Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAACAccredited with "A" Grade (CGPA : 3.18)

**Department of Computer Science and Engineering (Data Science)**

**Subject: Machine Learning – I (DJ19DSC402)**

**AY: 2022-23**

**Name: Yash Thakar** **SAP ID.: 60009210205**

Colab Links:

Customer Segmentation and Customer Churn Classification:

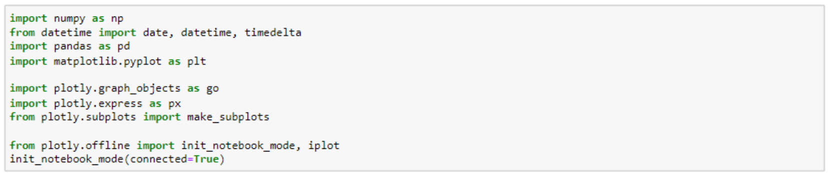
https://colab.research.google.com/drive/1e4k24Q2WofuHN8ykUimO9jBuBHrqbgd8

Sales Prediction:

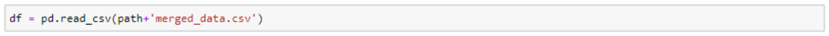
https://colab.research.google.com/drive/1FtT0cTZpG4SUipMQbaPrn1kjfS8wiq02

# Customer Segmentation and Customer Churn Classification

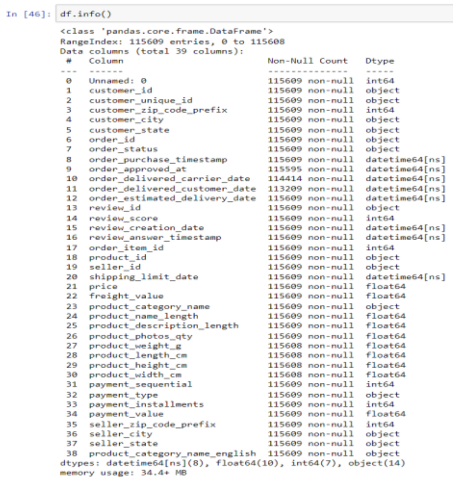
Importing Necessary Libraries and processed dataset from previously done EDA



```
import numpy as np
from datetime import date, datetime, timedelta
import pandas as pd
import matplotlib.pyplot as plt

import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)
```



```
df = pd.read_csv(path+'merged_data.csv')
```

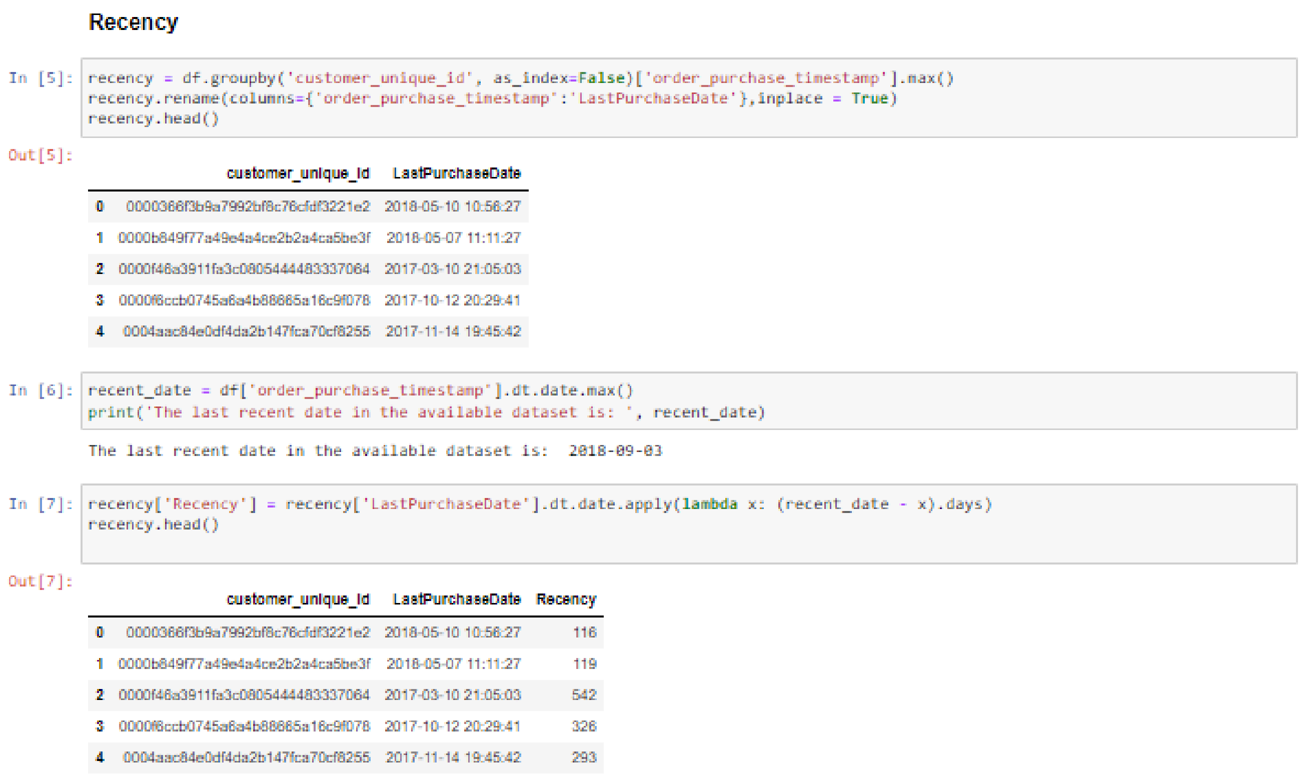


```
In [46]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115609 entries, 0 to 115608
Data columns (total 39 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Unnamed: 0                     115609 non-null  int64
 1   customer_id                    115609 non-null  object
 2   customer_unique_id             115609 non-null  object
 3   customer_zip_code_prefix       115609 non-null  int64
 4   customer_city                  115609 non-null  object
 5   customer_state                 115609 non-null  object
 6   order_id                       115609 non-null  object
 7   order_status                   115609 non-null  object
 8   order_purchase_timestamp       115609 non-null  datetime64[ns]
 9   order_approved_at              115595 non-null  datetime64[ns]
 10  order_delivered_carrier_date   114414 non-null  datetime64[ns]
 11  order_delivered_customer_date  113209 non-null  datetime64[ns]
 12  order_estimated_delivery_date  115609 non-null  datetime64[ns]
 13  review_id                      115609 non-null  object
 14  review_score                   115609 non-null  int64
 15  review_creation_date           115609 non-null  datetime64[ns]
 16  review_answer_timestamp        115609 non-null  datetime64[ns]
 17  order_item_id                  115609 non-null  int64
 18  product_id                     115609 non-null  object
 19  seller_id                      115609 non-null  object
 20  shipping_limit_date            115609 non-null  datetime64[ns]
 21  price                          115609 non-null  float64
 22  freight_value                  115609 non-null  float64
 23  product_category_name          115609 non-null  object
 24  product_name_length            115609 non-null  float64
 25  product_description_length     115609 non-null  float64
 26  product_photos_qty             115609 non-null  float64
 27  product_weight_g               115608 non-null  float64
 28  product_length_cm              115608 non-null  float64
 29  product_height_cm              115608 non-null  float64
 30  product_width_cm               115608 non-null  float64
 31  payment_sequential             115609 non-null  int64
 32  payment_type                   115609 non-null  object
 33  payment_installments           115609 non-null  int64
 34  payment_value                  115609 non-null  float64
 35  seller_zip_code_prefix         115609 non-null  int64
 36  seller_city                    115609 non-null  object
 37  seller_state                   115609 non-null  object
 38  product_category_name_english  115609 non-null  object
dtypes: datetime64[ns](8), float64(10), int64(7), object(14)
memory usage: 34.4+ MB
```

RFM analysis:
RFM analysis is a marketing technique used to quantitatively rank and group customers based on the recency, frequency and monetary total of their recent transactions to identify the best customers and perform targeted marketing campaigns.

RFM analysis ranks each customer on the following factors:

- Recency: How recent was the customer's last purchase? Customers who recently made a purchase will still have the product on their mind and are more likely to purchase or use the product again. Businesses often measure recency in days. But, depending on the product, they may measure it in years, weeks or even hours.
- Frequency: How often did this customer make a purchase in a given period? Customers who purchased once are often are more likely to purchase again. Additionally, first time customers may be good targets for follow-up advertising to convert them into more frequent customers.
- Monetary: How much money did the customer spend in a given period? Customers who spend a lot of money are more likely to spend money in the future and have a high value to a business.



## Recency

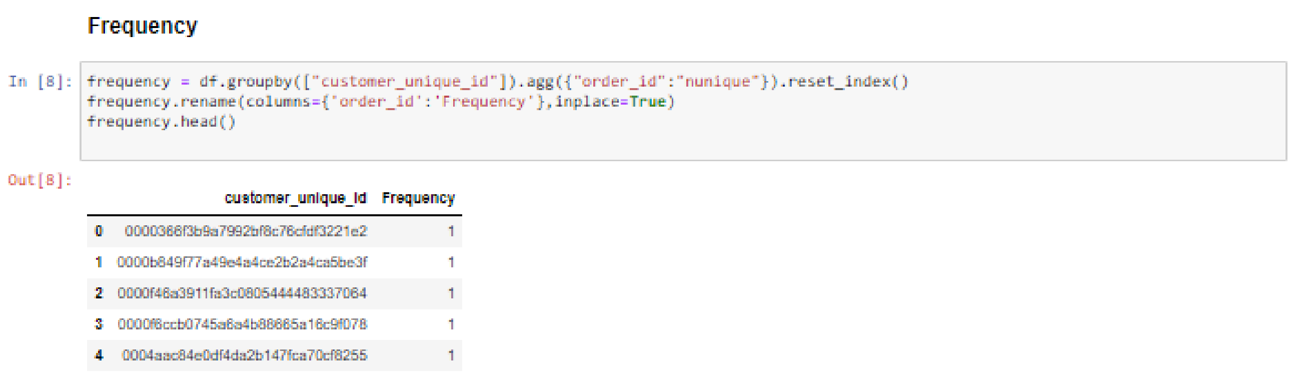
```
In [5]: recency = df.groupby('customer_unique_id', as_index=False)['order_purchase_timestamp'].max()
        recency.rename(columns={'order_purchase_timestamp':'LastPurchaseDate'},inplace = True)
        recency.head()
Out[5]:
```

| | customer_unique_id | LastPurchaseDate |
|---|---|---|
| 0 | 0000366f3b9a7992bf8c76cfdf3221e2 | 2018-05-10 10:56:27 |
| 1 | 0000b849f77a49e4a4ce2b2a4ca5be3f | 2018-05-07 11:11:27 |
| 2 | 0000f46a3911fa3c0805444483337064 | 2017-03-10 21:05:03 |
| 3 | 0000f6ccb0745a6a4b88665a16c9f078 | 2017-10-12 20:29:41 |
| 4 | 0004aac84e0df4da2b147fca70cf8255 | 2017-11-14 19:45:42 |

```
In [6]: recent_date = df['order_purchase_timestamp'].dt.date.max()
        print('The last recent date in the available dataset is: ', recent_date)

        The last recent date in the available dataset is:  2018-09-03
```

```
In [7]: recency['Recency'] = recency['LastPurchaseDate'].dt.date.apply(lambda x: (recent_date - x).days)
        recency.head()
Out[7]:
```

| | customer_unique_id | LastPurchaseDate | Recency |
|---|---|---|---|
| 0 | 0000366f3b9a7992bf8c76cfdf3221e2 | 2018-05-10 10:56:27 | 116 |
| 1 | 0000b849f77a49e4a4ce2b2a4ca5be3f | 2018-05-07 11:11:27 | 119 |
| 2 | 0000f46a3911fa3c0805444483337064 | 2017-03-10 21:05:03 | 542 |
| 3 | 0000f6ccb0745a6a4b88665a16c9f078 | 2017-10-12 20:29:41 | 326 |
| 4 | 0004aac84e0df4da2b147fca70cf8255 | 2017-11-14 19:45:42 | 293 |

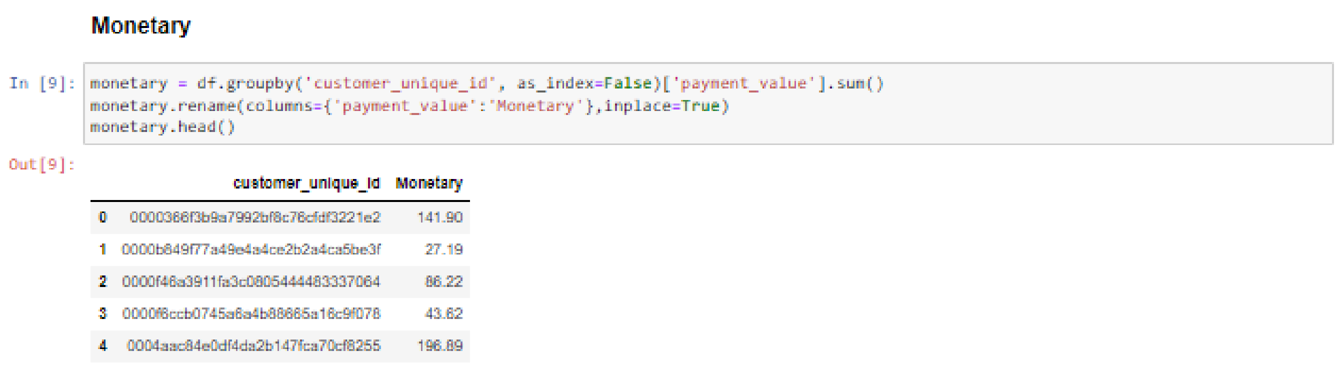


### Frequency

In [8]:
```
frequency = df.groupby(["customer_unique_id"]).agg({"order_id":"nunique"}).reset_index()
frequency.rename(columns={'order_id':'Frequency'},inplace=True)
frequency.head()
```

Out[8]:

| | customer_unique_id | Frequency |
|---|---|---|
| **0** | 0000366f3b9a7992bf8c76cfdf3221e2 | 1 |
| **1** | 0000b849f77a49e4a4ce2b2a4ca5be3f | 1 |
| **2** | 0000f46a3911fa3c0805444483337064 | 1 |
| **3** | 0000f6ccb0745a6a4b88665a16c9f078 | 1 |
| **4** | 0004aac84e0df4da2b147fca70cf8255 | 1 |



### Monetary

In [9]:
```
monetary = df.groupby('customer_unique_id', as_index=False)['payment_value'].sum()
monetary.rename(columns={'payment_value':'Monetary'},inplace=True)
monetary.head()
```

Out[9]:

| | customer_unique_id | Monetary |
|---|---|---|
| **0** | 0000366f3b9a7992bf8c76cfdf3221e2 | 141.90 |
| **1** | 0000b849f77a49e4a4ce2b2a4ca5be3f | 27.19 |
| **2** | 0000f46a3911fa3c0805444483337064 | 86.22 |
| **3** | 0000f6ccb0745a6a4b88665a16c9f078 | 43.62 |
| **4** | 0004aac84e0df4da2b147fca70cf8255 | 196.89 |

## Merging RFM

In [10]:
```
rfm = recency.merge(frequency, on='customer_unique_id')
rfm = rfm.merge(monetary, on='customer_unique_id').drop(columns='LastPurchaseDate')
rfm.head()
```

Out[10]:

| | customer_unique_id | Recency | Frequency | Monetary |
|---|---|---|---|---|
| **0** | 0000366f3b9a7992bf8c76cfdf3221e2 | 116 | 1 | 141.90 |
| **1** | 0000b849f77a49e4a4ce2b2a4ca5be3f | 119 | 1 | 27.19 |
| **2** | 0000f46a3911fa3c0805444483337064 | 542 | 1 | 86.22 |
| **3** | 0000f6ccb0745a6a4b88665a16c9f078 | 326 | 1 | 43.62 |
| **4** | 0004aac84e0df4da2b147fca70cf8255 | 293 | 1 | 196.89 |

Labelling RFM (to get scores):

Recency Labels

```
ll_r = rfm.Recency.quantile(0.25)
mid_r = rfm.Recency.quantile(0.50)
ul_r = rfm.Recency.quantile(0.75)
print(ll_r, mid_r, ul_r)
```

```
119.0 223.0 352.0
```

```
def recency_label(recent):
    if recent <= ll_r:
        return 1
    elif (recent > ll_r) and (recent <= mid_r):
        return 2
    elif (recent > mid_r) and (recent <= ul_r):
        return 3
    elif recent > ul_r:
        return 4
```

```
rfm['recency_label'] = rfm.Recency.apply(recency_label)
rfm.head()
```

| | customer_unique_id | Recency | Frequency | Monetary | recency_label |
|---|---|---|---|---|---|
| 0 | 0000366f3b9a7992bf8c76cfdf3221e2 | 116 | 1 | 141.90 | 1 |
| 1 | 0000b849f77a49e4a4ce2b2a4ca5be3f | 119 | 1 | 27.19 | 1 |
| 2 | 0000f46a3911fa3c0805444483337064 | 542 | 1 | 86.22 | 4 |
| 3 | 0000f6ccb0745a6a4b88665a16c9f078 | 326 | 1 | 43.62 | 3 |
| 4 | 0004aac84e0df4da2b147fca70cf8255 | 293 | 1 | 196.89 | 3 |

Recency label breakdown:

1 - These are the customers who whose visit date(s) are the most recent. (Recency value within the 25% quantile)

2 - These are the customers who whose visit date(s) are not very recent. (Recency value between 25% and 50% quantile)

3 - These are the customers who whose visit date(s) are somewhat recent. (Recency value between 50% and 75% quantile)

4 - These are the customers who whose visit date(s) are the oldest. (Recency value more than 75% quantile)

Frequency Lables:

```
rfm.Frequency.value_counts()
```

```
1     90589
2      2581
3       179
4        30
5         9
6         3
7         3
9         1
15        1
Name: Frequency, dtype: int64
```

```
def frequency_label(frequent):
    if frequent == 1:
        return 4
    elif frequent == 2:
        return 3
    elif frequent == 3:
        return 2
    elif frequent > 3:
        return 1
```

```
rfm['frequency_label'] = rfm.Frequency.apply(frequency_label)
rfm.head()
```

| | customer_unique_id | Recency | Frequency | Monetary | recency_label | frequency_label |
|---|---|---|---|---|---|---|
| **0** | 0000366f3b9a7992bf8c76cfdf3221e2 | 116 | 1 | 141.90 | 1 | 4 |
| **1** | 0000b849f77a49e4a4ce2b2a4ca5be3f | 119 | 1 | 27.19 | 1 | 4 |
| **2** | 0000f46a3911fa3c0805444483337064 | 542 | 1 | 86.22 | 4 | 4 |
| **3** | 0000f6ccb0745a6a4b88665a16c9f078 | 326 | 1 | 43.62 | 3 | 4 |
| **4** | 0004aac84e0df4da2b147fca70cf8255 | 293 | 1 | 196.89 | 3 | 4 |

Frequency label breakdown:

1 - These are the most frequent customers. (Frequency > 3)

2 - These are the frequent frequent customers. (Frequency = 3)

3 - These are the somewhat frequent customers. (Frequency = 2)

4 - These are the least frequent customers. (Frequency = 1)

Monetary Labels:

```
ll_m = rfm.Monetary.quantile(0.25)
mid_m = rfm.Monetary.quantile(0.50)
ul_m = rfm.Monetary.quantile(0.75)
print(ll_m, mid_m, ul_m)
```

```
64.0 113.03 203.39
```

```
def monetary_label(money):
    if money <= ll_m:
        return 4
    elif (money > ll_m) and (money <= mid_m):
        return 3
    elif (money > mid_m) and (money <= ul_m):
        return 2
    elif money > ul_m:
        return 1
```

```
rfm['monetary_label'] = rfm.Monetary.apply(monetary_label)
rfm.head()
```

| | customer_unique_id | Recency | Frequency | Monetary | recency_label | frequency_label | monetary_label |
|---|---|---|---|---|---|---|---|
| **0** | 0000366f3b9a7992bf8c76cfdf3221e2 | 116 | 1 | 141.90 | 1 | 4 | 2 |
| **1** | 0000b849f77a49e4a4ce2b2a4ca5be3f | 119 | 1 | 27.19 | 1 | 4 | 4 |
| **2** | 0000f46a3911fa3c0805444483337064 | 542 | 1 | 86.22 | 4 | 4 | 3 |
| **3** | 0000f6ccb0745a6a4b88665a16c9f078 | 326 | 1 | 43.62 | 3 | 4 | 4 |
| **4** | 0004aac84e0df4da2b147fca70cf8255 | 293 | 1 | 196.89 | 3 | 4 | 2 |

Monetary label breakdown:

1 - These are the customers who spend large amount. (Monetary value within the 25% quantile)

2 - These are the customers who spend good amount. (Monetary value between 25% and 50% quantile)

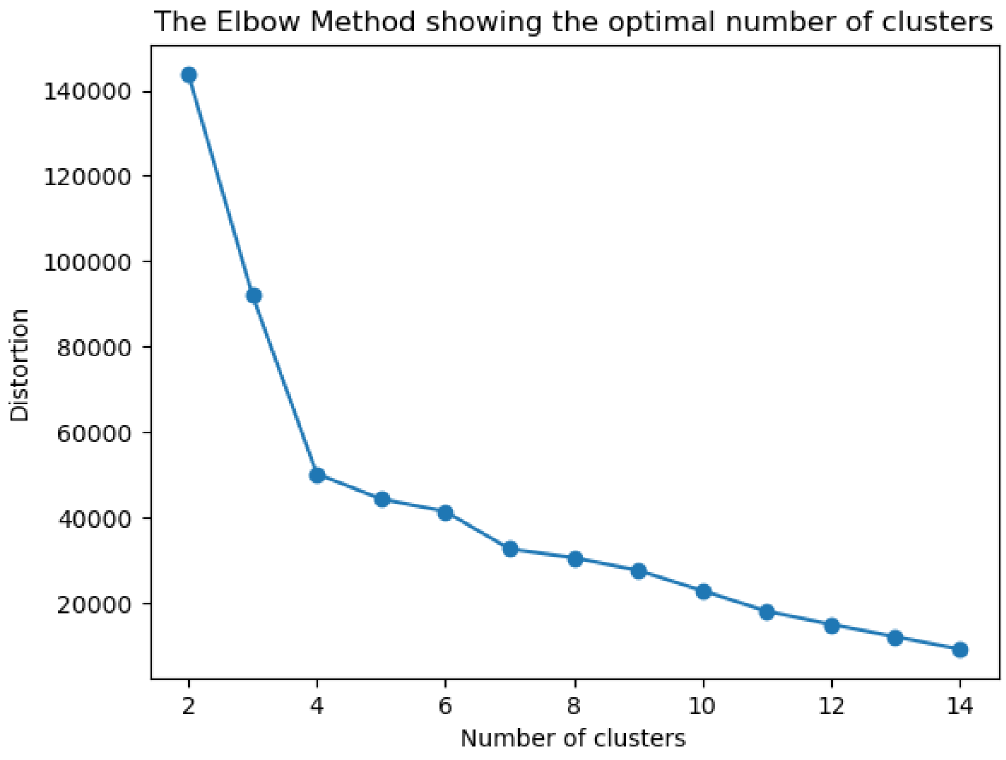
3 - These are the customers who spend moderately. (Monetary value between 50% and 75% quantile)

4 - These are the customers who spend the least. (Monetary value more than 75% quantile

Elbow Method to find optimal number of clusters:

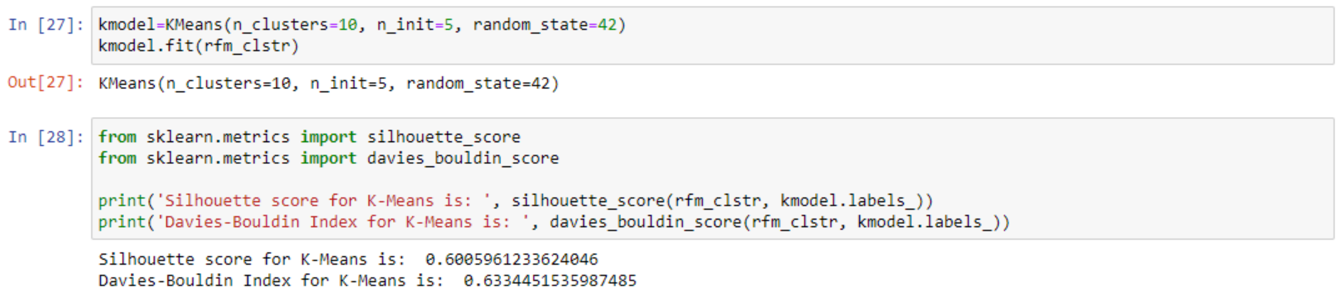
```
: rfm_clstr = rfm.drop(["customer_unique_id","Recency","Frequency","Monetary"],axis='columns')
```

```
: from sklearn.cluster import KMeans

  distortions=[]
  for i in range(2,15):
      kmodel=KMeans(n_clusters=i,n_init=5, random_state=42)
      kmodel.fit(rfm_clstr)
      distortions.append(kmodel.inertia_) # KMeans inertia = Sum of Squares Errors (SSE)

  plt.plot(range(2,15), distortions, marker='o')
  plt.xlabel('Number of clusters')
  plt.ylabel('Distortion')
  plt.title('The Elbow Method showing the optimal number of clusters')
  plt.show()
```



The number of clusters found to be optimal from the above chart is 10

Applying k-means with n_clusters = 10



```
In [27]: kmodel=KMeans(n_clusters=10, n_init=5, random_state=42)
         kmodel.fit(rfm_clstr)

Out[27]: KMeans(n_clusters=10, n_init=5, random_state=42)

In [28]: from sklearn.metrics import silhouette_score
         from sklearn.metrics import davies_bouldin_score

         print('Silhouette score for K-Means is: ', silhouette_score(rfm_clstr, kmodel.labels_))
         print('Davies-Bouldin Index for K-Means is: ', davies_bouldin_score(rfm_clstr, kmodel.labels_))

         Silhouette score for K-Means is:  0.6005961233624046
         Davies-Bouldin Index for K-Means is:  0.6334451535987485
```

We use Intrinsic evaluation measures as do not require ground truth labels.

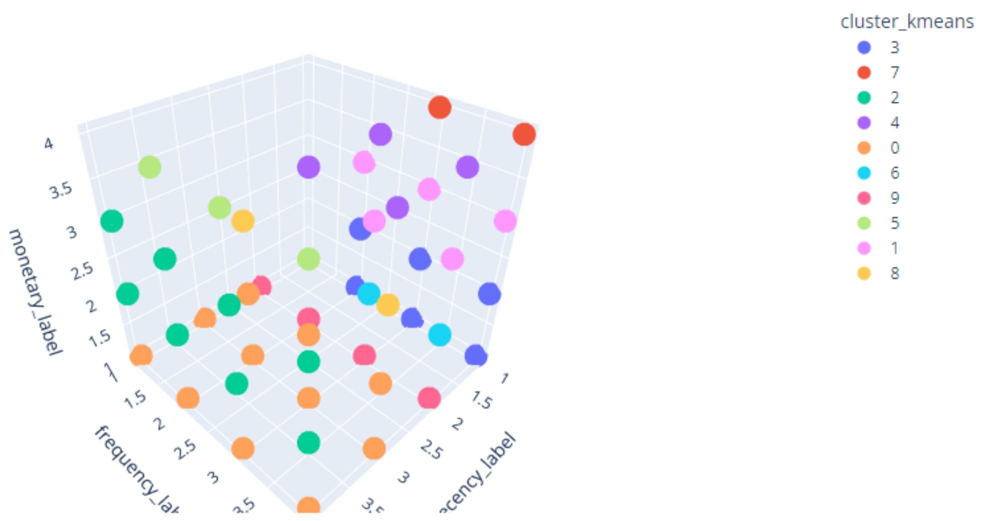
Silhouette Score measures the between-cluster distance against within-cluster distance. A higher score signifies better-defined clusters. The best value is 1 and the worst value is -1.
The Silhouette score for our clustering results is 0.6005

Davies-Bouldin Index measures the size of clusters against the average distance between clusters. A lower score signifies better-defined clusters. The Davies-Bouldin Index (DBI) has the lowest possible value of 0 and does not have an upper limit.
The Davies-Bouldin Index for our clustering results is 0.6334

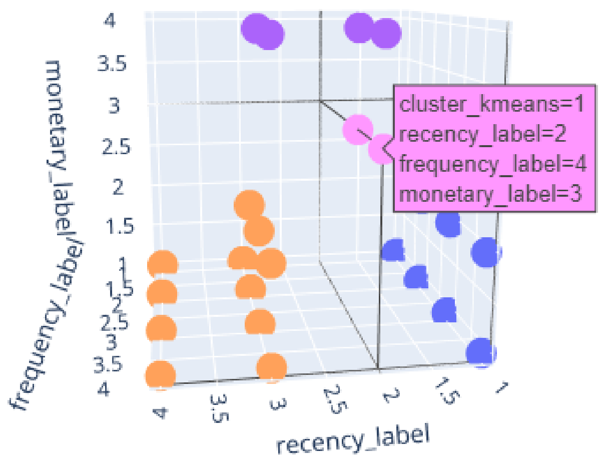
Overall indicating well-defined clusters.

Plotting clusters in 3 dimensions:

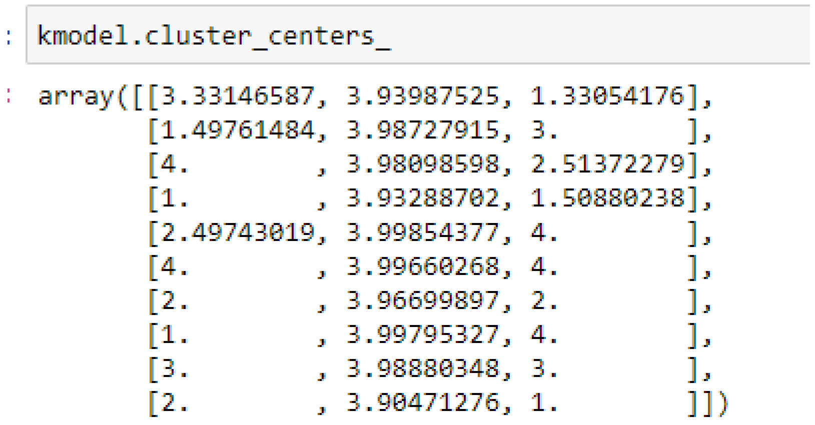
```
rfm_clstr["cluster_kmeans"] = kmodel.fit_predict(rfm_clstr)

rfm_clstr["cluster_kmeans"] = rfm_clstr["cluster_kmeans"].astype(str)

fig= px.scatter_3d(rfm_clstr, x='recency_label', y='frequency_label', z='monetary_label', color='cluster_kmeans',opacity=1)
fig.update_traces(marker_size = 10)
fig.show()
```



Selecting a certain cluster for better understanding of the visual:



Finding the cluster centers to better understand the clustering results:



```
kmodel.cluster_centers_
```

```
array([[3.33146587, 3.93987525, 1.33054176],
       [1.49761484, 3.98727915, 3.        ],
       [4.        , 3.98098598, 2.51372279],
       [1.        , 3.93288702, 1.50880238],
       [2.49743019, 3.99854377, 4.        ],
       [4.        , 3.99660268, 4.        ],
       [2.        , 3.96699897, 2.        ],
       [1.        , 3.99795327, 4.        ],
       [3.        , 3.98880348, 3.        ],
       [2.        , 3.90471276, 1.        ]])
```

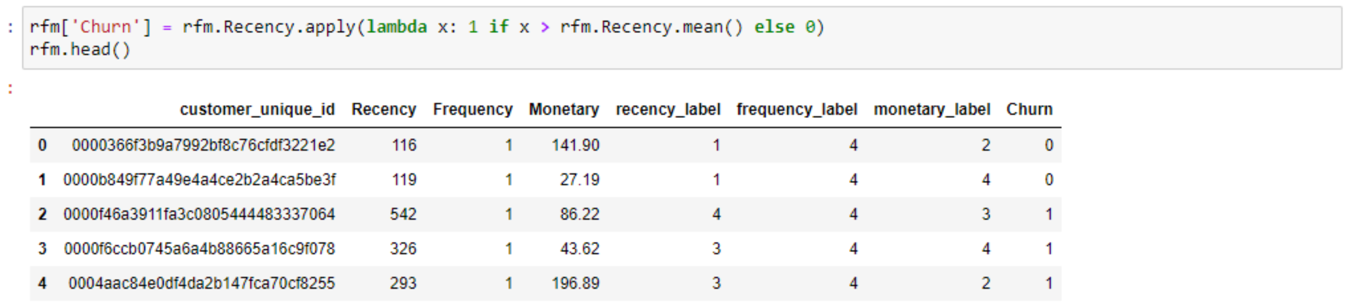
## CHURN CLASSIFICATION

The churn rate, also known as the rate of attrition or customer churn, is the rate at which customers stop doing business with an entity.

We used the recency column to frame the target variable. If the customer's recency falls above the average value of recency, we consider such customers as churned. The rest of the customers as not churned.

We used the mean of recency as the threshold as the recency is normally or symmetrically distributed.

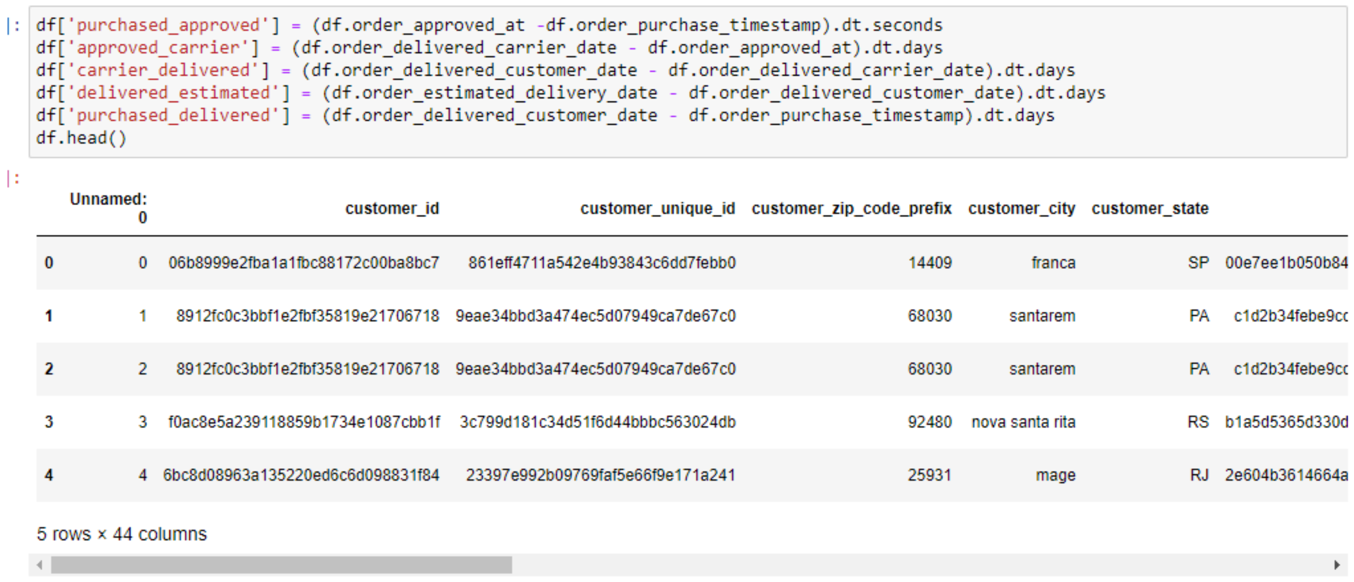
We will have to impute the target variable to the main dataframe and do the further classification algorithm.



```
rfm['Churn'] = rfm.Recency.apply(lambda x: 1 if x > rfm.Recency.mean() else 0)
rfm.head()
```

| | customer_unique_id | Recency | Frequency | Monetary | recency_label | frequency_label | monetary_label | Churn |
|---|---|---|---|---|---|---|---|---|
| **0** | 0000366f3b9a7992bf8c76cfdf3221e2 | 116 | 1 | 141.90 | 1 | 4 | 2 | 0 |
| **1** | 0000b849f77a49e4a4ce2b2a4ca5be3f | 119 | 1 | 27.19 | 1 | 4 | 4 | 0 |
| **2** | 0000f46a3911fa3c0805444483337064 | 542 | 1 | 86.22 | 4 | 4 | 3 | 1 |
| **3** | 0000f6ccb0745a6a4b88665a16c9f078 | 326 | 1 | 43.62 | 3 | 4 | 4 | 1 |
| **4** | 0004aac84e0df4da2b147fca70cf8255 | 293 | 1 | 196.89 | 3 | 4 | 2 | 1 |

## Finding certain new calculated attributes

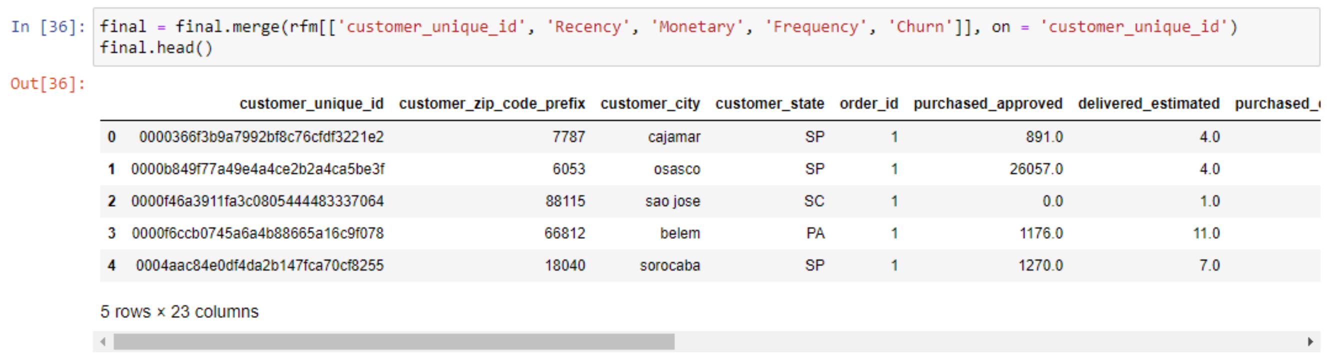


```
df['purchased_approved'] = (df.order_approved_at -df.order_purchase_timestamp).dt.seconds
df['approved_carrier'] = (df.order_delivered_carrier_date - df.order_approved_at).dt.days
df['carrier_delivered'] = (df.order_delivered_customer_date - df.order_delivered_carrier_date).dt.days
df['delivered_estimated'] = (df.order_estimated_delivery_date - df.order_delivered_customer_date).dt.days
df['purchased_delivered'] = (df.order_delivered_customer_date - df.order_purchase_timestamp).dt.days
df.head()
```

| | Unnamed: 0 | customer_id | customer_unique_id | customer_zip_code_prefix | customer_city | customer_state | |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 06b8999e2fba1a1fbc88172c00ba8bc7 | 861eff4711a542e4b93843c6dd7febb0 | 14409 | franca | SP | 00e7ee1b050b84 |
| **1** | 1 | 8912fc0c3bbf1e2fbf35819e21706718 | 9eae34bbd3a474ec5d07949ca7de67c0 | 68030 | santarem | PA | c1d2b34febe9cc |
| **2** | 2 | 8912fc0c3bbf1e2fbf35819e21706718 | 9eae34bbd3a474ec5d07949ca7de67c0 | 68030 | santarem | PA | c1d2b34febe9cc |
| **3** | 3 | f0ac8e5a239118859b1734e1087cbb1f | 3c799d181c34d51f6d44bbbc563024db | 92480 | nova santa rita | RS | b1a5d5365d330d |
| **4** | 4 | 6bc8d08963a135220ed6c6d098831f84 | 23397e992b09769faf5e66f9e171a241 | 25931 | mage | RJ | 2e604b3614664a |

5 rows × 44 columns

## Grouping by customer id and merging with RFM data



```
In [36]: final = final.merge(rfm[['customer_unique_id', 'Recency', 'Monetary', 'Frequency', 'Churn']], on = 'customer_unique_id')
         final.head()
```

```
Out[36]:
                 customer_unique_id  customer_zip_code_prefix  customer_city  customer_state  order_id  purchased_approved  delivered_estimated  purchased_
0  0000366f3b9a7992bf8c76cfdf3221e2                      7787        cajamar              SP         1               891.0                  4.0
1  0000b849f77a49e4a4ce2b2a4ca5be3f                      6053         osasco              SP         1             26057.0                  4.0
2  0000f46a3911fa3c0805444483337064                     88115       sao jose              SC         1                 0.0                  1.0
3  0000f6ccb0745a6a4b88665a16c9f078                     66812          belem              PA         1              1176.0                 11.0
4  0004aac84e0df4da2b147fca70cf8255                     18040       sorocaba              SP         1              1270.0                  7.0

5 rows × 23 columns
```

The Churn data obtained is balanced but as seen in the previous EDA, certain attributes have a lot of outliers

Outlier Treatment:

```
final_outlierTreated = final.copy()
```

```
for i in final_outlierTreated.select_dtypes(include = np.number).columns:
    q1 = final_outlierTreated[i].quantile(0.25)
    q3 = final_outlierTreated[i].quantile(0.75)
    iqr = q3 - q1
    ul = q3 + 1.5*iqr
    ll = q1 - 1.5*iqr
    final_outlierTreated[i] = np.where(final_outlierTreated[i]>ul,ul,final_outlierTreated[i])
    final_outlierTreated[i] = np.where(final_outlierTreated[i]<ll,ll,final_outlierTreated[i])
```

Grouping states in north south east and west to label encode

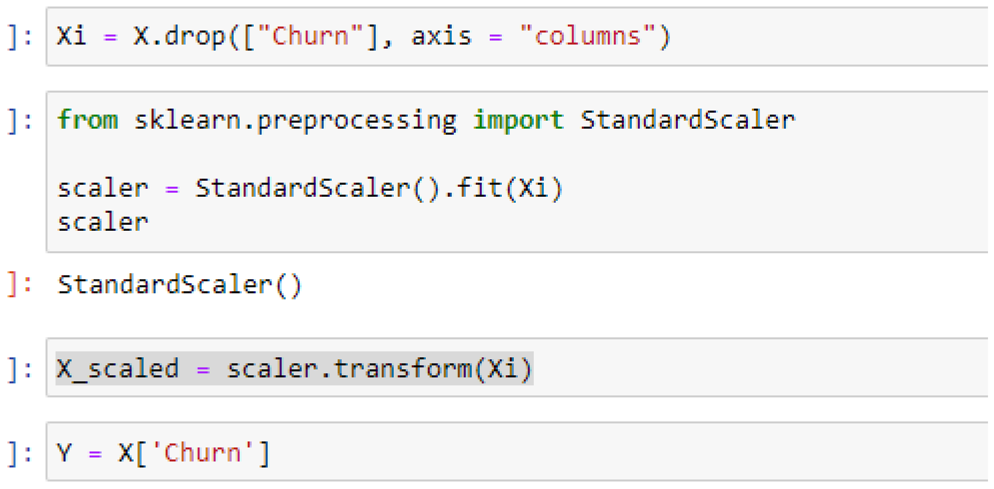
```
def state_encoding(state):
    if state in ['RS', 'SC', 'PR']:
        return 'southern'
    elif state in ['SP', 'RJ', 'MG', 'ES']:
        return 'southeastern'
    elif state in ['MT', 'MS', 'GO', 'DF']:
        return 'centralwestern'
    elif state in ['MA', 'PI', 'CE', 'RN', 'PB', 'PE', 'AL', 'SE', 'BA']:
        return 'northeastern'
    else:
        return 'northern'
```

```
final_outlierTreated['customer_state'] = final_outlierTreated['customer_state'].apply(state_encoding)
```

Since the payment_value feature is same as that of the Monetary feature, the former is dropped.

Similarly, customer_city is a multi-class feature, so encoding it would be useless. So we drop the feature.

After deleting Null values, Standard Scaler is applied



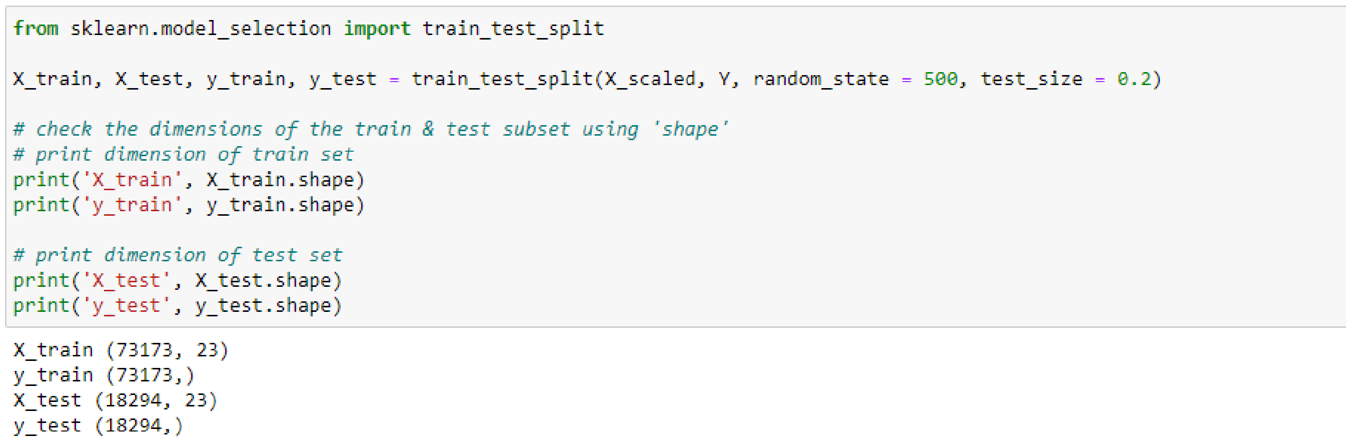
```
]: Xi = X.drop(["Churn"], axis = "columns")
```

```
]: from sklearn.preprocessing import StandardScaler

   scaler = StandardScaler().fit(Xi)
   scaler
```

```
]: StandardScaler()
```

```
]: X_scaled = scaler.transform(Xi)
```

```
]: Y = X['Churn']
```

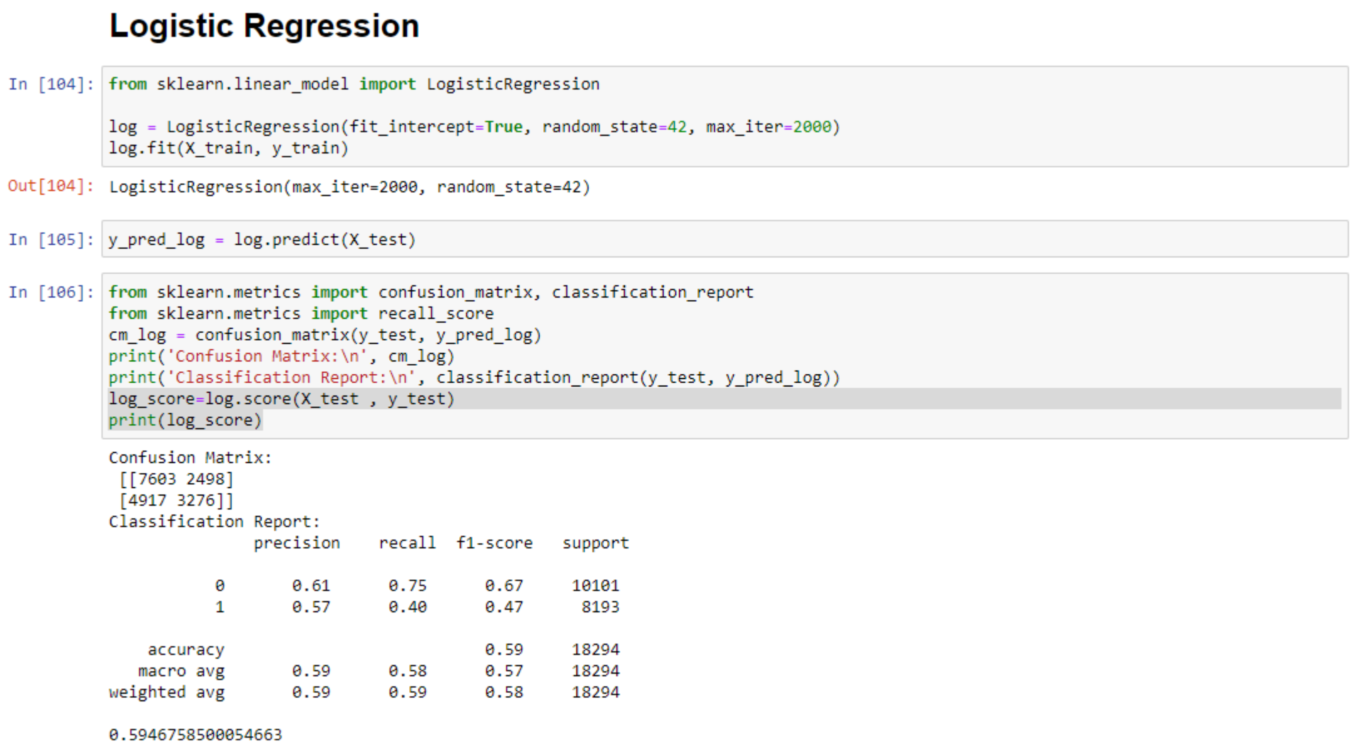


```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, Y, random_state = 500, test_size = 0.2)

# check the dimensions of the train & test subset using 'shape'
# print dimension of train set
print('X_train', X_train.shape)
print('y_train', y_train.shape)

# print dimension of test set
print('X_test', X_test.shape)
print('y_test', y_test.shape)
```

```
X_train (73173, 23)
y_train (73173,)
X_test (18294, 23)
y_test (18294,)
```

Applying Classification Models on Churn Data :

Applying Logistic Regression as it is one of the common Classification models that is robust to outliers.



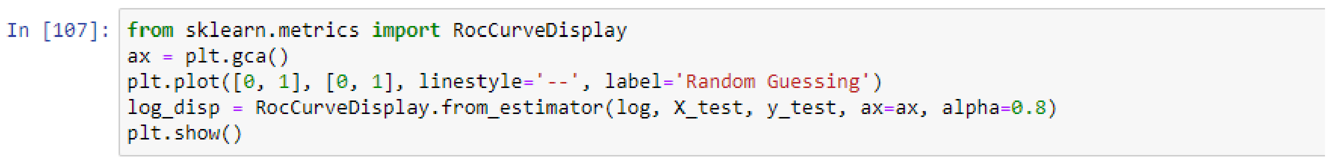
## Logistic Regression

```
In [104]: from sklearn.linear_model import LogisticRegression

          log = LogisticRegression(fit_intercept=True, random_state=42, max_iter=2000)
          log.fit(X_train, y_train)

Out[104]: LogisticRegression(max_iter=2000, random_state=42)
```

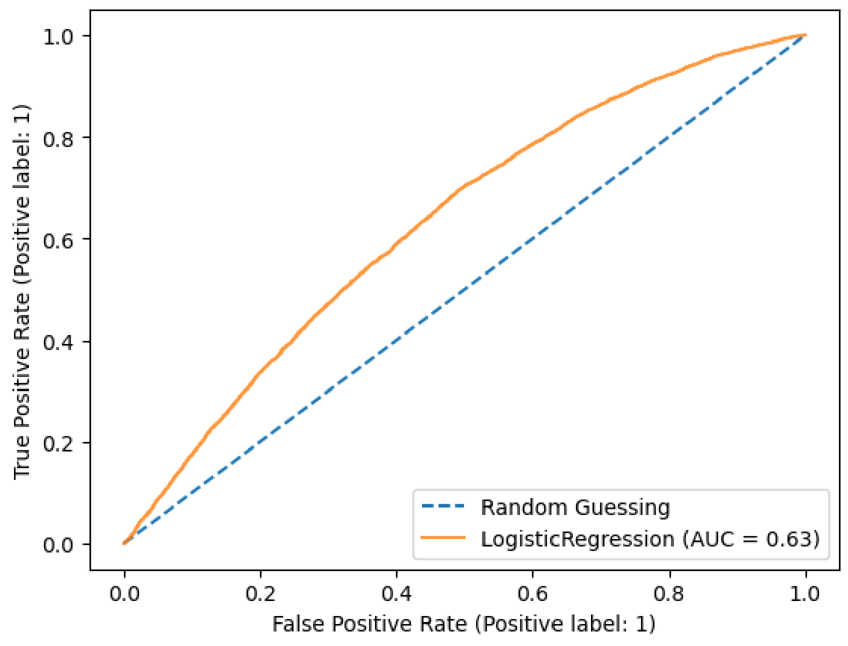
```
In [105]: y_pred_log = log.predict(X_test)
```

```
In [106]: from sklearn.metrics import confusion_matrix, classification_report
          from sklearn.metrics import recall_score
          cm_log = confusion_matrix(y_test, y_pred_log)
          print('Confusion Matrix:\n', cm_log)
          print('Classification Report:\n', classification_report(y_test, y_pred_log))
          log_score=log.score(X_test , y_test)
          print(log_score)

          Confusion Matrix:
           [[7603 2498]
           [4917 3276]]
          Classification Report:
                        precision    recall  f1-score   support

                     0       0.61      0.75      0.67     10101
                     1       0.57      0.40      0.47      8193

              accuracy                           0.59     18294
             macro avg       0.59      0.58      0.57     18294
          weighted avg       0.59      0.59      0.58     18294

          0.5946758500054663
```

ROC Curve:



```
In [107]: from sklearn.metrics import RocCurveDisplay
          ax = plt.gca()
          plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guessing')
          log_disp = RocCurveDisplay.from_estimator(log, X_test, y_test, ax=ax, alpha=0.8)
          plt.show()
```



Applying various Bagging and Boosting methods to improve accuracy:

## Random Forest

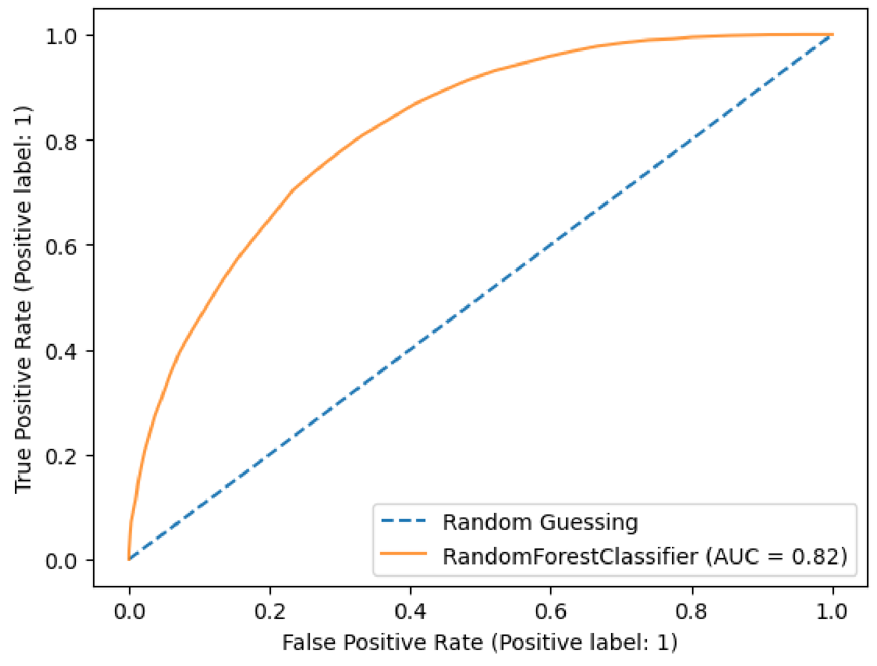
```
In [108]: from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import roc_curve, auc
          rfc = RandomForestClassifier(n_estimators=50, random_state=0)
          rfc.fit(X_train, y_train)
          y_pred_rf = rfc.predict(X_test)
```

```
In [109]: from sklearn.metrics import confusion_matrix, classification_report
          from sklearn.metrics import recall_score
          cm = confusion_matrix(y_test, y_pred_rf)
          print('Confusion Matrix:\n', cm)
          print('Classification Report:\n', classification_report(y_test, y_pred_rf))
          RFC=rfc.score(X_test , y_test)
          print(RFC)
```

```
Confusion Matrix:
 [[8009 2092]
 [2788 5405]]
Classification Report:
               precision    recall  f1-score   support

           0       0.74      0.79      0.77     10101
           1       0.72      0.66      0.69      8193

    accuracy                           0.73     18294
   macro avg       0.73      0.73      0.73     18294
weighted avg       0.73      0.73      0.73     18294

0.7332458729638133
```

ROC Curve:

```
In [110]: from sklearn.metrics import RocCurveDisplay
          ax = plt.gca()
          plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guessing')
          rfc_disp = RocCurveDisplay.from_estimator(rfc, X_test, y_test, ax=ax, alpha=0.8)
          plt.show()
```



## AdaBoost

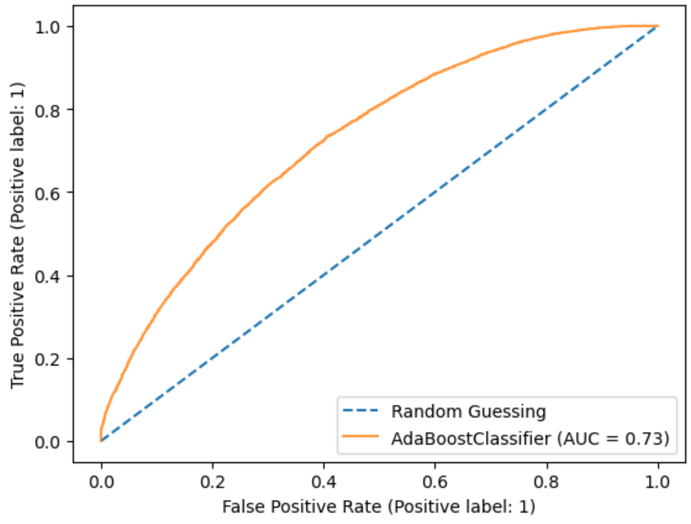
```
In [111]: from sklearn.ensemble import AdaBoostClassifier
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.metrics import accuracy_score
          abc = AdaBoostClassifier(DecisionTreeClassifier(max_depth=2), n_estimators=50,learning_rate=0.1, random_state=0)
          abc.fit(X_train, y_train)

          y_pred_abc = abc.predict(X_test)
```

```
In [112]: from sklearn.metrics import confusion_matrix, classification_report
          from sklearn.metrics import recall_score
          cm_ada = confusion_matrix(y_test, y_pred_abc)
          print('Confusion Matrix:\n', cm_ada)
          print('Classification Report:\n', classification_report(y_test, y_pred_abc))
          ada_score=log.score(X_test , y_test)
          print(ada_score)
```

```
Confusion Matrix:
 [[7729 2372]
 [3852 4341]]
Classification Report:
               precision    recall  f1-score   support

           0       0.67      0.77      0.71     10101
           1       0.65      0.53      0.58      8193

    accuracy                           0.66     18294
   macro avg       0.66      0.65      0.65     18294
weighted avg       0.66      0.66      0.65     18294

0.5946758500054663
```

ROC Curve:

```
In [113]: from sklearn.metrics import RocCurveDisplay
          ax = plt.gca()
          plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guessing')
          ada_disp = RocCurveDisplay.from_estimator(abc, X_test, y_test, ax=ax, alpha=0.8)
          plt.show()
```



## XGBoost

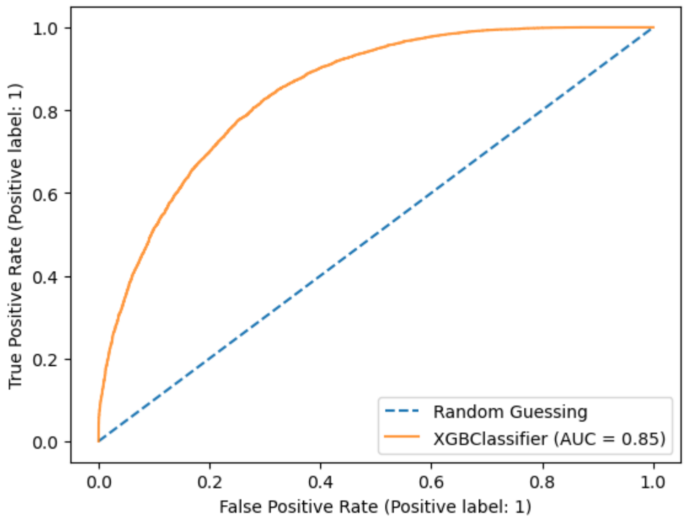
```
In [114]: from sklearn.metrics import accuracy_score
          from xgboost import XGBClassifier
          model= XGBClassifier(n_estimators = 50,random_state=0)
          model.fit(X_train,y_train)

          # Predict on the test set and calculate accuracy
          y_pred_xg = model.predict(X_test)
```

```
In [115]: from sklearn.metrics import confusion_matrix, classification_report
          from sklearn.metrics import recall_score
          cm_xg = confusion_matrix(y_test, y_pred_xg)
          print('Confusion Matrix:\n', cm_xg)
          print('Classification Report:\n', classification_report(y_test, y_pred_xg))
          xg_score=model.score(X_test , y_test)
          print(xg_score)
```

```
Confusion Matrix:
 [[7809 2292]
 [2149 6044]]
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.77      0.78     10101
           1       0.73      0.74      0.73      8193

    accuracy                           0.76     18294
   macro avg       0.75      0.76      0.75     18294
weighted avg       0.76      0.76      0.76     18294

0.75724281185088
```

ROC Curve:

```
In [116]: from sklearn.metrics import RocCurveDisplay
          ax = plt.gca()
          plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guessing')
          xg_disp = RocCurveDisplay.from_estimator(model, X_test, y_test, ax=ax, alpha=0.8)
          plt.show()
```



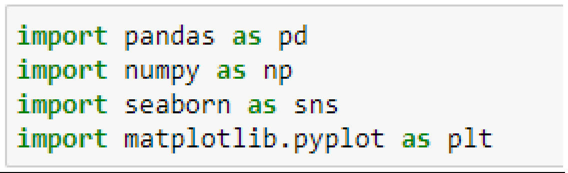
| Model | Accuracy | AUC of ROC |
|---|---|---|
| Logistic Regression | 0.5946 | 0.63 |
| Random Forest (Bagging) | 0.7332 | 0.82 |
| AdaBoost (Boosting) | 0.5946 | 0.73 |
| XGBoost (Boosting) | 0.7574 | 0.85 |

From the above table we can conclude that the XGBoost Classification Model is the Best choice as it has the highest accuracy score and the highest AUC of ROC .

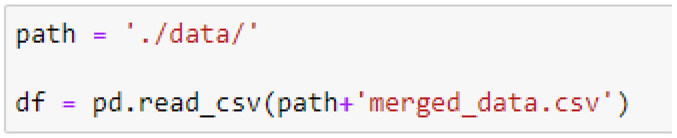
XGBoost is also Robust to outliers, highly accurate and XGBoost includes regularization techniques that help to prevent overfitting, which is could be a problem in Churn Classification Problems. It uses a combination of L1 and L2 regularization to reduce the complexity of the model, resulting in more robust and accurate predictions.

## SALES PREDICTION

Importing merged data and necessary libraries

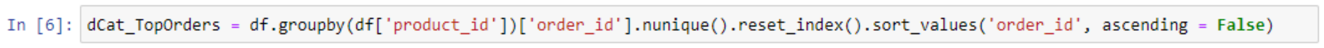


```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```



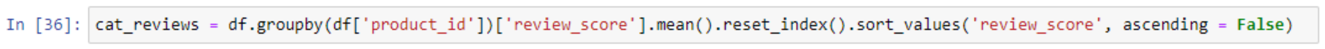
```
path = './data/'

df = pd.read_csv(path+'merged_data.csv')
```

Finding Sales amounts of Products by using groupby method

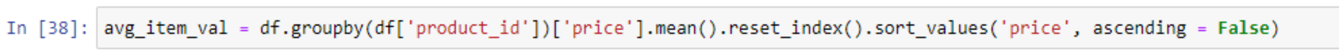


```
In [6]: dCat_TopOrders = df.groupby(df['product_id'])['order_id'].nunique().reset_index().sort_values('order_id', ascending = False)
```

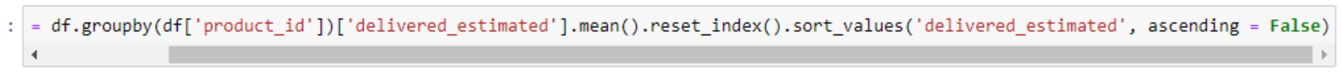
Also finding various other attributes like avg_review_score, avg_price, avegare of estimated_delivery of the product



```
In [36]: cat_reviews = df.groupby(df['product_id'])['review_score'].mean().reset_index().sort_values('review_score', ascending = False)
```

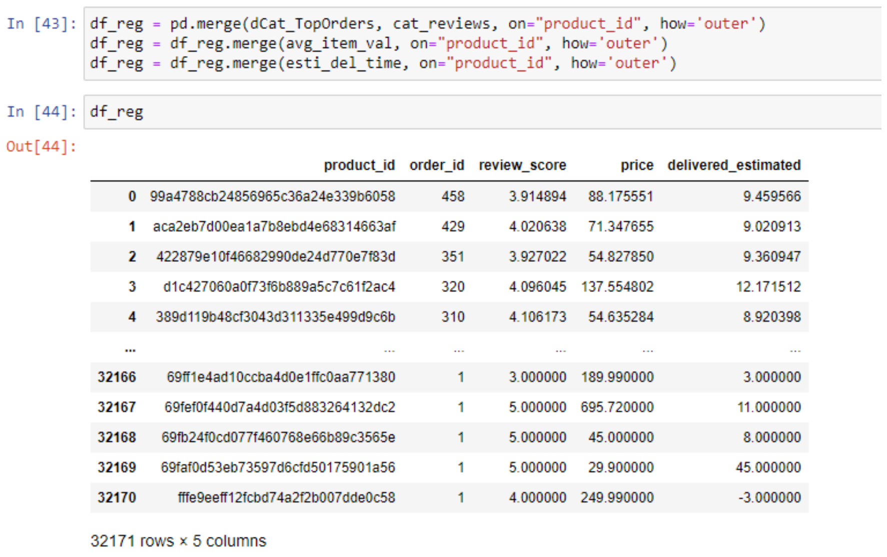


```
In [38]: avg_item_val = df.groupby(df['product_id'])['price'].mean().reset_index().sort_values('price', ascending = False)
```



```
: = df.groupby(df['product_id'])['delivered_estimated'].mean().reset_index().sort_values('delivered_estimated', ascending = False)
```

Merging all DataFrames:



```
In [43]: df_reg = pd.merge(dCat_TopOrders, cat_reviews, on="product_id", how='outer')
         df_reg = df_reg.merge(avg_item_val, on="product_id", how='outer')
         df_reg = df_reg.merge(esti_del_time, on="product_id", how='outer')

In [44]: df_reg
Out[44]:
```

| | product_id | order_id | review_score | price | delivered_estimated |
|---|---|---|---|---|---|
| 0 | 99a4788cb24856965c36a24e339b6058 | 458 | 3.914894 | 88.175551 | 9.459566 |
| 1 | aca2eb7d00ea1a7b8ebd4e68314663af | 429 | 4.020638 | 71.347655 | 9.020913 |
| 2 | 422879e10f46682990de24d770e7f83d | 351 | 3.927022 | 54.827850 | 9.360947 |
| 3 | d1c427060a0f73f6b889a5c7c61f2ac4 | 320 | 4.096045 | 137.554802 | 12.171512 |
| 4 | 389d119b48cf3043d311335e499d9c6b | 310 | 4.106173 | 54.635284 | 8.920398 |
| ... | ... | ... | ... | ... | ... |
| 32166 | 69ff1e4ad10ccba4d0e1ffc0aa771380 | 1 | 3.000000 | 189.990000 | 3.000000 |
| 32167 | 69fef0f440d7a4d03f5d883264132dc2 | 1 | 5.000000 | 695.720000 | 11.000000 |
| 32168 | 69fb24f0cd077f460768e66b89c3565e | 1 | 5.000000 | 45.000000 | 8.000000 |
| 32169 | 69faf0d53eb73597d6cfd50175901a56 | 1 | 5.000000 | 29.900000 | 45.000000 |
| 32170 | fffe9eeff12fcbd74a2f2b007dde0c58 | 1 | 4.000000 | 249.990000 | -3.000000 |

32171 rows × 5 columns

## Merging with product_id to get product_category and renaming incorrectly labelled data

```
In [46]: dict = {'order_id': 'sales_amt',
                 'review_score': 'avg_review_score',
                 'price': 'avg_price',
                 'delivered_estimated': 'estimated_delivery',
                }

         # call rename () method
         df_reg.rename(columns=dict,
                   inplace=True)
```

```
In [47]: a = df.loc[:, ['product_id','product_category']]
```

```
In [48]: sales = pd.merge(df_reg,a, on="product_id", how='right')
```

```
In [49]: sales
```

Out[49]:

| | product_id | sales_amt | avg_review_score | avg_price | estimated_delivery | product_category |
|---|---|---|---|---|---|---|
| **0** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | Furniture |
| **1** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | Furniture |
| **2** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | Furniture |
| **3** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | Furniture |
| **4** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | Furniture |
| **...** | ... | ... | ... | ... | ... | ... |
| **115604** | ea9b0b855335919945731f9368f83dc9 | 1 | 5.000000 | 193.000000 | 17.000000 | Home & Garden |
| **115605** | 0c800efe70e04ffcc3b266946e3e4826 | 1 | 4.000000 | 389.000000 | 11.000000 | Home & Garden |
| **115606** | 775596b5ab8f1cb5890c7263c1c92bc4 | 1 | 5.000000 | 139.000000 | 13.000000 | Home & Garden |
| **115607** | cc9e875c2df286dbed83efe01191162c | 1 | 5.000000 | 129.000000 | 18.000000 | Home & Garden |
| **115608** | cc9e875c2df286dbed83efe01191162c | 1 | 5.000000 | 129.000000 | 18.000000 | Home & Garden |

## Label Encoding Product_id and deleting null values

```
In [56]: from sklearn import preprocessing

         label_encoder = preprocessing.LabelEncoder()

         sales['product_category']= label_encoder.fit_transform(sales['product_category'])
```

```
In [58]: sales
```

Out[58]:

| | product_id | sales_amt | avg_review_score | avg_price | estimated_delivery | product_category |
|---|---|---|---|---|---|---|
| **0** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | 6 |
| **1** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | 6 |
| **2** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | 6 |
| **3** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | 6 |
| **4** | a9516a079e37a9c9c36b9b78b10169e8 | 40 | 3.393443 | 119.285082 | 10.245902 | 6 |
| **...** | ... | ... | ... | ... | ... | ... |
| **115604** | ea9b0b855335919945731f9368f83dc9 | 1 | 5.000000 | 193.000000 | 17.000000 | 7 |
| **115605** | 0c800efe70e04ffcc3b266946e3e4826 | 1 | 4.000000 | 389.000000 | 11.000000 | 7 |
| **115606** | 775596b5ab8f1cb5890c7263c1c92bc4 | 1 | 5.000000 | 139.000000 | 13.000000 | 7 |
| **115607** | cc9e875c2df286dbed83efe01191162c | 1 | 5.000000 | 129.000000 | 18.000000 | 7 |
| **115608** | cc9e875c2df286dbed83efe01191162c | 1 | 5.000000 | 129.000000 | 18.000000 | 7 |

115609 rows × 6 columns

```
In [59]: sales.isnull().sum()

Out[59]: product_id              0
         sales_amt               0
         avg_review_score        0
         avg_price               0
         estimated_delivery    935
         product_category        0
         dtype: int64
```

```
In [61]: sales.dropna(axis="rows",inplace=True)
```

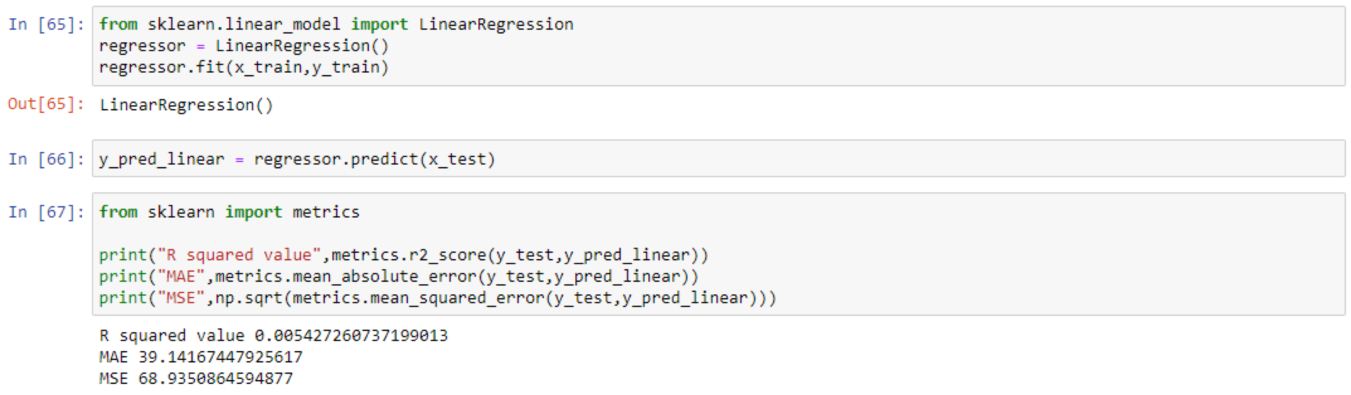
## Train Test Split

```
In [64]: from sklearn.model_selection import train_test_split

         x = sales.drop(['product_id','sales_amt'], axis='columns')
         y = sales[['sales_amt']]
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 1/3, random_state = 101)
```

## Applying Regression Models:

Applying Linear Regression as it is the simplest and most common Regression algorithms

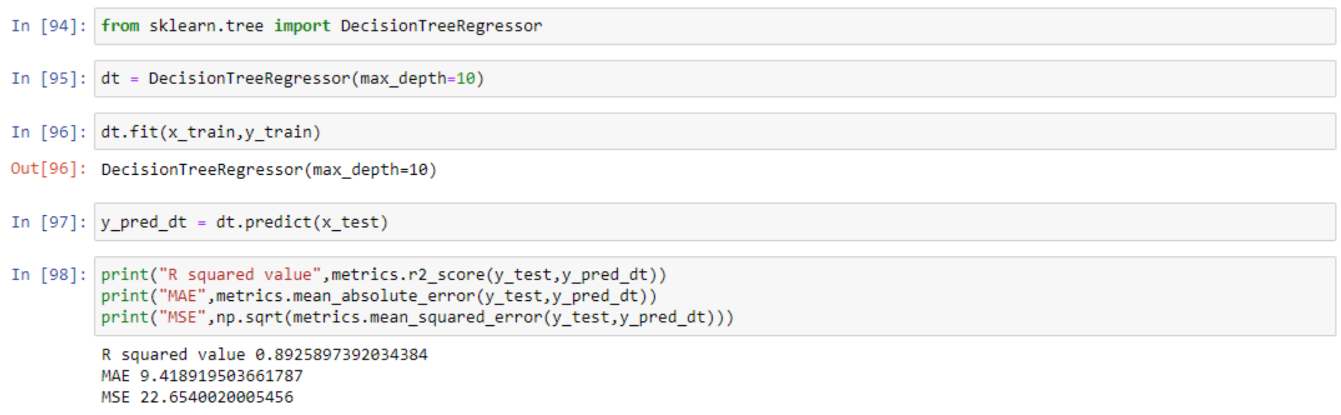
### Linear Regression



```
In [65]: from sklearn.linear_model import LinearRegression
         regressor = LinearRegression()
         regressor.fit(x_train,y_train)

Out[65]: LinearRegression()

In [66]: y_pred_linear = regressor.predict(x_test)

In [67]: from sklearn import metrics

         print("R squared value",metrics.r2_score(y_test,y_pred_linear))
         print("MAE",metrics.mean_absolute_error(y_test,y_pred_linear))
         print("MSE",np.sqrt(metrics.mean_squared_error(y_test,y_pred_linear)))

         R squared value 0.005427260737199013
         MAE 39.14167447925617
         MSE 68.9350864594877
```

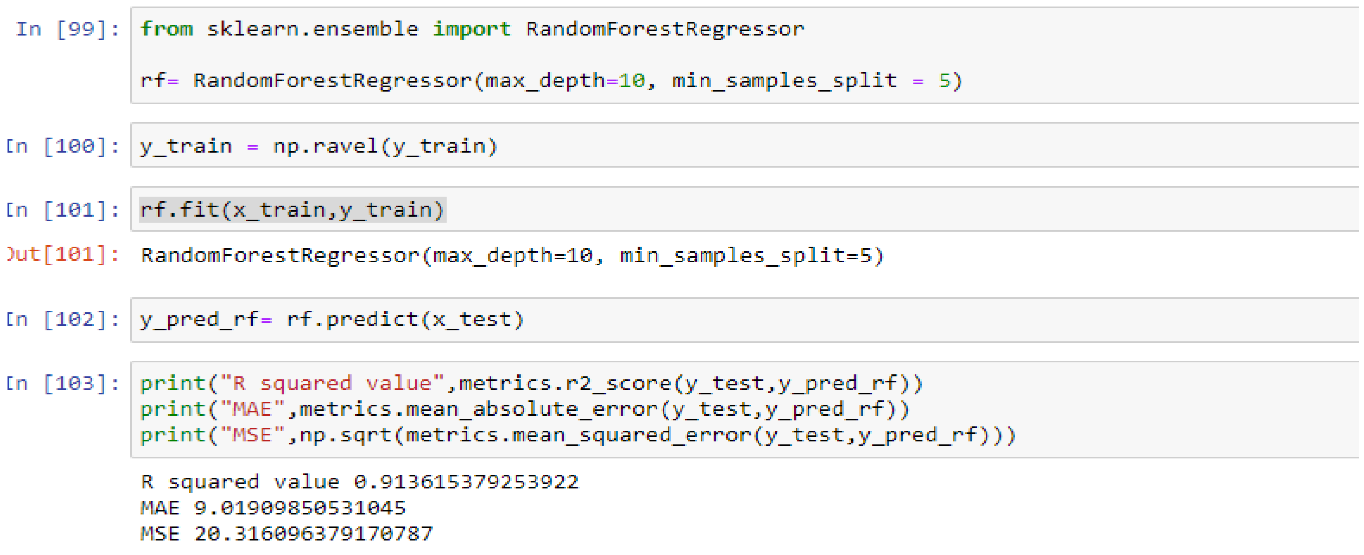
Applying Tree based Regressors as they have high explainability thus you can explain the decisions,  identify possible events that might occur, and see potential outcomes. The analysis helps you determine what the best decision would be.
High explainability can also be a huge advantage when working with models that will be making business decisions so as to better understand how and why certain decisions are made

## DecisionTree Regression



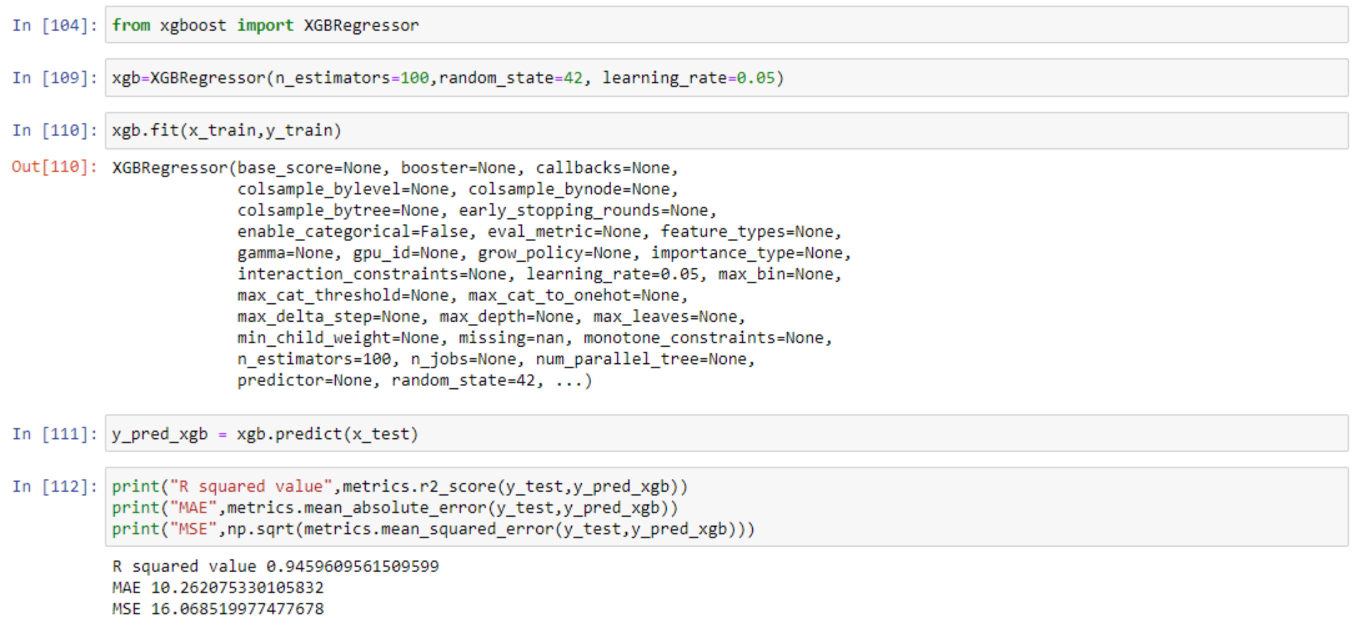
```
In [94]: from sklearn.tree import DecisionTreeRegressor

In [95]: dt = DecisionTreeRegressor(max_depth=10)

In [96]: dt.fit(x_train,y_train)

Out[96]: DecisionTreeRegressor(max_depth=10)

In [97]: y_pred_dt = dt.predict(x_test)

In [98]: print("R squared value",metrics.r2_score(y_test,y_pred_dt))
         print("MAE",metrics.mean_absolute_error(y_test,y_pred_dt))
         print("MSE",np.sqrt(metrics.mean_squared_error(y_test,y_pred_dt)))

         R squared value 0.8925897392034384
         MAE 9.418919503661787
         MSE 22.6540020005456
```

## RandomForestRegressor



```
 In [99]: from sklearn.ensemble import RandomForestRegressor

          rf= RandomForestRegressor(max_depth=10, min_samples_split = 5)

[n [100]: y_train = np.ravel(y_train)

[n [101]: rf.fit(x_train,y_train)

Out[101]: RandomForestRegressor(max_depth=10, min_samples_split=5)

[n [102]: y_pred_rf= rf.predict(x_test)

[n [103]: print("R squared value",metrics.r2_score(y_test,y_pred_rf))
          print("MAE",metrics.mean_absolute_error(y_test,y_pred_rf))
          print("MSE",np.sqrt(metrics.mean_squared_error(y_test,y_pred_rf)))

          R squared value 0.913615379253922
          MAE 9.01909850531045
          MSE 20.316096379170787
```

## XGBoostRegressor



```
In [104]: from xgboost import XGBRegressor

In [109]: xgb=XGBRegressor(n_estimators=100,random_state=42, learning_rate=0.05)

In [110]: xgb.fit(x_train,y_train)

Out[110]: XGBRegressor(base_score=None, booster=None, callbacks=None,
                       colsample_bylevel=None, colsample_bynode=None,
                       colsample_bytree=None, early_stopping_rounds=None,
                       enable_categorical=False, eval_metric=None, feature_types=None,
                       gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                       interaction_constraints=None, learning_rate=0.05, max_bin=None,
                       max_cat_threshold=None, max_cat_to_onehot=None,
                       max_delta_step=None, max_depth=None, max_leaves=None,
                       min_child_weight=None, missing=nan, monotone_constraints=None,
                       n_estimators=100, n_jobs=None, num_parallel_tree=None,
                       predictor=None, random_state=42, ...)

In [111]: y_pred_xgb = xgb.predict(x_test)

In [112]: print("R squared value",metrics.r2_score(y_test,y_pred_xgb))
          print("MAE",metrics.mean_absolute_error(y_test,y_pred_xgb))
          print("MSE",np.sqrt(metrics.mean_squared_error(y_test,y_pred_xgb)))

          R squared value 0.9459609561509599
          MAE 10.262075330105832
          MSE 16.068519977477678
```

| Model | R-squared value | MAE | MSE |
|---|---|---|---|
| Linear Regression | 0.00542 | 39.14 | 68.93 |
| DecisionTree Regression | 0.8925 | 9.41 | 20.31 |
| RandomForestRegressor | 0.9136 | 9.02 | 20.32 |
| XGBoostRegressor | 0.9459 | 10.26 | 16.06 |

From the Above table we can observe that the XGBoost Regressor has the highest value of R-sqauare score and has the lowest value of Mean Absolute Error and Mean Squared Error

This is possibly because, XGBoost is a Boosting method that uses several weak learners and improves on the weak learners in every iteration to improve its results.