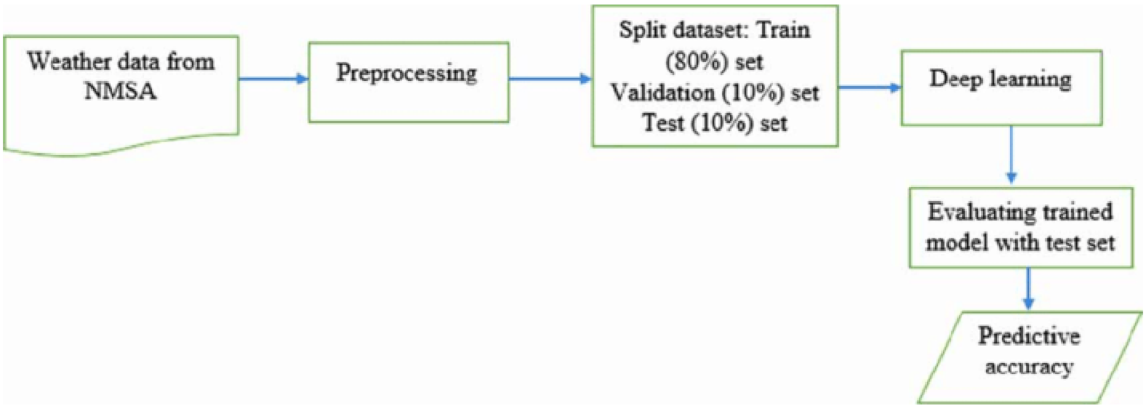
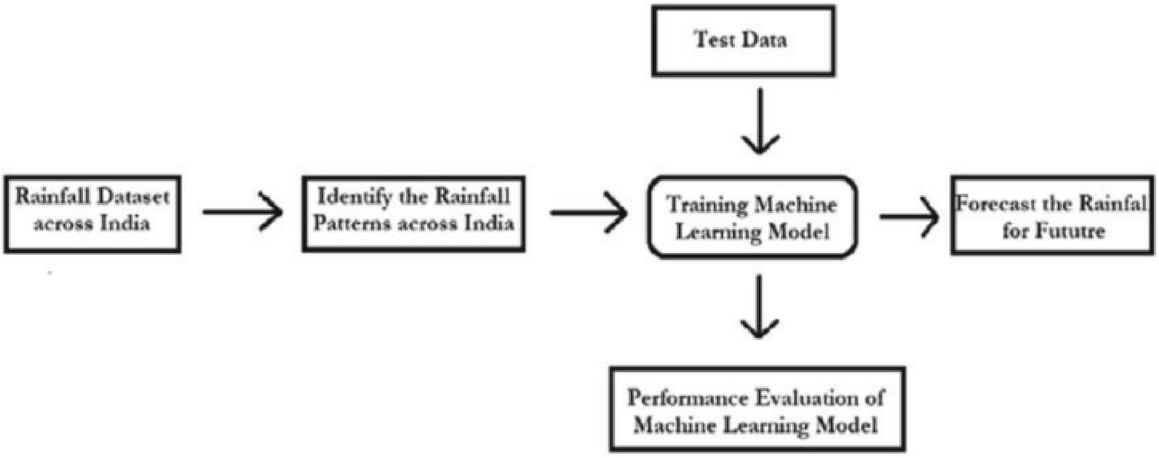
Rainfall Prediction

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: Project


Weather data from NMSA
Preprocessing
Split dataset: Train (80%) set Validation (10%) set Test (10%) set
Deep learning
Evaluating trained model with test set
Predictive accuracy


Test Data
Rainfall Dataset across India
Identify the Rainfall Patterns across India
Training Machine Learning Model
Forecast the Rainfal for Fututre
Performance Evaluation of Machine Learning Model

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. o Collect the dataset or Create the dataset ● Data Preprocessing.
  - o Import the Libraries. o Importing the dataset. o Checking for Null Values. o Data Visualization. o Taking care of Missing Data. o Feature Scaling.
  - o Splitting Data into Train and Test.
- Model Building o Import the model building Libraries o Initializing the model o Training and testing the model oEvaluation of Model o Save the Model
- Application Building o Create an HTML file o Build a Python Code

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://www.kaggle.com/datasets/rajanand/rainfall-in-india

Milestone 2: Data Preprocessing

Data Pre-processing includes the following main tasks

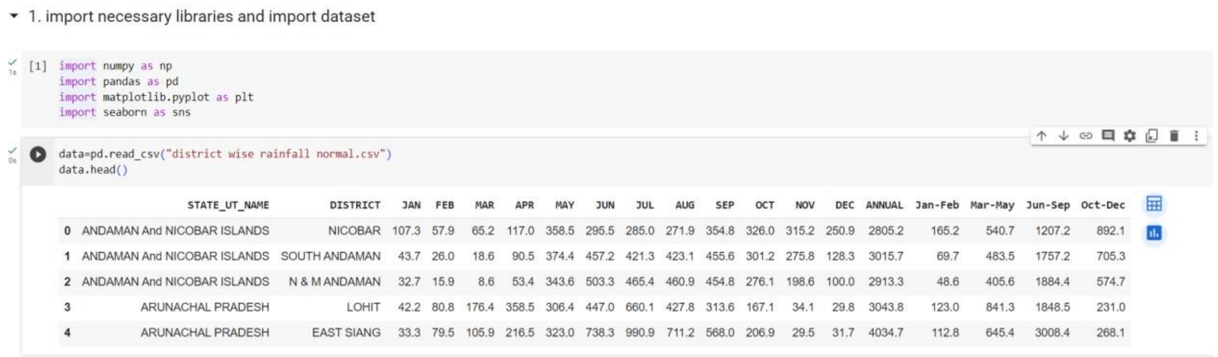
- o Import the Libraries. o Importing the dataset. o Checking for Null Values.
- o Data Visualization. o Feature Scaling. o

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 multidimensional 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

Activity 2: Importing the Dataset



1. import necessary libraries and import dataset

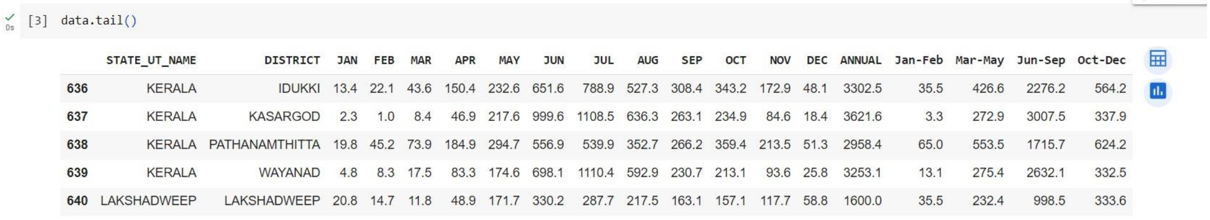
```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

```
data=pd.read_csv("district wise rainfall normal.csv")
data.head()
```

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	358.5	295.5	285.0	271.9	354.8	326.0	315.2	250.9	2805.2	165.2	540.7	1207.2	892.1
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	374.4	457.2	421.3	423.1	455.6	301.2	275.8	128.3	3015.7	69.7	483.5	1757.2	705.3
2	ANDAMAN And NICOBAR ISLANDS	N & M ANDAMAN	32.7	15.9	8.6	53.4	343.6	503.3	465.4	460.9	454.8	276.1	198.6	100.0	2913.3	48.6	405.6	1884.4	574.7
3	ARUNACHAL PRADESH	LOHIT	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	3043.8	123.0	841.3	1848.5	231.0
4	ARUNACHAL PRADESH	EAST SIANG	33.3	79.5	105.9	216.5	323.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7	4034.7	112.8	645.4	3008.4	268.1

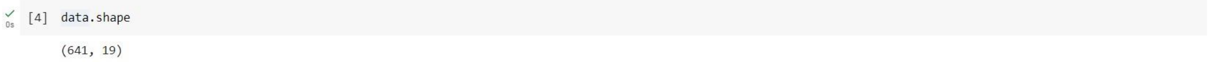
Activity 3: Analyse the data

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



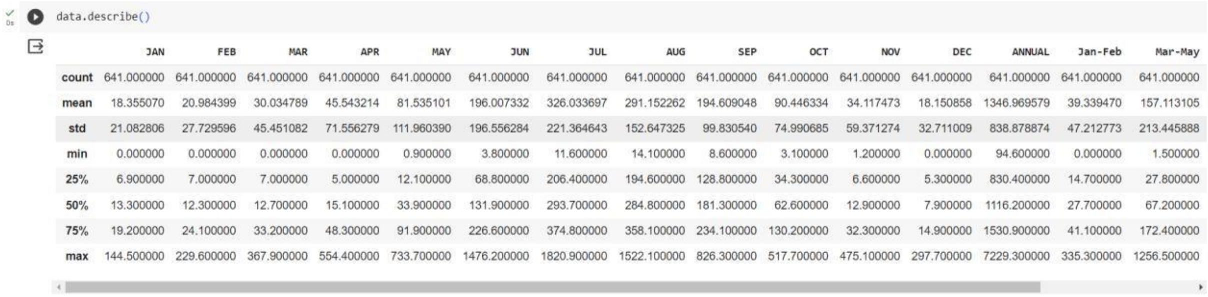
```
[3] data.tail()
```

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
636	KERALA	IDUKKI	13.4	22.1	43.6	150.4	232.6	651.6	788.9	527.3	308.4	343.2	172.9	48.1	3302.5	35.5	426.6	2276.2	564.2
637	KERALA	KASARGOD	2.3	1.0	8.4	46.9	217.6	999.6	1108.5	636.3	263.1	234.9	84.6	18.4	3621.6	3.3	272.9	3007.5	337.9
638	KERALA	PATHANAMTHITTA	19.8	45.2	73.9	184.9	294.7	556.9	539.9	352.7	266.2	359.4	213.5	51.3	2958.4	65.0	553.5	1715.7	624.2
639	KERALA	WAYANAD	4.8	8.3	17.5	83.3	174.6	698.1	1110.4	592.9	230.7	213.1	93.6	25.8	3253.1	13.1	275.4	2632.1	332.5
640	LAKSHADWEEP	LAKSHADWEEP	20.8	14.7	11.8	48.9	171.7	330.2	287.7	217.5	163.1	157.1	117.7	58.8	1600.0	35.5	232.4	998.5	333.6



```
[4] data.shape

(641, 19)
```

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

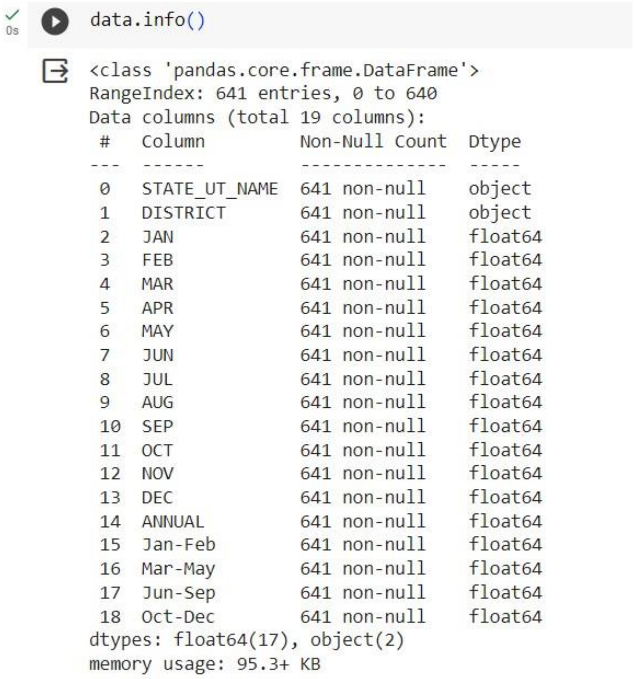


```
data.describe()
```

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May
count	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000
mean	18.355070	20.984399	30.034789	45.543214	81.535101	196.007332	326.033697	291.152262	194.609048	90.446334	34.117473	18.150858	1346.969579	39.339470	157.113105
std	21.082806	27.729596	45.451082	71.556279	111.960390	196.556284	221.364643	152.647325	99.830540	74.990685	59.371274	32.711009	838.878874	47.212773	213.445888
min	0.000000	0.000000	0.000000	0.000000	0.900000	3.800000	11.600000	14.100000	8.600000	3.100000	1.200000	0.000000	94.600000	0.000000	1.500000
25%	6.900000	7.000000	7.000000	5.000000	12.100000	68.800000	206.400000	194.600000	128.800000	34.300000	6.600000	5.300000	830.400000	14.700000	27.800000
50%	13.300000	12.300000	12.700000	15.100000	33.900000	131.900000	293.700000	284.800000	181.300000	62.600000	12.900000	7.900000	1116.200000	27.700000	67.200000
75%	19.200000	24.100000	33.200000	48.300000	91.900000	226.600000	374.800000	358.100000	234.100000	130.200000	32.300000	14.900000	1530.900000	41.100000	172.400000
max	144.500000	229.600000	367.900000	554.400000	733.700000	1476.200000	1820.900000	1522.100000	826.300000	517.700000	475.100000	297.700000	7229.300000	335.300000	1256.500000

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

- info() gives information about the data



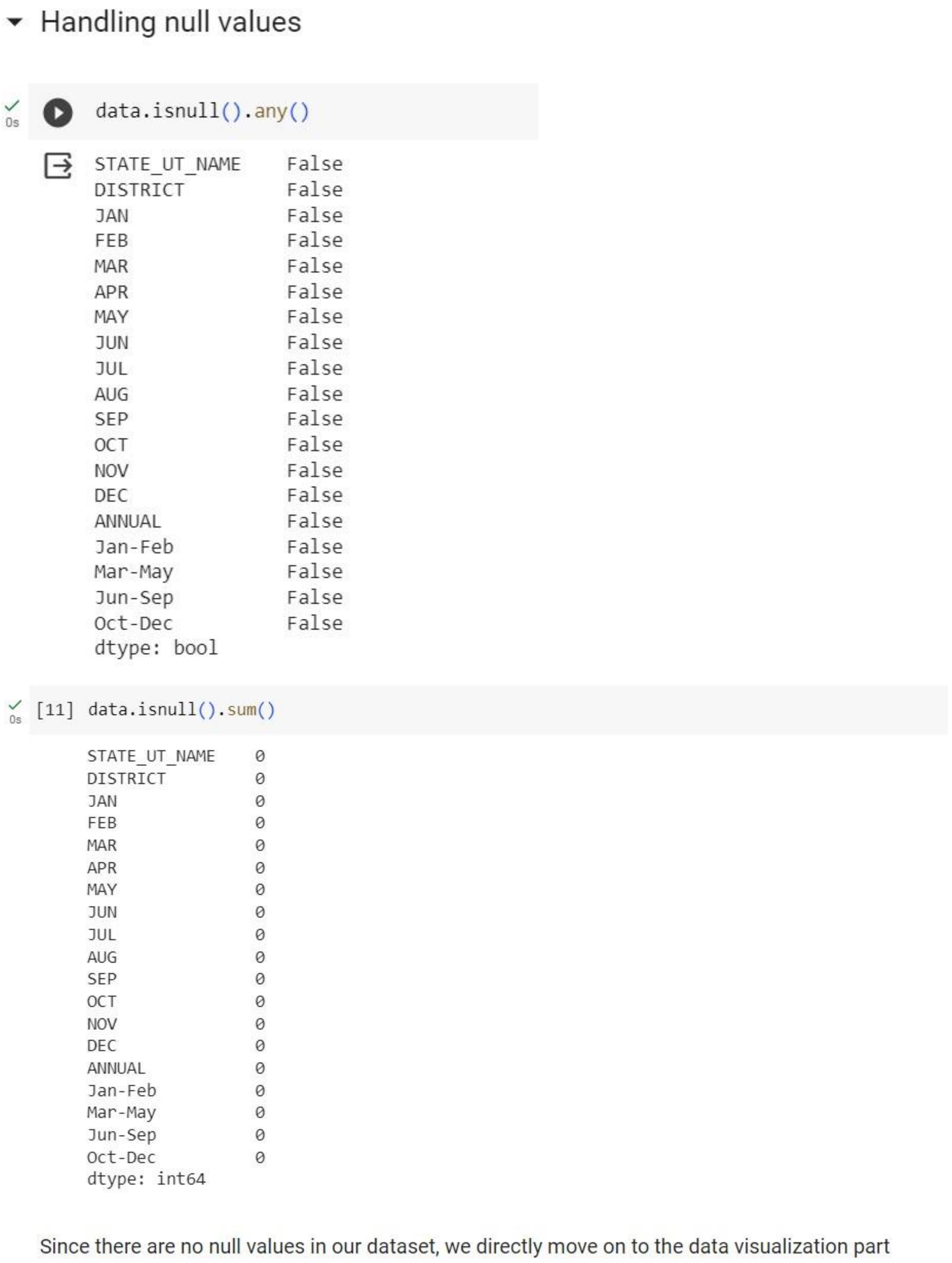
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 641 entries, 0 to 640
Data columns (total 19 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   STATE_UT_NAME  641 non-null    object
 1   DISTRICT      641 non-null    object
 2   JAN           641 non-null    float64
 3   FEB           641 non-null    float64
 4   MAR           641 non-null    float64
 5   APR           641 non-null    float64
 6   MAY           641 non-null    float64
 7   JUN           641 non-null    float64
 8   JUL           641 non-null    float64
 9   AUG           641 non-null    float64
 10  SEP           641 non-null    float64
 11  OCT           641 non-null    float64
 12  NOV           641 non-null    float64
 13  DEC           641 non-null    float64
 14  ANNUAL        641 non-null    float64
 15  Jan-Feb       641 non-null    float64
 16  Mar-May       641 non-null    float64
 17  Jun-Sep       641 non-null    float64
 18  Oct-Dec       641 non-null    float64
dtypes: float64(17), object(2)
memory usage: 95.3+ KB
```

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,



Handling null values

```
data.isnull().any()
```

```
STATE_UT_NAME    False
DISTRICT         False
JAN              False
FEB              False
MAR              False
APR              False
MAY              False
JUN              False
JUL              False
AUG              False
SEP              False
OCT              False
NOV              False
DEC              False
ANNUAL           False
Jan-Feb          False
Mar-May          False
Jun-Sep          False
Oct-Dec          False
dtype: bool
```

```
[11] data.isnull().sum()
```

```
STATE_UT_NAME    0
DISTRICT         0
JAN              0
FEB              0
MAR              0
APR              0
MAY              0
JUN              0
JUL              0
AUG              0
SEP              0
OCT              0
NOV              0
DEC              0
ANNUAL           0
Jan-Feb          0
Mar-May          0
Jun-Sep          0
Oct-Dec          0
dtype: int64
```

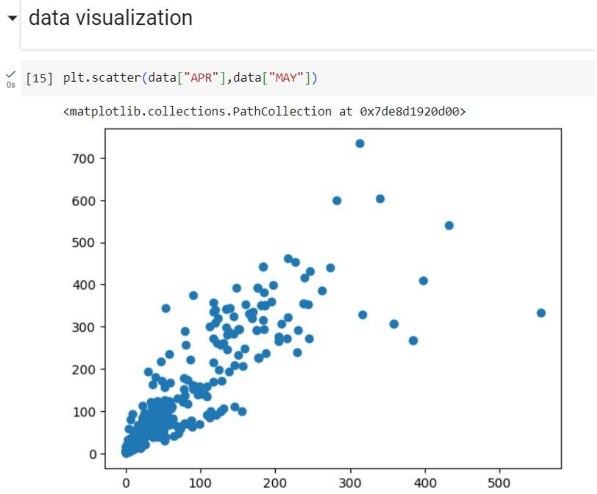
Since there are no null values in our dataset, we directly move on to the data visualization part

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.



data visualization

```
[15] plt.scatter(data["APR"],data["MAY"])
```

```
<matplotlib.collections.PathCollection at 0x7de8d1920d00>
```

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

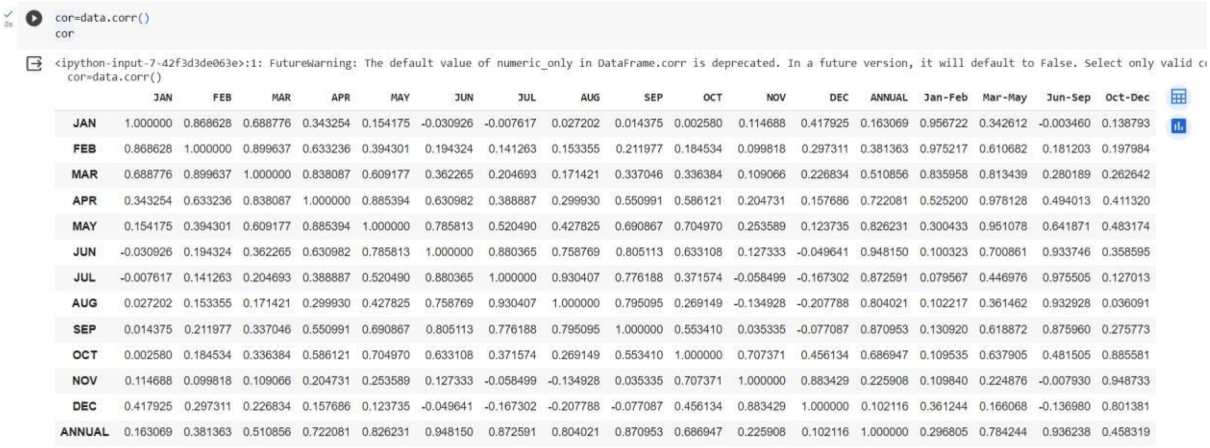
xlabel: Set the label for the x-axis. **ylabel:**

Set the label for the y-axis.

title: Set a title for the axes.

Legend: Place a legend on the axes.

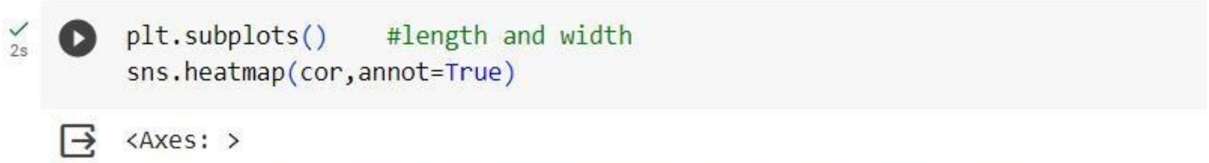
1. data.corr() gives the correlation between the columns



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

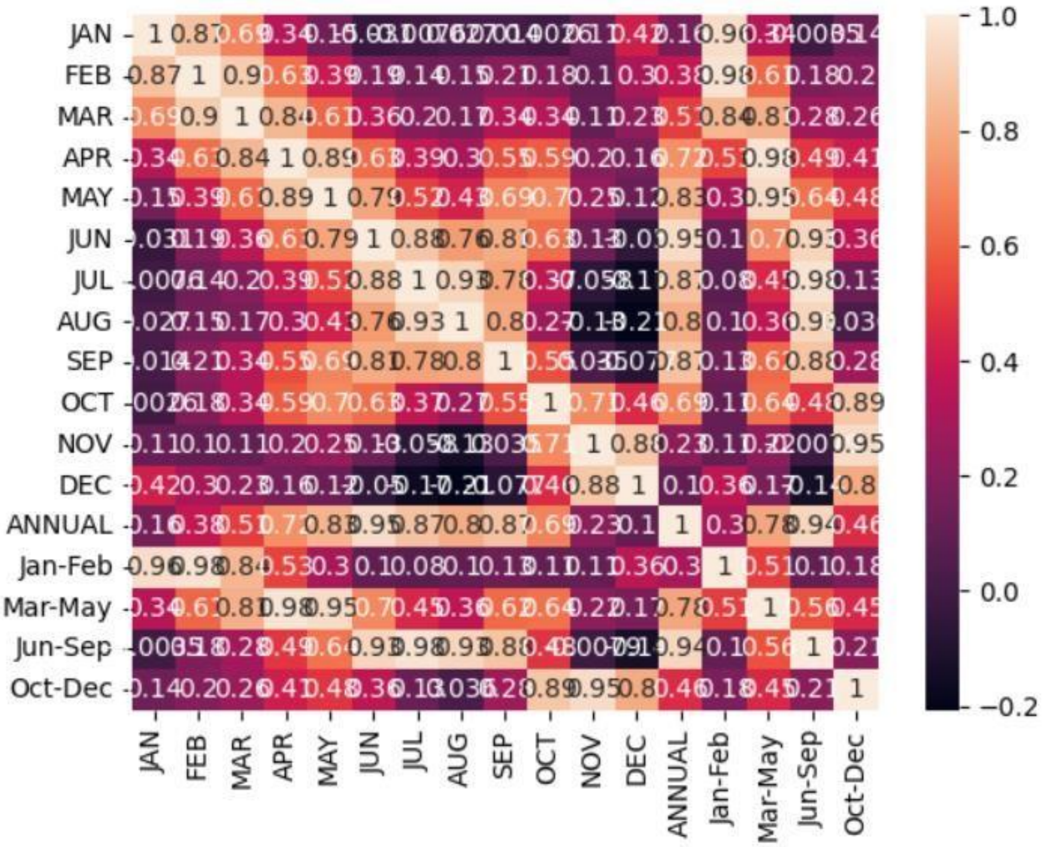
```
<ipython-input-7-42f3d3de063e>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid c
  cor=data.corr()
```

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
JAN	1.000000	0.868628	0.688776	0.343254	0.154175	-0.030926	-0.007617	0.027202	0.014375	0.002580	0.114688	0.417925	0.163069	0.956722	0.342612	-0.003460	0.138793
FEB	0.868628	1.000000	0.899637	0.633236	0.394301	0.194324	0.141263	0.153355	0.211977	0.184534	0.099818	0.297311	0.381363	0.975217	0.610682	0.181203	0.197984
MAR	0.688776	0.899637	1.000000	0.838087	0.609177	0.362265	0.204693	0.171421	0.337046	0.336384	0.109066	0.226834	0.510856	0.835958	0.813439	0.280189	0.262642
APR	0.343254	0.633236	0.838087	1.000000	0.885394	0.630982	0.388887	0.299930	0.550991	0.586121	0.204731	0.157686	0.722081	0.525200	0.978128	0.494013	0.411320
MAY	0.154175	0.394301	0.609177	0.885394	1.000000	0.785813	0.520490	0.427825	0.690867	0.704970	0.253589	0.123735	0.826231	0.300433	0.951078	0.641871	0.483174
JUN	-0.030926	0.194324	0.362265	0.630982	0.785813	1.000000	0.880365	0.758769	0.805113	0.633108	0.127333	-0.049641	0.948150	0.100323	0.700861	0.933746	0.358595
JUL	-0.007617	0.141263	0.204693	0.388887	0.520490	0.880365	1.000000	0.930407	0.776188	0.371574	-0.058499	-0.167302	0.872591	0.079567	0.446976	0.975505	0.127013
AUG	0.027202	0.153355	0.171421	0.299930	0.427825	0.758769	0.930407	1.000000	0.795095	0.269149	-0.134928	-0.207788	0.804021	0.102217	0.361462	0.932928	0.036091
SEP	0.014375	0.211977	0.337046	0.550991	0.690867	0.805113	0.776188	0.795095	1.000000	0.553410	0.035335	-0.077087	0.870953	0.130920	0.618872	0.875960	0.275773
OCT	0.002580	0.184534	0.336384	0.586121	0.704970	0.633108	0.371574	0.269149	0.553410	1.000000	0.707371	0.456134	0.686947	0.109535	0.637905	0.481505	0.885581
NOV	0.114688	0.099818	0.109066	0.204731	0.253589	0.127333	-0.058499	-0.134928	0.035335	0.707371	1.000000	0.883429	0.225908	0.109840	0.224876	-0.007930	0.948733
DEC	0.417925	0.297311	0.226834	0.157686	0.123735	-0.049641	-0.167302	-0.207788	-0.077087	0.456134	0.883429	1.000000	0.102116	0.361244	0.166068	-0.136980	0.801381
ANNUAL	0.163069	0.381363	0.510856	0.722081	0.826231	0.948150	0.872591	0.804021	0.870953	0.686947	0.225908	0.102116	1.000000	0.296805	0.784244	0.936238	0.458319



```
plt.subplots()     #length and width
sns.heatmap(cor,annot=True)
```

```
<Axes: >
```



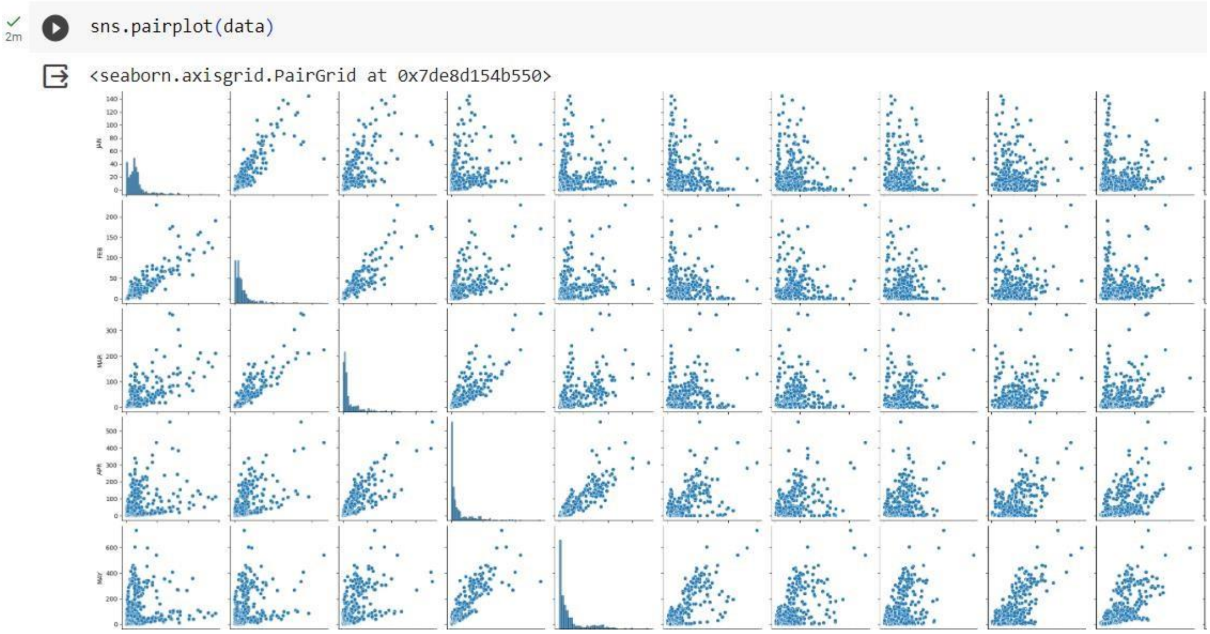
- Correlation strength varies based on colour, lighter the colour between two variables, more the strength between the variables, darker the colour displays the weaker correlation ● We can see the correlation scale values on left side of the above image

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

Code:- sns.pairplot(data)

The output is as shown below



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

```
<seaborn.axisgrid.PairGrid at 0x7de8d154b550>
```

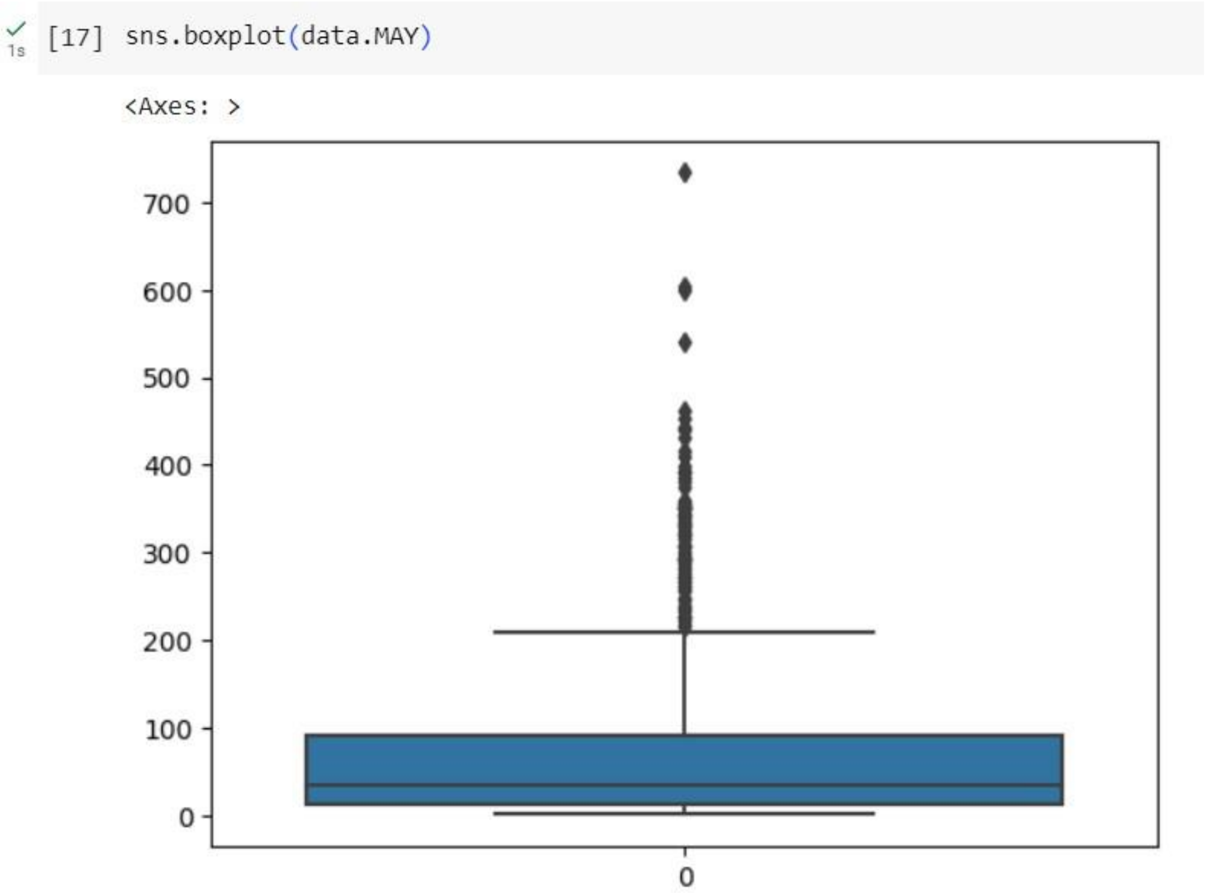
Pair plot usually gives pair wise relationships of the columns in the dataset

From the above pairplot we infer that

1.from the above plot we can draw inferences such as linearity and strength between the variables

2.how features are correlated(positive, neutral and negative)

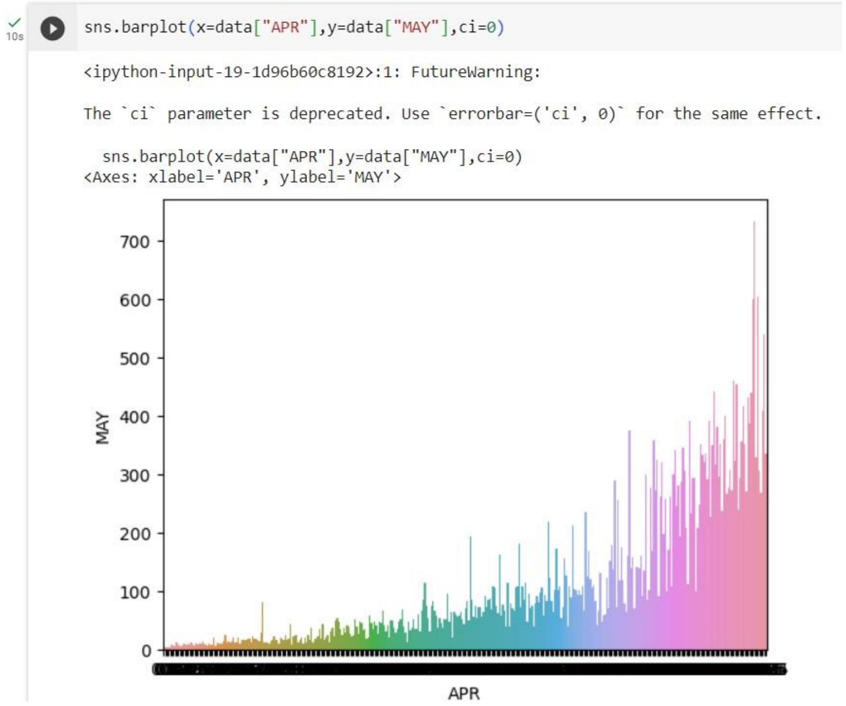
3.Box Plot: jupyter has a built-in function to create boxplot called boxplot(). A boxplot plot is a type of plot that shows the spread of data in all the quartiles



```
[17] sns.boxplot(data.MAY)
```

```
<Axes: >
```

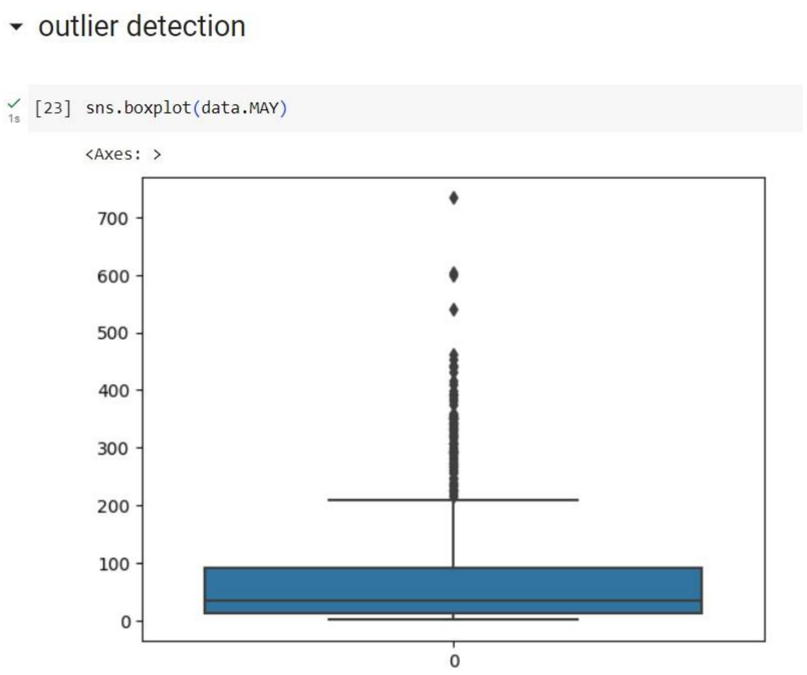
From the above box plot we infer how the datapoints are spread and the existence of the outliers



```
sns.barplot(x=data["APR"],y=data["MAY"],ci=0)
```

```
<ipython-input-19-1d96b60c8192>:1: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=('ci', 0)` for the same effect.

  sns.barplot(x=data["APR"],y=data["MAY"],ci=0)
<Axes: xlabel='APR', ylabel='MAY'>
```

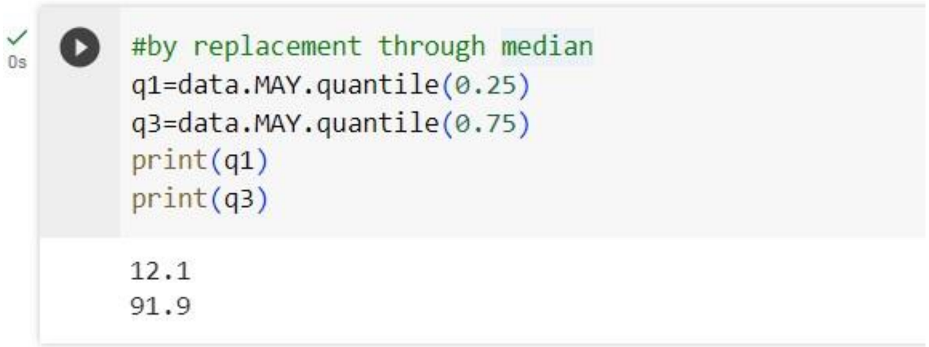
OUTLIER DETECTION



outlier detection

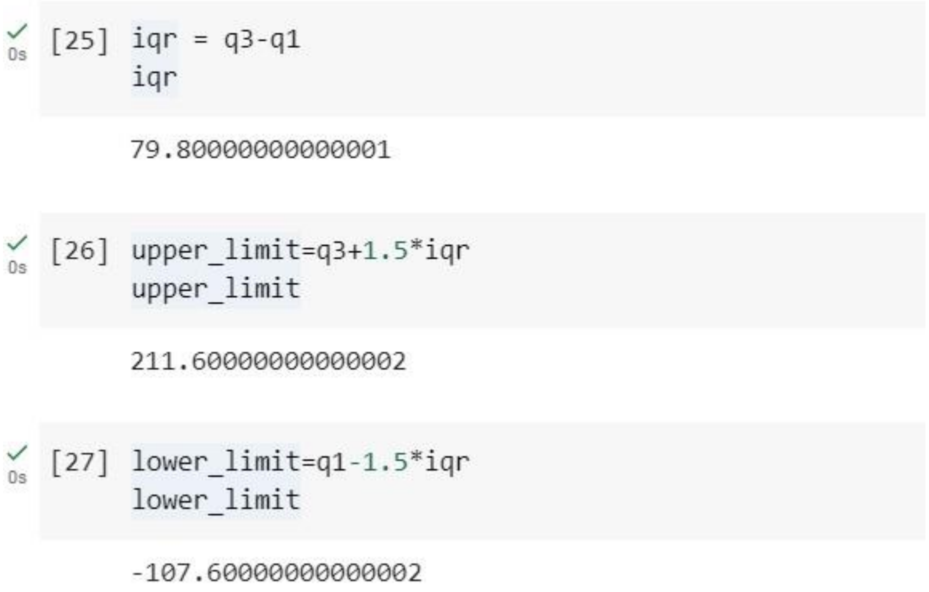
```
[23] sns.boxplot(data.MAY)
```

```
<Axes: >
```



```
#by replacement through median
q1=data.MAY.quantile(0.25)
q3=data.MAY.quantile(0.75)
print(q1)
print(q3)
```

```
12.1
91.9
```



```
[25] iqr = q3-q1
     iqr
```

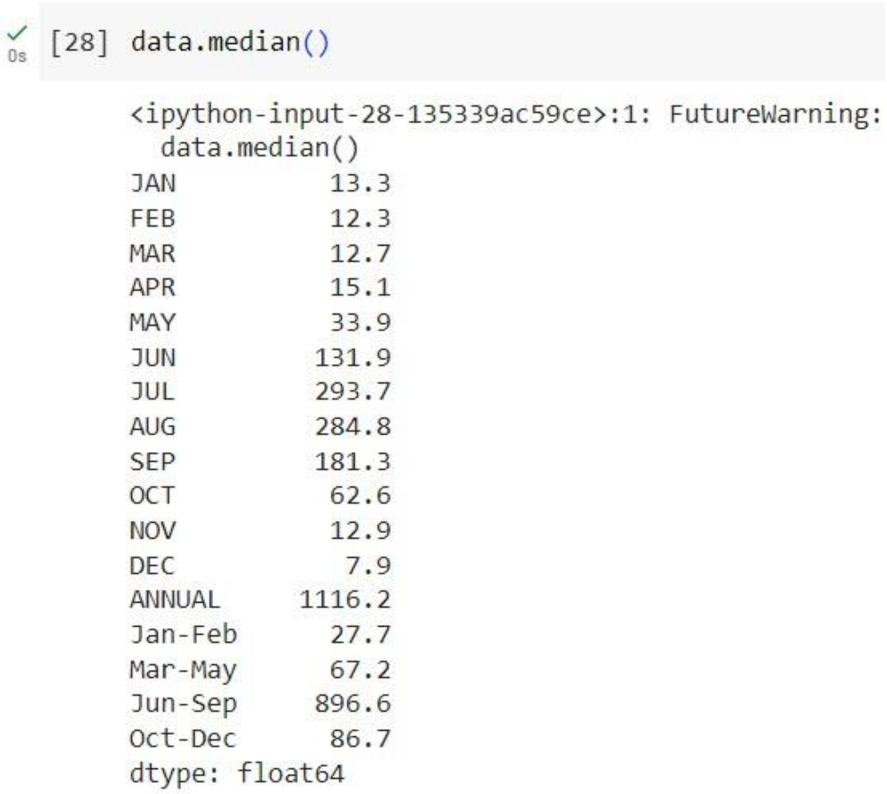
```
79.80000000000001
```

```
[26] upper_limit=q3+1.5*iqr
     upper_limit
```

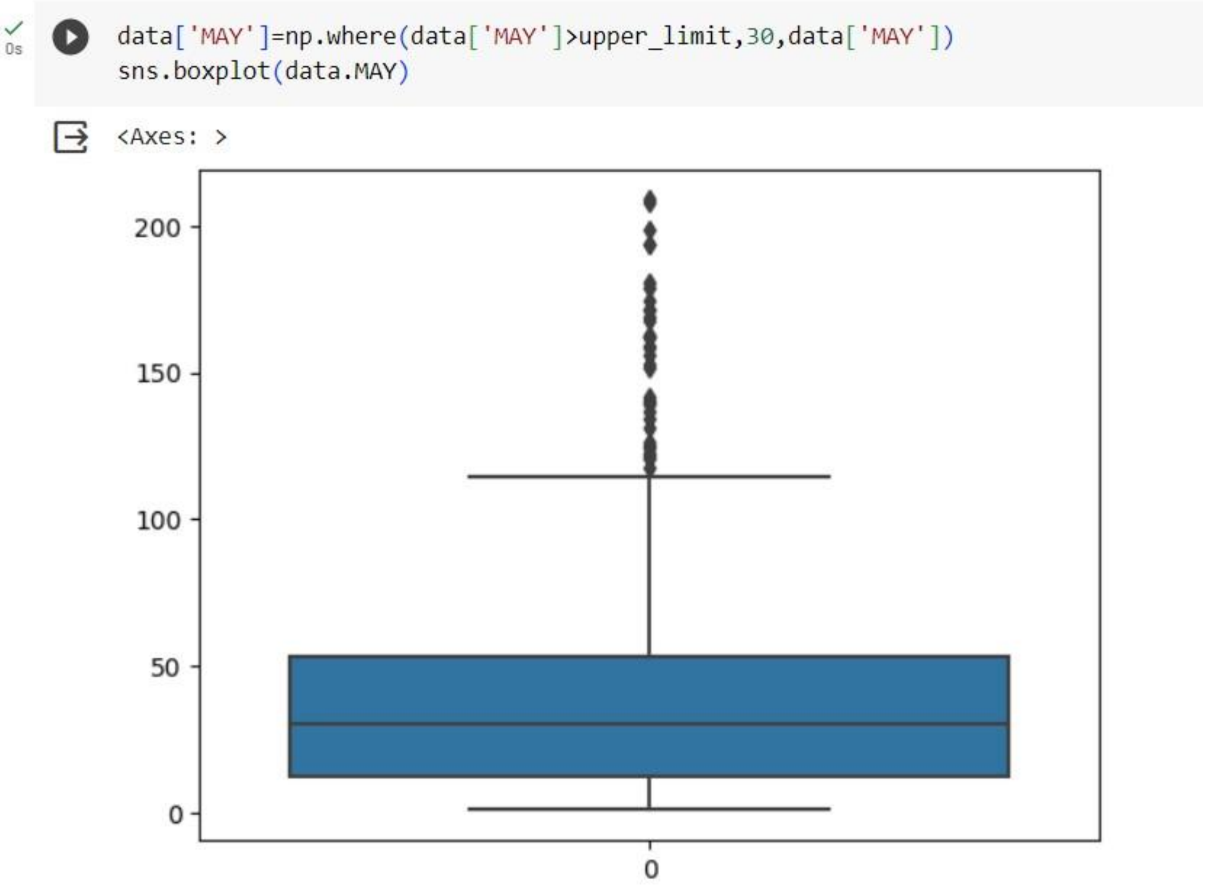
```
211.60000000000002
```

```
[27] lower_limit=q1-1.5*iqr
     lower_limit
```

```
-107.60000000000002
```



```
[28] data.median()

     <ipython-input-28-135339ac59ce>:1: FutureWarning:
       data.median()
     JAN           13.3
     FEB           12.3
     MAR           12.7
     APR           15.1
     MAY           33.9
     JUN          131.9
     JUL          293.7
     AUG          284.8
     SEP          181.3
     OCT           62.6
     NOV           12.9
     DEC            7.9
     ANNUAL      1116.2
     Jan-Feb       27.7
     Mar-May       67.2
     Jun-Sep      896.6
     Oct-Dec       86.7
     dtype: float64
```



```
data['MAY']=np.where(data['MAY']>upper_limit,30,data['MAY'])
sns.boxplot(data.MAY)
```

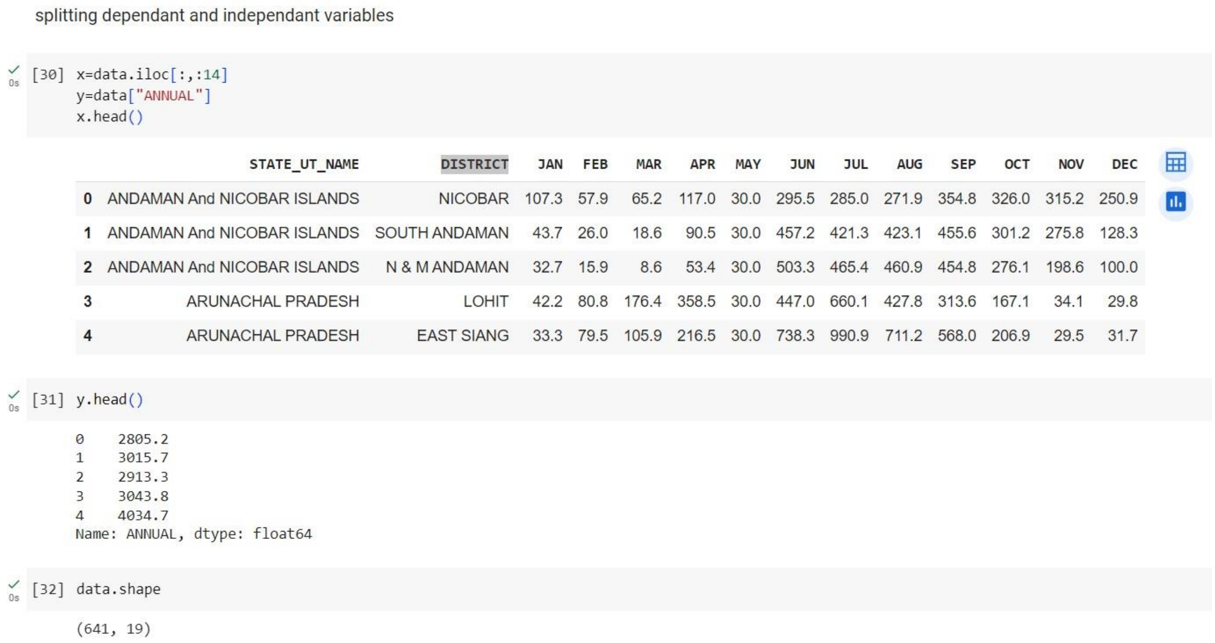
```
<Axes: >
```

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.



splitting dependant and independant variables

```
[30] x=data.iloc[:,:14]
     y=data["ANNUAL"]
     x.head()
```

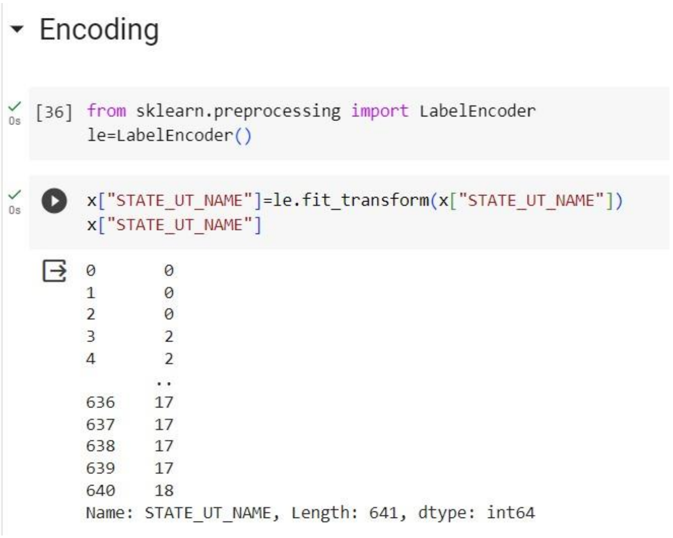
	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	30.0	295.5	285.0	271.9	354.8	326.0	315.2	250.9
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	30.0	457.2	421.3	423.1	455.6	301.2	275.8	128.3
2	ANDAMAN And NICOBAR ISLANDS	N & M ANDAMAN	32.7	15.9	8.6	53.4	30.0	503.3	465.4	460.9	454.8	276.1	198.6	100.0
3	ARUNACHAL PRADESH	LOHIT	42.2	80.8	176.4	358.5	30.0	447.0	660.1	427.8	313.6	167.1	34.1	29.8
4	ARUNACHAL PRADESH	EAST SIANG	33.3	79.5	105.9	216.5	30.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7

```
[31] y.head()
```

```
0    2805.2
1    3015.7
2    2913.3
3    3043.8
4    4034.7
Name: ANNUAL, dtype: float64
```

```
[32] data.shape
```

```
(641, 19)
```

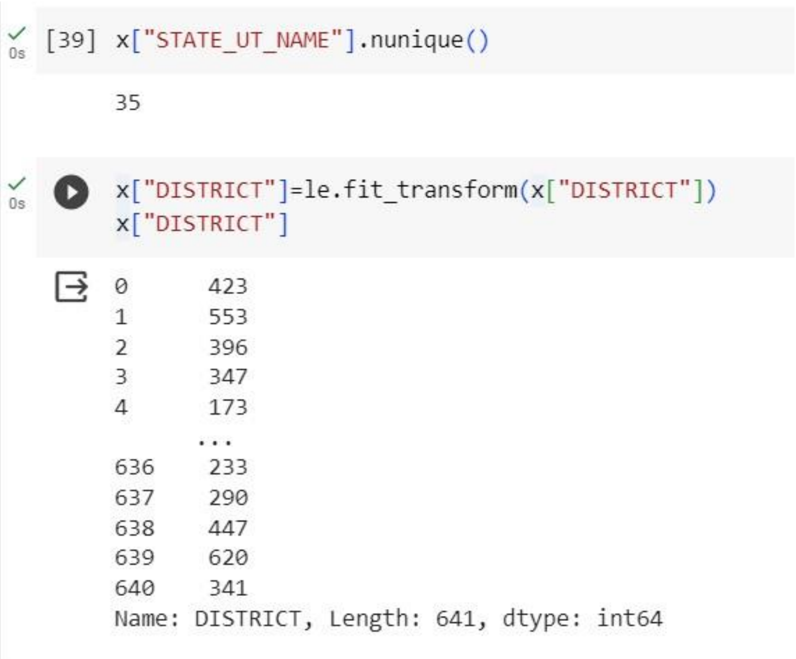


Encoding

```
[36] from sklearn.preprocessing import LabelEncoder
     le=LabelEncoder()
```

```
x["STATE_UT_NAME"]=le.fit_transform(x["STATE_UT_NAME"])
x["STATE_UT_NAME"]
```

```
0       0
1       0
2       0
3       2
4       2
       ..
636    17
637    17
638    17
639    17
640    18
Name: STATE_UT_NAME, Length: 641, dtype: int64
```

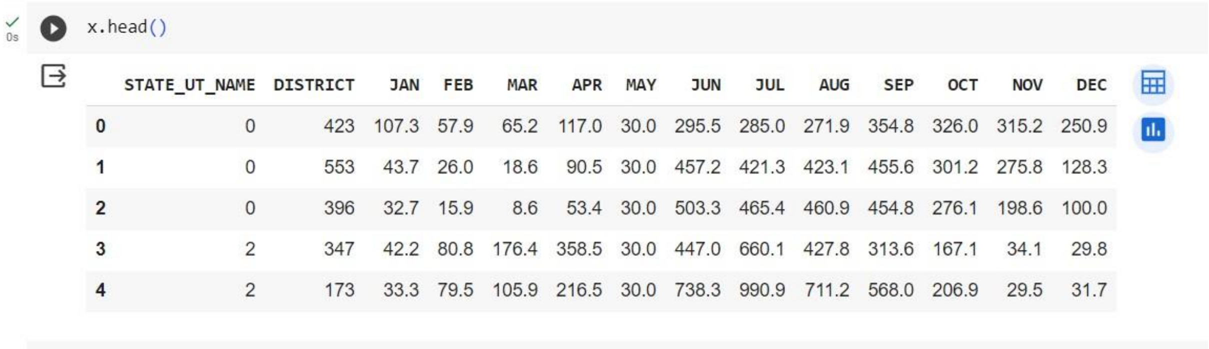


```
[39] x["STATE_UT_NAME"].nunique()
```

```
35
```

```
x["DISTRICT"]=le.fit_transform(x["DISTRICT"])
x["DISTRICT"]
```

```
0      423
1      553
2      396
3      347
4      173
      ...
636    233
637    290
638    447
639    620
640    341
Name: DISTRICT, Length: 641, dtype: int64
```



```
x.head()
```

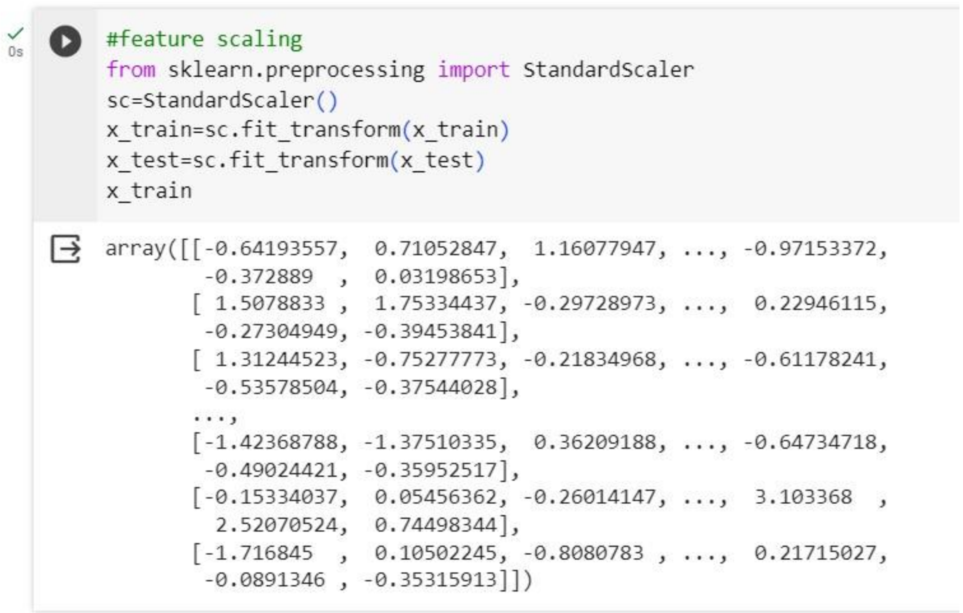
	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
0	0	423	107.3	57.9	65.2	117.0	30.0	295.5	285.0	271.9	354.8	326.0	315.2	250.9
1	0	553	43.7	26.0	18.6	90.5	30.0	457.2	421.3	423.1	455.6	301.2	275.8	128.3
2	0	396	32.7	15.9	8.6	53.4	30.0	503.3	465.4	460.9	454.8	276.1	198.6	100.0
3	2	347	42.2	80.8	176.4	358.5	30.0	447.0	660.1	427.8	313.6	167.1	34.1	29.8
4	2	173	33.3	79.5	105.9	216.5	30.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7

Activity 7: Feature Scaling

There is huge disparity between the x values so let us use feature scaling.

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

- 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



```
#feature scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
x_train
```

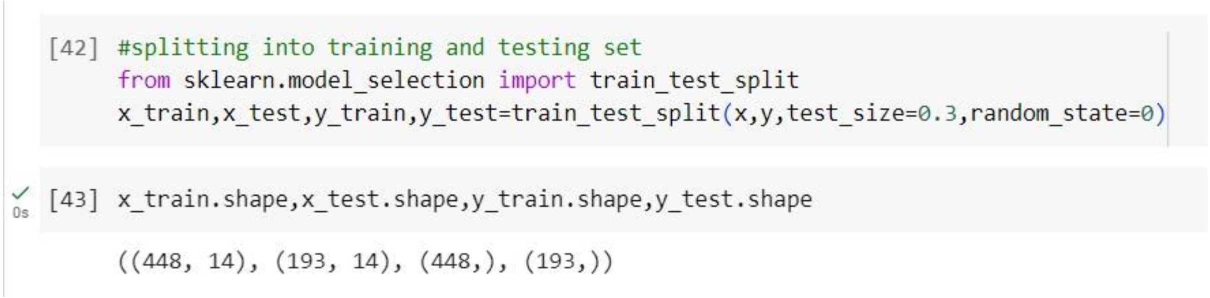
```
array([[-0.64193557,  0.71052847,  1.16077947, ..., -0.97153372,
        -0.372889  ,  0.03198653],
       [ 1.5078833 ,  1.75334437, -0.29728973, ...,  0.22946115,
        -0.27304949, -0.39453841],
       [ 1.31244523, -0.75277773, -0.21834968, ..., -0.61178241,
        -0.53578504, -0.37544028],
       ...,
       [-1.42368788, -1.37510335,  0.36209188, ..., -0.64734718,
        -0.49024421, -0.35952517],
       [-0.15334037,  0.05456362, -0.26014147, ...,  3.103368  ,
         2.52070524,  0.74498344],
       [-1.716845  ,  0.10502245, -0.8080783 , ...,  0.21715027,
        -0.0891346 , -0.35315913]])
```

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,random_state =0)



```
[42] #splitting into training and testing set
     from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
```

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

     ((448, 14), (193, 14), (448,), (193,))
```

Milestone 3: Model Building:

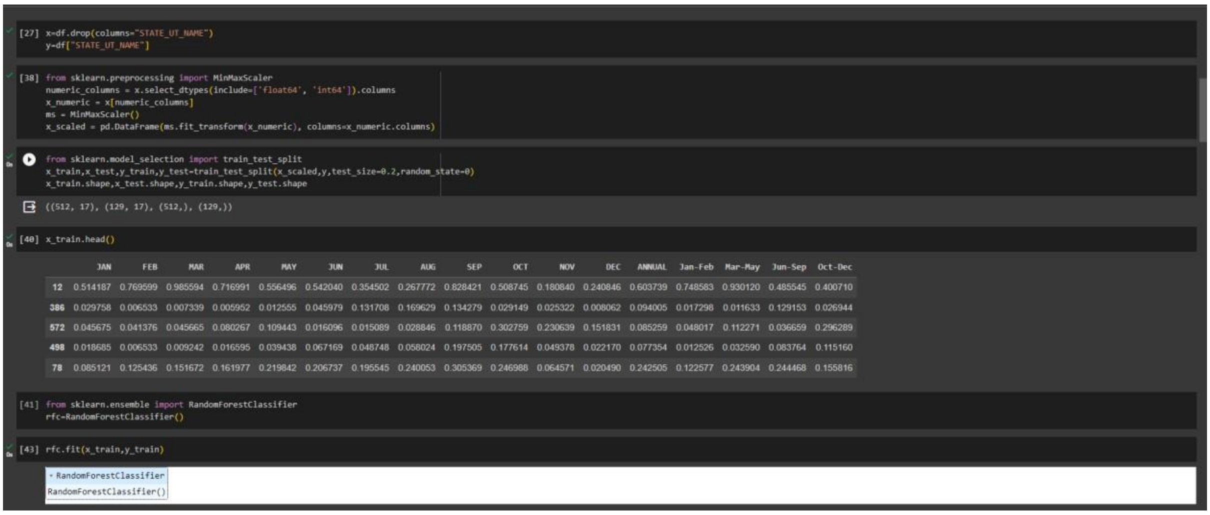
Model building includes the following main tasks

- 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

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



```
[27] x=df.drop(columns="STATE_UT_NAME")
     y=df["STATE_UT_NAME"]
```

```
[38] from sklearn.preprocessing import MinMaxScaler
     numeric_columns = x.select_dtypes(include=['float64', 'int64']).columns
     x_numeric = x[numeric_columns]
     ms = MinMaxScaler()
     x_scaled = pd.DataFrame(ms.fit_transform(x_numeric), columns=x_numeric.columns)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.2,random_state=0)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((512, 17), (129, 17), (512,), (129,))
```

```
[40] x_train.head()
```

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
12	0.514187	0.769599	0.985594	0.716991	0.556496	0.542040	0.354502	0.267772	0.828421	0.508745	0.180840	0.240846	0.603739	0.748583	0.930120	0.485545	0.400710
386	0.029758	0.006533	0.007339	0.005952	0.012555	0.045979	0.131708	0.169629	0.134279	0.029149	0.025322	0.008062	0.094005	0.017298	0.011633	0.129153	0.026944
572	0.045675	0.041376	0.045665	0.080267	0.109443	0.016096	0.015089	0.028846	0.118870	0.302759	0.230639	0.151831	0.085259	0.048017	0.112271	0.036659	0.296289
498	0.018685	0.006533	0.009242	0.016595	0.039438	0.067169	0.048748	0.058024	0.197505	0.177614	0.049378	0.022170	0.077354	0.012526	0.032590	0.083764	0.115160
78	0.085121	0.125436	0.151672	0.161977	0.219842	0.206737	0.195545	0.240053	0.305369	0.246988	0.064571	0.020490	0.242505	0.122577	0.243904	0.244468	0.155816

```
[41] from sklearn.ensemble import RandomForestClassifier
     rfc=RandomForestClassifier()
```

```
[43] rfc.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

Activity 2: Model Evaluation

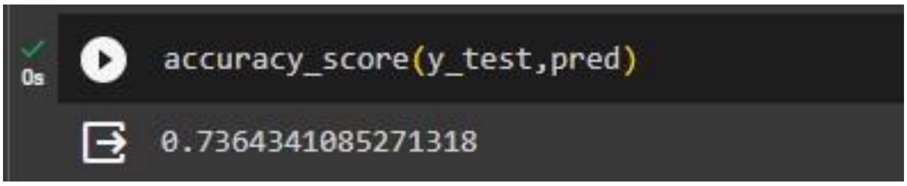
Regression Evaluation Metrics:

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

- Accuracy_score
- Confusion matrix

1. Accuracy_Score

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

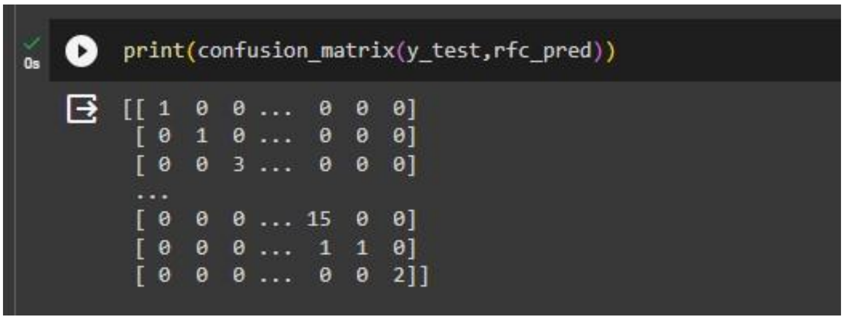


```
accuracy_score(y_test,pred)
```

```
0.7364341085271318
```

2. Confusion Matrix

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



```
print(confusion_matrix(y_test,rfc_pred))
```

```
[[ 1  0  0 ...  0  0  0]
 [ 0  1  0 ...  0  0  0]
 [ 0  0  3 ...  0  0  0]
 ...
 [ 0  0  0 ... 15  0  0]
 [ 0  0  0 ...  1  1  0]
 [ 0  0  0 ...  0  0  2]]
```

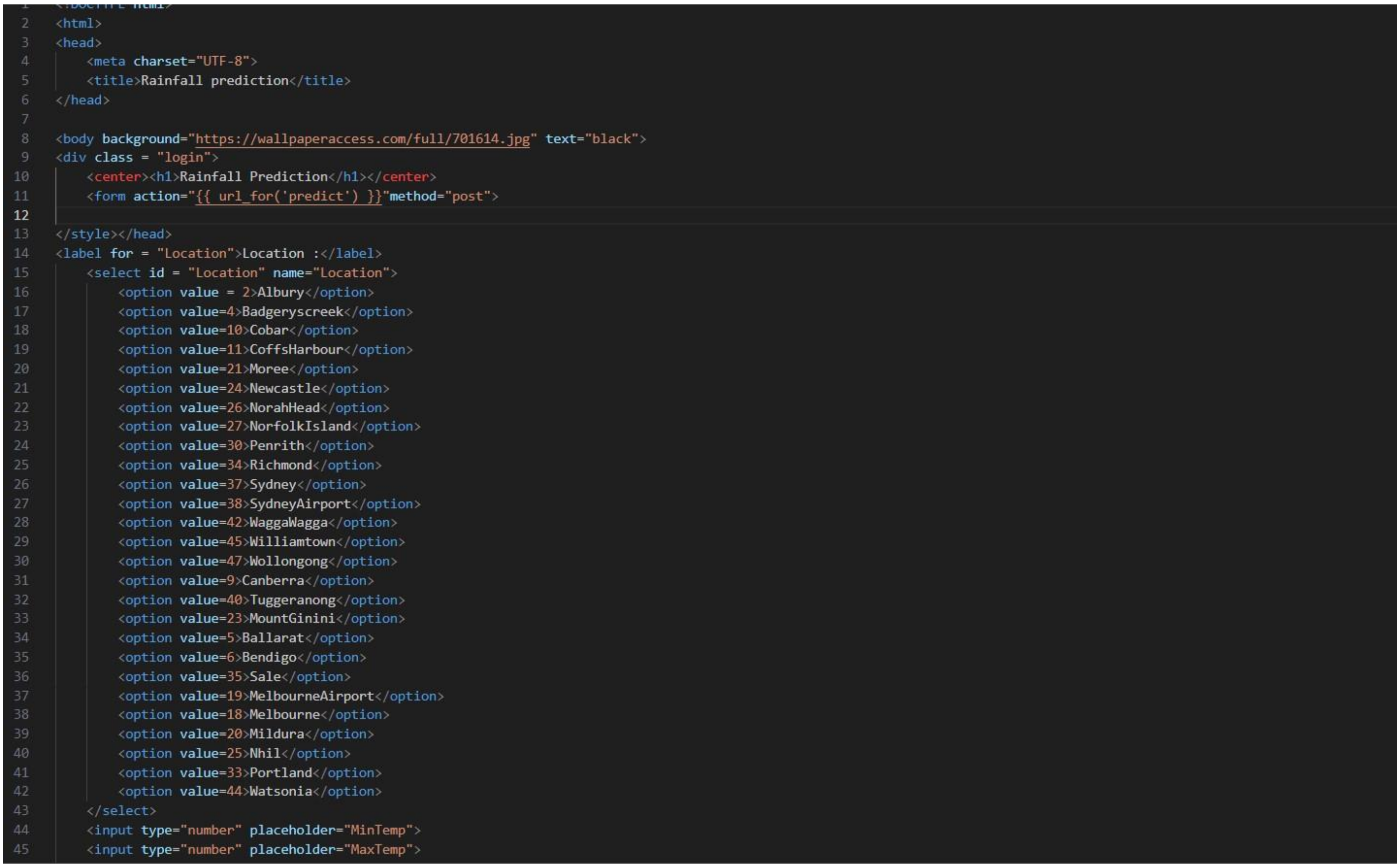
Milestone 4 : Application Building

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

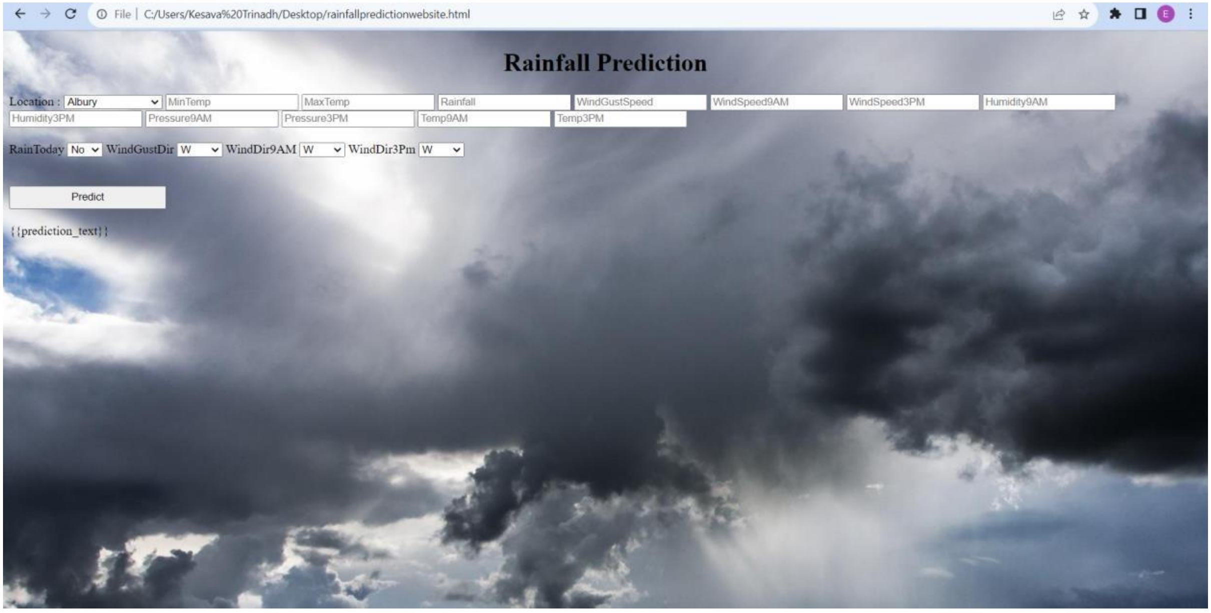
- Building HTML Pages
- Building server-side script

Activity 1: Build HTML Code



```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Rainfall prediction</title>
</head>

<body background="https://wallpaperaccess.com/full/701614.jpg" text="black">
<div class = "login">
    <center><h1>Rainfall Prediction</h1></center>
    <form action="{{ url_for('predict') }}"method="post">
    
</style></head>
<label for = "Location">Location :</label>
    <select id = "Location" name="Location">
        <option value = 2>Albury</option>
        <option value=4>Badgeryscreek</option>
        <option value=10>Cobar</option>
        <option value=11>CoffsHarbour</option>
        <option value=21>Moree</option>
        <option value=24>Newcastle</option>
        <option value=26>NorahHead</option>
        <option value=27>NorfolkIsland</option>
        <option value=30>Penrith</option>
        <option value=34>Richmond</option>
        <option value=37>Sydney</option>
        <option value=38>SydneyAirport</option>
        <option value=42>WaggaWagga</option>
        <option value=45>Williamtown</option>
        <option value=47>Wollongong</option>
        <option value=9>Canberra</option>
        <option value=40>Tuggeranong</option>
        <option value=23>MountGinini</option>
        <option value=5>Ballarat</option>
        <option value=6>Bendigo</option>
        <option value=35>Sale</option>
        <option value=19>MelbourneAirport</option>
        <option value=18>Melbourne</option>
        <option value=20>Mildura</option>
        <option value=25>Nhil</option>
        <option value=33>Portland</option>
        <option value=44>Watsonia</option>
    </select>
    <input type="number" placeholder="MinTemp">
    <input type="number" placeholder="MaxTemp">
```

The html page looks like



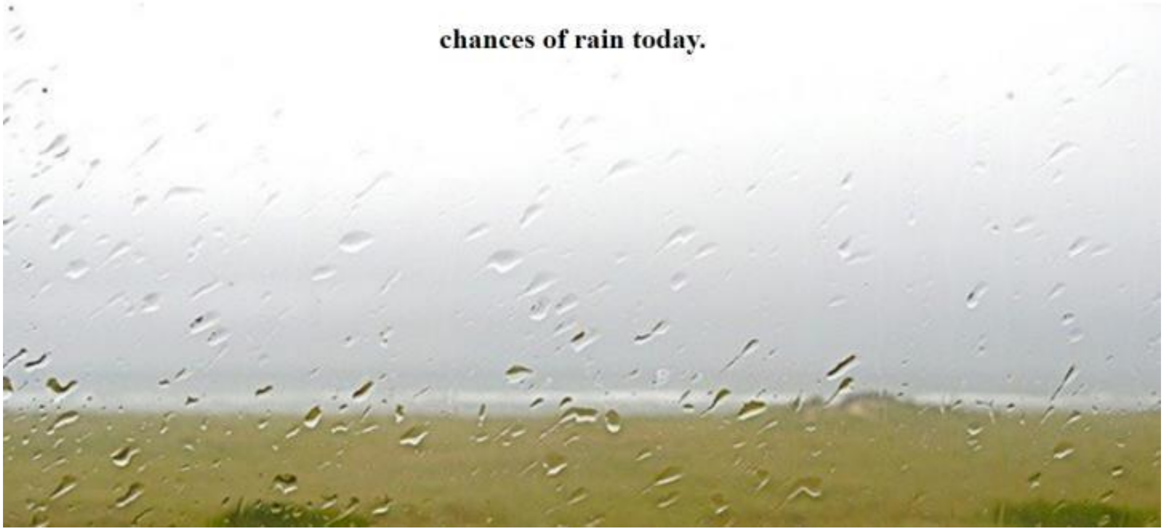
File | C:/Users/Kesava%20Trinadh/Desktop/rainfallpredictionwebsite.html

Rainfall Prediction

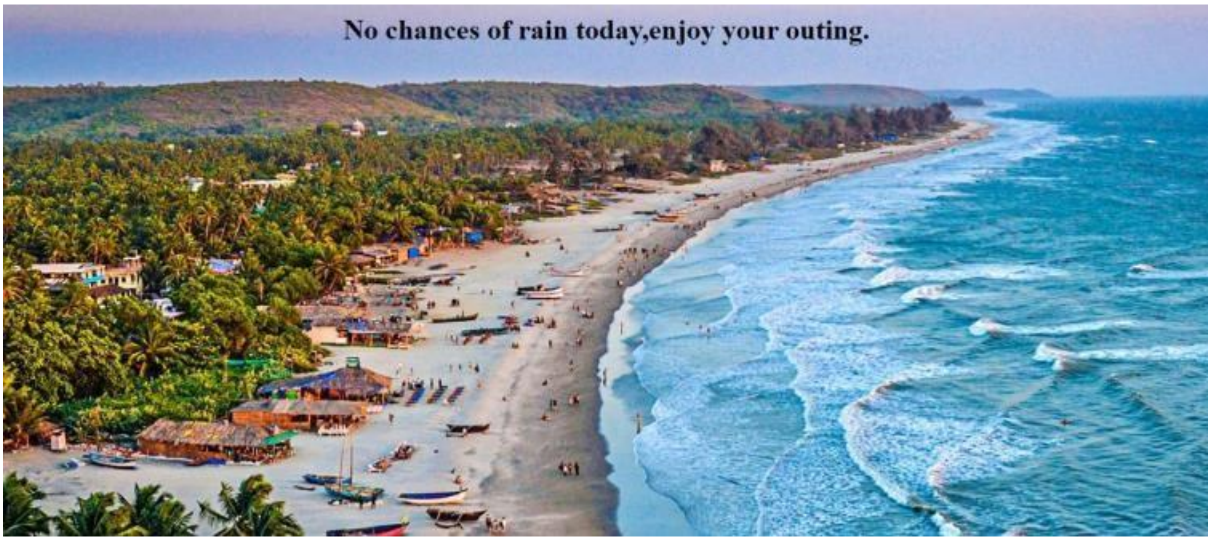
Location : Albury MinTemp MaxTemp Rainfall WindGustSpeed WindSpeed9AM WindSpeed3PM Humidity9AM
Humidity3PM Pressure9AM Pressure3PM Temp9AM Temp3PM

RainToday No WindGustDir W WindDir9AM W WindDir3Pm W

{{prediction_text}}



chances of rain today.

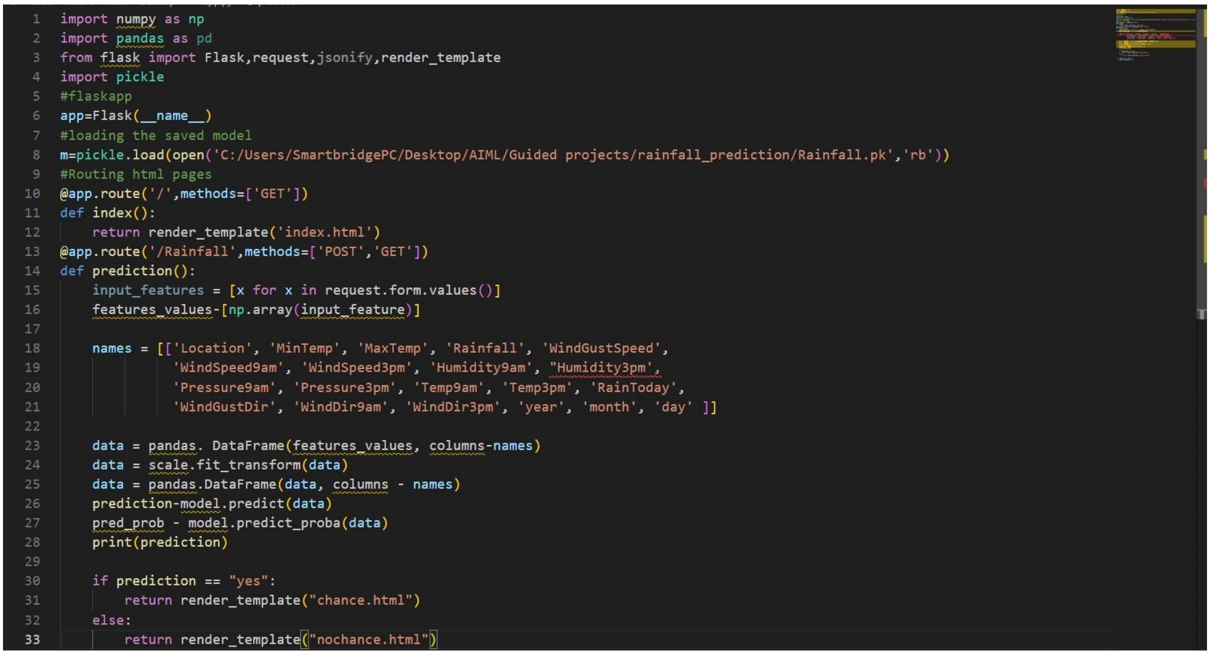


No chances of rain today,enjoy your outing.

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.

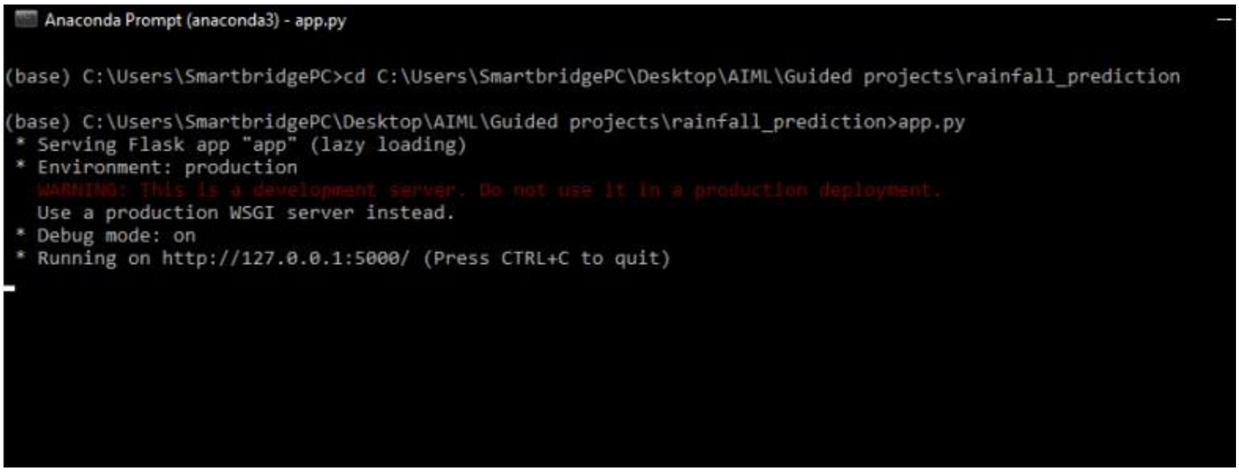
In order to develop web api with respect to our model, we basically use Flask framework which is written in python.



```python
import numpy as np
import pandas as pd
from flask import Flask,request,jsonify,render_template
import pickle
#flaskapp
app=Flask(__name__)
#loading the saved model
m=pickle.load(open('C:/Users/SmartbridgePC/Desktop/AIML/Guided projects/rainfall_prediction/Rainfall.pk','rb'))
#Routing html pages
@app.route('/',methods=['GET'])
def index():
    return render_template('index.html')
@app.route('/Rainfall',methods=['POST','GET'])
def prediction():
    input_features = [x for x in request.form.values()]
    features_values-[np.array(input_feature)]

    names = [['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed',
              'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', "Humidity3pm',
              'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm', 'RainToday',
              'WindGustDir', 'WindDir9am', 'WindDir3pm', 'year', 'month', 'day' ]]

    data = pandas. DataFrame(features_values, columns-names)
    data = scale.fit_transform(data)
    data = pandas.DataFrame(data, columns - names)
    prediction-model.predict(data)
    pred_prob - model.predict_proba(data)
    print(prediction)

    if prediction == "yes":
        return render_template("chance.html")
    else:
        return render_template("nochance.html")
```

Activity 3: Run the App

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



Anaconda Prompt (anaconda3) - app.py

```
(base) C:\Users\SmartbridgePC>cd C:\Users\SmartbridgePC\Desktop\AIML\Guided projects\rainfall_prediction

(base) C:\Users\SmartbridgePC\Desktop\AIML\Guided projects\rainfall_prediction>app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```