Machine learning - Linear regression exercise

Algerian Forest fire dataset

Attribute Information: 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 â€œFireâ€ and â€œnot Fireâ€

importing libraries

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

data reading and cleaning

In [2]:

```
df = pd.read_csv(r"C:\Users\annah\Downloads\Algerian_forest_fires_dataset_UPDATE.csv",head
df
```

Out[2]:

	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

dropping empty rows

In [3]:

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

In [4]:
```
df.loc[:122,'region'] ='bejaia'
df.loc[122:,'region'] ='Sidi-Bel Abbes'
```

stripping the names of columns

In [5]:
```
df.columns= [i.strip() for i in df.columns]
df.columns
```

Out[5]:
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'region'],
      dtype='object')
```

Converting categorical value to numerical for ease of ML

not fire = 0 , fire = 1

In [6]:
```
df['Classes'].unique()
```

Out[6]:
```
array(['not fire   ', 'fire   ', 'fire', 'fire ', 'not fire', 'not fire ',
       'not fire     ', nan, 'not fire    '], dtype=object)
```

In [7]:
```
df['Classes'] = df['Classes'].replace(to_replace ='not fire   ', value = 0)
```

In [8]:
```
df['Classes'] = df['Classes'].replace(to_replace ='not fire', value = 0)
df['Classes'] = df['Classes'].replace(to_replace ='not fire ', value = 0)
df['Classes'] = df['Classes'].replace(to_replace ='not fire     ', value = 0)
df['Classes'] = df['Classes'].replace(to_replace ='not fire    ', value = 0)
```

In [9]:
```
df['Classes'] = df['Classes'].replace(to_replace = 'fire   ', value = 1)
df['Classes'] = df['Classes'].replace(to_replace = 'fire', value = 1)
df['Classes'] = df['Classes'].replace(to_replace = 'fire ', value = 1)
```

In [10]:
```
df['Classes'].unique()
```

Out[10]:
```
array([ 0.,  1., nan])
```

dropping null values

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
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
region         0
dtype: int64
```

In [12]:
```
df.dropna(inplace = True)
```

In [13]:
```
df.isnull().sum()
```

Out[13]:
```
day            0
month          0
year           0
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        0
region         0
dtype: int64
```

replacing date month year feature with datefeature

In [14]:
```
df['date'] = pd.to_datetime(df[['day','month','year']])   # adding new column with day , r
df.drop(['day','month','year'], axis =1, inplace = True)
```

In [15]:
```
df.head()
```

Out[15]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
0	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	0.0	bejaia	2012-06-01
1	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	0.0	bejaia	2012-06-02
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0.0	bejaia	2012-06-03
3	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	0.0	bejaia	2012-06-04
4	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	0.0	bejaia	2012-06-05

checking datatypes

In [16]:
```
df.dtypes
```

Out[16]:
```
Temperature            object
RH                     object
Ws                     object
Rain                   object
FFMC                   object
DMC                    object
DC                     object
ISI                    object
BUI                    object
FWI                    object
Classes               float64
region                 object
date           datetime64[ns]
dtype: object
```

Observation: many numerical values are in text format, now we

convert them to relevent data type

In [17]:
```
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)
```

In [18]:
```
df['DMC'] = df['DMC'].astype(float)
df['ISI'] = df['ISI'].astype(float)
df['BUI'] = df['BUI'].astype(float)
df['DC'] = df['DC'].astype(float)
df['FWI'] = df['FWI'].astype(float)
```

In [19]:
```
df.dtypes
```

Out[19]:
```
Temperature               int32
RH                        int32
Ws                        int32
Rain                    float64
FFMC                    float64
DMC                     float64
DC                      float64
ISI                     float64
BUI                     float64
FWI                     float64
Classes                 float64
region                   object
date             datetime64[ns]
dtype: object
```

lets use label encoder for converting catogorical or taxt values to numericals , which is best for feeding data ml algorithum

in the begining we manually encoded the data , now we are using tool to encode data

In [20]:
```
from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
```

In [21]:
```
df['region']= LE.fit_transform(df['region'])
```

In [22]:
```
df.head()
```

Out[22]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
0	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0.0	1	2012-06-01
1	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0.0	1	2012-06-02
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0.0	1	2012-06-03
3	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0.0	1	2012-06-04
4	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0.0	1	2012-06-05

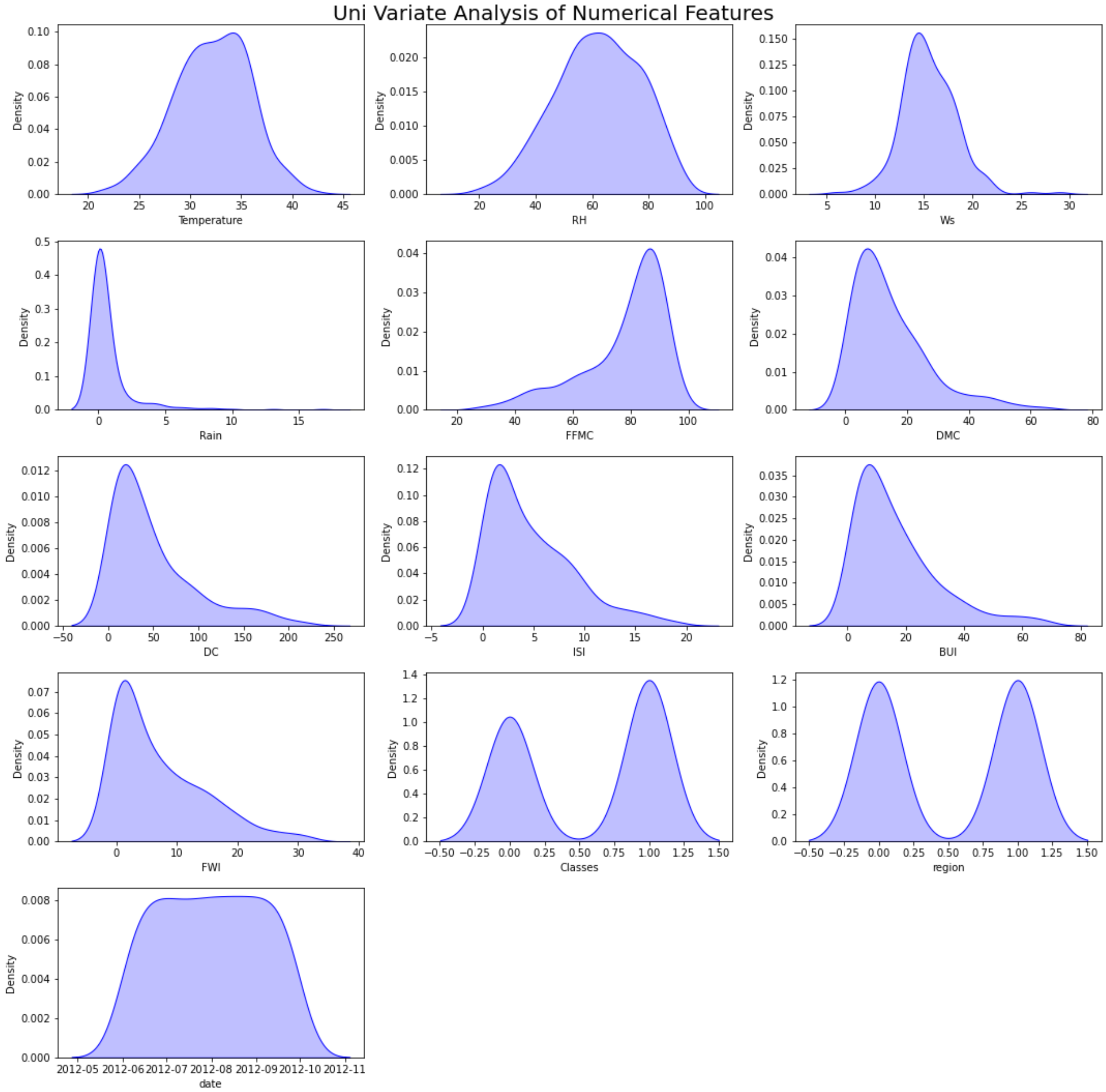
Uni Variate analysis

In [23]:
```
feature = [i for i in df.columns if df[i].dtype != 'O']
```

In [24]:
```
feature
```

Out[24]:
```
['Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'DC',
 'ISI',
 'BUI',
 'FWI',
 'Classes',
 'region',
 'date']
```

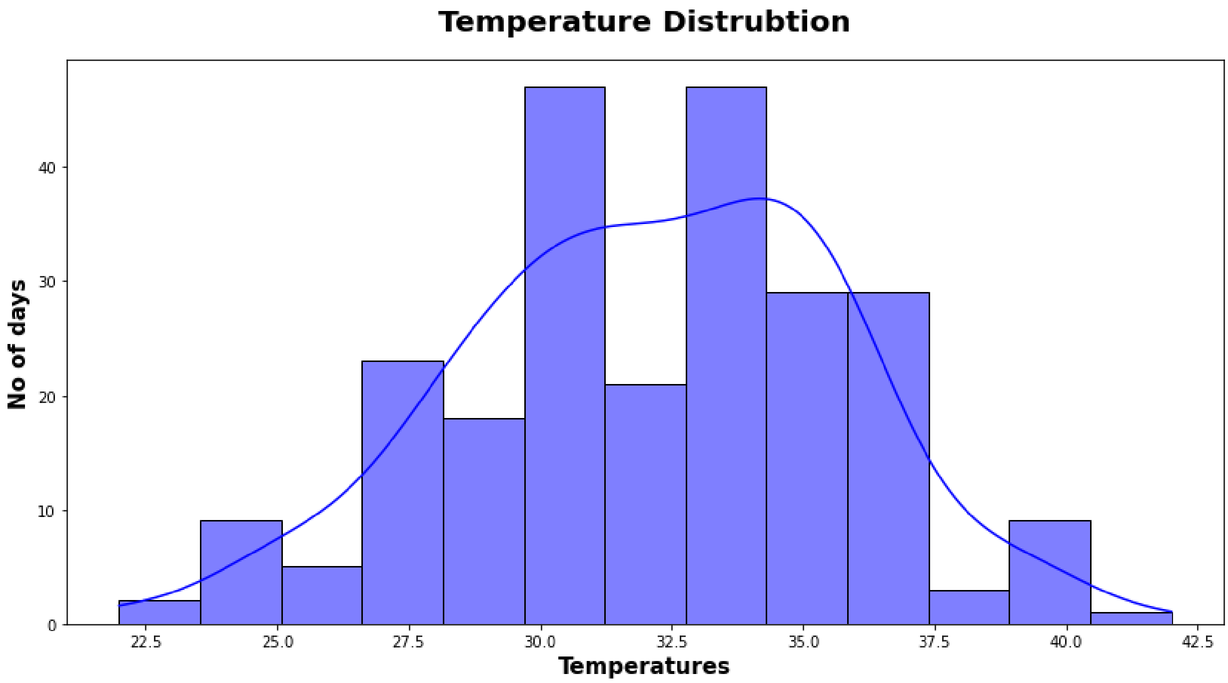
In [25]:
```
plt.figure(figsize =(15,15))
plt.suptitle('Uni Variate Analysis of Numerical Features', fontsize = 20 )

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



Visualizing Target Feture

In [26]:

```
plt.subplots(figsize=(14,7))
sns.histplot(x=df.Temperature,ec = 'black', color = 'blue', kde= True)
plt.title('Temperature Distrubtion', weight = 'bold',fontsize = 20, pad = 20)
plt.ylabel('No of days', weight = 'bold',fontsize = 15)
plt.xlabel('Temperatures', weight = 'bold',fontsize = 15)
plt.show()
```



Observation: More ofen temperature is in range of 27.5 to 37

..

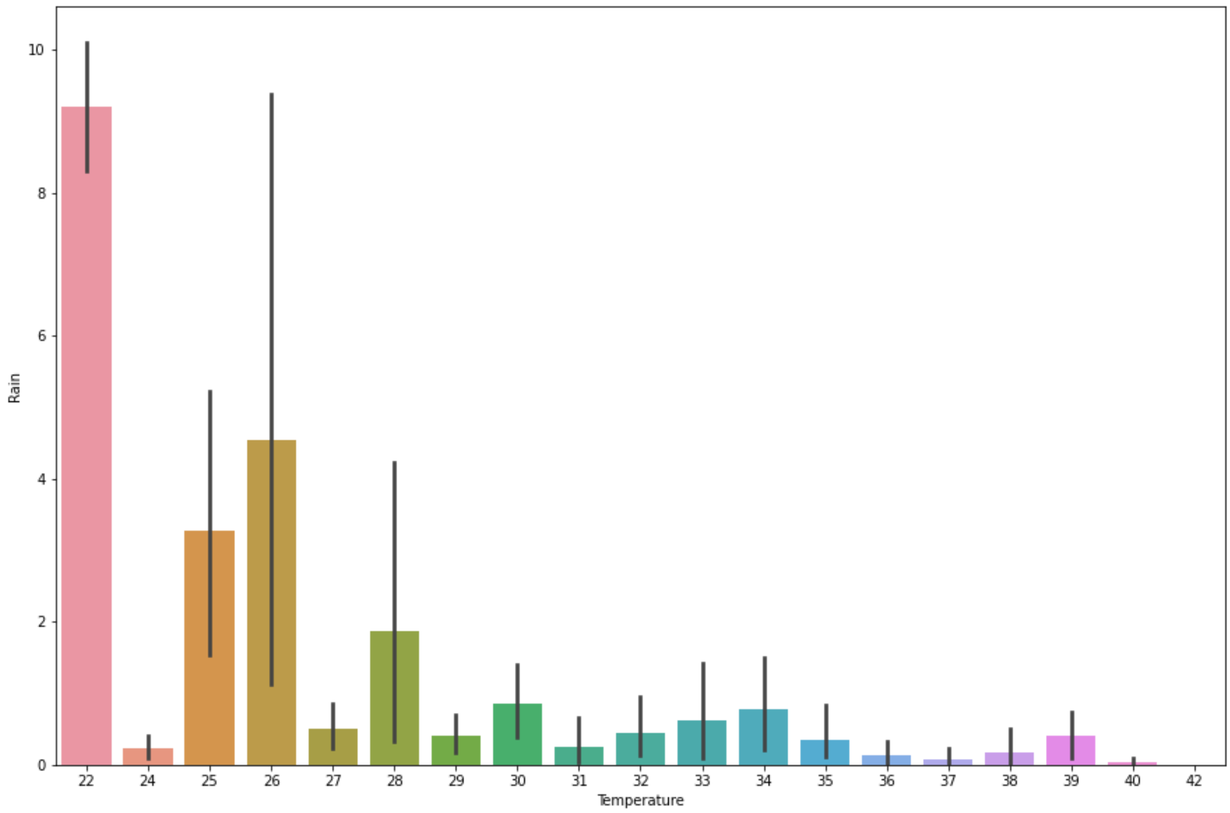
Temperauture vs Rain

In [27]:

```
plt.subplots(figsize=(15,10))
sns.barplot(x = 'Temperature', y = 'Rain', data = df)
```

Out[27]:

```
<AxesSubplot:xlabel='Temperature', ylabel='Rain'>
```

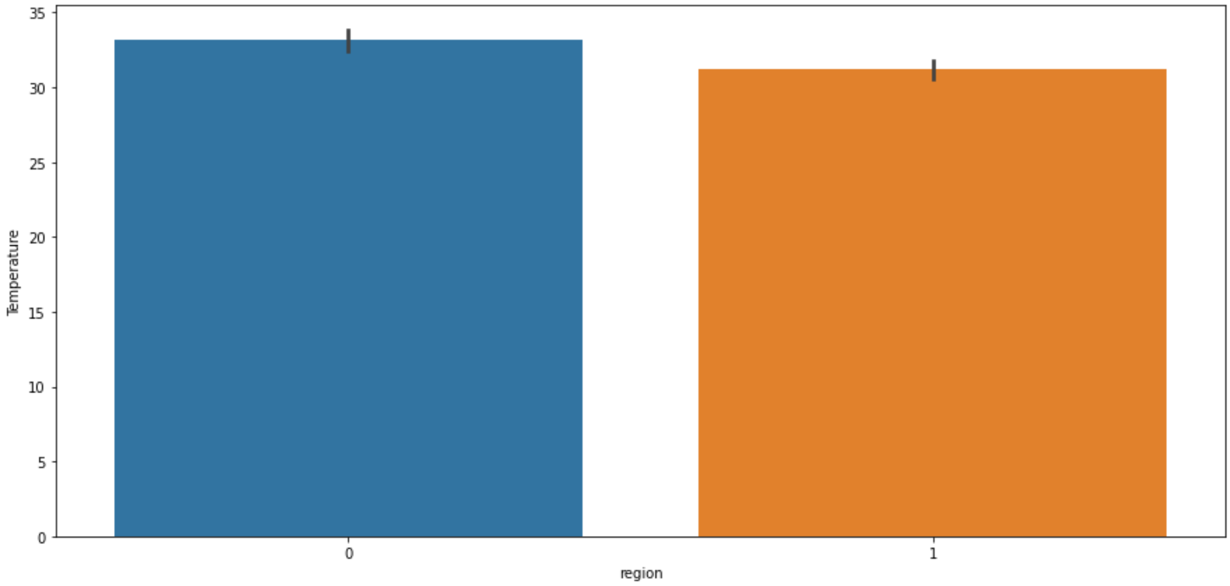


Observation: When temparures are in 22 most rain occured

Which region has most temperature?

In [28]:
```
plt.subplots(figsize=(15,7))
sns.barplot(x = 'region', y = 'Temperature', data = df)
```

Out[28]: <AxesSubplot:xlabel='region', ylabel='Temperature'>



Observation: region 0 i.e. 'Sidi-Bel Abbes' has highest temperature

correlation of features

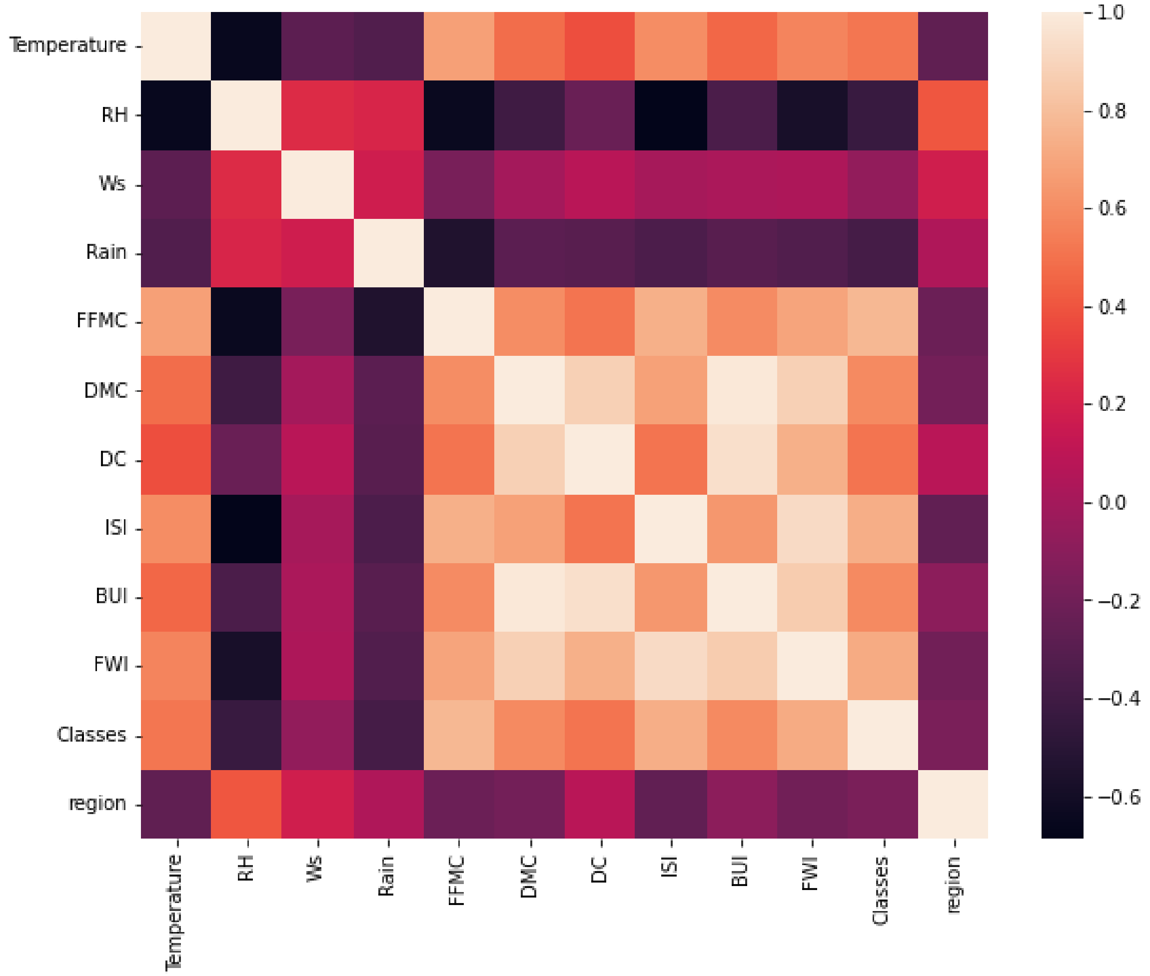
In [29]:
```
df.corr()
```

Out[29]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	
Temperature	1.000000	-0.651400	-0.284510	-0.326492	0.676568	0.485687	0.376284	0.603871	0.4
RH	-0.651400	1.000000	0.244048	0.222356	-0.644873	-0.408519	-0.226941	-0.686667	-0.3
Ws	-0.284510	0.244048	1.000000	0.171506	-0.166548	-0.000721	0.079135	0.008532	0.0
Rain	-0.326492	0.222356	0.171506	1.000000	-0.543906	-0.288773	-0.298023	-0.347484	-0.2
FFMC	0.676568	-0.644873	-0.166548	-0.543906	1.000000	0.603608	0.507397	0.740007	0.5
DMC	0.485687	-0.408519	-0.000721	-0.288773	0.603608	1.000000	0.875925	0.680454	0.9
DC	0.376284	-0.226941	0.079135	-0.298023	0.507397	0.875925	1.000000	0.508643	0.9
ISI	0.603871	-0.686667	0.008532	-0.347484	0.740007	0.680454	0.508643	1.000000	0.6
BUI	0.459789	-0.353841	0.031438	-0.299852	0.592011	0.982248	0.941988	0.644093	1.0
FWI	0.566670	-0.580957	0.032368	-0.324422	0.691132	0.875864	0.739521	0.922895	0.8
Classes	0.516015	-0.432161	-0.069964	-0.379097	0.769492	0.585658	0.511123	0.735197	0.5
region	-0.269555	0.402682	0.181160	0.040013	-0.222241	-0.192089	0.078734	-0.263197	-0.0

In [30]:
```
plt.figure(figsize = (10,8))
sns.heatmap(df.corr())
```

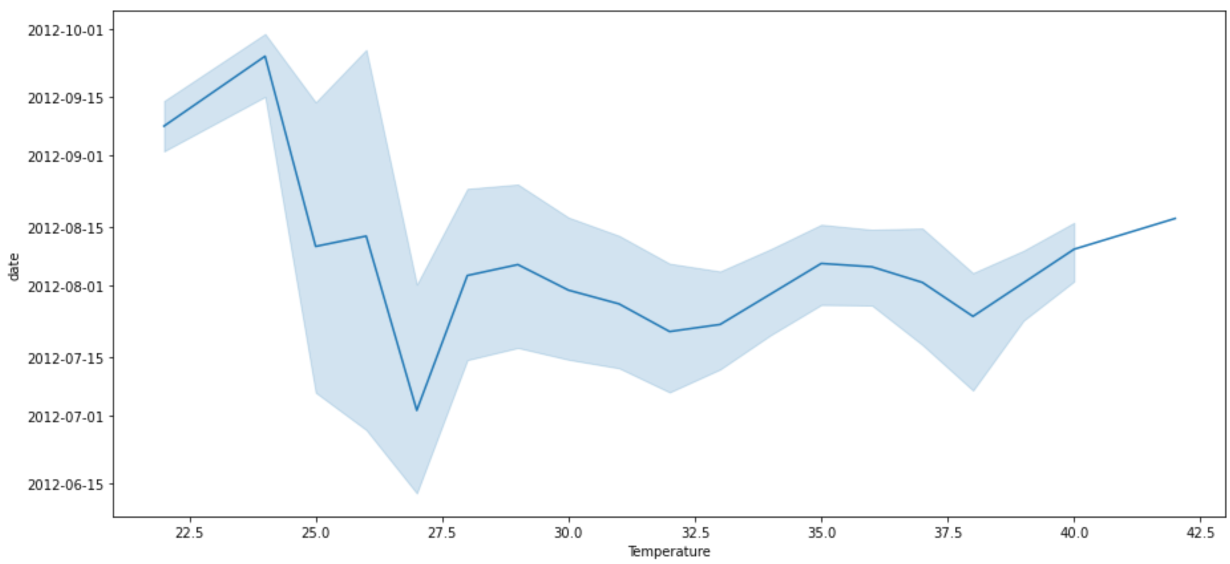
Out[30]: <AxesSubplot:>



Temperature Vs date feature

In [31]:
```
plt.subplots(figsize=(15,7))
sns.lineplot(x = 'Temperature', y = 'date', data = df)
```

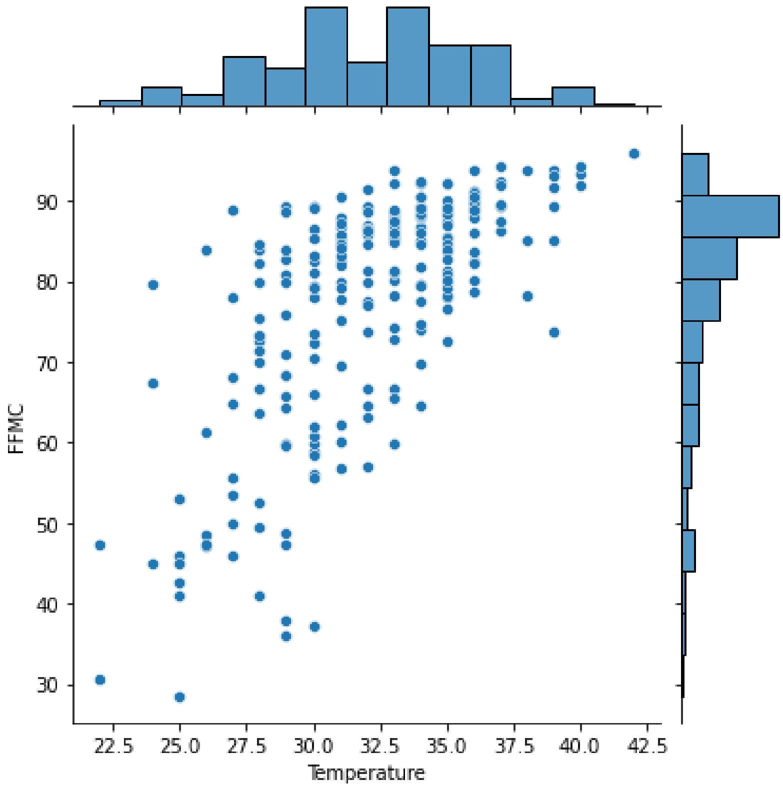
Out[31]: <AxesSubplot:xlabel='Temperature', ylabel='date'>



Temperature Vs FFMC feature

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

Out[32]: <seaborn.axisgrid.JointGrid at 0x15af0db3130>

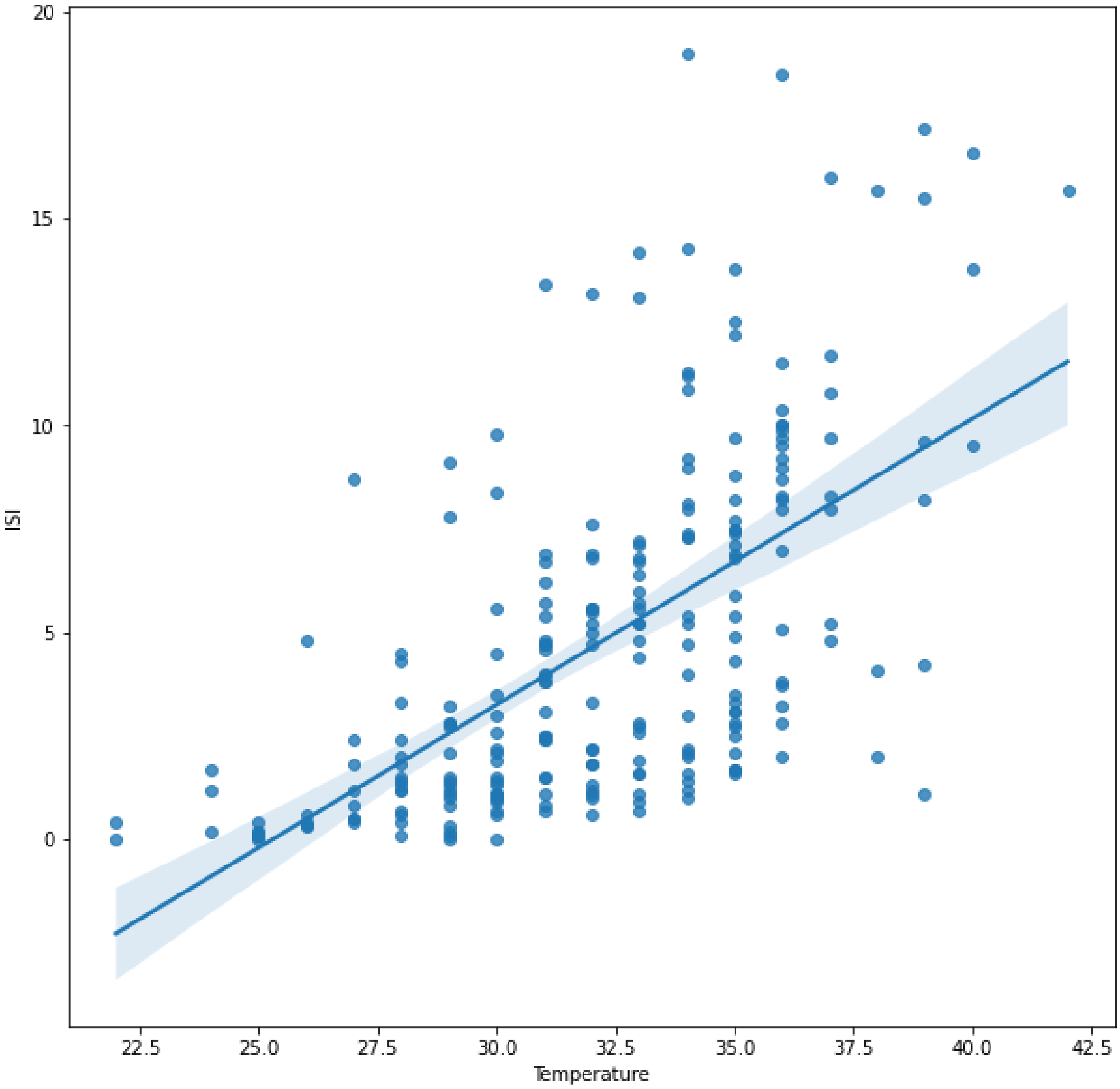


Temperature Vs ISI feature

In [33]:

```
plt.subplots(figsize=(10,10))
sns.regplot(x = 'Temperature', y = 'ISI', data = df)
```

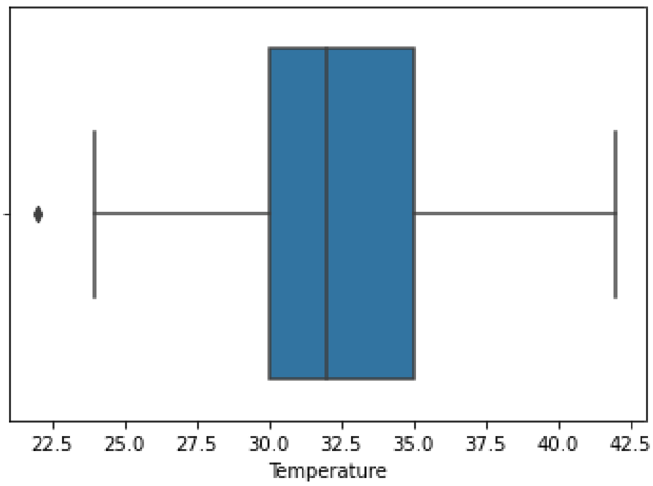
Out[33]: <AxesSubplot:xlabel='Temperature', ylabel='ISI'>



Checking the outliers of the target " Temperature " feature

In [34]:
```
sns.boxplot(df['Temperature'])
```

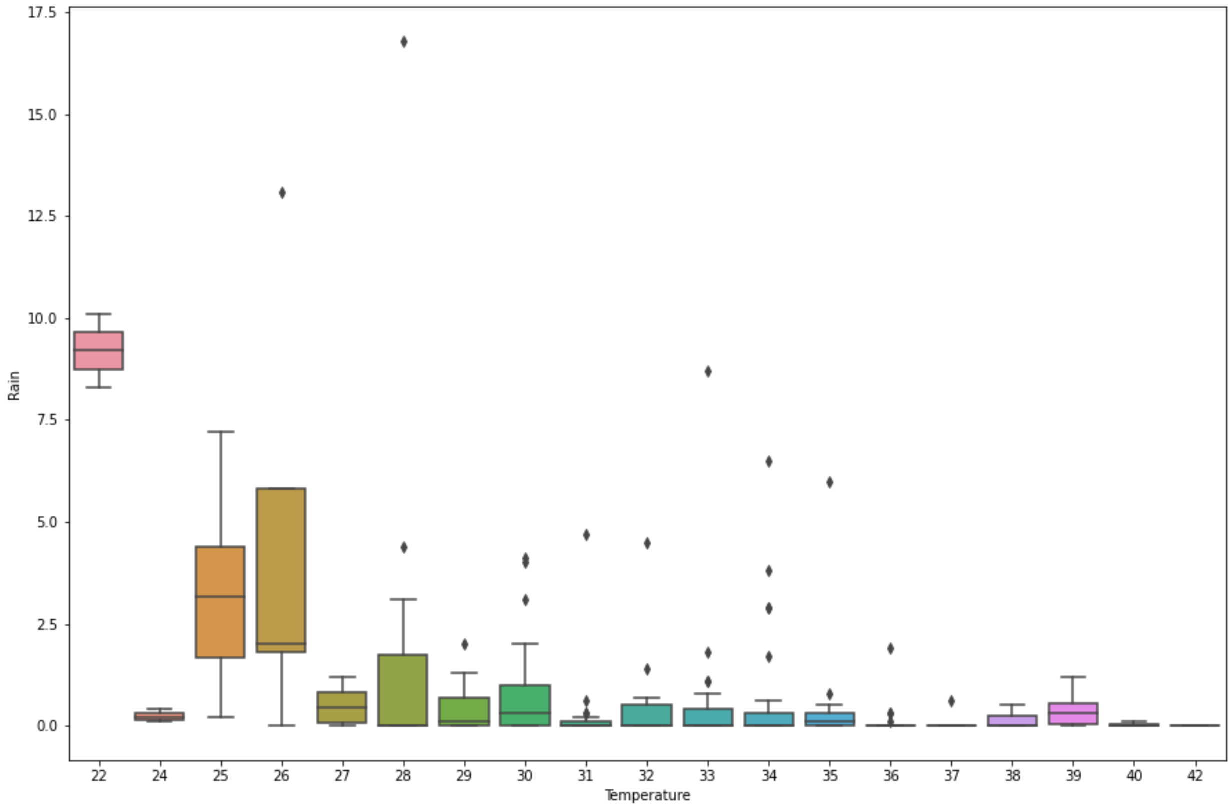
Out[34]: <AxesSubplot:xlabel='Temperature'>



Boxplot of Rain vs Temperature

In [35]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(x= 'Temperature', y = 'Rain', data = df)
```

Out[35]: <AxesSubplot:xlabel='Temperature', ylabel='Rain'>

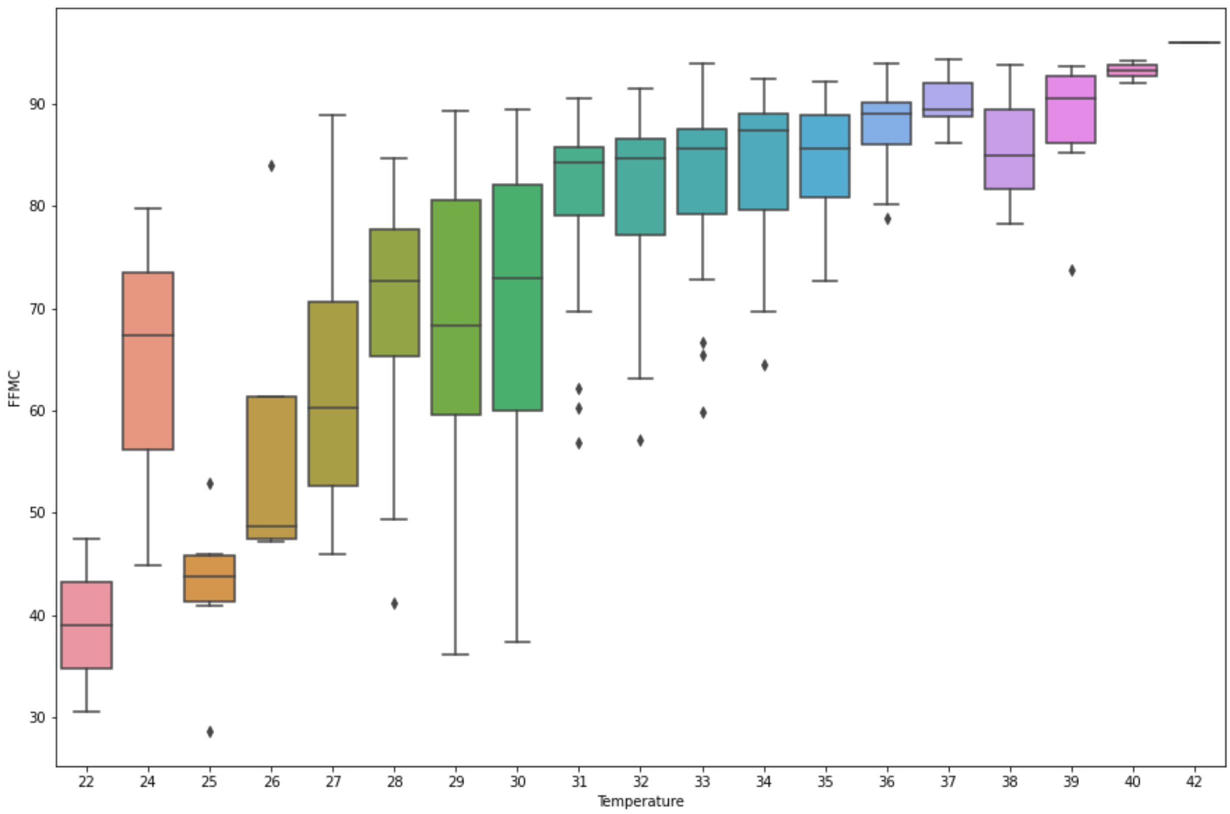


Boxplot of FFMC vs Temperature

In [36]:

```
plt.subplots(figsize=(15,10))
sns.boxplot(x= 'Temperature', y = 'FFMC', data = df)
```

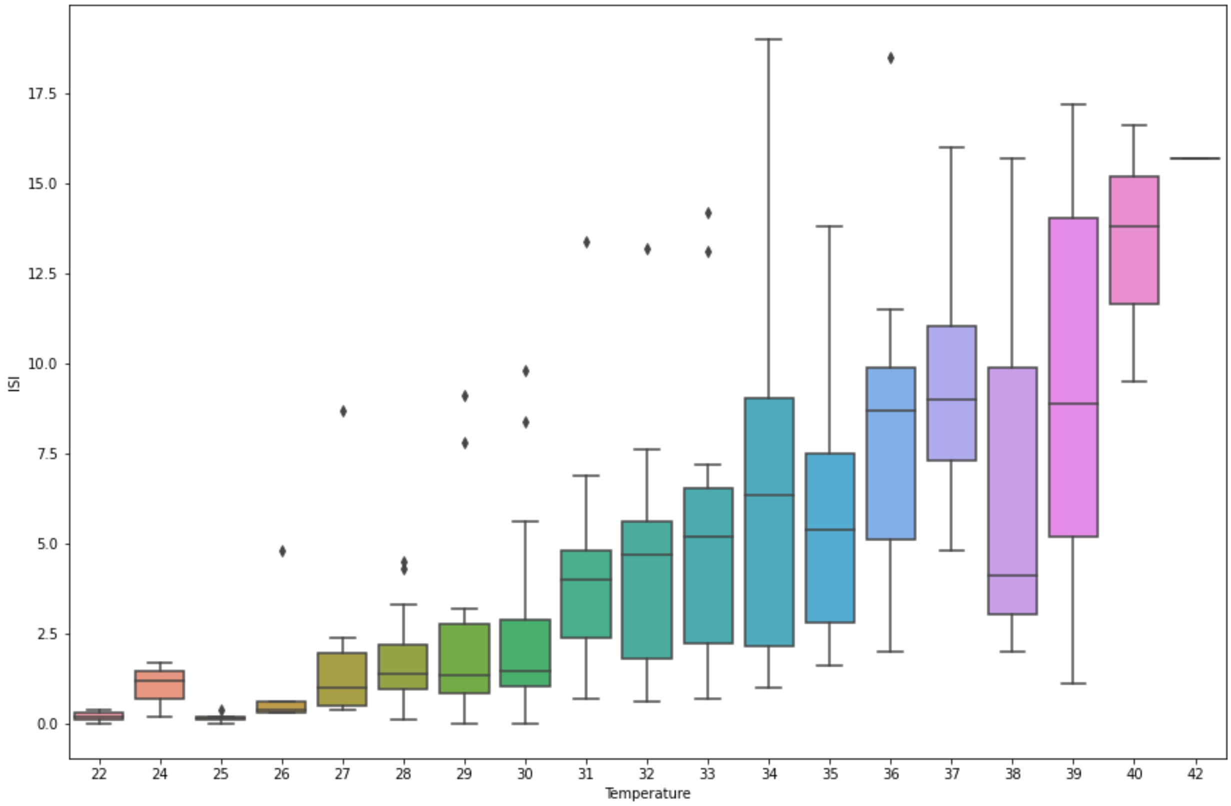
Out[36]: <AxesSubplot:xlabel='Temperature', ylabel='FFMC'>



Boxplot of ISI vs Temperature

In [37]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(x= 'Temperature', y = 'ISI', data = df)
```

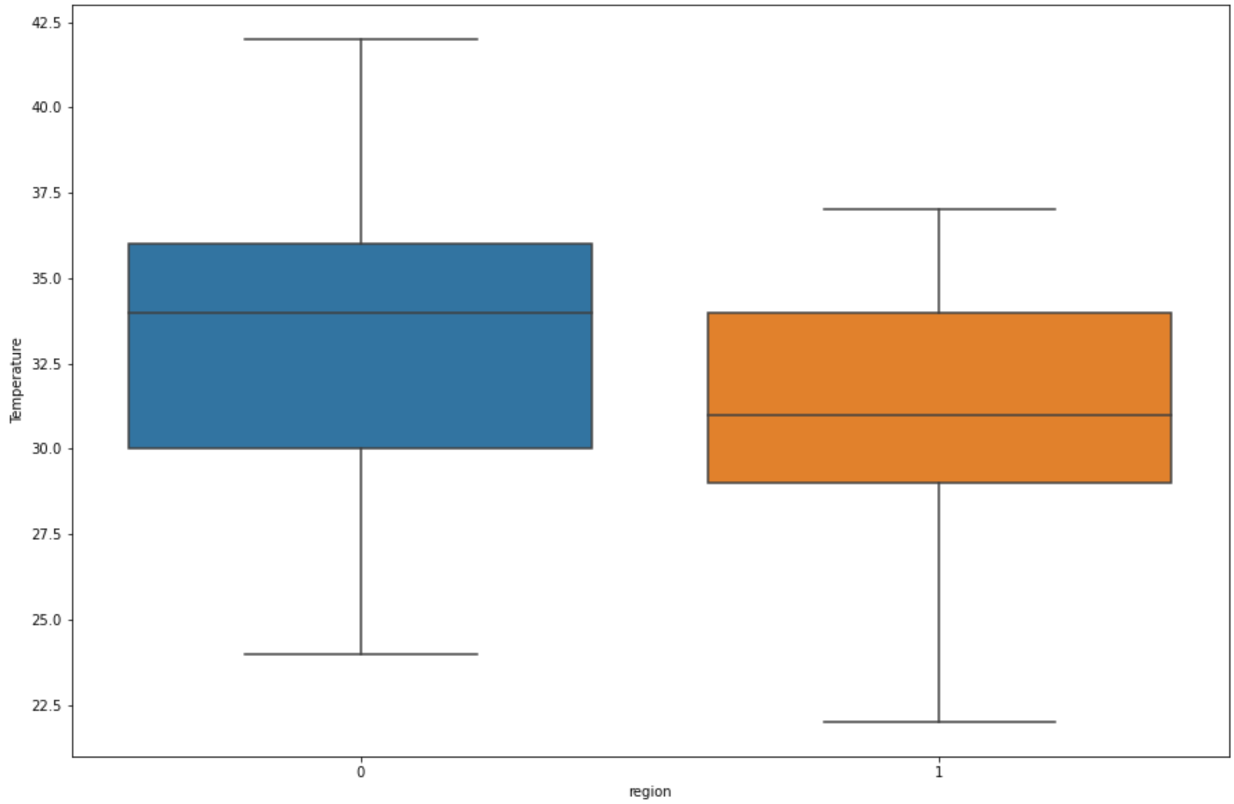
Out[37]: <AxesSubplot:xlabel='Temperature', ylabel='ISI'>



Boxplot of Region vs Temperature

In [38]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(x= 'region', y = 'Temperature', data = df)
```

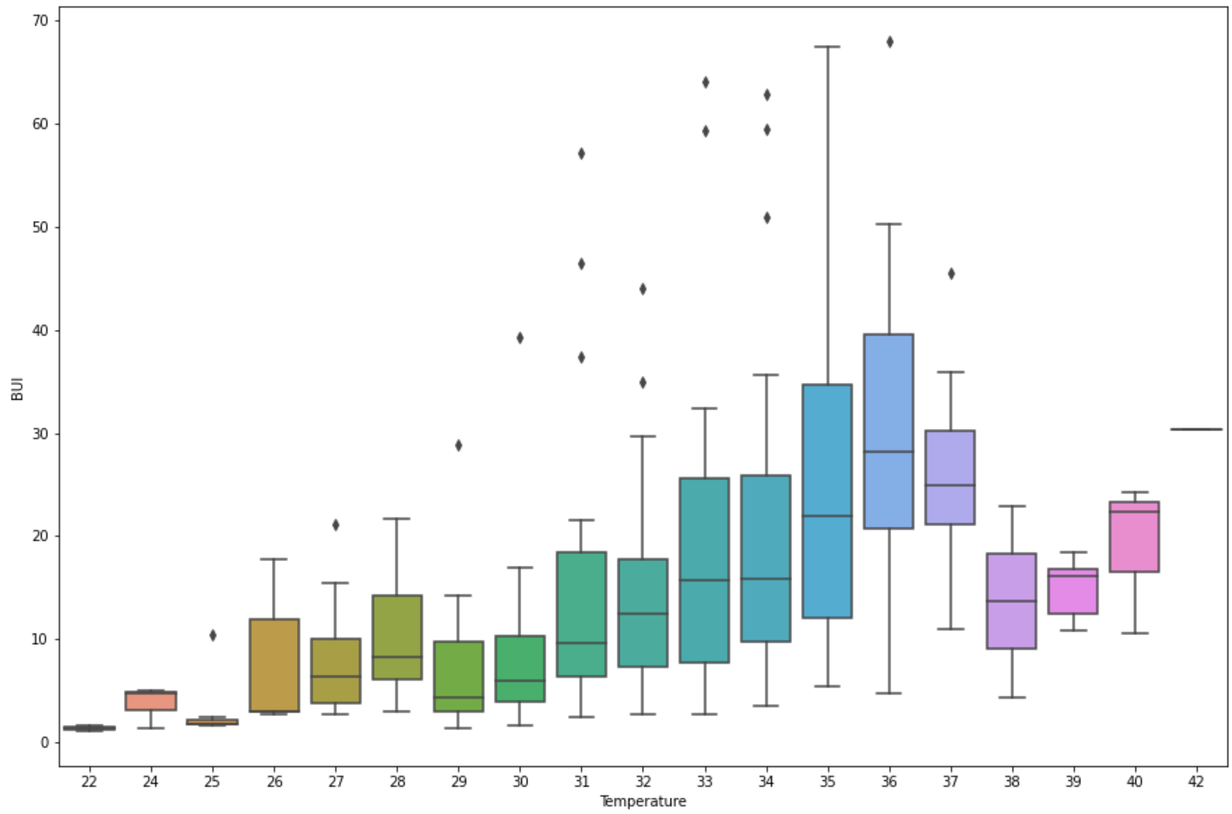
Out[38]: <AxesSubplot:xlabel='region', ylabel='Temperature'>



Boxplot of BUI vs Temperature

In [39]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(x= 'Temperature', y = 'BUI', data = df)
```

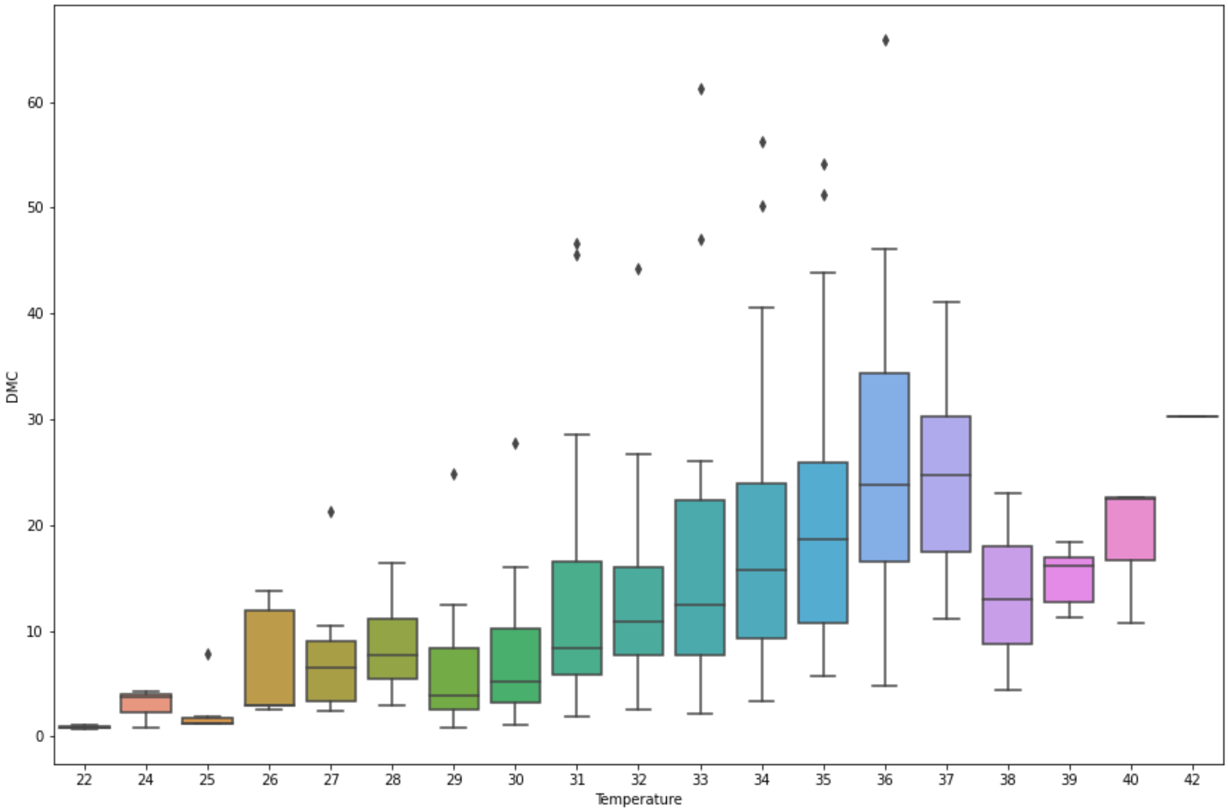
Out[39]: <AxesSubplot:xlabel='Temperature', ylabel='BUI'>



Boxplot of DMC vs Temperature

In [40]:
```
plt.subplots(figsize=(15,10))
sns.boxplot(x= 'Temperature', y = 'DMC', data = df)
```

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



Model Training

Creating Dependent & Independent Features

In [41]:
```
df.columns
```

Out[41]:
```
Index(['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
       'FWI', 'Classes', 'region', 'date'],
      dtype='object')
```

In [42]:
```
## independent Features  Here classes is being excluded since fire and no fire condition

x = pd.DataFrame(df, columns= ['RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI','FWI

## Dependent Feature

y = pd.DataFrame(df, columns = ['Temperature'])
```

In [43]:
```
x
```

Out[43]:

	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region
0	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	1
1	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	1
2	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	1
3	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	1
4	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	1

	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	region
...	...	...	...	...	...	...	...	...	...	...
239	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	0
240	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	0
241	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	0
242	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	0
243	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	0

243 rows × 10 columns

In [44]:

```
y
```

Out[44]:

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

243 rows × 1 columns

Train Test Split

In [45]:

```
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 [46]:

```
x_train.shape
```

Out[46]: (162, 10)

In [47]:

```
y_train.shape
```

Out[47]: (162, 1)

In [48]:

```
x_test.shape
```

Out[48]: (81, 10)

In [49]:
```
y_test.shape
```
Out[49]:
```
(81, 1)
```

Standardizing or Feature Scaling

In [50]:
```
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
```

In [51]:
```
Scaler
```
Out[51]:
```
StandardScaler()
```

In [52]:
```
x_train = Scaler.fit_transform(x_train)
```

In [53]:
```
x_test = Scaler.transform(x_test)
```

In [54]:
```
x_train
```
Out[54]:
```
array([[ 0.06835876,  0.89673457, -0.42406458, ...,  0.38986031,
         0.52024214,  1.01242284],
       [ 0.99672801, -0.58185068,  0.40434065, ..., -1.0010797 ,
        -0.93452011,  1.01242284],
       [ 0.53254338,  0.52708826, -0.42406458, ...,  2.19373563,
         1.30997022, -0.9877296 ],
       ...,
       [-2.45150064, -0.95149699, -0.42406458, ...,  0.44781614,
         1.72561657, -0.9877296 ],
       [ 0.06835876, -0.58185068, -0.42406458, ..., -0.76925637,
        -0.8098262 , -0.9877296 ],
       [ 1.0630401 , -1.3211433 , -0.42406458, ..., -0.26214282,
        -0.82368108,  1.01242284]])
```

In [55]:
```
x_test
```
Out[55]:
```
array([[ 4.66231295e-01, -5.81850675e-01, -4.24064583e-01,
         3.74594357e-01, -4.06694465e-01, -3.03195163e-01,
        -3.54971234e-01, -3.70810009e-01, -4.49599362e-01,
         1.01242284e+00],
       [-9.26322582e-01,  1.26638088e+00, -4.24064583e-01,
         7.58044470e-01, -2.28559462e-01, -3.17928133e-01,
         1.05400512e+00, -2.76631780e-01,  4.78677502e-01,
        -9.87729597e-01],
       [-4.62137956e-01,  1.57441947e-01, -4.24064583e-01,
         7.77878096e-01,  8.14802699e-01,  1.80888129e-01,
         8.39076521e-01,  5.49238849e-01,  8.80468981e-01,
        -9.87729597e-01],
       [ 1.19566428e+00,  2.00567350e+00,  3.00789995e+00,
        -1.88643906e+00, -9.49582093e-01, -8.58838594e-01,
        -1.02363798e+00, -9.57612824e-01, -9.34520112e-01,
         1.01242284e+00],
       [ 1.12935219e+00, -5.81850675e-01,  7.59371462e-01,
        -1.87982786e+00, -1.01744305e+00, -8.60943304e-01,
        -1.04751893e+00, -9.86590741e-01, -9.34520112e-01,
        -9.87729597e-01],
       [-3.29513777e-01,  1.57441947e-01, -4.24064583e-01,
         7.24988426e-01,  1.15410747e+00,  2.65076527e-01,
         6.24147925e-01,  8.31773537e-01,  8.52759223e-01,
        -9.87729597e-01],
```

```
       [-1.96889599e-01,  8.96734570e-01, -2.46549176e-01,
        -2.24454621e-03, -2.45524700e-01, -3.16875778e-02,
        -5.93780785e-01, -1.53475633e-01, -5.88148148e-01,
         1.01242284e+00],
       [-8.60010492e-01,  1.57441947e-01, -4.24064583e-01,
         8.43990185e-01,  1.56975581e+00,  4.73442814e-01,
         1.14952894e+00,  1.18675302e+00,  1.44851900e+00,
        -9.87729597e-01],
       [ 4.66231295e-01,  1.57441947e-01, -4.24064583e-01,
         5.86153040e-01, -1.18285412e-01, -4.82095510e-01,
         1.70409778e-01, -2.54898342e-01, -2.00981263e-02,
        -9.87729597e-01],
       [-1.05894676e+00, -5.81850675e-01, -9.86196704e-03,
         2.42002891e-02, -6.01794706e-01, -8.35686784e-01,
        -6.89304605e-01, -6.82322615e-01, -8.23681083e-01,
         1.01242284e+00],
       [-3.29513777e-01, -2.06043592e+00, -3.05720978e-01,
         7.70899598e-02, -2.87937796e-01, -6.06273398e-01,
        -7.37066515e-01, -3.99787926e-01, -8.09826205e-01,
        -9.87729597e-01],
       [ 2.04666942e-03, -5.81850675e-01, -4.24064583e-01,
         6.52265128e-01, -2.79455177e-01, -2.42158574e-01,
         2.18171689e-01, -2.69387301e-01, -6.24324775e-03,
         1.01242284e+00],
       [-1.92100392e+00,  1.57441947e-01, -3.05720978e-01,
         7.38210843e-01,  2.29501975e-01, -7.16770671e-02,
         6.71909835e-01,  9.28366593e-02,  5.06387259e-01,
        -9.87729597e-01],
       [ 1.06304010e+00, -2.12204364e-01, -4.24064583e-01,
         5.46485787e-01,  1.21348580e+00,  1.84150429e+00,
         3.24309267e-03,  1.53448802e+00,  5.34097016e-01,
         1.01242284e+00],
       [-6.61074224e-01,  5.27088259e-01, -1.28205572e-01,
         2.02702928e-01,  5.51841504e-01,  2.11932601e+00,
        -4.74376009e-01,  1.04186343e+00, -1.30937155e-01,
        -9.87729597e-01],
       [-6.61074224e-01,  8.96734570e-01, -4.24064583e-01,
         7.84489305e-01,  6.28185077e-01,  6.69180840e-01,
         1.07788607e+00,  6.94128433e-01,  1.12985679e+00,
         1.01242284e+00],
       [ 1.12935219e+00,  1.57441947e-01,  6.41027858e-01,
        -1.96577357e+00, -9.58064712e-01, -8.58838594e-01,
        -1.04751893e+00, -9.57612824e-01, -9.34520112e-01,
         1.01242284e+00],
       [-1.98731601e+00,  5.27088259e-01, -4.24064583e-01,
         1.09521612e+00,  2.46467213e-01, -3.45289362e-01,
         2.98836248e+00,  4.93697842e-02,  1.75332633e+00,
        -9.87729597e-01],
       [ 1.34670848e-01, -5.81850675e-01, -4.24064583e-01,
         4.86984908e-01, -1.43733270e-01,  1.21956250e-01,
        -1.63923593e-01, -3.03194870e-02, -1.72501791e-01,
         1.01242284e+00],
       [ 1.12935219e+00,  2.00567350e+00,  9.51679820e+00,
        -1.62860192e+00, -4.66072799e-01, -8.37791494e-01,
        -9.75876066e-01, -5.73655427e-01, -9.06810355e-01,
         1.01242284e+00],
       [-1.30577509e-01,  8.96734570e-01, -2.46549176e-01,
         2.02702928e-01, -2.11594223e-01,  8.81756547e-01,
        -4.50495054e-01,  1.00081139e-01, -3.66470091e-01,
         1.01242284e+00],
       [-5.94762135e-01, -5.81850675e-01,  1.41026129e+00,
        -4.38584329e-01, -2.70972558e-01, -8.29372654e-01,
        -8.08709381e-01, -4.14276884e-01, -8.37535962e-01,
        -9.87729597e-01],
       [ 6.83587588e-02,  5.27088259e-01, -4.24064583e-01,
         6.65487546e-01,  1.50189485e+00,  2.04566116e+00,
         5.04743150e-01,  1.80977823e+00,  1.22684094e+00,
```

```
        1.01242284e+00],
       [ 6.83587588e-02, -2.43008223e+00,  2.85997044e-01,
        -2.20414438e-01, -2.11594223e-01, -6.86252377e-01,
        -8.56471291e-01, -3.49076572e-01, -8.51390840e-01,
        -9.87729597e-01],
       [ 5.32543384e-01,  5.27088259e-01, -2.46549176e-01,
        -4.98085209e-01, -9.32616855e-01, -3.87383562e-01,
        -7.60947470e-01, -8.05478761e-01, -8.65245719e-01,
         1.01242284e+00],
       [-6.42654199e-02,  1.57441947e-01, -4.24064583e-01,
         7.05154799e-01,  1.95571498e-01, -2.94776323e-01,
         5.52505060e-01,  1.31473882e-02,  3.67838474e-01,
        -9.87729597e-01],
       [-5.94762135e-01, -1.69078961e+00, -4.24064583e-01,
         3.21704687e-01, -6.95103517e-01, -6.77833537e-01,
        -5.22137919e-01, -7.25789490e-01, -7.12842055e-01,
         1.01242284e+00],
       [-2.63201688e-01,  8.96734570e-01, -3.64892781e-01,
         6.38675421e-02, -4.83038037e-01, -7.11508896e-01,
        -5.46018875e-01, -5.73655427e-01, -6.85132298e-01,
        -9.87729597e-01],
       [ 2.04666942e-03, -5.81850675e-01, -2.46549176e-01,
        -3.53005904e-02, -7.20551375e-01, -8.10430265e-01,
        -7.13185560e-01, -7.76500844e-01, -8.37535962e-01,
         1.01242284e+00],
       [-9.26322582e-01, -3.53902117e+00,  7.59371462e-01,
        -1.04681554e+00, -1.94628985e-01, -4.29477761e-01,
        -9.75876066e-01, -3.12854176e-01, -8.92955476e-01,
        -9.87729597e-01],
       [-5.94762135e-01, -9.51496987e-01, -4.24064583e-01,
         7.31599634e-01, -3.64281369e-01, -3.78964722e-01,
         4.09219329e-01, -3.85298968e-01,  4.91762665e-02,
         1.01242284e+00],
       [-5.28450045e-01, -5.81850675e-01, -4.24064583e-01,
         5.99375458e-01, -5.00003276e-01, -6.33634628e-01,
         7.48859580e-02, -5.80899906e-01, -2.69485941e-01,
        -9.87729597e-01],
       [-1.32419512e+00, -2.12204364e-01,  7.00199660e-01,
         3.41538313e-01, -4.06694465e-01, -8.12534974e-01,
        -3.54971234e-01, -5.22944072e-01, -5.18873755e-01,
        -9.87729597e-01],
       [ 6.83587588e-02, -2.12204364e-01, -4.24064583e-01,
         6.32431502e-01,  4.71256622e-04,  3.21903696e-01,
         2.42052644e-01,  1.50792493e-01,  2.15434809e-01,
         1.01242284e+00],
       [ 1.26197637e+00,  2.00567350e+00, -4.24064583e-01,
         5.13429743e-01,  2.78277035e+00,  3.19272809e+00,
        -6.83997726e-02,  3.12102896e+00,  8.80468981e-01,
         1.01242284e+00],
       [ 1.12935219e+00,  1.57441947e-01, -1.87377374e-01,
        -1.14598367e+00, -9.15651616e-01, -4.50524861e-01,
        -9.04233201e-01, -8.05478761e-01, -8.92955476e-01,
         1.01242284e+00],
       [-7.93698403e-01,  1.57441947e-01,  1.82446390e+00,
         2.42002891e-02, -5.25451133e-01, -8.20953814e-01,
        -6.41542695e-01, -6.17122302e-01, -7.68261569e-01,
        -9.87729597e-01],
       [-2.63201688e-01,  1.57441947e-01,  4.93098352e-02,
        -1.93969602e-01, -6.10277325e-01, -8.46210334e-01,
        -7.37066515e-01, -6.89567094e-01, -8.37535962e-01,
        -9.87729597e-01],
       [-6.61074224e-01,  1.26638088e+00, -4.24064583e-01,
         7.97711723e-01,  2.46467213e-01,  1.05434276e+00,
         1.26893371e+00,  5.56483328e-01,  1.17142143e+00,
         1.01242284e+00],
       [-1.19157094e+00, -5.81850675e-01, -4.24064583e-01,
         1.10146004e-01, -7.96894947e-01, -8.06220845e-01,
```

```
 -6.41542695e-01, -8.34456678e-01, -8.23681083e-01,
  1.01242284e+00],
[-5.28450045e-01, -2.12204364e-01, -4.24064583e-01,
  8.04322932e-01,  1.19652056e+00,  3.93463835e-01,
  8.62957476e-01,  8.75240413e-01,  1.06058240e+00,
 -9.87729597e-01],
[-1.45681930e+00, -2.12204364e-01, -4.24064583e-01,
  8.10934140e-01, -7.58723161e-02, -5.47341519e-01,
  8.86838431e-01, -2.25920425e-01,  4.37112866e-01,
 -9.87729597e-01],
[-5.28450045e-01, -5.81850675e-01, -4.24064583e-01,
  7.77878096e-01,  3.73706501e-01,  3.89254415e-01,
  6.48028880e-01,  4.11593744e-01,  6.58790924e-01,
  1.01242284e+00],
[-1.45681930e+00, -2.79972854e+00, -3.64892781e-01,
  4.47317654e-01,  9.08111510e-01,  7.89149308e-01,
 -4.74376009e-01,  9.18707288e-01, -1.72501791e-01,
 -9.87729597e-01],
[-2.63201688e-01,  5.27088259e-01, -4.24064583e-01,
  6.78709964e-01,  5.13669718e-02,  1.78783419e-01,
  5.28624105e-01,  1.21814576e-01,  4.23257988e-01,
  1.01242284e+00],
[-1.72206765e+00, -5.81850675e-01, -4.24064583e-01,
  1.13488337e+00,  1.84119962e+00,  8.05986988e-01,
  2.70179102e+00,  1.42582083e+00,  2.69545807e+00,
 -9.87729597e-01],
[-4.62137956e-01, -2.12204364e-01,  1.29191768e+00,
 -1.54302349e-01, -6.01794706e-01, -8.20953814e-01,
 -7.37066515e-01, -6.82322615e-01, -8.37535962e-01,
 -9.87729597e-01],
[-9.92634671e-01, -9.51496987e-01, -4.24064583e-01,
  8.70435020e-01,  6.79080792e-01,  1.26270905e+00,
  9.58481297e-01,  9.54929684e-01,  1.17142143e+00,
  1.01242284e+00],
[ 2.67295027e-01,  1.26638088e+00, -4.24064583e-01,
 -1.14635096e-01, -9.58064712e-01, -6.77833537e-01,
 -6.41542695e-01, -8.85168032e-01, -8.37535962e-01,
 -9.87729597e-01],
[ 9.96728010e-01, -5.81850675e-01, -4.24064583e-01,
  2.55592598e-01, -6.69655660e-01, -3.55812912e-01,
 -4.98256964e-01, -5.66410948e-01, -6.43567662e-01,
  1.01242284e+00],
[ 3.33607116e-01, -5.81850675e-01, -4.24064583e-01,
  5.46485787e-01, -1.77663746e-01, -1.13771266e-01,
 -2.06378624e-02, -1.46231154e-01, -1.17082276e-01,
  1.01242284e+00],
[-2.58412482e+00, -2.43008223e+00, -4.24064583e-01,
  1.24727392e+00,  1.36617295e+00,  5.87097152e-01,
  2.63014815e+00,  1.02737448e+00,  2.37679587e+00,
 -9.87729597e-01],
[ 2.67295027e-01, -5.81850675e-01, -4.24064583e-01,
  3.61371940e-01, -7.12068756e-01, -5.53655649e-01,
 -3.78852189e-01, -6.67833656e-01, -6.02003026e-01,
  1.01242284e+00],
[-9.26322582e-01, -1.69078961e+00, -4.24064583e-01,
  8.10934140e-01, -3.72763988e-01, -3.24242263e-01,
  5.04743150e-01, -3.56321051e-01,  1.18450659e-01,
  1.01242284e+00],
[ 1.26197637e+00, -2.12204364e-01, -1.87377374e-01,
 -2.13105379e+00, -1.12771710e+00, -8.67257434e-01,
 -1.07139989e+00, -1.07352449e+00, -9.48374990e-01,
  1.01242284e+00],
[-1.65575556e+00, -2.12204364e-01, -4.24064583e-01,
  9.89436779e-01,  3.14752298e+00,  2.08775536e+00,
  1.79431473e+00,  2.80227188e+00,  2.77858734e+00,
 -9.87729597e-01],
[-5.28450045e-01, -2.12204364e-01, -4.24064583e-01,
```

```
         7.91100514e-01,  5.68806742e-01, -1.09561846e-01,
         7.91314611e-01,  3.31904473e-01,  7.14210438e-01,
        -9.87729597e-01],
       [-3.29513777e-01, -9.51496987e-01, -3.05720978e-01,
         1.56424466e-01,  3.82189120e-01,  8.31243508e-01,
        -6.17661740e-01,  5.92705724e-01, -4.21889605e-01,
        -9.87729597e-01],
       [ 1.79247308e+00, -2.12204364e-01, -4.24064583e-01,
        -6.83199056e-01, -5.93312087e-01, -7.11508896e-01,
        -8.32590336e-01, -6.60589177e-01, -8.65245719e-01,
        -9.87729597e-01],
       [ 1.34670848e-01, -2.12204364e-01, -3.64892781e-01,
         2.82037434e-01, -1.60698508e-01,  2.86123627e-01,
        -4.50495054e-01,  2.03918674e-02, -3.94179848e-01,
         1.01242284e+00],
       [ 6.65167563e-01, -2.12204364e-01, -4.24064583e-01,
         6.25820293e-01,  1.06079866e+00,  1.65208039e+00,
         2.18171689e-01,  1.36062052e+00,  7.00355559e-01,
         1.01242284e+00],
       [-1.30577509e-01, -9.51496987e-01,  3.45168846e-01,
        -8.41868068e-01, -8.56273282e-01, -8.60943304e-01,
        -8.80352246e-01, -8.92412512e-01, -8.92955476e-01,
         1.01242284e+00],
       [ 1.26197637e+00,  2.37531982e+00,  7.32744151e+00,
        -1.98560720e+00, -9.91995189e-01, -8.71466853e-01,
        -1.04751893e+00, -9.79346262e-01, -9.34520112e-01,
         1.01242284e+00],
       [-1.72206765e+00, -9.51496987e-01, -4.24064583e-01,
         1.01588161e+00,  1.10321175e+00,  8.19667603e-02,
         1.67490995e+00,  7.88306662e-01,  1.60092267e+00,
        -9.87729597e-01],
       [-5.94762135e-01, -5.81850675e-01, -4.24064583e-01,
         9.16713482e-01,  4.38598538e+00,  2.71074951e+00,
         1.26893371e+00,  3.75129865e+00,  2.66774832e+00,
        -9.87729597e-01],
       [ 1.65984890e+00, -9.51496987e-01, -4.24064583e-01,
        -4.05528285e-01, -9.83512570e-01, -6.71519407e-01,
        -8.32590336e-01, -9.06901470e-01, -8.79100598e-01,
         1.01242284e+00],
       [ 2.00982937e-01,  2.37531982e+00, -1.87377374e-01,
        -5.90642132e-01, -3.13385653e-01,  4.79756943e-01,
        -6.89304605e-01, -5.92974038e-02, -6.57422540e-01,
         1.01242284e+00],
       [ 3.99919206e-01,  5.27088259e-01,  2.35701012e+00,
        -9.87314662e-01, -8.73238520e-01, -8.52524464e-01,
        -8.56471291e-01, -8.99656991e-01, -8.92955476e-01,
        -9.87729597e-01],
       [-1.32419512e+00,  1.57441947e-01, -4.24064583e-01,
         9.29935899e-01,  2.70642678e+00,  1.87728436e+00,
         1.62714804e+00,  2.46178136e+00,  2.44607026e+00,
        -9.87729597e-01],
       [ 8.64103831e-01,  1.63602719e+00, -9.86196704e-03,
        -9.27813783e-01, -9.83512570e-01, -8.27267944e-01,
        -8.08709381e-01, -9.57612824e-01, -8.79100598e-01,
         1.01242284e+00],
       [ 1.39460055e+00,  8.96734570e-01, -4.24064583e-01,
         4.40706446e-01, -5.89070777e-02,  1.67207514e-02,
        -4.45188175e-02, -1.58305286e-02, -7.55176406e-02,
         1.01242284e+00],
       [ 6.65167563e-01, -2.12204364e-01, -4.24064583e-01,
         6.25820293e-01, -1.77663746e-01, -2.14797345e-01,
         2.18171689e-01, -1.96942509e-01,  3.53213880e-02,
         1.01242284e+00],
       [ 6.83587588e-02, -2.12204364e-01, -3.05720978e-01,
        -6.50143012e-01, -8.81721139e-01, -6.73624117e-01,
        -8.32590336e-01, -8.27212199e-01, -8.79100598e-01,
        -9.87729597e-01],
```

```
       [ 3.99919206e-01, -9.51496987e-01, -9.86196704e-03,
        -6.96421474e-01, -6.95103517e-01, -8.25163234e-01,
        -8.56471291e-01, -7.54767407e-01, -8.79100598e-01,
         1.01242284e+00],
       [-6.61074224e-01,  1.57441947e-01, -4.24064583e-01,
         8.17545349e-01,  1.98540415e+00,  2.37820533e+00,
         1.36445753e+00,  2.26618042e+00,  2.14126293e+00,
         1.01242284e+00],
       [ 3.33607116e-01, -2.12204364e-01, -4.24064583e-01,
         5.92764249e-01,  8.23285318e-01,  6.51290806e-02,
         1.22647868e-01,  5.56483328e-01,  3.12418959e-01,
        -9.87729597e-01],
       [-5.94762135e-01, -1.69078961e+00, -3.64892781e-01,
         4.34095237e-01, -4.91520657e-01, -4.67362540e-01,
        -3.78852189e-01, -5.01210635e-01, -5.18873755e-01,
        -9.87729597e-01],
       [-5.94762135e-01,  8.96734570e-01, -3.64892781e-01,
         1.69646883e-01, -8.39308043e-01, -7.00985346e-01,
        -7.13185560e-01, -8.05478761e-01, -8.51390840e-01,
        -9.87729597e-01],
       [-1.72206765e+00,  1.57441947e-01, -4.24064583e-01,
         9.96047988e-01,  3.99578490e+00,  2.49817380e+00,
         2.00924332e+00,  3.46151949e+00,  3.24965322e+00,
        -9.87729597e-01],
       [ 9.30415920e-01,  2.00567350e+00,  6.41027858e-01,
        -1.23192939e+00, -1.04289090e+00, -8.44105624e-01,
        -8.56471291e-01, -1.00107970e+00, -9.06810355e-01,
         1.01242284e+00],
       [-5.94762135e-01,  8.96734570e-01, -4.24064583e-01,
         8.10934140e-01,  4.92463170e-01,  1.31322209e+00,
         1.19729085e+00,  8.17284579e-01,  1.28226046e+00,
         1.01242284e+00]])
```

Model Training

In [56]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [57]:
```
regression.fit(x_train,y_train)
```

Out[57]:
```
LinearRegression()
```

Coefficient

In [58]:
```
print(regression.coef_)
```

```
[[-1.45732761 -0.717256   -0.25440877  0.93258152 -0.086711    0.37465144
   0.27747737  0.4158806  -0.43324618 -0.21483906]]
```

Intercept

In [59]:
```
print(regression.intercept_)
```

```
[32.07407407]
```

Prediction for test data

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

Out[60]:
```
array([[31.91542619],
       [33.41347513],
       [33.75705803],
       [25.63417825],
       [28.58107364],
       [33.58644509],
       [31.59658411],
       [34.54785466],
       [31.92804418],
       [33.46312854],
       [34.17191426],
       [32.87181188],
       [35.69800686],
       [32.06946942],
       [34.19782311],
       [33.31076196],
       [27.5787481 ],
       [35.8730327 ],
       [32.71447703],
       [24.44223792],
       [32.07952363],
       [32.48151929],
       [33.10475631],
       [33.39861071],
       [30.06510796],
       [32.90452085],
       [34.01239466],
       [31.8611212 ],
       [31.90160855],
       [34.84713309],
       [34.01757119],
       [33.8425015 ],
       [34.14831209],
       [32.76392984],
       [31.01797214],
       [28.78834064],
       [32.52461299],
       [31.97093079],
       [33.21312226],
       [33.81987361],
       [34.25420282],
       [35.18995362],
       [34.05634613],
       [37.4695465 ],
       [32.67941211],
       [36.69665871],
       [32.25857945],
       [35.47503603],
       [30.63524534],
       [30.99735899],
       [32.36222047],
       [39.29508339],
       [32.06152996],
       [35.13479292],
       [27.66994587],
       [36.8508119 ],
       [34.01206441],
       [34.20878638],
       [28.95100031],
       [32.3437525 ],
       [32.58085649],
       [31.37955199],
       [24.06427869],
       [36.5676846 ],
```

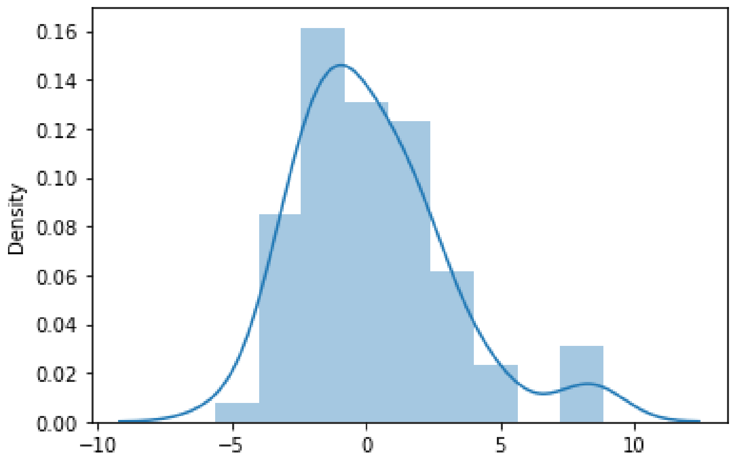
```
       [35.92484593],
       [29.45616188],
       [29.63261858],
       [29.33639913],
       [35.96246726],
       [28.09464045],
       [29.72502191],
       [31.62917964],
       [31.44022863],
       [30.88970589],
       [34.68956932],
       [32.69611482],
       [34.64218595],
       [32.40703152],
       [36.89841364],
       [27.26318207],
       [33.51016711]])
```

Residuals

Deviation from actual(test values) to prediction (values based on algorithm input)

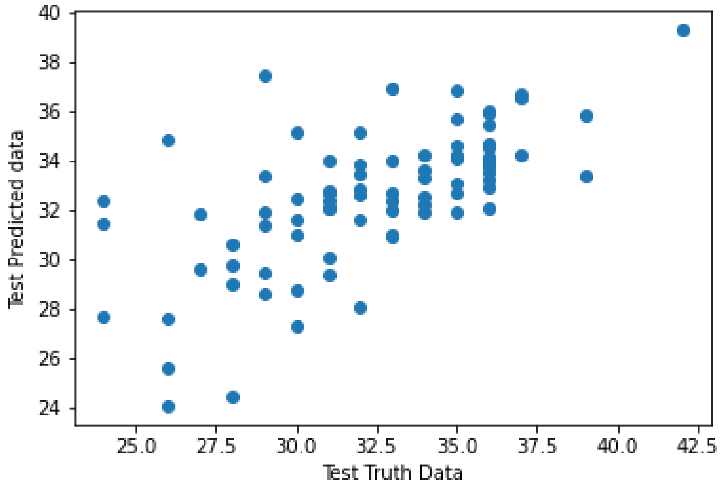
In [61]:
```
sns.distplot(reg_pred - y_test)
```

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



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')
```



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

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

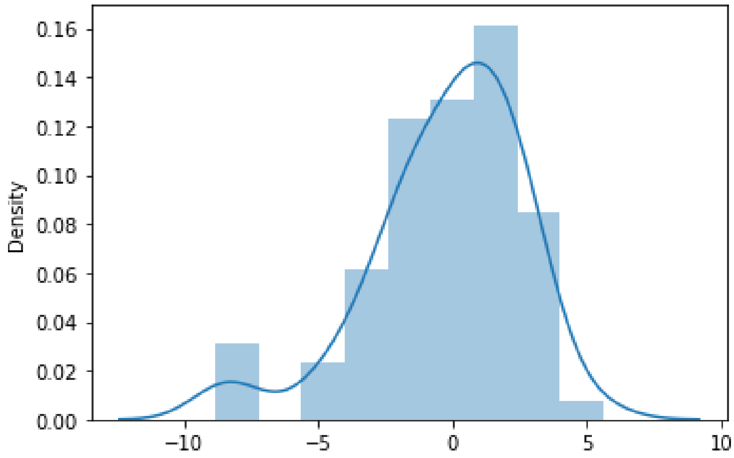
Out[64]:

	Temperature
46	-2.915426
226	-4.413475
181	2.242942
116	0.365822
124	0.418926
...	...
127	0.357814
242	-8.407032
208	-3.898414
102	2.736818
78	2.489833

81 rows × 1 columns

In [65]:
```
sns.distplot(residual, kde = True)
```

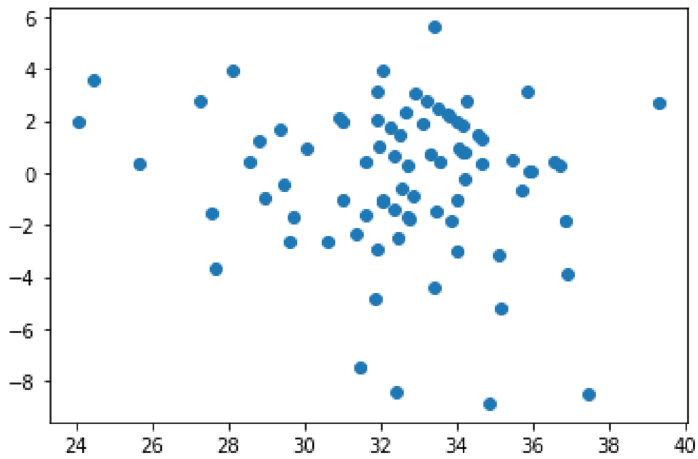
Out[65]: <AxesSubplot:ylabel='Density'>



Scatter plot with residual & preddiction

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

Out[66]: <matplotlib.collections.PathCollection at 0x15af3ec3ca0>



Perfomance 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)))
```

```
8.174371809630447
2.172275225121574
2.8590858346035097
```

R Square & Adusjested R squre

In [68]:
```
from sklearn.metrics import r2_score
score = r2_score(y_test,reg_pred)
print(score)
```

```
0.4094946991655799
```

In [69]:
```
1-(1-score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)  ## Adjuested r square value
```

Out[69]: 0.32513679904637705

Ridege Regression Algorithm

In [70]:
```
from sklearn.linear_model import Ridge
```

In [71]:
```
ridge = Ridge()
```

In [72]:
```
ridge.fit(x_train,y_train)
```

Out[72]:
```
Ridge()
```

In [73]:
```
### Coefficet

print(ridge.coef_)
```

```
[[-1.43943941 -0.7160425  -0.25512583  0.93891733  0.00628512  0.39583232
   0.21860463  0.25319759 -0.32843448 -0.2115963 ]]
```

In [74]:
```
### Intercept

print(ridge.intercept_)
```

```
[32.07407407]
```

In [75]:
```
ridge_pred = ridge.predict(x_test)
```

In [76]:
```
ridge_pred
```

Out[76]:
```
array([[31.91887704],
       [33.40547855],
       [33.7841496 ],
       [25.65877916],
       [28.59628135],
       [33.61269672],
       [31.57237247],
       [34.58220054],
       [31.94550746],
       [33.43851023],
       [34.14495825],
       [32.87807079],
       [35.68376782],
       [32.05328506],
       [34.12572945],
       [33.32321603],
       [27.6017316 ],
       [35.85757436],
       [32.70857194],
       [24.44565836],
       [32.05384911],
       [32.44763793],
       [33.10187308],
       [33.37584311],
       [30.06579191],
       [32.92096234],
       [34.00042913],
       [31.8486332 ],
       [31.90052827],
       [34.80344826],
       [34.01584014],
       [33.83517025],
```

```
       [34.11938297],
       [32.76314968],
       [30.96480422],
       [28.8008761 ],
       [32.49763677],
       [31.95505986],
       [33.21426994],
       [33.79814275],
       [34.28426514],
       [35.1775836 ],
       [34.06163911],
       [37.40153722],
       [32.68521823],
       [36.74923227],
       [32.2382069 ],
       [35.46612357],
       [30.62847251],
       [31.00390306],
       [32.3683325 ],
       [39.31753237],
       [32.06138777],
       [35.12402939],
       [27.69725536],
       [36.89094403],
       [34.02931899],
       [34.14861455],
       [28.97112893],
       [32.32424147],
       [32.57290377],
       [31.37931447],
       [24.08751857],
       [36.58453813],
       [35.97633862],
       [29.48444481],
       [29.60111185],
       [29.33696858],
       [35.9935057 ],
       [28.11176783],
       [29.74959355],
       [31.65019828],
       [31.42927662],
       [30.89353381],
       [34.69714026],
       [32.71561768],
       [34.62315379],
       [32.38650331],
       [36.95489951],
       [27.28065096],
       [33.51390328]])
```

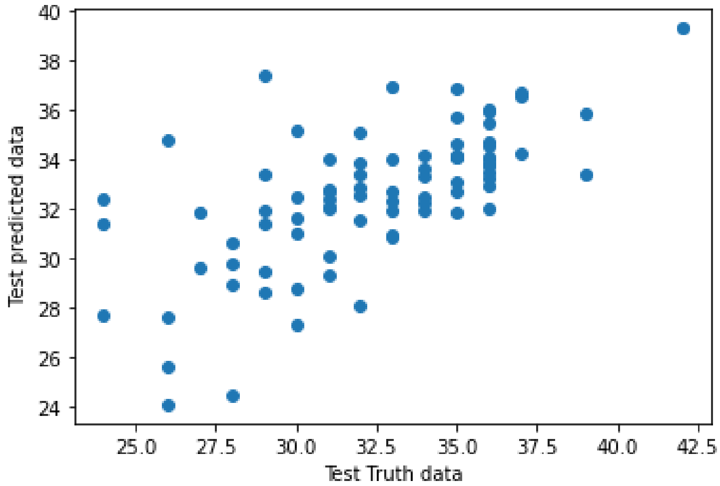
Assumption on Ridge Regression

In [77]:

```
plt.scatter(y_test,ridge_pred)
plt.xlabel('Test Truth data')
plt.ylabel(' Test predicted data')
```

Out[77]:

```
Text(0, 0.5, ' Test predicted data')
```



In [78]:

```
## Residual

residual = y_test - ridge_pred
```

In [79]:

```
residual
```

Out[79]:

	Temperature
46	-2.918877
226	-4.405479
181	2.215850
116	0.341221
124	0.403719
...	...
127	0.376846
242	-8.386503
208	-3.954900
102	2.719349
78	2.486097

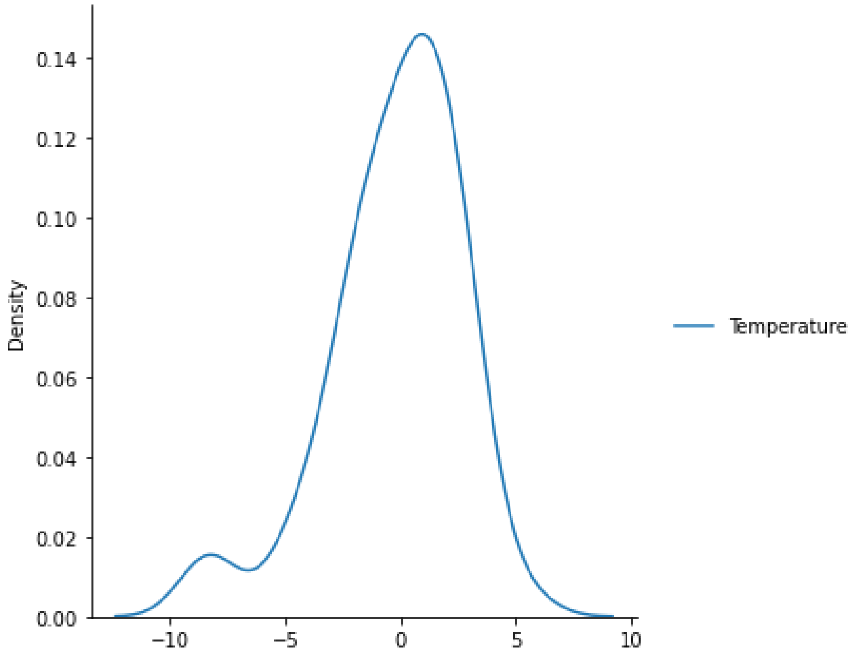
81 rows × 1 columns

In [80]:

```
sns.displot(residual , kind = 'kde' )
```

Out[80]:

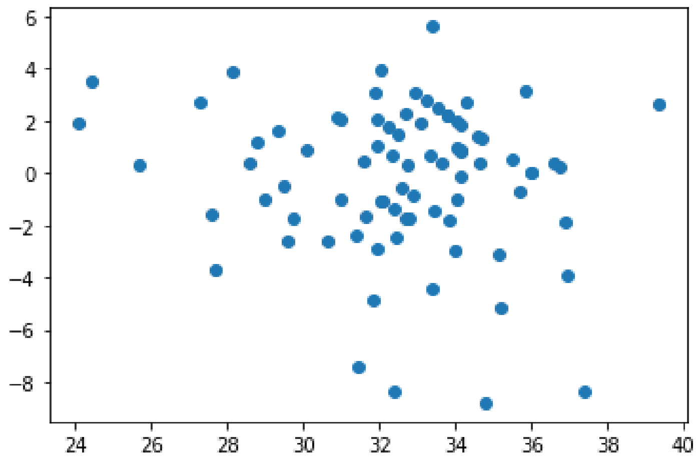
```
<seaborn.axisgrid.FacetGrid at 0x15af3eaaf40>
```



Scatter plot with residual & preddiction

In [81]:
```
plt.scatter(ridge_pred, residual)
```

Out[81]: <matplotlib.collections.PathCollection at 0x15af3e114c0>



Performance Matrics

In [82]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,ridge_pred))
print(mean_absolute_error(y_test,ridge_pred))
print(np.sqrt(mean_squared_error(y_test,ridge_pred)))
```

```
8.14652515846668
2.1690799660862154
2.8542118278899133
```

R square

In [83]:
```
from sklearn.metrics import r2_score
ridge_score = r2_score(y_test,ridge_pred)
print(score)
```

```
0.4094946991655799
```

Adjusted R Square

In [84]:
```
1-(1-ridge_score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)  ## Adjuested r square v
```

Out[84]:
```
0.3274357745001335
```

Lasso Regression

In [85]:
```
from sklearn.linear_model import Lasso
```

In [86]:
```
lasso = Lasso()
```

In [87]:
```
lasso.fit(x_train,y_train)
```

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

Coefficients and Intercepts

In [88]:
```
print(lasso.coef_)
```

```
[-0.88423537 -0.          -0.           0.88313134  0.           0.
  0.           0.           0.          -0.        ]
```

In [89]:
```
print(lasso.intercept_)
```

```
[32.07407407]
```

In [90]:
```
## prediction for test data

lasso_pred = lasso.predict(x_test)
```

In [91]:
```
lasso_pred
```

Out[91]:
```
array([31.99263189, 33.56261409, 33.16968133, 29.35085197, 29.41532603,
       33.00570181, 32.24618859, 33.57987995, 32.17946599, 33.03180429,
       32.43352237, 32.64830012, 34.42463082, 31.61671515, 32.83763259,
       33.35142638, 29.33942467, 34.79854886, 32.38506498, 29.63719153,
       32.36854863, 32.21265622, 32.60134175, 31.81897394, 31.16330572,
       32.75364414, 32.88409128, 32.36320974, 32.04108928, 31.96868565,
       33.24608236, 33.07067555, 33.54659742, 32.57214892, 31.41161583,
       30.06340682, 32.79726231, 32.13550568, 33.36310351, 33.22497663,
       33.25167109, 34.07840658, 33.22831682, 33.75728546, 32.90619636,
       34.59903828, 32.34644356, 33.72050521, 31.73648451, 31.41845375,
       32.26170559, 35.46055533, 32.15686124, 33.60932262, 29.07618954,
       34.41195434, 33.23999395, 32.50358516, 29.88575148, 32.20406944,
       32.0385909 , 31.44605525, 29.20463799, 34.49394409, 33.4095622 ,
       30.24824223, 31.37474327, 30.84852284, 34.06622977, 30.49062147,
       31.23012062, 32.0385909 , 31.43946717, 31.10541974, 33.38061921,
       32.30257555, 32.9833469 , 32.74980427, 34.4764284 , 30.16341195,
       33.31614515])
```

Performance Matrics

In [92]:
```
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)))
```

```
8.69462464944341
2.370968686727018
2.9486648927003234
```

R Square

In [93]:
```
from sklearn.metrics import r2_score
lasso_score = r2_score(y_test,lasso_pred)
print(lasso_score)
```

```
0.3719123543887275
```

Adjusted R square

In [94]:
```
1-(1-lasso_score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)  ## Adjuested r square
```

Out[94]:
```
0.28218554787283134
```

Elastic - Net regression

In [95]:
```
from sklearn.linear_model import ElasticNet
```

In [96]:
```
elastic = ElasticNet()
```

In [97]:
```
elastic.fit(x_train,y_train)
```

Out[97]:
```
ElasticNet()
```

Coefficeient & Intercepts

In [98]:
```
print(elastic.coef_)
```

```
[-0.77155493 -0.27327033 -0.02945645  0.70980198  0.11177449  0.
  0.20914599  0.04593431  0.12829249 -0.        ]
```

In [99]:
```
print(elastic.intercept_)
```

```
[32.07407407]
```

In [100…
```
## Prediction for test data

elastic_pred = elastic.predict(x_test)
```

In [101…
```
elastic_pred
```

Out[101…
```
array([31.9573208 , 33.23686908, 33.35699564, 28.69175409, 29.50702659,
       33.21952486, 31.75246849, 33.96237735, 32.10800305, 32.71913539,
       32.6464949 , 32.70818669, 34.28160866, 31.98757886, 32.58127398,
```

```
33.38088877, 29.25544946, 35.1329971 , 32.41345187, 28.81942678,
31.92062398, 32.01321369, 32.87611371, 32.19248582, 30.76147509,
32.77885483, 32.9241576 , 31.80574265, 31.84088675, 32.63570681,
33.35824365, 32.97725238, 33.16526411, 32.6259555 , 31.48228091,
29.90876877, 32.28704956, 31.73349961, 33.28556103, 32.87586141,
33.61368581, 34.06687234, 33.48616338, 34.31378768, 32.80354959,
35.56194932, 31.98083723, 34.20080511, 31.06352775, 31.3703163 ,
32.33015611, 36.68465163, 32.02910227, 33.90164802, 29.13016242,
35.33664404, 33.44975962, 32.59500157, 29.89482886, 32.07728159,
32.39209933, 31.39172713, 28.33121558, 35.11153428, 34.62534381,
30.33956758, 30.58992082, 30.41863187, 34.79501455, 29.86618003,
31.0520072 , 32.09680635, 31.20338706, 31.12722538, 34.01998343,
32.49121998, 33.0901132 , 32.02987963, 35.52196854, 29.45679071,
33.38350807])
```

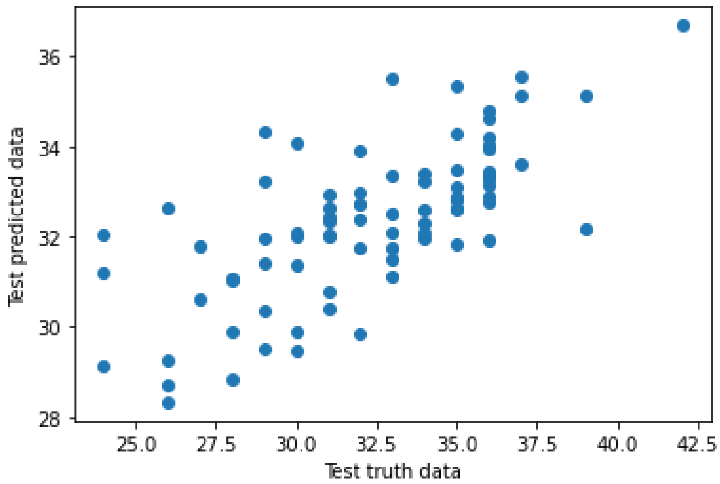
Assumptions of elastic net regression

In [102…

```
plt.scatter(y_test,elastic_pred)
plt.xlabel('Test truth data')
plt.ylabel('Test predicted data')
```

Out[102…

```
Text(0, 0.5, 'Test predicted data')
```



Performance Matrix

In [103…

```
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)))
```

```
7.918267418307841
2.2796482511865195
2.8139416160090884
```

R Square

In [104…

```
from sklearn.metrics import r2_score
elastic_score = r2_score(y_test,elastic_pred)
print(elastic_score)
```

```
0.4279953257782332
```

Adjusted R square

In [105…

```
1-(1-elastic_score)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)  ## Adjuested r square
```

Out[105…

```
0.3462803723179807
```