

**NAME : RITAM MAJUMDER**  
**COLLEGE : TECHNO INTERNATIONAL NEWTOWN**  
**BRANCH : ELECTRONICS AND COMMUNICATION**  
**ENGINEERING**  
**YEAR : 3<sup>RD</sup>**

**Github profile link: <https://github.com/ritam0210>**

**Github repository for the projects:**  
**<https://github.com/ritam0210/Major-Project>**

# MAJOR PROJECT 1

## Text Emotion Predictor using Support Vector Classifier (SVC)

### 1. Taking data and creating Dataframe

```
#Creating dataframe
import pandas as pd
df= pd.read_csv('/content/tweet_emotions.csv')
df
```

	tweet_id	sentiment	content
0	1956967341	empty	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	wants to hang out with friends SOON!
4	1956968416	neutral	@dannycastillo We want to trade with someone w...
...	...	...	...
39995	1753918954	neutral	@JohnLloydTaylor
39996	1753919001	love	Happy Mothers Day All my love
39997	1753919005	love	Happy Mother's Day to all the mommies out ther...
39998	1753919043	happiness	@niariley WASSUP BEAUTIFUL!!! FOLLOW ME!! PEE...
39999	1753919049	love	@mopedronin bullet train from tokyo the gf ...

40000 rows × 3 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   tweet_id    40000 non-null  int64
1   sentiment   40000 non-null  object
2   content     40000 non-null  object
dtypes: int64(1), object(2)
memory usage: 937.6+ KB
```

```
[ ] df.shape
```

```
(40000, 3)
```

```
[ ] df.size
```

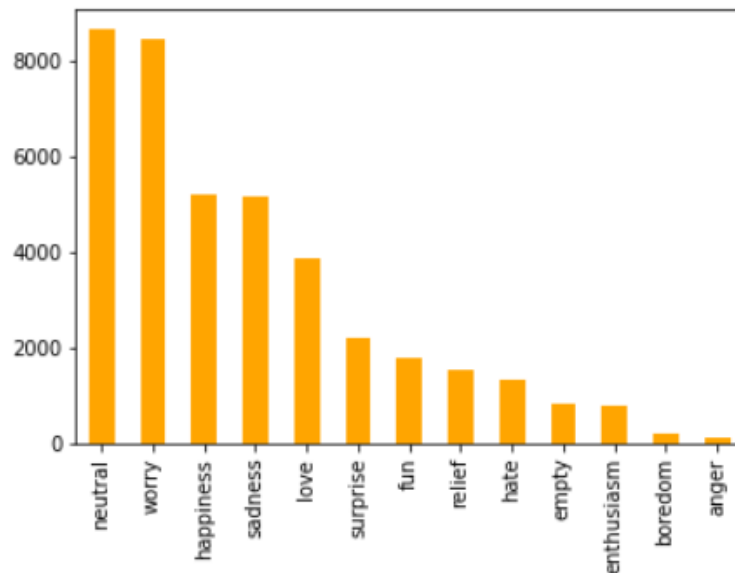
```
120000
```

2. Pre-processing or data cleaning is not required for this dataset

### 3. Data Visualisation

```
[ ] #Data Visualisation
df['sentiment'].value_counts().plot(kind='bar', color='orange')
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f45b972af10>



```
▶ df['sentiment'].value_counts()
```

```
↳ neutral      8638
   worry       8459
   happiness    5209
   sadness     5165
   love        3842
   surprise     2187
   fun         1776
   relief      1526
   hate        1323
   empty       827
   enthusiasm  759
   boredom     179
   anger       110
   Name: sentiment, dtype: int64
```

#### 4. Dividing data into input and output

```
▶ #Dividing data into input and output
x = df.iloc[:,1]
y = df.iloc[:,0]
print(x)
print(y)
```

```
↳ 0      @tiffanylue i know i was listenin to bad habi...
   1      Layin n bed with a headache ughhhh...waitin o...
   2      Funeral ceremony...gloomy friday...
   3      wants to hang out with friends SOON!
   4      @dannycastle We want to trade with someone w...
      ...
39995      @JohnLloydTaylor
39996      Happy Mothers Day All my love
39997      Happy Mother's Day to all the mommies out ther...
39998      @niariley WASSUP BEAUTIFUL!!! FOLLOW ME!! PEE...
39999      @mopedronin bullet train from tokyo the gf ...
Name: content, Length: 40000, dtype: object
0      empty
1      sadness
2      sadness
3      enthusiasm
4      neutral
      ...
39995      neutral
39996      love
39997      love
39998      happiness
39999      love
Name: sentiment, Length: 40000, dtype: object
```

#### 5. Training and Testing variables

```
[ ] #train_test_split
    from sklearn.model_selection import train_test_split
    x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=0)
```

## 6. Normalization of data

```
[ ] #Applying TFIDF Vectorizer
    from sklearn.feature_extraction.text import TfidfVectorizer
    vec = TfidfVectorizer()
    xtr_v = vec.fit_transform(x_train)
    xts_v = vec.transform(x_test)
```

```
[ ] xtr_v
```

```
<30000x39437 sparse matrix of type '<class 'numpy.float64'>'
    with 356125 stored elements in Compressed Sparse Row format>
```

## 7. Running a Classifier

```
[ ] #Applying Classifier
    from sklearn.svm import SVC
    model=SVC()
```

## 8. Fitting the model

```
[ ] #Model fitting
    model.fit(xtr_v,y_train)

    SVC()
```

## 9. Predicting the Output

```
▶ #Predicting the Output
  y_pred = model.predict(xts_v)
  y_pred
  #Predicted values

📄 array(['sadness', 'worry', 'neutral', ..., 'happiness', 'neutral',
        'neutral'], dtype=object)
```

```

▶ y_test #Actual values

12836    sadness
10913    neutral
4214     sadness
8198      empty
31403      fun
...
30790    neutral
5690     sadness
17736    surprise
12098    neutral
5315      empty
Name: sentiment, Length: 10000, dtype: object

```

## 10. Evaluation: Accuracy Score

```

[ ] #Accuracy
    from sklearn.metrics import accuracy_score
    accuracy_score(y_pred,y_test)*100

```

35.35

- Pipelining of the model for deployment

```

[ ] #Pipelining
    from sklearn.pipeline import make_pipeline
    text_model = make_pipeline(TfidfVectorizer(),SVC())
    text_model.fit(x_train,y_train)

```

Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer()), ('svc', SVC())])

```

[ ] #Predictor variable
    y_pred1 = text_model.predict(x_test)
    y_pred1

```

array(['sadness', 'worry', 'neutral', ..., 'happiness', 'neutral',  
 'neutral'], dtype=object)

```

[ ] #Accuracy of pipelined model
    accuracy_score(y_pred1,y_test)*100

```

35.35

- Individual prediction

```
[ ] #Individual prediction
a1= df['content'][14050]
a1
```

'gotta go twitterers (?) my stupid sister wants to go on facebook going to montreal 2morrow so i wont be on for a while! bye!! XoXox

```
[ ] a1= vec.transform([a1])
model.predict(a1)
```


array(['happiness'], dtype=object)


- Creating a new file using JOBLIB and dumping the trained pipelined model in it


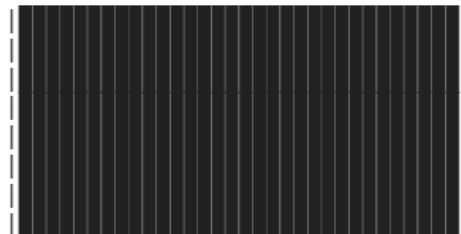
```
[ ] #JOBLIB
import joblib
joblib.dump(text_model, 'tweet_emotions')
```

['tweet\_emotions']

- Temporary deployment/ Local Deployment

 #Temporary deployment

 !pip install streamlit --quiet

9.2 MB	6.7 MB/s
235 kB	46.9 MB/s
181 kB	60.0 MB/s
78 kB	5.4 MB/s
164 kB	49.7 MB/s
4.7 MB	36.0 MB/s
63 kB	1.4 MB/s
51 kB	7.3 MB/s

Building wheel for validators (setup.py) ... done

```
[ ] %%writefile app.py
%%writefile is amagic command to create app.py file
import streamlit as st
import joblib
model = joblib.load('tweet_emotions')
st.title('TEXT EMOTIONS CLASSIFIER') #creates a title in web app
ip = st.text_input('Enter the text') #creates a text box in web app
op = model.predict([ip])
if st.button('Predict'):
    st.title(op[0]) # st.button will create a button with name Predict
    #st.title(op[0]) # the output will be displayed as a title
```

Writing app.py

↑ ↓ ↶ ↷ ⚙ 📄 🗑 ⋮

🔊

!streamlit run app.py & npx localtunnel --port 8501

... 2022-09-24 07:02:26.065 INFO numexpr.utils: NumExpr defaulting to 2 threads.

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to False.

You can now view your Streamlit app in your browser.

Network URL: <http://172.28.0.2:8501>

External URL: <http://34.125.200.94:8501>

npx: installed 22 in 4.213s

your url is: <https://light-numbers-grin-34-125-200-94.local.lt>

Tunnel website ahead! x +

light-numbers-grin-34-125-200-94.local.lt

light-numbers-grin-34-125-200-94.local.lt

Friendly Reminder

This website is served via a [localtunnel](#). This is just a reminder to always check the website address you're giving personal, financial, or login details to is actually the real/official website.

Phishing pages often look similar to pages of known banks, social networks, email portals or other trusted institutions in order to acquire personal information such as usernames, passwords or credit card details.

Please proceed with caution.

Click to Continue

If you're the developer...

You and other visitors will only see this page from a standard web browser once per IP every 7 days.

Webhook, IPN, and other non-browser requests "should" be directly tunnelled to your localhost. If your webhook/ipn provider happens to send requests using a real browser user-agent header, those requests will unfortunately also be blocked / be forced to see this tunnel reminder page. FYI, this page returns a 401 HTTP Status.

Options to bypass this page:

1. Set and send a `Bypass-Tunnel-Reminder` request header (its value can be anything).
2. or, Set and send a custom / non-standard browser `User-Agent` request header.

This localtunnel session is sponsored by: [Tunez.com](#) - [music discovery made easy](#)

app - Streamlit x +

light-numbers-grin-34-125-200-94.local.lt

TEXT EMOTIONS CLASSIFIER

Enter the text

The Earth revolves around the Sun in 365 days

Predict

neutral

Made with Streamlit

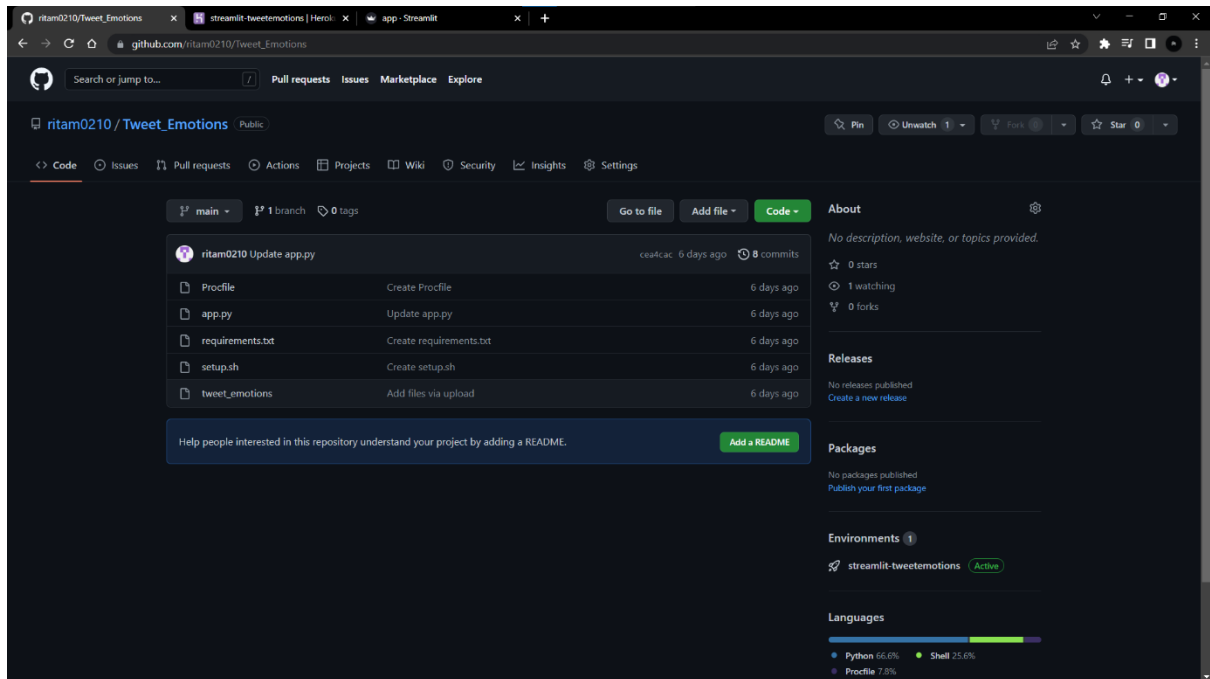
8



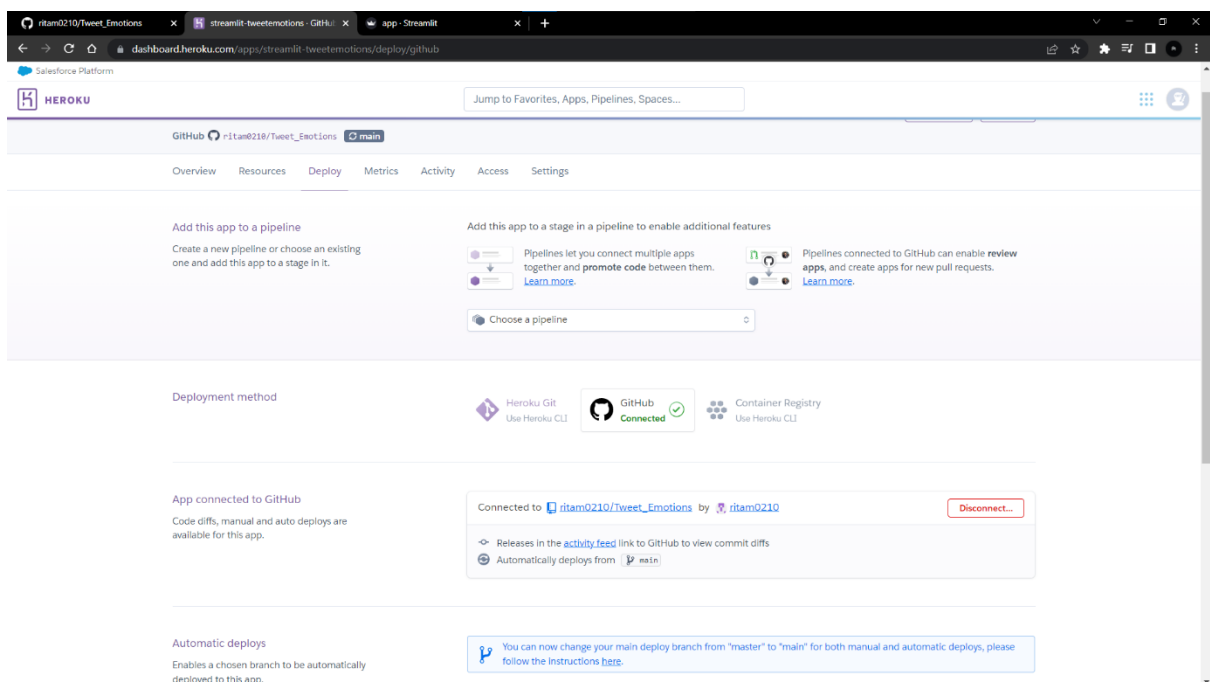
- **Permanent deployment using Heroku:** <https://streamlit-tweetemotions.herokuapp.com/>

**Github Repository:** [https://github.com/ritam0210/Tweet\\_Emotions](https://github.com/ritam0210/Tweet_Emotions)

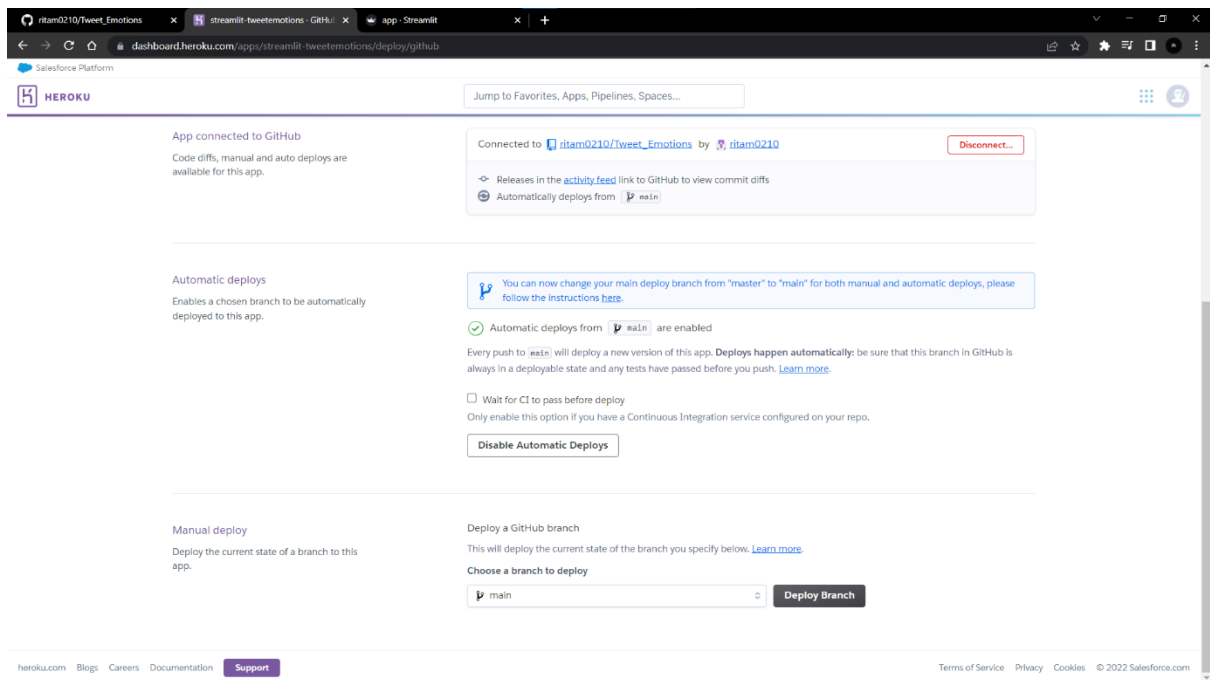
Uploaded and added required files for the web app



**Connecting the Github account and adding the repository on Heroku:**



## Enabling Automatic Deploys for the app:



App connected to GitHub

Code diffs, manual and auto deploys are available for this app.

Connected to **ritam0210/Tweet\_Emotions** by **ritam0210** [Disconnect...](#)

- Releases in the [activity feed](#) link to GitHub to view commit diffs
- Automatically deploys from **main**

**Automatic deploys**

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please [follow the instructions here](#).

☒ Automatic deploys from **main** are enabled

Every push to **main** will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch in GitHub is always in a deployable state and any tests have passed before you push. [Learn more](#)

☐ Wait for CI to pass before deploy

Only enable this option if you have a Continuous Integration service configured on your repo.

[Disable Automatic Deploys](#)

**Manual deploy**

Deploy the current state of a branch to this app.

Deploy a GitHub branch

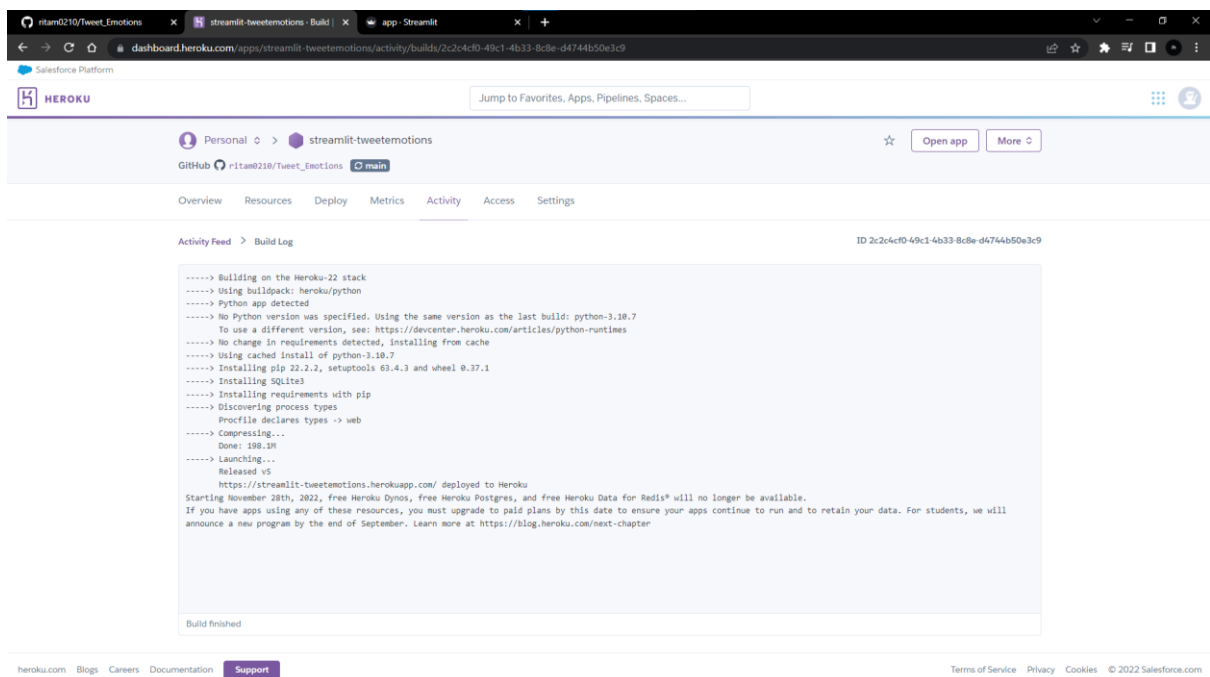
This will deploy the current state of the branch you specify below. [Learn more](#)

Choose a branch to deploy

**main**

[Deploy Branch](#)

Made some minor changes in the app.py file in the Github repository and the app started building:



Personal > streamlit-tweetemotions

GitHub **ritam0210/Tweet\_Emotions** **main**

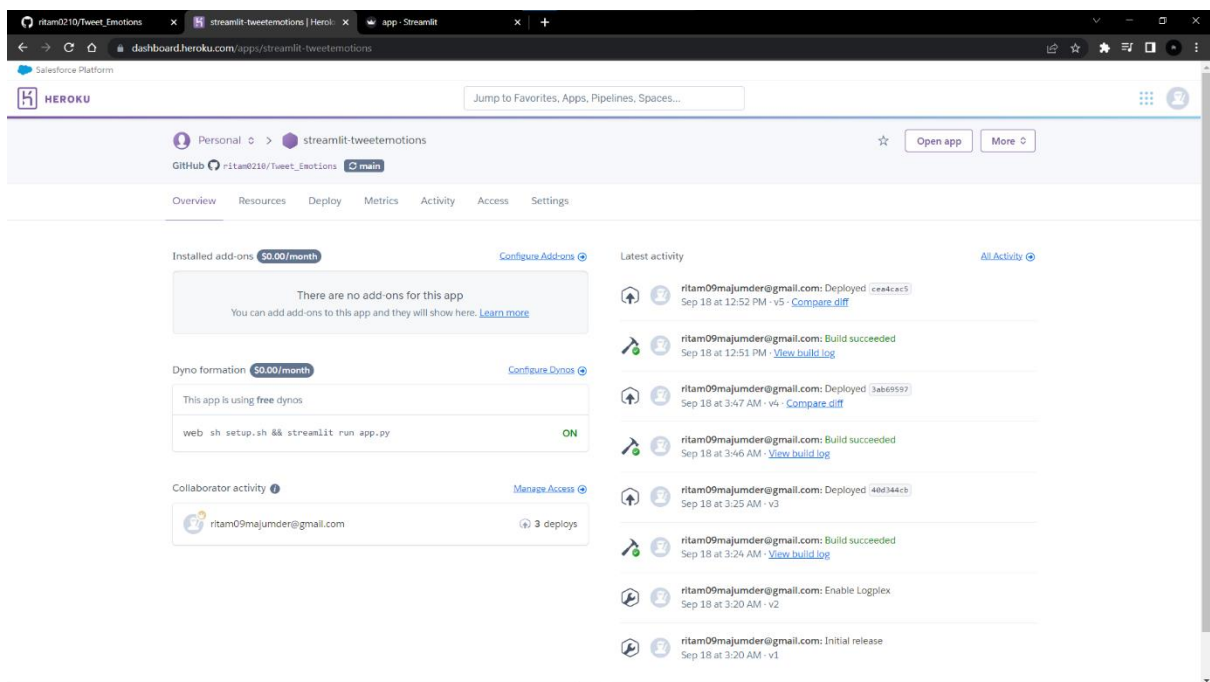
Overview Resources Deploy Metrics Activity Access Settings

Activity Feed > Build Log ID 2c2c4cfd-49c1-4b33-8c8e-d4744b50e3c9

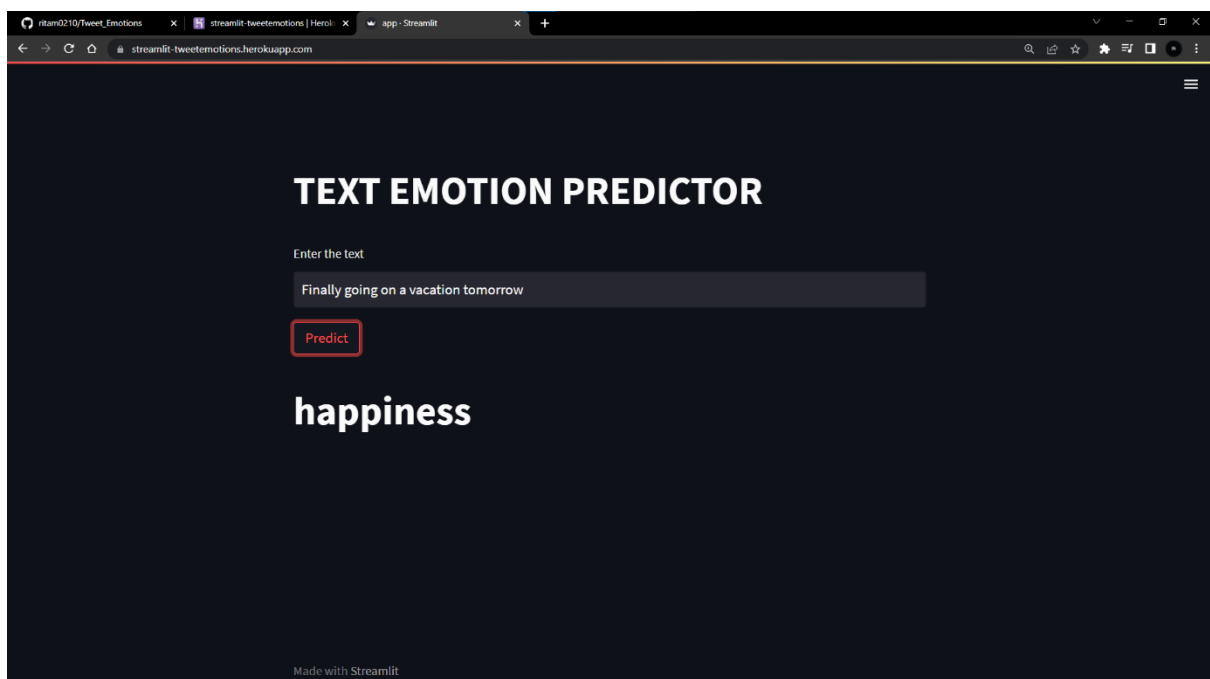
```
-----> Building on the Heroku-22 stack
-----> Using buildpack: heroku/python
-----> Python app detected
-----> No Python version was specified. Using the same version as the last build: python-3.10.7
    To use a different version, see: https://devcenter.heroku.com/articles/python-runtimes
-----> No change in requirements detected, installing from cache
-----> Using cached install of python-3.10.7
-----> Installing pip 22.2.2, setuptools 63.4.3 and wheel 0.37.1
-----> Installing SQLite3
-----> Installing requirements with pip
-----> Discovering process types
Procfile declares types -> web
-----> Compressing...
Done: 198.3M
-----> Launching...
Released v5
https://streamlit-tweetemotions.herokuapp.com/ deployed to Heroku
Starting November 28th, 2022, free Heroku Dynos, free Heroku Postgres, and free Heroku Data for Redis® will no longer be available.
If you have apps using any of these resources, you must upgrade to paid plans by this date to ensure your apps continue to run and to retain your data. For students, we will
announce a new program by the end of September. Learn more at https://blog.heroku.com/next-chapter
```

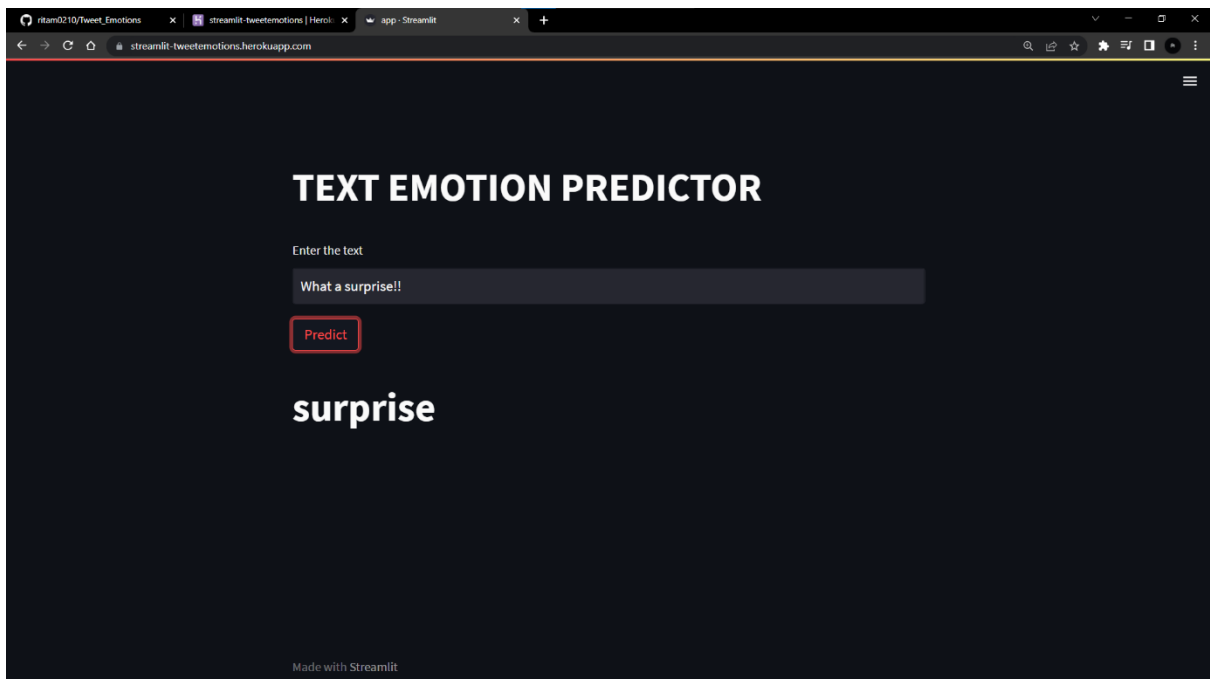
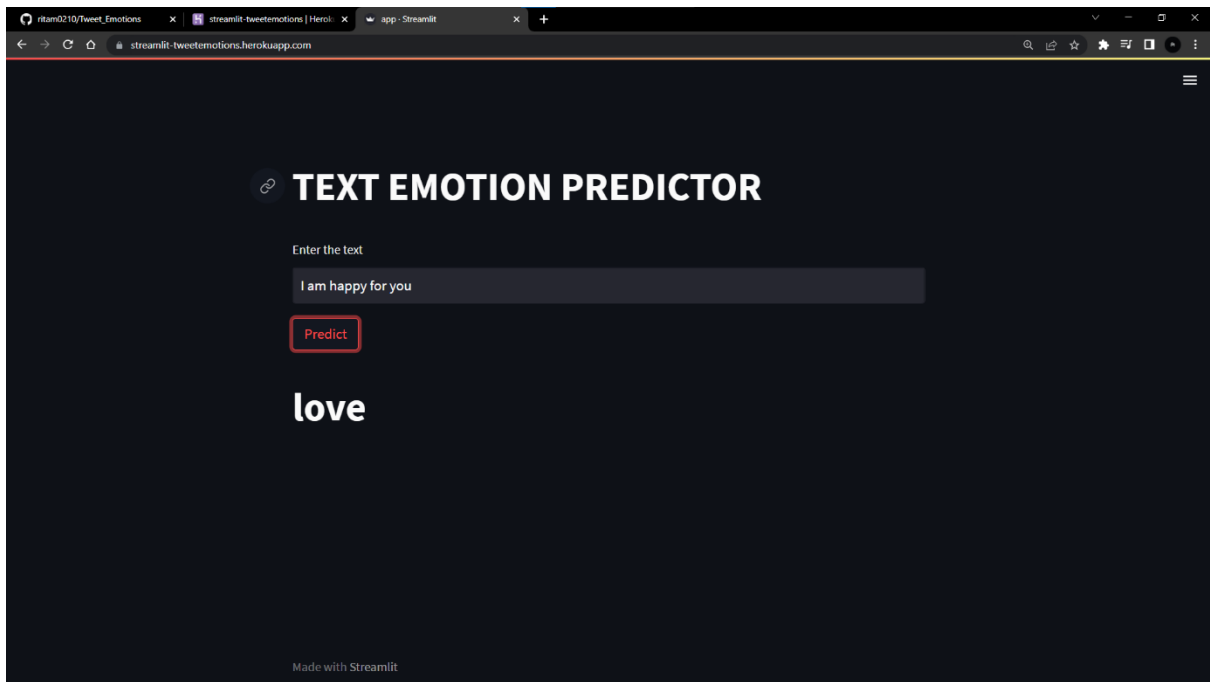
Build finished

App built successfully and ready to be opened:



Heroku app screenshots:





## **MAJOR PROJECT 2**

### **Object Detection (in both real-time and from a video file) using openCV**

```
#OBJECT DETECTION USING OPENCV
```

```
import cv2
```

```
#importing the tensorflow trained model:
```

```
conf_file = 'ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt'
```

```
weights_file = 'frozen_inference_graph.pb'
```

```
mod = cv2.dnn_DetectionModel(conf_file, weights_file)
```

```
#defining the object classes using the coco dataset:
```

```
classObjs = []
```

```
obj_file = 'objs.txt'
```

```
with open(obj_file,'rt') as fr:
```

```
    classObjs = fr.read().rstrip('\n').split('\n')
```

```
#defining input frame parameters according to the model configuration:
```

```
mod.setInputSize(320,320)
```

```
mod.setInputScale(1.0/127.5)
```

```
mod.setInputMean((127.5,127.5,127.5))
```

```
mod.setInputSwapRB(True)
```

```
#cap= cv2.VideoCapture(0) #for video capture through webcam
```

```
cap= cv2.VideoCapture('Busy_city.mp4') #for video capture through video file
```

```

while True:

    ret, frame = cap.read()

    #passing the frames to the model with a detection threshold of 0.5
    clsIn, confidence, bbox = mod.detect(frame,confThreshold=0.5)
    print(clsIn)

    if (len(clsIn) != 0):

        #zip is used as we have to consider 3 variables
        #flattening is used to convert the variables to an 1D-array
        for ClassIndex, conf, boxes in zip(clsIn.flatten(), confidence.flatten(), bbox):

            if (ClassIndex<=80):

                cv2.rectangle(frame,boxes,(255,0,0),2)  #blue rectangle frame
                cv2.putText(frame,classObjs[ClassIndex-1].upper(),
                            (boxes[0]+10,boxes[1]+35), cv2.FONT_HERSHEY_TRIPLEX, fontScale=0.7,
                            color= (0,255,0), thickness=2)

                #this shows the name of the objects in the rectangular frame in uppercase in green
color

            cv2.imshow('Object Detection',frame)

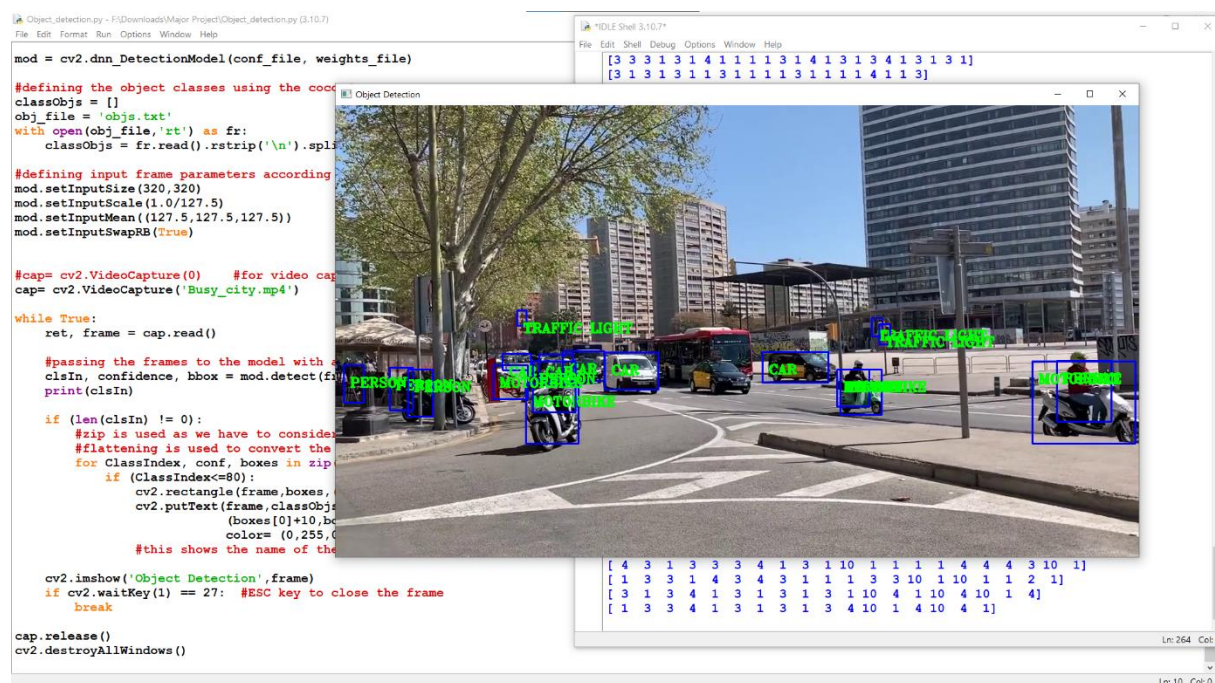
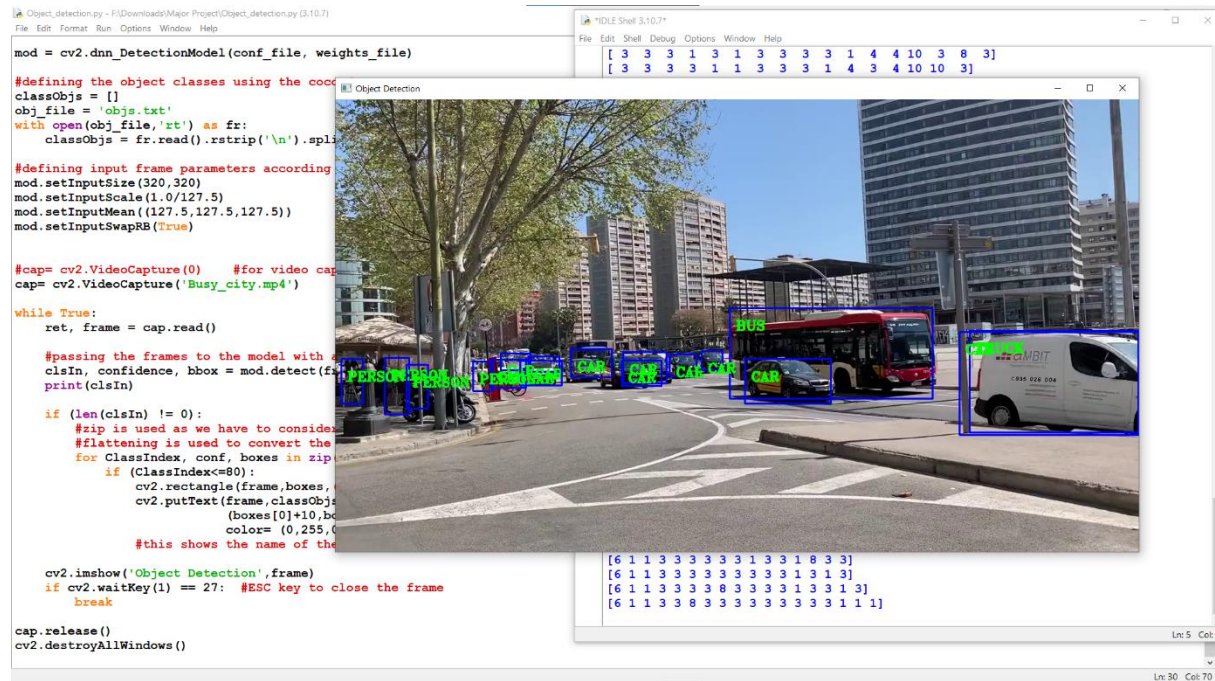
            if cv2.waitKey(1) == 27: #ESC key to close the frame
                break

cap.release()
cv2.destroyAllWindows()

```

## OUTPUT:

From a video file:



## From a webcam:

