RETAIL DATA INSIGHTS

An Analytics approach to find insights from Retail Data

Abstract

Due to the impact of COVID 19, more businesses are moving online. This means more opportunities for B2B or B2C. With more online orders and increasing competition, it becomes extremely essential to find insights from historical data and plan the business as per Sales Trend, Customer Behaviour and Customer Segmentation. This project is a demonstration of finding insights for better customer engagement and increasing the growth of the business.

Nitish Bhardwaj

Contents

Table of Figures

1. Business Problem

COVID 19 had a hard hit on all the industries in the year 2020. It forced most of the businesses to move online. It inspired customers to buy and try more products online. Only the organizations which are constantly evolving and making continuous structural changes aligned to customer needs would be able to sustain in this ever-changing time.

The year 2020 motivated a lot of the organizations to spend more money on finding insights on their data to provide a better customer experience to increase customer engagement and sales.

The business problem is having raw retail data from various sources and finding the opportunities to find insights that would help re-align the business for better growth.

2. Data Collection

The raw data for this business problem is fetched from the SQLite database. The database consists of three tables: Transactions, Products and Segments.

Data is fetched from the database using the SQLite3 connector provided in Python.

A function is defined to accept the database name and return the connection object if a connection has been established or an error in case of failure.

Figure 1 is a screenshot of the function is given below.

```
#Function to establish a connection with Sqlite3
def sqlite_connection(dbname):
    """
    Function to establish a connection with the database

    Attributes:
        dbname(str): database name with the extenstion. Example: sample.db

    Returns:
        SQLlite3 connection object
    """
    try:
        conn = sqlite3.connect(dbname)
        print("Connection Established to:", dbname)
        return conn
    except Error:
        print(Error)
```

Figure 1: Function to establish a connection with the SQLite database

Based on the above function, three dataframes are created for easier access to data in python and pandas. The screenshots of these dataframes are given below in Figures 2, 3 and 4.

Dataframe for transactions table: **dfTransactions**

Dataframe shape: (2666, 6)

	trans_id	trans_dt	cust_id	prod_id	item_qty	item_price
0	1	2016-01-02 10:06:00	9085146	223029	1	42.99
1	2	2016-01-02 10:30:00	1215814	252270	1	103.95
2	2	2016-01-02 10:30:00	1215814	260383	1	74.99
3	4	2016-01-02 11:33:00	18511160	269119	1	51.99
4	4	2016-01-02 11:33:00	18511160	411162	1	59.99

Figure 2: Dataframe created from the Transactions table

Dataframe for products table: **dfProducts**

Dataframe shape: (1863, 4)

	prod_id	prod_name	brand	category
0	242151	Product 242151	Y	Make up
1	245067	Product 245067	D	Women
2	279311	Product 279311	C	Women
3	75231178	Product 75231178	C	Make up
4	218423	Product 218423	S	Women

s

Figure 3: Dataframe created from the Products table

Dataframe for segments table: **dfSegments**

Dataframe shape: (6124, 4)

	cust_id	seg_name	update_at	active_flag
0	4402	ONE-OFFS	2014-06-01 00:00:00	N
1	4402	LAPSED	2015-12-01 00:00:00	N
2	4402	LAPSED	2015-06-01 00:00:00	N
3	4402	LAPSED	2014-01-01 00:00:00	N
4	4402	ONE-OFFS	2016-02-01 00:00:00	Y

Figure 4: Dataframe created from the Segments table

3. Data Understanding

The semantic description of the three tables mentioned in the Data Collection section is mentioned below:

1. `**transactions**`: contains detailed information about each product a customer has purchased. A transaction consists of one or more products purchased by a customer indexed by unique transaction id. Following columns are present in this table:

- `trans_id`: the transaction id
- `cust_id`: the customer id
- `prod_id`: the product id
- `item_qty`: the quantity of the product that is being purchased
- `item_price`: the per-unit price of the product (NOTE: the total revenue for a product is `item_qty * item_price`)

2. `**products**`: contains detailed attributes about each product.

- `prod_id`: the product id (same meaning as in `transactions`)
- `prod_name`: the product name
- `brand`: the brand of the product
- `category`: the category of the product

3. `**segments**`: contains a history of customer segmentation labelling for each customer. Segments are computed periodically for all current customers and appended to the table after each computation. The current (most up to date) active segment for each customer is specified by `active_flag = 'Y'` column.

- `cust_id`: the customer id (same meaning as in `transactions`)
- `seg_name`: the segment of this customer
- `update_dt`: the date when this segment was updated
- `active_flag`: whether or not this segment is the active segment for this customer

Further investigation of data revealed that none of the columns have any missing information.

4. Data Mining

In this section, the data is mined from the database as per the request. Four primary business requests are addressed as per the business requirement. The business requests and the developed solution is given below:

```
1. Find the current active segment for each customer sorted by the segment
   update date.  The output should contain three columns: `cust_id`,
   `seg_name`, `updated_at`.  Here is some sample output:

        cust_id     seg_name        updated_at
        4402        LAPSED          2014-06-01 00:00:00
        11248       ONE-OFFS        2015-10-01 00:00:00
```

Developed SQL for Requirement 1:

```
SELECT cust_id,seg_name,update_at
FROM segments
WHERE active_flag = 'Y'
GROUP BY cust_id,seg_name
HAVING MAX(update_at)
ORDER BY cust_id, update_at DESC
```

Assumptions:

The output given in the sample query is ambiguous. For '4402' and '11248' customer id's seg_name and Updated_at is not active in the table. It is assumed to select the active segment.

A screenshot of the output(first five rows) from the above SQL is given below in figure 5.

	cust_id	seg_name	update_at
0	4402	ONE-OFFS	2016-02-01 00:00:00
1	11248	LOYAL	2016-02-01 00:00:00
2	12064	INFREQUENT	2016-06-01 00:00:00
3	15088	ONE-OFFS	2016-02-01 00:00:00
4	66706	INFREQUENT	2016-04-01 00:00:00

Figure 5: Output of SQL 1, Data Mining problem 1

```
2. For each product purchased between Jan 2016 and May 2016 (inclusive), find
   the number of distinct transactions.  The output should contain `prod_id`,
   `prod_name` and distinct transaction columns.  Here is some sample output:

        prod_id     prod_name       count
        199922      Product 199922  1
        207344      Product 207344  1
        209732      Product 209732  1
```

Developed SQL for Requirement 2:

```
SELECT trans.Prod_id, prod.Prod_name, COUNT(DISTINCT(trans.trans_id))
as count
FROM transactions trans LEFT OUTER JOIN products prod ON trans.Prod_id
= prod.Prod_id
WHERE (strftime('%Y', trans.TRANS_DT) = '2016' --Only consider 2016
transactions
AND strftime('%m', trans.TRANS_DT) BETWEEN '01' AND '05') --Only
consider transactions between Jan and May
GROUP BY trans.Prod_id
```

A screenshot of the output(first five rows) from the above SQL is given below in figure 6.

	prod_id	prod_name	count
0	199922	Product 199922	1
1	207344	Product 207344	1
2	209732	Product 209732	1
3	209999	Product 209999	1
4	211200	Product 211200	1

Figure 6: Output of SQL 2, Data Mining problem 2

```
3. Find the most recent segment of each customer as of 2016-03-01.
   *Hint*: You cannot simply use `active_flag` since that is as of the current
   date *not* 2016-03-01.  The output should contain the `cust_id`, `seg_name`
   and `update_at`  columns and should have at most one row per customer.  Here
   is some sample output:

       cust_id  seg_name    update_at
       4402     ONE-OFFS    2016-02-01 00:00:00
       11248    LOYAL       2016-02-01 00:00:00
       126169   ONE-OFFS    2015-03-01 00:00:00
```

Developed SQL for Requirement 3:

```
SELECT cust_id,seg_name,update_at
FROM segments
WHERE DATE(update_at) <= '2016-03-01' and active_flag = 'Y'
GROUP by cust_id
HAVING MAX(update_at)
```

Assumptions:

As of '2016-03-01' date is considered inclusive for the query.

A screenshot of the output(first five rows) from the above SQL is given below in figure 7.

	cust_id	seg_name	update_at
0	4402	ONE-OFFS	2016-02-01 00:00:00
1	11248	LOYAL	2016-02-01 00:00:00
2	12064	INFREQUENT	2016-02-01 00:00:00
3	15088	ONE-OFFS	2016-02-01 00:00:00
4	66706	ONE-OFFS	2016-02-01 00:00:00

Figure 7: Output of SQL 3, Data Mining problem 3

```
4. Find the most popular category (by revenue) for each active segment.
   *Hint*: The current (most up to date) active segment is specified by `active_f
lag = 'Y'` column in the segments table.
   Here is the some sample output:

   seg_name    category    revenue
   INFREQUENT  Women       20264
```

Developed SQL for Requirement 4:

```
SELECT inn_rev.Seg_name, inn_rev.category, inn_rev.revenue
FROM (SELECT seg.Seg_name, prod.category,
SUM(trans.item_qty*trans.item_price) as revenue
              FROM transactions trans INNER JOIN segments seg ON
seg.cust_id=trans.cust_id
                                      INNER JOIN products prod ON
prod.prod_id=trans.prod_id
              WHERE seg.active_flag = 'Y'
              GROUP BY seg.Seg_name, prod.category
              ORDER BY seg_name ASC, revenue desc) inn_rev
GROUP BY inn_rev.Seg_name
HAVING MAX(inn_rev.revenue)
```

Assumptions:

It is assumed that for each cust_id in the transactions table, there is a matching cust_id in the segment table and prod_id in the products table.

A screenshot of the output from the above SQL is given below in figure 8.

	Seg_name	category	revenue
0	INFREQUENT	Women	20264.39
1	LOYAL	Women	17346.20
2	NEW	Women	3141.35
3	ONE-OFFS	Women	10690.73
4	VIP	Women	29134.07

Figure 8: Output of SQL 4, Data Mining problem 4

5. Data Processing

Most of the analysis is based on the Transactions table. Transactions table is having the details of the date and time for each transaction in column 'trans_dt'. This column is processed to fetch the month, week, day, weekday and hour information using Python and strftime() function.

A snippet of the code is given below in figure 9.

```
%%timeit
dfTransactions['month'] = dfTransactions['trans_dt'].apply(lambda x: x.strftime('%m'))
dfTransactions['week'] = dfTransactions['trans_dt'].apply(lambda x: x.strftime('%W'))
dfTransactions['day'] = dfTransactions['trans_dt'].apply(lambda x: x.strftime('%d'))
dfTransactions['weekday'] = dfTransactions['trans_dt'].apply(lambda x: x.strftime('%w'))
dfTransactions['hour'] = dfTransactions['trans_dt'].apply(lambda x: x.strftime('%H'))
```

```
79 ms ± 1.74 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Figure 9: Python code to process data into a month, week, day, weekday and hour

New columns having month, week, day, weekday and hour information are created after processing the data. The resulting dataframe is given below in figure 10.

dfTransactions dataframe after processing

	trans_id	trans_dt	cust_id	prod_id	item_qty	item_price	month	week	day	weekday	hour
0	1	2016-01-02 10:06:00	9085146	223029	1	42.99	01	00	02	6	10
1	2	2016-01-02 10:30:00	1215814	252270	1	103.95	01	00	02	6	10
2	2	2016-01-02 10:30:00	1215814	260383	1	74.99	01	00	02	6	10
3	4	2016-01-02 11:33:00	18511160	269119	1	51.99	01	00	02	6	11
4	4	2016-01-02 11:33:00	18511160	411162	1	59.99	01	00	02	6	11

Figure 10: Dataframe after processing the python code given in figure 9

6. Insights

Data is analyzed and several visualizations are created in the Jupyter-notebook to uncover the hidden information in the data. Please note that the visualizations created in Jupyter-notebook are highly interactive and will show more information on mouse click or mouse hover. Specific data points can also be selected in these visualizations.

6.1 Analysis based on each Month

A screenshot for the outcome of this analysis is given below in figure 11.

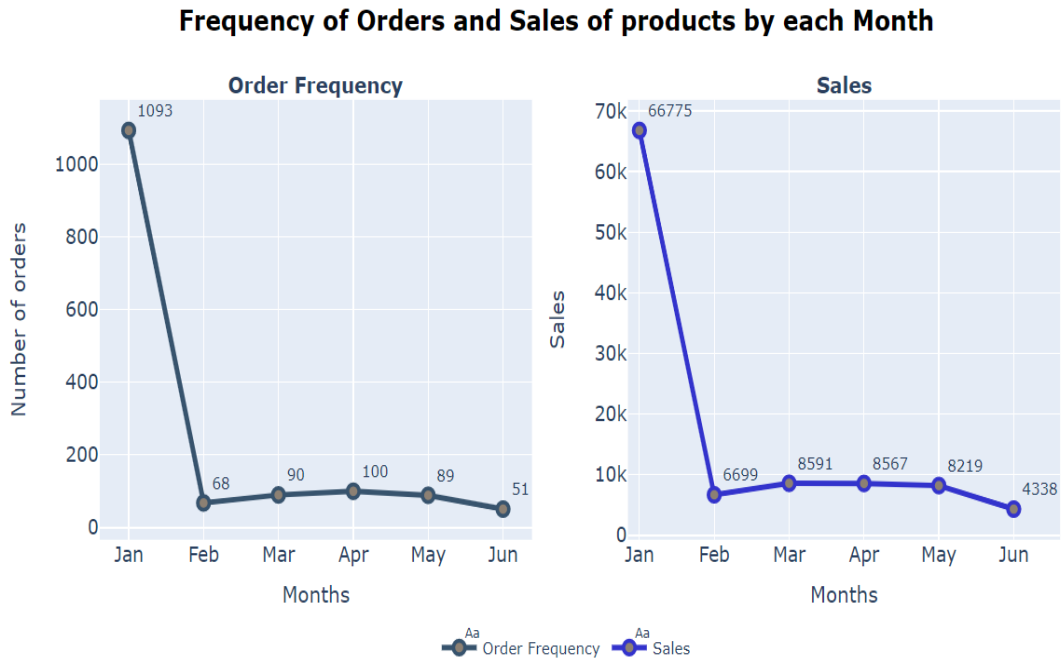

Figure 11: Visualization for Analysis based on each Month

Insights from above Visualization:

1. The highest number of orders were placed in the first month i.e. January month of the fiscal year: 2016. This also generated the highest revenue in the year.
2. Sales dropped exponentially from February month onwards.
3. In March, sales started rising by 1500 units, however, it dropped again in June.

6.2 Analysis based on each Day of Month

A screenshot for the outcome of this analysis is given below in figure 12.

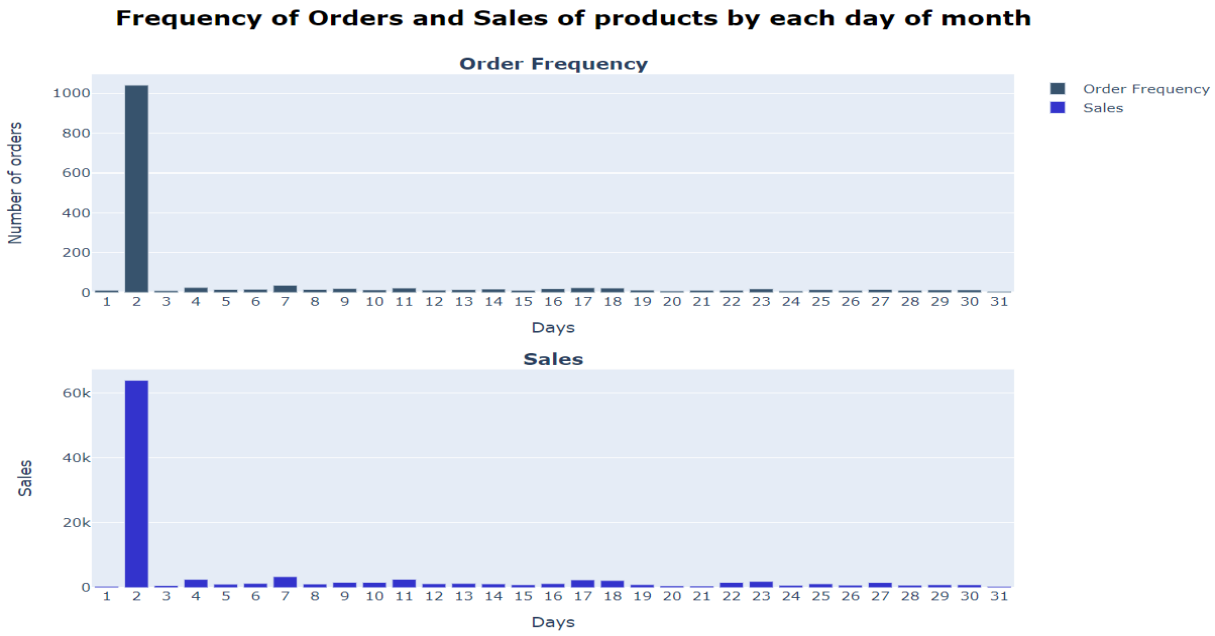

Figure 12: Visualization for Analysis based on each Day of Month

Insights from above Visualization:

1. The highest sales pattern is noticed on the 2nd day of the month.
2. Sales drop drastically from 3rd day onwards.

6.3 Analysis based on each Day of a Week

A screenshot for the outcome of this analysis is given below in figure 13.

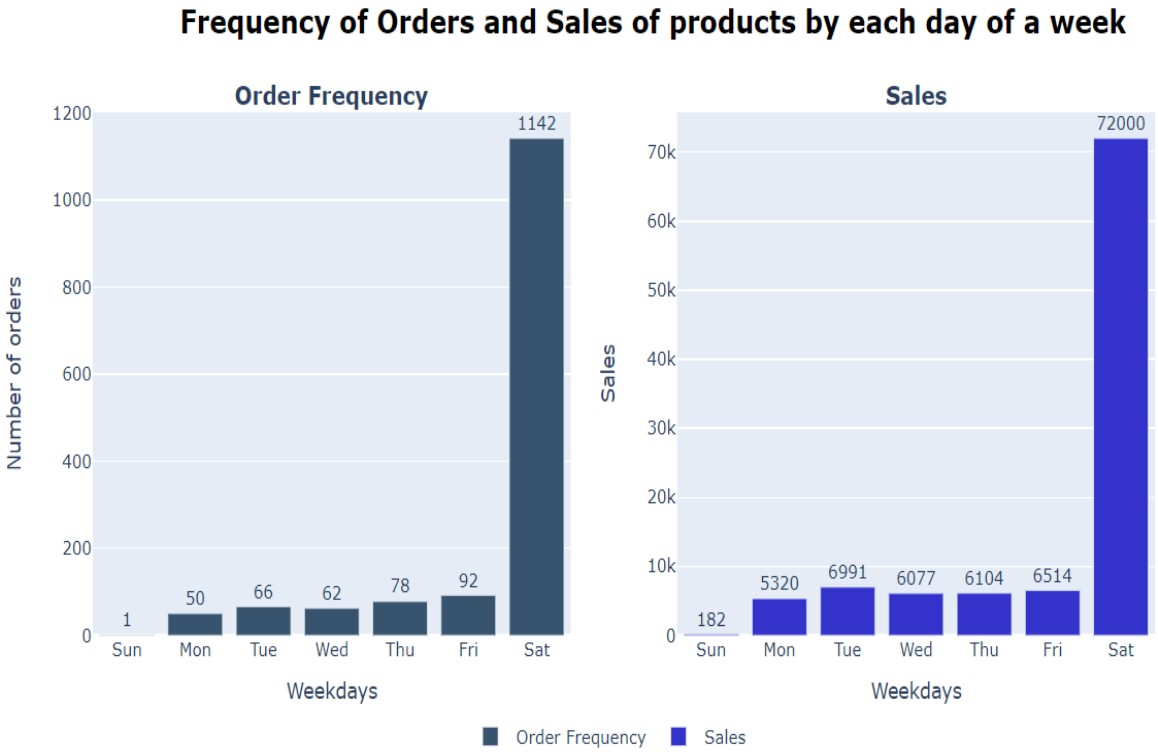

Figure 13: Visualization for Analysis based on each Day of a Week

Insights from above Visualization:

1. The highest sales pattern is noticed on Saturday which is the starting of the weekend.
2. Sales are lowest on a Sunday.
3. Due to a high number of orders on Saturday, there is an opportunity for greater acceptance on offers by customers on this day of the week.

6.4 Analysis based on each Hour of a Day

A screenshot for the outcome of this analysis is given below in figure 14.

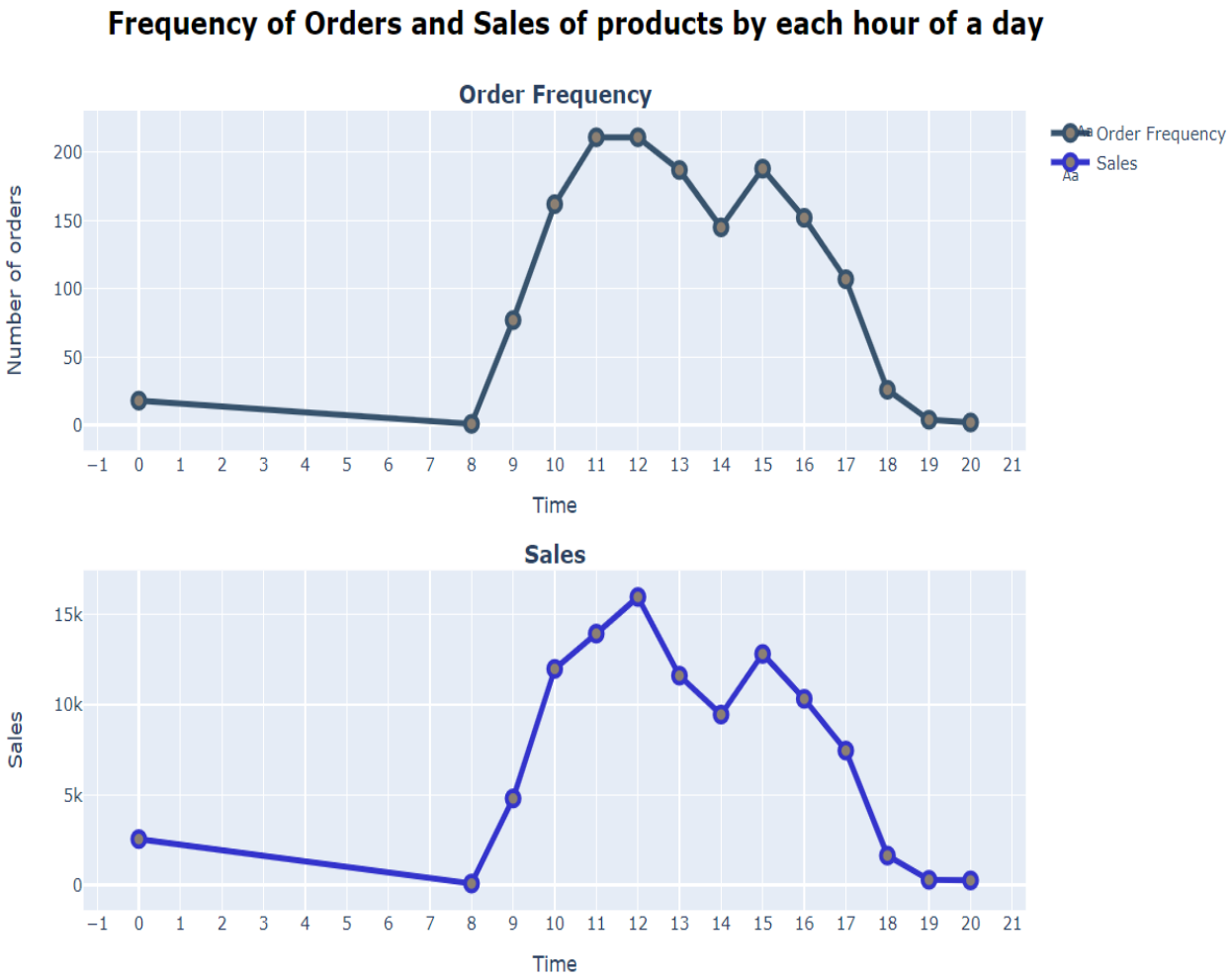

Figure 14: Visualization for Analysis based on each Hour of a Day

Insights from above Visualization:

1. Customers start placing orders from 8 AM onwards.
2. Most orders are placed between 11 AM to 12 PM.
3. Order starts to reduce from noon onwards possibly due to lunch break. However, at 3 PM a high number of orders are placed after which again orders start to reduce continuously.
4. The highest sales pattern is noticed at noon.
5. The time window from 10 AM to 12 PM and 2 PM to 4 PM could help to create higher revenues due to higher attention of customers in this period.

6.5 Analysis of Top sold products and high revenue-generating products

A screenshot for the outcome of this analysis is given below in figure 15.

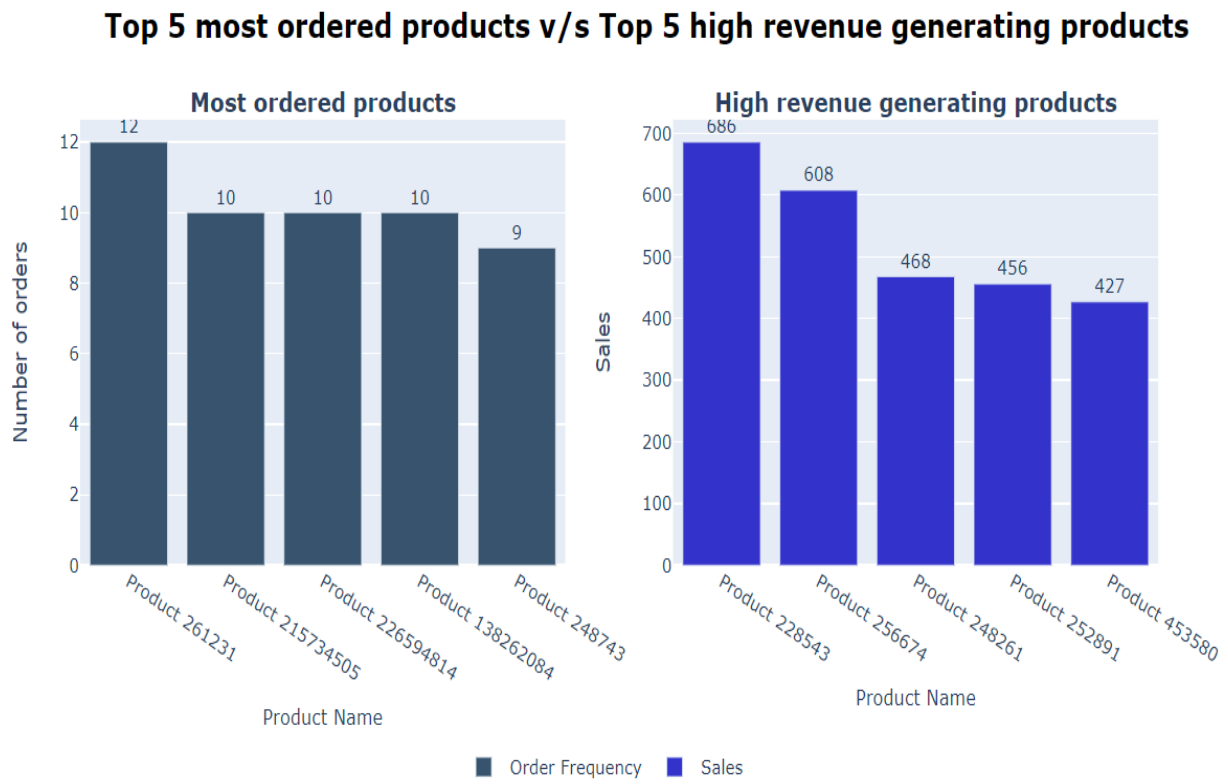

Figure 15: Analysis of Top sold products and high revenue-generating products

Insights from above Visualization:

1. Based on the above information on most sold products and the highest revenue generating products, similar products can be introduced to the customers.

6.6 Analysis of the highest revenue-generating categories

A screenshot for the outcome of this analysis is given below in figure 16.

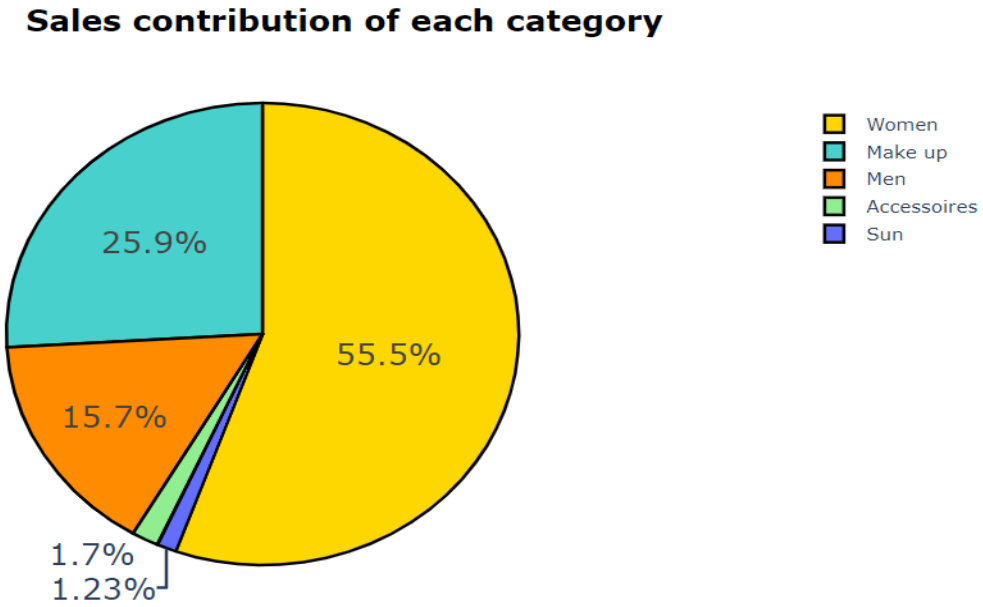

Figure 16: Analysis of the highest revenue-generating categories

Insights from above Visualization:

1. The highest revenue is generated from the Women category. The second-highest revenue-generating category is Makeup followed by Men's category.
2. Based on the above insights, more investment could be done in the Women and Makeup category to bring more customers.

6.7 Analysis of Brands from Top 3 categories generating the highest revenues

A screenshot for the outcome of this analysis is given below in figure 17.

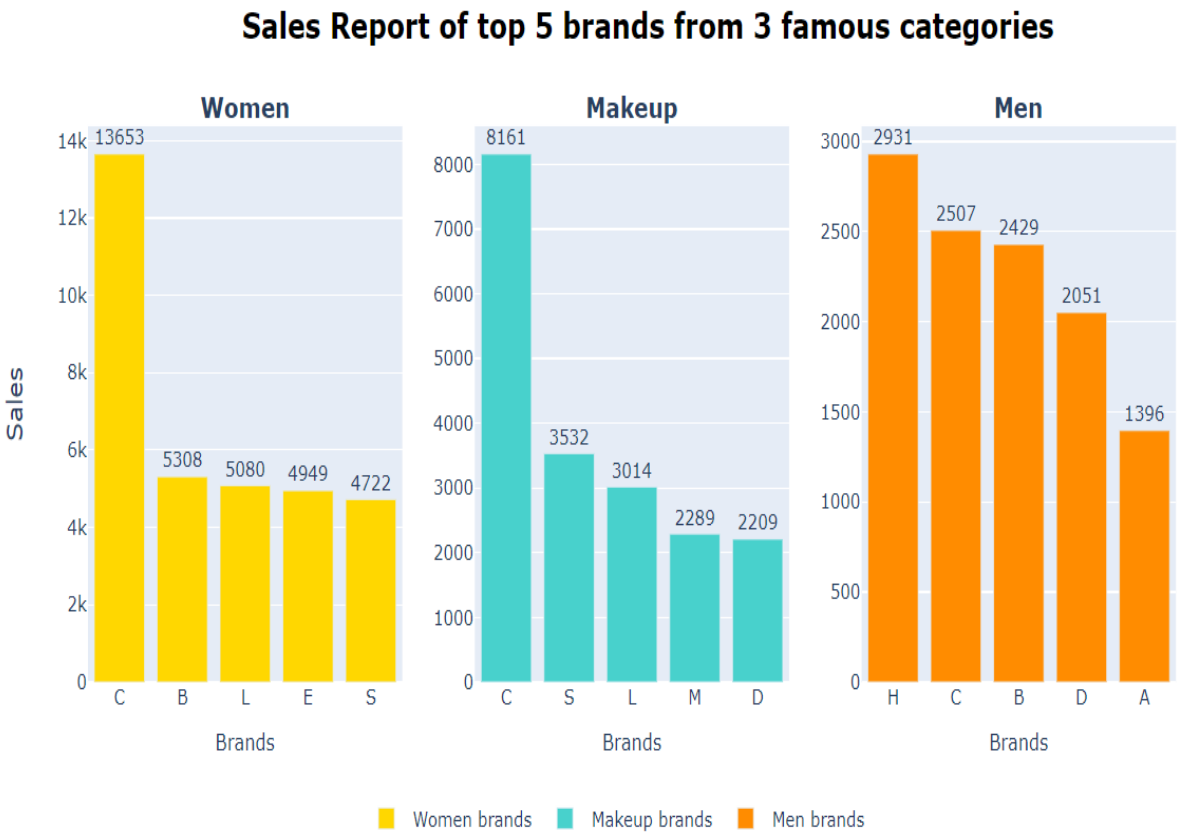

Figure 17: Analysis of Brands from Top 3 categories generating the highest revenues

Insights from above Visualization:

1. Based on the above visualization, it is clear that which brands are contributing to creating the highest revenue. More investment could be done in these brands or similar brands.
2. Brand C from the Women and Makeup category generates high revenue. It is the most liked brand by customers based on revenue results. An opportunity to bring more products from this category.
3. In Men's category, the share of H, C, B and D brand is almost similar and highest in this category.

6.8 Analysis based on Recency, Frequency, and Monetary(RFM)

RFM (Recency, Frequency, Monetary) analysis is a customer segmentation technique that uses past purchase behaviour to divide customers into groups [1][2]. RFM helps divide customers into various categories or clusters to identify customers who are more likely to respond to promotions and also for future personalization services.

RECENCY (R): Days since last purchase

FREQUENCY (F): Total number of purchases

MONETARY VALUE (M): Total money customer spent

In this analysis, first of all, Recency, Frequency and Monetary were calculated. Later, data was divided by python quantiles() functions into the distribution of 0.25, 0.50., 0.75 and remaining. Then, a custom function [3] was created to assign a number in the range of 1 to 4 for each Recency, Frequency and Monetary row. A Screenshot of the function is given below in figure 18.

```
# Arguments (x = value, p = recency, monetary_value, frequency, d = quartiles dict)
def RScore(x,p,d):
    if x <= d[p][0.25]:
        return 4
    elif x <= d[p][0.50]:
        return 3
    elif x <= d[p][0.75]:
        return 2
    else:
        return 1
# Arguments (x = value, p = recency, monetary_value, frequency, k = quartiles dict)
def FMScore(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

Figure 18: Python custom functions to calculate Recency, Frequency and Monetary quartiles

Using the above functions and clubbing the new values in the range of 1 to 4, the RFM score is computed. Here, Best Recency score = 4 means most recent payment, Best Frequency score = 4 means the most number of payments and Best Monetary score = 4 means most money spent.

With different combination of these scores, we can create different segments. These segments are created based on the business need [4].

A screenshot for the outcome of this analysis is given below in figure 19.

Customer Segments	Customer Segment Details	Number of Customers
Champions	Best customers who bought most recently, most often, and are heavy spenders. Reward these customers. They can become early adopters for new products and will help promote your brand.	162
BigSpenders	Number of customers who spends a lot of money	243
Loyalists	Number of customers who orders alot and spends a lot of money	204
Potential Loyalists	Number of customers with average order frequency and who spend a good amount.	502
At Risk	Number of customers who are not ordering from quite some time but they used to spend a lot.	33

Figure 19: Customer Segment details created based on RFM Analysis

7. Conclusion

- The highest revenue was generated in January. Since then the sales have been dropping.
- 2nd day of a month is most important to generated high revenue.
- Most orders are placed at the starting of the weekend i.e. Saturday.
- The time window from 10 AM to 12 PM and 2 PM to 4 PM is extremely important. This is the time when we can create the most noise and get the complete attention of the customer.
- Some products help to generate high revenue while some products are sold most. Both types of products are important for better customer engagement.
- The highest revenue is generated from the Women's category followed by the Makeup category.
- Brand C has created the highest revenue and is the most revenue-generating brand from the Women and Makeup category both.
- Using RFM analysis, customer segments were created. Based on the discussion with the domain experts and stakeholders, further segmentation could be created using RFM score or by further application of K-means on Recency, Frequency and Monetary score

References

[1] "RFM Analysis for Customer Segmentation | CleverTap", CleverTap, 2020. [Online]. Available: https://clevertap.com/blog/rfm-analysis/. [Accessed: 29- May- 2020].

[2] "Who Is Your Golden Goose?: Cohort Analysis", Medium, 2020. [Online]. Available: https://towardsdatascience.com/who-is-your-golden-goose-cohort-analysis-50c9de5dbd31. [Accessed: 29- May- 2020].

[3] 2020. [Online]. Available: https://github.com/joaolcorreia/RFManalysis. [Accessed: 30- May- 2020].

[4] "RFM Segmentation using Quartiles and Jenks Natural Breaks", Medium, 2020. [Online]. Available: https://towardsdatascience.com/rfm-segmentation-using-quartiles-and-jenks-natural-breaks-924f4d8baee1. [Accessed: 31s- May- 2020].