

FIT3164 - User Guide

Prepared by MDS20

Cheng Tze Pin 32870663

Lim Jun Yi 32625510

Prateek Tripathi 31835643

Table of Contents

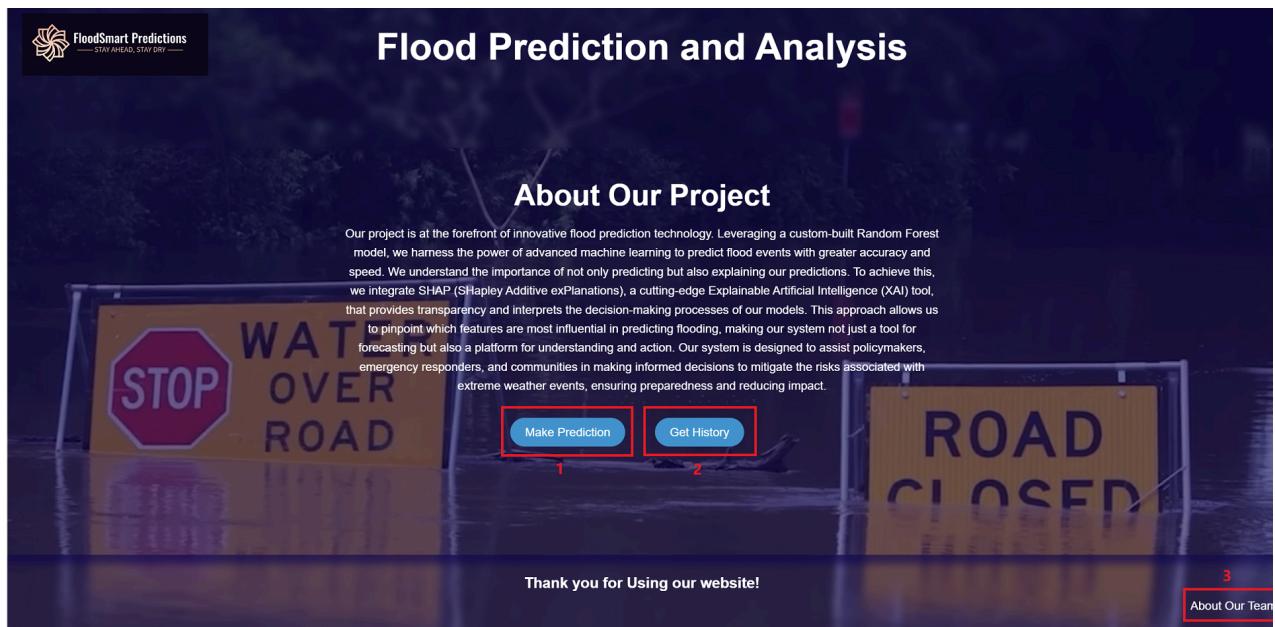
1. End User Guide	3
1.1 Home Page	4
Navigate around the menu	4
1.2 About Our Team Page	5
1.3 Prediction & Analysis Page	6
1.4 Prediction History Page	10
2. Technical Guide	12
2.1 Accessing the Application	12
2.1.1 Webpage	14
2.2 Accessing the Weather API	18
2.3 Accessing the MySQL Database	21
2.4 Accessing the Application APIs	24
2.4.1 Prediction Model API	24
2.5 Running the Application as a Whole	28

1. End User Guide

Welcome to the FloodSmart Predictions User Guide! This guide is crafted to assist you in navigating and utilising our Flood Prediction and Analysis web application effectively. Whether you're a policymaker, emergency responder, or a concerned citizen, this guide will provide detailed instructions on accessing and interpreting the vital information our tool offers. Our application leverages advanced machine learning technology to provide accurate flood event predictions, helping you make informed decisions and prepare for potential flood risks. Follow the instructions in this guide to learn how to make the most out of our innovative prediction platform.

1.1 Home Page

Navigate around the menu



(Please note that due to a formatting issue with the website, when zooming in/out, there will be website elements formatting bug. For a better user experience, keep the website's zoom at between 75% - 125%).)

Once you launch our Flood Prediction webpage, You will see a Home Page. This is the landing page for our prediction web application, in this page, you will see there are 3 interactive buttons.

- **1. "Make Predictions"**

This button will lead you to the prediction sub-page, where flood prediction results are being shown.

- **2. "Get History"**

This button will lead you to a page where you can get the prediction history of all previous prediction attempts.

- **3. “About Our Team”**

This link will lead you to an About page where the description of the developers and contributors of this flood prediction application is located.

1.2 About Our Team Page



On this page, you'll meet the dedicated team members behind our Flood Prediction and Analysis application. This section provides a brief overview of each individual contributing to the development and success of our application.

- **1. Lim Jun Yi (model and XAI)**
- **2. Cheng Tze Pin (API and database)**
- **3. Prateek Tripathi (Web UI)**

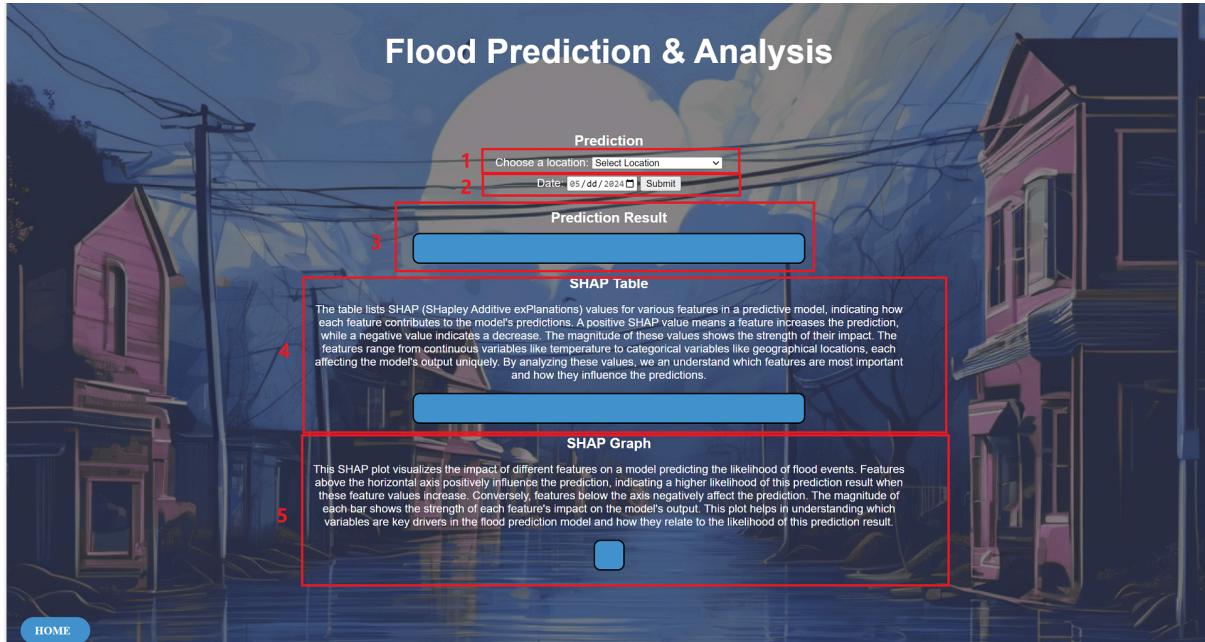
To return to the Home page, click on the “HOME” button at the bottom left corner of the website.



To exit the Home page, click on the “x” mark on your browser.

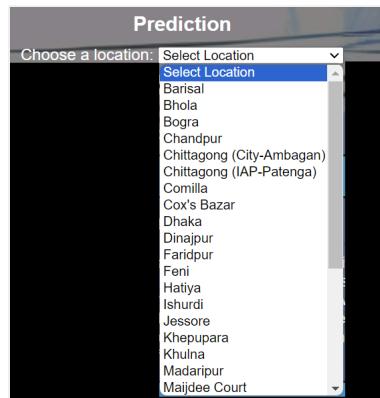
The detailed guides on the Prediction page and History page will be shown in the next section.

1.3 Prediction & Analysis Page



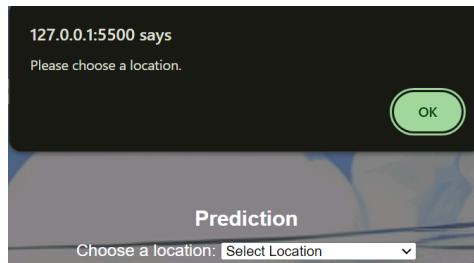
Upon entering the prediction page, you will see there are some attributes on the websites with descriptions, don't worry if you're confused, we will walk you through them.

- **1. “Choose a location”**



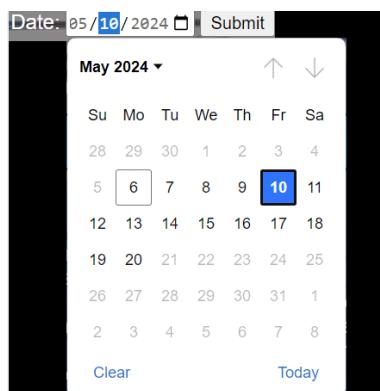
For this attribute, you will need to choose the location/city in Bangladesh that you want to make a flood prediction for. For example, if you wanted to predict the flood status for “Dhaka”, you would select “Dhaka” in the location attribute.

(note that due to dataset and API limitations, not all cities in Bangladesh are included in this prediction software and hence no prediction for those locations is available.)



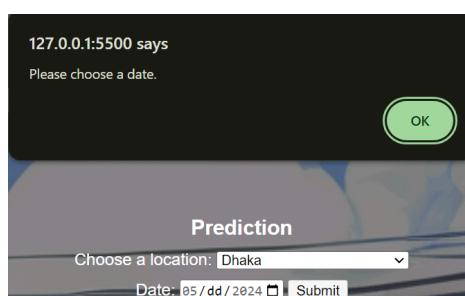
If you leave the location attribute blank, an error message will appear. A location must be chosen for the prediction to proceed.

- 2. “Date”



For this attribute, you will need to select the date that you want to predict, that is, the flood prediction model will try to predict whether it will have a flood or not during that particular date with a confidence score(ranging from 0-1). Let's say you wanted to predict a flood event for “Dhaka” on **10 May 2024**, then you would click the corresponding date on the date selector.

(note that the prediction date can only be today or up to the next 2 weeks, this is due to that Weather API can only supply the next 2 weeks of forecast data.)



If you leave the date attribute blank, an error message will appear. A date must be chosen for the prediction to proceed.

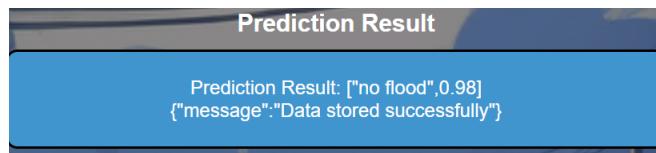
- 3. Prediction Result

Assume you predicted a flood event in “Dhaka” on **10 May 2024** and clicked the **Submit button**.



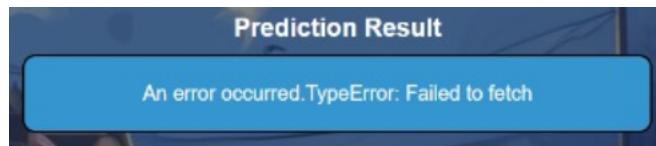
A screenshot of a web-based prediction form titled "Prediction". It has a dropdown menu labeled "Choose a location: Dhaka". Below it is a date input field showing "Date: 05/10/2024" and a "Submit" button. To the right of the "Submit" button is a red rectangular box containing the word "Click".

The below will be the Result output:

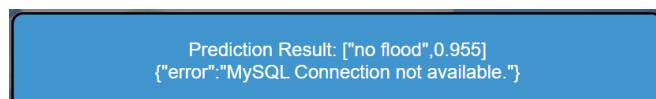


The prediction result will return 2 lines of string message. The first line of string indicates the result of a flood event. In this case, the flood prediction result for “Dhaka” on **10 May 2024** is there will be no flood with a confidence of 0.98.

The second line of string indicates if the prediction result has been successfully stored in the database. In this example, our result for “Dhaka” on **10 May 2024** has been stored in the database and later can be retrieved in the History Page.



An error message will occur if the Administrator has not set up the prediction API. “Failed to fetch”, showing that there isn’t a connection with the prediction model and hence cannot get the prediction result. To deal with this situation, please contact the software administrator or run the “PredictionModelAPI.py” file.



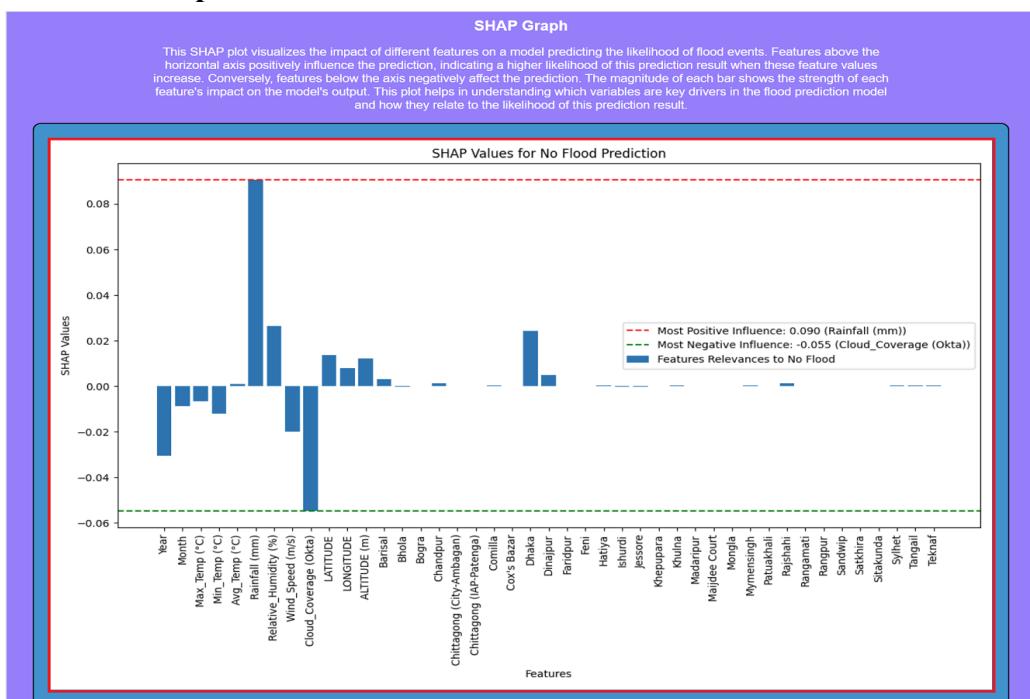
An error message will occur if the Administrator has not set up the database API. “MySQL connection not available”, showing that there isn’t a connection with the database and hence cannot store the result. To deal with this situation, please contact the software administrator or run the “DatabaseAPI.py” file.

- 4. SHAP Table

SHAP Table	
The table lists SHAP (SHapley Additive exPlanations) values for various features in a predictive model, indicating how each feature contributes to the model's predictions. A positive SHAP value means a feature increases the prediction, while a negative value indicates a decrease. The magnitude of these values shows the strength of their impact. The features range from continuous variables like temperature to categorical variables like geographical locations, each affecting the model's output uniquely. By analyzing these values, we understand which features are most important and how they influence the predictions.	
Attributes	SHAP Value
Year	-0.016236522929707103
Month	-0.008546525373901102
Max_Temp (°C)	-0.006632776221846582
Min_Temp (°C)	-0.006674046523748097
Avg_Temp (°C)	0.008575996314711629
Rainfall (mm)	0.0766234833458512
Relative_Humidity (%)	0.029178036413816322
Wind_Speed (m/s)	-0.013706245260891865
Cloud_Coverage (Okta)	0.04188896031343777
LATITUDE	0.010434610114749455
LONGITUDE	0.004419038563636597
ALTITUDE (m)	0.007633715511059567
Bansal	0.0019384699416950982
Bhola	-0.00017951955402921335
Bogra	9.722803070655378e-05
Chandpur	0.0007205209790680235
Chittagong (City-Ambagan)	4.021820974687047e-06
Chittagong (AP-Patenga)	1.9599457910309012e-05
Comilla	0.00043782289388175115
Cox's Bazar	-4.648134608603386e-06
Dhaka	0.021164971178310158
Dinajpur	0.0030137320896363557
Faridpur	5.888970737794372e-05
Feni	0.00010509666597300947
Hatiya	5.8827922267800335e-05
Ishurdi	-0.0006459670216922388
Jessore	-7.363463621240368e-05
Khepupara	-2.047092207495201e-05
Khulna	0.00014902545795854357
Madaripur	8.667894610601779e-05
Majidee Court	0.00023695182938513693
Monga	0.00014315189446605716
Mymensingh	0.000379717625687097
Patuakhali	0.0002337504332504126
Rajshahi	0.001632309923639821
Rangamati	2.1402232008821214e-05
Rangpur	-3.873978917300186e-05
Sandwip	0.00013755056591710824
Satkhira	5.646918851198857e-05
Sitakunda	0.0001072687585972588
Sylhet	0.00044301965473835924
Tangail	0.00013405284281505024
Teknaf	0.0005716966692074416

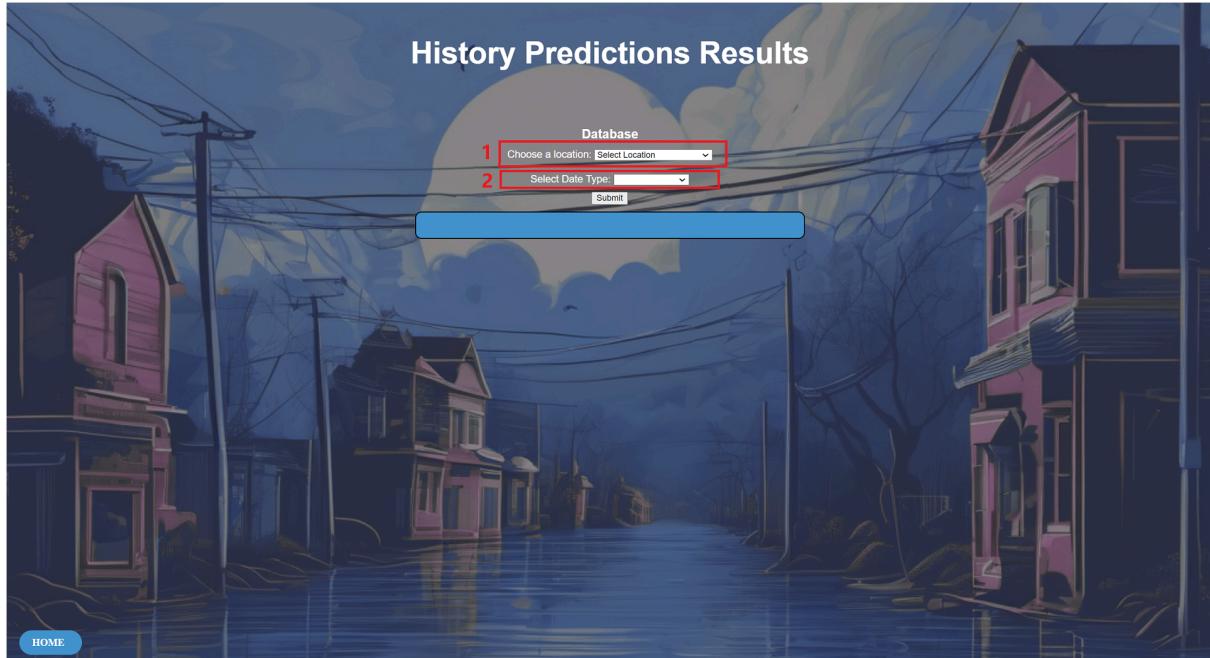
This should be the SHAP(SHapley Additive exPlanations)/XAI table output by our prediction model for a flood event prediction for “Dhaka” on **“10 May 2024”**. The SHAP table will tell you the information about how each attribute contributed to the prediction result, a positive value means positive influence and vice versa, a detailed description of how to interpret this table is shown on the website under “SHAP Table ”.

- 5. SHAP Graph



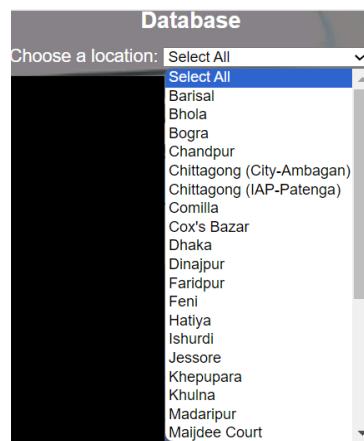
The red circled plot will be the example SHAP values plot for the flood prediction of “Dhaka” on **10 May 2024**. This plot visualised the SHAP table above to provide a better & easier understanding of the different attribute’s SHAP values. A detailed description of how to interpret the plot is shown on the website under “SHAP Graph”.

1.4 Prediction History Page



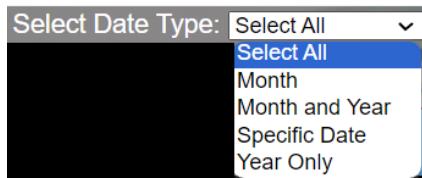
Upon entering the prediction history page, there will be mainly 2 attributes that control how you access the historical prediction data.

- 1. **“Choose a location”**

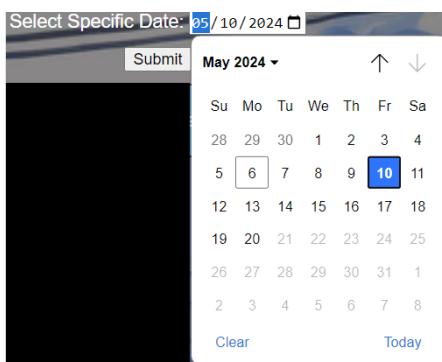


For this attribute, you can choose which location you want to filter the historical prediction data on. You can either leave this field on “Select All” and all the locations will be shown or choose a specific location. For example, we choose “Dhaka”.

- 2. “Select Date Type”



For this attribute, you can choose the date format type that you want to filter the historical prediction data on. You can either leave this field on “Select All” and all the data in the database for the selected city will be shown or choose a specific date format. For example, choosing “Specific Date”:



A calendar date selector will pop out after choosing the “Specific Date” format. Here you can filter the prediction result based on a specific date you like. For example, we want to see all predictions made for “Dhaka” on **10 May 2024**.

City	Year	Month	Day	Result	Probability	Prediction Made
Dhaka	2024	5	10	"no flood"	0.98	2024-05-06 17:41:46

After clicking submit, we can see the historic result for “Dhaka” on **10 May 2024** that we did earlier.

2. Technical Guide

2.1 Accessing the Application

IMPORTANT NOTE!!!

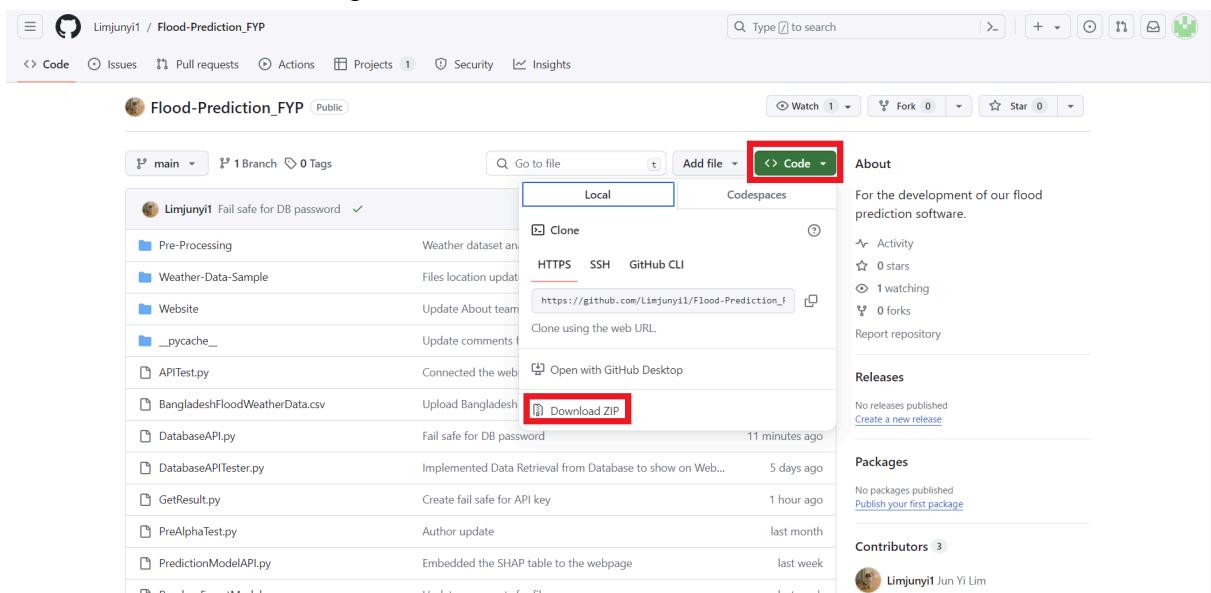
You need to install the libraries used in the files if you do not have them using the following command by replacing the LIBRARY_NAME with the actual library:

`pip install LIBRARY_NAME`

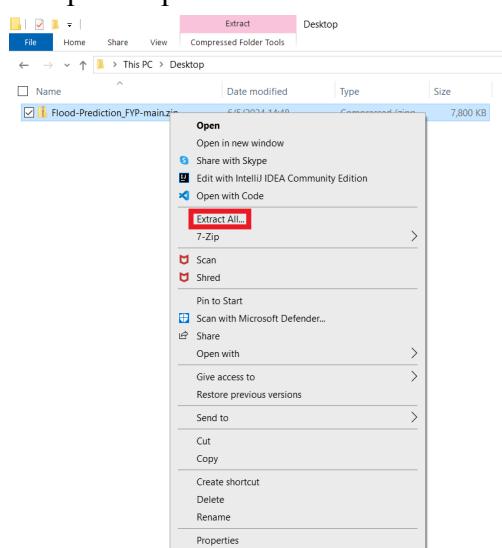
If you wish to start the application without knowing the codes, please jump to [2.5 Running the Application as a Whole](#)

To access the Application's source code, the administrator should follow the steps below:

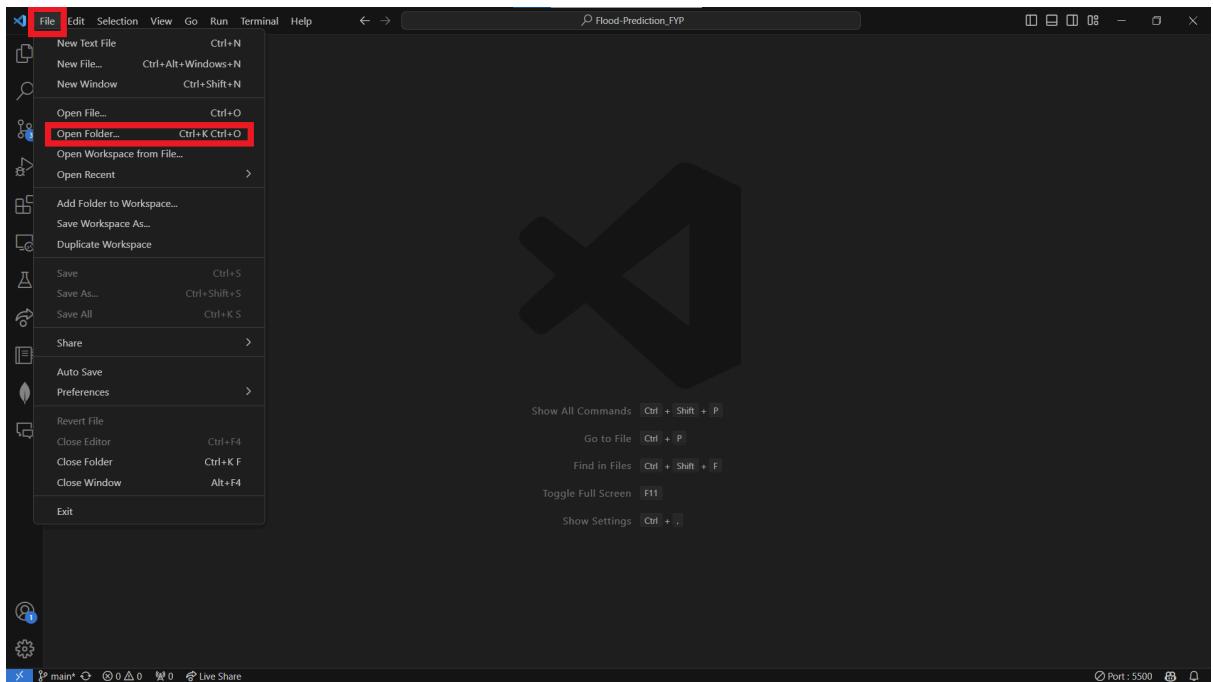
1. Go to GitHub and access our repository via https://github.com/Limjuniyi1/Flood-Prediction_FYP.
2. Download the codes as a zip folder



3. Unzip the zip folder

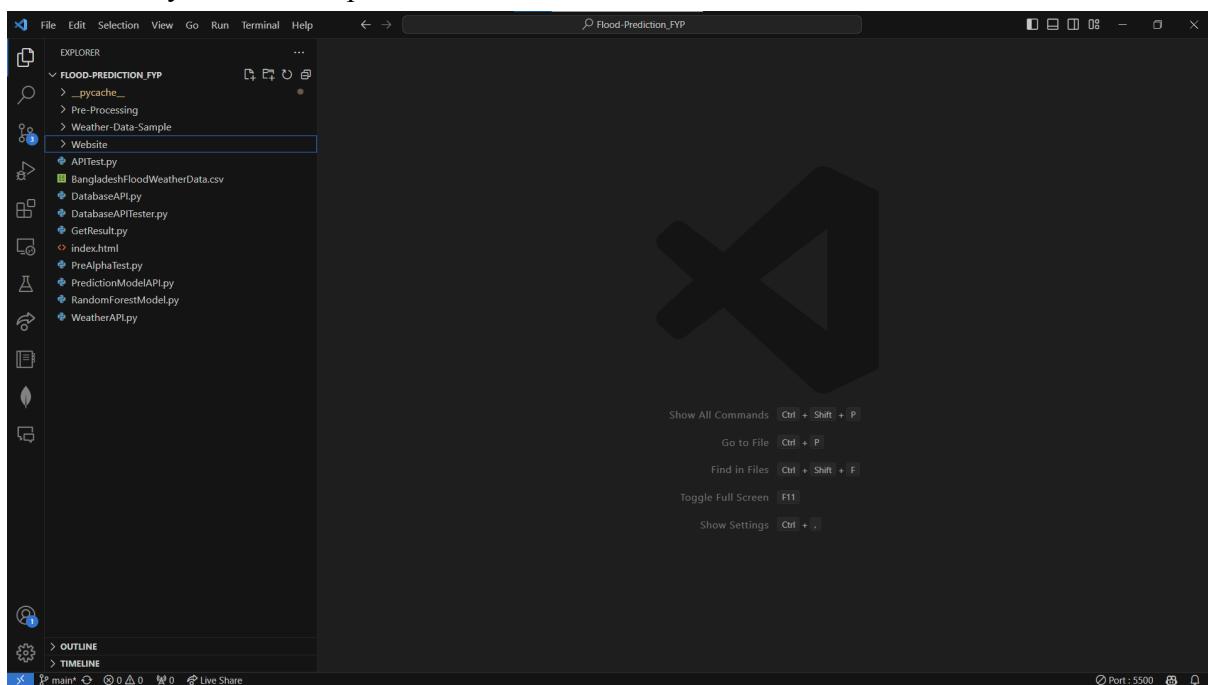


- Open the folder in an IDE of your choice e.g. Visual Studio Code



Need an IDE? Follow the link to [Download Visual Studio Code - Mac, Linux, Windows](#)

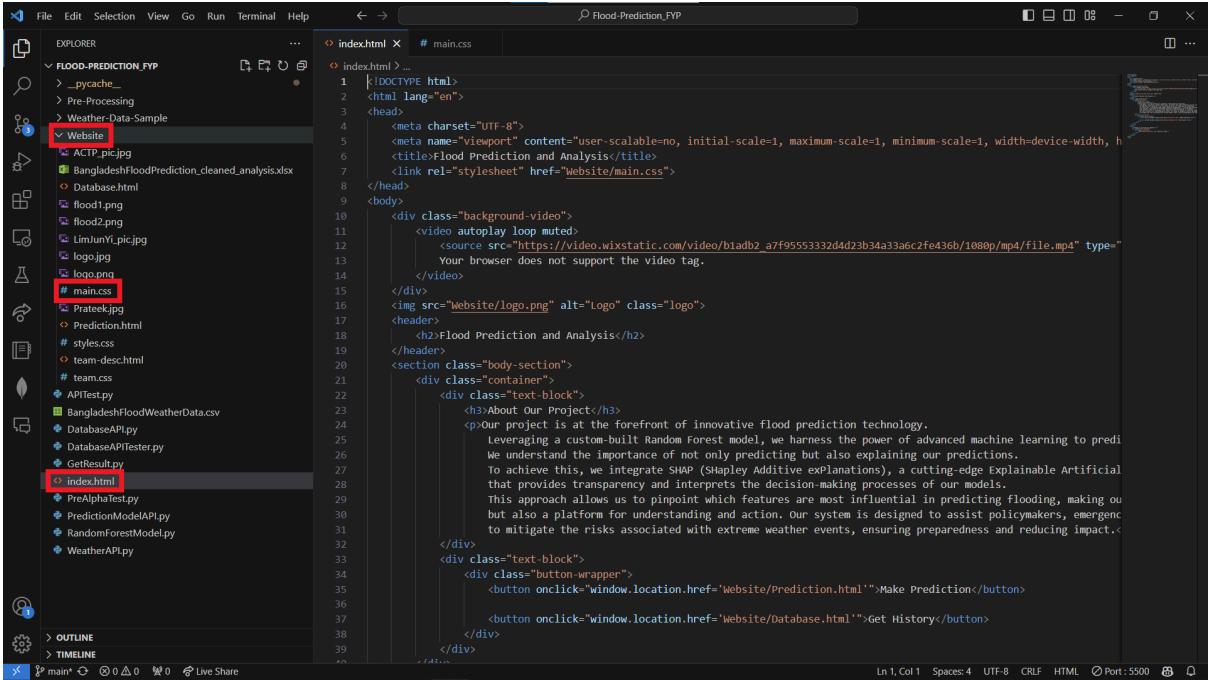
This is what you should expect!



2.1.1 Webpage

As shown in the End User Guide previously, there will be 4 web pages for the application:

1. Home Page - The source code is stored in a html file called index.html. A CSS file named main.css is used to style the webpage.

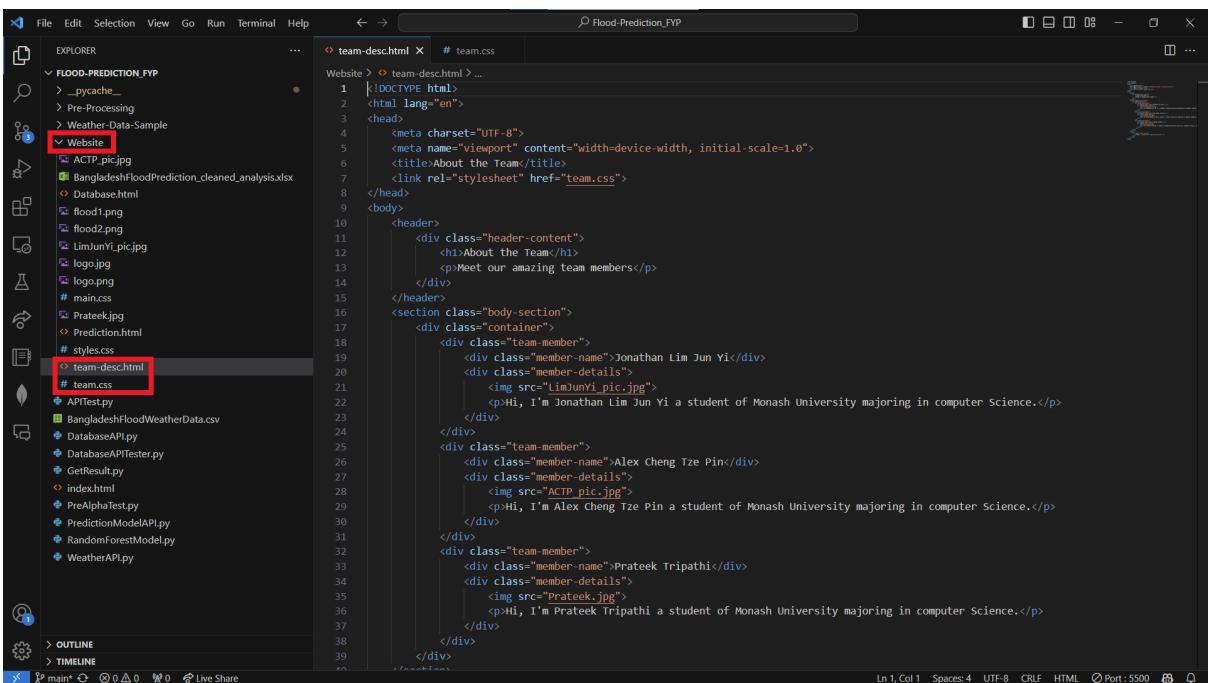


The screenshot shows the VS Code interface with the 'index.html' file open in the editor. The file content is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width, height=device-height, shrink-to-fit=no">
    <title>Flood Prediction and Analysis</title>
    <link rel="stylesheet" href="Website/main.css">
</head>
<body>
    <div class="background-video">
        <video autoplay loop muted>
            <source src="https://video.wixstatic.com/video/b1adb2_a7f955333d4d23b34a33a6c2fe436b/1000p/mp4/file.mp4" type="video/mp4">
            Your browser does not support the video tag.
        </video>
    </div>
    
    <header>
        <h2>Flood Prediction and Analysis</h2>
    </header>
    <section class="body-section">
        <div class="container">
            <div class="text-block">
                <h3>About Our Project</h3>
                <p>Our project is at the forefront of innovative flood prediction technology. Leveraging a custom-built Random Forest model, we harness the power of advanced machine learning to predict flooding. We understand the importance of not only predicting but also explaining our predictions. To achieve this, we integrate SHAP (SHapley Additive Explanations), a cutting-edge Explainable Artificial Intelligence technique that provides transparency and interprets the decision-making processes of our models. This approach allows us to pinpoint which features are most influential in predicting flooding, making our system not only a powerful tool for prediction but also a platform for understanding and action. Our system is designed to assist policymakers, emergency management agencies, and the general public in mitigating the risks associated with extreme weather events, ensuring preparedness and reducing impact.</p>
            </div>
            <div class="text-block">
                <div class="button-wrapper">
                    <button onclick="window.location.href='Website/Prediction.html'">Make Prediction</button>
                    <br>
                    <button onclick="window.location.href='Website/Database.html'">Get History</button>
                </div>
            </div>
        </div>
    </section>
</body>
</html>
```

The 'EXPLORER' sidebar on the left shows files like 'index.html', 'main.css', 'Prateek.jpg', 'Prediction.html', 'team-desc.html', and 'team.css'. The status bar at the bottom indicates 'Ln 1, Col 1' and 'Port: 5500'.

2. Team Description Page - The source code is stored in a html file called team-desc.html. A CSS file named team.css is used to style the webpage.

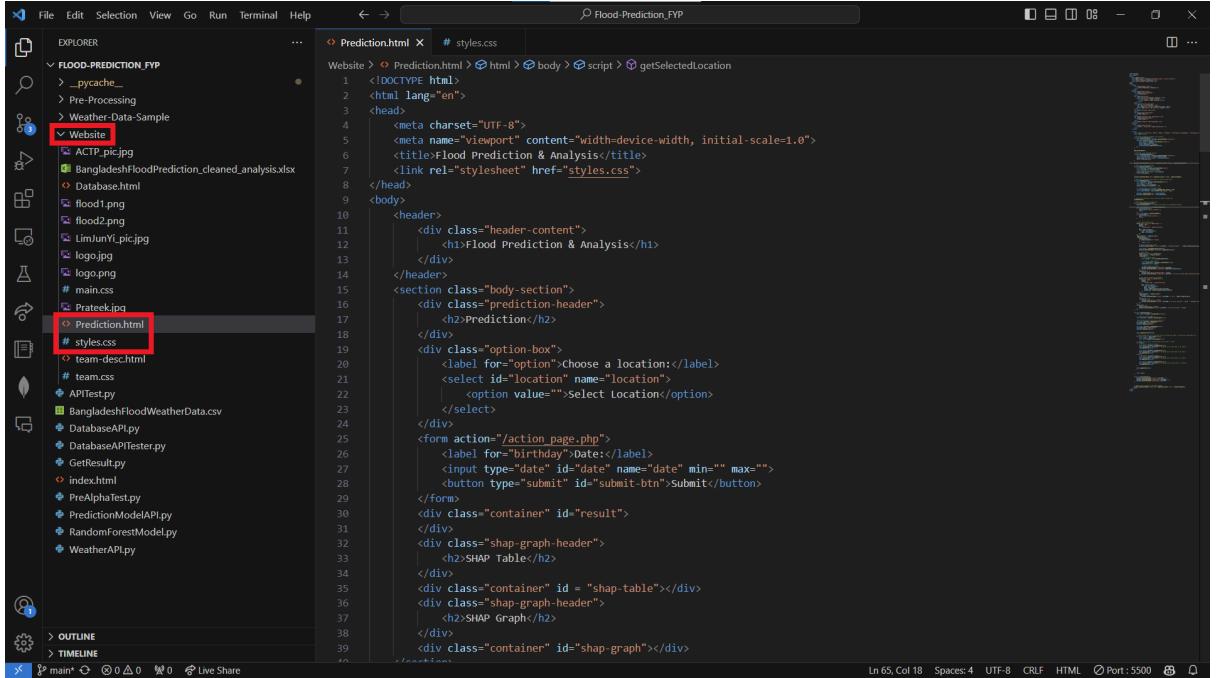


The screenshot shows the VS Code interface with the 'team-desc.html' file open in the editor. The file content is as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About the Team</title>
    <link rel="stylesheet" href="team.css">
</head>
<body>
    <header>
        <div class="header-content">
            <h1>About the Team</h1>
            <p>Meet our amazing team members</p>
        </div>
    </header>
    <section class="body-section">
        <div class="container">
            <div class="team-member">
                <div class="member-name">Jonathan Lim Jun Yi</div>
                <div class="member-details">
                    
                    <p>Hi, I'm Jonathan Lim Jun Yi a student of Monash University majoring in computer Science.</p>
                </div>
            </div>
            <div class="team-member">
                <div class="member-name">Alex Cheng Tze Pin</div>
                <div class="member-details">
                    
                    <p>Hi, I'm Alex Cheng Tze Pin a student of Monash University majoring in computer Science.</p>
                </div>
            </div>
            <div class="team-member">
                <div class="member-name">Prateek Tripathi</div>
                <div class="member-details">
                    
                    <p>Hi, I'm Prateek Tripathi a student of Monash University majoring in computer Science.</p>
                </div>
            </div>
        </div>
    </section>
</body>
</html>
```

The 'EXPLORER' sidebar on the left shows files like 'team-desc.html', 'team.css', 'Prateek.jpg', 'Prediction.html', 'index.html', and 'main.css'. The status bar at the bottom indicates 'Ln 1, Col 1' and 'Port: 5500'.

3. Prediction Page - The source code is stored in a html file called prediction.html. A CSS file named styles.css is used to style the webpage.

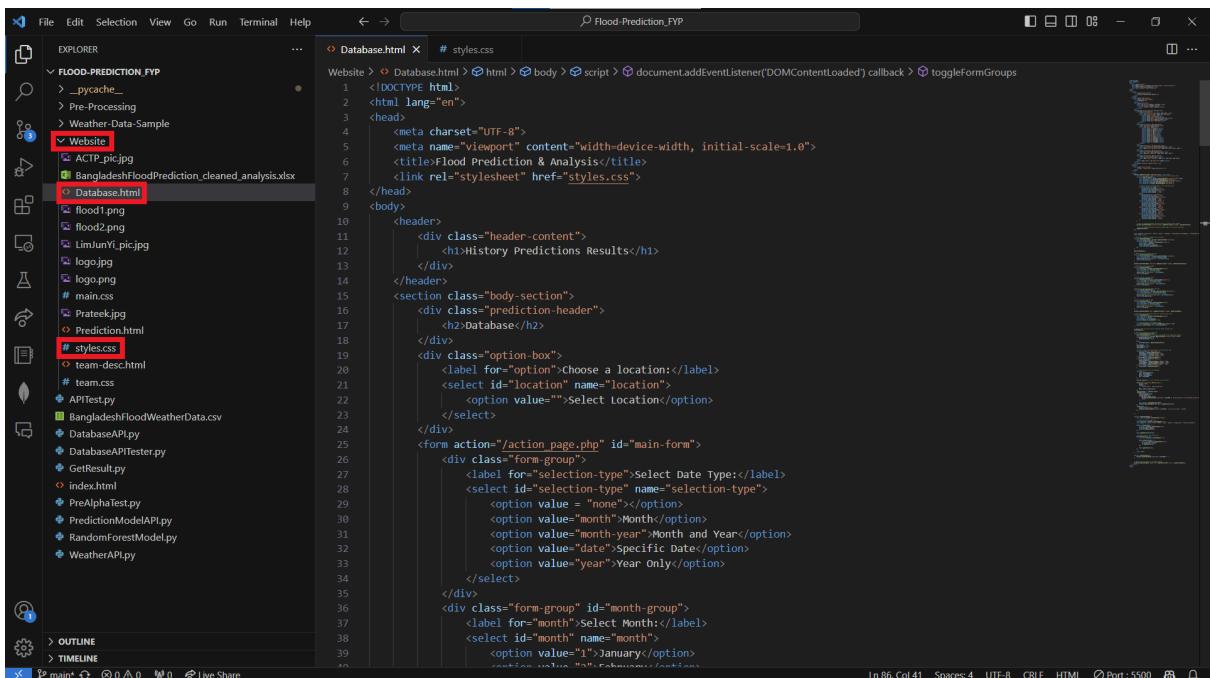


```

File Edit Selection View Go Run Terminal Help < > Flood-Prediction_FYP
EXPLORER
FLOOD-PREDICTION_FYP
> __pycache__
> Pre-Processing
> Weather-Data-Sample
Website
  ACTP.pic.jpg
  BangladeshFloodPrediction_cleaned_analysis.xlsx
  Database.html
  flood1.png
  flood2.png
  LimJunYi.pic.jpg
  logo.jpg
  # main.css
  Prateek.jpg
  Prediction.html
  # styles.css
  team-desc.html
  # team.css
  APITest.py
  DatabaseAPI.py
  DatabaseAPITester.py
  GetResult.py
  index.html
  PreAlphaTest.py
  PredictionModelAPI.py
  RandomForestModel.py
  WeatherAPI.py
Outline > Timeline
Live Share
Prediction.html # styles.css
Website > Prediction.html > HTML > body > script > getSelectedLocation
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Flood Prediction & Analysis</title>
7   <link rel="stylesheet" href="styles.css">
8 </head>
9 <body>
10   <header>
11     <div class="header-content">
12       <h1>Flood Prediction & Analysis</h1>
13     </div>
14   </header>
15   <section class="body-section">
16     <div class="prediction-header">
17       <h2>Prediction</h2>
18     </div>
19     <div class="option-box">
20       <label for="option">Choose a location:</label>
21       <select id="location" name="location">
22         <option value="">Select Location</option>
23       </select>
24     </div>
25     <form action="/action_page.php">
26       <label for="birthday">Date:</label>
27       <input type="date" id="date" name="date" min="" max="">
28       <button type="submit" id="submit-btn">Submit</button>
29     </form>
30     <div class="container" id="result">
31     </div>
32     <div class="shap-graph-header">
33       <h2>SHAP Table</h2>
34     </div>
35     <div class="container" id = "shap-table"></div>
36     <div class="shap-graph-header">
37       <h2>SHAP Graph</h2>
38     </div>
39     <div class="container" id="shap-graph"></div>
Ln 65, Col 18 Spaces: 4 UTF-8 CRLF HTML Port: 5500

```

4. Past Results Page - The source code is stored in a html file called database.html. A CSS file named styles.css is used to style the webpage.



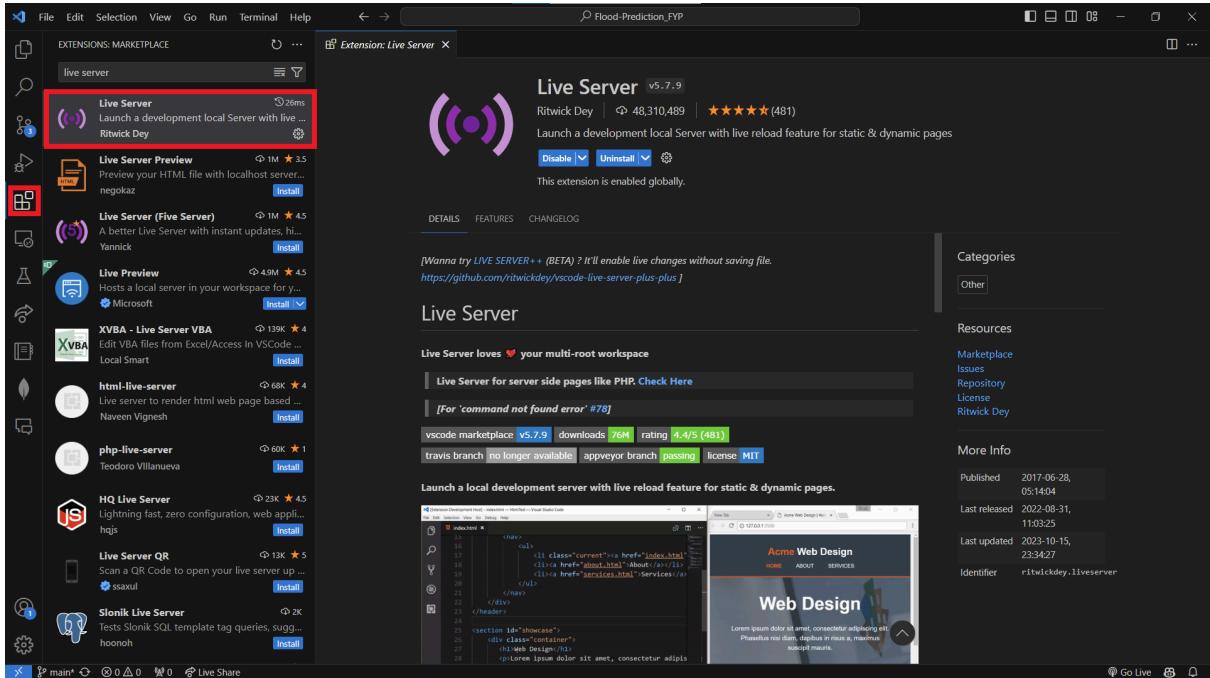
```

File Edit Selection View Go Run Terminal Help < > Flood-Prediction_FYP
EXPLORER
FLOOD-PREDICTION_FYP
> __pycache__
> Pre-Processing
> Weather-Data-Sample
Website
  ACTP.pic.jpg
  BangladeshFloodPrediction_cleaned_analysis.xlsx
  Database.html
  flood1.png
  flood2.png
  LimJunYi.pic.jpg
  logo.jpg
  # main.css
  Prateek.jpg
  Prediction.html
  # styles.css
  team-desc.html
  # team.css
  APITest.py
  DatabaseAPI.py
  DatabaseAPITester.py
  GetResult.py
  index.html
  PreAlphaTest.py
  PredictionModelAPI.py
  RandomForestModel.py
  WeatherAPI.py
Outline > Timeline
Live Share
Database.html # styles.css
Website > Database.html > HTML > body > script > document.addEventListener('DOMContentLoaded') callback > toggleFormGroups
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Flood Prediction & Analysis</title>
7   <link rel="stylesheet" href="styles.css">
8 </head>
9 <body>
10   <header>
11     <div class="header-content">
12       <h1>History Predictions Results</h1>
13     </div>
14   </header>
15   <section class="body-section">
16     <div class="prediction-header">
17       <h2>Database</h2>
18     </div>
19     <div class="option-box">
20       <label for="option">Choose a location:</label>
21       <select id="location" name="location">
22         <option value="">Select Location</option>
23       </select>
24     </div>
25     <form action="/action_page.php" id="main-form">
26       <div class="form-group">
27         <label for="selection-type">Select Date Type:</label>
28         <select id="selection-type" name="selection-type">
29           <option value = "none"></option>
30           <option value="month">Month</option>
31           <option value="month-year">Month and Year</option>
32           <option value="date">Specific Date</option>
33           <option value="year">Year only</option>
34         </select>
35       </div>
36       <div class="form-group" id="month-group">
37         <label for="month">Select Month:</label>
38         <select id="month" name="month">
39           <option value="1">January</option>
           <option value="2">February</option>
           <option value="3">March</option>
           <option value="4">April</option>
           <option value="5">May</option>
           <option value="6">June</option>
           <option value="7">July</option>
           <option value="8">August</option>
           <option value="9">September</option>
           <option value="10">October</option>
           <option value="11">November</option>
           <option value="12">December</option>
         </select>
40       </div>
Ln 86, Col 41 Spaces: 4 UTF-8 CRLF HTML Port: 5500

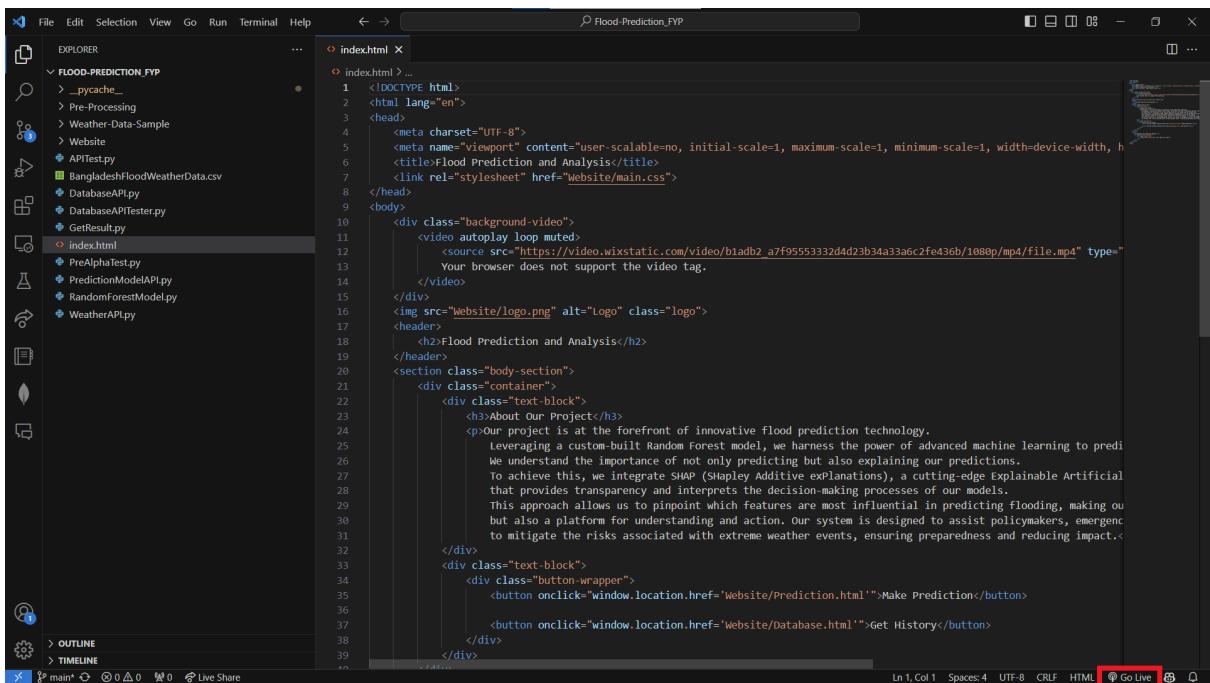
```

To host the webpage locally, follow these steps:

1. In the Visual Studio Code, download the live server extension



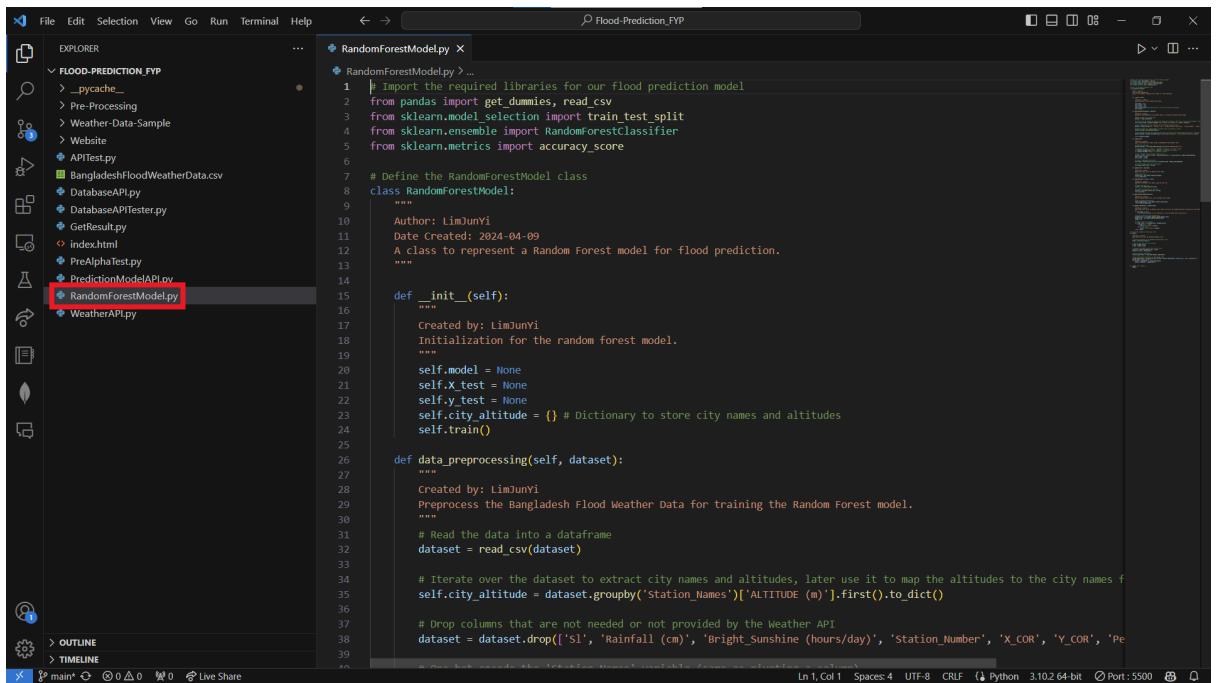
2. Click the Go Live button at the bottom right



2.1.2 Prediction Model

The prediction model consists of two Python files with one being the model itself and another one storing the codes to get the prediction results:

1. RandomForestModel.py - This file defines the Random Forest Model Class to train, test as well as use the prediction model.



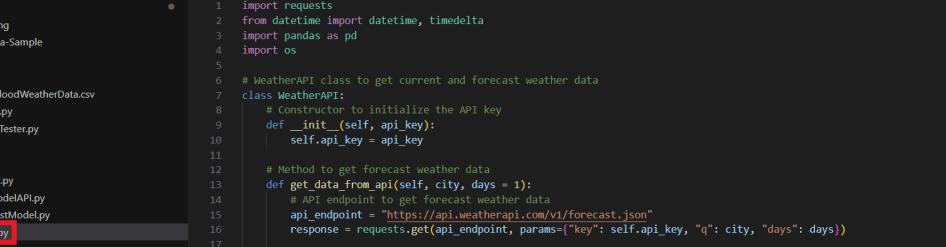
```
File Edit Selection View Go Run Terminal Help <- > Flood-Prediction_FYP
EXPLORER FLOOD-PREDICTION_FYP
RandomForestModel.py > ...
> __pycache__
> Pre-Processing
> Weather-Data-Sample
> Website
APITest.py
> BangladeshFloodWeatherData.csv
DatabaseAPI.py
DatabaseAPITester.py
GetResult.py
index.html
PreAlphaTest.py
PredictionModelAPI.py
RandomForestModel.py
WeatherAPI.py
OUTLINE > TIMELINE
X main* 0 △ 0 Live Share
RandomForestModel.py > ...
# Import the required libraries for our flood prediction model
1 from pandas import get_dummies, read_csv
2 from sklearn.model_selection import train_test_split
3 from sklearn.ensemble import RandomForestClassifier
4 from sklearn.metrics import accuracy_score
5
# Define the RandomForestModel class
6 class RandomForestModel:
7     """
8         Author: LimJunYi
9         Date Created: 2024-04-09
10        A class to represent a Random Forest model for flood prediction.
11    """
12
13    def __init__(self):
14        """
15            Created by: LimJunYi
16            Initialization for the random forest model.
17        """
18        self.model = None
19        self.X_test = None
20        self.y_test = None
21        self.city_altitude = {} # Dictionary to store city names and altitudes
22        self.train()
23
24    def data_preprocessing(self, dataset):
25        """
26            Created by: LimJunYi
27            Preprocess the Bangladesh Flood Weather Data for training the Random Forest model.
28        """
29
30        # Read the data into a dataframe
31        dataset = read_csv(dataset)
32
33        # Iterate over the dataset to extract city names and altitudes, later use it to map the altitudes to the city names if needed
34        self.city_altitude = dataset.groupby('Station_Names')['ALTITUDE (m)'].first().to_dict()
35
36        # Drop columns that are not needed or not provided by the Weather API
37        dataset = dataset.drop(['$1', 'Rainfall (cm)', 'Bright_Sunshine (hours/day)', 'Station_Number', 'X_COR', 'Y_COR'])
38
39    def train(self):
40        """
41            Created by: LimJunYi
42            Train the Random Forest model using the preprocessed data.
43        """
44        X_train = dataset.drop(['y'], axis=1)
45        y_train = dataset['y']
46
47        self.model = RandomForestClassifier(n_estimators=100, random_state=42)
48        self.model.fit(X_train, y_train)
49
50    def predict(self, X_test):
51        """
52            Created by: LimJunYi
53            Predict the flood risk for the given weather data.
54        """
55        return self.model.predict(X_test)
56
57    def get_accuracy(self, X_test, y_test):
58        """
59            Created by: LimJunYi
60            Calculate the accuracy of the model on the test data.
61        """
62        y_pred = self.predict(X_test)
63        accuracy = accuracy_score(y_test, y_pred)
64
65        return accuracy
66
67    def get_feature_importance(self):
68        """
69            Created by: LimJunYi
70            Get the feature importance for the Random Forest model.
71        """
72        feature_importance = self.model.feature_importances_
73
74        return feature_importance
75
76    def get_top_features(self, n_features):
77        """
78            Created by: LimJunYi
79            Get the top n features based on their importance.
80        """
81        sorted_importance = sorted(self.model.feature_importances_.items(), reverse=True)
82
83        top_features = [feature for feature, importance in sorted_importance[:n_features]]
84
85        return top_features
86
87    def get_top_cities(self, n_cities):
88        """
89            Created by: LimJunYi
90            Get the top n cities based on their altitude.
91        """
92        sorted_cities = sorted(self.city_altitude.items(), key=lambda x: x[1], reverse=True)
93
94        top_cities = [city for city, altitude in sorted_cities[:n_cities]]
95
96        return top_cities
97
98    def get_top_weather(self, n_weather):
99        """
100           Created by: LimJunYi
101          Get the top n weather variables based on their importance.
102      """
103      sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
104
105      top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
106
107      return top_weather
108
109    def get_top_stations(self, n_stations):
110        """
111            Created by: LimJunYi
112            Get the top n stations based on their importance.
113        """
114        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
115
116        top_stations = [station for station, importance in sorted_stations[:n_stations]]
117
118        return top_stations
119
120    def get_top_periods(self, n_periods):
121        """
122            Created by: LimJunYi
123            Get the top n periods based on their importance.
124        """
125        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
126
127        top_periods = [period for period, importance in sorted_periods[:n_periods]]
128
129        return top_periods
130
131    def get_top_rainfall(self, n_rainfall):
132        """
133            Created by: LimJunYi
134            Get the top n rainfall values based on their importance.
135        """
136        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
137
138        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
139
140        return top_rainfall
141
142    def get_top_sunshine(self, n_sunshine):
143        """
144            Created by: LimJunYi
145            Get the top n sunshine values based on their importance.
146        """
147        sorted_sunshine = sorted(self.model.feature_importances_.items(), reverse=True)
148
149        top_sunshine = [sunshine for sunshine, importance in sorted_sunshine[:n_sunshine]]
150
151        return top_sunshine
152
153    def get_top_altitude(self, n_altitude):
154        """
155            Created by: LimJunYi
156            Get the top n altitude values based on their importance.
157        """
158        sorted_altitude = sorted(self.model.feature_importances_.items(), reverse=True)
159
160        top_altitude = [altitude for altitude, importance in sorted_altitude[:n_altitude]]
161
162        return top_altitude
163
164    def get_top_weather_api(self, n_weather_api):
165        """
166            Created by: LimJunYi
167            Get the top n weather API variables based on their importance.
168        """
169        sorted_weather_api = sorted(self.model.feature_importances_.items(), reverse=True)
170
171        top_weather_api = [weather_api for weather_api, importance in sorted_weather_api[:n_weather_api]]
172
173        return top_weather_api
174
175    def get_top_stations_api(self, n_stations_api):
176        """
177            Created by: LimJunYi
178            Get the top n station API variables based on their importance.
179        """
180        sorted_stations_api = sorted(self.model.feature_importances_.items(), reverse=True)
181
182        top_stations_api = [station_api for station_api, importance in sorted_stations_api[:n_stations_api]]
183
184        return top_stations_api
185
186    def get_top_periods_api(self, n_periods_api):
187        """
188            Created by: LimJunYi
189            Get the top n period API variables based on their importance.
190        """
191        sorted_periods_api = sorted(self.model.feature_importances_.items(), reverse=True)
192
193        top_periods_api = [period_api for period_api, importance in sorted_periods_api[:n_periods_api]]
194
195        return top_periods_api
196
197    def get_top_cities_api(self, n_cities_api):
198        """
199            Created by: LimJunYi
200            Get the top n city API variables based on their importance.
201        """
202        sorted_cities_api = sorted(self.model.feature_importances_.items(), reverse=True)
203
204        top_cities_api = [city_api for city_api, importance in sorted_cities_api[:n_cities_api]]
205
206        return top_cities_api
207
208    def get_top_sunshine_api(self, n_sunshine_api):
209        """
210            Created by: LimJunYi
211            Get the top n sunshine API variables based on their importance.
212        """
213        sorted_sunshine_api = sorted(self.model.feature_importances_.items(), reverse=True)
214
215        top_sunshine_api = [sunshine_api for sunshine_api, importance in sorted_sunshine_api[:n_sunshine_api]]
216
217        return top_sunshine_api
218
219    def get_top_rainfall_api(self, n_rainfall_api):
220        """
221            Created by: LimJunYi
222            Get the top n rainfall API variables based on their importance.
223        """
224        sorted_rainfall_api = sorted(self.model.feature_importances_.items(), reverse=True)
225
226        top_rainfall_api = [rainfall_api for rainfall_api, importance in sorted_rainfall_api[:n_rainfall_api]]
227
228        return top_rainfall_api
229
230    def get_top_altitude_api(self, n_altitude_api):
231        """
232            Created by: LimJunYi
233            Get the top n altitude API variables based on their importance.
234        """
235        sorted_altitude_api = sorted(self.model.feature_importances_.items(), reverse=True)
236
237        top_altitude_api = [altitude_api for altitude_api, importance in sorted_altitude_api[:n_altitude_api]]
238
239        return top_altitude_api
240
241    def get_top_weather(self, n_weather):
242        """
243            Created by: LimJunYi
244            Get the top n weather variables based on their importance.
245        """
246        sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
247
248        top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
249
250        return top_weather
251
252    def get_top_stations(self, n_stations):
253        """
254            Created by: LimJunYi
255            Get the top n stations based on their importance.
256        """
257        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
258
259        top_stations = [station for station, importance in sorted_stations[:n_stations]]
260
261        return top_stations
262
263    def get_top_periods(self, n_periods):
264        """
265            Created by: LimJunYi
266            Get the top n periods based on their importance.
267        """
268        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
269
270        top_periods = [period for period, importance in sorted_periods[:n_periods]]
271
272        return top_periods
273
274    def get_top_rainfall(self, n_rainfall):
275        """
276            Created by: LimJunYi
277            Get the top n rainfall values based on their importance.
278        """
279        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
280
281        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
282
283        return top_rainfall
284
285    def get_top_sunshine(self, n_sunshine):
286        """
287            Created by: LimJunYi
288            Get the top n sunshine values based on their importance.
289        """
290        sorted_sunshine = sorted(self.model.feature_importances_.items(), reverse=True)
291
292        top_sunshine = [sunshine for sunshine, importance in sorted_sunshine[:n_sunshine]]
293
294        return top_sunshine
295
296    def get_top_altitude(self, n_altitude):
297        """
298            Created by: LimJunYi
299            Get the top n altitude values based on their importance.
300        """
301        sorted_altitude = sorted(self.model.feature_importances_.items(), reverse=True)
302
303        top_altitude = [altitude for altitude, importance in sorted_altitude[:n_altitude]]
304
305        return top_altitude
306
307    def get_top_weather_api(self, n_weather_api):
308        """
309            Created by: LimJunYi
310            Get the top n weather API variables based on their importance.
311        """
312        sorted_weather_api = sorted(self.model.feature_importances_.items(), reverse=True)
313
314        top_weather_api = [weather_api for weather_api, importance in sorted_weather_api[:n_weather_api]]
315
316        return top_weather_api
317
318    def get_top_stations_api(self, n_stations_api):
319        """
320            Created by: LimJunYi
321            Get the top n station API variables based on their importance.
322        """
323        sorted_stations_api = sorted(self.model.feature_importances_.items(), reverse=True)
324
325        top_stations_api = [station_api for station_api, importance in sorted_stations_api[:n_stations_api]]
326
327        return top_stations_api
328
329    def get_top_periods_api(self, n_periods_api):
330        """
331            Created by: LimJunYi
332            Get the top n period API variables based on their importance.
333        """
334        sorted_periods_api = sorted(self.model.feature_importances_.items(), reverse=True)
335
336        top_periods_api = [period_api for period_api, importance in sorted_periods_api[:n_periods_api]]
337
338        return top_periods_api
339
340    def get_top_cities_api(self, n_cities_api):
341        """
342            Created by: LimJunYi
343            Get the top n city API variables based on their importance.
344        """
345        sorted_cities_api = sorted(self.model.feature_importances_.items(), reverse=True)
346
347        top_cities_api = [city_api for city_api, importance in sorted_cities_api[:n_cities_api]]
348
349        return top_cities_api
350
351    def get_top_sunshine_api(self, n_sunshine_api):
352        """
353            Created by: LimJunYi
354            Get the top n sunshine API variables based on their importance.
355        """
356        sorted_sunshine_api = sorted(self.model.feature_importances_.items(), reverse=True)
357
358        top_sunshine_api = [sunshine_api for sunshine_api, importance in sorted_sunshine_api[:n_sunshine_api]]
359
360        return top_sunshine_api
361
362    def get_top_rainfall_api(self, n_rainfall_api):
363        """
364            Created by: LimJunYi
365            Get the top n rainfall API variables based on their importance.
366        """
367        sorted_rainfall_api = sorted(self.model.feature_importances_.items(), reverse=True)
368
369        top_rainfall_api = [rainfall_api for rainfall_api, importance in sorted_rainfall_api[:n_rainfall_api]]
370
371        return top_rainfall_api
372
373    def get_top_altitude_api(self, n_altitude_api):
374        """
375            Created by: LimJunYi
376            Get the top n altitude API variables based on their importance.
377        """
378        sorted_altitude_api = sorted(self.model.feature_importances_.items(), reverse=True)
379
380        top_altitude_api = [altitude_api for altitude_api, importance in sorted_altitude_api[:n_altitude_api]]
381
382        return top_altitude_api
383
384    def get_top_weather(self, n_weather):
385        """
386            Created by: LimJunYi
387            Get the top n weather variables based on their importance.
388        """
389        sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
390
391        top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
392
393        return top_weather
394
395    def get_top_stations(self, n_stations):
396        """
397            Created by: LimJunYi
398            Get the top n stations based on their importance.
399        """
400        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
401
402        top_stations = [station for station, importance in sorted_stations[:n_stations]]
403
404        return top_stations
405
406    def get_top_periods(self, n_periods):
407        """
408            Created by: LimJunYi
409            Get the top n periods based on their importance.
410        """
411        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
412
413        top_periods = [period for period, importance in sorted_periods[:n_periods]]
414
415        return top_periods
416
417    def get_top_rainfall(self, n_rainfall):
418        """
419            Created by: LimJunYi
420            Get the top n rainfall values based on their importance.
421        """
422        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
423
424        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
425
426        return top_rainfall
427
428    def get_top_sunshine(self, n_sunshine):
429        """
430            Created by: LimJunYi
431            Get the top n sunshine values based on their importance.
432        """
433        sorted_sunshine = sorted(self.model.feature_importances_.items(), reverse=True)
434
435        top_sunshine = [sunshine for sunshine, importance in sorted_sunshine[:n_sunshine]]
436
437        return top_sunshine
438
439    def get_top_altitude(self, n_altitude):
440        """
441            Created by: LimJunYi
442            Get the top n altitude values based on their importance.
443        """
444        sorted_altitude = sorted(self.model.feature_importances_.items(), reverse=True)
445
446        top_altitude = [altitude for altitude, importance in sorted_altitude[:n_altitude]]
447
448        return top_altitude
449
450    def get_top_weather_api(self, n_weather_api):
451        """
452            Created by: LimJunYi
453            Get the top n weather API variables based on their importance.
454        """
455        sorted_weather_api = sorted(self.model.feature_importances_.items(), reverse=True)
456
457        top_weather_api = [weather_api for weather_api, importance in sorted_weather_api[:n_weather_api]]
458
459        return top_weather_api
460
461    def get_top_stations_api(self, n_stations_api):
462        """
463            Created by: LimJunYi
464            Get the top n station API variables based on their importance.
465        """
466        sorted_stations_api = sorted(self.model.feature_importances_.items(), reverse=True)
467
468        top_stations_api = [station_api for station_api, importance in sorted_stations_api[:n_stations_api]]
469
470        return top_stations_api
471
472    def get_top_periods_api(self, n_periods_api):
473        """
474            Created by: LimJunYi
475            Get the top n period API variables based on their importance.
476        """
477        sorted_periods_api = sorted(self.model.feature_importances_.items(), reverse=True)
478
479        top_periods_api = [period_api for period_api, importance in sorted_periods_api[:n_periods_api]]
480
481        return top_periods_api
482
483    def get_top_cities_api(self, n_cities_api):
484        """
485            Created by: LimJunYi
486            Get the top n city API variables based on their importance.
487        """
488        sorted_cities_api = sorted(self.model.feature_importances_.items(), reverse=True)
489
490        top_cities_api = [city_api for city_api, importance in sorted_cities_api[:n_cities_api]]
491
492        return top_cities_api
493
494    def get_top_sunshine_api(self, n_sunshine_api):
495        """
496            Created by: LimJunYi
497            Get the top n sunshine API variables based on their importance.
498        """
499        sorted_sunshine_api = sorted(self.model.feature_importances_.items(), reverse=True)
500
501        top_sunshine_api = [sunshine_api for sunshine_api, importance in sorted_sunshine_api[:n_sunshine_api]]
502
503        return top_sunshine_api
504
505    def get_top_rainfall_api(self, n_rainfall_api):
506        """
507            Created by: LimJunYi
508            Get the top n rainfall API variables based on their importance.
509        """
510        sorted_rainfall_api = sorted(self.model.feature_importances_.items(), reverse=True)
511
512        top_rainfall_api = [rainfall_api for rainfall_api, importance in sorted_rainfall_api[:n_rainfall_api]]
513
514        return top_rainfall_api
515
516    def get_top_altitude_api(self, n_altitude_api):
517        """
518            Created by: LimJunYi
519            Get the top n altitude API variables based on their importance.
520        """
521        sorted_altitude_api = sorted(self.model.feature_importances_.items(), reverse=True)
522
523        top_altitude_api = [altitude_api for altitude_api, importance in sorted_altitude_api[:n_altitude_api]]
524
525        return top_altitude_api
526
527    def get_top_weather(self, n_weather):
528        """
529            Created by: LimJunYi
530            Get the top n weather variables based on their importance.
531        """
532        sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
533
534        top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
535
536        return top_weather
537
538    def get_top_stations(self, n_stations):
539        """
540            Created by: LimJunYi
541            Get the top n stations based on their importance.
542        """
543        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
544
545        top_stations = [station for station, importance in sorted_stations[:n_stations]]
546
547        return top_stations
548
549    def get_top_periods(self, n_periods):
550        """
551            Created by: LimJunYi
552            Get the top n periods based on their importance.
553        """
554        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
555
556        top_periods = [period for period, importance in sorted_periods[:n_periods]]
557
558        return top_periods
559
560    def get_top_rainfall(self, n_rainfall):
561        """
562            Created by: LimJunYi
563            Get the top n rainfall values based on their importance.
564        """
565        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
566
567        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
568
569        return top_rainfall
570
571    def get_top_sunshine(self, n_sunshine):
572        """
573            Created by: LimJunYi
574            Get the top n sunshine values based on their importance.
575        """
576        sorted_sunshine = sorted(self.model.feature_importances_.items(), reverse=True)
577
578        top_sunshine = [sunshine for sunshine, importance in sorted_sunshine[:n_sunshine]]
579
580        return top_sunshine
581
582    def get_top_altitude(self, n_altitude):
583        """
584            Created by: LimJunYi
585            Get the top n altitude values based on their importance.
586        """
587        sorted_altitude = sorted(self.model.feature_importances_.items(), reverse=True)
588
589        top_altitude = [altitude for altitude, importance in sorted_altitude[:n_altitude]]
590
591        return top_altitude
592
593    def get_top_weather_api(self, n_weather_api):
594        """
595            Created by: LimJunYi
596            Get the top n weather API variables based on their importance.
597        """
598        sorted_weather_api = sorted(self.model.feature_importances_.items(), reverse=True)
599
600        top_weather_api = [weather_api for weather_api, importance in sorted_weather_api[:n_weather_api]]
601
602        return top_weather_api
603
604    def get_top_stations_api(self, n_stations_api):
605        """
606            Created by: LimJunYi
607            Get the top n station API variables based on their importance.
608        """
609        sorted_stations_api = sorted(self.model.feature_importances_.items(), reverse=True)
610
611        top_stations_api = [station_api for station_api, importance in sorted_stations_api[:n_stations_api]]
612
613        return top_stations_api
614
615    def get_top_periods_api(self, n_periods_api):
616        """
617            Created by: LimJunYi
618            Get the top n period API variables based on their importance.
619        """
620        sorted_periods_api = sorted(self.model.feature_importances_.items(), reverse=True)
621
622        top_periods_api = [period_api for period_api, importance in sorted_periods_api[:n_periods_api]]
623
624        return top_periods_api
625
626    def get_top_cities_api(self, n_cities_api):
627        """
628            Created by: LimJunYi
629            Get the top n city API variables based on their importance.
630        """
631        sorted_cities_api = sorted(self.model.feature_importances_.items(), reverse=True)
632
633        top_cities_api = [city_api for city_api, importance in sorted_cities_api[:n_cities_api]]
634
635        return top_cities_api
636
637    def get_top_sunshine_api(self, n_sunshine_api):
638        """
639            Created by: LimJunYi
640            Get the top n sunshine API variables based on their importance.
641        """
642        sorted_sunshine_api = sorted(self.model.feature_importances_.items(), reverse=True)
643
644        top_sunshine_api = [sunshine_api for sunshine_api, importance in sorted_sunshine_api[:n_sunshine_api]]
645
646        return top_sunshine_api
647
648    def get_top_rainfall_api(self, n_rainfall_api):
649        """
650            Created by: LimJunYi
651            Get the top n rainfall API variables based on their importance.
652        """
653        sorted_rainfall_api = sorted(self.model.feature_importances_.items(), reverse=True)
654
655        top_rainfall_api = [rainfall_api for rainfall_api, importance in sorted_rainfall_api[:n_rainfall_api]]
656
657        return top_rainfall_api
658
659    def get_top_altitude_api(self, n_altitude_api):
660        """
661            Created by: LimJunYi
662            Get the top n altitude API variables based on their importance.
663        """
664        sorted_altitude_api = sorted(self.model.feature_importances_.items(), reverse=True)
665
666        top_altitude_api = [altitude_api for altitude_api, importance in sorted_altitude_api[:n_altitude_api]]
667
668        return top_altitude_api
669
670    def get_top_weather(self, n_weather):
671        """
672            Created by: LimJunYi
673            Get the top n weather variables based on their importance.
674        """
675        sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
676
677        top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
678
679        return top_weather
680
681    def get_top_stations(self, n_stations):
682        """
683            Created by: LimJunYi
684            Get the top n stations based on their importance.
685        """
686        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
687
688        top_stations = [station for station, importance in sorted_stations[:n_stations]]
689
690        return top_stations
691
692    def get_top_periods(self, n_periods):
693        """
694            Created by: LimJunYi
695            Get the top n periods based on their importance.
696        """
697        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
698
699        top_periods = [period for period, importance in sorted_periods[:n_periods]]
700
701        return top_periods
702
703    def get_top_rainfall(self, n_rainfall):
704        """
705            Created by: LimJunYi
706            Get the top n rainfall values based on their importance.
707        """
708        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
709
710        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
711
712        return top_rainfall
713
714    def get_top_sunshine(self, n_sunshine):
715        """
716            Created by: LimJunYi
717            Get the top n sunshine values based on their importance.
718        """
719        sorted_sunshine = sorted(self.model.feature_importances_.items(), reverse=True)
720
721        top_sunshine = [sunshine for sunshine, importance in sorted_sunshine[:n_sunshine]]
722
723        return top_sunshine
724
725    def get_top_altitude(self, n_altitude):
726        """
727            Created by: LimJunYi
728            Get the top n altitude values based on their importance.
729        """
730        sorted_altitude = sorted(self.model.feature_importances_.items(), reverse=True)
731
732        top_altitude = [altitude for altitude, importance in sorted_altitude[:n_altitude]]
733
734        return top_altitude
735
736    def get_top_weather_api(self, n_weather_api):
737        """
738            Created by: LimJunYi
739            Get the top n weather API variables based on their importance.
740        """
741        sorted_weather_api = sorted(self.model.feature_importances_.items(), reverse=True)
742
743        top_weather_api = [weather_api for weather_api, importance in sorted_weather_api[:n_weather_api]]
744
745        return top_weather_api
746
747    def get_top_stations_api(self, n_stations_api):
748        """
749            Created by: LimJunYi
750            Get the top n station API variables based on their importance.
751        """
752        sorted_stations_api = sorted(self.model.feature_importances_.items(), reverse=True)
753
754        top_stations_api = [station_api for station_api, importance in sorted_stations_api[:n_stations_api]]
755
756        return top_stations_api
757
758    def get_top_periods_api(self, n_periods_api):
759        """
760            Created by: LimJunYi
761            Get the top n period API variables based on their importance.
762        """
763        sorted_periods_api = sorted(self.model.feature_importances_.items(), reverse=True)
764
765        top_periods_api = [period_api for period_api, importance in sorted_periods_api[:n_periods_api]]
766
767        return top_periods_api
768
769    def get_top_cities_api(self, n_cities_api):
770        """
771            Created by: LimJunYi
772            Get the top n city API variables based on their importance.
773        """
774        sorted_cities_api = sorted(self.model.feature_importances_.items(), reverse=True)
775
776        top_cities_api = [city_api for city_api, importance in sorted_cities_api[:n_cities_api]]
777
778        return top_cities_api
779
780    def get_top_sunshine_api(self, n_sunshine_api):
781        """
782            Created by: LimJunYi
783            Get the top n sunshine API variables based on their importance.
784        """
785        sorted_sunshine_api = sorted(self.model.feature_importances_.items(), reverse=True)
786
787        top_sunshine_api = [sunshine_api for sunshine_api, importance in sorted_sunshine_api[:n_sunshine_api]]
788
789        return top_sunshine_api
790
791    def get_top_rainfall_api(self, n_rainfall_api):
792        """
793            Created by: LimJunYi
794            Get the top n rainfall API variables based on their importance.
795        """
796        sorted_rainfall_api = sorted(self.model.feature_importances_.items(), reverse=True)
797
798        top_rainfall_api = [rainfall_api for rainfall_api, importance in sorted_rainfall_api[:n_rainfall_api]]
799
800        return top_rainfall_api
801
802    def get_top_altitude_api(self, n_altitude_api):
803        """
804            Created by: LimJunYi
805            Get the top n altitude API variables based on their importance.
806        """
807        sorted_altitude_api = sorted(self.model.feature_importances_.items(), reverse=True)
808
809        top_altitude_api = [altitude_api for altitude_api, importance in sorted_altitude_api[:n_altitude_api]]
810
811        return top_altitude_api
812
813    def get_top_weather(self, n_weather):
814        """
815            Created by: LimJunYi
816            Get the top n weather variables based on their importance.
817        """
818        sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
819
820        top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
821
822        return top_weather
823
824    def get_top_stations(self, n_stations):
825        """
826            Created by: LimJunYi
827            Get the top n stations based on their importance.
828        """
829        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
830
831        top_stations = [station for station, importance in sorted_stations[:n_stations]]
832
833        return top_stations
834
835    def get_top_periods(self, n_periods):
836        """
837            Created by: LimJunYi
838            Get the top n periods based on their importance.
839        """
840        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
841
842        top_periods = [period for period, importance in sorted_periods[:n_periods]]
843
844        return top_periods
845
846    def get_top_rainfall(self, n_rainfall):
847        """
848            Created by: LimJunYi
849            Get the top n rainfall values based on their importance.
850        """
851        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
852
853        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
854
855        return top_rainfall
856
857    def get_top_sunshine(self, n_sunshine):
858        """
859            Created by: LimJunYi
860            Get the top n sunshine values based on their importance.
861        """
862        sorted_sunshine = sorted(self.model.feature_importances_.items(), reverse=True)
863
864        top_sunshine = [sunshine for sunshine, importance in sorted_sunshine[:n_sunshine]]
865
866        return top_sunshine
867
868    def get_top_altitude(self, n_altitude):
869        """
870            Created by: LimJunYi
871            Get the top n altitude values based on their importance.
872        """
873        sorted_altitude = sorted(self.model.feature_importances_.items(), reverse=True)
874
875        top_altitude = [altitude for altitude, importance in sorted_altitude[:n_altitude]]
876
877        return top_altitude
878
879    def get_top_weather_api(self, n_weather_api):
880        """
881            Created by: LimJunYi
882            Get the top n weather API variables based on their importance.
883        """
884        sorted_weather_api = sorted(self.model.feature_importances_.items(), reverse=True)
885
886        top_weather_api = [weather_api for weather_api, importance in sorted_weather_api[:n_weather_api]]
887
888        return top_weather_api
889
890    def get_top_stations_api(self, n_stations_api):
891        """
892            Created by: LimJunYi
893            Get the top n station API variables based on their importance.
894        """
895        sorted_stations_api = sorted(self.model.feature_importances_.items(), reverse=True)
896
897        top_stations_api = [station_api for station_api, importance in sorted_stations_api[:n_stations_api]]
898
899        return top_stations_api
900
901    def get_top_periods_api(self, n_periods_api):
902        """
903            Created by: LimJunYi
904            Get the top n period API variables based on their importance.
905        """
906        sorted_periods_api = sorted(self.model.feature_importances_.items(), reverse=True)
907
908        top_periods_api = [period_api for period_api, importance in sorted_periods_api[:n_periods_api]]
909
910        return top_periods_api
911
912    def get_top_cities_api(self, n_cities_api):
913        """
914            Created by: LimJunYi
915            Get the top n city API variables based on their importance.
916        """
917        sorted_cities_api = sorted(self.model.feature_importances_.items(), reverse=True)
918
919        top_cities_api = [city_api for city_api, importance in sorted_cities_api[:n_cities_api]]
920
921        return top_cities_api
922
923    def get_top_sunshine_api(self, n_sunshine_api):
924        """
925            Created by: LimJunYi
926            Get the top n sunshine API variables based on their importance.
927        """
928        sorted_sunshine_api = sorted(self.model.feature_importances_.items(), reverse=True)
929
930        top_sunshine_api = [sunshine_api for sunshine_api, importance in sorted_sunshine_api[:n_sunshine_api]]
931
932        return top_sunshine_api
933
934    def get_top_rainfall_api(self, n_rainfall_api):
935        """
936            Created by: LimJunYi
937            Get the top n rainfall API variables based on their importance.
938        """
939        sorted_rainfall_api = sorted(self.model.feature_importances_.items(), reverse=True)
940
941        top_rainfall_api = [rainfall_api for rainfall_api, importance in sorted_rainfall_api[:n_rainfall_api]]
942
943        return top_rainfall_api
944
945    def get_top_altitude_api(self, n_altitude_api):
946        """
947            Created by: LimJunYi
948            Get the top n altitude API variables based on their importance.
949        """
950        sorted_altitude_api = sorted(self.model.feature_importances_.items(), reverse=True)
951
952        top_altitude_api = [altitude_api for altitude_api, importance in sorted_altitude_api[:n_altitude_api]]
953
954        return top_altitude_api
955
956    def get_top_weather(self, n_weather):
957        """
958            Created by: LimJunYi
959            Get the top n weather variables based on their importance.
960        """
961        sorted_weather = sorted(self.model.feature_importances_.items(), reverse=True)
962
963        top_weather = [weather for weather, importance in sorted_weather[:n_weather]]
964
965        return top_weather
966
967    def get_top_stations(self, n_stations):
968        """
969            Created by: LimJunYi
970            Get the top n stations based on their importance.
971        """
972        sorted_stations = sorted(self.model.feature_importances_.items(), reverse=True)
973
974        top_stations = [station for station, importance in sorted_stations[:n_stations]]
975
976        return top_stations
977
978    def get_top_periods(self, n_periods):
979        """
980            Created by: LimJunYi
981            Get the top n periods based on their importance.
982        """
983        sorted_periods = sorted(self.model.feature_importances_.items(), reverse=True)
984
985        top_periods = [period for period, importance in sorted_periods[:n_periods]]
986
987        return top_periods
988
989    def get_top_rainfall(self, n_rainfall):
990        """
991            Created by: LimJunYi
992            Get the top n rainfall values based on their importance.
993        """
994        sorted_rainfall = sorted(self.model.feature_importances_.items(), reverse=True)
995
996        top_rainfall = [rainfall for rainfall, importance in sorted_rainfall[:n_rainfall]]
997
998        return top_rainfall
999
1000    def get_top_sunshine(self, n_sunshine):

```

Note: If you are not able to run the GetResult.py, please follow the instructions in [2.2 Accessing to the Weather API](#), then restart your IDE, you should be able to run it without problem (providing the city and date are valid).

2.2 Accessing the Weather API

The WeatherAPI class is defined in a Python file named WeatherAPI.py. The file will connect to the WeatherAPI and process the response to fit in the prediction model.

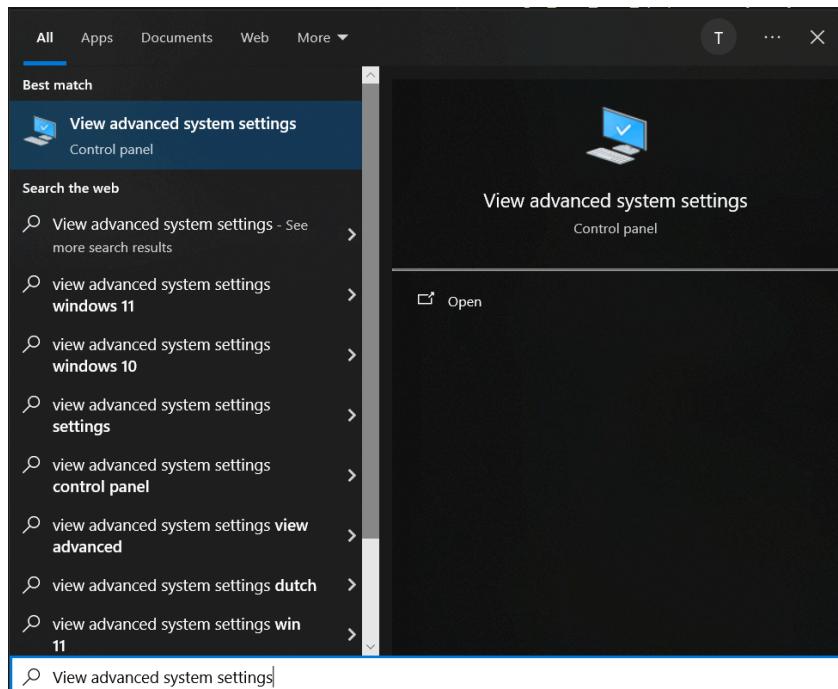


```
File Edit Selection View Go Run Terminal Help < > Flood-Prediction_FYP FLOOD-PREDICTION_FYP EXPLORER WeatherAPI.py x WeatherAPI.py (main) 1 import requests 2 from datetime import datetime, timedelta 3 import pandas as pd 4 import os 5 6 # WeatherAPI class to get current and forecast weather data 7 class WeatherAPI: 8     # Constructor to initialize the API key 9     def __init__(self, api_key): 10         self.api_key = api_key 11 12     # Method to get forecast weather data 13     def get_data_from_api(self, city, days = 1): 14         # API endpoint to get forecast weather data 15         api_endpoint = "https://api.weatherapi.com/v1/forecast.json" 16         response = requests.get(api_endpoint, params={"key": self.api_key, "q": city, "days": days}) 17 18         # check if the response is successful 19         if response.status_code == 200: 20             forecast_data = response.json() 21         else: 22             print("Error:", response.status_code) 23             return None 24 25         # Return the forecast data 26         return forecast_data 27 28     def count_date(self, current, days): 29         # Get the current date 30         date = datetime.strptime(current, "%Y-%m-%d %H:%M") 31 32         # Extract the date 33         date = date.date() 34 35         # calculate the future date 36         future_date = date + timedelta(days=1) 37 38         # Return the future date 39         return future_date.strftime("%Y-%m-%d") 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

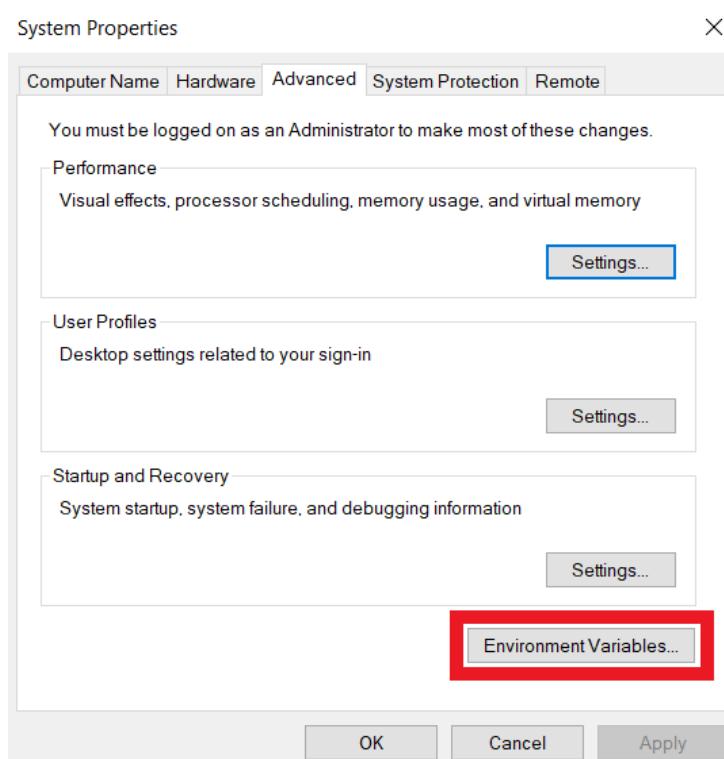
For more information on the WeatherAPI, please refer to [Weather and Geolocation API JSON and XML - WeatherAPI.com](#)

To use the Weather API, the administrator will have to do the prerequisite steps below to store the API key locally:

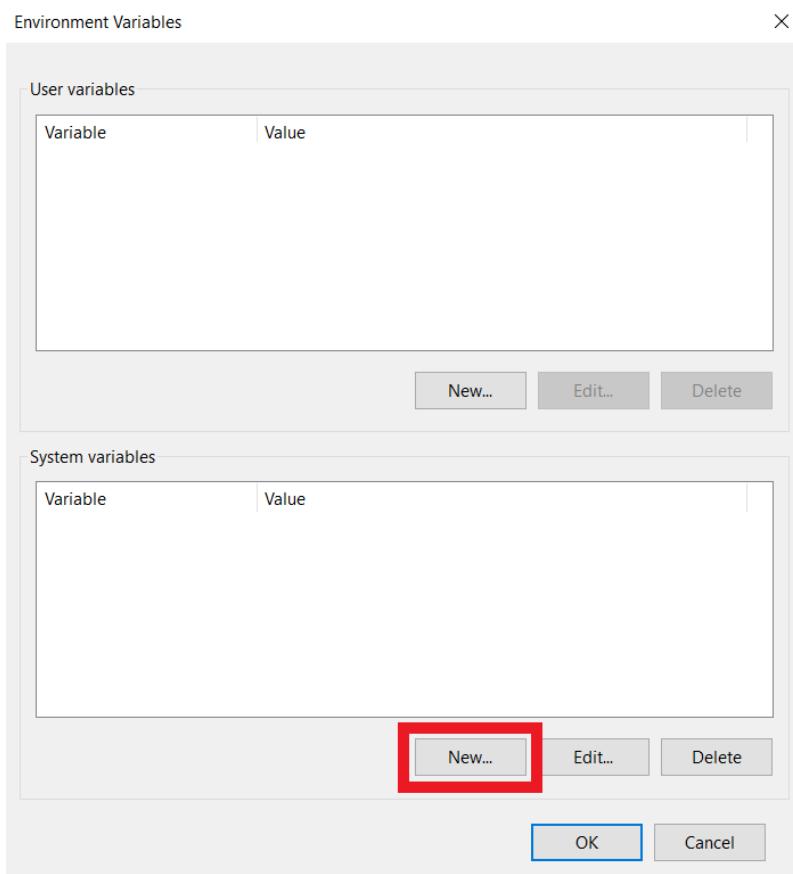
1. Go to “View advanced system settings”



2. Select Environment Variables

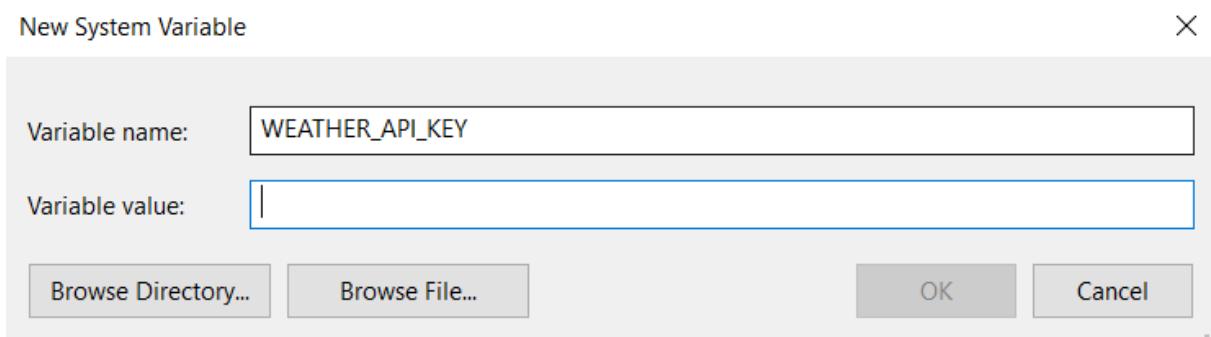


3. Select New for System variables

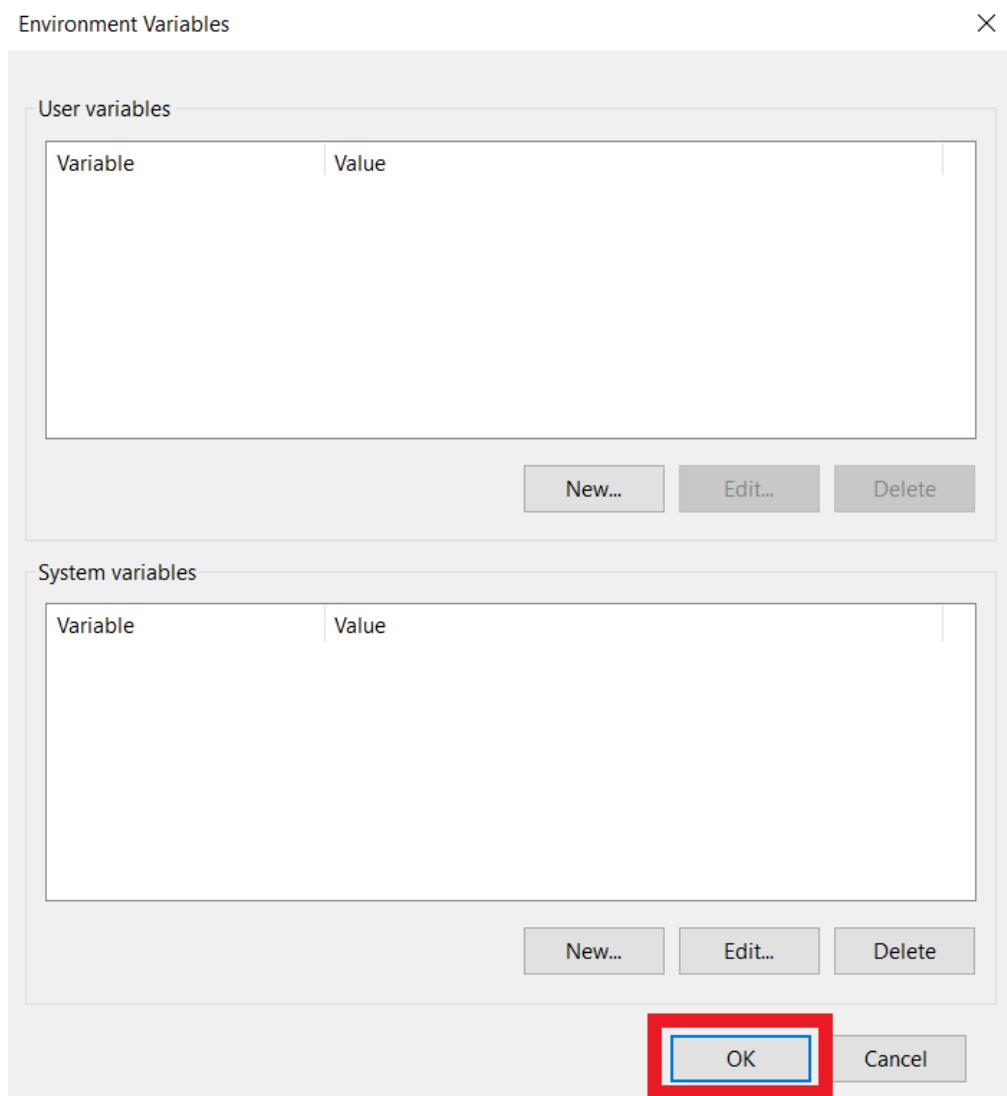


4. Enter the variable name as “WEATHER_API_KEY” and the key

Note: The key will be given once you have been assigned as an administrator



5. Click OK

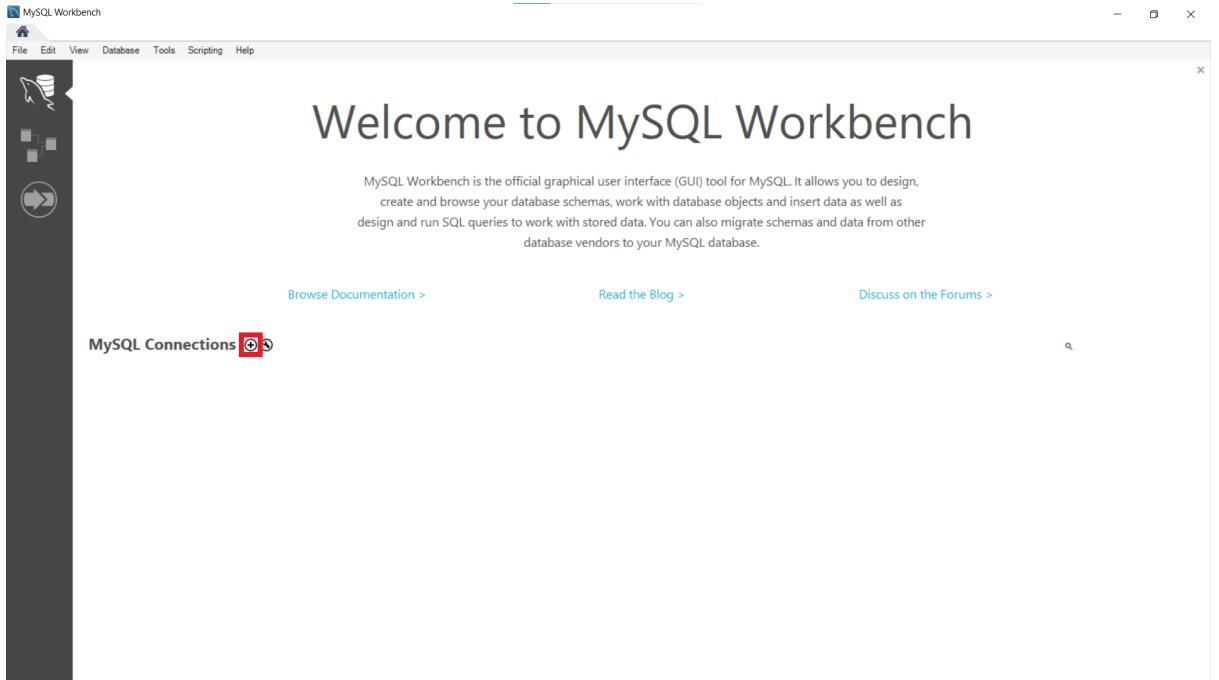


After doing this, you should be able to access the WeatherAPI.

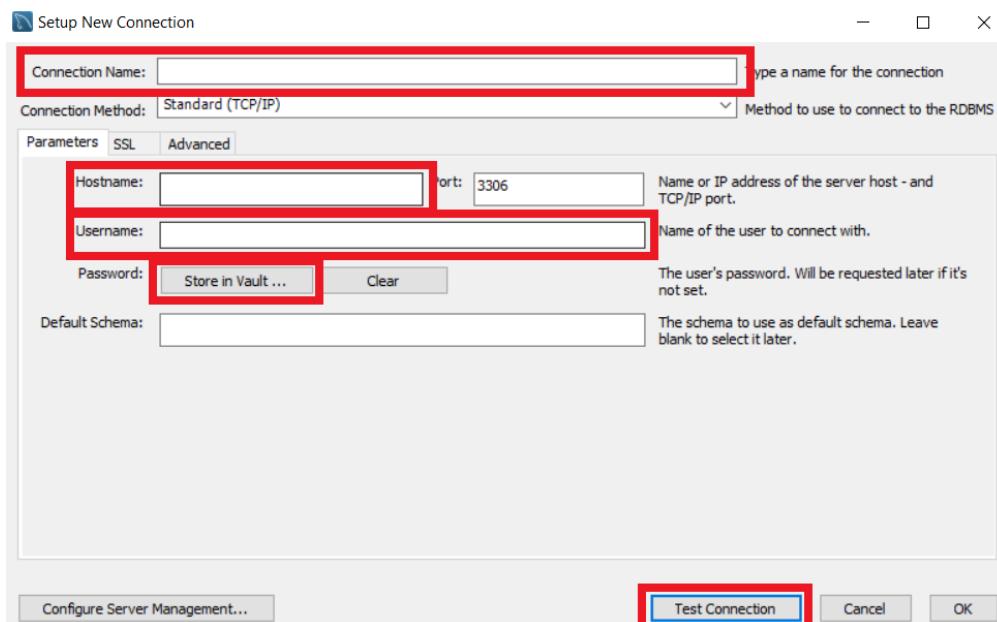
2.3 Accessing the MySQL Database

To access the MySQL Database, you will need to follow the steps below:

1. Download and Install the MySQL Workbench via [Download MySQL Workbench](#).
Note: You will need an Oracle account to download. If you do not have one, you can sign up for an account at [Oracle | Create Account](#)
2. In the MySQL Workbench, click on the plus sign



3. Fill in the details below, then click the Test Connection button:
 - a. Connection Name: Any name of your choice
 - b. Hostname: flood-prediction-fyp.mysql.database.azure.com
 - c. Username and password: To be given once you have been assigned as an administrator



4. You will receive this if you have connected successfully

MySQL Workbench

i Successfully made the MySQL connection

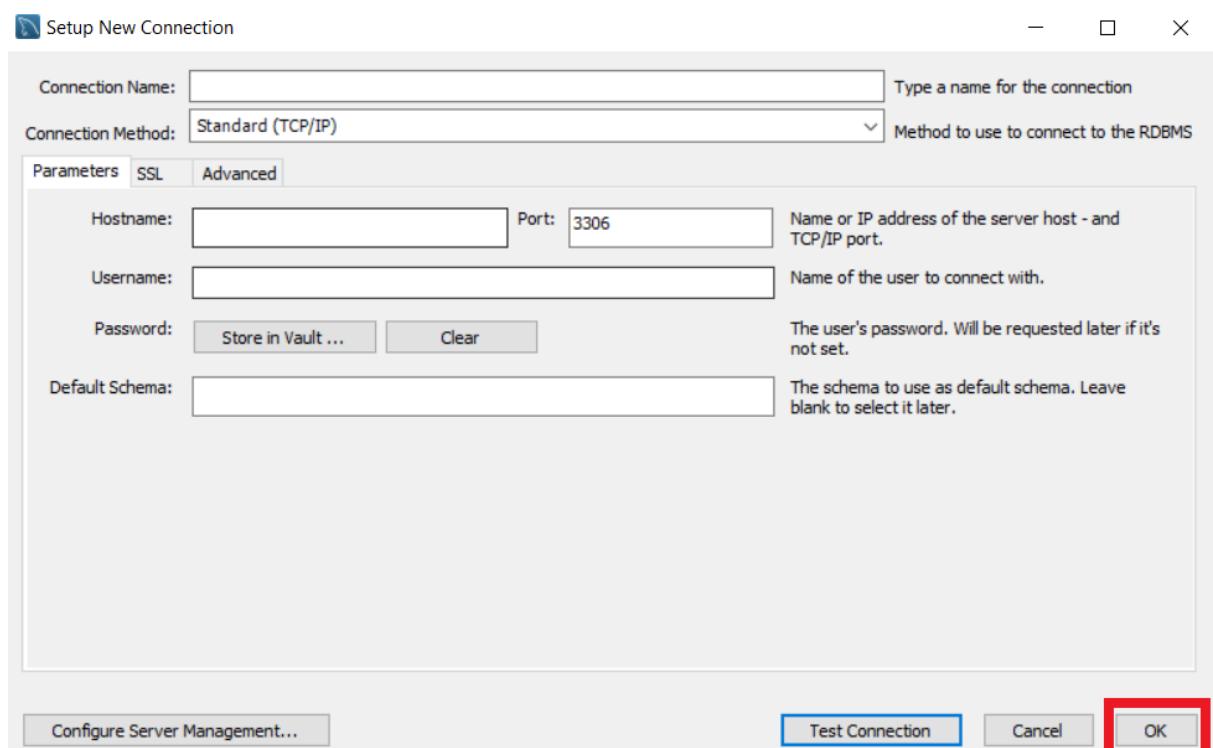
Information related to this connection:

Host: flood-prediction-fyp.mysql.database.azure.com
Port: 3306
User: actp
SSL: enabled with ECDHE-RSA-AES128-GCM-SHA256

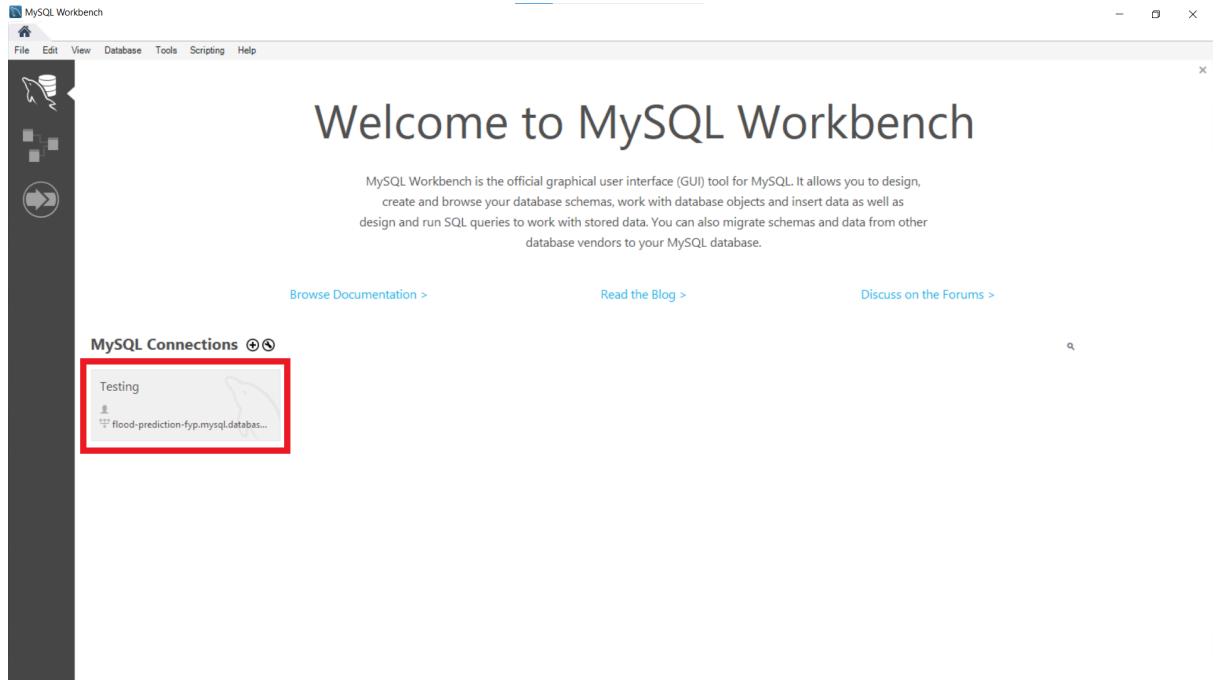
A successful MySQL connection was made with the parameters defined for this connection.

OK

5. Click the OK button



6. The connection will appear on the home page of the workbench



2.4 Accessing the Application APIs

2.4.1 Prediction Model API

The source code of the Prediction Model API is stored in a Python file called `PredictionModelAPI.py`

```

File Explorer: FLOOD-PREDICTION_FYP
  - __pycache__
  - Pre-Processing
  - Weather-Data-Sample
  - Website
  - APITest.py
  - BangladeshFloodWeatherData.csv
  - DatabaseAPITester.py
  - GetResult.py
  - index.html
  - PreAlphaTest.py
  - PredictionModelAPI.py (selected)
  - RandomForestModel.py
  - WeatherAPI.py

PredictionModelAPI.py
  import datetime
  from io import BytesIO
  from flask import Flask, request, jsonify, send_file
  from GetResult import getResult # Import your prediction model implementation
  from flask_cors import CORS

  app = Flask(__name__)
  CORS(app)
  getResult = getResult()

  @app.route('/predict', methods=['POST'])
  def predict():
      data = request.json # Assuming JSON data is sent in the request body
      city = data['city'] # Extract the city from the JSON data
      date = data['date'] # Extract the date from the JSON data
      prediction = getResult.get_prediction_result(city, date) # Call your prediction model function with city and date
      global result
      result = prediction[0][0]
      global shap_table
      shap_table = prediction[1]

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

```

Terminal output:

```

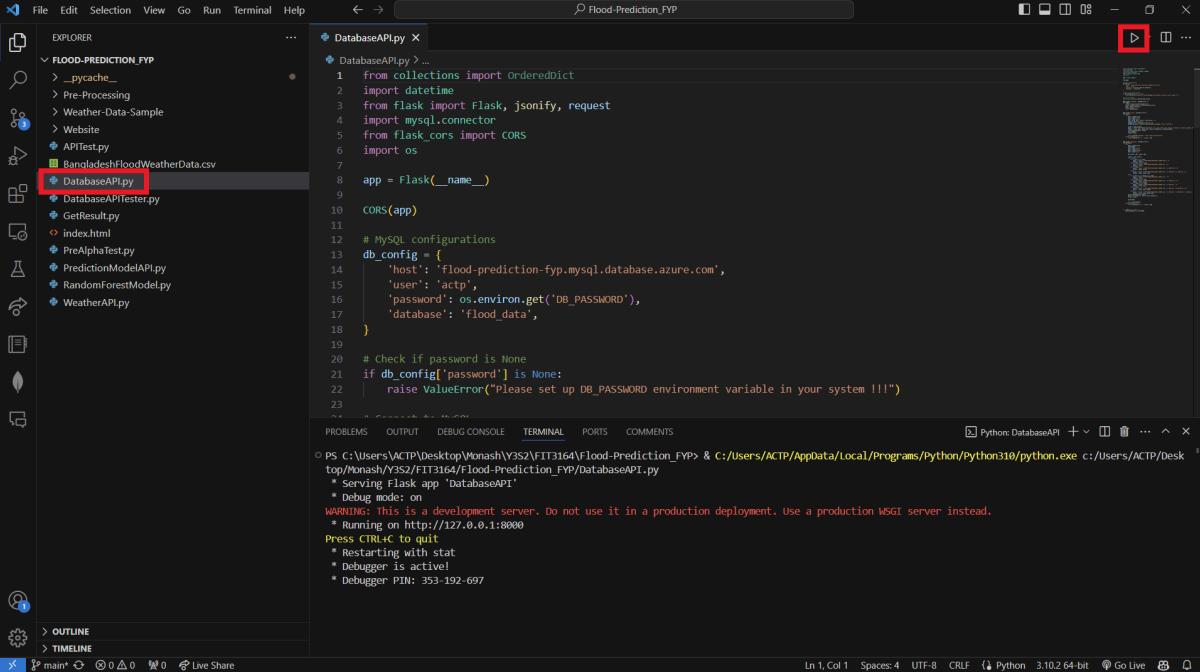
PS C:\Users\ACTP\Desktop\Wanash\VS2\FTT3164\Flood-Prediction_FYP> & C:/Users/ACTP/AppData/Local/Programs/Python/Python310/python.exe c:/Users/ACTP/Desktop/Monach/Y3S2/FTT3164/Flood-Prediction_FYP/PredictionModelAPI.py
* Serving Flask app "PredictionModelAPI"
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit.
* Restarting with stat
* Debugger is active!
* Debugger PIN: 353-192-697

```

Bottom status bar: Ln 1, Col 1 | Spaces: 4 | UTF-8 | CRLF | Python 3.10.2 64-bit | Go Live | Q

2.4.2 Database API

The source code of the Database API is stored in a Python file named DatabaseAPI.py. This file is used to connect the database and the web page. You will have to run the file to use the API.



```

File Edit Selection View Go Run Terminal Help ⌘ F Flood-Prediction_FYP

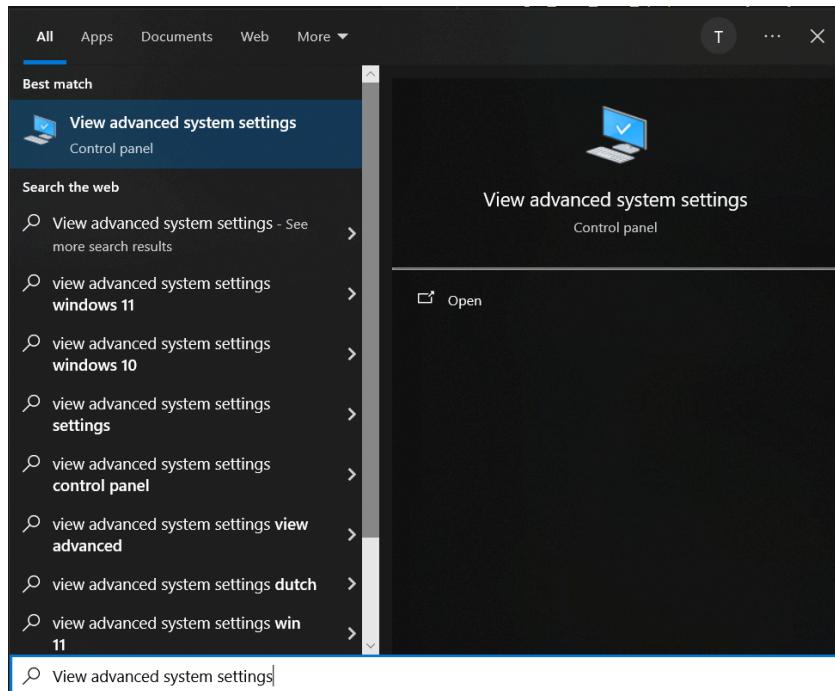
EXPLORER
FLOOD-PREDICTION_FYP
  > __pycache__ ...
  > Pre-Processing
  > Weather-Data-Sample
  > Website
    APITest.py
    BangladeshFloodWeatherData.csv
  DatabaseAPI.py (highlighted)
  DatabaseAPItester.py
  GetResult.py
  index.html
  PreAlphaTest.py
  PredictionModelAPI.py
  RandomForestModel.py
  WeatherAPI.py

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\ACTP\Desktop\Monash\Y3S2\FTT3164\Flood-Prediction_FYP> & C:/Users/ACTP/AppData/Local/Programs/Python/Python310/python.exe c:/Users/ACTP/Desktop/Monash/Y3S2/FTT3164/Flood-Prediction_FYP/DatabaseAPI.py
* Serving Flask app "DatabaseAPI"
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:8000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 353-192-697

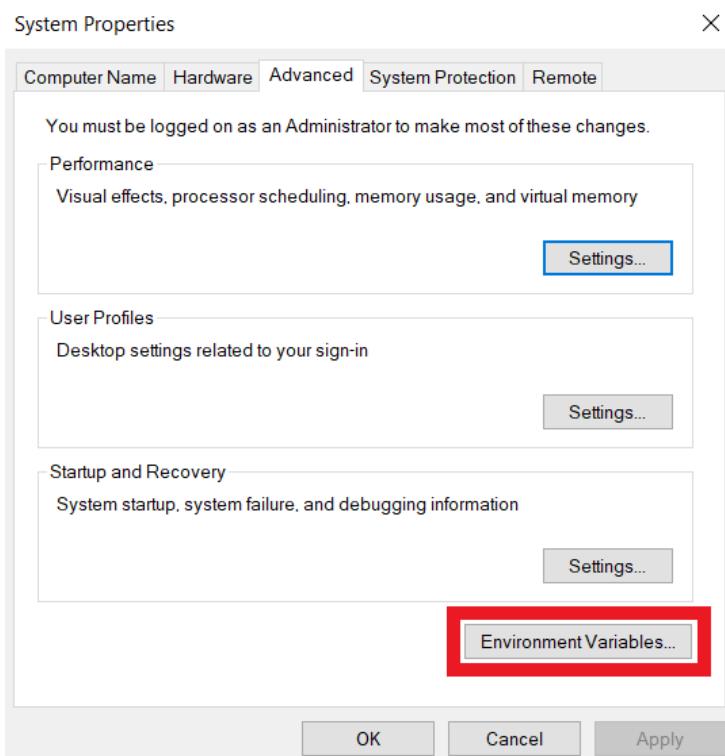
```

If you have not set up the environment variable, please follow the steps below:

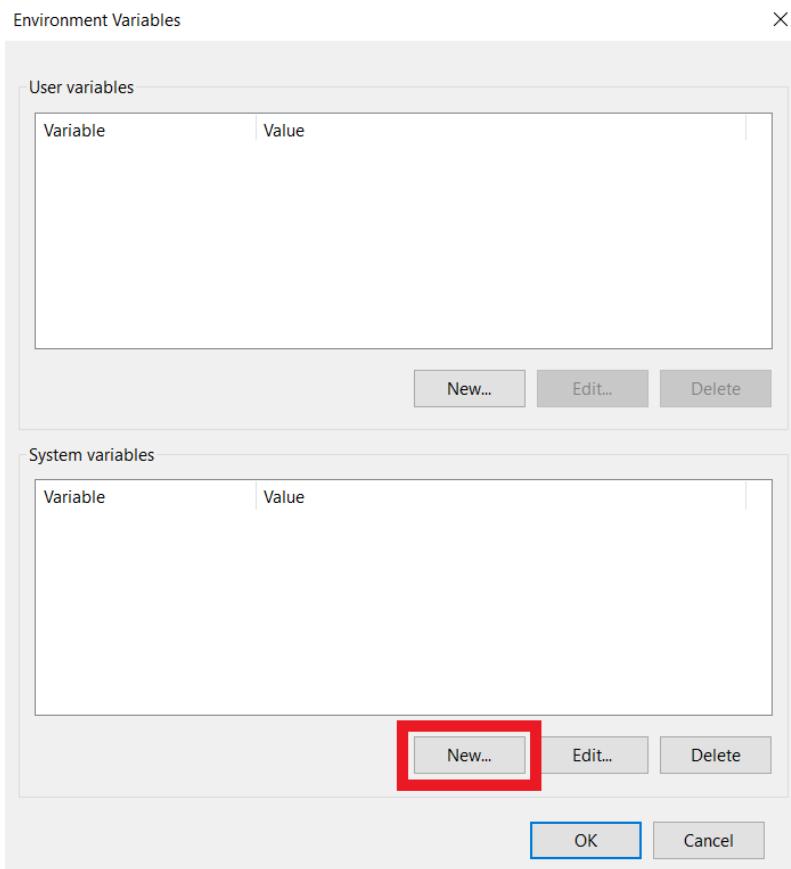
1. Go to “View advanced system settings”



2. Select Environment Variables

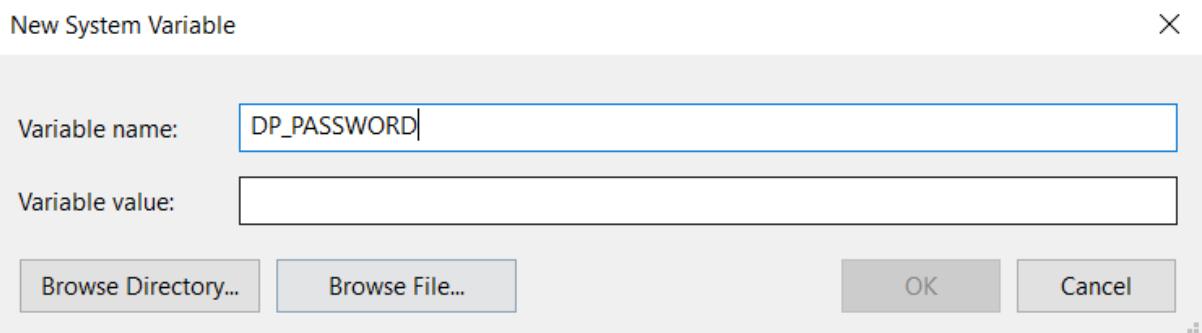


3. Select New for System variables

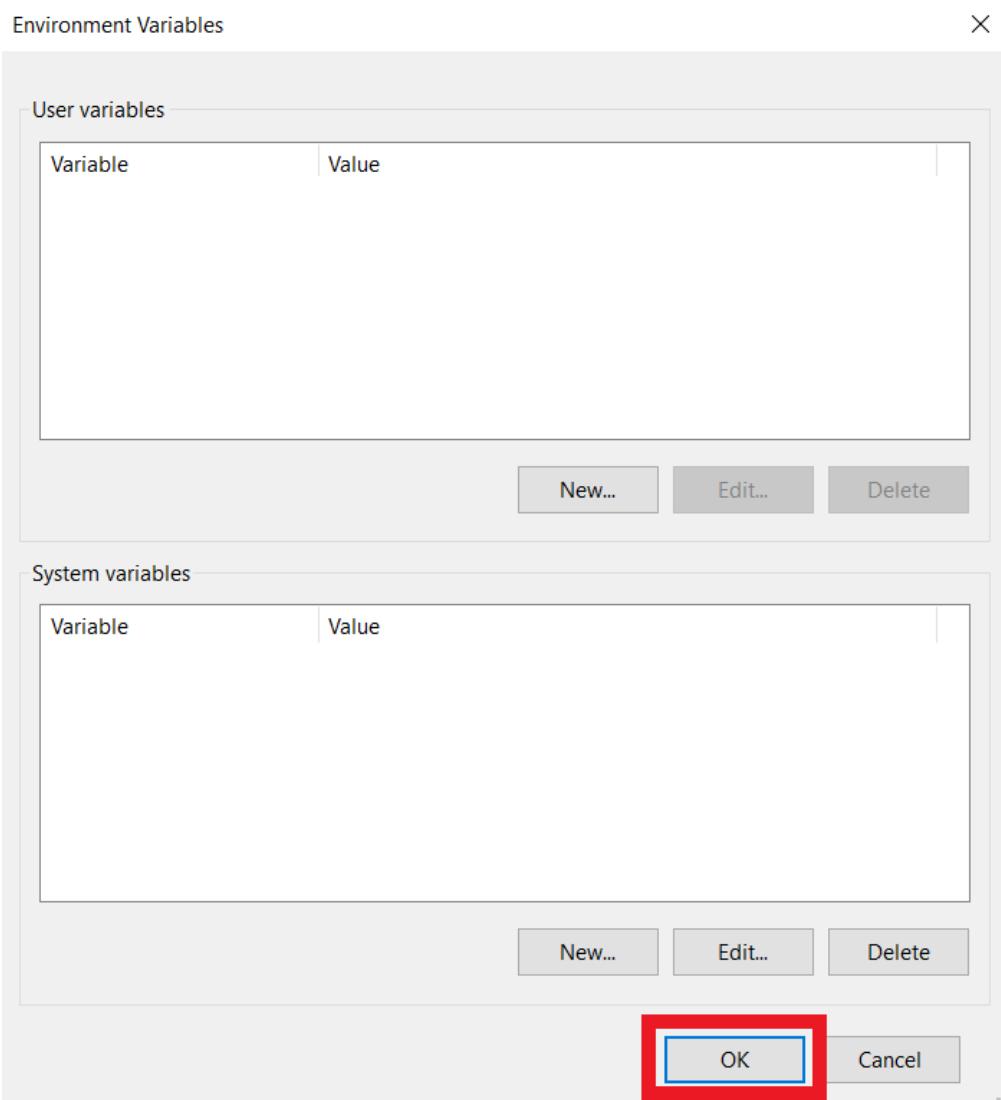


4. Enter the variable name as “DB_PASSWORD” and the key

Note: The key will be given once you have been assigned as an administrator



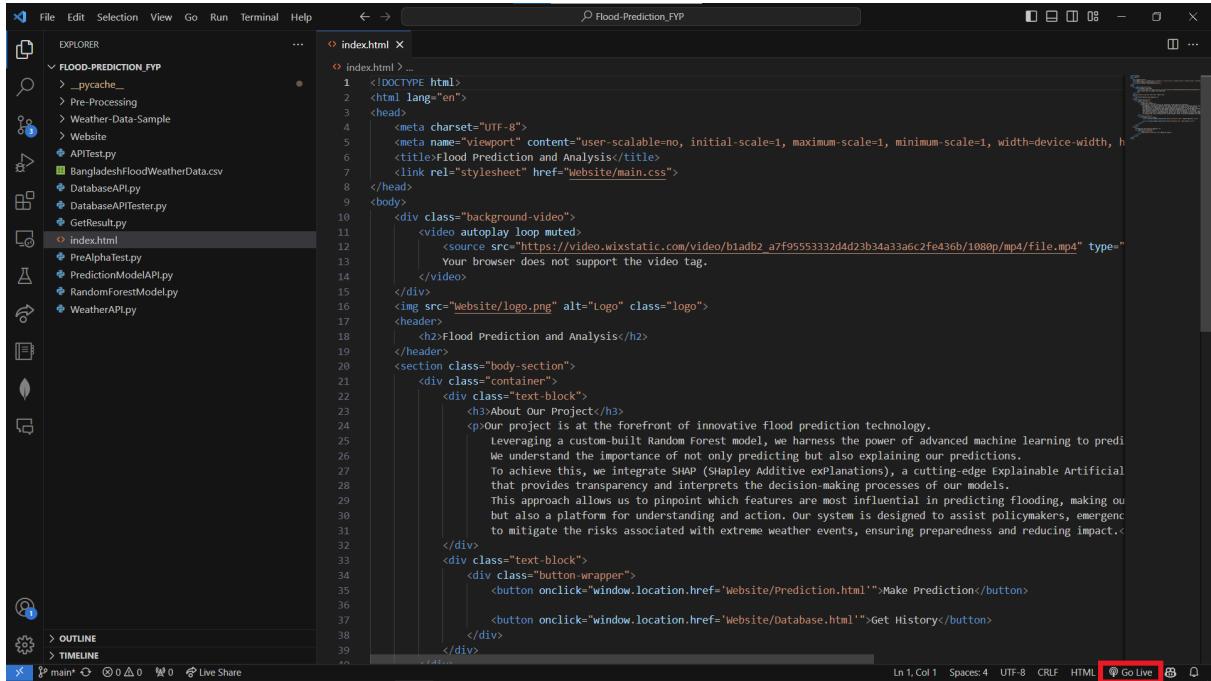
5. Click OK



2.5 Running the Application as a Whole

There are 3 steps to run the application:

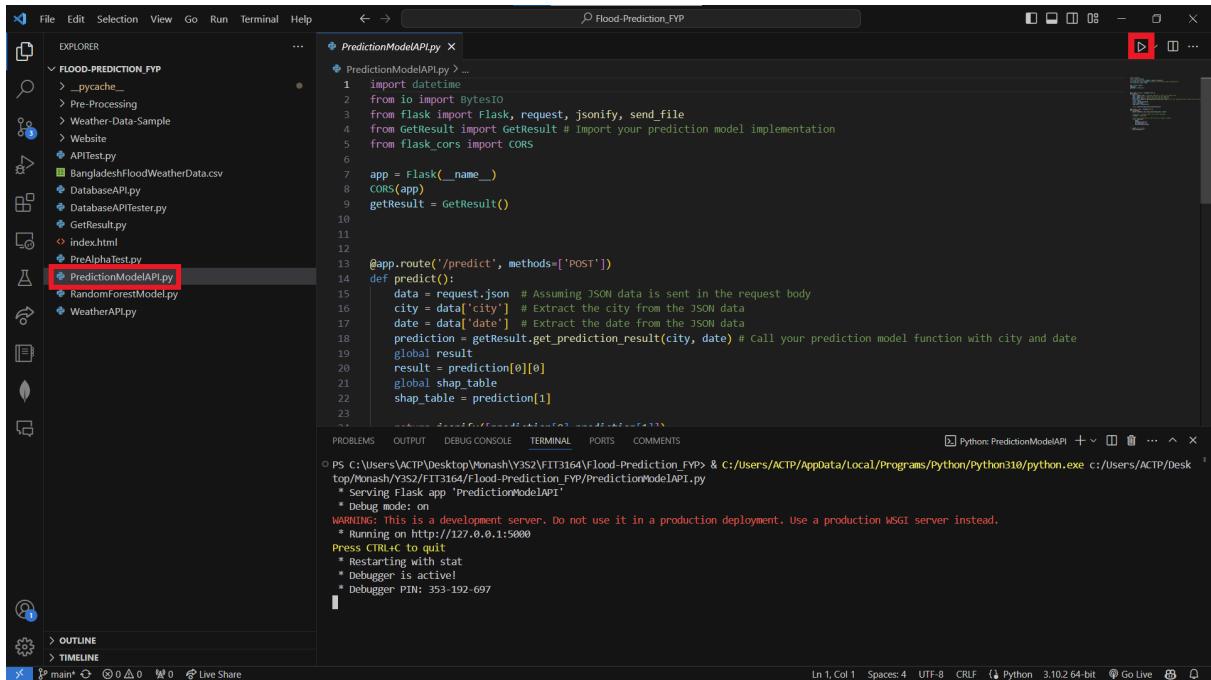
1. Run the live server to host the webpage



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width, height=device-height">
    <title>Flood Prediction and Analysis</title>
    <link rel="stylesheet" href="Website/main.css">
</head>
<body>
    <div class="background-video">
        <video autoplay loop muted>
            <source src="https://video.wixstatic.com/video/b1adb2_a7f9553332d4d23b34a33a6c2fe436b/1080p/mp4/file.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
    
    <header>
        <h2>Flood Prediction and Analysis</h2>
    </header>
    <section class="body-section">
        <div class="container">
            <div class="text-block">
                <h3>About Our Project</h3>
                <p>Our project is at the forefront of innovative flood prediction technology. Leveraging a custom-built Random Forest model, we harness the power of advanced machine learning to predict flooding. We understand the importance of not only predicting but also explaining our predictions. To achieve this, we integrate SHAP (SHapley Additive exPlanations), a cutting-edge Explainable Artificial Intelligence (XAI) technique that provides transparency and interprets the decision-making processes of our models. This approach allows us to pinpoint which features are most influential in predicting flooding, making our system not only a powerful tool for emergency management but also a platform for understanding and action. Our system is designed to assist policymakers, emergency responders, and the public in mitigating the risks associated with extreme weather events, ensuring preparedness and reducing impact.</p>
            </div>
            <div class="text-block">
                <div class="button-wrapper">
                    <button onclick="window.location.href='Website/Prediction.html'">Make Prediction</button>
                    <button onclick="window.location.href='Website/Database.html'">Get History</button>
                </div>
            </div>
        </div>
    </section>
</body>

```

2. Run the Prediction Model API



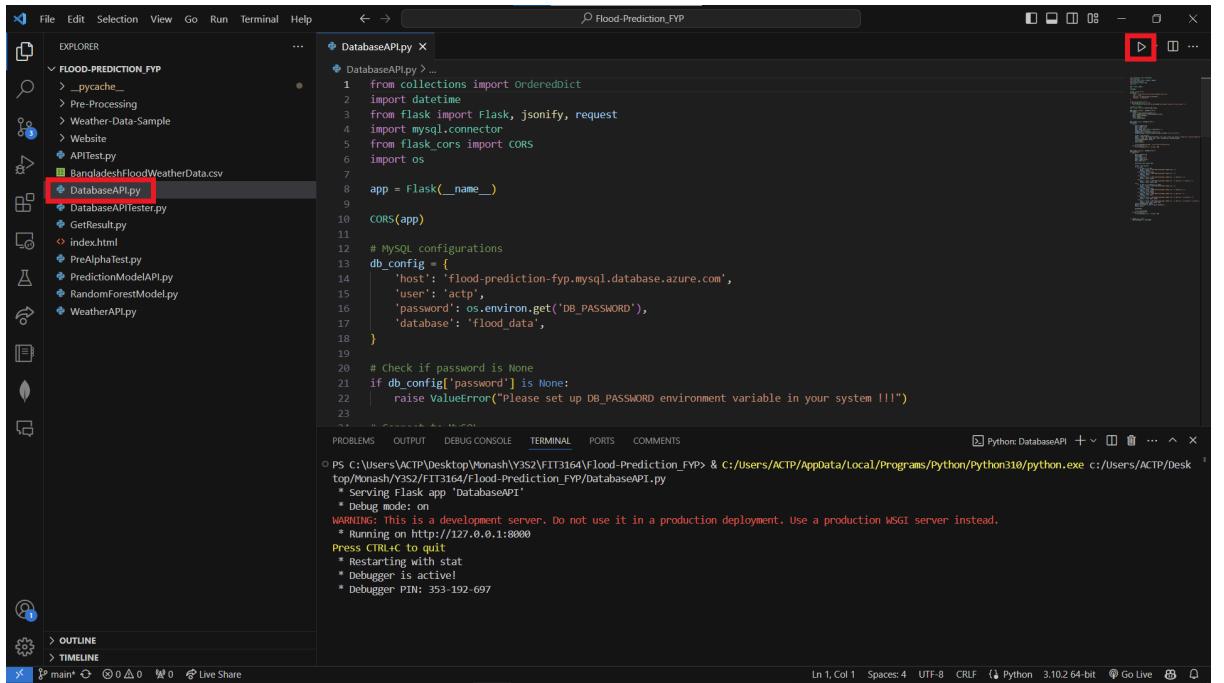
```
import datetime
from io import BytesIO
from flask import Flask, request, jsonify, send_file
from getResult import getResult # import your prediction model implementation
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
getResult = getResult()

@app.route('/predict', methods=['POST'])
def predict():
    data = request.json # Assuming JSON data is sent in the request body
    city = data['city'] # Extract the city from the JSON data
    date = data['date'] # Extract the date from the JSON data
    prediction = getResult.get_prediction_result(city, date) # Call your prediction model function with city and date
    global result
    result = prediction[0]
    global shap_table
    shap_table = prediction[1]

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\ACTP\Desktop\Monash\Y3S2\FTT316\Flood-Prediction_FYP> & C:/Users/ACTP/AppData/Local/Programs/Python/Python310/python.exe c:/Users/ACTP/Desktop/Monash/Y3S2/FTT316/Flood-Prediction_FYP/PredictionModelAPI.py
* Serving Flask app "PredictionModelAPI"
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 353-192-697
```

3. Run the Database API



```
DatabaseAPI.py
1  from collections import OrderedDict
2  import datetime
3  from flask import Flask, jsonify, request
4  import mysql.connector
5  from flask_cors import CORS
6  import os
7
8  app = Flask(__name__)
9
10 CORS(app)
11
12 # MySQL configurations
13 db_config = {
14     'host': 'flood-prediction-fyp.mysql.database.azure.com',
15     'user': 'actp',
16     'password': os.environ.get("DB_PASSWORD"),
17     'database': 'flood_data',
18 }
19
20 # Check if password is None
21 if db_config['password'] is None:
22     raise ValueError("Please set up DB_PASSWORD environment variable in your system !!!")
23
```

The screenshot shows the Visual Studio Code interface with the 'DatabaseAPI.py' file open in the center editor pane. The file contains Python code for a Flask application. The code imports necessary modules, sets up a Flask app with CORS, and defines a database configuration dictionary. A warning is present about the development server. The bottom status bar indicates the file has 1 line, 1 character, and is using UTF-8 encoding.

FIT3164 - Data Science Software Project

Software Test / QA Report

Prepared by MDS20
Prateek Tripathi 31835643
Lim Jun Yi 32625510
Cheng Tze Pin 32870663

Table of Contents

1. Introduction	3
2. Description of Test Approach	4
3. Black Box Testing	5
3.1: Prediction Page	5
3.2: Database Page	7
4. Integration Testing	9
5. Usability Testing	18
6. Recommendation for improvements	22
7. Limitation of Testing	23
8. Appendix	24

1. Introduction

In the evolving landscape of software development, assuring the quality and dependability of supplied products is critical. The main objective of our project is to build a predictive model that can predict floods, and be able to explain the key factors responsible for the prediction. To do this, we use the random forest prediction model and SHAP explainable AI (XAI) as they seem to be the candidates that offer the best and most consistent results for the project. The outcome of the project is provided in the form of a website, where a user can specify location and date to get the predictions required. The Test Report provided below is a cornerstone in our pursuit of excellence, precisely developed to assess the software's performance against set requirements and specifications. This study attempts to give a transparent and thorough review of the software's functionality, robustness, and user experience by employing rigorous testing procedures and detailed analyses.

Through this report, we aim to show that the developed software matches the requirements set for the software.

2. Description of Test Approach

The approach implemented by our team on our software is Manual Testing. This approach was chosen over an automated testing approach because the software is still relatively in its beginning stages and thus would be unable to profit from automated testing platforms like PyTest, or other such platforms. Our testing is done in 3 parts: Black Box Testing, Integration Testing and lastly Usability Testing. The Black Box testing is mainly conducted on our website. It focuses on the functional aspect of the software, ensuring that all specified functionalities work as expected. We can validate core features, like user input handling, API integrations, and data retrieval by simulating various user scenarios and comparing the outputs with the expected results. The aim of conducting the integration testing is to ensure all components of the software work together seamlessly. This consists of testing interactions between the front end and back end, communicating with other APIs, and integrating the prediction model into the user experience. Lastly, usability testing is conducted to ensure that the software is intuitive, easy to navigate, and provides a satisfactory user experience. This is conducted by volunteers, who shall be given the link to our website along with our user guide and their comments and feedback are taken into account.

3. Black Box Testing

3.1: Prediction Page

The **test input screenshots** and the **results screenshots** are added in the **appendix** with the appropriate **S. No.**

Test S. No	What is being tested?	How is it being tested?	What are the expected outputs?	What are the actual outputs being observed?
3.1	Error handling for invalid date input.	Submitting invalid input, such as not selecting the date and verifying the system's response.	Upon clicking the submit button an error message is shown that provides a suggested action for the user to take.	After clicking the submit button with an empty date field, the user is prompted to select a date.
3.2	Error handling for invalid location input.	Submitting invalid input, such as not selecting a location and verifying the system's response.	Upon clicking the submit button an error message is shown that provides a suggested action for the user to take.	After clicking the submit button with an empty location field, the user is prompted to select a location.
3.3	Display of results for the specific prediction.	Submitting a selected location, and a specified date, then verifying the system's response.	A text stating the prediction result and the accuracy of the prediction between 0 and 1, where 1 stands for 100% accuracy and 0 stands for 0% accuracy.	After submitting the input, a valid prediction made using the prediction model is shown in the result section, with a confidence score alongside it.
3.4	Display of SHAP graph for a specific prediction.	Submitting a selected location, and a specified date, then verifying the system's response	Upon submitting the input, the SHAP graph should be displayed, illustrating the contribution of each feature to the prediction outcome.	After submitting the input, a valid SHAP graph for the selected location and prediction made is outputted

Test S. No	What is being tested?	How is it being tested?	What are the expected outputs?	What are the actual outputs being observed?
3.5	Display of SHAP table for a specific prediction.	Submitting a selected location, and a specified date, then verifying the system's response.	Upon submitting the input, the SHAP table should be displayed, illustrating the contribution of each feature to the prediction outcome.	After submitting the input, a valid SHAP table for the selected location and prediction made is outputted.

3.2: Database Page

Similarly to the above section, the **test input screenshots** and **results screenshots** are added in the **appendix** with the appropriate **S. No.**

Test S. No	What is being tested?	How is it being tested?	What are the expected outputs?	What are the actual outputs being observed?
3.6	Retrieval of flood prediction history for a specific month.	Submitting a selected month, and verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified month that have been stored in the database will be shown	After submitting the month, the user is shown the predictions made related to the specified month, regardless of location
3.7	Retrieval of flood prediction history for a specific month and year.	Submitting a selected month and year, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified month in the specified year, that have been stored in the database, will be shown.	After submitting the specified month and the year, the user is shown the predictions made related to the specified details, regardless of location.
3.8	Retrieval of flood prediction history for a specific year.	Submitting a selected year, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified year, that have been stored in the database, will be shown.	Upon clicking the submit button, all the predictions made for the specified year, that have been stored in the database, will be shown.
3.9	Retrieval of flood prediction history for a specific date.	Submitting a selected date, then verifying the system's response.	Upon clicking the submit button, all the predictions for the specified date, in this case, 1 st May 2024, that have been stored in the database, will be shown.	After submitting the specified date, the user is shown the predictions made related to the specified date, regardless of location
3.10	Retrieval of flood prediction history for a specified location in a specific month.	Submitting a selected location, and a selected month, with appropriate inputs, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified location in the specified month, that have been stored in the database, will be shown.	After submitting the specified month and location, the user is shown the predictions made in the specified month, for the specified location.

Test S. No	What is being tested?	How is it being tested?	What are the expected outputs?	What are the actual outputs being observed?
3.11	Retrieval of flood prediction history for a specified location in a specific month and year.	Submitting a selected location, and a selected month and year, with appropriate inputs, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified location in the specified month and year, that have been stored in the database, will be shown.	After submitting the specified location, month and year, the user is shown the predictions made in the specified month and year, for the specified location.
3.12	Retrieval of flood prediction history for a specified location in a specific year.	Submitting a selected location, and a selected year, with appropriate inputs, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified location in the specified year, that have been stored in the database, will be shown.	After submitting the specified location and year, the user is shown the predictions made in the specified year, for the specified location.
3.13	Retrieval of flood prediction history for a specified location on a specific date.	Submitting a selected location, and a selected date, with appropriate inputs, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified location on the specified date, that have been stored in the database, will be shown.	After submitting the specified location and date, the user is shown the predictions made on the specified date, for the specified location.
3.14	Retrieval of flood prediction history for a specified location.	Submitting a selected location, with appropriate inputs, then verifying the system's response.	Upon clicking the submit button, all the predictions made for the specified location, that has been stored in the database, will be shown.	After submitting the specified location, the user is shown all the predictions made for the specified location that are available in the database.

4. Integration Testing

Integration testing is an essential step in the software testing life cycle that focuses on ensuring that different modules and components of a system work together seamlessly. The goal of integration testing is to identify any discrepancies or issues that may arise when individual units of the application are combined and to validate that they work together as expected.

Given the complexity of our application, which involves making API calls to retrieve weather information, processing this data through a trained prediction model, and displaying the results on the user interface, integration testing helps confirm that all these components function cohesively.

The key objectives of conducting the integration testing are:

1. To confirm that the inputs, provided by the user are accurately received and processed by the backend.
2. To confirm that the back-end results are correctly displayed in the front-end.
3. To ensure any errors or exceptions are handled gracefully and communicated properly to the user.

Test Execution:

Case 1: Prediction Request

Objective: Verify that the prediction request sent from the front-end is correctly received and processed by the back-end and that the prediction result is displayed accurately on the website. Additionally, test to compare the SHAP table and graph output to ensure they align with the prediction results.

Input: {"city": "Dhaka", "date": "2024-05-17"}

The result on the website:

The screenshot shows a web application interface. At the top, there is a header with the word "Prediction". Below it is a form with two fields: "Choose a location:" followed by a dropdown menu containing "Dhaka", and "Date:" followed by a date input field showing "17 - 05 - 2024" and a calendar icon. To the right of the date input is a "Submit" button. Below the form, the word "Prediction Result" is displayed. A blue callout box contains the text "Prediction Result: ["no flood",0.925]" and "{\"message\":\"Data stored successfully\"}".

SHAP Table

The table lists SHAP (SHapley Additive exPlanations) values for various features in a predictive model, indicating how each feature contributes to the model's predictions. A positive SHAP value means a feature increases the prediction, while a negative value indicates a decrease. The magnitude of these values shows the strength of their impact. The features range from continuous variables like temperature to categorical variables like geographical locations, each affecting the model's output uniquely. By analyzing these values, we can understand which features are most important and how they influence the predictions.

Attributes	SHAP Value
Year	-0.032785166650268614
Month	0.00485640412325707
Max_Temp (°C)	0.007289811799944202
Min_Temp (°C)	-0.046768707547127204
Avg_Temp (°C)	-0.013590238911768773
Rainfall (mm)	0.08542030978541362
Relative_Humidity (%)	0.0293563114280464
Wind_Speed (m/s)	-0.01020475202618718
Cloud_Coverage (Okta)	0.05418178662060897
LATITUDE	0.012209533251422692
LONGITUDE	0.004562488998769671
ALTITUDE (m)	0.007578035933853639
Barisal	0.0018004175423582452
Bhola	-0.00022031705726421354
Bogra	-0.0002950130290760619
Chandpur	0.0008168883746556891
Chittagong (City-Ambagan)	-6.440060356662434e-06
Chittagong (IAP-Patenga)	0.0001290078857151203
Comilla	0.0002356309997162605
Cox's Bazar	-0.00015559236073517215
Dhaka	0.018342897935195373
Dinajpur	0.0033350728810156837
Faridpur	7.024255623783972e-05
Feni	0.00022611269523244067
Hatiya	0.00013424720086465067
Ishurdi	-0.0006756957299972381
Jessore	-0.00023127029684351282
Khepupara	-4.0371005262369476e-05
Khulna	0.000256869518484564
Madaripur	7.526346196346176e-05
Majdee Court	0.000279639065006994
Mongla	7.18048324936524e-05
Mymensingh	0.00012171347551038032
Patuakhali	-2.173187567451481e-06
Rajshahi	0.001151619167943617
Rangamati	-7.616857094297193e-05
Rangpur	4.5142167273525725e-06
Sandwip	-0.00011003839740715541
Satkhira	0.0001797183926550974
Sitakunda	-3.5703375149143216e-05
Sylhet	0.00037497505105241267
Tangail	2.396214167047119e-06
Teknaf	6.436241690373735e-05

API test code:

```
import requests
from PIL import Image
from io import BytesIO

# Define the URLs of your Flask API endpoints
predict_url = 'http://127.0.0.1:5000/predict'
graph_url = 'http://127.0.0.1:5000/graph'

# Define the input data (city and date)
data = {
    'city': 'Dhaka',
    'date': '2024-05-17'
}

# Send a POST request to the Flask API endpoint to get the prediction
response = requests.post(predict_url, json=data)

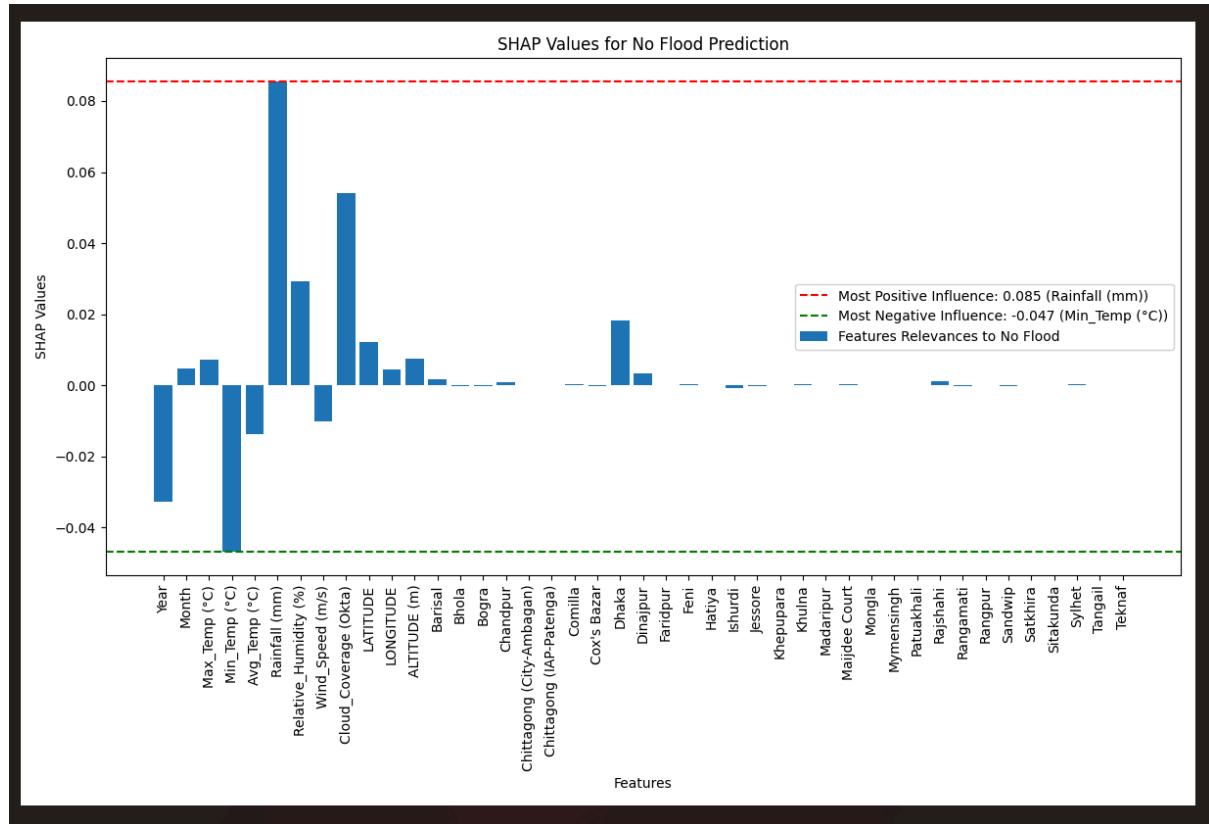
# Check if the prediction request was successful (status code 200)
if response.status_code == 200:
    # Print the predicted result
    prediction = response.json()
    print('Prediction Result:', prediction)

    # If the prediction was successful, fetch the SHAP graph
    graph_response = requests.get(graph_url)

    # Check if the graph request was successful (status code 200)
    if graph_response.status_code == 200:
        # Display the SHAP graph using PIL
        img = Image.open(BytesIO(graph_response.content))
        img.show() # This will open the default image viewer and display the image
    else:
        # Print an error message if the graph request was not successful
        print('Error fetching graph:', graph_response.text)
else:
    # Print an error message if the prediction request was not successful
    print('Error fetching prediction:', response.text)
```

API test result:

```
PS C:\Users\Prateek Tripathi\Desktop\Flood prediction\Flood-Prediction_FYP> & "C:/Users/Prateek Tripathi/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Prateek Tripathi/Desktop/Flood prediction/Flood-Prediction_FYP/APITest.py"
Prediction Result: [['no flood', 0.925], [['Year', '-0.032785166650268614', '0.032785166650268774'], ['Month', '0.00485640412325707', '-0.0048564041232570555'], ['Max_Temp (°C)', '0.007289811799944202', '-0.007289811799944146'], ['Min_Temp (°C)', '-0.046768707547127204', '0.04676870754712772'], ['Avg_Temp (°C)', '-0.013590238911768773', '0.013590238911768871'], ['Rainfall (mm)', '0.08542030978541362', '-0.08542030978541253'], ['Relative_Humidity (%)', '0.0293563114280464', '-0.029356311428046337'], ['Wind_Speed (m/s)', '-0.01020475202618718', '0.010204752026187246'], ['Cloud_Coverage (Okta)', '0.05418178662060897', '-0.05418178662060899'], ['LATITUDE', '0.012209533251422692', '-0.012209533251422915'], ['LONGITUDE', '0.004562488998769671', '-0.0045624889987697015'], ['ALTITUDE (m)', '0.007578035933853639', '-0.007578035933853607'], ['Barisal', '0.0018004175423582452', '-0.0018004175423582773'], ['Bhola', '-0.00022031705726421354', '0.0002203170572642142'], ['Bogra', '-0.0002950130290760619', '0.00029501302907605667'], ['Chandpur', '0.0008168883746556891', '-0.000816888374655692'], ['Chittagong (City-Ambagan)', '-6.440060356662434e-06', '6.440060356662149e-06'], ['Chittagong (AP-Patenga)', '0.0001290078857151203', '-0.00012900788571511892'], ['Comilla', '0.0002356309997162605', '-0.00023563099971625907'], ['Cox's Bazar', '-0.0001559236073517215', '0.0001559236073516914'], ['Dhaka', '0.018342897935195373', '-0.018342897935195418'], ['Dinajpur', '0.0033350728810157006'], ['Faridpur', '7.024255623783972e-05', '-7.024255623784604e-05], ['Feni', '0.00022611269523244067', '-0.0002261126952324404'], ['Hatiya', '0.00013424720086465067', '-0.00013424720086464907], ['Ishurdi', '-0.0006756957299972381', '0.000675695729997237'], ['Jessore', '-0.00023127029684351282', '0.00023127029684349816], ['Khepupara', '-4.0371005262369476e-05', '4.037100526237481e-05], ['Khulna', '0.00256869518484564', '-0.0002568695184845711], ['Madaripur', '7.526346196346176e-05', '-7.52634619634605e-05], ['Majdee Court', '0.000279639065005994', '-0.0002796390650069902], ['Mongla', '7.18048324936524e-05', '-7.18048324936599e-05], ['Myrmensingh', '0.00012171347551038032', '-0.00012171347551038198], ['Patuakhali', '-2.173187567451481e-06', '2.1731875674485807e-06], ['Rajshahi', '0.001151619167943617', '-0.0011516191679436206], ['Rangamati', '-7.616857094297193e-05', '7.616857094297193e-05], ['Rangpur', '4.5142167273525725e-06', '-4.514216727350071e-06], ['Sandwip', '0.00011003839740715541', '0.00011003839740715449], ['Satkhira', '0.0001797183926550974', '-0.00017971839265509934], ['Sitakunda', '-3.5703375149143216e-05', '-3.570337514914816e-05], ['Sylhet', '0.00037497505105241267', '-0.00037497505105241413], ['Tangail', '2.396214167047119e-06', '-2.396214167047119e-06], ['Teknaf', '6.436241690373735e-05', '-6.436241690373738e-05']]]
```



Status: Passed

As shown above, the prediction request made from the website and the manually executed API test both result in the same prediction output as well as the SHAP table and graph. This confirms that the front-end and back-end are integrated successfully and communicate effectively, ensuring that the flood prediction feature works as intended.

Case 2: Database storing and Retrieval

Objective: Verify that the prediction data is accurately retrieved and correctly stored in the database. As in the previous case, this test will involve first checking the results displayed on the website, and then confirming the same results by manually sending the inputs to the API. Additionally, we will check if the prediction in the database includes the correct timestamp to ensure that the prediction is made and recorded accurately in the database. This will ensure consistency between the frontend display and the backend data storage.

Input: {"city": "Sylhet", "date": "2024-05-28"} (The data specified is prediction date)

Time: "7:45 AM" | Date (generated): "2024-05-17"

The result on the website:

Database storing (website)

The screenshot shows a "Prediction" form with "Choose a location: Sylhet" and "Date: 28 - 05 - 2024". Below it, a "Prediction Result" box displays: "Prediction Result: ["no flood", 0.95] {"message": "Data stored successfully"}".

The screenshot shows a "Flood Prediction & Analysis" page with a "Prediction" form and a "Prediction Result" box (same as above). Below is a "SHAP Graph" section with the following text: "This SHAP plot visualizes the impact of different features on a model predicting the likelihood of flood events. Features above the horizontal axis positively influence the prediction, indicating a higher likelihood of this prediction result when these feature values increase. Conversely, features below the axis negatively affect the prediction. The magnitude of each bar shows the strength of each feature's impact on the model's output. This plot helps in understanding which variables are key drivers in the flood prediction model and how they relate to the likelihood of this prediction result." The SHAP Values for No Flood Prediction chart shows a single positive bar at approximately 0.127. A legend indicates: Most Positive Influence: 0.127 (Rainfall (mm)), Most Negative Influence: -0.033 (Min_Temp (*C)), and Features Relevances to No Flood.

Database Storing (database)

```
18 •   SELECT * FROM PREDICTION_RESULT ORDER BY PREDICTION MADE DESC;  
19
```

	City	Year	Month	Day	Result	Probability	Prediction_Made
▶	Chandpur	2024	5	29	"no flood"	0.94	2024-05-17 08:11:09
	Sylhet	2024	5	28	"no flood"	0.95	2024-05-17 07:45:28
	Bhola	2024	5	23	"no flood"	0.93	2024-05-17 04:26:29

Data Retrieval (website)

Database

Choose a location: Sylhet

Select Date Type: Specific Date

Select Specific Date: 28 - 05 - 2024

Submit

City	Year	Month	Day	Result	Probability	Prediction Made
Sylhet	2024	5	28	"no flood"	0.95	2024-05-17 07:45:28

Test Code:

```
• DatabaseAPITester.py > TestGetData > test_get_data
1 import unittest
2 import requests
3 import json
4
5 Codiumate: Add more tests
6 class TestGetData(unittest.TestCase):
7     BASE_URL = 'http://127.0.0.1:8000' # Replace with your actual server URL
8
9     Codiumate: Add more tests
10    def test_get_data(self):
11        # Define the data to send
12        data = [
13            'city': 'sylhet',
14            'year': '2024',
15            'month': '5',
16            'day': '28'
17        ]
18
19        # Send a POST request to the /get_data endpoint
20        response = requests.post(f'{self.BASE_URL}/get_data', json=data)
21
22        # Check that the status code is 200
23        self.assertEqual(response.status_code, 200)
24
25        # Check that the response is JSON
26        self.assertEqual(response.headers['Content-Type'], 'application/json')
27
28
29        # Parse the JSON response
30        response_data = response.json()
31
32        # Check that the response data is as expected
33        # This will depend on what your function is supposed to return
34        # Here's an example:
35        print(response_data)
36
37
38    if __name__ == '__main__':
39        unittest.main()
```

Test result:

```
PS C:\Users\Prateek Tripathi\Desktop\Flood prediction\Flood-Prediction_FYP> & "C:/Users/Prateek Tripathi/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Prateek Tripathi/Desktop/Flood prediction/Flood-Prediction_FYP/DatabaseAPITester.py"
[['Sylhet', 2024, 5, 28, "no flood", 0.95, '2024-05-17 07:45:28']]
.
-----
Ran 1 test in 0.145s
OK
```

Data Retrieval (Database)

```
18 •   SELECT *
19     FROM PREDICTION_RESULT
20     WHERE CITY = 'SYLHET'
21         AND YEAR = 2024
22         AND MONTH = 5
23         AND DAY = 28;
24
```

The screenshot shows a database result grid with the following columns: City, Year, Month, Day, Result, Probability, and Prediction_Made. The data row is as follows:

	City	Year	Month	Day	Result	Probability	Prediction_Made
▶	Sylhet	2024	5	28	"no flood"	0.95	2024-05-17 07:45:28

Status: Passed

As shown above, the prediction made from the website is successfully stored in the database, and correctly retrieved. This can be further confirmed by looking at the test case results where we are able to retrieve the new prediction made. We can confirm this by checking the timestamp of the prediction made. This confirms that the front-end and back-end are integrated successfully and communicate effectively, ensuring that the database feature works as intended.

5. Usability Testing

The purpose of this usability test is to evaluate the ease of use, efficiency, and overall user satisfaction of our flood prediction application. The following outlines the usability test for our application, focusing on the prediction and historical data retrieval.

The objectives we are trying to achieve are as follows:

- Assess how easily users can make flood predictions using the application.
- Evaluate how effectively users can interpret prediction results and the accompanying SHAP (SHapley Additive exPlanations) graphs and tables.
- Determine the ease with which users can retrieve and interpret historical flood data.
- Collect user feedback on the overall user experience and identify any usability issues.

The main group of participants representing the target user base are students in Monash University, Friends and Family members of the development team and any volunteers that are optioned to be included in this testing.

Our methodology:

- Type of Test: Remote usability testing/Onsite testing
- Location: Remote (participants' environments)/Onsite(developers' environments)
- Tools: Laptops, Internet, Python IDE/VSCode

Testing tasks:

Task 1: Making a Prediction

Scenario: You want to make a flood prediction for today in Dhaka.

Steps:

1. Click on the "Make Prediction" button.
2. Select "Dhaka" as the location for the prediction.
3. Select the current date (today) as the Date for prediction.
4. Interpret the prediction result.
5. Review the SHAP graph and SHAP table for an explanation of the prediction.

Success Criteria:

The user completes all steps without significant errors.

The user accurately interprets the prediction result and the SHAP explanations.

Task 2: Retrieving Historical Data

Scenario: You want to view historical flood data for Dhaka.

Steps:

1. Go back to the home page.
2. Click on the "Get History" button.
3. Select "Dhaka" as the location for historical data.
4. Select "Month and Year" as the filter criteria.
5. Select May 2024.
6. Interpret the historical data results.

Success Criteria:

The user successfully navigates to and uses the historical data retrieval feature.

The user accurately interprets the historical data.

Task 3: Learning About the Team

Scenario: You want to learn more about the development team behind the application.

Steps:

1. Go back to the home page.
2. Click on the "About Our Team" button.

Success Criteria:

The user successfully navigates to the "About Our Team" section and reviews the information.

Metrics

- **User Satisfaction:** Participant feedback was gathered through post-task and post-test questionnaires.
- **Ease of Use:** Ratings on a scale of 1 to 5 for each task.

Responses Results Table:

The table below shows the counts of each response and mean scores.

(details breakdown and visualisations are located in the appendix)

No	Question	1(Very Poor)	2(Poor)	3(Normal)	4(Good)	5(Very Good)	Mean
1	How would you rate your overall experience with the application?	1	0	2	4	3	3.8
2	The layout of the website is visually appealing.	1	2	1	3	3	3.5
3	The website loads quickly.	0	1	1	3	5	4.2
4	How easy was it to navigate through the application?	0	2	5	1	2	3.3
5	How easy was it to make a prediction?	0	0	2	1	7	4.5
6	How well do you understand the prediction results?	0	1	4	2	3	3.7

7	Were the SHAP graph and SHAP table helpful in explaining the prediction?	2	2	4	2	0	2.6
8	How easy was it to retrieve historical data?	0	0	1	5	4	4.3
9	How well do you understand the historical data results?	1	3	0	0	6	3.7
10	Were the historical data results presented in a clear and understandable manner?	0	2	3	3	2	3.5
11	How easy was it to find information about the development team?	1	2	3	1	3	3.3
12	Was the information about the team sufficient and helpful?	2	4	2	0	2	2.6

6. Recommendation for improvements

After performing the tests, it is crucial to understand where are the downsides of our application, based on the test results and user feedback. From that, there are a few improvements that can be made to enhance the user experience:

1. **SHAP table and graph:** As SHAP(SHapley Additive exPlanations) is not a common term and technique to explain the relevance of features, the value and the graph are not easy to understand by the public. From the usability test, the feature got a low mark on average, indicating that users do not understand how it works. Although we have a description of how to read the SHAP table and graph, it may seem too overwhelming and still does not work well. Hence, future work on this part should include a brief but more informative instruction to guide the user.
2. **Information about the team:** Based on the test results, the user does not seem to be satisfied with the information about the team provided with a low mark on average. Hence, besides focusing on the technical aspects, we should also focus on the information aspects in the future.

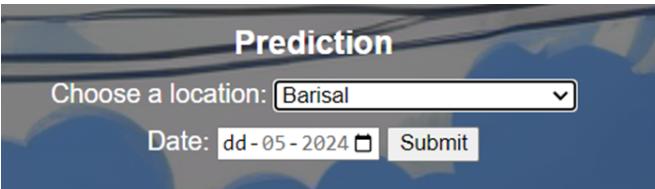
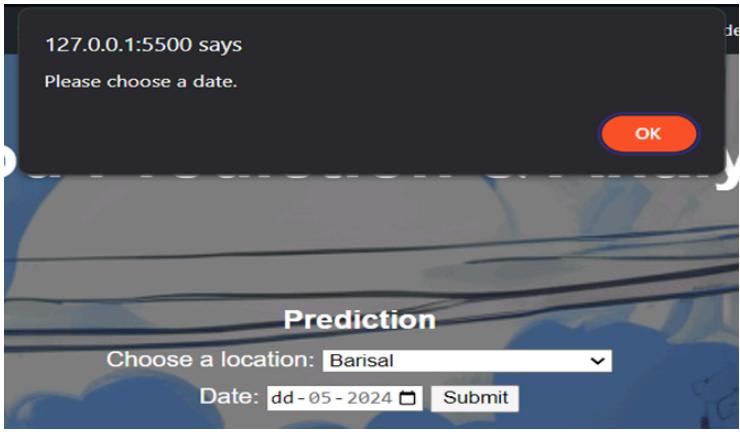
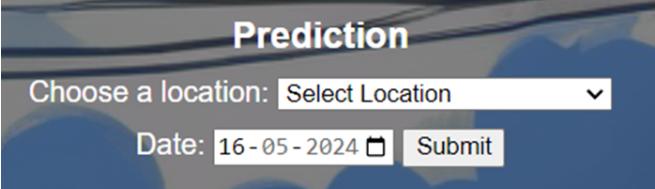
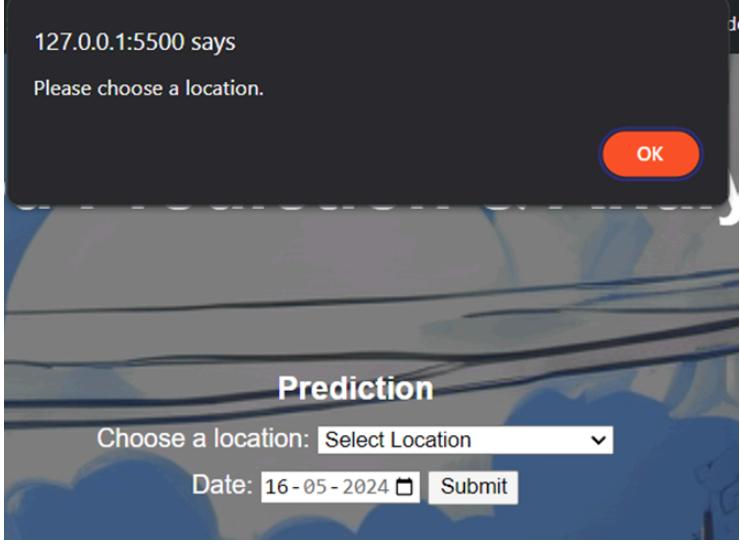
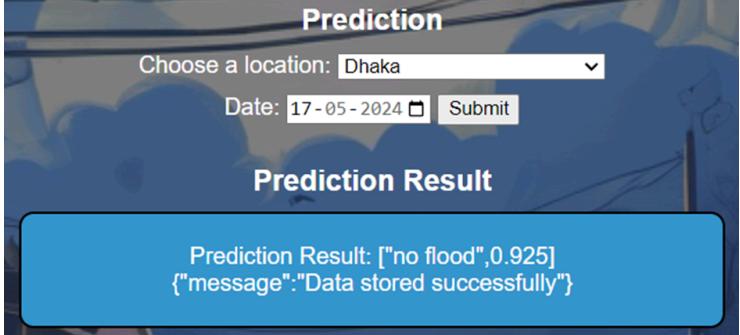
7. Limitation of Testing

Although the testing we performed has included every aspect of the application, there are still some limitations:

1. **Lack of Server Hosting:** The application has not been hosted on a live server. Hence, we are not able to simulate a real world where the application would be deployed and accessed by multiple users simultaneously.
2. **No Load Testing:** Due to the absence of server hosting, we are unable to perform load testing which is crucial for understanding how the application behaves under high-traffic conditions. Without this, we cannot actually gauge the performance, stability and responsiveness of the application when subjected to a large number of concurrent users.
3. **Limited Security Testing:** Although our application does not allow users to manually input anything, which reduces the risk of injection attacks, there are still scenarios where injections could occur, such as through manipulating URLs or other forms of indirect input.

8. Appendix

Screenshots of each Black Box test input and output.

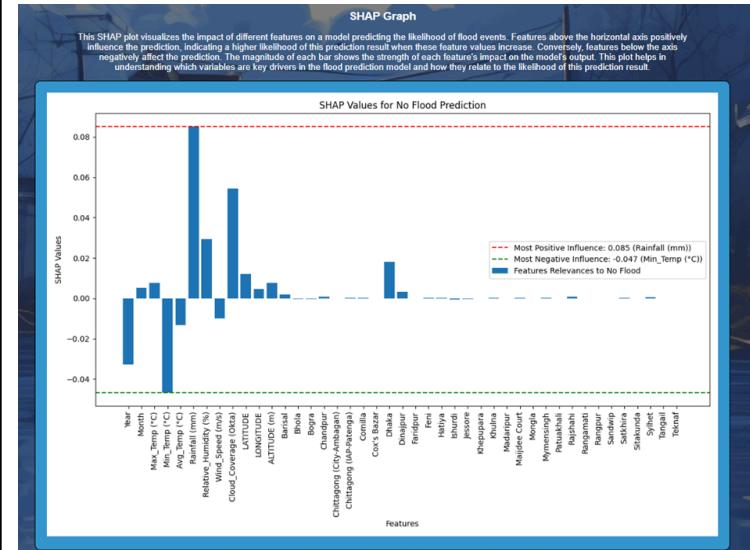
Test S. No	Input Being Tested	Output Result
3.1		
3.2		
3.3		

3.4

Prediction

Choose a location:

Date:



3.5

Prediction

Choose a location:

Date:

SHAP Table

The table lists SHAP (Shapley Additive ExPlanations) values for various features in a predictive model, indicating how each feature contributes to the model's predictions. A positive SHAP value means a feature increases the prediction, while a negative value indicates a decrease. The magnitude of these values shows the strength of their impact. The features range from continuous variables like temperature to categorical variables like geographical locations, each affecting the model's output uniquely. By analyzing these values, we understand which features are most important and how they influence the predictions.

Attributes	SHAP Value
Year	-0.032835986591372185
Month	0.0050665096881372185
Max_Temp (°C)	0.007116549526509398
Min_Temp (°C)	-0.04777052151231447
Avg_Temp (°C)	-0.01321668131275108
Rainfall (mm)	0.0850749562599278
Relative_Humidity (%)	0.002600040240200711
Wind_Speed (m/s)	0.007265454952327
Cloud_Coverage (Okta)	0.05435434576183776
LATITUDE	0.01295517138934331
LONGITUDE	0.00158252148045350
ALTITUDE (m)	0.00158252148045350
Barisal	0.00119167998133616
Bogra	0.00119167998133616
Chittagong	0.00081723281778576
Chittagong (City-Ambagan)	-6.440060356624346e-06
Chittagong (IAP-Patenga)	0.0012073787894499792
Cox's Bazar	-0.00115427135363734
Dhaka	0.01519605201694294
Dinajpur	0.0031814469069111043
Feni	0.00029207359126256005
Hatya	0.0001304659281873675
Jhurdi	-0.000675981715248028
Jessore	0.00029207359126256005
Khepupara	4.01090674694716e-05
Khulna	0.000257599589700127
Madijalpur	7.77984181793997e-06
Mongla	0.00029207359126256005
Mymensingh	7.191018906450774e-05
Pirojshahi	0.0001202414053942721
Rajshahi	-3.014175178517466e-06
Ramgarh	0.00029207359126256005
Rangpur	7.60732254191587e-05
Sandwip	3.424749124198604e-06
Satkhira	-9.7991763731859e-05
Shaklunda	0.00029207359126256005
Sylhet	-3.533613429361278e-05
Tangail	4.1562260695241865e-06
Teknaf	6.44917359581016e-05

3.6

Database

Choose a location:

Select Date Type:

Select Month:

Database

Choose a location:

Select Date Type:

Select Month:

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Dhaka	2024	5	1	"no flood"	0.9	2024-04-30 20:51:29
Bogra	2024	5	1	"no flood"	0.92	2024-04-30 20:55:28
Feni	2024	5	1	"no flood"	0.955	2024-04-30 21:01:30
Chittagong (City-Ambagan)	2024	5	2	"no flood"	0.92	2024-04-30 21:34:23
Chittagong (IAP-Patenga)	2024	5	3	"no flood"	0.9	2024-04-30 21:34:32

3.7

Database

Choose a location: Select All

Select Date Type: Month and Year

Select Month and Year: May , 2024

Database

Choose a location: Select All

Select Date Type: Month and Year

Select Month and Year: May , 2024

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Dhaka	2024	5	1	"no flood"	0.9	2024-04-30 20:51:29
Bogra	2024	5	1	"no flood"	0.92	2024-04-30 20:55:28
Feni	2024	5	1	"no flood"	0.955	2024-04-30 21:01:30
Chittagong (City-Ambagan)	2024	5	2	"no flood"	0.92	2024-04-30 21:34:23
Chittagong (IAP-Patenga)	2024	5	3	"no flood"	0.9	2024-04-30 21:34:32

3.8

Database

Choose a location: Select All

Select Date Type: Year Only

Select Year: 2024

Database

Choose a location: Select All

Select Date Type: Year Only

Select Year: 2024

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Dhaka	2024	5	1	"no flood"	0.9	2024-04-30 20:51:29
Bogra	2024	5	1	"no flood"	0.92	2024-04-30 20:55:28
Feni	2024	5	1	"no flood"	0.955	2024-04-30 21:01:30
Chittagong (City-Ambagan)	2024	5	2	"no flood"	0.92	2024-04-30 21:34:23
Chittagong (IAP-Patenga)	2024	5	3	"no flood"	0.9	2024-04-30 21:34:32
Jessore	2024	4	30	"no flood"	0.93	2024-04-30 21:35:34
Cox's Bazar	2024	5	1	"no flood"	0.965	2024-04-30 21:36:29
Hatiya	2024	4	30	"no flood"	0.955	2024-04-30 21:36:53

3.9

Database

Choose a location: Select All

Select Date Type: Specific Date

Select Specific Date: 01 - 05 - 2024

Submit

Database

Choose a location: Select All

Select Date Type: Specific Date

Select Specific Date: 01 - 05 - 2024

Submit

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Dhaka	2024	5	1	"no flood"	0.9	2024-04-30 20:51:29
Bogra	2024	5	1	"no flood"	0.92	2024-04-30 20:55:28
Feni	2024	5	1	"no flood"	0.955	2024-04-30 21:01:30
Cox's Bazar	2024	5	1	"no flood"	0.965	2024-04-30 21:36:29
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:18:36
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:59:36

3.10

Database

Choose a location: Barisal

Select Date Type: Month

Select Month: May

Submit

Database

Choose a location: Barisal

Select Date Type: Month

Select Month: May

Submit

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:18:36
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:59:36
Barisal	2024	5	3	"no flood"	0.9	2024-05-01 21:03:24
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:46
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:48

3.11

Database

Choose a location: Barisal

Select Date Type: Month and Year

Select Month and Year: May , 2024

Database

Choose a location: Barisal

Select Date Type: Month and Year

Select Month and Year: May , 2024

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:18:36
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:59:36
Barisal	2024	5	3	"no flood"	0.9	2024-05-01 21:03:24
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:46
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:48

3.12

Database

Choose a location: Barisal

Select Date Type: Year Only

Select Year: 2024

Database

Choose a location: Barisal

Select Date Type: Year Only

Select Year: 2024

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:18:36
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:59:36
Barisal	2024	5	3	"no flood"	0.9	2024-05-01 21:03:24
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:46
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:48

3.13

Database

Choose a location: Barisal

Select Date Type: Specific Date

Select Specific Date: 01-05-2024

Submit

Database

Choose a location: Barisal

Select Date Type: Specific Date

Select Specific Date: 01-05-2024

Submit

City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:18:36
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:59:36

3.14

Database

Choose a location: Barisal

Select Date Type: Select All

Submit

Database

Choose a location: Barisal

Select Date Type: Select All

Submit

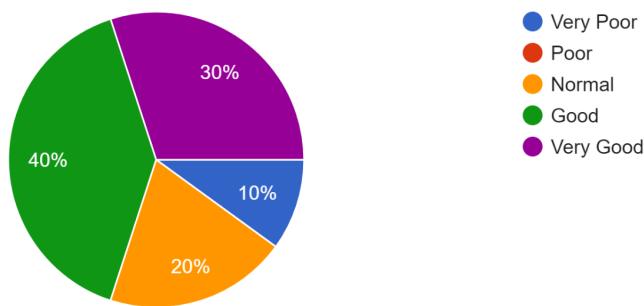
City	Year	Month	Day	Result	Probability	Prediction Made
Barisal	2024	5	1	"no flood"	0.925	2024-04-30 20:50:02
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:18:36
Barisal	2024	5	1	"no flood"	0.935	2024-05-01 18:59:36
Barisal	2024	5	3	"no flood"	0.9	2024-05-01 21:03:24
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:46
Barisal	2024	5	16	"no flood"	0.93	2024-05-13 14:51:48

Usability Testing Responses breakdown and visualisations

Form: <https://forms.gle/wy26cXUV53Txx7XJ8>

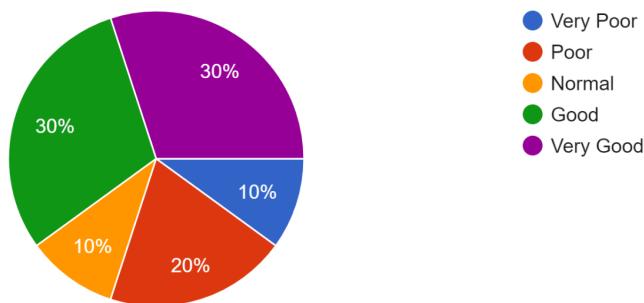
How would you rate your overall experience with the application?

10 responses



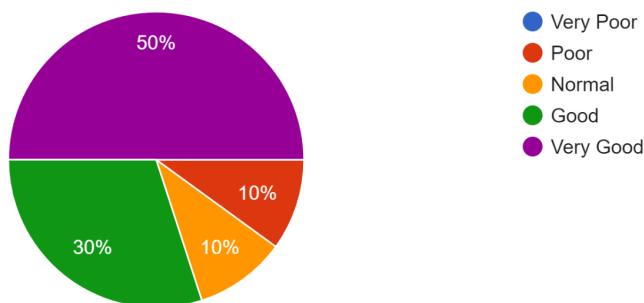
The layout of the website is visually appealing.

10 responses



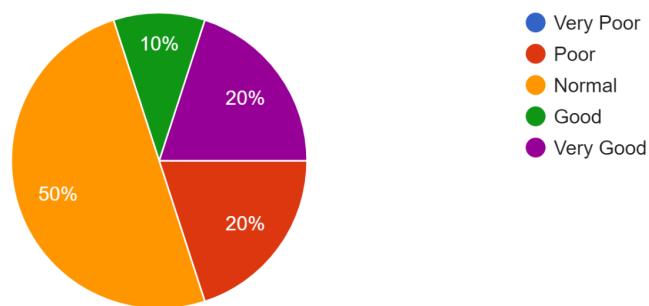
The website loads quickly.

10 responses



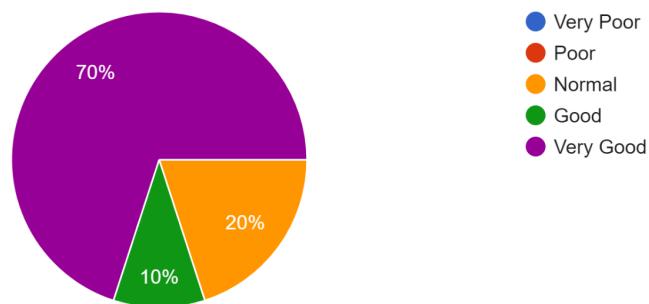
How easy was it to navigate through the application?

10 responses



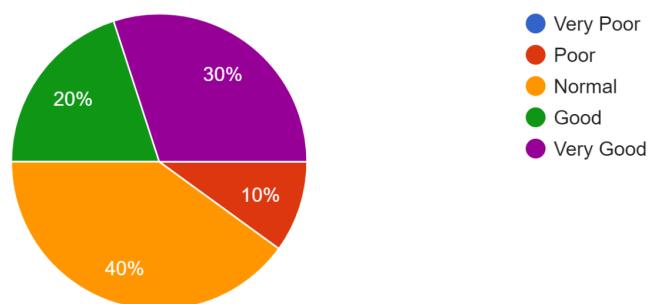
How easy was it to make a prediction?

10 responses



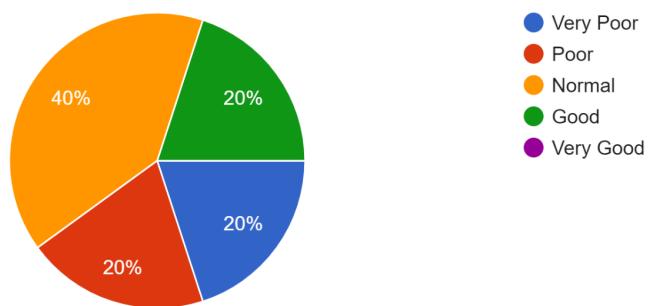
How well you understand the prediction results?

10 responses



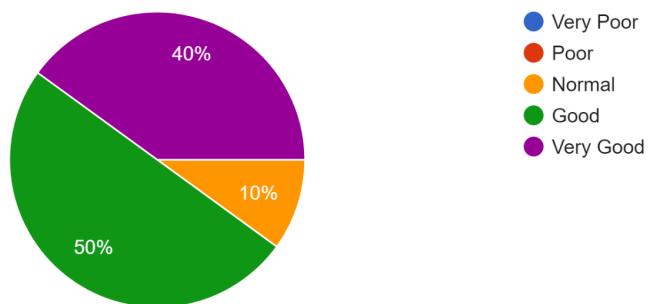
Were the SHAP graph and SHAP table helpful in explaining the prediction?

10 responses



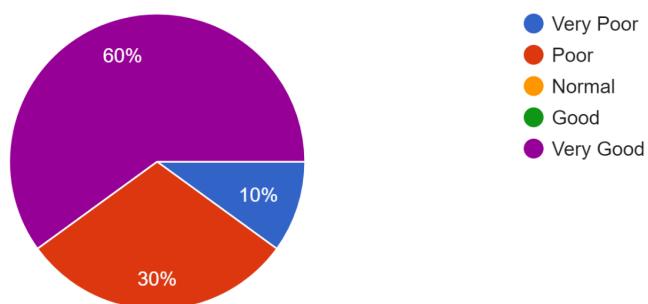
How easy was it to retrieve historical data?

10 responses



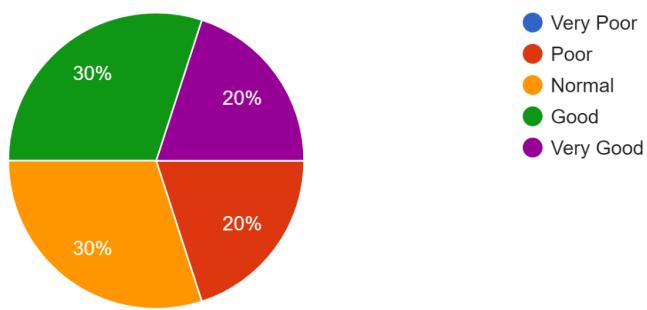
How well you understand the historical data results?

10 responses



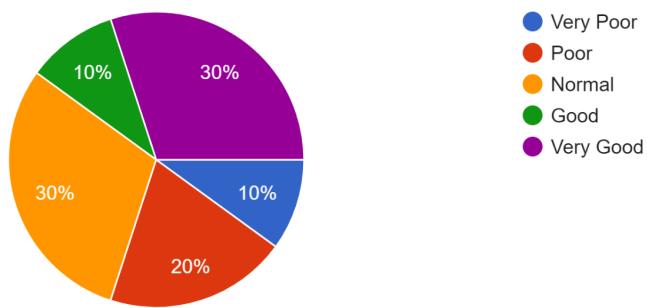
Were the historical data results presented in a clear and understandable manner?

10 responses



How easy was it to find information about the development team?

10 responses



Was the information about the team sufficient and helpful?

10 responses

