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

Data Analysis

Dataset Description

Row ID: Unique identification number of the Row

Order ID: Unique identification number of the Order

Order Date: Date of order

Ship Date: Shipping date of order

Ship Mode: Shipping mode of the order

Customer ID: Unique identification number of the customer

Customer Name: Name of the customer

Segment: Segment of market

City: City name where customer lives

State: State name where customer lives

Country: Country name where customer lives

Postal Code: Postal code of the destination

Market: Market from where the product was purchased

Region: Region

Product ID: Unique identification number of the product

Category: Category of the product

Sub-Category: Sub-Category of the product

Product Name: Name of the product

Sales: Amount of sales

Quantity: Quantity of product

Discount: Discount on the product value

Profit: Profit made from the sales

Shipping Cost: Cost of shipping

Order Priority: Proirity of the order

Tasks to be performed:

Import required libraries and load the dataset

Generate the dataset report using sweetviz

Perform necessary data preprocessing: Check missing values Check datatype of columns Fill missing values with mean, median or 0

Perform Exploratory Data Analysis (EDA) on the dataset Plot Univariate Distributions Plot Bi-Variate Distributions

Pre-process that data set for modeling

Handle Missing values present in the dataset

Encode the categorical variables present

Split the data into training and testing set using sklearn's train_test_split function

Modelling

Build and evaluate an Interactive Linear Regression

```
Global_superstore= pd.read_csv('/content/Global_Superstore2.csv')
```

```
Global_superstore.head(5)
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	City	State	...	Product ID	Category
0	32298	CA-2012-124891	31-07-2012	31-07-2012	Same Day	RH-19495	Rick Hansen	Consumer	New York City	New York	...	TEC-AC-10003033	Technology
1	26341	IN-2013-77878	05-02-2013	07-02-2013	Second Class	JR-16210	Justin Ritter	Corporate	Wollongong	New South Wales	...	FUR-CH-10003950	Furniture
2	25330	IN-2013-71249	17-10-2013	18-10-2013	First Class	CR-12730	Craig Reiter	Consumer	Brisbane	Queensland	...	TEC-PH-10004664	Technology
3	13524	ES-2013-1579342	28-01-2013	30-01-2013	First Class	KM-16375	Katherine Murray	Home Office	Berlin	Berlin	...	TEC-PH-10004583	Technology
4	47221	SG-2013-4320	05-11-2013	06-11-2013	Same Day	RH-9495	Rick Hansen	Consumer	Dakar	Dakar	...	TEC-SHA-10000501	Technology

5 rows × 24 columns

```
Global_superstore.value_counts().sum()
```

```
9991
```

```
store=Global_superstore
```

```
store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51290 entries, 0 to 51289
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Row ID          51290 non-null  int64  
 1   Order ID        51290 non-null  object 
 2   Order Date      51290 non-null  object 
 3   Ship Date       51290 non-null  object 
 4   Ship Mode       51290 non-null  object 
 5   Customer ID     51290 non-null  object 
 6   Customer Name   51290 non-null  object 
 7   Segment         51290 non-null  object 
 8   City            51290 non-null  object 
 9   State           51290 non-null  object 
 10  Country         51290 non-null  object 
 11  Postal Code     9994 non-null   float64
 12  Market          51290 non-null  object 
 13  Region          51290 non-null  object 
 14  Product ID      51290 non-null  object 
 15  Category        51290 non-null  object 
 16  Sub-Category    51290 non-null  object 
 17  Product Name    51290 non-null  object 
 18  Sales           51290 non-null  float64
 19  Quantity        51290 non-null  int64  
 20  Discount        51282 non-null  float64
 21  Profit          51277 non-null  float64
 22  Shipping Cost   51282 non-null  float64
 23  Order Priority  51286 non-null  object 
dtypes: float64(5), int64(2), object(17)
memory usage: 9.4+ MB
```

```
store.columns
```

```
Index(['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Ship Mode',
       'Customer ID', 'Customer Name', 'Segment', 'City', 'State', 'Country',
       'Postal Code', 'Market', 'Region', 'Product ID', 'Category',
       'Sub-Category', 'Product Name', 'Sales', 'Quantity', 'Discount',
       'Profit', 'Shipping Cost', 'Order Priority'],
      dtype='object')
```

```
# Checking for missing values
def check_miss(store):
    '''
    data: requires a DataFrame object.
    ---
    returns: A DataFrame with details about missing values
    '''
    cnull=[sum(store[y].isnull()) for y in store.columns]
    miss=pd.DataFrame({'Null Values':
                        [any(store[x].isnull()) for x in store.columns],
                    'Count_Nulls':cnull,
                    'Percentage_Nulls':list((np.array(cnull)*100)/store.shape[0]),
                    'MValues':cnull,
                    'Dtype':store.dtypes
                      })
    return miss.sort_values(by='MValues',ascending=False)

check_miss(store)
```

	Null Values	Count_Nulls	Percentage_Nulls	MValues	Dtype
Postal Code	True	41296	80.514720	41296	float64
Profit	True	13	0.025346	13	float64
Shipping Cost	True	8	0.015598	8	float64
Discount	True	8	0.015598	8	float64
Order Priority	True	4	0.007799	4	object
Region	False	0	0.000000	0	object
Quantity	False	0	0.000000	0	int64
Sales	False	0	0.000000	0	float64
Product Name	False	0	0.000000	0	object
Sub-Category	False	0	0.000000	0	object
Category	False	0	0.000000	0	object
Product ID	False	0	0.000000	0	object
Row ID	False	0	0.000000	0	int64
Order ID	False	0	0.000000	0	object
Country	False	0	0.000000	0	object
State	False	0	0.000000	0	object
City	False	0	0.000000	0	object
Segment	False	0	0.000000	0	object
Customer Name	False	0	0.000000	0	object
Customer ID	False	0	0.000000	0	object
Ship Mode	False	0	0.000000	0	object
Ship Date	False	0	0.000000	0	object
Order Date	False	0	0.000000	0	object

Order Date	False	0	0.000000	0	object
Market	False	0	0.000000	0	object

Datatype conversion

```
# Convert data types for optimization
store['Row ID'] = store['Row ID'].astype('int32')
store['Order ID'] = store['Order ID'].astype('category')
store['Order Date'] = pd.to_datetime(store['Order Date'], format='%d-%m-%Y')
store['Ship Date'] = pd.to_datetime(store['Ship Date'], format='%d-%m-%Y')
store['Customer ID'] = store['Customer ID'].astype('category')
store['Customer Name'] = store['Customer Name'].astype('category')
store['Segment'] = store['Segment'].astype('category')
store['City'] = store['City'].astype('category')
store['State'] = store['State'].astype('category')
store['Country'] = store['Country'].astype('category')
store['Market'] = store['Market'].astype('category')
store['Region'] = store['Region'].astype('category')
store['Product ID'] = store['Product ID'].astype('category')
store['Category'] = store['Category'].astype('category')
store['Sub-Category'] = store['Sub-Category'].astype('category')
store['Product Name'] = store['Product Name'].astype('str')
store['Sales'] = store['Sales'].astype('int64')
```

```
store
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	City	State	...	Product ID	Cate
0	32298	CA-2012-124891	2012-07-31	2012-07-31	Same Day	RH-19495	Rick Hansen	Consumer	New York City	New York	...	TEC-AC-10003033	Techn
1	26341	IN-2013-77878	2013-02-05	2013-02-07	Second Class	JR-16210	Justin Ritter	Corporate	Wollongong	New South Wales	...	FUR-CH-10003950	Furn
2	25330	IN-2013-71249	2013-10-17	2013-10-18	First Class	CR-12730	Craig Reiter	Consumer	Brisbane	Queensland	...	TEC-PH-10004664	Techn
3	13524	ES-2013-1579342	2013-01-28	2013-01-30	First Class	KM-16375	Katherine Murray	Home Office	Berlin	Berlin	...	TEC-PH-10004583	Techn
4	47221	SG-2013-4320	2013-11-05	2013-11-06	Same Day	RH-9495	Rick Hansen	Consumer	Dakar	Dakar	...	TEC-SHA-10000501	Techn
...	...	...	...	...	...	...	...	...	...	...	...	...	
51285	29002	IN-2014-62366	2014-06-19	2014-06-19	Same Day	KE-16420	Katrina Edelman	Corporate	Kure	Hiroshima	...	OFF-FA-10000746	O Sup
51286	35398	US-2014-102288	2014-06-20	2014-06-24	Standard Class	ZC-21910	Zuschuss Carroll	Consumer	Houston	Texas	...	OFF-AP-10002906	O Sup

51287	40470	US-2013-155768	2013-12-02	2013-12-02	Same Day	LB-16795	Laurel Beltran	Home Office	Oxnard	California	...	OFF-EN-10001219	(Sup
51288	9596	MX-2012-140767	2012-02-18	2012-02-22	Standard Class	RB-19795	Ross Baird	Home Office	Valinhos	São Paulo	...	OFF-BI-10000806	(Sup
51289	6147	MX-2012-134460	2012-05-22	2012-05-26	Second Class	MC-18100	Mick Crebagga	Consumer	Tipitapa	Managua	...	OFF-PA-10004155	(Sup

51290 rows × 24 columns

```
# keeping the original data aside
orig_data=store.copy()

#filling missing
mean_filled_data=store.copy()

mean_filled_data['Shipping Cost'].fillna(mean_filled_data['Shipping Cost'].mean(),inplace=True)
mean_filled_data['Profit'].fillna(mean_filled_data['Profit'].mean(),inplace=True)
mean_filled_data['Discount'].fillna(mean_filled_data['Discount'].mean(),inplace=True)
```

```
<ipython-input-31-ec934b5e12d5>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  mean_filled_data['Shipping Cost'].fillna(mean_filled_data['Shipping Cost'].mean(),inplace=True)
<ipython-input-31-ec934b5e12d5>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  mean_filled_data['Profit'].fillna(mean_filled_data['Profit'].mean(),inplace=True)
<ipython-input-31-ec934b5e12d5>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  mean_filled_data['Discount'].fillna(mean_filled_data['Discount'].mean(),inplace=True)
```

```
# Filling with mode as Order Priority is categorical
mean_filled_data['Order Priority'].fillna(mean_filled_data['Order Priority'].mode(),inplace=True
```

```
<ipython-input-32-3341eeda6cb6>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  mean_filled_data['Order Priority'].fillna(mean_filled_data['Order Priority'].mode(),inplace=True)
```

```
zero_filled_data=store.copy()

zero_filled_data['Shipping Cost'].fillna(0,inplace=True)
zero_filled_data['Profit'].fillna(0,inplace=True)
zero_filled_data['Discount'].fillna(0,inplace=True)
```

```
<ipython-input-34-1e0f80a35cbb>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  zero_filled_data['Shipping Cost'].fillna(0,inplace=True)
```

```
<ipython-input-34-1e0f80a35cbb>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  zero_filled_data['Profit'].fillna(0,inplace=True)
<ipython-input-34-1e0f80a35cbb>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  zero_filled_data['Discount'].fillna(0,inplace=True)
```

```
# Filling with mode as Order Priority is categorical
zero_filled_data['Order Priority'].fillna(zero_filled_data['Order Priority'].mode(),inplace=True
```

```
<ipython-input-35-5d30b51ac3cb>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series throug
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[


  zero_filled_data['Order Priority'].fillna(zero_filled_data['Order Priority'].mode(),inplace=True)
```

Exploratory Data Analysis

Univariate Distributions

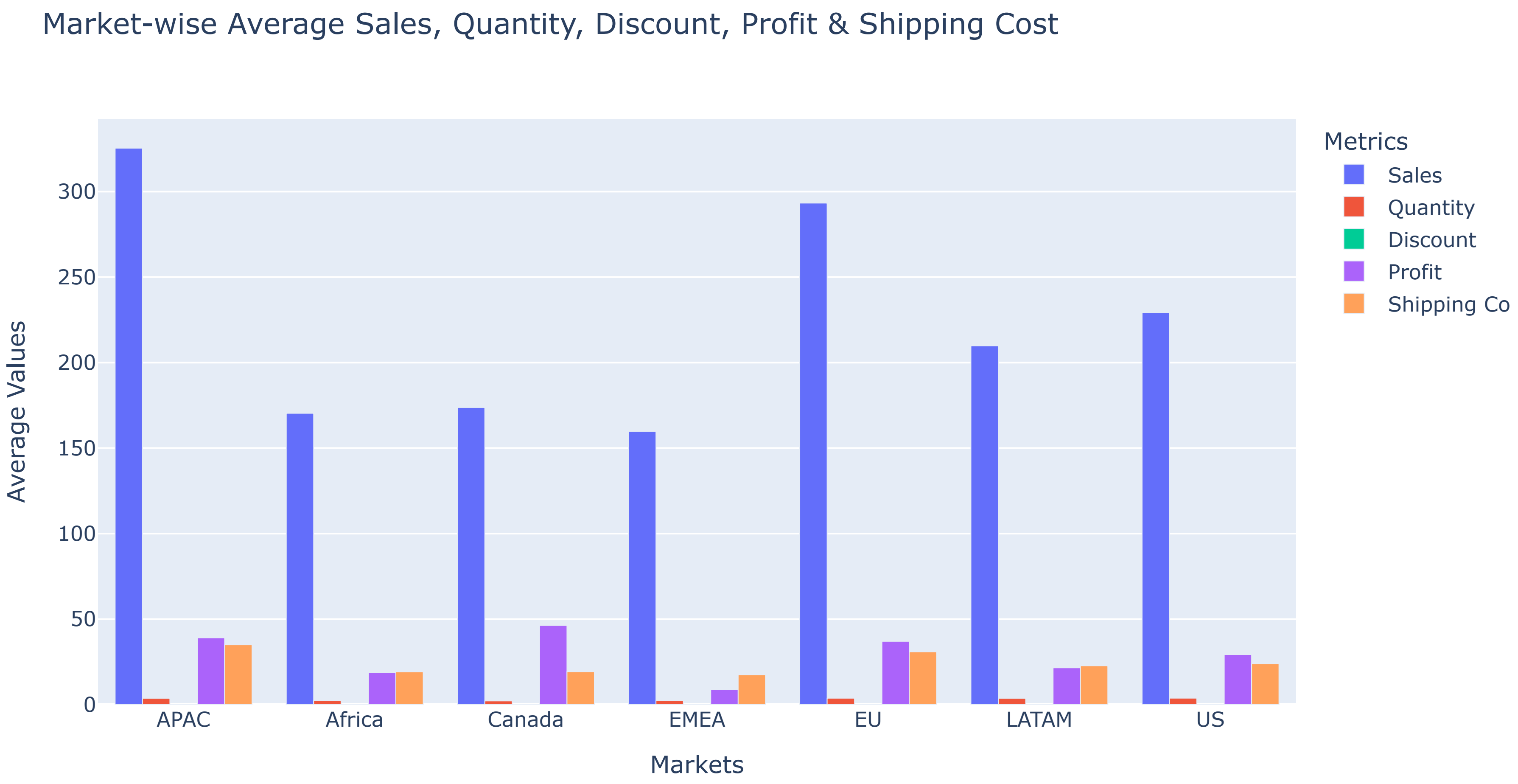
```
# importing required libraries
import plotly.express as px
import plotly.graph_objects as go
```

```
Start coding or generate with AI.
```

```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# Grouping by Market and calculating mean values
shipcst_market = store.groupby('Market').mean(numeric_only=True)

# Extracting market names (index)
markets = shipcst_market.index

# Creating the bar chart
fig = go.Figure(data=[
    go.Bar(name='Sales', x=markets, y=shipcst_market['Sales']),
    go.Bar(name='Quantity', x=markets, y=shipcst_market['Quantity']),
    go.Bar(name='Discount', x=markets, y=shipcst_market['Discount']),
    go.Bar(name='Profit', x=markets, y=shipcst_market['Profit']),
    go.Bar(name='Shipping Cost', x=markets, y=shipcst_market['Shipping Cost'])
])

# Change the bar mode to group
fig.update_layout(
    barmode='group',
    title="Market-wise Average Sales, Quantity, Discount, Profit & Shipping Cost",
    xaxis_title="Markets",
    yaxis_title="Average Values",
    legend_title="Metrics",
)
```

```
# Show the figure
fig.show()
```

```
<ipython-input-38-a64503b559c6>:6: FutureWarning: The default of observed=False is deprecated and will be changed to Tr
  shipcst_market = store.groupby('Market').mean(numeric_only=True)
```



Observations:

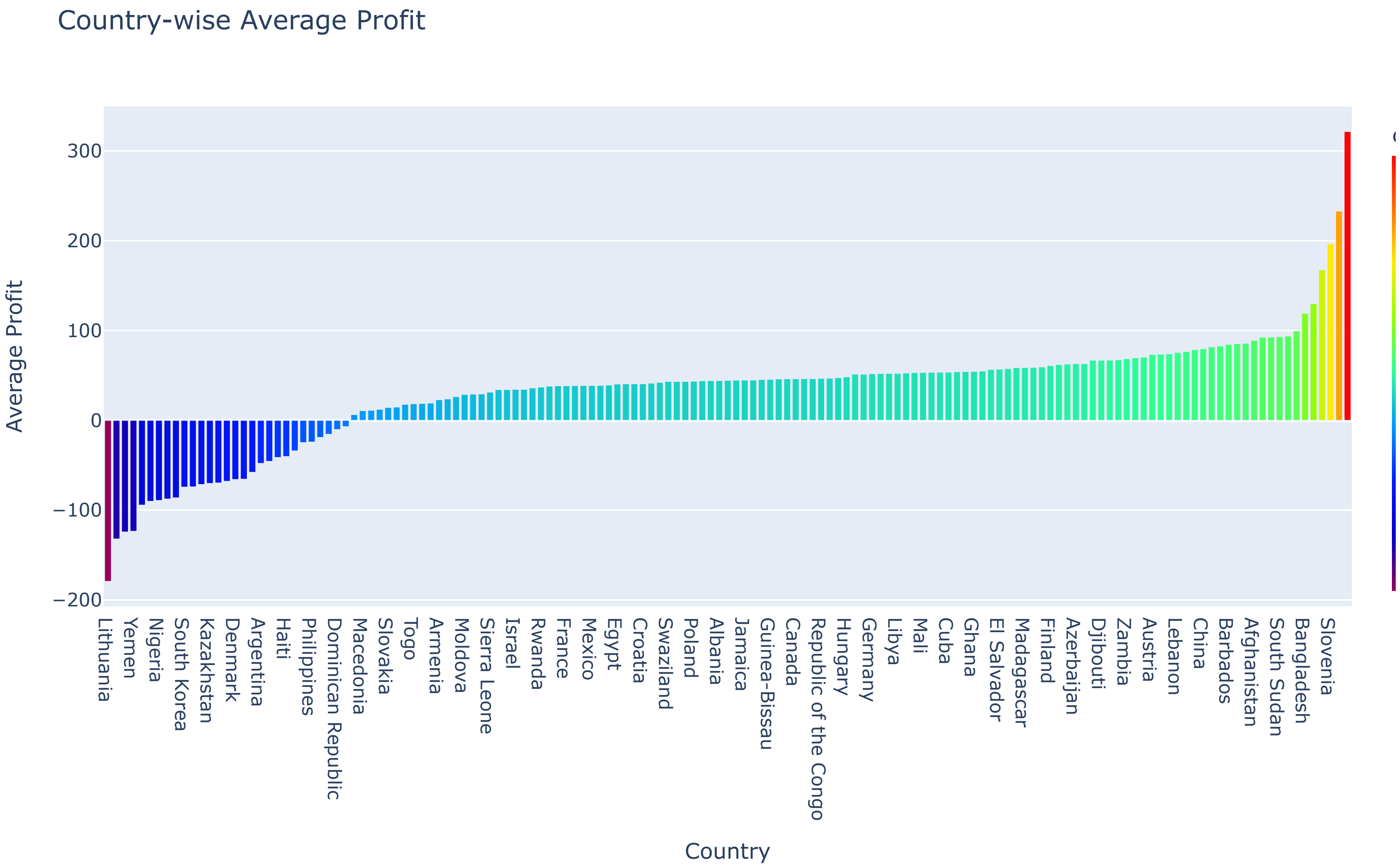
APAC has highest sales while canada makes higest profit

EMEA has highest discount but the sales are lowest

Shipping cost in APAC markest is highest while in other markets its lower

```
# Group by 'Country' and calculate the mean values
country_profit = store.groupby('Country').mean(numeric_only=True)

# Sort by Profit in ascending order
country_profit = country_profit.sort_values(by='Profit')

# Create a bar chart with color intensity based on profit values
fig = px.bar(
    x=country_profit.index,
    y=country_profit['Profit'],
    color=country_profit['Profit'],
    color_continuous_scale=px.colors.sequential.Rainbow,
    height=600,
    width=1000,
    labels={'x': 'Country', 'y': 'Average Profit'},
    title="Country-wise Average Profit"
)

# Show the figure
fig.show()
```

```
ipython-input-41-6855e20c8b65>:2: FutureWarning:

e default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=Fa
```



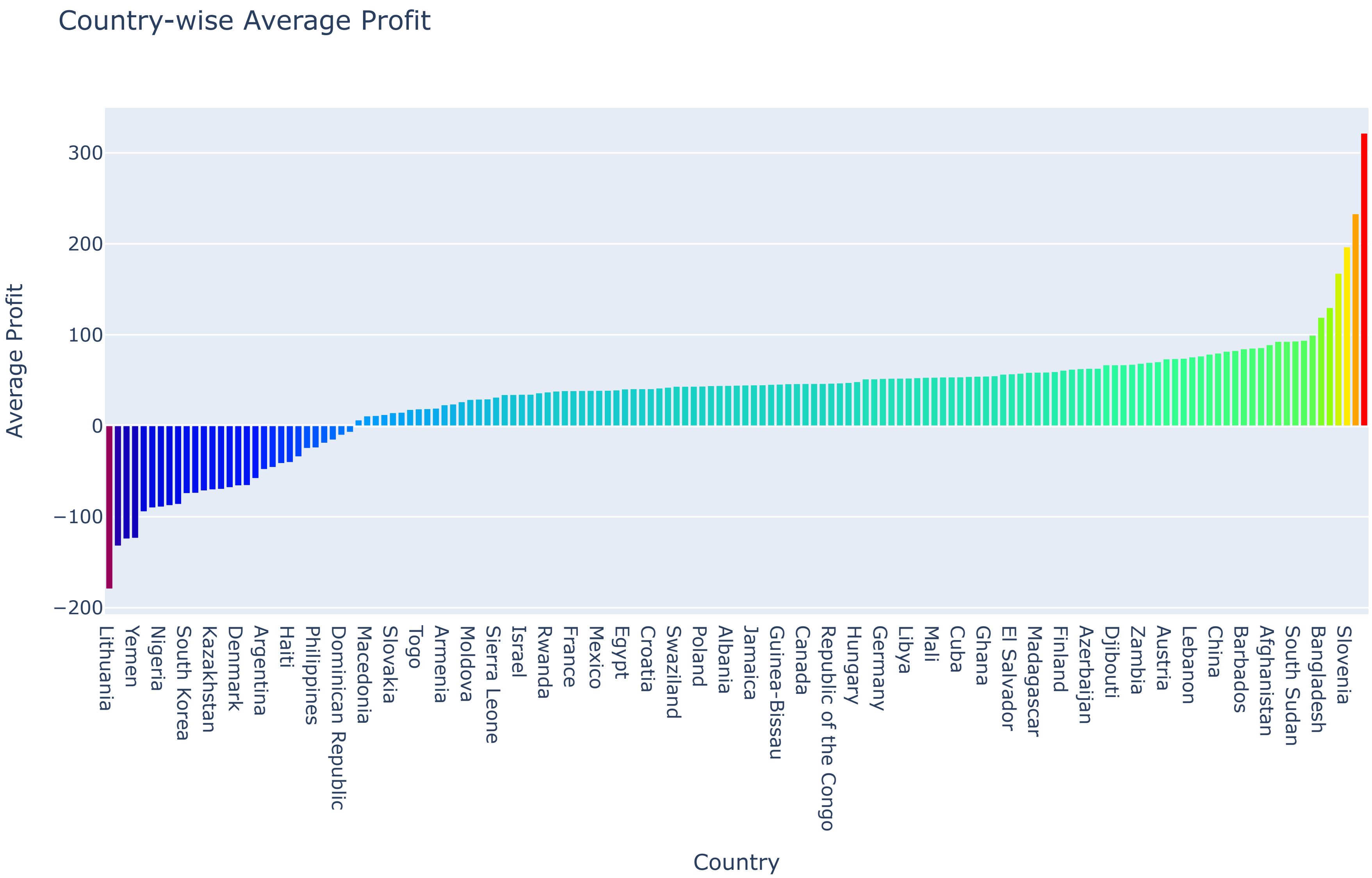
Country-wise Average Profit

Observations:

Lesotho made highest sale while uganda is at lowest

```
# Grouping by 'Country' and calculating mean values
country_profit = store.groupby('Country').mean(numeric_only=True)

# Sorting by Profit in ascending order
country_profit = country_profit.sort_values(by='Profit')

# Creating a bar chart
fig = px.bar(
    x=country_profit.index,
    y=country_profit['Profit'],
    color=country_profit['Profit'],
    color_continuous_scale=px.colors.sequential.Rainbow,
    height=600,
    width=1000,
    labels={'x': 'Country', 'y': 'Average Profit'},
    title="Country-wise Average Profit"
)

# Display the figure
fig.show()
```

```
<ipython-input-42-7e4bc8b8964e>:2: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=Fa
```



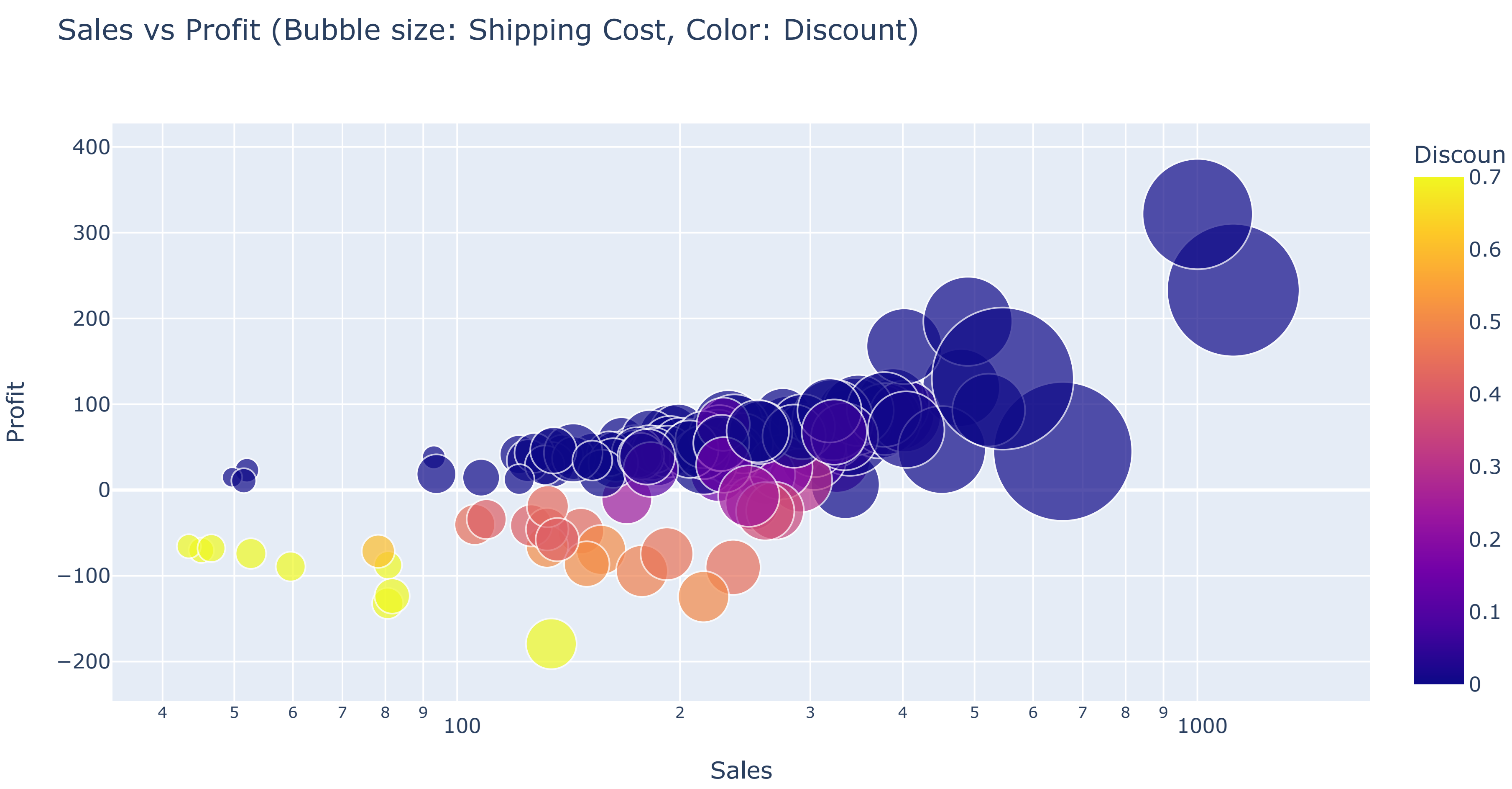
Obeservations

In Lithuania the sore suffered heaviest loss while in Montenegro store made really good profit

In 29 countries the store suffered loss

```
# Grouping by 'Country' and calculating mean values
country_sales = store.groupby('Country').mean(numeric_only=True)

# Creating a scatter plot
fig = px.scatter(
    country_sales,
    x="Sales",
    y="Profit",
    size="Shipping Cost",
    color="Discount",
    hover_name=country_sales.index,
    log_x=True,
    size_max=60,
    title="Sales vs Profit (Bubble size: Shipping Cost, Color: Discount)"
)

# Display the figure
fig.show()
```

```
<ipython-input-44-12ea983c3c94>:2: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=Fa
```



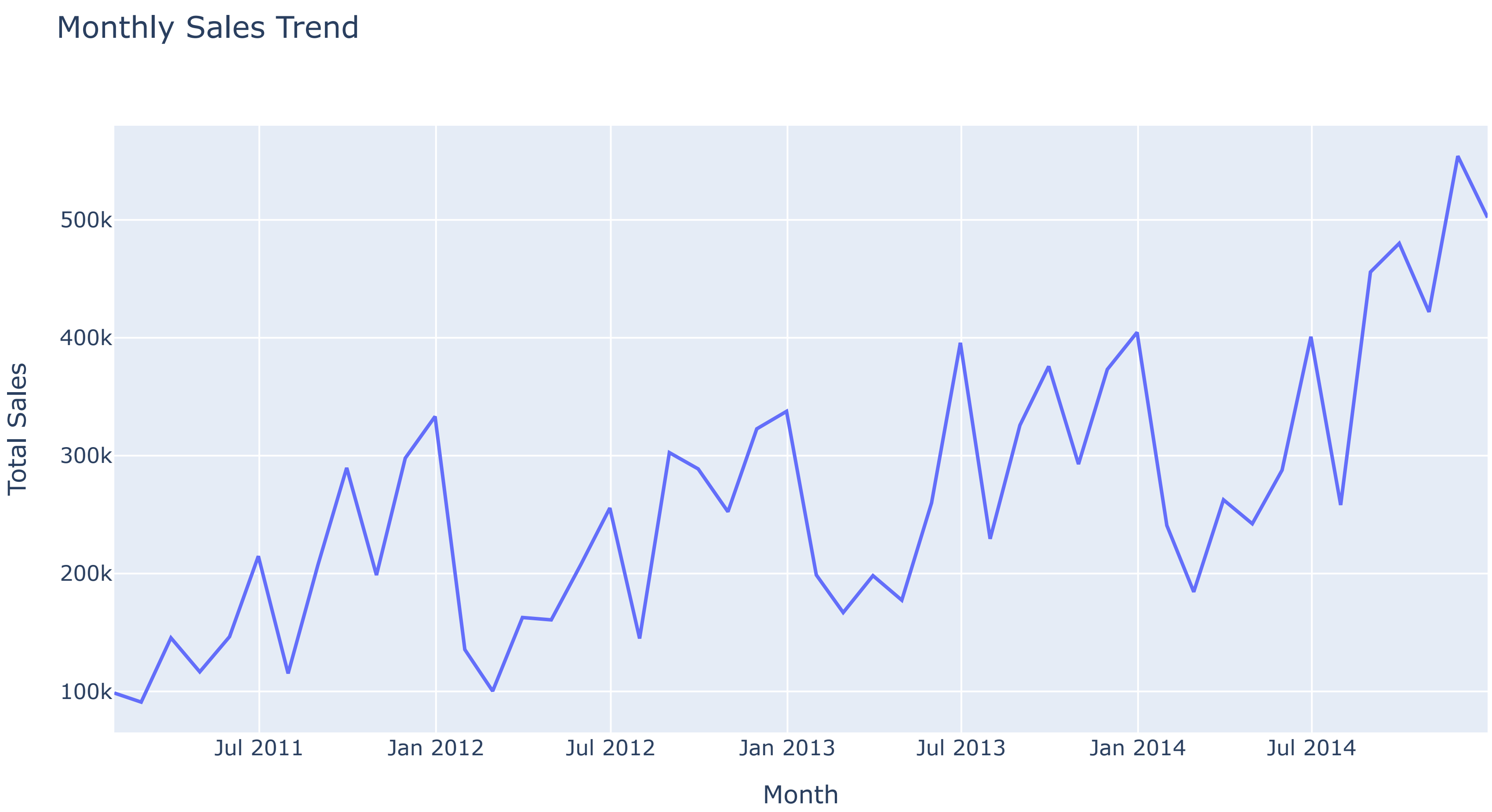
Observations

If the discount is high there will be loss

For higher sales the shipping cost is also high

```
# Creating a copy of the dataset
monthly_sales = store.copy()

# Converting 'Order Date' to datetime (if not already)
monthly_sales['Order Date'] = pd.to_datetime(monthly_sales['Order Date'])

# Setting 'Order Date' as index
monthly_sales.set_index('Order Date', inplace=True)

# Aggregating sales by day and then resampling by month
monthly_sales = monthly_sales.resample('M').sum(numeric_only=True)

# Creating a line chart
fig = px.line(
    x=monthly_sales.index,
    y=monthly_sales['Sales'],
    labels={'x': 'Month', 'y': 'Total Sales'},
    title="Monthly Sales Trend"
)

# Display the figure
fig.show()
```

```
<ipython-input-45-a34587e55f19>:11: FutureWarning:

'M' is deprecated and will be removed in a future version, please use 'ME' instead.
```



Observations

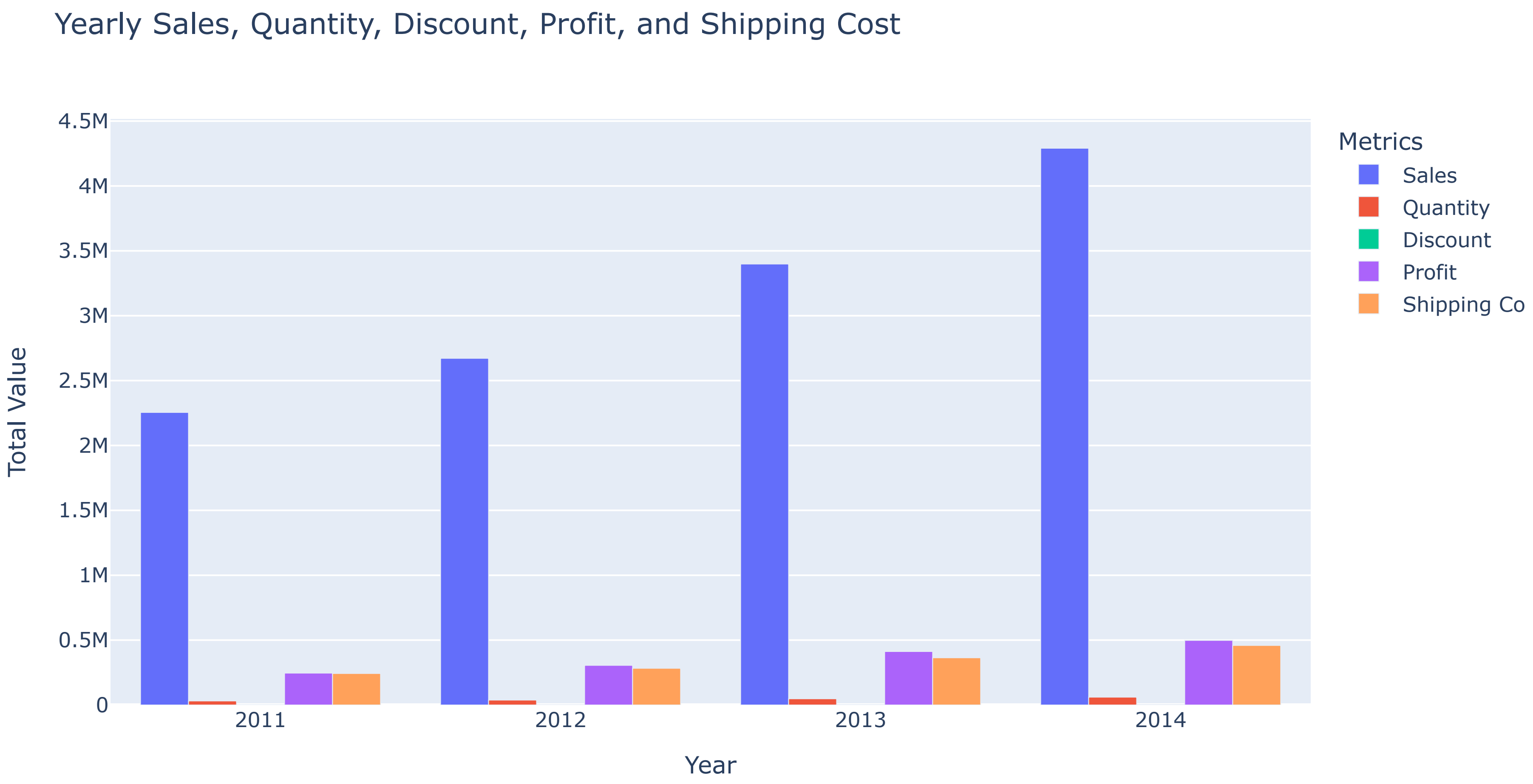
Every june, september, november and december the sales increase really high

Every july the sales are least in the respective year

```
#Yearly Analysis
import pandas as pd

# Creating a copy of the dataset
yearly_sales = store.copy()

# Converting 'Order Date' to datetime (if not already)
yearly_sales['Order Date'] = pd.to_datetime(yearly_sales['Order Date'])

# Setting 'Order Date' as index
yearly_sales.set_index('Order Date', inplace=True)

# Resampling to get yearly sales sum
yearly_sales = yearly_sales.resample('Y').sum(numeric_only=True)

# Display the result
print(yearly_sales)
```

```
               Row ID  Postal Code    Sales  Quantity  Discount       Profit  \
Order Date
2011-12-31  235388025  113271247.0  2254780     31443  1333.294  246013.43554
2012-12-31  277692065  111208247.0  2671802     38111  1548.774  305706.92910
2013-12-31  347629160  140529941.0  3398695     48136  1935.322  412354.10818
2014-12-31  454648445  186563217.0  4290838     60622  2511.588  497995.15946

            Shipping Cost
Order Date
2011-12-31      243032.15
2012-12-31      283052.86
2013-12-31      364146.49
2014-12-31      459184.21
<ipython-input-47-87134d9eb0f7>:14: FutureWarning:

'Y' is deprecated and will be removed in a future version, please use 'YE' instead.
```

```
import pandas as pd
import plotly.graph_objects as go

# Converting 'Order Date' index to year format (YYYY)
year = yearly_sales.index.strftime('%Y')

# Creating a grouped bar chart
fig = go.Figure(data=[
    go.Bar(name='Sales', x=year, y=yearly_sales['Sales']),
    go.Bar(name='Quantity', x=year, y=yearly_sales['Quantity']),
    go.Bar(name='Discount', x=year, y=yearly_sales['Discount']),
    go.Bar(name='Profit', x=year, y=yearly_sales['Profit']),
    go.Bar(name='Shipping Cost', x=year, y=yearly_sales['Shipping Cost'])
])

# Change the bar mode to 'group'
fig.update_layout(
    barmode='group',
    title="Yearly Sales, Quantity, Discount, Profit, and Shipping Cost",
    xaxis_title="Year",
    yaxis_title="Total Value",
    legend_title="Metrics"
)

# Display the figure
fig.show()
```



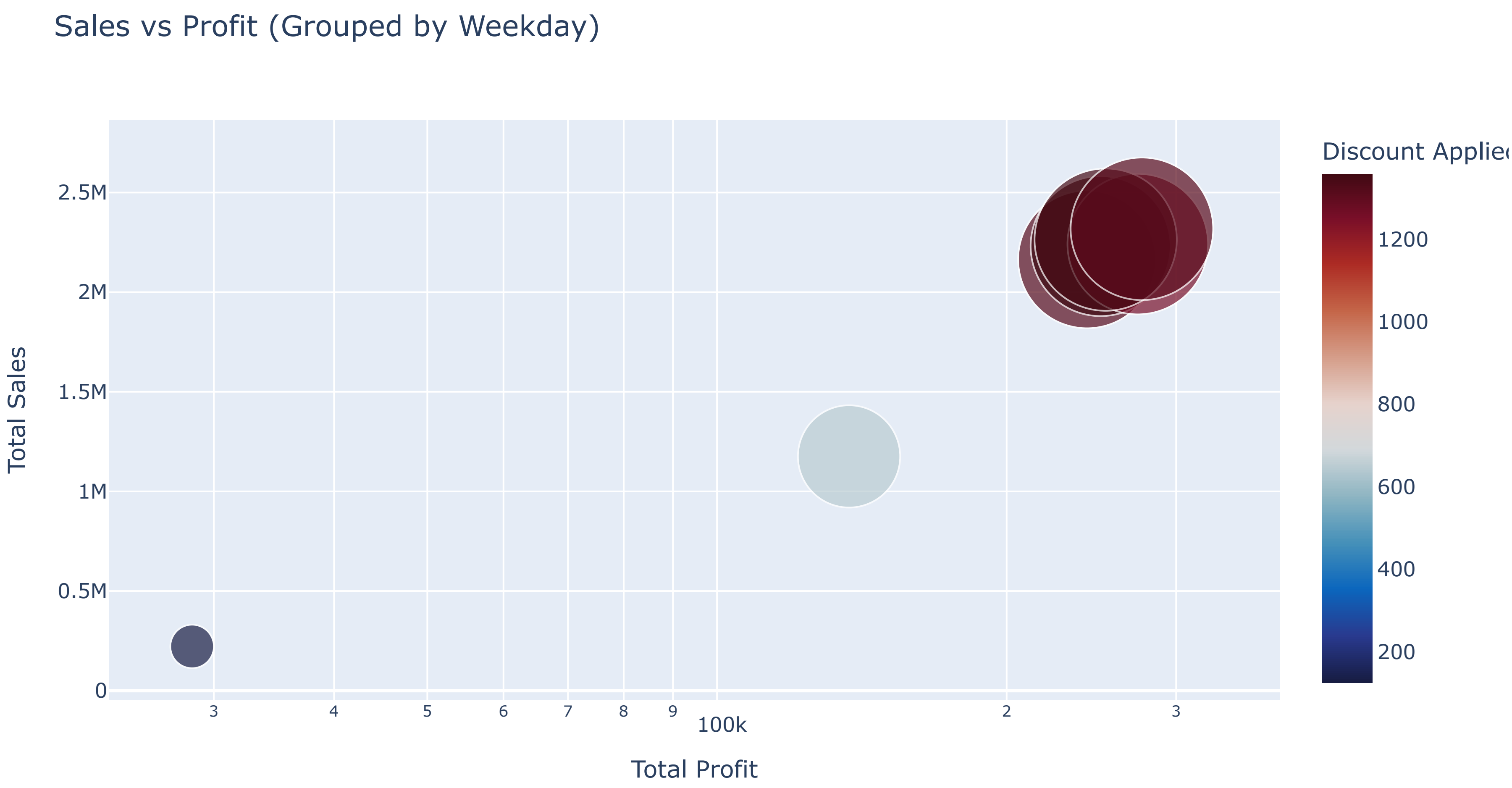
Obeservations

The sales are increasing on yearly basis

```
import pandas as pd
import plotly.express as px
```

```
# Creating a copy of the dataset
weekday = store.copy()

# Converting 'Order Date' to datetime (if not already)
weekday['Order Date'] = pd.to_datetime(weekday['Order Date'])

# Extracting day names
weekday['Day'] = weekday['Order Date'].dt.day_name()

# Aggregating by day
weekday = weekday.groupby('Day').sum(numeric_only=True)

# Sorting by 'Sales'
weekday = weekday.sort_values(by='Sales')

# Creating a scatter plot
fig = px.scatter(
    weekday,
    x="Profit",
    y="Sales",
    size="Shipping Cost",
    color="Discount",
    hover_name=weekday.index,
    log_x=True,
    size_max=60,
    color_continuous_scale=px.colors.cmocean.balance,
    title="Sales vs Profit (Grouped by Weekday)",
    labels={"Sales": "Total Sales", "Profit": "Total Profit", "Discount": "Discount Applied"}
)

# Show the figure
```

```
fig.show()
```



Obsevations

Except weekends the sales and profit made is high

Least profit and sales made is on sunday

Highest profit and sales made is on Friday

```
import pandas as pd
import plotly.express as px

# Function to generate formatted labels (Day_Segment)
def day_segment(x):
    return [str(i[0]) + '_' + str(i[1]) for i in x]

# Function to assign colors based on segment type
def color_segment(x):
    color_map = {'Consumer': '#09CDEF', 'Corporate': '#AB09EF', 'Home Office': '#ABCD09'}
    return [color_map.get(i[1].strip().title(), '#000000') for i in x]

# Function to update legend names
def update_legend(fig, names):
    for i, name in enumerate(names):
        fig.data[i].name = name
    return fig

# Create a copy of the dataset
weekday = store.copy()

# Convert 'Order Date' to datetime if not already
weekday['Order Date'] = pd.to_datetime(weekday['Order Date'])

# Extract day names
weekday['Day'] = weekday['Order Date'].dt.day_name()

# Group by Day & Segment and sum numeric values
weekday = weekday.groupby(['Day', 'Segment']).sum(numeric_only=True)
```

```
# Sort by Sales for better visualization
weekday = weekday.sort_values(by='Sales')

# Create scatter plot
fig = px.scatter(
    weekday,
    x="Profit",
    y="Sales",
    size="Shipping Cost",
    color=color_segment(weekday.index),
    hover_name=day_segment(weekday.index),
    log_x=True,
    size_max=60,
    title="Sales vs Profit by Day and Segment",
    labels={"Sales": "Total Sales", "Profit": "Total Profit"}
)
```

```
<ipython-input-51-3d224f623ab6>:29: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=Fa
```

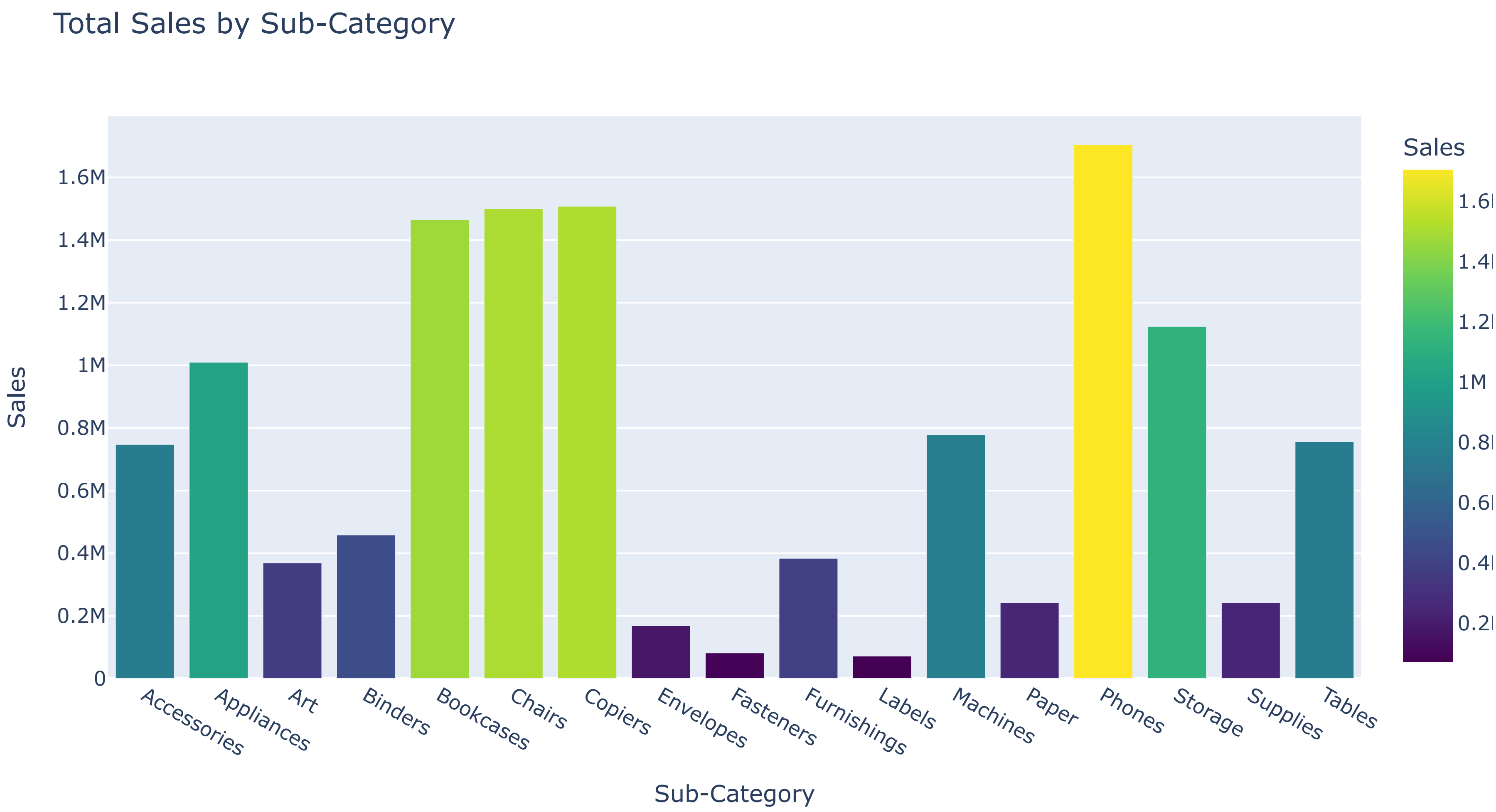
Observations

Consumer segment purchases more than corporate and home office and is really profitable

Home Office segment is least profitable

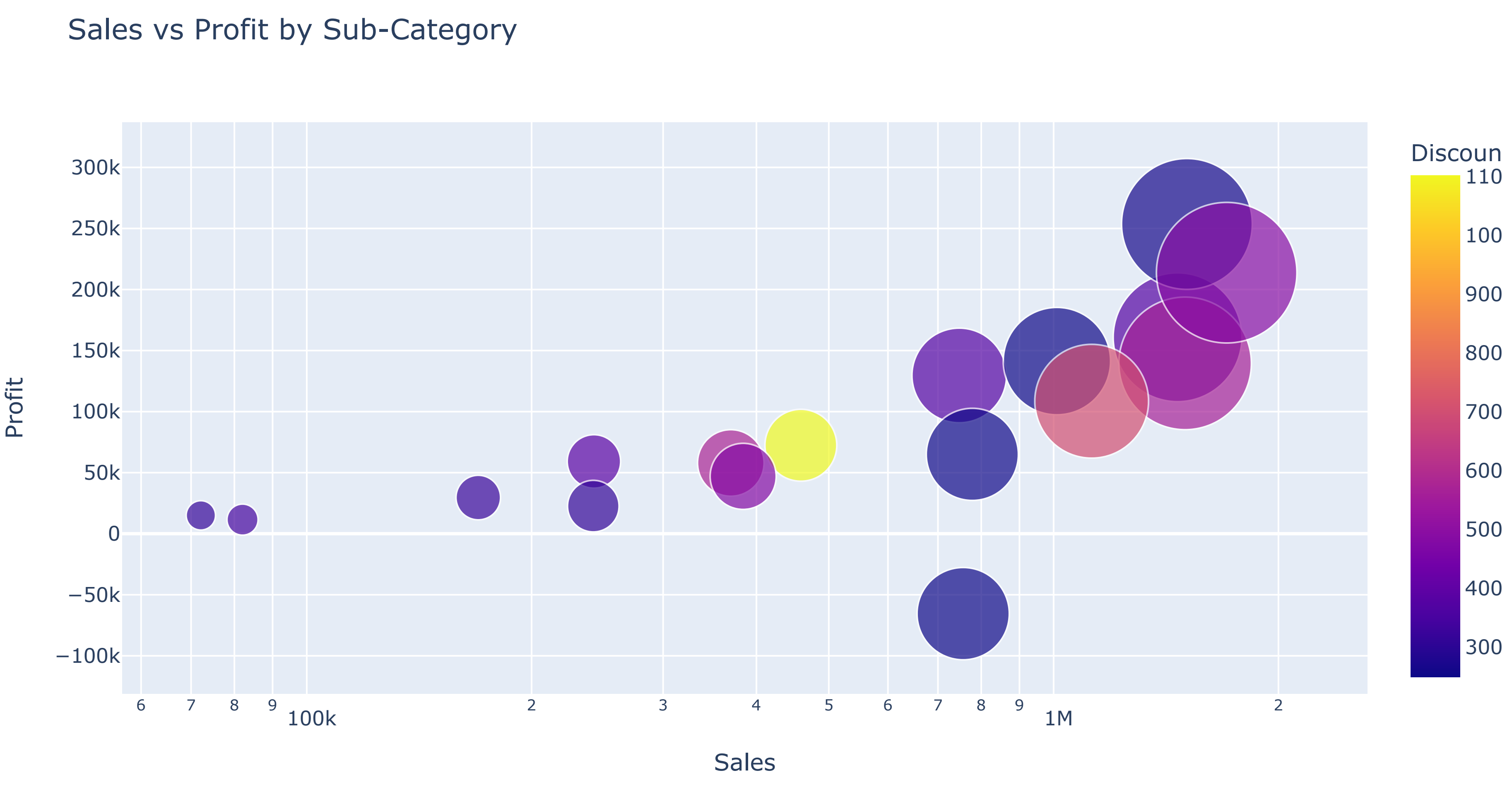
```
import plotly.express as px

# Create a bar chart to visualize sales by sub-category
```

```
fig = px.bar(
    categ_sales,
    x=categ_sales.index,
    y="Sales",
    title="Total Sales by Sub-Category",
    labels={"x": "Sub-Category", "y": "Sales"},
    color="Sales",
    color_continuous_scale=px.colors.sequential.Viridis
)

fig.show()
```



```
import plotly.express as px

# Ensure categ_sales only contains numeric columns
categ_sales = store.groupby('Sub-Category').sum(numeric_only=True)

# Create scatter plot
fig = px.scatter(
    categ_sales,
    categ_sales,
    x="Sales",
    y="Profit",
    size="Shipping Cost",
    color="Discount",
    hover_name=categ_sales.index,
    log_x=True,
    size_max=60,
    title="Sales vs Profit by Sub-Category"
)

fig.show()
```

```
<ipython-input-54-e94601f2d372>:4: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=Fa
```



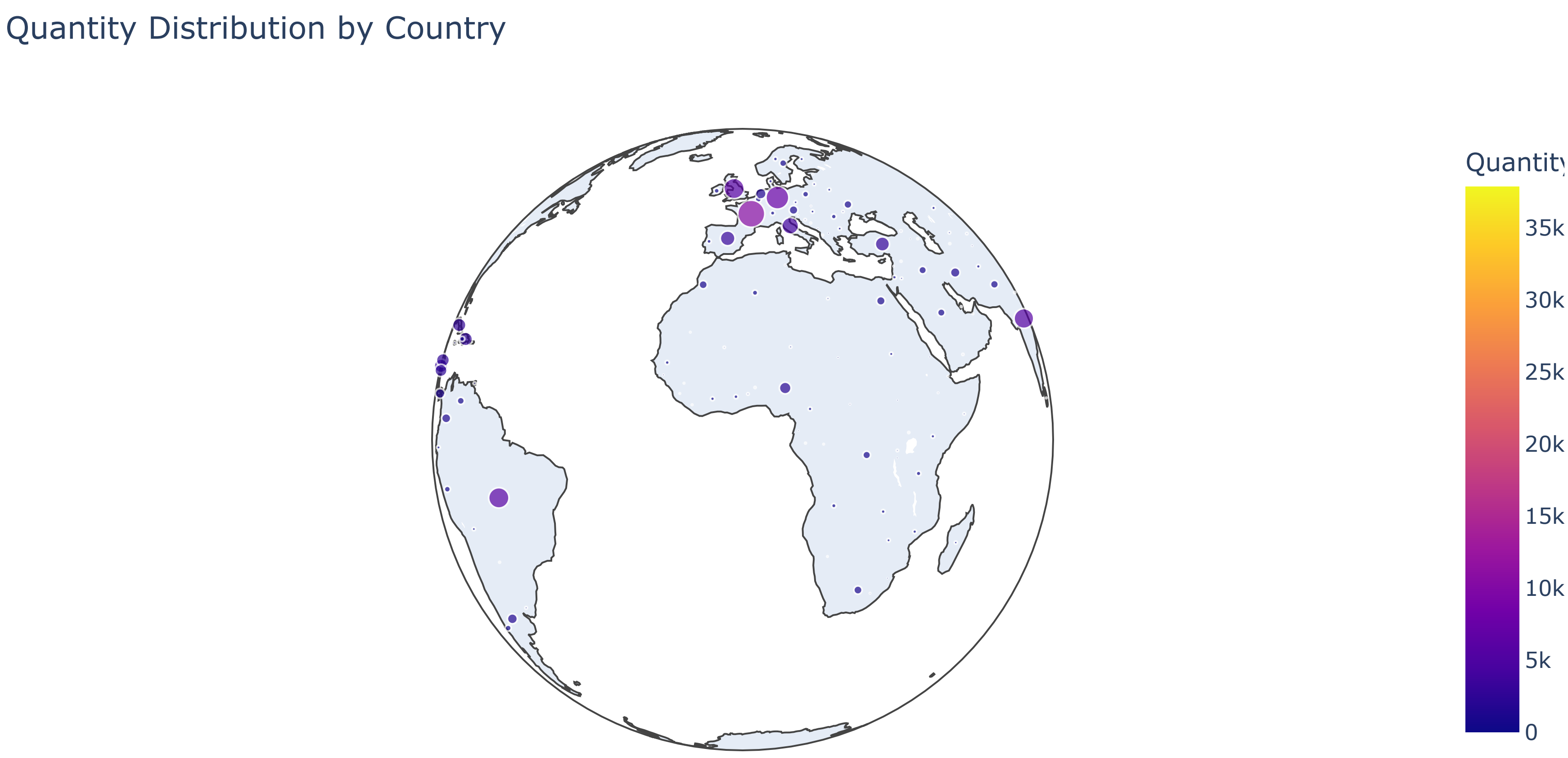
Observations

Copier are 2nd highest selling Sub-Category but makes most of the profit

Tables generate loss in general

```
import plotly.express as px

# Ensure country_profit only contains numeric columns
country_profit = store.groupby('Country').sum(numeric_only=True)

# Create a geographical scatter plot
fig = px.scatter_geo(
    country_profit,
    locations=country_profit.index,
    hover_name=country_profit.index,
    locationmode='country names',
    size=country_profit["Quantity"],
    color=country_profit["Quantity"],
    projection='orthographic',
    title="Quantity Distribution by Country"
)

fig.show()
```

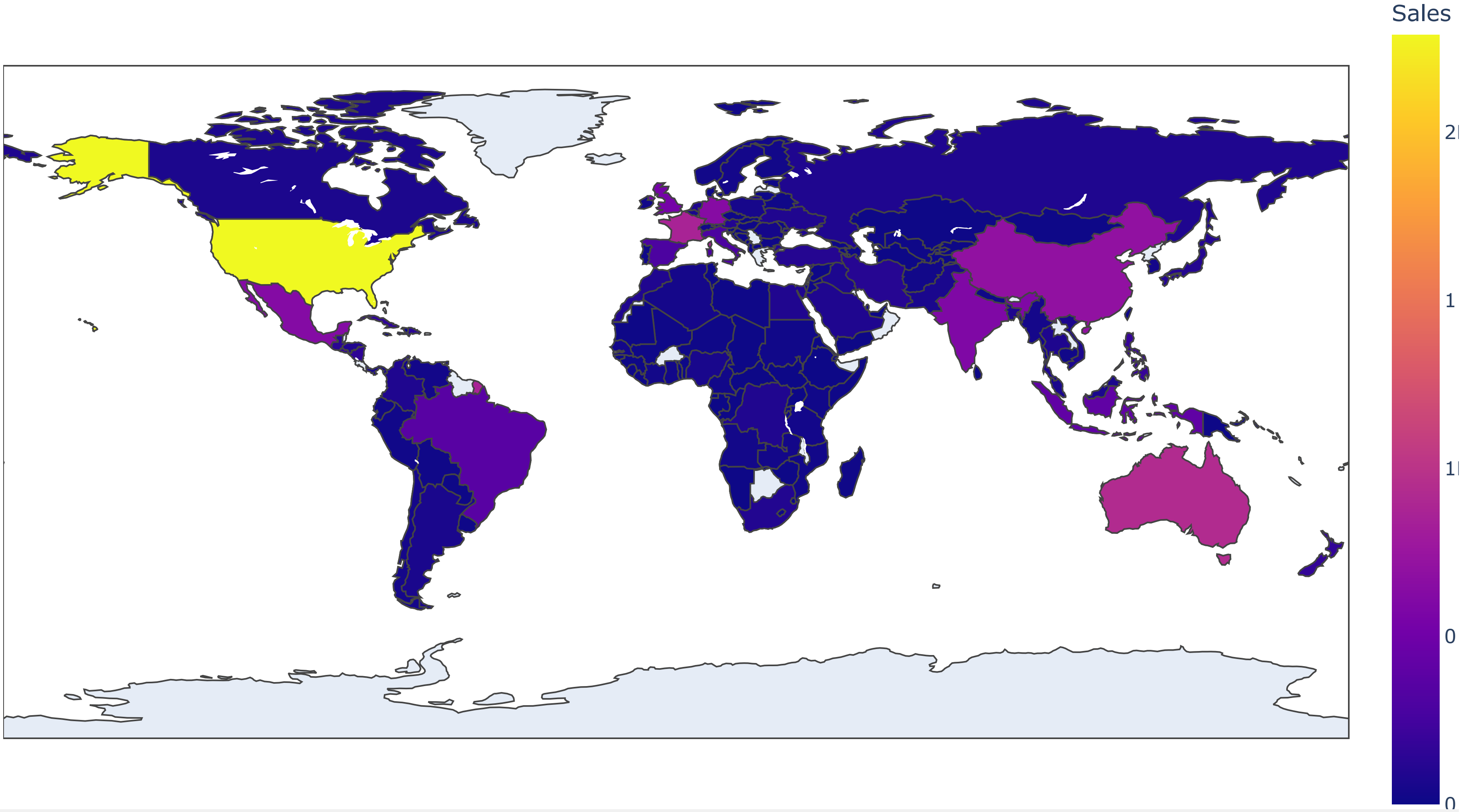
```
<ipython-input-57-7d8e43c902eb>:4: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=Fa
```



Observations

Highest average quantity bought is from slovenia

```
fig = px.choropleth(country_profit, color="Sales",locationmode='country names',
                    locations=country_profit.index,)
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



Observations

Except Chad, most of the countries make sales less than 400K

Training and Testing Set

```
#Dropping unnecessary columns
drop=['Row ID','Order ID','Order Date','Ship Date',
      'Customer ID','Customer Name','Postal Code','Product ID',
     'Product Name']
mean_filled_data.drop(drop,axis=1,inplace=True)
```

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target variable (y)
mX = mean_filled_data.drop('Sales', axis=1)
my = mean_filled_data["Sales"]
```