Project Report for

Sales Forecasting for a Retail Store in Preparation for Black Friday Sales

Vedanti Patil
Data Management for Data Science
10 July 2024

Table of contents

1. Project Definition
2. How does this project relate to the lectures/papers we discussed?
3. Novelty and Importance
4. Existing Issues in Current Data Management Practices
4. What Kind of Data is Used?
5. How is the Data Analyzed?
6. Exploratory Data Analysis

1. Project Definition

The purpose of this project is to conduct an inventory and sales analysis on a Retail Company's previous 3 years of Black Friday sales data, in order to predict the upcoming year's Black Friday inventory needs. This analysis will determine which products were most purchased from each respective product category, from each individual store location, as well as which were least purchased.

2. How does this project relate to the lectures/papers we discussed?

This project relies heavily on the knowledge we learned in this class, specifically on what we learned about the process of cleaning and transforming data to make the data meaningful to an insightful analysis. In this case, this project takes arbitrary sales data from a company, muddled with a lot of unnecessary information, and uses this data to create an output that can boost the company's future sales and revenue. This is done using an extensive series of queries on this data through MySQL, and the transformed data is then modeled and graphed using Python and Matplotlib, which are both key things we covered in this course.

3. Novelty and Importance

The analysis done in this project has several real-world and practical applications that can boost a real company's revenue if implemented:

1. **Boosting Sales Opportunities by Meeting Customer Demands:**
    - An accurate and detailed forecast can help avoid having out-of-stock products, thereby reducing a loss of sales on high-demand products.
    - Having high-demand products in stock will create higher customer satisfaction, making it more likely for the customer to keep shopping at the retailer in the future.

2. **Avoiding Overstocking:**
    - Effectively avoiding overstocking on specific products can lead to multiple benefits:
        - Reduced holding costs in storage, as overstock takes up unnecessary warehouse space.
        - Reduced capital spent on buying excess inventory, just for it to later be marked down and lead to a loss.

3. **Creating a Competitive Edge:**
    - Preparing for Black Friday sales with an accurate and effective forecast can give the retailer a competitive advantage over others, as they will be more prepared to meet customer demands.

4. **Planning Black Friday Specific Deals:**
    - Having an accurate forecast of which products are most frequently purchased can help the retailer make decisions on which products to have promotions, sales, and discounts on.

5. **Streamlining Inventory:**
    - In high-volume cases such as that of Black Friday, an accurate forecast can even lead to streamlining general operations on inventory all the way from inventory acquisition to sales.
    - Knowing the general quantities in which inventory is needed can help prepare in the product creation process itself.

4. Existing Issues in Current Data Management Practices

Currently, there are several issues that exists when it comes to the analysis of data in the context of Black Friday sales:

1. **Forecasting issues:** Due to rapid change in the market and consumer behavior, out-of-date data and historical analysis can lead to inaccurate projections. Additionally, since Black Friday sales are so influenced by external factors, it is necessary to analyze data specific to this event for accuracy.

2. **Scalability challenges:** Due to the high volume and velocity of data during Black Friday, there is a large issue with processing such saturated data efficiently and extracting useful information from such high-volume data.

3. **Data quality:** Having redundant data and unnecessary information related to sales can lead to increased time in making necessary and accurate projections.

4. **Data integration:** Since there are several diverse data sets that retailers make their predictions from, such as in-store sales, online sales, third-party vendors, etc., integrating all of these data sets can be arduous and complex.

Although this project can not solve all of these complex issues, it aims to solve the issue of data quality and forecasting, ensuring relevant and necessary data is used to make insightful and clear predictions and judgements.

5. What Kind of Data is Used?

Using a relational database on MySQL, the following 4 tables are created, serving as the primary data for this project:

1. **Sales Data:** A dataset from Kaggle titled “Black Friday Sales Data” is used as a foundational data source, being uploaded onto MySQL (some of the columns are later removed, thus not shown in the table displayed). There are 5000 transactions used from this table.

SQL query along with sample data

```
2     #create sales data table#
3     CREATE TABLE sales_data (
4         transaction_id INTEGER PRIMARY KEY,
5         product_id TEXT,
6         gender TEXT,
7         age TEXT,
8         occupation INTEGER,
9         city_category TEXT,
10        marital_status INTEGER,
11        product_category1 INTEGER,
12        product_category2 INTEGER,
13        product_category3 INTEGER,
14        purchase INTEGER
15        );
16
17    SELECT * FROM sales_data;
```

esult Grid | Filter Rows: | Edit: | Export/Import:

transaction_id	product_id	city_category	product_category1	product_quantity
1	P00069042	A	3	1
2	P00248942	A	14	7
3	P00087842	A	4	4
4	P00085442	A	12	12
5	P00285442	C	3	5
6	P00193542	A	3	3
7	P00184942	B	1	3
8	P00346142	B	2	4

2. **Category Master:** Then, based on the numbered categories given in the Sales Data table (#1-18), and using the help of ChatGPT to generate random product category names, a Category Master Table is created, storing the category number and its respective category name.

```
19      #create category master table#
20  •   CREATE TABLE category_master (
21          category_id INTEGER PRIMARY KEY,
22          category_name TEXT
23      );
```

Using the logic shown below, data in the Category Master is populated:

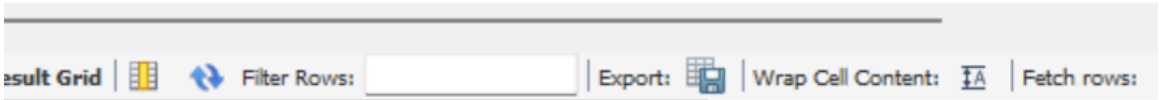
```
27      #find out how many unique product categories exist in sales table#
28  •   SELECT DISTINCT product_category1 from sales_data ORDER BY product_category1 DESC;
29
30      #given unique product categories, enter category id and created name into table#
31  •   INSERT INTO category_master (category_id, category_name) VALUES
32      (1, "Electronics"),
33      (2, "Clothing"),
34      (3, "Home Appliances"),
35      (4, "Furniture")
```

esult Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

category_id	category_name
1	Electronics
2	Clothing
3	Home Appliances
4	Furniture
5	Groceries
6	Toys
7	Books
8	Sports Equipment
9	Beauty Products
10	Health Products
11	Automotive
12	Jewelry
13	Footwear
14	Office Supplies

3. **Product Master:** Then, based on the unique product ids (1748 values) given in the Sales Data table, and once again the help of ChatGPT to generate unique product names in accordance with the given categories, a Product Master Table is created:

```
53      #find unique product ids from sales data table#
54 •    SELECT COUNT(DISTINCT product_id) from sales_data;
55
56      #create product master table#
57 •    CREATE TABLE product_master (
58          product_id TEXT,
59          product_name TEXT,
60          category_id TEXT,
61          unit_price INTEGER
62      );
63
64 •    select * from product_master;
65
```



product_id	product_name	category_id	unit_price
P00069042	Refrigerator 19	3	8009
P00248942	Pencil Sharpener 69	14	11429
P00087842	Bed 20	4	1422
P00085442	Cufflinks 26	12	1057
P00285442	Food Processor 84	3	5855
P00193542	Air Conditioner 42	3	7897

4. **Inventory Data:** Finally, using the product ids from before, and the locations A, B, C, a cartesian product of the two is done to get a table with each product at each of the three cities and the randomized stock quantity of each:

```
84      #create inventory data table#
85  •   CREATE TABLE inventory_data (
86          product_id TEXT,
87          location TEXT,
88          stock_quantity INTEGER
89      );
90
91  •   SELECT * FROM inventory_data;
92
```

Result Grid | Filter Rows: | Export:

product_id	location	stock_quantity
P00000142	A	23
P00000142	B	138
P00000142	C	18
P00000342	A	130
P00000342	B	145

```
93      #populating columns of inventory data#
94  •   CREATE TEMPORARY TABLE product_locations
95      AS
96      SELECT product_id, 'A' AS Location FROM (
97          SELECT DISTINCT product_id FROM sales_data
98      ) AS product_ids
99      UNION
00      SELECT product_id, 'B' AS Location FROM (
01          SELECT DISTINCT product_id FROM sales_data
02      ) AS product_ids
03      UNION
04      SELECT product_id, 'C' AS Location FROM (
05          SELECT DISTINCT product_id FROM sales_data
06      ) AS product_ids;
07
08  •   SELECT * FROM product_locations ORDER BY product_id;
09
10  •   INSERT INTO inventory_data (product_id, location)
11      SELECT product_id, Location FROM product_locations ORDER BY product_id;
12
13  •   UPDATE inventory_data
14      SET stock_quantity = FLOOR(RAND() * 151);
15
16  •   SELECT * FROM inventory_data;
```

6. How is the Data Analyzed?

5.1 Data Collection: Data collection was done as mentioned in the section above.

5.2 Data Storage: All the tables were created in a MySQL DB, to allow for efficient queries and data cleaning/transformation.

5.3 Data Cleaning:

The below unnecessary columns were removed from the sales_data table, columns were altered as needed, and missing values were updated using update queries:

```
19      #cleaning the data from initial sales_data table#
20
21 •    ALTER TABLE sales_data
22      DROP COLUMN gender,
23      DROP COLUMN age,
24      DROP COLUMN occupation,
25      DROP COLUMN marital_status,
26      DROP COLUMN purchase;
27
28 •    ALTER TABLE sales_data
29      DROP COLUMN product_category2,
30      DROP COLUMN product_category3;
```

```
#creating quantity column for sales_data table#
•  ALTER TABLE sales_data
   ADD COLUMN product_quantity INT;
•  UPDATE sales_data
   SET product_quantity = FLOOR(RAND() * 15);

#updating category on sales data products according to product master#
•  UPDATE sales_data sd
   JOIN product_master pm
   ON sd.product_id = pm.product_id
   SET sd.product_category1 = pm.category_id;
```

5.4 Data Transformation: Through extensive transformation, the combined_sales_data table was created using the previously created sales and master tables. Primary and Foreign key relationships were used to join different tables and fetch necessary data from each table as shown in the steps below:

- **Step 1**: A Three Years Sales mock data was created in xlsx, by taking the cleaned sales_data table and randomly percent-increasing/decreasing each category of products' sales over the span of the 2 following years. This data was then uploaded to the MySQL table below:

```
#using sales data table to create a mock 3-year sales data in xlx, uploading that here#
CREATE TABLE three_years_sales_data (
    sales_year INT,
    transaction_id INTEGER,
    product_id TEXT,
    city_category TEXT,
    product_category1 INTEGER,
    product_quantity INTEGER,
    PRIMARY KEY (transaction_id)
    );

SELECT * FROM three_years_sales_data;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

sales_year	transaction_id	product_id	city_category	product_category1	product_quantity
2021	1	P00069042	A	3	1
2021	2	P00248942	A	14	7
2021	3	P00087842	A	4	4
2021	4	P00085442	A	12	12
2021	5	P00285442	C	3	5
2021	6	P00193542	A	3	3
2021	7	P00184942	B	1	3
2021	8	P00346142	B	2	4

Step 2: Created a base combined_sales_date table using three_years_sales_data created above:

```
CREATE TABLE combined_sales_data AS
SELECT * FROM three_years_sales_data;
```

Step 3: Added product_unit_price column into combined_sale_data table and populated this data by joining this table with the product_master table, using product_id as a primary key to map between the 2 tables:

```
#add product unit price#
ALTER TABLE combined_sales_data
ADD COLUMN product_unit_price INT;
UPDATE combined_sales_data csd
JOIN product_master pm
ON csd.product_id = pm.product_id
SET csd.product_unit_price = pm.unit_price;
```

Step 4: Added total_purchase_amount column into the combined_sale_data table and populated the data using the transformation logic shown below:

```
#add transaction purchase amount#
ALTER TABLE combined_sales_data
ADD COLUMN total_purchase_amount INT;
UPDATE combined_sales_data
SET total_purchase_amount = product_quantity * product_unit_price;
```

Step 5: Added product_name column into the combined_sale_data table and populated the data using the transformation logic shown below:

```
#add product name next to product id#
ALTER TABLE combined_sales_data
ADD COLUMN product_name TEXT
AFTER product_id;
UPDATE combined_sales_data csd
JOIN product_master pm
ON csd.product_id = pm.product_id
SET csd.product_name = pm.product_name;
```

Step 6: Added category_name column into the combined_sale_data table and populated data using the transformation logic shown below:

```
#add category name next to category id#
ALTER TABLE combined_sales_data
ADD COLUMN category_name TEXT
AFTER product_category1;
UPDATE combined_sales_data csd
JOIN category_master cm
ON csd.product_category1 = cm.category_id
SET csd.category_name = cm.category_name;
```

After all of the transformation, the below combined table is ready for data analysis.

```
203 •  SELECT * FROM combined_sales_data;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

sales_year	transaction_id	product_id	product_name	city_category	product_category1	category_name	product_quantity	product_unit_price	total_purcha
2021	1	P00069042	Refrigerator 19	A	3	Home Appliances	1	8009	8009
2021	2	P00248942	Pencil Sharpener 69	A	14	Office Supplies	7	11429	80003
2021	3	P00087842	Bed 20	A	4	Furniture	4	1422	5688
2021	4	P00085442	Cufflinks 26	A	12	Jewelry	12	1057	12684
2021	5	P00285442	Food Processor 84	C	3	Home Appliances	5	5855	29275
2021	6	P00193542	Air Conditioner 42	A	3	Home Appliances	3	7897	23691
2021	7	P00184942	USB Flash Drive 107	B	1	Electronics	3	11643	34929
2021	8	P00346142	Scarf 88	B	2	Clothing	4	12110	48440
2021	9	P0097242	Yoga Mat 93	B	8	Sports Equipment	14	7668	107352
2021	10	P00274942	Aquarium Filter 84	A	15	Pet Supplies	12	4102	49224
2021	11	P00251242	Yo-Yo 16	A	6	Toys	2	2029	4058
2021	12	P00014542	Cello 61	A	17	Music Instruments	8	3957	31656
2021	13	P00031342	Flute 22	A	17	Music Instruments	2	6073	12146
2021	14	P00145042	Gaming Monitor 8	A	18	Video Games	0	11457	0
2021	15	P00231342	Face Wash 9	A	9	Beauty Products	12	5378	64536

7. Exploratory Data Analysis

Now, using the previously created combined_sales_data table, the following steps are taken to do thorough analysis on this data.

Step1: Using the combined dataset, a sales_summary table is created with aggregated values that for each location, show the number of items of a certain product that were sold, for all the products listed.

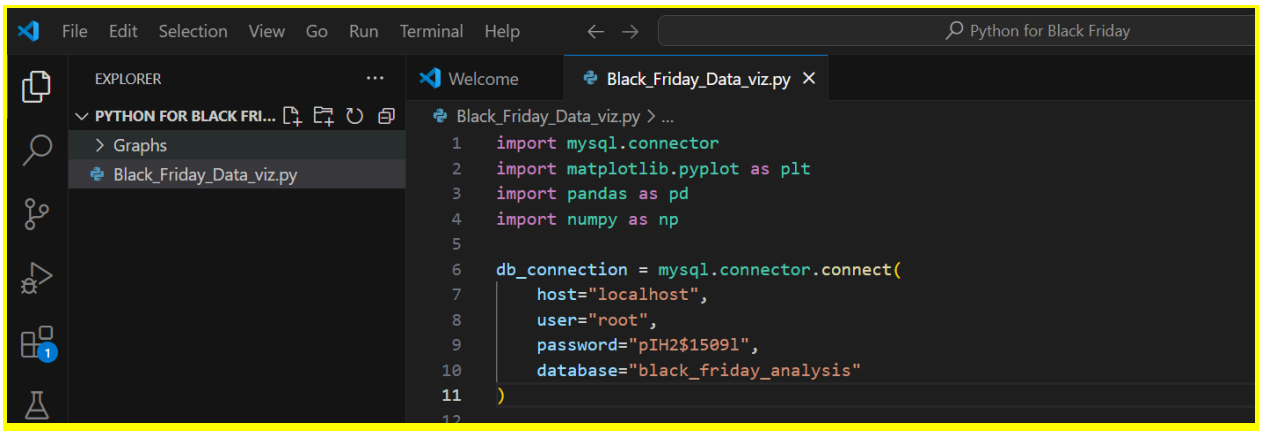
```sql
#create an aggragated table, which by city, shows the total quantity of a product purchased for each product#
CREATE TABLE sales_summary AS
SELECT
    sales_year,
    city_category,
    product_id,
    product_name,
    SUM(product_quantity) AS total_quantity,
    SUM(total_purchase_amount) AS total_amount_dollars
FROM
    combined_sales_data
GROUP BY
    sales_year,
    city_category,
    product_id,
    product_name;

SELECT * FROM sales_summary ORDER BY sales_year ASC, city_category ASC;
```

t Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

ales_year	city_category	product_id	product_name	total_quantity	total_amount_dollars
)21	A	P00166042	Video Doorbell 84	9	47007
)21	A	P00086242	Clarinet 80	7	131187
)21	A	P00083942	Bracelet 2	7	9828
)21	A	P00367142	Gaming Chair 4	14	20972
)21	A	P00316242	Surfboard 106	8	14744
)21	A	P00102842	Piano 1	11	45441
)21	A	P00149342	Face Wash 99	6	4572

Step 2: Using this table and some queries run on the combined_sales_data table, several data analysis graphs and regression plots are created using Python in VS code and the Matplotlib Library.

A. In Python - DB connection string is created:



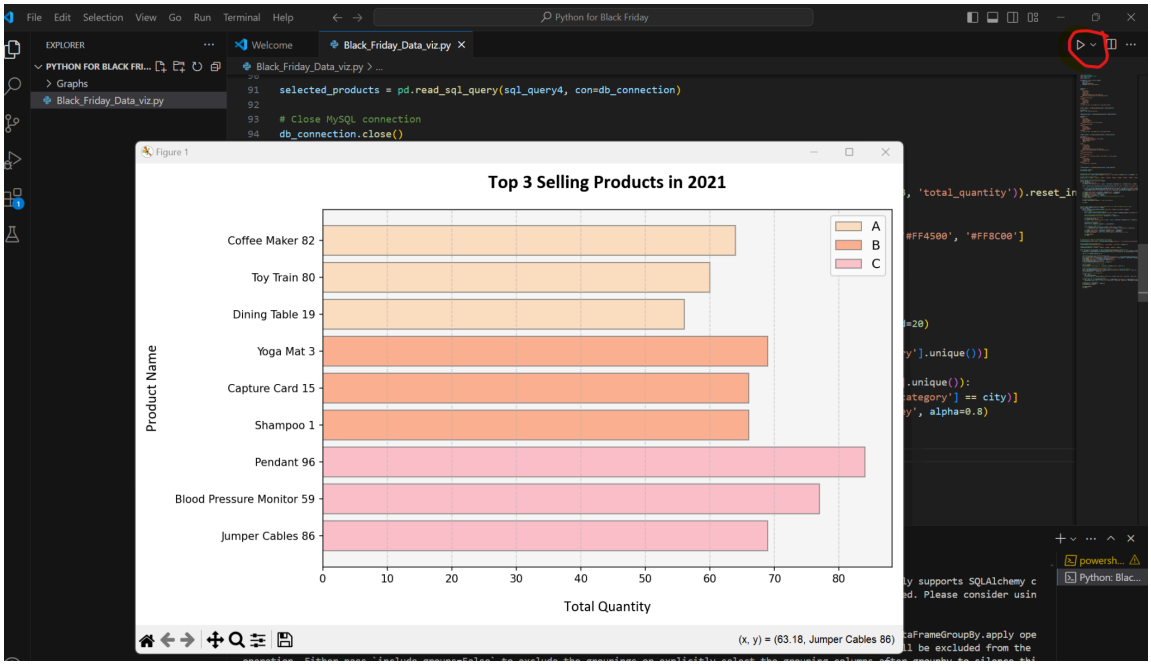
B. In Python code, multiple sql queries are written to create data sets for each analysis point (refer attached python code file along with this project submission for more details).

C. Various charts are created from the dataframes obtained from these queries, an example of the code for one of the graphs is shown.

(refer attached python code file along with this project submission for more details)

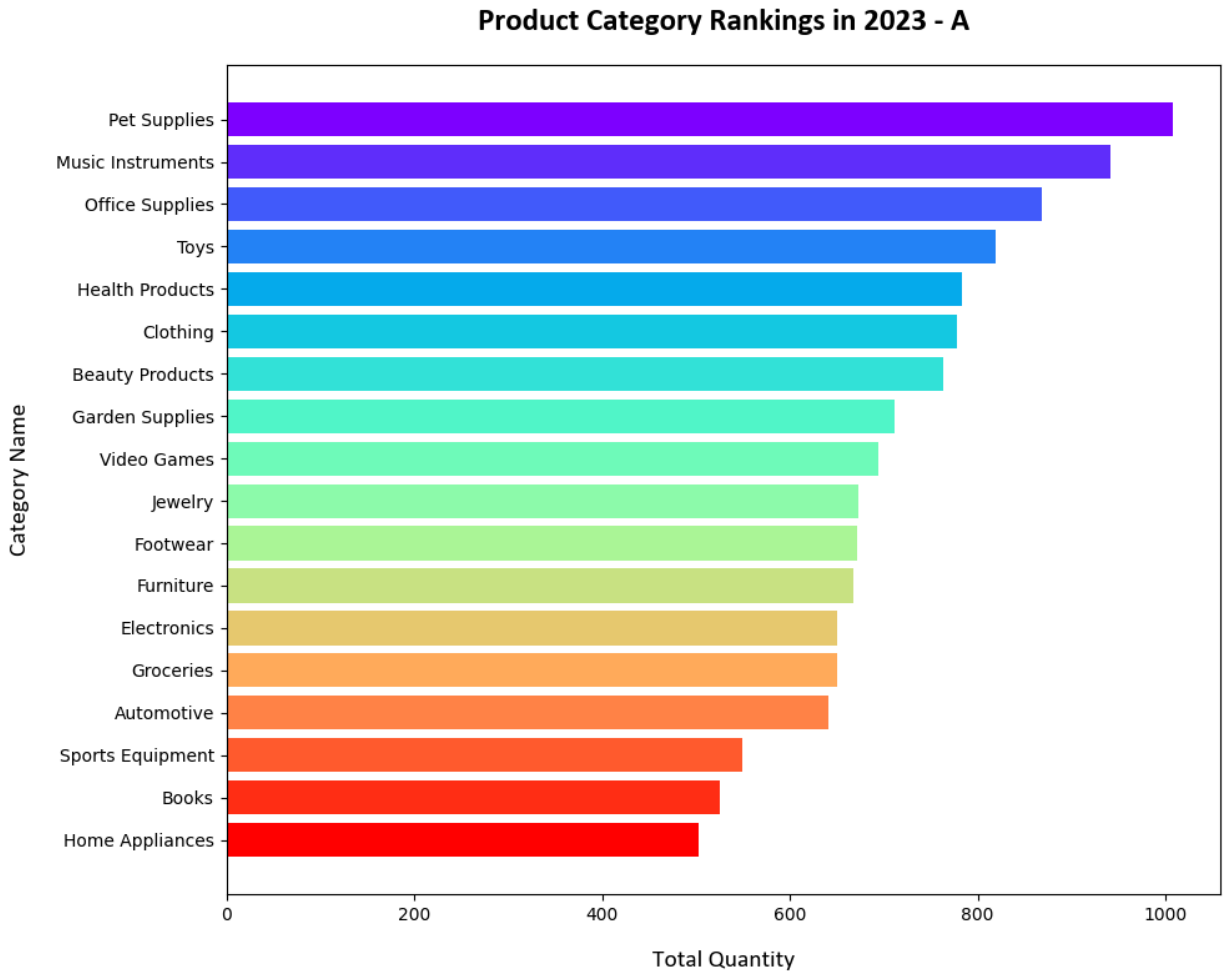
```python
# Process data to get top 3 selling products per year by city
top3_per_year_city = product_summary.groupby(['sales_year', 'city_category']).apply(lambda x: x.nlargest(3, 'total_quantity')).reset_in

# Create a pastel color palette
pastel_colors = ['#FFD8B1', '#FFA07A', '#FFB6C1', '#FFC0CB', '#FFD700', '#FF69B4', '#FFE4E1', '#FF6347', '#FF4500', '#FF8C00']

# Create a plot for top 3 each year by city
years = top3_per_year_city['sales_year'].unique()
for year in years:
    plt.figure(figsize=(10, 6))
    plt.title(f'Top 3 Selling Products in {year}', fontsize=18, fontweight='bold', fontname='Calibri', pad=20)

    colors = pastel_colors[:len(top3_per_year_city[top3_per_year_city['sales_year'] == year]['city_category'].unique())]

    for i, city in enumerate(top3_per_year_city[top3_per_year_city['sales_year'] == year]['city_category'].unique()):
        data = top3_per_year_city[(top3_per_year_city['sales_year'] == year) & (top3_per_year_city['city_category'] == city)]
        plt.barh(data['product_name'], data['total_quantity'], label=city, color=colors[i], edgecolor='grey', alpha=0.8)

    plt.xlabel('Total Quantity', fontsize=14, fontname='Calibri', labelpad=15)
    plt.ylabel('Product Name', fontsize=14, fontname='Calibri', labelpad=15)
    plt.legend(loc='upper right', fontsize=12)
    plt.grid(axis='x', linestyle='--', alpha=0.5)
    plt.gca().invert_yaxis()  # Invert y-axis to show top selling products at the top
    plt.tight_layout()

    # Customize background color
    plt.gca().set_facecolor('#F5F5F5')  # Light grey background

    plt.show()
```

D. When the Python code is executed using the Run button highlighted in red, the expected charts are displayed sucessfully one after another.

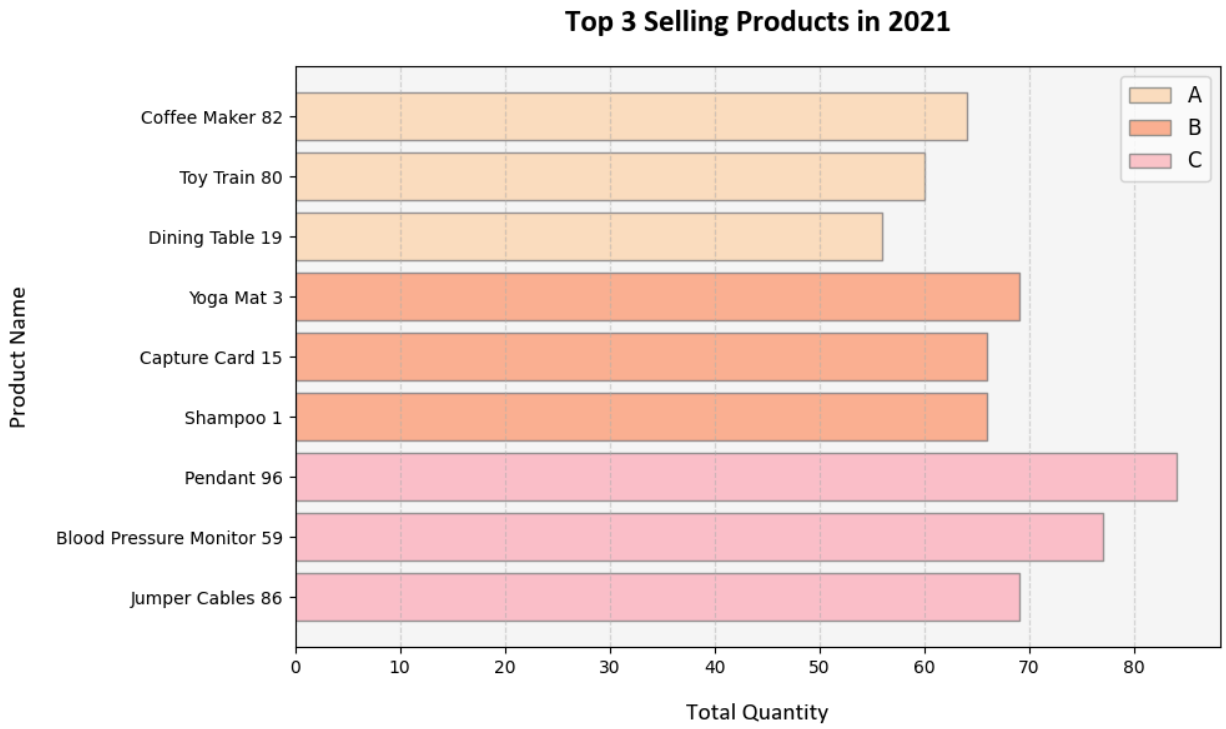


E. Below are sample charts generated from my analysis

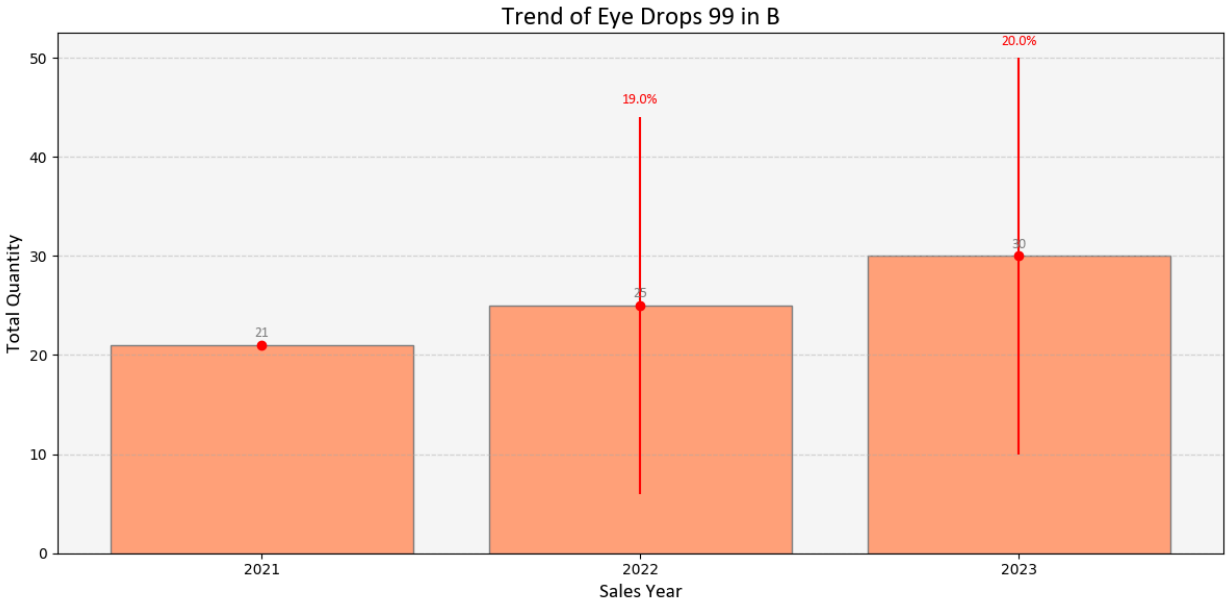
1. **City-wise category rankings for last three years** (chart for one city in one year is shown as an example):



2. **Top 3 Selling Products in Each City by Year** (chart for one year is shown as an example):



3. **3-Years Sales Trend For Each Product** (chart for one such product is shown as an example)



Step 3: Forecasting/Projection of the proposed sales inenvtory for the upcoming year

A. Using the aforementioned analysis with the trends in each product's sales, the below table is created outlining the projected stock expectancy for each product, meaning the # of items that will be needed of that product in stock for next year's sale, based on the increasing/decreasing trends in sale from the previous 3 years.

```
275 •  CREATE TABLE projected_values AS
276    SELECT
277        city_category,
278        product_id,
279        product_name,
280        SUM(CASE WHEN sales_year = 2021 THEN total_quantity ELSE 0 END) AS quantity_2021,
281        SUM(CASE WHEN sales_year = 2022 THEN total_quantity ELSE 0 END) AS quantity_2022,
282        SUM(CASE WHEN sales_year = 2023 THEN total_quantity ELSE 0 END) AS quantity_2023
283    FROM
284        sales_summary
285    GROUP BY
286        city_category,
287        product_id,
288        product_name;
289
290 •  SELECT * FROM projected_values ORDER BY city_category ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

city_category	product_id	product_name	quantity_2021	quantity_2022	quantity_2023	projected_quantity_2024	inventory_2024	stock_adjustment
A	P00282342	Shampoo 55	14	17	18	20	39	-19
A	P00257842	Oxfords 98	7	7	9	10	128	-118
A	P00303942	Clipboard 84	3	3	4	5	91	-86
A	P00212442	Desk Lamp 75	8	9	11	13	78	-65
A	P00354842	Cereal 97	12	15	19	24	149	-125
A	P00296042	Cat Toys 43	13	14	15	16	142	-126
A	P00156042	Surfboard 34	2	2	2	2	61	-59
A	P00191342	Blouse 7	4	5	6	7	96	-89
A	P00339942	TV Stand 28	6	7	7	8	109	-101
A	P00057942	Controller 19	3	3	3	3	112	-109

projected_values 40

B. By creating the below temporary table and logic, the calculation of % change in the sales for each year was done:

```
#Calculate percent change for each year#
CREATE TEMPORARY TABLE pct_changes AS
SELECT
    city_category,
    product_id,
    product_name,
    quantity_2021,
    quantity_2022,
    quantity_2023,
    ((IFNULL(quantity_2022, 0) - IFNULL(quantity_2021, 0)) / NULLIF(quantity_2021, 0)) * 100 AS pct_change_2021_2022,
    ((IFNULL(quantity_2023, 0) - IFNULL(quantity_2022, 0)) / NULLIF(quantity_2022, 0)) * 100 AS pct_change_2022_2023
FROM
    projected_values;

SELECT * FROM pct_changes;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

city_category	product_id	product_name	quantity_2021	quantity_2022	quantity_2023	pct_change_2021_2022	pct_change_2022_2023
A	P00069042	Refrigerator 19	11	11	11	0.0000	0.0000
A	P00248942	Pencil Sharpener 69	18	20	25	11.1111	25.0000
A	P00087842	Bed 20	4	5	5	25.0000	0.0000
A	P00085442	Cufflinks 26	12	12	14	0.0000	16.6667
C	P00285442	Food Processor 84	12	12	12	0.0000	0.0000

D. Then another temporary table was created to calculate the average annual growth rate:

```
#Calculate the average annual growth rate (AAGR)#
CREATE TEMPORARY TABLE growth_rates AS
SELECT
    city_category,
    product_id,
    product_name,
    quantity_2021,
    quantity_2022,
    quantity_2023,
    (IFNULL(pct_change_2021_2022, 0) + IFNULL(pct_change_2022_2023, 0)) / 2 AS avg_annual_growth_rate
FROM
    pct_changes;

SELECT * FROM growth_rates;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

city_category	product_id	product_name	quantity_2021	quantity_2022	quantity_2023	avg_annual_growth_rate
A	P00069042	Refrigerator 19	11	11	11	0.00000000
A	P00248942	Pencil Sharpener 69	18	20	25	18.05555000
A	P00087842	Bed 20	4	5	5	12.50000000
A	P00085442	Cufflinks 26	12	12	14	8.33335000
C	P00285442	Food Processor 84	12	12	12	0.00000000

E. Then, the existing projected_values table was updated by joining it with the growth_rates table to update projected product quantities for upcoming black friday sale in 2024:

```
#Project the quantity for 2024 using the AAGR and add it to the projected_values table#
UPDATE projected_values pv
JOIN growth_rates gr ON pv.city_category = gr.city_category AND pv.product_id = gr.product_id
SET pv.projected_quantity_2024 = IFNULL(gr.quantity_2023, 0) * (1 + (IFNULL(gr.avg_annual_growth_rate, 0) / 100));

SELECT * FROM projected_values ORDER BY product_id ASC, city_category ASC;
```

ult Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

city_category	product_id	product_name	quantity_2021	quantity_2022	quantity_2023	projected_quantity_2024	inventory_2024	stock_adjustment
B	P00000142	Rice 6	17	21	27	34	138	-104
C	P00000142	Rice 6	56	70	90	114	18	+96
B	P00000342	Electric Grill 87	2	2	2	2	145	-143

F. Then, the stock_adjustment column in the projected_values table was updated by calculating project_quantity - inventory for each year as below:

```
ALTER TABLE projected_values MODIFY COLUMN stock_adjustment VARCHAR(20);

UPDATE projected_values
SET stock_adjustment = CASE
    WHEN projected_quantity_2024 - inventory_2024 > 0 THEN CONCAT('+', projected_quantity_2024 - inventory_2024)
    ELSE projected_quantity_2024 - inventory_2024
END;
```

G. Finally, the projected_values table is updated which effectively compares the projected stock expectancy (based on the calculations done) to the current inventory available (from the Inventory Data), and gives, for each product, a value by which they need to increase/decrease their current stock to match this projected expectancy.

city_category	product_id	product_name	quantity_2021	quantity_2022	quantity_2023	projected_quantity_2024	inventory_2024	stock_adjustment
B	P00000142	Rice 6	17	21	27	34	138	-104
C	P00000142	Rice 6	56	70	90	114	18	+96
B	P00000342	Electric Grill 87	2	2	2	2	145	-143
B	P00000442	RC Car 82	4	5	6	7	73	-66
A	P00000542	Plant Pot 56	6	7	7	8	82	-74
B	P00000542	Plant Pot 56	3	3	3	3	31	-28
A	P00000642	Skateboard 105	8	9	11	13	35	-22
B	P00000642	Skateboard 105	35	40	48	56	147	-91
C	P00000642	Skateboard 105	16	18	21	24	80	-56
A	P00001042	Gaming Monitor 44	8	9	10	11	60	-49
B	P00001042	Gaming Monitor 44	17	18	19	20	40	-20
B	P00001142	Action Figure 0	3	4	5	6	64	-58
C	P00001142	Action Figure 0	10	12	14	17	89	-72
B	P00001242	Juice 9	12	15	19	24	63	-39
C	P00001242	Juice 9	8	10	13	17	75	-58
B	P00001542	Slow Cooker 67	6	6	6	6	115	-109
A	P00001642	Harmonica 63	6	6	7	8	118	-110
B	P00001642	Harmonica 63	14	14	18	21	133	-112
C	P00001642	Harmonica 63	12	12	15	17	7	+10
A	P00001742	Harmonica 9	11	11	14	16	125	-109
C	P00001742	Harmonica 9	23	23	28	31	56	-25
C	P00001942	Cat Scratcher 83	12	13	14	15	117	-102
A	P00002042	Spark Plug 41	6	7	8	9	55	-46

Final Conclusions:

As seen in the table above, it is apparent that there are stock adjustments necessary for a lot of the products shown in just the head of this table (out of 3000 rows). For example, many of the rows shown above call for a stock reduction, illustrating that currently, those products are overstocked.

This analysis can be very helpful for the retailer, as they will now know how to adjust their stock accordingly, whether that means an increase/decrease in quantity, shifting stock from one location to the other, etc. For example, in the first two rows of this table, we can see that for the Rice 6 product, there is an overstock in City B and an understock in City C, so that shift can be made.

Additionally, the various charts and graphs created before will also be very insightful to the retailer, accurately and specifically showing the various trends in each product, the highest selling products and categories at each location, etc.

Therefore, this analysis is deemed as successful and complete. Though, there are a couple of limitations to the approach taken here, such as using several queries to do a task step by step instead of the use of more complex queries to get the job done faster. This was solely done for the purpose of readability and clarity in the logic executed.

Other than that, the project was implemented as per initial proposal, with no major changes.

Note: I have attached my sql scripts and python code for reference along with the final project report in the canvas submission. If required, a running demo of the project can be done.