**Code:**

**Sentiment Analysis for twitter data**

import streamlit as st

import pickle

import re

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.corpus import stopwords

import nltk

import time

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.options import Options

# Download stopwords once, using Streamlit's caching

@st.cache\_resource

def load\_stopwords():

nltk.download('stopwords')

return stopwords.words('english')

# Load model and vectorizer once

@st.cache\_resource

def load\_model\_and\_vectorizer():

with open('model.pkl', 'rb') as model\_file:

model = pickle.load(model\_file)

with open('vectorizer.pkl', 'rb') as vectorizer\_file:

vectorizer = pickle.load(vectorizer\_file)

return model, vectorizer

# Define sentiment prediction function

def predict\_sentiment(text, model, vectorizer, stop\_words):

# Preprocess text

text = re.sub('[^a-zA-Z]', ' ', text)

text = text.lower()

text = text.split()

text = [word for word in text if word not in stop\_words]

text = ' '.join(text)

text = [text]

text = vectorizer.transform(text)

# Predict sentiment

sentiment = model.predict(text)

return "Negative" if sentiment == 0 else "Positive"

# Function to scrape a tweet from a tweet URL using Selenium

def scrape\_tweet(tweet\_url):

# tweet\_id = tweet\_url.split("/")[-1] # Extract tweet ID

url = f"{tweet\_url}" # Updated URL for X.com

# Set up Selenium WebDriver (using Chrome)

options = Options()

options.headless = True # Run in headless mode (no browser window)

driver = webdriver.Chrome(options=options)

try:

driver.get(url)

time.sleep(3) # Allow time for the page to load the tweet content

# Find the tweet content by inspecting the page structure

tweet\_text = None

tweet\_div = driver.find\_element(By.XPATH, "//div[@data-testid='tweetText']") # Updated XPath

tweet\_text = tweet\_div.text

return tweet\_text if tweet\_text else None

except Exception as e:

print(f"Error: {e}")

return None

finally:

driver.quit() # Close the browser

# Function to create a colored card

def create\_card(tweet\_text, sentiment):

color = "green" if sentiment == "Positive" else "red"

card\_html = f"""

<div style="background-color: {color}; padding: 10px; border-radius: 5px; margin: 10px 0;">

<h5 style="color: white;">{sentiment} Sentiment</h5>

<p style="color: white;">{tweet\_text}</p>

</div>

"""

return card\_html

# Main app logic

def main():

st.title("X.com Sentiment Analysis")

# Load stopwords, model, and vectorizer only once

stop\_words = load\_stopwords()

model, vectorizer = load\_model\_and\_vectorizer()

# User input: either text input or tweet link

st.subheader("Enter a Tweet URL")

tweet\_url = st.text\_input("Paste the tweet link here")

if st.button("Fetch Tweet"):

tweet\_text = scrape\_tweet(tweet\_url)

if tweet\_text:

sentiment = predict\_sentiment(tweet\_text, model, vectorizer, stop\_words) # Predict sentiment of the tweet text

# Create and display the colored card for the tweet

card\_html = create\_card(tweet\_text, sentiment)

st.markdown(card\_html, unsafe\_allow\_html=True)

else:

st.write("Could not fetch tweet or an error occurred.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Sentiment Analysis for customer review**

import streamlit as st

import pickle

import re

from sklearn.feature\_extraction.text import TfidfVectorizer

from nltk.corpus import stopwords

import nltk

# Download stopwords once, using Streamlit's caching

@st.cache\_resource

def load\_stopwords():

nltk.download('stopwords')

return stopwords.words('english')

# Load model and vectorizer once

@st.cache\_resource

def load\_model\_and\_vectorizer():

with open('model.pkl', 'rb') as model\_file:

model = pickle.load(model\_file)

with open('vectorizer.pkl', 'rb') as vectorizer\_file:

vectorizer = pickle.load(vectorizer\_file)

return model, vectorizer

# Define sentiment prediction function

def predict\_sentiment(text, model, vectorizer, stop\_words):

# Preprocess text

text = re.sub('[^a-zA-Z]', ' ', text) # Remove non-alphabetic characters

text = text.lower() # Convert to lowercase

text = text.split() # Tokenize into words

text = [word for word in text if word not in stop\_words] # Remove stopwords

text = ' '.join(text) # Join words back into a single string

text = [text] # Convert to list for vectorizer

text = vectorizer.transform(text) # Transform using vectorizer

# Predict sentiment

sentiment = model.predict(text)

return "Negative" if sentiment == 0 else "Positive"

# Function to create a colored card

def create\_card(review\_text, sentiment):

color = "green" if sentiment == "Positive" else "red"

card\_html = f"""

<div style="background-color: {color}; padding: 10px; border-radius: 5px; margin: 10px 0;">

<h5 style="color: white;">{sentiment} Sentiment</h5>

<p style="color: white;">{review\_text}</p>

</div>

"""

return card\_html

# Main app logic

def main():

st.title("Customer Review Sentiment Analysis")

# Load stopwords, model, and vectorizer only once

stop\_words = load\_stopwords()

model, vectorizer = load\_model\_and\_vectorizer()

# User input: customer review text

st.subheader("Enter a Customer Review")

review\_text = st.text\_area("Type or paste the review here")

if st.button("Analyze Sentiment"):

if review\_text.strip(): # Ensure input is not empty

sentiment = predict\_sentiment(review\_text, model, vectorizer, stop\_words)

# Display result in a colored card

card\_html = create\_card(review\_text, sentiment)

st.markdown(card\_html, unsafe\_allow\_html=True)

else:

st.write("Please enter a review to analyze.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

### **How does it work?**

1. **Downloading Stopwords**:
   * The app needs to know common words like "the", "is", "in" to ignore them when analyzing text. These common words are called **stopwords**. It uses the **nltk** library to download these words.
2. **Loading the Model**:
   * The app uses a **model** (which is like a brain) to determine if the tweet is positive or negative. This model has already been trained on lots of data to understand sentiment.
   * The model is saved in a file called model.pkl, and the app **loads** this model when it starts.
3. **Getting Sentiment**:
   * When you give the app a tweet, it first **cleans** the text by removing unwanted characters (like numbers or symbols).
   * Then, it checks if the tweet is **positive** or **negative** using the trained model. The model looks at the words in the tweet and decides if it's a happy (positive) or sad (negative) tweet.
4. **Scraping a Tweet from a Link**:
   * If you provide a **link** to a tweet, the app will use **Selenium** (a tool that controls a web browser) to open the tweet on the website **X.com** (formerly Twitter).
   * The app looks for the tweet’s text using the browser, and once it finds it, it returns the tweet’s content to analyze the sentiment.
5. **Creating the Result**:
   * The app then shows the sentiment (positive or negative) with a **colored card**. The card is green if the tweet is positive, and red if it's negative.
6. **Streamlit's Magic**:
   * **Streamlit** helps make this app look good by creating buttons, input boxes, and displaying the results. It also lets the user interact with the app in a simple way.

### **Main Parts:**

* **stopwords**: Words that don’t matter for analysis (like "the", "and").
* **model**: The brain that knows whether a tweet is good or bad.
* **Selenium**: The tool that opens the webpage and grabs the tweet.
* **Streamlit**: The tool that lets you interact with the app.

### **Steps for the User:**

1. Choose if you want to:
   * **Analyze your own text** by typing it in.
   * **Fetch a tweet** by entering a tweet link.
2. If you enter a tweet link, it fetches the tweet's text and analyzes if it's positive or negative.
3. The result is displayed with a **green card** for positive or **red card** for negative sentiment.

### **model.pkl: The Pre-trained Machine Learning Model**

This file contains a **pre-trained machine learning model** that predicts the sentiment of text (positive or negative). It has already been trained on a dataset of text and sentiment labels, so it can make predictions without retraining.

#### **Why is it needed?**

* The model.pkl file acts as the brain of the program. It uses the patterns learned during training to predict the sentiment of new input text.
* When you input text or fetch a tweet, this model analyzes it and returns the predicted sentiment:
  + **Positive sentiment** → If the model predicts a value like 1.
  + **Negative sentiment** → If the model predicts a value like 0.

#### **How is it created?**

The file is created by:

1. Training a machine learning model (like Logistic Regression, SVM, or Random Forest) on labeled data (e.g., tweets with positive or negative sentiments).
2. Saving the trained model to a .pkl file using Python's pickle module:

### **vectorizer.pkl: The Text Vectorizer**

This file contains a **pre-trained vectorizer**, typically created using TfidfVectorizer from **scikit-learn**. The vectorizer converts raw text data (like a tweet) into numerical data that the machine learning model can understand.

#### **Why is it needed?**

* Machine learning models only understand numbers, not raw text. The vectorizer transforms text into a numerical format (e.g., a sparse matrix of word frequencies or importance scores).
* This vectorization ensures that the input text is in the same format as the data used during training, making the predictions consistent.

Vectorizer.pkl converts the plain text into machine code and feeds it into model.pkl. The model.pkl is a trained machine learning model for sentiment analysis which analyze the machine code from vectorizer and returns the output.

### **Example:**

* You type in a tweet link: https://x.com/elonmusk/status/1704353292131246087.
* The app goes to that link, gets the tweet, and says, “This tweet is **positive**” or “This tweet is **negative**”.

### **Manual for Twitter and Customer Sentiment Analysis with Streamlit and Selenium**

#### **Objective:**

The goal of this project is to build a simple web application that analyzes the sentiment of a tweet (whether it's positive or negative) using machine learning. The app will let users enter text or provide a link to a tweet to fetch its sentiment. This will help you learn how to use Python libraries such as **Streamlit**, **Selenium**, **scikit-learn**, and **pickle** to create an interactive sentiment analysis tool.

#### **Materials Needed:**

1. **A computer with Python installed**.
2. **Google Chrome browser** (for Selenium to work).
3. **Text editor** (like Visual Studio Code or Jupyter Notebook).
4. **Internet connection** (for downloading necessary libraries).

#### **Pre-Lab Setup:**

Before starting the lab, you need to set up your environment.

**Step 1: Install Required Python Packages**

You will need several Python libraries. You can install them using the pip command. Open a command prompt or terminal and run:

***pip install streamlit selenium scikit-learn nltk pickle-mixin***

These packages are:

* **Streamlit**: To create the web app.
* **Selenium**: To fetch tweet data from the internet.
* **scikit-learn**: For machine learning tools like text vectorization and model loading.
* **nltk**: For processing text (like removing common words).
* **pickle-mixin**: For loading and saving machine learning models.

#### **Lab Instructions:**

1. **Introduction to the Program:**The web application will allow users to:
   * Input text and analyze its sentiment.
   * Enter a tweet URL, and the app will fetch the tweet’s text and analyze its sentiment.
2. **Program Code Overview:**The code consists of several parts:
   * **Loading Libraries**: The program uses libraries like streamlit, pickle, nltk, and selenium.
   * **Machine Learning Model**: A pre-trained machine learning model will predict whether the text or tweet is positive or negative.
   * **Text Preprocessing**: The program cleans the text to remove unnecessary words.
   * **Fetching Tweets**: Selenium is used to fetch tweets from **X.com** (formerly Twitter) by using the tweet's URL.
   * **Displaying Results**: The app shows results with colored cards.

#### **Step-by-Step Guide:**

**Step 1: Loading Required Libraries**

At the start of the code, we import all the necessary Python libraries. The important libraries are:

* **streamlit**: To create the interactive web page.
* **selenium**: To automatically browse and fetch tweets.
* **scikit-learn**: To handle machine learning tasks like text analysis.
* **nltk**: For text processing tasks like removing unnecessary words.

**Step 2: Loading the Machine Learning Model**

The model used for sentiment analysis is saved in a file called model.pkl. The vectorizer.pkl file is used to convert the tweet’s text into numbers that the model can understand.

# Load model and vectorizer

@st.cache\_resource

def load\_model\_and\_vectorizer():

with open('model.pkl', 'rb') as model\_file:

model = pickle.load(model\_file)

with open('vectorizer.pkl', 'rb') as vectorizer\_file:

vectorizer = pickle.load(vectorizer\_file)

return model, vectorizer

**Step 3: Text Preprocessing**

The program cleans the text before analyzing it:

* It removes special characters (like @, #, and URLs).
* It converts all text to lowercase.
* It removes common words that don’t help in sentiment analysis (like "the", "and").

# Preprocess text

def predict\_sentiment(text, model, vectorizer, stop\_words):

text = re.sub('[^a-zA-Z]', ' ', text) # Remove non-alphabetical characters

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

text = text.split() # Split the text into words

text = [word for word in text if word not in stop\_words] # Remove stopwords

text = ' '.join(text) # Rejoin words into a cleaned-up text

text = [text] # Prepare text for the model

text = vectorizer.transform(text) # Convert the text to numbers

sentiment = model.predict(text) # Predict sentiment using the model

return "Negative" if sentiment == 0 else "Positive"

**Step 4: Fetching Tweets Using Selenium**

When a user provides a tweet URL, the program opens that link using **Selenium** to extract the tweet's text. This is done by simulating a browser opening and reading the tweet's content.

# Function to scrape a tweet from a tweet URL using Selenium

def scrape\_tweet(tweet\_url):

url = f"{tweet\_url}" # Use the provided tweet URL

options = Options()

options.headless = True # Run the browser in the background (no window)

driver = webdriver.Chrome(options=options)

try:

driver.get(url) # Go to the tweet's page

time.sleep(3) # Wait for the page to load

tweet\_div = driver.find\_element(By.XPATH, "//div[@data-testid='tweetText']") # Find tweet text

tweet\_text = tweet\_div.text

return tweet\_text if tweet\_text else None

except Exception as e:

print(f"Error: {e}")

return None

finally:

driver.quit() # Close the browser

**Step 5: Displaying the Result**

After the sentiment of the tweet is determined, the result is shown in a colored card. If the sentiment is **positive**, the card is green; if it’s **negative**, the card is red.

# Function to create a colored card for displaying the sentiment result

def create\_card(tweet\_text, sentiment):

color = "green" if sentiment == "Positive" else "red"

card\_html = f"""

<div style="background-color: {color}; padding: 10px; border-radius: 5px; margin: 10px 0;">

<h5 style="color: white;">{sentiment} Sentiment</h5>

<p style="color: white;">{tweet\_text}</p>

</div>

"""

return card\_html

**Step 6: Streamlit Web Interface**

Streamlit helps create an easy interface where you can input either text or a tweet URL to analyze the sentiment.

def main():

st.title("X.com Sentiment Analysis")

stop\_words = load\_stopwords()

model, vectorizer = load\_model\_and\_vectorizer()

# User input: text or tweet URL

st.subheader("Enter a Tweet URL")

tweet\_url = st.text\_input("Paste the tweet link here")

if st.button("Fetch Tweet"):

tweet\_text = scrape\_tweet(tweet\_url)

if tweet\_text:

sentiment = predict\_sentiment(tweet\_text, model, vectorizer, stop\_words)

card\_html = create\_card(tweet\_text, sentiment)

st.markdown(card\_html, unsafe\_allow\_html=True)

else:

st.write("Could not fetch tweet or an error occurred.")





