Introduction

Sentiment Analysis helps computers understand whether a piece of text expresses a positive, negative, or neutral feeling. In this project, we apply sentiment analysis to tweets from Twitter using tools like TextBlob and VADER.

Our interactive Streamlit app lets users:

- Analyze the sentiment of a single tweet.
- Upload a CSV of tweets for batch analysis.
- Visualize sentiment counts with charts.
- Switch between sentiment models for comparison.

Real-World Uses

- Brand Monitoring: Know how people feel about a product or service.
- Social Media Tracking: Understand public opinion during events or trends.
- Customer Feedback: Quickly summarize reviews or support messages.
- Stock Market Insights: Gauge investor mood based on tweets.
- Politics & Policy: Analyze voter sentiment or reactions to decisions.
- Emergency Response: Detect distress or panic during crises.

Understand what kind of data we are working with and prepare for cleaning and analysis.

Import Required Libraries

```
import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

- pandas as pd: A powerful Python library for working with data, especially tables like spreadsheets. pd is just a shortcut so we don't type pandas every time.
- matplotlib.pyplot as plt: This is used to make charts and graphs. Think of it as your paintbrush for data.
- seaborn as sns: A prettier and easier way to draw charts. Built on top of matplotlib.

Load the Dataset (CSV File)

```
df = pd.read_csv("data/Tweets.csv")
```

- pd.read_csv(...): This reads a CSV (comma-separated values) file – just like an Excel sheet – and loads it into a table format called a **DataFrame**, stored in the variable df.
- "data/Tweets.csv": This is the file path. It says "Go into the data folder and find Tweets.csv".

Preview the First 5 Rows

```
df.head()
```

- This displays the **first 5 rows** of the dataset so you can see what kind of data you're working with (columns like tweet text and sentiment).

Understand the Structure of the Data

```
df.info()
```

- This tells you:
    - How many rows and columns the dataset has.
    - What data types each column contains (like text or numbers).
    - If there are any **missing values** (like empty cells).

Count Missing Data

```
print(df.isnull().sum())
```

- Checks each column to see how many missing or null (empty) values exist.
- isnull() returns True for missing values.
- .sum() adds them up for each column.

Check for Duplicate Tweets

```
print("\nDuplicate rows: ", df.duplicated().sum())
```

- df.duplicated() finds repeated rows (same tweet more than once).
- .sum() counts how many duplicates there are.

List All Column Names

```
print(df.columns.tolist())
```

- Shows all the column names in a list format.
- Helps you know what kind of information is stored – like text (tweet content), sentiment (positive/negative/neutral), etc.

Visualize Sentiment Distribution

```
sns.countplot(x='sentiment', data=df, palette='viridis')
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Number of Tweets')
plt.show()
```

This draws a **bar chart** to show:

- How many tweets are **Positive**, **Negative**, or **Neutral**.
- sns.countplot(...) makes the bar chart.
- palette='viridis' gives it pretty colors.
- plt.title, xlabel, ylabel label the chart.
- plt.show() displays it.

Why this matters: It gives you a quick summary of public opinion based on tweets.

Print Example Tweets by Sentiment

```
for sentiment in df['sentiment'].unique():
  print(f"\n--- {sentiment.upper()} TWEETS ---")
  print(df[df['sentiment'] == sentiment]['text'].head(3).tolist())
```

Let's break this down:

- df['sentiment'].unique() gets all the unique sentiment labels like 'Positive', 'Negative', 'Neutral'.
- for sentiment in ...: loops through each type.
- df[df['sentiment'] == sentiment] filters the dataset to include only tweets of that sentiment.
- ['text'].head(3).tolist() gets the first 3 tweets of that sentiment and puts them in a list.

- print(...) prints out a few example tweets for each category.

Why this is useful: It helps you see what kind of tweets people post under each sentiment-real!

Make the text easier to analyze by removing unnecessary or distracting parts.

Import Required Libraries

```
import re

import string
```

What each one does:

- re: Python's **regular expression** library — great for **searching and removing text patterns**, like links or hashtags.
- string: Provides easy access to things like punctuation marks so we can **remove them**.

Define a Function to Clean Tweets

```
def clean_text(text):

  if not isinstance(text, str):  # Skip if it's not a string

    return ""
```

- This function will **take one tweet as input**.
- If it's not a text string (maybe it's a number or empty), it just returns an empty string.

Now, the actual cleaning steps:

```
# Remove URLs (like http://... or www...)

text = re.sub(r'http\S+|www\S+', '', text)
```

- Deletes **links** in tweets — we don't need them for analyzing the tweet's meaning.

```
# Remove mentions and hashtags
text = re.sub(r'@\w+', '', text)    # @mentions
text = re.sub(r'#\w+', '', text)    # hashtags
```

- Deletes @username mentions and #hashtags, which can distract from the real sentiment.

```
# Remove punctuation
text = text.translate(str.maketrans('', '', string.punctuation))
```

- Removes symbols like ., !, ?, , etc.

```
# Remove numbers
text = re.sub(r'\d+', '', text)
```

- Removes any **numbers**, which usually don't help with emotion analysis.

```
# Remove extra whitespace
text = re.sub(r'\s+', ' ', text).strip()
```

- If the tweet had multiple spaces, it **converts them into a single space** and trims the ends.

```
# Convert to lowercase
text = text.lower()
```

- Makes everything lowercase (HELLO → hello). Helps in treating "Happy" and "happy" as the same word.

Apply Cleaning to All Tweets

```
df['cleaned_text'] = df['text'].apply(clean_text)
```

- We now run our clean_text() function **on every tweet** in the dataset.
- The result is stored in a **new column** called 'cleaned_text'.

View the Original vs. Cleaned Tweets

```
df[['text', 'cleaned_text']].head(10)
```

- This shows the **first 10 rows** side by side:
    - text is the original messy tweet.
    - cleaned_text is the cleaned version.

This is helpful to **see what got removed.**

What we did:

- Removed:
    - **URLs**
    - **Mentions (@username)**
    - **Hashtags**
    - **Punctuation**
    - **Numbers**
    - **Extra whitespace**
- Lowercased all text
- Stored the cleaned version in a new column (cleaned_text)
- Generated a Word Cloud to show the most common words after cleaning

Why it's useful:
Raw tweets are full of noise — symbols, emojis, URLs — which hurt analysis.
Cleaning ensures we only work with meaningful words. The WordCloud then gives a quick idea of what people are talking about.

Use a basic AI model (TextBlob) to predict sentiment, and then visualize the findings.

Import Libraries (Tools You Need)

```
import pandas as pd
from textblob import TextBlob
import seaborn as sns
import matplotlib.pyplot as plt
import re
import string
```

What each one is for:

TextBlob: A simple tool that can read a sentence and **guess the emotion** (positive, negative, neutral).

Analyze the Sentiment (Emotion) of Each Tweet

Define a function that uses TextBlob:

```
def get_sentiment(text):
    analysis = TextBlob(text)
    polarity = analysis.sentiment.polarity
```

- TextBlob(text) creates a sentiment object from the cleaned text.
- .sentiment.polarity gives a number between -1 and 1:
    - **> 0 = Positive**
    - **< 0 = Negative**
    - **0 = Neutral**

Decide the label:

```
if polarity > 0:
    return 'Positive'
elif polarity < 0:
    return 'Negative'
else:
    return 'Neutral'
```

- Converts that number into a word label (Positive, Negative, Neutral).

Apply to all tweets:

```
df['sentiment'] = df['cleaned_text'].apply(get_sentiment)
```

- Runs the get_sentiment() function on every cleaned tweet.
- Stores the result in a new column called 'sentiment'.

Count How Many Tweets in Each Category

```
sentiment_counts = df['sentiment'].value_counts().reset_index()
sentiment_counts.columns = ['Sentiment', 'Count']
```

- Counts how many tweets are **Positive, Negative, or Neutral**.
- Prepares this in a clean table called sentiment_counts.

Draw a Bar Chart of Sentiments

```
plt.figure(figsize=(8, 5))
sns.barplot(data=sentiment_counts, x='Sentiment', y='Count', palette='pastel')
plt.title('Sentiment Distribution')
plt.ylabel('Number of Tweets')
plt.xlabel('Sentiment')
plt.tight_layout()
plt.show()
```

- Makes a **bar chart** using Seaborn.
- palette='pastel' gives it nice light colors.
- X-axis = Sentiment (Positive/Negative/Neutral).
- Y-axis = How many tweets fall in each.

Create a Word Cloud

Import WordCloud:

```
from wordcloud import WordCloud
```

Combine all cleaned tweets into one big string:

```
all_words = ' '.join(df['cleaned_text'])
```

Generate the WordCloud:

```
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate(all_words)
```

- Creates an image of words.
- Words that appear **more often** are shown **bigger**.

Display the WordCloud:

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Most Frequent Words in Tweets", fontsize=16)
plt.tight_layout()
plt.show()
```

- This draws the actual image.
- axis('off') hides the graph lines because it's a word image.
- imshow() displays the word cloud.
- tight_layout() ensures nothing overlaps.

What we did:

- Used the TextBlob library to:
    - Calculate **polarity** of each cleaned tweet
    - Classify each tweet as **Positive**, **Negative**, or **Neutral**
- Added this result to a new column (sentiment)
- Counted the number of tweets per sentiment
- Created a bar chart to visualize the distribution
- Reused the WordCloud to highlight top words across all tweets

Why it's useful:
We transform unstructured text into structured insights:

- See how people feel in general (are they happy or angry?)
- Track trends over time (if applied to live data)
- Provide visual feedback to stakeholders

Sentiment analysis mini app

Above functions are the **engine** behind the app.

clean_text(text)

This cleans a tweet before analysis by:

- Removing links, mentions, hashtags, punctuation, numbers, extra spaces
- Lowercasing the text

This is the **same cleaning function** from your earlier code block. It ensures that input tweets are clean and consistent before any model (TextBlob/VADER) is applied.

---

analyze_textblob(text)

- Uses TextBlob to get **polarity score**:
    - 0 → Positive
    - < 0 → Negative
    - = 0 → Neutral

This replicates your third code block's sentiment analysis logic using TextBlob.

---

analyze_vader(text)

- Uses **VADER (Valence Aware Dictionary for Sentiment Reasoning)**.
- Analyzes sentiment and returns:
    - Positive (compound score ≥ 0.05)
    - Negative (compound score ≤ -0.05)
    - Neutral (in-between)

You introduced VADER here to give **users a choice** between two models.

Streamlit UI Section

This turns the sentiment analysis code into a **web interface** users can interact with easily — without writing code.

```
st.title("Tweet Sentiment Analyzer")
```

Sets the main app title.

Sidebar: Model & File Upload

```
st.sidebar.selectbox("Choose Sentiment Model", ["TextBlob", "VADER"])
```

- Adds a dropdown menu so the user can **choose which sentiment model** to use.
- TextBlob = rule-based + machine learning
- VADER = lexicon-based, designed for social media

```
st.sidebar.file_uploader(...)
```

- Lets users **upload a CSV file** that contains tweets for bulk analysis.
- Must have a column named text

Single Tweet Analysis Section

```
st.text_area("Enter a tweet")
```

- User inputs **a single tweet**.

When "Analyze Sentiment" button is clicked:

- The tweet is cleaned (clean_text)
- Chosen model is applied (TextBlob or VADER)
- Sentiment result is shown immediately using st.write()

This gives **instant feedback** to the user.

Batch Tweet Analysis Section

```
if uploaded_file: ...
```

This section handles **CSV input** from the user for batch analysis.

What happens:

1. File is read using Pandas (pd.read_csv)
2. It checks for a 'text' column — error shown if missing
3. Clean text using the same clean_text() function
4. Apply the selected sentiment model on all tweets
5. Display first 10 rows of results (original + sentiment)

Visualization: Sentiment Distribution

```
sentiment_counts = df['sentiment'].value_counts()

fig, ax = plt.subplots()

sentiment_counts.plot(kind='bar', color=['green', 'red', 'gray'], ax=ax)

st.pyplot(fig)
```

- Counts how many tweets are Positive/Negative/Neutral.
- Creates a **bar chart** using Matplotlib and shows it using st.pyplot.
- Green = Positive, Red = Negative, Gray = Neutral (for visual clarity)