Topic	WEB APP DEPLOYMENT: Digi Diary	
Class Description	**The student will learn to create a diary and save entries in it.**	
Class	**PRO C118**	
Class time	**45 mins**	
Goal	● To integrate the ML model for sentiment analysis by creating Python Web Application(Digi Diary). ● To save entries using the Digi Diary.	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone	
Class structure	**Warm-Up** **Teacher-led Activity 1** **Student-led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit	**jQuery** by John Resig **Flask** by Armin Ronacher and contributors	

WARM-UP SESSION - 10 mins	
Teacher Starts Slideshow **Slide 1 to 4** Refer to speaker notes and follow the instructions on each slide.	
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities. • Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide 5 to 10**	
Following are the session deliverables: • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Teacher Action	**Student Action**
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities.	**ESR**: Hi, thanks! Yes, I am excited about it! Click on the slide show tab and present the

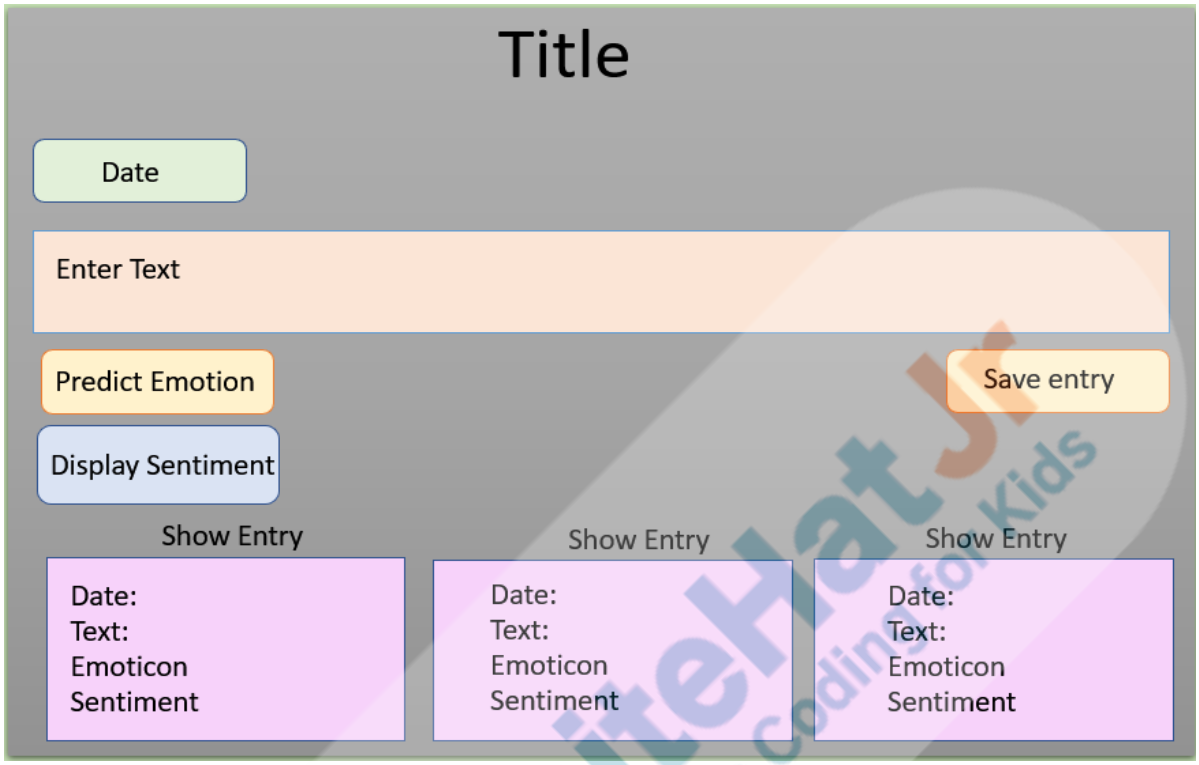
• Quizzes.	slides
Can you tell me what we learned in the previous class? Yes, we created an AJAX call and API to predict emotions. Do you remember how AJAX is useful for data transfer?	**ESR:** We learned to call an **API** using **AJAX** and **jQuery**. **ESR:** Yes, **AJAX** is used to transfer data to any element in an HTML page without reloading the page.
So,tell me how was your weekend? ***Note:*** *Depending upon how the student's weekend went, the teacher should give feedback.* E.g. Ohh great!! I can make out that it went very well. So, the reaction depends on how your weekends went. Let's create a diary to save these memories. Do you remember we had created a ML model to predict sentiment on the webpage? How will it be if we integrate the model to create a personal diary where you can keep a record of how your day went, your memorable moments? Also, we can make the diary predict the sentiment as well. Yes!! So let's learn how to add entries to our webpage. We can save and display the entries.	**ESR:** Varied **ESR:** Yes **ESR**: It'll be fun

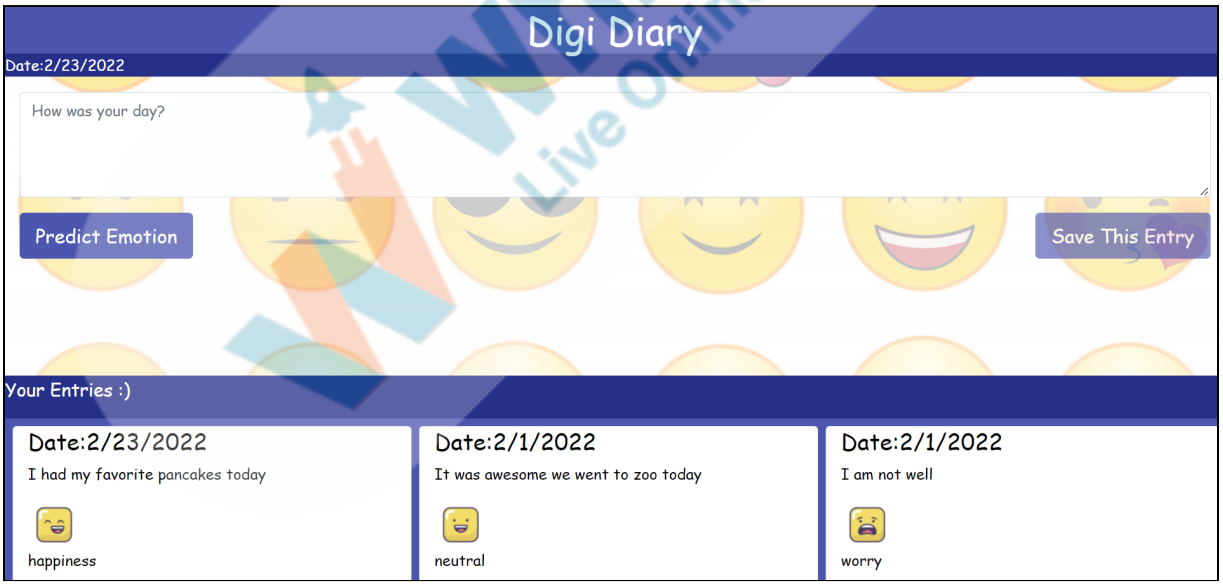
Teacher Ends Slideshow
TEACHER-LED ACTIVITY - 10 mins
Teacher Initiates Screen Share

ACTIVITY

- **Creating an API for saving the diary entry**
- **Creating AJAX call for transferring data from HTML page to Python file**
- **Deploy Machine Learning model in the HTML page to save the entry**

Teacher Action	Student Action
Note: *Open Teacher activity 1 for the boilerplate code. Create a folder and save all the files. Check the directory structure. It has all the required files.* We'll start with an **HTML** page in the **templates** folder. This page would have the elements in the following manner: 1. The name of the webpage '**Digi Diary**' is displayed. 2. A paragraph for displaying the **date.** 3. A text area for entering **text.** 4. A '**Predict Emotion**' button for sending the request 5. A '**Save This Entry**' button for saving the entry. 6. A paragraph for displaying **the predicted emotion** as a result. The display is kept as **'none'** to hide it. 7. An **image (emoticon)** will be shown according to the emotion. Initially, this is also hidden. 8. There are three columns created in a row for displaying the saved entries along with the emoticons and emotions.	





Let's check **index.html** now. The **id** of the elements will be used for creating **AJAX** call.

[Teacher Activity 2:](#) [Bootstrap Documentation](#)

Bootstrap is used to create the HTML page. It is a web development framework. Bootstrap is used for simplifying **the development of web pages**. The purpose of adding it to a web project is to apply Bootstrap's color, size, font and layout to our project.

Thus, it's links are added to the **index.html** file.

Note: *The previous class code is given as the boilerplate along with the HTML page.*

templates > index.html > html > body > div.bg

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <title>Digi-Diary</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>
```

```html
<body style="overflow-y: hidden;">

    <script src="/static/index.js"></script>

    <div class="bg"
        style="color: #FFF; background-image: linear-gradient(to bottom, rgba(255,255,255,0.6) 0%,
        rgba(255,255,255,0.9) 100%),url(./static/assets/bg/bg-emoji.png);
        background-repeat: no-repeat; background-size: cover; background-position: center center;">

        <!--App Name-->
        <div class=" container-fluid text-center"
            style="background-color: #4C56AF; font-family: cursive; min-height: 8vh;">
            <h1>Digital Diary</h1>
        </div>

        <!--Day and Date-->
        <div class="text-left mb-2"
            style="background-color: #263088; font-family: cursive; width: 100%; min-height: 3vh;">
            <p id="display_date">Date</p>
        </div>
```

```html
        <!--Input New Entry and Detect Emotion-->
        <div class="container-fluid" style=" min-height: 25vh;">
            <div style=" color: black; font-family: cursive; width: 100%;">
                <textarea class="form-control" id="text" name="text_input" rows="4" maxlength="200"
                    placeholder="How was your day?"></textarea>
            </div>

            <div class="row mt-3">
                <div class="col-6 text-left">
                    <button class="btn btn-lg" id="predict_button" type="submit" name="submit_b"
                    value="submit_p" style="color: white; background-color: #4C56AF;
                    font-family: cursive; ">Predict Emotion</button>
                </div>

                <div class="col-6 text-right ">
                    <button class="btn btn-lg" id="save_button" type="submit" name="submit_b"
                    value="submit_s" style="color: white;
                    background-color: #4C56AF; font-family: cursive; ">Save This Entry</button>
                </div>
            </div>

        </div>
```

In your entries columns,saved entries are appended using the variables **date, entry, URL for emoticons** and **emotions.**	

```html
<!--Predicted Emotion-->
<div class="container-fluid" id="prediction_container"
    style="color: black; font-family: cursive; font-size:20px; min-height: 20vh;">
    <p id="prediction" style="display: none"></p>
    <img id="emo_img_url" src="" width="55" min-height="50" alt="" style="display: none">
</div>

<!--Your Entries Columns-->
<div class="container-fluid text-left " style="min-height: 40vh ">
    <div class="row" style="background-color: #263088; color: #FFF; font-family: cursive;
    font-size: larger; min-height: 4vh "> <p>Your Entries :)</p>
    </div>
    <div class="row row-col-3" style="background-color: #4C56AF; color: #000;
    font-family: cursive; min-height: 35vh ">
        {% for entry in entries %}
        <div class="col mt-2 mb-2 ml-2 border rounded" style=" background-color: white; min-heigh
            <h4>{{entry["date"]}}</h4>
            <p>{{entry["entry"]}}</p>
            <img src="{{entry['emotion_url']}}" width="55" min-height="50" alt="">
            <p>{{entry["emotion"]}}</p>
        </div>
        {% endfor %}
    </div>
```

Then let's check the **index.js** file. It has the variable **display_date** created to display the date.

The **$(document).ready()** function is used to initialize the document.

The **$(#display_date).html()** function is used to select the **display_date** element of the HTML page and display the date. In addition to the Predict Emotion button, we have a save button which is disabled when text is not entered by the user.

static > index.js > $() callback > click() callback > error

```
var date = new Date()
let display_date= "Date:" + date.toLocaleDateString()

$(document).ready(function () {
    $("#display_date").html(display_date)
    $('#save_button').prop('disabled', true);
})
```

Now, we'll call the function for prediction which we had written in the last class.

When the **predict_button** is clicked the text value is given to the Flask API for prediction.

On success the results are displayed. Also, the **save_button** is enabled once text is entered and emotions are displayed.

```javascript
let predicted_emotion;
$(function () {
    $("#predict_button").click(function () {
        let input_data = {
            "text": $("#text").val()
        }
        $.ajax({
            type: 'POST',
            url: "/predict-emotion",
            data: JSON.stringify(input_data),
            dataType: "json",
            contentType: 'application/json',
            success: function (result) {
                $("#prediction").html(result.data.predicted_emotion)
                $("#emo_img_url").attr('src', result.data.predicted_emotion_img_url);
                $('#prediction').css("display", "");
                $('#emo_img_url').css("display", "");
                predicted_emotion = result.data.predicted_emotion
                $('#save_button').prop('disabled', false);
            },
            error: function (result) {
                alert(result.responseJSON.message)
            }
```

We have to write one more AJAX call and API to save the entry. I'll be writing the code to show the entry on the HTML page.

Let's check the **model_prediction** file where the function for prediction is written. Here, we'll be writing a code to display the entry.

```python
import pandas as pd
import numpy as np
import tensorflow
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
from datetime import datetime
# Make Code and URL Dictionary for different Emotions
emo_code_url = {
    "empty": [0, "./static/assets/emoticons/Empty.png"],
    "sadness": [1, "./static/assets/emoticons/Sadness.png"],
    "enthusiasm": [2, "./static/assets/emoticons/Enthusiastic.png"],
    "neutral": [3, "./static/assets/emoticons/Neutral.png"],
    "worry": [4, "./static/assets/emoticons/Worry.png"],
    "surprise": [5, "./static/assets/emoticons/Surprise.png"],
    "love": [6, "./static/assets/emoticons/Love.png"],
    "fun": [7, "./static/assets/emoticons/Fun.png"],
    "hate": [8, "./static/assets/emoticons/Hate.png"],
    "happiness": [9, "./static/assets/emoticons/Happiness.png"],
    "boredom": [10, "./static/assets/emoticons/Boredom.png"],
    "relief": [11, "./static/assets/emoticons/Relief.png"],
    "anger": [12, "./static/assets/emoticons/Anger.png"],
}
```

```python
train_data = pd.read_csv("./static/assets/data_files/tweet_emotions.csv")
training_sentences = []

for i in range(len(train_data)):
    sentence = train_data.loc[i, "content"]
    training_sentences.append(sentence)

model = load_model("./static/assets/model_file/Tweets_Text_Emotion.h5")

vocab_size = 40000
max_length = 100
trunc_type = "post"
padding_type = "post"
oov_tok = "<OOV>"

tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
tokenizer.fit_on_texts(training_sentences)
```

```python
def predict(text):
    predicted_emotion_img_url=""
    predicted_emotion=""

    if  text!="":
        sentence = []
        sentence.append(text)

        sequences = tokenizer.texts_to_sequences(sentence)

        padded = pad_sequences(
            sequences, maxlen=max_length, padding=padding_type, truncating=trunc_type
        )
        testing_padded = np.array(padded)

        predicted_class_label = np.argmax(model.predict(testing_padded), axis=-1)

        for key, value in emo_code_url.items():
            if value[0]==predicted_class_label:
                predicted_emotion_img_url=value[1]
                predicted_emotion=key
        return predicted_emotion, predicted_emotion_img_url
```

Let's write the code for displaying the entries on the HTML page.

1. Define a function **show_entry**.
2. In the folder **data_files**, a CSV file by name **data_entry** is created for saving the entries.
3. This file is read by using the **read_csv** function of **pandas.**
4. To read the entries a list is created. Whenever an entry is saved, it is placed at the bottom of the file. To access the latest entry the **iloc()** function is used with the '**-1**' index. This helps locate the latest entry first.
5. Since three entries are displayed at a time, three variables each for the date, text entry, emotion and url are created.
6. The values entered by the user are first saved in the CSV file and accessed by this function for displaying it on the HTML page.

```python
#Display entry
def show_entry():
    day_entry_list = pd.read_csv("./static/assets/data_files/data_entry.csv")

    day_entry_list = day_entry_list.iloc[::-1]

    date1 = (day_entry_list['date'].values[0])
    date2 =(day_entry_list['date'].values[1])
    date3 = (day_entry_list['date'].values[2])

    entry1 = day_entry_list['text'].values[0]
    entry2 = day_entry_list['text'].values[1]
    entry3 = day_entry_list['text'].values[2]

    emotion1 = day_entry_list["emotion"].values[0]
    emotion2 = day_entry_list["emotion"].values[1]
    emotion3 = day_entry_list["emotion"].values[2]

    emotion_url_1=""
    emotion_url_2=""
    emotion_url_3=""
```

7. As multiple emoticons are present, we need to display the right emotion. So, a **for** loop is used with the key and value of the **emo_code_url dictionary.**
8. The emotions are checked and then the emoticons are displayed. The URL of each emoticon is present at index number **1** of the **emo_code_url.** Thus, depending upon the emotion, emoticons are displayed using their URL.

```python
    for key, value in emo_code_url.items():
        if key==emotion1:
            emotion_url_1 = value[1]
        if key==emotion2:
            emotion_url_2 = value[1]
        if key==emotion3:
            emotion_url_3 = value[1]
```

9. These entries are then returned to the HTML page to be displayed.	

```
return [
    {
        "date": date1,
        "entry": entry1,
        "emotion": emotion1,
        "emotion_url": emotion_url_1
    },
    {
        "date": date2,
        "entry": entry2,
        "emotion": emotion2,
        "emotion_url": emotion_url_2
    },
    {
        "date": date3,
        "entry": entry3,
        "emotion": emotion3,
        "emotion_url": emotion_url_3
    }
```

So, this was the code for saving entries. Now it's your turn to write the AJAX call for sending data and a Flask API to save the entry. Are you excited to create the diary?	**ESR:** Yes.
Teacher Stops Screen Share	
So now it's your turn. Please share your screen with me.	
Teacher Starts Slideshow **Slide 11 to 13**	

<**Note**: Only Applicable for Classes with VA> Refer to speaker notes and follow the instructions on each slide.	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	**ESR:** Yes
Teacher Ends Slideshow	
STUDENT-LED ACTIVITY - 20 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Full Screen.**	
Student Initiates Screen Share	
ACTIVITY • **Creating an AJAX call to send the diary entries to Flask API** • **Writing an API to save diary entry**	
Teacher Action	**Student Action**
Note: *If time permits then the teacher can make the student write the code for saving entry. In that case use the link: Teacher Activity1 for boilerplate code.* ***Note:*** *Along with the teacher boilerplate code, teacher activity code is also given to the student as boilerplate code.* ***Note:*** *Guide the student to open Student Activity1 for boilerplate code. Download the folder and run the file to check the HTML page being displayed.*	

In the **index.js** file write the AJAX call to save the entry. The steps are as follows:

1. Entry is saved when the **Save This Entry** button is clicked. So write using the **click()** method of **jQuery.**
2. As soon as this button is clicked**, the date, text** and **emotions** are saved in the **save_data** variable.

```
$("#save_button").click(function () {
    save_data = {
        "date": display_date,
        "text": $("#text").val(),
        "emotion": predicted_emotion
    }
```

3. Now create an AJAX call. Define the type of request sent, url of the API.
4. This call will send the data in JSON format.
5. On successful saving of entry, the window is reloaded for entering the text again.
6. If any error occurs, an error message will be displayed.

```
    $.ajax({
        type: 'POST',
        url: "/save-entry",
        data: JSON.stringify(save_data),
        dataType: "json",
        contentType: 'application/json',
        success: function () {
            alert("Your entry has been saved successfully!")
            window.location.reload()
        },
        error: function (result) {
            alert(result.responseJSON.message)
        }
    });

});
})
```

Open **app.py** for writing the API.

1. Use the **route** function for redirecting the Flask to save the entry.
2. Inside the API we'll be first saving the data (date, text and emotions) in different variables.
3. All the entries are stored in a comma-separated manner using the variable **entry.**
4. The **file_handler** variable is used to open the CSV file for saving the entry,
5. The **open()** method is used to open the CSV file. **'a'** stands for the **append** method so that data can be appended in the file using the **write()** method.
6. The entries are then appended and saved in the file.
7. On success, the data is sent in **JSON** format.

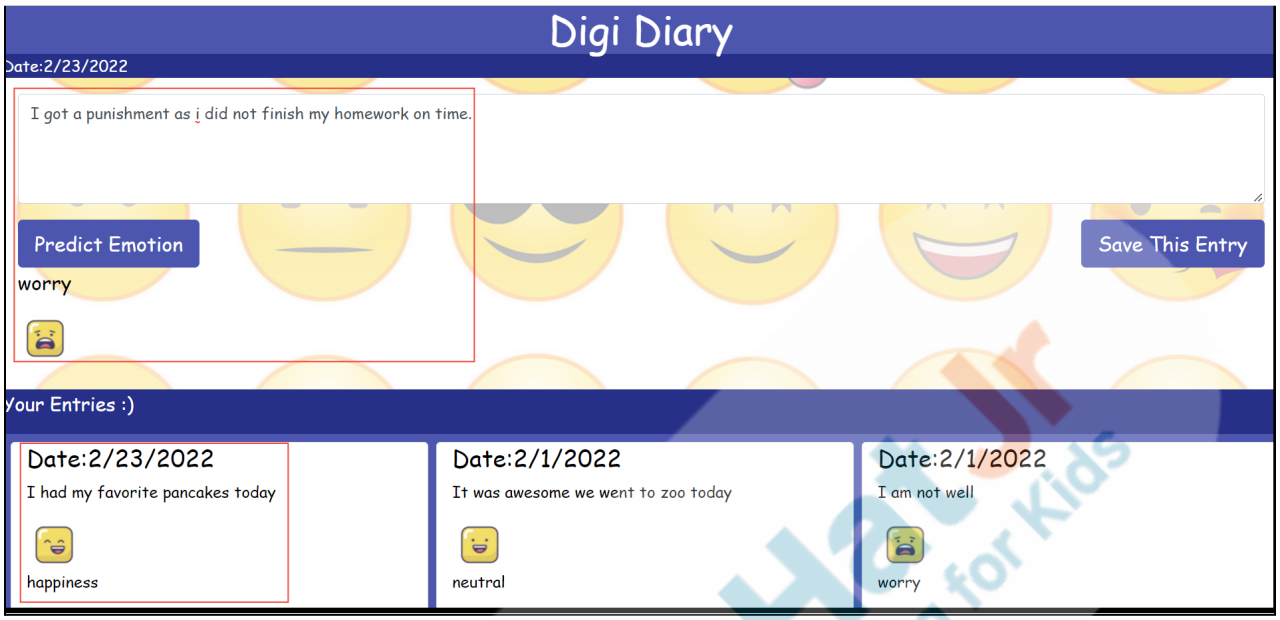
```python
# Save entry
@app.route("/save-entry", methods=["POST"])
def save_entry():
    date = request.json.get("date")
    save_text = request.json.get("text")
    emotion = request.json.get("emotion")

    entry = date + "," + save_text + "," + emotion+ ","
    file_handler = open('./static/assets/datafiles/data_entry.csv' , 'a')
    file_handler.write(entry + '\n')

    return jsonify("Success")


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

Now, you can run this file using the command prompt. Let's check if it can save the entry.



We have successfully created our diary that predicts emotions and saves the entry that can be displayed on the HTML page.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

Teacher Starts Slideshow
Slide 14 to 19
<**Note**: Only Applicable for Classes with VA>

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Continue WRAP-UP Session
Slide 20 to 25
<**Note**: Only Applicable for Classes with VA>

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	Student Action
You get "hats-off" for your excellent work! In the next class we'll learn about creating a chatbot.	*Make sure you have given at least 2 hats-off during the class for:*   

PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections
• **Teacher Clicks** ✖ End Class

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Teacher Boilerplate Code	https://github.com/procodingclass/PRO-C118-Teacher-Boilerplate-Code
Teacher Activity 2	Bootstrap Documentation	https://getbootstrap.com/docs/4.1/getting-started/introduction/
Teacher Activity 3	Reference Code	https://github.com/procodingclass/PRO-C118-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/411fd531-6e8a-4281-93b2-531777ba18c1.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C118-Project-Reference-Code
Teacher Reference 3	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/99220667-3f89-4ba9-8cef-d6a3c764e542.html
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/419a879b-bbfd-45bf-a37d-17cc8292fede.pdf
Teacher Reference 5	SIMPLIFYING JavaScript - BASICS OF jQuery	https://s3-whjr-curriculum-uploads.whjr.online/35abdd8f-3d70-4cef-a2

		c2-dc82053efd1b.pdf
Student Activity 1	Student Boilerplate Code	https://github.com/procodingclass/PRO-C118-Student-Boilerplate-Code