Importing

In [1]:
```
import numpy as np
import pandas as pd

import calendar

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

import os
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

Reading CSV files for all three dataset

In [2]:
```
order = pd.read_csv("List_of_order.csv")
order.head()
```

Out[2]:

	Order ID	Order Date	CustomerName	State	City
0	B-25601	1/4/2022	Bharat	Gujarat	Ahmedabad
1	B-25602	1/4/2022	Pearl	Maharashtra	Pune
2	B-25603	3/4/2022	Jahan	Madhya Pradesh	Bhopal
3	B-25604	3/4/2022	Divsha	Rajasthan	Jaipur
4	B-25605	5/4/2022	Kasheen	West Bengal	Kolkata

In [3]:
```
details = pd.read_csv("Order_Details.csv")
details.head()
```

Out[3]:

	Order ID	Amount	Profit	Quantity	Category	Sub-Category
0	B-25601	1275	-1148	7	Furniture	Bookcases
1	B-25601	66	-12	5	Clothing	Stole
2	B-25601	8	-2	3	Clothing	Hankerchief
3	B-25601	80	-56	4	Electronics	Electronic Games
4	B-25602	168	-111	2	Electronics	Phones

In [4]:
```
target = pd.read_csv("Sales_Target.csv")
target.head()
```

Out[4]:

	Month of Order Date	Category	Target
0	18-Apr	Furniture	10400
1	18-May	Furniture	10500
2	18-Jun	Furniture	10600
3	18-Jul	Furniture	10800
4	18-Aug	Furniture	10900

Checking if there is any null values present in the dataset

In [5]:
```
order.isnull().sum()
```

Out[5]:
```
Order ID        0
Order Date      0
CustomerName    0
State           0
City            0
dtype: int64
```

In [6]:
```
details.isnull().sum()
```

Out[6]:
```
Order ID        0
Amount          0
Profit          0
Quantity        0
Category        0
Sub-Category    0
dtype: int64
```

In [7]:
```
target.isnull().sum()
```

Out[7]:
```
Month of Order Date    0
Category               0
Target                 0
dtype: int64
```

In [8]:
```
target.shape
```

Out[8]:
```
(36, 3)
```

In [9]:
```
details.shape
```

Out[9]:
```
(1500, 6)
```

In [10]:
```
order.shape
```

Out[10]:
```
(500, 5)
```

Changing the columns type to appropriate data type

In [11]:
```
order.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Order ID      500 non-null    object
 1   Order Date    500 non-null    object
 2   CustomerName  500 non-null    object
 3   State         500 non-null    object
 4   City          500 non-null    object
dtypes: object(5)
memory usage: 19.7+ KB
```

In [12]:
```
# Changing the Order Date variable to datetime data type
order['Order Date'] = order['Order Date'].astype('datetime64[ns]')
```

```
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '13-04-2022' in D
D/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '15-04-2022' in D
D/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '17-04-2022' in D
D/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '18-04-2022' in D
D/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '20-04-2022' in D
D/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '22-04-2022' in D
D/MM/YYYY format. Provide format or specify infer_datetime_format=True for consistent parsing.
  cache_array = _maybe_cache(arg, format, cache, convert_listlike)
C:\Users\hp\anaconda3\lib\site-packages\pandas\core\tools\datetimes.py:1063: UserWarning: Parsing '23-04-2022' in D
```

In [13]:
```
order.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   Order ID      500 non-null    object        
 1   Order Date    500 non-null    datetime64[ns]
 2   CustomerName  500 non-null    object        
 3   State         500 non-null    object        
 4   City          500 non-null    object        
dtypes: datetime64[ns](1), object(4)
memory usage: 19.7+ KB
```

In [14]:
```
# Cleaning the detail dataset
details.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Order ID      1500 non-null   object
 1   Amount        1500 non-null   int64 
 2   Profit        1500 non-null   int64 
 3   Quantity      1500 non-null   int64 
 4   Category      1500 non-null   object
 5   Sub-Category  1500 non-null   object
dtypes: int64(3), object(3)
memory usage: 70.4+ KB
```

In [15]:
```
# Chaning the Category and Sub-category variable to categorical data type
details['Category'] = details['Category'].astype('category')
details['Sub-Category'] = details['Sub-Category'].astype('category')
details.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype   
---  ------        --------------  -----   
 0   Order ID      1500 non-null   object  
 1   Amount        1500 non-null   int64   
 2   Profit        1500 non-null   int64   
 3   Quantity      1500 non-null   int64   
 4   Category      1500 non-null   category
 5   Sub-Category  1500 non-null   category
dtypes: category(2), int64(3), object(1)
memory usage: 50.7+ KB
```

In [16]:
```
target.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Month of Order Date  36 non-null     object
 1   Category             36 non-null     object
 2   Target               36 non-null     int64 
dtypes: int64(1), object(2)
memory usage: 992.0+ bytes
```

In [17]:
```
target['Category'] = target['Category'].astype('category')
target.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36 entries, 0 to 35
Data columns (total 3 columns):
 #   Column               Non-Null Count  Dtype   
---  ------               --------------  -----   
 0   Month of Order Date  36 non-null     object  
 1   Category             36 non-null     category
 2   Target               36 non-null     int64   
dtypes: category(1), int64(1), object(1)
memory usage: 872.0+ bytes
```

In [18]:
```
# Cleanded Details data
details.head()
```

Out[18]:

	Order ID	Amount	Profit	Quantity	Category	Sub-Category
0	B-25601	1275	-1148	7	Furniture	Bookcases
1	B-25601	66	-12	5	Clothing	Stole
2	B-25601	8	-2	3	Clothing	Hankerchief
3	B-25601	80	-56	4	Electronics	Electronic Games
4	B-25602	168	-111	2	Electronics	Phones

In [19]:
```
# Cleaned Order Data
order.head()
```

Out[19]:

	Order ID	Order Date	CustomerName	State	City
0	B-25601	2022-01-04	Bharat	Gujarat	Ahmedabad
1	B-25602	2022-01-04	Pearl	Maharashtra	Pune
2	B-25603	2022-03-04	Jahan	Madhya Pradesh	Bhopal
3	B-25604	2022-03-04	Divsha	Rajasthan	Jaipur
4	B-25605	2022-05-04	Kasheen	West Bengal	Kolkata

In [20]:
```
# Cleaned Target Dataset
target.head()
```

Out[20]:

	Month of Order Date	Category	Target
0	18-Apr	Furniture	10400
1	18-May	Furniture	10500
2	18-Jun	Furniture	10600
3	18-Jul	Furniture	10800
4	18-Aug	Furniture	10900

Making a new dataframe containing the Amount, Profit and Quantity of the different orders. Then joining it with the Order datasets by taking Order ID as the Primary Key.

In [21]:
```
profits = details.groupby('Order ID').sum().reset_index()
profits.head()
```

Out[21]:

	Order ID	Amount	Profit	Quantity
0	B-25601	1429	-1218	19
1	B-25602	3889	975	22
2	B-25603	2025	-180	25
3	B-25604	222	22	11
4	B-25605	75	0	7

Now I will merge two datasets order details and list of order As this two dataset have common relation between them using Order ID

In [22]:
```
df = pd.merge(order, profits)
df.head()
```

Out[22]:

	Order ID	Order Date	CustomerName	State	City	Amount	Profit	Quantity
0	B-25601	2022-01-04	Bharat	Gujarat	Ahmedabad	1429	-1218	19
1	B-25602	2022-01-04	Pearl	Maharashtra	Pune	3889	975	22
2	B-25603	2022-03-04	Jahan	Madhya Pradesh	Bhopal	2025	-180	25
3	B-25604	2022-03-04	Divsha	Rajasthan	Jaipur	222	22	11
4	B-25605	2022-05-04	Kasheen	West Bengal	Kolkata	75	0	7

Sales Trend Analysis

Trend analysis is to find patterns in data, such as ups & downs. A “trend” is an upwards or downwards shift in a data set over time. In retail, this analysis of past trends in sales or revenue; allows to predict the future market. This analysis useful for budgeting and forecasting. Total sales of any business on a trend line may obtain some significant information

In [23]:
```
profits = details.groupby('Order ID').sum().reset_index()
profits.head()
```

Out[23]:

	Order ID	Amount	Profit	Quantity
0	B-25601	1429	-1218	19
1	B-25602	3889	975	22
2	B-25603	2025	-180	25
3	B-25604	222	22	11
4	B-25605	75	0	7

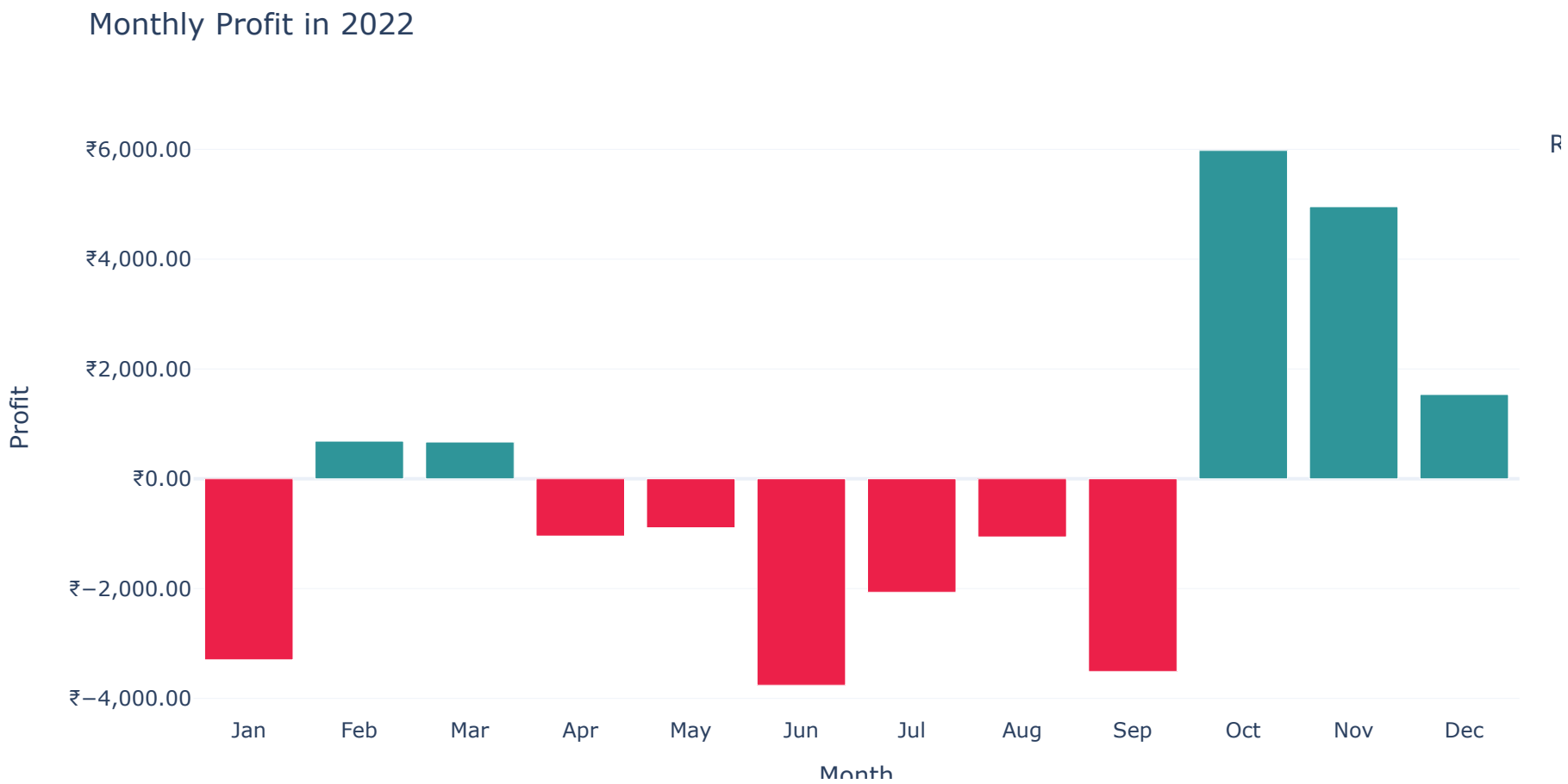
the code extracts year and month information from the 'Order Date' column, adds corresponding columns to the DataFrame, groups the data by year and month, calculates the sum for each group, and sorts the resulting DataFrame.

In [24]:
```
df['Year'] = pd.DatetimeIndex(df['Order Date']).year
df['Month_Number'] = pd.DatetimeIndex(df['Order Date']).month
df['Month'] = df['Month_Number'].apply(lambda x: calendar.month_abbr[x])
year_month = df.groupby(['Year', 'Month','Month_Number']).sum().sort_values(['Year','Month_Number'])
year_month
```

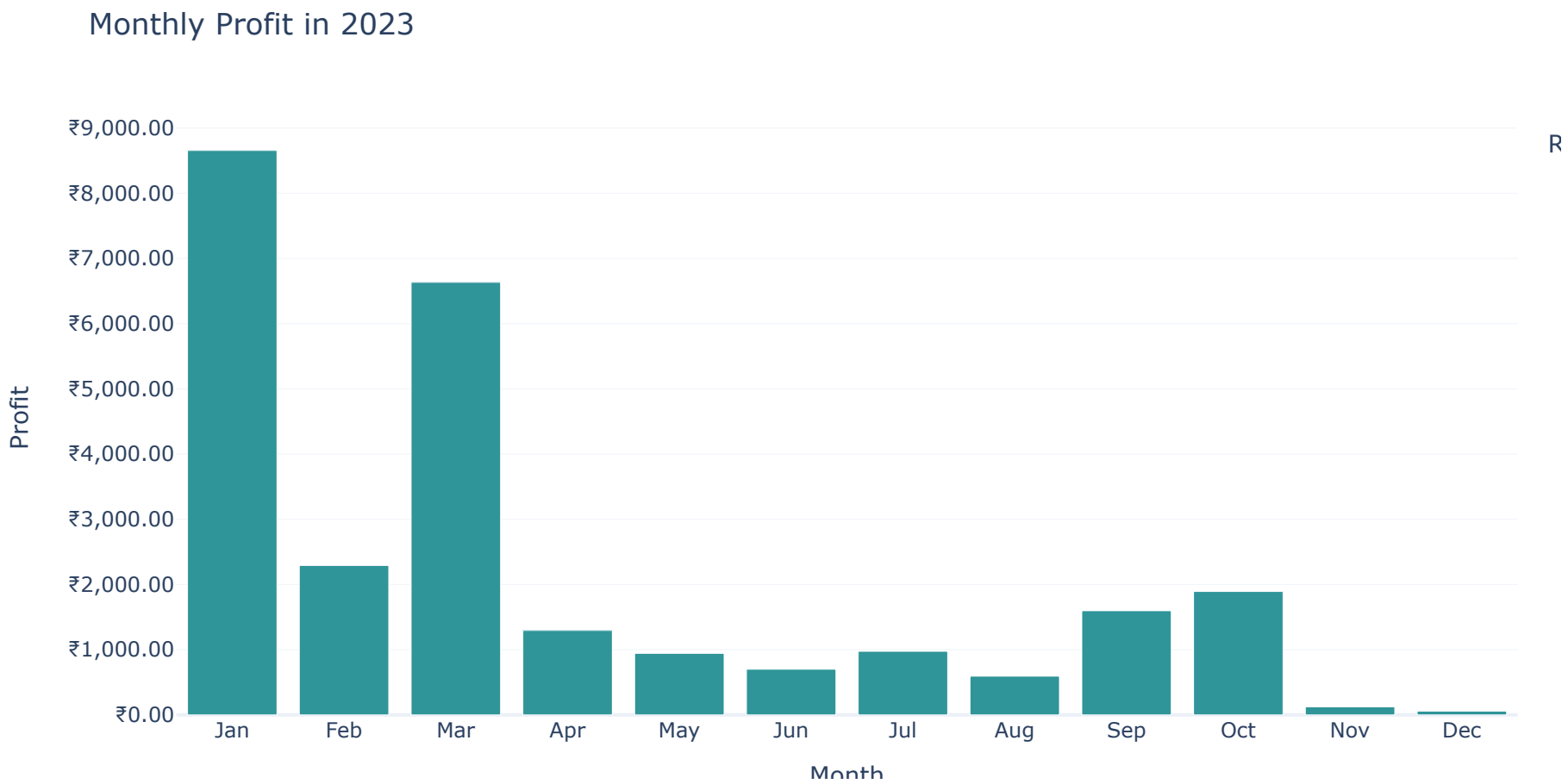
Out[24]:

			Amount	Profit	Quantity
Year	**Month**	**Month_Number**			
2022	**Jan**	**1**	18035	-3296	203
	Feb	**2**	6566	685	58
	Mar	**3**	7434	669	144
	Apr	**4**	26170	-1043	337
	May	**5**	20422	-891	306
	Jun	**6**	17406	-3759	353
	Jul	**7**	15682	-2065	239
	Aug	**8**	45269	-1059	601
	Sep	**9**	20210	-3509	310
	Oct	**10**	32758	5979	414
	Nov	**11**	38858	4955	433
	Dec	**12**	23892	1535	209
2023	**Jan**	**1**	50448	8655	640
	Feb	**2**	15894	2291	253
	Mar	**3**	39700	6633	485
	Apr	**4**	11079	1295	106
	May	**5**	4390	943	63
	Jun	**6**	3392	700	52
	Jul	**7**	5116	975	67
	Aug	**8**	6557	594	83
	Sep	**9**	5583	1597	70
	Oct	**10**	14147	1892	149
	Nov	**11**	2235	122	35
	Dec	**12**	259	57	5

In [25]:
```
year_month = year_month.reset_index()
year_month["Color"] = np.where(year_month["Profit"]<0, 'Loss', 'Profit')
year_month_2022 = year_month[year_month['Year']==2022]
fig = px.bar(year_month_2022, x='Month_Number', y='Profit', color='Color',
             title="Monthly Profit in 2022",
             labels=dict(Month_Number="Month", Profit="Profit", Color="Results"),
             color_discrete_map={
                 'Loss': '#EC2049',
                 'Profit': '#2F9599'},
             hover_data=["Month", "Profit"],
             template='plotly_white')

fig.update_layout(yaxis_tickprefix = '₹', yaxis_tickformat = ',.2f')

fig.update_layout(
    xaxis = dict(
        tickvals = [1, 2, 3, 4, 5, 6, 7,8 ,9, 10, 11, 12],
        ticktext = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    )
)
fig.show()
```


Monthly Profit in 2022
₹6,000.00
₹4,000.00
₹2,000.00
₹0.00
₹−2,000.00
₹−4,000.00
Profit
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month

In [26]:
```
year_month_2023 = year_month[year_month['Year']==2023]
fig = px.bar(year_month_2023, x='Month_Number', y='Profit', color='Color',
             title="Monthly Profit in 2023",
             labels=dict(Month_Number="Month", Profit="Profit", Color="Results"),
             color_discrete_map={
                 'Loss': '#EC2049',
                 'Profit': '#2F9599'},
             hover_data=["Month", "Profit"],
             template='plotly_white')

fig.update_layout(yaxis_tickprefix = '₹', yaxis_tickformat = ',.2f')

fig.update_layout(
    xaxis = dict(
        tickvals = [1, 2, 3, 4, 5, 6, 7,8 ,9, 10, 11, 12],
        ticktext = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    )
)
fig.show()
```

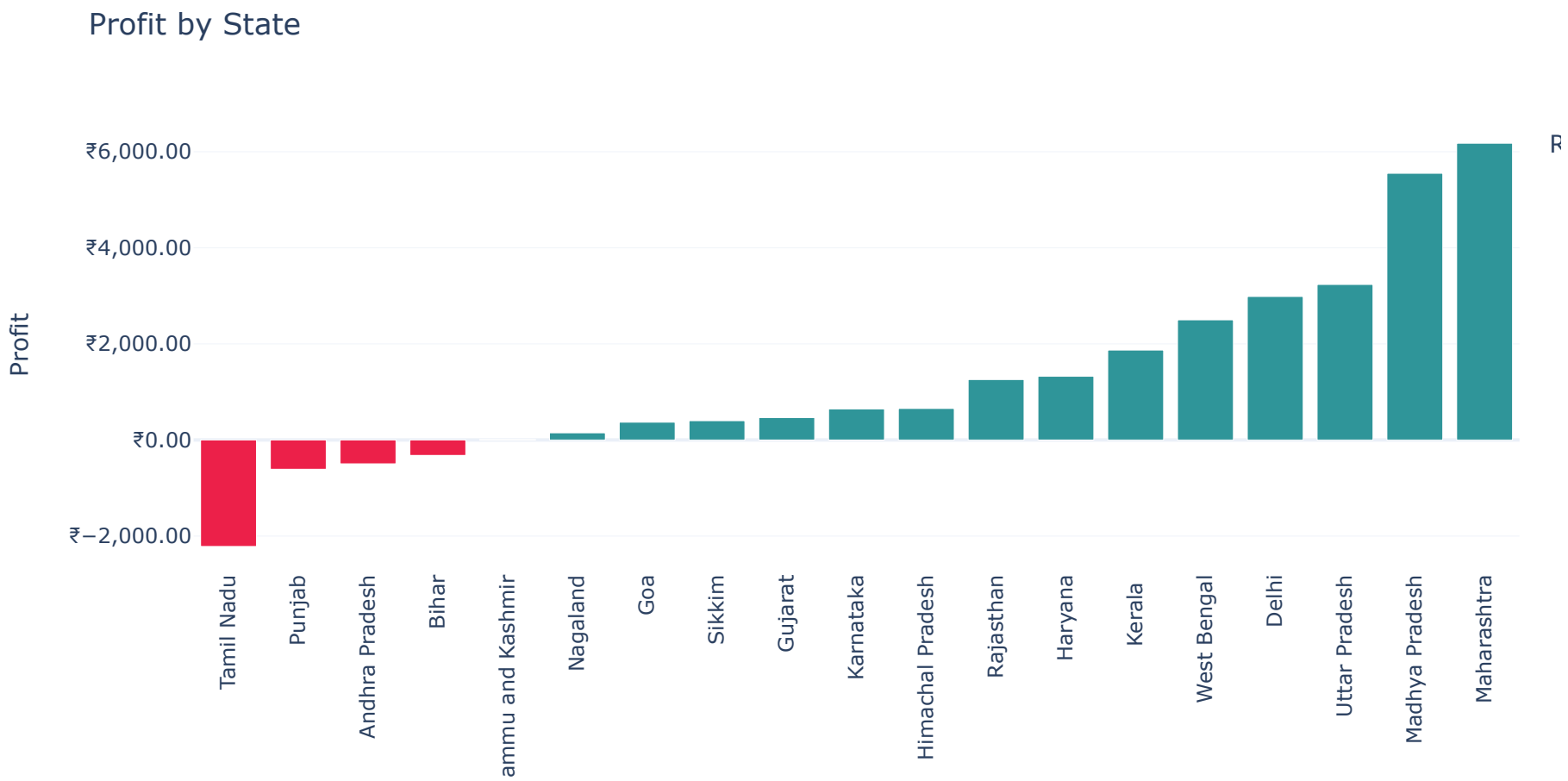

Monthly Profit in 2023
Profit
₹9,000.00
₹8,000.00
₹7,000.00
₹6,000.00
₹5,000.00
₹4,000.00
₹3,000.00
₹2,000.00
₹1,000.00
₹0.00
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month

In [27]:
```
orders_by_state = order.groupby(['State']).size().reset_index(name='Total Orders').sort_values(['Total Orders'])
orders_by_state
```

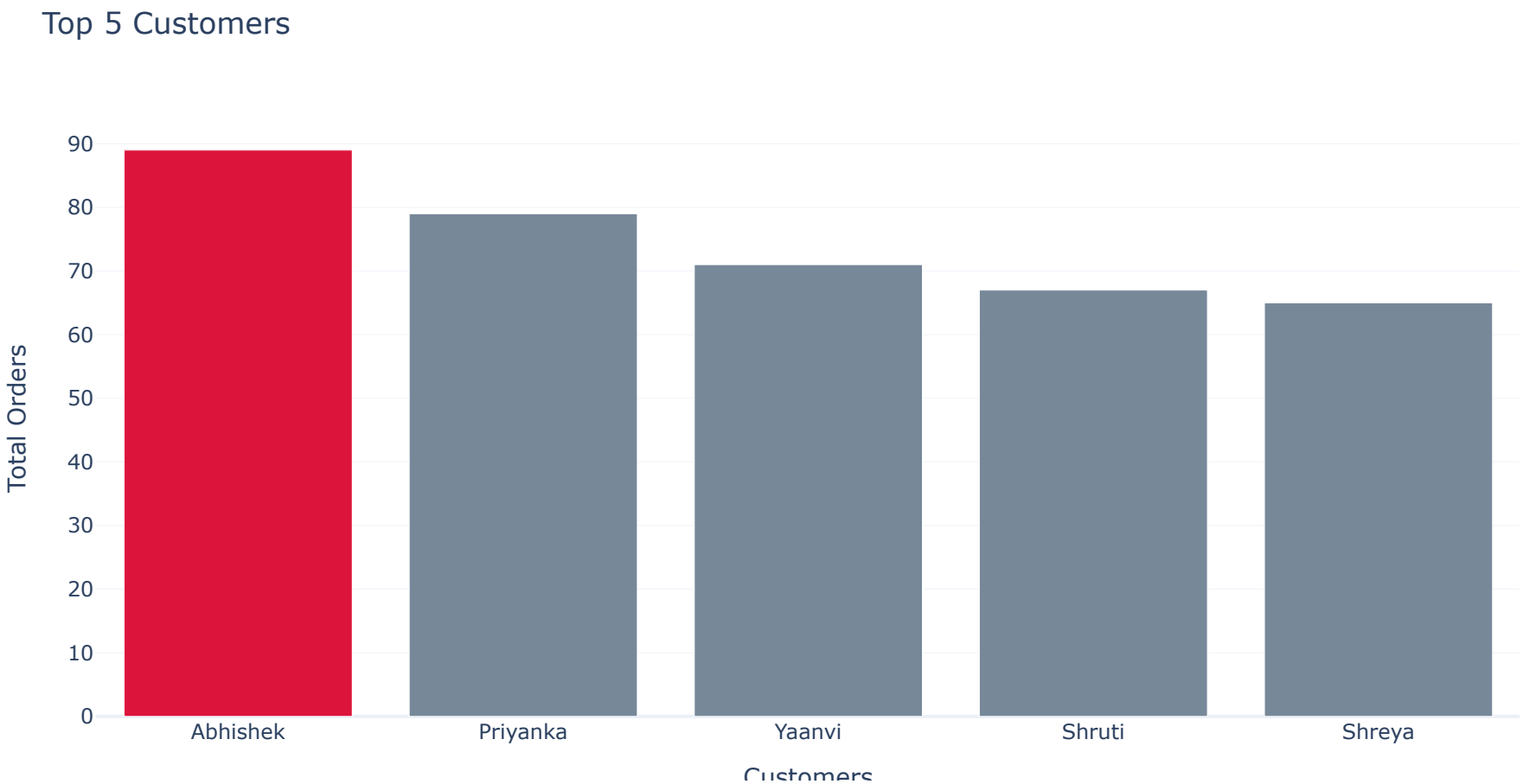
Out[27]:

	State	Total Orders
16	Tamil Nadu	8
15	Sikkim	12
3	Goa	14
5	Haryana	14
6	Himachal Pradesh	14
7	Jammu and Kashmir	14
0	Andhra Pradesh	15
12	Nagaland	15
9	Kerala	16
1	Bihar	16
8	Karnataka	21
17	Uttar Pradesh	22
2	Delhi	22
18	West Bengal	22
13	Punjab	25
4	Gujarat	27
14	Rajasthan	32
11	Maharashtra	90
10	Madhya Pradesh	101

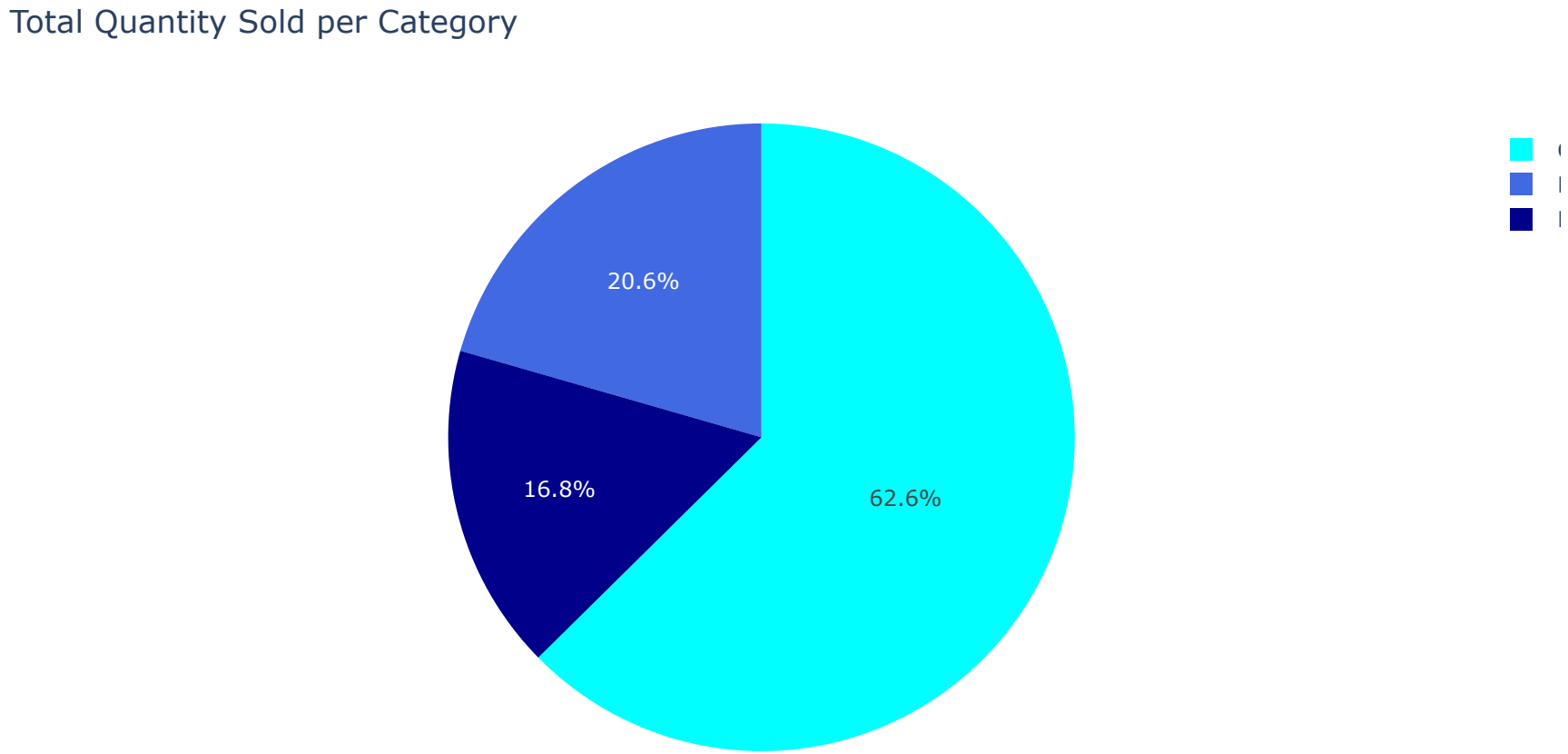
In [28]:
```
profit_by_state = df.groupby('State').sum().reset_index().sort_values(['Profit'])
profit_by_state["Color"] = np.where(profit_by_state["Profit"]<0, 'Loss', 'Profit')


fig = px.bar(profit_by_state, x='State', y='Profit',
             color='Color', color_discrete_map={
                 'Loss': '#EC2049',
                 'Profit': '#2F9599'},
             title="Profit by State",
             labels=dict(Color="Results"),
             template='plotly_white')

# Disabling Zoom
fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True

fig.update_layout(yaxis_tickprefix = '₹', yaxis_tickformat = ',.2f')

fig.update_xaxes(
        tickangle = -90,
        title_text = "States",
)


fig.show()
```


Profit by State
Profit
₹6,000.00
₹4,000.00
₹2,000.00
₹0.00
₹−2,000.00
Tamil Nadu
Punjab
Andhra Pradesh
Bihar
ammu and Kashmir
Nagaland
Goa
Sikkim
Gujarat
Karnataka
Himachal Pradesh
Rajasthan
Haryana
Kerala
West Bengal
Delhi
Uttar Pradesh
Madhya Pradesh
Maharashtra

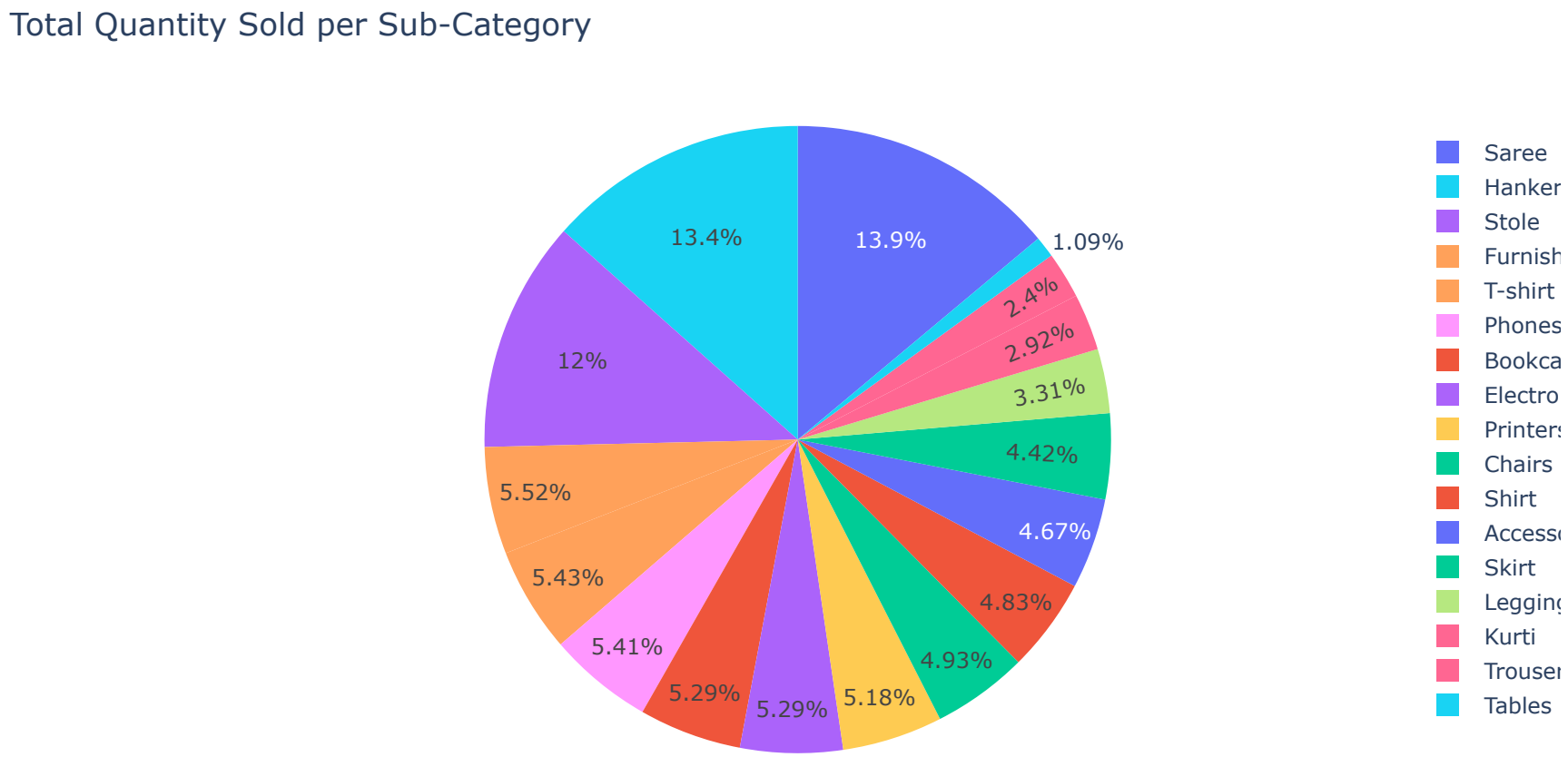
In [29]:
```
top_customers = df.groupby('CustomerName').sum().reset_index().sort_values(['Quantity'], ascending=False).head(5)

colors = ['lightslategray',] * 5
colors[0] = 'crimson'

fig = go.Figure(data=[go.Bar(
    x=top_customers['CustomerName'],
    y=top_customers['Quantity'],
    marker_color=colors # marker color can be a single color value or an iterable,
)])
fig.update_layout(title_text='Top 5 Customers',
                  template='plotly_white')
fig.update_xaxes(title_text='Customers')
fig.update_yaxes(title_text='Total Orders')

fig.show()
```


Top 5 Customers
Total Orders
90
80
70
60
50
40
30
20
10
0
Abhishek
Priyanka
Yaanvi
Shruti
Shreya
Customers

In [30]:
```
details_category = details.groupby('Category').sum().reset_index()
fig = px.pie(details_category, values='Quantity', names='Category', color='Category',
             color_discrete_map={'Clothing':'cyan',
                                 'Electronics':'royalblue',
                                 'Furniture':'darkblue'},
            title='Total Quantity Sold per Category')
fig.show()
```



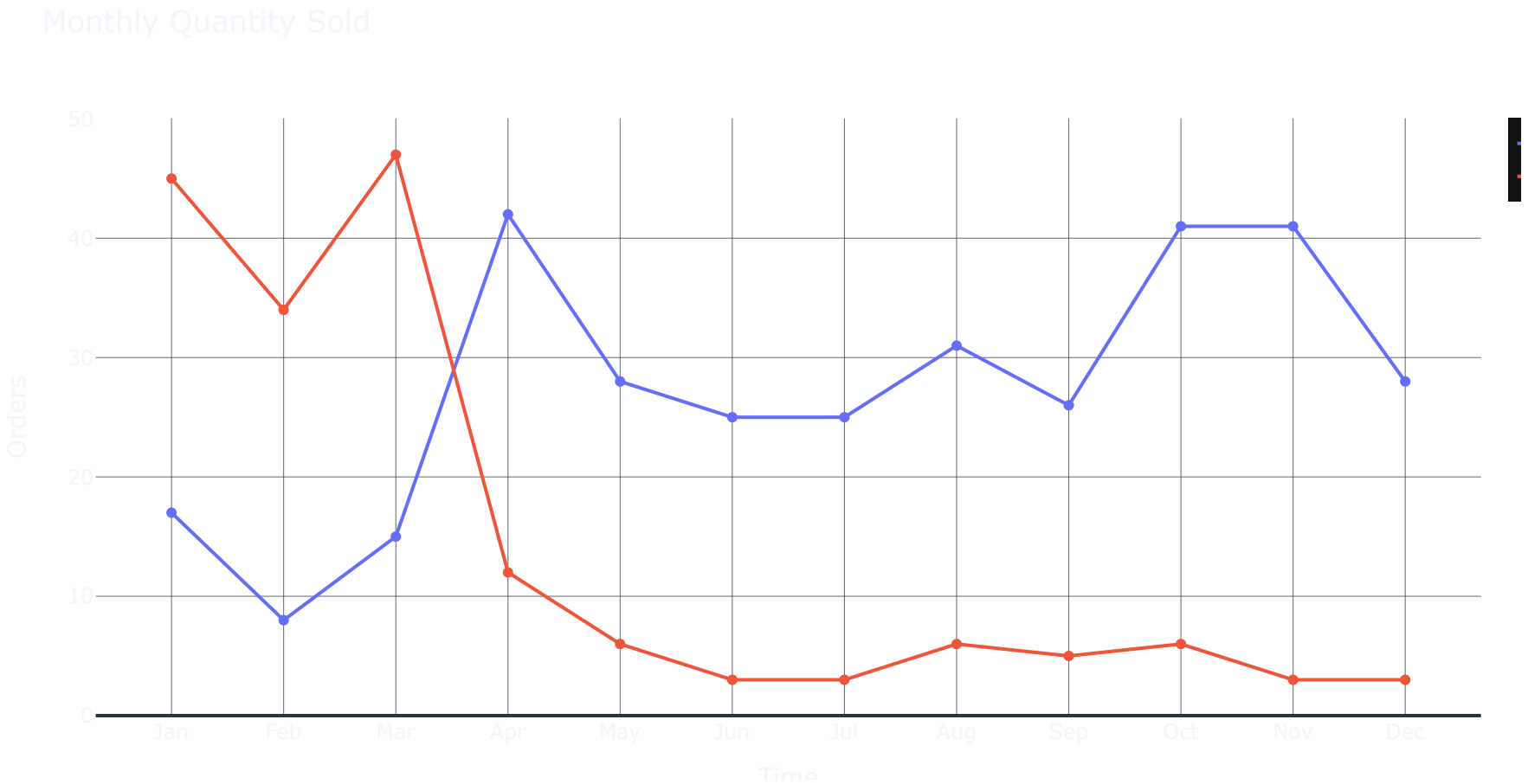
In [31]:
```
details_subcategory = details.groupby('Sub-Category').sum().reset_index()
fig = px.pie(details_subcategory, values='Quantity', names='Sub-Category', color='Sub-Category',
             title='Total Quantity Sold per Sub-Category')
fig.show()
```



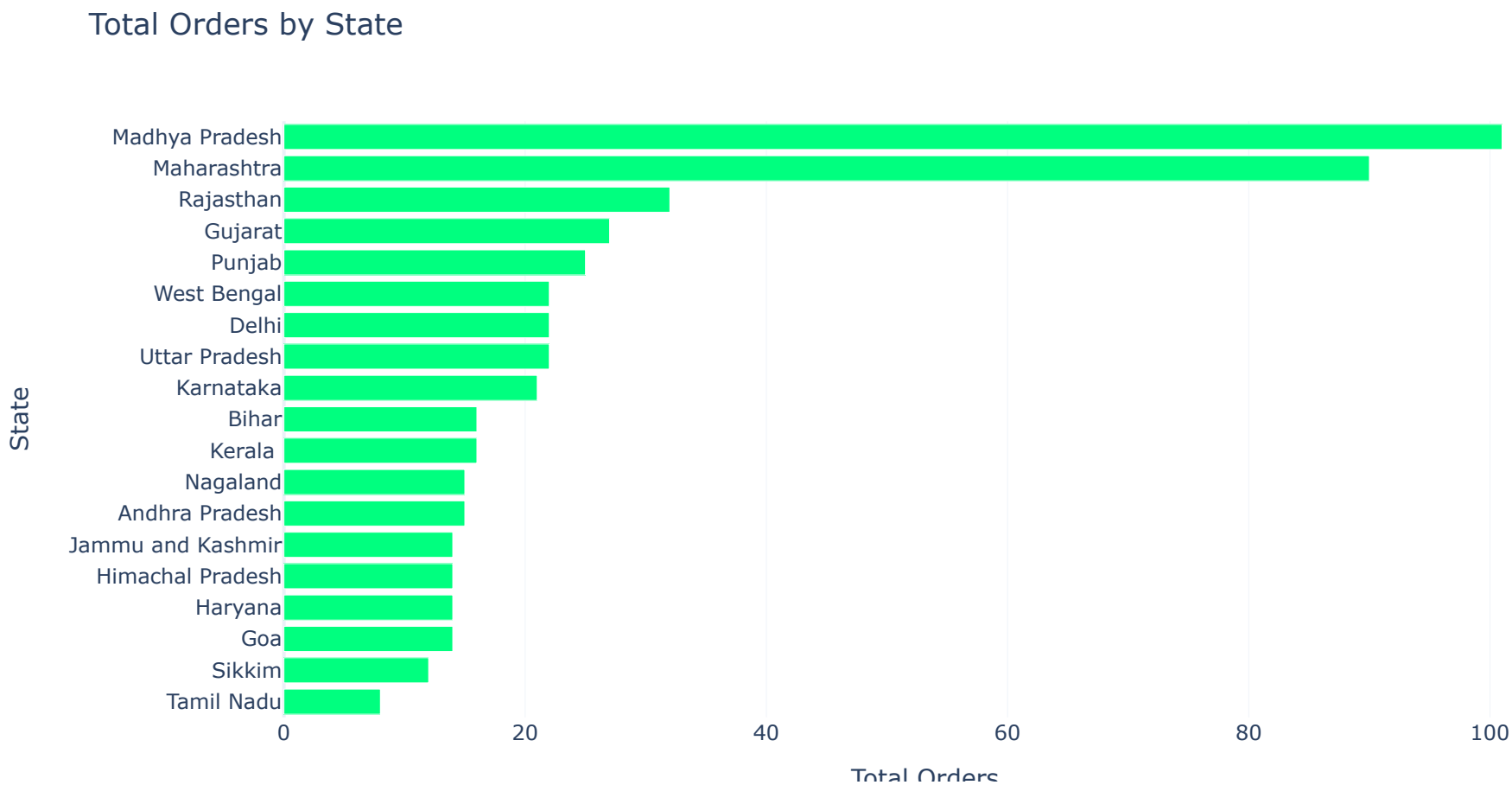
In [32]:

```
date_orders = order.groupby('Order Date').size().reset_index(name="Orders")
date_orders['Month'] = pd.DatetimeIndex(date_orders['Order Date']).month
date_orders['Year'] = pd.DatetimeIndex(date_orders['Order Date']).year

date_orders_2022 = date_orders[date_orders['Year']==2022]
date_orders_2023 = date_orders[date_orders['Year']==2023]

month_2022 = date_orders_2022.groupby('Month').sum().reset_index()
month_2023 = date_orders_2023.groupby('Month').sum().reset_index()

fig = go.Figure()
fig.add_trace(go.Scatter(
    name='2022',
    x=month_2022['Month'],
    y=month_2022['Orders'],
    connectgaps=True # override default to connect the gaps
))
fig.add_trace(go.Scatter(
    name='2023',
    x=month_2023['Month'],
    y=month_2023['Orders'],
    connectgaps=True # override default to connect the gaps
))
fig.update_layout(title_text='Monthly Quantity Sold',
                  template='plotly_dark')
fig.update_xaxes(title_text='Time')
fig.update_yaxes(title_text='Orders')
fig.update_layout(
    xaxis = dict(
        tickvals = [1, 2, 3, 4, 5, 6, 7,8 ,9, 10, 11, 12],
        ticktext = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
    )
)

fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True
```

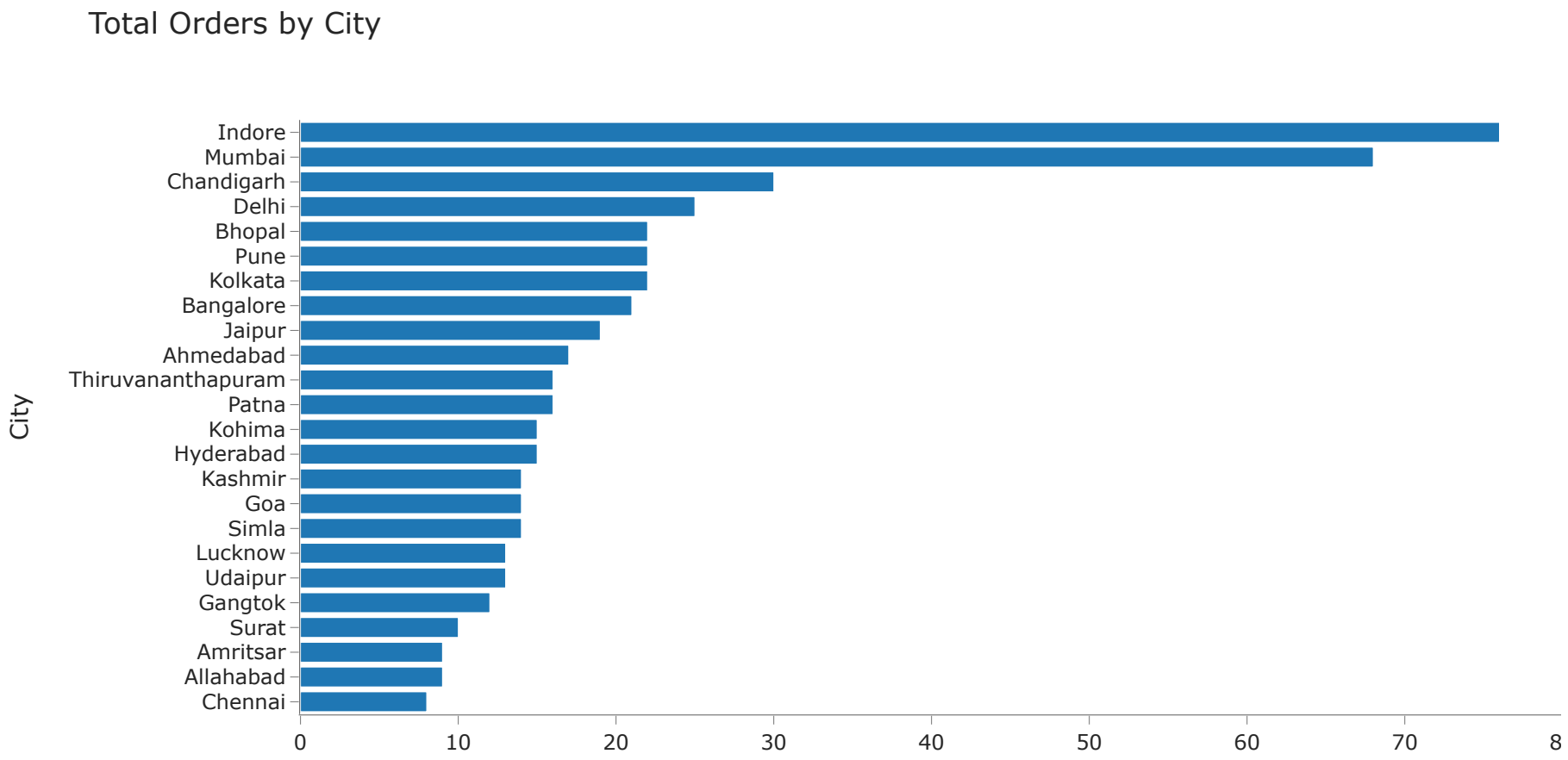
```
fig.show()
```


Monthly Quantity Sold
50
40
30
20
10
0
Orders
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Time

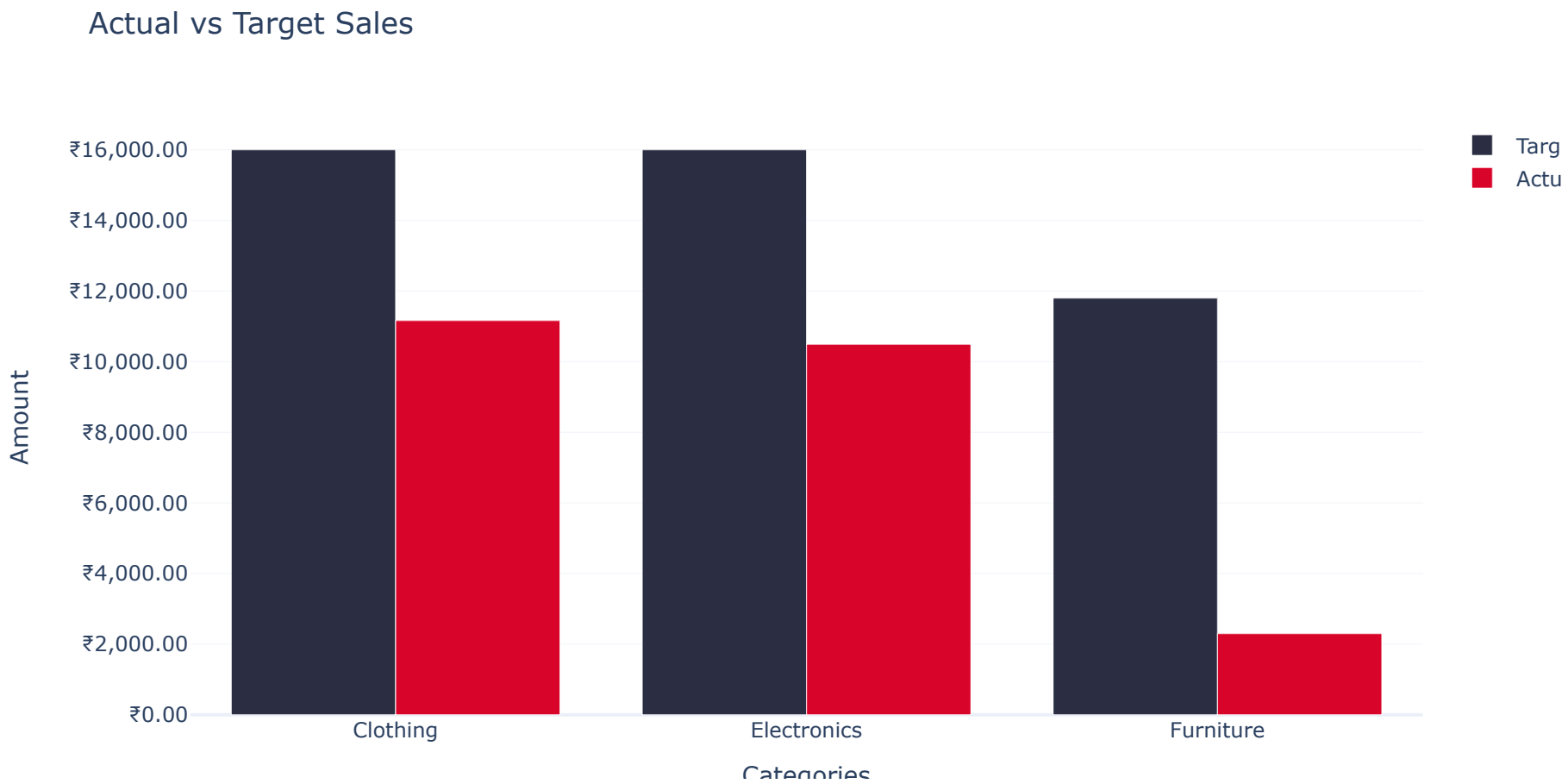
In [33]:
```
fig = px.bar(orders_by_state, y='State', x='Total Orders',
             title="Total Orders by State",
             color_discrete_sequence=["springgreen"],
             template='plotly_white')

# Disabling Zoom
fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True

fig.show()
```


Total Orders by State
Madhya Pradesh
Maharashtra
Rajasthan
Gujarat
Punjab
West Bengal
Delhi
Uttar Pradesh
Karnataka
Bihar
Kerala
Nagaland
Andhra Pradesh
Jammu and Kashmir
Himachal Pradesh
Haryana
Goa
Sikkim
Tamil Nadu
State
0
20
40
60
80
100
Total Orders

In [34]:
```
orders_by_city = order.groupby(['City']).size().reset_index(name='Total Orders').sort_values(['Total Orders'])

fig = px.bar(orders_by_city, y='City', x='Total Orders',
             title="Total Orders by City",
             template='simple_white')

fig.layout.yaxis.tickmode='linear'
# Disabling Zoom
fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True

fig.show()
```


Total Orders by City
City
Indore
Mumbai
Chandigarh
Delhi
Bhopal
Pune
Kolkata
Bangalore
Jaipur
Ahmedabad
Thiruvananthapuram
Patna
Kohima
Hyderabad
Kashmir
Goa
Simla
Lucknow
Udaipur
Gangtok
Surat
Amritsar
Allahabad
Chennai
0
10
20
30
40
50
60
70
8

In [35]:
```
target_category = target.groupby('Category').max().reset_index()
details_category = details.groupby('Category').sum().reset_index()

target_category['Actual_Amount'] = details_category['Profit']

fig = go.Figure(data=[
    go.Bar(name='Target', x=target_category['Category'], y=target_category['Target'],
           marker_color='#2b2d42'),
    go.Bar(name='Actual Amount', x=target_category['Category'], y=target_category['Actual_Amount'],
           marker_color='#d90429')
])

fig.update_layout(title_text='Actual vs Target Sales',
                  template='plotly_white')

fig.update_xaxes(title_text='Categories')
fig.update_yaxes(title_text='Amount')

fig.update_layout(yaxis_tickprefix = '₹', yaxis_tickformat = ',.2f')


fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True

fig.show()
```


Actual vs Target Sales
Targ
Actu
Amount
₹16,000.00
₹14,000.00
₹12,000.00
₹10,000.00
₹8,000.00
₹6,000.00
₹4,000.00
₹2,000.00
₹0.00
Clothing
Electronics
Furniture
Categories

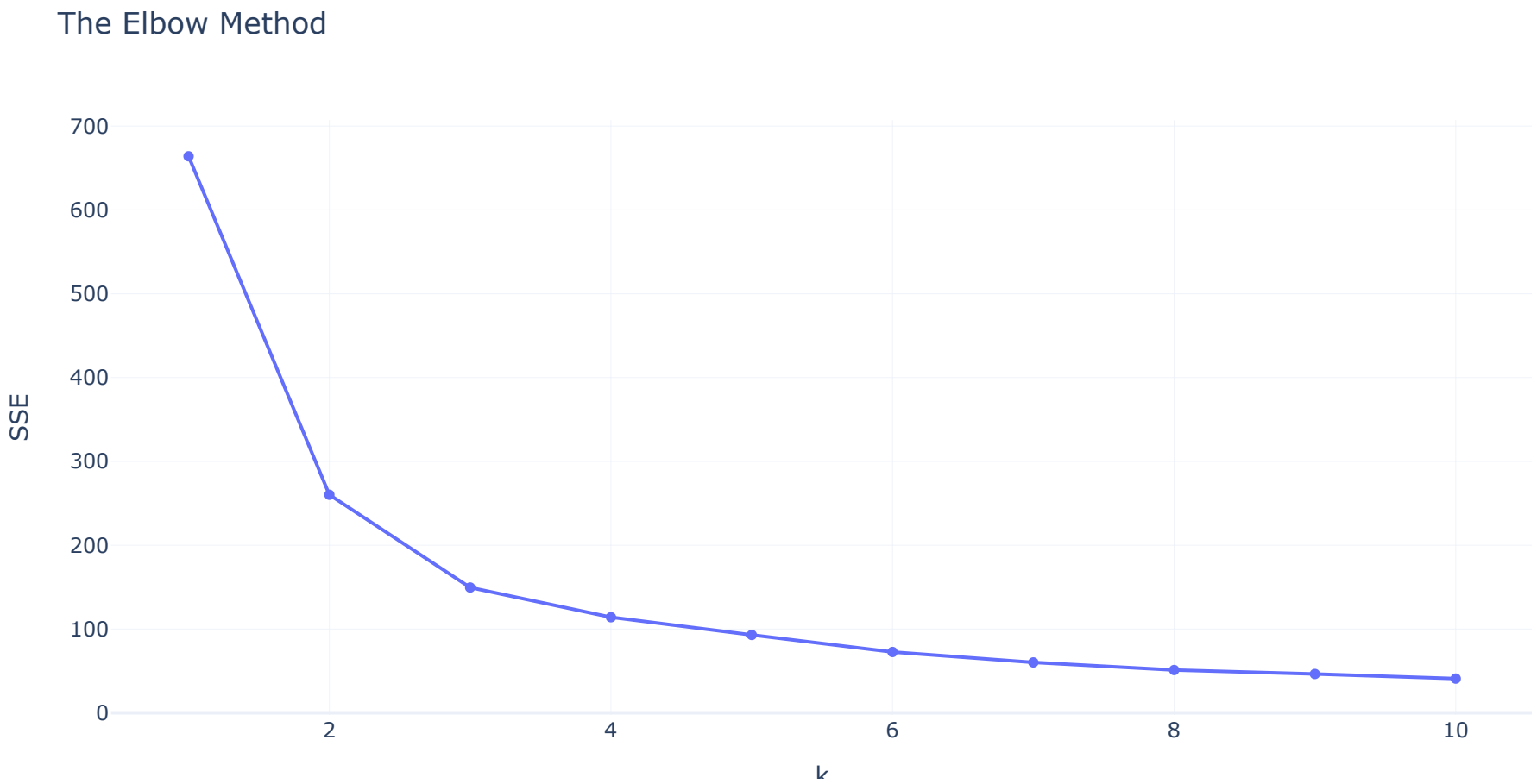
In [36]:
```
customer_seg = df.groupby('CustomerName').sum().reset_index()
customer_seg = customer_seg[['CustomerName', 'Amount', 'Quantity']]
customer_seg.head()
```

Out[36]:

	CustomerName	Amount	Quantity
0	Aakanksha	74	8
1	Aarushi	4701	49
2	Aashna	1931	32
3	Aastha	3276	28
4	Aayush	556	18

In [37]:
```
# Standardizing
customer_seg2 = customer_seg[['Amount', 'Quantity']]
scaler = StandardScaler()
scaler.fit(customer_seg2)

customers_normalized = scaler.transform(customer_seg2)
customers_normalized

# Elbow Method to find best number of clusters
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(customers_normalized)
    sse[k] = kmeans.inertia_ # SSE to closest cluster centroid

# Plotting SSE
fig = go.Figure()
fig.add_trace(go.Scatter(
    x=list(sse.keys()),
    y=list(sse.values()),
    connectgaps=True # override default to connect the gaps
))

fig.update_layout(title_text='The Elbow Method',
                  template='plotly_white')
fig.update_xaxes(title_text='k')
fig.update_yaxes(title_text='SSE')
fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True

fig.show()
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can
avoid it by setting the environment variable OMP_NUM_THREADS=2.
```


The Elbow Method
SSE
700
600
500
400
300
200
100
0
2
4
6
8
10
K

In [38]:
```
# KMeans
model = KMeans(n_clusters=3)
model.fit(customers_normalized)
customer_seg['Cluster'] = model.labels_ + 1
customer_seg['Cluster'] = customer_seg['Cluster'].astype('category')
customer_seg.head()
```

Out[38]:

	CustomerName	Amount	Quantity	Cluster
0	Aakanksha	74	8	1
1	Aarushi	4701	49	2
2	Aashna	1931	32	3
3	Aastha	3276	28	3
4	Aayush	556	18	1

In [39]:
```
customer_seg.groupby('Cluster').agg({
    'Amount':'mean',
    'Quantity':'count'}).round(2)
```

Out[39]:

	Amount	Quantity
Cluster		
1	451.36	208
2	5237.32	28
3	1989.32	96

In [40]:
```
fig = px.scatter(customer_seg, x="Quantity", y="Amount",
                 color="Cluster",
                 template='plotly_white',
                 title="Amount vs Quantity - Customer Segmentation")
fig.layout.xaxis.fixedrange = True
fig.layout.yaxis.fixedrange = True

fig.show()
```

In [41]:
```
details.columns
```

Out[41]:
```
Index(['Order ID', 'Amount', 'Profit', 'Quantity', 'Category', 'Sub-Category'], dtype='object')
```

In [42]:
```
# constructing a heatmap to understand the correlation

plt.figure(figsize=(10,10))
sns.heatmap(details.corr(), cmap='Blues',annot = True)
```

Out[42]: <AxesSubplot:>