Researched Notebooks

All notebooks are listed below:

Table of Contents

1. 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.

1.1. 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()
```

1.2. 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:

index	antecedents	consequents	antecedent support	consequent support	support	Confidence	lift	leverage	conviction

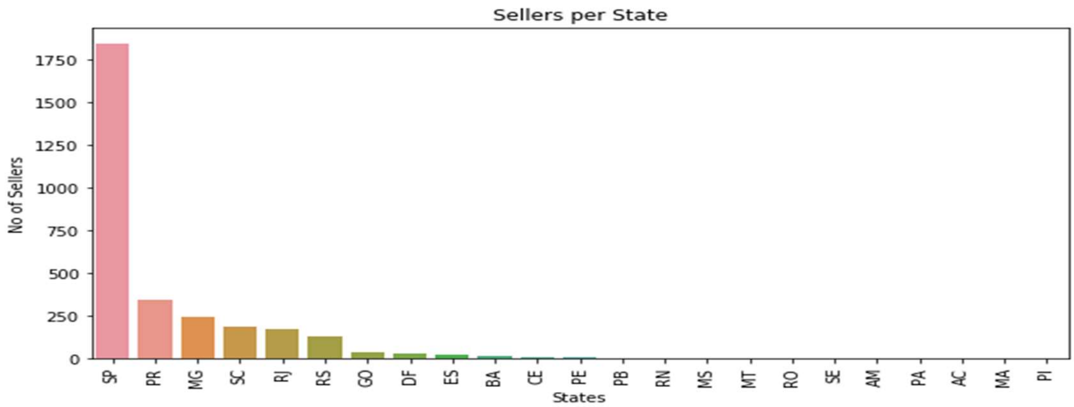
87	(368c6c730842d78016ad823897a372db, 0bcc3eeca39...	(53759a2ecddad2bb87a079a1f1519f73)	0.000166	0.012757	0.000166	1	78.38961	0.000164	inf

3. Delivery Days, Payments, Customers Analysis

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

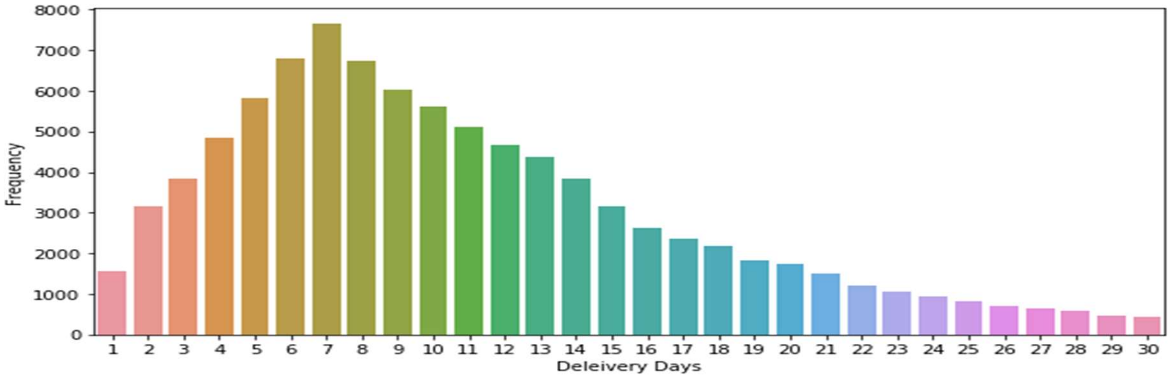
- Graph for sellers per state

```
sns.barplot(x=df1['seller_state'].value_counts().index,y=df1['seller_state'
].value_counts().values)
```



- Delivery Time Frequency Graphic

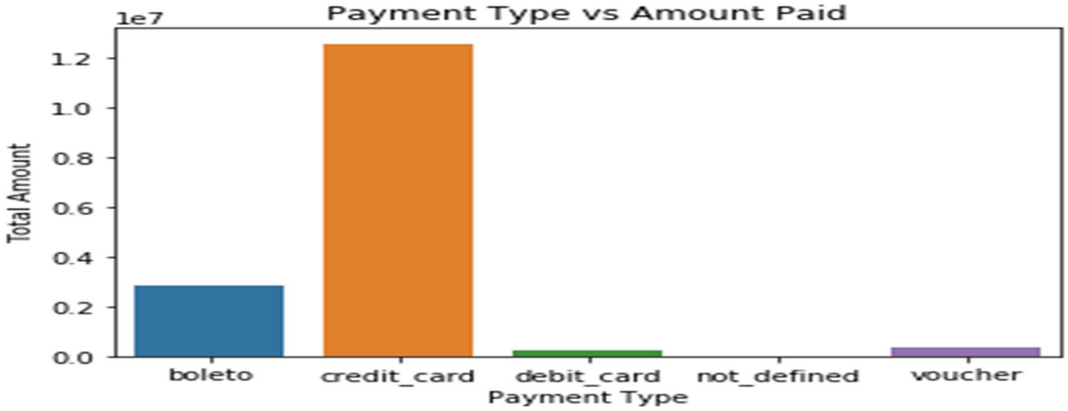
```
X=pd.to_datetime(df2['order_delivered_customer_date'])-pd.to_datetime(df2['
order_purchase_timestamp'])
for i in range(0,len(X)):
    X[i]=X[i].days
sns.barplot(x=X.value_counts().sort_values(ascending=False).head(30).index,
y=X.value_counts().sort_values(ascending=False).head(30).values)
```



- Amount Paid by Payment Type

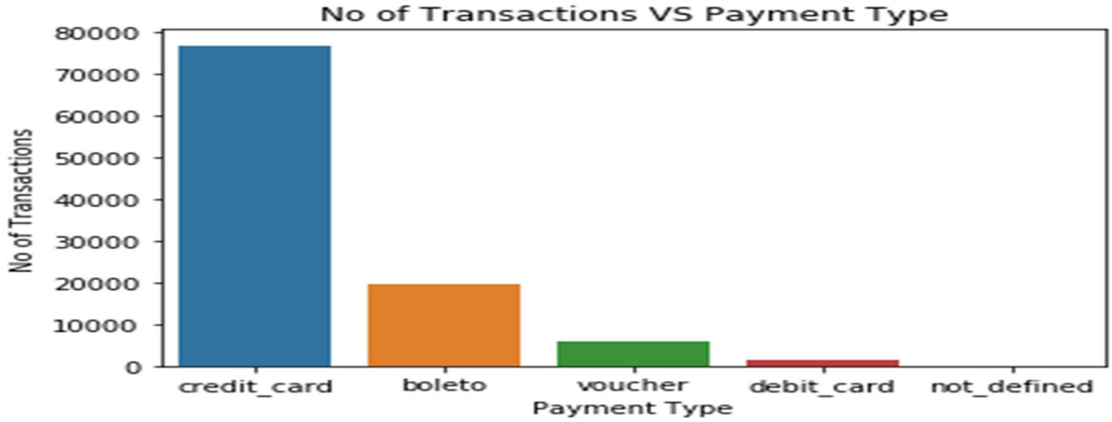
```
sum1=[]
group=[]
```

```python
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)
```



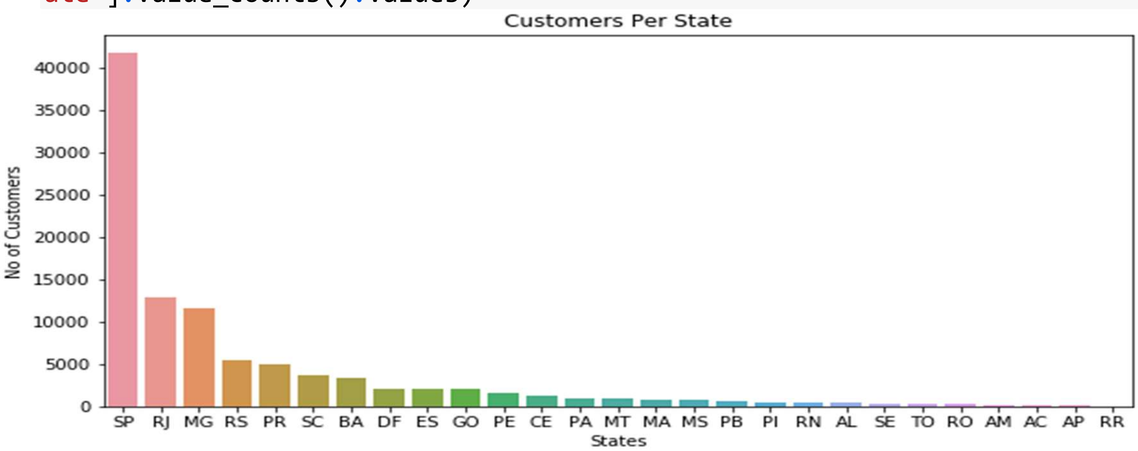
➢ Number of Transaction by Payment Type

```python
sns.barplot(x=df5['payment_type'].value_counts().index,y=df5['payment_type'
].value_counts().values)
```



➢ Customers Per State

```python
sns.barplot(x=df6['customer_state'].value_counts().index,y=df6['customer_st
ate'].value_counts().values)
```



4. 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())
```

```
                              order_id  payment_sequential payment_type  \
25   5cfd514482e22bc992e7693f0e3e8df7                   2      voucher
75   3689194c14ad4e2e7361ebd1df0e77b0                   2      voucher
102  21b8b46679ea6482cbf911d960490048                   2      voucher
121  ea9184ad433a404df1d72fa0a8764232                   4      voucher
139  82ffe097d8ddbf319a523b9bbe7725d5                   2      voucher
```

- Distribution of price

```
print(olist_item_order.price.describe())
```

```
count    112650.000000
mean        120.653739
std         183.633928
min           0.850000
25%          39.900000
50%          74.990000
75%         134.900000
max        6735.000000
Name: price, dtype: float64
```

- Merging Tables

```
# merge order and customer demo data
df_process_v1 = olist_order.merge(olist_customer,on = 'customer_id', how = 'in
ner')

# 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 = 'le
ft')

# 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 = 'in
ner')
```

➢ Filling the missing values

```
df_transaction_v1['product_height_cm'].fillna(df_transaction_v1['product_heigh
t_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_lengt
h_cm'].mean(),inplace = True)
df_transaction_v1['product_height_cm'].fillna(df_transaction_v1['product_heigh
t_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_n
ew'] = '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 orde
r_id
olist_payment_grp_pvt = olist_payment_grp.pivot(index = 'order_id',columns='pa
yment_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

5. 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

	CUSTOMER_ID	ORDER_DATE
52263	2d1bf256227e4d22d10ea6c0b81809d7	2018-06-12
46645	12bf514b8d413d8cbe66a2665f4b724c	2018-01-20
37546	83c6df0d47130de38c99cebe96521e8a	2018-06-16
94756	29b186723b197669f69b7d63c3e27c07	2017-08-30
14771	a59129ed35da4c3e3f2a005b4c6582fc	2017-08-10

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
```

	frequency_cal	recency_cal	T_cal	frequency_holdout	duration_holdout
CUSTOMER_ID					
0000366f3b9a7992bf8c76cfdf3221e2	0.0	0.0	51.0	0.0	90
0000b849f77a49e4a4ce2b2a4ca5be3f	0.0	0.0	54.0	0.0	90
0000f46a3911fa3c0805444483337064	0.0	0.0	477.0	0.0	90
0000f6ccb0745a6a4b88665a16c9f078	0.0	0.0	261.0	0.0	90
0004aac84e0df4da2b147fca70cf8255	0.0	0.0	228.0	0.0	90

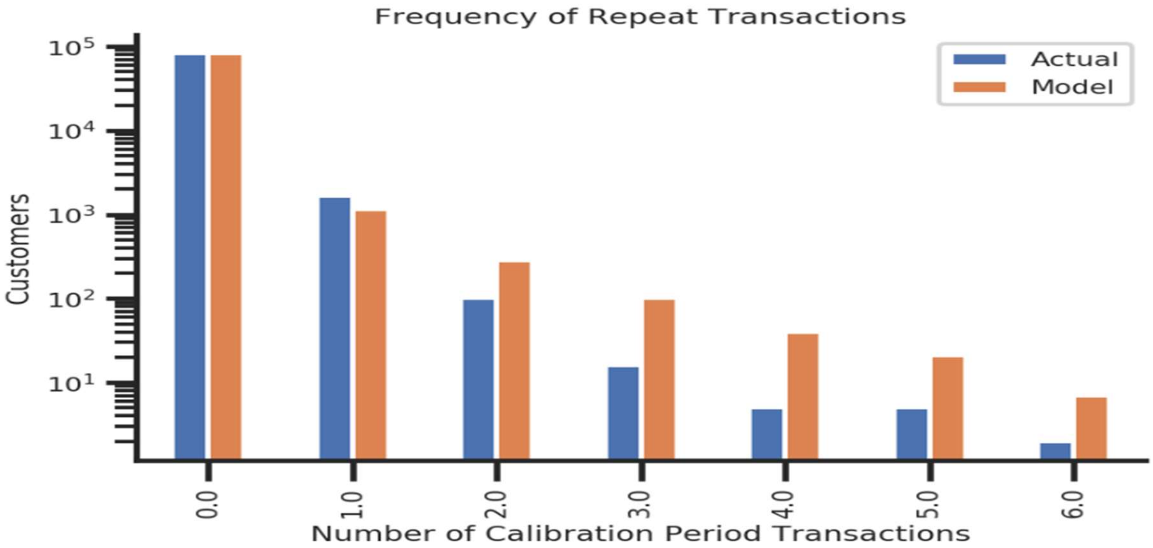
- Training model - MBG/NBD

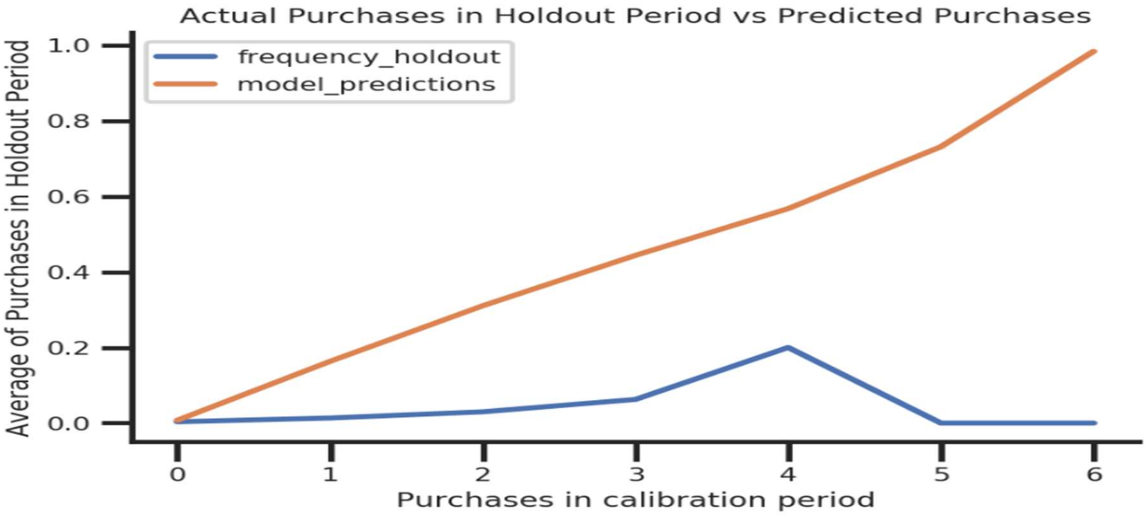
```
from lifetimes import ModifiedBetaGeoFitter
```

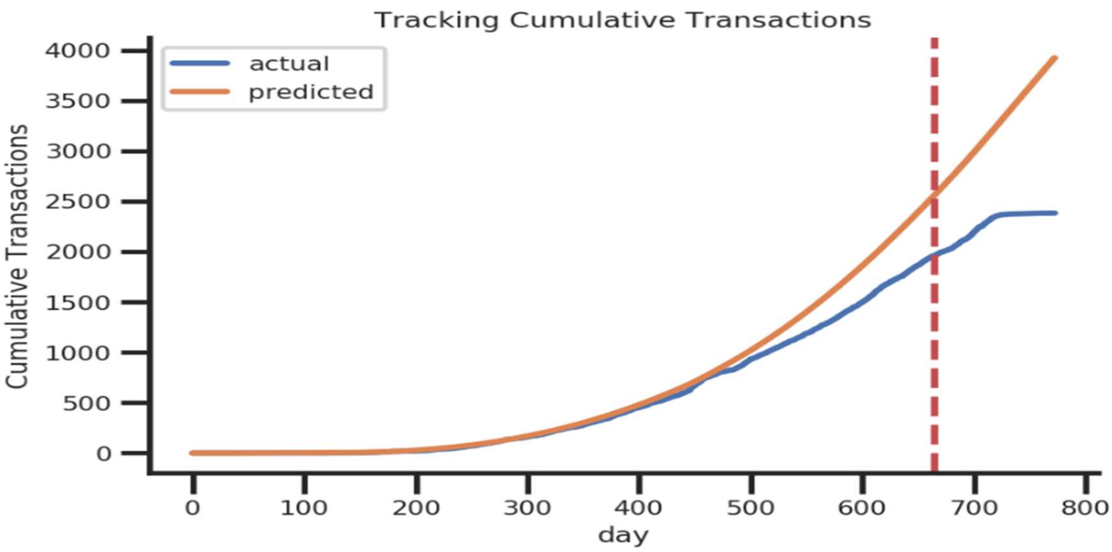
- Estimating customer lifetime value using the Gamma-Gamma model
- Predictions for each customer
  t = 90 # days to predict in the future

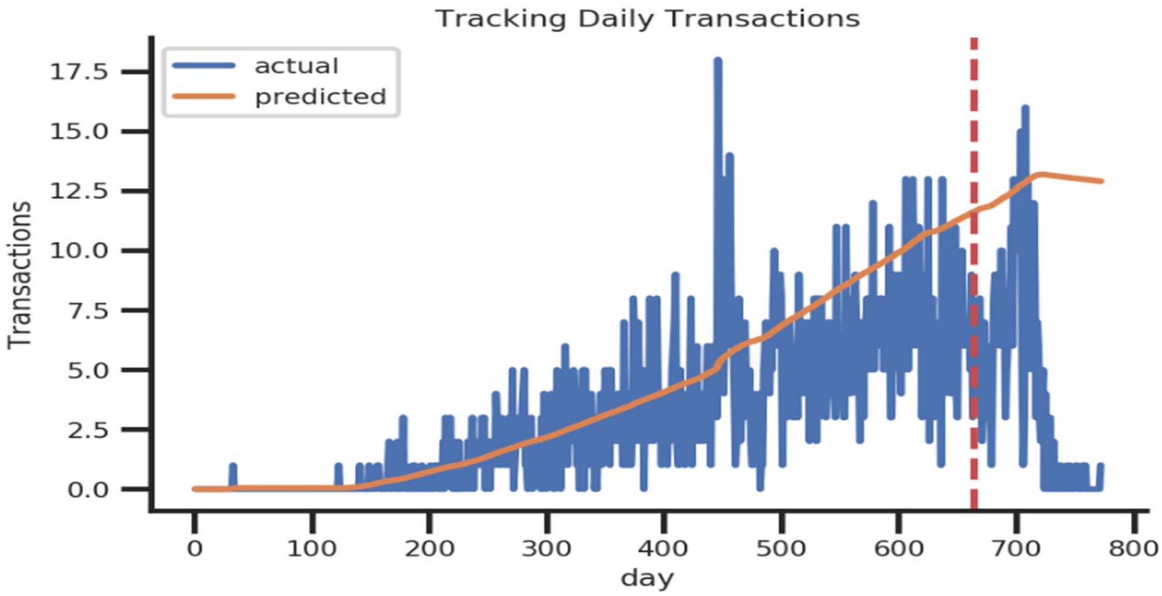
CUSTOMER_ID	a12a52a129241056f2224794d70774ee	f2ba66a2a2704983864ee770bb9afdb6
frequency_cal	0.000000	0.000000
recency_cal	0.000000	0.000000
T_cal	352.000000	466.000000
frequency_holdout	0.000000	0.000000
duration_holdout	90.000000	90.000000
predicted_purchases	0.004967	0.004098
p_alive	0.340000	0.330000

- Model Evaluating


Frequency of Repeat Transactions
Actual
Model
Customers
10^5
10^4
10^3
10^2
10^1
0.0
1.0
2.0
3.0
4.0
5.0
6.0
Number of Calibration Period Transactions


Actual Purchases in Holdout Period vs Predicted Purchases
frequency_holdout
model_predictions
Average of Purchases in Holdout Period
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
Purchases in calibration period


Tracking Cumulative Transactions
actual
predicted
Cumulative Transactions
4000
3500
3000
2500
2000
1500
1000
500
0
100
200
300
400
500
600
700
800
day



Tracking Daily Transactions
actual
predicted
Transactions
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
100
200
300
400
500
600
700
800
day