Data Glacier

DEPLOYMENT ON FLASK

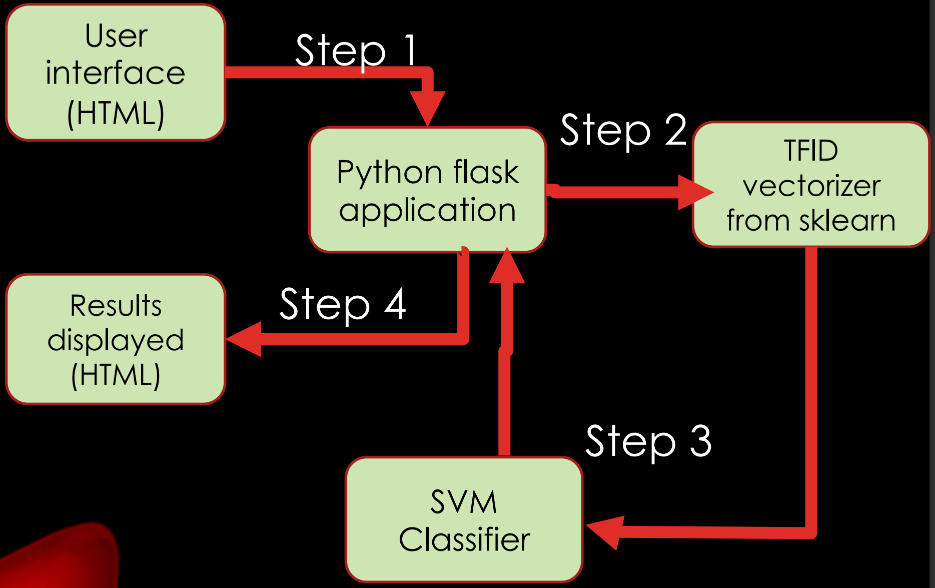
Name : Manal Shahab

Date : 27 July 2022

TABLE OF CONTENT

Data Glacier

INTRODUCTION



Application of workflow

This project includes deploying machine learning model(SVM) using the Flask Framework. As a demonstration, our model help to predict the spam and non spam comment of YouTube. Focusing on both

1. Building a machine learning model for YouTube comments SD
2. Then create an API for the model, using Flask, the Python micro- framework for building web application.

This API allows us to utilize predictive capabilities through HTTP requests

DATA INFORMATION

The samples were extracted from the comments section of five videos that were among the 10 most viewed on YouTube during the collection period. The table below lists the datasets, the YouTube video ID, the number of samples in each class and the total number of samples per dataset.

Dataset	YouTube ID	Spam	Ham	Total
Psy	9bZkp7q19f0	175	175	350
KatyPerry	CevxZvSJLk8	175	175	350
LMFAO	KQ6zr6kCPj8	236	202	438
Eminem	uelHwf8o7_U	245	203	448
Shakira	pRpeEdMmmQ0	174	196	370

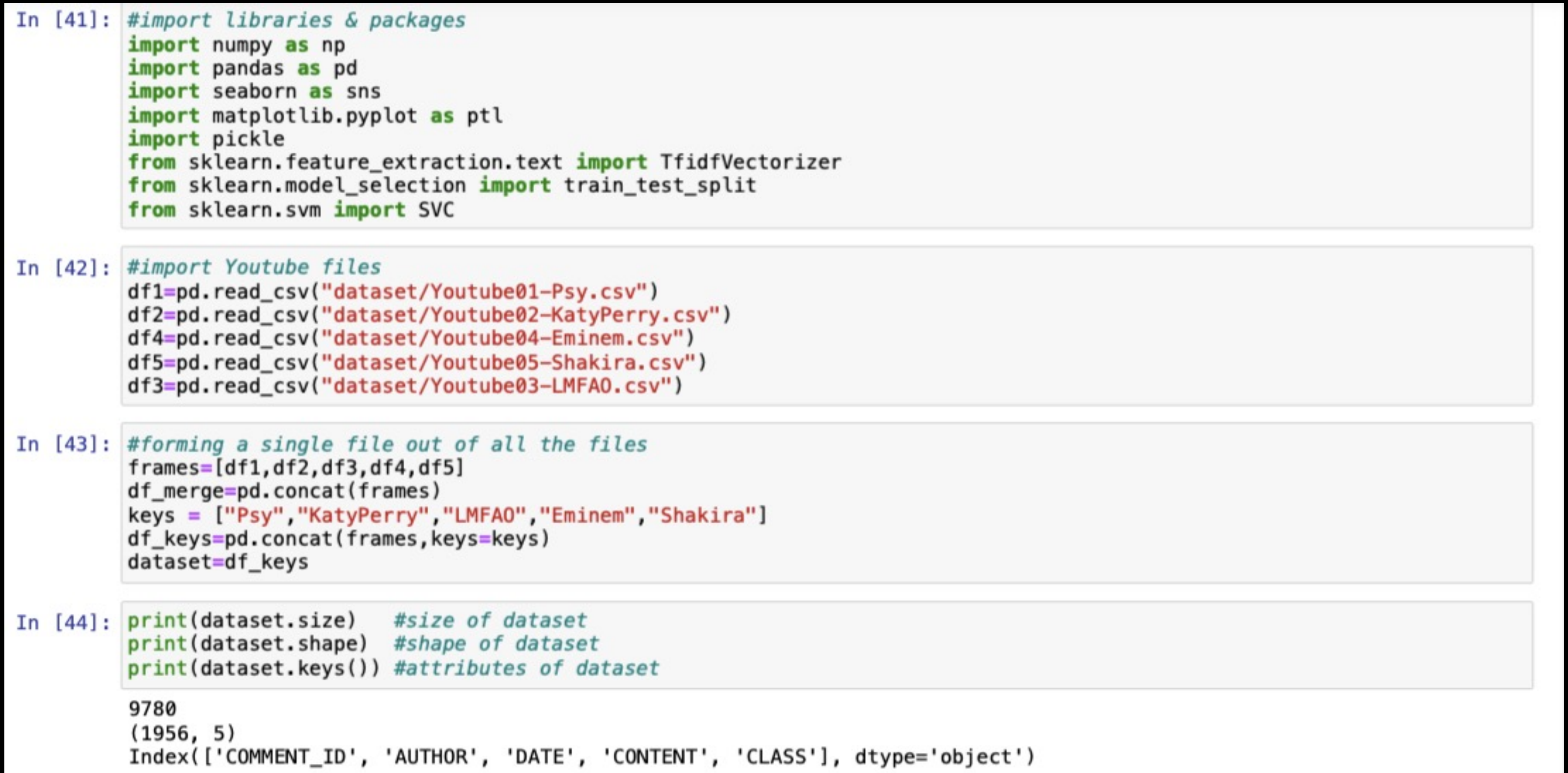
ATTRIBUTE INFORMATION

The collection is composed of one CSV file per dataset, where each line has the following attributes:

Attributes	Example
COMMENT_ID	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU
AUTHOR	Julius NM
DATE	2013-11-07 T 06:20:48
CONTENT	Huh, anyway check out this YouTube channel: kobyoshi02
Class	1 (Spam)

BUILDING A MODEL

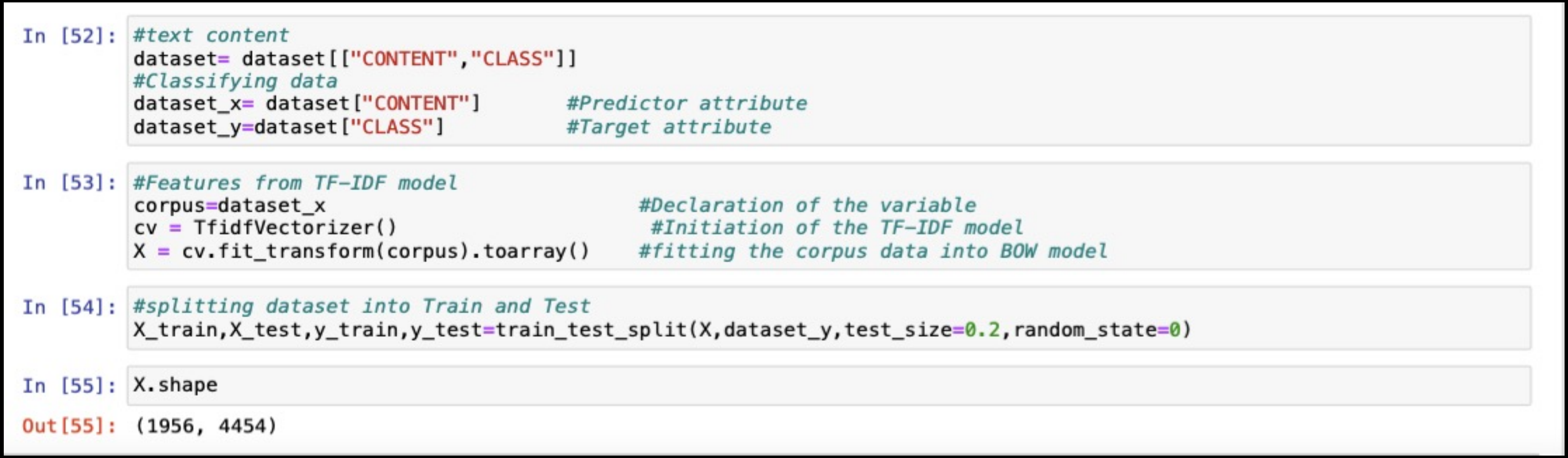
1)IMPORT REQUIRED LIBRARIES AND DATASET



```
In [41]: #import libraries & packages
         import numpy as np
         import pandas as pd
         import seaborn as sns
         import matplotlib.pyplot as ptl
         import pickle
         from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.model_selection import train_test_split
         from sklearn.svm import SVC

In [42]: #import Youtube files
         df1=pd.read_csv("dataset/Youtube01-Psy.csv")
         df2=pd.read_csv("dataset/Youtube02-KatyPerry.csv")
         df4=pd.read_csv("dataset/Youtube04-Eminem.csv")
         df5=pd.read_csv("dataset/Youtube05-Shakira.csv")
         df3=pd.read_csv("dataset/Youtube03-LMFAO.csv")

In [43]: #forming a single file out of all the files
         frames=[df1,df2,df3,df4,df5]
         df_merge=pd.concat(frames)
         keys = ["Psy","KatyPerry","LMFAO","Eminem","Shakira"]
         df_keys=pd.concat(frames,keys=keys)
         dataset=df_keys

In [44]: print(dataset.size)   #size of dataset
         print(dataset.shape)  #shape of dataset
         print(dataset.keys()) #attributes of dataset

         9780
         (1956, 5)
         Index(['COMMENT_ID', 'AUTHOR', 'DATE', 'CONTENT', 'CLASS'], dtype='object')
```

2)DATA PREPROCESSING

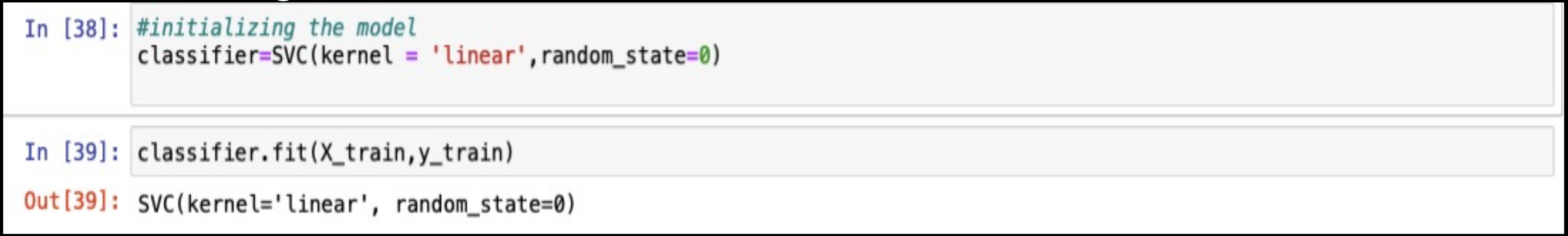
The dataset used here is split into 80% for the training set and the remaining 20% for the test set. We fed our dataset into a Term Frequency-Inverse document frequency (TF-IDF) vectorizer which transforms words into numerical features (numpy arrays) for training and testing



```
In [52]: #text content
         dataset= dataset[["CONTENT","CLASS"]]
         #Classifying data
         dataset_x= dataset["CONTENT"]       #Predictor attribute
         dataset_y=dataset["CLASS"]          #Target attribute

In [53]: #Features from TF-IDF model
         corpus=dataset_x                          #Declaration of the variable
         cv = TfidfVectorizer()                     #Initiation of the TF-IDF model
         X = cv.fit_transform(corpus).toarray()    #fitting the corpus data into BOW model

In [54]: #splitting dataset into Train and Test
         X_train,X_test,y_train,y_test=train_test_split(X,dataset_y,test_size=0.2,random_state=0)

In [55]: X.shape

Out[55]: (1956, 4454)
```

3)BUILD MODEL

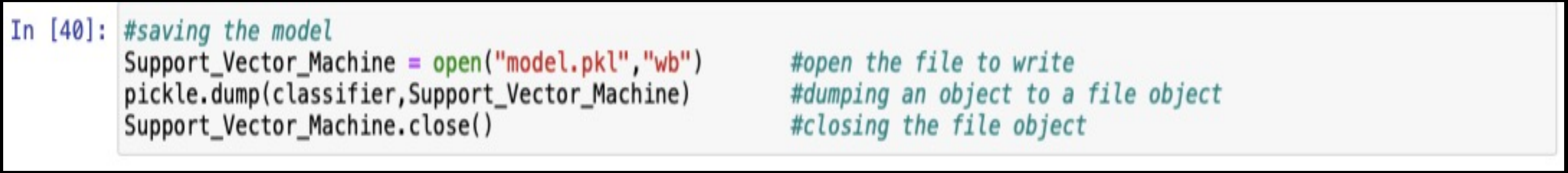
After data preprocessing, we implement machine learning model to classify the YouTube spam comments. For this purpose, we implement Support Vector Machine (SVM) using scikit-learn. After importing and initialize SVM model we fit into training dataset.



```
In [38]: #initializing the model
         classifier=SVC(kernel = 'linear',random_state=0)

In [39]: classifier.fit(X_train,y_train)

Out[39]: SVC(kernel='linear', random_state=0)
```

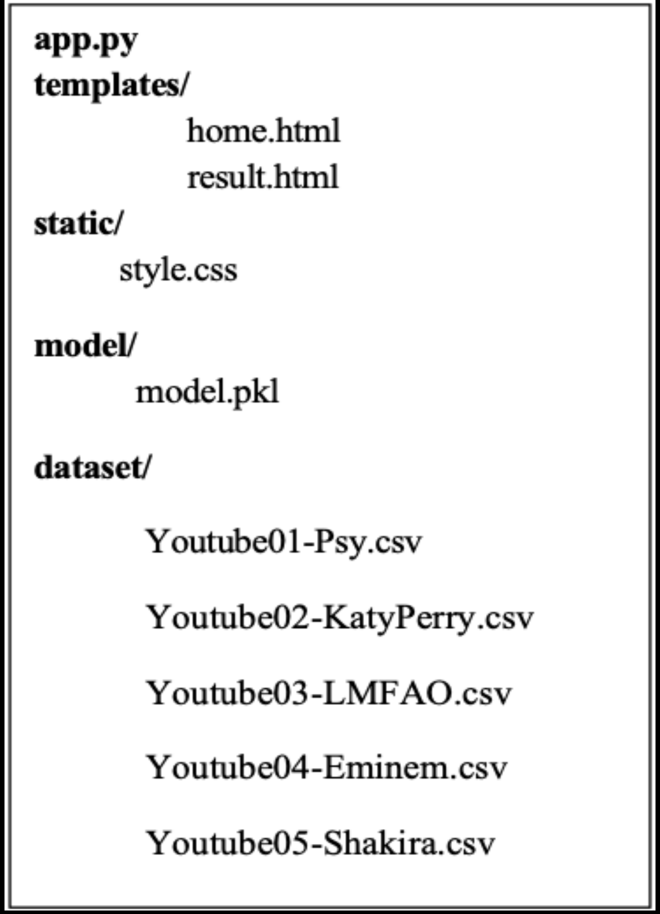
4)SAVE THE MODEL

After that we save our model using pickle



```
In [40]: #saving the model
         Support_Vector_Machine = open("model.pkl","wb")        #open the file to write
         pickle.dump(classifier,Support_Vector_Machine)         #dumping an object to a file object
         Support_Vector_Machine.close()                         #closing the file object
```

TURNING MODEL INTO WEB APPLICATION



```
app.py
templates/
            home.html
            result.html
static/
       style.css

model/
        model.pkl

dataset/

         Youtube01-Psy.csv

         Youtube02-KatyPerry.csv

         Youtube03-LMFAO.csv

         Youtube04-Eminem.csv

         Youtube05-Shakira.csv
```

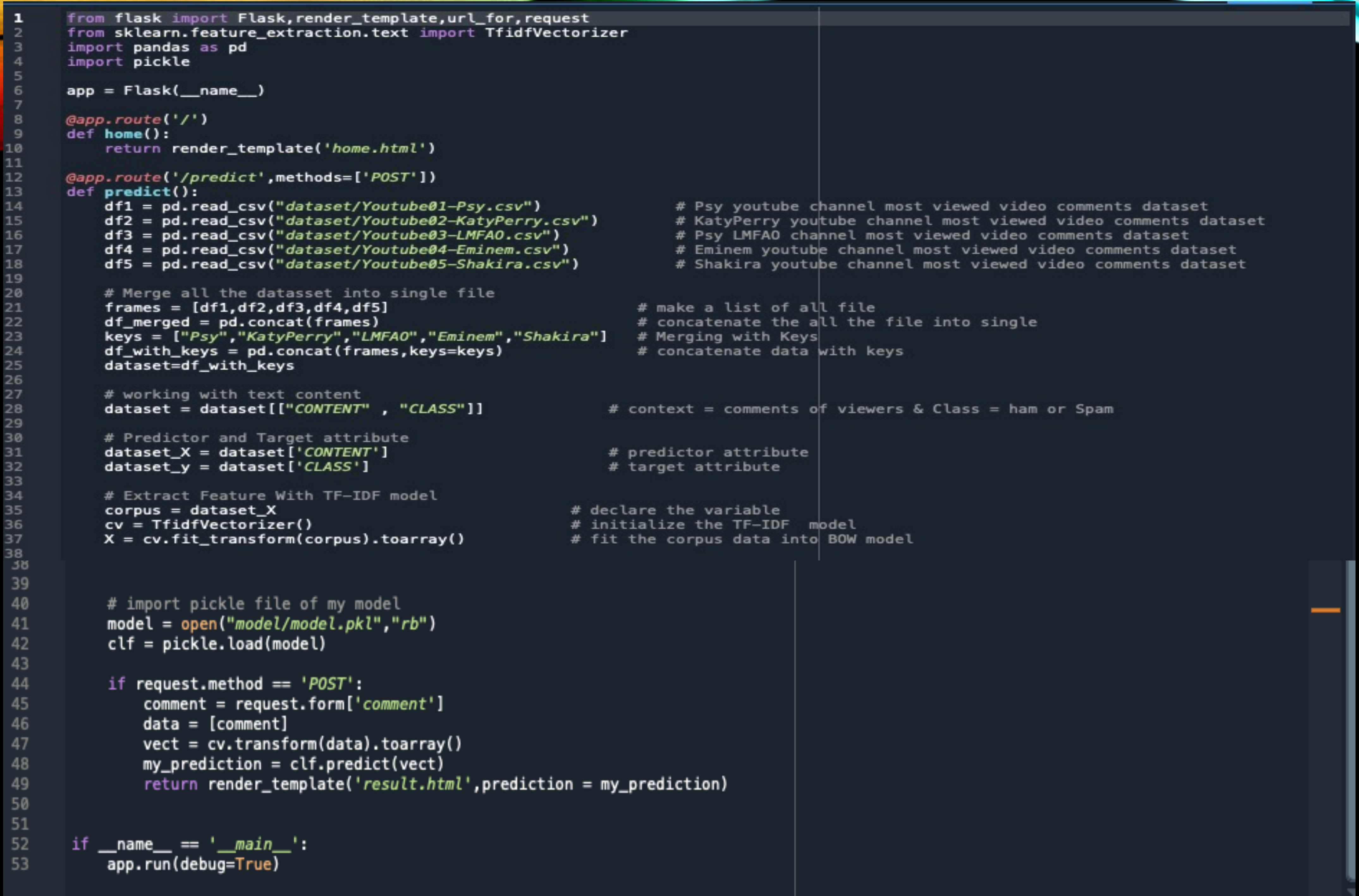
We develop a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of spam or ham(not spam).

First, we create a folder for this project called YouTube Spam Filtering, this is the directory tree inside the folder. We will explain each file.

The sub-directory templates are the directory in which Flask will look for static HTML files for rendering in the web browser, in our case, we have two HTML files: *home.html* and *result.html*.

App.py

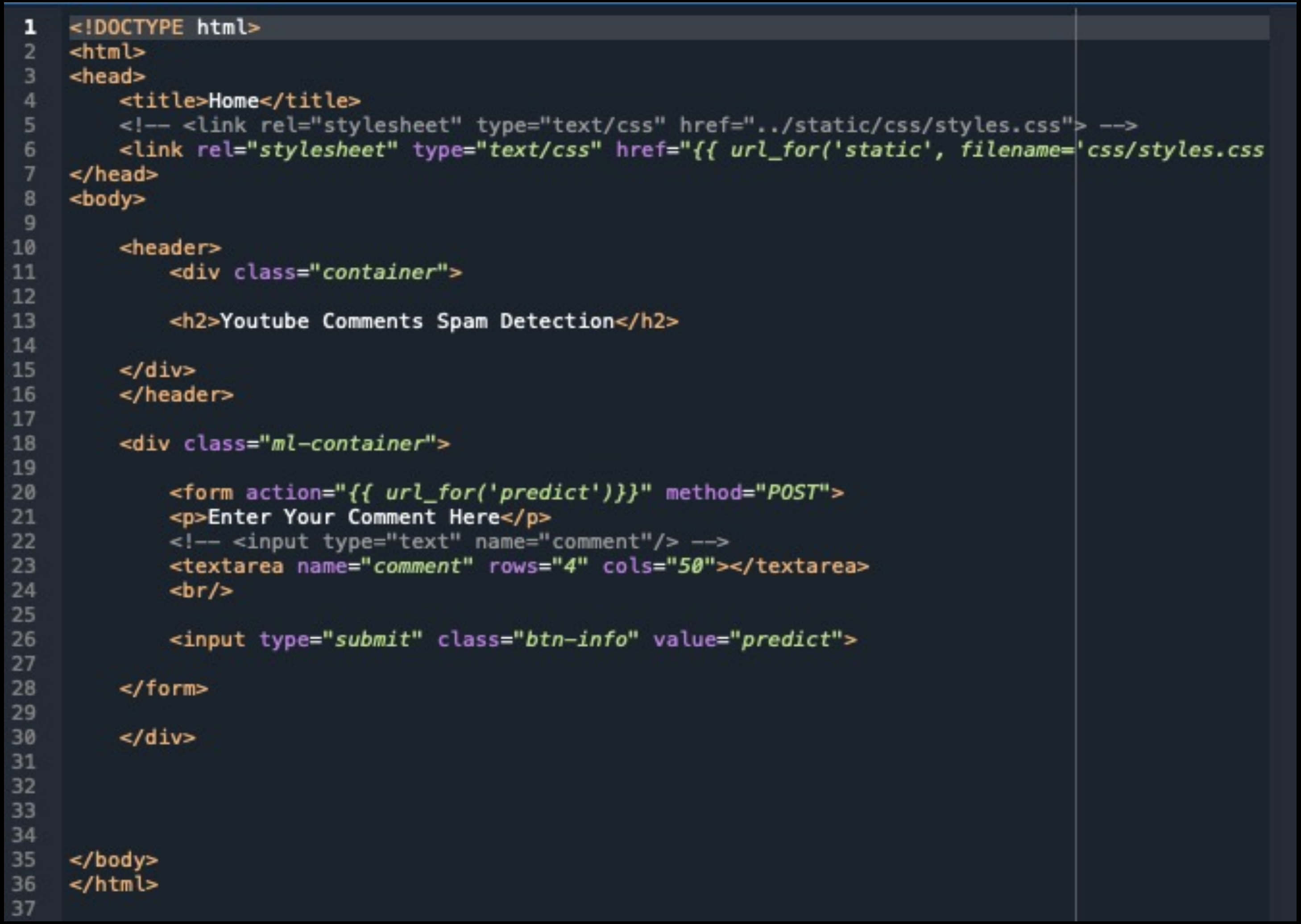
- We ran our application as a single module; thus we initialized a new Flask instance with the argument *__name__* to let Flask know that it can find the HTML template folder (*templates*) in the same directory where it is located.
- Next, we used the route decorator *(@app.route('/'))* to specify the URL that should trigger the execution of the home function.
- Our *home* function simply rendered the *home.html* HTML file, which is located in the *templates* folder.
- Inside the *predict* function, we access the spam data set, pre-process the text, and make predictions, then store the model. We access the new message entered by the user and use our model to make a prediction for its label.
- we used the *POST* method to transport the form data to the server in the message body. Finally, by setting the *debug=True* argument inside the app.run method, we further activated Flask's debugger.
- At last, we used the run function to only run the application on the serve when this script is directly executed by the Python interpreted, which we ensured using the if statement with *__name__=='__main__'*



```python
from flask import Flask,render_template,url_for,request
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import pickle

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/predict',methods=['POST'])
def predict():
    df1 = pd.read_csv("dataset/Youtube01-Psy.csv")                 # Psy youtube channel most viewed video comments dataset
    df2 = pd.read_csv("dataset/Youtube02-KatyPerry.csv")           # KatyPerry youtube channel most viewed video comments dataset
    df3 = pd.read_csv("dataset/Youtube03-LMFAO.csv")               # Psy LMFAO channel most viewed video comments dataset
    df4 = pd.read_csv("dataset/Youtube04-Eminem.csv")              # Eminem youtube channel most viewed video comments dataset
    df5 = pd.read_csv("dataset/Youtube05-Shakira.csv")             # Shakira youtube channel most viewed video comments dataset

    # Merge all the datasset into single file
    frames = [df1,df2,df3,df4,df5]                             # make a list of all file
    df_merged = pd.concat(frames)                              # concatenate the all the file into single
    keys = ["Psy","KatyPerry","LMFAO","Eminem","Shakira"]      # Merging with Keys
    df_with_keys = pd.concat(frames,keys=keys)                 # concatenate data with keys
    dataset=df_with_keys

    # working with text content
    dataset = dataset[["CONTENT" , "CLASS"]]                # context = comments of viewers & Class = ham or Spam

    # Predictor and Target attribute
    dataset_X = dataset['CONTENT']                          # predictor attribute
    dataset_y = dataset['CLASS']                            # target attribute

    # Extract Feature With TF-IDF model
    corpus = dataset_X                                  # declare the variable
    cv = TfidfVectorizer()                              # initialize the TF-IDF  model
    X = cv.fit_transform(corpus).toarray()              # fit the corpus data into BOW model



    # import pickle file of my model
    model = open("model/model.pkl","rb")
    clf = pickle.load(model)

    if request.method == 'POST':
        comment = request.form['comment']
        data = [comment]
        vect = cv.transform(data).toarray()
        my_prediction = clf.predict(vect)
        return render_template('result.html',prediction = my_prediction)


if __name__ == '__main__':
    app.run(debug=True)
```

home.html

The following are the contents of the *home.html* file that will render a text form where a user can enter a message.



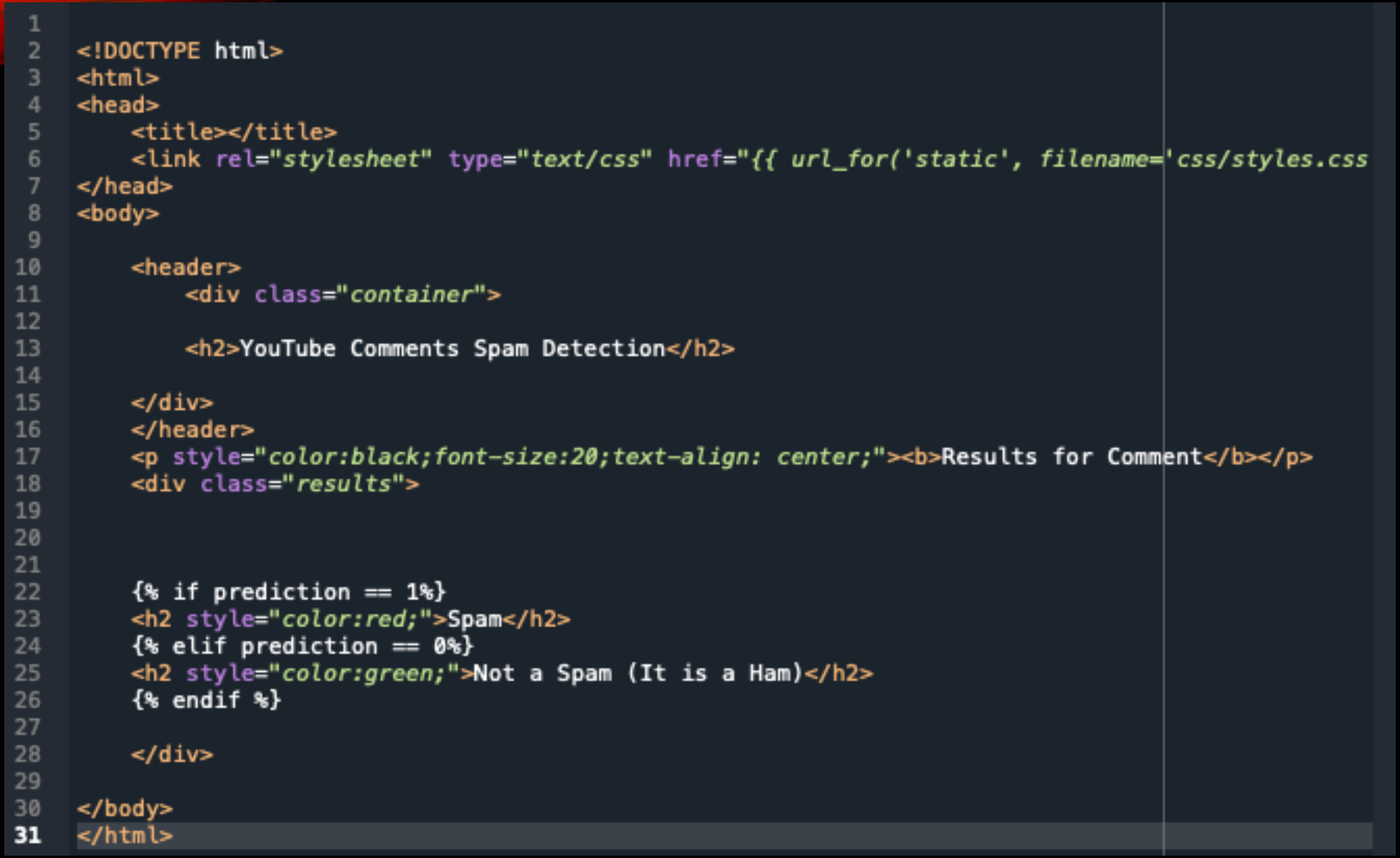
```
<!DOCTYPE html>
<html>
<head>
    <title>Home</title>
    <!-- <link rel="stylesheet" type="text/css" href="../static/css/styles.css"> -->
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/styles.css
</head>
<body>

    <header>
        <div class="container">

        <h2>Youtube Comments Spam Detection</h2>

    </div>
    </header>

    <div class="ml-container">

        <form action="{{ url_for('predict')}}" method="POST">
        <p>Enter Your Comment Here</p>
        <!-- <input type="text" name="comment"/> -->
        <textarea name="comment" rows="4" cols="50"></textarea>
        <br/>

        <input type="submit" class="btn-info" value="predict">

    </form>

    </div>




</body>
</html>

```

Style.css

In the header section of *home.html*, we loaded *styles.css* file. CSS is to determine how the look and feel of HTML documents. *styles.css* has to be saved in a sub-directory called *static*, which is the default directory where Flask looks for static files such as CSS.

Result.html

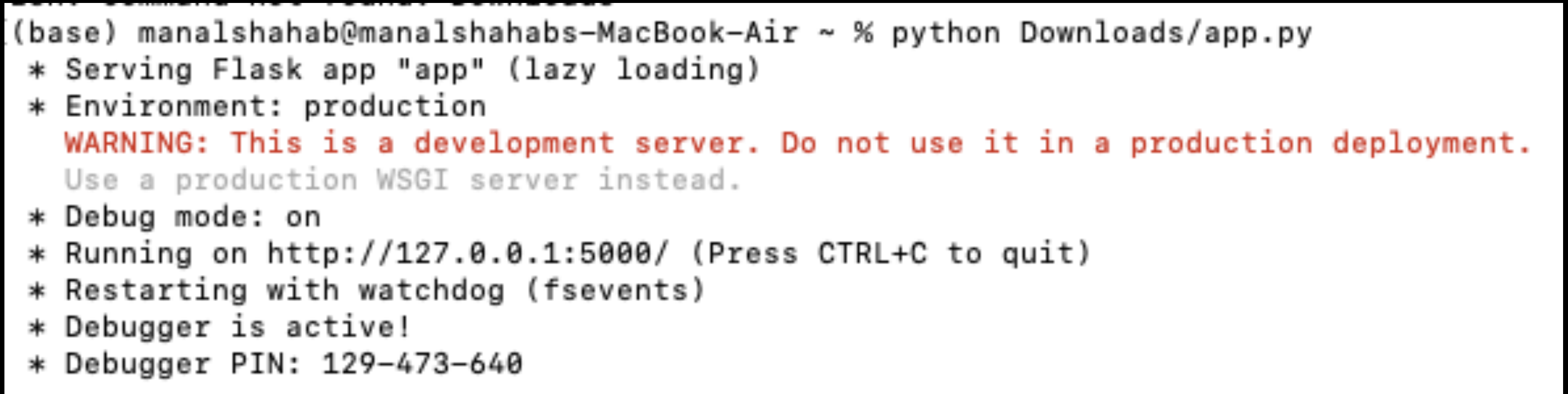
we create a result.html file that will be rendered via the *render_template ('result.html', prediction=my_prediction)*line return inside the *predict* function, which we defined in the *app.py* script to display the text that a user-submitted via the text field.
From *result.html* we can see that some code using syntax not normally found in HTML files:*{% if prediction ==1%},{% elif prediction == 0%},{% endif %}*This is Jinja syntax, and it is used to access the prediction returned from our HTTP request within the HTML file.



```

<!DOCTYPE html>
<html>
<head>
    <title></title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/styles.css
</head>
<body>

    <header>
        <div class="container">

        <h2>YouTube Comments Spam Detection</h2>

    </div>
    </header>
    <p style="color:black;font-size:20;text-align: center;"><b>Results for Comment</b></p>
    <div class="results">



    {% if prediction == 1%}
    <h2 style="color:red;">Spam</h2>
    {% elif prediction == 0%}
    <h2 style="color:green;">Not a Spam (It is a Ham)</h2>
    {% endif %}

    </div>

</body>
</html>
```

Running Procedure

Once we have done all of the above, we can start running the API by either double click *app.py*, or executing the command from the Terminal:



```
(base) manalshahab@manalshahabs-MacBook-Air ~ % python Downloads/app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with watchdog (fsevents)
 * Debugger is active!
 * Debugger PIN: 129-473-640
```

Now we could open a web browser and navigate to http://127.0.0.1:5000/ we should see a simple website with the content like so

YouTube Comments Spam Detection

Data Glacier

THANK YOU