Researched Notebooks

All notebooks are listed below:

Table of Contents

[1. Perform Market basket analysis with E-comm data 1](#_Toc48820817)

[1.1. Data Preprocessing 1](#_Toc48820818)

[1.2. Recommend similar products using Apriori 2](#_Toc48820819)

[3. Delivery Days, Payments, Customers Analysis 3](#_Toc48820820)

[4. Olist Data Pre Processing 4](#_Toc48820821)

[5. E-Commerce predicting customer lifetime value 6](#_Toc48820822)

# Perform Market basket analysis with E-comm data

<https://www.kaggle.com/sasha18/perform-market-basket-analysis-with-e-comm-data>

Here we attempt to group common patterns of items together using **Association rules** using **Apriori** algorithm.

**Data:** brazilian-ecommerce/olist\_order\_items\_dataset.csv

* Using only half data due to memory issues
  + data = data.head(30000)
* Append Quantity column. Since, there's 1 product on every row we can easily use Quantity as 1 which will be added subsequently.
  + data.insert(7, 'quantity',1)
  + data.shape: (30000,7) -> (30000, 8)

We can't use this data the way it is, we got to further convert this into a format that is accepted by Apriori algorithm.

## Data Preprocessing

**Things to do:**

* Total no. of unique orders and products
* Whether to drop products that are purchased within a given threshold
* Average no. of products purchased in a given order
* Drop Some products that were purchased below 10 times

item\_freq = data['product\_id'].value\_counts()

data = data[data.isin(item\_freq.index[item\_freq >= 10]).values]

data['product\_id'].value\_counts()

## Recommend similar products using Apriori

**Things to do:**

* Create a basket of all products, orders and quantity
* Perform One hot encoding so as to convert all quantities into format suitable for apriori algorithm
* Build list of frequent itemsets
* Generate rules based on your requirements (conf>0.5)
* Create a basket of all products, orders and quantity

basket = (data.groupby(['order\_id','product\_id'])['quantity']).sum().unstack().reset\_index().fillna(0).set\_index('order\_id')

* Convert 0.0 to 0, convert the units to One hot encoded values

def encode\_units(x):

if x<= 0:

return 0

if x>=1:

return 1

* Building Frequent Itemsets

frequent\_itemsets = apriori(basket\_sets, min\_support = 0.0001, use\_colnames = True)

frequent\_itemsets['length'] = frequent\_itemsets['itemsets'].apply(lambda x: len(x))

* Create Rules

rules = association\_rules(frequent\_itemsets, metric = 'support', min\_threshold = 0.0001)

* Products having 50% confidence likely to be purchased together

rules[rules['confidence'] >= 0.50]

**Sample Result:**

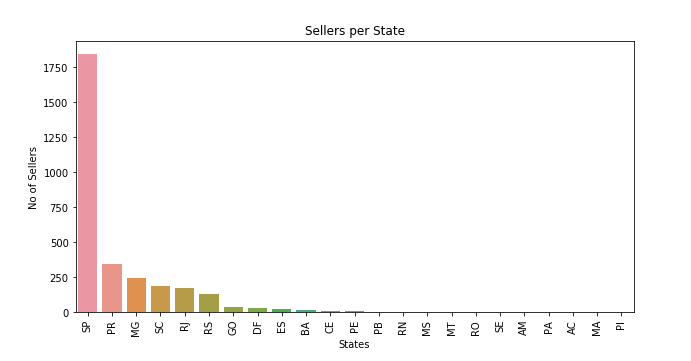
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **İndex** | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **Confidence** | **lift** | **leverage** | **conviction** |
| **87** | **(368c6c730842d78016ad823897a372db, 0bcc3eeca39...** | **(53759a2ecddad2bb87a079a1f1519f73)** | **0.000166** | **0.012757** | **0.000166** | **1** | **78.38961** | **0.000164** | **inf** |

# Delivery Days, Payments, Customers Analysis

<https://www.kaggle.com/dhruvgupta2801/delivery-days-payments-customers-analysis>

* Graph for sellers per state

sns.barplot(x=df1['seller\_state'].value\_counts().index,y=df1['seller\_state'].value\_counts().values)



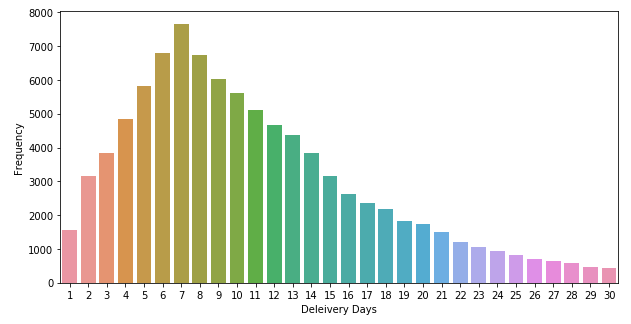
* Delivery Time Frequency Graphic

X=pd.to\_datetime(df2['order\_delivered\_customer\_date'])-pd.to\_datetime(df2['order\_purchase\_timestamp'])

for i **in** range(0,len(X)):

X[i]=X[i].days

sns.barplot(x=X.value\_counts().sort\_values(ascending=False).head(30).index,y=X.value\_counts().sort\_values(ascending=False).head(30).values)



* Amount Paid by Payment Type

sum1=[]

group=[]

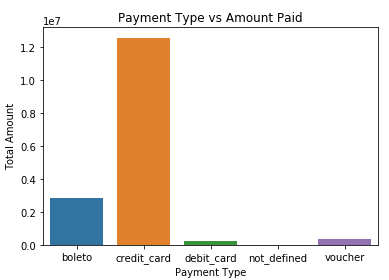
for groups,frame **in** df5.groupby('payment\_type'):

sum1.append(sum(frame['payment\_value']))

group.append(groups)

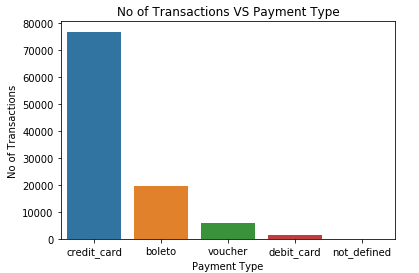
plt.figure()

sns.barplot(x=group,y=sum1)



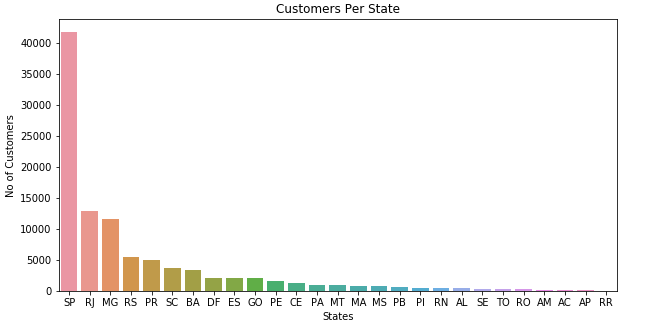
* Number of Transaction by Payment Type

sns.barplot(x=df5['payment\_type'].value\_counts().index,y=df5['payment\_type'].value\_counts().values)



* Customers Per State

sns.barplot(x=df6['customer\_state'].value\_counts().index,y=df6['customer\_state'].value\_counts().values)

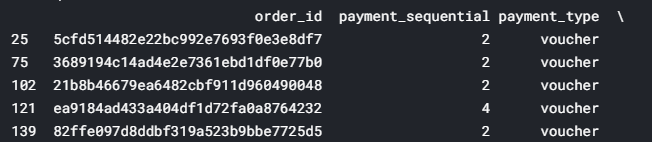


# Olist Data Pre Processing

<https://www.kaggle.com/samuraijack013/olist-data-pre-processing>

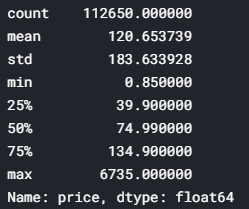
* Getting one order id where payment was made through more than one method

print(olist\_payment[olist\_payment['payment\_sequential'] > 1].head())



* Distribution of price

print(olist\_item\_order.price.describe())



* Merging Tables

*# merge order and customer demo data*

df\_process\_v1 = olist\_order.merge(olist\_customer,on = 'customer\_id', how = 'inner')

*# merge order item information with order information, this will bring data at order\_id - item\_id level*

df\_process\_v2 = olist\_item\_order.merge(df\_process\_v1,on = 'order\_id',how = 'left')

*# merge product data with above trasnaction data*

*#check shape*

print(df\_process\_v2.shape)

df\_process\_v2 = df\_process\_v2.merge(olist\_products,on = 'product\_id',how = 'inner')

* Filling the missing values

df\_transaction\_v1['product\_height\_cm'].fillna(df\_transaction\_v1['product\_height\_cm'].mean(),inplace = True)

df\_transaction\_v1['product\_weight\_g'].fillna(df\_transaction\_v1['product\_weight\_g'].mean(),inplace = True)

df\_transaction\_v1['product\_width\_cm'].fillna(df\_transaction\_v1['product\_width\_cm'].mean(),inplace = True)

df\_transaction\_v1['product\_length\_cm'].fillna(df\_transaction\_v1['product\_length\_cm'].mean(),inplace = True)

df\_transaction\_v1['product\_height\_cm'].fillna(df\_transaction\_v1['product\_height\_cm'].mean(),inplace = True)

* Bring the payment info

*#filtering out payment type as voucher separately and rest as others*

olist\_payment.loc[olist\_payment.payment\_type.isin(['voucher']),'payment\_type\_new'] = 'voucher'

olist\_payment['payment\_type\_new'].fillna('other',inplace = True)

olist\_payment\_grp = olist\_payment.groupby(['order\_id','payment\_type\_new']).agg({'payment\_value':'sum'}).reset\_index()

*# pivot up data to get voucher, non voucher revenue corresponding to each order\_id*

olist\_payment\_grp\_pvt = olist\_payment\_grp.pivot(index = 'order\_id',columns='payment\_type\_new',values = 'payment\_value').reset\_index()

olist\_payment\_grp\_pvt.fillna(0,inplace = True)

*#merge this information with transactional data*

df\_transaction\_v2 = df\_transaction\_v1.merge(olist\_payment\_grp\_pvt[['order\_id','other','voucher']],on = 'order\_id', how = 'left')

* Creating Campaign Data

# E-Commerce predicting customer lifetime value

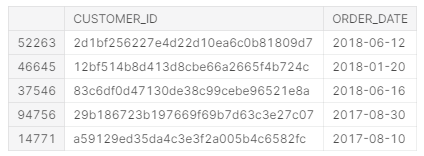
<https://www.kaggle.com/dhimananubhav/e-commerce-predicting-customer-lifetime-value>

Objectives

The primary goal of this work is to build a probabilistic model for forecasting customer lifetime value in non-contractual setting on an individual level.

Using the results of this exercise, managers should be able to:

* Distinguish active customers from inactive customers.
* Generate transaction forecasts for individual customers.
* Predict the purchase volume of the entire customer base.
* Creating new data named elog



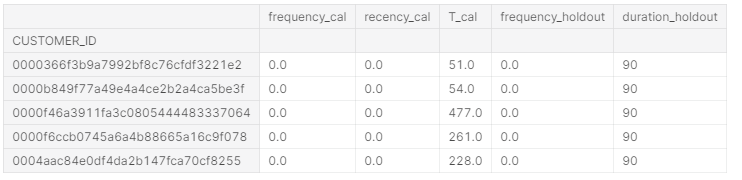
What is RFM Matrix:

**Recency:** when your customer last bought a product from you

**Frequency:** how many times your customers have bought from you

**Monetary value:**  monetary value, total purchase

* Creating RFM matrix based on transaction logs
  + Spliting calibration and holdout period

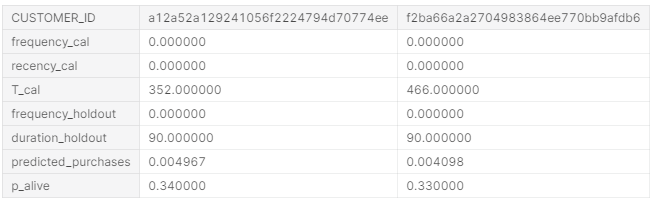
from lifetimes.utils import calibration\_and\_holdout\_data

* Training model - MBG/NBD

from lifetimes import ModifiedBetaGeoFitter

* Estimating customer lifetime value using the Gamma-Gamma model
* Predictions for each customer

t = 90 # days to predict in the future



* Model Evaluating

