Marketing Analysis

By

Sheral Simon Waskar	20BCE1182
Anjali Jain	20BCE1320
Rahul Sunov	20BCE1424

A project report submitted to

DR. PRIYADARSHINI R
ASSISTANT PROFESSOR
SCHOOL OF COMPUTER SCIENCE ENGINEERING

in partial fulfilment of the requirements for the course of

CSE3505 – Foundations of Data Analytics

in

B.Tech. COMPUTER SCIENCE AND ENGINEERING

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report entitled "**MARKETING ANALYSIS**" is a bonafide work of **SHERAL SIMON WASKAR - 20BCE1182, ANJALI JAIN - 20BCE1320, RAHUL SUNOV - 20BCE1424** who carried out the Project work under my supervision and guidance for **CSE 3505 - FOUNDATIONS OF DATA ANALYTICS.**

Dr. PRIYADARSHINI R

Assistant Professor

School of Computer Science Engineering (SCOPE),

VIT University, Chennai

Chennai – 600 127.

ACKNOWLEDGMENT

We would like to express my deepest gratitude to Dr. Priyadarshini R for giving us this opportunity for learning and professional development. We concerned so many wonderful people and professionals who led us through this Project period.

We would also like to extend my gratitude towards our classmates for their co-operation and for sharing their knowledge with us.

I am also obliged to VIT University for giving us this opportunity that we perceive as a big milestone in my career development. We will strive to use gained skills and knowledge in the best possible way. we would also like to thank our friends and family members for their constant support and encouragement without which this Project would not have been possible.

Sheral Simon Waskar

(20BCE1182)

Anjali Jain

(20BCE1320)

Rahul Sunov

(20BCE1424)

TABLE OF CONTENTS

ABSTRACT

Market Basket Analysis is a technique for identifying relationships and identifying instances between products purchased jointly by couples. A scene event is when two or more things happen. Market basket analysis creates if-then rules, for example, scenarios for whether items a and b are likely to be purchased. Purchased item rules are probable in nature, that is, available from incidents. An interesting observation is that frequency is the percentage of baskets that are used. Rules, prices, product placements and different variants of cross-selling strategies. For simplicity, think of a market research basket in a supermarket. Market Basket analytics collects transaction data, including the inventory of all items purchased by a customer.This technique determines the relationship a product purchases to another product. This relationship is used to build if-then items purchased. Shopping basket analysis, also known as rule association or affinity analysis, is a data processing technique that can be used in various fields such as education, marketing, computer science, and science. The goal is to enable retailers to recognize buyer behaviour. This allows retailers to be aware of buyer behaviour and allows tailors to make the right decisions. In this project we have used 3 million grocery orders from more than 200,000 Instacart users' data for analysing and predicting which previously purchased item will be in the user's next order. We have employed apriori algorithm to forecast the sales of the product. We used ANN algorithm and XGBoost model to obtain the best accurate results.

INTRODUCTION

Instacart is an American technology that provides same-day grocery delivery and pickup services in the United States and Canada. Customers purchase goods from a variety of retailer partners using the Instacart mobile app or Instacart.com. A personal shopper from Instacart shops and delivers the order. Our goals are to examine the 3 million grocery orders made by more than 200,000 Instacart users that have been anonymized and publicly shared by the company, discover hidden associations between products for better cross-selling and upselling, perform customer segmentation for targeted marketing, anticipate customer behaviour, and develop a machine learning model to predict which previously purchased product will be in a user's next order.

LITERATURE REVIEW

Comparative Analysis Of Apriori And Fp-Growth Algorithms For Frequent Pattern Mining Using Apache Spark

FP-Growth technique generates a tree dynamically and uses pattern fragment growth to unearth the frequent itemsets from the dataset. It is many times faster and less resource consuming than apriori algorithm. The paper has employed Istacart open-source data available on the Kaggle website, having approximately 3 millions transactions. FP-Growth algorithm was developed for big data platforms. Apriori algorithm generates candidate itemsets and tests if they are frequent. This paper analyzes the performance of both algorithms on the basis of execution time, accuracy, number of scans, and cost of computation. FP-Growth and apriori have been used in several studies for miscellaneous case studies. This study performed comparison of both algorithms using PySpark. The data is about Instacart market (of size 197mb) having 32,434,489 rows. In they experiments they found that Frequent Pattern Growth is exponentially faster than Apriori method when deployed on a windows machine using Pyspark. They performed the testing using pySpark and a windows server. On a subset of complete dataset, the time taken by both algorithms is illustrated in Fig 2 and Fig 3. FP-Growth outperforms Apriori Algorithm when using 'Spark'. They can see that the time taken to process a greater number of rows increase. But in comparison, the execution time and complexity of the data are almost negligible. The main reason for this is that FP-G Growth scans the whole database only twice.

Market basket analysis of heterogeneous data sources for recommendation system improvement

Increase the quality of the recommendation system is possible when analyzing a larger amount of data, which can be obtained from external heterogeneous sources. Within one market area, a range of offered products may be similar, while the characteristics or associative rules formed for them may differ. Using of MBA for improving ways of arranging products on store shelves has been identified. Analysis of the most frequent customers' transactions was used to create association rules (AR). AR is used to identify relations in big datasets of transactions. There are several algorithms, including the most common "Apriori algorithm". In retail, the analysis of purchases is a crucial part of understanding the customer's behaviour. The data on purchases allow shops to adjust promotions and store individual preferences. In this study, they are trying to improve the quality of built recommendation system by using external heterogeneous data sources. Market Basket Analysis or MBA is a field of modelling techniques based upon the theory that if you buy a certain group of items, you are more (or less) likely to buy another group. MBA is applied not only in retail but also in biology, chemistry and ecology. Recommendation system (RS) is an AI algorithm that filter information about customers' behaviour and suggests additional products to them. It is based on a variety of factors such as past purchases, demographic info, their search history and etc. There are three main approaches: collaborative filtering, content-based filtering and hybrid recommendation systems. Recommendation systems (RS) are based on such MBA methods as association rules and collaborative filtering. The strength of rules is calculating for this and consists of three measures [16]: support, confidence and lift. This study proposes a scheme to improve the quality of constructed RS by using external data sources. In the first step, they construct a new recommendation system (RS) based on the union of original and/or pre-integrated datasets and then evaluate its performance. They require a prior model of customers' behaviour that determines the general behaviour of all possible customers.

Migrating a Recommendation System to Cloud Using ML Workflow

Inference is the stage of the machine learning workflow where predictions or inferences are made using data from the actual world using a trained model. A recommendation system enhances the customer experience by presenting the most pertinent goods based on past customer behaviour. The cloud or on-premises deployment of machine learning models created for recommendation systems allows for batch or real-time inference. A recommendation system should respect service level agreements while being cost-effective (SLAs). In this paper, they discuss the on-premise deployment of they iPrescribe recommendation engine. They demonstrate a strategy for moving an on-premises recommendation system installation to the cloud utilising ML workflow. They also provide they research on the effectiveness of the recommendation system model when used with various virtual instances.

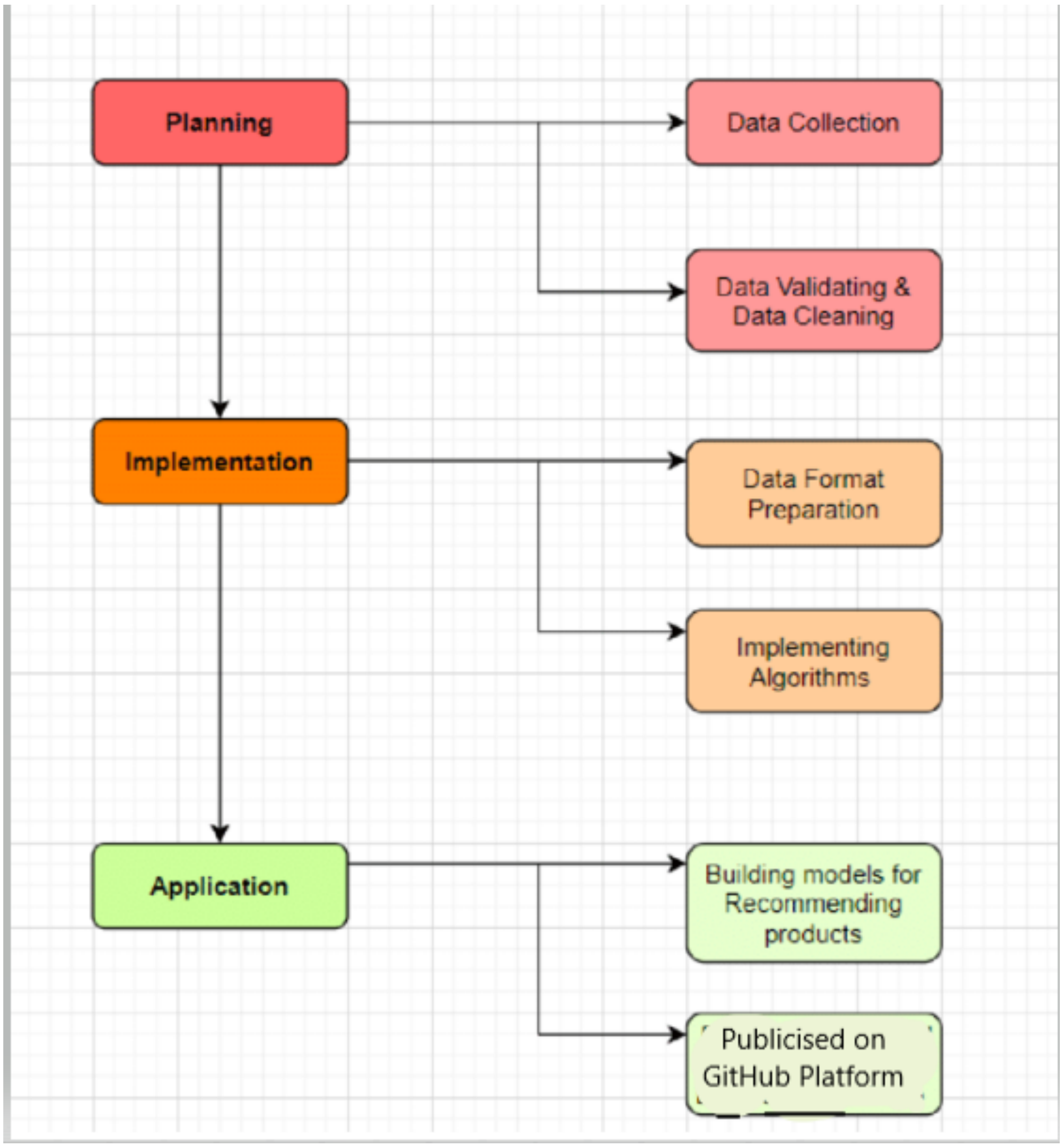
Market Basket Analysis Using Fp Growth And Apriori Algorithm: A Case Study Of Mumbai Retail Store

Companies today have access to enormous volumes of data, but the insight derived from that data is lacking. Although the idea of data mining is still novel and evolving, businesses across a range of industries are relying on it to make strategic decisions since big data is regarded as a valuable resource. The methods that filter through stored information can now disclose facts that might otherwise go overlooked. A highly helpful method for identifying co-occurring commodities in consumer buying baskets is market basket analysis. Decisions concerning marketing initiatives like cross-sell campaigns, inventory management, and promotional assistance can be made using this information as a foundation. The primary goal of the study paper is to determine how the various products in a grocery store variety interact with one another and how to use these relationships for marketing purposes. They will learn important details about co-occurrences and co-purchases of products by mining association rules from transactional data. Decisions concerning marketing initiatives like cross-sell campaigns, inventory management, and promotional assistance can be made using this information as a foundation. Two algorithms are used to find association rules. (Apriori method and FP growth). Using R tool and Rapid miner, one may discover frequently occurring item sets. According to the article, FP growth is substantially slower in Rapid Miner and faster in R Programming when using the Apriori approach. The sample size for the analysis is 300, and the data were gathered from a Mumbai retail store.

Market Basket Analysis Using Apriori and FP Growth Algorithm

Market basket analysis finds out customers' purchasing patterns by discovering important associations among different products. It not only assists in decision making process but also increases sales for many business organizations. Apriori and FP Growth are the two most common algorithms for mining frequent itemsets. Mining association rules is one of the most important measurements. An association rule is of the form X => Y , where X is referred to the antecedent and Y is referred as the consequent. They have used Apriori and FP Growth algorithms for discovering popular items in transactional datasets. Apriori algorithm has been used to find the itemsets which are frequent by using JAVA programming language. Authors have tried to build a model of customers purchase behavior in the e-commerce, which is known as customer purchase prediction. They have used extended bakery datasets to detect outliers. For Bakery Shop dataset there are four columns- Date, Time, Transaction, Item. They have first checked for 'NONE' values in these four columns. Then they used TransactionEncoder to map the items on per-transaction. This means if product is present on that transaction, then the value of the product is 1, otherwise 0. With 55% product reduction the required time for both algorithms are less than with the same amount of product reduction. In order to support their proposed approach more precisely they have done sampling without replacement in the datasets. The number of generated rules for 55% reduction are totally similar to the rules which are generated using without reduction. They have. chosen 1500 transactions and made associations among frequent itemsets in transactions for each sample. Then they have again randomly chosen another 1500 transactions and made. associations among them. There are many rules generated in each sample but they have taken only those which have similarity with the rules which are.generated from without product reduction. When they take 50% product reduction, the number of association rules and frequent itemsets generated are totally different to the rules which they get from without reduction. Both Apriori and FFP Growth algorithms have been tested for this purpose and the results show that both algorithms take shorter time when using reduced products. After 50% reduction, they have done sampling without replacement. As there are 9465 transactions, each has 1893 transactions in it.

DESIGN


Planning
Data Collection
Data Validating & Data Cleaning
Implementation
Data Format Preparation
Implementing Algorithms
Application
Building models for Recommending products
Publicised on GitHub Platform

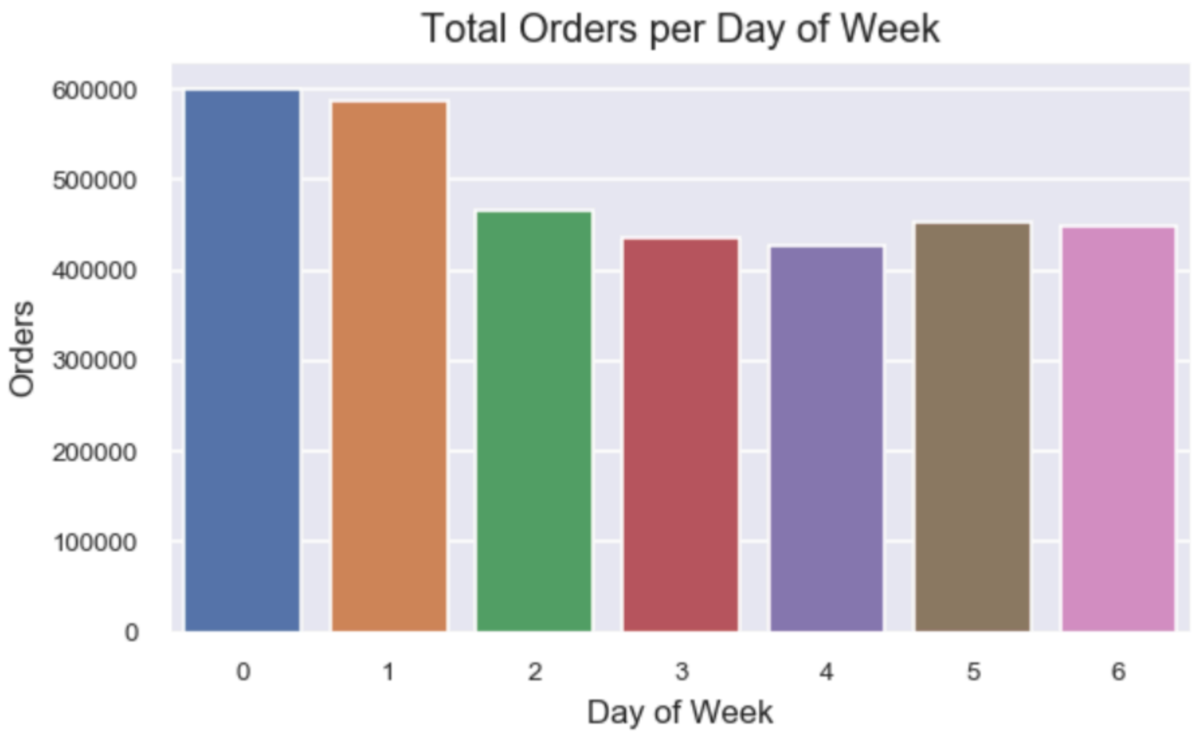
DATA DESCRIPTION

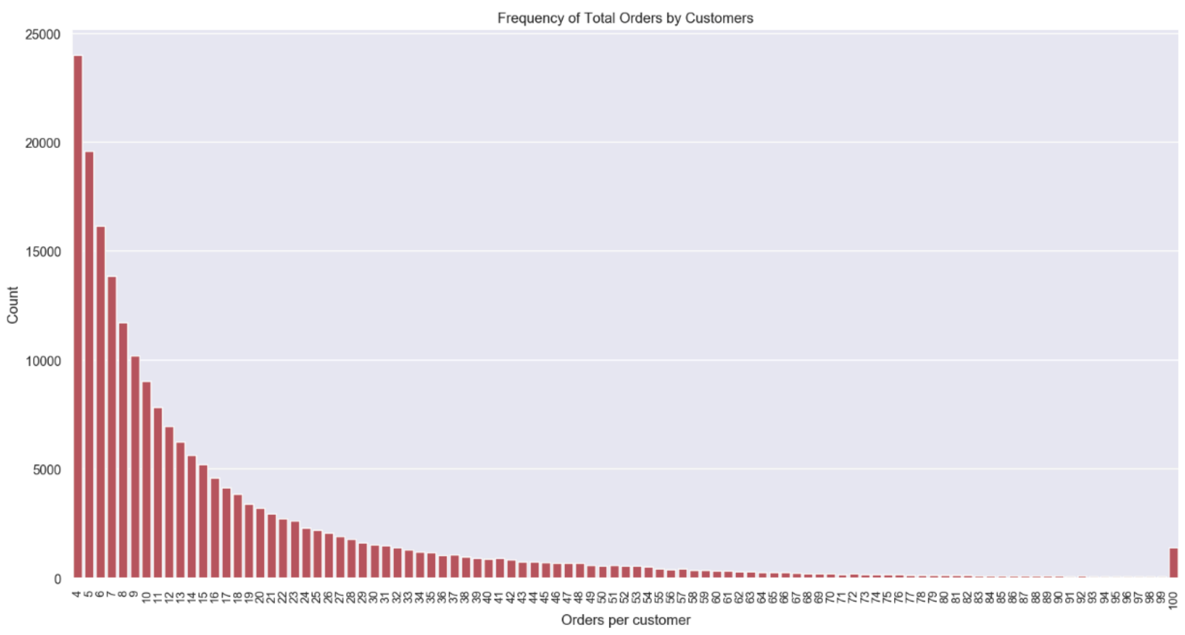
aisles: There are 134 separate aisles in this file, all of them are different.

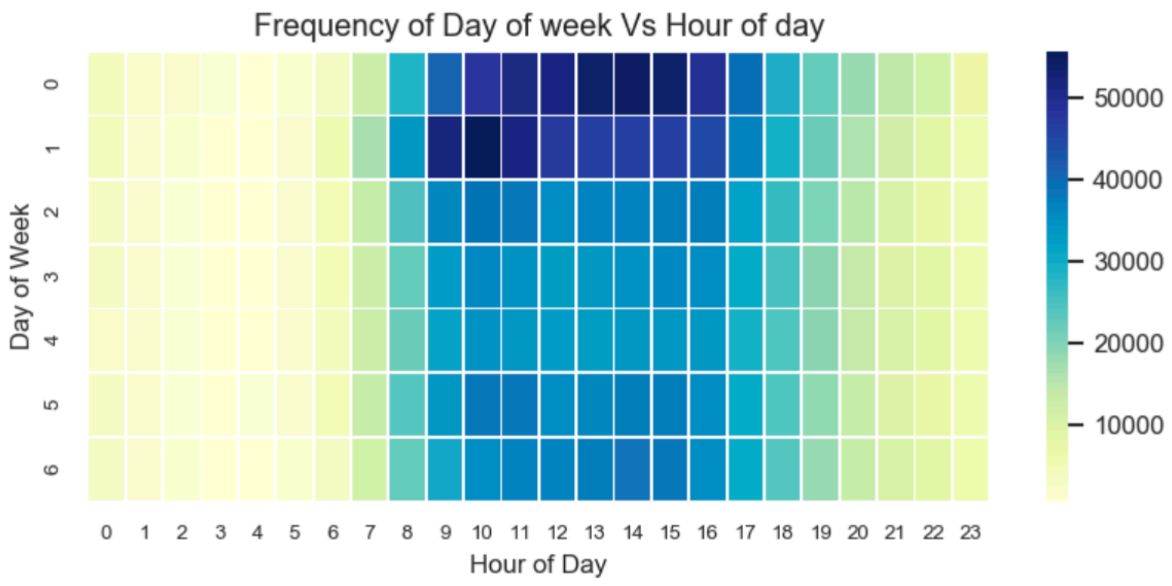
departments: There are a total of 21 distinct departments included in this file.

orders: This file includes all of the orders that various users have placed. Following is what we may infer from the analysis below:

- A total of 206209 users have placed 3421083 orders.
- The Prior, Train, and Test sets of orders are the three sets. While order distributions in the Train and Test sets are comparable, order distributions in the Prior set are different.
- The range of a customer's total orders is 0 to 100.
- Assuming that most people shop for groceries on the weekends, we can map 0 and 1 as Saturday and Sunday, respectively, based on the "Orders VS Day of Week" plot.
- The majority of orders are placed in the morning.
- The peaks at 7, 14, and 21 and 30 in the "Orders VS Days since Prior Order" graph indicate that customers place one order per week.
- Saturday afternoons and Sunday mornings are the busiest times for orders, according to the heatmap between "Day of Week" and "Hour of Day."



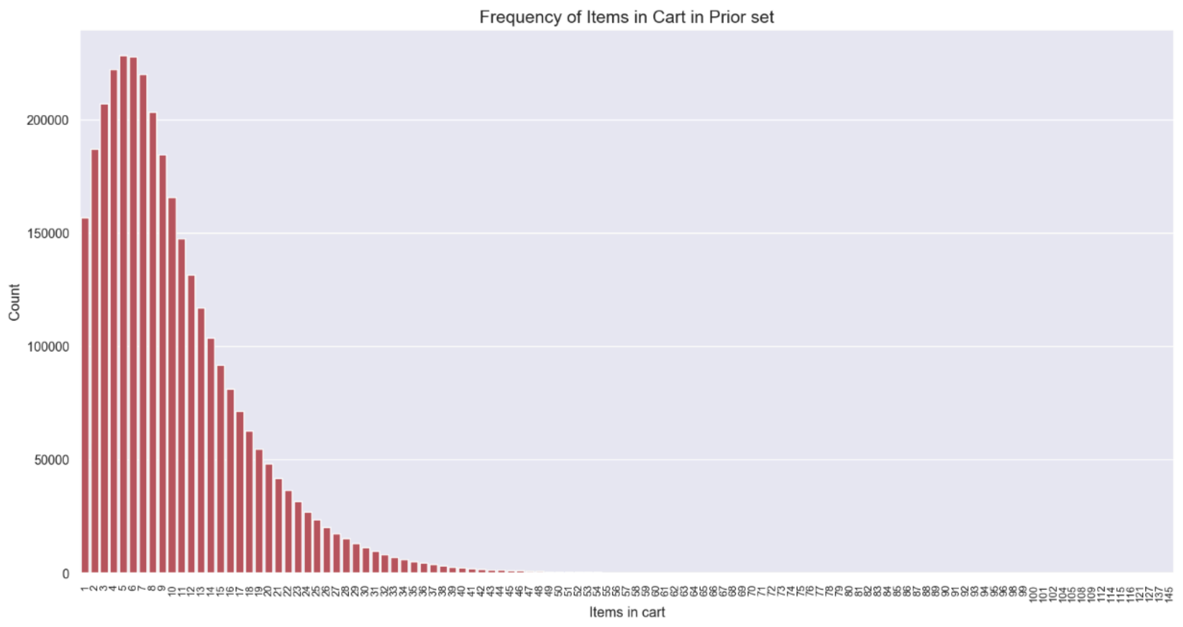




products: This file includes a list of all 49688 items, together with information about each one's aisle and department. Different aisles and sections have varying numbers of merchandise.

order products prior: This file contains details on the items that were ordered and the order in which they were put in the shopping cart. It also lets us know whether or not the goods was ordered again.

- This file contains details on the 3214874 orders that total 49677 goods were ordered through.
- According to the "Count VS Things in Cart" plot, the majority of customers only order 1 to 15 items, with orders including a maximum of 145 items.
- In this set, 58.97% of the elements are repeats.



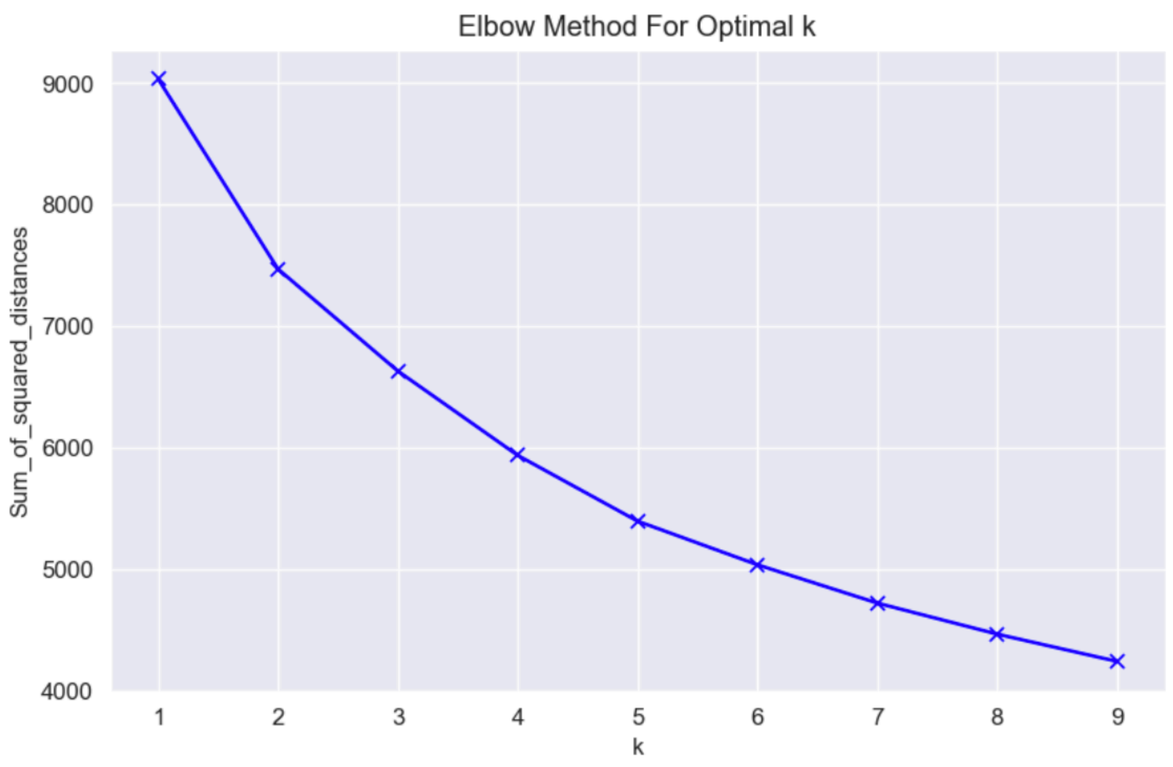
order products train: This file contains details on the items that were ordered and the order in which they were put in the shopping cart. It also lets us know whether or not the goods was ordered again.

- There is data on a total of 131209 orders, through which a total of 39123 products were ordered, in this dataset.
- According to the "Count VS Things in Cart" plot, the majority of customers only order 1 to 15 items, with orders including a maximum of 145 items.
- In this set, 59.86% of the elements are repeats.

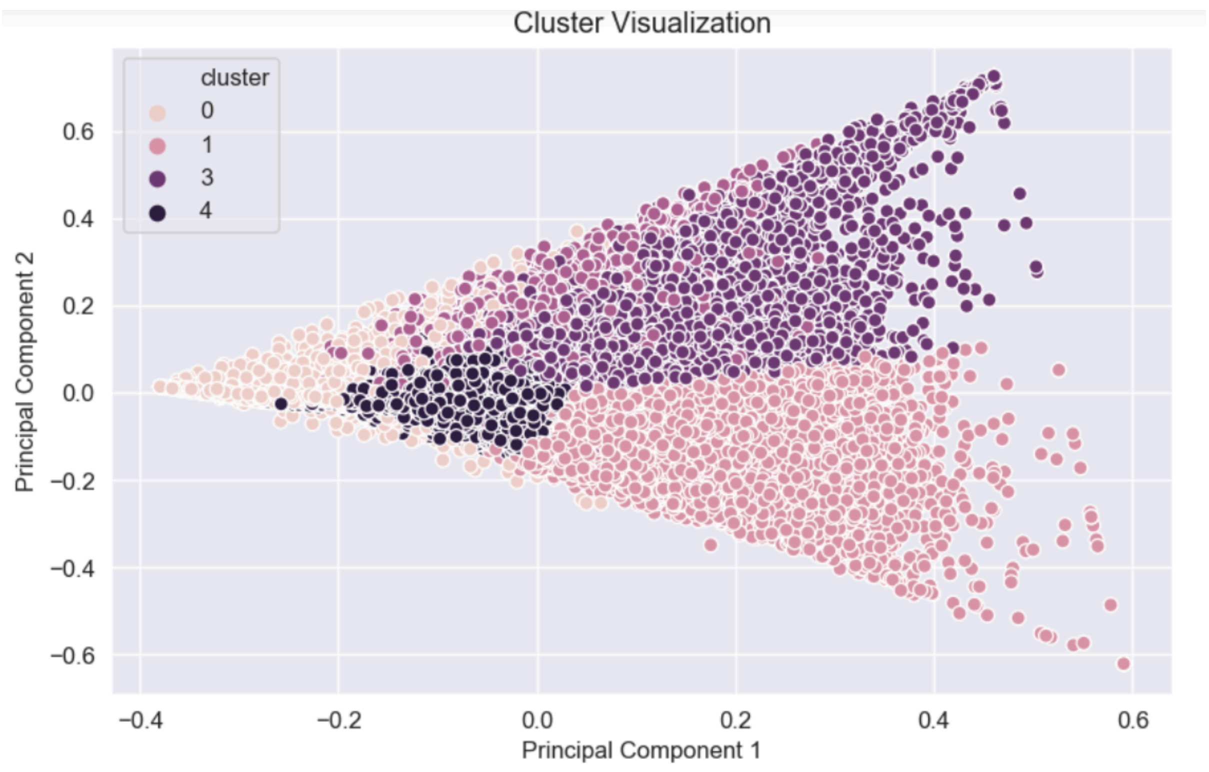
CUSTOMER SEGMENTATION

Customer segmentation is the practise of classifying customers into groups based on shared traits so that businesses may effectively and appropriately market to each group. Utilizing information on the things that consumers purchase, we can segment the population. We used aisles that symbolise different product categories because there are thousands of customers and hundreds of thousands of products.

As KMeans does not work well on higher dimensions, we next used principal component analysis to lower the number of dimensions. I performed KMeans clustering using 10 major components. Using the Elbow approach, which is depicted below, we determined that 5 clusters was the ideal quantity.



The clustering can be seen as the following visualisation of the first two main components.



The clustering produces 5 tidy clusters, and after examining the most prevalent products there, we can draw the following conclusions:

- 5428 customers in Cluster 1 show a very strong affinity for the sparkling water and water seltzer aisle.
- 55784 consumers make up Cluster 2, majority of them order fresh vegetables and then fruits.
- 7948 consumers are found in cluster 3 who primarily purchase packaged vegetables and fresh fruits.
- 37949 consumers fall under cluster 4 and they strongly like fruits, followed by fresh vegetables.
- 99100 customers from Cluster 5 place orders for goods from numerous aisles. Their average number of orders is modest in comparison to other clusters, indicating that they are either infrequent Instacart users or new customers with little orders as of yet.

MARKET BASKET ANALYSIS

A modelling technique called market basket analysis is based on the idea that, depending on whatever group of goods you purchase, you are either more or less likely to purchase an other set of goods as well. The retailer may use market basket analysis to get information about a customer's buying habits. Cross-selling and up-selling strategies, as well as sales campaigns, loyalty programmes, store layouts, and discount schemes, can all be influenced by the information provided.

The goods that customers frequently purchase together are examined by market basket analysis, which uses the data to determine which products should be marketed or sold cross-sell. The phrase refers to the amount of items that grocery store customers load into their trolleys while they are there.

A transaction database, relational database, or other information repository can be searched for regular patterns using association rule mining, which is used to identify associations between various items in a set.

The most popular method to identify these trends is market basket analysis, which is a crucial method employed by major retailers like Amazon, Flipkart, etc. to examine customer purchasing patterns by identifying associations between the various items that customers place in their "shopping baskets." By collecting knowledge on which products customers usually buy together, the identification of these associations can assist businesses in creating marketing strategies. Among the tactics could be:

- Trend-based layout changes for the store
- An examination of consumer behaviour
- Cross-selling in internet stores through catalogue design
- customised newsletters that include sales and other extras

Matrices

Support: This is how popular something is by default. The proportion of transactions involving item A to all transactions, expressed mathematically, is the support of item A.

Confidence: A customer who purchased both A and B is likely to have confidence. It is the proportion between the volume of transactions involving B to those involving A and both.

- Assurance (A -> B) = Support (A, B) / Support (B)

Lift: Increasing A's sales when B's sales increase.

- Confidence (A, B)/Support = Lift (A => B) (B)

- No association exists within the itemset if Lift (A => B) = 1.
- Lift (A => B) > 1 denotes a positive correlation within the itemset, i.e., A and B are more likely to be bought together while shopping for other items in the itemset.
- Lift (A => B) 1 denotes a negative correlation between the items in the set, i.e., A and B are not likely to be purchased together.

Apriori Algorithm: Apriori algorithm makes the erroneous assumption that every subset of a frequent itemset must also be frequent. Market Basket Analysis is powered by an algorithm. Let's use the example of a transaction that also includes grapes and mango. According to the a priori rule, if "Grapes, Apple, Mango" are common, then "Grapes, Mango" must also be common.

We used the Mlxtend Python library's apriori method to identify associations among the top 100 most common items, and as a result, 28 product pairs (a total of 56 rules) with lift greater than 1 were identified.

ML MODELS AND FEATURE EXTRACTION

We can forecast which previously purchased products will be in a user's future order using this anonymised transactional data of consumer orders over time. This would support a user's recommendation of the products.

In order to create a model, we must take features from prior orders in order to comprehend users' buying habits and the level of popularity of a given product. From the user's transactional data, the following features were retrieved.

Features at the Product Level: To comprehend the product's acceptance by Users

```
(1) The typical add-to-cart order per product
(2) Total orders for the product
(3) Total orders for reorders of the product
```

```
(4) Reorder percentage
(5) Total users (unique users)
(6) Is the product organic?
(7) The proportion of users that purchase the product a second time.
```

Aisle and Department Level Features: To identify whether an aisle and department are associated with regularly occurring products (vegetables, fruits, soda, water, etc.) or sporadic products (medicines, personal-care, etc.)

```
(8) A product aisle's total orders and reorders, as well as its reorder percentage
(9) Aisle add-to-cart average and mean
(10) Exclusive aisle users
(10) A product department's total orders and reorders, as well as its reorder
percentage
(11) The department's mean and standard add-to-cart order
(12) Specialized department users
(13) Aisle feature binary encoding (Because one-hot encoding results in many
features and make datarame sparse)
(14) Department feature binary encoding (Because one-hot encoding results in many
features and make datarame sparse)
```

User Level Features: to record user behaviour and purchasing patterns

```
(15) User's average and standard deviation of the order's day of the week
(16) User's average and standard deviation of the order's hour
(17) User's average and standard deviation of the order's days since the preceding
order
(18) Total orders made by a user
(19) total products purchased
(20) total unique products purchased
(21) total reordered products
(22) average order size of a user
(23) and mean number of reordered items across all purchases
(24) percentage of items in the user's last three orders that were reordered
(25) The sum of the user's most recent three orders
```

Features at the user-product level: To record user ordering and reordering patterns for certain products

```
(26) User's average days since preceding order
(27) User's average add-to-cart order for a product
```

```
(28) Total orders, reorders, and reorder % for the user's products
(29) the user's order number from the most recent purchase
(30) the recent three orders's history of the user's product purchases
```

ML MODELS

We created a dataframe with all the products the user has previously purchased, user level features, product level features, asile and department level features, user-product level features, and information about the current order like the day of the week, hour of the day, etc. using the extracted features. The number of previously purchased things that the user ordered this time would be indicated on the Traget by the word "reordered."

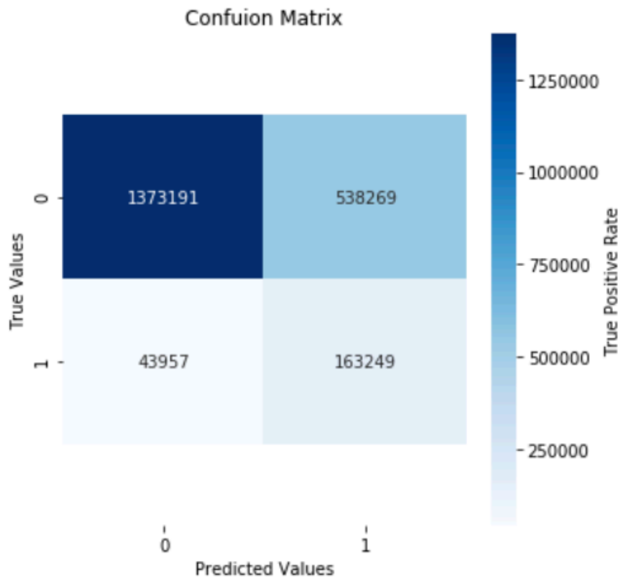
Because the dataframe is so large, we downcast it to fit the data in my memory to reduce the memory use. Because StandardScaler needs 16 GB of RAM to function, we picked MinMaxScaler. We used the conventional model-building procedure and depended on XGBoost because it handles enormous data, can be parallelized, and prioritises features. In order to determine this model's optimal performance while ignoring some inherent unpredictability in each of these models, we also created neural networks. By allocating class weights (0:1, 1:10), we employed cost-sensitive learning to balance the data. Since SMOTE has limited memory and would increase the data size, I have not employed random-upsampling. Additionally, as random-down-sampling omits data that could be crucial and introduce bias.

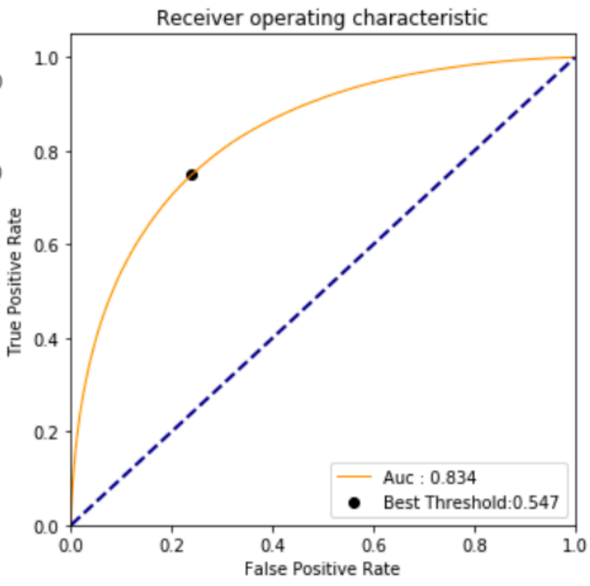
We used the AUC Score for model evaluation since we could manipulate the F1 score by altering the threshold. Below is a performance comparison of both of these models utilising the classification report, ROC curve, and confusion matrix. To comprehend significant features that aid in predicting product reorder, the feature critical plot from the XGBoost model is also shown. Both models perform about equally well, with XGBoost marginally outperforming the other in terms of ROC-AUC.

Neural Network Model Architecture and Performance:

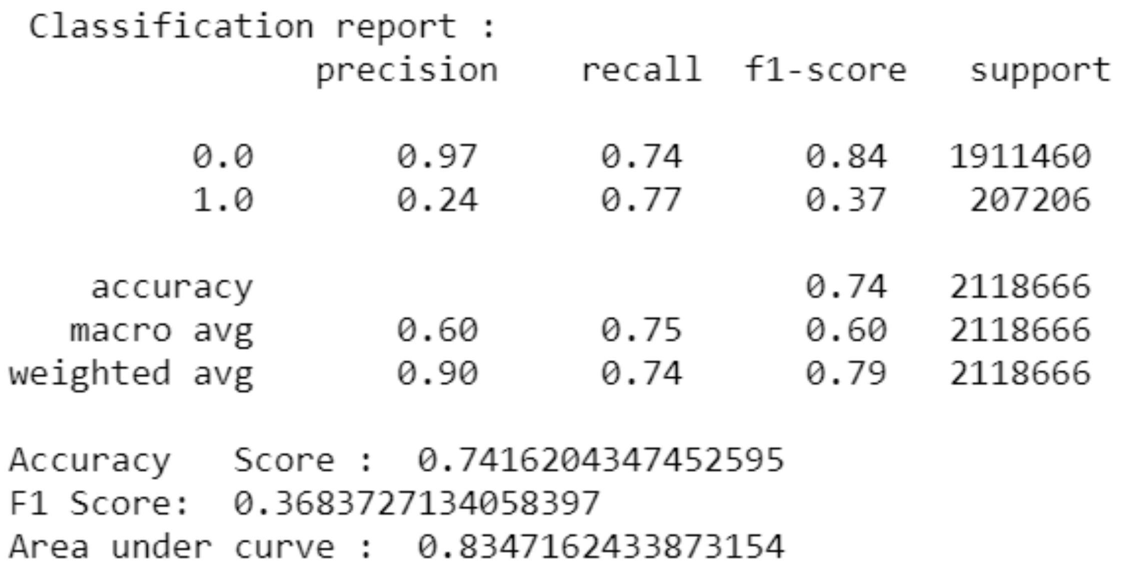
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                4480
_________________________________________________________________
dense_2 (Dense)              (None, 15)                975
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 64
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 5
=================================================================
Total params: 5,524
Trainable params: 5,524
Non-trainable params: 0
_________________________________________________________________
```

```
 Classification report :
               precision    recall  f1-score   support

         0.0       0.97      0.72      0.83   1911460
         1.0       0.23      0.79      0.36    207206

    accuracy                           0.73   2118666
   macro avg       0.60      0.75      0.59   2118666
weighted avg       0.90      0.73      0.78   2118666

Accuracy  Score :  0.7251921728106271
F1 Score: 0.3592928105783494
Area under curve :  0.8344086677839623
```

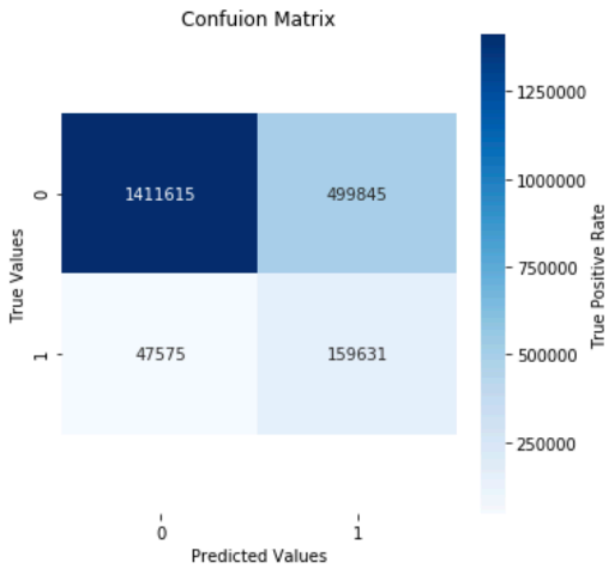


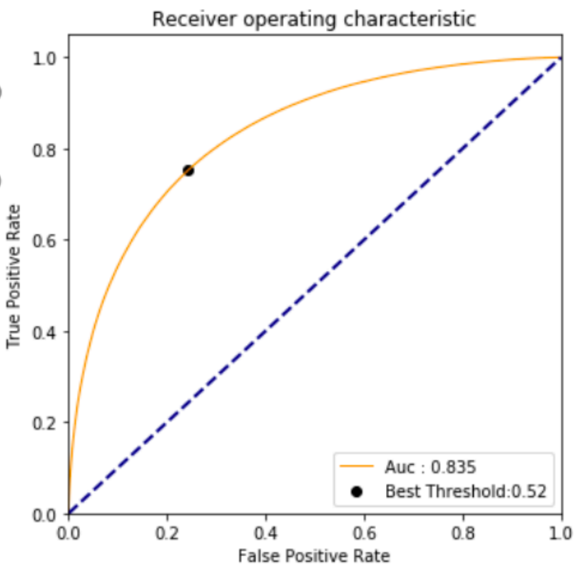


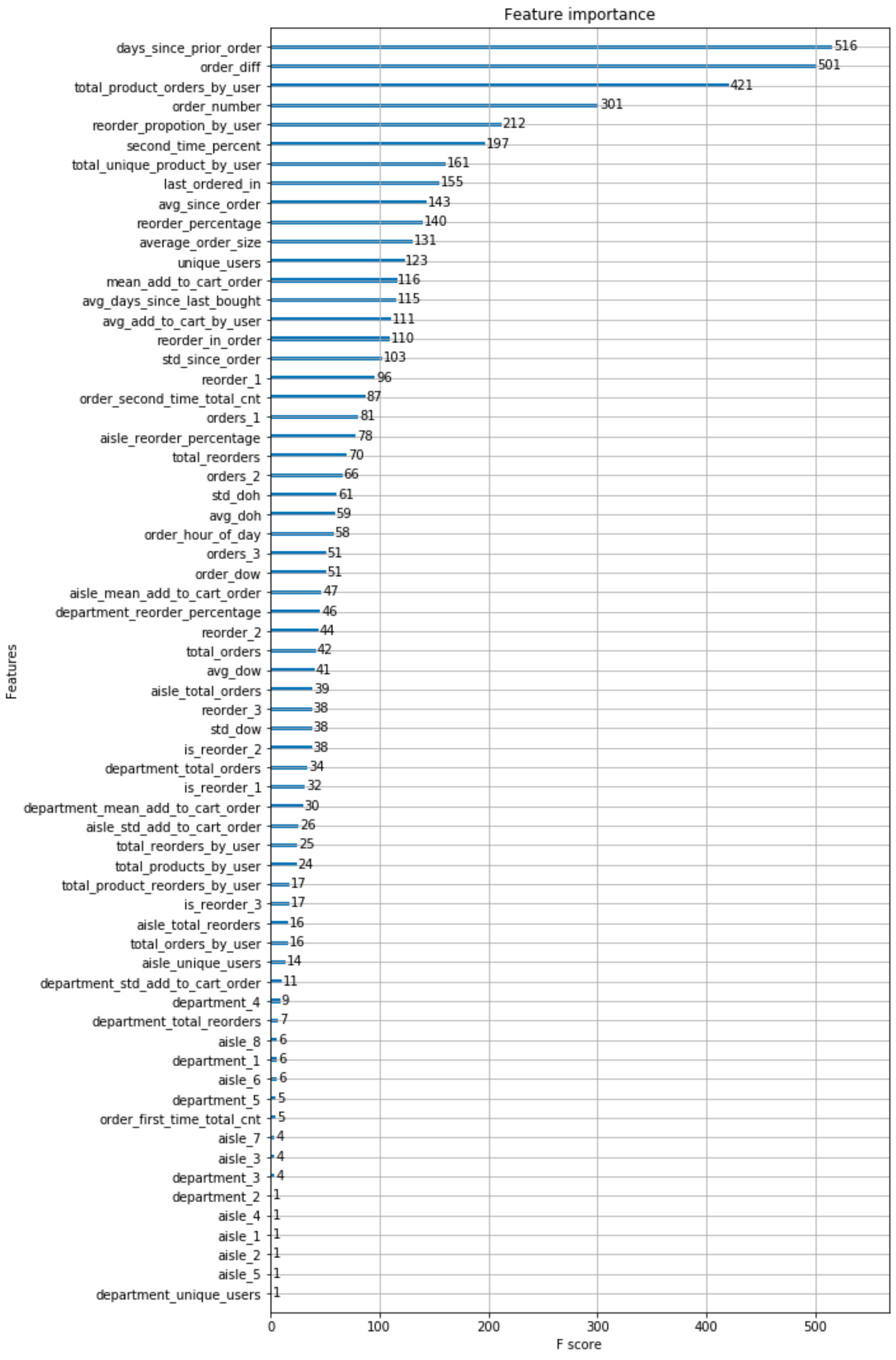
XGBoost Model's Performance and Feature Importance:



```
  Classification report :
                precision    recall  f1-score   support

         0.0       0.97      0.74      0.84   1911460
         1.0       0.24      0.77      0.37    207206

    accuracy                           0.74   2118666
   macro avg       0.60      0.75      0.60   2118666
weighted avg       0.90      0.74      0.79   2118666

Accuracy   Score :  0.7416204347452595
F1 Score:  0.3683727134058397
Area under curve :  0.8347162433873154
```



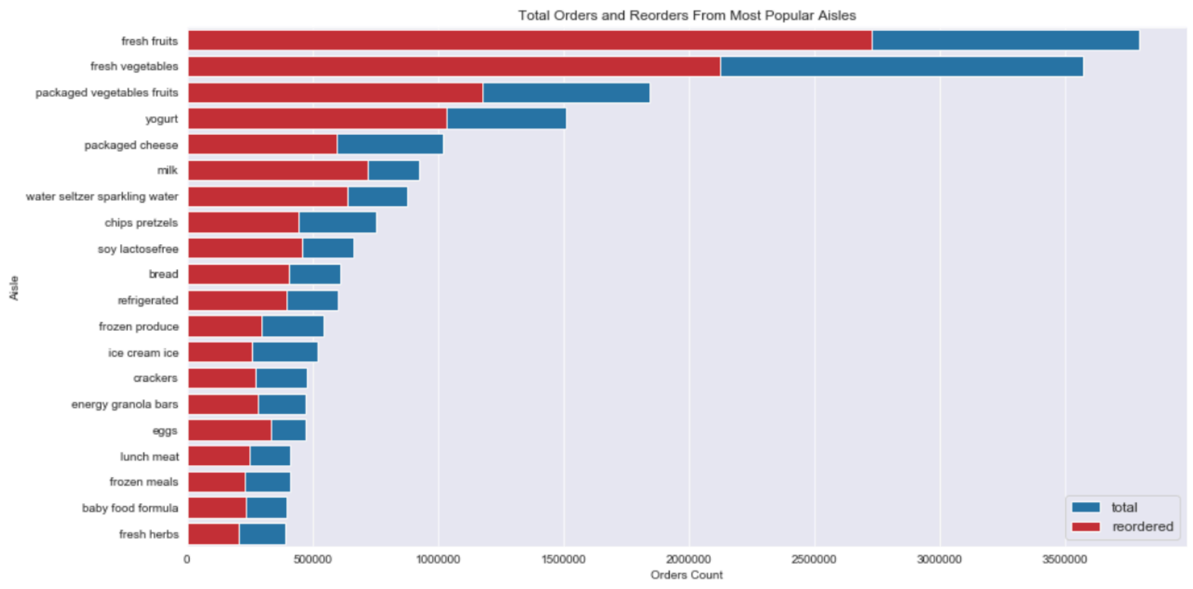



Feature importance
days_since_prior_order 516
order_diff 501
total_product_orders_by_user 421
order_number 301
reorder_propotion_by_user 212
second_time_percent 197
total_unique_product_by_user 161
last_ordered_in 155
avg_since_order 143
reorder_percentage 140
average_order_size 131
unique_users 123
mean_add_to_cart_order 116
avg_days_since_last_bought 115
avg_add_to_cart_by_user 111
reorder_in_order 110
std_since_order 103
reorder_1 96
order_second_time_total_cnt 87
orders_1 81
aisle_reorder_percentage 78
total_reorders 70
orders_2 66
std_doh 61
avg_doh 59
order_hour_of_day 58
orders_3 51
order_dow 51
aisle_mean_add_to_cart_order 47
department_reorder_percentage 46
reorder_2 44
total_orders 42
avg_dow 41
aisle_total_orders 39
reorder_3 38
std_dow 38
is_reorder_2 38
department_total_orders 34
is_reorder_1 32
department_mean_add_to_cart_order 30
aisle_std_add_to_cart_order 26
total_reorders_by_user 25
total_products_by_user 24
total_product_reorders_by_user 17
is_reorder_3 17
aisle_total_reorders 16
total_orders_by_user 16
aisle_unique_users 14
department_std_add_to_cart_order 11
department_4 9
department_total_reorders 7
aisle_8 6
department_1 6
aisle_6 6
department_5 5
order_first_time_total_cnt 5
aisle_7 4
aisle_3 4
department_3 4
department_2 1
aisle_4 1
aisle_1 1
aisle_2 1
aisle_5 1
department_unique_users 1
Features
0
100
200
300
400
500
F score

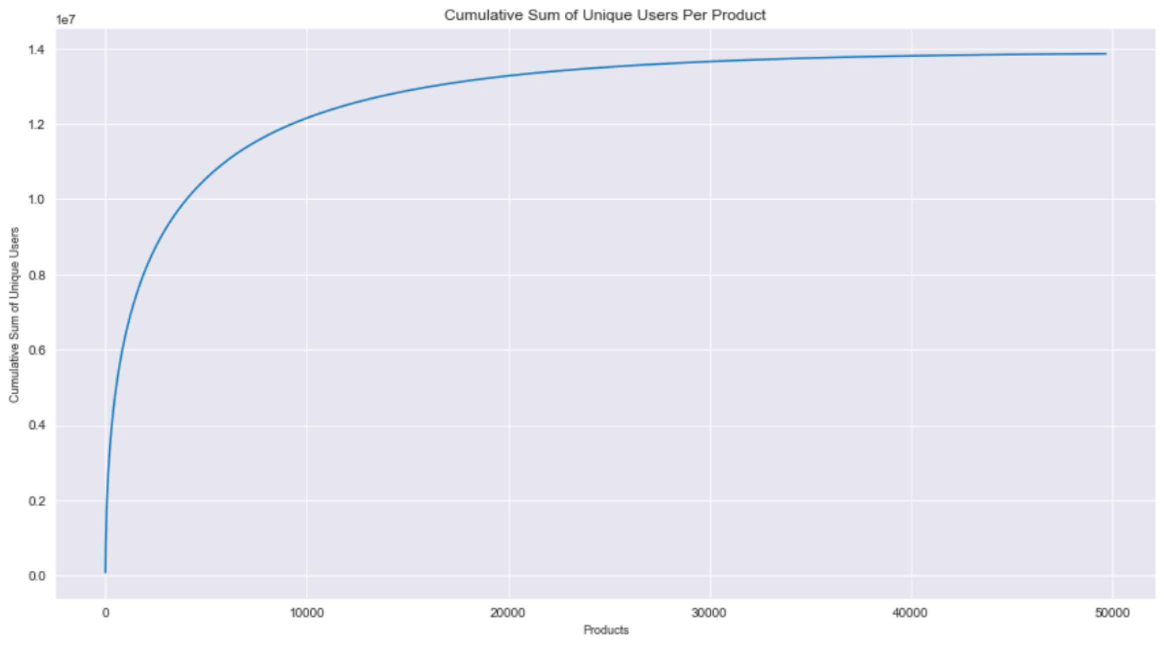
EXPLORATORY DATA ANALYSIS

In order to accommodate the dataframe into my memory, we reduced the size of the dataframe by 50% (from 4.1 GB to 2.0 GB) by type conversion, without losing any of the analysis's information.

Based on total products purchased, this plot displays the most popular aisles.



We can see that 85% of users only purchase 10000 products out of a total of 49688 products in the accompanying plot of cumulative total users per product vs. products. We should only have only 10,000 items if we want to maximise the use of shelf space. Here, we make the assumption that the other 39688 products' profits won't be particularly high. If these products had prices, we could have determined which ones had the most income, reorder rate, and overall product sales.



CONCLUSION

We can perform exploratory data analysis on the distribution, correlation, and correction of our data and find useful patterns that can be used for our analysis.
We can draw the conclusion whether age plays a factor in increasing the revenue of the firm and can accordingly perform the marketing analysis to get better results.
We can understand how customers purchase different goods and determine the times of the year when they are most active so as to generate the maximum revenue and increase the production during those times.

FUTURE SCOPE

Utilize Collaborative filtering to recommend products to a customer.

REFERENCES

[1]Computers & Electrical Engineering.(2020, June 12).A machine learning-based approach to enhancing social media marketing.Retrieved on November 12, 2022 from https://www.sciencedirect.com/science/article/abs/pii/S0045790620305784

[2]Machine Learning in Marketing: A Systematic ... - ResearchGate. Retrieved on November 12, 2022 from https://www.researchgate.net/publication/345434602_Machine_Learning_in_Marketing_A_Systematic_Literature_and_Text_Mining_Research

[3]Machine Learning in Marketing: A Systematic Literature and Text Mining.Retrieved on November 12, 2022 from https://www.researchgate.net/publication/345434602_Machine_Learning_in_Marketing_A_Systematic_Literature_and_Text_Mining_Research

[4]IEEE Xplore Full-Text PDF.Retrieved on November 12, 2022 from https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8746184

[5]IEEE Xplore.Towards the Adoption of Machine Learning-Based Analytical Tools in Digital Marketing.Retrieved on November 12, 2022 from https://ieeexplore.ieee.org/document/8746184

Appendix- 1(Code Folder)

ANN Model

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import gc
pd.options.mode.chained_assignment = None

root = 'C:/Data/instacart-market-basket-analysis/'

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import roc_auc_score, roc_curve, precision_score, recall_score,
f1_score
from imblearn.over_sampling import SMOTE
df = pd.read_pickle(root + 'Finaldata.pkl')
df.head()
def reduce_memory(df):

    """
    This function reduce the dataframe memory usage by converting it's type for easier
handling.

    Parameters: Dataframe
    Return: Dataframe
    """

    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB")

    for col in df.columns:
        if df[col].dtypes in ["int64", "int32", "int16"]:
            cmin = df[col].min()
            cmax = df[col].max()

            if cmin > np.iinfo(np.int8).min and cmax < np.iinfo(np.int8).max:
                df[col] = df[col].astype(np.int8)

            elif cmin > np.iinfo(np.int16).min and cmax < np.iinfo(np.int16).max:
                df[col] = df[col].astype(np.int16)

            elif cmin > np.iinfo(np.int32).min and cmax < np.iinfo(np.int32).max:
                df[col] = df[col].astype(np.int32)

        if df[col].dtypes in ["float64", "float32"]:

            cmin = df[col].min()
            cmax = df[col].max()

            if cmin > np.finfo(np.float16).min and cmax < np.finfo(np.float16).max:
                df[col] = df[col].astype(np.float16)

            elif cmin > np.finfo(np.float32).min and cmax < np.finfo(np.float32).max:
                df[col] = df[col].astype(np.float32)
print("")
    print("___MEMORY USAGE AFTER COMPLETION:___")
    mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage is: ",mem_usg," MB")
    print("This is ",100*mem_usg/start_mem_usg,"% of the initial size")

    return df



df = reduce_memory(df)
df['order_diff'] = df.order_number - df.last_ordered_in
```

```
df.drop(['user_id', 'product_id'], axis = 1, inplace = True)


df.head()
df.shape
label = 'reordered'
x_cols = df.columns.drop('reordered')


X = df[x_cols]
y = df[label]


X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.25)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
y_train.value_counts()
np.ceil(y_train.value_counts()[0]/y_train.value_counts()[1])
y_test.value_counts()
# freeing memory
del df, X, y
gc.collect()
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.regularizers import l2
from keras.callbacks import History
from keras import backend as K
from sklearn.preprocessing import MinMaxScaler

In [16]:
sc = MinMaxScaler()
X_train_sc = sc.fit_transform(X_train)
```

```
X_test_sc = sc.transform(X_test)

In [17]:
input_dim =  X_train_sc.shape[1]
input_dim
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

In [19]:
history = History()

classifier = Sequential()

classifier.add(Dense(units = 64, activation = 'relu', input_dim = input_dim))
classifier.add(Dense(units = 15, activation = 'relu'))
classifier.add(Dense(units = 4, activation = 'relu'))
classifier.add(Dense(units = 1, activation ='sigmoid'))

classifier.compile(optimizer = "adam", loss = 'binary_crossentropy', metrics = ['accuracy',
f1_m, precision_m, recall_m])
```

```
classifier.summary()
%%time
# fit the model
classifier.fit(X_train_sc, y_train, epochs=50, batch_size=512, validation_split=0.15,
verbose=1,class_weight= {0:1, 1:10},
        callbacks = [history, keras.callbacks.EarlyStopping(monitor='val_loss',
                                            min_delta=0, patience=10, verbose=0, mode='auto')])
eval_model=classifier.evaluate(X_train_sc, y_train)
print('loss: ', eval_model[0], 'and Accuracy: ', eval_model[1])
fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 4))

# Accuracy
ax[0].plot(history.history['acc'])
ax[0].plot(history.history['val_acc'])
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('# Epoch')
ax[0].legend(['train', 'test'], loc='upper left')
ax[0].set_title('Accuracy')

# Loss
ax[1].plot(history.history['loss'])
ax[1].plot(history.history['val_loss'])
ax[1].set_ylabel('Loss')
ax[1].set_xlabel('# Epoch')
ax[1].legend(['train', 'test'], loc='upper left')
ax[1].set_title('Loss')
probabilities = classifier.predict_proba(X_test_sc)
predictions = classifier.predict_classes(X_test_sc)

print ("\n Classification report : \n",classification_report(y_test, predictions))
print ("Accuracy   Score : ",accuracy_score(y_test, predictions))

#confusion matrix
```

```
conf_matrix = confusion_matrix(y_test,predictions)
plt.figure(figsize=(12,12))
plt.subplot(221)
sns.heatmap(conf_matrix, fmt = "d",annot=True, cmap='Blues')
b, t = plt.ylim()
plt.ylim(b + 0.5, t - 0.5)
plt.title('Confuion Matrix')
plt.ylabel('True Values')
plt.xlabel('Predicted Values')

#f1-score
f1 = f1_score(y_test, predictions)
print("F1 Score: ", f1)

#roc_auc_score
model_roc_auc = roc_auc_score(y_test,probabilities)
print ("Area under curve : ",model_roc_auc,"\n")
fpr,tpr,thresholds = roc_curve(y_test,probabilities)
gmeans = np.sqrt(tpr * (1-fpr))
ix = np.argmax(gmeans)
threshold = np.round(thresholds[ix],3)

plt.subplot(222)
plt.plot(fpr, tpr, color='darkorange', lw=1, label = "Auc : %.3f" %model_roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.scatter(fpr[ix], tpr[ix], marker='o', color='black', label='Best Threshold:' + str(threshold))
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

plt.show()
```

Data Description and Analysis

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', 150)

import gc
import os

# matplotlib and seaborn for plotting
import matplotlib
matplotlib.rcParams['figure.dpi'] = 120 #resolution
matplotlib.rcParams['figure.figsize'] = (8,6) #figure size

import matplotlib.pyplot as plt
sns.set_style('darkgrid')
import seaborn as sns
color = sns.color_palette()

root = 'C:/Data/instacart-market-basket-analysis/'
os.listdir(root)
aisles = pd.read_csv(root + 'aisles.csv')
departments = pd.read_csv(root + 'departments.csv')
orders = pd.read_csv(root + 'orders.csv')
order_products_prior = pd.read_csv(root + 'order_products__prior.csv')
order_products_train = pd.read_csv(root + 'order_products__train.csv')
products = pd.read_csv(root + 'products.csv')
aisles.head()
aisles.tail()
len(aisles.aisle.unique())
aisles.aisle.unique()
departments.head()
departments.tail()
len(departments.department.unique())
```

```
departments.department.unique()
orders.head(12)
orders.tail()
orders.info()
len(orders.order_id.unique())
len(orders.user_id.unique())
orders.eval_set.value_counts()
orders.order_number.describe().apply(lambda x: format(x, '.2f'))
order_number = orders.groupby('user_id')['order_number'].max()
order_number = order_number.value_counts()
```

In [322]:

```
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.barplot(x = order_number.index, y = order_number.values, color = color[3])
ax.set_xlabel('Orders per customer')
ax.set_ylabel('Count')
ax.xaxis.set_tick_params(rotation=90, labelsize=10)
ax.set_title('Frequency of Total Orders by Customers')
fig.savefig('Frequency of Total Orders by Customers.png')

fig, ax = plt.subplots(figsize = (8,4))
ax = sns.kdeplot(orders.order_number[orders.eval_set == 'prior'], label = "Prior set", lw = 1)
ax = sns.kdeplot(orders.order_number[orders.eval_set == 'train'], label = "Train set", lw = 1)
ax = sns.kdeplot(orders.order_number[orders.eval_set == 'test'], label = "Test set", lw = 1)
ax.set_xlabel('Order Number')
ax.set_ylabel('Count')
ax.tick_params(axis = 'both', labelsize = 10)
ax.set_title('Distribution of Orders in Various Sets')
fig.savefig('Distribution of Orders in Various Sets.png')
plt.show()

fig, ax = plt.subplots(figsize = (5,3))
```

```
ax = sns.countplot(orders.order_dow)
ax.set_xlabel('Day of Week', size = 10)
ax.set_ylabel('Orders', size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
ax.set_title('Total Orders per Day of Week')
fig.savefig('Total Orders per Day of Week.png')
plt.show()

temp_df = orders.groupby('order_dow')['user_id'].nunique()

fig, ax = plt.subplots(figsize = (5,3))
ax = sns.barplot(x = temp_df.index, y = temp_df.values)
ax.set_xlabel('Day of Week', size = 10)
ax.set_ylabel('Total Unique Users', size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
ax.set_title('Total Unique Users per Day of Week')
fig.savefig('Total Unique Users per Day of Week.png')
plt.show()

fig, ax = plt.subplots(figsize = (10,5))
ax = sns.countplot(orders.order_hour_of_day, color = color[2])
ax.set_xlabel('Hour of Day', size = 10 )
ax.set_ylabel('Orders', size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
ax.set_title('Total Orders per Hour of Day')
fig.savefig('Total Orders per Hour of Day.png')
plt.show()

fig, ax = plt.subplots(figsize = (10,5))
ax = sns.countplot(orders.days_since_prior_order, color = color[2])
ax.set_xlabel('Days since prior order', size = 10)
ax.set_ylabel('Orders', size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
ax.set_title('Orders VS Days since prior order')
fig.savefig('Orders VS Days since prior order.png')
plt.show()

temp_df = orders.groupby(["order_dow",
"order_hour_of_day"])["order_number"].aggregate("count").reset_index()
temp_df = temp_df.pivot('order_dow', 'order_hour_of_day', 'order_number')
temp_df.head()

ax = plt.subplots(figsize=(7,3))
ax = sns.heatmap(temp_df, cmap="YlGnBu", linewidths=.5)
ax.set_title("Frequency of Day of week Vs Hour of day", size = 12)
ax.set_xlabel("Hour of Day", size = 10)
ax.set_ylabel("Day of Week", size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=10)
fig = ax.get_figure()
fig.savefig("Frequency of Day of week Vs Hour of day.png")
plt.show()

order_products_prior.head(10)
order_products_prior.tail()
len(order_products_prior.order_id.unique())
len(order_products_prior.product_id.unique())
add_to_cart_order_prior =
order_products_prior.groupby('order_id')['add_to_cart_order'].count()
add_to_cart_order_prior = add_to_cart_order_prior.value_counts()
add_to_cart_order_prior.head()
add_to_cart_order_prior.tail()
add_to_cart_order_prior.index.max()
fig, ax = plt.subplots(figsize = (15,8))
```

```
ax = sns.barplot(x = add_to_cart_order_prior.index, y = add_to_cart_order_prior.values,
color = color[3])
ax.set_xlabel('Items in cart')
ax.set_ylabel('Count')
ax.xaxis.set_tick_params(rotation=90, labelsize = 9)
ax.set_title('Frequency of Items in Cart in Prior set', size = 15)
fig.savefig('Frequency of Items in Cart in Prior set.png')

fig, ax = plt.subplots(figsize=(3,3))
ax = sns.barplot(x = order_products_prior.reordered.value_counts().index,
            y = order_products_prior.reordered.value_counts().values, color = color[3])
ax.set_xlabel('Reorder', size = 10)
ax.set_ylabel('Count', size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
ax.ticklabel_format(style='plain', axis='y')
ax.set_title('Reorder Frequency in Prior Set')
fig.savefig('Reorder Frequency in Prior Set')
plt.show()

print('Percentage of reorder in prior set:',
format(order_products_prior[order_products_prior.reordered
==1].shape[0]*100/order_products_prior.shape[0], '.2f'))
order_products_train.head(10)
order_products_train.tail()
len(order_products_train.order_id.unique())
len(order_products_train.product_id.unique())
add_to_cart_order_train =
order_products_prior.groupby('order_id')['add_to_cart_order'].count()
add_to_cart_order_train = add_to_cart_order_train.value_counts()
add_to_cart_order_train.head()
add_to_cart_order_train.tail()
add_to_cart_order_train.index.max()
```

```
fig, ax = plt.subplots(figsize = (15,8))
ax = sns.barplot(x = add_to_cart_order_train.index, y = add_to_cart_order_train.values, color
= color[2])
ax.set_xlabel('Items in cart')
ax.set_ylabel('Count')
ax.xaxis.set_tick_params(rotation=90, labelsize = 8)
ax.set_title('Frequency of Items in Cart in Train set', size = 15)
fig.savefig('Frequency of Items in Cart in Train set.png')

fig, ax = plt.subplots(figsize=(3,3))
ax = sns.barplot(x = order_products_train.reordered.value_counts().index,
            y = order_products_train.reordered.value_counts().values, color = color[2])
ax.set_xlabel('Reorder', size = 10)
ax.set_ylabel('Count', size = 10)
ax.tick_params(axis = 'both', labelsize = 8)
ax.set_title('Reorder Frequency in Train Set')
fig.savefig('Reorder Frequency in Train Set')
plt.show()

print('Percentage of reorder in train set:',
    format(order_products_train[order_products_train.reordered ==
1].shape[0]*100/order_products_train.shape[0], '.2f'))
products.head(10)
products.tail()
len(products.product_name.unique())
len(products.aisle_id.unique())
len(products.department_id.unique())

temp_df = products.groupby('aisle_id')['product_id'].count()

fig, ax = plt.subplots(figsize = (15,6))
ax = sns.barplot(x = temp_df.index, y = temp_df.values, color = color[3])
ax.set_xlabel('Aisle Id')
ax.set_ylabel('Total products in aisle')
ax.xaxis.set_tick_params(rotation=90, labelsize = 7)
ax.set_title('Total Products in Aisle VS Aisle ID', size = 12)
fig.savefig('Total Products in Aisle VS Aisle ID.png')

temp_df = products.groupby('department_id')['product_id'].count()

fig, ax = plt.subplots(figsize = (8,5))
ax = sns.barplot(x = temp_df.index, y = temp_df.values, color = color[2])
ax.set_xlabel('Department Id')
ax.set_ylabel('Total products in department')
ax.xaxis.set_tick_params(rotation=90, labelsize = 9)
ax.set_title('Total Products in Department VS Department ID', size = 10)
fig.savefig('Total Products in Department VS Department ID.png')

temp_df = products.groupby('department_id')['aisle_id'].nunique()

fig, ax = plt.subplots(figsize = (8,5))
ax = sns.barplot(x = temp_df.index, y = temp_df.values)
ax.set_xlabel('Department Id')
ax.set_ylabel('Total Aisles in department')
ax.xaxis.set_tick_params(rotation=90, labelsize = 9)
ax.set_title('Total Aisles in Department VS Department ID', size = 10)
fig.savefig('Total Aisles in Department VS Department ID.png')
```

Data Preparation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import gc
pd.options.mode.chained_assignment = None
```

```
root = 'C:/Data/instacart-market-basket-analysis/'
```

Reading all data

In [2]:

```
orders = pd.read_csv(root + 'orders.csv',
            dtype={
                'order_id': np.int32,
                'user_id': np.int64,
                'eval_set': 'category',
                'order_number': np.int16,
                'order_dow': np.int8,
                'order_hour_of_day': np.int8,
                 'days_since_prior_order': np.float32})
order_products_train = pd.read_csv(root + 'order_products__train.csv',
                    dtype={
                        'order_id': np.int32,
                        'product_id': np.uint16,
                        'add_to_cart_order': np.int16,
                        'reordered': np.int8})

order_products_prior = pd.read_csv(root + 'order_products__prior.csv',
                    dtype={
                        'order_id': np.int32,
                        'product_id': np.uint16,
                        'add_to_cart_order': np.int16,
                        'reordered': np.int8})
product_features = pd.read_pickle(root + 'product_features.pkl')

user_features = pd.read_pickle(root + 'user_features.pkl')

user_product_features = pd.read_pickle(root + 'user_product_features.pkl')
```

```
products = pd.read_csv(root +'products.csv')

aisles = pd.read_csv(root + 'aisles.csv')

departments = pd.read_csv(root + 'departments.csv')
```

merging train order data with orders

```
train_orders = orders.merge(order_products_train, on = 'order_id', how = 'inner')
train_orders.head()
train_orders.drop(['eval_set', 'add_to_cart_order', 'order_id'], axis = 1, inplace = True)
```

unique user_ids in train data

```
train_users = train_orders.user_id.unique()
train_users[:10]

user_product_features.shape

user_product_features.head()\
df = user_product_features[user_product_features.user_id.isin(train_users)]
df.head()
df = df.merge(train_orders, on = ['user_id', 'product_id'], how = 'outer')
df.head()
df.order_number.fillna(df.groupby('user_id')['order_number'].transform('mean'), inplace =
True)
df.order_dow.fillna(df.groupby('user_id')['order_dow'].transform('mean'), inplace = True)
df.order_hour_of_day.fillna(df.groupby('user_id')['order_hour_of_day'].transform('mean'),
inplace = True)
```

```
df.days_since_prior_order.fillna(df.groupby('user_id')['days_since_prior_order'].\
                                                  transform('mean'), inplace = True)
df.reordered.value_counts()


df.reordered.isnull().sum()
df = df[df.reordered != 0]
df.shape
```

Now imputing 0 in reordered as they were not reordered by user in his/her last order.

```
df.reordered.fillna(0, inplace = True)
df.isnull().sum()
df.head()
product_features.head()
user_features.head()
df = df.merge(product_features, on = 'product_id', how = 'left')
df = df.merge(user_features, on = 'user_id', how = 'left')
df.head()
df.shape
df.isnull().sum().sort_values(ascending = False)
df.to_pickle(root + 'Finaldata.pkl')
df2 = pd.read_pickle(root +'Finaldata.pkl')
df2.head()
```

Customer segmentation

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', 150)

import gc
import os
```

```
# matplotlib and seaborn for plotting
import matplotlib
matplotlib.rcParams['figure.dpi'] = 120 #resolution
matplotlib.rcParams['figure.figsize'] = (8,6) #figure size

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
color = sns.color_palette()

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

from sklearn.decomposition import PCA

root = 'C:/Data/instacart-market-basket-analysis/'
aisles = pd.read_csv(root + 'aisles.csv')
departments = pd.read_csv(root + 'departments.csv')
orders = pd.read_csv(root + 'orders.csv')
order_products_prior = pd.read_csv(root + 'order_products__prior.csv')
order_products_train = pd.read_csv(root + 'order_products__train.csv')
products = pd.read_csv(root + 'products.csv')
order_products = order_products_prior.merge(products, on ='product_id', how='left')
order_products = order_products.merge(aisles, on ='aisle_id', how='left')
order_products = order_products.merge(departments, on ='department_id', how='left')
order_products = order_products.merge(orders, on='order_id', how='left')
order_products.shape
order_products.head()
order_products.user_id.nunique()
cross_df = pd.crosstab(order_products.user_id, order_products.aisle)
cross_df.head()
cross_df.tail()
df = cross_df.div(cross_df.sum(axis=1), axis=0)
```

```
df.head()
df.shape
pca = PCA(n_components=10)
df_pca = pca.fit_transform(df)
df_pca = pd.DataFrame(df_pca)
df_pca.head()
Sum_of_squared_distances = []
K = range(1,10)
for k in K:
    km = KMeans(n_clusters=k)
    km = km.fit(df_pca)
    Sum_of_squared_distances.append(km.inertia_)

plt.subplots(figsize = (8, 5))
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
clusterer = KMeans(n_clusters=5,random_state=42).fit(df_pca)
centers = clusterer.cluster_centers_
c_preds = clusterer.predict(df_pca)
print(centers)
temp_df = df_pca.iloc[:, 0:2]
temp_df.columns = ["pc1", "pc2"]
temp_df['cluster'] = c_preds
temp_df.head()

fig, ax = plt.subplots(figsize = (8, 5))
ax = sns.scatterplot(data = temp_df, x = "pc1", y = "pc2", hue = "cluster")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_title("Cluster Visualization")
plt.show();
```

```
cross_df['cluster'] = c_preds

cluster1 = cross_df[cross_df.cluster == 0]
cluster2 = cross_df[cross_df.cluster == 1]
cluster3 = cross_df[cross_df.cluster == 2]
cluster4 = cross_df[cross_df.cluster == 3]
cluster5 = cross_df[cross_df.cluster == 4]
cluster1.shape
cluster1.drop('cluster',axis=1).mean().sort_values(ascending=False)[0:10]
cluster2.shape
cluster2.drop('cluster',axis=1).mean().sort_values(ascending=False)[0:10]
cluster3.shape
cluster3.drop('cluster',axis=1).mean().sort_values(ascending=False)[0:10]
cluster4.shape
cluster4.drop('cluster',axis=1).mean().sort_values(ascending=False)[0:10]
cluster5.shape
cluster5.drop('cluster',axis=1).mean().sort_values(ascending=False)[0:10]
```

Exploratory Data analytics

```
import numpy as np
import pandas as pd
pd.set_option('max_columns', 150)

# matplotlib and seaborn for plotting
import matplotlib
matplotlib.rcParams['figure.dpi'] = 120 #resolution
matplotlib.rcParams['figure.figsize'] = (8,6) #figure size
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
color = sns.color_palette()

root = 'C:/Data/instacart-market-basket-analysis/'
```

```
aisles = pd.read_csv(root + 'aisles.csv')
departments = pd.read_csv(root + 'departments.csv')
orders = pd.read_csv(root + 'orders.csv')
order_products_prior = pd.read_csv(root + 'order_products__prior.csv')
order_products_train = pd.read_csv(root + 'order_products__train.csv')
products = pd.read_csv(root + 'products.csv')
aisles.head()
departments.head()
products.head()
orders.head()
order_products_prior.head()
order_products_train.head()
order_products = order_products_prior.append(order_products_train)
order_products.shape
order_products = order_products.merge(products, on ='product_id', how='left')
order_products = order_products.merge(aisles, on ='aisle_id', how='left')
order_products = order_products.merge(departments, on ='department_id', how='left')
order_products = order_products.merge(orders, on='order_id', how='left')
order_products.shape
order_products.head()
order_products.tail()
order_products.info()
def reduce_memory(df):

    """
    This function reduce the dataframe memory usage by converting it's type for easier
handling.

    Parameters: Dataframe
    Return: Dataframe
    """

    start_mem_usg = df.memory_usage().sum() / 1024**2
    print("Memory usage of properties dataframe is :",start_mem_usg," MB")
```

```
for col in df.columns:
    if df[col].dtypes in ["int64", "int32", "int16"]:

        cmin = df[col].min()
        cmax = df[col].max()

        if cmin > np.iinfo(np.int8).min and cmax < np.iinfo(np.int8).max:
            df[col] = df[col].astype(np.int8)

        elif cmin > np.iinfo(np.int16).min and cmax < np.iinfo(np.int16).max:
            df[col] = df[col].astype(np.int16)

        elif cmin > np.iinfo(np.int32).min and cmax < np.iinfo(np.int32).max:
            df[col] = df[col].astype(np.int32)

    if df[col].dtypes in ["float64", "float32"]:

        cmin = df[col].min()
        cmax = df[col].max()

        if cmin > np.finfo(np.float16).min and cmax < np.finfo(np.float16).max:
            df[col] = df[col].astype(np.float16)

        elif cmin > np.finfo(np.float32).min and cmax < np.finfo(np.float32).max:
            df[col] = df[col].astype(np.float32)

print("")
print("___MEMORY USAGE AFTER COMPLETION:___")
mem_usg = df.memory_usage().sum() / 1024**2
print("Memory usage is: ",mem_usg," MB")
print("This is ",100*mem_usg/start_mem_usg,"% of the initial size")

return df
```

```
order_products = reduce_memory(order_products)
del products, orders, order_products_prior, order_products_train, aisles, departments,
reduce_memory, root
%whos
order_products.head()
temp_df = order_products.groupby("aisle")["reordered"].agg(['count',
'sum']).rename(columns = {'count':'total','sum':'reorders'})
temp_df = temp_df.sort_values('total', ascending=False).reset_index()
fig, ax = plt.subplots(figsize = (15,8))
ax = sns.barplot(y = temp_df.aisle[0:20], x = temp_df.total[0:20], color=color[0], label =
"total")
ax = sns.barplot(y = temp_df.aisle[0:20], x = temp_df.reorders[0:20], color=color[3], label =
"reordered")
ax.set_ylabel("Aisle")
ax.set_xlabel("Orders Count")
ax.set_title("Total Orders and Reorders From Most Popular Aisles")
ax.legend(loc = 4, prop={'size': 12})
plt.show()
temp_df["reorder_ratio"] = temp_df.reorders/temp_df.total
temp_df = temp_df.sort_values("reorder_ratio", ascending=False).reset_index()
fig, ax = plt.subplots(figsize = (13,8))
ax = sns.barplot(y = temp_df.aisle[0:20], x = temp_df.reorder_ratio[0:20], color=color[0])
ax.set_ylabel("Aisles")
ax.set_xlabel("Reorder Ratio")
ax.set_title("Aisles with Highest Reorder Ratio")
ax.tick_params(axis = 'both', labelsize = 12)
plt.show()
fig, ax = plt.subplots(figsize = (13,8))
ax = sns.barplot(y = temp_df.aisle[-21:], x = temp_df.reorder_ratio[-21:], color=color[0])
ax.set_ylabel("Aisles")
ax.set_xlabel("Reorder Ratio")
ax.set_title("Aisles with Lowest Reorder Ratio")
ax.tick_params(axis = 'both', labelsize = 12)
plt.show()
```

```
temp_df = order_products.groupby("department")["reordered"].agg(['count',
'sum']).rename(columns = {'count':'total','sum':'reorders'})
temp_df = temp_df.sort_values('total', ascending=False).reset_index()
fig, ax = plt.subplots(figsize = (15,8))
ax = sns.barplot(y = temp_df.department, x = temp_df["total"], color=color[0], label =
"total")
ax = sns.barplot(y = temp_df.department, x = temp_df["reorders"], color=color[3], label =
"reordered")
ax.set_ylabel("Department")
ax.set_xlabel("Frequency")
ax.legend(loc = 4, prop={'size': 12})
ax.set_title("Total Orders and Reorders From Departments")
plt.show()
temp_df["reorder_ratio"] = temp_df.reorders/temp_df.total
temp_df = temp_df.sort_values("reorder_ratio", ascending=False).reset_index()
fig, ax = plt.subplots(figsize = (13,8))
ax = sns.barplot(y = temp_df.department, x = temp_df.reorder_ratio, color=color[0])
ax.set_ylabel("Departments")
ax.set_xlabel("Reorder Ratio")
ax.set_title("Departments with Highest Reorder Ratio")
ax.tick_params(axis = 'both', labelsize = 12)
plt.show()
temp_df = order_products.groupby("product_name")["reordered"].agg(['count',
'sum']).rename(columns = {'count':'total','sum':'reorders'})
temp_df = temp_df.sort_values('total', ascending=False).reset_index()
fig, ax = plt.subplots(figsize = (10,7))
ax = sns.barplot(y = temp_df.product_name[0:20], x = temp_df.total[0:20], color=color[0],
label = "total")
ax = sns.barplot(y = temp_df.product_name[0:20], x = temp_df.reorders[0:20],
color=color[3], label = "reordered")
ax.set_ylabel("Product")
ax.set_xlabel("Total Orders")
ax.set_title("Most Popular Products")
ax.legend(loc = 4, prop={'size': 12})
```

```
plt.show()
temp_df["reorder_ratio"] = temp_df.reorders/temp_df.total
temp_df.sort_values("reorder_ratio", ascending=False).head(10)
product_unique_users =
order_products.groupby('product_name')['user_id'].nunique().reset_index().rename(columns=
{'user_id':'total_users'})
product_unique_users.sort_values('total_users', ascending = False).head(10)
product_unique_users = product_unique_users.merge(temp_df, on='product_name',
how='left')
product_unique_users.sort_values("reorder_ratio", ascending=False).head(20)
temp_df = product_unique_users.sort_values("total_users", ascending=False)
temp_df['cum_users'] = temp_df['total_users'].cumsum()
temp_df = temp_df.reset_index(drop=True)
temp_df.head()
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.lineplot(x = temp_df.index, y=temp_df.cum_users)
ax.set_xlabel("Products", size = 9)
ax.set_ylabel("Cumulative Sum of Unique Users", size = 9)
ax.set_title("Cumulative Sum of Unique Users Per Product", size = 12)
plt.show()
fig, ax = plt.subplots(figsize=(15,8))
ax = sns.scatterplot(y = product_unique_users.total, x = product_unique_users.total_users)
ax.set_xlabel("Product Buyers", size = 9)
ax.set_ylabel("Number of Product Purchased", size = 9)
ax.set_title("Total Product Orders VS Total Unique Product Buyers", size = 12)
plt.show()
fig, ax = plt.subplots(figsize=(10,5))
ax = sns.scatterplot(x = product_unique_users.total, y = product_unique_users.reorder_ratio,
color = color[3])
ax.set_xlabel("Number of Products Purchased")
ax.set_ylabel("Reorder Percentage")
ax.set_title("Reorder Percentage VS Total Orders")
plt.show()
fig, ax = plt.subplots(figsize=(10,5))
```

```
ax = sns.scatterplot(x = product_unique_users.total_users, y =
product_unique_users.reorder_ratio, color = color[0])
ax.set_xlabel("Total Unique Users")
ax.set_ylabel("Reorder Percentage")
ax.set_title("Reorder Percentage VS Total Unique Users")
plt.show()
product_unique_users['Organic'] =
product_unique_users.product_name.str.contains("Organic")
product_unique_users.head()
fig, ax = plt.subplots(figsize = (5,5))
ax = sns.barplot(x = product_unique_users.groupby('Organic').size().index, y =
product_unique_users.groupby('Organic').size().values)
ax.set_xlabel("Organic Product", size = 9)
ax.set_ylabel("Total Products", size = 9)
ax.set_title("Total Organic and Inorganic products", size = 10)
plt.show()
fig, ax = plt.subplots(figsize = (5,5))
ax = sns.barplot(x = product_unique_users.groupby('Organic')['reorder_ratio'].mean().index,
y = product_unique_users.groupby('Organic')['reorder_ratio'].mean().values)
ax.set_xlabel("Organic Product", size = 9)
ax.set_ylabel("Mean reorder ratio", size = 9)
ax.set_title("Mean Reorder Ratio of Organic/Inorganic Products", size = 10)
plt.show()
temp_df = order_products.groupby('add_to_cart_order')['reordered'].mean().reset_index()
temp_df.head()
fig, ax = plt.subplots(figsize=(13,6))
ax = sns.lineplot(x=temp_df.add_to_cart_order, y=temp_df.reordered, lw = 1, marker='o')
ax.set_xlabel("Add to Cart Order")
ax.set_ylabel("Reorder Ratio")
ax.set_title("Add to Cart Order VS Reorder Ratio")
plt.show()
temp_df = order_products.groupby(['order_dow',
'product_name']).size().reset_index(name='counts')
temp_df = temp_df.sort_values(['order_dow', 'counts'], ascending=[True, False])
temp_df = temp_df.groupby('order_dow').head(5).reset_index(drop = True)

ax = sns.catplot(x="order_dow", y="counts", hue="product_name", data=temp_df,
kind="bar", legend=False)
ax.add_legend(title="Product")
ax.set_axis_labels("Day of Week", "Total Orders of Most Frequent Products")
ax.fig.suptitle("Most Popular Products on different Days of Week", va="baseline",
ha="center")
ax.savefig("Most Popular Products on Different Days of Week.png")
```