```
"""
# Author: Komal Shahid
# Course: DSC640 - Data Presentation and Visualization
# Assignment: Milestone 3 - Interactive Dashboard and Visualizations
# Date: January 2024
#
# Description: This script creates an interactive dashboard and static visualizations
# analyzing childcare costs across different states, income levels, and time periods.
# The analysis includes geographic distribution, cost burden analysis, correlation
# networks, and temporal trends.
"""

import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import seaborn as sns
from fpdf import FPDF
from plotly.subplots import make_subplots
import os
from datetime import datetime

class ChildcareCostAnalysis:
    def __init__(self):
        """Initialize with the childcare dataset"""
        self.data = pd.read_excel(’../../../data/nationaldatabaseofchildcareprices.xlsx’)
        # Create output directory
        os.makedirs(’../output’, exist_ok=True)
        os.makedirs(’../output/temp’, exist_ok=True)

        # Add custom color schemes
        self.color_schemes = {
            ’main’: px.colors.qualitative.Set3,
            ’sequential’: px.colors.sequential.Viridis,
            ’diverging’: px.colors.diverging.RdYlBu_r
        }

        # Preprocess data
        self.preprocess_data()

    def preprocess_data(self):
        """Preprocess the data for visualizations"""
        # Create income brackets with more intuitive labels
        self.data[’Income_Bracket’] = pd.qcut(
            self.data[’MHI_2018’],
            q=5,
            labels=[’Very Low Income’, ’Low Income’, ’Middle Income’, ’Upper Middle’, ’Hi
gh Income’]
        )
        #data cleaning
        # Calculate annual costs and cost ratios
        self.data[’Annual_Cost_Infant’] = self.data[’MCInfant’] * 12
        self.data[’Cost_Income_Ratio’] = (self.data[’Annual_Cost_Infant’] / self.data[’MH
I_2018’]) * 100

        # Convert StudyYear to datetime
        self.data[’Year’] = pd.to_datetime(self.data[’StudyYear’].astype(str), format=’%Y
’)

        # Clean up any missing values
        self.data = self.data.dropna(subset=[’MCInfant’, ’MCToddler’, ’MCPreschool’, ’MHI
_2018’])

    def create_enhanced_dashboard(self):
        """Create an enhanced interactive dashboard with key visualizations"""

        fig = make_subplots(
            rows=3, cols=2,
            subplot_titles=(
                '<b>Geographic Distribution of Childcare Costs</b>',
                '<b>Cost Burden Across States</b>',
                '<b>Relationship Network of Key Factors</b>',
                '<b>Income Level Cost Distribution</b>',
                '<b>Variable Correlation Analysis</b>',
                '<b>State Cost Impact Analysis</b>'
            ),
            specs=[
                [{"type": "choropleth"}, {"type": "scattergeo"}],
                [{"type": "scatter"}, {"type": "pie"}],
                [{"type": "scatter"}, {"type": "sunburst"}]
            ],
            vertical_spacing=0.15,
            horizontal_spacing=0.15  # Increased spacing
        )

        #  Choropleth Map
        state_costs = self.data.groupby('State_Abbreviation').agg({
            'MCInfant': 'mean',
            'MHI_2018': 'mean'
        }).reset_index()

        # Dynamic bubble sizing
        state_metrics = state_costs.copy()
        state_metrics['Annual_Cost'] = state_metrics['MCInfant'] * 12
        state_metrics['CostIncomeRatio'] = (state_metrics['Annual_Cost'] / state_metrics[
'MHI_2018']) * 100

        state_metrics['bubble_size'] = 20 + (
            (state_metrics['CostIncomeRatio'] - state_metrics['CostIncomeRatio'].min()) *

            (50 - 20) / (state_metrics['CostIncomeRatio'].max() - state_metrics['CostInco
meRatio'].min())
        )

        # Add choropleth with left-side legend
        fig.add_trace(
            go.Choropleth(
                locations=state_costs['State_Abbreviation'],
                z=state_costs['MCInfant'],
                locationmode='USA-states',
                colorscale=self.color_schemes['sequential'],
                colorbar=dict(
                    title="Monthly Cost ($)",
                    x=-0.15,  # Move to the left
                    len=0.7,
                    thickness=15,
                    titleside="right"
                ),
                hovertemplate=(
                    "<b>%{location}</b><br>" +
                    "Average Monthly Cost: $%{z:,.0f}<br>" +
                    "<extra></extra>"
                )
            ),
            row=1, col=1
        )

        # Add cost burden map with right-side legend
        fig.add_trace(
            go.Scattergeo(
                locations=state_metrics['State_Abbreviation'],
                locationmode='USA-states',
                text=state_metrics['State_Abbreviation'],
                mode='markers+text',
                marker=dict(
                    size=state_metrics['bubble_size'],
                    color=state_metrics['CostIncomeRatio'],
                    colorscale=self.color_schemes['diverging'],
                    showscale=True,
                    colorbar=dict(
                        title="Cost/Income Ratio (%)",
                        x=1.15,  # Move to the right
                        len=0.7,
                        thickness=15
                    )
                ),
                hovertemplate=(
                    "<b>%{text}</b><br>" +
                    "Annual Cost: $%{customdata[0]:,.0f}<br>" +
                    "Median Income: $%{customdata[1]:,.0f}<br>" +
                    "Cost Burden: %{marker.color:.1f}%<br>" +
                    "<extra></extra>"
                ),
                customdata=state_metrics[['Annual_Cost', 'MHI_2018']].values
            ),
            row=1, col=2
        )

        # Add income distribution donut chart
        income_dist = self.data.groupby('Income_Bracket').agg({
            'Annual_Cost_Infant': 'mean',
            'MHI_2018': 'mean'
        }).reset_index()

        fig.add_trace(
            go.Pie(
                labels=income_dist['Income_Bracket'],
                values=income_dist['Annual_Cost_Infant'],
                hole=0.6,
                marker=dict(colors=px.colors.qualitative.Set3),
                textinfo='label+percent',
                textposition='outside',
                showlegend=True,
                legendgroup="right",
                legendgrouptitle_text="Income Levels",
                hovertemplate=(
                    "<b>%{label}</b><br>" +
                    "Average Annual Cost: $%{value:,.0f}<br>" +
                    "<extra></extra>"
                )
            ),
            row=2, col=2
        )

        #  Network Analysis with 3D-like bubbles
        variables = {
            'MCInfant': 'Infant Care\nCost',
            'MCToddler': 'Toddler Care\nCost',
            'MCPreschool': 'Preschool\nCost',
            'MHI_2018': 'Median\nIncome',
            'TotalPop': 'Total\nPopulation',
            'H_Under6_BothWork': 'Working Parents\nwith Young Children'
        }

        corr_matrix = self.data[list(variables.keys())].corr().abs()

        # Create network layout
        pos = {
            'MCInfant': [-1.5, 1.2],
            'MCToddler': [0, 1.5],
            'MCPreschool': [1.5, 1.2],
            'MHI_2018': [-1.5, -1.2],
            'TotalPop': [0, -1.5],
            'H_Under6_BothWork': [1.5, -1.2]
        }

        node_colors = px.colors.qualitative.Bold
        node_gradients = []
        for i, color in enumerate(node_colors[:len(variables)]):
            node_gradients.append(f’radialGradient(circle at 30% 30%, {color} 0%, rgb(45,
45,45) 90%)’)

        node_x = []
        node_y = []
        node_text = []
        node_size = []
        node_color = []

        for i, var in enumerate(variables.keys()):
            x, y = pos[var]
            node_x.append(x)
            node_y.append(y)
            node_text.append(variables[var])
            node_size.append(40 + corr_matrix[var].mean() * 30)
            node_color.append(node_colors[i])

        # Add edges with gradient colors and dynamic width
        edge_x = []
        edge_y = []
        edge_colors = []
        edge_widths = []
        edge_hover = []

        for i, var1 in enumerate(variables.keys()):
            for j, var2 in enumerate(variables.keys()):
                if i < j and corr_matrix.loc[var1, var2] > 0.3:
                    x0, y0 = pos[var1]
                    x1, y1 = pos[var2]
                    edge_x.extend([x0, x1, None])
                    edge_y.extend([y0, y1, None])

                    correlation = corr_matrix.loc[var1, var2]
                    # Use a simpler color scheme based on correlation strength
                    color = px.colors.sequential.Viridis[int(correlation * 8)]
                    edge_colors.extend([color] * 3)
                    edge_widths.extend([correlation * 6] * 3)
                    edge_hover.extend([f"{variables[var1]} â\206\224 {variables[var2]}<br
>Correlation: {correlation:.2f}"] * 3)


        fig.add_trace(
            go.Scatter(
                x=edge_x,
                y=edge_y,
                mode=’lines’,
                line=dict(
                    width=4,
                    color=edge_colors[0] if edge_colors else ’rgba(150,150,150,0.5)’
                ),
                hoverinfo=’text’,
                hovertext=edge_hover,
                name=’Correlations’
            ),
            row=2, col=1
        )

        # Add colored edges for strong correlations
        for i, var1 in enumerate(variables.keys()):
            for j, var2 in enumerate(variables.keys()):
                if i < j and corr_matrix.loc[var1, var2] > 0.6:  # Only strongest correla
tions
                    x0, y0 = pos[var1]
                    x1, y1 = pos[var2]
                    correlation = corr_matrix.loc[var1, var2]

                    fig.add_trace(
                        go.Scatter(
                            x=[x0, x1],
                            y=[y0, y1],
                            mode='lines',
                            line=dict(
                                width=correlation * 6,
                                color=f'rgba({min(255, int(correlation * 255))}, 0, {max(
0, int((1-correlation) * 255))}, {correlation})'
                            ),
                            hoverinfo='text',
                            hovertext=f"{variables[var1]} â\206\224 {variables[var2]}<br>
Correlation: {correlation:.2f}",
                            showlegend=False
                        ),
                        row=2, col=1
                    )

        # Add nodes with 3D-like appearance
        fig.add_trace(
            go.Scatter(
                x=node_x,
                y=node_y,
                mode='markers+text',
                marker=dict(
                    size=node_size,
                    color=node_color,
                    line=dict(width=2, color='white'),
                    symbol='circle',
                    gradient=dict(
                        type='radial',
                        color='rgb(45,45,45)'
                    )
                ),
                text=node_text,
                textposition="middle center",
                textfont=dict(size=11, color='white', family='Arial Black'),
                hoverinfo='text',
                name='Variables'
            ),
            row=2, col=1
        )

     #bubble correlation plot
        var_x = []
        var_y = []
        corr_values = []
        corr_text = []

        for i, var1 in enumerate(variables.keys()):
            for j, var2 in enumerate(variables.keys()):
                var_x.append(variables[var1])
                var_y.append(variables[var2])
                correlation = corr_matrix.loc[var1, var2]
                corr_values.append(abs(correlation))
                corr_text.append(f"{correlation:.2f}")

        fig.add_trace(
            go.Scatter(
                x=var_x,
                y=var_y,
                mode='markers+text',
                marker=dict(
                    size=[cv * 50 for cv in corr_values],
                    color=corr_values,
                    colorscale='RdYlBu_r',
                    showscale=True,
                    colorbar=dict(
                        title="Correlation<br>Strength",
                        x=-0.15,  # Move to the left
                        len=0.7,
                        thickness=15,
                        titleside="right"
                    )
                ),
                text=corr_text,
                textfont=dict(color='black', size=10),
                textposition='middle center',
                hovertemplate=(
                    "<b>Variables:</b><br>" +
                    "%{x} â\206\224 %{y}<br>" +
                    "<b>Correlation:</b> %{text}<br>" +
                    "<extra></extra>"
                )
            ),
            row=3, col=1
        )

        # Rsunburst chart
        state_hierarchy = self.data.groupby(['State_Abbreviation', 'Income_Bracket']).agg
({
            'Annual_Cost_Infant': 'mean',
            'MHI_2018': 'mean'
        }).reset_index()

        # Calculate cost burden for each group
        state_hierarchy['Cost_Burden'] = (state_hierarchy['Annual_Cost_Infant'] / state_h
ierarchy['MHI_2018']) * 100

        # Create labels and parents for sunburst
        labels = list(state_hierarchy['State_Abbreviation'].unique()) + \
                list(state_hierarchy.apply(lambda x: f"{x['State_Abbreviation']}-{x['Inco
me_Bracket']}", axis=1))

        parents = [''] * len(state_hierarchy['State_Abbreviation'].unique()) + \
                 list(state_hierarchy['State_Abbreviation'])

        values = list(state_hierarchy.groupby('State_Abbreviation')['Cost_Burden'].mean()
) + \
                list(state_hierarchy['Cost_Burden'])

        # Create color scale based on cost burden
        colors = px.colors.sequential.Viridis

        fig.add_trace(
            go.Sunburst(
                labels=labels,
                parents=parents,
                values=values,
                branchvalues='total',
                marker=dict(
                    colors=values,
                    colorscale=colors,
                    showscale=True,
                    colorbar=dict(
                        title="Cost Burden (%)",
                        x=1.15,
                        len=0.7,
                        thickness=15
                    )
                ),
                hovertemplate=(
                    "<b>%{label}</b><br>" +
                    "Cost Burden: %{value:.1f}%<br>" +
                    "<extra></extra>"
                )
            ),
            row=3, col=2
        )

        # legend positioning and add 6th visualization
        fig.update_layout(
            width=2400,
            height=2400,
            template='plotly_white',
            showlegend=True,
            legend=dict(
                x=1.2,  # Move legend to the right side
                y=0.5,
                xanchor='left',
                yanchor='middle',
                bgcolor='rgba(255,255,255,0.8)',
                bordercolor='rgba(0,0,0,0.2)',
                borderwidth=1
            ),
            margin=dict(t=150, b=100, r=300, l=300)  # Increased right margin for legend
        )

        #  colorbar positions for each subplot
        for i, trace in enumerate(fig.data):
            if hasattr(trace, 'colorbar'):
                if i == 0:  # First choropleth
                    trace.colorbar.x = -0.2
                    trace.colorbar.title.side = 'right'
                elif i == 1:  # Second map
                    trace.colorbar.x = 1.2
                    trace.colorbar.title.side = 'right'
                elif i == 4:  # Correlation plot
                    trace.colorbar.x = -0.2
                    trace.colorbar.title.side = 'right'
                elif i == 5:  # Sunburst
                    trace.colorbar.x = 1.2
                    trace.colorbar.title.side = 'right'
                trace.colorbar.len = 0.7
                trace.colorbar.thickness = 15

        # Cost Trend Comparison
        yearly_trends = self.data.groupby(['Year', 'Income_Bracket']).agg({
            'Annual_Cost_Infant': 'mean',
            'MHI_2018': 'mean'
        }).reset_index()

        colors = px.colors.qualitative.Set3
        for i, income_level in enumerate(['Very Low Income', 'Low Income', 'Middle Income
', 'Upper Middle', 'High Income']):
            mask = yearly_trends['Income_Bracket'] == income_level
            fig.add_trace(
                go.Scatter(
                    x=yearly_trends[mask]['Year'],
                    y=yearly_trends[mask]['Annual_Cost_Infant'],
                    name=income_level,
                    mode='lines+markers',
                    line=dict(width=3, color=colors[i]),
                    marker=dict(size=8, color=colors[i]),
                    legendgroup="income_trends",
                    legendgrouptitle_text="Income Levels",
                    hovertemplate=(
                        "<b>%{x|%Y}</b><br>" +
                        "Income Level: " + income_level + "<br>" +
                        "Annual Cost: $%{y:,.0f}<br>" +
                        "<extra></extra>"
                    )
                ),
                row=2, col=1
            )

        # Save the dashboard
        fig.write_html("../output/dashboard.html")
        return fig

    def generate_static_visualizations(self):
        """Generate enhanced static visualizations for the report"""
        plt.style.use('seaborn-v0_8-darkgrid')
        # Set consistent style parameters
        plt.rcParams.update({
            'figure.figsize': (15, 10),
            'font.size': 12,
            'axes.titlesize': 16,
            'axes.labelsize': 14,
            'xtick.labelsize': 12,
            'ytick.labelsize': 12,
            'axes.grid': True,
            'grid.alpha': 0.3
        })

        # 1.  Cost Distribution - Violin Plot with Swarm Overlay
        plt.figure(figsize=(15, 10))

        # Create violin plot
        sns.violinplot(data=self.data, x='Income_Bracket', y='Annual_Cost_Infant',
                      palette='viridis', inner='box')

        # Add swarm plot overlay
        sns.swarmplot(data=self.data, x='Income_Bracket', y='Annual_Cost_Infant',
                     color='white', alpha=0.5, size=4)

        plt.title('Distribution of Annual Childcare Costs by Income Level\nViolin Plot wi
th Data Points',
                 pad=20, fontsize=16, fontweight='bold')
        plt.xlabel('Income Bracket', fontsize=14)
        plt.ylabel('Annual Cost ($)', fontsize=14)
        plt.xticks(rotation=45, ha='right')

        # Add median cost annotations
        medians = self.data.groupby('Income_Bracket')['Annual_Cost_Infant'].median()
        for i, median in enumerate(medians):
            plt.text(i, median, f'${median:,.0f}',
                    horizontalalignment='center',
                    verticalalignment='bottom',
                    fontweight='bold',
                    color='red')

        plt.tight_layout()
        plt.savefig('../output/cost_distribution.png', dpi=300, bbox_inches='tight')
        plt.close()

        # 2. Enhanced Time Series - Multi-faceted Analysis
        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 15), height_ratios=[2, 1])

        # Upper plot: Stacked area chart
        yearly_data = self.data.groupby('Year').agg({
            'MCInfant': 'mean',
            'MCToddler': 'mean',
            'MCPreschool': 'mean'
        }).reset_index()

        ax1.fill_between(yearly_data['Year'], 0, yearly_data['MCInfant'],
                        alpha=0.7, label='Infant Care', color='#FF9999')
        ax1.fill_between(yearly_data['Year'], yearly_data['MCInfant'],
                        yearly_data['MCInfant'] + yearly_data['MCToddler'],
                        alpha=0.7, label='Toddler Care', color='#66B2FF')
        ax1.fill_between(yearly_data['Year'], yearly_data['MCInfant'] + yearly_data['MCTo
ddler'],
                        yearly_data['MCInfant'] + yearly_data['MCToddler'] + yearly_data[
'MCPreschool'],
                        alpha=0.7, label='Preschool', color='#99FF99')

        # Add trend lines
        for column in ['MCInfant', 'MCToddler', 'MCPreschool']:
            z = np.polyfit(range(len(yearly_data)), yearly_data[column], 1)
            p = np.poly1d(z)
            ax1.plot(yearly_data['Year'], p(range(len(yearly_data))),
                    '--', color='black', alpha=0.5)
        ax1.set_title('Temporal Analysis of Childcare Costs\nStacked Area Chart with Tren
d Lines',
                     pad=20, fontsize=16, fontweight='bold')
        ax1.set_ylabel('Monthly Cost ($)', fontsize=14)
        ax1.legend(loc='upper left', bbox_to_anchor=(1.05, 1))

        # Lower plot: Year-over-Year Change
        yoy_changes = yearly_data.set_index('Year').pct_change() * 100

        for column, color in zip(['MCInfant', 'MCToddler', 'MCPreschool'],
                                ['#FF9999', '#66B2FF', '#99FF99']):
            ax2.bar(yoy_changes.index, yoy_changes[column],
                   alpha=0.7, color=color, width=200,
                   label=f'{column.replace("MC", "")} YoY Change')

            # Add value labels
            for idx, value in zip(yoy_changes.index, yoy_changes[column]):
                if not pd.isna(value):
                    ax2.text(idx, value, f'{value:+.1f}%',
                            ha='center', va='bottom' if value > 0 else 'top',
                            fontsize=10)

        ax2.set_title('Year-over-Year Change in Costs', fontsize=14)
        ax2.set_ylabel('Percentage Change (%)', fontsize=12)
        ax2.axhline(y=0, color='black', linestyle='-', alpha=0.3)
        ax2.legend(loc='upper left', bbox_to_anchor=(1.05, 1))

        plt.tight_layout()
        plt.savefig('../output/time_series.png', dpi=300, bbox_inches='tight')
        plt.close()

        # 3.  Correlation Analysis - Clustermap with Annotations
        plt.figure(figsize=(15, 12))
        variables = {
            'MCInfant': 'Infant Care',
            'MCToddler': 'Toddler Care',
            'MCPreschool': 'Preschool',
            'MHI_2018': 'Median Income',
            'TotalPop': 'Population',
            'H_Under6_BothWork': 'Working Parents'
        }

        corr_matrix = self.data[list(variables.keys())].corr()

        # Create clustermap
        g = sns.clustermap(
            corr_matrix,
            annot=True,
            cmap='RdYlBu_r',
            center=0,
            vmin=-1,
            vmax=1,
            fmt='.2f',
            square=True,
            xticklabels=[variables[col] for col in corr_matrix.columns],
            yticklabels=[variables[col] for col in corr_matrix.columns],
            figsize=(15, 12),
            dendrogram_ratio=0.1,
            cbar_pos=(0.02, 0.8, 0.03, 0.2)
        )

        # Rotate labels
        plt.setp(g.ax_heatmap.get_xticklabels(), rotation=45, ha='right')
        plt.setp(g.ax_heatmap.get_yticklabels(), rotation=0)

        # Add title
        g.fig.suptitle('Hierarchical Clustering of Correlation Matrix',
                      fontsize=16, fontweight='bold', y=1.02)
        plt.savefig('../output/correlation.png', dpi=300, bbox_inches='tight')
        plt.close()

        # 4. State Analysis - Radial Plot with Multiple Metrics
        # Prepare data
        state_metrics = self.data.groupby('State_Abbreviation').agg({
            'Annual_Cost_Infant': 'mean',
            'MHI_2018': 'mean',
            'TotalPop': 'mean',
            'H_Under6_BothWork': 'mean'
        }).reset_index()

        # Calculate additional metrics
        state_metrics['Cost_Burden'] = (state_metrics['Annual_Cost_Infant'] / state_metri
cs['MHI_2018']) * 100
        state_metrics['Working_Parent_Ratio'] = (state_metrics['H_Under6_BothWork'] / sta
te_metrics['TotalPop']) * 100

        # Sort by cost burden
        state_metrics = state_metrics.sort_values('Cost_Burden', ascending=True)

        # Create figure with secondary y-axis
        fig = plt.figure(figsize=(20, 12))
        ax = plt.subplot(111, projection='polar')

        # Calculate angles for each state
        angles = np.linspace(0, 2*np.pi, len(state_metrics), endpoint=False)

        # Plot cost burden
        burden_values = state_metrics['Cost_Burden'].values
        ax.plot(angles, burden_values, 'o-', linewidth=2, label='Cost Burden (%)', color=
'red')
        ax.fill(angles, burden_values, alpha=0.25, color='red')

        # Plot working parent ratio
        ratio_values = state_metrics['Working_Parent_Ratio'].values
        ax.plot(angles, ratio_values, 'o-', linewidth=2, label='Working Parent Ratio (%)'
, color='blue')
        ax.fill(angles, ratio_values, alpha=0.25, color='blue')

        # Set the labels
        ax.set_xticks(angles)
        ax.set_xticklabels(state_metrics['State_Abbreviation'], fontsize=8)

        # Add legend and title
        plt.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))
        plt.title('State-Level Analysis: Cost Burden vs Working Parent Ratio\nRadial Visu
alization',
                 pad=20, fontsize=16, fontweight='bold')

        plt.tight_layout()
        plt.savefig('../output/state_costs.png', dpi=300, bbox_inches='tight')
        plt.close()

if __name__ == "__main__":
    analysis = ChildcareCostAnalysis()
    analysis.create_enhanced_dashboard()
    analysis.generate_static_visualizations()
```