10/17/2024

Super-Store Sales Analysis

Data Analyst Track (group B)

Prepared By:

1. Ramy Safwat
2. Magid Atef
3. Moustafa Gaber
4. Mohamed Abdel Aziz
5. Aya Mamdouh
6. Ahmed Rabii
7. Abdel Halim Mahmoud

▪Index:

1. Phase 1:Build Data Model, Data Cleaning, and Preprocessing

- **Problem definition**
- **Objective Setting**
- **Scope of Analysis**
- **Deliverables**
- **Benefits**
- **Data Source**
- **Identify Entities and Attributes**
- **Schema Designs**
- **Data Model & Relationship Establishment**
- **Cleaning and Preprocessing Data**

A. Python-Based Cleaning and Preprocessing (pandas)

1. Load Data.
2. Explore and Understand Data.
3. Handling Missing Values.
4. Handling Duplicated Value
5. Describing Data
6. Feature Engineering
7. Encoding Categorical Variables
8. Splitting Ship Date and Order date
9. Save the updated Data Frame to a CSV file.

B. SQL-Based Cleaning and Preprocessing

1. Handling Duplicated Product_ID Values in Product Table
2. Handling Duplicated Post_Code Values in Regions table
3. Joining Tables
4. Adding Price columns
5. Adding Quarter columns

2. Phase 2 Analysis Questions Phase:

A. Setting Questions to Ask

B. Analysis Stage

1. Sales and Profit Analysis
2. Region/State/City analysis
3. Customer Analysis
4. Category/Subcategory/Product Analysis
5. Correlations of Sales and Profit to specific variables

3. Phase 3 Forecasting Questions Phase

1. Sales and Profit Forecasting in the next three years
2. Sales and Profit Forecasting per Regions
3. Sales and Profit Forecasting per State
4. Sales Forecasting per Category and Sub_Category

4. Phase 4 Data Visualizations

Phase 1:Build Data Model, Data Cleaning, and Preprocessing

- **Problem definition:**

  In the fast-paced e-commerce landscape, grasping customer behavior and sales trends is crucial for success. Facing increasing demands and fierce competition, a major Superstore seeks our expertise to determine what strategies work best for them.

- **Objective:**

  - To analyze sales data to identify Key customer segments, Regions, Categories, Products and uncover insights that can optimize sales and profitability.

- **Scope:**

  - **Sales Patterns:** Analyze sales trends over time to identify seasonal variations, peak periods, and sales cycles.
  - **Regions:** Compare sales performance across different regions to identify high-performing markets and areas for expansion.
  - **Categories:** Evaluate the popularity of different product categories to assess customer preferences and identify potential growth opportunities.
  - **Product Performance:** Analyze individual product sales to identify best-sellers, underperforming items, and potential product line adjustments.
  - **Customer Preferences:** Understand customer preferences based on demographic data, purchase history, and other relevant factors.
  - **Customer Segments:** Identify target customer segments and their characteristics to tailor marketing efforts and optimize product offerings.

- **Deliverables:**

  - Detailed analysis of sales data, including visualizations and key findings.
  - Identification of customer segments and their preferences as well as other key parameters
  - Actionable recommendations to refine sales strategies and drive business growth.
  - Development of a regression model to predict sales or profit.

- **Benefits:**

  - Improved understanding of customer behavior and preferences.
  - Optimized product offerings and marketing campaigns.
  - Increased sales and profitability.
  - Enhanced decision-making capabilities

Data Sources:

Dataset: https://www.kaggle.com/datasets/vivek468/superstore-dataset-final

Identify Entities and Attributes:

Entities:

- **Customers :** we have 793 customers
- **Segments :** There are three segments Consumer ,Corporate , Home office
- **Cities :** there are 531 Cities
- **States:** 49 States
- **Regions:** There are 4 regions Central, East, West, and South
- **Categories:** There are three categories Furniture , Office Supplies, and Technology
- **Sub-Categories:** There are 17 sub-categories.
- **Products:** there are 1850 products

Dataset Description

1. **Row ID:** Unique identifier for each row.
2. **Order ID:** Unique Order ID for each Customer.
3. **Order Date:** Order Date of the product.
4. **Ship Date:** Shipping Date of the Product.
5. **Ship Mode:** Shipping Mode specified by the Customer.
6. **Customer ID:** Unique ID to identify each Customer.
7. **Customer Name:** Name of the Customer.
8. **Segment:** The segment where the Customer belongs.
9. **Country:** Country of residence of the Customer.
10. **City:** City of residence of the Customer.
11. **State:** State of residence of the Customer.
12. **Postal Code:** Postal Code of every Customer.
13. **Region:** Region where the Customer belongs.
14. **Product ID:** Unique ID of the Product.
15. **Category:** Category of the product ordered.
16. **Sub-Category:** Sub-Category of the product ordered.
17. **Product Name:** Name of the Product
18. **Sales:** Sales of the Product.
19. **Quantity:** Quantity of the Product.
20. **Discount:** Discount provided.
21. **Profit:** Profit/Loss incurred.

Schema Designs

Data will be divided into 4 tables according to pillars of Analysis into

a) Sales table and its attributes are.

1. **Row ID:** Unique identifier for each row.
2. **Order ID:** Unique Order ID for each Customer.
3. **Order Date:** Order Date of the product.
4. **Ship Date:** Shipping Date of the Product.
5. **Ship Mode:** Shipping Mode specified by the Customer.
6. **Customer ID:** Unique ID to identify each Customer.
7. **Postal Code:** Postal Code of every Customer.
8. **Product ID:** Unique ID of the Product.
9. **Sales:** Sales of the Product.
10. **Quantity:** Quantity of the Product.
11. **Discount:** Discount provided.
12. **Profit:** Profit/Loss incurred.

b) Customer table and its attributes are.

1. **Customer ID:** Unique ID to identify each Customer.
2. **Customer Name:** Name of the Customer.
3. **Segment:** The segment where the Customer belongs.

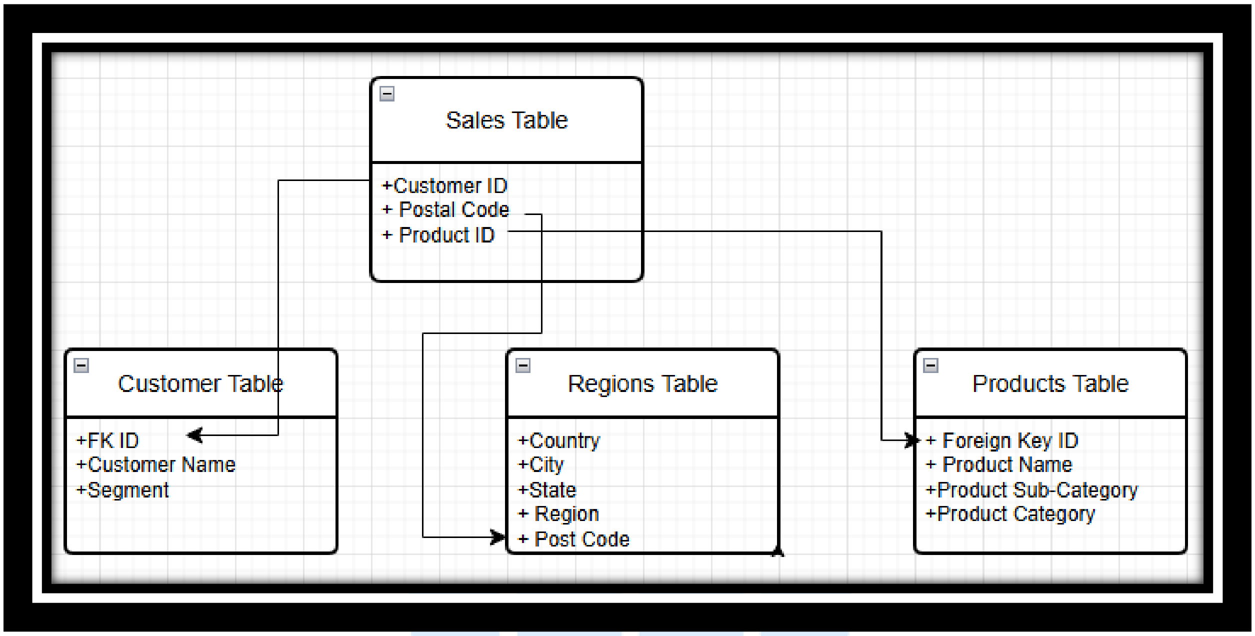
c) Regions table and its attributes are.

1. **Country:** Country of residence of the Customer.
2. **City:** City of residence of the Customer.
3. **State:** State of residence of the Customer.
4. **Post_Code:** Postal Code of every Customer.
5. **Region:** Region where the Customer belongs.

d) Products table and its attributes are.

1. **Product_ID:** Unique ID of the Product.
2. **Category:** Category of the product ordered.
3. **Sub-Category:** Sub-Category of the product ordered.
4. **Product Name:** Name of the Product

Data Model & Relationship Establishment



As we can Notice from the table the parent table is Sales table and is in relationship with the following tables as the following:

1. Customer Table : and the primary key in the sales table is Customer ID and is related to the foreign key(Customer ID) in the customer table.
2. Regions table and the primary key in the sales table is Postal Code and is related to the Post Code which acts as a foreign key in the Regions table.
3. Products Table: : and the primary key in the sales table is Product ID and is related to the Foreign Key ID(Product_ID) which acts as a Foreign Id in the Products table.

Cleaning and Preprocessing Data

A. Python-Based Cleaning and Preprocessing (pandas)

1. Load Data

```
from AutoClean import AutoClean
import pandas as pd.
resultant = pd.read_csv("sales.csv")
pipeline = AutoClean(resultant)
x=pipeline.output
print(x)
```

Python Report.

02-10-2024 13:19:52.75 - INFO - Started validation of input parameters...
02-10-2024 13:19:52.75 - INFO - Completed validation of input parameters.
02-10-2024 13:19:52.75 - INFO - Started handling of duplicates... Method: "AUTO."
02-10-2024 13:19:52.78 - DEBUG - 0 missing values found.
02-10-2024 13:19:52.78 - INFO - Completed handling of duplicates in 0.029127 seconds.
02-10-2024 13:19:52.78 - INFO - Started handling of missing values...
02-10-2024 13:19:52.78 - DEBUG - 0 missing values found.
02-10-2024 13:19:52.78 - INFO - Completed handling of missing values in 0.005096 seconds.
02-10-2024 13:19:52.78 - INFO - Started handling of outliers... Method: "WINZ."
02-10-2024 13:19:54.52 - DEBUG - Outlier imputation of 1167 value(s) succeeded for feature "Sales."
02-10-2024 13:19:54.75 - DEBUG - Outlier imputation of 170 value(s) succeeded for feature "Quantity."
02-10-2024 13:19:55.85 - DEBUG - Outlier imputation of 856 value(s) succeeded for feature "Discount."
02-10-2024 13:19:58.33 - DEBUG - Outlier imputation of 1881 value(s) succeeded for feature "Profit."
02-10-2024 13:19:58.33 - INFO - Completed handling of outliers in 5.542315 seconds.
02-10-2024 13:19:58.33 - INFO - Started conversion of DATETIME features... Granularity: s.
02-10-2024 13:19:58.45 - DEBUG - Conversion to DATETIME succeeded for feature "Ship Date."

02-10-2024 13:19:58.56 - DEBUG - Conversion to DATETIME succeeded for feature "Order Date."
02-10-2024 13:19:58.56 - INFO - Completed conversion of DATETIME features in 0.2363 seconds.
02-10-2024 13:19:58.57 - INFO - Started encoding categorical features... Method: "AUTO."
02-10-2024 13:19:58.57 - DEBUG - Encoding to ONEHOT succeeded for feature "Ship Mode."
02-10-2024 13:19:58.59 - DEBUG - Encoding skipped for feature "Customer ID."
02-10-2024 13:19:58.59 - DEBUG - Skipped encoding for DATETIME feature "Ship Date"
02-10-2024 13:19:58.60 - DEBUG - Encoding skipped for feature "Order ID."
02-10-2024 13:19:58.60 - DEBUG - Skipped encoding for DATETIME feature "Order Date"
02-10-2024 13:19:58.62 - DEBUG - Encoding skipped for feature "Product ID."
02-10-2024 13:19:58.62 - INFO - Completed encoding of categorical features in 0.04828 seconds.
02-10-2024 13:19:58.62 - INFO - Started feature type conversion...
02-10-2024 13:19:58.62 - DEBUG - Conversion to type INT succeeded for feature "Row ID."
02-10-2024 13:19:58.62 - DEBUG - Conversion to type INT succeeded for feature "Postal Code."
02-10-2024 13:19:58.64 - DEBUG - Conversion to type FLOAT succeeded for feature "Sales."
02-10-2024 13:19:58.64 - DEBUG - Conversion to type INT succeeded for feature "Quantity."
02-10-2024 13:19:58.64 - DEBUG - Conversion to type FLOAT succeeded for feature "Discount."
02-10-2024 13:19:58.66 - DEBUG - Conversion to type FLOAT succeeded for feature "Profit."
02-10-2024 13:19:58.66 - DEBUG - Conversion to type INT succeeded for feature "Day."
02-10-2024 13:19:58.66 - DEBUG - Conversion to type INT succeeded for feature "Month."
02-10-2024 13:19:58.66 - DEBUG - Conversion to type INT succeeded for feature "Year."
02-10-2024 13:19:58.66 - INFO - Completed feature type conversion for 9 feature(s) in 0.043861 seconds.
02-10-2024 13:19:58.66 - INFO - AutoClean process completed in 5.917362 seconds.

2. Explore and Understand Data

Jupyter Platform

- from AutoClean import AutoClean
- import pandas as pd.
- df=pd.read_csv("sales.csv")
- pipline=AutoClean(df)

[AutoClean process completed in 6.269056 seconds Logfile saved to: C:\Users\ramys\autoclean.log]

- pipline.output

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Postal Code	Product ID	Sales	Quantity	Discount	Profit	Day	Month	Year	Ship Mode_First Class	Ship Mode_Same Day	Mode_S
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	42420	FUR-BO-10001798	261.960	2	0.00	41.9136	11	11	2016	False	False	
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	42420	FUR-CH-10000454	498.930	3	0.00	70.8169	11	11	2016	False	False	
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	90036	OFF-LA-10000240	14.620	2	0.00	6.8714	16	6	2016	False	False	
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	33311	FUR-TA-10000577	498.930	5	0.45	-39.7241	18	10	2015	False	False	
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	33311	OFF-ST-10000760	22.368	2	0.20	2.5164	18	10	2015	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9989	9990	CA-2014-110422	2014-01-21	2014-01-23	Second Class	TB-21400	33180	FUR-FU-10001889	25.248	3	0.20	4.1028	23	1	2014	False	False	
9990	9991	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	92627	FUR-FU-10000747	91.960	2	0.00	15.6332	3	3	2017	False	False	
9991	9992	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	92627	TEC-PH-10003645	258.576	2	0.20	19.3932	3	3	2017	False	False	
9992	9993	CA-2017-121258	2017-02-26	2017-03-03	Standard Class	DB-13060	92627	OFF-PA-10004041	29.600	4	0.00	13.3200	3	3	2017	False	False	
9993	9994	CA-2017-119914	2017-05-04	2017-05-09	Second Class	CC-12220	92683	OFF-AP-10002684	243.160	2	0.00	70.8169	9	5	2017	False	False	

9994 rows × 19 columns

- from IPython.display import display
- display(df.head())
- display(df.tail())

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Postal Code	Product ID	Sales	Quantity	Discount	Profit
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	42420	FUR-BO-10001798	261.9600	2	0.00	41.9136
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	42420	FUR-CH-10000454	731.9400	3	0.00	219.5820
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	90036	OFF-LA-10000240	14.6200	2	0.00	6.8714
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	33311	FUR-TA-10000577	957.5775	5	0.45	-383.0310
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	33311	OFF-ST-10000760	22.3680	2	0.20	2.5164

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Postal Code	Product ID	Sales	Quantity	Discount	Profit
9989	9990	CA-2014-110422	1/21/2014	1/23/2014	Second Class	TB-21400	33180	FUR-FU-10001889	25.248	3	0.2	4.1028
9990	9991	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	92627	FUR-FU-10000747	91.960	2	0.0	15.6332
9991	9992	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	92627	TEC-PH-10003645	258.576	2	0.2	19.3932
9992	9993	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	92627	OFF-PA-10004041	29.600	4	0.0	13.3200
9993	9994	CA-2017-119914	5/4/2017	5/9/2017	Second Class	CC-12220	92683	OFF-AP-10002684	243.160	2	0.0	72.9480

- display(df.info())

```
<Class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Row ID        9994 non-null   int64
 1   Order ID      9994 non-null   object
 2   Order Date    9994 non-null   object
 3   Ship Date     9994 non-null   object
 4   Ship Mode     9994 non-null   object
 5   Customer ID   9994 non-null   object
 6   Postal Code   9994 non-null   int64
 7   Product ID    9994 non-null   object
 8   Sales         9994 non-null   float64
 9   Quantity      9994 non-null   int64
 10  Discount      9994 non-null   float64
 11  Profit        9994 non-null   float64
dtypes: float64(3), int64 (3), object(6)
memory usage: 937.1+ KB
None
```

3. Handling Missing Values

```
missing_values = pd.DataFrame({'Feature': df.columns,
                               'No. of Missing Values': df.isnull().sum().values,
                               '% of Missing Values': ((df.isnull().sum().values)/len(df)*100)})
unique_values = pd.DataFrame({'Feature': df.columns,
                               'No. of Unique Values': df.nunique().values})
feature_types = pd.DataFrame({'Feature': df.columns,
                               'DataType': df.dtypes})
merged_df = pd.merge(missing_values, unique_values, on='Feature', how='left')
merged_df = pd.merge(merged_df, feature_types, on='Feature', how='left')

merged_df.
```

	Feature	No. of Missing Values	% of Missing Values	No. of Unique Values	DataType
0	Row ID	0	0.0	9994	int64
1	Order ID	0	0.0	5009	object
2	Order Date	0	0.0	1237	object
3	Ship Date	0	0.0	1334	object
4	Ship Mode	0	0.0	4	object
5	Customer ID	0	0.0	793	object
6	Postal Code	0	0.0	631	int64
7	Product ID	0	0.0	1862	object
8	Sales	0	0.0	5825	float64
9	Quantity	0	0.0	14	int64

4. Handling Duplicated Values Python-Based Cleaning and preprocessing (pandas)

Count duplicate rows.

```
duplicates = df.duplicated().sum()
   # Print the results
print(f"Number of duplicate rows : {duplicates}")
```

Number of duplicate rows: 0

Count duplicate values in columns.

```
duplicated_counts = df.apply(lambda x: x.duplicated().sum())
   # Create a DataFrame to display the results
duplicated_values = pd.DataFrame({
      'Feature': df.columns,
      'No. of Duplicated Values': duplicated_counts.values
   })
duplicated_values.
```

	Feature	No. of Duplicated Values
0	Row ID	0
1	Order ID	4985
2	Order Date	8757
3	Ship Date	8660
4	Ship Mode	9990
5	Customer ID	9201
6	Postal Code	9363
7	Product ID	8132
8	Sales	4169
9	Quantity	9980
10	Discount	9982
11	Profit	2707

5. Describing Data

df[['Sales', 'Quantity', 'Discount', 'Profit']].describe()

	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000
mean	229.858001	3.789574	0.156203	28.656896
std	623.245101	2.225110	0.206452	234.260108
min	0.444000	1.000000	0.000000	-6599.978000
25%	17.280000	2.000000	0.000000	1.728750
50%	54.490000	3.000000	0.200000	8.666500
75%	209.940000	5.000000	0.200000	29.364000
max	22638.480000	14.000000	0.800000	8399.976000

6. Feature Engineering: Create new features from existing data to improve model performance.

- <u>Creating Time frame column</u>

Convert columns to datetime

df['Order Date'] = pd.to_datetime(df['Order Date'])

df['Ship Date'] = pd.to_datetime(df['Ship Date'])

Calculate the time frame (difference in days)

df['Time Frame'] = (df['Ship Date'] - df['Order Date']).dt.days

df

7. Encoding Categorical Variables:

- <u>**Label Encoding**: Convert categorical values to numerical values.</u>

 # Rank the 'Ship Mode' column

df['Ship Mode Rank'] = df['Ship Mode'].rank(method='dense')

Display the DataFrame with the new rank column

Df

8. Splitting Ship Date and Order date

```
# Convert the date column to datetime format
df['date'] = pd.to_datetime(df['date'])

# Extract day, month, and year into separate columns
df['day'] = df['date'].dt.day
df['month'] = df['date'].dt.month
df['year'] = df['date'].dt.year
# Display the DataFrame
print(df)
```

9. Save the updated DataFrame to a CSV file.

```
df.to_csv('updated_sales.csv', index=False)
# Display a message to confirm the file has been saved
print("CSV file has been saved as 'updated_sales.csv'")
```

B. SQL –Based Cleaning

A. Handling Duplicated Product_ID Values in Products table

On selecting Product_ID as a primary we got the following message

MySQL said:

#1062 - Duplicate entry 'FUR-CH-10001146' for key 'PRIMARY'

SELECT Product_Name, Product_ID, COUNT(*) FROM products GROUP BY Product_ID HAVING COUNT(*) > 1 ORDER BY `products`.`Product_ID` ASC;

Name of Product	Product ID	Frequency
DMI Eclipse Executive Suite Bookcases	FUR-BO-10002213	2
Global Value Mid-Back Manager's Chair, Gray	FUR-CH-10001146	2
DAX Wood Document Frame	FUR-FU-10001473	2
Tenex Contemporary Contur Chairmats for Low and Me...	FUR-FU-10004017	2
Howard Miller 13" Diameter Goldtone Round Wall Clo...	FUR-FU-10004091	2
Eldon Image Series Desk Accessories, Burgundy	FUR-FU-10004270	2
Howard Miller 13-3/4" Diameter Brushed Chrome Roun...	FUR-FU-10004848	2
Howard Miller 14-1/2" Diameter Chrome Round Wall C...	FUR-FU-10004864	2
Belkin 7 Outlet SurgeMaster II	OFF-AP-10000576	2
Sanford Colorific Colored Pencils, 12/Box	OFF-AR-10001149	2
Avery Arch Ring Binders	OFF-BI-10002026	2
Ibico Hi-Tech Manual Binding System	OFF-BI-10004632	2
Avery Binding System Hidden Tab Executive Style In...	OFF-BI-10004654	2
White Dual Perf Computer Printout Paper, 2700 Shee...	OFF-PA-10000357	2
Xerox 1952	OFF-PA-10000477	2
Adams Phone Message Book, Professional, 400 Messag...	OFF-PA-10000659	2
Xerox 2	OFF-PA-10001166	2
Xerox 1881	OFF-PA-10001970	2

RSVP Cards & Envelopes, Blank White, 8-1/2" X 11",...	OFF-PA-10002195	2
Xerox 1916	OFF-PA-10002377	2
Xerox 1992	OFF-PA-10003022	2
Fellowes Personal Hanging Folder Files, Navy	OFF-ST-10001228	2
Acco Perma 3000 Stacking Storage Drawers	OFF-ST-10004950	2
Logitech G19 Programmable Gaming Keyboard	TEC-AC-10002049	2
Maxell 4.7GB DVD-RW 3/Pack	TEC-AC-10002550	2
Logitech?P710e Mobile Speakerphone	TEC-AC-10003832	2
Swingline SM12-08 MicroCut Jam Free Shredder	TEC-MA-10001148	2
Cisco Unified IP Phone 7945G VoIP phone	TEC-PH-10001530	2
ClearOne CHATAttach 160 -?speaker phone	TEC-PH-10001795	2
Samsung Galaxy Note 2	TEC-PH-10002200	2
Panasonic KX T7731-B Digital phone	TEC-PH-10002310	2
OtterBox Commuter Series Case - iPhone 5 & 5s	TEC-PH-10004531	2

B. Handling Duplicated Post_Code Values in Regions table

```
SELECT City, Post_Code, COUNT(*)
FROM regions
GROUP BY Post_Code
HAVING COUNT(*) > 1;
```

City	Post_Code	COUNT(*)
Encinitas	92024	2

C. Joining Tables

```
CREATE TABLE final_table AS
SELECT *
FROM sales_updated_sales_1 AS s
INNER JOIN products AS p ON s.Product_ID = p.ID
INNER JOIN regions AS r ON s.Postal_Code = r.Post_Code
INNER JOIN customers AS c ON s.Customer_ID = c.Cs_ID;
```

D. Adding Price columns

```
ALTER TABLE final_table.
ADD COLUMN item_price DECIMAL(10, 2);

UPDATE sales_data.
SET item_price = sales / (Quantity * (1 - Discount))
WHERE Quantity > 0 AND (1 - Discount) > 0;
```

E. Adding Quarter columns

```
ALTER TABLE final_table.
ADD COLUMN Order_date DATE;


UPDATE final_table.
SET Order_date = DATE(CONCAT(Order_Date_Year, '-', Order_Date_Month, '-', Order_Date_day));


ALTER TABLE final_table.
ADD COLUMN Shipping_date DATE;
UPDATE final_table.
SET Shipping_date = DATE(CONCAT(Ship_Date_Year_Date_Year, '-', Ship_Date_Month, '-', Ship_Date_day));


ALTER TABLE final_table.
ADD COLUMN Quarter VARCHAR(2);
UPDATE final_table.
SET Quarter = CASE
   WHEN MONTH(Order_date) IN (1, 2, 3) THEN 'Q1'
   WHEN MONTH(Order_date) IN (4, 5, 6) THEN 'Q2'
   WHEN MONTH(Order_date) IN (7, 8, 9) THEN 'Q3'
   WHEN MONTH(Order_date) IN (10, 11, 12) THEN 'Q4'
   ELSE 'Unknown'
END;

ALTER TABLE final_table.
ADD COLUMN Quarter_shippinjg VARCHAR(2);
UPDATE final_table.
SET Quarter = CASE
   WHEN MONTH(Shipping_date) IN (1, 2, 3) THEN 'Q1'
   WHEN MONTH(Shipping_date) IN (4, 5, 6) THEN 'Q2'
   WHEN MONTH(Shipping_date) IN (7, 8, 9) THEN 'Q3'
   WHEN MONTH(Shipping_date) IN (10, 11, 12) THEN 'Q4'
   ELSE 'Unknown'
```

END;

- Then we drop extra tables to minimize no. of columns

Phase 2 Analysis Questions Phase

A. Questions to Ask

1. Sales & Profit

- Sales and profit trend across years to assess the growth trend.
- Sales across Quarters to assess seasonality.
- Regions / State/Cities contributions to Profit
- Customer and its segmentation type contributions to profit
- Category and sub-category and products contributions to profit

2. Correlations between Sales &

- Price
- Shipping mode
- Time frame
- Quantity
- Discount

B. Analysis stage

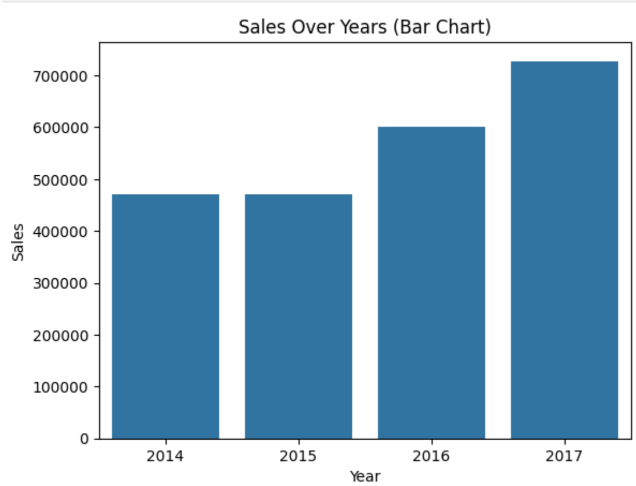
1. Sales and Profit Analysis

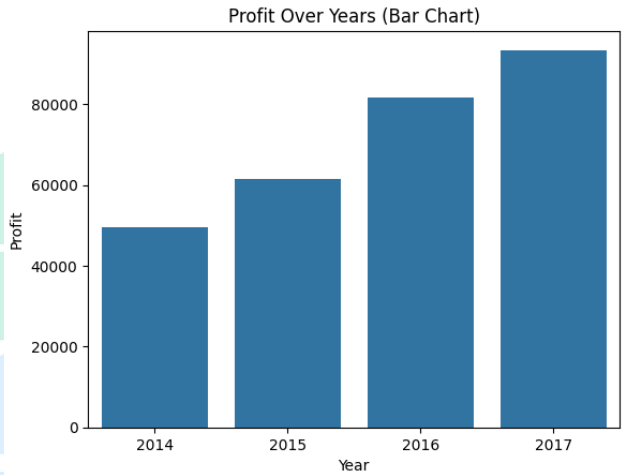
Sales and profit trend across years to assess the growth trend.

```
mydata2=df.groupby(['Order_date _year'], as_index=False)[['Sales','Profit','Discount','Quantity']].sum ()
mydata2
```

	Order_date _year	Sales	Profit	Discount	Quantity
0	2014	471609.0180	49543.9741	315.46	7581
1	2015	470532.5090	61618.6037	327.09	7979
2	2016	601705.6479	81795.1743	400.32	9837
3	2017	727515.3569	93439.2696	518.22	12476

```
sns.barplot(x='Order_date _year', y='Sales', data=mydata2)
plt.title('Sales Over Years (Bar Chart)')
plt.xlabel('Year')
plt.ylabel('Sales')
plt.show()
sns.barplot(x='Order_date _year', y='Profit', data=mydata2)
plt.title('Profit Over Years (Bar Chart)')
plt.xlabel('Year')
plt.ylabel('Profit')
plt.show()
```



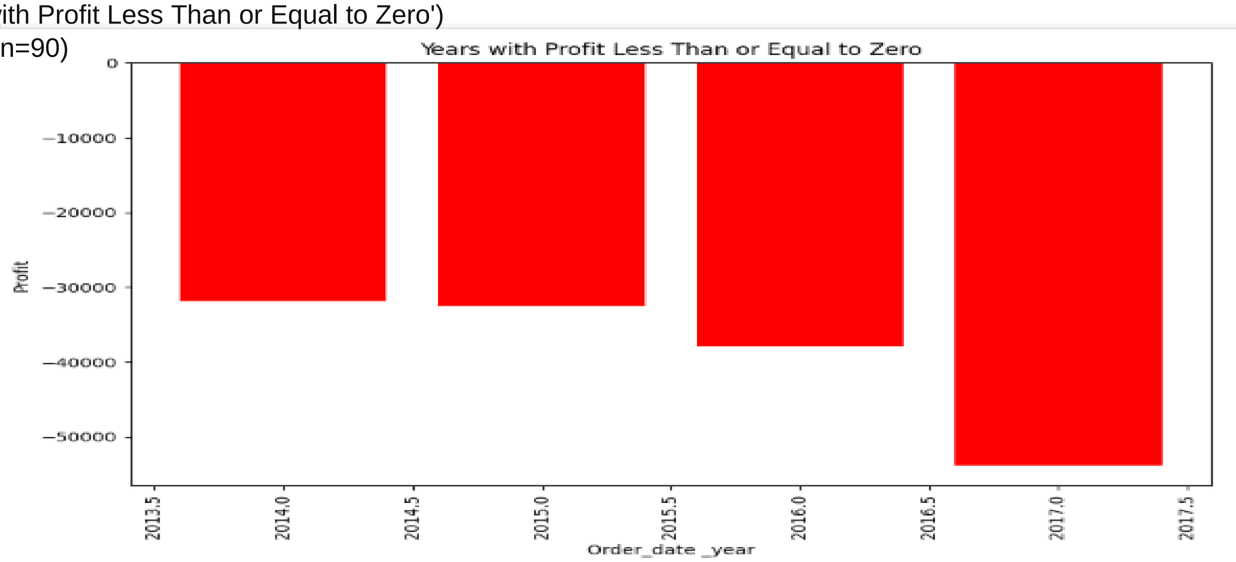


Negative profit across years

```
negative_profits = df.query('Profit <= 0')
losing_yrs_profits = negative_profits.groupby('Order_date _year')['Profit'].sum().reset_index()
topyrs_negative_profits = losing_yrs_profits.sort_values(by='Profit').head(10)
topyrs_negative_profits
```

Order_date _year	Loss
2017	-53836.1934
2016	-37872.9297
2015	-32529.3909
2014	-31892.7717

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topyrs_negative_profits['Order_date _year'], topyrs_negative_profits['Profit'], color='red')
plt.xlabel('Order_date _year')
plt.ylabel('Profit')
plt.title('Years with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```

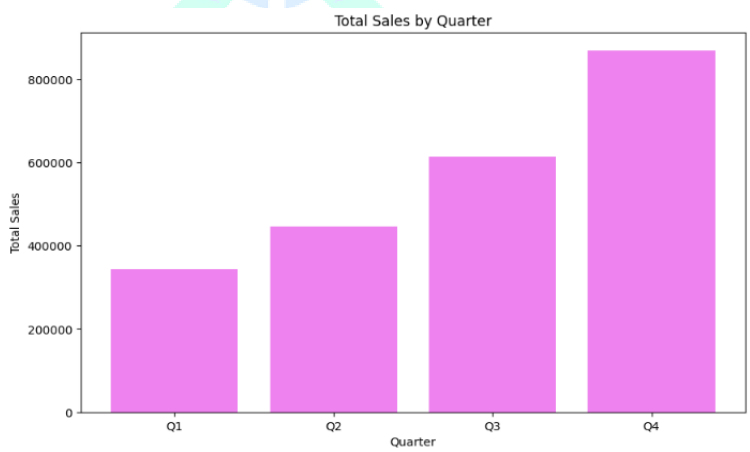


2. Sales across Quarters to assess seasonality

```
quarterly_sales = df.groupby('Quarter')['Sales'].sum().reset_index()

quarterly_sales
```

	Quarter	Sales
0	Q1	343043.1356
1	Q2	445509.6196
2	Q3	613932.1057
3	Q4	868877.6709

```
# Create a bar chart
plt.figure(figsize=(10, 6))
plt.bar(quarterly_sales['Quarter'], quarterly_sales['Sales'], color='violet')
plt.xlabel('Quarter')
plt.ylabel('Total Sales')
plt.title('Total Sales by Quarter')
plt.show ()
```



3. Regions-State-City Analysis

Regions / State/Cities contributions to sales /or Profit

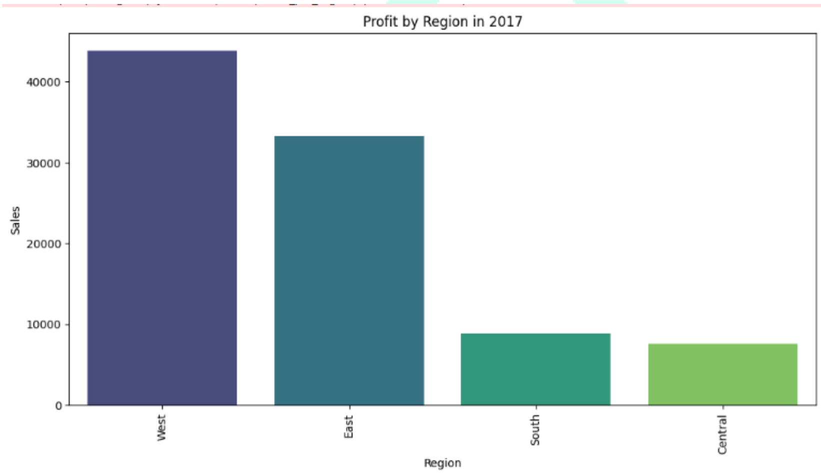
```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'Regions', and 'Sales'

specific_year = 2017

df_specific_year = df[df ['Order_date _year'] == specific_year]

# Group by 'Regions' and calculate the sum of 'Sales'

profit_per_region = df_specific_year.groupby('Region')['Profit'].sum().reset_index()

# Sort the DataFrame by 'Sales' in descending order

profit_per_region = profit_per_region.sort_values(by='Profit', ascending=False)
```

Display the result

profit_per_region.

Region	Profit
West	43808.9561
East	33230.5614
South	8848.9079
Central	7550.8442

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Region', y='profit', data=profit_per_region, palette='viridis')
# Add labels and title
plt.xlabel('Region')
plt.ylabel ('Profit')
plt.title(f'Profit by Region in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```



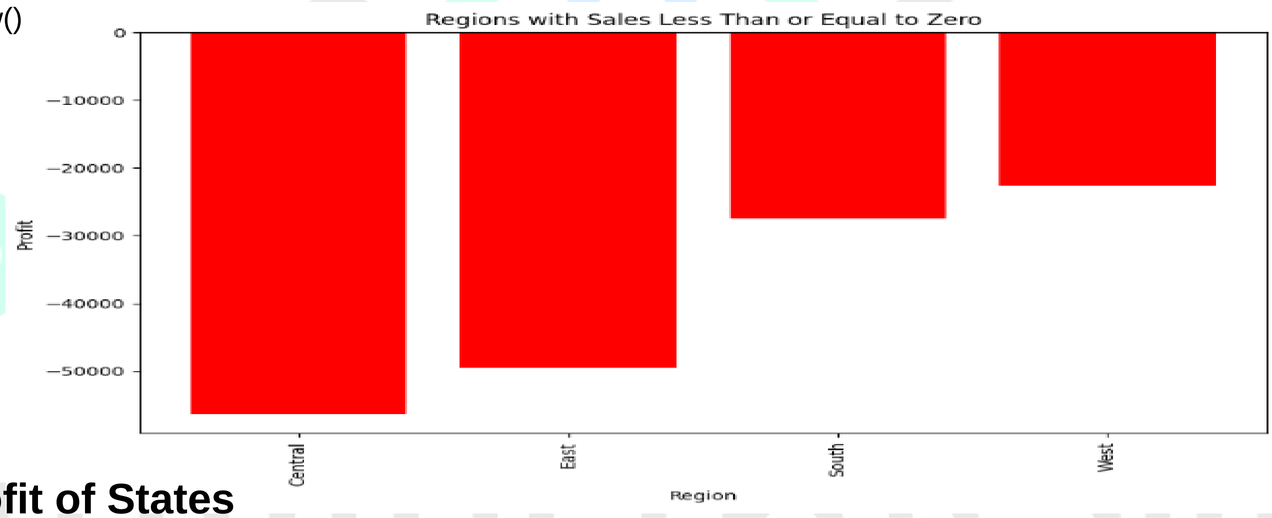
Regions with negative profits

negative_profits = df.query('Profit <= 0')
losing_region_profits = negative_profits.groupby('Region')['Profit'].sum().reset_index()
topregion_negative_profits = losing_region_profits.sort_values(by='Profit')
topregion_negative_profits

Region	Loss
Central	-56314.8850
East	-49590.6075
South	-27504.8323
West	-22720.9609

/

import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topregion_negative_profits['Region'], topregion_negative_profits['Profit'], color='red')
plt.xlabel('Region')
plt.ylabel('Profit')
plt.title('Regions with Sales Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()



Profit of States

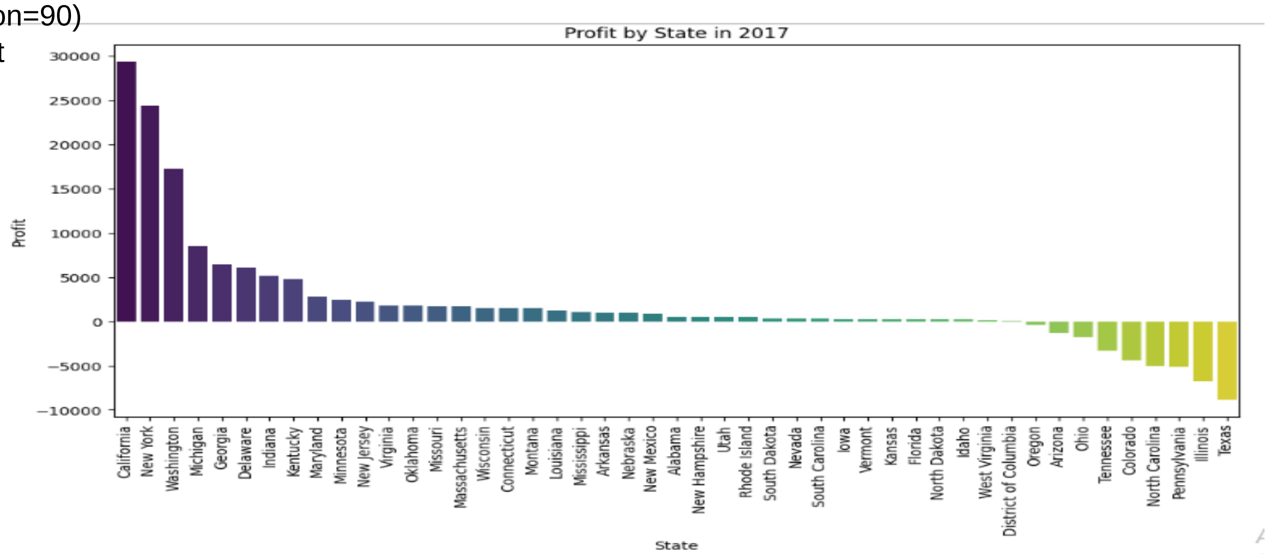
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'State', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'State' and calculate the sum of 'Sales'
profit_per_state = df_specific_year.groupby('State')['Profit'].sum().reset_index()
# Sort the DataFrame by 'Sales' in descending order
profit_per_state = profit_per_state.sort_values(by='Profit', ascending=False)
# Display the result
profit_per_state.

State	Profit
California	29366.4589
New York	24357.0717
Washington	17256.7798
Michigan	8487.7618
Georgia	6447.9819
Delaware	6053.2049
Indiana	5139.5257
Kentucky	4751.7214
Maryland	2780.6070
Minnesota	2459.8789
New Jersey	2266.0707
Virginia	1806.0146
Oklahoma	1754.7516
Missouri	1733.4305
Massachusetts	1710.5869
Wisconsin	1523.0807
Connecticut	1479.7616
Montana	1465.9255
Louisiana	1212.9041
Mississippi	1028.5620
Arkansas	959.4027
Nebraska	921.0303
New Mexico	827.3904
Alabama	496.3771

New Hampshire	480.2140
Utah	477.1747
Rhode Island	467.4135
South Dakota	346.0717
Nevada	305.1947
South Carolina	294.6375
Iowa	275.8638
Vermont	263.9759
Kansas	263.3644
Florida	244.1266
North Dakota	230.1497
Idaho	199.0086
West Virginia	185.9216
District of Columbia	35.0640
Oregon	-377.1257
Arizona	-1276.0025
Ohio	-1736.5270
Tennessee	-3304.2866
Colorado	-4435.8483
North Carolina	-5088.5334
Pennsylvania	-5112.8034
Illinois	-6745.5600
Texas	-8838.5049

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='State', y='Sales', data=sales_per_state, palette='viridis')
# Add labels and title
```

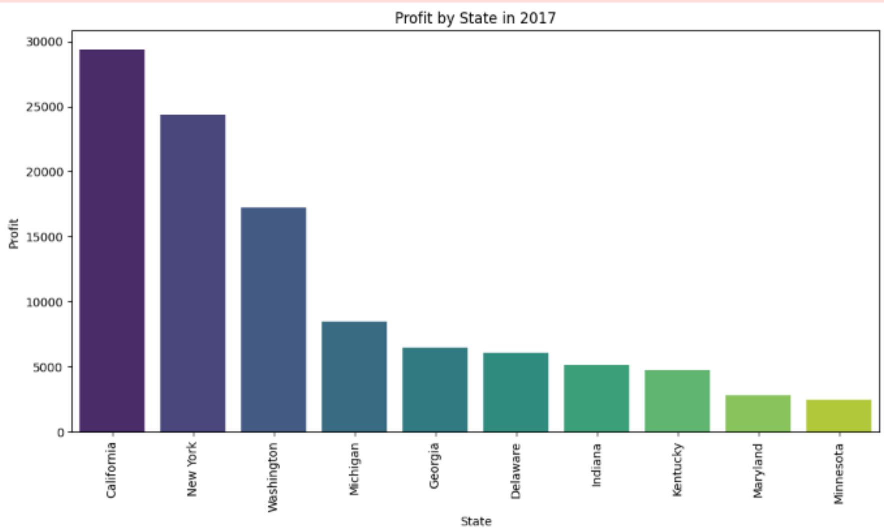
```
plt.xlabel('State')
plt.ylabel('Sales')
plt.title(f'Sales by State in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```




```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'State', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'State' and calculate the sum of 'Sales'
profit_per_state = df_specific_year.groupby('State')['Profit'].sum().reset_index()
# Sort the DataFrame by 'Sales' in descending order
profit_per_state = profit_per_state.sort_values(by='Profit', ascending=False).head(10)
# Display the result
profit_per_state.
```

State	Profit
California	29366.4589
New York	24357.0717
Washington	17256.7798
Michigan	8487.7618
Georgia	6447.9819
Delaware	6053.2049
Indiana	5139.5257
Kentucky	4751.7214
Maryland	2780.6070
Minnesota	2459.8789

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='State', y='Profit', data=profit_per_state, palette='viridis')
# Add labels and title
plt.xlabel('State')
plt.ylabel('Profit')
plt.title(f'Profit by State in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```



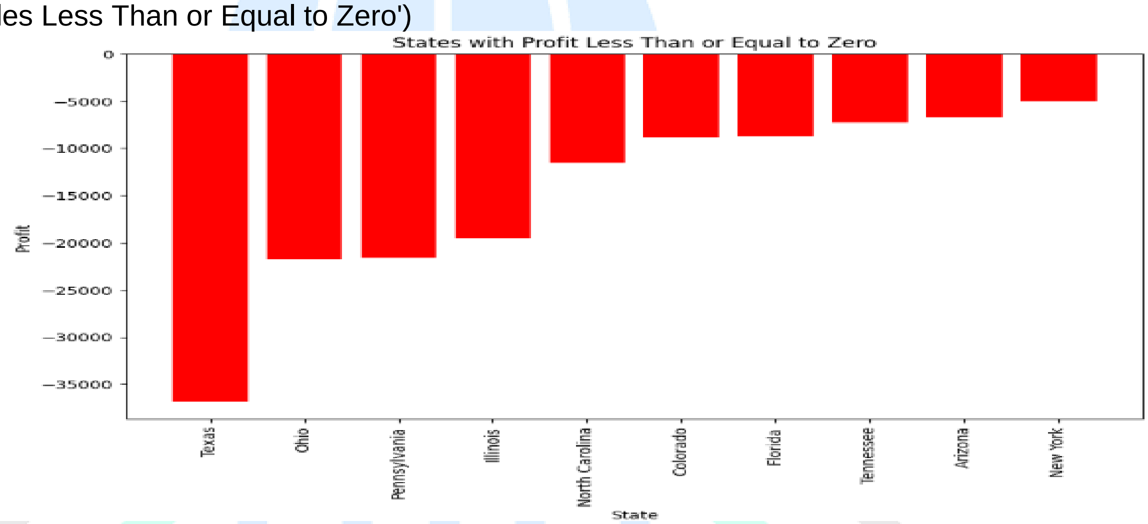
States with Negative Profit

```
negative_profits = df.query('Profit <= 0')
losing_State_profits = negative_profits.groupby('State')['Profit'].sum().reset_index()
topstate_negative_profits = losing_State_profits.sort_values(by='Profit').head(10)
topstate_negative_profits
```

State	Loss
Texas	-36813.1875
Ohio	-21750.0002
Pennsylvania	-21602.8515
Illinois	-19501.6975
North Carolina	-11557.9854

Colorado	-8900.9048
Florida	-8689.8295
Tennessee	-7257.0174
Arizona	-6656.7675
New York	-5031.1378

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topstate_negative_profits['State'], topstate_negative_profits['Profit'], color='red')
plt.xlabel('State')
plt.ylabel('Profit')
plt.title('States with Sales Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```



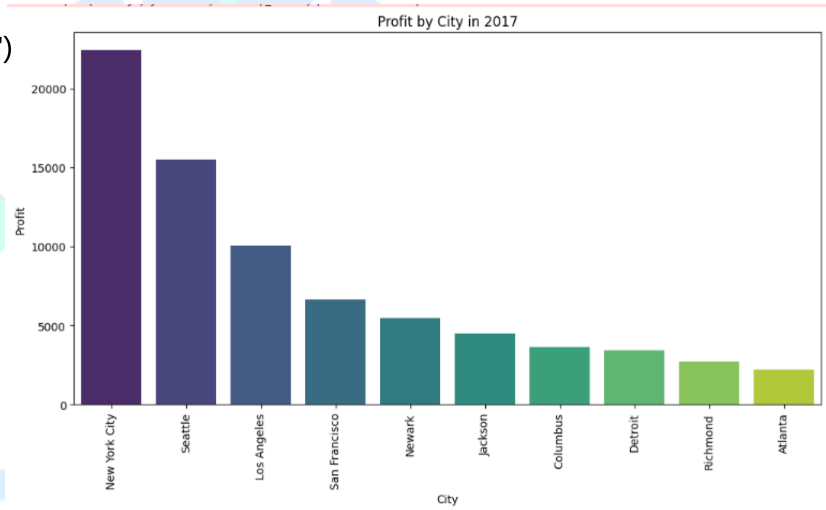
Profit by Cities

```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'State', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'State' and calculate the sum of 'Sales'
profit_per_city = df_specific_year.groupby('City')['Profit'].sum().reset_index()
# Sort the DataFrame by 'Sales' in descending order
top_cities = profit_per_city.sort_values(by='Profit', ascending=False).head(10)
# Display the result
top_cities
```

City	Profit
New York City	22406.0271
Seattle	15518.6970

Los Angeles	10059.2901
San Francisco	6617.9550
Newark	5468.2674
Jackson	4520.9059
Columbus	3615.2543
Detroit	3417.3567
Richmond	2738.7960
Atlanta	2232.2704

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='City', y='Profit', data=top_cities, palette='viridis')
# Add labels and title
plt.xlabel('City')
plt.ylabel('Profit')
plt.title(f'Profit by City in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```

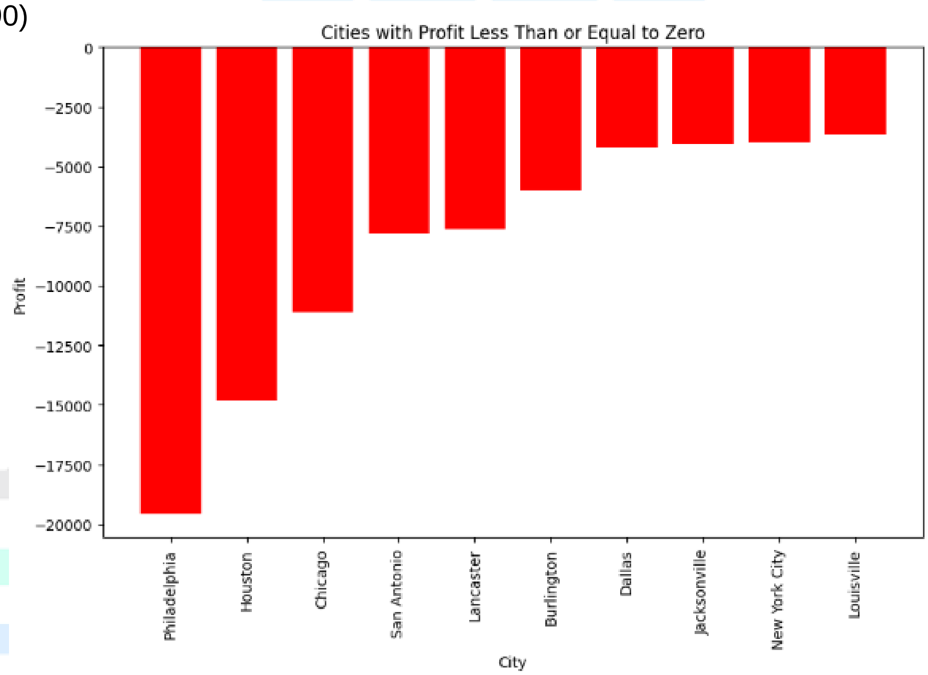



```
negative_profits = df.query('Profit <= 0')
losing_city_profits = negative_profits.groupby('City')['Profit'].sum().reset_index()
topcity_negative_profits = losing_city_profits.sort_values(by='Profit').head(10)
topcity_negative_profits
```

City	Loss
Philadelphia	-19590.7411
Houston	-14785.3668
Chicago	-11120.6271

San Antonio	-7831.0254
Lancaster	-7632.4946
Burlington	-5999.3318
Dallas	-4208.5218
Jacksonville	-4059.9857
New York City	-3966.0226
Louisville	-3694.1045

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topcity_negative_profits['City'], topcity_negative_profits['Profit'], color='red')
plt.xlabel('City')
plt.ylabel('Profit')
plt.title('Cities with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```



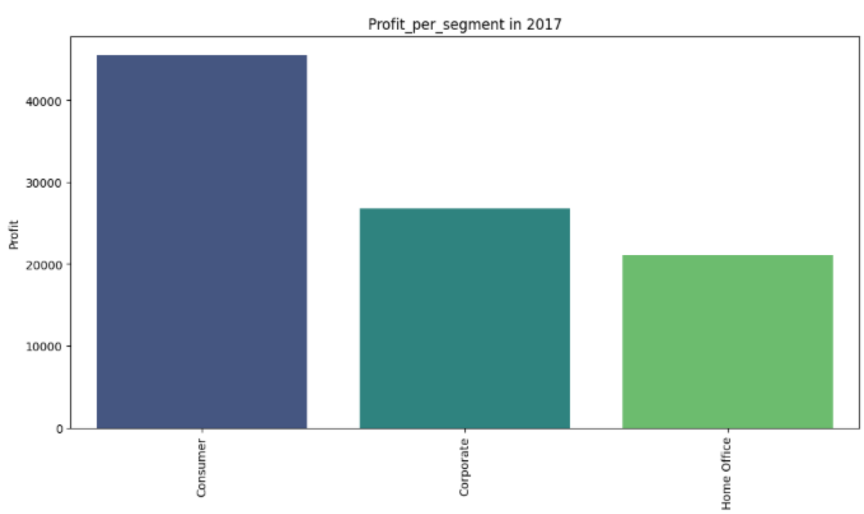
3. Customer Analysis

Profit by Customer Segment

```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'Segment', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'Segment' and calculate the sum of 'Sales'
Profit_per_segment = df_specific_year.groupby('Segment')['Profit'].sum().reset_index()
# Sort the DataFrame by 'Profit' in descending order
Profit_per_segment = Profit_per_segment.sort_values(by='Profit', ascending=False)
```

	Segment	Profit
0	Consumer	45568.2391
1	Corporate	26782.3633
2	Home Office	21088.6672

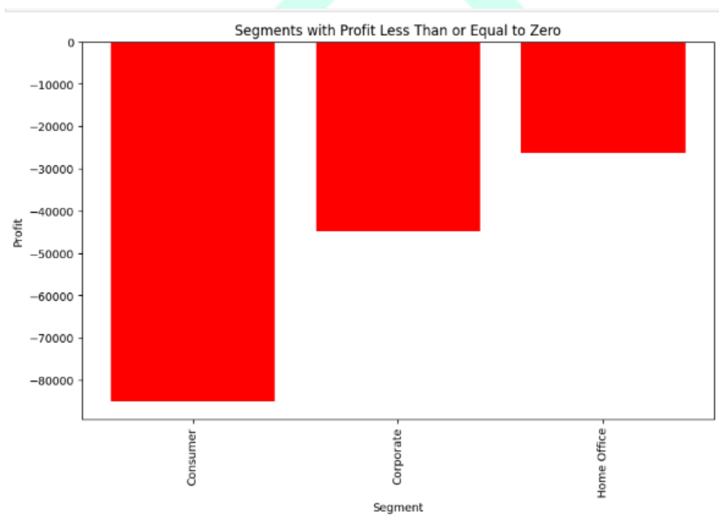
```
# Display the result
print(Profit_per_segment)
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Segment', y='Profit', data=Profit_per_segment, palette='viridis')
# Add labels and title
plt.xlabel('Segment')
plt.ylabel('Profit')
plt.title(f'Profit_per_segment in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```



```
negative_profits = df.query('Profit <= 0')
losing_seg_profits = negative_profits.groupby('Segment')['Profit'].sum().reset_index()
topseg_negative_profits = losing_seg_profits.sort_values(by='Profit').head(10)
print(topseg_negative_profits)
```

	Segment	Loss
0	Consumer	-84945.7112
1	Corporate	-44787.2076
2	Home Office	-26398.3669

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topseg_negative_profits['Segment'], topseg_negative_profits['Profit'], color='red')
plt.xlabel('Segment')
plt.ylabel('Profit')
plt.title('Segments with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```

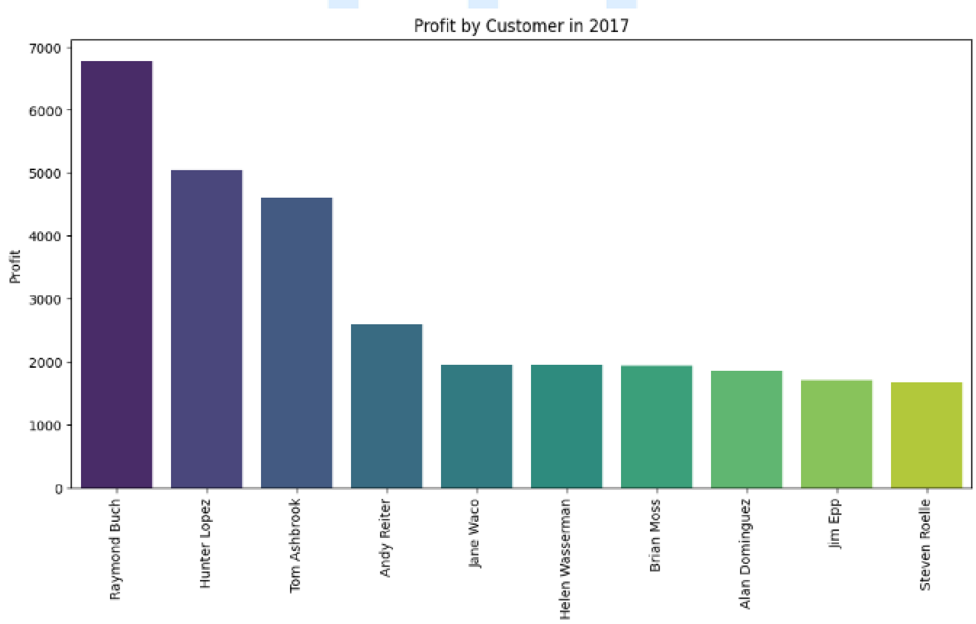


Sales Per Customers

```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'Customer', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'Customer' and calculate the sum of 'Sales'
profit_per_customer = df_specific_year.groupby('Customer_Name')['Profit'].sum().reset_index()
# Sort and get top ten customers
top_customers = profit_per_customer.sort_values(by='Profit', ascending=False).head(10)
# Display the result
top_customers
```

Customer Name	Profit
Raymond Buch	6780.8963
Hunter Lopez	5045.8564
Tom Ashbrook	4599.2073
Andy Reiter	2607.6814
Jane Waco	1953.2680
Helen Wasserman	1946.6943
Brian Moss	1938.1873
Alan Dominguez	1866.9279
Jim Epp	1703.5561
Steven Roelle	1676.3122

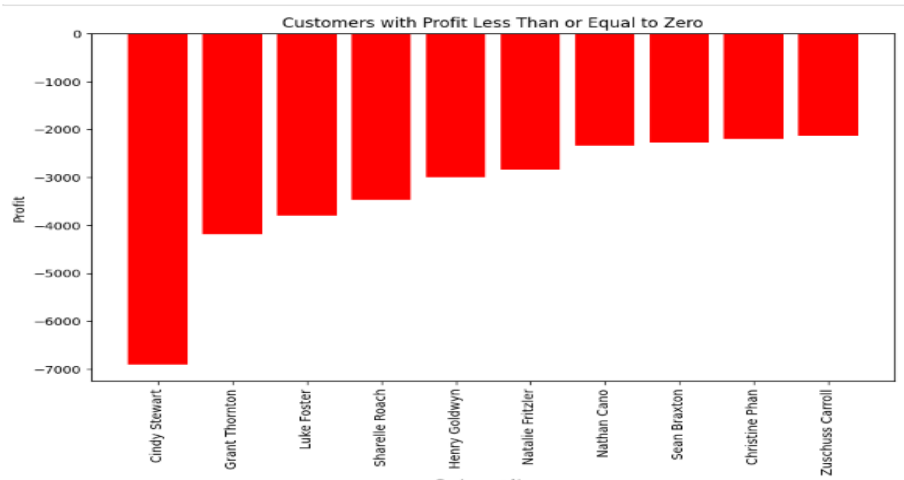
```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Customer_Name', y='Profit', data=top_customers, palette='viridis')
# Add labels and title
plt.xlabel('Customer_Name')
plt.ylabel('Profit')
plt.title(f'Profit by Customer in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```



```
negative_profits = df.query('Profit <= 0')

losing_cust_profits = negative_profits.groupby('Customer_Name')['Profit'].sum().reset_index()

topcust_negative_profits = losing_cust_profits.sort_values(by='Profit').head(10)

print(topcust_negative_profits)
```

	Customer Name	Loss
128	Cindy Stewart	-6904.3700
249	Grant Thornton	-4187.1078
389	Luke Foster	-3805.5490
571	Sharelle Roach	-3467.1258
268	Henry Goldwyn	-2998.8345
462	Natalie Fritzler	-2833.4696
464	Nathan Cano	-2342.6546
562	Sean Braxton	-2279.3647
119	Christine Phan	-2191.5504
650	Zuschuss Carroll	-2130.2964

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topcust_negative_profits['Customer_Name'], topcust_negative_profits['Profit'], color='red')
plt.xlabel('Customer_Name')
plt.ylabel('Profit')
plt.title('Customers with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```



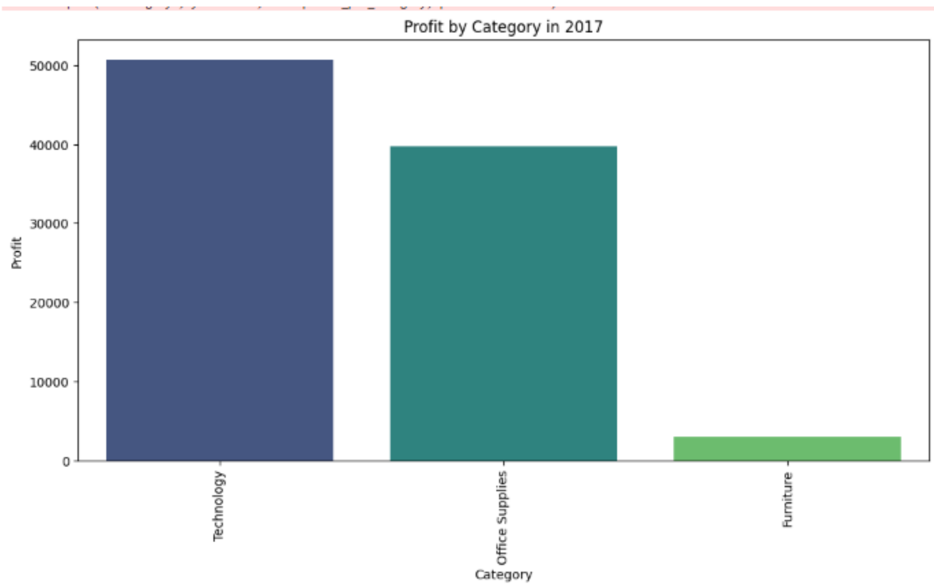
4. Category-Sub Category – Product analysis

Sales per Category

```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'Category', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'Category' and calculate the sum of 'Sales'
profit_per_category = df_specific_year.groupby('Category')['Profit'].sum().reset_index()
# Sort the DataFrame by 'Sales' in descending order
profit_per_category = profit_per_category.sort_values(by='Profit', ascending=False)
# Display the result
profit_per_category.
```

Category	Profit
Technology	50684.2566
Office Supplies	39736.6217
Furniture	3018.3913

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Category', y='Profit', data=profit_per_category, palette='viridis')
# Add labels and title
plt.xlabel('Category')
plt.ylabel('Profit')
plt.title(f'Profit by Category in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```



negative_profits = df.query('Profit <= 0')

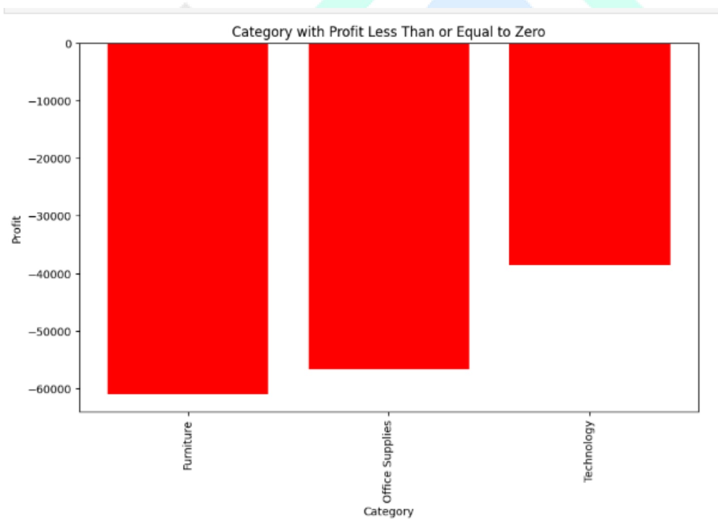
losing_categ_profits = negative_profits.groupby('Category')['Profit'].sum().reset_index()

topcat_negative_profits = losing_categ_profits.sort_values(by='Profit').head(10)

topcat_negative_profits

Category	Loss
Furniture	-60936.1090
Office Supplies	-56615.2585
Technology	-38579.9182

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topcat_negative_profits['Category'], topcat_negative_profits['Profit'], color='red')
plt.xlabel('Category')
plt.ylabel('Profit')
plt.title('Category with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```



Sales per Sub_Category

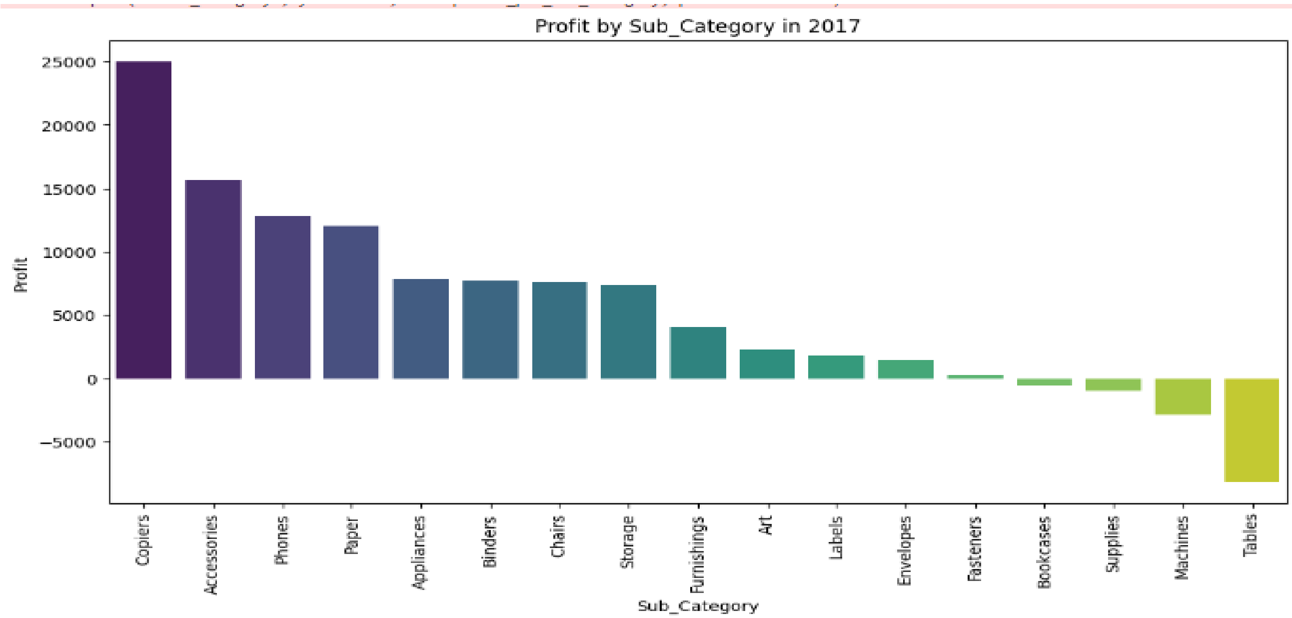
```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'Sub_Category', and 'Profit'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'Sub_Category' and calculate the sum of 'Sales'
profit_per_sub_category = df_specific_year.groupby('Sub_Category')['Profit'].sum().reset_index()
# Sort the DataFrame by 'Sales' in descending order
profit_per_sub_category = profit_per_sub_category.sort_values(by='Profit', ascending=False)
```

```
# Display the result
profit_per_sub_category.
```

Sub_Category	Profit
Copiers	25031.7902
Accessories	15672.3570
Phones	12849.3250
Paper	12040.8434
Appliances	7865.2683
Binders	7669.7418
Chairs	7643.5493
Storage	7402.8007
Furnishings	4099.1628
Art	2221.9631
Labels	1744.6093
Envelopes	1441.7590
Fasteners	304.9489
Bookcases	-583.6261
Supplies	-955.3128
Machines	-2869.2156
Tables	-8140.6947

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Sub_Category', y='Profit', data=profit_per_sub_category, palette='viridis')
# Add labels and title
plt.xlabel('Sub_Category')
plt.ylabel('Profit')
plt.title(f'Profit by Sub_Category in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
```

plt.show()



negative_profits = df.query('Profit <= 0')

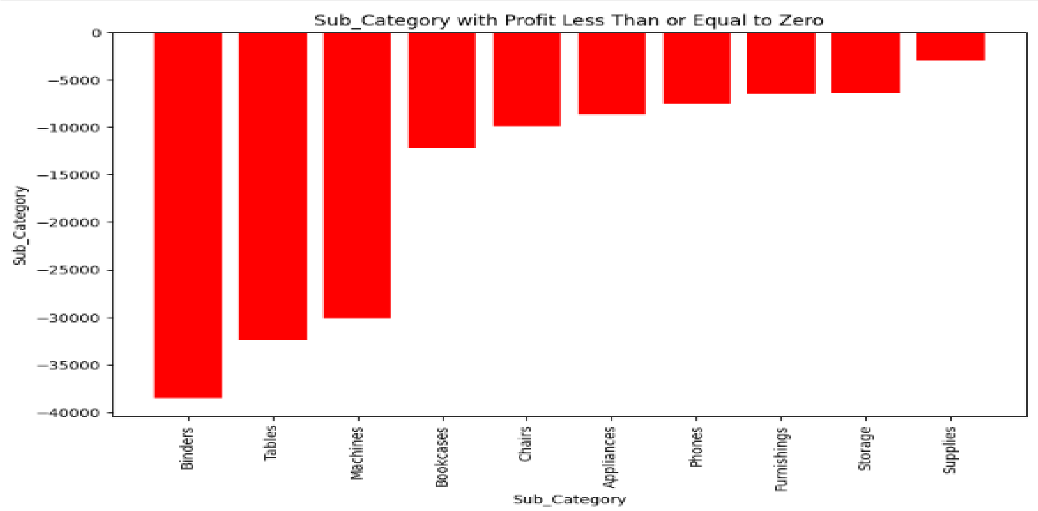
losing_Subcateg_profits = negative_profits.groupby('Sub_Category')['Profit'].sum().reset_index()

topsubcat_negative_profits = losing_Subcateg_profits.sort_values(by='Profit').head(10)

topsubcat_negative_profits

Sub_Category	Loss
Binders	-38510.4964
Tables	-32412.1483
Machines	-30118.6682
Bookcases	-12152.2060
Chairs	-9880.8413
Appliances	-8629.6412
Phones	-7530.6235
Furnishings	-6490.9134
Storage	-6426.3038
Supplies	-3015.6219

```
import matplotlib.pyplot as plt.
plt.figure(figsize=(10, 6))
plt.bar(topsubcat_negative_profits['Sub_Category'], topsubcat_negative_profits['Profit'], color='red')
plt.xlabel('Sub_Category')
plt.ylabel('Sub_Category')
plt.title('Sub_Category with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()
```



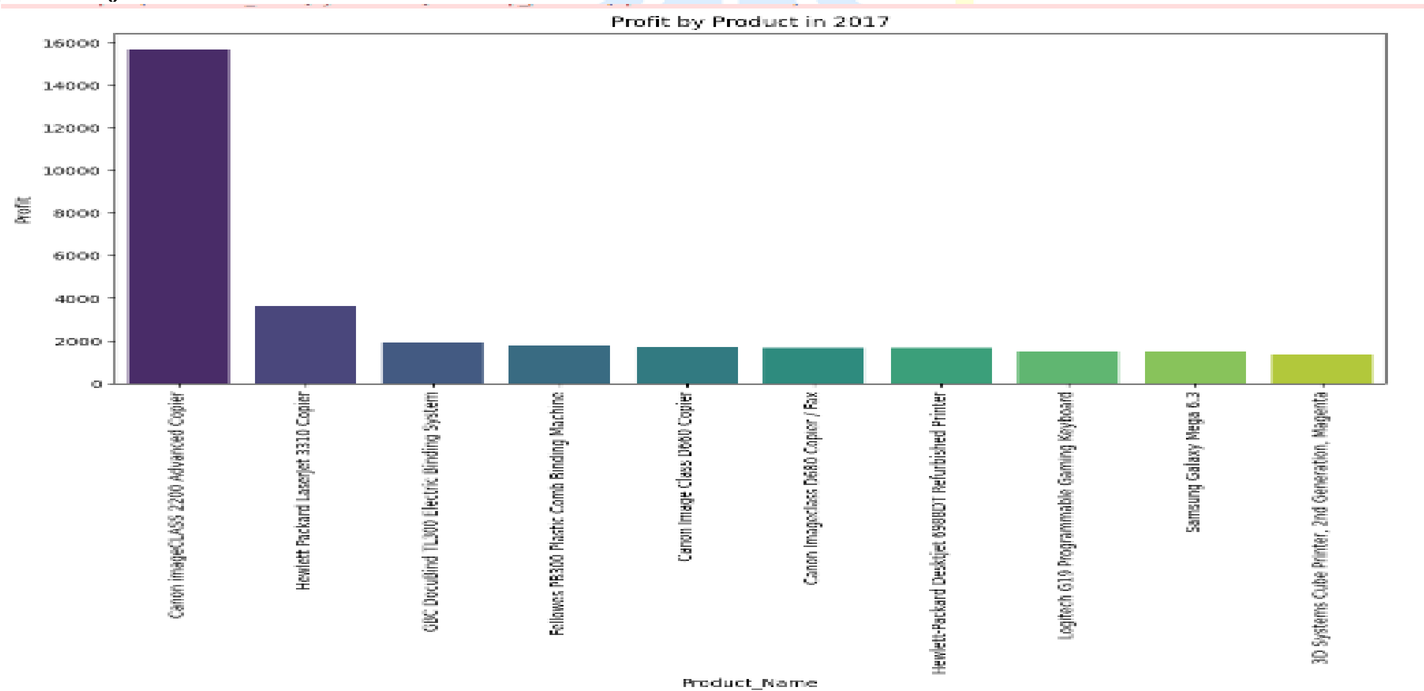
Sales per Product

```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'product', and 'Sales'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'Sub_Category' and calculate the sum of 'Sales'
profit_per_product = df_specific_year.groupby('Product_Name')['Profit'].sum().reset_index()
# Sort and get top ten products
top_products = profit_per_product.sort_values(by='Profit', ascending=False).head(10)
# Display the result
top_products
```

Product_Name	**Profit**
Canon imageCLASS 2200 Advanced Copier	15679.9552
Hewlett Packard LaserJet 3310 Copier	3623.9396
GBC DocuBind TL300 Electric Binding System	1910.5887
Fellowes PB300 Plastic Comb Binding Machine	1753.7148
Canon Image Class D660 Copier	1691.9718
Canon Imageclass D680 Copier / Fax	1679.9760
Hewlett-Packard Desktjet 6988DT Refurbished Pr...	1668.2050
Logitech G19 Programmable Gaming Keyboard	1493.3574
Samsung Galaxy Mega 6.3	1469.9650

3D Systems Cube Printer, 2nd Generation, Magenta	1351.9896

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Product_Name', y='Profit', data=top_products, palette='viridis')
# Add labels and title
plt.xlabel('Product_Name')
plt.ylabel('Profit')
plt.title(f'Profit by Product in {specific_year}')
plt.xticks(rotation=90)
# Show the plot
plt.show()
```




```
negative_profits = df.query('Profit <= 0')

losing_product_profits = negative_profits.groupby('Product_Name')['Profit'].sum().reset_index()

product_negative_profits = losing_product_profits.sort_values(by='Profit')

product_negative_profits
```


Product_Name	Loss
Cubify CubeX 3D Printer Double Head Print	-9239.9692
GBC DocuBind P400 Electric Binding System	-6859.3896
Lexmark MX611dhe Monochrome Laser Printer	-5269.9690

GBC Ibimaster 500 Manual ProClick Binding System	-5098.5660
GBC DocuBind TL300 Electric Binding System	-4162.0336
...	...
Sauder Facets Collection Locker/File Cabinet, ...	0.0000
Belkin OmniView SE Rackmount Kit	0.0000
Safco Value Mate Steel Bookcase, Baked Enamel ...	0.0000
Tenex 46" x 60" Computer Anti-Static Chairmat,...	0.0000
Eldon Radial Chair Mat for Low to Medium Pile ...	0.0000

negative_profits = df.query('Profit <= 0')

losing_product_profits = negative_profits.groupby('Product_Name')['Profit'].sum().reset_index()

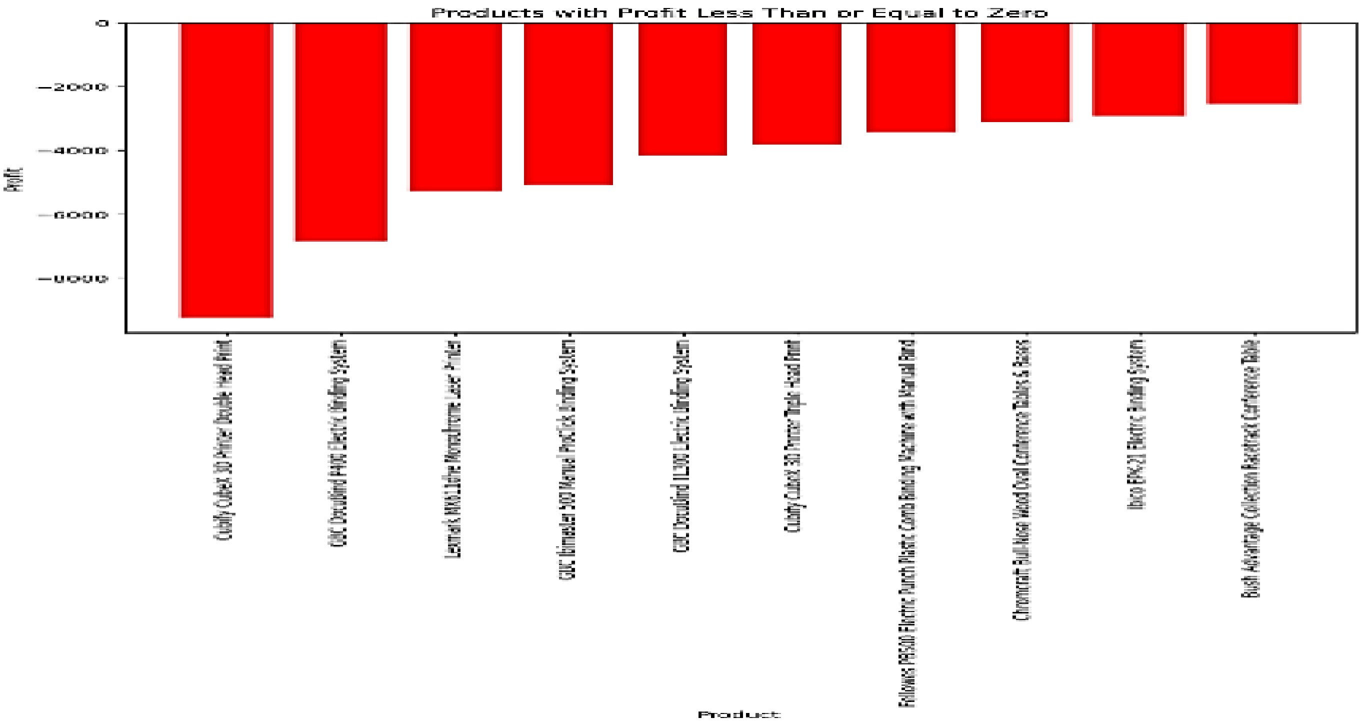
top_negative_profits = losing_product_profits.sort_values(by='Profit').head(10)

top_negative_profits

Product_Name	**Profit**
Cubify CubeX 3D Printer Double Head Print	-9239.9692
GBC DocuBind P400 Electric Binding System	-6859.3896
Lexmark MX611dhe Monochrome Laser Printer	-5269.9690
GBC Ibimaster 500 Manual ProClick Binding System	-5098.5660
GBC DocuBind TL300 Electric Binding System	-4162.0336
Cubify CubeX 3D Printer Triple Head Print	-3839.9904
Fellowes PB500 Electric Punch Plastic Comb Bin...	-3431.6730
Chromcraft Bull-Nose Wood Oval Conference Tabl...	-3107.5272
Ibico EPK-21 Electric Binding System	-2929.4845
Bush Advantage Collection Racetrack Conference...	-2545.2600

import matplotlib.pyplot as plt.

plt.figure(figsize=(10, 6))
plt.bar(top_negative_profits['Product_Name'], top_negative_profits['Profit'], color='red')
plt.xlabel('Product')
plt.ylabel('Profit')
plt.title('Products with Profit Less Than or Equal to Zero')
plt.xticks(rotation=90)
plt.show()



5. Correlations between Sales & Profit to

- Price
- Quantity
- Discount
- Shipping mode
- Time frame

	Row_ID	Order_date_month	Order_date_day	Order_date_year	Time_Frame	Ship_Mode_Rank	Postal_Code	item_price	Quantity	Discount	Sales	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	4997.500000	7.809686	15.468481	2015.722233	3.958175	3.334601	55190.379428	74.845245	3.789574	0.156203	227.272617	28.656896
std	2885.163629	3.284654	8.748327	1.123555	1.747567	0.925478	32063.693350	171.582405	2.225110	0.206452	560.821012	234.260108
min	1.000000	1.000000	1.000000	2014.000000	0.000000	1.000000	1040.000000	0.990000	1.000000	0.000000	0.444000	-6599.978000
25%	2499.250000	5.000000	8.000000	2015.000000	3.000000	3.000000	23223.000000	6.480000	2.000000	0.000000	17.280000	1.728750
50%	4997.500000	9.000000	15.000000	2016.000000	4.000000	4.000000	56430.500000	19.980000	3.000000	0.200000	54.490000	8.666500
75%	7495.750000	11.000000	23.000000	2017.000000	5.000000	4.000000	90008.000000	76.980000	5.000000	0.200000	209.940000	29.364000
max	9994.000000	12.000000	31.000000	2017.000000	7.000000	4.000000	99301.000000	3999.990000	14.000000	0.800000	9999.999000	8399.976000

```
# Assuming df is your original DataFrame and it has columns 'Order_date _year', 'Product_Name', 'Profit', and 'item_price'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]

# Group by 'Product_Name' and calculate the sum of 'Profit'
sales_per_product = df_specific_year.groupby('Product_Name')['Sales'].sum().reset_index()

# Sort and get top ten products
top_products = sales_per_product.sort_values(by='Sales', ascending=False).head(10)

# Merge to get item_price for the top products
top_products_with_price = top_products.merge(df[['Product_Name', 'item_price','Quantity','Discount']], on='Product_Name', how='left').drop_duplicates()

# Display the result
top_products_with_price
```

Product_Name	Sales	item_price	Quantity	Discount
Canon imageCLASS 2200 Advanced Copier	29999.9997	3125.00	4	0.2
Canon imageCLASS 2200 Advanced Copier	29999.9997	3333.33	3	0.0
Canon imageCLASS 2200 Advanced Copier	29999.9997	3499.99	4	0.4
Canon imageCLASS 2200 Advanced Copier	29999.9997	2000.00	5	0.0
Canon imageCLASS 2200 Advanced Copier	29999.9997	2500.00	4	0.0
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	832.81	1	0.2
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	832.81	7	0.2
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	832.81	5	0.0
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	832.81	2	0.0
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	832.81	2	0.2
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	2	0.0
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	3	0.0
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	5	0.8
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	2	0.7
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	5	0.7
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	3	0.2
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	6	0.7
GBC DocuBind TL300 Electric Binding System	10943.2780	896.99	6	0.2

```
# Assuming df is your original DataFrame and it has columns 'Order_date _year', 'Product_Name', 'Profit', and 'item_price'
specific_year = 2017
df_specific_year = df[df['Order_date _year'] == specific_year]
# Group by 'Product_Name' and calculate the sum of 'Profit'
sales_per_product = df_specific_year.groupby('Product_Name')['Sales'].sum().reset_index()
# Sort and get top ten products
top_products = sales_per_product.sort_values(by='Sales', ascending=False).head(10)
# Merge to get item_price for the top products
top_products_with_Details = top_products.merge(df[['Product_Name','Ship_Mode_Rank','Time_Frame']],
on='Product_Name', how='left').drop_duplicates()
# Display the result
top_products_with_Details
```

Product_Name	Sales	Ship_Mode_Rank	Time_Frame
Canon imageCLASS 2200 Advanced Copier	29999.9997	2	2
Canon imageCLASS 2200 Advanced Copier	29999.9997	4	5
Canon imageCLASS 2200 Advanced Copier	29999.9997	4	4
Canon imageCLASS 2200 Advanced Copier	29999.9997	4	7
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	4	4
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	3	4
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	2	2
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	4	7
Martin Yale Chadless Opener Electric Letter Op...	11825.9020	3	5
GBC DocuBind TL300 Electric Binding System	10943.2780	2	3
GBC DocuBind TL300 Electric Binding System	10943.2780	4	7
GBC DocuBind TL300 Electric Binding System	10943.2780	4	4
GBC DocuBind TL300 Electric Binding System	10943.2780	2	2
GBC DocuBind TL300 Electric Binding System	10943.2780	4	6
GBC DocuBind TL300 Electric Binding System	10943.2780	4	5
Hewlett Packard LaserJet 3310 Copier	9239.8460	3	3
Hewlett Packard LaserJet 3310 Copier	9239.8460	2	2
Hewlett Packard LaserJet 3310 Copier	9239.8460	2	1
Hewlett Packard LaserJet 3310 Copier	9239.8460	3	2

Correlation between Sales & Price

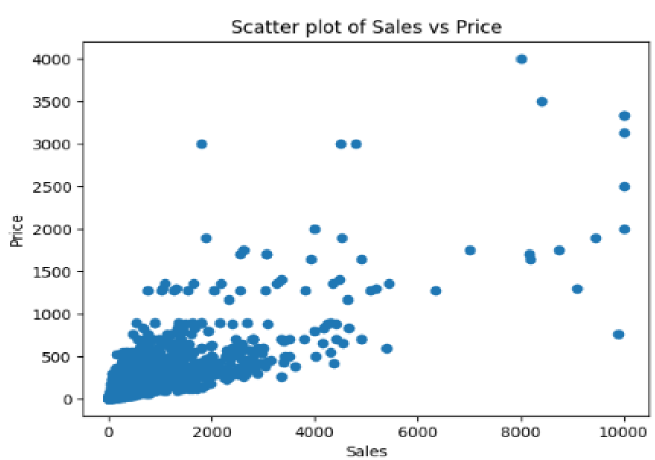
```
# Calculate the correlation
correlation = df[['Sales', 'item_price']].corr()
correlation
```

	Sales	item_price
Sales	1.000000	0.829405
item_price	0.829405	1.000000

Sales and Item Price (0.829405): Strong positive correlation. This suggests that as the item price increases, sales tend to increase as well. This could mean that higher-priced items are selling well, or that there is a general trend where pricier items contribute more to sales.

In short, your data indicates a strong relationship between item prices and sales, implying that higher item prices are associated with higher sales.

```
# Visualize the correlation using a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation between Sales and Price')
plt.show()
# Visualize the relationship using a scatter plot
plt.scatter(df['Sales'], df['item_price'])
plt.xlabel('Sales')
plt.ylabel('Price')
plt.title('Scatter plot of Sales vs Price')
plt.show()
```

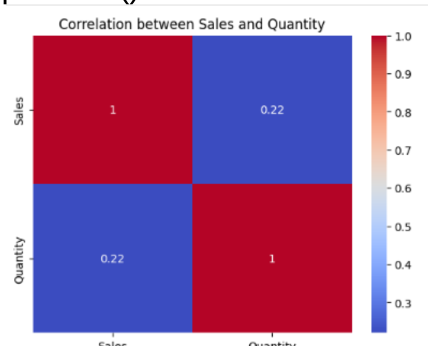


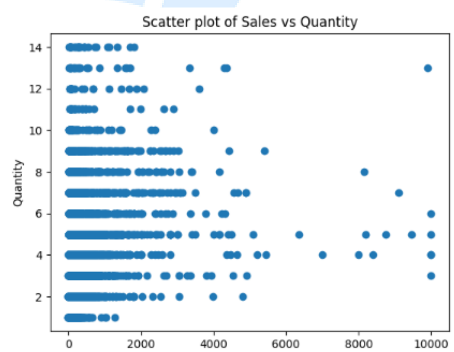
Correlation between Sales & Quantity

```
# Calculate the correlation
correlation = df[['Sales', 'Quantity']].corr()
correlation
```

	Sales	Quantity
Sales	1.000000	0.220121
Quantity	0.220121	1.000000

```
# Visualize the correlation using a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation between Sales and Quantity')
plt.show()

# Visualize the relationship using a scatter plot
plt.scatter(df['Sales'], df['Quantity'])
plt.xlabel('Sales')
plt.ylabel('Quantity')
plt.title('Scatter plot of Sales vs Quantity')
plt.show()
```





Sales and Quantity (0.220121): A weak positive correlation. This suggests that as the quantity sold increases, sales tend to increase slightly, but the relationship is not very strong. So, in essence, while there is a slight positive relationship between sales and quantity, it is not particularly strong. What kind of relationship were you hoping to see here?

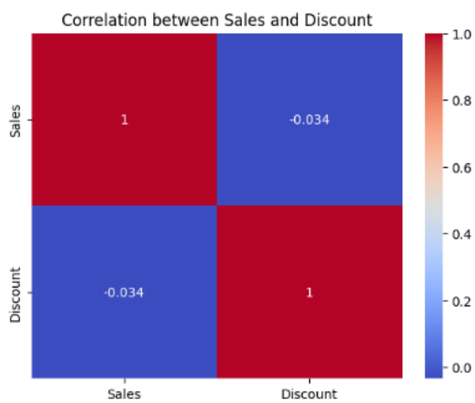
Correlation between Sales & Discount

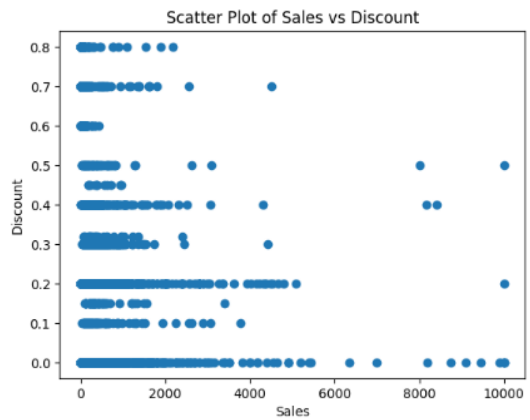
```
correlation = df[['Sales', 'Discount']].corr()
print(correlation)
```

	Sales	Discount
Sales	1.000000	-0.033509
Discount	-0.033509	1.000000

```
# Visualize the correlation using a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation between Sales and Discount')
plt.show()
```

```
# Visualize the relationship using a scatter plot
plt.scatter(df['Sales'], df['Discount'])
plt.xlabel('Sales')
plt.ylabel('Discount')
plt.title('Scatter Plot of Sales vs Discount')
plt.show()
```





Sales and Discount (-0.033509): Very weak negative correlation. This suggests that there is almost no relationship between sales and discounts. In other words, changes in discount levels do not have a strong impact on sales in your data.

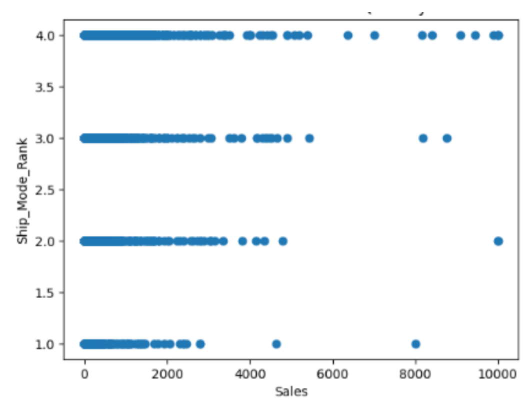
In short, your data indicates that there is no meaningful relationship between discounts and sales in your dataset.

Correlation between Sales & Ship Mode Rank

```
# Calculate the correlation
correlation = df[['Sales', 'Ship_Mode_Rank']].corr()
correlation
```

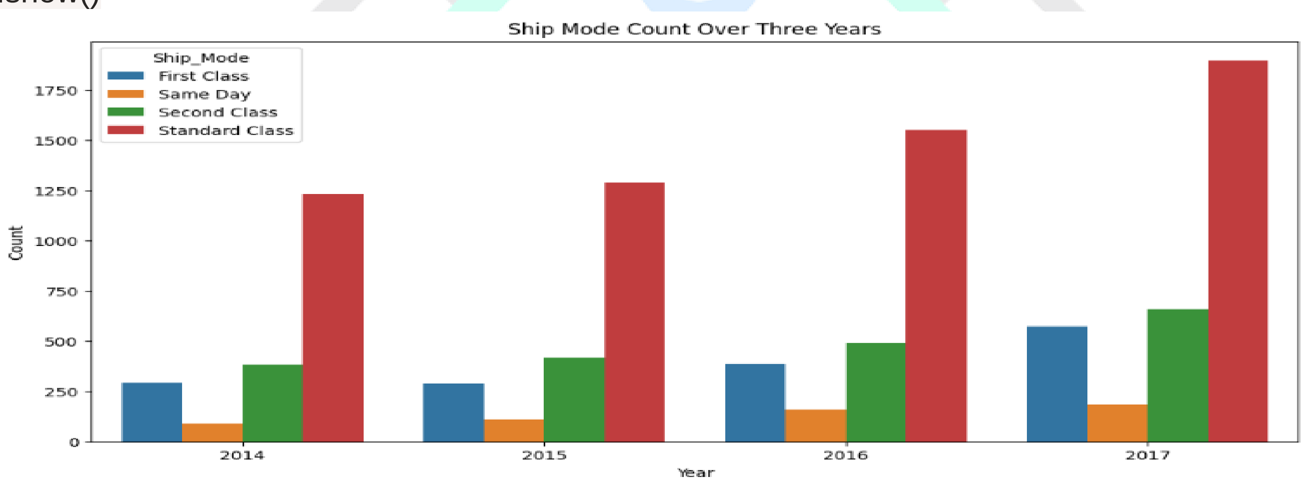
	Sales	Ship_Mode_Rank
Sales	1.000000	-0.004893
Ship_Mode_Rank	-0.004893	1.000000

```
# Visualize the correlation using a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation between Sales and Ship_Mode_Rank')
plt.show()

# Visualize the relationship using a scatter plot
plt.scatter(df['Sales'], df['Ship_Mode_Rank'])
plt.xlabel('Sales')
plt.ylabel('Ship_Mode_Rank')
plt.title('Scatter Plot of Discount vs Quantity')
plt.show()
```



Sales and Ship Mode Rank (-0.004893): Extremely weak negative correlation. This suggests that changes in the rank of shipping mode have almost no impact on sales.
In essence, your data shows that the shipping mode's rank does not significantly affect sales.

```
# Set the figure size
plt.figure(figsize=(12, 6))
# Create the bar plot
sns.barplot(x='Order_date _year', y='count', hue='Ship_Mode', data=ship_mode_counts)
# Add labels and title
plt.xlabel('Year')
plt.ylabel('Count')
plt.title('Ship Mode Count Over Three Years')
# Show the plot
plt.show()
```

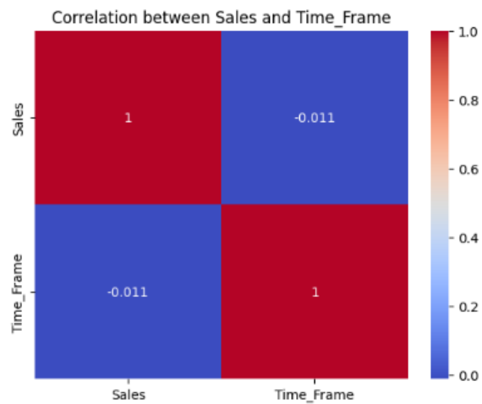


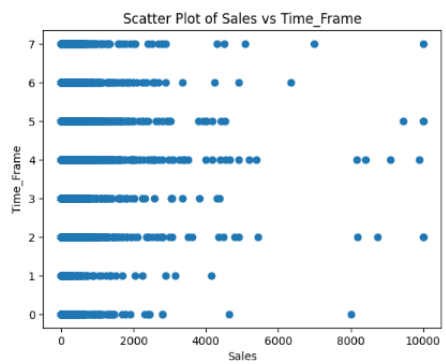
Correlation between Sales & Time_Frame:

```
# Calculate the correlation
correlation = df[['Sales', 'Time_Frame']].corr()
print(correlation)
```

	Sales	Time_Frame
Sales	1.000000	-0.010859
Time_Frame	-0.010859	1.000000

```
# Visualize the correlation using a heatmap
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title('Correlation between Sales and Time_Frame')
plt.show()
# Visualize the relationship using a scatter plot
plt.scatter(df['Sales'], df['Time_Frame'])
plt.xlabel('Sales')
plt.ylabel('Time_Frame')
plt.title('Scatter Plot of Sales vs Time_Frame')
plt.show()
```





Sales and Time Frame (-0.010859): Extremely weak negative correlation. This suggests that the time frame has virtually no impact on sales. In essence, your data shows that the time frame does not significantly affect sales.

Correlation bets ween Profit & :

- Price
- Quantity
- Discount
- Ship_Mode_Rank
- Timeframe

The same correlations as in sales

[30]:
```
# Calculate the correlation
correlation = df[['Profit', 'item_price']].corr()
correlation
```

[30]:

	Profit	item_price
Profit	1.000000	0.190976
item_price	0.190976	1.000000

[31]:
```
# Calculate the correlation
correlation = df[['Profit', 'Quantity']].corr()
correlation
```

[31]:

	Profit	Quantity
Profit	1.000000	0.066253
Quantity	0.066253	1.000000

[28]:
```
# Calculate the correlation
correlation = df[['Profit', 'Discount']].corr()
correlation
```

[28]:

	Profit	Discount
Profit	1.000000	-0.219487
Discount	-0.219487	1.000000

[32]:
```
# Calculate the correlation
correlation = df[['Profit', 'Ship_Mode_Rank']].corr()
correlation
```

[32]:

	Profit	Ship_Mode_Rank
Profit	1.000000	-0.005767
Ship_Mode_Rank	-0.005767	1.000000

[34]:
```
# Calculate the correlation
correlation = df[['Profit', 'Time_Frame']].corr()
correlation
```

[34]:

	Profit	Time_Frame
Profit	1.000000	-0.004649
Time_Frame	-0.004649	1.000000

Phase 3 Forecasting Questions Phase

A. What are the forecasts for both Sales and Profit in the coming three years?

Import necessary libraries:

```
import pandas as pd.
from sklearn.linear_model import LinearRegression
import numpy as np.
import matplotlib.pyplot as plt.
```

Prepare the data: mydata2 is your Data Frame:

```
# Rename the 'Order_date _year' column for convenience
mydata2 = mydata2.rename(columns={'Order_date _year': 'Year'})

# Create features (X) and target (y) variables
X = mydata2[['Year']]
y_sales = mydata2['Sales']
y_profit = mydata2['Profit']
```

Create and train the model for Sales:

```
# Initialize the model
model_sales = LinearRegression()
# Fit the model
model_sales.fit(X, y_sales)
```

Create and train the model for Profit:

```
# Initialize the model
model_profit = LinearRegression()
# Fit the model
model_profit.fit(X, y_profit)
```

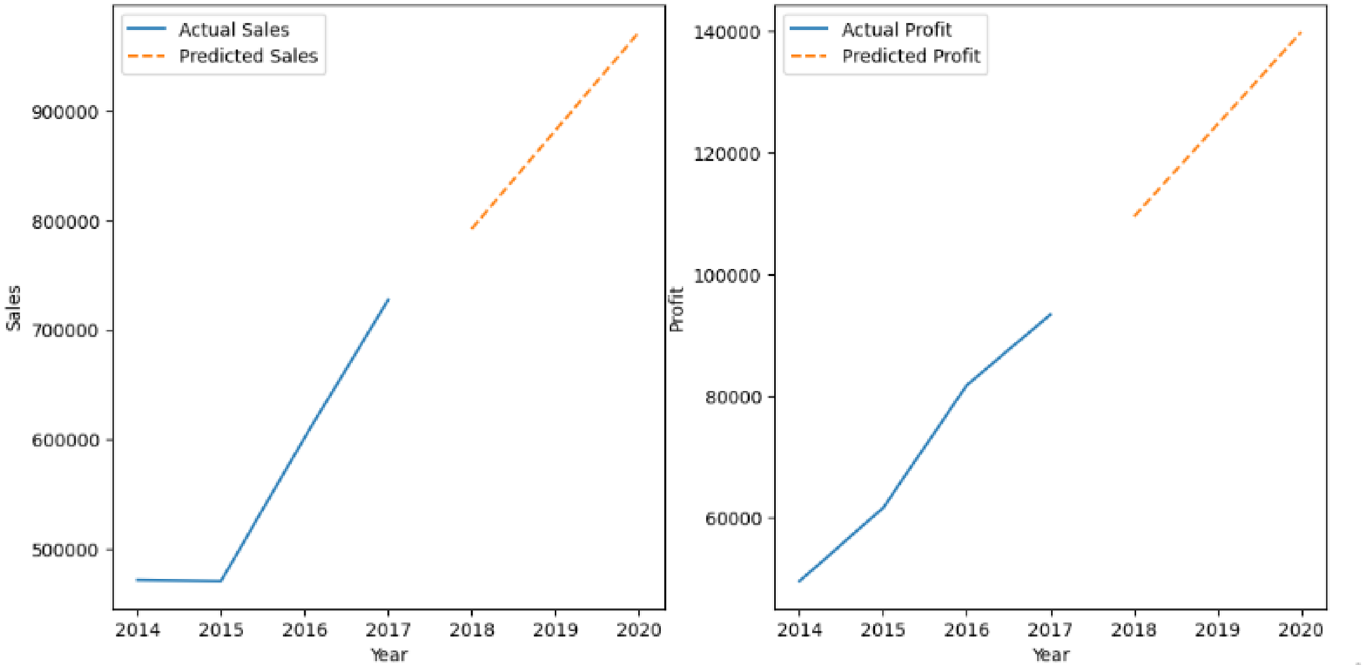
Make predictions for the next three years:

```
# Define the next three years
future_years = np.array([[2018], [2019], [2020]])
# Predict sales
predicted_sales = model_sales.predict(future_years)
# Predict profit
predicted_profit = model_profit.predict(future_years)
```

<u>Display the results:</u>

```
# Create a DataFrame for the forecast
forecast = pd.DataFrame({
    'Year': future_years.flatten(),
    'Predicted_Sales': predicted_sales,
    'Predicted_Profit': predicted_profit.
})

print(forecast)

# Optionally, plot the forecast
plt.figure(figsize=(12, 6))

# Sales plot
plt.subplot(1, 2, 1)
plt.plot(mydata2['Year'], mydata2['Sales'], label='Actual Sales')
plt.plot(forecast['Year'], forecast['Predicted_Sales'], label='Predicted Sales', linestyle='--')
plt.xlabel('Year')
plt.ylabel('Sales')
plt.legend()

# Profit plot
plt.subplot(1, 2, 2)
plt.plot(mydata2['Year'], mydata2['Profit'], label='Actual Profit')
plt.plot(forecast['Year'], forecast['Predicted_Profit'], label='Predicted Profit', linestyle='--')
plt.xlabel('Year')
plt.ylabel('Profit')
plt.legend()

plt.show()
```

	Year	Predicted_Sales	Predicted_Profit
0	2018	792563.67185	109564.86970
1	2019	882452.88741	124751.11541
2	2020	972342.10297	139937.36112



B. What are the forecast for sales & Profit for Regions in the Coming three years?

A. Sales forecast per region

Prepare the data: Ensure your Data Frame is formatted correctly.

```
# Assuming df is your original DataFrame
# Group by 'Order_date _year' and 'Region' and calculate the sum of 'Sales'
sales_per_region_year = df.groupby(['Order_date _year',
'Region'])['Sales'].sum().reset_index()

# Pivot the DataFrame to get years as rows and regions as columns
pivot_table = sales_per_region_year.pivot(index='Order_date _year', columns='Region',
values='Sales').fillna(0)

# Separate the years for features (X) and sales data (y)
X = np.array(pivot_table.index).reshape(-1, 1)
```

Create and train models for each region:

```
models = {}
predictions = {}

# Forecast for the next three years
future_years = np.array([[2018], [2019], [2020]])

# Train a model for each region
for region in pivot_table.columns:
    y = pivot_table[region].values
    model = LinearRegression()
    model.fit(X, y)
    models[region] = model
    predictions[region] = model.predict(future_years)
```

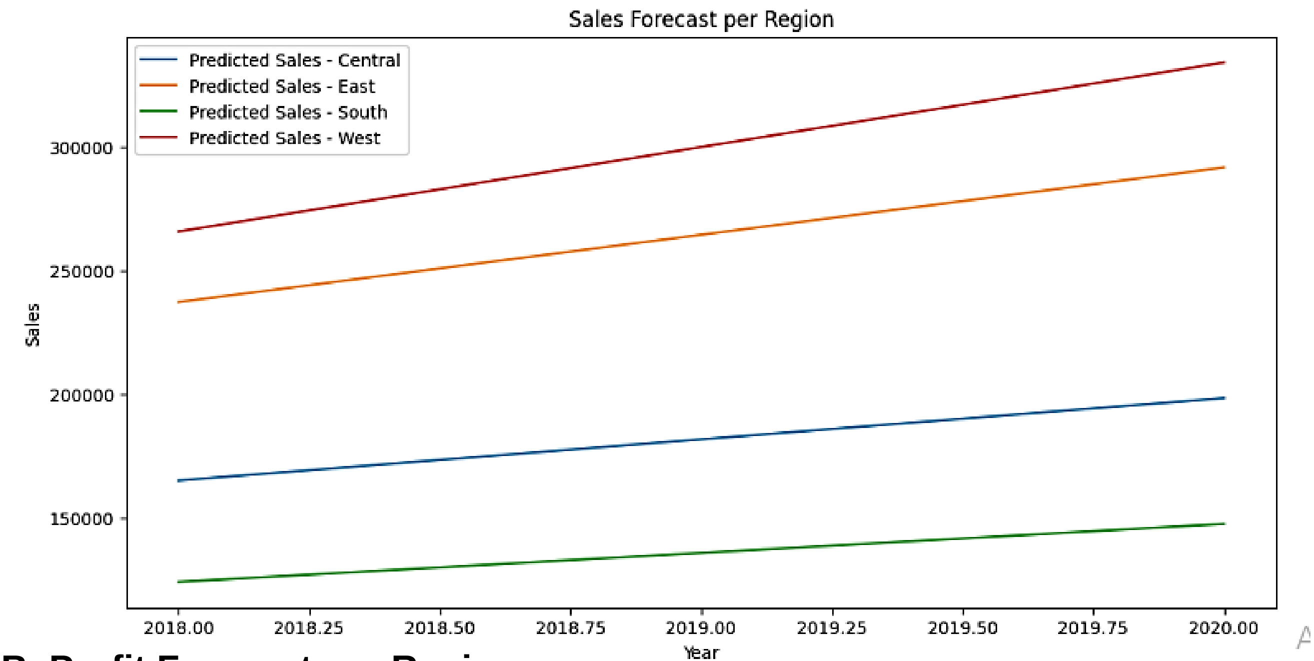
Create a forecast Data Frame:

```
# Create a DataFrame for the forecast
forecast_data = {
    'Year': future_years.flatten()
}
for region, prediction in predictions.items():
    forecast_data[region] = prediction
forecast_df = pd.DataFrame(forecast_data)
```

Display the results:

```
print(forecast_df)
# Optionally, plot the forecast
plt.figure(figsize=(12, 6))
for region in forecast_df.columns[1:]:
    plt.plot(forecast_df['Year'], forecast_df[region], label=f'Predicted Sales - {region}')
plt.xlabel('Year')
plt.ylabel('Sales')
plt.legend()
plt.title('Sales Forecast per Region')
plt.show()
```

	Year	Central	East	South	West
0	2018	165143.75885	237385.64830	124107.28755	265926.97715
1	2019	181827.26832	264631.77744	135841.86008	300151.98157
2	2020	198510.77779	291877.90658	147576.43261	334376.98599



B. <u>Profit Forecast per Regions</u>

Prepare your Data:

```
# Assuming df is your original DataFrame
# Group by 'Order_date _year' and 'Region' and calculate the sum of 'Sales'
profit_per_region_year = df.groupby(['Order_date _year', 'Region'])['Profit'].sum().reset_index()

# Pivot the DataFrame to get years as rows and regions as columns
pivot_table = profit_per_region_year.pivot(index='Order_date _year', columns='Region', values='Profit').fillna(0)

# Separate the years for features (X) and sales data (y)
X = np.array(pivot_table.index).reshape(-1, 1)
```

Create and train models for each region:

```
models = {}
predictions = {}

# Forecast for the next three years
future_years = np.array([[2018], [2019], [2020]])

# Train a model for each region
for region in pivot_table.columns:
    y = pivot_table[region].values
    model = LinearRegression()
    model.fit(X, y)
    models[region] = model
    predictions[region] = model.predict(future_years)
```

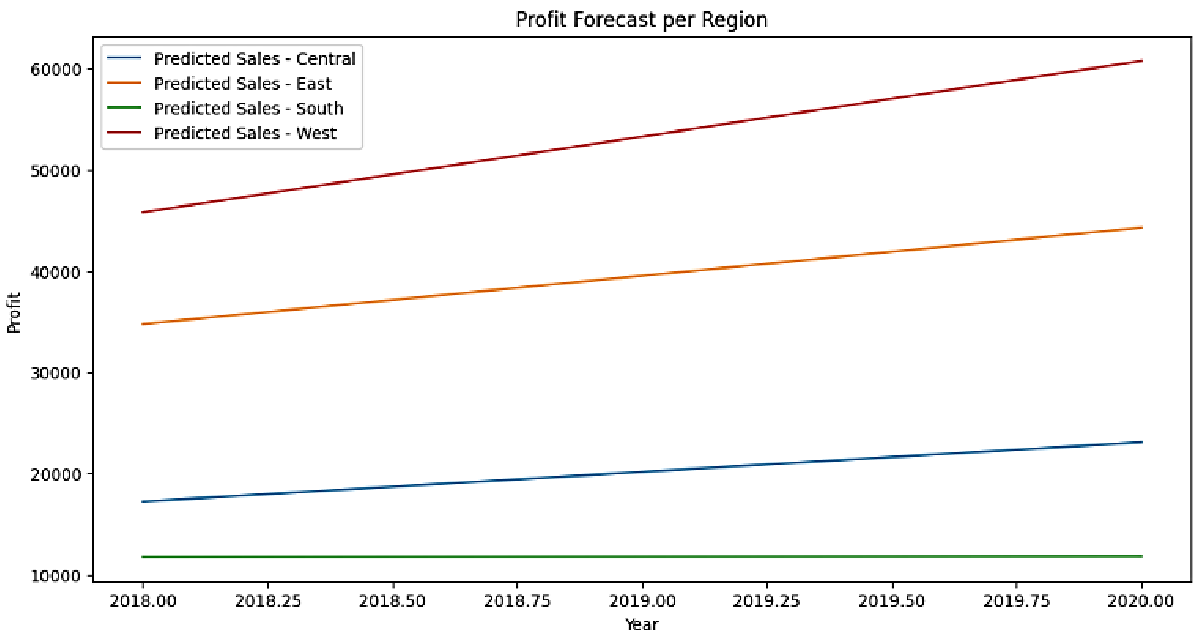
Create a forecast Data Frame:

```
# Create a DataFrame for the forecast
forecast_data = {
    'Year': future_years.flatten()
}

for region, prediction in predictions.items():
    forecast_data[region] = prediction

forecast_df = pd.DataFrame(forecast_data)
```

Display the results:

```
print(forecast_df)
# Optionally, plot the forecast
plt.figure(figsize=(12, 6))
for region in forecast_df.columns[1:]:
    plt.plot(forecast_df['Year'], forecast_df[region], label=f'Predicted Sales - {region}')
plt.xlabel('Year')
plt.ylabel('Profit')
plt.legend()
plt.title('Profit Forecast per Region')
plt.show()
```

	Year	Central	East	South	West
0	2018	17230.64895	34771.55470	11760.75210	45801.91395
1	2019	20152.27228	39527.89858	11790.10991	53280.83464
2	2020	23073.89561	44284.24246	11819.46772	60759.75533



C. What are the forecast for sales & Profit for States in the Coming three years?

Sales per Top 10 state

Prepare data frame:

```
# assuming df is your DataFrame and it has columns 'Order_date _year', 'Region', 'Profit', and 'Sales'
# Group by 'Order_date _year' and 'Region' and calculate the sum of 'Profit'
sales_per_state_year = df.groupby(['Order_date _year', 'State'])['Sales'].sum().reset_index()

# Sort the DataFrame by 'Profit' in descending order
sales_per_state_year = sales_per_state_year.sort_values(by='Sales', ascending=False).head(10)

# Display the result
sales_per_state_year
```

Order_date _year	State	Sales
2017	California	146388.3445
2016	California	131551.9115
2017	New York	92723.0269
2014	California	91303.5310
2015	California	88443.8445
2015	New York	80320.6870
2016	New York	71844.1020
2014	New York	64788.4870
2017	Washington	61539.9359
2014	Texas	50625.1766

- **Ensure your Data Frame only includes the top 10 states**:

```
# Assuming df is your original DataFrame and it has columns 'Order_date _year', 'State', and 'Sales'

# Group by 'Order_date _year' and 'State' and calculate the sum of 'Sales'

sales_per_state_year = df.groupby(['Order_date _year', 'State'])['Sales'].sum().reset_index()


# Sort and get the top 10 states by sales
```

```
top_states = sales_per_state_year.groupby('State')['Sales'].sum().nlargest(10).index


# Filter the DataFrame for these top states
filtered_sales = sales_per_state_year[sales_per_state_year['State'].isin(top_states)]
```

- **Forecast sales for the next three years for these top states:**

```
# Pivot the DataFrame to get years as rows and states as columns
pivot_table = filtered_sales.pivot(index='Order_date _year', columns='State', values='Sales').fillna(0)


# Separate the years for features (X) and sales data (y)
X = np.array(pivot_table.index).reshape(-1, 1)
```

- **Create and train models for each top state:**

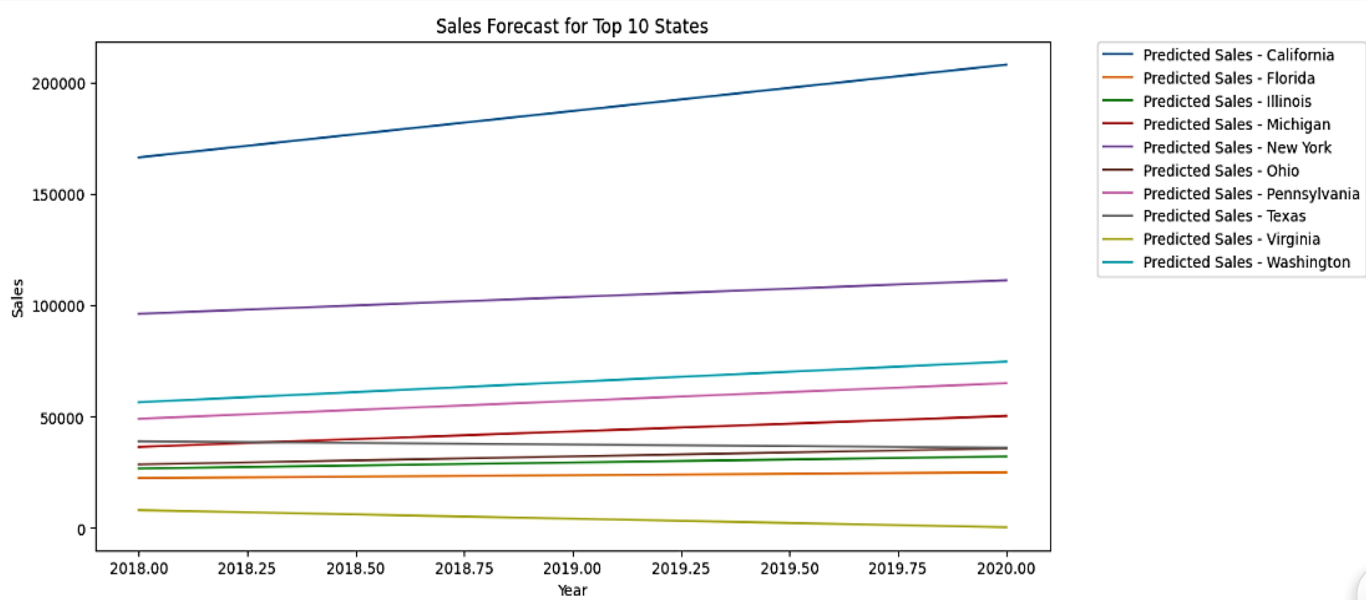
```
models = {}
predictions = {}

# Forecast for the next three years
future_years = np.array([[2018], [2019], [2020]])

# Train a model for each state
for state in pivot_table.columns:
    y = pivot_table[state].values
    model = LinearRegression()
    model.fit(X, y)
    models[state] = model
    predictions[state] = model.predict(future_years)
```

Create a forecast Data Frame:

```
# Create a DataFrame for the forecast
forecast_data = {
    'Year': future_years.flatten()
}


for state, prediction in predictions.items():
    forecast_data[state] = prediction


forecast_df = pd.DataFrame(forecast_data)
```

Visualize the forecast for the top 10 states:

```
plt.figure(figsize=(12, 6))
for state in forecast_df.columns[1:]:
    plt.plot(forecast_df['Year'], forecast_df[state], label=f'Predicted Sales - {state}')

plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Sales Forecast for Top 10 States')
# Place the legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)
plt.show()
```



Profit per Top 10 state

profit_per_state_year = df.groupby(['Order_date _year', 'State'])['Profit'].sum().reset_index()

# Sort the DataFrame by 'Profit' in descending order
profit_per_state_year = profit_per_state_year.sort_values(by='Profit', ascending=False).head(10)

Display the result
profit_per_state_year.

Order_date _year	State	Profit
2017	California	29366.4589
2017	New York	24357.0717
2016	California	20005.7161
2015	New York	19277.5826
2017	Washington	17256.7798
2016	New York	16654.9495
2015	California	14371.2630
2014	New York	13748.9448
2014	California	12637.9491
2016	Indiana	10385.1339

Prepare the data: Make sure your Data Frame is set up correctly.

Assuming df is your original DataFrame and it has columns 'Order_date _year', 'State', and 'Profit'

Group by 'Order_date _year' and 'State' and calculate the sum of 'Profit'

profit_per_state_year = df.groupby(['Order_date _year', 'State'])['Profit'].sum().reset_index()

Sort and get the top 10 states by profit

top_states = profit_per_state_year.groupby('State')['Profit'].sum().nlargest(10).index

Filter the DataFrame for these top states

filtered_profit = profit_per_state_year[profit_per_state_year['State'].isin(top_states)]

Forecast profit for the next three years for these top states:

```
# Pivot the DataFrame to get years as rows and states as columns

pivot_table = filtered_profit.pivot(index='Order_date _year', columns='State', values='Profit').fillna(0)

# Separate the years for features (X) and profit data (y)

X = np.array(pivot_table.index).reshape(-1, 1)
```

Create and train models for each top state:

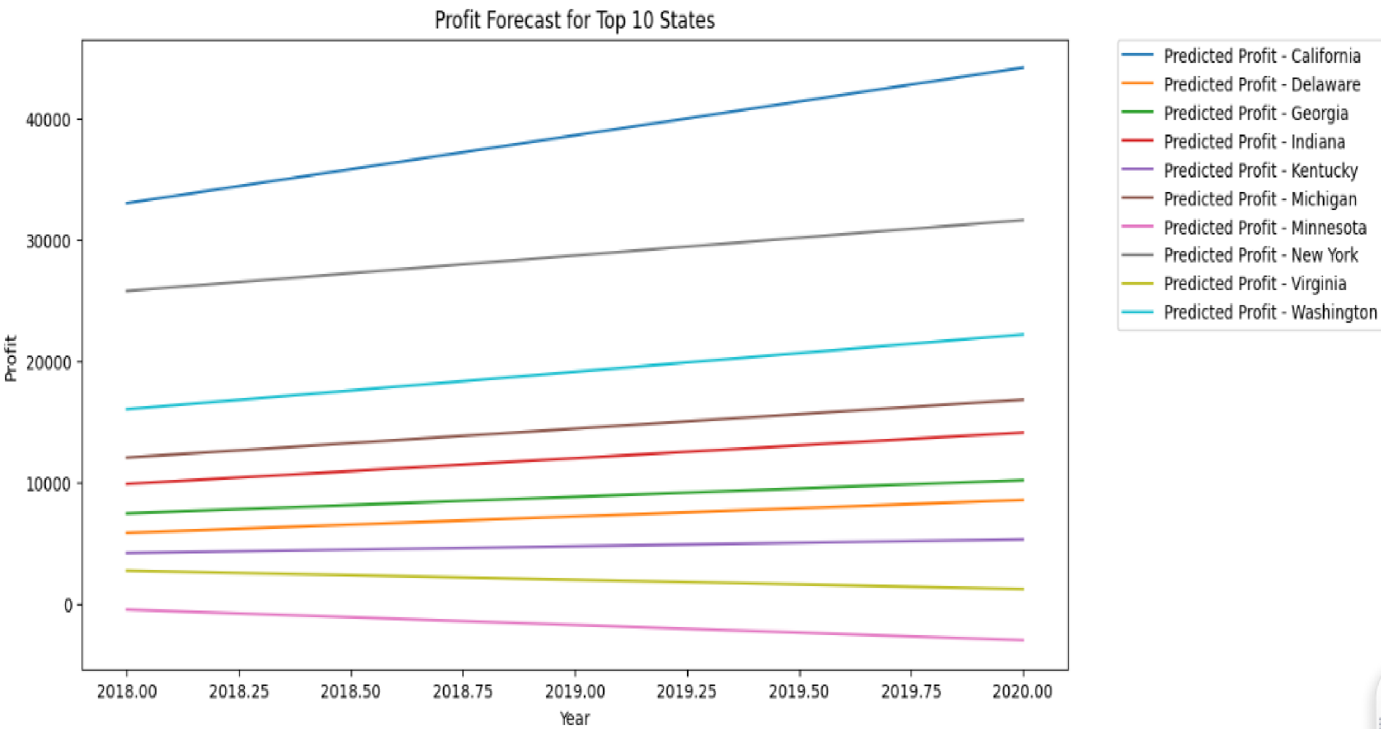
```
models = {}

predictions = {}

# Forecast for the next three years

future_years = np.array([[2018], [2019], [2020]])

# Train a model for each state

for state in pivot_table.columns:

    y = pivot_table[state].values

    model = LinearRegression()

    model.fit(X, y)

    models[state] = model

    predictions[state] = model.predict(future_years)
```

Create a forecast DataFrame:

```
# Create a DataFrame for the forecast
forecast_data = {
    'Year': future_years.flatten()
}

for state, prediction in predictions.items():
    forecast_data[state] = prediction

forecast_df = pd.DataFrame(forecast_data)
```

Visualize the forecast for the top 10 states:

```
plt.figure(figsize=(12, 6))

for state in forecast_df.columns[1:]:
    plt.plot(forecast_df['Year'], forecast_df[state], label=f'Predicted Profit - {state}')

plt.xlabel('Year')
plt.ylabel('Profit')
plt.title('Profit Forecast for Top 10 States')

# Place the legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

plt.show()
```



D. What are the forecast for sales for Categories in the Coming three years?

Prepare the data: Ensure your Data Frame is formatted correctly.

```
# Assuming df is your original DataFrame and it has columns 'Order_date _year', 'Category', and 'Sales'
# Group by 'Order_date _year' and 'Category' and calculate the sum of 'Sales'
sales_per_cat_year = df.groupby(['Order_date _year', 'Category'])['Sales'].sum().reset_index()

# Pivot the DataFrame to get years as rows and categories as columns
pivot_table = sales_per_cat_year.pivot(index='Order_date _year', columns='Category', values='Sales').fillna(0)

# Separate the years for features (X) and sales data (y)
X = np.array(pivot_table.index).reshape(-1, 1)
```

Create and train models for each category:

```
models = {}
predictions = {}

# Forecast for the next three years
future_years = np.array([[2018], [2019], [2020]])

# Train a model for each category
for category in pivot_table.columns:
    y = pivot_table[category].values
    model = LinearRegression()
    model.fit(X, y)
    models[category] = model
    predictions[category] = model.predict(future_years)
```

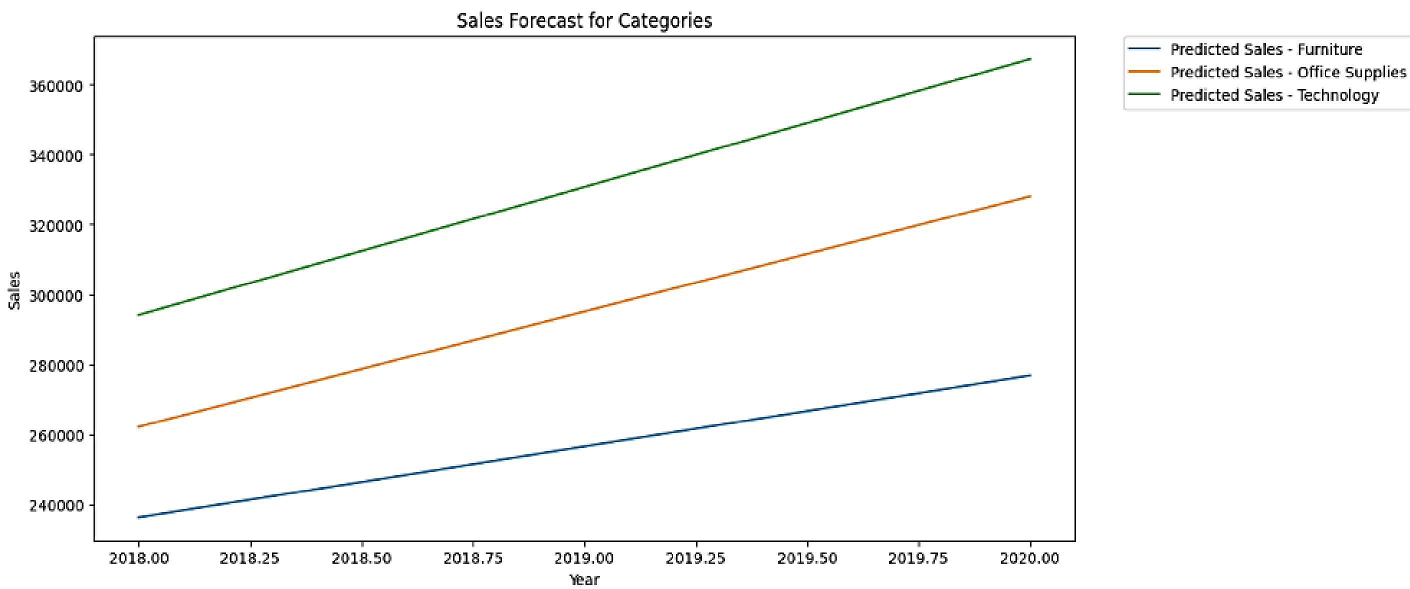
Create a forecast DataFrame:

```
# Create a DataFrame for the forecast
forecast_data = {
    'Year': future_years.flatten()
}

for category, prediction in predictions.items():
    forecast_data[category] = prediction

forecast_df = pd.DataFrame(forecast_data)
```

Visualize the forecast for the categories:

```
Print(forecast_df)
plt.figure(figsize=(12, 6))

for category in forecast_df.columns[1:]:
    plt.plot(forecast_df['Year'], forecast_df[category], label=f'Predicted Sales - {category}')

plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Sales Forecast for Categories')

# Place the legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

plt.show()
```

	Year	Furniture	Office Supplies	Technology
0	2018	236241.56065	262178.9600	294143.15120
1	2019	256538.20538	295145.8408	330768.84123
2	2020	276834.85011	328112.7216	367394.53126



E. What are the forecast for sales for Sub_Categories in the Coming three years?

<u>Prepare the data: Ensure your Data Frame is formatted correctly.</u>

```
# Assuming df is your DataFrame and it has columns 'Order_date _year', 'sub-Category', 'Profit', and 'Sales'
# Group by 'Order_date _year' and 'sub-Category' and calculate the sum of 'Sales'
sales_per_subcat_year = df.groupby(['Order_date _year', 'Sub_Category'])['Sales'].sum().reset_index()

# Sort the DataFrame by 'Sales' in descending order
sales_per_subcat_year = sales_per_subcat_year.sort_values(by='Sales', ascending=False)

# Display the result
sales_per_subcat_year
```

Order_date _year	Sub_Category	Sales
2017	Phones	105340.516
2017	Chairs	95554.353
2016	Chairs	83918.645
2016	Phones	78962.030
2014	Phones	77390.806
...	...	...
2015	Supplies	1952.482
2016	Fasteners	960.134
2017	Fasteners	857.594
2014	Fasteners	661.328
2015	Fasteners	545.224

Prepare the data: Ensure your Data Frame is formatted correctly.

```
# Assuming df is your original DataFrame and it has columns 'Order_date _year', 'Sub_Category', and 'Sales'
# Group by 'Order_date _year' and 'Sub_Category' and calculate the sum of 'Sales'
sales_per_subcat_year = df.groupby(['Order_date _year', 'Sub_Category'])['Sales'].sum().reset_index()

# Pivot the DataFrame to get years as rows and sub-categories as columns
pivot_table = sales_per_subcat_year.pivot(index='Order_date _year', columns='Sub_Category', values='Sales').fillna(0)


# Separate the years for features (X) and sales data (y)
X = np.array(pivot_table.index).reshape(-1, 1)
```

Create and train models for each sub-category:

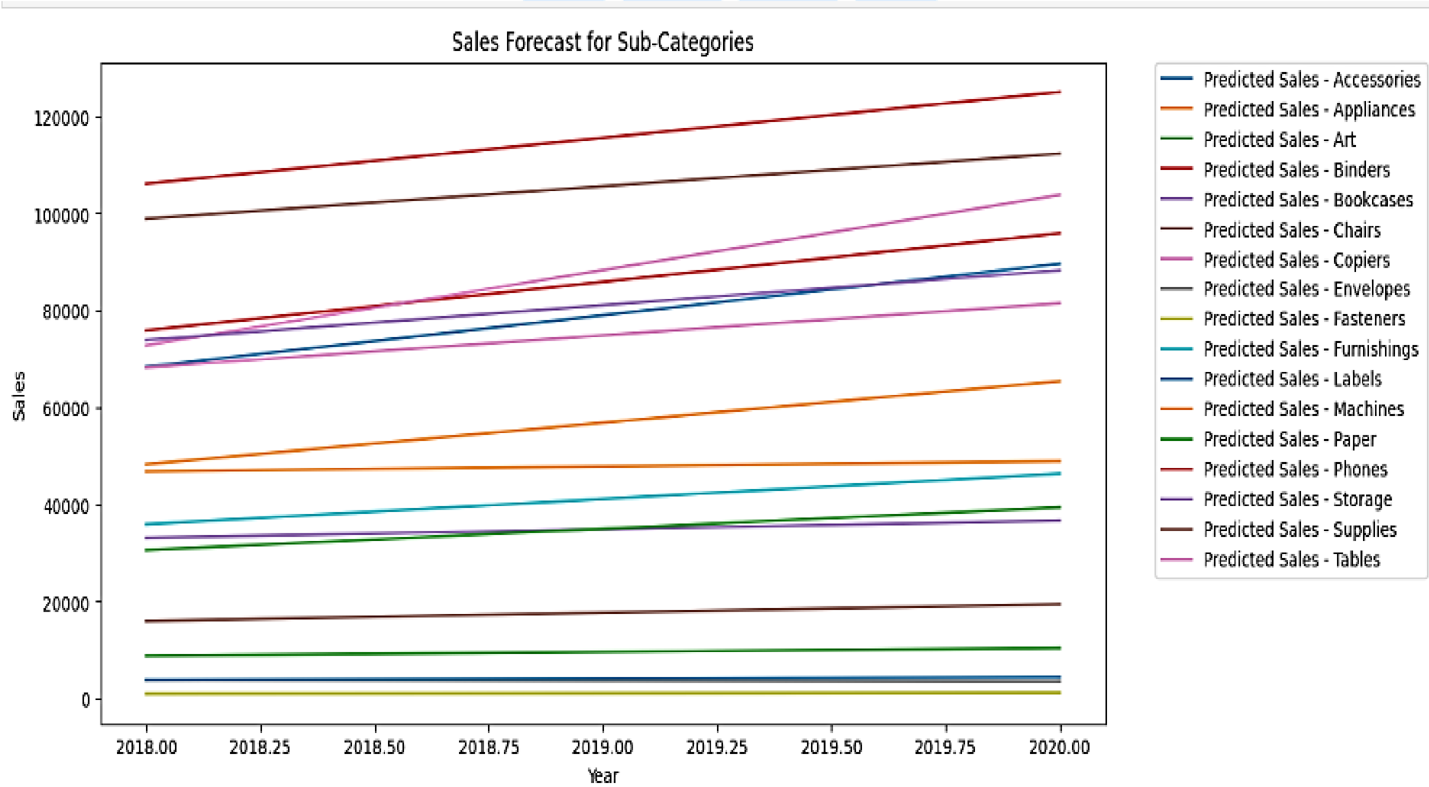
```
models = {}
predictions = {}

# Forecast for the next three years
future_years = np.array([[2018], [2019], [2020]])

# Train a model for each sub-category
for sub_category in pivot_table.columns:
    y = pivot_table[sub_category].values
    model = LinearRegression()
    model.fit(X, y)
    models[sub_category] = model
    predictions[sub_category] = model.predict(future_years)
```

Create a forecast Data Frame:

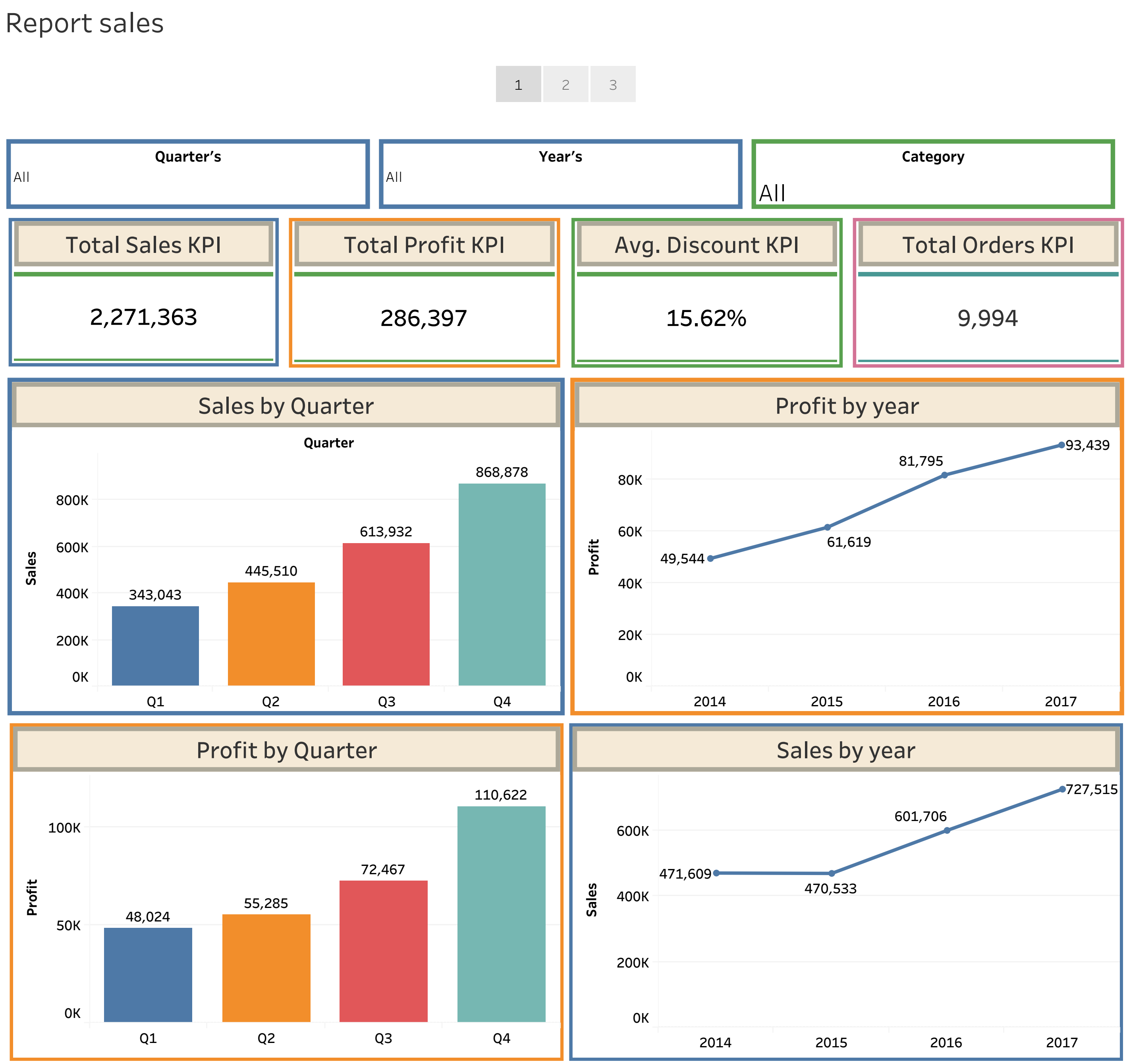
```
# Create a DataFrame for the forecast
forecast_data = {
    'Year': future_years.flatten()
}

for sub_category, prediction in predictions.items():
    forecast_data[sub_category] = prediction

forecast_df = pd.DataFrame(forecast_data)
```

Visualize the forecast for the sub-categories:

```
plt.figure(figsize=(12, 6))

for sub_category in forecast_df.columns[1:]:
   plt.plot(forecast_df['Year'], forecast_df[sub_category], label=f'Predicted Sales - {sub_category}')

plt.xlabel('Year')
plt.ylabel('Sales')
plt.title('Sales Forecast for Sub-Categories')

# Place the legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

plt.show()
```

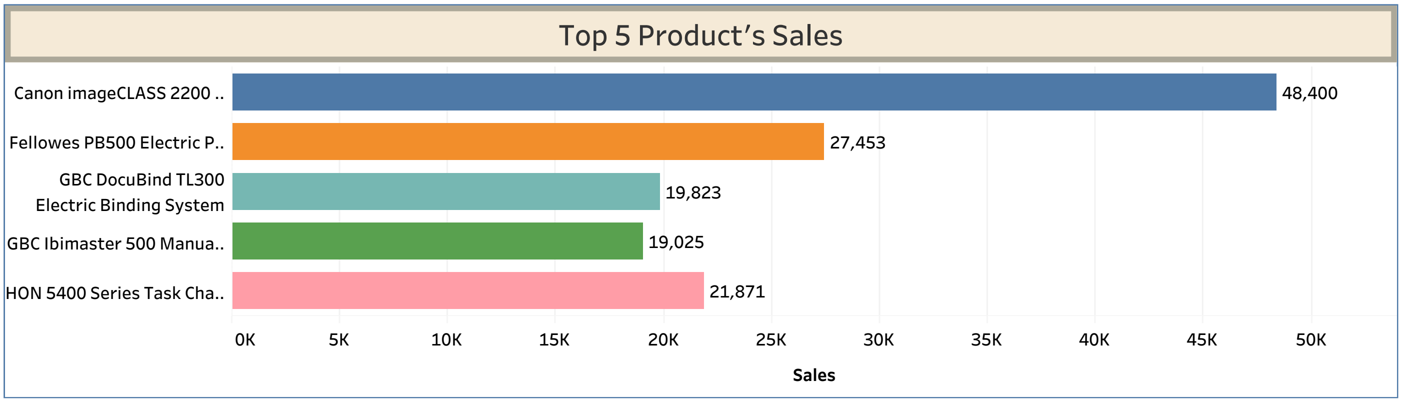


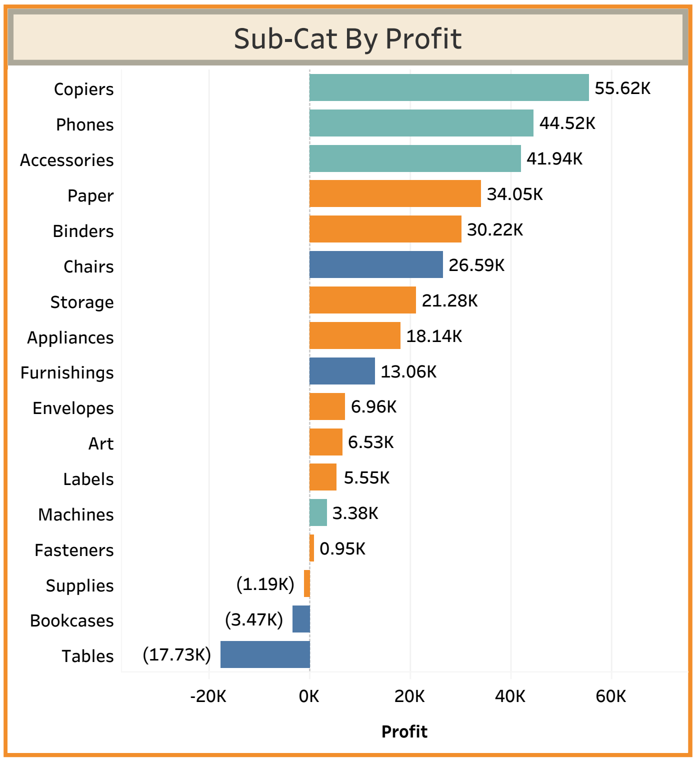
4.Phase 4 Data Visualizations

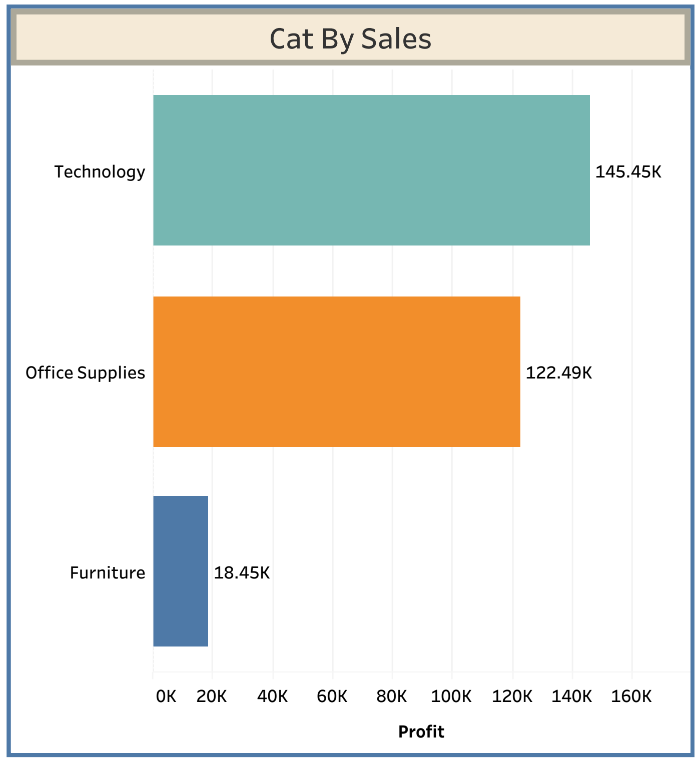


Report sales

1 2 3



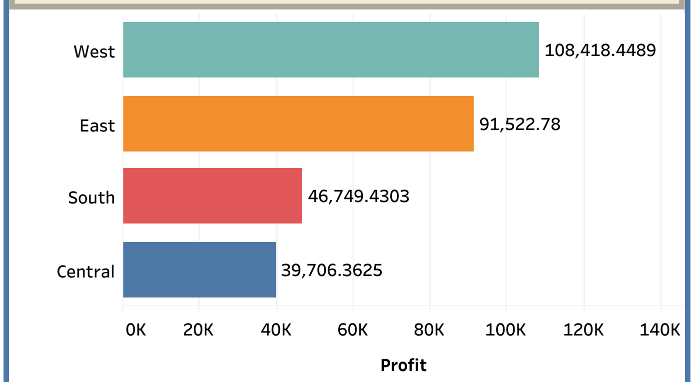




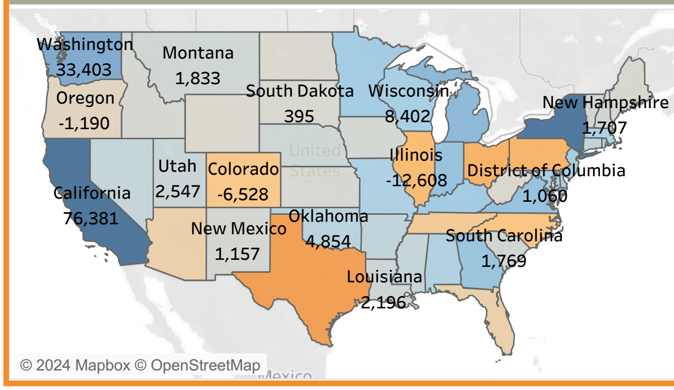
Report sales

1 2 3

Region by Profit



States by Profit



City by Profit