Hieu Hoang, Hasan Khan

Data 3401 Project

SuperStore Data Analysis

Introduction

Running a profitable superstore is different from other retail business in many ways as a superstore has a huge range of products to market unlike small retail stores where the product category range is relatively limited. The goal of our analyses is understanding the characteristics of the company sales and profit and therefore improve the company profit. Eventually after getting insights on how the sales and profit vary, we will try to make a prediction model to predict sales.

- First we need to understand who buys what and where to gain an insight of the companys customer base
- To Perform Exploratory Data Analysis
- Identify key strong and weak areas for improving profits

The Problem Statement

We will need to take a look to see trends, for data collected in the Superstore.

- What is the overall sales trend?
- What are the most selling products?
- What is the most preferred ship mode?
- Which are the most profitable category and sub-category?

Table of contents:

- Geographical analysis
- Top sellers and categories
- Sales, sold items & profits
- Shipping insights
- Machine Learning
- Conclusion

Data characteristics

Information provided in dataset:

- Row ID - unique identifier of the record
- Order ID - identifier of particular order
- Order Date - purchase order timestamp
- Ship Date - delivery timestamp
- Ship Mode - picked delivery option
- Customer ID - unique identifier of the customer
- Customer Name - name & and surname of customer
- Segment - customer's segment e.g. customer classification
- Country - customer country (data only for US)
- City - customer city
- State - customer state
- Postal Code - unique identifier of the customer localization
- Region - particular region of the US
- Product ID - product identifier
- Category - product main category
- Sub-Category - additional category of the product
- Sales - sum of sales for order
- Quantity - amount of the product
- Discount - discount rate
- Profit - total profit from order

In [1]:

```
#Import Libraries
```

```
import tensorflow as tf
import pandas as pd
import seaborn as sns
import numpy as np
import csv
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# This part of the code is just to supress any unwanted warnings that may appear

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#Import csv
df = pd.read_csv('Superstore.csv',encoding='cp1252')
```

- The default encoding is platform dependent, but any text encoding supported by Python can be used.

In [3]:

```
#Overview of data
df.head()
```

Out[3]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Catego
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-10001798	Furnitu

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Catego
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-10000454	Furnitu
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-10000240	Offi Suppli
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-10000577	Furnitu
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-10000760	Offi Suppli

5 rows × 21 columns

Exploratory Data Analysis

In [4]:

```
#Observation
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
```

- The total number of non null count is a 9,994, which matches the number of row. Therefore, there are no missing values.

In [5]:

```
#Analyzing the numerical attributes
df.describe()
```

Out[5]:

	Row ID	Postal Code	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	4997.500000	55190.379428	229.858001	3.789574	0.156203	28.656896
std	2885.163629	32063.693350	623.245101	2.225110	0.206452	234.260108
min	1.000000	1040.000000	0.444000	1.000000	0.000000	-6599.978000
25%	2499.250000	23223.000000	17.280000	2.000000	0.000000	1.728750
50%	4997.500000	56430.500000	54.490000	3.000000	0.200000	8.666500
75%	7495.750000	90008.000000	209.940000	5.000000	0.200000	29.364000
max	9994.000000	99301.000000	22638.480000	14.000000	0.800000	8399.976000

In [6]:
```
#Check to see if we have any columns with null values
df.isnull().sum()
```

Out[6]:
```
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 [7]:
```
#Check the shape of the data frame
df.shape
```

Out[7]:
```
(9994, 21)
```

In [8]:
```
#Check the unique values in each column
df.nunique()
```

Out[8]:
```
Row ID           9994
Order ID         5009
Order Date       1237
Ship Date        1334
Ship Mode           4
Customer ID       793
Customer Name     793
Segment             3
Country             1
```

```
City             531
State             49
Postal Code      631
Region             4
Product ID      1862
Category           3
Sub-Category      17
Product Name    1850
Sales           5825
Quantity          14
Discount          12
Profit          7287
dtype: int64
```

In [9]:
```
#Cleaning up the data by changing the format of date column
df['Order Date']= df['Order Date'].str.replace('/','-')
df['Ship Date']= df['Ship Date'].str.replace('/','-')
df.head()
```

Out[9]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Category	Su Categ
0	1	CA-2016-152156	11-8-2016	11-11-2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-10001798	Furniture	Bookca
1	2	CA-2016-152156	11-8-2016	11-11-2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-10000454	Furniture	Ch
2	3	CA-2016-138688	6-12-2016	6-16-2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-10000240	Office Supplies	Lal
3	4	US-2015-108966	10-11-2015	10-18-2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-10000577	Furniture	Tal

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Category	Su Categ
4	5	US-2015-108966	10-11-2015	10-18-2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-10000760	Office Supplies	Stora

5 rows × 21 columns

In [10]:

```
#Cleaning up the data by changing the format of date column
df.rename(columns={'Order Date': 'Order_date'},inplace=True)
df.rename(columns={'Ship Date': 'Ship_date'},inplace=True)
df.head()
```

Out[10]:

	Row ID	Order ID	Order_date	Ship_date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Categor
0	1	CA-2016-152156	11-8-2016	11-11-2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-BO-10001798	Furnitur
1	2	CA-2016-152156	11-8-2016	11-11-2016	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420	South	FUR-CH-10000454	Furnitur
2	3	CA-2016-138688	6-12-2016	6-16-2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036	West	OFF-LA-10000240	Offic Supplie
3	4	US-2015-108966	10-11-2015	10-18-2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	FUR-TA-10000577	Furnitur

	Row ID	Order ID	Order_date	Ship_date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code	Region	Product ID	Categor
4	5	US-2015-108966	10-11-2015	10-18-2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311	South	OFF-ST-10000760	Offic Supplie

5 rows × 21 columns

In [11]:

```
#Print the maximum an minimum of the order and ship date columns to see the range of date this data covers
print(df['Order_date'].max())
print(df['Order_date'].min())
print(df['Ship_date'].max())
print(df['Ship_date'].min())

#Our data is taken from Jan 2015 to Sep 2017
#add metadata
```

```
9-9-2017
1-1-2017
9-9-2017
1-1-2015
```

In [12]:

```
df.dtypes
```

Out[12]:

```
Row ID              int64
Order ID           object
Order_date         object
Ship_date          object
Ship Mode          object
Customer ID        object
Customer Name      object
Segment            object
Country            object
City               object
State              object
Postal Code         int64
Region             object
Product ID         object
Category           object
Sub-Category       object
```

```
Product Name      object
Sales            float64
Quantity           int64
Discount         float64
Profit           float64
dtype: object
```

In [13]:

```
#Calling out all columns in dataset
df.columns
```

Out[13]:

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

In [14]:

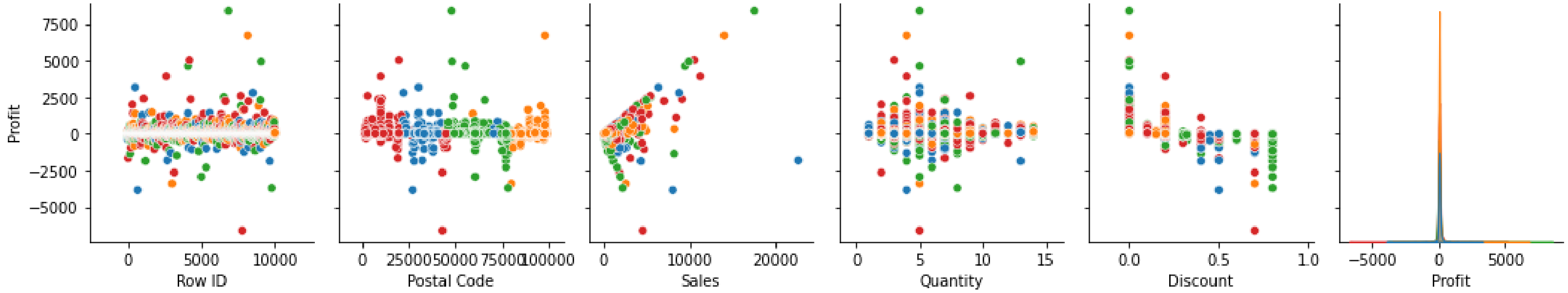
```
#Now change the 'Order_date' and 'Ship_date' dtype to datetime64[ns]
df['Order_date']= pd.to_datetime(df.Order_date)
df['Ship_date']= pd.to_datetime(df.Ship_date)
df.dtypes
```

Out[14]:

```
Row ID                    int64
Order ID                 object
Order_date       datetime64[ns]
Ship_date        datetime64[ns]
Ship Mode                object
Customer ID              object
Customer Name            object
Segment                  object
Country                  object
City                     object
State                    object
Postal Code               int64
Region                   object
Product ID               object
Category                 object
Sub-Category             object
Product Name             object
Sales                   float64
Quantity                  int64
Discount                float64
Profit                  float64
dtype: object
```

In [15]:
```
#Pairplot of the sales data
sns.pairplot(df,hue='Region')
```

Out[15]: <seaborn.axisgrid.PairGrid at 0x231911af088>

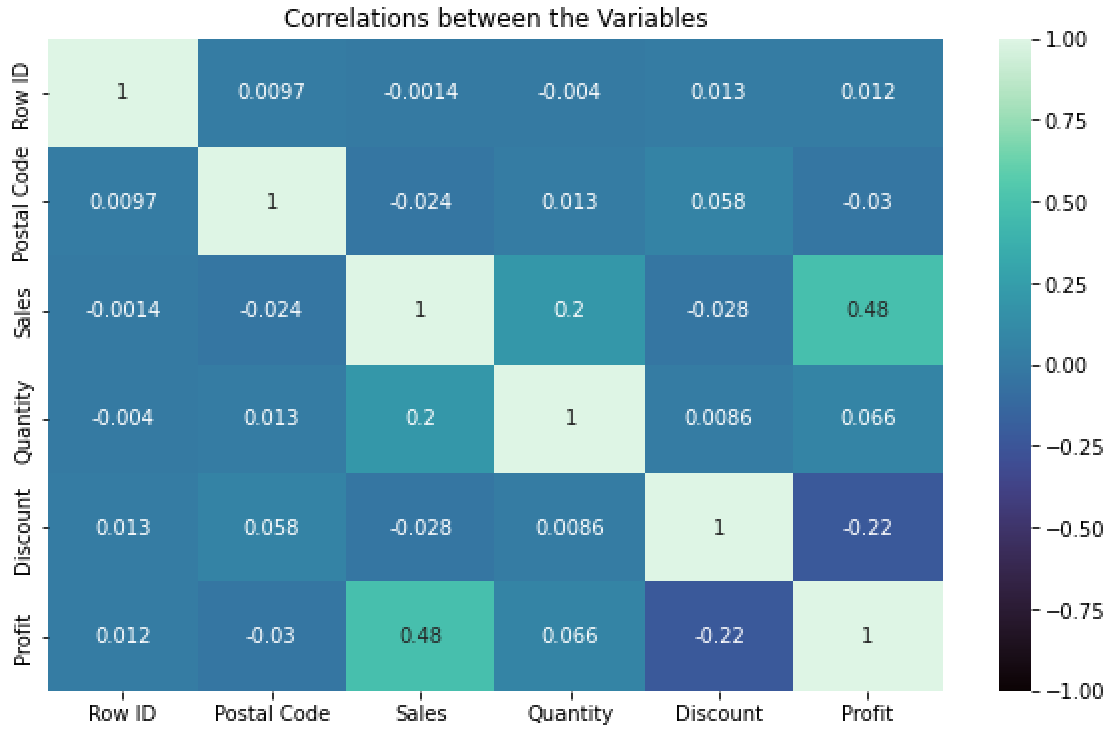


Observation of Pairplot

- In above we see that there is some relation between sales and profit and also there is some relation between Discount and Profit.
- Now To see what exact relation between those entities we plot the heat_map. so we get more clearity.

In [16]:

```
#Heatmap of the Sales data
corr = df.corr()
plt.figure(figsize=(10, 6))
sns.heatmap(corr, annot=True, vmin=-1.0, cmap='mako')
plt.title('Correlations between the Variables')
plt.show()
```



Observation of Heatmap

From Above map we infer that,

- There is a postive correlation with sales and profits.
- Discount and profit have a negative correlation.
- Quantity and Profit are less moderately correlated.

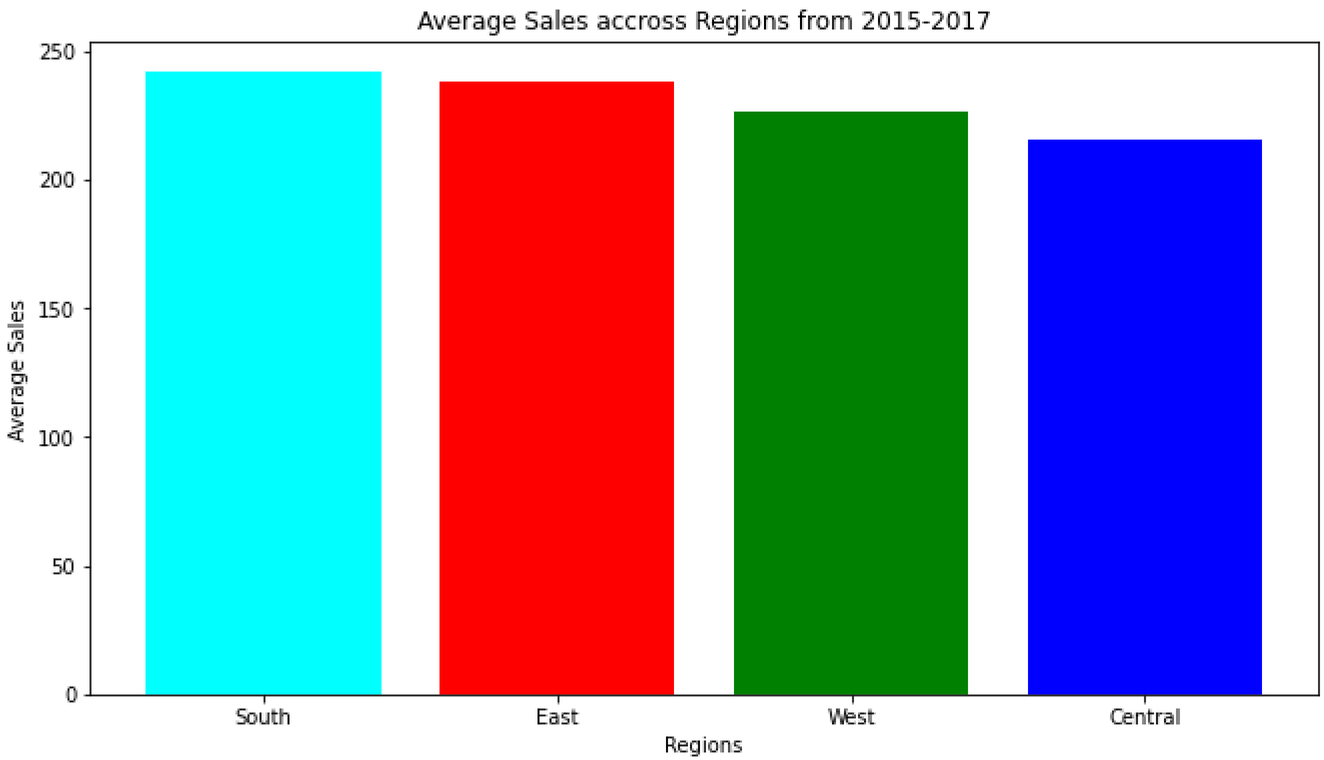
Sales Across the United States

- Suppose we are hired by this Superstore. Our employers are curious to understand the average Sales across different Regions. We will address the request by creating a bar plot because we have 'Regions' as your categorical feature and 'Sales' will be our numerical feature.

In [17]:

```
#Lets start by creating a subset of our dataframe therefore its easier to work with our target variables.
df_salereg = df[['Region','Sales']]
df_salereg = df_salereg.groupby('Region').mean().sort_values(by='Sales', ascending=False)

fig, ax = plt.subplots(figsize=(11,6))
ax.bar(df_salereg.index,df_salereg.Sales, color=['cyan', 'red', 'green', 'blue'])
ax.set_xlabel('Regions')
ax.set_ylabel('Average Sales')
ax.set_title('Average Sales accross Regions from 2015-2017')
plt.show()
```



- After close inspection, we can conclude that Sales/Revenue on average tend to be highest in the South Region and lowest in the Central Region.

In [18]:
```
#Average for revenue/sales for region wise
df_avg= df.groupby("Region")["Sales"].mean()
df_avg
```

Out[18]:
```
Region
Central    215.772661
East       238.336110
South      241.803645
West       226.493233
Name: Sales, dtype: float64
```

Linechart for average sales overtime

- Our employers are now trying to track their progress overtime. They want to learn about the trend of their average Sales and Profit over the years they have been open. We will show this trend on a line chart as this will be the best plot to display a trend over the years for both Sales and Profit on the same plot.

In [19]:
```
#Average Sales and Profit subset of dataframe
df_trend = df[['Order_date','Sales','Profit']].sort_values('Order_date') # Chronological Ordering
df_trend = df_trend.groupby('Order_date').mean() # Groupby to get the average Sales and Profit on each day
df_trend.head()
```

Out[19]:

	Sales	Profit
Order_date		
2014-01-03	16.448000	5.551200
2014-01-04	96.020000	-21.996700
2014-01-05	19.536000	4.884000
2014-01-06	489.677778	150.894711
2014-01-07	43.579000	-35.981050

In [20]:
```
#Define a function called plot_trend incase we need a similar plot later to plot a linechart
def plot_trend(axes, x, y, color, xlabel, ylabel):

  #Plot the inputs x,y in the provided color
  axes.plot(x,y, color= color)

  #Set the x-axis label
  axes.set_xlabel(xlabel)
```

```
    #Set the y-axis label
    axes.set_ylabel(ylabel,color=color)
    #Set the colors tick params for y-axis
    axes.tick_params('y', colors= color)
```

In [21]:

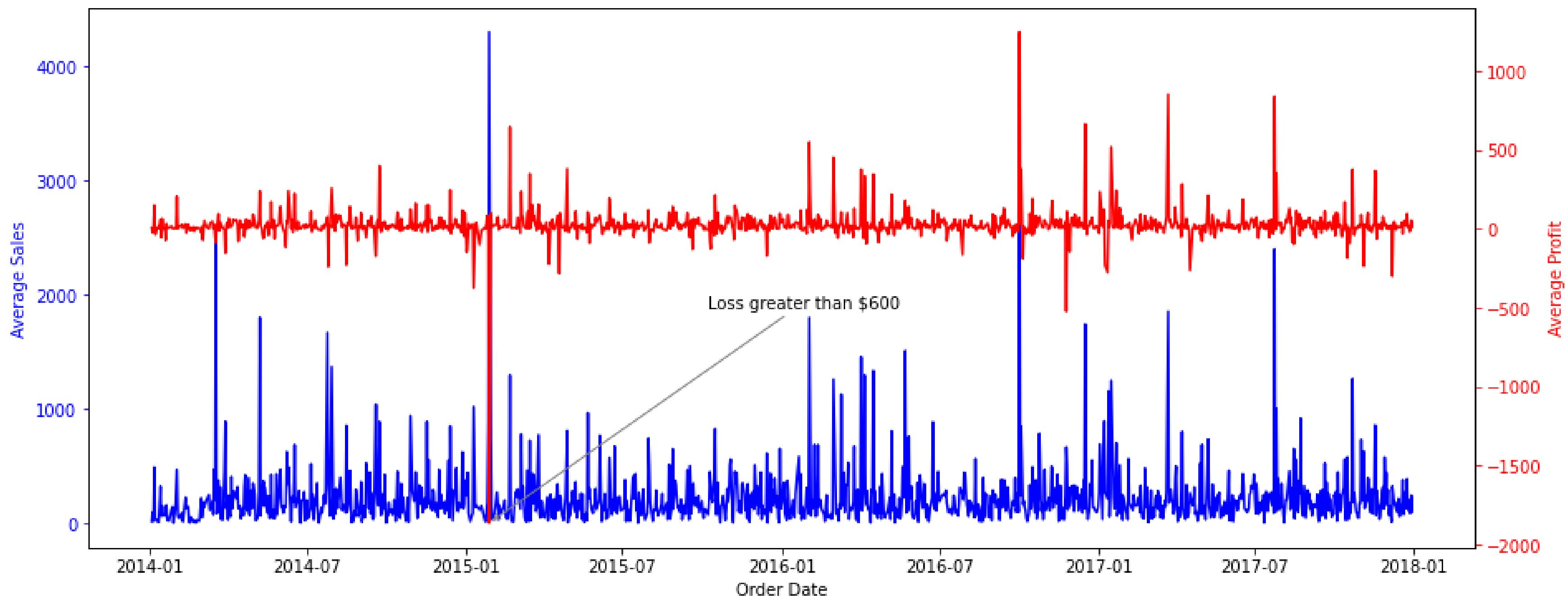
```
fig, ax = plt.subplots(figsize=(15,6))

#Plot
plot_trend(ax,df_trend.index,df_trend.Sales, "blue",'Order Date','Average Sales')

#Create a twin Axes object that shares the x-axis
ax2 = ax.twinx()

#Plot the relative Profit data in red
plot_trend(ax2,df_trend.index,df_trend.Profit, "red",'Order Date','Average Profit')

ax2.annotate("Loss greater than $600",
xy=(pd.Timestamp('2015-01-28'),-1862.3124),xytext=(pd.Timestamp('2015-10-07'),-500),arrowprops={'arrowstyle': '->', 'colc
plt.show()
```



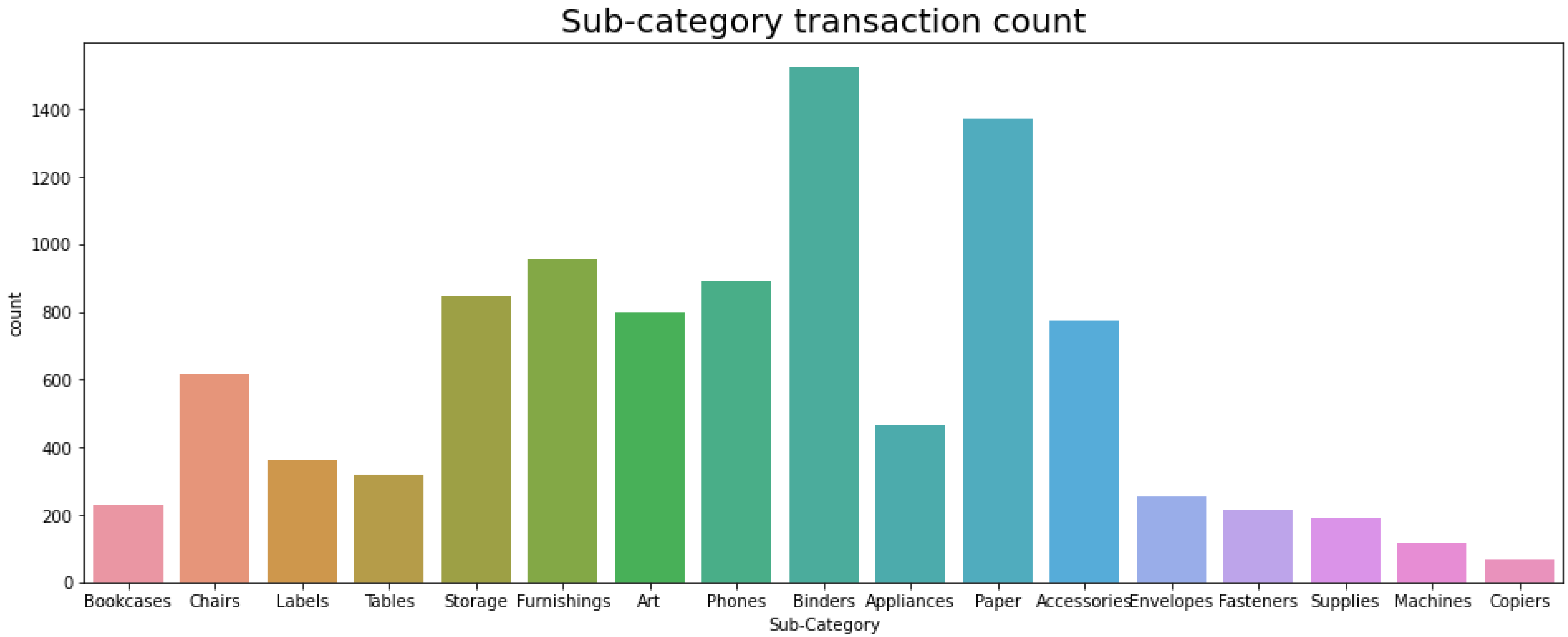
Observation on the linechart

- looks like there is a point where the average loss is greater 600 and we would like to point it out on our graph.
- lets find out the date where the avg profit is less than 600 as it can be crucial information to avoid loss in the future.
- Order_date : 2015-01-28. Now lets point out this information on our graph.
- normally you would assume that with greater sales we would expect greater profit.
- There can be a bunch of reasons as to why that is for example that transcation can be where we offered the most discount resulting in loss.
- Our employers can greatly benefit from this observation to make important business decisions in the future.

Sub-category transactions for all products

In [22]:
```
plt.figure(figsize=(16,6))
sns.countplot(df['Sub-Category'])
plt.title('Sub-category transaction count',fontsize=20)
```

Out[22]:
```
Text(0.5, 1.0, 'Sub-category transaction count')
```



In [23]:

```
#How much each sub-category brought in revenue, rounded to 1 decimal point
dfsub = df.groupby(["Sub-Category"]).sum().sort_values("Sales", ascending=False).head(20)
dfsub = dfsub[["Sales"]].round(1)
dfsub.reset_index(inplace=True)
dfsub
```

Out[23]:

	Sub-Category	Sales
0	Phones	330007.1
1	Chairs	328449.1
2	Storage	223843.6
3	Tables	206965.5
4	Binders	203412.7
5	Machines	189238.6
6	Accessories	167380.3

	Sub-Category	Sales
7	Copiers	149528.0
8	Bookcases	114880.0
9	Appliances	107532.2
10	Furnishings	91705.2
11	Paper	78479.2
12	Supplies	46673.5
13	Art	27118.8
14	Envelopes	16476.4
15	Labels	12486.3
16	Fasteners	3024.3

Observations from the 'Sub-Category transaction' plot above

- Top item sold are binder, followed by papers.
- The two items that are sold the least, copiers and machines.
- Top gross selling items are phones and chairs.
- The items that are sold the least are labels and fasteners.

In [24]:

```
#Proportions of the sub-category being sold
df['Sub-Category'].value_counts(normalize=True)

#Binders are the top product being sold around 15% of the total orders
```

Out[24]:

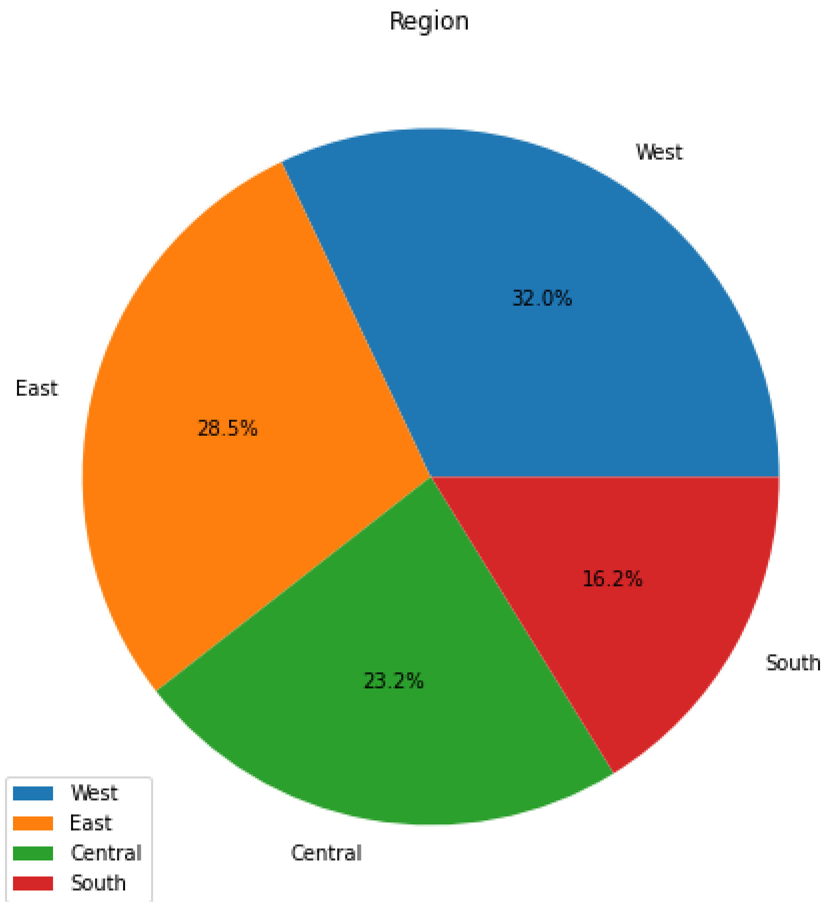
```
Binders        0.152391
Paper          0.137082
Furnishings    0.095757
Phones         0.088953
Storage        0.084651
Art            0.079648
Accessories    0.077547
Chairs         0.061737
```

```
Appliances     0.046628
Labels         0.036422
Tables         0.031919
Envelopes      0.025415
Bookcases      0.022814
Fasteners      0.021713
Supplies       0.019011
Machines       0.011507
Copiers        0.006804
Name: Sub-Category, dtype: float64
```

Number of customers in selected regions

In [25]:

```
#Number of customers in selected regions
plt.figure(figsize=(8,8))
plt.title('Region')
plt.pie(df['Region'].value_counts(), labels=df['Region'].value_counts().index,autopct='%1.1f%%')
plt.legend()
plt.show()
```



- West has 32% of the entire country transactions, which is the highest among all regions. The tells us that our primary customer base is in the West.
- The least amount of transaction is in the South, at 16.2%.

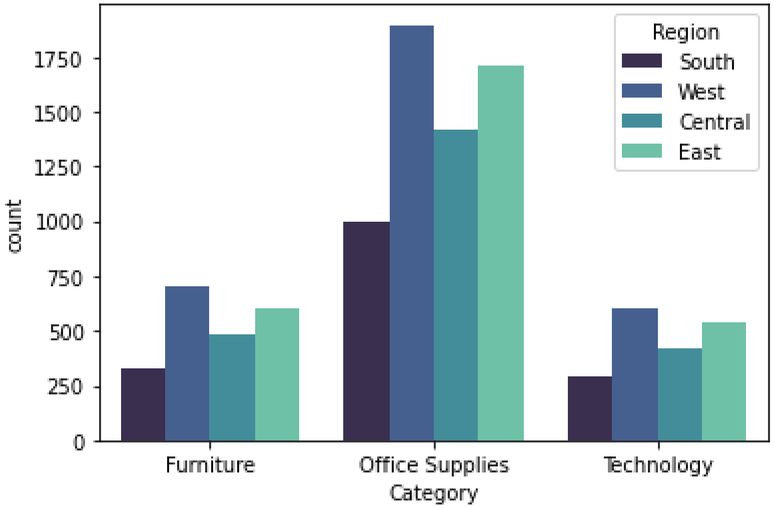
Countplot to compare Categories and Regions in United States

In [26]:

```
#let's plot some graphs for better undertsanding and visualization
sns.countplot(data = df , x = 'Category' , hue = 'Region'  , palette = 'mako')
```

Out[26]:

```
<AxesSubplot:xlabel='Category', ylabel='count'>
```



- The western regions selling the most in all three categories.
- We will need to know if all cities in the west coast sell more than the other regions.

Top 20 states with the highest gross sale & profit

In [27]:

```
#lets review sales by state, on the second line we are rounding to 1 decimal point
#States sales are listed in the descending order

dfstate = df.groupby(["State"]).sum().sort_values("Sales", ascending=False).head(20)
dfstate = dfstate[["Sales"]].round(1)
dfstate.reset_index(inplace=True)
dfstate
```

Out[27]:

	State	Sales
0	California	457687.6
1	New York	310876.3
2	Texas	170188.0
3	Washington	138641.3
4	Pennsylvania	116511.9

	State	Sales
5	Florida	89473.7
6	Illinois	80166.1
7	Ohio	78258.1
8	Michigan	76269.6
9	Virginia	70636.7
10	North Carolina	55603.2
11	Indiana	53555.4
12	Georgia	49095.8
13	Kentucky	36591.8
14	New Jersey	35764.3
15	Arizona	35282.0
16	Wisconsin	32114.6
17	Colorado	32108.1
18	Tennessee	30661.9
19	Minnesota	29863.2

In [28]:

```
#Most ordered item per city
df["City"].value_counts()
```

Out[28]:

```
New York City      915
Los Angeles        747
Philadelphia       537
San Francisco      510
Seattle            428
                  ...
Glenview             1
Missouri City        1
Rochester Hills      1
Palatine             1
Manhattan            1
Name: City, Length: 531, dtype: int64
```

In [29]:

```
#City min order
df["City"].value_counts().min()
```

Out[29]:

```
1
```

In [30]:

```
#City max order
df["City"].value_counts().max()
```

Out[30]:

```
915
```

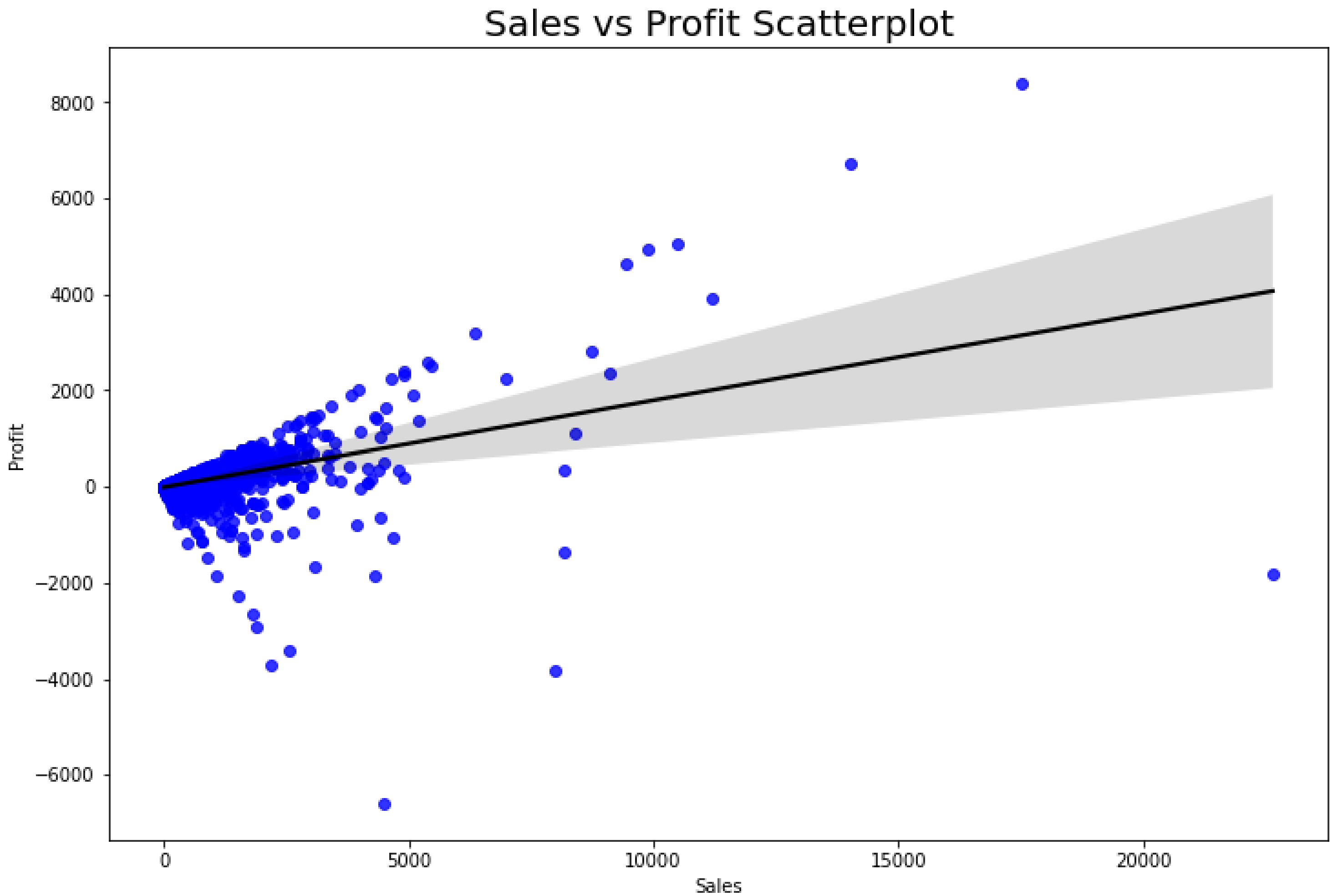
Scatterplot for profits

In [31]:

```
#Scatterplot sales vs profits
fig,axes=plt.subplots(1,1,figsize=(12,8))
sns.regplot(x='Sales', y='Profit', data=df, scatter_kws={"color":"blue"}, line_kws={'color':"black"})
plt.title('Sales vs Profit Scatterplot',fontsize=20)
```

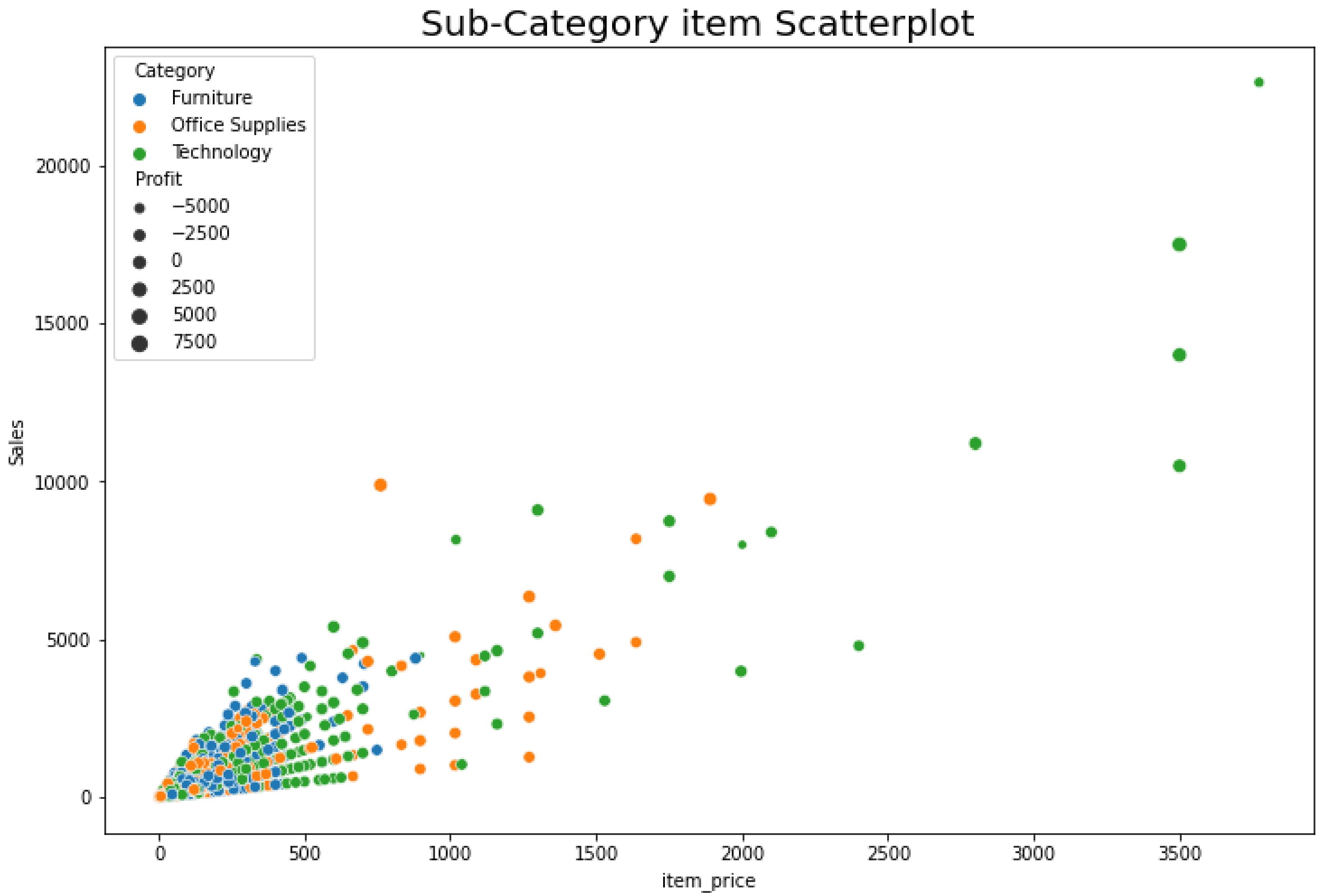
Out[31]:

```
Text(0.5, 1.0, 'Sales vs Profit Scatterplot')
```



In [32]:
```
#Scatterplot for Sub Category
df['item_price']= (df.Sales/df.Quantity)
fig,axes=plt.subplots(1,1,figsize=(12,8))
sns.scatterplot(x='item_price',y='Sales',hue='Category',size='Profit',data=df)
plt.title('Sub-Category item Scatterplot  ',fontsize=20)
```

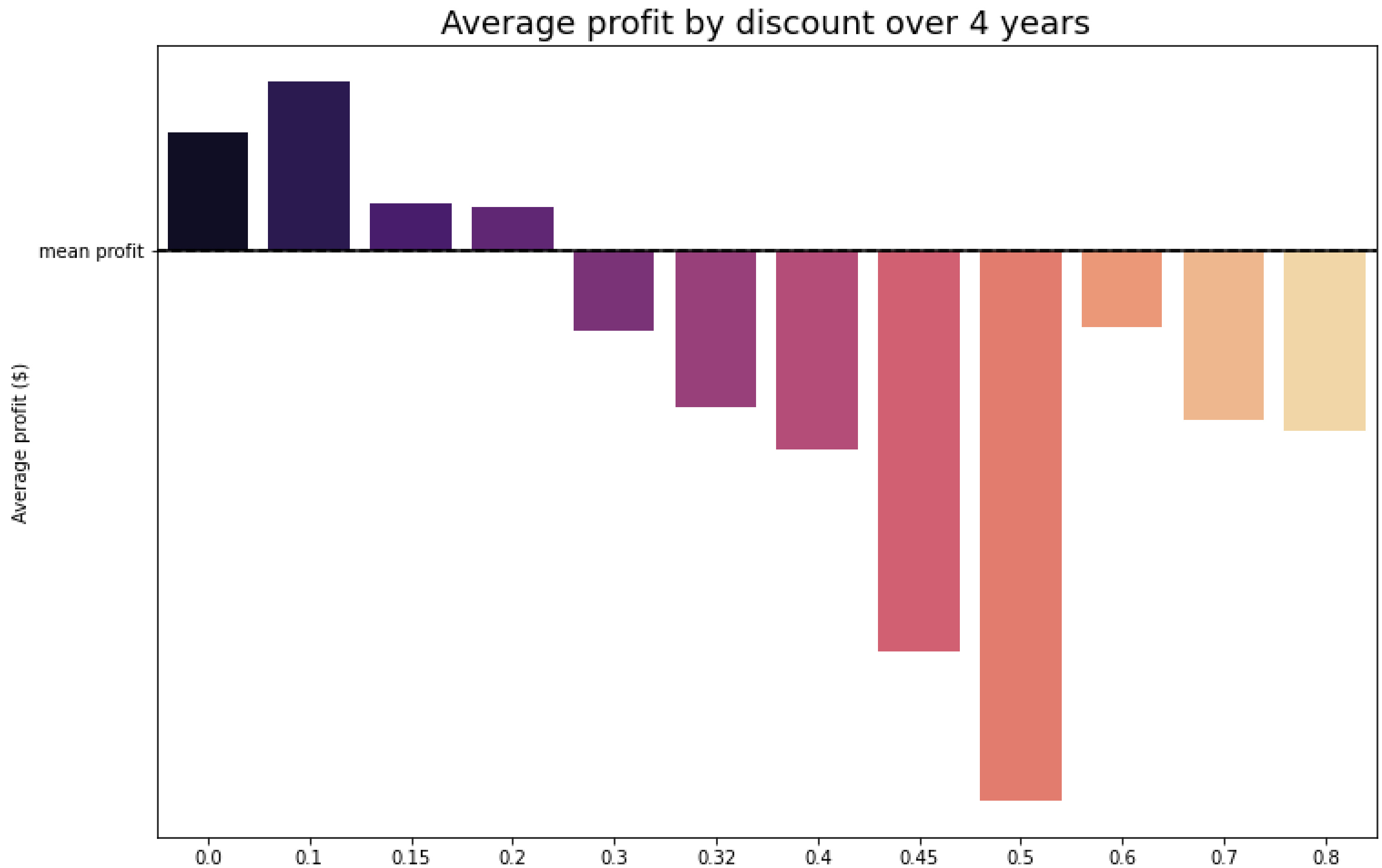
Out[32]:
```
Text(0.5, 1.0, 'Sub-Category item Scatterplot  ')
```



- The scatterplot shows a positive correlated Sales and item price

In [33]:

```
#Average profit by discount over 4 years
fig,axes=plt.subplots(1,1,figsize=(12,8))
ax = sns.barplot(x = "Discount", y = "Profit", data = df, palette = 'magma',ci=None)
ax.set_title("Average profit by discount over 4 years", fontsize = 18)
plt.axhline(y = 'mean profit', color='k', linestyle='--')
plt.axhline(y = 0, color='k', linestyle='-')
plt.xlabel("")
plt.ylabel("Average profit ($)")
plt.show()
```



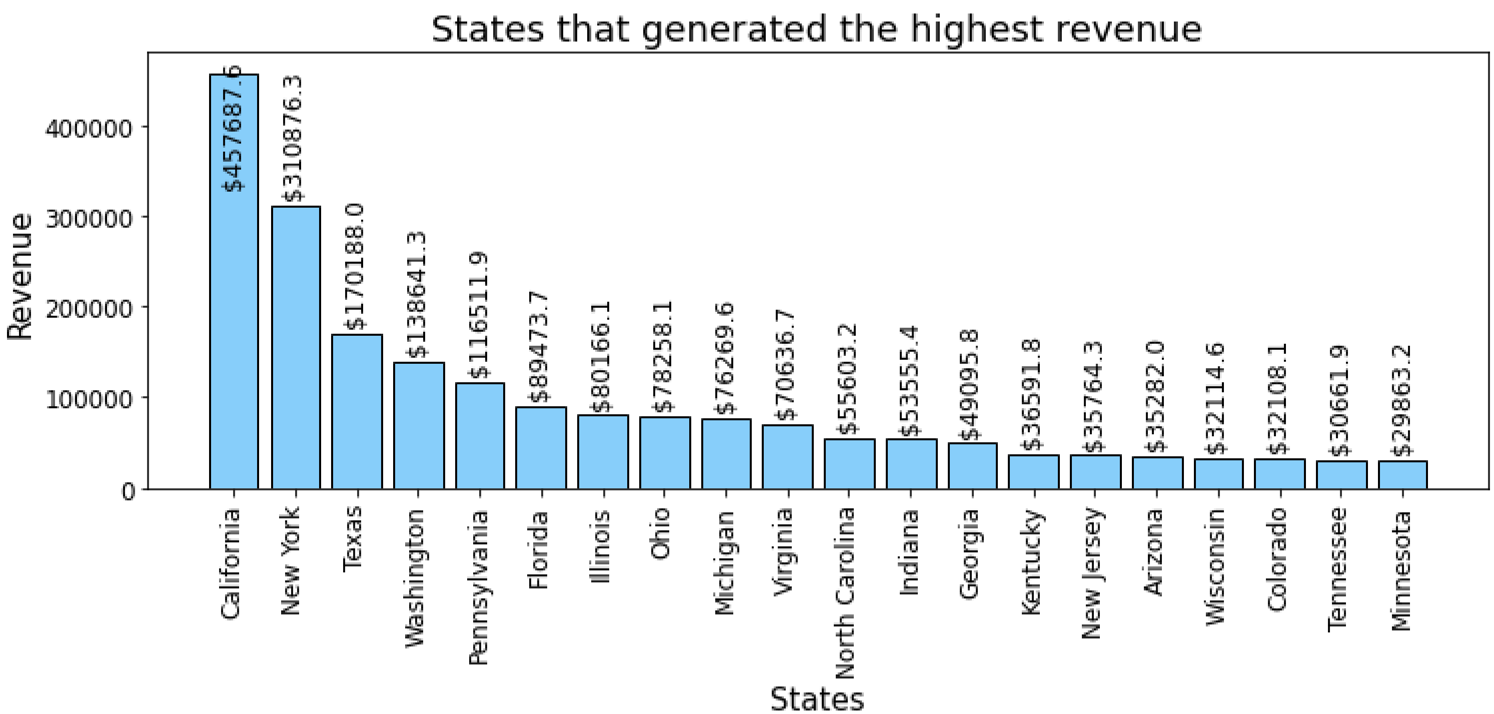
- We can see that sales made without discount and with 20% of discount are more profitable. Higher discount had negative profit.

Sales by states

In [34]:

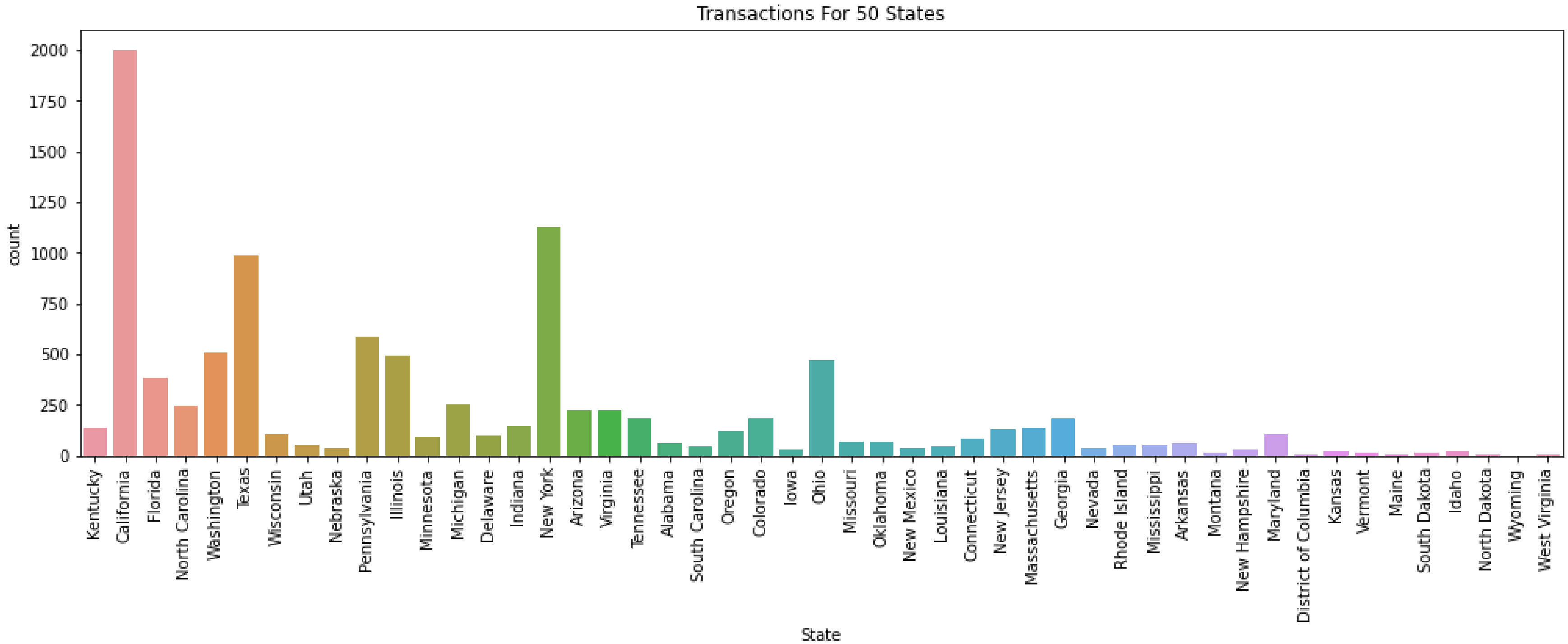
```
#Plotting the sales by state table
plt.figure(figsize = (12,4)) # width and height of figure is defined in inches
plt.title("States that generated the highest revenue", fontsize=18)
plt.bar(dfstate["State"], dfstate["Sales"],color= '#87CEFA',edgecolor='black', linewidth = 1)
plt.xlabel("States",fontsize=15) # x axis shows the States
plt.ylabel("Revenue",fontsize=15) # y axis shows the Revenue
plt.xticks(fontsize=12, rotation=90)
```

```
plt.yticks(fontsize=12)
for k,v in dfstate["Sales"].items(): #To show the exact revenue generated on the figure
    if v>400000:
        plt.text(k,v-120000,'$'+ str(v), fontsize=12,rotation=90,color='k', horizontalalignment='center');
    else:
        plt.text(k,v+15000,'$'+ str(v), fontsize=12,rotation=90,color='k', horizontalalignment='center');
```



In [35]:
```
#Top transactions for 50 states
plt.figure(figsize=(17,5))
sns.countplot(df['State'])
plt.xticks(rotation=90)
plt.title('Transactions For 50 States')
```

Out[35]:
```
Text(0.5, 1.0, 'Transactions For 50 States')
```



- You can see a pattern of gross sales with thw number of transactions.
- We should be concern with the states with low transactions and sales.

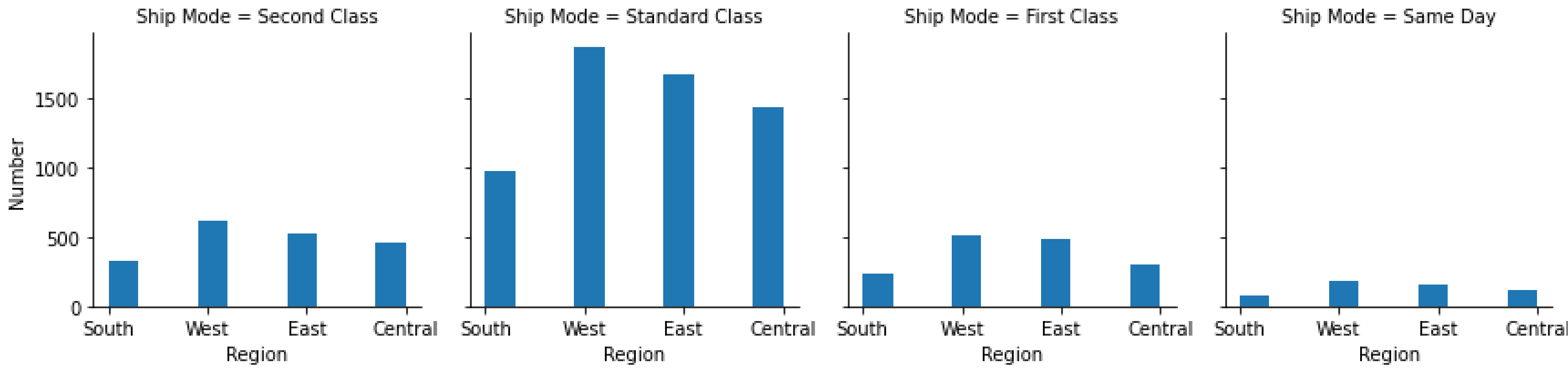
Ship mode in each region

In [36]:

```
#We will visualize the region column from ship mode
region_hist = sns.FacetGrid(df, col='Ship Mode', palette='rainbow')
region_hist.map(plt.hist, 'Region')
region_hist.set_ylabels('Number')
```

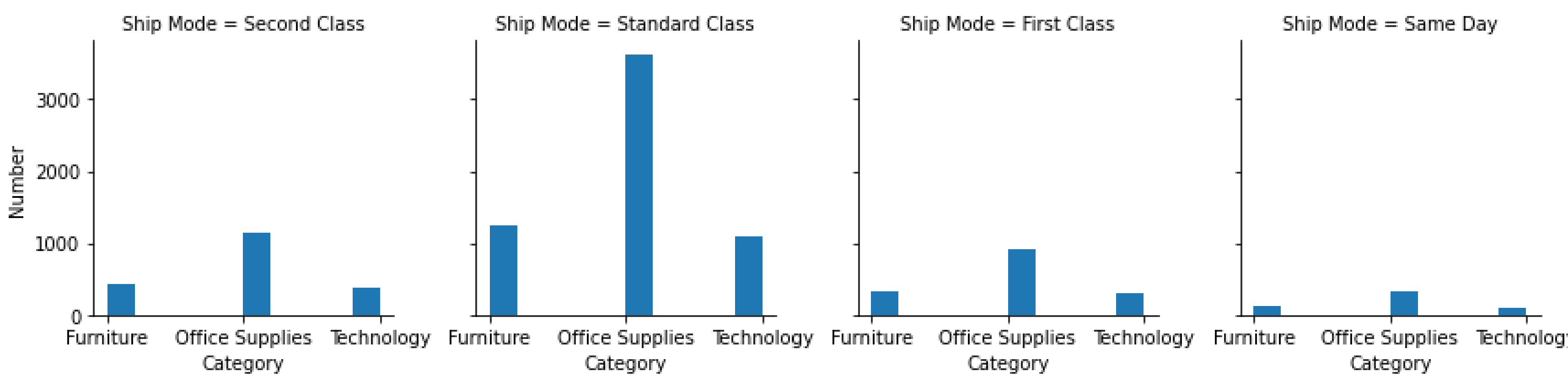
Out[36]:

```
<seaborn.axisgrid.FacetGrid at 0x2319b0445c8>
```



In [37]:
```
#We will visualize the category column for the ship mode
category_hist = sns.FacetGrid(df, col='Ship Mode', palette='rainbow')
category_hist.map(plt.hist, 'Category')
category_hist.set_ylabels('Number')
```

Out[37]: <seaborn.axisgrid.FacetGrid at 0x2319ad1ecc8>



In [38]:
```
#Ship mode sale, profit and discounts
df_ship = df.groupby(['Ship Mode'])['Sales', 'Profit', 'Discount'].mean()
df_ship
```

Out[38]:

	Sales	Profit	Discount
Ship Mode			
First Class	228.497024	31.839948	0.164610

	Sales	Profit	Discount
Ship Mode			
Same Day	236.396179	29.266591	0.152394
Second Class	236.089239	29.535545	0.138895
Standard Class	227.583067	27.494770	0.160023

- Sales is highest in same day.
- Profit is highest in first class.
- Discount is highest in first class.
- Standard shipping has the most orders for prefered shipping.
- Same day and first class has the lowest standard for shipping.

Sales prediction(ML section)

In [39]:

```
#Data has a good number of categorical columns here
#First thing we need to do is remove columns that we are not going to use.
#Make a model that makes predictions regardless of which customer is buying it.
#Take any purchase of a product,see where its coming from/make a prediction based on that regardless of which customer is
#Then we convert our date columns into seperate numerical columns in "encode_dates" function.
#Now we are left with all categorical columns that are not encoded, no numerical columns left.
#In order to move onto the next step we first have to figure out which type of categorical columns there are
#Okay,looking at the columns, we dont have any binary columns
#Checking for any ordering in the categorical columns we find that there isnt any order
#Lets take a look at the 'Ship Mode' column now
#we can include prefix so we know where each of these column is originaly coming from
#pd.get_dummies(X['Ship Mode'], prefix='Ship Mode')
#Now define the function 'onehot_encode'
#After we onehot_encode all of our categorical columns we should be almost ready to go as all our columns will now be num
#last thing to do is scale the data
#last thing to do is scale the data
#mean and std are all over the place which can confuse some models especially neural network

#logistic regression
#linear regression
```

Preprocessing

In [40]:

```
#Define a function to encode the dates for a given df and a column
def encode_dates(df,column):
    df= df.copy()
    df[column]=pd.to_datetime(df[column])

    #now we will create seperate columns for Year,Month and day to get numerical columns
    df[column + '_Year']= df[column].apply(lambda x:x.year)
    df[column + '_Month']= df[column].apply(lambda x:x.month)
    df[column + '_Day']= df[column].apply(lambda x:x.day)
    #now since we will have 3 new columns, we will drop the original column
    df= df.drop(column,axis=1)

    return df


def onehot_encode(df,column):
    df= df.copy()
    #create the dummies
    dummies= pd.get_dummies(df[column], prefix=column)
    #now concat the original df and dummies side by side
    df= pd.concat([df,dummies],axis=1)
    #and then just drop the original column from which we created the dummies
    df= df.drop(column,axis=1)
    return df
```

In [41]:

```
#Start by creating a copy of the dataframe then return the df

def preprocess_inputs(df):
    df= df.copy()

    #Drop unnecessary columns
    df= df.drop(['Row ID','Product Name','Country','Customer Name','Quantity','Discount','Profit'],axis=1)

    #Drop customer specific feature columns
    df= df.drop(['Order ID','Customer ID'],axis=1)

    #Extract date features
    df= encode_dates(df, column= 'Order_date')
    df= encode_dates(df, column= 'Ship_date')

    #One_hot encode categorical features
```

```
    for column in ['Ship Mode','Segment','City','State','Postal Code','Region','Product ID','Category','Sub-Category']:
        df= onehot_encode(df,column=column)

    #Split df into X and y
    y= df['Sales']# what we are trying to predict
    #and everything else in X
    X= df.drop('Sales',axis=1)

    #Now we do our Train-test split
    X_train,X_test,y_train,y_test= train_test_split(X,y,train_size=0.7,shuffle= True,random_state=1)


    #Scale X
    scaler= StandardScaler()
    scaler.fit(X_train)
    #transform both X_train and test after fitting it only to train set
    X_train= pd.DataFrame(scaler.transform(X_train),columns=X.columns)
    X_test=  pd.DataFrame(scaler.transform(X_test),columns= X.columns)


    return X_train,X_test,y_train,y_test
```

In [42]:

```
X_train,X_test,y_train,y_test= preprocess_inputs(df)
```

In [43]:

```
#Get a better sense of this if we create a dict that maps a column name
#To the lentgh of unique values in the column and then mapping the column name to that.
#For every column in X.columns
#{column: len(X[column].unique()) for column in X.columns}
```

In [44]:

```
#thi is just 70% of the data and does not contain the sales column
# sales is contained in y_train and y_test
X_train
```

Out[44]:

	item_price	Order_date_Year	Order_date_Month	Order_date_Day	Ship_date_Year	Ship_date_Month	Ship_date_Day	Ship Mode_First Class	Sh Mode_San Da
0	-0.383934	1.142593	0.354245	-0.628414	1.126475	0.370886	-0.102628	-0.431039	-0.2403
1	0.296437	0.251872	1.271437	0.740589	0.237119	1.271077	1.368680	-0.431039	-0.2403

	item_price	Order_date_Year	Order_date_Month	Order_date_Day	Ship_date_Year	Ship_date_Month	Ship_date_Day	Ship Mode_First Class	Sh Mode_San Da
2	-0.059802	-1.529569	1.271437	0.398338	-1.541592	1.271077	0.576437	2.319978	-0.2403
3	-0.403865	1.142593	-2.091600	0.398338	1.126475	-2.029622	0.802792	-0.431039	-0.2403
4	-0.358145	1.142593	0.048515	-1.312915	1.126475	0.070823	-1.008049	2.319978	-0.2403
...	...	...	...	...	...	...	...	...	
6990	-0.336614	1.142593	0.659976	0.854673	1.126475	0.670950	1.481858	-0.431039	-0.2403
6991	-0.405763	0.251872	-1.785869	-0.172079	0.237119	-1.729558	-0.102628	2.319978	-0.2403
6992	-0.373765	0.251872	-1.174408	-0.856581	0.237119	-1.129431	-0.442161	-0.431039	-0.2403
6993	0.121395	-1.529569	0.965707	1.425091	-1.541592	1.271077	-1.687115	-0.431039	-0.2403
6994	0.978787	1.142593	-1.174408	-0.970664	1.126475	-1.129431	-0.442161	-0.431039	-0.2403

6995 rows × 3111 columns

In [45]:

```
#Lets take a look at the 'Ship Mode' column now
#Since we dont know what class is greater or lesser for ex we dont know if if standard class is greater or lesss than Sec
#We will treat it like a nominal feature like all the others
#X['Ship Mode'].unique()
```

In [46]:

```
#last thing to do is scale the data
#Mean and std are all over the place which can confuse some models especially neural network
X_train.describe()
```

Out[46]:

	item_price	Order_date_Year	Order_date_Month	Order_date_Day	Ship_date_Year	Ship_date_Month	Ship_date_Day	Ship Mode_First Class	Mc
count	6.995000e+03	6.995000e+03	6.995000e+03	6.995000e+03	6.995000e+03	6.995000e+03	6.995000e+03	6.995000e+03	6.99
mean	1.574469e-17	6.208386e-14	-4.850381e-17	-9.446816e-17	-9.623664e-14	1.350996e-16	-1.005629e-16	-5.078933e-17	2.94

	item_price	Order_date_Year	Order_date_Month	Order_date_Day	Ship_date_Year	Ship_date_Month	Ship_date_Day	Ship Mode_First Class	Mc
std	1.000071e+00	1.000071e+00	1.000071e+00	1.000071e+00	1.000071e+00	1.000071e+00	1.000071e+00	1.000071e+00	1.00
min	-4.154169e-01	-1.529569e+00	-2.091600e+00	-1.655166e+00	-1.541592e+00	-2.029622e+00	-1.687115e+00	-4.310385e-01	-2
25%	-3.798392e-01	-6.388484e-01	-8.686772e-01	-8.565809e-01	-6.522366e-01	-8.293678e-01	-8.948715e-01	-4.310385e-01	-2
50%	-3.042093e-01	2.518721e-01	3.542455e-01	5.608769e-02	2.371191e-01	3.708862e-01	1.054922e-02	-4.310385e-01	-2
75%	1.612618e-02	1.142593e+00	9.657068e-01	8.546727e-01	1.126475e+00	9.710133e-01	9.159700e-01	-4.310385e-01	-2
max	2.516115e+01	1.142593e+00	1.271437e+00	1.767341e+00	2.015831e+00	1.271077e+00	1.708213e+00	2.319978e+00	4.16

8 rows × 3111 columns

In [47]:

```
#Sales is contained in y_train and y_test
#Test set contains the other 30% of the data
y_train
```

Out[47]:

```
1963     24.900
9348    842.720
8795    211.168
9389      2.040
5090      8.784
          ...
2895     35.880
7813     10.560
905      12.960
5192    397.600
235     617.976
Name: Sales, Length: 6995, dtype: float64
```

i

Training

In [48]:

```
#Create a 2 hidden layer neural network
#We will play around with the number of neurons in each layer but first,
#Shape : size of the feature vector going into the input
inputs= tf.keras.Input(shape=(X_train.shape[1],))
#Crate our two dense layers
x= tf.keras.layers.Dense(1024,activation='relu')(inputs)
x= tf.keras.layers.Dense(1024,activation='relu')(x)
#For our outputs, also our dense layer,passing 1 value(predicted sale/price)
outputs= tf.keras.layers.Dense(1,activation='linear')(x)

#Create our model
model= tf.keras.Model(inputs=inputs,outputs=outputs)
print(model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 3111)]            0

 dense (Dense)               (None, 1024)              3186688

 dense_1 (Dense)             (None, 1024)              1049600

 dense_2 (Dense)             (None, 1)                 1025

=================================================================
Total params: 4,237,313
Trainable params: 4,237,313
Non-trainable params: 0
_________________________________________________________________
None
```

In [49]:

```
#Compile the model
model.compile(
    optimizer='adam',
    loss='mse'

)

#Fit the model and store it in history
history= model.fit(
    X_train,
    y_train,
```

```
    validation_split= 0.2,
    batch_size= 32,
    epochs=100,#large because we use the callback function
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience= 5,
            restore_best_weights=True
        ),
        tf.keras.callbacks.ReduceLROnPlateau()#helps the model converge easily,when it notices val_loss not improving it
    ]
)
```

```
Epoch 1/100
175/175 [==============================] - 9s 43ms/step - loss: 389376.8125 - val_loss: 168389.1875 - lr: 0.0010
Epoch 2/100
175/175 [==============================] - 6s 35ms/step - loss: 269358.1562 - val_loss: 367710.2500 - lr: 0.0010
Epoch 3/100
175/175 [==============================] - 6s 35ms/step - loss: 166918.4219 - val_loss: 1329754.1250 - lr: 0.0010
Epoch 4/100
175/175 [==============================] - 6s 33ms/step - loss: 104140.7422 - val_loss: 4805650.5000 - lr: 0.0010
Epoch 5/100
175/175 [==============================] - 6s 34ms/step - loss: 67910.8984 - val_loss: 11165938.0000 - lr: 0.0010
Epoch 6/100
175/175 [==============================] - 6s 34ms/step - loss: 52466.6875 - val_loss: 10409054.0000 - lr: 0.0010
```

Results

In [50]:

```
#RESULTS
test_loss= model.evaluate(X_test,y_test,verbose=0)

print('test loss: {:.5f}'.format(test_loss))
```

```
test loss: 265525.50000
```

In [51]:

```
y_pred= np.squeeze(model.predict(X_test))

test_r2= r2_score(y_test,y_pred)

print('Test R2 score: {:.5f}'.format(test_r2))
```

```
Test R2 score: 0.28307
```

- We used mean squared error as our loss function. The test loss value of 277178.46875 is relatively a very high value,in an ideal model the lower the MSE the higher the accuracy of prediction as there would be excellent match between the actual and predicted data set.
- The R^2 value of 0.25161 is not a good result as ideally it should be as close to 1 as possible or atleast higher than 0.9 to rely on our model.
- These results indicate that we need to try and build a different model most likely using a different method to accurately predict sales as we already tried changing the parameters to get better results.

Conclusion

- Western region is where the vast majority of the supplies are purchased.
- Southern region has the the lowest number of orders are sold.
- Superstore should focus primary on discounts and more online sales for consumers in the south.
- Office supplies have the mores orders in category.
- Binders have the most orders in the Subcategory from office supplies.
- Technology has the least about of orders sold.
- The Superstore could improve inventory and offer discounts and promotions to increase sales in all regions for technology and furnitures orders.
- Profit is high in First class shipping.
- Standard shipping is the most common shipping method.

Work Cited

“Superstore Dataset.” Www.kaggle.com, www.kaggle.com/datasets/vivek468/superstore-dataset-final.

- Dataset/ Author Vivek Chowdhury

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