Logistic Regression :

Logistic Regression Practical Implementation on Algerian Forest Fires Dataset

Life Cycle of Machine Learning Project

```
1.Understanding the Problem Statement
2.Data Collection
3.Exploratory data analysis
4.Data Cleaning
5.Data Pre-Processing
6.Model Training
7.Choose best model
```

1.Problem Statement

```
by using logistic regression to predict  out the calsses it is fire or not fire

1.if data is not imbalance then 90% accuracy
2.if data is imbalance
2.1.without handling imballance data do logistic regression and do precison recall score
2.2.handle imbalance data create a model
```

2.Data Collection

The Dataset is collected from UIC machine learning repository Dataset link : https://archive.ics.uci.edu/ml/datasets/Algerian+Forest+Fires+Dataset++ (https://archive.ics.uci.edu/ml/datasets/Algerian+Forest+Fires+Dataset++)

Data Set Information:

The dataset includes 244 instances that regroup a data of two regions of Algeria,namely the Bejaia region located in the northeast of Algeria and the Sidi Bel-abbes region located in the northwest of Algeria.

122 instances for each region.

The period from June 2012 to September 2012. The dataset includes 11 attribues and 1 output attribue (class) The 244 instances have been classified into â€˜fireâ€™ (138 classes) and â€˜not fireâ€™ (106 classes) classes.

Attribute Information:

1.Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012) Weather data observations 2.Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42 3.RH : Relative Humidity in %: 21 to 90 4.Ws :Wind speed in km/h: 6 to 29 5.Rain: total day in mm: 0 to 16.8 FWI Components 6.Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5 7.Duff Moisture Code (DMC) index from the FWI system: 1.1

to 65.9 8.Drought Code (DC) index from the FWI system: 7 to 220.4 9.Initial Spread Index (ISI) index from the FWI system: 0 to 18.5 10.Buildup Index (BUI) index from the FWI system: 1.1 to 68 11.Fire Weather Index (FWI) Index: 0 to 31.1 12.Classes: two classes, namely â€œFireâ€ and â€œnot Fireâ€

2.1 Importing data and required packages

In [578]:

```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import scipy.stats as stats

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

2.2 Import the dataset

In [579]:

```
df = pd.read_csv(r"C:\Users\DHARAVATH RAMDAS\Downloads\Algerian_forest_fires_dataset_UPDATE
df
```

Out[579]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Class
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not f
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not f
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not f
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not f
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not f
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	f
242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not f
243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not f
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not f
245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not f

246 rows × 14 columns

2.3 we have to remove the unnecessary rows form dataset after observation

In [580]:

```
df[121:126]
```

Out[580]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FW
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
123	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FW
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2
125	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2

Observation: see the 122 and 123 row it contains the nan values and categorical values, so we have to remove the 122 and 123 rows because it is different from the data

2.4 Remove unnecessary rows form dataset

In [581]:

```
df.drop(index=[122,123],inplace=True)

# it will give index gap so we reset index
df.reset_index(inplace=True)
df
```

Out[581]:

	index	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI
0	0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5
1	1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4
2	2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1
3	3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0
4	4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5
240	242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0
241	243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2
242	244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7
243	245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5

244 rows × 15 columns

In [582]:

```
# we have to drop the index column

df.drop('index',axis=1,inplace=True)
```

In [583]:

```
# Check the data
df.loc[122:126]
```

Out[583]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classe
122	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	not fi
123	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	not fi
124	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	not fi
125	04	06	2012	30	64	14	0	79.4	5.2	15.4	2.2	5.6	1	not fi
126	05	06	2012	32	60	14	0.2	77.1	6	17.6	1.8	6.5	0.9	not fi

Observation: we sucessfully removed the nan values

2.5 adding new feature named 'Region' in a dataset where 'bejaia'=0 and 'sidi-bel abbes'=1

In [584]:

```
df.shape
```

Out[584]:

```
(244, 14)
```

In [585]:

```
# 1.method

df['Region'] = 0
for i in range(len(df)):
    if i >= 122:
        df['Region'][i] = 1
```

In [586]:

```
# 2.method

# df.loc[:122,'Region'] = 0
# df.loc[122:,'Region'] = 1
```

In [587]:

```
df.head(3)
```

Out[587]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire

Observation: we added new column sucessfully

Observation: see the above bejaia is 122 and sidi-bel abbes is 122 times both are same equal

2.6 Check datatypes of data

In [590]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   day          244 non-null    object
 1   month        244 non-null    object
 2   year         244 non-null    object
 3   Temperature  244 non-null    object
 4    RH          244 non-null    object
 5    Ws          244 non-null    object
 6   Rain         244 non-null    object
 7   FFMC         244 non-null    object
 8   DMC          244 non-null    object
 9   DC           244 non-null    object
 10  ISI          244 non-null    object
 11  BUI          244 non-null    object
 12  FWI          244 non-null    object
 13  Classes      243 non-null    object
 14  Region       244 non-null    int64 
dtypes: int64(1), object(14)
memory usage: 28.7+ KB
```

2.7 describe is used for stats analysis

In [591]:

```
df.describe(include='all')
```

Out[591]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FW
count	244	244	244	244	244	244	244	244	244	244	244	244	24
unique	31	4	1	19	62	18	39	173	166	198	106	174	12
top	01	07	2012	35	64	14	0	88.9	7.9	8	1.1	3	0.
freq	8	62	244	29	10	43	133	8	5	5	8	5	1
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

Observation: see the describe it is given all in nan values so we have to do some data cleaning operation

3.Data cleaning

3.1 see the columns

In [592]:

```
df.columns
```

Out[592]:

```
Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain ', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes  ', 'Region'],
      dtype='object')
```

observation: see the some columns contain some extra spaces so we have to remove it

3.2 Stripping the names of columns

In [593]:

```
df.columns = [col.strip() for col in df.columns]
df.columns
```

Out[593]:

```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'Region'],
      dtype='object')
```

observation : we removed all extra spaces

3.3 see the classes

In [594]:

```
df['Classes'].unique()
```

Out[594]:

```
array(['not fire   ', 'fire   ', 'fire', 'fire ', 'not fire', 'not fire ',
       'not fire     ', nan, 'not fire    '], dtype=object)
```

3.4 Stripping the classes feature column

In [595]:

```
df['Classes'] = df.Classes.str.strip()
```

In [596]:

```
### check classes
df['Classes'].unique()
```

Out[596]:

```
array(['not fire', 'fire', nan], dtype=object)
```

observation: see we got nan values so we have to identify it and set the correct

In [597]:

```
df.iloc[165]
```

Out[597]:

```
day                14
month              07
year             2012
Temperature        37
RH                 37
Ws                 18
Rain              0.2
FFMC             88.9
DMC              12.9
DC             14.6 9
ISI              12.5
BUI              10.4
FWI             fire
Classes          NaN
Region             1
Name: 165, dtype: object
```

observation: see some miss match positions we have to reset it

In [598]:

```
df.at[165,'DC'] = 14.6
df.at[165,'ISI'] = 9
df.at[165,'BUI'] = 12.5
df.at[165,'FWI'] = 10.4
df.at[165,'Classes'] = 'fire'
```

In [599]:

```
df.loc[165]
```

Out[599]:

```
day              14
month            07
year           2012
Temperature      37
RH               37
Ws               18
Rain            0.2
FFMC           88.9
DMC            12.9
DC             14.6
ISI               9
BUI            12.5
FWI            10.4
Classes        fire
Region            1
Name: 165, dtype: object
```

3.5 Changing the classes where fire : 1 and not fire :0

In [600]:

```
df['Classes'] = df['Classes'].replace('not fire','0')
df['Classes'] = df['Classes'].replace('fire','1')
```

3.6 Check The Null Values

In [601]:

```
df.isnull().sum().sum()
```

Out[601]:

```
0
```

3.7 Checking the datatype of each columns

In [602]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   day          244 non-null    object
 1   month        244 non-null    object
 2   year         244 non-null    object
 3   Temperature  244 non-null    object
 4   RH           244 non-null    object
 5   Ws           244 non-null    object
 6   Rain         244 non-null    object
 7   FFMC         244 non-null    object
 8   DMC          244 non-null    object
 9   DC           244 non-null    object
 10  ISI          244 non-null    object
 11  BUI          244 non-null    object
 12  FWI          244 non-null    object
 13  Classes      244 non-null    object
 14  Region       244 non-null    int64 
dtypes: int64(1), object(14)
memory usage: 28.7+ KB
```

observation: see that day is object so its wrong data type so we have change its datatypes

3.8 Changing the datatype of the columns

In [603]:

```
df=df.astype({'day':'int','month':'int','year':'int','Temperature':'int', 'RH':'int', 'Ws':
```

In [604]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          244 non-null    int32  
 1   month        244 non-null    int32  
 2   year         244 non-null    int32  
 3   Temperature  244 non-null    int32  
 4   RH           244 non-null    int32  
 5   Ws           244 non-null    int32  
 6   Rain         244 non-null    float64
 7   FFMC         244 non-null    float64
 8   DMC          244 non-null    float64
 9   DC           244 non-null    float64
 10  ISI          244 non-null    float64
 11  BUI          244 non-null    float64
 12  FWI          244 non-null    float64
 13  Classes      244 non-null    int32  
 14  Region       244 non-null    int64  
dtypes: float64(7), int32(7), int64(1)
memory usage: 22.0 KB
```

3.9 we have to drop the year because year is only same to all 2012

In [606]:

```
df.drop(['year'],axis=1,inplace=True)
```

In [607]:

```
df.head()
```

Out[607]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Regic
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

4.Exploring the data

4.1 Shape of the dataset

In [608]:

```
df.shape
```

Out[608]:

```
(244, 14)
```

observation: finally we got 244 rows and 14 columns

4.2 Columns of dataset

In [609]:

```
df.columns
```

Out[609]:

```
Index(['day', 'month', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'D
C',
       'ISI', 'BUI', 'FWI', 'Classes', 'Region'],
      dtype='object')
```

4.3 Checking nan values in dataset

In [610]:

```
df.isnull().sum()
```

Out[610]:

```
day            0
month          0
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        0
Region         0
dtype: int64
```

observation: we dont have any null values

4.4 unique value of classes feature

In [611]:

```
df['Classes'].unique()
```

Out[611]:

```
array([0, 1])
```

4.5 Balanced data or not balanced data

In [612]:

```
df['Classes'].value_counts()
```

Out[612]:

```
1    138
0    106
Name: Classes, dtype: int64
```

observation : it is small difference in your balanced data so it is slightely balanced data

4.6 Describe is used for statistics analysis

In [613]:

```
df.describe()
```

Out[613]:

	day	month	Temperature	RH	Ws	Rain	FFMC	2
count	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	
mean	15.754098	7.500000	32.172131	61.938525	15.504098	0.760656	77.887705	
std	8.825059	1.112961	3.633843	14.884200	2.810178	1.999406	14.337571	
min	1.000000	6.000000	22.000000	21.000000	6.000000	0.000000	28.600000	
25%	8.000000	7.000000	30.000000	52.000000	14.000000	0.000000	72.075000	
50%	16.000000	7.500000	32.000000	63.000000	15.000000	0.000000	83.500000	
75%	23.000000	8.000000	35.000000	73.250000	17.000000	0.500000	88.300000	
max	31.000000	9.000000	42.000000	90.000000	29.000000	16.800000	96.000000	

4.7 transform

In [615]:

```
df.describe().T
```

Out[615]:

	count	mean	std	min	25%	50%	75%	max
day	244.0	15.754098	8.825059	1.0	8.000	16.00	23.000	31.0
month	244.0	7.500000	1.112961	6.0	7.000	7.50	8.000	9.0
Temperature	244.0	32.172131	3.633843	22.0	30.000	32.00	35.000	42.0
RH	244.0	61.938525	14.884200	21.0	52.000	63.00	73.250	90.0
Ws	244.0	15.504098	2.810178	6.0	14.000	15.00	17.000	29.0
Rain	244.0	0.760656	1.999406	0.0	0.000	0.00	0.500	16.8
FFMC	244.0	77.887705	14.337571	28.6	72.075	83.50	88.300	96.0
DMC	244.0	14.673361	12.368039	0.7	5.800	11.30	20.750	65.9
DC	244.0	49.288115	47.619662	6.9	13.275	33.10	68.150	220.4
ISI	244.0	4.759836	4.154628	0.0	1.400	3.50	7.300	19.0
BUI	244.0	16.673361	14.201648	1.1	6.000	12.45	22.525	68.0
FWI	244.0	7.049180	7.428366	0.0	0.700	4.45	11.375	31.1
Classes	244.0	0.565574	0.496700	0.0	0.000	1.00	1.000	1.0
Region	244.0	0.500000	0.501028	0.0	0.000	0.50	1.000	1.0

5.Analysis:

5.1.Numerical and Categorical Columns

Numerical Features

In [616]:

```
# we get numerical feature from dataset so create numerical feature
```

In [617]:

```
num_fea = [fea for fea in df.columns if df[fea].dtype != 'O']

print("numerical features ",num_fea,"length of numerical feature",len(num_fea))
```

```
numerical features  ['day', 'month', 'Temperature', 'RH', 'Ws', 'Rain', 'FFM
C', 'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'Region'] length of numeric
al feature 14
```

Categorical Features

In [618]:

```
cat_fea = [fea for fea in df.columns if df[fea].dtype == 'O']

print("categorical features",cat_fea,"length of categorical feature", len(cat_fea))
```

```
categorical features [] length of categorical feature 0
```

5.2.Univariate Analysis:

> the term univariate analysis refers to the analysis of one variable prefix"uni" means "one" the purpose of univariate analysis is to understand analysis is to undestand the distribution of values for a single variable

In [619]:

```
df.var()
```

Out[619]:

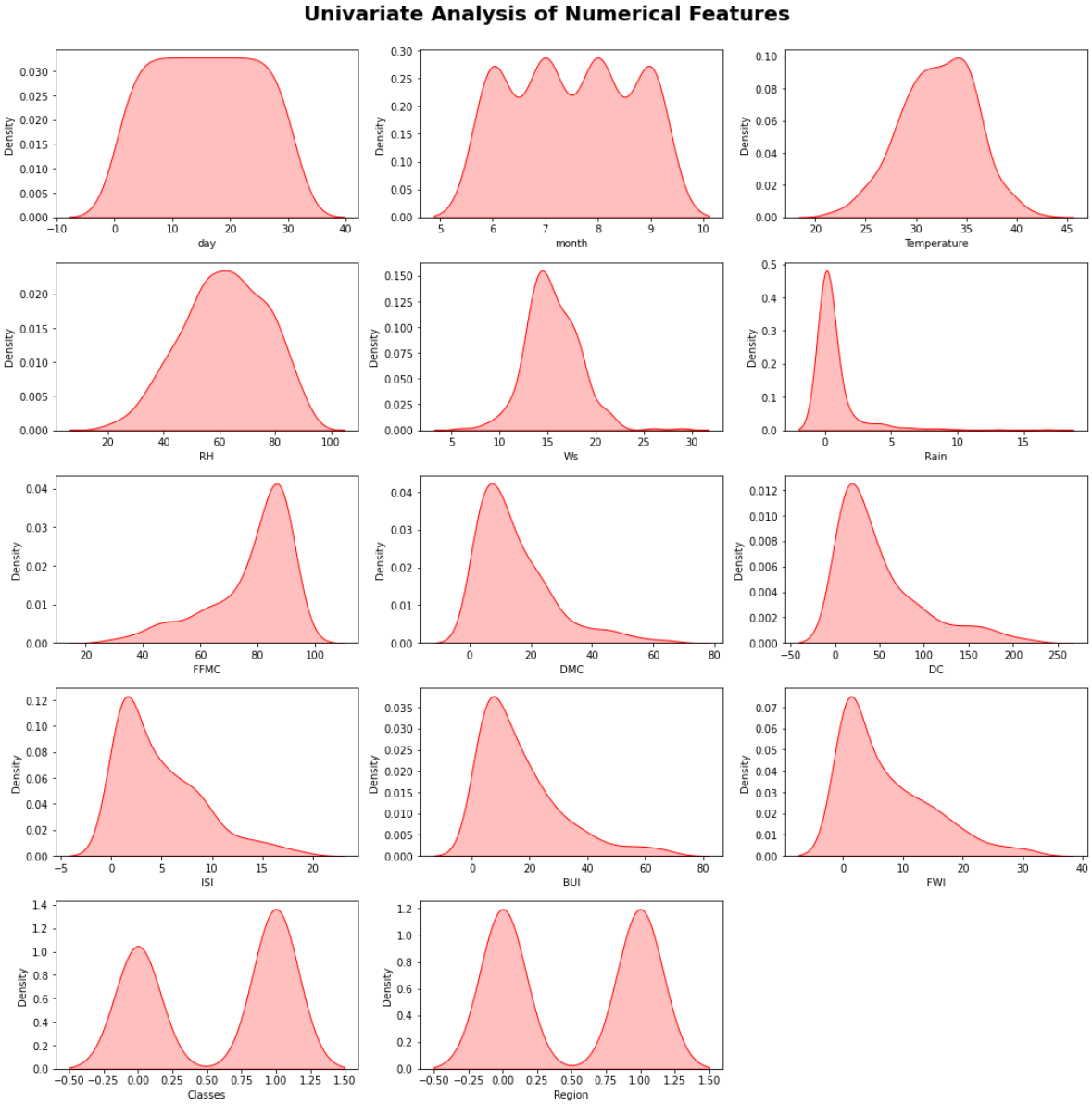
```
day              77.881670
month             1.238683
Temperature      13.204817
RH              221.539415
Ws                7.897102
Rain              3.997623
FFMC            205.565939
DMC             152.968382
DC             2267.632245
ISI              17.260932
BUI             201.686818
FWI              55.180617
Classes           0.246711
Region            0.251029
dtype: float64
```

5.3.Numerical Feature Analysis

In [620]:

```
plt.figure(figsize = (15,15))
plt.suptitle('Univariate Analysis of Numerical Features',fontsize=20,fontweight='bold',alph

for i in range(0,len(num_fea)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=df[num_fea[i]],shade=True,color='r')
    plt.xlabel(num_fea[i])
    plt.tight_layout()
```



Observation: see that some are right skew , some left skew and some are normal distribution

In [621]:

```
df.head(3)
```

Out[621]:

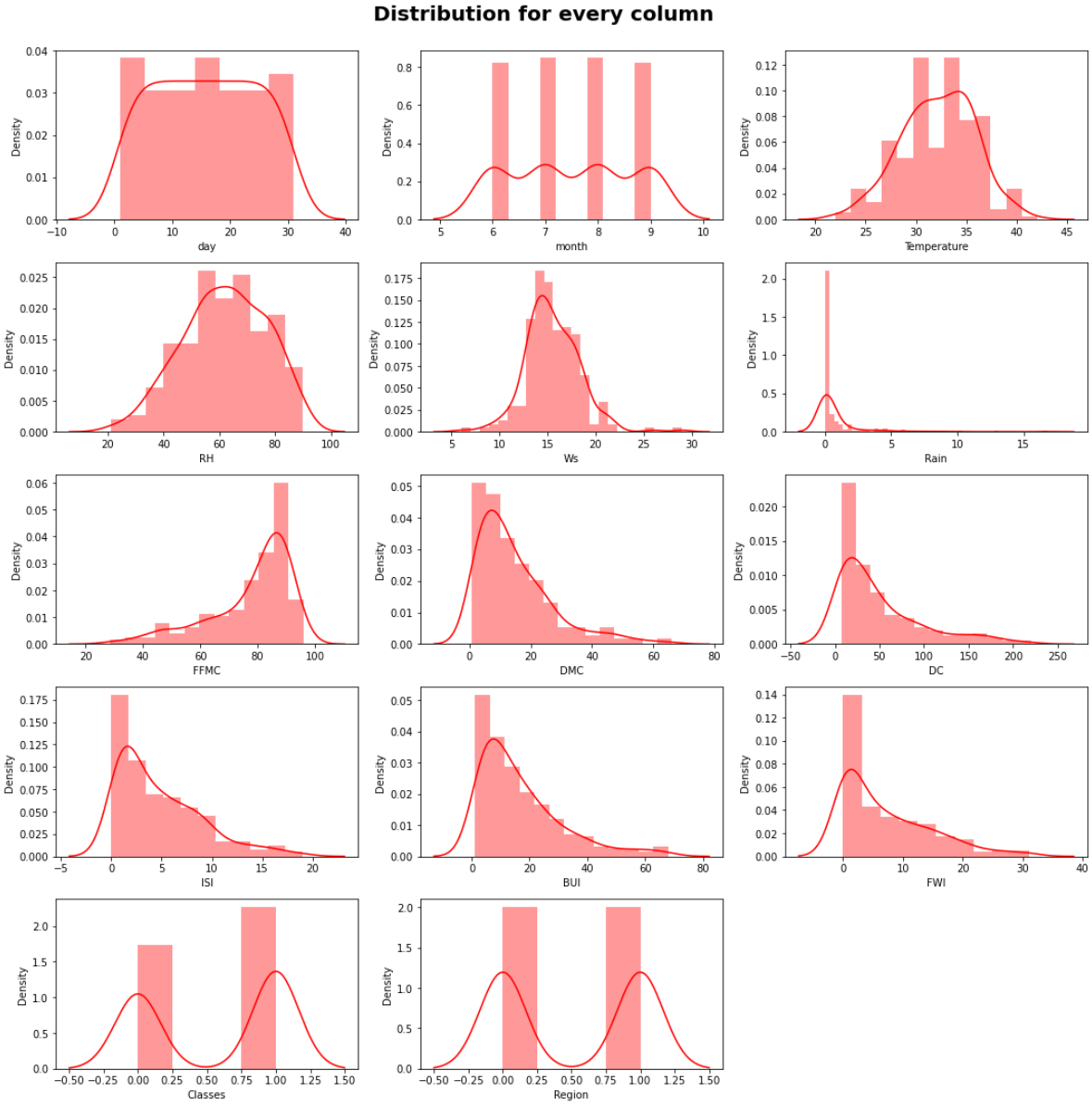
	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Regior
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	(
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	(
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	(

5.4 Distribution for every column

In [622]:

```
plt.figure(figsize = (15,15))
plt.suptitle('Distribution for every column ',fontsize=20,fontweight='bold',alpha=1,y=1)

for i in range(0,len(num_fea)):
    plt.subplot(5,3,i+1)
    sns.distplot(x=df[num_fea[i]],color='r')
    plt.xlabel(num_fea[i])
    plt.tight_layout()
```

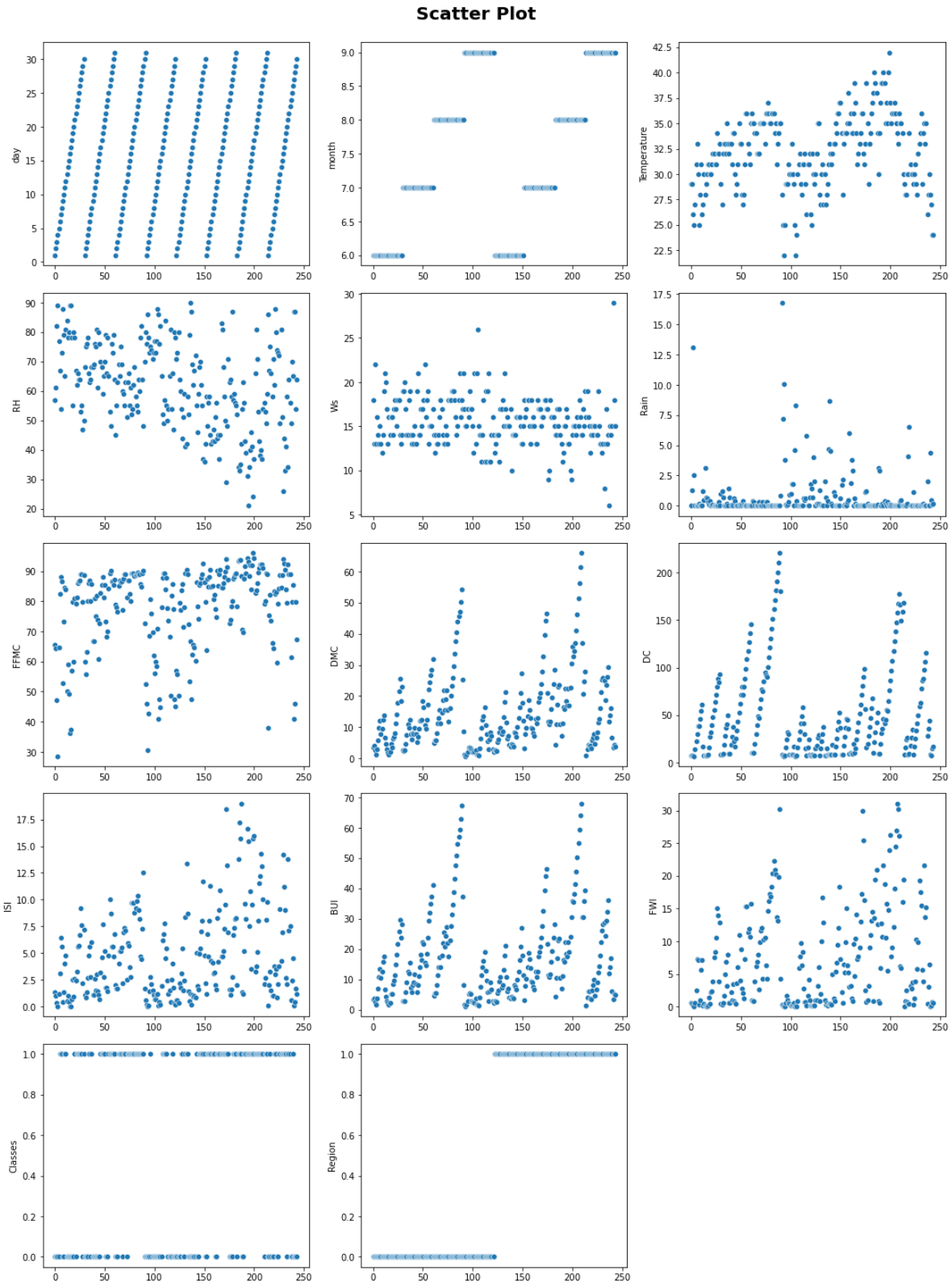


5.5 Scatter plot to see the trends in each numerical columns

In [624]:

```
plt.figure(figsize=(15,20))
plt.suptitle("Scatter Plot",fontsize=20,fontweight='bold',alpha=1,y=1)

for i in range(0,len(num_fea)):
    plt.subplot(5,3,i+1)
    sns.scatterplot(x=df.index,y=num_fea[i],data=df)
    plt.tight_layout()
```

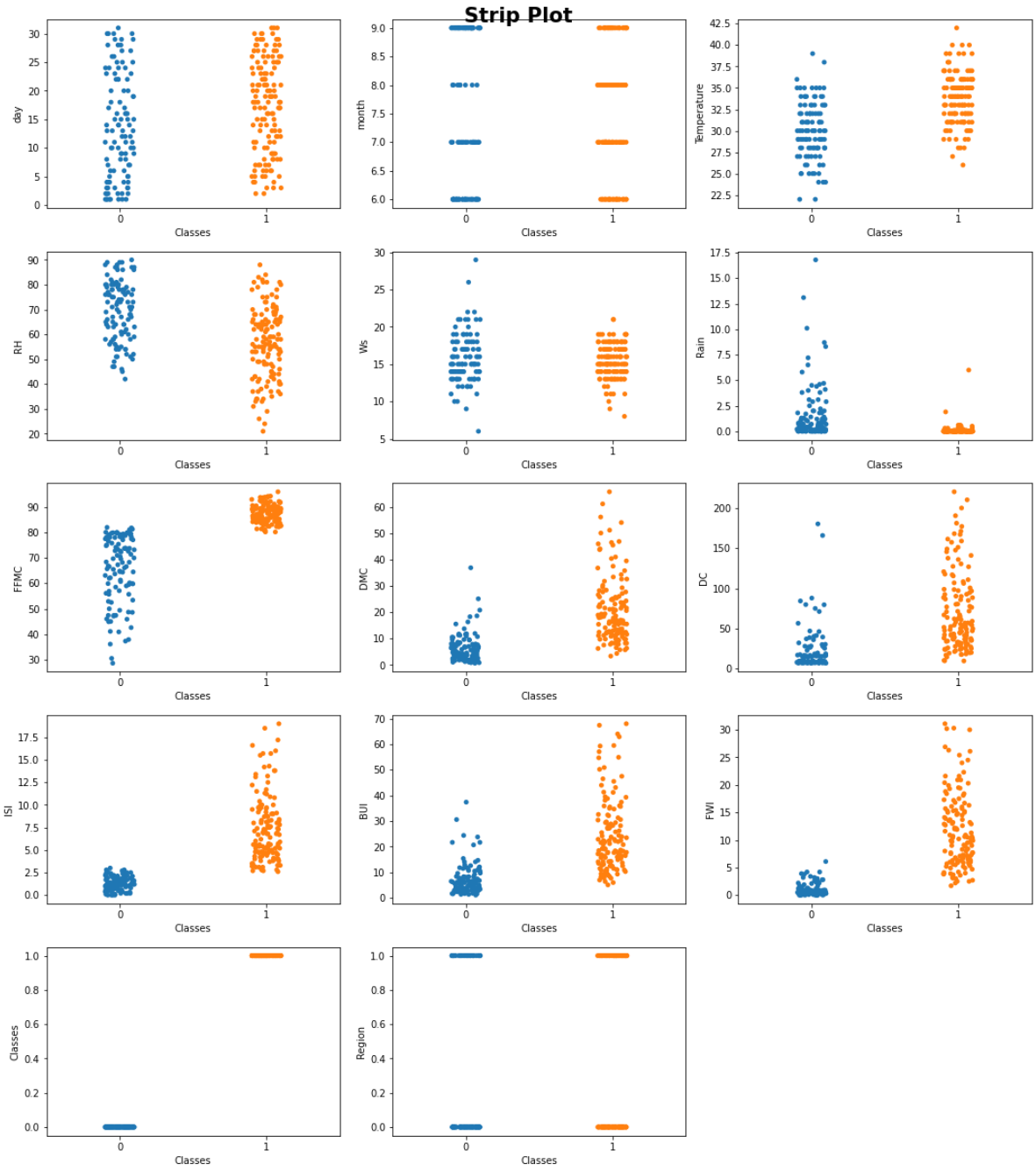


5.6 Strip Plot

In [625]:

```
plt.figure (figsize=(15,20))
plt.suptitle('Strip Plot',fontsize=21,fontweight='bold')

for i in range(len(num_fea)):
    plt.subplot(6,3,i+1)
    sns.stripplot(x='Classes',y=num_fea[i],data=df)
    plt.tight_layout()
```



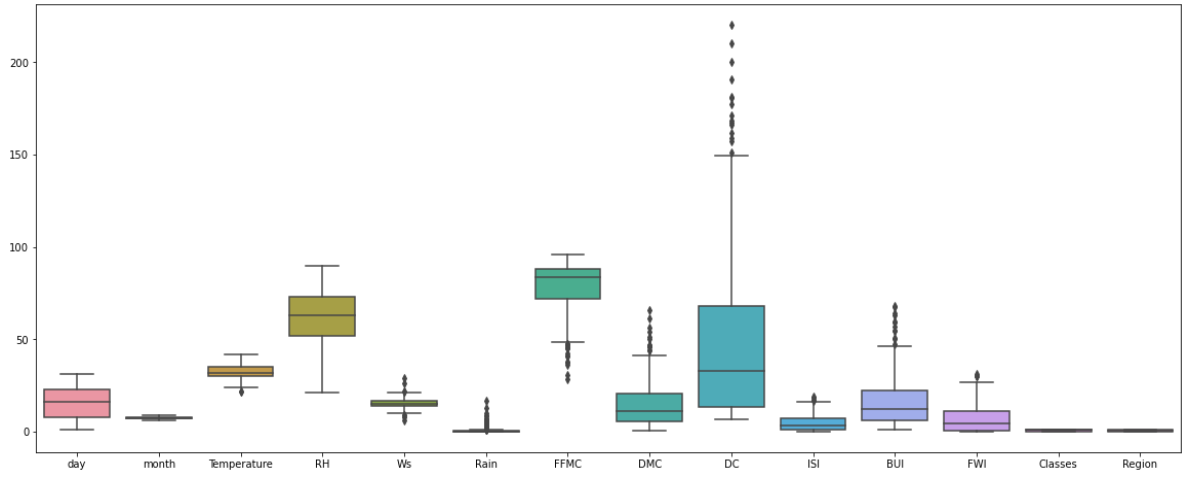
5.7 Box plot:

In [626]:

```
plt.figure(figsize=(20,8))
sns.boxplot(data=df)
```

Out[626]:

<AxesSubplot:>



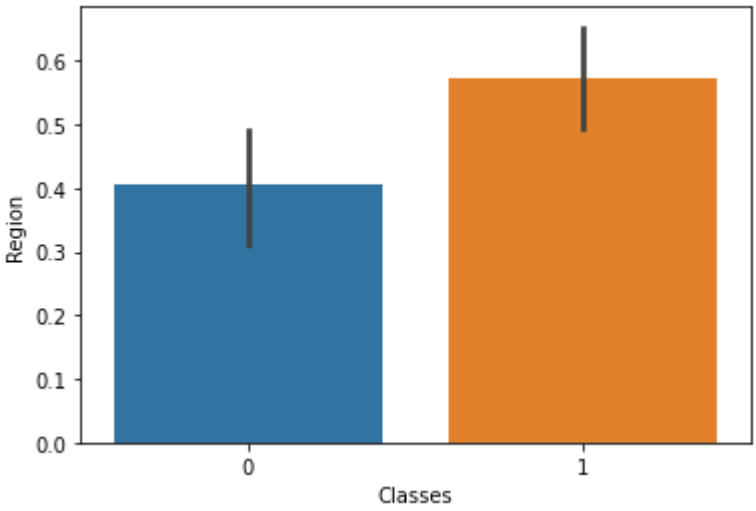
5.8 Barplot:

In [627]:

```
sns.barplot(x="Classes",y='Region',data=df)
```

Out[627]:

<AxesSubplot:xlabel='Classes', ylabel='Region'>



observation: 1.as comparision fire is more in region 2.not fire is less in region

5.9 Corr Visualization

In [628]:

```
df.corr()
```

Out[628]:

	day	month	Temperature	RH	Ws	Rain	FFM
day	1.000000e+00	2.232788e-17	0.095772	-0.074209	0.047001	-0.112265	0.2240
month	2.232788e-17	1.000000e+00	-0.059017	-0.037884	-0.041447	0.035322	0.0155
Temperature	9.577222e-02	-5.901677e-02	1.000000	-0.654443	-0.278132	-0.326786	0.6774
RH	-7.420934e-02	-3.788419e-02	-0.654443	1.000000	0.236084	0.222968	-0.6456
Ws	4.700086e-02	-4.144673e-02	-0.278132	0.236084	1.000000	0.170169	-0.1632
Rain	-1.122654e-01	3.532207e-02	-0.326786	0.222968	0.170169	1.000000	-0.5440
FFMC	2.240321e-01	1.557668e-02	0.677491	-0.645658	-0.163255	-0.544045	1.0000
DMC	4.915710e-01	6.817778e-02	0.483105	-0.405133	-0.001246	-0.288548	0.6023
DC	5.279285e-01	1.276719e-01	0.370498	-0.220330	0.076245	-0.296804	0.5039
ISI	1.793008e-01	6.354476e-02	0.605971	-0.688268	0.012245	-0.347862	0.7407
BUI	5.172239e-01	8.556743e-02	0.456415	-0.349685	0.030303	-0.299409	0.5902
FWI	3.502343e-01	8.173226e-02	0.566839	-0.580457	0.033957	-0.324755	0.6914
Classes	2.017844e-01	2.233266e-02	0.518119	-0.435023	-0.066529	-0.379449	0.7701
Region	4.662229e-16	-9.586232e-17	0.273496	-0.406424	-0.176829	-0.041080	0.2246

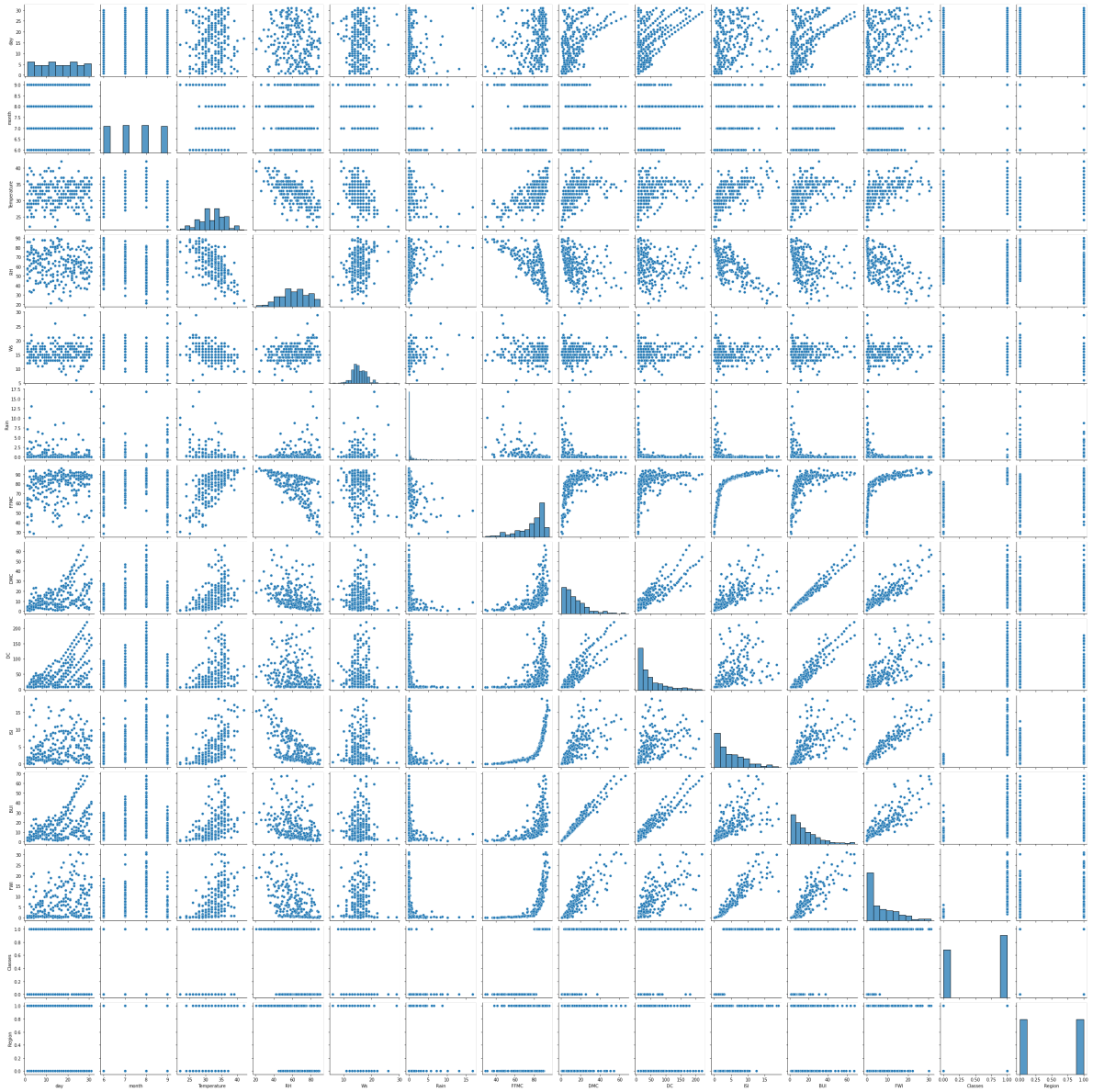
5.9.1 Pairplot:

In [629]:

```
sns.pairplot(df)
```

Out[629]:

```
<seaborn.axisgrid.PairGrid at 0x2660814ffa0>
```



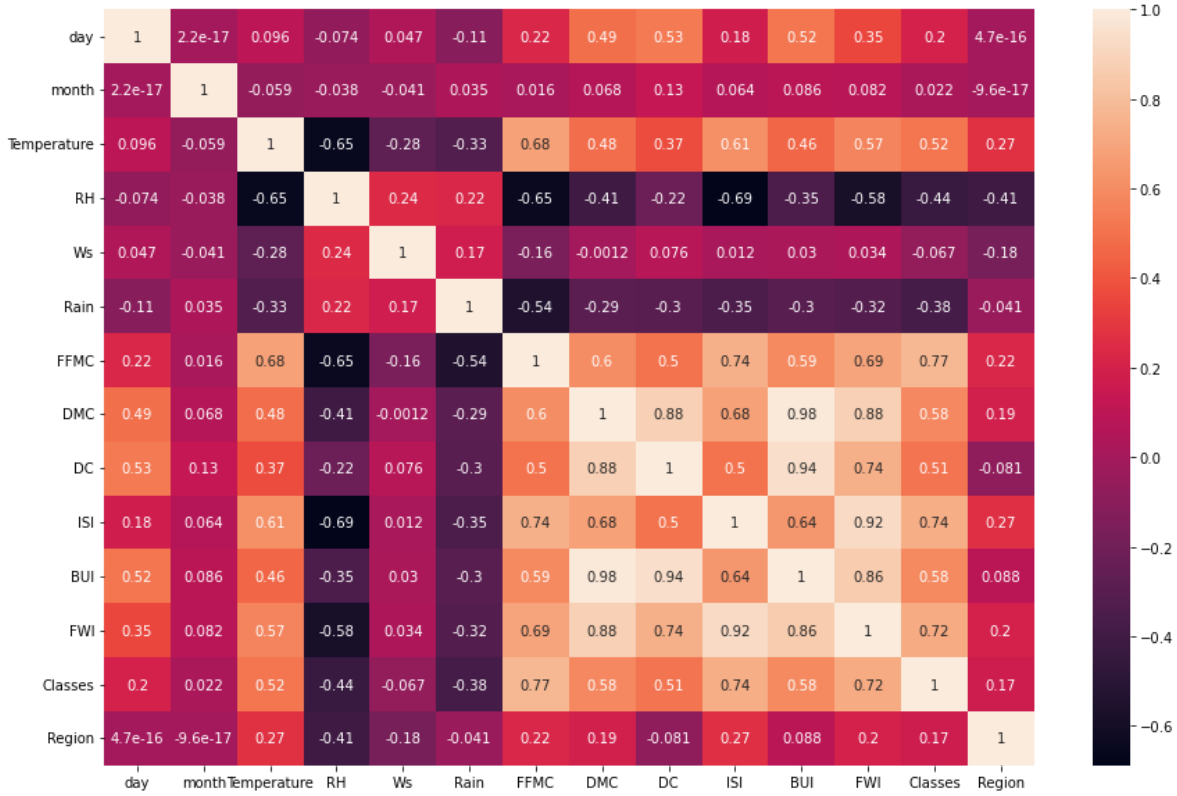
5.9.2 Heatmap:

In [630]:

```
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True)
```

Out[630]:

<AxesSubplot:>



6.Plot data in logistic Regression

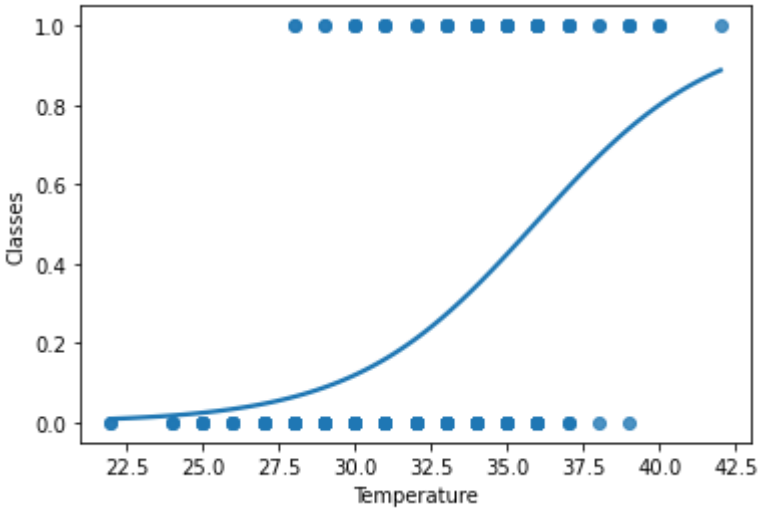
Day vs Classes

In [754]:

```
sns.regplot(x='Temperature',y='Classes',data=df,logistic=True,ci=None)
```

Out[754]:

```
<AxesSubplot:xlabel='Temperature', ylabel='Classes'>
```



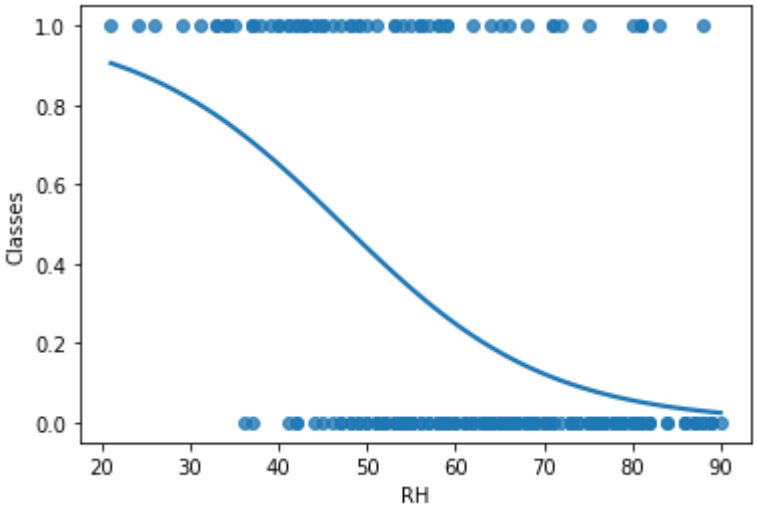
RH vs Classes

In [755]:

```
sns.regplot(x='RH',y='Classes',data=df,logistic=True,ci=None)
```

Out[755]:

```
<AxesSubplot:xlabel='RH', ylabel='Classes'>
```



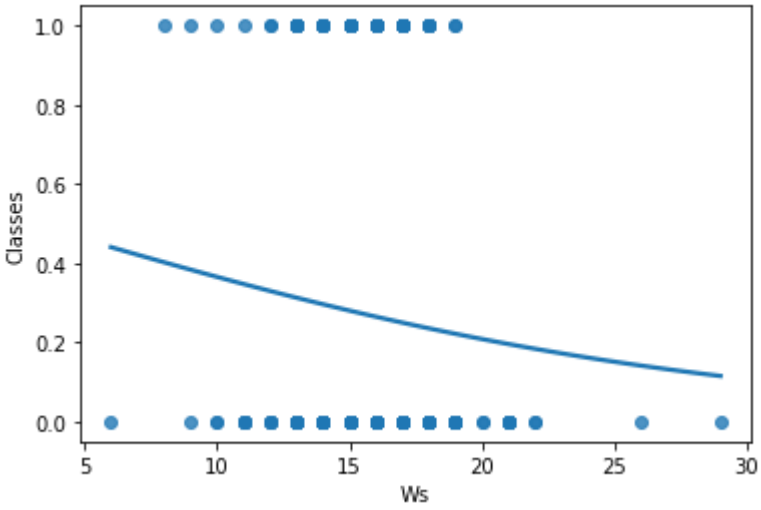
Ws vs Classes

In [756]:

```
sns.regplot(x='Ws',y='Classes',data=df,logistic=True,ci=None)
```

Out[756]:

```
<AxesSubplot:xlabel='Ws', ylabel='Classes'>
```



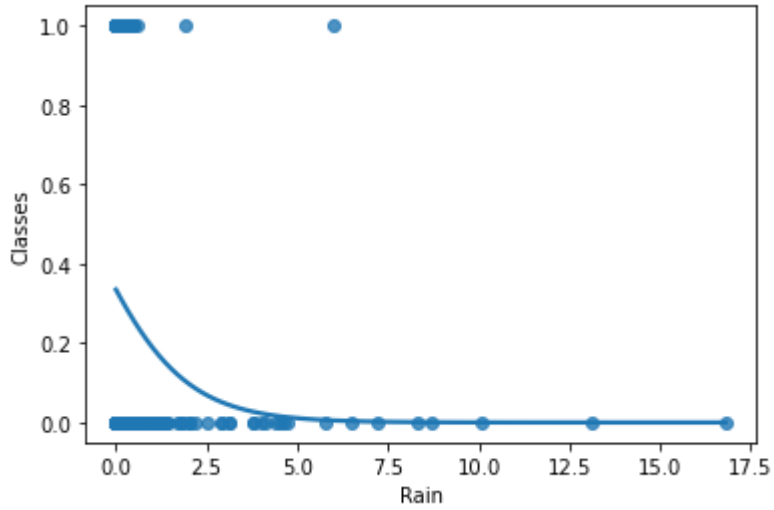
Rain vs Classes

In [757]:

```
sns.regplot(x='Rain',y='Classes',data=df,logistic=True,ci=None)
```

Out[757]:

```
<AxesSubplot:xlabel='Rain', ylabel='Classes'>
```



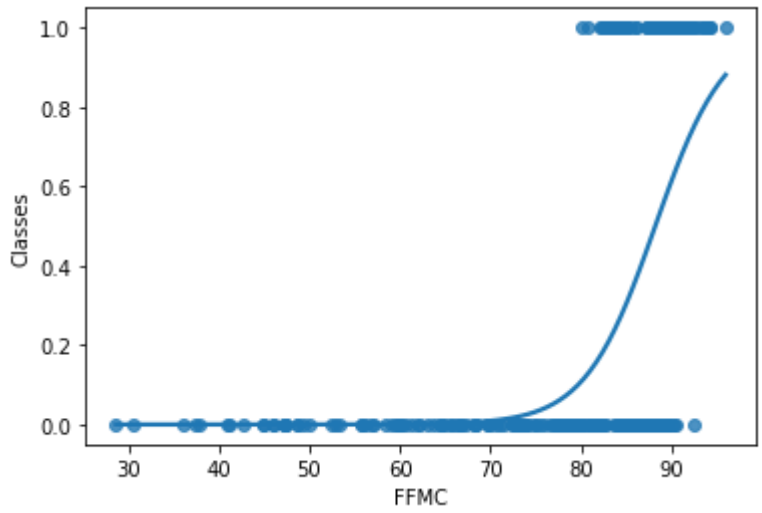
FFMC vs Classes

In [758]:

```
sns.regplot(x='FFMC',y='Classes',data=df,logistic=True,ci=None)
```

Out[758]:

```
<AxesSubplot:xlabel='FFMC', ylabel='Classes'>
```



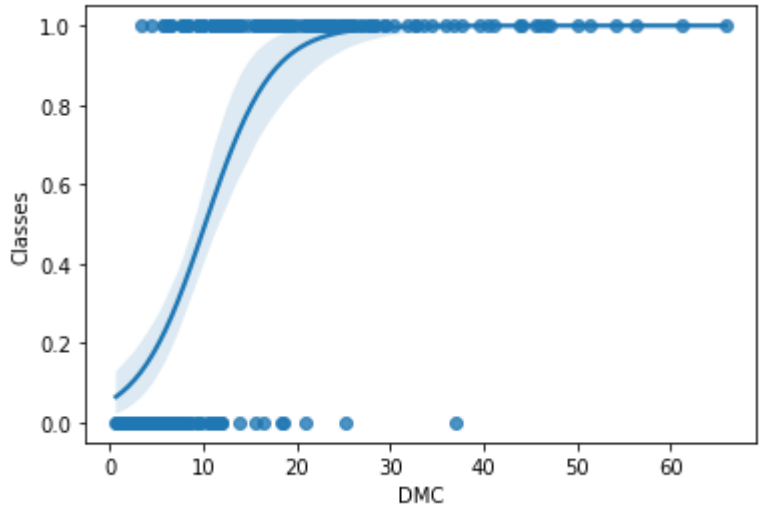
DMC vs Classes

In [637]:

```
sns.regplot(x='DMC',y='Classes',data=df,logistic=True)
```

Out[637]:

```
<AxesSubplot:xlabel='DMC', ylabel='Classes'>
```



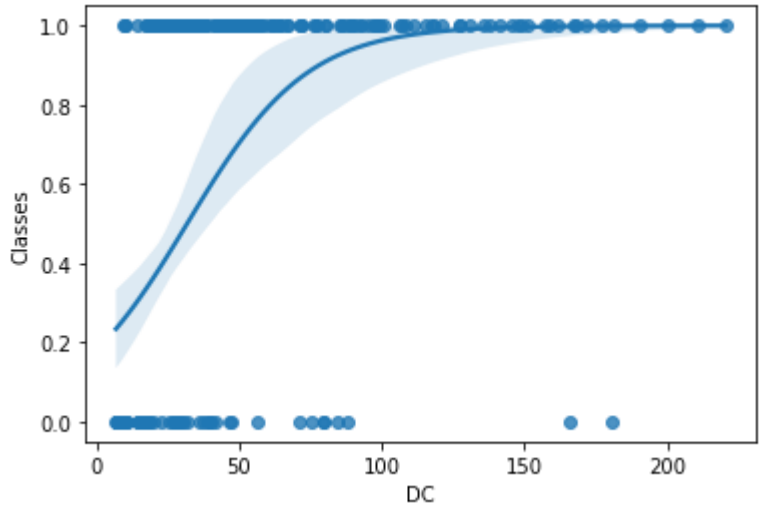
DC vs Classes

In [638]:

```
sns.regplot(x='DC',y='Classes',data=df,logistic=True)
```

Out[638]:

```
<AxesSubplot:xlabel='DC', ylabel='Classes'>
```



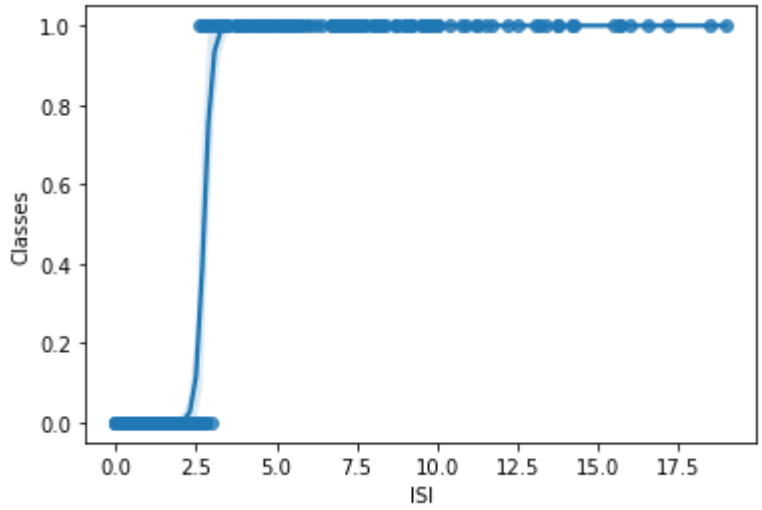
ISI vs Classes

In [639]:

```
sns.regplot(x='ISI',y='Classes',data=df,logistic=True)
```

Out[639]:

```
<AxesSubplot:xlabel='ISI', ylabel='Classes'>
```



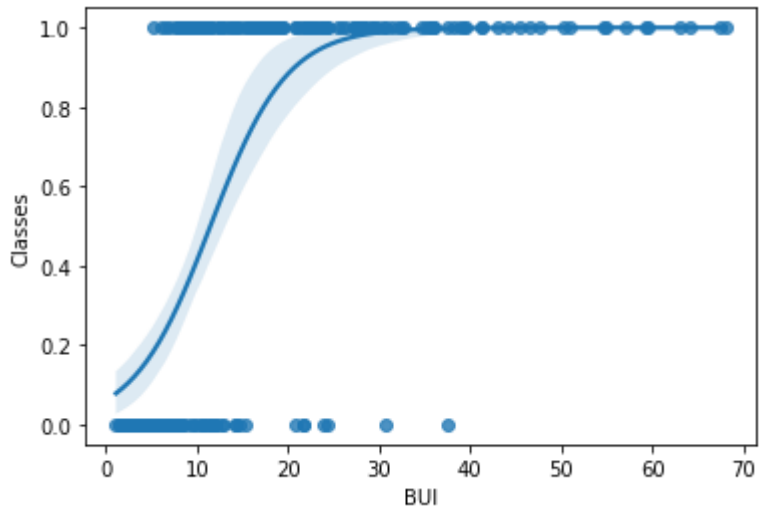
BUI vs Classes

In [640]:

```
sns.regplot(x='BUI',y='Classes',data=df,logistic=True)
```

Out[640]:

```
<AxesSubplot:xlabel='BUI', ylabel='Classes'>
```



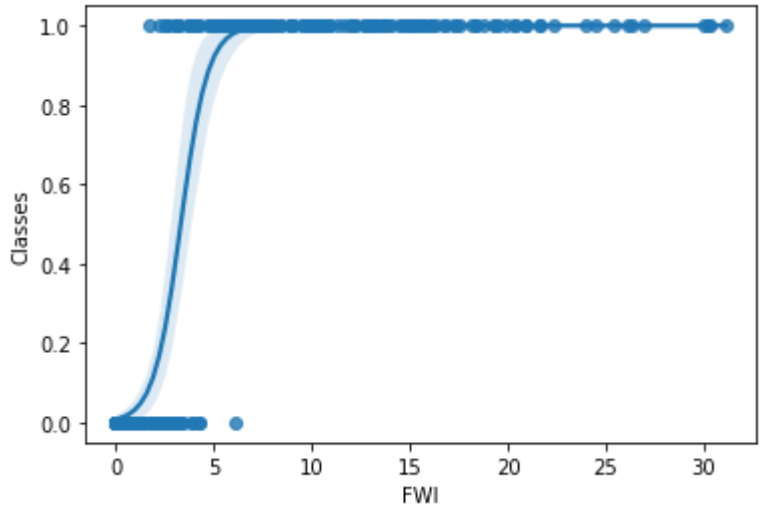
FWI vs Classes

In [641]:

```
sns.regplot(x='FWI',y='Classes',data=df,logistic=True)
```

Out[641]:

```
<AxesSubplot:xlabel='FWI', ylabel='Classes'>
```



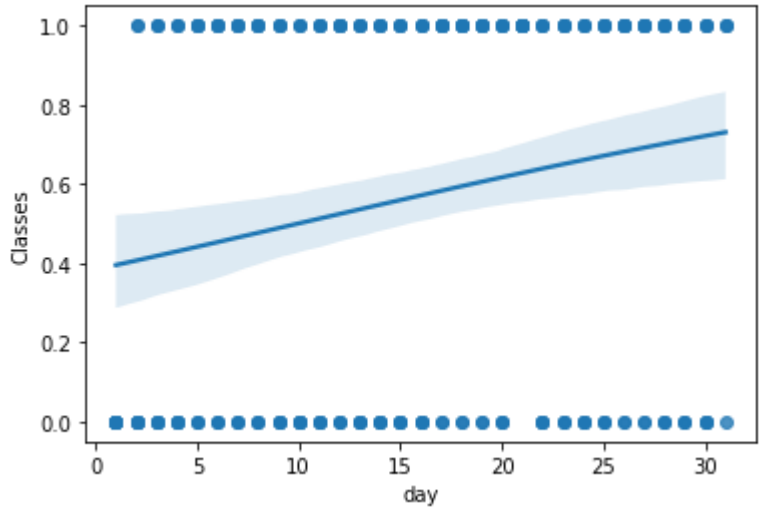
Region vs Classes

In [642]:

```
sns.regplot(x='day',y='Classes',data=df,logistic=True)
```

Out[642]:

```
<AxesSubplot:xlabel='day', ylabel='Classes'>
```



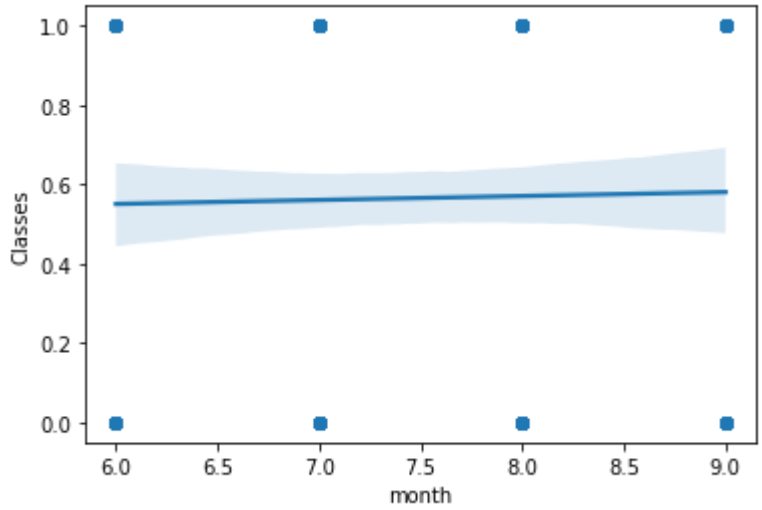
Month vs Classes

In [643]:

```
sns.regplot(x='month',y='Classes',data=df,logistic=True)
```

Out[643]:

```
<AxesSubplot:xlabel='month', ylabel='Classes'>
```



In [644]:

```
df.head(2)
```

Out[644]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Regior
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	(
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	(

In [ ]:

```

```

7.Model

7.1 Machine Learning libraries

In [645]:

```
### Machine Learning libraries
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report


### To save the model
import pickle
```

7.2 Independent and Dependent Feature Seperation

Independent feature

In [646]:

```
X = df.drop('Classes',axis=1)
```

Dependent feature

In [647]:

```
y = df['Classes']
```

In [648]:

```
# Checking
```

In [649]:

```
X.head(3)
```

Out[649]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0

In [650]:

```
y.head(3)
```

Out[650]:

```
0    0
1    0
2    0
Name: Classes, dtype: int32
```

7.3 Spliting the data into Train and Test split

In [651]:

```
# spliting the data into train test split
# it will return 4 different paremeters
# output feature of x train is y train and x test is y test
# test size = 0.25   if 1000 in 25% of data
# random state
```

In [652]:

```
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=355)
```

In [653]:

```
# checking the data

X_train.head(3)
```

Out[653]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
38	9	7	32	68	14	1.4	66.6	7.7	9.2	1.1	7.4	0.6	0
53	24	7	28	78	16	0.1	70.0	9.6	79.7	1.4	14.7	1.3	0
170	19	7	34	58	16	0.0	88.1	27.8	61.1	7.3	27.7	13.0	1

In [654]:

```
y_train.head()
```

Out[654]:

```
38     0
53     0
170    1
46     1
58     1
Name: Classes, dtype: int32
```

In [655]:

```
X_test.head(3)
```

Out[655]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
96	5	9	29	75	16	0.0	80.8	3.4	24.0	2.8	5.1	1.7	0
132	11	6	31	42	21	0.0	90.6	18.2	30.5	13.4	18.0	16.7	1
121	30	9	25	78	14	1.4	45.0	1.9	7.5	0.2	2.4	0.1	0

In [656]:

```
y_test.head(3)
```

Out[656]:

```
96     1
132    1
121    0
Name: Classes, dtype: int32
```

In [657]:

```
# checking the shape of data

X_train.shape
```

Out[657]:

```
(183, 13)
```

In [658]:

```
y_train.shape
```

Out[658]:

```
(183,)
```

In [659]:

```
X_test.shape
```

Out[659]:

```
(61, 13)
```

In [660]:

```
y_test.shape
```

Out[660]:

```
(61,)
```

7.4 Standardizing or Feature scalling the dataset

In [661]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler
```

Out[661]:

```
StandardScaler
StandardScaler()
```

In [662]:

```
# Apply data
```

In [663]:

```
X_train = scaler.fit_transform(X_train)
```

In [664]:

```
X_test = scaler.transform(X_test)
```

In [665]:

```
#data lekage we dont need to leak the data of test to train data
#avoid datalekage use transform
#example :
#    is eaxm paper is x_train if you get before exam is called parer lekage
# f to f' we convert mean and std in fit and transform
```

In [666]:

```
X_train
```

Out[666]:

```
array([[-0.82473448, -0.43394895, -0.07323543, ..., -0.70181037,
        -0.88736547, -1.07375098],
       [ 0.91806471, -0.43394895, -1.19007573, ..., -0.19950741,
        -0.79463296, -1.07375098],
       [ 0.33713165, -0.43394895,  0.48518472, ...,  0.6950047 ,
         0.75532479,  0.93131463],
       ...,
       [-1.17329431, -0.43394895, -0.07323543, ..., -0.35088638,
        -0.06602034, -1.07375098],
       [ 1.61518438,  0.47884022,  0.76439479, ...,  1.36244836,
        -0.41045539, -1.07375098],
       [-0.82473448, -1.34673811, -1.4692858 , ..., -0.6398826 ,
        -0.71514794,  0.93131463]])
```

In [667]:

```
X_test
```

```
       [-1.14281914e-02,  1.39162938e+00, -6.31655579e-01,
         1.88279537e-01, -5.01405908e-01, -4.17392085e-01,
        -4.28901039e-03, -9.63983119e-01, -7.51091242e-01,
        -6.92710227e-01, -9.21997964e-01, -8.60870467e-01,
        -1.07375098e+00],
       [ 8.01878094e-01, -1.34673811e+00, -7.32354294e-02,
        -1.47960343e-02,  9.66710590e-01, -3.56866926e-01,
         2.26972898e-01, -5.74971673e-01, -8.82138569e-02,
        -3.59991141e-01, -4.19695007e-01, -4.63445403e-01,
        -1.07375098e+00],
       [-9.40921089e-01,  4.78840217e-01,  1.32281494e+00,
        -4.20947177e-01, -1.60249328e+00, -4.17392085e-01,
         6.47449095e-01, -3.41564805e-01, -6.57874109e-01,
         9.15561886e-02, -4.54099319e-01, -1.85247858e-01,
         9.31314629e-01],
       [ 9.18064706e-01,  1.39162938e+00, -1.74849588e+00,
        -8.94790177e-01, -3.43763890e+00,  7.93111110e-01,
        -1.18162236e+00, -2.87103202e-01, -4.94226255e-01,
        -1.00166366e+00, -3.92171558e-01, -9.13860475e-01,
         9.31314629e-01].
```

7.5 Model Training

In [668]:

```
from sklearn.linear_model import LogisticRegression

# create object
logistic_regre = LogisticRegression()
logistic_regre
```

Out[668]:

```
LogisticRegression
LogisticRegression()
```

In [669]:

```
logistic_regre.fit(X_train,y_train)
```

Out[669]:

```
LogisticRegression
LogisticRegression()
```

7.6 Prediction

In [670]:

```
logistic_regre_pred = logistic_regre.predict(X_test)
logistic_regre_pred
```

Out[670]:

```
array([0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1])
```

7.7 Performance Metrics

7.7.1.Confusion Matrix

In [671]:

```
from sklearn.metrics import confusion_matrix

confusion_mat = confusion_matrix(y_test,logistic_regre_pred)
confusion_mat
```

Out[671]:

```
array([[27,  0],
       [ 1, 33]], dtype=int64)
```

In [672]:

```
truly_positive=confusion_mat[0][0]
falsely_positive=confusion_mat[0][1]
falsely_negative=confusion_mat[1][0]
truly_negative=confusion_mat[1][1]
```

7.7.2 Accuracy Score

In [673]:

```
accuracy= round(accuracy_score(y_test,logistic_regre_pred),4)
accuracy
```

Out[673]:

```
0.9836
```

In [674]:

```
### manual caluculation for accuracy

accuracy_manual=round(((truly_positive+truly_negative)/(truly_positive+falsely_positive+fal
print("Accuracy of our model is {}".format(accuracy_manual))
```

```
Accuracy of our model is 0.9836
```

7.7.3 Precision Score

In [675]:

```
precision_manual= round(truly_positive/(truly_positive+falsely_positive),4)
print("Precision of our model is : ",precision_manual)
```

```
Precision of our model is :  1.0
```

7.7.4 Recall Score

In [676]:

```
recall_manual=round(truly_positive/(truly_positive+falsely_negative),4)
print("Recall of our model is {}".format(recall_manual))
```

```
Recall of our model is 0.9643
```

7.7.5 F-1 Score

Giving equal importance to falsely positive and falsely negative

In [677]:

```
f1_score=2*(precision_manual*recall_manual)/(precision_manual+recall_manual)
print("F-1 Score of our model is {} ".format(round(f1_score,4)))
```

```
F-1 Score of our model is 0.9818
```

7.7.6 Classification Report

In [678]:

```
print(classification_report(y_test, logistic_regre_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        27
           1       1.00      0.97      0.99        34

    accuracy                           0.98        61
   macro avg       0.98      0.99      0.98        61
weighted avg       0.98      0.98      0.98        61
```

7.7.7.Saving the Model

In [679]:

```
### Writing model to a file that will be used while deployment
with open('model_Logistic_regression_algerian_ff.sav','wb') as f:
    pickle.dump(logistic_regre,f)
```

In [680]:

```
df.head()
```

Out[680]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Regic
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

as per your question

in balanced data i got 98% acccuracy

8. Creating imbalance:

8.1.method to create the imbalance data

In [681]:

```
for i in range(len(df)):
    if i <= 150:
        df['Classes'][i] = int(0)
```

In [682]:

```
df['Classes'].value_counts()
```

Out[682]:

```
0    177
1     67
Name: Classes, dtype: int64
```

In [683]:

```
# copy the classes
dfi = df
```

In [684]:

```
dfi.head()
```

Out[684]:

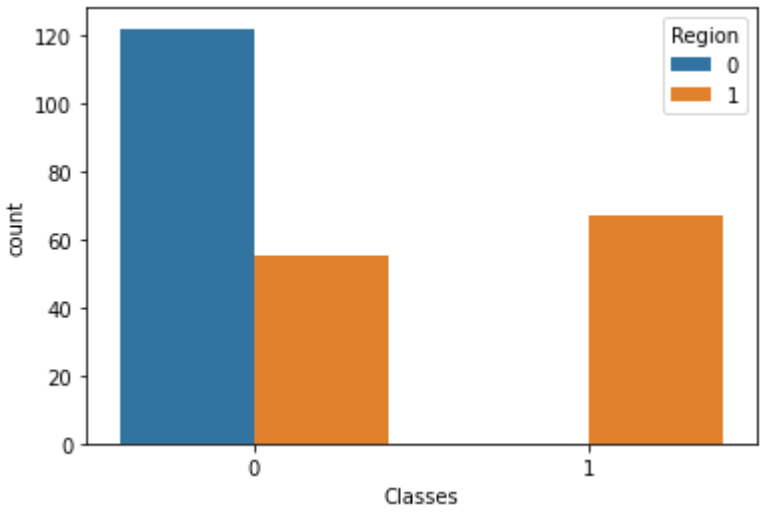
	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Regic
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

In [687]:

```
sns.countplot(data=dfi,x='Classes',hue='Region')
```

Out[687]:

```
<AxesSubplot:xlabel='Classes', ylabel='count'>
```



In [ ]:

```
```

8.2 method to create imbalance data

In [257]:

```
Xi = df.drop('Classes',axis=1)
yi = df['Classes']
```

In [155]:

```
# spliting data into 89 and 11 precent rartion using train test split
```

In [156]:

```
X_traini,X_testi,y_traini,y_testi = train_test_split(Xi,yi,test_size=0.09,random_state=15)
```

In [157]:

```
print(X_traini.shape,y_traini.shape)
```

```
(222, 13) (222,)
```

In [158]:

```
print(X_testi.shape,y_testi.shape)
```

```
(22, 13) (22,)
```

In [159]:

```
### to create imbalance data we use output feature in that we have 1 and 0 so we have to re
```

In [160]:

```
y_traini = y_traini.replace(1,0)
y_traini.head()
```

Out[160]:

```
167    0
107    0
144    0
97     0
120    0
Name: Classes, dtype: int32
```

In [161]:

```
y_test = y_test.replace(0,1)
y_test.head()
```

Out[161]:

```
72     1
149    1
29     1
191    1
10     1
Name: Classes, dtype: int32
```

In [162]:

```
# Adding the to train datasets
```

In [163]:

```
traini = X_traini.join(pd.DataFrame(y_traini))
traini.head()
```

Out[163]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
167	16	7	31	83	17	0.0	84.5	19.4	33.1	4.7	19.2	7.3	1
107	16	9	30	65	14	0.0	78.1	3.2	15.7	1.9	4.2	0.8	0
144	23	6	33	59	16	0.8	74.2	7.0	8.3	1.6	6.7	0.8	1
97	6	9	29	74	19	0.1	75.8	3.6	32.2	2.1	5.6	0.9	0
120	29	9	26	80	16	1.8	47.4	2.9	7.7	0.3	3.0	0.1	0

In [164]:

```
# Adding the test data
```

In [165]:

```
testi = X_testi.join(pd.DataFrame(y_testi))
testi.head()
```

Out[165]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Cla
127	6	6	35	54	11	0.1	83.7	8.4	26.3	3.1	9.3	3.1	1	
222	9	9	30	80	15	0.0	83.1	7.9	34.5	3.5	10.0	3.7	1	
163	12	7	36	44	13	0.0	90.1	12.6	19.4	8.3	12.5	9.6	1	
93	2	9	22	86	15	10.1	30.5	0.7	7.0	0.0	1.1	0.0	0	
183	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1	

In [166]:

```
# We are checking the shape of it
```

In [167]:

```
traini.shape
```

Out[167]:

```
(222, 14)
```

In [168]:

```
testi.shape
```

Out[168]:

```
(22, 14)
```

In [169]:

```
# we combining the two train and test data at one datframe
```

In [170]:

```
dfi = pd.concat([traini,testi],ignore_index=True,sort=True)
```

In [171]:

```
dfi.head()
```

Out[171]:

	BUI	Classes	DC	DMC	FFMC	FWI	ISI	RH	Rain	Region	Temperature	Ws	day	mon
0	19.2	0	33.1	19.4	84.5	7.3	4.7	83	0.0	1	31	17	16	
1	4.2	0	15.7	3.2	78.1	0.8	1.9	65	0.0	0	30	14	16	
2	6.7	0	8.3	7.0	74.2	0.8	1.6	59	0.8	1	33	16	23	
3	5.6	0	32.2	3.6	75.8	0.9	2.1	74	0.1	0	29	19	6	
4	3.0	0	7.7	2.9	47.4	0.1	0.3	80	1.8	0	26	16	29	

In [172]:

```
dfi.shape
```

Out[172]:

```
(244, 14)
```

In [173]:

```
dfi['Classes'].value_counts()
```

Out[173]:

```
0    230
1     14
Name: Classes, dtype: int64
```

In [686]:

```
dfi.tail()
```

Out[686]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Re
239	26	9	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	1	
240	27	9	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	0	
241	28	9	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	0	
242	29	9	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	0	
243	30	9	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	0	

8.3 Analysis:

8.3.1 Correlation

In [689]:

```
dfi.corr()
```

Out[689]:

	day	month	Temperature	RH	Ws	Rain	FFM
day	1.000000e+00	2.232788e-17	0.095772	-0.074209	0.047001	-0.112265	0.2240
month	2.232788e-17	1.000000e+00	-0.059017	-0.037884	-0.041447	0.035322	0.0155
Temperature	9.577222e-02	-5.901677e-02	1.000000	-0.654443	-0.278132	-0.326786	0.6774
RH	-7.420934e-02	-3.788419e-02	-0.654443	1.000000	0.236084	0.222968	-0.6456
Ws	4.700086e-02	-4.144673e-02	-0.278132	0.236084	1.000000	0.170169	-0.1632
Rain	-1.122654e-01	3.532207e-02	-0.326786	0.222968	0.170169	1.000000	-0.5440
FFMC	2.240321e-01	1.557668e-02	0.677491	-0.645658	-0.163255	-0.544045	1.0000
DMC	4.915710e-01	6.817778e-02	0.483105	-0.405133	-0.001246	-0.288548	0.6023
DC	5.279285e-01	1.276719e-01	0.370498	-0.220330	0.076245	-0.296804	0.5039
ISI	1.793008e-01	6.354476e-02	0.605971	-0.688268	0.012245	-0.347862	0.7407
BUI	5.172239e-01	8.556743e-02	0.456415	-0.349685	0.030303	-0.299409	0.5902
FWI	3.502343e-01	8.173226e-02	0.566839	-0.580457	0.033957	-0.324755	0.6914
Classes	1.405051e-02	2.108278e-01	0.421532	-0.470391	-0.094220	-0.180701	0.4513
Region	4.662229e-16	-9.586232e-17	0.273496	-0.406424	-0.176829	-0.041080	0.2246

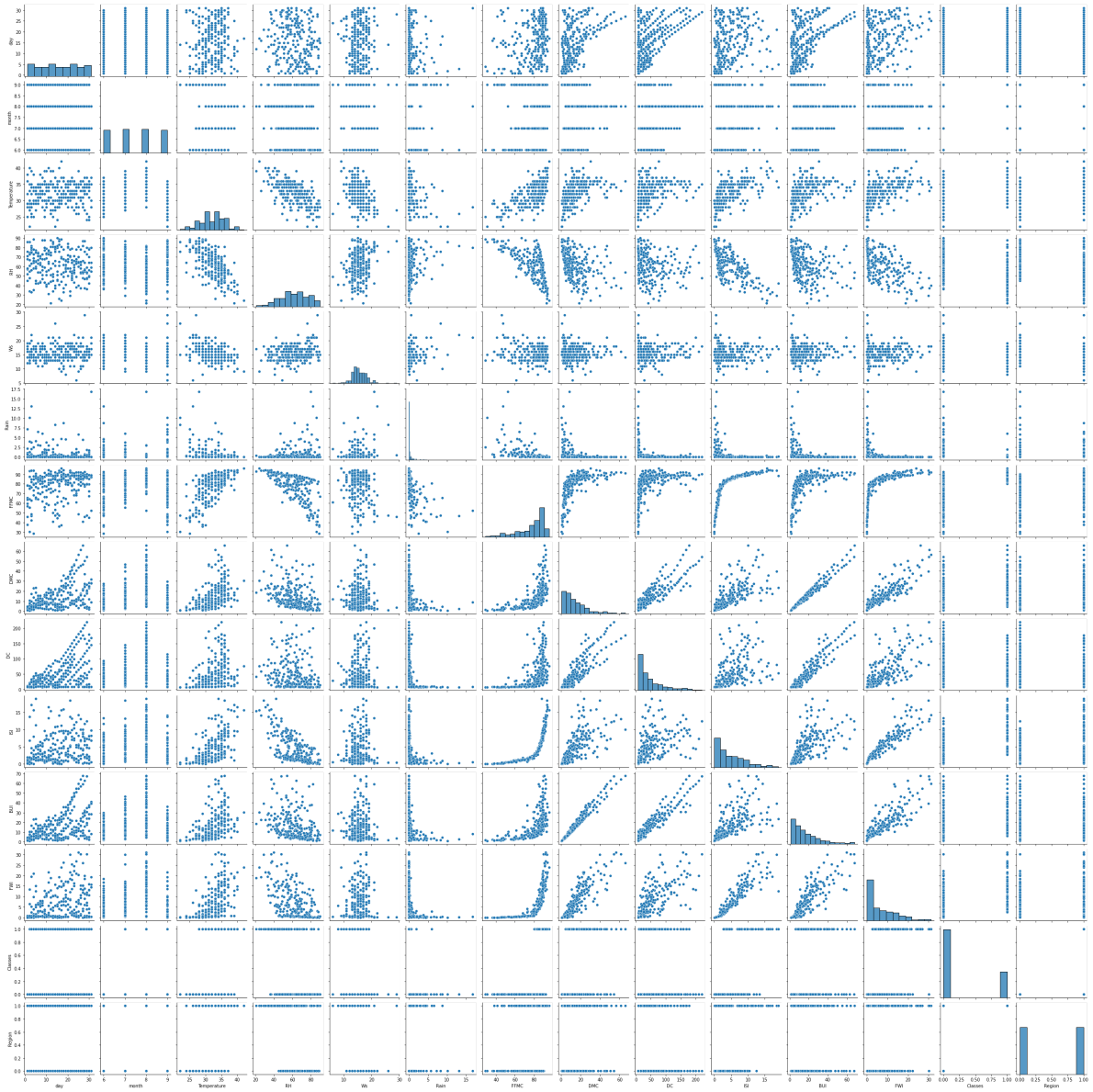
8.3.2 Pairplot

In [690]:

```
sns.pairplot(data=dfi)
```

Out[690]:

```
<seaborn.axisgrid.PairGrid at 0x266065e09a0>
```



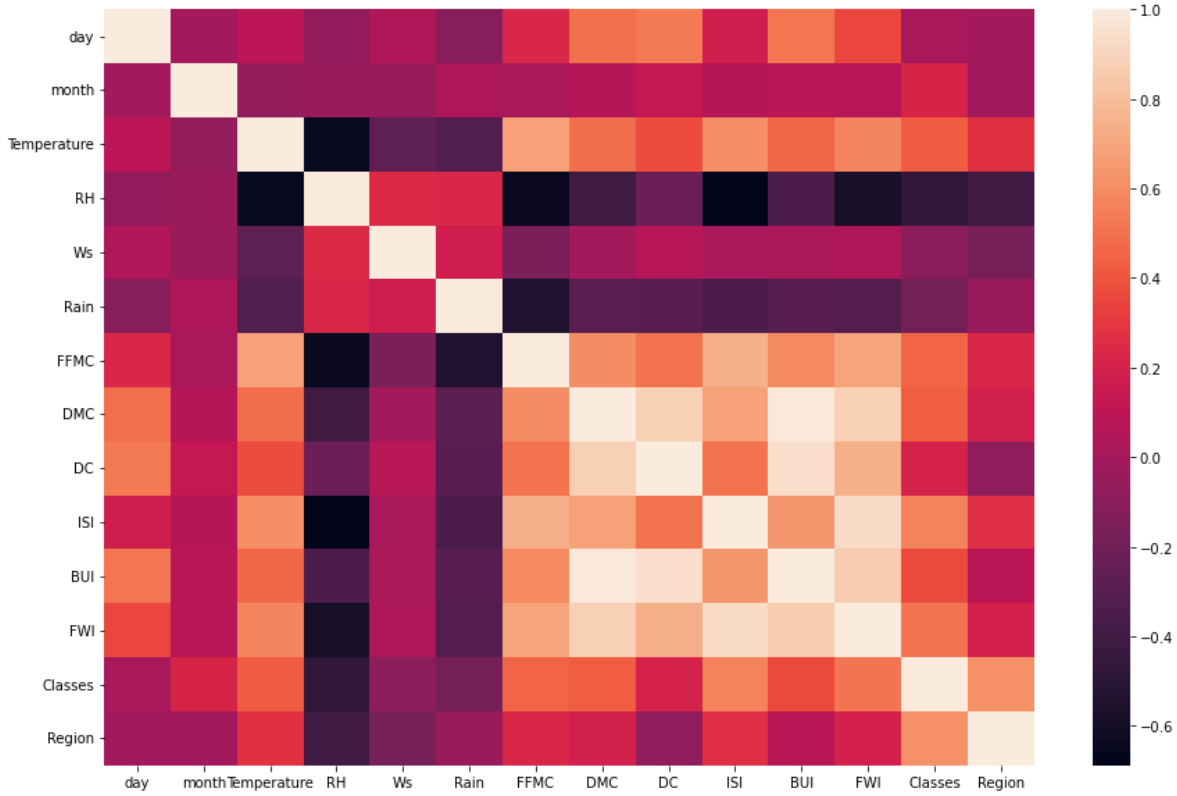
8.3.3 Heatmap:

In [693]:

```
plt.figure(figsize=(15,10))
sns.heatmap(dfi.corr())
```

Out[693]:

<AxesSubplot:>

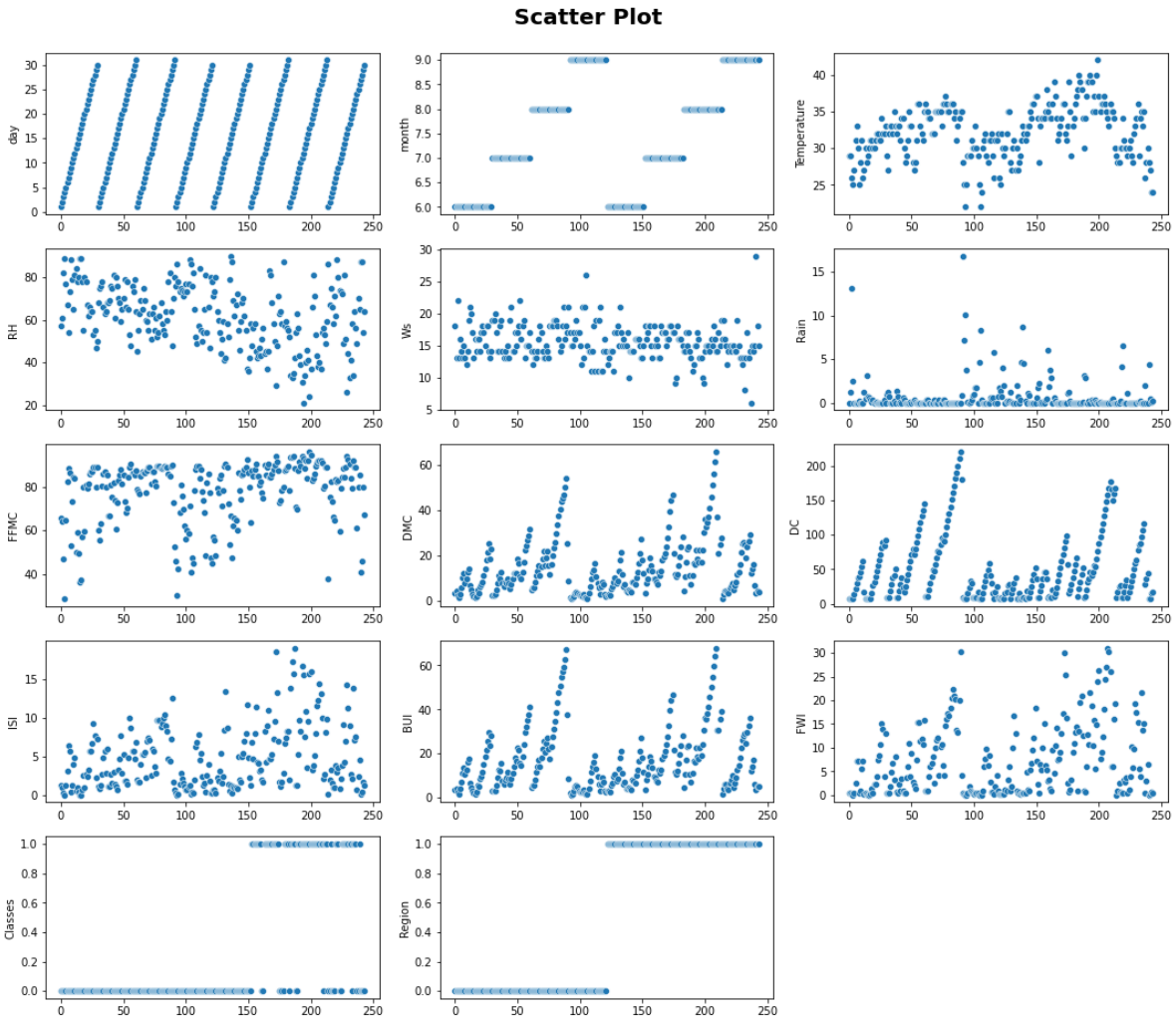


8.4 Scatter plot

In [722]:

```
plt.figure(figsize=(15,20))
plt.suptitle("Scatter Plot",fontsize=20,fontweight='bold',alpha=1,y=1)

for i in range(0,len(dfi.columns)):
    plt.subplot(8,3,i+1)
    sns.scatterplot(x=dfi.index,y=dfi[dfi.columns[i]],data=df)
    plt.tight_layout()
```

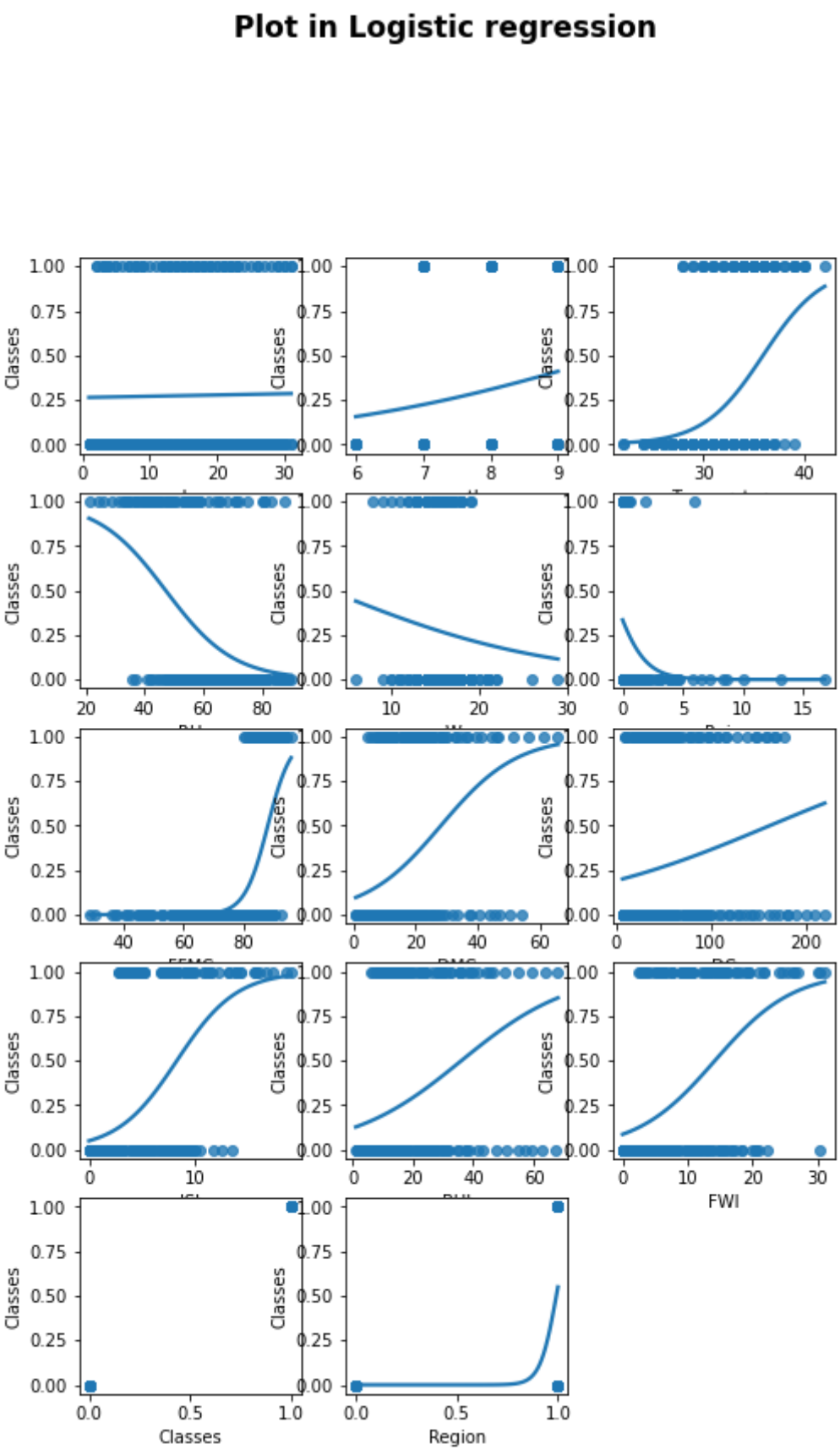


8.5 Plot in Logistic regression

In [760]:

```
plt.figure (figsize=(8,20))
plt.suptitle('Plot in Logistic regression',fontsize=17,fontweight='bold')

for i in range(len(dfi.columns)):
    plt.subplot(8,3,i+1)
    sns.regplot(data=dfi,x=dfi[dfi.columns[i]],y='Classes',logistic=True,ci=None)
```



In [ ]:

```
```

9.Logistic Regression on imbalance dataset

9.1 Independent and Dependent feature seperation

In [761]:

```
dfi.head()
```

Out[761]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Regic
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

In [463]:

```
Xi = dfi.drop("Classes",axis=1)
yi = dfi['Classes']
```

In [464]:

```
Xi.head(3)
```

Out[464]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0

In [465]:

```
yi.head()
```

Out[465]:

```
0    0
1    0
2    0
3    0
4    0
Name: Classes, dtype: int32
```

9.2 With out handiling imbalance data we do logistic regression

9.2.1 independent and dependent feature separation

In [466]:

```
X_b = dfi.drop(columns = ['Classes'])
y_b = dfi['Classes']
```

Before we fit our data to a model, let's visualize the relationship between our independent variables and the categories.

In [467]:

```
X_b.head(3)
y_b.head(3)
```

Out[467]:

```
0    0
1    0
2    0
Name: Classes, dtype: int32
```

9.2.2 Spliting the data into train and test split

In [471]:

```
X_train_b,X_test_b,y_train_b,y_test_b = train_test_split(X_scaled,y, test_size= 0.25, rando
```

9.2.3 Standardizing or Feature scalling the dataset

In [483]:

```
scalar = StandardScaler()
X_train_b = scalar.fit_transform(X_train_b)
X_test_b = scalar.transform(X_test_b)
```

In [484]:

```
X_train
```

Out[484]:

```
array([[ 0.23403727, -1.306737  , -0.37004382, ..., -0.98314165,
        -0.92875818, -1.01857744],
       [ 0.5787488 , -1.306737  , -0.65322289, ..., -0.5661262 ,
        -0.65983983, -1.01857744],
       [-1.02990499,  1.34469202, -0.37004382, ..., -0.70513135,
        -0.52538066,  0.98176139],
       ...,
       [ 1.61288338,  1.34469202, -2.06911826, ..., -0.98314165,
        -0.94220409, -1.01857744],
       [ 1.03836417,  1.34469202, -1.21958104, ..., -0.16996151,
        -0.55227249,  0.98176139],
       [ 0.23403727, -0.42292733,  0.19631433, ...,  0.51116373,
         0.26792846,  0.98176139]])
```

In [485]:

```
X_test
```

Out[485]:

```
array([[-0.45538578,  0.46088235,  0.76267247, ...,  0.28875549,
        -0.39092148, -1.01857744],
       [ 1.38307569, -1.306737  ,  1.32903062, ...,  0.73357197,
         1.51839877,  0.98176139],
       [ 1.61288338, -1.306737  ,  0.19631433, ...,  0.81697506,
         0.77887332, -1.01857744],
       ...,
       [-0.11067425,  1.34469202, -0.08686475, ...,  0.08719802,
         0.36204988,  0.98176139],
       [ 1.61288338, -0.42292733, -0.37004382, ...,  1.4494485 ,
         0.48306314, -1.01857744],
       [-1.6044242 ,  1.34469202, -1.21958104, ..., -0.87193753,
        -0.84808267,  0.98176139]])
```

9.2.4 Model Training

In [ ]:

```
log_reg_b = LogisticRegression()
log_reg_b.fit(X_train_b,y_train_b)
```

Prediction

In [486]:

```
y_pred_b = log_reg_b.predict(X_test_b)
y_pred_b
```

Out[486]:

```
array([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0])
```

In [487]:

```
accuracy_b = accuracy_score(y_test_b,y_pred_b)
accuracy_b
```

Out[487]:

```
0.9344262295081968
```

Performance Metrics

Confusion Matrix

In [475]:

```
conf_mat = confusion_matrix(y_test_b,y_pred_b)
conf_mat
```

Out[475]:

```
array([[50,  0],
       [ 3,  8]], dtype=int64)
```

In [368]:

```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

Accuracy

In [369]:

```
# Breaking down the formula for Accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negativ
Accuracy
```

Out[369]:

```
0.9836065573770492
```

Precision

In [370]:

```
# Precison
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[370]:

```
0.6666666666666666
```

Recall

In [371]:

```
# Recall
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[371]:

```
1.0
```

F1 Score

In [372]:

```
# F1 Score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[372]:

```
0.8
```

> i got accuracy is 98% percent

Handiling imbalance data by using imblearn:

In [374]:

```
import imblearn
```

In [375]:

```
from imblearn.combine import SMOTETomek
```

In [376]:

```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\dharavath ramdas
\anaconda3\lib\site-packages (0.9.1)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\dharavath ram
das\anaconda3\lib\site-packages (from imbalanced-learn) (1.1.2)
Requirement already satisfied: scipy>=1.3.2 in c:\users\dharavath ramdas\ana
conda3\lib\site-packages (from imbalanced-learn) (1.9.1)
Requirement already satisfied: joblib>=1.0.0 in c:\users\dharavath ramdas\an
aconda3\lib\site-packages (from imbalanced-learn) (1.1.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\dharavath ramdas\an
aconda3\lib\site-packages (from imbalanced-learn) (1.23.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\dharavath ra
mdas\anaconda3\lib\site-packages (from imbalanced-learn) (2.2.0)
```

In [377]:

```
from imblearn.under_sampling import RandomUnderSampler
```

In [378]:

```
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import NearMiss
```

In [379]:

```
smk=SMOTETomek(random_state=42)
smk
```

Out[379]:

```
SMOTETomek(random_state=42)
```

In [380]:

```
Xb,yb = smk.fit_resample(Xi,yi)
```

In [381]:

```
Xb.head()
```

Out[381]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0

In [382]:

```
yb.head()
```

Out[382]:

```
0    1
1    1
2    1
3    1
4    1
Name: Classes, dtype: int32
```

In [383]:

```
print(Xb.shape,yb.shape)
```

```
(452, 13) (452,)
```

In [384]:

```
from collections import Counter
print('Original dataset shape {}'.format(Counter(yi)))
print('Resampled dataset shape {}'.format(Counter(yb)))
```

```
Original dataset shape Counter({1: 229, 0: 15})
Resampled dataset shape Counter({1: 226, 0: 226})
```

we have to join the two datasets

In [386]:

```
dfb = Xb.join(pd.DataFrame(yb))
dfb.head()
```

Out[386]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Classe
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

Analysis

In [388]:

```
dfb.head(3)
```

Out[388]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Classes
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	1
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	1
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	1

In [389]:

```
dfb.tail(3)
```

Out[389]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI
449	6	9	33	71	14	5.735429	63.792064	3.498222	8.972571	0.971683
450	21	9	30	54	9	0.972189	71.692309	15.782841	62.303258	1.456509
451	20	9	28	86	15	3.303833	40.302787	5.104878	8.049826	0.100000

In [390]:

```
dfb.shape
```

Out[390]:

```
(452, 14)
```

info is used to check the data type, rows ,columns

In [391]:

```
dfb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          452 non-null    int32  
 1   month        452 non-null    int32  
 2   Temperature  452 non-null    int32  
 3   RH           452 non-null    int32  
 4   Ws           452 non-null    int32  
 5   Rain         452 non-null    float64
 6   FFMC         452 non-null    float64
 7   DMC          452 non-null    float64
 8   DC           452 non-null    float64
 9   ISI          452 non-null    float64
 10  BUI          452 non-null    float64
 11  FWI          452 non-null    float64
 12  Region       452 non-null    int64  
 13  Classes      452 non-null    int32  
dtypes: float64(7), int32(6), int64(1)
memory usage: 39.0 KB
```

describe is used for the analysis stats

In [392]:

```
dfb.describe()
```

Out[392]:

	day	month	Temperature	RH	Ws	Rain	FFMC	
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	4
mean	15.438053	8.150442	30.548673	64.455752	15.176991	1.047645	72.939145	
std	9.040592	1.085859	3.628518	12.933920	3.103478	1.897596	14.023498	
min	1.000000	6.000000	22.000000	21.000000	6.000000	0.000000	28.600000	
25%	7.000000	7.000000	28.000000	56.000000	14.000000	0.000000	64.833985	
50%	15.000000	9.000000	30.000000	66.000000	15.000000	0.200000	75.624129	
75%	24.000000	9.000000	33.000000	73.000000	17.000000	1.092167	84.500000	
max	31.000000	9.000000	42.000000	90.000000	29.000000	16.800000	96.000000	

In [393]:

```
## Checking null values
```

In [394]:

```
dfb.isnull().sum().sum()
```

Out[394]:

```
0
```

Correlation

In [396]:

```
dfb.corr()
```

Out[396]:

	day	month	Temperature	RH	Ws	Rain	FFMC	
day	1.000000	-0.078554	-0.081154	-0.125062	0.029158	-0.205115	0.157416	0.4
month	-0.078554	1.000000	-0.371031	0.149827	-0.059898	0.148030	-0.275189	-0.2
Temperature	-0.081154	-0.371031	1.000000	-0.506855	-0.156711	-0.152830	0.597160	0.5
RH	-0.125062	0.149827	-0.506855	1.000000	0.346930	0.218420	-0.675780	-0.4
Ws	0.029158	-0.059898	-0.156711	0.346930	1.000000	-0.023985	-0.182952	-0.0
Rain	-0.205115	0.148030	-0.152830	0.218420	-0.023985	1.000000	-0.501146	-0.3
FFMC	0.157416	-0.275189	0.597160	-0.675780	-0.182952	-0.501146	1.000000	0.6
DMC	0.453995	-0.238359	0.536940	-0.456360	-0.092532	-0.313365	0.605083	1.0
DC	0.453586	-0.170435	0.461474	-0.329129	-0.053424	-0.324732	0.537067	0.9
ISI	0.135950	-0.283944	0.632090	-0.635466	0.043092	-0.347052	0.727839	0.6
BUI	0.450949	-0.230436	0.530590	-0.420528	-0.053556	-0.328525	0.615078	0.9
FWI	0.274484	-0.273768	0.621214	-0.557940	0.042480	-0.328167	0.678699	0.8
Region	-0.019658	0.355908	-0.098361	-0.167643	-0.166074	0.062424	-0.072283	-0.0
Classes	0.033318	-0.677184	0.505410	-0.232890	0.115614	-0.171734	0.418963	0.3

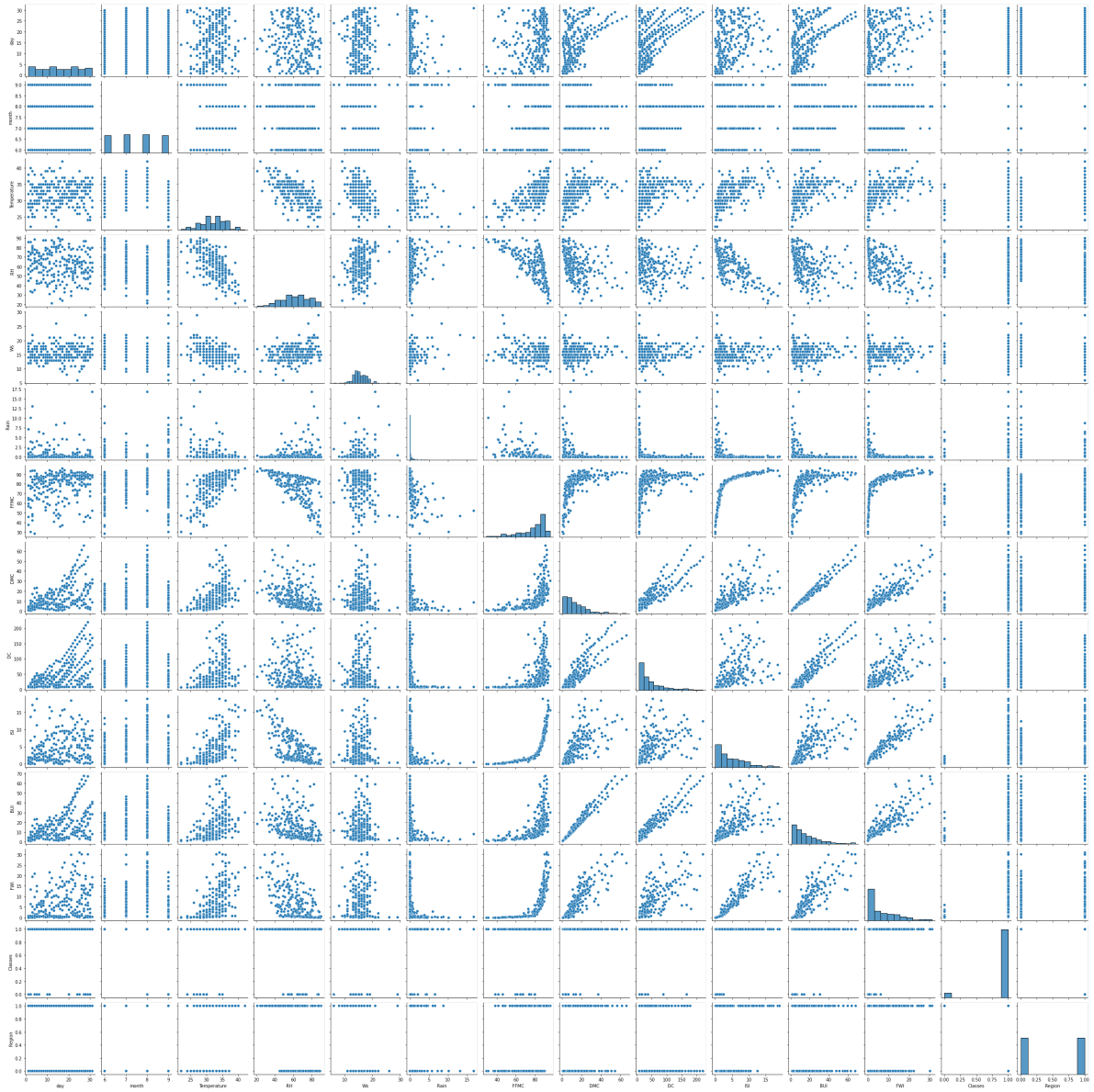
Pairplot

In [398]:

```
sns.pairplot(data=df)
```

Out[398]:

```
<seaborn.axisgrid.PairGrid at 0x26677046cd0>
```



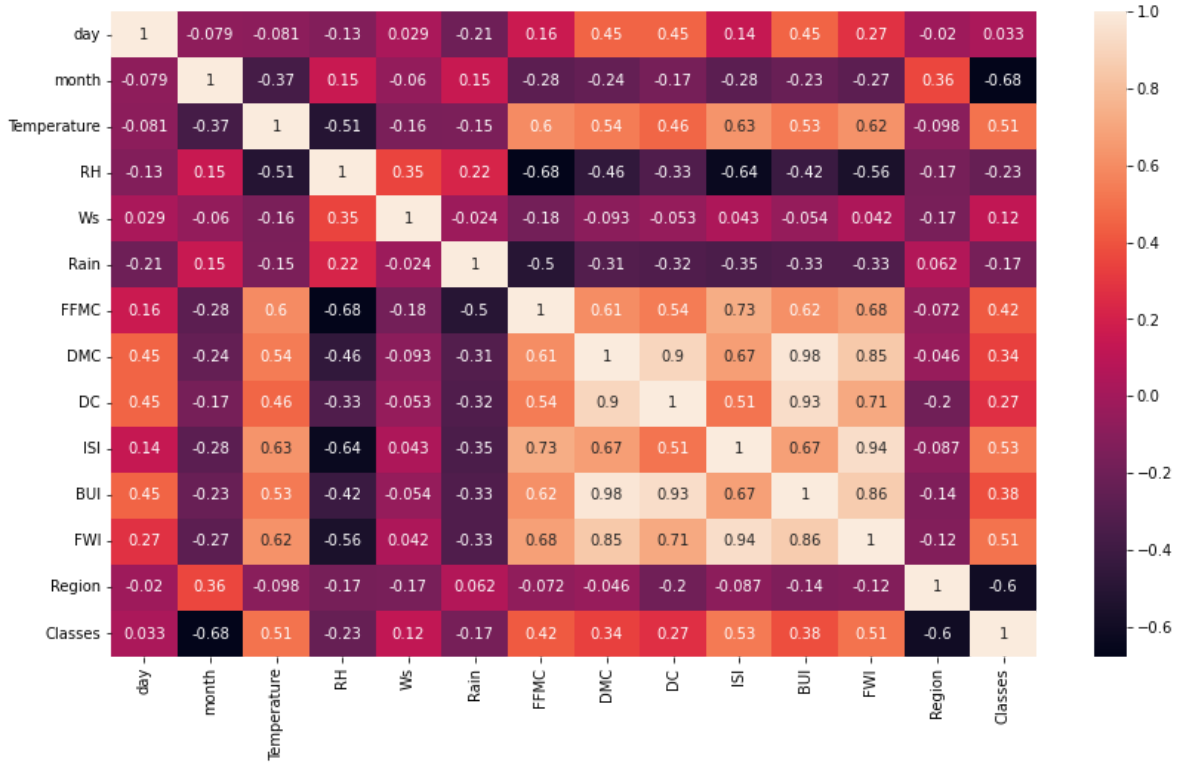
Heatmap

In [400]:

```
plt.figure(figsize=(14,8))
sns.heatmap(dfb.corr(),annot=True)
```

Out[400]:

<AxesSubplot:>

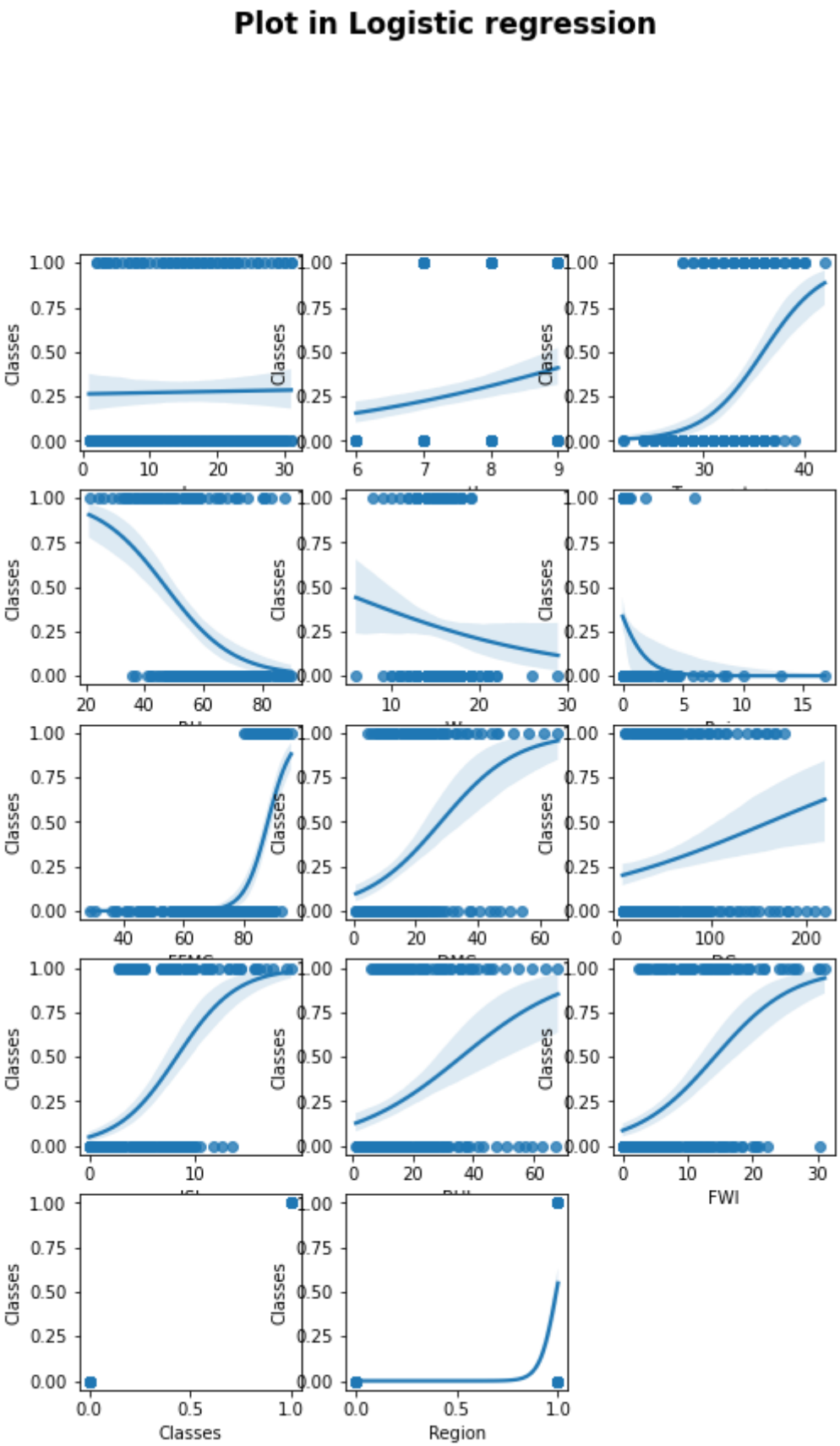


Plot in logistic regression

In [753]:

```
plt.figure (figsize=(8,20))
plt.suptitle('Plot in Logistic regression',fontsize=17,fontweight='bold')

for i in range(len(dfi.columns)):
    plt.subplot(8,3,i+1)
    sns.regplot(data=dfi,x=dfi[dfi.columns[i]],y='Classes',logistic=True)
```



In [ ]:

```
```

In [ ]:

```
```

Independent and Dependent feature seperation

In [725]:

```
X1 = dfb.drop('Classes',axis=1)
y1 = dfb['Classes']
```

In [726]:

```
# checking
```

In [727]:

```
X1.head(3)
```

Out[727]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0
1	2	6	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0
2	3	6	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0

In [728]:

```
y1.head(3)
```

Out[728]:

```
0    1
1    1
2    1
Name: Classes, dtype: int32
```

In [729]:

```
print(X1.shape, y1.shape)
```

```
(452, 13) (452,)
```

Spliting the data into train and test split

In [730]:

```
# spliting the data into train test split
# it will return 4 different paremeters
# output feature of x train is y train and x test is y test
# test size = 0.25   if 1000 in 25% of data
# random state
```

In [731]:

```
from sklearn.model_selection import train_test_split

X_train1,X_test1,y_train1,y_test1= train_test_split(X,y,test_size=0.25,random_state=355)
```

In [732]:

```
X_train1.shape,  y_train1.shape
```

Out[732]:

```
((183, 13), (183,))
```

In [733]:

```
X_test1.shape , y_test1.shape
```

Out[733]:

```
((61, 13), (61,))
```

Standardizing or Feature scalling the dataset:

In [734]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler
```

Out[734]:

StandardScaler

StandardScaler()

In [735]:

```
# Apply data
```

In [736]:

```
X_train1 = scaler.fit_transform(X_train1)
```

In [737]:

```
X_test1 = scaler.transform(X_test1)
```

In [738]:

```
#data lekage we dont need to leak the data of test to train data
#avoid datalekage use transform
#example :
#    is eaxm paper is x_train if you get before exam is called parer lekage
# f to f' we convert mean and std in fit and transform
```

In [739]:

```
X_train1
```

Out[739]:

```
array([[-0.82473448, -0.43394895, -0.07323543, ..., -0.70181037,
        -0.88736547, -1.07375098],
       [ 0.91806471, -0.43394895, -1.19007573, ..., -0.19950741,
        -0.79463296, -1.07375098],
       [ 0.33713165, -0.43394895,  0.48518472, ...,  0.6950047 ,
         0.75532479,  0.93131463],
       ...,
       [-1.17329431, -0.43394895, -0.07323543, ..., -0.35088638,
        -0.06602034, -1.07375098],
       [ 1.61518438,  0.47884022,  0.76439479, ...,  1.36244836,
        -0.41045539, -1.07375098],
       [-0.82473448, -1.34673811, -1.4692858 , ..., -0.6398826 ,
        -0.71514794,  0.93131463]])
```

In [740]:

```
X_test1
```

```
         1.00058182e+00,  2.06779796e+00,  6.72060790e-01,
        -1.37784459e+00, -1.06512610e+00, -9.02310145e-01,
        -8.82835418e-01, -1.04585349e+00, -9.27107977e-01,
        -1.07375098e+00],
       [-4.76174640e-01, -4.33948947e-01,  1.04360487e+00,
        -1.23324946e+00, -8.68435032e-01, -4.17392085e-01,
         8.36663383e-01, -2.32641600e-01, -6.74446044e-01,
         8.28291306e-01, -3.50886383e-01,  3.04909721e-01,
         9.31314629e-01],
       [ 1.73137099e+00, -4.33948947e-01,  7.64394795e-01,
         1.20587680e-01,  5.99681465e-01, -4.17392085e-01,
         6.33433221e-01,  1.26894258e+00,  1.94184813e+00,
         4.71806572e-01,  1.62392113e+00,  1.11300735e+00,
        -1.07375098e+00],
       [ 1.15043793e+00, -4.33948947e-01,  1.04360487e+00,
        -6.24022749e-01,  1.33373971e+00, -4.17392085e-01,
         7.73591954e-01,  1.17468702e-01,  9.66175483e-01,
         1.23230734e+00,  4.33531933e-01,  1.06001734e+00,
        -1.07375098e+00],
       [ 2.20945033e-01, -1.34673811e+00, -3.52445504e-01,
```

Model Training

In [741]:

```
logistic_regre1= LogisticRegression()
logistic_regre1
```

Out[741]:

```
LogisticRegression
LogisticRegression()
```

In [742]:

```
logistic_regre1.fit(X_train1,y_train1)
```

Out[742]:

```
LogisticRegression
LogisticRegression()
```

prediction

In [743]:

```
logistic_regre1_pred1 = logistic_regre1.predict(X_test1)
logistic_regre1_pred1
```

Out[743]:

```
array([0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0])
```

Performance Metrics:

1.Confusion Matrix

In [744]:

```
confusion_mat1=confusion_matrix(y_test1, logistic_regre1_pred1)
confusion_mat1
```

Out[744]:

```
array([[44,  3],
       [ 1, 13]], dtype=int64)
```

In [745]:

```
truly_positive=confusion_mat1[0][0]
falsely_positive=confusion_mat1[0][1]
falsely_negative=confusion_mat1[1][0]
truly_negative=confusion_mat1[1][1]
```

Accuracy Score

In [746]:

```
accuracy1=round(accuracy_score(y_test1, logistic_regre1_pred1),4)
accuracy1
```

Out[746]:

```
0.9344
```

In [747]:

```
### manual calcualtion for accuracy
accuracy_manual1=round(((truly_positive+truly_negative)/(truly_positive+falsely_positive+fa
print("Accuracy of our model is {}".format(accuracy_manual1))
```

```
Accuracy of our model is 0.9344
```

Precision Score

In [748]:

```
precision_manual1=round(truly_positive/(truly_positive+falsely_positive),4)
print("Precision of our model is {}".format(precision_manual1))
```

```
Precision of our model is 0.9362
```

Recall Score

In [749]:

```
recall_manual1=round(truly_positive/(truly_positive+falsely_negative),4)
print("Recall of our model is {}".format(recall_manual1))
```

```
Recall of our model is 0.9778
```

F-1 Score

In [750]:

```
f1_score1=2*(precision_manual1*recall_manual1)/(precision_manual1+recall_manual1)
print("F-1 Score of our model is {} ".format(round(f1_score1,4)))
```

```
F-1 Score of our model is 0.9565
```

Classification Report

In [751]:

```
print(classification_report(y_test1, logistic_regre1_pred1))
```

```
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        47
           1       0.81      0.93      0.87        14

    accuracy                           0.93        61
   macro avg       0.90      0.93      0.91        61
weighted avg       0.94      0.93      0.94        61
```

In [ ]:

```

```

I am solved all the three problem statements by using logistic regression

Handiling imbalance data refernce : https://github.com/krishnaik06/Handle-Imbalanced-Dataset/blob/master/Handling%20Imbalanced%20Data-%20Under%20Sampling.ipynb (https://github.com/krishnaik06/Handle-Imbalanced-Dataset/blob/master/Handling%20Imbalanced%20Data-%20Under%20Sampling.ipynb)

follow me on
GitHub : https://github.com/dharavathramdas101
Linkedin : https://www.linkedin.com/in/dharavath-ramdas-a283aa213/

In [ ]: