TrustLab Data Scientist Technical Assessment - Data Visualization

In [ ]:
```
import os
from wordcloud import WordCloud, STOPWORDS
import plotly.express as px
import pandas as pd
import ipywidgets as widgets
from IPython.display import display
from IPython.display import clear_output
```

Load data, merge them and convert them to pandas dataframe

In [ ]:
```
def json_to_dataframe(folder):
    # get all json files from given folder
    file_list = os.listdir(folder)
    json_files = [file for file in file_list if file.endswith('.json')]

    dataframes = []
    # read each json file and append to the list of dataframes
    for js in json_files:
        df = pd.read_json(os.path.join(folder, js))
        dataframes.append(df)

    # concatenate all dataframes into a single dataframe
    df_final = pd.concat(dataframes, ignore_index=True)

    return df_final

data = json_to_dataframe("data/archive")
```

Data distribution

This plot allows the visualization the distribution of a feature in the dataset. Please select a feature from the dropdown menu below and use the slider to adjust the number of bins for the plot

In [ ]:
```
def plot_distributions(df):
    # Get a list of column names for the user to choose from
    column_names = ["sentiment_pattern", "subjective_pattern", "industry"]

    # Create a select box widget for selecting the x-axis column
    x_column_widget = widgets.Select(
        options=column_names,
        description='Select feature:',
        disabled=False
    )

    # Create a slider widget for selecting the number of bins
    num_bins_widget = widgets.IntSlider(
        value=100,
```

```
        min=10,
        max=200,
        step=1,
        description='Select number of bins:',
        disabled=False,
        continuous_update=False,
        orientation='horizontal',
        readout=True,
        readout_format='d'
    )

    # Create an output widget for displaying the plot
    output_widget = widgets.Output()

    # Define a function to update the plot when the widgets are changed
    def update_plot(change):
        x_column = x_column_widget.value
        num_bins = num_bins_widget.value
        fig = px.histogram(df, x=x_column, nbins=num_bins)
        with output_widget:
            output_widget.clear_output()
            fig.show()

    # Display the widgets and plot
    display(x_column_widget)
    display(num_bins_widget)
    display(output_widget)
    x_column_widget.observe(update_plot, names='value')
    num_bins_widget.observe(update_plot, names='value')
    update_plot(None)  # initial plot


plot_distributions(data)
```

```
Select(description='Select feature:', options=('sentiment_pattern', 'subjective_pa
ttern', 'industry'), value='…
IntSlider(value=100, continuous_update=False, description='Select number of bin
s:', max=200, min=10)
Output()
```

Geographical data

This map plot shows the geographic locations of tweets in the dataset. Use the checkbox below to toggle the display of usernames on the map.

In [ ]:

```
def create_map_plot(df):
    # Select the columns we need for the map plot
    map_data = df[['screen_name', 'latitude', 'longitude']]

    # Drop rows with missing latitude or longitude values
    map_data = map_data.dropna(subset=['latitude', 'longitude'])

    # Create a checkbox to toggle the display of usernames on the map
    show_usernames = widgets.Checkbox(
        value=True,
        description='Show usernames'
    )

    # Define a function to update the map plot
    def update_map_plot(change):
```

```
        show_names = show_usernames.value
        fig = px.scatter_mapbox(map_data,
                                lat="latitude",
                                lon="longitude",
                                hover_name="screen_name" if show_names else None,
                                zoom=2,
                                height=600)
        fig.update_layout(mapbox_style="open-street-map")
        clear_output(wait=True)
        fig.show()

    # Display the checkbox and map plot
    display(show_usernames)
    show_usernames.observe(update_map_plot, names='value')
    update_map_plot(None)


create_map_plot(data)
```

Sentiment Time Series

This plot allows the visualization of the average sentiment and subjectivity scores over time for a given dataset

In [ ]:

```
def create_sentiment_time_series(df):
    # Filter the data to include only rows with sentiment and subjectivity scores
    df = df.dropna(subset=["sentiment_pattern", "subjective_pattern"])

    # Convert the 'created_at' column to datetime and drop rows with invalid value
    df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce')
    df.dropna(subset=['created_at'], inplace=True)

    # Group the data by date and calculate the mean sentiment and subjectivity sco
    grouped_data = df.groupby(
        df["created_at"].dt.date).agg(
        {"sentiment_pattern": "mean", "subjective_pattern": "mean"}
    )

    # Create a line plot of sentiment and subjectivity scores over time using Plot
    fig = px.line(grouped_data, x=grouped_data.index, y=[
                  "sentiment_pattern", "subjective_pattern"],
                  title="Sentiment and Subjectivity Scores over Time")

    # Set the x-axis label and tick format
    fig.update_xaxes(title_text="Date", tickformat="%b %d, %Y")

    # Set the y-axis label and range
    fig.update_yaxes(title_text="Score", range=[-1.0, 1.0])

    # Show the plot
    fig.show()


create_sentiment_time_series(data)
```

Sentiment vs Subjectivity

This plot allows you to visualize the average sentiment and subjectivity scores over time for a given dataset

In [ ]:

```
def scatter_sentiment_subjectivity(df):
    # Create a scatter plot of sentiment scores and subjectivity scores
    fig = px.scatter(df, x='sentiment_pattern', y='subjective_pattern',
                     title='Sentiment vs. Subjectivity',
                     labels={'sentiment_pattern': 'Sentiment',
                             'subjective_pattern': 'Subjectivity'})

    # Show the scatter plot in Jupyter notebook using Plotly
    fig.show()


scatter_sentiment_subjectivity(data)
```

Sentiment Heatmap

In [ ]:

```
def sentiment_heatmap(df):
    # Drop rows with invalid datetime values in the 'created_at' column
    df['created_at'] = pd.to_datetime(df['created_at'], errors='coerce')
    df.dropna(subset=['created_at'], inplace=True)

    # Add columns for day of the week and time of day
    df['weekday'] = df['created_at'].dt.weekday
    df['hour'] = df['created_at'].dt.hour

    # Create a pivot table with the mean sentiment score for each weekday and hour
    pivot_df = df.pivot_table(
        index='hour', columns='weekday', values='sentiment_pattern', aggfunc='mean

    # Define constants
    COLOR_SCALE = 'RdYlGn'
    TICK_TEXT = [f'{i:02d}:00' for i in range(24)]
    TICK_VALS = [i for i in range(24)]
    WEEKDAYS = ['Monday', 'Tuesday', 'Wednesday',
                'Thursday', 'Friday', 'Saturday', 'Sunday']

    # Create a heatmap with Plotly
    fig = px.imshow(pivot_df, x=WEEKDAYS, y=TICK_VALS,
                    color_continuous_scale=COLOR_SCALE, zmin=-1, zmax=1)
    fig.update_layout(
        title='Sentiment Scores by Day of the Week and Time of Day',
        xaxis_title='Day of the Week',
        yaxis_title='Time of Day (Hour)',
        xaxis={'tickmode': 'array', 'tickvals': [
            i for i in range(7)], 'ticktext': WEEKDAYS},
        yaxis={'tickmode': 'array', 'tickvals': TICK_VALS, 'ticktext': TICK_TEXT},
    )
    # Display the heatmap
    fig.show()


sentiment_heatmap(data)
```

Description Wordclouds

In [ ]:

```
def create_description_wordcloud(df):
    # Get the descriptions from the DataFrame
    descriptions = df["description"].dropna().astype(str).tolist()

    # Combine the descriptions into a single string
    combined_descriptions = " ".join(descriptions)

    # Remove common stopwords from the string
    stopwords = set(STOPWORDS)
    stopwords.update(["http", "https", "co", "com"])
    filtered_descriptions = " ".join(
        [word for word in combined_descriptions.split() if word.lower() not in stc

    # Create the word cloud using WordCloud
    wc = WordCloud(width=800, height=400, background_color="white").generate(filte

    # Convert the WordCloud object to a PIL image
    wc_image = wc.to_image()

    # Create a Plotly figure using the PIL image
    fig = px.imshow(wc_image, binary_string=True)

    # Update the layout of the word cloud
    fig.update_layout(
        title="Word cloud of user descriptions",
        xaxis=dict(visible=False),
        yaxis=dict(visible=False),
        hovermode=False
    )

    # Show the word cloud using Plotly
    fig.show()



create_description_wordcloud(data)
```