✅ Step 1: Import Libraries and Initialize Twitter Client

Code:

python

```python
import os
import re
import json
import tweepy
from textblob import TextBlob
from langdetect import detect

# Load Twitter credentials
bearer_token = os.getenv("BEARER_TOKEN")
if not bearer_token:
    raise ValueError("Missing BEARER_TOKEN in environment variables.")

# Initialize Tweepy client
try:
    client = tweepy.Client(bearer_token=bearer_token)
    print("Twitter API client initialized successfully.")
except Exception as e:
    print("Error connecting to Twitter API:", e)
    exit(1)
```

📘 Explanation:

`import os`

- Used to interact with the operating system.

- `os.getenv("BEARER_TOKEN")` fetches environment variables securely (used here to avoid hardcoding sensitive data like API keys).

`import re`

- Used for **regular expressions**. You'll use this to clean text data from tweets.

`import json`

- Enables **reading and writing** JSON files — useful for storing processed tweets in a structured format.

`import tweepy`

- A Python wrapper for the **Twitter API**. Used here to search for recent tweets.

`from textblob import TextBlob`

- A simple NLP tool for performing **sentiment analysis**, part-of-speech tagging, and more.
- We use it here to analyze how positive or negative a tweet is.

`from langdetect import detect`

- Detects the **language** of a given text.
- We use it to filter only English tweets.

✅ Step 2: Define Tweet Cleaning Function

Code:

```python
# Function to clean tweet text
def clean_tweet(tweet):
    tweet = re.sub(r"http\S+", "", tweet)  # remove URLs
    tweet = re.sub(r"@\S+", "", tweet)     # remove mentions
    tweet = re.sub(r"#", "", tweet)        # remove hashtags symbol (not the word)
    tweet = re.sub(r"\n", " ", tweet)      # replace newlines with space
    return tweet.strip()                   # remove leading/trailing whitespace
```

📘 Explanation:

- `re.sub(pattern, replacement, text)` is used for **text substitution**.

- Removes unwanted characters (like links, mentions, hashtags) to prepare for clean analysis.
- This improves accuracy for both language detection and sentiment analysis.

✅ Step 3: Fetch Tweets from Twitter API

Code:

```python
print("Fetching tweets about 'python'...")

try:
    response = client.search_recent_tweets(query="python", max_results=50)
```

📘 Explanation:

- This searches Twitter for the **50 most recent tweets** that mention `"python"`.
- `query="python"` is the search keyword.
- `max_results=50` limits the result set (max 100 allowed by Twitter per request).
- `client.search_recent_tweets()` is from **Tweepy** and connects to the Twitter **Recent Search** API.

✅ Step 4: Analyze and Process the Tweets

Code:

```python
    if response.data:
        cleaned_tweets = []

        for tweet in response.data:
            raw_text = tweet.text
            cleaned = clean_tweet(raw_text)
```

```
        try:
            if detect(cleaned) == "en":
                blob = TextBlob(cleaned)
                polarity = round(blob.sentiment.polarity, 3)
                sentiment = (
                    "positive" if polarity > 0 else
                    "negative" if polarity < 0 else
                    "neutral"
                )
                cleaned_tweets.append({
                    "text": cleaned,
                    "polarity": polarity,
                    "sentiment": sentiment
                })
        except:
            continue
```

📘 Explanation:

- `response.data` contains tweet data objects.
- `cleaned_tweets = []` is a list to store processed tweet data.
- `clean_tweet(raw_text)` removes unnecessary characters.
- `detect(cleaned)` ensures only English tweets are analyzed.
- `TextBlob(cleaned)` creates a sentiment object.
- `blob.sentiment.polarity` gives a value between -1 (negative) and +1 (positive).
- Sentiment is categorized based on polarity:
  - `> 0` : Positive
  - `< 0` : Negative
  - `== 0` : Neutral
- Each processed tweet is stored in a dictionary and added to `cleaned_tweets`.

✅ Step 5: Output and Save Processed Tweets

Code:

```python
        # Print to console
        for tweet in cleaned_tweets:
            print("Tweet:", tweet["text"])
            print("Sentiment:", tweet["sentiment"], "Polarity:", tweet["polarity"])
            print("-" * 60)

        # Save to JSON
        with open("tweets.json", "w", encoding="utf-8") as f:
            json.dump(cleaned_tweets, f, ensure_ascii=False, indent=2)

        print(f"{len(cleaned_tweets)} tweets saved to tweets.json.")

    else:
        print("No tweets found.")

except tweepy.TooManyRequests:
    print("Rate limit exceeded. Please try again later.")
except Exception as e:
    print("Error occurred while fetching tweets:", e)
```

📘 Explanation:

- Each tweet and its sentiment is printed to the console for live feedback.
- `json.dump(..., indent=2)` formats and saves data as a readable `.json` file.
- Error handling:
  - `TooManyRequests`: Avoids crashing if Twitter's **rate limit** is exceeded.
  - Generic `Exception`: Catches unexpected issues.

✅ Final Step for Students:

Once the five steps are understood and tested individually, ask students to:

1. **Set `BEARER_TOKEN` as an environment variable**.
2. **Run the complete script** as a single `.py` file to observe sentiment analysis in action.

Codeboosters Tech

team_codeboosters

www.codeboosters.in

✅ Code Block: Tweet Sentiment Analysis

📌 Full Code (for reference)

python

```
if response.data:
    cleaned_tweets = []

    for tweet in response.data:
        raw_text = tweet.text
        cleaned = clean_tweet(raw_text)

        try:
            if detect(cleaned) == "en":
                blob = TextBlob(cleaned)
                polarity = round(blob.sentiment.polarity, 3)
                sentiment = (
                    "positive" if polarity > 0 else
                    "negative" if polarity < 0 else
                    "neutral"
                )
                cleaned_tweets.append({
                    "text": cleaned,
                    "polarity": polarity,
                    "sentiment": sentiment
                })
        except:
            continue
```

🔍 Line-by-Line Explanation

`if response.data:`

- This checks if Twitter returned any tweets.
- `response.data` is a list of `Tweet` objects from the Twitter API.
- If it's empty (`None` or `[]`), we skip the rest.

`cleaned_tweets = []`

- Initializes an empty list to store the **final results** (cleaned tweets + analysis).

`for tweet in response.data:`

- Loop through each tweet returned by the API.

`raw_text = tweet.text`

- Extracts the raw tweet text from each tweet object.

`cleaned = clean_tweet(raw_text)`

- Calls your `clean_tweet()` function to:
  - Remove URLs, mentions, hashtags, line breaks.
  - Make the text easier and more accurate for language detection and sentiment analysis.

`try:`

- Begins a `try` block to catch and handle exceptions.
- Some tweets may cause errors in language detection or analysis (e.g., gibberish text).

`if detect(cleaned) == "en":`

- Uses `langdetect.detect()` to check if the cleaned tweet is in **English**.
- Sentiment analysis tools like `TextBlob` are **trained on English text**, so this filters out non-English content.

`blob = TextBlob(cleaned)`

- `TextBlob` creates a **sentiment object** for the tweet.
- Internally, it uses a **lexicon-based approach** (dictionary of words with polarity values).

`polarity = round(blob.sentiment.polarity, 3)`

- `blob.sentiment.polarity` returns a **float value between -1.0 and +1.0**.
  - `-1.0` = very negative
  - `0.0` = neutral
  - `+1.0` = very positive
- Example:
  - `"I love Python"` → polarity = `0.5`
  - `"Python is terrible"` → polarity = `-0.7`
- `round(..., 3)` keeps 3 decimal places for neat output.

`sentiment = (...)`

- This block assigns a **label** based on the polarity value:

- `positive` if polarity > 0
- `negative` if polarity < 0
- `neutral` if polarity == 0

- It turns raw numbers into human-friendly sentiment labels.

`cleaned_tweets.append({...})`

- Appends a dictionary with the cleaned tweet, its polarity, and the sentiment label to the results list.

`except: continue`

- If any error occurs (e.g., `detect()` fails or weird characters), it skips that tweet and continues.

📊 About Polarity in Sentiment Analysis

Polarity Value	Interpretation
-1.0 to -0.1	Negative sentiment
-0.1 to +0.1	Neutral sentiment (near zero)
+0.1 to +1.0	Positive sentiment

How is it used?

- TextBlob uses a **sentiment lexicon**, scoring individual words and combining them.
- For example:
  - `"love"` = +0.5
  - `"hate"` = -0.8
  - `"not good"` = negative (due to negation handling)

🧪 Example Output

Imagine the following three tweets are returned:

```json
[
  {
    "text": "I love coding in Python! 🐍",
    "polarity": 0.625,
    "sentiment": "positive"
  },
  {
    "text": "Python's indentation rules are so annoying.",
    "polarity": -0.3,
    "sentiment": "negative"
  },
  {
    "text": "Python is a programming language.",
    "polarity": 0.0,
    "sentiment": "neutral"
  }
]
```

Each one shows:

- Cleaned text
- Computed polarity
- Labeled sentiment

JSON File Storage

✅ Code Block

python

```python
# Print to console
for tweet in cleaned_tweets:
    print("Tweet:", tweet["text"])
    print("Sentiment:", tweet["sentiment"], "Polarity:", tweet["polarity"])
    print("-" * 60)

# Save to JSON
with open("tweets.json", "w", encoding="utf-8") as f:
    json.dump(cleaned_tweets, f, ensure_ascii=False, indent=2)

print(f"{len(cleaned_tweets)} tweets saved to tweets.json.")

else:
    print("No tweets found.")

except tweepy.TooManyRequests:
    print("Rate limit exceeded. Please try again later.")
except Exception as e:
    print("Error occurred while fetching tweets:", e)
```

🔍 Line-by-Line Detailed Explanation

👇 Console Output Section

python

```python
for tweet in cleaned_tweets:
    print("Tweet:", tweet["text"])
```

```
    print("Sentiment:", tweet["sentiment"], "Polarity:", tweet["polarity"])
    print("-" * 60)
```

- Iterates through each processed tweet stored in `cleaned_tweets`.
- `tweet["text"]` prints the cleaned tweet.
- `tweet["sentiment"]` shows the assigned sentiment: `"positive"`, `"neutral"`, or `"negative"`.
- `tweet["polarity"]` is the sentiment strength, a float between -1.0 and 1.0.
- `print("-" * 60)` prints a line of 60 dashes to visually separate tweets in the console.

👇 Saving to JSON File

```python
with open("tweets.json", "w", encoding="utf-8") as f:
    json.dump(cleaned_tweets, f, ensure_ascii=False, indent=2)
```

💡 Explanation of Key Parts:

`with`:

- **Context manager**: Automatically handles opening and closing files.
- Ensures the file is properly closed even if an error occurs during the write.
- Cleaner and safer than manually using `f = open(...)` + `f.close()`.

`open("tweets.json", "w", encoding="utf-8")`:

- `open(...)` opens a file named `tweets.json`.
- `"w"` means **write mode**:
  - If the file exists, it is **overwritten**.
  - If the file does not exist, it is **created**.
- `encoding="utf-8"` ensures that all characters (including emojis and non-English letters) are correctly saved.

`as f`:

- Assigns the opened file object to variable `f` so you can use it inside the `with` block.

`json.dump(...)`:

- Serializes the list of dictionaries (`cleaned_tweets`) into JSON format.
- `ensure_ascii=False`: Allows Unicode characters (like emojis) to be saved properly.
- `indent=2`: Makes the JSON output nicely formatted and human-readable (2-space indentation).

👇 Final Summary Output

```python
print(f"{len(cleaned_tweets)} tweets saved to tweets.json.")
```

- `len(cleaned_tweets)` returns the number of tweets processed.
- Uses an **f-string** to include that count in the message.

👇 If No Tweets Were Found

```python
else:
    print("No tweets found.")
```

- This `else` is paired with the `if response.data:` condition.
- If no tweets were returned by the Twitter API, it prints this message.

👇 Error Handling

```python
```

```
except tweepy.TooManyRequests:
    print("Rate limit exceeded. Please try again later.")
except Exception as e:
    print("Error occurred while fetching tweets:", e)
```

- Handles specific error if the **Twitter API rate limit** is exceeded.
- Generic `except Exception` block catches any other kind of error, printing the message for debugging.

🧪 Sample Output on Console

vbnet

```
Tweet: I love Python programming. It's amazing!
Sentiment: positive Polarity: 0.5
------------------------------------------------------------
Tweet: Python is okay, not too exciting though.
Sentiment: neutral Polarity: 0.0
------------------------------------------------------------
Tweet: I hate dealing with Python errors!
Sentiment: negative Polarity: -0.6
------------------------------------------------------------
3 tweets saved to tweets.json.
```

📄 Sample `tweets.json` File Output

json


```
[
  {
    "text": "I love Python programming. It's amazing!",
    "polarity": 0.5,
    "sentiment": "positive"
  },
  {
    "text": "Python is okay, not too exciting though.",
```

```
    "polarity": 0.0,
    "sentiment": "neutral"
  },
  {
    "text": "I hate dealing with Python errors!",
    "polarity": -0.6,
    "sentiment": "negative"
  }
]
```

✅ Full Script for Reference

html

```
<script>
  fetch("tweets.json")
    .then(res => res.json())
    .then(data => {
      const container = document.getElementById("tweets");
      container.innerHTML = "";
      data.forEach(tweet => {
        const div = document.createElement("div");
        div.className = "tweet";
        div.textContent = tweet.text;
        container.appendChild(div);
      });
    });
</script>
```

🔍 Line-by-Line Detailed Explanation

`fetch("tweets.json")`

- This uses the **Fetch API**, a modern JavaScript feature to **load external resources asynchronously**.
- It sends a **GET request** to retrieve the `tweets.json` file that your Python script saved.
- This file must be located in the same directory as the HTML file unless a path is specified.

🔁 **What it does:**

- Starts a **promise chain**: when `tweets.json` is loaded, the promise resolves with a **Response object**.

`.then(res => res.json())`

- `res` here is the **HTTP Response** returned by `fetch`.
- `.json()` is a method that reads the body of the response and parses it as **JSON**.
- This returns another promise which resolves to the actual JavaScript array of tweets.

🔁 **Why?**

- Because `tweets.json` is in JSON format, you must parse it to convert it into a usable JS object/array.

`.then(data => {`

- `data` here is now the **JavaScript array of tweet objects**, like:

```js
[
  { text: "I love Python!", polarity: 0.5, sentiment: "positive" },
```

```
  { text: "Python is tough.", polarity: -0.3, sentiment: "negative" }
]
```

`const container = document.getElementById("tweets");`

- Finds the `<div id="tweets">` in your HTML (initially says "Loading...").
- This is where all the tweets will be displayed.

`container.innerHTML = "";`

- Clears the `"Loading..."` text.
- Ensures the container is empty before adding new tweets.

`data.forEach(tweet => {`

- Iterates through each tweet object in the `data` array.

🛠 Inside the Loop (for each tweet):

`const div = document.createElement("div");`

- Dynamically **creates a new** `<div>` **element** for each tweet.
- This is done in JavaScript instead of writing static HTML.

`div.className = "tweet";`

- Assigns the CSS class `tweet` to the `<div>`, which gives it styling:

```
css
```

```
.tweet {
  padding: 10px;
  border-bottom: 1px solid #ccc;
}
```

`div.textContent = tweet.text;`

- Sets the **inner text** of the tweet's div to the tweet's actual text (i.e., `tweet.text`).
- This avoids HTML injection because `textContent` treats it as plain text.

`container.appendChild(div);`

- Appends the new tweet `<div>` to the main container `<div id="tweets">`.
- This adds the tweet to the web page.

🧪 Example Output in Browser

If `tweets.json` contains:

```json
[
  { "text": "I love Python!" },
  { "text": "Python is awesome for data analysis!" }
]
```

You'll see this in the browser:

```markdown
Recent Tweets about Python

I love Python!
--------------------------------------
```

```
Python is awesome for data analysis!
-------------------------------------
```

Each line is in a separate styled `<div>` block under the heading.
