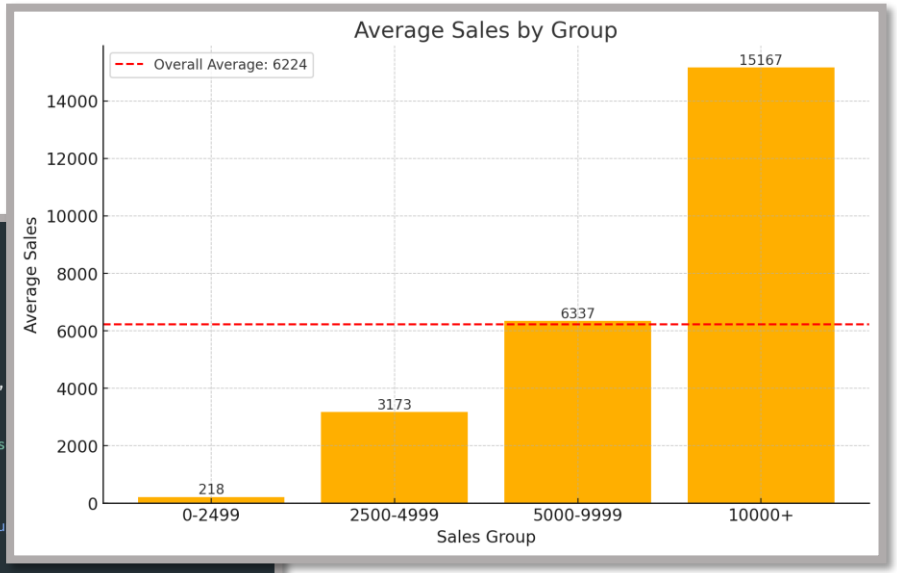
CHAT GPT + PYTHON NO-CODE

	A	B	C	D	E
1	Order Date	Ship Date	Customer Name	City	State
2	11 November 2021	13 November 2021	Aaron Bergman	Oklahoma City	Oklahoma
3	05 February 2021	07 February 2021	Justin Ritter	Wollongong	New South Wales
4	17 October 2021	18 October 2021	Craig Reiter	Brisbane	Queensland
5	28 January 2021	30 January 2021	Katherine Murray	Berlin	Berlin
6	05 November 2021	06 November 2021	Rick Hansen	Dakar	Dakar
7	28 June 2021	01 July 2021	Jim Mitchum	Sydney	New South Wales
8	06 November 2019	08 November 2019	Toby Swindell	Porirua	Wellington
9	14 April 2020	18 April 2020	Mick Brown	Hamilton	Waikato
10	11 November 2021	13 November 2021	Aaron Bergman	Oklahoma City	Oklahoma
11	06 March 2019	07 March 2019	Aaron Bergman	Seattle	Washington
12	06 March 2019	07 March 2019	Aaron Bergman	Seattle	Washington
13	19 April 2020	22 April 2020	Anthony Jacobs	Kabul	Kabul
14	26 December 2019	28 December 2019	Magdelene Morse	Jizan	Jizan
15	13 November 2020	13 November 2020	Vicky Freymann	Toledo	Parana

```
import matplotlib.pyplot as plt

# Create a new column to group 'Sales' into the specified ranges
bins = [0, 2499, 4999, 9999, float('inf')]
labels = ['0-2499', '2500-4999', '5000-9999', '10000+']
data['Sales Group'] = pd.cut(data['Sales'], bins=bins, labels=labels,

# Calculate the average sales for each group
avg_sales_per_group = data.groupby('Sales Group')['Sales'].mean().res

# Plotting the column chart
plt.figure(figsize=(10, 6))
bars = plt.bar(avg_sales_per_group['Sales Group'], avg_sales_per_grou

# Adding data labels as whole numbers
for bar in bars:
    yval = bar.get_height()
lt.text(bar.get_x() + bar.get_width()/2, yval, int(yval), ha='center', va='bottom')

culate the overall average of the four groups
ll_avg = avg_sales_per_group['Sales'].mean()

ing a horizontal reference line
xhline(y=overall_avg, color='red', linestyle='--', label=f'Overall Average: {int(overall_avg)}')

ing labels and title
label('Sales Group')
label('Average Sales')
itle('Average Sales by Group')
egend()

play the chart
how()
```

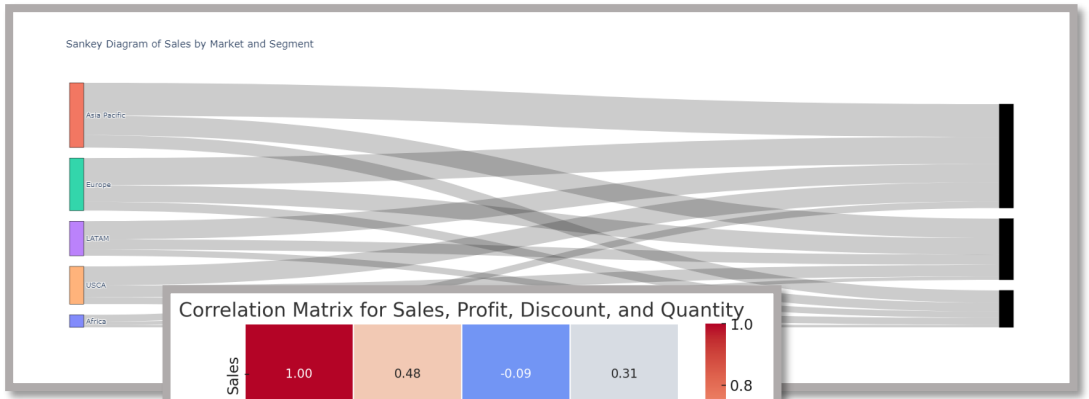


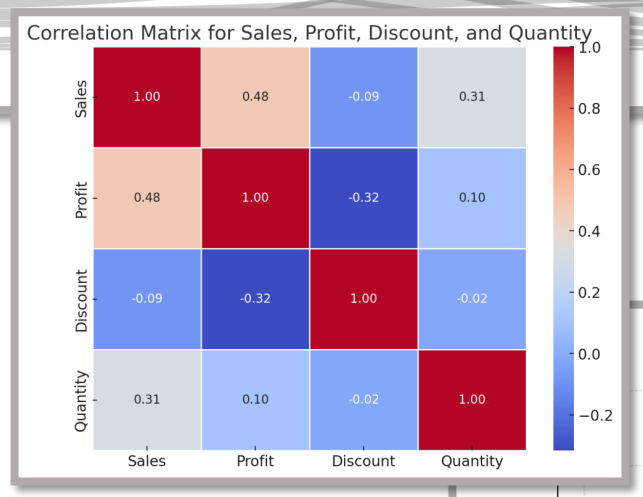
Global Superstore Orders.xlsx
Spreadsheet

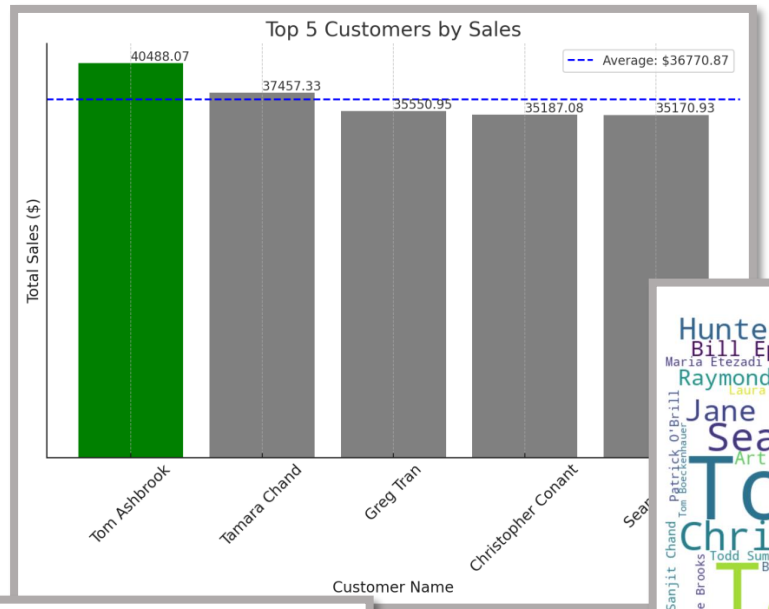
Group Sales into 0-2499, 2500-4999, 5000-9999, 10000+
Then create a column chart to show average Sales
Show data labels as a whole number.
Create a horizontal reference line to show Average.

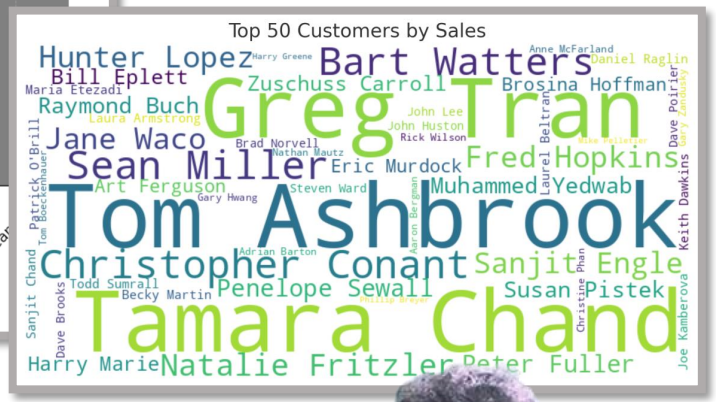
HAVISH M
CONSULTING

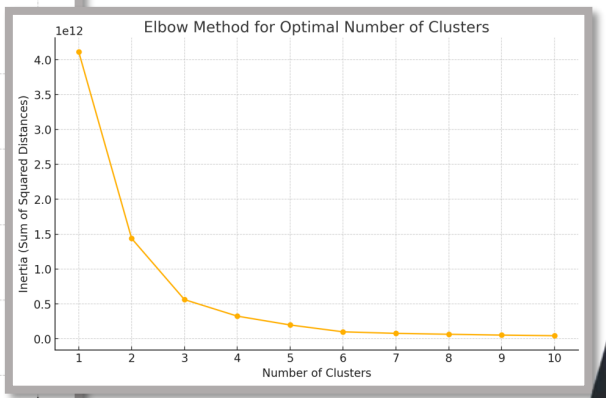
CHAT GPT + PYTHON NO-CODE





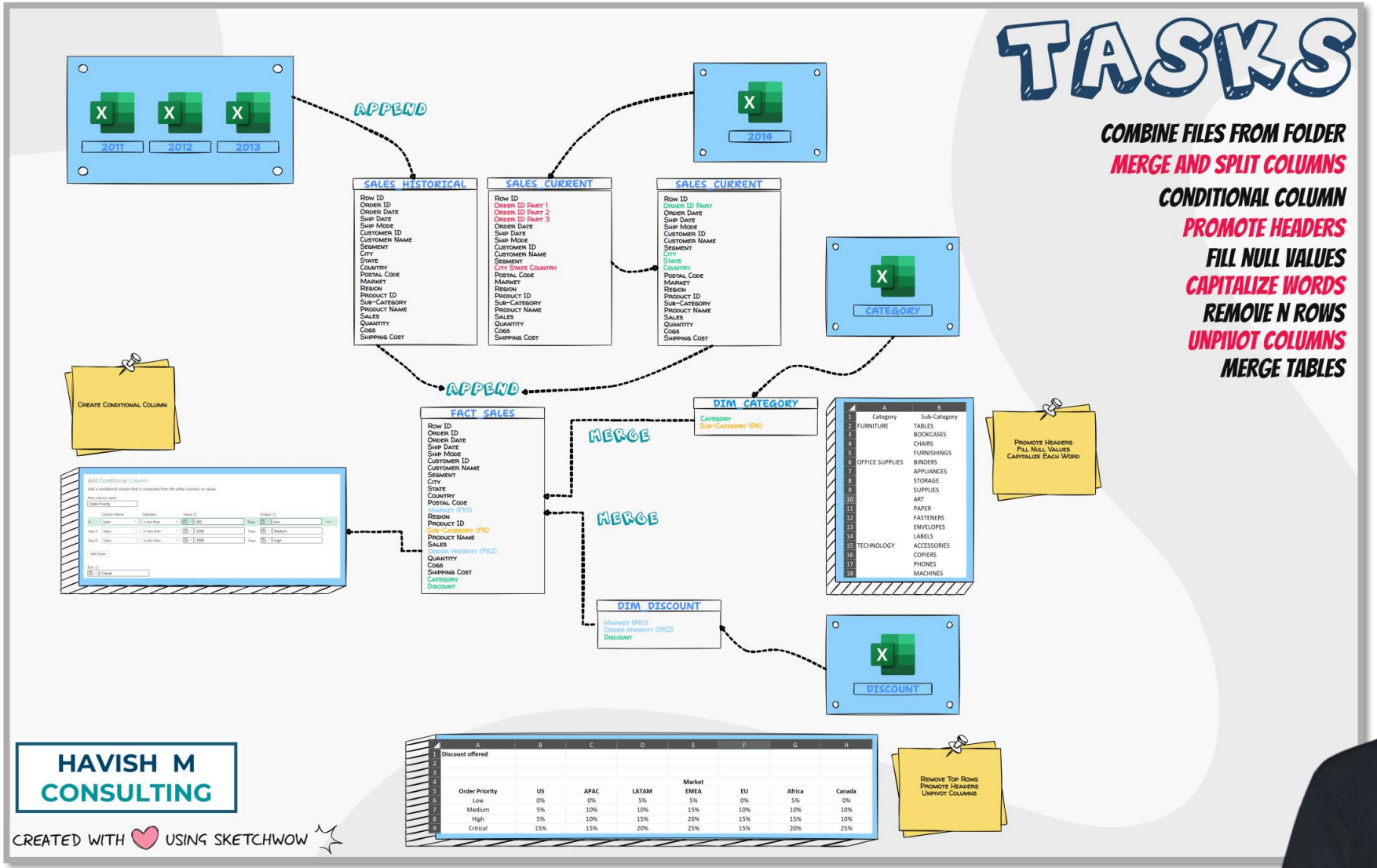






HAVISH M
CONSULTING

CHAT GPT + PYTHON NO-CODE



HAVISH M
CONSULTING

CHAT GPT + PYTHON NO-CODE

Combine and Clean Files from Folder

File Edit View Insert Runtime Tools Help

Comment Share

Files

..
.config
sample_data
2011.xlsx
2012.xlsx
2013.xlsx
2014.xlsx
Category.xlsx
Discount.xlsx
Fact_Sales_With_Discounts.csv

+ Code + Text

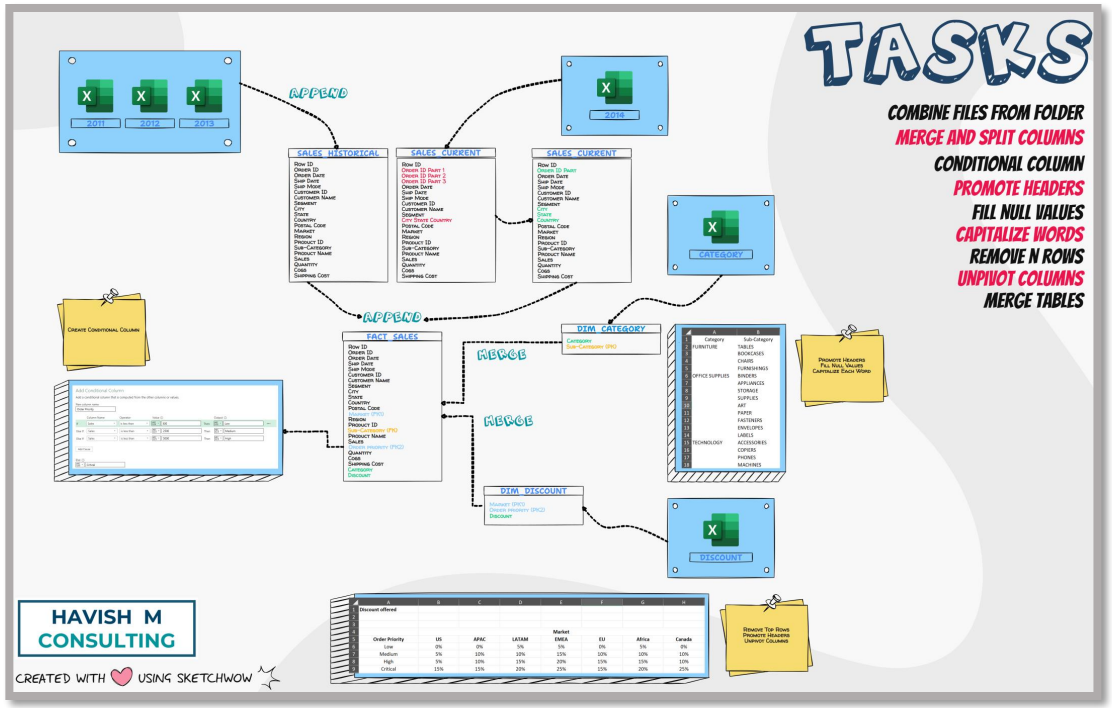
RAM Disk Gemini

18s

```python
# Append current and historical sales data
fact_sales = pd.concat([sales_historical, sales_current], ignore_index=True)

# Add Order Priority column based on sales amount
def assign_priority(sales):
    if sales < 1000:
        return "Low"
    elif sales < 2500:
        return "Medium"
    elif sales < 5000:
        return "High"
    else:
        return "Critical"
fact_sales['Order Priority'] = fact_sales['Sales'].apply(assign_priority)

# Load category data and adjust the format
dim_category = pd.read_excel('Category.xlsx')
dim_category['Category'] = dim_category['Category'].ffill()
dim_category['Category'] = dim_category['Category'].str.title()
dim_category['Sub-Category'] = dim_category['Sub-Category'].str.title()

# Merge category data with sales data
merged_data = pd.merge(fact_sales, dim_category, on='Sub-Category', how='left')

# Load and process discount data
dim_discount = pd.read_excel('Discount.xlsx', skiprows=3)
dim_discount.columns = dim_discount.iloc[0]
dim_discount = dim_discount.drop(dim_discount.index[0])
dim_discount_melted = pd.melt(dim_discount, id_vars=['Order Priority'], var_name='Market', value_name='Discount')

# Merge discount data with sales data
final_merged_data = pd.merge(merged_data, dim_discount_melted, on=['Market', 'Order Priority'], how='left')

# Save the final merged DataFrame to a CSV file
final_merged_data.to_csv('Fact_Sales_With_Discounts.csv', index=False)
```

HAVISH M CONSULTING

CHAT GPT + PYTHON NO-CODE




```python
import pandas as pd

# Load historical sales data from Excel files
df_2011 = pd.read_excel('/mnt/data/2011.xlsx')
df_2012 = pd.read_excel('/mnt/data/2012.xlsx')
df_2013 = pd.read_excel('/mnt/data/2013.xlsx')
sales_current = pd.read_excel('/mnt/data/2014.xlsx')

# Append historical sales data into a single DataFrame
sales_historical = pd.concat([df_2011, df_2012, df_2013], ignore_index=True)

# Combine parts of Order ID in current sales data
sales_current['Order ID'] = sales_current['Order ID Part 1'].astype(str) + '-' + sales_current['Order ID Part
2'].astype(str) + '-' + sales_current['Order ID Part 3'].astype(str)
sales_current.drop(['Order ID Part 1', 'Order ID Part 2', 'Order ID Part 3'], axis=1, inplace=True)

# Split 'City State Country' into separate columns
sales_current[['City', 'State', 'Country']] = sales_current['City State Country'].str.split(', ', expand=True)
sales_current.drop('City State Country', axis=1, inplace=True)

# Append current and historical sales data
fact_sales = pd.concat([sales_historical, sales_current], ignore_index=True)

# Add Order Priority column based on sales amount
def assign_priority(sales):
    if sales < 1000:
        return "Low"
    elif sales < 2500:
        return "Medium"
    elif sales < 5000:
        return "High"
    else:
        return "Critical"
fact_sales['Order Priority'] = fact_sales['Sales'].apply(assign_priority)

# Load category data and adjust the format
dim_category = pd.read_excel('/mnt/data/Category.xlsx')
dim_category['Category'] = dim_category['Category'].ffill()
dim_category['Category'] = dim_category['Category'].str.title()
dim_category['Sub-Category'] = dim_category['Sub-Category'].str.title()

# Merge category data with sales data
merged_data = pd.merge(fact_sales, dim_category, on='Sub-Category', how='left')

# Load and process discount data
dim_discount = pd.read_excel('/mnt/data/Discount.xlsx', skiprows=3)
dim_discount.columns = dim_discount.iloc[0]
dim_discount = dim_discount.drop(dim_discount.index[0])
dim_discount_melted = pd.melt(dim_discount, id_vars=['Order Priority'], var_name='Market', value_name='Discount')

# Merge discount data with sales data
final_merged_data = pd.merge(merged_data, dim_discount_melted, on=['Market', 'Order Priority'], how='left')

# Save the final merged DataFrame to a CSV file
final_merged_data.to_csv('/mnt/data/Fact_Sales_With_Discounts.csv', index=False)
```

HAVISH M
CONSULTING

Trainer Profile

15+ years of experience across Marketing and Sales in CPG, Academics, Research, Consulting and Analytics.

1300+ Workshops across 75+ clients.

Senior Leadership Workshops at Big 4 and Fortune 500.

Member of NASSCOM, CII Young Indians and the Microsoft Partner Network.

Listed in 40u40 Analytics, Indian Achievers Award and Top 10 Trending Speakers by SpeakIn.

Faculty at IMT Ghaziabad, SRCC, Masters Union.

YouTube channel with 1300+ videos, 2+ lakh views and 6000 subscribers.

15+ Books on Analytics – used by Delhi University and IMT Ghaziabad,

Technologies: Excel (VBA, Power Query, Power Pivot), Power BI and Tableau, Python, PowerPoint.

WORKSHOP FLOW

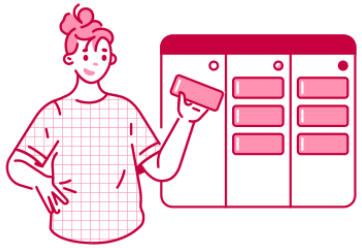
HAVISH M CONSULTING

Enhance Your Experience

Ask a question on chat window

Save Broader Questions for end of workshop

Watch Again



Download attachments

HAVISH M CONSULTING

.

DATA VISUALIZATION	Top 5 Customers Correlation Matrix Cluster Analysis Sankey Wordcloud RFM Analysis
ETL	Show Columns Describe Columns Description of each Column Combine Files Combine Columns Split Columns Append Files Create Conditional Column Clean Files – Fill Null Values and Change Case Merge on 1 column / Merge on 2 Columns Remove Top Rows Unpivot Data Export Data

Show Columns

Show Columns

Show Columns

```python
import pandas as pd

# Load the Excel file to see the columns
file_path = '/mnt/data/Global Superstore Orders.xlsx'
data = pd.read_excel(file_path)
data.columns.tolist()
```

The Excel file contains the following columns:

1. Order Date
2. Ship Date
3. Customer Name
4. City
5. State
6. Country
7. Region
8. Market
9. Category
10. Segment
11. Sub-Category
12. Sales
13. Quantity
14. Discount
15. Profit

Describe Columns

Describe Columns

Describe Columns

```
# Describe the columns to give a statistical summary and data types
data.describe(include='all').T
```

Here's a detailed description of the columns in the "Global Superstore Orders" dataset:

1. **Order Date** and **Ship Date**: Represent dates ranging from January 1, 2019, to December 31, 2022, for orders, and up to January 7, 2023, for shipments.
2. **Customer Name**: The dataset includes 796 unique customer names, with 'Muhammed Yedwab' appearing most frequently (108 times).
3. **City**, **State**, and **Country**: Represent geographical details with 'New York City', 'California', and 'United States' appearing most frequently among 3650 cities, 1102 states, and 165 countries.
4. **Region** and **Market**: There are 23 regions and 5 markets with 'Western Europe' and 'Asia Pacific' being the most frequent respectively.
5. **Category** and **Sub-Category**: Includes 3 main categories ('Office Supplies' being the most common) and 17 sub-categories ('Binders' being the most frequent).
6. **Segment**: Includes 3 segments, with 'Consumer' as the most frequent.
7. **Sales**, **Quantity**, **Discount**, and **Profit**: These are numerical columns with the following characteristics:
   - **Sales**: Ranges from $0.444 to $22,638.48 with an average of $246.49.
   - **Quantity**: Ranges from 1 to 14 units per order with an average of approximately 3.5 units.
   - **Discount**: Ranges from 0 to 85%, with many transactions having no discount.
   - **Profit**: Varies widely from a loss of $6599.978 to a gain of $8399.976 with an average profit of $28.61 per transaction.

These columns provide comprehensive transactional data, useful for analysis in sales performance, customer behavior, and regional business trends. [›-]

Description of each column

Create a Table

Description of each column

Possible Analysis

Description of each column

Here's a table summarizing the description of each column in the dataset along with potential analyses that could be conducted using the data:

Column	Description	Possible Analysis
Order Date	Date when the order was placed	Trend analysis over time, seasonality in orders
Ship Date	Date when the order was shipped	Shipping time analysis, delays in shipping
Customer Name	Names of customers placing orders	Customer segmentation, frequency of orders per customer
City	City where the customer or order is located	Geographical sales distribution, city-wise performance
State	State where the customer or order is located	State-wise sales analysis, regional market trends
Country	Country where the customer or order is located	International sales analysis, country market penetration
Region	Region classification of the order location	Regional sales trends, comparison of markets within regions
Market	Market classification of the order location	Market share analysis, growth opportunities in different markets
Category	Broad classification of the ordered items	Sales performance by category, inventory management
Segment	Type of customer segment (e.g., Consumer, Corporate)	Customer segmentation effectiveness, segment-specific marketing
Sub-Category	Specific classification of ordered items	Product line performance, targeted promotions
Sales	Total sales amount of the order	Sales trend analysis, high-value orders
Quantity	Number of items per order	Order size analysis, bulk buying trends
Discount	Discount given on the order	Impact of discounts on sales, optimal discount strategies
Profit	Profit made from the order	Profitability analysis, identifying high and low profit margins

This table provides a structured overview of the columns, facilitating an in-depth analysis of various business aspects such as customer behavior, regional sales performance, and overall business health through key financial metrics.

Top 5 Customers

Show Top 5 Customers by Sales

Show as a Column chart

Add data labels and remove y axis, shot Top Customer in Green Colour, others in Gray, also add an average of 5 as a horizontal line

Top 5 Customers

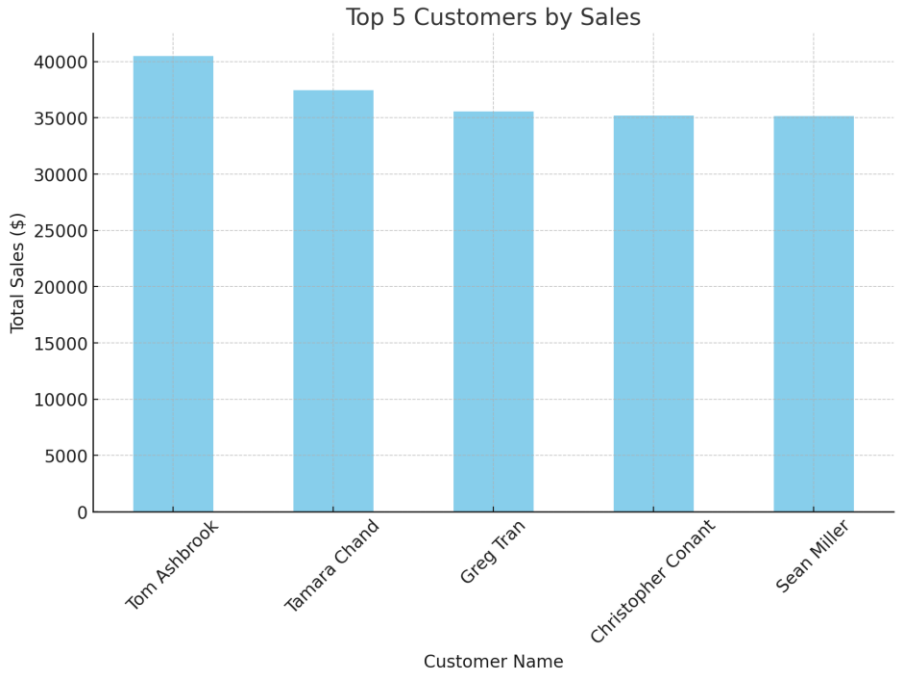
```
# Group data by 'Customer Name' and sum the 'Sales' for each customer
top_customers_by_sales = data.groupby('Customer Name')['Sales'].sum().sort_values(ascending=False).head(5)

# Display the top 5 customers
top_customers_by_sales
```

The top 5 customers by sales are:

1. **Tom Ashbrook** - $40,488.07
2. **Tamara Chand** - $37,457.33
3. **Greg Tran** - $35,550.95
4. **Christopher Conant** - $35,187.08
5. **Sean Miller** - $35,170.93

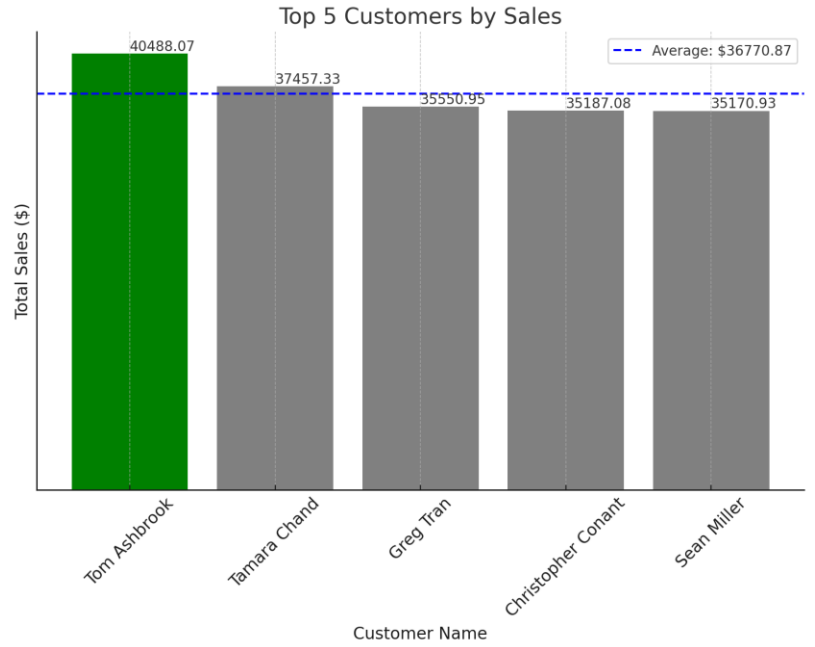
Top 5 Customers

```python
import matplotlib.pyplot as plt

# Plotting the top 5 customers by sales as a column chart
plt.figure(figsize=(10, 6))
top_customers.plot(kind='bar', color='skyblue')
plt.title('Top 5 Customers by Sales')
plt.xlabel('Customer Name')
plt.ylabel('Total Sales ($)')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



Top 5 Customers

```python
# Adding data labels and customizing the chart as per the request
plt.figure(figsize=(10, 6))
bars = plt.bar(top_customers.index, top_customers, color=['green' if i == 0 else 'gray' for i in
range(len(top_customers))])

# Adding data labels
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), va='bottom')  # va: vertical alignment

# Adding average line
average_sales = top_customers.mean()
plt.axhline(y=average_sales, color='blue', linestyle='--', label=f'Average: ${round(average_sales, 2)}')

plt.title('Top 5 Customers by Sales')
plt.xlabel('Customer Name')
plt.ylabel('Total Sales ($)')
plt.yticks([])  # Removing y-axis labels
plt.xticks(rotation=45)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

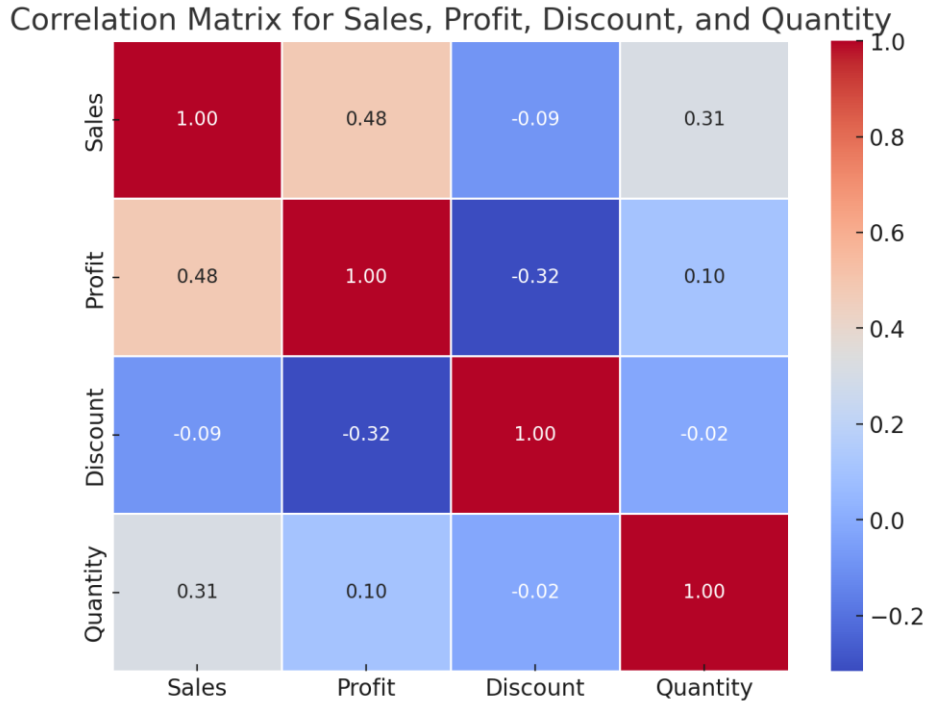


Correlation Matrix

Create a correlation matrix for Sales, Profit, Discount, Quantity.

Correlation Matrix

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix for Sales, Profit, Discount, and Quantity')
plt.show()
```



Cluster Analysis

Show Sum of Sales on x-axis

Show Sum of Profit on y-axis

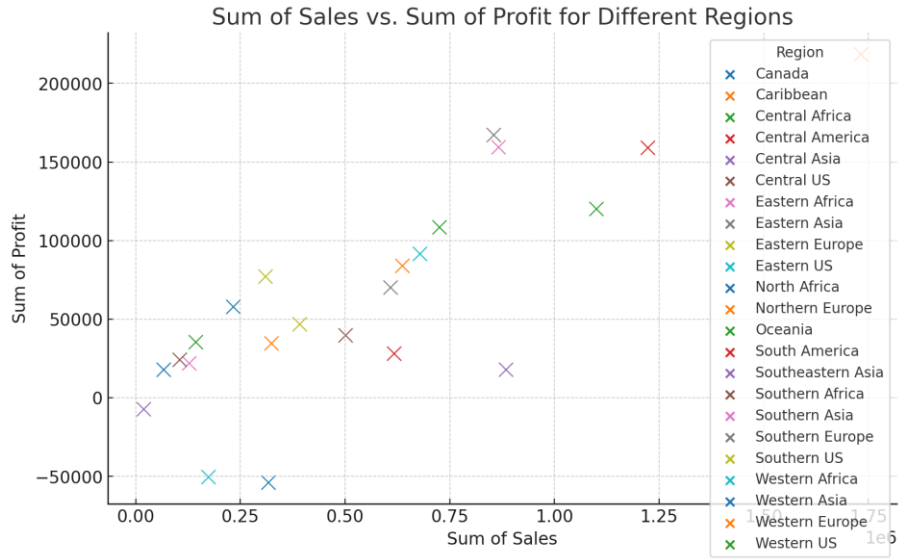
Create a cluster for Regions

Create 5 clusters

Can you do elbow method

Cluster Analysis

```python
# Aggregate the data to get the sum of Sales and Profit for each Region
region_summary = orders_data.groupby('Region').agg({'Sales': 'sum', 'Profit': 'sum'}).reset_index()

# Encode the 'Region' column for clustering
region_summary_encoded = region_summary.copy()
label_encoders['Region'] = LabelEncoder()
region_summary_encoded['Region_encoded'] = label_encoders['Region'].fit_transform(region_summary['Region'])

# Plot the sum of Sales (x-axis) vs. sum of Profit (y-axis), coloring by clusters (regions)
plt.figure(figsize=(10, 6))
sns.scatterplot(data=region_summary_encoded, x='Sales', y='Profit', hue='Region', palette='tab10', s=100)
plt.title('Sum of Sales vs. Sum of Profit for Different Regions')
plt.xlabel('Sum of Sales')
plt.ylabel('Sum of Profit')
plt.legend(title='Region')
plt.show()
```

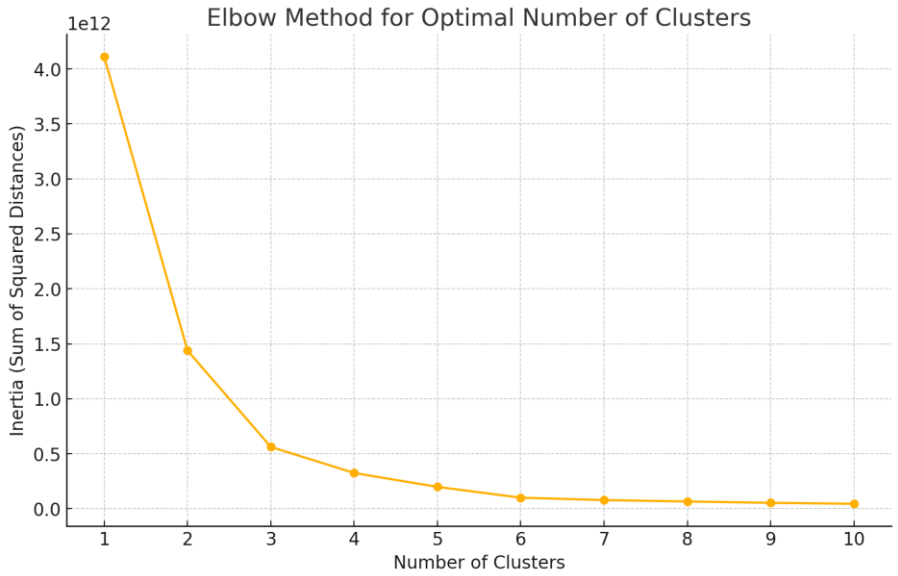


Cluster Analysis

```python
from sklearn.cluster import KMeans

# Prepare data for clustering
X = region_summary[['Sales', 'Profit']]

# Apply KMeans clustering with 5 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
region_summary['Cluster'] = kmeans.fit_predict(X)

# Plot the sum of Sales (x-axis) vs. sum of Profit (y-axis) with clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(data=region_summary, x='Sales', y='Profit', hue='Cluster', palette='viridis', s=100)
plt.title('Sum of Sales vs. Sum of Profit with 5 Clusters')
plt.xlabel('Sum of Sales')
plt.ylabel('Sum of Profit')
plt.legend(title='Cluster')
plt.show()
```

Cluster Analysis

```python
# Calculate the sum of squared distances (inertia) for a range of cluster numbers (1 to 10) using the elbow method
inertia_values = []
cluster_range = range(1, 11)

for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

# Plot the inertia values to identify the "elbow"
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, inertia_values, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.xticks(cluster_range)
plt.show()
```



Sankey

Create a Sankey to show sum of sales on Market and Segment

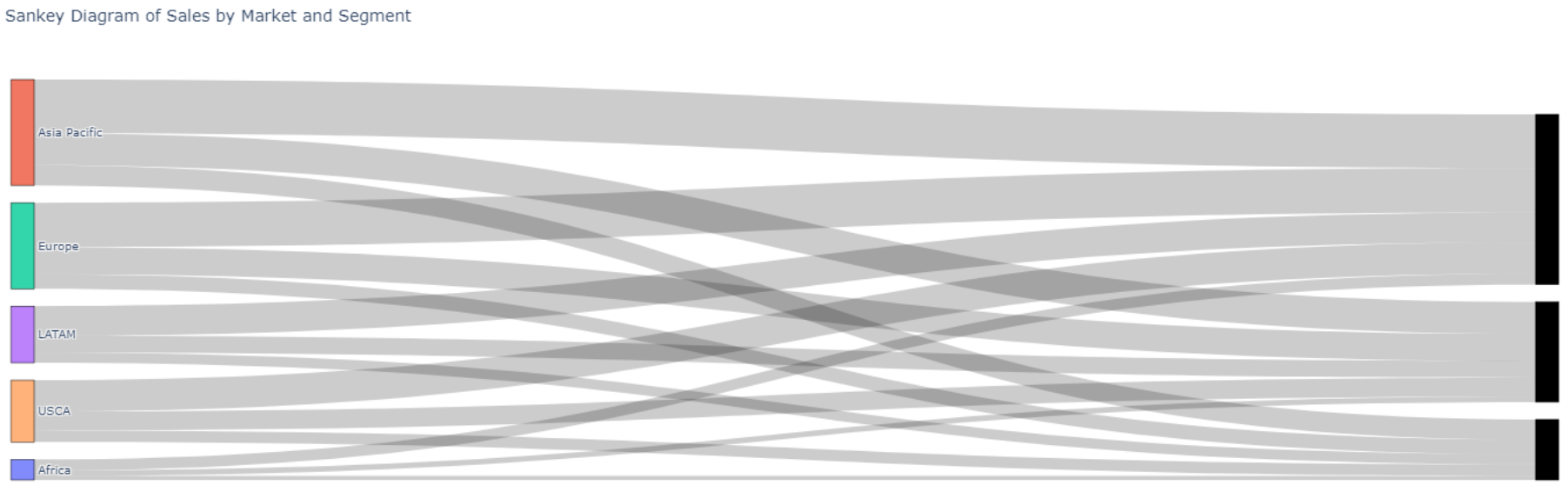
Sankey

Create a Sankey to show sum of sales on Market and Segment

Sankey

```python
import plotly.graph_objects as go

# Prepare the data for the Sankey diagram
market_segment_summary = df.groupby(['Market', 'Segment']).agg({'Sales': 'sum'}).reset_index()

# Create unique labels for 'Market' and 'Segment'
markets = market_segment_summary['Market'].unique().tolist()
segments = market_segment_summary['Segment'].unique().tolist()

# Create labels for the Sankey diagram
labels = markets + segments

# Define source and target indices for the Sankey diagram
source_indices = [labels.index(market) for market in market_segment_summary['Market']]
target_indices = [labels.index(segment) + len(markets) for segment in market_segment_summary['Segment']]

# Define the sales values for the links
sales_values = market_segment_summary['Sales'].tolist()

# Create the Sankey diagram
fig = go.Figure(go.Sankey(
    node=dict(
        pad=15,
        thickness=20,
        line=dict(color="black", width=0.5),
        label=labels,
    ),
    link=dict(
        source=source_indices,
        target=target_indices,
        value=sales_values,
    )
))

# Set the title and display the Sankey diagram
fig.update_layout(title_text="Sankey Diagram of Sales by Market and Segment", font_size=10)
fig.show()
```

Sankey



Wordcloud

Create a wordcloud for Top 15 customers

Wordcloud

```python
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Aggregate sales by customer names and sort to get the top 50 customers
top_customers = orders_data.groupby('Customer Name')['Sales'].sum().sort_values(ascending=False).head(50)

# Generate a word cloud using the customer names and their sales as weights
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(top_customers)

# Plot the word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Top 50 Customers by Sales')
plt.show()
```

RFM Analysis

Perform an RFM analysis

50% weight to M, 30% to F and 20% to R

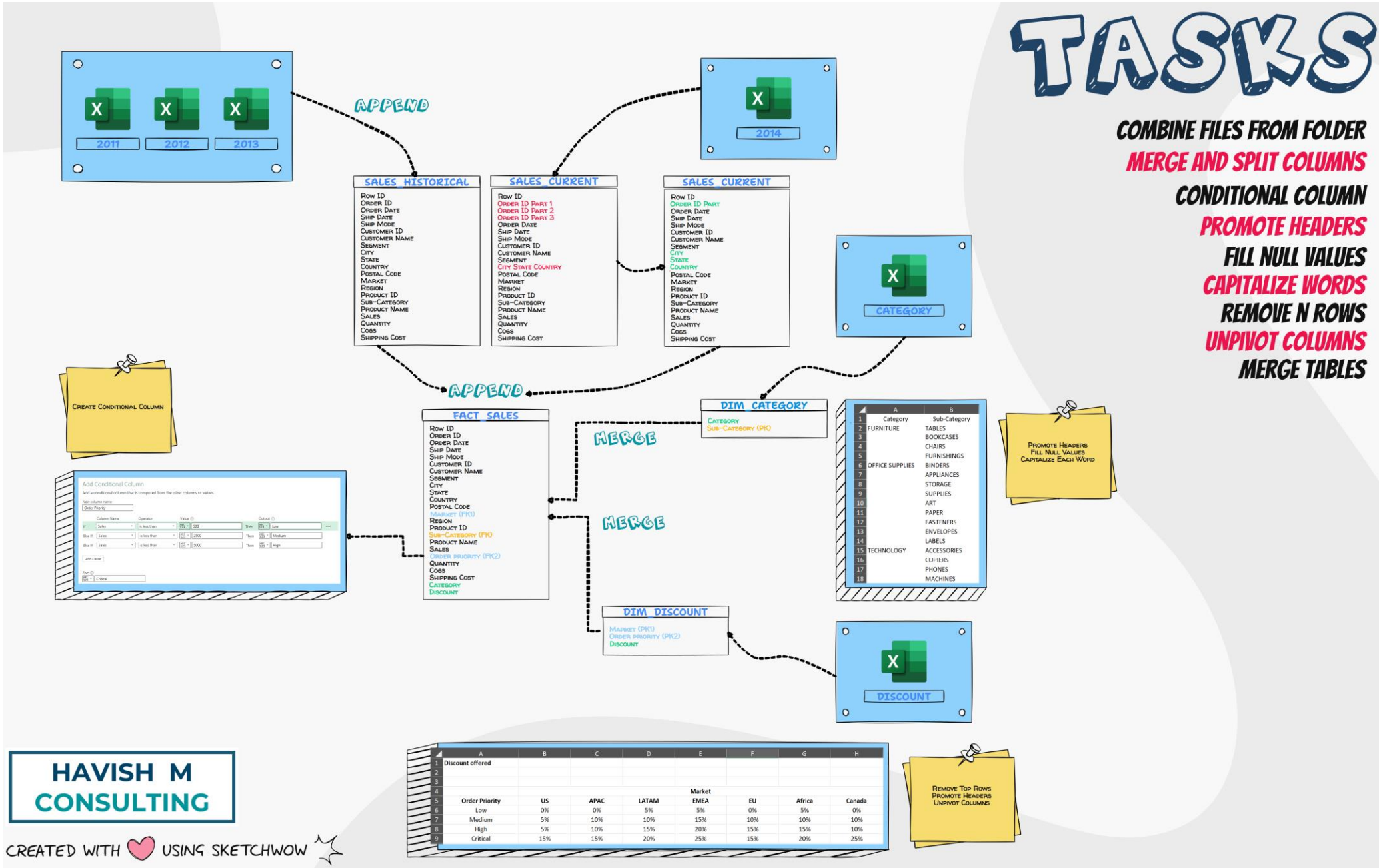
Find Top 5 customers

RFM Analysis

```python
from datetime import datetime

# Assuming 'Fact_Sales' is already loaded and has 'Order Date', 'Customer ID', and 'Sales' columns

# Set the current date to the latest date in the dataset for consistency in recency calculation
current_date = Fact_Sales['Order Date'].max() + pd.Timedelta(days=1)

# Calculate RFM components
rfm = Fact_Sales.groupby('Customer ID').agg({
    'Order Date': lambda x: (current_date - x.max()).days,  # Recency: Days since last purchase
    'Order ID': 'count',                                    # Frequency: Count of transactions
    'Sales': 'sum'                                          # Monetary: Sum of sales
}).rename(columns={'Order Date': 'Recency', 'Order ID': 'Frequency', 'Sales': 'Monetary'})

# Scale the RFM data
rfm_scaled = rfm.copy()
for column in rfm.columns:
    rfm_scaled[column] = (rfm[column] - rfm[column].min()) / (rfm[column].max() - rfm[column].min())

# Calculate weighted RFM score
rfm_scaled['RFM_Score'] = (
    rfm_scaled['Monetary'] * 0.50 +
    rfm_scaled['Frequency'] * 0.30 +
    (1 - rfm_scaled['Recency']) * 0.20  # Recency is inverse; more recent should have higher score
)

# Find top 5 customers based on RFM Score
top_5_customers = rfm_scaled.sort_values('RFM_Score', ascending=False).head(5)

top_5_customers[['RFM_Score']]  # Return only the RFM Score for clarity
```

The top 5 customers based on the weighted RFM scores are:

1. Customer ID **GT-14710** with an RFM score of **0.910**
2. Customer ID **TA-21385** with an RFM score of **0.897**
3. Customer ID **BW-11110** with an RFM score of **0.870**
4. Customer ID **BE-11335** with an RFM score of **0.866**
5. Customer ID **ZC-21910** with an RFM score of **0.858**



HAVISH M
CONSULTING

Combine Files

Can you append these 3 files

Call the df Sales_Historical

Combine files

```
import pandas as pd

# Load the Excel files
file_2011 = pd.read_excel("/mnt/data/2011.xlsx")
file_2012 = pd.read_excel("/mnt/data/2012.xlsx")
file_2013 = pd.read_excel("/mnt/data/2013.xlsx")

# Combine all files into one DataFrame
Sales_Historical = pd.concat([file_2011, file_2012, file_2013], ignore_index=True)

# Save the combined file
Sales_Historical.to_excel("/mnt/data/Sales_Historical.xlsx", index=False)
Sales_Historical.head()
```

Check 2014

Load this

Call it Sales_Current

Then tell me whether Sales_Historical and Sales_Current have matching columns

Check 2014

```
# Load the newly uploaded Excel file for 2014
Sales_Current = pd.read_excel("/mnt/data/2014.xlsx")

# Identify differences in columns between Sales_Historical and Sales_Current
historical_columns = set(Sales_Historical.columns)
current_columns = set(Sales_Current.columns)

# Columns in Sales_Historical not in Sales_Current
columns_in_historical_not_in_current = historical_columns - current_columns

# Columns in Sales_Current not in Sales_Historical
columns_in_current_not_in_historical = current_columns - historical_columns

columns_in_historical_not_in_current, columns_in_current_not_in_historical
```

Combine Columns

In Sales_Current, please combine Order ID Part1, 2, 3 into a single Column

Combine Columns

```
# Combine the Order ID parts into a single column for Sales_Current
Sales_Current['Order ID'] = Sales_Current['Order ID Part 1'].astype(str) + "-" + \
                            Sales_Current['Order ID Part 2'].astype(str) + "-" + \
                            Sales_Current['Order ID Part 3'].astype(str)

# Remove the original Order ID parts columns
Sales_Current.drop(['Order ID Part 1', 'Order ID Part 2', 'Order ID Part 3'], axis=1, inplace=True)

Sales_Current.head()
```

Split Columns

Split City State Country into 3 columns using comma and space as the delimiter

Split Columns

```
# Split the 'City State Country' column into three separate columns using comma and space as the delimiter
Sales_Current[['City', 'State', 'Country']] = Sales_Current['City State Country'].str.split(', ', expand=True)

# Remove the original 'City State Country' column
Sales_Current.drop('City State Country', axis=1, inplace=True)

Sales_Current.head()
```

Combine Historical and Current

Append Sales_Historical and Sales_Current into a new df called Fact_Sales

Combine Historical and Current

```
# Append Sales_Current to Sales_Historical into a new DataFrame called Fact_Sales
Fact_Sales = pd.concat([Sales_Historical, Sales_Current], ignore_index=True)

Fact_Sales.head()
```

Create Order Priority

In the Fact_Sales please add a column for Order Priority

Sales < 1000 Low

Sales < 2500 Medium

Sales < 5000 High

else Critical

Create Order Priority

```python
# Define a function to determine order priority based on Sales value
def determine_order_priority(sales):
    if sales < 1000:
        return "Low"
    elif sales < 2500:
        return "Medium"
    elif sales < 5000:
        return "High"
    else:
        return "Critical"

# Apply the function to the Sales column in Fact_Sales DataFrame
Fact_Sales['Order Priority'] = Fact_Sales['Sales'].apply(determine_order_priority)

Fact_Sales.head()
```

Clean Category File

Read the file

Call it Dim_Category

Fill missing category values

Change to proper case

Clean Category File

```
# Load the newly uploaded Excel file
Dim_Category = pd.read_excel("/mnt/data/Category.xlsx")
Dim_Category.head()

# Fill missing values in 'Category' column using forward fill method
Dim_Category['Category'].fillna(method='ffill', inplace=True)
Dim_Category.head()

# Convert Category and Sub-Category to Proper Case (Title Case)
Dim_Category['Category'] = Dim_Category['Category'].str.title()
Dim_Category['Sub-Category'] = Dim_Category['Sub-Category'].str.title()

Dim_Category.head()
```

Merge Category Data

Merge Fact_Sales with Dim_Category on the Sub-category column

Merge Category Data

```
# Merge Fact_Sales DataFrame with Dim_Category on the Sub-Category column
merged_sales = pd.merge(Fact_Sales, Dim_Category, on='Sub-Category', how='left')

merged_sales.head()
```

Clean Discount File

Call the file Dim_Discount

Reformat headers properly

Unpivot my data

Clean Discount File

```
# Load the newly uploaded Excel file
Dim_Discount = pd.read_excel("/mnt/data/Discount.xlsx")
Dim_Discount.head()

# It appears the actual data starts from the fourth row, and the headers need to be set properly
# Dropping the initial rows and setting the new header
Dim_Discount.columns = Dim_Discount.iloc[3]
Dim_Discount = Dim_Discount.drop(Dim_Discount.index[0:4])
Dim_Discount.reset_index(drop=True, inplace=True)

# Display the cleaned DataFrame
Dim_Discount.head()

# Unpivot the data using pandas melt to convert it from wide to long format
Dim_Discount = pd.melt(Dim_Discount, id_vars='Order Priority', var_name='Market', value_name='Discount Rate')

Dim_Discount.head()
```

Merge Discount File

Merge Market and Order Priority in Dim_Discount with Fact_Sales

Merge Discount File

```
# Merge the unpivoted_discount with Fact_Sales on both 'Market' and 'Order Priority'
merged_sales_discount = pd.merge(Fact_Sales, unpivoted_discount, on=['Market', 'Order Priority'], how='left')

merged_sales_discount.head()
```

Export data

Export merged data

Export Data

```
# Save the merged DataFrame to an Excel file
merged_sales_discount.to_excel("/mnt/data/Merged_Sales_Discount.xlsx", index=False)

"Merged_Sales_Discount.xlsx"
```

Combine Code

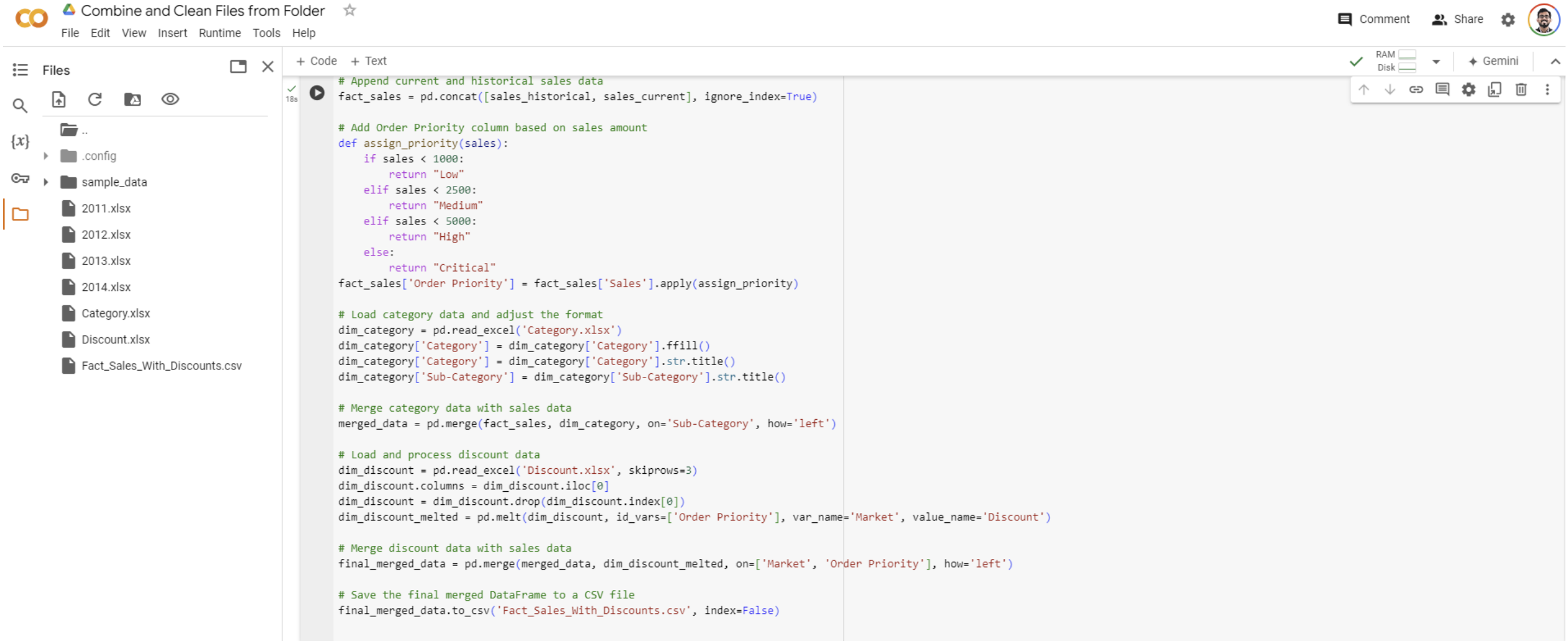
From the very first step, combine all codes into a single code

Combine Code

```python
import pandas as pd

# Load historical sales data from Excel files
df_2011 = pd.read_excel('/mnt/data/2011.xlsx')
df_2012 = pd.read_excel('/mnt/data/2012.xlsx')
df_2013 = pd.read_excel('/mnt/data/2013.xlsx')
sales_current = pd.read_excel('/mnt/data/2014.xlsx')

# Append historical sales data into a single DataFrame
sales_historical = pd.concat([df_2011, df_2012, df_2013], ignore_index=True)

# Combine parts of Order ID in current sales data
sales_current['Order ID'] = sales_current['Order ID Part 1'].astype(str) + '-' + sales_current['Order ID Part
2'].astype(str) + '-' + sales_current['Order ID Part 3'].astype(str)
sales_current.drop(['Order ID Part 1', 'Order ID Part 2', 'Order ID Part 3'], axis=1, inplace=True)

# Split 'City State Country' into separate columns
sales_current[['City', 'State', 'Country']] = sales_current['City State Country'].str.split(', ', expand=True)
sales_current.drop('City State Country', axis=1, inplace=True)

# Append current and historical sales data
fact_sales = pd.concat([sales_historical, sales_current], ignore_index=True)

# Add Order Priority column based on sales amount
def assign_priority(sales):
    if sales < 1000:
        return "Low"
    elif sales < 2500:
        return "Medium"
    elif sales < 5000:
        return "High"
    else:
        return "Critical"
fact_sales['Order Priority'] = fact_sales['Sales'].apply(assign_priority)
```

```python
# Load category data and adjust the format
dim_category = pd.read_excel('/mnt/data/Category.xlsx')
dim_category['Category'] = dim_category['Category'].ffill()
dim_category['Category'] = dim_category['Category'].str.title()
dim_category['Sub-Category'] = dim_category['Sub-Category'].str.title()

# Merge category data with sales data
merged_data = pd.merge(fact_sales, dim_category, on='Sub-Category', how='left')

# Load and process discount data
dim_discount = pd.read_excel('/mnt/data/Discount.xlsx', skiprows=3)
dim_discount.columns = dim_discount.iloc[0]
dim_discount = dim_discount.drop(dim_discount.index[0])
dim_discount_melted = pd.melt(dim_discount, id_vars=['Order Priority'], var_name='Market', value_name='Discount')

# Merge discount data with sales data
final_merged_data = pd.merge(merged_data, dim_discount_melted, on=['Market', 'Order Priority'], how='left')

# Save the final merged DataFrame to a CSV file
final_merged_data.to_csv('/mnt/data/Fact_Sales_With_Discounts.csv', index=False)
```

Combine Code



THANK YOU!

HAVISH M CONSULTING