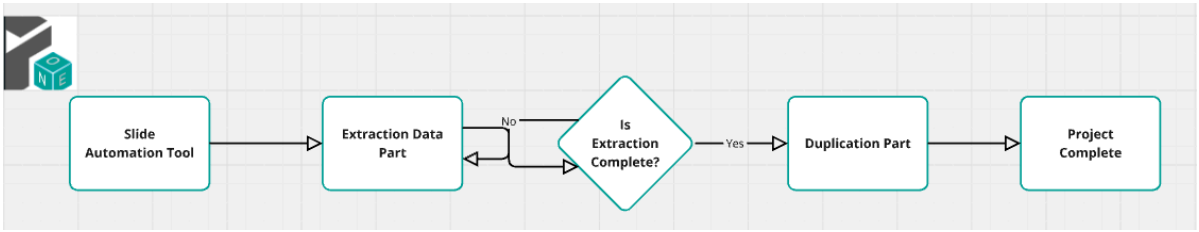
Slide Automation Tool Documentation

slide Automation Flow:



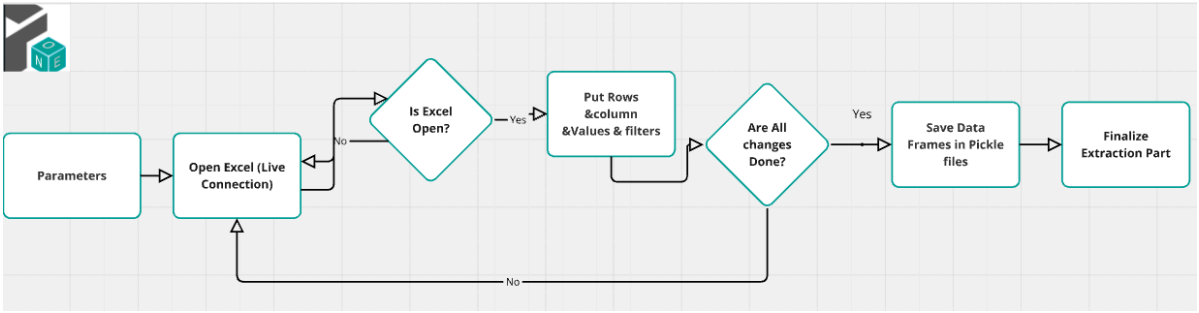
Extraction Part

Duplication Part

Section Name

- Landscape
- Pricing
- PPA
- Mix&Assortment
- Promotion
- Financials
- Pricing_CBC
- Innovation_CBC

Extraction Data Part



Example parameters

is setup defines various parameters for analyzing retail or market data related to client manufacturer, including product categories, financial settings, geographic regions, and time periods for analysis.

```
f_name = os.path.join(os.path.dirname(os.getcwd()),"Sarantis Poland
Dataset.xlsx")
client_manuf = ["Sarantis"]
client_brands = ["Jan Niezbedny", "Stella", "Anna Zaradna"]
decimals = 2
sign = "After"
currency = 'zł'
currency = ' '+ currency if sign.lower() == 'after' else  currency + ' '
categories = ["Garbage Bags"]
sectors = ["Draw Tape", "Wave Top", "Flat Top", "T-Shirt"]
segments = ["Scented", "No Scent"]
subsegments= ["0-20L", "21-35L", "36-60L", "61-120L", "121L And More"]
subcategories= []
national = True
customareas= ""
areas = ['NATIONAL',"CHANNEL"]
regions_RET  = []
channels_RET = []
market_RET = []
regions_CHAN = ["Discounters", "Drugstores", "Groceries", "Hyper/Super"]
channels_CHAN = ["Discounters", "Drugstores", "Groceries Large", "Groceries
Medium", "Groceries Small", "Hypermarkets", "Supermarkets"]
market_CHAN = []
regions_CUST = []
channels_CUST = []
market_CUST = []
data_source = "DATA SOURCE: Trade Panel/Retailer Data | Ending Sep 2024"
years = ['2021', '2022','2023']
ManufOrTopC ="Top Companies"
BrandOrTopB="Top Brands"
```

Open Excel

Example: This code opens Excel (Live Connection) and clear old data it has:

```
f_path = Path.cwd()

excel = client.gencache.EnsureDispatch('Excel.Application')
excel.Visible = True  # False
wb = excel.Workbooks.Open(f_name)
ws=wb.Sheets([s.Name for s in wb.Sheets][0])
s_name = [s.Name for s in wb.Sheets][0]
## If changed we'll need to change the iloc's of the cleaning
pvtTable = ws.PivotTables(1)

#change report layout
pvtTable.RowAxisLayout(1)   #RowAxisLayout(1) for tabular form

#change pivot table style
#Select from Design tab, try out Medium9 or Medium3
pvtTable.TableStyle2 = "pivotStyleMedium21"
pvtTable.ClearTable()

pvtTable.TableRange2.Cut(ws.Range("A16"))

fieldsNamePosition={}
for i in range(1,pvtTable.CubeFields.Count+1):
    fieldsNamePosition[str(pvtTable.CubeFields(i))]=i
```

Add Data in Excel

This code snippet processes and generates data related to market segments using pivot tables in Excel and stores the results as pickled DataFrames for later use. Here's a breakdown of the process:

1. **Initial Setup**:
    - **Dictionaries**: Two dictionaries, `sectors_dfs` and `sectors_P12M_dfs`, are defined to hold the DataFrames for segment data for different time periods.
    - Pivot Table Setup

      : A pivot table (

      ```
      pvtTable
      ```

) is prepared with row, column, filter, and value fields. These fields are:
        - **Rows**: `[Products].[Sector]`
        - **Columns**: `[Calendar].[Year]`
        - **Filters**: Multiple filters such as scope, category, market area, region, channel, and time period.
        - **Values**: A set of measures such as volume sales, value sales, price, growth contribution, etc.
2. **Pivot Table Calculation**:

- The pivot table is cleared (`pvtTable.ClearTable()`) and then set up with the `set_excel_fields` function, which likely applies the field configurations to the pivot table.
- The filter for `[Products].[Category]` is applied to select a category from the `categories` list.
- The filter for `[Scope].[Scope]` is set to "Category", implying that the analysis will be focused on category-level data.

3. **Data Calculation for Different Time Periods**:
   - **P3Y (Past 3 Years)**: The pivot table is used to calculate data for segments by calling `calculate_category_data(sectors_dfs)` after setting the scope to "Category". This function likely processes and stores the results in `sectors_dfs`.
   - **P12M (Past 12 Months)**: The pivot table is adjusted to focus on the "P12M" time period, and `calculate_category_data(sectors_P12M_dfs)` is called to generate the results for the last 12 months, which are stored in `sectors_P12M_dfs`.
4. **Data Storage**:
   - The resulting DataFrames (

     ```
     sectors_dfs
     ```

     and

     ```
     sectors_P12M_dfs
     ```

) are pickled (serialized) and saved as

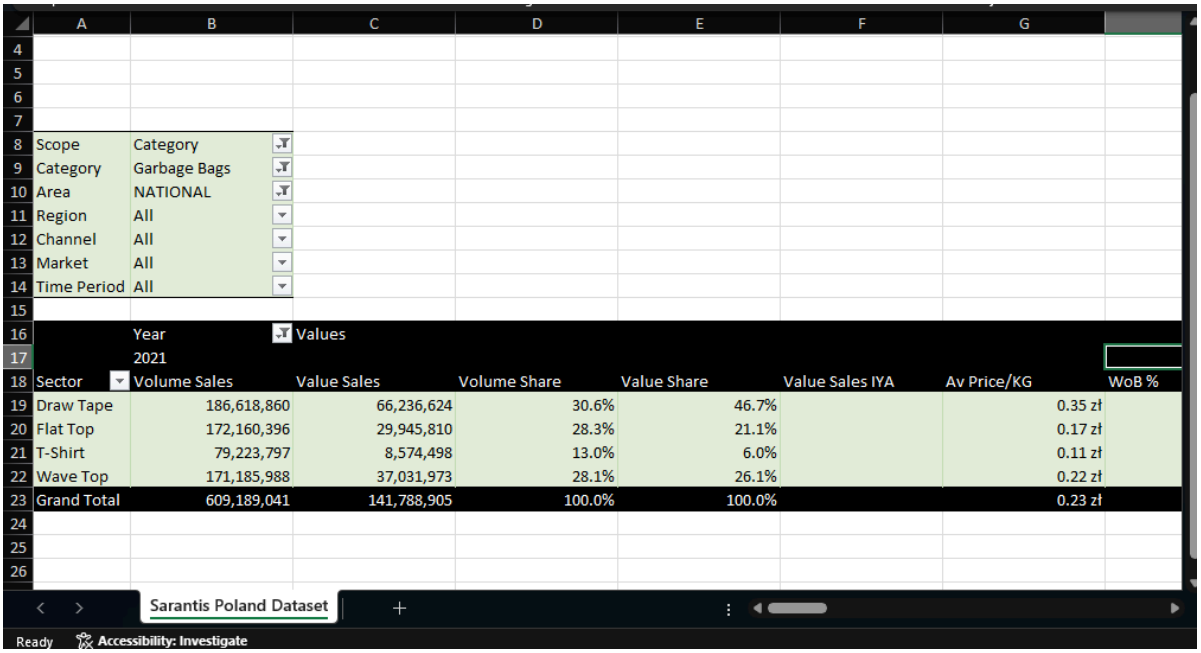
     ```
     .pickle
     ```

     files:
     - `sectors_dfs.pickle`: Stores the 3-year segment data.
     - `sectors_P12M_dfs.pickle`: Stores the 12-month segment data.

```
# Dictionary of DataFrames
sectors_dfs = {}
sectors_P12M_dfs = {}
if len(sectors)!=0:

    row_list = ['[Products].[Sector]']
    column_list=['[Calendar].[Year]']
    filter_list=['[Scope].[Scope]','[Products].[Category]','[Market].
[Area]','[Market].[Region]','[Market].[Channel]','[Market].[Market]','[Time
Logic].[Time Period]']
    value_list=['[Measures].[Volume Sales]','[Measures].[Value
Sales]','[Measures].[Volume Share]','[Measures].[Value Share]','[Measures].[Value
Sales IYA]','[Measures].[Av Price/KG]','[Measures].[WoB %]','[Measures].[Relative
Price]','[Measures].[Growth Contribution]','[Measures].[Volume Sales
IYA]','[Measures].[IYA Price/KG]']

    pvtTable.ClearTable()
```

```
    pvtTable =
set_excel_fields(row_list,column_list,filter_list,value_list,pvtTable)

    #Select the filter values for each filter
    pvtTable.PivotFields("[Products].[Category].[Category]").ClearAllFilters()
    pvtTable.PivotFields('[Products].[Category].[Category]').CurrentPageName =
f'[Products].[Category].&[{categories[0]}]'

    # Segments Dataframes For P3Y For Area = NATIONAL, REGION, CHANNEL,CUSTOM

    pvtTable.PivotFields("[Scope].[Scope].[Scope]").ClearAllFilters()
    pvtTable.PivotFields('[Scope].[Scope].[Scope]').CurrentPageName = '[Scope].
[Scope].&[Category]'

    calculate_category_data(sectors_dfs)

    # Segments Dataframes For P12M For Area = NATIONAL, REGION, CHANNEL,CUSTOM

    pvtTable.CubeFields(list(filter_dictionary_keys(fieldsNamePosition,
'[Calendar].[Year]').values())[0]).Orientation = 0

    pvtTable.PivotFields("[Time Logic].[Time Period].[Time
Period]").ClearAllFilters()
    pvtTable.PivotFields('[Time Logic].[Time Period].[Time
Period]').CurrentPageName = '[Time Logic].[Time Period].&[P12M]'

    calculate_category_data(sectors_P12M_dfs )


with open('Landscape Datasets/sectors_dfs.pickle', 'wb') as handle:
    pickle.dump(sectors_dfs, handle, protocol=pickle.HIGHEST_PROTOCOL)

with open('Landscape Datasets/sectors_P12M_dfs.pickle', 'wb') as handle:
    pickle.dump(sectors_P12M_dfs, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Example : How Live Connection Work

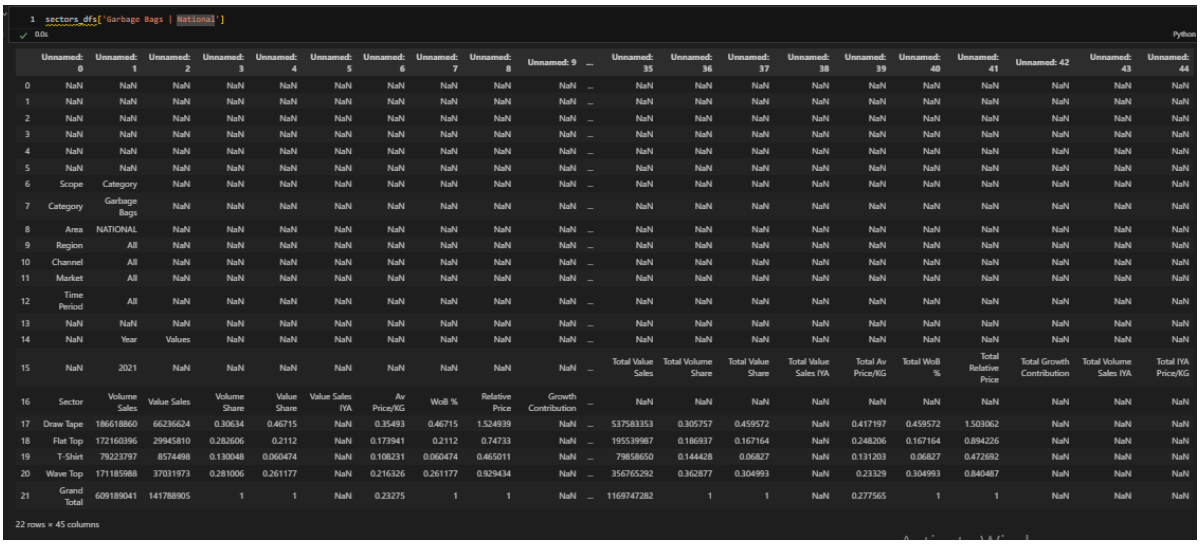


Scope	Category	
Category	Garbage Bags	
Area	NATIONAL	
Region	All	
Channel	All	
Market	All	
Time Period	All	

	Year	Values					
	2021						
Sector	Volume Sales	Value Sales	Volume Share	Value Share	Value Sales IYA	Av Price/KG	WoB %
Draw Tape	186,618,860	66,236,624	30.6%	46.7%		0.35 zł	
Flat Top	172,160,396	29,945,810	28.3%	21.1%		0.17 zł	
T-Shirt	79,223,797	8,574,498	13.0%	6.0%		0.11 zł	
Wave Top	171,185,988	37,031,973	28.1%	26.1%		0.22 zł	
Grand Total	609,189,041	141,788,905	100.0%	100.0%		0.23 zł	

After data frame saved

we save many dataframes in dictionary and then use it in duplication part



```
sectors_dfs['Garbage Bags | National']
```

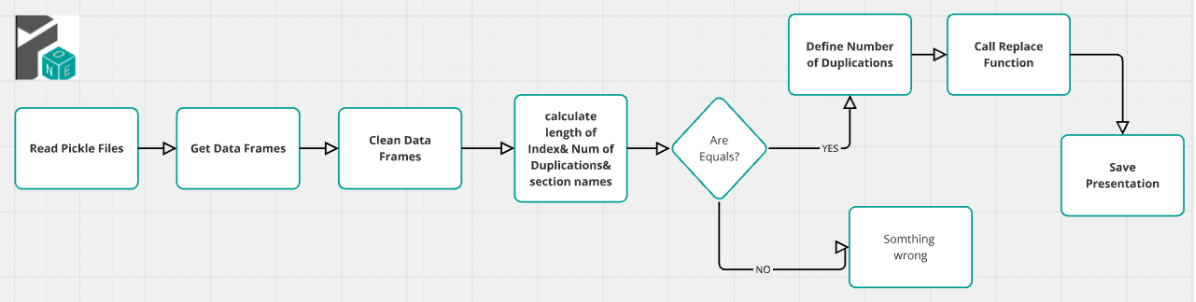
0.0s

Python

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	...	Unnamed: 35	Unnamed: 36	Unnamed: 37	Unnamed: 38	Unnamed: 39	Unnamed: 40	Unnamed: 41	Unnamed: 42	Unnamed: 43	Unnamed: 44
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	Scope	Category	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	Category	Garbage Bags	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	Area	NATIONAL	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	Region	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	Channel	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	Market	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	Time Period	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	Year	Values	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	NaN	2021	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	Total Value Sales	Total Volume Share	Total Value Share	Total Value Sales IYA	Total Av Price/KG	Total WoB %	Total Relative Price	Total Growth Contribution	Total Volume Sales IYA	Total IYA Price/KG
16	Sector	Volume Sales	Value Sales	Volume Share	Value Share	Value Sales IYA	Av Price/KG	WoB %	Relative Price	Growth Contribution	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	Draw Tape	186618860	66236624	0.30634	0.46715	NaN	0.35493	0.46715	1.524939	NaN	...	537583353	0.305757	0.459572	NaN	0.417197	0.459572	1.503062	NaN	NaN	NaN
18	Flat Top	172160396	29945810	0.282606	0.2112	NaN	0.173941	0.2112	0.74733	NaN	...	195539987	0.186937	0.167164	NaN	0.248206	0.167164	0.894226	NaN	NaN	NaN
19	T-Shirt	79223797	8574498	0.130048	0.060474	NaN	0.108231	0.060474	0.465011	NaN	...	79858650	0.144428	0.06827	NaN	0.131203	0.06827	0.472692	NaN	NaN	NaN
20	Wave Top	171185988	37031973	0.281006	0.261177	NaN	0.216326	0.261177	0.929434	NaN	...	356765292	0.362877	0.304993	NaN	0.23329	0.304993	0.840487	NaN	NaN	NaN
21	Grand Total	609189041	141788905	1	1	NaN	0.23275	1	1	NaN	...	1169747282	1	1	NaN	0.277565	1	1	NaN	NaN	NaN

22 rows × 45 columns

Duplication Part



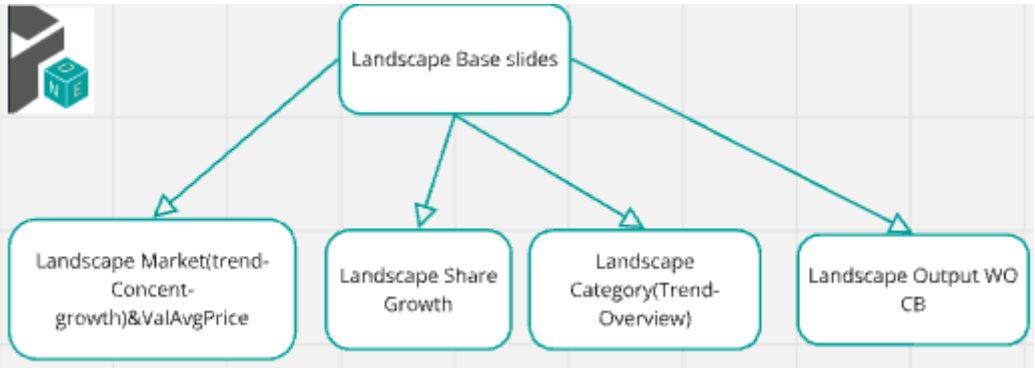
Sections

- Landscape
- Pricing
- PPA
- Mix&Assortment
- Promotion
- Financials
- Pricing_CBC
- Innovation_CBC

Landscape Section

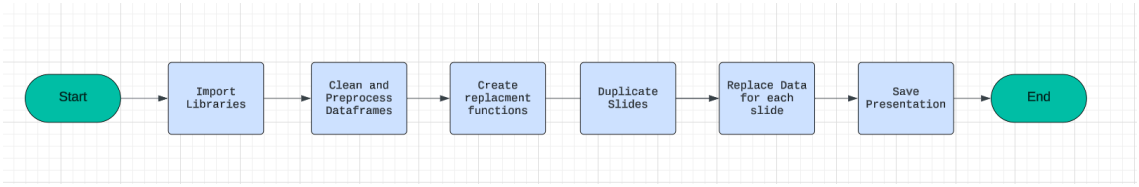
Introduction

In the slide automation landscape: from13 slide base we create 5 decks



1. Landscape Market(trend-Concent-growth)&ValAvgPrice Slides:
    - Market Trends Analysis
    - Market Concentration
    - Market growth contributors
    - Value Sales & Avg Price
2. Landscape ShareGrowth Slides:
    - Share and Growth by Manufacturer/Brands
    - Share and Growth By Manufacturer
    - Momentum Analysis
3. Landscape Category(Trend-Overview)Slides :
    - Category Trends
    - Share Evolution index analysis
    - Category Overview
4. Landscape Output WO CB Slides:
    - Market Trends Analysis
    - Market Concentration
    - Share and Growth by Manufacturer/Brands
    - Share and Growth By Manufacturer
5. Landscape RPVM Slides:
    - Revenue by Price vs. Volume vs. Mix analysis

Project Steps

- Project Flow
- 

- Step 1: Import Libraries we use
- Step 2: modified Data frames: cleaning and preprocessing the data frames
- Step 3: Write Functions to Create Slides: Define functions to Automatically generate slides based on the base slides
- Step 4: Duplicate Slides: Use functions or methods to duplicate existing slides as needed for the presentation.
- Step 5: Replace Data in Slides: update information from the cleaned data frames to slides
- Step 6: Save Presentation

Step 1: Import Libraries we use

Ex: Libraries we use

- This script sets up an environment for working with PowerPoint presentations, data manipulation, filesystem operations, and COM (Component Object Model) object access.
- It imports necessary modules such as 'pptx' for PowerPoint automation, 'win32com' for COM object access and Windows automation, 'pandas' and 'numpy' for data manipulation,
- 'pathlib' for working with filesystem paths, 're' for regular expression operations, and various other modules for general-purpose tasks like file operations and timing functions.
- By importing these modules, the script prepares itself for tasks such as creating or modifying PowerPoint presentations, analyzing data using pandas and numpy, interacting
- with the Windows environment using win32com, and performing filesystem operations using shutil and os. Overall, this script provides a comprehensive setup for automating tasks
- related to PowerPoint presentations and general-purpose Python programming.

```
# Import necessary module for working with PowerPoint presentations
from pptx import Presentation
# Import the win32com.client module, aliasing it as win32 for convenience
import win32com.client as win32
# Import pandas for data manipulation and analysis
import pandas as pd
# Import numpy for numerical computing
import numpy as np
# Import the Path class from pathlib for working with filesystem paths
from pathlib import Path
# Import re for regular expression operations
import re
# Import sys for access to interpreter-related functions
import sys
# Import time for various time-related functions
import time
# Assign win32.constants to a shorter alias win32c for easier access
win32c = win32.constants
# Import shutil for high-level file operations
import shutil
# Import os for operating system dependent functionality
import os
# Import win32com.client again for COM object and functions access
import win32com.client
# Import warnings for warning control functionality
import warnings
```

Step 2: modified Data frames

- This function takes a dictionary of dataframes and a category type as input.
- It iterates over each dataframe in the dictionary and performs cleaning operations,
- such as renaming columns, removing unwanted rows, converting data types, and
- separating totals from the main data. The cleaned dataframes and totals are
- stored in separate dictionaries. Finally, it returns two dictionaries:

- one containing cleaned sector segment data and the other containing totals.

Parameters:

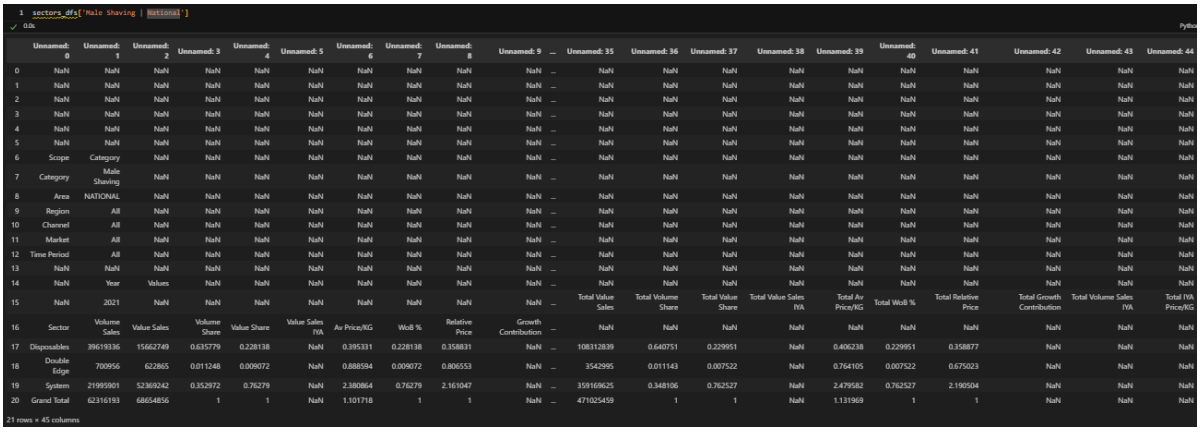
- - inputdic: A dictionary of dataframes where each dataframe represents data.

Returns:

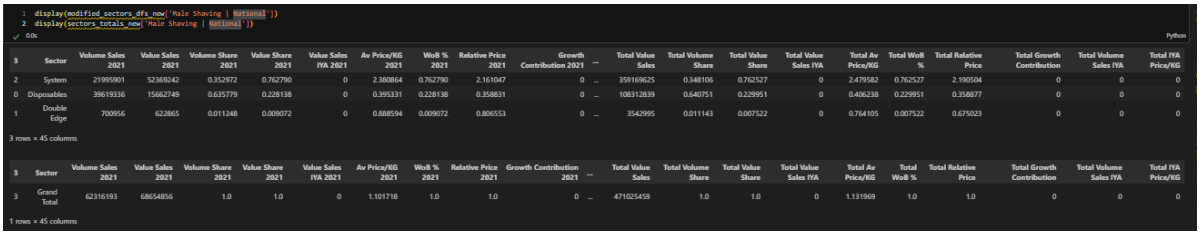
- - outputdic: A dictionary containing cleaned sector segment dataframes for each sector.
- - totaloutputdic: A dictionary containing totals dataframes for each sector.

```
def secsegclean(inputdic):
    outputdic={}
    totaloutputdic={}
    for s in inputdic.keys():
        t = inputdic[s].copy()
        t=DetectHeader(t).fillna(0)
        mod = t[(~t[t.columns[0]].astype(str).str.contains('Grand Total'))]
        mod = mod.sort_values([col for col in mod.columns if 'Value Share' in
col], ascending=False)
        tot = t[(t[t.columns[0]].astype(str).str.contains('Grand Total'))]
        if not mod.empty:
            outputdic[s] = mod
        if not tot.empty:
            totaloutputdic[s] = tot
    return outputdic,totaloutputdic
```

Example: input dataframe before clean

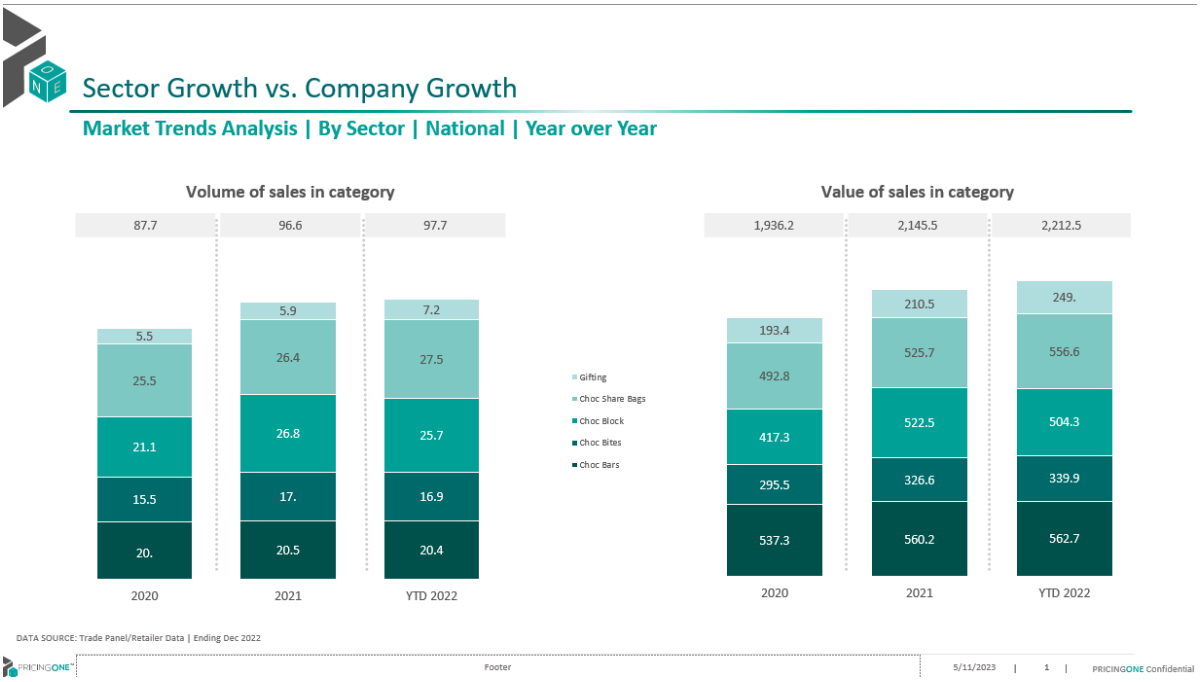


Example: How to call the function & show the DataFrame output



Step 3: Write Functions to Create Slides

To create slides we need some function
Example:Market Trends slides



- Totals_Table_Fill: function populates a table on a slide with totals data from a specified dataframe. It formats the text in the cells, including font size, font name, and alignment.
    - It takes Parameters:
      table (Table): Table shape in the slide.
      list_duplicates (list): List of duplicate names for identifying slides.
      df_totals (dict): Dictionary of total DataFrames for each duplicate name.
      cols (list): Columns in the DataFrame.
      slidenum (int): Slide number.
    - Returns:
      Table: Updated table shape.

```
def Totals_Table_Fill(table, list_duplicates, df_totals, cols, slidenum):

    for i, row in enumerate(table.rows):
        if i != 0:
            for j, cell in enumerate(row.cells):
                cell.text = str(round(df_totals[list_duplicates[slidenum]]
[cols].iloc[0, 1:4][j] / 1000000, 1 ))
                cell.text_frame.paragraphs[0].runs[0].font.size = Pt(10)
                cell.text_frame.paragraphs[0].runs[0].font.name = 'Nexa Book'
                cell.text_frame.paragraphs[0].alignment = PP_ALIGN.CENTER
    return table
```

- Column_Chart_Fill:function customizes a column chart on a slide by filling series with specific colors based on their names and adding formatted data labels to each point in the series

- It takes Parameters:
  chart (Chart): Chart shape in the slide.
  scope (list): List of scope names.

```
def Column_Chart_Fill(chart, scope):

    client_colors = [RGBColor(0, 80, 75), RGBColor(0, 108, 109), RGBColor(0, 160,
151), RGBColor(126, 202, 196), RGBColor(153, 199, 197), RGBColor(178, 223, 220)]
    gray_colors = [RGBColor(217, 217, 217), RGBColor(191, 191, 191),
RGBColor(166, 166, 166), RGBColor(155, 152, 152), RGBColor(127, 127, 127)]

    for i, series in enumerate(chart.series):
        if series.name in scope:
            series.format.fill.solid()
            series.format.fill.fore_color.rgb = client_colors[i if i <
len(client_colors) else -1]
        else:
            series.format.fill.solid()
            series.format.fill.fore_color.rgb = gray_colors[i if i <
len(gray_colors) else -1]
        for j, point in enumerate(series.points):
            data_label = point.data_label
            data_label.has_text_frame = True
            data_label.text_frame.text =  str(round(series.values[j], 1))
            data_label.text_frame.paragraphs[0].runs[0].font.color.rgb =
RGBColor(255, 255, 255)
```

- Markete_Trends function, which automates the creation and updating of market trend analysis slides in a PowerPoint presentation. It fills the slides with charts and tables using data from provided dataframes. The function processes each slide based on a list of duplicate names, adding relevant data and formatting to charts and tables. It supports customization through parameters like position offset and slide grouping criteria. This function is useful for generating detailed, data-driven presentations on market trends, enhancing efficiency and consistency in reporting.
  - It takes Parameters:
    prs (Presentation): PowerPoint presentation object.
    list_duplicates (list): List of duplicate names for identifying slides.
    modified_df (dict): Dictionary of modified DataFrames for each duplicate name.
    df_totals (dict): Dictionary of total DataFrames for each duplicate name.
    scope (list): List of scope names.
    position (int, optional): Position offset for slides. Defaults to 0.
    slide_by (str, optional): Slide grouping criteria. Defaults to ''.

```
def Market_Trends(prs, list_duplicates, modified_df, df_totals, scope,
position=0, slide_by=''):

    for slidenum in range(len(list_duplicates)):
        shapes = prs.slides[slidenum + position].shapes
        charts = []
```

```
        tables = []
        title = shapes.title.text
        shapes[4].text = data_source
        shapes[5].text = f'Market Trends Analysis | By {slide_by} | ' +
list_duplicates[slidenum] + ' | Year over Year'
        shapes[5].text_frame.paragraphs[0].font.bold = True

        for shape in shapes:
            if shape.has_chart:
                charts.append(shape)
            if shape.has_table:
                tables.append(shape)

        for chartnum in range(2):
            chart = charts[chartnum].chart
            table = tables[chartnum].table
            chart_data = CategoryChartData()
            chart_data.categories = ['2021', '2022', 'YTD 2023']

            volume_cols = [c for c in
modified_df[list_duplicates[slidenum]].columns[modified_df[list_duplicates[sliden
um]].columns.str.contains(f'{slide_by}|Volume Sales')]]
            value_cols = [c for c in
modified_df[list_duplicates[slidenum]].columns[(modified_df[list_duplicates[slide
num]].columns.str.contains(f'{slide_by}|Value Sales')) & ~
(modified_df[list_duplicates[slidenum]].columns.str.contains('IYA'))]]

            if chartnum == 0:
                for i in range(modified_df[list_duplicates[slidenum]].shape[0]):
                    series_name = modified_df[list_duplicates[slidenum]]
[volume_cols].iloc[i, 0]
                    number = modified_df[list_duplicates[slidenum]]
[volume_cols].iloc[i, 1:4] / 1000000
                    series = chart_data.add_series(series_name, number)
                    chart.replace_data(chart_data)
                    Column_Chart_Fill(chart, scope)
                Totals_Table_Fill(table, list_duplicates, df_totals, volume_cols,
slidenum)

            elif chartnum == 1:
                value_cols = [c for c in
modified_df[list_duplicates[slidenum]].columns[(modified_df[list_duplicates[slide
num]].columns.str.contains(f'{slide_by}|Value Sales')) & ~
(modified_df[list_duplicates[slidenum]].columns.str.contains('IYA'))]]
                for i in range(modified_df[list_duplicates[slidenum]].shape[0]):
                    series_name = modified_df[list_duplicates[slidenum]]
[value_cols].iloc[i, 0]
                    number = modified_df[list_duplicates[slidenum]]
[value_cols].iloc[i, 1:4] / 1000000
                    series = chart_data.add_series(series_name, number)
                    chart.replace_data(chart_data)
                    Column_Chart_Fill(chart, scope)
                Totals_Table_Fill(table, list_duplicates, df_totals, value_cols,
slidenum)
```

Step 4: Duplicate Slides

- prepares data and configurations for generating market analysis slides in a PowerPoint presentation. It creates index and duplication lists dynamically based on the presence of segment data and the number of data keys in various dictionaries. These lists are used to control the slide generation process. The script also defines a comprehensive list of section names for organizing slide titles, ensuring each slide is labeled appropriately based on its content. This setup allows for automated, consistent, and dynamic creation of market trend analysis slides, which is particularly useful for large presentations with multiple sections and varying data inputs.

```
# This script prepares index and duplication lists for generating PowerPoint
slides
# with various market trends and growth analysis. It dynamically adjusts based on
the presence
# of segment data and compiles a list of section names for slide titles.
index = [
    *[0]*(2+(1 if sectors else 0)+(1 if segments else 0)+(1 if subcategories else
0)+(1 if subsegments else 0)),
           *[1]*(2+(1 if sectors else 0)+(1 if segments else 0)+(1 if
subcategories else 0)+(1 if subsegments else 0)),
               *[2]*((1 if sectors else 0) +(1 if segments else 0)+(1 if
subcategories else 0)+(1 if subsegments else 0)),
               *[2]*section_number,
               *[3]*((1 if sectors else 0) +(1 if segments else 0)+(1 if
subcategories else 0)+(1 if subsegments else 0)),
             *[3]*section_number_Avg,
                ]
index=[i for i in index if i !=[]]

duplication_1 = [len(modified_manuf_dfs_new.keys()),
len(modified_brands_share_new.keys()),len(modified_sectors_dfs_new.keys())if
sectors else 0,len(modified_segment_dfs_new.keys())if segments else
0,len(modified_subsegment_dfs_new.keys())if subsegments else
0,len(modified_subcategories_dfs_new.keys())if subcategories else 0]
duplication_2 = [len(modified_manuf_dfs_new.keys()),
len(modified_brands_share_new.keys()), len(modified_sectors_dfs_new.keys())if
sectors else 0, len(modified_segment_dfs_new.keys())if segments else
0,len(modified_subsegment_dfs_new.keys())if subsegments else
0,len(modified_subcategories_dfs_new.keys())if subcategories else 0]
duplication_3 = [len(modified_sectors_P12M_new.keys()) if sectors else 0,
len(modified_segment_P12M_new.keys())if segments else 0,
len(modified_subsegment_P12M_new.keys())if subsegments else 0,
len(modified_subcategories_P12M_new.keys())if subcategories else
0,*duplication_num]
duplication_4 = [len(modified_sectors_clients_new.keys())if sectors else
0,len(modified_segment_clients_new.keys())if segments else 0,
len(modified_subsegment_clients_new.keys())if subsegments else 0,
len(modified_subcategories_clients_new.keys())if subcategories else 0,
*duplication_num_Avg]

duplication = duplication_1 + duplication_2  + duplication_3  + duplication_4
duplication = [item for item in duplication if item !=0]
```

```
section_names_slide1 = ["Market Trends by Manufacturer","Market Trends by
Brands","Market Trends by Sectors"] + (["Market Trends by Segments"] if
len(segments)>0 else [])+(["Market Trends by SubSegments"] if len(subsegments)>0
else [])+(["Market Trends by SubCategory"] if len(subcategories)>0 else [])
section_names_slide2 = ["Market Concentration By Manufacturer", "Market
Concentration By Brands", "Market Concentration By Sectors"]+ (["Market
Concentration By Segments"] if len(segments)>0 else [])+(["Market Concentration
by SubSegments"] if len(subsegments)>0 else [])+(["Market Concentration by
SubCategory"] if len(subcategories)>0 else [])
section_names_slide3 = (["Market Growth By Sectors"]if len(sectors)>0 else [])+
(["Market Growth By Segments"]if len(segments)>0 else [])+(["Market Growth By
SubSegments"]if len(subsegments)>0 else [])+(["Market Growth By SubCategory"]if
len(subcategories)>0 else [])+[*section_name_Growth]
section_names_slide4 = (["Value Vs AvgPrice By Sectors"]if len(sectors)>0 else
[])+(["Value Vs AvgPrice By Segments"]if len(segments)>0 else [])+(["Value Vs
AvgPrice By SubSegments"]if len(subsegments)>0 else [])+(["Value Vs AvgPrice By
SubCategory"]if len(subcategories)>0 else [])+[*section_name_Avg]


section_names = [ *section_names_slide1
,*section_names_slide2,*section_names_slide3, *section_names_slide4
                 ]
path = os.getcwd() + '\Landscape base.pptx'
new_pre = os.getcwd() + '\Landscape duplicate Market(Trends).pptx'
```

Duplication Function

We use the duplication function to duplicate slides by number of the duplicate and save it in the duplication deck to use it to replace data.
Duplicate slides in a PowerPoint presentation.

Parameters:

- index (list): List of slide indices to duplicate.
- duplication (list): List specifying the number of times each slide should be duplicated.
- section_names (list): List of names for sections to be added.
- path (str): Path to the PowerPoint presentation file.
- new_pre (str): Path to save the duplicated presentation.

Returns:

- str: A message indicating success or failure.

```
####New_With_duplicate
import pythoncom
defslideDuplication(index=[0,1],duplication=[1,1],section_names=
[''],path='',new_pre=''):
    lis=[]
    iflen(index)!=len(duplication)!=len(section_names):
        return'The Index list not equal the Duplication number list in length'
    app = win32.Dispatch("PowerPoint.Application")
    presentation = app.Presentations.Open(path)
    # Iterate through the slides in the original presentation and copy them to
the new presentation
    for i inrange(len(index)):
        iftype(index[i])==list:
```

```
            # If index is a list of slide indices
            for num_duplicate inrange(duplication[i]):
                for k in index[i]:
                    slide=presentation.Slides[k]
                    duplicated_slide = slide.Duplicate()
                    duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-
(duplication[i]*len(index[i])))
        else:
            # If index is a single slide index
            slide=presentation.Slides[index[i]]
            for num_duplicate inrange(duplication[i]):
                duplicated_slide = slide.Duplicate()
                duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-duplication[i])
    # Add sections to the new presentation
    for j inrange(len(lis)):
        if duplication[j]!=0:
            presentation.SectionProperties.AddBeforeSlide(lis[j],section_names[j]
)
    # presentation.ApplyTheme(themePath)
    presentation.SectionProperties.Delete(1, True)
    presentation.SaveAs(new_pre)
    presentation.Close()
    # Close the original presentation and PowerPoint application
    app.Quit()
```

Step 5: Replace Data in Slides

- This script uses the Market_Trends function to generate market trend analysis slides in a PowerPoint presentation for various categories such as Top Companies, Top Brands, Sectors, and Segments. It initializes a position counter p and increments it after each call to ensure the slides are added sequentially. The calculate_position function is used to determine the correct position for each set of slides, allowing for dynamic and organized slide generation based on the provided data dictionaries and scope lists. This approach enables efficient creation of comprehensive market analysis presentations.

```
#  calls the Market_Trends function to generate PowerPoint slides
# for different categories (Top Companies, Top Brands, Sector, Segment) and
# updates the position variable for each call to ensure slides are generated in
the correct order.

p=0
Market_Trends(prs, list(modified_manuf_dfs_new.keys()), modified_manuf_dfs_new,
modified_manuf_totals_new, client_manuf ,position = calculate_position(p),
slide_by = 'Top Companies')
p+=1
Market_Trends(prs, list(modified_brands_share_new.keys()),
modified_brands_share_new, modified_brands_totals_new, client_brands ,position
=calculate_position(p), slide_by = 'Top Brands')
p+=1
Market_Trends(prs, list(modified_sectors_dfs_new.keys()),
modified_sectors_dfs_new, sectors_totals_new, sectors ,position =
calculate_position(p), slide_by = 'Sector')
```

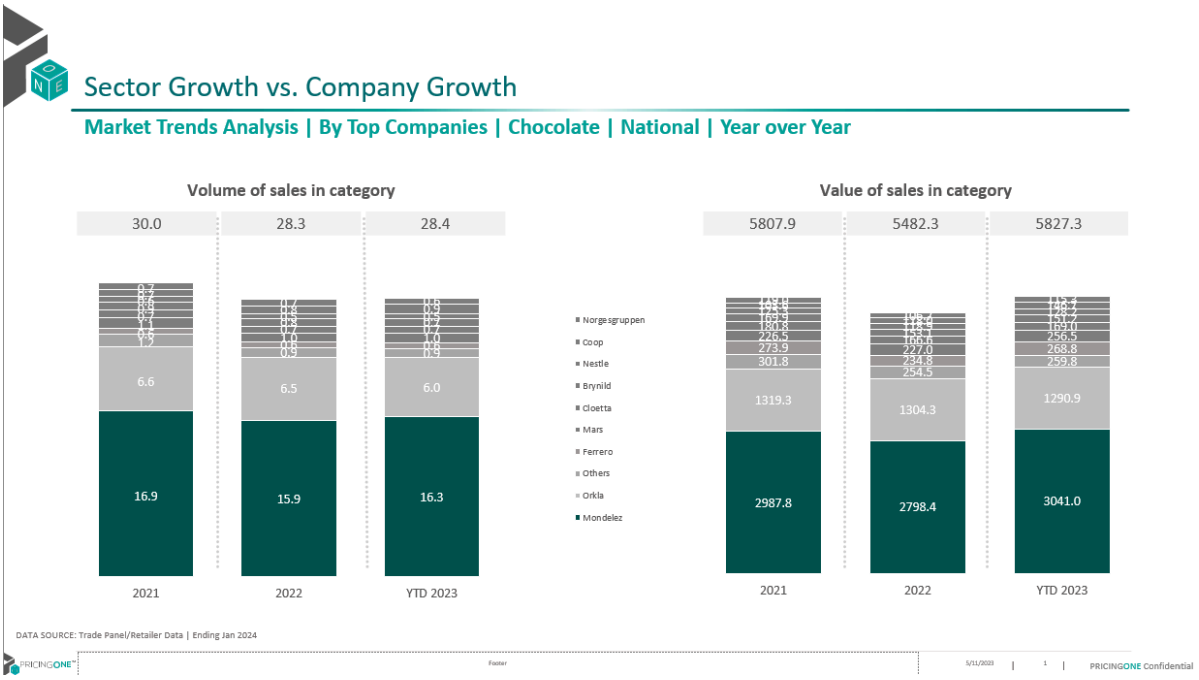
```
p+=1
if len(segments)!=0:
    Market_Trends(prs, list(modified_segment_dfs_new.keys()),
modified_segment_dfs_new, segment_totals_new, segments ,position =
calculate_position(p), slide_by = 'Segment')
    p+=1
```

Step 6: Save Presentation

- performs two main tasks: saving the current PowerPoint presentation to a file and opening that file using the PowerPoint application. The outputPath variable is constructed using the current working directory, ensuring the presentation is saved in the correct location. After saving the presentation, the script uses win32com.client to dispatch the PowerPoint application and open the saved presentation. This automation allows for seamless transition from generating the presentation to viewing or editing it in PowerPoint, streamlining the workflow for creating market analysis slides.

```
# This script saves the generated PowerPoint presentation to a specified path
# and then opens the saved presentation using the PowerPoint application.

# Define the output path for the PowerPoint presentation
outputPath = os.getcwd() + "\\Landscape output.pptx"

# Save the PowerPoint presentation to the specified output path
prs.save(outputPath)

# Initialize the PowerPoint application using win32com client
app = win32.Dispatch("PowerPoint.Application")

# Open the saved PowerPoint presentation
presentation = app.Presentations.Open(outputPath)
```

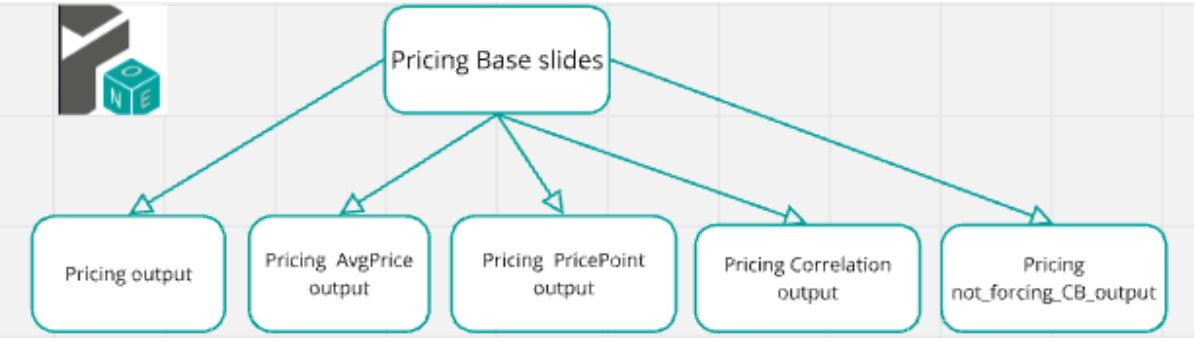
Ex:Market Trends Slide OutPut After Replacement Data



Pricing Section

Introduction

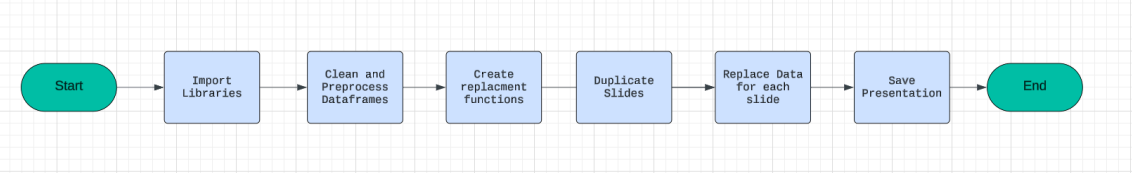
In the slide automation pricing : from12 slide base we create 5 decks



1. Pricing Output Slides:
    - Price Positioning Analysis
    - Share and Growth By Brands(**Leadership Table**)
    - Value Sales Vs Avg Price
2. Pricing Avg&Shelf Price Output Slides:
    - Avg Price/Vol
    - Shelf Price/Vol
3. Pricing Price Point Output Slides :
    - Price Point Distribution Analysis by product
    - Price Point Comparison Analysis by Product
    - Price Point Distribution Analysis by brand
    - Price Point Distribution by brand by Sector
4. Pricing Correlation Output Slides:
    - Price Correlation Analysis P3Y
    - Price Correlation Analysis P12M
5. Pricing not_forcing_CB_Output Slides
    - Price Positioning Analysis
    - Share and Growth By Brands(**Leadership Table**)
    - Price Point Distribution Analysis by brand

Project Steps

- Project Flow



Step 1: Import Libraries we use

Ex: Libraries we use

- This script sets up an environment for working with PowerPoint presentations, data manipulation, filesystem operations, and COM (Component Object Model) object access.
- It imports necessary modules such as 'pptx' for PowerPoint automation, 'win32com' for COM object access and Windows automation, 'pandas' and 'numpy' for data manipulation,
- 'pathlib' for working with filesystem paths, 're' for regular expression operations, and various other modules for general-purpose tasks like file operations and timing functions.
- By importing these modules, the script prepares itself for tasks such as creating or modifying PowerPoint presentations, analyzing data using pandas and numpy, interacting
- with the Windows environment using win32com, and performing filesystem operations using shutil and os. Overall, this script provides a comprehensive setup for automating tasks
- related to PowerPoint presentations and general-purpose Python programming.

```
# Import necessary module for working with PowerPoint presentations
from pptx import Presentation
# Import the win32com.client module, aliasing it as win32 for convenience
import win32com.client as win32
# Import pandas for data manipulation and analysis
import pandas as pd
# Import numpy for numerical computing
import numpy as np
# Import the Path class from pathlib for working with filesystem paths
from pathlib import Path
# Import re for regular expression operations
import re
# Import sys for access to interpreter-related functions
import sys
# Import time for various time-related functions
import time
# Assign win32.constants to a shorter alias win32c for easier access
win32c = win32.constants
# Import shutil for high-level file operations
import shutil
# Import os for operating system dependent functionality
import os
# Import win32com.client again for COM object and functions access
import win32com.client
# Import warnings for warning control functionality
```

```
import warnings
```

Step 2: modified Data frame

EX: input dataframes before cleaning

```
price_positioning_brands['Chocolate | National']
```

0.0s Python

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Category	Chocolate	NaN	NaN	NaN	NaN	NaN	NaN
2	Scope	Category	NaN	NaN	NaN	NaN	NaN	NaN
3	Time Period	P12M	NaN	NaN	NaN	NaN	NaN	NaN
4	Area	NATIONAL	NaN	NaN	NaN	NaN	NaN	NaN
5	Region	All	NaN	NaN	NaN	NaN	NaN	NaN
6	Channel	All	NaN	NaN	NaN	NaN	NaN	NaN
7	Market	All	NaN	NaN	NaN	NaN	NaN	NaN
8	Sector	All	NaN	NaN	NaN	NaN	NaN	NaN
9	Segment	All	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	NaN	Values	NaN	NaN	NaN	NaN	NaN	NaN
12	Top Brands	Relative Price	Av Price/Unit	Value Sales	Value Share	Value Share DYA	Av Price/KG	IYA Price/KG
13	After Eight	0.860127	31.325988	22140300	0.003772	0.000312	177.065739	1.098774
14	All Others	1.704416	30.90972	121895110	0.020767	-0.002114	350.871197	1.063071
15	Anthon Berg	2.11939	32.496994	46119734	0.007857	-0.001605	436.297823	0.902908
16	Bounty	1.003635	17.514835	24206868	0.004124	0.000886	206.608469	1.115473
17	Cadbury	10.708729	74.728814	4409	0.000001	0.000001	2204.5	NaN
18	Cadbury Dairy Milk	1.808386	70.843284	18986	0.000003	0.000003	372.27451	NaN
19	Cloetta	1.981	40.899944	8139048	0.001387	-0.00042	407.808798	1.181783
20	Cloetta Pops	1.119816	47.954082	111663718	0.019024	-0.000246	230.52536	1.007913
21	Coop Private Label	0.751613	26.22216	148965340	0.025379	0.003777	154.727067	0.99715
22	Daim	1.404606	21.599754	60490975	0.010306	0.000112	289.152418	1.01764
23	Fazer	1.448618	29.637093	3311056	0.000564	0.000132	298.212735	1.060132
24	Ferrero Collection	2.452722	165.761864	8093323	0.001379	0.000079	504.917524	1.080959
25	Ferrero Rocher	1.695311	66.937852	19459369	0.003315	-0.000043	348.996897	0.993448

cleaning Code

- This code processes a dictionary of DataFrames, modified_price_positioning_sorted, by performing a series of operations on each DataFrame. Specifically, it iterates over each key in the dictionary, makes a copy of the DataFrame to avoid altering the original, filters out rows where the 'Top Brands' column has the value 'Others', replaces all NaN values with 0, and then updates the dictionary with the modified DataFrame. This ensures that the DataFrames only include data from specified brands and that missing values are handled appropriately.

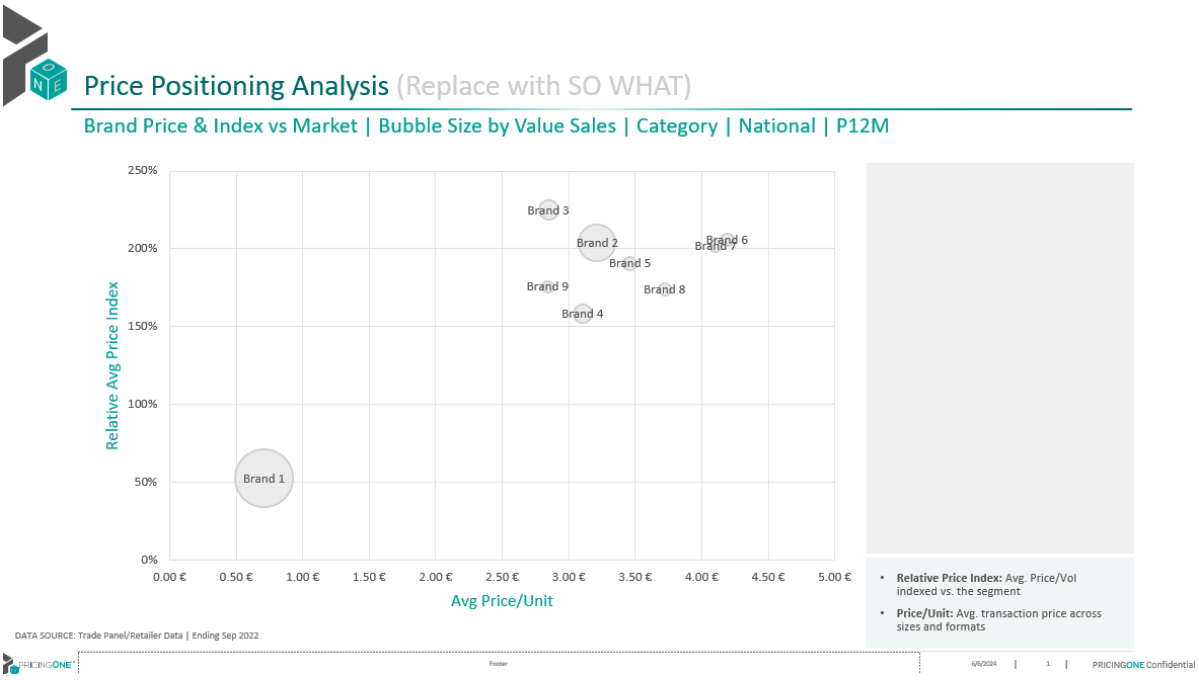
```
# Iterate over each key in the dictionary 'modified_price_positioning_sorted'
for k in modified_price_positioning_sorted.keys():

    # Create a copy of the DataFrame associated with the current key to avoid
modifying the original data
    df = modified_price_positioning_sorted[k].copy()

    # Filter out rows where the 'Top Brands' column has the value 'Others'
    df = df[df['Top Brands'] != 'Others']

    # Replace all NaN values in the DataFrame with 0
    df = df.replace(np.nan, 0)

    # Update the dictionary with the modified DataFrame
```

```
    modified_price_positioning_sorted[k] = df
```

Data frame after cleaning

Step 3: Write Functions to Create Slide

Example slide : Price Positioning Analysis Slide



- PricePositioning Function: generates slides for a PowerPoint presentation, focusing on price positioning analysis with bubble chart visualizations. It iterates through a specified number of slides (numOfDuplicates)
    - parameters :
      prs: PowerPoint presentation object.
      modified_price_positioning_sorted: Dictionary containing sorted price positioning dataframes.
      numOfDuplicates: Number of duplicate slides to generate.
      position: Position index to start adding slides (default is 0).

```
def
pricePositioning(prs,modified_price_positioning_sorted,numOfDuplicates,position=0
):
    for slidenum in range(numOfDuplicates):
        # Extract market and corresponding dataframe
        market=list(modified_price_positioning_sorted.keys())[slidenum]
        df=modified_price_positioning_sorted[market].reset_index(drop=True)
        # Access shapes in the slide
        shapes = prs.slides[slidenum+position].shapes
        charts = []
        tables = []
        title = shapes.title.text
        # Update text boxes in the slide
        shapes[4].text = data_source
        shapes[5].text = 'Brand Price & Index vs Market | Bubble Size by Value
Sales | '+market+' | P12M'
```

```
        shapes[5].text_frame.paragraphs[0].font.bold = True

        for shape in shapes:
            if shape.has_chart:
                shape_id = shape.shape_id
                charts.append(shape)
        chart = charts[0].chart
        charts[0].left = Inches(0.57) # Adjust left position
        chart_name = charts[0].name
        chart_type = chart.chart_type
        # Add bubble chart data
        chart_data = BubbleChartData()
        chart_data.categories = df['Av Price/Unit'].unique().tolist()
        series = chart_data.add_series("Relative Price Index")
        series.has_data_labels = True

        # Add data points to the bubble chart
        for i in range(df.shape[0]):
            series.add_data_point(df['Av Price/Unit'].iloc[i], df['Relative
Price'].iloc[i], df['Value Sales'].iloc[i])
        chart.replace_data(chart_data)

        # Update chart formatting
        xlsx_file=BytesIO()
        with chart_data._workbook_writer._open_worksheet(xlsx_file) as (workbook,
worksheet):
            chart_data._workbook_writer._populate_worksheet(workbook, worksheet)
            worksheet.write(0, 4, "labels")
            worksheet.write_column(1, 4, df['Top Brands'], None)

        chart._workbook.update_from_xlsx_blob(xlsx_file.getvalue())

        category_axis = chart.category_axis
        if sign == 'Before':
            category_axis.tick_labels.number_format = f'{currency}#,##0.00'  if
decimals == 2 else f'{currency}#,##0'
        else:
            category_axis.tick_labels.number_format = f'#,##0.00{currency}'  if
decimals == 2 else f'#,##0{currency}'

        category_axis.auto_axis = True

        value_axis = chart.value_axis
        value_axis.tick_labels.number_format = '0%'
        value_axis.auto_axis = True

        # Customize data labels for each point in the chart
        for i,point in enumerate(chart.series[0].points):
            if df['Top Brands'].iloc[i]=="Others":
                point.format.fill.background()
                point.data_label.text_frame.text=''
                point.format.line.width = Pt(0)

            else:

                data_label = point.data_label
```

```
                data_label.has_text_frame=True
                data_label.text_frame.text=df['Top Brands'].iloc[i]
                data_label.text_frame.paragraphs[0].runs[0].font.size = Pt(10)
                data_label.position = XL_LABEL_POSITION.CENTER
                point.format.fill.solid()
                point.format.fill.fore_color.rgb = RGBColor(245,245,245)
                point.format.line.color.rgb = RGBColor(207,206,206)  # Set the
desired RGB color value
                point.format.line.width = Pt(1)
```

Step 4: Duplicate Slides

- This code is preparing data and configurations for generating a PowerPoint presentation with multiple sections, each requiring a different number of slides based on various price and distribution analyses. It includes:
    - Index List: Specifies the starting slide positions for different sections.
    - Duplication List: Indicates the number of slides to be generated for each section, based on the length of different datasets.
    - Section Names: Provides names for each section in the presentation.
    - Paths: Defines the file paths for the base PowerPoint template and the new duplicated presentation.

```
# Define the index list for slide positions
index = [0, 1, 2, 3, 4, 3, 4, 5, 5, 5, 5, 5, 5, 6, 7, 7]

# Define the duplication list representing the number of slides to be generated
for each section
duplication = [
    len(modified_price_positioning_sorted.keys()),  # Number of price positioning
slides
    len(modified_brands_segments_leadership.keys()),  # Number of segments
leadership slides
    len(modified_brands_sector_leadership.keys()),  # Number of sectors
leadership slides
    len(all_brands_sector.keys()),  # Number of sector avg price/vol comparison
slides
    len(all_brands_sector.keys()),  # Number of sector shelf price/vol comparison
slides
    len(all_brands_segment.keys()),  # Number of segment avg price/vol comparison
slides
    len(all_brands_segment.keys()),  # Number of segment shelf price/vol
comparison slides
    len(sectorP3mPD.keys()),  # Number of category price point distribution
analysis P3M slides
    len(sectorP12mPD.keys()),  # Number of category price point distribution
analysis P12M slides
    len(segmentP3mPD.keys()),  # Number of sector price point distribution
analysis P3M slides
    len(segmentP12mPD.keys()),  # Number of sector price point distribution
analysis P12M slides
    len(sub_segmentP3mPD.keys()),  # Number of segment price point distribution
analysis P3M slides
```

```
    len(sub_segmentP12mPD.keys()),  # Number of segment price point distribution
analysis P12M slides
    len(modified_brandPriceDistribution.keys()),  # Number of price point
distribution analysis by brand slides
    len(modified_sectorsPriceDistribution.keys()),  # Number of price point
distribution by brand by sector slides
    len(modified_segmentPriceDistribution.keys())  # Number of price point
distribution by brand by segment slides
]

# Define the section names to be used in the presentation
section_names = [
    "Price Positioning Analysis",
    "Segments Leadership Analysis",
    "Sectors Leadership Analysis",
    "Sector Avg Price/Vol Comparison",
    "Sector Shelf Price/Vol Comparison",
    "Segment Avg Price/Vol Comparison",
    "Segment Shelf Price/Vol Comparison",
    "Category Price Point Distribution Analysis P3M",
    "Category Price Point Distribution Analysis P12M",
    "Sector Price Point Distribution Analysis P3M",
    "Sector Price Point Distribution Analysis P12M",
    "Segment Price Point Distribution Analysis P3M",
    "Segment Price Point Distribution Analysis P12M",
    "Price Point Distribution Analysis By Brand",
    "Price Point Distribution By Brand By Sector",
    "Price Point Distribution By Brand By Segment"
]

# Define paths for the base PowerPoint template and the new duplicated
presentation
path = os.getcwd() + '\Pricing slide base.pptx'
new_pre = os.getcwd() + '\Pricing duplicated.pptx'
```

Duplication Function

We use the duplication function to duplicate slides by number of the duplicate and save it in the duplication deck to use it to replace data.
Duplicate slides in a PowerPoint presentation.

Parameters:
- index (list): List of slide indices to duplicate.
- duplication (list): List specifying the number of times each slide should be duplicated.
- section_names (list): List of names for sections to be added.
- path (str): Path to the PowerPoint presentation file.
- new_pre (str): Path to save the duplicated presentation.

Returns:
- str: A message indicating success or failure.

```
####New_With_duplicate
import pythoncom
defslideDuplication(index=[0,1],duplication=[1,1],section_names=
[''],path='',new_pre=''):
```

```
    lis=[]
    iflen(index)!=len(duplication)!=len(section_names):
        return'The Index list not equal the Duplication number list in length'
    app = win32.Dispatch("PowerPoint.Application")
    presentation = app.Presentations.Open(path)
    # Iterate through the slides in the original presentation and copy them to
the new presentation
    for i inrange(len(index)):
        iftype(index[i])==list:
            # If index is a list of slide indices
            for num_duplicate inrange(duplication[i]):
                for k in index[i]:
                    slide=presentation.Slides[k]
                    duplicated_slide = slide.Duplicate()
                    duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-
(duplication[i]*len(index[i])))
        else:
            # If index is a single slide index
            slide=presentation.Slides[index[i]]
            for num_duplicate inrange(duplication[i]):
                duplicated_slide = slide.Duplicate()
                duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-duplication[i])
    # Add sections to the new presentation
    for j inrange(len(lis)):
        if duplication[j]!=0:
            presentation.SectionProperties.AddBeforeSlide(lis[j],section_names[j]
)
    # presentation.ApplyTheme(themePath)
    presentation.SectionProperties.Delete(1, True)
    presentation.SaveAs(new_pre)
    presentation.Close()
    # Close the original presentation and PowerPoint application
    app.Quit()
```

Step 5: Replace Data in Slide

- This part of the code calls the pricePositioning function to generate slides for the "Price Positioning Analysis" section of the presentation. It uses the prs PowerPoint presentation object, the dictionary modified_price_positioning_sorted containing the sorted price positioning dataframes, and the first element of the duplication list to determine the number of slides to generate. The position variable is set to 0, indicating that the slides should be added starting from the first position.

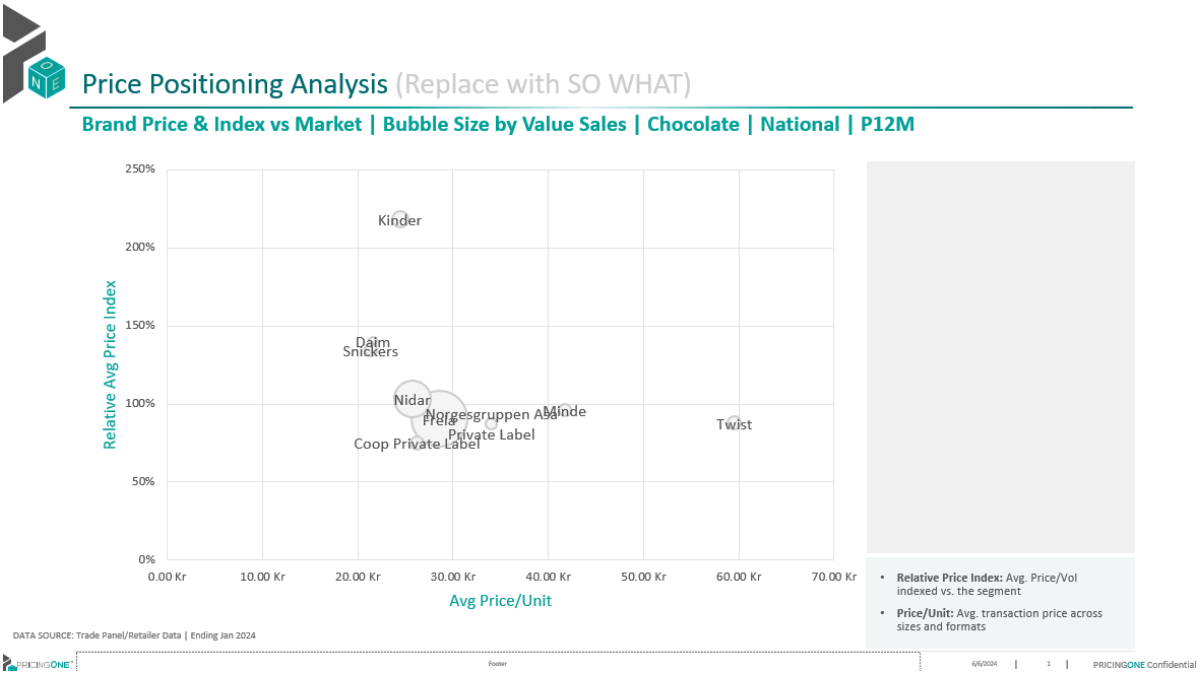
```
# Set the initial position for slide insertion to 0
position = 0

# Call the pricePositioning function to generate slides for price positioning
analysis
pricePositioning(prs, modified_price_positioning_sorted, duplication[0],
position)
```

Step 6: Save Presentation

- This code is responsible for finalizing the creation of a PowerPoint presentation by saving it to a specified file path and then opening it using Microsoft PowerPoint. Initially, it defines the output path for the new PowerPoint presentation by combining the current working directory with the filename Pricing output.pptx. The script then saves the modified presentation (prs object) to this specified path. After saving, it uses the win32com.client.Dispatch method to create an instance of the PowerPoint application, and then it opens the saved presentation within this application. This process ensures that the newly created presentation is both saved and immediately available for viewing or further editing in Microsoft PowerPoint.

```
# Define the output path for the new PowerPoint presentation
outputPath = os.getcwd() + "\\Pricing output.pptx"

# Save the modified presentation to the specified output path
prs.save(outputPath)

# Open the saved PowerPoint presentation using the PowerPoint application
app = win32.Dispatch("PowerPoint.Application")
presentation = app.Presentations.Open(outputPath)
```

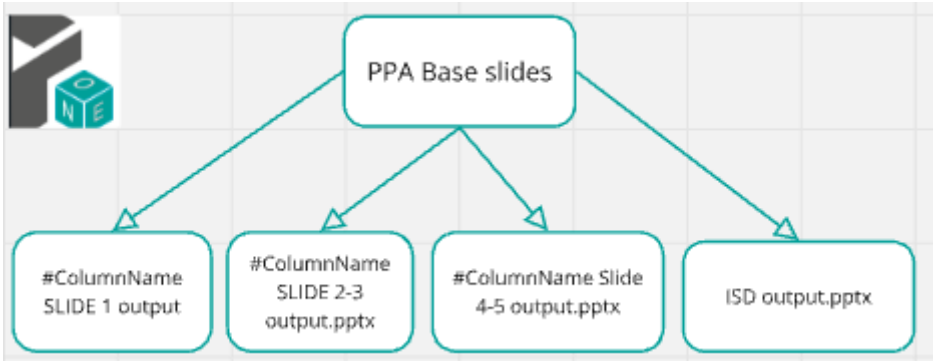
Example: OutPut Slide After Replacement Data "PricePositioning Slide OutPut"



PPA Section

Introduction

In the slide automation PPA: from 5 slide base we create 4 decks

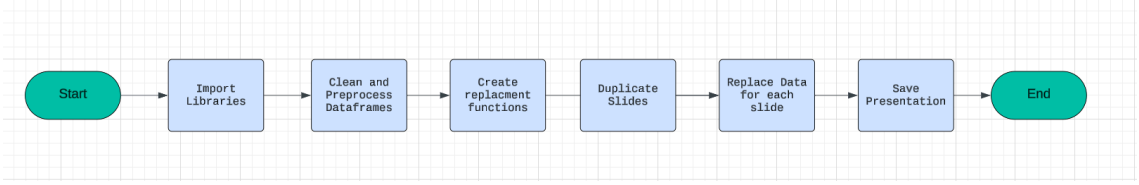


Ex:columnName = ['Base Price Bracket',"Size Bracket","Portion Count Bracket"]

1. #ColumnName SLIDE 1output Slides:
    - Brand Share Topline
2. #ColumnName SLIDE 2-3output Slides:
    - #ColumnName by Sector/Segment
3. #ColumnName Slide4-5 output Slides:
    - Brackets Analysis By Sector
    - BracketsAnalysis By Segment
4. ISD output Slides :
    - Inter-size
      Discount Analysis

Project Steps

- Project Flow



- Step 1: Import Libraries we use
- Step 2: modified Data frames: cleaning and preprocessing the data frames
- Step 3: Write Functions to Create Slides: Define functions to dynamically generate slides based on the base slides
- Step 4: Duplicate Slides: Use functions or methods to duplicate existing slides as needed for the presentation.
- Step 5: Replace Data in Slide: update information from the cleaned data frames to slides
- Step 6: Save Presentation

Step 1: Import Libraries we use

Ex: Libraries we use

- This script sets up an environment for working with PowerPoint presentations, data manipulation, filesystem operations, and COM (Component Object Model) object access.
- It imports necessary modules such as 'pptx' for PowerPoint automation, 'win32com' for COM object access and Windows automation, 'pandas' and 'numpy' for data manipulation,
- 'pathlib' for working with filesystem paths, 're' for regular expression operations, and various other modules for general-purpose tasks like file operations and timing functions.
- By importing these modules, the script prepares itself for tasks such as creating or modifying PowerPoint presentations, analyzing data using pandas and numpy, interacting
- with the Windows environment using win32com, and performing filesystem operations using shutil and os. Overall, this script provides a comprehensive setup for automating tasks
- related to PowerPoint presentations and general-purpose Python programming.

```
# Import necessary module for working with PowerPoint presentations
from pptx import Presentation
# Import the win32com.client module, aliasing it as win32 for convenience
import win32com.client as win32
# Import pandas for data manipulation and analysis
import pandas as pd
# Import numpy for numerical computing
import numpy as np
# Import the Path class from pathlib for working with filesystem paths
from pathlib import Path
# Import re for regular expression operations
import re
# Import sys for access to interpreter-related functions
import sys
# Import time for various time-related functions
import time
# Assign win32.constants to a shorter alias win32c for easier access
win32c = win32.constants
# Import shutil for high-level file operations
import shutil
# Import os for operating system dependent functionality
import os
# Import win32com.client again for COM object and functions access
import win32com.client
# Import warnings for warning control functionality
import warnings
```

Step 2: modified Data frame

EX: input dataframes before cleaning

```
1  share_topline_base_price_bracket['Chocolate | National']
✓ 0.0s
```

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
0	NaN	NaN	NaN	NaN	NaN	NaN
1	Category	Chocolate	NaN	NaN	NaN	NaN
2	Time Period	P12M	NaN	NaN	NaN	NaN
3	Scope	Category	NaN	NaN	NaN	NaN
4	Area	NATIONAL	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
213	NaN	Smarties	0.001054	0.138387	0.792435	2.147643
214	NaN	Toffifee	0.00016	0.045473	420.474227	1.328147
215	NaN	Twist	0.000074	0.002603	NaN	1.166107
216	110+ kr Total	NaN	0.030481	NaN	0.969802	2.044675
217	Grand Total	NaN	1	NaN	1.072379	1

218 rows × 6 columns

Cleaning Data Frame

- This code is part of a data processing pipeline for cleaning and preparing bracket-related data. It selects the appropriate DataFrame based on columnName, processes elements in a brackets list, and iterates over ppaDf keys to rename columns, remove rows, forward-fill missing values, replace NaNs, and sort by 'Value Share'. Cleaned DataFrames are stored in ToplineBracket. The script methodically handles data preparation, crucial for accurate analysis and reporting, but the use of brackets and commented-out lines suggest the code may still be in development or require additional context.

```
def process_bracket_data(dic,col="Base Price Bracket",bymanuf=False):
    ToplineBracket_brand = {}
    for key in dic.keys():
        df = dic[key].copy()
        df=DetectHeader(df)
        df=df[:-1]
        if col== "Base Price Bracket":
            df= df.rename(columns={"Base Price\xa0Bracket":"Base Price Bracket"})
        df[col] = df[col].ffill()
        if bymanuf:
            df[["Value Share"   ,"Company WoB %"    ,"Value Sales IYA"
,"Relative Price"]] = df[["Value Share" ,"Company WoB %"    ,"Value Sales IYA"
,"Relative Price"]].replace(np.nan, 0).astype(float)
            if ManufOrTopC == "Manufacturer": df= df.rename(columns=
{"Manufacturer":"Top Companies"})
        else:
            df[["Value Share"   ,"Brand WoB %"  ,"Value Sales IYA"  ,"Relative
Price"]] = df[["Value Share" ,"Brand WoB %"  ,"Value Sales IYA"  ,"Relative
Price"]].replace(np.nan, 0).astype(float)
            if BrandOrTopB == "Brand": df= df.rename(columns={"Brand":"Top
Brands"})
        df = df.sort_values('Value Share', ascending=False)
        df=df[~(df[col].str.contains('0-0'))]
```

```
        if df.shape[0]==0:
            print(key)
        else:
            ToplineBracket_brand[key] = df

    return ToplineBracket_brand
```

Data Frame After cleaning

```
1  ToplineBracket['Chocolate | National']
```

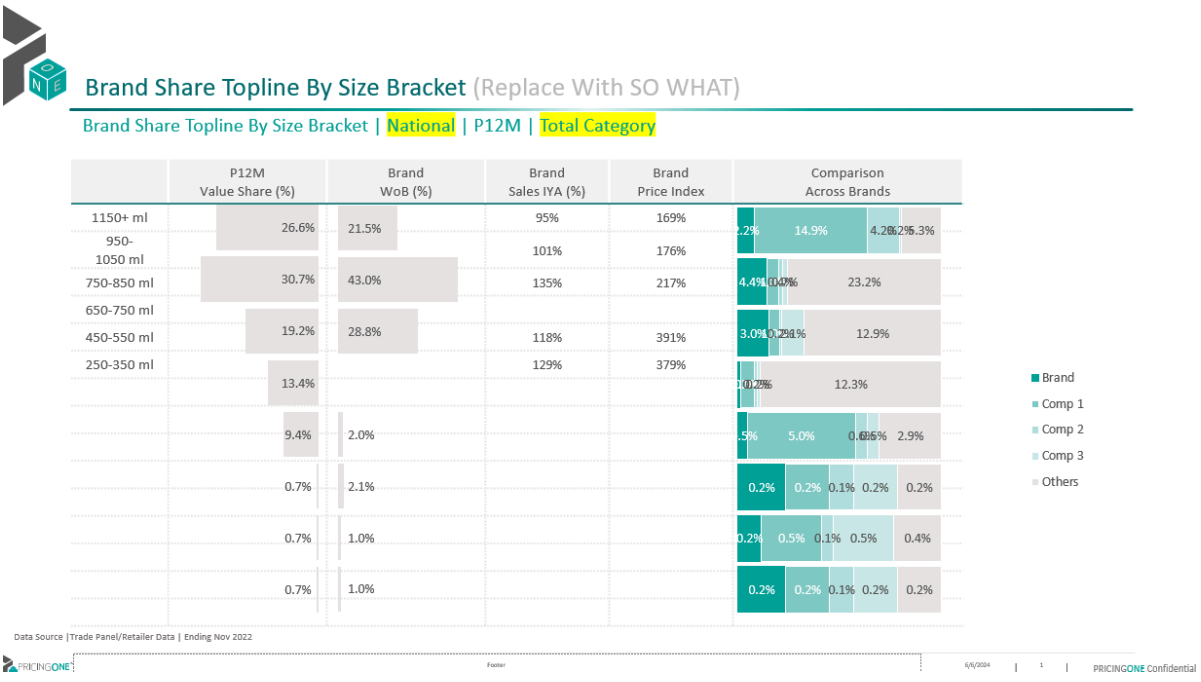
✓ 0.0s

12	Base Price Bracket	Top Brands	Value Share	Brand WoB %	Value Sales IYA	Relative Price
114	40-50 kr Total	NaN	3.739741e-01	0.000000	1.870917	0.866969
98	40-50 kr	Freia	2.583914e-01	0.533148	1.943682	0.803773
90	30-40 kr Total	NaN	1.547746e-01	0.000000	0.485190	0.932589
67	20-30 kr Total	NaN	1.508749e-01	0.000000	2.423107	1.203051
39	10-20 kr Total	NaN	1.304607e-01	0.000000	0.666329	1.293478
...	...	...	...	...	...	...
74	30-40 kr	Fazer	1.294816e-08	0.000000	0.003721	0.000000
127	50-60 kr	Minde Banansjokolade	8.688898e-09	0.000000	0.000000	0.000000
71	30-40 kr	Cloetta Pops	5.451857e-09	0.000000	0.329897	0.000000
58	20-30 kr	Plamil	4.088893e-09	0.000000	0.000000	0.000000
55	20-30 kr	Minde Banansjokolade	2.896299e-09	0.000000	0.000000	0.000000

204 rows × 6 columns

Step 3: Write Functions to Create Slides

Example slide : Brand Share Topline By Size Bracket



- brandShareToplinfunction:Generate a PowerPoint slide presentation with data visualizations and tables showing brand share topline metrics by various brackets for a specified number of slides.

- Parameters:
  prs (pptx.presentation.Presentation): The PowerPoint presentation object where slides will be added or modified.
  modifiedShareToplineBracket (dict): Dictionary containing data frames for different markets.
  bracketsValue (list): List of size brackets to be considered.
  clientElement (str): Name of the client brand to highlight in the presentation.

```
def brandShareTopline(prs, modifiedShareToplineBracket, bracketsValue,
numOfDuplicates, slide_by, clientElement, position=0):
    for slidenum in range(numOfDuplicates):
        # Get the market name and its corresponding data frame
        market = list(modifiedShareToplineBracket.keys())[slidenum]
        df = modifiedShareToplineBracket[market].copy()
        # Extract and sort 'Size' from the slide_by column
        df['Size'] = df[f"{slide_by}"].apply(lambda x: x.split('-')[1].split(' ')
[0] if '-' in x else 9999).astype(float)
        df = df.sort_values(by=['Size'], ascending=False)
        # Filter total brand data and clean up column values
        dfTotalBrand = df[df[f"{slide_by}"].str.contains('Total')]
        dfTotalBrand[f"{slide_by}"] = dfTotalBrand[f"{slide_by}"].str.replace('
Total', '')
        dfTotalBrand = dfTotalBrand[dfTotalBrand['Value Share'] > .01]

        # Filter the main data frame based on size
        df =
df[df['Size'].isin(dfTotalBrand['Size'].unique())].sort_values(by='Value Share',
ascending=False)

        # Get the top 3 brands excluding the client element
        dfTopSales = df[(df['Top Brands'].notna()) & (df['Top Brands'] !=
clientElement)].drop_duplicates(subset='Top Brands')['Top
Brands'].iloc[:3].to_list()
        dfBrandInScope = df[df['Top Brands'].isin(dfTopSales)]

        # Calculate the 'Other' category for the data frame
        dfOther = df[(~df['Top Brands'].isin(dfTopSales + [clientElement])) &
(~df[f"{slide_by}"].str.contains('Total'))].groupby([f"{slide_by}", 'Size'])
['Value Share'].sum().reset_index().sort_values(by='Size', ascending=False)
        missingOtherBracket = list(set(bracketsValue) - set(dfTotalBrand[f"
{slide_by}"].unique()))
        missingOtherBracket = pd.DataFrame({f"{slide_by}": missingOtherBracket,
'Size': [float(x.split('-')[1].split(' ')[0]) if '-' in x else 9999 for x in
missingOtherBracket]})
        dfOther = pd.concat([dfOther,
missingOtherBracket]).sort_values(by='Size', ascending=False)
        dfTotalBrand = pd.concat([dfTotalBrand,
missingOtherBracket]).sort_values(by='Size', ascending=False)

        # Filter the client's brand data
        dfClientBrand = df[df['Top Brands'] == clientElement]

        # Access slide shapes to update text and formatting
        shapes = prs.slides[slidenum + position].shapes
        shapes[4].text = data_source
```

```
        shapes[5].text = f'Brand Share Topline By {slide_by} | {market} | P12M'
        # Format text as bold and set font size
        shapes[5].text_frame.paragraphs[0].font.bold = True
        for p in range(len(shapes[5].text_frame.paragraphs)):
            shapes[5].text_frame.paragraphs[p].font.size = Pt(12)
        shapes[6].text_frame.paragraphs[0].runs[0].text =
shapes[6].text_frame.paragraphs[0].runs[0].text.replace('Size Bracket', slide_by)
        shapes[6].text_frame.paragraphs[0].font.size = Pt(16)
        # Create tables and charts
        tables, charts = createTableAndChart(shapes)
        # Adjust table row numbers
        table = tables[0].table
        num_rows_to_remove = len(table.rows) - dfTotalBrand[f"
{slide_by}"].nunique() - 1
        for _ in range(num_rows_to_remove):
            if len(table.rows) > 1:  # Skip removing the first row if there is
more than one row
                row = table.rows[1]
                remove_row(table, row)
        # Set table row height
        table_height = Inches(3.81)  # Specify the desired table height
        total_row_height = table_height - table.rows[0].height
        num_rows = len(table.rows) - 1  # Exclude the first row
        if num_rows > 0:
            cell_height = total_row_height / num_rows
            for row in range(1, len(table.rows)):
                table.rows[row].height = int(cell_height)

        # Replace the table data
        for i, row in enumerate(table.rows):
            for j, cell in enumerate(row.cells):
                if i == 0:
                    # Update header cells
                    if j in [2, 3, 4]:
                        cell.text = cell.text.replace('Brand', clientElement)
                        for paragraph in cell.text_frame.paragraphs:
                            paragraph.font.name = 'Nexa Bold'
                            paragraph.font.size = Pt(9)
                            paragraph.alignment = PP_ALIGN.CENTER
                            paragraph.font.color.rgb = RGBColor(87, 85, 85)
                            paragraph.font.bold = False
                        continue
                # Update data cells
                sizeBracket = dfTotalBrand[f"{slide_by}"].unique()[i - 1]
                if j == 0:
                    cell.text = sizeBracket
                    cell.text_frame.paragraphs[0].font.name = 'Nexa Bold'
                    cell.text_frame.paragraphs[0].font.size = Pt(9)
                    cell.text_frame.paragraphs[0].alignment = PP_ALIGN.CENTER
                if j == 3 or j == 4:
                    if j == 3:
                        value = dfClientBrand[dfClientBrand[f"{slide_by}"] ==
sizeBracket]['Value Sales IYA'].unique()
                        # Exclude Brand 'Brand WoB %' < 5%
                        if value and dfClientBrand[dfClientBrand[f"{slide_by}"]
== sizeBracket]['Brand WoB %'].unique()[0] < .0005:
```

```
                            value = [0]
                        cell.text = '' if (len(value) == 0) or
(int(round(float(value[0]) * 100, 0)) == 0) else (str(int(round(float(value[0]) *
100, 0))) + '%' if int(round(float(value[0]) * 100, 0)) <= 1000 else 'Large')
                    else:
                        value = dfClientBrand[dfClientBrand[f"{slide_by}"] ==
sizeBracket]['Relative Price'].unique()
                        # Exclude Brand 'Brand WoB %' < 5%
                        if value and dfClientBrand[dfClientBrand[f"{slide_by}"]
== sizeBracket]['Brand WoB %'].unique()[0] < .0005:
                            value = [0]
                        cell.text = '' if len(value) == 0 or
(int(round(float(value[0]) * 100, 0)) == 0) else str(int(round(float(value[0]) *
100, 0))) + '%'
                    cell.text_frame.paragraphs[0].font.name = 'Nexa Book'
                    cell.text_frame.paragraphs[0].font.size = Pt(8)
                    cell.text_frame.paragraphs[0].alignment = PP_ALIGN.CENTER
        # Update chart data
        for chartNum in [0, 1]:
            chart = charts[chartNum].chart
            chart_data = CategoryChartData()
            chart_data.categories = ['']
            if chartNum == 0:
                missingBrandBracket = list(set(dfTotalBrand[f"
{slide_by}"].unique()) - set(dfClientBrand[f"{slide_by}"].unique()))
                missingBrandBracket = pd.DataFrame({'Top Brands': clientElement,
f"{slide_by}": missingBrandBracket, 'Size': [float(x.split('-')[1].split(' ')[0])
if '-' in x else 9999 for x in missingBrandBracket]})
                dfClientBrand2 = pd.concat([dfClientBrand,
missingBrandBracket]).sort_values(by='Size', ascending=False).replace(np.nan,
None)
                # Exclude Value Share less than 5%
                dfClientBrand2['Brand WoB %'] = np.where(dfClientBrand2['Brand
WoB %'] < .0005, None, dfClientBrand2['Brand WoB %'])
                brandWob = dfClientBrand2['Brand WoB %'].to_list()
                chart_data.add_series('Brand WoB %', brandWob)
            else:
                valueShare = dfTotalBrand['Value Share'].replace(np.nan,
None).to_list()
                chart_data.add_series('Value Share', valueShare)
            chart.replace_data(chart_data)
        # Update the comparison chart
        chart2 = charts[2].chart
        chart_data2 = CategoryChartData()
        chart_data2.categories = dfTotalBrand[f"{slide_by}"].unique()
        missingBrandBracket = list(set(dfTotalBrand[f"{slide_by}"].unique()) -
set(dfClientBrand[dfClientBrand['Top Brands'] == clientElement][f"
{slide_by}"].unique()))
        missingBrandBracket = pd.DataFrame({'Top Brands': clientElement, f"
{slide_by}": missingBrandBracket, 'Size': [float(x.split('-')[1].split(' ')[0])
if '-' in x else 9999 for x in missingBrandBracket]})
        dfClientBrand2 = pd.concat([dfClientBrand[dfClientBrand['Top Brands'] ==
clientElement], missingBrandBracket]).sort_values(by='Size', ascending=False)
        valueShare = dfClientBrand2['Value Share'].replace(np.nan,
None).to_list()
        chart_data2.add_series(clientElement, valueShare)
```

```
        for brand in dfBrandInScope['Top Brands'].unique():
            missingBrandBracket = list(set(dfTotalBrand[f"{slide_by}"].unique())
- set(dfBrandInScope[dfBrandInScope['Top Brands'] == brand][f"
{slide_by}"].unique()))
            missingBrandBracket = pd.DataFrame({'Top Brands': brand, f"
{slide_by}": missingBrandBracket, 'Size': [float(x.split('-')[1].split(' ')[0])
if '-' in x else 9999 for x in missingBrandBracket]})
            dfClientBrand2 = pd.concat([dfBrandInScope[dfBrandInScope['Top
Brands'] == brand], missingBrandBracket]).sort_values(by='Size', ascending=False)
            valueShare = dfClientBrand2['Value Share'].replace(np.nan,
None).to_list()
            chart_data2.add_series(brand, valueShare)
        valueShare = dfOther['Value Share'].replace(np.nan, None).to_list()
        chart_data2.add_series('Others', valueShare)

        chart2.replace_data(chart_data2)
```

Step 4: Duplicate Slides

Duplicate Slides: this part of code calculate duplication values, and define section names for generating or updating a PowerPoint presentation.

```
# Generate a list of indices based on various category, sector, segment, and
channel indices
if runSlide1:
    for num, col in enumerate(columnName):

        #Cleaning
        ppaDf_brand =eval(f"share_topline_{col}".replace(" ","_").lower())
        ppaDf_manuf =eval(f"share_topline_{col}".replace("
","_").lower()+"_manuf")
        ToplineBracket_brand=process_bracket_data(ppaDf_brand, col)
        ToplineBracket_manuf=process_bracket_data(ppaDf_manuf, col,bymanuf=True)
        bracketsValue_brand= list(set([value.replace(' Total','') for val in
ToplineBracket_brand.values() if col in val.columns for value in
val[col].unique() if 'Total' in value]))
        bracketsValue_manuf= list(set([value.replace(' Total','') for val in
ToplineBracket_manuf.values() if col in val.columns for value in
val[col].unique() if 'Total' in value]))

        #Dupli
        index1=[0]
        duplication1 =
[len(ToplineBracket_manuf.keys())*len(client_manuf)+len(ToplineBracket_brand.keys
())*len(client_brands)]
        section_names_slide1 = ["Brand Share Topline By "+col]
        duplication1 = [item for item in duplication1 if item !=0]
        section_names1 = section_names_slide1
        new_pre1 = os.getcwd() + '\\PPA slide 1 duplicate.pptx'
        if num==0:
slideDuplication(index1,duplication1,section_names1,path,new_pre1)

        #Check
        print(index1)
```

```
        print(len(index1))
        print(duplication1)
        print(len(duplication1))
        print(section_names1)
        print(len(section_names1))
```

Duplication Function

We use the duplication function to duplicate slides by number of the duplicate and save it in the duplication deck to use it to replace data.
Duplicate slides in a PowerPoint presentation.

Parameters:

- index (list): List of slide indices to duplicate.
- duplication (list): List specifying the number of times each slide should be duplicated.
- section_names (list): List of names for sections to be added.
- path (str): Path to the PowerPoint presentation file.
- new_pre (str): Path to save the duplicated presentation.

Returns:

- str: A message indicating success or failure.

```
####New_With_duplicate
import pythoncom
defslideDuplication(index=[0,1],duplication=[1,1],section_names=
[''],path='',new_pre=''):
    lis=[]
    iflen(index)!=len(duplication)!=len(section_names):
        return'The Index list not equal the Duplication number list in length'
    app = win32.Dispatch("PowerPoint.Application")
    presentation = app.Presentations.Open(path)
    # Iterate through the slides in the original presentation and copy them to
the new presentation
    for i inrange(len(index)):
        iftype(index[i])==list:
            # If index is a list of slide indices
            for num_duplicate inrange(duplication[i]):
                for k in index[i]:
                    slide=presentation.Slides[k]
                    duplicated_slide = slide.Duplicate()
                    duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-
(duplication[i]*len(index[i])))
        else:
            # If index is a single slide index
            slide=presentation.Slides[index[i]]
            for num_duplicate inrange(duplication[i]):
                duplicated_slide = slide.Duplicate()
                duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-duplication[i])
    # Add sections to the new presentation
    for j inrange(len(lis)):
        if duplication[j]!=0:
            presentation.SectionProperties.AddBeforeSlide(lis[j],section_names[j]
)
```

```
    # presentation.ApplyTheme(themePath)
    presentation.SectionProperties.Delete(1, True)
    presentation.SaveAs(new_pre)
    presentation.Close()
    # Close the original presentation and PowerPoint application
    app.Quit()
```

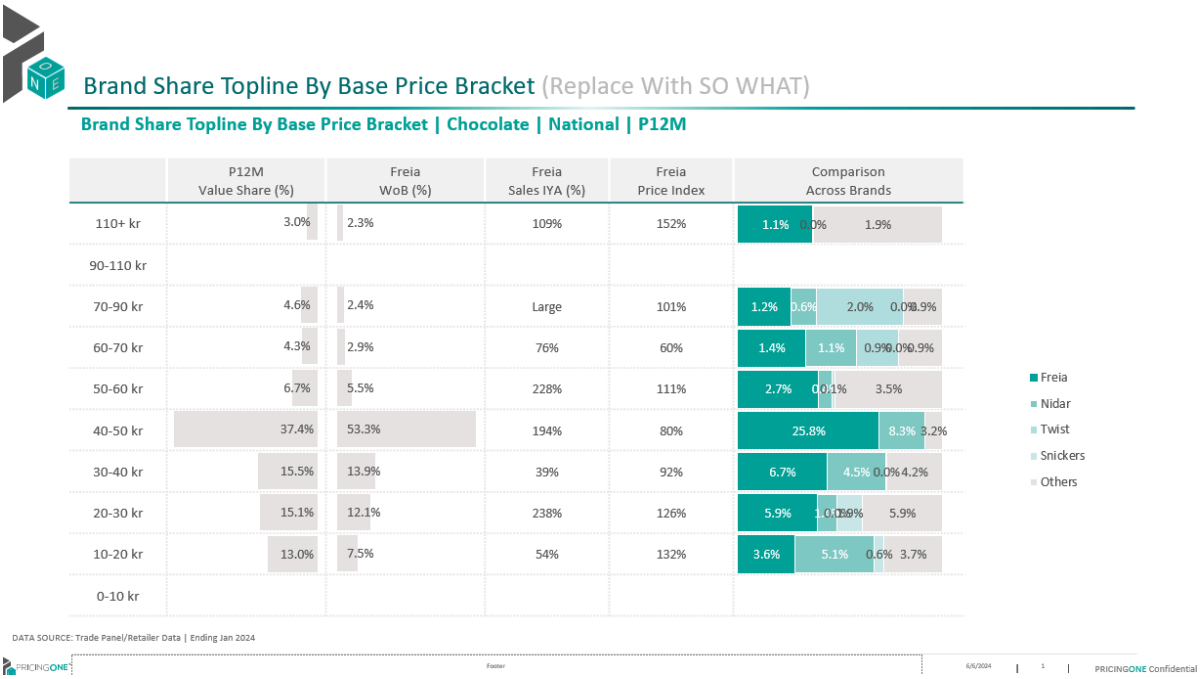
Step 5: Replace Data in Slide

```
if runSlide1:
        #Filling
        prs1 = Presentation(new_pre1)
        sectionPosition = 0
        position=0
        numOfDuplicates=duplication1[sectionPosition]
        for i ,clientElement in enumerate(client_manuf):

 brandShareTopline(prs1,ToplineBracket_manuf,bracketsValue_manuf,int(numOfDuplica
tes/(len(client_manuf)+len(client_brands))),slide_by = col,clientElement=
clientElement,position= position)
            position=int(position+
(numOfDuplicates/(len(client_manuf)+len(client_brands))))
        for i,clientElement in enumerate(client_brands):

 brandShareTopline(prs1,ToplineBracket_brand,bracketsValue_brand,int(numOfDuplica
tes/(len(client_manuf)+len(client_brands))),slide_by = col,clientElement=
clientElement,position= position)
            position=int(position+
(numOfDuplicates/(len(client_manuf)+len(client_brands))))
        outputPath = os.getcwd()+'\\'+col+" SLIDE 1 output.pptx"
        prs1.save(outputPath)

        #Section Renaming
        RenameSections(outputPath,oldname=columnName[0],newname=col)
```

Step 6: Save Presentation

```
# Define the path for saving the output presentation
# This combines the current working directory with the column name and
"output.pptx"
outputPath = os.getcwd()+'\\'+col+" SLIDE 1 output.pptx"
# Save the PowerPoint presentation to the specified output path
prs.save(outputPath)
# Use the win32 library to open the saved PowerPoint presentation
# This dispatches the PowerPoint application and opens the presentation
app = win32.Dispatch("PowerPoint.Application")
presentation = app.Presentations.Open(outputPath)
```

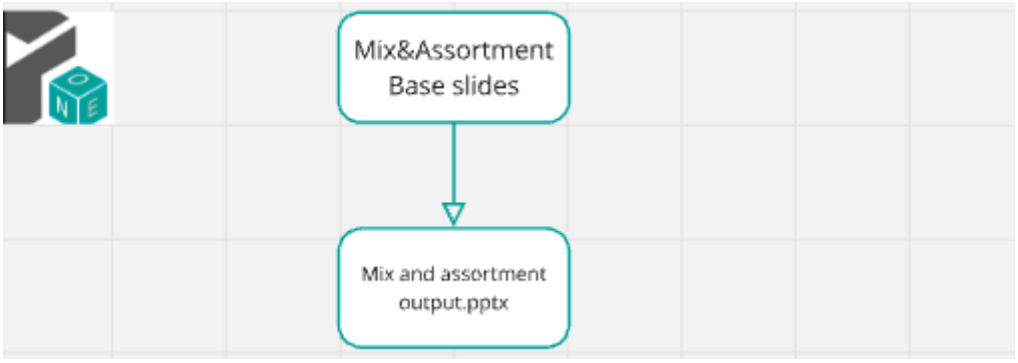
Example: OutPut Slide After Replacement Data "Brand Share Topline By Size Bracket Slide OutPut"



Mix&Assortment Section

Introduction

In slide automation Mix&Assortment: using 8 slide base we create 1 deck



1-Mix and assortment output Slides:

- SKU Share By Brand
- Cumulative Product Share
- Top 50% cumulative share
- Brand Cumulative Product Share
- Top 20 cumulative share
- SKU Productivity Analysis with TM%
- SKU Productivity Analysis with WD
- Sectors Fair Share

Project Steps

- Project Flow
  
- Step 1: Import Libraries we use
- Step 2: modified Data frames: cleaning and preprocessing the data frames
- Step 3: Write Functions to Create Slides: Define functions to dynamically generate slides based on the base slides
- Step 4: Duplicate Slides: Use functions or methods to duplicate existing slides as needed for the presentation.
- Step 5: Replace Data in Slide: update information from the cleaned data frames to slides
- Step 6: Save Presentation

Step 1: Import Libraries we use

Ex: Libraries we use

- This script sets up an environment for working with PowerPoint presentations, data manipulation, filesystem operations, and COM (Component Object Model) object access.
- It imports necessary modules such as 'pptx' for PowerPoint automation, 'win32com' for COM object access and Windows automation, 'pandas' and 'numpy' for data manipulation,
- 'pathlib' for working with filesystem paths, 're' for regular expression operations, and various other modules for general-purpose tasks like file operations and timing functions.
- By importing these modules, the script prepares itself for tasks such as creating or modifying PowerPoint presentations, analyzing data using pandas and numpy, interacting
- with the Windows environment using win32com, and performing filesystem operations using shutil and os. Overall, this script provides a comprehensive setup for automating tasks
- related to PowerPoint presentations and general-purpose Python programming.

```
# Import necessary module for working with PowerPoint presentations
from pptx import Presentation
# Import the win32com.client module, aliasing it as win32 for convenience
import win32com.client as win32
# Import pandas for data manipulation and analysis
import pandas as pd
# Import numpy for numerical computing
import numpy as np
# Import the Path class from pathlib for working with filesystem paths
from pathlib import Path
# Import re for regular expression operations
import re
# Import sys for access to interpreter-related functions
import sys
# Import time for various time-related functions
import time
# Assign win32.constants to a shorter alias win32c for easier access
win32c = win32.constants
# Import shutil for high-level file operations
import shutil
# Import os for operating system dependent functionality
import os
# Import win32com.client again for COM object and functions access
import win32com.client
# Import warnings for warning control functionality
import warnings
```

Step 2: modified Data frames

EX: input dataframes before cleaning

```
1 assortment['Chocolate | National']
✓ 0.0s
```

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Category	Chocolate	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Scope	Category	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	MonthYear	(Multiple Items)	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Area	NATIONAL	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
1153	NaN	Mf Twix Xtra Salted Caramel 75 Gr	0.000618	0.002514	0.42	0.011352	86324.452381	NaN	NaN
1154	NaN	Mf Twix Xtra White 75 Gr	0.0	0	0	0.011927	NaN	NaN	NaN
1155	NaN	Twix 12-Pk 600 Gr	0.00001	0	0	0.011918	NaN	NaN	NaN
1156	Twix Total	NaN	0.011927	0.010356	0.99	NaN	NaN	NaN	NaN
1157	Grand Total	NaN	1	1	0.99	NaN	NaN	NaN	NaN

1158 rows × 9 columns

---

cleaning Code

- mixAssortmentCleaning function: Cleans and processes assortment and cumulative share data to provide modified data for brand-specific analysis.
    - Args:
      assortment (dict): Dictionary containing assortment data.
      cumulativeShare (dict): Dictionary containing cumulative share data.
    - Returns:
      tuple:
        - assortmentModifiedBrand (dict): Dictionary containing cleaned and modified assortment data for brands.
        - assortmentModifiedTotal (dict): Dictionary containing cleaned and modified total assortment data.
        - assortmentClient (dict): Dictionary containing client-specific cleaned and modified assortment data.

```
def mixAssortmentCleaning(assortment, cumulativeShare):
    # Initialize dictionaries to store modified data
    cumulativeShareModifiedBrand = {}
    assortmentModified = {}
    assortmentModifiedBrand = {}
    assortmentModifiedTotal = {}
    assortmentClient = {}
    # Process cumulative share data
    for key, value in cumulativeShare.items():
        dfcumulative = value.iloc[11:].reset_index(drop=True)  # Adjust the
dataframe to remove unnecessary rows
        dfcumulative.columns = dfcumulative.iloc[0]  # Set the first row as the
column headers
        dfcumulative = dfcumulative.iloc[1:]  # Remove the row used for headers
        if dfcumulative.shape[0] != 0:  # If the dataframe is not empty
            newKey = key
            if key.split(' | ')[0] not in categories:  # Adjust key if it does
not match category format
                newKey = key.split(' | ')[1] + ' | ' + key.split(' | ')[0]
            cumulativeShareModifiedBrand[newKey] = dfcumulative.replace(np.nan,
0)  #
    # Process assortment data
```

```
    for key, value in assortment.items():
        df = value.iloc[12:].reset_index(drop=True)  # Adjust the dataframe to
remove unnecessary rows
        df.columns = df.iloc[0]  # Set the first row as the column headers
        df = df.iloc[1:]  # Remove the row used for headers
        df['Top Brands'] = df['Top Brands'].ffill()  # Forward fill 'Top Brands'
colum
        # Replace specific values in 'Top Brands' as per 'valueToReplace'
dictionary
        for val, replacer in valueToReplace.items():
            df['Top Brands'] = df['Top Brands'].str.replace(val, replacer)
        dfBrand = df[~df['Top Brands'].str.contains('Total')]  # Filter out rows
containing 'Total' in 'Top Brands'
        dfTotal = df[df['Top Brands'].str.contains('Total') & (df['Top Brands']
!= 'Grand Total')].reset_index(drop=True)
        dfTotal['Top Brands'] = dfTotal['Top Brands'].str.replace(' Total', '')
 # Adjust 'Top Brands' column for total rows
        if df.shape[0] != 0:  # If the dataframe is not empty
            newKey = key
            if key.split(' | ')[0] not in categories:  # Adjust key if it does
not match category format
                newKey = key.split(' | ')[1] + ' | ' + key.split(' | ')[0]
            # Process client-specific data for each brand
            for brand in client_brands_competitor:
                if df[df['Top Brands'] == brand].shape[0] > 0:  # Check if brand
data exists in dataframe
                    assortmentClient[newKey + ' | ' + brand] = df[df['Top
Brands'] == brand].replace(np.nan, 0)
                    assortmentClient[newKey + ' | ' + brand] =
assortmentClient[newKey + ' | ' + brand].merge(
                        cumulativeShareModifiedBrand[newKey], how='left',
on='Product')
            # Store modified data for total, brand, and overall assortment
            assortmentModified[newKey] = df.replace(np.nan, 0)
            assortmentModifiedBrand[newKey] = dfBrand.replace(np.nan,
0).drop(columns=['Cumulative Product Share'])
            assortmentModifiedBrand[newKey] =
assortmentModifiedBrand[newKey].merge(
                cumulativeShareModifiedBrand[newKey], how='left', on='Product')
            assortmentModifiedTotal[newKey] = dfTotal.replace(np.nan, 0)
    return assortmentModifiedBrand, assortmentModifiedTotal, assortmentClient
```

Calling function & data frame After cleaning

```
assortmentBrand,assortmentTotal,assortmentClient=mixAssortmentCleaning(assortment,cumulativeShare)
```

✓ 0.5s

```
assortmentBrand['National | Chocolate']
```

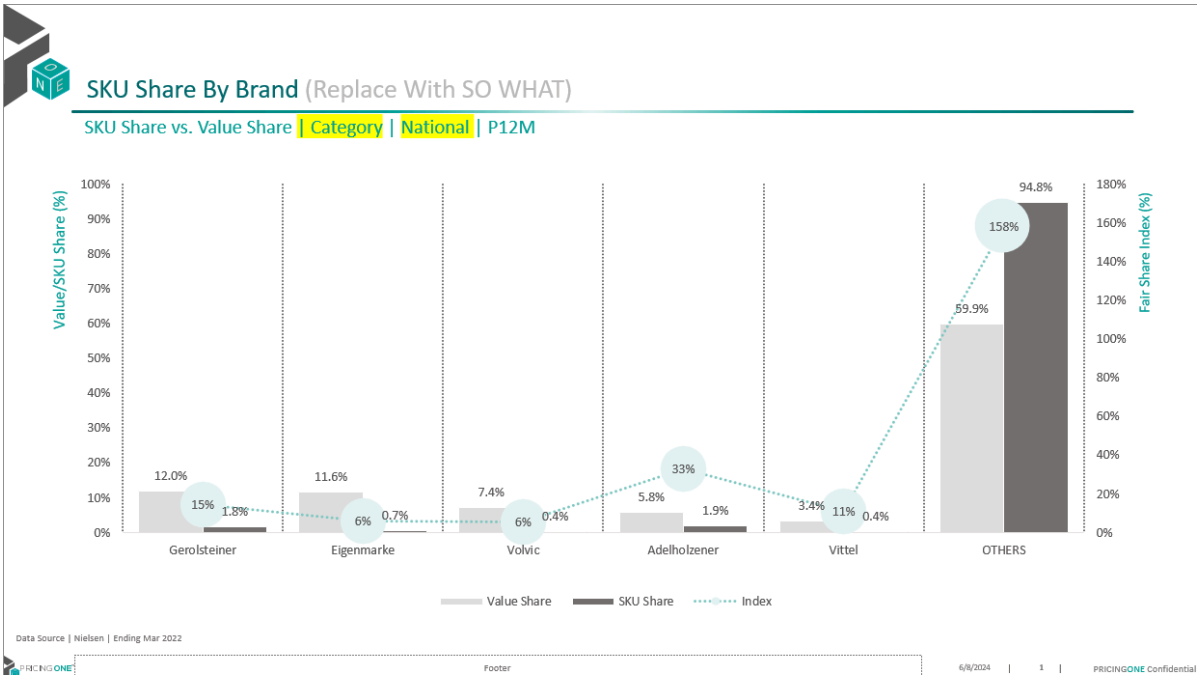
✓ 0.0s

	Top Brands	Product	Value Share	SKU Share	WD	Product Sales Rate	Net Sales	Trade Margin %	Cumulative Product Share
0	After Eight	After Eight Julekuler 91 Gr	1.688372e-07	0.000000	0.00	0.000000	0	0	0.999991
1	After Eight	After Eight Mini Eggs 90 Gr	1.158995e-04	0.001796	0.30	22675.966667	0	0	0.987178
2	After Eight	Nestle After Eight 200 Gr	2.941239e-03	0.004190	0.70	246625.357143	0	0	0.788592
3	After Eight	Nestle After Eight 300 Gr	1.531631e-07	0.000000	0.00	0.000000	0	0	0.999991
4	After Eight	Nestle After Eight 400 Gr	1.107409e-08	0.000000	0.00	0.000000	0	0	1.0
...	...	...	...	...	...	...	...	...	...
1104	Twix	Mf Twix Mini 20 Gr	8.456342e-06	0.000299	0.05	9927.000000	0	0	0.999246
1105	Twix	Mf Twix Single 50 Gr	3.050706e-05	0.000060	0.01	179063.000000	0	0	0.997089
1106	Twix	Mf Twix Xtra Salted Caramel 75 Gr	6.177001e-04	0.002514	0.42	86324.452381	0	0	0.946677
1107	Twix	Mf Twix Xtra White 75 Gr	1.359557e-07	0.000000	0.00	0.000000	0	0	0.999993
1108	Twix	Twix 12-Pk 600 Gr	1.008577e-05	0.000000	0.00	0.000000	0	0	0.999153

1109 rows × 9 columns

Step 3: Write Functions to Create Slides

Example Slide : SKU Share By Brand



Replacement function

- SkuShareByBrand function:Updates PowerPoint slides with SKU and Value Share data by brand for a given market.
    - Args:
      prs (Presentation): The PowerPoint presentation object.
      assortmentTotalSorted (dict): Dictionary containing sorted assortment data by market.
      numOfDuplicates (int): Number of slides to duplicate and update.
      position (int, optional): Starting position for slide updates. Default is 0.

```
def SkuShareByBrand(prs, assortmentTotalSorted, numOfDuplicates, position=0):
    for slidenum in range(numOfDuplicates):
        market = list(assortmentTotalSorted.keys())[slidenum]
```

```
        df = assortmentTotalSorted[market].copy()

        # Sort the dataframe by 'Value Share' in descending order
        df = df.sort_values('Value Share', ascending=False)

        # Ensure that 'Others' is the last row in the dataframe
        df = pd.concat([df[df['Top Brands'] != 'Others'], df[df['Top Brands'] ==
'Others']]).reset_index(drop=True)

        # Get the shapes in the current slide
        shapes = prs.slides[slidenum + position].shapes

        # Update text in specific shapes
        shapes[4].text = data_source
        shapes[5].text = shapes[5].text.replace('National', market.split(' | ')
[1]).replace('Category', market.split(' | ')[0])

        # Format the text in the shapes
        shapes[5].text_frame.paragraphs[0].font.size = Pt(12)
        shapes[5].text_frame.paragraphs[0].font.name = 'Nexa Bold (Headings)'
        shapes[6].text_frame.paragraphs[0].font.size = Pt(16)
        shapes[6].text_frame.paragraphs[0].font.name = 'Nexa Bold (Headings)'

        # Create tables and charts from the shapes
        tables, charts = createTableAndChart(shapes)
        chart = charts[0].chart

        # Prepare chart data
        chart_data = CategoryChartData()
        chart_data.categories = df['Top Brands'].tolist()
        chart_data.add_series('Value Share', df['Value Share'])
        chart_data.add_series('SKU Share', df['SKU Share'])

        # Calculate the index (SKU Share / Value Share) and handle division by
zero
        chart_data.add_series('Index', df['SKU Share'] / df['Value
Share'].replace(0, 1))

        # Replace the chart data with the prepared data
        chart.replace_data(chart_data)
```

Step 4: Duplicate Slides

Duplicate Slides:

generate a PowerPoint presentation with multiple sections, each containing data visualizations and tables

```
# Define the indexes for different sections of the presentation
index = [0,1,2,3 if len(client_brands)>0 else None ,4,5 if len(client_brands)>0
else None,6 if len(client_brands)>0 else None, *[7]*((1 if sectors else 0) + (1
if segments else 0) + (1 if subsegments else 0) + (1 if subcategories else 0)),0]
index = [x for x in index if x is not None]
```

```
duplication =
[len(assortmentBrandSorted.keys()),len(assortmentBrand.keys()),len(cumulativeShar
eTop50.keys()), len(assortmentClient.keys()) if len(client_brands) >0 else
None,len(assortmentBrand.keys()) , len(assortmentClientBrand.keys()) if
len(client_brands)>0 else None , len(assortmentClientBrand.keys()) if
len(client_brands)>0 else None,
               len(final_Sector.keys()) if len(sectors) >0 else None,
len(final_Segment.keys()) if len(segments) > 0 else None,
len(final_SubCategory.keys()) if len(subcategories) >0 else None,
len(final_Subsegment.keys()) if len(subsegments) > 0 else None
,len(assortmentBrandNOTSorted.keys())]
duplication = [x for x in duplication if x is not None]

section_names = ["SKU Share By Brand","Cumulative Product Shares","Top 50%
cumulative share","Brand Cumulative Product Share" if len(client_brands) >0 else
None,"Top 20 cumulative share","SKU Productivity Analysis with TM%" if
len(client_brands)>0 else None, "SKU Productivity Analysis with WD" if
len(client_brands)>0  else None,
                "Sectors Fair Share" if len(sectors) > 0 else None, "Segments
Fair Share" if len(segments) > 0 else None, "SubCategory Fair Share" if
len(subcategories) > 0 else None, "SubSegment Fair Share" if len(subsegments) > 0
else None,"SKU Share By Brand no client prio" ]
section_names = [x for x in section_names if x is not None]

path = os.getcwd() + '//Assortment base Oct 2024.pptx'
new_pre = os.getcwd() + '//slide duplicated1.pptx'


length = len(duplication)
for i in reversed(range(length)):
    if duplication[i]==0:
        del duplication[i]
        del index[i]
        del section_names[i]
```

Duplication Function

We use the duplication function to duplicate slides by number of the duplicate and save it in the duplication deck to use it to replace data.
Duplicate slides in a PowerPoint presentation.

Parameters:

- index (list): List of slide indices to duplicate.
- duplication (list): List specifying the number of times each slide should be duplicated.
- section_names (list): List of names for sections to be added.
- path (str): Path to the PowerPoint presentation file.
- new_pre (str): Path to save the duplicated presentation.

Returns:

- str: A message indicating success or failure.

```
####New_With_duplicate
import pythoncom
defslideDuplication(index=[0,1],duplication=[1,1],section_names=
[''],path='',new_pre=''):
```

```
    lis=[]
    iflen(index)!=len(duplication)!=len(section_names):
        return'The Index list not equal the Duplication number list in length'
    app = win32.Dispatch("PowerPoint.Application")
    presentation = app.Presentations.Open(path)
    # Iterate through the slides in the original presentation and copy them to
the new presentation
    for i inrange(len(index)):
        iftype(index[i])==list:
            # If index is a list of slide indices
            for num_duplicate inrange(duplication[i]):
                for k in index[i]:
                    slide=presentation.Slides[k]
                    duplicated_slide = slide.Duplicate()
                    duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-
(duplication[i]*len(index[i])))
        else:
            # If index is a single slide index
            slide=presentation.Slides[index[i]]
            for num_duplicate inrange(duplication[i]):
                duplicated_slide = slide.Duplicate()
                duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-duplication[i])
    # Add sections to the new presentation
    for j inrange(len(lis)):
        if duplication[j]!=0:
            presentation.SectionProperties.AddBeforeSlide(lis[j],section_names[j]
)
    # presentation.ApplyTheme(themePath)
    presentation.SectionProperties.Delete(1, True)
    presentation.SaveAs(new_pre)
    presentation.Close()
    # Close the original presentation and PowerPoint application
    app.Quit()
```

Step 5: Replace Data in Slide

- Generate the slides for SKU Share By Brand using the provided data.
    - prs: The PowerPoint presentation object where slides will be added or modified.
    - assortmentTotalSorted: The data set containing SKU share information, sorted accordingly.
    - duplication[0]: The number of duplicates to create for this section, specified by the first element in the duplication list.
    - position=0: The starting position for slide creation in this section.

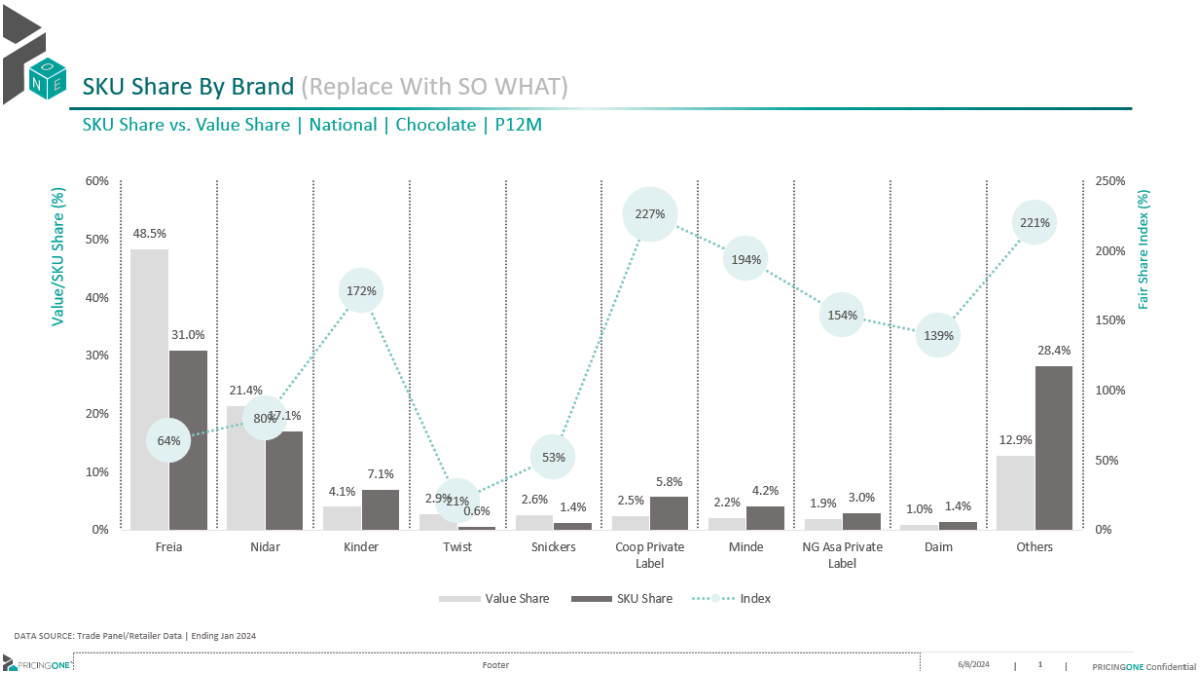
-
```
posItr = 0

ind=0

SkuShareByBrand(prs,assortmentBrandSorted,duplication[ind],position=posI
tr)

posItr = posItr+len(assortmentBrandSorted)

ind+=1
```

Step 6: Save Presentation

```
# Set the output path for the PowerPoint presentation
outputPath = os.getcwd() + "\\Mix and assortment doc output.pptx"

# Save the current PowerPoint presentation to the specified path
prs.save(outputPath)

# Open the saved PowerPoint presentation using the PowerPoint application
app = win32.Dispatch("PowerPoint.Application")
presentation = app.Presentations.Open(outputPath)
```

Example: OutPut Slide

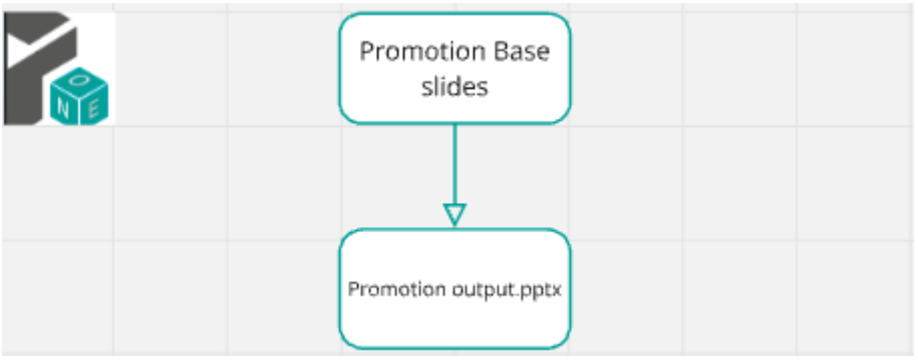
After Replacement Data "SKU Share By Brand"



Promotion Section

Introduction

In the slide automation Promotion: using19 slide base we create deck

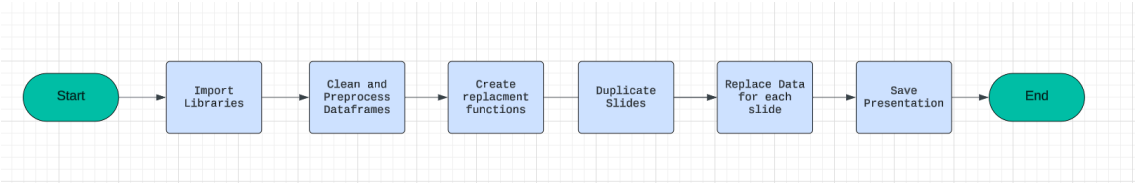


1-Promotion output slides:

- Promo Value Sales
- Promo evolution
- VSOD Summary by Sector
- Value uplift by retailer by brand
- Volume Uplift vs discount depth
- Value Uplift vs Promo Efficiency Quadrant
- Top 20 promotions
- Bottom 20 promotions
- Volume Sold on Deal
- Promo share vs Value Share
- Promo Sales by total size
- Promo Sales by promo type
- Feature Share vs. Fair Share
- Display Share vs. Fair Share
- Promo Frequency learnings
- Promo sales per retailer
- Value Uplift vs discount depth
- Seasonality Index
- Promotional Frequency Analysis

Project Steps

- Project Flow



- Step 1: Import Libraries we use

Step 1: Import Libraries we use

Ex: Libraries we use

- This script sets up an environment for working with PowerPoint presentations, data manipulation, filesystem operations, and COM (Component Object Model) object access.
- It imports necessary modules such as 'pptx' for PowerPoint automation, 'win32com' for COM object access and Windows automation, 'pandas' and 'numpy' for data manipulation,
- 'pathlib' for working with filesystem paths, 're' for regular expression operations, and various other modules for general-purpose tasks like file operations and timing functions.
- By importing these modules, the script prepares itself for tasks such as creating or modifying PowerPoint presentations, analyzing data using pandas and numpy, interacting
- with the Windows environment using win32com, and performing filesystem operations using shutil and os. Overall, this script provides a comprehensive setup for automating tasks
- related to PowerPoint presentations and general-purpose Python programming.

```
# Import necessary module for working with PowerPoint presentations
from pptx import Presentation
# Import the win32com.client module, aliasing it as win32 for convenience
import win32com.client as win32
# Import pandas for data manipulation and analysis
import pandas as pd
# Import numpy for numerical computing
import numpy as np
# Import the Path class from pathlib for working with filesystem paths
from pathlib import Path
# Import re for regular expression operations
import re
# Import sys for access to interpreter-related functions
import sys
# Import time for various time-related functions
import time
# Assign win32.constants to a shorter alias win32c for easier access
win32c = win32.constants
# Import shutil for high-level file operations
import shutil
# Import os for operating system dependent functionality
import os
# Import win32com.client again for COM object and functions access
import win32com.client
# Import warnings for warning control functionality
import warnings
```

Step 2: modified Data frame

EX: input dataframes before cleaning

```
1 promotions_brands_P12M['Chocolate | Extra'][12:]
```

✓ 0.0s Python

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10
12	Top Brands	Total Size	Promo Value	VSOD	VSOD IYA	Value Share	Promo Share	Value Uplift (v. base) Normalized	Value Uplift Normalized IYA	Volume Uplift (v. Base) Normalized	Volume Uplift Normalized IYA
13	After Eight	200GR	107310	0.037478	1.050033	0.003684	0.000311	-0.07536	0.456348	0.081989	0.598522
14	After Eight	90GR	34952	0.184211	0.709211	0.000317	0.000101	0.073454	30.05422	1.156863	16.932264
15	After Eight	0	142262	0.044774	0.9805	0.004002	0.000412	-0.052947	0.382478	0.121359	0.960285
16	All Others	100GR	254022	0.277887	1.868039	0.001331	0.000736	0.31869	16.651836	0.83208	5.026991
...	...	...	...	...	...	...	...	...	...	...	...
253	Twix	150GR	0	0	0	0.000001	0	0	0	0	0
254	Twix	20GR	550	0.14	0.598356	0.000012	0.000002	0.003791	0.041824	0.857143	1.797909
255	Twix	75GR	1078443	0.119496	2.547485	0.009466	0.003125	0.012193	-0.41119	0.07425	1.384742
256	Twix	0	1078993	0.119511	2.283769	0.009479	0.003126	0.012193	-0.405482	0.07425	1.4281
257	Grand Total	0	345134688	0.436316	0.996462	1	1	0.327254	1.308964	0.731802	1.134206

246 rows × 11 columns

Cleaning Data Frame

- cleaningData function: Clean and preprocess data in a dictionary of DataFrames.
  - Parameters:
    data (dict): Dictionary containing DataFrames.
  - Returns:
    dict: Dictionary containing cleaned DataFrames.

```
def cleaningData(data):
    cleaned_data = {}
    # Iterate over each key-value pair in the input dictionary
    for key in data:
        # Skip the first 11 rows if there are NaN values
        df = data[key].iloc[11:]
        if data[key].iloc[11,:].isna().any():
            df = data[key].iloc[12:]
        # Set column names and skip the first row
        df.columns = df.iloc[0]
        df = df.iloc[1:]
        # Perform specific cleaning operations based on the DataFrame columns and
key
        if df.shape[0] > 0 and not 'National' in key:
            if 'Top Brands' in df.columns and 'Product' in df.columns:
                df['Top Brands'] = df['Top Brands'].fillna(method='ffill')
                df['Product'].fillna('', inplace=True)
                df.fillna(0, inplace=True)
                df['Top Brands'] = df['Top Brands'].apply(lambda x: 'Grand Total'
if 'Grand Total' in x else x.replace('Total', '').strip())
            elif 'Top Brands' in df.columns:
                df['Top Brands'] = df['Top Brands'].fillna(method='ffill')
                df.fillna(0, inplace=True)
                df['Top Brands'] = df['Top Brands'].apply(lambda x: 'Grand Total'
if 'Grand Total' in x else x.replace('Total', '').strip())
```

```
                df = df[~df['Top Brands'].str.contains('Total', case=False)]
                df = df[df['Total Size'] == 0].reset_index(drop=True)
                df['VSOD Evaluation vs YA']=df['VSOD IYA']-1
                df['Promo Value Uplift vs YA']=df['Value Uplift Normalized
IYA']-1

            elif 'End of Week' in df.columns and 'Product' in df.columns:
                df['Product'] = df['Product'].fillna(method='ffill')
                df = df[(df['End of Week'].str.contains('2023|2024')) & (df['End
of Week'].notna())]
                df['End of Week'] = pd.to_datetime(df['End of Week'])
                df = df[(df['End of Week'] >= start_date) & (df['End of Week'] <=
end_date)]
                df = df[~df['Product'].str.contains('Total',
case=False)].reset_index(drop=True)
                df = df[df['Promo Sales'] > 10000]
                df = df.dropna(subset=['Value Uplift (v. base) Normalized'])
                df.fillna(0, inplace=True)
                df = df.reset_index(drop=True)

            elif 'End of Week' in df.columns:
                df['End of Week'] = df['End of Week'].astype(str)
                df = df[~df['End of Week'].str.contains('Total',
case=False)].reset_index(drop=True)
                df['End of Week'] = pd.to_datetime(df['End of Week'])
                df['End of Week'] = df['End of Week'].dt.strftime("%d-%b-%y")
                df = df[(df['End of Week'].str.contains('-21|-22|-23|Jan-24')) &
(df['End of Week'].notna())]
                df['End of Week'] = pd.to_datetime(df['End of Week'])
                df = df[(df['End of Week'] >= start_date) & (df['End of Week'] <=
end_date)]
                df = df.dropna()

            elif 'Grand Total' in df.columns:
                df.fillna(0, inplace=True)
            # Check if the key matches specific categories and modify the key
accordingly
            if key.split(' | ')[0] in categories and len(key.split(' | ')) == 3:
                modified_key = key.split(' | ')[1] + ' | ' + key.split(' | ')[2]
+ ' | ' + key.split(' | ')[0]
                if df.shape[0] > 0:
                    cleaned_data[modified_key] = df
            else:
                if df.shape[0] > 0:
                    cleaned_data[key] = df
    return cleaned_data
```

Example: out put data frame after cleaning

```
modified_promotionBrandsP12M['Chocolate | Extra']
```

0.0s

	Top Brands	Total Size	Promo Value	VSOD	VSOD IYA	Value Share	Promo Share	Value Uplift (v. base) Normalized	Value Uplift Normalized IYA	Volume Uplift (v. Base) Normalized	Volume Uplift Normalized IYA	VSOD Evaluation vs YA	Promo Value Uplift vs YA
0	After Eight	0	142262	0.044774	0.980500	0.004002	0.000412	-0.052947	0.382478	0.121359	0.960285	-0.019500	-0.617522
1	All Others	0	1345902	0.127715	1.024267	0.014680	0.003900	0.030648	-1.910962	0.272667	2.954697	0.024267	-2.910962
2	Anthon Berg	0	3159154	0.543118	1.250703	0.006404	0.009153	0.412896	1.607176	0.946329	1.676777	0.250703	0.607176
3	Bounty	0	720265	0.158226	0.938610	0.004509	0.002087	0.053809	6.657652	0.229042	5.062335	-0.061390	5.657652
4	Cloetta	0	134216	0.109897	1.264334	0.001205	0.000389	-0.004717	0.103600	0.128676	2.546440	0.264334	-0.896400
5	Cloetta Pops	0	2240493	0.132561	6.038283	0.016972	0.006492	-0.037039	0.000000	0.047914	0.000000	5.038283	-1.000000
6	Coop Private Label	0	21883421	0.251406	1.004636	0.094973	0.063405	0.033694	0.542698	0.079599	0.583963	0.004636	-0.457302
7	Daim	0	1498659	0.173040	1.390213	0.009824	0.004342	0.171359	0.581397	0.468172	0.695607	0.390213	-0.418603
8	Fazer	0	59577	0.258721	0.573361	0.000349	0.000173	-0.266355	-0.628201	0.602151	0.433085	-0.426639	-1.628201
9	Ferrero Collection	0	109159	0.081767	3.500940	0.002162	0.000316	-0.024861	0.147943	0.626712	2.367580	2.500940	-0.852057
10	Ferrero Rocher	0	2120480	0.793027	0.978104	0.002583	0.006144	0.908856	0.903167	1.475941	0.958355	-0.021896	-0.096833
11	Freia	0	223643525	0.593847	1.034926	0.455472	0.647989	0.655490	1.461519	1.442216	1.275730	0.034926	0.461519
12	Kinder	0	4650681	0.094911	0.785139	0.052013	0.013475	0.064571	0.872318	0.226450	1.748173	-0.214861	-0.127682
13	Lindt	0	0	0.000000	0.000000	0.000020	0.000000	0.000000	0.000000	0.000000	0.000000	-1.000000	-1.000000
14	Lion	0	1138819	0.173942	1.728685	0.006685	0.003300	0.053134	-1.999631	0.139224	2.593842	0.728685	-2.999631
15	Minde	0	4014647	0.205001	1.324734	0.024370	0.011632	0.114571	0.777831	0.328579	1.283893	0.324734	-0.222169
16	Minde Banansjokolade	0	144225	0.052714	2.337497	0.002766	0.000418	0.000000	0.000000	0.000000	0.000000	1.337497	-1.000000
17	Nidar	0	56567024	0.331337	0.832924	0.205422	0.163898	0.246872	1.281968	0.513363	1.091808	-0.167076	0.281968
18	Panda	0	4464853	0.605112	1.452511	0.008286	0.012937	1.210137	0.902164	2.137433	0.925080	0.452511	-0.097836
19	Plamil	0	9452	0.009182	0.096181	0.000730	0.000027	0.000000	0.000000	0.000000	0.000000	-0.903819	-1.000000
20	Quality Street	0	140029	0.141605	0.960326	0.001524	0.000406	-0.131971	-1.316571	0.475904	0.392348	-0.039674	-2.316571
21	Sfinx	0	666321	0.156817	0.328493	0.007327	0.001931	-0.149193	-0.232850	0.124725	0.150433	-0.671507	-1.232850
22	Smarties	0	1748396	0.168451	1.067591	0.008886	0.005066	0.470869	0.763573	0.465613	0.900714	0.067591	-0.236427
23	Snickers	0	1247535	0.058060	1.177163	0.022917	0.003615	-0.125574	7.827734	0.000000	0.000000	0.177163	6.827734
24	Sport Lunch	0	37437	0.008559	1.524639	0.004742	0.000108	0.000000	0.000000	0.000000	0.000000	0.524639	-1.000000
25	Toffifee	0	731984	0.182470	1.334811	0.004974	0.002121	0.030775	-1.042195	0.227264	0.676936	0.334811	-2.042195
26	Twist	0	11437179	0.497775	0.886195	0.026724	0.033138	1.105574	0.905302	2.233278	0.818025	-0.113805	-0.094698
27	Twix	0	1078993	0.119511	2.283769	0.009479	0.003126	0.012193	-0.405482	0.074250	1.428100	1.283769	-1.405482

Step 3: Write Functions to Create Slides

Example slide : Promo Value Sales base slide

Replacement function

- promoValueSales function:Generate PowerPoint slides for promo value sales

```
def promoValueSales(prs, promotionsBrandDF, numOfDuplicates, position=0):
    # Loop through each slide number
    for slidenum in range(numOfDuplicates):
        # Get market from promotionsBrandDF keys
        market = list(promotionsBrandDF.keys())[slidenum]
        # Retrieve DataFrame for the current market
        df = promotionsBrandDF[market].reset_index(drop=True)
        # Remove rows with 'Others' in 'Top Brands' column and filter by 'Value
Share'
        df = df[~df['Top Brands'].str.contains('Others', case=False)]
        df = df[df['Value Share'] > 0.01]
```

```
        # Select client brands
        df_client = selectClientBrands(promotionsBrandDF[market],'Top Brands',
'Promo Value')
        number_of_brands_needed = 5 - len(df_client)
        # Filter top brands and concatenate with client brands
        df = df[~df['Top Brands'].isin(client_brands)]
        df = df.sort_values(by='Promo Value',
ascending=False).head(number_of_brands_needed)
        df = pd.concat([df, df_client], ignore_index=True)
        df = df.sort_values(by='Promo Value', ascending=False)
        # Update title
        shapes = prs.slides[slidenum + position].shapes
        titlNumber = get_shape_number(shapes, "Promo Value Sales | Category |
National | P12M")
        shapes[titlNumber - 1].text = data_source
        shapes[titlNumber + 1].text_frame.paragraphs[0].font.size = Pt(16)
        shapes[titlNumber + 1].text_frame.paragraphs[0].font.name = 'Nexa Bold
(Headings)'
        shapes[titlNumber].text = shapes[titlNumber].text.replace('Category',
market.split(' | ')[0]).replace(
            'National', market.split(' | ')[1])
        shapes[titlNumber].text_frame.paragraphs[0].font.size = Pt(12)
        shapes[titlNumber].text_frame.paragraphs[0].font.name = 'Nexa Bold
(Headings)'
        # Create table and chart
        tables, charts = createTableAndChart(shapes)
        table = tables[0].table

        # Remove unnecessary rows
        num_rows_to_remove = len(table.rows) - df['Top Brands'].nunique() - 1
        table_height = get_table_height(table)
        for _ in range(num_rows_to_remove):
            if len(table.rows) > 1:
                row = table.rows[1]
                remove_row(table, row)

        # Adjust row heights
        total_row_height = table_height - table.rows[0].height
        num_rows = len(table.rows) - 1
        if num_rows > 0:
            cell_height = total_row_height / num_rows
            for row in range(1, table.rows.__len__()):
                table.rows[row].height = int(cell_height)
        # Populate table cells
        for i, row in enumerate(table.rows):
            for j, cell in enumerate(row.cells):
                if i == 0:  # Header row
                    continue
                if j == 0:  # Brand column
                    cell.text = df['Top Brands'].iloc[i - 1]
                    cell.text_frame.paragraphs[0].font.name = 'Nexa Bold'
                elif j == 1:  # Promo Value sales column
                    value = df['Promo Value'].iloc[i - 1]
                    if len(str(value)) > 3:
                        formatted_value = '{:,}'.format(int(value))
                        cell.text = str(formatted_value)
```

```
                        cell.text_frame.paragraphs[0].font.name = 'Nexa Book'
                    else:
                        cell.text = str(df['Promo Value'].iloc[i - 1])
                        cell.text_frame.paragraphs[0].font.name = 'Nexa Book'
                elif j == 2:  # Volume Sold on Deal (VSOD) column
                    cell.text = str(int(round(df['VSOD'].replace(np.nan,
0).iloc[i - 1] * 100, 0))) + '%'
                    cell.text_frame.paragraphs[0].font.name = 'Nexa Book'
                else:  # VSOD IYA column
                    cell.text = str(int(round(df['VSOD IYA'].replace(np.nan,
0).iloc[i - 1] * 100, 0)))
                    cell.text_frame.paragraphs[0].font.name = 'Nexa Book'
                # Set font size and alignment
                cell.text_frame.paragraphs[0].font.size = Pt(8)
                cell.text_frame.paragraphs[0].alignment = PP_ALIGN.CENTER
```

Step 4: Duplicate Slides:

- This code generates indices, duplication factors, and section names for PowerPoint slides
- based on different promotional data sources. It sets up paths for the base and duplicated slides,
- and ensures the correct indices and duplication values for each section of the presentation.

```
# Generate indices for slides containing promo value data for different
categories, sectors, and segments
slidePromoValueIndex = [
    [i + 15 for i in catDuplication.values()],  # Adjust category duplication
indices by adding 15
    [i + 15 for i in secDuplication.values()],  # Adjust sector duplication
indices by adding 15
    [i + 15 for i in segDuplication.values()]   # Adjust segment duplication
indices by adding 15
]
# Create a list of slide indices, conditional on the presence of promo type,
feature share, and display share data
index = [
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
    12 if promo_type != False else None,  # Conditional index for promo type
    13 if feature_share != False else None,  # Conditional index for feature
share
    14 if display_share != False else None,  # Conditional index for display
share
    15, *slidePromoValueIndex, 20  # Base index 15, adjusted promo value indices,
and final index 20
]
# Remove None values from the index list
index = [x for x in index if x is not None]
# Calculate the lengths of various datasets
len_brands = len(modified_promotionBrandsP12M)
len_Prod = len(modified_promotionProductsP12M)
len_modified_prod = len(new_modified_promotionProductsP12M)
len_client_market = len(client_brands) * len(regions_RET)
```

```
# Define duplication factors for each section based on the lengths of relevant
datasets
duplication = [
    len_brands, len(promotionsBrandSortedTotal), len(promotionsBrandsWithMarket),
len(concated),
    len_Prod, len_modified_prod, len_modified_prod, len(top20clientonly),
len(bottom20clientonly),
    len_client_market, len_brands, len(newModifiedBrands),
    len(newModifiedBrands) if promo_type != False else None,  # Conditional
duplication factor for promo type
    len_brands if feature_share != False else None,  # Conditional duplication
factor for feature share
    len_brands if display_share != False else None,  # Conditional duplication
factor for display share
    len(modified_promotionEndOfWeek), 1, 1,
    1 if len(segments) > 0 else None,  # Conditional duplication factor for
segments
    len(modified_valueUplift)
]

# Remove None values from the duplication list
duplication = [x for x in duplication if x is not None]

# Define section names for each part of the presentation
section_names = [
    "Promo Value Sales", "Promo Evolution", "VSOD Summary", "Value uplift by
retailer by brand",
    "Volume Uplift vs discount depth", "Value Uplift vs Promo Efficiency
Quadrant", "Top 20 promotions",
    "Top 20 promotions CLIENT ONLY", "Bottom 20 promotions CLIENT ONLY", "Volume
Sold on Deal",
    "Promo share vs Value Share", "Promo Sales by total size",
    "Promo Sales by promo type" if promo_type != False else None,  # Conditional
section name for promo type
    "Feature Share vs Fair Share" if feature_share != False else None,  #
Conditional section name for feature share
    "Display Share vs Fair Share" if display_share != False else None,  #
Conditional section name for display share
    "Promo Frequency learnings", "Category Promo sales per retailer", "Sector
Promo sales per retailer",
    "Segment Promo sales per retailer" if len(segments) > 0 else None,  #
Conditional section name for segments
    "Value Uplift vs discount depth"
]
# Remove None values from the section names list
section_names = [x for x in section_names if x is not None]

# Define paths for the base PowerPoint file and the duplicated PowerPoint file
path = os.getcwd() + '//slide base.pptx'
new_pre = os.getcwd() + '//slide duplicated.pptx'

# Define the data source string to be used in the presentation
data_source = "DATA SOURCE: Trade Panel/Retailer Data | Ending Jan 2024"
```

Duplication Function

We use the duplication function to duplicate slides by number of the duplicate and save it in the duplication deck to use it to replace data.
Duplicate slides in a PowerPoint presentation.
Parameters:

- index (list): List of slide indices to duplicate.
    - duplication (list): List specifying the number of times each slide should be duplicated.
        - section_names (list): List of names for sections to be added.
        - path (str): Path to the PowerPoint presentation file.
        - new_pre (str): Path to save the duplicated presentation.
          Returns:
        - str: A message indicating success or failure.

```
####New_With_duplicate
import pythoncom
defslideDuplication(index=[0,1],duplication=[1,1],section_names=
[''],path='',new_pre=''):
    lis=[]
    iflen(index)!=len(duplication)!=len(section_names):
        return'The Index list not equal the Duplication number list in length'
    app = win32.Dispatch("PowerPoint.Application")
    presentation = app.Presentations.Open(path)
    # Iterate through the slides in the original presentation and copy them to
the new presentation
    for i inrange(len(index)):
        iftype(index[i])==list:
            # If index is a list of slide indices
            for num_duplicate inrange(duplication[i]):
                for k in index[i]:
                    slide=presentation.Slides[k]
                    duplicated_slide = slide.Duplicate()
                    duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-
(duplication[i]*len(index[i])))
        else:
            # If index is a single slide index
            slide=presentation.Slides[index[i]]
            for num_duplicate inrange(duplication[i]):
                duplicated_slide = slide.Duplicate()
                duplicated_slide.MoveTo(presentation.Slides.Count)
            lis.append(presentation.Slides.count+1-duplication[i])
    # Add sections to the new presentation
    for j inrange(len(lis)):
        if duplication[j]!=0:
            presentation.SectionProperties.AddBeforeSlide(lis[j],section_names[j]
)
    # presentation.ApplyTheme(themePath)
    presentation.SectionProperties.Delete(1, True)
    presentation.SaveAs(new_pre)
    presentation.Close()
    # Close the original presentation and PowerPoint application
```

```
    app.Quit()
```

Step 5: Replace Data in Slides

- Call the promoValueSales function to generate slides for promotional value sales
- prs: PowerPoint presentation object
- modified_promotionBrandsP12M: Dictionary containing promotion data for different markets
- duplication[posItr]: Number of slides to duplicate for the current market
- position=posItr: Starting position to add slides in the presentation

```
promoValueSales(prs, modified_promotionBrandsP12M, duplication[posItr],
position=posItr)

# Increment the position iterator by 1 to move to the next section for the next
function call
posItr += 1
```

Step 6: Save Presentation

- performs two main tasks: saving the current PowerPoint presentation to a file and opening that file using the PowerPoint application. The outputPath variable is constructed using the current working directory, ensuring the presentation is saved in the correct location. After saving the presentation, the script uses win32com.client to dispatch the PowerPoint application and open the saved presentation. This automation allows for seamless transition from generating the presentation to viewing or editing it in PowerPoint, streamlining the workflow for creating market analysis slides.

```
# This script saves the generated PowerPoint presentation to a specified path
# and then opens the saved presentation using the PowerPoint application.

# Define the output path for the PowerPoint presentation
outputPath=os.getcwd() + "\\Promotion doc output.pptx"

# Save the PowerPoint presentation to the specified output path
prs.save(outputPath)

# Initialize the PowerPoint application using win32com client
app = win32.Dispatch("PowerPoint.Application")

# Open the saved PowerPoint presentation
presentation = app.Presentations.Open(outputPath)
```

Example slide : Promo Value Sales slide after replacement

Promo Value Sales (Replace With SO WHAT)

Promo Value Sales | Chocolate | Extra | P12M

Brand	Promo Value sales	Volume Sold on Deal (VSOD)	VSOD IYA
Freia	223,643,525	59%	103
Nidar	56,567,024	33%	83
Coop Private Label	21,883,421	25%	100
Twist	11,437,179	50%	89
Daim	1,498,659	17%	139

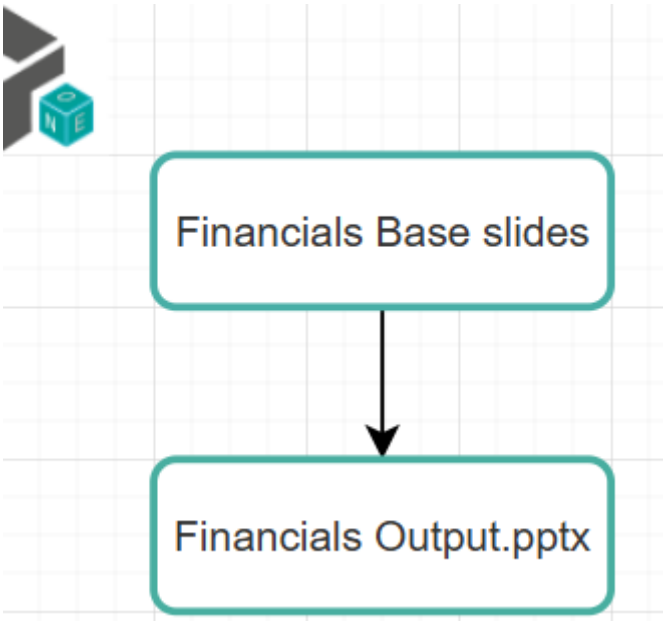
DATA SOURCE: Trade Panel/Retailer Data | Ending Jan 2024

PRICINGONE Footer 6/9/2024 | 1 | PRICINGONE Confidential

Financials Section

Introduction

In the slide automation Financials: using10 slide base we create deck

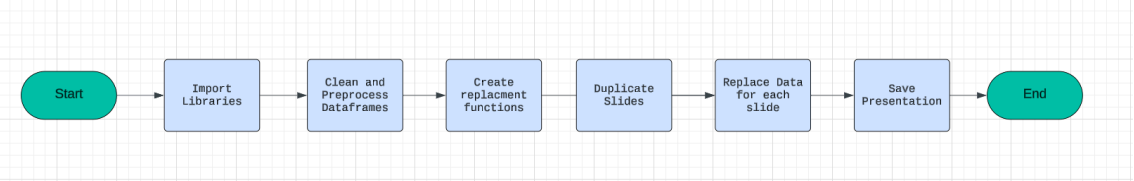


- 1-Financials output slides:
  - Mix Analysis
  - Trade Margin Analysis
  - Sector KPIs
  - SKU KPIs
  - Mix Matrix
  - Mix Matrix by brand

- Sector Spending pool
- Product Spending pool
- SKU Profitability Analysis with TM%
- Trade margin table vs Competition

Project Steps

- Project Flow



- Step 1: Import Libraries we use
- Step 2: modified Data frames: cleaning and preprocessing the data frames
- Step 3: Write Functions to Create Slides: Define functions to dynamically generate slides based on the base slides
- Step 4: Duplicate Slides: Use functions or methods to duplicate existing slides as needed for the presentation.
- Step 5: Replace Data in Slide: update information from the cleaned data frames to slides
- Step 6: Save Presentation

Step 1: Import Libraries we use

Ex: Libraries we use

- This script sets up an environment for working with PowerPoint presentations, data manipulation, filesystem operations, and COM (Component Object Model) object access.
- It imports necessary modules such as 'pptx' for PowerPoint automation, 'win32com' for COM object access and Windows automation, 'pandas' and 'numpy' for data manipulation,
- 'pathlib' for working with filesystem paths, 're' for regular expression operations, and various other modules for general-purpose tasks like file operations and timing functions.
- By importing these modules, the script prepares itself for tasks such as creating or modifying PowerPoint presentations, analyzing data using pandas and numpy, interacting
- with the Windows environment using win32com, and performing filesystem operations using shutil and os. Overall, this script provides a comprehensive setup for automating tasks
- related to PowerPoint presentations and general-purpose Python programming.

```
# Import necessary module for working with PowerPoint presentations
from pptx import Presentation
# Import the win32com.client module, aliasing it as win32 for convenience
import win32com.client as win32
# Import pandas for data manipulation and analysis
import pandas as pd
# Import numpy for numerical computing
import numpy as np
# Import the Path class from pathlib for working with filesystem paths
```

```python
from pathlib import Path
# Import re for regular expression operations
import re
# Import sys for access to interpreter-related functions
import sys
# Import time for various time-related functions
import time
# Assign win32.constants to a shorter alias win32c for easier access
win32c = win32.constants
# Import shutil for high-level file operations
import shutil
# Import os for operating system dependent functionality
import os
# Import win32com.client again for COM object and functions access
import win32com.client
# Import warnings for warning control functionality
import warnings
```

Step 2: modified Data frame

EX: input dataframes before cleaning

```python
brands_client_dfs['Edgewell | Amazon 1P | 3 Blade']
```

✓ 0.6s Python

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12	Unnamed: 13	Unnamed: 14
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Scope	SubSegment	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Category	Manual Shave Women	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Area	RETAILER	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Region	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	Channel	Amazon 1P	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	Market	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	MonthYear	(Multiple Items)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	Sector	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	Segment	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	Top Companies	Edgewell	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	SubSegment	3 Blade	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
12	SubCategory	All	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	Values	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	Top Brands	Gross Margin %	Gross Margin % IYA	Gross Profit	Net Sales	Net Sales/Kg	Net Sales/Kg IYA	Volume Sales	Trade Margin %	Trade Profit	Trade Profit/Kg	Av Price/KG	Value Sales IYA	Value Sales	Total WoB %
16	Hydro Silk	0.526789	0.684533	41795.518	79340.148	1.373808	0.95142	58927	0.423119	58192.852	1.007634	2.374497	49.600142	139922	0.747231
17	Intuition	0.506294	0.854395	13416.9563	26500.31	1.135598	0.999937	23336	0.439574	20785.69	0.890713	2.026311	1.386646	47286	0.252523
18	Skintimate	NaN	NaN	NaN	NaN	NaN	NaN	12	NaN	NaN	NaN	3.833333	1	46	0.000246
19	Grand Total	0.521658	0.870452	55212.4743	105840.458	1.305254	1.140011	82275	0.427329	78978.542	0.973986	2.275953	5.063245	187254	1

Cleaning Data Frame

- cleaningData function: Clean and preprocess data in a dictionary of DataFrames.
  - Parameters:
    data (dict): Dictionary containing DataFrames.
  - Returns:
    dict: Dictionary containing cleaned DataFrames.

```
modified_brand_P12M_dfs={}
for k in brands_client_dfs.keys():
    t=brands_client_dfs[k].copy()
    t=DetectHeader(t)
    t=t[:-1]
    t['Value Sales IYA'] = t['Value Sales IYA'].astype(float).fillna(-199)
    t=t.fillna(0)
    t = t[t['Net Sales'].astype(float) >= 1000]
    total= t[(t['Top Brands'].str.contains( ' Total')) & ~(t['Top
Brands'].isin(['Grand Total','All Others Total'])) & ~(t['Top Brands'].isin([i+'
Total' for i in client_brands]))]
    if not t.empty and len(t['Top Brands']) != 1:
        modified_brand_P12M_dfs[k]=t
```

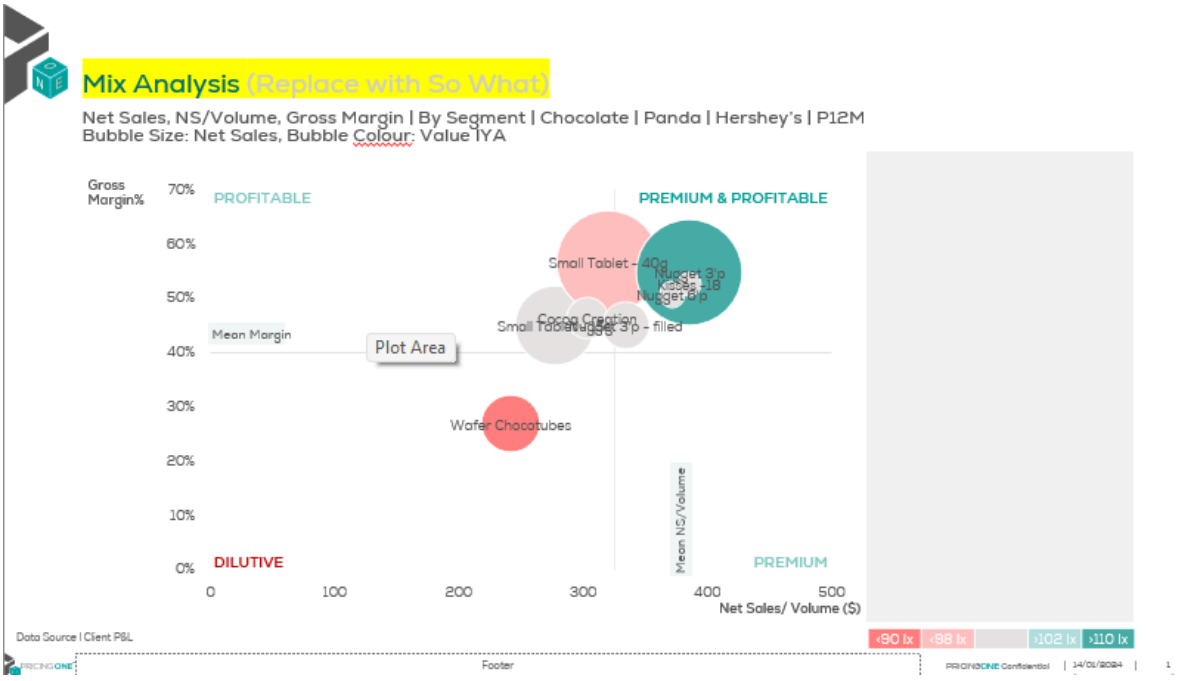
Example: out put data frame after cleaning

```
1 modified_brand_P12M_dfs['Edgewell | Amazon 1P | 3 Blade']
✓ 0.0s                                                                    Python
```

	Top Brands	Gross Margin %	Gross Margin % IYA	Gross Profit	Net Sales	Net Sales/Kg	Net Sales/Kg IYA	Volume Sales	Trade Margin %	Trade Profit	Trade Profit/Kg	Av Price/KG	Value Sales IYA	Value Sales	Total WoB %
0	Hydro Silk	0.526789	0.684533	41795.5180	79340.148	1.373808	0.951420	58927	0.423119	58192.852	1.007634	2.374497	49.600142	139922	0.747231
1	Intuition	0.506294	0.854395	13416.9563	26500.310	1.135598	0.999937	23336	0.439574	20785.690	0.890713	2.026311	1.386646	47286	0.252523

Step 3: Write Functions to Create Slides

Example slide :Mix Analysis base slide



Replacement function

- Mixanalysis function:Generate PowerPoint slides for Mix analysis

```
def
Mixanalysis(prs,dataframe,numOfDuplicates,Inscop='Sector',position=0,label_col=''
):
    for slidenum in range(numOfDuplicates):
        market=list(dataframe.keys())[slidenum]
        df=dataframe[market].copy()
```

```
        shapes = prs.slides[slidenum+position].shapes
        charts = []
        tables = []
        title = shapes.title.text

        headerNumber = get_shape_number(shapes, "Mix Analysis (Replace with So
What)")

        titleNumber = headerNumber-1
        shapes[titleNumber-1].text = data_source
        shapes[titleNumber].text ='Net Sales, NS/Volume, Gross Margin | By '+
Inscop +' | '+ market + ' | P12M''\n''Bubble Size: Net Sales, Bubble Colour:
Value IYA'
        shapes[titleNumber].text_frame.paragraphs[0].font.bold = False

        for shape in shapes:
            if shape.has_chart:
                shape_id = shape.shape_id
                charts.append(shape)
        chart = charts[0].chart

        # charts[0].left = Inches(0.57)
        chart_name = charts[0].name
        chart_type = chart.chart_type
        chart_data = BubbleChartData()
        chart_data.categories = df['Net Sales/Kg'].unique().tolist()
        series = chart_data.add_series('Gross Margin %')
        series.has_data_labels = True
        for i in range(df.shape[0]):
            series.add_data_point(df['Net Sales/Kg'].iloc[i], df['Gross Margin
%'].iloc[i], df['Net Sales'].iloc[i])
        chart.replace_data(chart_data)
        xlsx_file=BytesIO()
        with chart_data._workbook_writer._open_worksheet(xlsx_file) as (workbook,
worksheet):
            chart_data._workbook_writer._populate_worksheet(workbook, worksheet)
            worksheet.write(0, 4, "labels")
            worksheet.write_column(1, 4, df[label_col], None)

        chart._workbook.update_from_xlsx_blob(xlsx_file.getvalue())

        category_axis = chart.category_axis
        category_axis.auto_axis = True
        category_axis=chart.category_axis
        category_axis.minimum_scale=round(df["Net Sales/Kg"].min()-1)
        category_axis.maximum_scale=round(df["Net Sales/Kg"].max()+1)
        currencywithoutspace =currency.strip()  # Remove the leading space
        category_axis.axis_title.text_frame.text = f"Net Sales/ Volume
({currencywithoutspace})"  # Set the axis title text
        category_axis.axis_title.text_frame.paragraphs[0].font.size = Pt(8)
        category_axis.axis_title.text_frame.paragraphs[0].font.name = 'Nexa Bold'
        category_axis.has_number_format = True

        if sign.lower() == 'before':
            category_axis.tick_labels.number_format = f'#,##0.00'  if decimals ==
2 else f'#,##0'
```

```
        else:
            category_axis.tick_labels.number_format = f'#,##0.00'  if decimals ==
2 else f'#,##0'

        value_axis = chart.value_axis
        value_axis.minimum_scale = 0
        value_axis.maximum_scale=round(df["Gross Margin %"].max()+0.3 ,1)
        value_axis.tick_labels.number_format = '0%'
        value_axis.auto_axis = True

        df['Weight'] = df['Net Sales'] / sum(df['Net Sales'])

        df['Gross Margin % Weighted'] = df['Gross Margin %'] * df['Weight']
        Gross_Weight_value = sum(df['Gross Margin % Weighted']) /
sum(df['Weight'])

        vertical_cross_value = Gross_Weight_value  # Set the cross point based on
the average Gross Margin %
        value_axis.crosses = XL_AXIS_CROSSES.CUSTOM
        value_axis.crosses_at = vertical_cross_value
        # calculate the Weighted values
        df['Net Sales/Kg Weighted'] = df['Net Sales/Kg'] * df['Weight']
        Net_Sales_Weight_value = sum(df['Net Sales/Kg Weighted']) /
sum(df['Weight'])

        horizontal_cross_value = Net_Sales_Weight_value
        category_axis.crosses = XL_AXIS_CROSSES.CUSTOM
        category_axis.crosses_at = horizontal_cross_value



        chart_top = charts[0].top
        chart_height = charts[0].height
        chart_width = charts[0].width
        chart_down=chart_top + chart_height
        chart_left=charts[0].left
        chart_right=chart_left + chart_width

        proportion=vertical_cross_value / value_axis.maximum_scale
# Get axis scaling details
        vertical_min_scale = value_axis.minimum_scale
        vertical_max_scale = value_axis.maximum_scale

    # Calculate the proportion of the vertical_cross_value within the axis range
        vertical_value_range = vertical_max_scale - vertical_min_scale
        vertical_value_proportion = (vertical_cross_value - vertical_min_scale) /
vertical_value_range
    # Calculate the position of the vertical_cross_value in chart coordinates
        chart_y_position = (1 - vertical_value_proportion) * chart_height  #
Invert because Y axis increases downwards
    # Convert chart y position to slide position in inches
        slide_vertical_position = (chart_top + chart_y_position)*0.975
        min_scale = category_axis.minimum_scale
        max_scale = category_axis.maximum_scale
        value_range = max_scale - min_scale
        horizontal_proportion = (horizontal_cross_value - min_scale) /
value_range
```

```
        chart__position=  horizontal_proportion* chart_width
        slide_horizontal_position = (chart_left + chart__position)*0.975

        if pd.isna(slide_vertical_position):
            slide_vertical_position = 0  # or any other default value
        if pd.isna(slide_horizontal_position):
            slide_horizontal_position = 0  # or any other default value

        # Set the positions of the shapes based on calculated values (converting
from EMUs to inches)
        shapes[8].top = Inches(slide_vertical_position/ 914400 )

        shapes[15].left = Inches(slide_horizontal_position / 914400)

        for i,point in enumerate(chart.series[0].points):

            point.format.fill.solid()

            value_sales_iya = df['Value Sales IYA'].iloc[i]

            # if pd.isnull(value_sales_iya):  # Check if value_sales_iya is NaN
            #     value_sales_iya = 0  # Replace NaN with 0
            value_sales_iya *= 100
            # print(value_sales_iya)
            if (value_sales_iya == -19900.0):
                point.format.fill.fore_color.rgb = RGBColor(230, 229, 229)  #
Light Grey
            elif value_sales_iya < 90:
                point.format.fill.fore_color.rgb = RGBColor(255, 128, 128)  #
Light Red
                # print(f"Color set to: {point.format.fill.fore_color.rgb} for
{df[label_col].iloc[i]}")

            elif 90 <= value_sales_iya < 98:
                point.format.fill.fore_color.rgb = RGBColor(255, 191, 191)  #
Light Pink

            elif (98 <= value_sales_iya < 102):
                point.format.fill.fore_color.rgb = RGBColor(230, 229, 229)  #
Light Grey

            elif 102 <= value_sales_iya < 110:
                point.format.fill.fore_color.rgb = RGBColor(178, 223, 220)  #
Light Blue
            else:
                point.format.fill.fore_color.rgb = RGBColor(72, 174, 166)  # Dark
Blue
```

Step 4: Duplicate Slides:

- This code generates indices, duplication factors, and section names for PowerPoint slides
- based on different promotional data sources. It sets up paths for the base and duplicated slides,
- and ensures the correct indices and duplication values for each section of the presentation.

```
index = [
    *[0]*((1 if len(client_brands)>1 else 0)+(1 if regions_RET else 0)+(1 if
channels_RET else 0)+(1 if market_RET else 0)
          +(1 if regions_CHAN else 0)+(1 if channels_CHAN else 0)+(1 if
market_CHAN else 0)
          +(1 if regions_CUST else 0)+(1 if channels_CUST else 0)+(1 if
market_CUST else 0)
          +(1 if sectors else 0)+(1 if segments else 0)+(1 if subsegments else
0)+(1 if subcategories else 0)+(1 if len(client_brands)>1 else 0)),

    *[1]*((1 if sectors else 0)+(1 if segments else 0)+(1 if subsegments else 0)+
(1 if subcategories else 0)),
    *[2]*((1 if sectors else 0)+(1 if segments else 0)+(1 if subsegments else 0)+
(1 if subcategories else 0)),
    *[3]*((1 if len(client_brands)>1 else 0)+(1 if len(client_brands)>1 else 0)),
    *[4]*((1 if retailerDuplication !=0 else 0)+(1 if channelDuplication !=0 else
0 )+(1 if custDuplication !=0 else 0 )),
    *[5]*((1 if sectors else 0)+(1 if segments else 0)+(1 if subsegments else 0)+
(1 if subcategories else 0)),
    *[6]*((1 if sectors else 0)+(1 if segments else 0)+(1 if subsegments else 0)+
(1 if subcategories else 0)),
    *[7]*(1 if len(client_brands)>1 else 0),
    *[8]*( 1 if len(client_brands)>1 else 0),
    *[9]*((1 if sectors else 0)+(1 if segments else 0)+(1 if subsegments else 0)+
(1 if subcategories else 0))
          ]


# slide 0
duplication_1=[(len(modified_brand_P12M_dfs.keys()) if len(client_brands)>1 else
0)]
duplication_2 = [len(modified_retailer_regions.keys()) if len(regions_RET)>0 else
0, len(modified_retailer_channels) if len(channels_RET)>0 else 0,
len(modified_retailer_markets) if len(market_RET)>0 else 0,
                 len(modified_channels_regions.keys()) if len(regions_CHAN)>0
else 0, len(modified_channels_channels) if len(channels_CHAN)>0 else 0,
len(modified_channels_markets) if len(market_CHAN) >0 else 0,
                 len(modified_cust_regions.keys()) if len(regions_CUST)>0 else 0,
(len(modified_cust_channels) if len(channels_CUST)>0 else 0),
(len(modified_cust_markets) if len(market_CUST)>0 else 0)]

duplication_3 = [(len(modified_sectors_P12M_mix_analysis_dfs.keys()) if
len(sectors)>0 else 0), (len(modified_segment_P12M_mix_analysis_dfs.keys()) if
len(segments)>0 else 0), (len(modified_subsegment_P12M_mix_analysis_dfs) if
len(subsegments)>0 else 0), (len(modified_subcategory_P12M_mix_analysis_dfs) if
len(subcategories)>0 else 0)]

duplication_4 = [len(modified_product_sec_seg_P12M_dfs.keys())]
```

```
# Slide 1
duplication_5 = [(len(modified_sector_Trade_Margin_dfs.keys()) if len(sectors)!=0
else 0), (len(modified_segment_Trade_Margin_dfs.keys()) if len(segments)!=0 else
0), (len(modified_subsegment_Trade_Margin_dfs) if len(subsegments)!=0 else 0),
(len(modified_subcategory_Trade_Margin_dfs) if len(subcategories)!=0 else 0)]

#Slide 2
duplication_6 = [(len(merged_data_sector.keys()) if len(sectors)!=0 else 0),
(len(merged_data_segment.keys()) if len(segments)!=0 else 0),
(len(merged_data_subsegment) if len(subsegments)!=0 else 0),
(len(merged_data_subcategory) if len(subcategories)!=0 else 0)]

# Slide 3
duplication_7 =
[len(modified_product_dfs.keys()),len(modified_brand_product_dfs.keys())]
#slide 4
duplication_8 = [retailerDuplication,channelDuplication,custDuplication]

# Slide 5
duplication_9=[(len(secfinaldic.keys())if len(sectors)!=0 else 0),
(len(segfinaldic.keys())if len(segments)!=0 else 0),(len(subsegfinaldic.keys())if
len(subsegments)!=0 else 0),(len(subcatfinaldic.keys())if len(subcategories)!=0
else 0)]

# Slide 5
duplication_10 = [(len(modified_sector_P12M.keys()) if len(sectors)!=0 else 0),
(len(modified_segment_P12M.keys()) if len(segments)!=0 else 0),
(len(modified_subsegment_P12M) if len(subsegments)!=0 else 0),
(len(modified_subcategory_P12M) if len(subcategories)!=0 else 0)]

# Slide 6
duplication_11 = [len(modified_products_p12m)+ len(modified_products_p12m_brand)]

# Slide 7
duplication_12 = [len(modified_product_P12M_dfs) +
len(modified_brand_product_P12M_dfs)]

# Slide 8
duplication_13 = [(len(client_manuf_sector_dfs_new.keys()) if len(sectors)!=0
else 0), (len(client_manuf_segment_dfs_new.keys()) if len(segments)!=0 else 0),
(len(client_manuf_subsegment_dfs_new) if len(subsegments)!=0 else 0),
(len(client_manuf_subcategory_dfs_new) if len(subcategories)!=0 else 0)]


duplication = duplication_1 + duplication_2 + duplication_3 + duplication_4 +
duplication_5 + duplication_6 + duplication_7 + duplication_8+duplication_9 +
duplication_10 + duplication_11+duplication_12+duplication_13

duplication = [item for item in duplication if item !=0]

section_1 = (["Mix Analysis by brand"] if len(client_brands)>1 else [])
```

```
section_2 = (["Mix Analysis by Retailer for Region"] if len(regions_RET)!=0 else
[])+ (["Mix Analysis by Retailer for Channel"] if len(channels_RET)!=0 else [])+
(["Mix Analysis by Retailer for Market"] if len(market_RET)!=0 else [])+(["Mix
Analysis by Channel for Region"] if len(regions_CHAN)!=0 else [])+ (["Mix
Analysis by Channel for Channel"] if len(channels_CHAN)!=0 else [])+ (["Mix
Analysis by Channel for Market"] if len(market_CHAN)!=0 else [])+([f"Mix Analysis
by {customareas} for Region"] if len(regions_CUST)!=0 else [])+ ([f"Mix Analysis
by {customareas} for Channel"] if len(channels_CUST)!=0 else [])+ ([f"Mix
Analysis by {customareas} for Market"] if len(market_CUST)!=0 else [])
section_3 = (["Mix Analysis by Sector"] if len(sectors)!=0 else [])+ (["Mix
Analysis by Segment"] if len(segments)!=0 else [])+ (["Mix Analysis by
SubSegment"] if len(subsegments)!=0 else [])+ (["Mix Analysis by SubCategory"] if
len(subcategories)!=0 else [])
section_4 = ["Mix Analysis by"+ f'{prodORitem}' if len(client_brands)>1 else []]

section_5 = (["Trade Margin Analysis by Sector"] if len(sectors)!=0 else [])+
(["Trade Margin Analysis by Segment"] if len(segments)!=0 else [])+ (["Trade
Margin Analysis by SubSegment"] if len(subsegments)!=0 else [])+(["Trade Margin
Analysis by SubCategory"] if len(subcategories)!=0 else [])

section_6 = (["Sector KPI"] if len(sectors)!=0 else [])+ (["Segment KPI"] if
len(segments)!=0 else [])+ (["SubSegment KPI"] if len(subsegments)!=0 else [])+
(["SubCategory KPI"] if len(subcategories)!=0 else [])

section_7 = ("SKU KPIs Summary By Manufacture" if len(client_brands)>1 else
[],"SKU KPIs Summary By Brand" if len(client_brands)>1 else [])

section_8 = ['Mix Matrix By Retailer'if retailerDuplication != 0 else[] ,'Mix
Matrix By Channel'if channelDuplication != 0 else[],'Mix Matrix By Custom
Region'if custDuplication != 0
else[]]#,*section_names_slide4]#*section_names_slide3,*section_names_slide4]

section_9=["Mix Matrix By Brands by Sector" if len(sectors)!=0 else [],"Mix
Matrix By Brands by Segment" if len(segments)!=0 else [],"Mix Matrix By Brands by
SubSegment"if len(subsegments)!=0 else [],"Mix Matrix By Brands by SubCategory"if
len(subcategories)!=0 else []]

section_10 = (["Sector Spending Pool"] if len(sectors)!=0 else [])+ (["Segment
Spending Pool"] if len(segments)!=0 else [])+ (["SubSegment Spending Pool"] if
len(subsegments)!=0 else [])+ (["SubCategory Spending Pool"] if
len(subcategories)!=0 else [])

section_11 = (["Product Spending Pool"] if len(client_brands)>1 else [])

section_12 = [("SKU Profitability" if len(client_brands)>1 else [])]

section_13 = (["Trade Margin Table By Sector"] if len(sectors)!=0 else [])+
(["Trade Margin Table By Segment"] if len(segments)!=0 else [])+ (["Trade Margin
Table By SubSegment"] if len(subsegments)!=0 else [])+ (["Trade Margin Table By
SubCategory"] if len(subcategories)!=0 else [])

section_names = [*section_1 , *section_2 , *section_3 , *section_4 , *section_5 ,
*section_6 , *section_7 , *section_8, *section_9 , *section_10 ,
*section_11,*section_12,*section_13]

section_names = [item for item in section_names if item !=[]]
```

```
path = os.getcwd() + "\Financials base.pptx"
new_pre = os.getcwd() + '\Financials Slide Duplicate.pptx'
```

Step 5: Replace Data in Slides

- Call the Mixanalysis function to generate slides for Mix analysis
- prs: PowerPoint presentation object
- modified_brand_P12M_dfs : Dictionary containing promotion data for different markets
- duplication[p]: Number of slides to duplicate for the current market
- position=p: Starting position to add slides in the presentation

```
p=0
if len(client_brands)>1:

 Mixanalysis(prs,modified_brand_P12M_dfs,duplication[p],Inscop='Brand',position =
calculate_position(p),label_col='Top Brands')
    p+=1
```

Step 6: Save Presentation

- performs two main tasks: saving the current PowerPoint presentation to a file and opening that file using the PowerPoint application. The outputPath variable is constructed using the current working directory, ensuring the presentation is saved in the correct location. After saving the presentation, the script uses win32com.client to dispatch the PowerPoint application and open the saved presentation. This automation allows for seamless transition from generating the presentation to viewing or editing it in PowerPoint, streamlining the workflow for creating market analysis slides.

```
outputPath=os.getcwd() + "\Financials_Final_Output.pptx"
prs.save(outputPath)
```

Example slide : Mix analysis slide after replacement