```python
#Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
import plotly.io as pio
pio.templates.default = "none"

class PlotGraphs:

    #function to plot vertical bar plot
    @staticmethod
    def VerticalBarPlot(inDF,inValueI, inLabelI,title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        fig = px.bar(df, x=labels, y=vals,
                hover_data=[labels], color=vals, height=400,
                title=title, color_continuous_scale='Blugrn')
        fig.show()

    #horizontal bar plot
    @staticmethod
    def HorizontalBarPlot(inDF,inValueI, inLabelI,title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        fig = px.bar(df, x=vals, y=labels,
                hover_data=[labels], color=vals, height=400,
                title=title, color_continuous_scale='Blugrn', orientation='h')
        fig.show()

    #Polar Bar Plot
    @staticmethod
    def PolarBarPlot(inDF,inValueI, inLabelI,title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        fig = px.bar_polar(df, r=vals, theta=labels,
                    color=vals, title=title,
                    color_continuous_scale='Blugrn')
        fig.show()

    #circular bar graph
    @staticmethod
    def AddLabels(angles, values, labels, offset, ax):
        for ang,val,label, in zip(angles, values, labels):
            rot, align = PlotGraphs.GetLabelRot(ang, offset)
            ax.text(x=ang,y=val+4,s=label,ha=align,va="center",rotation=rot,rotation_mode="anchor")

    @staticmethod
    def GetLabelRot(ang, offset):
        rot = np.rad2deg(ang + offset)
        if ang <= np.pi:
            align = "right"
            rot = rot + 180
        else: align = "left"
        return rot, align

    @staticmethod
    def get_color(name, number):
        pal = list(sns.color_palette(palette=name, n_colors=number).as_hex())
        return pal

    @staticmethod
    def CircularBarPlot(inDf,inValueI, inLabelI):
        vals = inDf[inDf.columns[inValueI]]
        vals = (vals/vals.max()) * 80
        labels = inDf[inDf.columns[inLabelI]]
        cWidth = 2 * np.pi / len(vals)
        colors = PlotGraphs.get_color("blend:#AFE1AF,#1c4a60", len(inDf))
        colors.reverse()
        fig, ax = plt.subplots(figsize=(20,10),subplot_kw={"projection":"polar"})
        ax.set_theta_offset(np.pi/2)
        ax.set_ylim(-100, 100)
        ax.set_frame_on(False)
        ax.set_xticks([])
        ax.set_yticks([])
        angs = np.linspace(0, 2 * np.pi, len(inDf), endpoint=False)
        ax.bar(angs,vals,width=cWidth,linewidth=2,color=colors,edgecolor="white")
        PlotGraphs.AddLabels(angs, vals, labels, np.pi/2, ax)

    #slider bar plot
    @staticmethod
    def SliderBarPlot(inDF, inValueI, inLabelI, inSizeI, title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        size = df[df.columns[inSizeI]]
        fig = px.bar(df, x=labels, y=vals, animation_frame=size, color=vals, color_continuous_scale='Blugrn',
                title=title)
```

```
        fig.update_layout(margin=dict(l=20, r=20, t=20, b=200))
        fig['layout']['updatemenus'][0]['pad']=dict(r= 10, t= 150)
        fig['layout']['sliders'][0]['pad']=dict(r= 10, t= 150,)

        fig["layout"].pop("updatemenus") # optional, drop animation buttons
        fig.show()

    #Pie chart
    @staticmethod
    def PieChart(inDF, inValueI, inLabelI, title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        colors = PlotGraphs.get_color("blend:#AFE1AF,#1c4a60", len(df))
        colors.reverse()
        fig = go.Figure(data=[go.Pie(labels=labels, values=vals,
                                title = title, marker_colors=colors
                            )])
        fig.show()

    #Exploded Pie chart
    @staticmethod
    def ExplodedPieChart(inDF, inValueI, inLabelI, title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        pull = [0.2]
        for i in range(len(df)-1):
            pull.append(0)
        colors = PlotGraphs.get_color("blend:#AFE1AF,#1c4a60", len(df))
        colors.reverse()
        fig = go.Figure(data=[go.Pie(labels=labels, values=vals, pull=pull,
                                title = title, marker_colors=colors
                            )])
        fig.show()

    #donut chart
    @staticmethod
    def DonutChart(inDF, inValueI, inLabelI, title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]
        colors = PlotGraphs.get_color("blend:#AFE1AF,#1c4a60", len(df))
        colors.reverse()

        fig = go.Figure(data=[go.Pie(labels=labels, values=vals, hole=.3,title = title,
                            marker_colors = colors)])
        fig.update_traces(textinfo='value')
        fig.show()

    #Radial Plot
    @staticmethod
    def RadialPlot(inDF,inValueI, inLabelI,title):
        df = inDF.sort_values(by=inDF.columns[inValueI],ascending=False)
        vals = df[df.columns[inValueI]]
        labels = df[df.columns[inLabelI]]

        plt.gcf().set_size_inches(8, 8)
        sns.set_style('darkgrid')
        label = df[df.columns[inLabelI]].to_numpy()

        colors = PlotGraphs.get_color("blend:#AFE1AF,#1c4a60", len(df))
        colors.reverse()

        #set max value
        max_val = max(df[df.columns[inValueI]])*1.01
        ax = plt.subplot(projection='polar')

        #set the subplot
        ax.set_theta_zero_location('N')
        ax.set_theta_direction(1)
        ax.set_rlabel_position(0)
        ax.set_thetagrids([], labels=[])
        ax.set_rgrids(range(len(df)), labels = label)

        #set the projection
        for i in range(len(df)):
            ax.barh(i, list(df[df.columns[inValueI]])[i]*2*np.pi/max_val,
                    label=list(df[df.columns[inLabelI]])[i]
                    ,color=colors[i]
                )
        plt.legend(bbox_to_anchor=(1, 1), loc=2,labels=label)
        plt.title(title)
        plt.show()

    #scatter plot
    @staticmethod
    def ScaterPlot(inDF, cordinates, title):
        '''
        For corditnates, plese provide input in the following format(array):
        [x-axis column number, y-axis column number, column number for color, column number for size]
        ex: [0,1,2,3]
```

```
        The minimum size required is 2 i.e. you need to atleast provide x and y axis values for the funtion to work.
        Color and Size are optional.
        The maximum length of the array permitted is 4.

        Make sure that the data type of the column for Color and Size is integer or float.
        '''
        df = inDF
        if len(cordinates) < 2:
            return "Error: The size of the array provided is less than 2. Please provide atleast 2 column numbers."
        elif len(cordinates) == 2:
            fig = px.scatter(df,
                             x=df[df.columns[cordinates[0]]],
                             y=df[df.columns[cordinates[1]]],
                             color=df[df.columns[cordinates[1]]],
                             color_continuous_scale='Blugrn')
        elif len(cordinates) == 3:
            fig = px.scatter(df,
                             x=df[df.columns[cordinates[0]]],
                             y=df[df.columns[cordinates[1]]],
                             color=df[df.columns[cordinates[2]]],
                             color_continuous_scale='Blugrn')
        elif len(cordinates) == 4:
            fig = px.scatter(df,
                             x=df[df.columns[cordinates[0]]],
                             y=df[df.columns[cordinates[1]]],
                             color=df[df.columns[cordinates[2]]],
                             size=df[df.columns[cordinates[3]]],
                             color_continuous_scale='Blugrn')
        elif len(cordinates) > 4:
            return "Error: The size of the array provided is greater than 4. Please provide a minimum of 2 and a maximum of 4 column numbers."
        fig.update_layout(title=title)
        fig.show()

    #line scatter plot
    @staticmethod
    def LineScaterPlot(inDF, cordinates, title):
        '''
        For corditnates, plese provide input in the following format(array):
        [x-axis column number, y-axis column number, column number for density]
        ex: [0,1,2]

        The minimum size required is 2 i.e. you need to atleast provide x and y axis values for the funtion to work.
        Color and Size are optional.
        The maximum length of the array permitted is 3.

        Make sure that the data type of the column for Density is integer or float.
        '''
        df = inDF
        fig = go.Figure()
        if len(cordinates) < 2:
            return "Error: The size of the array provided is less than 2. Please provide atleast 2 column numbers."
        elif len(cordinates) == 2:
            fig.add_trace(go.Scatter(x=df[df.columns[cordinates[0]]],
                                     y=df[df.columns[cordinates[1]]],
                                     mode='lines+markers',
                                     marker=dict(size=10, color='#1c4a60'),
                                     line = dict(color = '#AFE1AF',width = 3)))
        elif len(cordinates) == 3:
            fig.add_trace(go.Scatter(x=df[df.columns[cordinates[0]]],
                                     y=df[df.columns[cordinates[1]]],
                                     mode='lines+markers',
                                     marker=dict(size=10, color='#1c4a60'),
                                     line = dict(color = '#AFE1AF',width = 3),
                                     text=df[df.columns[cordinates[2]]]))
        elif len(cordinates) > 3:
            return "Error: The size of the array provided is greater than 3. Please provide a minimum of 2 and a maximum of 3 column numbers."
        fig.update_layout(title=title)
        fig.show()
```