```
#Importing basic packages
import os
import warnings
import requests
import numpy as np
import pandas as pd
import calendar
import datetime
import xlrd
#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
from datetime import timedelta
from openpyxl import Workbook
from openpyxl import load_workbook
# Importing all the necessary packages
import os
import PyPDF2
import re
import pandas as pd
from PyPDF2 import PdfReader
import PyPDF2
from PyPDF2 import PdfFileReader
from typing import List
# !/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 snowflake.connector.pandas_tools import pd_writer
from sqlalchemy import create_engine
from sqlalchemy.types import Integer, Text, String, DateTime
from snowflake.sqlalchemy import URL
import pandas as pd
import numpy as n
import os
import json
from datetime import date

# Import advanced visualisation libraries
import plotly.graph_objs as go
from plotly.subplots import make_subplots
```

```
C:\Users\prapa001\Anaconda3\envs\customer_analytics\lib\site-packages\snowflake\connector\options.py:96: UserWarning: You have an incompatible version of 'pyarrow' installed (14.0.1), please install a version that adheres to: 'pyarrow<
6.1.0,>=6.0.0; extra == "pandas"'
  warn_incompatible_dep(
C:\Users\prapa001\Anaconda3\envs\customer_analytics\lib\site-packages\snowflake\connector\vendored\requests\__init__.py:102: RequestsDependencyWarning: urllib3 (1.26.6) or chardet (5.2.0)/charset_normalizer (2.0.12) doesn't match a sup
ported version!
  warnings.warn("urllib3 ({}) or chardet ({})/charset_normalizer ({}) doesn't match a supported "
```

```
# Replace 'your_file.xlsx' with the name of your Excel file
file_name = 'HiTouch_Soms_Amzn_Plr_3_Python_test_2.xlsx'
file_path = r'C:\Users\prapa001\Downloads\\' + file_name

# Read the Excel file
Amazon_plr3 = pd.read_excel(file_path)
```

```
# Amazon Python Rates Data Frame
Amazon_plr3_Py_Rating = Amazon_plr3[['SKU', 'SKU_Description','SKU Dropship','Division', 'Department', 'Class',
       'Division_Name', 'Department_Name', 'Class_Name', 'Active_FC_Count',
       'Inactive_FC_Count', 'Sales_Total_Units_Net',
       'ADJUSTED_NET_SALES_W_COU_AMT_$', '% of Sales', 'Sales_$_SMS_COS',
       'FC_Variable_Handling_Expense_Final', 'E-commerce Fee', 'Holding_Cost',
       'Fixed_Expense_Final', 'Total_Distribution',
       'Delivery_Expense_Courier_Final', 'Delivery_Expense_Fleet_Final',
       'Delivery_Expense_UPS_Final', 'Delivery_Expense_FedEx_Final',
       'Total_Delivery_Costs', 'Contribution_Margin', 'LENGTH', 'WIDTH',
       'HEIGHT', 'FC_DIMs_Weight_lbs',  'SKU Model Num',
       'DISCONTINUED FLAG', 'ONHAND QTY', 'SKU ParentChild']]
```

```
# Assuming 'Amazon_plr3' is a DataFrame
# We will use .loc[] to avoid SettingWithCopyWarning
Amazon_plr3_Py_Rating['Sales Per Unit'] = Amazon_plr3_Py_Rating.loc[:, 'ADJUSTED_NET_SALES_W_COU_AMT_$'] / Amazon_plr3_Py_Rating.loc[:, 'Sales_Total_Units_Net']
Amazon_plr3_Py_Rating['Delivery Per Unit'] = Amazon_plr3_Py_Rating.loc[:, 'Total_Delivery_Costs'] / Amazon_plr3_Py_Rating.loc[:, 'Sales_Total_Units_Net']
Amazon_plr3_Py_Rating['Distribution Variable Per Unit'] = Amazon_plr3_Py_Rating.loc[:, 'FC_Variable_Handling_Expense_Final'] / Amazon_plr3_Py_Rating.loc[:, 'Sales_Total_Units_Net']
Amazon_plr3_Py_Rating['Contribution Margin Per Unit'] = Amazon_plr3_Py_Rating.loc[:, 'Contribution_Margin'] / Amazon_plr3_Py_Rating.loc[:, 'Sales_Total_Units_Net']
Amazon_plr3_Py_Rating['Delivery + Variable Per Unit'] = Amazon_plr3_Py_Rating.loc[:, 'Delivery Per Unit'] + Amazon_plr3_Py_Rating.loc[:, 'Distribution Variable Per Unit']
```

...

```python
# Functions for Dim 1, Dim 2, Dim 3
def calculate_dim1(length, width, height):
    # DIM 1 is the largest dimension
    return max(length, width, height)

def calculate_dim3(length, width, height):
    # DIM 3 is the smallest dimension
    return min(length, width, height)

def calculate_dim2(length, width, height, dim1, dim3):
    # DIM 2 is the remaining dimension (neither the largest nor the smallest)
    # Create a list of the dimensions and sort it
    dimensions = sorted([length, width, height])

    # DIM 1 is the largest and DIM 3 is the smallest, so DIM 2 will be the middle value
    return dimensions[1]  # This is the middle value after sorting

def calculate_girth(dim1, dim2, dim3):
    # Girth calculation according to the provided formula
    return dim1 + 2 * (dim2 + dim3)

# Now, applying the functions to the DataFrame
Amazon_plr3_Py_Rating['DIM 1'] = Amazon_plr3_Py_Rating.apply(lambda row: calculate_dim1(row['LENGTH'], row['WIDTH'], row['HEIGHT']), axis=1)
Amazon_plr3_Py_Rating['DIM 3'] = Amazon_plr3_Py_Rating.apply(lambda row: calculate_dim3(row['LENGTH'], row['WIDTH'], row['HEIGHT']), axis=1)
Amazon_plr3_Py_Rating['DIM 2'] = Amazon_plr3_Py_Rating.apply(lambda row: calculate_dim2(row['LENGTH'], row['WIDTH'], row['HEIGHT'], row['DIM 1'], row['DIM 3']), axis=1)
Amazon_plr3_Py_Rating['Girth'] = Amazon_plr3_Py_Rating.apply(lambda row: calculate_girth(row['DIM 1'], row['DIM 2'], row['DIM 3']), axis=1)
Amazon_plr3_Py_Rating['Length + Girth'] = Amazon_plr3_Py_Rating['LENGTH'] + Amazon_plr3_Py_Rating['Girth']
Amazon_plr3_Py_Rating['Ounces'] = Amazon_plr3_Py_Rating['FC_DIMs_Weight_lbs']/0.0625

# Amazon standard size tier
def amazon_size_tier_flag(dim1, dim2, dim3):
    if dim1 <= 15 and dim2 <= 12 and dim3 <= 0.75:
        return "Small Standard Size"
    else:
        return "Large Standard Size"

 # Applying function into the data frame
Amazon_plr3_Py_Rating['Amazon Size Tier'] = Amazon_plr3_Py_Rating.apply(
    lambda row: amazon_size_tier_flag(row['DIM 1'], row['DIM 2'], row['DIM 3']),
    axis=1
)

# Define the rate brackets as a nested dictionary
rate_brackets = {
    (0, 4): {"Small Standard Size": 3.42, "Large Standard Size": 4.16},
    (4, 8): {"Small Standard Size": 3.60, "Large Standard Size": 4.38},
    (8, 12): {"Small Standard Size": 3.78, "Large Standard Size": 4.54},
    (12, 16): {"Small Standard Size": 3.97, "Large Standard Size": 5.05},
    (16, 24): {"Large Standard Size": 5.70},  # No small standard size rate in this bracket
    (24, 32): {"Large Standard Size": 5.99},
    (32, 40): {"Large Standard Size": 6.60},
    (40, 48): {"Large Standard Size": 6.89},
    (48, 320): {"Large Standard Size": 7.67},
```

```
}

# Default rate if none of the brackets match
default_rate = 6.3

# Function to get the rate based on ounces and size tier
def get_rate(ounces, size_tier):
    for (lower_bound, upper_bound), rates in rate_brackets.items():
        if lower_bound < ounces <= upper_bound:
            # Return the corresponding rate if it exists, otherwise the default rate for the tier
            return rates.get(size_tier, default_rate)
    # If no brackets matched, return the default rate
    return default_rate

# Applying the function to the DataFrame
Amazon_plr3_Py_Rating['Az Standard Rates'] = Amazon_plr3_Py_Rating.apply(
    lambda row: get_rate(row['Ounces'], row['Amazon Size Tier']), axis=1
)

#  Weight function over 3 lbs
def calculate_weight_adjustment(row):
    if row['Amazon Size Tier'] == "Large Standard Size" and row['FC_DIMs_Weight_lbs'] > 3:
        return row['FC_DIMs_Weight_lbs'] - 3
    else:
        return "-"

# Apply the function to each row in the DataFrame
Amazon_plr3_Py_Rating['Adjusted_Weight_Over_3lb'] = Amazon_plr3_Py_Rating.apply(calculate_weight_adjustment, axis=1)

# Define the function
def calculate_adjusted_weight(adjusted_weight_over_3lb):
    try:
        result = adjusted_weight_over_3lb * 0.16
        return result
    except Exception as e:
        return 0

# Apply the function to the DataFrame column
Amazon_plr3_Py_Rating['Multiplying on $0.16 over 3LB'] = Amazon_plr3_Py_Rating['Adjusted_Weight_Over_3lb'].apply(calculate_adjusted_weight).round(2)

# Amazon Standard rates 01
Amazon_plr3_Py_Rating['Amazon_Standard_Rates_01'] = Amazon_plr3_Py_Rating['Az Standard Rates']  + Amazon_plr3_Py_Rating['Multiplying on $0.16 over 3LB']

# Low FBA Rate implmentation
low_fba_rate_brackets = {
    (0, 4, 10): {"Small Standard Size": 2.45, "Large Standard Size": 3.09},
    (4, 8, 10): {"Small Standard Size": 2.63, "Large Standard Size": 3.31},
    (8, 12, 10): {"Small Standard Size": 2.81, "Large Standard Size": 3.47},
    (12, 16, 10): {"Small Standard Size": 3.00, "Large Standard Size": 3.98},
    (16, 24, 10): {"Large Standard Size": 4.63},  # Assuming only large standard size in this bracket
    (24, 32, 10): {"Large Standard Size": 4.92},
    (32, 40, 10): {"Large Standard Size": 5.33},
    (40, 48, 10): {"Large Standard Size": 5.62},
    (48, 320, 10): {"Large Standard Size": 6.4},
```

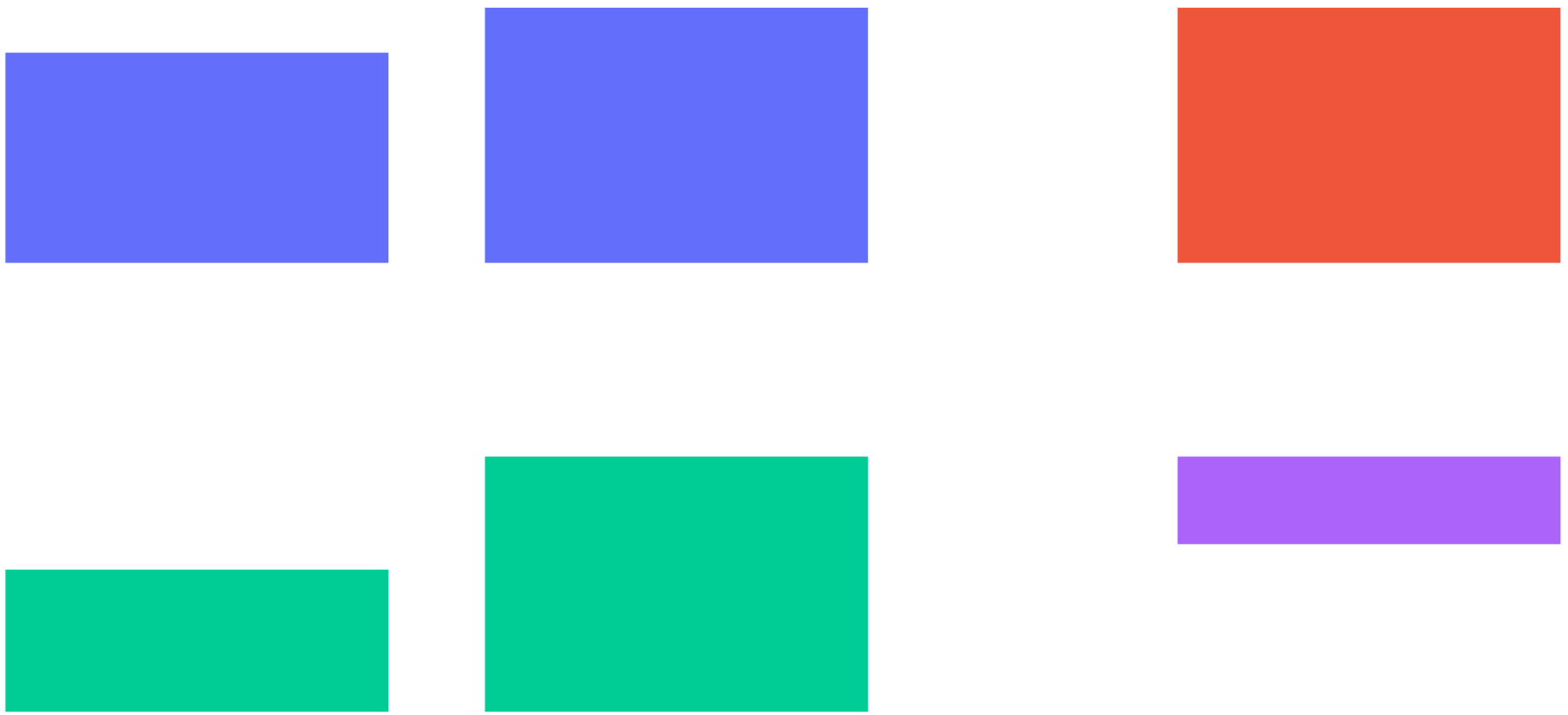
```
}

# Function for implementing nested dictionary
def get_low_fba_rate(ounces, sales_per_unit, amazon_size_tier, rate_brackets):
    for (lower_bound, upper_bound, sales_bound), rates in rate_brackets.items():
        if lower_bound < ounces <= upper_bound and sales_per_unit < sales_bound:
            return rates.get(amazon_size_tier, 0)
    return 0

# Applying the refactored function to the DataFrame
Amazon_plr3_Py_Rating['Low_FBA_Standard_Rate_Python'] = Amazon_plr3_Py_Rating.apply(
    lambda row: get_low_fba_rate(
        row['Ounces'],
        row['Sales Per Unit'],
        row['Amazon Size Tier'],
        low_fba_rate_brackets
    ), axis=1
)

# Embedding low FBA and Amazon Standard Rates
def calculate_amazon_rate(Low_FBA_Standard_Rate_Python, Amazon_Standard_Rates_01):
    if Low_FBA_Standard_Rate_Python == 0:
        return Amazon_Standard_Rates_01
    else:
        return Low_FBA_Standard_Rate_Python

# Apply the function to the DataFrame
Amazon_plr3_Py_Rating['Final_Amazon_Rates'] = Amazon_plr3_Py_Rating.apply(lambda row: calculate_amazon_rate(row['Low_FBA_Standard_Rate_Python'], row['Amazon_Standard_Rates_01']), axis=1)

# Over Size Flag Criteria
def determine_package_size(row):
    weight_lbs = row['FC_DIMs_Weight_lbs']
    dim1 = row['DIM 1']
    dim2 = row['DIM 2']
    length_girth = row['Length + Girth']

    if (20 < weight_lbs < 70) or (18 < dim1 <= 60 and 14 < dim2 <= 30 and 62 < length_girth <= 130):
        return "Small Oversize"
    elif (70 < weight_lbs <= 150) and (60 < dim1 <= 108):
        return "Medium Oversize"
    elif 130 < length_girth <= 165:
        return "Large Oversize"
    elif dim1 > 108 or length_girth > 165:
        return "Special Oversize"
    elif length_girth > 165:  # This condition seems redundant due to the previous condition.
        return "Special Oversize"
    else:
        return "-"

# Applying the function
Amazon_plr3_Py_Rating['Oversize_Flag_Python'] = Amazon_plr3_Py_Rating.apply(determine_package_size, axis=1)

# Oversize Flag rates
def get_oversize_rate(oversize_flag):
```

```
    if oversize_flag == "Small Oversize":
        return 10.7
    elif oversize_flag == "Medium Oversize":
        return 21.6
    elif oversize_flag == "Large Oversize":
        return 92.5
    elif oversize_flag == "Special Oversize":
        return 161
    else:
        return "0"

# Assuming Amazon_plr3 is your DataFrame and it has a column named 'Oversize_Flag_Python'
# You can apply this function to the column like this:

Amazon_plr3_Py_Rating['Oversize_Rate_Py'] = Amazon_plr3_Py_Rating['Oversize_Flag_Python'].apply(get_oversize_rate)

# Define the function
def calculate_amazon_rate(Final_Amazon_Rates, Oversize_Rate_Py):
    if Oversize_Rate_Py == '0':
        return Final_Amazon_Rates
    else:
        return Oversize_Rate_Py

#Final AR w/OS - SR = Final  Amazon rates includes standard amazon rates with Low FBA rates
Amazon_plr3_Py_Rating['Final AR w/OS - SR'] = Amazon_plr3_Py_Rating.apply(lambda row: calculate_amazon_rate(row['Final_Amazon_Rates'], row['Oversize_Rate_Py']), axis=1)

# Oversize additional weight function
def calculate_weight(row):
    if row['Oversize_Flag_Python'] in ['Small Oversize', 'Medium Oversize']:
        return row['FC_DIMs_Weight_lbs'] - 1
    elif row['Oversize_Flag_Python'] in ['Large Oversize', 'Special Oversize']:
        return row['FC_DIMs_Weight_lbs'] - 92
    else:
        return 0

# Apply the function to the DataFrame.
Amazon_plr3_Py_Rating['Oversize_additional_weight'] = Amazon_plr3_Py_Rating.apply(calculate_weight, axis=1)

# Final Additional weight function
def final_additonal_weight(Oversize_additional_weight):
    if Oversize_additional_weight <= 0:
        return 0
    else:
         return Oversize_additional_weight

# Final Additional Weight apply funciton
Amazon_plr3_Py_Rating['Final_additional_weight'] = Amazon_plr3_Py_Rating['Oversize_additional_weight'].apply(final_additonal_weight)

# Over size additional total rates
def over_size_additional_total_rates(row):
    if row['Oversize_Flag_Python'] in ['Small Oversize', 'Meidume Oversize']:
      return '0.42'
    elif row['Oversize_Flag_Python'] in ['Large Oversize', 'Special Oversize']:
        return '0.83'
```

```
    else:
        return '0'

# Applying functions into the Data Frame
Amazon_plr3_Py_Rating['Oversize additional rates'] = Amazon_plr3_Py_Rating.apply(over_size_additional_total_rates, axis = 1)

# convert pversize additional rates to float 64 data type
Amazon_plr3_Py_Rating['Oversize additional rates'] = Amazon_plr3_Py_Rating['Oversize additional rates'].astype(float)
# Over Size net expense
def over_size_net_expense(row):
    if row['Oversize additional rates'] == '0':
         return '0'
    elif row['Final_additional_weight'] == '0':
        return '0'
    elif row['Final_additional_weight'] * row['Oversize additional rates'] == '0':
        return '0'
    else:
        return row['Final_additional_weight'] * row['Oversize additional rates']

# Applyinh function into the data frame
Amazon_plr3_Py_Rating['Oversize_additional_total_rates'] = Amazon_plr3_Py_Rating.apply(over_size_net_expense, axis = 1)

# Final Amazon Fullfilment expense
Amazon_plr3_Py_Rating['Final_Actual_Rates'] = Amazon_plr3_Py_Rating['Final AR w/OS - SR'] + Amazon_plr3_Py_Rating['Oversize_additional_total_rates']

# CTS Delivery + Variable Per Unit
Amazon_plr3_Py_Rating['Net Delivery + Variable Per Unit'] = Amazon_plr3_Py_Rating['Delivery Per Unit'] + Amazon_plr3_Py_Rating['Distribution Variable Per Unit']
Amazon_plr3_Py_Rating['Net Delivery + Variable Per Unit'] = Amazon_plr3_Py_Rating['Net Delivery + Variable Per Unit'] * -1
Amazon_plr3_Py_Rating['Net Delivery + Variable Per Unit'] = Amazon_plr3_Py_Rating['Net Delivery + Variable Per Unit'].round(2)

# Applying the condition directly within the 'apply' method.
Amazon_plr3_Py_Rating['Flag'] = Amazon_plr3_Py_Rating.apply(lambda row: "Amazon is winning" if row['Net Delivery + Variable Per Unit'] > row['Final_Actual_Rates'] else "We are winning", axis=1)

# Amazon net fullfillment expense
Amazon_plr3_Py_Rating['Amazon_Net_Fullfillment_Expense'] = Amazon_plr3_Py_Rating['Sales_Total_Units_Net'] *  Amazon_plr3_Py_Rating['Final_Actual_Rates']
```

...

```
# Pe Unit Data df
Per_Unit = Amazon_plr3_Py_Rating.loc[:, ['SKU', 'SKU_Description', 'Class_Name', 'SKU Dropship', 'Sales_Total_Units_Net', 'Sales Per Unit', 'Delivery Per Unit', 'Distribution Variable Per Unit', 'Contribution Margir
# Per unit savings
Per_Unit['Savings_Per_Unit'] = Amazon_plr3_Py_Rating['Net Delivery + Variable Per Unit']  - Amazon_plr3_Py_Rating['Final_Actual_Rates']
# Net_Savings
Per_Unit['Net_Savings'] = Per_Unit['Savings_Per_Unit'] * Per_Unit['Sales_Total_Units_Net']
# Net Delivery adn Variable Expense
Per_Unit['Net_Delivery_Variable_Expense'] = Per_Unit['Net Delivery + Variable Per Unit'] * Per_Unit['Sales_Total_Units_Net']
# Per Unit
Per_Unit['Net_Amzn_Fullfillment_Expense'] = Per_Unit['Final_Actual_Rates'] * Per_Unit['Sales_Total_Units_Net']
# Final Summary Svings
Per_Unit['Active_FC_Count'] = Per_Unit['Active_FC_Count'].fillna(0).astype('int64')
# Filtering SKU with active atleast 1 FC and SKU which are Non - Drop-ship SKUs
Flag = Per_Unit[(Per_Unit['SKU Dropship'] != 1.0) & (Per_Unit['Active_FC_Count'] > 0.0)]
Flag_01 = Flag.groupby('Flag').agg({'SKU' : 'nunique','Net Delivery + Variable Per Unit' : 'mean', 'Final_Actual_Rates' : 'mean','Sales_Total_Units_Net' : 'sum', 'Net_Savings' : 'sum', 'Net_Delivery_Variable_Expense
# Create a 2x2 subplot layout
fig = make_subplots(rows=2, cols=2, subplot_titles=('Count of SKUs', 'Net Delivery + Variable Per Unit',   'Final Actual Rates - (Labor + Packaging + Delivery + Customer Service)', 'Net Savings'))
# Plot 1: Count of SKUs
fig.add_trace(go.Bar(x=Flag_01['Flag'], y=Flag_01['SKU'], name='Count of SKUs'), row=1, col=1)
# Plot 2: Net Delivery + Variable Per Unit
fig.add_trace(go.Bar(x=Flag_01['Flag'], y=Flag_01['Net Delivery + Variable Per Unit'], name='Net Delivery + Variable Per Unit'), row=1, col=2)
# Plot 3: Final_Actual_Rates
fig.add_trace(go.Bar(x=Flag_01['Flag'], y=Flag_01['Final_Actual_Rates'], name='Final_Actual_Rates'), row=2, col=1)

# Plot 4: Net Savings
fig.add_trace(go.Bar(x=Flag_01['Flag'], y=Flag_01['Net_Savings'], name='Net Savings'), row=2, col=2)
# Add x-axis and y-axis labels
for i in range(1, 3):  # Rows
    for j in range(1, 3):  # Columns
        fig.update_xaxes(title_text='Flag', row=i, col=j)
        fig.update_yaxes(title_text='Value', row=i, col=j)

# Update layout with black background
fig.update_layout(
    height=800,
    width=2000,
    title_text="SKU  w/NDS & Active atleast 1 FC Savings Metrics - P8",
    showlegend=False,
    plot_bgcolor='black',  # Set plot background to black
    paper_bgcolor='black',  # Set paper background to black
    font=dict(color='white')  # Set font color to white for visibility
)
# fig show
fig.show()
Flag_01
```



	Flag	SKU	Net Delivery + Variable Per Unit	Final_Actual_Rates	Sales_Total_Units_Net	Net_Savings	Net_Delivery_Variable_Expense	Net_Amzn_Fullfillment_Expense
0	Amazon is winning	2293	8.13	5.09	82668	180152.92	630710.58	450557.66
1	We is winning	2782	4.89	9.12	103499	-314807.54	610851.15	925658.69

```
Per_Unit[Per_Unit['Flag'] == 'Amazon is winning']
```

	SKU	SKU_Description	Class_Name	SKU Dropship	Sales_Total_Units_Net	Sales Per Unit	Delivery Per Unit	Distribution Variable Per Unit	Contribution Margin Per Unit	Final_Actual_Rates	Net Delivery + Variable Per Unit	Flag	Active_FC_Count	Savings_Per_Unit	Net_Savir
0	24388284	HP 910 CMY/HP 910XL BLK COMBO	OEM INKJET CARTRIDGES	NaN	6625	79.874927	-7.851518	-0.725730	-1.458977	4.3800	8.58	Amazon is winning	20	4.2000	27825.0000
1	2030289	HP 952 XL BLK/STD CLR 4PK	OEM INKJET CARTRIDGES	NaN	2566	124.890000	-7.685846	-0.717384	2.748154	4.3800	8.40	Amazon is winning	21	4.0200	10315.3200
2	24408522	HP 952XL BLK/CMY INK 5PACK	OEM INKJET CARTRIDGES	NaN	2137	205.890000	-7.490731	-0.743921	4.119996	5.0500	8.23	Amazon is winning	20	3.1800	6795.6600
3	2767341	COPPERTOP C ALKALINE BOX OF 12	BATTERIES	NaN	2165	19.429215	-7.125871	-0.764154	-1.241805	5.9900	7.89	Amazon is winning	21	1.9000	4113.5000
4	404061	BADGE INSERT 2-1/4X3-1/2 WHITE	NAME TAGS	NaN	1860	16.490081	-6.967825	-0.840529	0.179201	6.3000	7.81	Amazon is winning	21	1.5100	2808.6000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2743	647057	11 X 8 1/2 3 HOLE TOP LOAD NON	Sheet Protectors - Core	NaN	1	11.990000	-4.194700	-0.880400	2.084719	5.0500	5.08	Amazon is winning	18	0.0300	0.0300
2744	83910	5-SUBJECT NOTEBOOK 9.5X6	Notebooks	NaN	3	14.990000	-5.436667	-0.271467	6.513066	5.7000	5.71	Amazon is winning	17	0.0100	0.0300
2747	24551049	RY24 BS PASSAGES APT 8X11 W	Appointment Books - Wireb	NaN	1	18.110000	-5.228200	-1.080800	1.166366	6.3000	6.31	Amazon is winning	0	0.0100	0.0100
2754	24546510	AY24 BS ENTERPRISE 7X9 W	Academic Calendars	NaN	6	16.653333	-5.381667	-0.897050	1.759830	3.7800	6.28	Amazon is winning	0	2.5000	15.0000
2760	503396	ANGEL SOFT TOILET TISSUE	BATH TISSUE	NaN	2	101.810000	-22.695000	-0.667400	-9.386861	23.3588	23.36	Amazon is winning	21	0.0012	0.0024

2743 rows × 17 columns

```
# Validation views
```

```
df = px.data.tips()
fig = px.histogram(Per_Unit, x="Contribution Margin Per Unit", color="Flag",
                   marginal="box", # or violin, rug
                   hover_data=Per_Unit.columns).update_layout(
                               template='plotly_dark',
                               plot_bgcolor='rgba(0, 0, 0, 0)',
                               paper_bgcolor='rgba(0, 0, 0, 0)')
fig.show()
```