



**TWEETVIBES**

**(Twitter Sentiment Analysis)**

A Project report submitted in fulfilment of the requirement completion of Final Project for the Diploma in Post Graduate in Bigdata Analysis.

**Course Co-ordinator:**

Vineeta Madam

**Project Guide:**

Malkeet Singh Rathor

**Project By:**

**Dhanush Shetty** (Project Lead) (230340325008)

**Dhiraj Patil** (230340325009)

**Pratik Landge** (230340325038)

**Gourav Priyadarshan** (230340325013)

**Santosh Prajapati** (230340325052)

**Gopalkrishna Umak** (230340325012)

**TweetVibes [Twitter Sentiment Analysis]**

**using NLP (NATURAL LANGUAGE PROCESSING)**

**Abstract:**

The aim of this project was to develop a sentiment analysis model using Natural Language Processing (NLP) techniques to analyse and classify the sentiment of tweets from the Twitter platform. Sentiment analysis is a valuable tool for understanding public opinion, customer feedback, and trends.

In the age of social media, extracting valuable insights from the massive volume of data generated on platforms like Twitter has become increasingly important. "TweetVibes" is a project that leverages Natural Language Processing (NLP) techniques for sentiment analysis on Twitter data. The goal of this project is to develop an intelligent system that can automatically determine the sentiment behind tweets, categorizing them as positive, negative.

The exponential growth of user-generated content on Twitter presents both opportunities and challenges. While tweets can reflect diverse opinions and emotions, manually processing and analysing this content is unfeasible. TweetVibes aims to address this challenge by employing advanced NLP methodologies to automate sentiment analysis and extract meaningful insights from Twitter data.

This project addresses the problem of sentiment analysis in twitter; that is classifying tweets according to the sentiment expressed in them: positive, negative. Twitter is an online micro-blogging and social-networking platform which allows users to write short status updates of maximum length 140 characters. It is a rapidly expanding service with over 200 million registered, out of which 100 million are active users and half of them log on twitter on a daily basis, generating nearly 250 million tweets per day. Due to this large amount of usage we hope to achieve a reflection of public sentiment by analysing the sentiments expressed in the tweets.

Analysing the public sentiment is important for many applications such as firms trying to find out the response of their products in the market, predicting political elections and predicting socioeconomic phenomena like stock exchange.

The aim of this project is to develop a functional classifier for accurate and automatic sentiment classification of an unknown tweet stream.

In this project, we utilized a dataset of tweets, pre-processed the text, built and trained a sentiment classification model, and evaluated its performance.

**Introduction:**

In today's digital era, social media platforms like Twitter have transformed the way we communicate, express opinions, and share information. Twitter serves as a global hub where people from diverse backgrounds engage in conversations, express emotions, and contribute to trending topics. However, the sheer volume of user-generated content on Twitter poses challenges in understanding and interpreting the sentiments and emotions underlying these interactions. This is where "TweetVibes" comes into play.

Social media platforms like Twitter have become important sources of information and public sentiment. Sentiment analysis involves determining whether a piece of text expresses a positive, negative.

This project focused on using NLP techniques to automatically classify the sentiment of tweets as positive, negative.

We have chosen to work with twitter since we feel it is a better approximation of public sentiment as opposed to conventional internet articles and web blogs.

The reason is that the amount of relevant data is much larger for twitter, as compared to traditional blogging sites. Moreover, the response on twitter is more prompt and also more general (since the number of users who tweet is substantially more than those who write web blogs on a daily basis). Sentiment analysis of public is highly critical in macro-scale socioeconomic phenomena like predicting the stock market rate of a particular firm.

This could be done by analysing overall public sentiment towards that firm with respect to time and using economics tools for finding the correlation between public sentiment and the firm’s stock market value. Firms can also estimate how well their product is responding in the market, which areas of the market is it having a favourable response and in which a negative response. If firms can get this information they can analyse the reasons behind geographically differentiated response, and so they can market their product in a more optimized manner by looking for appropriate solutions like creating suitable market segments.

Predicting the results of popular political elections and polls is also an emerging application to sentiment analysis.

**Problem Statement:**

"Develop a sentiment analysis model using Natural Language Processing (NLP) techniques to analyse tweets from social media platform. The model should classify each tweet as positive, negative based on the sentiment expressed in the text. Evaluate and compare the performance of different machine learning and deep learning algorithms for accurate sentiment classification."

**Background:**

With the growing influence of social media, understanding public sentiment has become crucial for businesses, brands, and policymakers. Twitter, being a platform with millions of daily tweets, serves as a valuable source of real-time public opinion. Sentiment analysis, a subfield of Natural Language Processing (NLP), aims to automatically categorize text into positive, negative, or neutral sentiments. This project seeks to harness NLP techniques to build an accurate sentiment analysis model specifically tailored for tweets.

The project aims to develop an effective sentiment analysis model that can analyse and categorize the sentiment of tweets as positive, negative, or neutral. The model's predictions will help individuals and organizations gauge public sentiment, track trends, and make informed decisions based on the collective voice of Twitter users.

In this project, we try to implement a Twitter sentiment analysis model that helps to overcome the challenges of identifying the sentiments of the tweets. The necessary details regarding the dataset are: The dataset provided is the Sentiment140 Dataset which consists of 1,600,000 tweets that have been extracted using the Twitter API.

**LITERATURE REVIEW**

Limitations of Prior Art Sentiment analysis of in the domain of micro-blogging is a relatively new research topic so there is still a lot of room for further research in this area. Decent amount of related prior work has been done on sentiment analysis of user reviews, documents, web blogs/articles and general phrase level sentiment analysis.

These differ from twitter mainly because of the limit of 140 characters per tweet which forces the user to express opinion compressed in very short text. The best results reached in sentiment classification use supervised learning techniques such as Naive Bayes and Support Vector Machines, but the manual labelling required for the supervised approach is very expensive.

Some work has been done on unsupervised and semi-supervised approaches, and there is a lot of room of improvement. Various researchers testing new features and classification techniques often just compare their results to base-line performance. There is a need of proper and formal comparisons between these results arrived through different features and classification techniques in order to select the best features and most efficient classification techniques for particular applications

**Library Used:**

**Pyspark**

Pyspark.sql is a module in PySpark, which is the Python library for Apache Spark, a powerful distributed data processing framework.

pyspark.sql in Short:

pyspark.sql provides a high-level API for working with structured and semi-structured data in Spark, making it easier to perform data manipulation, querying, and analysis using SQL-like syntax.

In summary, pyspark.sql simplifies data manipulation, querying, and analysis in Apache Spark by introducing Data Frames and SQL-like operations. This module enhances code readability, optimizes query execution, and provides an efficient way to work with structured and semi-structured data at scale.

**These are the steps why we use pyspark.sql: -**

**Structured Data Processing**

**SQL-Like Operations**

**Optimization**

**Code Readability**

**Integration**

**Broad Data Format Support**

**Distributed Processing**

**Data Source Connectivity**

**Data Exploration**

**Community and SupportRegex**

**Sklearn:**

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

It provides essential tools and algorithms for various tasks, enabling practitioners to quickly build, evaluate, and deploy machine learning models.

This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

**Important features of scikit-learn:**

● Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.

● Accessible to everybody and reusable in various contexts.

● Built on the top of NumPy, SciPy, and matplotlib.

● Open source, commercially usable – BSD license

**Pill:** PIL (Python Imaging Library)

PIL (Python Imaging Library), now known as "Pillow," is a widely used open-source library for working with images in Python. It provides various image processing capabilities, making it an essential tool for tasks such as opening, manipulating, and saving image files.

**Here's a quick overview:**

* + - * + **Image Processing:**
        + **Filtering and Effects:**
        + **Drawing and Text:**
        + **Image Formats:**
        + **Image Data Manipulation:**
        + **Image Transformation:**
        + **Basic Image Analysis:**
        + **Community and Documentation:**

Pillow is a popular and versatile library for image manipulation in Python, suitable for tasks ranging from simple image editing to more complex image processing workflows.

**Matplotlib:**

Matplotlib is a widely used open-source Python library for creating static, interactive, and animated visualizations. It provides a flexible and comprehensive set of tools for generating various types of plots and charts.

Matplotlib is a powerful library for creating a wide variety of static and interactive visualizations in Python. Its flexibility, customization options, and compatibility with other libraries make it an essential tool for data visualization tasks.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

**● Create publication quality plots.**

**● Make interactive figures that can zoom, pan, update.**

**● Customize visual style and layout.**

**● Export to many file formats.**

**● Embed in JupyterLab and Graphical User Interfaces.**

**● Use a rich array of third-party packages built on Matplotlib**

**Wordcloud:**

Wordcloud is a Python library used for creating visual representations of text data, where the size of each word indicates its frequency or importance within the text.

It's a popular tool for visually summarizing the most common words in a document, article, or piece of text.

Wordcloud are particularly useful for gaining quick insights from text data and effectively conveying the most frequent terms within a document. They're often used in data analysis, text mining, and data visualization tasks.

**StremLit:**

Streamlit is an open-source Python library that allows you to create interactive web applications for data visualization and analysis with minimal code. It's designed to turn data scripts into shareable web apps quickly.

Here's a overview: -

Streamlit is well-suited for data scientists, analysts, and developers who want to quickly share their data insights with others through interactive web apps, without the need for extensive web development knowledge.

**Pandas:**

Pandas is an open-source Python library widely used for data manipulation, analysis, and preparation. It provides versatile data structures and tools designed to make working with structured data intuitive and efficient.

Pandas is an essential tool for data manipulation and analysis in Python. It's commonly used in data cleaning, preparation, exploration, and transformation tasks before using other libraries for modelling or visualization.

**Selenium:**

Selenium is an open-source automated testing framework primarily used for web applications. It allows you to automate interactions with web browsers, simulate user actions, and perform various testing tasks.

While Selenium is primarily known for its testing capabilities, it's also used for web scraping and other tasks that involve automated interactions with web pages. It's a versatile tool for ensuring the quality and functionality of web applications.

**DATA EXTRACTION WITH SCRAPER**

A web scraper is a software tool or program used to extract data from websites. It automates the process of gathering information from web pages by simulating human browsing behaviour and retrieving specific content. Web scraping is commonly used to extract data for various purposes, such as research, data analysis, content aggregation, and more.

Here are the key points about web scrapers:

Data Extraction:

Web scrapers are designed to navigate websites, access HTML content, and extract specific data elements, such as text, images, links, tables, and more.

Automation:

Instead of manually copying and pasting information from websites, web scrapers automate the data extraction process, which can save a significant amount of time and effort.

Techniques:

Web scraping can be done using various techniques, such as parsing HTML and XML documents, using regular expressions, or utilizing specialized libraries or tools designed for web scraping.

HTTP Requests:

Web scrapers make HTTP requests to the targeted websites, retrieve the HTML content of web pages, and then parse the content to extract the desired information.

Ethical and Legal Considerations:

While web scraping can be a useful tool, it's important to use it ethically and legally. Many websites have terms of service that outline whether web scraping is allowed or prohibited. Additionally, scraping massive amounts of data from a website can strain the server and potentially violate the site's terms of use.

Robots.txt:

Some websites provide a "robots.txt" file, which outlines which parts of the website can be crawled and scraped by search engines and other automated tools. It's important to respect these directives when conducting web scraping.

APIs vs. Web Scraping:

In some cases, websites provide Application Programming Interfaces (APIs) that allow developers to access and retrieve data in a structured and authorized manner. APIs are preferred when available, as they provide a more controlled and efficient way of accessing data compared to scraping.

Tools and Libraries: There are various programming libraries and tools available in different programming languages to facilitate web scraping, such as Python's Beautiful Soup, Scrapy, or using browser automation tools like Selenium.

Web scraping can be a valuable technique for data collection and analysis, but it's essential to be mindful of ethical considerations and the terms of use of the websites you're scraping. Always ensure that your scraping activities are aligned with legal and ethical guidelines.

**MODEL CREATION:**

In machine learning, a model is a mathematical representation or algorithm that learns patterns and relationships from data. Models are trained on datasets to make predictions or decisions without being explicitly programmed for every possible scenario.

There are various types of models used in machine learning, each designed for specific tasks and data types.

**Logistic Regression:**

Logistic Regression is a statistical and machine learning algorithm used for binary classification tasks. Despite its name, it's actually a type of regression algorithm used for predicting the probability of an instance belonging to a particular class. It's widely used due to its simplicity and interpretability, especially when the relationship between the features and the target variable is relatively linear.

**Applications:**

Logistic Regression is used in a wide range of fields, including medicine (predicting disease outcomes), finance (credit risk assessment), marketing (customer churn prediction), and more.

Used for binary classification tasks, it predicts the probability of an instance belonging to a certain class.

**Support Vector Machines (SVM):**

Support Vector Machines (SVM) are a powerful and versatile machine learning algorithm primarily used for classification and regression tasks. They work by finding the optimal hyperplane that best separates data points of different classes in a high-dimensional space. SVMs are particularly effective when dealing with complex decision boundaries and are well-suited for tasks involving both linear and non-linear relationships.

**Applications:**

SVMs are used in various domains, including image classification, text classification, bioinformatics, finance, and more.

Used for both classification and regression tasks, SVM finds a hyperplane that best separates data points of different classes.

**Naive Bayes:**

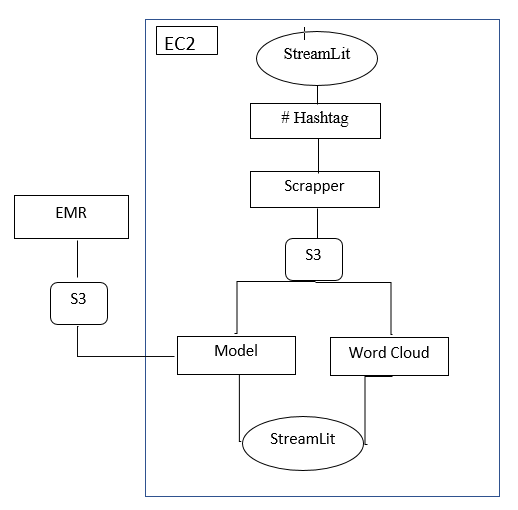
Naive Bayes is a probabilistic machine learning algorithm that's commonly used for classification tasks. It's based on the principles of Bayes' theorem and the assumption of conditional independence between features. Despite its simple assumptions, Naive Bayes often performs surprisingly well in various text classification and sentiment analysis tasks.

**Applications:**

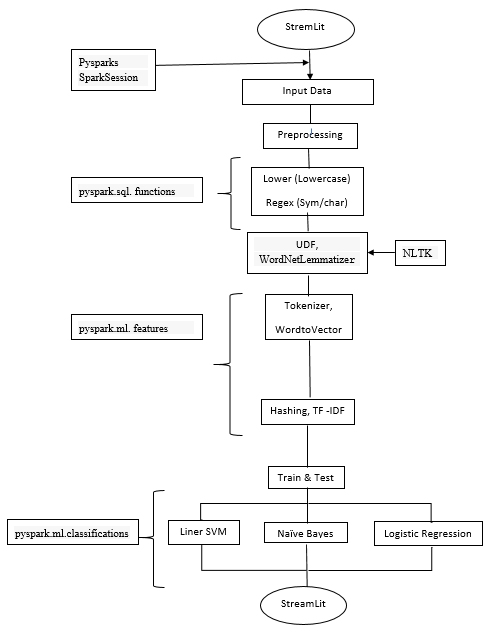
Naive Bayes is commonly used in text classification (spam detection, sentiment analysis), document categorization, recommendation systems, and more.

Probabilistic model based on Bayes' theorem; used for classification tasks.

**COMPLETE PROJECT WORKFLOW**



**Program Workflow:**

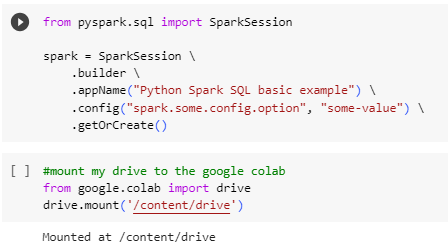


**Data Collection:**

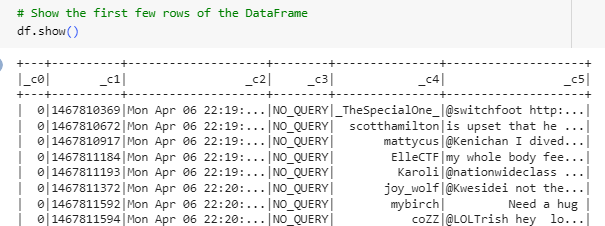
Data collection refers to the process of gathering raw information or data from various sources. In the context of a project like Twitter sentiment analysis using NLP, data collection involves obtaining a set of tweets that will be used to train and test the sentiment analysis model.

This data is essential for training the model to recognize patterns and relationships between text and sentiment labels, enabling it to make accurate predictions on new, unseen tweets. Data collection often includes retrieving, gathering, and sometimes cleaning or pre-processing the data to ensure its quality and suitability for analysis.

We obtained a dataset of tweets from Twitter's API. The dataset contained a mix of user-generated tweets that were labelled as positive, negative, based on the sentiment expressed in the text.



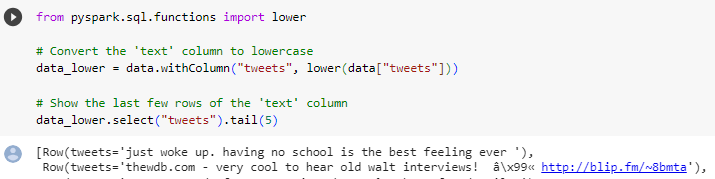




**Data Pre-processing (Initial Stage):**

**Data pre-processing is a crucial step in NLP tasks. We performed the following pre-processing steps:**

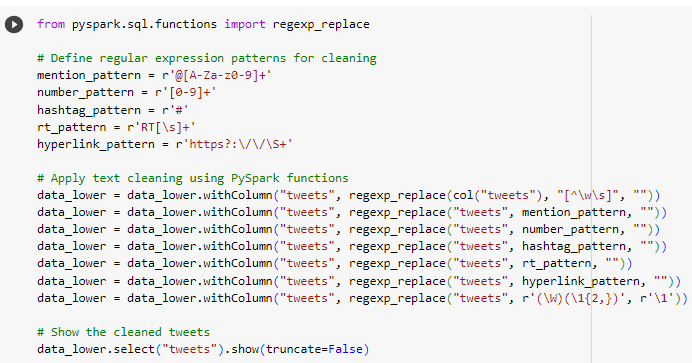
* **Lowercasing:** Converting all text to lowercase to ensure uniformity..



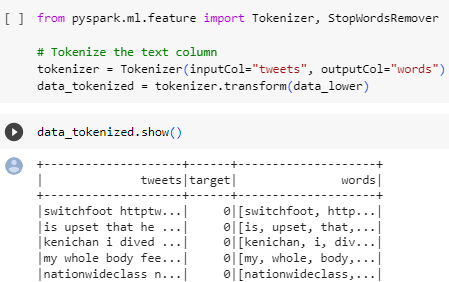
* **Text Cleaning:**

Removal of special characters, URLs, and mentions.

(Remove mentions, Remove numbers, Remove hashtags, Remove retweet tags, Remove hyperlinks, Remove non-alphabetic characters (except spaces),Replace multiple spaces with a single space**)**

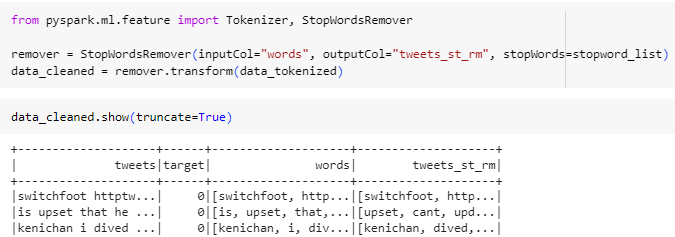


* **Tokenization:** Breaking down tweets into individual words or tokens.

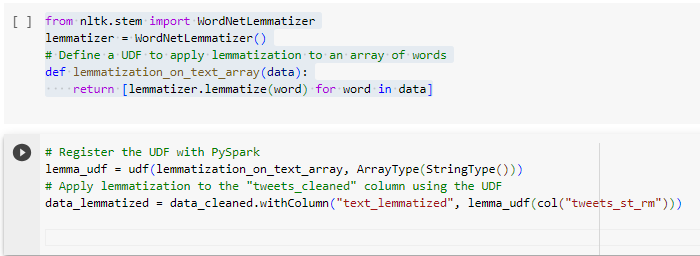


* **Stop word Removal:**

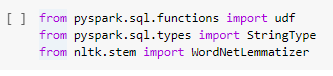
Eliminating common words that don't contribute significantly to sentiment analysis.

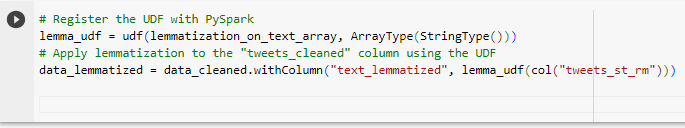


* **Lemmatization:** Reducing words to their Dictionary meaning .



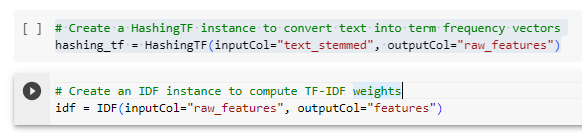
* **UDF:** To execute specific tasks within a programming language.





* **Hashing TF-ITF:** Converts text data into a fixed-size numeric vector.

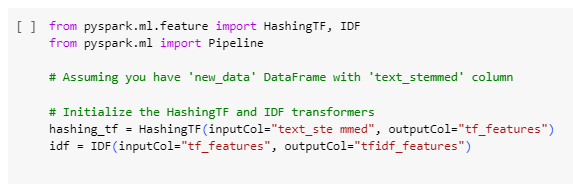




**Data Pre-processing (Later Stage):**

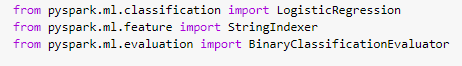
Feature extraction is the process of selecting and transforming relevant information from raw data to create a more compact and meaningful representation, which is easier for machine learning algorithms to understand and process.

For text data, converting it into a numerical representation is essential for machine learning models. We used techniques like TF-IDF (Term Frequency-Inverse Document Frequency) and word embeddings such as Word2Vec or Glove to represent words as vectors.



**Model Selection:**

Choosing the most appropriate machine learning algorithm or model for a specific task.We experimented with various machine learning and deep learning models for sentiment classification, including:



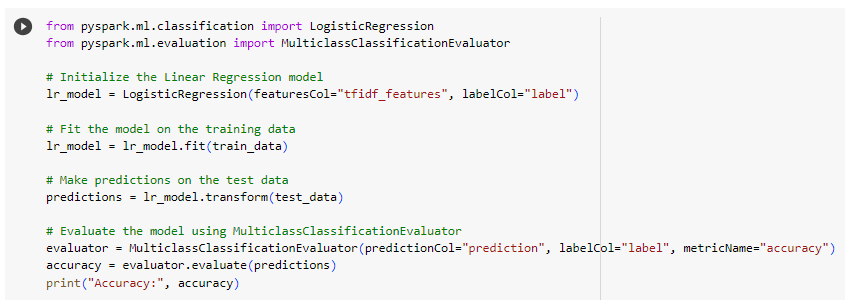
* **Naive Bayes**



* **Support Vector Machines (SVM)**



* **Logistic Regression**



**Model Training:**

We split the pre-processed data into training and testing sets. We trained the selected models on the training set and fine-tuned hyperparameters to optimize performance. For Machine learning models, we used frameworks like Genism.

**Train model with (1.6 Million dataset).**



**Evaluation:**

The performance of the models was evaluated using metrics such as accuracy. Measure the proportion of correct predictions made by a classification model over the total number of predictions

**Results:**

The results demonstrated that the Machine learning models, especially transformer-based models like which gives the prediction on given input.

* **Naive Bayes [Accuracy Score: 74.0%]**
* **Support Vector Machines (SVM) [Accuracy Score: 76.0%]**
* **Logistic Regression [Accuracy Score: 75.0%]**

**Full Program:-**

**!pip install pyspark**

**from pyspark.sql import SparkSession**

**spark = SparkSession.builder.appName("Python Spark SQL basic example").config("spark.some.config.option","some-value").getOrCreate()**

**# mount googel drive**

**from google.colab import drive**

**drive.mount('/content/drive')**

**df = spark.read.csv("/content/drive/MyDrive/dataset Tweet.csv")**

**df = spark.read.option("header","false").option("encoding","ISO-8859-1").option("delimeter",",").csv("/content/drive/MyDrive/dataset Tweet.csv")**

**df.show()**

**df = df.withColumnRenamed("\_c0", "target").withColumnRenamed("\_c1","id").withColumnRenamed("\_c2","date").withColumnRenamed("\_c3","flag").withColumnRenamed("\_c4","user").withColumnRenamed("\_c5","tweet")**

**#renamed column names**

**# Get the number of rows in the DataFrame**

**num\_rows = df.count()**

**print("Number of rows in the DataFrame:", num\_rows)**

**#get number of columns (length of column names list)**

**num\_columns = len(df.columns)**

**print("Shape of the DataFrame: ({}, {})".format(num\_rows, num\_columns))**

**# to count unique values for specific column .**

**column\_name = "target"**

**# to calculate number of unique values in specific column**

**num\_uniq\_values = df.select (column\_name).distinct().count()**

**print("Number of unique values in column '{}': {}".format(column\_name, num\_uniq\_values))**

**# to calculate count of each unique value in specified column**

**value\_counts = df.groupby(column\_name).count()**

**value\_counts.show()**

**# to select specific columns**

**data = df.select("tweet","target")**

**from pyspark.sql.functions import lower**

**# now converting "tweet" column in lowercase**

**data\_lower = data.withColumn("tweet",lower(data["tweet"]))**

**data\_lower.select('tweet').tail(5) # to find lower 5 tweet**

**from pyspark.sql.functions import regexp\_replace,col**

**# Define regular expression patterns for cleaning**

**mention\_pattern = r'@[A-Za-z0-9]+'**

**number\_pattern = r'[0-9]+'**

**hashtag\_pattern = r'#'**

**rt\_pattern = r'RT[\s]+'**

**hyperlink\_pattern = r'https?:\/\/\S+'**

**# Apply text cleaning using PySpark functions**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace(col("tweet"), "[^\w\s]", ""))**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace("tweet", mention\_pattern, ""))**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace("tweet", number\_pattern, ""))**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace("tweet", hashtag\_pattern, ""))**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace("tweet", rt\_pattern, ""))**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace("tweet", hyperlink\_pattern, ""))**

**data\_lower = data\_lower.withColumn("tweet", regexp\_replace("tweet", r'(\W)(\1{2,})', r'\1'))**

**# Show the cleaned tweets**

**data\_lower.select("tweet").show(truncate=False)**

**data\_lower.show()**

**!pip install nltk**

**from pyspark.sql.functions import udf**

**from pyspark.sql.types import StringType**

**from nltk.stem import WordNetLemmatizer**

**import nltk**

**# Download NLTK resources**

**nltk.download('wordnet')**

**from pyspark.ml.feature import Tokenizer, StopWordsRemover**

**# Tokenize the text column**

**tokenizer = Tokenizer(inputCol="tweet", outputCol="tweet\_tok")**

**data\_tokenized = tokenizer.transform(data\_lower)**

**data\_tokenized.show()**

**from pyspark.ml.feature import StopWordsRemover**

**# Remove stop words from the tokenized words**

**stopwords\_remover = StopWordsRemover(inputCol="tweet\_tok", outputCol="tweet\_st")**

**data\_filtered = stopwords\_remover.transform(data\_tokenized)**

**data\_filtered.show()**

**from pyspark.sql.functions import udf**

**from pyspark.sql.types import ArrayType, StringType**

**from nltk.stem import WordNetLemmatizer**

**import nltk**

**# Initialize the lemmatizer**

**lemmatizer = WordNetLemmatizer()**

**# Define a UDF for lemmatization**

**def lemmatize\_words(words):**

**lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]**

**return lemmatized\_words**

**# Create a UDF for lemmatization**

**lemmatize\_udf = udf(lemmatize\_words, ArrayType(StringType()))**

**# Apply lemmatization using the UDF**

**data\_lemmatized = data\_filtered.withColumn("tweet\_lem", lemmatize\_udf("tweet\_st"))**

**data\_lemmatized.show()**

**# Specify the list of columns to delete**

**columns\_to\_delete = ["tweet", "tweet\_tok","tweet\_st"]**

**# Drop the specified columns from the DataFrame**

**new\_data = data\_lemmatized.drop(\*columns\_to\_delete)**

**new\_data.show()**

**# Show the resulting DataFrame without the deleted columns**

**new\_data.show(truncate=False)**

**from pyspark.ml.feature import HashingTF, IDF**

**from pyspark.ml import Pipeline**

**# Initialize the HashingTF and IDF transformers**

**hashing\_tf = HashingTF(inputCol="tweet\_lem", outputCol="tf\_features")**

**idf = IDF(inputCol="tf\_features", outputCol="tfidf\_features")**

**# Create a pipeline to chain the HashingTF and IDF stages**

**pipeline = Pipeline(stages=[hashing\_tf, idf])**

**# Fit and transform the data using the pipeline**

**model = pipeline.fit(new\_data)**

**transformed\_data = model.transform(new\_data)**

**# Show the resulting DataFrame with TF-IDF features**

**transformed\_data.select("target", "tfidf\_features").show(truncate=False)**

**from pyspark.sql.functions import expr**

**from pyspark.ml.feature import StringIndexer**

**# Assuming your 'target' column contains string labels**

**indexer = StringIndexer(inputCol="target", outputCol="label")**

**indexed\_data = indexer.fit(transformed\_data).transform(transformed\_data)**

**indexed\_data.show()**

**# Specify the list of columns you want to delete**

**columns\_to\_delete = ["target","tweet\_lem","tf\_features"]**

**indexed\_data = indexed\_data.drop(\*columns\_to\_delete)**

**print("Total number of rows in indexed\_data:", indexed\_data.count())**

**indexed\_data.show()**

**indexed\_data.printSchema()**

**# Split the data into training and testing sets**

**train\_data, test\_data = indexed\_data.randomSplit([0.8, 0.2], seed=123)**

**# Assuming your DataFrame is named 'data'**

**distinct\_values = train\_data.select("label").distinct()**

**# Show distinct values**

**distinct\_values.show()**

**from pyspark.ml.classification import LogisticRegression**

**from pyspark.ml.evaluation import MulticlassClassificationEvaluator**

**# Initialize the Linear Regression model**

**lr\_model = LogisticRegression(featuresCol="tfidf\_features", labelCol="label")**

**# Fit the model on the training data**

**lr\_model = lr\_model.fit(train\_data)**

**# Make predictions on the test data**

**predictions = lr\_model.transform(test\_data)**

**# Evaluate the model using MulticlassClassificationEvaluator**

**evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label", metricName="accuracy")**

**accuracy = evaluator.evaluate(predictions)**

**print("Accuracy:", accuracy)**

**from pyspark.ml.classification import NaiveBayes**

**# Initialize the NaiveBayes model**

**nb = NaiveBayes(featuresCol="tfidf\_features", labelCol="label")**

**# Train the model**

**nb\_model = nb.fit(train\_data)**

**# Make predictions on the test data**

**predictions = nb\_model.transform(test\_data)**

**# Evaluate the model using MulticlassClassificationEvaluator**

**evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label", metricName="accuracy")**

**accuracy = evaluator.evaluate(predictions)**

**print("Accuracy:", accuracy)**

**from pyspark.ml.classification import LinearSVC**

**# Initialize the LinearSVC (Support Vector Machine) model**

**svm = LinearSVC(featuresCol="tfidf\_features", labelCol="label")**

**# Train the model**

**svm\_model = svm.fit(train\_data)**

**# Make predictions on the test data**

**predictions = svm\_model.transform(test\_data)**

**# Evaluate the model using MulticlassClassificationEvaluator**

**evaluator = MulticlassClassificationEvaluator(predictionCol="prediction", labelCol="label", metricName="accuracy")**

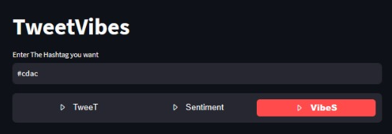
**accuracy = evaluator.evaluate(predictions)**

**print("Accuracy:", accuracy)**

**APP CREATION (STREAMLIT GUI)**

**What is Streamlit?**

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers. Data scientists or machine learning engineers are not web developers and they're not interested in spending weeks learning to use these frameworks to build web apps. Instead, they want a tool that is easier to learn and to use, as long as it can display data and collect needed parameters for modelling. Streamlit allows you to create a stunning-looking application with only a few lines of code.



**Why should data scientists use Streamlit?**

The best thing about Streamlit is that you don't even need to know the basics of web development to get started or to create your first web application. So if you're somebody who's into data science and you want to deploy your models easily, quickly, and with only a few lines of code, Streamlit is a good fit. One of the important aspects of making an application successful is to deliver it with an effective and intuitive user interface. Many of the modern data heavy apps face the challenge of building an effective user interface quickly, without taking complicated steps.

Streamlit is a promising open-source Python library, which enables developers to build attractive user interfaces in no time.

Streamlit is the easiest way especially for people with no front-end knowledge to put their code into a web application:

● No front-end (html, js, css) experience or knowledge is required.

● You don't need to spend days or months to create a web app, you can create a really beautiful machine learning or data science app in only a few hours or even minutes.

● It is compatible with the majority of Python libraries (e.g. pandas, matplotlib, seaborn, plotly, Keras, PyTorch, SymPy(latex)).

● Less code is needed to create amazing web apps.

● Data caching simplifies and speeds up computation pipelines.

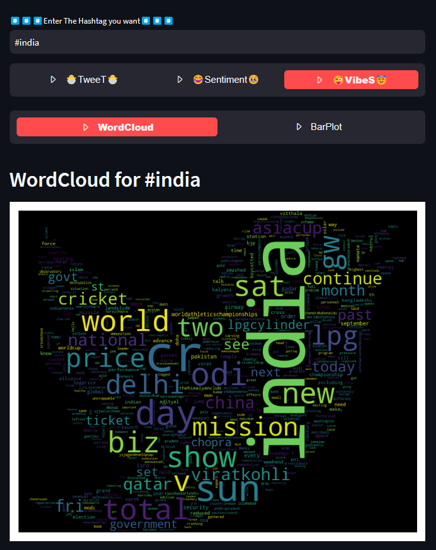
Type this command to install Streamlit

* ➢ pip install streamlit
* ➢ Importing Streamlit Library
* ➢ import streamlit as st

**Conclusion:**

In this project, we successfully developed a sentiment analysis model using NLP techniques to analyse and classify the sentiment of tweets.

The model demonstrated promising accuracy and performance, which indicates its potential application in understanding public sentiment, brand perception, and trend analysis on social media platforms like Twitter.



**Future Enhancements:**

* **Emotion Detection**: Extend the sentiment analysis to detect specific emotions expressed in the text.
* **Domain Adaptation:** Fine-tune the model to be more accurate for specific domains or industries.
* **Real-time Analysis:** Implement the model to analyse streaming tweets and provide up-to-date sentiment insights.

**References:**

By implementing this project, we gained insights into the process of sentiment analysis using NLP techniques, the challenges involved, and the potential applications of such models in real-world scenarios.