# Spark Streaming Project: Sentiment Analysis and Clustering

## Overview

This project involves processing API data as a stream, performing sentiment analysis on the textual data, and clustering the messages based on sentiment and location. The project also includes creating a dynamic dashboard to visualize the cluster sizes in sliding windows. (link)

## Team Members


- Dana Aubakirova
- Diego Andres Torres Guarin
- Benedictus Kent Rachmat


## Project Structure

- `server.py` : Python server script to connect to the API and send individual messages to a socket.
- `analyze.py` : Spark Streaming script to read data, perform sentiment analysis, clustering, and window operations.
- `visualize.py`: Run cluster visualizations (scatter-plot) in real-time, refreshes when the button is pressed.

## Requirements

- Docker
- Python 3.x
- Pyspark
- TextBlob
- Jupyter Notebook
- Matplotlib, Seaborn, or Plotly (for visualization)
- Dash

## Instructions

### Step 1: Get and set the API key

Register and obtain access to NEWS API, change the `NEWSAPI_KEY` variable of `server.py` with your API key (don't use ours more than a couple times please :3)

### Step 2: Set up the docker

At the same level as the docker file run:

```
$ docker build -t my-pyspark-notebook .
```

This will create the image with the necessary files and dependencies

Then run the container,

```
$ docker run -it --rm -p 8888:8888 -p 8050:8050 my-pyspark-notebook
```

and follow the link that looks like *http://127.0.0.1:8888/lab?token=7abaa2f60f10a23ed923*

**Step 3: Run the project**

```
# to run the server
$ python server.py
```

Here, we provide a simple way to stream news articles from the News API to a client in real-time using a socket connection. This code could be used for a variety of applications, such as news aggregation or sentiment analysis.The code starts by defining the News API key and the port number for the socket connection. It then defines a function called *"fetch_articles"* which sends a GET request to the News API endpoint and retrieves the top news articles from the US.

In the main function we set up a server socket to listen for incoming connections on the specified port number. When a client connects to the server, the function enters a loop and continuously fetches articles from the News API using the *"fetch_articles"* function. For each article, it converts the JSON data into a string and sends it to the client using the client socket. The function prints out the article and the message that was sent for debugging purposes.

The function then waits for one second before sending the next article to the client. After all the articles have been sent, the function waits for two minutes before fetching new articles from the News API. This process continues until the program is terminated.

```
# to run spark streaming
$ python analyze.py
```

We use Spark Streaming to perform sentiment analysis on news articles in real-time using TextBlob library. We retrieve articles via a socket connection (described above) and perform sentiment analysis on each article. For that we implement the "sentiment_analysis" function which is using TextBlob, and add the sentiment score to the article. The sentiment score ranges from -1 to 1, with negative scores indicating negative sentiment and positive scores indicating positive sentiment. We use these sentiment scores in streaming k-means clustering to group the articles based on their sentiment. The code prints and saves the cluster number along with the article's title and sentiment score to a CSV file. The resulting solution provides a scalable solution for monitoring news sentiment in real-time.

**Step 4: Visualize data**

```
# to run the scatter-plot visualizations
$ python visualize.py
```

Click to the one of the generated links (the one running on the localhost: http://127.0.0.1:8050 works)

The code is designed to create a web application using Dash, a Python framework for building analytical web applications. The application displays a scatter/bar plot of the average sentiment per cluster based on data loaded from a CSV file called *'predictions.csv'*. The data is updated in real-time.

The layout of the app is defined using HTML and CSS styles. It includes a title, a button to reload the data, the titles of the best and worst news and a scatter plot. The scatter plot is created using the dcc.Graph component from Dash Core Components and is assigned the ID *'sentiment-per-cluster'*.

We define a callback function to update the scatter plot and the titles of the best/worst articles when the reload button is clicked. The callback function takes the number of clicks on the reload button as an input and outputs updated versions of the graph and the titles based on the updated data from the CSV file. The scatter plot is customized to show different colors for each cluster and is updated with a new title and axis labels.

The app is run on a local server using the app.run_server() function with host, debug, and port settings. The app can be accessed through a web browser at the specified port.