Early Warning System

August 1, 2024

0.1 Description:

The project aims to enhance early warning systems for natural disasters using advanced data analytics by integrating diverse datasets, developing predictive models, and creating real-time alert systems. This approach will improve prediction accuracy and response times, providing actionable insights for better disaster preparedness. The initiative supports SDG 13 and SDG 11 by increasing community resilience and reducing the impact of natural disasters.

Dataset url : https://www.kaggle.com/datasets/naiyakhalid/flood-prediction-dataset

0.1.1 Name of the columns with their meanings

- **MonsoonIntensity**: Measures the intensity of monsoon rains.

- **TopographyDrainage**: Evaluates the area's topography and natural drainage capacity.
- **RiverManagement**: Assesses the effectiveness of river management practices.
- **Deforestation**: Indicates the extent of deforestation in the region.
- **Urbanization**: Reflects the level of urbanization and its impact.
- **ClimateChange**: Examines the effects of climate change on the region.
- **DamsQuality**: Evaluates the condition and quality of dams.
- **Siltation**: Measures the accumulation of silt in water bodies.
- **AgriculturalPractices**: Assesses the impact of agricultural practices on the environment.
- **Encroachments**: Indicates the level of encroachment on natural waterways.
- **IneffectiveDisasterPreparedness**: Evaluates the effectiveness of disaster preparedness measures.
- **DrainageSystems**: Assesses the quality and efficiency of drainage systems.
- **CoastalVulnerability**: Measures the vulnerability of coastal areas to flooding.
- **Landslides**: Indicates the susceptibility to landslides.
- **Watersheds**: Evaluates the health and management of watersheds.
- **DeterioratingInfrastructure**: Assesses the condition of infrastructure and its role inflood risk.
- **PopulationScore**: Reflects population density and its impact on flood risk.
- **WetlandLoss**: Measures the extent of wetland loss.
- **InadequatePlanning**: Evaluates the adequacy of planning and zoning regulations.
- **PoliticalFactors**: Assesses political factors that may influence flood risk and management.
- **FloodProbability**: Indicates the overall probability of flooding based on various factors.

0.1.2 Step 1 : Importing libraries like Numpy, Pandas, Matplotlib, Seaborn and scikit learn(Sklearn)

[1]:

```python
# Numpy
import numpy as np

# Pandas
import pandas as pd

# Matplotlib
import matplotlib.pyplot as plt


# Seaborn
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')
```

0.1.3 Step2 : Load the Dataset

[2]:
```python
# Load flood dataset
df = pd.read_csv("C:\\Users\\varsh\\OneDrive\\Documents\\Flood Dataset.csv")
```

0.1.4 Step 3 : Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a step in the Data Analysis Process, where a number of techniques are used to better understand the dataset being used.

In this step, we will perform the below operations:

3.1) Understanding Your Variables 3.1.1) Head of the dataset 3.1.2) The shape of the dataset 3.1.3) List types of all columns 3.1.4) Info of the dataset 3.1.5) Summary of the dataset

3.2) Data Cleaning 3.2.1) Check the DUPLICATES 3.2.2) Check the NULL values

3.1.1) Head of the Dataset This head(n) function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it. By default it will show 5 rows.

[3]:
```python
# Display first fifteen records of data
df.head(15)
```

[3]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Encroachment
0	3	8	6	6	4	4	6	2	3	[illegible]
1	8	4	5	7	7	9	1	5	5	[illegible]
2	3	10	4	1	7	5	4	7	4	[illegible]
3	4	4	2	7	3	4	1	4	6	[illegible]
4	3	7	5	2	5	8	5	2	7	[illegible]
5	6	6	6	4	6	4	3	1	3	[illegible]
6	6	7	4	5	5	5	4	8	8	[illegible]
7	7	3	5	5	6	6	6	7	6	[illegible]
8	6	3	5	4	5	11	3	2	9	[illegible]
9	4	3	5	6	2	3	7	7	10	[illegible]
10	5	1	7	4	5	7	4	3	0	[illegible]
11	6	9	1	4	3	7	5	8	4	[illegible]
12	4	9	4	1	5	4	2	8	4	[illegible]
13	6	3	7	9	7	4	11	7	8	[illegible]
14	8	1	9	4	6	7	6	3	4	[illegible]

geSystems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning	PoliticalFactors	FloodProbability
10	7	4	2	3	4	3	2	6	0.450
9	2	6	2	1	1	9	1	3	0.475
7	4	4	8	6	1	8	3	6	0.515
4	2	6	6	8	8	6	6	10	0.520
7	6	5	3	3	4	4	3	4	0.475
10	5	9	5	5	7	3	3	2	0.470
8	4	5	4	7	7	5	4	8	0.570
4	6	9	7	10	6	5	4	5	0.585
2	8	7	5	4	9	6	5	7	0.580
7	6	5	6	7	5	7	4	8	0.555
6	4	8	5	5	7	4	2	6	0.455
8	4	3	5	6	4	6	14	3	0.555
5	7	7	3	4	2	3	6	3	0.450
4	5	4	2	3	4	6	7	4	0.525
4	3	2	6	4	5	4	2	8	0.480

[4]:
```
# Display last five records of the data
df.tail(15)
```

[4]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Encroachm
49985	6	4	7	2	7	8	5	5	3	
49986	4	7	5	6	6	5	4	5	10	
49987	11	3	6	9	4	5	3	6	3	
49988	2	7	5	6	5	4	7	6	2	
49989	5	9	4	3	4	3	4	4	2	
49990	5	8	3	7	3	5	2	8	12	
49991	3	10	1	3	2	2	6	5	6	
49992	5	8	7	6	6	6	4	2	7	
49993	4	4	5	5	13	7	2	7	10	
49994	6	5	3	5	9	4	6	6	3	
49995	3	7	4	7	5	9	4	6	10	
49996	3	10	3	8	3	3	4	4	3	
49997	4	4	5	7	2	1	4	5	6	
49998	4	5	4	4	6	3	10	2	6	
49999	4	5	6	3	5	6	5	4	9	

geSystems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning	PoliticalFactors	FloodProbability
6	9	13	7	5	5	7	2	3	0.580
7	6	6	3	6	2	7	2	7	0.525
2	5	1	7	3	0	3	2	3	0.420
12	4	4	5	4	4	7	5	4	0.525
7	2	4	4	3	5	6	10	5	0.495
2	6	5	4	5	5	6	4	4	0.580
6	3	3	3	4	2	7	5	4	0.435
5	1	8	3	6	4	5	3	7	0.520
6	9	0	7	4	5	6	3	0	0.525
6	3	6	8	2	9	7	5	4	0.535
7	3	8	8	6	1	5	4	2	0.535
8	6	3	6	4	4	2	4	5	0.510
4	6	4	1	5	1	6	4	3	0.430
6	3	4	7	6	2	4	0	11	0.515
2	4	4	5	6	7	8	10	7	0.580

[5]:
```
# Display randomly any number of records of data
df.sample()
```

[5]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Encroachm
49993	4	4	5	5	13	7	2	7	10	

geSystems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning	PoliticalFactors	FloodProbability
6	9	0	7	4	5	6	3	0	0.525

3.1.2) The shape of the dataset This shape() function gives us the number of rows and columns of the dataset.

[6]:
```
#Number of rows and columns
df.shape
```

[6]: (50000, 21)

[7]:
```
#List the types of all columns.
df.dtypes
```

[7]:
```
MonsoonIntensity                    int64
TopographyDrainage                  int64
RiverManagement                     int64
Deforestation                       int64
Urbanization                        int64
ClimateChange                       int64
DamsQuality                         int64
Siltation                           int64
AgriculturalPractices               int64
Encroachments                       int64
IneffectiveDisasterPreparedness     int64
DrainageSystems                     int64
CoastalVulnerability                int64
Landslides                          int64
Watersheds                          int64
DeterioratingInfrastructure         int64
PopulationScore                     int64
WetlandLoss                         int64
InadequatePlanning                  int64
PoliticalFactors                    int64
FloodProbability                  float64
dtype: object
```

3.1.3) Info of the dataset info() is used to check the Information about the data and the datatypes of each respective attribute.

[8]:
```
#finding out if the dataset contains any null value
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   MonsoonIntensity                 50000 non-null  int64  
 1   TopographyDrainage               50000 non-null  int64  
 2   RiverManagement                  50000 non-null  int64  
 3   Deforestation                    50000 non-null  int64  
 4   Urbanization                     50000 non-null  int64  
 5   ClimateChange                    50000 non-null  int64  
 6   DamsQuality                      50000 non-null  int64  
 7   Siltation                        50000 non-null  int64  
 8   AgriculturalPractices            50000 non-null  int64  
 9   Encroachments                    50000 non-null  int64  
 10  IneffectiveDisasterPreparedness  50000 non-null  int64  
 11  DrainageSystems                  50000 non-null  int64  
 12  CoastalVulnerability             50000 non-null  int64  
 13  Landslides                       50000 non-null  int64  
 14  Watersheds                       50000 non-null  int64  
 15  DeterioratingInfrastructure      50000 non-null  int64  
 16  PopulationScore                  50000 non-null  int64  
 17  WetlandLoss                      50000 non-null  int64  
 18  InadequatePlanning               50000 non-null  int64  
 19  PoliticalFactors                 50000 non-null  int64  
 20  FloodProbability                 50000 non-null  float64
dtypes: float64(1), int64(20)
memory usage: 8.0 MB
```

3.1.4) Summary of the dataset The describe() method is used for calculating some statistical data like percentile, mean and std of the numerical values of the Series or DataFrame. It analyzes both numeric and object series and also the DataFrame column sets of mixed data types.

[9]:
```
# Statistical summary
df.describe()
```

[9]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Enc
count	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	5
mean	4.991480	4.984100	5.01594	5.008480	4.989060	4.988340	5.01536	4.988600	5.006120	
std	2.236834	2.246488	2.23131	2.222743	2.243159	2.226761	2.24500	2.232642	2.234588	
min	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	
25%	3.000000	3.000000	3.00000	3.000000	3.000000	3.000000	3.00000	3.000000	3.000000	
50%	5.000000	5.000000	5.00000	5.000000	5.000000	5.000000	5.00000	5.000000	5.000000	
75%	6.000000	6.000000	6.00000	6.000000	6.000000	6.000000	6.00000	6.000000	6.000000	
max	16.000000	18.000000	16.00000	17.000000	17.000000	17.000000	16.00000	16.000000	16.000000	

tems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning	PoliticalFactors	FloodProbability
0000	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	50000.000000
6060	4.999920	4.984220	4.97982	4.988200	4.984980	5.00512	4.994360	4.990520	0.499660
8107	2.247101	2.227741	2.23219	2.231134	2.238279	2.23176	2.230011	2.246075	0.050034
0000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	0.000000	0.285000
0000	3.000000	3.000000	3.00000	3.000000	3.000000	3.00000	3.000000	3.000000	0.465000
0000	5.000000	5.000000	5.00000	5.000000	5.000000	5.00000	5.000000	5.000000	0.500000
0000	6.000000	6.000000	6.00000	6.000000	6.000000	6.00000	6.000000	6.000000	0.535000
0000	17.000000	16.000000	16.00000	17.000000	19.000000	22.00000	16.000000	16.000000	0.725000

Observation: In the above table, the min value of columns 'Glucose','Blood Pressure', 'SkinThickness', 'Insulin', 'BMI is zero (0). It is clear that this values can't be zero.So we im- pute the mean values of these respective columns instead of zero.

0.1.5 3.2) Data Cleaning

3.2.1) Drop the Duplicates check is there any duplicate rows are exist or not, if exist then we should remove from the dataframe.

[10]
```
# check the shape before drop the duplicates
df.shape
```

[10]: (50000, 21)

[11]:
```
df=df.drop_duplicates()
```

[12]:
```
# check the shape after drop the duplicates
df.shape
```

[12]: (50000, 21)

Before droping and after droping the duplicates the data set has same shape so no duplicates are there in the dataset.

0.1.6 3.2.2) Check the NULL Values

Using isnull.sum() function we can see the null values present in the every column in the dataset.

[13]:

```
# count of null, values

# checking the missing values in any column
# Display number of null values in every column in dataset
df.isnull().sum()
```

```
MonsoonIntensity                   0
TopographyDrainage                 0
RiverManagement                    0
Deforestation                      0
Urbanization                       0
ClimateChange                      0
DamsQuality                        0
Siltation                          0
AgriculturalPractices              0
Encroachments                      0
IneffectiveDisasterPreparedness    0
DrainageSystems                    0
CoastalVulnerability               0
Landslides                         0
Watersheds                         0
DeterioratingInfrastructure        0
PopulationScore                    0
WetlandLoss                        0
InadequatePlanning                 0
PoliticalFactors                   0
FloodProbability                   0
dtype: int64
```

There is no NULL values in the given dataset.

[14]:

```
df.columns
```

```
Index(['MonsoonIntensity', 'TopographyDrainage', 'RiverManagement',
       'Deforestation', 'Urbanization', 'ClimateChange', 'DamsQuality',
       'Siltation', 'AgriculturalPractices', 'Encroachments',
       'IneffectiveDisasterPreparedness', 'DrainageSystems',
       'CoastalVulnerability', 'Landslides', 'Watersheds',
       'DeterioratingInfrastructure', 'PopulationScore', 'WetlandLoss',
       'InadequatePlanning', 'PoliticalFactors', 'FloodProbability'],
      dtype='object')
```

[15]:

```
print("No. of zero values in TopographyDrainage",df[df[" TopographyDrainage
"]==0].shape[0])
```

No. of zero values in TopographyDrainage 5

[16]:

```
print("No. of zero values in RiverManagment",df[df[" RiverManagment"]==0].
↪shape[0])
```

No. of zero values in RiverManagment 335

[17]:

```
print("No. of zero values in DrainageSystem" ,df[df["DrainageSystems"]==0].
↪shape[0])
```

No. of zero values in DrainageSystems 335

[18]:

```
print("No. of zero values in Watersheds" ,df[df["Watersheds"]==0].shape[0])
```

No. of zero values Watersheds 313

[19]:

```
print("No. of zero values in PoliticalFactors",df[df["PoliticalFactors"]==0].shape[0])
```

No. of zero valuesPoliticalFactors363

[20]: Replace no.of zero values with mean of that columns

```
df['TopographyDrainage'] = df['TopographyDrainage'].replace(0,df['TopographyDrainage'].mean())
print('No. of zero values in TopographyDrainage',df[df['TopographyDrainage']==0].shape[0])
```

```
No. of zero values in TopographyDrainage 0
```

[21]:

```python
# Replace no. of zero values  with mean of that columns
df['RiverManagement'] = df['RiverManagement'].replace(0,df['RiverManagement'].mean())
df['DrainageSystems'] = df['DrainageSystems'].replace(0,df['DrainageSystems'].mean())
df['Watersheds'] = df['Watersheds'].replace(0,df['Watersheds'].mean())
df['PoliticalFactors'] = df['PoliticalFactors'].replace(0,df['PoliticalFactors'].mean())


df.describe()
```

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Enc
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	5(
mean	4.991480	5.017294	5.049547	5.008480	4.989060	4.988340	5.01536	4.988600	5.006120	
std	2.236834	2.209108	2.192953	2.222743	2.243159	2.226761	2.24500	2.232642	2.234588	
min	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	
25%	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.00000	3.000000	3.000000	
50%	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000	5.000000	
75%	6.000000	6.000000	6.000000	6.000000	6.000000	6.000000	6.00000	6.000000	6.000000	
max	16.000000	18.000000	16.000000	17.000000	17.000000	17.000000	16.00000	16.000000	16.000000	

```python
# Replace no. of zero values  with mean of that columns
df['RiverManagement'] = df['RiverManagement'].replace(0,df['RiverManagement'].mean())
df['DrainageSystems'] = df['DrainageSystems'].replace(0,df['DrainageSystems'].mean())
df['Watersheds'] = df['Watersheds'].replace(0,df['Watersheds'].mean())
df['PoliticalFactors'] = df['PoliticalFactors'].replace(0,df['PoliticalFactors'].mean())


df.describe()
```

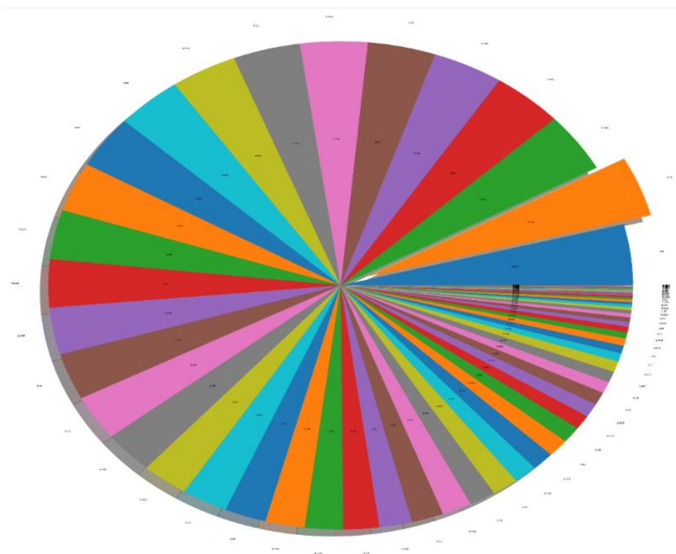
Encroachments	...	DrainageSystems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning
50000.000000	...	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.00000	50000.000000
5.006380	...	5.039601	4.999920	4.984220	5.010994	4.988200	4.984980	5.00512	4.994360
2.241633	...	2.200020	2.247101	2.227741	2.196920	2.231134	2.238279	2.23176	2.230011
0.000000	...	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.00000	0.000000
3.000000	...	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.00000	3.000000
5.000000	...	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000
6.000000	...	6.000000	6.000000	6.000000	6.000000	6.000000	6.000000	6.00000	6.000000
18.000000	...	17.000000	17.000000	16.000000	16.000000	17.000000	19.000000	22.00000	16.000000

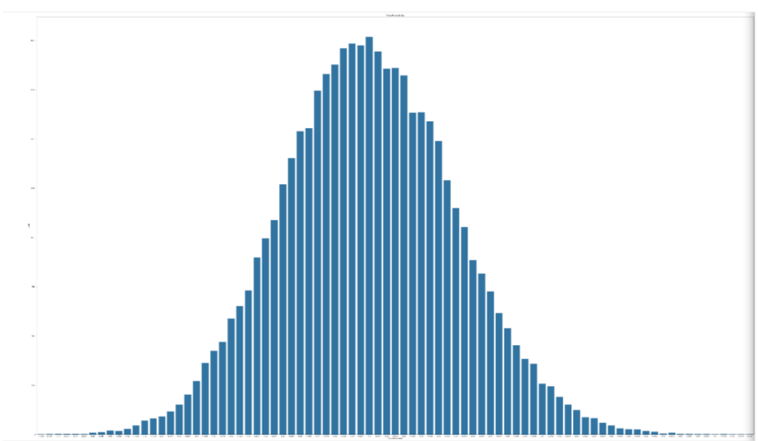
```python
# Replace no. of zero values  with mean of that columns
df['RiverManagement'] = df['RiverManagement'].replace(0,df['RiverManagement'].mean())
df['DrainageSystems'] = df['DrainageSystems'].replace(0,df['DrainageSystems'].mean())
df['Watersheds'] = df['Watersheds'].replace(0,df['Watersheds'].mean())
df['PoliticalFactors'] = df['PoliticalFactors'].replace(0,df['PoliticalFactors'].mean())


df.describe()
```

ms	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning	PoliticalFactors	FloodProbability
000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.00000	50000.000000	50000.000000	50000.000000
501	4.999920	4.984220	5.010994	4.988200	4.984980	5.00512	4.994360	5.026751	0.499660
020	2.247101	2.227741	2.196920	2.231134	2.238279	2.23176	2.230011	2.205158	0.050034
000	0.000000	0.000000	1.000000	0.000000	0.000000	0.00000	0.000000	1.000000	0.285000
000	3.000000	3.000000	3.000000	3.000000	3.000000	3.00000	3.000000	3.000000	0.465000
000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000	5.000000	0.500000
000	6.000000	6.000000	6.000000	6.000000	6.000000	6.00000	6.000000	6.000000	0.535000
000	17.000000	16.000000	16.000000	17.000000	19.000000	22.00000	16.000000	16.000000	0.725000

[22]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
# Ensure the column 'FloodProbability' exists
if 'FloodProbability' in df.columns:
    # Get the number of unique values in 'FloodProbability' column
    num_unique_values = df['FloodProbability'].nunique()
# outcome count plot
f,ax=plt.subplots(1,2,figsize=(10,5))
df['FloodProbability'].value_counts().plot.pie(explode=[0.1 if i == 1 else 0 for i in range(num_unique_values)], autopct='%1.1f%%',ax=ax[0], shadow=True)
ax[0].set_title('FloodProbability')
ax[0].set_ylabel('')
sns.countplot(x='FloodProbability',data=df,ax=ax[1])
ax[1].set_title('FloodProbability')
N,P=df['FloodProbability'].value_counts()
print('Negative (0) :',N)
print('Positive (1) :',M)

plt.grid()
plt.show()
```





[23]:

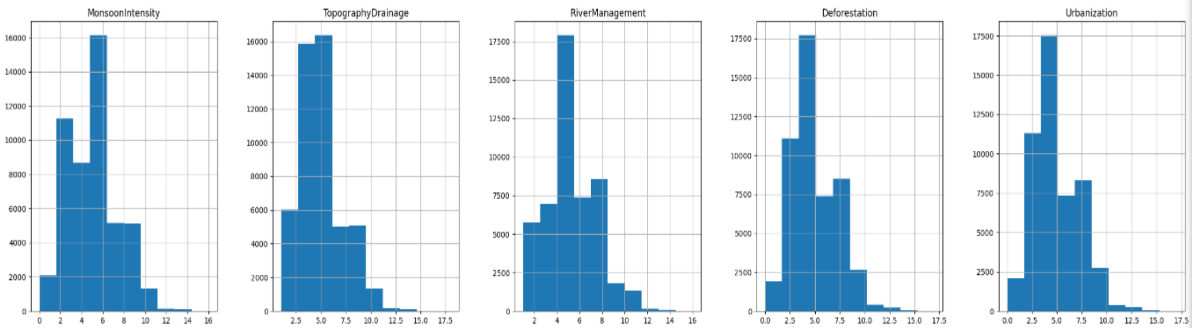
0.1.7 4.2) Histograms

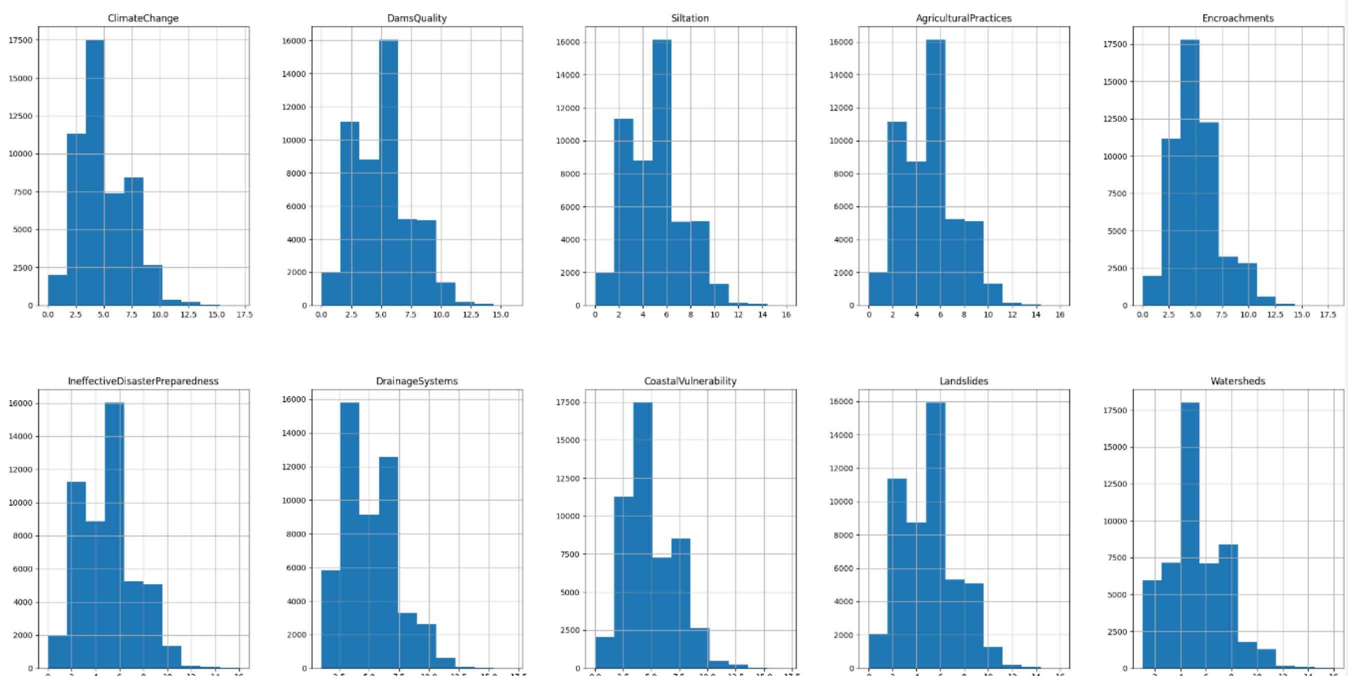
Histograms are one of the most common graphs used to display numeric data.

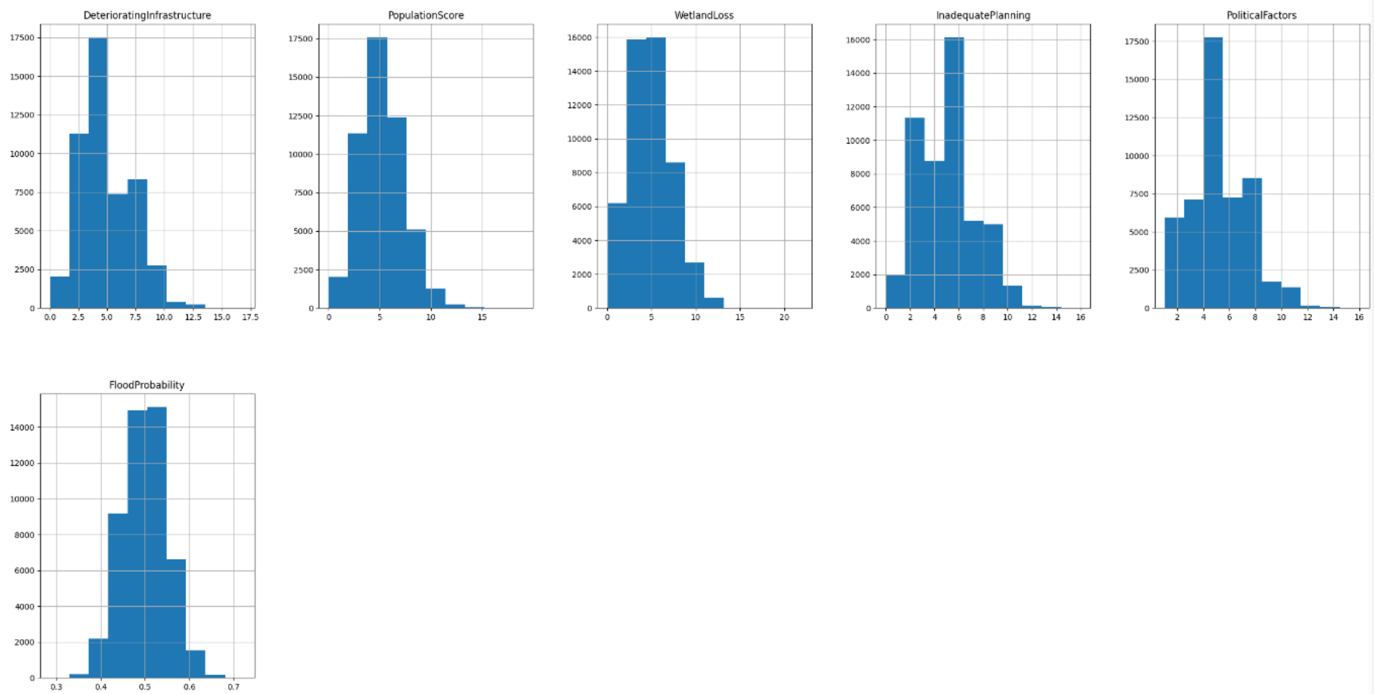
distribution of the data - Whether the data is normally distributed or if it's skewed (to the left or right)

```python
# Histogram of each feature
df.hist(bins = 10, figsize = (30,40))
plt.show()
```

```
No. of zero values in TopographyDrainage 0
```





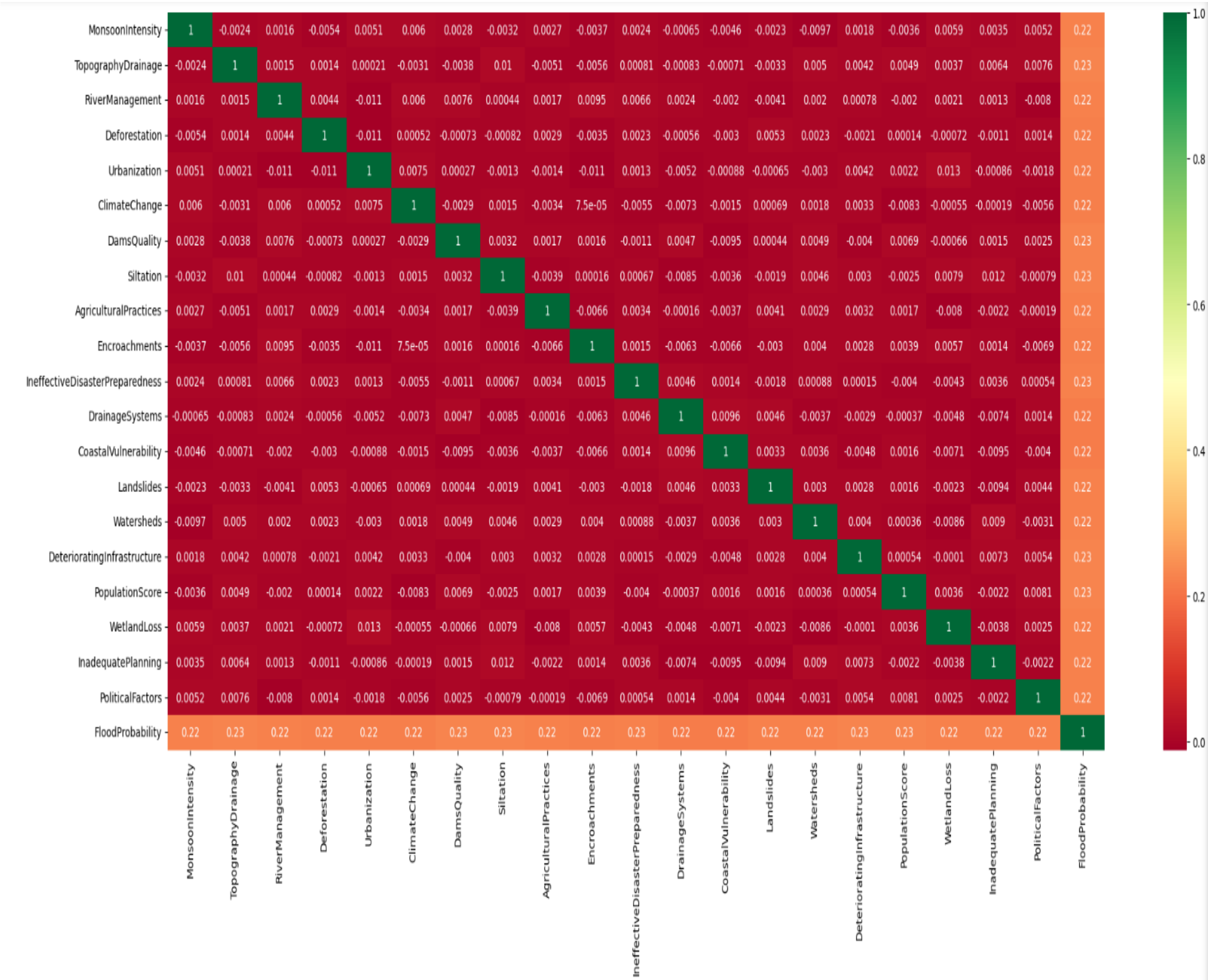


4.5) Analyzing relationships between variables

Correlation analysis in data science is a statistical technique used to mea- sure the strength and direction of the relationship between two or more variables in a dataset. It helps data scientists understand how changes in one variable are associated with changes in another. By calculating correlation coefficients, such as Pearson's correlation coefficient for contin- uous variables or rank-based correlations for non-linear or ordinal data, data scientists can identify patterns and dependencies in the data. This analysis is valuable for feature selection, identifying potential predictor variables, and gaining insights into the interactions between different aspectsof the dataset, facilitating better decision-making and predictive modeling.

[24]:

```
import seaborn as sns
#get correlations of each features in dataset
import matplotlib.pyplot as plt

cormat = df.corr()
top_corr_features = cormat.index
plt.figure(figsize= (25,10))
#plot heat map
g =sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")

# display the plot
plt.show()
```


MonsoonIntensity
TopographyDrainage
RiverManagement
Deforestation
Urbanization
ClimateChange
DamsQuality
Siltation
AgriculturalPractices
Encroachments
IneffectiveDisasterPreparedness
DrainageSystems
CoastalVulnerability
Landslides
Watersheds
DeterioratingInfrastructure
PopulationScore
WetlandLoss
InadequatePlanning
PoliticalFactors
FloodProbability
1.0
0.8
0.6
0.4
0.2
0.0

0.1 5.) Split the data frame into X & y

[25]:

```
target_name = 'FloodProbability'

# separate object for target features
y = df[target_name]

#separate object for input features
x = df.drop(target_name, axis = 1)

x.head()
```

```
No. of zero values in TopographyDrainage 0
```

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Encroachments
0	3	8.0	6.0	6	4	4	6	2	3	2
1	8	4.0	5.0	7	7	9	1	5	5	4
2	3	10.0	4.0	1	7	5	4	7	4	9
3	4	4.0	2.0	7	3	4	1	4	6	4
4	3	7.0	5.0	2	5	8	5	2	7	5

```python
target_name = 'FloodProbability'

# separate object for target features
y = df[target_name]

#separate object for input features
x = df.drop(target_name, axis = 1)

x.head()
```

```
No. of zero values in TopographyDrainage 0
```

IneffectiveDisasterPreparedness	DrainageSystems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePla
5	10.0	7	4	2.0	3	4	3	
6	9.0	2	6	2.0	1	1	9	
2	7.0	4	4	8.0	6	1	8	
9	4.0	2	6	6.0	8	8	6	
7	7.0	6	5	3.0	3	4	4	


```python
target_name = 'FloodProbability'

# separate object for target features
y = df[target_name]

#separate object for input features
x = df.drop(target_name, axis = 1)

x.head()
```

```
No. of zero values in TopographyDrainage 0
```


paredness	DrainageSystems	CoastalVulnerability	Landslides	Watersheds	DeterioratingInfrastructure	PopulationScore	WetlandLoss	InadequatePlanning	PoliticalFactors
5	10.0	7	4	2.0	3	4	3	2	6.0
6	9.0	2	6	2.0	1	1	9	1	3.0
2	7.0	4	4	8.0	6	1	8	3	6.0
9	4.0	2	6	6.0	8	8	6	6	10.0
7	7.0	6	5	3.0	3	4	4	3	4.0

[26]:

```
y.head()
```

```
No. of zero values in TopographyDrainage 0

0    0.450
1    0.475
2    0.515
3    0.520
4    0.475
Name: FloodProbability, dtype: float64
```

0.1 6) Apply Feature Scalling

```
!pip install scikit-learn

# Apply Standard scaler
# separate object for target features
Y = df[target_name]

#separate object for input features
X = df.drop(target_name, axis = 1)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
SSX = scaler.transform(X)
```

7) Train Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(SSX, Y, test_size=0.2,random_state=7)

X_train.shape,Y_train.shape
```

```
((40000, 20), (40000,))
```

[27]:

```
X_test.shape, y_test.shape
```

[27]: ((10000, 20), (10000,))

0.2 8) Build the Classification Algorithms

0.2.1 8.1) Logistic Regression

[28]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='liblinear', multi_class='ovr')
lr.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(multi_class='ovr', solver='liblinear')
```

0.2.2 8.2) KNeighborsClassifier(KNN)

[29]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

0.2.3 8.3) Naive-Bayes Classifier

[30]:

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
```

```
GaussianNB
GaussianNB()
```

0.2.4 8.4) Support Vector Machine

[31]:

```
from sklearn.svm import SVC
sv = SVC()
sv.fit(X_train, y_train)
```

```
SVC
SVC()
```

8.5) Decision Tree

[32]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

0.2.5 8.6) Random Forest

[33]:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

0.3 9) Making Prediction

0.3.1 9.1) Making Prediction on test by using Logistic Regression

[34]:

```
# display the shape of test data
X_test.shape
```

```
(10000, 21)
```

[35]:

```python
# making prediction on test dataset
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
# Assuming 'data' is your DataFrame
df['FloodRisk'] = (df['FloodProbability'] > 0.5).astype(int)
# Define features (X) and target (y)
X = df.drop(columns=['FloodProbability', 'FloodRisk'])
y = df['FloodRisk']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create and fit the Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
# Predict probabilities
y_pred_proba = model.predict_proba(X_test)[:, 1]
# Add predicted probabilities to the test set for inspection
X_test_with_pred = X_test.copy()
X_test_with_pred['Predicted_FloodProbability'] = y_pred_proba
# Print the first few rows of the test set with the predicted probabilities
print(X_test_with_pred.head())
# Make predictions using the model
lr_pred = model.predict(X_test)
# Display the shape of the predicted data
print(lr_pred.shape)
# Optionally, add the predictions to X_test_with_pred for inspection
X_test_with_pred['Predicted_FloodRisk'] = lr_pred
print(X_test_with_pred.head())
```

```
33553    0
9427     0
199      0
12447    0
39489    0
Name: FloodRisk, dtype: int32
       MonsoonIntensity  TopographyDrainage  RiverManagement  Deforestation  \
33553                 7                 3.0              4.0              4
9427                  4                 7.0              7.0              3
199                   2                 3.0              3.0              4
12447                 5                 4.0              5.0              7
39489                 3                 9.0              5.0              3

       Urbanization  ClimateChange  DamsQuality  Siltation  \
33553             5              6            2          5
9427              5              7            5          2
199               5              4            7          7
12447             8              2            5          5
39489             3              4            6          9

       AgriculturalPractices  Encroachments  ...  DrainageSystems  \
33553                      5              4  ...              6.0
9427                       7              6  ...              3.0
199                        2              3  ...              8.0
12447                      5              8  ...              6.0
39489                      0              5  ...              5.0

       CoastalVulnerability  Landslides  Watersheds  \
33553                     4           6         5.0
9427                      5           5         9.0
199                       7           5         5.0
12447                     2           3         4.0
39489                     7           2         6.0
```

```
       DeterioratingInfrastructure  PopulationScore  WetlandLoss  \
33553                            4                2            5
9427                             2               10            3
199                              2                4            5
12447                            5                5            2
39489                            5                8            3

       InadequatePlanning  PoliticalFactors  Predicted_FloodProbability
33553                   5               3.0                1.498805e-09
9427                    2               4.0                4.566341e-02
199                     7               3.0                7.892715e-07
12447                   3               4.0                1.210414e-08
39489                   5               3.0                1.412181e-07

[5 rows x 21 columns]
(10000,)
       MonsoonIntensity  TopographyDrainage  RiverManagement  Deforestation  \
33553                 7                 3.0              4.0              4
9427                  4                 7.0              7.0              3
199                   2                 3.0              3.0              4
12447                 5                 4.0              5.0              7
39489                 3                 9.0              5.0              3

       Urbanization  ClimateChange  DamsQuality  Siltation  \
33553             5              6            2          5
9427              5              7            5          2
199               5              4            7          7
12447             8              2            5          5
39489             3              4            6          9
```

[36]:

```python
# Train and Test Score of Logistic Regression
from sklearn.metrics import accuracy_score
print("Train Accuracy score of Logistic Regression",lr.score(X_train,y_train)*100);
print("Test Accuracy score of Logistic Regression",lr.score(X_test,y_test)*100);
print("Accuracy of score Logistic Regression",accuracy_score(y_test,lr_pred)*100)
```

```
Train Accuracy score of Logistic Regression 99.3375
Test Accuracy score of Logistic Regression 99.42999999999999
Accuracy of score Logistic Regression 99.44
```

0.3.2 9.2) Making Prediction on test by using KNN

[37]:

```python
# Train and Test Score of KNN
print("Train Accuracy score of KNN",knn.score(X_train,y_train)*100);
print("Test Accuracy score of KNN",knn.score(X_test,y_test)*100);
print("Accuracy of score KNN",accuracy_score(y_test,knn_pred)*100)
```

```
Train Accuracy score of Logistic Regression 99.3375
Test Accuracy score of Logistic Regression 99.42999999999999
Accuracy of score Logistic Regression 99.44
Train Accuracy score of KNN 90.9825
```

0.3.3 9.3) Making Prediction on test by using Naive-Bayes

[38]:

```
# Train and Test Score of Naive-Bayes
print("Train Accuracy score of Naive-Bayes",nb.score(X_train,y_train)*100);
print("Test Accuracy score of Naive-Bayes",nb.score(X_test,y_test)*100);
print("Accuracy of score Naive-Bayes",accuracy_score(y_test,nb_pred)*100)
```

```
Train Accuracy score of Naive-Bayes 90.9575
Test Accuracy score of Naive-Bayes 90.44
Accuracy of score Naive-Bayes 90.44
```

9.4) Making Prediction on test by using SVM

[39]:

```
# Train and Test Score of SVM
print("Train Accuracy score of SVM",sv.score(X_train,y_train)*100);
print("Test Accuracy score of SVM",sv.score(X_test,y_test)*100);
print("Accuracy of score SVM",accuracy_score(y_test,sv_pred)*100)
```

```
Train Accuracy score of SVM 99.2525
Test Accuracy score of SVM 98.14
Accuracy of score SVM 98.14
```

0.3.4 9.5) Making Prediction on test by using Decission Tree

[40]:

```
# Train and Test Score of Decision Tree
print("Train Accuracy score of Decision Tree",dt.score(X_train,y_train)*100);
print("Test Accuracy score of Decision Tree",dt.score(X_test,y_test)*100);
print("Accuracy of score Decision Tree",accuracy_score(y_test,dt_pred)*100)
```

```
Train Accuracy score of Decision Tree 100.0
Test Accuracy score of Decision Tree 69.46
Accuracy of score Decision Tree 69.46
```

0.3.5 9.6) Making Prediction on test by using Random Forest

[41]:

```
# Train and Test Score of Random Forest
print("Train Accuracy score of Random Forest",rf.score(X_train,y_train)*100);
print("Test Accuracy score of Random Forest",rf.score(X_test,y_test)*100);
print("Accuracy of score Random Forest",accuracy_score(y_test,rf_pred)*100)
```

```
Train Accuracy score of Random Forest 100.0
Test Accuracy score of Random Forest 89.35
Accuracy of score Random Forest 89.35
```

0.4 10.2) Confusion Matrix

Confusion metrices is a Table which is used to describe the performance of classificaion problem

It visualizes the accuracy of a classifier by comparing predicted values with actual values.

The terms used in confusion matrices are True Positive(TP), True Negetive(TN), False Positive(FP) and False Negetive(FN)

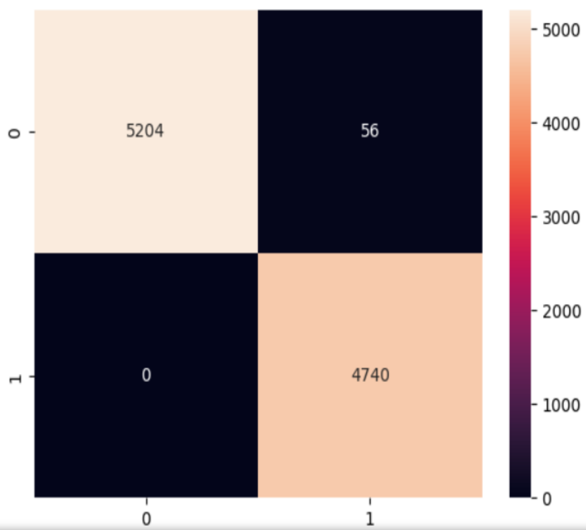
0.4.1 10.2.1) Confusion Matrix of Logistic Regression

[42]:

```
array([[5204,   56],
       [   0, 4740]], dtype=int64)
```

```
sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt="d")
```

```
<Axes: >
```



[43]: (86, 11, 24, 33)

```
TN = cm[0,0]
FP = cm[0,1]
FN = cm[1,0]
TP = cm[1,1]
TN, FP, FN, TP
```

```
(5204, 56, 0, 4740)
```

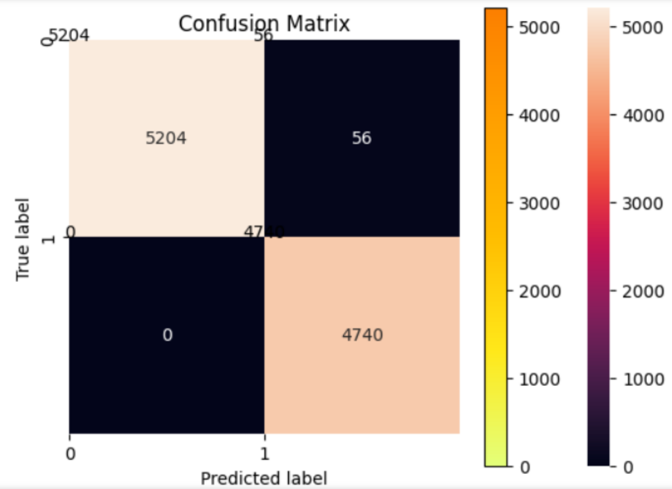
[44]:

```
# Making the Confusion Matrix Of Logistic Regression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm=confusion_matrix(y_test, lr_pred)
print('TN - True Negative {}'.format(cm[0,0]))
print('FP False Positive {}'.format(cm[0,1]))
print('FN False Negative {}'.format(cm[1,0]))
print('TP True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 5204
FP False Positive 56
FN False Negative 0
TP True Positive 4740
Accuracy Rate: 99.44
Misclassification Rate: 0.5599999999999999
```

[45]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Assuming y_test and lr_pred are defined
cm = confusion_matrix(y_test, lr_pred)

# Plot confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
plt.title('Confusion Matrix')
plt.colorbar()
classNames = ['0', '1']

tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames)
plt.yticks(tick_marks, classNames)

for i in range(len(classNames)):
    for j in range(len(classNames)):
        plt.text(j, i, cm[i, j], horizontalalignment="center", color="black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



[46]:

```
tick_marks = np.arange(len(classNames))
plt.xticks(tick_marks, classNames)
plt.yticks(tick_marks, classNames)

for i in range(len(classNames)):
    for j in range(len(classNames)):
        plt.text(j, i, cm[i, j], horizontalalignment="center", color="black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()


pd.crosstab(y_test, lr_pred, margins=False)
```

col_0	0	1
FloodRisk		
0	5204	56
1	0	4740

[47]:

```
pd.crosstab(y_test, lr_pred, margins=True)
```

col_0	0	1	All
FloodRisk			
0	5204	56	5260
1	0	4740	4740
All	5204	4796	10000

[48]:

```
pd.crosstab(y_test, lr_pred, rownames=['Actual values'], colnames=['Predicted_ ↪values'], margins=True)
```

Predicted_ values	0	1	All
Actual values			
0	5204	56	5260
1	0	4740	4740
All	5204	4796	10000

0.5 PRECISION (PPV-Positive Predictive value)

It is the ratio of correctly predicted positive (TP) observations to the total predicted positive (TP+FP) observations.

Precision=TP/(TP+FP)

Where TP=True Positive

FP–False Positive

[49]:

```
TP,FP
```

[49]: (4740, 56)

[50]:

```
from sklearn.metrics import confusion_matrix

# Assuming y_test and lr_pred are defined
cm = confusion_matrix(y_test, lr_pred)

# Extract True Positives (TP), False Positives (FP), True Negatives (TN), False Negatives (FN)
TP = cm[1, 1]
FP = cm[0, 1]

# Calculate precision
Precision = TP / (TP + FP)

# Print precision
print("Precision:", Precision)
```

```
Precision: 0.9883236030025021
```

[51]:

```
# print precision score
precision_Score = TP / float (TP + FP)*100
print('Precision score: {0:0.4f}'.format(precision_Score))
```

```
Precision score: 98.8324
```

[52]:

```
from sklearn.metrics import precision_score
print("precision Score is:", precision_score(y_test,lr_pred)*100)
print("Mircro Average precision Score is:", precision_score(y_test, lr_pred,average='micro')*100)
print("Marcro Average precision Score is:", precision_score(y_test, lr_pred,average='macro')*100)
print("Weighted Average precision Score is:", precision_score (y_test, lr_pred,average='weighted')*100)
print("precision Score on Non weighted score is:", precision_score (y_test,lr_pred, average=None)*100)
```

```
precision Score is: 98.83236030025022
Mircro Average precision Score is: 99.44
Marcro Average precision Score is: 99.4161801501251
Weighted Average precision Score is: 99.44653878231861
precision Score on Non weighted score is: [100.         98.8323603]
```

[53]:

```
print('Classification Report of Logistic Regression: \n',classification_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:
               precision    recall  f1-score   support

           0     1.0000    0.9894    0.9946      5260
           1     0.9883    1.0000    0.9941      4740

    accuracy                         0.9944     10000
   macro avg     0.9942    0.9947    0.9944     10000
weighted avg     0.9945    0.9944    0.9944     10000
```

0.6 Recall (True Positive Rate (TPR))

It is ratio of correctly predicted positive(Tp) observations to the total observations which are actually true.

[54]:
```
recall_score = TP / float (TP + FN)*100
print("recall score", recall_score)
```

recall score 100.0

[55]:
```
from sklearn.metrics import recall_score
print("Recall or Sensitivity score :",recall_score (y_test, lr_pred)*100)
```

Recall or Sensitivity score : 100.0

[56]:
```
print("Mircro Average Recall Score is:", recall_score(y_test, lr_pred,average='micro')*100)
print("Marcro Average Recall Score is:", recall_score (y_test, lr_pred,average='macro')*100)
print("Weighted Average Recall Score is: ", recall_score (y_test, lr_pred,average='weighted')*100)
print("Recall Score on Non weighted score is:", recall_score (y_test, lr_pred,average=None)*100)
```

```
Mircro Average Recall Score is: 99.44
Marcro Average Recall Score is: 99.46768060836501
Weighted Average Recall Score is:  99.44
Recall Score on Non weighted score is: [ 98.93536122 100.        ]
```

[57]:
```
print('Classification Report of Logistic Regression: \n',classification_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:
              precision    recall  f1-score   support

           0     1.0000    0.9894    0.9946      5260
           1     0.9883    1.0000    0.9941      4740

    accuracy                         0.9944     10000
   macro avg     0.9942    0.9947    0.9944     10000
weighted avg     0.9945    0.9944    0.9944     10000
```

0.7 False Positive Rate (FPR)

[58]:
```
FPR = FP / float (FP+ TN)*100
print('False Positive Rate: {0:0.4f}'.format(FPR))
```

False Positive Rate: 1.0646

0.8 Specificity

[59]:
```
specificity = TP / float (TN+ FP)*100
print('Specificity : {0:0.4f}'.format(specificity))
```

Specificity : 90.1141

0.9 F1 Score

[60]:
```
from sklearn.metrics import f1_score
print('f1_score of macro :',f1_score(y_test, lr_pred)*100)
```

f1_score of macro : 99.41275167785236

[61]:
```
print('Mircro Average F1 Score is:', f1_score (y_test, lr_pred,average='micro')*100)
print('Marcro Average F1 Score is:', f1_score (y_test, lr_pred,average='macro')*100)
print('Weighted Average F1 Score is:', f1_score(y_test, lr_pred,average='weighted')*100)
print('F1 Score on Non weighted score is:', f1_score (y_test, lr_pred,average=None)*100)
```

```
Mircro Average F1 Score is: 99.44
Marcro Average F1 Score is: 99.43879174106685
Weighted Average F1 Score is: 99.440145824354
F1 Score on Non weighted score is: [99.4648318  99.41275168]
```

[62]:

0.10 Classification Report of Logistic Regression

```
from sklearn.metrics import classification_report
print('Classification Report of Logistic Regression: \n', classification_report(y_test, lr_pred, digits=4))
```

```
Classification Report of Logistic Regression:
               precision    recall  f1-score   support

           0     1.0000    0.9894    0.9946      5260
           1     0.9883    1.0000    0.9941      4740

    accuracy                         0.9944     10000
   macro avg     0.9942    0.9947    0.9944     10000
weighted avg     0.9945    0.9944    0.9944     10000
```

0.11 ROC Curve & ROC AUC

ROC curve is one the important evaluating metrics that should be used to check the performance of an classification model.It is also called relative operating characteristic curve, because it is a comparison of two main characteristics (TPR and FPR). It is plotted between sensitivity(aka recall aka True Positive Rate) and False Positive Rate (FPR = 1-specificity).

ROC (Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things (e.g If a patient has a disease or no).

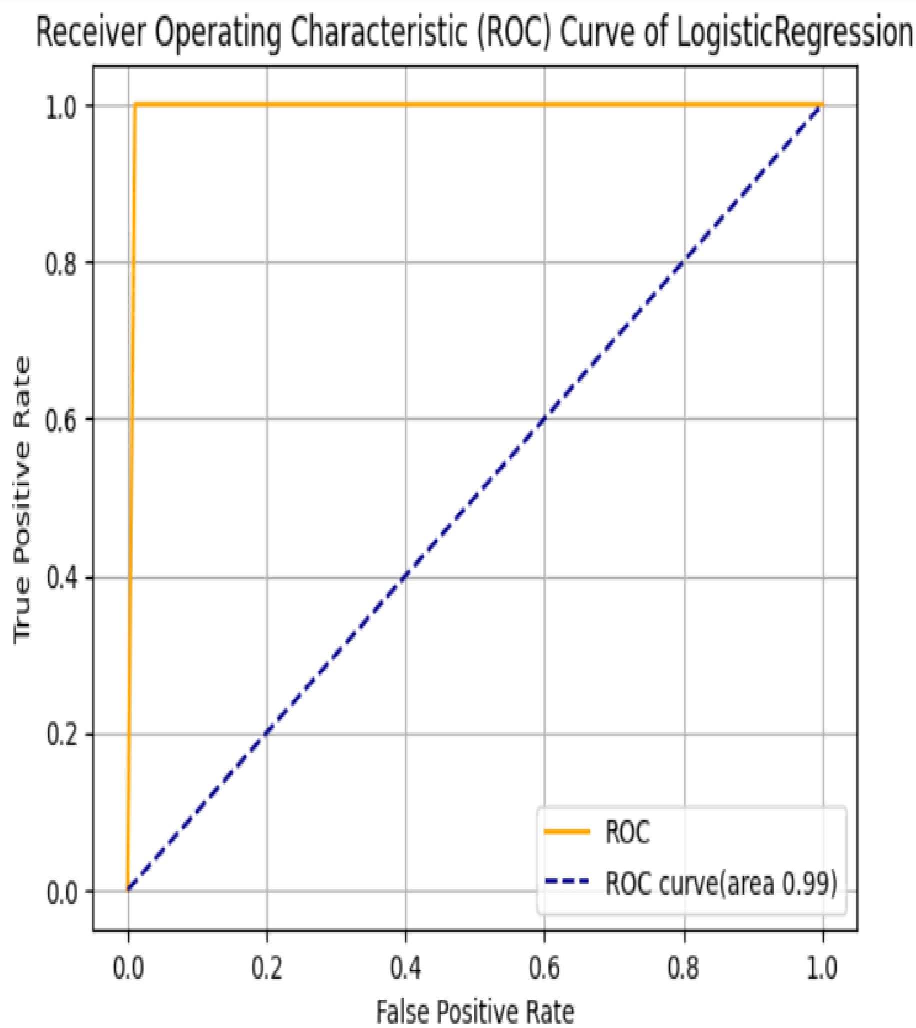
Area Under Curve (AUC) helps us to choose the best model amongst the models for which we have plotted the ROC curves

[63]:
```
# Area Under Curve

auc = roc_auc_score(y_test, lr_pred)
print("ROC AUC SCORE of Logistic Regression is", auc)
```

```
ROC AUC SCORE of Logistic Regression is 0.9946768060836502
```

[64]:

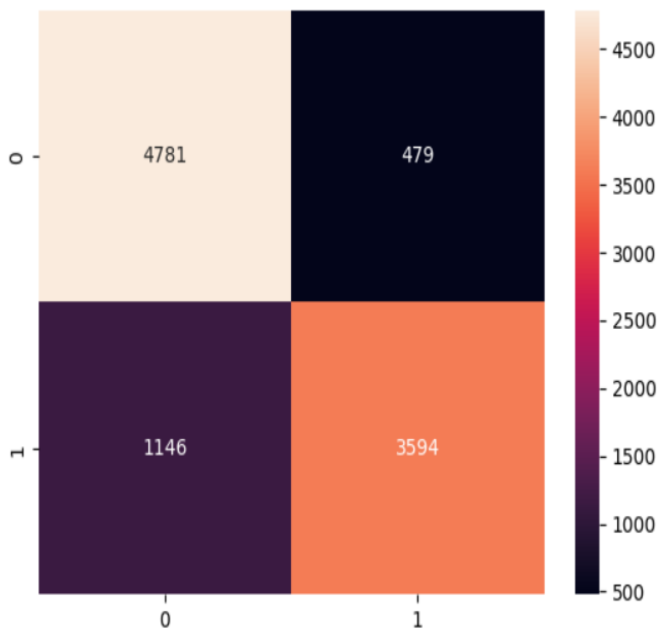
```
fpr, tpr, thresholds = roc_curve (y_test, lr_pred)
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='ROC curve(area %0.2f)' % auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of LogisticRegression')
plt.legend()
plt.grid()
plt.show()
```



[65]:

```
sns.heatmap(confusion_matrix(y_test,knn_pred),annot=True,fmt="d")
```

```
<Axes: >
```



[66]:

```
# making the confusion matrix of knn
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, knn_pred)
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np. sum(cm))*100))
print('Misclassification Rate: {}'.format(np.divide(np. sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 4781
FP - False Positive 479
FN - False Negative 1146
TP - True Positive 3594
Accuracy Rate: 83.75
Misclassification Rate: 16.25
```

[67]:

```
# classification report of knn
print(' Classification Report of KNN: \n',classification_report(y_test,knn_pred,digits=4))
```

```
 Classification Report of KNN: 
               precision    recall  f1-score   support

           0     0.8066    0.9089    0.8547      5260
           1     0.8824    0.7582    0.8156      4740

    accuracy                         0.8375     10000
   macro avg     0.8445    0.8336    0.8352     10000
weighted avg     0.8426    0.8375    0.8362     10000
```

0.12 Area Under Curve of KNN

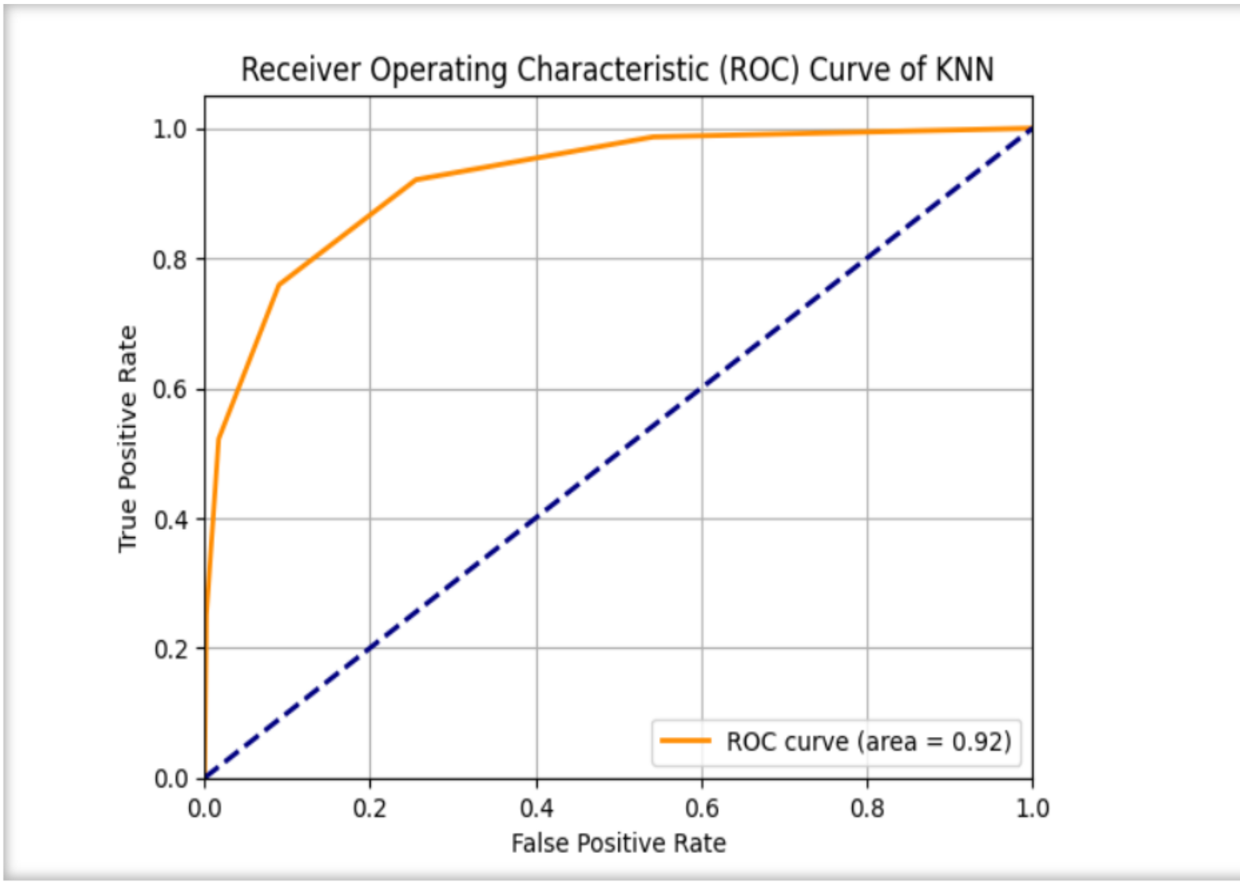
[68]:

```
# Area Under Curve

auc = roc_auc_score(y_test, knn_pred)
print("ROC AUC SCORE of KNN is", auc)
```

ROC AUC SCORE of KNN is 0.8335816046589979

[69]:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Assuming data is prepared and split into X and y
X = df.drop(columns=['FloodProbability', 'FloodRisk'])
y = df['FloodRisk']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train KNN model
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
# Predict probabilities
y_pred_proba = knn.predict_proba(X_test)[:, 1]
# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of KNN')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



0.12.1 10.2.3 Confusion Matrix of "Naive Bayes"

[70]:

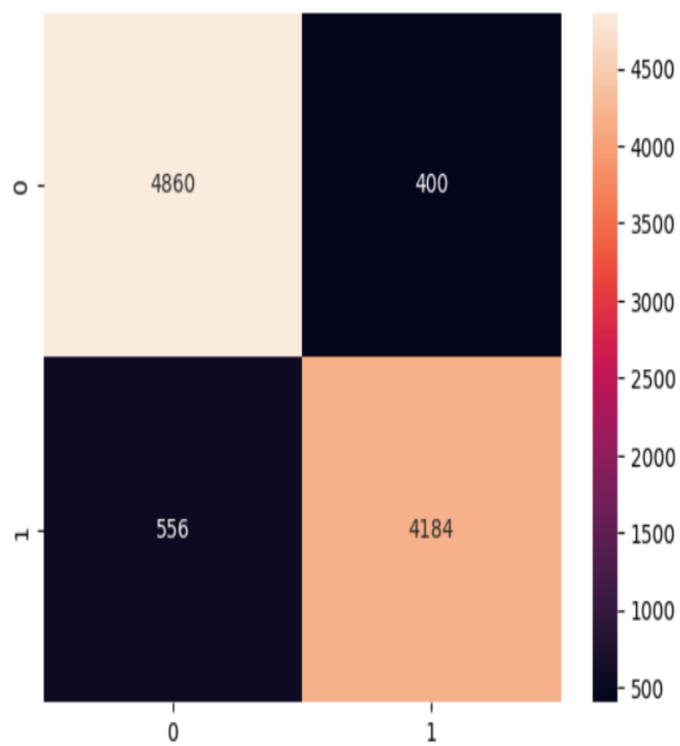
```
# making the confusion matrix of Naive Bayes
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, nb_pred)
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate of Naive Bayes: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 4860
FP - False Positive 400
FN - False Negative 556
TP - True Positive 4184
Accuracy Rate: 90.44
Misclassification Rate of Naive Bayes: 9.56
```

[71]:

```
sns.heatmap(confusion_matrix(y_test,nb_pred),annot=True,fmt="d")
```

```
<Axes: >
```



0.13 Classification report of "Naive Bayes"

[72]:

```
# classification report of Naive Bayes
print(' Classification Report of Naive Bayes: \n',classification_report(y_test,nb_pred,digits=4))
```

```
Misclassification Rate of Naive Bayes: 9.56
 Classification Report of Naive Bayes:
               precision    recall  f1-score   support

           0     0.8973    0.9240    0.9105      5260
           1     0.9127    0.8827    0.8975      4740

    accuracy                         0.9044     10000
   macro avg     0.9050    0.9033    0.9040     10000
weighted avg     0.9046    0.9044    0.9043     10000
```

0.14 Roc AUC Score of Naive Bayes

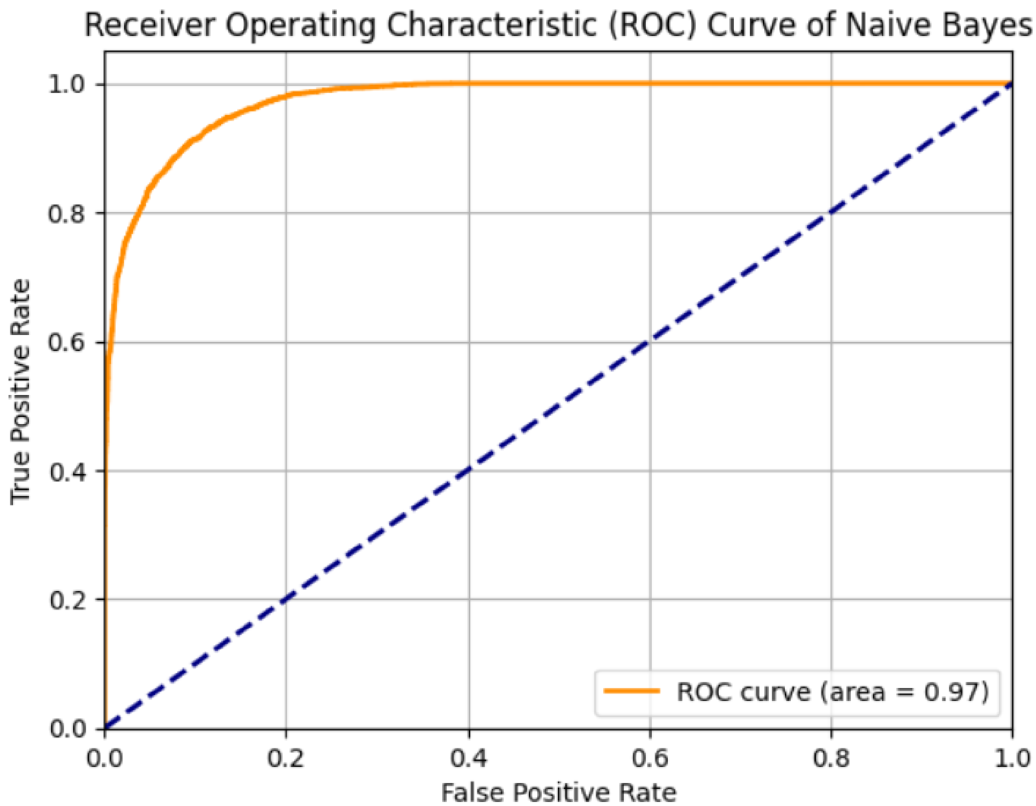
[73]:

```
# Area Under Curve

auc = roc_auc_score(y_test, nb_pred)
print("ROC AUC SCORE of Naive Bayes is", auc)
```

ROC AUC SCORE of Naive Bayes is 0.9033273972822512

[74]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Assuming data is prepared and split into X and y
X = df.drop(columns=['FloodProbability', 'FloodRisk'])
y = df['FloodRisk']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train Naive Bayes model
nb = GaussianNB()
nb.fit(X_train, y_train)
# Predict probabilities
y_pred_proba = nb.predict_proba(X_test)[:, 1]
# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Naive Bayes')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



Confusion matric of “SVM”

[75]:

```
# making the confusion matrix of SVM
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, sv_pred)
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))*100))
print('Misclassification Rate of SVM: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))*100))
```

```
TN - True Negative 5150
FP - False Positive 110
FN - False Negative 76
TP - True Positive 4664
Accuracy Rate: 98.14
Misclassification Rate of SVM: 1.8599999999999999
```

0.15 Classification Report of SVM

[76]:

```
# classification report of SVM
print(' Classification Report of SVM: \n',classification_report(y_test,sv_pred,digits=4))
```

```
 Classification Report of SVM:
               precision    recall  f1-score   support

           0     0.9855    0.9791    0.9823      5260
           1     0.9770    0.9840    0.9804      4740

    accuracy                         0.9814     10000
   macro avg     0.9812    0.9815    0.9814     10000
weighted avg     0.9814    0.9814    0.9814     10000
```

0.16 Roc AUC of SVM

[77]:

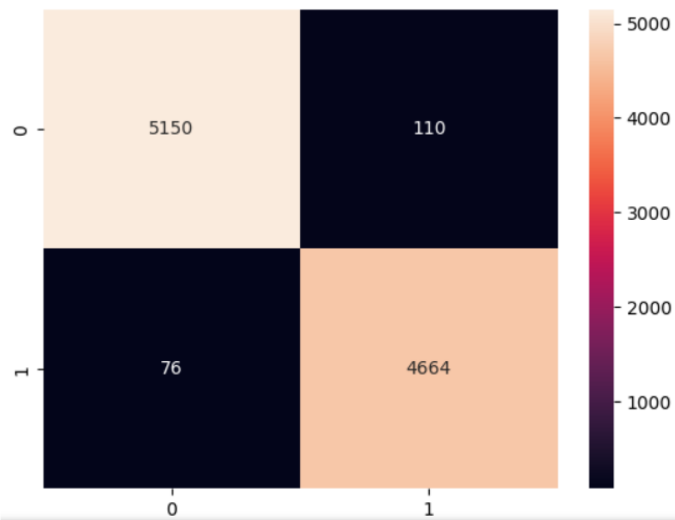
```
# Area Under Curve
from sklearn.metrics import roc_auc_score
auc = round(roc_auc_score(y_test, sv_pred)*100,2)
print("ROC AUC SCORE of SVM is", auc)
```

ROC AUC SCORE of SVM is 98.15

[78]:

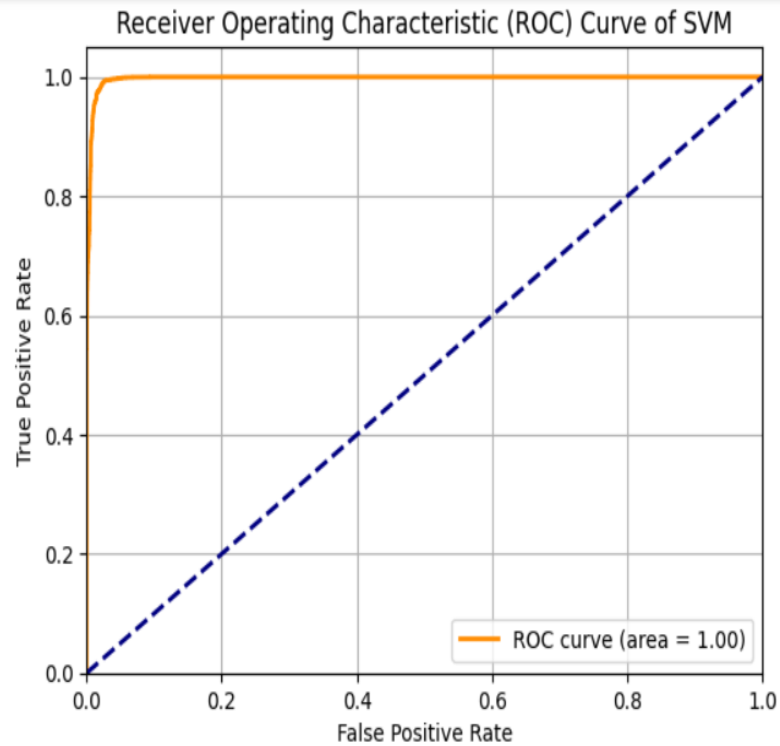
```
sns.heatmap(confusion_matrix(y_test,sv_pred),annot=True,fmt="d")
```

```
<Axes: >
```



[79]:

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Assuming data is prepared and split into X and y
X = df.drop(columns=['FloodProbability', 'FloodRisk'])
y = df['FloodRisk']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train SVM model with probability=True to enable probability estimates
svm = SVC(probability=True, random_state=42)
svm.fit(X_train, y_train)
# Predict probabilities
y_pred_proba = svm.predict_proba(X_test)[:, 1]
# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of SVM')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

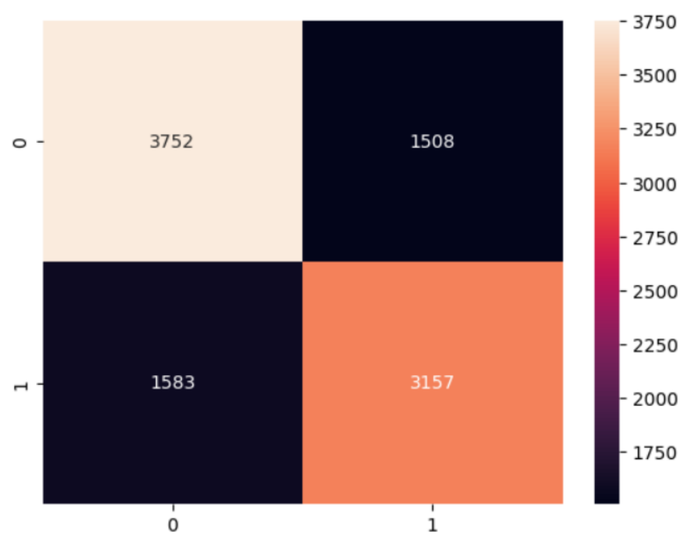


0.16.1 Confusion Matrix of "Decision Tree"

[80]:

```
sns.heatmap(confusion_matrix(y_test,dt_pred),annot=True,fmt="d")
```

```
<Axes: >
```



[81]:

```
# making the confusion matrix of DT
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, dt_pred)
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))))
print('Misclassification Rate of DT: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))))
```

```
TN - True Negative 3794
FP - False Positive 1466
FN - False Negative 1565
TP - True Positive 3175
Accuracy Rate: 0.6969
Misclassification Rate of DT: 0.3031
```

[82]:

```
# classification report of DT
print(' Classification Report of Decision Tree: \n',classification_report(y_test,dt_pred,digits=4))
```

```
 Classification Report of Decision Tree:
               precision    recall  f1-score   support

           0     0.7067    0.7167    0.7117      5260
           1     0.6806    0.6698    0.6752      4740

    accuracy                         0.6945     10000
   macro avg     0.6936    0.6933    0.6934     10000
weighted avg     0.6943    0.6945    0.6944     10000
```

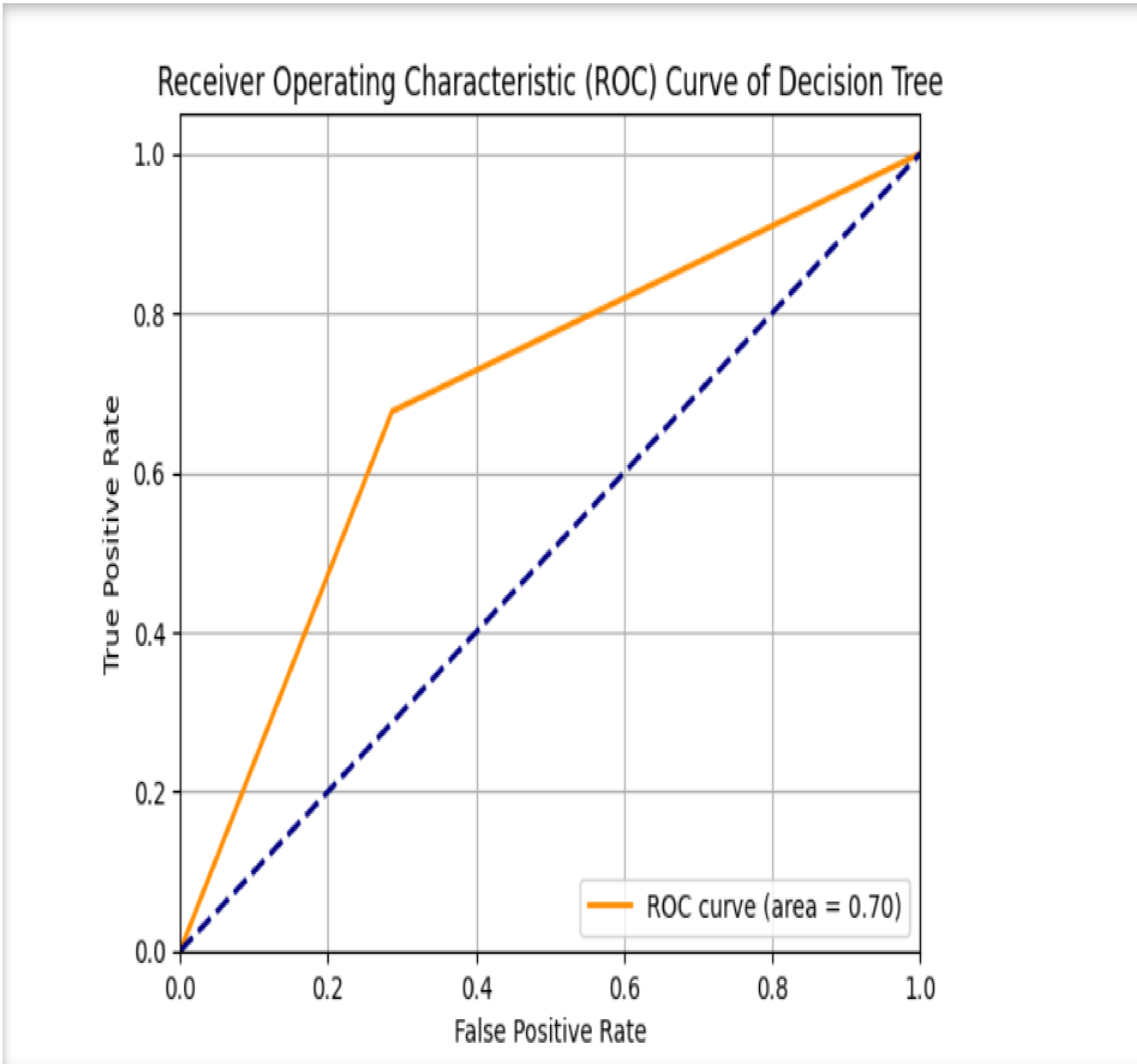
[83]:

```
# Area Under Curve
from sklearn.metrics import roc_auc_score
auc = round(roc_auc_score(y_test, dt_pred)*100,2)
print("ROC AUC SCORE of DT is", auc)
```

ROC AUC SCORE of DT is 69.29

[84]:

```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Assuming data is prepared and split into X and y
X = df.drop(columns=['FloodProbability', 'FloodRisk'])
y = df['FloodRisk']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train Decision Tree model
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
# Predict probabilities
y_pred_proba = dt.predict_proba(X_test)[:, 1]
# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Decision Tree')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```

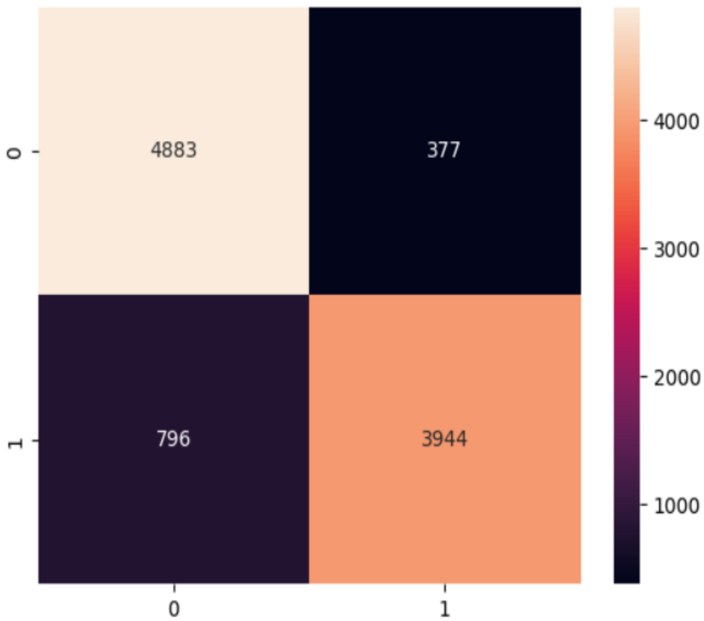

Receiver Operating Characteristic (ROC) Curve of Decision Tree
True Positive Rate
False Positive Rate
ROC curve (area = 0.70)
0.0
0.2
0.4
0.6
0.8
1.0

0.17 RANDOM FOREST

[85]:

```
# heat map of random forest
sns.heatmap(confusion_matrix(y_test,rf_pred),annot=True,fmt="d")
```

```
<Axes: >
```



[86]:

```
# making the confusion matrix of Random Forest
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve
cm = confusion_matrix(y_test, rf_pred)
print('TN - True Negative {}'.format(cm[0,0]))
print('FP - False Positive {}'.format(cm[0,1]))
print('FN - False Negative {}'.format(cm[1,0]))
print('TP - True Positive {}'.format(cm[1,1]))
print('Accuracy Rate: {}'.format(np.divide(np.sum([cm[0,0],cm[1,1]]),np.sum(cm))))
print('Misclassification Rate of Random Forest: {}'.format(np.divide(np.sum([cm[0,1],cm[1,0]]),np.sum(cm))))
```

```
TN - True Negative 4934
FP - False Positive 326
FN - False Negative 764
TP - True Positive 3976
Accuracy Rate: 0.891
Misclassification Rate of Random Forest: 0.109
```

0.17 Classification report of Random Forest

[87]:

```
# classification report of Random forest
print(' Classification Report of Random Forest: \n',classification_report(y_test,rf_pred,digits=4))
```

```
 Classification Report of Random Forest:
               precision    recall  f1-score   support

           0     0.8618    0.9376    0.8981      5260
           1     0.9233    0.8331    0.8759      4740

    accuracy                         0.8881     10000
   macro avg     0.8925    0.8854    0.8870     10000
weighted avg     0.8909    0.8881    0.8876     10000
```

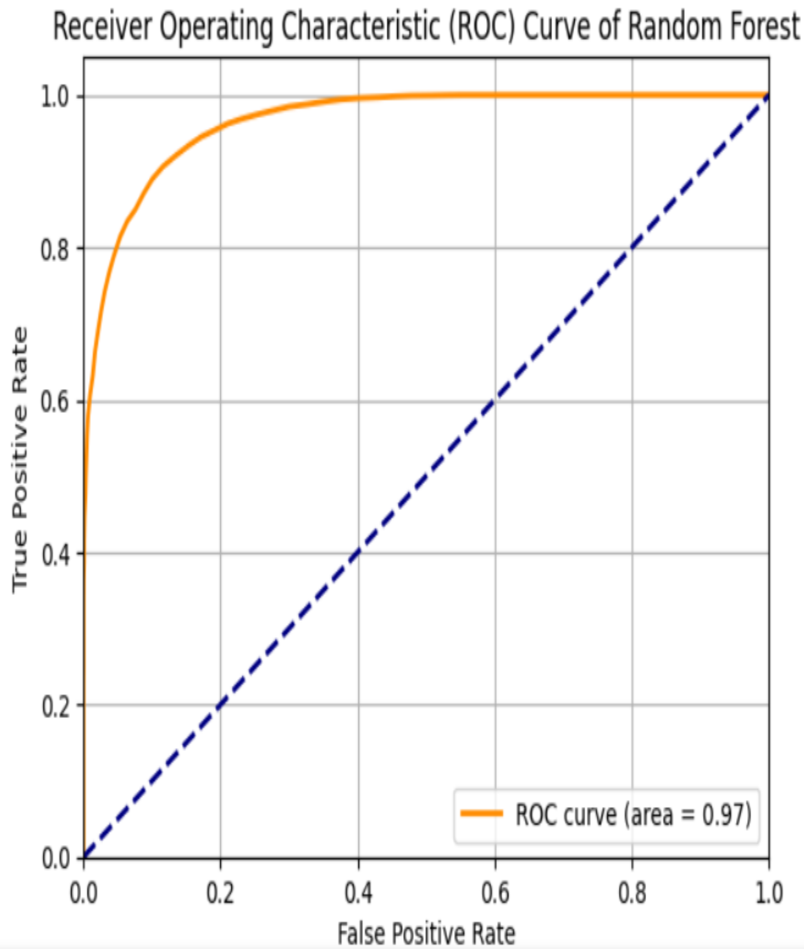
[88]:

```
# Area Under Curve
from sklearn.metrics import roc_auc_score
auc = round(roc_auc_score(y_test, rf_pred)*100,2)
print("ROC AUC SCORE of Random Forest is", auc)
```

```
ROC AUC SCORE of Random Forest is 88.82
```

[89]:

```
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Assuming data is prepared and split into X and y
X = df.drop(columns=['FloodProbability', 'FloodRisk'])
y = df['FloodRisk']
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Train Random Forest model
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
# Predict probabilities
y_pred_proba = rf.predict_proba(X_test)[:, 1]
# Compute ROC curve
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve of Random Forest')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```


Receiver Operating Characteristic (ROC) Curve of Random Forest
True Positive Rate
False Positive Rate
ROC curve (area = 0.97)
0.0
0.2
0.4
0.6
0.8
1.0