import streamlit as st

import plotly.express as px

import plotly.graph\_objects as go

from plotly.subplots import make\_subplots

import numpy as np

import pandas as pd

import re

import datetime

import os

import matplotlib.pyplot as plt

# Initialize session state

if 'selected\_suggestion' not in st.session\_state:

    st.session\_state.selected\_suggestion = ""

if 'plot\_type' not in st.session\_state:

    st.session\_state.plot\_type = "Line Plot"

if 'x\_column' not in st.session\_state:

    st.session\_state.x\_column = None

if 'y\_column' not in st.session\_state:

    st.session\_state.y\_column = None

if 'hist\_column' not in st.session\_state:

    st.session\_state.hist\_column = None

if 'label\_column' not in st.session\_state:

    st.session\_state.label\_column = None

if 'value\_column' not in st.session\_state:

    st.session\_state.value\_column = None

if 'saved\_plots' not in st.session\_state:

    st.session\_state.saved\_plots = []

if 'multiple\_plots' not in st.session\_state:

    st.session\_state.multiple\_plots = []

if 'use\_plotly' not in st.session\_state:

    st.session\_state.use\_plotly = True

# --- Streamlit App Title (Centered with Icon) ---

st.markdown("<h1 style='text-align: center;'>📊 Data Visualization Generator</h1>", unsafe\_allow\_html=True)

# --- App Description without Image ---

st.markdown("""

<div style='text-align: center;'>

    <p style='font-size: 18px;'>

        Welcome to the <strong>Data Visualization Generator</strong> 🎯 <br>

        This tool lets you turn your data into beautiful, interactive charts in just a few clicks. Whether you're a student, analyst, or data enthusiast, this app helps you explore, filter, and present your datasets visually.

    </p>

    <p style='font-size: 16px;'>

        📂 Upload your CSV or Excel files <br>

        📈 Choose from line plots, bar charts, pie charts, histograms, and more <br>

        🎨 Customize themes, filters, labels, and trends <br>

        💾 Export your plots as PNG, HTML, or CSV <br>

        ⚡ Powered by Plotly, Matplotlib, and Streamlit for a smooth experience

    </p>

    <br>

    <p style='font-size: 14px; color: gray;'>Interactive dashboards made simple with automation</p>

</div>

""", unsafe\_allow\_html=True)

# --- File Uploader with Icon ---

uploaded\_file = st.file\_uploader("📂 Choose a CSV file", type=["csv", "xlsx"])

# --- Cached function to load data ---

@st.cache\_data

def load\_data(file):

    """Caches data loading to improve performance on reruns."""

    if file is not None:

        try:

            if file.name.endswith('.csv'):

                data = pd.read\_csv(file)

            elif file.name.endswith('.xlsx'):

                data = pd.read\_excel(file)

            return data

        except Exception as e:

            st.error(f"Error processing file: {e}")

            return None

    return None

# Load data using the cached function

data = load\_data(uploaded\_file)

# Main app logic when data is loaded

if data is not None:

    st.write("Data Preview:")

    st.write(data.head())

    # --- Customization Options ---

    st.sidebar.header("⚙️ Plot Customization")

    # Input for plot title

    plot\_title = st.sidebar.text\_input("Plot Title", "My Plot")

    column\_options = data.columns.tolist()

    # --- Plot Library Selection ---

    plot\_library = "Plotly" if st.session\_state.use\_plotly else "Matplotlib"

    # --- Advanced Customization Options ---

    with st.sidebar.expander("🎨 Advanced Settings"):

        template = st.selectbox(

            "Select Template",

            ["plotly", "plotly\_white", "plotly\_dark", "ggplot2", "seaborn"],

            disabled=not st.session\_state.use\_plotly

        )

        if st.session\_state.use\_plotly:

            color\_sequence = st.selectbox(

                "Color Sequence",

                ["Plotly", "D3", "Viridis", "Rainbow"]

            )

        st.markdown("\*\*Common Settings\*\*")

        opacity = st.slider("Opacity", 0.1, 1.0, 0.8)

        if not st.session\_state.use\_plotly:

            figsize\_x = st.slider("Figure Width", 5, 15, 10)

            figsize\_y = st.slider("Figure Height", 3, 10, 6)

    # --- Structured Visualization Suggestions ---

    st.sidebar.subheader("💡 Suggested Visualizations")

    # Get column types

    numeric\_cols\_suggest = data.select\_dtypes(include=np.number).columns.tolist()

    categorical\_cols\_suggest = data.select\_dtypes(include=['object', 'category']).columns.tolist()

    # First dropdown - select visualization type

    visualization\_type = st.sidebar.selectbox(

        "Select Visualization Type",

        ["", "Line Plot", "Scatter Plot", "Bar Chart", "Pie Chart", "Doughnut Chart", "Histogram"],

        key='visualization\_type\_select'

    )

    # Generate suggestions based on selected visualization type

    if visualization\_type:

        suggestions = []

        if visualization\_type in ["Line Plot", "Scatter Plot"]:

            if len(numeric\_cols\_suggest) >= 2:

                # Generate all numeric column pairs

                pairs = [(numeric\_cols\_suggest[i], numeric\_cols\_suggest[j])

                        for i in range(len(numeric\_cols\_suggest))

                        for j in range(i+1, len(numeric\_cols\_suggest))]

                suggestions = [f"'{x}' vs '{y}'" for x, y in pairs]

        elif visualization\_type == "Bar Chart":

            if len(categorical\_cols\_suggest) >= 1 and len(numeric\_cols\_suggest) >= 1:

                # Generate all category vs numeric combinations

                suggestions = [f"'{cat}' vs '{num}'"

                             for cat in categorical\_cols\_suggest

                             for num in numeric\_cols\_suggest]

        elif visualization\_type in ["Pie Chart", "Doughnut Chart"]:

            if len(categorical\_cols\_suggest) >= 1 and len(numeric\_cols\_suggest) >= 1:

                # Generate all label vs value combinations

                suggestions = [f"'{cat}' values from '{num}'"

                             for cat in categorical\_cols\_suggest

                             for num in numeric\_cols\_suggest]

        elif visualization\_type == "Histogram":

            if len(numeric\_cols\_suggest) >= 1:

                suggestions = [f"Distribution of '{num}'"

                             for num in numeric\_cols\_suggest]

        # Display column combination dropdown if there are suggestions

        if suggestions:

            selected\_option = st.sidebar.selectbox(

                f"Select {visualization\_type} Option",

                [""] + suggestions,

                key='column\_combination\_select'

            )

            if selected\_option:

                # Parse the selected option to get columns

                if visualization\_type in ["Line Plot", "Scatter Plot", "Bar Chart"]:

                    cols = re.findall(r"'(.\*?)'", selected\_option)

                    if len(cols) >= 2:

                        st.session\_state.update({

                            "plot\_type": visualization\_type,

                            "x\_column": cols[0],

                            "y\_column": cols[1] if visualization\_type != "Histogram" else None,

                            "selected\_suggestion": f"{visualization\_type}: {selected\_option}"

                        })

                elif visualization\_type in ["Pie Chart", "Doughnut Chart"]:

                    cols = re.findall(r"'(.\*?)'", selected\_option)

                    if len(cols) >= 2:

                        st.session\_state.update({

                            "plot\_type": visualization\_type,

                            "label\_column": cols[0],

                            "value\_column": cols[1],

                            "selected\_suggestion": f"{visualization\_type}: {selected\_option}"

                        })

                elif visualization\_type == "Histogram":

                    col = re.findall(r"'(.\*?)'", selected\_option)[0]

                    st.session\_state.update({

                        "plot\_type": visualization\_type,

                        "hist\_column": col,

                        "selected\_suggestion": f"{visualization\_type}: {selected\_option}"

                    })

    # --- Manual Plot Configuration ---

    plot\_type = st.sidebar.selectbox(

        "Select Plot Type",

        ["Line Plot", "Scatter Plot", "Bar Chart", "Histogram", "Pie Chart", "Doughnut Chart"],

        index=["Line Plot", "Scatter Plot", "Bar Chart", "Histogram", "Pie Chart", "Doughnut Chart"].index(st.session\_state.plot\_type)

    )

    # Update plot type in session state

    if plot\_type != st.session\_state.plot\_type:

        st.session\_state.update({

            "plot\_type": plot\_type,

            "selected\_suggestion": ""

        })

    # Column selection based on plot type

    if plot\_type in ['Line Plot', 'Scatter Plot']:

        st.sidebar.subheader("Select Columns")

        if len(numeric\_cols\_suggest) >= 2:

            x\_col = st.sidebar.selectbox(

                "X-axis Column",

                numeric\_cols\_suggest,

                index=numeric\_cols\_suggest.index(st.session\_state.x\_column) if st.session\_state.x\_column in numeric\_cols\_suggest else 0

            )

            y\_options = [col for col in numeric\_cols\_suggest if col != x\_col]

            y\_col = st.sidebar.selectbox(

                "Y-axis Column",

                y\_options,

                index=y\_options.index(st.session\_state.y\_column) if st.session\_state.y\_column in y\_options else 0

            )

            st.session\_state.update({"x\_column": x\_col, "y\_column": y\_col})

    elif plot\_type == 'Bar Chart':

        st.sidebar.subheader("Select Columns")

        if len(categorical\_cols\_suggest) > 0 and len(numeric\_cols\_suggest) > 0:

            x\_col = st.sidebar.selectbox(

                "Category Column",

                categorical\_cols\_suggest,

                index=categorical\_cols\_suggest.index(st.session\_state.x\_column) if st.session\_state.x\_column in categorical\_cols\_suggest else 0

            )

            y\_col = st.sidebar.selectbox(

                "Value Column",

                numeric\_cols\_suggest,

                index=numeric\_cols\_suggest.index(st.session\_state.y\_column) if st.session\_state.y\_column in numeric\_cols\_suggest else 0

            )

            st.session\_state.update({"x\_column": x\_col, "y\_column": y\_col})

    elif plot\_type in ['Pie Chart', 'Doughnut Chart']:

        st.sidebar.subheader("Select Columns")

        if len(categorical\_cols\_suggest) > 0 and len(numeric\_cols\_suggest) > 0:

            label\_col = st.sidebar.selectbox(

                "Label Column",

                categorical\_cols\_suggest,

                index=categorical\_cols\_suggest.index(st.session\_state.label\_column) if st.session\_state.label\_column in categorical\_cols\_suggest else 0

            )

            value\_col = st.sidebar.selectbox(

                "Value Column",

                numeric\_cols\_suggest,

                index=numeric\_cols\_suggest.index(st.session\_state.value\_column) if st.session\_state.value\_column in numeric\_cols\_suggest else 0

            )

            st.session\_state.update({"label\_column": label\_col, "value\_column": value\_col})

    elif plot\_type == 'Histogram':

        st.sidebar.subheader("Select Column")

        if len(numeric\_cols\_suggest) > 0:

            hist\_col = st.sidebar.selectbox(

                "Column",

                numeric\_cols\_suggest,

                index=numeric\_cols\_suggest.index(st.session\_state.hist\_column) if st.session\_state.hist\_column in numeric\_cols\_suggest else 0

            )

            st.session\_state.update({"hist\_column": hist\_col})

    # --- Data Filtering Section ---

    with st.sidebar.expander("🔍 Data Filters"):

        filtered\_data = data.copy()

        # Determine columns involved in the current plot

        filter\_columns = []

        if plot\_type in ['Line Plot', 'Scatter Plot', 'Bar Chart']:

            filter\_columns = [st.session\_state.x\_column, st.session\_state.y\_column]

        elif plot\_type == 'Histogram':

            filter\_columns = [st.session\_state.hist\_column]

        elif plot\_type in ['Pie Chart', 'Doughnut Chart']:

            filter\_columns = [st.session\_state.label\_column, st.session\_state.value\_column]

        filter\_columns = [col for col in filter\_columns if col is not None]

        # Apply filters only to relevant columns

        for col in filter\_columns:

            if col in data.columns:

                if pd.api.types.is\_numeric\_dtype(data[col]):

                    min\_val, max\_val = float(data[col].min()), float(data[col].max())

                    if min\_val != max\_val:

                        selected\_range = st.slider(

                            f"Filter {col}", min\_val, max\_val, (min\_val, max\_val)

                        )

                        filtered\_data = filtered\_data[

                            (filtered\_data[col] >= selected\_range[0]) &

                            (filtered\_data[col] <= selected\_range[1])

                        ]

                elif pd.api.types.is\_string\_dtype(data[col]) or pd.api.types.is\_categorical\_dtype(data[col]):

                    options = data[col].dropna().unique().tolist()

                    selected = st.multiselect(

                        f"Filter {col}", options, default=options

                    )

                    filtered\_data = filtered\_data[filtered\_data[col].isin(selected)]

        # Update data with filtered version

        data = filtered\_data

    # --- Generate Plot Button ---

    if st.button("Generate Plot"):

        if plot\_type in ['Line Plot', 'Scatter Plot', 'Bar Chart', 'Histogram', 'Pie Chart', 'Doughnut Chart']:

            if st.session\_state.use\_plotly:

                fig = None

                if plot\_type in ['Line Plot', 'Scatter Plot'] and st.session\_state.x\_column and st.session\_state.y\_column:

                    if plot\_type == 'Line Plot':

                        fig = px.line(data, x=st.session\_state.x\_column, y=st.session\_state.y\_column,

                                    title=plot\_title, template=template)

                    else:  # Scatter Plot

                        fig = px.scatter(data, x=st.session\_state.x\_column, y=st.session\_state.y\_column,

                                       title=plot\_title, template=template,

                                       trendline="ols" if st.sidebar.checkbox("Show Trendline") else None)

                elif plot\_type == 'Bar Chart' and st.session\_state.x\_column and st.session\_state.y\_column:

                    fig = px.bar(data, x=st.session\_state.x\_column, y=st.session\_state.y\_column,

                                title=plot\_title, template=template)

                elif plot\_type == 'Histogram' and st.session\_state.hist\_column:

                    fig = px.histogram(data, x=st.session\_state.hist\_column,

                                     title=plot\_title, template=template,

                                     nbins=st.sidebar.slider("Number of Bins", 5, 50, 20))

                elif plot\_type in ['Pie Chart', 'Doughnut Chart'] and st.session\_state.label\_column and st.session\_state.value\_column:

                    fig = px.pie(data, names=st.session\_state.label\_column, values=st.session\_state.value\_column,

                                title=plot\_title, template=template,

                                hole=0.4 if plot\_type == 'Doughnut Chart' else 0)

                if fig is not None:

                    st.plotly\_chart(fig, use\_container\_width=True)

            else:  # Matplotlib version

                plt.figure(figsize=(figsize\_x, figsize\_y))

                if plot\_type in ['Line Plot', 'Scatter Plot'] and st.session\_state.x\_column and st.session\_state.y\_column:

                    if plot\_type == 'Line Plot':

                        plt.plot(data[st.session\_state.x\_column], data[st.session\_state.y\_column])

                    else:  # Scatter Plot

                        plt.scatter(data[st.session\_state.x\_column], data[st.session\_state.y\_column])

                    plt.xlabel(st.session\_state.x\_column)

                    plt.ylabel(st.session\_state.y\_column)

                elif plot\_type == 'Bar Chart' and st.session\_state.x\_column and st.session\_state.y\_column:

                    plt.bar(data[st.session\_state.x\_column], data[st.session\_state.y\_column])

                    plt.xticks(rotation=45)

                elif plot\_type == 'Histogram' and st.session\_state.hist\_column:

                    plt.hist(data[st.session\_state.hist\_column], bins=20)

                    plt.xlabel(st.session\_state.hist\_column)

                    plt.ylabel("Frequency")

                elif plot\_type in ['Pie Chart', 'Doughnut Chart'] and st.session\_state.label\_column and st.session\_state.value\_column:

                    plt.pie(data[st.session\_state.value\_column],

                           labels=data[st.session\_state.label\_column],

                           autopct='%1.1f%%')

                    plt.axis('equal')

                plt.title(plot\_title)

                plt.grid(True)

                st.pyplot(plt)

    # --- Export Section ---

    if st.checkbox("Show Export Options"):

        st.subheader("Export Visualization")

        export\_format = st.selectbox("Select Format", ["PNG", "HTML", "CSV"])

        if st.button("Export"):

            timestamp = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")

            filename = f"export\_{plot\_title}\_{timestamp}.{export\_format.lower()}"

            if export\_format == "PNG":

                if st.session\_state.use\_plotly and fig:

                    fig.write\_image(filename)

                else:

                    plt.savefig(filename)

                st.success(f"Saved as {filename}")

            elif export\_format == "HTML":

                if st.session\_state.use\_plotly and fig:

                    fig.write\_html(filename)

                    st.success(f"Saved as {filename}")

                else:

                    st.warning("HTML export only available for Plotly visualizations")

            elif export\_format == "CSV":

                data.to\_csv(filename, index=False)

                st.success(f"Data saved as {filename}")

# --- Features Section ---

st.sidebar.markdown("---")

with st.sidebar.expander("ℹ️ Features"):

    st.markdown("""

    \*\*Key Features:\*\*

    - Interactive Plotly visualizations

    - Data filtering options

    - Multiple export formats

    - Smart plot suggestions

    - Support for CSV/Excel files

    """)

# --- Run Instructions ---

st.sidebar.markdown("")