Algerian Forest Fire ML_Linear_Regression Practical Implementation

Submitted By - Ambarish Singh

Life cycle of Machine learning Project

- Understanding the Problem Statement
- Data Collection
- Data Cleaning
- Exploratory data analysis
- Data Pre-Processing
- Model Training
- Choose best model

1) Problem statement.

- The dataset Comprises 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.
- If User can Predict the Temperature Based on Different- Different Features.
- Prediction result can be used for Forest Fire Situation Tackers & Make Correct Preventions to Avoid it in future.

2) Data Collection.

- The Dataset is collected from Website named, UCI Machine Learning Repository.
- The data consists of 15 columns and 244 rows.

In [1]:
```
#comment
#observations
```

Importing Pandas, Numpy, Matplotlib, Seaborn and Warings Library.

2.1 Import Data and Required Packages

In [1]:
```
# Importing required libraries for EDA
# The main aim is to understand data in better way

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings

warnings.filterwarnings("ignore")

%matplotlib inline
```

Loading CSV Data as Pandas DataFrame

In [17]:
```
df = pd.read_csv('Algerian_forest_fires_dataset_UPDATE.csv',header = 1)

```

Show Top 5 Records

In [18]:
```
df.head()
```

Out[18]:

	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

3) DATA Cleaning

Removing Unnecessary Rows From Dataset

In [19]:
```
## Removing Unnecessary Rows From Dataset after Observing the Dataset.

df.drop(index=[122,123], inplace=True)
df.reset_index(inplace=True)
df.drop('index', axis=1, inplace=True)
```

Adding New Feature, named 'Region' in a Dataset

In [22]:
```
## Adding New Feature, named 'Region' in a Dataset

df.loc[:122, 'region'] = 'bejaia'
df.loc[122:, 'region'] = 'Sidi-Bel Abbes'
```

Stripping the names of the columns

In [23]:
```
# Stripping the names of the columns

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

Out[23]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'region'],
      dtype='object')
```

Stripping the Classes Features data

In [139]:
```
# Stripping the Classes Features data

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

Out[139]:
```
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)
```

In [280]:
```
df['Classes'].dtype
```

Out[280]:
```
dtype('float64')
```

In [25]:
```
1 df.head()
```

Out[25]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	reg
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	be
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire	be
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	be
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire	be
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire	be

Changing The DataTypes of the Columns

In [26]:

```
# Changing The DataTypes of the Columns

df['day']=df['day'].astype(int)
df['month']=df['month'].astype(int)
df['year']=df['year'].astype(int)
df['Temperature']=df['Temperature'].astype(int)
df['RH']=df['RH'].astype(int)
df['Ws']=df['Ws'].astype(float)
df['Rain']=df['Rain'].astype(float)
df['FFMC']=df['FFMC'].astype(float)
df['DMC']=df['DMC'].astype(float)
df['ISI']=df['ISI'].astype(float)
df['BUI']=df['BUI'].astype(float)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          244 non-null    int32  
 1   month        244 non-null    int32  
 2   year         244 non-null    int32  
 3   Temperature  244 non-null    int32  
 4   RH           244 non-null    int32  
 5   Ws           244 non-null    float64
 6   Rain         244 non-null    float64
 7   FFMC         244 non-null    float64
 8   DMC          244 non-null    float64
 9   DC           244 non-null    object 
 10  ISI          244 non-null    float64
 11  BUI          244 non-null    float64
 12  FWI          244 non-null    object 
 13  Classes      243 non-null    object 
 14  region       244 non-null    object 
dtypes: float64(6), int32(5), object(4)
memory usage: 24.0+ KB
```

Adding New Feature,named 'Date' by Replacing Unnecessary feature like 'day','month','year'

In [27]:

```
## Adding New Feature,named 'Date' by Replacing Unnecessary feature like 'da

df['date'] = pd.to_datetime(df[['day','month','year']])
df.drop(['day', 'month', 'year'], axis=1, inplace=True)
```

In [28]:
```
## Showing Updated Dataset after Modification Done.
df
```

Out[28]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
0	29	57	18.0	0.0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	bejaia	2012-06-01
1	29	61	13.0	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire	bejaia	2012-06-02
2	26	82	22.0	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	bejaia	2012-06-03
3	25	89	13.0	2.5	28.6	1.3	6.9	0.0	1.7	0	not fire	bejaia	2012-06-04
4	27	77	16.0	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire	bejaia	2012-06-05
...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	30	65	14.0	0.0	85.4	16.0	44.5	4.5	16.9	6.5	fire	Sidi-Bel Abbes	2012-09-26
240	28	87	15.0	4.4	41.1	6.5	8	0.1	6.2	0	not fire	Sidi-Bel Abbes	2012-09-27
241	27	87	29.0	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire	Sidi-Bel Abbes	2012-09-28
242	24	54	18.0	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire	Sidi-Bel Abbes	2012-09-29
243	24	64	15.0	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire	Sidi-Bel Abbes	2012-09-30

244 rows × 13 columns

4) EXPLORING DATA

4.1) Profile of the Data

Shape of the dataset

In [29]:
```
# getting shape and size
df.shape

```

Out[29]:
```
(244, 13)
```

Observation :-

- In this Dataset there are 13 Columns & 244 Rows

Columns of the Dataset

In [30]:
```
1 df.columns
```

Out[30]:
```
Index(['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
       'FWI', 'Classes', 'region', 'date'],
      dtype='object')
```

Check Missing Value in Dataset

In [31]:
```
1 ## Check if Missing Value Present or Not in Dataset.
2 df.isnull().sum()
3
```

Out[31]:
```
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        1
region         0
date           0
dtype: int64
```

Observations:-

- We Got one NUll Value in 'Classes' Feature

In [32]:
```
1 ## Unique Value of Classes feature
2
3 df['Classes'].unique()
```

Out[32]:
```
array(['not fire', 'fire', nan], dtype=object)
```

Handling Categorical Feature Classes

In [33]:
```
## Handling Categorical Feature Classes

df['Classes']=df['Classes'].map({'not fire':0,'fire':1})
df.head()
```

Out[33]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
0	29	57	18.0	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0.0	bejaia	2012-06-01
1	29	61	13.0	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0.0	bejaia	2012-06-02
2	26	82	22.0	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0.0	bejaia	2012-06-03
3	25	89	13.0	2.5	28.6	1.3	6.9	0.0	1.7	0	0.0	bejaia	2012-06-04
4	27	77	16.0	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0.0	bejaia	2012-06-05

Focus on Replacing Null Value

In [34]:
```
# Focus on Replacing Null Value
# The best Way of Replacing Null Value by using mode

df['Classes'].mode() [0]
```

Out[34]:
```
1.0
```

In [35]:
```
df['Classes']=df['Classes'].fillna(df['Classes'].mode()[0])
```

In [36]:
```
df.isnull().sum()
```

Out[36]:
```
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        0
region         0
date           0
dtype: int64
```

Observations

- Now We have Zero Null Value in dataset

In [37]:
```
df['Classes'].unique()
```

Out[37]:
```
array([0., 1.])
```

Check Datatypes in the dataset

In [38]:
```
# Check Null & getting feature datatypes
df.info()

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   Temperature  244 non-null    int32         
 1   RH           244 non-null    int32         
 2   Ws           244 non-null    float64       
 3   Rain         244 non-null    float64       
 4   FFMC         244 non-null    float64       
 5   DMC          244 non-null    float64       
 6   DC           244 non-null    object        
 7   ISI          244 non-null    float64       
 8   BUI          244 non-null    float64       
 9   FWI          244 non-null    object        
 10  Classes      244 non-null    float64       
 11  region       244 non-null    object        
 12  date         244 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(7), int32(2), object(3)
memory usage: 23.0+ KB
```

Observations

- There is total 244 rows and 13 columns.
- There are No Null Value in Dataset
- There is total 4 data types float64, int64, object and datetime64.
- Dtypes Included float64 = 7 Columns, int64 = 2 Columns, object = 3 Columns and datetime64 = 1
- Total Memory Usage is 23.0+ KB

Checking the usage of the memory by the dataset

In [39]:
```
## Checking the usage of the memory by the dataset

df.memory_usage()
```

Out[39]:
```
Index           128
Temperature     976
RH              976
Ws             1952
Rain           1952
FFMC           1952
DMC            1952
DC             1952
ISI            1952
BUI            1952
FWI            1952
Classes        1952
region         1952
date           1952
dtype: int64
```

4.1.1 Numerical and Categorical Columns

Numerical Dataset

In [40]:
```
# 1. Getting Numerical features from dataset
# 2. Creating Numerical dataframe
numerical_features = [feature for feature in df.columns if df[feature].dtype

# Print Numerical Features
print('We have {} numerical features : {}'.format(len(numerical_features), n

```

```
We have 10 numerical features : ['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DM
C', 'ISI', 'BUI', 'Classes', 'date']
```

Categorical Dataset

In [41]:
```
# 1. Getting Categorical features from dataset
# 2. Creating Categorical dataframe
categorical_features = [feature for feature in df.columns if df[feature].dty

# print columns
print('\n We have {} categorical features : {}'.format(len(categorical_featu
```

```
 We have 3 categorical features : ['DC', 'FWI', 'region']
```

4.1.2 Feature Information

In [42]:
```
1 df.head(2)
```

Out[42]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
0	29	57	18.0	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0.0	bejaia	2012-06-01
1	29	61	13.0	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0.0	bejaia	2012-06-02

Weather data observations:-

- Temperature : temperature noon (temperature max) in Celsius degrees: 22 to 42
- RH : Relative Humidity in %: 21 to 90
- Ws :Wind speed in km/h: 6 to 29
- Rain: total day in mm: 0 to 16.8

FWI Components

- (FFMC) Fine Fuel Moisture Code index from the FWI system: 28.6 to 92.5
- (DMC) Duff Moisture Code index from the FWI system: 1.1 to 65.9
- (DC) Drought Code index from the FWI system: 7 to 220.4
- (ISI) Initial Spread Index from the FWI system: 0 to 18.5
- (BUI) Buildup Index from the FWI system: 1.1 to 68
- (FWI) Fire Weather Index: 0 to 31.1
- Classes: two classes, namely Fire and not Fire.
- Region: Two Regions, namely Bejaia Region indicated with 0 and Sidi Bel-Abbes Region indicated with 1.

DATE Observations (DD/MM/YYYY) :-

- Date :- Date Displayed in (DD/MM/YYYY) format in dataset

Univariate Analysis

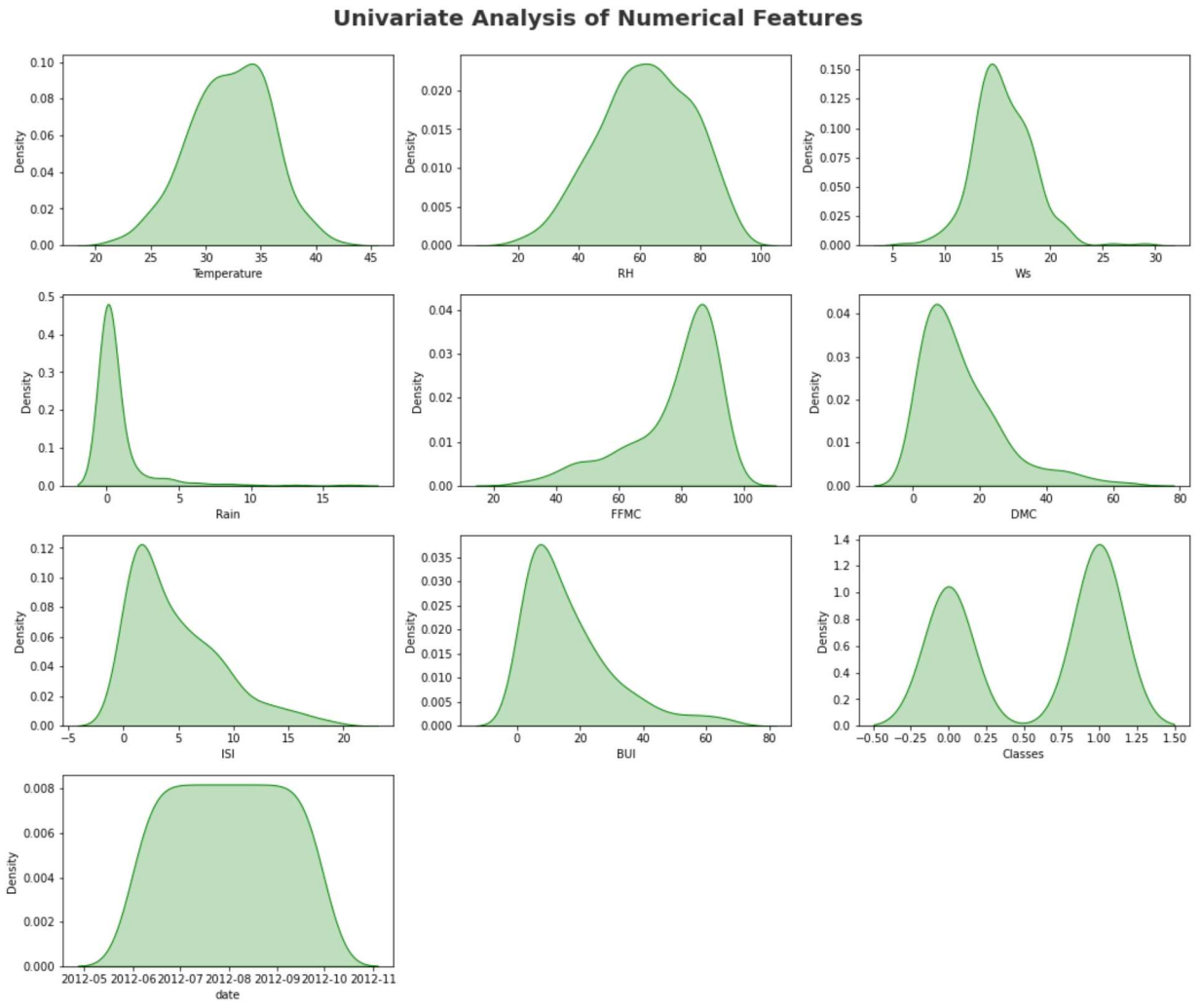
- The term univariate analysis refers to the analysis of one variable prefix “uni” means “one.” The purpose of univariate analysis is to understand the distribution of values for a single variable.

In [43]:
```
1 df.var()
```

Out[43]:
```
Temperature     13.204817
RH             221.539415
Ws               7.897102
Rain             3.997623
FFMC           205.565939
DMC            152.968382
ISI             17.433281
BUI            201.777024
Classes          0.246711
dtype: float64
```

Numerical Features Analysis

In [44]:
```
1 plt.figure(figsize=(15, 15))
2 plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontw
3
4 for i in range(0, len(numerical_features)):
5     plt.subplot(5, 3, i+1)
6     sns.kdeplot(x=df[numerical_features[i]],shade=True, color='g')
7     plt.xlabel(numerical_features[i])
8     plt.tight_layout()
```



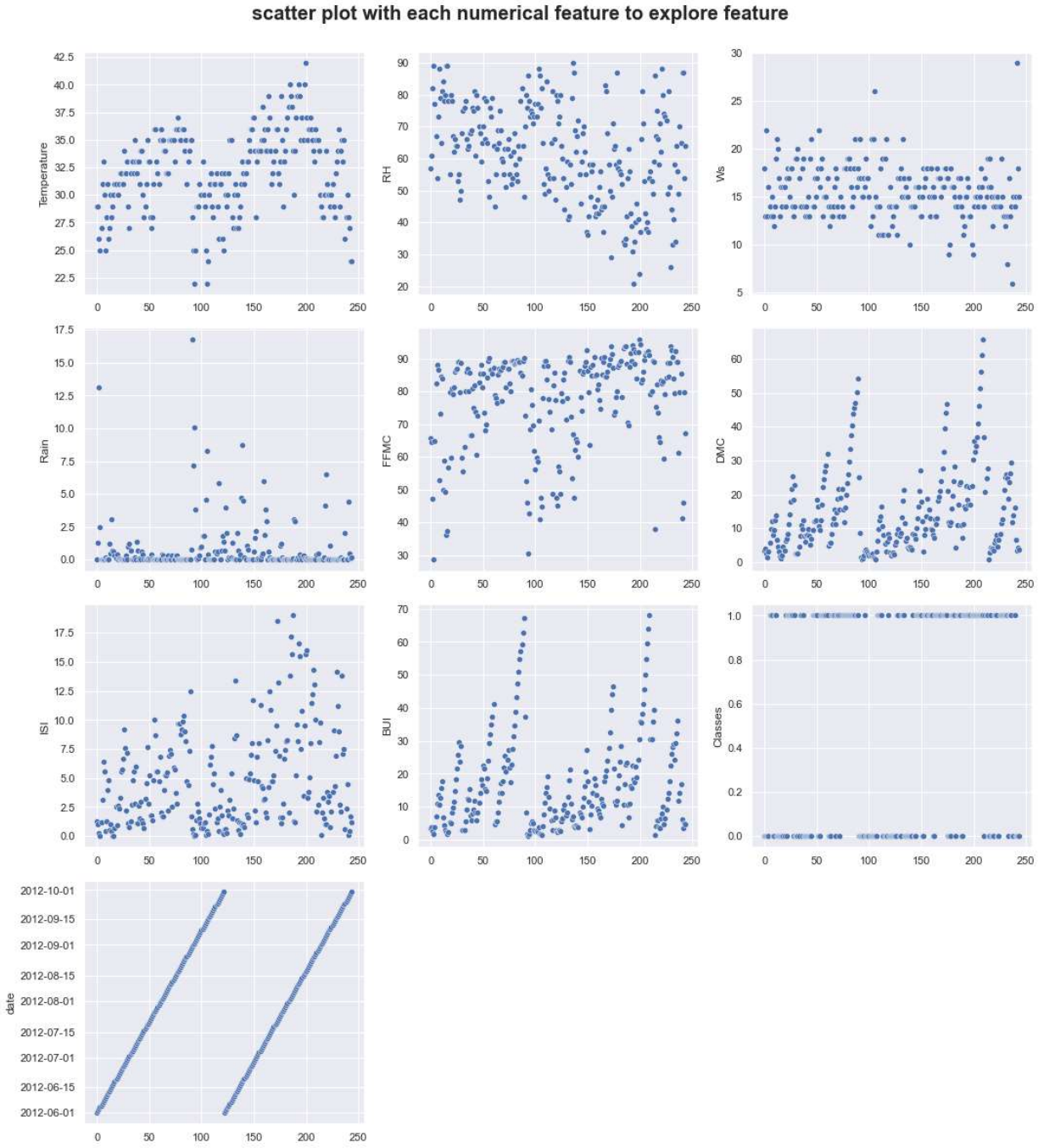
Observations

- Rain,ISI,BUI,DMC are right skewed and postively skewed.
- FFMC is a Left skewed and Negetively skewed.
- Outliers in Rain, ISI, BUI, DMC and FFMC

Scatter plot to see the trends in each numerical column

In [48]:

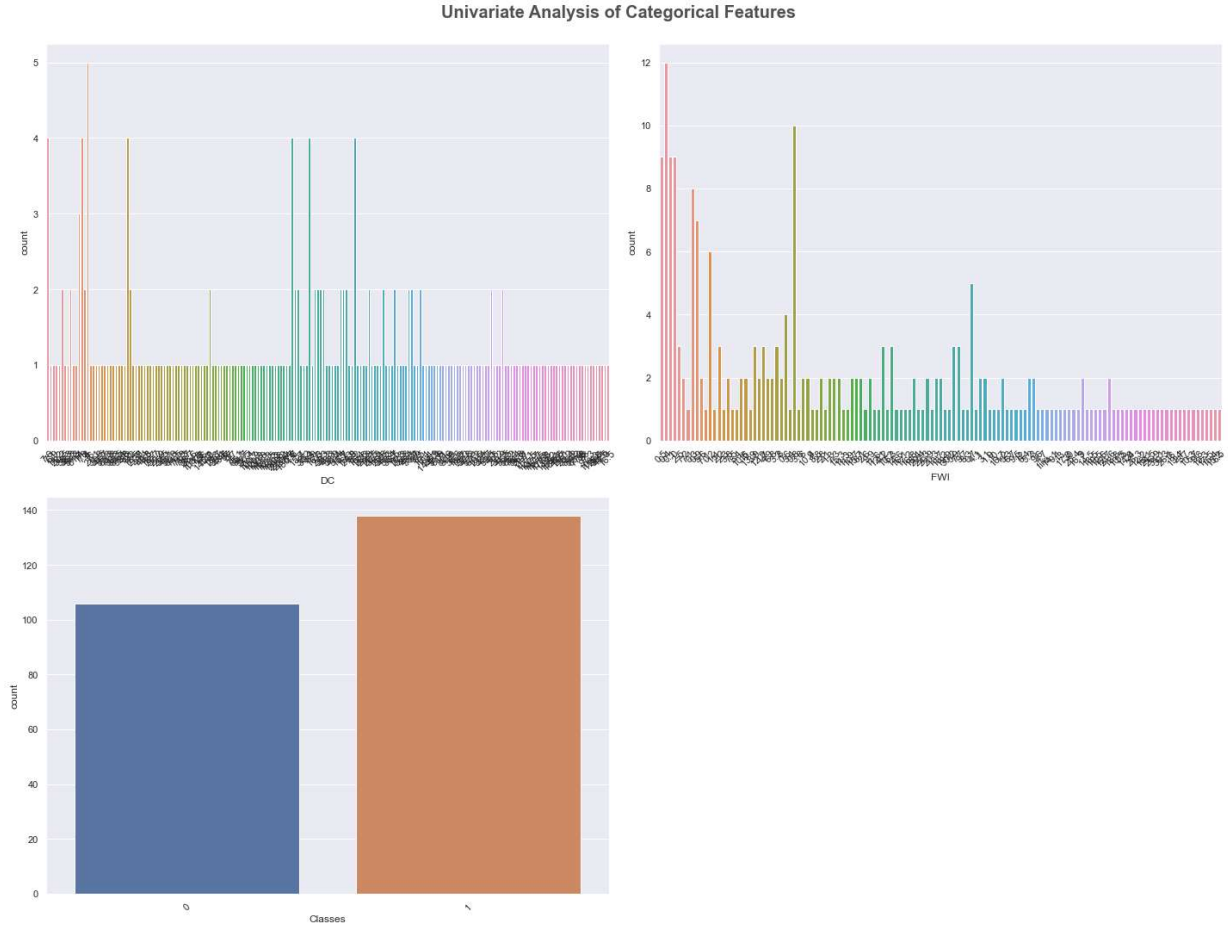
```
# scatter plot to see the trends in each numerical column

plt.figure(figsize=(15, 20))
plt.suptitle('scatter plot with each numerical feature to explore feature',

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



Categorical Features Analysis

In [46]:

```
# categorical columns Analysis

plt.figure(figsize=(20, 15))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fon
cat1 = ['DC', 'FWI', 'Classes']
for i in range(0, len(cat1)):
    plt.subplot(2, 2, i+1)
    sns.countplot(x=df[cat1[i]])
    plt.xlabel(cat1[i])
    plt.xticks(rotation=45)
    plt.tight_layout()

```



observation -

- Extreme value of Temperature is above 40
- Most of the time RH is above 30
- WS values lie between 10 to 20

Bivariate analysis and multivariate analysis

In [49]:
```
# stripplot (categorical vs numerical)
# scatterplot / pairplot (numerical vs numerical) (check correlation)
# boxplot (outlies)
# heatmap (correlation)
# lineplot (trend in numerical feature with time)
```

Multicolleniarity in numerical features

In [50]:
```
df.corr()
```

Out[50]:

	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	B
Temperature	1.000000	-0.654443	-0.278132	-0.326786	0.677491	0.483105	0.607551	0.4555(
RH	-0.654443	1.000000	0.236084	0.222968	-0.645658	-0.405133	-0.690637	-0.3485
Ws	-0.278132	0.236084	1.000000	0.170169	-0.163255	-0.001246	0.015248	0.0297
Rain	-0.326786	0.222968	0.170169	1.000000	-0.544045	-0.288548	-0.347105	-0.2991
FFMC	0.677491	-0.645658	-0.163255	-0.544045	1.000000	0.602391	0.739730	0.5896
DMC	0.483105	-0.405133	-0.001246	-0.288548	0.602391	1.000000	0.674499	0.9820
ISI	0.607551	-0.690637	0.015248	-0.347105	0.739730	0.674499	1.000000	0.6358
BUI	0.455504	-0.348587	0.029756	-0.299171	0.589652	0.982073	0.635891	1.0000(
Classes	0.518119	-0.435023	-0.066529	-0.379449	0.770114	0.584188	0.735511	0.5838

In [51]:

```
## Plotting Heatmap

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

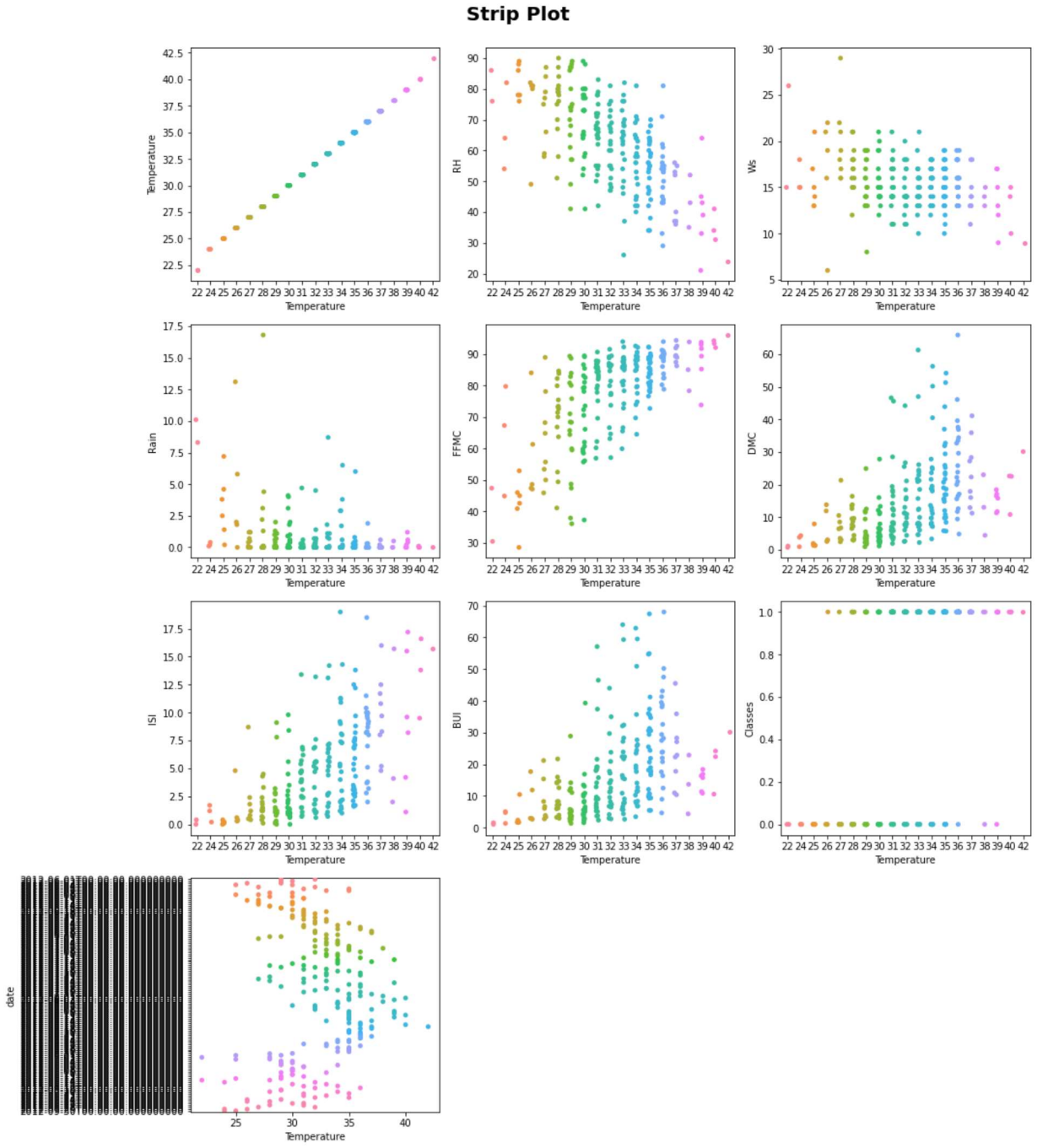
observation -

- Highly +ve correlated features are DMC and BUI
- Highly -ve correlated features are RH and Temp, RH and FFMC, RH and ISI

strip plot to see the relationship between numerical features and target

In [45]:

```
# strip plot to see the relationship between numerical features and target
## Targeted Feature is "Temperature"


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

for i in range(0, len(numerical_features)):
    plt.subplot(5, 3, i+1)
    sns.stripplot(y=numerical_features[i], x='Temperature', data=df)
    plt.tight_layout()
```



observation -

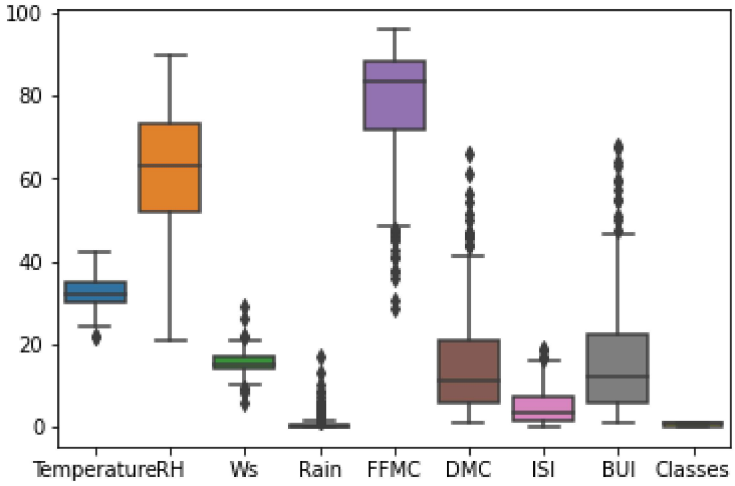
Note :- Here Targeted Feature is "Temperature"

- places with higher 'RH' has Lower 'Temperature'
- places with lower 'WS' has higher 'Temperature'
- places with FFMC > 80 has higher 'Temperature'
- places wiht ISI > 15.0 has higher 'Temperature'

Boxplot to find Outliers in the features

In [46]:
```
## Boxplot to find Outliers in the features
sns.boxplot(data = df,orient="v")

```

Out[46]: <AxesSubplot:>



Observation:-

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

4.2) Statistical Analysis

In [47]:
```
# Display summary statistics for a dataframe
df.describe()

```

Out[47]:

	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	
count	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.0
mean	32.172131	61.938525	15.504098	0.760656	77.887705	14.673361	4.774180	16.6
std	3.633843	14.884200	2.810178	1.999406	14.337571	12.368039	4.175318	14.2
min	22.000000	21.000000	6.000000	0.000000	28.600000	0.700000	0.000000	1.1
25%	30.000000	52.000000	14.000000	0.000000	72.075000	5.800000	1.400000	6.0
50%	32.000000	63.000000	15.000000	0.000000	83.500000	11.300000	3.500000	12.2
75%	35.000000	73.250000	17.000000	0.500000	88.300000	20.750000	7.300000	22.5
max	42.000000	90.000000	29.000000	16.800000	96.000000	65.900000	19.000000	68.0

Observation

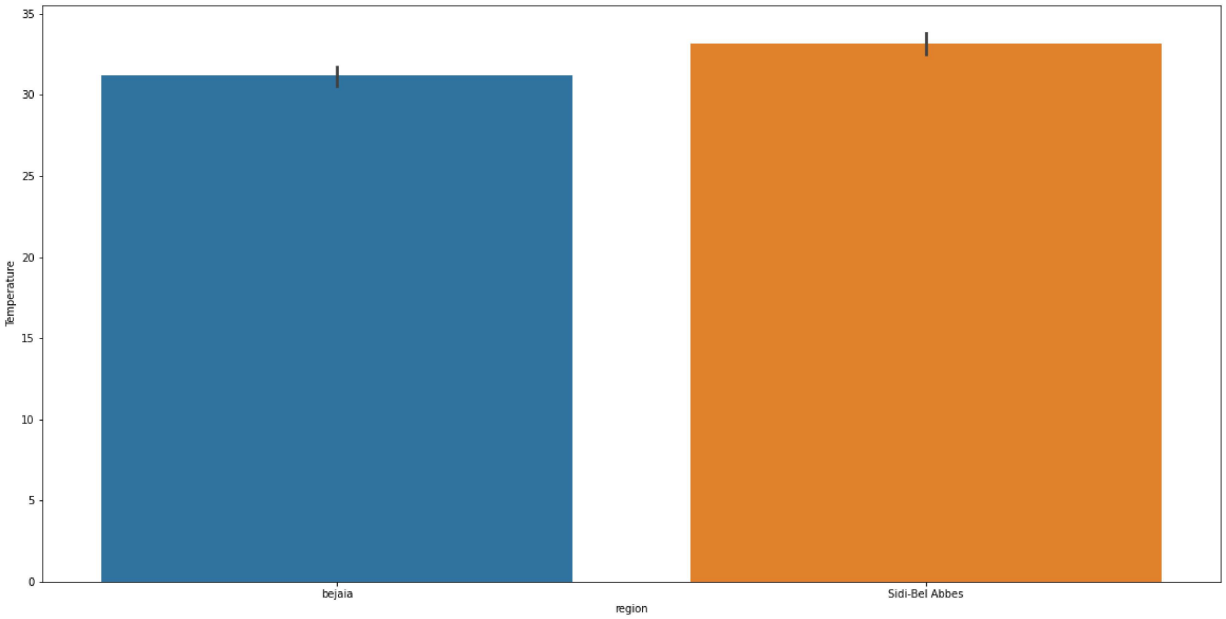
- df.describe() return all Statistics Summary of Numeric Columns.
- Its Return function like:- count(), mean(), std(), min(), 25%(), 50%(), 75%(), max().

4.3) Graphical Analysis

Which area has most of the time High Temperatures ?

In [48]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

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

Out[48]: <AxesSubplot:xlabel='region', ylabel='Temperature'>



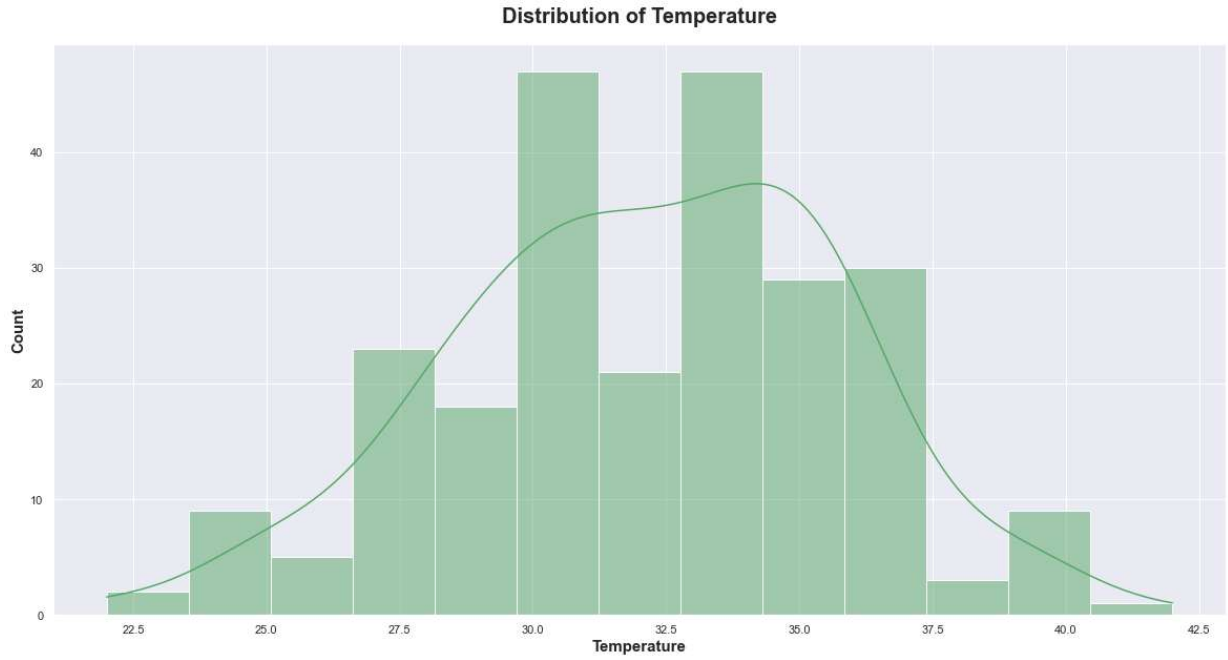
Observation

- Sidi-Bel-Abbes Region has Most of the Time has Higher Temperature.

Temperature Range which is in most of the places ?

In [67]:

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



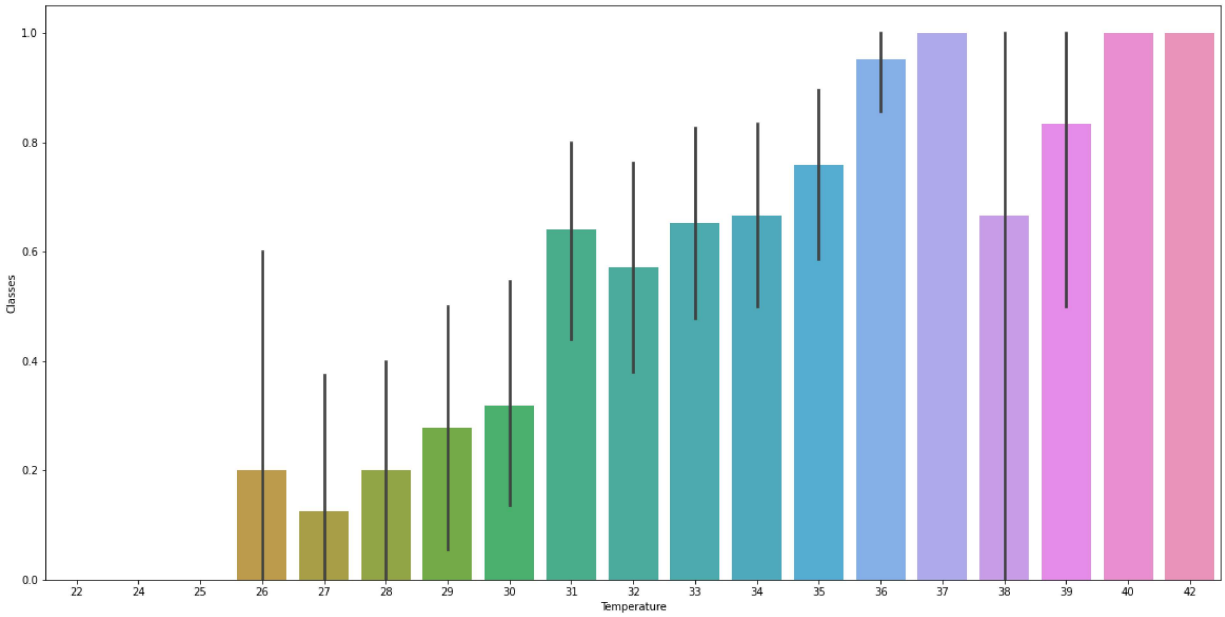
Observation:-

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

Highest Temperature attained

In [49]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

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

Out[49]: <AxesSubplot:xlabel='Temperature', ylabel='Classes'>



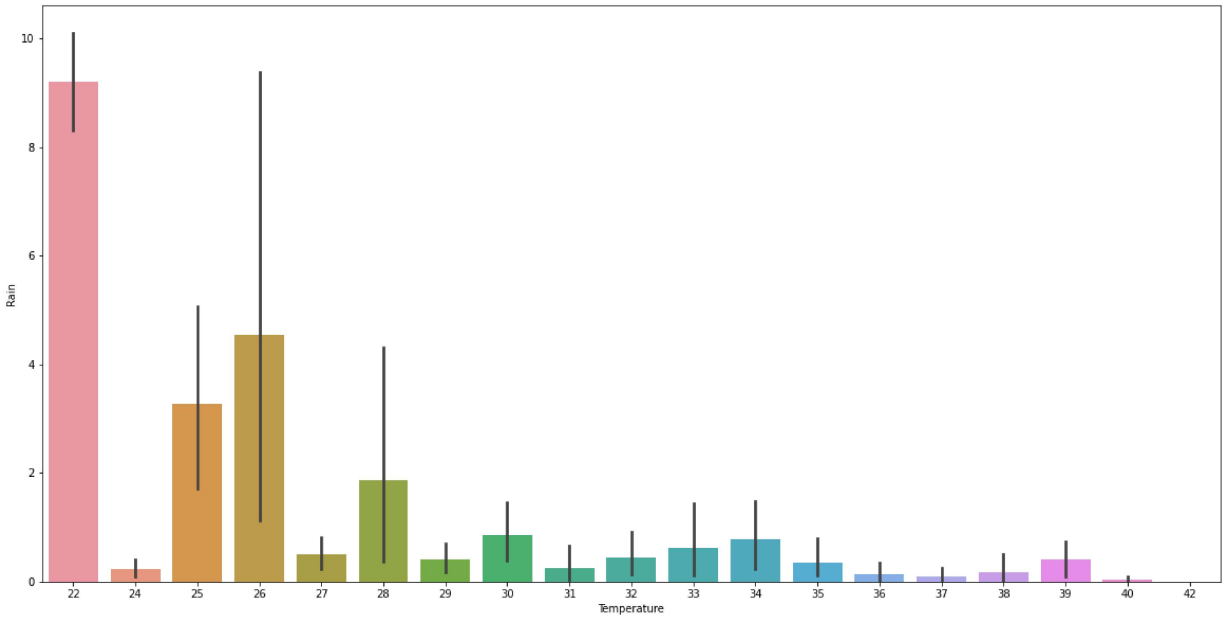
Observation:-

- Highest temperature is 42,40,37

What are most time rain happens in Respect with Temperature

In [50]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

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

Out[50]: <AxesSubplot:xlabel='Temperature', ylabel='Rain'>



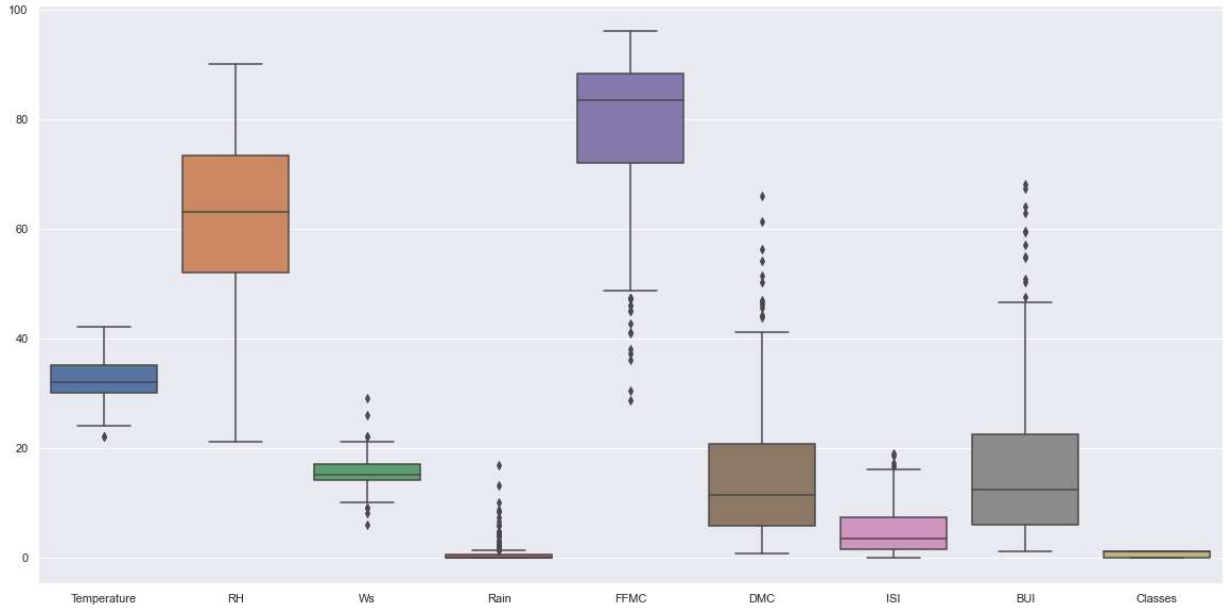
Observation

- Where Rain is Higher , the Temperature is low.
- Where Rain is Low, the Temperature is high.

Boxplot to find Outliers in the features

In [73]:
```
## Boxplot to find Outliers in the features
sns.boxplot(data = df,orient="v")

```

Out[73]: <AxesSubplot:>



Observation:-

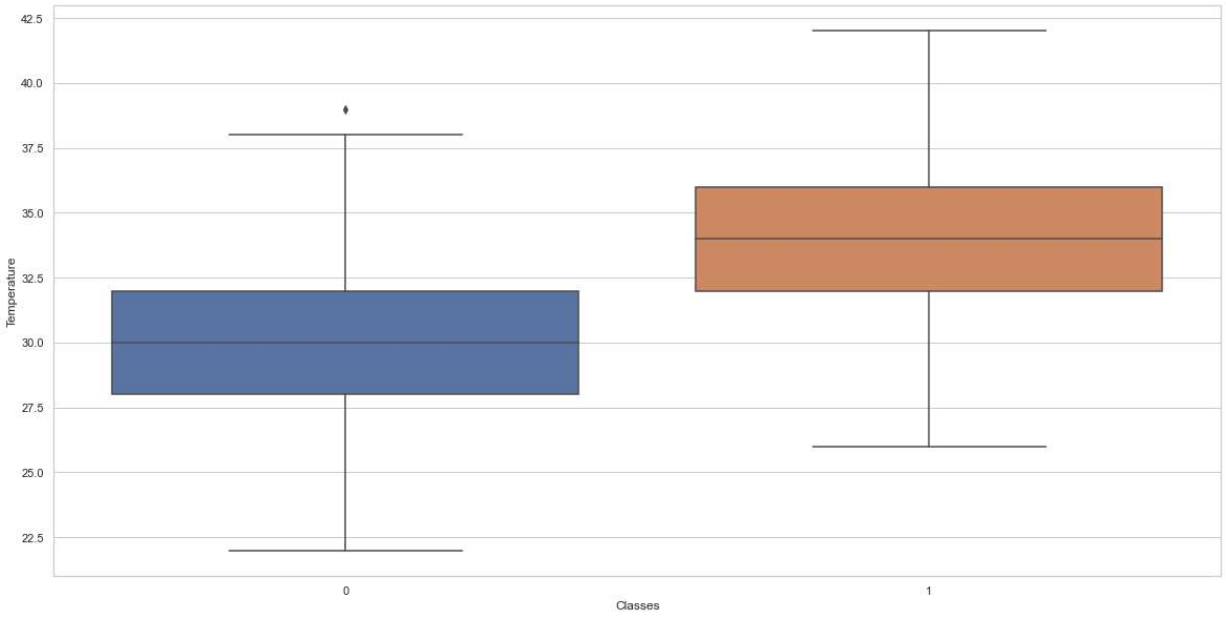
- Ws, Rain, FFMC, DMC BUI has many outliers

Boxplot of Class Vs Temperature

In [74]:

```
# Python program to illustrate
# boxplot using inbuilt data-set
# given in seaborn

# importing the required module
import seaborn

# use to set style of background of plot
seaborn.set(style="whitegrid")

# loading data-set

seaborn.boxplot(x ='Classes', y ='Temperature', data = df)

```

Out[74]: <AxesSubplot:xlabel='Classes', ylabel='Temperature'>



Observations:-

Note :- Here, Classes contain 0 = Fire, 1 = Not Fire.

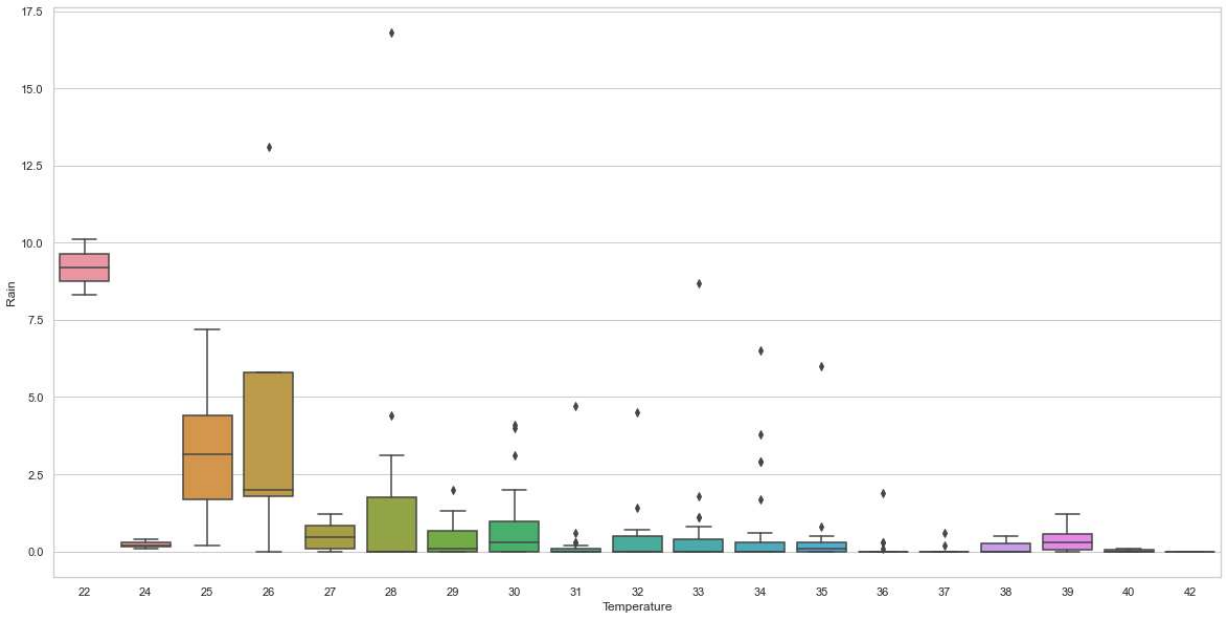
- One day at lower temperature fires occur

Boxplot of Temperature Vs Rain

In [51]:

```
# Python program to illustrate
# boxplot using inbuilt data-set
# given in seaborn

# importing the required module
import seaborn

# use to set style of background of plot
seaborn.set(style="whitegrid")

# loading data-set

seaborn.boxplot(x ='Temperature', y ='Rain', data = df)

```

Out[51]: <AxesSubplot:xlabel='Temperature', ylabel='Rain'>

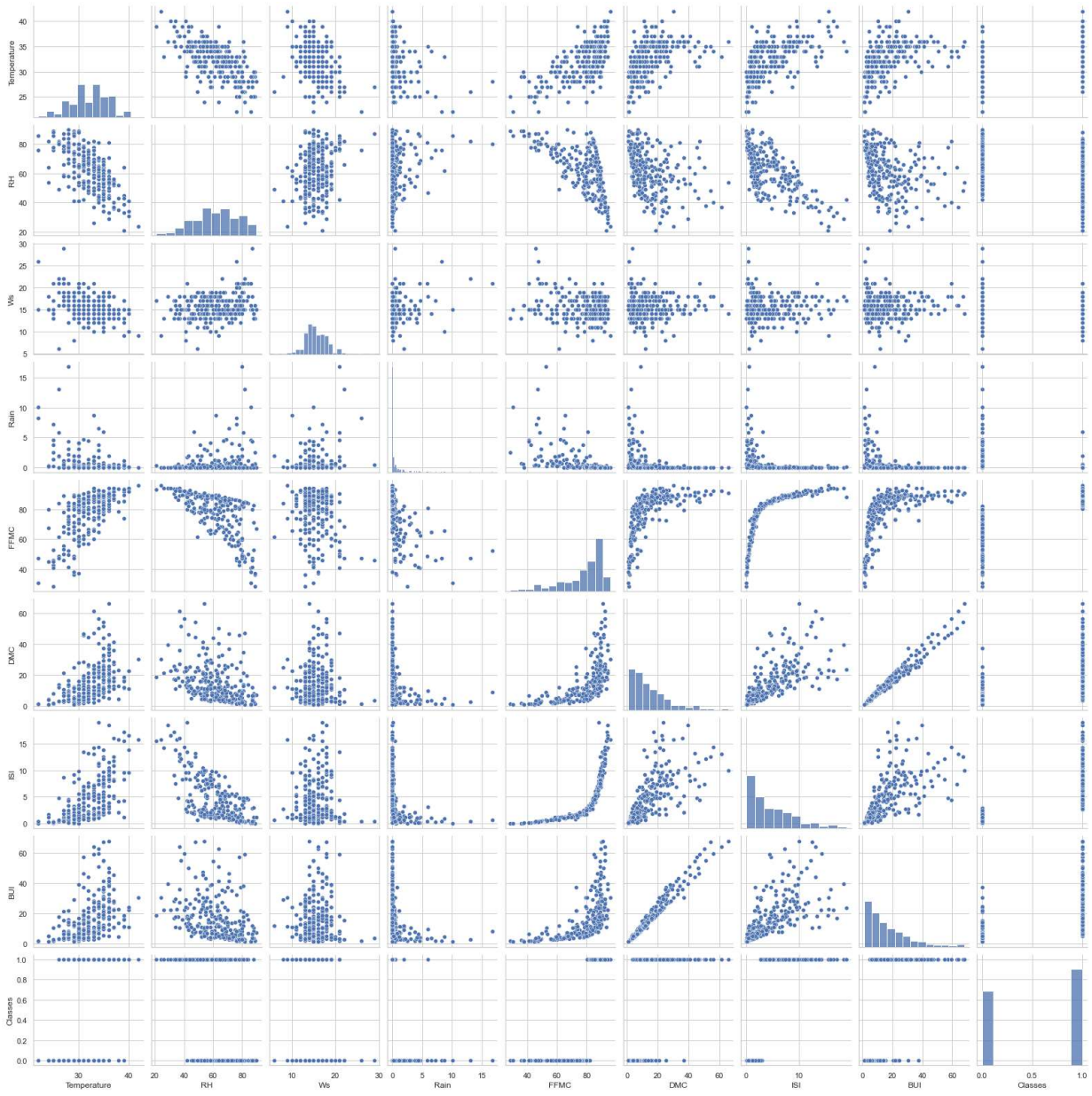


Observation:-

- Where Rain is Higher , the Temperature is low.
- Where Rain is Low, the Temperature is high.

In [52]:
```
1 import seaborn as sns
2 sns.pairplot(df)
```

Out[52]: <seaborn.axisgrid.PairGrid at 0x20e1802f5b0>

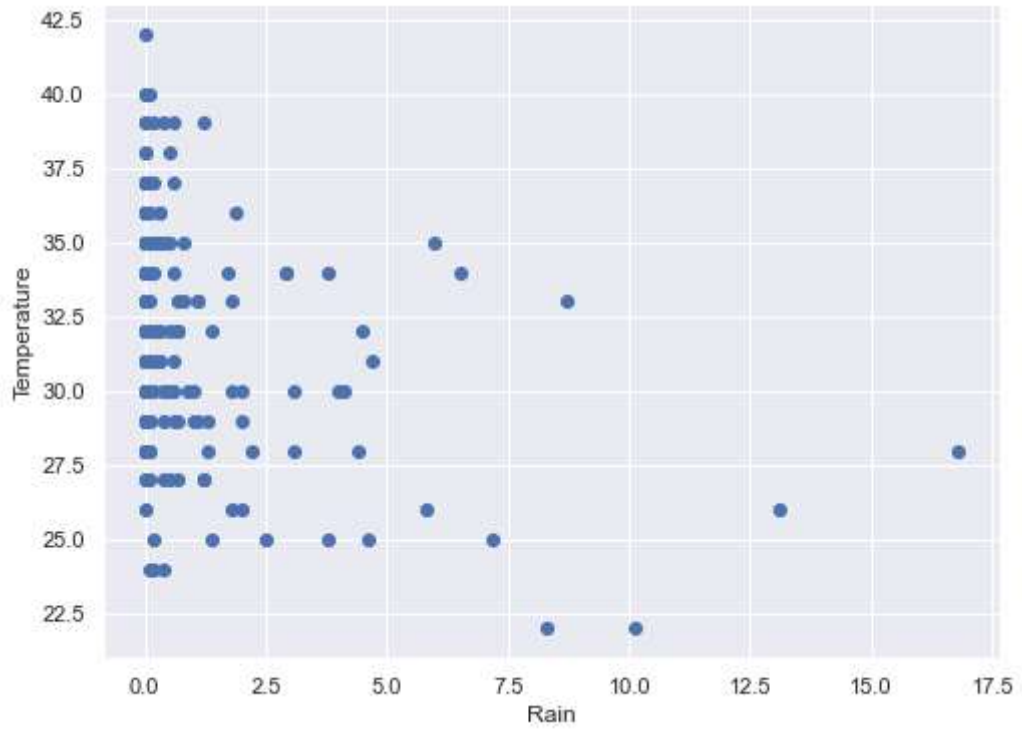


In [54]:
```
1 sns.set(rc = {'figure.figsize' : (10,8)})
2 sns.heatmap(df.corr(), annot = True)
3
```

Out[54]: <AxesSubplot:>

In [59]:
```
1 plt.scatter(df['Rain'],df['Temperature'])
2 plt.xlabel("Rain")
3 plt.ylabel("Temperature")
```

Out[59]: Text(0, 0.5, 'Temperature')

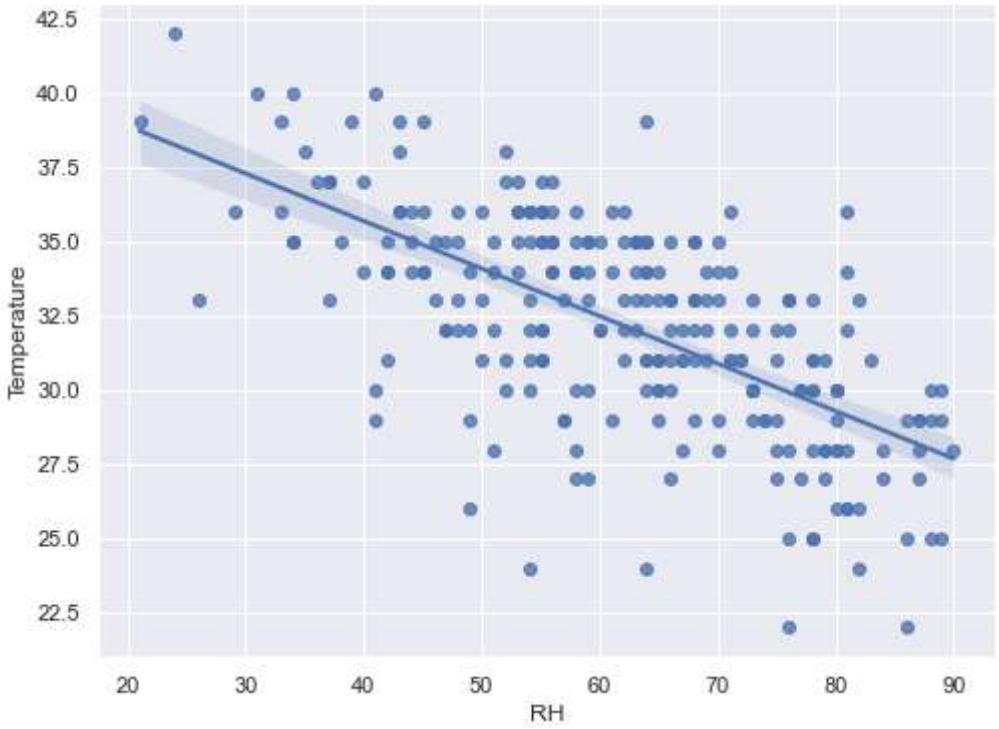


Observation:-

- Where Rain is Higher , the Temperature is low.
- Where Rain is Low, the Temperature is high.

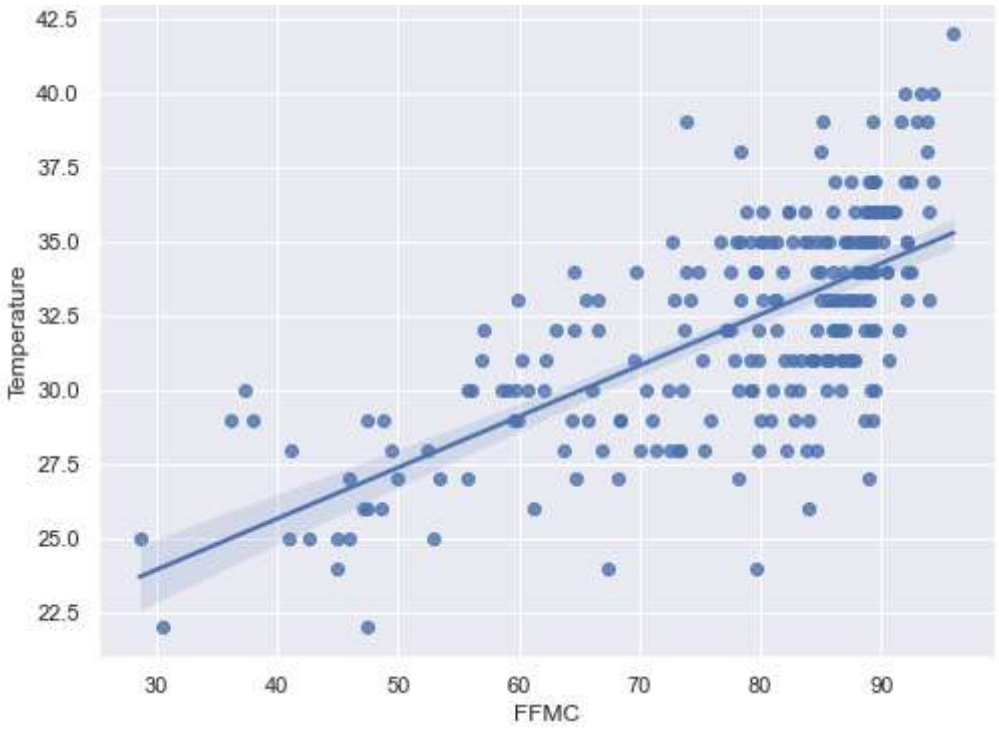
In [58]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.regplot(x = "RH", y = "Temperature", data = df)
```

Out[58]: <AxesSubplot:xlabel='RH', ylabel='Temperature'>



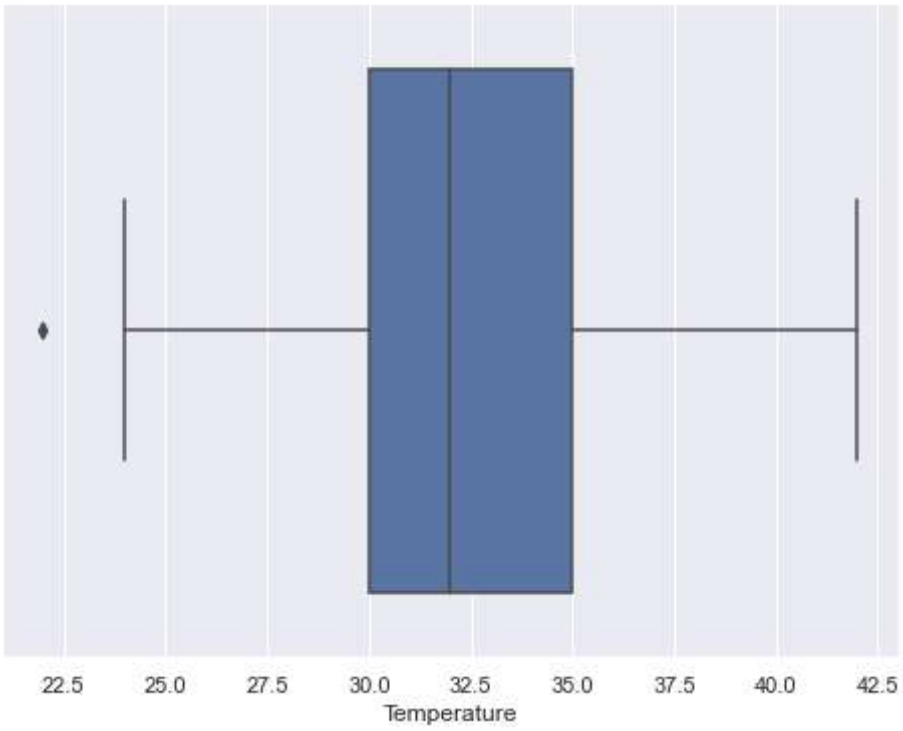
In [62]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.regplot(x="FFMC",y="Temperature",data=df)
```

Out[62]: <AxesSubplot:xlabel='FFMC', ylabel='Temperature'>



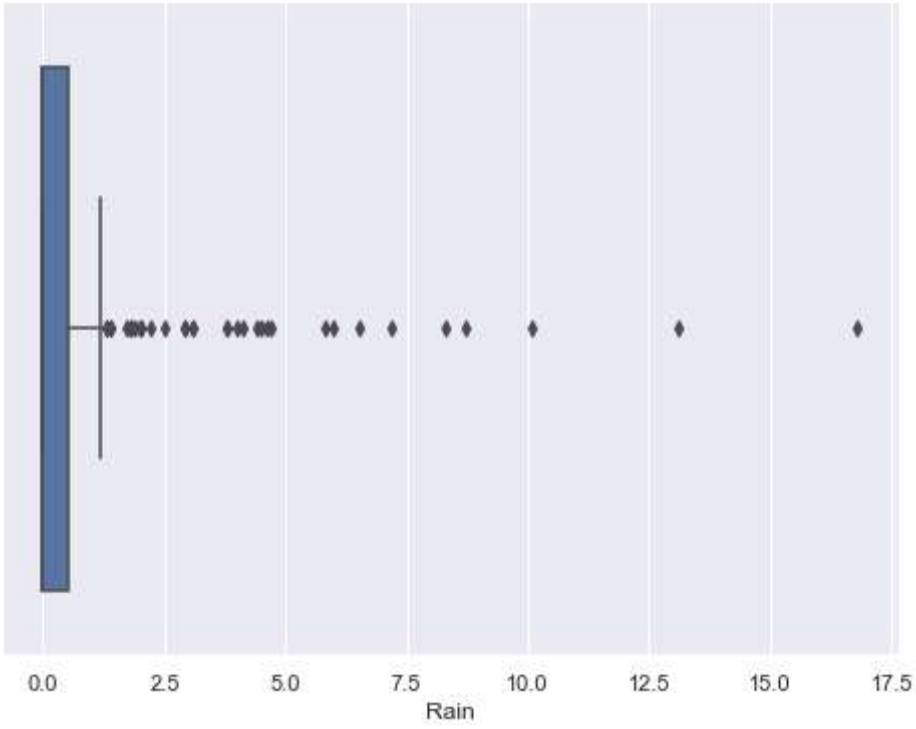
In [63]:
```
1 sns.boxplot(df['Temperature'])
```

Out[63]: <AxesSubplot:xlabel='Temperature'>



In [65]:
```
1 sns.boxplot(df['Rain'])
```

Out[65]: <AxesSubplot:xlabel='Rain'>



In [242]:
```
1 df.head(3)
```

Out[242]:

	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	BUI	Classes	region	date
0	29	57	18.0	0.0	65.7	3.4	1.3	3.4	0.0	bejaia	2012-06-01
1	29	61	13.0	1.3	64.4	4.1	1.0	3.9	0.0	bejaia	2012-06-02
2	26	82	22.0	13.1	47.1	2.5	0.3	2.7	0.0	bejaia	2012-06-03

In [243]:
```
## Independent and Dependent Feature

x = df.iloc[:, 1:-2]
y = df.iloc[:,0]
  ## y = Tagetted feature "Temperature"
```

In [244]:
```
x
```

Out[244]:

	RH	Ws	Rain	FFMC	DMC	ISI	BUI	Classes
0	57	18.0	0.0	65.7	3.4	1.3	3.4	0.0
1	61	13.0	1.3	64.4	4.1	1.0	3.9	0.0
2	82	22.0	13.1	47.1	2.5	0.3	2.7	0.0
3	89	13.0	2.5	28.6	1.3	0.0	1.7	0.0
4	77	16.0	0.0	64.8	3.0	1.2	3.9	0.0
...	...	...	...	...	...	...	...	...
239	65	14.0	0.0	85.4	16.0	4.5	16.9	1.0
240	87	15.0	4.4	41.1	6.5	0.1	6.2	0.0
241	87	29.0	0.5	45.9	3.5	0.4	3.4	0.0
242	54	18.0	0.1	79.7	4.3	1.7	5.1	0.0
243	64	15.0	0.2	67.3	3.8	1.2	4.8	0.0

244 rows × 8 columns

In [245]:
```
y
```

Out[245]:
```
0      29
1      29
2      26
3      25
4      27
       ..
239    30
240    28
241    27
242    24
243    24
Name: Temperature, Length: 244, dtype: int32
```

In [246]:
```
from sklearn.model_selection import train_test_split
```

In [247]:
```
X_train,X_test, Y_train, Y_test = train_test_split(x,y, test_size = 0.33, ra
```

In [248]:
```
1 X_train
```

Out[248]:

	RH	Ws	Rain	FFMC	DMC	ISI	BUI	Classes
114	54	11.0	0.5	73.7	7.9	1.2	9.6	0.0
65	65	13.0	0.0	86.8	11.1	5.2	11.5	1.0
132	42	21.0	0.0	90.6	18.2	13.4	18.0	1.0
207	40	18.0	0.0	92.1	56.3	14.3	59.5	1.0
162	56	15.0	2.9	74.8	7.1	1.6	6.8	0.0
...	...	...	...	...	...	...	...	...
106	82	15.0	0.4	44.9	0.9	0.2	1.4	0.0
14	80	17.0	3.1	49.4	3.0	0.4	3.0	0.0
92	76	17.0	7.2	46.0	1.3	0.2	1.8	0.0
179	57	16.0	0.0	87.5	15.7	6.7	15.7	1.0
102	77	21.0	1.8	58.5	1.9	1.1	2.4	0.0

163 rows × 8 columns

In [249]:
```
1 X_train.shape
```

Out[249]:
```
(163, 8)
```

In [250]:
```
1 Y_train
```

Out[250]:
```
114    32
65     34
132    31
207    34
162    34
       ..
106    24
14     28
92     25
179    33
102    30
Name: Temperature, Length: 163, dtype: int32
```

In [251]:
```
1 Y_train.shape
```

Out[251]:
```
(163,)
```

In [252]:
```
1 X_test.shape
```

Out[252]:
```
(81, 8)
```

In [253]:
```
1 Y_test.shape
```

Out[253]:
```
(81,)
```

Data Cleaning for Better Model Prediction

- Converting Object Dtype to Float Dtype for Model Accuracy in X_train dataset

In [221]:
```
1 df['FWI'].dtype
```

Out[221]:
```
dtype('O')
```

In [223]:
```
1 df['DC'].dtype
```

Out[223]:
```
dtype('O')
```

In [226]:
```
1  ## In DC & FWI Feature there are many Object Dtype Available, after Observi
2 df = df.drop(columns = ('DC'),  axis = 1)
3 df = df.drop(columns = ('FWI'),  axis = 1)
```

In [254]:
```
1 df
```

Out[254]:

	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	BUI	Classes	region	date
0	29	57	18.0	0.0	65.7	3.4	1.3	3.4	0.0	bejaia	2012-06-01
1	29	61	13.0	1.3	64.4	4.1	1.0	3.9	0.0	bejaia	2012-06-02
2	26	82	22.0	13.1	47.1	2.5	0.3	2.7	0.0	bejaia	2012-06-03
3	25	89	13.0	2.5	28.6	1.3	0.0	1.7	0.0	bejaia	2012-06-04
4	27	77	16.0	0.0	64.8	3.0	1.2	3.9	0.0	bejaia	2012-06-05
...	...	...	...	...	...	...	...	...	...	...	...
239	30	65	14.0	0.0	85.4	16.0	4.5	16.9	1.0	Sidi-Bel Abbes	2012-09-26
240	28	87	15.0	4.4	41.1	6.5	0.1	6.2	0.0	Sidi-Bel Abbes	2012-09-27
241	27	87	29.0	0.5	45.9	3.5	0.4	3.4	0.0	Sidi-Bel Abbes	2012-09-28
242	24	54	18.0	0.1	79.7	4.3	1.7	5.1	0.0	Sidi-Bel Abbes	2012-09-29
243	24	64	15.0	0.2	67.3	3.8	1.2	4.8	0.0	Sidi-Bel Abbes	2012-09-30

244 rows × 11 columns

In [255]:
```
import numpy as np
df.info()  ## Now After Data Cleaning
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   Temperature  244 non-null    int32         
 1   RH           244 non-null    int32         
 2   Ws           244 non-null    float64       
 3   Rain         244 non-null    float64       
 4   FFMC         244 non-null    float64       
 5   DMC          244 non-null    float64       
 6   ISI          244 non-null    float64       
 7   BUI          244 non-null    float64       
 8   Classes      244 non-null    float64       
 9   region       244 non-null    object        
 10  date         244 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(7), int32(2), object(1)
memory usage: 19.2+ KB
```

Standardize or Feature Scalling the Dataset

In [256]:
```
## Standardize or Feature Scalling the Dataset

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

```

In [257]:
```
scaler     ## Scaler is an Object , an Standard Scaler Objects
```

Out[257]:
```
StandardScaler()
```

In [258]:
```
X_train = scaler.fit_transform(X_train)       ## Fit_transport means  Applyi
```

In [259]:
```
X_test = scaler.transform(X_test)
```

In [260]:
```
1 X_train
```

Out[260]:
```
array([[-0.60257784, -1.68484146, -0.17054229, ..., -0.80014076,
        -0.47763563, -1.04390785],
       [ 0.14460201, -0.93856657, -0.39436188, ...,  0.16132584,
        -0.3471914 ,  0.95793896],
       [-1.41768313,  2.04653297, -0.39436188, ...,  2.13233237,
         0.09906517,  0.95793896],
       ...,
       [ 0.89178186,  0.5539832 ,  2.82864022, ..., -1.04050741,
        -1.01314351, -1.04390785],
       [-0.39880152,  0.18084575, -0.39436188, ...,  0.52187581,
        -0.058841  ,  0.95793896],
       [ 0.9597073 ,  2.04653297,  0.41138865, ..., -0.82417743,
        -0.9719506 , -1.04390785]])
```

In [261]:
```
1 X_test
```

Out[261]:
```
array([[ 0.07667657, -0.19229169, -0.39436188,  0.67685449, -0.03052244,
         0.28150916,  0.11966162,  0.95793896],
       [-0.60257784, -0.93856657, -0.39436188,  0.77931297, -0.37009673,
         0.44976582, -0.38838431,  0.95793896],
       [-1.01013048,  0.18084575, -0.39436188,  0.73832958, -0.52803826,
         0.54591248, -0.60121437,  0.95793896],
       [-0.67050328,  0.5539832 , -0.17054229,  0.23286778,  0.48278753,
        -0.43959079,  0.96411636,  0.95793896],
       [-1.48560857, -2.0579789 , -0.34959796,  1.03887443,  0.63283198,
         1.19490243,  0.52472528,  0.95793896],
       [ 0.07667657, -2.43111635,  0.14280514, -0.20428836, -0.22794936,
        -0.82417743, -0.35405689, -1.04390785],
       [-1.62145945, -0.19229169, -0.30483404,  0.85444918,  0.09583078,
         0.88242579, -0.05197552,  0.95793896],
       [ 1.16348363, -0.56542913,  0.50091648, -1.91876013, -0.97817163,
        -1.01647075, -0.95821962, -1.04390785],
       [ 1.09555819, -1.31170402, -0.39436188, -0.24527175, -0.40168504,
        -0.7761041 , -0.27167106, -1.04390785],
       [ 0.68800554, -1.31170402,  0.41138865, -1.15373687, -0.97817163,
```

Model Training

Linear Regression Model

In [262]:
```
1 ## Linear Regression
2 from sklearn.linear_model import LinearRegression
```

In [263]:
```
1 regression = LinearRegression()
```

In [264]:
```
1 regression
```

Out[264]:
```
LinearRegression()
```

In [265]:
```
regression.fit(X_train, Y_train)
```

Out[265]:
```
LinearRegression()
```

print the Coefficients and the intercept

In [266]:
```
## print the Coefficients
print(regression.coef_)

```

```
[-1.04543262 -0.47357018  0.10021995  1.82327758  0.12573135  0.17088561
  0.24335231 -0.26484938]
```

In [267]:
```
## print the intercept
print(regression.intercept_)

```

```
31.98159509202454
```

In [268]:
```
## Prediction for the Test data
reg_pred = regression.predict(X_test)

```

In [269]:
```
reg_pred

```

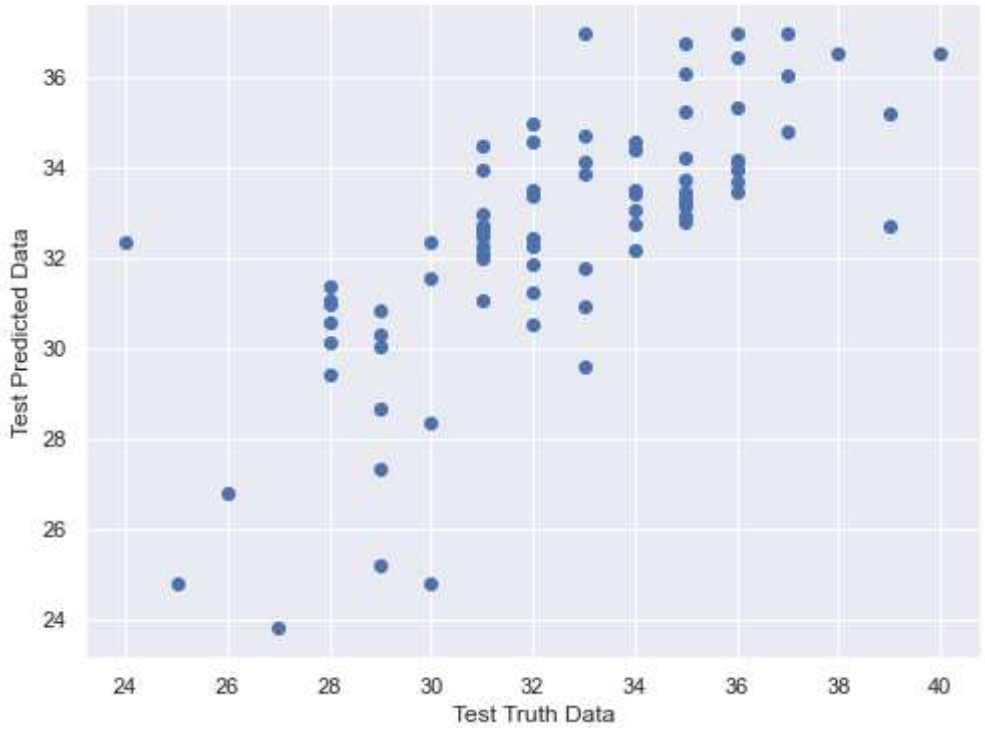
Out[269]:
```
array([33.00674784, 34.11950967, 33.8855133 , 32.79419423, 36.52615796,
       32.71539397, 35.19161778, 27.33139756, 30.99796245, 29.58586948,
       29.42460255, 33.4274263 , 33.95352094, 33.48023065, 34.19311614,
       32.19700985, 36.97985439, 25.21750399, 32.26401384, 33.50420952,
       30.93833937, 28.3669283 , 34.99954884, 28.69083985, 36.52529739,
       26.82074265, 32.70997918, 33.27291072, 32.91903121, 34.58048311,
       34.51522327, 31.57661772, 32.62103567, 33.31912978, 32.71199731,
       33.38364766, 30.30169034, 34.25125238, 31.78783181, 23.79809099,
       33.47076807, 33.73980521, 32.48126685, 24.81636379, 36.06878401,
       32.43639042, 31.25547958, 30.55355193, 35.25600003, 34.59119977,
       36.96746325, 30.86723899, 31.05572071, 34.39878504, 33.69137609,
       32.28932331, 36.99087506, 32.36649978, 30.1224896 , 36.46783802,
       33.08294334, 30.02673524, 33.96201881, 32.01368053, 31.86600619,
       24.78309326, 33.14413455, 30.60056421, 36.77240924, 34.81131382,
       32.7574173 , 31.0870985 , 33.24905999, 34.74176468, 36.08014323,
       31.39779738, 33.50195551, 32.08081288, 35.33279912, 32.35919129,
       34.14751369])
```

Assumption of Linear Regression.

In [270]:
```
## Assumption of Linear Regression.

import seaborn as sns
import matplotlib.pyplot as plt
```

In [271]:
```
## Relationship Between Real Data & Predicted Data
plt.scatter(Y_test,reg_pred)          ## IF we get Linear Manner , it is goo
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")      ## When we take Test data & Predictio

```

Out[271]: Text(0, 0.5, 'Test Predicted Data')



In [272]:
```
## Calculating Residual
residuals = Y_test - reg_pred

```

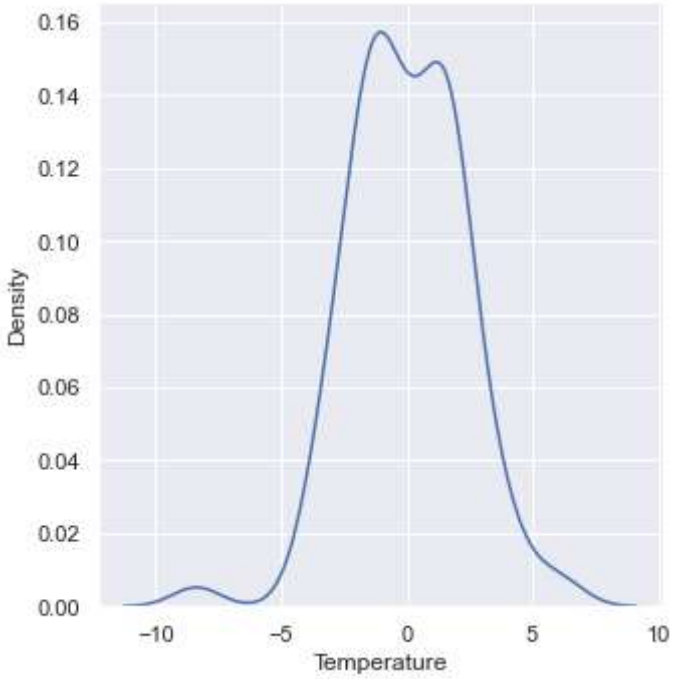
In [273]:
```
residuals
```

Out[273]:
```
24    -2.006748
6     -1.119510
153   -0.885513
211    2.205806
198    3.473842
          ...
180    0.498044
5     -1.080813
56     0.667201
125   -2.359191
148    1.852486
Name: Temperature, Length: 81, dtype: float64
```

In [274]:

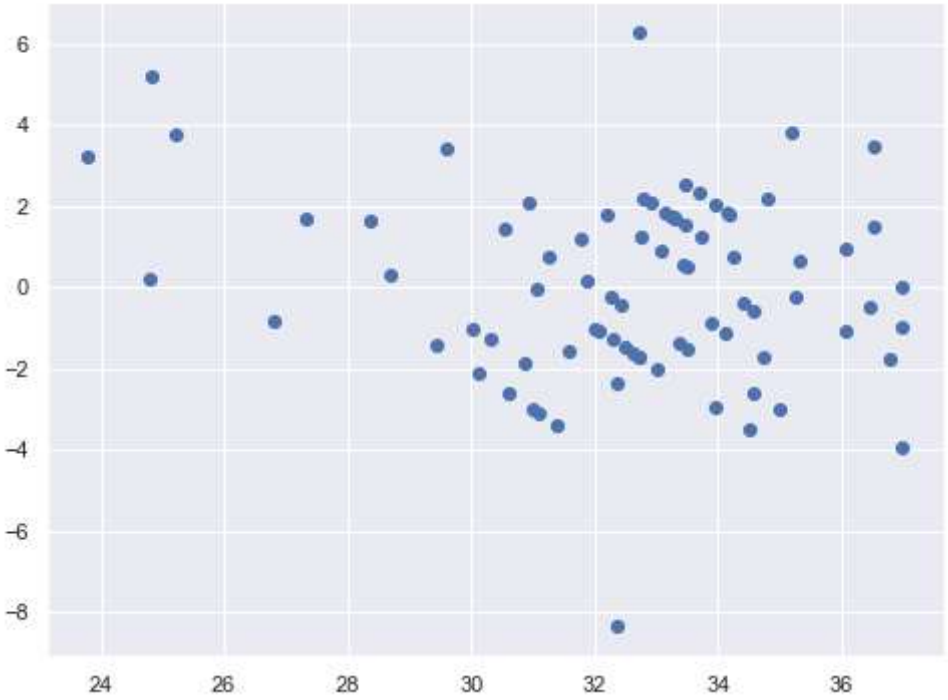
```
## Distribution of residual are Approximately Normal Distribution
sns.displot(residuals, kind ="kde")

```

Out[274]: <seaborn.axisgrid.FacetGrid at 0x20e1caeb850>



In [275]:
```
## Scatter Plot with predictions and residual
### Uniform Distributions
plt.scatter(reg_pred, residuals)    ## Uniform Distributions :- Model is in

```

Out[275]: <matplotlib.collections.PathCollection at 0x20e1e5c8dc0>



Performance Metrics

In [276]:
```
## Performance Metrics

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

print(mean_squared_error(Y_test, reg_pred))
print(mean_absolute_error(Y_test,reg_pred))
print(np.sqrt(mean_squared_error(Y_test, reg_pred)))
```

```
5.199575881104852
1.8282040182595614
2.280257854082483
```

R Squared and Adjusted R-Squared

In [277]:
```
## R Squared
from sklearn.metrics import r2_score
score=r2_score(Y_test,reg_pred)
print(score)

```

```
0.5159015558971345
```

In [278]:

```
## Adjusted R square
#display adjusted R-squared
1 - (1-score)*(len(Y_test)-1)/(len(Y_test)-X_test.shape[1]-1)

```

Out[278]: 0.46211283988570495

Thank you