Assumptions:

- Used dataset correct_twitter_201904.tsv
- Used ts1 column for timestamp.

1. Prerequisites

For End-Users:

To access the deployed dashboard, users will only need:

- A **web browser** (Chrome, Firefox, Edge, etc.).

The **IP address** (or domain name) of the EC2 instance where the Streamlit app is deployed.
Example URL format:
Copy code

```
http://52.66.164.66:8501/
```

Users do not need to install anything locally; they can simply view the dashboard by navigating to the URL in their browser.

2. Code with Use Case and Comments

Below is the Python code for your Streamlit dashboard, which includes search functionality, dynamic graph updates based on the "text" column, and a scrolling feature for visual elements.

```
# Import necessary libraries for data manipulation, visualization, and Streamlit app
import pandas as pd
import streamlit as st
import plotly.express as px

# Load the Twitter dataset from a TSV file, using low memory to avoid memory overload
```

```
df_original = pd.read_csv("correct_twitter_201904.tsv", sep='\t', low_memory=False)

# Create an empty DataFrame to store selected columns and processed data
df = pd.DataFrame()

# Extract the 'ts2' column as a datetime type and handle any errors
df['date'] = pd.to_datetime(df_original[' ts2'], errors='coerce')

# Convert the 'like_count' column to numeric, replacing any invalid values with NaN
df['like_count'] = pd.to_numeric(df_original['like_count'], errors='coerce')

# Convert 'ts2' to time format (hours and minutes), and handle any format errors
df['time'] = pd.to_datetime(df_original[' ts2'], format='%H:%M', errors='coerce').dt.time

# Extract additional relevant columns from the original DataFrame
df['id'] = df_original['id']
df['place_id'] = df_original['place_id']
df['text'] = df_original['text']

# Set up the Streamlit app title
st.title('Tweet Analysis Dashboard')

# Create a search box where users can input a keyword to filter tweets by the 'text' column
search_term = st.text_input("Search for a keyword in tweets:", "")

# Filter the DataFrame based on the search term (case-insensitive)
filtered_df = df[df['text'].str.contains(search_term, case=False)]

# Print the column names of the filtered DataFrame for debugging purposes
print(filtered_df.columns)

# If there are tweets containing the search term, proceed with the visualizations
if not filtered_df.empty:

   # Graph 1: Group tweets by date and count the number of tweets per day
   tweets_per_day =
filtered_df.groupby(filtered_df['date'].dt.date).size().reset_index(name='Tweet Count')
   # Display the number of tweets per day using a line chart
   st.subheader(f"Number of Tweets containing '{search_term}' per day")
   fig1 = px.line(tweets_per_day, x='date', y='Tweet Count', title="Tweets Over Time")
   st.plotly_chart(fig1)

   # Graph 2: Count the number of unique users posting tweets each day
   unique_users = filtered_df.groupby('date')['id'].nunique().reset_index(name='Unique Users')
```

```
    # Display the number of unique users per day using a bar chart
    st.subheader(f"Number of Unique Users posting '{search_term}' per day")
    fig2 = px.bar(unique_users, x='date', y='Unique Users', title="Unique Users Per Day")
    st.plotly_chart(fig2)

    # Graph 3: Calculate the average likes per tweet for each day
    avg_likes = filtered_df.groupby('date')['like_count'].mean().reset_index(name='Average Likes')
    # Display the average number of likes per day using a line chart
    st.subheader(f"Average Likes for Tweets containing '{search_term}' per day")
    fig3 = px.line(avg_likes, x='date', y='Average Likes', title="Average Likes Over Time")
    st.plotly_chart(fig3)

    # Graph 4: Group tweets by 'place_id' and count the number of tweets for each location
    place_tweets = filtered_df.groupby('place_id').size().reset_index(name='Tweet Count')
    # Display the number of tweets by place using a bar chart
    st.subheader(f"Tweets containing '{search_term}' by Place")
    fig4 = px.bar(place_tweets, x='place_id', y='Tweet Count', title="Tweets by Place")
    st.plotly_chart(fig4)

    # Graph 5: Convert the 'time' column to hours and group tweets by time of day
    filtered_df['time'] = pd.to_datetime(filtered_df['time'], format='%H:%M').dt.hour
    tweets_by_time = filtered_df.groupby('time').size().reset_index(name='Tweet Count')
    # Display the number of tweets by time of day using a line chart
    st.subheader(f"Tweets containing '{search_term}' by Time of Day")
    fig5 = px.line(tweets_by_time, x='time', y='Tweet Count', title="Tweets by Time of Day")
    st.plotly_chart(fig5)

    # Identify the user who posted the most tweets containing the search term
    top_user = filtered_df['id'].value_counts().idxmax()
    st.subheader(f"The user who posted the most tweets containing '{search_term}' is: {top_user}")

# If no tweets contain the search term, display a message
else:
    st.write("No tweets found containing the search term.")
```

3. Code Walkthrough: Tweet Analysis Dashboard

This code creates a **Streamlit dashboard** that allows users to search for specific keywords in a dataset of tweets and provides various visualizations related to the search term, such as tweets per day, unique users, likes, locations, time of posting, and the most frequent poster.

1. Importing Libraries

```
import pandas as pd
import streamlit as st
import plotly.express as px
```

- **pandas (pd)**: Used for data manipulation and analysis.
- **streamlit (st)**: A framework for creating interactive web applications.
- **plotly.express (px)**: For creating visualizations and charts.

2. Loading and Preparing Data

```
df_original = pd.read_csv("correct_twitter_201904.tsv", sep='\t', low_memory=False)
df = pd.DataFrame()
```

- The dataset (`correct_twitter_201904.tsv`) is loaded into the `df_original` DataFrame.
- `low_memory=False` ensures the data is read into memory efficiently, avoiding type inference issues.
- An empty DataFrame `df` is created to store selected and processed columns from the original dataset.

3. Extracting and Cleaning Data

```
df['date'] = pd.to_datetime(df_original[' ts2'], errors='coerce')
df['like_count'] = pd.to_numeric(df_original['like_count'], errors='coerce')
df['time'] = pd.to_datetime(df_original[' ts2'], format='%H:%M', errors='coerce').dt.time
df['id'] = df_original['id']
df['place_id'] = df_original['place_id']
df['text'] = df_original['text']
```

- **`date`**: Converts the `ts2` column into a proper date format. Any invalid dates are replaced with NaT (Not a Time).
- **`like_count`**: Converts the `like_count` column into numeric values, replacing invalid entries with NaN.
- **`time`**: Extracts the time (hours and minutes) from the `ts2` column and converts it to a `time` data type.
- Other columns such as `id`, `place_id`, and `text` are selected for analysis.

4. Setting Up the Dashboard

```
st.title('Tweet Analysis Dashboard')
```

- This sets the title of the Streamlit app to "Tweet Analysis Dashboard."

5. Creating the Search Box

python

Copy code

```
search_term = st.text_input("Search for a keyword in tweets:", "")
```

- A **text input box** is created where users can type a keyword to search for within the tweets' text.
- The input value is stored in the variable `search_term`.

6. Filtering Data Based on Search Term

```
filtered_df = df[df['text'].str.contains(search_term, case=False)]
```

- Filters the DataFrame `df` to include only rows where the `text` column contains the `search_term`. This is case-insensitive (`case=False`).
- The filtered DataFrame is stored in `filtered_df`.

7. Checking if Data is Available

```
if not filtered_df.empty:
```

- Checks if the `filtered_df` is not empty. If there are results, visualizations will be generated. Otherwise, a message will be displayed.

8. Visualization 1: Tweets Per Day

```
tweets_per_day = filtered_df.groupby(filtered_df['date'].dt.date).size().reset_index(name='Tweet Count')
st.subheader(f"Number of Tweets containing '{search_term}' per day")
fig1 = px.line(tweets_per_day, x='date', y='Tweet Count', title="Tweets Over Time")
st.plotly_chart(fig1)
```

- The tweets are grouped by date, and the number of tweets is counted for each day.
- A **line chart** is created showing the number of tweets over time.
- The chart is displayed using `st.plotly_chart()`.

9. Visualization 2: Unique Users Per Day

```
unique_users = filtered_df.groupby('date')['id'].nunique().reset_index(name='Unique Users')
st.subheader(f"Number of Unique Users posting '{search_term}' per day")
```

  - `fig2 = px.bar(unique_users, x='date', y='Unique Users', title="Unique Users Per Day")`
  - `st.plotly_chart(fig2)`

- The unique users posting tweets are counted for each day.
- A **bar chart** is generated showing the number of unique users per day.
- The chart is displayed using `st.plotly_chart()`.

10. Visualization 3: Average Likes Per Day

  - `avg_likes = filtered_df.groupby('date')['like_count'].mean().reset_index(name='Average Likes')`
  - `st.subheader(f"Average Likes for Tweets containing '{search_term}' per day")`
  - `fig3 = px.line(avg_likes, x='date', y='Average Likes', title="Average Likes Over Time")`
  - `st.plotly_chart(fig3)`

- The average number of likes per tweet is calculated for each day.
- A **line chart** is generated showing the average likes over time.
- The chart is displayed using `st.plotly_chart()`.

11. Visualization 4: Tweets by Place

  - `place_tweets = filtered_df.groupby('place_id').size().reset_index(name='Tweet Count')`
  - `st.subheader(f"Tweets containing '{search_term}' by Place")`
  - `fig4 = px.bar(place_tweets, x='place_id', y='Tweet Count', title="Tweets by Place")`
  - `st.plotly_chart(fig4)`

- The tweets are grouped by `place_id`, counting how many tweets were posted from each location.
- A **bar chart** is generated showing the number of tweets by place.
- The chart is displayed using `st.plotly_chart()`.

12. Visualization 5: Tweets by Time of Day

  - `filtered_df['time'] = pd.to_datetime(filtered_df['time'], format='%H:%M').dt.hour`
  - `tweets_by_time = filtered_df.groupby('time').size().reset_index(name='Tweet Count')`
  - `st.subheader(f"Tweets containing '{search_term}' by Time of Day")`
  - `fig5 = px.line(tweets_by_time, x='time', y='Tweet Count', title="Tweets by Time of Day")`
  - `st.plotly_chart(fig5)`

- The tweets are grouped by the hour of the day to analyze what time of day tweets were posted.
- A **line chart** is generated showing the number of tweets posted at different times of day.
- The chart is displayed using `st.plotly_chart()`.

13. Finding the User Who Posted the Most Tweets

  - `top_user = filtered_df['id'].value_counts().idxmax()`
  - `st.subheader(f"The user who posted the most tweets containing '{search_term}' is: {top_user}")`

- The `id` of the user who posted the most tweets containing the search term is identified.
- A subheader displays the ID of the top user.

14. No Results Found Case

  - `else:`
  - `    st.write("No tweets found containing the search term.")`

- If no tweets containing the search term are found, a message is displayed saying no results were found.
    -

---

4. Deployment on AWS EC2

Deployed the dashboard on AWS EC2 instance so that It will be accessible for the end user.