Algerian_Forest_Fire_Linear_Regression

November 23, 2022

# 1 Linear Regression on ALgerian Forest

# 2 Problem Statement

- To predict the temperature using Algerian Forest Fire Dataset

Dataset: https://archive.ics.uci.edu/ml/datasets/Algerian+Forest+Fires+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€

### 2.1 Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.metrics import r2_score
import missingno as msno

warnings.filterwarnings('ignore')
%matplotlib inline
```

## 3 Data Collection

[2]:
```
df = pd.read_csv("/content/Algerian_forest_fires_dataset_UPDATE.csv", header=1)
df
```

[2]:
```
    day month  year Temperature  RH  Ws Rain   FFMC  DMC    DC  ISI   BUI  \
0    01    06  2012          29  57  18     0  65.7  3.4   7.6  1.3   3.4
1    02    06  2012          29  61  13   1.3  64.4  4.1   7.6    1   3.9
2    03    06  2012          26  82  22  13.1  47.1  2.5   7.1  0.3   2.7
3    04    06  2012          25  89  13   2.5  28.6  1.3   6.9    0   1.7
4    05    06  2012          27  77  16     0  64.8    3  14.2  1.2   3.9
..   ..   ...   ...         ...  ..  ..   ...   ...  ...   ...  ...   ...
241  26    09  2012          30  65  14     0  85.4   16  44.5  4.5  16.9
242  27    09  2012          28  87  15   4.4  41.1  6.5     8  0.1   6.2
243  28    09  2012          27  87  29   0.5  45.9  3.5   7.9  0.4   3.4
244  29    09  2012          24  54  18   0.1  79.7  4.3  15.2  1.7   5.1
245  30    09  2012          24  64  15   0.2  67.3  3.8  16.5  1.2   4.8

      FWI    Classes
0     0.5   not fire
1     0.4   not fire
2     0.1   not fire
3       0   not fire
```

```
4    0.5   not fire
..   ...           ...
241  6.5       fire
242    0   not fire
243  0.2   not fire
244  0.7   not fire
245  0.5  not fire

[246 rows x 14 columns]
```

# 4 Analyzing Data

### Checking Null Values

[3]:
```
df[df.isnull().any(axis=1)]
```

[3]:
```
                                day month  year Temperature   RH   Ws Rain   \
122  Sidi-Bel Abbes Region Dataset   NaN   NaN         NaN  NaN  NaN   NaN
167                              14    07  2012          37   37   18   0.2

     FFMC   DMC      DC  ISI   BUI      FWI Classes
122   NaN   NaN     NaN  NaN   NaN      NaN        NaN
167  88.9  12.9  14.6 9  12.5  10.4  fire           NaN
```

[4]:
```
df.isnull().sum()
```

[4]:
```
day            0
month          1
year           1
Temperature    1
 RH            1
 Ws            1
Rain           1
FFMC           1
DMC            1
DC             1
ISI            1
BUI            1
FWI            1
Classes        2
dtype: int64
```

### Drop rows which have null.

[5]:
```
df.drop([122,123, 167],axis=0, inplace=True)
df = df.reset_index()
df.head()
```

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

         FWI    Classes
      0  0.5  not fire
      1  0.4  not fire
      2  0.1  not fire
      3    0  not fire
      4  0.5  not fire
```

Columns

```
[6]: df.columns
```

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

*Some columns have extra spaces, we have to remove them.*

**Columns name having extra space**

```
[7]: [x for x in df.columns if ' ' in x]
```

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

**Remove extra space in column name**

```
[8]: df.columns = df.columns.str.strip()
     df.columns
```

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

**Function to remove extra space in the data**

```
[9]: import re
     def RemoveExtraSpace(s):
       return s.replace(" ", "")
```

Remove extra space from data.

```
[10]: df['Classes'] = df['Classes'].apply(RemoveExtraSpace)
```

```
[11]: df.head(3)
```

[11]:
```
   index day month  year Temperature  RH  Ws  Rain  FFMC  DMC   DC  ISI  BUI  \
0      0  01    06  2012          29  57  18     0  65.7  3.4  7.6  1.3  3.4
1      1  02    06  2012          29  61  13   1.3  64.4  4.1  7.6    1  3.9
2      2  03    06  2012          26  82  22  13.1  47.1  2.5  7.1  0.3  2.7

   FWI  Classes
0  0.5  notfire
1  0.4  notfire
2  0.1  notfire
```

**Drop extra index column, which was created for reset_index**

[12]:
```
df.drop(['index'],axis=1, inplace=True)
```

**Create one region,just to identify the two region i.e.,Sidi-Bel Abbes Region and Bejaia Region**

[13]:
```
df.loc[:122, 'Region'] = 0
df.loc[122:, 'Region'] = 1
```

**Check Null values in all the features**

[14]:
```
df.isna().sum()
```

[14]:
```
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
```

Map `Classes` feature as 1 and 0 for fire and not fire respectively.

[15]:
```
df['Classes'] = df['Classes'].map({'notfire' : 0, 'fire' : 1})
```

[16]:
```
df.head(5)
```

[16]:
```
  day month  year Temperature  RH  Ws  Rain  FFMC  DMC   DC  ISI  BUI  FWI  \
0  01    06  2012          29  57  18     0  65.7  3.4  7.6  1.3  3.4  0.5
1  02    06  2012          29  61  13   1.3  64.4  4.1  7.6    1  3.9  0.4
```

```
2  03    06  2012           26  82  22  13.1  47.1  2.5   7.1  0.3  2.7  0.1
3  04    06  2012           25  89  13   2.5  28.6  1.3   6.9    0  1.7    0
4  05    06  2012           27  77  16     0  64.8    3  14.2  1.2  3.9  0.5

   Classes  Region
0        0     0.0
1        0     0.0
2        0     0.0
3        0     0.0
4        0     0.0
```

**Check duplictes values in all the column**

[17]:
```
df.duplicated().sum()
```

[17]:
```
0
```

**Check data types**

[18]:
```
df.dtypes
```

[18]:
```
day               object
month             object
year              object
Temperature       object
RH                object
Ws                object
Rain              object
FFMC              object
DMC               object
DC                object
ISI               object
BUI               object
FWI               object
Classes            int64
Region           float64
dtype: object
```

[19]:
```
# Convert features to it's logical datatypes
convert_features = {
    'Temperature' : 'int64','RH':'int64', 'Ws' :'int64', 'DMC': 'float64', 'DC':␣
 ↪'float64', 'ISI' : 'float64',
    'BUI': 'float64', 'FWI' : 'float64', 'Region' : 'object', 'Rain' :␣
 ↪'float64', 'FFMC' : 'float64', 'Classes' : 'object',
    'day' : 'object', 'month' : 'object', 'year' : 'object'
}

df = df.astype(convert_features)
```

[20]:
```
df.head()
```

[20]:
```
  day month  year  Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI  BUI  FWI  \
0  01    06  2012           29  57  18   0.0  65.7  3.4   7.6  1.3  3.4  0.5
1  02    06  2012           29  61  13   1.3  64.4  4.1   7.6  1.0  3.9  0.4
2  03    06  2012           26  82  22  13.1  47.1  2.5   7.1  0.3  2.7  0.1
3  04    06  2012           25  89  13   2.5  28.6  1.3   6.9  0.0  1.7  0.0
4  05    06  2012           27  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5

  Classes Region
0       0    0.0
1       0    0.0
2       0    0.0
3       0    0.0
4       0    0.0
```

[21]:
```
#converted dtypes
df.dtypes
```

[21]:
```
day             object
month           object
year            object
Temperature      int64
RH               int64
Ws               int64
Rain           float64
FFMC           float64
DMC            float64
DC             float64
ISI            float64
BUI            float64
FWI            float64
Classes         object
Region          object
dtype: object
```

Check unique values

[22]:
```
df.nunique()
```

[22]:
```
day             31
month            4
year             1
Temperature     19
RH              62
Ws              18
Rain            39
FFMC           173
```

```
DMC            165
DC             197
ISI            106
BUI            173
FWI            125
Classes          2
Region           2
dtype: int64
```

[23]:
```python
# statistics of data
df.describe().T
```

[23]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Temperature | 243.0 | 32.152263 | 3.628039 | 22.0 | 30.00 | 32.0 | 35.00 | 42.0 |
| RH | 243.0 | 62.041152 | 14.828160 | 21.0 | 52.50 | 63.0 | 73.50 | 90.0 |
| Ws | 243.0 | 15.493827 | 2.811385 | 6.0 | 14.00 | 15.0 | 17.00 | 29.0 |
| Rain | 243.0 | 0.762963 | 2.003207 | 0.0 | 0.00 | 0.0 | 0.50 | 16.8 |
| FFMC | 243.0 | 77.842387 | 14.349641 | 28.6 | 71.85 | 83.3 | 88.30 | 96.0 |
| DMC | 243.0 | 14.680658 | 12.393040 | 0.7 | 5.80 | 11.3 | 20.80 | 65.9 |
| DC | 243.0 | 49.430864 | 47.665606 | 6.9 | 12.35 | 33.1 | 69.10 | 220.4 |
| ISI | 243.0 | 4.742387 | 4.154234 | 0.0 | 1.40 | 3.5 | 7.25 | 19.0 |
| BUI | 243.0 | 16.690535 | 14.228421 | 1.1 | 6.00 | 12.4 | 22.65 | 68.0 |
| FWI | 243.0 | 7.035391 | 7.440568 | 0.0 | 0.70 | 4.2 | 11.45 | 31.1 |

**Segregate categorical feature from the dataset**

[24]:
```python
categorical_features = [feature for feature in df.columns if df[feature].dtypes␣
↪== 'O']
categorical_features
```

[24]: ['day', 'month', 'year', 'Classes', 'Region']

**Check Value_counts() of Classes and Region feature**

[25]:
```python
feature = ['Region', 'Classes']
for x in categorical_features:
  if x in feature:
    print(df.groupby(x)[x].value_counts())
```

```
Classes  Classes
0        0          106
1        1          137
Name: Classes, dtype: int64
Region  Region
0.0     0.0       122
1.0     1.0       121
Name: Region, dtype: int64
```

**Segregate numerical feature from the datasetr**

[26]:
```python
numerical_features = [x for x in df.columns if df[x].dtype != 'O']
numerical_features
```

[26]: ['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI']

**Segregate discrete feature from the numerical feature**

[27]:
```python
## Discrete features are those whose data is whole number means there is no␣
↪decimal value
discrete_features = [x for x in numerical_features if df[x].dtypes == 'int64']
discrete_features
```

[27]: ['Temperature', 'RH', 'Ws']

**Segregate Continuous feature from the numerical feature**

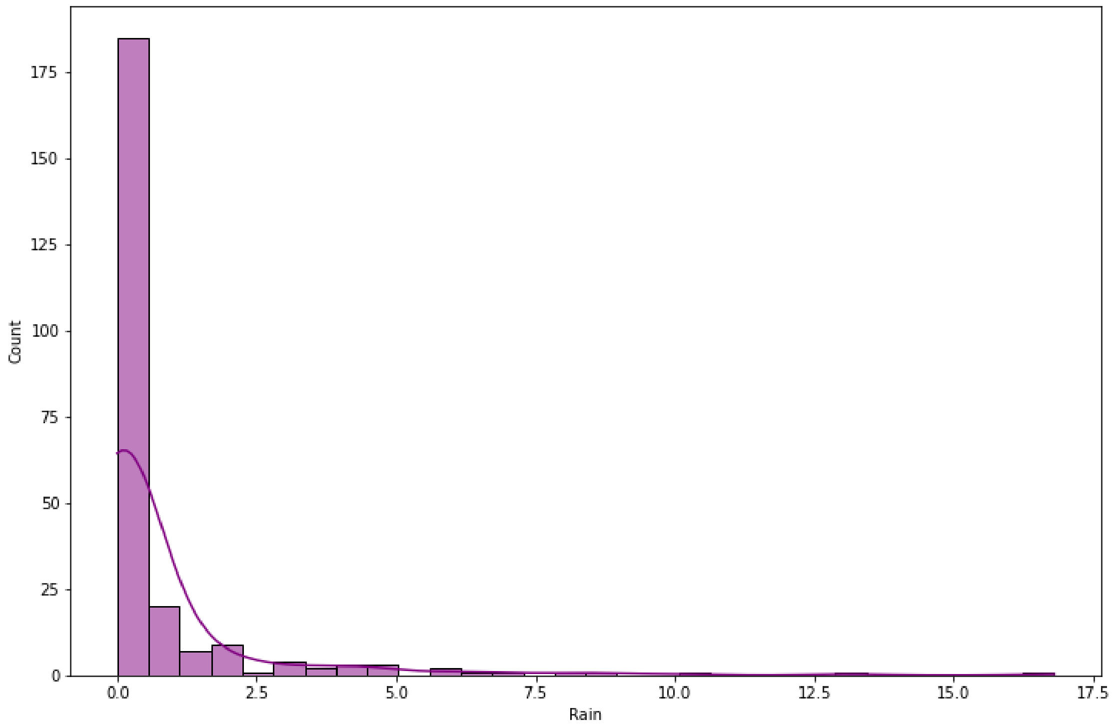
[28]:
```python
## Continuous features are those features where data has decimal value
continuous_feature = [fea for fea in numerical_features if fea not in␣
↪discrete_features]
continuous_feature
```

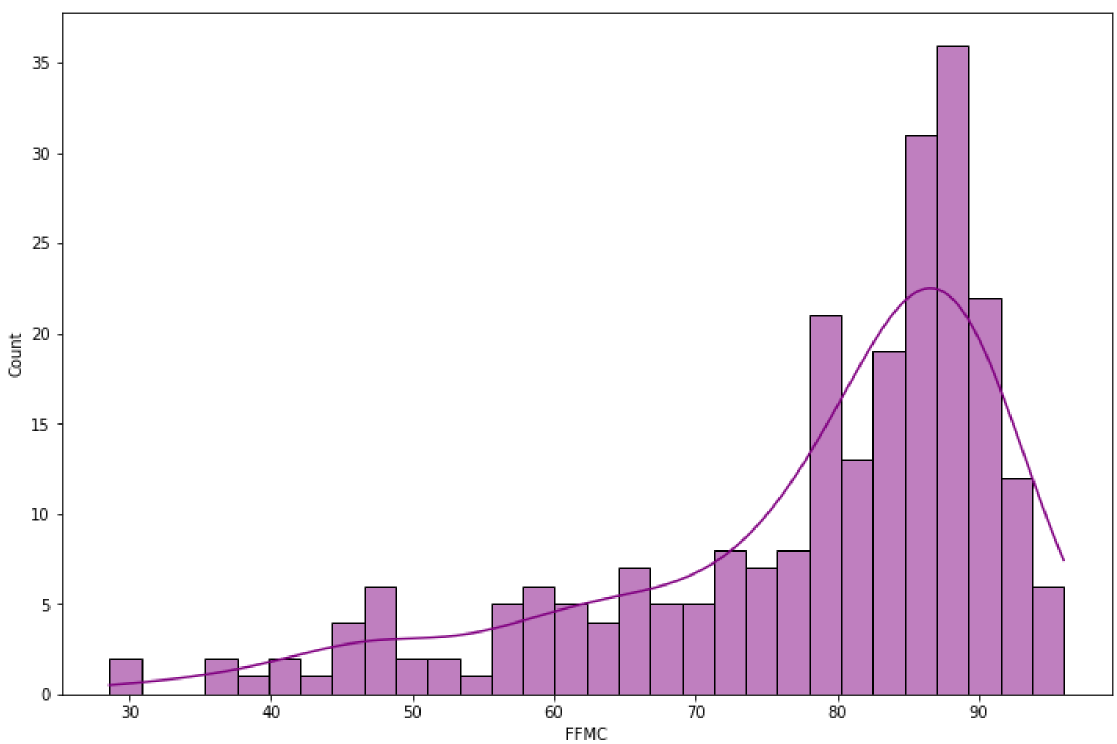
[28]: ['Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI']

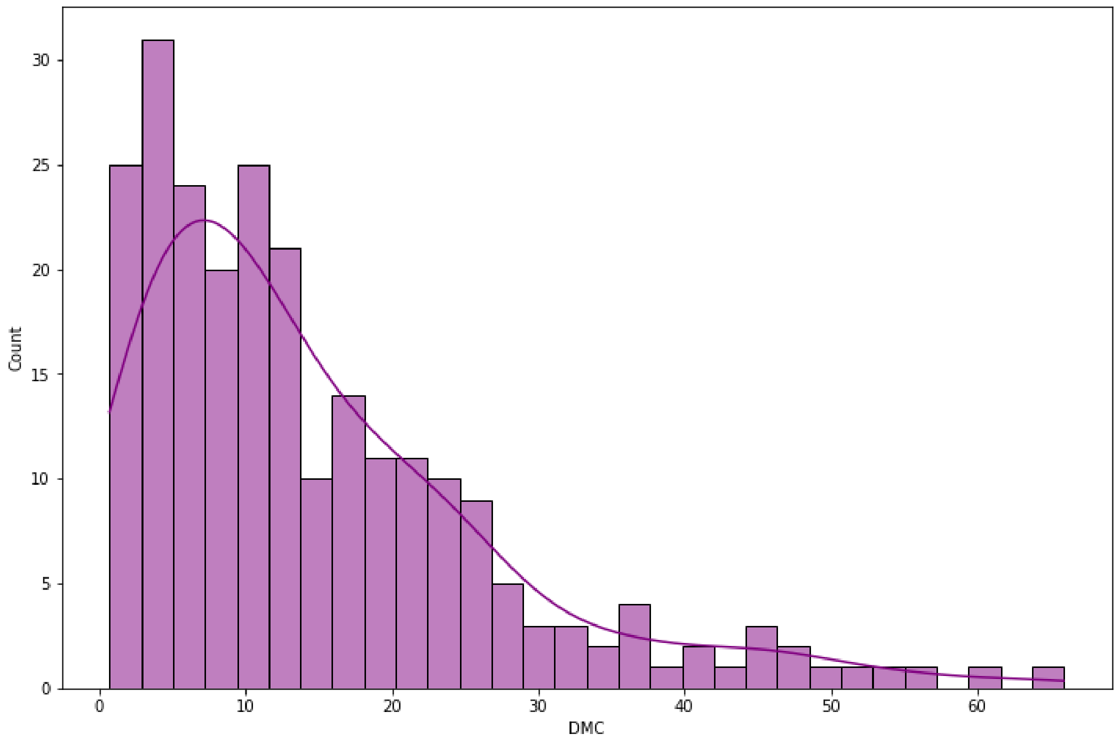
# 5 Graphical Analysis

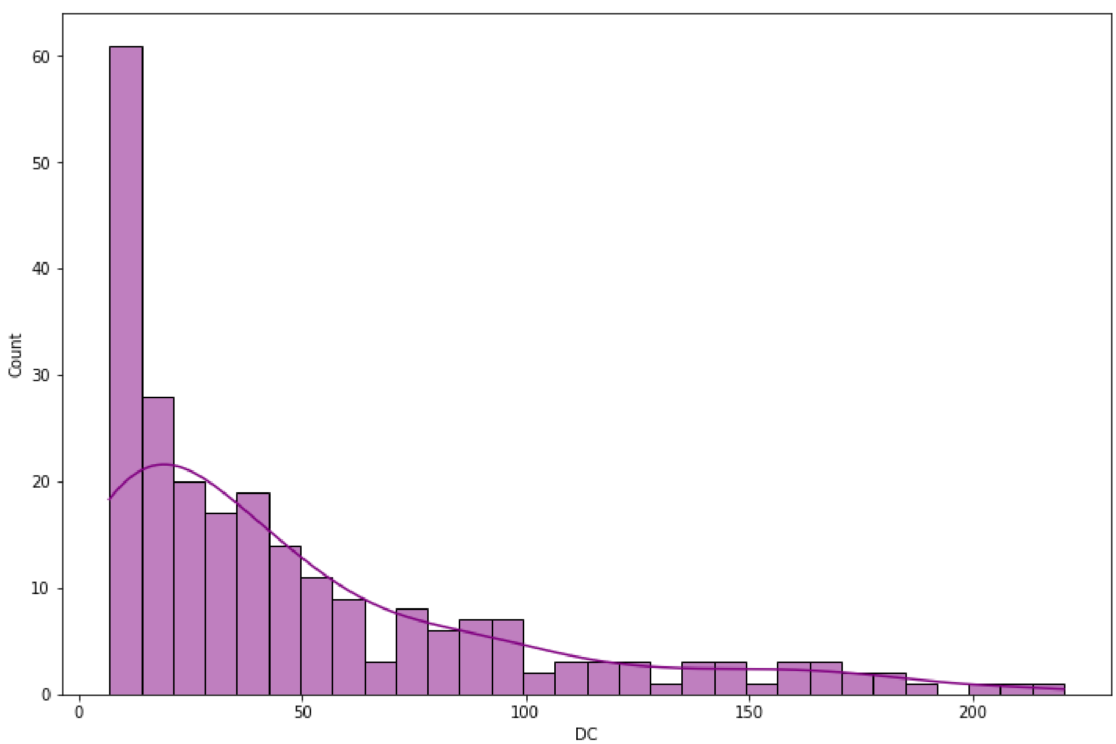
checking distribution of continuous numerical features

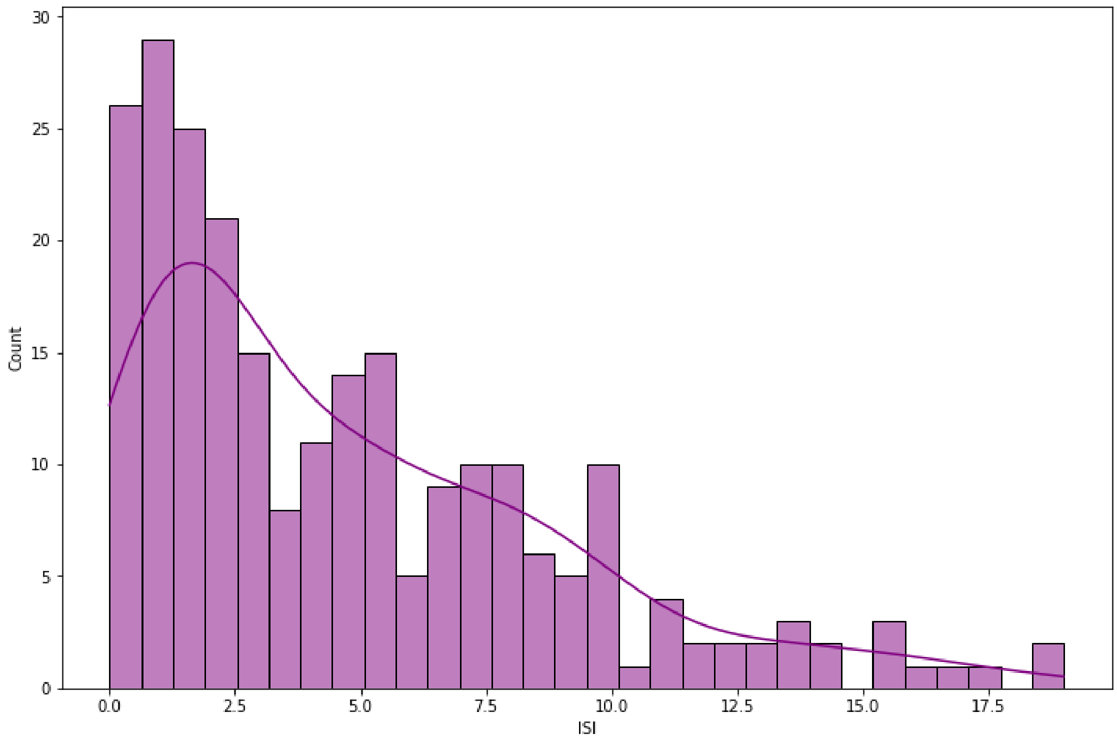
[29]:
```python
for feature in continuous_feature:
  plt.figure(figsize=(12,8))
  sns.histplot(data=df, x= feature, kde=True, bins = 30, color='purple')
  plt.show()
```

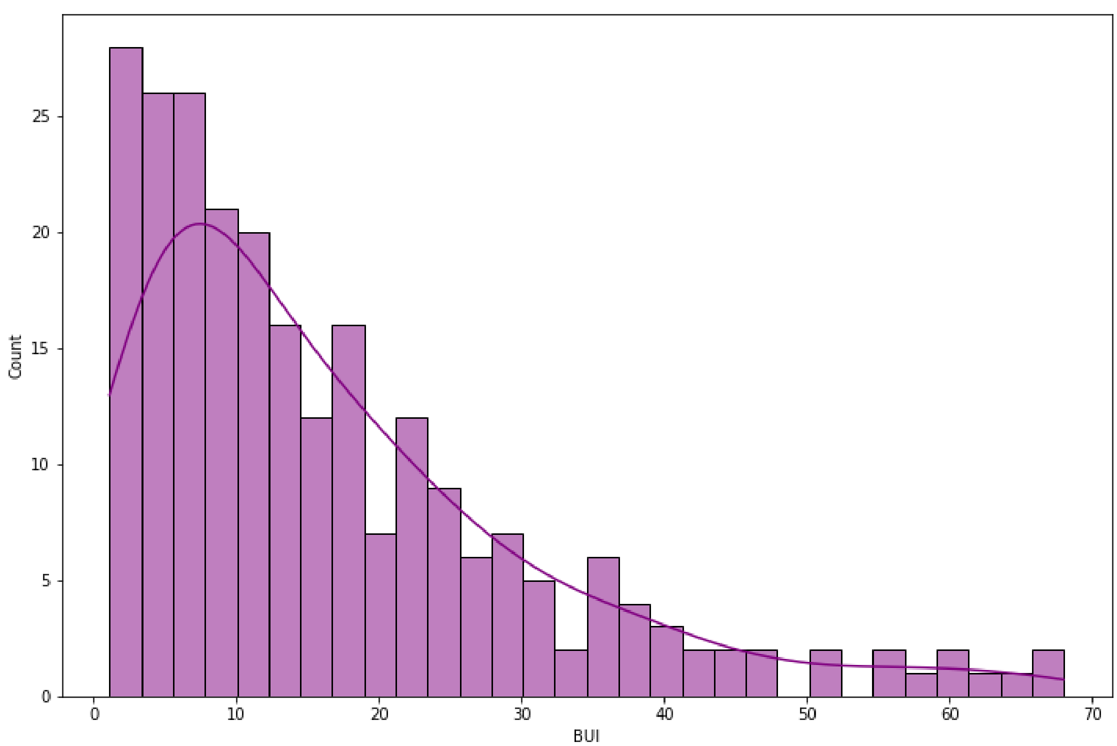

Count
Rain
0
25
50
75
100
125
150
175
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

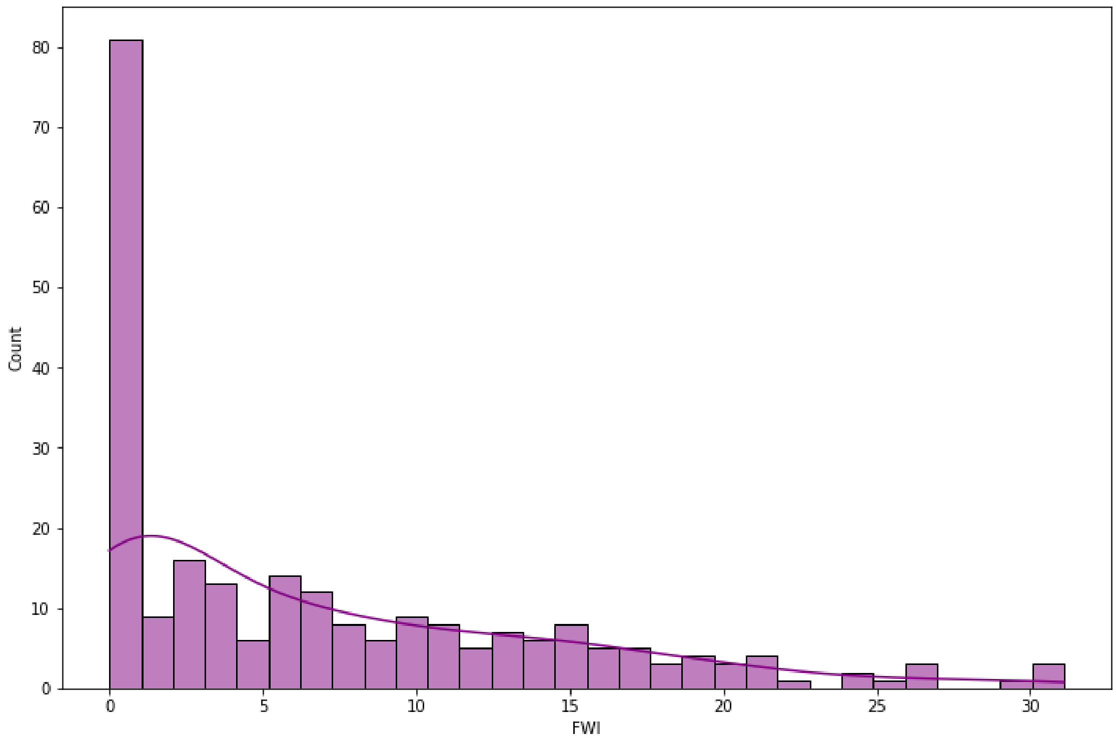

Count
FFMC
0
5
10
15
20
25
30
35
30
40
50
60
70
80
90


Count
DMC
0
5
10
15
20
25
30
0
10
20
30
40
50
60


Count
DC
0
10
20
30
40
50
60
0
50
100
150
200


Count
ISI
0
5
10
15
20
25
30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Count
BUI
0
5
10
15
20
25
0
10
20
30
40
50
60
70

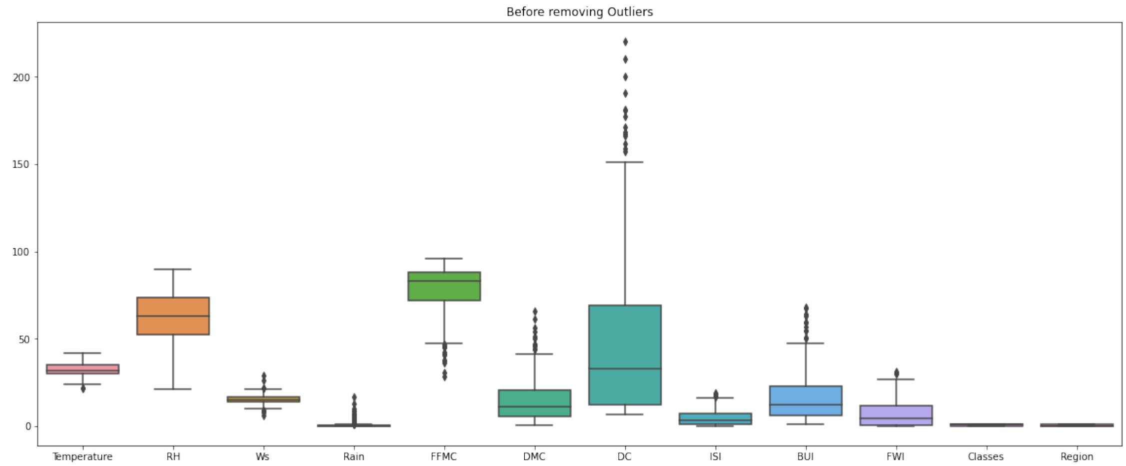



# 6 Outliers Handling

Before Removing Outliers

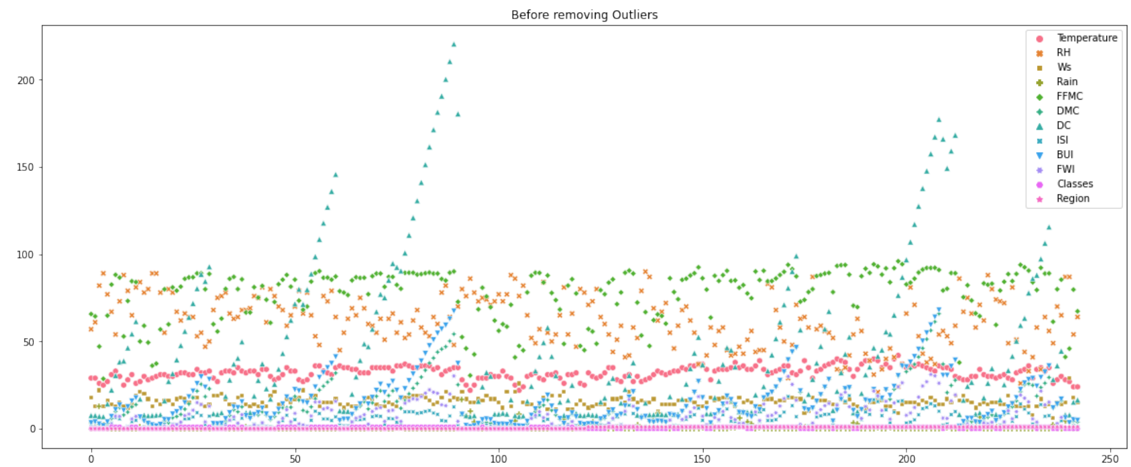
[30]:
```
plt.figure(figsize=(20,8))
sns.boxplot(data=df)
plt.title("Before removing Outliers")
```

[30]: Text(0.5, 1.0, 'Before removing Outliers')



[31]:
```python
plt.figure(figsize=(20,8))
sns.scatterplot(data=df)
plt.title("Before removing Outliers")
```

[31]: Text(0.5, 1.0, 'Before removing Outliers')




**Function to Find upper and lower Boundaries.**

[32]:
```python
def find_boundaries(df, variable):
  IQR = df[variable].quantile(0.75) - df[variable].quantile(0.25)
  lower_boundary = df[variable].quantile(0.25) - (IQR*1.5)
  upper_boundary = df[variable].quantile(0.75) + (IQR*1.5)
  return lower_boundary, upper_boundary
```

[33]:
```python
# Upper and lower boundaries of every feature
for feature in numerical_features:
  print(feature,"------", find_boundaries(df,feature))
```

```
Temperature ------ (22.5, 42.5)
RH ------ (21.0, 105.0)
Ws ------ (9.5, 21.5)
Rain ------ (-0.75, 1.25)
FFMC ------ (47.17499999999999, 112.975)
DMC ------ (-16.699999999999992, 43.29999999999999)
DC ------ (-72.77499999999999, 154.22499999999997)
ISI ------ (-7.374999999999998, 16.025)
BUI ------ (-18.974999999999998, 47.625)
FWI ------ (-15.425, 27.575)
```

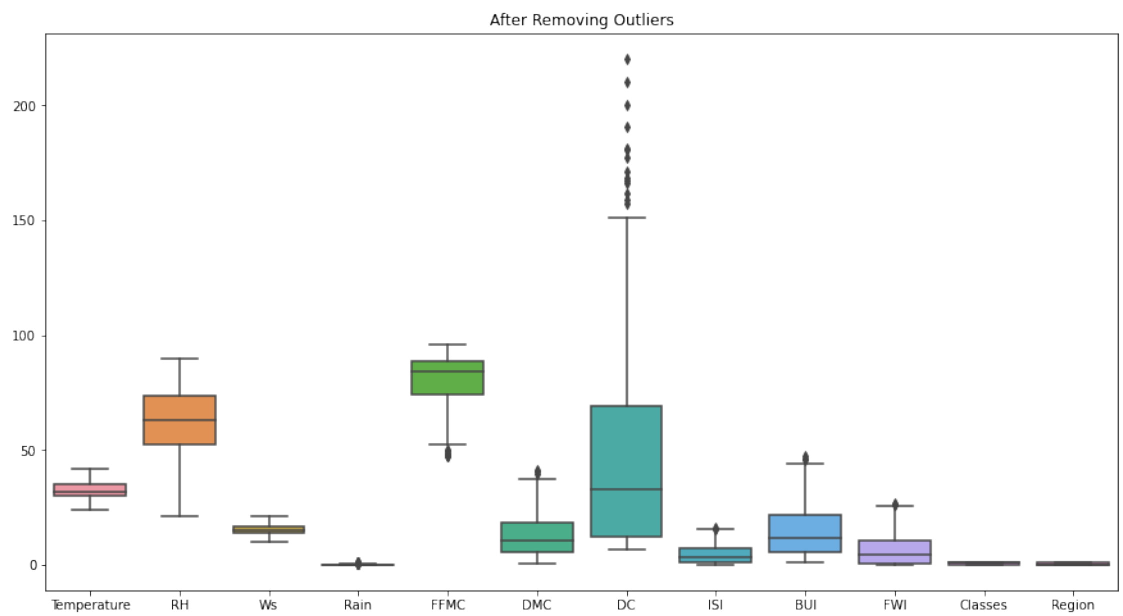
**Deletion of Outliers**

[34]:
```
outliers_columns = ['Temperature', 'Ws', 'Rain', 'FFMC','DMC','ISI','BUI','FWI']
for x in outliers_columns:
  lower_boundary, upper_boundary = find_boundaries(df,x)
  outliers = np.where(df[x] > upper_boundary, True, np.where(df[x] <␣
↪lower_boundary,True, False))
  outliers_df = df.loc[outliers,x]
  df_trimed = df.loc[~outliers, x]
  df[x] = df_trimed
```

After Removal of Outliers

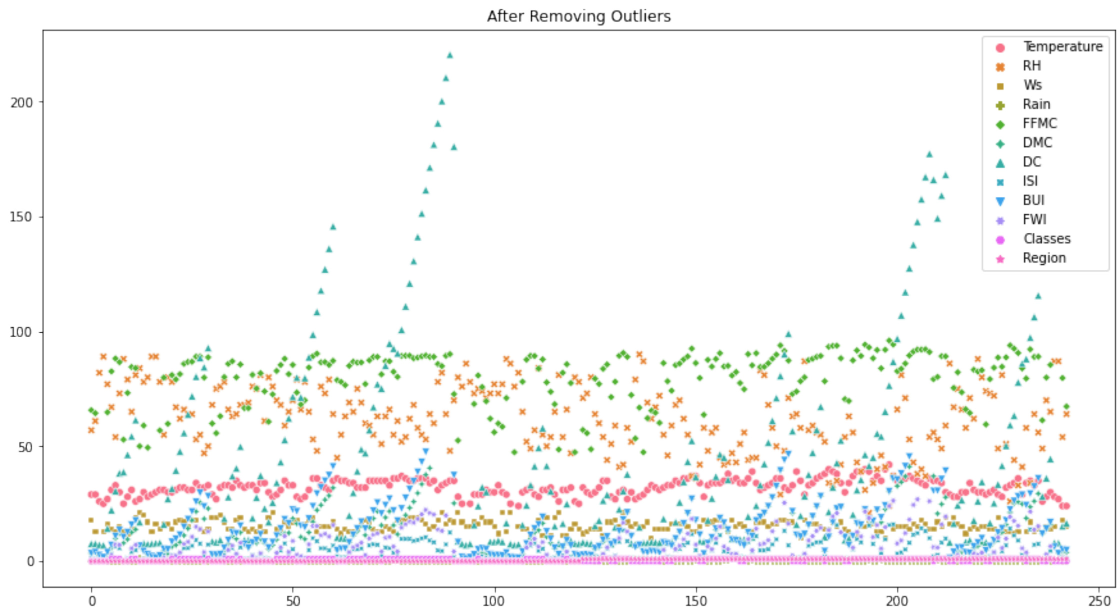
[35]:
```
plt.figure(figsize=(15, 8))
sns.boxplot(data=df)
plt.title("After Removing Outliers")
```

[35]: Text(0.5, 1.0, 'After Removing Outliers')




[36]:
```
plt.figure(figsize=(15, 8))
sns.scatterplot(data=df)
plt.title("After Removing Outliers")
```

[36]: Text(0.5, 1.0, 'After Removing Outliers')

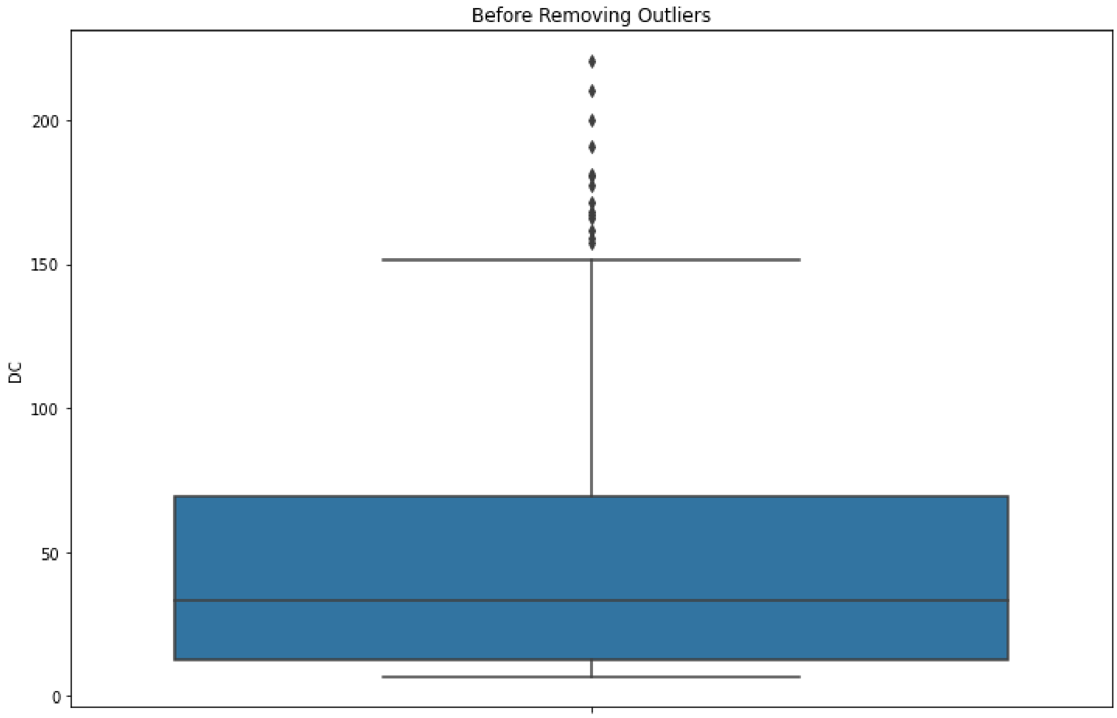



We can see that there are many Outliers still remaining in DC feature.

**Outliers Handling in DC feature**

[37]:
```
plt.figure(figsize = (12,8))
sns.boxplot(data = df, y ="DC")
plt.title("Before Removing Outliers")
```

[37]: Text(0.5, 1.0, 'Before Removing Outliers')




```
[38]: # These are the outliers of DC feature
dc_outliers = df[df['DC'] >= 154]['DC']
dc_outliers
```

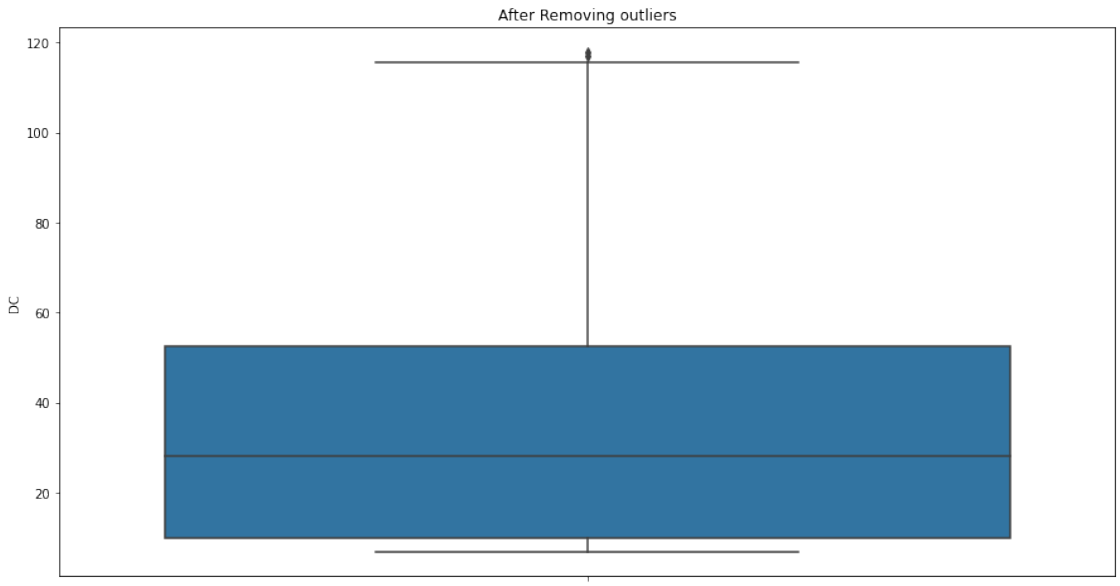
```
[38]: 83      161.5
84      171.3
85      181.3
86      190.6
87      200.2
88      210.4
89      220.4
90      180.4
206     157.5
207     167.2
208     177.3
209     166.0
211     159.1
212     168.2
Name: DC, dtype: float64
```

```
[39]: df['DC'] = df[df['DC'] < 118]['DC']
```

```
[40]: plt.figure(figsize = (15,8))
sns.boxplot(data = df, y = 'DC')
```

```
plt.title("After Removing outliers")
```

[40]: Text(0.5, 1.0, 'After Removing outliers')




Check null value in each column after removing the outliers

[41]:
```
df.isna().sum()
```

[41]:
```
day             0
month           0
year            0
Temperature     2
RH              0
Ws              8
Rain           35
FFMC           13
DMC            12
DC             25
ISI             4
BUI            11
FWI             4
Classes         0
Region          0
dtype: int64
```

Fill all the null values with mean

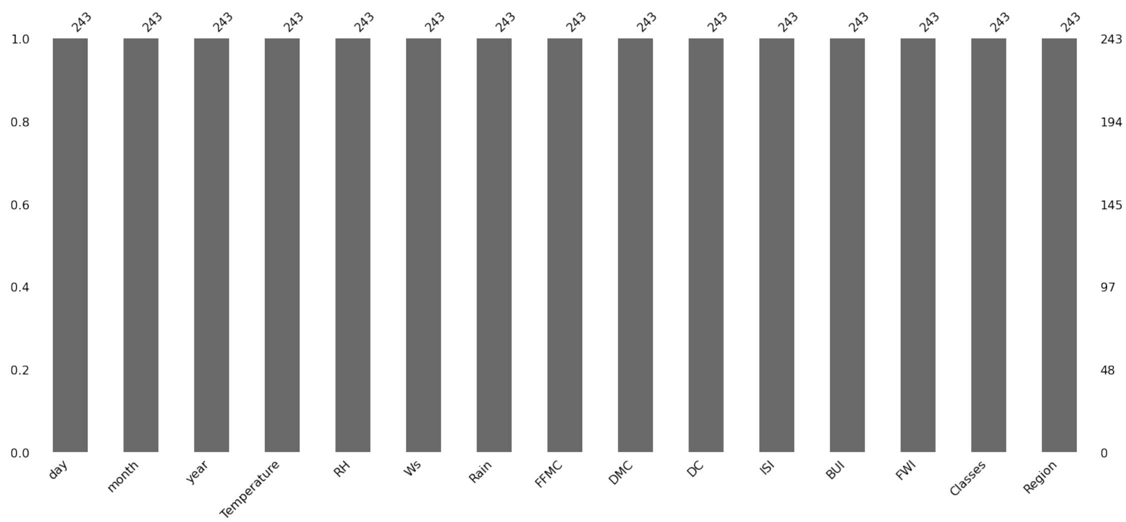
[42]:
```
df.fillna(df.mean().round(1), inplace = True)
```

```
[43]: # check null value of each column
df.isnull().sum()
```

```
[43]: 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
```

```
[44]: msno.bar(df)
```

```
[44]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc46e13ed10>
```




We can see, our data does not contain any null value.

## 7 Statistical Analysis

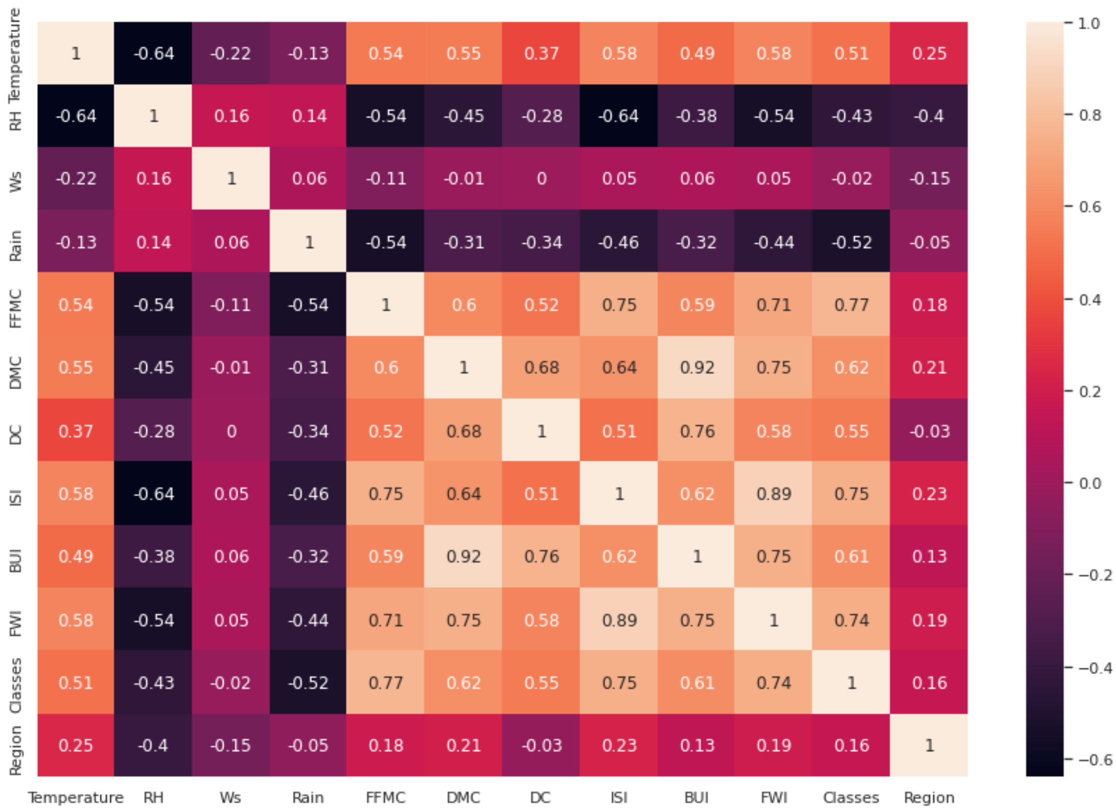
[45]:
```python
data = round(df.corr(), 2)
data
```

[45]:
```
             Temperature    RH    Ws  Rain  FFMC   DMC    DC   ISI   BUI  \
Temperature         1.00 -0.64 -0.22 -0.13  0.54  0.55  0.37  0.58  0.49
RH                 -0.64  1.00  0.16  0.14 -0.54 -0.45 -0.28 -0.64 -0.38
Ws                 -0.22  0.16  1.00  0.06 -0.11 -0.01  0.00  0.05  0.06
Rain               -0.13  0.14  0.06  1.00 -0.54 -0.31 -0.34 -0.46 -0.32
FFMC                0.54 -0.54 -0.11 -0.54  1.00  0.60  0.52  0.75  0.59
DMC                 0.55 -0.45 -0.01 -0.31  0.60  1.00  0.68  0.64  0.92
DC                  0.37 -0.28  0.00 -0.34  0.52  0.68  1.00  0.51  0.76
ISI                 0.58 -0.64  0.05 -0.46  0.75  0.64  0.51  1.00  0.62
BUI                 0.49 -0.38  0.06 -0.32  0.59  0.92  0.76  0.62  1.00
FWI                 0.58 -0.54  0.05 -0.44  0.71  0.75  0.58  0.89  0.75
Classes             0.51 -0.43 -0.02 -0.52  0.77  0.62  0.55  0.75  0.61
Region              0.25 -0.40 -0.15 -0.05  0.18  0.21 -0.03  0.23  0.13

              FWI  Classes  Region
Temperature  0.58     0.51    0.25
RH          -0.54    -0.43   -0.40
Ws           0.05    -0.02   -0.15
Rain        -0.44    -0.52   -0.05
FFMC         0.71     0.77    0.18
DMC          0.75     0.62    0.21
DC           0.58     0.55   -0.03
ISI          0.89     0.75    0.23
BUI          0.75     0.61    0.13
FWI          1.00     0.74    0.19
Classes      0.74     1.00    0.16
Region       0.19     0.16    1.00
```

[46]:
```python
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(data = data, annot = True)
```

[46]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc469c7c0d0>
```




**Observations:**

- BUI and DMC are 92% positively correlated
- FWI and ISI are 89% positively correlated
- No features are more than 95% positively correlated, therefore we cannot drop any feature

# 8 Model Building

*Independent Variable Vs Target Variable distribution.*

Convert `day`, `month`, `year` feature into one `date` feature.

[47]:
```
df['Date'] = pd.to_datetime(df[['day', 'month', 'year']])
```

[48]:
```
df.head()
```

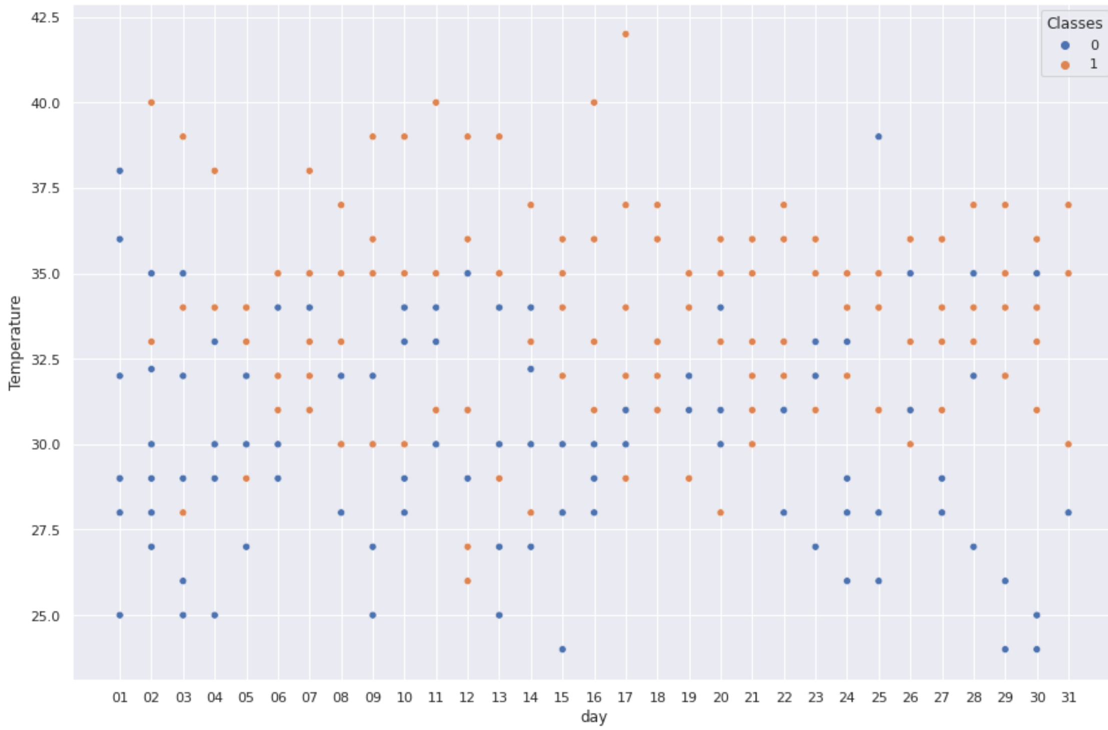
[48]:
```
  day month  year  Temperature  RH    Ws  Rain  FFMC  DMC    DC  ISI  BUI  \
0  01    06  2012         29.0  57  18.0   0.0  65.7  3.4   7.6  1.3  3.4
1  02    06  2012         29.0  61  13.0   0.2  64.4  4.1   7.6  1.0  3.9
2  03    06  2012         26.0  82  15.5   0.2  80.0  2.5   7.1  0.3  2.7
3  04    06  2012         25.0  89  13.0   0.2  80.0  1.3   6.9  0.0  1.7
4  05    06  2012         27.0  77  16.0   0.0  64.8  3.0  14.2  1.2  3.9
```

```
   FWI  Classes  Region        Date
0  0.5        0     0.0 2012-06-01
1  0.4        0     0.0 2012-06-02
2  0.1        0     0.0 2012-06-03
3  0.0        0     0.0 2012-06-04
4  0.5        0     0.0 2012-06-05
```

## 8.1 Scatterplot day Vs temperature

[49]:
```python
sns.scatterplot(data=df, x= 'day', y = 'Temperature', hue = 'Classes')
```

[49]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc469a7b150>



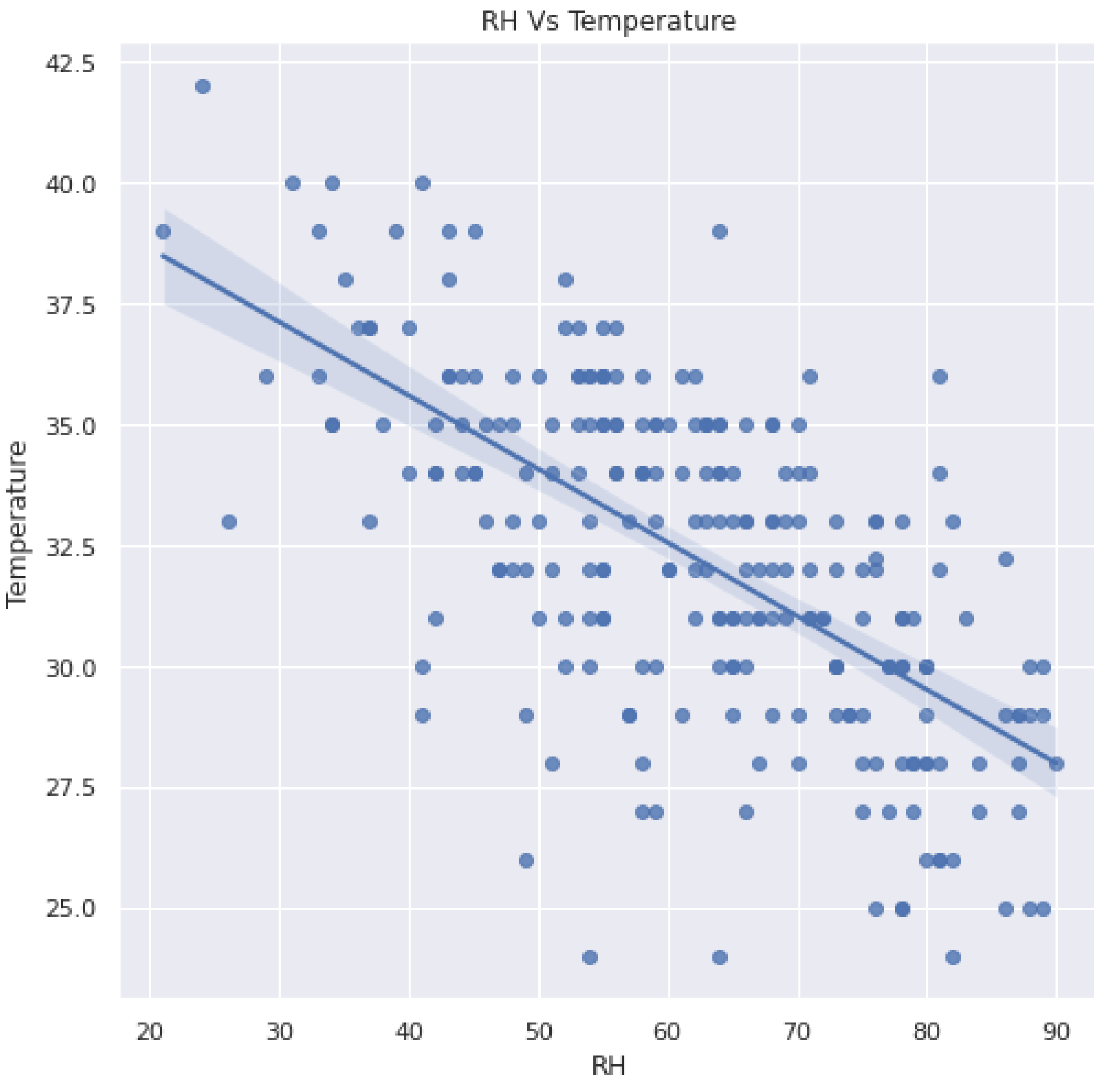

## 8.2 Regression Plot

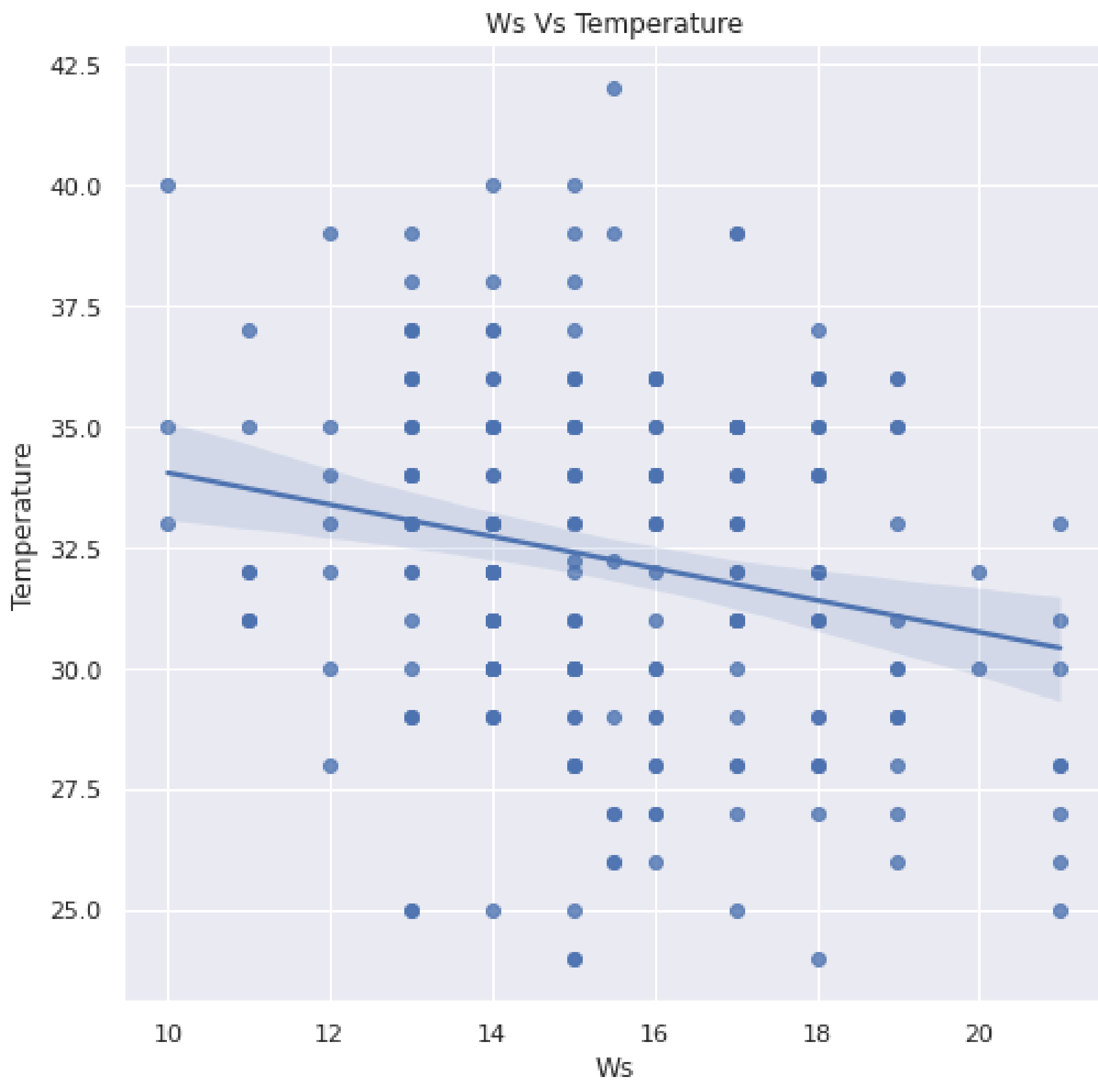
[50]:
```python
consider_feature = [feature for feature in df.columns if feature not in␣
 ↪['Temperature', 'day', 'month', 'year', 'Date', 'Region', 'Classes' ]]
consider_feature
```

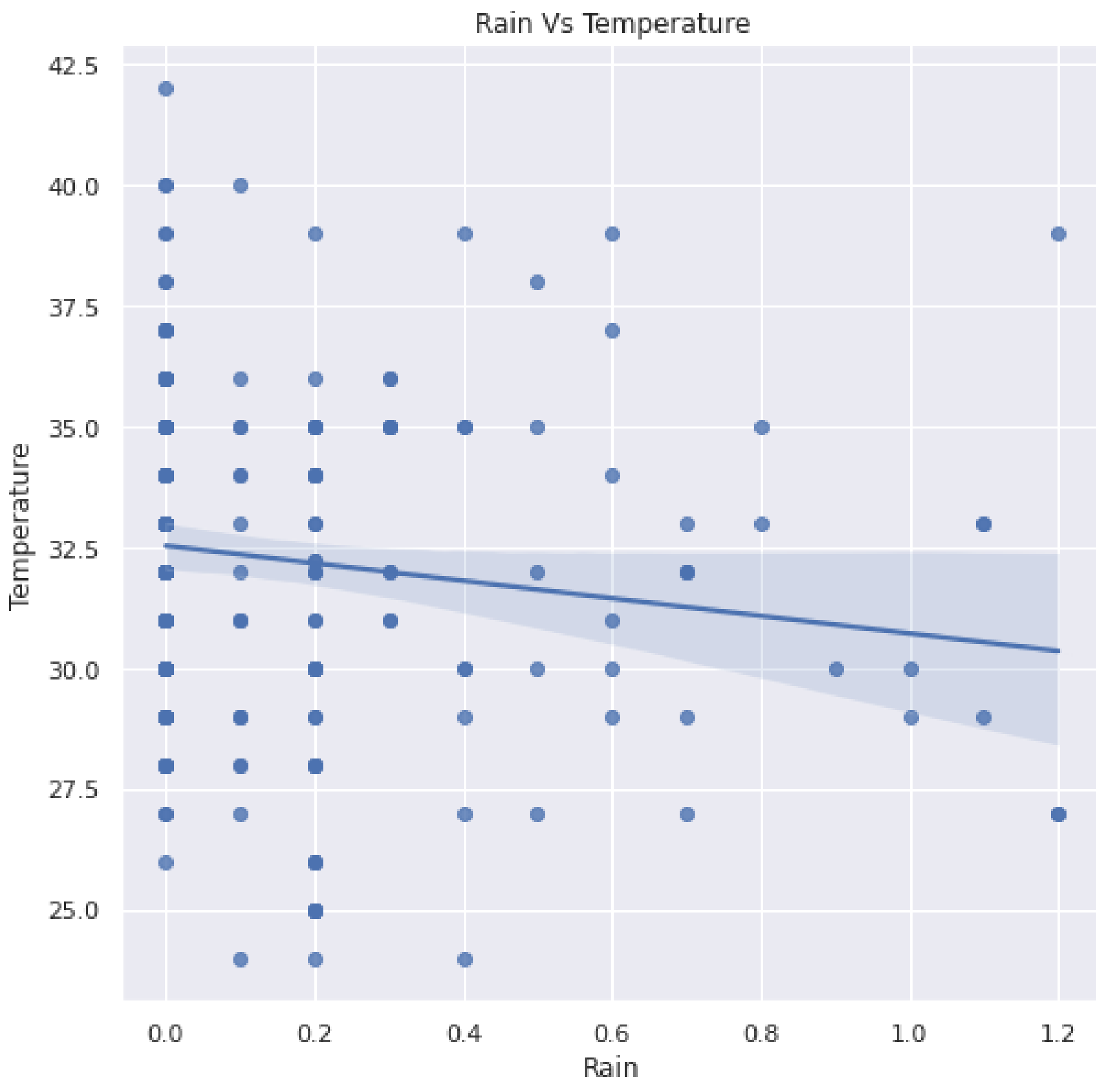
[50]: ['RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI']

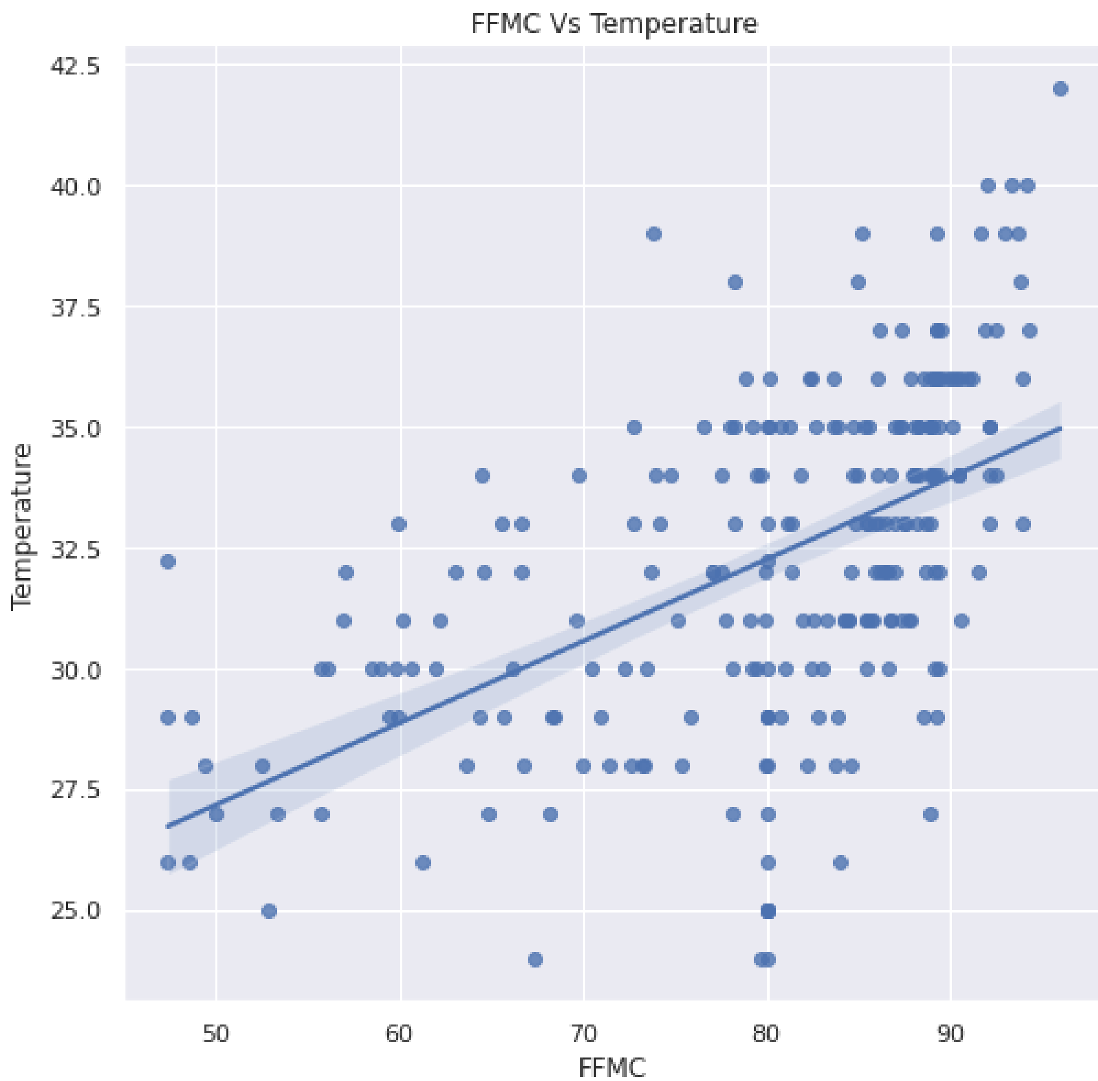
[51]:

```
for feature in consider_feature:
  sns.set(rc={'figure.figsize':(8,8)})
  sns.regplot(x = df[feature], y = df['Temperature'])
  plt.xlabel(feature)
  plt.ylabel('Temperature')
  plt.title("{} Vs Temperature".format(feature))
  plt.show()
```

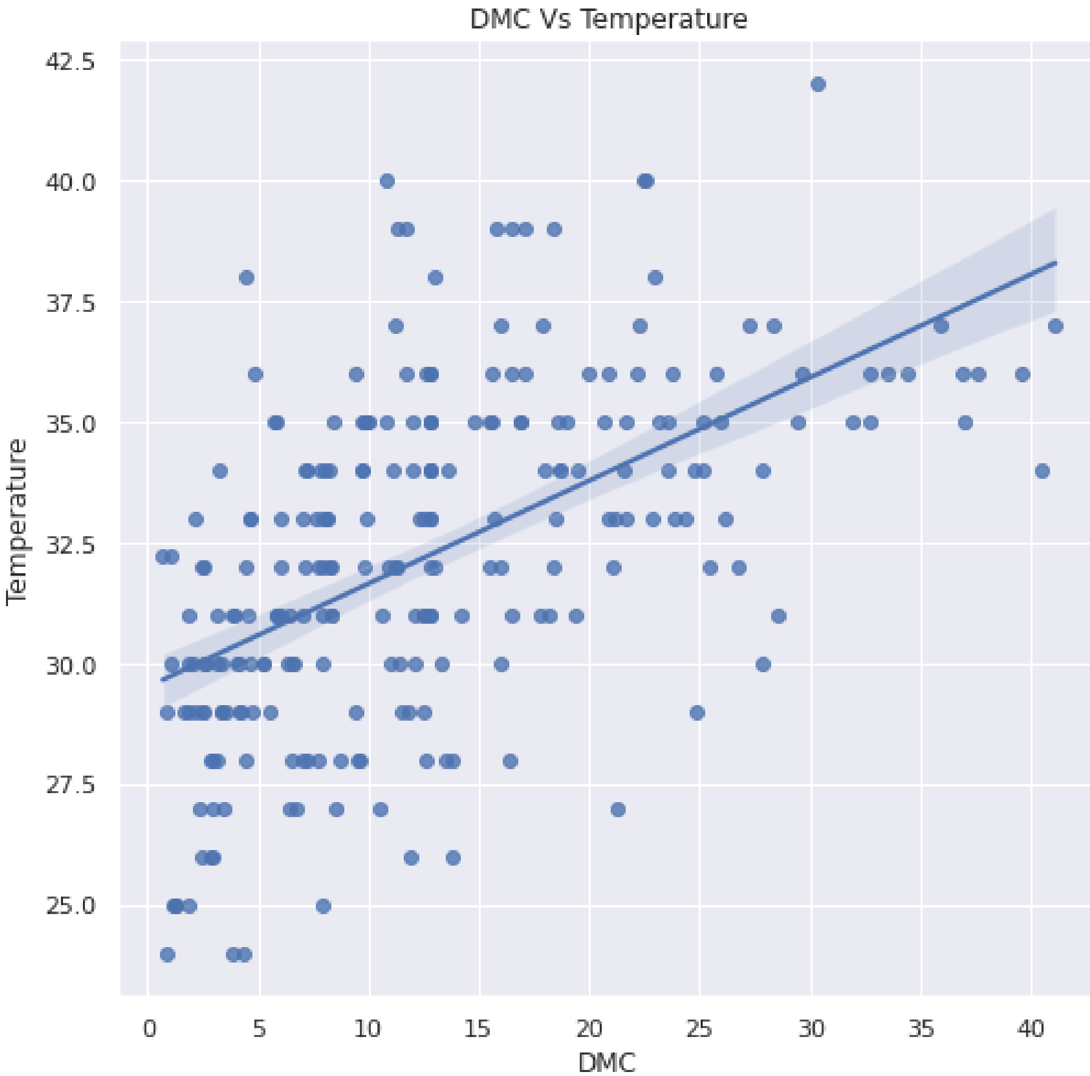



Ws Vs Temperature
Temperature
Ws
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
10
12
14
16
18
20

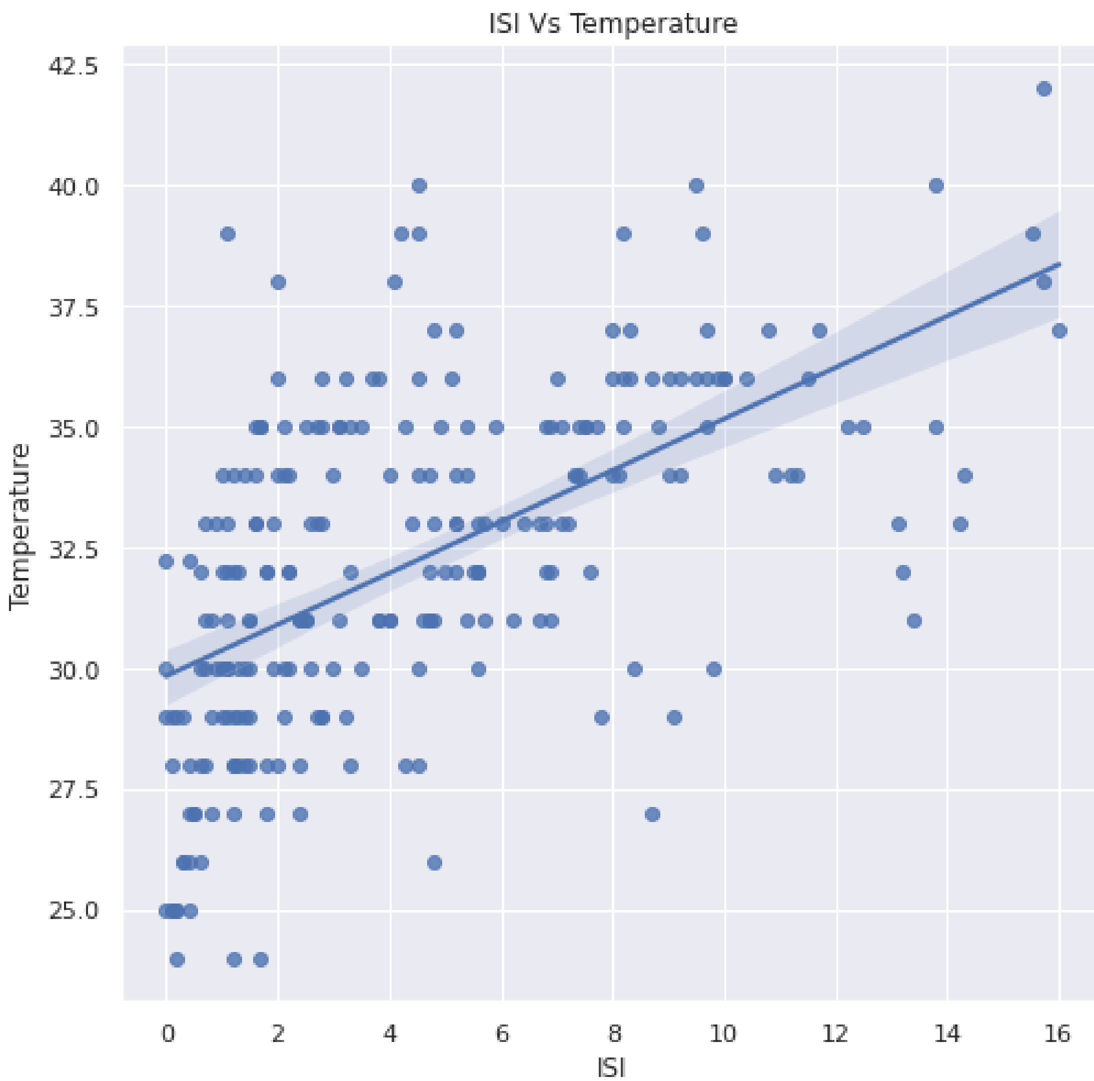

Rain Vs Temperature
Temperature
Rain
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2

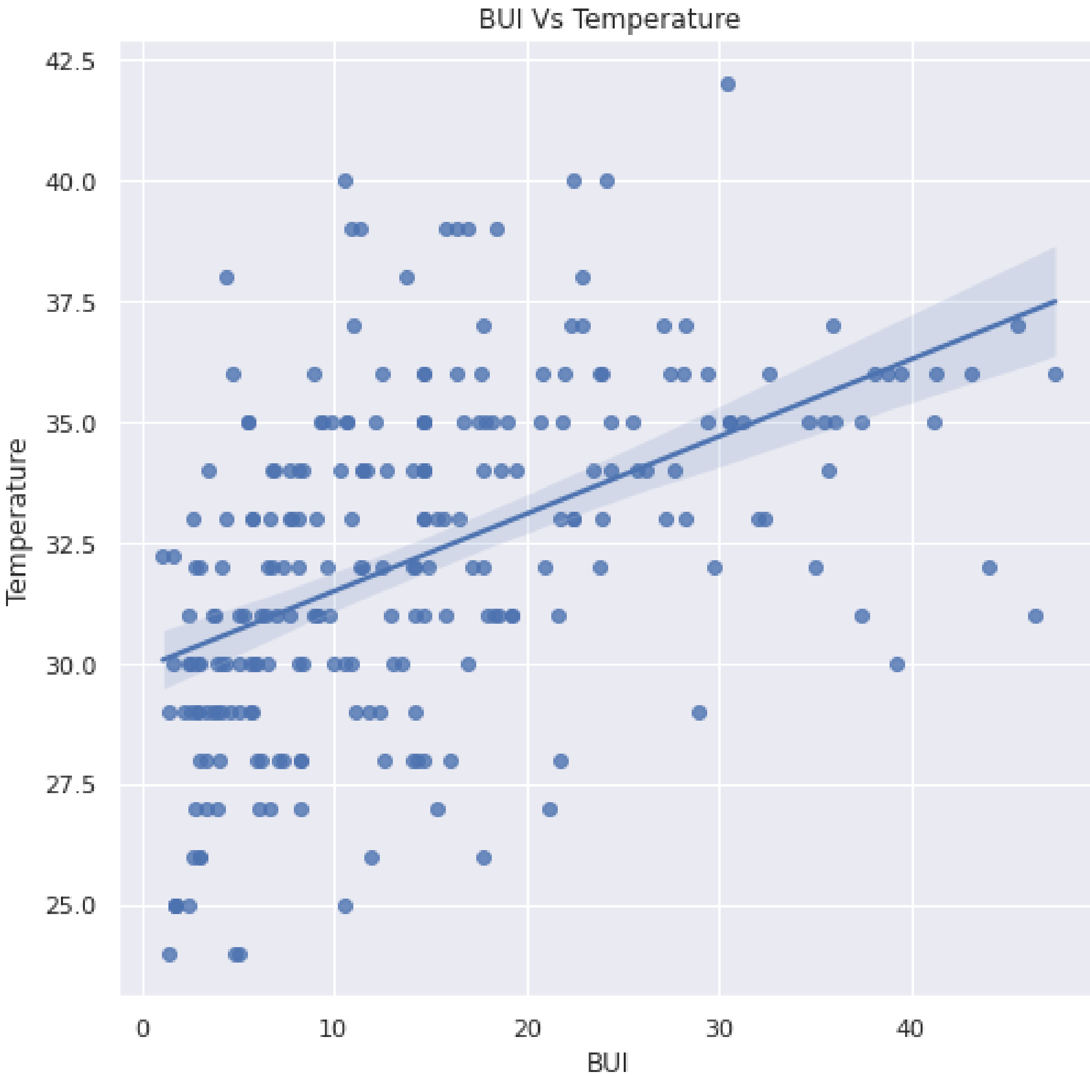


FFMC Vs Temperature
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
Temperature
50
60
70
80
90
FFMC

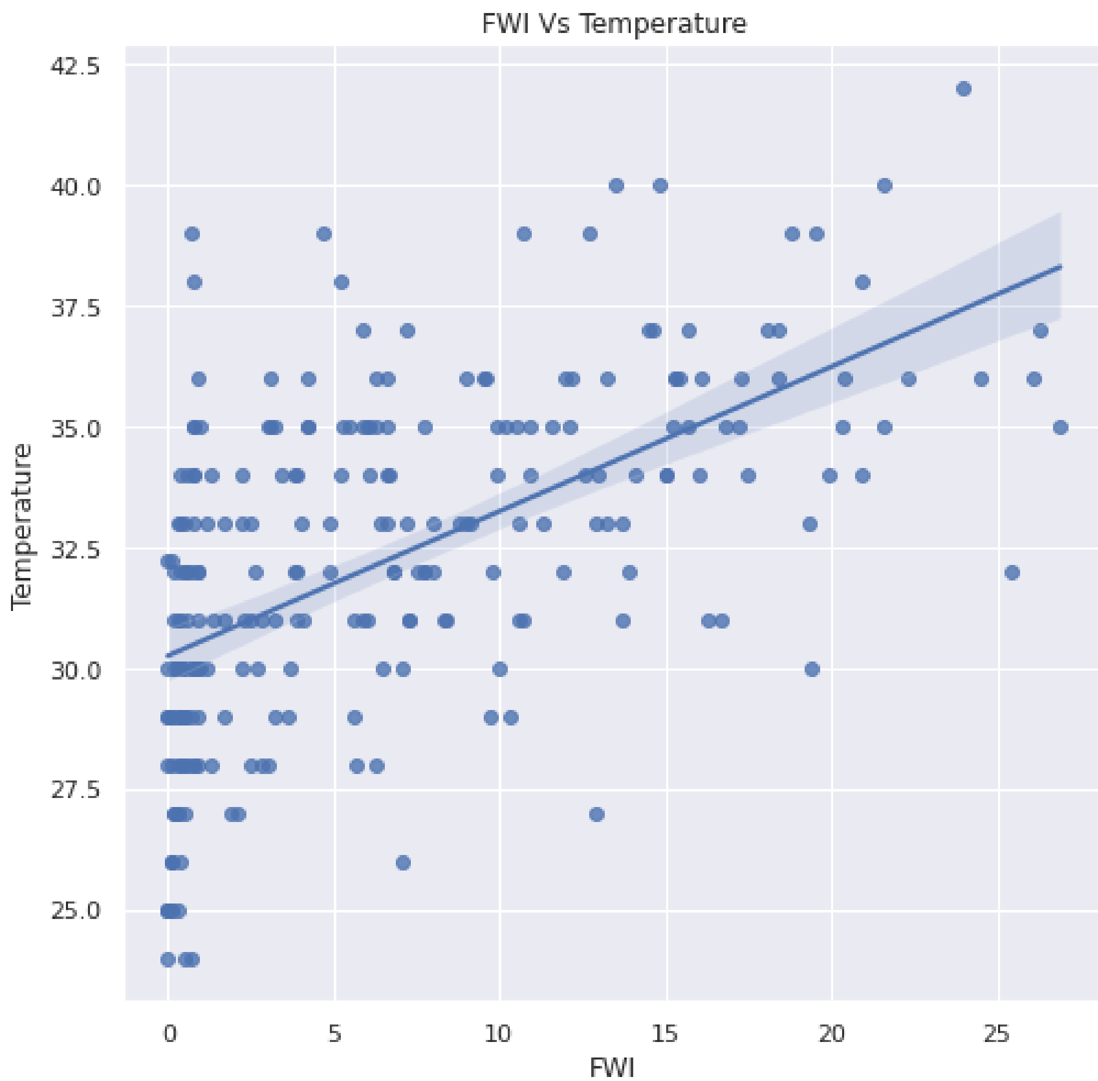

DMC Vs Temperature
Temperature
DMC
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
0
5
10
15
20
25
30
35
40


DC Vs Temperature
Temperature
DC
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
20
40
60
80
100
120


ISI Vs Temperature
Temperature
ISI
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
0
2
4
6
8
10
12
14
16



BUI Vs Temperature
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
Temperature
0
10
20
30
40
BUI




- Shaded region is basically with respect to Ridge and Lasso egression

### 8.3 Segregate Dependent and Independent feature

[52]:
```
# X: independent feature, y: dependent feature
X = df[['RH', 'Ws', 'Rain','FFMC', 'DMC', 'ISI','DC',
       'FWI', 'Classes', 'Region']]
y = df[['Temperature']]
```

[53]:
```
X.head()
```

[53]:

| | RH | Ws | Rain | FFMC | DMC | ISI | DC | FWI | Classes | Region |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 57 | 18.0 | 0.0 | 65.7 | 3.4 | 1.3 | 7.6 | 0.5 | 0 | 0.0 |
| 1 | 61 | 13.0 | 0.2 | 64.4 | 4.1 | 1.0 | 7.6 | 0.4 | 0 | 0.0 |

```
2  82  15.5   0.2  80.0  2.5  0.3   7.1  0.1        0     0.0
3  89  13.0   0.2  80.0  1.3  0.0   6.9  0.0        0     0.0
4  77  16.0   0.0  64.8  3.0  1.2  14.2  0.5        0     0.0
```

[54]:
```
y.head() # dependent feature
```

[54]:
```
   Temperature
0         29.0
1         29.0
2         26.0
3         25.0
4         27.0
```

### 8.4 Split the data into training and testing dataset

[55]:
```
# random state train test split will be same with all using random_state = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33,␣
↪random_state= 42)
```

[56]:
```
# creating a StandardScal;ar object
scaler = StandardScaler()
scaler
```

[56]:
```
StandardScaler()
```

[57]:
```
# using fit_transform to standardise train data
X_train = scaler.fit_transform(X_train)
```

[58]:
```
# here using only transform to avoid data leakage
# (traing mean and training standard deviation will be used for standard␣
↪isolation of test when we use transform on test data)

X_test = scaler.transform(X_test)
```

# 9 Linear Regression Model

[59]:
```
# creating linear regression model
linear_reg = LinearRegression()
linear_reg
```

[59]:
```
LinearRegression()
```

[60]:
```
pd.DataFrame(X_train).isnull().sum()
```

[60]:
```
0    0
1    0
2    0
```

```
3    0
4    0
5    0
6    0
7    0
8    0
9    0
dtype: int64
```

[61]:
```
# passing training data(x and y) to the model:
linear_reg.fit(X_train, y_train)
```

[61]:
```
LinearRegression()
```

Printing co-efficients and intercept of best fit hyperplane

[63]:
```
print(" Co-efficient of Independent features is {}".format(linear_reg.coef_))
print("Intercept of best fit hyper plane is {}".format(linear_reg.intercept_))
```

```
 Co-efficient of Independent features is [[-1.62572989 -0.60047117  0.28921192
-0.05938927  0.76367662  0.00760549
  -0.15916525  0.43191456  0.62332559 -0.26791113]]
Intercept of best fit hyper plane is [32.1617284]
```

Prdiction of Test data

[64]:
```
linear_reg_pred = linear_reg.predict(X_test)
linear_reg_pred[:5]
```

[64]:
```
array([[32.87012119],
       [34.23661089],
       [30.17838715],
       [32.47680473],
       [32.63326791]])
```

[65]:
```
# the difference between y_test and linear_reg_pred

residual_linear_reg = y_test - linear_reg_pred
residual_linear_reg[:5]
```

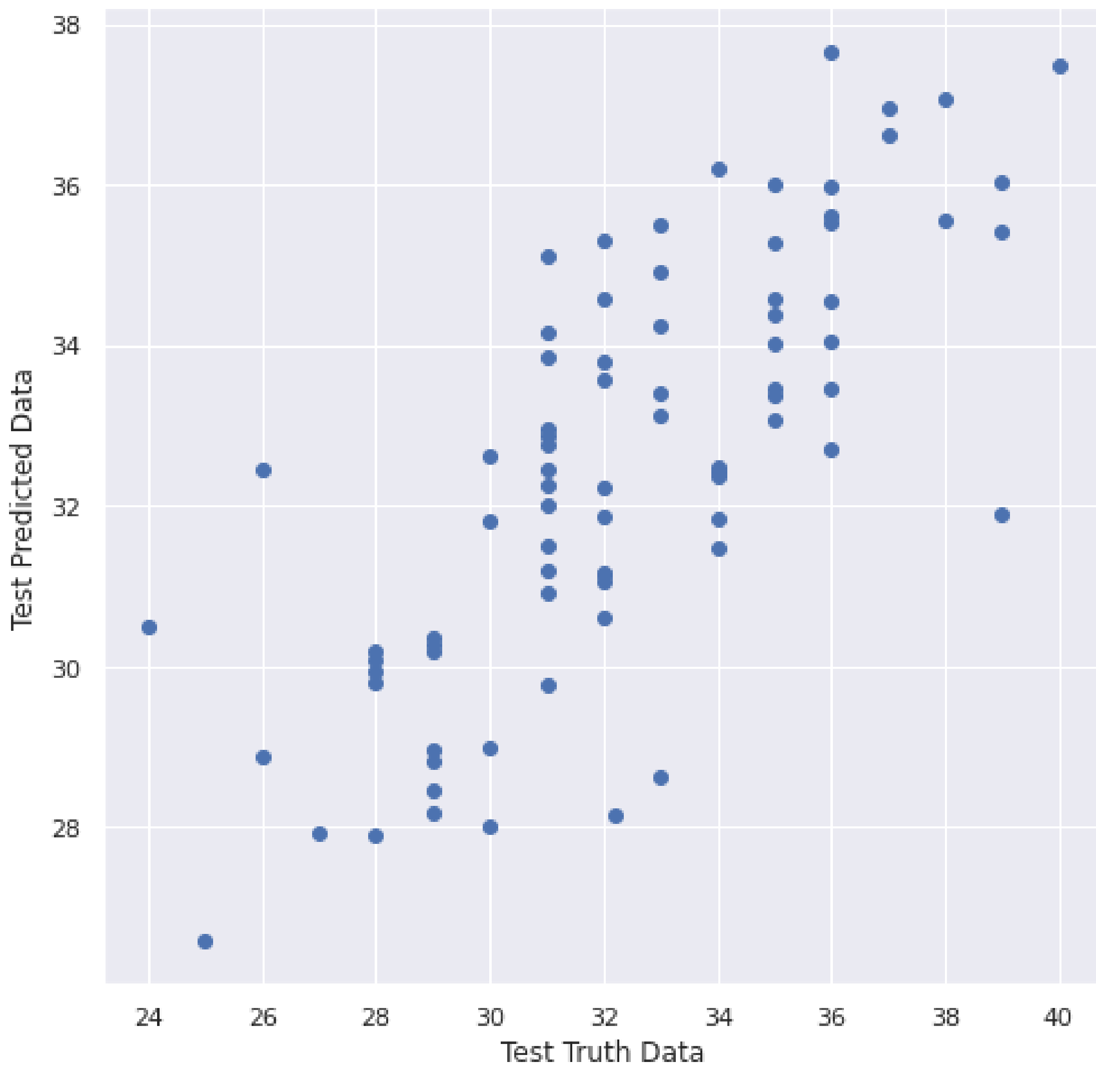
[65]:
```
     Temperature
24     -1.870121
6      -1.236611
152    -2.178387
232     1.523195
238    -2.633268
```

## 9.1 Validation of Linear Regression assumptions

### 9.1.1 Linear Relationship

[67]:
```
plt.scatter(y_test, linear_reg_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

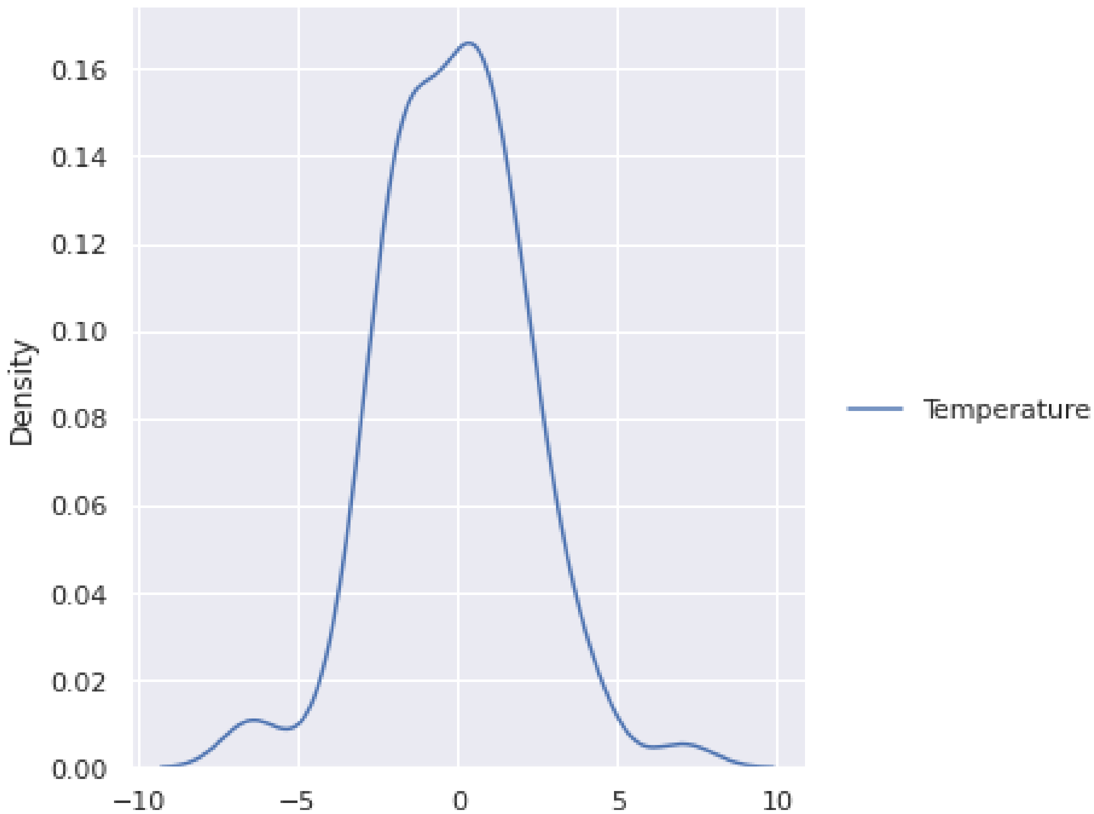
[67]: Text(0, 0.5, 'Test Predicted Data')




### 9.1.2 Residuals should be normally distributed

[68]:
```
sns.displot(data = residual_linear_reg, kind = 'kde')
```

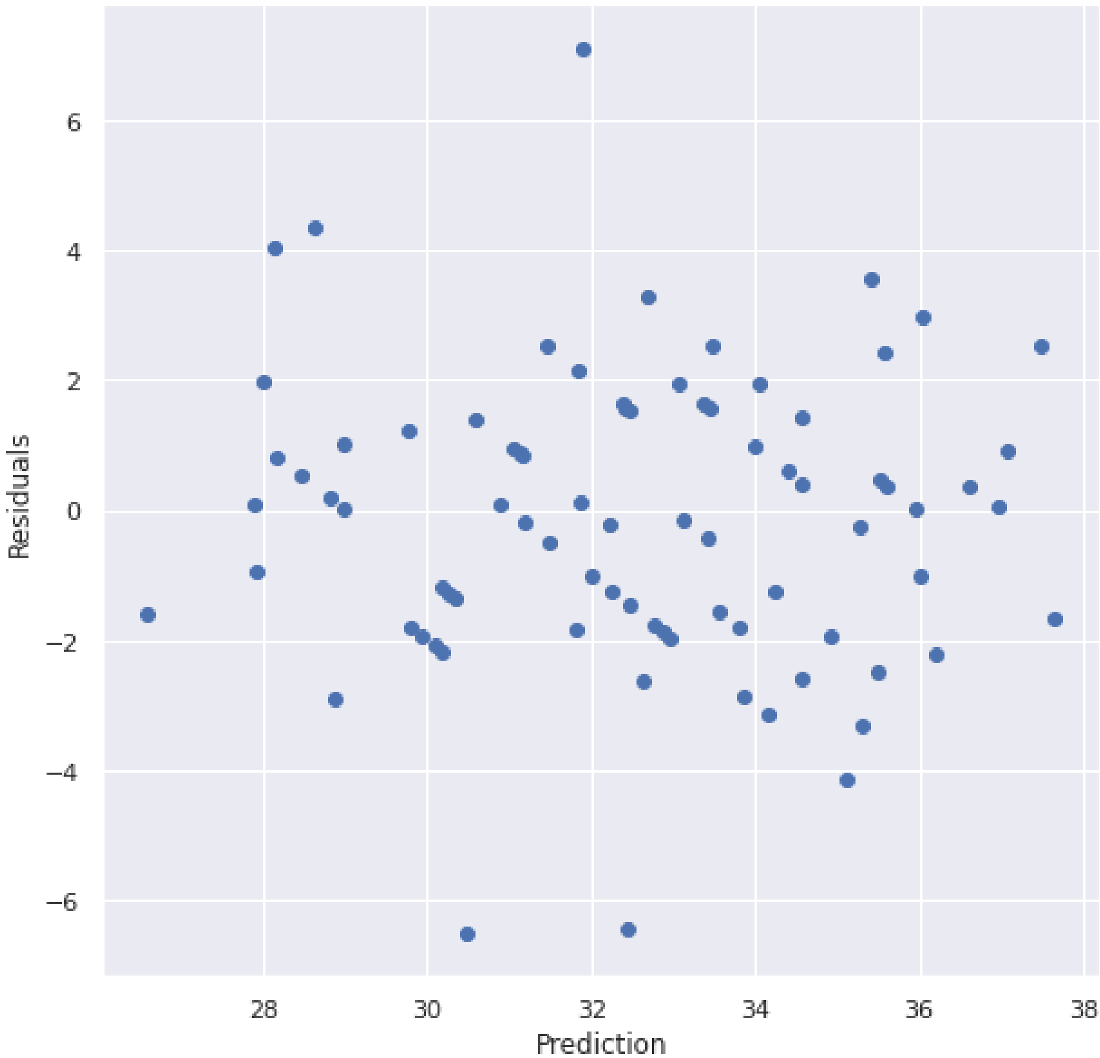
[68]: <seaborn.axisgrid.FacetGrid at 0x7fc46836b690>




### 9.1.3 Residual and Predicted values should follow Uniform Distribution.

[69]:
```
plt.scatter(linear_reg_pred, residual_linear_reg)
plt.xlabel("Prediction")
plt.ylabel("Residuals")
```

[69]:
```
Text(0, 0.5, 'Residuals')
```




## 9.2 Cost Function

```
[71]: print(f"MSE: {round(mean_squared_error(y_test, linear_reg_pred), 2)}")
print(f"MAE: {round(mean_absolute_error(y_test, linear_reg_pred), 2)}")
print(f"RMSE: {round(np.sqrt(mean_squared_error(y_test,linear_reg_pred)),2)}")
```

```
MSE: 5.01
MAE: 1.74
RMSE: 2.24
```

### 9.3 Performance Metrics

[74]:
```
linear_score = r2_score(y_test, linear_reg_pred)
print(f"R-square Accuracy: {round(linear_score*100,2)}%")
print(f"Adjusted R-Square Accuracy : {round((1 -␣
↪(1-linear_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))*100,2)}%")
```

```
R-square Accuracy: 56.52%
Adjusted R-Square Accuracy : 50.31%
```

## 10 Ridge Regression Model

[75]:
```
# creating Ridge Regression Model
ridge_reg = Ridge()
ridge_reg
```

[75]: Ridge()

[76]:
```
# passing training data to the model
ridge_reg.fit(X_train, y_train)
```

[76]: Ridge()

[77]:
```
# printing co-efficients and intercept of best fit hyperplan
print("Co-efficients of Independent features is {}".format(ridge_reg.coef_))
print("Intercept of best fit hyper plane is {}".format(ridge_reg.intercept_))
```

```
Co-efficients of Independent features is [[-1.60178318 -0.59652426  0.29122038
-0.04397446  0.75428229  0.0345659
  -0.15042974  0.41984204  0.606996   -0.25830502]]
Intercept of best fit hyper plane is [32.1617284]
```

#### 10.0.1 Prediction of Test data

[78]:
```
ridge_reg_pred = ridge_reg.predict(X_test)
```

[79]:
```
residual_ridge_reg = y_test - ridge_reg_pred
residual_ridge_reg[:5]
```

[79]:

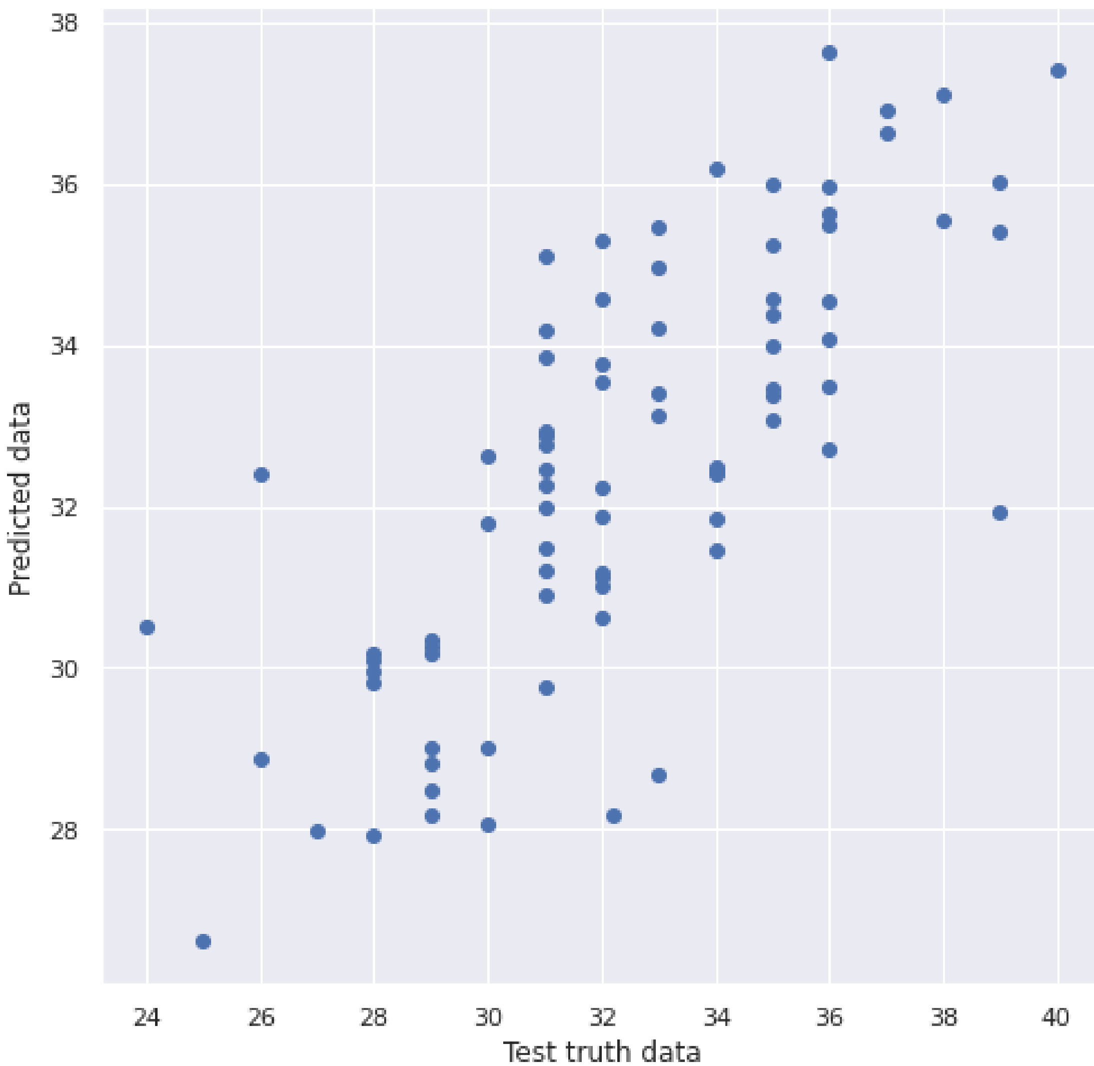
| | Temperature |
|---|---|
| 24 | -1.866884 |
| 6 | -1.217160 |
| 152 | -2.170534 |
| 232 | 1.510022 |
| 238 | -2.634740 |

## 10.1 Validation of Ridge Regression Assumptions

### 10.1.1 Linear Relationship

[80]:
```
plt.scatter(x=y_test, y=ridge_reg_pred)
plt.xlabel("Test truth data")
plt.ylabel("Predicted data")
```

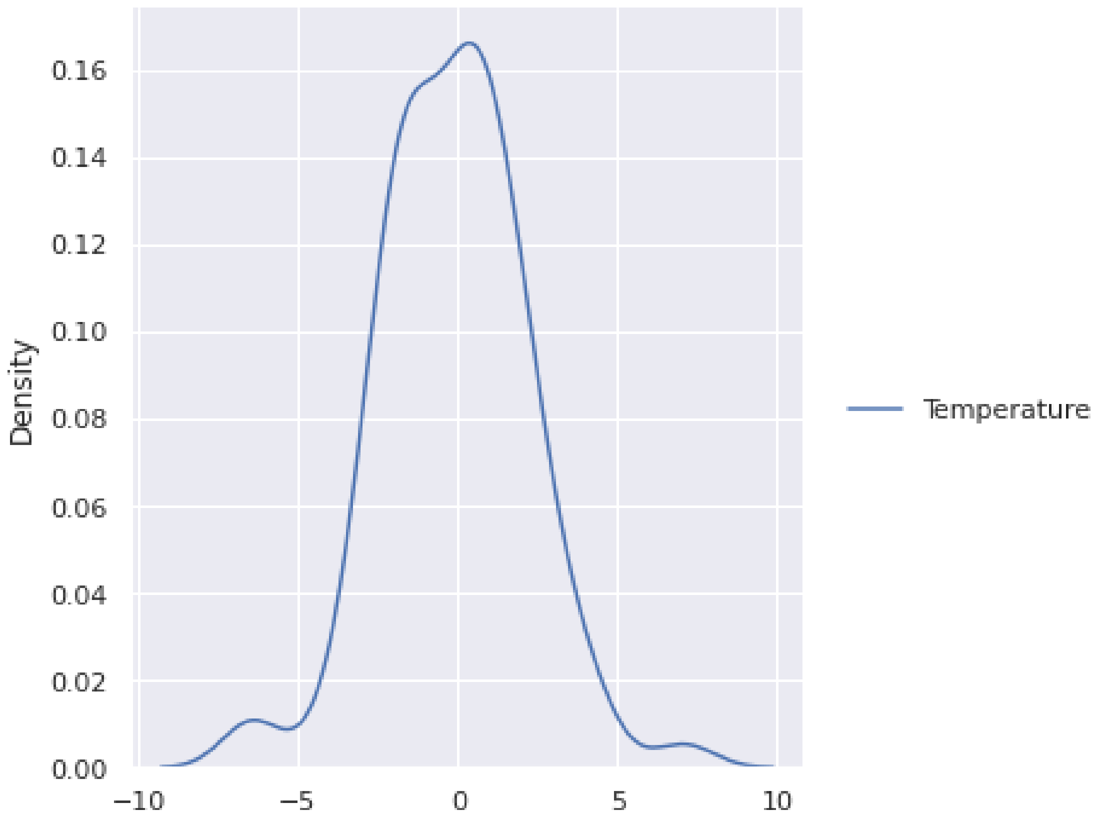
[80]: Text(0, 0.5, 'Predicted data')




### 10.1.2 Residual should be Normally Distributed

[81]:
```
sns.displot(data = residual_ridge_reg, kind='kde')
```

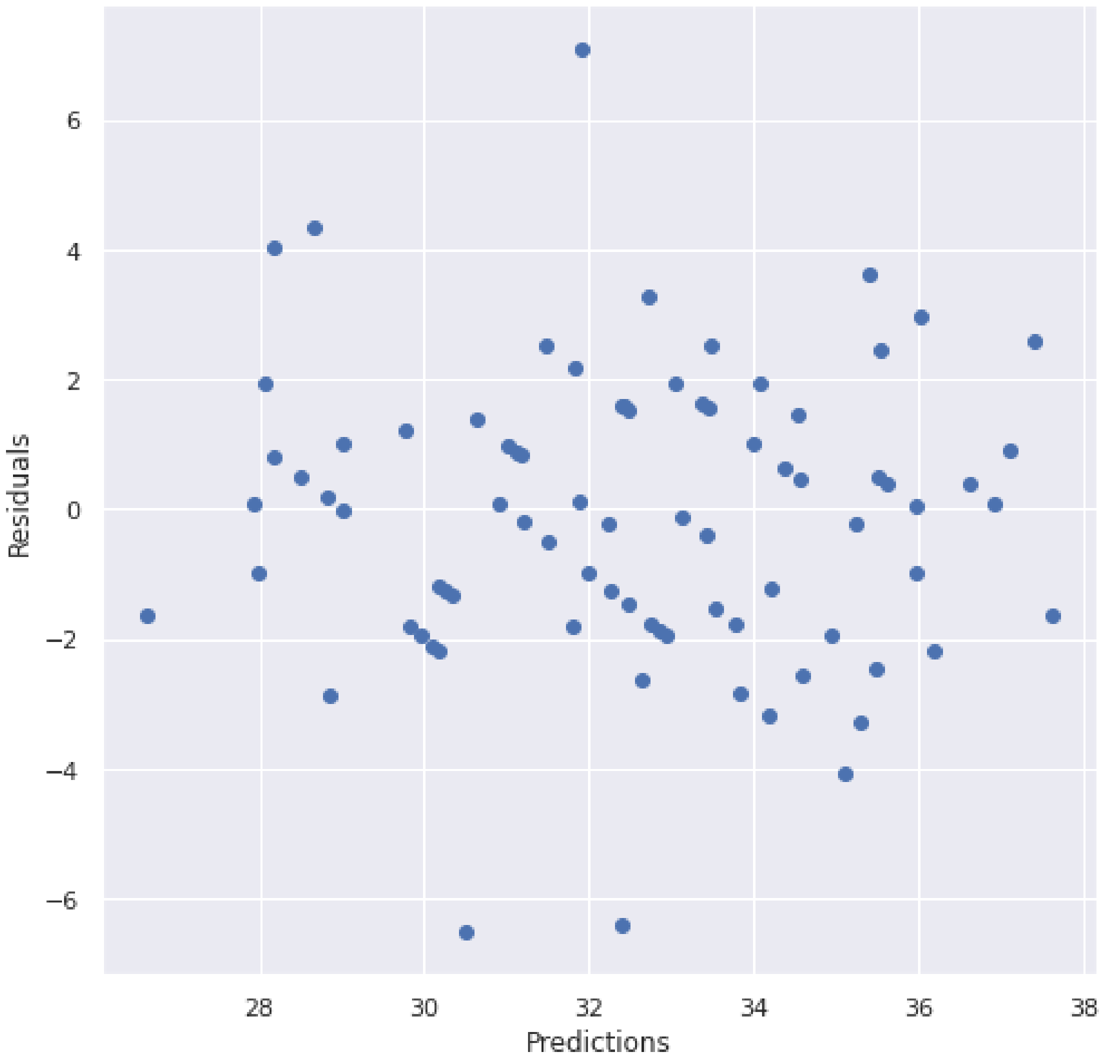
[81]: <seaborn.axisgrid.FacetGrid at 0x7fc469590d10>




#### 10.1.3 Residual and Predicted values should follow Uniform Distribution

[83]:
```
plt.scatter(x=ridge_reg_pred, y=residual_ridge_reg)
plt.xlabel('Predictions')
plt.ylabel("Residuals")
```

[83]: Text(0, 0.5, 'Residuals')




### 10.2 Cost Function Values

[84]:
```
print(f"MSE : {round(mean_squared_error(y_test,ridge_reg_pred),2)}")
print(f"MAE : {round(mean_absolute_error(y_test,ridge_reg_pred),2)}")
print(f"RMSE : {round(np.sqrt(mean_squared_error(y_test,ridge_reg_pred)),2)}")
```

```
MSE : 4.99
MAE : 1.74
RMSE : 2.23
```

### 10.3 Performance Metrics

[85]:
```
Ridge_score = r2_score(y_test,ridge_reg_pred)
print(f"R-Square Accuracy : {round(Ridge_score*100,2)}%")
print(f"Adjusted R-Square Accuracy : {round((1 - (1-Ridge_score)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))*100,2)}%")
```

```
R-Square Accuracy : 56.67%
Adjusted R-Square Accuracy : 50.48%
```

## 11 Lasso Regression Model

[86]:
```
# creating Lasso regression model
lasso_reg = Lasso()
lasso_reg
```

[86]:
```
Lasso()
```

[87]:
```
# Passing training data(X and y) to the model
lasso_reg.fit(X_train, y_train)
```

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

[88]:
```
# Printing co-efficients and intercept of best fit hyperplane
print("Co-efficients of independent features is {}".format(lasso_reg.coef_))
print("Intercept of best fit hyper plane is {}".format(lasso_reg.intercept_))
```

```
Co-efficients of independent features is [-1.08278202 -0.          -0.
0.          0.23127133  0.
  0.          0.2378896   0.          0.          ]
Intercept of best fit hyper plane is [32.1617284]
```

#### 11.0.1 Prediction of Test data

[89]:
```
lasso_reg_pred = lasso_reg.predict(X_test)
lasso_reg_pred[:5]
```

[89]:
```
array([32.16299347, 32.74098733, 32.05836623, 32.55720977, 32.07186032])
```

[90]:
```
y_test = y_test.squeeze()
residual_lasso_reg = y_test - lasso_reg_pred
residual_lasso_reg[:5]
```

[90]:
```
24    -1.162993
6      0.259013
152   -4.058366
232    1.442790
238   -2.071860
```

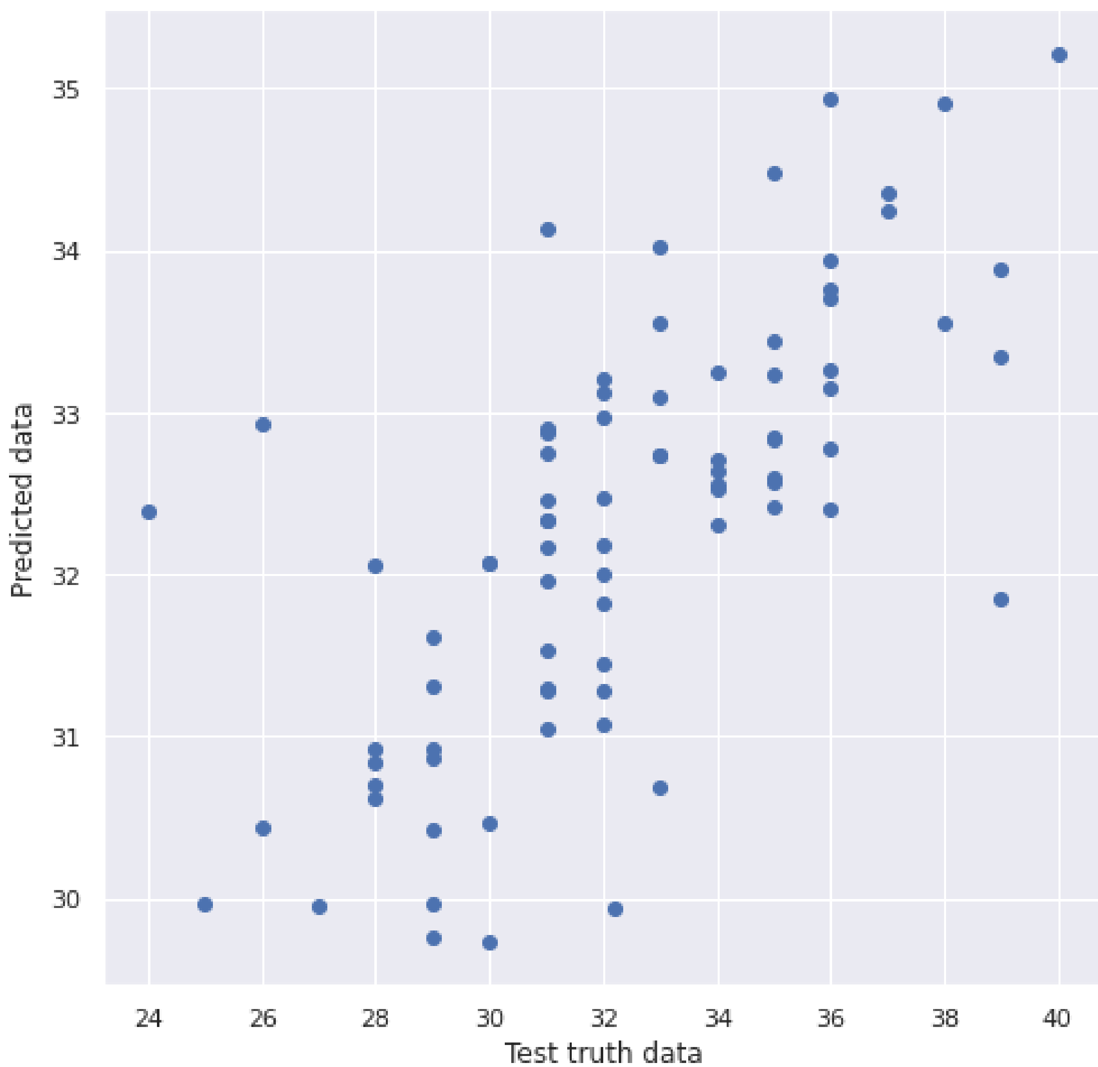
```
Name: Temperature, dtype: float64
```

### 11.1 Validation of Lasso Regression assumptions

## 12 Linear Relationship

[91]:
```
plt.scatter(y_test, lasso_reg_pred)
plt.xlabel("Test truth data")
plt.ylabel("Predicted data")
```

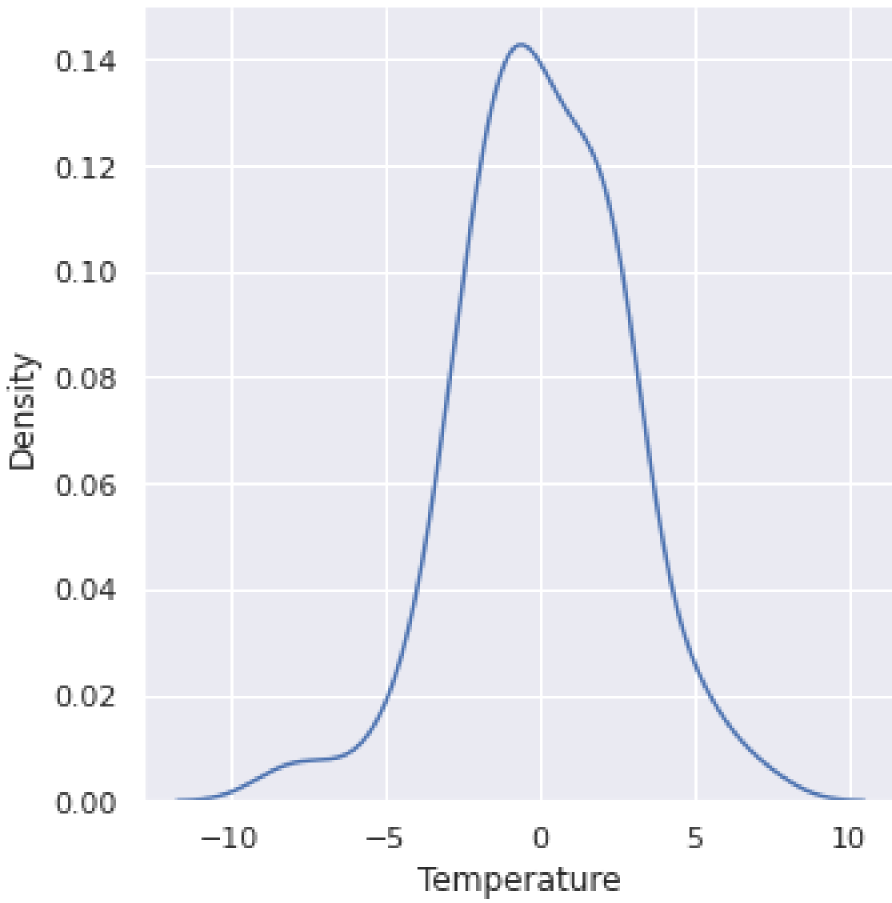
[91]:
```
Text(0, 0.5, 'Predicted data')
```



### 12.0.1 Residual should be Normally Distributed

[92]:
```
sns.displot( residual_lasso_reg, kind='kde')
```

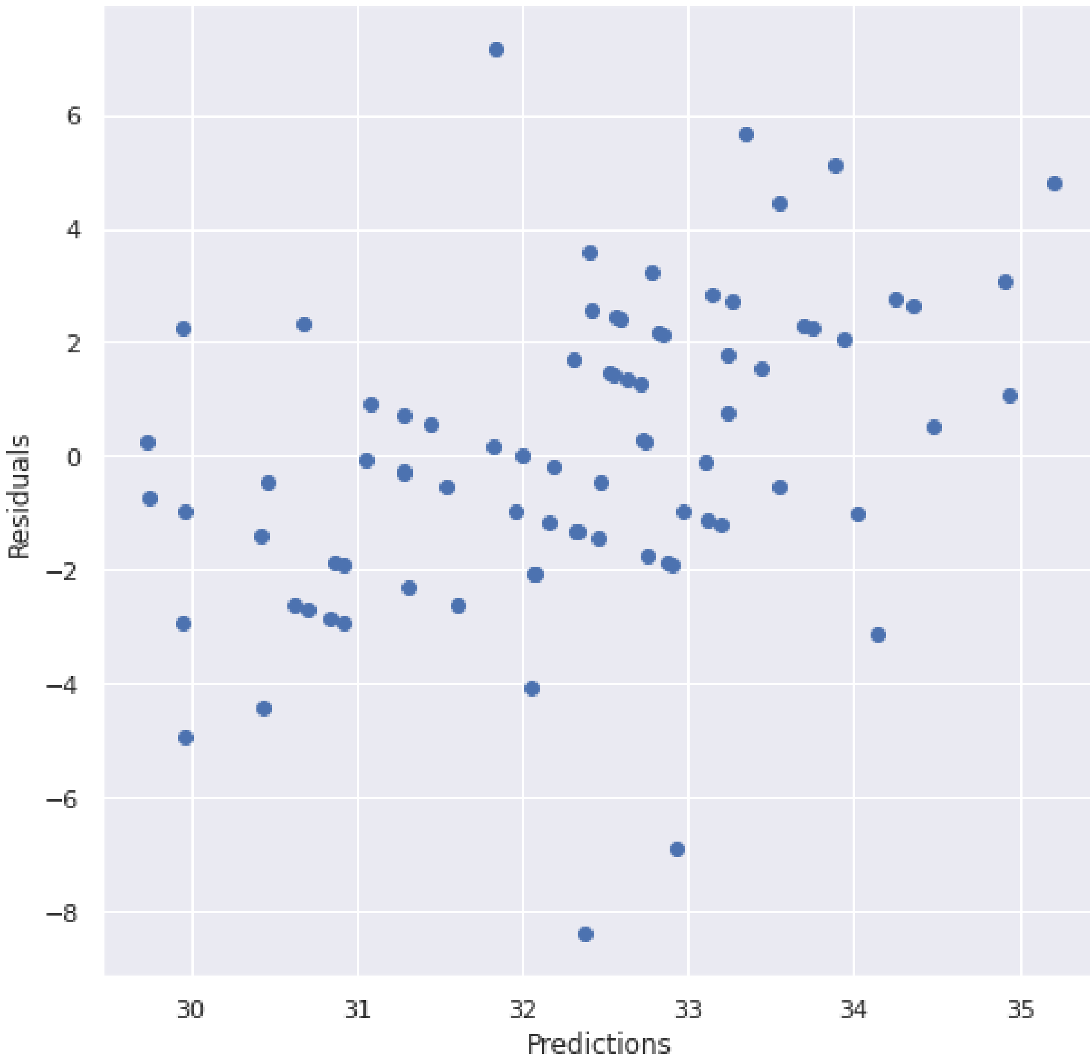
[92]: <seaborn.axisgrid.FacetGrid at 0x7fc4696a2150>




### 12.0.2 Residual and Predicted values should follow Uniform Distribution

[93]:
```
plt.scatter(lasso_reg_pred, residual_lasso_reg)
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

[93]: Text(0, 0.5, 'Residuals')




## 12.1 Cost Function

[96]:
```
print(f"MSE: {round(mean_squared_error(y_test,lasso_reg_pred),2)}")
print(f"MAE: {round(mean_absolute_error(y_test,lasso_reg_pred),2)}")
print(f"RMSE: {round(np.sqrt(mean_squared_error(y_test,lasso_reg_pred)),2)}")
```

```
MSE: 7.06
MAE: 2.07
RMSE: 2.66
```

### 12.2 Performance Metrics

[97]:
```
lasso_score = r2_score(y_test,lasso_reg_pred)
print(f"R-Square Accuracy: {round(lasso_score*100,2)}%")
print(f"Adjusted R-Square Accuracy: {round((1 - (1-lasso_score)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))*100,2)}%")
```

```
R-Square Accuracy: 38.7%
Adjusted R-Square Accuracy: 29.94%
```

## 13 Elastic Net Regression Model

[98]:
```
# creating Elastic-Net regression model
elastic_reg = ElasticNet()
elastic_reg
```

[98]:
```
ElasticNet()
```

[99]:
```
# Passing training data(X and y) to the model
elastic_reg.fit(X_train, y_train)
```

[99]:
```
ElasticNet()
```

[100]:
```
# Printing co-efficients and intercept of best fit hyperplane
print("Co-efficients of independent features is {}".format(elastic_reg.coef_))
print("Intercept of best fit hyper plane is {}".format(elastic_reg.intercept_))
```

```
Co-efficients of independent features is [-0.79936853 -0.05286721 -0.
0.15025684  0.32720261  0.25459529
  0.          0.24934961  0.16518801  0.        ]
Intercept of best fit hyper plane is [32.1617284]
```

#### 13.0.1 Prediction of Test data

[101]:
```
elastic_reg_pred = elastic_reg.predict(X_test)
```

[102]:
```
residual_elastic_reg = y_test - elastic_reg_pred
residual_elastic_reg[:5]
```

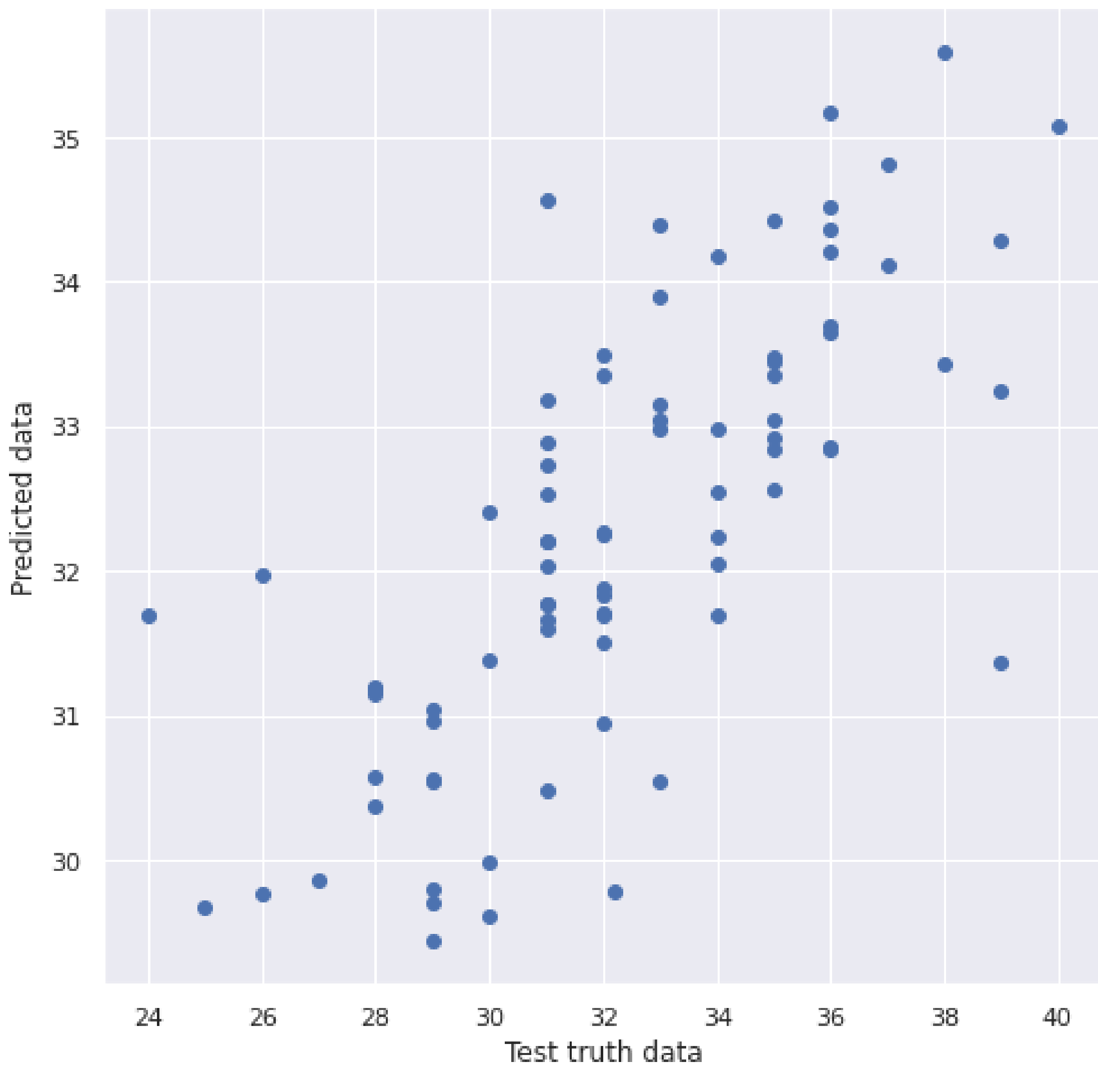
[102]:
```
24    -1.534535
6      0.012176
152   -3.192632
232    1.759639
238   -2.406341
Name: Temperature, dtype: float64
```

### 13.1 Validation of Elastic Regression assumption

#### 13.1.1 Linear Relationship

[103]:
```
plt.scatter(y_test, elastic_reg_pred)
plt.xlabel("Test truth data")
plt.ylabel("Predicted data")
```

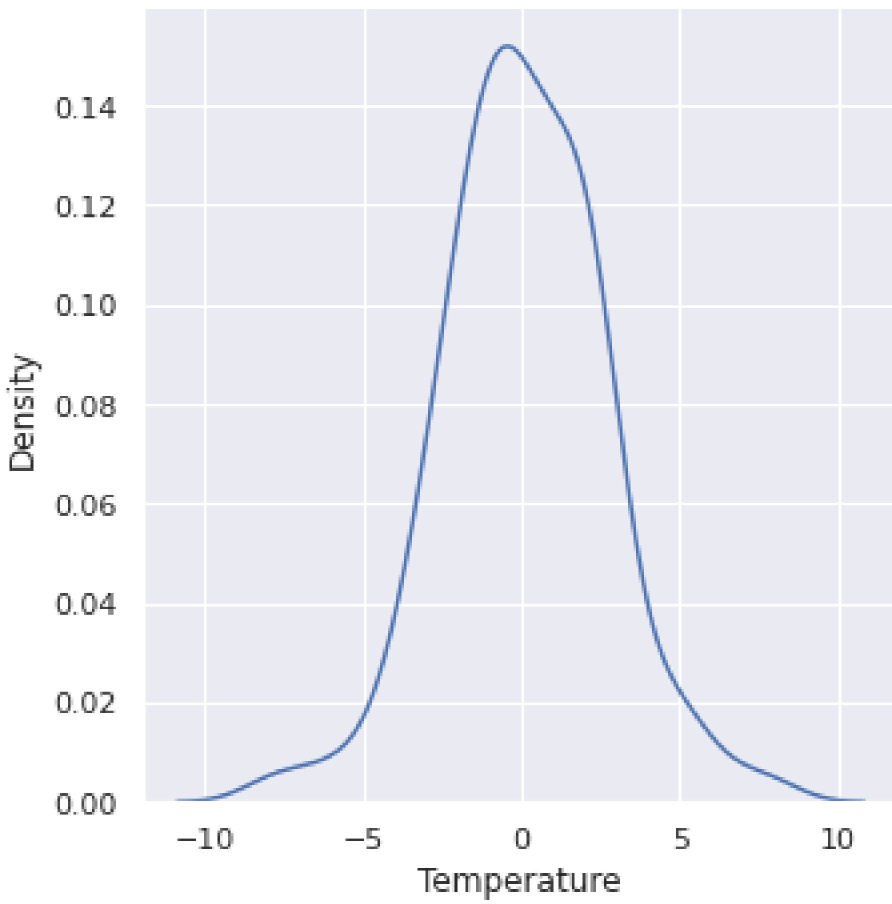
[103]: Text(0, 0.5, 'Predicted data')




#### 13.1.2 Residual should be Normally Distributed

[104]:
```
sns.displot( residual_elastic_reg, kind='kde')
```

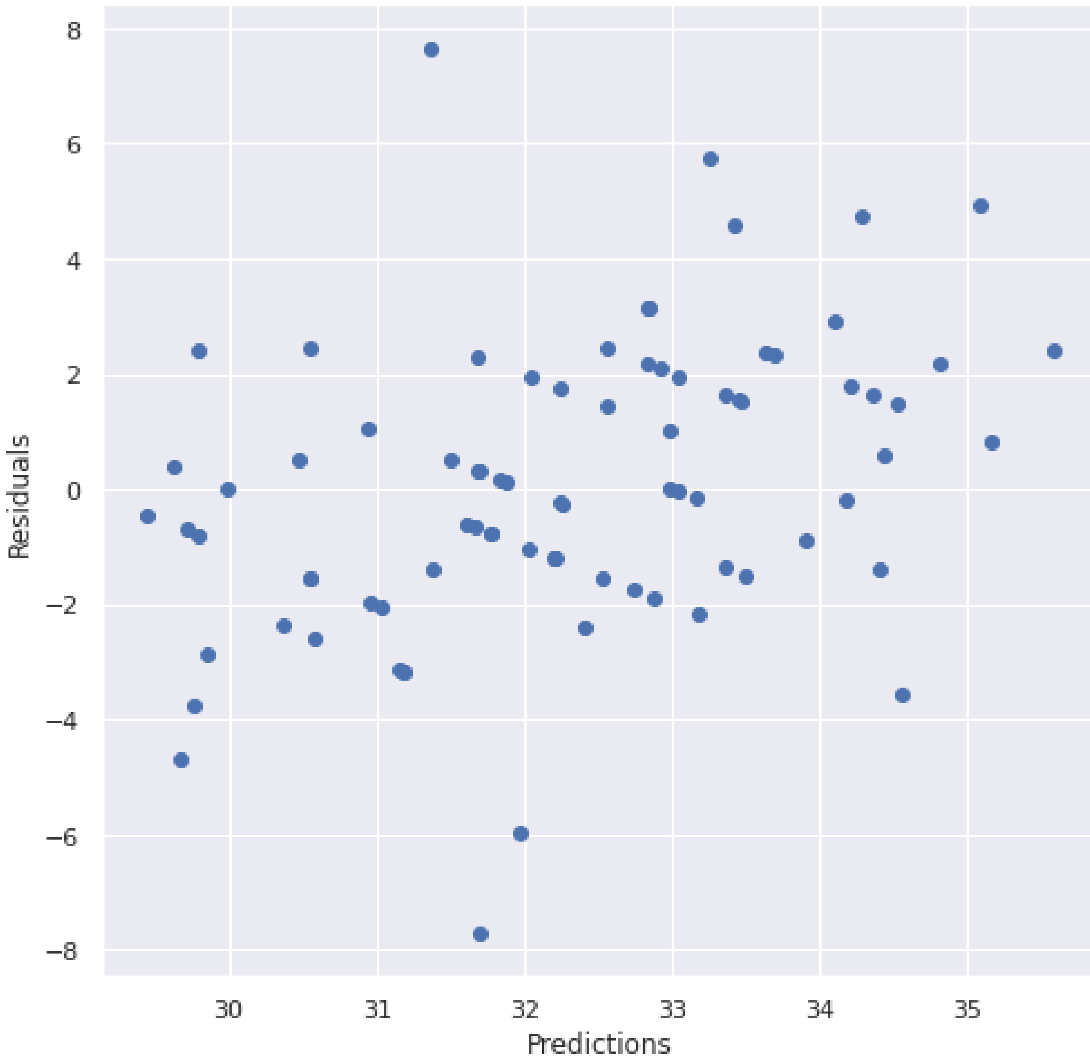
[104]: <seaborn.axisgrid.FacetGrid at 0x7fc46970dcd0>




#### 13.1.3 Residual and Predicted values should follow uniform distribution

[105]:
```
plt.scatter(elastic_reg_pred, residual_elastic_reg)
plt.xlabel('Predictions')
plt.ylabel('Residuals')
```

[105]: Text(0, 0.5, 'Residuals')




### 13.2 Cost Function Values

[107]:
```
print(f"MSE: {round(mean_squared_error(y_test,elastic_reg_pred),2)}")
print(f"MAE: {round(mean_absolute_error(y_test,elastic_reg_pred),2)}")
print(f"RMSE: {round(np.sqrt(mean_squared_error(y_test,elastic_reg_pred)),2)}")
```

```
MSE: 6.37
MAE: 1.95
RMSE: 2.52
```

### 13.3 Performance Metrics

[108]:
```
Elastic_score = r2_score(y_test,elastic_reg_pred)
print(f"R-Square Accuracy : {round(Elastic_score*100,2)}%")
print(f"Adjusted R-Square Accuracy : {round((1 -␣
↪(1-Elastic_score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1))*100,2)}%")
```

```
R-Square Accuracy : 44.71%
Adjusted R-Square Accuracy : 36.81%
```

## 14 Comparision of all Models

Models: 1. Linear Regression 2. Ridge Regression 3. Lasso Regression 4. Elastic Net Regression

### 14.1 Cost Function Values

[109]:
```
print("-------------------------------------------------------")
print(f"MSE:\n1. Linear Regression :␣
↪{round(mean_squared_error(y_test,linear_reg_pred),2)}\n2. Ridge Regression :␣
↪{round(mean_squared_error(y_test,ridge_reg_pred),2)}\n3. Lasso Regression :␣
↪{round(mean_squared_error(y_test,lasso_reg_pred),2)}\n4. ElasticNet Regression␣
↪: {round(mean_squared_error(y_test,elastic_reg_pred),2)}")
print("-------------------------------------------------------")
print(f"MAE:\n1. Linear Regression :␣
↪{round(mean_absolute_error(y_test,linear_reg_pred),2)}\n2. Ridge Regression :␣
↪{round(mean_absolute_error(y_test,ridge_reg_pred),2)}\n3. Lasso Regression :␣
↪{round(mean_absolute_error(y_test,lasso_reg_pred),2)}\n4. ElasticNet␣
↪Regression : {round(mean_absolute_error(y_test,elastic_reg_pred),2)}")
print("-------------------------------------------------------")
print(f"RMSE:\n1. Linear Regression : {round(np.
↪sqrt(mean_squared_error(y_test,linear_reg_pred)),2)}\n2. Ridge Regression :␣
↪{round(np.sqrt(mean_squared_error(y_test,ridge_reg_pred)),2)}\n3. Lasso␣
↪Regression : {round(np.sqrt(mean_squared_error(y_test,lasso_reg_pred)),2)}\n4.␣
↪ElasticNet Regression : {round(np.
↪sqrt(mean_squared_error(y_test,elastic_reg_pred)),2)}")
print("-------------------------------------------------------")
```

```
-------------------------------------------------------
MSE:
1. Linear Regression : 5.01
2. Ridge Regression : 4.99
3. Lasso Regression : 7.06
4. ElasticNet Regression : 6.37
-------------------------------------------------------
MAE:
1. Linear Regression : 1.74
2. Ridge Regression : 1.74
3. Lasso Regression : 2.07
```

```
4. ElasticNet Regression : 1.95
-------------------------------------------------------
RMSE:
1. Linear Regression : 2.24
2. Ridge Regression : 2.23
3. Lasso Regression : 2.66
4. ElasticNet Regression : 2.52
-------------------------------------------------------
```

### 14.2 Performance Metrics

[112]:
```
print("-------------------------------------------------------")
print("R-Square Accuracy:")
print("-------------------------------------------------------")
print(f"1. Linear Regression : {round(linear_score*100,2)}%\n2. Ridge Regression␣
↪: {round(Ridge_score*100,2)}%\n3. Lasso Regression :␣
↪{round(lasso_score*100,2)}%\n4. ElasticNet Regression :␣
↪{round(Elastic_score*100,2)}%")
print("-------------------------------------------------------")
print("Adjusted R-Square Accuracy:")
print("-------------------------------------------------------")
print(f"Linear Regression : {round((1 - (1-linear_score)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))*100,2)}%")
print(f"Ridge Regression : {round((1 - (1-Ridge_score)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))*100,2)}%")
print(f"Lasso Regression : {round((1 - (1-lasso_score)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))*100,2)}%")
print(f"ElasticNet Regression : {round((1 - (1-Elastic_score)*(len(y_test)-1)/
↪(len(y_test)-X_test.shape[1]-1))*100,2)}%")
print("-------------------------------------------------------")
```

```
-------------------------------------------------------
R-Square Accuracy:
-------------------------------------------------------
1. Linear Regression : 56.52%
2. Ridge Regression : 56.67%
3. Lasso Regression : 38.7%
4. ElasticNet Regression : 44.71%
-------------------------------------------------------
Adjusted R-Square Accuracy:
-------------------------------------------------------
Linear Regression : 50.31%
Ridge Regression : 50.48%
Lasso Regression : 29.94%
ElasticNet Regression : 36.81%
-------------------------------------------------------
```

## 15 Conclusion

- If you use the date feature without categorizing then our accuracy will be around 50 % and after the inclusion of categorization it has increased to 66 %, though it is not so good.
- We can remove skewness from the data and also can use some method to handle imbalanced data in Rain feature. This is just a basic model. I will add all the possible techniques to improve accuracy in next session.