Data Visualization project

Introduction

The dataset used for the project is the Tableau superstore dataset. The dataset contains information about the orders, Returns and the employees. The data was in 3 separte spreadsheets which have been merged together for the project. The dataset consists of 9994 observations and 21 features. The sample Dataset includes data for the sales of multiple products sold by a company along with the information related to geography, product categories, and subcategories, sales, and profits, etc

The dataset is available at https://www.kaggle.com/datasets/vivek468/superstore-dataset-final

Data Description

- Row ID - Unique ID for each row.
- Order ID - Unique Order ID for each Customer.
- Order Date - Order Date of the product.
- Ship Date - Shipping Date of the Product.
- Ship Mode -Shipping Mode specified by the Customer.
- Customer ID - Unique ID to identify each Customer.
- Customer Name - Name of the Customer.
- Segment - The segment where the Customer belongs.
- Country - Country of residence of the Customer.
- City - City of residence of of the Customer.
- State - State of residence of the Customer.
- Postal Code - Postal Code of every Customer.
- Region - Region where the Customer belong.
- Product ID - Unique ID of the Product.
- Category - Category of the product ordered.
- Sub-Category - Sub-Category of the product ordered.
- Product Name - Name of the Product
- Sales - Sales of the Product.
- Quantity - Quantity of the Product.
- Discount - Discount provided.
- Profit - Profit/Loss incurred.
- Sales Person - Employee in charge of a Region
- Returned - Order ids that were returned

The datset contained no null values.

Goals : The goal was to analyse and create a visualisation to gain better insights into the categorical sales treends and patterns of the superstore to improve marketing and sales strategies.

Tasks :

- To identify the trends in the sales data.
- Identify under performing product categories
- Identify products with better profits
- Identify stores/areas of concern

Data Preparation

In [ ]:

```
import pandas as pd
import numpy as np
import altair as alt
import streamlit as st
import openpyxl
import panel as pn
from panel.interact import interact

alt.data_transformers.disable_max_rows() #workaround for using a dataset larger than 5000 rows

df = pd.read_excel("sample_-_superstore.xls",sheet_name = None) # Import the dataset from each excel sheet
keys = list(df.keys())
```

```
df_names =[] # creating an empty list for the dataframe names
# using loop to assign the values of the keys to a pandas dataframe. Adding the names of the datframe into the empty list
for i in keys:
    globals()[f"df_{i}"] = df.get(i)
    df_names.append(f"df_{i}")
print(df_names)
```

```
['df_Orders', 'df_Returns', 'df_People']
```

In [ ]:

```
df_Orders.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 [ ]:

```
df_Returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 296 entries, 0 to 295
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Returned  296 non-null    object
 1   Order ID  296 non-null    object
dtypes: object(2)
memory usage: 4.8+ KB
```

In [ ]:

```
df_People.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Person  4 non-null      object
 1   Region  4 non-null      object
dtypes: object(2)
memory usage: 192.0+ bytes
```

In [ ]:

```
#Merging the Orders and People dataframes via inner join
df = pd.merge(df_Orders,df_People, on = 'Region', how ='inner')
df.head()
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Region	Product ID	Category	Sub-Category	Pro N
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	South	FUR-BO-10001798	Furniture	Bookcases	Som Colle Bool
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	South	FUR-CH-10000454	Furniture	Chairs	Hon D F Uphols Stac Cha
2	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	South	FUR-TA-10000577	Furniture	Tables	Bre CR Series Rectan
3	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	South	OFF-ST-10000760	Office Supplies	Storage	Eldon 'N Roll Sy
4	13	CA-2017-114412	2017-04-15	2017-04-20	Standard Class	AA-10480	Andrew Allen	Consumer	United States	Concord	...	South	OFF-PA-10002365	Office Supplies	Paper	Xerox

5 rows × 22 columns

In [ ]:

```python
#Left join on the df and returns dataframe
df = pd.merge(df,df_Returns, on = 'Order ID', how ='left')

#The left joint creates null values for the returns column for products that were not returned.
#Replacing null with 'No
df['Returned'].fillna('No', inplace =True)
```

In [ ]:

```python
df.to_excel("output.xlsx", index=False)
df.head()
```

Out[ ]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Product ID	Category	Sub-Category	Product Name	
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	26
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	73
2	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	95
3	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	OFF-ST-10000760	Office Supplies	Storage	Eldon Fold 'N Roll Cart System	2
4	13	CA-2017-114412	2017-04-15	2017-04-20	Standard Class	AA-10480	Andrew Allen	Consumer	United States	Concord	...	OFF-PA-10002365	Office Supplies	Paper	Xerox 1967	1

5 rows × 23 columns

Low Fidelity Prototype Visualizations

Based on the tasks elicited, the low fidelity prototypes/sketches for the dashboard were

- A pie chart - to visualise Regional sales
- A geographical/map - to visualise regional sales more granualarly

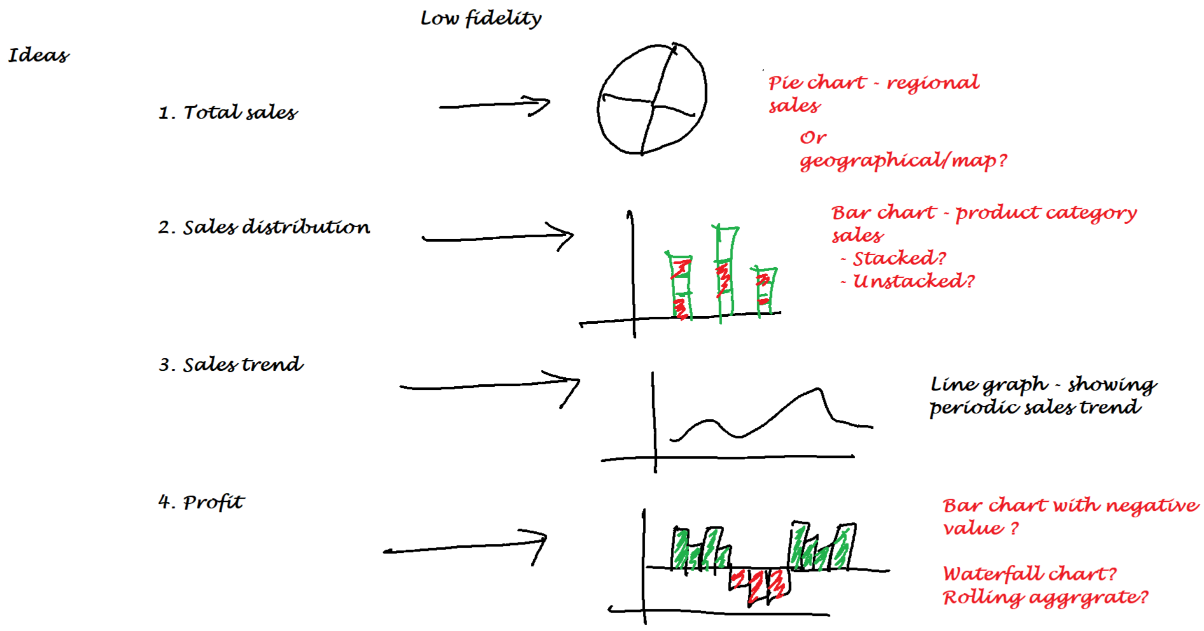
- Bar chart showing the sales of each product category in each reagion - stacked with sub-categories or regional sales with produt category sales stacked
- A line graph showing the sales trend
- Bar chart for profit showing negative values and periodic changes, or a waterfall chart or rolling aggregrate charts

In [ ]:

```
# import image module
from IPython.display import Image

# get the image
Image(url="Lowfidelity.png")
```

Out[ ]:



Visualizations similar to the low fidelity prototypes were generated to create the numerous iterations

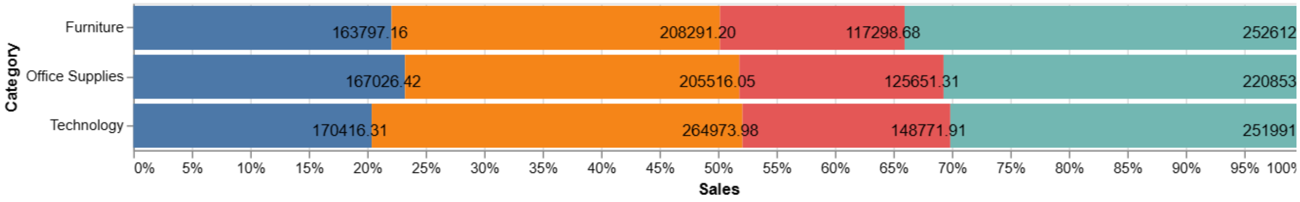
In [ ]:

```
bars = alt.Chart(filtered_data).mark_bar().encode(
    x=alt.X('sum(Sales):Q', stack='normalize', title='Sales'),
    y=alt.Y('Category:N', title='Category'),
    color=alt.Color('Region:N')
)

text = alt.Chart(filtered_data).mark_text(dx=-15, dy=4, color='black').encode(
    x=alt.X('sum(Sales):Q', stack='normalize', title='Sales'),
    y=alt.Y('Category:N', title='Category'),
    detail = 'Region:N',
    text=alt.Text('sum(Sales):Q', format='.2f')
)

# Combine the bars and text
stacked_bar_chart = (bars + text).properties(
    width=800,
    height=100
)
stacked_bar_chart
```

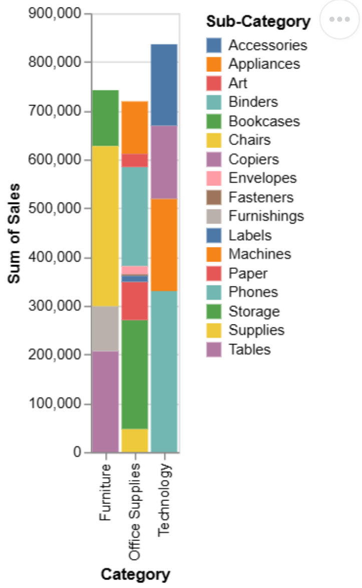
Out[ ]:



In [ ]:

```
alt.Chart(df).mark_bar().encode(x="Category",y="sum(Sales):Q", color ='Sub-Category')
```

Out[ ]:

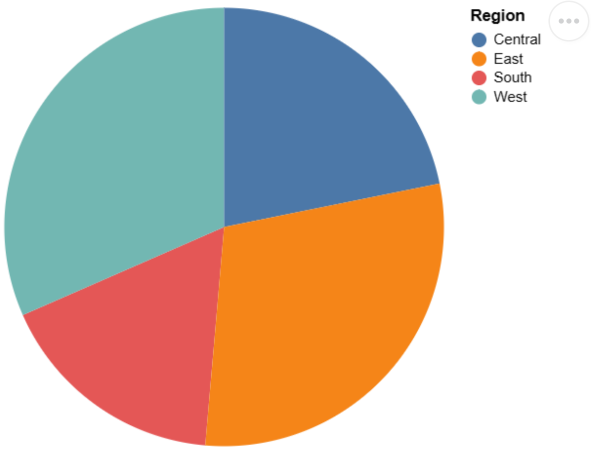


In [ ]:

```
pie_chart = alt.Chart(df).mark_arc().encode(
    theta="sum(Sales):Q",
    color="Region:N"
)

pie_chart
```

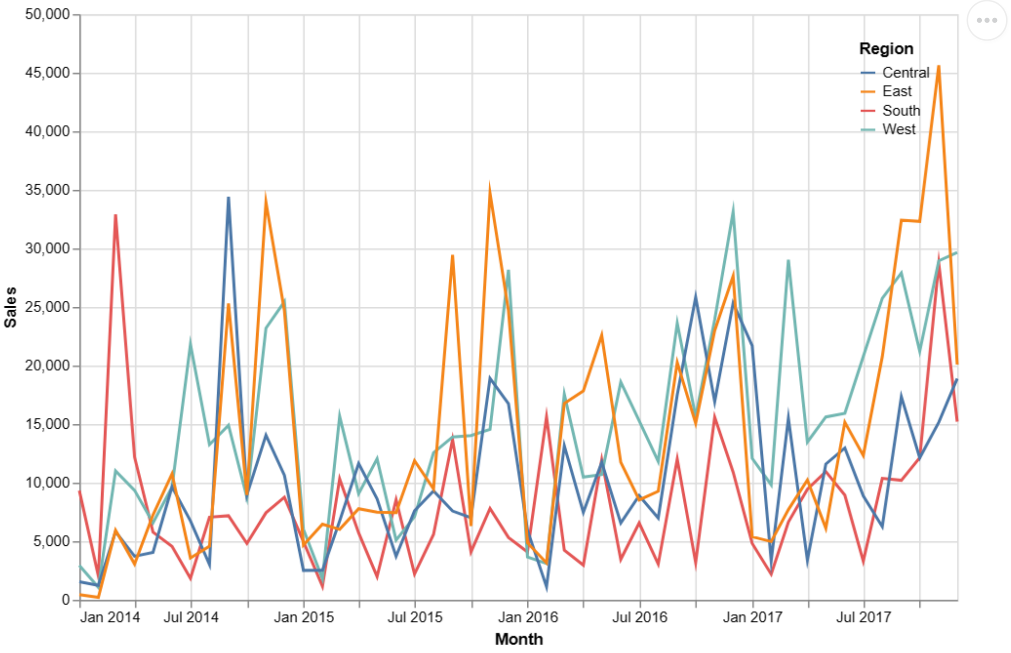
Out[ ]:



In [ ]:

```
profit_over_time_chart = alt.Chart(df).mark_line().encode(
    x=alt.X('yearmonth(Order Date):T', title='Month'),
    y=alt.Y('sum(Sales):Q', title=f'Sales'),
    color=alt.Color('Region:N', legend=alt.Legend(orient='top-right'))
).properties(
    width=600,
    height=400)
profit_over_time_chart
```

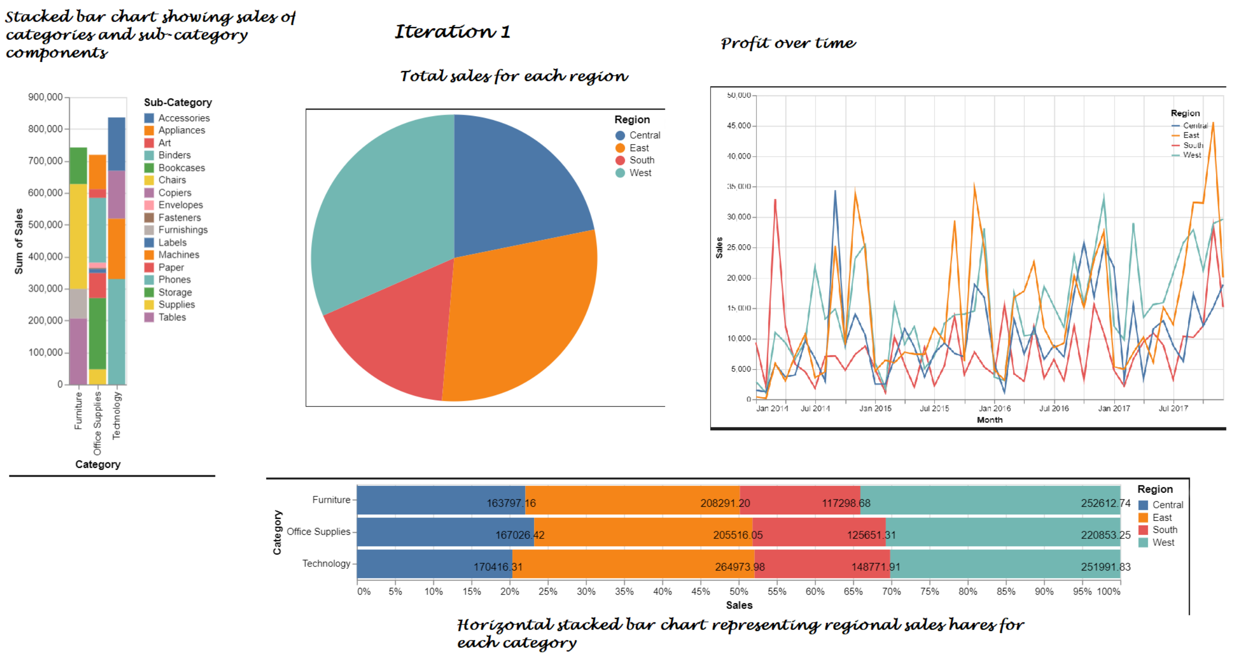
Out[ ]:



In [ ]:

```
# get the image
Image(url="iteration1.png")
```

Out[ ]:


Stacked bar chart showing sales of categories and sub-category components

Iteration 1

Total sales for each region

Profit over time

Horizontal stacked bar chart representing regional salesshares for each category

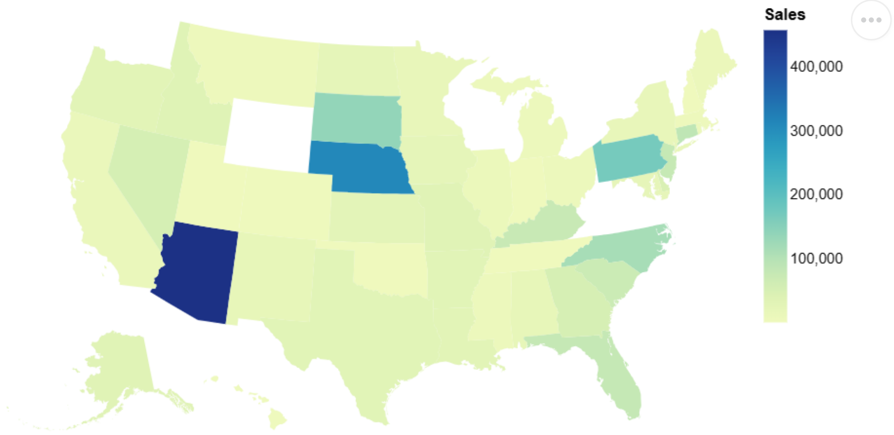
Following the first iteration, it was evident that we needed interaction and filters to further explore the data, The pie chart was ineffective and wasnt conveying any insights other than the total percentage of sales of each region.The profit over time for each region would require smoothening to elicit any trend patterns. The stacked categories chart gave only a rough idea about the sales for each category & subcategory.

Few more charts were developed along with ideas to incorporate interactivity

In [ ]:

```
from vega_datasets import data
states = alt.topo_feature(data.us_10m.url, 'states')
state_sales_data = df.groupby('State')['Sales'].sum().reset_index()
state_sales_data['id'] =state_sales_data.index +1
source = state_sales_data
map =alt.Chart(states).mark_geoshape().encode(
    color='Sales:Q',
    tooltip=['State:N', 'Sales:Q']
).transform_lookup(
    lookup='id',
    from_=alt.LookupData(source, 'id', list(source.columns))
).properties(
    width=500,
    height=300
).project(
    type='albersUsa'
)
map
```

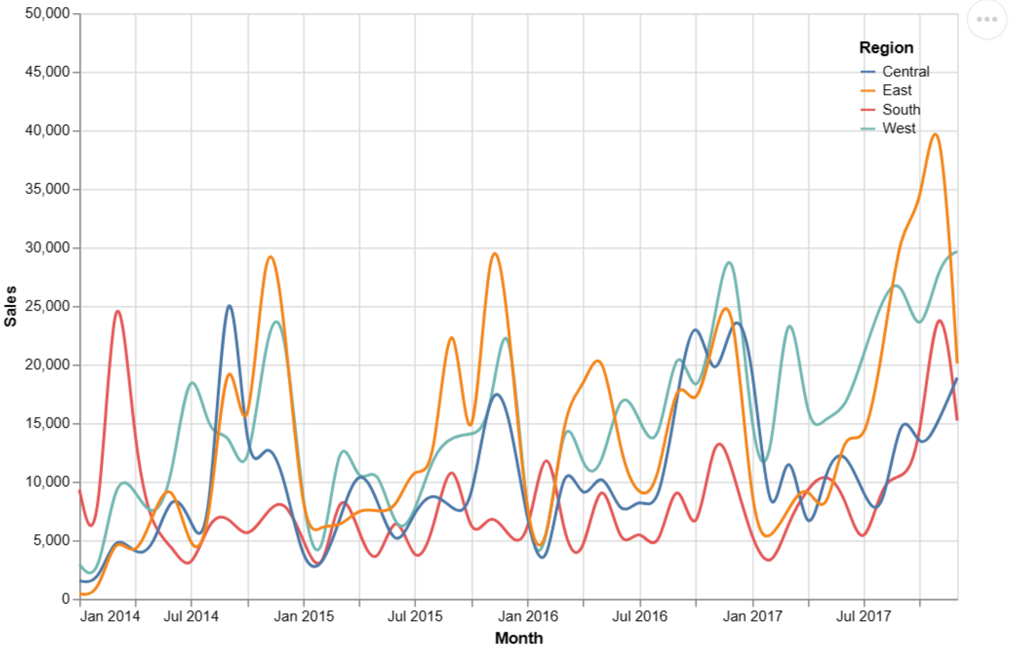
Out[ ]:



In [ ]:

```
profit_over_time_chart = alt.Chart(df).mark_line(interpolate='basis').encode(
    x=alt.X('yearmonth(Order Date):T', title='Month'),
    y=alt.Y('sum(Sales):Q', title=f'Sales'),
    color=alt.Color('Region:N', legend=alt.Legend(orient='top-right'))
).properties(
    width=600,
    height=400)
profit_over_time_chart
```

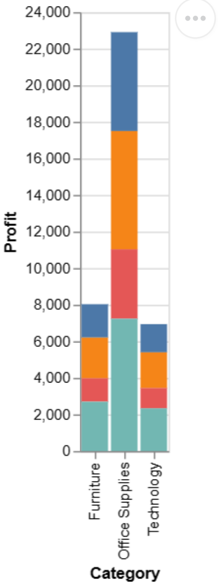
Out[ ]:



In [ ]:

```
bar = alt.Chart(df).mark_bar().encode(
    y=alt.X('sum(Quantity):Q', title ='Profit'),
    x=alt.Y('Category:N', title='Category'),
    color=alt.Color('Region:N', legend=None))
bar
```

Out[ ]:

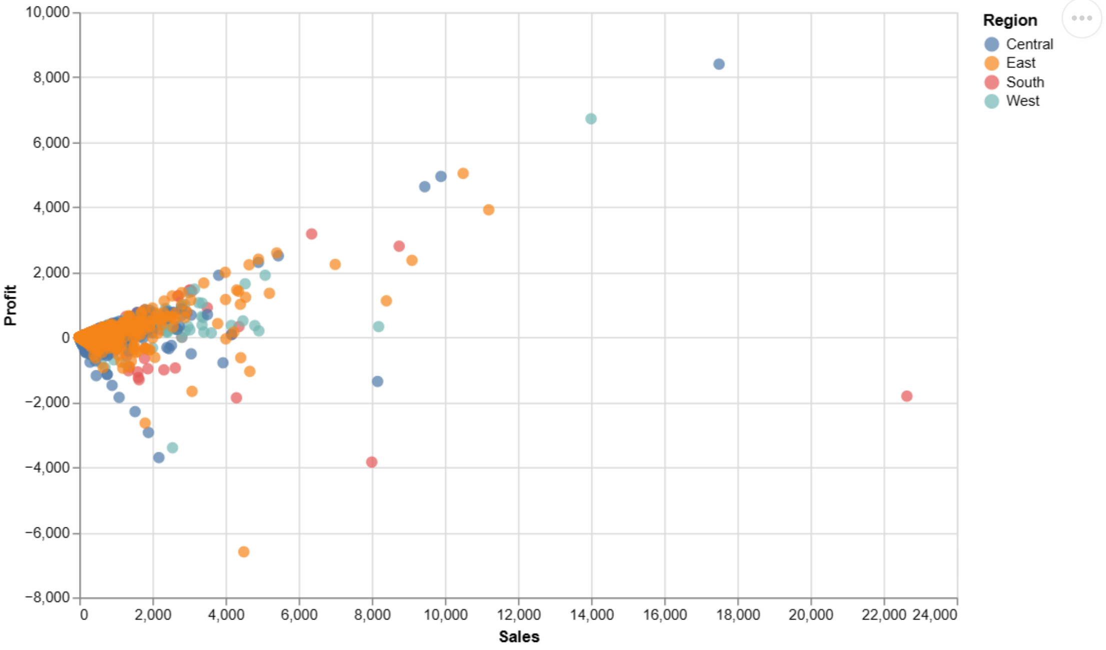


In [ ]:

```
scatter_chart = alt.Chart(df).mark_circle(size=60).encode(
    x=alt.X('Sales', title='Sales'),
    y=alt.Y('Profit', title='Profit'),
    color=alt.Color('Region:N', legend=alt.Legend(title='Region')),
    tooltip=['Sales', 'Profit', 'Region']
).properties(
    width=600,
    height=400)

scatter_chart
```

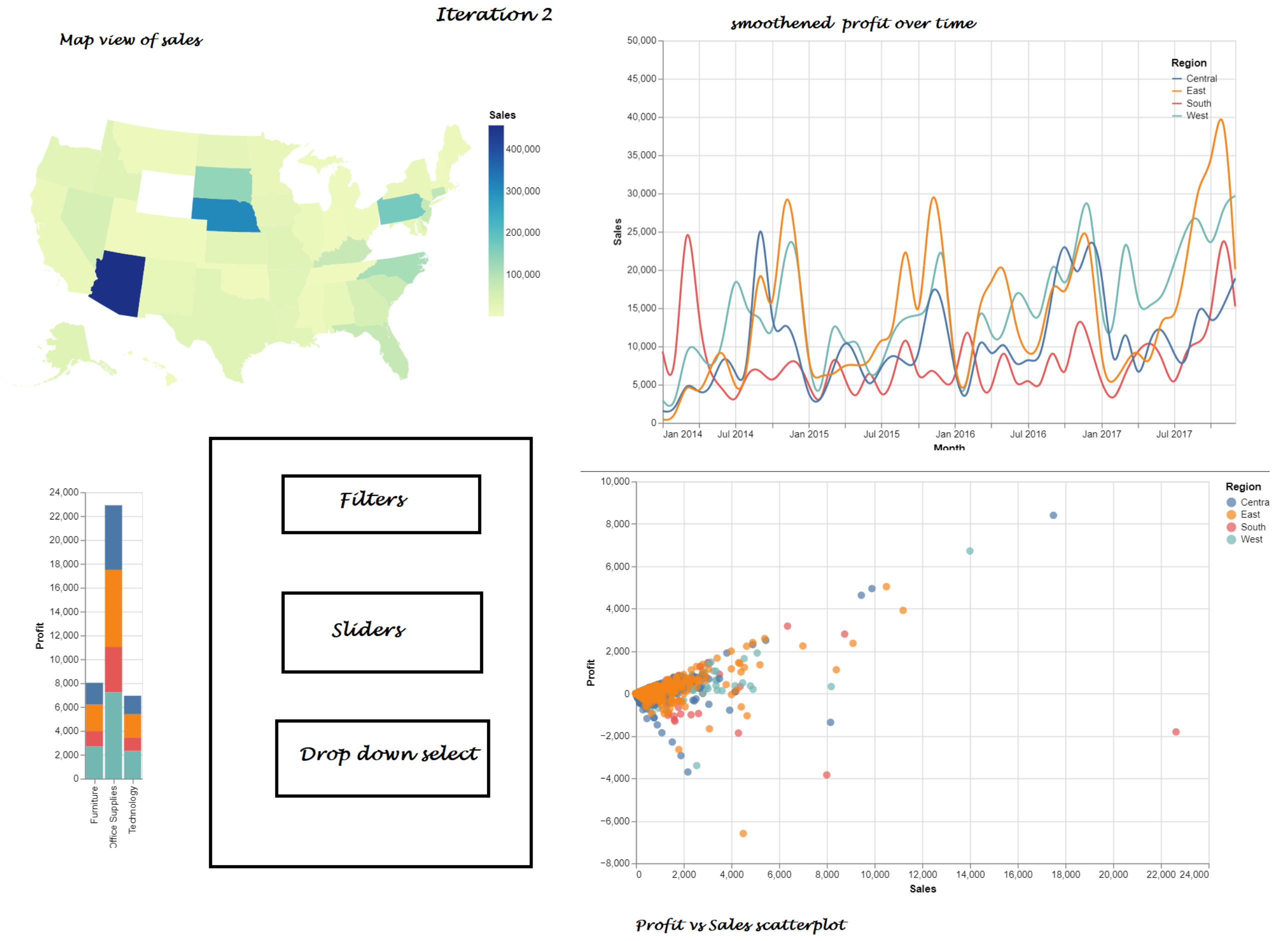
Out[ ]:



In [ ]:

```
# get the image
Image(url="iteration2.png")
```

Out[ ]:

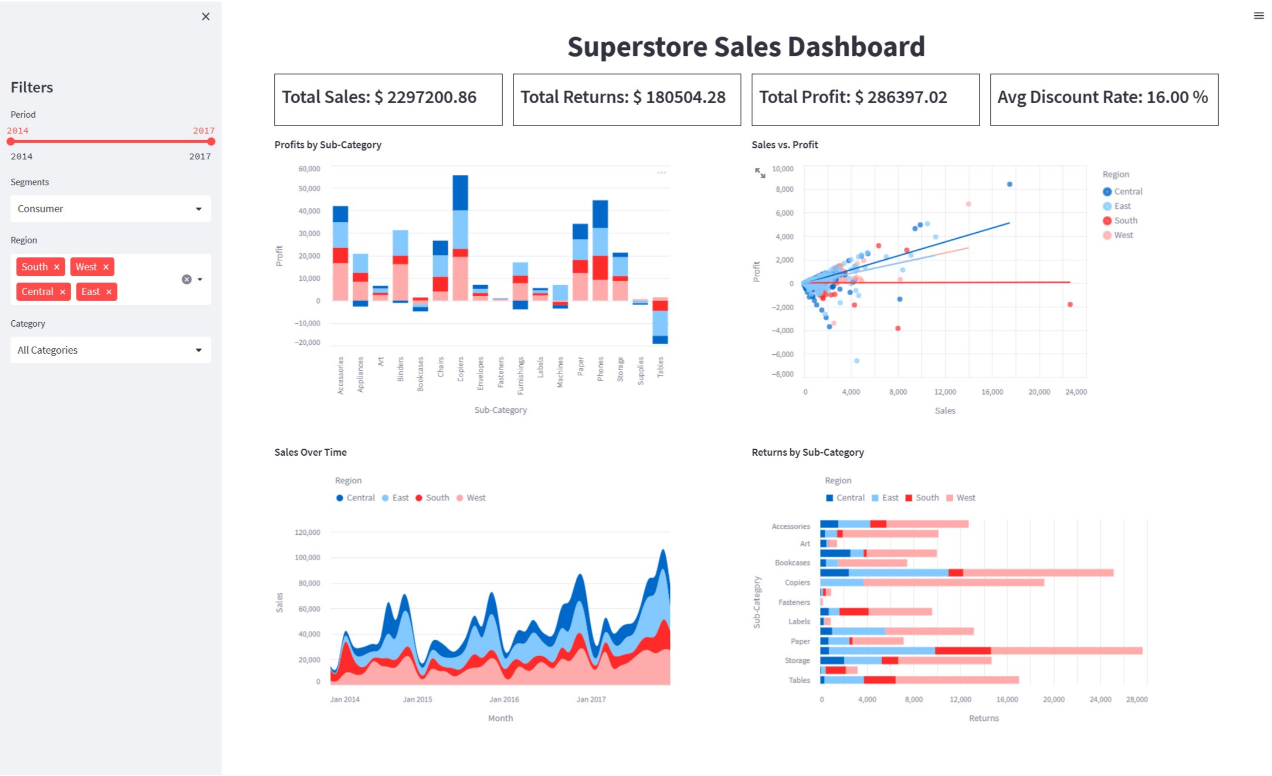


Final Dashboard & Code for the Streamlit app implementation

In [ ]:

```
# Final Dashboard
Image(url="Dashboard.jpg")
```

Out[ ]:



Filters

Period

2014 2017

Segments

Consumer

Region

South × West × Central × East ×

Category

All Categories

Superstore Sales Dashboard

Total Sales: $ 2297200.86

Total Returns: $ 180504.28

Total Profit: $ 286397.02

Avg Discount Rate: 16.00 %

In [ ]:

```
import pandas as pd
import numpy as np
import altair as alt
import streamlit as st


alt.data_transformers.disable_max_rows()
# Load the Superstore dataset

df = pd.read_excel("output.xlsx")

region_list = list(df['Region'].unique())
segment_list = list(df['Segment'].unique())
category_list = list(df['Category'].unique())
category_list.insert(0, "All Categories")
df['YYYY'] = df['Order Date'].apply(lambda x: x.year)

min_year = df['YYYY'].min()
max_year = df['YYYY'].max()

# Create the Streamlit app
st.set_page_config(layout="wide")
st.markdown("<h1 style='text-align: center;'>Superstore Sales Dashboard</h1>", unsafe_allow_html=True)
#st.title("Superstore Sales Dashboard")
#st.markdown("---")

# Sidebar filters
st.sidebar.title("Filters")
start_year, end_year = st.sidebar.slider("Period", min_value=int(min_year), max_value=int(max_year),
    value=(int(min_year),int(max_year)))
segment = st.sidebar.selectbox('Segments', segment_list)
region = st.sidebar.multiselect("Region", df['Region'].unique(),default =region_list)
category = st.sidebar.selectbox("Category", category_list)
if category != "All Categories":
    filtered_data = df[(df['YYYY'] >= start_year) & (df['YYYY'] <= end_year) & (df['Category'] == category) & (df['Region'].
else:
    filtered_data = df[(df['YYYY'] >= start_year) & (df['YYYY'] <= end_year) & (df['Region'].isin(region))]

# Calculate total sales and profit
total_sales = round(filtered_data['Sales'].sum(),2)
total_profit = round(filtered_data['Profit'].sum(),2)
total_returns = round(filtered_data.loc[filtered_data['Returned'] == 'Yes', 'Sales'].sum(), 2)
Average_discount = round(filtered_data['Discount'].mean(),2)

# Display total sales and profit
col1, col2,col3,col4 = st.columns(4)
with col1:
    st.markdown(
        f"""
        <div style='border: 1px solid black; padding: 10px;'>
            <h3>Total Sales: $ {total_sales}</h3>
        </div>
        """,
        unsafe_allow_html=True
    )

with col2:
    st.markdown(
        f"""
        <div style='border: 1px solid black; padding: 10px;'>
            <h3>Total Returns: $ {total_returns}</h3>
        </div>
        """,
        unsafe_allow_html=True
    )
with col3:
    st.markdown(
        f"""
        <div style='border: 1px solid black; padding: 10px;'>
            <h3>Total Profit: $ {total_profit}</h3>
        </div>
        """,
        unsafe_allow_html=True
    )
with col4:
    st.markdown(
        f"""
        <div style='border: 1px solid black; padding: 10px;'>
            <h3>Avg Discount Rate: {Average_discount*100:.2f} %</h3>
        </div>
        """,
        unsafe_allow_html=True
    )

# Bar chart showing sales by sub-category
col1, col2 = st.columns(2)

# Bar chart showing sales by sub-category
subcat_sales_chart = alt.Chart(filtered_data).mark_bar().encode(
    x=alt.X('Sub-Category:N', title= 'Sub-Category'),
    y=alt.Y('sum(Profit):Q', title = 'Profit'),
    color=alt.Color('Region:N', legend=None),
    tooltip=['Sub-Category:N', 'sum(Sales):Q', 'sum(Profit):Q']
).properties(
    width=600,
    height=400
)
# Display the charts
col1, col2 = st.columns(2)
scatter_chart= alt.Chart(filtered_data).mark_circle(size=60).encode(
    x=alt.X('Sales', title='Sales'),
    y=alt.Y('Profit', title='Profit'),
    color=alt.Color('Region:N', legend=alt.Legend(title='Region')),
    tooltip=['Sales', 'Profit', 'Region']
).properties(
    width=600,
    height=400
)
trendlines1 = scatter_chart.transform_regression(
    on='Sales',
    regression='Profit',
    groupby=['Region']
).mark_line()


# Line chart showing sales over time
profit_over_time_chart = alt.Chart(filtered_data).mark_area(interpolate='basis').encode(
    x=alt.X('yearmonth(Order Date):T', title='Month'),
    y=alt.Y('sum(Sales):Q', title=f'Sales'),
    color=alt.Color('Region:N', legend=alt.Legend(orient='top'))
).properties(
    width=600,
    height=400
)

# Bar chart showing returns by sub-category
category_returns_chart = alt.Chart(filtered_data[filtered_data['Returned'] == 'Yes']).mark_bar().encode(
    y=alt.X('Sub-Category:N', title='Sub-Category'),
    x=alt.Y('sum(Sales):Q', title='Returns'),
    color=alt.Color('Region:N', legend=alt.Legend(orient='top'))
```

```
).properties(
    width=600,
    height=400
)

col1.write(f" **Profits by Sub-Category**")
col1.altair_chart(subcat_sales_chart)
col2.write(f" **Sales vs. Profit**")
col2.altair_chart(scatter_chart+trendlines1)

col1.write(f" **Sales Over Time**")
col1.altair_chart(profit_over_time_chart)

col2.write(f" **Returns by Sub-Category**")
col2.altair_chart(category_returns_chart)
```

```
Out[ ]: DeltaGenerator()
```

After numerous iterations the visualizations evolved from low fidelity (initial design sketches) to final dashboard visualisation using the Altair library in Streamlit with filters provided for the sales periods measured in years, multi-select Region selection option , single-select option for segments and single-select option for categories which included 'all categories in addition to the unique product categories. The charts included were profits by sub-category with stacks indicating the region. This chart would help the employees identify the product categories & sub-categories That were underperforming in particukar regions. The smoothened stacked area chart for sales over time was a much better indicator of the general trend of sales over time in comparison to the earlier prototypes.It helps to indentify the seasonality trends for the overall sales and also for each individual region. The Sales vs Profit scatterplot helps identify the trend of profitability of each region. The returns by sub-category helps identify the regions,product & sub-categories receiving higher return. The period, segment and category filters helps in examine the data in much more granularity and in conjunction with the 4 charts would help uncover the factors resulting in underperforming regions/categories/subcategories. For example, central region has 500k sales, the profit is just around 39,000 which could be attributed to the higher than average discount rate of 24% incomparison to the mean discount rate of 16% for all 4 regions.

My approach at evaluation was to provide the dataset information & visualisation link to my colleagues and friends, to analyse the interpretability and readbility of the visualizations provided. The evaluation procedure involved providing the evaluators with access to the Streamlit app and a few specific tasks to perform, such as analyzing sales performance in different regions or identifying trends in specific categories. It was helpful in pinpointing the visualizations that provided a biased view of the data or the ones that provided too information that it was tough to fnd meaningful insights from the data. The evaluators provided feedback on the usability, clarity, and effectiveness of the visualizations, as well as any suggestions for improvement. One of such suggestions was the incorporation of aggregrate values which has been implemented in the dashboard.

The evaluators were easily able to identify the underperforming products and regions and were able to come up with multiple reasons for the underperformances. Based on these findings, I believe the incorporation of the aggregrate values, the returns information, profit and sales information of the products in the dashboard make sitan effective tool to find the insights into the data.

For future iterations I would like to incorporate the sales information in a map view, (using the map as a filter) with interactions with the other charts(profit, return, product info) on the dashboard, providing even more granular information like products underperforming only in certain states or postal code. It would help to develop strategies like understocking or product withdrawal from these states instead of the entire region.