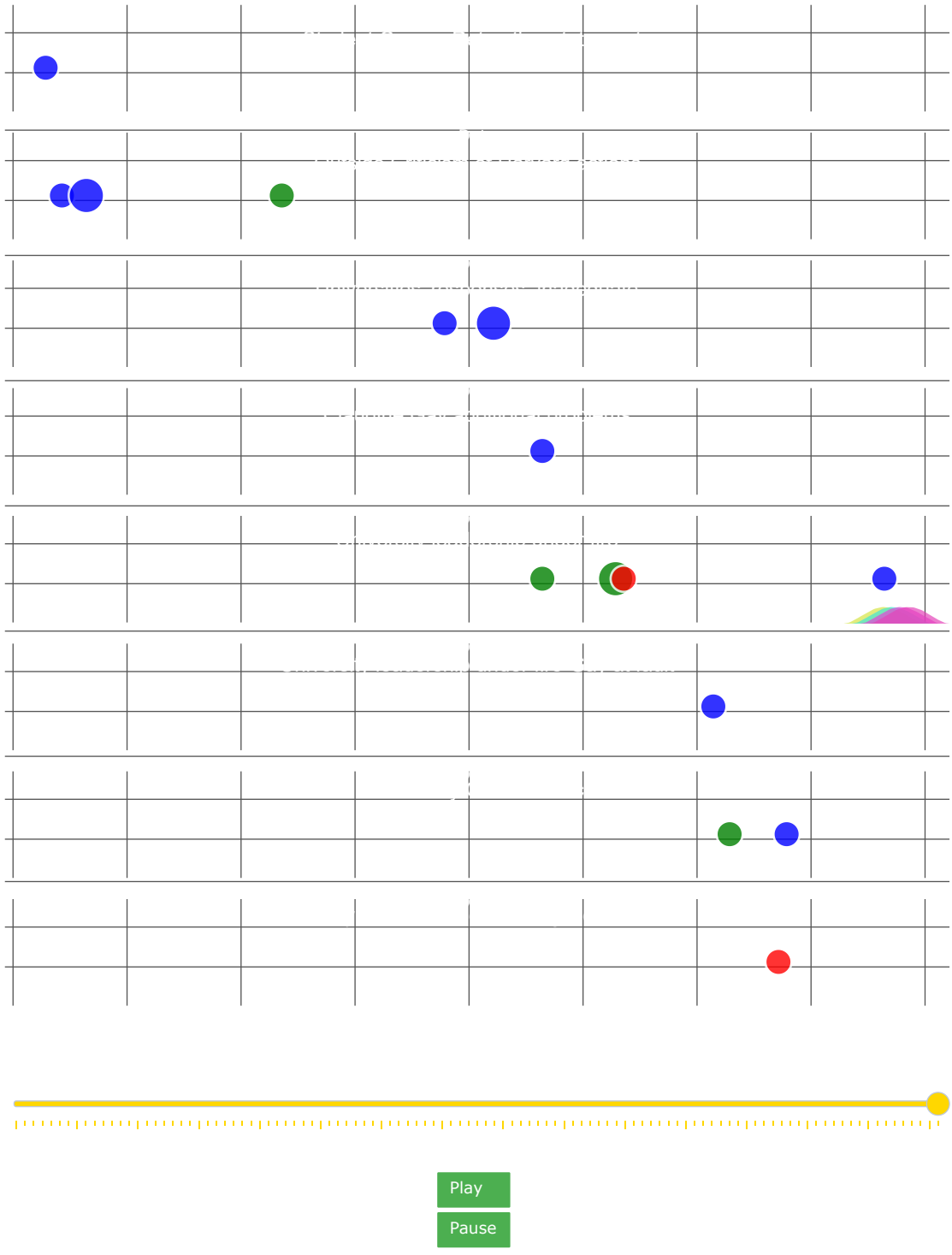
```
import warnings
import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np
import textwrap
import colorsys

# Suppress warnings
warnings.filterwarnings("ignore")

# Load the dataset from the provided Excel file
file_path = '/content/EpochTimes.xlsx'
df = pd.read_excel(file_path)

# Convert the 'Date' column to datetime format and drop rows with missing dates
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y', errors='coerce')
df = df.dropna(subset=['Date'])

# Sort the DataFrame by date to ensure linear progression
df = df.sort_values(by='Date').copy()

# Function to create a mountain shape with enhanced scaling for low scores and added offset
def create_mountain(center_date, height, width_days, offset_index=0):
    half_width = pd.Timedelta(days=width_days/2)
    offset = pd.Timedelta(days=offset_index * 1)
    start_date = center_date - half_width + offset
    end_date = center_date + half_width + offset
    dates = pd.date_range(start=start_date, end=end_date, periods=50)
    x_range = np.linspace(-1, 1, len(dates))

    enhanced_height = np.log1p(height) * 10
    y = enhanced_height * (1 - x_range**2)**2
    return dates, y

# Get all unique sub-narratives
sub_narratives = df['Current Sub-Narrative'].unique()

# Define fallacy columns
fallacy_columns = df.columns.difference(['Date', 'Current Sub-Narrative', 'Title', 'Media Bias', 'TOTALS', 'Sentiment'])

# Function to generate a unique color for each fallacy with transparency
def generate_color_map(fallacy_columns):
    n = len(fallacy_columns)
    hsv_tuples = [(x * 1.0 / n, 0.7, 0.9) for x in range(n)]
    rgb_tuples = map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples)
    color_map = {}
    for fallacy, rgb in zip(fallacy_columns, rgb_tuples):
        rgb_int = tuple(int(x * 255) for x in rgb)
        color_map[fallacy] = f'rgba{rgb_int + (0.7,)}'
    return color_map

# Generate color map for fallacies
fallacy_colors = generate_color_map(fallacy_columns)

# Define colors for the sentiment
sentiment_colors = {
    'Misinformation': 'red',
    'Potential Misinformation': 'blue',
    'No Misinformation': 'green'
}

# Define the lifespan of a mountain in days
total_days = (df['Date'].max() - df['Date'].min()).days
expansion_days = int(total_days/10)
contraction_days = expansion_days/2

# Create subplots
fig = make_subplots(rows=len(sub_narratives), cols=1,
                    shared_xaxes=True,
                    vertical_spacing=0.02)

# Prepare all scatter traces for mountains and bubbles
all_scatter_traces = []
all_bubble_traces = []

# Scale factor for mountain height
scale_factor = 2.5

# Track the earliest and latest x-axis dates (after applying offsets)
min_x_axis_date = pd.Timestamp.max
```

```
    max_x_axis_date = pd.Timestamp.min

    for i, sub_narrative in enumerate(sub_narratives, start=1):
        sub_df = df[df['Current Sub-Narrative'] == sub_narrative]
        sub_df = sub_df.sort_values('Date')
        date_counts = sub_df['Date'].value_counts().sort_index()
        date_offsets = {date: 0 for date in date_counts.index}

        # Initialize counters for each sentiment color
        color_counters = {'red': 0, 'blue': 0, 'green': 0}

        for j, (index, row) in enumerate(sub_df.iterrows()):
            fallacy_offset_counter = 0
            for fallacy in fallacy_columns:
                if row[fallacy] > 0:
                    # Calculate width to next article or use default if it's the last article
                    if j < len(sub_df) - 1:
                        width_days = (sub_df.iloc[j+1]['Date'] - row['Date']).days
                    else:
                        width_days = 10  # Default width for the last article

                    all_scatter_traces.append((row['Date'], row[fallacy], fallacy, fallacy_offset_counter, i, width_days))

                    offset_date = row['Date'] + pd.Timedelta(days=fallacy_offset_counter)
                    min_x_axis_date = min(min_x_axis_date, offset_date)
                    max_x_axis_date = max(max_x_axis_date, offset_date)

                    fallacy_offset_counter += 1

            # Create bubble trace with offset and increasing size
            offset = date_offsets[row['Date']]
            color = sentiment_colors[row['Sentiment']]
            color_counters[color] += 1
            size = 15 + color_counters[color] * 8  # Base size of 15, increasing by 8 for each new bubble of the same color

            bubble = go.Scatter(
                x=[row['Date'] + pd.Timedelta(days=offset)],
                y=[df['TOTALS'].max() * scale_factor / 2],  # Center of the subplot
                mode='markers',
                marker=dict(
                    size=size,
                    color=color,
                    symbol='circle',
                    line=dict(color='white', width=2),
                    opacity=0.8,
                    sizemode='diameter',
                ),
                name=sub_narrative,
                hoverinfo='text',
                text=f'Sub-Narrative: {sub_narrative}<br>Date: {row["Date"].date()}<br>Article: {row["Title"]}<br>Sentiment: {row["Sentiment"
                showlegend=False
            )
            all_bubble_traces.append((bubble, i, row['Date'], row['TOTALS']))

            # Update offset for the next bubble on the same date
            date_offsets[row['Date']] += 1

    # Create frames
    all_dates = pd.date_range(min_x_axis_date, max_x_axis_date, freq='D')
    frames = []

    for date in all_dates:
        frame_data = []
        for article_date, score, fallacy, offset_index, subplot, width_days in all_scatter_traces:
            days_passed = (date - article_date).days
            if days_passed < 0:
                remaining_height = 0
            elif 0 <= days_passed <= expansion_days:
                remaining_height = score * scale_factor * (np.sin((days_passed / expansion_days) * (np.pi / 2)))**2
            elif expansion_days < days_passed <= (expansion_days + contraction_days):
                contraction_progress = (days_passed - expansion_days) / contraction_days
                remaining_height = score * scale_factor * (1 - contraction_progress)**2
            else:
                remaining_height = 0

            x_mountain, y_mountain = create_mountain(article_date, remaining_height, width_days, offset_index=offset_index)
            new_scatter = go.Scatter(
                x=x_mountain,
                y=y_mountain,
                fill='tozeroy',
                fillcolor=fallacy_colors[fallacy],
                line=dict(width=0),
                name=fallacy,
```

```
                hoverinfo='text',
                text=f'Date: {article_date.date()}<br>Fallacy: {fallacy}<br>Score: {score}',
                showlegend=False
            )
            frame_data.append(new_scatter)

        for bubble, subplot, article_date, total_fallacies in all_bubble_traces:
            new_bubble = go.Scatter(
                x=bubble.x,
                y=bubble.y,
                mode=bubble.mode,
                marker=bubble.marker,
                name=bubble.name,
                hoverinfo=bubble.hoverinfo,
                text=bubble.text,
                showlegend=bubble.showlegend,
                visible=(article_date <= date)  # Visible on and after its specific date
            )
            frame_data.append(new_bubble)

        frames.append(go.Frame(data=frame_data, name=f'frame_{date.date()}'))

    # Add initial state for each subplot
    for article_date, score, fallacy, offset_index, subplot, width_days in all_scatter_traces:
        x_mountain, y_mountain = create_mountain(article_date, 0, width_days, offset_index=offset_index)
        initial_scatter = go.Scatter(
            x=x_mountain,
            y=y_mountain,
            fill='tozeroy',
            fillcolor=fallacy_colors[fallacy],
            line=dict(width=0),
            name=fallacy,
            hoverinfo='text',
            text=f'Date: {article_date.date()}<br>Fallacy: {fallacy}<br>Score: {score}',
            showlegend=False
        )
        fig.add_trace(initial_scatter, row=subplot, col=1)

    # Add bubbles to initial state (invisible)
    for bubble, subplot, _, _ in all_bubble_traces:
        invisible_bubble = go.Scatter(
            x=bubble.x,
            y=bubble.y,
            mode=bubble.mode,
            marker=bubble.marker,
            name=bubble.name,
            hoverinfo=bubble.hoverinfo,
            text=bubble.text,
            showlegend=bubble.showlegend,
            visible=False
        )
        fig.add_trace(invisible_bubble, row=subplot, col=1)

    # Assign frames to the figure
    fig.frames = frames

    # Create slider steps
    steps = []
    for date in all_dates:
        step = dict(
            method="animate",
            args=[[f'frame_{pd.to_datetime(date).date()}'],
                  {"frame": {"duration": 200, "redraw": True},
                   "mode": "immediate",
                   "transition": {"duration": 200}}],
            label=pd.to_datetime(date).strftime("%b %d")
        )
        steps.append(step)

    # Customize the layout
    fig.update_layout(
        title='EPOCH TIMES Narrative',
        showlegend=False,
        sliders=[dict(
            active=0,
            steps=steps,
            currentvalue={"prefix": "Date: ", "font": {"color": "white"}},
            pad={"t": 50},
            bgcolor='gold',
            tickcolor='gold',
            font=dict(color='white')
        )],
        height=150 * len(sub_narratives)
```

```
        height 150 * len(sub_narratives),
        updatemenus=[dict(
            type='buttons',
            showactive=False,
            x=0.5,
            y=-0.15,
            xanchor='center',
            yanchor='top',
            pad={"r": 10, "t": 10},
            buttons=[
                dict(label='Play',
                     method='animate',
                     args=[None, {'frame': {'duration': 300, 'redraw': True}, 'fromcurrent': True, 'mode': 'immediate'}],
                     visible=True),
                dict(label='Pause',
                     method='animate',
                     args=[[None], {'frame': {'duration': 0, 'redraw': False}, 'mode': 'immediate', 'transition': {'duration': 0}}],
                     visible=True)
            ],
            font=dict(size=14),
            bordercolor='#FFFFFF',
            borderwidth=2,
            bgcolor='#4CAF50'
        )],
        plot_bgcolor='#2F2F2F',  # Lighter dark grey background
        paper_bgcolor='#2F2F2F',  # Lighter dark grey background
        font=dict(color='white')
    )

    # Update y-axis for each subplot
    max_height = df['TOTALS'].max() * scale_factor * 1.2
    for i in range(1, len(sub_narratives) + 1):
        fig.update_yaxes(range=[0, max_height], showticklabels=False, title_text="", row=i, col=1, showgrid=True, gridcolor='#555555')

    # Update x-axis range and title for all subplots
    fig.update_xaxes(range=[min_x_axis_date - pd.Timedelta(days=5), max_x_axis_date + pd.Timedelta(days=5)],
                     title_text="Date", showgrid=True, gridcolor='#555555')

    # Add subplot titles and separation lines
    for i, sub_narrative in enumerate(sub_narratives, start=1):
        fig.add_annotation(
            text=textwrap.fill(sub_narrative, width=20),
            xref="paper", yref="paper",
            x=0.5, y=1 - (i-1)/len(sub_narratives) - 0.02,
            showarrow=False,
            font=dict(size=18, color='white', family='Arial'),
            xanchor='center',
            yanchor='top'
        )

        if i < len(sub_narratives):
            fig.add_shape(
                type="line",
                x0=0, x1=1, y0=1-i/len(sub_narratives), y1=1-i/len(sub_narratives),
                xref="paper", yref="paper",
                line=dict(color="#555555", width=1)
            )

    # Show the figure
    fig.show()
```




```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go

# Load the dataset
file_path = '/content/EpochTimes.xlsx'
df = pd.read_excel(file_path)

# Convert the 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y', errors='coerce')

# Define fallacy columns
fallacy_columns = df.columns.difference(['Date', 'Current Sub-Narrative', 'Title', 'Media Bias', 'TOTALS', 'Sentiment'])

# Calculate total fallacies for each article
df['Total Fallacies'] = df[fallacy_columns].sum(axis=1)
```

```
# Save original fallacies for hover display
df['Original Fallacies'] = df['Total Fallacies']

# Set a minimum size for articles with 0 fallacies
df['Total Fallacies'] = df['Total Fallacies'].apply(lambda x: 0.5 if x == 0 else x)

# Count the number of articles per sub-narrative
article_count = df.groupby('Current Sub-Narrative')['Title'].count().reset_index()
article_count.columns = ['Current Sub-Narrative', 'Article Count']

# Merge the article count with the original dataframe
df = df.merge(article_count, on='Current Sub-Narrative', how='left')

# Offset dates slightly for articles with the same start date within a sub-narrative
df['Date'] = df['Date'] + pd.to_timedelta(df.groupby(['Current Sub-Narrative', 'Date']).cumcount(), unit='D')

# Sort the dataframe by date
df = df.sort_values(by='Date')

# Define a custom color function
def custom_color(sentiment, depth):
    if depth == 0:  # Main sub-narrative level
        return 'purple'
    else:  # Article level
        color_map = {
            'Misinformation': 'red',
            'Potential Misinformation': 'darkblue',
            'No Misinformation': 'green'
        }
        return color_map.get(sentiment, 'gray')

# Assign colors to each unique item (sub-narrative or article)
color_dict = {}
for _, row in df.iterrows():
    sub_narrative = row['Current Sub-Narrative']
    title = row['Title']

    if sub_narrative not in color_dict:
        color_dict[sub_narrative] = custom_color(None, 0)

    if title not in color_dict:
        color_dict[title] = custom_color(row['Sentiment'], 1)

# Create frames for each date
frames = []
unique_dates = df['Date'].unique()
for date in unique_dates:
    frame_df = df[df['Date'] <= date]
    treemap = px.treemap(
        frame_df,
        path=['Current Sub-Narrative', 'Title'],
        values='Total Fallacies',
        color='Sentiment',
        hover_data={'Date': True, 'Total Fallacies': False, 'Sentiment': True, 'Article Count': True, 'Original Fallacies': True},
        title='EPOCH TIMES Narrative Treemap'
    )

    # Apply consistent colors
    treemap.data[0].marker.colors = [color_dict[id.split('/')[-1]] for id in treemap.data[0]['ids']]

    # Update hover template
    treemap.update_traces(
        hovertemplate=[
            '<b>%{label}</b><br>Total Articles: %{customdata[3]}' if len(id.split('/')) == 1 else '<b>%{label}</b><br>Total Fallacies: %
        ]
    )

    frame = go.Frame(data=treemap.data, name=str(date.date()))
    frames.append(frame)

# Create the initial treemap
initial_df = df[df['Date'] == unique_dates[0]]
fig = px.treemap(
    initial_df,
    path=['Current Sub-Narrative', 'Title'],
    values='Total Fallacies',
    color='Sentiment',
    hover_data={'Date': True, 'Total Fallacies': False, 'Sentiment': True, 'Article Count': True, 'Original Fallacies': True},
    title='EPOCH TIMES Narrative Treemap'
)

# Apply consistent colors to the initial treemap
```

```
fig.data[0].marker.colors = [color_dict[id.split('/')[-1]] for id in fig.data[0]['ids']]

# Update the layout for better visibility
fig.update_layout(
    font=dict(color='white'),
    paper_bgcolor='#2F2F2F',
    plot_bgcolor='#2F2F2F',
    margin=dict(t=50, l=25, r=25, b=25),
    sliders=[{
        'steps': [{'method': 'animate', 'label': str(date.date()), 'args': [[str(date.date())], {'frame': {'duration': 500, 'redraw': Tr
        'transition': {'duration': 300},
        'x': 0.1, 'xanchor': 'left', 'y': 0, 'yanchor': 'top'
    }]
)

# Update traces for better text visibility and hover template
fig.update_traces(
    textfont=dict(color='white', size=10),
    textposition="middle center",
    texttemplate='<b>%{label}</b>',  # Show only the article name without the fallacy count
    hovertemplate=[
        '<b>%{label}</b><br>Total Articles: %{customdata[3]}' if len(id.split('/')) == 1 else '<b>%{label}</b><br>Total Fallacies: %{cus
    ],
    branchvalues='total'
)

# Add animation frames
fig.frames = frames

# Show the figure
fig.show()
```