PharmaSUG 2024 - Paper DV - 246

AutoVis Oncology Presenter: Automated Python-Driven Statistical Analysis and Visualizations for Powerful Presentations

Indraneel Narisetty, Jazz Pharmaceuticals

ABSTRACT

In late-phase or first-in-human clinical studies, understanding clinical data is vital for informed decision making, such as selecting the appropriate drug dose and evaluating its efficacy and safety. The traditional process of converting ADaM datasets into TLFs (tables, listings, and figures) and integrating them into clinical PowerPoint presentations has historically been a time-consuming task. Medical monitors and clinical teams create these presentations to conclude on dose selection, escalation, and drug effectiveness. Addressing this need, we've introduced "AutoVis Oncology Presenter," an innovative Python-based tool designed to streamline the transformation of Oncology clinical trials data into clear, impactful PowerPoint presentations. It's particularly adept at handling key ADaM datasets like ADTTE and ADRS, which are crucial for assessing treatment effectiveness. This paper will demonstrate how to build this tool, complete with Python code and practical examples. The goal of AutoVis is to make important safety and efficacy data both comprehensible and visually appealing using Python packages. It automates the generation of detailed tables and striking graphs, such as spider plots showing patient responses, waterfall plots, and swimmer plots, all neatly incorporated into PowerPoint presentations. Moreover, it helps in comparing CSR (Clinical Study Report) tables when they are generated, thereby enhancing the efficiency and clarity of presentations. This feature is particularly beneficial for clinical teams who need to regularly share their findings, be it in meetings, conferences, or reports. AutoVis accelerates the sharing of vital information, thereby advancing our understanding and treatment of cancer.

INTRODUCTION

Imagine diving into the world of data analysis for oncology clinical trials in an attempt to make sense of complex information. A clear and effective presentation of these data is crucial for informed decisions that could impact patients' lives. However, traditional methods of creating visualizations and integrating them into presentations every month, every quarter, or at every review meeting are tedious and time consuming. With AutoVis, the code is simply submitted using a terminal, and the presentation will be ready for you in seconds.

This is where the AutoVis Oncology Presenter comes in. It is a game changer —a Python-based tool designed to simplify the process of turning ADaM data into stunning visualizations and seamlessly adding them to PowerPoint presentations.

It harnesses the power of Python libraries, such as Pandas, Matplotlib, Pyreadstat, python-pptx, to create beautiful graphs and charts with just a few simple commands. These visuals not only look great, but also help to understand the effectiveness of the data.

This paper aims to demonstrate how Python can transform the way clinical trial data is analyzed and presented. Through practical examples, I will illustrate how AutoVis can save time, decrease errors, and enhance the impact of presentations. By providing you with the ability to visually represent data, we hope to empower you to make informed decisions and contribute to research advancements.

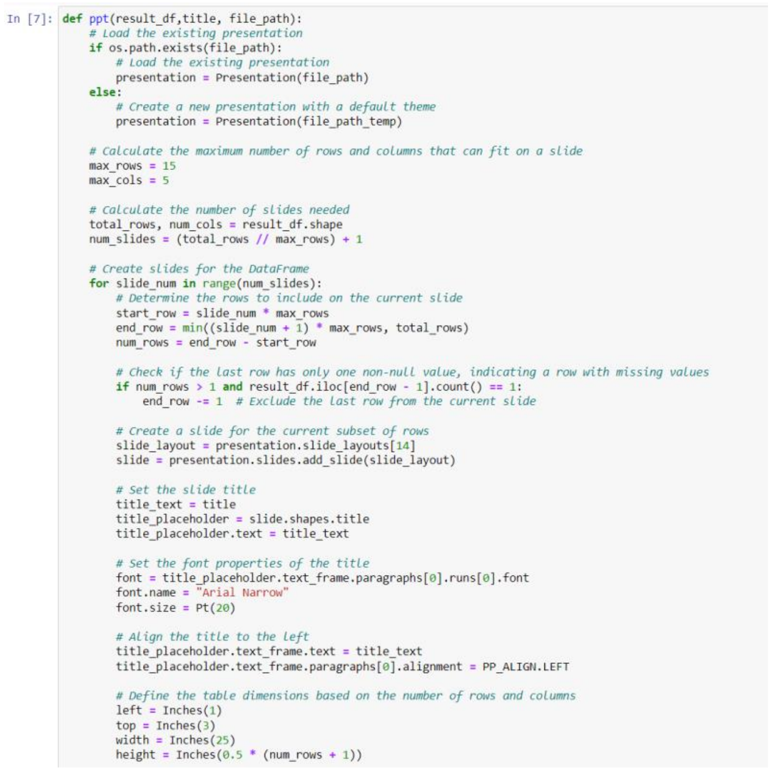
PYTHON PACKAGES USED

- To develop the AutoVis Oncology Presenter, we relied on a diverse set of Python packages, each contributing unique functionalities to the tool's capabilities. Below is an overview of the packages utilized in this project:

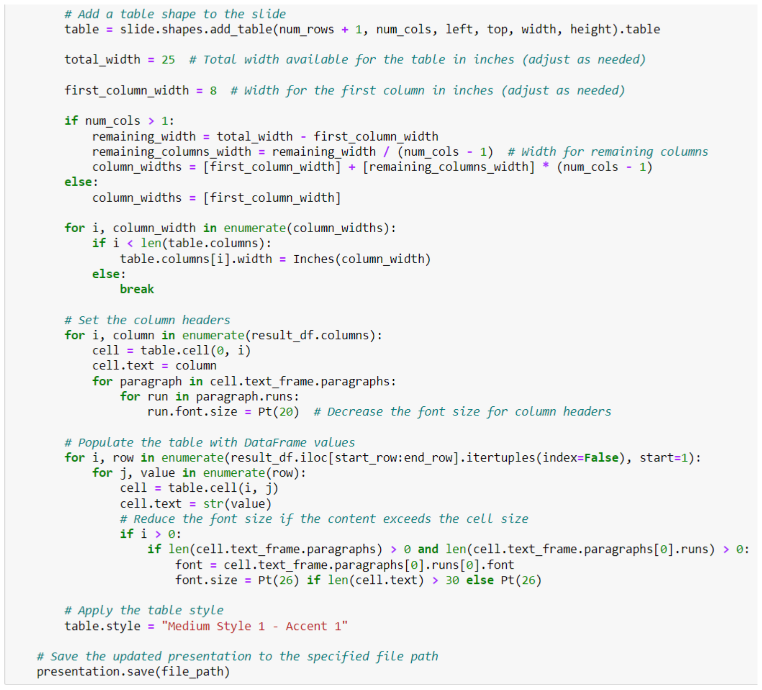
- To develop the AutoVis Oncology Presenter, we relied on a diverse set of Python packages, each contributing unique functionalities to the tool's capabilities. Below is an overview of the packages utilized in this project:
- Pandas: A fundamental library for data manipulation and analysis that serves as the backbone of our data processing pipeline.
- Os: Facilitates file and directory operations, enabling seamless interaction between data files and presentation assets.
- Pyreadstat: This enables reading and writing data in SPSS, SAS, and Stata formats, ensuring compatibility with common data sources in clinical trials.
- NumPy: Provides essential functionality for array manipulation, mathematical operations, and statistical analysis, enhancing the robustness and efficiency of data-processing tasks.
- Python-pptx: This allows for the creation and manipulation of PowerPoint presentations programmatically, enabling dynamic generation of slides, text, shapes, images, and tables.
- Plotly and plotly.express: Provides interactive and web-based visualization capabilities, enabling the creation of interactive plots, dashboards, and web applications directly from Python.
- Kaleido: Used for static image export from Plotly figures, ensuring compatibility with static formats such as PowerPoint presentations.
- Colorsy and auxiliary packages: Utilized for color manipulation, styling, and additional functionalities to enhance the visual aesthetics of presentations.

PYTHON PPT FUNCTION

The primary purpose of the ppt function is to enhance the efficiency and standardization of presenting clinical trial data. By automating the generation of PowerPoint slides, this function significantly reduces the manual effort involved in preparing presentations, enabling you and your team to concentrate more on analyzing and interpreting the data. The ppt function generates PowerPoint slides from Python Dataframe data, thereby facilitating the integration of clinical trial results into the presentation format. It requires three parameters: result_df, which represents the dataframe containing the pertinent data; title, which specifies the title of the slides; and file_path, which indicates the file path where the presentation will be saved.



```
In [7]: def ppt(result_df,title, file_path):
            # Load the existing presentation
            if os.path.exists(file_path):
                # Load the existing presentation
                presentation = Presentation(file_path)
            else:
                # Create a new presentation with a default theme
                presentation = Presentation(file_path_temp)

            # Calculate the maximum number of rows and columns that can fit on a slide
            max_rows = 15
            max_cols = 5

            # Calculate the number of slides needed
            total_rows, num_cols = result_df.shape
            num_slides = (total_rows // max_rows) + 1

            # Create slides for the DataFrame
            for slide_num in range(num_slides):
                # Determine the rows to include on the current slide
                start_row = slide_num * max_rows
                end_row = min((slide_num + 1) * max_rows, total_rows)
                num_rows = end_row - start_row

                # Check if the last row has only one non-null value, indicating a row with missing values
                if num_rows > 1 and result_df.iloc[end_row - 1].count() == 1:
                    end_row -= 1  # Exclude the last row from the current slide

                # Create a slide for the current subset of rows
                slide_layout = presentation.slide_layouts[14]
                slide = presentation.slides.add_slide(slide_layout)

                # Set the slide title
                title_text = title
                title_placeholder = slide.shapes.title
                title_placeholder.text = title_text

                # Set the font properties of the title
                font = title_placeholder.text_frame.paragraphs[0].runs[0].font
                font.name = "Arial Narrow"
                font.size = Pt(20)

                # Align the title to the left
                title_placeholder.text_frame.text = title_text
                title_placeholder.text_frame.paragraphs[0].alignment = PP_ALIGN.LEFT

                # Define the table dimensions based on the number of rows and columns
                left = Inches(1)
                top = Inches(3)
                width = Inches(25)
                height = Inches(0.5 * (num_rows + 1))
```



```
        # Add a table shape to the slide
        table = slide.shapes.add_table(num_rows + 1, num_cols, left, top, width, height).table

        total_width = 25  # Total width available for the table in inches (adjust as needed)

        first_column_width = 8  # Width for the first column in inches (adjust as needed)

        if num_cols > 1:
            remaining_width = total_width - first_column_width
            remaining_columns_width = remaining_width / (num_cols - 1)  # Width for remaining columns
            column_widths = [first_column_width] + [remaining_columns_width] * (num_cols - 1)
        else:
            column_widths = [first_column_width]

        for i, column_width in enumerate(column_widths):
            if i < len(table.columns):
                table.columns[i].width = Inches(column_width)
            else:
                break

        # Set the column headers
        for i, column in enumerate(result_df.columns):
            cell = table.cell(0, i)
            cell.text = column
            for paragraph in cell.text_frame.paragraphs:
                for run in paragraph.runs:
                    run.font.size = Pt(20)  # Decrease the font size for column headers

        # Populate the table with DataFrame values
        for i, row in enumerate(result_df.iloc[start_row:end_row].itertuples(index=False), start=1):
            for j, value in enumerate(row):
                cell = table.cell(i, j)
                cell.text = str(value)
                # Reduce the font size if the content exceeds the cell size
                if i > 0:
                    if len(cell.text_frame.paragraphs) > 0 and len(cell.text_frame.paragraphs[0].runs) > 0:
                        font = cell.text_frame.paragraphs[0].runs[0].font
                        font.size = Pt(26) if len(cell.text) > 30 else Pt(26)

        # Apply the table style
        table.style = "Medium Style 1 - Accent 1"

    # Save the updated presentation to the specified file path
    presentation.save(file_path)
```

Display 1. Python ppt functions to display Python Dataframe in a PowerPoint presentation.

Key Features:

1. **Dynamic Slide Creation:** This function dynamically creates slides based on the size of the dataframe, ensuring that data are presented comprehensively while avoiding overcrowded slides.

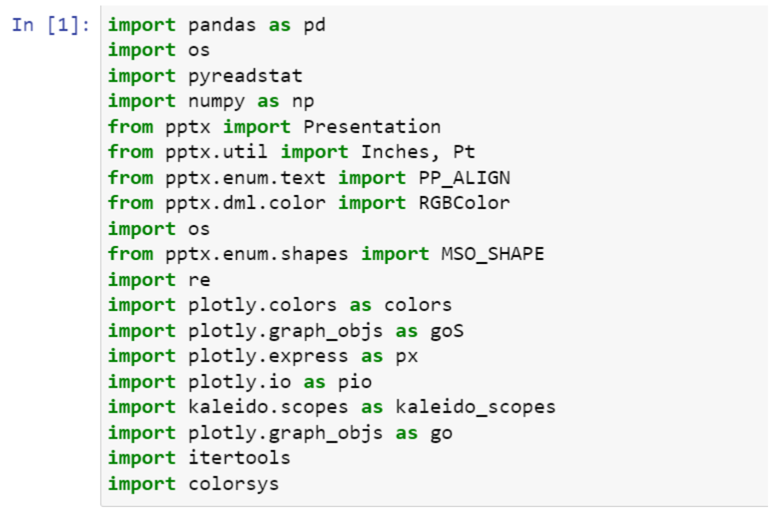
2. **Integration with Company Template:** This leverages your company's standard PowerPoint template, preserves brand identity, and ensures consistency in presentation style across all materials.
3. **Table Formatting:** The function formats tables within slides, adjusting column widths and font sizes as required to optimize readability and visual appeal.
4. **Automatic Slide Title Setting:** The title of each slide is automatically set based on the specified title parameter, ensuring coherence and clarity in slide organization.

STREAMLINING DATA PROCESSING: FROM SAS DATASETS TO PRESENTATION

READING SAS DATASETS INTO PYTHON DATAFRAME S

A Python script was developed to facilitate the conversion of the SAS® data files into Python data frames. The script, written in the Python programming language, utilizes the pandas, pyreadstat, and OS libraries for data manipulation, reading SAS files, and file system operations, respectively.

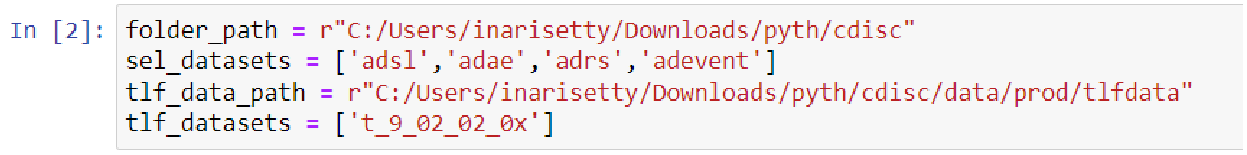
The script begins by defining the folder paths in which the SAS ADaM data files are located, and specifies the datasets of interest. The folder_path variable points to the directory containing the Analysis Data Model (ADaM) datasets, whereas tlf_data_path points to the directory containing the TLF (Tables, Listings, and Figures) datasets. Additionally, sel_datasets and tlf_datasets lists specify the datasets to be processed from each folder:



```
In [1]: import pandas as pd
        import os
        import pyreadstat
        import numpy as np
        from pptx import Presentation
        from pptx.util import Inches, Pt
        from pptx.enum.text import PP_ALIGN
        from pptx.dml.color import RGBColor
        import os
        from pptx.enum.shapes import MSO_SHAPE
        import re
        import plotly.colors as colors
        import plotly.graph_objs as goS
        import plotly.express as px
        import plotly.io as pio
        import kaleido.scopes as kaleido_scopes
        import plotly.graph_objs as go
        import itertools
        import colorsys
```

Display 2. Importing necessary python modules.

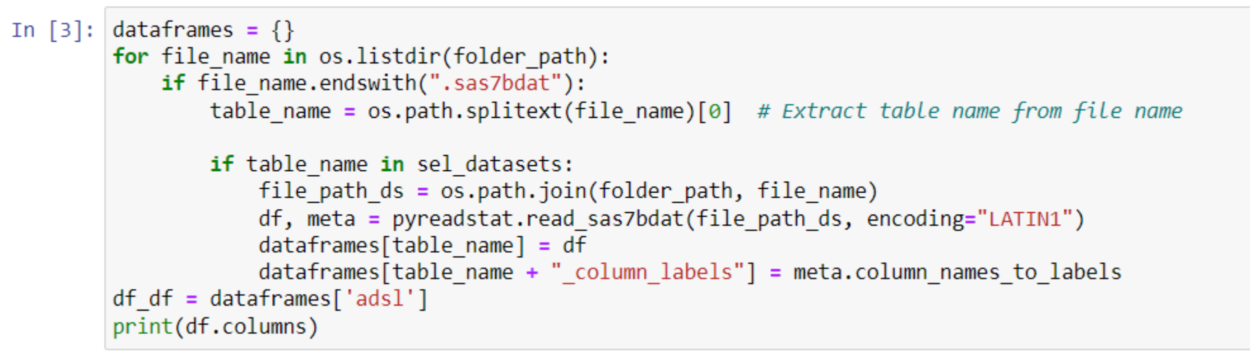
The script begins by importing the necessary Python libraries: os for file system operations, pandas for data manipulation, and pyreadstat for reading SAS data files.



```
In [2]: folder_path = r"C:/Users/inarisetty/Downloads/pyth/cdisc"
        sel_datasets = ['adsl','adae','adrs','advent']
        tlf_data_path = r"C:/Users/inarisetty/Downloads/pyth/cdisc/data/prod/tlfdata"
        tlf_datasets = ['t_9_02_02_0x']
```

Display 3. Python variables are defined.

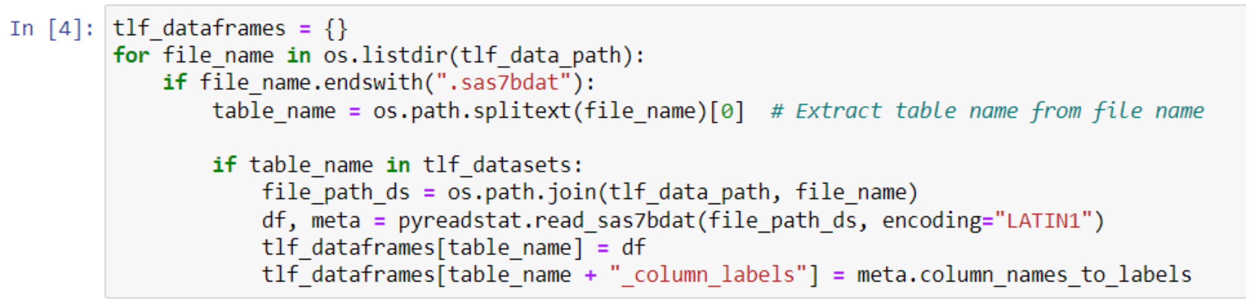
Next, the script defines the folder paths and the datasets of interest. It also initializes the empty dictionaries (Dataframe s and tlf_Dataframe s) to store the resulting data frames.



```
In [3]: dataframes = {}
        for file_name in os.listdir(folder_path):
            if file_name.endswith(".sas7bdat"):
                table_name = os.path.splitext(file_name)[0]  # Extract table name from file name

                if table_name in sel_datasets:
                    file_path_ds = os.path.join(folder_path, file_name)
                    df, meta = pyreadstat.read_sas7bdat(file_path_ds, encoding="LATIN1")
                    dataframes[table_name] = df
                    dataframes[table_name + "_column_labels"] = meta.column_names_to_labels
        df_df = dataframes['adsl']
        print(df.columns)
```

Display 4. Reading and converting SAS ADaM datasets into Python data frames.

The script iterates each file in the folder specified by folder_path. For each file ending with sas7bdat extension, it extracts the table name from the file name and checks if it matches any of the datasets specified in the sel_datasets. If a match is found, the script reads the SAS® data file using pyreadstat.read_sas7bdat(), converts it into a pandas dataframe (df), and stores it in a database dictionary. Additionally, it stores the column labels associated with the dataframe in a separate dictionary entry with keys suffixed by _column_labels.



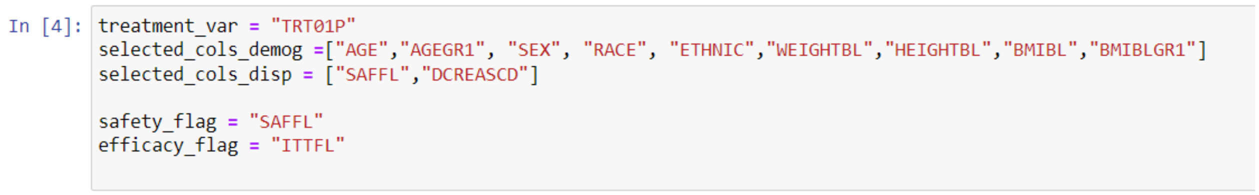
```
In [4]: tlf_dataframes = {}
        for file_name in os.listdir(tlf_data_path):
            if file_name.endswith(".sas7bdat"):
                table_name = os.path.splitext(file_name)[0]  # Extract table name from file name

                if table_name in tlf_datasets:
                    file_path_ds = os.path.join(tlf_data_path, file_name)
                    df, meta = pyreadstat.read_sas7bdat(file_path_ds, encoding="LATIN1")
                    tlf_dataframes[table_name] = df
                    tlf_dataframes[table_name + "_column_labels"] = meta.column_names_to_labels
```

Display 5. Reading and converting SAS TLF output datasets into Python data frames.

Similarly, the script iterates each file in the TLF data-path (tlf_data_path). It follows the same procedure as before to read the SAS TLF data files, but this time it checks if the table name matches any of the datasets specified in the tlf_datasets(output-ready QC dataset). If a match is found, the resulting dataframe is stored in the tlf_Dataframe dictionary along with the corresponding column labels.

DEFINING TREATMENT VARIABLES, SELECTING COLUMNS, POPULATION FLAGS, AND FILE LOCATIONS.

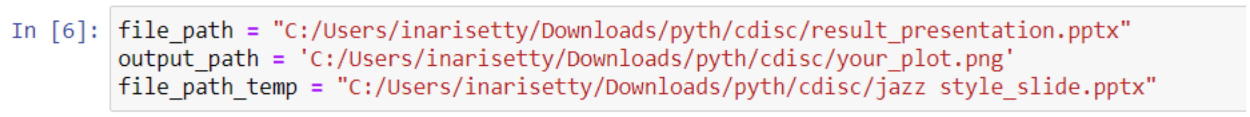
Next, we define treatment variables, selecting columns needed for tables, and specify flags for safety and efficacy assessments.



```
In [4]: treatment_var = "TRT01P"
        selected_cols_demog =["AGE","AGEGR1", "SEX", "RACE", "ETHNIC","WEIGHTBL","HEIGHTBL","BMIBL","BMIBLGR1"]
        selected_cols_disp = ["SAFFL","DCREASCD"]

        safety_flag = "SAFFL"
        efficacy_flag = "ITTFL"
```

Display 6. Defining treatment variables, population flags, and column selection.

Next, we specify the file paths for the input and output files required to generate the presentation slides and save the plot images.



```
In [6]: file_path = "C:/Users/inarisetty/Downloads/pyth/cdisc/result_presentation.pptx"
        output_path = 'C:/Users/inarisetty/Downloads/pyth/cdisc/your_plot.png'
        file_path_temp = "C:/Users/inarisetty/Downloads/pyth/cdisc/jazz style_slide.pptx"
```

Display 7. The locations of the paths were defined.

The file_path_temp variable refers to the company standard PowerPoint template, which serves as a blueprint for the presentation slides generated by the tool. This template embodies a company's branding, formatting, and stylistic preferences, ensuring consistency and professionalism across all presentations.

By specifying the company standard power-point template file, the tool can seamlessly integrate the generated data into the existing design and layout. This ensures that the output presentation maintains the same visual identity and adheres to established standards, which are crucial for effective communication and conveying a unified message.

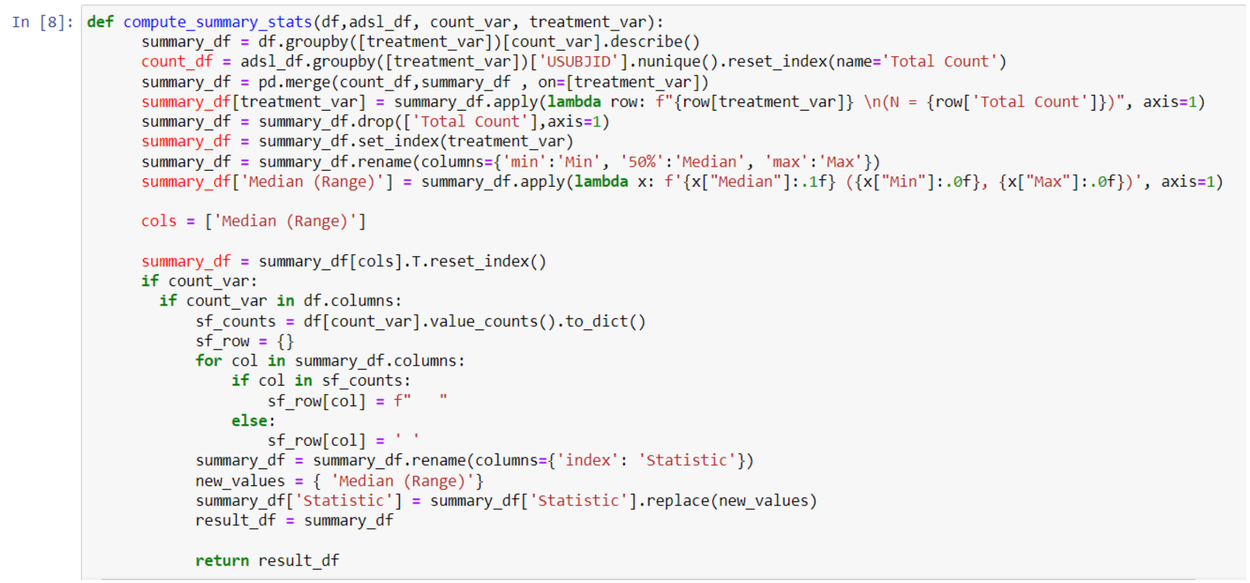
Using the company standard template also streamlines the presentation creation process by eliminating the need to manually format slides or adjust the settings. The tool automatically applies predefined styles, fonts, colors, and layouts from the template, saving time and effort, while ensuring a polished and cohesive presentation.

Overall, leveraging the company's standard PowerPoint template with all its settings ensures that the output data are presented in a consistent and professional manner, aligning with the company's branding guidelines and enhancing the impact and credibility of the presentation.

DEVELOPING PYTHON FUNCTIONS TO PRODUCE FREQUENCY COUNTS, SUMMARY STATISTICS, AND COLUMN ARRANGEMENTS.

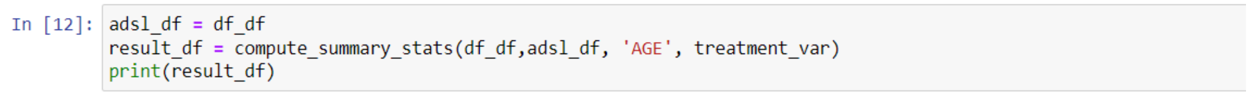
Compute Summary Stats function

The compute_summary_stats function streamlines the process of generating summary statistics from data. First, it groups the main data frame by the treatment variable and calculates descriptive statistics for the specified variable of interest, resulting in a summary dataframe (summary_df). Additionally, it calculates the total number of subjects in each treatment group using ADSL data and merges this information into the summary dataframe for clarity. The function then formats the summary data, renames columns for readability, computes the median and range (and can include additional statistics if needed) for each treatment group, and transposes the dataframe for better presentation. It also handles missing data, if specified, ensuring that all relevant statistics are captured. Ultimately, the function returns a dataframe (result_df) containing the computed summary statistics, empowering users to gain insight into their data distribution and characteristics across different treatment groups.



```
In [8]: def compute_summary_stats(df,adsl_df, count_var, treatment_var):
              summary_df = df.groupby([treatment_var])[count_var].describe()
              count_df = adsl_df.groupby([treatment_var])['USUBJID'].nunique().reset_index(name='Total Count')
              summary_df = pd.merge(count_df,summary_df , on=[treatment_var])
              summary_df[treatment_var] = summary_df.apply(lambda row: f"{row[treatment_var]} \n(N = {row['Total Count']})", axis=1)
              summary_df = summary_df.drop(['Total Count'],axis=1)
              summary_df = summary_df.set_index(treatment_var)
              summary_df = summary_df.rename(columns={'min':'Min', '50%':'Median', 'max':'Max'})
              summary_df['Median (Range)'] = summary_df.apply(lambda x: f'{x["Median"]:.1f} ({x["Min"]:.0f}, {x["Max"]:.0f})', axis=1)

              cols = ['Median (Range)']

              summary_df = summary_df[cols].T.reset_index()
              if count_var:
                if count_var in df.columns:
                    sf_counts = df[count_var].value_counts().to_dict()
                    sf_row = {}
                    for col in summary_df.columns:
                        if col in sf_counts:
                            sf_row[col] = f"    "
                        else:
                            sf_row[col] = ' '
                    summary_df = summary_df.rename(columns={'index': 'Statistic'})
                    new_values = { 'Median (Range)'}
                    summary_df['Statistic'] = summary_df['Statistic'].replace(new_values)
                    result_df = summary_df

                    return result_df
```

Display 8. Python functions to generate the summary statistics.



```
In [12]: adsl_df = df_df
         result_df = compute_summary_stats(df_df,adsl_df, 'AGE', treatment_var)
         print(result_df)
```

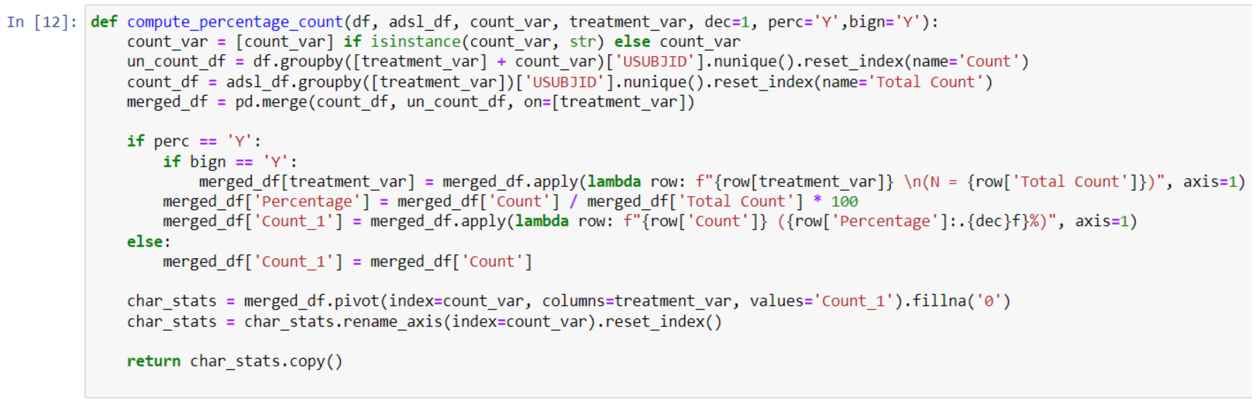
Display 9. Example call of the compute_summary_stats function.

```
TRT01P       Statistic Placebo \n(N = 86) Xanomeline High Dose \n(N = 84)  \
0       Median (Range)      76.0 (52, 89)                    76.0 (56, 88)

TRT01P Xanomeline Low Dose \n(N = 84)
0                       77.5 (51, 88)
```

Output 1. Output from a compute_summary_stats Function

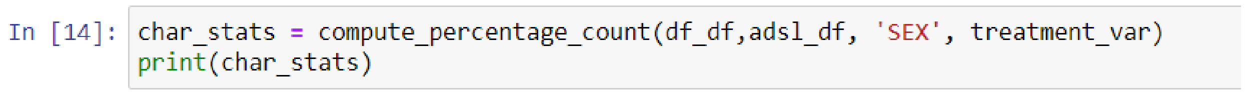
Compute Percentage Count function

The compute_percentage_count function simplifies the computation of percentage counts from the data, such as the frequency counts in SAS®. However, it extends this functionality by providing percentages and leveraging the treatment column for grouping. By grouping the main data frame according to the treatment variable and specified count variable(s), the unique subject count within each group was calculated. Concurrently, it computes the total subject count per treatment group using the demographic dataframe and merges these counts to facilitate subsequent calculations. Depending on the specified parameters, the function calculates the percentage of subjects within each group and formats the counts.



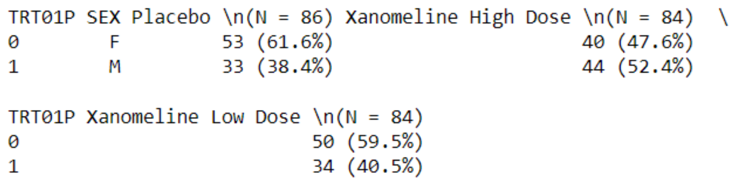
```
In [12]: def compute_percentage_count(df, adsl_df, count_var, treatment_var, dec=1, perc='Y',bign='Y'):
             count_var = [count_var] if isinstance(count_var, str) else count_var
             un_count_df = df.groupby([treatment_var] + count_var)['USUBJID'].nunique().reset_index(name='Count')
             count_df = adsl_df.groupby([treatment_var])['USUBJID'].nunique().reset_index(name='Total Count')
             merged_df = pd.merge(count_df, un_count_df, on=[treatment_var])

             if perc == 'Y':
                 if bign == 'Y':
                     merged_df[treatment_var] = merged_df.apply(lambda row: f"{row[treatment_var]} \n(N = {row['Total Count']})", axis=1)
                 merged_df['Percentage'] = merged_df['Count'] / merged_df['Total Count'] * 100
                 merged_df['Count_1'] = merged_df.apply(lambda row: f"{row['Count']} ({row['Percentage']:.{dec}f}%)", axis=1)
             else:
                 merged_df['Count_1'] = merged_df['Count']

             char_stats = merged_df.pivot(index=count_var, columns=treatment_var, values='Count_1').fillna('0')
             char_stats = char_stats.rename_axis(index=count_var).reset_index()

             return char_stats.copy()
```

Display 10. Python functions to generate the frequency count statistics.



```
In [14]: char_stats = compute_percentage_count(df_df,adsl_df, 'SEX', treatment_var)
         print(char_stats)
```

Display 11. **Example call of compute_percentage_count function.**

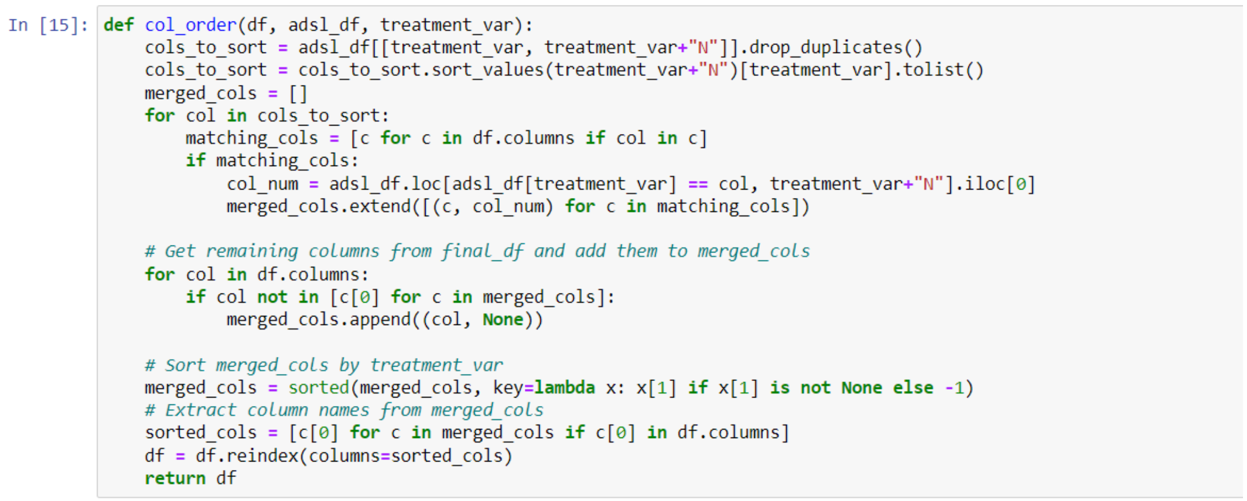


```
TRT01P SEX Placebo \n(N = 86) Xanomeline High Dose \n(N = 84)  \
0        F          53 (61.6%)                        40 (47.6%)
1        M          33 (38.4%)                        44 (52.4%)

TRT01P Xanomeline Low Dose \n(N = 84)
0                          50 (59.5%)
1                          34 (40.5%)
```

Output 2. Output from compute_percentage_count Function

Column order function

This function, col_order, helps arrange the columns in the dataframe based on the specified treatment values. It looks at both your main dataframe and the ADSL dataframe that contains treatment data. First, it finds unique treatment values and sorts them according to their numeric order. It then goes through each treatment value, finds matching columns in the dataframe, and assigns a numerical order to each treatment value. After sorting the columns according to the treatment variable, the dataframe was updated with the newly organized columns. Overall, col_order makes it easier to analyze the data by ensuring a consistent column order based on treatment values.

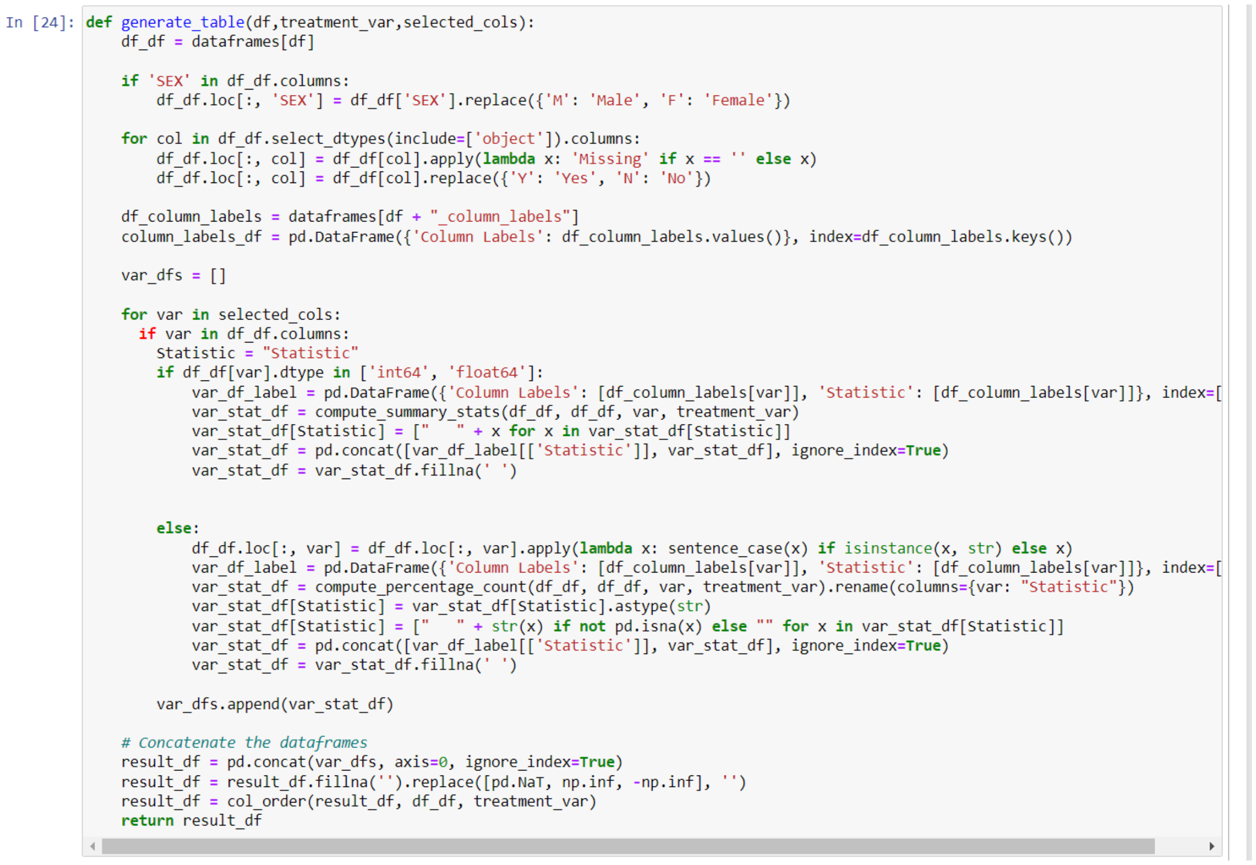


```
In [15]: def col_order(df, adsl_df, treatment_var):
             cols_to_sort = adsl_df[[treatment_var, treatment_var+"N"]].drop_duplicates()
             cols_to_sort = cols_to_sort.sort_values(treatment_var+"N")[treatment_var].tolist()
             merged_cols = []
             for col in cols_to_sort:
                 matching_cols = [c for c in df.columns if col in c]
                 if matching_cols:
                     col_num = adsl_df.loc[adsl_df[treatment_var] == col, treatment_var+"N"].iloc[0]
                     merged_cols.extend([(c, col_num) for c in matching_cols])

             # Get remaining columns from final_df and add them to merged_cols
             for col in df.columns:
                 if col not in [c[0] for c in merged_cols]:
                     merged_cols.append((col, None))

             # Sort merged_cols by treatment_var
             merged_cols = sorted(merged_cols, key=lambda x: x[1] if x[1] is not None else -1)
             # Extract column names from merged_cols
             sorted_cols = [c[0] for c in merged_cols if c[0] in df.columns]
             df = df.reindex(columns=sorted_cols)
             return df
```

Display 11. Python function to order the treatment columns based on numeric treatment column.

COMBINING ALL THE PREVIOUSLY DISCUSSED FUNCTIONS INTO A SINGLE COHESIVE SOLUTION.

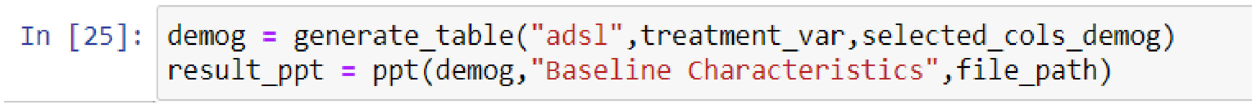
The generate_table function integrates various functionalities to process and summarize data. It begins by fetching the dataframe corresponding to the specified dataset (df) and adjusting specific columns if necessary. For instance, it converts abbreviation codes to meaningful labels, handles missing values, and standardizes the format of the categorical variables. Next, it retrieves the column labels and uses them as row labels for each category. It then iterates over each selected column to compute the relevant statistics. For numerical variables, summary statistics such as the Median and Range (required for our outputs) are used. For categorical variables, the percentage counts across the treatment groups were computed. The computed statistics for each column were then concatenated into a single dataframe(result_df). Finally, the columns are ordered based on the numeric treatment variable to ensure consistency in presentation across the different analyses.



```
In [24]: def generate_table(df,treatment_var,selected_cols):
             df_df = dataframes[df]

             if 'SEX' in df_df.columns:
                 df_df.loc[:, 'SEX'] = df_df['SEX'].replace({'M': 'Male', 'F': 'Female'})

             for col in df_df.select_dtypes(include=['object']).columns:
                 df_df.loc[:, col] = df_df[col].apply(lambda x: 'Missing' if x == '' else x)
                 df_df.loc[:, col] = df_df[col].replace({'Y': 'Yes', 'N': 'No'})

             df_column_labels = dataframes[df + "_column_labels"]
             column_labels_df = pd.DataFrame({'Column Labels': df_column_labels.values()}, index=df_column_labels.keys())

             var_dfs = []

             for var in selected_cols:
               if var in df_df.columns:
                 Statistic = "Statistic"
                 if df_df[var].dtype in ['int64', 'float64']:
                     var_df_label = pd.DataFrame({'Column Labels': [df_column_labels[var]], 'Statistic': [df_column_labels[var]]}, index=[
                     var_stat_df = compute_summary_stats(df_df, df_df, var, treatment_var)
                     var_stat_df[Statistic] = ["    " + x for x in var_stat_df[Statistic]]
                     var_stat_df = pd.concat([var_df_label[['Statistic']], var_stat_df], ignore_index=True)
                     var_stat_df = var_stat_df.fillna(' ')


                 else:
                     df_df.loc[:, var] = df_df.loc[:, var].apply(lambda x: sentence_case(x) if isinstance(x, str) else x)
                     var_df_label = pd.DataFrame({'Column Labels': [df_column_labels[var]], 'Statistic': [df_column_labels[var]]}, index=[
                     var_stat_df = compute_percentage_count(df_df, df_df, var, treatment_var).rename(columns={var: "Statistic"})
                     var_stat_df[Statistic] = var_stat_df[Statistic].astype(str)
                     var_stat_df[Statistic] = ["    " + str(x) if not pd.isna(x) else "" for x in var_stat_df[Statistic]]
                     var_stat_df = pd.concat([var_df_label[['Statistic']], var_stat_df], ignore_index=True)
                     var_stat_df = var_stat_df.fillna(' ')

                 var_dfs.append(var_stat_df)

             # Concatenate the dataframes
             result_df = pd.concat(var_dfs, axis=0, ignore_index=True)
             result_df = result_df.fillna('').replace([pd.NaT, np.inf, -np.inf], '')
             result_df = col_order(result_df, df_df, treatment_var)
             return result_df
```

Display 12. Integrating all functions into a single function generates both frequency counts and summary statistics.

Here is the final function call encapsulating the entire process: it generates a table summarizing the baseline demographic characteristics. The generated table data frame is then used in the ppt function to produce a PowerPoint presentation titled "Baseline Characteristics," containing the summarized demographic data, ready for further analysis and presentation.



```
In [25]: demog = generate_table("adsl",treatment_var,selected_cols_demog)
         result_ppt = ppt(demog,"Baseline Characteristics",file_path)
```

Display 13. Example calls of the generate_table and ppt functions.

Utilizing the generate_table function, the selection of columns and datasets can be tailored, allowing for the creation of tables with different configurations.

DEVLOPING PYTHON FUNCTION TO GENERATE SPIDER PLOT.

To generate a spider plot PowerPoint presentation, the dataset was filtered, and specific values were set to zero for baseline variables. Then, organize the data and create a spider plot using Plotly. Each treatment variable value was assigned a unique color, facilitating easy interpretation. Subsequently, each subject's data are plotted on the graph, with treatment duration on the x-axis and percent change from baseline on the y-axis. Finally, the plot was embedded into a PowerPoint presentation for seamless sharing and presentation.

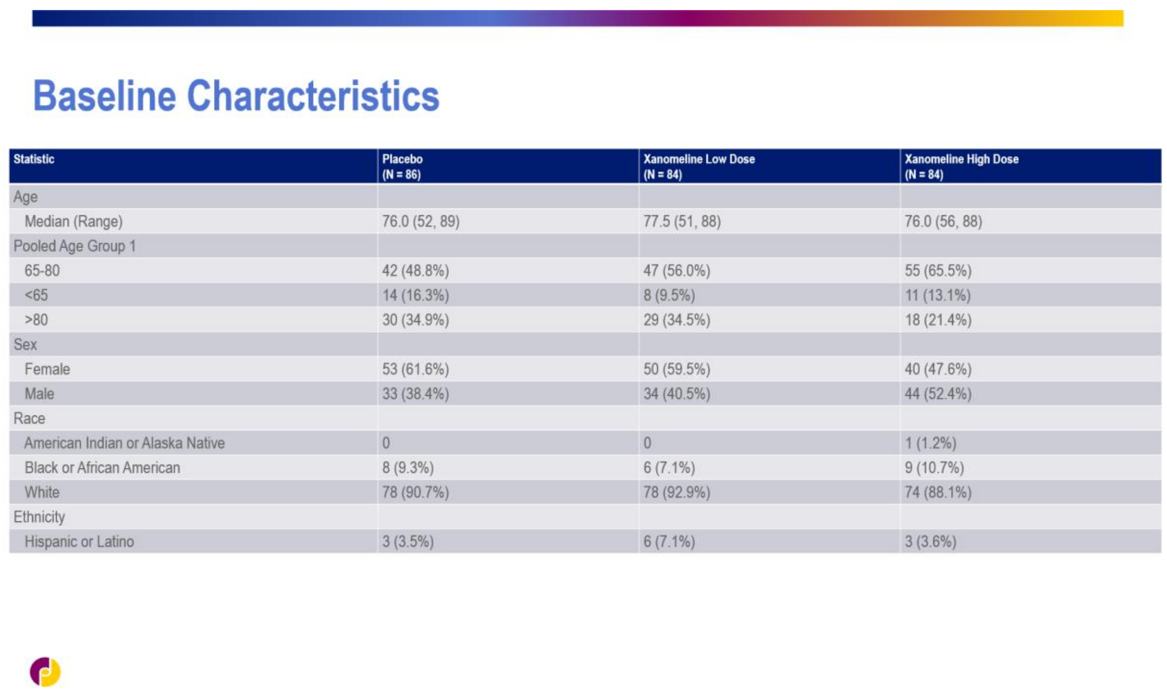
```
In [66]: def spider_plot_ppt(df, treatment_var, baseline_var,baseline_var_values, day, pchg, title, file_path,output_path):
    df_filtered = df.copy()
    df_filtered.loc[df_filtered[baseline_var].str.upper() == baseline_var_values, day] = 0
    df_filtered.loc[df_filtered[baseline_var].str.upper() == baseline_var_values, pchg] = 0
    df_filtered.set_index(day, inplace=True)
    fig = go.Figure()
    treatment_var_values = df_filtered[treatment_var].unique()
    treatment_var_label = column_labels.get(treatment_var, treatment_var)
    colors = px.colors.qualitative.Plotly[:len(treatment_var_values)]
    color_map = dict(zip(treatment_var_values, colors))
    for subj, group in df_filtered.groupby('SUBJID'):
            group.sort_values(by=day, inplace=True)
            treatment_var_value = group[treatment_var].iloc[0]
            color = color_map[treatment_var_value]
            fig.add_trace(go.Scatter(x=group.index,y=group[pchg],
                    mode='lines+markers',
                    name=f'{treatment_var_label}: {treatment_var_value}',
                    marker=dict(size=5, color=color,),
                    line=dict(width=1, color=color,),
                    hovertemplate='USUBJID: %{text}<br>PCHGB: %{y}<extra></extra>',
                    text=[subj] * len(group),
                    showlegend=False))
    for treat_value, color in color_map.items():
            fig.add_trace(go.Scatter(x=[None],y=[None],
                    mode='markers',
                    name=f'{treatment_var_label}: {treat_value}',
                    marker=dict(
                        size=12,
                        line=dict(width=2, color=color),
                        color=color)))
    fig.update_layout(xaxis_title='Treatment Duration (months)',
            yaxis_title='Percent Change from Baseline in Sum of Diameters',
            legend=dict(orientation='v',
                yanchor='top',
                y=1,
                xanchor='right',
                x=1,
                bgcolor='white',
                bordercolor='gray',
                borderwidth=1,),
            xaxis=dict(showgrid=True, gridcolor='lightgray'),
            yaxis=dict(showgrid=True, gridcolor='lightgray'),
            margin=dict(l=20, r=20, t=60, b=20),
            paper_bgcolor='white',
            plot_bgcolor='white',
            width=1000,
            height=600)
    add_plot_to_ppt(fig, title, file_path,output_path)
```

Display 14. Python function to generate spider_plot_ppt.

```
In [67]: spider_df['ADY'] = spider_df['ADY']/30.4375 #convert to months
pchg='PCHG'
day = "ADY"
baseline_var = 'AVISIT'
baseline_var_values = 'BASELINE'

spider_plot_ppt(spider_df, treatment_var, baseline_var,baseline_var_values, day, pchg, "Spider plot", file_path,output_path)
```

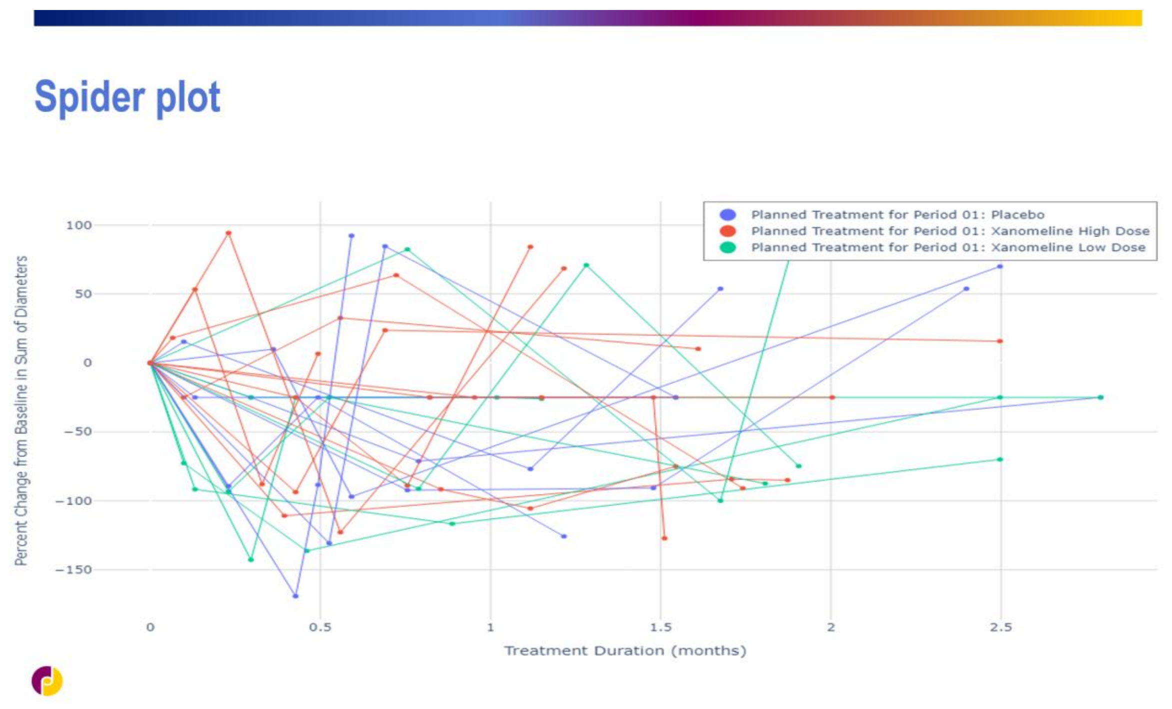
Display 15. Example Call of spider_plot_ppt function.

Here is a presentation generated from these functions, providing a polished and professional overview of the key baseline characteristics and spider plot. This PowerPoint presentation offers visually appealing slides, presenting the summarized data and graphs in a clear and concise manner. With its sleek design and organized layout, the presentation is well suited for use in meetings, conferences, and other professional settings. It provides a streamlined platform for presenting important information, enhancing communication, and facilitating productive discussion.



Baseline Characteristics

Statistic	Placebo (N = 86)	Xanomeline Low Dose (N = 84)	Xanomeline High Dose (N = 84)
Age			
Median (Range)	76.0 (52, 89)	77.5 (51, 88)	76.0 (56, 88)
Pooled Age Group 1			
65-80	42 (48.8%)	47 (56.0%)	55 (65.5%)
<65	14 (16.3%)	8 (9.5%)	11 (13.1%)
>80	30 (34.9%)	29 (34.5%)	18 (21.4%)
Sex			
Female	53 (61.6%)	50 (59.5%)	40 (47.6%)
Male	33 (38.4%)	34 (40.5%)	44 (52.4%)
Race			
American Indian or Alaska Native	0	0	1 (1.2%)
Black or African American	8 (9.3%)	6 (7.1%)	9 (10.7%)
White	78 (90.7%)	78 (92.9%)	74 (88.1%)
Ethnicity			
Hispanic or Latino	3 (3.5%)	6 (7.1%)	3 (3.6%)



Spider plot

Output 2. Final output slides on presentation deck.

Note: The data used to generate the spider plot were simulated and may not accurately represent the specific context of an oncology trial. This is intended solely for illustration purposes.

CONCLUSION

In conclusion, the process outlined in this conversation demonstrates a comprehensive approach for analyzing and presenting data using Python. Each step was meticulously planned and executed by reading the SAS datasets to generate PowerPoint presentations. By leveraging Python libraries and custom functions, users can efficiently process data, create visualizations, and produce professional presentations. Whether summarizing the data or plotting complex data trends, the tools and techniques discussed here offer versatile solutions for data analysis and communication. Once Python functions are set up, they can be easily adapted and applied to various other studies, providing a reusable framework for data analysis and presentation. Leveraging open-source technologies such as Python and its rich library ecosystem enhances accessibility and flexibility, allowing users to harness the collective knowledge and resources of the developer community. This emphasis on open-source technology promotes collaboration and innovation, ensuring transparency and reproducibility in data analysis workflows. By embracing open-source tools, researchers can efficiently tackle diverse analytical challenges while contributing to a culture of shared knowledge and continuous improvements in the field of data science.

REFERENCES

Pandas Development Team. Pandas: Powerful data analysis tools for Python, v1.3.3. Available at: https://pandas.pydata.org/docs/

Rousselet, G. Pyreadstat: Read and Write SPSS and Stata files into/from pandas data frames. Available at: https://github.com/Roche/pyreadstat

Harris, C.R., Millman, K.J., van der Walt, S.J., et al. Array programming using NumPy. Nature 585, 357–362 (2020). Available at: https://www.num.py.org/

Scanny. Python-pptx Documentation. Available at: https://python-pptx.readthedocs.io/en/latest/index.html

Plotly Technologies, Inc. Python Graphing Library version 5.3.1. Available at: https://plotly.com/python/

Plotly Technologies, Inc. Kaleido: Static image export for web-based visualization libraries. Available at: https://github.com/plotly/Kaleido

Python Software Foundation. Colorsys Documentation. Available at: https://docs.python.org/3/library/colorsys.html

CDISC SDTM-ADaM Pilot Project. (n.d.). CDISC Pilot Project: Updated Pilot Submission Package. GitHub. https://github.com/cdisc-org/sdtm-adam-pilot-project/tree/master/updated-pilot-submission-package/900172/m5/datasets/cdiscpilot01/analysis/adam/datasets

ACKNOWLEDGMENTS

I would like to express my gratitude to Jagan Mogan Achi for his encouragement and support throughout the process of learning Python and undertaking this project. His guidance and motivation have been invaluable, and I am truly grateful for his assistance.

CONTACT INFORMATION

Your comments and questions have been valued and encouraged. Contact the author at:

Indraneel Narisetty
Jazz Pharmaceuticals
201-450-1480
Indraneel.narisetty@jazzpharma.com