Importing Required Libraries

In [ ]:
```
# 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_repor

# 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 [ ]:
```
df = pd.read_csv(r"D:\iNeuron\Machine Learning\Implementation\Logistic Regression\L
df.head()
```

Out[ ]:

	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 attributess

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.

Checking the null value

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
day            0
month          1
year           1
Temperature    1
 RH            1
 Ws            1
Rain           1
FFMC           1
DMC            1
DC             1
ISI            1
BUI            1
FWI            1
Classes        2
dtype: int64
```

In [ ]:
```
df.drop([122,123,167],axis = 0,inplace = True)
```

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

Out[ ]:
```
(243, 14)
```

Data Cleaning

In [ ]:
```
# columns name having extra spaces
columns_with_spaces = [fea for fea in df.columns if " " in fea]
columns_with_spaces
```

Out[ ]:
```
[' RH', ' Ws', 'Rain ', 'Classes  ']
```

In [ ]:
```
# removing the space
df.columns = df.columns.str.strip()
# strip() method Remove spaces at the beginning and at the end of the string
df.columns
```

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

In [ ]:
```
# function to remove extra space in data
def remove_space(x):
    return x.replace(" ","")
```

In [ ]:
```
df['Classes'] = df['Classes'].apply(remove_space)
df.head(2)
```

Out[ ]:

	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	notfire
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	notfire

Creating Region feature

In [ ]:
```
df.loc[:122,'Region'] = 0 # Bejaia region
df.loc[122:,'Region'] = 1 #  Sidi Bel-abbes region

df.iloc[120:125]
```

Out[ ]:

	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	notfire
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	notfire
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	notfire
125	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	notfire
126	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	notfire

Datatypes and Describe

In [ ]:
```
# here it is visible that all datatypes are in object
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 243 entries, 0 to 245
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          243 non-null    object 
 1   month        243 non-null    object 
 2   year         243 non-null    object 
 3   Temperature  243 non-null    object 
 4   RH           243 non-null    object 
 5   Ws           243 non-null    object 
 6   Rain         243 non-null    object 
 7   FFMC         243 non-null    object 
 8   DMC          243 non-null    object 
 9   DC           243 non-null    object 
 10  ISI          243 non-null    object 
 11  BUI          243 non-null    object 
 12  FWI          243 non-null    object 
 13  Classes      243 non-null    object 
 14  Region       243 non-null    float64
dtypes: float64(1), object(14)
memory usage: 38.5+ KB
```

Convert notfire and fire to 0 and 1 respectively for Classes feature

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

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

In [ ]:
```
df['Classes'] = df['Classes'].replace({"notfire":0,'fire':1})
df['Classes'].unique()
```

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

Changing datatype to Numerical from object

In [ ]:
```
###### changing datatypes of features to numerical for numerical features as all ar

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

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

Out[ ]:
```
day              int64
month            int64
year             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
```

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

Now again check the null value and duplicates

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

Out[ ]:
```
day            0
month          0
year           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 [ ]:
```
df.duplicated().value_counts()
```

Out[ ]:
```
False    243
dtype: int64
```

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

Out[ ]:
```
(243, 15)
```

- After data cleaning there are 243 rows and 15 columns.
- There is no null value in dataset.
- There is no duplicate observation in dataset

Creating a copy of dataframe from original Dataframe

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

Out[ ]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	R
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

Statistical Analysis

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

Out[ ]:

	day	month	year	Temperature	RH	Ws	Rain	
day	78.190729	-0.003639	0.0	3.119138	-9.969476	1.188603	-1.993174	28.5
month	-0.003639	1.242764	0.0	-0.229653	-0.681903	-0.124987	0.077762	0.2
year	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.0
Temperature	3.119138	-0.229653	0.0	13.162670	-35.043482	-2.901949	-2.372850	35.2
RH	-9.969476	-0.681903	0.0	-35.043482	219.874333	10.173809	6.604836	-137.2
Ws	1.188603	-0.124987	0.0	-2.901949	10.173809	7.903887	0.965886	-6.7
Rain	-1.993174	0.077762	0.0	-2.372850	6.604836	0.965886	4.012837	-15.6
FFMC	28.544043	0.272433	0.0	35.222858	-137.215388	-6.718952	-15.634746	205.9
DMC	53.863133	0.938676	0.0	21.837668	-75.071928	-0.025120	-7.169025	107.3
DC	222.524339	6.722457	0.0	65.071727	-160.400449	10.604530	-28.456455	347.0
ISI	6.632060	0.303838	0.0	9.101371	-42.298446	0.099643	-2.891688	44.1
BUI	65.061368	1.349400	0.0	23.734918	-74.653741	1.257586	-8.546509	120.8
FWI	23.079143	0.685464	0.0	15.297068	-64.096917	0.677079	-4.835502	73.7
Classes	0.891321	0.013298	0.0	0.930330	-3.184454	-0.097745	-0.377380	5.4
Region	0.003639	0.001037	0.0	0.489984	-2.991651	-0.255178	-0.040159	1.5

In [ ]:
```
data.describe().T.sort_values(by='std' , ascending = False)\
                    .style.background_gradient(cmap='GnBu')\
                    .bar(subset=["max"], color='#BB0000')\
                    .bar(subset=["mean",], color='green')
```

Out[ ]:

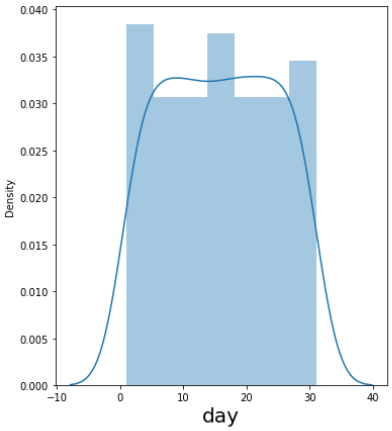
	count	mean	std	min	25%	50%	75%	
DC	243.000000	47.931276	43.678238	6.900000	12.350000	33.100000	69.100000	154.22
RH	243.000000	62.041152	14.828160	21.000000	52.500000	63.000000	73.500000	90.00
FFMC	243.000000	78.210185	13.372705	47.175000	71.850000	83.300000	88.300000	96.00
BUI	243.000000	16.176029	12.759347	1.100000	6.000000	12.400000	22.650000	47.62
DMC	243.000000	14.297942	11.233125	0.700000	5.800000	11.300000	20.800000	43.30
day	243.000000	15.761317	8.842552	1.000000	8.000000	16.000000	23.000000	31.00
FWI	243.000000	6.988889	7.301158	0.000000	0.700000	4.200000	11.450000	27.57
ISI	243.000000	4.712757	4.064045	0.000000	1.400000	3.500000	7.250000	16.02
Temperature	243.000000	32.156379	3.616741	22.500000	30.000000	32.000000	35.000000	42.00
Ws	243.000000	15.465021	2.565226	9.500000	14.000000	15.000000	17.000000	21.50
month	243.000000	7.502058	1.114793	6.000000	7.000000	8.000000	8.000000	9.00
Region	243.000000	0.497942	0.501028	0.000000	0.000000	0.000000	1.000000	1.00
Classes	243.000000	0.563786	0.496938	0.000000	0.000000	1.000000	1.000000	1.00
Rain	243.000000	0.311317	0.465000	0.000000	0.000000	0.000000	0.500000	1.25

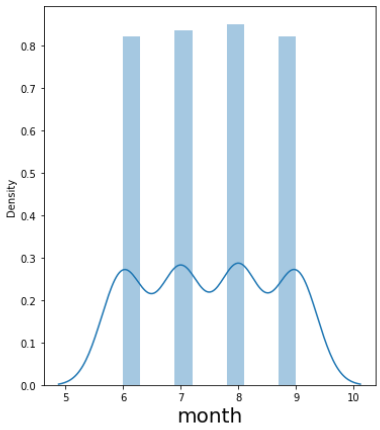
Checking the distribution of the features

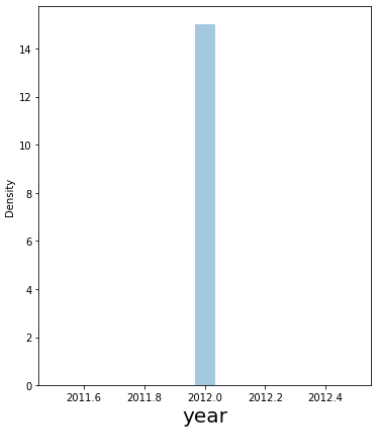
In [ ]:
```
len(df.columns)
```

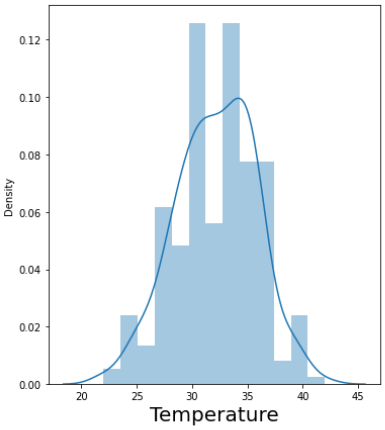
Out[ ]:
```
15
```

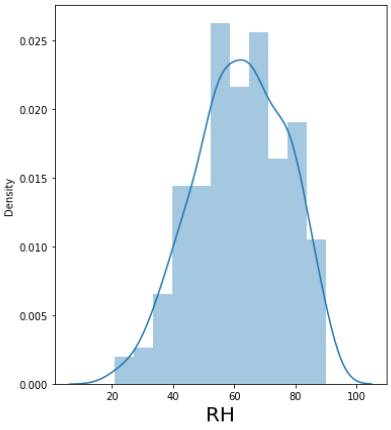
In [ ]:
```
# let's see how data is distributed for every column
plt.figure(figsize=(20,40), facecolor='white')
plotnumber = 1

for column in data:
    if plotnumber<=15 :     # as there are 15 columns in the data
        ax = plt.subplot(5,3,plotnumber)
        sns.distplot(data[column],kde= True)
        plt.xlabel(column,fontsize=20)

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

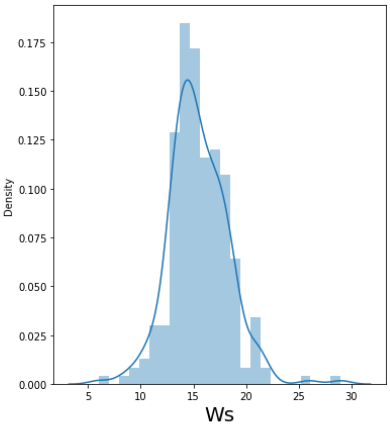

Density
day

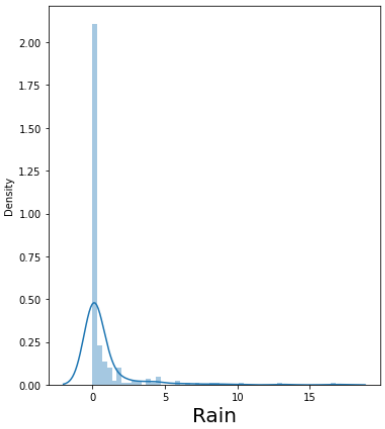

Density
month

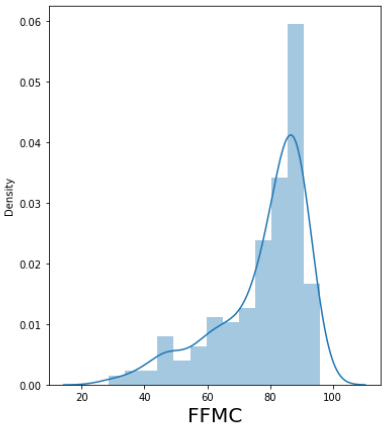

Density
year

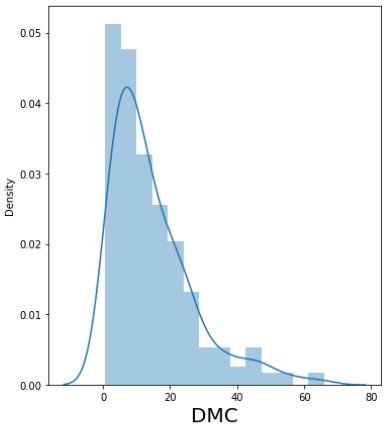

Density
Temperature

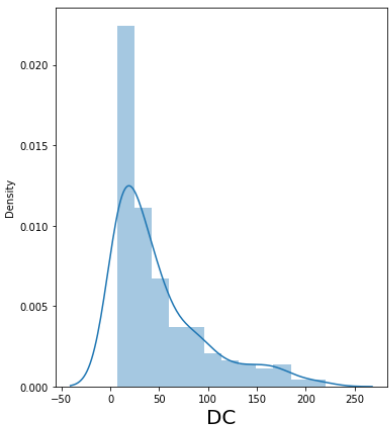

Density
RH

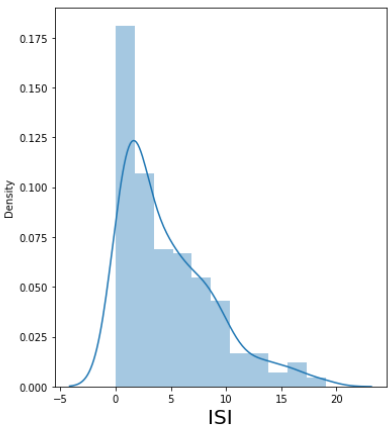

Density
Ws

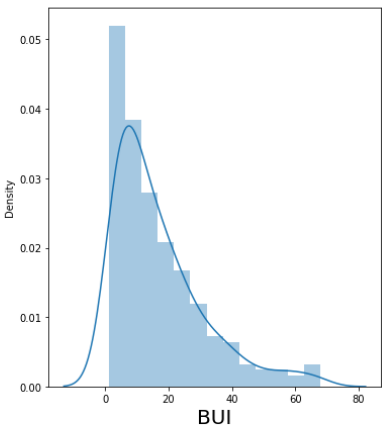

Density
Rain

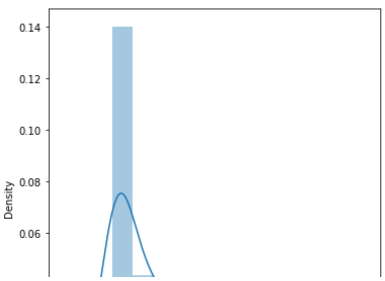

Density
FFMC

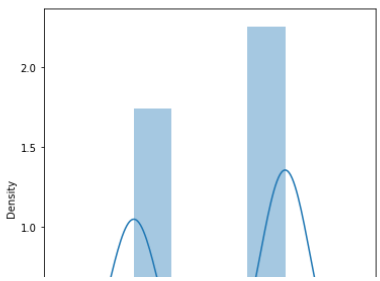

Density
DMC

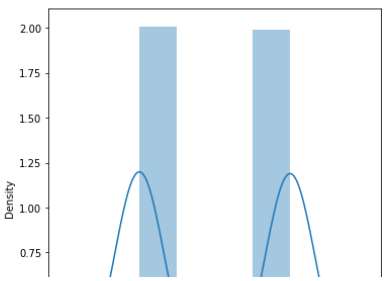

Density
DC

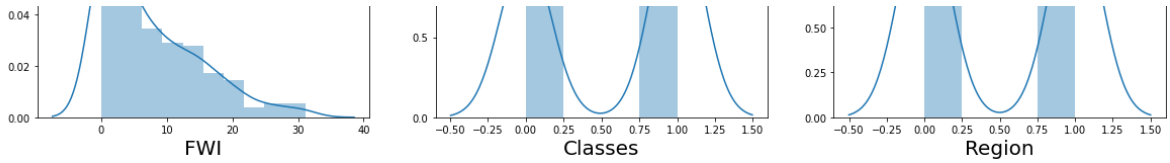

Density
ISI


Density
BUI


Density


Density

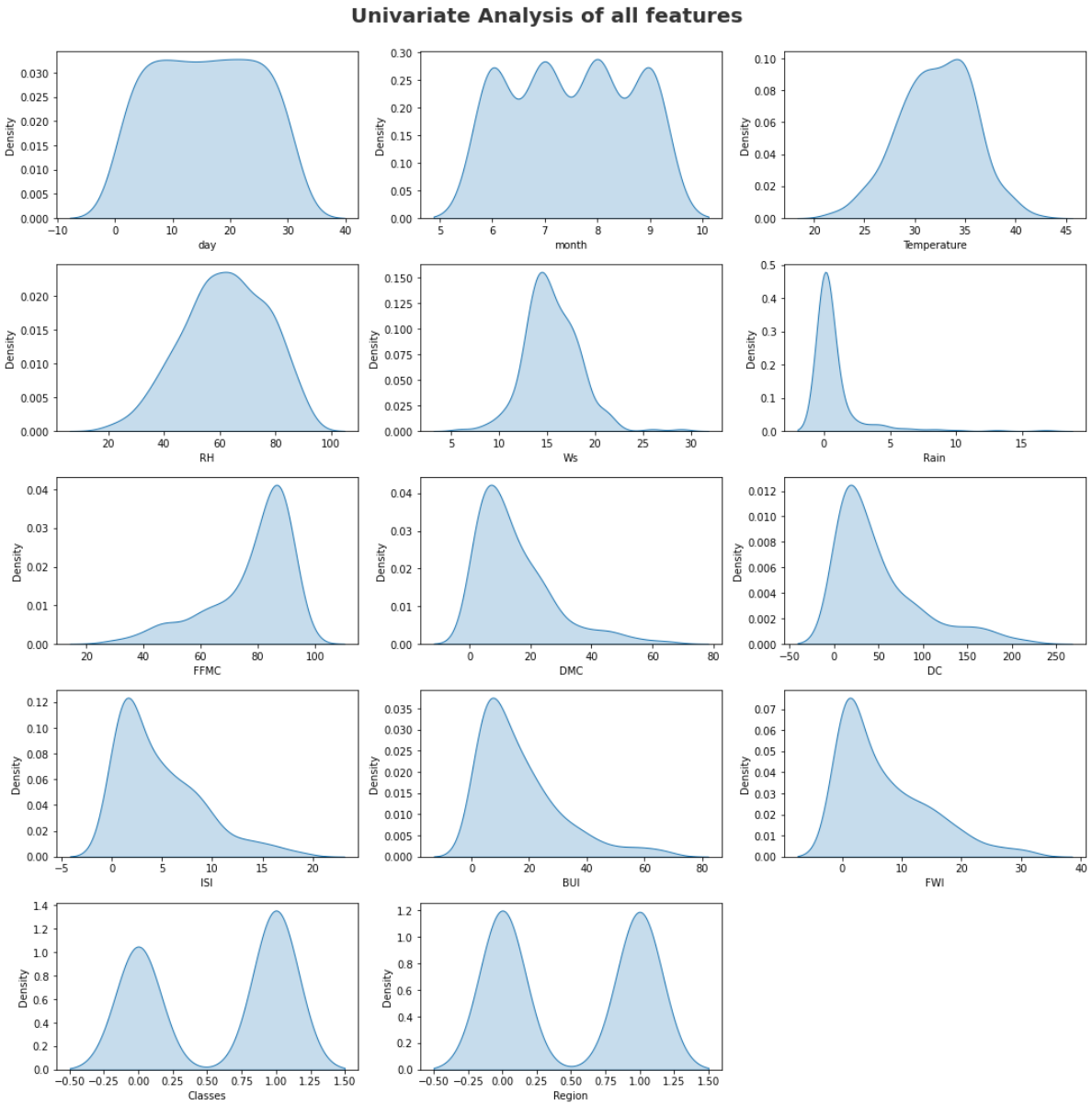

Density



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

Univariate Analysis

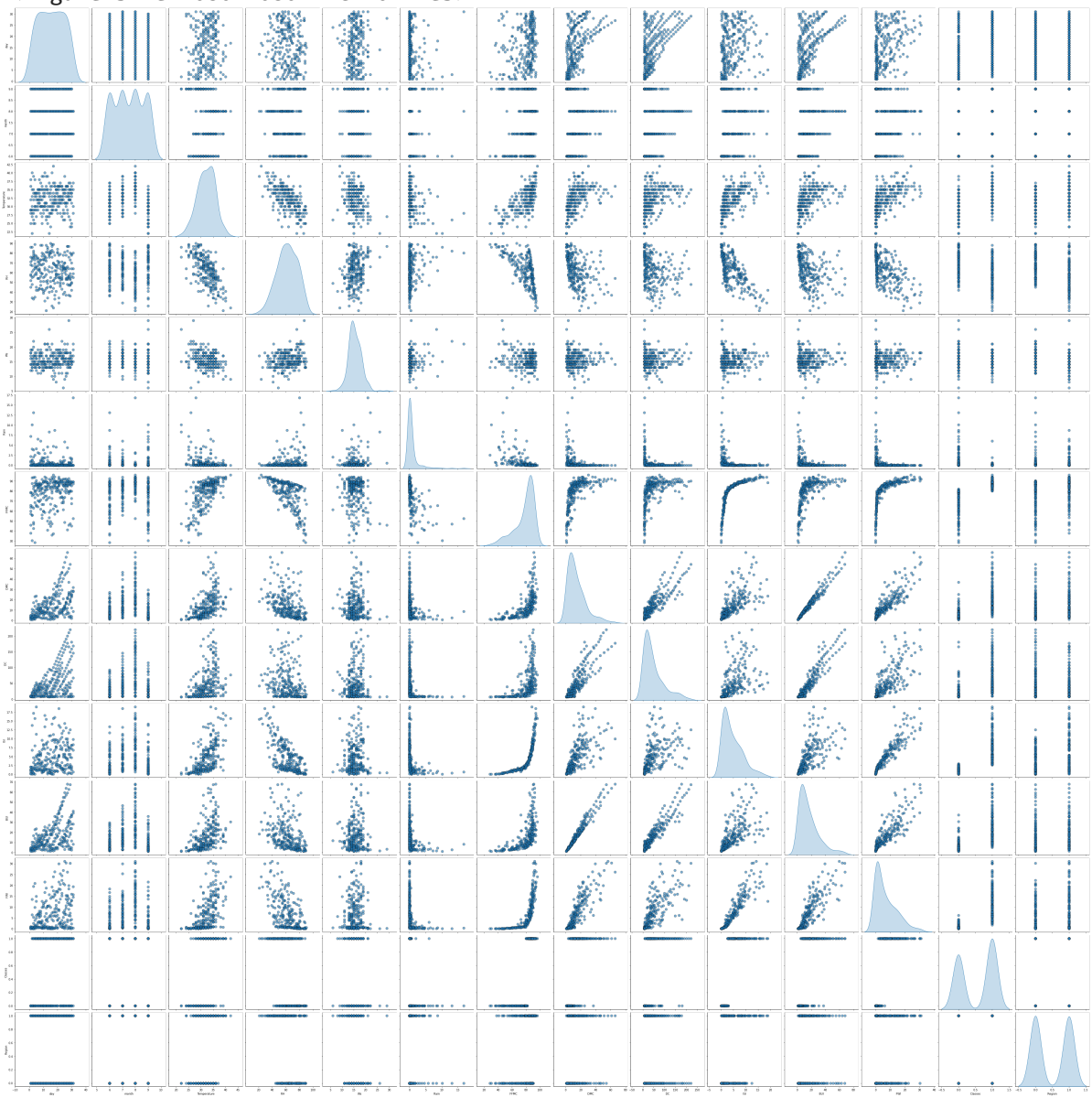
In [ ]:
```
plt.figure(figsize = (15,15))
plt.suptitle('Univariate Analysis of all features', fontsize = 20, fontweight = "b

for i in range(0, len(data.columns)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(data[data.columns[i]], shade =True, palette="ch:s=.25,rot=-.25")
    plt.xlabel(data.columns[i])
    plt.tight_layout()
```



Multivariate Analysis

In [ ]:
```
plt.figure(figsize=(15,15))
plt.suptitle('Multivariate Analysis', fontsize=20, fontweight='bold', alpha=0.8, y
sns.pairplot(data,  diag_kind = 'kde',palette="ch:s=.25,rot=-.25",
             plot_kws = {'alpha': 0.6, 's': 80, 'edgecolor': 'k'},
             size = 4)
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x1a03897e560>

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

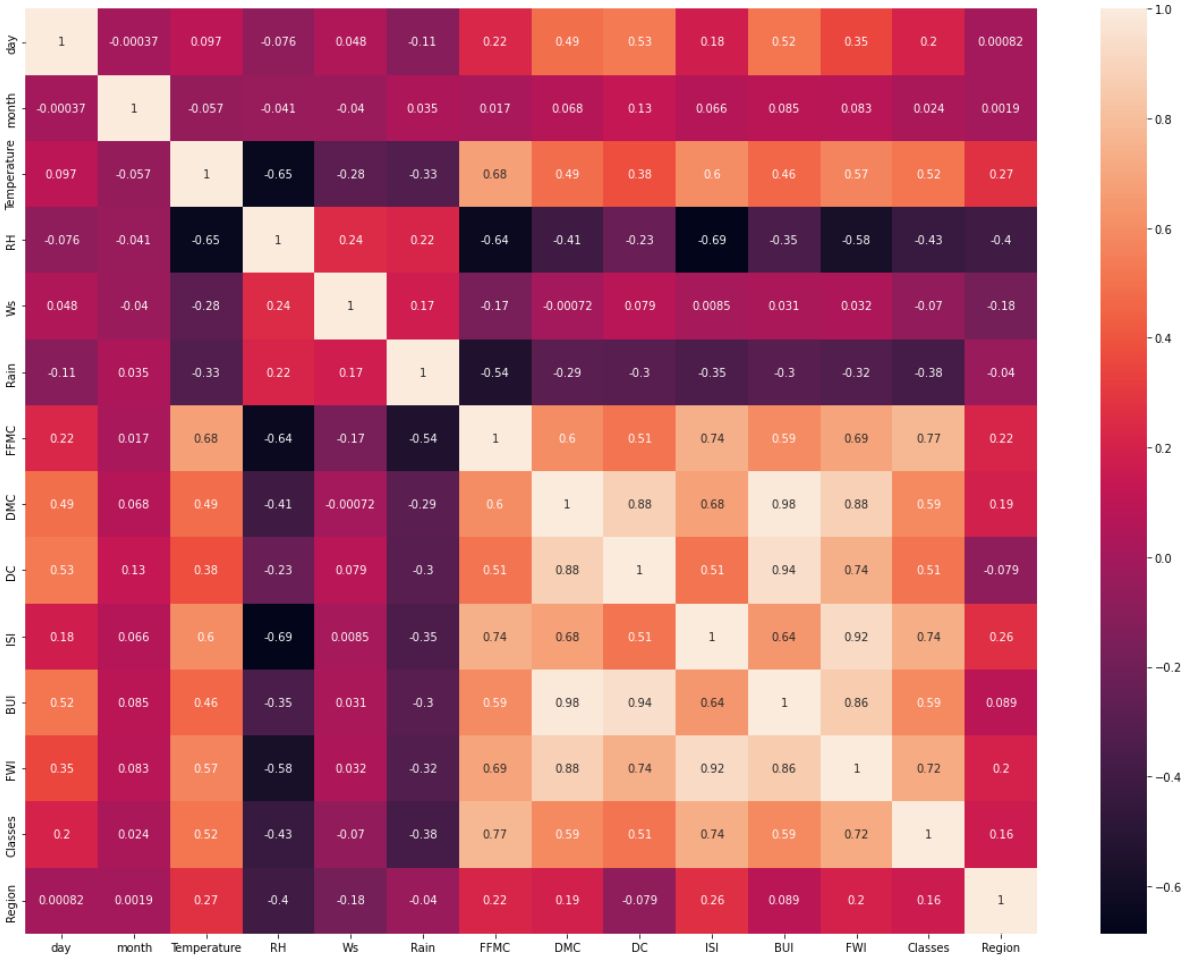


In [ ]:
```
## Drop Year columns
data.drop("year", axis = 1, inplace = True)
```

Heatmap of Correlation

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

Out[ ]:
```
<AxesSubplot: >
```



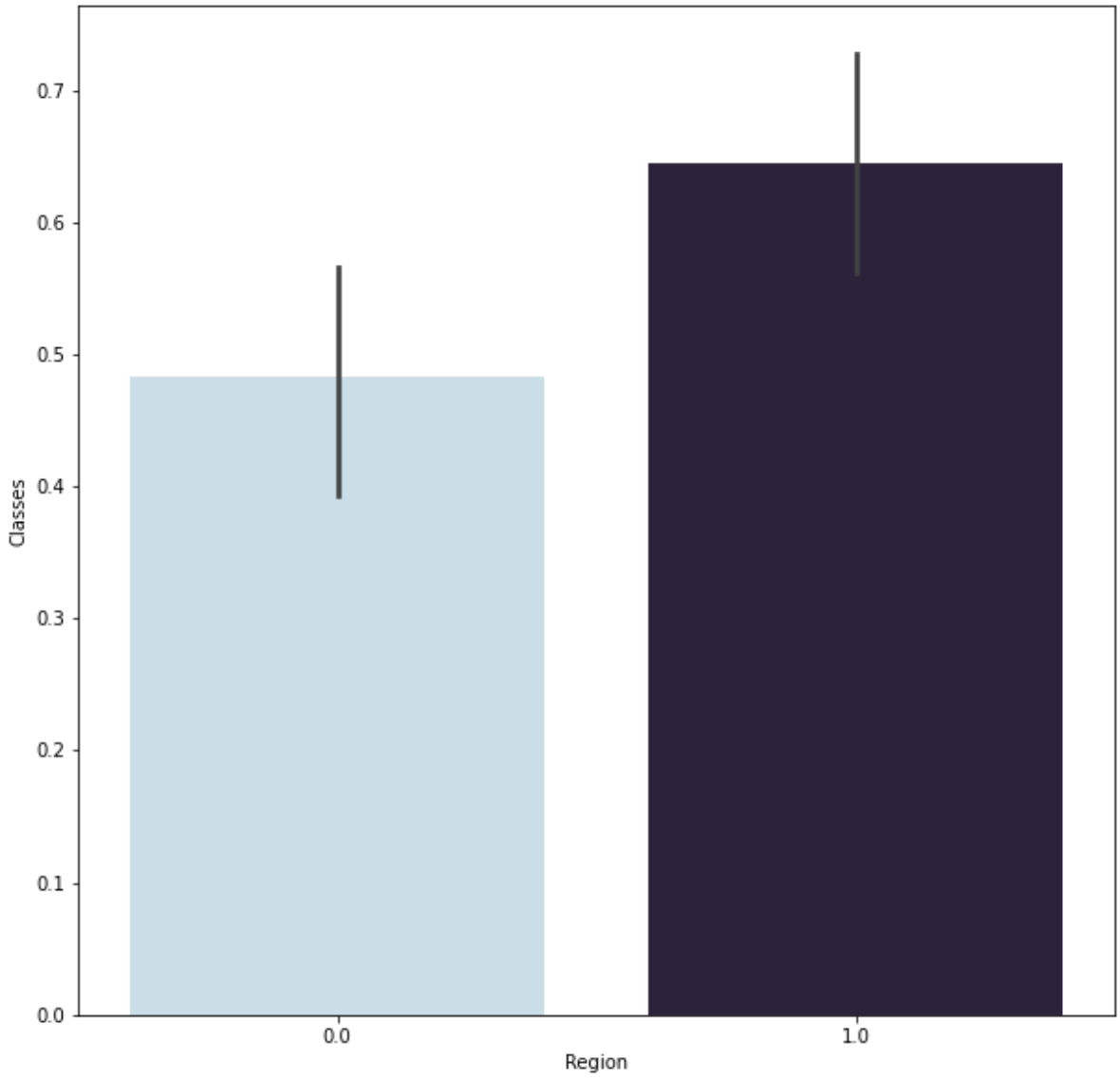
Visualisation of Target Feature

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

Out[ ]:
```
1    137
0    106
Name: Classes, dtype: int64
```

In [ ]:
```
plt.figure(figsize = (10,10))
sns.barplot(x= 'Region', y = 'Classes', data= data, palette="ch:s=.25,rot=-.25")
```

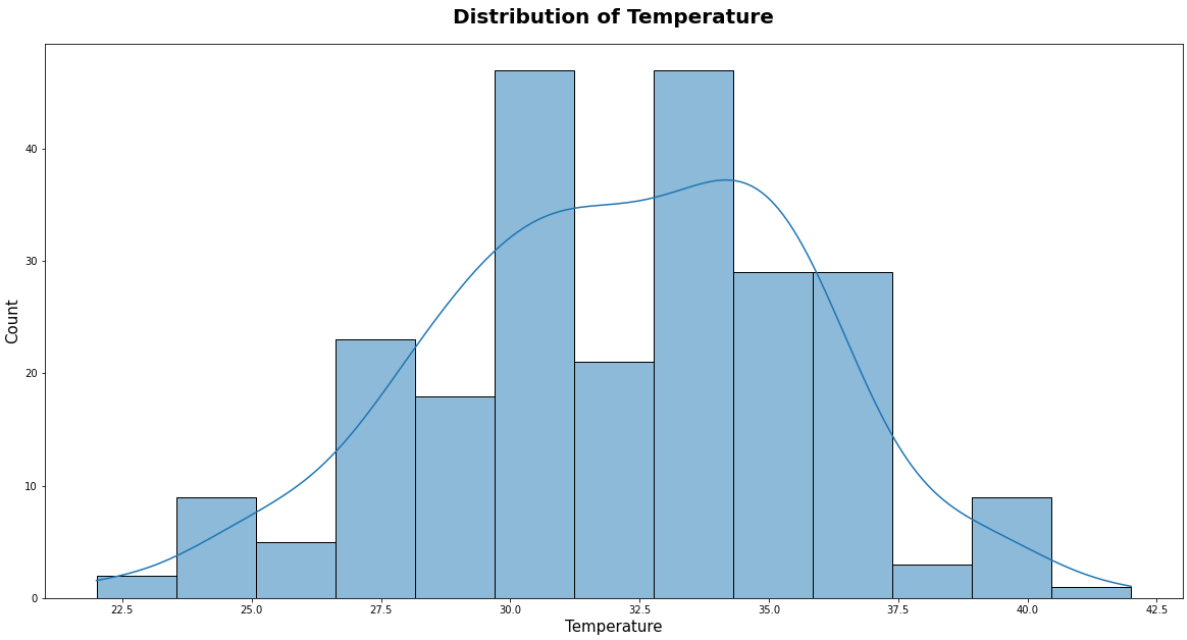
Out[ ]:
```
<AxesSubplot: xlabel='Region', ylabel='Classes'>
```



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

Visualisation of Tmperature Feature

In [ ]:
```
plt.figure(figsize = (20,10))
sns.histplot(data['Temperature'],kde = True, palette="ch:s=.25,rot=-.25")
plt.title("Distribution of Temperature",weight = 'bold',fontsize=20,pad=20)
plt.xlabel("Temperature",fontsize=15)
plt.ylabel("Count",fontsize=15)
plt.show()
```

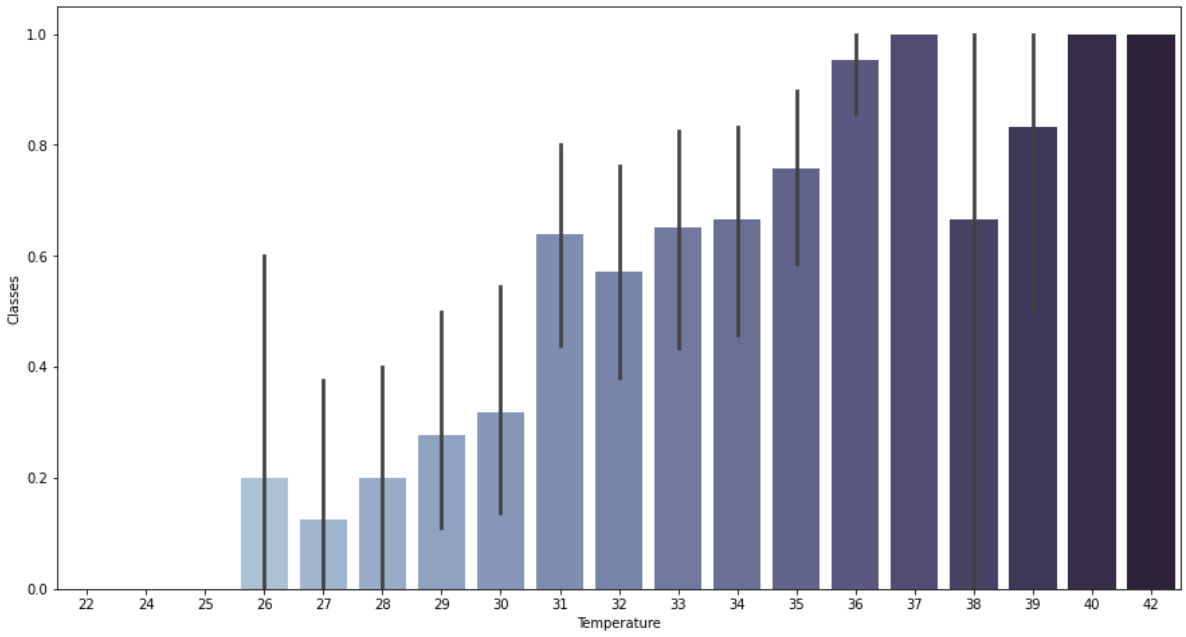


- `Temperature` occur most of the time in range 32.5 to 35.0

Highest Temperature Attained

In [ ]:
```
plt.figure(figsize = (15,8))
sns.barplot(x="Temperature",y="Classes",data=data, palette="ch:s=.25,rot=-.25")
```

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

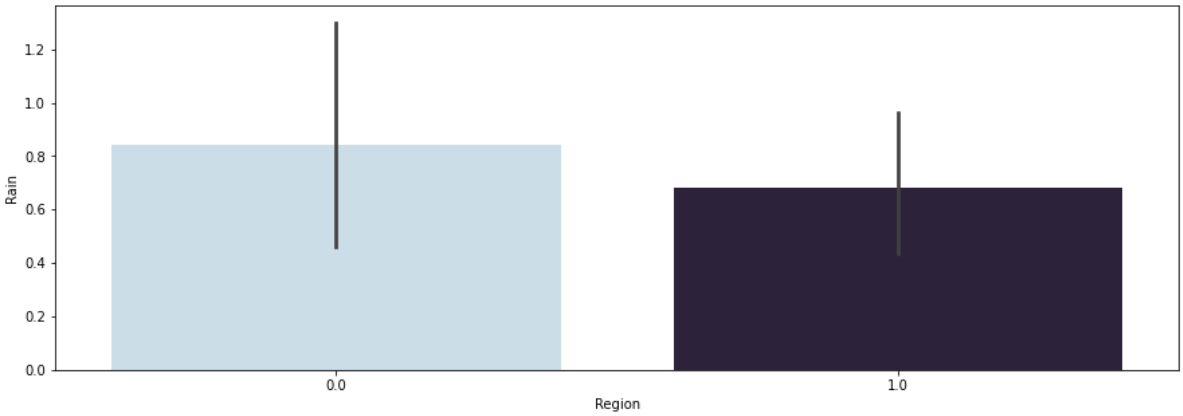


- Highest Temperature is `37,40` and `42`

Which region is mostly effected by rain

In [ ]:
```
plt.figure(figsize = (15,5))
sns.barplot(x = data['Region'], y = data['Rain'], palette="ch:s=.25,rot=-.25")
```

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

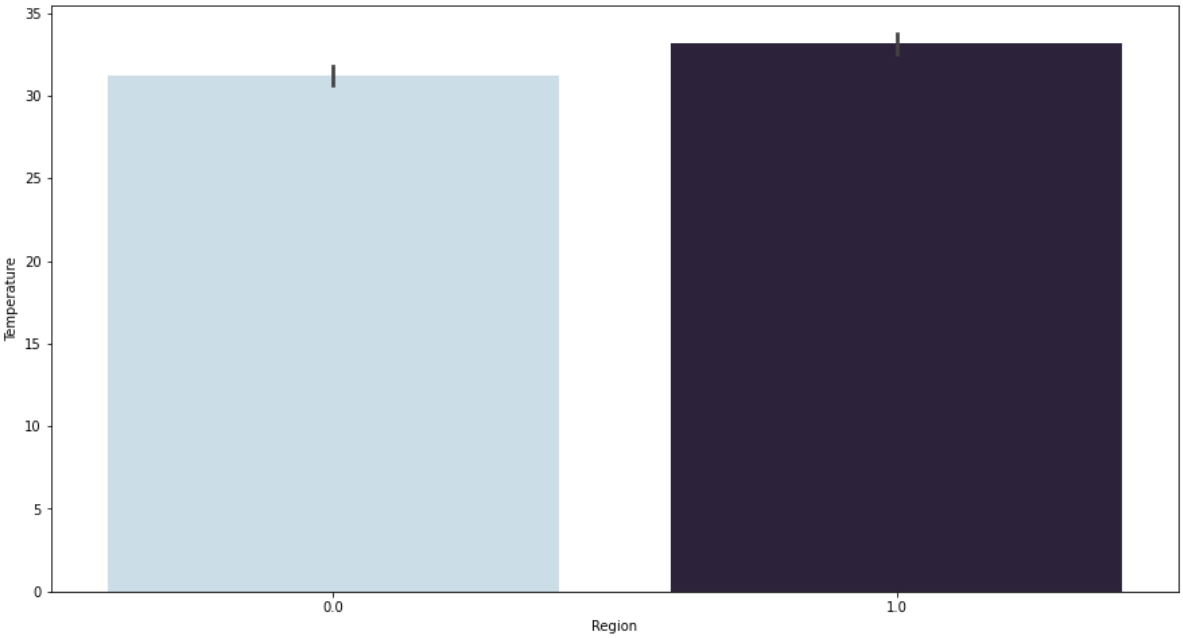


- `Bejaia region` is the region in which most of the time rain happens

Which region is mostly effected by Temperature

In [ ]:
```
plt.figure(figsize = (15,8))
sns.barplot(x = data['Region'], y = data['Temperature'], palette="ch:s=.25,rot=-.2
```

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

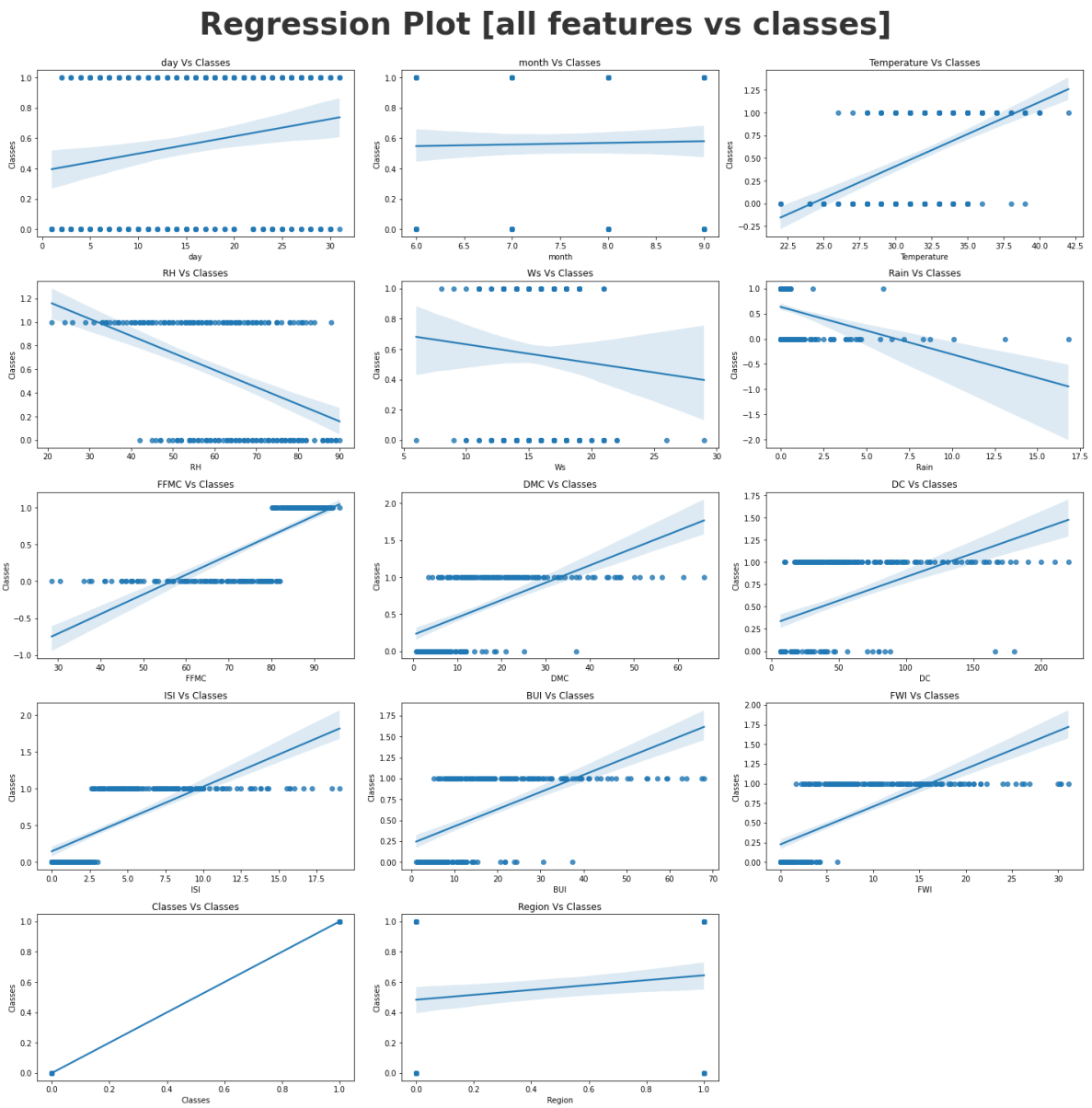


- `Sidi - Bel Abbes region` mostly effected by temperature

Regression Plot

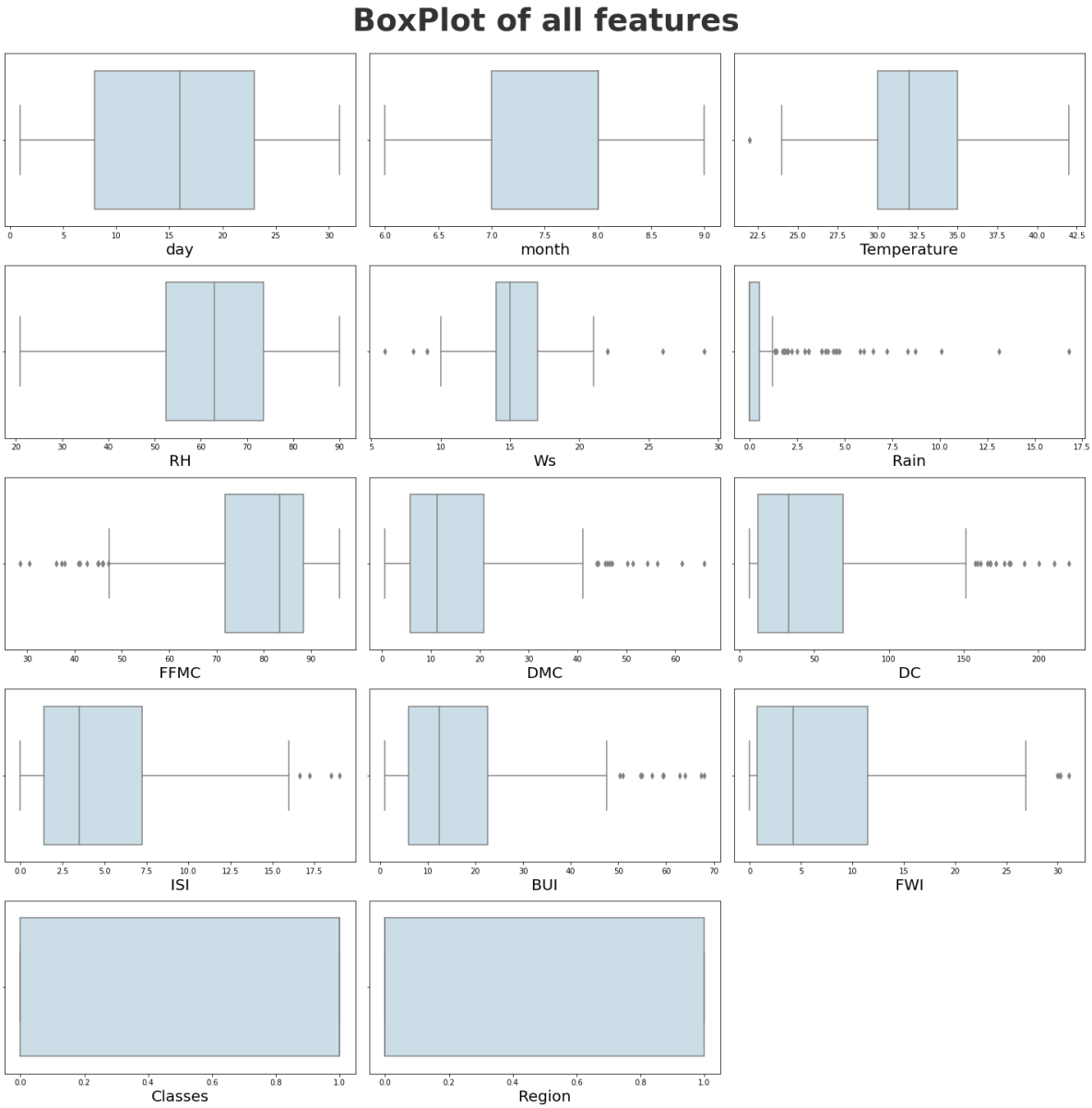
In [ ]:
```
plt.figure(figsize = (20,20))
plt.suptitle('Regression Plot [all features vs classes]', fontsize = 40, fontweigh

for i in range(0, len(data.columns)):
    plt.subplot(5,3,i+1)
    sns.regplot(x= data[data.columns[i]],y = data['Classes'], data = data)
    plt.xlabel(data.columns[i])
    plt.ylabel("Classes")
```

```
    plt.title("{} Vs Classes".format(data.columns[i]))
    plt.tight_layout()
```



Boxplot to find outliers

In [ ]:
```
plt.figure(figsize = (20,20))
plt.suptitle('BoxPlot of all features', fontsize = 40, fontweight = "bold", alpha =

for i in range(0, len(data.columns)):
    plt.subplot(5,3,i+1)
    sns.boxplot(x= data[data.columns[i]], data = data, palette="ch:s=.25,rot=-.25"
    plt.xlabel(data.columns[i],fontsize = 20)
    #plt.ylabel("Classes")
    #plt.title("{} .format(data.columns[i]))
    plt.tight_layout()
```



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

Handling the Outliers

In [ ]:
```
def find_boundaries(data,column):
    IQR = data[column].quantile(0.75) - data[column].quantile(0.25)
    lower_boundary = data[column].quantile(0.25) - (1.5 * IQR)
    higher_boundary = data[column].quantile(0.75) + (1.5 * IQR)
    print(column, "---", "IQR --->",IQR)
    print("Lower Boundary:",lower_boundary)
    print("Higher Boundary:",  higher_boundary)
    print("-------------------------------------------------------------------------")
    data.loc[data[column] <= lower_boundary, column] = lower_boundary
    data.loc[data[column] >= higher_boundary, column] =  higher_boundary
```

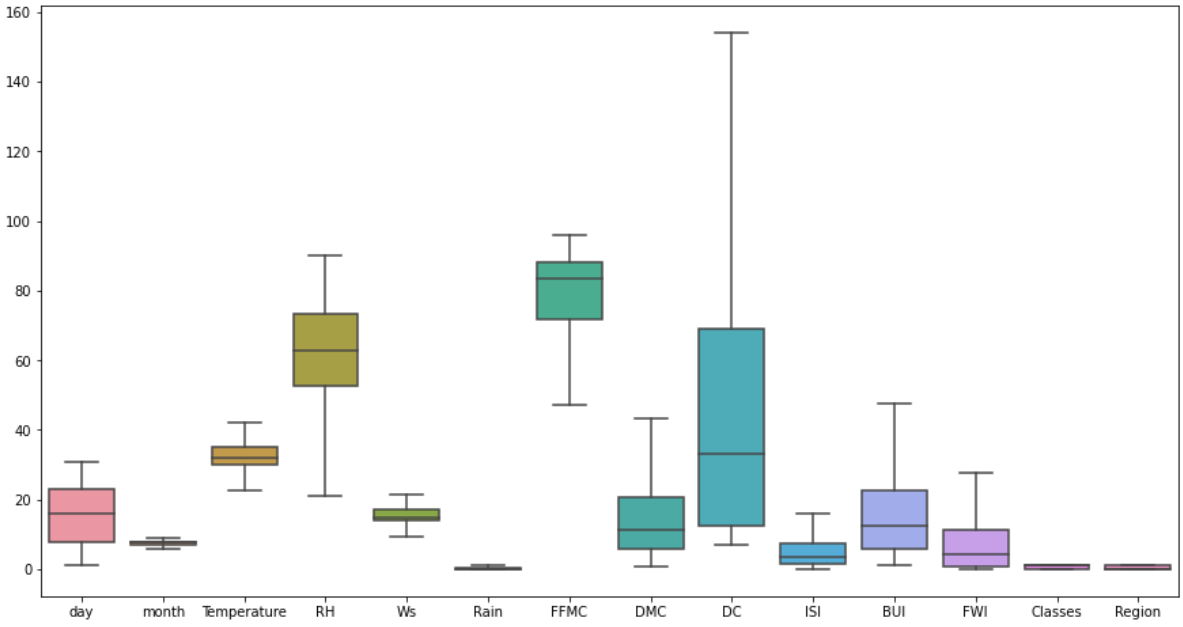
In [ ]:
```
for columns in data:
    find_boundaries(data,columns)
```

```
day --- IQR ---> 15.0
Lower Boundary: -14.5
Higher Boundary: 45.5
--------------------------------------------------------------------------
month --- IQR ---> 1.0
Lower Boundary: 5.5
Higher Boundary: 9.5
--------------------------------------------------------------------------
Temperature --- IQR ---> 5.0
Lower Boundary: 22.5
Higher Boundary: 42.5
--------------------------------------------------------------------------
RH --- IQR ---> 21.0
Lower Boundary: 21.0
Higher Boundary: 105.0
--------------------------------------------------------------------------
Ws --- IQR ---> 3.0
Lower Boundary: 9.5
Higher Boundary: 21.5
--------------------------------------------------------------------------
Rain --- IQR ---> 0.5
Lower Boundary: -0.75
Higher Boundary: 1.25
--------------------------------------------------------------------------
FFMC --- IQR ---> 16.450000000000003
Lower Boundary: 47.17499999999999
Higher Boundary: 112.975
--------------------------------------------------------------------------
DMC --- IQR ---> 14.999999999999996
Lower Boundary: -16.699999999999992
Higher Boundary: 43.29999999999999
--------------------------------------------------------------------------
DC --- IQR ---> 56.74999999999999
Lower Boundary: -72.77499999999999
Higher Boundary: 154.22499999999997
--------------------------------------------------------------------------
ISI --- IQR ---> 5.85
Lower Boundary: -7.374999999999998
Higher Boundary: 16.025
--------------------------------------------------------------------------
BUI --- IQR ---> 16.65
Lower Boundary: -18.974999999999998
Higher Boundary: 47.625
--------------------------------------------------------------------------
FWI --- IQR ---> 10.75
Lower Boundary: -15.425
Higher Boundary: 27.575
--------------------------------------------------------------------------
Classes --- IQR ---> 1.0
Lower Boundary: -1.5
Higher Boundary: 2.5
--------------------------------------------------------------------------
Region --- IQR ---> 1.0
Lower Boundary: -1.5
Higher Boundary: 2.5
--------------------------------------------------------------------------
```

Rechecking the outliers after dropping it

In [ ]:
```
plt.figure(figsize = (15,8))
sns.boxplot(data = data)
```

Out[ ]: <AxesSubplot: >



- Outlier is not present in any of the feature

Creating Independent and Dependent Features

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

Independent Features

In [ ]:
```
X.head()
```

Out[ ]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
0	1.0	6.0	29.0	57	18.0	0.00	65.700	3.4	7.6	1.3	3.4	0.5	0.0
1	2.0	6.0	29.0	61	13.0	1.25	64.400	4.1	7.6	1.0	3.9	0.4	0.0
2	3.0	6.0	26.0	82	21.5	1.25	47.175	2.5	7.1	0.3	2.7	0.1	0.0
3	4.0	6.0	25.0	89	13.0	1.25	47.175	1.3	6.9	0.0	1.7	0.0	0.0
4	5.0	6.0	27.0	77	16.0	0.00	64.800	3.0	14.2	1.2	3.9	0.5	0.0

Dependent Features

In [ ]:
```
y.head()
```

Out[ ]:
```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Classes, dtype: float64
```

Importing sklearn libraries for machine learning

In [ ]:
```
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_s
```

Train Test Split

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_stat
```

Logistic Regression Model Training

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

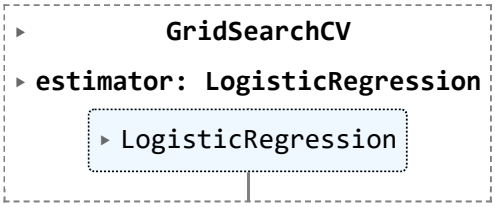
In [ ]:
```
from sklearn.model_selection import GridSearchCV
parameter = {'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,40,50],'
```

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

Standardizing or Feature Selection

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

Out[ ]:



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

```
{'C': 5, 'max_iter': 100, 'penalty': 'l2'}
```

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

```
0.9647058823529411
```

Prediction

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

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

Accuracy Score

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

```
0.958904109589041
```

Classification Report

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

```
              precision    recall  f1-score   support

         0.0       0.93      0.96      0.95        28
         1.0       0.98      0.96      0.97        45

    accuracy                           0.96        73
   macro avg       0.95      0.96      0.96        73
weighted avg       0.96      0.96      0.96        73
```

Performance Matrix

Confusion Matrix

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

Out[ ]:
```
array([[27,  1],
       [ 2, 43]], dtype=int64)
```

In [ ]:
```
true_positive = conf_Max[0][0]
false_positive = conf_Max[0][1]
false_negative = conf_Max[1][0]
true_negative = conf_Max[1][1]
print('true_positive:',true_positive)
print('false_positive:',false_positive)
print('true_negative:',true_negative)
print('false_negative:',false_negative)
```

```
true_positive: 27
false_positive: 1
true_negative: 43
false_negative: 2
```

Calculate using theorical formula of Accuracy, Precision, Recall, F1-score

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

Out[ ]:
```
0.958904109589041
```

Precision

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

Out[ ]:
```
0.9642857142857143
```

Recall

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

Out[ ]:
```
0.9310344827586207
```

F1-Score

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

Out[ ]:
```
0.9473684210526316
```

Area Under Curve

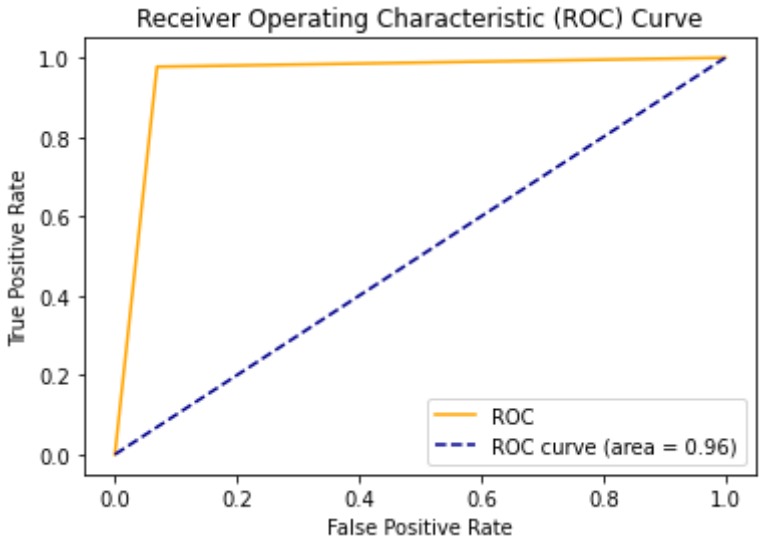
In [ ]:
```
auc = roc_auc_score(y_pred, y_test)
auc
```

Out[ ]:
```
0.9599206349206351
```

Roc

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

In [ ]:
```
plt.plot(fpr, tpr, color='orange', 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('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



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 Inbalance dataset from the original balanced dataset

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

Out[ ]:
```
(243, 15)
```

In [ ]:
```
# creating imbalance
# 1. splitting data in 9:1 percent ratio using train test split
X1 = pd.DataFrame(df, columns = ['day', 'month','Temperature','RH','Ws','Rain','FF
y1 = pd.DataFrame(df, columns = ['Classes'])
```

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

Out[ ]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	R
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	

In [ ]:
```
y1.head()
```

Out[ ]:

	Classes
0	0
1	0
2	0
3	0
4	0

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

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

Out[ ]:
```
((218, 13), (218, 1))
```

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

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

Out[ ]:

	Classes
158	1
186	1
11	1
75	1
132	1

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

Out[ ]:

	Classes
158	1
186	1
11	1
75	1
132	1

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

Out[ ]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
158	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0
186	2	8	40	34	14	0.0	93.3	10.8	21.4	13.8	10.6	13.5	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
132	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0

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

Out[ ]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region	Class
158	5	7	34	45	18	0.0	90.5	18.7	46.4	11.3	18.7	15.0	1.0	
186	2	8	40	34	14	0.0	93.3	10.8	21.4	13.8	10.6	13.5	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	
132	9	6	27	59	18	0.1	78.1	8.5	14.7	2.4	8.3	1.9	1.0	

In [ ]:
```
# combining X_test_imb with y_test_imb
test_imb = X_test_imb.join(pd.DataFrame(y_test_imb))
```

In [ ]:
```
test_imb.shape, train_imb.shape
```

Out[ ]:
```
((25, 14), (218, 14))
```

In [ ]:
```
# 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[ ]:

	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	2	8	40	34	14	0.0	93.3	10.8	21.4	13.8	10.6	13.5	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

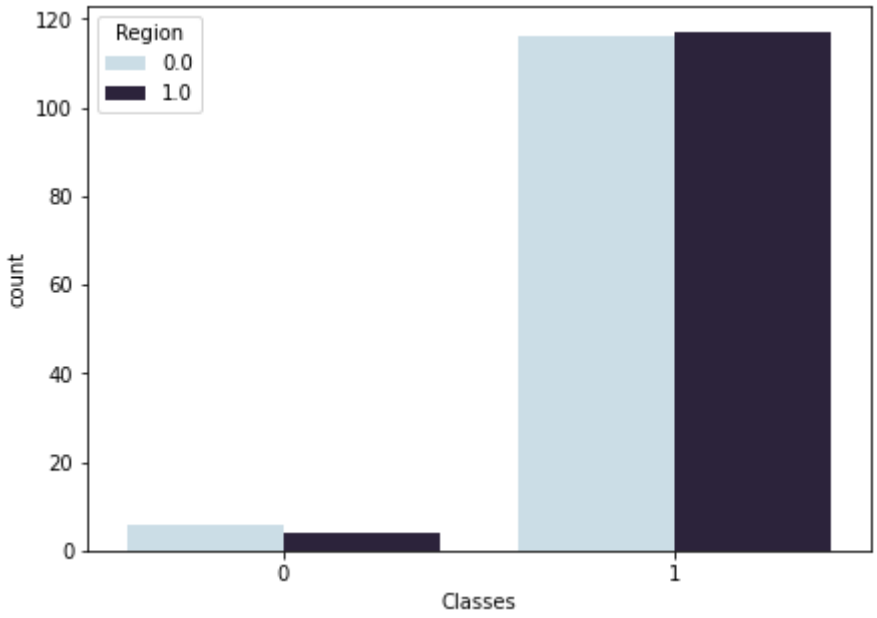
Checking the inbalancing

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

Out[ ]:
```
1    233
0     10
Name: Classes, dtype: int64
```

In [ ]:
```
# 0: 'Bejaia region' and 1: 'Sidi Bel-abbes region'
plt.figure(figsize=(7,5))
sns.countplot(data = df_imb, x='Classes', hue='Region',palette="ch:s=.25,rot=-.25"
```

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



Logistic regression on inbalance data

In [ ]:
```
# Seperating Independent and Dependent Feature
y1 = df_imb['Classes']
X1 = df_imb.drop(columns='Classes')
```

In [ ]:
```
# Handling Inbalance dataset by doing Upsampling

#for Upsampling
from imblearn.combine import SMOTETomek
```

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

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

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

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

Out[ ]:
```
((462, 13), (462,))
```

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

Out[ ]:

	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	2	8	40	34	14	0.0	93.3	10.8	21.4	13.8	10.6	13.5	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 [ ]:
```
data_bal.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 462 entries, 0 to 461
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          462 non-null    int64  
 1   month        462 non-null    int64  
 2   Temperature  462 non-null    int64  
 3   RH           462 non-null    int64  
 4   Ws           462 non-null    int64  
 5   Rain         462 non-null    float64
 6   FFMC         462 non-null    float64
 7   DMC          462 non-null    float64
 8   DC           462 non-null    float64
 9   ISI          462 non-null    float64
 10  BUI          462 non-null    float64
 11  FWI          462 non-null    float64
 12  Region       462 non-null    float64
 13  Classes      462 non-null    int64  
dtypes: float64(8), int64(6)
memory usage: 50.7 KB
```

Statistical analysis on balanced dataset

In [ ]:
```
data_bal.describe().T.sort_values(by='std' , ascending = False)\
                      .style.background_gradient(cmap='GnBu')\
                      .bar(subset=["max"], color='#BB0000')\
                      .bar(subset=["mean",], color='green')
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	
DC	462.000000	34.945863	39.327681	6.900000	10.031819	17.000000	44.356671	220.40
RH	462.000000	65.311688	12.901018	21.000000	58.000000	66.000000	74.000000	90.00
FFMC	462.000000	72.862961	12.396395	28.600000	66.147513	71.630880	83.975000	96.00
BUI	462.000000	11.977233	11.990177	1.100000	3.943767	6.968680	16.493361	68.00
DMC	462.000000	10.574172	10.489520	0.700000	3.403368	6.747498	14.275362	65.90
day	462.000000	13.136364	8.524102	1.000000	6.000000	11.000000	20.000000	31.00
FWI	462.000000	4.059045	6.258516	0.000000	0.500000	0.671919	5.275000	31.10
ISI	462.000000	3.056536	3.504310	0.000000	1.100014	1.394974	3.950000	19.00
Temperature	462.000000	31.515152	3.112285	22.000000	29.000000	31.000000	34.000000	42.00
Ws	462.000000	14.764069	2.598716	6.000000	13.000000	15.000000	16.000000	29.00
Rain	462.000000	0.707544	1.491676	0.000000	0.000000	0.245454	0.893367	16.80
month	462.000000	7.439394	1.113712	6.000000	6.000000	7.000000	8.000000	9.00
Classes	462.000000	0.500000	0.500542	0.000000	0.000000	0.500000	1.000000	1.00
Region	462.000000	0.419552	0.445455	0.000000	0.000000	0.226789	1.000000	1.00

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

Out[ ]:

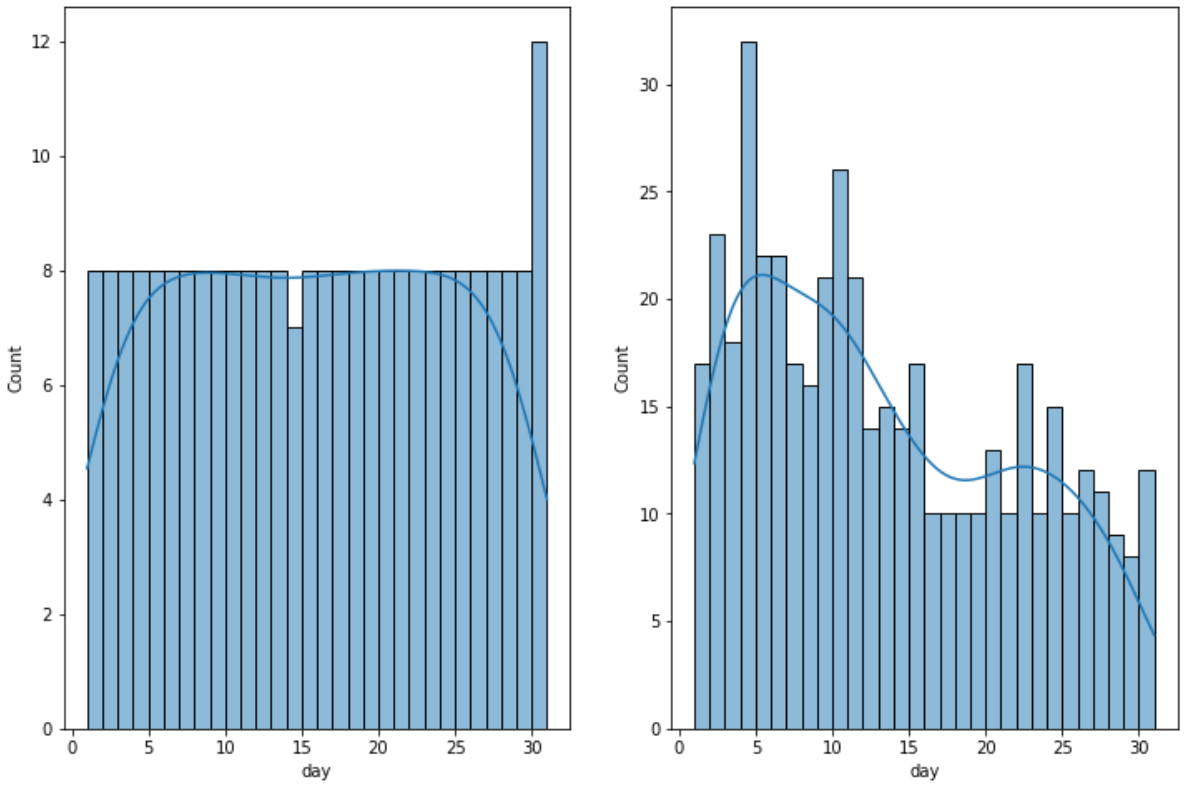
	day	month	Temperature	RH	Ws	Rain	FFMC	D
day	1.000000	-0.107320	0.321220	-0.155548	-0.064154	-0.022625	0.327835	0.594
month	-0.107320	1.000000	-0.092981	0.104735	-0.047298	0.021905	0.066278	-0.032
Temperature	0.321220	-0.092981	1.000000	-0.583157	-0.330115	-0.186737	0.624004	0.525
RH	-0.155548	0.104735	-0.583157	1.000000	0.030149	0.167300	-0.663478	-0.480
Ws	-0.064154	-0.047298	-0.330115	0.030149	1.000000	0.070438	-0.004014	0.086
Rain	-0.022625	0.021905	-0.186737	0.167300	0.070438	1.000000	-0.437343	-0.203
FFMC	0.327835	0.066278	0.624004	-0.663478	-0.004014	-0.437343	1.000000	0.663
DMC	0.594731	-0.032510	0.525637	-0.480403	0.086625	-0.203563	0.663535	1.000
DC	0.574985	0.071909	0.386239	-0.335709	0.167874	-0.244289	0.598120	0.888
ISI	0.285177	0.071730	0.548005	-0.650963	0.165874	-0.275653	0.769671	0.720
BUI	0.604690	-0.003777	0.493814	-0.442458	0.114467	-0.220791	0.661820	0.985
FWI	0.419683	0.064495	0.530075	-0.573893	0.170384	-0.252110	0.728945	0.874
Region	0.205881	-0.045563	0.322424	-0.325305	-0.136707	-0.021243	0.245795	0.279
Classes	0.330209	0.056423	0.222792	-0.283852	0.297672	0.044036	0.444318	0.430

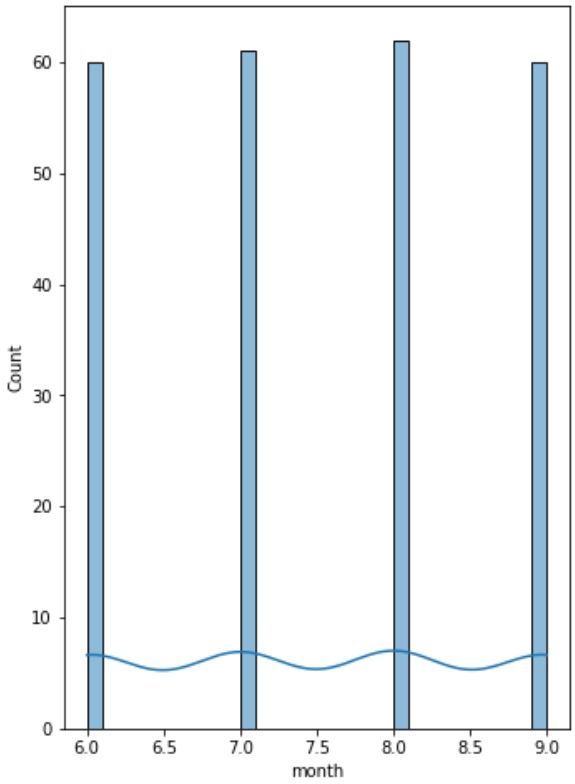
In [ ]:
```
num_data_bal = [fea for fea in data_bal.columns if data_bal[fea].dtype != 'O']
num_data_bal
```

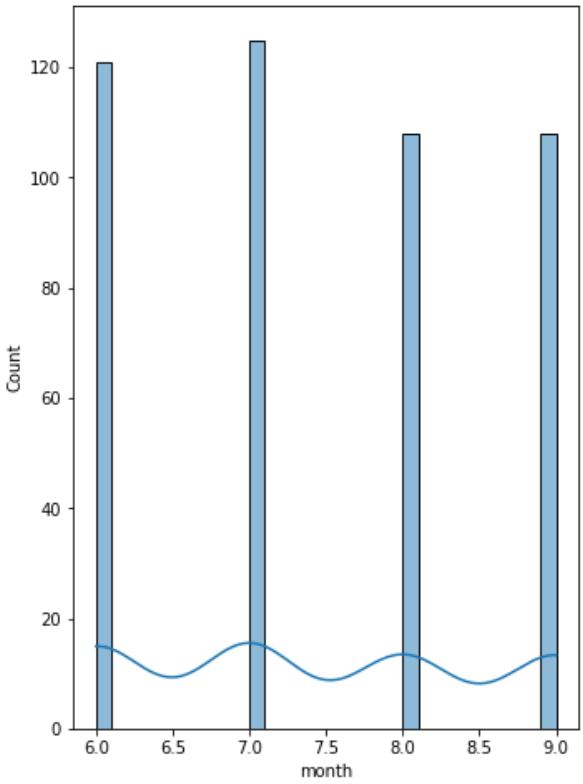
Out[ ]:
```
['day',
 'month',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Region',
 'Classes']
```

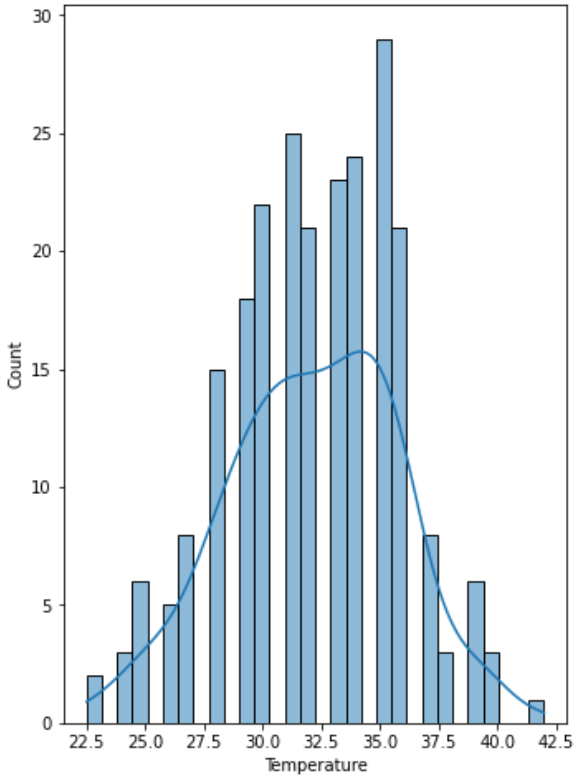
Comparing the distribution in feature for Original and Balanced dataset

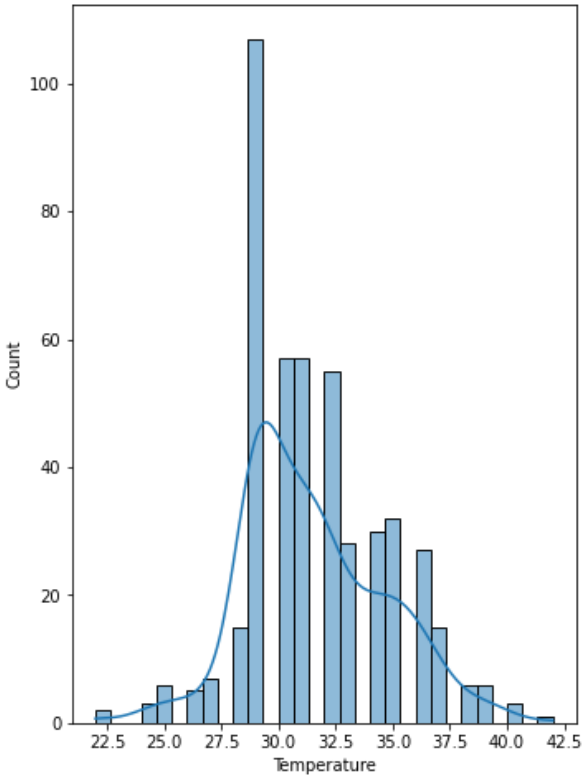
In [ ]:
```
for i in num_data_bal:
    plt.figure(figsize = (12,8))
    plt.subplot(121)
    sns.histplot(data = data,x = i, kde = True,bins = 30)

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

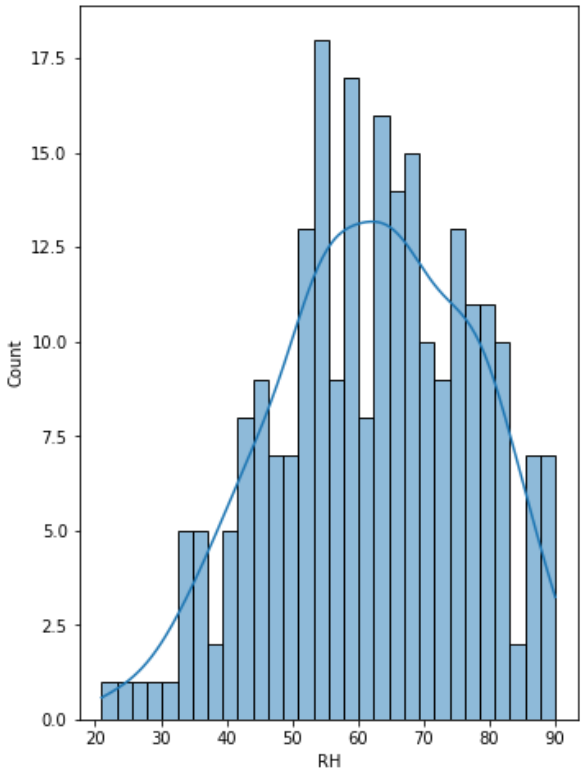


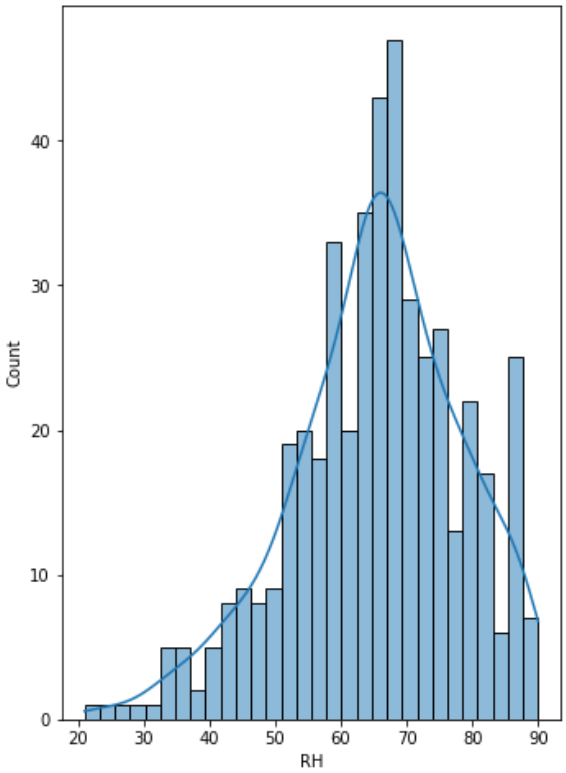

Count
month

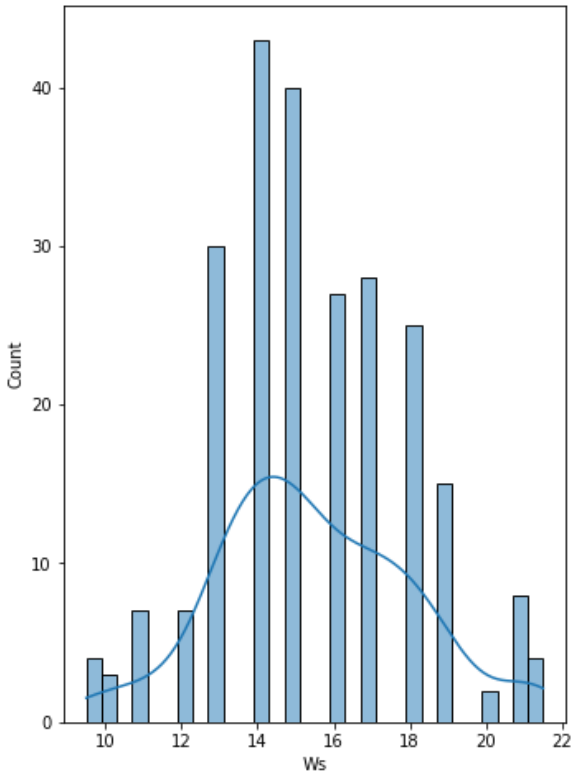

Count
month

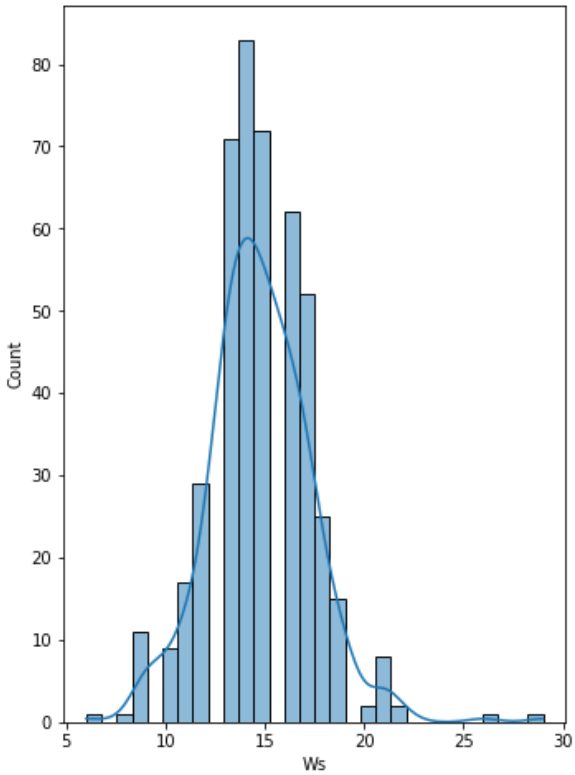

Count
Temperature

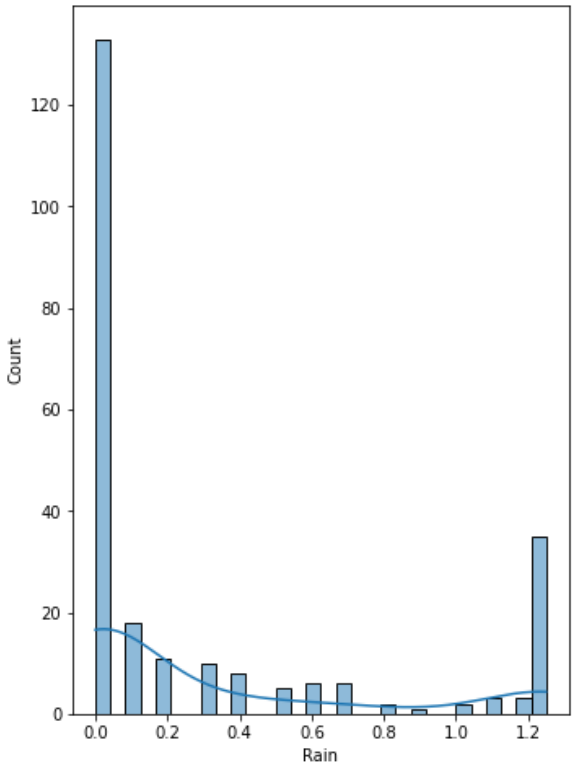

Count
Temperature

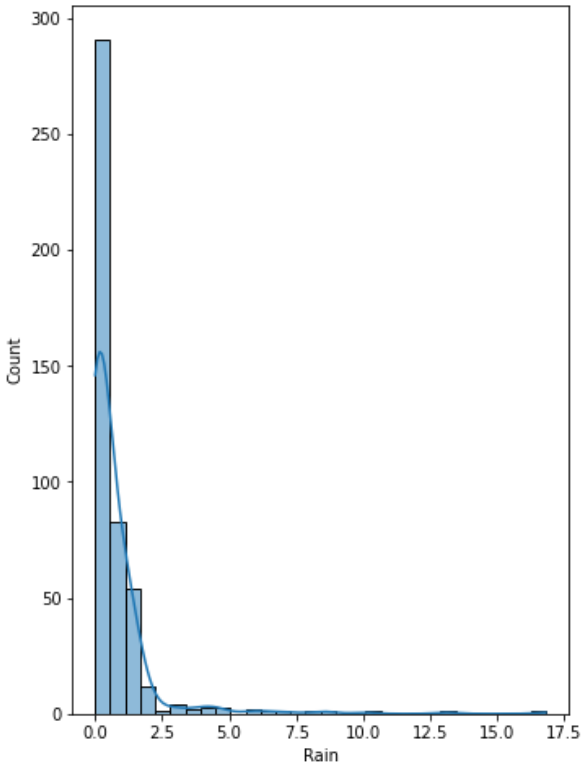

Count
RH

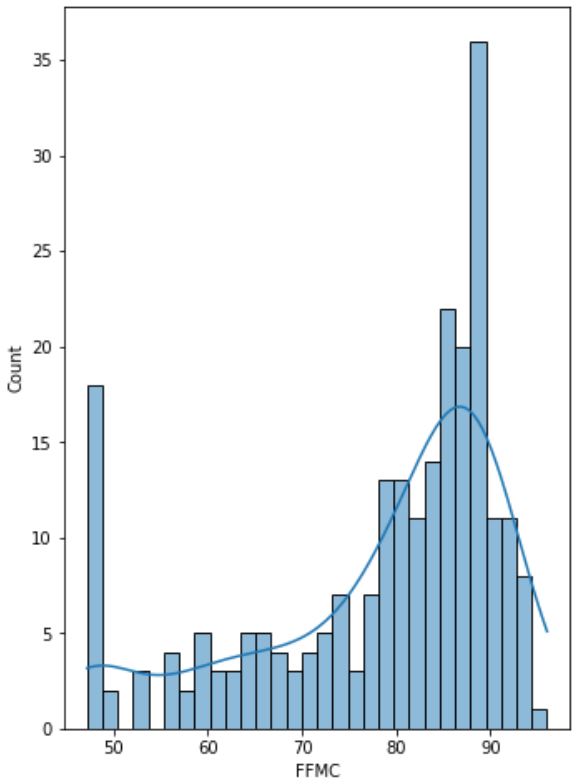

Count
RH

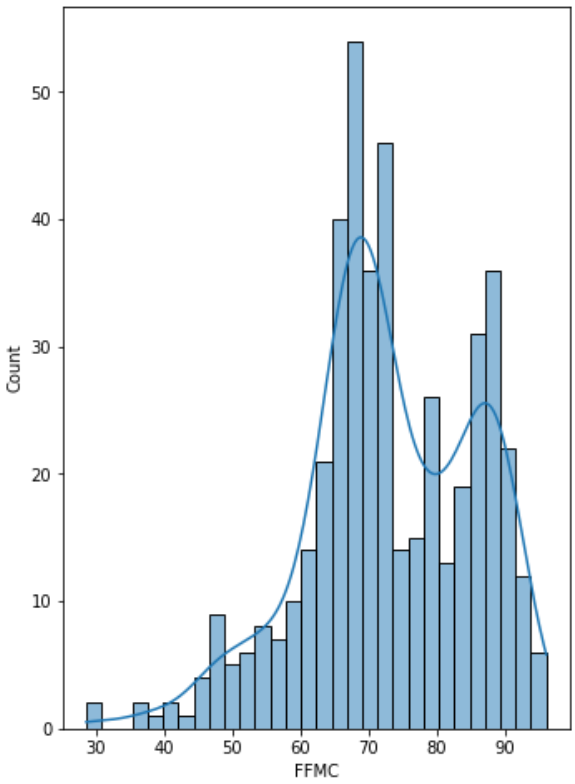

Count
Ws

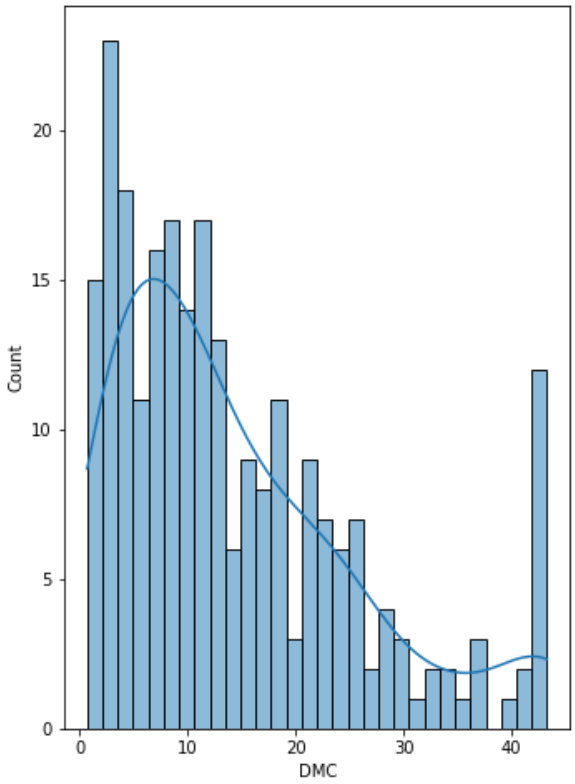

Count
Ws

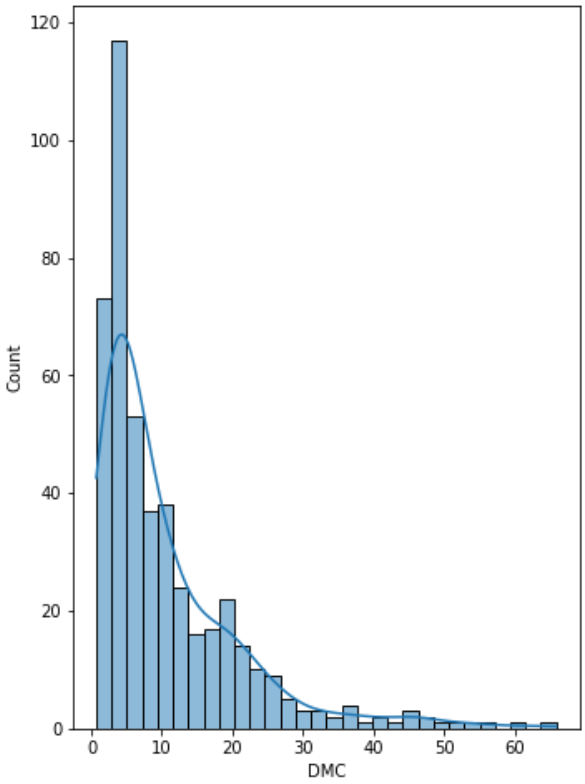

Count
Rain
0
20
40
60
80
100
120
0.0
0.2
0.4
0.6
0.8
1.0
1.2

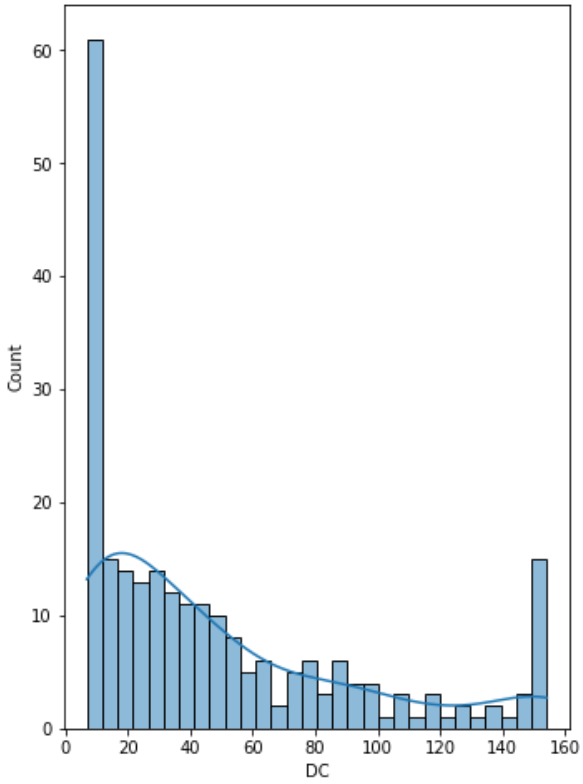

Count
Rain
0
50
100
150
200
250
300
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

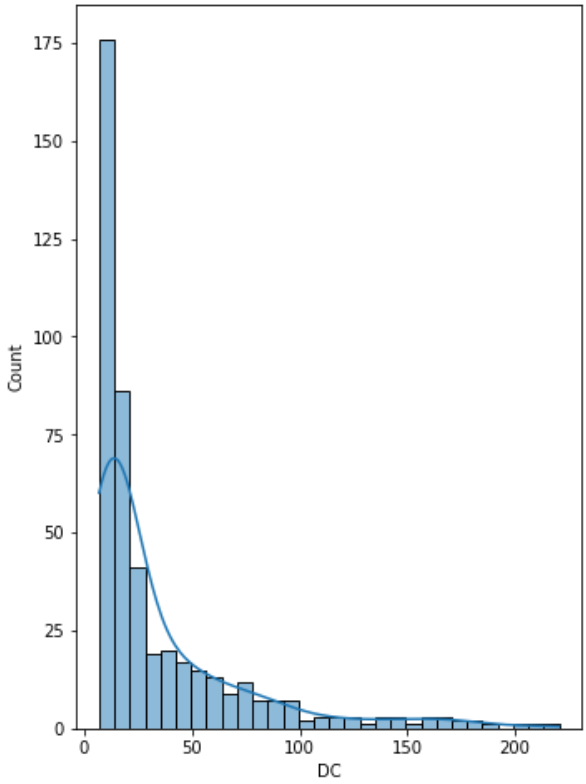

Count
FFMC
0
5
10
15
20
25
30
35
50
60
70
80
90

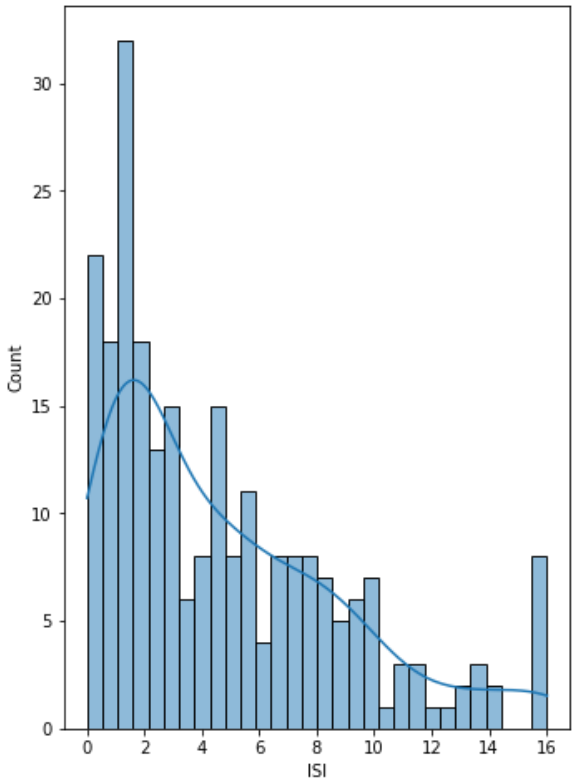

Count
FFMC
0
10
20
30
40
50
30
40
50
60
70
80
90

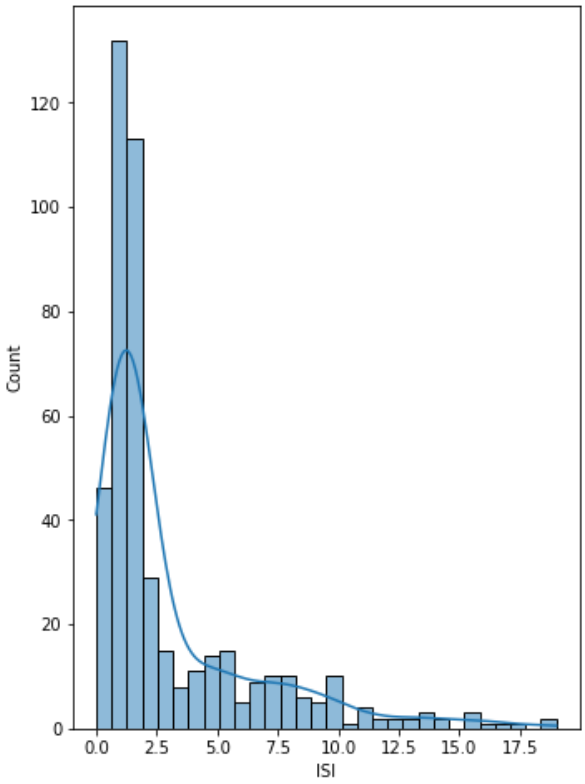

Count
DMC

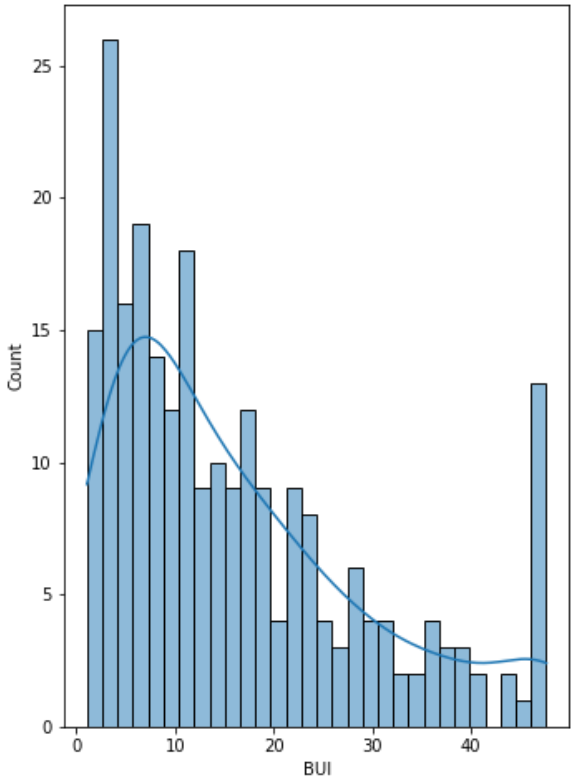

Count
DMC

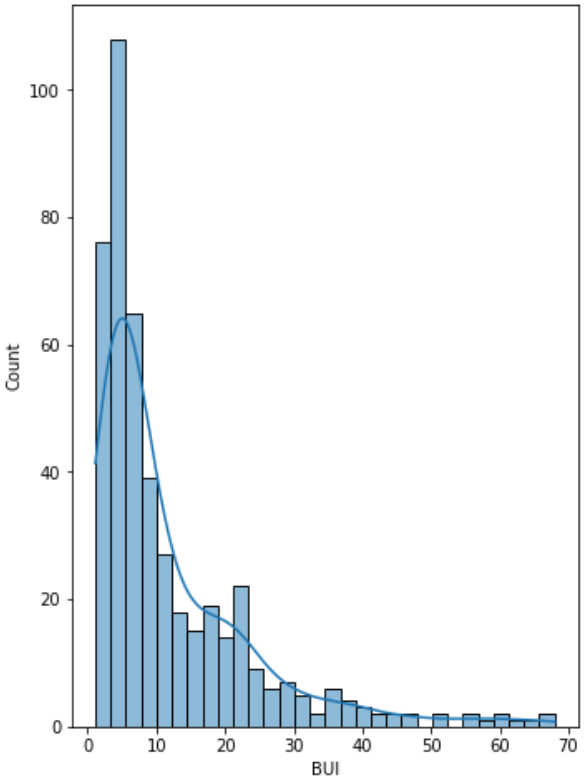

Count
DC

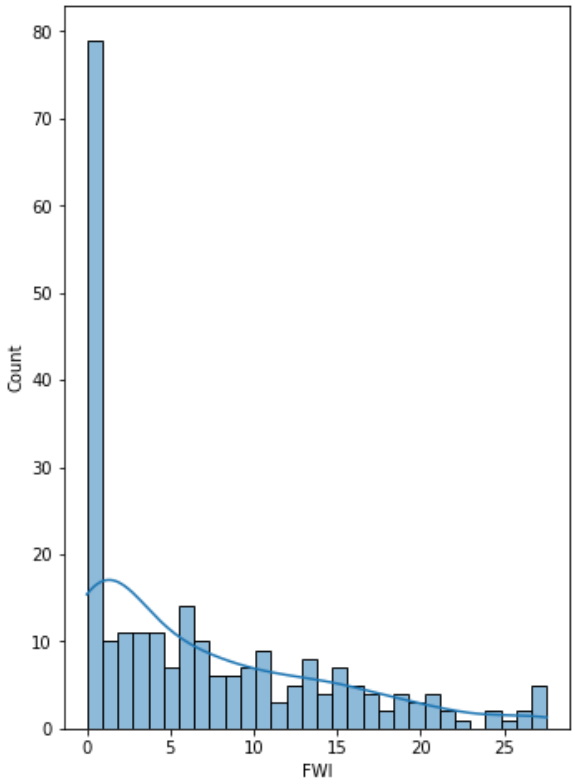

Count
DC

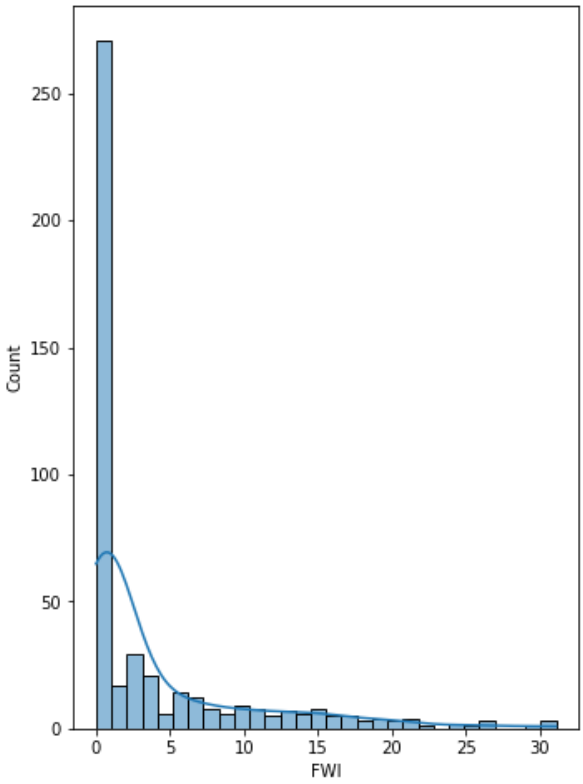

Count
ISI

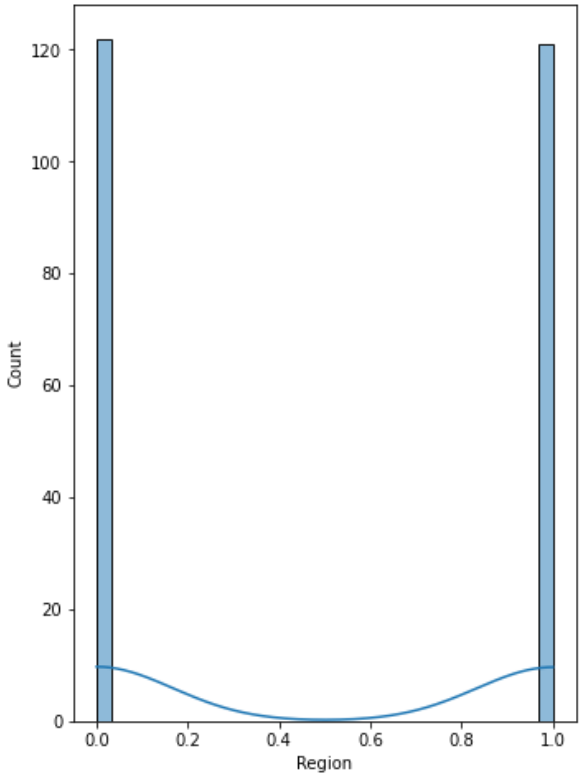

Count
ISI

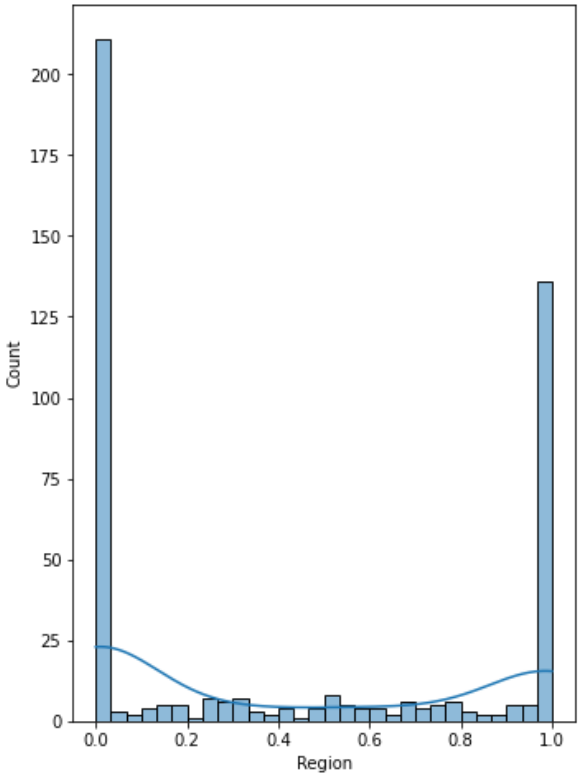

Count
BUI

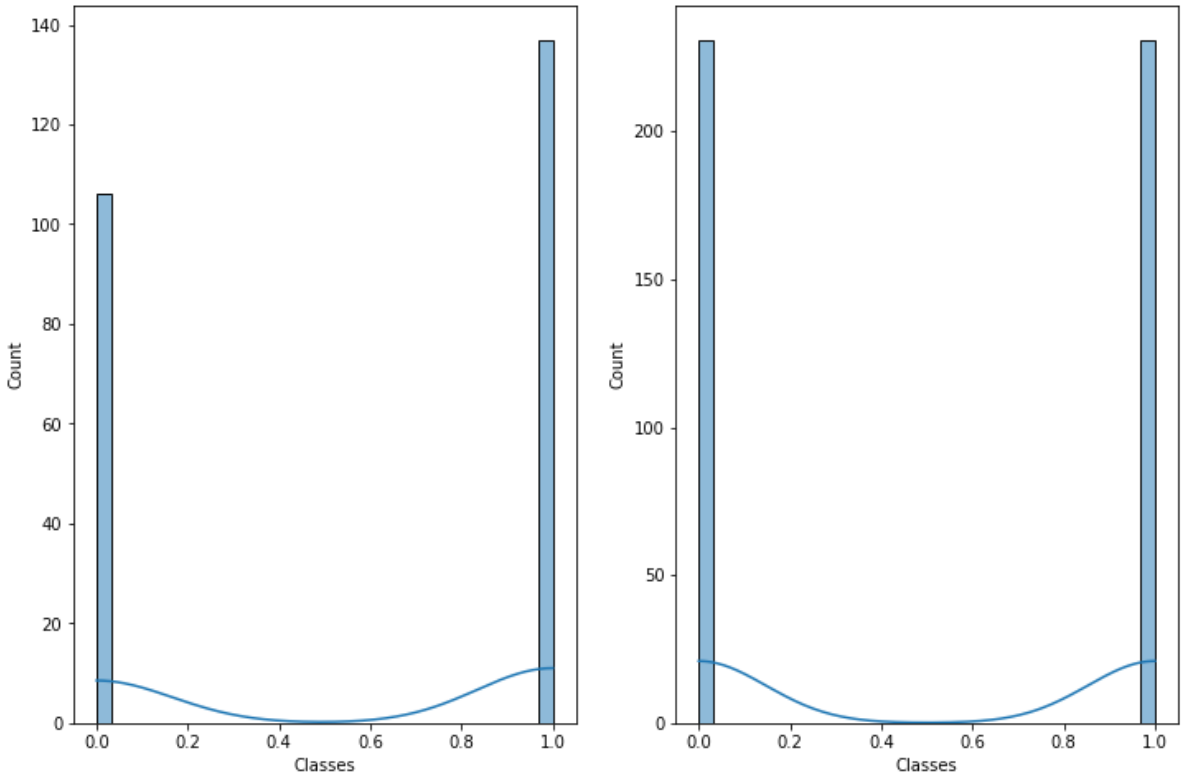

Count
BUI


Count
FWI


Count
FWI

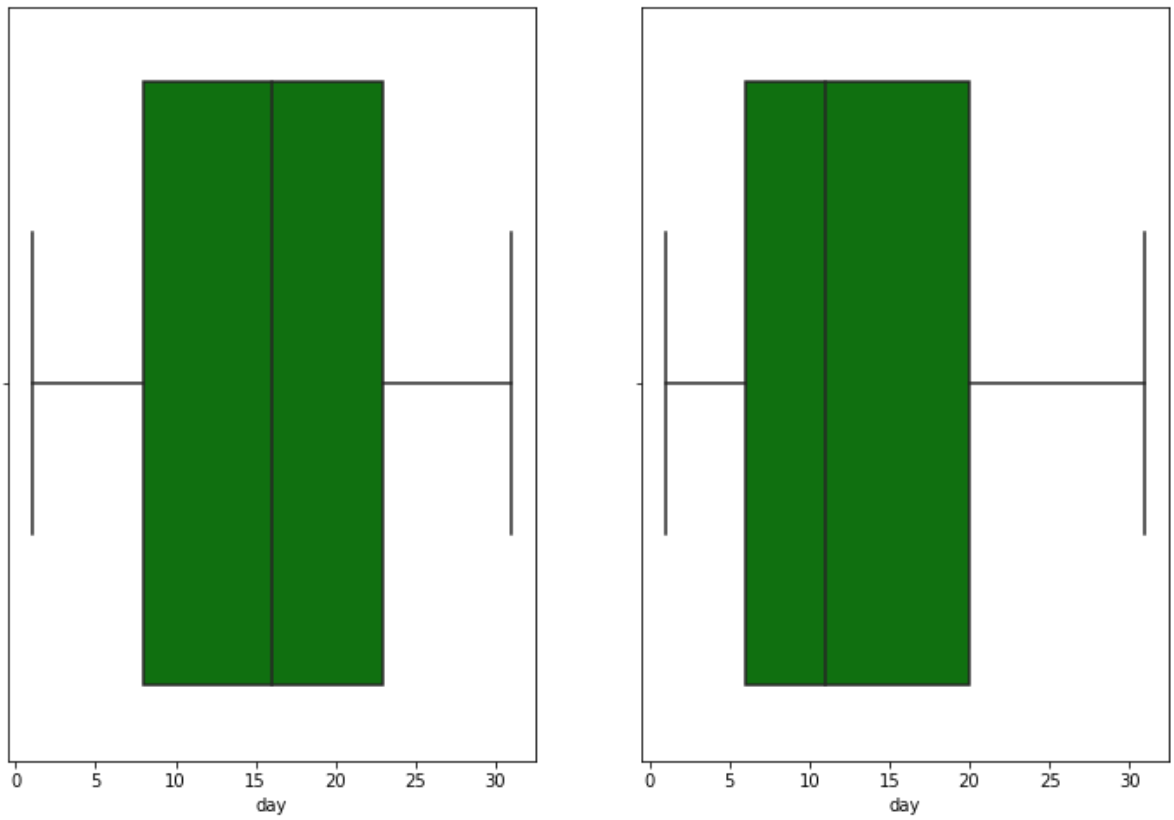

Count
Region

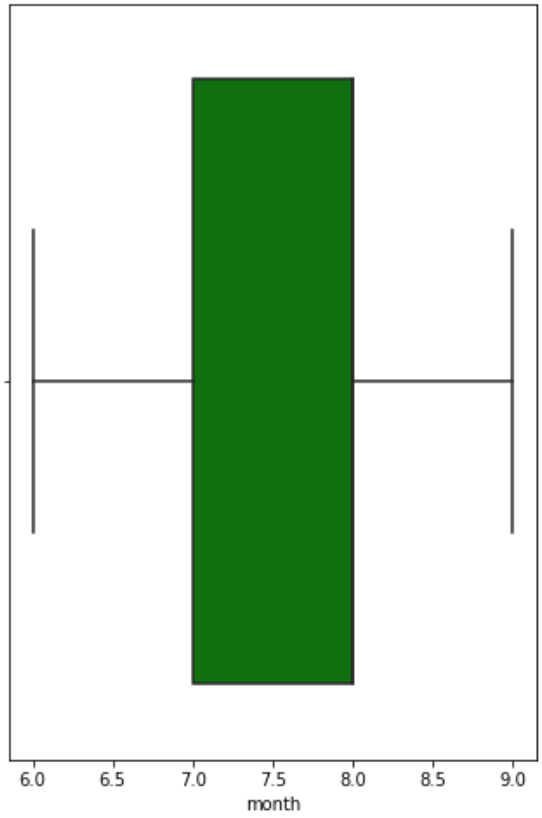

Count
Region

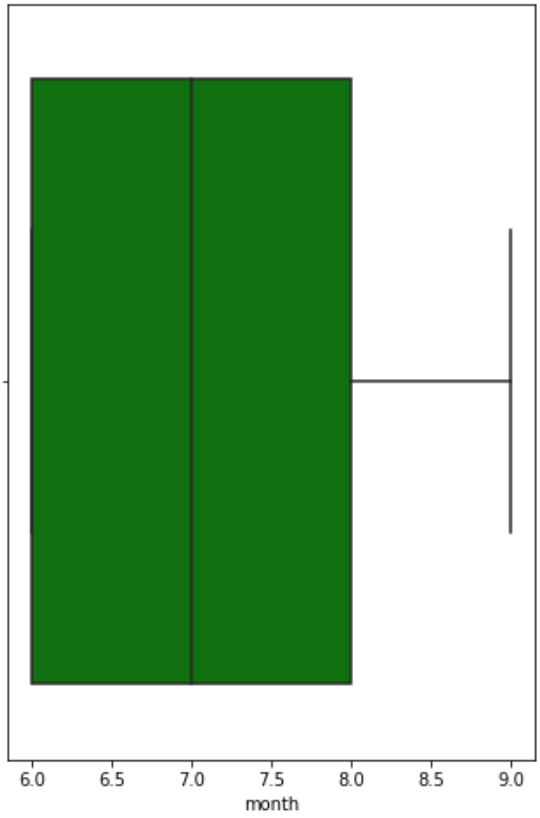


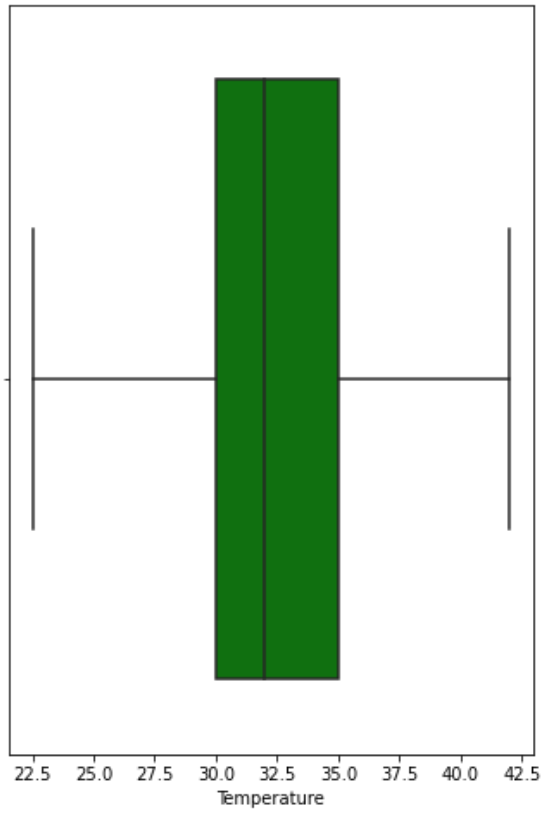
Checking the Outliers for Original and Balanced Dataset

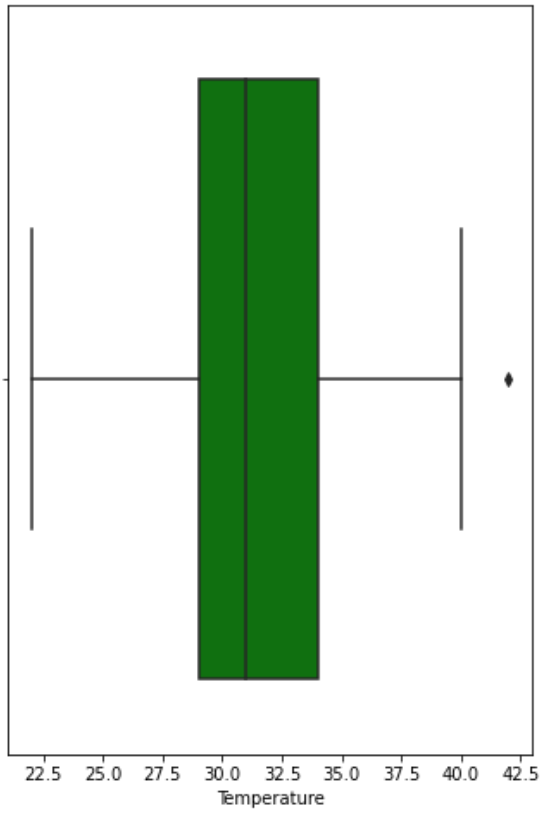
In [ ]:
```
for i in num_data_bal:
    plt.figure(figsize = (12,8))
    plt.subplot(121)
    sns.boxplot(data = data,x = i,color ='g')

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

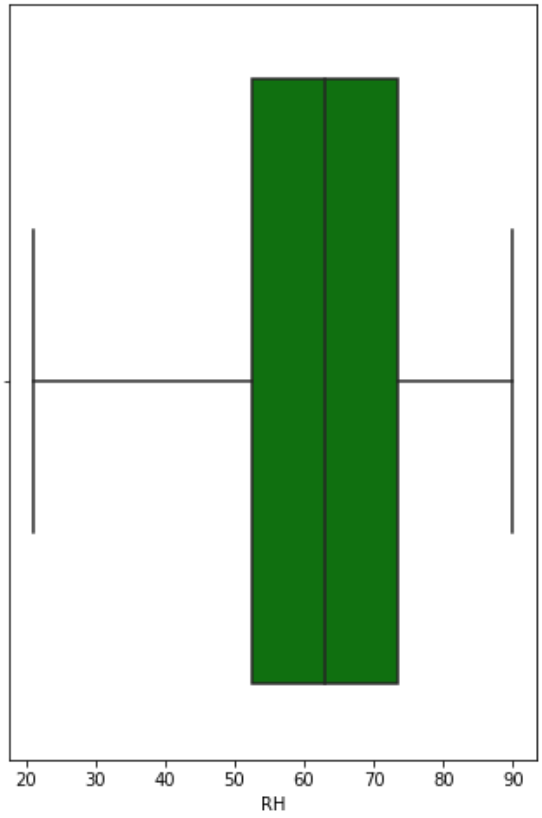


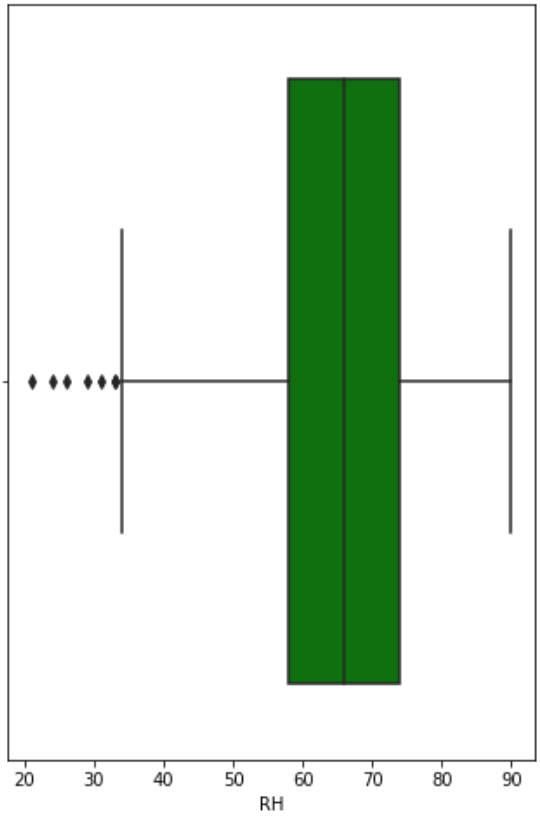

6.0
6.5
7.0
7.5
8.0
8.5
9.0
month

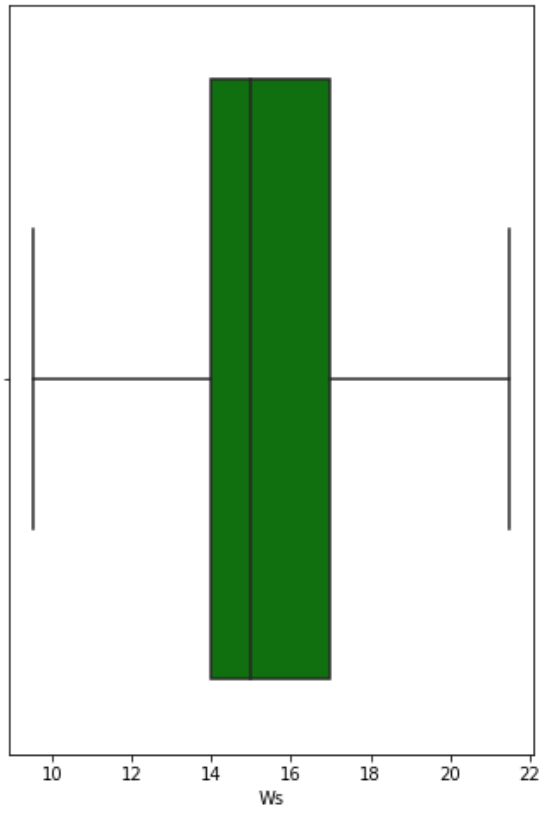

6.0
6.5
7.0
7.5
8.0
8.5
9.0
month

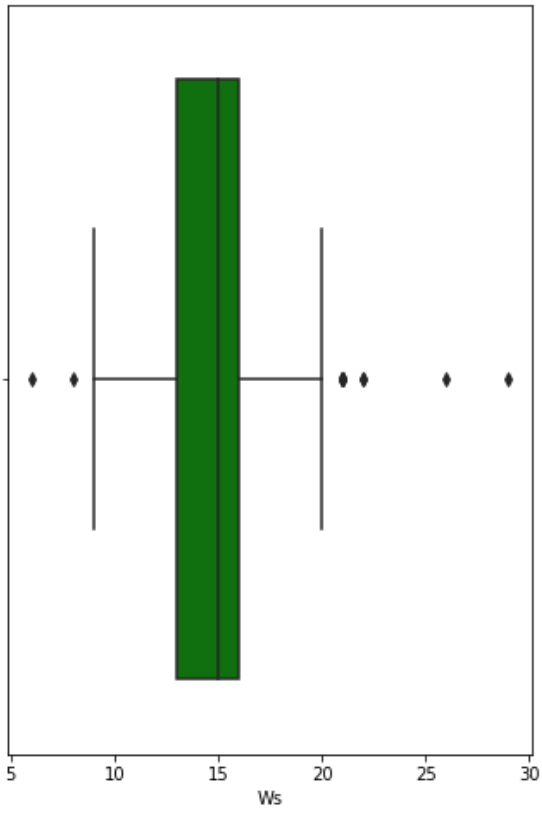

22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
Temperature

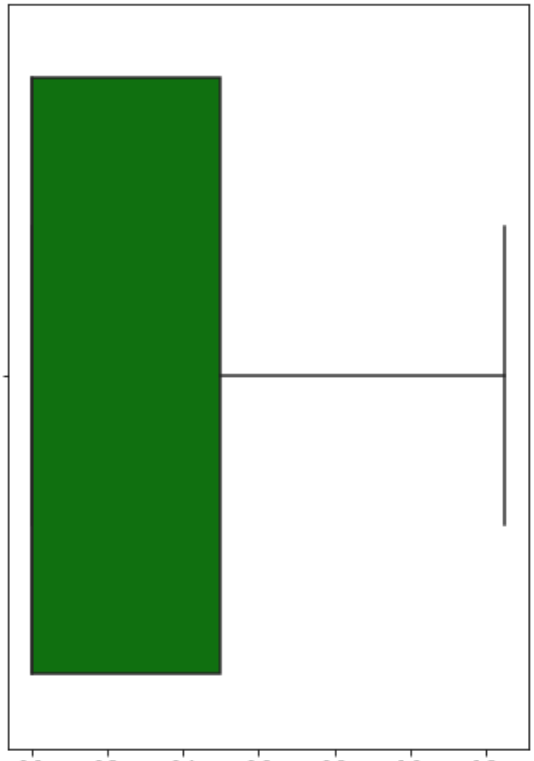

22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
Temperature

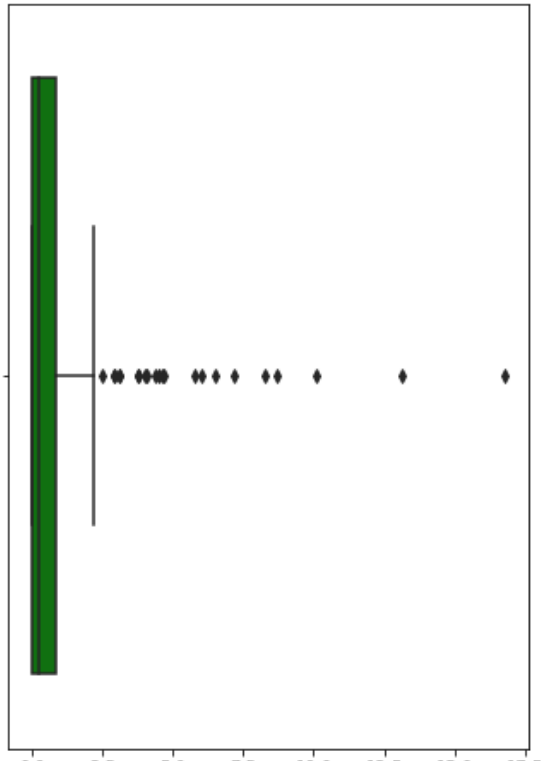

20
30
40
50
60
70
80
90
RH

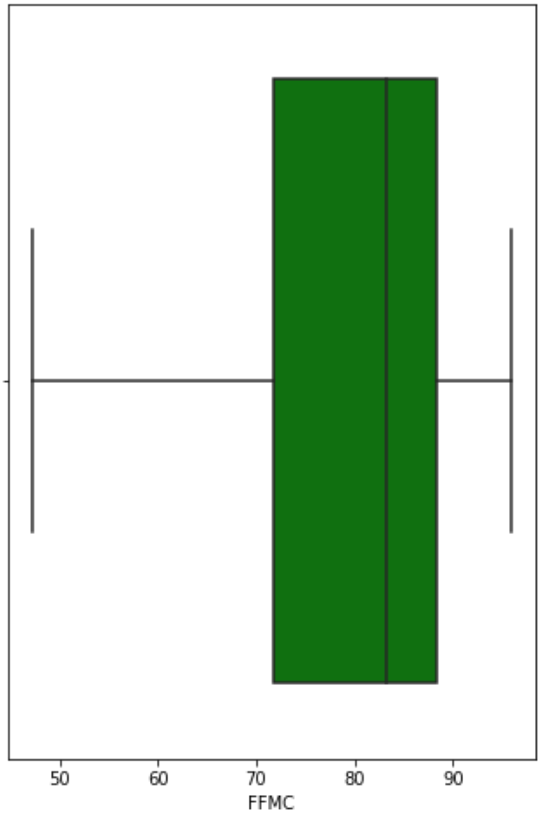

20
30
40
50
60
70
80
90
RH

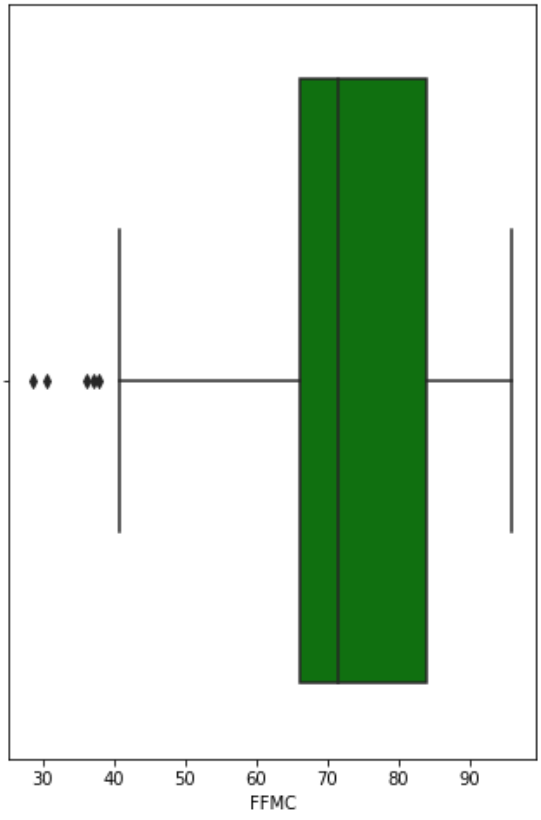

10
12
14
16
18
20
22
Ws

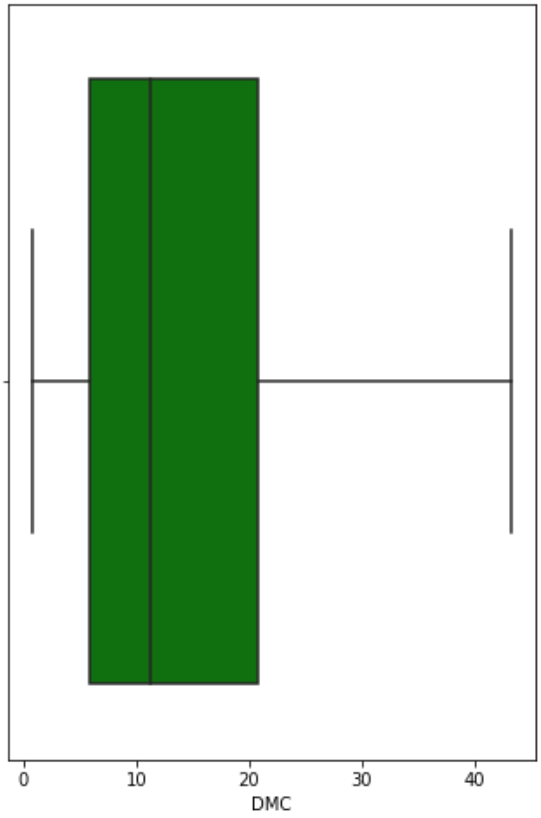

5
10
15
20
25
30
Ws

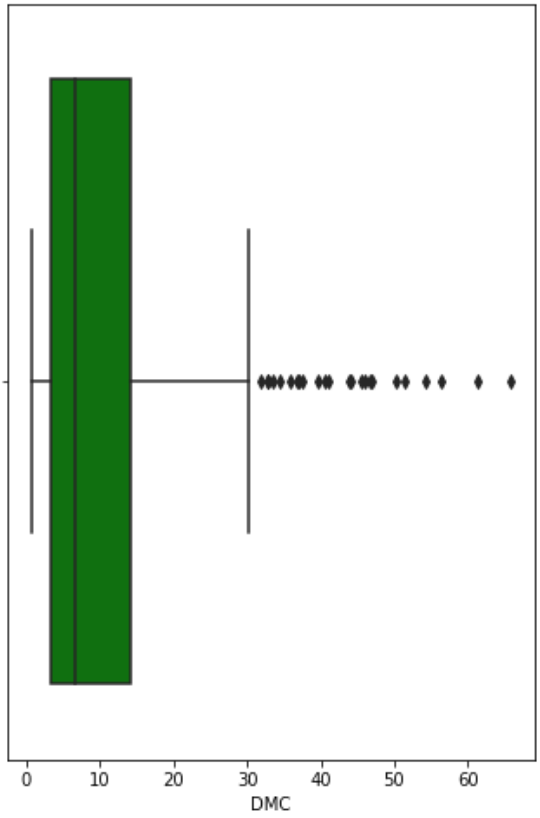

0.0
0.2
0.4
0.6
0.8
1.0
1.2
Rain

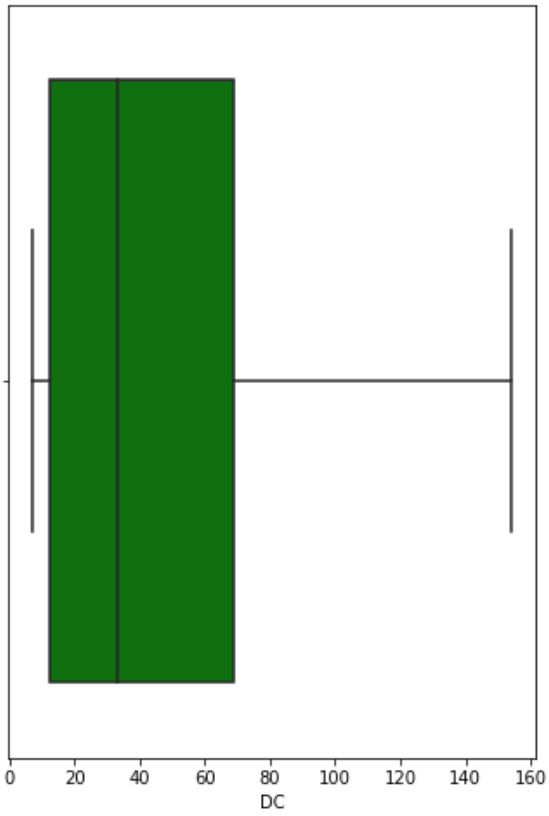

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Rain

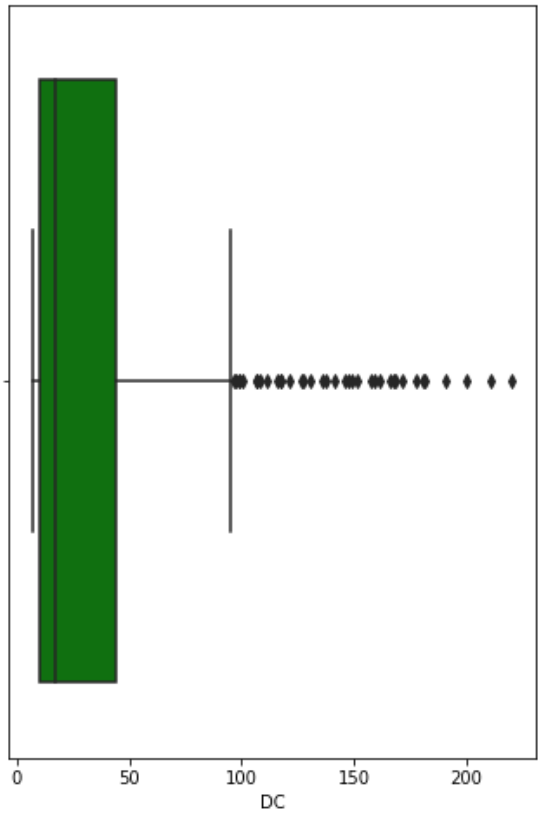

50
60
70
80
90
FFMC

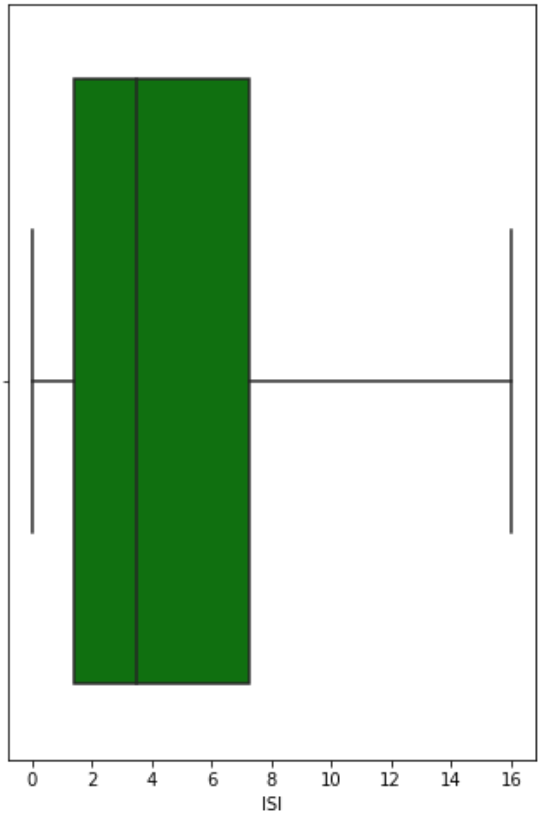

30
40
50
60
70
80
90
FFMC

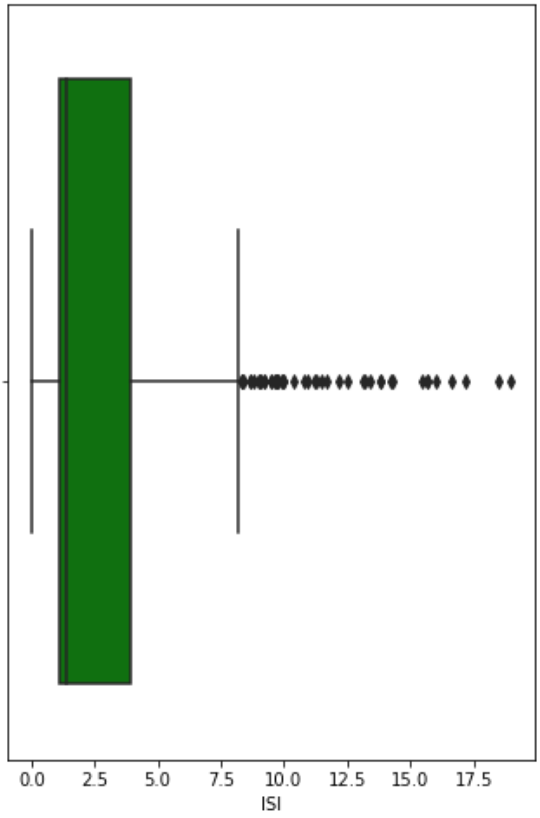

0
10
20
30
40
DMC

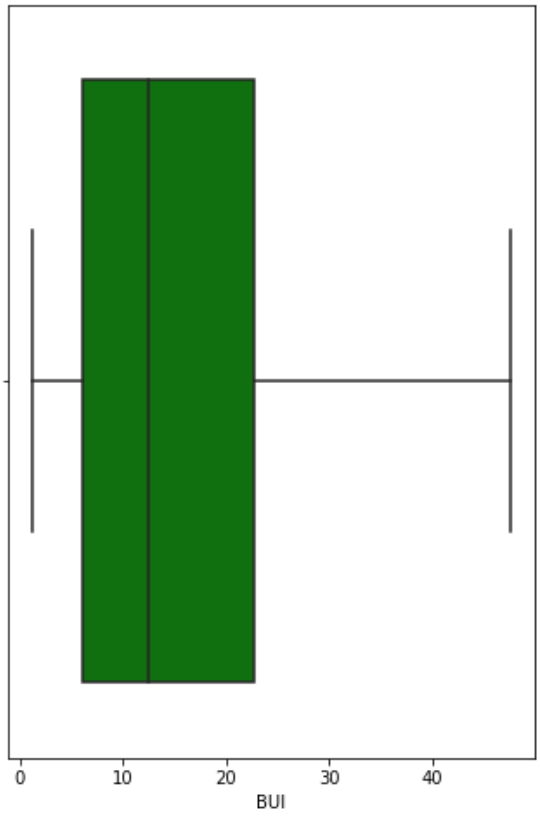

0
10
20
30
40
50
60
DMC

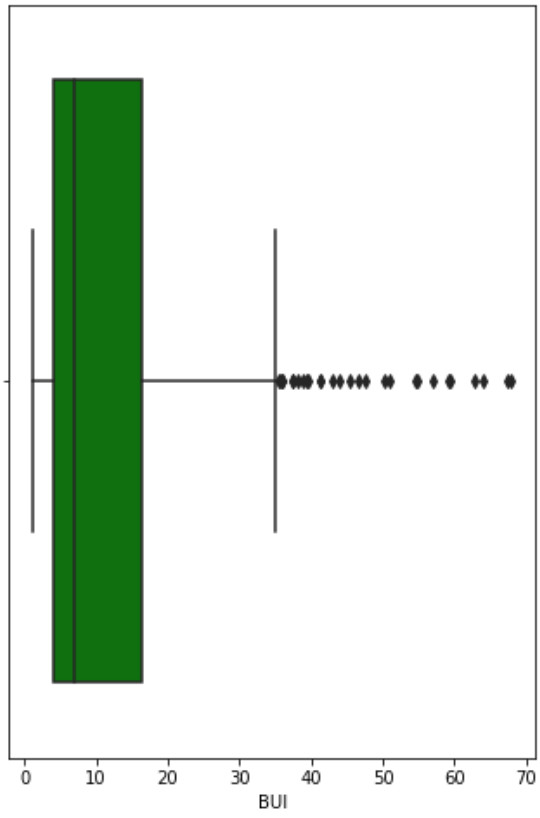

0
20
40
60
80
100
120
140
160
DC

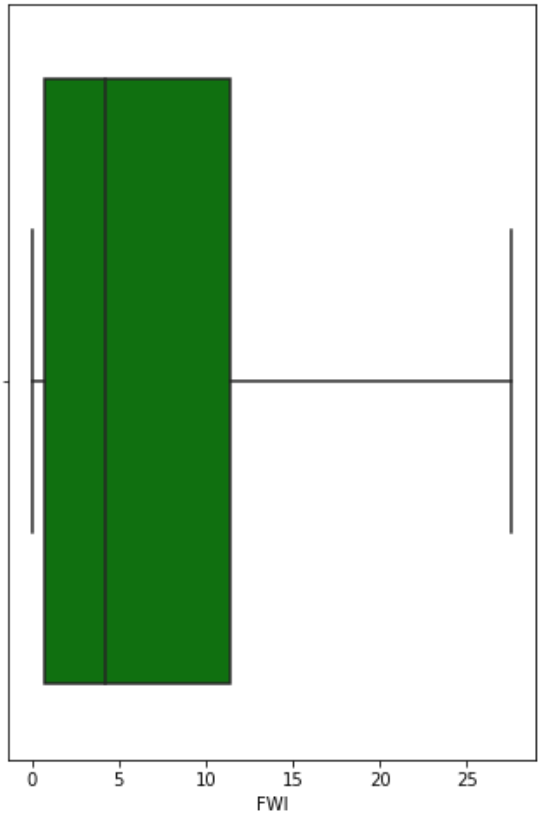

0
50
100
150
200
DC

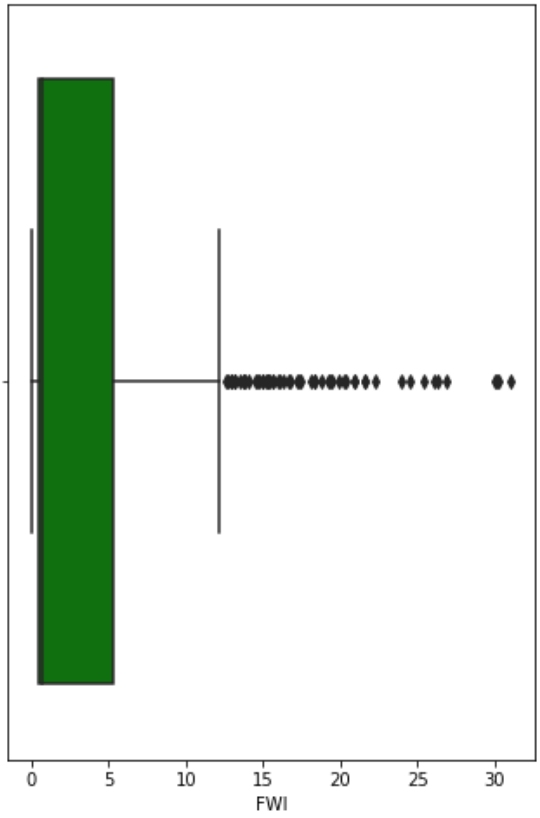

0
2
4
6
8
10
12
14
16
ISI

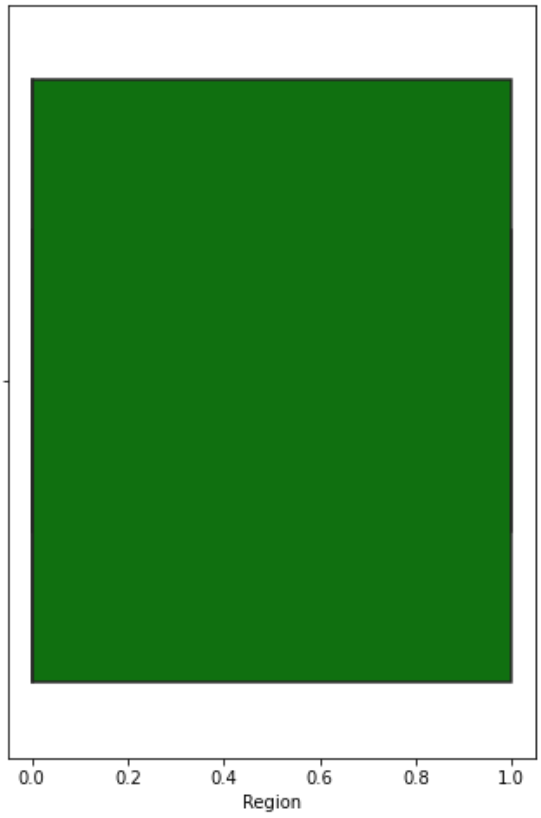

0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
ISI

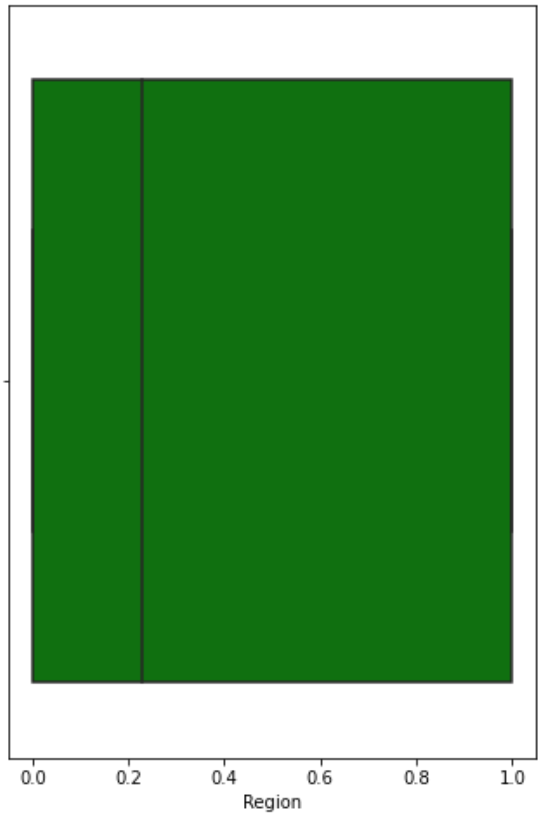

0
10
20
30
40
BUI

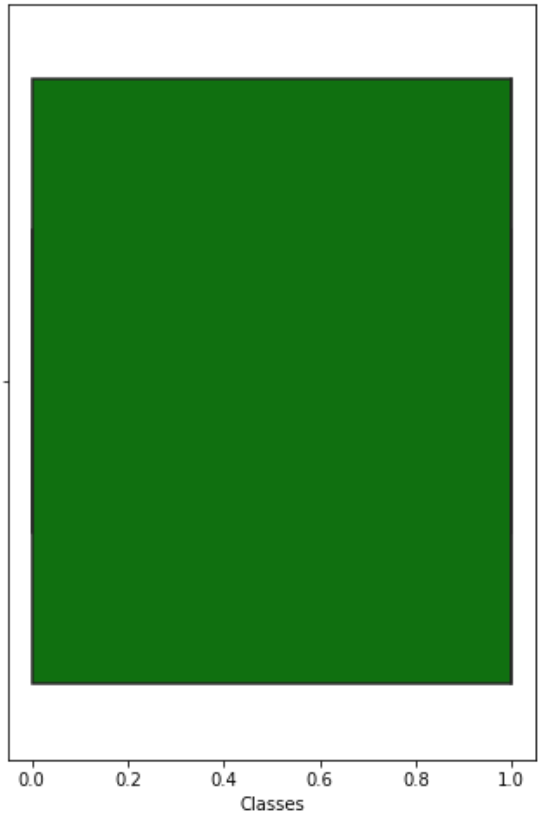

0
10
20
30
40
50
60
70
BUI

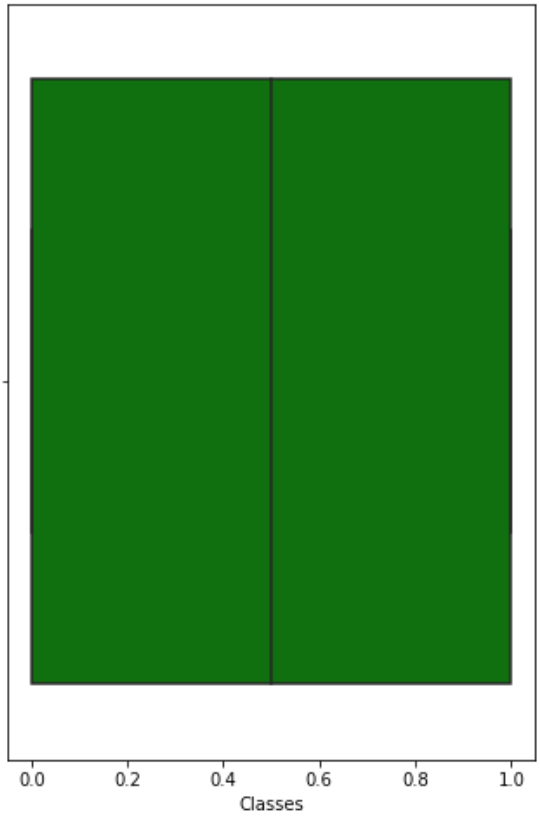

0
5
10
15
20
25
FWI


0
5
10
15
20
25
30
FWI


0.0
0.2
0.4
0.6
0.8
1.0
Region


0.0
0.2
0.4
0.6
0.8
1.0
Region





Test Train Split

In [ ]:
```
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,rand
```

Logistic Regression Model

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

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

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

Standarizing or Feature Scaling

In [ ]:
```
classifier_regressor_bal.fit(X_train1,y_train1)
print(classifier_regressor_bal.best_params_)
print(classifier_regressor_bal.best_score_)
```

```
{'C': 20, 'max_iter': 200, 'penalty': 'l2'}
0.9132692307692307
```

Prediction

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

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

Accuracy

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

```
0.920863309352518
```

Classification Report

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

```
              precision    recall  f1-score   support

           0       1.00      0.87      0.93        87
           1       0.83      1.00      0.90        52

    accuracy                           0.92       139
   macro avg       0.91      0.94      0.92       139
weighted avg       0.93      0.92      0.92       139
```

Performance Metrics

Confusion Metrics

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

Out[ ]:
```
array([[76, 11],
       [ 0, 52]], dtype=int64)
```

In [ ]:
```
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]
print('true_positive:',true_positive)
print('false_positive:',false_positive)
print('true_negative:',true_negative)
print('false_negative:',false_negative)
```

```
true_positive: 76
false_positive: 11
true_negative: 52
false_negative: 0
```

Precision

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

Out[ ]:
```
0.8735632183908046
```

Recall

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

Out[ ]:
```
1.0
```

F1-Score

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

Out[ ]:
```
0.9325153374233129
```

Conclusion

Performance of Logistic Model on Original Dataset

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

```
              precision    recall  f1-score   support

         0.0       0.93      0.96      0.95        28
         1.0       0.98      0.96      0.97        45

    accuracy                           0.96        73
   macro avg       0.95      0.96      0.96        73
weighted avg       0.96      0.96      0.96        73
```

Performance of logistic model on balanced dataset which are created from imbalanced dataset

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

```
              precision    recall  f1-score   support

           0       1.00      0.87      0.93        87
           1       0.83      1.00      0.90        52

    accuracy                           0.92       139
   macro avg       0.91      0.94      0.92       139
weighted avg       0.93      0.92      0.92       139
```

In [ ]:
```
auc = roc_auc_score(y_pred, y_test)
print("Area under curve of original dataset",auc)
```

```
Area under curve of original dataset 0.9599206349206351
```

In [ ]:
```
auc1 = roc_auc_score(y_bal_pred, y_test1)
print("Area under curve of balaced dataset",auc1)
```

```
Area under curve of balaced dataset 0.9367816091954023
```

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 )`
- Also the area under curve `(AUC)` of `original dataset` is `more`, so that model is good