ML Logistic Regression Practical Implementation on Algerian Forest Fire Prediction Dataset

Submitted By :- Ambarish Singh

EDA and Feature Engineering

1. Data Profiling
2. Data Cleaning
3. Statistical Analysis
4. Graphical Analysis
5. Data Scaling

Logistic Regression Implementation

1. Logistic Regression on Original Dataset
2. Performance metrics for above model
3. Creating an imbalanced dataset from original dataset
4. Balancing the imbalanced Dataset
5. Logistic Regression on above dataset
6. Performance Metrics for above Dataset
7. Compairing the performance of Original balanced dataset and the balanced dataset that we create from an imbalanced one

What is Logistic Regression?

Ans : Logistic regression is one such regression algorithm which can be used for performing classification problems. It calculates the probability that a given value belongs to a specific class. If the probability is more than 50%, it assigns the value in that particular class else if the probability is less than 50%, the value is assigned to the other class. Therefore, we can say that logistic regression acts as a binary classifier.

Advantages of Logisitic Regression

1. It is very simple and easy to implement.
2. The output is more informative than other classification algorithms
3. It expresses the relationship between independent and dependent variables
4. Very effective with linearly seperable data

Disadvantages of Logisitic Regression

1. Not effective with data which are not linearly seperable
2. Not as powerful as other classification models
3. Multiclass classifications are much easier to do with other algorithms than logisitic regression
4. It can only predict categorical outcomes

Importing required libraries

In [135]:

```
### Pandas and Numpy
import pandas as pd
import numpy as np

### Visualisation libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

### For Q-Q Plot
import scipy.stats as stats

### To ignore warnings
import warnings
warnings.filterwarnings('ignore')

### 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

### To be able to see maximum columns on screen
pd.set_option('display.max_columns', 500)

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

Reading the Dataset

In [2]:

```
df = pd.read_csv('Algerian_forest_fires_dataset_UPDATE.csv',header = 1)
df.head()
```

Out[2]:

	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
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire

Info about dataset and its attributes

1. 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.
2. 122 instances for each region.
3. The period from June 2012 to September 2012.
4. The dataset includes 11 attribues and 1 output attribue (classes)

5. The 244 instances have been classified into fire (138 classes) and notfire (106 classes) classes.

Attributes

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 fire and not fire

In [3]:
```
# index 122, 123 need to be removed from dataset
df.iloc[121:].head(4)
```

Out[3]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
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	FWI	C
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	

Shape Of The Dataset

In [4]:
```
df.shape
```

Out[4]:
```
(246, 14)
```

Dropping the unwanted rows and column

In [5]:
```
## dropping rows having region name and headder
df.drop([122,123],inplace=True)  ## droping row 122,123 from dataset
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)

df.iloc[121:].head()
```

Out[5]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	not fire
122	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	not fire
123	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	not fire
124	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	not fire
125	04	06	2012	30	64	14	0	79.4	5.2	15.4	2.2	5.6	1	not fire

Shape of the data sfter dropping the column

In [130]:
```
df.shape
```

Out[130]: (244, 14)

Creating Region feature

In [7]:
```
### creating feature called Region 0 for Bejaia region and 1 for Sidi Bel-ab
df.loc[:122,'Region']=0
df.loc[122:, 'Region']=1

df.iloc[120:].head(8)
```

Out[7]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
120	29	09	2012	26	80	16	1.8	47.4	2.9	7.7	0.3	3	0.1	not fire
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	not fire
122	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	not fire
123	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	not fire
124	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	not fire
125	04	06	2012	30	64	14	0	79.4	5.2	15.4	2.2	5.6	1	not fire
126	05	06	2012	32	60	14	0.2	77.1	6	17.6	1.8	6.5	0.9	not fire
127	06	06	2012	35	54	11	0.1	83.7	8.4	26.3	3.1	9.3	3.1	fire

```
### <span style='color:green'>Datatypes and describe</span>
```

In [8]:
```
# here it is visible that all datatypes are in object
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    float64
dtypes: float64(1), object(14)
memory usage: 28.7+ KB
```

In [9]:
```
df.describe(include='all').T
```

Out[9]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
day	244	31	01	8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
month	244	4	07	62	NaN	NaN	NaN	NaN	NaN	NaN	NaN
year	244	1	2012	244	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Temperature	244	19	35	29	NaN	NaN	NaN	NaN	NaN	NaN	NaN
RH	244	62	64	10	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Ws	244	18	14	43	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Rain	244	39	0	133	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FFMC	244	173	88.9	8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DMC	244	166	7.9	5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DC	244	198	8	5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ISI	244	106	1.1	8	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BUI	244	174	3	5	NaN	NaN	NaN	NaN	NaN	NaN	NaN
FWI	244	127	0.4	12	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Classes	243	8	fire	131	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Region	244.0	NaN	NaN	NaN	0.5	0.501028	0.0	0.0	0.5	1.0	1.0

Data Cleaning

In [10]:
```
# here it is visible that some columns have spaces in the names like RH, Ws
df.columns
```

Out[10]:
```
Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain ', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes  ', 'Region'],
      dtype='object')
```

In [11]:
```
# stripping spaces from column names
df.columns= [col_name.strip() for col_name in df.columns]
df.columns
```

Out[11]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'Region'],
      dtype='object')
```

In [12]:
```
## converting all feature values to string so that we can do data cleaning a
df=df.astype(str)
```

In [13]:
```
## somes values in colums also have space
for feature in ['Rain', 'FFMC','DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes']:
    df[feature]= df[feature].str.replace(" ","")
```

Checking the unique data points of FWI feature

In [14]:
```
df['FWI'].unique()
```

Out[14]:
```
array(['0.5', '0.4', '0.1', '0', '2.5', '7.2', '7.1', '0.3', '0.9', '5.6',
       '0.2', '1.4', '2.2', '2.3', '3.8', '7.5', '8.4', '10.6', '15',
       '13.9', '3.9', '12.9', '1.7', '4.9', '6.8', '3.2', '8', '0.6',
       '3.4', '0.8', '3.6', '6', '10.9', '4', '8.8', '2.8', '2.1', '1.3',
       '7.3', '15.3', '11.3', '11.9', '10.7', '15.7', '6.1', '2.6', '9.9',
       '11.6', '12.1', '4.2', '10.2', '6.3', '14.6', '16.1', '17.2',
       '16.8', '18.4', '20.4', '22.3', '20.9', '20.3', '13.7', '13.2',
       '19.9', '30.2', '5.9', '7.7', '9.7', '8.3', '0.7', '4.1', '1',
       '3.1', '1.9', '10', '16.7', '1.2', '5.3', '6.7', '9.5', '12',
       '6.4', '5.2', '3', '9.6', '4.7', 'fire', '14.1', '9.1', '13',
       '17.3', '30', '25.4', '16.3', '9', '14.5', '13.5', '19.5', '12.6',
       '12.7', '21.6', '18.8', '10.5', '5.5', '14.8', '24', '26.3',
       '12.2', '18.1', '24.5', '26.9', '31.1', '30.3', '26.1', '16',
       '19.4', '2.7', '3.7', '10.3', '5.7', '9.8', '19.3', '17.5', '15.4',
       '15.2', '6.5'], dtype=object)
```

Here we got 'fire' data point in FWI column , so to remove this it is replaced by mode

In [15]:
```
df[df['FWI']=='fire'].index
```

Out[15]:
```
Int64Index([165], dtype='int64')
```

In [16]:
```
df['FWI'].mode()
```

Out[16]:
```
0    0.4
dtype: object
```

In [17]:
```
df.loc[165,'FWI']='0.4'
```

In [18]:
```
### replacing nan value wit fire to make data equal to the info given in dat
df[df['Classes']== 'nan'].index
df.loc[165,'Classes']='fire'
```

In [19]:
```
### encoding classes feature
df['Classes']=df['Classes'].str.replace('notfire','0')
df['Classes']=df['Classes'].str.replace('fire','1')
```

In [20]:
```
### Dropping year feature as data is realted to year 2012.
df.drop('year', axis=1, inplace=True)
```

Changing datatype to Numerical from Object

In [21]:
```
### changing datatypes of features to numerical for numerical features as al

datatype_convert={'day':'int64', 'month':'int64', 'Temperature':'int64', 'RH
                  'Rain':'float64', 'FFMC':'float64', 'DMC':'float64', 'DC':
                  'FWI':'float64', 'Classes':'int64', 'Region':'float64'}

df=df.astype(datatype_convert)
df.dtypes
```

Out[21]:
```
day              int64
month            int64
Temperature      int64
RH               int64
Ws               int64
Rain           float64
FFMC           float64
DMC            float64
DC             float64
ISI            float64
BUI            float64
FWI            float64
Classes          int64
Region         float64
dtype: object
```

Observation

- So, all the features are converted from categorical to numerical datatypes

In [22]:
```
df.shape
```

Out[22]:
```
(244, 14)
```

2.7 Checking Null values and Duplicates

In [23]:
```
### checking for null values

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

Out[23]:
```
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
```

In [24]:
```
df[df.duplicated()]
```

Out[24]:

day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region

Observation

1. There is no null value in dataset.
2. Total 244 roww and 15 columns is present.
3. There is no duplicate observation in dataset.

Creating a Copy of dataframe from Original Dataframe

In [25]:
```
data=df.copy()
data.head()
```

Out[25]:

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

Statistical Analysis

In [26]:
```
data.describe().T
```

Out[26]:

	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	49.288484	47.619393	6.9	13.275	33.10	68.150	220.4
ISI	244.0	4.774180	4.175318	0.0	1.400	3.50	7.300	19.0
BUI	244.0	16.664754	14.204824	1.1	6.000	12.25	22.525	68.0
FWI	244.0	7.008197	7.437383	0.0	0.700	4.20	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

Observation

No missing values is present in the dataset

In [27]:
```
1 data.cov()
```

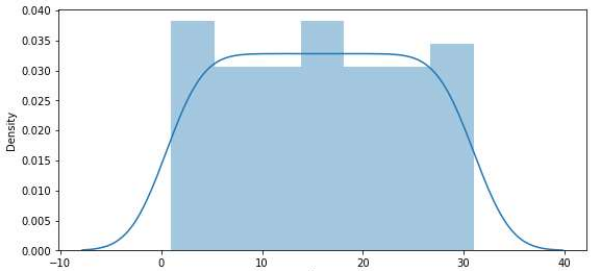
Out[27]:

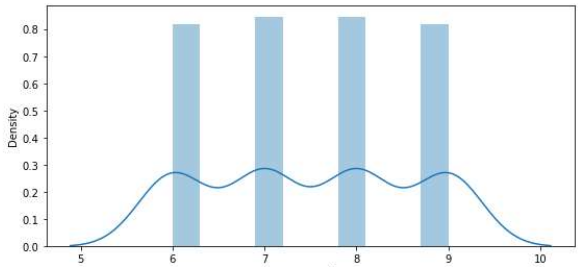
	day	month	Temperature	RH	Ws	Rain	F
day	7.788167e+01	4.605370e-16	3.071308	-9.747689	1.165621	-1.980908	28.34
month	4.605370e-16	1.238683e+00	-0.238683	-0.627572	-0.129630	0.078601	0.24
Temperature	3.071308e+00	-2.386831e-01	13.204817	-35.396782	-2.840215	-2.374270	35.29
RH	-9.747689e+00	-6.275720e-01	-35.396782	221.539415	9.874739	6.635431	-137.78
Ws	1.165621e+00	-1.296296e-01	-2.840215	9.874739	7.897102	0.956129	-6.57
Rain	-1.980908e+00	7.860082e-02	-2.374270	6.635431	0.956129	3.997623	-15.59
FFMC	2.834676e+01	2.485597e-01	35.297598	-137.785533	-6.577727	-15.595918	205.56
DMC	5.365433e+01	9.384774e-01	21.712423	-74.580245	-0.043306	-7.135415	106.82
DC	2.218594e+02	6.766276e+00	64.113719	-156.174991	10.204060	-28.259196	344.04
ISI	6.548769e+00	2.866255e-01	9.218043	-42.920524	0.178913	-2.897687	44.28
BUI	6.483903e+01	1.356790e+00	23.512265	-73.700941	1.187799	-8.496825	120.09
FWI	2.303207e+01	6.962963e-01	15.102287	-63.152169	0.606139	-4.800293	73.18
Classes	8.845038e-01	1.234568e-02	0.935168	-3.216117	-0.092862	-0.376833	5.48
Region	0.000000e+00	0.000000e+00	0.497942	-3.030864	-0.248971	-0.041152	1.61

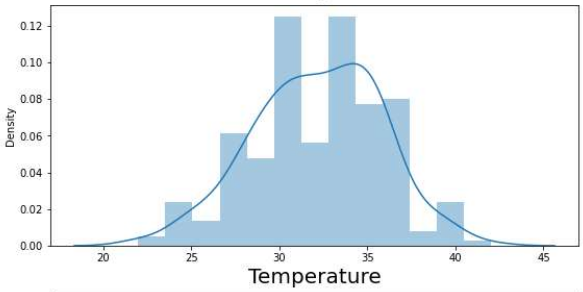
Checking the distribution of the features

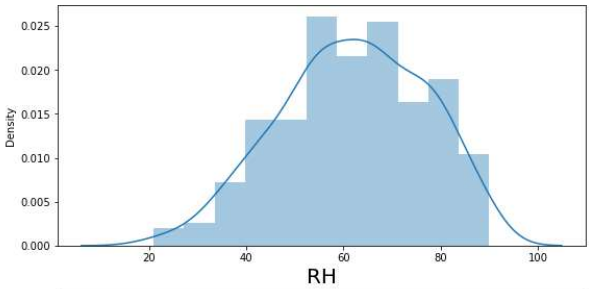
In [28]:

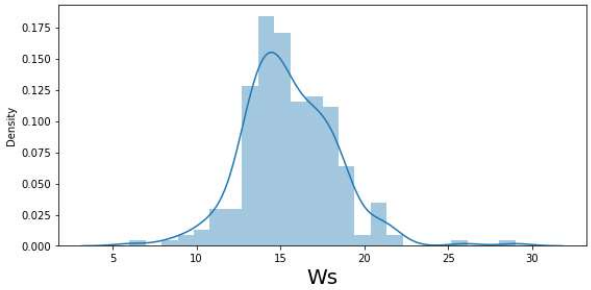
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,40), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=15 :     # as there are 15 columns in the data
        ax = plt.subplot(8,2,plotnumber)
        sns.distplot(data[column])
        plt.xlabel(column,fontsize=20)

    plotnumber+=1
plt.show()
```

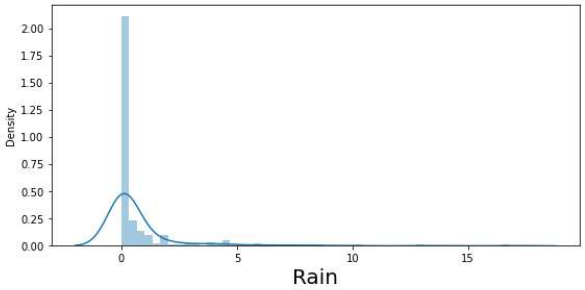


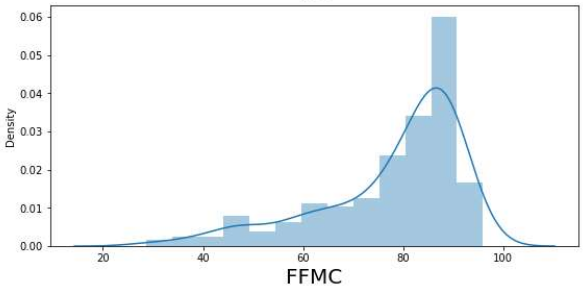


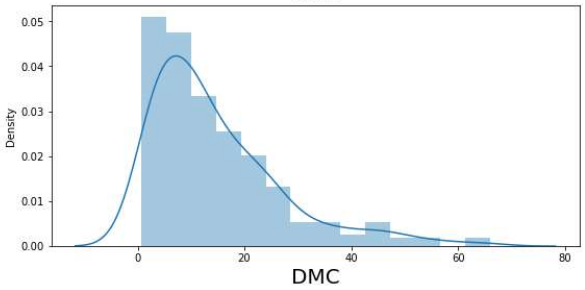


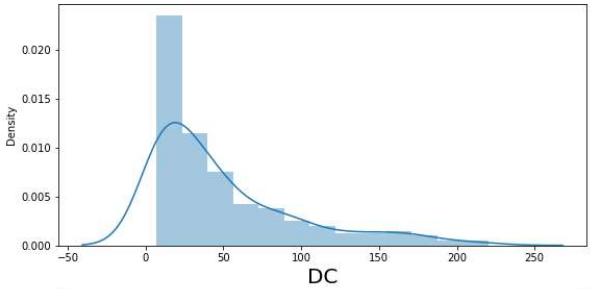


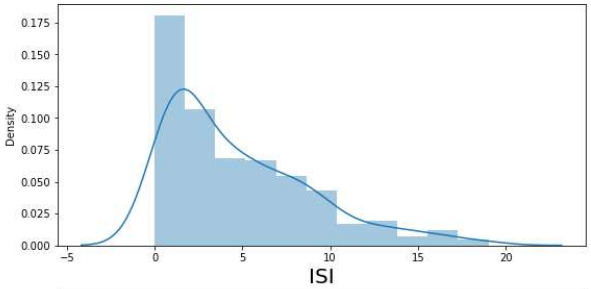


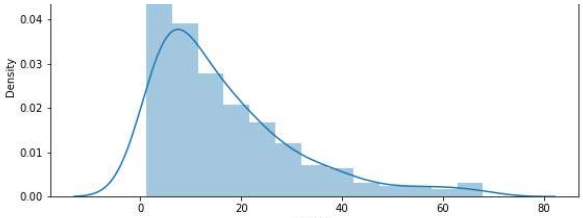


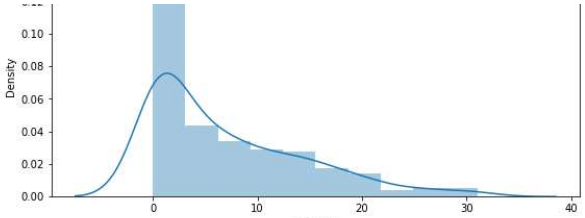


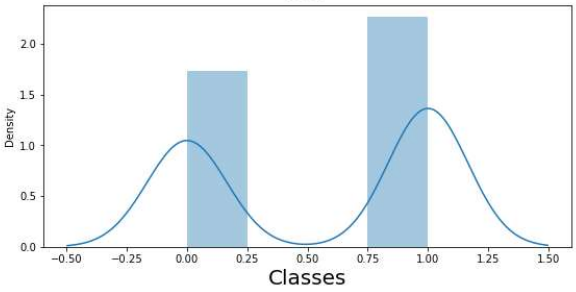


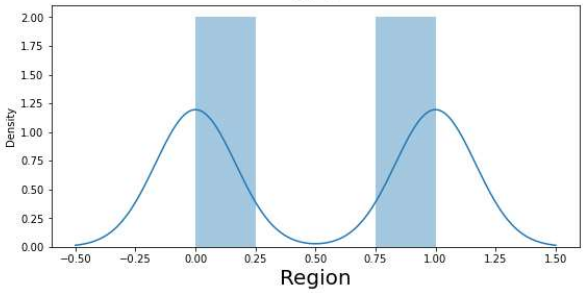












Observation

- Rain, DMC, DC, FWI,ISI,BUI are rightly skewed(log normal distribution)
- There is no variance in the year attribute

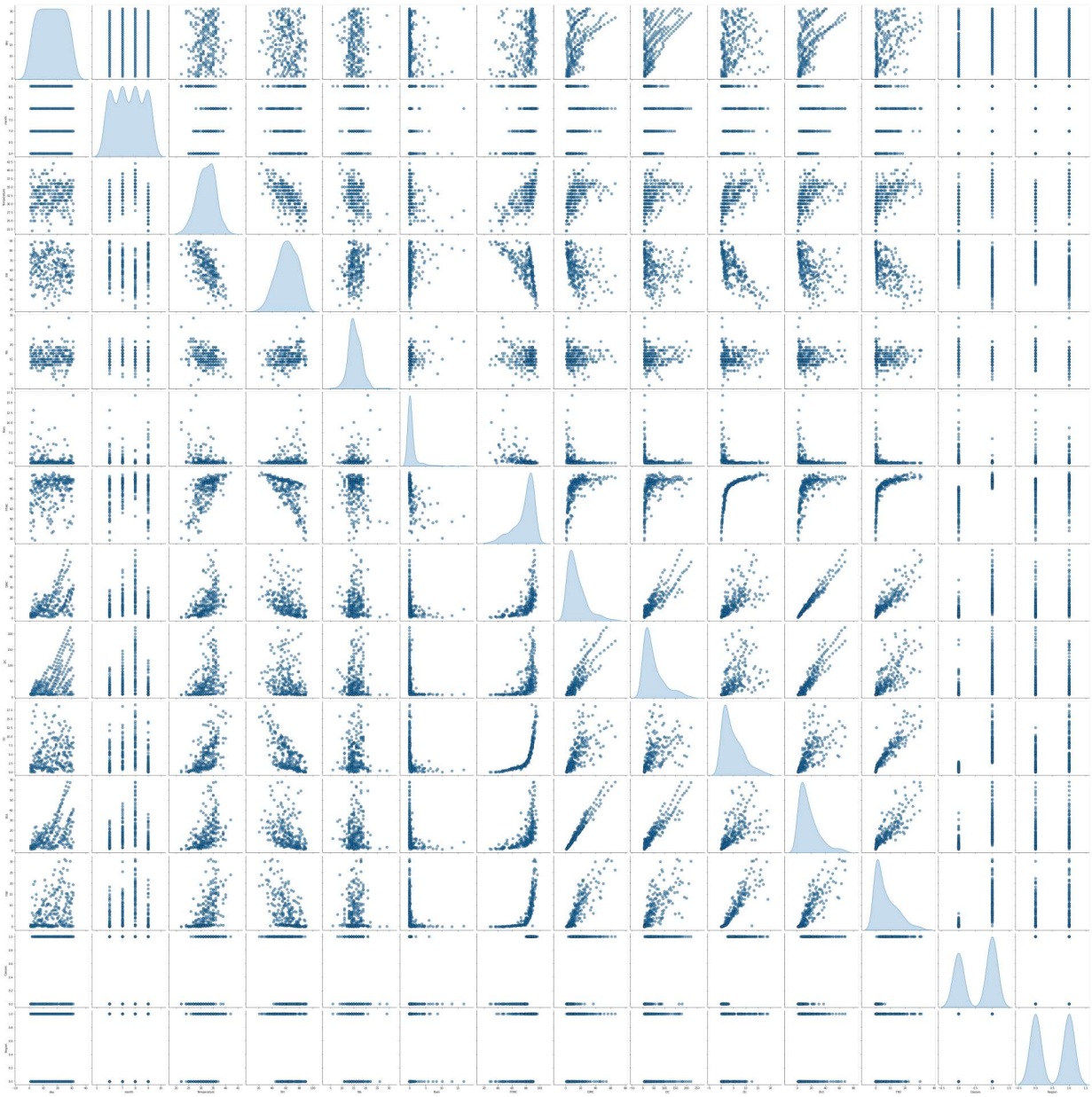
Multivariate Analysis

In [29]:

```
plt.figure(figsize=(15,15))
plt.suptitle('Multivariate Analysis', fontsize=20, fontweight='bold', alpha=
sns.pairplot(data,  diag_kind = 'kde',
             plot_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'},
             size = 4)
```

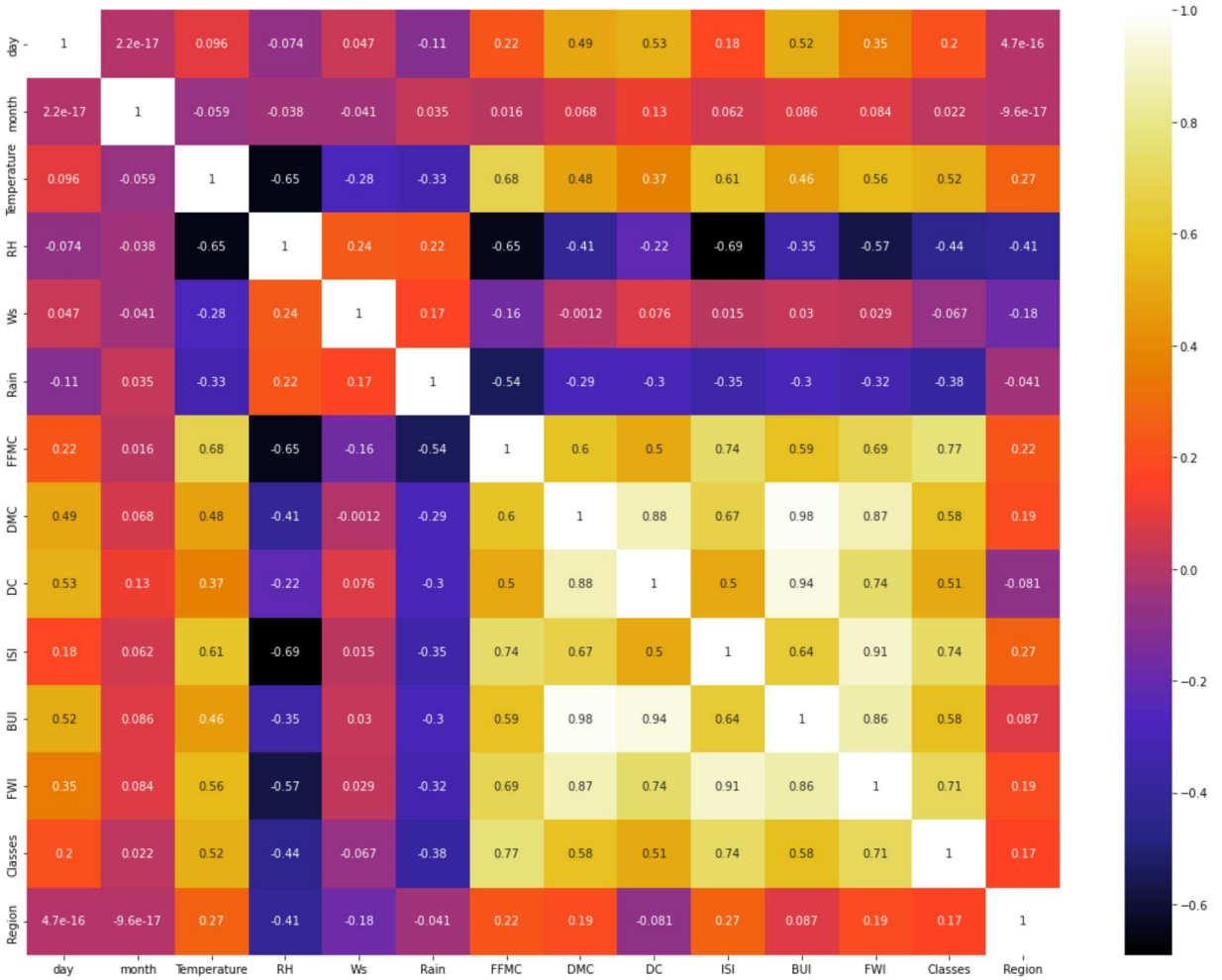
Out[29]:

```
<seaborn.axisgrid.PairGrid at 0x1fb75a096d0>

<Figure size 1080x1080 with 0 Axes>
```



In [30]:
```
1 plt.figure(figsize = (20,15))
2 sns.heatmap(data.corr(),cmap="CMRmap", annot=True)
```

Out[30]: <AxesSubplot:>



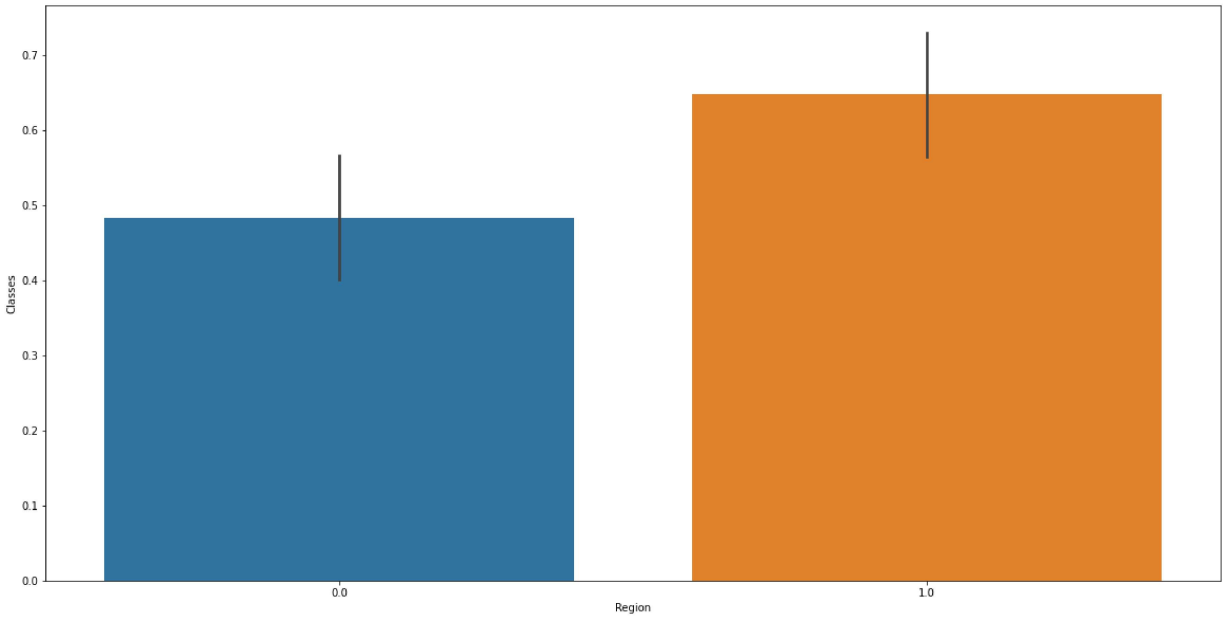
Visualisation of Target Feature

In [31]:
```
1 data.Classes.value_counts()
```

Out[31]:
```
1    138
0    106
Name: Classes, dtype: int64
```

In [32]:
```
1 import matplotlib
2 matplotlib.rcParams['figure.figsize']=(20,10)
3
4 sns.barplot(x="Region",y="Classes",data=data)
```

Out[32]:
```
<AxesSubplot:xlabel='Region', ylabel='Classes'>
```



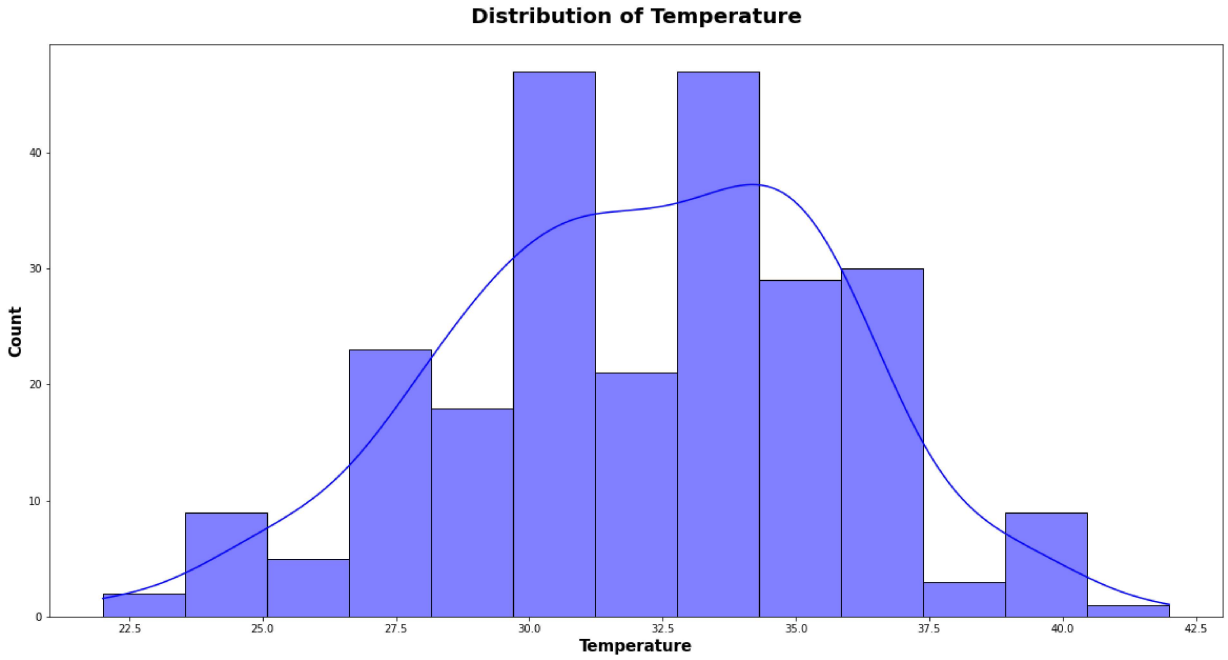
Observation

- Sidi-Bel Abbes region has most of the fire happen

Visualisation of Temperature Feature

In [33]:

```
plt.subplots(figsize=(20,10))
sns.histplot("Distribution of Temperature",x=data.Temperature,color='b',kde=
plt.title("Distribution of Temperature",weight='bold',fontsize=20,pad=20)
plt.xlabel("Temperature",weight='bold',fontsize=15)
plt.ylabel("Count",weight='bold',fontsize=15)
plt.show()
```



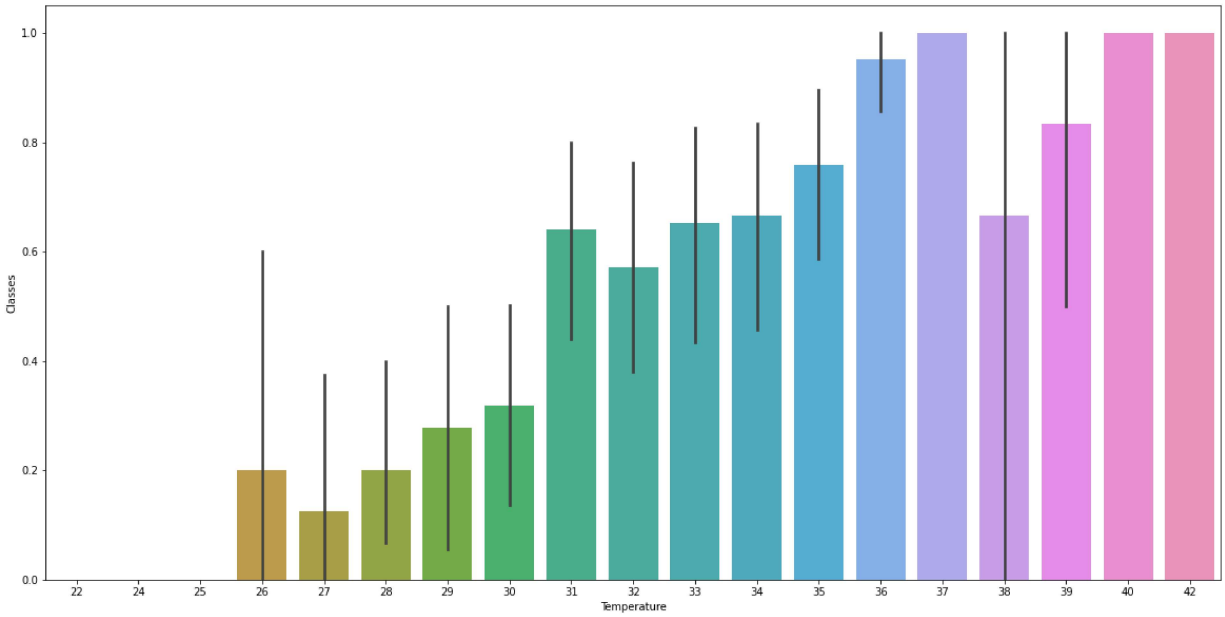
Observation

- Temperature occur most of the time in range 32.5 to 35.0

Highest Temperature attained

In [34]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

sns.barplot(x="Temperature",y="Classes",data=data)
```

Out[34]: <AxesSubplot:xlabel='Temperature', ylabel='Classes'>



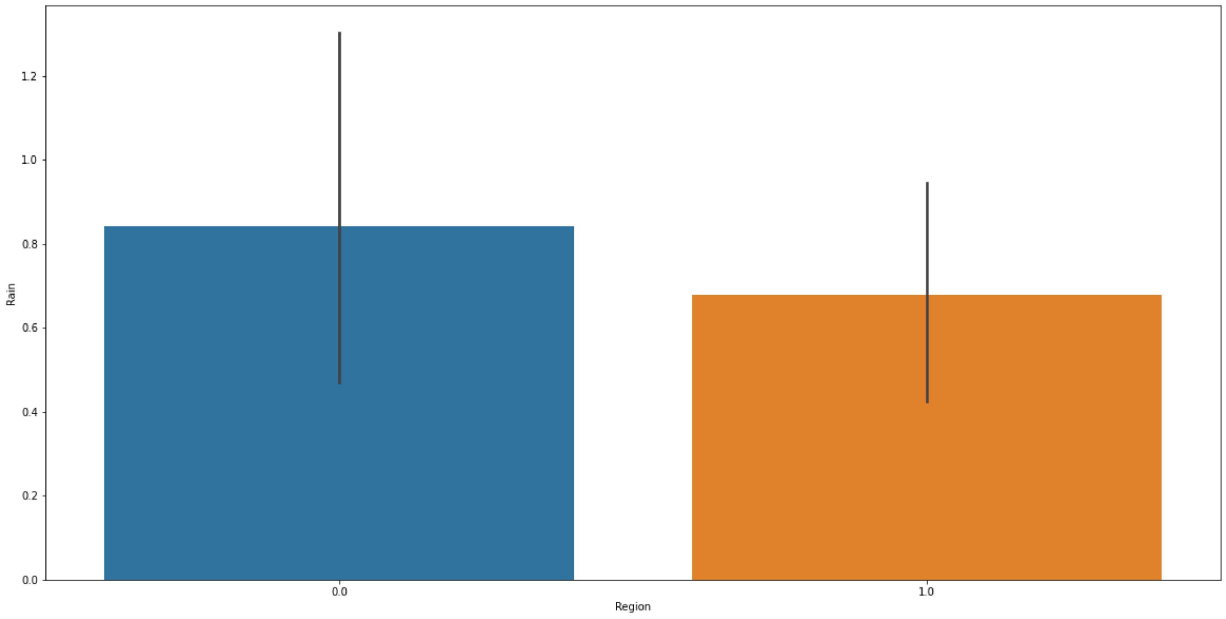
Observation

- Highest temperature is 42,40,37

Which region is mostly effected by rain

In [35]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

sns.barplot(x="Region",y="Rain",data=data)
```

Out[35]: <AxesSubplot:xlabel='Region', ylabel='Rain'>



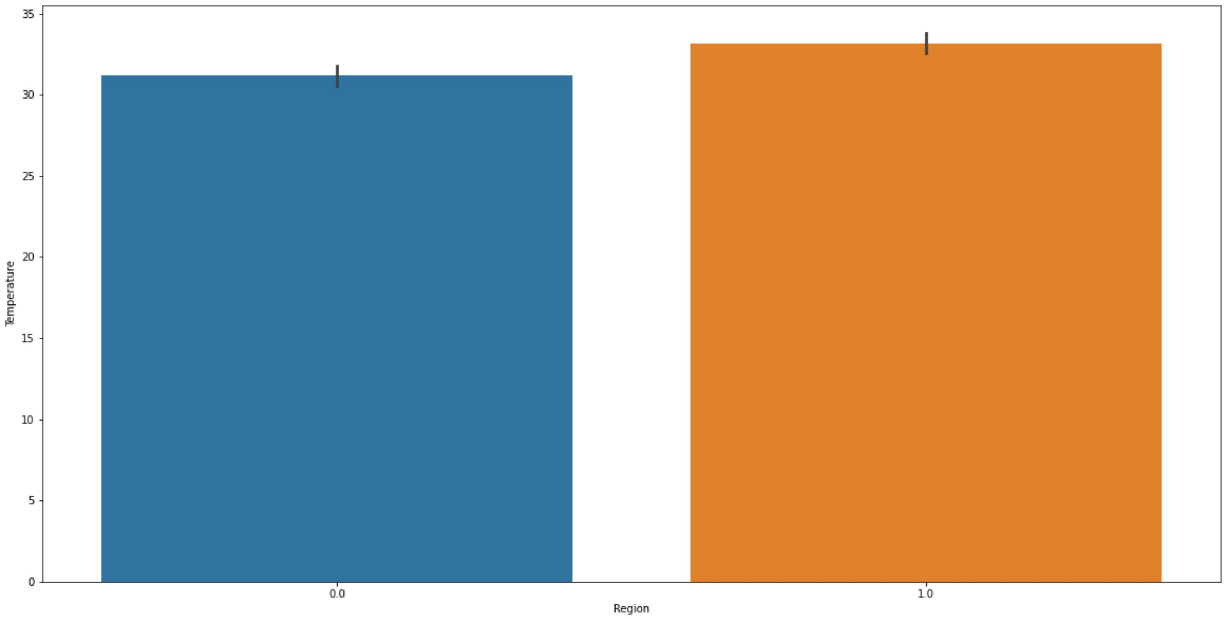
Observation

- Bejaia is the region in which most of the time rain happens

Which region is highly effected by Temperature

In [36]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

sns.barplot(x="Region",y="Temperature",data=data)
```

Out[36]: <AxesSubplot:xlabel='Region', ylabel='Temperature'>



Observation

- Sidi-Bel Abbes region mostly effected by Temperature

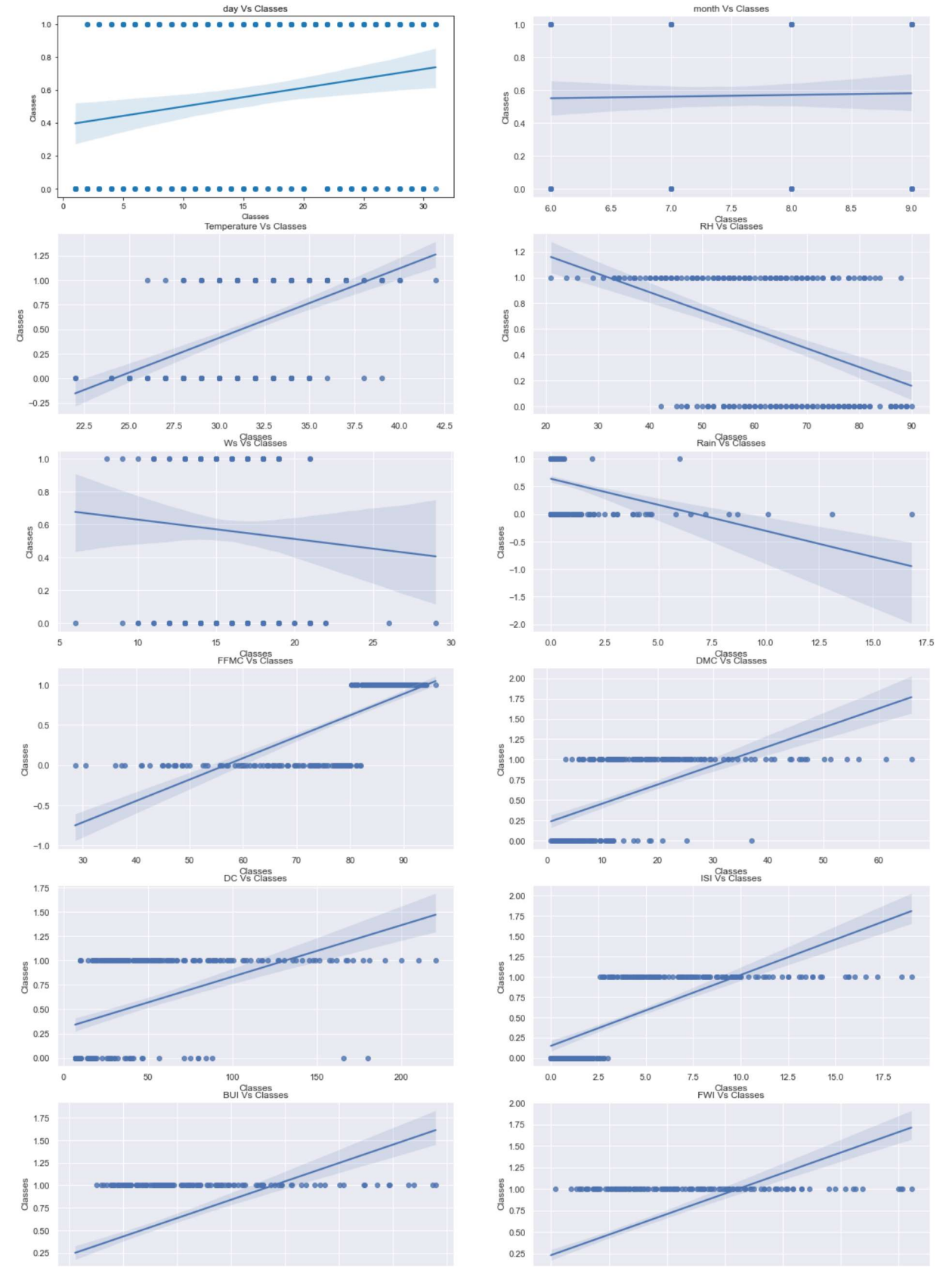
Reg plot

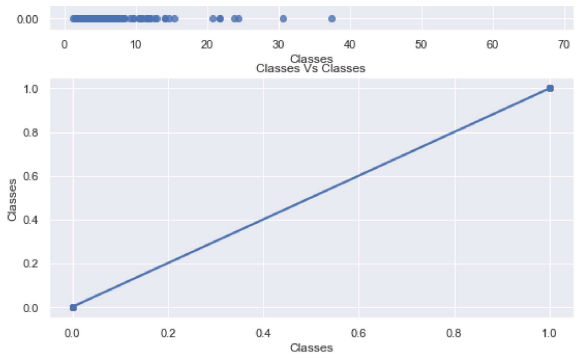
In [37]:
```
num_col=[feature for feature in data.columns if data[feature].dtype != 'O']
num_col
```

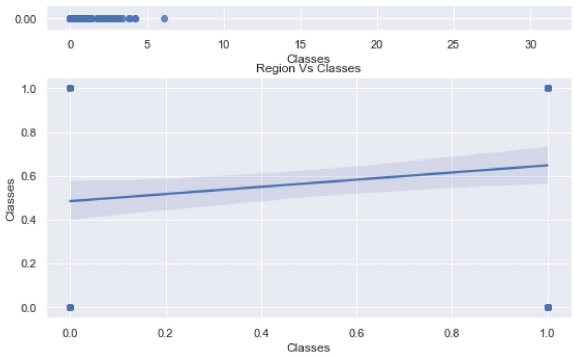
Out[37]:
```
['day',
 'month',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Classes',
 'Region']
```

In [38]:

```
plt.figure(figsize=(20,40))
for i in enumerate(num_col):
    plt.subplot(8,2,i[0]+1)
    sns.set(rc={'figure.figsize':(8,10)})
    sns.regplot(data=data,x=i[1],y='Classes')
    plt.xlabel('Classes')
    plt.title('{} Vs Classes'.format(i[1]))
```



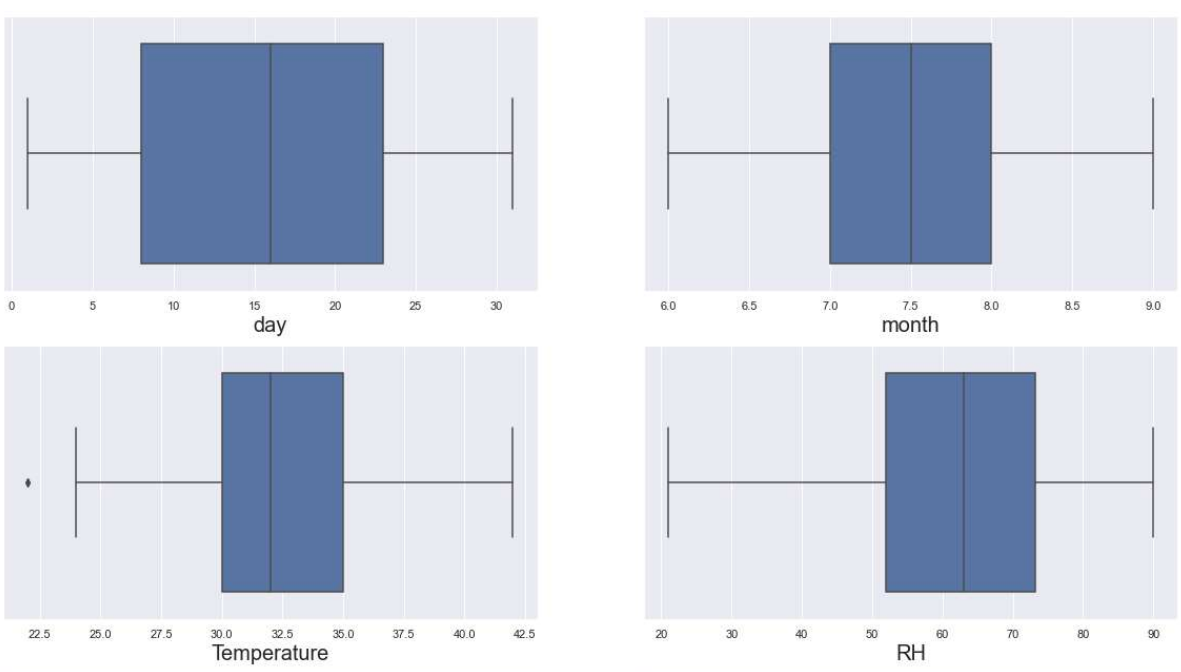




Boxplot to find outliers

In [39]:

```
plt.figure(figsize=(20,45), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=15 :     # as there are 15 columns in the data
        ax = plt.subplot(8,2,plotnumber)
        sns.boxplot(data[column])
        plt.xlabel(column,fontsize=20)

    plotnumber+=1
plt.show()
```



Observation

- Ws, Rain, FFMC, DMC BUI has many outliers

Dropping the outliers

In [40]:

```
def outliers_imputation_mild(data,column):
    IQR=data[column].quantile(0.75)-data[column].quantile(0.25)
    lower_fence=data[column].quantile(0.25)-(IQR*1.5)
    upper_fence=data[column].quantile(0.75)+(IQR*1.5)
    print("IQR:",IQR)
    print(f"Lower Fence {column}:",lower_fence)
    print(f"Upper Fence {column}:",upper_fence)
    print("_____________________________________________")
    data.loc[data[column]<=lower_fence,column]=lower_fence
    data.loc[data[column]>=upper_fence,column]=upper_fence
```

In [41]:

```
columns=data.columns
```

In [42]:
```
for col in columns:
    outliers_imputation_mild(data,col)
```

```
IQR: 15.0
Lower Fence day: -14.5
Upper Fence day: 45.5
```

```
IQR: 1.0
Lower Fence month: 5.5
Upper Fence month: 9.5
```

```
IQR: 5.0
Lower Fence Temperature: 22.5
Upper Fence Temperature: 42.5
```

```
IQR: 21.25
Lower Fence RH: 20.125
Upper Fence RH: 105.125
```

```
IQR: 3.0
Lower Fence Ws: 9.5
Upper Fence Ws: 21.5
```

```
IQR: 0.5
Lower Fence Rain: -0.75
Upper Fence Rain: 1.25
```

```
IQR: 16.224999999999994
Lower Fence FFMC: 47.73750000000001
Upper Fence FFMC: 112.63749999999999
```

```
IQR: 14.95
Lower Fence DMC: -16.624999999999996
Upper Fence DMC: 43.175
```

```
IQR: 54.87500000000001
Lower Fence DC: -69.03750000000002
Upper Fence DC: 150.46250000000003
```

```
IQR: 5.9
Lower Fence ISI: -7.450000000000001
Upper Fence ISI: 16.150000000000002
```

```
IQR: 16.525
Lower Fence BUI: -18.787499999999998
Upper Fence BUI: 47.3125
```

```
IQR: 10.675
Lower Fence FWI: -15.312500000000004
Upper Fence FWI: 27.387500000000003
```

```
IQR: 1.0
Lower Fence Classes: -1.5
Upper Fence Classes: 2.5
```

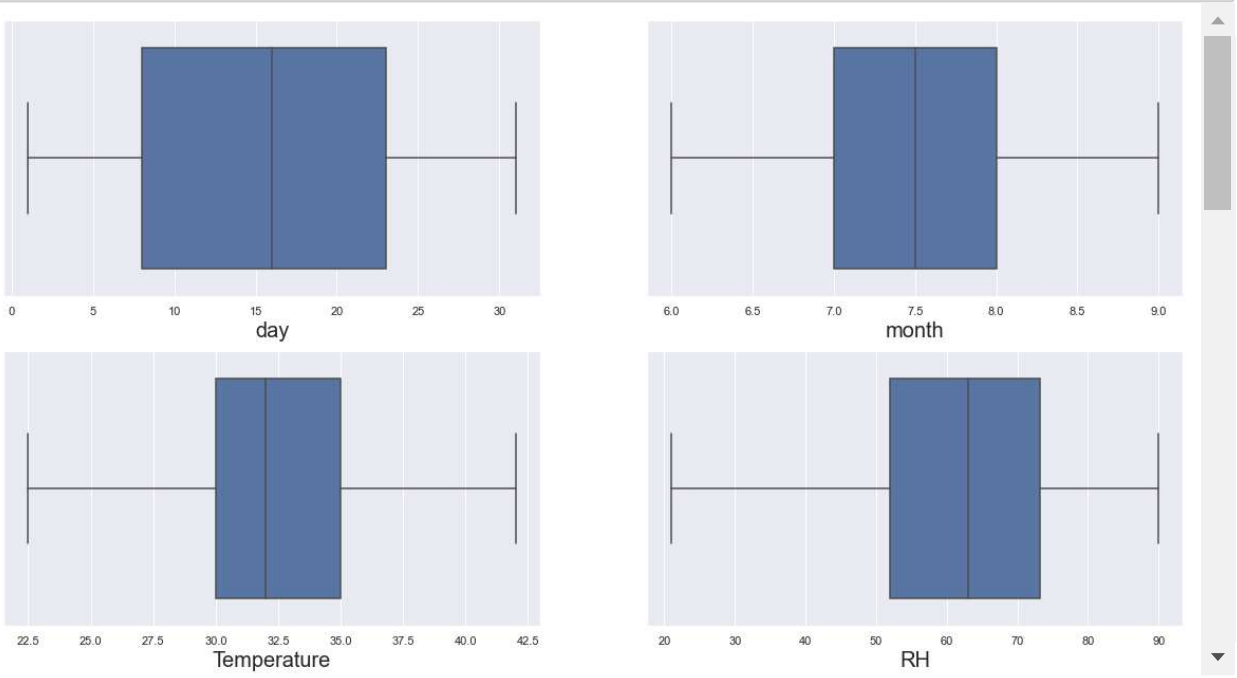
```
IQR: 1.0
Lower Fence Region: -1.5
```

```
Upper Fence Region: 2.5
```

Rechecking the outliers after dropping it

In [43]:

```
plt.figure(figsize=(20,45), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=15 :     # as there are 15 columns in the data
        ax = plt.subplot(8,2,plotnumber)
        sns.boxplot(data[column])
        plt.xlabel(column,fontsize=20)

    plotnumber+=1
plt.show()
```



Observation

- Outlier is not present in any of the features

Creating Independent and Dependent Features

In [131]:

```
X = data.drop(columns = ['Classes'])
y = data['Classes']
```

Independent Features

In [45]:
```
1 X
```

Out[45]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	1.0	6.0	29.0	57.0	18.0	0.00	65.7000	3.4	7.6	1.3	3.4	0.5	0.0
1	2.0	6.0	29.0	61.0	13.0	1.25	64.4000	4.1	7.6	1.0	3.9	0.4	0.0
2	3.0	6.0	26.0	82.0	21.5	1.25	47.7375	2.5	7.1	0.3	2.7	0.1	0.0
3	4.0	6.0	25.0	89.0	13.0	1.25	47.7375	1.3	6.9	0.0	1.7	0.0	0.0
4	5.0	6.0	27.0	77.0	16.0	0.00	64.8000	3.0	14.2	1.2	3.9	0.5	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	26.0	9.0	30.0	65.0	14.0	0.00	85.4000	16.0	44.5	4.5	16.9	6.5	1.0
240	27.0	9.0	28.0	87.0	15.0	1.25	47.7375	6.5	8.0	0.1	6.2	0.0	1.0
241	28.0	9.0	27.0	87.0	21.5	0.50	47.7375	3.5	7.9	0.4	3.4	0.2	1.0
242	29.0	9.0	24.0	54.0	18.0	0.10	79.7000	4.3	15.2	1.7	5.1	0.7	1.0
243	30.0	9.0	24.0	64.0	15.0	0.20	67.3000	3.8	16.5	1.2	4.8	0.5	1.0

244 rows × 13 columns

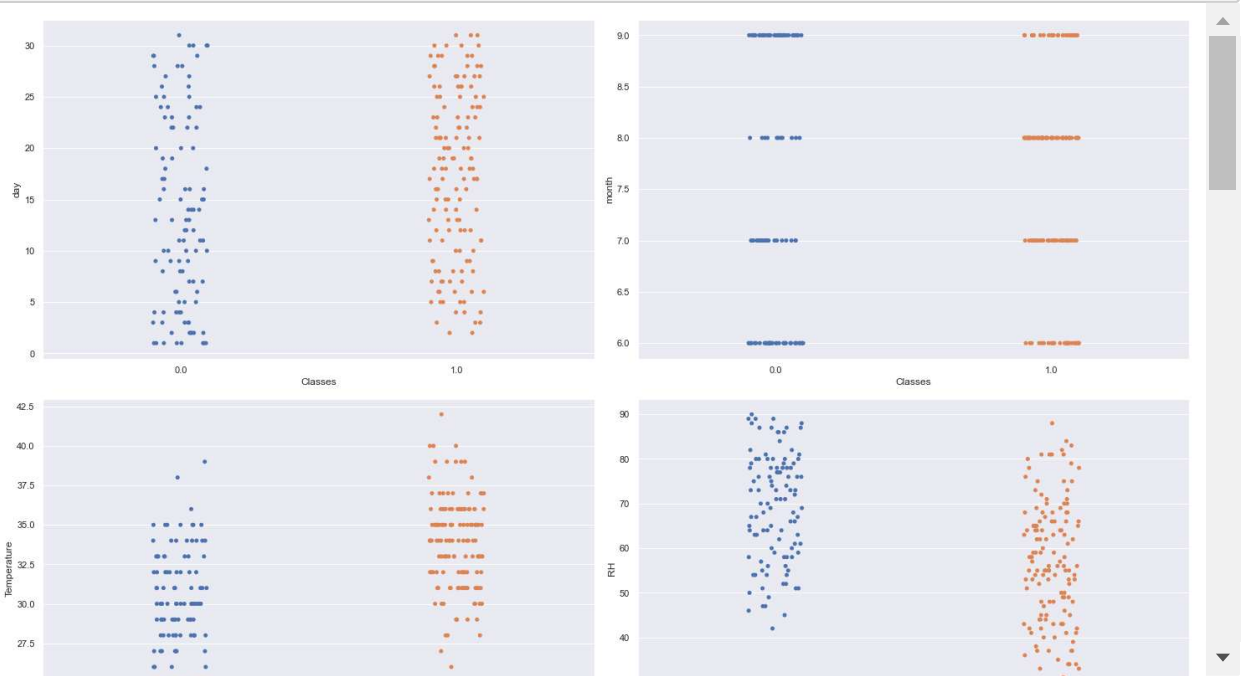
Dependent Features

In [46]:
```
1 y
```

Out[46]:
```
0      0.0
1      0.0
2      0.0
3      0.0
4      0.0
      ...
239    1.0
240    0.0
241    0.0
242    0.0
243    0.0
Name: Classes, Length: 244, dtype: float64
```

Visualizing the relationship between our independent and dependent Features

In [47]:
```
plt.figure(figsize=(20,50), facecolor='white')
plotnumber = 1

for column in X:
    if plotnumber<=15 :
        ax = plt.subplot(8,2,plotnumber)
        sns.stripplot(y,X[column])
    plotnumber+=1
plt.tight_layout()
```



Importing sklearn libraries for Machine Learning

In [48]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import  LogisticRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc
```

Train test split

In [49]:
```
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_sta
```

Logistic Regression Model Training

In [50]:
```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
```

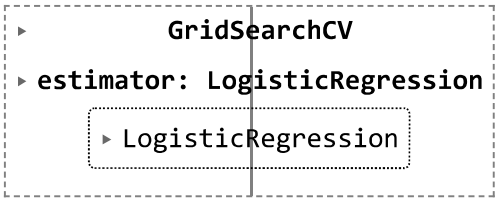
In [51]:
```
from sklearn.model_selection import GridSearchCV
parameter={'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,40,5
```

In [52]:
```
classifier_regressor=GridSearchCV(classifier,param_grid=parameter,scoring='a
```

Standardizing or Feature Selection

In [53]:
```
classifier_regressor.fit(X_train,y_train)
```

Out[53]:



In [54]:
```
print(classifier_regressor.best_params_)  ## Best parameter
```

```
{'C': 50, 'max_iter': 200, 'penalty': 'l2'}
```

In [55]:
```
print(classifier_regressor.best_score_)  ## Best Score
```

```
0.9615615615615617
```

Prediction

In [56]:
```
y_pred = classifier_regressor.predict(X_test)
```

In [57]:
```
y_pred
```

Out[57]:
```
array([1., 1., 1., 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1.,
       0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 0., 1., 1.,
       0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 0., 1., 1., 1.,
       0., 1., 1., 1., 1., 1., 0., 0., 1., 1.])
```

Acuracy score

In [58]:
```

from sklearn.metrics import accuracy_score,classification_report
score=accuracy_score(y_pred,y_test)
print(score)
```

```
0.9672131147540983
```

Classification Report

In [59]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.92      0.96        26
         1.0       0.95      1.00      0.97        35

    accuracy                           0.97        61
   macro avg       0.97      0.96      0.97        61
weighted avg       0.97      0.97      0.97        61
```

Performance Metrics

Confusion Metrics

In [60]:
```
conf_mat=confusion_matrix(y_pred,y_test)
```

In [61]:
```
conf_mat
```

Out[61]:
```
array([[24,  2],
       [ 0, 35]], dtype=int64)
```

In [62]:
```
true_positive = conf_mat[0][0]
false_positive = conf_mat[0][1]
false_negative = conf_mat[1][0]
true_negative = conf_mat[1][1]
```

Breaking down the formula for Accuracy

In [63]:
```

Accuracy = (true_positive + true_negative) / (true_positive +false_positive
Accuracy
```

Out[63]:
```
0.9672131147540983
```

Precision

In [64]:
```
Precision = true_positive/(true_positive+false_positive)
Precision
```

Out[64]:
```
0.9230769230769231
```

Recall

In [65]:
```
Recall = true_positive/(true_positive+false_negative)
Recall
```

Out[65]: 1.0

F1 Score

In [66]:
```
F1_Score = 2*(Recall * Precision) / (Recall + Precision)

F1_Score
```

Out[66]: 0.9600000000000001

Area Under Curve

In [67]:
```
auc = roc_auc_score(y_test, y_pred)
auc
```

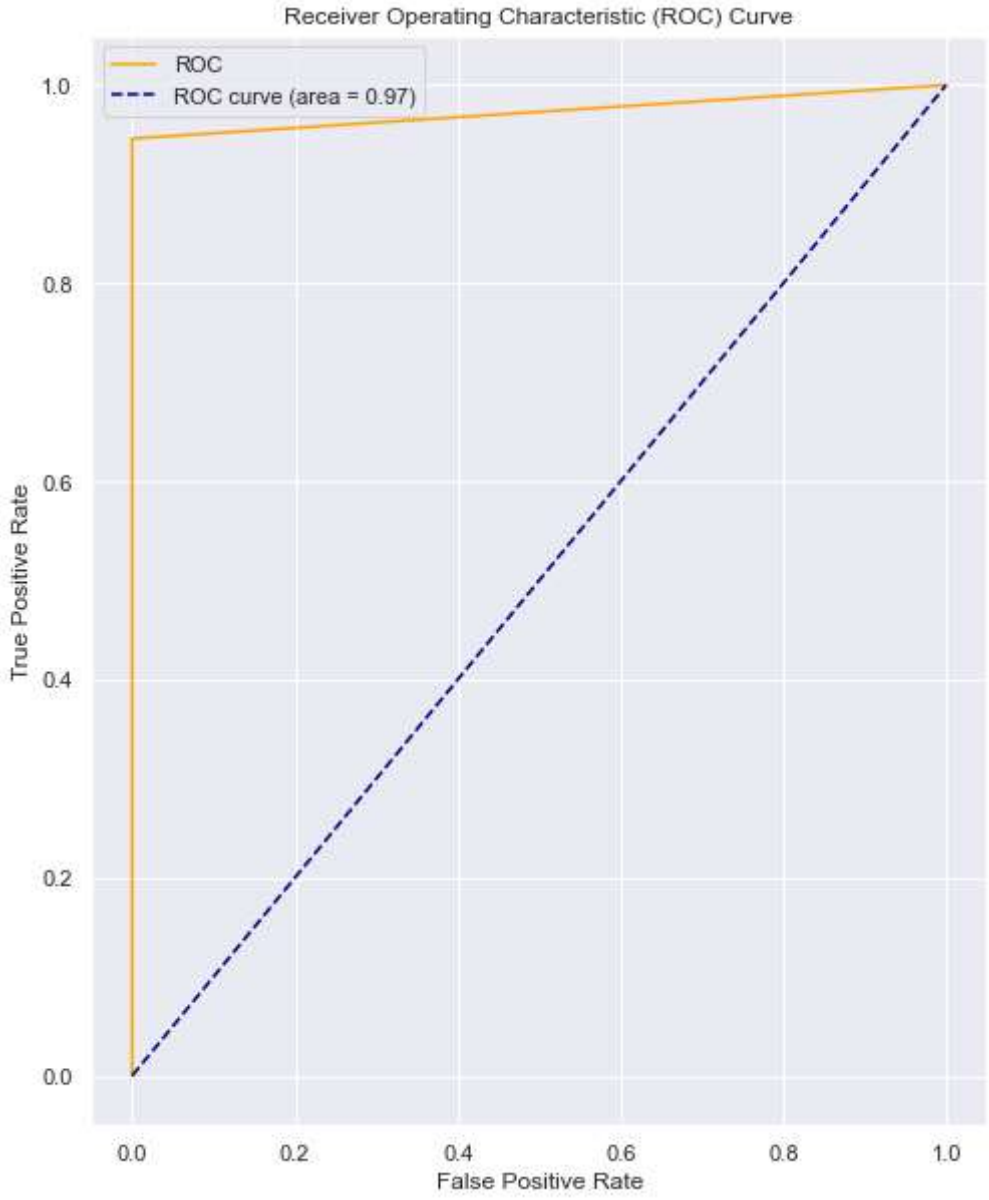
Out[67]: 0.972972972972973

Roc

In [68]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

In [69]:

```
plt.plot(fpr, tpr, color='orange', label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



What is the significance of Roc curve and AUC?

In real life, we create various models using different algorithms that we can use for classification purpose. We use AUC to determine which model is the best one to use for a given dataset. Suppose we have created Logistic regression, SVM as well as a clustering model for classification purpose. We will calculate AUC for all the models seperately. The model with highest AUC value will be the best model to use.

Creating Imbalance dataset from the original balanced dataset

In [70]:
```
1 df.head()
```

Out[70]:

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

In [71]:
```
1 df.shape
```

Out[71]:
```
(244, 14)
```

In [72]:
```
1 ### Creating imbalance
2 ### 1. splitting data in 90:10 percent ratio using train test split
3 X1 = pd.DataFrame(df, columns = ['day', 'month','Temperature','RH','Ws','Rai
4 y1=pd.DataFrame(df,columns = ['Classes'])
```

In [73]:
```
1 X_train_imb, X_test_imb, y_train_imb, y_test_imb = train_test_split(X1, y1,
2
```

In [74]:
```
## Both will have same shape
X_train_imb.shape, y_train_imb.shape
```

Out[74]: ((219, 13), (219, 1))

Replacing all values as 1 in y_train and all values as zero in y_test to create imbalance

In [75]:
```
y_train_imb=y_train_imb.replace(0,1)
y_train_imb.head()
```

Out[75]:

	Classes
156	1
183	1
11	1
75	1
130	1

In [76]:
```
y_test_imb=y_test_imb.replace(1,0)
y_test_imb.head()

```

Out[76]:

	Classes
48	0
216	0
101	0
38	0
86	0

In [77]:
```
X_train_imb.head()
```

Out[77]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
156	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0
183	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0
11	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0
75	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0
130	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0

In [78]:
```
### Combining X_train_imb and y_train_imb
train_imb=X_train_imb.join(pd.DataFrame(y_train_imb))
train_imb.head()

```

Out[78]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Cla
156	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0	
183	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0	
11	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0	
75	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0	
130	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0	

In [79]:
```
### Combining X_test_imb and y_test_imb
test_imb=X_test_imb.join(pd.DataFrame(y_test_imb))
test_imb.head()
```

Out[79]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Class
48	19	7	35	59	17	0.0	88.1	12.0	52.8	7.7	18.2	10.9	0.0	
216	3	9	28	75	16	0.0	82.2	4.4	24.3	3.3	6.0	2.5	1.0	
101	10	9	33	73	12	1.8	59.9	2.2	8.9	0.7	2.7	0.3	0.0	
38	9	7	32	68	14	1.4	66.6	7.7	9.2	1.1	7.4	0.6	0.0	
86	26	8	31	78	18	0.0	85.8	45.6	190.6	4.7	57.1	13.7	0.0	

In [80]:
```
### Checking the shape of imbalanced Data
train_imb.shape, test_imb.shape
```

Out[80]:
```
((219, 14), (25, 14))
```

In [81]:
```
### Combining train_imb dataset and test_imb dataset into data_imb dataset
df_imb=pd.concat([train_imb, test_imb], ignore_index=True, sort=False)
df_imb.head()
```

Out[81]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Classes
0	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0	1
1	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0	1
2	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0	1
3	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0	1
4	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0	1

In [82]:
```
1 df_imb.shape
```

Out[82]:
```
(244, 14)
```

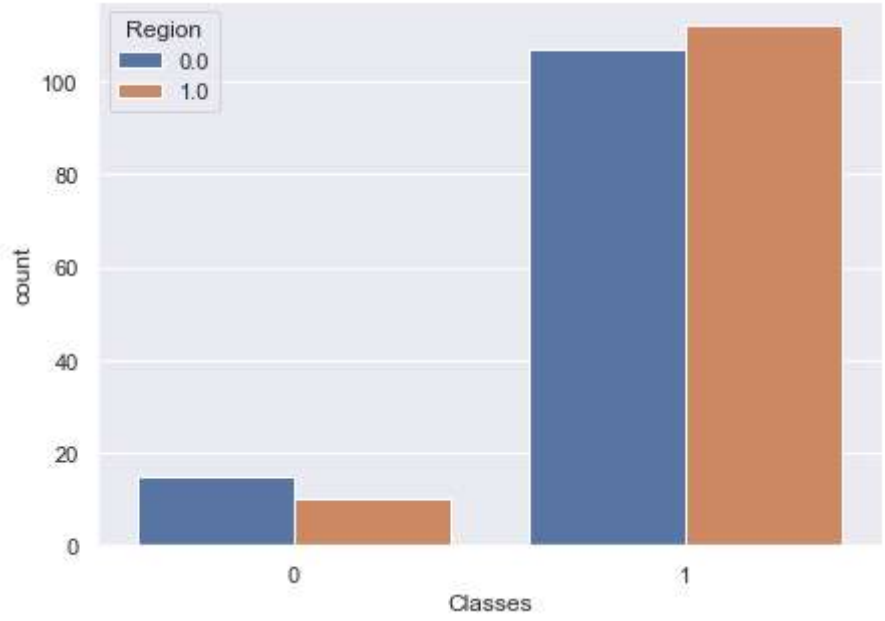
Checking the imbalancing

In [83]:
```
1 df_imb.Classes.value_counts()
```

Out[83]:
```
1    219
0     25
Name: Classes, dtype: int64
```

In [84]:
```
1 ## 0 is 'Bejaia' and 1 is 'Sidi Bel-abbes region'
2 plt.figure(figsize=(7,5))
3 sns.countplot(data=df_imb,x='Classes',hue='Region')
```

Out[84]:
```
<AxesSubplot:xlabel='Classes', ylabel='count'>
```



Logistic Regression on imbalanced Dataset

In [85]:
```
1 df_imb.head()
```

Out[85]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Classes
0	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0	1
1	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0	1
2	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0	1
3	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0	1
4	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0	1

Separating Independent and Dependent feature

In [86]:
```
X1 = df_imb.drop(columns = ['Classes'])
y1 = df_imb['Classes']
```

In [87]:
```
X1
```

Out[87]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0
1	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0
2	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0
3	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0
4	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	26	8	33	37	16	0.0	92.2	61.3	167.2	13.1	64.0	30.3	1.0
240	1	6	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0.0
241	2	9	28	67	19	0.0	75.4	2.9	16.3	2.0	4.0	0.8	1.0
242	11	8	35	63	13	0.0	88.9	21.7	77.0	7.1	25.5	12.1	0.0
243	9	8	39	43	12	0.0	91.7	16.5	30.9	9.6	16.4	12.7	1.0

244 rows × 13 columns

In [88]:
```
y1
```

Out[88]:
```
0      1
1      1
2      1
3      1
4      1
      ..
239    0
240    0
241    0
242    0
243    0
Name: Classes, Length: 244, dtype: int64
```

9.2 Handling Imbalance dataset by Doing Upsampling

In [92]:
```
### for upsampling
from imblearn.combine import SMOTETomek
```

In [93]:
```
smk=SMOTETomek()
smk
```

Out[93]:
```
SMOTETomek()
```

In [94]:
```
X_bal,y_bal=smk.fit_resample(X1,y1)
```

In [95]:
```
X_bal.head()
```

Out[95]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0
1	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0
2	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0
3	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0
4	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0

In [96]:
```
y_bal.head()
```

Out[96]:
```
0    1
1    1
2    1
3    1
4    1
Name: Classes, dtype: int64
```

In [97]:
```
X_bal.shape,y_bal.shape
```

Out[97]:
```
((418, 13), (418,))
```

In [98]:
```
## Creating Balanced data from imbalanced data
data_bal=X_bal.join(pd.DataFrame(y_bal))
data_bal.head()
```

Out[98]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Classes
0	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0	1
1	1	8	38	52	14	0.0	78.3	4.4	10.5	2.0	4.4	0.8	1.0	1
2	12	6	26	81	19	0.0	84.0	13.8	61.4	4.8	17.7	7.1	0.0	1
3	15	8	36	55	13	0.3	82.4	15.6	92.5	3.7	22.0	6.3	0.0	1
4	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0	1

EDA on balanced Dataset

In [99]:
```
1 data_bal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          418 non-null    int64  
 1   month        418 non-null    int64  
 2   Temperature  418 non-null    int64  
 3   RH           418 non-null    int64  
 4   Ws           418 non-null    int64  
 5   Rain         418 non-null    float64
 6   FFMC         418 non-null    float64
 7   DMC          418 non-null    float64
 8   DC           418 non-null    float64
 9   ISI          418 non-null    float64
 10  BUI          418 non-null    float64
 11  FWI          418 non-null    float64
 12  Region       418 non-null    float64
 13  Classes      418 non-null    int64  
dtypes: float64(8), int64(6)
memory usage: 45.8 KB
```

Statistical analysis on Balanced Dataset

In [100]:
```
1 data_bal.describe().T
```

Out[100]:

	count	mean	std	min	25%	50%	75%	max
day	418.0	14.696172	8.807618	1.0	7.000000	15.000000	22.000000	31.0
month	418.0	7.480861	1.015877	6.0	7.000000	7.000000	8.000000	9.0
Temperature	418.0	31.980861	3.214866	22.0	30.000000	32.000000	34.000000	42.0
RH	418.0	62.775120	13.849037	21.0	54.000000	64.000000	73.000000	90.0
Ws	418.0	15.521531	2.500986	6.0	14.000000	16.000000	17.000000	29.0
Rain	418.0	0.533750	1.568869	0.0	0.000000	0.000252	0.350145	16.8
FFMC	418.0	78.810110	12.492976	28.6	72.900000	83.163600	87.695431	96.0
DMC	418.0	15.231814	13.187586	0.7	4.571247	12.100000	21.038634	65.9
DC	418.0	53.835568	49.403601	6.9	15.731636	36.900000	80.997993	220.4
ISI	418.0	4.760849	3.813000	0.0	1.525000	3.782504	7.178022	19.0
BUI	418.0	17.693459	15.203004	1.1	5.800000	14.200000	24.108046	68.0
FWI	418.0	7.207544	7.206697	0.0	0.800000	4.865415	11.892403	31.1
Region	418.0	0.440332	0.455094	0.0	0.000000	0.206118	1.000000	1.0
Classes	418.0	0.500000	0.500599	0.0	0.000000	0.500000	1.000000	1.0

In [101]:
```
1 data_bal.corr()
```

Out[101]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DM
day	1.000000	-0.108529	0.211355	-0.176460	0.124351	-0.063525	0.270341	0.5934
month	-0.108529	1.000000	-0.146233	0.103669	-0.037592	0.015503	0.016336	-0.0128
Temperature	0.211355	-0.146233	1.000000	-0.664806	-0.225132	-0.259071	0.649440	0.4360
RH	-0.176460	0.103669	-0.664806	1.000000	0.160422	0.171421	-0.596665	-0.3534
Ws	0.124351	-0.037592	-0.225132	0.160422	1.000000	0.107410	-0.048108	0.1412
Rain	-0.063525	0.015503	-0.259071	0.171421	0.107410	1.000000	-0.519199	-0.2347
FFMC	0.270341	0.016336	0.649440	-0.596665	-0.048108	-0.519199	1.000000	0.5839
DMC	0.593453	-0.012879	0.436084	-0.353486	0.141288	-0.234785	0.583924	1.0000
DC	0.625145	0.047628	0.354880	-0.231208	0.215898	-0.264828	0.538637	0.9032
ISI	0.289943	0.038870	0.626105	-0.674726	0.099356	-0.320360	0.762807	0.6472
BUI	0.618514	0.001046	0.419888	-0.317463	0.175456	-0.250724	0.586998	0.9860
FWI	0.466327	0.037585	0.562470	-0.552254	0.141710	-0.291710	0.708014	0.8619
Region	-0.005290	0.035952	0.210250	-0.365080	-0.110678	-0.014956	0.204390	0.1531
Classes	0.138421	-0.044798	0.020861	-0.061225	-0.019154	0.202278	-0.113698	-0.0901

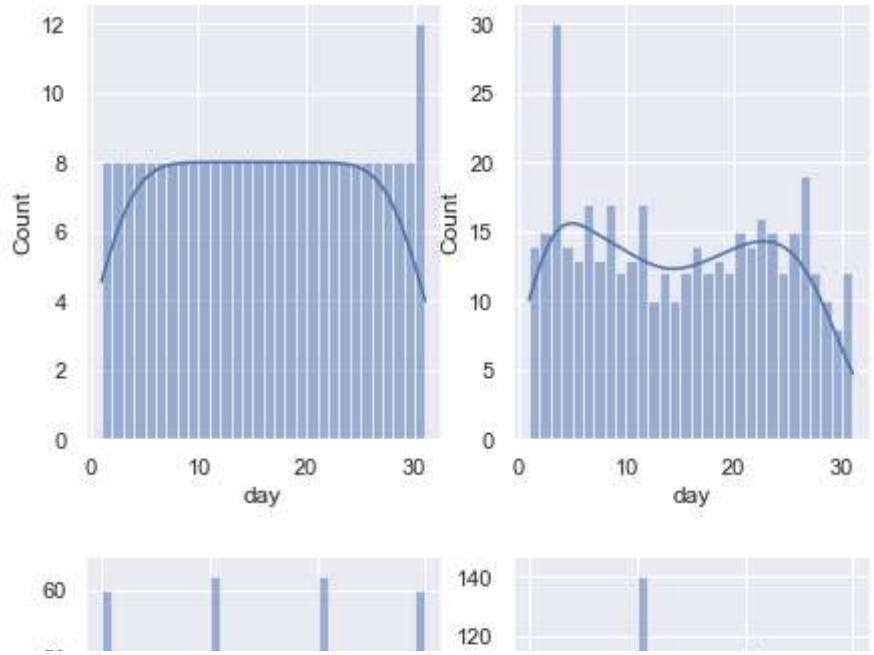
In [103]:
```
1 num_bal_col=[feature for feature in data_bal.columns if data_bal[feature].dt
2 num_bal_col
```

Out[103]:
```
['day',
 'month',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Region',
 'Classes']
```

Compairing the feature for Original and Balanced Dataset

In [104]:

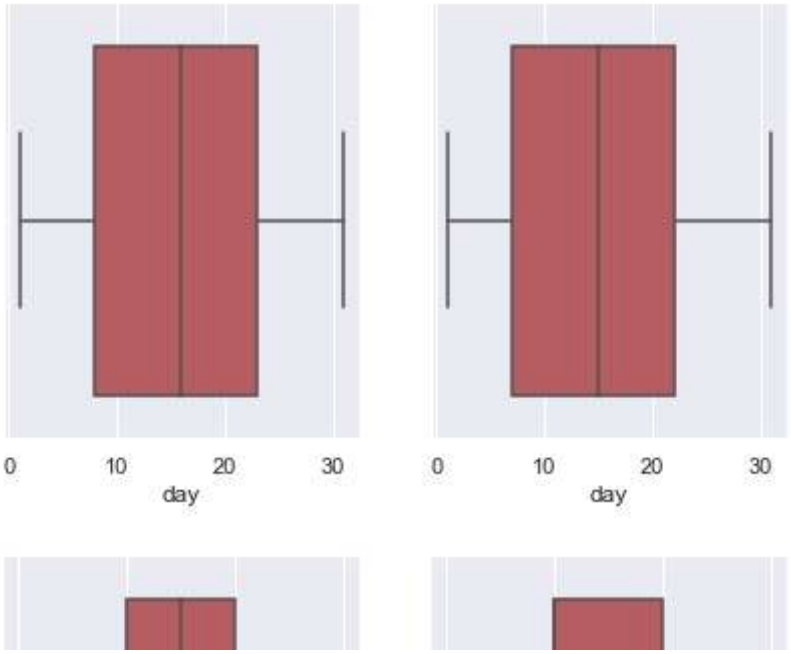
```
for i in num_col:
    plt.figure(figsize=(7,4))
    plt.subplot(121)
    sns.histplot(data=data,x=i,kde=True,bins=30)


    plt.subplot(122)
    sns.histplot(data=data_bal,x=i,kde=True,bins=30)

```



Checking the Outliers for Original and Balanced Dataset

In [134]:

```
for i in num_col:
    plt.figure(figsize=(7,4))
    plt.subplot(121)
    sns.boxplot(data=data,x=i,color='r')


    plt.subplot(122)
    sns.boxplot(data=data_bal,x=i,color='r')
```



Test train Split

In [106]:

```
from sklearn.model_selection import train_test_split
X_train1,X_test1,y_train1,y_test1=train_test_split(X_bal,y_bal,test_size=0.3
```

In [107]:
```
1 X_train1
```

Out[107]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	
7	2	7	33	48	16	0.000000	87.600000	7.900000	17.800000	6.800000	
246	21	7	33	63	16	0.891180	75.856370	21.051869	60.472667	2.644101	2
405	17	8	35	54	17	0.000000	89.385340	20.049842	109.906082	9.623770	2
351	4	9	28	76	15	0.000000	80.867762	4.185890	23.384087	3.050205	
333	28	6	32	48	13	0.377013	79.216508	18.640666	81.723792	2.142240	2
...	...	...	...	...	...	...	...	...	...	...	
321	25	6	32	49	13	0.244161	81.575172	19.014230	83.022636	3.112038	2
69	15	6	28	90	15	0.000000	66.800000	7.200000	14.700000	1.200000	
121	6	7	32	63	14	0.000000	87.000000	10.900000	37.000000	5.600000	1
238	22	7	31	67	17	0.000000	87.749003	44.492435	98.844223	7.532269	4
169	23	8	36	53	16	0.000000	89.500000	37.600000	161.500000	10.400000	4

292 rows × 13 columns

In [108]:
```
1 X_test1
```

Out[108]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	
307	20	8	35	37	17	0.000000	91.766036	23.042046	99.407826	13.164552	2
310	23	7	31	71	17	0.000000	87.139229	46.492820	108.817720	6.664203	4
414	8	7	37	42	12	0.098951	89.391152	16.170165	35.583663	7.851872	1
163	28	7	33	57	16	0.000000	87.500000	15.700000	37.600000	6.700000	1
178	12	6	27	58	17	0.000000	88.900000	21.300000	37.800000	8.700000	2
...	...	...	...	...	...	...	...	...	...	...	
199	8	8	32	60	18	0.300000	77.100000	11.300000	47.000000	2.200000	1
12	14	8	37	40	13	0.000000	91.900000	22.300000	55.500000	10.800000	2
339	11	9	29	84	12	0.434024	68.323518	2.503550	14.743341	1.079438	
133	14	8	33	66	14	0.000000	87.000000	21.700000	94.700000	5.700000	2
46	6	9	29	74	19	0.100000	75.800000	3.600000	32.200000	2.100000	

126 rows × 13 columns

In [109]:
```
1 y_train1
```

Out[109]:
```
7      1
246    0
405    0
351    0
333    0
      ..
321    0
69     1
121    1
238    0
169    1
Name: Classes, Length: 292, dtype: int64
```

In [110]:
```
1 y_test1
```

Out[110]:
```
307    0
310    0
414    0
163    1
178    1
      ..
199    1
12     1
339    0
133    1
46     1
Name: Classes, Length: 126, dtype: int64
```

Logistic Regression Model

In [111]:
```
1 from sklearn.linear_model import LogisticRegression
2 classifier_bal=LogisticRegression()
3 classifier_bal
```

Out[111]:

▾ LogisticRegression

LogisticRegression()

In [112]:
```
1 from sklearn.model_selection import GridSearchCV
2 parameter_bal={'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,
```

In [113]:
```
1 classifier_regressor_bal=GridSearchCV(classifier,param_grid=parameter,scorin
```

Standarizing or Feature Scaling

In [114]:
```
1 classifier_regressor_bal.fit(X_train1,y_train1)
```

Out[114]:

GridSearchCV

estimator: LogisticRegression

LogisticRegression

In [115]:
```
1 print(classifier_regressor_bal.best_params_)
```

```
{'C': 50, 'max_iter': 300, 'penalty': 'l2'}
```

In [116]:
```
1 print(classifier_regressor_bal.best_score_)
```

```
0.6436002337814143
```

Prediction

In [117]:
```
1 y_bal_pred = classifier_regressor_bal.predict(X_test1)
```

In [118]:
```
1 y_bal_pred
```

Out[118]:
```
array([1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0], dtype=int64)
```

Acurracy

In [119]:
```
1
2 from sklearn.metrics import accuracy_score,classification_report
3 bal_score=accuracy_score(y_bal_pred,y_test1)
4 print(bal_score)
```

```
0.6666666666666666
```

Classification Report

In [120]:
```
print(classification_report(y_bal_pred,y_test1))
```

```
              precision    recall  f1-score   support

           0       0.66      0.70      0.68        63
           1       0.68      0.63      0.66        63

    accuracy                           0.67       126
   macro avg       0.67      0.67      0.67       126
weighted avg       0.67      0.67      0.67       126
```

Performance Metrics

Confusion Metrics

In [121]:
```
conf_mat_bal=confusion_matrix(y_bal_pred,y_test1)
```

In [122]:
```
conf_mat_bal
```

Out[122]:
```
array([[44, 19],
       [23, 40]], dtype=int64)
```

In [123]:
```
true_positive = conf_mat_bal[0][0]
false_positive = conf_mat_bal[0][1]
false_negative = conf_mat_bal[1][0]
true_negative = conf_mat_bal[1][1]
```

Precision

In [124]:
```
bal_Precision = true_positive/(true_positive+false_positive)
bal_Precision
```

Out[124]:
```
0.6984126984126984
```

Recall

In [125]:
```
bal_recall = true_positive/(true_positive+false_negative)
bal_recall
```

Out[125]:
```
0.6567164179104478
```

F1 score

In [126]:
```
F1_Score_bal = 2*(bal_recall * bal_Precision) / (bal_recall + bal_Precision)
F1_Score_bal
```

Out[126]: 0.676923076923077

Conclusion

Performance of Logistic Model on Original Dataset

In [128]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.92      0.96        26
         1.0       0.95      1.00      0.97        35

    accuracy                           0.97        61
   macro avg       0.97      0.96      0.97        61
weighted avg       0.97      0.97      0.97        61
```

Performance of Logistic Model on Balanced Dataset which are created from imbalanced dataset

In [129]:
```
print(classification_report(y_bal_pred,y_test1))
```

```
              precision    recall  f1-score   support

           0       0.66      0.70      0.68        63
           1       0.68      0.63      0.66        63

    accuracy                           0.67       126
   macro avg       0.67      0.67      0.67       126
weighted avg       0.67      0.67      0.67       126
```

Observation

- It seems that model is good when we predict from original dataset
- It seems that model is very bad when we try to predict from balanced(created from an imbalanced dataset)

Thank You