



School of Computer Sciences
Universiti Sains Malaysia
Academic Session: Semester II, 2021/2022

Predictive Analysis Study of Product Data in Four Techniques

CDS513: Predictive Business Analytics

GROUP 8

Name:	Matric No:
NUR SHAHIRAH MAT AKHIR	P-COM0189/21
MUTHUK KUMARAN SUBRAMANIAM	P-COM0025/21
ZHAO DAN	P-COM0255/20
SABAH ANWAR AZMI	P-COM0024/21

Table of Contribution

	ZHAO DAN	SHAHIRAH	MUTHUK	SABAH
Report Component				
Abstract & Background	√	√		
Problem Statement	√	√		
Objectives & Motivation	√		√	
Data Description	√		√	
Scope & Reference	√			
Literature Review	√			
Data Preprocessing	√	√	√	√
Modelling	√	√	√	√
Experiment Result & Analysis	√	√	√	√
Conclusion & Future Research	√	√	√	√
Report Format & Neatness	√			
Poster Component				
Abstract & Background	√			
Data Description	√			
Problem Statement & Objectives	√			
Approaches	√	√	√	√
Experiments & Analysis	√	√	√	√
Findings & Conclusion	√	√	√	√
Poster Format & Neatness	√			

Table of Contents

Abstract.....	4
1. Introduction	4
❖ Domain Background.....	4
❖ Problem Statement & Research Problem	4
❖ Objective & Research Scope & Motivation	5
2. Literature review	5
3. Methodology	7
❖ Dataset.....	7
❖ Project Framework.....	8
4. Experiments & Analysis	10
❖ Experiments Setup.....	10
❖ Result Analysis	28
5. Future Research & Conclusion.....	32
References	34
Appendix.....	35
❖ Project Source Code	35
❖ Sample Output.....	35

Abstract:

Nowadays, people would like to purchase whatever they want from either online or offline retailers. Due to such demand from the clients, many players rushed into the retail industry and wish to gain profit from the industry. It gives a good impact on our economy; however, players can have problems with planning, overproduction, and hustling customer acquisition. This is not just startup companies' problem, even well-known companies face the same problem. Given the Amazon Product dataset, we performed MBA (Market Basket Association), Regression, and Time Series Analysis. Given the Ratings Electronics dataset, we performed RS (Recommendation System). By implementing the above four techniques, we expect to understand the behavior of customers, provide the right recommendation, check the relationship of different variables, and see the trends through time, so that retailers can have proper strategies for sales and other prospects

1. Introduction

❖ Problem Background

Retail industry, as a traditional industry, is actively integrated with internet technology, and huge amount of data are generated from daily sales activities. Take Amazon.com as an example, it is a listed globally active online mail order company with a wide range of products. According to Amazon, as the market leader in online trading, it has the world's largest selection of books, CDs and videos. The total sales of Amazon.com rise from 39,304 in 2014 to 162,360 million US dollars in 2022 (Statista, 2022). Therefore, the industry calls for solution to help them understand the product and sales data.

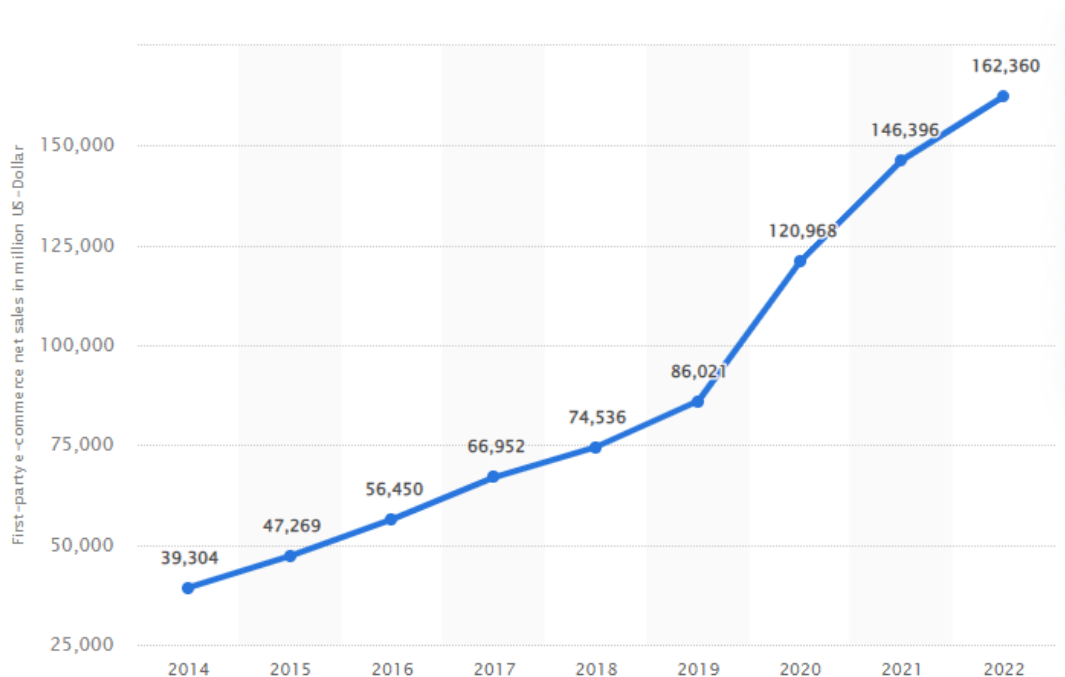


Figure 1. Net sales of Amazon.com from 2014 to 2022 (Statista, 2022)

❖ Problem Statement & Research Problem

➤ Problem Statement

Tremendous purchase data is generated every day in the Amazon.com, but lack of

analysis makes the data valueless and results in losing opportunities for revenue and growth. Because it is becoming increasingly difficult for retailers to gain meaningful insights from the huge amount of data. However, sales data analysis is crucial for any business entity because the insights generated from it can provide guidance for the future. Therefore, we wish to find a solution that cope well with big data volume and business application scenario.

➤ **Research Problems**

There are four problems we need to focus on,

- i. What products tend to be sold together?
- ii. How to recommend what customers want to them?
- iii. How to define the relationship between different variables?
- iv. What is the trend of the business over time?

❖ **Objective & Research Scope & Motivation**

➤ **Objectives**

According to the four research problems mentioned above, there are four objectives.

- i. To analyze what products tend to be sold together by implementing MBA (Market Basket Analysis) technique.
- ii. To find which product should be recommended to which customer by developing a recommendation system.
- iii. To find what factor is related to the target, e.g., sales, by performing regression analysis.
- iv. To understand the trend over time by implementing time series analysis.

➤ **Research Scope**

For dataset, the whole project is based on two datasets, 'Product Sales' and 'Ratings Electronics'. Therefore, other similar data will not be considered in this project. Besides, we focus on predictive analysis, any other types of analysis such as descriptive and prescriptive will be out of our bound, even if we can derive satisfied insights from those perspective. For purpose, we aim at understanding the basket of products, building a recommendation system, checking the variables relationship, and finding trends overtime. Finally, the techniques we use in the project are limited to market basket analysis, recommendation system, regression and time series analysis. Though other predictive techniques may also be able to drive insights from the data, we shall ignore other techniques.

➤ **Motivation**

The motivation of the project is to help Amazon.com to understand their product and sales data better, so that they can learn from the data and generate meaningful insights from it. Hopefully, this project can help Amazon.com to generate more profit, and earn customers' loyalty.

2. Literature review

➤ **Retail Industry**

Retail industry refers to business that sales product to customers directly. In retailing, understanding customer is essential to succeed. Retailers have provided customized shopping experiences by using predictive analytics to understand the drivers of profitability, loyalty and other activity for each customer panel and frame specific strategy for each segment (Sharma & Dadhich, 2014).

➤ **Market Basket Analysis**

Market Basket Analysis (MBA) refers to unsupervised machine learning techniques that could generate items which tend to be purchased together. MBA finds the repetitive items that are usually bought together by the customers in a large dataset. This helps the business to suggest the promotions of the products and the layout of the shops as well. (Isa et al, 2018) There are three machine learning algorithms available to do MBA. Apriori principle reduces the number of candidates itemset explored while generating frequent itemset with the help of support measure. FP-growth is an algorithm that generates frequent itemset from an FP-tree by traversing the FP-tree in a bottom-up approach. (Khan et al, 2017) Eclat, also known as Equivalence Class Clustering and Bottom-Up Lattice Traversal, is a vertical database layout algorithm used for mining frequent itemset.

➤ **Recommendation System**

The recommender system helps to understand the users and recommend suitable products based on other users' buying patterns and their browsing history. Dynamic group recommendation algorithm proposed the concept of activity level and quantified the importance of different members in group decision-making with it. Based on this, we weighted the preferences of each member according to their activity level and aggregated them to generate group recommendations by simulating the group decision-making process in real scenarios. (Jia, 2021) There are also works done using purchase data, automatic recommendation system was set up in three group mining algorithms in the experiments, so that group recommendation methods have experimented under different group sizes mined by different group mining algorithms to verify their robustness and the effectiveness of the group mining algorithms proposed to develop a recommendation system. (Bodapati, 2008)

➤ **Regression**

Regression refers to analysis that is done to find the relationship of the variables. There are many types of regression, for example, linear regression, logical regression, polynomial regression, ridge regression, and lasso regression. Ridge regression is used when data suffers from multicollinearity that is, independent variables are highly correlated. To compensate this problem, Ridge Regression, uses the shrinkage parameter, which uses the squares of coefficients to reduce the multicollinearity. Lasso regression differs from ridge regression in a way that it uses absolute values in the penalty function, instead of squares. This leads to penalizing values which causes some of the parameter estimates to turn out exactly zero. (Gupta, 2017)

➤ **Time Series Analysis**

Time Series Analysis refers to actions to explore the trends over time. The most common used technique in time series analysis is neural network. To adapt CNNs (convolutional neural network) to time series datasets, researchers utilize multiple layers of causal convolutions – i.e., convolutional filters designed to ensure only past information is used for forecasting. RNN is a popular solution in sequential modelling,

given the natural interpretation of time series data as sequences of inputs and targets, many RNN-based architectures have been developed for temporal forecasting applications. (Lim, & Zohren, 2021) Time series problem can also be conducted by statistics approach, ARIMA model perhaps is the most well-known one among all in statistics for time series analysis. It is a combination of autoregressive (AR) and moving average (MA) models, with a term of differencing (I) used to overcome the non-stationarity of some time series. (Mohamed, 2020)

3. Methodology

❖ Dataset

The main dataset, [Product Sales](#), is acquired from Kaggle, and it describes the sales data of Amazon in 2019. There are 12 subsets in the datasets and each subset represent a particular month in 2019. The data size of each subset can be seen in the table 1.

Table 1. Sub Dataset Overview of Main Dataset

Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Number of rows	9723	12036	15226	18383	16635	13622	14371	12011	11686	20379	17661	25117

Each subset shares the same attributes, meaning of the attributes, and the data type of the column. The info can be seen in Table 2.

Table 2. Main Dataset Description

Column name	Data type	Description	Example
<i>Order ID</i>	object	An Order ID is a unique number given to each order which enables Amazon to keep track of the order and for future reference in case there is dispute from the customer. This is a nominal attribute.	141234
<i>Product</i>	object	Name of the product that has been sold. This is a nominal attribute.	IPhone
<i>Quantity Ordered</i>	object	It is the total number of units of the product originally ordered under the order ID. This is a discrete attribute.	1
<i>Price Each</i>	object	The price of each product. This is a continuous attribute.	700
<i>Order Date</i>	object	This is the date and time when the customer placed an order. This is a continuous attribute.	01/22/19 21:25
<i>Purchase Address</i>	object	The address for product delivery. This is a nominal attribute.	944 Walnut St, Boston, MA 02215

The second dataset, [Ratings Electronic](#), comes from Kaggle too, and it is a part of Amazon Reviews data. This dataset can be linked to our main dataset because products in these two datasets are all electronic product, and they are all from Amazon.com. The dataset consists of 7,824,482 data rows, and the detail of this dataset is shown in table 3.

Table 3. Additional Dataset Description

Column name	Data type	Description	Example
User Id	object	User Unique ID	A2CX7LUOHB2NDG
Product Id	object	Product Unique ID	0321732944
Rating	object	Rating of the corresponding product by the corresponding user	5.0
Time stamp	object	Time of the ratings being made	1341100800

❖ Project Framework

➤ Market Basket Analysis

The tool used for the Market Basket Analysis is Python. The Apriori and FP growth approaches are used in this analysis.

- *Apriori*

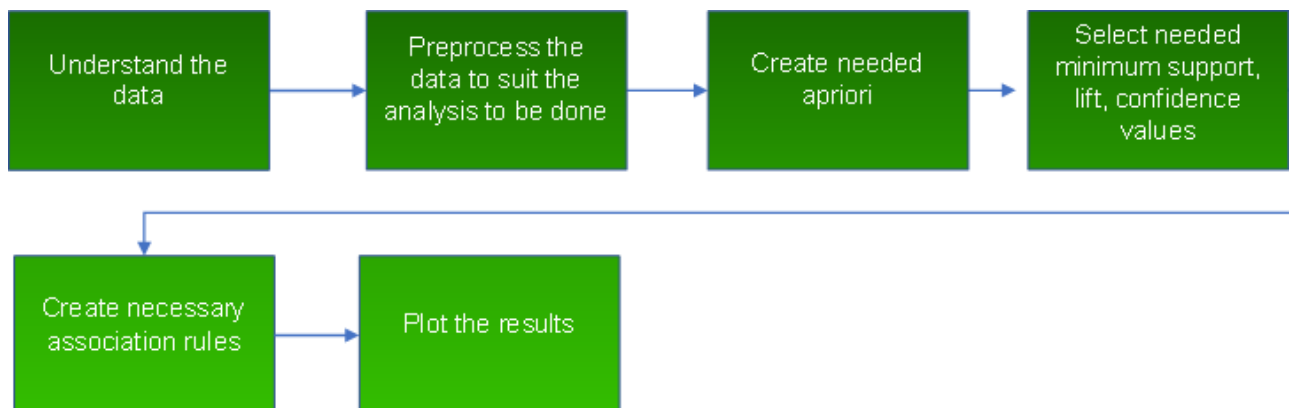


Figure 2: Apriori Workflow

- *FP Growth*

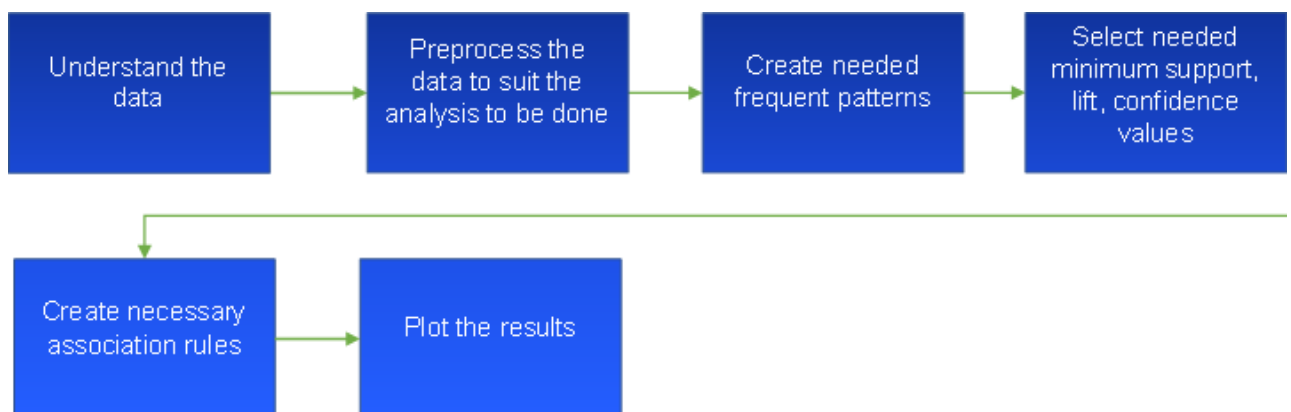


Figure 3: FP Growth Workflow

➤ Recommendation System

The tool used to implement recommendation system is Python and its libraries. There are two kinds of approach for recommendation system known as popularity based and collaborative based.

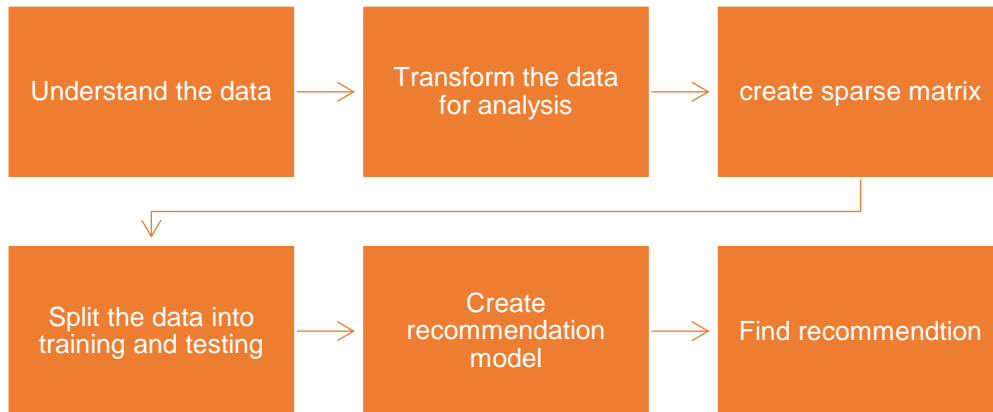


Figure 4: Recommendation Workflow

➤ Regression

This study uses Python and its packages and libraries to perform multiple linear regression of the sales dataset. This approach is used to predict the total sales based on the contributing attributes. Figure 4 shows the steps taken to perform the model prediction.

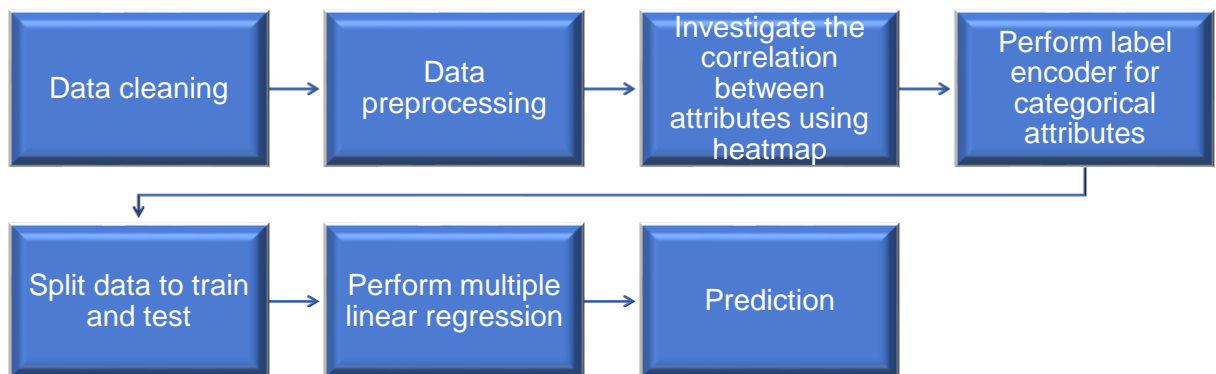


Figure 5: Multiple Linear Regression Workflow

➤ Time Series Analysis

The tool that is used to implement time series analysis is Python, and two kinds of methods, which are statistical approach and machine learning approach, have been included.

Statistics approach – ARIMA

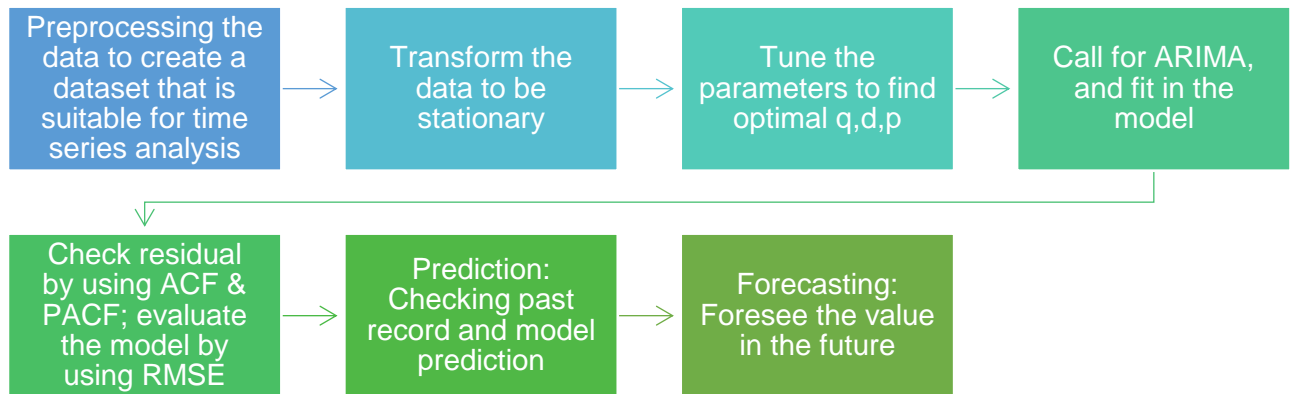


Figure 6: ARIMA Workflow

Machine Learning approach – Xgboost

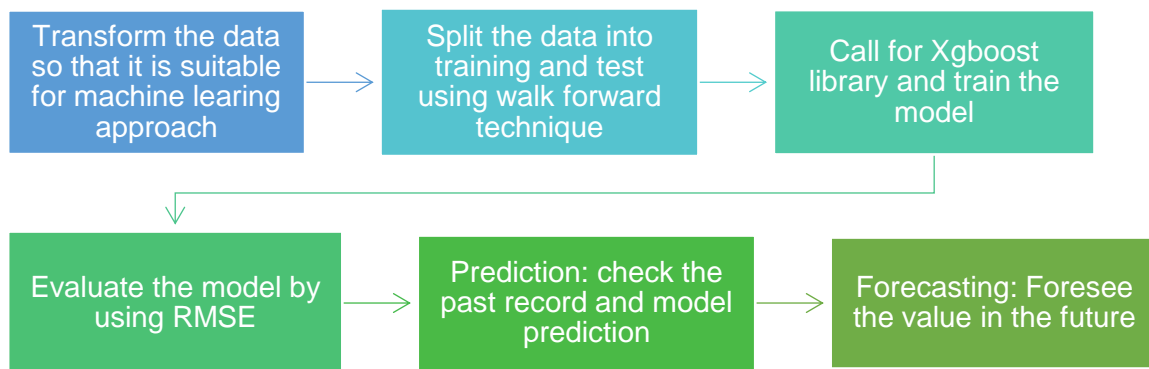


Figure 7: Xgboost Workflow

4. Experiments & Analysis

❖ Experiments Setup

➤ Market Basket Analysis (*Muthuk Kumaran Subramaniam*)

• Data Preprocessing

For the Market Basket Analysis, we used the Sales Dataset. There are 12 separate sales data files for each month in 2019. As a first step, all these 12 files were merged into a single CSV file. During the merger there is UserID column created based on the address column. Each address has a unique UserID. Also, there is a month column created to indicate which month the sales data belongs to. The Sales column was created by multiplying ordered quantities with its price.

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	User ID
0	176558	USB-C Charging Cable	2	11.95	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4	23.90	20320242233431656
1	176559	Bose SoundSport Headphones	1	99.99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4	99.99	3328378251253699731
2	176560	Google Phone	1	600.00	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	600.00	9004662985858720289
3	176560	Wired Headphones	1	11.99	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	11.99	9004662985858720289
4	176561	Wired Headphones	1	11.99	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4	11.99	7679727096119635556
...
185945	259353	AAA Batteries (4-pack)	3	2.99	2019-09-17 20:56:00	840 Highland St, Los Angeles, CA 90001	9	8.97	681895665040070616
185946	259354	iPhone	1	700.00	2019-09-01 16:00:00	216 Dogwood St, San Francisco, CA 94016	9	700.00	2726353450276452667
185947	259355	iPhone	1	700.00	2019-09-23 07:39:00	220 12th St, San Francisco, CA 94016	9	700.00	8627822551035339885
185948	259356	34in Ultrawide Monitor	1	379.99	2019-09-19 17:30:00	511 Forest St, San Francisco, CA 94016	9	379.99	279909385725152336
185949	259357	USB-C Charging Cable	1	11.95	2019-09-30 00:18:00	250 Meadow St, San Francisco, CA 94016	9	11.95	18758260738238176

185950 rows × 9 columns

Out of 185950 rows of data there are 178437 unique Order IDs. So, the Order ID can't be used to create the associations.

```
df['Order ID'].nunique()
```

178437

The same goes for the User ID.

```
df['User ID'].nunique()
```

140787

By further formatting the data, 3 new columns (City, State and Pincode) were created by pulling the information from the address column. Two more columns (Hour and DayTime) were created based on the time mentioned in the Order Date column. The Hour column contains every single sales time record while DayTime column's data divided into 4 categories, namely morning, afternoon, evening and night.

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Sales	User ID	City	State	Pincode	Hour	DayTime
0	176558	USB-C Charging Cable	2	11.95	2019-04-19 08:46:00	917 1st St, Dallas, TX 75001	4	23.90	20320242233431656	Dallas	TX	75001	8.46	Morning
1	176559	Bose SoundSport Headphones	1	99.99	2019-04-07 22:30:00	682 Chestnut St, Boston, MA 02215	4	99.99	3328378251253699731	Boston	MA	2215	22.30	Night
2	176560	Google Phone	1	600.00	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	600.00	9004662985858720289	Los Angeles	CA	90001	14.38	Afternoon
3	176560	Wired Headphones	1	11.99	2019-04-12 14:38:00	669 Spruce St, Los Angeles, CA 90001	4	11.99	9004662985858720289	Los Angeles	CA	90001	14.38	Afternoon
4	176561	Wired Headphones	1	11.99	2019-04-30 09:27:00	333 8th St, Los Angeles, CA 90001	4	11.99	7679727096119635556	Los Angeles	CA	90001	9.27	Morning

Only data from 4 columns were used in the analysis. Those are Product, Price Each, State and Hour.

Hour will be used as the index for the analysis as it has a decent number of unique values.

```
df['Hour'].nunique()
```

1440

- **Experiment steps- APRIORI**

Since the dataset is so huge, we split the analysis into multiple categories of state. The first selected state was New York.

```

basket = (df[df['State'] == 'NY']
          .groupby(['Hour', 'Product'])['Price Each']
          .sum().unstack().reset_index().fillna(0)
          .set_index('Hour'))

print(basket.head(25))

```

Product	20in Monitor	27in 4K Gaming Monitor	27in FHD Monitor \
Hour			
0.0	0.00	0.00	299.98
0.01	0.00	0.00	0.00
0.02	0.00	0.00	0.00
0.03	0.00	0.00	0.00
0.04	109.99	0.00	0.00
0.05	0.00	0.00	299.98
0.06	109.99	0.00	0.00
0.07	109.99	389.99	149.99
0.08	0.00	389.99	0.00
0.09	0.00	0.00	149.99
0.1	109.99	779.98	149.99
0.11	0.00	0.00	0.00

The frequent itemsets were created with a minimum support of 0.45

```

%%time
frequent_itemsets = apriori(basket_sets, min_support=0.45, use_colnames=True)
frequent_itemsets

```

Wall time: 8 ms

	support	itemsets
0	0.484397	(27in FHD Monitor)
1	0.743262	(AA Batteries (4-pack))
2	0.757447	(AAA Batteries (4-pack))
3	0.682270	(Apple AirPods Headphones)
4	0.640426	(Bose SoundSport Headphones)
...
64	0.466667	(USB-C Charging Cable, Lightning Charging Cabl...
65	0.451773	(Wired Headphones, Lightning Charging Cable, A...
66	0.490780	(USB-C Charging Cable, Wired Headphones, Light...
67	0.464539	(USB-C Charging Cable, Lightning Charging Cabl...
68	0.490071	(USB-C Charging Cable, Wired Headphones, Light...

69 rows × 2 columns

The rules were created with a Lift value above 1 and Confidence value above 0.8

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(AA Batteries (4-pack))	(AAA Batteries (4-pack))	0.743262	0.757447	0.624113	0.839695	1.108586	0.061132	1.513070
1	(AAA Batteries (4-pack))	(AA Batteries (4-pack))	0.757447	0.743262	0.624113	0.823970	1.108586	0.061132	1.458488
3	(Apple AirPods Headphones)	(AA Batteries (4-pack))	0.682270	0.743262	0.573759	0.840956	1.131439	0.066654	1.614259
4	(Bose SoundSport Headphones)	(AA Batteries (4-pack))	0.640426	0.743262	0.538298	0.840532	1.130868	0.062294	1.609959
6	(Lightning Charging Cable)	(AA Batteries (4-pack))	0.764539	0.743262	0.632624	0.827458	1.113279	0.064371	1.487974
...
349	(USB-C Charging Cable, Wired Headphones, AAA B...	(Lightning Charging Cable)	0.543972	0.764539	0.490071	0.900913	1.178374	0.074183	2.376297
350	(USB-C Charging Cable, Lightning Charging Cabl...	(Wired Headphones)	0.568085	0.713475	0.490071	0.862672	1.209112	0.084756	2.086422
351	(Wired Headphones, Lightning Charging Cable, A...	(USB-C Charging Cable)	0.539007	0.763121	0.490071	0.909211	1.191438	0.078744	2.609107
355	(Wired Headphones, Lightning Charging Cable)	(USB-C Charging Cable, AAA Batteries (4-pack))	0.607801	0.641135	0.490071	0.806301	1.257616	0.100388	1.852696
356	(Wired Headphones, AAA Batteries (4-pack))	(USB-C Charging Cable, Lightning Charging Cable)	0.607801	0.646809	0.490071	0.806301	1.246584	0.096940	1.823404

Then results were plotted for visual analysis in figure 8.

```
plt.figure(figsize=(50,50), dpi=100)
plt.scatter(ante1,conse1,s=1200,c='blue')

plt.title('New York Buying Pattern',fontsize=60)
plt.xlabel('Antecedents',fontsize=50)
plt.xticks(rotation=90,fontsize=30)
plt.ylabel('Consequences',fontsize=50)
plt.yticks(rotation=0,fontsize=30)
plt.savefig('ScatterPlot_NY.png', bbox_inches='tight', pad_inches=1, facecolor='w')
plt.show()
```

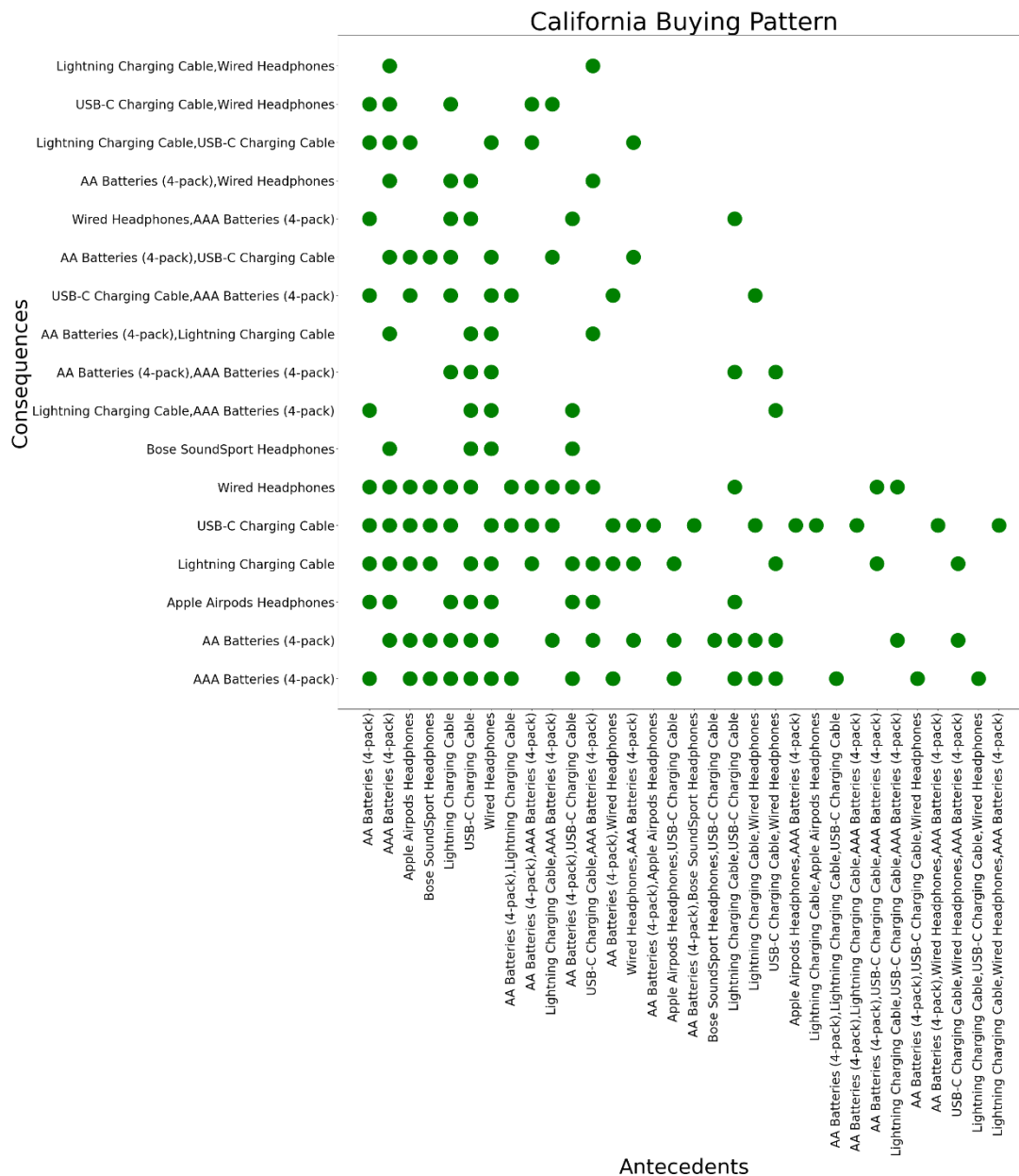



figure 9: California Buying Pattern



Figure 10: Massachusetts Buying Pattern

- Experiment steps- FP Growth**

The selected columns for FP-Growth are the Hour and Product. The selected data is already filtered for the state of Massachusetts. The TransactionEncoder library is used to transform the data.

```
from mlxtend.preprocessing import TransactionEncoder
```

```
te = TransactionEncoder()
te_ary = te.fit(df).transform(df)
df2 = pd.DataFrame(te_ary, columns=te.columns_)
df2
```


➤ Recommendation System (Sabah Anwar Azmi)

• Experiment steps

There were two datasets but Product Sales dataset is not compatible for recommendation system so we will be using Rating Electronics dataset in this section. The dataset contains four attributes in the Rating Electronics dataset namely, "User ID", "Product ID", "Rating", "Timestamp". The Timestamp attribute is not necessary for analysis so we will be removing/dropping it from dataset.

```
electronics.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7824482 entries, 0 to 7824481
Data columns (total 3 columns):
#   Column      Dtype
---  ---
0   userId      object
1   productId   object
2   rating      float64
dtypes: float64(1), object(2)
memory usage: 179.1+ MB
```

```
electronics.isna().sum()

userId      0
productId   0
rating      0
dtype: int64
```

From the above figure, we can see that the dataset contains 7,824,482 observations and 3 columns. The only numeric variable is **rating** and there is no missing value. The rating variable is very important for this project, and it is also numeric value. Figure 12 shows the distribution of ratings.

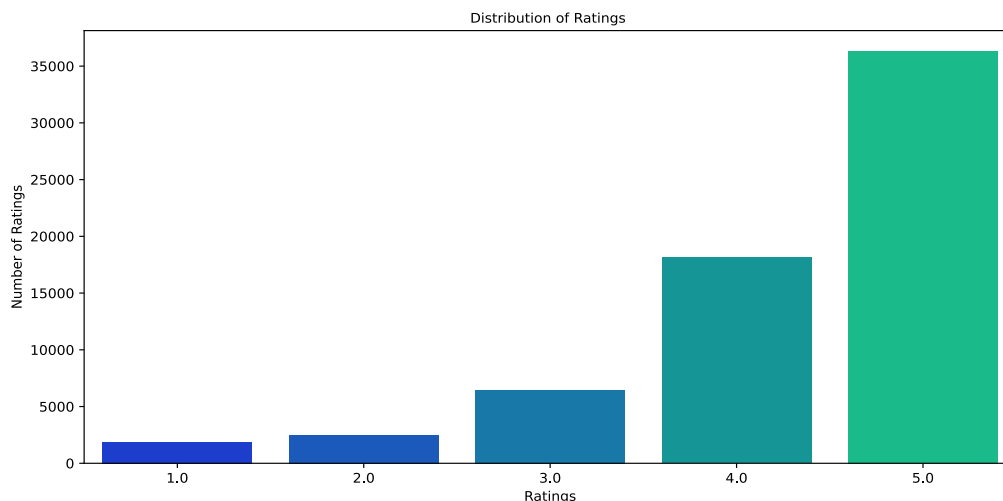


Figure 12: Ratings Distribution

The rating 5 has the highest count (4,347,541) followed by 4 (1,485,781), then followed by 1 (901,765), 3 (633,073), and lastly 2 (456,322). One sheer remark is that the distribution is heavily more skewed to the left (1, 2, and 3) than the right (4 and 5).

Before moving forwards, we need to restrict the data. Firstly, the data has 7824482 observations, thus too large for the computations required by the model. Second,

usually for rating datasets, there are many users who only rate a few products, and many products rated by very few users. This makes it possible to reduce the dataset using some logical assumptions. Thus, we will take users who have given at least 50 rating, and the products who has at least 5 rating.

First restriction

```
users = electronics.userId
ratings_count = dict()
for user in users:
    if user in ratings_count:
        ratings_count[user] += 1
    else:
        ratings_count[user] = 1
```

```
# users to have at least 50 ratings to be considred
RATINGS_CUTOFF = 50
remove_users = []
for user, num_ratings in ratings_count.items():
    if num_ratings < RATINGS_CUTOFF:
        remove_users.append(user)
electronics = electronics.loc[~electronics.userId.isin(remove_users)]
```

```
electronics.shape
```

```
(125871, 3)
```

Looking at the figure above, after the first round of restriction, the data now dropped from **7,824,482** to **125,871** observations. This is a huge size reduction. If we wanted the data size to be bigger than 125,871, we could set the cut-off to any number lower than 50, if we wanted a size lower than 125,871, we could increase the cut-off above 50.

Second restriction

```
users = electronics.productId
ratings_count = dict()
for user in users:
    if user in ratings_count:
        ratings_count[user] += 1
    else:
        ratings_count[user] = 1
```

```
# item to have at least 5 ratings to be considred
RATINGS_CUTOFF = 5
remove_users = []
for user, num_ratings in ratings_count.items():
    if num_ratings < RATINGS_CUTOFF:
        remove_users.append(user)
electronics = electronics.loc[~electronics.productId.isin(remove_users)]
```

```
electronics.shape
```

```
(65290, 3)
```

From the above figure. the observations dropped now to 65,290 after the second round of restriction. With this reasonable size, we think that the computations will be not taking longer to run.

```
# Finding number of unique users
print("Unique Users:",electronics['userId'].nunique())
# Finding number of unique products
print("Unique Products:", electronics['productId'].nunique())

Unique Users: 1540
Unique Products: 5689
```

The figure xx shows that the new data contains 1,540 unique users and 5,689 unique items. So, the total possible unique (user, item) compels is $1540 \times 5689 = 8761060$. This means not every user has rated every item in the dataset because there are only 65290 observations/ratings. I can then build recommendation systems to recommend items to users which they have not interacted with.

```
# minimum interaction of a user of an item
print("Minimum:", electronics.groupby(['userId', 'productId']).count()['rating'].min())
# maximum interaction of a user of an item
print("Maximum:",electronics.groupby(['userId', 'productId']).count()['rating'].max())

Minimum: 1
Maximum: 1
```

Both the minimum and the maximum interaction of a couple (user, item) is 1. This means that there is one and only one interaction between a pair (user, item).

- Recommendation System Setup

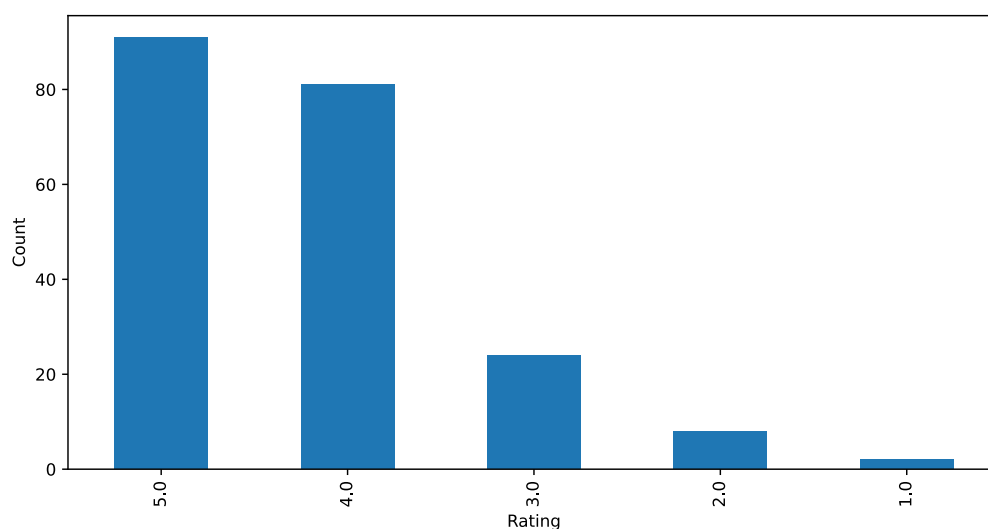


Figure 13: Ratings Count

With 295 interactions, user ADLVFFE4VBT8 has interacted with the greatest number of items. With 206 interactions in total, B0088CJT4U is the most interacted item by users. The figure above shows distribution of ratings of the item B0088CJT4U. The count of the ratings is decreasing from 5 to 1, with 5 and 4 being the highest. Thus, the item B0088CJT4U is liked by most users.

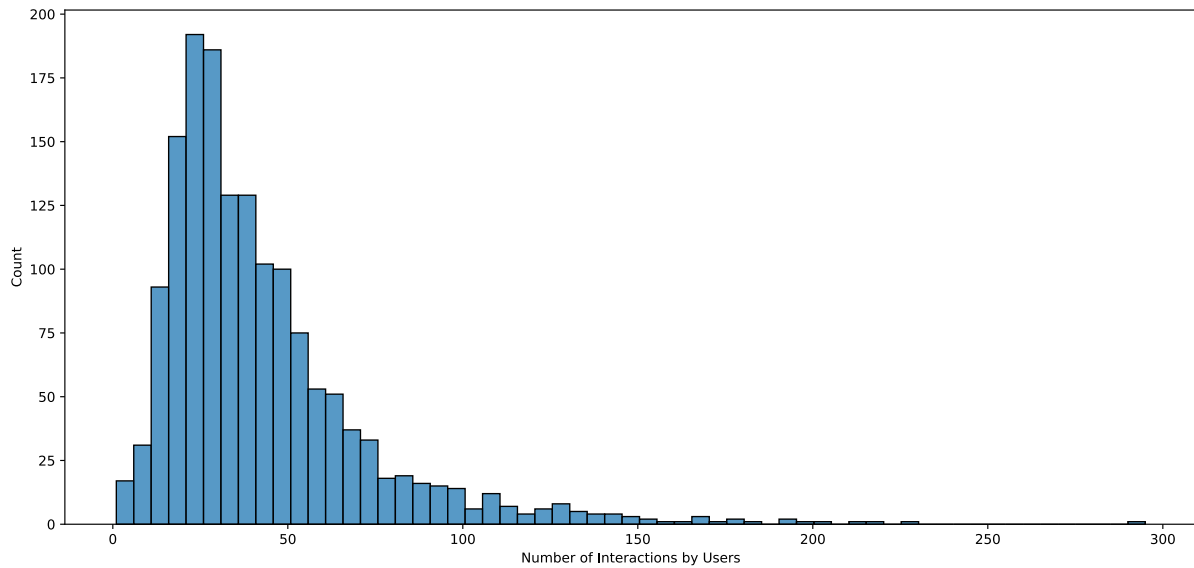


Figure 14: Interaction Counts

As we can see, in the figure above, the user-item distribution is heavily skewed to the right. This means that only a few users interacted with more than 50 items, which was the threshold we set earlier when restricting the data. This also means we have plenty room of recommending items to consumers who have not rated them before

➤ Regression (*Nur Shahirah Mat Akhir*)

• Data Preprocessing

Regression analysis will focus on the sales dataset in Section 3. There are six attributes in this dataset which are “*OrderID*”, “*Product*”, “*Quantity Ordered*”, “*Price Each*”, “*Order Date*” and “*Purchase Address*”. The dataset does not have total sales for each “*OrderID*” hence, additional attributes - “*Total Sales*” was calculated as the target variable for this analysis. From this dataset, further analysis was done to predict the total sales of this store by diving into the attributing factors.

Figure xx: Additional “*Total Sales*” column as the attribute target

```
df['Total Sales'] = df['Quantity Ordered']*df['Price Each']
df.head(2)
```

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Total Sales
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	23.90
1	176559	Bose SoundSport Headphones	1	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215	99.99

A heatmap was plotted as shown in figure 15 to find the correlated attributes to the target variable. Negative values show a negative correlation whereas positive values show a positive correlation between those attributes. Any attributes that gave a correlation value close to 0 will be ignored for the regression analysis as the correlation between the attributes and the target variable was too low.



Figure 15: Heatmap plot of the attributes

- **Experiment Steps -Regression**

From Figure 16, the “*OrderID*” shows no significant correlation to the Total Sales therefore, this attribute can be dropped before modelling our model. In contrast, “*Price Each*” shows a significant correlation to the “*Total Sales*” with a correlation value of 1. A graph of “*Total Sales*” vs “*Price Each*” is plotted in Figure 16 to analyse the correlation between the attributes in deep.

Figure 16: Scatter plot of Total Sales vs Price Each

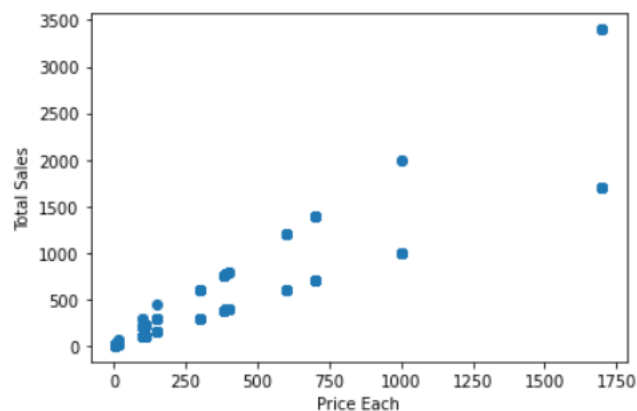


Figure 16: Scatter plot of Total Sales vs Price Each

After finding the suitable attributes from this dataset, unnecessary attributes such as “*OrderID*”, “*OrderDate*” and “*Purchase Address*” were dropped from this data set. As the product was a categorical attribute, it had to be encoded using LabelEncoder packages in Python as shown in Figure 17.

```

: from sklearn.preprocessing import LabelEncoder
df.apply(LabelEncoder().fit_transform)

```

	Product	Price Each	Total Sales
0	15	2	16
1	7	5	30
2	9	13	43
3	17	3	8
4	17	3	8
...
185945	5	0	4
185946	18	14	44
185947	18	14	44
185948	3	10	39
185949	15	2	6

185950 rows × 3 columns

Figure 17: LabelEncoder of the categorical variable

This dataset was split into the independent variables which are in X and the dependent variable, Y which was also known as the target variable. The data set then was split to train and test the data set for the prediction by 70% and 30% respectively before performing the linear regression.

Then, a plot of actual values vs fitted values was plotted to visualise the output of the modelling as shown in figure 18.

Figure 18: Comparison of actual and fitted value

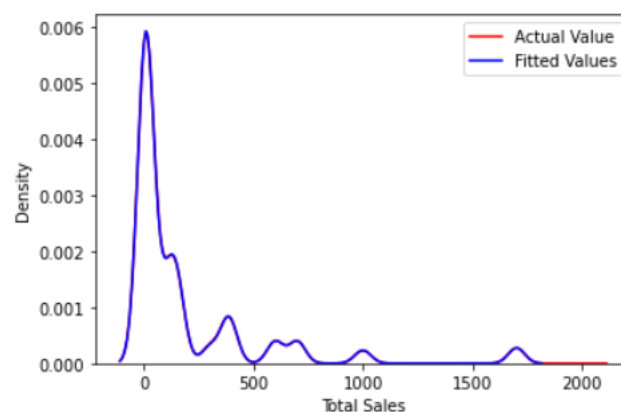


Figure 18: Comparison of actual and fitted value

➤ Time Series Analysis (*Zhao Dan*)

There were two datasets for this project, only 'Product Sales' dataset will be used in time series analysis experiment.

- **Data Preprocessing**

As the dataset consists of 12 sub datasets, the first step will be merging the 12 sub

datasets to one dataset. The result of this step is shown in figure 19.

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559	Bose SoundSport Headphones	1	99.99	04/07/19 22:30	682 Chestnut St, Boston, MA 02215
3	176560	Google Phone	1	600	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560	Wired Headphones	1	11.99	04/12/19 14:38	669 Spruce St, Los Angeles, CA 90001

Figure 19: Preprocessing Results

The second step is treating null value, creating two new columns which are named 'Date' and 'Sales', and drop unnecessary attributes. The dataset after treatment is shown in figure 20

	Date	Sales
0	2019-04-19	23.90
2	2019-04-07	99.99
3	2019-04-12	600.00
4	2019-04-12	11.99
5	2019-04-30	11.99

Figure 20: Data Preparation

After preprocessing, there are 185950 rows in the dataset.

- **Experiment steps-ARIMA**

As the dataset for now describe the sales amount of a single order, the first step of ARIMA is to generate daily total amount of sales. Therefore, I use 'groupby' function, and the dataset is shown in figure 21

	Sales
Date	
2019-01-01	65681.94
2019-01-02	70813.20
2019-01-03	47046.20
2019-01-04	62012.21
2019-01-05	46524.63

Figure 21: Data ready for analysis

The dataset is now ready for experiment.

The first step in ARIMA is to check the stationary of the data, and the plot can be seen from the figure 22. It is clear that the data is not stationary, because the line moved up and down.

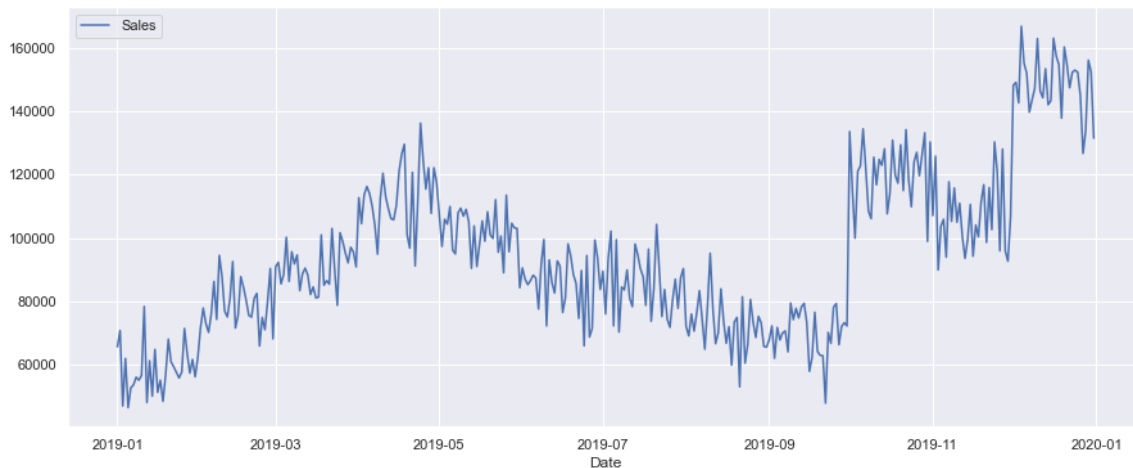


Figure 22: Data Stationary

Therefore, differencing is required. From figure 23, as the p-value of the dataset after one time differencing is less than 0.05, the dataset is stationary now.

```
check_stationarity(ts_Sales_log_diff)
```

```
The test statistic: -14.483573
p-value: 0.000000
Critical Values:
1%: -3.449
5%: -2.870
10%: -2.571
```

Figure 23: Check Stationary

After that, using auto ARIMA to find the optimal value of p, q and d.

```
Best model: ARIMA(0,1,1) (0,0,0) [0]
Total fit time: 4.137 seconds
```

The best model is ARIMA (0,1,1), and the set of parameters will be used to fit into the ARIMA model.

After fitting the data, I would like to know whether there are meaningful information remaining in the dataset, so I use ACF (Figure 24), and PACF (Figure 25) residual plot to check the residual.

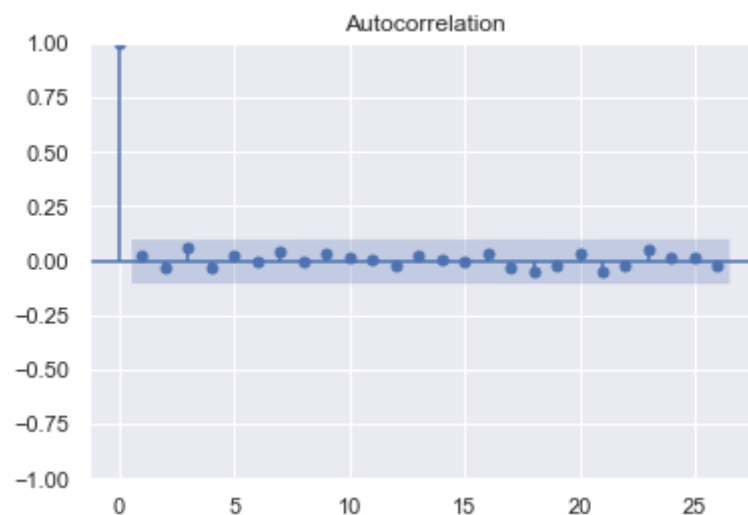


Figure 24: ACF Plot

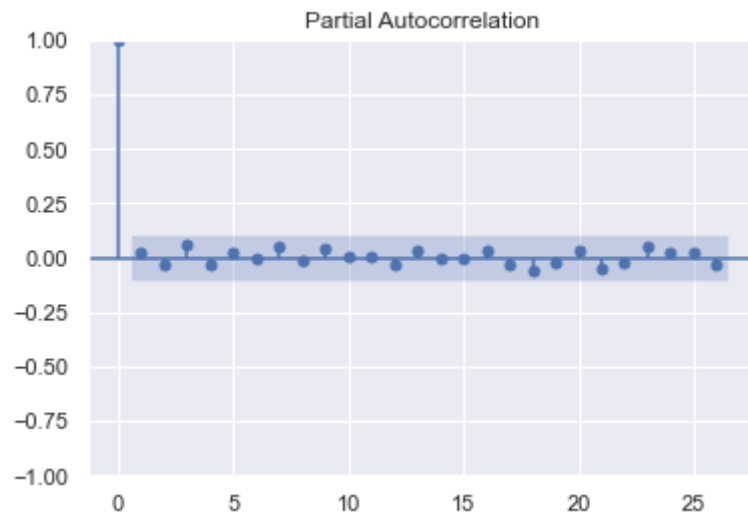


Figure 25: PACF Plot

Finally, check the prediction accuracy, forecast the daily sales of next month and calculate the evaluation Metrics.

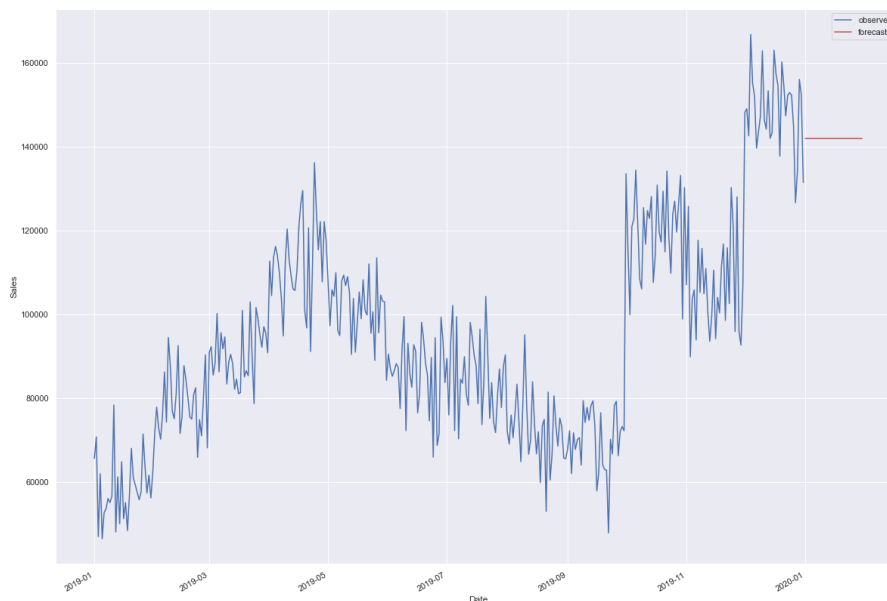


Figure 26: ARIMA (0,1,1) Forecast

However, the forecasting sales is constant, which could indicate the limited forecasting ability. Therefore, I choose to find the parameters manually.

I tried two times differencing, and the optimal parameter set is (0,0,1) from figure 26

Best model: ARIMA(0,0,1) (0,0,0) [0]

Total fit time: 1.469 seconds

However, I have already differenced the dataset for two times, therefore, I manually set the value of d as 2. Finally, the model should be ARIMA (0,2,1).

Figure 28 and Figure 29 show the ACF and PACF residual plot under this circumstance.

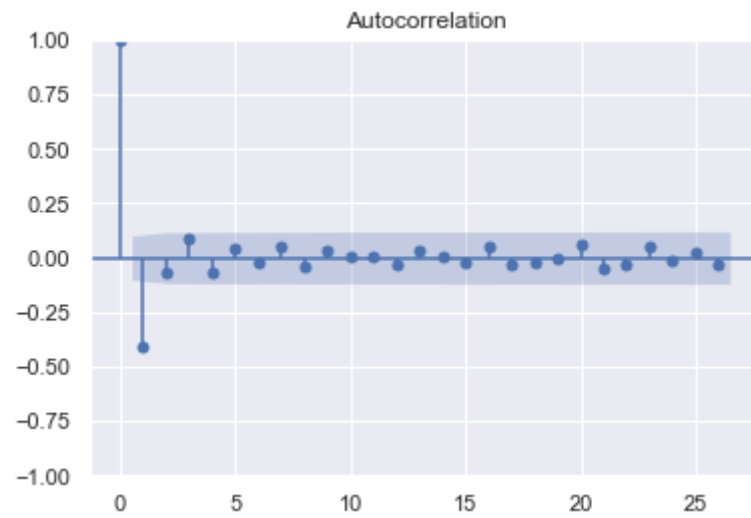


Figure 28: ACF Plot

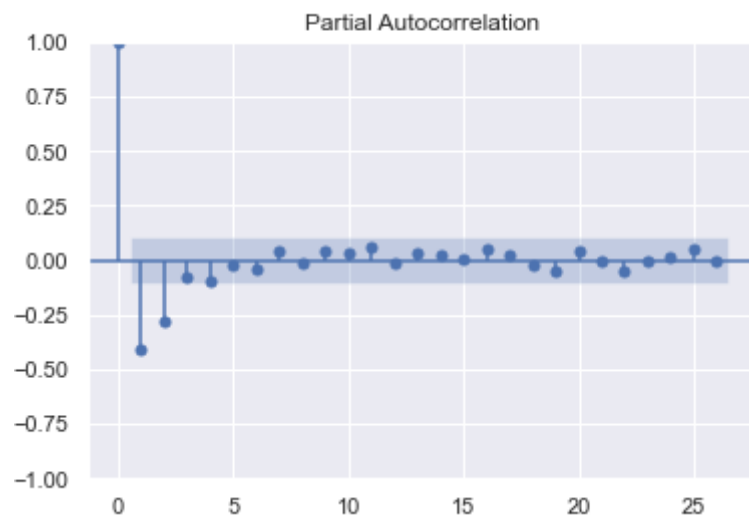


Figure 29: PACF Plot

Finally, the forecasting ability of the model is reflected from Figure 30

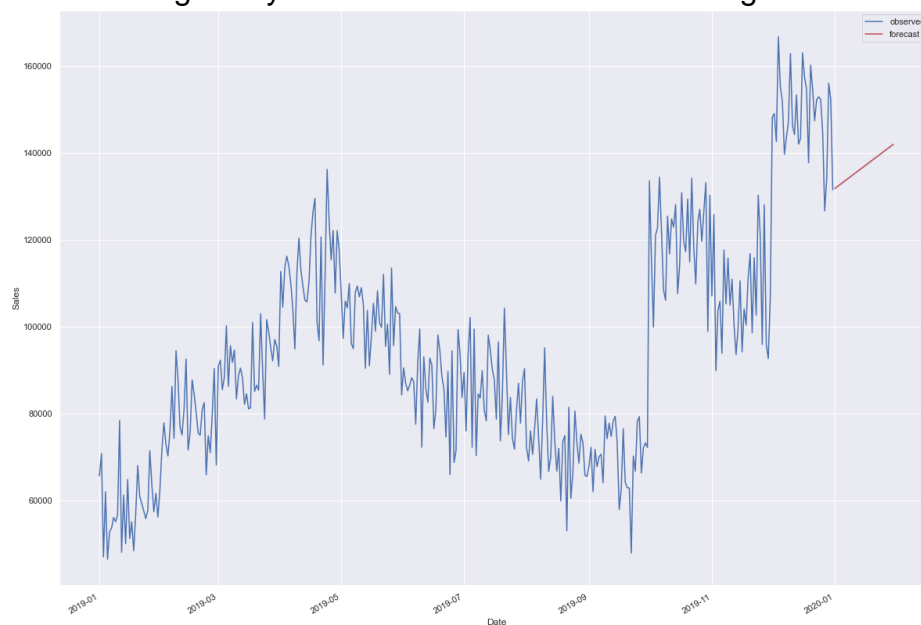


Figure 30: ARIMA (0,2,1) Forecast

- **Experiment steps-Xgboost**

The first step of setting up a Xgboost experiment is to transform the dataset which is suitable for machine learning problem. And then splitting the data by applying walk forward approach. The data after transformation is shown in Figure 31

```
array([[ 65681.94,  70813.2 ,  47046.2 , ...,  46524.63,  52777.49,
        53676.42],
       [ 70813.2 ,  47046.2 ,  62012.21, ...,  52777.49,  53676.42,
        56112.47],
       [ 47046.2 ,  62012.21,  46524.63, ...,  53676.42,  56112.47,
        55153.13],
       ...,
       [152214.45, 152888.82, 152268.42, ..., 126628.05, 134015.5 ,
        156024.62],
       [152888.82, 152268.42, 144912.02, ..., 134015.5 , 156024.62,
        152319.81],
       [152268.42, 144912.02, 126628.05, ..., 156024.62, 152319.81,
        131454.3 ]])
```

Figure 31: Data Transformation

And then, calculate the evaluation metrics and forecast the sales of the next month. The forecasting plot can be seen in Figure 32

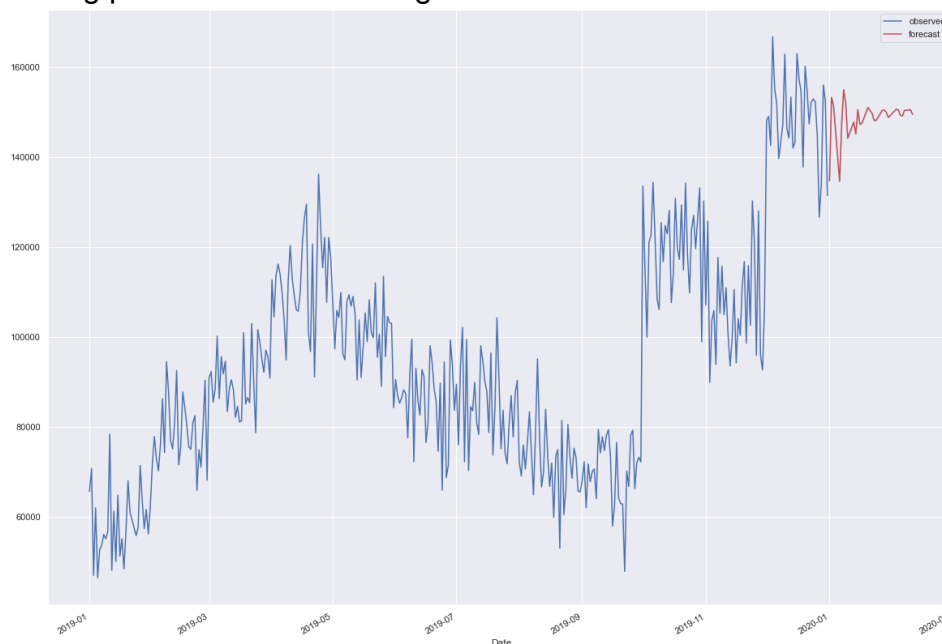


Figure 32: Xgboost Forecast

❖ Result Analysis

➤ Market Basket Analysis

Working with a huge dataset for Market Basket Analysis is cumbersome. The order id which is typically used as an index doesn't have any significant contribution to this analysis due to its uniqueness. The quantity parameter which is frequently used as a base for setting up the baskets also doesn't yield any good.

After trying multiple combinations of parameters, finally, the “Product”, “Price Each” and the “Hour” do yield some results. “Hour” parameter used the split the dataset as the way of transaction happening at a particular time. So, this Market Basket Analysis is all about what is being bought together at a particular time. The analysis is split into state levels to reduce the data handling by the algorithm and for easier visual presentation.

For Apriori, every state has different minimum support to produce the frequent itemsets. For instance, New York requires 0.45 as the minimum support, anything less than 0.8 for California yields too many itemsets, while Massachusetts requires 0.4.

The same goes for the confidence level. California was able to work with anything above 0.9, while New York and Massachusetts used values above 0.8.

For the FP Growth, the data was divided according to the state parameter as well. The items used for this analysis are the “Product” and “Hour” parameters. Doing this FP growth was supposed to be easier compared to Apriori, but it was the other way around as there is not sufficient documentation available explaining the method and how it could be done with Python. Plotting the FP-Tree was the toughest part of it as not a single source of it could be found on this topic. In the end, the same method as Apriori was to plot the graph.

The Apriori and FP Growth plots for the same state, Massachusetts, did show some differences in associations. This could be due to the different mechanics used in them. For FP Growth it could be because it is not expected to be presentable visually the same as Apriori.

➤ Recommendation System

Working with huge datasets requires more computation power, so we decided to add some restrictions and reduce our dataset so that we can perform the operations.

Popularity-based recommender system where we first calculated how many users have rated a product using python package for data manipulation. Prediction of the products was done based on popularity.

Model-based Collaborative Filtering is a personalized recommender system, the recommendations are based on the past behavior of the user, and it is not dependent on any additional information. The popularity-based recommender system is non-personalized, and the recommendations are based on frequency counts, which may be not suitable for the user.

First, Rank-based recommendation systems provide recommendations based on the most popular items. Useful to solve cold start problems, where new users get into the system and the machine is not able to recommend items to them due to the fact that they did not have any historical interactions in the dataset.

Second, the User-based RMSE value was 1.05 while the RMSE of the Item-based model was 1.06. Therefore, the User-based and the Item-based Collaborative Models performed nearly the same.

All the tuned Collaborative Filtering Models have performed better than the baseline models. Moreover, the user-user-based tuned model performed better with an RMSE of 0.9887

RMSE is calculated by the mean value of all the dissimilarity squared between the original rating and the predicted rating which then moves forward to compute the

square root of the result. Large errors might influence the RMSE rating, furnishing the RMSE metric most treasure when notable large errors are unwanted. The RMSE value between predicted ratings by the model and the original rating can be given by the formula:

RMSE

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (p_i - o_i)^2}{n}}$$

Where: p_i is the original rating, o_i is the predicted rating by the system and n is the number of ratings.

➤ Regression

It is important to choose the correct attributes for the regression analysis. Other than plotting a heatmap plot, the correlation can be determined by calculating the correlation coefficient as shown in equation 1.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \quad (1)$$

Where the r is the correlation coefficient, x is the value of the independent variable whereas the y is the value of the dependent variable or also known as the target variable.

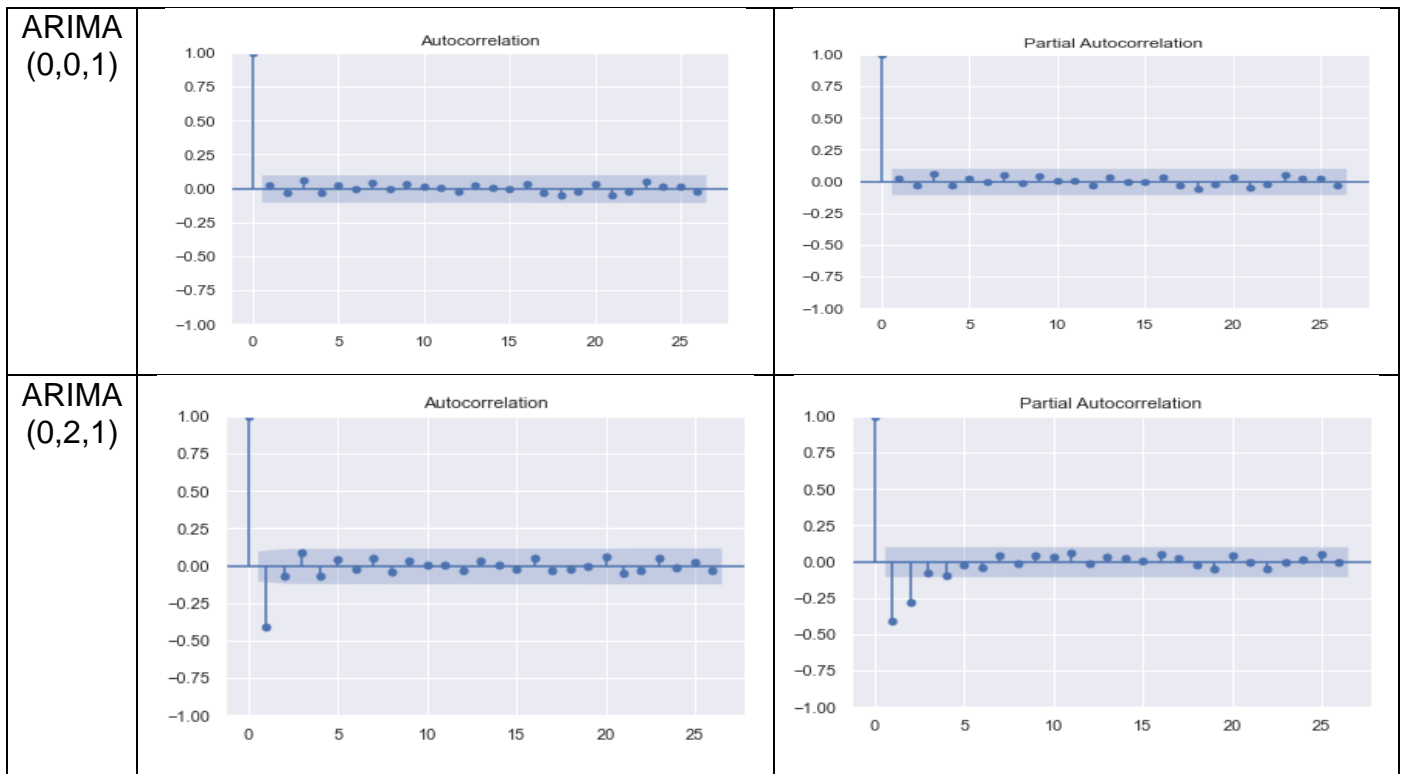
To confirm a reliable result from the regression analysis, an R-squared error was computed from the model. An R-squared error of 0.999 was obtained which is reliable as it is close to 1. This means that the regression model covers most of the variance of the values of the target variables which also means it is a good model.

➤ Time Series Analysis

For the residual analysis in ARIMA models, the results can be seen in table 4. ACF demonstrates the residual status of the MA (p) model, and PACF demonstrates the residual status of AR (q). For ARIMA (0,1,1), we could see that the only bar that exceeds the threshold boundary is 0, which means that no significant residual remains. The residual is safe to be considered as white noise. However, the model forecast result is constant. Therefore, I considered tuning the parameters manually, and the new model is ARIMA (0,2,1). However, there are one more bar exceeds the threshold in the ACF plot, and two more bars exceed the threshold in the PACF plot. This is a sign that ARIMA (0,1,1) is better than ARIMA (0,2,1).

Table 4: Residual Analysis Comparison

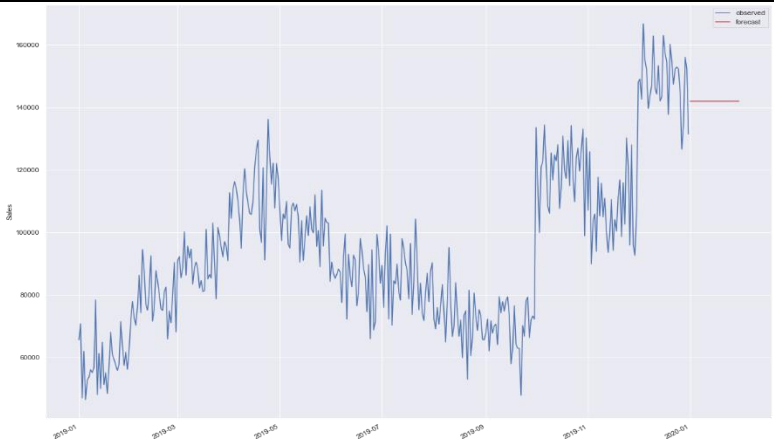
Model Name	ACF Residual Plot	PACF Residual Plot
------------	-------------------	--------------------





From table 5, It is clear that the forecasting result of ARIMA (0,0,1) is constant, while the forecasting result of ARIMA (0,2,1) is moving upward at a constant speed after a second time differencing. Both situations are abnormal in time series analysis, the reason for this is that the dataset has no obvious trend so the model is unable to forecast future sales based on the data given.

When it comes to the machine learning model, as Xgboost forecasts future values by training the past data, it is not necessary to detect time trends to forecast future values. Therefore, an obvious fluctuation can be witnessed in the XGBRegressor forecast.

Table 5: Forecasting Comparison

Model Name	Forecasting Output	RMSE
ARIMA (0,0,1)		11496.5

ARIMA (0,2,1)		13890.2
XGBRegressor		10723.6

Regardless of the pattern of the forecasting value, we will choose the model based on the evaluation metrics (RMSE). The lower the RMSE, the better the model performs. In the ARIMA model, ARIMA (0,0,1) has a lower value of RMSE than ARIMA (0,2,1), which reaches the same conclusion with residual analysis. Therefore, the best model is XGBRegressor model, because it has the lowest RMSE value (10723.6).

5. Future Research & Conclusion

Market Basket Analysis is not good in handling huge datasets as high computing processing power is required and the outcome of the analysis could not be visually presentable.

The regression analysis gives a good value of R-squared, however, there should be more inputs that are correlated to the target variable to avoid any bias in our result. A feature selection such as Dimensionality Reduction would be helpful to identify and select subset input variables that are most relevant to the target variable.

The Collaborative model is a user/item-tailored model that uses the user-item-ratings data to find similarities and make predictions rather than just predicting a random rating based on the distribution of the data. This could be the reason why the Collaborative Filtering performed well. In the future, for enhancing more, the evaluation of the model "timestamp" column that was discarded can be made in use so that it can be checked whether the users have purchased the product or not. Some more models can be applied for new enhancements.

For time series analysis, ARIMA is based on the trend of the time series data, so if there is no obvious trend in the data, ARIMA is unable to perform well. However, for the Xgboost approach, it is based on training the past data, so it can perform well based on the pattern in the past. For our dataset, as there is no obvious trend in the data, the machine learning model performs better than ARIMA. Based on what was revealed from the experiment, there are some future works that can be done. Firstly, more data should be collected, because our dataset includes only one year of data, this may not be enough for time series analysis. If the trend is cyclical, the range of cyclical trend may exceed one year, and thus the pattern will be missed out if only one year of data is available. Secondly, for ARIMA models, we can try SARIMAX or ARIMAX in the future. SARIMAX includes seasonal and external factors, and ARIMAX includes external factors. This means that extra information about the time series can be included in the model, and potentially, the model will perform better.

Overall, the four predictive analysis techniques we successfully implemented in the project can help Amazon.com to understand what items tend to be purchased together; what to recommend to which customers; understand the relationship between different factors; show the trend of the business over time.

References

- Statista, (2022). E-Commerce net sales of amazon.com from 2014 to 2022. Retrieve from: <https://www.statista.com/forecasts/1218313/amazon-revenue-development-ecommercedb>
- Sharma, N., & Dadhich, M. (2014). Predictive business analytics: The way ahead. *Journal of Commerce and Management Thought*.
- Isa, N., Kamaruzzaman, N. A., Ramlan, M. A., Mohamed, N., & Puteh, M., (2018). "Market basket analysis of customer buying patterns at corm café". *Int. J. Eng. Technol*, vol 7, p. 119-123, 2018.
- Khan, M. A., Solaiman, K. M., & Pritom, T. H. (2017). *Market basket Analysis for improving the effectiveness of marketing and sales using Apriori, FP Growth and Eclat Algorithm* (Doctoral dissertation, BRAC Univeristy).
- Jia, J., Yao, Y., Lei, Z., & Liu, P. (2021). Dynamic Group Recommendation Algorithm Based on Member Activity Level. *Scientific Programming*, 2021.
- Bodapati, A. V. (2008). Recommendation systems with purchase data. *Journal of marketing research*, 45(1), 77-93.
- Gupta, A., Sharma, A., & Goel, A. (2017). Review of regression analysis models. *Int. J. Eng. Res. Technol*, 6(08), 58-61.
- Lim, B., & Zohren, S. (2021). Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194), 20200209.
- Mohamed Belaid, Stéphane Lecoecue, Anthony Fleury, Baptiste Herve, (2020). Improvement of sales prediction by fitted-to-product time-series models. *MOSIM'20 – November 12-14, 2020 - Agadir – Morocco*
- Recall and Precision at k for Recommender Systems. Retrieve from https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

Appendix

❖ Project Source Code

Please find the source code [here](#).

❖ Sample Output

Market Basket Analysis:

Apriori

```
In [11]: basket = (df[df['State'] == 'NY']
                .groupby(['Hour', 'Product'])['Price Each']
                .sum().unstack().reset_index().fillna(0)
                .set_index('Hour'))

print(basket.head(25))
```

Product	20in Monitor	27in 4K Gaming Monitor	27in FHD Monitor
Hour			
0.0	0.00	0.00	299.98
0.01	0.00	0.00	0.00
0.02	0.00	0.00	0.00
0.03	0.00	0.00	0.00
0.04	109.99	0.00	0.00
0.05	0.00	0.00	299.98
0.06	109.99	0.00	0.00
0.07	109.99	389.99	149.99
0.08	0.00	389.99	0.00
0.09	0.00	0.00	149.99
0.1	109.99	779.98	149.99
0.11	0.00	0.00	0.00
0.12	0.00	0.00	0.00
0.13	0.00	389.99	0.00
0.14	0.00	389.99	149.99
0.15	0.00	389.99	0.00
0.16	0.00	0.00	0.00

```
In [12]: def encode_units(x):
        if x <= 0:
            return 0
        if x >= 1:
            return 1

basket_sets = basket.applymap(encode_units)
```

```
In [13]: basket_sets.head(25)
```

Out[13]:

Product	20in Monitor	27in 4K Gaming Monitor	27in FHD Monitor	34in Ultrawide Monitor	AA Batteries (4-pack)	AAA Batteries (4-pack)	Apple AirPods Headphones	Bose SoundSport Headphones	Flatscreen TV	Google Phone	LG Dryer	LG Washing Machine	Lightning Charging Cable	Macbook Pro Laptop
Hour														
0.0	0	0	1	0	0	1	1	1	0	0	0	0	0	1
0.01	0	0	0	0	1	1	0	0	0	0	0	0	1	1
0.02	0	0	0	0	1	1	1	0	1	1	0	0	1	0
0.03	0	0	0	1	1	0	1	0	1	0	0	0	1	1
0.04	1	0	0	0	0	1	0	1	0	0	0	0	1	0
0.05	0	0	1	1	0	1	0	1	1	0	0	0	0	0
0.06	1	0	0	1	1	0	1	1	0	1	0	0	1	0
0.07	1	1	1	0	0	1	0	1	0	0	0	0	1	0
0.08	0	1	0	0	1	0	1	0	0	1	0	0	1	0

```
In [14]: %%time
frequent_itemsets = apriori(basket_sets, min_support=0.45, use_colnames=True)
frequent_itemsets
```

```
In [14]: %%time
frequent_itemsets = apriori(basket_sets, min_support=0.45, use_colnames=True)
frequent_itemsets
```

Wall time: 10 ms

```
Out[14]:
```

	support	itemsets
0	0.484397	(27in FHD Monitor)
1	0.743262	(AA Batteries (4-pack))
2	0.757447	(AAA Batteries (4-pack))
3	0.682270	(Apple AirPods Headphones)
4	0.640426	(Bose SoundSport Headphones)
...
64	0.466667	(AA Batteries (4-pack), Apple AirPods Headphon...
65	0.451773	(AA Batteries (4-pack), Wired Headphones, Ligh...
66	0.490780	(AA Batteries (4-pack), Wired Headphones, Ligh...
67	0.464539	(Apple AirPods Headphones, AAA Batteries (4-pa...
68	0.490071	(USB-C Charging Cable, Wired Headphones, Light...

69 rows × 2 columns

```
In [15]: rules1 = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
print(rules1)
```

	antecedents \	
0	(AA Batteries (4-pack))	
1	(AAA Batteries (4-pack))	
2	(AA Batteries (4-pack))	
3	(Apple AirPods Headphones)	
4	(AA Batteries (4-pack))	
..	...	
357	(AAA Batteries (4-pack), Lightning Charging Ca...	
358	(USB-C Charging Cable)	
359	(Wired Headphones)	
360	(Lightning Charging Cable)	
361	(AAA Batteries (4-pack))	

	consequents	antecedent support \
0	(AAA Batteries (4-pack))	0.743262
1	(AA Batteries (4-pack))	0.757447
2	(Apple AirPods Headphones)	0.743262
3	(AA Batteries (4-pack))	0.682270
4	(Bose SoundSport Headphones)	0.743262
..
357	(USB-C Charging Cable, Wired Headphones)	0.640426
358	(Wired Headphones, Lightning Charging Cable, A...	0.763121
359	(USB-C Charging Cable, AAA Batteries (4-pack),...	0.713475
360	(USB-C Charging Cable, Wired Headphones, AAA B...	0.764539
361	(USB-C Charging Cable, Wired Headphones, Light...	0.757447

	consequent support	support	confidence	lift	leverage	conviction
0	0.757447	0.624113	0.839695	1.108586	0.061132	1.513070
1	0.743262	0.624113	0.823970	1.108586	0.061132	1.458488
2	0.682270	0.573759	0.771947	1.131439	0.066654	1.393228
3	0.743262	0.573759	0.840956	1.131439	0.066654	1.614259
4	0.640426	0.538298	0.724237	1.130868	0.062294	1.303924
..
357	0.612766	0.490071	0.765227	1.248808	0.097640	1.649398
358	0.539007	0.490071	0.642193	1.191438	0.078744	1.288385
359	0.568085	0.490071	0.686879	1.209112	0.084756	1.379385
360	0.543972	0.490071	0.641002	1.178374	0.074183	1.270281
361	0.542553	0.490071	0.647004	1.192517	0.079116	1.295897

[362 rows x 9 columns]

```
In [16]: rules1 = rules1[ (rules1['lift']>=1) &
                        (rules1['confidence']>=0.8)]
rules1
```

Out[16]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(AA Batteries (4-pack))	(AAA Batteries (4-pack))	0.743262	0.757447	0.624113	0.839695	1.108586	0.061132	1.513070
1	(AAA Batteries (4-pack))	(AA Batteries (4-pack))	0.757447	0.743262	0.624113	0.823970	1.108586	0.061132	1.458488
3	(Apple AirPods Headphones)	(AA Batteries (4-pack))	0.682270	0.743262	0.573759	0.840956	1.131439	0.066654	1.614259
5	(Bose SoundSport Headphones)	(AA Batteries (4-pack))	0.640426	0.743262	0.538298	0.840532	1.130868	0.062294	1.609959
6	(AA Batteries (4-pack))	(Lightning Charging Cable)	0.743262	0.764539	0.632624	0.851145	1.113279	0.064371	1.581815
...
349	(USB-C Charging Cable, Wired Headphones, AAA B...	(Lightning Charging Cable)	0.543972	0.764539	0.490071	0.900913	1.178374	0.074183	2.376297
350	(USB-C Charging Cable, AAA Batteries (4-pack),...	(Wired Headphones)	0.568085	0.713475	0.490071	0.862672	1.209112	0.084756	2.086422
351	(Wired Headphones, Lightning Charging Cable, A...	(USB-C Charging Cable)	0.539007	0.763121	0.490071	0.909211	1.191438	0.078744	2.609107
355	(Wired Headphones, Lightning Charging Cable)	(USB-C Charging Cable, AAA Batteries (4-pack))	0.607801	0.641135	0.490071	0.806301	1.257616	0.100388	1.852696
356	(Wired Headphones, AAA Batteries (4-pack))	(USB-C Charging Cable, Lightning Charging Cable)	0.607801	0.646809	0.490071	0.806301	1.246584	0.096940	1.823404

169 rows × 9 columns

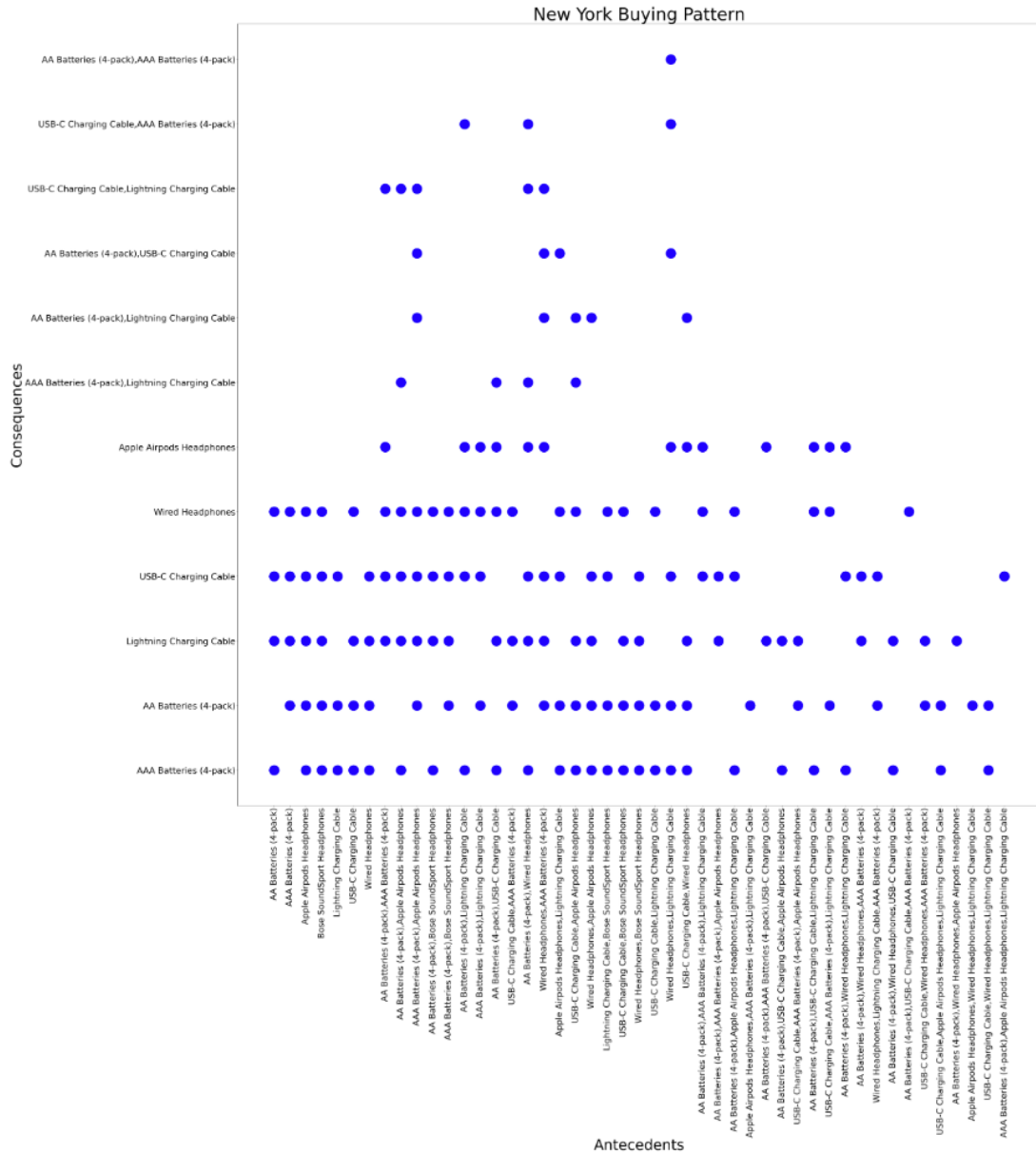
```
In [17]: rules1["antecedents"] = rules1["antecedents"].apply(lambda x: ','.join(list(x)).astype("unicode"))
rules1["consequents"] = rules1["consequents"].apply(lambda x: ','.join(list(x)).astype("unicode"))
```

```
In [18]: confidence = rules1['confidence']
lift1 = rules1['lift']
antel = []
for i in rules1["antecedents"]:
    antel.append(i)

consel = []
for i in rules1["consequents"]:
    consel.append(i)
```

```
In [19]: plt.figure(figsize=(50,50), dpi=100)
plt.scatter(antel,consel,s=1200,c='blue')

plt.title('New York Buying Pattern',fontsize=60)
plt.xlabel('Antecedents',fontsize=50)
plt.xticks(rotation=90,fontsize=30)
plt.ylabel('Consequences',fontsize=50)
plt.yticks(rotation=0,fontsize=30)
plt.savefig('ScatterPlot_NY.png', bbox_inches='tight', pad_inches=1, facecolor='w')
plt.show()
```



FP- Growth

```
In [36]: from mlxtend.frequent_patterns import fpgrowth
from mlxtend.preprocessing import TransactionEncoder
import pyfpgrowth
```

```
In [37]: df_fp = pd.read_csv("./fp.csv", encoding='unicode_escape')
df_fp.head()
```

```
Out[37]:
```

	20in Monitor	27in 4K Gaming Monitor	27in FHD Monitor	34in Ultrawide Monitor	AA Batteries (4-pack)	AAA Batteries (4-pack)	Apple AirPods Headphones	Bose SoundSport Headphones	Flatscreen TV	Google Phone	LG Dryer	LG Washing Machine	Lightning Charging Cable	Macbook Pro Laptop	ThinkPad Laptop	C
0	True	True	False	True	False	True	True	False	False	False	False	False	True	False	False	
1	False	False	True	False	True	True	False	False	False	False	False	False	False	False	False	
2	True	False	True	False	False	False	True	False	False	False	True	False	False	True	False	
3	False	False	False	False	True	True	True	True	False	False	False	False	False	False	False	
4	True	False	False	False	False	True	True	True	False	False	False	False	True	False	True	

```
In [38]: df_fp.columns
```

```
Out[38]: Index(['20in Monitor', '27in 4K Gaming Monitor', '27in FHD Monitor',
               '34in Ultrawide Monitor', 'AA Batteries (4-pack)',
               'AAA Batteries (4-pack)', 'Apple AirPods Headphones',
               'Bose SoundSport Headphones', 'Flatscreen TV', 'Google Phone',
               'LG Dryer', 'LG Washing Machine', 'Lightning Charging Cable',
               'Macbook Pro Laptop', 'ThinkPad Laptop', 'USB-C Charging Cable',
               'Vareebadd Phone', 'Wired Headphones', 'iPhone'],
              dtype='object')
```

```
In [39]: from mlxtend.frequent_patterns import fpgrowth

res = fpgrowth(df_fp, min_support=0.4, use_colnames=True)
res
```

Out[39]:

	support	itemsets
0	0.719311	(Lightning Charging Cable)
1	0.718593	(USB-C Charging Cable)
2	0.707825	(AAA Batteries (4-pack))
3	0.685571	(Wired Headphones)
4	0.620962	(Apple AirPods Headphones)
5	0.687006	(AA Batteries (4-pack))
6	0.400574	(27in FHD Monitor)
7	0.577172	(Bose SoundSport Headphones)
8	0.575018	(USB-C Charging Cable, Lightning Charging Cable)
9	0.577889	(AAA Batteries (4-pack), Lightning Charging Ca...
10	0.564968	(USB-C Charging Cable, AAA Batteries (4-pack))
11	0.488873	(USB-C Charging Cable, AAA Batteries (4-pack),...

```
In [40]: # importing required module
from mlxtend.frequent_patterns import association_rules

# creating association rules
res=association_rules(res, metric="lift", min_threshold=1)

# printing association rules
res
```

Out[40]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(USB-C Charging Cable)	(Lightning Charging Cable)	0.718593	0.719311	0.575018	0.800200	1.112453	0.058126	1.404849
1	(Lightning Charging Cable)	(USB-C Charging Cable)	0.719311	0.718593	0.575018	0.799401	1.112453	0.058126	1.402835
2	(AAA Batteries (4-pack))	(Lightning Charging Cable)	0.707825	0.719311	0.577889	0.816430	1.135017	0.068743	1.529058
3	(Lightning Charging Cable)	(AAA Batteries (4-pack))	0.719311	0.707825	0.577889	0.803393	1.135017	0.068743	1.486089
4	(USB-C Charging Cable)	(AAA Batteries (4-pack))	0.718593	0.707825	0.564968	0.786214	1.110746	0.056330	1.366670
...
255	(USB-C Charging Cable, Bose SoundSport Headpho...	(AAA Batteries (4-pack))	0.475233	0.707825	0.407753	0.858006	1.212173	0.071371	2.057659
256	(AAA Batteries (4-pack), Bose SoundSport Headp...	(USB-C Charging Cable)	0.480976	0.718593	0.407753	0.847761	1.179752	0.062127	1.848458
257	(USB-C Charging Cable)	(AAA Batteries (4-pack), Bose SoundSport Headp...	0.718593	0.480976	0.407753	0.567433	1.179752	0.062127	1.199068
258	(AAA Batteries (4-pack))	(USB-C Charging Cable, Bose SoundSport Headpho...	0.707825	0.475233	0.407753	0.576065	1.212173	0.071371	1.237847
259	(Bose SoundSport Headphones)	(USB-C Charging Cable, AAA Batteries (4-pack))	0.577172	0.564968	0.407753	0.706468	1.250457	0.081670	1.482059

260 rows × 9 columns

```
In [41]: frequent_itemsets = fpgrowth(df_fp, min_support=0.4, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```


Out[41]:

	support	itemsets	length
0	0.719311	(Lightning Charging Cable)	1
1	0.718593	(USB-C Charging Cable)	1
2	0.707825	(AAA Batteries (4-pack))	1
3	0.685571	(Wired Headphones)	1
4	0.620962	(Apple Airpods Headphones)	1
5	0.687006	(AA Batteries (4-pack))	1
6	0.400574	(27in FHD Monitor)	1
7	0.577172	(Bose SoundSport Headphones)	1
8	0.575018	(USB-C Charging Cable, Lightning Charging Cable)	2
9	0.577889	(AAA Batteries (4-pack), Lightning Charging Ca...)	2
10	0.564968	(USB-C Charging Cable, AAA Batteries (4-pack))	2
11	0.488873	(USB-C Charging Cable, AAA Batteries (4-pack),...	3

In [42]: # Sort values based on confidence
res.sort_values("confidence",ascending=False)

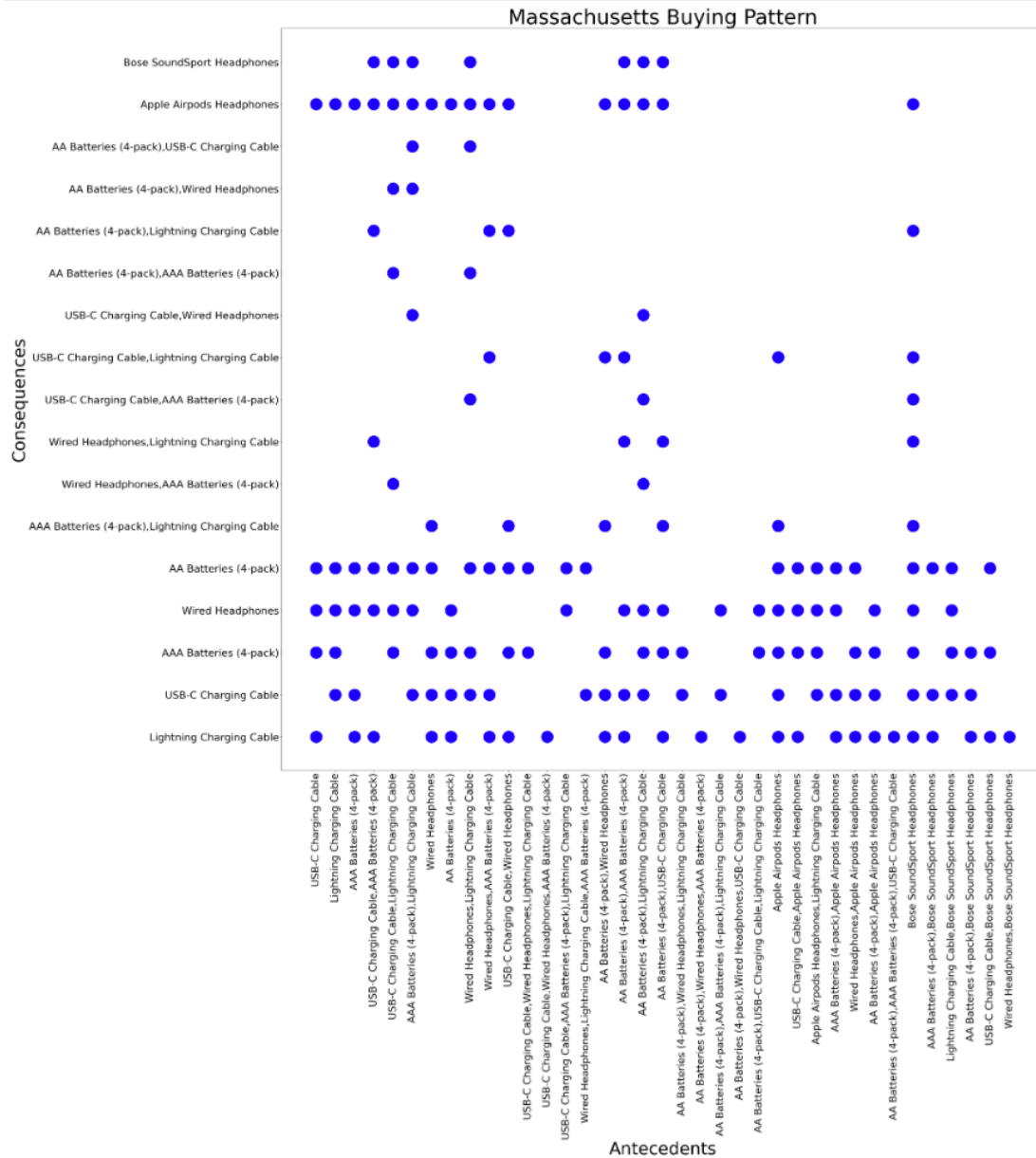
Out[42]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
39	(USB-C Charging Cable, Wired Headphones, AAA B...	(Lightning Charging Cable)	0.463747	0.719311	0.414214	0.893189	1.241729	0.080635	2.627901
85	(AA Batteries (4-pack), Wired Headphones, USB-...	(Lightning Charging Cable)	0.452979	0.719311	0.402728	0.889065	1.235996	0.076895	2.530212
243	(Wired Headphones, Bose SoundSport Headphones)	(Lightning Charging Cable)	0.459440	0.719311	0.406317	0.884375	1.229475	0.075837	2.427582
71	(AA Batteries (4-pack), Wired Headphones, AAA ...)	(Lightning Charging Cable)	0.460158	0.719311	0.404882	0.879875	1.223220	0.073885	2.336646
187	(AA Batteries (4-pack), AAA Batteries (4-pack)...	(Lightning Charging Cable)	0.467337	0.719311	0.409907	0.877112	1.219378	0.073746	2.284108
...
240	(Lightning Charging Cable)	(AA Batteries (4-pack), Bose SoundSport Headph...	0.719311	0.463029	0.405599	0.563872	1.217789	0.072537	1.231223
82	(Lightning Charging Cable)	(AA Batteries (4-pack), Wired Headphones, AAA ...)	0.719311	0.460158	0.404882	0.562874	1.223220	0.073885	1.234981
97	(USB-C Charging Cable)	(AA Batteries (4-pack), Wired Headphones, Ligh...	0.718593	0.465901	0.402728	0.560440	1.202916	0.067935	1.215075
96	(Lightning Charging Cable)	(AA Batteries (4-pack), Wired Headphones, USB-...	0.719311	0.452979	0.402728	0.559880	1.235996	0.076895	1.242891
234	(USB-C Charging Cable)	(AA Batteries (4-pack), Bose SoundSport Headph...	0.718593	0.463029	0.400574	0.557443	1.203903	0.067845	1.213335

260 rows × 9 columns

```
In [47]: plt.figure(figsize=(40,40), dpi=100)
plt.scatter(antel,conseq,s=1200,c='blue')

plt.title('Massachusetts Buying Pattern',fontsize=60)
plt.xlabel('Antecedents',fontsize=50)
plt.xticks(rotation=90,fontsize=30)
plt.ylabel('Consequences',fontsize=50)
plt.yticks(rotation=0,fontsize=30)
plt.savefig('ScatterPlot_MA_FP.png', bbox_inches = 'tight', pad_inches = 1, facecolor = 'w')
plt.show()
```



Recommendation System:

Application: Recommending top 10 items with 30 minimum interactions based on popularity

```
list(top_n_items(final_rating, 10, 30))
```

```
['B0052SCU8U',  
'B001TH7T2U',  
'B00BQ4F9ZA',  
'B00IVFDZBC',  
'B001TH7GUU',  
'B001QUA6RA',  
'B00316263Y',  
'B008EQZ25K',  
'B003ES5ZUU',  
'B0000BZL1P']
```

These items **'B0052SCU8U'**, **'B001TH7T2U B00BQ4F9ZA'**, **'B00IVFDZBC'**, **'B001TH7GUU'**, **'B001QUA6RA'**, **'B00316263Y'**, **'B008EQZ25K'**, **'B003ES5ZUU'**, and **'B0000BZL1P'** will be recommended to new users.

```
sim_options = {'name': 'cosine',  
               'user_based': True}  
  
algo_knn_user = KNNBasic(sim_options=sim_options, verbose=False)  
  
# Train the algorithm on the trainset, and predict ratings for the testset  
algo_knn_user.fit(trainset)  
predictions = algo_knn_user.test(testset)  
  
# Then compute RMSE  
accuracy.rmse(predictions)  
  
RMSE: 1.0552  
1.0551559636074466
```

Application 1: Predicting the rating for the user with `userId= 0` and for `itemId= 3906`, which the user has interacted with before

```
algo_knn_user.predict(0, 3906, r_ui=4, verbose=True)  
  
user: 0          item: 3906      r_ui = 4.00    est = 4.29    {'was_impossible': True, 'reason': 'Not enough neighbors.'}  
Prediction(uid=0, iid=3906, r_ui=4, est=4.291366722826364, details={'was_impossible': True, 'reason': 'Not enough neighbors.'})
```

Application 2: Predicting rating for the same `userId= 0` but for an item `itemId=100`, which this user has not interacted before

```
algo_knn_user.predict(0, 100, verbose=True)  
  
user: 0          item: 100      r_ui = None    est = 4.00    {'actual_k': 1, 'was_impossible': False}  
Prediction(uid=0, iid=100, r_ui=None, est=4.0, details={'actual_k': 1, 'was_impossible': False})
```

```
# Using the optimal similarity measure for user-user based collaborative filtering
sim_options = {'name': 'cosine', 'user_based': True}

# Creating an instance of KNNBasic with optimal hyperparameter values
similarity_algo_optimized = KNNBasic(sim_options=sim_options, k=40, min_k=6, Verbose=False)

# Training the algorithm on the trainset
similarity_algo_optimized.fit(trainset)

# Predicting ratings for the testset
predictions = similarity_algo_optimized.test(testset)

# Computing RMSE on testset
accuracy.rmse(predictions)

Computing the cosine similarity matrix...
Done computing similarity matrix.
RMSE: 0.9887

0.988698949219415
```

After tuning hyperparameters, RMSE for testset has reduced from 1.05 to 0.98 from 1.05. Therefore the model has improved after hyperparameter tuning

Predicted rating for `userId=0` and for `itemId= 3906` using this tuned/optimized user based collaborative filtering

```
similarity_algo_optimized.predict(0,3906, r_ui=4, verbose=True)

user: 0          item: 3906          r_ui = 4.00    est = 4.29    {'was_impossible': True, 'reason': 'Not enough neighbors.'}
Prediction(uid=0, iid=3906, r_ui=4, est=4.291366722826364, details={'was_impossible': True, 'reason': 'Not enough neighbors.'})
```

Predicting rating for the same `userId=0` but for an item which this user has not interacted before i.e. `itemId=100` , by using the optimized model as shown below

```
similarity_algo_optimized.predict(0,100, verbose=True)

user: 0          item: 100          r_ui = None    est = 4.29    {'was_impossible': True, 'reason': 'Not enough neighbors.'}
```

```
# Defining similarity measure
sim_options = {'name': 'cosine',
               'user_based': False}

# Defining Nearest neighbour algorithm
algo_knn_item = KNNBasic(sim_options=sim_options, verbose=False)

# Train the algorithm on the trainset or fitting the model on train dataset
algo_knn_item.fit(trainset)

# Predict ratings for the testset
predictions = algo_knn_item.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)

RMSE: 1.0697

1.0697030922594737
```

Predicting rating of `userId=0` for `itemId=3906`

```
algo_knn_item.predict(0, 3906, r_ui=4, verbose=True)

user: 0          item: 3906          r_ui = 4.00    est = 4.29    {'was_impossible': True, 'reason': 'Not enough neighbors.'}
Prediction(uid=0, iid=3906, r_ui=4, est=4.291366722826364, details={'was_impossible': True, 'reason': 'Not enough neighbors.'})
```

Predicting rating of `userId=0` (the same user) for `itemId=100`

```
algo_knn_item.predict(0, 100, verbose=True)

user: 0          item: 100          r_ui = None    est = 5.00    {'actual_k': 1, 'was_impossible': False}
Prediction(uid=0, iid=100, r_ui=None, est=5, details={'actual_k': 1, 'was_impossible': False})
```

Regression:

Regression Analysis on Sales Product

```
In [1]: import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [2]: df = pd.read_csv('Sales_2019_Merged.csv')
df.head()
```

Out[2]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001
1	176559	Bose SoundSport Headphones	1	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
2	176560	Google Phone	1	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
3	176560	Wired Headphones	1	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176561	Wired Headphones	1	11.99	04/30/19 09:27	333 8th St, Los Angeles, CA 90001

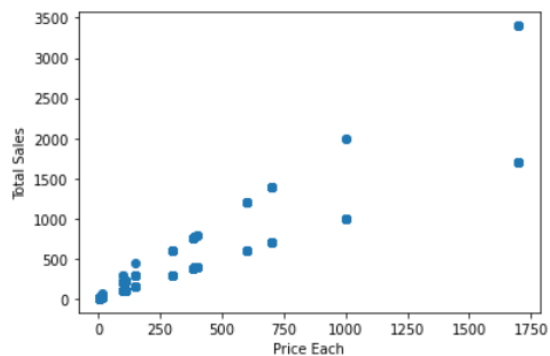
```
In [4]: df['Total Sales'] = df['Quantity Ordered']*df['Price Each']
df.head(2)
```

Out[4]:

	Order ID	Product	Quantity Ordered	Price Each	Order Date	Purchase Address	Total Sales
0	176558	USB-C Charging Cable	2	11.95	04/19/19 08:46	917 1st St, Dallas, TX 75001	23.90
1	176559	Bose SoundSport Headphones	1	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215	99.99

```
In [5]: plt.xlabel('Price Each')
plt.ylabel('Total Sales')
plt.scatter(df['Price Each'], df['Total Sales'])
```

Out[5]: <matplotlib.collections.PathCollection at 0x2565c63e880>



```
In [6]: # correlation plot
corr = df.corr()
sns.heatmap(corr, cmap = 'Wistia', annot= True)
```

Out[6]: <AxesSubplot:>



```
In [7]: df.columns
```

Out[7]: Index(['Order ID', 'Product', 'Quantity Ordered', 'Price Each', 'Order Date', 'Purchase Address', 'Total Sales'], dtype='object')

```
In [8]: df = df.drop(['Order ID', 'Quantity Ordered', 'Order Date', 'Purchase Address'], axis = 1)
```

```
In [9]: from sklearn.preprocessing import LabelEncoder

df.apply(LabelEncoder().fit_transform)
```

Out[9]:

	Product	Price Each	Total Sales
0	15	2	16
1	7	5	30
2	9	13	43
3	17	3	8
4	17	3	8
...
185945	5	0	4
185946	18	14	44
185947	18	14	44
185948	3	10	39
185949	15	2	6

185950 rows × 3 columns

```
In [10]: # one hot encoding

df = pd.get_dummies(df)

print(df.shape)
```

(185950, 21)

In [11]: `# splitting the data into dependent and independent variables`

```
x = df.drop('Total Sales', axis = 1)
y = df['Total Sales']

print(x.shape)
print(y.shape)
```

```
(185950, 20)
(185950,)
```

In [12]: `from sklearn.model_selection import train_test_split`

```
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size = 0.3)
```

In [13]: `from sklearn import linear_model`

```
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
```

Out[13]: `LinearRegression()`

In [14]: `y_pred = reg.predict(X_test)`

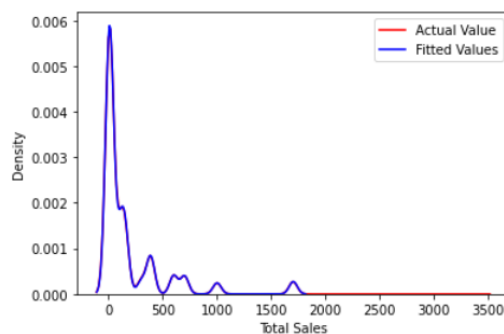
In [15]: `import seaborn as sns
%matplotlib inline
ax1 = sns.distplot(y_test, hist=False, color='r', label='Actual Value')
sns.distplot(y_pred, hist=False, color='b', label='Fitted Values', ax=ax1)
ax1.ticklabel_format(style='plain')
plt.legend(loc='best')
plt.show()`

C:\Users\Shahirah\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level funclexibility) or `kdeplot` (an axes-level function for kernel density plots).

warnings.warn(msg, FutureWarning)

C:\Users\Shahirah\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level funclexibility) or `kdeplot` (an axes-level function for kernel density plots).

warnings.warn(msg, FutureWarning)



In [17]: `import sklearn.metrics as metrics
import numpy as np
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = metrics.r2_score(y_test,y_pred)

print("Results of sklearn.metrics:")
print("MAE:",mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R-Squared:", r2)`

```
Results of sklearn.metrics:
MAE: 1.950525109781128
MSE: 174.54849602866227
RMSE: 13.21168028786128
R-Squared: 0.99843533752362
```

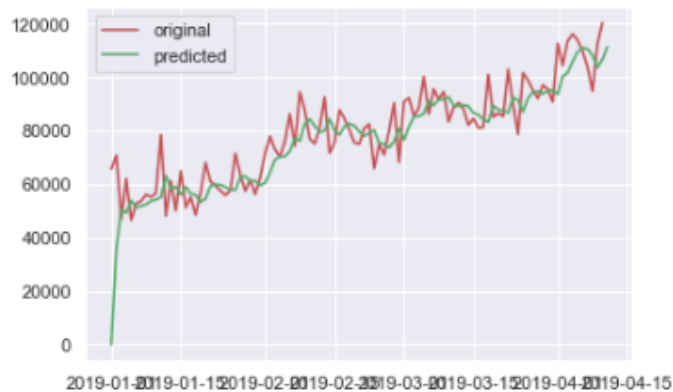
Time Series Analysis:

```
from math import sqrt
from sklearn.metrics import mean_squared_error

plt.plot(df1.Sales[:100], label = 'original', color='r' )
plt.plot(results.predict(0,100), label = 'predicted', color='g' )

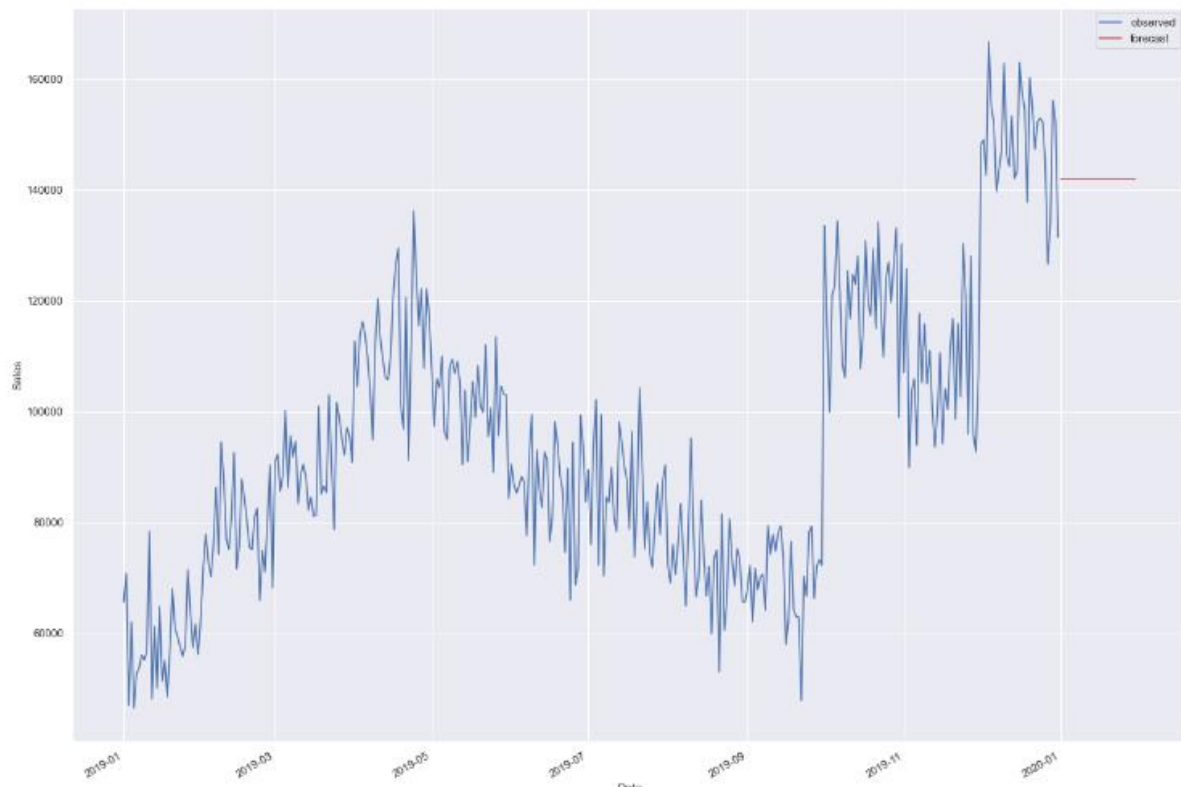
plt.legend()
```

<matplotlib.legend.Legend at 0x1b33f395430>



```
# Plot the real and predicted value in density curve
ci = df_forecast.values
ax = df1[:400].Sales.plot(label='observed', figsize=(20, 15))
df_forecast.plot(ax=ax, label='Forecast', color='r')
ax.fill_between(df_forecast.index,
               ci[:, -1],
               ci[:, 0], color='b', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')

plt.legend()
plt.show()
```




```

# Load specific evaluation tools
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse

# Calculate root mean squared error
rmse=rmse(df1.Sales, results.predict())
print("The rmse value is:",rmse)

# Calculate mean squared error
mse = mean_squared_error(df1.Sales, results.predict())
print("The mse value is:",mse)

```

The rmse value is: 11496.478368552902
The mse value is: 132169014.8786048

```

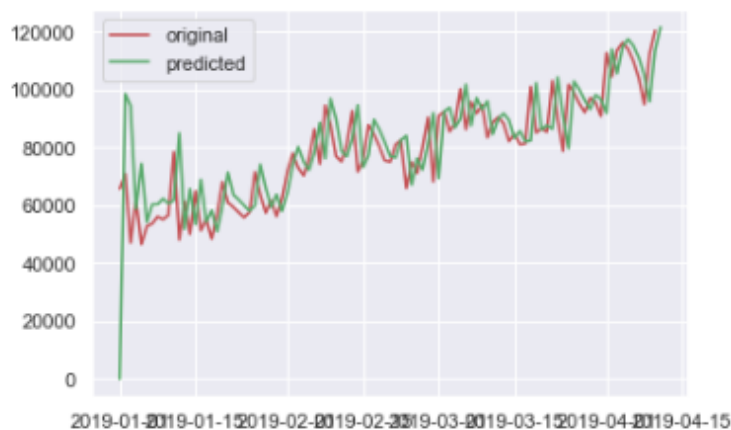
from math import sqrt
from sklearn.metrics import mean_squared_error

plt.plot(df1.Sales[:100],label = 'original', color='r' )
plt.plot(results1.predict(0,100),label = 'predicted', color='g' )

plt.legend()

```

<matplotlib.legend.Legend at 0x1b33f6d06d0>



```

# Plot the observed and forecasting value
ci2 = df_forecast1.values
ax2 = df1[:400].Sales.plot(label='observed', figsize=(20, 15))
df_forecast1.plot(ax=ax2,label='Forecast',color='r')
ax2.fill_between(df_forecast1.index,
                 ci2[:, -1],
                 ci2[:, 0], color='b', alpha=.25)
ax2.set_xlabel('Date')
ax2.set_ylabel('Sales')

plt.legend()
plt.show()

```



```
# Load specific evaluation tools
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse

# Calculate root mean squared error
rmse=rmse(df1.Sales, results1.predict())
print("The rmse value is:",rmse)

# Calculate mean squared error
mse = mean_squared_error(df1.Sales, results1.predict())
print("The mse value is:",mse)
```

The rmse value is: 13890.190028442816
The mse value is: 192937379.0262522

```

# Evaluate
rmse, y, yhat = walk_forward_validation(data, 20)
print('RMSE: %.3f' % rmse)

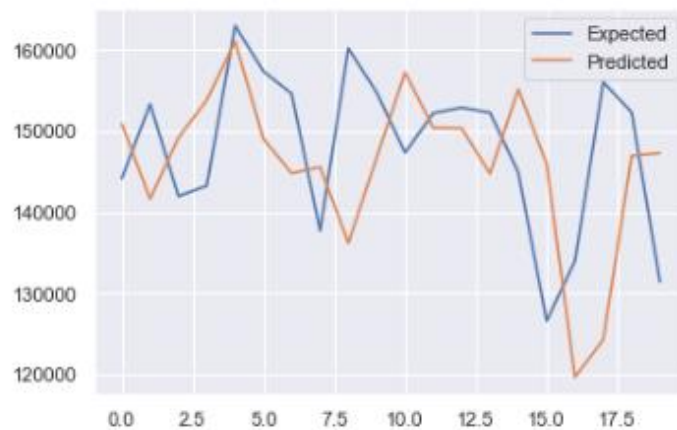
# plot expected vs predicted
pyplot.plot(y, label='Expected')
pyplot.plot(yhat, label='Predicted')
pyplot.legend()
pyplot.show()

```

```

>expected=144195.1, predicted=150891.0
>expected=153332.1, predicted=141653.0
>expected=141991.8, predicted=149190.8
>expected=143269.7, predicted=153828.3
>expected=162970.6, predicted=161026.1
>expected=157364.7, predicted=149048.8
>expected=154598.8, predicted=144804.5
>expected=137732.7, predicted=145574.8
>expected=160181.3, predicted=136225.8
>expected=154756.9, predicted=146753.7
>expected=147348.1, predicted=157223.5
>expected=152214.5, predicted=150409.6
>expected=152888.8, predicted=150345.8
>expected=152268.4, predicted=144777.5
>expected=144912.0, predicted=155159.5
>expected=126628.1, predicted=145952.0
>expected=134015.5, predicted=119720.9
>expected=156024.6, predicted=124322.3
>expected=152319.8, predicted=146953.6
>expected=131454.3, predicted=147289.9
RMSE: 10723.614

```



```

# Plot the forecast and actual sales
c1 = df_preds.values
ax1 = df1[:400].Sales.plot(label='observed', figsize=(20, 15))
df_preds.plot(ax=ax1, label='Forecast', color='r')
ax1.fill_between(df_forecast.index,
                 c1[:, -1],
                 c1[:, 0], color='b', alpha=.25)
ax.set_xlabel('Date')
ax.set_ylabel('Sales')

plt.legend()
plt.show()

```

