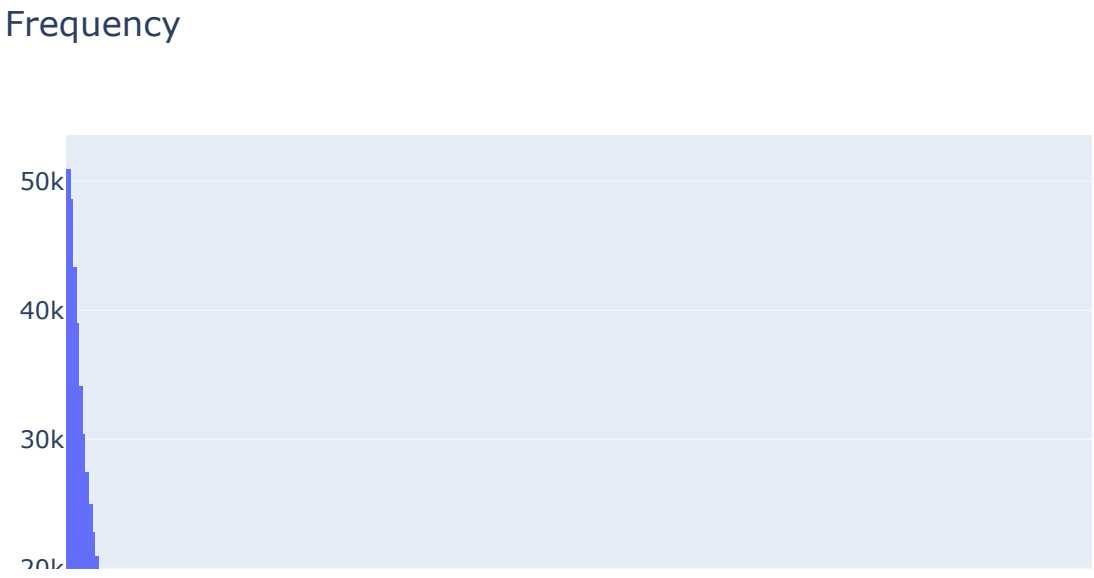
Out[43]:

```
count    600000.000000
mean         13.727433
std          10.420694
min           3.000000
25%           6.000000
50%          11.000000
75%          19.000000
max         524.000000
Name: Frequency, dtype: float64
```

Height: 768

In [44]:

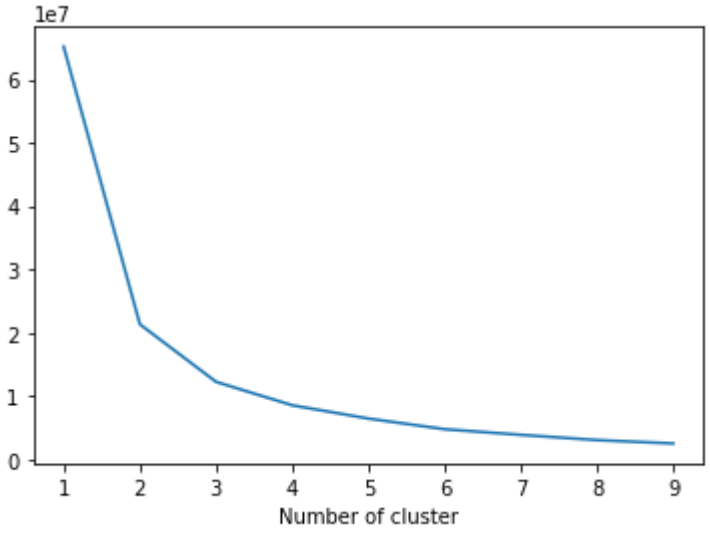
```
#plot frequency
plot_data = [
    go.Histogram(
        x=tx_user.query('Frequency < 1000')['Frequency']
    )
]

plot_layout = go.Layout(
        title='Frequency'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```



Height: 768

In [45]:

```
sse={}
tx_frequency = tx_user[['Frequency']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_frequency)
    tx_frequency["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



In [46]:

```
#clustering for frequency
kmeans = KMeans(n_clusters=4)
kmeans.fit(tx_user[['Frequency']])
tx_user['FrequencyCluster'] = kmeans.predict(tx_user[['Frequency']])
```

Height: 768

In [47]:

```
tx_user.groupby('FrequencyCluster')['Frequency'].describe()
```

Out[47]:

FrequencyCluster	count	mean	std	min	25%	50%	75%	max
0	154123.0	15.931860	2.846376	12.0	13.0	16.0	18.0	21.0
1	321444.0	6.307319	2.519180	3.0	4.0	6.0	8.0	11.0
2	98600.0	27.044665	3.585988	22.0	24.0	27.0	30.0	34.0
3	25833.0	42.075640	12.293973	35.0	36.0	39.0	44.0	524.0

In [48]:

```
#order frequency clusters and show the characteristics
tx_user = order_cluster('FrequencyCluster', 'Frequency',tx_user,True)
```

Monetary Value

In [49]:

```
#calculate monetary value, create a dataframe with it
tx_amount = tx_m.groupby('id').transamount.sum().reset_index()
```

In [50]:

```
tx_amount.columns = ['CustomerID','transamount']
```

In [51]:

```
tx_amount.head()
```

Out[51]:

	CustomerID	transamount
0	1	45462.22
1	2	2085.55
2	3	2468.68
3	4	32896.76
4	5	8546.00

In [52]:

```
#add Amount column to tx_user
tx_user = pd.merge(tx_user, tx_amount, on='CustomerID')
```

Height: 768

In [53]:

```
tx_user.transamount.describe()
```

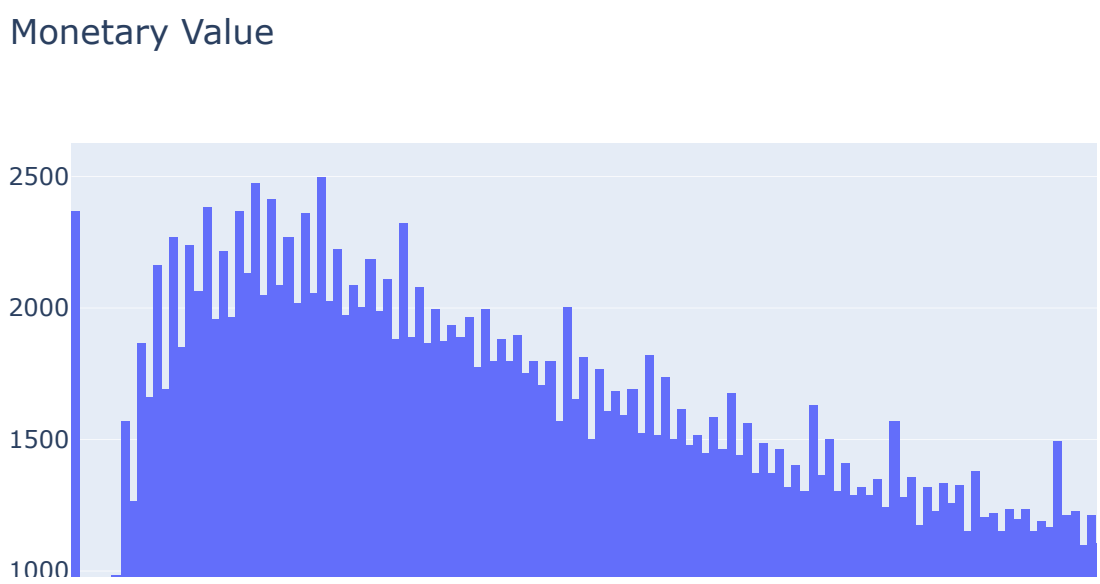
Out[53]:

```
count    6.000000e+05
mean     2.402960e+04
std      3.857610e+04
min      4.000000e-02
25%      4.084000e+03
50%      1.140000e+04
75%      2.857612e+04
max      2.050001e+06
Name: transamount, dtype: float64
```

Height: 768

In [54]:

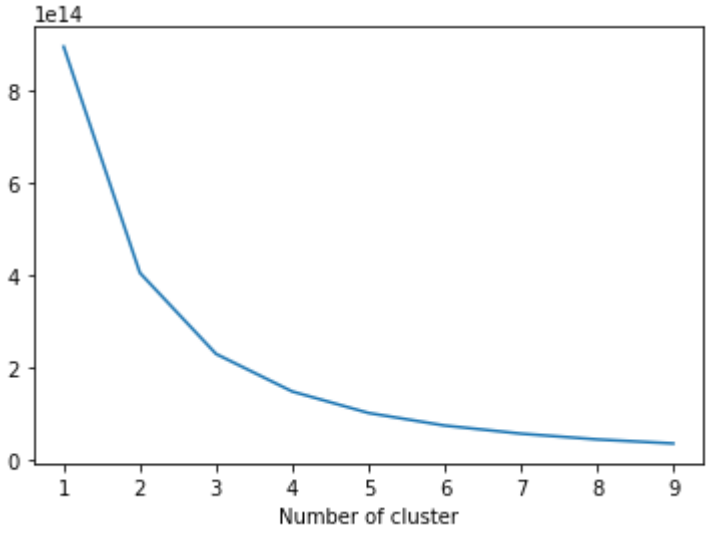
```
#plot Amount
plot_data = [
    go.Histogram(
        x=tx_user.query('transamount < 10000')['transamount']
    )
]

plot_layout = go.Layout(
        title='Monetary Value'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```



Height: 768

In [55]:

```
sse={}
tx_amount = tx_user[['transamount']]
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(tx_amount)
    tx_amount["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



In [56]:

```
#Amount clusters
kmeans = KMeans(n_clusters=4)
kmeans.fit(tx_user[['transamount']])
tx_user['AmountCluster'] = kmeans.predict(tx_user[['transamount']])
```

In [57]:

```
#ordering clusters and who the characteristics
tx_user = order_cluster('AmountCluster', 'transamount',tx_user,True)
```

Height: 768

In [58]:

```
tx_user.groupby('AmountCluster')['transamount'].describe()
```

Out[58]:

AmountCluster	count	mean	std	min	25%	50%	
0	461472.0	9797.829033	8134.019541	0.04	3001.1825	7374.0	15(
1	112886.0	52026.127788	16666.605637	30927.00	38019.0000	47902.0	63
2	23413.0	136739.861407	37927.891759	94453.40	106705.5000	124839.3	156
3	2229.0	368695.363647	154422.243706	252957.00	277648.3700	317070.0	397

Overall Segmentation

In [59]:

```
tx_user.head()
```

Out[59]:

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster	tran
0	253987	152.0	145	2	6	0	
1	174775	221.0	130	2	6	0	
2	518115	999.0	145	2	4	0	
3	164025	999.0	131	2	6	0	
4	223436	999.0	139	2	4	0	

In [60]:

```
#building overall segmentation
tx_user['OverallScore'] = tx_user['RecencyCluster'] + tx_user['FrequencyCluster'] + tx_user
```

Height: 768

In [61]:

```
tx_user.groupby('OverallScore')['Recency','Frequency','transamount'].mean()
```

Out[61]:

OverallScore	Recency	Frequency	transamount
0	344.208443	4.797577	5566.213660
1	219.922028	5.772195	8612.111184
2	126.601263	6.956985	11011.263956
3	51.655022	9.260179	13977.317223
4	34.870130	15.554076	22735.067581
5	24.536242	22.911336	34982.383106
6	17.275569	29.572266	56997.322794
7	12.784502	35.828957	95350.816726
8	10.997497	40.600819	185134.888120
9	8.953252	48.282520	393099.991301

In [62]:

```
tx_user.groupby('OverallScore')['Recency'].count()
```

Out[62]:

```
OverallScore
0     39867
1     67268
2    103128
3    135545
4    106599
5     77066
6     46903
7     18738
8      4394
9       492
Name: Recency, dtype: int64
```

In [63]:

```
#assign segment names
tx_user['Segment'] = 'Low-Value'
tx_user.loc[tx_user['OverallScore']>2,'Segment'] = 'Mid-Value'
tx_user.loc[tx_user['OverallScore']>4,'Segment'] = 'High-Value'
```

Height: 768

In [64]:

```
tx_user.head()
```

Out[64]:

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster	tran
0	253987	152.0	145	2	6	0	
1	174775	221.0	130	2	6	0	
2	518115	999.0	145	2	4	0	
3	164025	999.0	131	2	6	0	
4	223436	999.0	139	2	4	0	

Adding new features

In [65]:

```
tx_m.head()
```

Out[65]:

	bank	id	transdate	transamount
0	SBI	253987	2018-11-04	499.0
1	SBI	127456	2018-11-05	13700.0
2	SBI	513643	2018-11-21	460.0
3	SBI	3773	2018-11-25	158.0
4	SBI	346051	2018-11-21	3040.0

First, we create a dataframe with Customer ID and transday (not datetime). Then we will remove the duplicates since customers can do multiple purchases in a day and difference will become 0 for those.

In [66]:

```
#create a dataframe with CustomerID and Invoice Date
tx_day_order = tx_m[['id','transdate']]
```

In [67]:

```
#Convert Invoice Datetime to day
tx_day_order['transday'] = tx_m['transdate'].dt.date
```

In [68]:

```
tx_day_order = tx_day_order.sort_values(['id','transdate'])
```

Height: 768

In [69]:

```
#Drop duplicates
tx_day_order = tx_day_order.drop_duplicates(subset=['id','transday'],keep='first')
```

Next, by using shift, we create new columns with the dates of last 3 purchases and see how our dataframe looks like:

In [70]:

```
#shifting last 3 purchase dates
tx_day_order['Prevtransdate'] = tx_day_order.groupby('id')['transday'].shift(1)
tx_day_order['T2transdate'] = tx_day_order.groupby('id')['transday'].shift(2)
tx_day_order['T3transdate'] = tx_day_order.groupby('id')['transday'].shift(3)
```

In [71]:

```
tx_day_order = tx_day_order.rename(columns = {'id':'CustomerID'})
tx_day_order.head()
```

Out[71]:

	CustomerID	transdate	transday	Prevtransdate	T2transdate	T3transdate
5088386	1	2018-01-05	2018-01-05	NaN	NaN	NaN
5131398	1	2018-01-26	2018-01-26	2018-01-05	NaN	NaN
1652944	1	2018-02-21	2018-02-21	2018-01-26	2018-01-05	NaN
7023612	1	2018-03-24	2018-03-24	2018-02-21	2018-01-26	2018-01-05
5580547	1	2018-04-07	2018-04-07	2018-03-24	2018-02-21	2018-01-26

Let’s begin calculating the difference in days for each invoice date

In [72]:

```
tx_day_order['DayDiff'] = (tx_day_order['transday'] - tx_day_order['Prevtransdate']).dt.day
tx_day_order['DayDiff2'] = (tx_day_order['transday'] - tx_day_order['T2transdate']).dt.days
tx_day_order['DayDiff3'] = (tx_day_order['transday'] - tx_day_order['T3transdate']).dt.days
```

Height: 768

In [73]:

```
tx_day_order.head(10)
```

Out[73]:

	CustomerID	transdate	transday	Prevtransdate	T2transdate	T3transdate	DayDiff	Da
5088386	1	2018-01-05	2018-01-05	NaN	NaN	NaN	NaN	
5131398	1	2018-01-26	2018-01-26	2018-01-05	NaN	NaN	21.0	
1652944	1	2018-02-21	2018-02-21	2018-01-26	2018-01-05	NaN	26.0	
7023612	1	2018-03-24	2018-03-24	2018-02-21	2018-01-26	2018-01-05	31.0	
5580547	1	2018-04-07	2018-04-07	2018-03-24	2018-02-21	2018-01-26	14.0	
5879417	1	2018-04-11	2018-04-11	2018-04-07	2018-03-24	2018-02-21	4.0	
6185768	1	2018-04-30	2018-04-30	2018-04-11	2018-04-07	2018-03-24	19.0	
573952	1	2018-06-07	2018-06-07	2018-04-30	2018-04-11	2018-04-07	38.0	
5261275	1	2018-07-03	2018-07-03	2018-06-07	2018-04-30	2018-04-11	26.0	
6481527	1	2018-07-06	2018-07-06	2018-07-03	2018-06-07	2018-04-30	3.0	

For each customer ID, we utilize .agg() method to find out the mean and standard deviation of the difference between purchases in days

In [74]:

```
tx_day_diff = tx_day_order.groupby('CustomerID').agg({'DayDiff': ['mean','std']}).reset_ind
```

In [75]:

```
tx_day_diff.columns = ['CustomerID', 'DayDiffMean','DayDiffStd']
```

Height: 768

In [76]:

```
tx_day_diff.head()
```

Out[76]:

	CustomerID	DayDiffMean	DayDiffStd
0	1	17.000000	12.976903
1	2	45.750000	54.267086
2	3	146.666667	64.072875
3	4	56.500000	30.405592
4	5	15.000000	NaN

The calculation above is quite useful for customers who have many purchases. But we can't say the same for the ones with 1–2 purchases. For instance, it is too early to tag a customer as frequent who has only 2 purchases but back to back. We only keep customers who have > 3 purchases by using the following line:

In [77]:

```
tx_day_order_last = tx_day_order.drop_duplicates(subset=['CustomerID'],keep='last')
```

In [78]:

```
tx_day_order_last.head(10)
```

Out[78]:

	CustomerID	transdate	transday	Prevtransdate	T2transdate	T3transdate	DayDiff	Da
2285525	1	2019-03-23	2019-03-23	2019-03-20	2019-03-19	2019-02-21	3.0	
1174931	2	2018-08-03	2018-08-03	2018-08-01	2018-05-30	2018-02-04	2.0	
2228530	3	2019-03-26	2019-03-26	2018-08-20	2018-05-18	2018-01-10	218.0	
24074	4	2018-11-18	2018-11-18	2018-10-14	2018-07-28	NaN	35.0	
5674439	5	2018-07-21	2018-07-21	2018-07-06	NaN	NaN	15.0	
3765135	6	2019-02-05	2019-02-05	2018-10-14	2018-10-09	NaN	114.0	
6195585	7	2018-07-28	2018-07-28	2018-07-21	2018-07-03	2018-06-09	7.0	
2400782	8	2019-03-12	2019-03-12	2019-02-05	2018-12-10	2018-11-11	35.0	
2658981	9	2019-03-02	2019-03-02	2018-12-19	2018-12-10	2018-10-07	73.0	
2338118	10	2019-03-09	2019-03-09	2019-03-07	2019-03-03	2019-02-21	2.0	

Finally, we drop NA values, merge new dataframes with tx_user and apply .get_dummies() for converting categorical values Height: 768

In [79]:

```
tx_day_order_last = tx_day_order_last.dropna()
```

In [80]:

```
tx_day_order_last = pd.merge(tx_day_order_last, tx_day_diff, on='CustomerID')
```

In [81]:

```
tx_user = pd.merge(tx_user, tx_day_order_last[['CustomerID','DayDiff','DayDiff2','DayDiff3'
```

In [82]:

```
tx_user.head()
```

Out[82]:

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster	tran
0	174775	221.0	130	2	6	0	
1	518115	999.0	145	2	4	0	
2	164025	999.0	131	2	6	0	
3	128036	999.0	135	2	5	0	
4	159866	999.0	96	2	7	0	

In [83]:

```
len(tx_user)
```

Out[83]:

519487

Grouping the label

In [84]:

```
#create tx_class as a copy of tx_user before applying get_dummies
tx_class = tx_user.copy()
```

In [85]:

```
tx_class = pd.get_dummies(tx_class)
```

Height: 768

In [86]:

```
tx_class.tail(10)
```

Out[86]:

	CustomerID	NextPurchaseDay	Recency	RecencyCluster	Frequency	FrequencyCluster
519477	168261	4.0	0	3	90	3
519478	542926	19.0	1	3	35	3
519479	575493	5.0	4	3	35	3
519480	3031	999.0	7	3	58	3
519481	342432	84.0	7	3	54	3
519482	582670	5.0	0	3	101	3
519483	262181	12.0	5	3	45	3
519484	413669	5.0	3	3	58	3
519485	230273	17.0	14	3	50	3
519486	579598	999.0	30	3	37	3

Selecting a Machine Learning Model

Before jumping into choosing the model, we need to take two actions. First, we need to identify the classes in our label. Generally, percentiles give the right for that. Let's use .describe() method to see them in NextPurchaseDay:

In [87]:

```
tx_user.NextPurchaseDay.describe()
```

Out[87]:

```
count    519487.000000
mean        378.624437
std         440.702341
min           1.000000
25%          32.000000
50%          95.000000
75%         999.000000
max         999.000000
Name: NextPurchaseDay, dtype: float64
```

We will have three classes:

- 0–20: Customers that will purchase in 0–20 days — Class name: 2
- 21–49: Customers that will purchase in 21–49 days — Class name: 1
- ≥ 50: Customers that will purchase in more than 50 days — Class name: 0

Height: 768

In [88]:

```
tx_class['NextPurchaseDayRange'] = 2
tx_class.loc[tx_class.NextPurchaseDay>20,'NextPurchaseDayRange'] = 1
tx_class.loc[tx_class.NextPurchaseDay>50,'NextPurchaseDayRange'] = 0
```

In [89]:

```
tx_class.NextPurchaseDayRange.value_counts()/len(tx_user)
```

Out[89]:

```
0    0.653448
1    0.182149
2    0.164403
Name: NextPurchaseDayRange, dtype: float64
```

For this particular problem, we want to use the model which gives the highest accuracy. Let's split train and test tests and measure the accuracy of different models

In [90]:

```
tx_class = tx_class.drop('NextPurchaseDay',axis=1)
```

In [91]:

```
len(tx_class)
```

Out[91]:

```
519487
```

In [92]:

```
#train & test split
X, y = tx_class.drop('NextPurchaseDayRange',axis=1), tx_class.NextPurchaseDayRange
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=44)
```

In [93]:

```
#create an array of models
models = []
models.append(("LR",LogisticRegression()))
models.append(("NB",GaussianNB()))
models.append(("RF",RandomForestClassifier()))
models.append(("Dtree",DecisionTreeClassifier()))
models.append(("XGB",xgb.XGBClassifier()))
models.append(("KNN",KNeighborsClassifier()))
```

Height: 768

In [94]:

```
#measure the accuracy
for name,model in models:
    kfold = KFold(n_splits=2, random_state=22)
    cv_result = cross_val_score(model,X_train,y_train, cv = kfold,scoring = "accuracy")
    print(name, cv_result)
```

```
LR [nan nan]
NB [0.75411343 0.75519986]
RF [0.77828629 0.77937284]
Dtree [0.72610506 0.72541556]
[17:36:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old beha
vior.
[17:36:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old beha
vior.
XGB [0.78298804 0.78549429]
KNN [0.61013499 0.61133141]
```

Let's move forward with XGBoost to show how we can improve an existing model with some advanced techniques.

For improving it further, we'll do Hyperparameter Tuning.

In [95]:

```
xgb_model = xgb.XGBClassifier().fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'
       .format(xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.2f}'
       .format(xgb_model.score(X_test[X_train.columns], y_test)))
```

```
[17:38:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old beha
vior.
Accuracy of XGB classifier on training set: 0.80
Accuracy of XGB classifier on test set: 0.78
```

In [96]:

```
y_pred = xgb_model.predict(X_test)
```

Height: 768

In [99]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89     68042
           1       0.54      0.40      0.46     18939
           2       0.63      0.80      0.70     16917

    accuracy                           0.78    103898
   macro avg       0.68      0.70      0.68    103898
weighted avg       0.78      0.78      0.78    103898
```

XGBClassifier has many parameters. For this example, we will select max_depth and min_child_weight. The code below will generate the best values for these parameters:

In [100]:

```
from sklearn.model_selection import GridSearchCV

param_test1 = {
 'max_depth':range(3,10,2),
 'min_child_weight':range(1,6,2)
}
gsearch1 = GridSearchCV(estimator = xgb.XGBClassifier(),
param_grid = param_test1, scoring='accuracy',n_jobs=-1,iid=False, cv=2)
gsearch1.fit(X_train,y_train)
gsearch1.best_params_, gsearch1.best_score_
```

```
[17:53:58] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old beha
vior.
```

Out[100]:

```
({'max_depth': 3, 'min_child_weight': 1}, 0.7858148334532599)
```

The algorithm says the best values are 3 and 1 for max_depth and min_child_weight respectively. Check out how it improves accuracy

Height: 768

In [101]:

```
xgb_model = xgb.XGBClassifier(max_depth=3, min_child_weight=5).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'
      .format(xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.2f}'
      .format(xgb_model.score(X_test[X_train.columns], y_test)))
```

```
[17:54:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old beha
vior.
Accuracy of XGB classifier on training set: 0.79
Accuracy of XGB classifier on test set: 0.79
```

Our score increased from 78% to 79%.

In [102]:

```
y_pred = xgb_model.predict(X_test)
```

In [103]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89     68042
           1       0.54      0.40      0.46     18939
           2       0.63      0.81      0.71     16917

    accuracy                           0.79    103898
   macro avg       0.69      0.70      0.68    103898
weighted avg       0.78      0.79      0.78    103898
```

In [104]:

```
y_pred
```

Out[104]:

```
array([0, 1, 0, ..., 0, 0, 2], dtype=int64)
```

In [ ]: