ML Logistic Regression Practical Implementation on Algerian Forest Fire Prediction Dataset

EDA and Feature Engineering

Data Profiling

Data Cleaning

Statistical Analysis

Graphical Analysis

Data Scaling

Logistic Regression Implementation

Logistic Regression on Original Dataset

Performance metrics for above model

Creating an imbalanced dataset from original dataset

Balancing the imbalanced Dataset

Logistic Regression on above dataset

Performance Metrics for above Dataset

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.

In [1]:
```
# importing the library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```
# importing the dataset
df=pd.read_csv(r"C:\Users\Mukul\Downloads\Algerian_forest_fires_dataset_UPDATE.cs
```

In [3]:
```
# top five dataset
df.head()
```

Out[3]:

	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

In [4]:
```
#last five dataset
df.tail()
```

Out[4]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire
242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire
243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire

In [5]:
```
# chech the index 122,123 need to be remove from dataset
df[121:130]
```

Out[5]:

	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	
125	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	
126	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	
127	04	06	2012	30	64	14	0	79.4	5.2	15.4	2.2	5.6	1	
128	05	06	2012	32	60	14	0.2	77.1	6	17.6	1.8	6.5	0.9	
129	06	06	2012	35	54	11	0.1	83.7	8.4	26.3	3.1	9.3	3.1	

In [6]:
```
#check the shape
df.shape
```

Out[6]: (246, 14)

In [7]:
```
# droping the rows having region
df.drop([122,123],inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
```

In [8]:
```
df.iloc[121:].head()
```

Out[8]:

	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

In [9]:
```
# check the shape
df.shape
```

Out[9]: (244, 14)

creating the Region feature

In [10]:
```
df.loc[:122,'Region']=0
df.loc[122:,'Region']=1

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

Out[10]:

	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

In [11]:
```
# check the info
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 [12]:
```
df.describe(include='all').T
```

Out[12]:

	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 [13]:
```
# here it is visible that same columns have space in the name like RH,WS , RAIN ,
df.columns
```

Out[13]:
```
Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain ', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes  ', 'Region'],
      dtype='object')
```

In [14]:
```
# stripping the space from the columns
df.columns=[col_name.strip() for col_name in df.columns]
df.columns
```

Out[14]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'Region'],
      dtype='object')
```

In [15]:
```
#  coverting all feature values to string so that we can do data cleaning
df=df.astype(str)
```

In [16]:
```
# some values in columns also have space
for feature in ['Rain','FFMC','DMC','DC','ISI','BUI','FWI','Classes']:
    df[feature]=df[feature].str.replace(" ","")
```

Check the Unique data point of FWI

In [17]:
```
df['FWI'].unique()
```

Out[17]:
```
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 inFWI columns

In [18]:
```
df[df['FWI']=='fire'].index
```

Out[18]:
```
Int64Index([165], dtype='int64')
```

In [19]:
```
df['FWI'].mode()
```

Out[19]:
```
0    0.4
Name: FWI, dtype: object
```

In [20]:
```
df.loc[165,'FWI']='0.4'
```

In [21]:
```
#replacing nan values with fire to make equal to the info given dataset
df[df['Classes']=='nan'].index
df.loc[165,'Classes']='fire'
```

In [22]:
```
df['Classes'].unique()
```

Out[22]:
```
array(['notfire', 'fire'], dtype=object)
```

In [23]:
```
df['Classes']=df['Classes'].str.replace('notfire','0')
df['Classes']=df['Classes'].str.replace('fire','1')
```

In [24]:
```
#df['Classes']=df['Classes'].str.replace('nan','0')
```

In [ ]:
```

```

In [ ]:
```

```

In [25]:
```
# droping year fetureas data is realated tp your 2012
df.drop('year',axis=1,inplace=True)
```

Changing the datatype Numerical to objects

In [26]:
```
datatype_convert={'day':'int64','month':'int64','Temperature':'int64','RH':'int64
                  'Rain':'float64','FFMC':'float64','DMC':'float64','DC':'float64'
                  'FWI':'float64','Classes':'int64','Region':'float64'}
df=df.astype(datatype_convert)
df.dtypes
```

Out[26]:
```
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: All the features are converted from categorical to numerical
datatype
```

In [27]:
```
df.shape
```

Out[27]:
```
(244, 14)
```

Checking Null values and Duplicated

In [28]:
```
df.isnull().sum()
```

Out[28]:
```
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 [29]:
```
df.duplicated().sum()
```

Out[29]:
```
0
```

Obsevation---> 1.There are no null values 2.Total rows 244 and columns are 14 3.There are no duplicated values

Creating the Copy of dataframe from Original Dataframe

In [30]:
```
data=df.copy()
data.head()
```

Out[30]:

	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 [31]:
```
df.describe()
```

Out[31]:

	day	month	Temperature	RH	Ws	Rain	FFMC	
count	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.0
mean	15.754098	7.500000	32.172131	61.938525	15.504098	0.760656	77.887705	14.6
std	8.825059	1.112961	3.633843	14.884200	2.810178	1.999406	14.337571	12.3
min	1.000000	6.000000	22.000000	21.000000	6.000000	0.000000	28.600000	0.7
25%	8.000000	7.000000	30.000000	52.000000	14.000000	0.000000	72.075000	5.8
50%	16.000000	7.500000	32.000000	63.000000	15.000000	0.000000	83.500000	11.3
75%	23.000000	8.000000	35.000000	73.250000	17.000000	0.500000	88.300000	20.7
max	31.000000	9.000000	42.000000	90.000000	29.000000	16.800000	96.000000	65.9

In [32]:
```
df.describe().T
```

Out[32]:

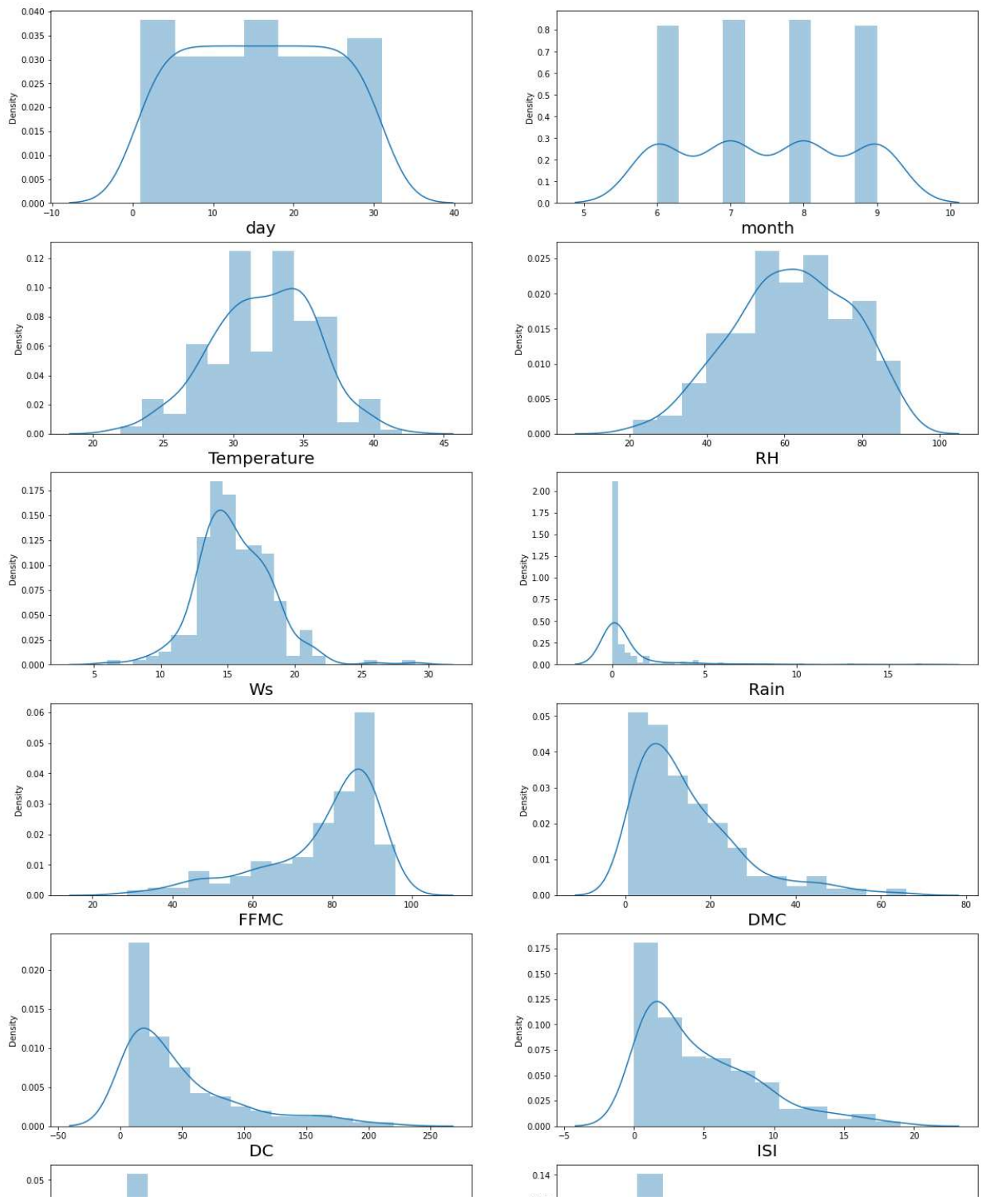
	count	mean	std	min	25%	50%	75%	max
day	244.0	15.754098	8.825059	1.0	8.000	16.00	23.000	31.0
month	244.0	7.500000	1.112961	6.0	7.000	7.50	8.000	9.0
Temperature	244.0	32.172131	3.633843	22.0	30.000	32.00	35.000	42.0
RH	244.0	61.938525	14.884200	21.0	52.000	63.00	73.250	90.0
Ws	244.0	15.504098	2.810178	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

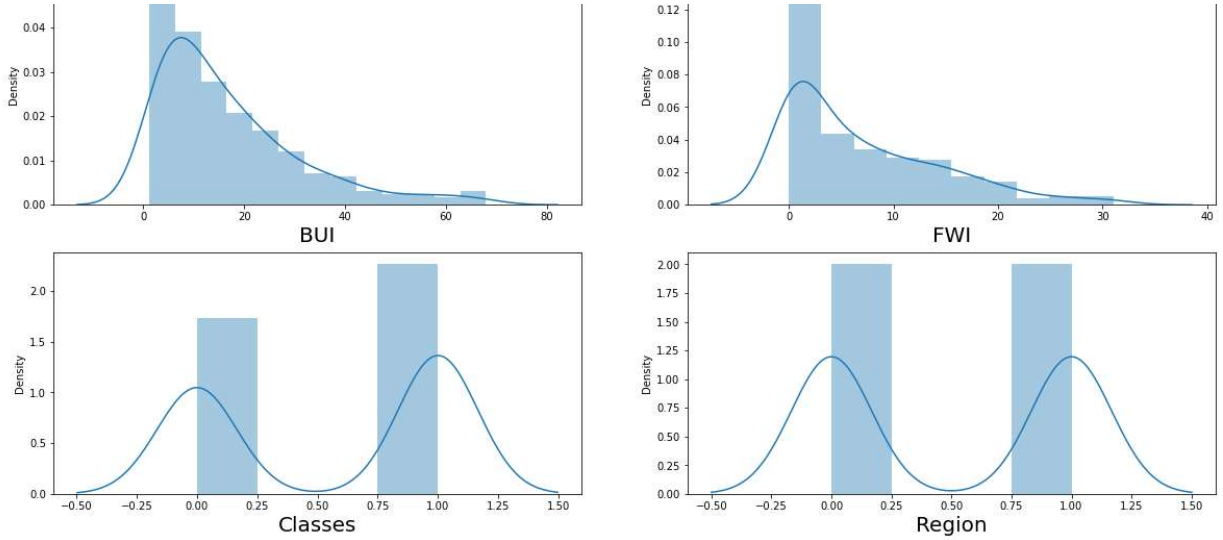
In [33]:
```
data.cov()
```

Out[33]:

	day	month	Temperature	RH	Ws	Rain	F
day	7.788167e+01	4.641920e-16	3.071308	-9.747689	1.165621	-1.980908	28.34
month	4.641920e-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	-2.193033e-17	0.000000e+00	0.497942	-3.030864	-0.248971	-0.041152	1.61

In [34]:
```
# cheking the distribution of the feature
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()
```



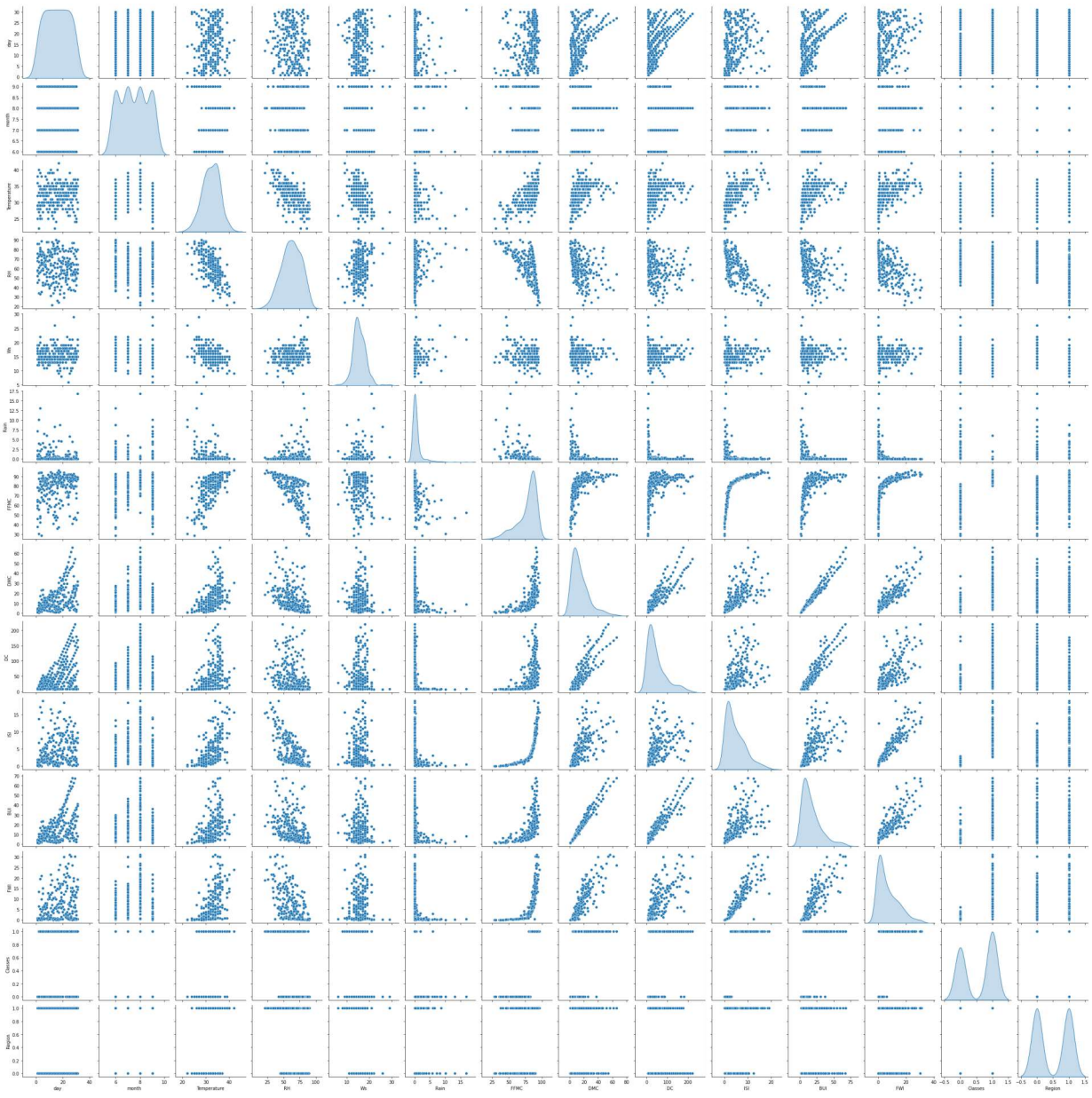



```
Observation---->Rain , DMC,DC,FWI,ISI,BUI are rightly skewd
                2. There is no variance in the attribute
```

Multivariate Analysis

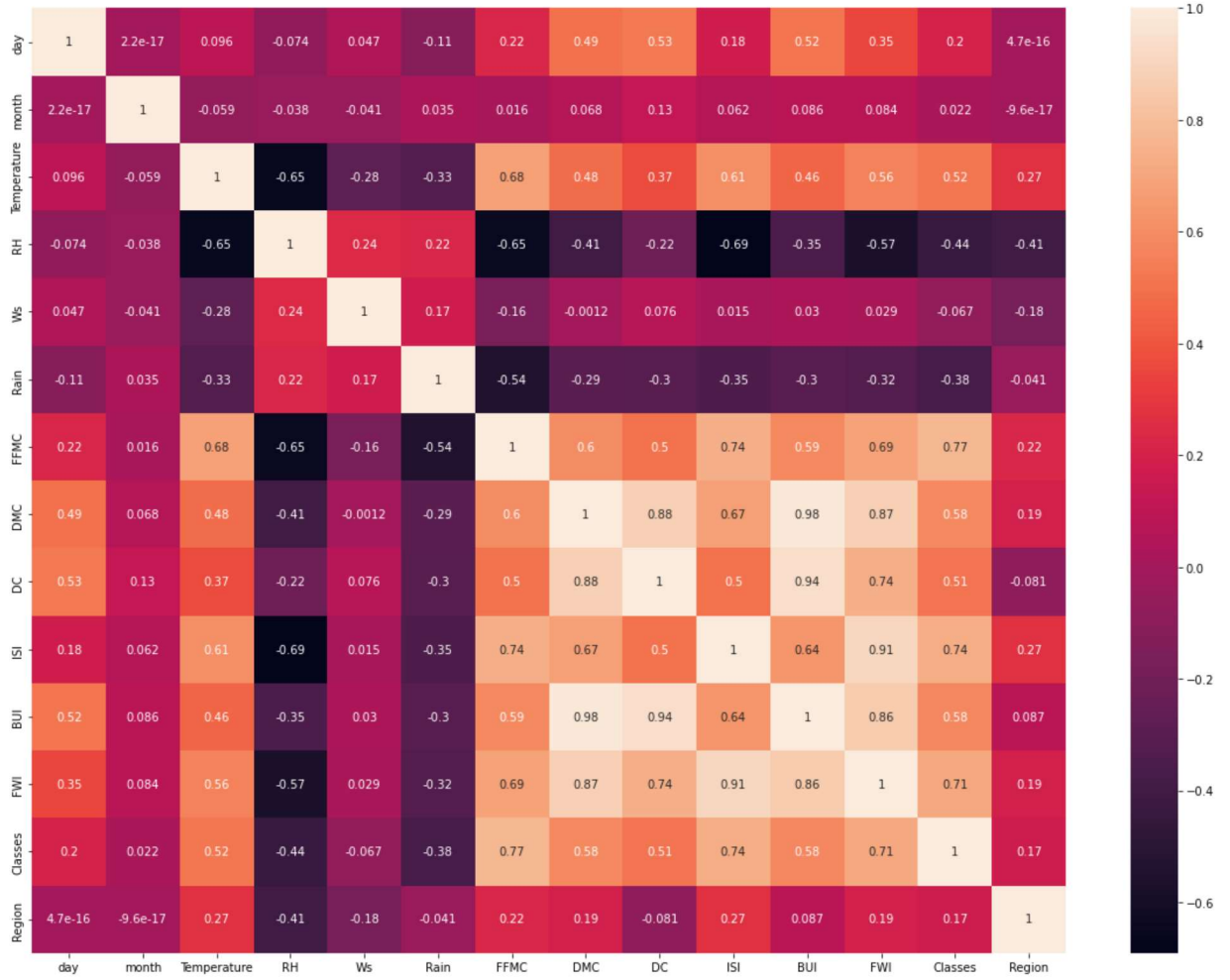
In [35]:
```
sns.pairplot(df,diag_kind='kde')
```

Out[35]: <seaborn.axisgrid.PairGrid at 0x1dee6e70eb0>



In [36]:
```
plt.figure(figsize=(20,15))
sns.heatmap(data.corr(),annot=True)
```

Out[36]: <AxesSubplot:>



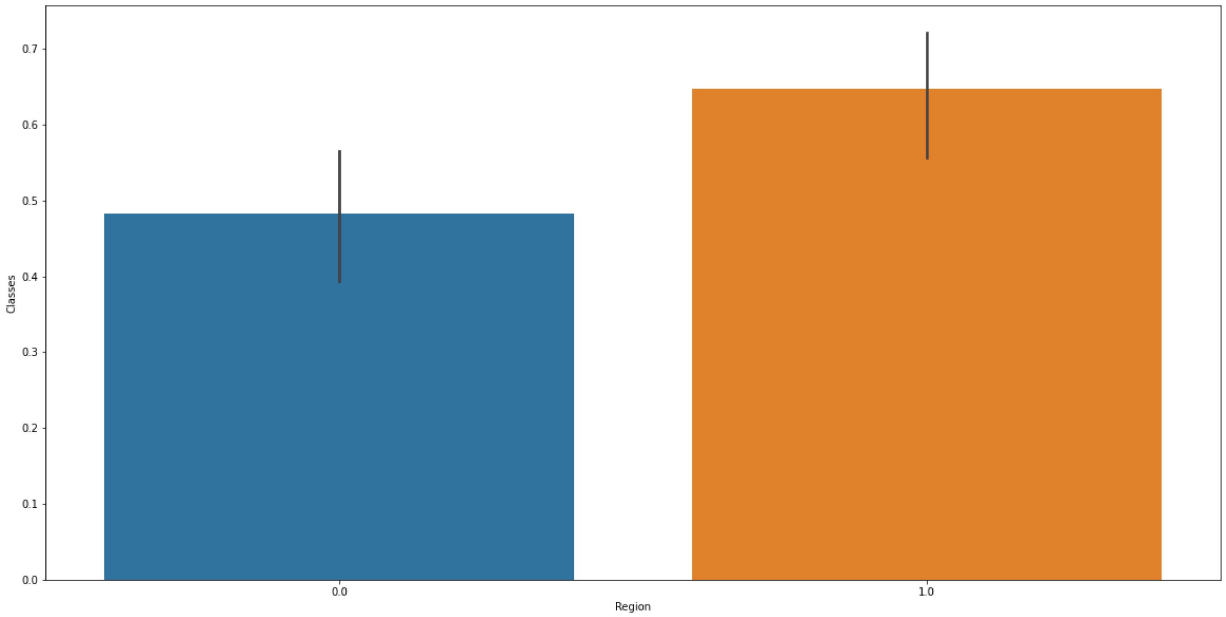
Visualisation of Target Feature

In [37]:
```
data.Classes.value_counts()
```

Out[37]:
```
1    138
0    106
Name: Classes, dtype: int64
```

In [38]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

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

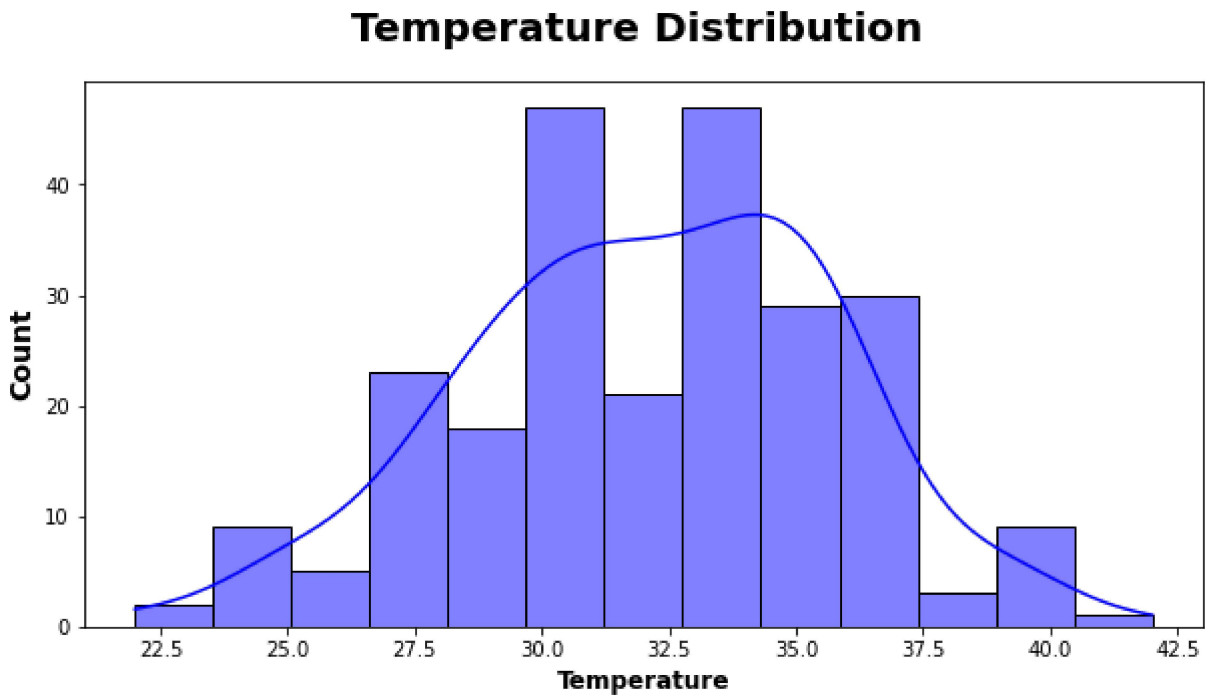
Out[38]:
```
<AxesSubplot:xlabel='Region', ylabel='Classes'>
```



Visulisation of Temperature Feature

In [ ]:
```

```

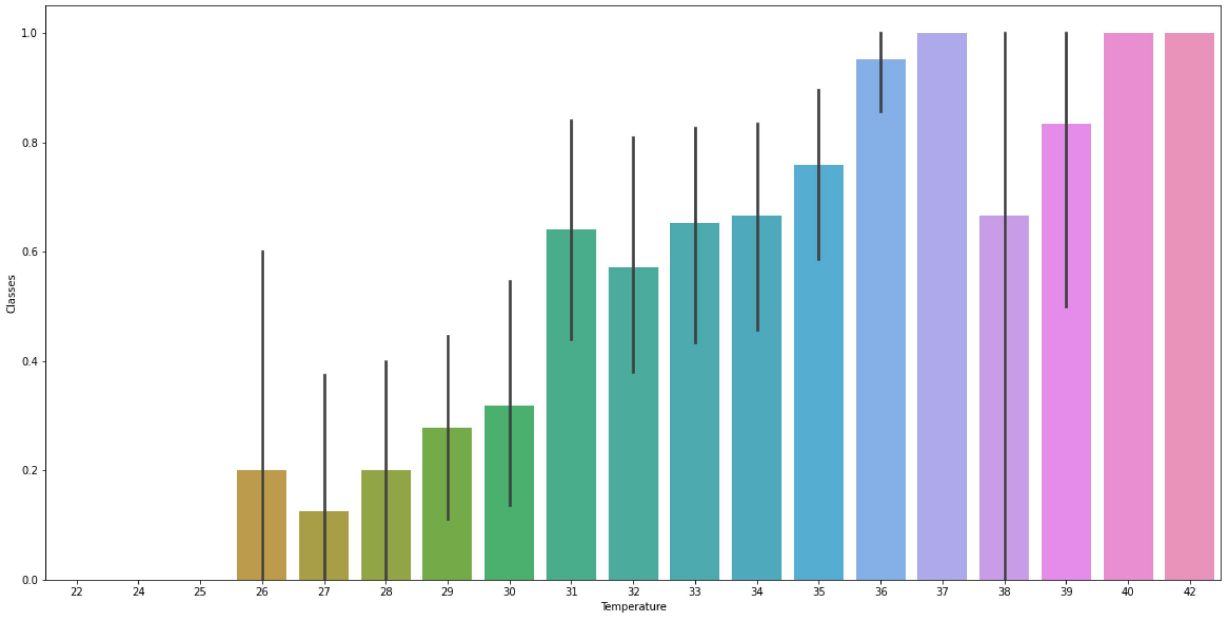
In [39]:
```
plt.subplots(figsize=(10,5))
sns.histplot(x=df.Temperature,ec='Black',color='blue',kde=True)
plt.title('Temperature Distribution ', weight='bold',fontsize=20,pad=20)
plt.ylabel('Count',weight='bold',fontsize=14)
plt.xlabel('Temperature',weight='bold',fontsize=12)
plt.show()
```



Heighest Temperature Attained

In [40]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

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

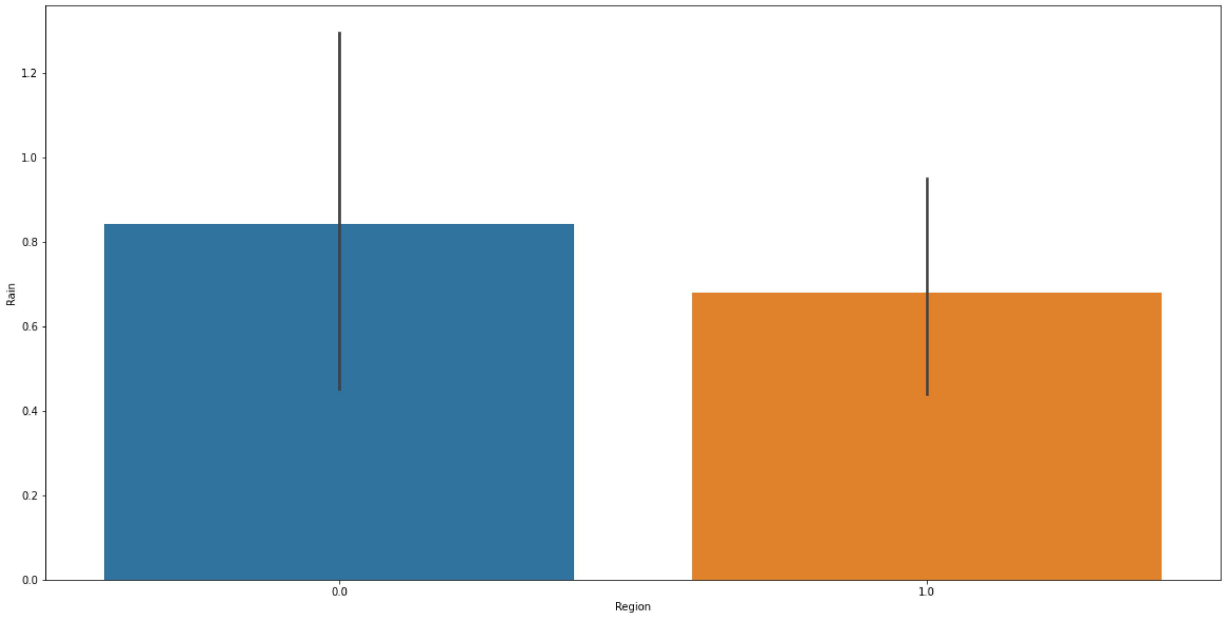
Out[40]: <AxesSubplot:xlabel='Temperature', ylabel='Classes'>



Which region is mostly effected by Rain

In [41]:
```
matplotlib.rcParams['figure.figsize']=(20,10)

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

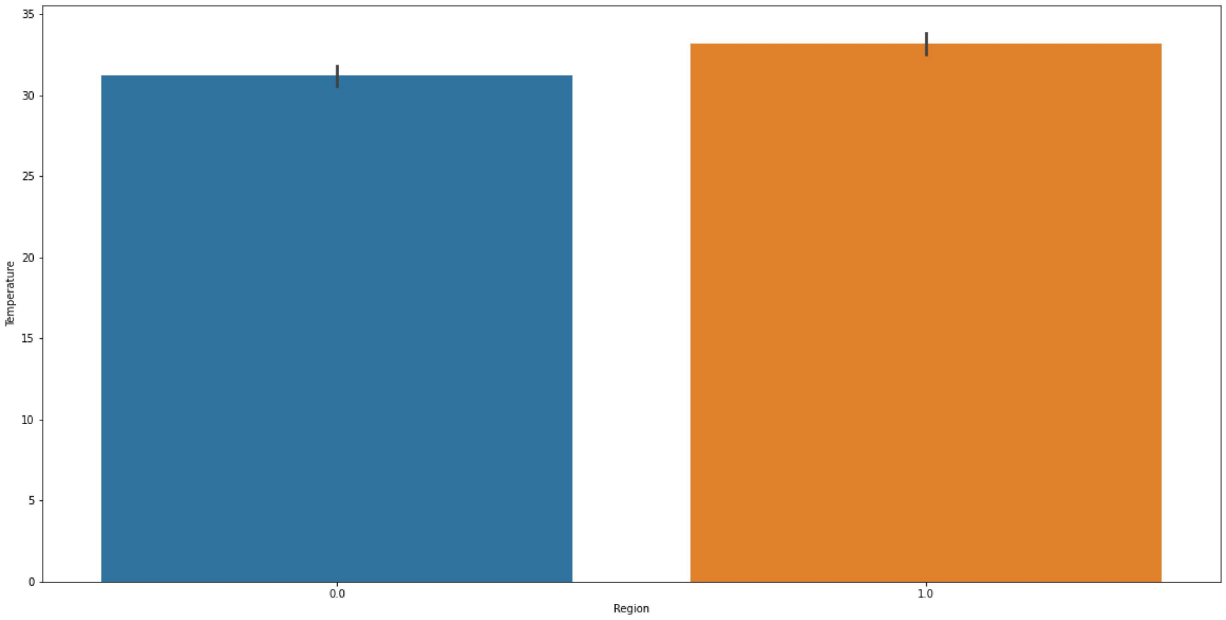
Out[41]: <AxesSubplot:xlabel='Region', ylabel='Rain'>



Which region is highly effected by Temperature

In [42]:
```
matplotlib.rcParams['figure.figsize']=(20,10)

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

Out[42]:
```
<AxesSubplot:xlabel='Region', ylabel='Temperature'>
```



REG PLOT

In [43]:
```
[feature for feature in df.columns]
```

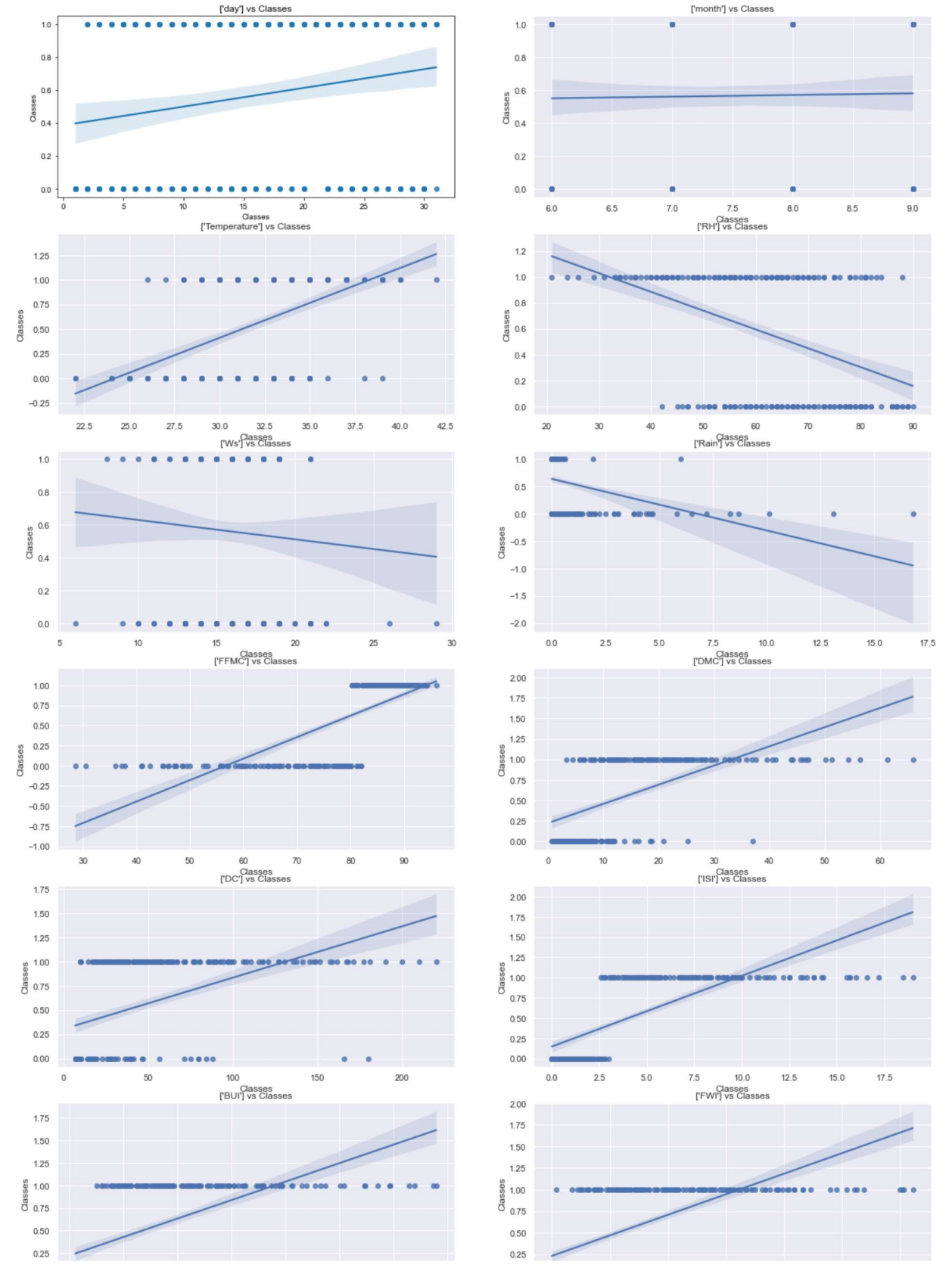
Out[43]:
```
['day',
 'month',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Classes',
 'Region']
```

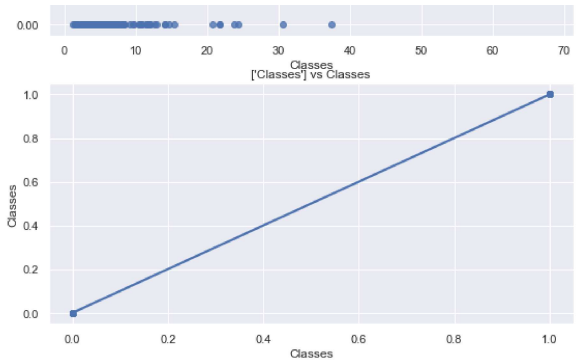
In [44]:
```
num_columns=[feature for feature in df.columns if df[feature].astype!='object']
```

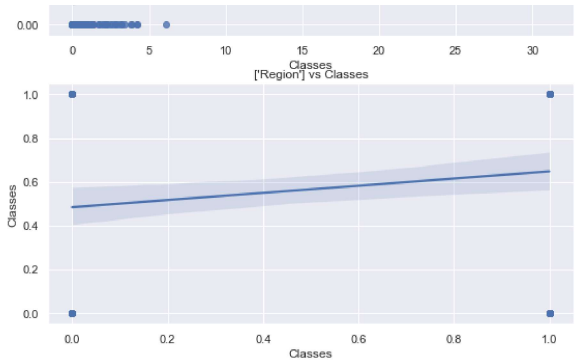
In [45]:
```
num_columns
```

Out[45]:
```
['day',
 'month',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Classes',
 'Region']
```

In [46]:
```
plt.figure(figsize=(20,40))
for i in enumerate(num_columns):
    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]]))
```

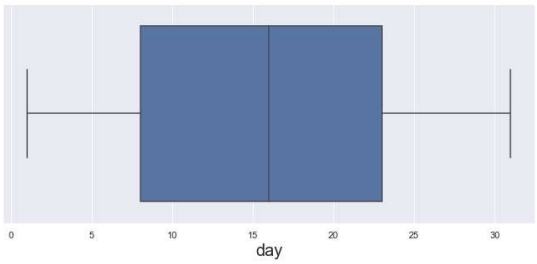


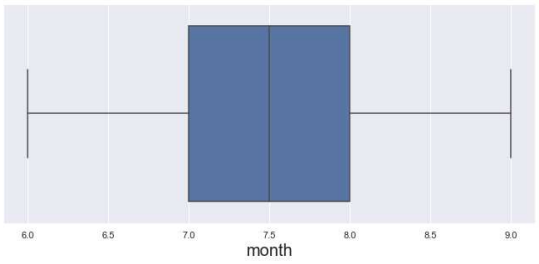


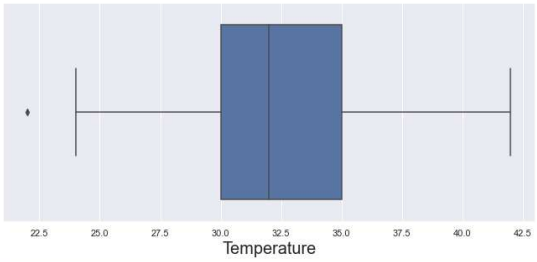


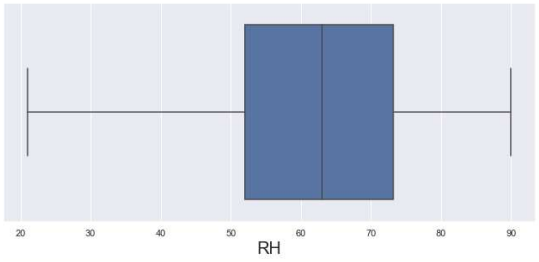
Boxplot to find the Outliers

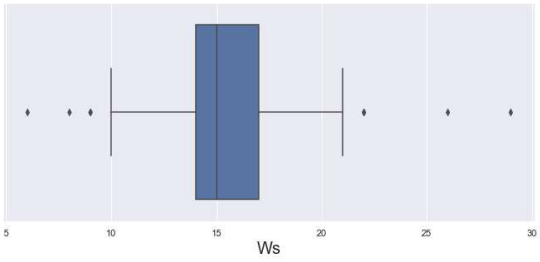
In [47]:
```
plt.figure(figsize=(25,45),facecolor='white')
plotnumber=1

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

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

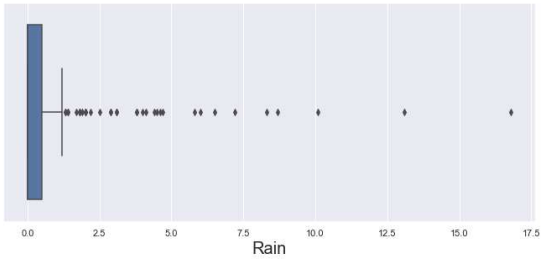


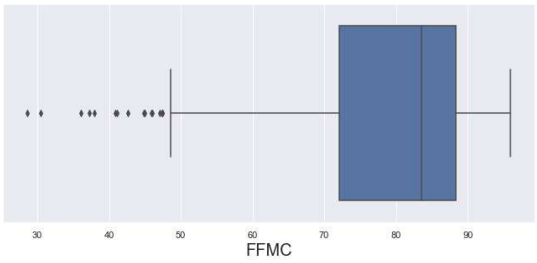


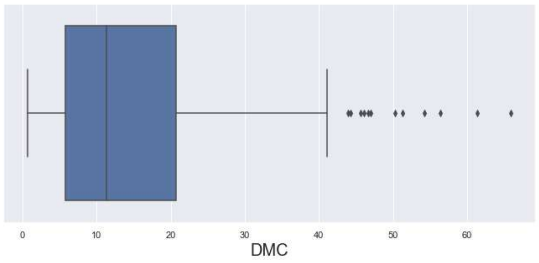


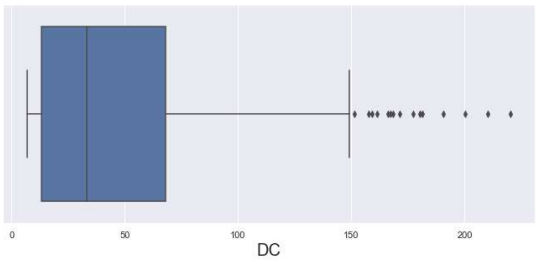


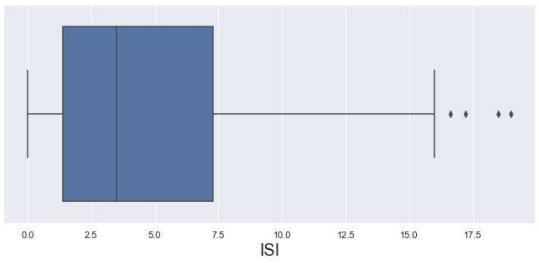


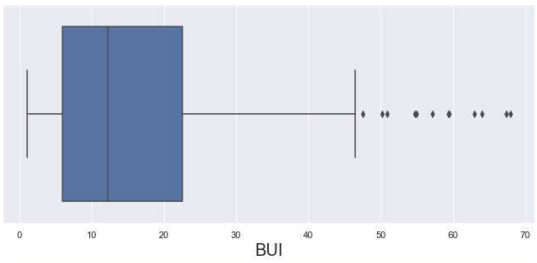


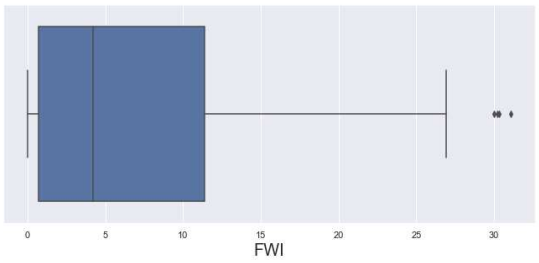


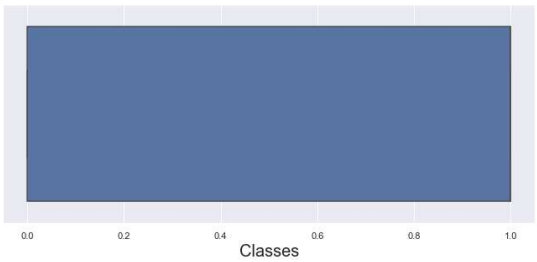


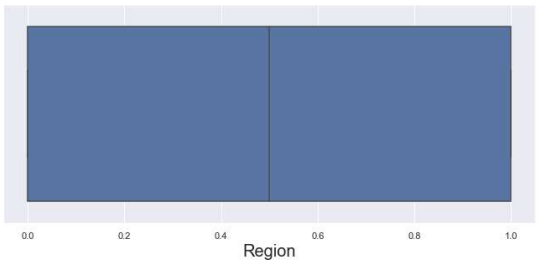













```
Observation :
    WS,DC,ISI,BUI,FWI,RAIN,TEMPERATURE,
```

Dropping the Outliers

In [48]:
```
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 [49]:
```
columns=data.columns
```

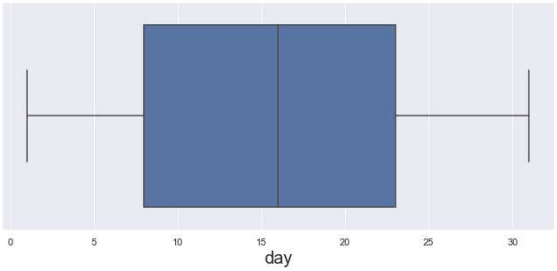
In [50]:
```
for i in columns:
    outliers_imputation_mild(data,i)
```

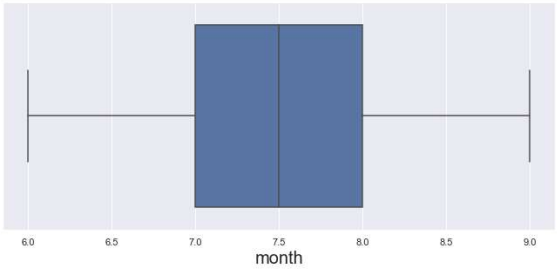
```
IQR: 15.0
Lower Fenceday: -14.5
Upper Fenceday: 45.5

--------------------------------------------------
IQR: 1.0
Lower Fencemonth: 5.5
Upper Fencemonth: 9.5

--------------------------------------------------
IQR: 5.0
Lower FenceTemperature: 22.5
Upper FenceTemperature: 42.5

--------------------------------------------------
IQR: 21.25
Lower FenceRH: 20.125
Upper FenceRH: 105.125

--------------------------------------------------
IQR: 3.0
Lower FenceWs: 9.5
Upper FenceWs: 21.5

--------------------------------------------------
IQR: 0.5
Lower FenceRain: -0.75
Upper FenceRain: 1.25

--------------------------------------------------
IQR: 16.224999999999994
Lower FenceFFMC: 47.73750000000001
Upper FenceFFMC: 112.63749999999999

--------------------------------------------------
IQR: 14.95
Lower FenceDMC: -16.624999999999996
Upper FenceDMC: 43.175

--------------------------------------------------
IQR: 54.87500000000001
Lower FenceDC: -69.03750000000002
Upper FenceDC: 150.46250000000003

--------------------------------------------------
IQR: 5.9
Lower FenceISI: -7.450000000000001
Upper FenceISI: 16.150000000000002

--------------------------------------------------
IQR: 16.525
Lower FenceBUI: -18.787499999999998
Upper FenceBUI: 47.3125

--------------------------------------------------
IQR: 10.675
Lower FenceFWI: -15.312500000000004
Upper FenceFWI: 27.387500000000003

--------------------------------------------------
IQR: 1.0
Lower FenceClasses: -1.5
Upper FenceClasses: 2.5

--------------------------------------------------
IQR: 1.0
```

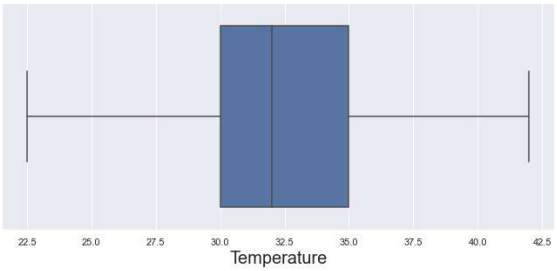
```
Lower FenceRegion: -1.5
Upper FenceRegion: 2.5
```

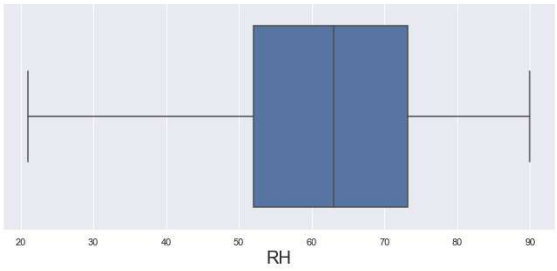
Rechecking the Outliers after Dropping

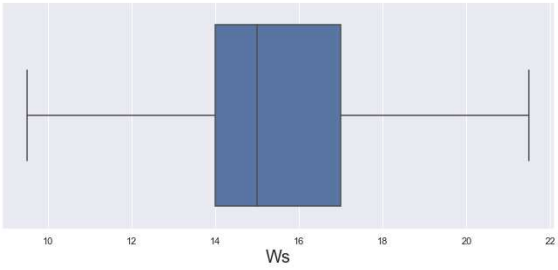
In [51]:
```
plt.figure(figsize=(25,45),facecolor='white')
plotnumber=1

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

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

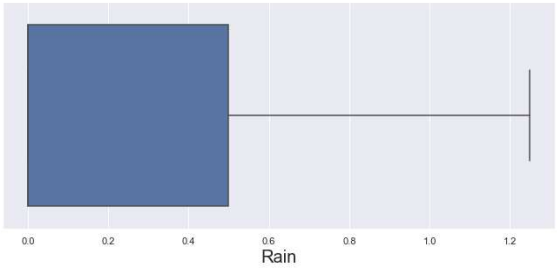

0 5 10 15 20 25 30
day

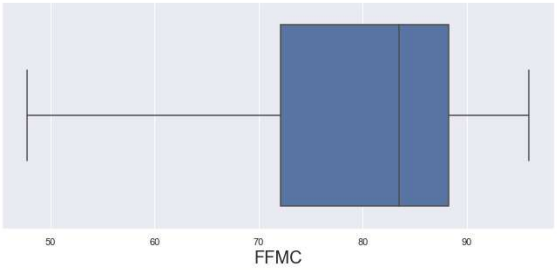

6.0 6.5 7.0 7.5 8.0 8.5 9.0
month

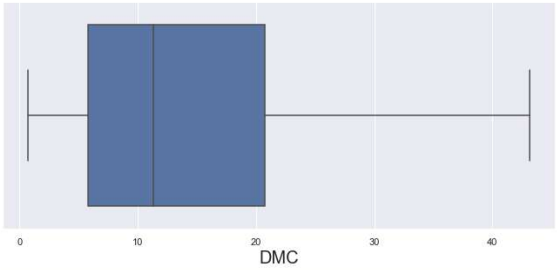

22.5 25.0 27.5 30.0 32.5 35.0 37.5 40.0 42.5
Temperature

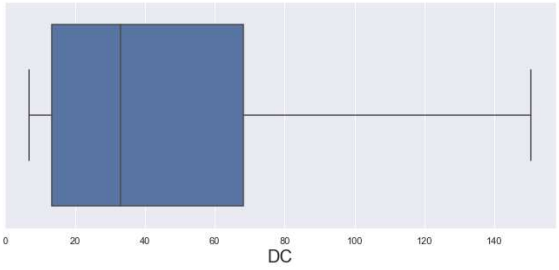

20 30 40 50 60 70 80 90
RH

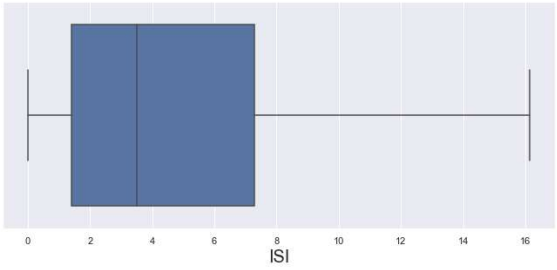

10 12 14 16 18 20 22
Ws

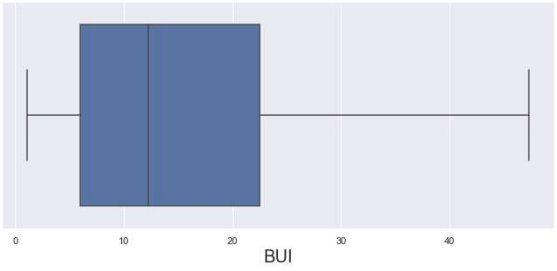

0.0 0.2 0.4 0.6 0.8 1.0 1.2
Rain

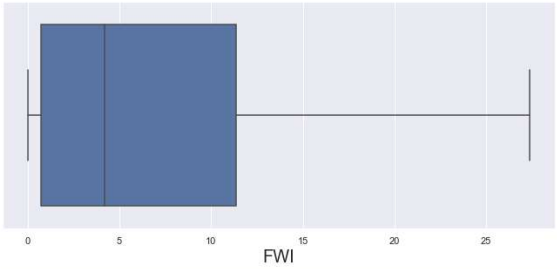

50 60 70 80 90
FFMC

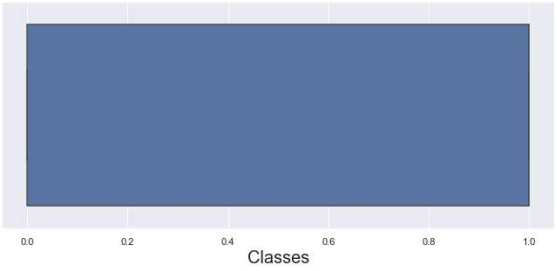

0 10 20 30 40
DMC

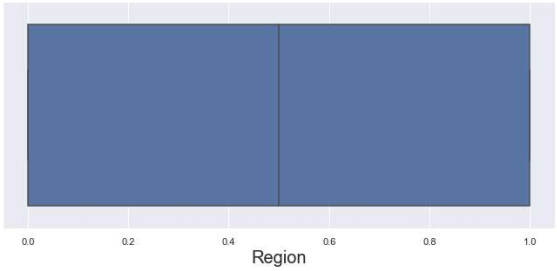

0 20 40 60 80 100 120 140
DC


0 2 4 6 8 10 12 14 16
ISI


0 10 20 30 40
BUI


0 5 10 15 20 25
FWI


0.0 0.2 0.4 0.6 0.8 1.0
Classes


0.0 0.2 0.4 0.6 0.8 1.0
Region

```
Observation:
    Outlier is not present in any of the feature
```

Creating Independent and Dependent Feature

In [52]:
```
x=data.drop(columns=['Classes'])
y=data['Classes']
```

Independent Feature

In [53]:
```
x
```

Out[53]:

	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

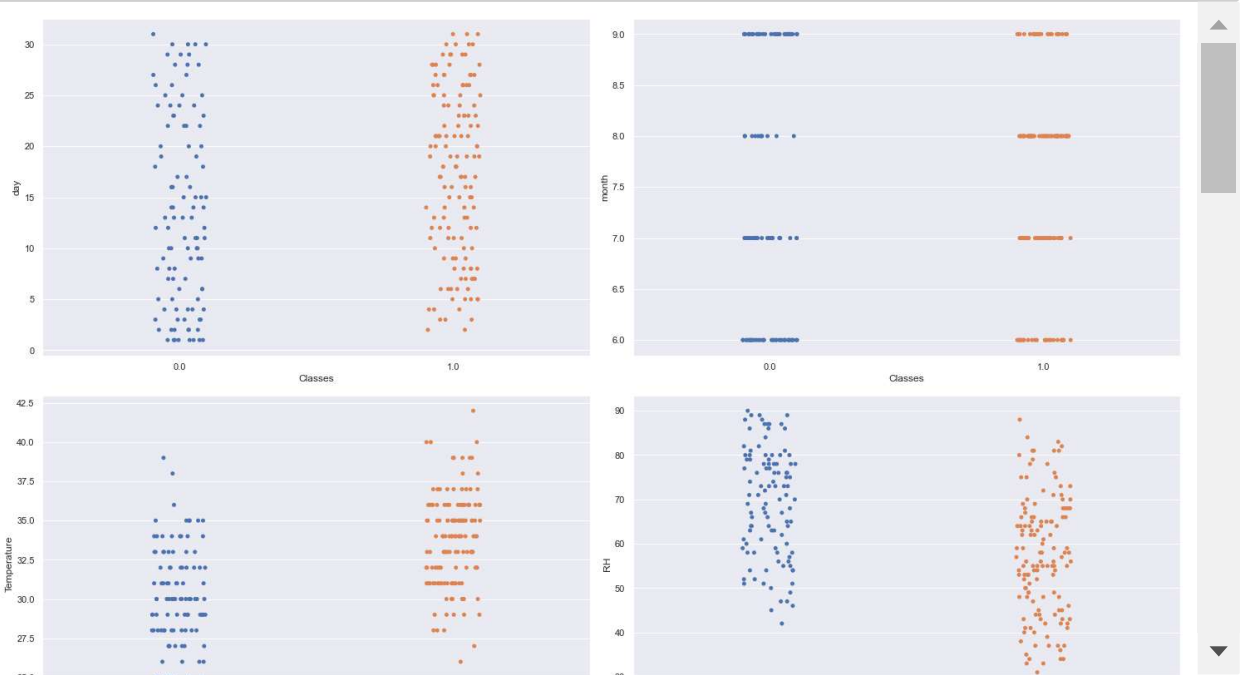
Dependent Feature

In [54]:
```
y
```

Out[54]:
```
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
```

Visualization the Independent and Dependent Feature

In [55]:
```
plt.figure(figsize=(20,50),facecolor='white')
plotnumber=1

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



Importing Sklearn libraries for Machine Learning

In [56]:
```
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_auc_sc
```

Train Test Split

In [57]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=1
```

Logistic Regression Model Training

In [58]:
```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
```

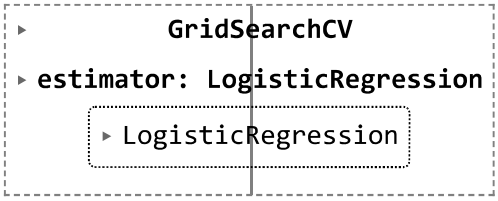
In [59]:
```
from sklearn.model_selection import GridSearchCV
parameter={'penalty':['l1','l2','elasticnet'],'max_iter':[100,200,300]}
```

In [60]:
```
classifier_regressor=GridSearchCV(classifier,param_grid=parameter,scoring='accura
```

Standardizing or Feature Selection

In [61]:
```
classifier_regressor.fit(x_train,y_train)
```

Out[61]:



In [62]:
```
print(classifier_regressor.best_params_)
```

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

In [63]:
```
print(classifier_regressor.best_score_)
```

```
0.9818181818181818
```

```
classifier_regressor.predict(
```

Prediction

In [64]:
```
y_pred=classifier_regressor.predict(x_test)
```

In [65]:
```
y_pred
```

Out[65]:
```
array([0., 1., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 0., 1., 1., 1.,
       0., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1., 0.,
       1., 0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0.,
       1., 1., 1., 1., 0., 1., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0.,
       1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0.])
```

Accuracy Score

In [66]:
```
from sklearn.metrics import accuracy_score,classification_report
score=accuracy_score(y_pred,y_test)
print(score)
```

```
0.9506172839506173
```

Classification Report

In [67]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.88      0.94        34
         1.0       0.92      1.00      0.96        47

    accuracy                           0.95        81
   macro avg       0.96      0.94      0.95        81
weighted avg       0.95      0.95      0.95        81
```

Performance Metrics

In [68]:
```
conf_mat=confusion_matrix(y_pred,y_test)
```

In [69]:
```
conf_mat
```

Out[69]:
```
array([[30,  4],
       [ 0, 47]], dtype=int64)
```

In [70]:
```
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 formual for Accuracy

In [71]:
```
Accuracy=(true_positive + true_negative)/((true_positive +false_positive + false_
Accuracy
```

Out[71]: 0.9506172839506173

Precision

In [72]:
```
Precision=true_positive/(true_positive+false_positive)
Precision
```

Out[72]: 0.8823529411764706

Recall

In [73]:
```
Recall=true_positive/(true_positive+false_negative)
Recall
```

Out[73]: 1.0

F1 Score

In [74]:
```
F1_Score=2*(Recall * Precision) / (Recall + Precision)
F1_Score
```

Out[74]: 0.9375

Area Under Curve

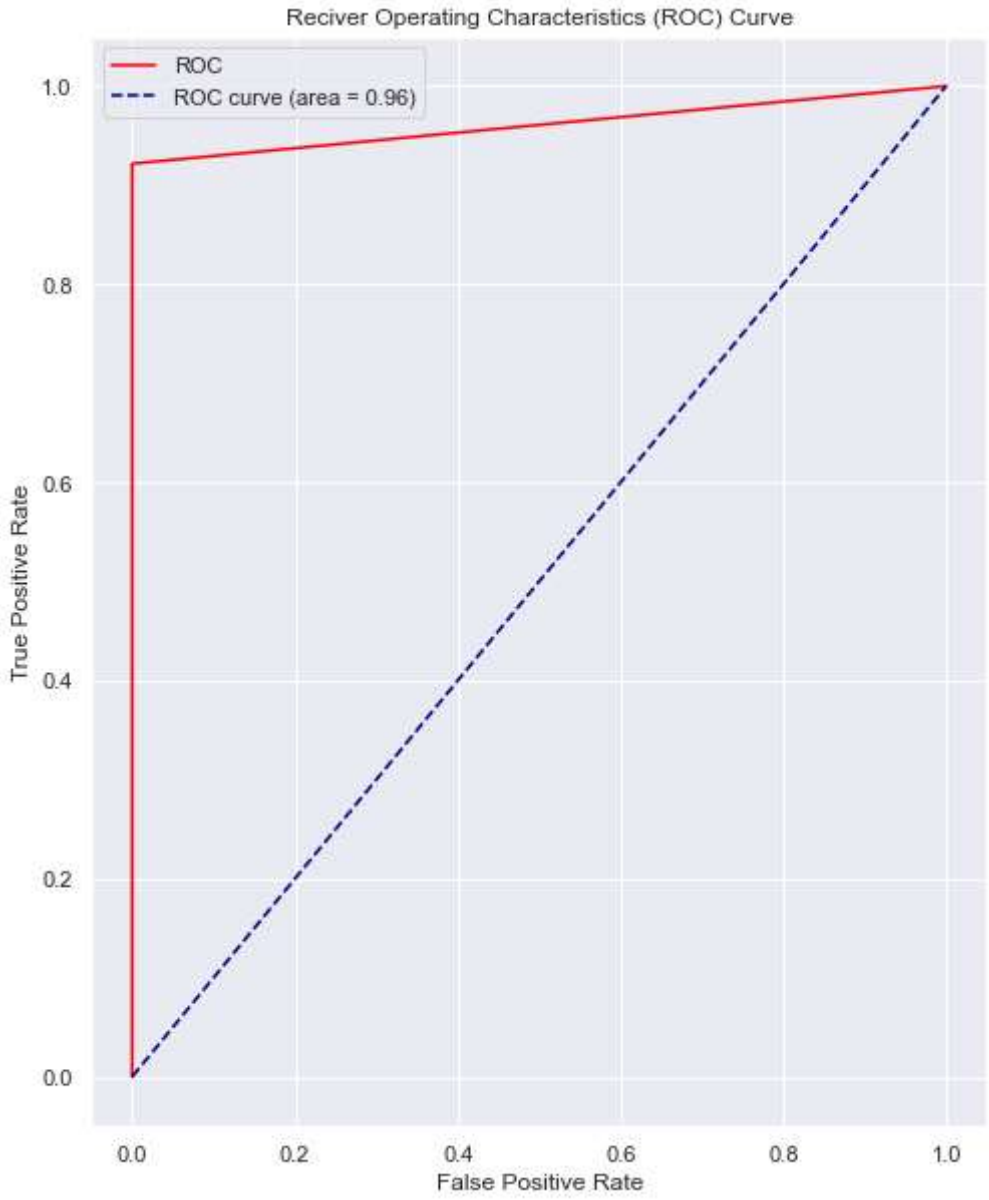
In [75]:
```
auc=roc_auc_score(y_test,y_pred)
auc
```

Out[75]: 0.9607843137254901

ROC

In [76]:
```
fpr , tpr, thresholds = roc_curve(y_test, y_pred)
```

In [77]:
```
plt.plot(fpr,tpr,color='red',label='ROC')
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--',label='ROC curve (area
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Reciver Operating Characteristics (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 balance dataset

In [78]:
```
df.head()
```

Out[78]:

	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 [79]:
```
df.shape
```

Out[79]:
```
(244, 14)
```

In [80]:
```
### Creating imbalance
### 1. splitting data in 90:10 percent ratio using train test split
X1 = pd.DataFrame(df, columns = ['day', 'month','Temperature','RH','Ws','Rain','F
y1=pd.DataFrame(df,columns = ['Classes'])
```

In [81]:
```
X_train_imb, X_test_imb, y_train_imb, y_test_imb = train_test_split(X1, y1, test_
```

In [82]:
```
## Both will have same shape
X_train_imb.shape, y_train_imb.shape
```

Out[82]:
```
((219, 13), (219, 1))
```

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

In [83]:
```
y_train_imb=y_train_imb.replace(0,1)
y_train_imb.head()
```

Out[83]:

	Classes
156	1
183	1
11	1
75	1
130	1

In [84]:
```
y_test_imb=y_test_imb.replace(1,0)
y_test_imb.head()
```

Out[84]:

	Classes
48	0
216	0
101	0
38	0
86	0

In [85]:
```
X_train_imb.head()
```

Out[85]:

	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 [86]:
```
### Combining X_train_imb and y_train_imb
train_imb=X_train_imb.join(pd.DataFrame(y_train_imb))
train_imb.head()
```

Out[86]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Class
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 [87]:
```
### Combining X_test_imb and y_test_imb
test_imb=X_test_imb.join(pd.DataFrame(y_test_imb))
test_imb.head()
```

Out[87]:

	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 [88]:
```
### Checking the shape of imbalanced Data
train_imb.shape, test_imb.shape
```

Out[88]:
```
((219, 14), (25, 14))
```

In [89]:
```
### 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[89]:

	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 [90]:
```
df_imb.shape
```

Out[90]:
```
(244, 14)
```

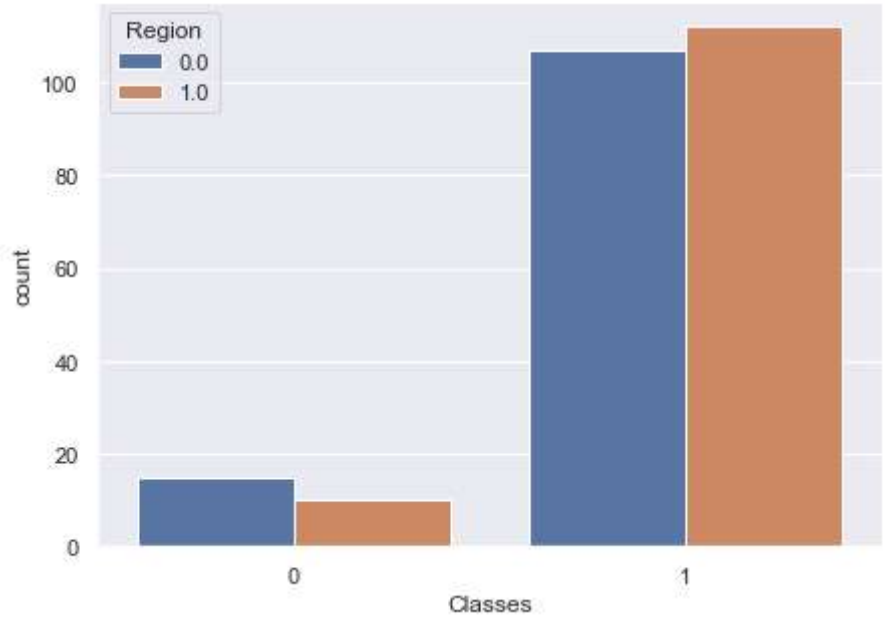
Checking the imbalancing

In [91]:
```
df_imb.Classes.value_counts()
```

Out[91]:
```
1    219
0     25
Name: Classes, dtype: int64
```

In [92]:
```
## 0 is 'Bejaia' and 1 is 'Sidi Bel-abbes region'
plt.figure(figsize=(7,5))
sns.countplot(data=df_imb,x='Classes',hue='Region')
```

Out[92]:
```
<AxesSubplot:xlabel='Classes', ylabel='count'>
```



Logistic Regression on imbalanced Dataset

In [93]:
```
df_imb.head()
```

Out[93]:

	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 [94]:
```
X1 = df_imb.drop(columns = ['Classes'])
y1 = df_imb['Classes']
```

In [95]:
```
X1
```

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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
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 [96]:
```
y1
```

Out[96]:
```
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
```

Handling Imbalance dataset by Doing Upsampling

In [97]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\mukul\anaconda3\lib\site-pa
ckages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\mukul\anaconda3\lib
\site-packages (from imblearn) (0.9.1)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\mukul\anaconda3
\lib\site-packages (from imbalanced-learn->imblearn) (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in c:\users\mukul\anaconda3\lib\si
te-packages (from imbalanced-learn->imblearn) (1.1.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\mukul\anaconda3\lib\si
te-packages (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\mukul\anaconda3
\lib\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\mukul\anaconda3\lib\sit
e-packages (from imbalanced-learn->imblearn) (1.9.3)
```

In [98]:
```
from imblearn.combine import SMOTETomek
```

In [99]:
```
smk=SMOTETomek()
smk
```

Out[99]:
```
SMOTETomek
SMOTETomek()
```

SMOTETomek()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [100]:
```
X_bal,y_bal=smk.fit_resample(X1,y1)
```

In [101]:
```
X_bal.head()
```

Out[101]:

	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 [102]:
```
y_bal.head()
```

Out[102]:
```
0    1
1    1
2    1
3    1
4    1
Name: Classes, dtype: int64
```

In [103]:
```
X_bal.shape,y_bal.shape
```

Out[103]:
```
((426, 13), (426,))
```

In [104]:
```
## Creating Balanced data from imbalanced data
data_bal=X_bal.join(pd.DataFrame(y_bal))
data_bal.head()
```

Out[104]:

	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 [105]:
```
data_bal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 426 entries, 0 to 425
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          426 non-null    int64  
 1   month        426 non-null    int64  
 2   Temperature  426 non-null    int64  
 3   RH           426 non-null    int64  
 4   Ws           426 non-null    int64  
 5   Rain         426 non-null    float64
 6   FFMC         426 non-null    float64
 7   DMC          426 non-null    float64
 8   DC           426 non-null    float64
 9   ISI          426 non-null    float64
 10  BUI          426 non-null    float64
 11  FWI          426 non-null    float64
 12  Region       426 non-null    float64
 13  Classes      426 non-null    int64  
dtypes: float64(8), int64(6)
memory usage: 46.7 KB
```

Statistical analysis on Balanced Dataset

In [106]:
```
data_bal.describe().T
```

Out[106]:

	count	mean	std	min	25%	50%	75%	max
day	426.0	14.938967	8.504727	1.0	8.000000	15.000000	22.000000	31.0
month	426.0	7.481221	0.977812	6.0	7.000000	7.000000	8.000000	9.0
Temperature	426.0	32.326291	3.250935	22.0	30.000000	32.000000	35.000000	42.0
RH	426.0	62.399061	13.597418	21.0	54.000000	64.000000	72.000000	90.0
Ws	426.0	15.373239	2.667299	6.0	13.000000	15.000000	17.000000	29.0
Rain	426.0	0.556720	1.560769	0.0	0.000000	0.000000	0.420344	16.8
FFMC	426.0	78.995473	12.687051	28.6	72.460945	84.250000	88.100000	96.0
DMC	426.0	16.242474	13.384957	0.7	5.932642	12.628229	21.912693	65.9
DC	426.0	56.103831	51.199544	6.9	15.700000	41.350000	82.215201	220.4
ISI	426.0	4.821947	3.725532	0.0	1.517114	4.343348	7.288649	19.0
BUI	426.0	18.741880	15.584273	1.1	6.050000	15.900000	24.815718	68.0
FWI	426.0	7.503535	7.019345	0.0	0.800000	6.000000	12.900000	31.1
Region	426.0	0.437384	0.459812	0.0	0.000000	0.228974	1.000000	1.0
Classes	426.0	0.500000	0.500588	0.0	0.000000	0.500000	1.000000	1.0

In [107]:
```
data_bal.corr()
```

Out[107]:

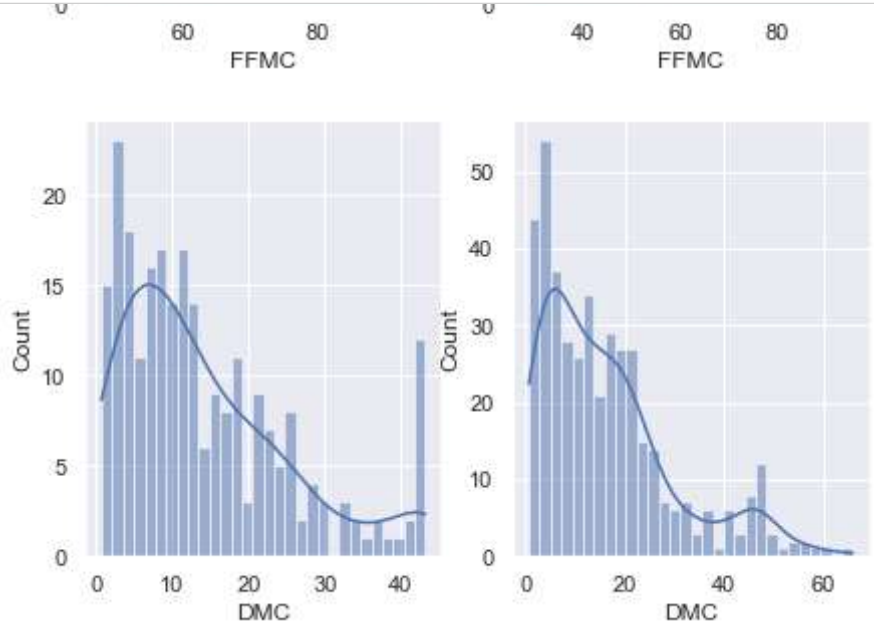
	day	month	Temperature	RH	Ws	Rain	FFMC	DM
day	1.000000	-0.075117	0.129482	-0.055844	0.178271	-0.097400	0.284709	0.5981
month	-0.075117	1.000000	-0.105766	0.075777	0.008558	0.018812	0.001738	0.0024
Temperature	0.129482	-0.105766	1.000000	-0.654792	-0.305237	-0.289204	0.658540	0.3831
RH	-0.055844	0.075777	-0.654792	1.000000	0.251624	0.191233	-0.591632	-0.2325
Ws	0.178271	0.008558	-0.305237	0.251624	1.000000	0.093905	-0.050617	0.1924
Rain	-0.097400	0.018812	-0.289204	0.191233	0.093905	1.000000	-0.540536	-0.2741
FFMC	0.284709	0.001738	0.658540	-0.591632	-0.050617	-0.540536	1.000000	0.5992
DMC	0.598196	0.002492	0.383121	-0.232524	0.192448	-0.274168	0.599200	1.0000
DC	0.627808	0.074991	0.303130	-0.100689	0.278204	-0.289466	0.535099	0.9031
ISI	0.273037	0.036548	0.627989	-0.641647	0.056734	-0.350561	0.777251	0.6214
BUI	0.622004	0.022738	0.363545	-0.187026	0.232711	-0.285394	0.593508	0.9855
FWI	0.465109	0.042699	0.547774	-0.475174	0.135501	-0.330213	0.731156	0.8542
Region	0.000283	0.015349	0.244785	-0.427873	-0.133443	-0.026479	0.214521	0.1439
Classes	0.123799	0.007211	-0.068678	-0.044593	0.022028	0.183159	-0.116160	-0.1558

In [110]:
```
num_bal_col=[feature for feature in data_bal.columns if data_bal[feature].dtype !
num_bal_col
```

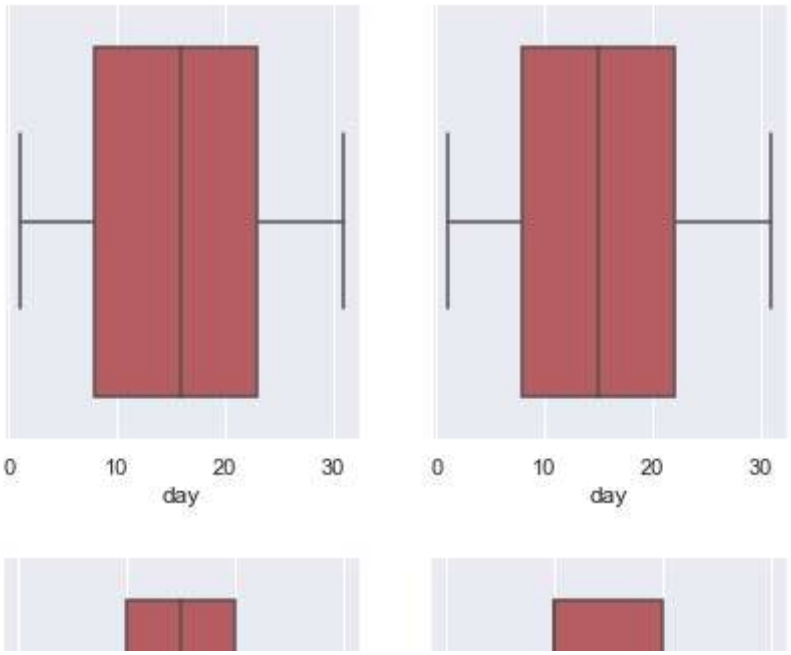
Out[110]:
```
['day',
 'month',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Region',
 'Classes']
```

Compairing the feature for Original and Balanced Dataset

In [112]:
```
for i in num_bal_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)
```



In [113]:
```
for i in num_bal_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')
```



Train Test Split

In [114]:
```
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=0.30,ran
```

In [116]:
```
X_train1
```

Out[116]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	
3	15	8	36	55	13	0.3	82.400000	15.600000	92.500000	3.700000	22.00
92	28	6	37	37	13	0.0	92.500000	27.200000	52.400000	11.700000	27.10
12	1	7	28	58	18	2.2	63.700000	3.200000	8.500000	1.200000	3.30
149	13	8	35	34	16	0.2	88.300000	16.900000	45.100000	7.500000	17.50
7	3	6	29	80	14	2.0	48.700000	2.200000	7.600000	0.300000	2.60
...	...	...	...	...	...	...	...	...	...	...	
321	15	7	33	63	14	0.0	87.526528	18.953057	80.493395	6.413264	23.40
69	17	9	34	44	12	0.0	92.500000	25.200000	63.300000	11.200000	26.20
121	12	9	31	72	14	0.0	84.200000	8.300000	25.200000	3.800000	9.10
238	22	8	34	66	19	0.0	87.356751	32.259492	151.447133	7.293507	41.93
169	20	7	33	65	15	0.1	81.400000	12.300000	62.100000	2.800000	16.50

298 rows × 13 columns

In [117]:
```
X_test1
```

Out[117]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	
103	29	8	35	53	17	0.500000	80.200000	20.700000	149.200000	2.700000	30
374	1	6	29	57	17	0.000000	65.869814	3.442233	7.707343	1.323756	3
325	27	6	32	44	13	0.243448	82.218652	19.380243	86.831756	4.386696	24
322	16	7	34	63	14	0.437597	82.495167	21.381747	68.884559	4.912014	23
133	30	7	31	79	15	0.000000	85.400000	28.500000	136.000000	4.700000	37
...	...	...	...	...	...	...	...	...	...	...	
55	17	7	29	70	14	0.000000	82.800000	9.400000	34.100000	3.200000	11
25	20	6	30	80	16	0.400000	59.800000	3.400000	27.100000	0.900000	5
168	31	7	35	64	17	0.000000	87.200000	31.900000	145.700000	6.800000	41
194	11	6	31	65	14	0.000000	84.500000	12.500000	54.300000	4.000000	15
386	6	7	35	42	14	0.284614	85.059007	15.551287	44.371729	4.571820	16

128 rows × 13 columns

In [119]:
```
y_train1
```

Out[119]:
```
3      1
92     1
12     1
149    1
7      1
      ..
321    0
69     1
121    1
238    0
169    1
Name: Classes, Length: 298, dtype: int64
```

In [120]:
```
y_test1
```

Out[120]:
```
103    1
374    0
325    0
322    0
133    1
      ..
55     1
25     1
168    1
194    1
386    0
Name: Classes, Length: 128, dtype: int64
```

Logistic Regression Model

In [121]:
```
from sklearn.linear_model import LogisticRegression
classifier_bal=LogisticRegression()
classifier_bal
```

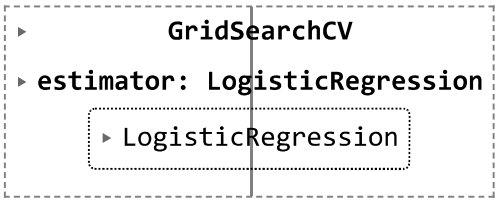
Out[121]:
```
LogisticRegression
LogisticRegression()
```

In [122]:
```
from sklearn.model_selection import GridSearchCV
parameter_bal={'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,40,50
```

In [123]:
```
classifier_regressor_bal=GridSearchCV(classifier,param_grid=parameter,scoring='ac
```

Standarizing or Feature Scaling

In [124]:
```
classifier_regressor_bal.fit(X_train1,y_train1)
```

Out[124]:


In [125]:
```
print(classifier_regressor_bal.best_params_)
```

```
{'max_iter': 100, 'penalty': 'l2'}
```

In [126]:
```
print(classifier_regressor_bal.best_score_)
```

```
0.6980790960451977
```

Prediction

In [127]:
```
y_bal_pred = classifier_regressor_bal.predict(X_test1)
```

In [128]:
```
y_bal_pred
```

Out[128]:
```
array([1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0], dtype=int64)
```

Acurracy

In [129]:
```
from sklearn.metrics import accuracy_score,classification_report
bal_score=accuracy_score(y_bal_pred,y_test1)
print(bal_score)
```

```
0.71875
```

Classification Report

In [130]:
```
print(classification_report(y_bal_pred,y_test1))
```

```
              precision    recall  f1-score   support

           0       0.73      0.77      0.75        69
           1       0.71      0.66      0.68        59

    accuracy                           0.72       128
   macro avg       0.72      0.71      0.72       128
weighted avg       0.72      0.72      0.72       128
```

Performance Metrics

Confusion Metrics

In [131]:
```
conf_mat_bal=confusion_matrix(y_bal_pred,y_test1)
```

In [132]:
```
conf_mat_bal
```

Out[132]:
```
array([[53, 16],
       [20, 39]], dtype=int64)
```

Precision

In [133]:
```
bal_Precision = true_positive/(true_positive+false_positive)
bal_Precision
```

Out[133]:
```
0.8823529411764706
```

Recall

In [134]:
```
bal_recall = true_positive/(true_positive+false_negative)
bal_recall
```

Out[134]:
```
1.0
```

F1 score

In [135]:
```
F1_Score_bal = 2*(bal_recall * bal_Precision) / (bal_recall + bal_Precision)
F1_Score_bal
```

Out[135]: 0.9375

Conclusion

Performance of Logistic Model on Original Dataset

In [136]:
```
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.88      0.94        34
         1.0       0.92      1.00      0.96        47

    accuracy                           0.95        81
   macro avg       0.96      0.94      0.95        81
weighted avg       0.95      0.95      0.95        81
```

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

In [137]:
```
print(classification_report(y_bal_pred,y_test1))
```

```
              precision    recall  f1-score   support

           0       0.73      0.77      0.75        69
           1       0.71      0.66      0.68        59

    accuracy                           0.72       128
   macro avg       0.72      0.71      0.72       128
weighted avg       0.72      0.72      0.72       128
```

```
Observation
It seems that model is good when we predict from original dataset

It seems that model is very bad when we
```

In [ ]:
```

```