Algerian Forest Fire Dataset- Tempreture Prediction

1.Data Collection

2.Exploratory Data Analysis

3.Data Claning

4.Linear Regression

5.Ridge Regression Model Training

6.Lasso Regression Model Training

7.Elasticet Regresson MOdel Training

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

In [2]:
```
df=pd.read_csv(r'C:\Users\Mukul\Downloads\Algerian_forest_fires_dataset_UPDATE.cs
```

In [3]:
```
df
```

Out[3]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire
242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire
243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire

246 rows × 14 columns

In [4]:
```
df[121:130]
```

Out[4]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
123	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	
125	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	
126	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	
127	04	06	2012	30	64	14	0	79.4	5.2	15.4	2.2	5.6	1	
128	05	06	2012	32	60	14	0.2	77.1	6	17.6	1.8	6.5	0.9	
129	06	06	2012	35	54	11	0.1	83.7	8.4	26.3	3.1	9.3	3.1	

Drop the row

In [5]:
```
df.drop([122,123],inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1,inplace=True)
```

In [6]:
```
df[121:130]
```

Out[6]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	not fire
122	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	not fire
123	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	not fire
124	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	not fire
125	04	06	2012	30	64	14	0	79.4	5.2	15.4	2.2	5.6	1	not fire
126	05	06	2012	32	60	14	0.2	77.1	6	17.6	1.8	6.5	0.9	not fire
127	06	06	2012	35	54	11	0.1	83.7	8.4	26.3	3.1	9.3	3.1	fire
128	07	06	2012	35	44	17	0.2	85.6	9.9	28.9	5.4	10.7	6	fire
129	08	06	2012	28	51	17	1.3	71.4	7.7	7.4	1.5	7.3	0.8	not fire

In [7]:
```
df.loc[:122,'region']='bejaia'
df.loc[122:,'region']='Sidi-Bel Abbes'
```

Stripping the name of the columns

In [8]:
```
df.columns=[i.strip() for i in df.columns]
df.columns
```

Out[8]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'region'],
      dtype='object')
```

In [9]:
```
df.drop('Classes',axis=1,inplace=True)
df.head()
```

Out[9]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	bejaia
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	bejaia
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	bejaia
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	bejaia
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	bejaia

In [10]:
```
df['date']=pd.to_datetime(df[['day','month','year']])
df.drop(['day','month','year'],axis=1,inplace=True)
```

In [11]:
```
df.head()
```

Out[11]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region	date
0	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	bejaia	2012-06-01
1	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	bejaia	2012-06-02
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	bejaia	2012-06-03
3	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	bejaia	2012-06-04
4	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	bejaia	2012-06-05

In [12]:
```
df.dtypes
```

Out[12]:
```
Temperature            object
RH                     object
Ws                     object
Rain                   object
FFMC                   object
DMC                    object
DC                     object
ISI                    object
BUI                    object
FWI                    object
region                 object
date           datetime64[ns]
dtype: object
```

In [13]:
```
df['Temperature']=df['Temperature'].astype(int)
df['RH']=df['RH'].astype(int)
df['Ws']=df['Ws'].astype(int)
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)
```

In [14]:
```
df.dtypes
```

Out[14]:
```
Temperature             int32
RH                      int32
Ws                      int32
Rain                  float64
FFMC                  float64
DMC                   float64
DC                     object
ISI                   float64
BUI                   float64
FWI                    object
region                 object
date           datetime64[ns]
dtype: object
```

Applying Labelencoding in DC,FWI,region features

In [15]:
```
from sklearn.preprocessing import LabelEncoder
LabelEncoder=LabelEncoder()
```

In [16]:
```
df['DC']=LabelEncoder.fit_transform(df['DC'])
df['FWI']=LabelEncoder.fit_transform(df['FWI'])
df['region']=LabelEncoder.fit_transform(df['region'])
```

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

Out[17]:
```
Temperature             int32
RH                      int32
Ws                      int32
Rain                  float64
FFMC                  float64
DMC                   float64
DC                      int32
ISI                   float64
BUI                   float64
FWI                     int32
region                  int32
date           datetime64[ns]
dtype: object
```

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

Out[18]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region	date
0	29	57	18	0.0	65.7	3.4	150	1.3	3.4	5	1	2012-06-01
1	29	61	13	1.3	64.4	4.1	150	1.0	3.9	4	1	2012-06-02
2	26	82	22	13.1	47.1	2.5	146	0.3	2.7	1	1	2012-06-03
3	25	89	13	2.5	28.6	1.3	136	0.0	1.7	0	1	2012-06-04
4	27	77	16	0.0	64.8	3.0	18	1.2	3.9	5	1	2012-06-05

In [19]:
```
# checking null values
df.isnull().sum()
```

Out[19]:
```
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
region         0
date           0
dtype: int64
```

Univariate Aanlysis

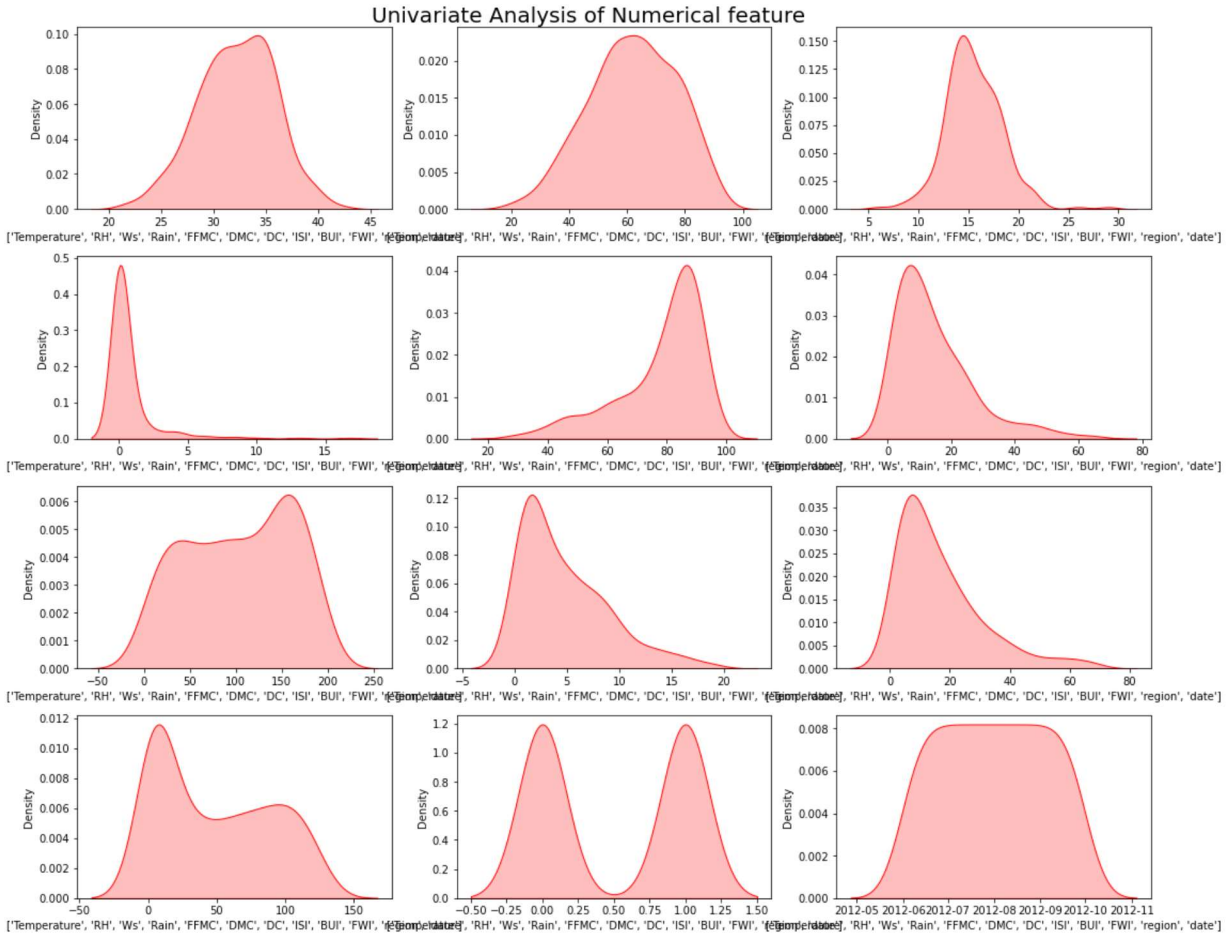
In [20]:
```
num_column=[feature for feature in df.columns if df[feature].dtypes!='object']
```

In [21]:
```
num_column
```

Out[21]:
```
['Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'region',
 'date']
```

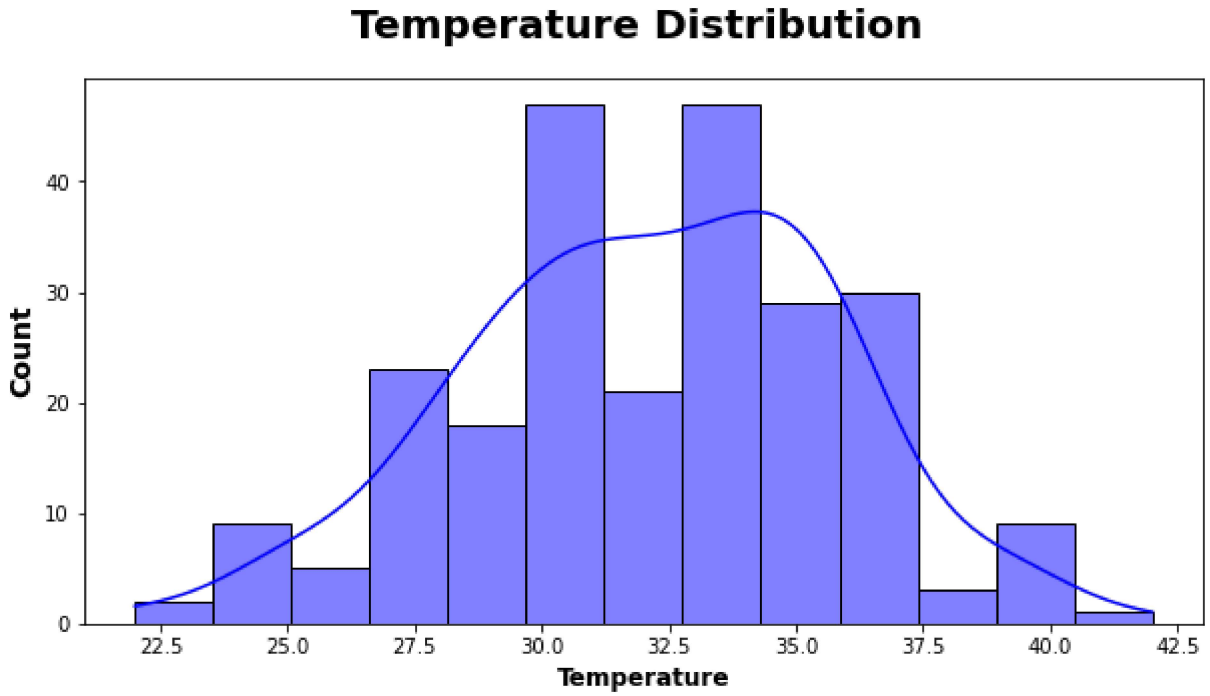
In [22]:
```
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis of Numerical feature',fontsize=20,fontweight=10

for i in range(0,len(num_column)):
    plt.subplot(5,3,i+1)
    sns.kdeplot(x=df[num_column[i]],shade=True,color='r')
    plt.xlabel(num_column)
    plt.tight_layout()
```



Visualization of Target Feature

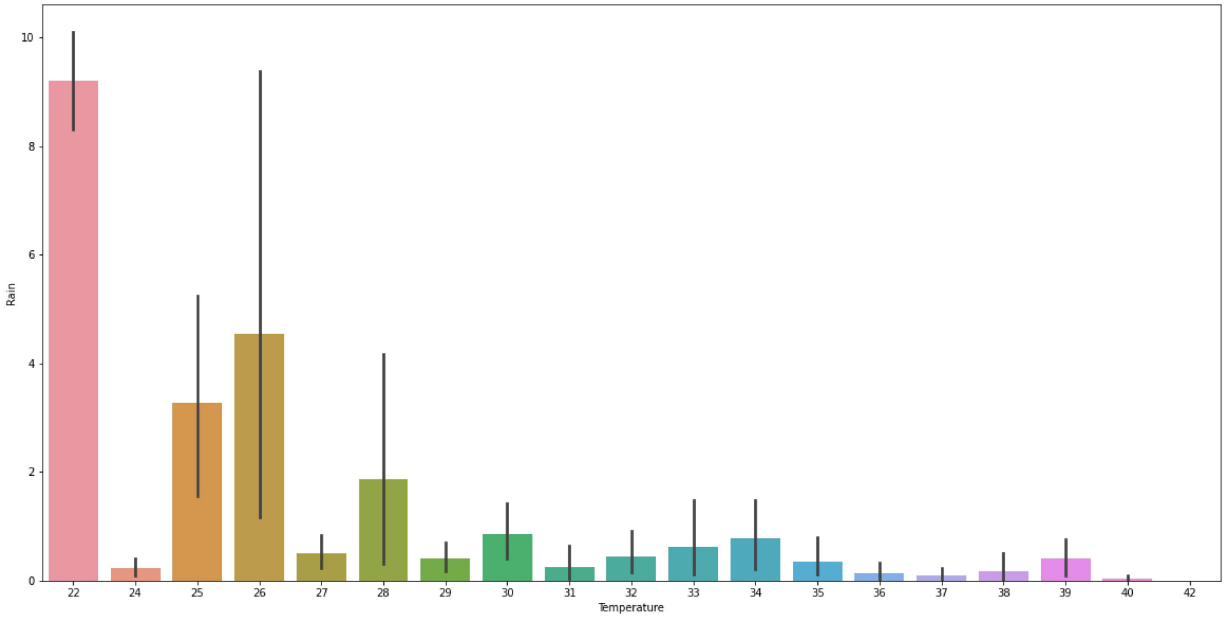
In [23]:
```
plt.subplots(figsize=(10,5))
sns.histplot(x=df.Temperature,ec='Black',color='blue',kde=True)
plt.title('Temperature Distribution ', weight='bold',fontsize=20,pad=20)
plt.ylabel('Count',weight='bold',fontsize=14)
plt.xlabel('Temperature',weight='bold',fontsize=12)
plt.show()
```



Temperature Vs Rain

In [24]:
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)
sns.barplot(x='Temperature',y='Rain',data=df)
```

Out[24]: <AxesSubplot:xlabel='Temperature', ylabel='Rain'>



Correlation of the features

In [25]:
```
df.corr()
```

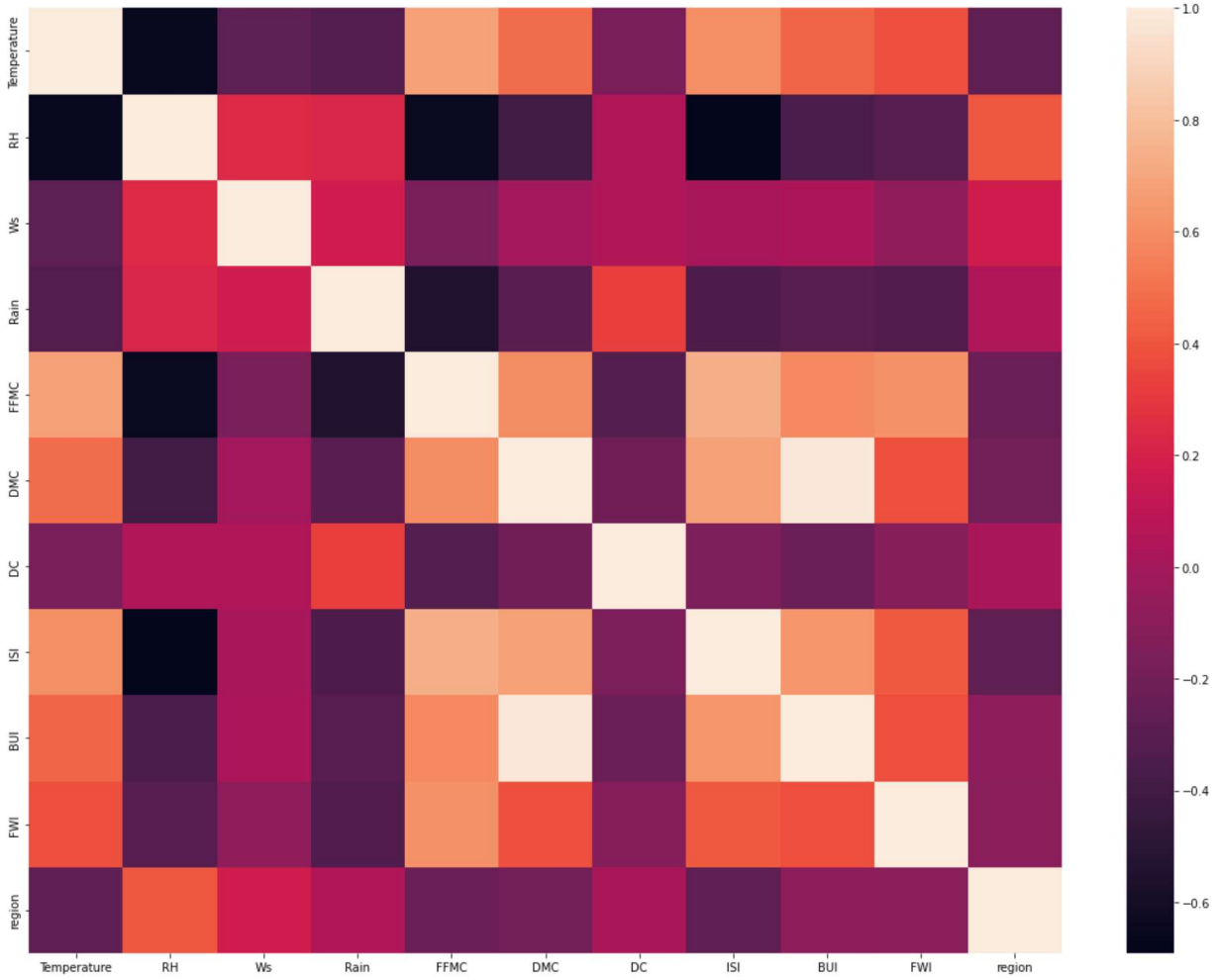
Out[25]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	I
Temperature	1.000000	-0.654443	-0.278132	-0.326786	0.677491	0.483105	-0.165840	0.6075
RH	-0.654443	1.000000	0.236084	0.222968	-0.645658	-0.405133	0.041651	-0.6906
Ws	-0.278132	0.236084	1.000000	0.170169	-0.163255	-0.001246	0.040958	0.0152
Rain	-0.326786	0.222968	0.170169	1.000000	-0.544045	-0.288548	0.324748	-0.3471
FFMC	0.677491	-0.645658	-0.163255	-0.544045	1.000000	0.602391	-0.319086	0.7397
DMC	0.483105	-0.405133	-0.001246	-0.288548	0.602391	1.000000	-0.200609	0.6744
DC	-0.165840	0.041651	0.040958	0.324748	-0.319086	-0.200609	1.000000	-0.1527
ISI	0.607551	-0.690637	0.015248	-0.347105	0.739730	0.674499	-0.152717	1.0000
BUI	0.455504	-0.348587	0.029756	-0.299171	0.589652	0.982073	-0.226445	0.6358
FWI	0.380581	-0.295093	-0.081447	-0.340412	0.617445	0.384628	-0.118684	0.4125
region	-0.273496	0.406424	0.176829	0.041080	-0.224680	-0.191094	0.016293	-0.2684

Multivariate analysis

In [26]:
```
import seaborn as sns
plt.figure(figsize=(20,15))
sns.heatmap(df.corr())
```

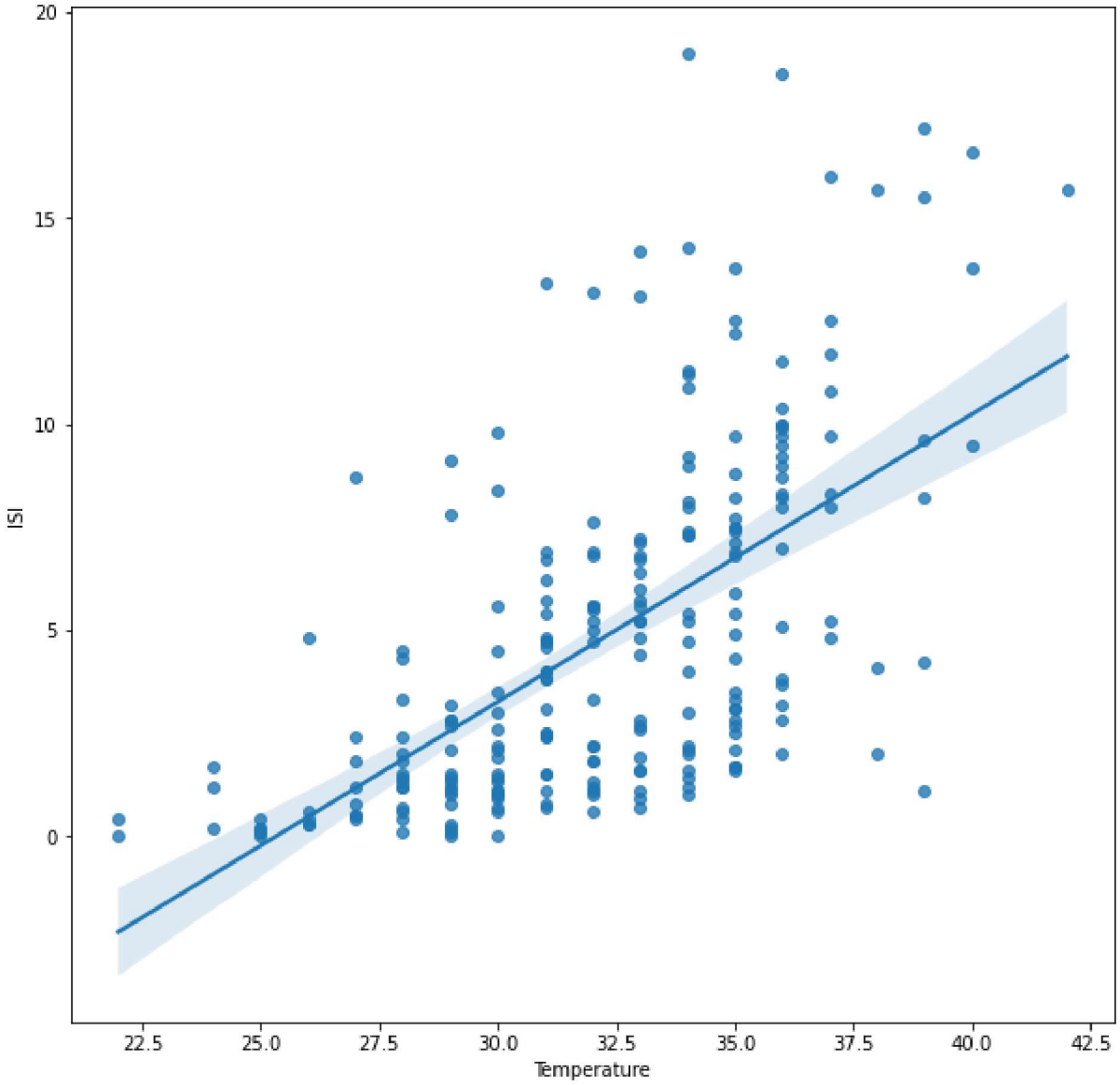
Out[26]: <AxesSubplot:>



Temperature Vs ISI

In [27]:
```
plt.figure(figsize=(10,10))
sns.regplot(x='Temperature',y='ISI',data=df)
```

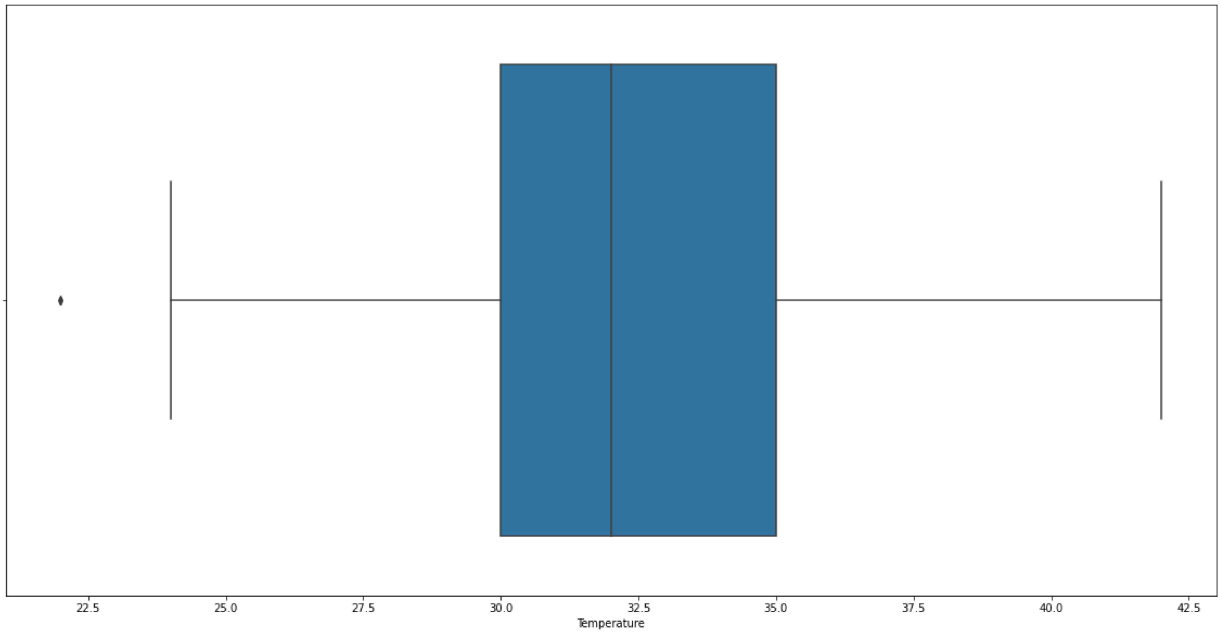
Out[27]: <AxesSubplot:xlabel='Temperature', ylabel='ISI'>



Checking the Outlier of the Temperature feature

In [28]:
```
sns.boxplot(df['Temperature'])
```

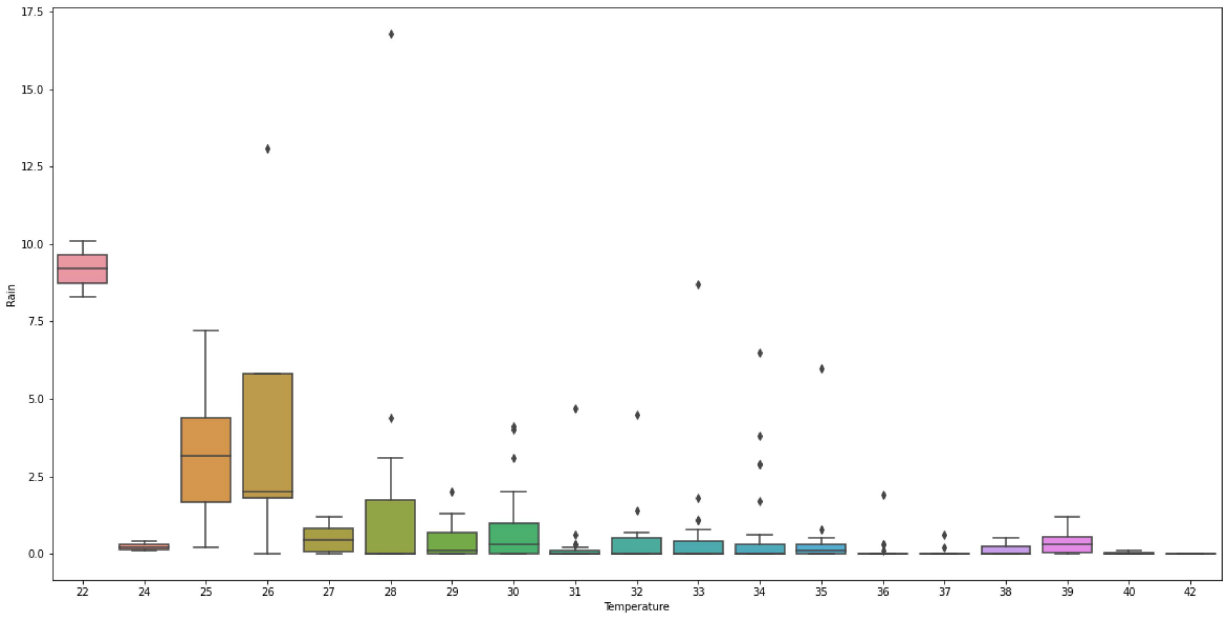
Out[28]:
```
<AxesSubplot:xlabel='Temperature'>
```



Boxplot of Rain VS Temperature

In [29]:
```
sns.boxplot(x='Temperature',y='Rain',data=df)
```

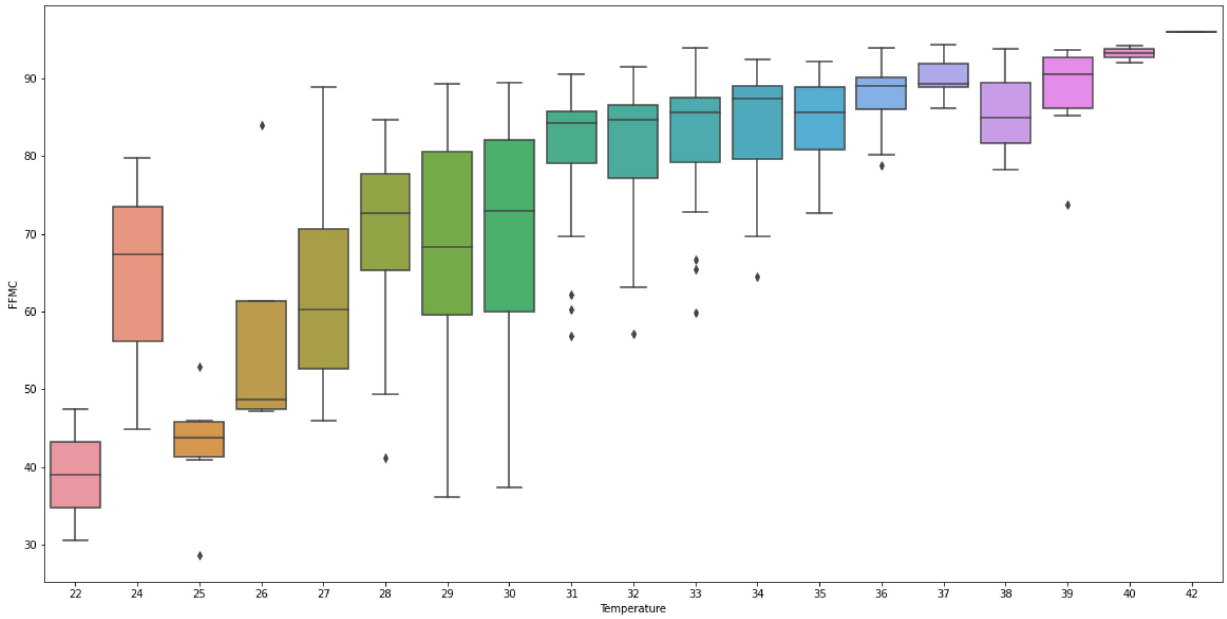
Out[29]:
```
<AxesSubplot:xlabel='Temperature', ylabel='Rain'>
```



FFMC VS Temperature

In [30]:
```
sns.boxplot(x='Temperature',y='FFMC',data=df)
```

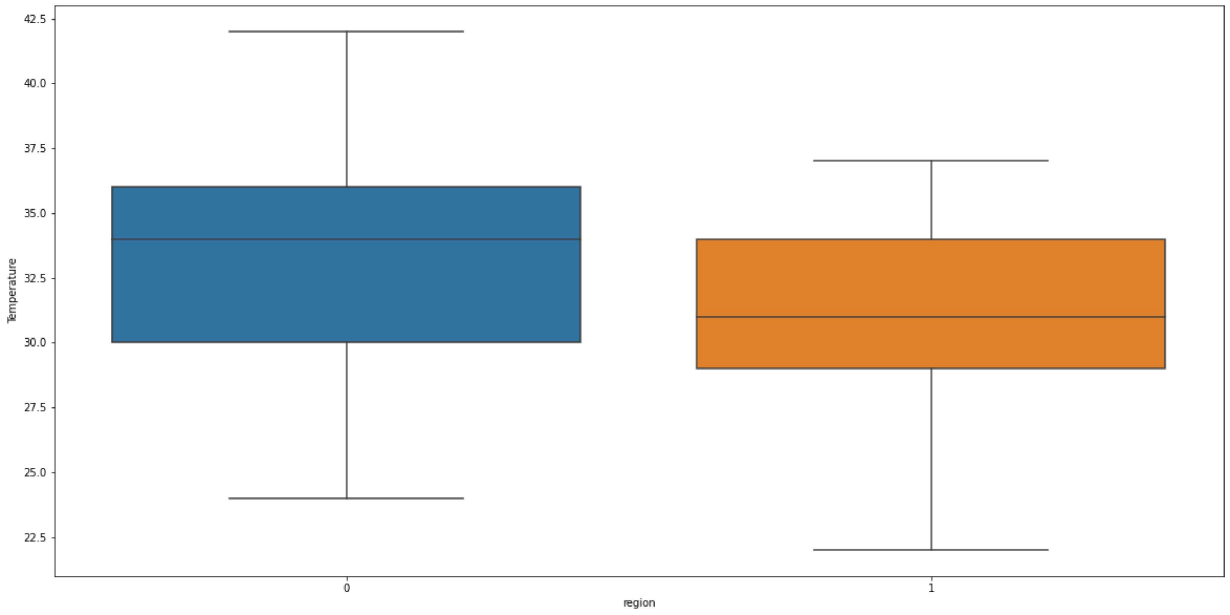
Out[30]:
```
<AxesSubplot:xlabel='Temperature', ylabel='FFMC'>
```



Region VS Temperature

In [31]:
```
sns.boxplot(x='region',y='Temperature',data=df)
```

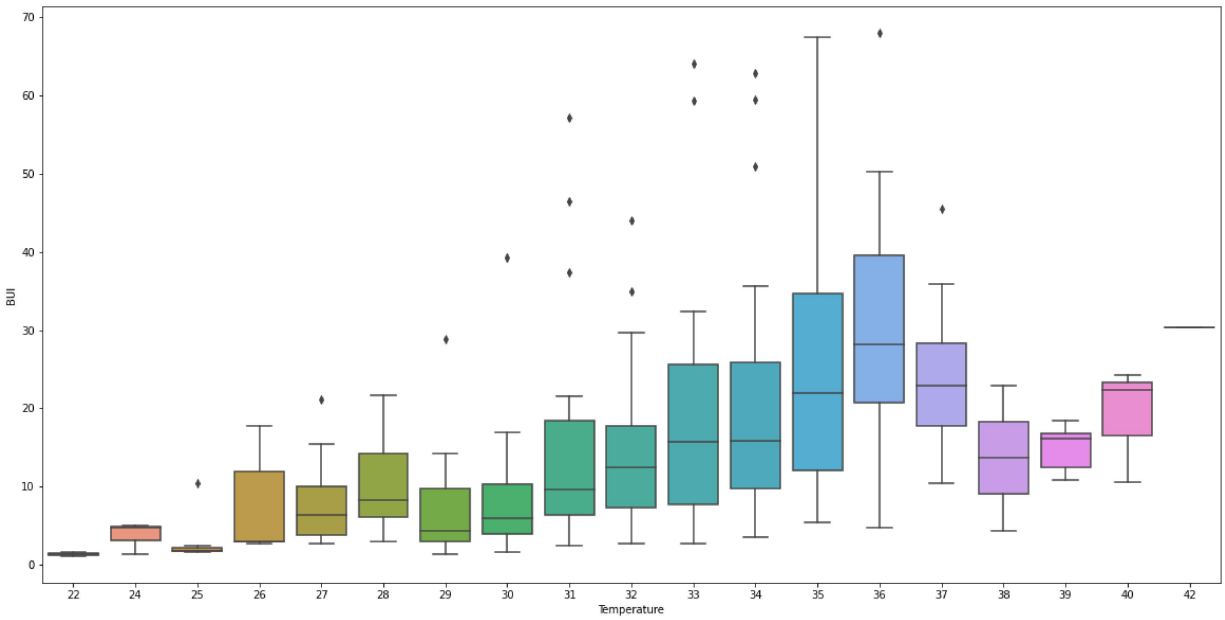
Out[31]:
```
<AxesSubplot:xlabel='region', ylabel='Temperature'>
```



BUI VS Temperature

In [32]:
```
sns.boxplot(x='Temperature', y='BUI',data=df)
```

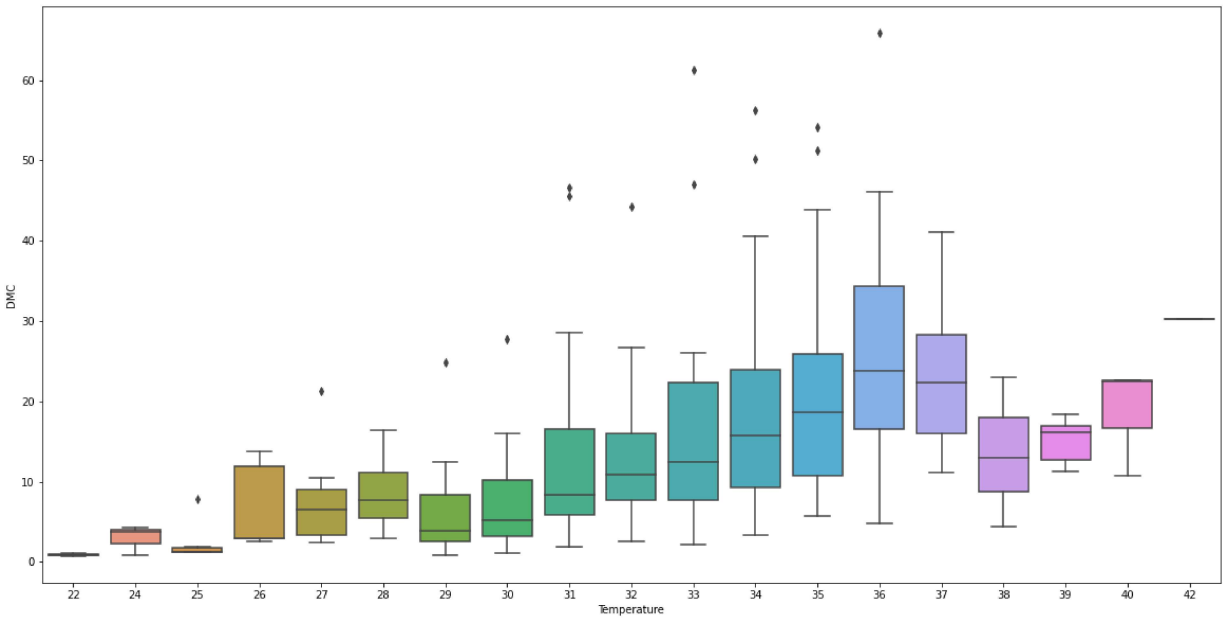
Out[32]:
```
<AxesSubplot:xlabel='Temperature', ylabel='BUI'>
```



DMC VS Temperature

In [33]:
```
sns.boxplot(x='Temperature' , y = 'DMC' , data=df)
```

Out[33]:
```
<AxesSubplot:xlabel='Temperature', ylabel='DMC'>
```



Creating Dependent and Independent Feature

In [34]:
```
df.columns
```

Out[34]:
```
Index(['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
       'FWI', 'region', 'date'],
      dtype='object')
```

In [35]:
```
#independent Feature
x=pd.DataFrame(df,columns=['RH','Ws','Rain','FFMC' ,'DMC' ,'DC','ISI','BUI','FWI'
```

In [36]:
```
#dependent Feature
y=pd.DataFrame(df,columns=['Temperature'])
```

In [37]:
```
x    #independennt feature
```

Out[37]:

	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region
0	57	18	0.0	65.7	3.4	150	1.3	3.4	5	1
1	61	13	1.3	64.4	4.1	150	1.0	3.9	4	1
2	82	22	13.1	47.1	2.5	146	0.3	2.7	1	1
3	89	13	2.5	28.6	1.3	136	0.0	1.7	0	1
4	77	16	0.0	64.8	3.0	18	1.2	3.9	5	1
...	...	...	...	...	...	...	...	...	...	...
239	65	14	0.0	85.4	16.0	112	4.5	16.9	106	0
240	87	15	4.4	41.1	6.5	164	0.1	6.2	0	0
241	87	29	0.5	45.9	3.5	153	0.4	3.4	2	0
242	54	18	0.1	79.7	4.3	25	1.7	5.1	7	0
243	64	15	0.2	67.3	3.8	34	1.2	4.8	5	0

244 rows × 10 columns

In [38]:
```
y   #dependent feature
```

Out[38]:

	Temperature
0	29
1	29
2	26
3	25
4	27
...	...
239	30
240	28
241	27
242	24
243	24

244 rows × 1 columns

Train_Test_Split

In [39]:
```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=10
```

In [40]:
```
x_train.shape
```

Out[40]: (163, 10)

In [41]:
```
x_test.shape
```

Out[41]: (81, 10)

In [42]:
```
y_train.shape
```

Out[42]: (163, 1)

In [43]:
```
y_test.shape
```

Out[43]: (81, 1)

In [44]:
```
#Independent training dataset
x_train
```

Out[44]:

	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region
237	49	6	2.0	61.3	11.9	77	0.6	11.9	4	0
78	54	18	0.0	89.4	20.0	8	9.7	27.5	47	1
25	64	18	0.0	86.8	17.8	157	6.7	21.6	20	1
124	80	14	2.0	48.7	2.2	150	0.3	2.6	1	0
176	64	9	1.2	73.8	11.7	28	1.1	11.4	7	0
...	...	...	...	...	...	...	...	...	...	...
64	69	13	0.0	85.0	8.2	53	4.0	8.2	86	1
15	89	13	0.7	36.1	1.7	150	0.0	2.2	0	1
228	51	13	0.0	88.7	16.0	122	6.9	17.8	124	0
125	64	14	0.0	79.4	5.2	26	2.2	5.6	10	0
9	79	12	0.0	73.2	9.5	114	1.3	12.6	9	1

163 rows × 10 columns

In [45]:
```
#Independent test dataset
x_test
```

Out[45]:

	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region
162	56	15	2.9	74.8	7.1	185	1.6	6.8	8	0
60	64	17	0.0	87.2	31.9	22	6.8	41.2	45	1
61	45	14	0.0	78.8	4.8	1	2.0	4.7	9	1
63	63	14	0.3	76.6	5.7	0	1.7	5.5	8	1
69	59	17	0.0	87.4	14.8	132	6.9	17.9	125	1
...	...	...	...	...	...	...	...	...	...	...
169	68	15	0.0	86.1	23.9	123	5.2	23.9	120	0
232	41	8	0.1	83.9	24.9	177	2.7	28.9	99	0
144	59	16	0.8	74.2	7.0	166	1.6	6.7	8	0
208	37	16	0.0	92.2	61.3	38	13.1	64.0	89	0
105	76	26	8.3	47.4	1.1	145	0.4	1.6	1	1

81 rows × 10 columns

Dependent Traing dataset

In [46]:
```
y_train
```

Out[46]:

	Temperature
237	26
78	36
25	31
124	29
176	39
...	...
64	34
15	29
228	32
125	30
9	28

163 rows × 1 columns

In [47]:
```
y_test
```

Out[47]:

	Temperature
162	34
60	35
61	36
63	35
69	35
...	...
169	33
232	29
144	33
208	33
105	22

81 rows × 1 columns

Standardization or Feature Scaling

In [48]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [49]:
```
scaler
```

Out[49]:
```
StandardScaler()
```

In [50]:
```
x_train=scaler.fit_transform(x_train)
```

In [51]:
```
x_test=scaler.transform(x_test)
```

In [52]:
```
x_train
```

Out[52]:
```
array([[-0.85631108, -3.36419461,  0.88853946, ..., -0.32535487,
        -1.03738328, -0.98176139],
       [-0.52508491,  0.99944243, -0.441414  , ...,  0.76565444,
        -0.01141751,  1.01857744],
       [ 0.13736742,  0.99944243, -0.441414  , ...,  0.35302912,
        -0.65562857,  1.01857744],
       ...,
       [-0.72382061, -0.81873967, -0.441414  , ...,  0.08727045,
         1.825777  , -0.98176139],
       [ 0.13736742, -0.45510325, -0.441414  , ..., -0.76595478,
        -0.89422526, -0.98176139],
       [ 1.13104591, -1.18237609, -0.441414  , ..., -0.27639932,
        -0.91808493,  1.01857744]])
```

In [53]:
```
x_test
```

Out[53]:
```
array([[-3.92594448e-01, -9.14668296e-02,  1.48701853e+00,
        -1.82411230e-01, -6.02677495e-01,  1.41938909e+00,
        -7.33442383e-01, -6.82030988e-01, -9.41944600e-01,
        -9.81761387e-01],
       [ 1.37367416e-01,  6.35806011e-01, -4.41414004e-01,
         6.64566895e-01,  1.37979749e+00, -1.44016117e+00,
         4.94418103e-01,  1.72378441e+00, -5.91368483e-02,
         1.01857744e+00],
       [-1.12129201e+00, -4.55103250e-01, -4.41414004e-01,
         9.08075201e-02, -7.86536062e-01, -1.80856948e+00,
        -6.38991577e-01, -8.28897625e-01, -9.18084931e-01,
         1.01857744e+00],
       [ 7.11221826e-02, -4.55103250e-01, -2.41920984e-01,
        -5.94627923e-02, -7.14591405e-01, -1.82611273e+00,
        -7.09829682e-01, -7.72948430e-01, -9.41944600e-01,
         1.01857744e+00],
       [-1.93858749e-01,  6.35806011e-01, -4.41414004e-01,
         6.78227832e-01,  1.28490116e-02,  4.89596678e-01,
         5.18030804e-01,  9.42640966e-02,  1.84963667e+00,
```

Model Training

In [54]:
```
from sklearn.linear_model import LinearRegression
regression =LinearRegression()
```

In [55]:
```
regression
```

Out[55]:
```
LinearRegression()
```

In [56]:
```
regression.fit(x_train,y_train)
```

Out[56]:
```
LinearRegression()
```

In [57]:
```
# coefficient
print(regression.coef_)
```

```
[[-1.27500995 -0.53842199 -0.21205266  0.70886534 -1.02729123 -0.32455869
   0.2501139   1.35400654  0.21687466 -0.23115864]]
```

In [58]:
```
# Intercept
print(regression.intercept_)
```

```
[32.17791411]
```

Predicition for Test Data

In [60]:
```
reg_pred=regression.predict(x_test)
reg_pred
```

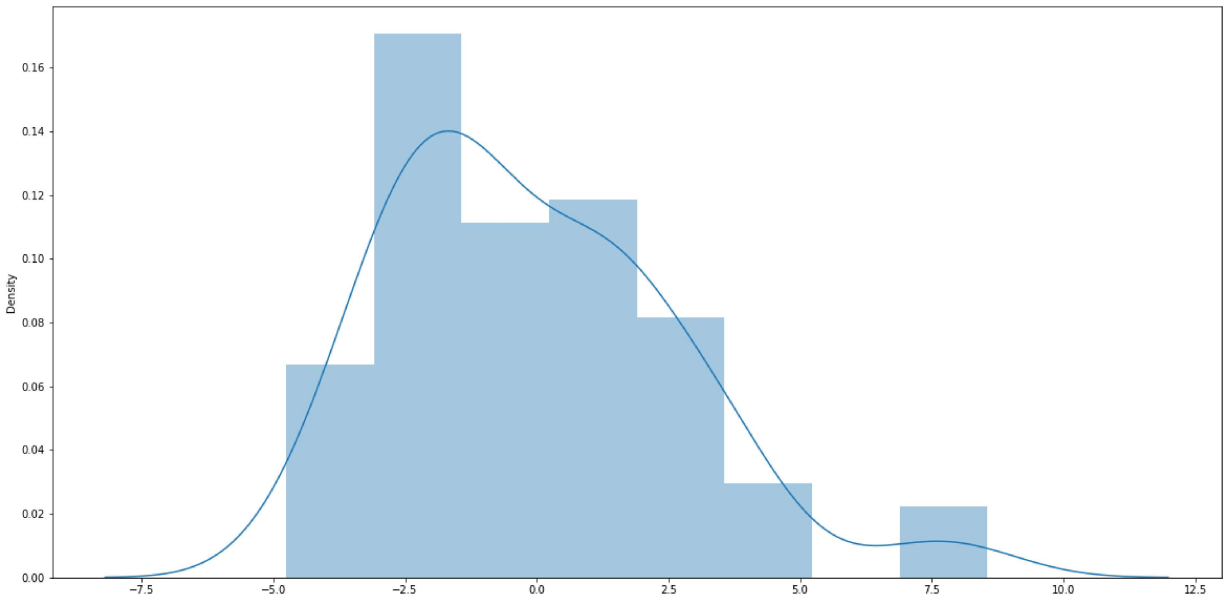
Out[60]:
```
array([[31.35728286],
       [33.48448971],
       [33.68885621],
       [32.00434093],
       [32.90791395],
       [35.12184072],
       [32.88392257],
       [34.41877632],
       [31.94627835],
       [32.98721461],
       [33.30675698],
       [27.36975802],
       [35.07784211],
       [29.24028529],
       [31.85823402],
       [32.41802576],
       [34.37535576],
       [28.09756027],
       [36.26505018],
       [34.01992072],
       [32.55507349],
       [34.37404142],
       [32.95376928],
       [33.26908507],
       [36.11652279],
       [29.43281151],
       [31.64047533],
       [32.38849979],
       [27.56606986],
       [32.22728534],
       [25.99441341],
       [27.23155419],
       [34.06867619],
       [31.64327002],
       [32.76692249],
       [31.05185077],
       [29.01675218],
       [33.06175783],
       [27.69372403],
       [35.63560078],
       [32.8693709 ],
       [33.63210892],
       [34.17783984],
       [31.5433198 ],
       [36.08261913],
       [33.41675348],
       [24.66437356],
       [35.74882134],
       [33.62798919],
       [29.69949092],
       [31.06668332],
       [32.38004487],
       [36.25233174],
```

```
       [32.16965552],
       [30.17098904],
       [30.08639562],
       [32.50310102],
       [36.07831078],
       [31.40637145],
       [33.45272801],
       [32.10289562],
       [32.80364988],
       [30.70110717],
       [24.64737332],
       [31.51727723],
       [36.35580039],
       [29.95761627],
       [29.70472774],
       [35.38938777],
       [34.07489424],
       [27.95824128],
       [32.55161796],
       [31.90597354],
       [31.60138869],
       [30.05790994],
       [31.14615789],
       [32.6846203 ],
       [36.0426106 ],
       [31.28209795],
       [36.91863039],
       [25.08477191]])
```

In [ ]:

In [61]:
```
import seaborn as sns
sns.distplot(reg_pred-y_test)
```

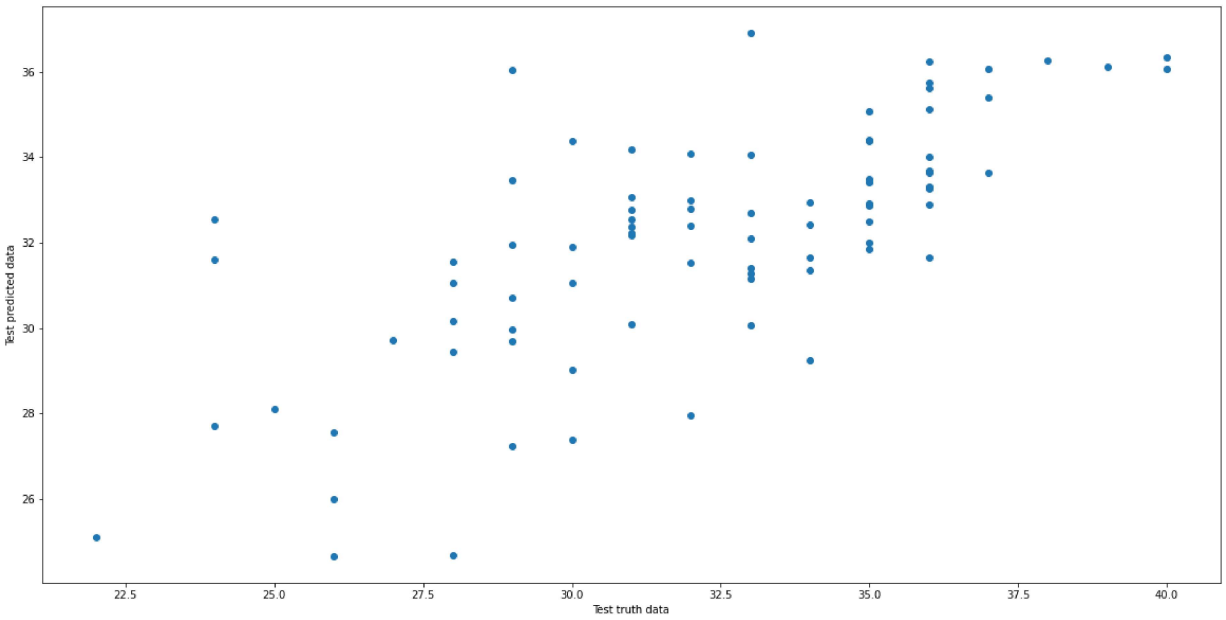
Out[61]: <AxesSubplot:ylabel='Density'>



Assumption of Linear Regression

In [62]:
```
plt.scatter(y_test,reg_pred)
plt.xlabel('Test truth data')
plt.ylabel('Test predicted data')
```

Out[62]:
```
Text(0, 0.5, 'Test predicted data')
```



Residuals

In [63]:
```
residual=y_test-reg_pred
```

In [64]:
```
residual
```

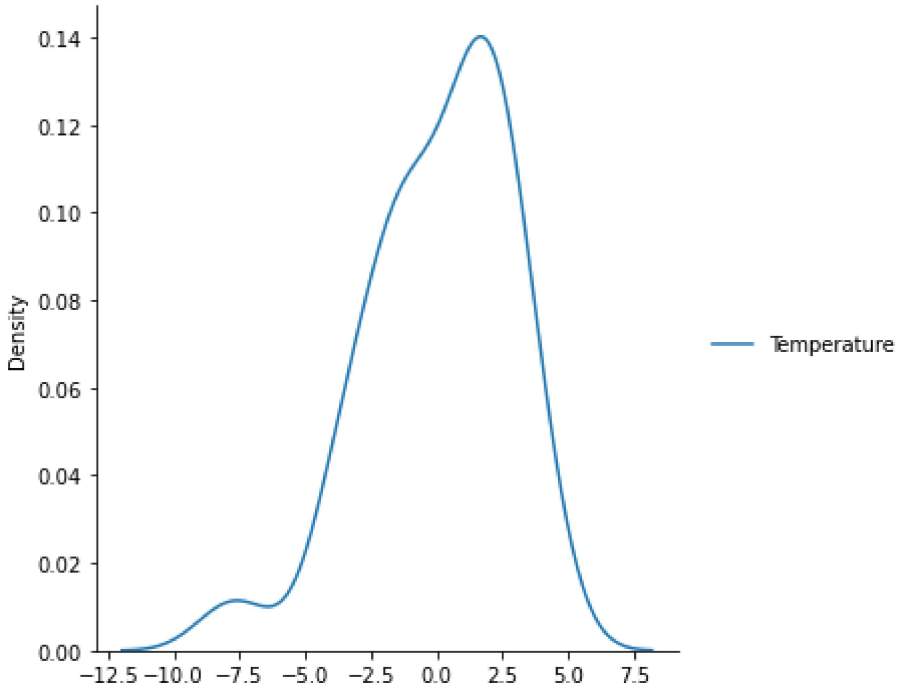
Out[64]:

	Temperature
162	2.642717
60	1.515510
61	2.311144
63	2.995659
69	2.092086
...	...
169	0.315380
232	-7.042611
144	1.717902
208	-3.918630
105	-3.084772

81 rows × 1 columns

In [65]:
```
sns.displot(residual,kind='kde')
```

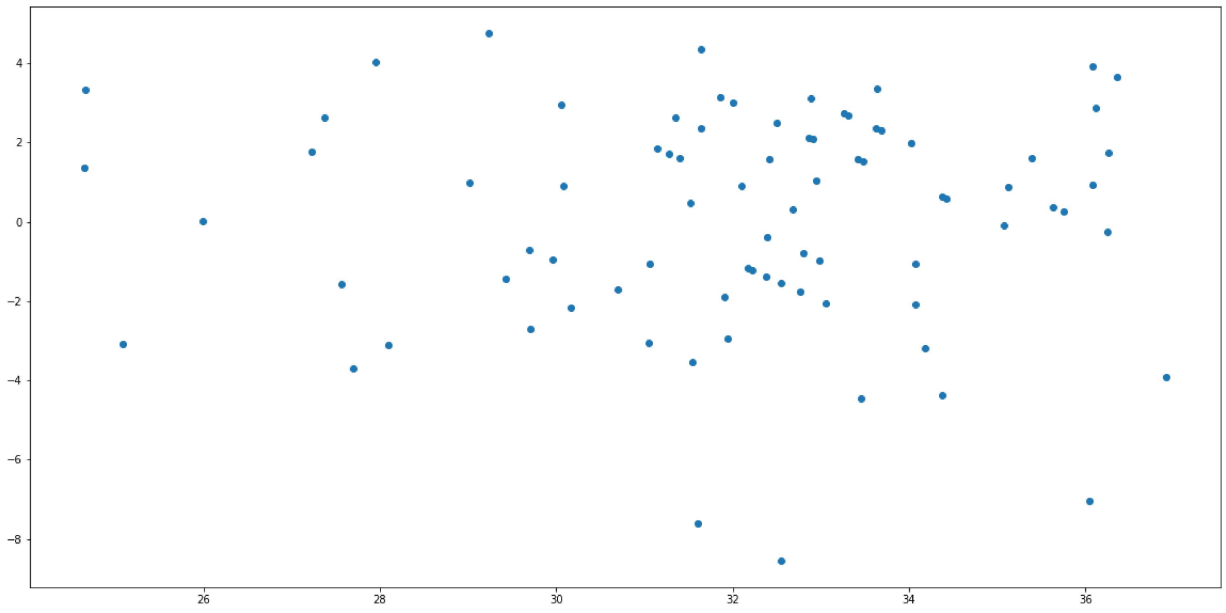
Out[65]:
```
<seaborn.axisgrid.FacetGrid at 0x25c730d33a0>
```



Scatterplot with prediction and residuals

In [66]:
```
plt.scatter(reg_pred,residual)
```

Out[66]:
```
<matplotlib.collections.PathCollection at 0x25c73d229d0>
```



Performance Metrics

In [67]:
```
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)))
```

```
7.504766973062434
2.2354993752323624
2.7394829755014785
```

R square

In [72]:
```
from sklearn.metrics import r2_score
ridge_score=r2_score(y_test,reg_pred)
print(ridge_score)
```

```
0.5037314185907533
```

Adjusted R Square

In [ ]:

In [74]:
```
1-(1-ridge_score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

Out[74]: 0.43283590696086094

Lasso Regression

In [76]:
```
from sklearn.linear_model import Lasso
lasso=Lasso()
lasso.fit(x_train,y_train)
```

Out[76]: Lasso()

In [77]:
```
lasso.fit(x_train,y_train)
```

Out[77]: Lasso()

In [78]:
```
# coefficient and intercepts
```

In [79]:
```
print(lasso.coef_)
```

```
[-0.71955751 -0.         -0.          0.89582004  0.         -0.
  0.          0.          0.         -0.        ]
```

In [80]:
```
# intercept
print(lasso.intercept_)
```

```
[32.17791411]
```

In [83]:
```
#  prediction for the test data
lasso_pred=lasso.predict(x_test)
lasso_pred
```

Out[83]:
```
array([32.29700076, 32.6744027 , 33.06609539, 32.07346965, 32.92497671,
       33.33947653, 33.32111992, 32.77042154, 32.11916885, 32.70983221,
       33.15976154, 30.29861247, 34.17172792, 30.95174825, 33.0931383 ,
       32.31497272, 32.93691477, 29.42489766, 34.46059856, 33.50695377,
       32.46152593, 33.02899752, 33.30888217, 32.80645043, 34.5498142 ,
       30.18680443, 32.38908351, 32.89121556, 29.47641605, 31.8492542 ,
       29.50217524, 28.6091198 , 33.21226395, 32.70054654, 32.64380834,
       31.80937418, 30.23515603, 32.53110125, 29.22810977, 33.62676377,
       32.55104126, 33.23190428, 33.93112391, 31.84411936, 34.06445535,
       33.20742879, 29.78847846, 33.80519505, 33.21966653, 30.53913152,
       31.62769114, 32.373594  , 33.92016988, 32.24993288, 31.51301599,
       31.26381066, 32.303719  , 34.28571873, 31.84095256, 33.47507571,
       32.27184094, 32.20868418, 31.42230192, 29.36272493, 32.24706577,
       34.47767146, 31.13749714, 31.41648274, 33.33947653, 33.04221928,
       30.62774778, 32.69215994, 32.20868418, 31.45674741, 31.17557904,
       31.67565808, 32.4164261 , 33.56882682, 32.11728577, 34.26736212,
       29.66708507])
```

Performance Matrics

In [84]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,lasso_pred))
print(mean_absolute_error(y_test,lasso_pred))
print(np.sqrt(mean_squared_error(y_test,lasso_pred)))
```

```
9.10609532182792
2.4978660766652743
3.0176307464346794
```

R square and Adjusted R Square

In [85]:
```
#  R squared
from sklearn.metrics import r2_score
lasso_score=r2_score(y_test,lasso_pred)
print(lasso_score)
```

```
0.39784019626969913
```

In [86]:
```
#  Adjusted squared
1-(1-lasso_score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

Out[86]:
```
0.31181736716537045
```

Elastic Net Regression

In [87]:
```
from sklearn.linear_model import ElasticNet
elastic=ElasticNet()
elastic
```

Out[87]:
```
ElasticNet()
```

In [89]:
```
elastic.fit(x_train,y_train)
```

Out[89]:
```
ElasticNet()
```

Coefficient and Intercept

In [90]:
```
# coefficient
print(elastic.coef_)
```

```
[-0.69396083 -0.10315403 -0.01507374  0.6926462   0.10752205 -0.
  0.28392506  0.07544656  0.05920494 -0.        ]
```

In [92]:
```
#intercept
print(elastic.intercept_)
```

```
[32.17791411]
```

In [93]:
```
1 elastic_pred = elastic.predict(x_test)
```

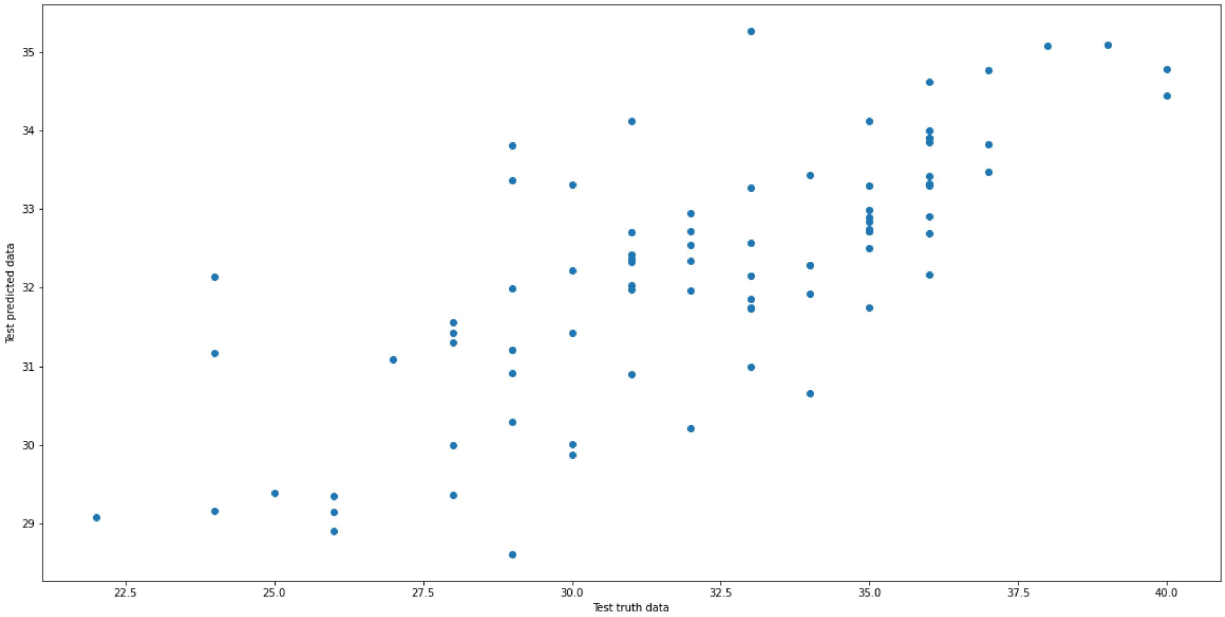
In [94]:
```
elastic_pred
```

Out[94]:
```
array([31.93076461, 32.89925279, 32.68965549, 31.74550697, 32.98836724,
       33.91299333, 33.41776043, 32.72078781, 31.98790062, 32.72490467,
       33.30044153, 29.87165926, 34.12217082, 30.65936304, 32.72255437,
       32.28333638, 32.84852051, 29.39040946, 35.0786389 , 33.86050655,
       32.42586731, 33.31820617, 33.43805315, 32.91463826, 35.09348452,
       29.99111145, 32.16696212, 32.5466324 , 29.35144729, 31.97557447,
       29.14216505, 28.60375228, 33.27376701, 32.28678221, 32.70561446,
       31.30843328, 30.01197805, 32.32347658, 29.15902087, 33.91042464,
       32.74766721, 33.47903767, 34.12940549, 31.56461381, 34.44958031,
       33.30036678, 29.35558024, 33.99794233, 33.3293338 , 30.2978916 ,
       31.4241864 , 32.36311525, 34.61701506, 32.03023331, 31.41948172,
       30.89419347, 32.508051  , 34.7691147 , 31.73370163, 33.3687946 ,
       32.1509131 , 32.34603101, 31.21296836, 28.90195873, 31.96458545,
       34.79046967, 30.91855019, 31.0841594 , 33.82121587, 32.95120309,
       30.20921418, 32.14232146, 32.22575701, 31.16525171, 30.98976689,
       31.86034438, 32.57697932, 33.81340658, 31.74661525, 35.27207415,
       29.07891997])
```

Assumption of Elastic Net Regression

In [97]:
```
plt.scatter(y_test,elastic_pred)
plt.xlabel('Test truth data')
plt.ylabel('Test predicted data')
```

Out[97]:
```
Text(0, 0.5, 'Test predicted data')
```



In [100]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

print(mean_squared_error(y_test,elastic_pred))
print(mean_absolute_error(y_test,elastic_pred))
print(np.sqrt(mean_squared_error(y_test,elastic_pred)))
```

```
8.346007590926808
2.3987645425349116
2.8889457576989583
```

R squared and Adjusted R squared

In [103]:
```
#   R squared
from sklearn.metrics import r2_score
elastic_score=r2_score(y_test,elastic_pred)
print(elastic_score)
```

```
0.4481026043251144
```

In [107]:
```
#  Adjusted R S quared
1-(1-elastic_score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

Out[107]:
```
0.36926011922870217
```

In [ ]: