Algerain Forest Fire

- Machine Learning LifeCycle
    1. Data Ingestion
    2. EDA
    3. Preprocessing
    4. Model Building
    5. Performance Metrics

- Problem Statement

```
 - 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 th
e Sidi Bel-abbes region located in the northwest of Algeria.
  122 instances for each region.
  The period from June 2012 to September 2012.
  The dataset includes 11 attribues and 1 output attribue (class)
  The 244 instances have been classified into fire (138 classes) and not fi
re (106 classes) classes.
```

- Feature Description
    1. Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012) Weather data observations
    2. Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42
    3. RH : Relative Humidity in %: 21 to 90
    4. Ws :Wind speed in km/h: 6 to 29
    5. Rain: total day in mm: 0 to 16.8 FWI Components
    6. Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5
    7. Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9
    8. Drought Code (DC) index from the FWI system: 7 to 220.4
    9. Initial Spread Index (ISI) index from the FWI system: 0 to 18.5
    10. Buildup Index (BUI) index from the FWI system: 1.1 to 68
    11. Fire Weather Index (FWI) Index: 0 to 31.1
    12. Classes: two classes, namely Fire and not Fire

- Machine Learning Models Used
    1. Linear Regression
    2. Ridge Regression
    3. Lasso Regression
    4. Elastic-Net Regression

In [1]:
```
## Importing the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:
```
## importing the Dataset
data=pd.read_csv(r'Algerian_Forest_Fire.csv',header=1)
```

In [3]:
```
## There are two regions Algeria and Bejaia in same file it needs to be
## converted into single dataframe for the anaylsis
data=data.drop([122,123],axis=0)
```

In [4]:
```
# Printing the Column Labels in the dataset
data.columns
```

Out[4]:
```
Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain ', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes  '],
      dtype='object')
```

In [5]:
```
## Renaming the Columns which has space in the column labels
data.rename(columns={' RH':'RH','Rain ':'Rain',' Ws':'Ws','Classes  ':'Classes'},inplace
```

In [6]:
```
data.columns
```

Out[6]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes'],
      dtype='object')
```

In [7]:
```
# Top Five Records in the dataset
data.head()
```

Out[7]:

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

In [8]:
```
# Sample five records in the dataset
data.sample(5)
```

Out[8]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
227	12	09	2012	31	72	14	0	84.2	8.3	25.2	3.8	9.1	3.9	fire
11	12	06	2012	26	81	19	0	84	13.8	61.4	4.8	17.7	7.1	fire
58	29	07	2012	32	73	15	0	86.6	26.7	127	5.6	35	11.9	fire
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
18	19	06	2012	31	55	16	0.1	79.9	4.5	16	2.5	5.3	1.4	not fire

In [9]:
```
# Checking whether there is any duplicate records in the dataset
data.duplicated().sum()
```

Out[9]:
```
0
```

Observation: There is no records which is duplicated in the dataset

In [10]:
```
## Checking the total number of records and columns with the dataset
data.shape
```

Out[10]:
```
(244, 14)
```

Exploratory Data Anaylsis

In [11]:
```
## Detailed information of each features in the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 244 entries, 0 to 245
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   day          244 non-null    object
 1   month        244 non-null    object
 2   year         244 non-null    object
 3   Temperature  244 non-null    object
 4   RH           244 non-null    object
 5   Ws           244 non-null    object
 6   Rain         244 non-null    object
 7   FFMC         244 non-null    object
 8   DMC          244 non-null    object
 9   DC           244 non-null    object
 10  ISI          244 non-null    object
 11  BUI          244 non-null    object
 12  FWI          244 non-null    object
 13  Classes      243 non-null    object
dtypes: object(14)
memory usage: 28.6+ KB
```

Observation:

- There are some features which should be converted to int and float type there is some data discriminacy
  in the dataset which need to be preprocessed

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
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        1
dtype: int64
```

Observation:

- There is one record which is null in the feature Classes

In [13]:
```
## Checking the datatype of FWI feature
data['FWI'].dtype
```

Out[13]:
```
dtype('O')
```

In [14]:
```
## Checking the Shape of FWI feature
data['FWI'].shape
```

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

Converting the Feature Datatype

In [15]:
```
### Rain,FFMC,DMC,DC,ISI,BUI,FWI has object type
float_var=['Rain','FFMC','DMC','DC','ISI','BUI','FWI']
```

In [16]:
```
### All the features which are in float_var are converted to float type
for col in float_var:
    data[col] = data[col].apply(pd.to_numeric, downcast='float', errors='coerce')
```

In [17]:
```
data.dtypes
```

Out[17]:
```
day              object
month            object
year             object
Temperature      object
RH               object
Ws               object
Rain            float32
FFMC            float32
DMC             float32
DC              float32
ISI             float32
BUI             float32
FWI             float32
Classes          object
dtype: object
```

In [18]:
```
### day,month,year,Temperature,RH,Ws are in object type
num_var=['day','month','year','Temperature','RH','Ws']
```

In [19]:
```
### All the features in num_var are converted into int type
for col in num_var:
    data[col]=data[col].astype('int')
```

In [20]:
```
data.dtypes
```

Out[20]:
```
day              int32
month            int32
year             int32
Temperature      int32
RH               int32
Ws               int32
Rain           float32
FFMC           float32
DMC            float32
DC             float32
ISI            float32
BUI            float32
FWI            float32
Classes         object
dtype: object
```

In [21]:
```
data.head()
```

Out[21]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	1	6	2012	29	57	18	0.0	65.699997	3.4	7.6	1.3	3.4	0.5	not fire
1	2	6	2012	29	61	13	1.3	64.400002	4.1	7.6	1.0	3.9	0.4	not fire
2	3	6	2012	26	82	22	13.1	47.099998	2.5	7.1	0.3	2.7	0.1	not fire
3	4	6	2012	25	89	13	2.5	28.600000	1.3	6.9	0.0	1.7	0.0	not fire
4	5	6	2012	27	77	16	0.0	64.800003	3.0	14.2	1.2	3.9	0.5	not fire

Statistical Analysis

In [22]:
```
### Storing All the numerical variable in num_col
num_col=data[data.dtypes[(data.dtypes=='int') | (data.dtypes=='float')|(data.dtypes=='f
```

In [23]:
```
### Describing the stastical analysis of the data
num_col.describe().T
```

Out[23]:

	count	mean	std	min	25%	50%	75%	max
day	244.0	15.754098	8.825059	1.0	8.000000	16.000000	23.000000	31.000000
month	244.0	7.500000	1.112961	6.0	7.000000	7.500000	8.000000	9.000000
year	244.0	2012.000000	0.000000	2012.0	2012.000000	2012.000000	2012.000000	2012.000000
Temperature	244.0	32.172131	3.633843	22.0	30.000000	32.000000	35.000000	42.000000
RH	244.0	61.938525	14.884200	21.0	52.000000	63.000000	73.250000	90.000000
Ws	244.0	15.504098	2.810178	6.0	14.000000	15.000000	17.000000	29.000000
Rain	244.0	0.760656	1.999407	0.0	0.000000	0.000000	0.500000	16.799999
FFMC	244.0	77.887680	14.337568	28.6	72.075003	83.500000	88.300003	96.000000
DMC	244.0	14.673360	12.368040	0.7	5.800000	11.300000	20.750000	65.900002
DC	243.0	49.430847	47.665600	6.9	12.350000	33.099998	69.099998	220.399994
ISI	244.0	4.774180	4.175318	0.0	1.400000	3.500000	7.300000	19.000000
BUI	244.0	16.664747	14.204824	1.1	6.000000	12.250000	22.525000	68.000000
FWI	243.0	7.035391	7.440567	0.0	0.700000	4.200000	11.450001	31.100000

Observation

1. Rain Feature is highly skewed on the right
2. year feature is not required as it has only one unique value through out the dataset
3. Rain Features has some outliers
4. BUI also right skewed and has some outliers

Drop the year Feature

In [24]:
```
### Unique values in the year featue
num_col['year'].unique()
```

Out[24]:
```
array([2012])
```

Observation:

- As there is only one year it has been analysed for both the regions it can be dropped from the dataset

In [25]:
```
### Droping the year feature
num_col.drop(labels=['year'],axis=1,inplace=True)
```

In [26]:
```
num_col.head()
```

Out[26]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI
0	1	6	29	57	18	0.0	65.699997	3.4	7.6	1.3	3.4	0.5
1	2	6	29	61	13	1.3	64.400002	4.1	7.6	1.0	3.9	0.4
2	3	6	26	82	22	13.1	47.099998	2.5	7.1	0.3	2.7	0.1
3	4	6	25	89	13	2.5	28.600000	1.3	6.9	0.0	1.7	0.0
4	5	6	27	77	16	0.0	64.800003	3.0	14.2	1.2	3.9	0.5

Correlation between the features

In [27]:
```
num_col.corr()
```

Out[27]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC
day	1.000000e+00	2.232788e-17	0.095772	-0.074209	0.047001	-0.112265	0.224032	0.49157
month	2.232788e-17	1.000000e+00	-0.059017	-0.037884	-0.041447	0.035322	0.015577	0.06817
Temperature	9.577222e-02	-5.901677e-02	1.000000	-0.654443	-0.278132	-0.326786	0.677491	0.48310
RH	-7.420934e-02	-3.788419e-02	-0.654443	1.000000	0.236084	0.222968	-0.645658	-0.40513
Ws	4.700086e-02	-4.144673e-02	-0.278132	0.236084	1.000000	0.170169	-0.163255	-0.00124
Rain	-1.122654e-01	3.532207e-02	-0.326786	0.222968	0.170169	1.000000	-0.544045	-0.28854
FFMC	2.240321e-01	1.557669e-02	0.677491	-0.645658	-0.163255	-0.544045	1.000000	0.60239
DMC	4.915710e-01	6.817778e-02	0.483105	-0.405133	-0.001246	-0.288548	0.602391	1.00000
DC	5.279524e-01	1.265111e-01	0.376284	-0.226941	0.079135	-0.298023	0.507397	0.87592
ISI	1.777266e-01	6.168012e-02	0.607551	-0.690637	0.015248	-0.347105	0.739730	0.67449
BUI	5.172292e-01	8.582162e-02	0.455504	-0.348587	0.029756	-0.299171	0.589652	0.98207
FWI	3.507809e-01	8.263884e-02	0.566670	-0.580957	0.032368	-0.324422	0.691132	0.87586

Observation:

- Temperature and Fine Fuel Moisture Code (FFMC) are Positively Correlated .
- Temperature and Relative Humidity (RH) are Negatively Correlated .
- Temperature and Initial Spread Index (ISI) are Positively Correlated .
- Buildup Index (BUI) and Duff Moisture Code (DMC) are highly Correlated but both of them are independent variable
  which leads to MultiCollinearity Which should be handled
- Drought Code (DC) and Buildup Index (BUI) are also highly Correlated they also make MultiCollinearity

In [28]:
```
### Make a Copy of the DataFrame
df=num_col.copy()
```

In [29]:
```
### Appending the Classes feature in df
df['Classes']=data['Classes']
```

In [30]:
```
df.head()
```

Out[30]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	1	6	29	57	18	0.0	65.699997	3.4	7.6	1.3	3.4	0.5	not fire
1	2	6	29	61	13	1.3	64.400002	4.1	7.6	1.0	3.9	0.4	not fire
2	3	6	26	82	22	13.1	47.099998	2.5	7.1	0.3	2.7	0.1	not fire
3	4	6	25	89	13	2.5	28.600000	1.3	6.9	0.0	1.7	0.0	not fire
4	5	6	27	77	16	0.0	64.800003	3.0	14.2	1.2	3.9	0.5	not fire

In [31]:
```
### showing the unique values in the Classes feature
df['Classes'].unique()
```

Out[31]:
```
array(['not fire    ', 'fire    ', 'fire', 'fire ', 'not fire', 'not fire ',
       'not fire     ', nan, 'not fire    '], dtype=object)
```

Observation:

- It seems that there are lot of spacing which should be stripped off.

In [32]:
```
### Stripping the object on both sides

for i in df.columns:
    if df[i].dtypes=='O':
        df[i]=df[i].str.strip()
```

In [33]:
```
df['Classes'].unique()
```

Out[33]:
```
array(['not fire', 'fire', nan], dtype=object)
```

Observation:

- There is some null values in the Classes feature which should be preprocessed

In [34]:
```
### There are one missing values which can be dropped off
df['Classes'].isnull().sum()
```

Out[34]:
```
1
```

In [35]:
```
df.isnull().sum()
```

Out[35]:
```
day            0
month          0
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             1
ISI            0
BUI            0
FWI            1
Classes        1
dtype: int64
```

In [36]:
```
df.dropna(inplace=True)
```

Observation: There are some missing values which can be dropped off from the dataframe

In [37]:
```
df.isnull().sum()
```

Out[37]:
```
day            0
month          0
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        0
dtype: int64
```

In [38]:
```
df['Classes'].unique()
```

Out[38]:
```
array(['not fire', 'fire'], dtype=object)
```

What is Maximum Temperature When the Fire has happened

In [39]:
```
df.groupby('Classes')['Temperature'].max()
```

Out[39]:
```
Classes
fire        42
not fire    39
Name: Temperature, dtype: int32
```

Observation: The maximum temperature was around 42 degree celsius

What was the Average Relative Humidity when the Fire happened where the Temperature was over 38 degree

In [40]:
```
df[(df['Classes']=='fire') & (df['Temperature']>38)]['RH'].mean()
```

Out[40]:
```
34.55555555555556
```

Observation: The average Humidity was around 34%

Display all Average Temperature of month where the fire had happened

In [41]:
```
df[df['Classes']=='fire'].groupby(['month'])['Temperature'].mean()
```

Out[41]:
```
month
6    32.440000
7    33.842105
8    35.647059
9    31.086957
Name: Temperature, dtype: float64
```

Display all the details of the list where the fire has happened with maximum temperature

In [42]:
```
df[(df['Classes']=='fire') & (df['Temperature']==max(df['Temperature']))]
```

Out[42]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
201	17	8	42	24	9	0.0	96.0	30.299999	76.400002	15.7	30.4	24.0	fire

Display the Average Realtive Humidity for each month where temperature was more than 38 degree

In [43]:
```
df[df['Temperature']>38].groupby('month')['RH'].mean()
```

Out[43]:
```
month
7    54.50
8    33.25
Name: RH, dtype: float64
```

Display the skewness of the dataset

In [44]:
```
df.skew()
```

Out[44]:
```
day            0.000365
month         -0.005207
Temperature   -0.191327
RH            -0.242790
Ws             0.555586
Rain           4.568630
FFMC          -1.320130
DMC            1.522983
DC             1.473460
ISI            1.140243
BUI            1.452745
FWI            1.147593
dtype: float64
```

Observation: It seems that the rain is highly positive skewed has more outliers

What is the Ram Space used by the dataset

In [45]:
```
df.memory_usage()
```

Out[45]:
```
Index          1944
day             972
month           972
Temperature     972
RH              972
Ws              972
Rain            972
FFMC            972
DMC             972
DC              972
ISI             972
BUI             972
FWI             972
Classes        1944
dtype: int64
```

What is the Temperature when Fire Weather Index was maximum

In [46]:
```
df[df['FWI']==max(df['FWI'])]
```

Out[46]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
209	25	8	34	40	18	0.0	92.099998	56.299999	157.5	14.3	59.5	31.1	fire

What is the Temperature when Fine Fuel Moisture Code was maximum

In [47]:
```
df[df['FFMC']==max(df['FFMC'])]
```

Out[47]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
201	17	8	42	24	9	0.0	96.0	30.299999	76.400002	15.7	30.4	24.0	fire

Covariance between Dependent and Independent Features

In [48]:
```
df.cov()
```

Out[48]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DM
day	78.190729	-0.003639	3.119138	-9.969476	1.188603	-1.993174	28.544042	53.86313
month	-0.003639	1.242764	-0.229653	-0.681903	-0.124987	0.077762	0.272433	0.93867
Temperature	3.119138	-0.229653	13.162670	-35.043482	-2.901949	-2.372850	35.222858	21.83766
RH	-9.969476	-0.681903	-35.043482	219.874333	10.173809	6.604836	-137.215386	-75.07192
Ws	1.188603	-0.124987	-2.901949	10.173809	7.903887	0.965886	-6.718952	-0.02512
Rain	-1.993174	0.077762	-2.372850	6.604836	0.965886	4.012837	-15.634746	-7.16902
FFMC	28.544042	0.272433	35.222858	-137.215386	-6.718952	-15.634746	205.912202	107.34297
DMC	53.863133	0.938676	21.837668	-75.071928	-0.025120	-7.169025	107.342971	153.58743
DC	222.524339	6.722457	65.071727	-160.400449	10.604530	-28.456455	347.051365	517.42774
ISI	6.632060	0.303838	9.101371	-42.298446	0.099643	-2.891688	44.113113	35.03222
BUI	65.061368	1.349400	23.734918	-74.653741	1.257586	-8.546509	120.872509	173.20320
FWI	23.079143	0.685464	15.297068	-64.096917	0.677079	-4.835502	73.791798	80.76453

In [49]:
```
### Finding the outliers in the dataset
def find_outliers_IQR(df):
    q1=df.quantile(0.25)
    q3=df.quantile(0.75)
    IQR=q3-q1
    lowerbound=q1-1.5*IQR
    upperbound=q3+1.5*IQR
    return lowerbound,upperbound
```

Display all the Skewed data of Rain

In [50]:
```
lowerfence,upperfence=find_outliers_IQR(df['Rain'])
```

In [51]:
```
print("The Lower Bound is",lowerfence,"\n"+"The Upper Bound is ",upperfence)
```

```
The Lower Bound is -0.75
The Upper Bound is  1.25
```

In [52]:
```
len(df[(df['Rain']<lowerfence)|(df['Rain']>upperfence)])
```

Out[52]: 35

Observation: There are 35 records in Rain which is skewed and has outliers which should be processed

Display all the Skewed data of Duff Moisture Code

In [53]:
```
lowerfence,upperfence=find_outliers_IQR(df['DMC'])
```

In [54]:
```
print("The Lower Bound is",lowerfence,"\n"+"The Upper Bound is ",upperfence)
```

```
The Lower Bound is -16.699999809265137
The Upper Bound is  43.30000019073486
```

In [55]:
```
len(df[(df['DMC']<lowerfence)|(df['DMC']>upperfence)])
```

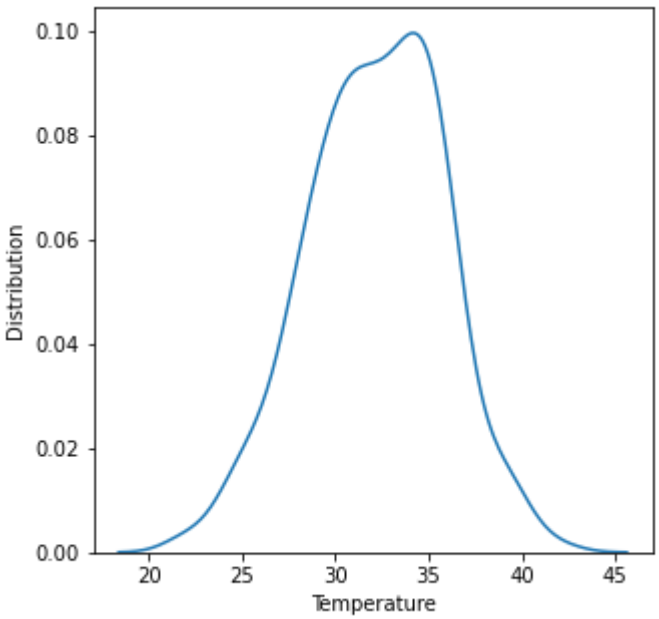
Out[55]: 12

Observation: There are 12 records in Duff Moisture Code which is skewed and has outliers which should be processed

Graph Analysis

In [56]:
```
graph_col=['Temperature','RH','Ws','Rain','FFMC','DMC','DC','ISI','BUI','FWI']
```

In [57]:
```
### Plotting the Kde Plot to check the distribution of the feature
fig, ax = plt.subplots(figsize=(5, 5))
for i in graph_col:
    sns.kdeplot(df[i])
    plt.xlabel(i)
    plt.ylabel('Distribution')
    plt.show()
```



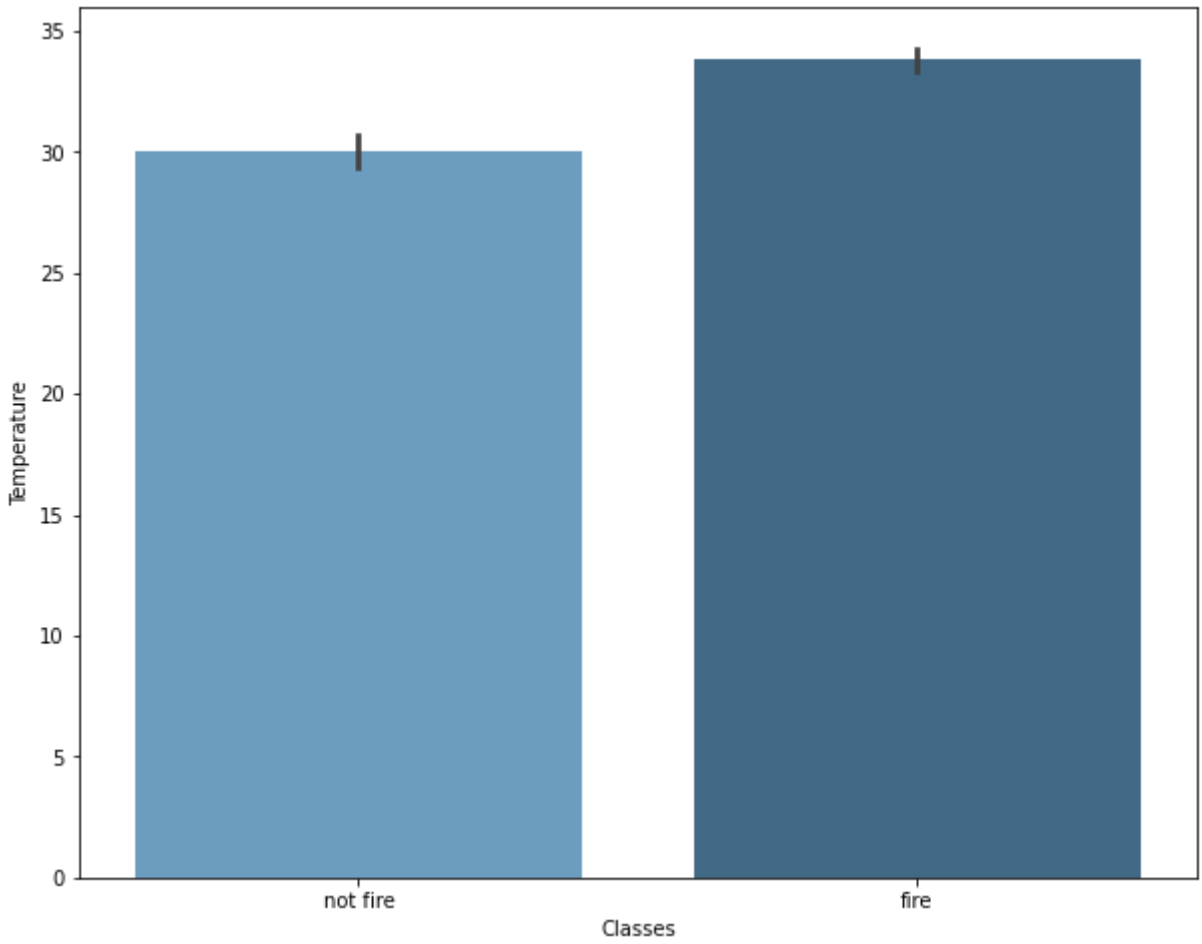
Observation:

1. Temperature is Normally Distributed.
2. Relative Humidity is slightly skewed right side.
3. Wind Speed is Right Skewed.
4. Rain is log normally ditributed right skewed.
5. Fine Fuel Moisture Code is left skewed.
6. Duff Moisture Code (DMC),Drought Code (DC) ,Initial Spread Index (ISI),Build up Index (BUI) ,Fire Weather Index (FWI)
   are slightly right skewed.

Display the Temperature when Fire has happened

In [58]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.barplot(x='Classes',y='Temperature',data=df,palette="Blues_d")
```

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



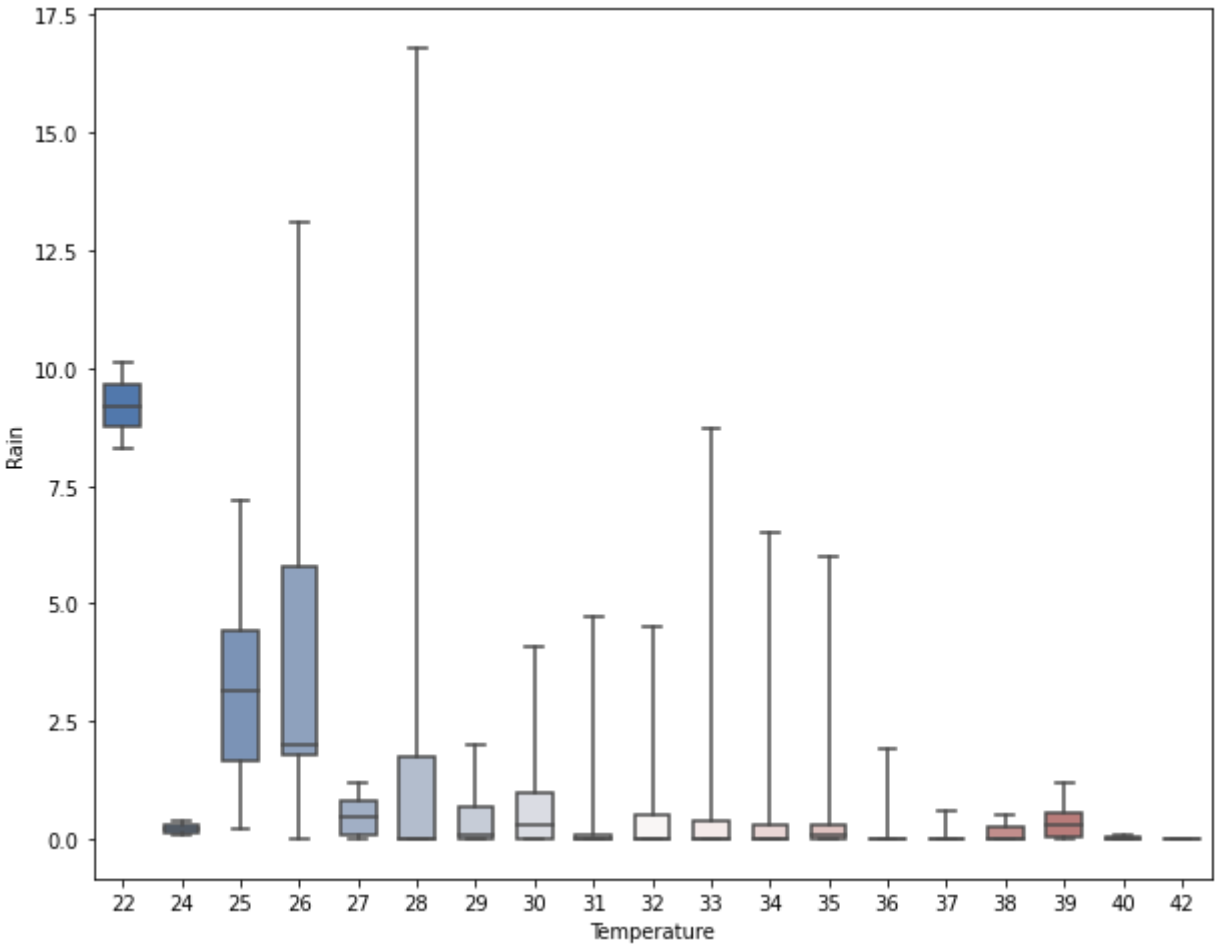
Observation:

- The Average temperature was around 33 Celsius degrees when fire has happened

Display the Distribution of Rain vs Temperature

In [59]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.boxplot(x="Temperature", y="Rain", data=df,
            whis=[0, 100], width=.6, palette="vlag")
```

Out[59]: <AxesSubplot:xlabel='Temperature', ylabel='Rain'>



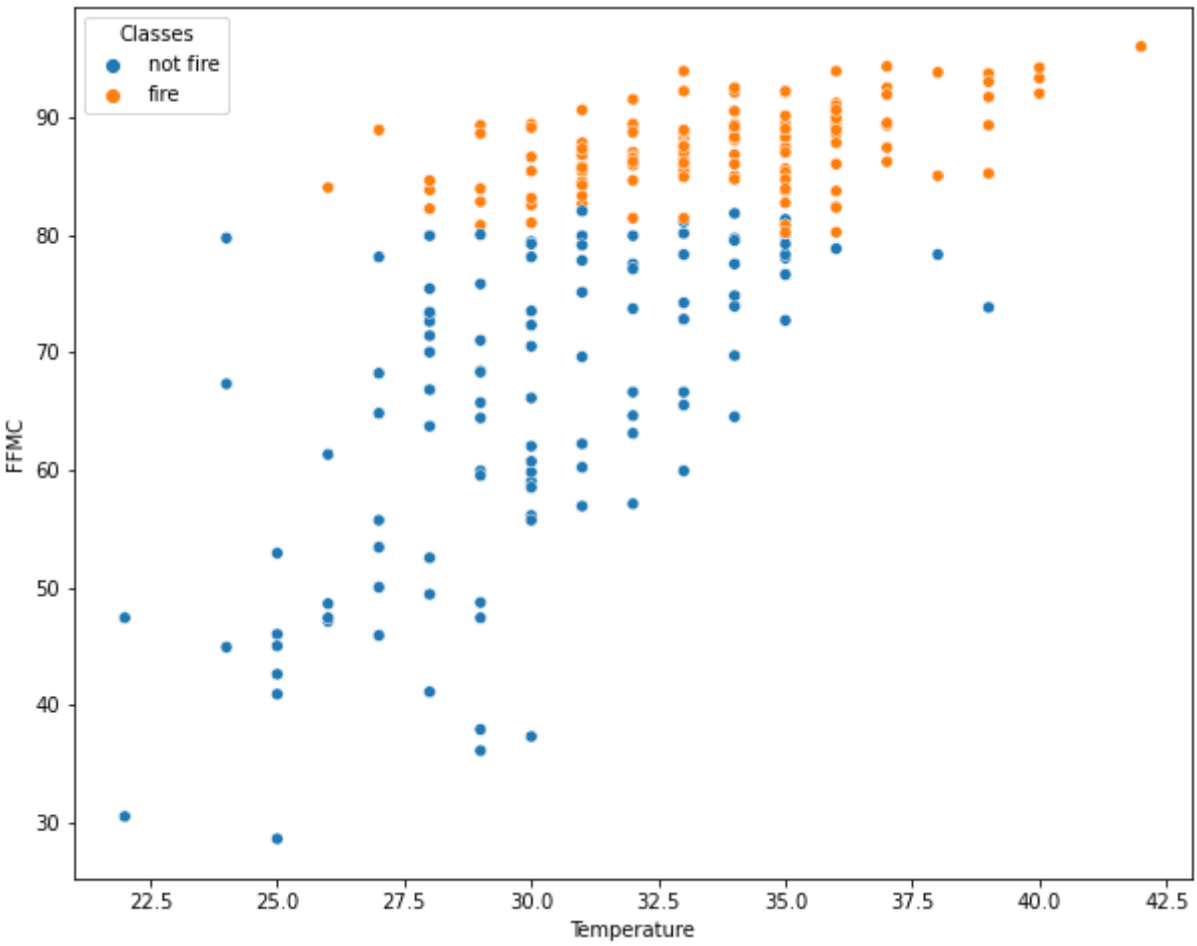
Observation:

- During 28 Celsius degrees has maximum rainfall
- Temperature is inversely correlated with rain

Show the Relation between Temperature and Fine Fuel Moisture Code With Respect to Classes

In [60]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.scatterplot(x='Temperature',y='FFMC',hue='Classes',data=df)
```

Out[60]: <AxesSubplot:xlabel='Temperature', ylabel='FFMC'>



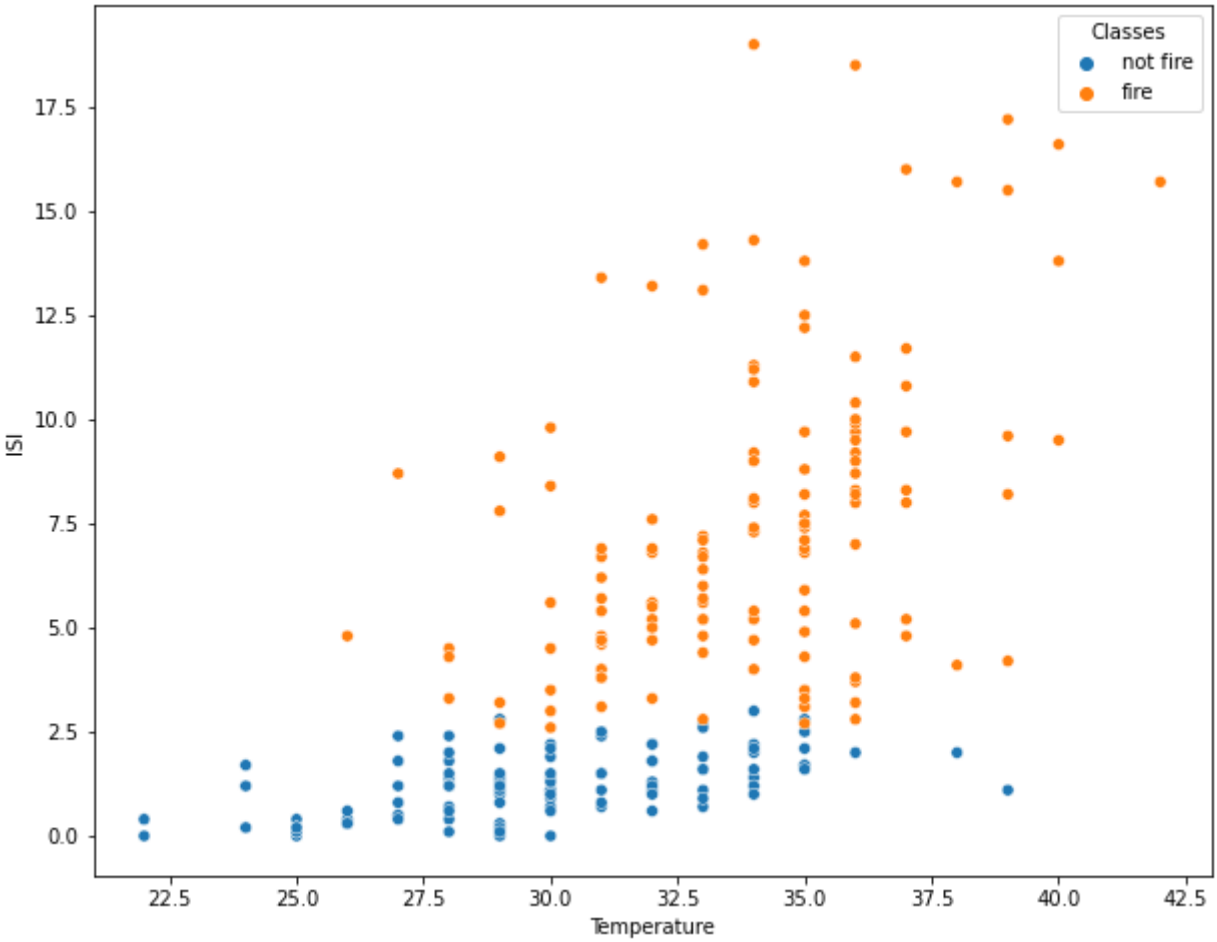
Observation:

- Temperature and Fine Fuel Moisture Code are linearly increasing it is highly correlated
- The Temperature and FFMC are High when Fire has happened

Show the relation between Temperature and Initial Spread Index

In [61]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.scatterplot(x='Temperature',y='ISI',hue='Classes',data=df)
```

Out[61]:
```
<AxesSubplot:xlabel='Temperature', ylabel='ISI'>
```



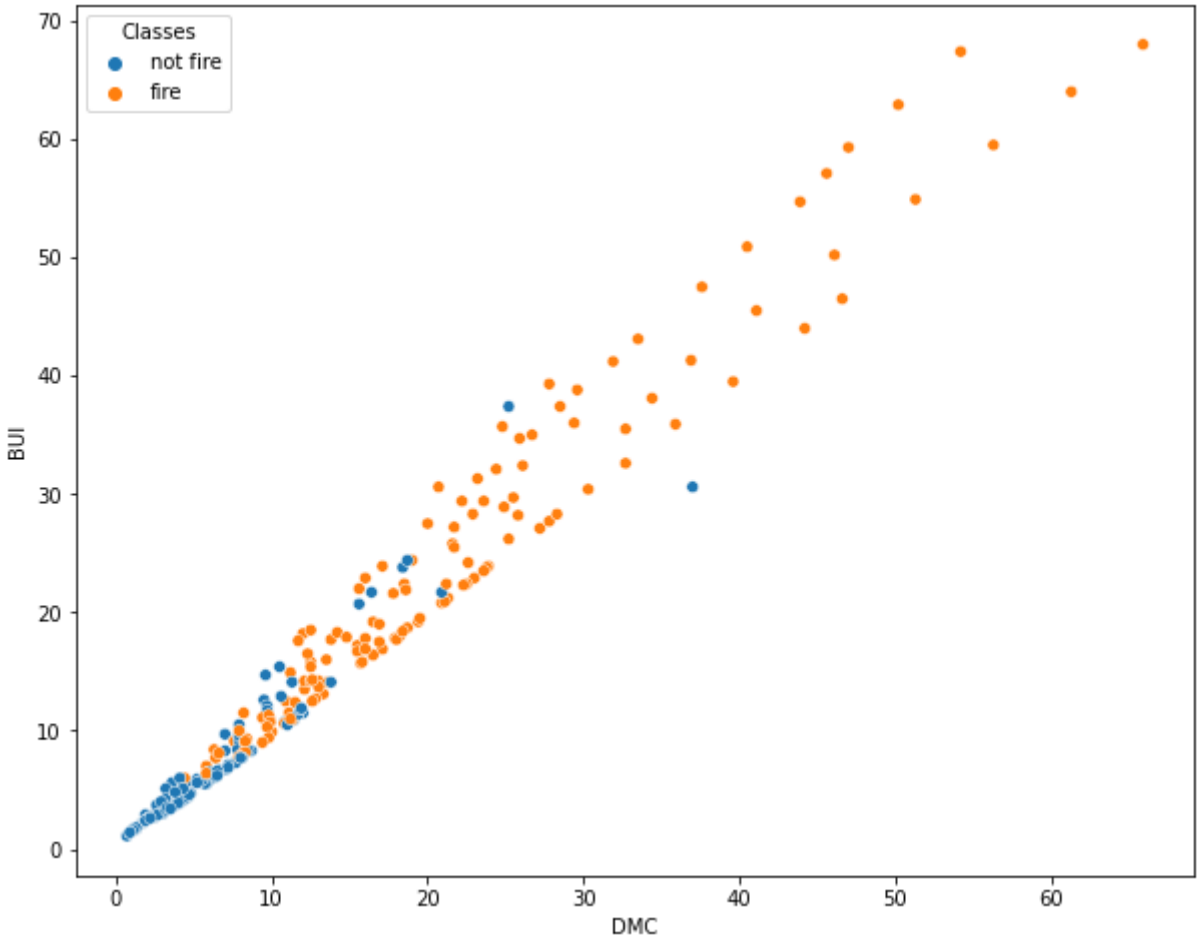
Observation:

- Temperature and Initial Spread Index are linearly increasing it is highly cor related
- The Temperature and ISI are High when Fire has happened

MultiCollinearity

In [62]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.scatterplot(x='DMC',y='BUI',hue='Classes',data=df)
```

Out[62]: <AxesSubplot:xlabel='DMC', ylabel='BUI'>



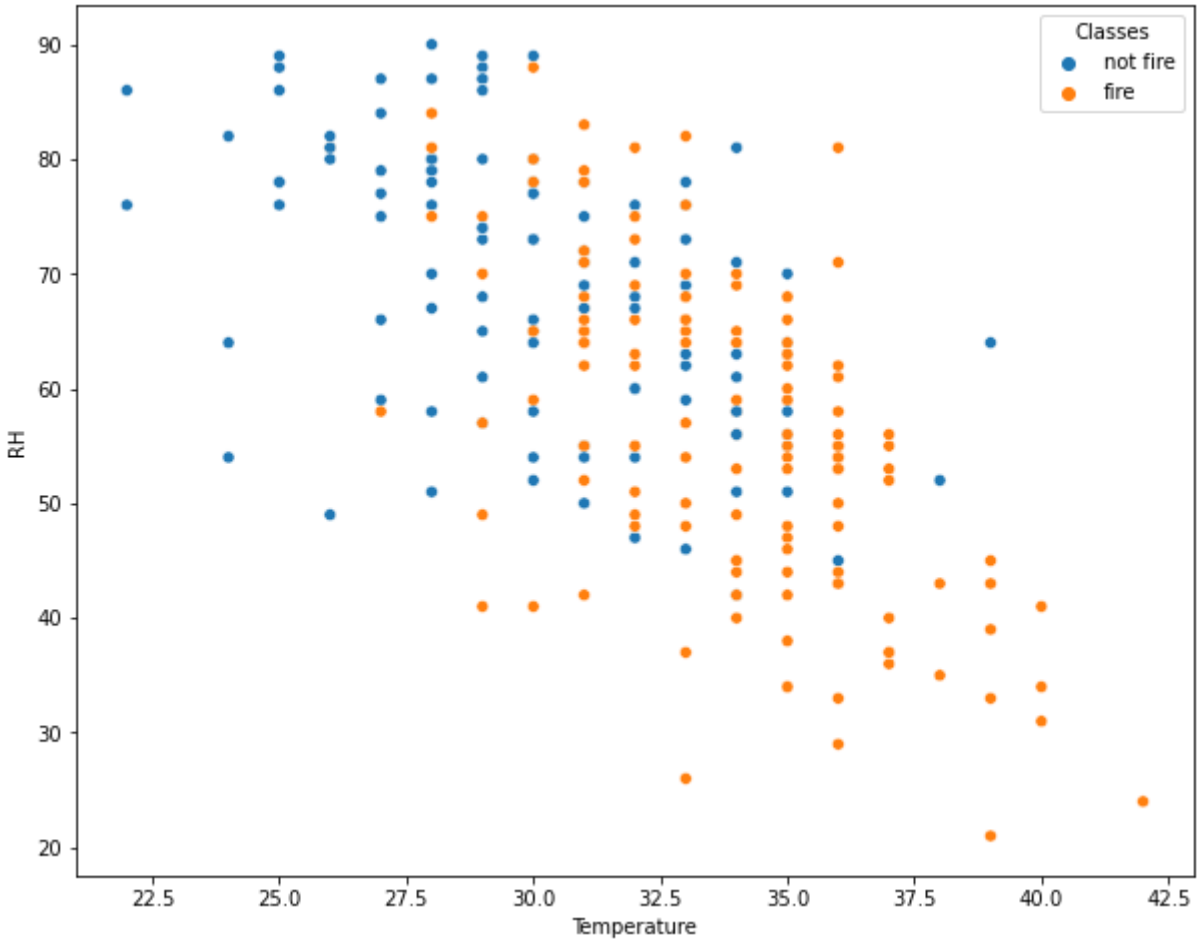
Observation:

- Duff Moisture Code and Buildup Index are two independent varaibles which are highly correlated.
- Either one of the feature should be dropped for the feature selection.

Show the relation between Temperature and Relative Humidity

In [63]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.scatterplot(x='Temperature',y='RH',hue='Classes',data=df)
```

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

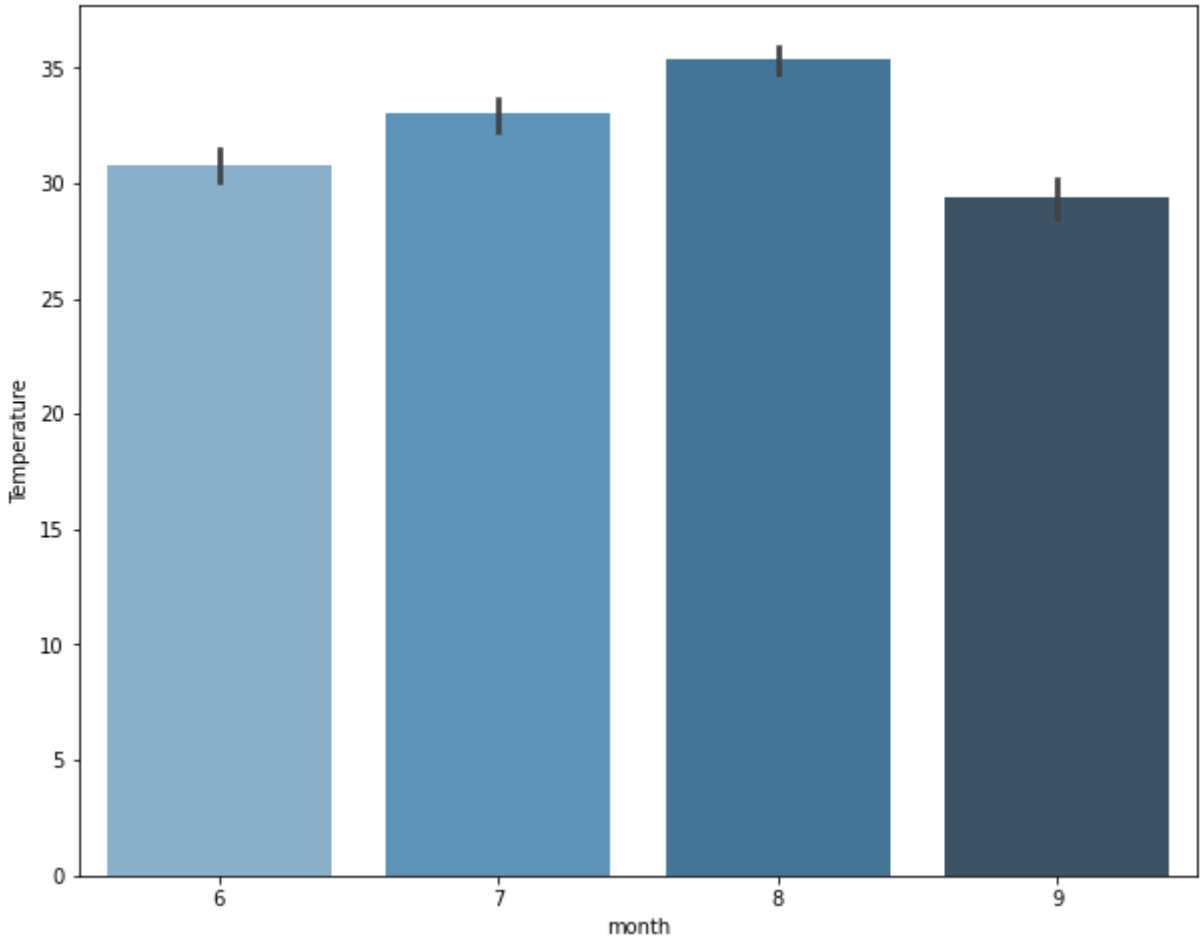


Observation:

- Temperature and Relative Humidity are negative Correlated
- Rumidity is less when the fire has happened but the temperature was large

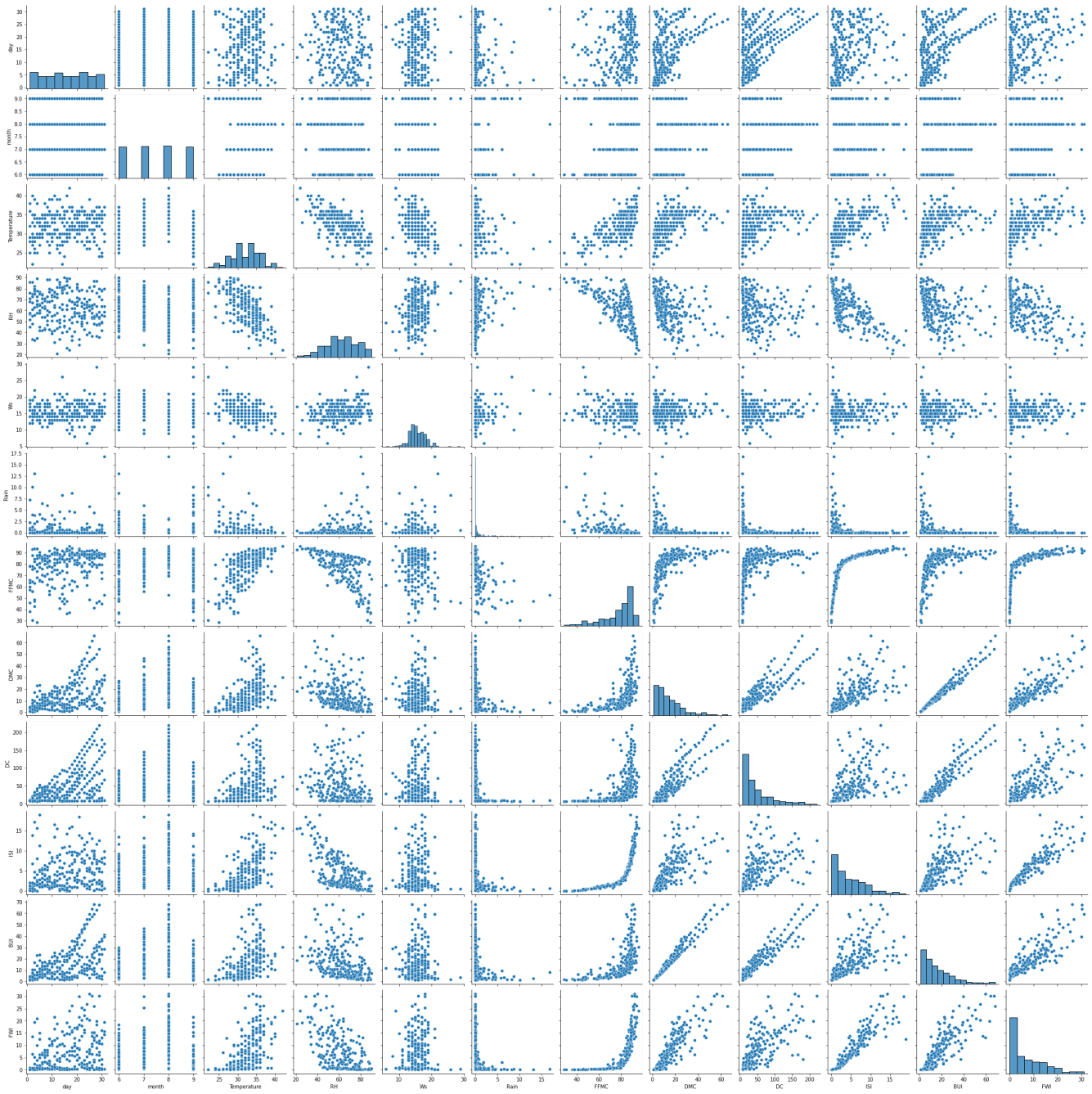
In [64]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.barplot(x='month',y='Temperature',data=df,palette="Blues_d")
```

Out[64]: <AxesSubplot:xlabel='month', ylabel='Temperature'>



In [65]:
```
sns.pairplot(df)
```

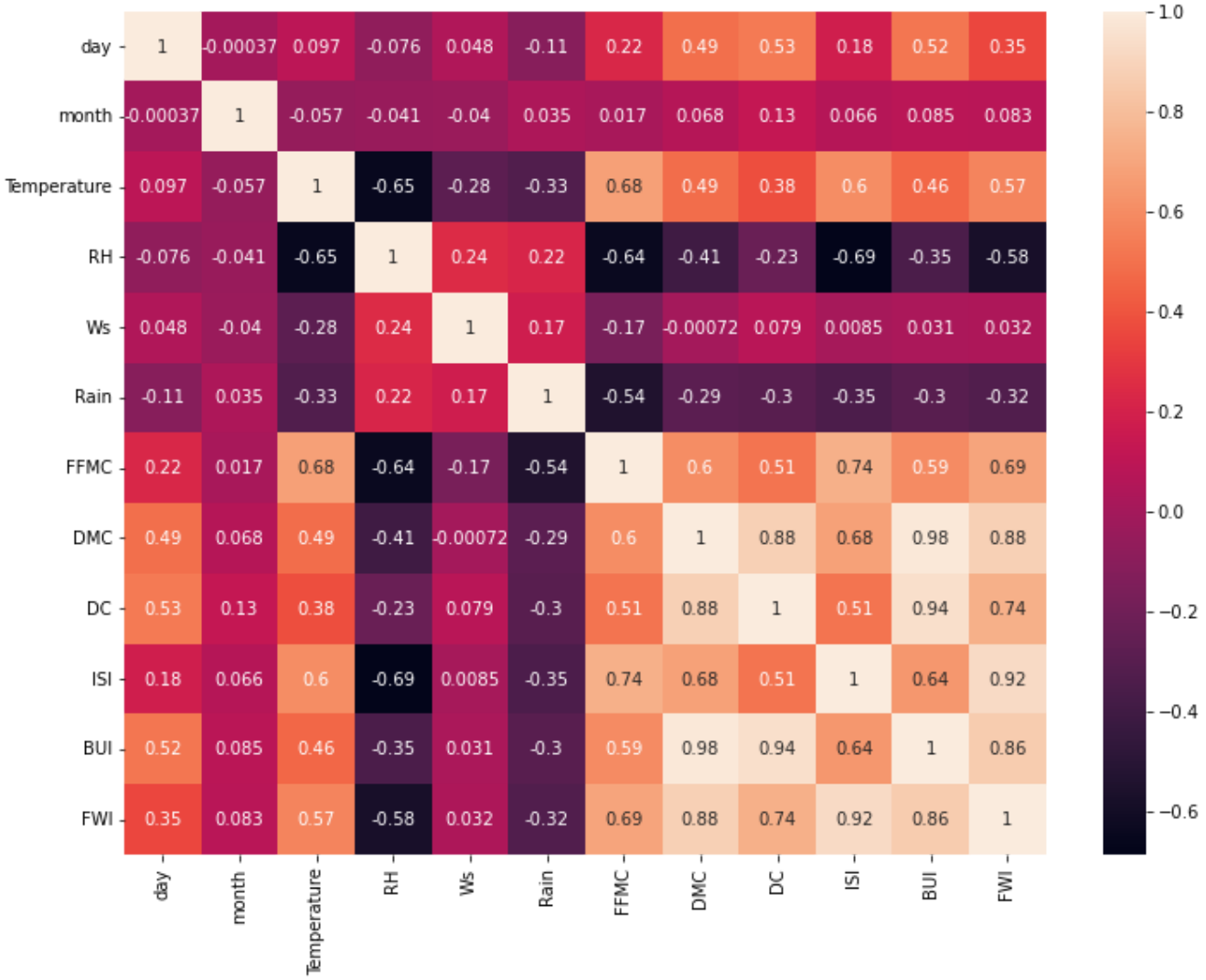
Out[65]:
```
<seaborn.axisgrid.PairGrid at 0x21c04cab370>
```



Display the Correlation Between the Features Ghapically

In [66]:
```
fig, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(df.corr(),annot=True)
```

Out[66]: <AxesSubplot:>



Observation:

- As Duff Moisture Code and Buildup Index are highly correlated there should be one feature since Duff Moisture Code is higly correlated with Temperature feature we can drop the Buildup Index

In [67]:
```
df.drop(labels=['BUI'],axis=1,inplace=True)
```

In [68]:
```
df.head()
```

Out[68]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	FWI	Classes
0	1	6	29	57	18	0.0	65.699997	3.4	7.6	1.3	0.5	not fire
1	2	6	29	61	13	1.3	64.400002	4.1	7.6	1.0	0.4	not fire
2	3	6	26	82	22	13.1	47.099998	2.5	7.1	0.3	0.1	not fire
3	4	6	25	89	13	2.5	28.600000	1.3	6.9	0.0	0.0	not fire
4	5	6	27	77	16	0.0	64.800003	3.0	14.2	1.2	0.5	not fire

Label Encoding

In [69]:
```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Classes'.
df['Classes']= label_encoder.fit_transform(df['Classes'])

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

Out[69]:
```
array([1, 0])
```

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

Out[70]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	FWI	Classes
0	1	6	29	57	18	0.0	65.699997	3.4	7.6	1.3	0.5	1
1	2	6	29	61	13	1.3	64.400002	4.1	7.6	1.0	0.4	1
2	3	6	26	82	22	13.1	47.099998	2.5	7.1	0.3	0.1	1
3	4	6	25	89	13	2.5	28.600000	1.3	6.9	0.0	0.0	1
4	5	6	27	77	16	0.0	64.800003	3.0	14.2	1.2	0.5	1

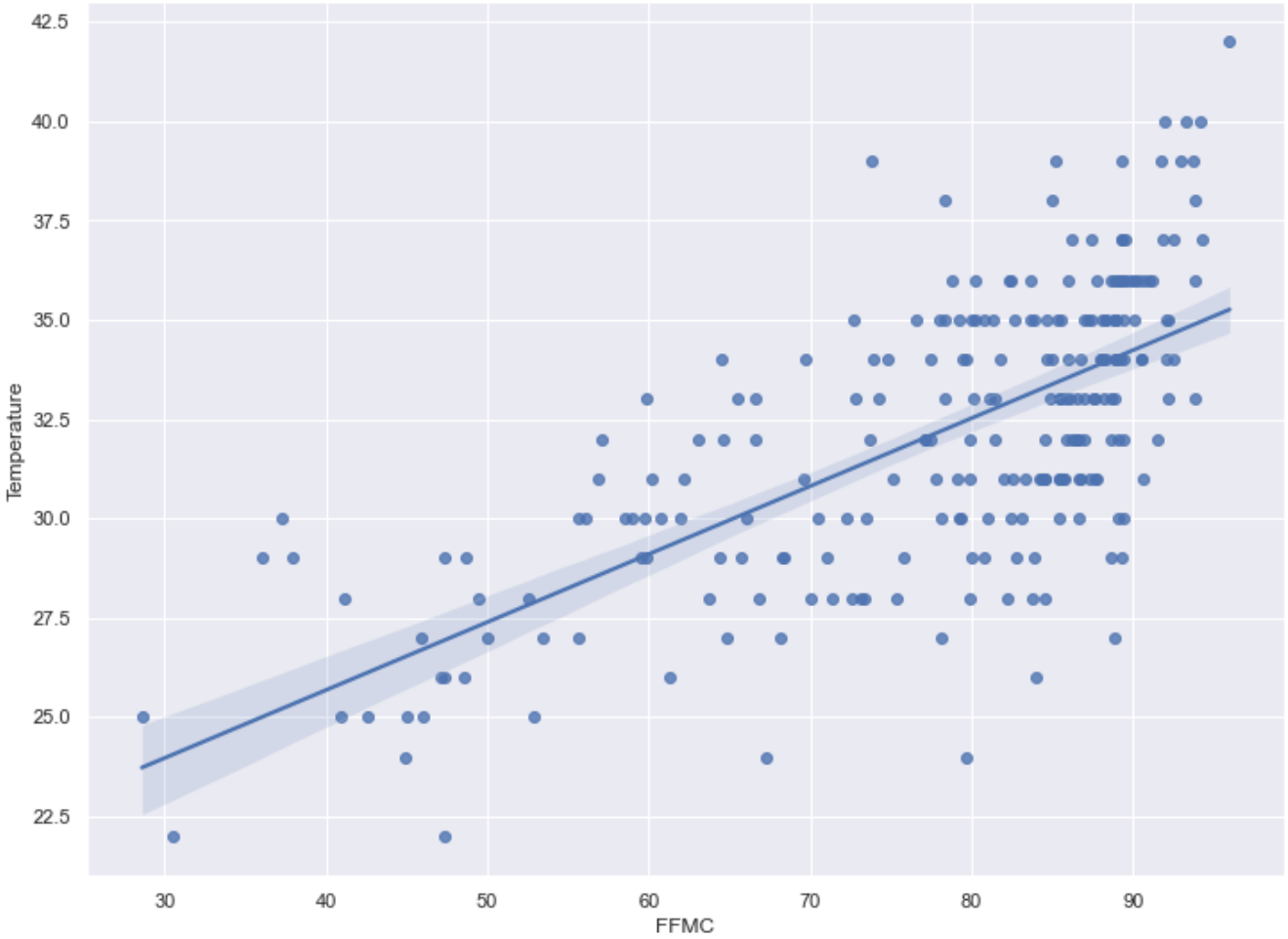
Observation:

-It changes all the Classes feature which has not fire to 1 and fire to 0

How Linearly The Temperature and Fine Fuel Moisture Code are Related

In [71]:
```
sns.set(rc={'figure.figsize':(12,9)})
sns.regplot(x='FFMC',y='Temperature',data=df)
```

Out[71]: <AxesSubplot:xlabel='FFMC', ylabel='Temperature'>



How Linearly The Temperature and Relative Humidity are Related

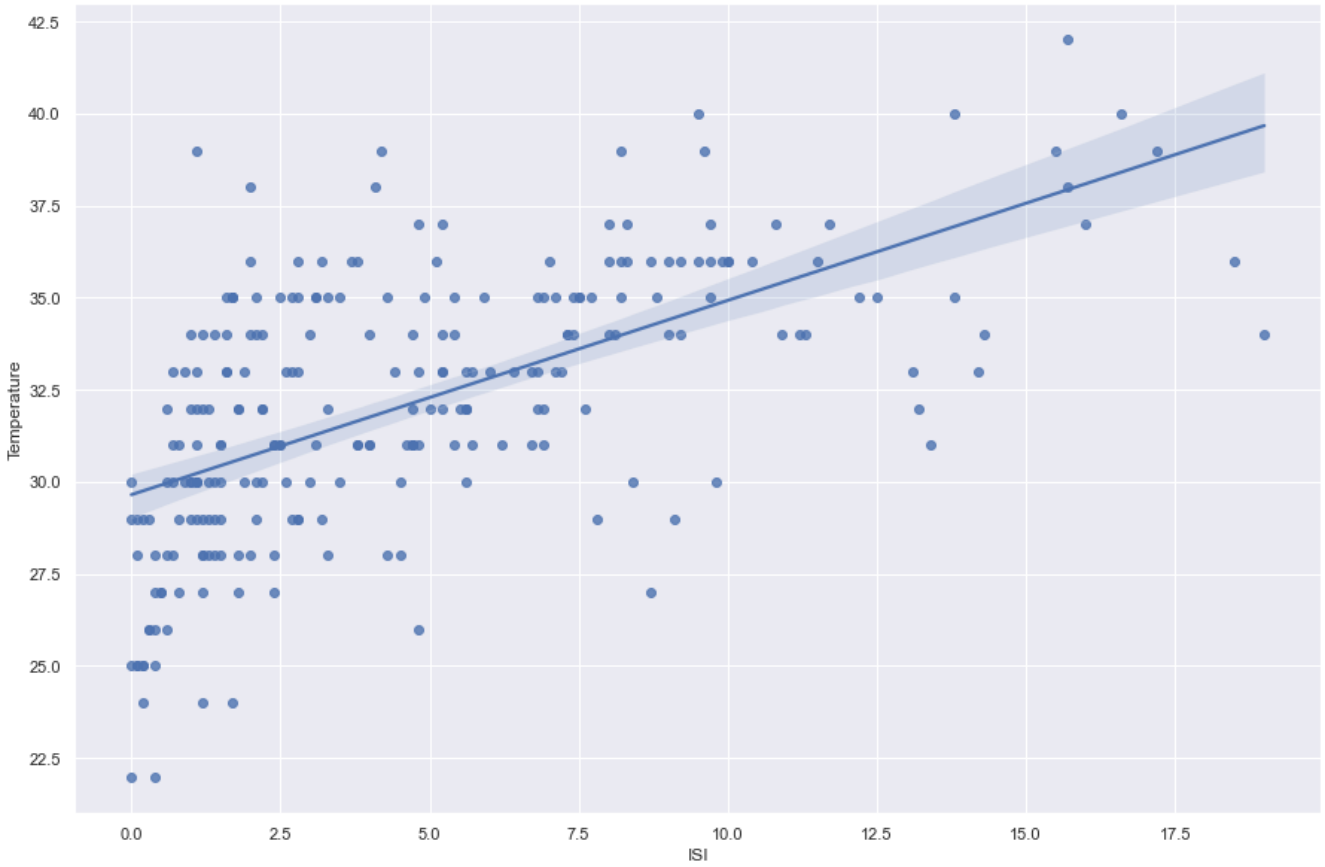
In [72]:
```
sns.set(rc={'figure.figsize':(12,9)})
sns.regplot(x='RH',y='Temperature',data=df)
```

Out[72]: <AxesSubplot:xlabel='RH', ylabel='Temperature'>

How Linearly The Temperature and Initial Spread Index are Related

In [73]:
```
sns.set(rc={'figure.figsize':(15,10)})
sns.regplot(x='ISI',y='Temperature',data=df)
```

Out[73]: <AxesSubplot:xlabel='ISI', ylabel='Temperature'>



In [74]:
```
sample_temp=df['Temperature']
```

In [75]:
```
df.drop(labels=['Temperature'],axis=1,inplace=True)
```

In [76]:
```
df['Temperature']=sample_temp
```

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

Out[77]:

	day	month	RH	Ws	Rain	FFMC	DMC	DC	ISI	FWI	Classes	Temperature
0	1	6	57	18	0.0	65.699997	3.4	7.6	1.3	0.5	1	29
1	2	6	61	13	1.3	64.400002	4.1	7.6	1.0	0.4	1	29
2	3	6	82	22	13.1	47.099998	2.5	7.1	0.3	0.1	1	26
3	4	6	89	13	2.5	28.600000	1.3	6.9	0.0	0.0	1	25
4	5	6	77	16	0.0	64.800003	3.0	14.2	1.2	0.5	1	27

Dependent and Independent Features

In [78]:
```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [79]:
```
df['Classes'].unique()
```

Out[79]:
```
array([1, 0])
```

In [80]:
```
X.head()
```

Out[80]:

	day	month	RH	Ws	Rain	FFMC	DMC	DC	ISI	FWI	Classes
0	1	6	57	18	0.0	65.699997	3.4	7.6	1.3	0.5	1
1	2	6	61	13	1.3	64.400002	4.1	7.6	1.0	0.4	1
2	3	6	82	22	13.1	47.099998	2.5	7.1	0.3	0.1	1
3	4	6	89	13	2.5	28.600000	1.3	6.9	0.0	0.0	1
4	5	6	77	16	0.0	64.800003	3.0	14.2	1.2	0.5	1

In [81]:
```
y.head()
```

Out[81]:
```
0    29
1    29
2    26
3    25
4    27
Name: Temperature, dtype: int32
```

Splitting the Dataset to Train and Test Feature

In [82]:
```
from sklearn.model_selection import train_test_split
```

In [83]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=11)
```

In [84]:
```
X_train.head()
```

Out[84]:

	day	month	RH	Ws	Rain	FFMC	DMC	DC	ISI	FWI	Classes
238	23	9	56	14	0.0	89.000000	29.400000	115.599998	7.5	15.2	0
233	18	9	33	13	0.1	90.599998	25.799999	77.800003	9.0	15.4	0
239	24	9	49	6	2.0	61.299999	11.900000	28.100000	0.6	0.4	1
198	14	8	40	13	0.0	91.900002	22.299999	55.500000	10.8	15.7	0
73	13	8	63	15	0.0	87.000000	19.000000	85.099998	5.9	10.2	0

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

Out[85]:
```
238    35
233    36
239    26
198    37
73     35
Name: Temperature, dtype: int32
```

In [86]:
```
print("X_train shape is ",X_train.shape)
print("y_train shape is ",y_train.shape)
```

```
X_train shape is  (162, 11)
y_train shape is  (162,)
```

In [87]:
```
print("X_test shape is ",X_test.shape)
print("y_test shape is ",y_test.shape)
```

```
X_test shape is  (81, 11)
y_test shape is  (81,)
```

Standardizing the Independent Features

In [88]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [89]:
```
X_train=scaler.fit_transform(X_train)
```

In [90]:
```
X_test=scaler.transform(X_test)
```

Model Training

In [91]:
```
from sklearn.linear_model import LinearRegression
```

In [92]:
```
regression=LinearRegression()
```

In [93]:
```
regression
```

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

In [94]:
```
regression.fit(X_train,y_train)
```

Out[94]:
```
LinearRegression()
```

In [95]:
```
## Print the Coefficients and the intercept
print(regression.coef_)
```

```
[-0.32525031 -0.55168719 -0.82572737 -0.44995213  0.25398265  1.28760797
  0.25418955  0.98869337  0.73195835 -0.72209454 -0.05773034]
```

In [96]:
```
print(regression.intercept_)
```

```
31.84567901234568
```

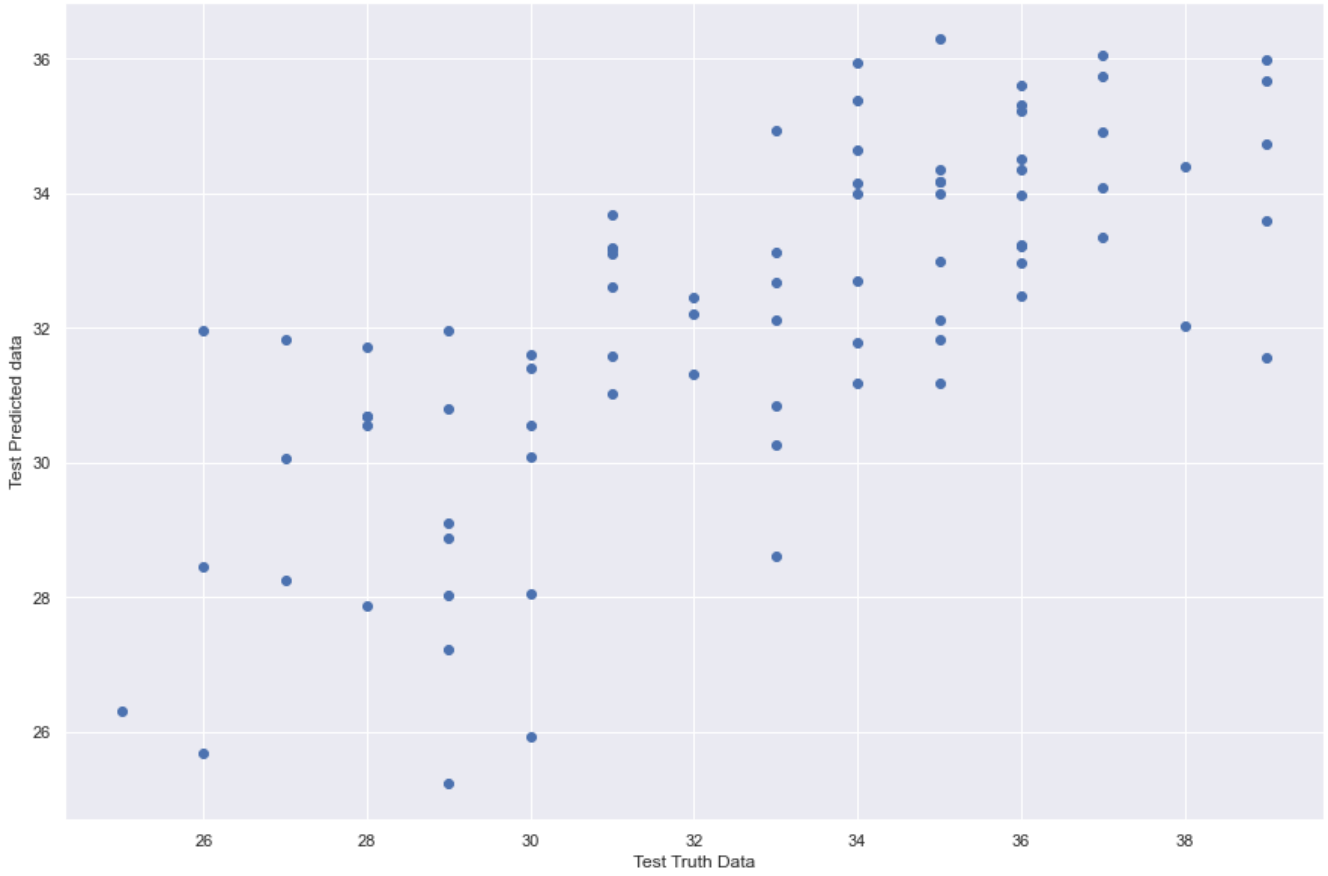
In [97]:
```
## Prediction for the test data
regression_pred=regression.predict(X_test)
```

In [98]:
```
regression_pred
```

Out[98]:
```
array([30.26515065, 31.00882578, 33.97216377, 34.50657382, 30.68970807,
       32.986536  , 34.35192609, 35.74002962, 33.98232054, 33.23717238,
       31.83157616, 34.63147665, 32.21228429, 34.16809151, 34.15479705,
       34.18021992, 30.56059095, 34.35811215, 32.11640042, 32.9586352 ,
       32.60279639, 35.21682143, 26.30350593, 32.03320562, 28.0180797 ,
       32.46962971, 31.17762763, 32.45781105, 28.25497111, 33.99194947,
       31.70289223, 33.67761215, 33.58479649, 31.5898657 , 28.05247447,
       31.96354957, 34.9390662 , 32.67067795, 36.2836806 , 27.22562961,
       30.05900032, 25.67319687, 30.844162  , 28.59614825, 33.18342925,
       25.24169735, 28.44700972, 27.87233434, 28.87002221, 25.92997604,
       35.65904973, 35.59652034, 30.79326573, 35.94306315, 30.54203391,
       33.10191871, 31.58229833, 33.35021373, 31.83475006, 31.39199406,
       35.38699347, 31.77089059, 30.08144496, 34.73222826, 31.55068042,
       34.07881214, 31.3204051 , 31.95762393, 34.91096036, 30.68004751,
       29.09815236, 36.03760377, 32.68787322, 35.31710951, 33.22035314,
       34.39110831, 32.11884417, 33.15635838, 35.98277233, 31.17914151,
       33.12988661])
```

Assumptions of Linear Regression

In [99]:
```
plt.scatter(y_test,regression_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted data")
plt.show()
```



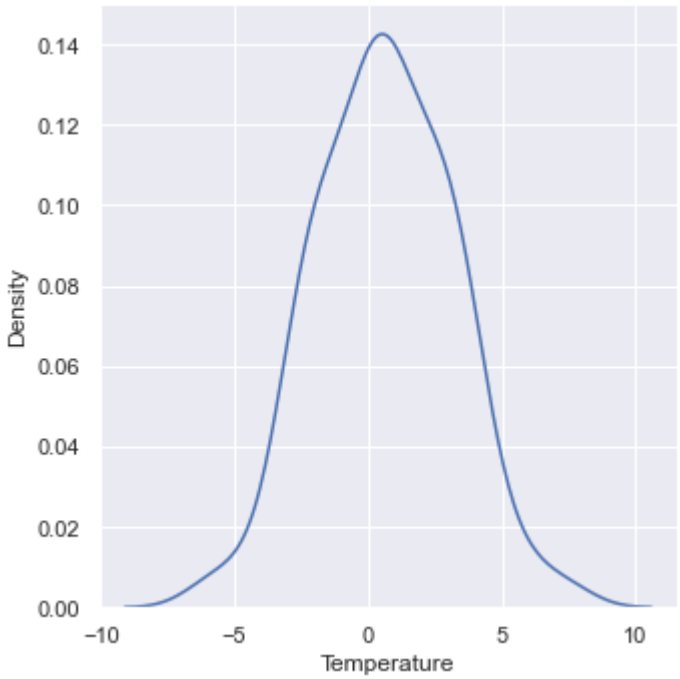
In [100]:
```
residuals=y_test-regression_pred
```

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

Out[101]:
```
39     2.734849
41    -0.008826
55     2.027836
165    1.493426
229   -2.689708
Name: Temperature, dtype: float64
```

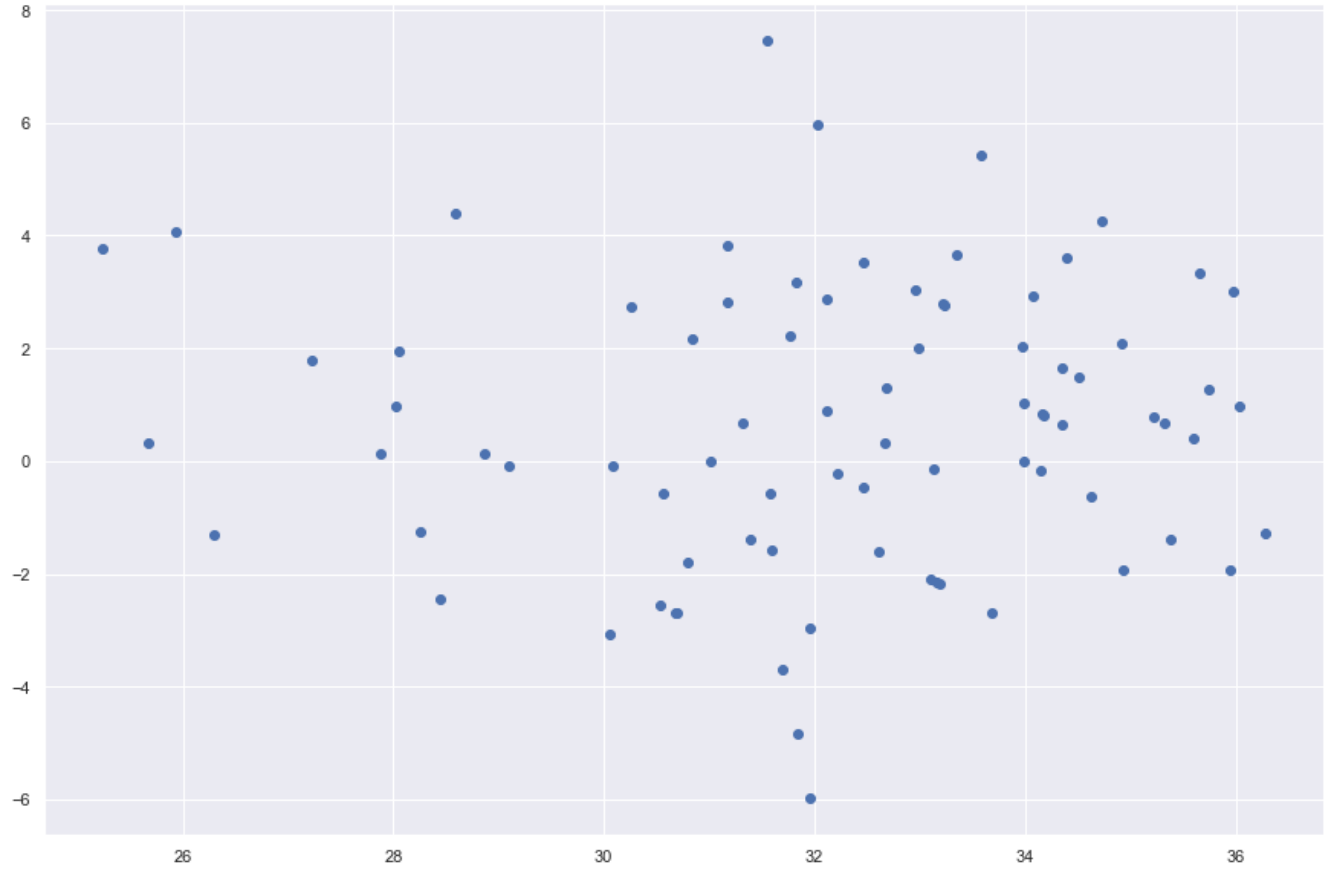
In [102]:
```
sns.displot(residuals,kind='kde')
```

Out[102]:
```
<seaborn.axisgrid.FacetGrid at 0x21c0d4dcc10>
```



In [103]:
```
plt.scatter(regression_pred,residuals)
```

Out[103]:
```
<matplotlib.collections.PathCollection at 0x21c0d8a64f0>
```



Calculating the Error

In [104]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,regression_pre
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,regression_p
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,r
```

```
The Mean Squared Error for the model is 6.651022710138245
The Mean Absolute Error for the model is 2.0663555576296475
The Root Mean Squared Error for the model is 2.5789576790126367
```

Performance Metrics of the Model

In [105]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,regression_pred)
print("The R2 Score for the model builded is",score)
```

```
The R2 Score for the model builded is 0.5014240665278434
```

In [106]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 0.4219409466989489
```

Ridge Regression

In [107]:
```
from sklearn.linear_model import Ridge
ridge=Ridge()
```

In [108]:
```
ridge.fit(X_train,y_train)
```

Out[108]:
```
Ridge()
```

In [109]:
```
ridge_pred=ridge.predict(X_test)
```

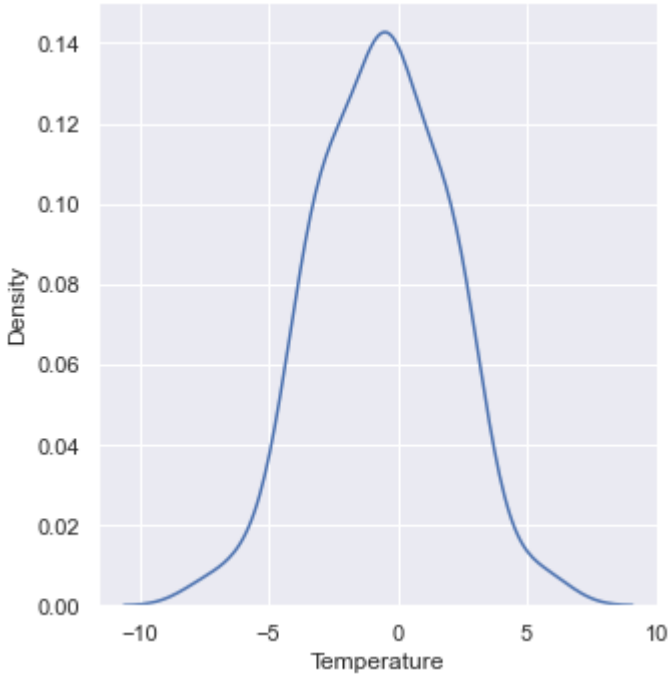
In [110]:
```
ridge_pred
```

Out[110]:
```
array([30.27356548, 30.98512337, 33.96056045, 34.50190218, 30.69818358,
       32.99330427, 34.39672248, 35.76929768, 33.9903573 , 33.26028422,
       31.80969456, 34.65959456, 32.20471906, 34.16037761, 34.16455094,
       34.17485684, 30.56374068, 34.37662224, 32.11440336, 32.96285589,
       32.58060596, 35.24585353, 26.34830107, 32.03104845, 28.05225902,
       32.47166113, 31.17793816, 32.43674164, 28.28460747, 34.00311992,
       31.65285399, 33.62550923, 33.60295552, 31.57666649, 28.08629548,
       31.95054515, 34.940029  , 32.66164398, 36.35578088, 27.26689435,
       30.02249231, 25.73115515, 30.81916108, 28.62802331, 33.2200882 ,
       25.31952026, 28.40007621, 27.8933483 , 28.88994955, 25.98964768,
       35.60150825, 35.65556054, 30.81227665, 36.03234329, 30.47859982,
       33.09941955, 31.55413994, 33.35591305, 31.82429976, 31.38202554,
       35.36968503, 31.75271506, 30.07992966, 34.74323182, 31.53419073,
       34.06292577, 31.33569948, 31.95705101, 34.94254239, 30.68320829,
       29.12267989, 36.07484378, 32.68418139, 35.316125  , 33.21917281,
       34.39232191, 32.06728127, 33.12098996, 35.95235572, 31.14998099,
       33.11007054])
```

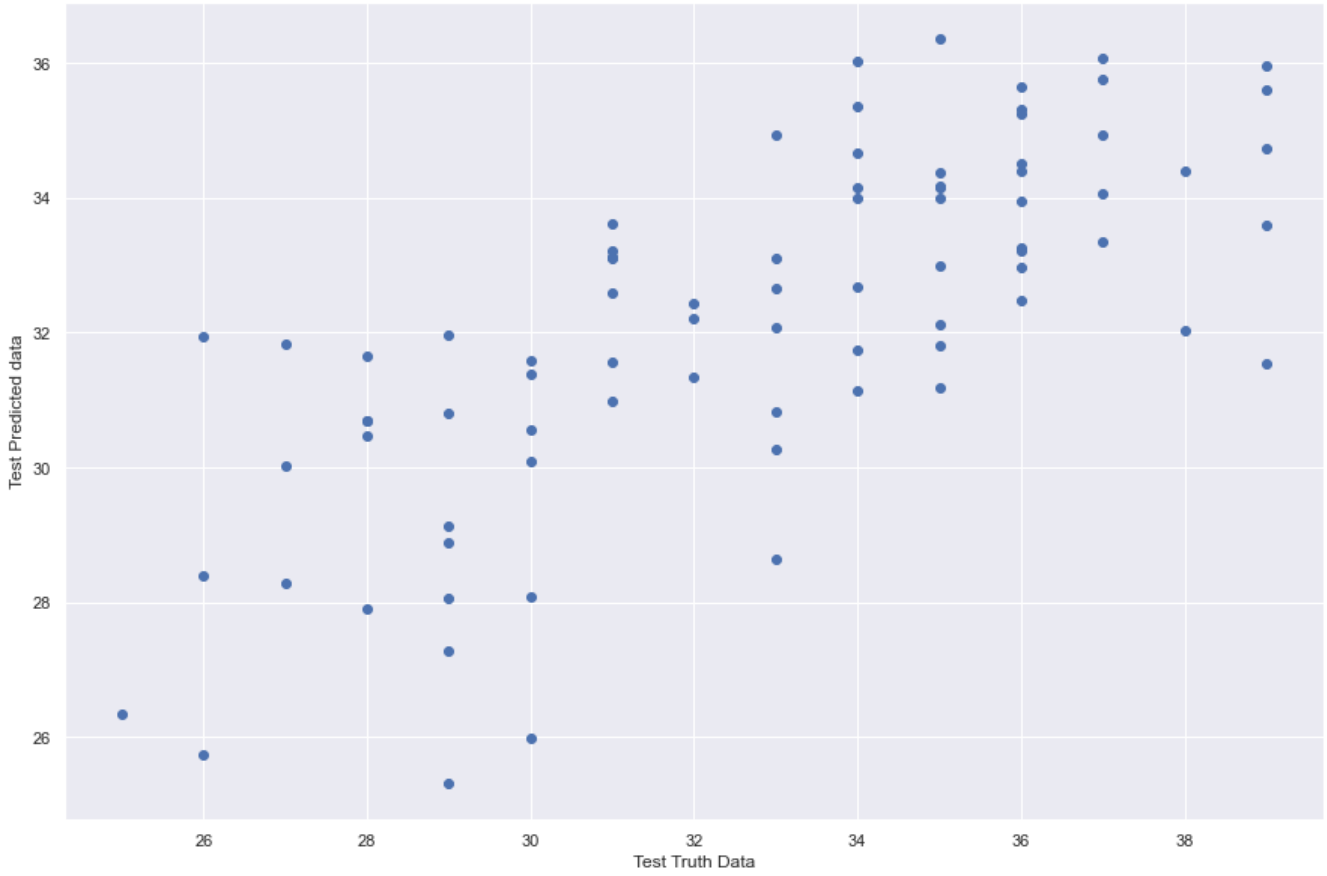
In [111]:
```
residual=ridge_pred-y_test
```

In [112]:
```
sns.displot(residual,kind='kde')
```

Out[112]:
```
<seaborn.axisgrid.FacetGrid at 0x21c0d8d4100>
```



In [113]:
```
plt.scatter(y_test,ridge_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted data")
plt.show()
```



Calculating the Error

In [114]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,ridge_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,ridge_pred))
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,
```

```
The Mean Squared Error for the model is 6.623472084700753
The Mean Absolute Error for the model is 2.0602110174343062
The Root Mean Squared Error for the model is 2.573610709625827
```

Performance Metrics

In [115]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,ridge_pred)
print("The R2 Score for the model builded is",score)
```

```
The R2 Score for the model builded is 0.5034893246684151
```

In [116]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 0.424335448890916
```

Lasso Regression

In [117]:
```
from sklearn.linear_model import Lasso
lasso=Lasso()
```

In [118]:
```
lasso.fit(X_train,y_train)
```

Out[118]:
```
Lasso()
```

In [119]:
```
lasso_pred=lasso.predict(X_test)
```

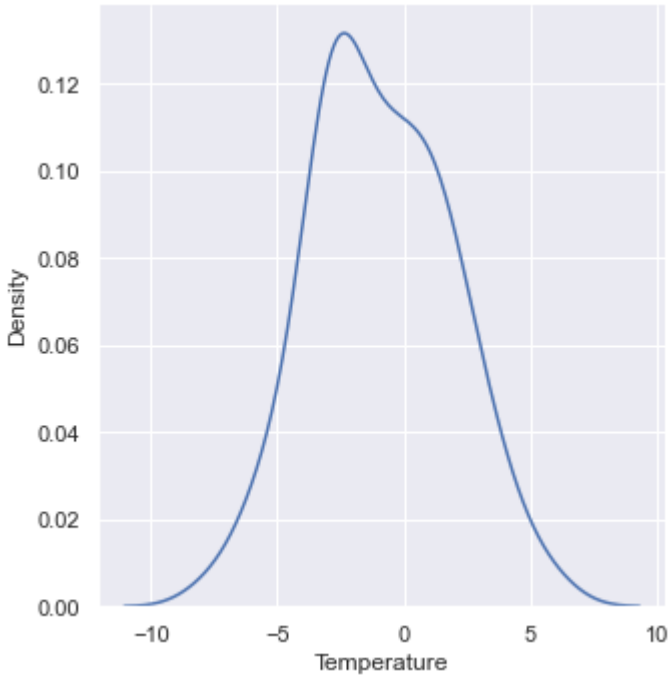
In [120]:
```
lasso_pred
```

Out[120]:
```
array([30.66221911, 31.22555026, 33.12098385, 33.33782614, 31.93070018,
       32.75366151, 33.27869614, 33.82474232, 32.69335456, 32.99505173,
       31.9295199 , 33.64251289, 32.10210686, 32.47762765, 33.10039985,
       32.91693133, 30.88686771, 33.44393743, 31.95049949, 32.70849243,
       32.31625281, 33.23971523, 27.02287838, 32.04826505, 29.95010242,
       32.26116504, 31.64011077, 31.75577774, 29.32347015, 33.40699698,
       30.97099365, 32.21029106, 32.84206539, 31.65859321, 30.18275653,
       31.89979734, 33.12031663, 32.19947976, 33.88132322, 28.62951057,
       30.88908738, 28.79966836, 31.57736497, 30.00753153, 32.53952522,
       27.8658602 , 28.72621105, 28.96527201, 30.71402963, 27.74325415,
       34.03609151, 33.65306912, 30.8793964 , 33.88904108, 30.7357015 ,
       32.59182834, 31.77000172, 32.77293451, 31.87630076, 31.35379406,
       32.93185331, 31.66122565, 29.65842251, 33.5361973 , 31.38261649,
       33.14418373, 31.95752607, 32.02088091, 33.18338089, 31.8349038 ,
       30.1304141 , 34.07974809, 32.42604777, 33.33360763, 32.37688954,
       32.88082826, 30.73995372, 32.11262312, 34.26044048, 31.66800569,
       32.28467754])
```

In [121]:
```
residual=lasso_pred-y_test
```

In [122]:
```
sns.displot(residual,kind='kde')
```

Out[122]: <seaborn.axisgrid.FacetGrid at 0x21c0f097e50>



Calculation the Error

In [123]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,lasso_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,lasso_pred))
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,
```

```
The Mean Squared Error for the model is 8.265102067478974
The Mean Absolute Error for the model is 2.3820826872896768
The Root Mean Squared Error for the model is 2.874909053775262
```

Performance Metrics

In [124]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,lasso_pred)
print("The R2 Score for the model builded is",score)
```

```
The R2 Score for the model builded is 0.38042897188508806
```

In [125]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 0.28165677899720354
```

Elastic Net Regression

In [126]:
```
from sklearn.linear_model import ElasticNet
```

In [127]:
```
elastic = ElasticNet(random_state=0)
```

In [128]:
```
elastic.fit(X_train,y_train)
```

Out[128]:
```
ElasticNet(random_state=0)
```

In [129]:
```
elastic_pred=elastic.predict(X_test)
```

In [130]:
```
elastic_pred
```

Out[130]:
```
array([30.32007329, 30.62849362, 33.44434918, 33.44559388, 31.60395251,
       32.76789206, 33.88217742, 34.5234838 , 33.21886391, 33.27182051,
       31.32321436, 34.2436066 , 31.88169277, 32.34655768, 33.44588338,
       32.87936362, 30.65260649, 33.75430321, 31.41332739, 32.68035549,
       32.24741161, 34.21345843, 27.63290233, 31.48134379, 29.715537  ,
       31.75659367, 31.0246927 , 31.18260669, 29.23899046, 33.69504347,
       30.47883983, 32.83177621, 32.79988727, 31.24890926, 29.80989008,
       31.61542413, 33.51619219, 31.75408187, 35.29538832, 28.63524758,
       30.60382831, 28.83784839, 30.68440236, 29.78478893, 33.15796974,
       28.12376044, 28.61876249, 28.91971221, 30.1411513 , 27.98833721,
       34.72538225, 34.8494712 , 30.5466458 , 35.4829652 , 30.36523242,
       32.63707799, 31.10475775, 32.66874449, 31.27404659, 31.11262502,
       33.94156321, 31.17736091, 29.55948724, 33.81517268, 31.04649904,
       33.45778954, 31.78279794, 31.73806541, 34.01686357, 31.3622155 ,
       29.83782929, 35.23320967, 32.04819372, 33.84195318, 32.81402498,
       32.9520279 , 30.5326578 , 32.34453875, 35.04514478, 30.89334041,
       32.44945411])
```

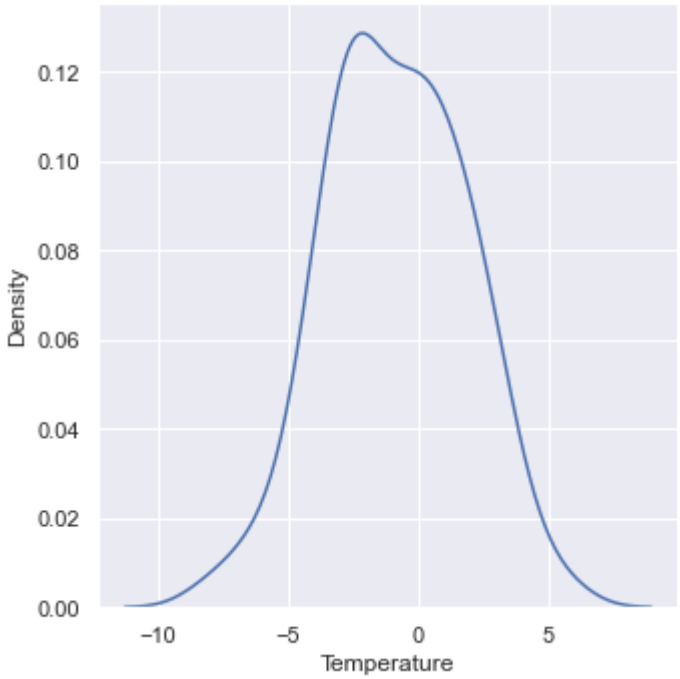
In [131]:
```
residual=elastic_pred-y_test
```

In [132]:
```
residual
```

Out[132]:
```
39    -2.679927
41    -0.371506
55    -2.555651
165   -2.554406
229    3.603953
          ...
141   -2.467342
59     1.344539
196   -3.954855
42    -3.106660
57    -0.550546
Name: Temperature, Length: 81, dtype: float64
```

In [133]:
```
sns.displot(residual,kind='kde')
```

Out[133]:
```
<seaborn.axisgrid.FacetGrid at 0x21c0f0ab400>
```



In [134]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print("The Mean Squared Error for the model is",mean_squared_error(y_test,elastic_pred))
print("The Mean Absolute Error for the model is",mean_absolute_error(y_test,elastic_pred
print("The Root Mean Squared Error for the model is",np.sqrt(mean_squared_error(y_test,e
```

```
The Mean Squared Error for the model is 7.829970048889066
The Mean Absolute Error for the model is 2.2888159309658063
The Root Mean Squared Error for the model is 2.7982083640946156
```

In [135]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,elastic_pred)
print("The R2 Score for the model builded is",score)
```

```
The R2 Score for the model builded is 0.4130474670860431
```

In [136]:
```
## Adjusted R square
Adjusted_r=1-(1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R Square for the model is",Adjusted_r)
```

```
The Adjusted R Square for the model is 0.31947532415773117
```