

FLOOD PREDICTION

Name : K.Y.Sai Santosh

Reg No : 39110520

College : Sathyabama Institute of Science and Technology

Branch : B.E , CSE.

Project Title : Flood prediction using Machine Learning.

INTRODUCTION

Floods are inevitable, but with timely alerts, their effects can be minimized. There are many people who die every year due to devastating floods, the number of people become homeless and many people die due to lack of proper help after a flood. The lack of timely alerts has always been an issue concerning it.

Delay in alerts in flood-prone areas is the biggest loophole of an economy. Conventional systems run a little low in forecasting floods at the right time so that proper actions could be taken before any disaster.

By using machine learning we can predict floods or forecast floods with better accuracy. This project aims at building predictive modeling based on the historical weather data of particular areas in order to predict the occurrence of floods.

The predictive model is built on different machine learning algorithms. The concerned authority monitor this flood prediction system through a web application.

LITERATURE SURVEY

For this project, we need to use classification algorithms as the output we will get is categorical values which is either floods will occur or not so, we need to build the most accurate classification algorithm to get the desired output

In this project, we will be using classification algorithms such as Decision tree, Random forest, KNN, and xgboost. We will train and test the data with these algorithms. From this best model is selected and saved in save format. We will be doing flask integration and cloud deployment.

THEORETICAL ANALYSIS

1. Install Required Libraries.
2. Data Collection.
 - Collect the dataset or Create the dataset
3. DataPreprocessing.
 - Import the Libraries.
 - Importing the dataset.
 - Understanding Data Type and Summary of features.
 - Take care of missing data
 - Data Visualization.
 - Drop the column from DataFrame & replace the missing value.
 - Splitting the Dataset into Dependent and Independent variables
 - Splitting Data into Train and Test.
4. Model Building
 - Training and testing the model
 - Evaluation of Model
 - Saving the Model

5. Application Building
 - Create an HTML file
 - Build a Python Code
6. Final UI
 - Dashboard Of the flask app.

HARDWARE / SOFTWARE REQUIREMENTS

Hardware Requirements :

- > Processor : Intel(R) Core(TM) i3 - 6100U 2.30GHz / Equivalent processor
- > RAM : 4Gb or more
- >Storage : A basic hard disk of storing files.

Software requirements :

- >Anaconda Software
- >Visual Studio code / Spyder Software

EXPERIMENTAL INVESTIGATIONS

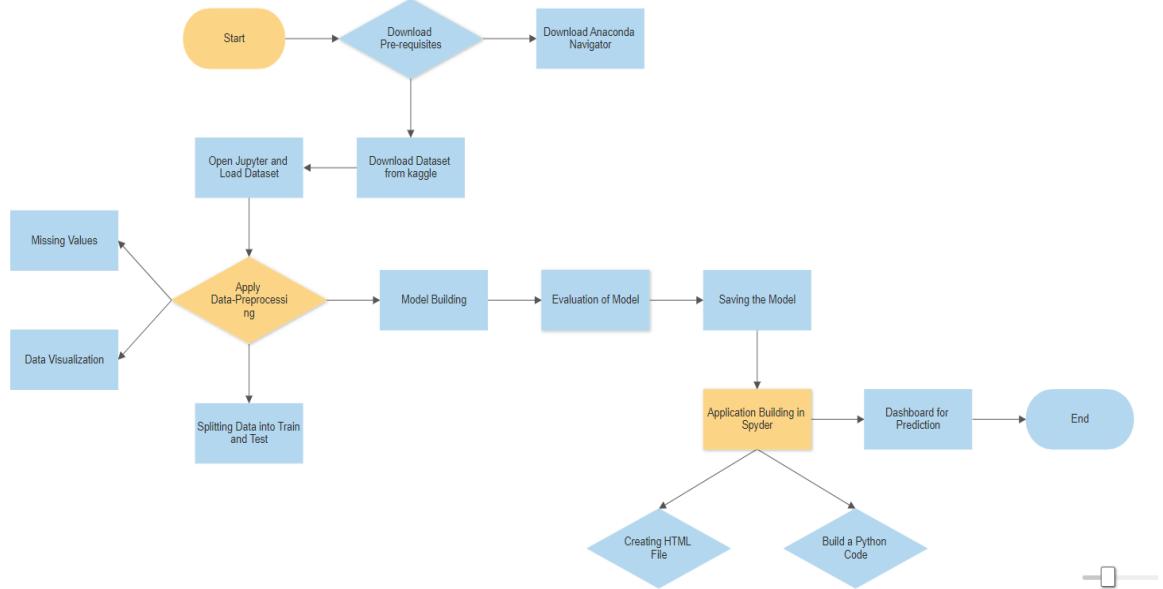
Category: Machine Learning

Skills Required:

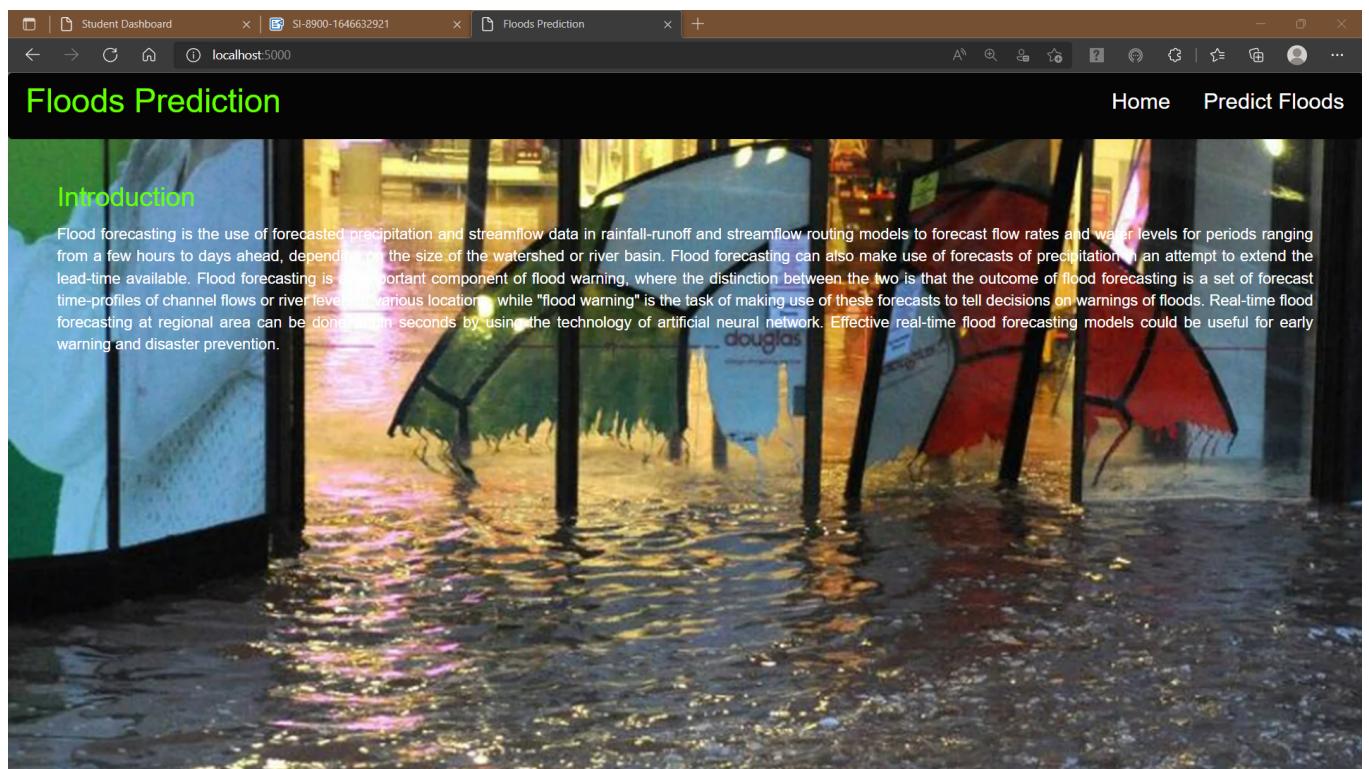
Python,Python Web Frame Works,Python For Data Analysis,Exploratory Data Analysis,Data Preprocessing Techniques,Machine Learning

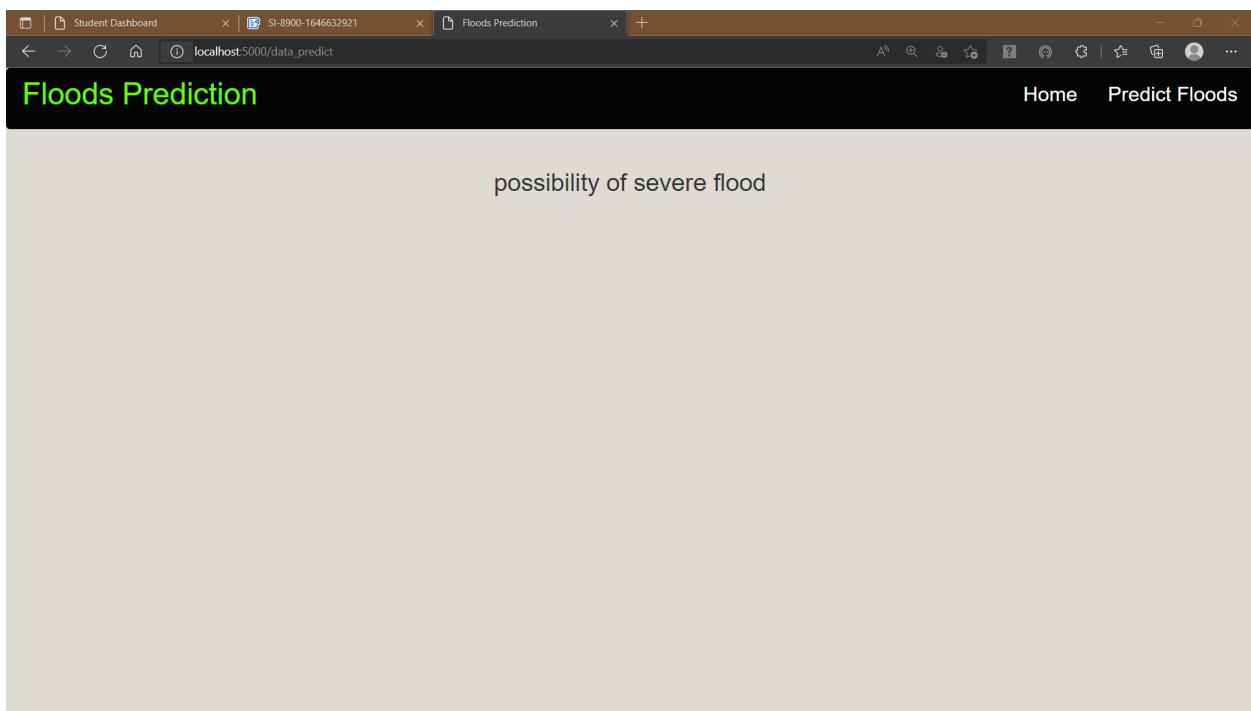
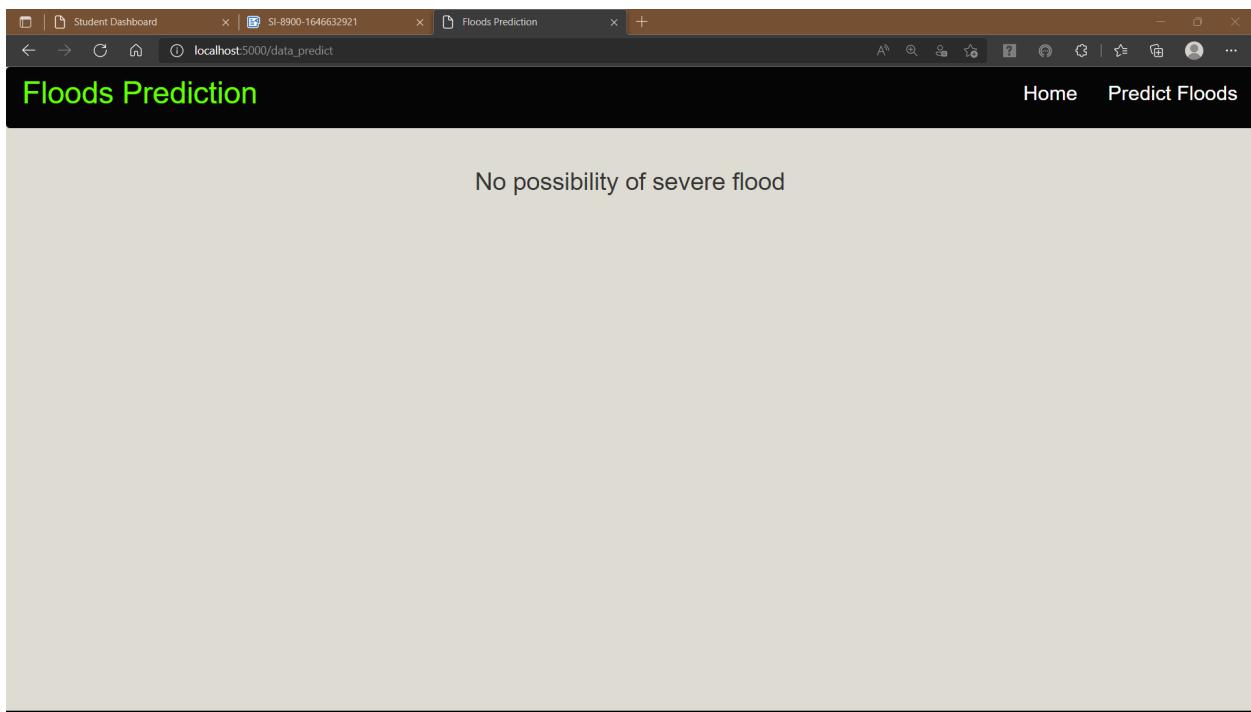
While working on the solution, I have seen the dataset of the floods and its quite frequent to see the whether forecast change drastically and cloud cover increases , rainfalls in the months April to June and lastly floods occur. There is interconnection of all the factors which leads to floods.

FLOW CHART



RESULT





ADVANTAGES AND DISADVANTAGES

ADVANTAGES :

- >The main purpose of flood warning is to save life by allowing people, support and emergency services time to prepare for flooding.
- >The model which I built is 96 % accurate so one can rely on the prediction done by the model.
- >With the help of this model , you can determine and manage environmental and water resource systems.
- >Knowledge about the characteristics of a river's drainage basin, such as soil-moisture conditions, ground temperature, snowpack, topography, vegetation cover, and impermeable land area, which can help to predict how extensive and damaging a flood might become.

DISADVANTAGES :

- >One of the algorithm used in this model is Random Forest, where it can become a plot as it gets prediction accuracy on complex problems is usually inferior to Gradient - Boosted Trees.

APPLICATIONS

The applications in flood prediction can be classified according to flood resource variables, i.e., water level, river flood, soil moisture, rainfall-discharge, precipitation, river inflow, peak flow, river flow, rainfall-runoff, flash flood, rainfall, streamflow, seasonal stream flow.

CONCLUSION

The summary of the project is that even though Floods are inevitable, they can be prevented with accurate measures like predicting the outcomes, changes to be taken to be on the safe side like building dams, reservoirs etc..and especially the technology has taken huge turn so, it will be matter of time before we gain control on what we are challenged.

FUTURE SCOPE

There can be future changes can be made to my model by adding the data like water level, soil moisture, average temperature of the area, rainfall - discharge , precipitation , drainage systems in the area etc.... in the dataset makes even more accurate and since the data will be enhanced, the analysis becomes the key part in the prediction.

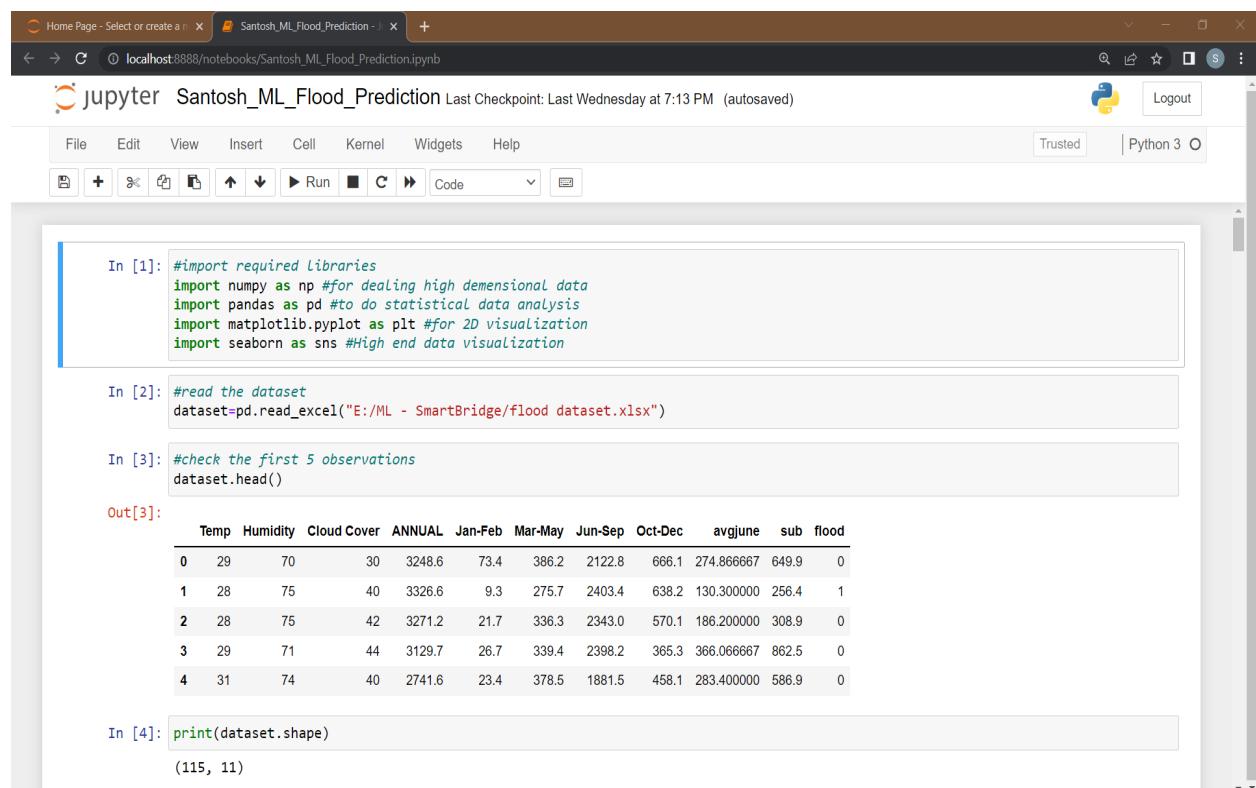
BIBILOGRAPHY

There are some references which are to be considered while making the project.

URL - 1 : [Flood prediction - Oxford Reference](#)

URL - 2 : [Flood Forecasting - an overview | ScienceDirect Topics](#)

APPENDIX - SOURCE CODE



The screenshot shows a Jupyter Notebook interface with the title "Santosh_ML_Flood_Prediction". The notebook has four cells:

- In [1]:

```
#import required Libraries
import numpy as np #for dealing high demensional data
import pandas as pd #to do statistical data analysis
import matplotlib.pyplot as plt #for 2D visualization
import seaborn as sns #High end data visualization
```
- In [2]:

```
#read the dataset
dataset=pd.read_excel("E:/ML - SmartBridge/flood dataset.xlsx")
```
- In [3]:

```
#check the first 5 observations
dataset.head()
```
- Out[3]:

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec	avgjune	sub	flood
0	29	70	30	3248.6	73.4	386.2	2122.8	666.1	274.866667	649.9	0
1	28	75	40	3326.6	9.3	275.7	2403.4	638.2	130.300000	256.4	1
2	28	75	42	3271.2	21.7	336.3	2343.0	570.1	186.200000	308.9	0
3	29	71	44	3129.7	26.7	339.4	2398.2	365.3	366.066667	862.5	0
4	31	74	40	2741.6	23.4	378.5	1881.5	458.1	283.400000	586.9	0
- In [4]:

```
print(dataset.shape)
```

The output for In [4] is (115, 11).

jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved)

In [5]: `print(dataset.info())`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115 entries, 0 to 114
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Temp        115 non-null    int64  
 1   Humidity    115 non-null    int64  
 2   Cloud Cover 115 non-null    int64  
 3   ANNUAL      115 non-null    float64 
 4   Jan-Feb     115 non-null    float64 
 5   Mar-May     115 non-null    float64 
 6   Jun-Sep     115 non-null    float64 
 7   Oct-Dec     115 non-null    float64 
 8   avgjune    115 non-null    float64 
 9   sub         115 non-null    float64 
 10  flood       115 non-null    int64  
dtypes: float64(7), int64(4)
memory usage: 10.0 KB
None
```

In [6]: `dataset.columns`

```
Index(['Temp', 'Humidity', 'Cloud Cover', 'ANNUAL', 'Jan-Feb', 'Mar-May',
       'Jun-Sep', 'Oct-Dec', 'avgjune', 'sub', 'flood'],
      dtype='object')
```

In [7]: `dataset.describe().T`

	count	mean	std	min	25%	50%	75%	max
Temp	115.0	29.600000	1.122341	28.0	29.000000	30.000000	31.000000	31.000000
Humidity	115.0	73.852174	2.947623	70.0	71.000000	74.000000	76.000000	79.000000
Cloud Cover	115.0	36.286957	4.330158	30.0	32.500000	36.000000	40.000000	44.000000
ANNUAL	115.0	2925.487826	422.112193	2068.8	2627.900000	2937.500000	3164.100000	4257.800000
Jan-Feb	115.0	27.739130	22.361032	0.3	10.250000	20.500000	41.600000	98.100000
Mar-May	115.0	377.253913	151.091850	89.9	276.750000	342.000000	442.300000	915.200000
Jun-Sep	115.0	2022.840870	386.254397	1104.3	1768.850000	1948.700000	2242.900000	3451.300000
Oct-Dec	115.0	497.636522	129.860643	166.6	407.450000	501.500000	584.550000	823.300000
avgjune	115.0	218.100870	62.547597	65.6	179.666667	211.033333	263.833333	366.066667
sub	115.0	439.801739	210.438813	34.2	285.000000	430.600000	577.650000	982.700000
flood	115.0	0.139130	0.347597	0.0	0.000000	0.000000	1.000000	

In [8]: `#checking null values
dataset.isnull().any()`

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb
Out[8]:	False	False	False	False	False

Home Page - Select or create a new notebook × Santosh_ML_Flood_Prediction - x +

jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved)

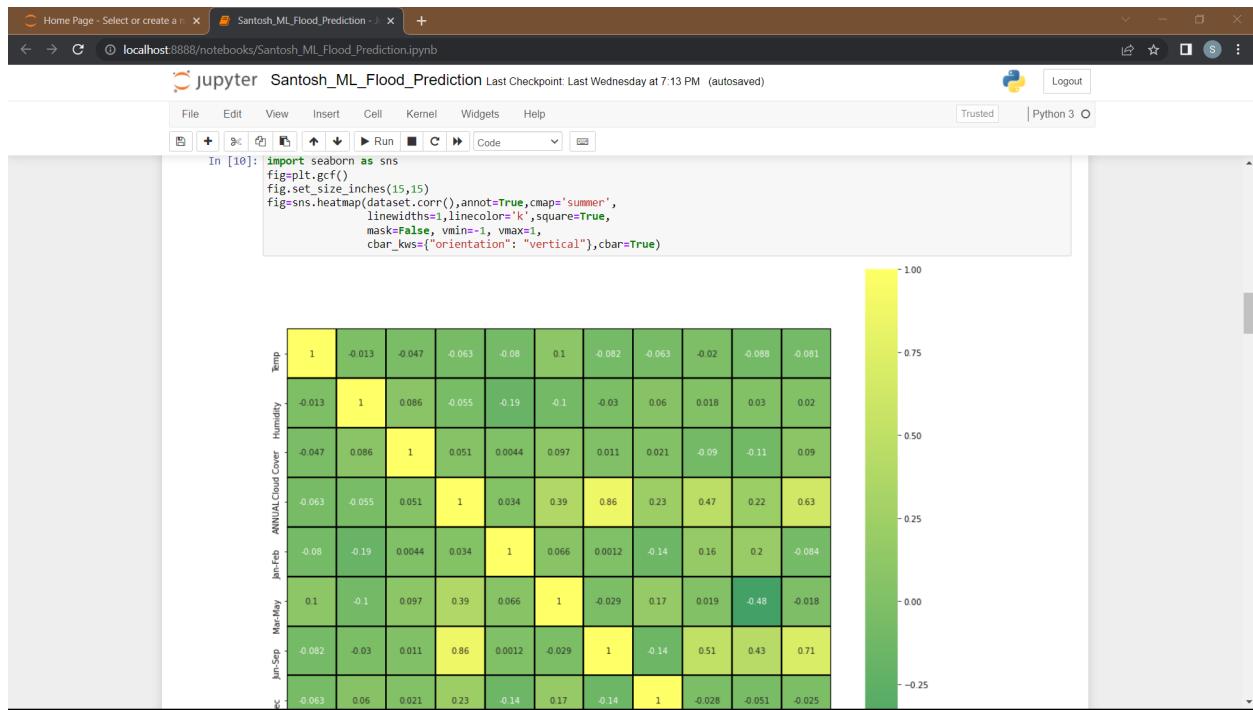
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [9]: #Correlation
dataset.corr()

Out[9]:

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec	avgjune	sub	flood
Temp	1.000000	-0.012727	-0.046568	-0.063014	-0.080076	0.099519	-0.081965	-0.063034	-0.019751	-0.088331	-0.080946
Humidity	-0.012727	1.000000	0.085824	-0.054767	-0.185965	-0.101232	-0.029583	0.059739	0.017656	0.029981	0.020250
Cloud Cover	-0.046568	0.085824	1.000000	0.051166	0.004376	0.096645	0.010833	0.020966	-0.089843	-0.106455	0.089801
ANNUAL	-0.063014	-0.054767	0.051166	1.000000	0.033639	0.387790	0.861190	0.232069	0.474644	0.220009	0.626874
Jan-Feb	-0.080076	-0.185965	0.004376	0.033639	1.000000	0.066479	0.001178	-0.143670	0.164691	0.201266	-0.084446
Mar-May	0.099519	-0.101232	0.096645	0.387790	0.066479	1.000000	-0.029007	0.171805	0.019183	-0.475750	-0.017598
Jun-Sep	-0.081965	-0.029583	0.010833	0.861190	0.001178	-0.029007	1.000000	-0.141467	0.511113	0.431997	0.705202
Oct-Dec	-0.063034	0.059739	0.020966	0.232069	-0.143670	0.171805	-0.141467	1.000000	-0.028055	-0.050862	-0.024852
avgjune	-0.019751	0.017656	-0.089843	0.474644	0.164691	0.019183	0.511113	-0.028055	1.000000	0.780445	0.379778
sub	-0.088331	0.029981	-0.106455	0.220009	0.201266	-0.475750	0.431997	-0.050862	0.780445	1.000000	0.349828
flood	-0.080946	0.020250	0.089801	0.626874	-0.084446	-0.017598	0.705202	-0.024852	0.379778	0.349828	1.000000

In [10]: import seaborn as sns
fig=plt.gcf()
fig.set_size_inches(15,15)
fig=sns.heatmap(dataset.corr(),annot=True,cmap='summer',
 linewidths=1,linecolor='k',square=True,
 mask=False, vmin=-1, vmax=1,
 cbar_kws={"orientation": "vertical"}, cbar=True)



jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved)

In [11]: `dataset.drop(["Oct-Dec"],axis=1,inplace=True)`

In [12]: `dataset.head()`

Out[12]:

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	avgjune	sub	flood
0	29	70	30	3248.6	73.4	386.2	2122.8	274.866667	649.9	0
1	28	75	40	3326.6	9.3	275.7	2403.4	130.300000	256.4	1
2	28	75	42	3271.2	21.7	336.3	2343.0	186.200000	308.9	0
3	29	71	44	3129.7	26.7	339.4	2398.2	366.066667	862.5	0
4	31	74	40	2741.6	23.4	378.5	1881.5	283.400000	586.9	0

In [13]: `dataset['flood'].value_counts()`

Out[13]:

0	99
1	16
Name: flood, dtype: int64	

In [14]: `#independent features
x=dataset.iloc[:,2:7].values`

In [15]: `#dependent feature
y=dataset.iloc[:,9:1].values`

In [16]: `x.shape`

jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved)

In [17]: `y.shape`

Out[17]:

(115, 1)

In [18]: `#split the data into train and test set from our x and y
#import train_test_split function
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=10)`

In [19]: `#checking the shape of our 4 variables
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)`

(86, 5)
(29, 5)
(86, 1)
(29, 1)

In [20]: `#import StandardScaler
from sklearn.preprocessing import StandardScaler
#create object to StandardScaler class
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)`

In [21]: `#import dump class from joblib`

jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved)

In [22]:

```
#pip install xgboost

import xgboost
import xgboost as xgb
#hyper parameter tuning to xgboost
xg_cla = xgb.XGBClassifier(objective ='reg:linear',learning_rate = 0.1,
                           max_depth = 5, n_estimators = 10)
```

In [23]:

```
#fit the model
xg_cla.fit(x_train,y_train)

D:\Anaconda\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
D:\Anaconda\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)

[17:42:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/objective/regression_obj.cu:188: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[23]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=5, min_child_weight=1, missing=np.nan,
              monotone_constraints='()', n_estimators=10, n_jobs=8,
              num_parallel_tree=1, objective='reg:linear', predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
```

jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved)

In [24]:

```
#predictions with unseen data by model
y_pred_xgb = xg_cla.predict(x_test)
y_pred_xgb
```

Out[24]:

```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0], dtype=int64)
```

In [25]:

```
#checking the accuracy score
from sklearn.metrics import accuracy_score,confusion_matrix
acc=accuracy_score(y_test,y_pred_xgb)
acc
```

Out[25]:

```
0.9655172413793104
```

In [26]:

```
#summary of predictions
cm2=confusion_matrix(y_test,y_pred_xgb)
cm2
```

Out[26]:

```
array([[25,  1],
       [ 0,  3]], dtype=int64)
```

In [27]:

```
y_pred_xgb.shape
```

Out[27]:

```
(29,)
```

In [28]:

```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
print("RMSE: %f" % (rmse))
```

Home Page · Select or create a ... × Santosh_ML_Flood_Prediction - × +

localhost:8888/notebooks/Santosh_ML_Flood_Prediction.ipynb

jupyter Santosh_ML_Flood_Prediction Last Checkpoint: Last Wednesday at 7:13 PM (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [29]: `dataset.head()`

Out[29]:

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	avgjune	sub	flood
0	29	70	30	3248.6	73.4	386.2	2122.8	274.866667	649.9	0
1	28	75	40	3326.6	9.3	275.7	2403.4	130.300000	256.4	1
2	28	75	42	3271.2	21.7	336.3	2343.0	186.200000	308.9	0
3	29	71	44	3129.7	26.7	339.4	2398.2	366.066667	862.5	0
4	31	74	40	2741.6	23.4	378.5	1881.5	283.400000	586.9	0

In [30]: `rand_pred = xg_cla.predict(sc.transform([[30,3248.6,73.4,386.2,2122.8]]))`
`rand_pred`

Out[30]: array([0], dtype=int64)

In [31]: `rand_pred1 = xg_cla.predict(sc.transform([[40,3326.6,9.3,275.7,2403.4]]))`
`rand_pred1`

Out[31]: array([1], dtype=int64)

In [32]: `data_dmatrix = xgb.DMatrix(data=x,label=y)`

In [33]: `params = {"objective":"reg:linear","colsample_bytree": 0.3,'learning_rate': 0.1,`
`'max_depth': 5, 'alpha': 10}`
`cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,`

The screenshot shows a Jupyter Notebook interface with the title "Santosh_ML_Flood_Prediction". The notebook has a Python 3 kernel. The code in cell 40 is:

```
In [40]: acc=accuracy_score(y_test,y_predict)
acc
```

The output is:

```
Out[40]: 0.9655172413793104
```

Cell 41 contains:

```
In [41]: acc=accuracy_score(y_test,y_pred_xgb)
acc
```

The output is:

```
Out[41]: 0.9655172413793104
```

Cell 42 contains:

```
In [42]: cm1=confusion_matrix(y_test,y_predict)
cm1
```

The output is:

```
Out[42]: array([[25,  1],
   [ 0,  3]], dtype=int64)
```

Cell 43 contains:

```
In [43]: cm2=confusion_matrix(y_test,y_pred_xgb)
cm2
```

The output is:

```
Out[43]: array([[25,  1],
   [ 0,  3]], dtype=int64)
```

Cell 44 contains:

```
In [44]: from joblib import load
dump(xg_cla,'floods.save')
```

The output is:

```
Out[44]: ['floods.save']
```

The screenshot shows the Spyder IDE interface. On the left, there is a code editor with the file "app.py" open, containing the following code:

```
1  from flask import Flask, render_template, request
2  # used to run/serve our application
3  # render_template is used for rendering the html pages
4  # import load from joblib to load the saved model file
5
6  from joblib import load
7
8  app=Flask(__name__) # our flask app
9  #load model file
10 model = load('C:/Users/ACER/floods.save')
11 sc = load('C:/Users/ACER/transform.save')
12
13
14 @app.route('/') # rendering the html template
15 def home():
16     return render_template('home.html')
17 @app.route('/predict') # rendering the html template
18 def index():
19     return render_template("index.html")
20
21
22 @app.route('/data_predict', methods=['POST']) # route for our prediction
23 def predict():
24     temp = request.form['temp']
25     Hum = request.form['Hum']
26     db = request.form['db']
27     ap = request.form['ap']
28     aa1 = request.form['aa1']
29
30     data = [[float(temp),float(Hum),float(db),float(ap),float(aa1)]]
31     prediction = model.predict(sc.transform(data))
32     output=prediction[0]
33     if (output==0):
34         return render_template('noChance.html', prediction='No possibility of severe flood')
35     else:
36         return render_template('chance.html', prediction='possibility of severe flood')
37
38
39 if __name__ == '__main__':
40     app.run(debug=False)
```

On the right, there is an IPython console window with the following output:

```
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.22.0 -- An enhanced Interactive Python.

In [1]: runfile('E:/ML - SmartBridge/Flask/Flask/app.py', wdir='E:/ML - SmartBridge/Flask/Flask')
* 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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```