Predictive model to predict flood based on monthly rainfall

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

df=pd.read_csv('E:/PYTHON_PROJECTS/1PYTHON_MODEL/kerala.csv')
df.head()
```

Out[1]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL RAINFALL
0	KERALA	1901	28.7	44.7	51.6	160.0	174.7	824.6	743.0	357.5	197.7	266.9	350.8	48.4	3248.6
1	KERALA	1902	6.7	2.6	57.3	83.9	134.5	390.9	1205.0	315.8	491.6	358.4	158.3	121.5	3326.6
2	KERALA	1903	3.2	18.6	3.1	83.6	249.7	558.6	1022.5	420.2	341.8	354.1	157.0	59.0	3271.2
3	KERALA	1904	23.7	3.0	32.2	71.5	235.7	1098.2	725.5	351.8	222.7	328.1	33.9	3.3	3129.7
4	KERALA	1905	1.2	22.3	9.4	105.9	263.3	850.2	520.5	293.6	217.2	383.5	74.4	0.2	2741.6

Exploration of the dataset

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118 entries, 0 to 117
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   SUBDIVISION       118 non-null    object 
 1   YEAR              118 non-null    int64  
 2   JAN               118 non-null    float64
 3   FEB               118 non-null    float64
 4   MAR               118 non-null    float64
 5   APR               118 non-null    float64
 6   MAY               118 non-null    float64
 7   JUN               118 non-null    float64
 8   JUL               118 non-null    float64
 9   AUG               118 non-null    float64
 10  SEP               118 non-null    float64
 11  OCT               118 non-null    float64
 12  NOV               118 non-null    float64
 13  DEC               118 non-null    float64
 14   ANNUAL RAINFALL  118 non-null    float64
 15  FLOODS            118 non-null    object 
dtypes: float64(13), int64(1), object(2)
memory usage: 14.9+ KB
```

In [3]:
```
df.shape
```

Out[3]:
```
(118, 16)
```

describe()

The describe() function summarizes the dataset's statistical properties, such as count, mean, min, and max:

In [4]:
```
df.describe()
```

Out[4]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AU
count	118.000000	118.000000	118.000000	118.000000	118.000000	118.000000	118.000000	118.000000	118.0000
mean	1959.500000	12.218644	15.633898	36.670339	110.330508	228.644915	651.617797	698.220339	430.3694
std	34.207699	15.473766	16.406290	30.063862	44.633452	147.548778	186.181363	228.988966	181.9804
min	1901.000000	0.000000	0.000000	0.100000	13.100000	53.400000	196.800000	167.500000	178.6000
25%	1930.250000	2.175000	4.700000	18.100000	74.350000	125.050000	535.550000	533.200000	316.7250
50%	1959.500000	5.800000	8.350000	28.400000	110.400000	184.600000	625.600000	691.650000	386.2500
75%	1988.750000	18.175000	21.400000	49.825000	136.450000	264.875000	786.975000	832.425000	500.1000
max	2018.000000	83.500000	79.000000	217.200000	238.000000	738.800000	1098.200000	1526.500000	1398.9000

It's also useful to see if any column has null values since it shows us the count of values in each one.

Corr()

The corr() function display the correlation between different variables in dataset

In [5]:
```
df.corr()
```

Out[5]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
YEAR	1.000000	-0.225531	0.003879	-0.012842	0.086865	-0.059661	-0.174938	-0.223403	0.044173	0.107655
JAN	-0.225531	1.000000	0.019613	0.078626	0.034807	0.071420	0.189375	0.034423	0.008677	-0.113502
FEB	0.003879	0.019613	1.000000	0.245375	0.123706	-0.083500	0.054114	0.005789	0.023259	0.066317
MAR	-0.012842	0.078626	0.245375	1.000000	0.074014	-0.102961	0.019000	0.018330	0.042411	0.143850
APR	0.086865	0.034807	0.123706	0.074014	1.000000	-0.114566	0.072990	0.014977	-0.047842	0.012928
MAY	-0.059661	0.071420	-0.083500	-0.102961	-0.114566	1.000000	0.001235	-0.046518	-0.124412	0.116860
JUN	-0.174938	0.189375	0.054114	0.019000	0.072990	0.001235	1.000000	0.094939	-0.014549	-0.052634
JUL	-0.223403	0.034423	0.005789	0.018330	0.014977	-0.046518	0.094939	1.000000	0.154467	0.209441
AUG	0.044173	0.008677	0.023259	0.042411	-0.047842	-0.124412	-0.014549	0.154467	1.000000	0.098215
SEP	0.107655	-0.113502	0.066317	0.143850	0.012928	0.116860	-0.052634	0.209441	0.098215	1.000000
OCT	-0.030223	-0.035044	0.053133	-0.023066	0.113172	0.197102	0.001156	0.025223	-0.181496	-0.032348
NOV	-0.130129	-0.011034	-0.162880	-0.032612	0.022206	0.094934	0.015967	-0.028526	-0.112729	-0.027615
DEC	-0.123643	-0.089809	-0.127025	0.026292	-0.110392	-0.118077	-0.085188	-0.013573	0.142090	-0.011007
ANNUAL RAINFALL	-0.198048	0.118648	0.061457	0.116103	0.112358	0.314723	0.453407	0.651990	0.413036	0.428344

The closer to 1, the stronger the correlation between these variables.

A minus sign means that these 2 variables are negatively correlated, i.e. one decreases with increasing the other and vice versa.

Replace

In order to train this python model, we need the values of our target output to be 0 & 1. So, we'll replace values in the Floos column (YES,NO) with (1,0 respectively)

In [6]:
```
df['FLOODS'].replace(['YES','NO'],[1,0],inplace=True)
df.head()
```

Out[6]:

	SUBDIVISION	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL RAINFALL
0	KERALA	1901	28.7	44.7	51.6	160.0	174.7	824.6	743.0	357.5	197.7	266.9	350.8	48.4	3248.6
1	KERALA	1902	6.7	2.6	57.3	83.9	134.5	390.9	1205.0	315.8	491.6	358.4	158.3	121.5	3326.6
2	KERALA	1903	3.2	18.6	3.1	83.6	249.7	558.6	1022.5	420.2	341.8	354.1	157.0	59.0	3271.2
3	KERALA	1904	23.7	3.0	32.2	71.5	235.7	1098.2	725.5	351.8	222.7	328.1	33.9	3.3	3129.7
4	KERALA	1905	1.2	22.3	9.4	105.9	263.3	850.2	520.5	293.6	217.2	383.5	74.4	0.2	2741.6

in place= True means we want this replacement to be reflected in the original dataset, i.e the change is permanent.

Feature Selection

In this step, we choose several features that contribute most to the target output. So, instead of training the model using every column in our dataset, we select only those that have the strongest relationship with the predicted variable.

Use the SelectKBest library to run a chi-squared statistical test and select the top 3 features that are most related to floods.

In [7]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

After, define X & Y

In [8]:
```
X= df.iloc[:,1:14] #all features
Y= df.iloc[:,-1] #target output (floods)
```

Select the top 3 features

In [9]:
```
best_features= SelectKBest(score_func=chi2, k=3)
fit= best_features.fit(X,Y)
```

Now we create data frames for the features and the score of each feature:

In [10]:
```
df_scores= pd.DataFrame(fit.scores_)
```

```
df_columns= pd.DataFrame(X.columns)
```

Finally, we'll combine all the features and their corresponding scores in one data frame:

In [11]:
```
features_scores= pd.concat([df_columns, df_scores], axis=1)
features_scores.columns= ['Features', 'Score']
features_scores.sort_values(by = 'Score')
```

Out[11]:

	Features	Score
4	APR	2.498771
2	FEB	2.571626
0	YEAR	2.866463
12	DEC	11.609546
10	OCT	12.650485
3	MAR	21.696518
1	JAN	48.413088
11	NOV	284.674615
5	MAY	656.812145
8	AUG	739.975818
9	SEP	1000.379273
6	JUN	1218.856252
7	JUL	1722.612175

Here, we notice that the top 3 features that are most related to the target output are:

'SEP' which is the rainfall index in September

'JUN' is the rainfall index in June

'JUL' is the rainfall index in July

Build the Model

Now it's time to get our hands dirty. First, split the dataset into X and Y:

In [12]:
```
X=df[['SEP','JUN','JUL']] # THE TOP 3 FEATURES
Y= df[['FLOODS']] # the target output
```

Second, split the dataset into train and test

In [13]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y, test_size=0.4, random_state=100)
```

Third, create a logistice regression body

In [14]:
```
logreg= LogisticRegression()
logreg.fit(X_train,y_train)
```

```
E:\python_App\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A c
olumn-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)
```

Out[14]:
```
LogisticRegression()
```

Finally, we predict the likelihood of a flood using the logistic regression body we created:

In [15]:
```
y_pred=logreg.predict(X_test)
print (X_test) #test dataset
print (y_pred) #predicted values
```

```
       SEP     JUN     JUL
84   117.6   828.7   388.9
85   235.4   597.9   324.8
37   223.2   681.6   648.6
45   199.4   919.0   671.7
97   517.6   732.5   641.4
26   335.6   720.2   888.2
101   99.0   503.1   318.7
112  318.6  1042.7   830.2
21   222.4   663.1  1025.1
33    48.4   852.9   415.0
11   136.8   948.2   833.6
46   394.5   556.1   669.3
96   292.2   544.2   970.5
25   322.7   563.9   885.2
32   469.7   859.3   773.4
99   195.8   633.8   343.2
108  326.5   438.2   924.9
51    57.4   576.7   430.0
29   411.5   633.1   401.7
64   150.1   597.7   465.1
28   268.9   946.6   844.0
90    48.5  1096.1   905.5
89   103.3   528.6   635.4
54   438.5   782.4   392.8
35   286.7   620.8   672.1
43   155.0   498.9   614.1
73   383.6   266.9  1004.2
80   376.6   912.4   489.8
36   139.8   485.6   970.5
75   116.2   196.8   641.5
22   254.3   722.5  1008.7
82   421.1   322.8   583.2
113  298.8   454.4   677.8
5    131.2   414.9   954.2
105  474.8   482.4   804.0
23   289.1  1011.7  1526.5
104  414.7   619.2   832.7
100  216.8   715.3   598.5
20   156.7   489.1   639.8
41    99.8   813.6   828.8
6    225.0   770.9   760.4
3    222.7  1098.2   725.5
12   176.9   541.7   763.2
77   119.4   758.1   686.7
62   223.9   393.3   720.2
78   211.7   582.9   662.2
74   457.7   864.4   531.3
68   216.4   550.5   818.8
[0 0 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 0 0 1 1 1
 0 0 1 1 1 0 0 0 0 1 1]
```

Evaluate the Model's Performance

Now we'll evaluate how well our model performed predictive analytics by running a classification report and a ROC curve

Classification Report

Classification Report is a performance evaluation report that is used to evaluate the performance of machine learning models by the following 5 criteria

Accuracy is a score used to evaluate the model's performance. The higher it is,the betterRecall measures the model's ability to correctly predict the true positive values.Precision is the ratio of true positive to the sum of both true and false positivesF-score combines precision and recall into one metric. Ideally, its value should be closest to 1, the betterSupport is the number of actual occurrences of each class in the dataset

In [16]:
```
from sklearn import metrics
from sklearn.metrics import classification_report
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print('Recall: ',metrics.recall_score(y_test, y_pred, zero_division=1))
print('Precision',metrics.precision_score(y_test, y_pred, zero_division=1))
print('CL Report:',metrics.classification_report(y_test, y_pred, zero_division=1))
```

```
Accuracy:  0.8541666666666666
Recall:  0.8846153846153846
Precision 0.8518518518518519
CL Report:               precision    recall  f1-score   support

           0       0.86      0.82      0.84        22
           1       0.85      0.88      0.87        26

    accuracy                           0.85        48
   macro avg       0.85      0.85      0.85        48
weighted avg       0.85      0.85      0.85        48
```

As you can see, the model's performance in numbers is:

Accuracy: 85%Recall: 88%Precision: 85%

We can safely conclude that this nodel predicted the likelihood of a flood well

ROC Curve

The receiver operating characteristic (ROC) curve is used to display the sensitivity and specificity of the logistic regression model by calculating the true positive and false positive rates

From the ROC curve, we can calculate the area under the curve(AUC) whose value ranges from 0 to 1. You'll remember that the closer to 1, the better it is for our predictive modeling

To determine the ROC curve, first define the metrics

In [19]:
```
y_pred_proba=logreg.predict_proba(X_test) [::,1]
```

Then, calculate the true positive and false positive rates:

In [20]:
```
false_positive_rate, true_positive_rate, _ = metrics.roc_curve(y_test, y_pred_proba)
```

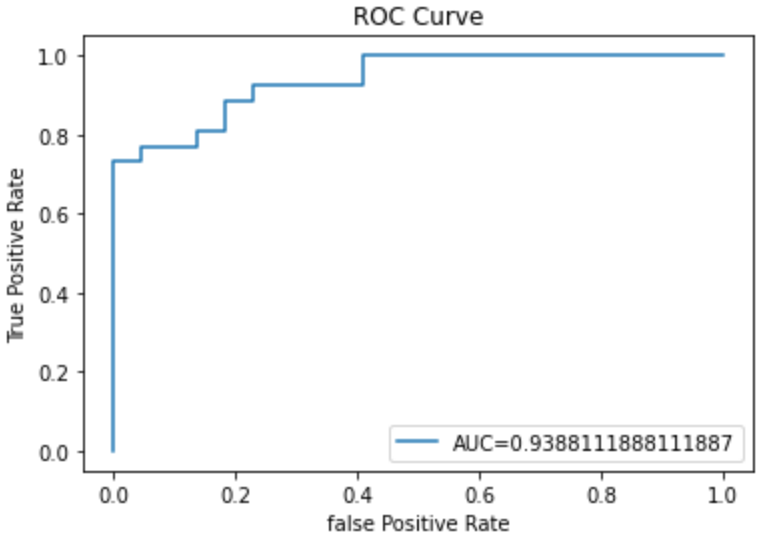
Next, calculate the AUC to see the model's performance:

In [21]:
```
auc= metrics.roc_auc_score(y_test, y_pred_proba)
```

Finally, plot the ROC curve:

In [22]:
```
plt.plot(false_positive_rate, true_positive_rate,label="AUC="+str(auc))
plt.title('ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('false Positive Rate')
plt.legend(loc=4)
```

Out[22]:
```
<matplotlib.legend.Legend at 0x2831f2c8520>
```



The AUC is 0.94, meaning that the model did a great job:

In [ ]: