Data Wrangling

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import matplotlib.cm as cm

import warnings
warnings.filterwarnings('ignore') # Ignore Warnings

pd.options.display.max_columns= None # display max Columns
pd.options.display.max_rows= None # display max Rows
```

In [ ]:
```
df=pd.read_csv('Sample - Superstore.csv',encoding='latin-1') ## read the data
```

In [ ]:
```
df.head()
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	Stat
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentuck
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentuck
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	Californ
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Floric
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Floric

In [ ]:
```
df.tail()
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	
9989	9990	CA-2014-110422	1/21/2014	1/23/2014	Second Class	TB-21400	Tom Boeckenhauer	Consumer	United States	Miami	
9990	9991	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	C
9991	9992	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	C
9992	9993	CA-2017-121258	2/26/2017	3/3/2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa Mesa	C
9993	9994	CA-2017-119914	5/4/2017	5/9/2017	Second Class	CC-12220	Chris Cortes	Consumer	United States	Westminster	C

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 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   Customer Name  9994 non-null   object 
 7   Segment        9994 non-null   object 
 8   Country        9994 non-null   object 
 9   City           9994 non-null   object 
 10  State          9994 non-null   object 
 11  Postal Code    9994 non-null   int64  
 12  Region         9994 non-null   object 
 13  Product ID     9994 non-null   object 
 14  Category       9994 non-null   object 
 15  Sub-Category   9994 non-null   object 
 16  Product Name   9994 non-null   object 
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64  
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country          0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

In [ ]:
```
df[df.duplicated()] # Check duplicated
```

Out[ ]:

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code	Region	Produ

convert to datatime

1. Order Date 9994 non-null object
2. Ship Date 9994 non-null object

In [ ]:
```
date_format = "%m/%d/%Y"
df['Order Date']=pd.to_datetime(df['Order Date'], format=date_format)
df['Ship Date']= pd.to_datetime( df['Ship Date'], format=date_format)
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 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   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
```

In [ ]:
```
nummeric_df= df[['Sales','Quantity','Discount','Profit']]
```

In [ ]:
```
nummeric_df.describe().transpose()
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Sales	9994.0	229.858001	623.245101	0.444	17.28000	54.4900	209.940	22638.480
Quantity	9994.0	3.789574	2.225110	1.000	2.00000	3.0000	5.000	14.000
Discount	9994.0	0.156203	0.206452	0.000	0.00000	0.2000	0.200	0.800
Profit	9994.0	28.656896	234.260108	-6599.978	1.72875	8.6665	29.364	8399.976

In [ ]:
```
df_categorical=df.drop(['Sales','Quantity','Discount','Profit','Row ID','Order ID','Orde
df_categorical.head()
```

Out[ ]:

	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Region	Product ID	Category	Cate
0	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-BO-10001798	Furniture	Booko
1	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	South	FUR-CH-10000454	Furniture	C
2	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	West	OFF-LA-10000240	Office Supplies	L
3	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	South	FUR-TA-10000577	Furniture	T
4	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	South	OFF-ST-10000760	Office Supplies	Sto

In [ ]:
```
num_unique= [(col ,df_categorical[col].nunique()) for col in df_categorical.columns]
num_unique
```

Out[ ]:
```
[('Ship Mode', 4),
 ('Customer ID', 793),
 ('Customer Name', 793),
 ('Segment', 3),
 ('Country', 1),
 ('City', 531),
 ('State', 49),
 ('Region', 4),
 ('Product ID', 1862),
 ('Category', 3),
 ('Sub-Category', 17),
 ('Product Name', 1850)]
```

In [ ]:
```
df['Order Date'].nunique()
```

Out[ ]:
```
1237
```

In [ ]:
```
df['Ship Date'].nunique()
```

Out[ ]:
```
1334
```

EDA

In this topic i need to answer this questions

1. What are the top selling products in the Super Store?

In [ ]:
```
df.head()
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	California	90036
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	Florida	33311

In [ ]:
```
product_group= df.groupby(['Product Name']).sum()['Sales']
```

In [ ]:
```
top_selling= product_group.sort_values(ascending=False)
top_selling.head(10)
```

Out[ ]:
```
Product Name
Canon imageCLASS 2200 Advanced Copier                                  61599.824
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind    27453.384
Cisco TelePresence System EX90 Videoconferencing Unit                  22638.480
HON 5400 Series Task Chairs for Big and Tall                           21870.576
GBC DocuBind TL300 Electric Binding System                             19823.479
GBC Ibimaster 500 Manual ProClick Binding System                       19024.500
Hewlett Packard LaserJet 3310 Copier                                   18839.686
HP Designjet T520 Inkjet Large Format Printer - 24" Color              18374.895
GBC DocuBind P400 Electric Binding System                              17965.068
High Speed Automatic Electric Letter Opener                            17030.312
Name: Sales, dtype: float64
```

In [ ]:
```
top_5_selling_prod= pd.DataFrame(top_selling[:10])
top_5_selling_prod.head()
```

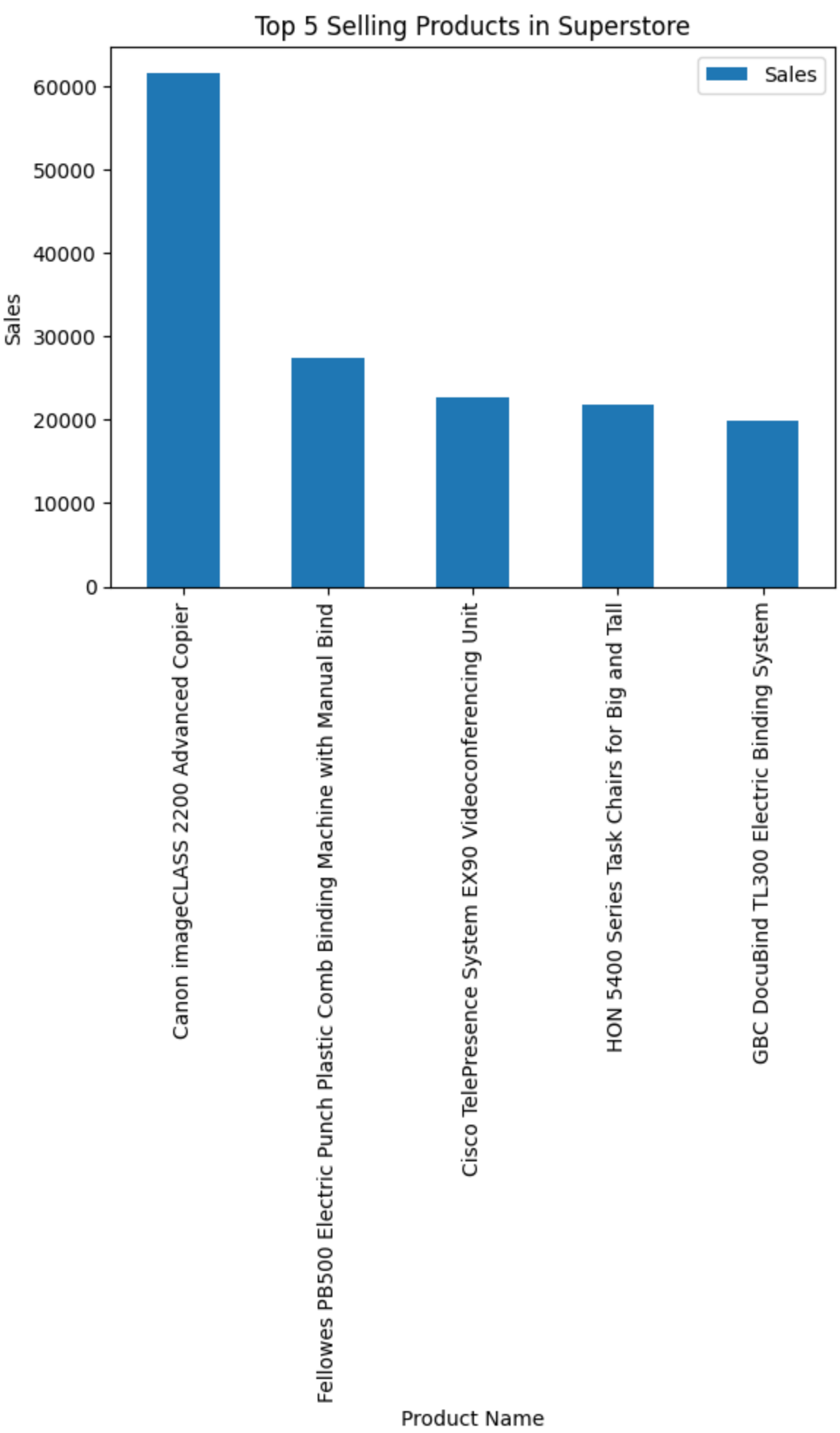
Out[ ]:

	Sales
Product Name	
Canon imageCLASS 2200 Advanced Copier	61599.824
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind	27453.384
Cisco TelePresence System EX90 Videoconferencing Unit	22638.480
HON 5400 Series Task Chairs for Big and Tall	21870.576
GBC DocuBind TL300 Electric Binding System	19823.479

In [ ]:
```
top_5_selling_prod = top_5_selling_prod.sort_values(by='Sales', ascending=False).head()
```

In [ ]:
```
plt.figure(figsize=(20, 10))
top_5_selling_prod.plot(kind='bar')
plt.xlabel('Product Name')
plt.ylabel('Sales')
plt.title('Top 5 Selling Products in Superstore')
plt.show()
```

```
<Figure size 2000x1000 with 0 Axes>
```



Sales Trend Over time (Yearly, Monthly)

1. **Trends by Months**

- After 2015 the sales start to incrase

In [ ]:
```
df['Order Date'].dt.month.value_counts()
```

Out[ ]:
```
11    1471
12    1408
9     1383
10     819
5      735
6      717
7      710
8      706
3      696
4      668
1      381
2      300
Name: Order Date, dtype: int64
```

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

# Group by 'Order Date' and calculate the sum of 'Sales' for each month
monthly_sales = df.groupby(pd.Grouper(key='Order Date', freq='M')).sum()

# Reset the index to make 'Order Date' a regular column
monthly_sales = monthly_sales.reset_index()

monthly_sales.head()
```

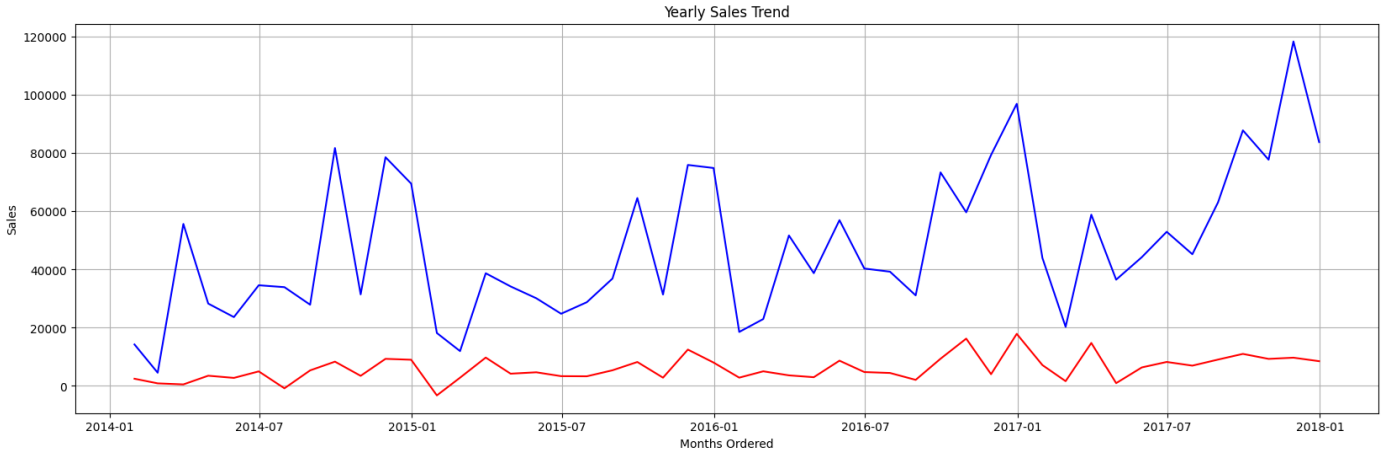
Out[ ]:

	Order Date	Row ID	Postal Code	Sales	Quantity	Discount	Profit
0	2014-01-31	346176	4342297	14236.895	284	10.00	2450.1907
1	2014-02-28	272235	3049775	4519.892	159	8.10	862.3084
2	2014-03-31	871575	8314538	55691.009	585	26.30	498.7299
3	2014-04-30	690625	7723827	28295.345	536	14.85	3488.8352
4	2014-05-31	611708	6497804	23648.287	466	18.95	2738.7096

- Observations:
- There is variation in sales throughout the year, with some months having higher sales and others having lower sales.
- The highest sales were observed in November (Month 11) with $352,461.07,
- the second-highest sales were in September (Month 9) with $307,649.95.
- The lowest sales were observed in February (Month 2) with $59,751.25.

In [ ]:
```
# Plot yearly sales trend
plt.figure(figsize=(20, 6))
plt.plot(monthly_sales['Order Date'], monthly_sales['Sales'], color='blue')
plt.plot(monthly_sales['Order Date'], monthly_sales['Profit'], color='red')
                                    Months Ordered')
```

```
plt.ylabel('Sales')
plt.title('Yearly Sales Trend')
plt.grid(True)
plt.show()
```



1. **Trends by Years**

In [ ]:
```
df['Order Date'].dt.year.value_counts()
```

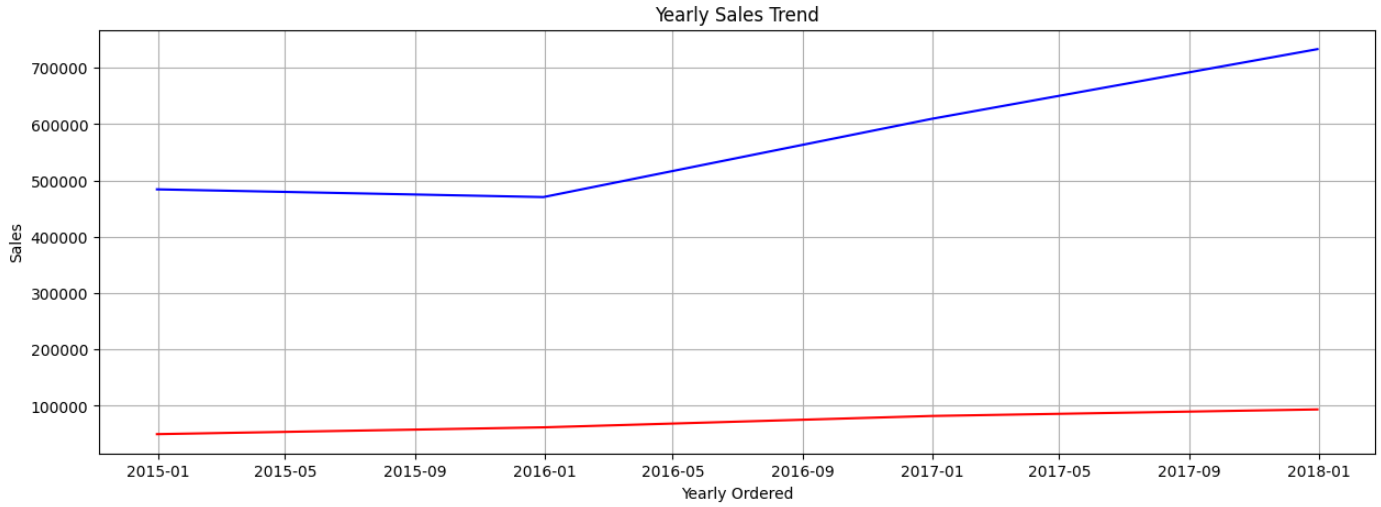
Out[ ]:
```
2017    3312
2016    2587
2015    2102
2014    1993
Name: Order Date, dtype: int64
```

In [ ]:
```
# Group by 'Order Date' and calculate the sum of 'Sales' for each month
yealy_sales = df.groupby(pd.Grouper(key='Order Date', freq='Y')).sum()

# Reset the index to make 'Order Date' a regular column
yealy_sales = yealy_sales.reset_index()

yealy_sales.head()
```

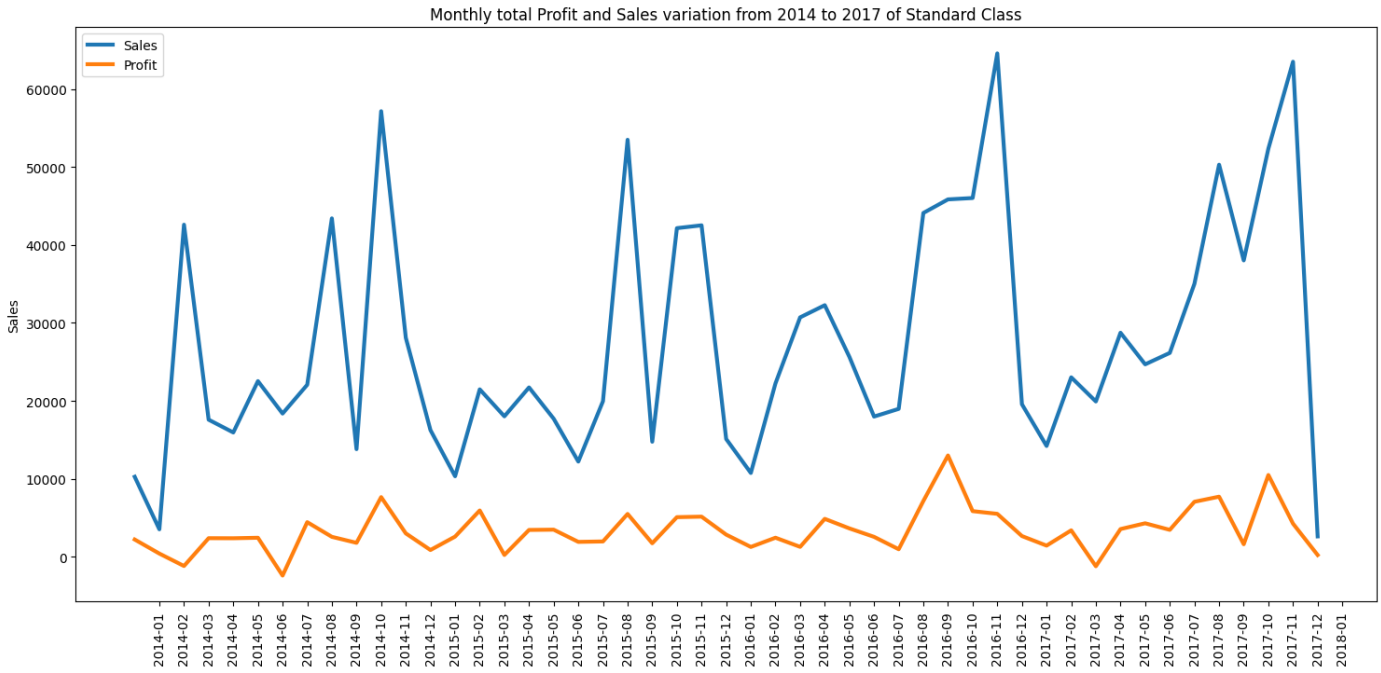
Out[ ]:

	Order Date	Row ID	Postal Code	Sales	Quantity	Discount	Profit
0	2014-12-31	9904015	113271247	484247.4981	7581	315.46	49543.9741
1	2015-12-31	10413696	111208247	470532.5090	7979	327.09	61618.6037
2	2016-12-31	12778804	141003420	609205.5980	9837	400.32	81795.1743
3	2017-12-31	16848500	186089738	733215.2552	12476	518.22	93439.2696

In [ ]:
```
# Plot yearly sales trend
plt.figure(figsize=(15, 5))
plt.plot(yealy_sales['Order Date'], yealy_sales['Sales'],color='b')
plt.plot(yealy_sales['Order Date'], yealy_sales['Profit'],color='r')
plt.xlabel('Yearly Ordered')
plt.ylabel('Sales')
plt.title('Yearly Sales Trend')
plt.grid(True)
plt.show()
```



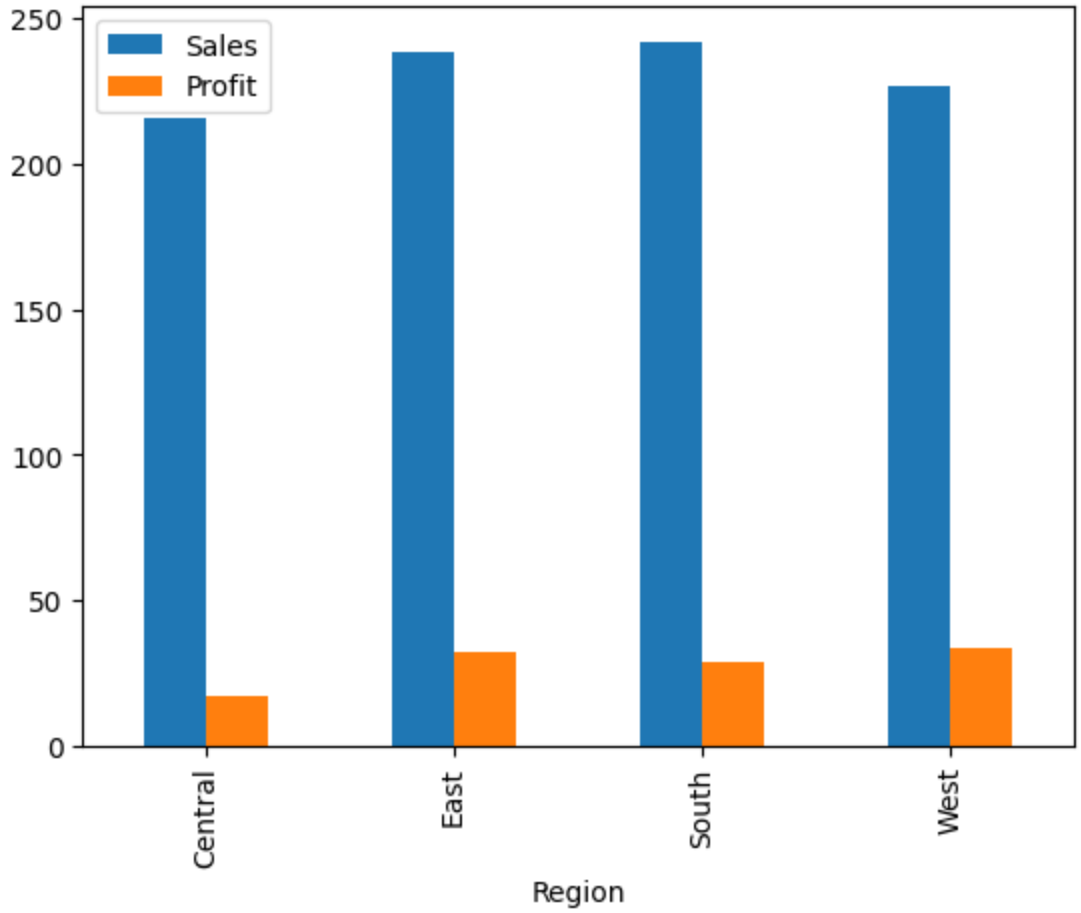
In [ ]:
```
stndclz_df = df.loc[df['Ship Mode'] == 'Standard Class']
stndclz_df = stndclz_df[['Ship Date', 'Sales', 'Profit']].sort_values(by='Ship Date')
stndclz_df['monthyr'] = pd.to_datetime(stndclz_df['Ship Date']).dt.to_period('M')
stndclz_df = stndclz_df.groupby('monthyr').agg({'Sales':'sum', 'Profit':'sum'}).reset_in

plt.figure(figsize=(18,8))
sns.lineplot(x = stndclz_df.index, y = 'Sales', data = stndclz_df, label = 'Sales', line
sns.lineplot(x = stndclz_df.index, y = 'Profit', data = stndclz_df, label = 'Profit', li

labels = stndclz_df['monthyr'].values
plt.xticks(range(1,stndclz_df.shape[0]+1), labels=labels)
plt.xticks(rotation=90)
plt.title('Monthly total Profit and Sales variation from 2014 to 2017 of Standard Class'
plt.show()
```



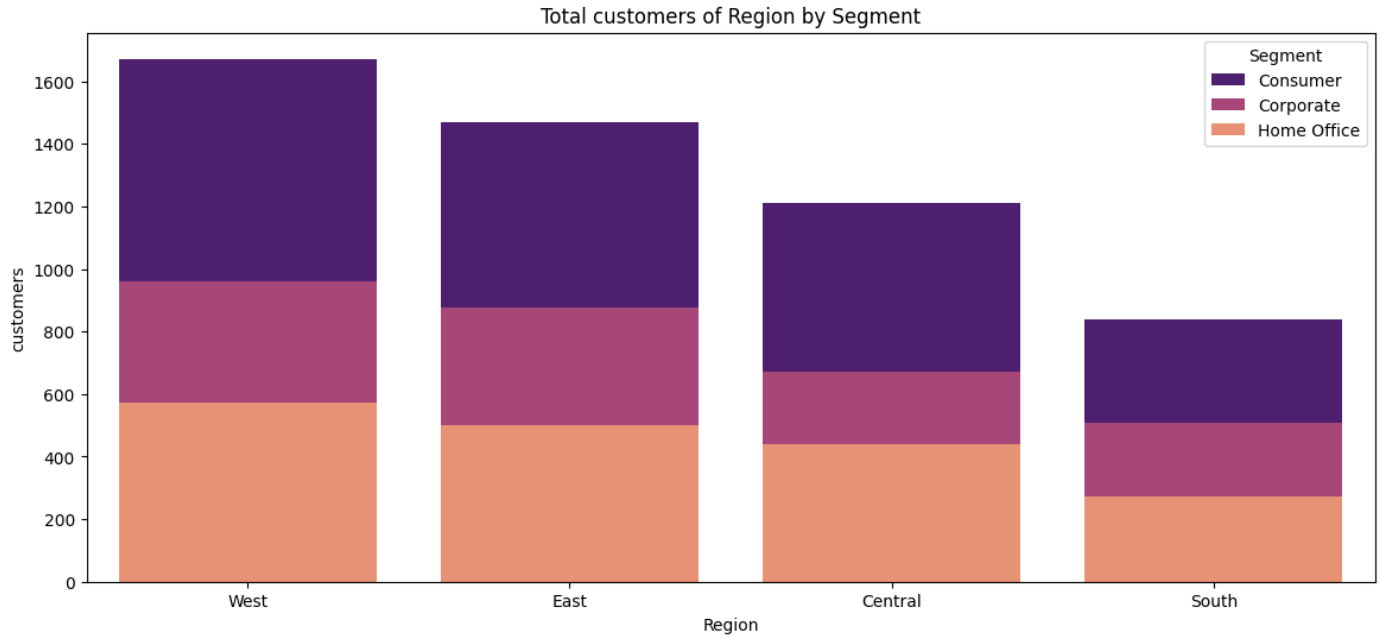
The trend suggests that the yearly sales had a slow start in 2014, but they started to increase from 2015 onwards, and the increase continued until 2017 in a linear fashion.

In [ ]:
```
region_group= df.groupby(['Region']).mean()[['Sales','Profit']]

region_group.plot(kind='bar')
```

Out[ ]:
```
<Axes: xlabel='Region'>
```

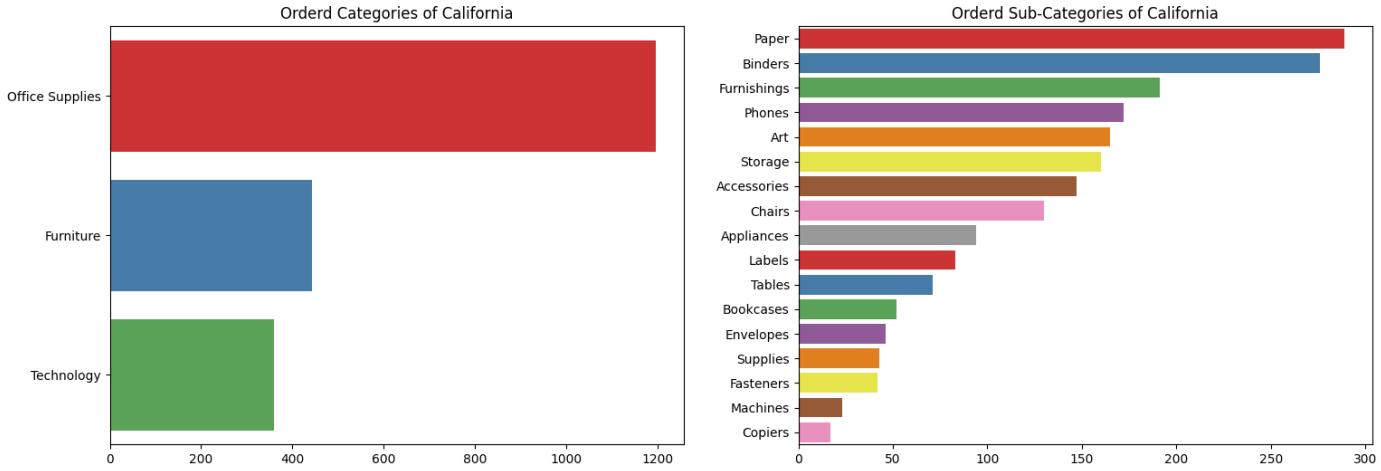


In [ ]:
```
plt.figure(figsize=(14,6))
region_df = df.groupby(['Region', 'Segment']).size().reset_index().rename(columns= {0:'c
region_df.pivot(columns = 'Region', index = 'Segment', values = 'customers')
sns.barplot(x='Region', y='customers', data=region_df, palette='magma', hue='Segment',do
plt.title("Total customers of Region by Segment")
plt.show()
```



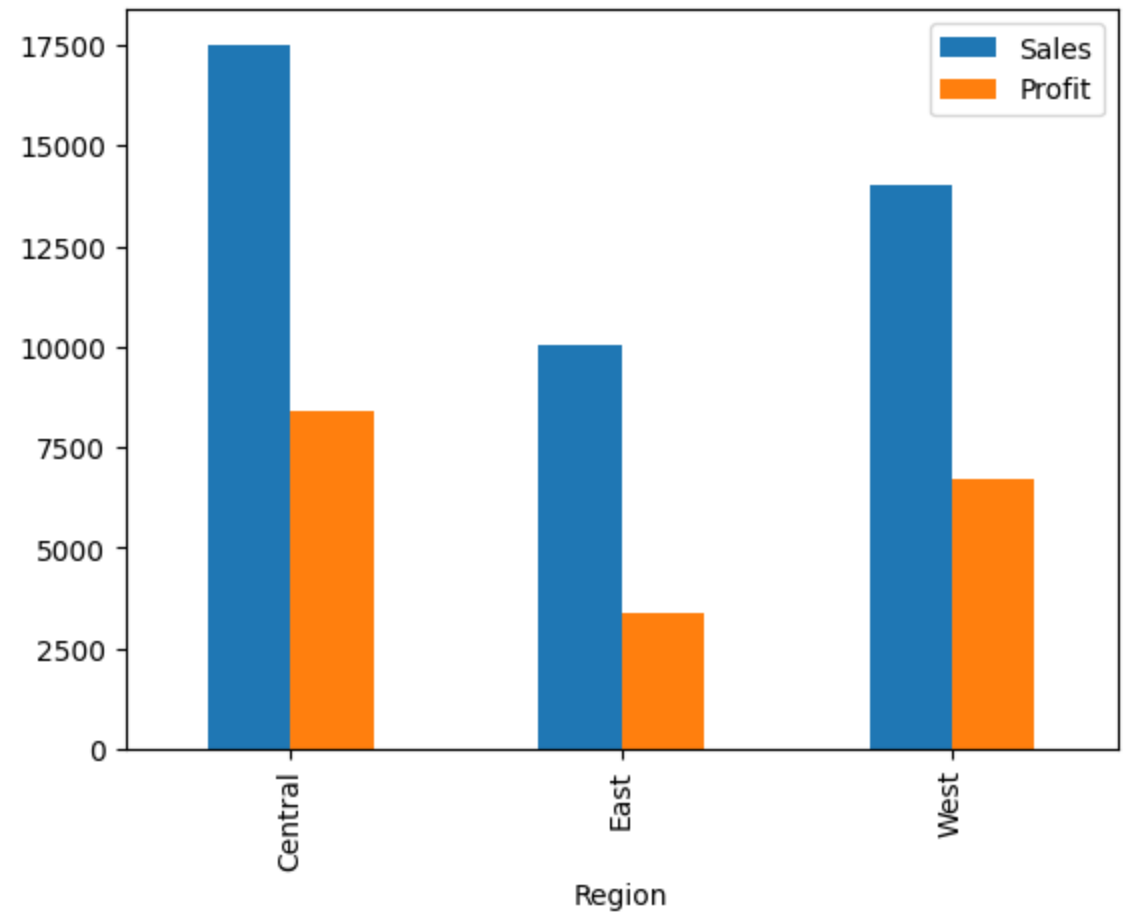
In [ ]:
```
california_df = df.loc[df['State'] == 'California']

plt.figure(figsize=(18,6))
plt.subplot(1,2,2)
sns.barplot(x = california_df['Sub-Category'].value_counts().values,
            y = california_df['Sub-Category'].value_counts().index, palette='Set1')
```

```
plt.title('Orderd Sub-Categories of California')

plt.subplot(1,2,1)
sns.barplot(x = california_df['Category'].value_counts().values,
            y = california_df['Category'].value_counts().index, palette='Set1')
plt.title('Orderd Categories of California')

plt.show()
```



In [ ]:
```
Canon_img= df[df['Product Name'] == 'Canon imageCLASS 2200 Advanced Copier']

Canon_img = Canon_img.groupby(['Region']).mean()[['Sales','Profit']]

Canon_img.plot(kind='bar')
```

Out[ ]:
```
<Axes: xlabel='Region'>
```



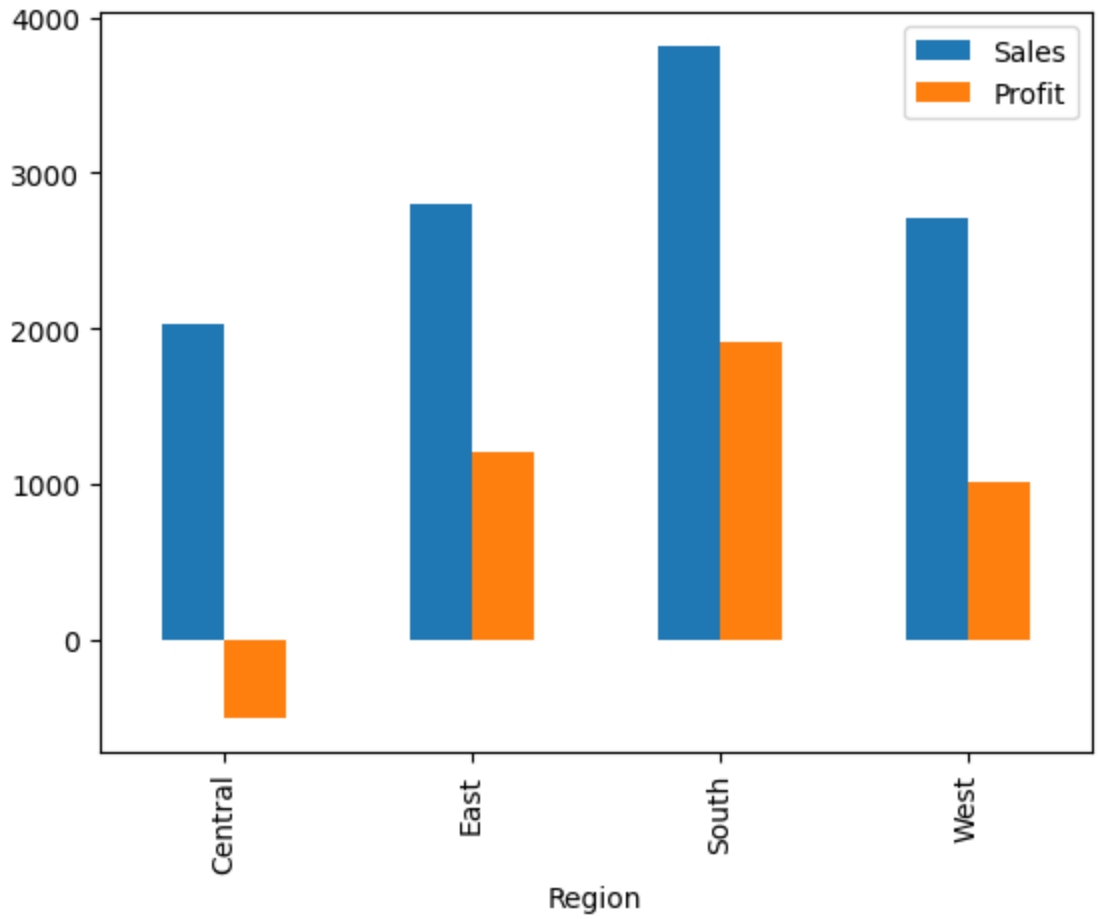
In [ ]:
```
Canon_img.head()
```

Out[ ]:

Region	Sales	Profit
Central	17499.950000	8399.9760
East	10033.304667	3359.9904
West	13999.960000	6719.9808

In [ ]:
```
# Filter the DataFrame to select rows with 'Product Name' equal to 'Fellowes PB500 Elect
Fellowes_PB= df[df['Product Name'] == 'Fellowes PB500 Electric Punch Plastic Comb Bindin

# Group by 'Region' and calculate the mean of 'Sales' and 'Profit'
Fellowes_PB = Fellowes_PB.groupby(['Region']).mean()[['Sales', 'Profit']]

Fellowes_PB.plot(kind='bar')
```

Out[ ]:
```
<Axes: xlabel='Region'>
```



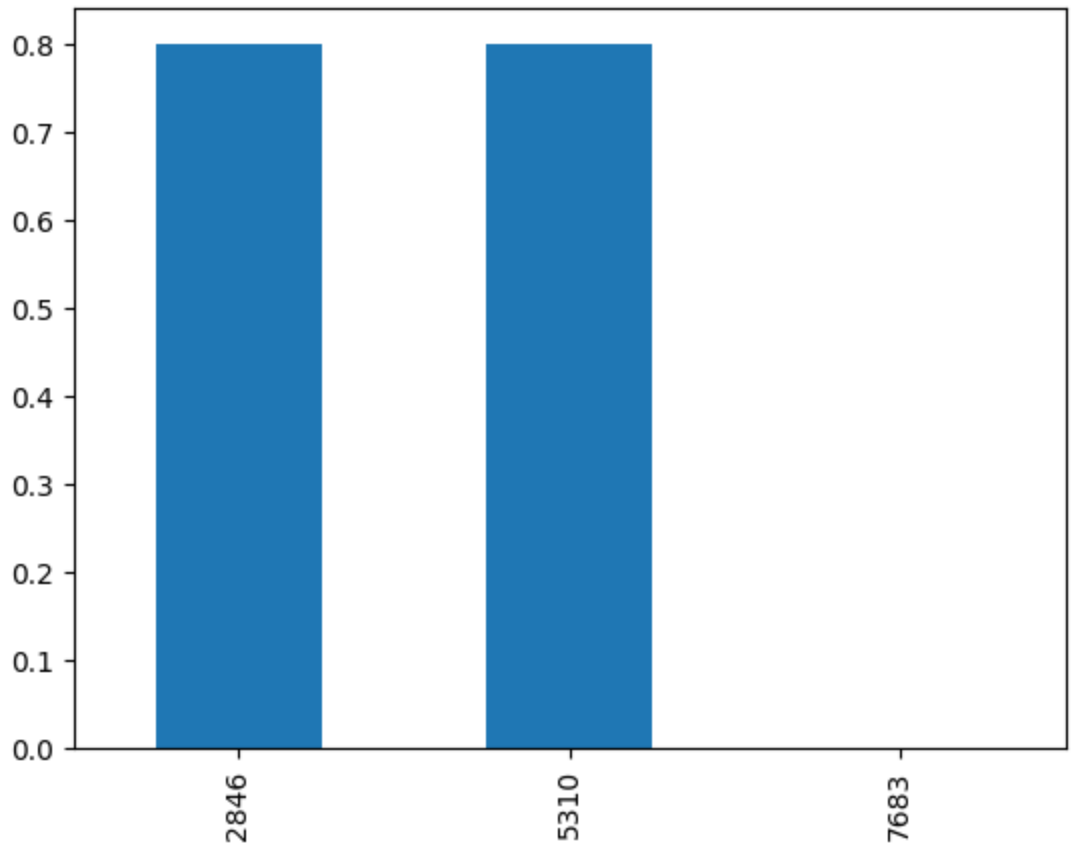
In [ ]:
```
Fellowes_filter = df[(df['Product Name'] == 'Fellowes PB500 Electric Punch Plastic Comb
Fellowes_filter
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Posta Code
2846	2847	CA-2017-152093	2017-09-10	2017-09-15	Standard Class	SN-20560	Skye Norling	Home Office	United States	Chicago	Illinois	60653
5310	5311	CA-2017-131254	2017-11-19	2017-11-21	First Class	NC-18415	Nathan Cano	Consumer	United States	Houston	Texas	77095
7683	7684	CA-2015-120782	2015-04-28	2015-05-01	First Class	SD-20485	Shirley Daniels	Home Office	United States	Midland	Michigan	48640

In [ ]:
```
Fellowes_filter['Discount'].plot(kind='bar')
```

Out[ ]:
```
<Axes: >
```



Which region Genrates the most sales

In [ ]:
```
df_place= df[['Country','City','State','Region']]
```

In [ ]:
```
for place in df_place.columns:

  print(place, ':' , df_place[place].nunique())
```

```
Country : 1
City : 531
State : 49
Region : 4
```

- Compare between
  - which city sales and profits more than other cities
  - which region's sales and profits are more than other regions
  - which state sales and profits are more than other states

In [ ]:
```
# add for df_places sales and profit
df_place=df[['Country','City','State','Region','Sales','Profit']]

df_place.head()
```

Out[ ]:

	Country	City	State	Region	Sales	Profit
0	United States	Henderson	Kentucky	South	261.9600	41.9136
1	United States	Henderson	Kentucky	South	731.9400	219.5820
2	United States	Los Angeles	California	West	14.6200	6.8714
3	United States	Fort Lauderdale	Florida	South	957.5775	-383.0310
4	United States	Fort Lauderdale	Florida	South	22.3680	2.5164

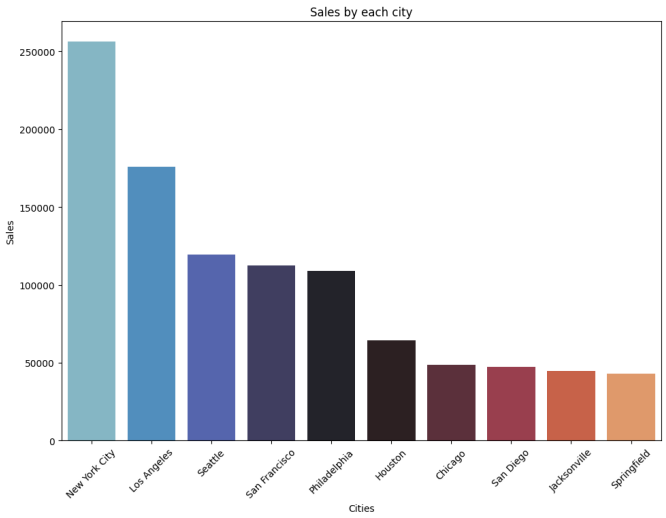
City

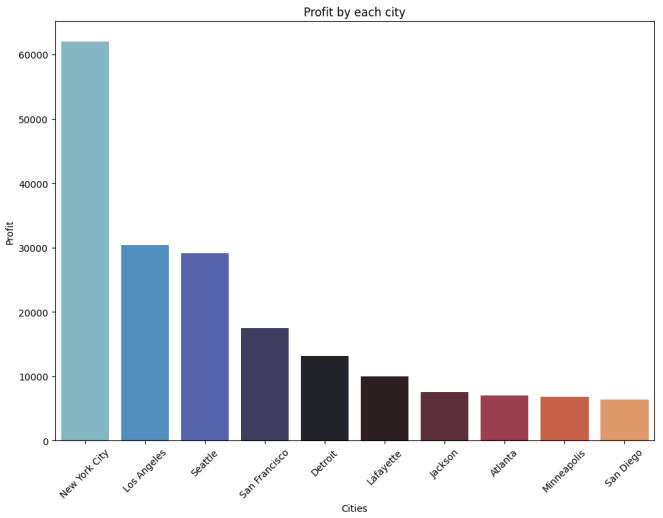
In [ ]:
```
# sort by most sales

city_df= df_place.groupby(['City'], as_index=False ).sum()
city_df_sales= city_df.sort_values(by='Sales', ascending=False)
city_sales=city_df_sales.head(10)
```

In [ ]:
```
# sort by most Profit

city_df_profit= city_df.sort_values(by='Profit', ascending=False)
city_profit=city_df_profit.head(10)
```

In [ ]:
```
# visualize both to combare between them
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(25, 8))

# for sales
sns.barplot(data=city_sales, x='City', y='Sales', palette='icefire', ax=ax1)
ax1.set_xlabel('Cities')
ax1.set_ylabel('Sales')
ax1.set_title('Sales by each city')
ax1.tick_params(axis='x', rotation=45)  # Rotate x-axis labels

# for profit
sns.barplot(data=city_profit, x='City', y='Profit', palette='icefire', ax=ax2)
ax2.set_xlabel('Cities')
ax2.set_ylabel('Profit')
ax2.set_title('Profit by each city')
ax2.tick_params(axis='x', rotation=45)  # Rotate x-axis labels
```

```
plt.show()
```



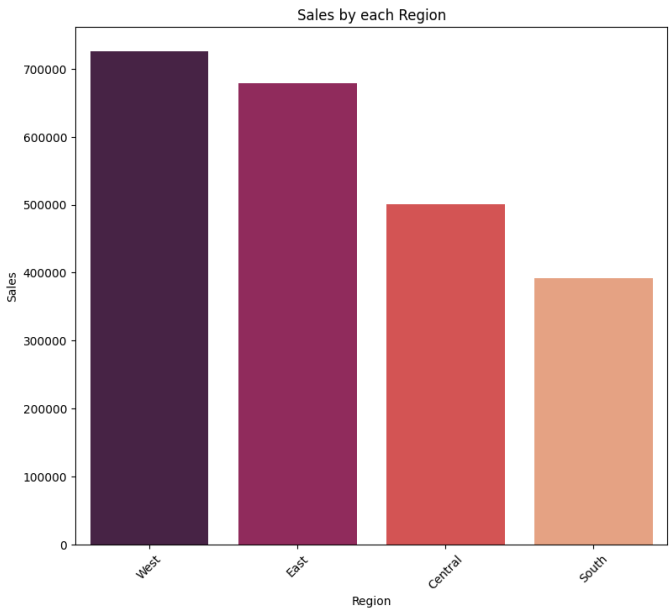


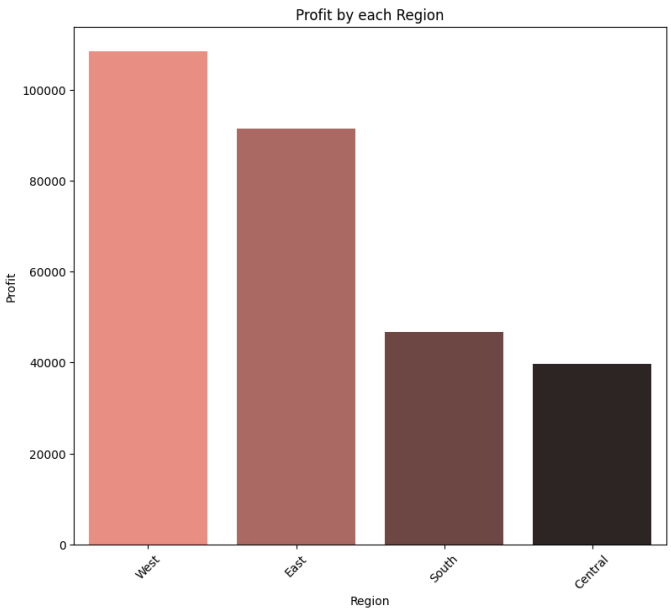
Region

In [ ]:
```
region_grouped= df_place.groupby(['Region'], as_index=False).sum()
region_sales= region_grouped.sort_values(by='Sales' ,ascending=False)
region_profit= region_grouped.sort_values(by='Profit' ,ascending=False)
print(region_sales)
print('='*50)
print(region_profit)
```

```
    Region        Sales       Profit
3     West  725457.8245  108418.4489
1     East  678781.2400   91522.7800
0  Central  501239.8908   39706.3625
2    South  391721.9050   46749.4303
==================================================
    Region        Sales       Profit
3     West  725457.8245  108418.4489
1     East  678781.2400   91522.7800
2    South  391721.9050   46749.4303
0  Central  501239.8908   39706.3625
```

In [ ]:
```
# visualize both to combare between them
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

# for sales
sns.barplot(data=region_sales, x='Region', y='Sales', palette='rocket', ax=ax1)
ax1.set_xlabel('Region')
ax1.set_ylabel('Sales')
ax1.set_title('Sales by each Region')
ax1.tick_params(axis='x', rotation=45)  # Rotate x-axis labels

# for profit
sns.barplot(data=region_profit, x='Region', y='Profit', palette='dark:salmon_r', ax=ax2)
ax2.set_xlabel('Region')
ax2.set_ylabel('Profit')
ax2.set_title('Profit by each Region')
ax2.tick_params(axis='x', rotation=45)  # Rotate x-axis labels

plt.show()
```





In [ ]:

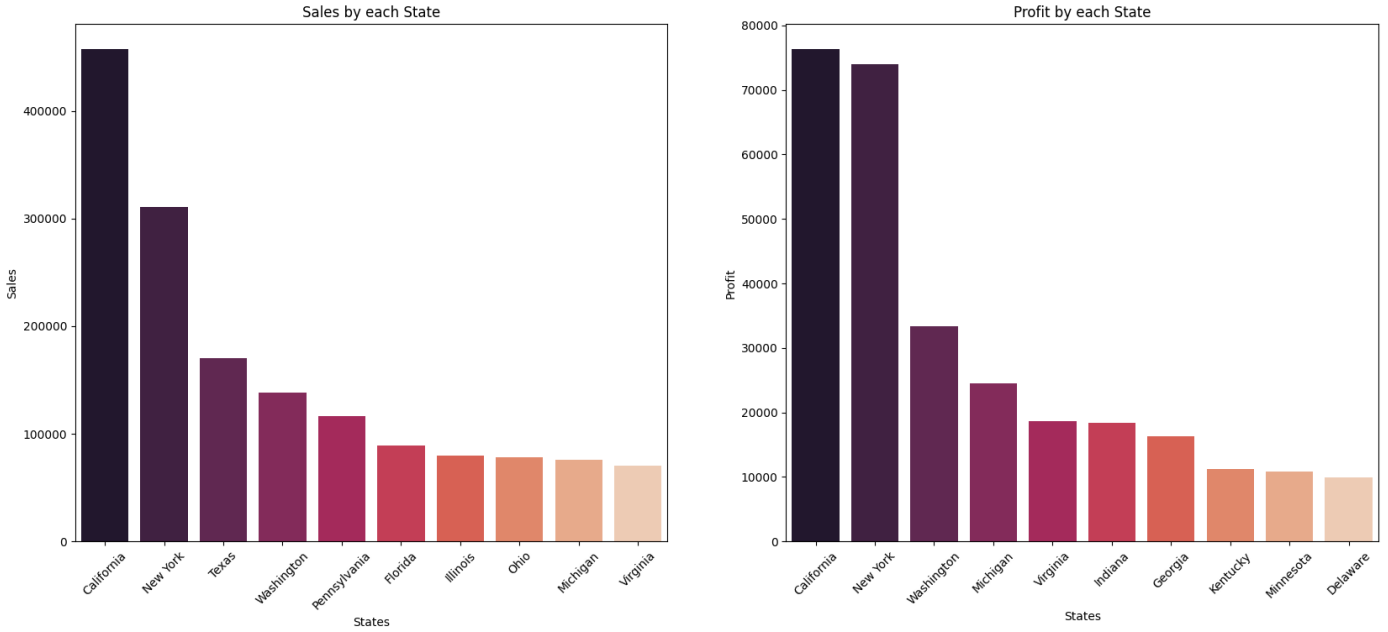
State

In [ ]:
```
state_grouped=df_place.groupby(['State'], as_index=False).sum()          # Group by State
state_sales= state_grouped.sort_values(by='Sales', ascending=False)      # Sort by Profit
state_profit= state_grouped.sort_values(by='Profit', ascending=False)   # Sort by Profit

# print each one
print('state_sales \n \n', state_sales.head())
print('-'*30)
print('state_profit \n \n', state_profit.head())
```

```
state_sales

            State        Sales      Profit
3      California  457687.6315  76381.3871
30       New York  310876.2710  74038.5486
41          Texas  170188.0458 -25729.3563
45     Washington  138641.2700  33402.6517
36   Pennsylvania  116511.9140 -15559.9603
------------------------------
state_profit

          State        Sales      Profit
3    California  457687.6315  76381.3871
30     New York  310876.2710  74038.5486
45   Washington  138641.2700  33402.6517
20     Michigan   76269.6140  24463.1876
44     Virginia   70636.7200  18597.9504
```

In [ ]:
```
# visualize both to combare between them
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))
state_sales_2= state_sales.head(10)
state_profit_2= state_profit.head(10)
# for sales
sns.barplot(data=state_sales_2, x='State', y='Sales', palette='rocket', ax=ax1)
ax1.set_xlabel('States')
ax1.set_ylabel('Sales')
ax1.set_title('Sales by each State')
ax1.tick_params(axis='x', rotation=45)  # Rotate x-axis labels

# for profit
```

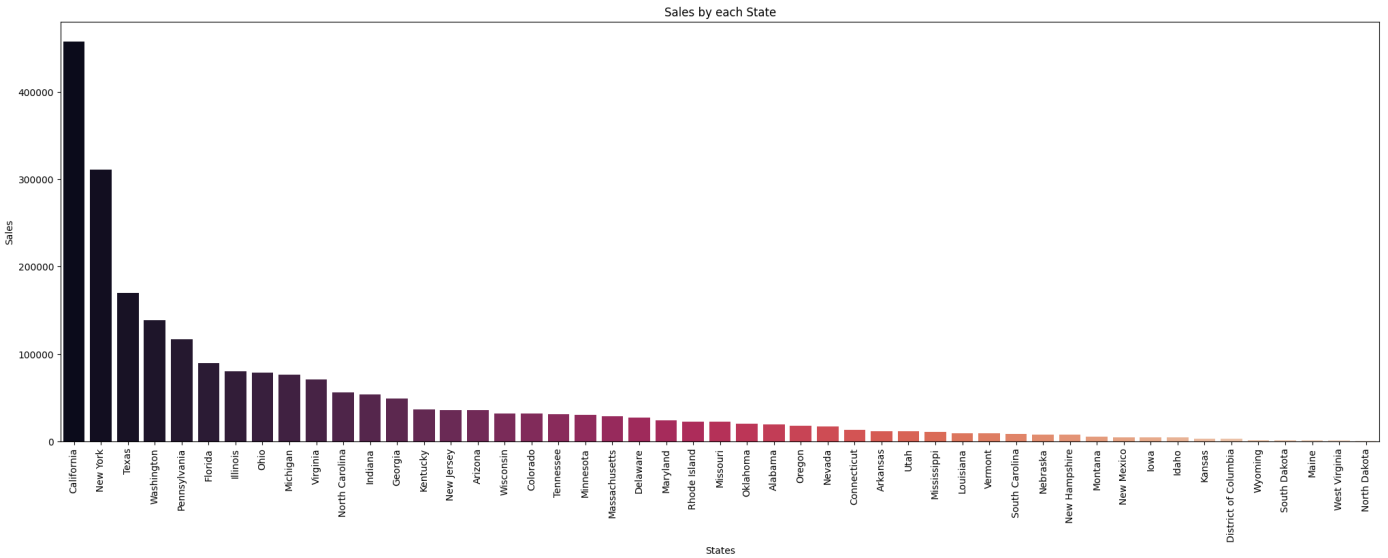
```
sns.barplot(data=state_profit_2, x='State', y='Profit', palette='rocket', ax=ax2)
ax2.set_xlabel('States')
ax2.set_ylabel('Profit')
ax2.set_title('Profit by each State')
ax2.tick_params(axis='x', rotation=45)  # Rotate x-axis labels

plt.show()
```



In [ ]:
```
len(state_sales)
```

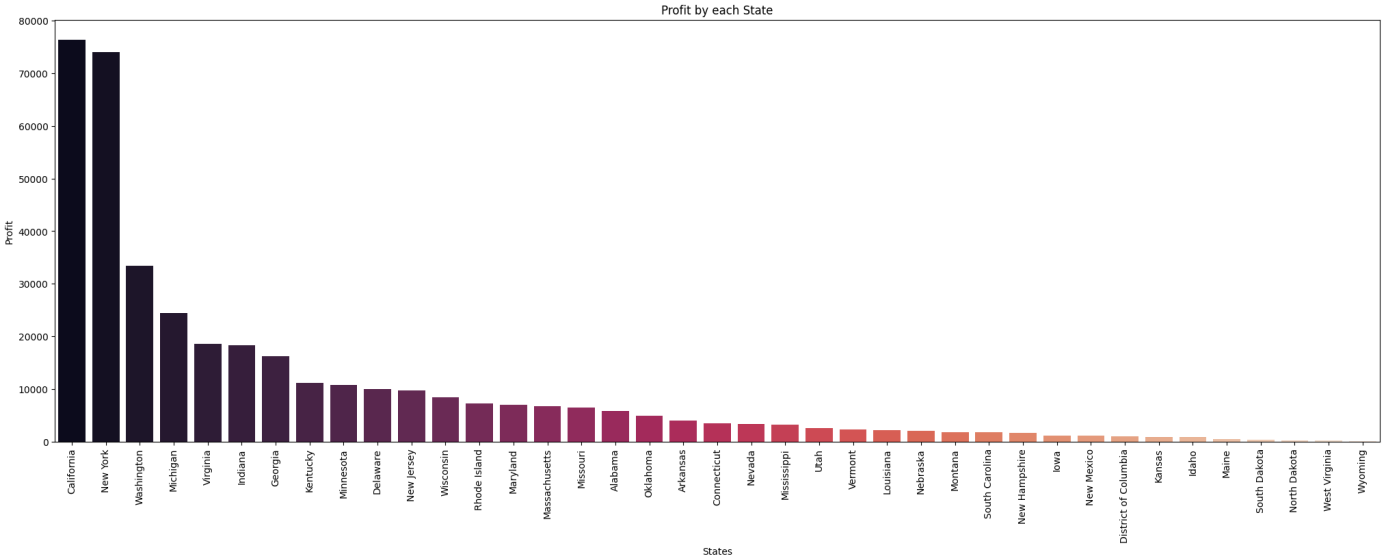
Out[ ]:
```
49
```

In [ ]:
```
plt.figure(figsize=(25,8))
sns.barplot(data=state_sales, x='State', y='Sales', palette='rocket')
plt.xlabel('States')
plt.ylabel('Sales')
plt.title('Sales by each State')
plt.tick_params(axis='x', rotation=90)  # Rotate x-axis labels
plt.show()
```



In [ ]:
```
state_profit_1 = state_profit[state_profit['Profit'] > 0] # delete any state contain neg

plt.figure(figsize=(25,8))

sns.barplot(data=state_profit_1, x='State', y='Profit', palette='rocket')
```

```
plt.xlabel('States')
plt.ylabel('Profit')
plt.title('Profit by each State')
plt.tick_params(axis='x', rotation=90)  # Rotate x-axis labels

plt.show()
```



Top Placeies are :

- Cities : New York , Los Angeles, Seattle, San Francisco
- State : California, New York
- Region : West

Top Profitable Products

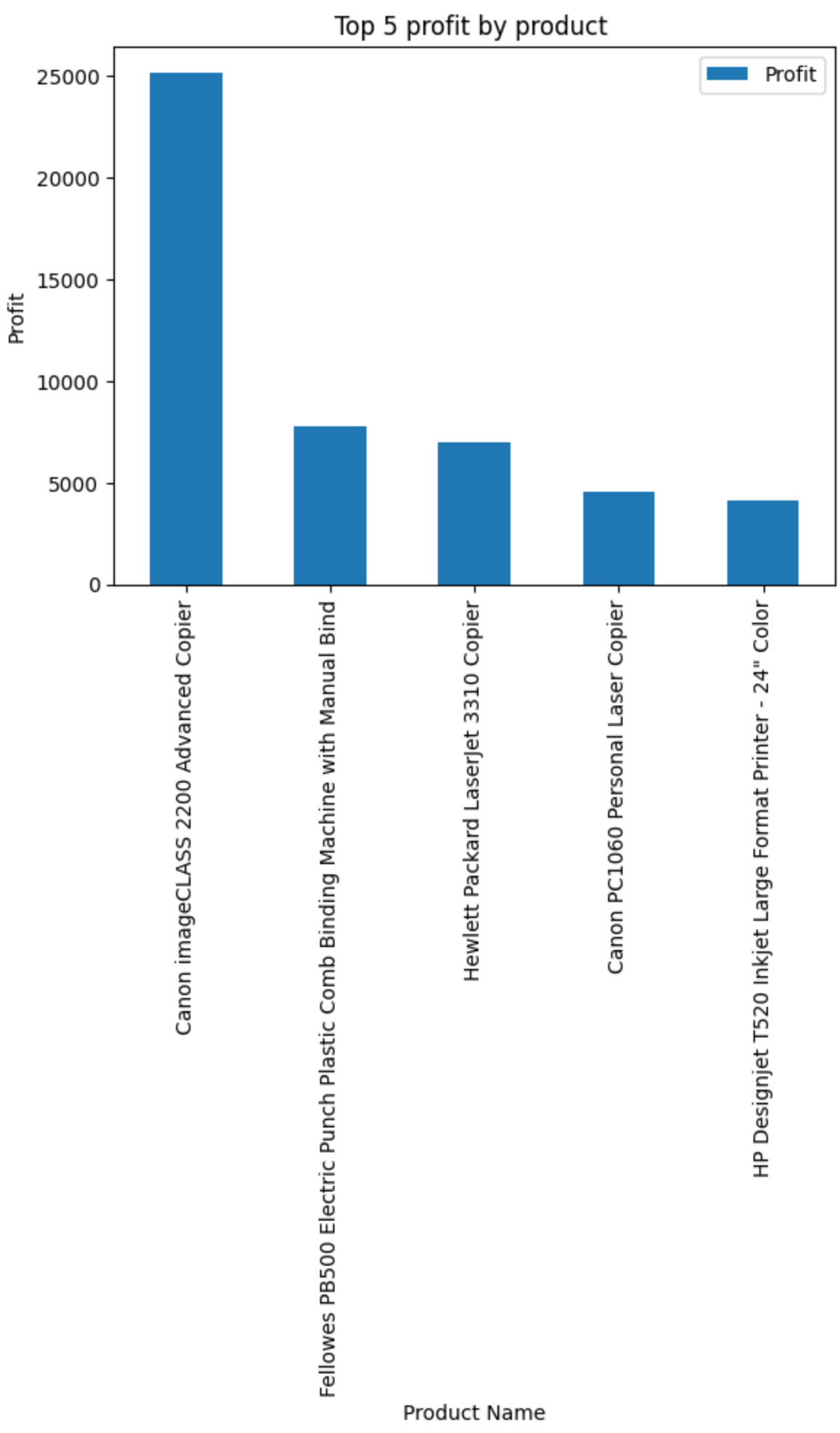
In [ ]:
```
profit_group= df.groupby(['Product Name']).sum()['Profit'] # groub product name by profi
top_profit_prod= profit_group.sort_values(ascending=False)
```

In [ ]:
```
top_profit= pd.DataFrame(top_profit_prod[:10])
top_profit.head()
```

Out[ ]:

Product Name	Profit
Canon imageCLASS 2200 Advanced Copier	25199.9280
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind	7753.0390
Hewlett Packard LaserJet 3310 Copier	6983.8836
Canon PC1060 Personal Laser Copier	4570.9347
HP Designjet T520 Inkjet Large Format Printer - 24" Color	4094.9766

In [ ]:
```
top_5_profit =top_profit.head()
top_5_profit.plot(kind='bar')
plt.xlabel('Product Name')
plt.ylabel('Profit')
plt.title('Top 5 profit by product ')
plt.show()
```

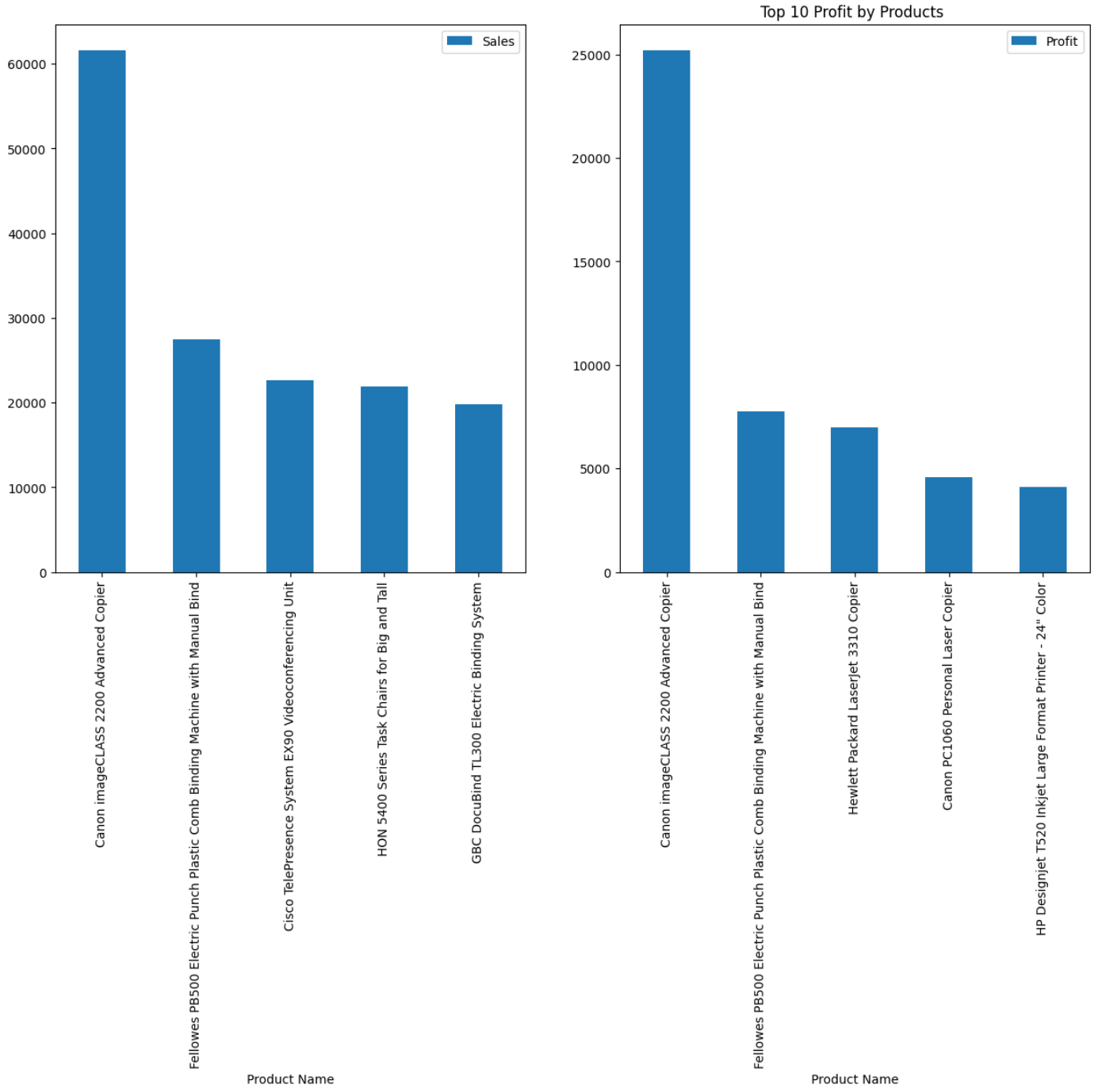


Are the best-selling products with the highest profit

In [ ]:
```
top_5_profit.index == top_5_selling_prod.index
```

Out[ ]:
```
array([ True,  True, False, False, False])
```

In [ ]:
```
fig,(ax1,ax2) = plt.subplots(1,2, figsize=(15,8))

# plot the top 10 selling by products in first columns
top_5_selling_prod.plot(kind='bar', y= 'Sales', ax=ax1)
plt.title('top 10 selling by products ')

# ploting the top 10 profit by products
top_5_profit.plot(kind='bar', y='Profit', ax=ax2)
plt.title('Top 10 Profit by Products ')
plt.show()
```



Top Products and Top Profits

ng products is :

1. Canon imageCLASS 2200 Advanced Copier
2. Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind
3. Cisco TelePresence System EX90 Videoconferencing Unit
4. HON 5400 Series Task Chairs for Big and Tall
5. GBC DocuBind TL300 Electric Binding System

==========

- **Top 5 products make Profits:**

1. Canon imageCLASS 2200 Advanced Copier
2. Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind
3. Hewlett Packard LaserJet 3310 Copier
4. Canon PC1060 Personal Laser Copier
5. HP Designjet T520 Inkjet Large Format Printer - 24" Color

==========

- **Top-selling and profitable products**

1. Canon imageCLASS 2200 Advanced Copier
2. Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind

Overall Trend:

- The sales trend shows fluctuations throughout the year, with some months performing better than others.
- There is no clear linear pattern in the monthly sales data, indicating that the sales are influenced by various factors and may not follow a consistent trend.

What is the impact of discounts on sales?

In [ ]:
```
df['Discount'].value_counts()
```

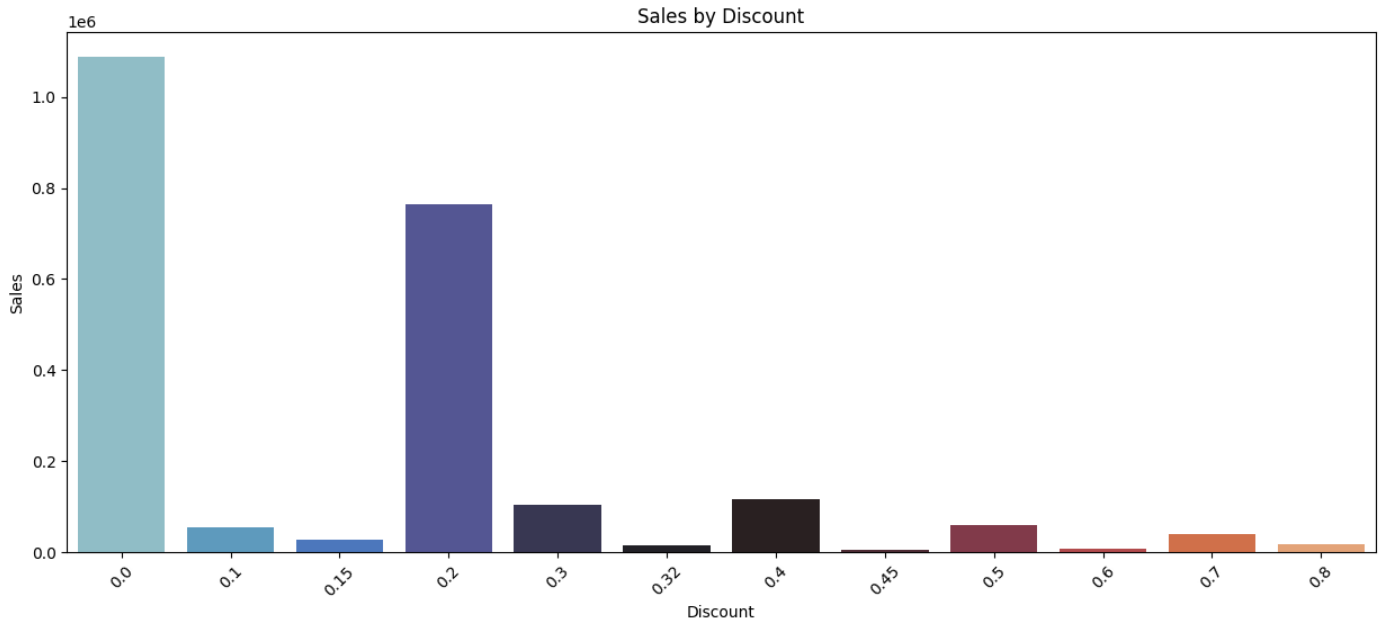
Out[ ]:
```
0.00    4798
0.20    3657
0.70     418
0.80     300
0.30     227
0.40     206
0.60     138
0.10      94
0.50      66
0.15      52
0.32      27
0.45      11
Name: Discount, dtype: int64
```

In [ ]:
```
discount_per_sales =df.groupby(['Discount']).sum()[['Sales']]
discount_per_sales= discount_per_sales.reset_index()
discount_per_sales
```

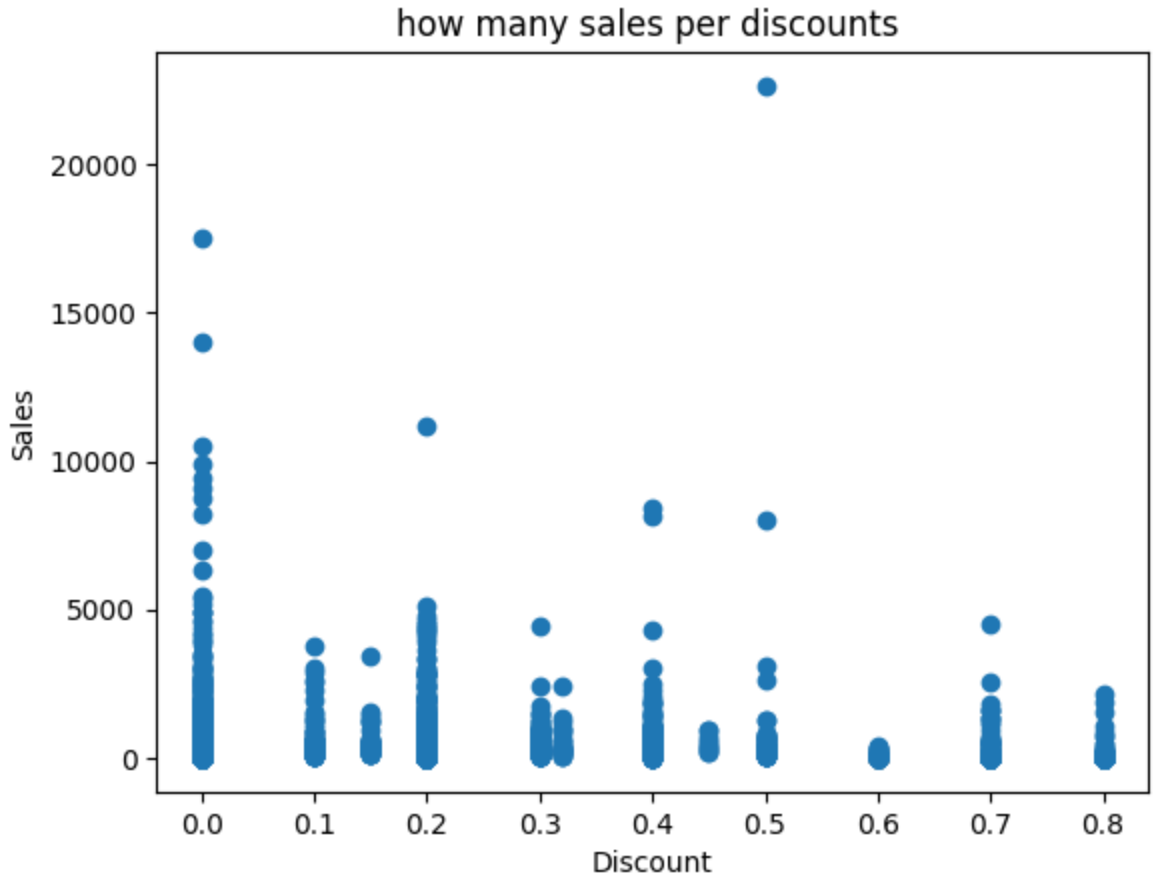
Out[ ]:

	Discount	Sales
0	0.00	1.087908e+06
1	0.10	5.436935e+04
2	0.15	2.755852e+04
3	0.20	7.645944e+05
4	0.30	1.032267e+05
5	0.32	1.449346e+04
6	0.40	1.164178e+05
7	0.45	5.484974e+03
8	0.50	5.891854e+04
9	0.60	6.644700e+03
10	0.70	4.062028e+04
11	0.80	1.696376e+04

In [ ]:
```
# for sales
plt.figure(figsize=(15,6))
sns.barplot(data=discount_per_sales, x='Discount', y='Sales', palette='icefire')
plt.xlabel('Discount')
plt.ylabel('Sales')
plt.title('Sales by Discount')
plt.tick_params(axis='x', rotation=45)
plt.show()
```

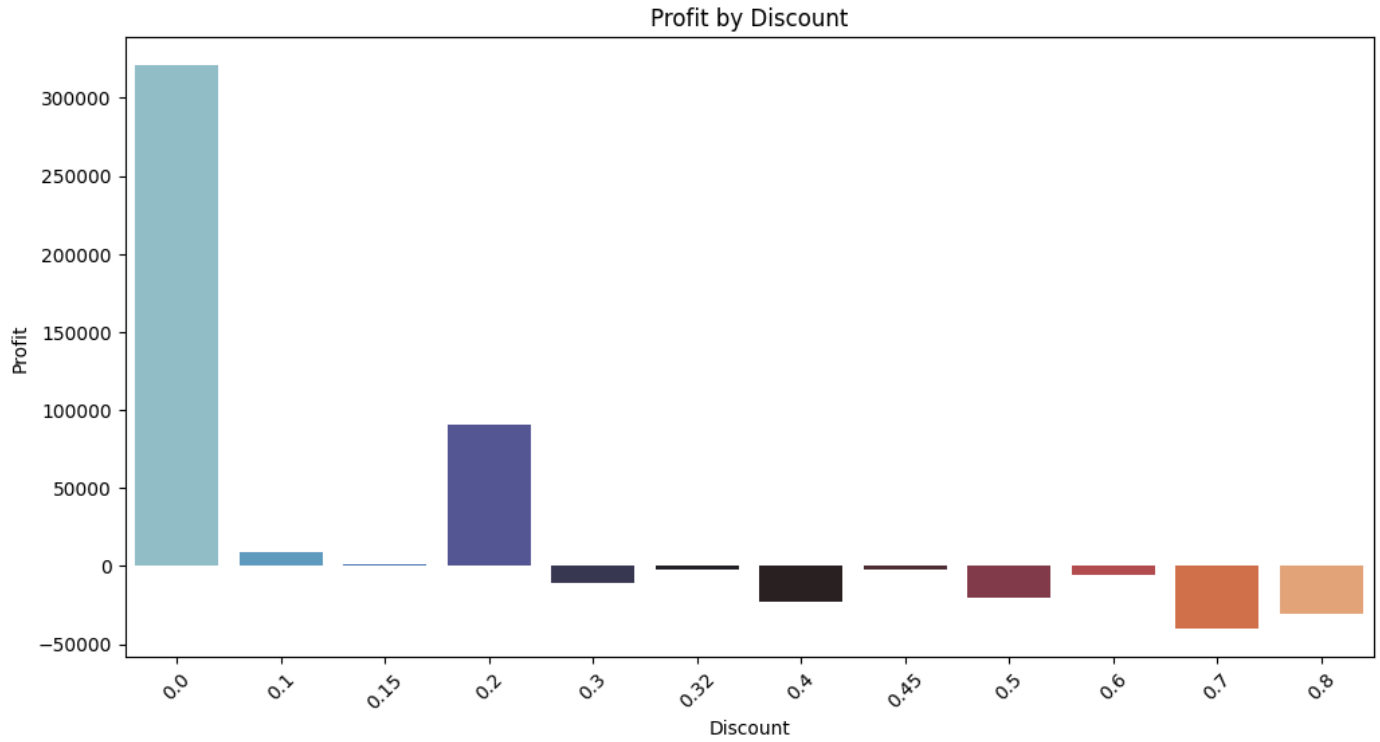


In [ ]:
```
# Scatter plot between Sales and Discounts

plt.scatter(df['Discount'] ,df['Sales'])
plt.xlabel('Discount')
plt.ylabel('Sales')
plt.title('how many sales per discounts ')
plt.show()
```



In [ ]:
```
# Group by 'Discount' 'Profit' for each discount level

discount_per_profit =df.groupby(['Discount']).sum()[['Profit']] # Group by Profit
discount_per_profit= discount_per_profit.reset_index()          # make it in df using re
# discount_per_profit
```

In [ ]:
```
# BarPlot for Profit
plt.figure(figsize=(12,6))
sns.barplot(data=discount_per_profit, x='Discount', y='Profit', palette='icefire')
plt.xlabel('Discount')
plt.ylabel('Profit')
plt.title('Profit by Discount')
plt.tick_params(axis='x', rotation=45)
plt.show()
```



- Dealing with sales and profit to see impact of Discount on each one of them
  - Sales:
    - This data represents discount levels and their corresponding sales figures. As discounts increase from 0% to 80%, sales vary accordingly, ranging from $5,484 to $1,087,908.
  - Profit:
    - This data displays profits associated with varying discount levels. Profit figures range from -$40,075 to $90,337. Higher discounts generally result in lower profits, with negative values observed beyond a 30% discount.

Top Profitable Products

In [ ]:
```
profit_group= df.groupby(['Product Name']).sum()['Profit'] # groub product name by profi
top_profit_prod= profit_group.sort_values(ascending=False)
```

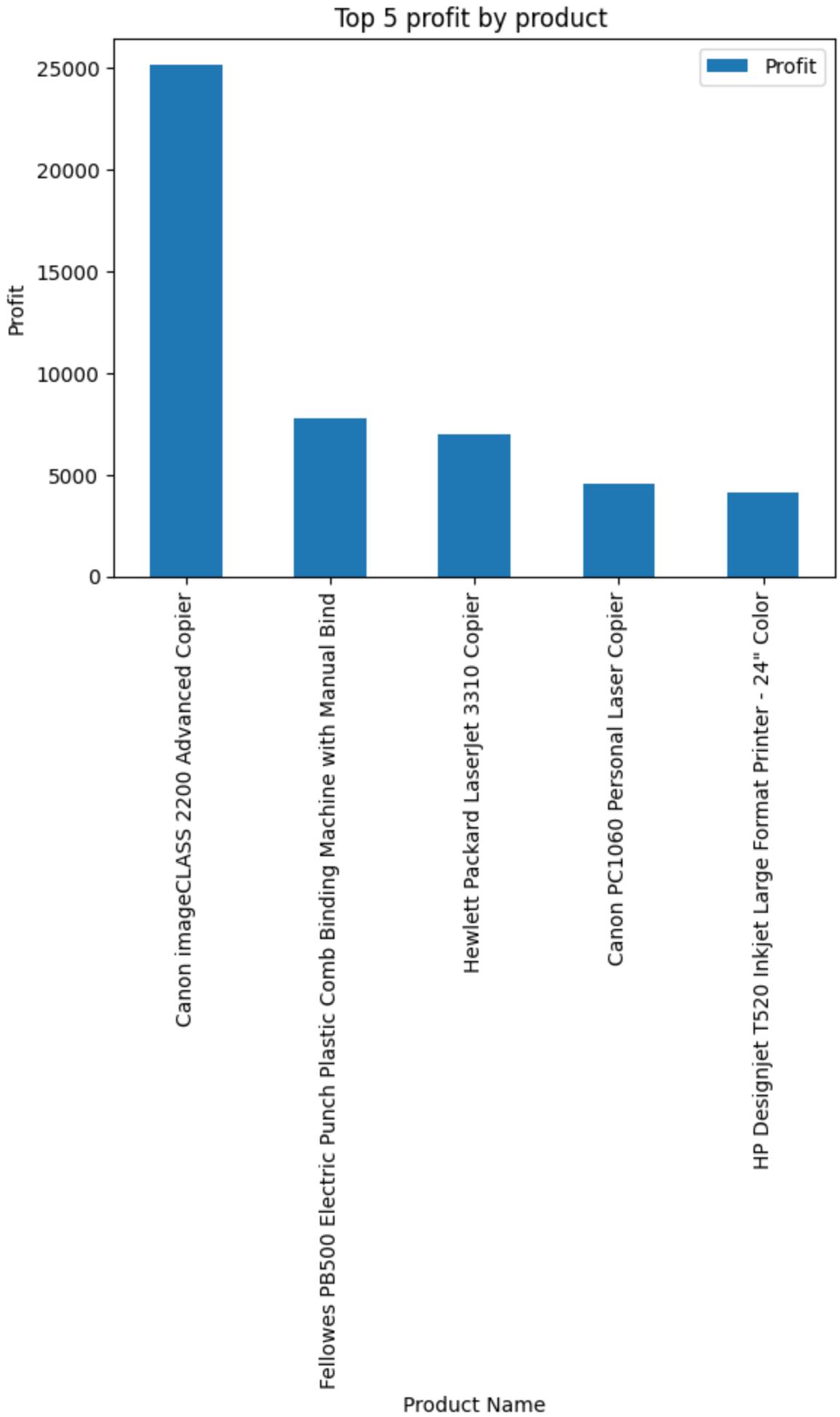
In [ ]:
```
top_profit= pd.DataFrame(top_profit_prod[:10])
top_profit.head()
```

Out[ ]:

Product Name	Profit
Canon imageCLASS 2200 Advanced Copier	25199.9280
Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind	7753.0390
Hewlett Packard LaserJet 3310 Copier	6983.8836
Canon PC1060 Personal Laser Copier	4570.9347
HP Designjet T520 Inkjet Large Format Printer - 24" Color	4094.9766

In [ ]:
```
top_5_profit =top_profit.head()
top_5_profit.plot(kind='bar')
plt.xlabel('Product Name')
rofit')
```

```
plt.title('Top 5 profit by product ')
plt.show()
```

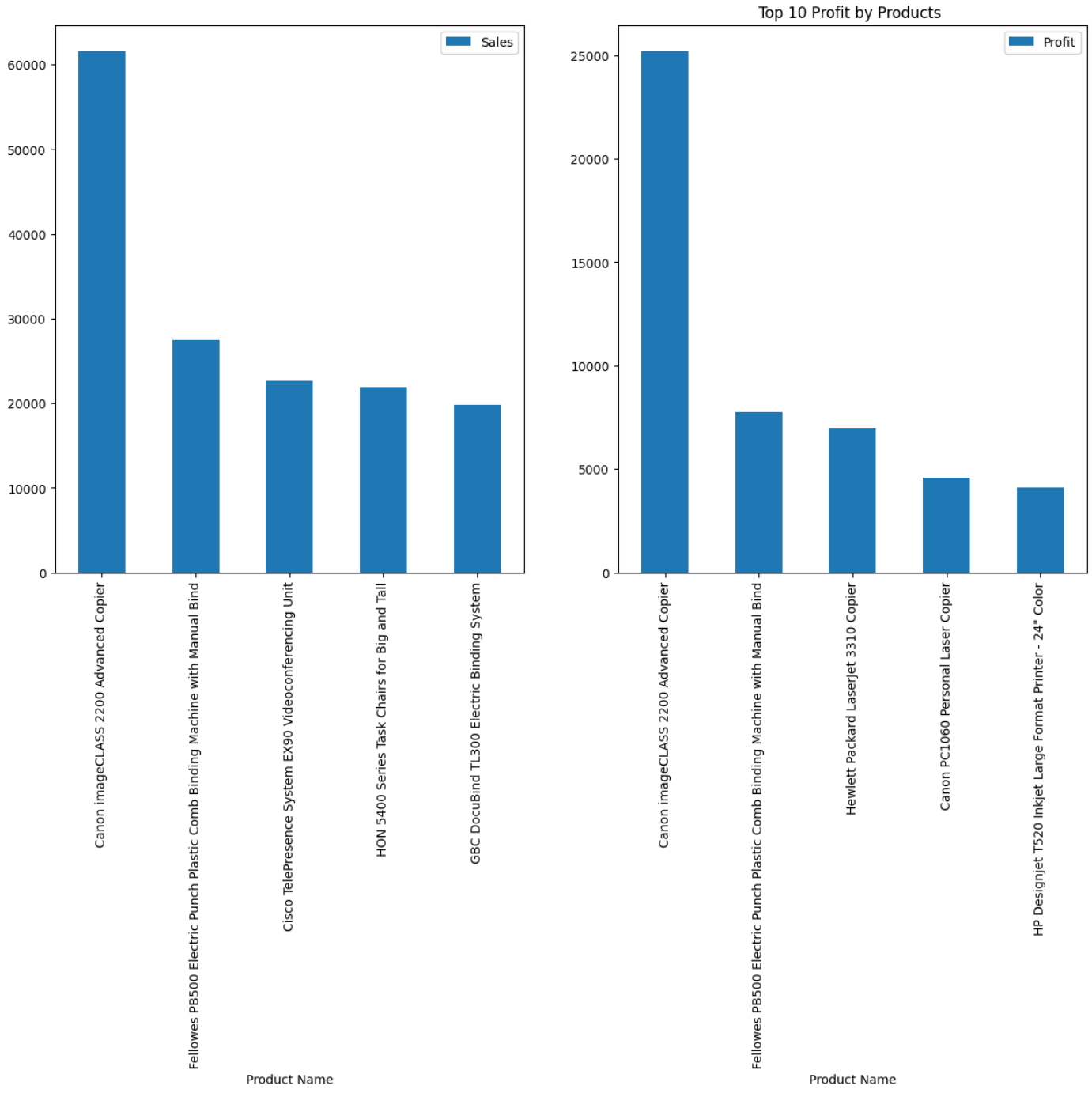


Are the best-selling products with the highest profit

In [ ]:
```
top_5_profit.index == top_5_selling_prod.index
```

Out[ ]:
```
array([ True,  True, False, False, False])
```

In [ ]:
```
fig,(ax1,ax2) = plt.subplots(1,2, figsize=(15,8))

# plot the top 10 selling by products in first columns
top_5_selling_prod.plot(kind='bar', y= 'Sales', ax=ax1)
plt.title('top 10 selling by products ')

# ploting the top 10 profit by products
top_5_profit.plot(kind='bar', y='Profit', ax=ax2)
plt.title('Top 10 Profit by Products ')
plt.show()
```



Top Products and Top Profits

- **Top 5 selling products is :**

1. Canon imageCLASS 2200 Advanced Copier
2. Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind
3. Cisco TelePresence System EX90 Videoconferencing Unit
4. HON 5400 Series Task Chairs for Big and Tall
5. GBC DocuBind TL300 Electric Binding System

==========

- **Top 5 products make Profits:**

1. Canon imageCLASS 2200 Advanced Copier
2. Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind
3. Hewlett Packard LaserJet 3310 Copier
4. Canon PC1060 Personal Laser Copier
5. HP Designjet T520 Inkjet Large Format Printer - 24" Color

==========

- **Top-selling and profitable products**

1. Canon imageCLASS 2200 Advanced Copier
2. Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind

Overall Trend:

- The sales trend shows fluctuations throughout the year, with some months performing better than others.
- There is no clear linear pattern in the monthly sales data, indicating that the sales are influenced by various factors and may not follow a consistent trend.

Profit by products

In [ ]:
```
df.Category.value_counts()
```

Out[ ]:
```
Office Supplies    6026
Furniture          2121
Technology         1847
Name: Category, dtype: int64
```

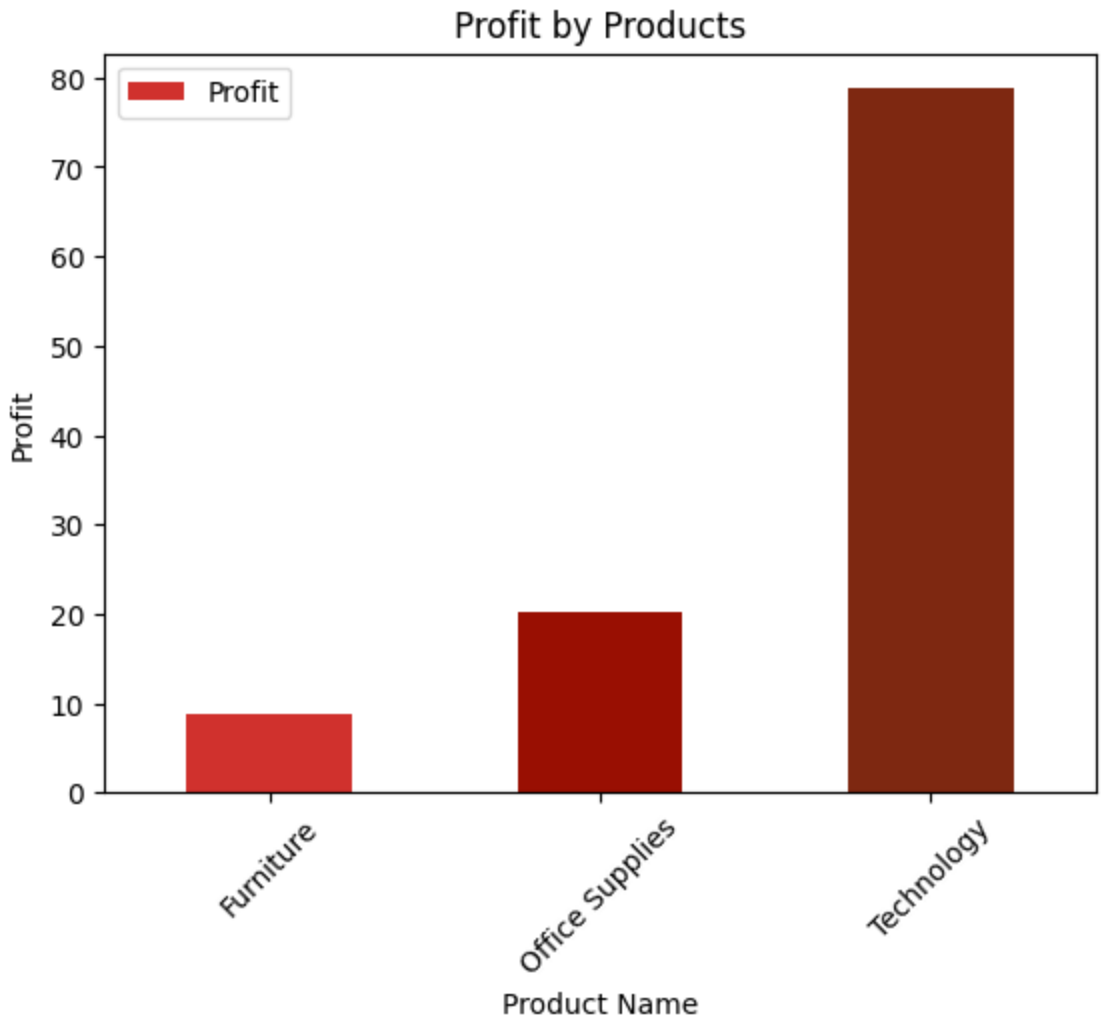
In [ ]:
```
profit_by_products= df.groupby('Category')['Profit'].mean().reset_index()
profit_by_products
```

Out[ ]:

	Category	Profit
0	Furniture	8.699327
1	Office Supplies	20.327050
2	Technology	78.752002

In [ ]:
```
# Create a bar plot with red color
plt.figure(figsize=(12, 6))
profit_by_products.plot(kind='bar', x='Category', y='Profit', color=['#D0312D','#990F02'
plt.xlabel('Product Name')
plt.ylabel('Profit')
plt.title('Profit by Products')
plt.xticks(rotation=45)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



In [ ]:
```
df1 =df.copy()
# df1.head(3)
```

In [ ]:
```
df1['Profit Margin']= df1['Profit'] / df1['Sales']
```

In [ ]:
```
avg_profit_margin_catig= df1.groupby('Category')['Profit Margin'].mean().reset_index()
avg_profit_margin_catig
```

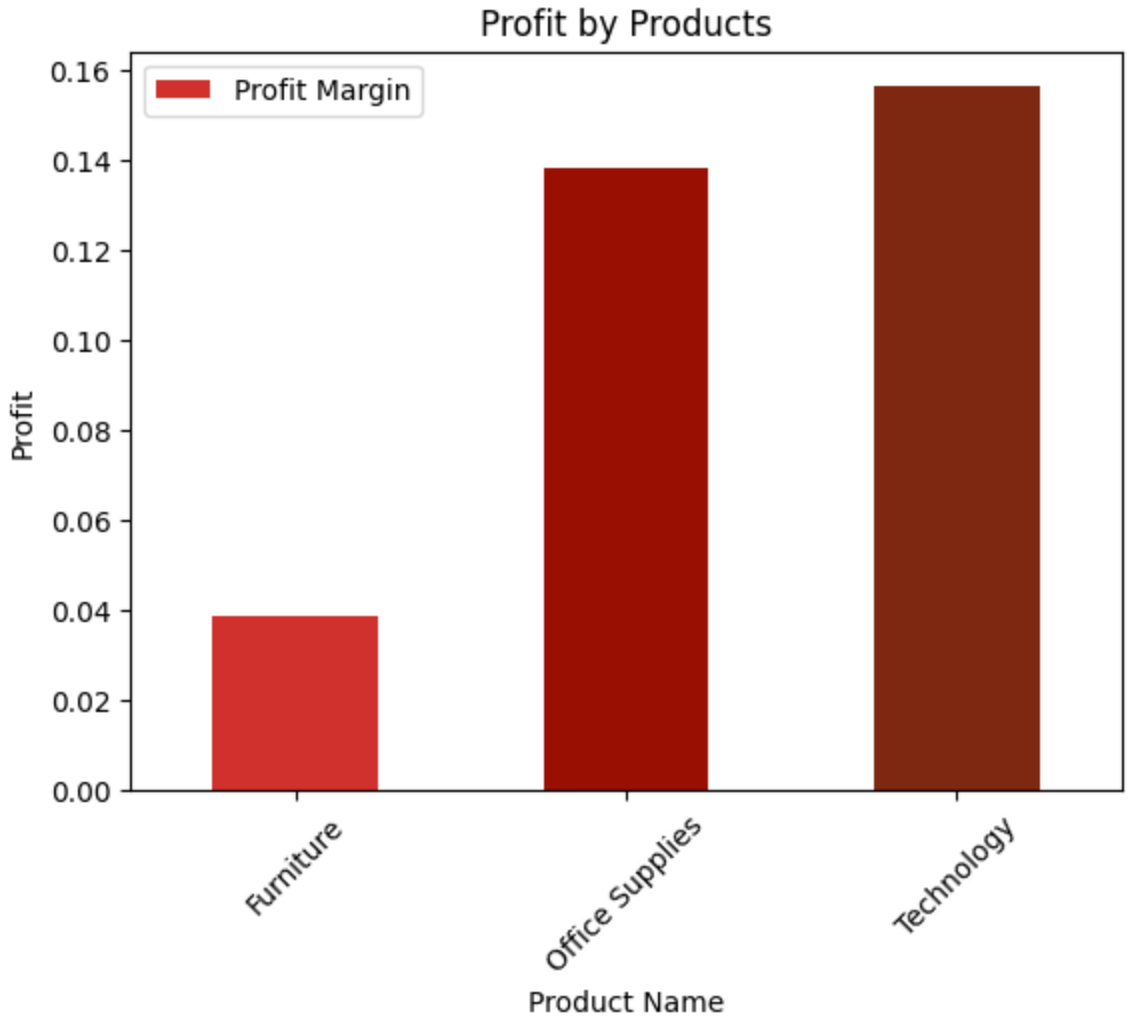
Out[ ]:

	Category	Profit Margin
0	Furniture	0.038784
1	Office Supplies	0.138030
2	Technology	0.156138

In [ ]:
```
plt.figure(figsize=(12, 6))
avg_profit_margin_catig.plot(kind='bar', x='Category', y='Profit Margin', color=['#D0312
plt.xlabel('Product Name')
plt.ylabel('Profit')
```

```
plt.title('Profit by Products')
plt.xticks(rotation=45)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



- Profit:
  - This data highlights the profits associated with different categories. Furniture yields a profit of $8.70, Office Supplies generates $20.33, and Technology leads with a profit of $78.75.
- Profit Margin:
  - The table presents distinct categories and their corresponding profit margins. Furniture has the lowest profit margin at 3.88%, followed by Office Supplies at 13.80%, while Technology boasts the highest profit margin at 15.61%.

Customers

1. Sales

In [ ]:
```
df.head(2)
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	State	Postal Code
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	Kentucky	42420

In [ ]:
```
df['Segment'].value_counts()
```

Out[ ]:
```
Consumer       5191
Corporate      3020
Home Office    1783
Name: Segment, dtype: int64
```

In [ ]:
```
df['Ship Mode'].value_counts()
```

Out[ ]:
```
Standard Class    5968
Second Class      1945
First Class       1538
Same Day           543
Name: Ship Mode, dtype: int64
```

1. Profit

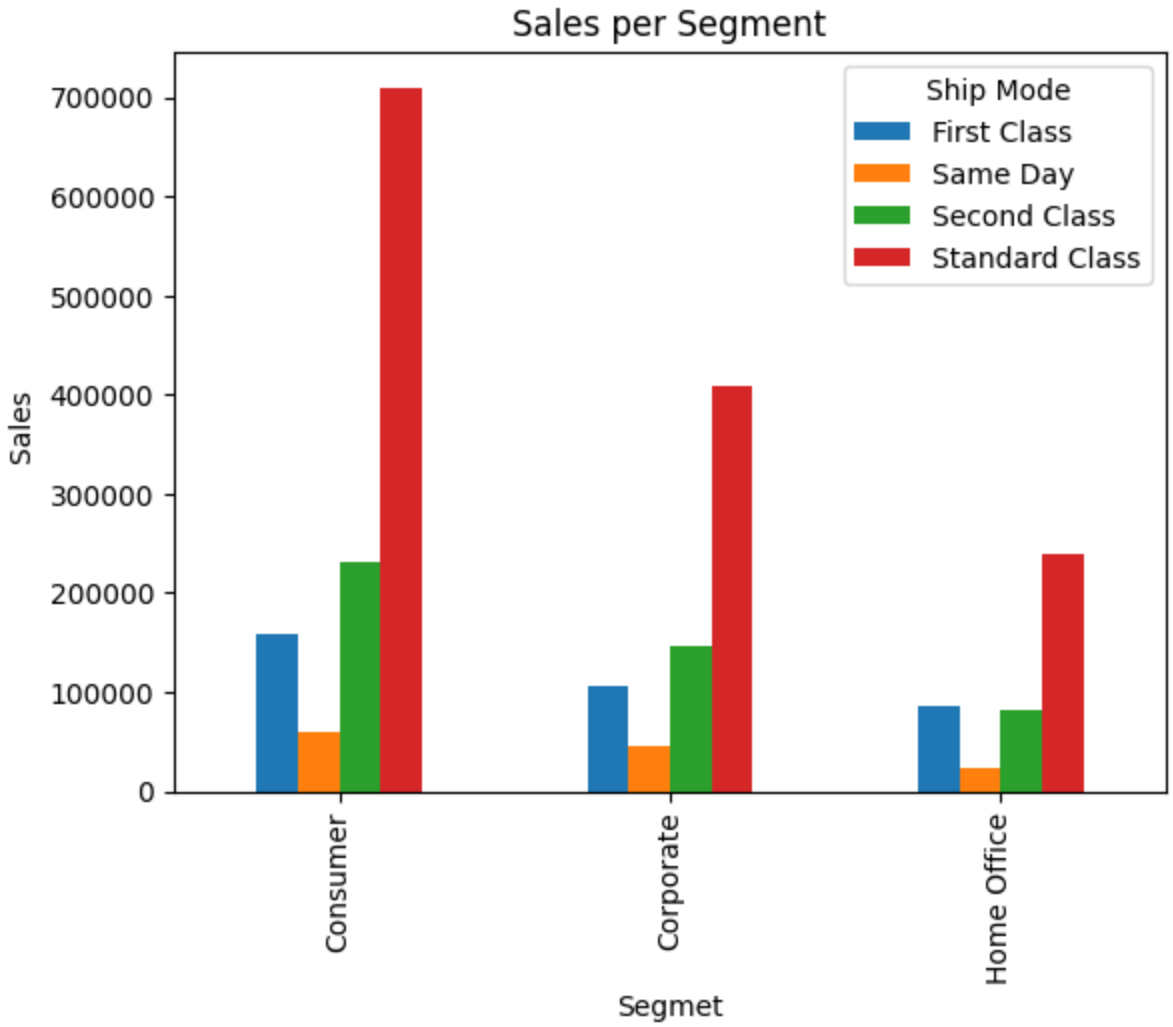
In [ ]:
```
pivot_table_sales= df.pivot_table(index='Segment' , columns='Ship Mode', values='Sales',
pivot_table_sales
```

Out[ ]:

Ship Mode / Segment	First Class	Same Day	Second Class	Standard Class
Consumer	159168.9650	60596.359	231498.9496	710137.0714
Corporate	105858.4699	45121.323	146126.0388	409040.5351
Home Office	86400.9880	22645.443	81568.5810	239038.1365

In [ ]:
```
plt.figure(figsize=(12,5))
pivot_table_sales.plot(kind='bar', stacked=False)    # by default stacked is false
plt.xlabel ('Segmet')
plt.ylabel('Sales ')
plt.title('Sales per Segment')
plt.show()
```

```
<Figure size 1200x500 with 0 Axes>
```



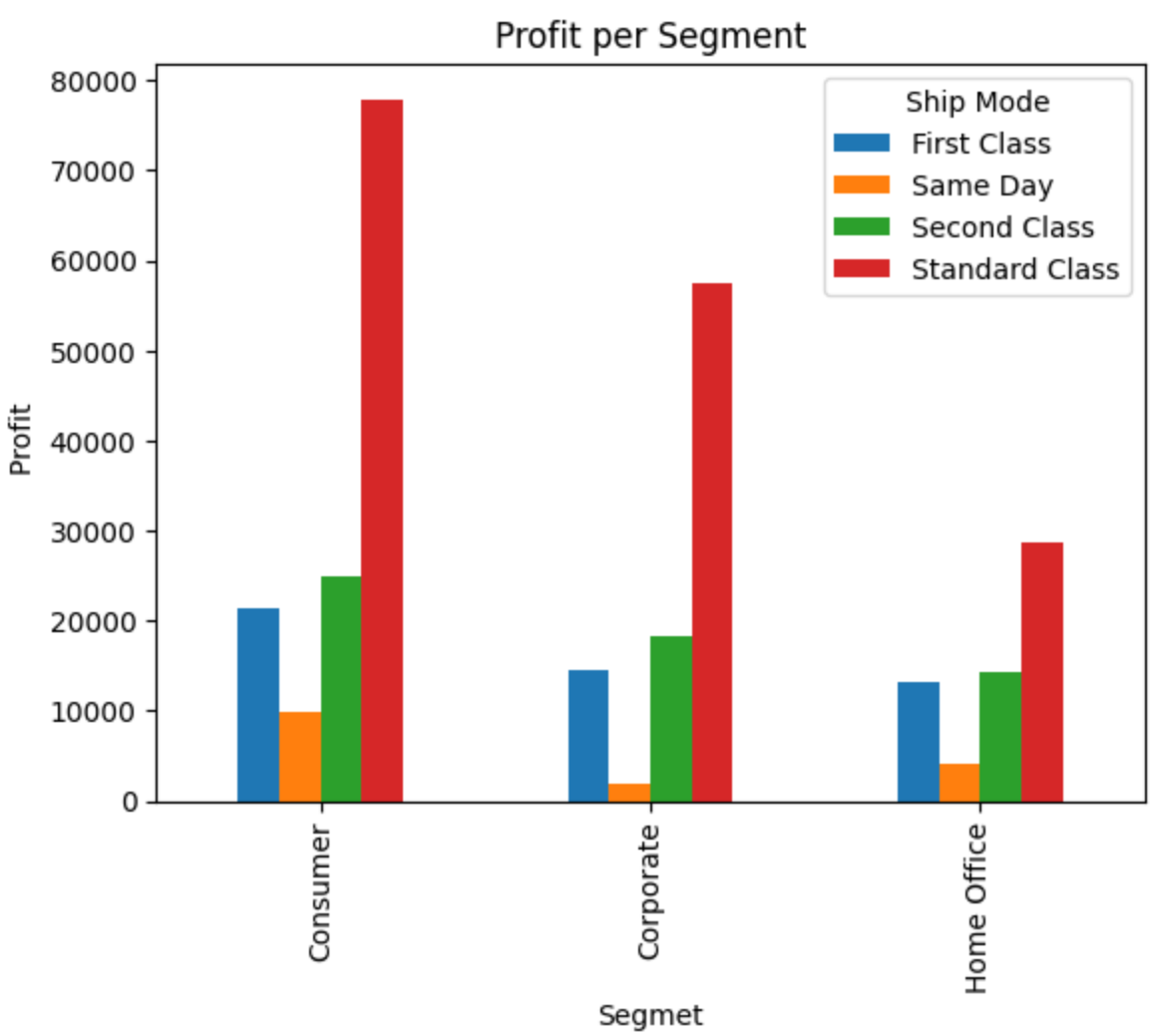
In [ ]:
```
pivot_table_profit= df.pivot_table(index='Segment' , columns='Ship Mode', values='Profit
pivot_table_profit
```

Out[ ]:

Segment	First Class	Same Day	Second Class	Standard Class
Consumer	21374.0436	9874.2054	24946.9112	77924.0490
Corporate	14464.4724	1818.1418	18225.7131	57470.8067
Home Office	13131.3239	4199.4117	14274.0111	28693.9318

In [ ]:
```
plt.figure(figsize=(12,5))
pivot_table_profit.plot(kind='bar', stacked=False)    # by default stacked is false
plt.xlabel ('Segmet')
plt.ylabel('Profit ')
plt.title('Profit per Segment')
plt.show()
```

```
<Figure size 1200x500 with 0 Axes>
```



In [ ]:
```
Sales_shipdf = df.groupby(['State','Ship Mode']).aggregate({'Sales':'mean'}).reset_index
Sales_shipdf = Sales_shipdf.pivot(index='State', columns='Ship Mode', values='Sales').re
Sales_shipdf['total'] = Sales_shipdf['First Class']+Sales_shipdf['Same Day']+Sales_shipd

Sales_shipdf = Sales_shipdf.sort_values(by = 'total', ascending = False)
Sales_shipdf.drop(columns ='total', inplace=True)
Sales_shipdf.head()
```

Out[ ]:

Ship Mode	State	First Class	Same Day	Second Class	Standard Class
26	Nevada	392.239600	475.944000	567.149667	356.484000
31	North Carolina	159.149263	875.506929	248.553333	173.284871
20	Michigan	520.668857	230.780222	356.993813	262.177506
12	Indiana	194.193125	483.973333	276.516857	413.876421
34	Oklahoma	356.334000	519.794286	184.144286	276.045745

This data illustrates sales figures for different shipping modes across customer segments. In the "First Class" shipping mode, the highest total sales are observed in the Consumer segment ($159,168.97), followed by Corporate ($105,858.47) and Home Office ($86,400.99). For "Same Day" shipping, the Consumer segment leads in sales ($60,596.36), while Corporate and Home Office have lower figures. In "Second Class" shipping, the Consumer segment again has the highest sales ($231,498.95), with Corporate and Home Office following. Lastly, in "Standard Class" shipping, the Consumer segment stands out with the highest sales ($710,137.07), followed by Corporate and Home Office. Overall, the Consumer segment consistently shows the highest sales across various shipping modes.

Machine Learning

Libraries

In [ ]:
```
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.metrics import classification_report,mean_absolute_error, mean_squared_erro
from sklearn.feature_selection import SelectKBest, f_regression,  f_classif
from sklearn.ensemble import RandomForestRegressor, BaggingRegressor, StackingRegressor
```

Preprocessing

check null values

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
Row ID           0
Order ID         0
Order Date       0
Ship Date        0
Ship Mode        0
Customer ID      0
Customer Name    0
Segment          0
Country          0
City             0
State            0
Postal Code      0
Region           0
Product ID       0
Category         0
Sub-Category     0
Product Name     0
Sales            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

Encoded Categorical Columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 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   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
```

Now I need to drop not nessasry columns to build a good ML model

columns like Row ID not nessasry and customer ID not nessesary to predect the sales, and Customer Name will not effect cause i am not going to make customer segments, ... etc

In [ ]:
```
df2 = df.drop(['Row ID', 'Order ID','City', 'Order Date', 'Ship Date', 'Customer ID', 'C
```

In [ ]:
```
df2.head()
```

Out[ ]:

	Ship Mode	Segment	State	Region	Category	Sub-Category	Sales	Quantity	Discount	Profit
0	Second Class	Consumer	Kentucky	South	Furniture	Bookcases	261.9600	2	0.00	41.9136
1	Second Class	Consumer	Kentucky	South	Furniture	Chairs	731.9400	3	0.00	219.5820
2	Second Class	Corporate	California	West	Office Supplies	Labels	14.6200	2	0.00	6.8714
3	Standard Class	Consumer	Florida	South	Furniture	Tables	957.5775	5	0.45	-383.0310
4	Standard Class	Consumer	Florida	South	Office Supplies	Storage	22.3680	2	0.20	2.5164

In [ ]:
```
df2.nunique()
```

Out[ ]:
```
Ship Mode          4
Segment            3
State             49
Region             4
Category           3
Sub-Category      17
Sales           5825
Quantity          14
Discount          12
Profit          7287
dtype: int64
```

In [ ]:
```
# Catigorical Columns
categorical_df =  df2.select_dtypes(include=['object'])

# Select Numerical Data frame
numerical_df = df2.select_dtypes(include=['number'])
```

In [ ]:
```
print(f'Shape of df2 is: \n {df2.shape}\n ')
print('='*20)
print(f'shape of Catigorical DataFrame: \n {categorical_df.shape} \n')
print('='*20)
print(f'Shape of Nummerical DataFrame: \n {numerical_df.shape}')
```

```
Shape of df2 is:
 (9994, 10)

====================
shape of Catigorical DataFrame:
 (9994, 6)

====================
Shape of Nummerical DataFrame:
 (9994, 4)
```

In [ ]:
```
categorical_df.head()
```

Out[ ]:

	Ship Mode	Segment	State	Region	Category	Sub-Category
0	Second Class	Consumer	Kentucky	South	Furniture	Bookcases
1	Second Class	Consumer	Kentucky	South	Furniture	Chairs
2	Second Class	Corporate	California	West	Office Supplies	Labels
3	Standard Class	Consumer	Florida	South	Furniture	Tables
4	Standard Class	Consumer	Florida	South	Office Supplies	Storage

In [ ]:
```
categorical_df.nunique()
```

Out[ ]:
```
Ship Mode        4
Segment          3
State           49
Region           4
Category         3
Sub-Category    17
dtype: int64
```

In [ ]:
```
numerical_df.head()
```

Out[ ]:

	Sales	Quantity	Discount	Profit
0	261.9600	2	0.00	41.9136
1	731.9400	3	0.00	219.5820
2	14.6200	2	0.00	6.8714
3	957.5775	5	0.45	-383.0310
4	22.3680	2	0.20	2.5164

In [ ]:
```
numerical_df.describe()
```

Out[ ]:

	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

In [ ]:
```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

columns_2_scaller = ['Sales', 'Quantity', 'Discount', 'Profit']
for i in columns_2_scaller:
    numerical_df[i] = scaler.fit_transform(numerical_df[[i]])
```

In [ ]:
```
numerical_df.head()
```

Out[ ]:

	Sales	Quantity	Discount	Profit
0	0.051510	-0.804303	-0.756643	0.056593
1	0.805633	-0.354865	-0.756643	0.815054
2	-0.345368	-0.804303	-0.756643	-0.093002
3	1.167688	0.544012	1.423149	-1.757484
4	-0.332935	-0.804303	0.212153	-0.111593

In [ ]:
```
categorical_df_encoded= pd.get_dummies(categorical_df)
```

In [ ]:
```
categorical_df_encoded.head()
```

Out[ ]:

	Ship Mode_First Class	Ship Mode_Same Day	Ship Mode_Second Class	Ship Mode_Standard Class	Segment_Consumer	Segment_Corporate	Segment_H C
0	0	0	1	0	1	0	
1	0	0	1	0	1	0	
2	0	0	1	0	0	1	
3	0	0	0	1	1	0	
4	0	0	0	1	1	0	

In [ ]:
```
categorical_df_encoded.shape
```

Out[ ]:
```
(9994, 80)
```

In [ ]:
```
df3=pd.concat([categorical_df_encoded,numerical_df], axis=1)
```

In [ ]:
```
df3.head()
```

Out[ ]:

	Ship Mode_First Class	Ship Mode_Same Day	Ship Mode_Second Class	Ship Mode_Standard Class	Segment_Consumer	Segment_Corporate	Segment_H C
0	0	0	1	0	1	0	
1	0	0	1	0	1	0	
2	0	0	1	0	0	1	
3	0	0	0	1	1	0	
4	0	0	0	1	1	0	

Spliting data

In [ ]:
```
# Selecting input features (X) and target variable (y)
X = df3.drop(['Sales'], axis=1)  # Drop 'Sales' and 'Profit' columns
y = df3['Sales']
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=15
X_train,X_valid, y_train,y_valid = train_test_split(X_train, y_train, test_size=0.2, ran
```

In [ ]:
```
X_train.head(2)
```

Out[ ]:

	Ship Mode_First Class	Ship Mode_Same Day	Ship Mode_Second Class	Ship Mode_Standard Class	Segment_Consumer	Segment_Corporate	Segme
9942	0	0	0	1	1	0	
9287	0	0	0	1	0	0	

In [ ]:
```
print(f' Train train dataset Shape: {X_train.shape}')
print(f'Target test variable Shape: {y_train.shape}')
```

```
 Train train dataset Shape: (6396, 83)
Target test variable Shape: (6396,)
```

Forcasting model for **Sales**

Random Forest Regressor

In [ ]:
```
# Create and train Random Forest Regressor model
RFR = RandomForestRegressor( n_estimators=10, max_depth=9, random_state=10)
random_RandomForest_Regressor=RFR.fit(X_train,y_train)
random_RandomForest_Regressor
```

Out[ ]:
```
▼                          RandomForestRegressor
RandomForestRegressor(max_depth=9, n_estimators=10, random_state=10)
```

In [ ]:
```
# Make predictions Validation dataset
rfr_valid = random_RandomForest_Regressor.predict(X_valid)

# Evaluate the model
mse_rf_valid= mean_squared_error(y_valid, rfr_valid)
rmse_rf_valid = np.sqrt(mse_rf_valid)
mae_rf_valid = mean_absolute_error(y_valid, rfr_valid)
r2_rf_valid = r2_score(y_valid, rfr_valid)

print("Mean Squared Error (MSE):", mse_rf_valid)
print("Root Mean Squared Error (RMSE):", rmse_rf_valid)
print("Mean Absolute Error (MAE):", mae_rf_valid)
print("R-squared (R2) Score:", r2_rf_valid)
```

```
Mean Squared Error (MSE): 0.32731900937025254
Root Mean Squared Error (RMSE): 0.5721180030118372
Mean Absolute Error (MAE): 0.15427853069218225
R-squared (R2) Score: 0.700919328638802
```

In [ ]:
```
# Make predictions Test dataset
rfr_test = random_RandomForest_Regressor.predict(X_test)

# Evaluate the model
mse_rf= mean_squared_error(y_test, rfr_test)
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test, rfr_test)
r2_rf = r2_score(y_test, rfr_test)

print("Mean Squared Error (MSE):", mse_rf)
print("Root Mean Squared Error (RMSE):", rmse_rf)
print("Mean Absolute Error (MAE):", mae_rf)
print("R-squared (R2) Score:", r2_rf)
```

```
Mean Squared Error (MSE): 0.2252185265828357
Root Mean Squared Error (RMSE): 0.47457194036609
Mean Absolute Error (MAE): 0.14370374596999022
R-squared (R2) Score: 0.7936438393094587
```

Decision Tree For Regression

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor

decision_tree_regressor = DecisionTreeRegressor( max_depth=9, random_state=10)
decision_tree_regressor.fit(X_train, y_train)
```

Out[ ]:
```
▼ DecisionTreeRegressor
DecisionTreeRegressor(max_depth=9, random_state=10)
```

In [ ]:
```
# Make predictions Validation dataset
dt_valid = random_RandomForest_Regressor.predict(X_valid)

# Evaluate the model
mse_dt_valid= mean_squared_error(y_valid, rfr_valid)
rmse_dt_valid = np.sqrt(mse_rf_valid)
mae_dy_valid = mean_absolute_error(y_valid, rfr_valid)
r2_dt_valid = r2_score(y_valid, rfr_valid)

print("Mean Squared Error (MSE):", mse_dt_valid)
print("Root Mean Squared Error (RMSE):", rmse_dt_valid)
print("Mean Absolute Error (MAE):", mae_dy_valid)
print("R-squared (R2) Score:", r2_dt_valid)
```

```
Mean Squared Error (MSE): 0.32731900937025254
Root Mean Squared Error (RMSE): 0.5721180030118372
Mean Absolute Error (MAE): 0.15427853069218225
R-squared (R2) Score: 0.700919328638802
```

In [ ]:
```
predictions = decision_tree_regressor.predict(X_test)
# Evaluate the model
mse_dt= mean_squared_error(y_test, predictions)
rmse_dt = np.sqrt(mse_dt)
mae_dt = mean_absolute_error(y_test, predictions)
r2_dt= r2_score(y_test, predictions)

print("Mean Squared Error (MSE):", mse_dt)
print("Root Mean Squared Error (RMSE):", rmse_dt)
print("Mean Absolute Error (MAE):", mae_dt)
print("R-squared (R2) Score:", r2_dt)
```

```
Mean Squared Error (MSE): 0.258240446694496
Root Mean Squared Error (RMSE): 0.5081736383309311
Mean Absolute Error (MAE): 0.16446917923357926
R-squared (R2) Score: 0.7633875510890238
```

Stacking model

In [ ]:
```
# Create base models
base_models = [
    ('decision_tree', DecisionTreeRegressor(random_state=42)),
    ('random_forest', RandomForestRegressor(random_state=42))
]

# Initialize the StackingRegressor
stacking_model = StackingRegressor(
    estimators=base_models,
    final_estimator=RandomForestRegressor(random_state=42)
)

# Fit the stacking model to the training data
stacking_model.fit(X_train, y_train)
```

In [ ]:
```
y_pred_stacking_train = stacking_model.predict(X_valid)

# Calculate evaluation metrics for the stacking model
mse_stacking = mean_squared_error(y_valid, y_pred_stacking_train)
r2_stacking = r2_score(y_valid, y_pred_stacking_train)
```

```
mae_stacking = mean_absolute_error(y_valid, y_pred_stacking_train)

print("Stacking Model - Mean Squared Error (MSE):", mse_stacking)
print("Stacking Model - R-squared (R2) Score:", r2_stacking)
print("Stacking Model - Mean Absolute Error (MAE):", mae_stacking)
```

```
Stacking Model - Mean Squared Error (MSE): 0.3726005909570416
Stacking Model - R-squared (R2) Score: 0.659544262010898
Stacking Model - Mean Absolute Error (MAE): 0.1465162561208837
```

In [ ]:
```
y_pred_stacking_test = stacking_model.predict(X_test)

# Calculate evaluation metrics for the stacking model
mse_stacking = mean_squared_error(y_test, y_pred_stacking_test)
r2_stacking = r2_score(y_test, y_pred_stacking_test)
mae_stacking = mean_absolute_error(y_test, y_pred_stacking_test)

print("Stacking Model - Mean Squared Error (MSE):", mse_stacking)
print("Stacking Model - R-squared (R2) Score:", r2_stacking)
print("Stacking Model - Mean Absolute Error (MAE):", mae_stacking)
```

```
Stacking Model - Mean Squared Error (MSE): 0.22373051104804928
Stacking Model - R-squared (R2) Score: 0.7950072314666908
Stacking Model - Mean Absolute Error (MAE): 0.13137841066299488
```

SVR

In [ ]:
```
from sklearn.svm import SVR
```

In [ ]:
```
svr_model = SVR(kernel='poly')
svr_model.fit(X_train,y_train)
```

Out[ ]:
```
▼ SVR
SVR(kernel='poly')
```

In [ ]:
```
svr_valid = svr_model.predict(X_valid)

# Calculate evaluation metrics
mse_valid = mean_squared_error(y_valid, svr_valid)
r2_valid = r2_score(y_valid, svr_valid)
mae_valid = mean_absolute_error(y_valid, svr_valid)

print("Mean Squared Error (MSE):", mse_valid)
print("R-squared (R2) Score:", r2_valid)
print("Mean Absolute Error (MAE):", mae_valid)
```

```
Mean Squared Error (MSE): 0.4768282735865434
R-squared (R2) Score: 0.5643084693961405
Mean Absolute Error (MAE): 0.1951957008342145
```

In [ ]:
```
svr_predictions = svr_model.predict(X_test)

# Calculate evaluation metrics
mse_svr = mean_squared_error(y_test, svr_predictions)
r2_svr = r2_score(y_test, svr_predictions)
mae_svr = mean_absolute_error(y_test, svr_predictions)

print("Mean Squared Error (MSE):", mse_svr)
print("R-squared (R2) Score:", r2_svr)
print("Mean Absolute Error (MAE):", mae_svr)
```

```
Mean Squared Error (MSE): 0.6585696158463452
R-squared (R2) Score: 0.39658650851038113
Mean Absolute Error (MAE): 0.18386224685395017
```

Neural Network for Regression model

In [ ]:
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,BatchNormalization
```

In [ ]:
```python
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```python
model = Sequential([
    Dense(500, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(250, activation='relu'),
    Dense(125, activation='relu'),
    Dense(75, activation='relu'),
    Dense(1)  # Output layer for regression
])
```

In [ ]:
```python
# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')
```

In [ ]:
```python
# Train the model
history=model.fit(X_train_scaled, y_train, epochs=48, verbose=1, validation_split=0.2)
```

```
Epoch 1/48
160/160 [==============================] - 3s 10ms/step - loss: 0.6472 - val_loss: 0.228
5
Epoch 2/48
160/160 [==============================] - 1s 8ms/step - loss: 0.4404 - val_loss: 0.2600
Epoch 3/48
160/160 [==============================] - 1s 9ms/step - loss: 0.5477 - val_loss: 0.1870
Epoch 4/48
160/160 [==============================] - 1s 8ms/step - loss: 0.4729 - val_loss: 0.2477
Epoch 5/48
160/160 [==============================] - 1s 8ms/step - loss: 0.3646 - val_loss: 0.1872
Epoch 6/48
160/160 [==============================] - 1s 7ms/step - loss: 0.4672 - val_loss: 0.2424
Epoch 7/48
160/160 [==============================] - 1s 8ms/step - loss: 0.4143 - val_loss: 0.2052
Epoch 8/48
160/160 [==============================] - 2s 12ms/step - loss: 0.2907 - val_loss: 0.174
5
Epoch 9/48
160/160 [==============================] - 2s 11ms/step - loss: 0.1240 - val_loss: 0.175
7
Epoch 10/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2364 - val_loss: 0.2705
Epoch 11/48
160/160 [==============================] - 1s 7ms/step - loss: 0.1988 - val_loss: 0.2298
Epoch 12/48
160/160 [==============================] - 1s 7ms/step - loss: 0.3717 - val_loss: 0.1693
Epoch 13/48
160/160 [==============================] - 1s 8ms/step - loss: 0.3055 - val_loss: 0.2131
Epoch 14/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2749 - val_loss: 0.2184
Epoch 15/48
160/160 [==============================] - 1s 7ms/step - loss: 0.1955 - val_loss: 0.4423
Epoch 16/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2502 - val_loss: 0.2175
Epoch 17/48
160/160 [==============================] - 2s 9ms/step - loss: 0.1516 - val_loss: 0.2062
Epoch 18/48
160/160 [==============================] - 2s 12ms/step - loss: 0.2951 - val_loss: 0.237
2
Epoch 19/48
160/160 [==============================] - 1s 9ms/step - loss: 0.3731 - val_loss: 0.1810
Epoch 20/48
160/160 [==============================] - 1s 8ms/step - loss: 0.3295 - val_loss: 0.1856
Epoch 21/48
160/160 [==============================] - 1s 7ms/step - loss: 0.3094 - val_loss: 0.1896
Epoch 22/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2605 - val_loss: 0.2106
Epoch 23/48
160/160 [==============================] - 1s 7ms/step - loss: 0.2051 - val_loss: 0.2218
Epoch 24/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2237 - val_loss: 0.1789
Epoch 25/48
160/160 [==============================] - 1s 8ms/step - loss: 0.1582 - val_loss: 0.1959
Epoch 26/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2354 - val_loss: 0.1722
Epoch 27/48
160/160 [==============================] - 2s 13ms/step - loss: 0.0812 - val_loss: 0.172
2
Epoch 28/48
160/160 [==============================] - 2s 10ms/step - loss: 0.0882 - val_loss: 0.206
5
Epoch 29/48
160/160 [==============================] - 1s 8ms/step - loss: 0.1799 - val_loss: 0.1743
```

```
Epoch 30/48
160/160 [==============================] - 1s 8ms/step - loss: 0.0724 - val_loss: 0.1662
Epoch 31/48
160/160 [==============================] - 1s 7ms/step - loss: 0.1129 - val_loss: 0.2224
Epoch 32/48
160/160 [==============================] - 1s 8ms/step - loss: 0.1365 - val_loss: 0.1765
Epoch 33/48
160/160 [==============================] - 1s 8ms/step - loss: 0.2135 - val_loss: 0.1667
Epoch 34/48
160/160 [==============================] - 1s 8ms/step - loss: 0.1019 - val_loss: 0.1596
Epoch 35/48
160/160 [==============================] - 1s 8ms/step - loss: 0.0626 - val_loss: 0.1671
Epoch 36/48
160/160 [==============================] - 1s 9ms/step - loss: 0.0690 - val_loss: 0.1762
Epoch 37/48
160/160 [==============================] - 3s 19ms/step - loss: 0.0867 - val_loss: 0.173
0
Epoch 38/48
160/160 [==============================] - 1s 9ms/step - loss: 0.1412 - val_loss: 0.1660
Epoch 39/48
160/160 [==============================] - 1s 8ms/step - loss: 0.0478 - val_loss: 0.1651
Epoch 40/48
160/160 [==============================] - 1s 9ms/step - loss: 0.0437 - val_loss: 0.1931
Epoch 41/48
160/160 [==============================] - 1s 9ms/step - loss: 0.0480 - val_loss: 0.1539
Epoch 42/48
160/160 [==============================] - 1s 8ms/step - loss: 0.0634 - val_loss: 0.1618
Epoch 43/48
160/160 [==============================] - 1s 7ms/step - loss: 0.1384 - val_loss: 0.1722
Epoch 44/48
160/160 [==============================] - 1s 7ms/step - loss: 0.0528 - val_loss: 0.1796
Epoch 45/48
160/160 [==============================] - 2s 11ms/step - loss: 0.0586 - val_loss: 0.186
7
Epoch 46/48
160/160 [==============================] - 2s 11ms/step - loss: 0.0806 - val_loss: 0.163
1
Epoch 47/48
160/160 [==============================] - 1s 8ms/step - loss: 0.1065 - val_loss: 0.1747
Epoch 48/48
160/160 [==============================] - 1s 8ms/step - loss: 0.1907 - val_loss: 0.1747
```

In [ ]:

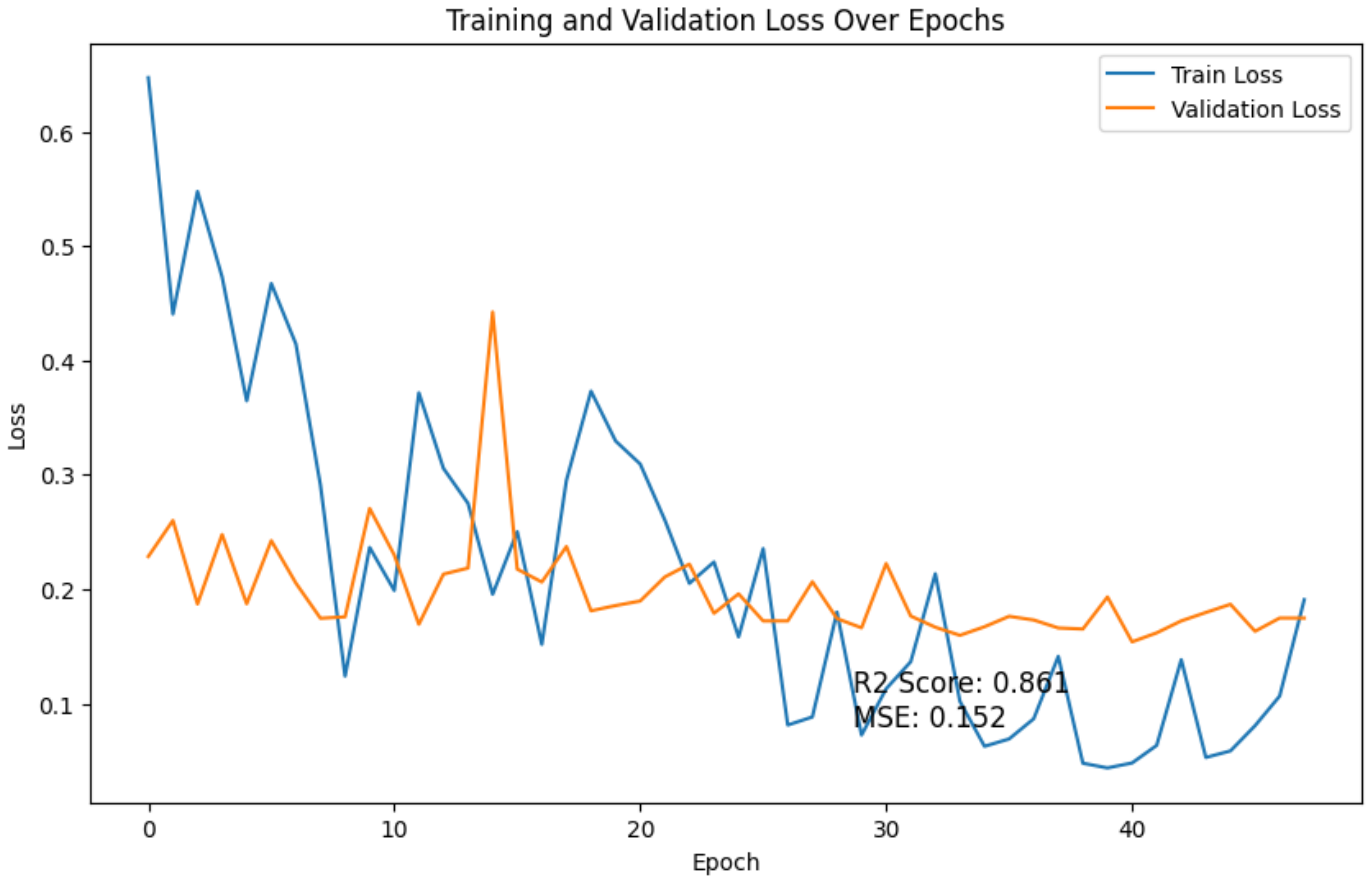
```python
predictions = model.predict(X_test_scaled)

# Calculate evaluation metrics
mse_nn = mean_squared_error(y_test, predictions)
r2_nn = r2_score(y_test, predictions)
mae_nn = mean_absolute_error(y_test, predictions)

print("Mean Squared Error (MSE):", mse)
print("R-squared (R2) Score:", r2)
print("Mean Absolute Error (MAE):", mae)
```

```
63/63 [==============================] - 0s 2ms/step
Mean Squared Error (MSE): 0.15156845184525997
R-squared (R2) Score: 0.8611256175095645
Mean Absolute Error (MAE): 0.198662905045953
```

In [ ]:

```python
# Plot training and validation loss over epochs
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
raining and Validation Loss Over Epochs')
```

```
plt.legend()

# Add evaluation metrics to the plot
plt.annotate(f'R2 Score: {r2:.3f}\nMSE: {mse:.3f}', xy=(0.6, 0.1), xycoords='axes fracti
plt.show()
```



DataFrame

In [ ]:

```
# Neural Network model
nn_predictions = model.predict(X_test_scaled)
mse_nn = mean_squared_error(y_test, nn_predictions)
r2_nn = r2_score(y_test, nn_predictions)
mae_nn = mean_absolute_error(y_test, nn_predictions)

# SVR model
svr_predictions = svr_model.predict(X_test)
mse_svr = mean_squared_error(y_test, svr_predictions)
r2_svr = r2_score(y_test, svr_predictions)
mae_svr = mean_absolute_error(y_test, svr_predictions)

# Stacking model
mse_stacking = mean_squared_error(y_test, y_pred_stacking_test)
r2_stacking = r2_score(y_test, y_pred_stacking_test)
mae_stacking = mean_absolute_error(y_test, y_pred_stacking_test)

# Decision Tree model
dt_predictions = decision_tree_regressor.predict(X_test)
mse_dt = mean_squared_error(y_test, dt_predictions)
rmse_dt = np.sqrt(mse_dt)
mae_dt = mean_absolute_error(y_test, dt_predictions)
r2_dt = r2_score(y_test, dt_predictions)

# Random Forest model
ndom_RandomForest_Regressor.predict(X_test)
```

```
mse_rf = mean_squared_error(y_test, rfr_test)
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test, rfr_test)
r2_rf = r2_score(y_test, rfr_test)

# Create a DataFrame to compare evaluation metrics
data = {
    'Model': ['Neural Network', 'SVR', 'Stacking', 'Decision Tree', 'Random Forest'],
    'Mean Squared Error (MSE)': [mse_nn, mse_svr, mse_stacking, mse_dt, mse_rf],
    'R-squared (R2) Score': [r2_nn, r2_svr, r2_stacking, r2_dt, r2_rf],
    'Mean Absolute Error (MAE)': [mae_nn, mae_svr, mae_stacking, mae_dt, mae_rf],
}

# Create a DataFrame and sort it by Mean Squared Error (MSE)
df = pd.DataFrame(data)
df_sorted = df.sort_values(by='Mean Squared Error (MSE)')
```

```
63/63 [==============================] - 1s 8ms/step
```

In [ ]:
```
df_sorted
```

Out[ ]:

	Model	Mean Squared Error (MSE)	R-squared (R2) Score	Mean Absolute Error (MAE)
0	Neural Network	0.151568	0.861126	0.198663
2	Stacking	0.223731	0.795007	0.131378
4	Random Forest	0.225219	0.793644	0.143704
3	Decision Tree	0.258240	0.763388	0.164469
1	SVR	0.658570	0.396587	0.183862

In [ ]:
```

```