INFOCEPTS DATA SCIENCE BASED PROJECT

Dissertation submitted to
Shri Ramdeobaba College of Engineering &
Management, Nagpur in partial fulfillment of
requirement for the award of
degree of

Bachelor of Engineering

In

Electronics and Computer Science

By

Deepshikha Singh (A-01)

Aaryan Babuta (A-02)

Muskan Asudani (A-06)

Aditya Shukla (A-42)

Guide
Dr. Parag Jawarkar

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

Electronics and Computer Science
Shri Ramdeobaba College of Engineering & Management,
Nagpur 440 013

(An Autonomous Institute affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur)

May 2025

Warehouse and Retail Sales Analysis

Deepshikha Singh
Electronics and Computer Science
Shri Ramdeobaba College of Engineering and Management
Nagpur, India

Aaryan Babuta
Electronics and Computer Science
Shri Ramdeobaba College of Engineering and Management
Nagpur, India

Muskan Asudani
Electronics and Computer Science
Shri Ramdeobaba College of Engineering And Management
Nagpur, India

Aditya Shukla
Electronics and Computer Science
Shri Ramdeobaba College Of Engineering And Management
Nagpur, India

Abstract

In the highly competitive retail industry, understanding historical sales data is essential for making informed business decisions. *Warehouse and Retail Sales Analysis*, aims to derive actionable insights from a dataset containing sales records from both warehouse and in-store channels over multiple years. The project focuses on analysing supplier performance, product demand, seasonal sales patterns, and item-type contributions. We used data analytics tools like Python and SQL to clean, preprocess, and analyse the data. Visualizations were employed to support insights and help the company optimize inventory management, supplier engagement, and promotional strategies. This case study demonstrates how a data-driven approach can support smarter, faster, and more effective retail decisions.

Introduction

Retail businesses face increasing complexity in managing product portfolios, supplier relationships, and customer demand. The company in question operates through two primary channels: physical retail stores and a central warehouse that supplies those stores. The available dataset, sourced from the US Government's open data portal, contains multi-year sales data categorized by item type, product name, supplier, and date.

Analysing this historical data can help answer crucial business questions, such as:

- Which products are most profitable?
- Which suppliers offer the most reliable and consistent supply?
- Are there seasonal trends that affect sales?
- Which item types drive the most revenue?

This report outlines the step-by-step analytical approach taken to answer these questions and propose recommendations to enhance overall business performance.

Objectives

The primary objectives of this project are:

1. **Sales Channel Analysis-** Calculate and visualise each year's total sales (retail and warehouse)
2. **Supplier Performance Evaluation-** Identify the top suppliers based on overall contribution to sales and consistency in supply.
3. **Product Trend Analysis-** Highlight the top 10 high-performing products based on quantity sold and revenue generated.
4. **Seasonality & Temporal Trends-** Identify patterns in sales over months and years to uncover seasonality or cyclic behaviour in demand.
5. **Actionable Recommendations-** Use the insights from data analysis to propose strategies for better inventory management, supplier selection, and promotional timing.

Methodology

Our approach is based on a structured data analysis pipeline involving the following phases:

1. **Data Collection**
   - **Source:** Open US Government Data Portal (https://catalog.data.gov/dataset/warehouse-and-retail-sales)
   - **Format:** CSV Format containing columns- Year, Month, Supplier, Item Code, Item Description, Item Type, Retail Sales, Retail Transfers, Warehouse Sales.

2. **Data Cleaning and Preprocessing**
   - **Missing Values:** Checked for and handled missing or null entries in key columns such as Supplier, Item Name, and Quantity.
   - **Duplicates**: Removed duplicate entries that could skew totals.

3. **Tools and Technologies Used**
   - **Python (Pandas, NumPy)** for data manipulation.
   - **Matplotlib & Seaborn** for visual analytics.
   - **Jupyter Notebook** for code development and documentation.
   - **SQL (optional)** for structured queries and cross-tab aggregations.

4. **Exploratory Data Analysis (EDA)**
   - **Grouping and Aggregation**: Grouped data by Item Type, Supplier, Year, and Source, etc.
   - **Trend Analysis**: Line charts for monthly/yearly trends.
   - **Supplier Ranking**: Based on total sales contribution.
   - **Product-Level Insights**: Identified top-selling products across different years.
   - **Visualizations**: Used bar plots, pie charts, etc. to illustrate findings.

Suggestions/Recommendations

Based on the insights derived from the analysis, we propose the following strategic recommendations:

1. **Strengthen Supplier Relationships**
   - Focus more on high-performing suppliers that consistently deliver products with high sales volumes.
   - Establish long-term contracts or preferred vendor agreements to maintain consistent stock.

2. **Rationalize Product Portfolio**
   - Identify and remove underperforming products that occupy shelf space but do not contribute significantly to sales.
   - Replace them with high-demand or trending items based on the sales data.

3. **Leverage Seasonal Sales Patterns**
   - Introduce seasonal promotions or discount campaigns during months with historically high sales volumes.
   - Increase inventory of top products before seasonal spikes to avoid stockouts.

4. **Optimize Store vs Warehouse Inventory**
   - Identify products that sell better in physical stores than through the warehouse, and adjust logistics accordingly.
   - Implement channel-specific strategies for pricing, marketing, and stocking.

5. **Invest in Predictive Planning**
   - Use past sales trends to forecast future demand by item type and supplier.
   - Plan bulk orders or promotions in advance to take advantage of predictable trends.

Result

From our data-driven analysis, we obtained the following results:

1. **Top Suppliers Identified**:
   [CROWN IMPORTS, MILLER BREWING COMPANY, ANHEUSER BUSCH INC, HEINEKEN USA and E & J GALLO WINERY]
   A few suppliers contributed maximum of the total quantity sold. Their consistency across years indicates strong reliability.

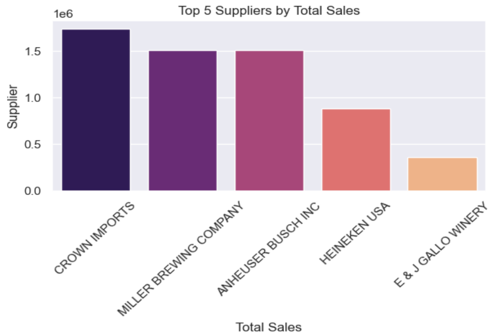


Figure-1: Top 5 Suppliers Based on Total Sales

2. **Top 10 Products Recognized**:

   [CORONA EXTRA LOOSE NR -12OZ, CORONA EXTRA 2/12 NR – 12OZ, HEINEKEN LOOSE NR - 12OZ, HEINEKEN 2/12 NR - 12OZ, MILLER LITE 30PK CAN - 12OZ, CORONA EXTRA 4/6 NR - 12OZ, MODELO ESPECIAL 24 LOOSE NR - 12OZ, BUD LIGHT 30PK CAN, HEINKEN 4/6 NR - 12OZ, CORONA EXTRA 18PK NR - 12OZ]

   These products not only had the highest quantities sold but also demonstrated consistent performance year after year.

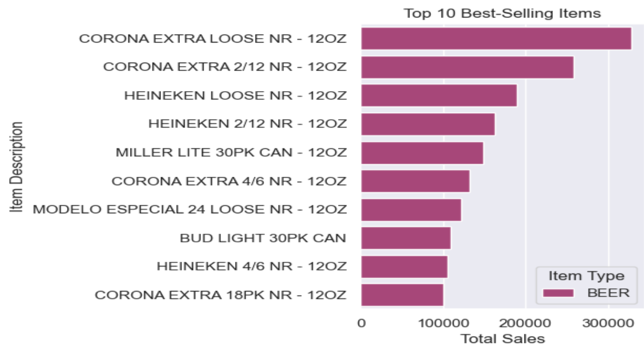


Figure-2: Top 10 Bestseller Item Description based on Total Sales

3. **Seasonal Trends Found**:

   Consistent increase in sales was identified in months like May to August, while retail sales peaked in March 2020 and warehouse sales peaked in July 2020. Months of January and February most show a decline.

Figure-3: Monthly Average Retail Sales

Figure-4: Monthly Average Warehouse Sales

4. **Item-Type Performance**:

   Certain categories, such as [BEER, LIQUOR, WINE], consistently showed high revenue contribution.

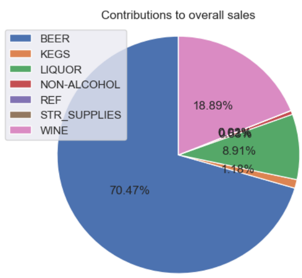


Figure-5: Item Type and its Contribution to Overall Sales

5. **Yearly Trend**:
   The graph shows a clear alternate-year pattern in sales, with increases in odd-numbered years followed by declines in even-numbered ones, suggesting a cyclical or seasonal influence. This trend may be driven by factors such as consumer purchasing cycles, market conditions, or internal business strategies, and should be considered in future planning and forecasting.

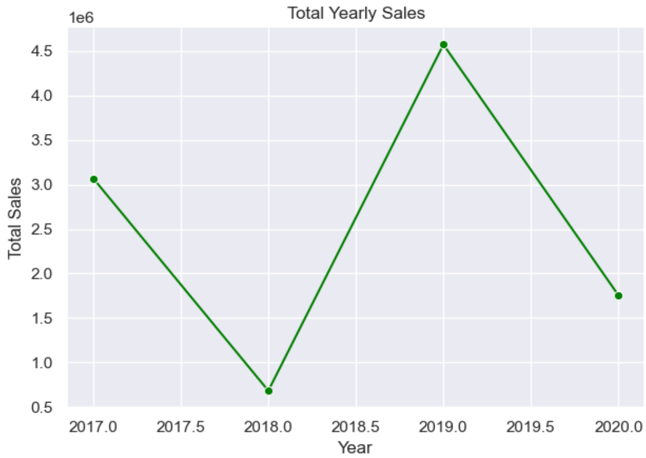


Figure-6: Yearly Sales Trend

Conclusion

The *Warehouse and Retail Sales Analysis* project demonstrates the power of historical data in guiding retail strategy. By cleaning and analysing multi-year sales data, we extracted meaningful insights into supplier performance, product demand, seasonal patterns, and inventory dynamics. These insights support better decision-making regarding product stocking, supplier negotiation, and marketing planning.

Our project validates that data analytics is not just a technical exercise—it's a strategic tool that can significantly improve operational efficiency and business profitability. With further refinement, such as incorporating regional or customer data, this analysis can evolve into a full-fledged decision support system for retail management.

Appendix

Import Libraries

Numpy: Used for numerical computing in Python.

Key Features:

- Works with arrays and matrices.
- Offers fast mathematical operations on large datasets.
- Forms the core of most scientific and machine learning libraries.

Pandas: Provides data structures for efficient data manipulation and analysis.

Key Features:

- Uses DataFrames for handling tabular data.
- Simplifies data cleaning, filtering, grouping, and merging.
- Supports file operations like reading/writing CSV, Excel, etc.

Matplotlib: Used for creating visualizations and plots.

Key Features:

- Produces static, animated, and interactive plots.
- Commonly used for line charts, bar plots, histograms, etc.
- Highly customizable for visual styling.

Seaborn: Built on top of Matplotlib for statistical visualizations.

Key Features:

- Provides attractive and informative plots with minimal code.
- Ideal for visualizing distributions, correlations, and categorical data.
- Integrates well with Pandas DataFrames.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Read Dataset

- .read_csv is a Pandas function used to load CSV data.

```
df = pd.read_csv('/Users/aaryanbabuta/Documents/Final Year
Project/Warehouse_and_Retail_Sales.csv')
```

Data Cleaning

The .head() function in Pandas is used to view the first few rows of a DataFrame.

1. By default, it shows the first 5 rows.
2. It's commonly used to quickly inspect the structure and contents of a dataset.

```
df.head()
```

```
   YEAR  MONTH                           SUPPLIER ITEM CODE  \
0  2020      1  REPUBLIC NATIONAL DISTRIBUTING CO    100009
1  2020      1                          PWSWN INC    100024
2  2020      1             RELIABLE CHURCHILL LLLP      1001
3  2020      1           LANTERNA DISTRIBUTORS INC    100145
4  2020      1                DIONYSOS IMPORTS INC    100293

                     ITEM DESCRIPTION ITEM TYPE  RETAIL SALES  \
0                 BOOTLEG RED - 750ML      WINE          0.00
1           MOMENT DE PLAISIR - 750ML      WINE          0.00
2  S SMITH ORGANIC PEAR CIDER - 18.7OZ      BEER          0.00
3     SCHLINK HAUS KABINETT - 750ML       WINE          0.00
4       SANTORINI GAVALA WHITE - 750ML     WINE          0.82

   RETAIL TRANSFERS  WAREHOUSE SALES
0               0.0              2.0
1               1.0              4.0
2               0.0              1.0
3               0.0              1.0
4               0.0              0.0
```

The .info() function provides a summary of the DataFrame's structure.

1. Displays the number of rows and columns.
2. Shows column names, data types, and non-null counts.
3. Helps identify missing values and understand the overall shape of the data.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307645 entries, 0 to 307644
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   YEAR              307645 non-null  int64
 1   MONTH             307645 non-null  int64
 2   SUPPLIER          307478 non-null  object
 3   ITEM CODE         307645 non-null  object
 4   ITEM DESCRIPTION  307645 non-null  object
 5   ITEM TYPE         307644 non-null  object
```

```
6   RETAIL SALES      307642 non-null  float64
7   RETAIL TRANSFERS  307645 non-null  float64
8   WAREHOUSE SALES   307645 non-null  float64
dtypes: float64(3), int64(2), object(4)
memory usage: 21.1+ MB
```

The .describe() function generates summary statistics of numerical columns in a DataFrame.

1. Includes metrics like count, mean, standard deviation, min, max, and quartiles (25%, 50%, 75%).
1. Helps understand the distribution and spread of numerical data.
2. Can also be used on categorical columns by specifying include='object'.

```
df.describe()
```

```
                YEAR          MONTH   RETAIL SALES  RETAIL TRANSFERS  \
count  307645.000000  307645.000000  307642.000000     307645.000000
mean     2018.438525       6.423862       7.024071          6.936465
std         1.083061       3.461812      30.986238         30.237195
min      2017.000000       1.000000      -6.490000        -38.490000
25%      2017.000000       3.000000       0.000000          0.000000
50%      2019.000000       7.000000       0.320000          0.000000
75%      2019.000000       9.000000       3.267500          3.000000
max      2020.000000      12.000000    2739.000000       1990.830000

       WAREHOUSE SALES
count    307645.000000
mean         25.294597
std         249.916798
min       -7800.000000
25%           0.000000
50%           1.000000
75%           5.000000
max       18317.000000
```

The .isnull().sum() function is used to identify missing values in a DataFrame.

- .isnull() returns a DataFrame of the same shape with True for missing values.
- .sum() then counts the number of missing (null) values in each column.

```
df.isnull().sum() # Count missing values
```

```
YEAR                  0
MONTH                 0
SUPPLIER            167
ITEM CODE             0
ITEM DESCRIPTION      0
ITEM TYPE             1
RETAIL SALES          3
RETAIL TRANSFERS      0
```

```
WAREHOUSE SALES       0
dtype: int64
```

The .dropna() function is used to remove missing values (NaN) from a DataFrame.

- By default, it removes any row with at least one missing value.
- The inplace=True argument modifies the original DataFrame directly, rather than returning a new one.

```
df.dropna(inplace=True) # Drop rows with missing value
```

The .drop_duplicates() function is used to remove duplicate rows from a DataFrame.

- By default, it keeps the first occurrence and removes the rest.
- The inplace=True argument ensures the original DataFrame is updated without creating a copy.

```
df.drop_duplicates(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 307477 entries, 0 to 307644
Data columns (total 9 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   YEAR              307477 non-null  int64
 1   MONTH             307477 non-null  int64
 2   SUPPLIER          307477 non-null  object
 3   ITEM CODE         307477 non-null  object
 4   ITEM DESCRIPTION  307477 non-null  object
 5   ITEM TYPE         307477 non-null  object
 6   RETAIL SALES      307477 non-null  float64
 7   RETAIL TRANSFERS  307477 non-null  float64
 8   WAREHOUSE SALES   307477 non-null  float64
dtypes: float64(3), int64(2), object(4)
memory usage: 23.5+ MB
```

The .columns attribute returns a list-like object containing the names of all columns in a DataFrame.

- It helps you view, access, or rename the columns.
- Useful for checking column names after loading or modifying data.

```
df.columns
```

```
Index(['YEAR', 'MONTH', 'SUPPLIER', 'ITEM CODE', 'ITEM DESCRIPTION',
       'ITEM TYPE', 'RETAIL SALES', 'RETAIL TRANSFERS', 'WAREHOUSE
SALES'],
      dtype='object')
```

Exploratory Style Data Analysis

1. Calculate and visualise each year's total sales (retail and warehouse).

```
total_retail_sales = sum(df['RETAIL SALES'])
total_retail_sales
```

```
2153459.389999994
```

```
total_warehouse_sales = sum(df['WAREHOUSE SALES'])
total_warehouse_sales
```

```
7802401.279999921
```

```
df['TOTAL SALES'] = df['RETAIL SALES'] + df['WAREHOUSE SALES']

df = df[df['TOTAL SALES'] >= 0]
```

.groupby() groups the DataFrame by the [entered] column then .sum() sums the values in the [column_name] column for [entered] column.

1. groupby('YEAR') groups the data by unique values in the YEAR column.
2. ['TOTAL SALES'] selects the column to perform aggregation on.
3. .sum() adds up the sales within each year group.

```
# Group the data by 'YEAR' and calculate total sales for each year
yearly_sales = df.groupby('YEAR')['TOTAL SALES'].sum()
```

The .reset_index() function is used to reset the index of a DataFrame to the default integer index.

- It moves the current index (especially after operations like groupby) back to a regular column.
- This makes the DataFrame easier to work with, especially when exporting or plotting.

```
# Convert the result into a DataFrame
yearly_sales = yearly_sales.reset_index()

yearly_sales.head()
```

```
   YEAR  TOTAL SALES
0  2017   3068825.20
1  2018    683749.29
2  2019   4571531.90
3  2020   1754426.26
```

sns.set()

- Configures Seaborn's default theme and improves the overall look and feel of plots.
- It applies a clean and attractive style to all plots in the session.

sns.lineplot()

- Creates a line plot using Seaborn, ideal for showing trends over time or continuous data.
- It automatically handles axes, legends, and confidence intervals.

plt.title()

- Adds a title to the plot.
- Helps describe what the plot represents for better understanding.

plt.xlabel()

- Sets the label for the x-axis, making the axis easier to interpret.

plt.ylabel()

- Sets the label for the y-axis, clarifying what the y-axis values represent.

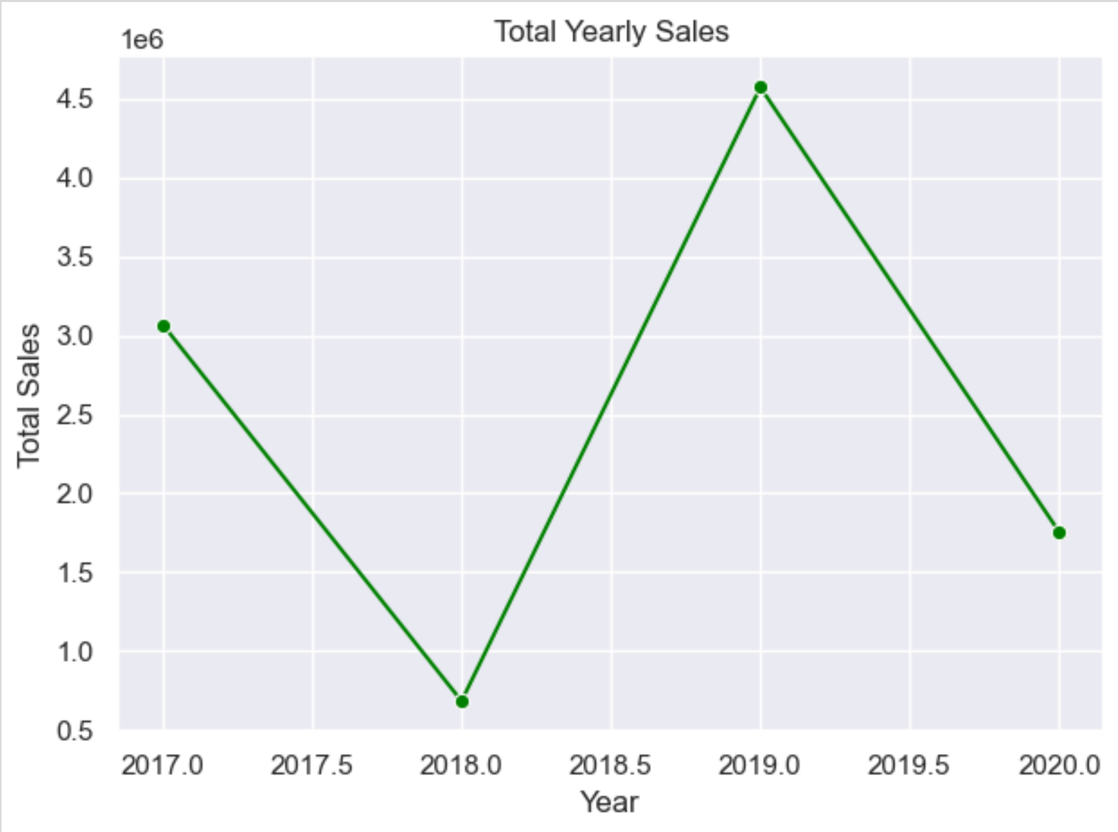
plt.tight_layout()

- Automatically adjusts spacing between plot elements to prevent overlaps.
- Improves readability, especially when axis labels or titles are long.

plt.show()

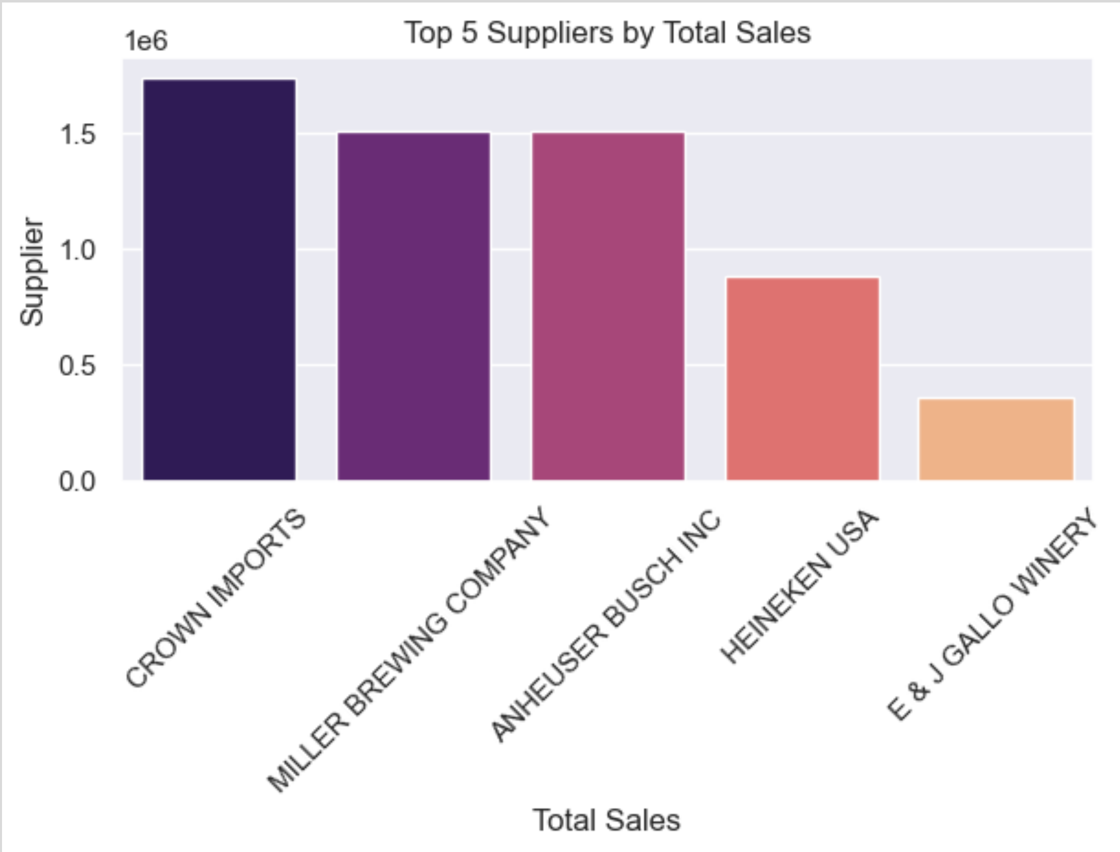
- Displays the final plot.
- Required to render the plot when using Matplotlib in some environments like

```
sns.set(style = 'darkgrid')
sns.lineplot(x = yearly_sales['YEAR'], y = yearly_sales['TOTAL SALES'],
marker = 'o', color = 'green')
plt.title('Total Yearly Sales')
plt.xlabel('Year')
plt.ylabel('Total Sales')
plt.tight_layout()
plt.show()
```



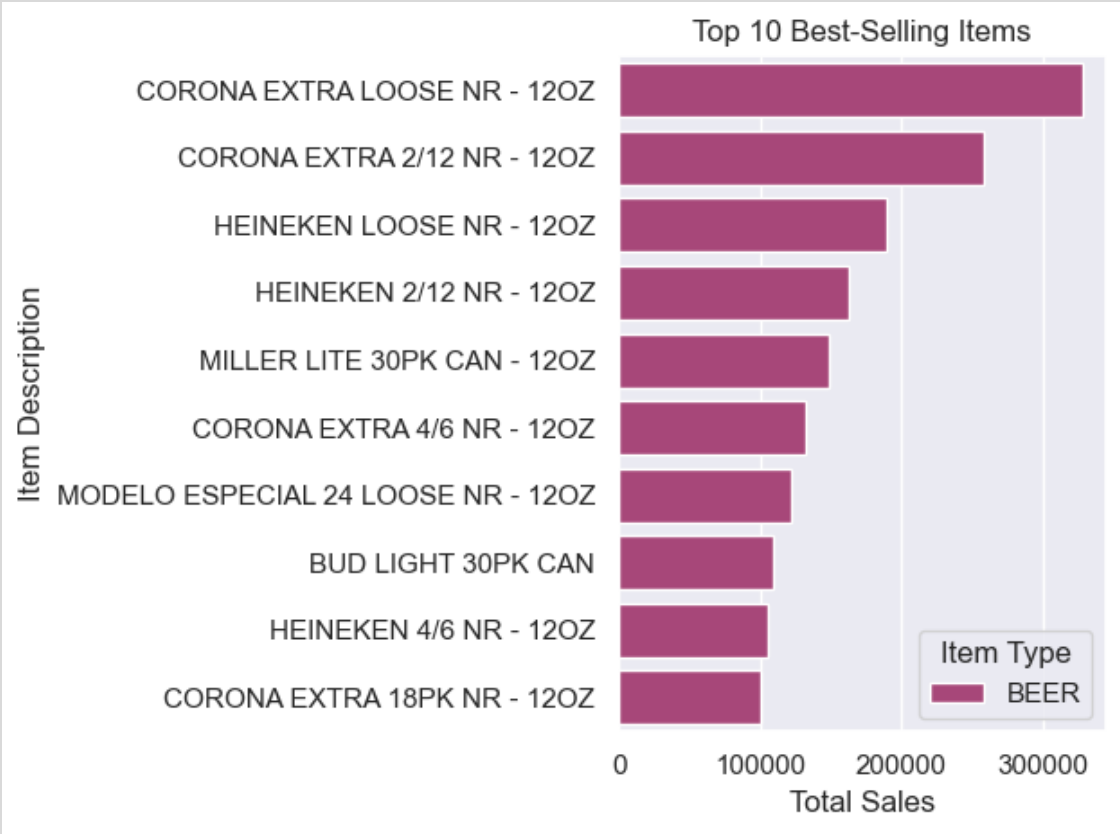
2. Determine the top 5 suppliers based on total sales (both retail and warehouse) for the entire dataset.

```
supplier_sales = df.groupby('SUPPLIER')['TOTAL SALES'].sum()

top5suppliers = supplier_sales.nlargest(5)

top5suppliers = top5suppliers.reset_index()

sns.barplot(x = 'SUPPLIER', y = 'TOTAL SALES', data = top5suppliers,
hue='SUPPLIER', palette='magma')
plt.title('Top 5 Suppliers by Total Sales')
plt.xlabel('Total Sales')
plt.ylabel('Supplier')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



3. Identify the top 10 best-selling items (based on total sales) and provide their descriptions and type

```
item_sales = df.groupby(['ITEM CODE', 'ITEM DESCRIPTION', 'ITEM
TYPE'])['TOTAL SALES'].sum()
top10items = item_sales.nlargest(10).reset_index()

sns.barplot(x='TOTAL SALES', y='ITEM DESCRIPTION', data=top10items,
hue='ITEM TYPE', palette = 'magma')
plt.title('Top 10 Best-Selling Items')
plt.xlabel('Total Sales')
plt.ylabel('Item Description')
plt.legend(title='Item Type')
plt.tight_layout()
plt.show()
```



Business Analysis

1. Calculate the monthly average retail sales and warehouse sales separately for each year.

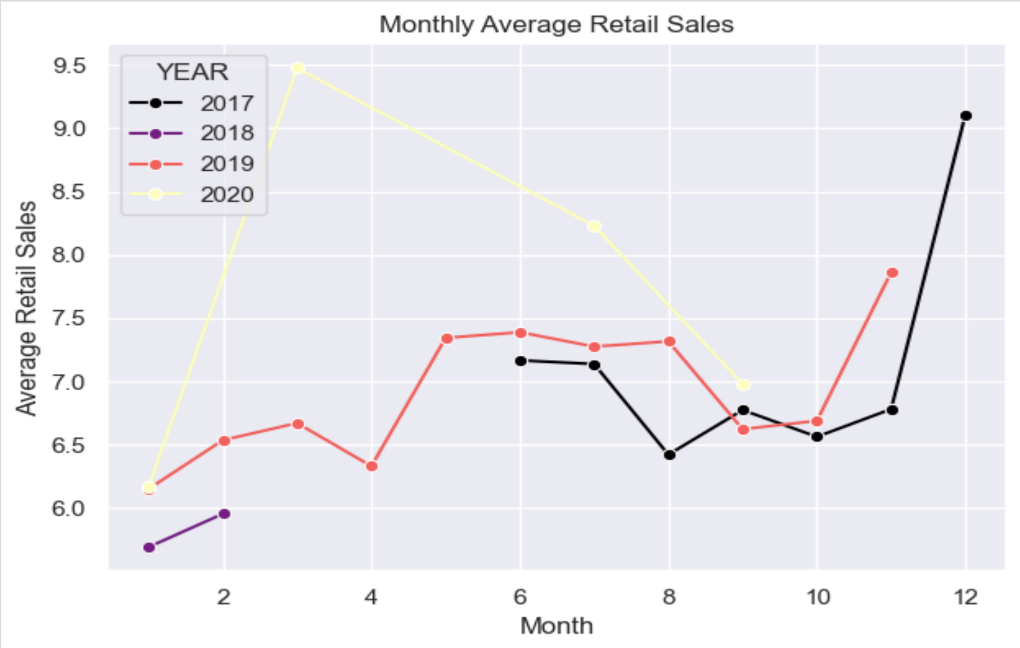
2. Analyse whether there are any seasonal trends in sales data. Provide visualisations to support your analysis

```
monthly_retail_sales = df.groupby(['YEAR', 'MONTH'])['RETAIL SALES'].mean().reset_index()

monthly_retail_sales.rename(columns = {'RETAIL SALES' : 'MONTHLY AVERAGE RETAIL SALES'}, inplace = True)

monthly_retail_sales
```

	YEAR	MONTH	MONTHLY AVERAGE RETAIL SALES
0	2017	6	7.167287
1	2017	7	7.138307
2	2017	8	6.422831
3	2017	9	6.774891
4	2017	10	6.563757
5	2017	11	6.782159
6	2017	12	9.108988

```
7    2018       1                        5.692464
8    2018       2                        5.955024
9    2019       1                        6.151864
10   2019       2                        6.536784
11   2019       3                        6.671756
12   2019       4                        6.331958
13   2019       5                        7.344643
14   2019       6                        7.389358
15   2019       7                        7.276014
16   2019       8                        7.317150
17   2019       9                        6.623562
18   2019      10                        6.690026
19   2019      11                        7.862720
20   2020       1                        6.168385
21   2020       3                        9.475533
22   2020       7                        8.230957
23   2020       9                        6.980214
```

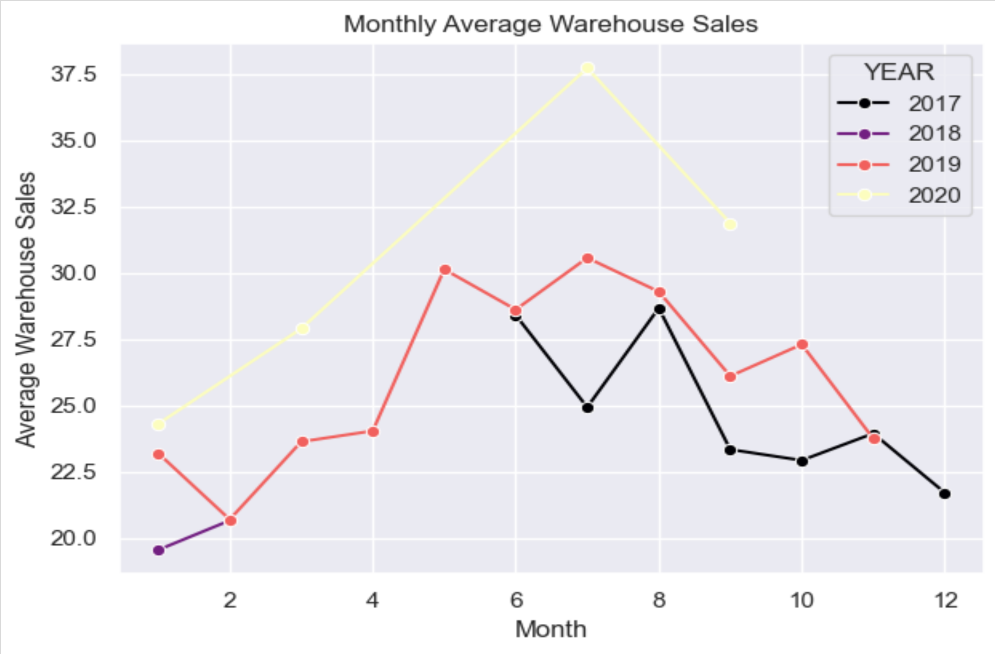
```
sns.set(style = 'darkgrid')
sns.lineplot(x = 'MONTH', y = 'MONTHLY AVERAGE RETAIL SALES',
hue='YEAR', data = monthly_retail_sales, marker = 'o', palette =
'magma')
plt.title('Monthly Average Retail Sales')
plt.xlabel('Month')
plt.ylabel('Average Retail Sales')
plt.legend(title = 'YEAR')
plt.tight_layout()
plt.show()
```



```
monthly_warehouse_sales = df.groupby(['YEAR', 'MONTH'])['WAREHOUSE SALES'].mean().reset_index()

monthly_warehouse_sales.rename(columns = {'WAREHOUSE SALES' : 'MONTHLY AVERAGE WAREHOUSE SALES'}, inplace = True)

monthly_warehouse_sales
```

```
YEAR  MONTH  MONTHLY AVERAGE WAREHOUSE SALES
0   2017      6                        28.404110
1   2017      7                        24.928805
2   2017      8                        28.669707
3   2017      9                        23.335576
4   2017     10                        22.926198
5   2017     11                        23.940930
6   2017     12                        21.708236
7   2018      1                        19.547531
8   2018      2                        20.658370
9   2019      1                        23.195540
10  2019      2                        20.696864
11  2019      3                        23.629749
12  2019      4                        24.036216
13  2019      5                        30.130190
14  2019      6                        28.610691
15  2019      7                        30.563290
16  2019      8                        29.290812
17  2019      9                        26.102344
18  2019     10                        27.298679
19  2019     11                        23.747031
20  2020      1                        24.300933
21  2020      3                        27.894574
22  2020      7                        37.722165
23  2020      9                        31.867968
```

```
sns.set(style = 'darkgrid')
sns.lineplot(x = 'MONTH', y = 'MONTHLY AVERAGE WAREHOUSE SALES', hue='YEAR', data = monthly_warehouse_sales, marker = 'o', palette = 'magma')
plt.title('Monthly Average Warehouse Sales')
plt.xlabel('Month')
plt.ylabel('Average Warehouse Sales')
plt.legend(title = 'YEAR')
plt.tight_layout()
plt.show()
```



3. Calculate the total sales for each item type and identify which item type contributes the most to overall sales.

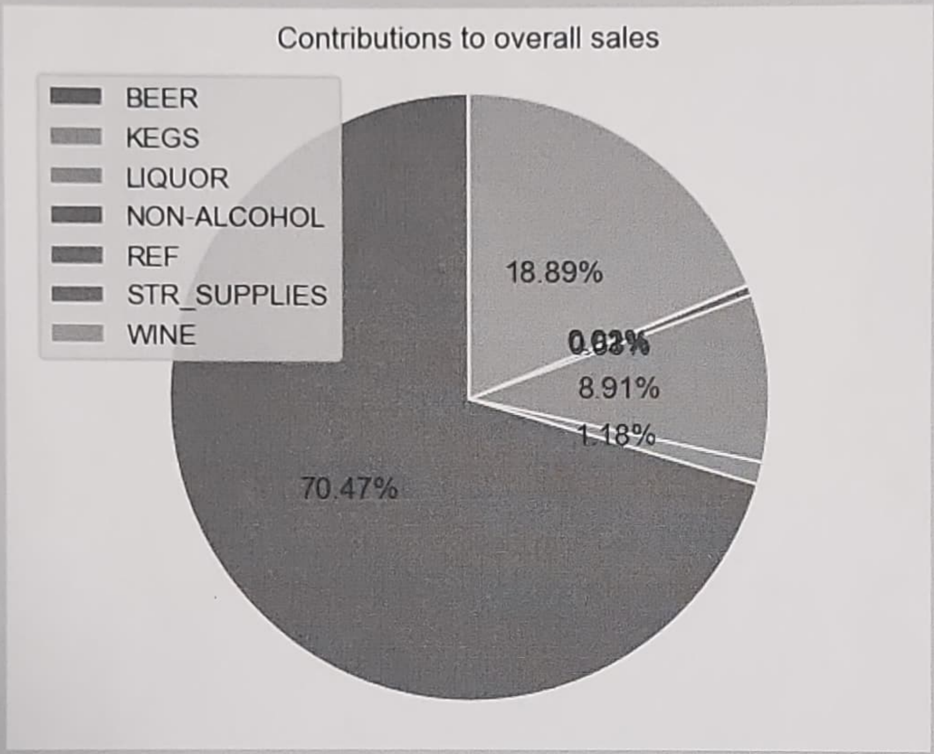
```
type_sale = df.groupby(['ITEM TYPE'])['TOTAL SALES'].sum()
item_type_sale = type_sale.reset_index()

item_type_sale
```

```
      ITEM TYPE  TOTAL SALES
0          BEER   7102183.38
1          KEGS    118623.00
2        LIQUOR    897642.90
3   NON-ALCOHOL     53300.32
4           REF       663.71
5  STR_SUPPLIES      2234.90
6          WINE   1903884.44
```

```
max_sales_index = item_type_sale['TOTAL SALES'].idxmax()
max_sales_item = item_type_sale.loc[max_sales_index]
print(max_sales_item)
```

```
ITEM TYPE            BEER
TOTAL SALES    7102183.38
Name: 0, dtype: object
```

```python
plt.pie(item_type_sale['TOTAL SALES'], labels = None,
autopct='%1.2f%%', startangle=90, pctdistance=0.5)
plt.title('Contributions to overall sales')
plt.axis('equal')
plt.legend(item_type_sale['ITEM TYPE'])
plt.show()
```



Ş-PSJ 15 May 25

Guide: Dr. Parag Jawarkar
Dean Training and Placement Cell
Shri Ramdeobaba College of Engineering and Management,
Nagpur, India

Co-Guide: Pratham Gangwal
Data Scientist
Infocepts
Nagpur, India

Name & Signature RRC Members

S. G. Daware

Laddha
S. V. Laddha