In [1]:

```
# Importing basic packages
import os
import warnings
import requests
import numpy as np
import pandas as pd
import pandas as pd
import calendar
import datetime

# Visualisations Libraries
import matplotlib.pyplot as plt
import plotly.express as px
import squarify
import seaborn as sns
from pprint import pprint as pp
from plotly.subplots import make_subplots
import plotly.graph_objects as go

#Import Dash
import dash
import dash_core_components as dcc
from dash import html

#Dash Boot Strap components installing for complex mult page application building
import dash_bootstrap_components as dbc

# Image Process package
import base64
from PIL import Image
```

```
C:\Users\prapa001\Anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.24.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/4190273687.py:22: UserWarning:
The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`
  import dash_core_components as dcc
```

In [2]:

```
# !/usr/bin/env/ python
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
import urllib
import pyodbc

# import tqdm as tqdm
import snowflake.connector
from sqlalchemy import create_engine
from snowflake.sqlalchemy import URL
import pandas as pd
import numpy as n
import os
import json
from datetime import date
```

In [3]:

```
# SQL and snow flake connection
os.chdir("C:\\Users\\prapa001\\OneDrive - Corporate\\Desktop\\python_trials")

credentials= json.load(open("credentials.json"))

cnxn_str = ("Driver={ODBC Driver 17 for SQL Server};"
            "Server=WINMPNDBp02;"
            "Database=ANALYSIS_PROJECTS;"
            "UID="+credentials['SQL']['user'] + ";"
            +"pwd=" + credentials['SQL']['password'] +";" +
            "Trusted_Connection=Yes;"
           )
sql_connection = pyodbc.connect(cnxn_str)

sf_connection = snowflake.connector.connect(
    user =credentials['SF']['user'],
    password=credentials['SF']['password'] ,
    role='SF_SCM_ANALYTICS_DBRL',
    account='staples.east-us-2.azure',
    warehouse='CAP_PRD_SC_WH',
    database='DATALAB_SANDBOX',
    schema='SCM_ANALYTICS',
    authenticator='externalbrowser'
    )


engine = create_engine(URL(
    user =credentials['SF']['user'],
    password=credentials['SF']['password'],
    role='SF_SCM_ANALYTICS_DBRL',
    account='staples.east-us-2.azure',
    warehouse='CAP_PRD_SC_WH',
    database='DATALAB_SANDBOX',
    schema='SCM_ANALYTICS',
    authenticator='externalbrowser'
))
```

```
Initiating login request with your identity provider. A browser window should have opened for you to complete the login. If you can't see it, check
existing browser windows, or your OS settings. Press CTRL+C to abort and try again...
```

In [4]:
```
#CTS Connecting
Query =  '''Select Fiscal_Year, Fiscal_Period,
       SUM(ADJUSTED_NET_SALES_W_COU_AMT_$) AS Net_Sales,
       SUM(Sales_$_SMS_COS) AS COGS,
       SUM(Sales_Total_Units_Net) AS Net_Units,
       SUM(Total_Distribution_Costs) AS Distribution_Costs,
       SUM(FC_Variable_Handling_Expense_Final) AS Distribution_Variable_Expense,
       SUM(Fixed_Expense_Final) AS Distribution_Expense_Final,
       SUM(Total_Delivery_Costs) AS Total_Delivery_Costs,
       SUM(Contribution_Margin) AS Contribution_Margin,
       SUM(Total_Delivery_Costs)/SUM(Sales_Total_Units_Net) AS Delivery_Cost_Per_Unit,
       SUM(Total_Distribution_Costs)/SUM(Sales_Total_Units_Net) AS Distributiob_Cost_Per_Unit
From linked.CTS_1P.[CTS_2.0_All_BUs_Data_V]
WHERE Master_Customer_Number = '1876074' --AND Fiscal_Year = '2022'
GROUP BY Fiscal_Year,Fiscal_Period'''
```

In [5]:
```
#
CTS_Master =  pd.read_sql(Query,sql_connection)
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/3409404880.py:2: UserWarning: pandas only supports SQLAlchemy connectable (engine/connectio
n) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  CTS_Master =  pd.read_sql(Query,sql_connection)
```

In [6]:
```
# Sales for last 12 Period
CTS_Master
# Filter by fiscal year
Fiscal_Year_CTS = CTS_Master[CTS_Master['Fiscal_Year'] == 2022]
Fiscal_Year_CTS
```

Out[6]:

	Fiscal_Year	Fiscal_Period	Net_Sales	COGS	Net_Units	Distribution_Costs	Distribution_Variable_Expense	Distribution_Expense_Final	Total_Delivery_Costs	Contribution
0	2022	6	526395.8309	-6.468677e+05	18599.0	-35643.6824	-8498.0270	-27086.3166	-64552.7247	15854
1	2022	5	660663.3742	-8.325368e+05	24767.0	-41501.7146	-11041.4365	-30399.6497	-41174.1728	27880
2	2022	11	744402.9612	-8.781144e+05	26324.0	-47837.8362	-12927.8416	-34848.0762	-79930.0450	-2691
3	2022	3	560014.5415	-6.689822e+05	21738.0	-30154.5230	-10719.5208	-19355.0222	-39654.9412	35857
4	2022	9	599959.0111	-7.397092e+05	21691.0	-38146.4707	-11079.6407	-27003.6203	-52885.1826	-26582
6	2022	7	744356.3389	-8.750659e+05	29227.0	-49692.6316	-13937.2454	-35680.5662	-69838.4608	14109
9	2022	10	669444.2203	-7.626021e+05	23999.0	-44865.8830	-12489.0850	-32332.9384	-72515.1629	20718
10	2022	4	667136.5442	-7.876751e+05	24822.0	-39289.6595	-12342.8858	-26891.3037	-48888.4939	64265
11	2022	8	851124.4155	-1.012418e+06	32486.0	-52242.9623	-16637.0438	-35547.8685	-70399.5209	13144
12	2022	12	649756.6230	-7.550500e+05	24218.0	-48754.8808	-12311.3796	-36397.0612	-69280.3180	7700
13	2022	1	605163.3179	-7.242944e+05	24129.0	-32193.3402	-12019.5223	-20082.2283	-47018.9390	69329

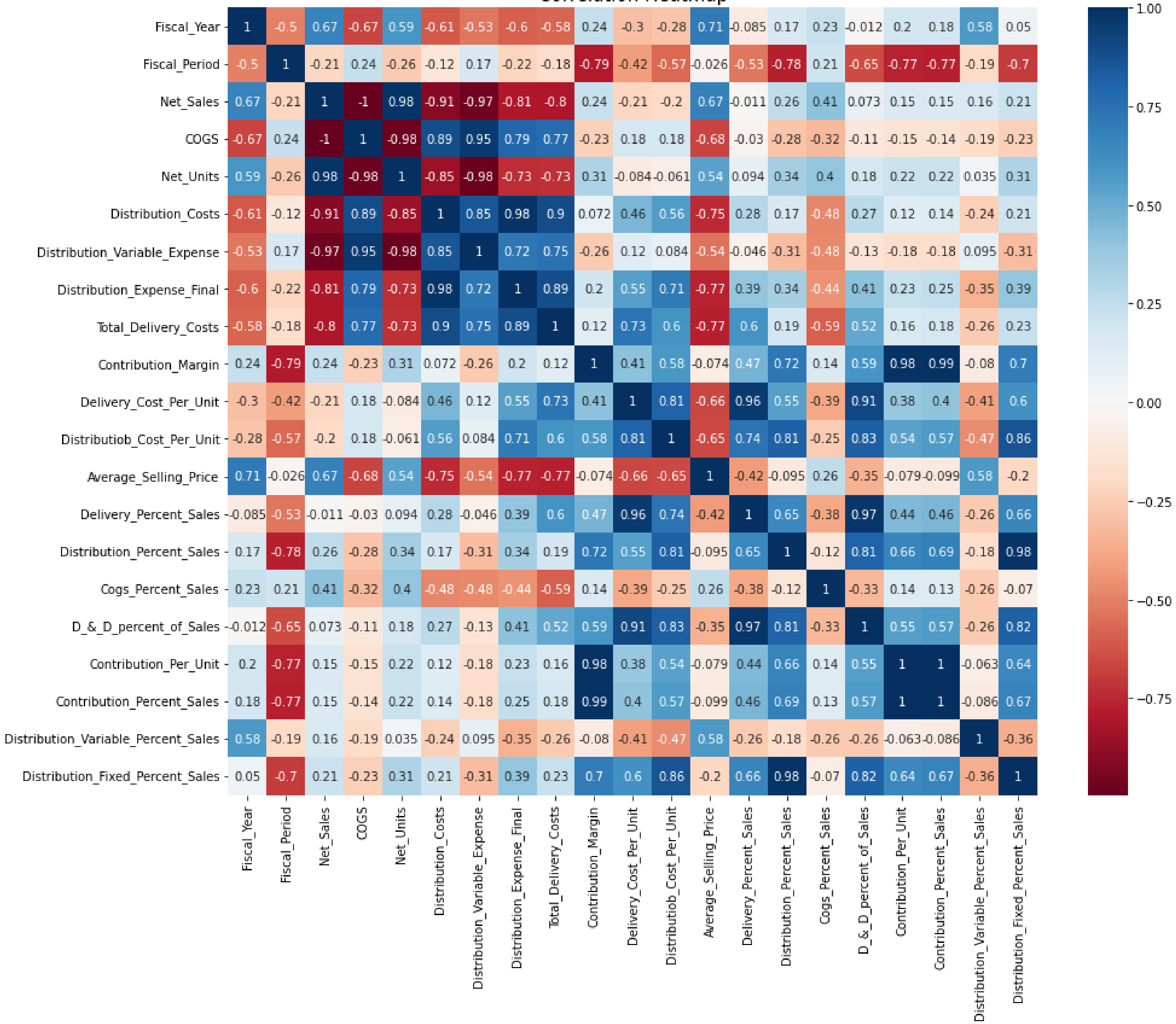
In [7]:

```
# Average Selling Price
CTS_Master["Average_Selling_Price"] = CTS_Master["Net_Sales"]/CTS_Master["Net_Units"]
# Delivery % of Sales
CTS_Master["Delivery_Percent_Sales"] = CTS_Master["Total_Delivery_Costs"]/CTS_Master["Net_Sales"] * 100
# Distribution % of Sales
CTS_Master["Distribution_Percent_Sales"] = CTS_Master["Distribution_Costs"]/CTS_Master["Net_Sales"] *100
#Cogs % of Sales
CTS_Master["Cogs_Percent_Sales"] = CTS_Master["COGS"]/CTS_Master["Net_Sales"] *100
# Delivery and Distribution as percent of Sales combined
CTS_Master["D_&_D_percent_of_Sales"] = CTS_Master["Delivery_Percent_Sales"] + CTS_Master["Distribution_Percent_Sales"]
# Contribution Margin Per Unit
CTS_Master["Contribution_Per_Unit"] = CTS_Master["Contribution_Margin"]/CTS_Master["Net_Units"]
# Contribution Margin As Percent of Sales
CTS_Master["Contribution_Percent_Sales"] = CTS_Master["Contribution_Margin"]/CTS_Master["Net_Sales"] *100
# Variable Distribution Expense
CTS_Master["Distribution_Variable_Percent_Sales"] = CTS_Master["Distribution_Variable_Expense"]/CTS_Master["Net_Sales"] *100
#Fixed Distribution Expense
CTS_Master["Distribution_Fixed_Percent_Sales"] = CTS_Master["Distribution_Expense_Final"]/CTS_Master["Net_Sales"] *100
```

In [8]:

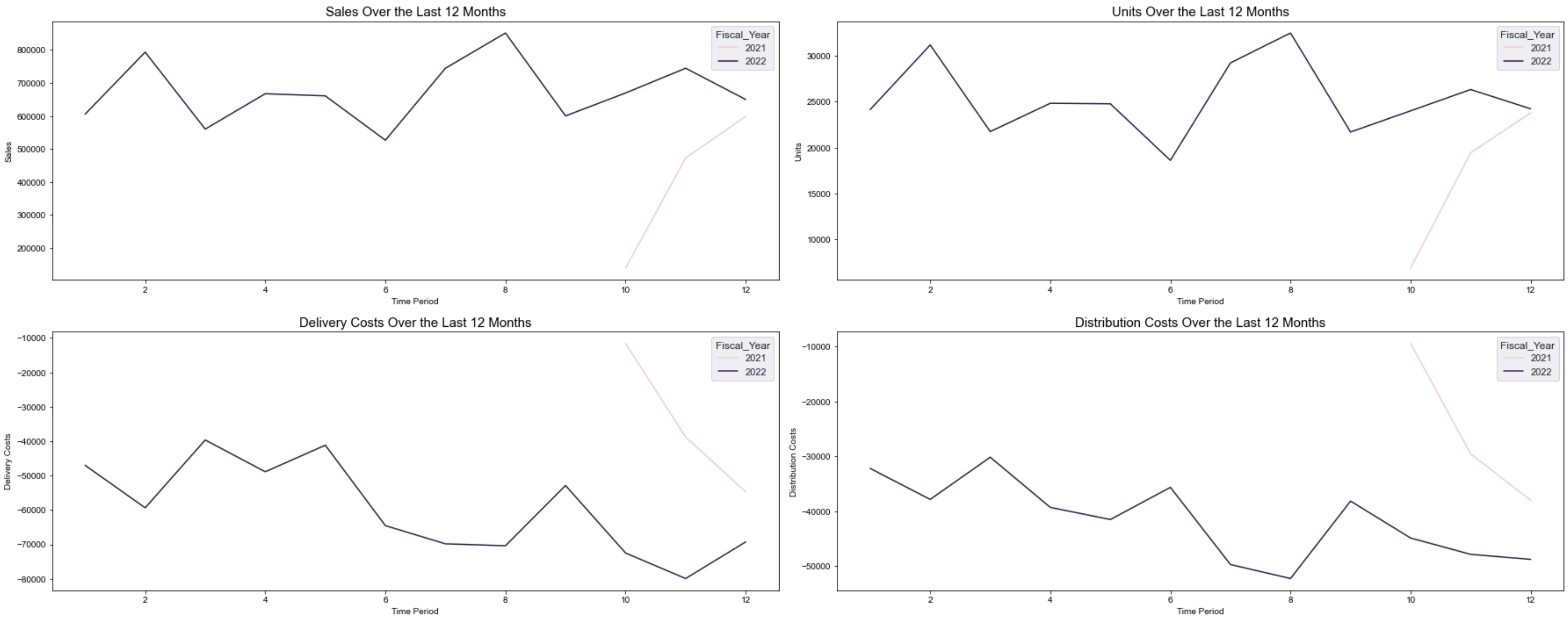
```
# Correlation Heat Map CTS
plt.figure(figsize=(15,12))
sns.heatmap(CTS_Master.corr(),annot=True,cmap='RdBu')
plt.title('Correlation Heatmap',fontsize=15)
plt.yticks(rotation =0)
plt.show()
```

Correlation Heatmap



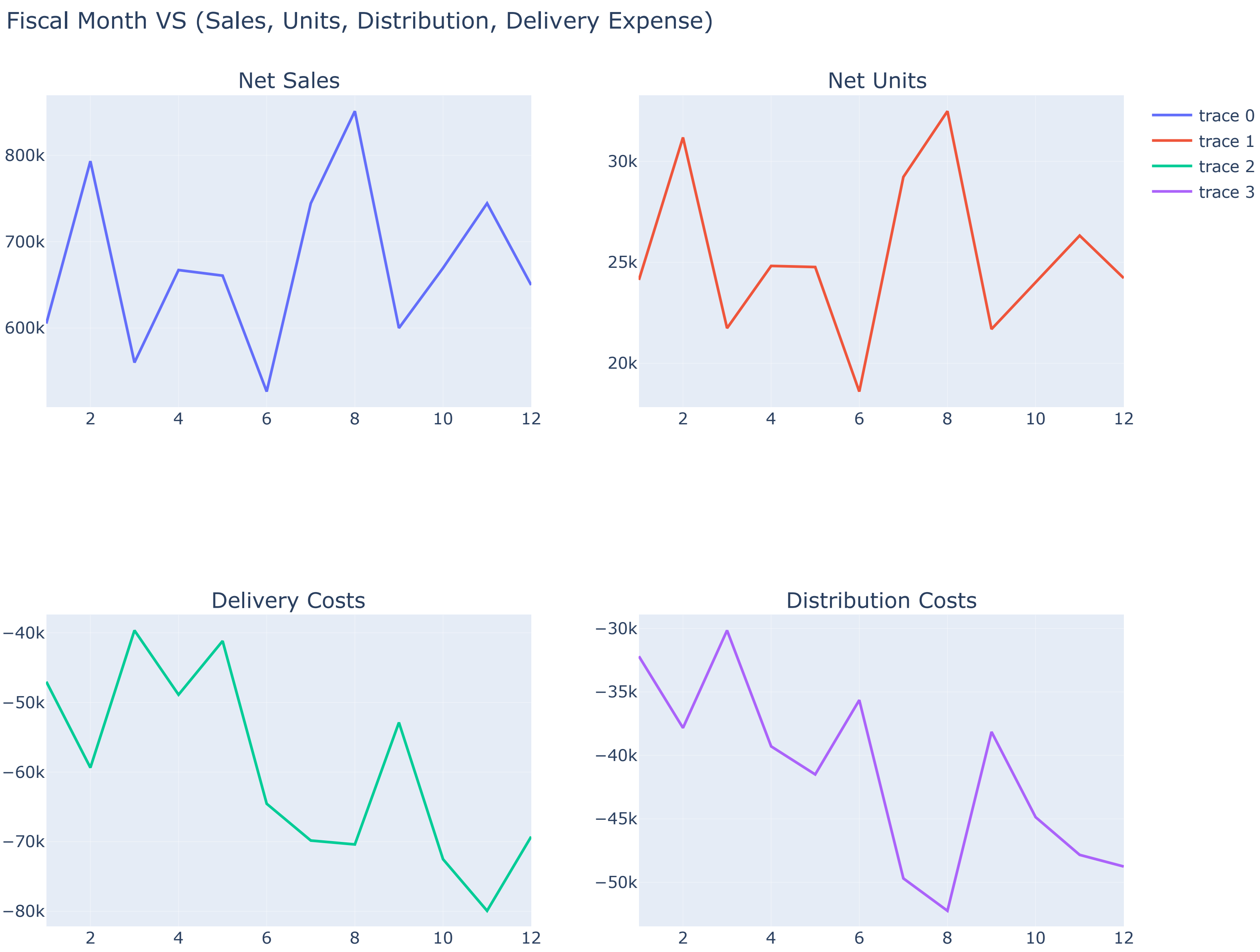
In [9]:

```
# plot the line chart using seaborn
import seaborn as sns

# Create a larger figure
fig, axs = plt.subplots(2, 2, figsize=(25, 10))

# set the background color of the figure to black
sns.set_theme(style='darkgrid')

# Sales plot
sns.lineplot(x='Fiscal_Period', y='Net_Sales',hue='Fiscal_Year', data=CTS_Master, markers=True, dashes=False, ax =axs[0,0])
axs[0,0].set_title('Sales Over the Last 12 Months', size = 15,color = 'black')
axs[0,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,0].set_ylabel('Sales', fontsize=10,color='black')

# Net_Units plot
sns.lineplot(x='Fiscal_Period', y='Net_Units',hue='Fiscal_Year', data=CTS_Master, markers=True, dashes=False,ax =axs[0,1])
axs[0,1].set_title('Units Over the Last 12 Months', size = 15,color = 'black')
axs[0,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,1].set_ylabel('Units', fontsize=10,color='black')

# Delivery  plot
sns.lineplot(x='Fiscal_Period', y='Total_Delivery_Costs',hue='Fiscal_Year', data=CTS_Master, markers=True, dashes=False,ax =axs[1,0])
axs[1,0].set_title('Delivery Costs Over the Last 12 Months', size = 15,color = 'black')
axs[1,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,0].set_ylabel('Delivery Costs', fontsize=10,color='black')

# Distribution plot
sns.lineplot(x='Fiscal_Period', y='Distribution_Costs',hue='Fiscal_Year', data=CTS_Master, markers=True, dashes=False,ax =axs[1,1])
axs[1,1].set_title('Distribution Costs Over the Last 12 Months', size = 15,color = 'black')
axs[1,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,1].set_ylabel('Distribution Costs', fontsize=10,color='black')

# adjust the space between the subplots
fig.tight_layout()
# show the chart
plt.show()
```


Sales Over the Last 12 Months
Sales
800000
700000
600000
500000
400000
300000
200000
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022
Units Over the Last 12 Months
Units
30000
25000
20000
15000
10000
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022
Delivery Costs Over the Last 12 Months
Delivery Costs
−10000
−20000
−30000
−40000
−50000
−60000
−70000
−80000
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022
Distribution Costs Over the Last 12 Months
Distribution Costs
−10000
−20000
−30000
−40000
−50000
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022

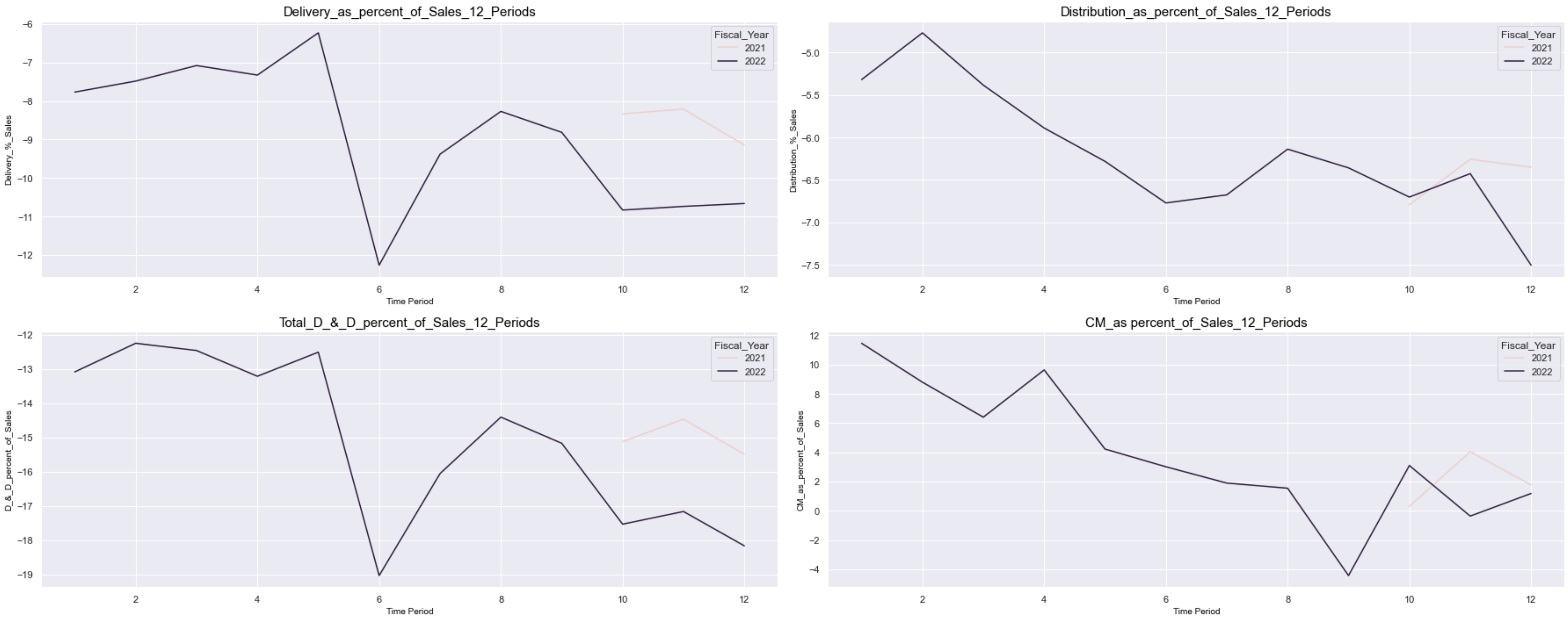
In [10]:

```
# fiscal_year_cts sorting
Fiscal_Year_CTS = Fiscal_Year_CTS.sort_values(by = 'Fiscal_Period',ascending =True)
Fiscal_Year_CTS
# PLOTLY CHARTS
# Converting timeperiod to integer

fig = make_subplots(rows=2, cols=2, start_cell="top-left", subplot_titles=("Net Sales", "Net Units ", "Delivery Costs", "Distribution Costs"))
#for x, Fiscal_Year in CTS_Master.groupby('Fiscal_Year'):
# add line chart to each subplot
fig.add_trace(go.Scatter(x=Fiscal_Year_CTS['Fiscal_Period'], y=Fiscal_Year_CTS['Net_Sales'], mode='lines'), row=1, col=1)
fig.add_trace(go.Scatter(x=Fiscal_Year_CTS['Fiscal_Period'], y=Fiscal_Year_CTS['Net_Units'], mode='lines'), row=1, col=2)
fig.add_trace(go.Scatter(x=Fiscal_Year_CTS['Fiscal_Period'], y=Fiscal_Year_CTS['Total_Delivery_Costs'], mode='lines'), row=2, col=1)
fig.add_trace(go.Scatter(x=Fiscal_Year_CTS['Fiscal_Period'], y=Fiscal_Year_CTS['Distribution_Costs'], mode='lines'), row=2, col=2)

# update layout of subplots
fig.update_layout(height=800, width=1000, title_text="Fiscal Month VS (Sales, Units, Distribution, Delivery Expense)")

# show the plot
fig.show()
```


Fiscal Month VS (Sales, Units, Distribution, Delivery Expense)
Net Sales
800k
700k
600k
2
4
6
8
10
12
Net Units
30k
25k
20k
2
4
6
8
10
12
trace 0
trace 1
trace 2
trace 3
Delivery Costs
−40k
−50k
−60k
−70k
−80k
2
4
6
8
10
12
Distribution Costs
−30k
−35k
−40k
−45k
−50k
2
4
6
8
10
12

In [11]:

```
# Subplots
fig, axs = plt.subplots(2, 2, figsize=(25, 10))

# set the background color of the figure to black
sns.set_theme(style='darkgrid')

# Delivery as percent of Sales plot
sns.lineplot(x='Fiscal_Period', y='Delivery_Percent_Sales', data=CTS_Master, hue='Fiscal_Year', markers=True, dashes=False,ax =axs[0,0])
axs[0,0].set_title('Delivery_as_percent_of_Sales_12_Periods', size = 15,color = 'black')
axs[0,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,0].set_ylabel('Delivery_%_Sales', fontsize=10,color='black')

# Distribution as percent of Sales plot
sns.lineplot(x='Fiscal_Period', y='Distribution_Percent_Sales', data=CTS_Master,hue='Fiscal_Year', markers=True, dashes=False, color='blue',ax =axs[
axs[0,1].set_title('Distribution_as_percent_of_Sales_12_Periods', size = 15,color = 'black')
axs[0,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,1].set_ylabel('Distribution_%_Sales', fontsize=10,color='black')

# D&D as percent of Sales
sns.lineplot(x='Fiscal_Period', y='D_&_D_percent_of_Sales', data=CTS_Master,hue='Fiscal_Year', markers=True, dashes=False, color='blue',ax =axs[1,0]
axs[1,0].set_title('Total_D_&_D_percent_of_Sales_12_Periods', size = 15,color = 'black')
axs[1,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,0].set_ylabel('D_&_D_percent_of_Sales', fontsize=10,color='black')

# contribution Margin as percent of Sales
sns.lineplot(x='Fiscal_Period', y='Contribution_Percent_Sales', data=CTS_Master,hue='Fiscal_Year', markers=True, dashes=False, color='blue',ax =axs[
axs[1,1].set_title('CM_as percent_of_Sales_12_Periods', size = 15,color = 'black')
axs[1,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,1].set_ylabel('CM_as_percent_of_Sales', fontsize=10,color='black')

# adjust the space between the subplots
fig.tight_layout()

# show the chart
plt.show()
```


Delivery_as_percent_of_Sales_12_Periods
Delivery_%_Sales
−6
−7
−8
−9
−10
−11
−12
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022
Distribution_as_percent_of_Sales_12_Periods
Distribution_%_Sales
−5.0
−5.5
−6.0
−6.5
−7.0
−7.5
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022
Total_D_&_D_percent_of_Sales_12_Periods
D_&_D_percent_of_Sales
−12
−13
−14
−15
−16
−17
−18
−19
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022
CM_as percent_of_Sales_12_Periods
CM_as_percent_of_Sales
12
10
8
6
4
2
0
−2
−4
Time Period
2
4
6
8
10
12
Fiscal_Year
2021
2022

In [12]:

```
# Carton Pick list. Unique Shipment combination is just for contract
Carton_Pick_list = '''Select TimePeriod,
       COUNT(DISTINCT CTN_ID) AS Count_of_Cartons,
       COUNT(DISTINCT CONCAT(ORD_ID, ORD_LINK_NMB, SHPMT_ID)) AS Unique_Shipments
FROM   [COST_TO_SERVE_ARCHIVE].[SC_Cost].[Carton_Pick_List_SC_Costs_Archive]
WHERE  STAT_IND <> '99'
                  AND PICK_CTL_CHAR NOT IN ('#','T')
                  AND PICK_TYPE NOT IN ('DUMMY WRAP AND LABEL', 'RSI', 'DNR')
                  AND YEAR = '2022'
                  AND CUST_ID = '1876074'
GROUP BY TimePeriod'''
 # SKU level DF
Carton_Pick_List =  pd.read_sql(Carton_Pick_list,sql_connection)
# Removing last 4 characters in the string
Carton_Pick_List["TimePeriod"] = Carton_Pick_List["TimePeriod"].str[:-4]
# Converting timeperiod to integer
Carton_Pick_List['TimePeriod'] = Carton_Pick_List['TimePeriod'].astype(int)
#Sort by month
Carton_Pick_List = Carton_Pick_List.sort_values(by="TimePeriod", ascending=True)
# Convert TimePeriod to Fiscal Month(Jan, Feb)
Carton_Pick_List['Month'] = Carton_Pick_List['TimePeriod'].apply(lambda x: calendar.month_abbr[x])
# Unique count for number of SKU's
Carton_Pick_List["Cartons_Per_Shipment"] = Carton_Pick_List["Count_of_Cartons"]/Carton_Pick_List["Unique_Shipments"]
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/1154217232.py:13: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

In [13]:

```
#  Merge CTS to carton pick list
CTS_Merge = Fiscal_Year_CTS.merge(Carton_Pick_List, how = "left", left_on ='Fiscal_Period', right_on = 'TimePeriod')
# Per Carton field Calculations
CTS_Merge['Sales_Per_Carton'] = CTS_Merge['Net_Sales']/CTS_Merge['Count_of_Cartons']
CTS_Merge['Delivery_Per_Carton'] = CTS_Merge['Total_Delivery_Costs']/CTS_Merge['Count_of_Cartons']
CTS_Merge['Distribution_Per_Carton'] = CTS_Merge['Distribution_Costs']/CTS_Merge['Count_of_Cartons']
CTS_Merge['CM_Per_Carton'] = CTS_Merge['Contribution_Margin']/CTS_Merge['Count_of_Cartons']
CTS_Merge['Dist_Variable_Per_Carton'] = CTS_Merge['Distribution_Variable_Expense']/CTS_Merge['Count_of_Cartons']
CTS_Merge['Distribution_Expense_Per_Carton'] = CTS_Merge['Distribution_Expense_Final']/CTS_Merge['Count_of_Cartons']
# Read CTS_Merge
CTS_Merge.head()
```

Out[13]:

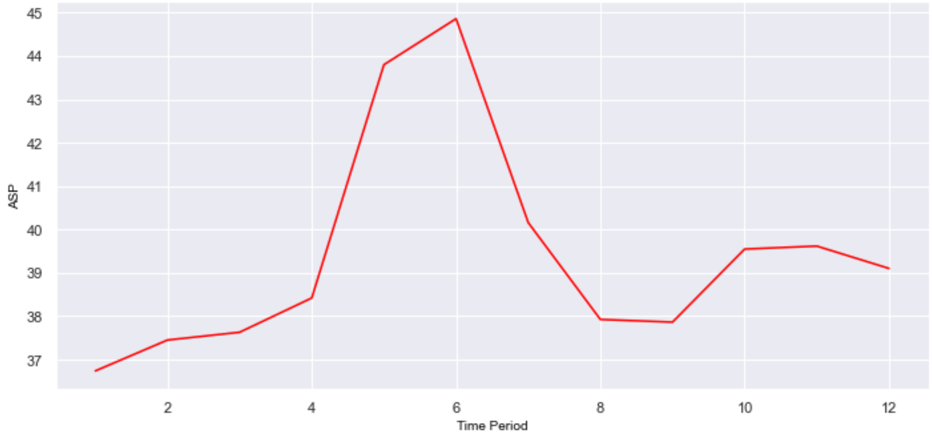
	Fiscal_Year	Fiscal_Period	Net_Sales	COGS	Net_Units	Distribution_Costs	Distribution_Variable_Expense	Distribution_Expense_Final	Total_Delivery_Costs	Contribution_Mar
0	2022	1	605163.3179	-724294.4104	24129.0	-32193.3402	-12019.5223	-20082.2283	-47018.9390	69329.784
1	2022	2	793248.5905	-952156.7233	31184.0	-37835.0541	-15095.1342	-22630.2707	-59370.5170	69760.530
2	2022	3	560014.5415	-668982.2332	21738.0	-30154.5230	-10719.5208	-19355.0222	-39654.9412	35857.158
3	2022	4	667136.5442	-787675.1292	24822.0	-39289.6595	-12342.8858	-26891.3037	-48888.4939	64265.211
4	2022	5	660663.3742	-832536.8145	24767.0	-41501.7146	-11041.4365	-30399.6497	-41174.1728	27880.308

5 rows × 23 columns

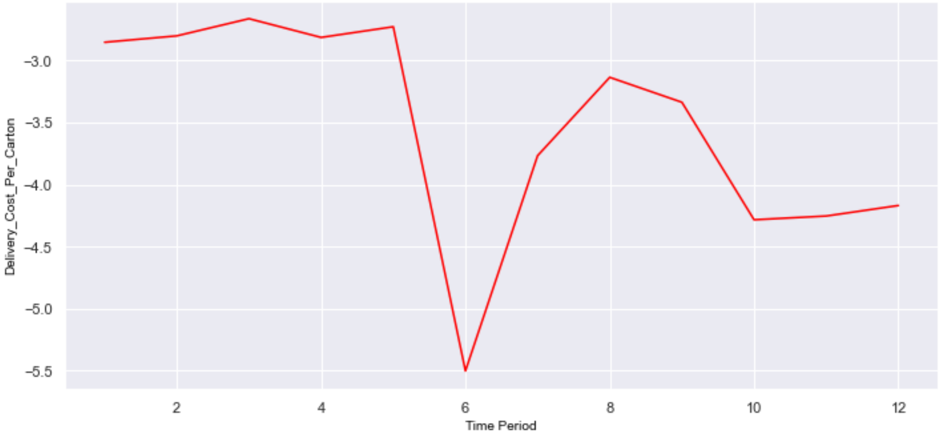
In [14]:

```
# PER Carton Subplots
# Subplots
fig, axs = plt.subplots(2, 2, figsize=(20, 10))

# set the background color of the figure to black
sns.set_theme(style='darkgrid')

# Sales Per Carton
sns.lineplot(x='Fiscal_Period', y='Sales_Per_Carton', data=CTS_Merge, markers=True, dashes=False, color='red', ax =axs[0,0])
axs[0,0].set_title('Sales_Per_Carton_11_Periods', size = 15,color = 'black')
axs[0,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,0].set_ylabel('ASP', fontsize=10,color='black')

# Delivery Per Carton
sns.lineplot(x='Fiscal_Period', y='Delivery_Per_Carton', data=CTS_Merge, markers=True, dashes=False, color='red', ax =axs[0,1])
axs[0,1].set_title('Delivery_Cost_Per_Carton_last_11_Periods', size = 15,color = 'black')
axs[0,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,1].set_ylabel('Delivery_Cost_Per_Carton', fontsize=10,color='black')

# Distribution Per Carton
sns.lineplot(x='Fiscal_Period', y='Distribution_Per_Carton', data=CTS_Merge, markers=True, dashes=False, color='red', ax =axs[1,0])
axs[1,0].set_title('Distribution_Cost_Per_Carton_last_11_Periods', size = 15,color = 'black')
axs[1,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,0].set_ylabel('Distribution_Cost_Per_Carton', fontsize=10,color='black')

# CM Per Carton
sns.lineplot(x='Fiscal_Period', y='CM_Per_Carton', data=CTS_Merge, markers=True, dashes=False, color='red', ax =axs[1,1])
axs[1,1].set_title('Contribution_Per_Carton_11_Periods', size = 15,color = 'black')
axs[1,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,1].set_ylabel('ASP', fontsize=10,color='black')

# Fixed and Variable Distribution Cost
sns.lineplot(x='Fiscal_Period', y='CM_Per_Carton', data=CTS_Merge, markers=True, dashes=False, color='red', ax =axs[1,1])
axs[1,1].set_title('Contribution_Per_Carton_11_Periods', size = 15,color = 'black')
axs[1,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,1].set_ylabel('ASP', fontsize=10,color='black')

# Adjust the space between the subplots
fig.tight_layout()

# show the chart
plt.show()
```

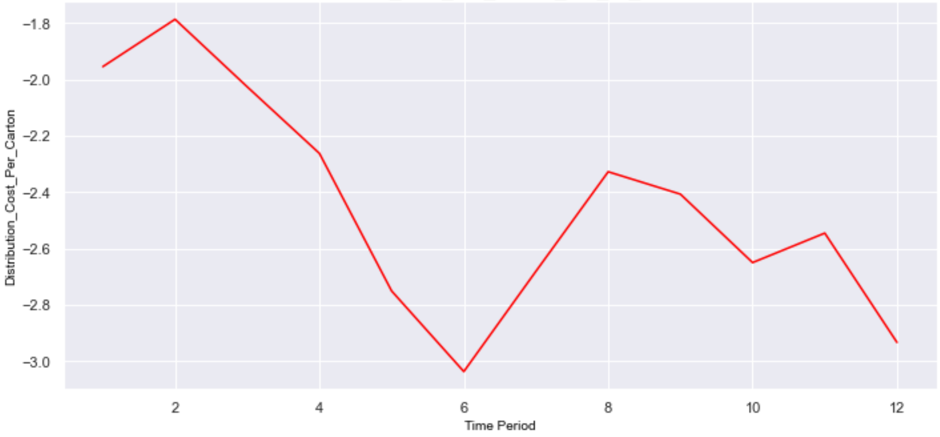
Sales_Per_Carton_11_Periods



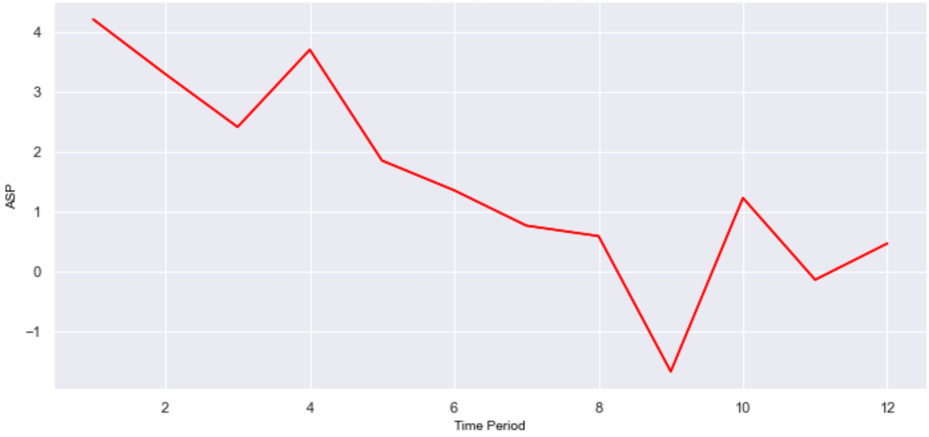
Delivery_Cost_Per_Carton_last_11_Periods



Distribution_Cost_Per_Carton_last_11_Periods

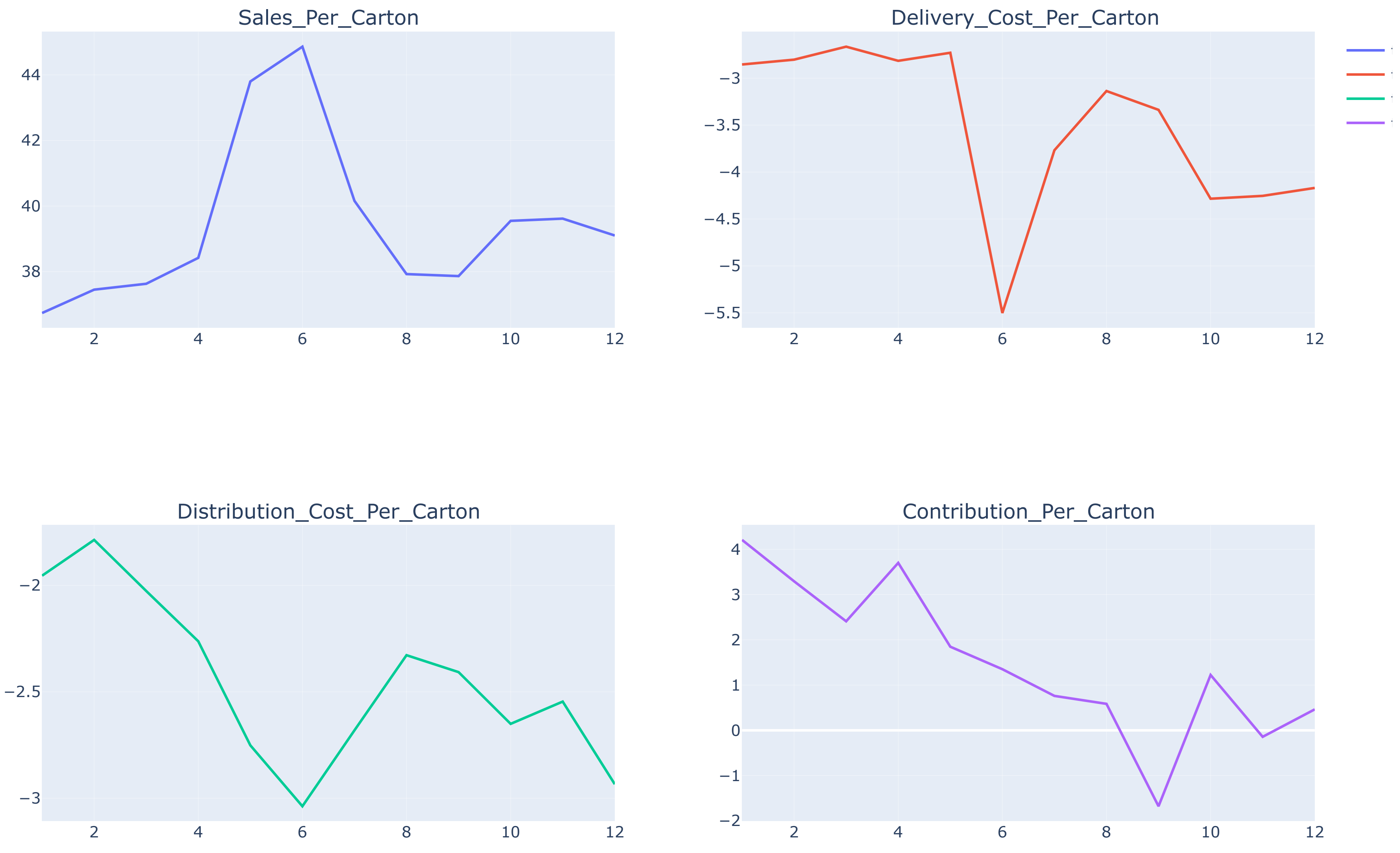


Contribution_Per_Carton_11_Periods



In [15]:

```
# fiscal_year_cts sorting
Fiscal_Year_CTS = Fiscal_Year_CTS.sort_values(by = 'Fiscal_Period',ascending =True)
Fiscal_Year_CTS
# PLOTLY CHARTS
# Converting timeperiod to integer

fig = make_subplots(rows=2, cols=2, start_cell="top-left", subplot_titles=("Sales_Per_Carton", "Delivery_Cost_Per_Carton ", "Distribution_Cost_Per_Ca
#for x, Fiscal_Year in CTS_Master.groupby('Fiscal_Year'):
# add line chart to each subplot
fig.add_trace(go.Scatter(x=CTS_Merge['Fiscal_Period'], y=CTS_Merge['Sales_Per_Carton'], mode='lines'), row=1, col=1)
fig.add_trace(go.Scatter(x=CTS_Merge['Fiscal_Period'], y=CTS_Merge['Delivery_Per_Carton'], mode='lines'), row=1, col=2)
fig.add_trace(go.Scatter(x=CTS_Merge['Fiscal_Period'], y=CTS_Merge['Distribution_Per_Carton'], mode='lines'), row=2, col=1)
fig.add_trace(go.Scatter(x=CTS_Merge['Fiscal_Period'], y=CTS_Merge['CM_Per_Carton'], mode='lines'), row=2, col=2)

# update layout of subplots
fig.update_layout(height=800, width=1200, title_text="Fiscal Month VS Per Carton Metrics")


# show the plot
#fig.show()
```



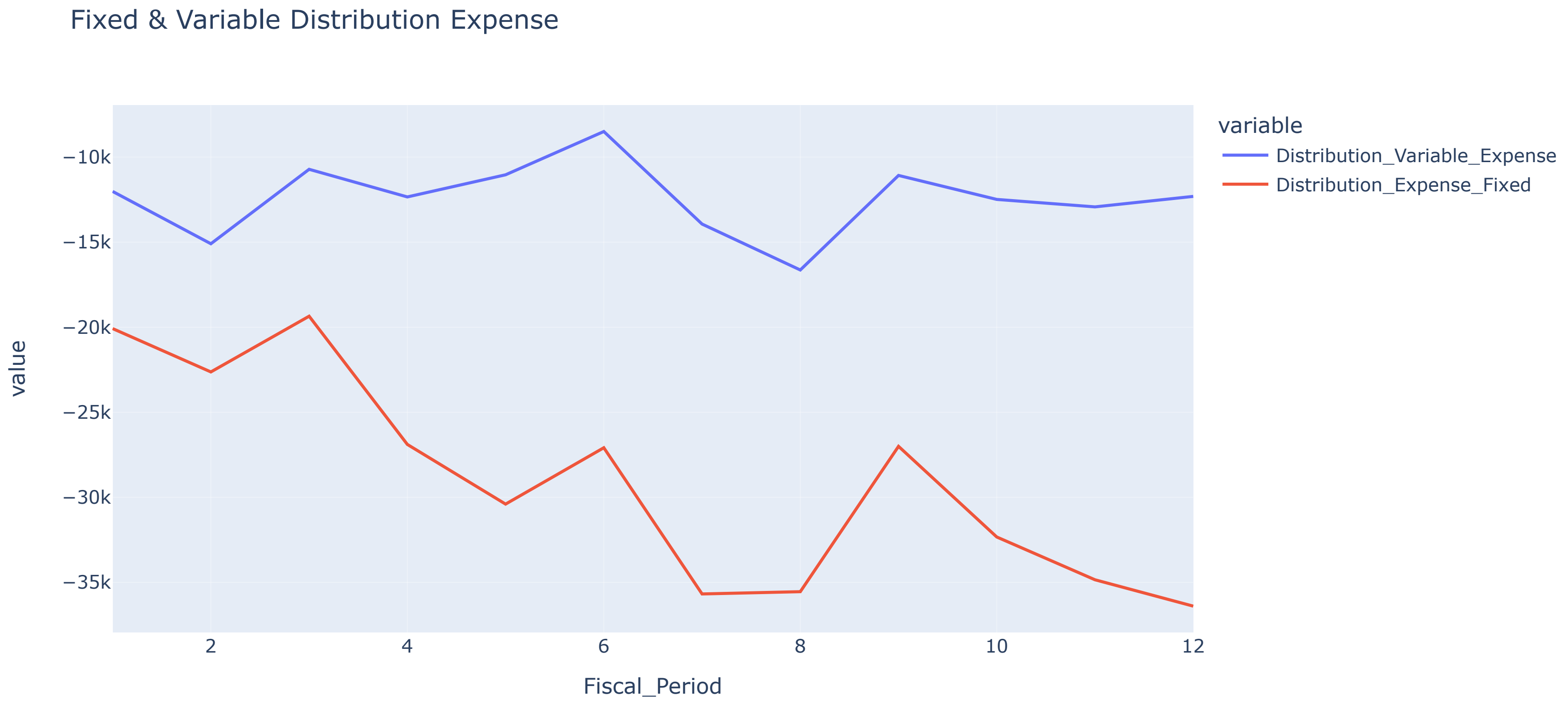
Fiscal Month VS Per Carton Metrics

In [16]:

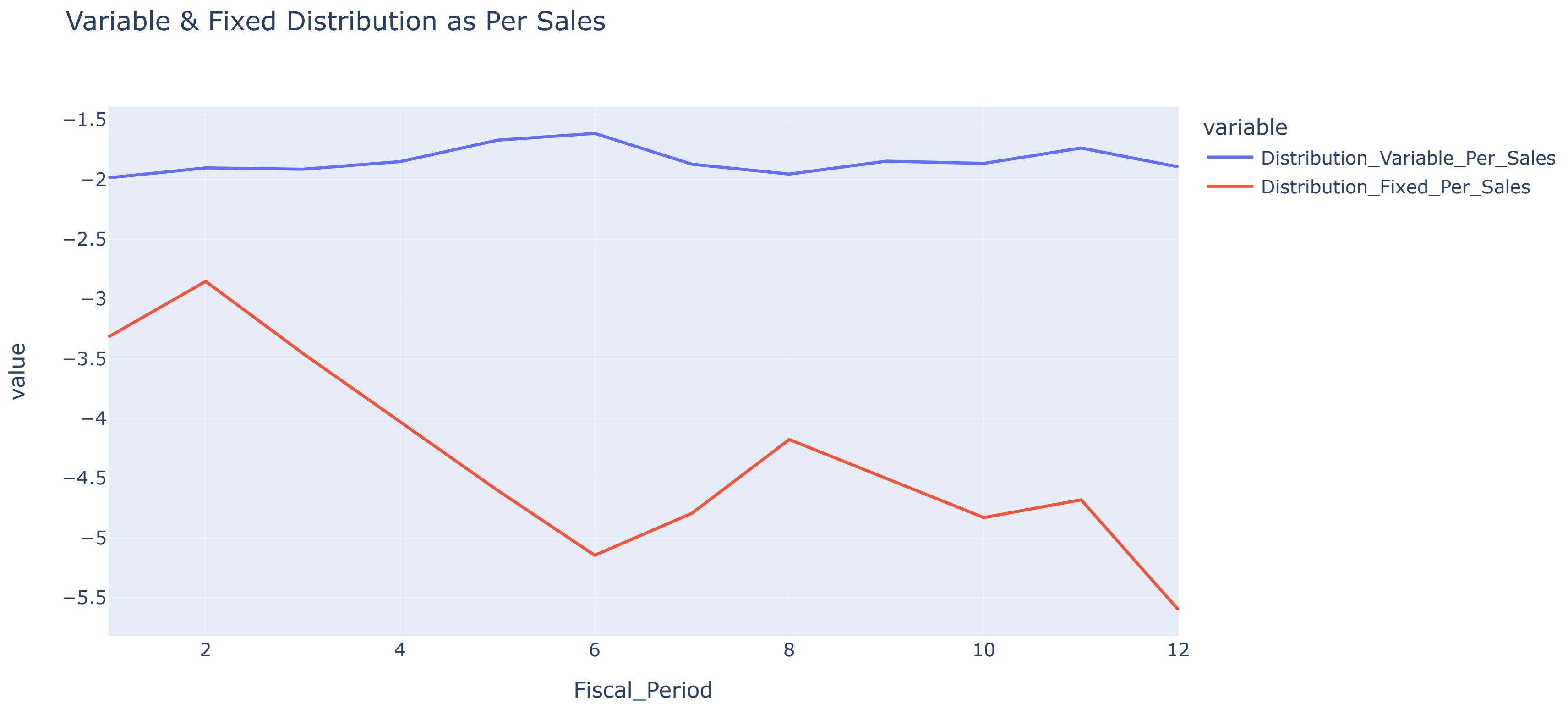
```
# Fixed VS Variable Distribution Expense AS Percent of Sales
Fixed_Variable = Fiscal_Year_CTS.groupby(['Fiscal_Period']).sum()[['Distribution_Variable_Expense','Distribution_Expense_Final']].reset_index()

# Plot
#Fixed_Variable.plot(x='Fiscal_Period', figsize=(15, 6), grid=True)
#Label the x and y axes
#plt.xlabel('Fiscal Period ', fontsize=15)
#plt.ylabel('Percent of Sales', fontsize=15)
# Set the title
#plt.title('Distribution Fixed VS Variable Expense ', fontsize=20)
```

In [18]:

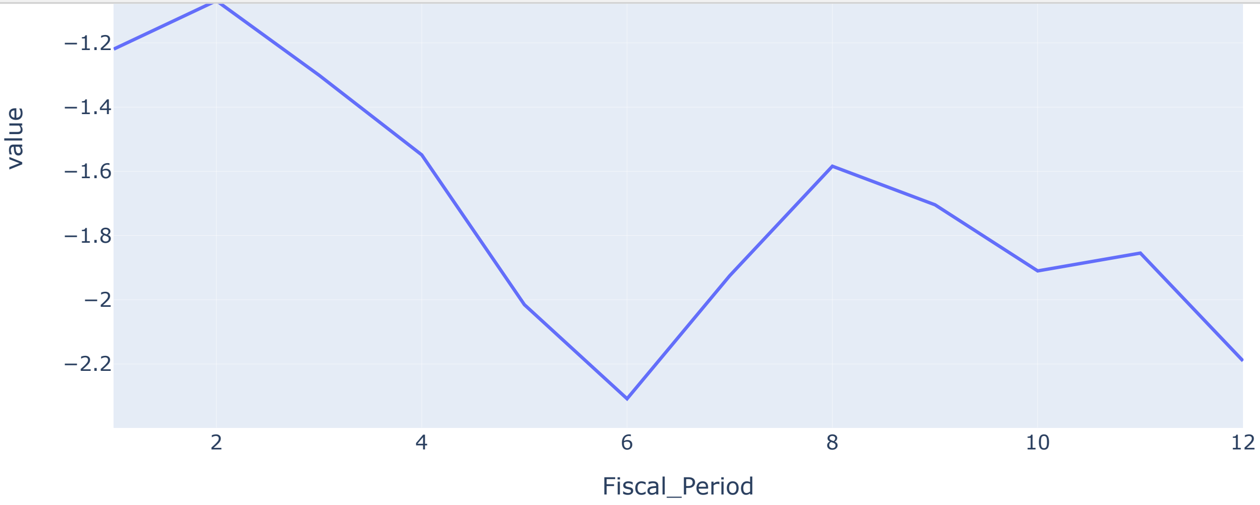
```
Fixed_Variable = Fiscal_Year_CTS.groupby(['Fiscal_Period']).sum()[['Distribution_Variable_Expense','Distribution_Expense_Final']].reset_index()
# Plotly Fixed VS Variable Distribution Expense
Fixed_Variable = Fixed_Variable.rename(columns = {'Distribution_Expense_Final' :'Distribution_Expense_Fixed'})
fig = px.line(Fixed_Variable, x="Fiscal_Period", y=Fixed_Variable.columns,title ="Fixed & Variable Distribution Expense")
fig.update_traces(textposition="bottom right")
fig.show()
```



```
In [19]: # Distirbution Variable and Fixed as Percent of Sales
         Fiscal_Year_CTS['Distribution_Variable_Per_Sales'] = Fiscal_Year_CTS['Distribution_Variable_Expense']/Fiscal_Year_CTS['Net_Sales'] * 100
         Fiscal_Year_CTS['Distribution_Fixed_Per_Sales'] = Fiscal_Year_CTS['Distribution_Expense_Final']/Fiscal_Year_CTS['Net_Sales'] * 100
         # Fixed_Variable Separate DataFrame
         Fixed_Variable_Per_Sales = Fiscal_Year_CTS.groupby(['Fiscal_Period']).sum()[['Distribution_Variable_Per_Sales','Distribution_Fixed_Per_Sales']].rese
         # Plot as Percent of Sales
         fig = px.line(Fixed_Variable_Per_Sales, x="Fiscal_Period", y=Fixed_Variable_Per_Sales.columns, title = 'Variable & Fixed Distribution as Per Sales')
         fig.update_traces(textposition="bottom right")
         fig.show()
```



In [20]:
```
# Merge Carton Pick list with
CTS_Merge['Fixed_Distribution_Per_Carton'] = CTS_Merge['Distribution_Expense_Final']/CTS_Merge['Count_of_Cartons']
CTS_Merge['Variable_Distribution_Per_Carton'] = CTS_Merge['Distribution_Variable_Expense']/CTS_Merge['Count_of_Cartons']
# Fixed_Variable Separate DataFrame
Fixed_Variable_Per_Carton = CTS_Merge.groupby(['Fiscal_Period']).sum()[['Fixed_Distribution_Per_Carton','Variable_Distribution_Per_Carton']].reset_i
# Plot as Percent of Sales
fig = px.line(Fixed_Variable_Per_Carton, x="Fiscal_Period", y=Fixed_Variable_Per_Carton.columns, title = 'Variable & Fixed Distribution as Per Carto
fig.update_traces(textposition="bottom right")
fig.show()
```



In [25]:

```
#SKU PROFILE DEEP DIVE
SKU_Profile = '''Select SKU_NUM,
                        SKU_Description,
                        Vendor_Id,
                        Vendor_Name,
                        Class_Name,
                        SUM(ADJUSTED_NET_SALES_W_COU_AMT_$) AS Net_Sales,
                        SUM(Sales_$_SMS_COS) AS COGS,
                        SUM(Sales_Total_Units_Net) AS Net_Units,
                        SUM(Total_Distribution_Costs) AS Distribution_Costs,
                        SUM(FC_Variable_Handling_Expense_Final) AS Distribution_Variable_Expense,
                        SUM(Fixed_Expense_Final) AS Distribution_Expense_Final,
                        SUM(Total_Delivery_Costs) AS Total_Delivery_Costs,
                        SUM(Contribution_Margin) AS Contribution_Margin
                 From linked.CTS_1P.[CTS_2.0_All_BUs_Data_V]
                 WHERE Master_Customer_Number = '1876074' AND
                       Fiscal_Year = '2022'
                 GROUP BY SKU_NUM,
                        SKU_Description,
                        Vendor_Id,
                        Vendor_Name,
                        Class_Name
                 ORDER BY Net_Sales DESC'''
```

In [26]:

```
# Read the SKU Profile DF
SKU_Profile_01 = pd.read_sql(SKU_Profile,sql_connection)
# Replace 0 or =negative sales SKU's with 0.1
SKU_Profile_01['New_Sales'] = SKU_Profile_01['Net_Sales'].apply(lambda x: 0.1 if x <= 0 else x)
```

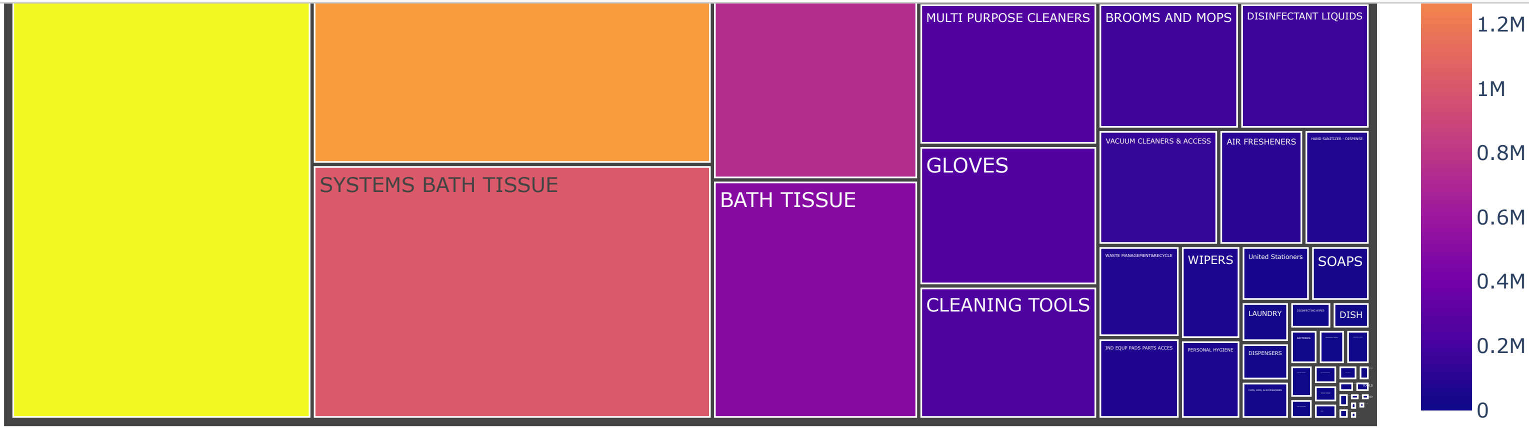
```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/1130100699.py:2: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

In [23]:

```
# Top 10 SKU's, On-hand, Annual Usage and Usage from the customer
TOP_10_SKUs = '''Select * From (Select TOP 10  SKU_NUM,
                        SKU_Description,
                        Vendor_Id,
                        Vendor_Name,
                        Class_Name,
                        SUM(ADJUSTED_NET_SALES_W_COU_AMT_$) AS Net_Sales,
                        SUM(Sales_$_SMS_COS) AS COGS,
                        SUM(Sales_Total_Units_Net) AS Net_Units,
                        SUM(Total_Distribution_Costs) AS Distribution_Costs,
                        SUM(FC_Variable_Handling_Expense_Final) AS Distribution_Variable_Expense,
                        SUM(Fixed_Expense_Final) AS Distribution_Expense_Final,
                        SUM(Total_Delivery_Costs) AS Total_Delivery_Costs,
                        SUM(Contribution_Margin) AS Contribution_Margin
                 From linked.CTS_1P.[CTS_2.0_All_BUs_Data_V]
                 WHERE Master_Customer_Number = '1876074' AND
                       Fiscal_Year = '2022'
                 GROUP BY SKU_NUM,
                        SKU_Description,
                        Vendor_Id,
                        Vendor_Name,
                        Class_Name
                 ORDER BY Net_Sales DESC) A LEFT JOIN
            (SELECT SKU_NUM,
                    sum(Sales_Total_Units_Net) as Annual_Usage
             FROM LINKED.CTS_1P.[CTS_2.0_All_BUs_Data_V]
             WHERE Fiscal_Year = '2022'
             GROUP BY SKU_NUM) B
             ON A.SKU_NUM = B.SKU_NUM
             LEFT JOIN
             (SELECT  SKU_Num,
                      SUM(FC_OH) AS ON_HAND,
                      FC_DIMs_Width,
                      FC_DIMs_Height,
                      FC_DIMs_Length,
                      FC_DIMs_Volume
              FROM linked.Prism.MASTER_DETAIL_HIST_V
              WHERE [YEAR] = 2022
                          AND TimePeriod in ('1_CTS','2_CTS','3_CTS','4_CTS','5_CTS','6_CTS','7_CTS','8_CTS','9_CTS','10_CTS','11_CTS')
              GROUP BY SKU_Num,FC_DIMs_Width,
                      FC_DIMs_Height,
                      FC_DIMs_Length,
                      FC_DIMs_Volume) C
              ON A.SKU_NUM = C.SKU_NUM'''

# Read the Top 10 SKU's File
Top_10_SKUs_ = pd.read_sql(TOP_10_SKUs,sql_connection)
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/640379347.py:47: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

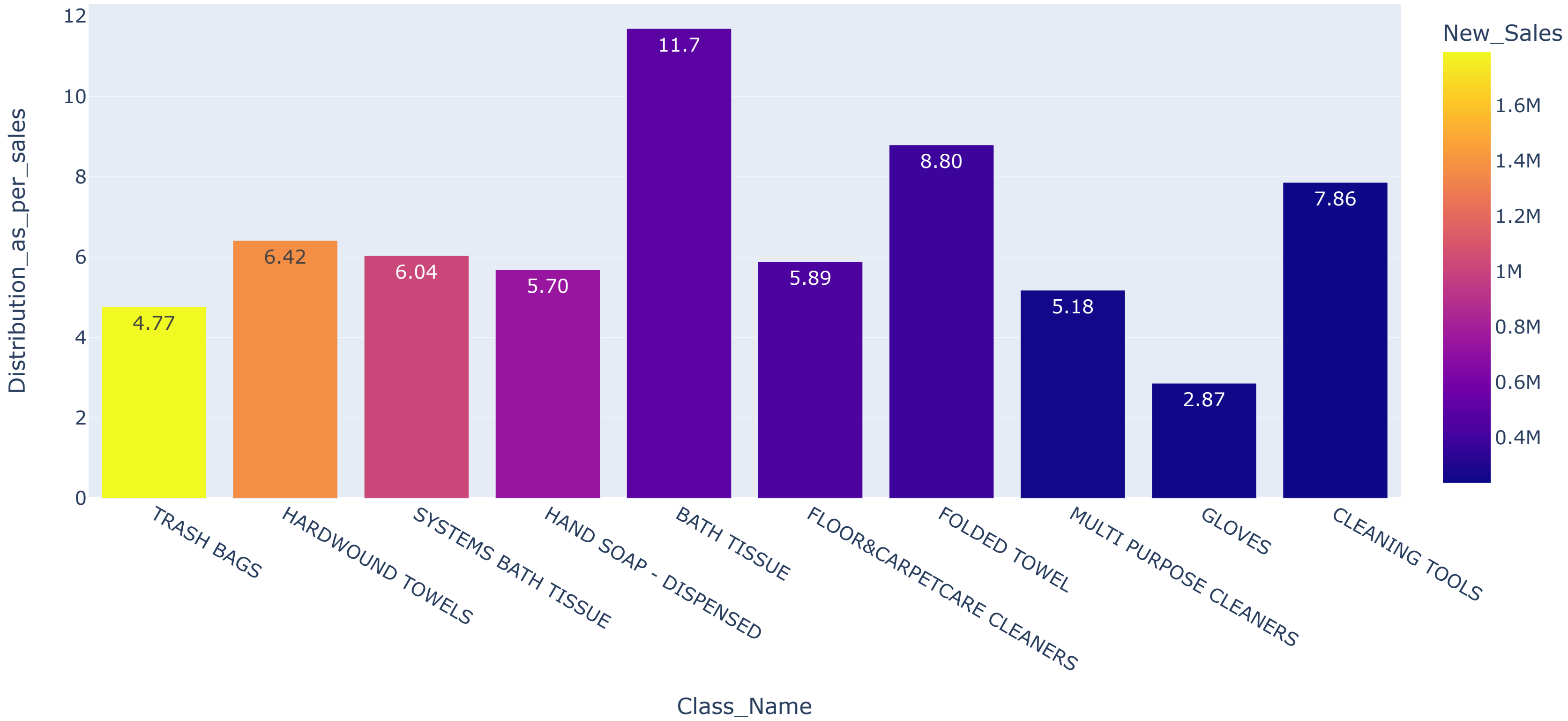
```
In [27]: # Class Level VS  Sales
         Class_Name_DF = SKU_Profile_01.groupby(['Class_Name']).sum()['New_Sales'].reset_index().sort_values(by='New_Sales',ascending=False)
         Class_Name_DF['New_Sales'] = Class_Name_DF['New_Sales'].astype(int)
         #
         # group data by zip code and calculate percentage usage
         Class_Name_DF = Class_Name_DF.groupby('Class_Name').sum().reset_index()
         # Replacing 0 to 0.1 values
         Class_Name_DF['New_Sales'] = Class_Name_DF['New_Sales'].apply(lambda x: 0.1 if x <= 0 else x)
         #Class_Name_DF['usage_percentage'] = (Class_Name_DF['New_Sales'] / total_sales) * 100
         # Class Names
         import plotly.express as px

         fig = px.treemap(Class_Name_DF,
                          path=['Class_Name'],
                          values='New_Sales',
                          color='New_Sales'
                         )
         fig.show()
```



In [28]:

```
# Class Level VS  Distribution Cost
Class_Name_Distribution_Cost = SKU_Profile_01.groupby(['Class_Name']).sum()[['New_Sales','Distribution_Costs']].reset_index().sort_values(by='New_Sa
# Distribution as percent of sales
Class_Name_Distribution_Cost['Distribution_as_per_sales'] = Class_Name_Distribution_Cost['Distribution_Costs']/Class_Name_Distribution_Cost['New_Sal
# Top 10 Classes with highest sales and their distirbution as percent of sales
Class_Name_Distribution_Cost_Top_10_Classes = Class_Name_Distribution_Cost.sort_values(by = 'New_Sales',ascending = False).head(10)
# Convert distribution cost to absolute value
Class_Name_Distribution_Cost_Top_10_Classes['Distribution_as_per_sales'] = Class_Name_Distribution_Cost_Top_10_Classes['Distribution_as_per_sales'].
# group data by zip code and calculate percentage usage
Class_Name_Distribution_Cost = Class_Name_Distribution_Cost.groupby('Class_Name').sum().reset_index()
# Converting Distribution Expense to absolute Expense
Class_Name_Distribution_Cost['Distribution_Costs_01'] = Class_Name_Distribution_Cost['Distribution_Costs'].abs()
# Replacing 0 to 0.1 values
Class_Name_Distribution_Cost['Distribution_Costs_01'] = Class_Name_Distribution_Cost['Distribution_Costs_01'].apply(lambda x: 0.1 if x <= 0 else x)
#Class_Name_DF['usage_percentage'] = (Class_Name_DF['New_Sales'] / total_sales) * 100
# Figue for the top 10 classes  and their distribution as percent of sales
fig = px.bar(Class_Name_Distribution_Cost_Top_10_Classes, x='Class_Name', y='Distribution_as_per_sales', color='New_Sales', text_auto='.3s')
fig.show()
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/3141905640.py:2: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
```



In [29]:

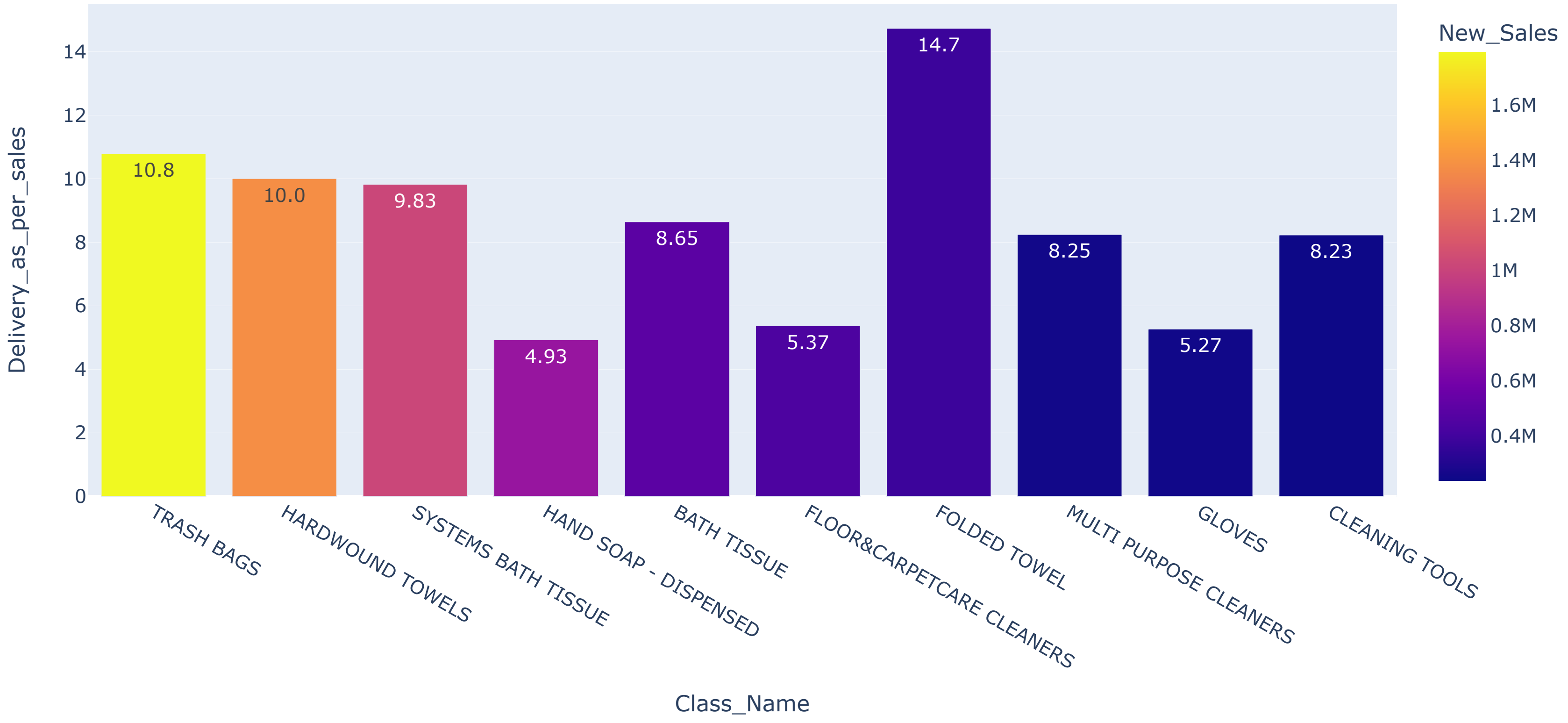
```
# Class Level VS  Distribution Cost
Class_Name_Distribution_Cost = SKU_Profile_01.groupby(['Class_Name']).sum()[['New_Sales','Distribution_Costs']].reset_index().sort_values(by='Distri
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/4084083132.py:2: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
```

In [30]:

```
# Class Level VS  Delivery Cost
Class_Name_Delivery_Cost = SKU_Profile_01.groupby(['Class_Name']).sum()[['New_Sales','Total_Delivery_Costs']].reset_index().sort_values(by='New_Sale
# Distribution as percent of sales
Class_Name_Delivery_Cost['Delivery_as_per_sales'] = Class_Name_Delivery_Cost['Total_Delivery_Costs']/Class_Name_Delivery_Cost['New_Sales'] * 100
# Top 10 Classes with highest sales and their distribution as percent of sales
Class_Name_Delivery_Cost_Top_10_Classes = Class_Name_Delivery_Cost.sort_values(by = 'New_Sales',ascending = False).head(10)
# Convert distribution cost to absolute value
Class_Name_Delivery_Cost_Top_10_Classes['Delivery_as_per_sales'] = Class_Name_Delivery_Cost_Top_10_Classes['Delivery_as_per_sales'].abs()
#Class_Name_DF['Distribution_Costs'] = Class_Name_DF['New_Sales'].astype(int)
# Delivery as percent of sales for the top 10 classes
# group data by zip code and calculate percentage usage
Class_Name_Delivery_Cost = Class_Name_Delivery_Cost.groupby('Class_Name').sum().reset_index()
# Converting Distribution Expense to absolute Expense
Class_Name_Delivery_Cost['Total_Delivery_Costs_01'] = Class_Name_Delivery_Cost['Total_Delivery_Costs'].abs()
# Replacing 0 to 0.1 values
Class_Name_Delivery_Cost['Total_Delivery_Costs_01'] = Class_Name_Delivery_Cost['Total_Delivery_Costs_01'].apply(lambda x: 0.1 if x <= 0 else x)
#Class_Name_DF['usage_percentage'] = (Class_Name_DF['New_Sales'] / total_sales) * 100
# Class Names
import plotly.express as px
# Figue for the top 10 classes  and their distribution as percent of sales
fig = px.bar(Class_Name_Delivery_Cost_Top_10_Classes, x='Class_Name', y='Delivery_as_per_sales', color='New_Sales', text_auto='.3s')
fig.show()
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/1805713919.py:2: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
```



In [31]:

```
# Class Level VS  Delivery Cost
Class_Name_Delivery_Cost = SKU_Profile_01.groupby(['Class_Name']).sum()[['New_Sales','Total_Delivery_Costs']].reset_index().sort_values(by='New_Sale
# Distribution as percent of sales
Class_Name_Delivery_Cost['Delivery_as_per_sales'] = Class_Name_Delivery_Cost['Total_Delivery_Costs']/Class_Name_Delivery_Cost['New_Sales'] * 100
# Top 10 Classes with highest sales and their distribution as percent of sales
Class_Name_Delivery_Cost_Top_10_Classes = Class_Name_Delivery_Cost.sort_values(by = 'New_Sales',ascending = False).head(10)
# Convert distribution cost to absolute value
Class_Name_Delivery_Cost_Top_10_Classes['Delivery_as_per_sales'] = Class_Name_Delivery_Cost_Top_10_Classes['Delivery_as_per_sales'].abs()
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/2623271454.py:2: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
```

In [32]:
```
Class_Name_Delivery_Cost
```

Out[32]:

	Class_Name	New_Sales	Total_Delivery_Costs	Delivery_as_per_sales
49	TRASH BAGS	1.789252e+06	-193075.5988	-10.790854
28	HARDWOUND TOWELS	1.364985e+06	-136610.8956	-10.008233
47	SYSTEMS BATH TISSUE	1.011490e+06	-99392.7874	-9.826370
27	HAND SOAP - DISPENSED	7.353948e+05	-36262.8913	-4.931078
1	BATH TISSUE	4.899109e+05	-42364.7875	-8.647448
20	FLOOR&CARPETCARE CLEANERS	4.359952e+05	-23416.0421	-5.370711
21	FOLDED TOWEL	3.796612e+05	-55942.9938	-14.734978
35	MULTI PURPOSE CLEANERS	2.533665e+05	-20898.5248	-8.248338
24	GLOVES	2.494689e+05	-13151.9283	-5.271971
7	CLEANING TOOLS	2.370206e+05	-19514.1363	-8.233098
6	BROOMS AND MOPS	1.780461e+05	-16884.3556	-9.483136

In [33]:
```
Class_Name_Delivery_Cost_Top_10_Classes
```

Out[33]:

	Class_Name	New_Sales	Total_Delivery_Costs	Delivery_as_per_sales
49	TRASH BAGS	1.789252e+06	-193075.5988	10.790854
28	HARDWOUND TOWELS	1.364985e+06	-136610.8956	10.008233
47	SYSTEMS BATH TISSUE	1.011490e+06	-99392.7874	9.826370
27	HAND SOAP - DISPENSED	7.353948e+05	-36262.8913	4.931078
1	BATH TISSUE	4.899109e+05	-42364.7875	8.647448
20	FLOOR&CARPETCARE CLEANERS	4.359952e+05	-23416.0421	5.370711
21	FOLDED TOWEL	3.796612e+05	-55942.9938	14.734978
35	MULTI PURPOSE CLEANERS	2.533665e+05	-20898.5248	8.248338
24	GLOVES	2.494689e+05	-13151.9283	5.271971
7	CLEANING TOOLS	2.370206e+05	-19514.1363	8.233098

In [34]:
```
elivery Cost
_Cost = SKU_Profile_01.groupby(['Class_Name']).sum()[['New_Sales','Total_Delivery_Costs']].reset_index().sort_values(by='New_Sales',ascending=False)
ercent of sales
_Cost['Delivery_as_per_sales'] = Class_Name_Delivery_Cost['Total_Delivery_Costs']/Class_Name_Distribution_Cost['New_Sales'] * 100
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/957483421.py:2: FutureWarning:

The default value of numeric_only in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will default to False. Either specify nu
meric_only or select only columns which should be valid for the function.
```

In [35]:
```
Class_Name_Delivery_Cost
```

. . .

In [36]:
```
# On-Hand Per Month
Top_10_SKUs_['Monthly_Usage'] =  Top_10_SKUs_['Annual_Usage']/12
Top_10_SKUs_['Avg_On_Hand'] = Top_10_SKUs_['ON_HAND']/12
Top_10_SKUs_['Turns'] = Top_10_SKUs_['Annual_Usage']/Top_10_SKUs_['Avg_On_Hand']
Top_10_SKUs_['MOS'] = Top_10_SKUs_['ON_HAND']/Top_10_SKUs_['Annual_Usage']
# Calculating Months of supply based on Turns
#Top_10_SKUs_['MOS'] = 12/Top_10_SKUs_['ON_HAND']
```

In [37]:
```
# Sorting by Sales and analysing Months of Supply,Turn Over and On-Hand
Top_10_SKUs_.sort_values(by = 'Net_Sales',ascending = False)
```

stribution_Costs	Distribution_Variable_Expense	...	SKU_Num	ON_HAND	FC_DIMs_Width	FC_DIMs_Height	FC_DIMs_Length	FC_DIMs_Volume	Monthly_Usage	Avg_On_Hand	Turns	MOS
-43994.4187	-11886.0994	...	498871.0	152238.0	17.1	8.60	22.70	3338.262	22087.750000	12686.500000	20.892524	0.574368
-24771.6374	-5799.3271	...	364374.0	66749.0	17.9	11.20	18.00	3608.640	9139.083333	5562.416667	19.716071	0.608641
-8363.3274	-1705.1189	...	812927.0	21449.0	14.5	11.50	19.50	3251.625	5347.500000	1787.416667	35.900974	0.334253
-8363.3274	-1705.1189	...	812927.0	7030.0	14.8	11.80	20.10	3510.264	5347.500000	585.833333	109.536273	0.109553
-5997.3592	-1514.4829	...	812833.0	66163.0	15.3	9.10	22.50	3132.675	11195.083333	5513.583333	24.365461	0.492500

In [38]:

```
# Top 10 SKU's
Top_10_sku_description = SKU_Profile_01.groupby(['SKU_NUM','SKU_Description','Vendor_Name', 'Vendor_Id'])['Net_Sales','Net_Units','Distribution_Cost
#
Top_10_sku_description['Distribution_as_per_of_Sales'] = Top_10_sku_description['Distribution_Costs']/Top_10_sku_description['Net_Sales']* 100
Top_10_sku_description
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/466464763.py:2: FutureWarning:

Indexing with multiple keys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
```

Out[38]:

	SKU_NUM	SKU_Description	Vendor_Name	Vendor_Id	Net_Sales	Net_Units	Distribution_Costs	Distribution_as_per_of_Sales
0	498871.0	TOWEL NON-PERF 800 RL NL	GA PACIFIC COMMERCIAL BUS	94611	453287.6104	19315.0	-43994.4187	-9.705630
1	364374.0	TISSUE TOILET JUMBO ROLL WE	KIMBERLY CLARK CORP	99581	367825.6872	11234.0	-24771.6374	-6.734613
2	812927.0	TISSUE BATHROOM 2-PLY PREMIUM	GA PACIFIC COMMERCIAL BUS	94611	233857.1645	6252.0	-8363.3274	-3.576255
3	812833.0	TOWEL ROLL ENMOTION FOR RECES	GA PACIFIC COMMERCIAL BUS	94611	178133.9085	3546.0	-5997.3592	-3.366770
4	647204.0	ENMOTION PAPER TOWELS	GA PACIFIC COMMERCIAL BUS	94611	128985.7934	2635.0	-6944.4864	-5.383916
5	744209.0	FLEX LOTION SOAP 1300ML	RUBBERMAID COMMERCIAL PRODUCTS	187151	128661.9745	5780.0	-11900.0517	-9.249082
6	915133.0	HIGH SPEED FLOOR FINISH 5GL	DIVERSEY, INC.	98961	128004.2443	2097.0	-7870.3741	-6.148526
7	394139.0	LINERS 38X58 1.5MIL REPRO	HERITAGE	126961	117373.8740	3664.0	-4188.3523	-3.568385

In [39]:

```
# Pick Type
Pick_Type = '''SELECT TIMEPERIOD,
       PICK_TYPE,
       COUNT(DISTINCT CTN_ID) AS Total_Cartons
FROM [COST_TO_SERVE_ARCHIVE].[SC_Cost].[Carton_Pick_List_SC_Costs_Archive]
WHERE     STAT_IND <> '99'
      AND PICK_CTL_CHAR NOT IN ('#','T')
      AND PICK_TYPE NOT IN ('DUMMY WRAP AND LABEL', 'RSI', 'DNR')
      AND YEAR = '2022'
      AND CUST_ID = '1876074'
GROUP BY TIMEPERIOD,
         PICK_TYPE'''
 #Pick Type
Pick_Type =  pd.read_sql(Pick_Type,sql_connection)
# Removing last 4 characters in the string
Pick_Type["TimePeriod"] = Pick_Type["TIMEPERIOD"].str[:-4]
# Converting timeperiod to integer
Pick_Type['TimePeriod'] = Pick_Type['TimePeriod'].astype(int)
#Sort by month
Pick_Type = Pick_Type.sort_values(by="TimePeriod", ascending=True)
# Convert TimePeriod to Fiscal Month(Jan, Feb)
Pick_Type['Month'] = Pick_Type['TimePeriod'].apply(lambda x: calendar.month_abbr[x])
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/1783254854.py:14: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

In [40]:

```
Order_Frequency = '''Select TimePeriod,
      Count(DISTINCT pick_crte_dt) AS Frequency
From [COST_TO_SERVE_ARCHIVE].[SC_Cost].[Carton_Pick_List_SC_Costs_Archive]
WHERE STAT_IND <> '99'
                  AND PICK_CTL_CHAR NOT IN ('#','T')
                  AND PICK_TYPE NOT IN ('DUMMY WRAP AND LABEL', 'RSI', 'DNR')
                  AND YEAR = '2022'
                  AND CUST_ID = '1876074'
GROUP BY TimePeriod'''
 # Freqeuncy of Order DF
Order_Frequency = pd.read_sql(Order_Frequency,sql_connection)
# Removing last 4 characters in the string
Order_Frequency["TimePeriod"] = Order_Frequency["TimePeriod"].str[:-4]
# Converting timeperiod to integer
Order_Frequency["TimePeriod"] = Order_Frequency["TimePeriod"].astype(int)
#Sort by month
Order_Frequency = Order_Frequency.sort_values(by="TimePeriod", ascending=True)
# Convert TimePeriod to Fiscal Month(Jan, Feb)
Order_Frequency['Month'] = Order_Frequency['TimePeriod'].apply(lambda x: calendar.month_abbr[x])
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/3138962890.py:11: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

In [41]:

```
# Prime-Switch %
Prime_Switch = '''Select Fiscal_Period, Prime_FC,
                         Pick_FC,
                         CASE WHEN PRIME_FC = Pick_FC then 'Prime' ELSE 'Switch' END AS FLAG,
                         SUM(Sales_Total_Units_Net) AS units
                  FROM  linked.CTS_1P.[CTS_2.0_All_BUs_Data_V]
                  WHERE Master_Customer_Number = '1876074'
                  AND Fiscal_Year = '2022'
                  GROUP BY Fiscal_Period, Prime_FC,
                           Pick_FC'''
 # Prime Switch DF
Prime_Switch_01 =  pd.read_sql(Prime_Switch,sql_connection)
Prime_Switch_ = Prime_Switch_01.groupby(['Fiscal_Period','FLAG'],group_keys=False).sum()['units'].reset_index()
Prime_Switch_['percent_of_total'] = Prime_Switch_.groupby(['FLAG'],group_keys=False)['units'].apply(lambda x: x / x.sum())*100
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/2074098924.py:12: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

In [42]:

```
Prime_Switch_ = Prime_Switch_01.groupby(['Fiscal_Period','FLAG'],group_keys=False).sum()['units'].reset_index()
Prime_Switch_['percent_of_total'] = Prime_Switch_.groupby(['FLAG'],group_keys=False)['units'].apply(lambda x: x / x.sum())*100
Prime_Switch_.head()
```

Out[42]:

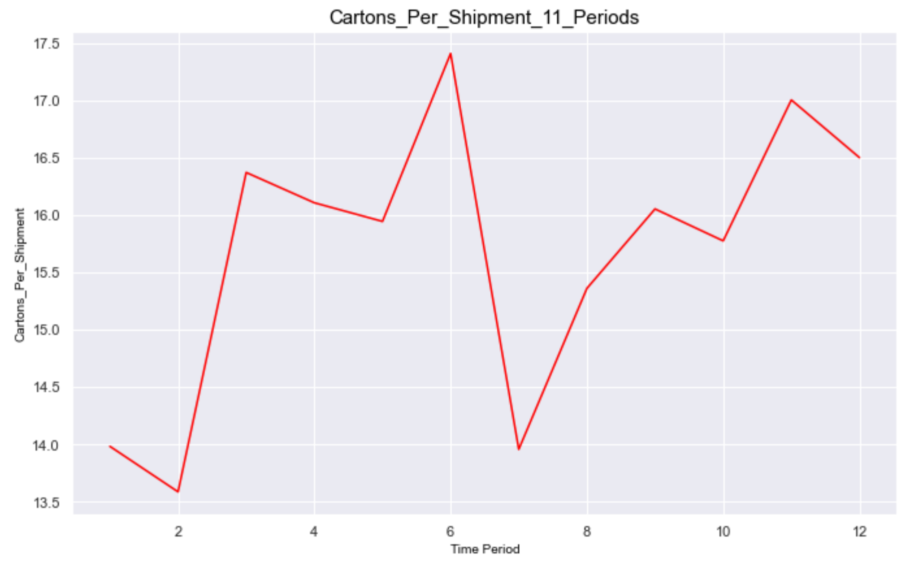
	Fiscal_Period	FLAG	units	percent_of_total
0	1	Prime	22161.0	7.615150
1	1	Switch	1968.0	16.168255
2	2	Prime	29178.0	10.026391
3	2	Switch	2006.0	16.480447
4	3	Prime	20963.0	7.203483

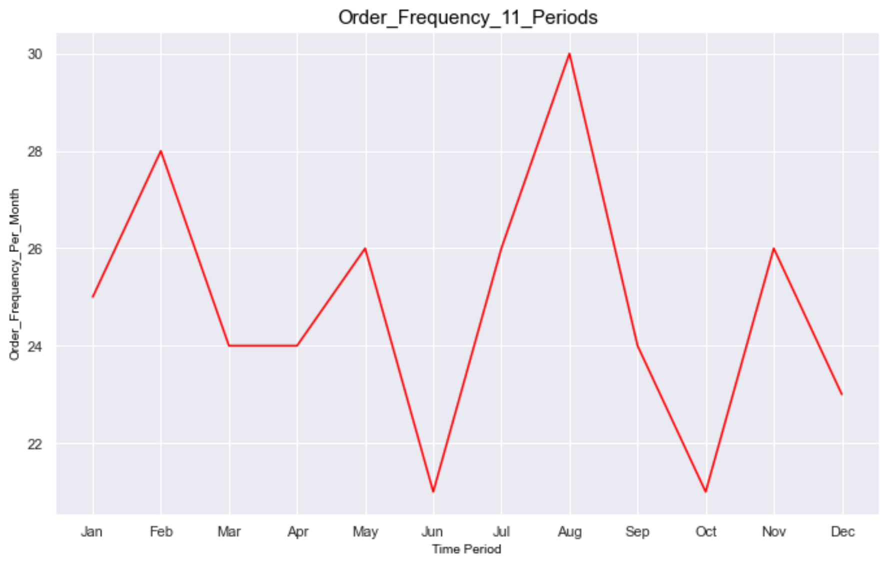
In [43]:

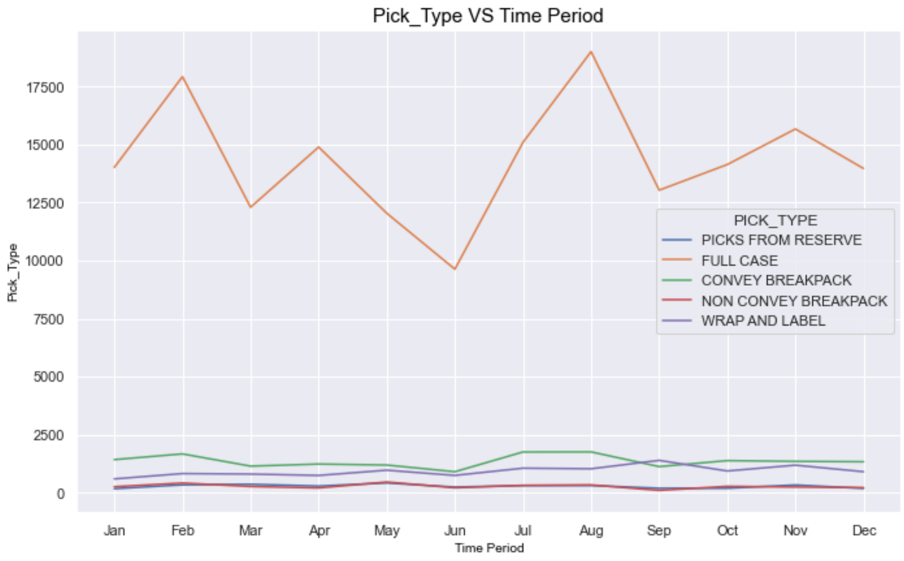
```
# Delivery Expense Deep Diving

# PER Carton Subplots
# Subplots
fig, axs = plt.subplots(2, 2, figsize=(25, 15))
# set the background color of the figure to black
sns.set_theme(style='darkgrid')

# Cartons Per Shipment
sns.lineplot(x='TimePeriod', y='Cartons_Per_Shipment', data=Carton_Pick_List, markers='o', dashes=True, color='red', ax =axs[0,0])
axs[0,0].set_title('Cartons_Per_Shipment_11_Periods', size = 15,color = 'black')
axs[0,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,0].set_ylabel('Cartons_Per_Shipment', fontsize=10,color='black')

# Order Frequency
sns.lineplot(x='Month', y='Frequency', data=Order_Frequency, markers=True, dashes=True, color='red', ax =axs[0,1])
axs[0,1].set_title('Order_Frequency_11_Periods', size = 15,color = 'black')
axs[0,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[0,1].set_ylabel('Order_Frequency_Per_Month', fontsize=10,color='black')

#Pick Type
sns.lineplot(x='Month', y='Total_Cartons', data=Pick_Type, color='red', hue = 'PICK_TYPE', ax =axs[1,0])
axs[1,0].set_title('Pick_Type VS Time Period', size = 15,color = 'black')
axs[1,0].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,0].set_ylabel('Pick_Type', fontsize=10,color='black')

# Prime Switch
sns.lineplot(x='Fiscal_Period', y='units', data=Prime_Switch_, color='red', hue = 'FLAG', ax =axs[1,1])
axs[1,1].set_title('Time Period VS Prime_Switch Units ', size = 15,color = 'black')
axs[1,1].set_xlabel('Time Period', fontsize=10,color='black')
axs[1,1].set_ylabel('Prime_Switch_Units', fontsize=10,color='black')
```

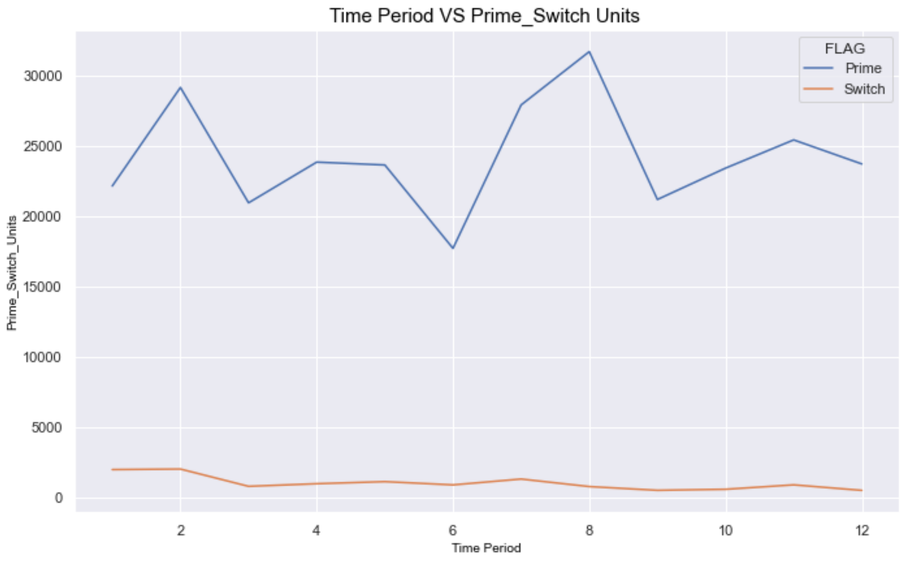
Out[43]:

```
Text(0, 0.5, 'Prime_Switch_Units')
```


Cartons_Per_Shipment_11_Periods
Cartons_Per_Shipment
17.5
17.0
16.5
16.0
15.5
15.0
14.5
14.0
13.5
2
4
6
8
10
12
Time Period


Order_Frequency_11_Periods
Order_Frequency_Per_Month
30
28
26
24
22
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Time Period


Pick_Type VS Time Period
Pick_Type
17500
15000
12500
10000
7500
5000
2500
0
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Time Period
PICK_TYPE
PICKS FROM RESERVE
FULL CASE
CONVEY BREAKPACK
NON CONVEY BREAKPACK
WRAP AND LABEL


Time Period VS Prime_Switch Units
Prime_Switch_Units
30000
25000
20000
15000
10000
5000
0
2
4
6
8
10
12
Time Period
FLAG
Prime
Switch

In [44]:

```
#CTS Connecting
Query_001 =  '''Select shp_to_nmb,
       COUNT(DISTINCT ctn_id) AS Total_Cartons,
       COUNT(DISTINCT CONCAT(ORD_ID, ORD_LINK_NMB, SHPMT_ID)) AS Unique_Shipments,
       Count(DISTINCT pick_crte_dt) AS Frequency
From [COST_TO_SERVE_ARCHIVE].[SC_Cost].[Carton_Pick_List_SC_Costs_Archive]
WHERE STAT_IND <> '99'
                  AND PICK_CTL_CHAR NOT IN ('#','T')
                  AND PICK_TYPE NOT IN ('DUMMY WRAP AND LABEL', 'RSI', 'DNR')
                  AND YEAR = '2022'
                  AND CUST_ID = '1876074'
GROUP BY shp_to_nmb'''
# Reading Carton Pick List DF
Cartons =  pd.read_sql(Query_001,sql_connection)
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/3844101273.py:14: UserWarning:

pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not t
ested. Please consider using SQLAlchemy.
```

In [45]:

```
# Monthly Frequency
Cartons['Monthly_Frequency'] = Cartons['Frequency']/12
# Weekly Frequency
Cartons['Weekly_Frequency'] = Cartons['Frequency']/52
# Zipcode - Cartons, Usage
Ship_to_locations = Cartons.sort_values(by= 'Total_Cartons',ascending = False)
# calculating Cartons Per Shipment
Ship_to_locations['Cartons_PeR_Shipment'] = Ship_to_locations['Total_Cartons']/Ship_to_locations['Unique_Shipments']
# calculate total usage units
total_cartons = Ship_to_locations['Total_Cartons'].sum()
# group data by zip code and calculate percentage usage
Ship_to_locations_001 = Ship_to_locations.groupby('shp_to_nmb').sum().reset_index()
 # calculate total usage units
total_sales = Ship_to_locations['Total_Cartons'].sum()
Ship_to_locations_001['usage_percentage'] = (Ship_to_locations['Total_Cartons'] / total_cartons) * 100
# Cumualtive sum of Percentage of usage column
Ship_to_locations_001['Cumulative_Per_Units'] = Ship_to_locations_001['Total_Cartons'].cumsum()
# print the result
Ship_to_locations_001 = Ship_to_locations_001.sort_values(by= 'usage_percentage',ascending = False)
```

```
In [46]: Ship_to_locations_001 = Ship_to_locations_001.sort_values(by = 'Total_Cartons',ascending = False)
         Ship_to_locations_001
```

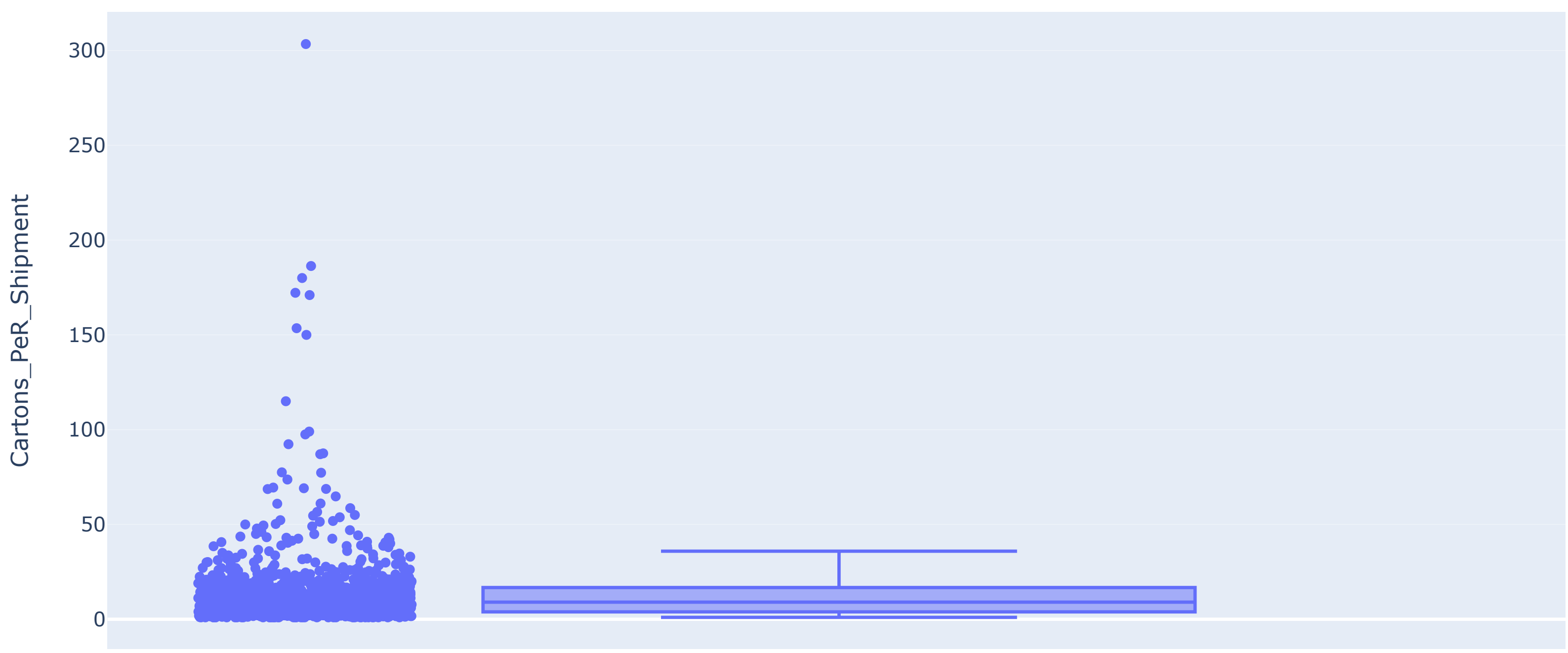
Out[46]:

	shp_to_nmb	Total_Cartons	Unique_Shipments	Frequency	Monthly_Frequency	Weekly_Frequency	Cartons_PeR_Shipment	usage_percentage	Cumulative_Per_Units
796	0199712126	5975	87	61	5.083333	1.173077	68.678161	2.902275	166501
291	0196437678	5765	19	15	1.250000	0.288462	303.421053	2.800270	75953
371	0196523112	4994	29	22	1.833333	0.423077	172.206897	2.425767	99125
355	0196500198	3570	41	35	2.916667	0.673077	87.073171	1.734079	91450
814	0199882558	2610	17	14	1.166667	0.269231	153.529412	1.267772	173291
...	...	...	...	...	...	...	...	...	...
1254	0205733891	1	1	1	0.083333	0.019231	1.000000	0.000486	205770
1253	0205726534	1	1	1	0.083333	0.019231	1.000000	0.000486	205769
1191	0204416150	1	1	1	0.083333	0.019231	1.000000	0.000486	204665
867	0200265053	1	1	1	0.083333	0.019231	1.000000	0.000486	179694
885	0200588285	1	1	1	0.083333	0.019231	1.000000	0.000486	181658

1268 rows × 9 columns

In [47]:

```
fig = px.box(Ship_to_locations_001, y="Cartons_PeR_Shipment", points="all")
fig.show()
```

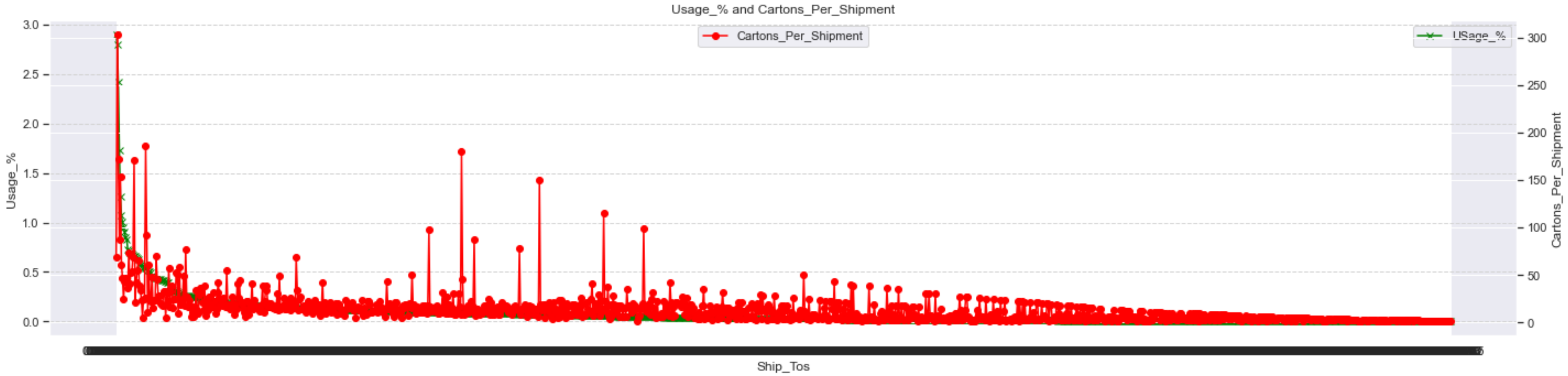


In [48]:

```
# Create the bar chart
fig = px.bar(Ship_to_locations_001, x='shp_to_nmb', y='usage_percentage', text='usage_percentage',text_auto='.3s',color='usage_percentage')
# Show the chart
#fig.show()
```

In [49]:

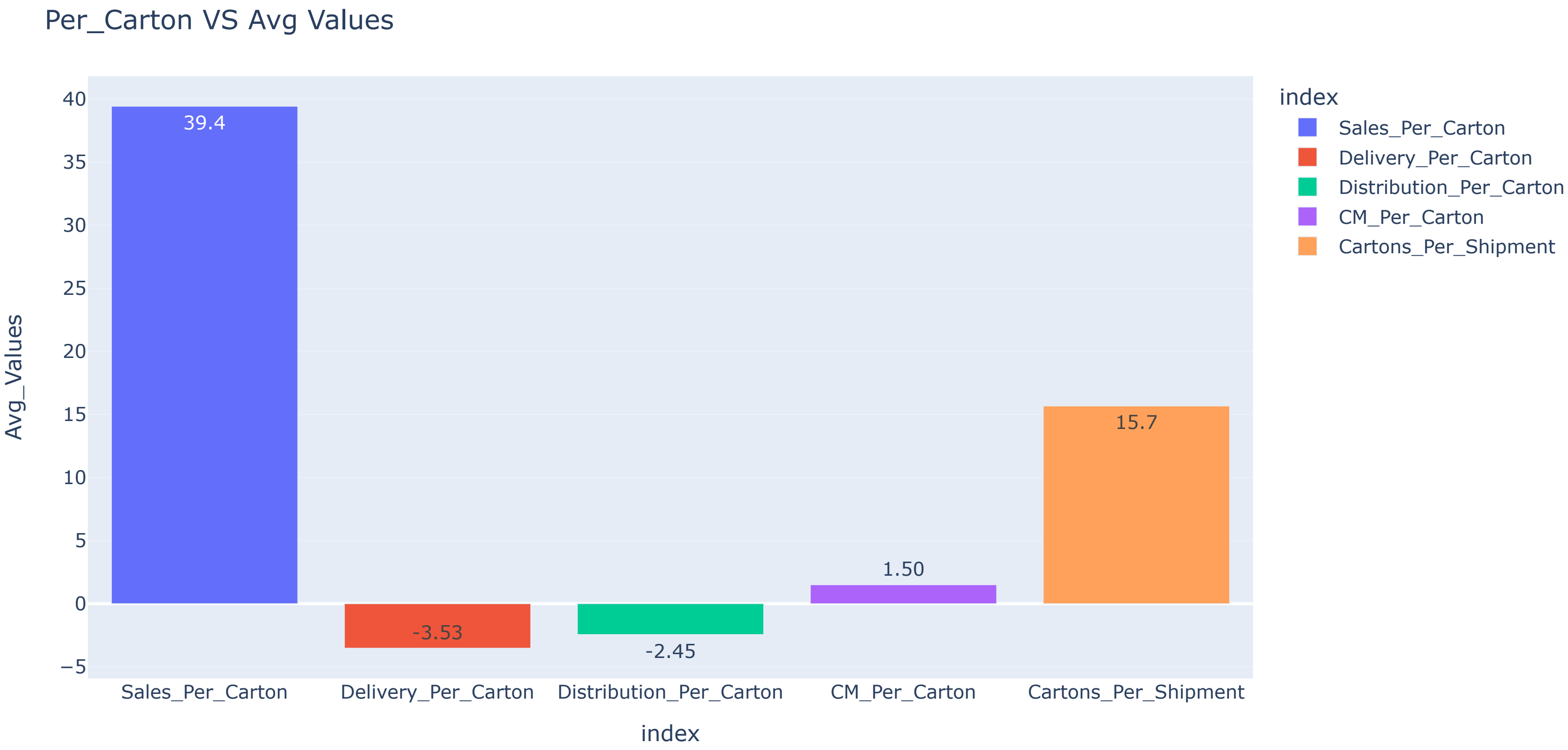
```
# Top Zipcode Analysis
Ship_To_Locations_Top =  Ship_to_locations_001.head(30)
Ship_To_Locations_Top = Ship_to_locations_001.sort_values(by = 'Total_Cartons', ascending = False)
fig, ax = plt.subplots(figsize=(20,5))
ax2 = ax.twinx()
ax.set_title('Usage_% and Cartons_Per_Shipment')
ax.set_xlabel('Ship_Tos')
ax.plot(Ship_To_Locations_Top['shp_to_nmb'], Ship_To_Locations_Top['usage_percentage'], color='green', marker='x')
ax2.plot(Ship_To_Locations_Top['shp_to_nmb'], Ship_To_Locations_Top['Cartons_PeR_Shipment'], color='red', marker='o')
ax.set_ylabel('Usage_%')
ax2.set_ylabel('Cartons_Per_Shipment')
ax.legend(['USage_%'])
ax2.legend(['Cartons_Per_Shipment'], loc='upper center')
#ax.set_xticks(gdp['date'].dt.date)
#ax.set_xticklabels(gdp['date'].dt.year, rotation=90)
ax.yaxis.grid(color='lightgray', linestyle='dashed')
plt.tight_layout()
#plt.show()
```

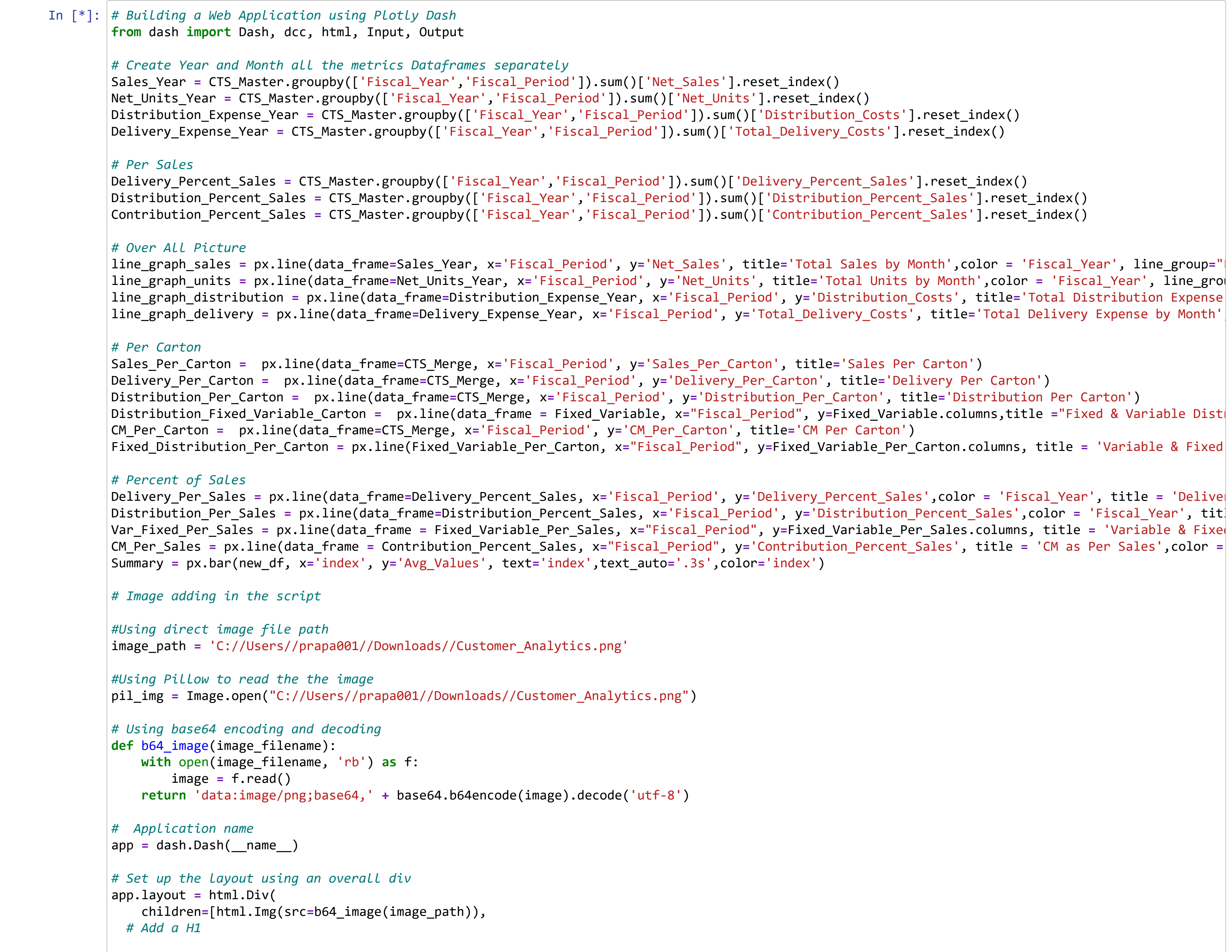


In [51]:

```
#Avg Values
Avg_values = CTS_Merge.mean()
Avg_values_carton_level  = Avg_values[["Sales_Per_Carton","Delivery_Per_Carton","Distribution_Per_Carton","CM_Per_Carton","Cartons_Per_Shipment"]]
new_df = pd.DataFrame(Avg_values_carton_level, columns=['Avg_Values'])
new_df = new_df.reset_index()
# Create the bar chart
fig = px.bar(new_df, x='index', y='Avg_Values', text='index',text_auto='.3s',color='index')
# Add labels inside the bars
for p in ax.containers:
                ax.annotate(format(p.get_height(), '.2f'),
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center',
                xytext = (0, 9),
                textcoords = 'offset points')
# Set the title of the chart
fig.update_layout(title_text='Per_Carton VS Avg Values')
# Show the chart
fig.show()
```

```
C:\Users\prapa001\AppData\Local\Temp\1/ipykernel_32820/1580614603.py:2: FutureWarning:

The default value of numeric_only in DataFrame.mean is deprecated. In a future version, it will default to False. In addition, specifying 'numeric_
only=None' is deprecated. Select only valid columns or specify the value of numeric_only to silence this warning.
```


Per_Carton VS Avg Values
Avg_Values
40
35
30
25
20
15
10
5
0
−5
39.4
-3.53
-2.45
1.50
15.7
Sales_Per_Carton
Delivery_Per_Carton
Distribution_Per_Carton
CM_Per_Carton
Cartons_Per_Shipment
index
index
Sales_Per_Carton
Delivery_Per_Carton
Distribution_Per_Carton
CM_Per_Carton
Cartons_Per_Shipment



```
In [*]: # Building a Web Application using Plotly Dash
        from dash import Dash, dcc, html, Input, Output

        # Create Year and Month all the metrics Dataframes separately
        Sales_Year = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Net_Sales'].reset_index()
        Net_Units_Year = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Net_Units'].reset_index()
        Distribution_Expense_Year = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Distribution_Costs'].reset_index()
        Delivery_Expense_Year = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Total_Delivery_Costs'].reset_index()

        # Per Sales
        Delivery_Percent_Sales = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Delivery_Percent_Sales'].reset_index()
        Distribution_Percent_Sales = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Distribution_Percent_Sales'].reset_index()
        Contribution_Percent_Sales = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()['Contribution_Percent_Sales'].reset_index()

        # Over All Picture
        line_graph_sales = px.line(data_frame=Sales_Year, x='Fiscal_Period', y='Net_Sales', title='Total Sales by Month',color = 'Fiscal_Year', line_group="
        line_graph_units = px.line(data_frame=Net_Units_Year, x='Fiscal_Period', y='Net_Units', title='Total Units by Month',color = 'Fiscal_Year', line_gro
        line_graph_distribution = px.line(data_frame=Distribution_Expense_Year, x='Fiscal_Period', y='Distribution_Costs', title='Total Distribution Expense
        line_graph_delivery = px.line(data_frame=Delivery_Expense_Year, x='Fiscal_Period', y='Total_Delivery_Costs', title='Total Delivery Expense by Month'

        # Per Carton
        Sales_Per_Carton =  px.line(data_frame=CTS_Merge, x='Fiscal_Period', y='Sales_Per_Carton', title='Sales Per Carton')
        Delivery_Per_Carton =  px.line(data_frame=CTS_Merge, x='Fiscal_Period', y='Delivery_Per_Carton', title='Delivery Per Carton')
        Distribution_Per_Carton =  px.line(data_frame=CTS_Merge, x='Fiscal_Period', y='Distribution_Per_Carton', title='Distribution Per Carton')
        Distribution_Fixed_Variable_Carton =  px.line(data_frame = Fixed_Variable, x="Fiscal_Period", y=Fixed_Variable.columns,title ="Fixed & Variable Dist
        CM_Per_Carton =  px.line(data_frame=CTS_Merge, x='Fiscal_Period', y='CM_Per_Carton', title='CM Per Carton')
        Fixed_Distribution_Per_Carton = px.line(Fixed_Variable_Per_Carton, x="Fiscal_Period", y=Fixed_Variable_Per_Carton.columns, title = 'Variable & Fixed

        # Percent of Sales
        Delivery_Per_Sales = px.line(data_frame=Delivery_Percent_Sales, x='Fiscal_Period', y='Delivery_Percent_Sales',color = 'Fiscal_Year', title = 'Delive
        Distribution_Per_Sales = px.line(data_frame=Distribution_Percent_Sales, x='Fiscal_Period', y='Distribution_Percent_Sales',color = 'Fiscal_Year', tit
        Var_Fixed_Per_Sales = px.line(data_frame = Fixed_Variable_Per_Sales, x="Fiscal_Period", y=Fixed_Variable_Per_Sales.columns, title = 'Variable & Fixe
        CM_Per_Sales = px.line(data_frame = Contribution_Percent_Sales, x="Fiscal_Period", y='Contribution_Percent_Sales', title = 'CM as Per Sales',color =
        Summary = px.bar(new_df, x='index', y='Avg_Values', text='index',text_auto='.3s',color='index')

        # Image adding in the script

        #Using direct image file path
        image_path = 'C://Users//prapa001//Downloads//Customer_Analytics.png'

        #Using Pillow to read the the image
        pil_img = Image.open("C://Users//prapa001//Downloads//Customer_Analytics.png")

        # Using base64 encoding and decoding
        def b64_image(image_filename):
            with open(image_filename, 'rb') as f:
                image = f.read()
            return 'data:image/png;base64,' + base64.b64encode(image).decode('utf-8')

        #  Application name
        app = dash.Dash(__name__)

        # Set up the layout using an overall div
        app.layout = html.Div(
            children=[html.Img(src=b64_image(image_path)),
          # Add a H1
```

```
  html.H1('Customer Analytics w/Delivery & Distribution Expense'), # passing the direct file path
  #f"Prepared: {datetime.now().date()}",
  " by  ", html.B(" Parth Prajapati,"),
  html.I("Associate Data Scientist"),
  # Add both graphs
  dcc.Graph(id='line_graph', figure=line_graph_sales),
  dcc.Graph(id='line_graph', figure=line_graph_units),
  dcc.Graph(id='line_graph', figure=line_graph_distribution),
  dcc.Graph(id='line_graph', figure=line_graph_delivery),
  dcc.Graph(id='line_graph', figure=Sales_Per_Carton),
  dcc.Graph(id='line_graph', figure=Delivery_Per_Carton),
  dcc.Graph(id='line_graph', figure=Distribution_Per_Carton),
  dcc.Graph(id='line_graph', figure=Fixed_Distribution_Per_Carton),
  dcc.Graph(id='line_graph', figure=CM_Per_Carton),
  dcc.Graph(id='line_graph', figure=Delivery_Per_Sales),
  dcc.Graph(id='line_graph', figure=Distribution_Per_Sales),
  dcc.Graph(id='line_graph', figure=Var_Fixed_Per_Sales),
  dcc.Graph(id='line_graph', figure=CM_Per_Sales),
  dcc.Graph(id='bar_graph', figure=Summary)
  ])

@app.callback(
    Output('dd-output-container', 'children'),
    Input('demo-dropdown', 'value')
)
def update_output(value):
    return f'You have selected {value}'

if __name__ == '__main__':
    app.run_server(debug=False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off

 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C to quit)
```

```
Exception on / [GET]
Traceback (most recent call last):
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\flask\app.py", line 2447, in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\flask\app.py", line 1952, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\flask\app.py", line 1821, in handle_user_exception
    reraise(exc_type, exc_value, tb)
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\flask\_compat.py", line 39, in reraise
    raise value
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\flask\app.py", line 1948, in full_dispatch_request
    rv = self.preprocess_request()
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\flask\app.py", line 2242, in preprocess_request
    rv = func()
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\dash\dash.py", line 1306, in _setup_server
    _validate.validate_layout(self.layout, self._layout_value())
  File "C:\Users\prapa001\Anaconda3\lib\site-packages\dash\_validate.py", line 408, in validate_layout
    raise exceptions.DuplicateIdError(
dash.exceptions.DuplicateIdError: Duplicate component id found in the initial layout: `line_graph`

127.0.0.1 - - [03/Mar/2023 17:38:31] "GET / HTTP/1.1" 500 -
127.0.0.1 - - [03/Mar/2023 17:38:31] "GET /favicon.ico HTTP/1.1" 200 -
127.0.0.1 - - [03/Mar/2023 17:38:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/Mar/2023 17:38:34] "GET /_dash-layout HTTP/1.1" 200 -
127.0.0.1 - - [03/Mar/2023 17:38:34] "GET /_dash-dependencies HTTP/1.1" 200 -
127.0.0.1 - - [03/Mar/2023 17:38:35] "GET /_dash-component-suites/dash/dcc/async-graph.js HTTP/1.1" 200 -
127.0.0.1 - - [03/Mar/2023 17:38:35] "GET /_dash-component-suites/dash/dcc/async-plotlyjs.js HTTP/1.1" 200 -
```

In [ ]:

```
# Application with Drop Down Menu
Sales_Year_01 = CTS_Master.groupby(['Fiscal_Year','Fiscal_Period']).sum()[['Net_Sales','Net_Units']].reset_index()
# Line Graphs
line_graph_sales_01 = px.line(data_frame=Sales_Year_01, x='Fiscal_Period', y='Net_Sales', title='Total Sales by Month',color = 'Fiscal_Year')
line_graph_units_01 = px.line(data_frame=Sales_Year_01, x='Fiscal_Period', y='Net_Units', title='Total Units by Month',color = 'Fiscal_Year')
# Create the dropdown options
Fiscal_Year = Sales_Year_01["Fiscal_Year"].unique()
dropdown_options = [{"label": str(year), "value": year} for year in Fiscal_Year]


# Define the app layout
app.layout = html.Div([
    dcc.Dropdown(
        id='year-dropdown',
        options=dropdown_options,
        value=Fiscal_Year[1]  # set the initial value to the first year
    ),
    html.Div([
  # Add both graphs
  dcc.Graph(id='line_graph', figure=line_graph_sales_01),
  dcc.Graph(id='line_graph', figure=line_graph_units_01)
    ])
])

# Define the callback to update the plots based on the selected year
@app.callback(
    [Output(component_id='line_graph', component_property='figure'),
     Output(component_id='line_graph', component_property='figure')],
    [Input(component_id='year-dropdown', component_property='value')]
)

def update_plots(selected_year):
    # Filter the data based on the selected year
    filtered_df = Sales_Year_01[Sales_Year_01["Fiscal_Year"] == selected_year]

    # Create the plots using Plotly Express
    fig1 = px.line(filtered_df, x="X1", y="X2")
    fig2 = px.line(filtered_df, x="X3", nbins=20)

    # Return the plots
    return fig1, fig2

if __name__ == '__main__':
    app.run_server(debug=False)
```