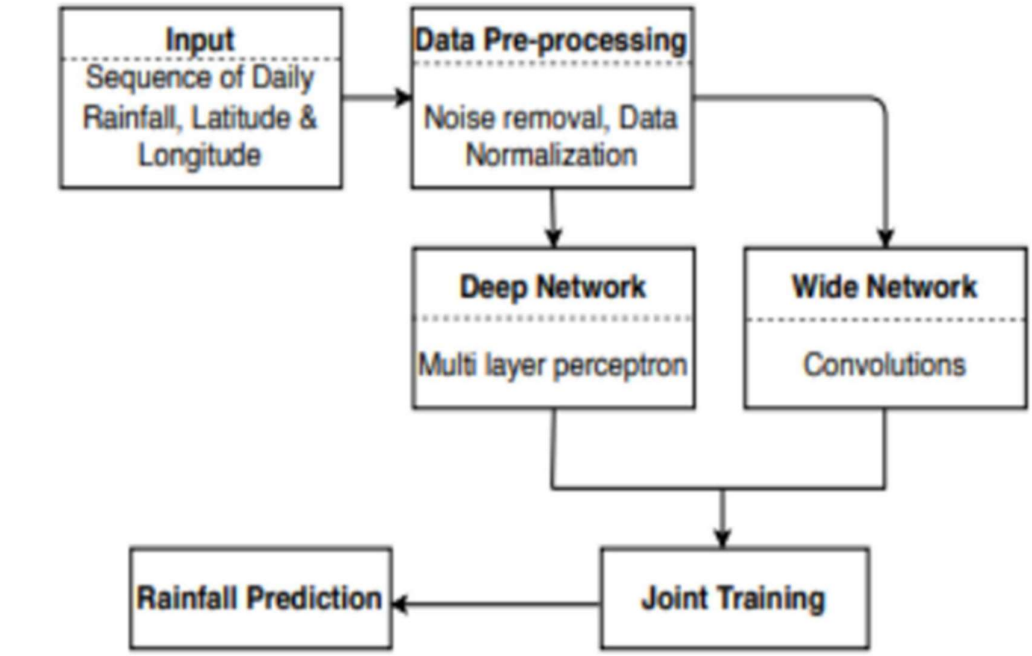
Rainfall Prediction

Project Description:

Particularly during the torrential rainfall event. Moreover, one of the major focuses of Climate change study is to understand whether there are extreme changes in the occurrence and frequency of heavy rainfall events. The accuracy level of the ML models used in predicting rainfall based on historical data has been one of the most critical concerns in hydrological studies. An accurate ML model could give early alerts of severe weather to help prevent natural disasters and destruction. Hence, there is needs to develop ML algorithms capable in predicting rainfall with acceptable level of precision and in reducing the error in the dataset of the projected rainfall from climate change model with the expected observable rainfall.

Technical Architecture:

Architecture Daigram :



Project Objectives:

By the end of this project:

- You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
- You will be able to know how to pre-process / clean the data using different data preprocessing techniques.
- You will able to analyse or get insights of data through visualization.
- Applying different algorithms according to dataset and based on visualization.
- You will able to know how to find accuracy of the model.
- You will be able to know how to build a web application using Flask framework.

Project Flow:

- User interacts with the UI (User Interface) to enter the input values
- Entered input values are analyzed by the model which is integrated
- Once model analyses the input the prediction is showcased on the UI To accomplish this, we have to complete all the activities and tasks listed below
- Data Collection.
    - Collect the dataset or Create the dataset
        - Data Preprocessing.
            - Import the Libraries.
            - Importing the dataset.
            - Checking for Null Values.
            - Data Visualization.
            - Taking care of Missing Data.
            - Feature Scaling.
            - Splitting Data into Train and Test.
- Model Building
- Import the model building Libraries
- Initializing the model
- Training and testing the model
- Evaluation of Model
- Save the Model
- Application Building o Create an HTML file
- Build a Python Code

Project Structure:

Create a Project folder which contains files as shown below

- A python file called app.py for server side scripting.
- We need the model which is saved and the saved model in this content is **Rainfall.pkl**
- Templates folder which contains index.HTML file, chance.HTML file, noChance.HTML file.
- Scale.pkl for scaling, encoder.pkl file for encoding the categorical data, imputer.pkl file for filling out the missing values

Milestone 1: Data Collection:

ML depends heavily on data, without data, it is impossible for an "AI" to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training **data set.** It is the actual **data set** used to train the model for performing various actions.

Activity1: Download The dataset

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository etc.
Please refer to the link given below to download the data set and to know about the dataset

https://docs.google.com/spreadsheets/d/1RA2OO0LZTeQykI_mvnensAjp6LM4YzWI1Tz0SUG5-Ao/edi t#gid=121883362

Milestone 2: Data Preprocessing

Data Pre-processing includes the following main tasks

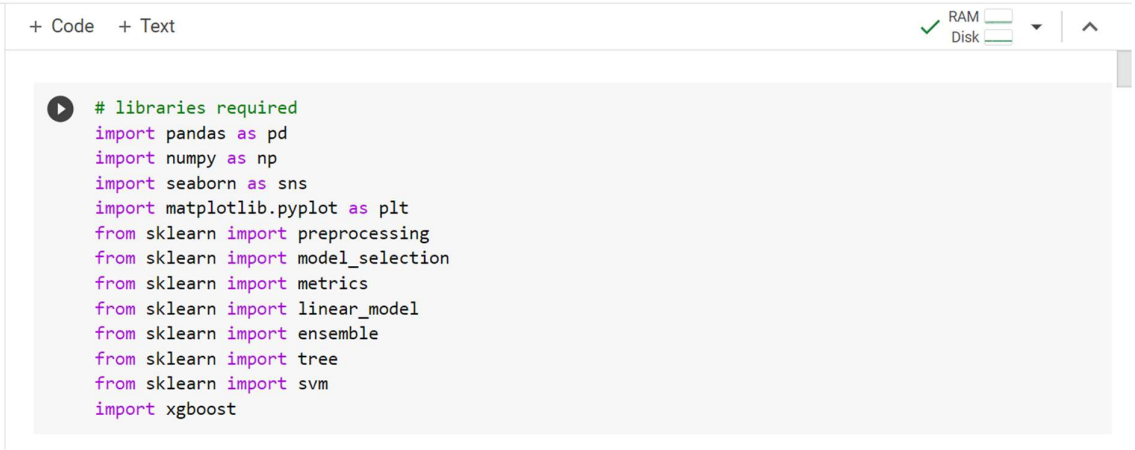
- Import the Libraries.
- Importing the dataset.
- Checking for Null
- Values.
- Data Visualization.
- Feature Scaling.
- Splitting Data into

Train and Test.

Activity 1: Import Necessary Libraries

- It is important to import all the necessary libraries such as pandas, numpy, matplotlib.
- **Numpy**- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.
- **Pandas**- It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **Seaborn**- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Matplotlib**- Visualisation with python. It is a comprehensive library for creating static,animated, and interactive visualizations in Python
- **Sklearn** – which contains all the modules required for model building

Importing libraries



Importing the Dataset

You might have your data in .csv files, .excel files

- Let's load a .csv data file into pandas using **read_csv() function.**We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- If your dataset is in some other location ,Then
- **Data=pd.read_csv(r"File_location/datasetname.csv")**

Note:r stands for "raw" and will cause backslashes in the string to be interpreted as actual backslashes rather than special characters.

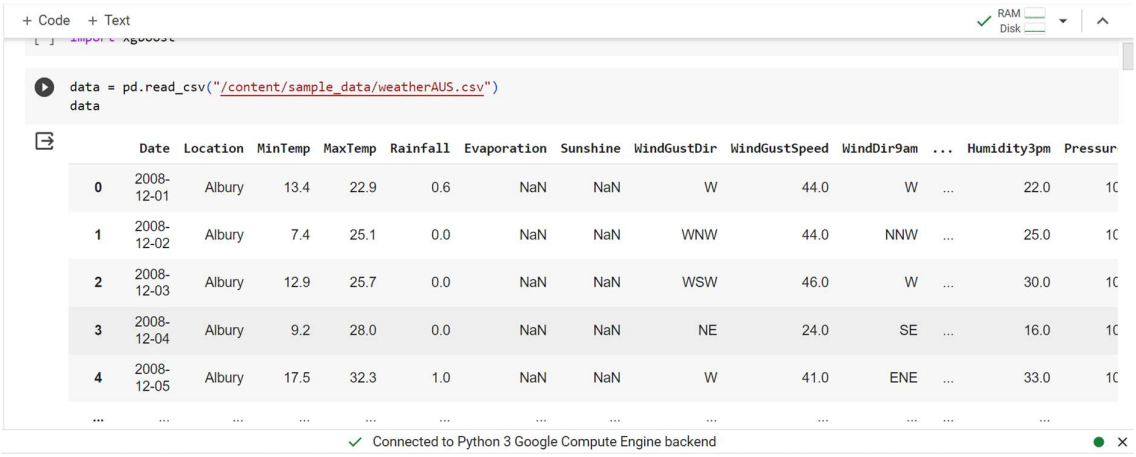
- If the dataset in same directory of your program, you can directly read it, without giving raw as r.
- Our Dataset weatherAus.csv contains following Columns
- Location, MinTemp, MaxTemp, Rainfall, WindGustSpeed,
- WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm
- Pressure9am, Pressure3pm, Temp9am, Temp3pm, RainToday,
- WindGustDir, WindDir9am, WindDir3pm,date
- Raintommorrow – output column

The output column to be predicted is **RainTommorow** .Based on the input variables we predict the chance of rain. The predicted output gives them a fair idea about it will rain or not.

Activity 3: Analyse the data

- head() method is used to return top n (5 by default) rows of a DataFrame or series.

- describe() method computes a summary of statistics like count, mean, standard deviation, min, max and quartile values.



```
+ Code + Text
data = pd.read_csv("/content/sample_data/weatherAUS.csv")
data
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressur
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	10
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	10
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	10
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	10
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Connected to Python 3 Google Compute Engine backend

```
data.head()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	1007.7
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	1010.6
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	1007.6
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	1017.6
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	1010.8

```
data.describe()
```

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Hu
count	141556.000000	141871.000000	140787.000000	81350.000000	74377.000000	132923.000000	140845.000000	139563.000000	140419.000000	138
mean	12.186400	23.226784	2.349974	5.469824	7.624853	39.984292	14.001988	18.637576	68.843810	
std	6.403283	7.117618	8.465173	4.188537	3.781525	13.588801	8.893337	8.803345	19.051293	
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	
25%	7.600000	17.900000	0.000000	2.600000	4.900000	31.000000	7.000000	13.000000	57.000000	
50%	12.000000	22.600000	0.000000	4.800000	8.500000	39.000000	13.000000	19.000000	70.000000	
75%	16.800000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	

From the data we infer that there are only decimal values and no categorical values

- info() gives information about the data

```
: data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 23 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Date           145460 non-null  object
 1   Location       145460 non-null  object
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object
 10  WindDir3pm     141232 non-null  object
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
 20  Temp3pm        141851 non-null  float64
```

Activity 4: Handling Missing Values

1. After loading it is important to check the complete information of data as it can indication many of the hidden information such as null values in a column or a row

2.Check whether any null values are there or not. if it is present then following can be done,

Rainfall prediction --1

File Edit View Insert Runtime Tools Help Last saved at 5:32 PM

+ Code + Text

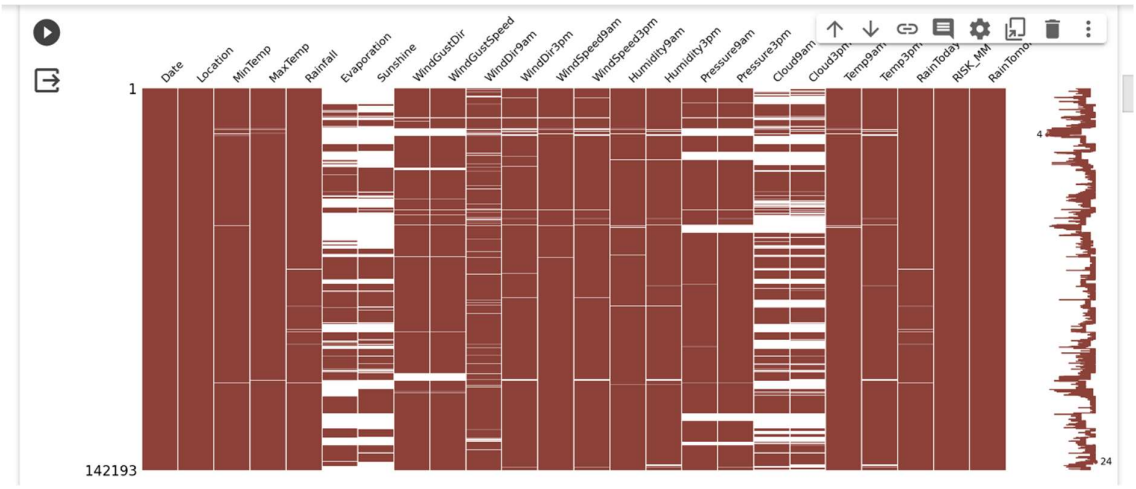
```
data.isnull().sum()
```

```
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
Temp9am           904
```

3. Missing matrix: It is way of representing the data in 2-D form. It gives coloured visual summary of the data

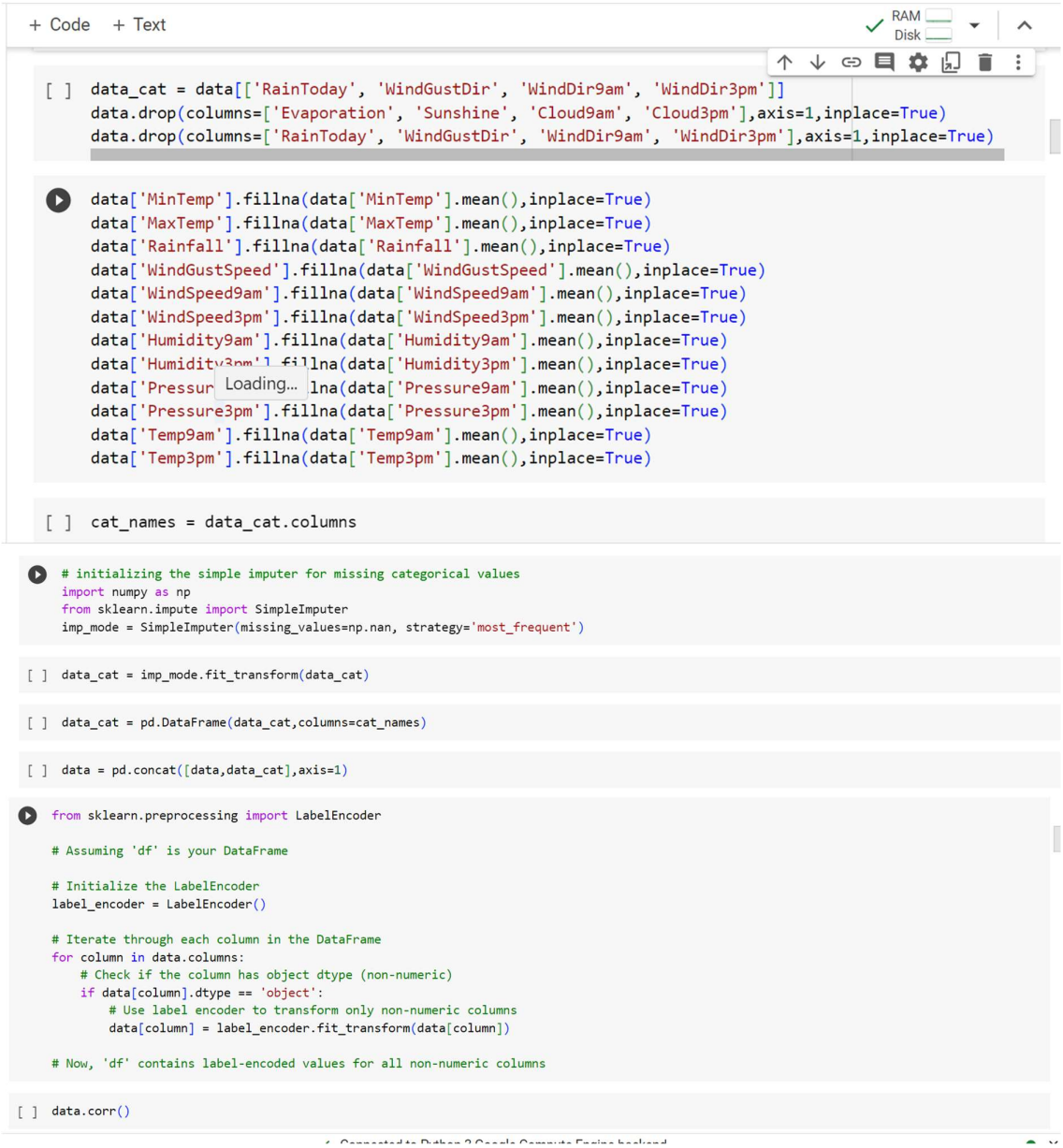
```
import missingno as msno
msno.matrix(data,color=(0.55, 0.255, 0.225), fontsize=16)
```

```
<Axes: >
```



4.Imputing data using Imputation method in sklearn.Simpleimputer

a.Filling NaN values with mean, median and mode using fillna() method.



```
[ ]  data_cat = data[['RainToday', 'WindGustDir', 'WindDir9am', 'WindDir3pm']]
     data.drop(columns=['Evaporation', 'Sunshine', 'Cloud9am', 'Cloud3pm'],axis=1,inplace=True)
     data.drop(columns=['RainToday', 'WindGustDir', 'WindDir9am', 'WindDir3pm'],axis=1,inplace=True)
```

```
data['MinTemp'].fillna(data['MinTemp'].mean(),inplace=True)
data['MaxTemp'].fillna(data['MaxTemp'].mean(),inplace=True)
data['Rainfall'].fillna(data['Rainfall'].mean(),inplace=True)
data['WindGustSpeed'].fillna(data['WindGustSpeed'].mean(),inplace=True)
data['WindSpeed9am'].fillna(data['WindSpeed9am'].mean(),inplace=True)
data['WindSpeed3pm'].fillna(data['WindSpeed3pm'].mean(),inplace=True)
data['Humidity9am'].fillna(data['Humidity9am'].mean(),inplace=True)
data['Humidity3pm'].fillna(data['Humidity3pm'].mean(),inplace=True)
data['Pressure9am'].fillna(data['Pressure9am'].mean(),inplace=True)
data['Pressure3pm'].fillna(data['Pressure3pm'].mean(),inplace=True)
data['Temp9am'].fillna(data['Temp9am'].mean(),inplace=True)
data['Temp3pm'].fillna(data['Temp3pm'].mean(),inplace=True)
```

```
[ ]  cat_names = data_cat.columns
```

```
# initializing the simple imputer for missing categorical values
import numpy as np
from sklearn.impute import SimpleImputer
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

```
[ ]  data_cat = imp_mode.fit_transform(data_cat)
```

```
[ ]  data_cat = pd.DataFrame(data_cat,columns=cat_names)
```

```
[ ]  data = pd.concat([data,data_cat],axis=1)
```

```
from sklearn.preprocessing import LabelEncoder

# Assuming 'df' is your DataFrame

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Iterate through each column in the DataFrame
for column in data.columns:
    # Check if the column has object dtype (non-numeric)
    if data[column].dtype == 'object':
        # Use label encoder to transform only non-numeric columns
        data[column] = label_encoder.fit_transform(data[column])

# Now, 'df' contains label-encoded values for all non-numeric columns
```

```
[ ]  data.corr()
```

From the heatmap, we see that there are missing values in the dataset

Activity 5: Data Visualisation

- Data visualization is where a given data set is presented in a graphical format. It helps the detection of patterns, trends and correlations that might go undetected in text-based data.
- Understanding your data and the relationship present within it is just as important as any algorithm used to train your machine learning model. In

fact, even the most sophisticated machine learning models will perform poorly on data that wasn't visualized and understood properly.

- To visualize the dataset we need libraries called Matplotlib and Seaborn.
- The Matplotlib library is a Python 2D plotting library which allows you to generate plots, scatter plots, histograms, bar charts etc.

Let's visualize our data using Matplotlib and searborn library.

Before diving into the code, let's look at some of the basic properties we will be using when plotting.

xlabel: Set the label for the x-

axis. **ylabel:** Set the label for the

y-axis.

title: Set a title for the axes.

Legend: Place a legend on the axes.

1. data.corr() gives the correlation between the columns

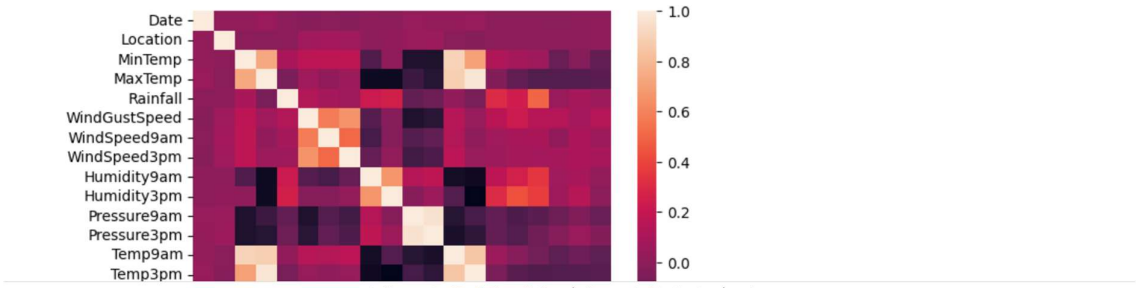
```
data.corr()
```

	Date	Location	MinTemp	MaxTemp	Rainfall	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pres
Date	1.000000	0.021490	0.016930	0.040321	-0.012453	-0.025140	-0.015892	-0.025815	0.000940	-0.010464	0
Location	0.021490	1.000000	-0.006194	-0.020490	-0.003457	0.069269	0.077038	0.064180	-0.002065	0.011066	0
MinTemp	0.016930	-0.006194	1.000000	0.733919	0.103314	0.173331	0.174946	0.174187	-0.232372	0.005913	-0
MaxTemp	0.040321	-0.020490	0.733919	1.000000	-0.074202	0.066329	0.014586	0.050381	-0.499777	-0.499725	-0
Rainfall	-0.012453	-0.003457	0.103314	-0.074202	1.000000	0.127250	0.085977	0.056762	0.221392	0.249609	-0
WindGustSpeed	-0.025140	0.069269	0.173331	0.066329	0.127250	1.000000	0.577864	0.658377	-0.209272	-0.025822	-0
WindSpeed9am	-0.015892	0.077038	0.174946	0.014586	0.085977	0.577864	1.000000	0.513079	-0.269027	-0.030991	-0
WindSpeed3pm	-0.025815	0.064180	0.174187	0.050381	0.056762	0.658377	0.513079	1.000000	-0.144270	0.015789	-0
Humidity9am	0.000940	-0.002065	-0.232372	-0.499777	0.221392	-0.209272	-0.269027	-0.144270	1.000000	0.659852	0
Humidity3pm	-0.010464	0.011066	0.005913	-0.499725	0.249609	-0.025822	-0.030991	0.015789	0.659852	1.000000	-0

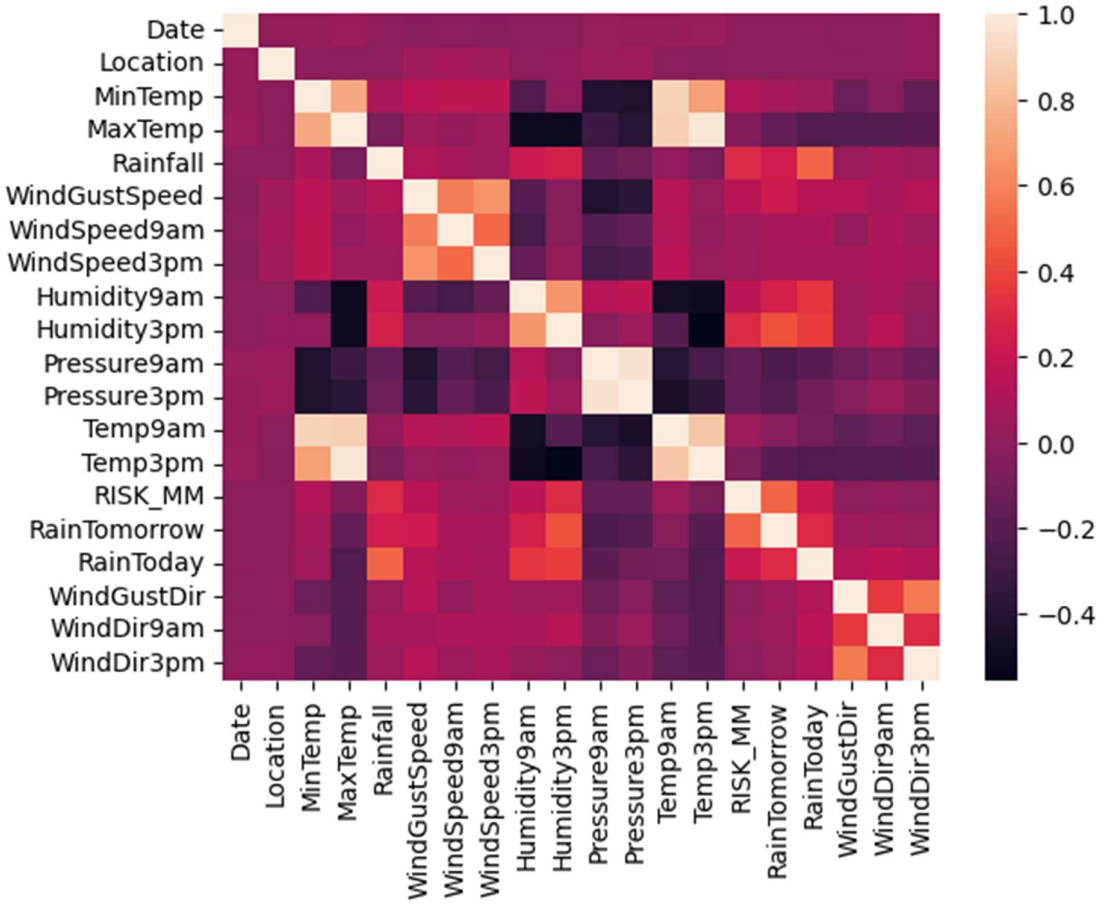
```
[ ] cor = data.corr()
```

```
sns.heatmap(data=cor,xticklabels=cor.columns.values,yticklabels=cor.columns.values)
```

```
<Axes: >
```



Connected to Python 3 Google Compute Engine backend

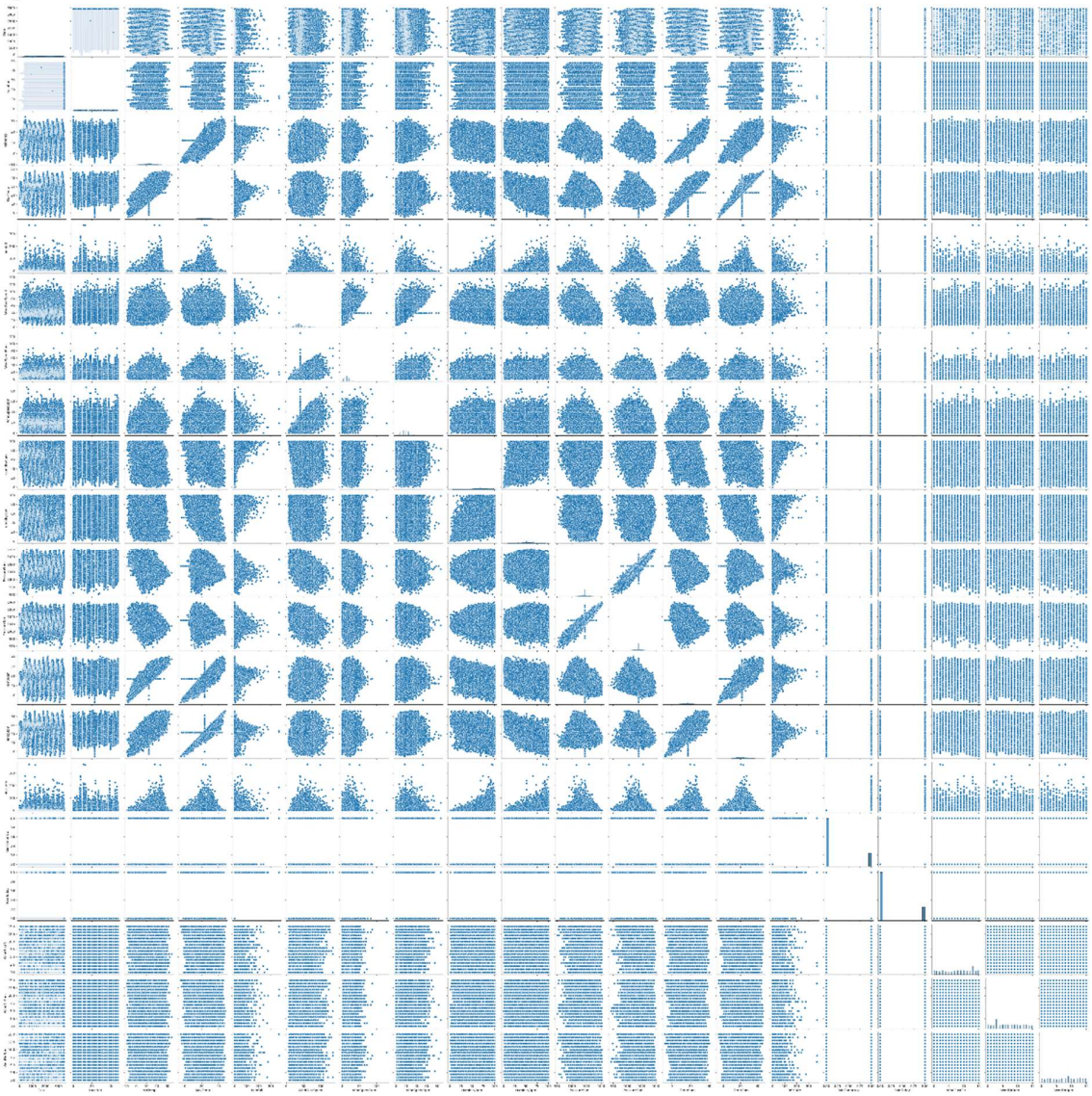


- Correlation strength varies based on colour, lighter the colour between two variables, more the strength between the variables, darker the colour displays the weaker correlation

- We can see the correlation scale values on left side of the above image

- 2.Pair Plot: Plot pairwise relationships in a dataset.
- By default, this function will create a grid of Axes such that each numeric variable in data will by shared across the y-axes across a single row and the x-axes across a single column.
- The diagonal plots are treated differently: a univariate distribution plot is drawn to show the marginal distribution of the data in each column.We implement this using the below code

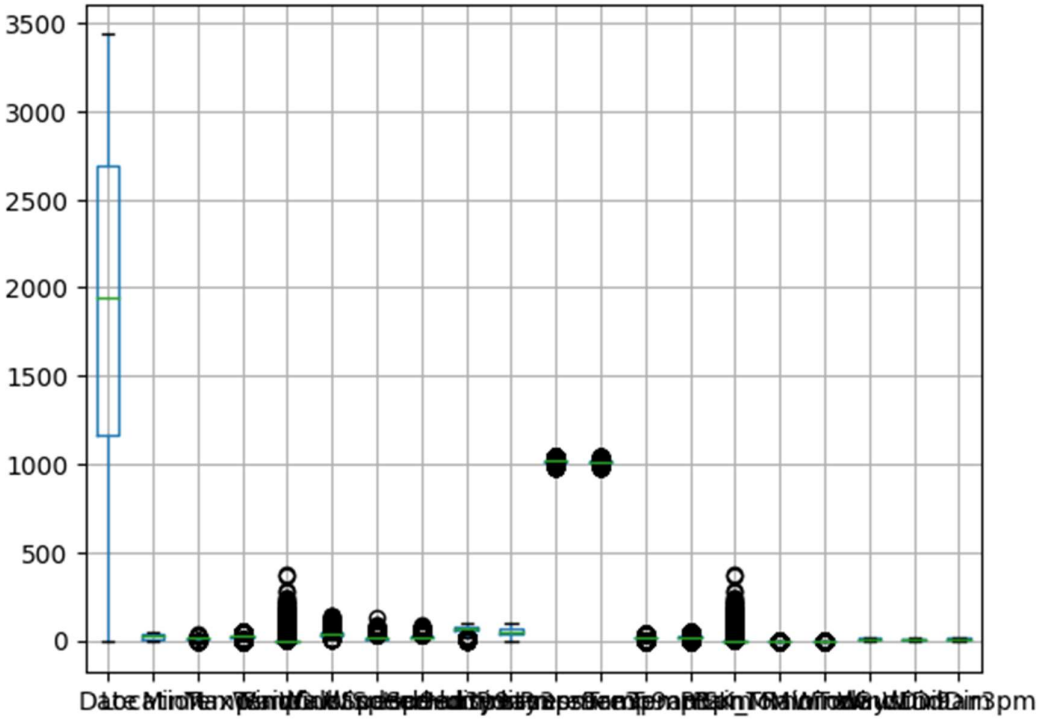
```
sns.pairplot(data)
```



- The output is as shown below

- Pair plot usually gives pair wise relationships of the columns in the dataset
- From the above pairplot we infer that
- from the above plot we can draw inferences such as linearity and strength between the variables
- how features are correlated(positive, neutral and negative)
- Box Plot: jupyter has a built-in function to create boxplot called boxplot(). A boxplot plot is a type of plot that shows the spread of data in all the quartiles

```
data.boxplot()
```



From the above box plot we infer how the datapoints are spread and the existence of the outliers

Activity 6: Splitting the Dataset into Dependent and Independent variable

- In machine learning, the concept of dependent variable (y) and independent variables(x) is important to understand. Here, Dependent variable is nothing but output in dataset and independent variable is all inputs in the dataset.

- With this in mind, we need to split our dataset into the matrix of independent variables and the vector or dependent variable. Mathematically, Vector is defined as a matrix that has just one column.

  To read the columns, we will use **iloc** of pandas (used to fix the indexes for selection) which takes two parameters — [row selection, column selection].

  Let's split our dataset into independent and dependent variables.

  y = data['RainTomorrow'] -
  independent x =
  data.drop('RainTomorrow',axis=1
)

Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data.

```
[ ] from sklearn.preprocessing import StandardScaler
```

```
#splitting x and y values
y = data['RainTomorrow']
x = data.drop('RainTomorrow',axis=1)
```

```
[ ] x = data.drop('Date',axis=1)
```

```
[ ] names = x.columns
```

```
[ ] names
```

```
Index(['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
```

```
[ ] names
```

```
Index(['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm', 'RISK_MM',
       'RainTomorrow', 'RainToday', 'WindGustDir', 'WindDir9am', 'WindDir3pm'],
      dtype='object')
```

```
[ ] sc = StandardScaler()
```

- After scaling the data will be converted into array form

- Loading the feature names before scaling and converting them back to dataframe after standard scaling is applied

Activity 8: Splitting the data into Train and Test

- When you are working on a model and you want to train it, you obviously have a dataset. But after training, we have to test the model on some test dataset. For this, you will a dataset which is different from the training set you used earlier. But it might not always be possible to have so much data during the development phase. In such cases, the solution is to split the dataset into two sets, one for training and the other for testing.
- But the question is, how do you split the data? You can't possibly manually split the dataset into two sets. And you also have to make sure you split the data in a random manner. To help us with this task, the Scikit library provides a tool, called the Model Selection library. There is a class in the library which is,**'train_test_split.'** Using this we can easily split the dataset into the training and the testing datasets in various proportions.
- The train-test split is a technique for evaluating the performance of a machine learning algorithm.
- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.
- In general you can allocate 80% of the dataset to training set and the remaining 20% to test set.We will create 4 sets— X_train (training part of the matrix of features), X_test (test part of the matrix of features), Y_train (training part of the dependent variables associated with the X train sets, and therefore also the same indices), Y_test (test part of the dependent variables associated with the X test sets, and therefore also the same indices.
- There are a few other parameters that we need to understand before we use the class:
- **test_size** — this parameter decides the size of the data that has to be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset
- **train_size** — you have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.

- **random_state** — here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the Random_state class, which will become the number generator. If you don't pass anything, the Random_state instance used by np.random will be used instead.
- Now split our dataset into train set and test using train_test_split class from scikit learn library.

  from sklearn import model_selection
  x_train,x_test,y_train,y_test=model_selection.train_test_split(x,y,test_size=0.2,ran dom_state =0)

Milestone 3: Model Buiding:

Model building includes the following main tasks

- Import the model building Libraries
- Initializing the model
  Training and testing the model
- Evaluation of Model
- Save the Model

Activity 1: Training and Testing the Model

- Once after splitting the data into train and test, the data should be fed to an algorithm to build a model.
- There are several Machine learning algorithms to be used depending on the data you are going to process such as images, sound, text, and numerical values. The algorithms that you can choose according to the objective that you might have it may be Classification algorithms are Regression algorithms. 1.Logistic Regression

  2.Decision Tree Classifier

  3.Random Forest Classifier

  4.KNN

  5.svm

5.xgboost

Steps in Building the model:-

- Initialize the model
- Fit the models with x_train and y_train
- Predict the y_train values and calculate the accuracy
- Predict the y_test values and calculate the accuracy

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Assuming 'x' is your DataFrame
numeric_columns = data.select_dtypes(include=['number']).columns
x_numeric = data[numeric_columns]

sc = StandardScaler()
x = sc.fit_transform(x_numeric)
```

```
[ ] x = sc.fit_transform(x)
```

```
[ ] x = sc.fit_transform(x)
```

```
# Assuming 'x' is your DataFrame and 'names' is a list of column names
x = pd.DataFrame(data, columns=names)
```

+ Code + Text

```
[ ] from sklearn import model_selection
```

```
[ ] x_train,x_test,y_train,y_test=model_selection.train_test_split(x,y,test_size=0.2,random_state =0)
```

```
[ ] x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

```
((113754, 19), (113754,), (28439, 19), (28439,))
```

```
import xgboost
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression

XGBoost = xgboost.XGBRFClassifier()
Rand_forest = RandomForestClassifier()
svm = SVC()
Dtree = DecisionTreeClassifier()
GBM = GradientBoostingClassifier()
log = LogisticRegression()
```

We are using the algorithm from Scikit learn library to build the model as shown below,

Once the model is trained, it's ready to make predictions. We can use the **predict** method on the model and pass **x_test** as a parameter to get the output as **y_pred.**

Notice that the prediction output is an array of real numbers corresponding to the input array.

Activity 2: Model Evaluation

After training the model, the model should be tested by using the test data which is been separated while splitting the data for checking the functionality of the model.

Regression Evaluation Metrics:

These model evaluation techniques are used to find out the accuracy of models built in classification type of machine learning models. We have three types of evaluation methods.

- Accuracy_score
- Confusion matrix
- Roc- Auc Curve

1. Accuracy_score

It is the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

[40]:
```
print("xgboost:",metrics.accuracy_score(y_test,t1))
print("Rand_forest:",metrics.accuracy_score(y_test,t2))
print("svm:",metrics.accuracy_score(y_test,t3))
print("Dtree:",metrics.accuracy_score(y_test,t4))
print("GBM:",metrics.accuracy_score(y_test,t5))
print("log:",metrics.accuracy_score(y_test,t6))
```

```
xgboost: 0.8420478919793242
Rand_forest: 0.8569218326945391
svm: 0.8525967861035901
Dtree: 0.7837476704525476
GBM: 0.8499947255529379
log: 0.8418369140968388
```

[ ]
```
# Check unique values in 'RainTomorrow'
print(data['RainTomorrow'].unique())
```

```
[0 1]
```

[ ]
```
Rand_forest.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
[ ] Rand_forest.fit(x_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
svm.fit(x_train, y_train)
```

```
▾ SVC
SVC()
```

```
[ ] Dtree.fit(x_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
GBM.fit(x_train, y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
[ ] log.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
log.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1)
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
[ ] p1 = XGBoost.predict(x_train)
    p2 = Rand_forest.predict(x_train)
    p3 = svm.predict(x_train)
    p4 = Dtree.predict(x_train)
    p5 = GBM.predict(x_train)
    p6 = log.predict(x_train)
```

```
print("xgboost:",metrics.accuracy_score(y_train,p1))
print("Rand_forest:",metrics.accuracy_score(y_train,p2))
print("svm:",metrics.accuracy_score(y_train,p3))
print("Dtree:",metrics.accuracy_score(y_train,p4))
print("GBM:",metrics.accuracy_score(y_train,p5))
print("log:",metrics.accuracy_score(y_train,p6))
```

```
xgboost: 1.0
Rand_forest: 1.0
svm: 0.8700001758179932
Dtree: 1.0
GBM: 1.0
log: 0.9997011094115372
```

Select the model,which gives the best accuracy of all,and generate predictions and find the accuracy with training and testing data

2. Confusion Matrix

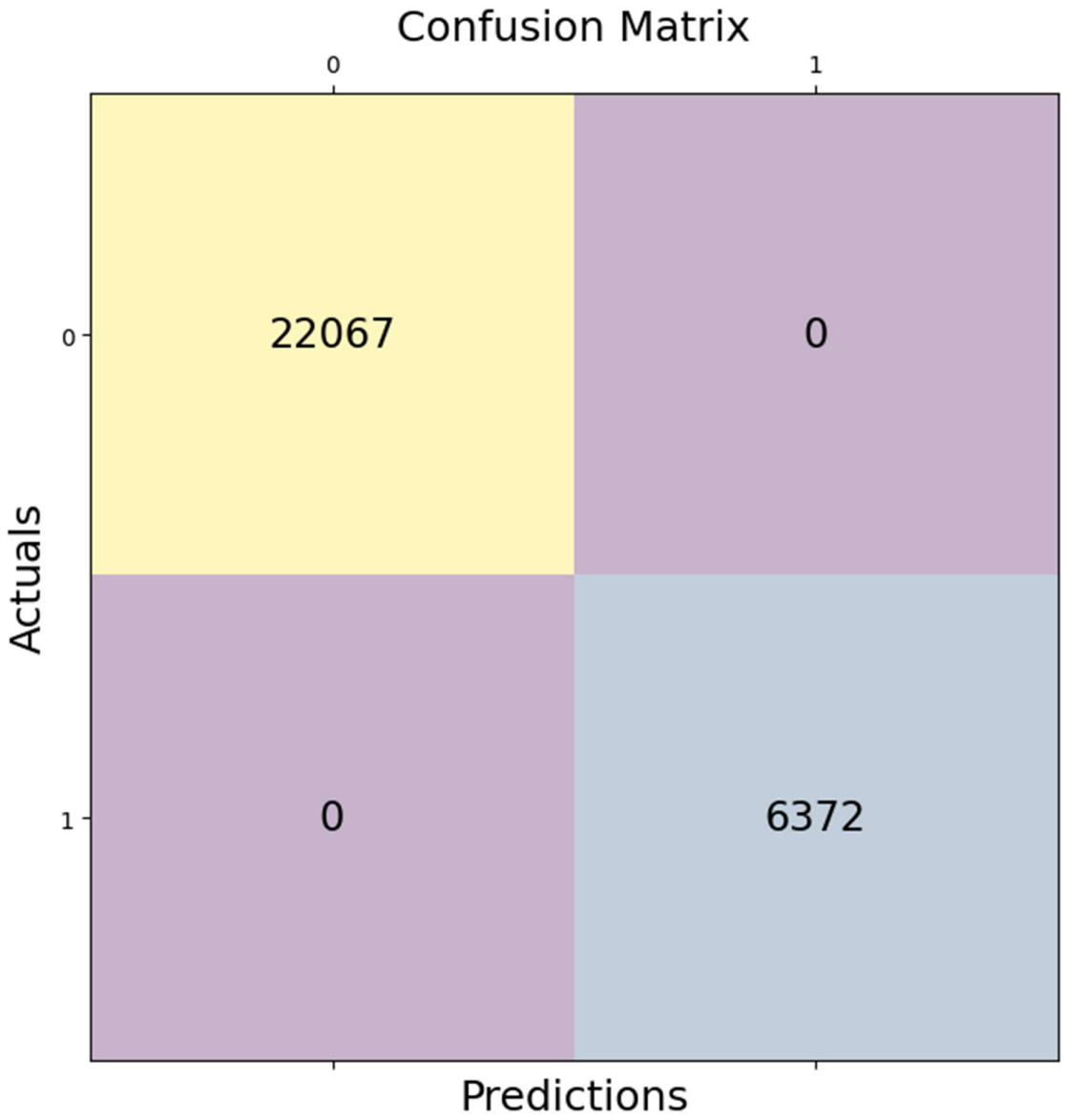
It is a matrix representation of the results of any binary testing

Fig: Confusion Matrix of prediction of rainfall

```
[ ] conf_matrix1 = metrics.confusion_matrix(y_test,t1)
```

```
fig, ax = plt.subplots(figsize=(7.5, 7.5))
ax.matshow(conf_matrix1, alpha=0.3)
for i in range(conf_matrix1.shape[0]):
    for j in range(conf_matrix1.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix1[i, j], va='center', ha='center', size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('Confusion Matrix', fontsize=18)
plt.show()
```



1. True Positive: 3242 (You have predicted the positive case correctly!)
2. True Negative: 21014 (You have predicted negative case correctly!)
3. False Positive: 1053 (You have predicted it will rain, but in actual it will not rain)
4. False Negative: 3130 (Wrong predictions)

3. Roc-Auc Curve

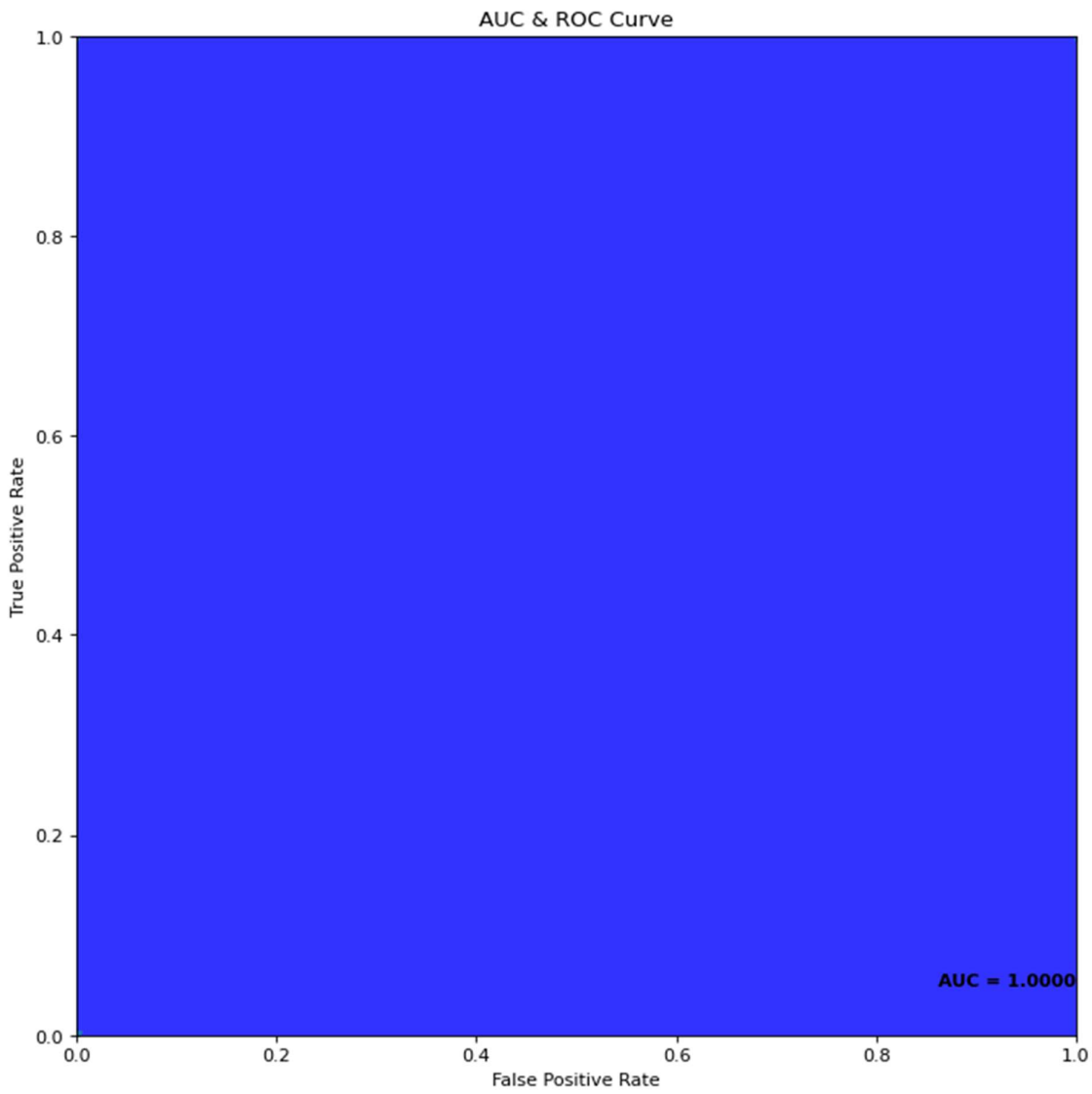
- AUC is the area under the ROC curve. AUC ROC indicates how well the probabilities from the positive classes are separated from the negative classes.

- AUC - ROC curve is a performance measurement for classification problem at various thresholds settings
- ROC is a probability curve and AUC represents degree or measure of separability.
- Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1sThe ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis. ● Code for Roc and performance metrics

```
auc = metrics.roc_auc_score(y_test, t1)

fpr, tpr, thresholds = metrics.roc_curve(y_test, t1)

plt.figure(figsize=(12, 10), dpi=80)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve")
plt.plot(fpr, tpr, 'v')
plt.fill_between(fpr, tpr, facecolor='blue', alpha=0.8)
plt.text(1, 0.05, f'AUC = {auc:.4f}', ha='right', fontsize=10, weight='bold', color='black')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



For testing the model we use the below method,

Activity 3: Save the Model

After building the model we have to save the model.

Pickle in **Python** is primarily **used** in serializing and deserializing a **Python** object structure. In other words, it's the process of converting a **Python** object into a byte stream to store it in a file/database, maintain program state across sessions, or transport data over the network. wb indicates write method and rd indicates read method.

This is done by the below code

Saving the model

```
import pickle
```

```
[ ] pickle.dump(XGBoost,open('rainfall.pkl','wb'))
    pickle.dump(label_encoder,open('encoder.pkl','wb'))
    pickle.dump(imp_mode,open('impter.pkl','wb'))
    pickle.dump(sc,open('scale.pkl','wb'))
```

Milestone 4 : Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script

Activity 1: Build HTML Code

In this HTML page, we will create the front end part of the web page. In this page we will accept input from the user and Predict the values. For more information regarding HTML **https://www.w3schools.com/html/**

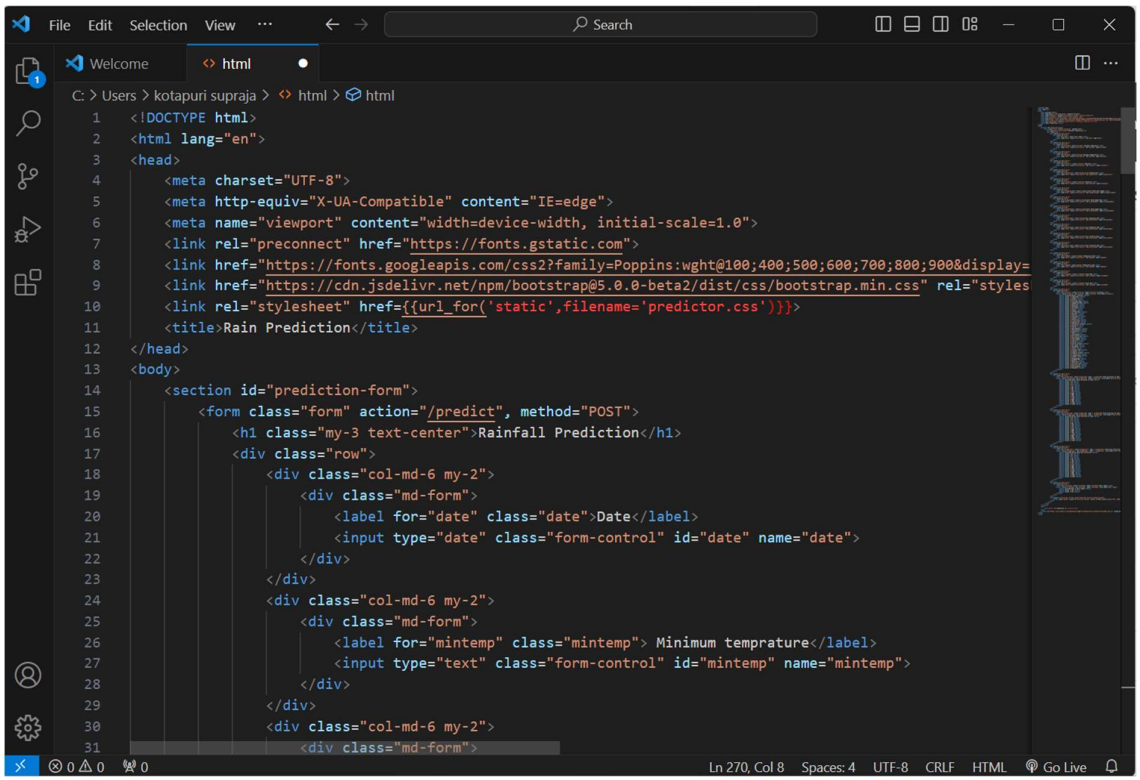
In our project we have 3 HTML files ,they are

1.home.html

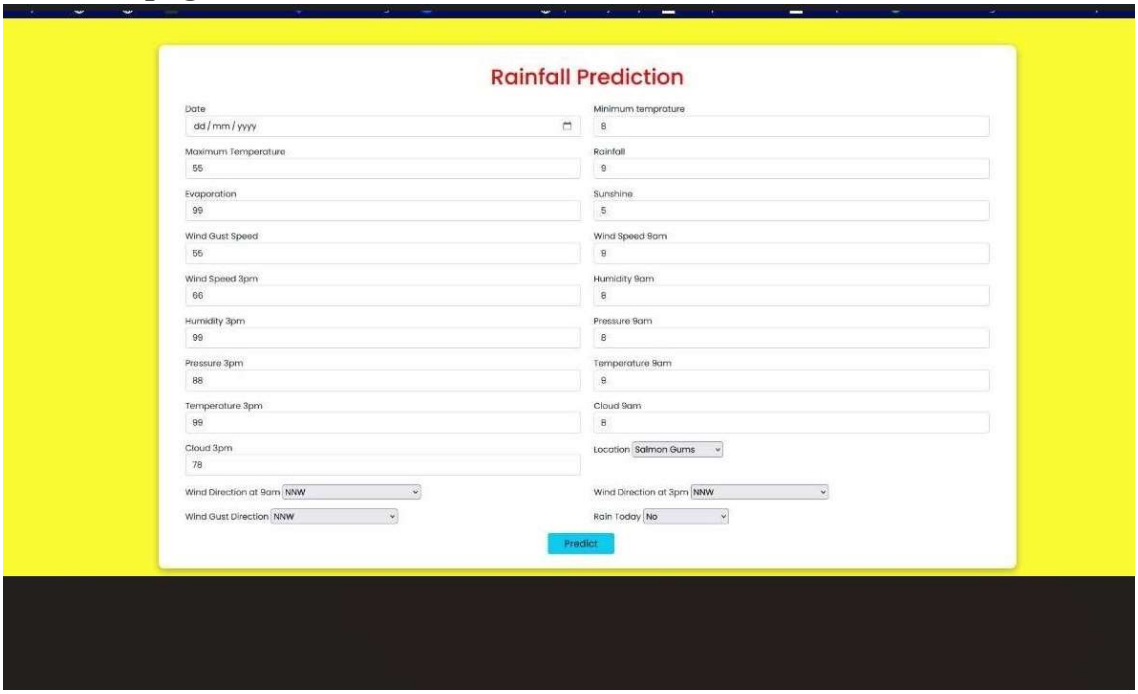
2.rain.html

3.sunny.html

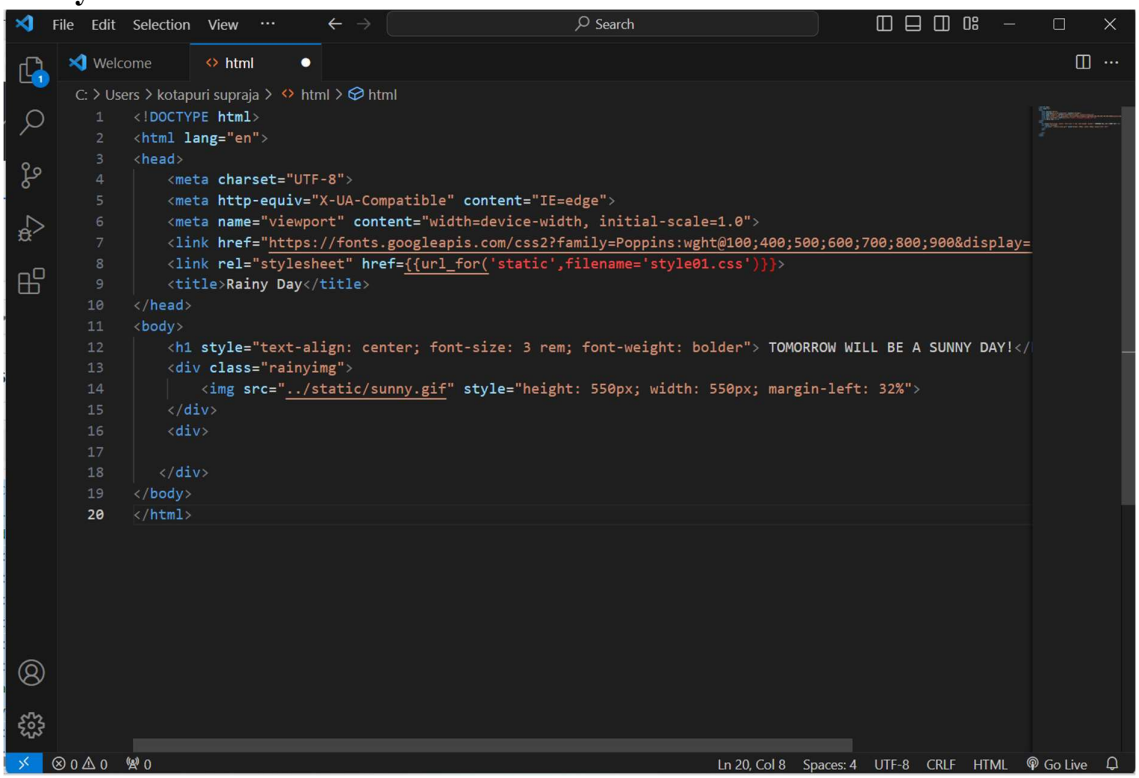
index.html



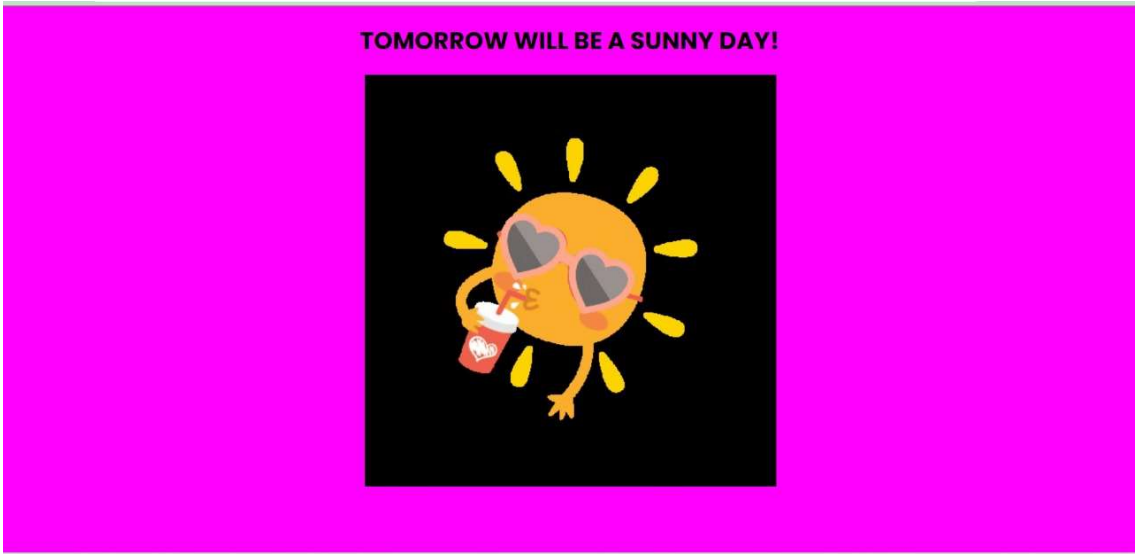
The html page looks like



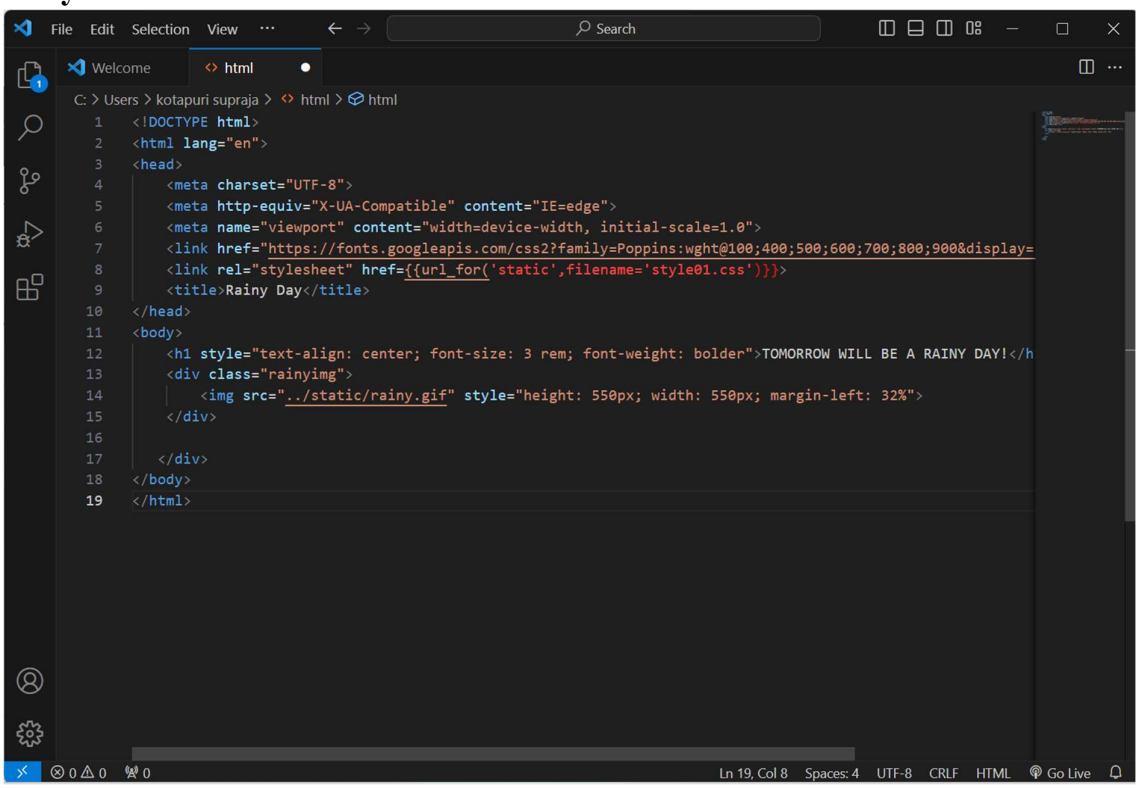
sunny.html



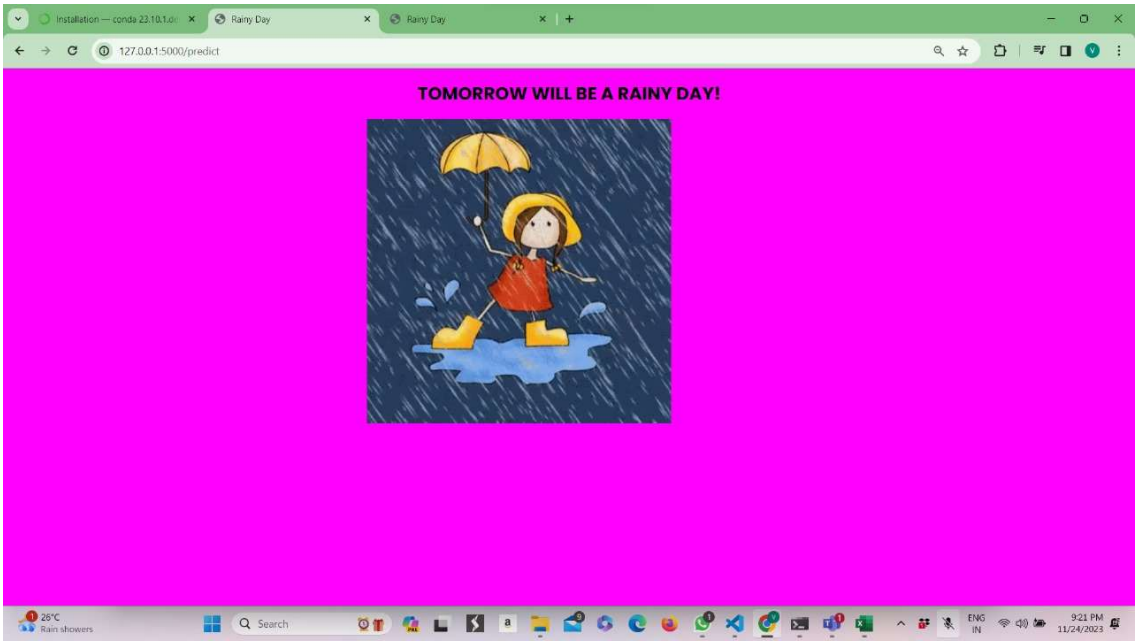
The html page looks like



rainy.html



The html page looks like



Activity 2: Main Python Script Let us build app.py flask file which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.

In order to develop web api with respect to our model, we basically use Flask framework which is written in python.

Line 1-3 We are importing necessary libraries like Flask to host our model request

Line 4 Initialise the Flask application

Line 5 Loading the model using pickle

Line 7 Routes the api url

Line 9 Rendering the template. This helps to redirect to home page. In this home page ,we give our input and ask the model to predict

Line 19 we are taking the inputs from the form

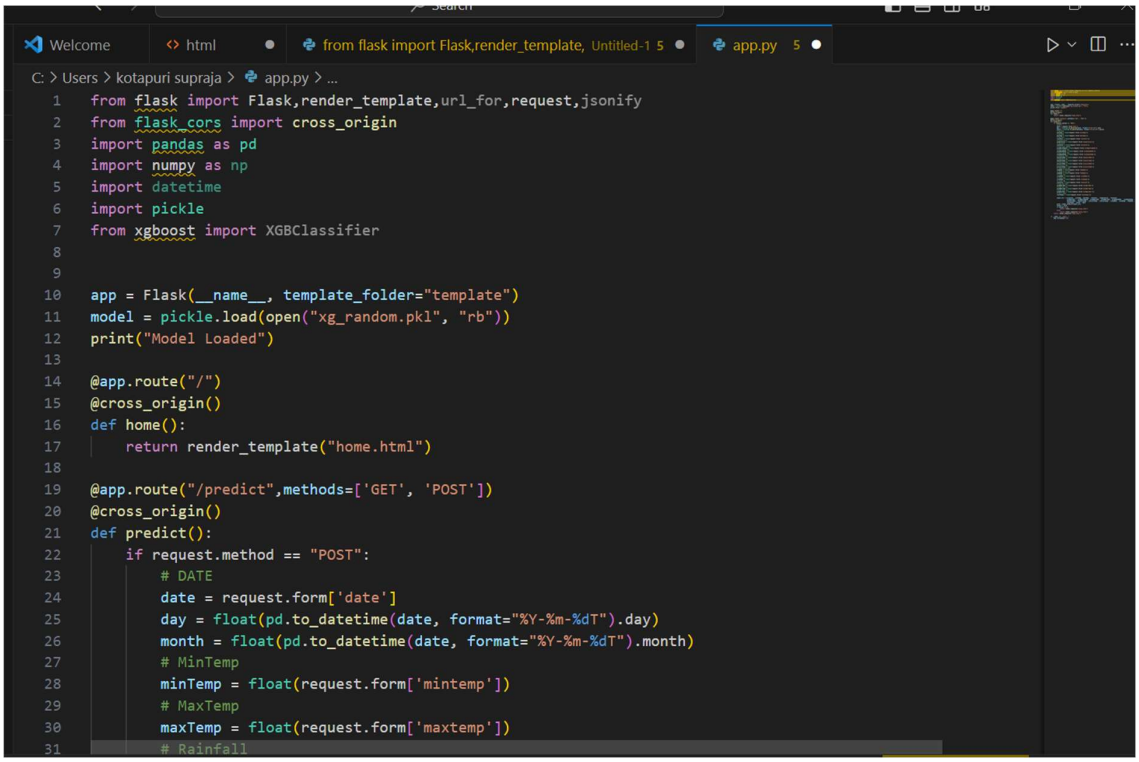
Line 21-23 Feature Scaling the inputs

Line 24 Predicting the values given by the user

Line 27-30 If output is false render noChance template

If output is True render chance template

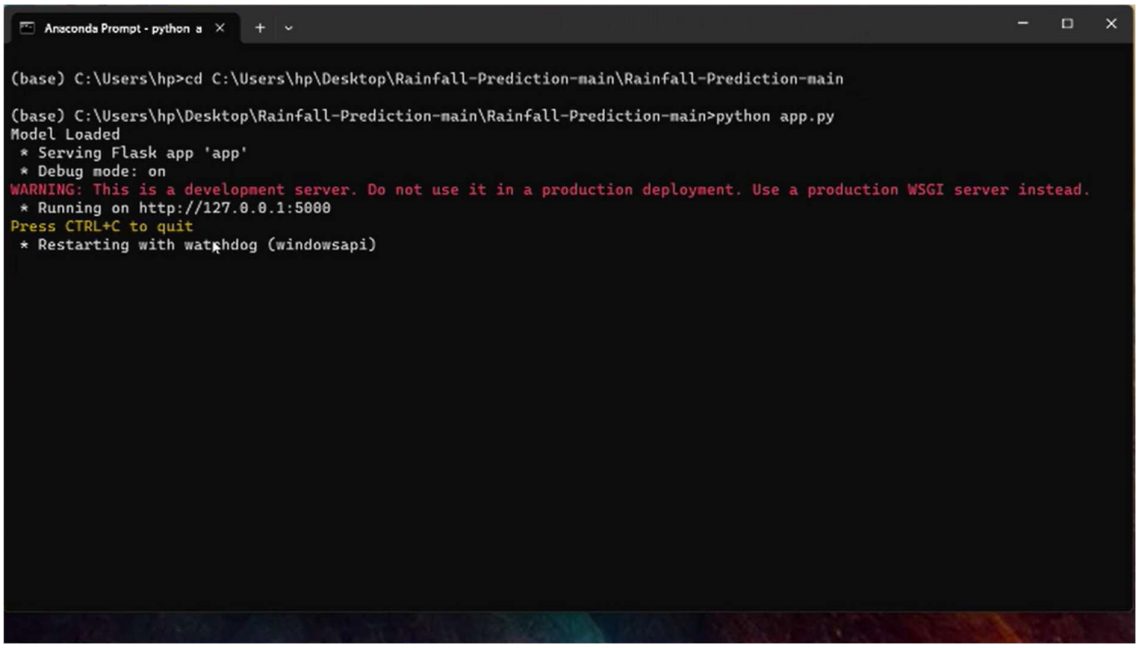
Line 31 The value of __name__ is set to __main__ when module run as main program other wise it is set to name of the module



Activity 3: Run the App

o Open anaconda prompt from the start menu o Navigate to the folder where your python script is. o Now type "python app.py" command

Navigate to the localhost where you can view your web page,Then it will run on local host:5000



Activity 4:

- Copy the http link and paste it in google link tab,it will display the form page
- Enter the values as per the form and click on predict buttion
- It will redirect to the page based on prediction output