Tweet Your Tale: Twitter Sentiment Analysis Deployment

IDS 594: Machine Learning Deployment

Faculty: Dr. Theja Tulabandhula

Team Members

Chakradhaar Viswatmula (67723905)
Nikhita Wadhawan (677862315)
Rahul Dhanasiri (650749738)

Important Links

https://tweetyourtale.herokuapp.com/

https://github.com/nwadha/Tweet-your-Tale---Twitter-Analysis

https://github.com/chakradhaar/gcp-twitter

https://twitter-sentiment-rdnwcv.uc.r.appspot.com/

https://drive.google.com/file/d/17i5I6M4bSRg8qXD1z7k6nfy1xMDfbWRC/view?usp=sharing

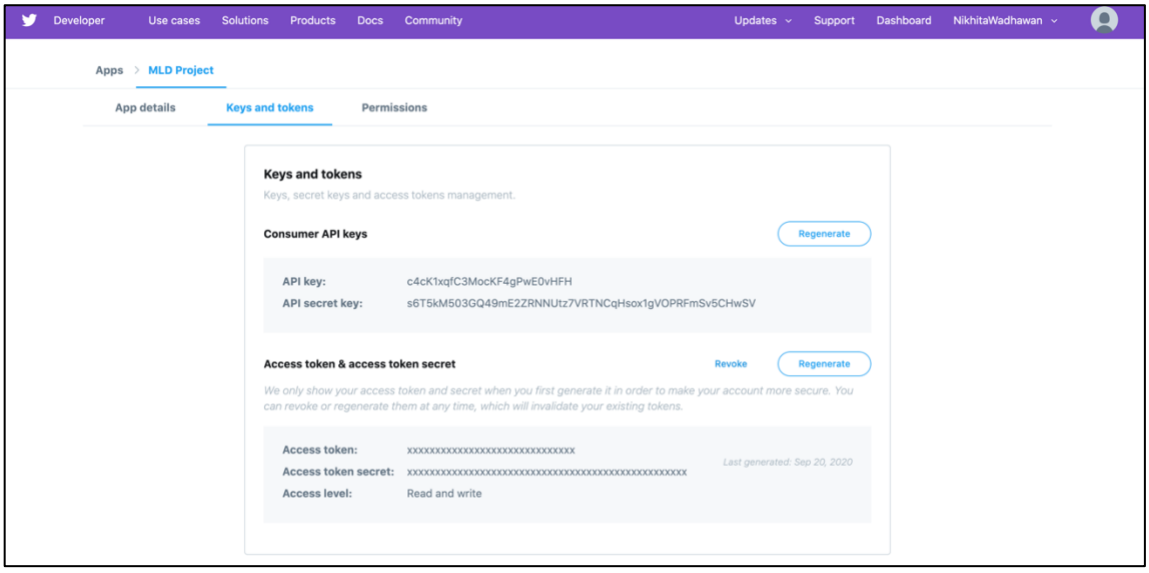
Aim for the Project

The aim for this project is to simulate a real-world process of model deployment. The goal was to deploy a Twitter sentiment analysis on different platforms and compare the ease of use and experience. The team worked with a real time model serving approach using Twitter API to get back sentiment associated to a topic in a number of selected tweets instantly.

Building the Web App

First and foremost, in order to utilize the Twitter API and gain authorization, a twitter developer account is required. Once we sign up, we can create an app and gain custom consumer API Keys and access tokens. These are used to stream tweets real time and analyze the sentiment associated with it. The sign up is free of cost, however the account creation takes a few days. [17]

Figure 1: A snippet of the Twitter Developer Account



Performing sentiment analysis on Twitter tweets becomes pain free using packages like Tweepy and Textblob. Tweepy is an easy to use and effective python library to stream tweets from twitter. It helps with better handling, connectivity, message routing and session management. Textblob is also extremely useful for processing textual data. It is a python library that provides a simple API that can perform natural language processing tasks like POS or parts of speech tagging, sentiment analysis, translation, tokenization, parsing and classification. [4][5]

It is a cumbersome process to change the style of a website and to maintain consistency. Using a template comes in handy here, because we define a basic layout for our page and define which element is subject to change. Since we have no prior experience of front-end development, we leveraged a pre-designed template (index.html) and placed it in the templates folder. Additionally, we placed the style.css file in the static folder, which is a place where any files needed by the web application reside. [10]

After getting the basics ready for what the front end will look like, we move on to the next step of creating a flask app for our web application.

A flask app is a web framework. As discussed in class, it provides tools and libraries that allow the creation of a web application which can be a webpage, a blog post or a commercial website. Flask is a micro framework. Micro frameworks are frameworks that have little to no dependencies on other external libraries. While this is beneficial because it keeps the framework light, fewer dependencies to update and watch out for bugs, it requires additional effort of adding plug-ins. Thus, using flask enables functions to be exposed to HTTP locations. Most of the tools used for building web services in the python environment work with Flask very well. [8][9]

After importing the necessary packages like numpy, flask, textblob and tweepy, we create a flask app followed by a function to calculate the different polarities of the incoming tweets. The first step is loading the flask library and creating a flask object with the special variable 'name' followed by defining a prediction function that specifies that the function is hosted at "/" and can be accessed by HTTP GET and POST commands. [18]

A route for the web application for connecting to the front end is required. In the next step we add the credentials we had obtained initially from our Twitter Developer account and assign them to variables and obtain authorization access to twitter API. Due to the limited scope, we are only streaming tweets in English language.

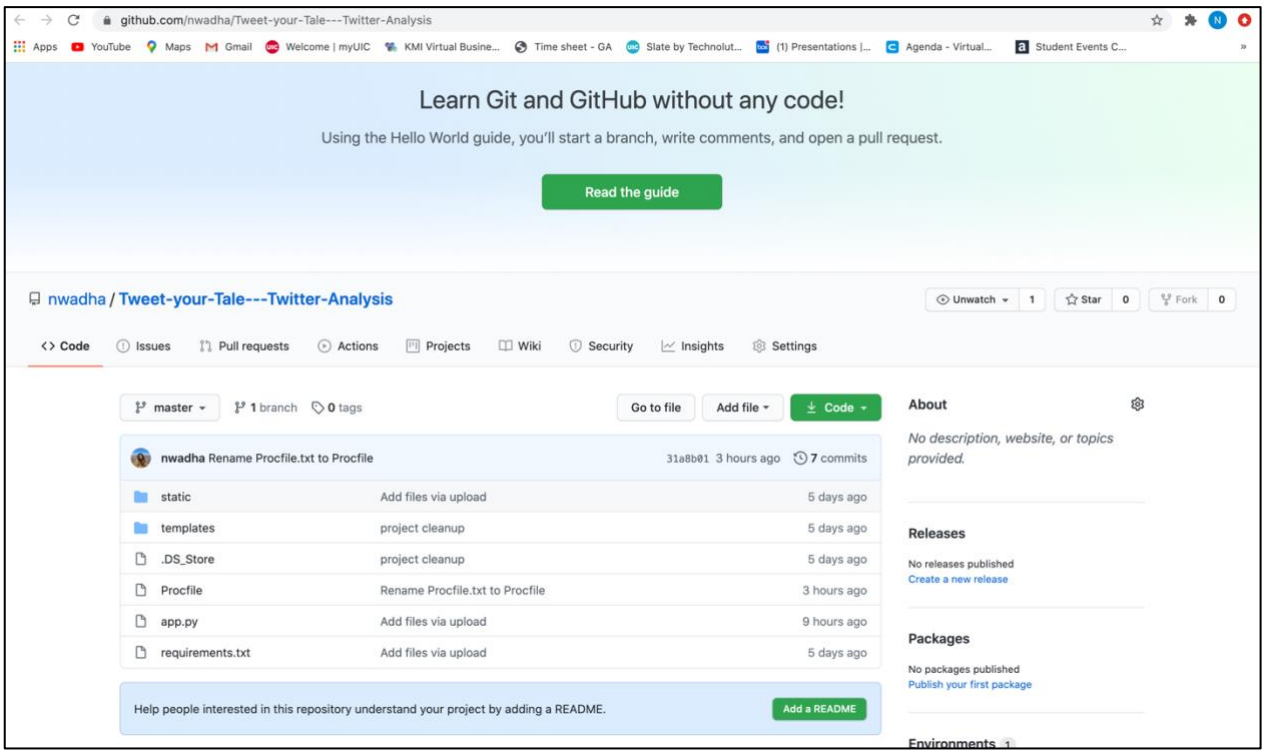
Post iterating through the different tweets and evaluating the overall sentiment, we send the output back to the front end. Please refer to the attached app.py file for more details.

We placed our flask app code, templates and static folder in one folder and created a repository on Github and uploaded the files there.

We completed deployment through two different methods:

1. Using Platform as a Service: Heroku
2. Using Cloud Environment Infrastructure as a Service: Google Cloud Platform

Figure 2: A snippet of the Github Repository used



Heroku Deployment

Heroku is an easy to use cloud platform as a service, supporting several programming languages like Python, PHP, Scala, Java etc. Heroku is well known for running applications in dynos. Dynos are virtual machines that are powered up and down on the basis of the application size.

Although Heroku platform charges by the dyno, the entire platform as well as every application that is built on the platform is deployed to AWS. But why are we using this instead of AWS directly? To simply put it, it's very convenient!

Heroku offers a great experience that is designed to make developers' lives easier with providing the ability to run applications at scale with a few commands on the Heroku CLI and Dashboard.

Benefits of Heroku

1. It provides a great developer experience. It is easy to navigate, login and manage applications
2. It is open and extensible, so it allows the convenience of using whichever language the developer is comfortable with. It provides the additional benefit of a huge network of add on options. These add ons are powerful features and functionalities that can be deployed at the click of a button to our app and provide additional functionality.
3. It has a free tier which allows access to the platform for small projects without shelling too many dollars.
4. It has Heroku Shield, which saves a great deal of time when we are trying to build HIPAA compliance.
5. Detailed logs which helps understand where we went wrong with error codes that can be looked up and the issue can be understood conveniently and rectified. [16]

We created two files, Procfile (no extension) and requirements.txt in order to deploy it on Heroku. The Procfile allows Heroku to be able to decide which file has to run first after deployment and the requirements.txt file contains the names of the packages used that allows Heroku to add these essential packages before deployment if not already available.

We have used the free tier for Heroku platform. In order to use the platform, we have to sign up on the website and create a new application. By clicking the button 'Create a new app', we can do so. We named our app 'tweetyourtale'.

Once we have created the app, we can connect our GitHub repository. When we open the app page, we can go down to 'Deployment Method' and click on connect to Github. Then we can add our GitHub account and search for the respective repository where we have uploaded our files. After choosing the required repository, we can simply click connect.

In order to deploy the branch now, we move to the 'Manual Deploy section' and click on 'Deploy Branch'. By doing so we are allowing all our dependencies to get installed and once the app has been successfully installed, we will get success message. We can view the logs using 'heroku logs' to see the detailed logs.

Using Heroku for deployment was convenient and hassle free. This is what our app looks like! Below is an example where we entered a word 'violence' and analyzed 500 tweets and see the sentiment associated with it.

Figure 3: A snippet of the app on Heroku

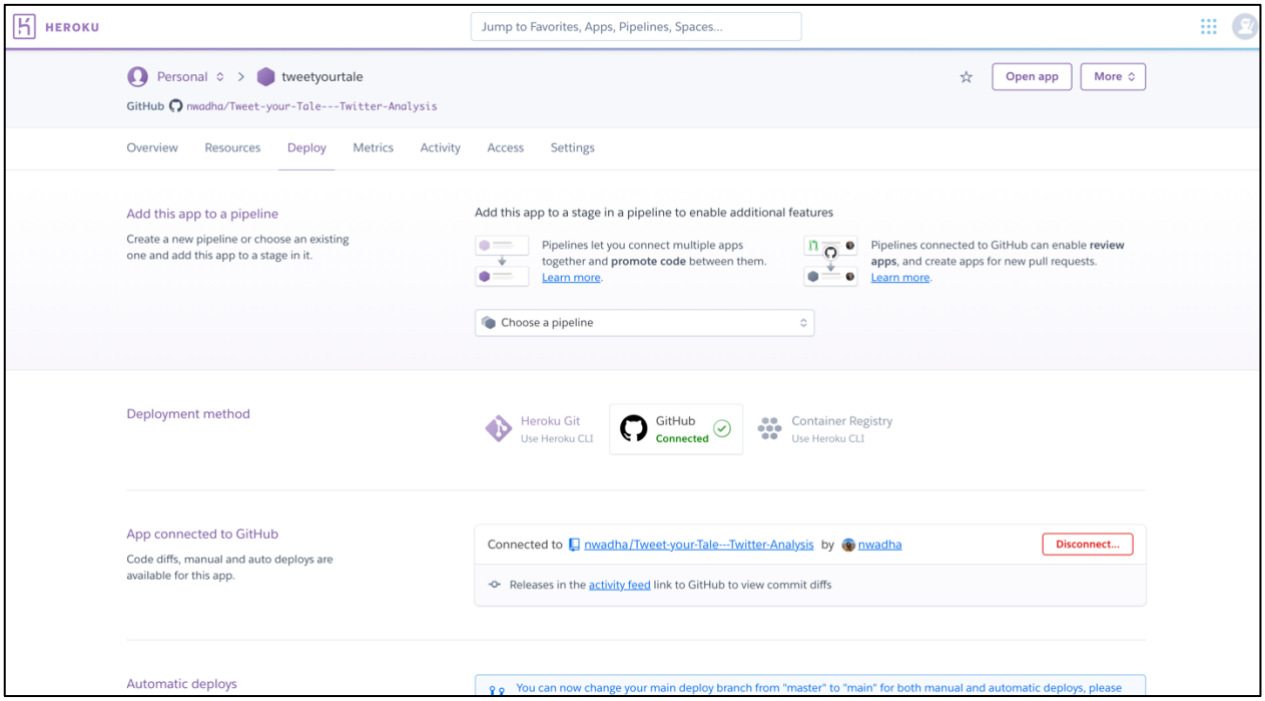
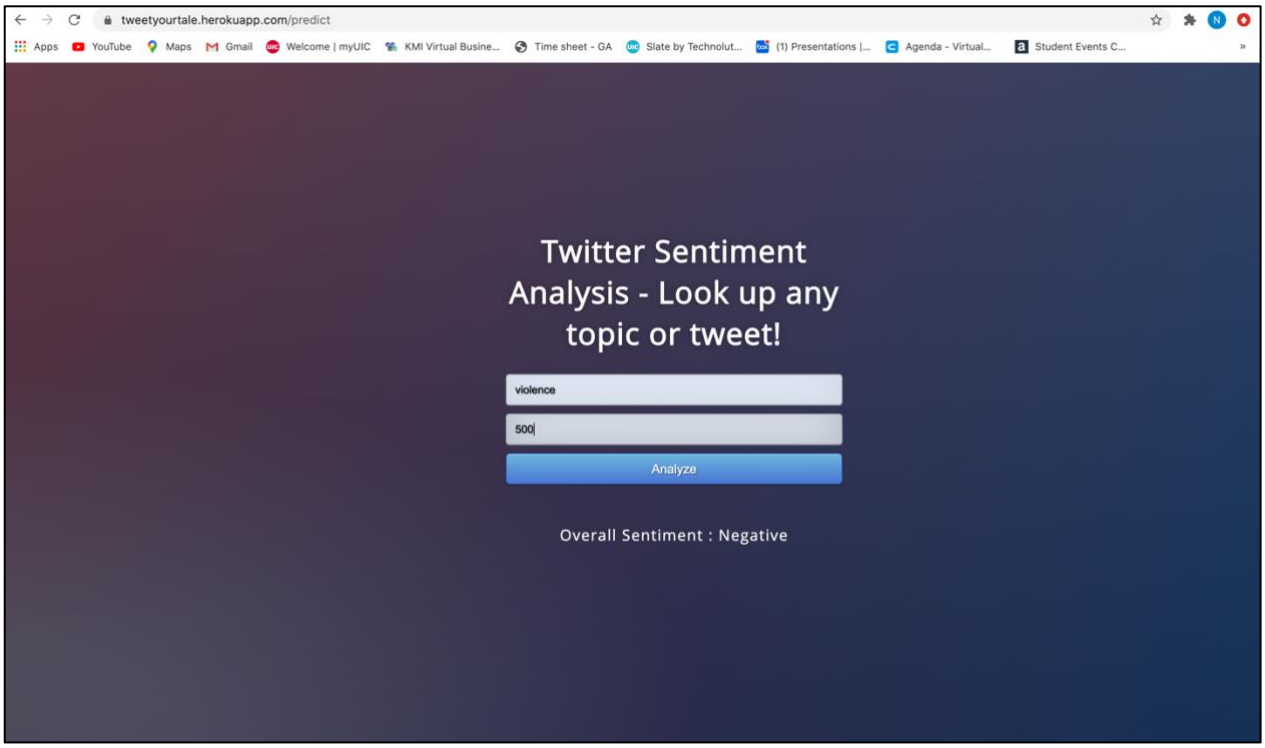


Figure 4: An example of a tweet search



GCP App Engine Deployment

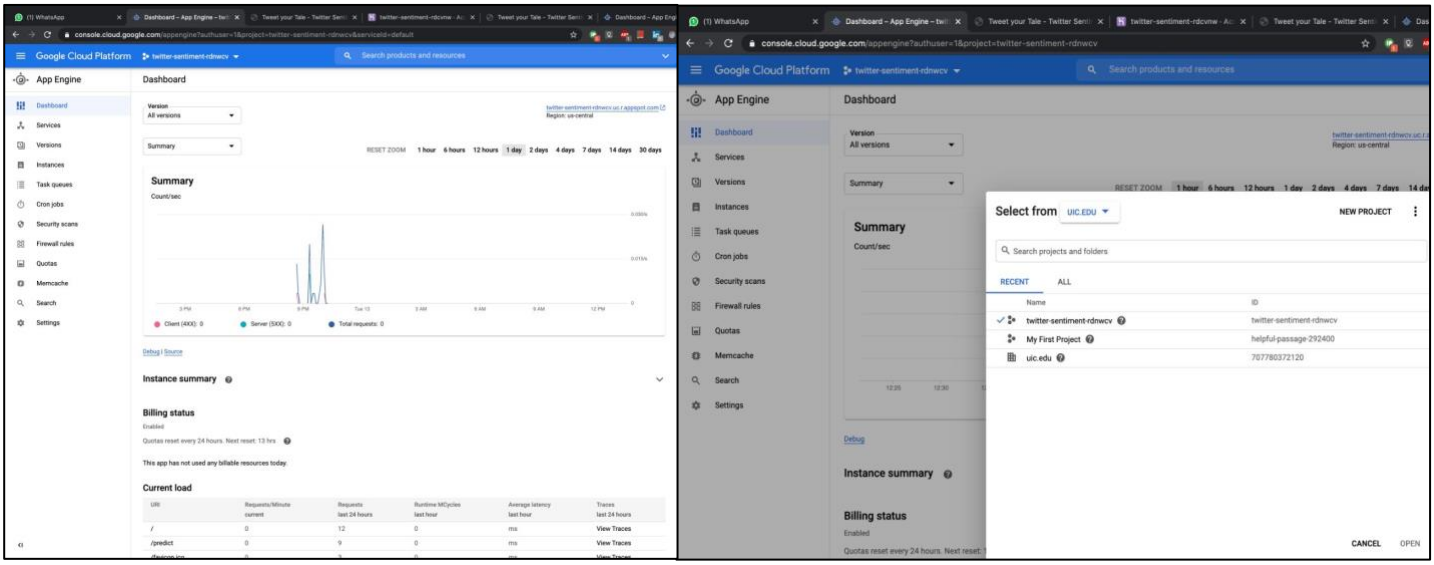
Google Cloud's functions as a service (FaaS) offering provides the ability to have serverless execution environment which can be used for building and connecting various cloud services. With Cloud Functions, we can write simple, single-purpose functions that are attached to events that are emitted from the cloud infrastructure and services. The function is triggered when an event being watched is fired. The code executes in a fully managed environment. It makes working on projects easier since there is no need to provision any infrastructure or worry about managing any servers.

Cloud Functions can be written using JavaScript, Python 3, Go, or Java. We can take the function and run it in any standard Node.js (Node.js 10), Python 3 (Python 3.7), Go (Go 1.11 or 1.13) or Java (Java 11) environment, which makes both portability and local testing a breeze.

Benefits of Cloud Functions

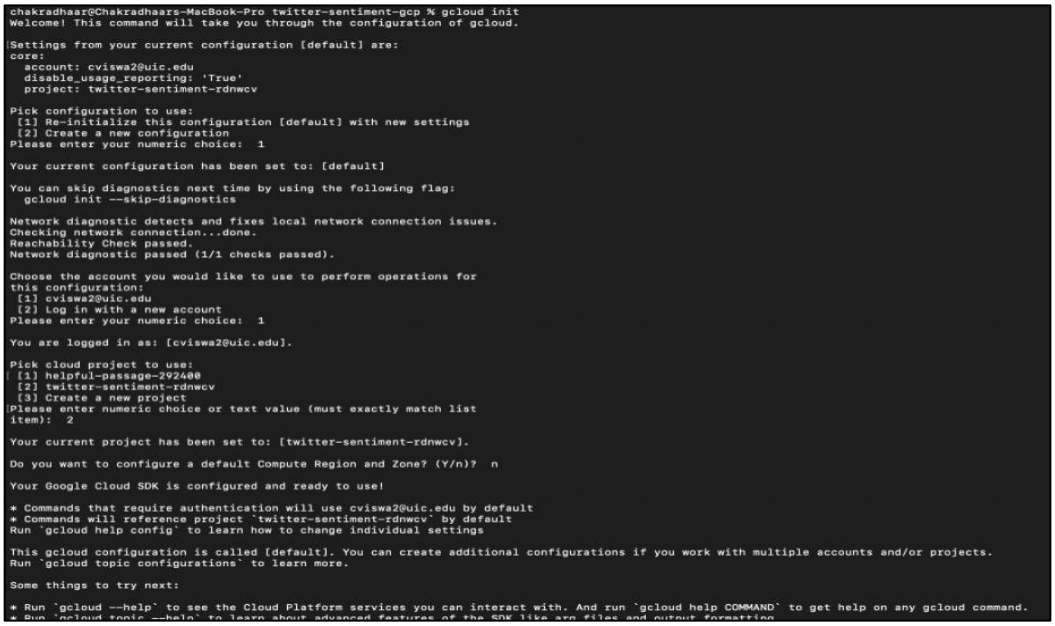
1. Lightweight APIs that compose loosely coupled logic into applications.
2. Data processing and ETL operations, for scenarios such as video transcoding and IoT streaming data.
3. Webhooks to respond to HTTP triggers.
4. Mobile backend functions.[19]

Figure 5: A snippet of project (twitter-sentiment-rdnwcv) in GCP console.



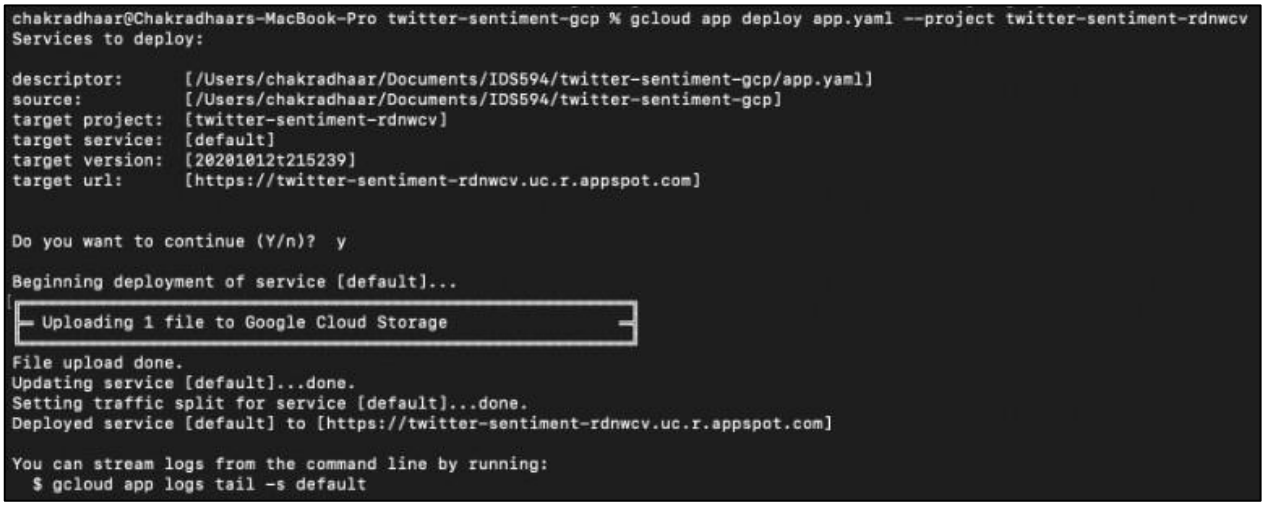
We started off by creating a project (twitter-sentiment-rdnwcv) in GCP console. We installed SDK onto our local machine and then initialized gcloud on our local repository and configured it, choosing the account to perform operations for the specified configuration and picking the cloud project to use. It also asks for a default Compute Region which we chose as US Central.

Figure 6: A snippet of the terminal



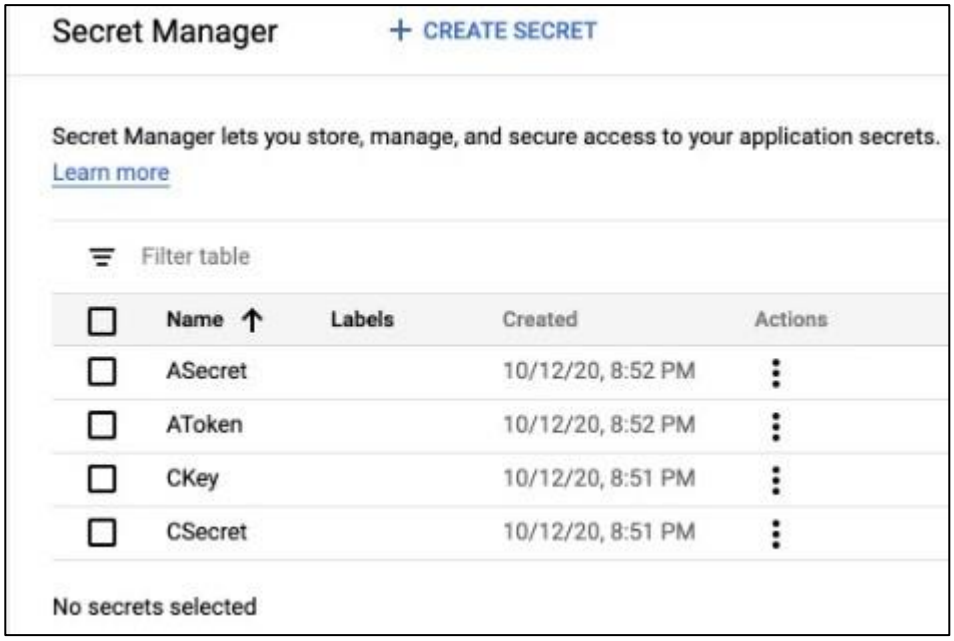
Once that's done, we just use the command to deploy our project. GCP deployment requires us to name our main python file as main.py and have a yaml file which provides the runtime environment information and the environment variables if any. Just like heroku we need a requirements.txt file here as well and that's it, our model is deployed with the url https://twitter-sentiment-rdnwcv.uc.r.appspot.com/

Figure 7: A snippet of uploading the file on gcloud



In addition to having just uploaded our model, we have also enabled the Secret Manager API provided by GCP. This service was specifically created in order to help hide any private variables such as API Keys to be hidden and prevent them from theft. For this, we had to set our twitter api keys on the secret manager dashboard and use the google.cloud.secretmanager package to retrieve it in our main.py. This is just good programming practice.

Figure 8 : A snippet of the Secret Manager



Cost Comparison

Platform/Service	Free Tier	Free Hours	Cost Control Measure	App Management
Heroku	Yes	550	Automatic cutoff after inactivity > 30 minutes	Heroku Dashboard and Heroku CLI
GCP	Yes	3 months		GCP console & CLI

References

1. https://tweetyourtale.herokuapp.com/
2. https://dashboard.heroku.com/apps/tweetyourtale/deploy/github
3. https://github.com/nwadha/Tweet-your-Tale---Twitter-Analysis
4. http://docs.tweepy.org/en/latest/
5. https://textblob.readthedocs.io/en/dev/
6. https://github.com/matplotlib/matplotlib/tree/master/lib/matplotlib/mpl-data/stylelib
7. https://chicagodatascience.github.io/MLOps/lecture1/flask/
8. https://pymbook.readthedocs.io/en/latest/flask.html#:~:text=Flask%20is%20a%20web%20framework,application%20or%20a%20commercial%20website.
9. https://pythonhow.com/how-a-flask-app-works/
10. https://github.com/amitthakur2013/sentiment-project-1
11. https://devcenter.heroku.com/articles/procfile
12. https://devcenter.heroku.com/articles/dyno-types
13. https://devcenter.heroku.com/articles/error-codes#h14-no-web-dynos-running
14. https://stackoverflow.com/questions/48696878/missing-required-flag-heroku-tail
15. https://devcenter.heroku.com/articles/github-integration
16. https://www.infoworld.com/article/3229350/5-foolish-reasons-youre-not-using-heroku.html
17. https://developer.twitter.com/en/apps
18. https://leanpub.com/ProductionDataScience/
19. https://cloud.google.com/docs/overview/cloud-platform-services