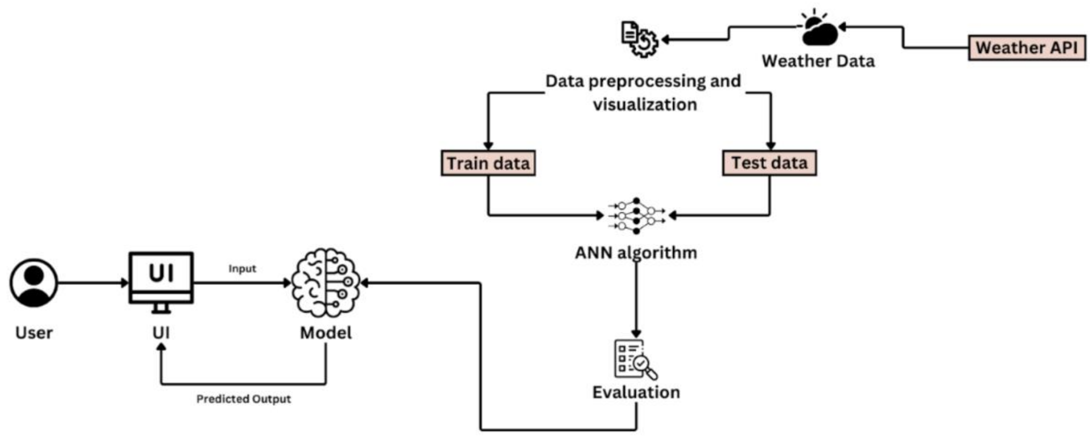
Rainfall Prediction

Team ID: PNT2023TMID-592801

Project Description:

As one of the world's most rainfall-prone countries, India relies heavily on accurate rainfall prediction. The livelihoods of numerous workers and the public's daily lives are closely intertwined with rainfall patterns. Ensuring safety, preparedness, and awareness is a critical need. This project addresses this need by exploring various predictive models, including XGBoost, Support Vector Classification (SVC), Logistic Regression, and Artificial Neural Networks (ANN). The goal is to select the most effective model and make it accessible to the public. The project's impact lies in delivering dependable rainfall predictions that empower individuals across India to make informed decisions, enhancing their safety, economic well-being, and overall quality of life.

Technical Architecture:



Pre-requisites:

1. Visual Studio Code
2. Python Packages: Flask-> pip install flask
   XGBoost-> pip install xgboost

   Type the given commands in VSCode powershell

Project Objectives:

1. Understand fundamental concepts of Logistic Regression, XGBoost and ANN
2. Know how to preprocess and clean the data
3. Understand how to build a simple web application using Flask

Project Flow:

1. The user interacts with UI which prompts the user to enter necessary values for the prediction
2. The details are sent to the model, which uses it to predict if it will rain or not. The model is integrated into the back-end of the flask file
3. The chosen model utilizes the user's input to predict whether rain is expected or not, and subsequently, displays the prediction on the Flask UI.

For the above workflow to be accomplished, we need to perform the following steps:

- Data Collection: https://www.kaggle.com/datasets/trisha2094/weatheraus/
- Data Cleaning and Preprocessing:
    - Import libraries-> pandas, numPy, seaborn
    - Import dataset
    - Check for null values and handle them
    - Perform Data visualisation
    - Handle outliers
    - Scale the data if needed
    - Split into train and test
- Model Building:
    - Import model building libraries
    - Initialize the models
    - Fit the model with the data
    - Evaluate all the models' performances'
    - Choose the best-performing model
    - Download the model as a pickle file
- Application building:
    - Create a front-end file using HTML and CSS
    - Create a backend file using Python Flask

Project Structure:

Create a project file which contains the following files

Today			
app	09-11-2023 17:42	Python source file	2 KB
model.pkl	09-11-2023 16:34	PKL File	432 KB
static	09-11-2023 17:40	File folder	
templates	09-11-2023 17:40	File folder	
rain	09-11-2023 16:37	File folder	

1) App.py is the server side which runs the model with the given data.
2) Model.pkl is the pickle file which contains the saved model
3) Template folder contains index.html, rain.html and norain.html

Milestone 1: Data Collection

While deciding the dataset it is crucial to take into consideration the quality of the dataset. For this project, we will be using weatherAUS dataset available at Kaggle.com.

Link to the dataset: https://www.kaggle.com/datasets/trisha2094/weatheraus/

Milestone 2: Data preprocessing and visualization

We will be building different models and comparing it on google colab and then we will download the model file.

1) Import the necessary libraries

```
[ ]  import pandas as pd
     import numpy as np
     import warnings
     warnings.filterwarnings("ignore")
```

2) Import the dataset as a dataframe

```
df=pd.read_csv('/content/weatherAUS.csv')
df.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9a
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	V
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNV
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	V
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE

3) Analyse the data:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 142193 entries, 0 to 142192
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           142193 non-null  object
 1   Location       142193 non-null  object
 2   MinTemp        141556 non-null  float64
 3   MaxTemp        141871 non-null  float64
 4   Rainfall       140787 non-null  float64
 5   Evaporation    81350 non-null   float64
 6   Sunshine       74377 non-null   float64
 7   WindGustDir    132863 non-null  object
 8   WindGustSpeed  132923 non-null  float64
 9   WindDir9am     132180 non-null  object
 10  WindDir3pm     138415 non-null  object
 11  WindSpeed9am   140845 non-null  float64
 12  WindSpeed3pm   139563 non-null  float64
 13  Humidity9am    140419 non-null  float64
```

```
numerical_attributes=[f for f in df.columns if df[f].dtypes!='O']
categorical_attributes=[f for f in df.columns if df[f].dtypes=='O']
discrete_val_attributes=[f for f in numerical_attributes if df[f].nunique()<25]
continuous_val_attributes=[f for f in numerical_attributes if f not in discrete_val_attributes]
print('numerical_attributes:',numerical_attributes)
print('categorical_attributes:',categorical_attributes)
print('discrete_val_attributes:',discrete_val_attributes)
print('continuous_val_attributes:',continuous_val_attributes)
```

```
numerical_attributes: ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm', 'Hu
categorical_attributes: ['Date', 'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm', 'RainToday', 'RainTomorrow']
discrete_val_attributes: ['Cloud9am', 'Cloud3pm']
continuous_val_attributes: ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed', 'WindSpeed9am', 'WindSpeed3pm'
```

4) Handle missing values:

```
df.isnull().sum()
```

```
Date                0
Location            0
MinTemp           637
MaxTemp           322
Rainfall         1406
Evaporation     60843
Sunshine        67816
WindGustDir      9330
WindGustSpeed    9270
WindDir9am      10013
WindDir3pm       3778
WindSpeed9am     1348
WindSpeed3pm     2630
Humidity9am      1774
Humidity3pm      3610
Pressure9am     14014
Pressure3pm     13981
Cloud9am        53657
Cloud3pm        57094
```

Method to fill in missing values is done by observing the distribution of each attribute and deciding which one of mean, mode and median is most suitable.

```
#The 75th percentile (Q3) and maximum values are relatively close for MinTemp,MaxTemp,Sunshine,Humidity9a
#Pressure9am,Pressure3pm,Cloud9am,Cloud3pm,Temp9am,Temp3pm
#this suggests that the data may not have an extremely skewed distribution.
import matplotlib.pyplot as plt
import seaborn as sns
thresh=22
small_dff=[]
for f in continuous_val_attributes:
  diff=df[f].max()-df[f].quantile(0.75)
  if diff<=thresh:
    small_dff.append(f)
print(small_dff)
```

```
fig,axes=plt.subplots(3,3,figsize=(15,10))
for i in range(len(small_dff)):
  att=small_dff[i]
  sns.distplot(df[att],ax=axes[i//3,i%3])
```

```
#MinTemp, MaxTemp,Temp9am and Temp3pm seem to have a normal distribution and thus we'll impute the null values with the mean
#the others will be imputed using median
norm=['MinTemp','MaxTemp','Temp9am','Temp3pm']
for i in norm:
  df[i]=df[i].fillna(df[i].mean())
```

```
df.isnull().sum()
```

```
rest=[]
for i in continuous_val_attributes:
  if i not in norm:
    rest.append(i)
print(rest)
```

```
fig,axes=plt.subplots(5,2,figsize=(15,20))
fig.tight_layout(pad=3.0) #padding for better spacing
for i in range(len(rest)):
    att=rest[i]
    sns.boxplot(df[att],ax=axes[i//2,i%2])
    axes[i//2,i%2].set_title(att)
plt.show()
```

```
for i in rest:
  df[i]=df[i].fillna(df[i].median())
```

```
df.isnull().sum()
```

```
discrete_val_attributes
```

```
for f in discrete_val_attributes:
  df[f]=df[f].fillna(df[f].mode()[0])
```

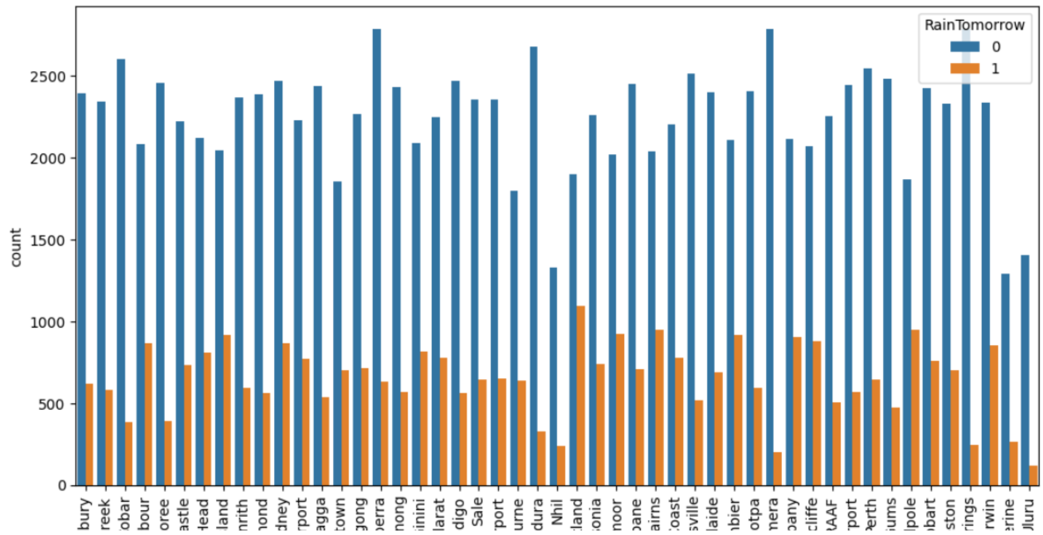
```
df[discrete_val_attributes].isnull().sum()
```

```
for i in ['WindGustDir','WindDir9am','WindDir3pm','RainToday']:
  df[i].fillna(df[i].mode()[0],inplace=True)
```

5) Perform data visualization to understand how change in the attributes affect the target variable. This can be done by find the correlation matrix, printing the heatmap and analyzing the pairplots.

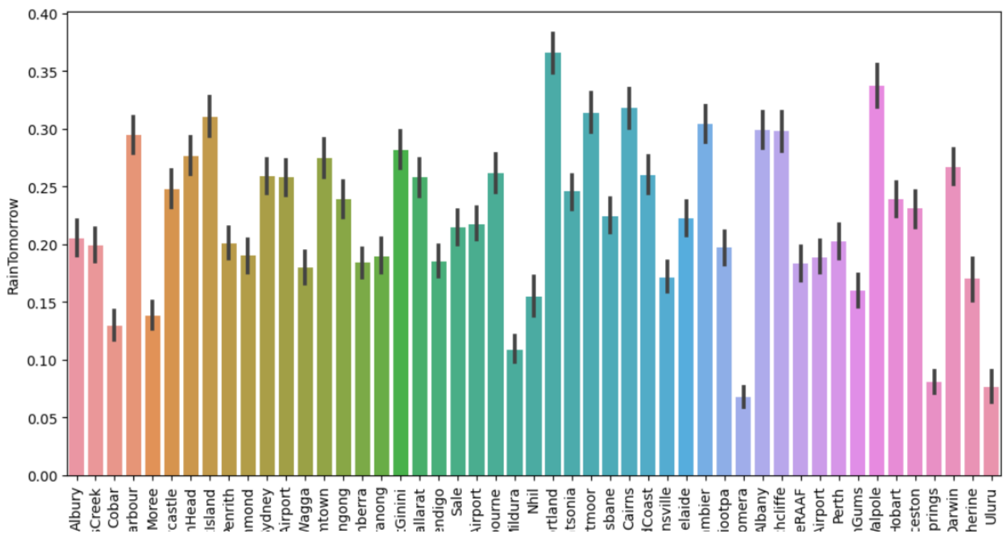
a. Countplot

```
plt.figure(figsize=(12, 6))
sns.countplot(x='Location', hue='RainTomorrow', data=df)
plt.xticks(rotation=90)
plt.show()
```



b. Barplot

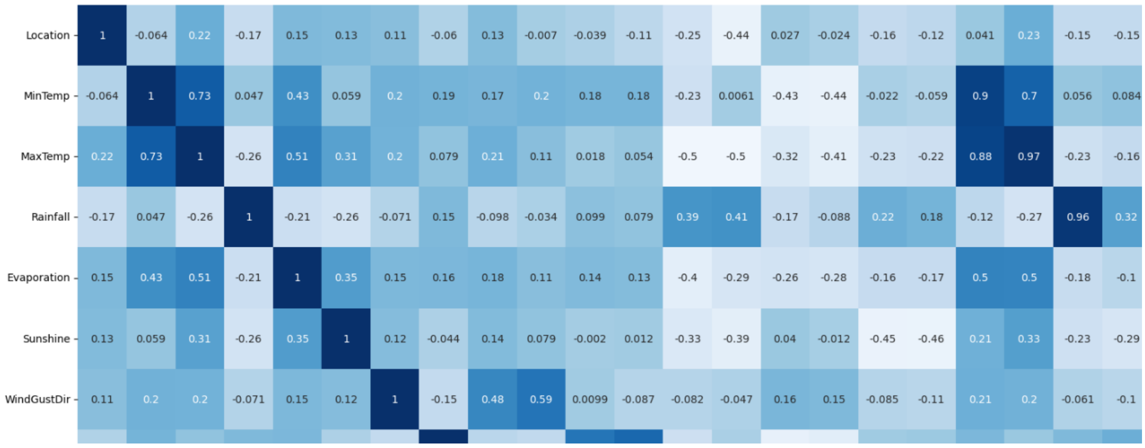
```
plt.figure(figsize=(12,6))
sns.barplot(x='Location',y='RainTomorrow',data=df)
plt.xticks(rotation=90)
plt.show()
```



c. Heatmap

```
[ ] plt.figure(figsize=(25,25))
    sns.heatmap(df.corr(),annot=True,cmap='Blues')
```

<Axes: >



6) To make model prediction and building better we assign numerical values to the location and the target variable

```
#have to encode location and date
df_new=df.groupby(['Location'])['RainTomorrow'].value_counts().sort_values(ascending=False).unstack()
df_new
```

```
location=df_new[1].sort_values(ascending=False).index
loc={}
for i in range(len(location)):
  loc[location[i]]=i
print(loc)
```

```
{'Portland': 0, 'Cairns': 1, 'Walpole': 2, 'Dartmoor': 3, 'MountGambier': 4, 'NorfolkIsland': 5, 'Albany': 6, 'Witchcliffe': 7, 'CoffsHarbour': 8, 'Sydney
```

```
[ ] df['Location']=df['Location'].map(loc)
```

```
[ ] df["Date"] = pd.to_datetime(df["Date"], format = "%Y-%m-%dT", errors = "coerce")
    df["Date_month"] = df["Date"].dt.month
    df["Date_day"] = df["Date"].dt.day
```

7) Outlier treatment

```
fig,axes=plt.subplots(5,3,figsize=(15,20))
fig.tight_layout(pad=3.0) #padding for better spacing
for i in range(len(continuous_val_attributes)):
    att=continuous_val_attributes[i]
    sns.boxplot(df[att],ax=axes[i//3,i%3])
    axes[i//3,i%3].set_title(att)
plt.show()
```

+ Code + Te

```
[ ] for i in continuous_val_attributes:
      q3=df[i].quantile(0.75)
      q1=df[i].quantile(0.25)
      iqr=q3-q1
      ll=q1-1.5*iqr
      ul=q3+1.5*iqr
      df.loc[df[i]>=ul,i]=ul
      df.loc[df[i]<=ll,i]=ll
```

```
[ ] fig,axes=plt.subplots(5,3,figsize=(15,20))
    fig.tight_layout(pad=3.0) #padding for better spacing
    for i in range(len(continuous_val_attributes)):
        att=continuous_val_attributes[i]
        sns.boxplot(df[att],ax=axes[i//3,i%3])
        axes[i//3,i%3].set_title(att)
    plt.show()
```

8) Splitting the data into train and test: after observing the heatmap, feature selection is done and then the data is split accordingly

```
[ ] dfp=df
    del dfp['Date_day']
    del dfp['Date_month']
```

```
[ ] dfp['']
```

```
[ ] x=dfp[["Location","MaxTemp","Sunshine","WindGustSpeed","Humidity9am","Pressure3pm","Cloud9am","Cloud3pm"]]
    y=dfp['RainToday']
    print(x.shape)
    print(y.shape)
```

```
[ ] x.head(20)
```

```
[ ] from sklearn.model_selection import train_test_split
```

```
[ ] X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

```
[ ] from imblearn.over_sampling import RandomOverSampler
```

```
[ ] #since there is imbalance of data we will use overSampler
    ros = RandomOverSampler(sampling_strategy='minority',random_state=22)
    X_train,y_train=ros.fit_resample(X_train, y_train)
```

Milestone 3: Model Building

For any problem statement, it is a good practice to try predicting using multiple models as results will differ for each model. Each model also focuses on different parts of the dataset to predict.

Steps for model building:

- Import libraries needed for all the models you want to build

```
[61] from sklearn.svm import SVC
     from xgboost import XGBClassifier
     from sklearn.linear_model import LogisticRegression
     from sklearn import metrics
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

- Make an instance of each model

```python
models = [LogisticRegression(), XGBClassifier()]
from sklearn.metrics import classification_report
for i in range(2):
  models[i].fit(X_train, y_train)
  print(models[i])
  train_preds = models[i].predict_proba(X_train)
  trainpred=models[i].predict(X_train)
  print('Training Accuracy : ', metrics.roc_auc_score(y_train, train_preds[:,1]))
  testpred=models[i].predict(X_test)
  val_preds = models[i].predict_proba(X_test)
  print('Validation Accuracy : ', metrics.roc_auc_score(y_test, val_preds[:,1]))
  report = classification_report(y_test,testpred)
  print(report)
  print()
```

```python
model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
history = model.fit(X_train, y_train, epochs=50, batch_size=64, validation_split=0.2, callbacks=[early_stopping])
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test accuracy: {accuracy}")
predictions = model.predict(X_test)
```

- Compare accuracies and classification report to check which model performed the best
- Save the chosen model as a pickle file and download it

```python
import pickle
pickle.dump(models[1],open('rainfall1.pkl','wb'))
```

Milestone 4: Application building

Steps:

- Create a project folder
- Open the folder in VSCode and create a virtual environment in it using the command “python -m venv nameofvirenv”
- Activate the virtual environment
- Pip install all the necessary libraries
- Create a templates folder, in that create a new file named “index.html”, this file be used for creation of the front-end UI. Add other routes to the interface while displaying if it will rain or not

templates > norain.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <title>No Rain</title>
</head>
<body>
    <h1>No Rain, Enjoy Your Day!</h1>
    <img src="{{ url_for('static', filename='norain.jpeg') }}" alt="No rain image">
</body>
</html>
```

templates > rain.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <title>Rainy Day</title>
</head>
<body>
    <h1>Chance of Rain, Carry an Umbrella</h1>
    <img src="{{ url_for('static', filename='rain.jpeg') }}" alt="Rainy Day Image">
</body>
</html>
```

```html
<!DOCTYPE html>
<html>
<head>
    <title>Rainfall Predictor</title>
    <style>
        body {
            background-color: rgb(34, 113, 240);
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 0;
            padding: 0;
        }
        h1 {
            background-color: rgba(0, 0, 0, 0.7);
            color: white;
            padding: 10px;
        }
        form {
            background-color: rgba(255, 255, 255, 0.8);
            padding: 20px;
            border-radius: 10px;
            margin: 20px auto;
            max-width: 400px;
        }
        label {
            display: block;
            margin-top: 10px;
        }
        input {
            width: 100%;
            padding: 10px;
            margin-top: 5px;
            margin-bottom: 15px;
            border: 1px solid #ccc;
            border-radius: 4px;
        }

        input[type="submit"] {
            background-color: #007BFF;
            color: white;
            padding: 10px 15px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }

        input[type="submit"]:hover {
            background-color: #0056b3;
        }
        p {
            background-color: rgba(255, 255, 255, 0.8);
            padding: 10px;
            border-radius: 10px;
            margin: 20px auto;
            max-width: 400px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>Rainfall Prediction</h1>
    <form method="POST">
        <label for="input1">Location</label>
        <select name="s" id="cityDropdown"></select>
        <label for="input2">Maximum Temperature(Celsius)</label>
        <input type="number" name="input2" step="any" required><br>
        <label for="input3">Sunshine</label>
        <input type="number" name="input3" step="any" required><br>
        <label for="input4">Wind Gust Speed</label>
        <input type="number" name="input4" step="any" required><br>
        <label for="input5">Humidity at 9am</label>
        <input type="number" name="input5" step="any" required><br>
        <label for="input6">Pressure at 3pm</label>
        <input type="number" name="input6" step="any" required><br>
        <label for="input7">Cloud at 9am</label>
        <input type="number" name="input7" step="any" required><br>
        <label for="input8">Cloud at 3pm</label>
        <input type="number" name="input8" step="any" required><br>
        <input type="submit" value="Predict">
    </form>
    <script>
        const cityMapping= {'Portland':0,'Cairns': 1,
        'Walpole': 2, 'Dartmoor': 3,
        'MountGambier': 4, 'NorfolkIsland': 5,
        'Albany': 6, 'Witchcliffe': 7,
        'CoffsHarbour': 8, 'Sydney': 9, 'Darwin': 10,
        'MountGinini': 11, 'NorahHead': 12, 'Ballarat': 13,
        'GoldCoast': 14, 'SydneyAirport': 15, 'Hobart': 16, 'Watsonia': 17,
        'Newcastle': 18, 'Wollongong': 19, 'Brisbane': 20,
        'Williamtown': 21, 'Launceston': 22, 'Adelaide': 23,
        'MelbourneAirport': 24, 'Perth': 25,
        'Sale': 26, 'Melbourne': 27, 'Canberra': 28, 'Albury': 29,
        'Penrith': 30, 'Nuriootpa': 31, 'BadgerysCreek': 32,
        'Tuggeranong': 33, 'PerthAirport': 34, 'Bendigo': 35,
        'Richmond': 36, 'WaggaWagga': 37, 'Townsville': 38,
        'PearceRAAF': 39, 'SalmonGums': 40, 'Moree': 41,
        'Cobar': 42, 'Mildura': 43, 'Katherine': 44,
        'AliceSprings': 45, 'Nhil': 46, 'Woomera': 47, 'Uluru': 48}
        const cityDropdown = document.getElementById('cityDropdown');
        for (const city in cityMapping) {
            const option = document.createElement('option');
            option.value = cityMapping[city];
            option.text = city;
            cityDropdown.appendChild(option);
        }
    </script>

    {% if prediction is not none %}
    <p>
        Prediction:
        {% if prediction == 1 %}
            It will rain. Carry an Umbrella!
        {% else %}
            No rain, Enjoy your Day!
        {% endif %}
    </p>
    {% endif %}
</body>
</html>
```

index.html

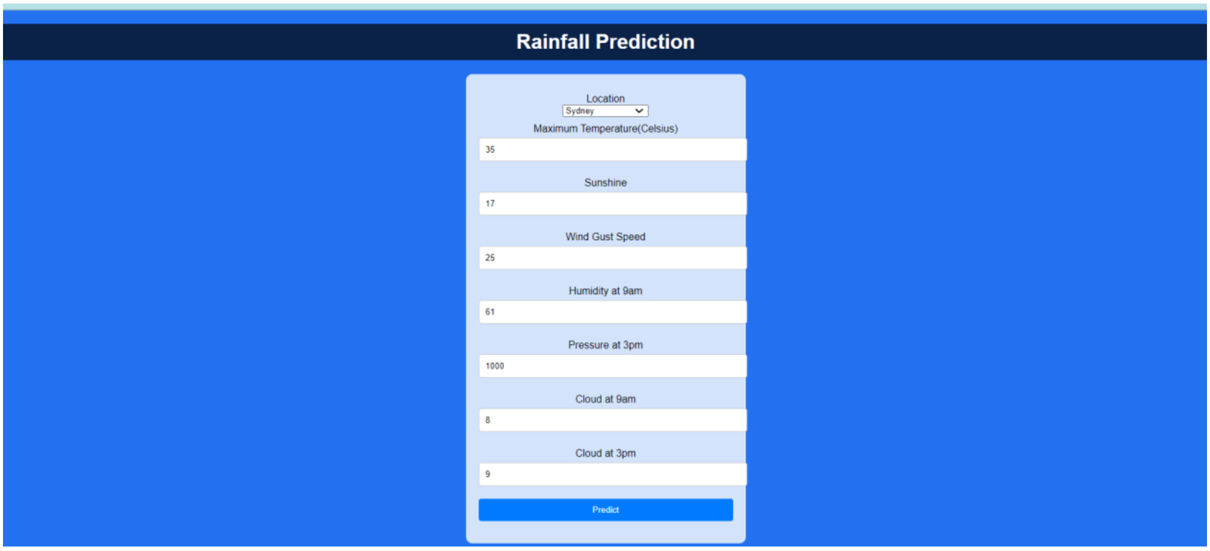
- Create a back-end file called app.py which will help connect the UI to the model, receive the data and send back the prediction

app.py > predict

```python
from flask import Flask, render_template,request,redirect,url_for
import pickle
import numpy as np
app = Flask(__name__)
with open("model.pkl", "rb") as model_file:
    model = pickle.load(model_file)
@app.route("/", methods=["GET", "POST"])
def predict():
    prediction = None
    if request.method == "POST":
        location = int(request.form["s"])  # Get the selected city as an integer
        input_values = [float(request.form[f"input{i}"]) for i in range(2, 9)]  # Updated field names
        input_values.insert(0, location)
        input_array = np.array(input_values).reshape(1, -1)
        prediction = model.predict(input_array)[0]
        if prediction==0:
            return redirect(url_for('norain'))
        else:
            return redirect(url_for('rainy'))
    return render_template("index.html", prediction=prediction)
@app.route("/norain")
def norain():
    return render_template('norain.html')
@app.route("/rainy")
def rainy():
    return render_template("rain.html")
if __name__ == "__main__":
    app.run(debug=True)
```

- After all the files have been made, run the command “python app.py” on PowerShell and view the app.

Milestone 5: Final Application view



Chance of Rain, Carry an Umbrella

No Rain, Enjoy Your Day!