<h2 align='center>Algerian Forest Fires Dataset

Life cycle of Machine learning Project

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

1. Problem Statement Need to make a linear regression model where output feature will be Temperature Linear Regression : In Linear Regression our main aim is to findout the best fit line so that our cost function will get reduced. Techniques in Linear Regression

- Ridge Regression (To reduce over-fitting)
- Lasso Regression (To reduce the features)
- Elastic net Regression (Combination of Ridge and Lasso, improves limitation of lasso and perform better than either of the model)

Life cycle of Machine learning Project

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

1. Problem Statement:

- **Create a linear regression model where output feature will be `Temperature`**

Linear Regression :

- In Linear Regression the main aim is to find out the **Best fit line with minimised error**.
- Techniques applied to achieve the objective are:
  - **Ridge Regression** (To reduce over-fitting)
  - **Lasso Regression** (For feature selection)
  - **Elastic net Regression** (It is a combination of both the models, it improves the limitations of Lasso and perform better than both models)

Feature Information about the dataset:

Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012) Weather data observations

Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42

RH : Relative Humidity in %: 21 to 90 (Relative humidity (RH) is a measure of how much moisture is in the air)

Ws : Wind speed in km/h: 6 to 29 (wind speed)

Rain: total day in mm: 0 to 16.8 (Rain in a day in mm)

FWI(Fire Weather Index) Components : 0 to 31.1

Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5 (numerical rating of the moisture content of litter and cured fine fuels)

Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9 (The Duff Moisture Code (DMC) is a numeric rating of the average moisture content of loosely compacted organic layers of moderate depth)

Drought Code (DC) index from the FWI system: 7 to 220.4 (The Drought Code (DC) is a numeric rating of the average moisture content of deep, compact organic layers)

Initial Spread Index (ISI) index from the FWI system: 0 to 18.5 (Initial Spread Index is a relative measure of how quickly a fire can be expected to spread)

Buildup Index (BUI) index from the FWI system: 1.1 to 68 (It is a numeric rating of the total amount of fuel available for combustion)

Fire Weather Index (FWI) Index: 0 to 31.1 (The Fire Weather Index (FWI) is a numeric rating of fire intensity. It is based on the ISI and the BUI, and is used as a general index of fire danger throughout the forested areas of Canada.)

Classes: two classes, namely as fire and as not fire (Result)

Region : There are two regions in the dataset Bejaia Region represented by 1 and Sidi Bel-Abbes Region represented by 1

2. Data Collection:

2.1 Import modules and data and create dataframe

```
# Importing the required libraries

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

sns.set()
%matplotlib inline
warnings.filterwarnings('ignore')

# Creating a dataframe removing the 1st row
```

```
df = pd.read_csv("dataset/Algerian_forest_fires_dataset_UPDATE.csv",
skiprows=1)
```

Show top 5 records

```
df.head()
```

```
  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
0  not fire
1  not fire
2  not fire
3  not fire
4  not fire
```

Shape of the dataset

```
df.shape
```

```
(246, 14)
```

Observations:

- There are 246 rows and 14 columns (features) in the dataset.

```
# Getting basic information about the dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 246 entries, 0 to 245
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   day          246 non-null    object
 1   month        245 non-null    object
 2   year         245 non-null    object
 3   Temperature  245 non-null    object
 4    RH          245 non-null    object
 5    Ws          245 non-null    object
 6   Rain         245 non-null    object
```

```
 7   FFMC         245 non-null    object
 8   DMC          245 non-null    object
 9   DC           245 non-null    object
 10  ISI          245 non-null    object
 11  BUI          245 non-null    object
 12  FWI          245 non-null    object
 13  Classes      244 non-null    object
dtypes: object(14)
memory usage: 27.0+ KB
```

Observations:

- Here we can see all the columns are of object type though they have numeric values.

3. Data Cleaning:

```
# Name of the columns:

df.columns
```

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

Observations:

- There are sapces in some column names.

```
# Trimming the spaces using list comprehension

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

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

Finding the Unique values in the column `'Classes'`

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

```
array(['not fire   ', 'fire   ', 'fire', 'fire ', 'not fire', 'not
fire ',
       nan, 'Classes  ', 'not fire     ', 'not fire    '],
dtype=object)
```

```
# trimming spaces of values in 'Classes' column

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

```
# Let's check it again

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

```
array(['not fire', 'fire', nan, 'Classes'], dtype=object)
```

Removing unnecessary rows

```
# finding index of unnecessary rows

df[df.Classes == 'Classes']
```

```
     day  month  year  Temperature   RH   Ws   Rain  FFMC  DMC  DC
ISI  BUI  \
123  day  month  year  Temperature   RH   Ws  Rain   FFMC  DMC  DC
ISI  BUI

     FWI  Classes
123  FWI  Classes
```

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

```
                               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
```

```
# Removing the rows

df.drop([122, 123], axis=0, inplace=True)
df[120:130]
```

```
    day month  year Temperature  RH  Ws Rain  FFMC  DMC    DC  ISI
BUI  FWI  \
120  29    09  2012          26  80  16  1.8  47.4  2.9   7.7  0.3
3  0.1
121  30    09  2012          25  78  14  1.4    45  1.9   7.5  0.2
2.4  0.1
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
130  07    06  2012          35  44  17  0.2  85.6  9.9  28.9  5.4
10.7    6
131  08    06  2012          28  51  17  1.3  71.4  7.7   7.4  1.5
7.3  0.8

      Classes
120  not fire
121  not fire
124  not fire
125  not fire
126  not fire
127  not fire
128  not fire
129      fire
130      fire
131  not fire
```

Adding a new column named Region

```
# making regions 'Bejaia' as 1 and 'Sidi-Bel Abbes' as 0

df.loc[:122, 'Region'] = 'Bejaia'
df.loc[122:, 'Region'] = 'Sidi-Bel Abbes'
df['Region'] = df['Region'].map({'Bejaia':1, 'Sidi-Bel Abbes':0})
df[120:130]
```

```
    day month  year Temperature  RH  Ws Rain  FFMC  DMC    DC  ISI
BUI  FWI  \
120  29    09  2012          26  80  16  1.8  47.4  2.9   7.7  0.3
3  0.1
121  30    09  2012          25  78  14  1.4    45  1.9   7.5  0.2
2.4  0.1
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
130  07    06  2012          35  44  17  0.2  85.6  9.9  28.9  5.4
10.7    6
131  08    06  2012          28  51  17  1.3  71.4  7.7   7.4  1.5
```

```
7.3  0.8

      Classes  Region
120  not fire       1
121  not fire       1
124  not fire       0
125  not fire       0
126  not fire       0
127  not fire       0
128  not fire       0
129      fire       0
130      fire       0
131  not fire       0
```

Replacing the `'Classes'` column categorical values with numerical values

```
df['Classes'] = df['Classes'].map({'not fire':0, 'fire':1})
df.sample(10)
```

```
    day month  year Temperature  RH  Ws Rain  FFMC   DMC     DC   ISI
BUI  \
127  04    06  2012          30  64  14    0  79.4   5.2   15.4   2.2
5.6
83   23    08  2012          36  53  16    0  89.5  37.6  161.5  10.4
47.5
121  30    09  2012          25  78  14  1.4    45   1.9    7.5   0.2
2.4
132  09    06  2012          27  59  18  0.1  78.1   8.5   14.7   2.4
8.3
75   15    08  2012          36  55  13  0.3  82.4  15.6   92.5   3.7
22
156  03    07  2012          34  56  17  0.1  84.7   9.7   27.3   4.7
10.3
104  13    09  2012          25  86  21  4.6  40.9   1.3    7.5   0.1
1.8
67   07    08  2012          32  69  16    0  86.5  15.5   48.6   5.5
17.2
124  01    06  2012          32  71  12  0.7  57.1   2.5    8.2   0.6
2.8
39   10    07  2012          33  69  13  0.7  66.6     6    9.3   1.1
5.8

      FWI  Classes  Region
127     1      0.0       0
83   22.3      1.0       1
121   0.1      0.0       1
132   1.9      0.0       0
75    6.3      1.0       1
156   5.2      1.0       0
104     0      0.0       1
67      8      1.0       1
```

```
124   0.2      0.0       0
39    0.5      0.0       1
```

Checking all the unique values in each columns

```python
for column in df.columns:
    print(f"The unique values in column {column}:")
    print(df[column].unique())
    print("-----------------------------------\n")
```

```
The unique values in column day:
['01' '02' '03' '04' '05' '06' '07' '08' '09' '10' '11' '12' '13' '14'
 '15' '16' '17' '18' '19' '20' '21' '22' '23' '24' '25' '26' '27' '28'
 '29' '30' '31']
----------------------------------

The unique values in column month:
['06' '07' '08' '09']
----------------------------------

The unique values in column year:
['2012']
----------------------------------

The unique values in column Temperature:
['29' '26' '25' '27' '31' '33' '30' '28' '32' '34' '35' '36' '37' '22'
 '24' '38' '39' '40' '42']
----------------------------------

The unique values in column RH:
['57' '61' '82' '89' '77' '67' '54' '73' '88' '79' '65' '81' '84' '78'
 '80' '55' '62' '66' '64' '53' '47' '50' '68' '75' '76' '63' '69' '70'
 '59' '48' '45' '60' '51' '52' '58' '86' '74' '71' '49' '44' '41' '42'
 '90' '87' '72' '46' '37' '36' '56' '43' '83' '29' '34' '33' '35' '39'
 '31' '21' '40' '24' '38' '26']
----------------------------------

The unique values in column Ws:
['18' '13' '22' '16' '14' '15' '12' '19' '21' '20' '17' '26' '11' '10'
'9'
 '8' '6' '29']
----------------------------------

The unique values in column Rain:
['0' '1.3' '13.1' '2.5' '0.2' '1.2' '0.5' '3.1' '0.7' '0.6' '0.3'
'0.1'
 '0.4' '1' '1.4' '0.8' '16.8' '7.2' '10.1' '3.8' '0.9' '1.8' '4.6'
'8.3'
 '5.8' '4' '2' '4.7' '8.7' '4.5' '1.1' '1.7' '2.2' '6' '1.9' '2.9'
'4.1'
 '6.5' '4.4']
```

```
----------------------------------

The unique values in column FFMC:
['65.7' '64.4' '47.1' '28.6' '64.8' '82.6' '88.2' '86.6' '52.9' '73.2'
 '84.5' '84' '50' '59' '49.4' '36.1' '37.3' '56.9' '79.9' '59.8' '81'
 '79.1' '81.4' '85.9' '86.7' '86.8' '89' '89.1' '88.7' '59.9' '55.7'
 '63.1' '80.1' '87' '80' '85.6' '66.6' '81.1' '75.1' '81.8' '73.9'
'60.7'
 '72.6' '82.8' '85.4' '88.1' '73.4' '68.2' '70' '84.3' '89.2' '90.3'
 '86.5' '87.2' '78.8' '78' '76.6' '85' '86.4' '77.1' '87.4' '88.9'
'81.3'
 '82.4' '80.2' '89.3' '89.4' '88.3' '88.6' '89.5' '85.8' '84.9' '90.1'
 '72.7' '52.5' '46' '30.5' '42.6' '68.4' '80.8' '75.8' '69.6' '62'
'56.1'
 '58.5' '71' '40.9' '47.4' '44.9' '78.1' '87.7' '83.8' '87.8' '77.8'
 '73.7' '68.3' '48.6' '82' '85.7' '77.5' '45' '57.1' '48.7' '79.4'
'83.7'
 '71.4' '90.6' '72.3' '53.4' '66.8' '62.2' '65.5' '64.6' '60.2' '86.2'
 '78.3' '74.2' '85.3' '86' '92.5' '79.7' '63.7' '87.6' '84.7' '88'
'90.5'
 '82.3' '74.8' '85.2' '84.6' '86.1' '89.9' '93.9' '91.5' '87.3' '72.8'
 '73.8' '87.5' '93.3' '93.7' '93.8' '70.5' '69.7' '91.7' '94.2' '93'
 '91.9' '83.9' '92' '96' '94.3' '82.7' '91.2' '92.1' '92.2' '91'
'79.2'
 '37.9' '75.4' '82.2' '73.5' '66.1' '64.5' '83.3' '82.5' '83.1' '59.5'
 '84.2' '79.5' '61.3' '41.1' '45.9' '67.3']
----------------------------------

The unique values in column DMC:
['3.4' '4.1' '2.5' '1.3' '3' '5.8' '9.9' '12.1' '7.9' '9.5' '12.5'
'13.8'
 '6.7' '4.6' '1.7' '1.1' '1.9' '4.5' '6.3' '7' '8.2' '11.2' '14.2'
'17.8'
 '21.6' '25.5' '18.4' '22.9' '2.4' '2.6' '7.6' '10.9' '9.7' '7.7' '6'
 '8.1' '7.8' '5.2' '9.4' '12' '12.3' '18.5' '16.4' '10.5' '9.6' '17.1'
 '22.2' '24.4' '26.7' '28.5' '31.9' '4.8' '5.7' '11.1' '13' '15.5'
'11.3'
 '14.8' '18.6' '21.7' '15.6' '19' '11.7' '16' '20' '23.2' '25.9'
'29.6'
 '33.5' '37.6' '40.5' '43.9' '45.6' '47' '50.2' '54.2' '25.2' '8.7'
'0.7'
 '1.2' '3.6' '3.2' '2.1' '2.2' '0.9' '6.4' '9.8' '13.5' '16.5' '10.6'
 '5.5' '8.3' '7.1' '2.9' '2.7' '8.4' '8.5' '13.3' '18.2' '21.3' '11.4'
 '7.2' '4.2' '3.9' '4.4' '3.8' '10' '12.8' '20.9' '27.2' '17.9' '13.6'
 '18.7' '8' '12.6' '12.9' '18' '19.4' '21.1' '23.9' '27.8' '32.7'
'39.6'
 '44.2' '46.6' '10.8' '11.8' '15.7' '19.5' '23.8' '28.3' '23' '23.6'
'11'
 '15.8' '22.5' '16.9' '22.3' '22.6' '30.3' '35.9' '34.4' '36.9' '41.1'
 '46.1' '51.3' '56.3' '61.3' '65.9' '37' '20.7' '24.8' '4' '3.3' '6.6'
```

```
 '4.7' '6.5' '11.5' '21.2' '25.8' '24.9' '26.1' '29.4' '11.9' '3.5'
'4.3']
----------------------------------

The unique values in column DC:
['7.6' '7.1' '6.9' '14.2' '22.2' '30.5' '38.3' '38.8' '46.3' '54.3'
'61.4'
 '17' '7.8' '7.4' '8' '16' '27.1' '31.6' '39.5' '47.7' '55.8' '63.8'
 '71.8' '80.3' '88.5' '84.4' '92.8' '8.6' '8.3' '9.2' '18.5' '27.9'
'37'
 '40.4' '49.8' '9.3' '18.7' '27.7' '37.2' '22.9' '25.5' '34.1' '43.1'
 '52.8' '62.1' '71.5' '79.9' '71.3' '79.7' '88.7' '98.6' '108.5'
'117.8'
 '127' '136' '145.7' '10.2' '10' '19.8' '29.7' '39.1' '48.6' '47' '57'
 '67' '77' '75.1' '85.1' '94.7' '92.5' '90.4' '100.7' '110.9' '120.9'
 '130.6' '141.1' '151.3' '161.5' '171.3' '181.3' '190.6' '200.2'
'210.4'
 '220.4' '180.4' '8.7' '7.5' '7' '15.7' '24' '32.2' '30.1' '8.4' '8.9'
 '16.6' '7.3' '24.3' '33.1' '41.3' '49.3' '57.9' '41.4' '30.4' '15.2'
 '7.7' '16.3' '24.9' '8.8' '8.2' '15.4' '17.6' '26.3' '28.9' '14.7'
'22.5'
 '37.8' '18.4' '25.6' '34.5' '43.3' '52.4' '36.7' '8.5' '17.8' '27.3'
 '36.8' '46.4' '45.1' '35.4' '9.7' '9.9' '9.5' '19.4' '10.4' '14.6 9'
 '24.1' '42.3' '51.6' '61.1' '71' '80.6' '90.1' '99' '56.6' '15.9'
'19.7'
 '28.3' '37.6' '47.2' '57.1' '67.2' '10.5' '21.4' '32.1' '42.7' '52.5'
 '9.1' '9.8' '20.2' '30.9' '41.5' '55.5' '54.2' '65.1' '76.4' '86.8'
 '96.8' '107' '117.1' '127.5' '137.7' '147.7' '157.5' '167.2' '177.3'
 '166' '149.2' '159.1' '168.2' '26.6' '17.7' '26.1' '25.2' '33.4'
'50.2'
 '59.2' '63.3' '77.8' '86' '88' '97.3' '106.3' '115.6' '28.1' '36.1'
 '44.5' '7.9' '16.5']
----------------------------------

The unique values in column ISI:
['1.3' '1' '0.3' '0' '1.2' '3.1' '6.4' '5.6' '0.4' '4' '4.8' '0.5'
'0.7'
 '2.5' '0.9' '2.6' '2.4' '3.3' '5.7' '6.7' '9.2' '7.6' '2.2' '7.2'
'1.1'
 '0.8' '2.7' '2.8' '6' '1.5' '3' '1.4' '3.2' '4.6' '7.7' '5.2' '1.8'
'10'
 '8.7' '4.7' '6.8' '2' '1.7' '5.5' '6.9' '7.4' '7.1' '5.9' '3.7' '9.7'
 '8.8' '9.9' '10.4' '9' '8.2' '4.4' '7.3' '12.5' '0.6' '0.2' '0.1'
'2.1'
 '1.9' '6.2' '7.8' '4.5' '5.4' '8.4' '13.4' '5' '1.6' '4.9' '7' '8'
'11.7'
 '11.3' '4.3' '4.1' '8.3' '4.2' '10.9' '9.5' '18.5' '13.2' '13.8'
'17.2'
 '15.7' '19' '9.6' '16.6' '15.5' '7.5' '10.8' '3.5' '16' '3.8' '5.1'
 '11.5' '12.2' '14.3' '13.1' '8.1' '9.8' '9.1' '14.2' '11.2']
```

```
----------------------------------

The unique values in column BUI:
['3.4' '3.9' '2.7' '1.7' '7' '10.9' '13.5' '10.5' '12.6' '15.8' '17.7'
 '6.7' '4.4' '3' '2.2' '1.6' '2.4' '5.3' '5.1' '8.4' '9.7' '11.5'
'14.9'
 '18.3' '21.6' '25.8' '29.7' '23.8' '28.3' '2.9' '2.8' '5.7' '9.1'
'12.5'
 '12.1' '15.4' '7.4' '5.8' '8.1' '9.2' '11.7' '5.9' '8.3' '11.1'
'14.2'
 '18.2' '16.5' '22.4' '21.7' '14.7' '18.5' '23.9' '29.4' '32.1' '35'
 '37.4' '41.2' '4.7' '5.5' '8.2' '17.2' '14.1' '17.9' '21.9' '25.5'
'20.7'
 '24.4' '27.2' '22' '17.6' '22.9' '27.5' '31.3' '34.7' '38.8' '43.1'
 '47.5' '50.9' '54.7' '57.1' '59.3' '62.9' '67.4' '1.8' '1.1' '5.6'
'2.6'
 '3.7' '1.4' '4.2' '7.7' '11.3' '16' '19.2' '12.9' '9.6' '6.2' '9'
'6.8'
 '6.5' '9.3' '10.7' '7.3' '13.1' '18' '21.2' '6.1' '7.1' '4.1' '3.8'
'9.9'
 '12.7' '16.4' '20.8' '27.1' '17.8' '3.3' '7.8' '10.3' '18.7' '16.7'
 '13.7' '9.4' '10.4' '20.9' '27.7' '32.6' '39.5' '44' '46.5' '11.4'
'11.8'
 '15.7' '19.5' '10.6' '16.9' '23.5' '6.9' '11' '18.4' '17.5' '22.3'
'19'
 '24.2' '30.4' '35.9' '35.5' '38.1' '41.3' '45.5' '50.2' '54.9' '59.5'
 '64' '68' '30.6' '35.7' '39.3' '4' '6' '3.5' '6.4' '10' '4.6' '6.6'
 '12.4' '14.3' '26.2' '28.2' '28.9' '32.4' '36' '11.9' '4.8']
----------------------------------

The unique values in column FWI:
['0.5' '0.4' '0.1' '0' '2.5' '7.2' '7.1' '0.3' '0.9' '5.6' '7.1 '
'0.2'
 '1.4' '2.2' '2.3' '3.8' '7.5' '8.4' '10.6' '15' '13.9' '3.9' '12.9'
'1.7'
 '4.9' '6.8' '3.2' '8' '0.6' '3.4' '0.8' '3.6' '6' '10.9' '4' '8.8'
'2.8'
 '2.1' '1.3' '7.3' '15.3' '11.3' '11.9' '10.7' '15.7' '6.1' '2.6'
'9.9'
 '11.6' '12.1' '4.2' '10.2' '6.3' '14.6' '16.1' '17.2' '16.8' '18.4'
 '20.4' '22.3' '20.9' '20.3' '13.7' '13.2' '19.9' '30.2' '5.9' '7.7'
'9.7'
 '8.3' '0.7' '4.1' '1' '3.1' '1.9' '10' '16.7' '1.2' '5.3' '6.7' '9.5'
 '12' '6.4' '5.2' '3' '9.6' '4.7' 'fire   ' '14.1' '9.1' '13' '17.3'
'30'
 '25.4' '16.3' '9' '14.5' '13.5' '19.5' '12.6' '12.7' '21.6' '18.8'
'10.5'
 '5.5' '14.8' '24' '26.3' '12.2' '18.1' '24.5' '26.9' '31.1' '30.3'
'26.1'
 '16' '19.4' '2.7' '3.7' '10.3' '5.7' '9.8' '19.3' '17.5' '15.4'
```

```
'15.2'
 '6.5']
----------------------------------

The unique values in column Classes:
[ 0.  1. nan]
----------------------------------

The unique values in column Region:
[1 0]
----------------------------------
```

Observations:

- There is a value `14.6 9` in column `DC` that we need to rectify.
- Also a value `fire` in the column `FWI`, that also needed to be rectified. We will transform this `fire` to `0`.

Handling the errors

```
df['DC'] = df['DC'].str.split(' ').str[0]
df['FWI'] = df['FWI'].str.replace('fire','0')
df.head()
```

```
  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       1
1      0.0       1
2      0.0       1
3      0.0       1
4      0.0       1
```

Let's check the datatypes of the columns

```
df.dtypes
```

```
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        float64
Region           int64
dtype: object
```

Observations:

- Other than `Classes` and `Region` all other are object types, though they have numerical values.
- There are also some columns like `date`, `month`, `year` which we don't require here, so instead we create a new column as `Date`. Then we can drop them.

```
df.head()
```

```
  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       1  
1      0.0       1  
2      0.0       1  
3      0.0       1  
4      0.0       1  
```

Converting the datatypes of the columns, creating new column and drop the unnecessary columns.

```
# Converting the data types
df = df.astype({'day':int, 'month':int, 'year':int, 
'Temperature':float, 'RH':int, 'Ws':int, 'Rain':float,
                'FFMC':float, 'DMC':float, 'DC':float, 'ISI':float, 
"BUI":float, 'FWI':float})
```

```python
# checking the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 244 entries, 0 to 245
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   day          244 non-null    int32
 1   month        244 non-null    int32
 2   year         244 non-null    int32
 3   Temperature  244 non-null    float64
 4   RH           244 non-null    int32
 5   Ws           244 non-null    int32
 6   Rain         244 non-null    float64
 7   FFMC         244 non-null    float64
 8   DMC          244 non-null    float64
 9   DC           244 non-null    float64
 10  ISI          244 non-null    float64
 11  BUI          244 non-null    float64
 12  FWI          244 non-null    float64
 13  Classes      243 non-null    float64
 14  Region       244 non-null    int64
dtypes: float64(9), int32(5), int64(1)
memory usage: 33.8 KB
```

```python
# creating new column

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

```
   day  month  year  Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI
BUI  \
0    1      6  2012         29.0  57  18   0.0  65.7  3.4   7.6  1.3
3.4
1    2      6  2012         29.0  61  13   1.3  64.4  4.1   7.6  1.0
3.9
2    3      6  2012         26.0  82  22  13.1  47.1  2.5   7.1  0.3
2.7
3    4      6  2012         25.0  89  13   2.5  28.6  1.3   6.9  0.0
1.7
4    5      6  2012         27.0  77  16   0.0  64.8  3.0  14.2  1.2
3.9

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

```python
# dropping the unnecessary columns

df.drop(columns=['day', 'month', 'year'], axis=1, inplace=True)
df.head()
```

```
   Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI  BUI  FWI  Classes  
Region  \
0         29.0  57  18   0.0  65.7  3.4   7.6  1.3  3.4  0.5      0.0  
1  
1         29.0  61  13   1.3  64.4  4.1   7.6  1.0  3.9  0.4      0.0  
1  
2         26.0  82  22  13.1  47.1  2.5   7.1  0.3  2.7  0.1      0.0  
1  
3         25.0  89  13   2.5  28.6  1.3   6.9  0.0  1.7  0.0      0.0  
1  
4         27.0  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5      0.0  
1  

        Date  
0 2012-06-01  
1 2012-06-02  
2 2012-06-03  
3 2012-06-04  
4 2012-06-05  
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 244 entries, 0 to 245
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype         
---  ------       --------------  -----         
 0   Temperature  244 non-null    float64       
 1   RH           244 non-null    int32         
 2   Ws           244 non-null    int32         
 3   Rain         244 non-null    float64       
 4   FFMC         244 non-null    float64       
 5   DMC          244 non-null    float64       
 6   DC           244 non-null    float64       
 7   ISI          244 non-null    float64       
 8   BUI          244 non-null    float64       
 9   FWI          244 non-null    float64       
 10  Classes      243 non-null    float64       
 11  Region       244 non-null    int64         
 12  Date         244 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(9), int32(2), int64(1)
memory usage: 32.9 KB
```

Observations:

- Now all the column data types are changed, and unnecessary columns are dropped.

- We have 9 (float64) kind, 2 (int32) kind, 1 (datetime64) kind and 1 (int64) kind data.

```
# seeing the dataframe

df.head()
```

```
   Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI  BUI  FWI  Classes  
Region  \
0         29.0  57  18   0.0  65.7  3.4   7.6  1.3  3.4  0.5      0.0  
1
1         29.0  61  13   1.3  64.4  4.1   7.6  1.0  3.9  0.4      0.0  
1
2         26.0  82  22  13.1  47.1  2.5   7.1  0.3  2.7  0.1      0.0  
1
3         25.0  89  13   2.5  28.6  1.3   6.9  0.0  1.7  0.0      0.0  
1
4         27.0  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5      0.0  
1

        Date
0 2012-06-01
1 2012-06-02
2 2012-06-03
3 2012-06-04
4 2012-06-05
```

Checking null values and duplicated values

```
df.isnull().sum()
```

```
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        1
Region         0
Date           0
dtype: int64
```

Observations:

- There is only 1 null value in the `Classes` column.

```
# let's see the row with the null value

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

```
     Temperature  RH  Ws  Rain  FFMC   DMC    DC   ISI   BUI  FWI
Classes  \
167         37.0  37  18   0.2  88.9  12.9  14.6  12.5  10.4  0.0
NaN

     Region       Date
167       0 2012-07-14
```

Observations:

- As the index of the row is more than 122 so we can place it in the `Sidi-Bel Abbes` region and provide it with value `0`.

```
df['Classes'] = df['Classes'].fillna(0)

# Again check for null values
df.isnull().sum()
```

```
Temperature    0
RH             0
Ws             0
Rain           0
FFMC           0
DMC            0
DC             0
ISI            0
BUI            0
FWI            0
Classes        0
Region         0
Date           0
dtype: int64
```

```
# Checking duplicates

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

```
Temperature    0.0
RH             0.0
Ws             0.0
Rain           0.0
FFMC           0.0
DMC            0.0
DC             0.0
ISI            0.0
BUI            0.0
FWI            0.0
Classes        0.0
Region         0.0
dtype: float64
```

Observations:

- Now there is no null values and also the dataframe has no duplicate values.

Let's save clean dataset for future use

```
try:
    df.to_csv("dataset/Algerian_forest_cleaned.csv")
except Exception as err:
    print("Error is: ", err)
else:
    print("Clean csv file created successfully.")
```

```
Clean csv file created successfully.
```

4. Exploratory data analysis

Using the cleaned dataframe

```
df = pd.read_csv("dataset/Algerian_forest_cleaned.csv", index_col=0)
df.head()
```

```
   Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI  BUI  FWI  Classes  
Region  \
0         29.0  57  18   0.0  65.7  3.4   7.6  1.3  3.4  0.5      0.0  
1
1         29.0  61  13   1.3  64.4  4.1   7.6  1.0  3.9  0.4      0.0  
1
2         26.0  82  22  13.1  47.1  2.5   7.1  0.3  2.7  0.1      0.0  
1
3         25.0  89  13   2.5  28.6  1.3   6.9  0.0  1.7  0.0      0.0  
1
4         27.0  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5      0.0  
1

         Date  
0  2012-06-01  
1  2012-06-02  
2  2012-06-03  
3  2012-06-04  
4  2012-06-05  
```

4.1 Basic Profile of the data

```
# Checking the details of the dataframe

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 244 entries, 0 to 245
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  244 non-null    float64
 1   RH           244 non-null    int64  
```

```
 2   Ws           244 non-null    int64
 3   Rain         244 non-null    float64
 4   FFMC         244 non-null    float64
 5   DMC          244 non-null    float64
 6   DC           244 non-null    float64
 7   ISI          244 non-null    float64
 8   BUI          244 non-null    float64
 9   FWI          244 non-null    float64
 10  Classes      244 non-null    float64
 11  Region       244 non-null    int64
 12  Date         244 non-null    object
dtypes: float64(9), int64(3), object(1)
memory usage: 26.7+ KB
```

Differentiating numerical and categorical columns

```
numerical_features = [feature for feature in df.columns if 
df[feature].dtypes != 'O']
categorical_features = [feature for feature in df.columns if 
df[feature].dtypes == 'O']


print(f"The number of Numerical features are: 
{len(numerical_features)}, and the column names are:\
n{numerical_features}")
print(f"\nThe number of Categorical features are: 
{len(categorical_features)}, and the column names are:\
n{categorical_features}")
```

```
The number of Numerical features are: 12, and the column names are:
['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 
'FWI', 'Classes', 'Region']

The number of Categorical features are: 1, and the column names are:
['Date']
```

Observations:

- In total there are 244 rows and 13 columns in the dataset.
- There are no null values in the dataset.
- Also we have 12 numerical columns and 1 categorical column.

4.2 Statistical Analysis of the data

```
df.describe().T
```

```
             count       mean        std   min     25%    50%     75%  
max
Temperature  244.0  32.172131   3.633843  22.0  30.000  32.00  35.000  
42.0
RH           244.0  61.938525  14.884200  21.0  52.000  63.00  73.250  
90.0
```

```
Ws            244.0  15.504098   2.810178   6.0  14.000  15.00  17.000
29.0
Rain          244.0   0.760656   1.999406   0.0   0.000   0.00   0.500
16.8
FFMC          244.0  77.887705  14.337571  28.6  72.075  83.50  88.300
96.0
DMC           244.0  14.673361  12.368039   0.7   5.800  11.30  20.750
65.9
DC            244.0  49.288115  47.619662   6.9  13.275  33.10  68.150
220.4
ISI           244.0   4.774180   4.175318   0.0   1.400   3.50   7.300
19.0
BUI           244.0  16.664754  14.204824   1.1   6.000  12.25  22.525
68.0
FWI           244.0   7.006557   7.438889   0.0   0.700   4.20  11.375
31.1
Classes       244.0   0.561475   0.497226   0.0   0.000   1.00   1.000
1.0
Region        244.0   0.500000   0.501028   0.0   0.000   0.50   1.000
1.0
```

Observations:

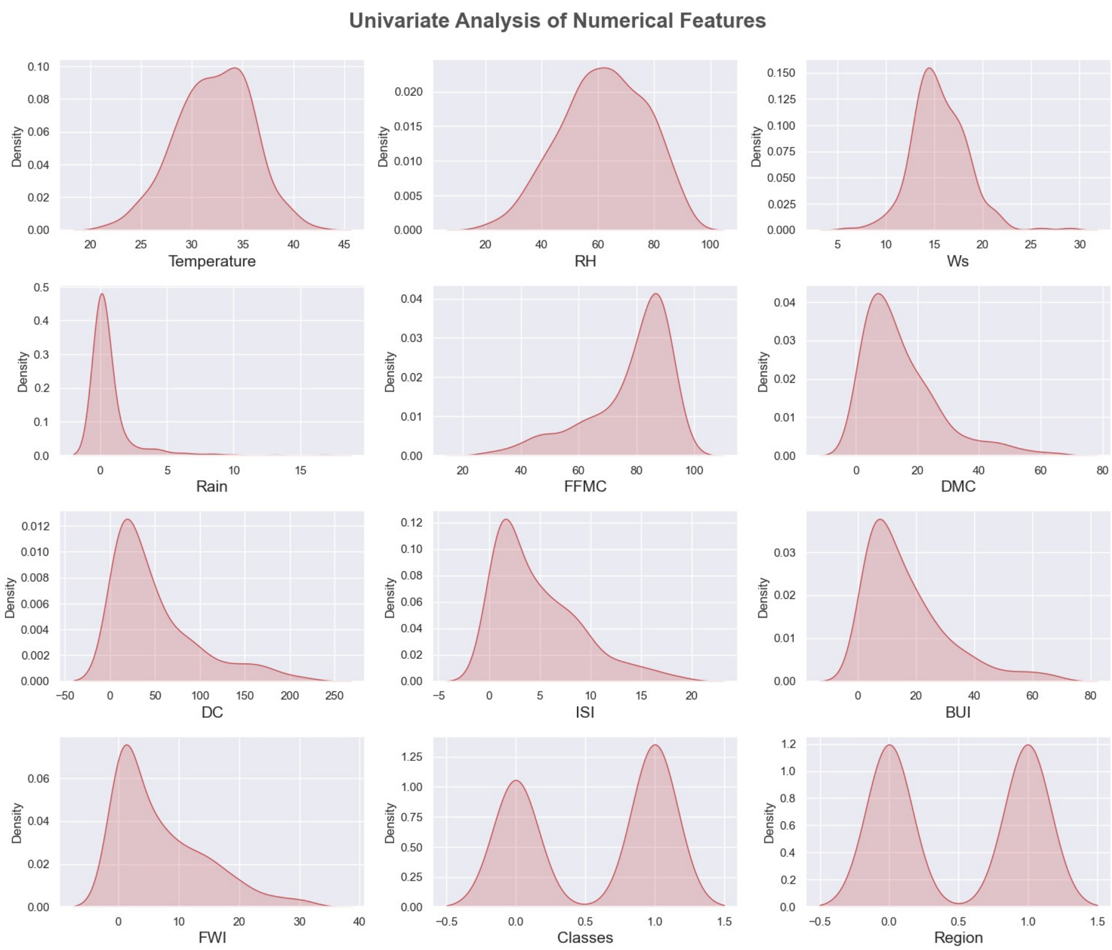
- There are possible Outliers in columns `Rain`, `DMC`, `DC`, `ISI`, `BUI`, `FWI`.

4.3 Graphical Analysis of the data

4.3.1 Univariate Analysis

- The univariate analysis is used to understand the distribution of values for a single variable.

```
# For numerical features

plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(numerical_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[numerical_features[i]],shade=True, color='r')
    plt.xlabel(numerical_features[i], fontsize=15)
    plt.tight_layout()
```

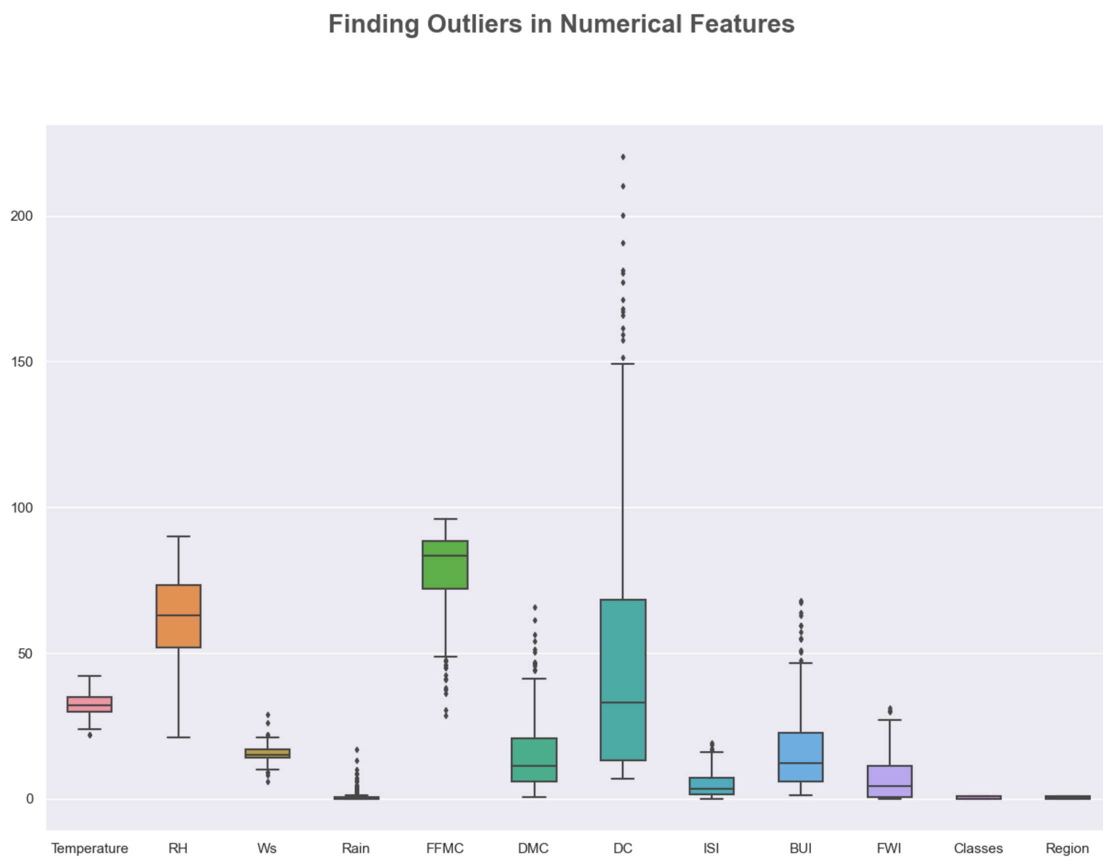


Observations:

- We can see there is some skewness in the data.
- The `Rain`, `DMC`, `DC`, `ISI`, `BUI`, `FWI` are mainly right skewed.
- The `FFMC` is left skewed.
- The `Temperature`, `RH`, `WS` has almost normal distribution, although `WS` seems to have some right skewness and `RH` has some left skewness.
- The `Classes` and `Region` though have numeric value but they mainly represent categorical variables.
- There are also outliers in many columns.

```
fig, ax = plt.subplots(figsize=(15,10))
plt.suptitle('Finding Outliers in Numerical Features', fontsize=20, fontweight='bold', alpha=0.8, y=1.)
sns.boxplot(data=df, width= 0.5, ax=ax, fliersize=3)
```

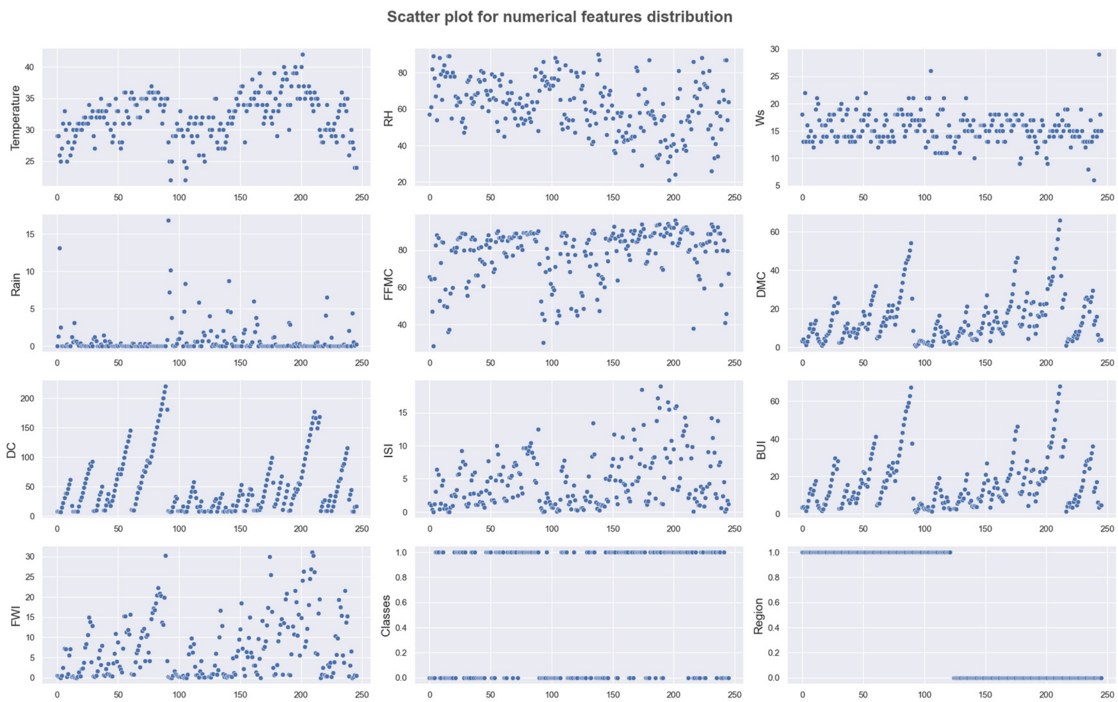
```
<AxesSubplot:>
```



Observations:

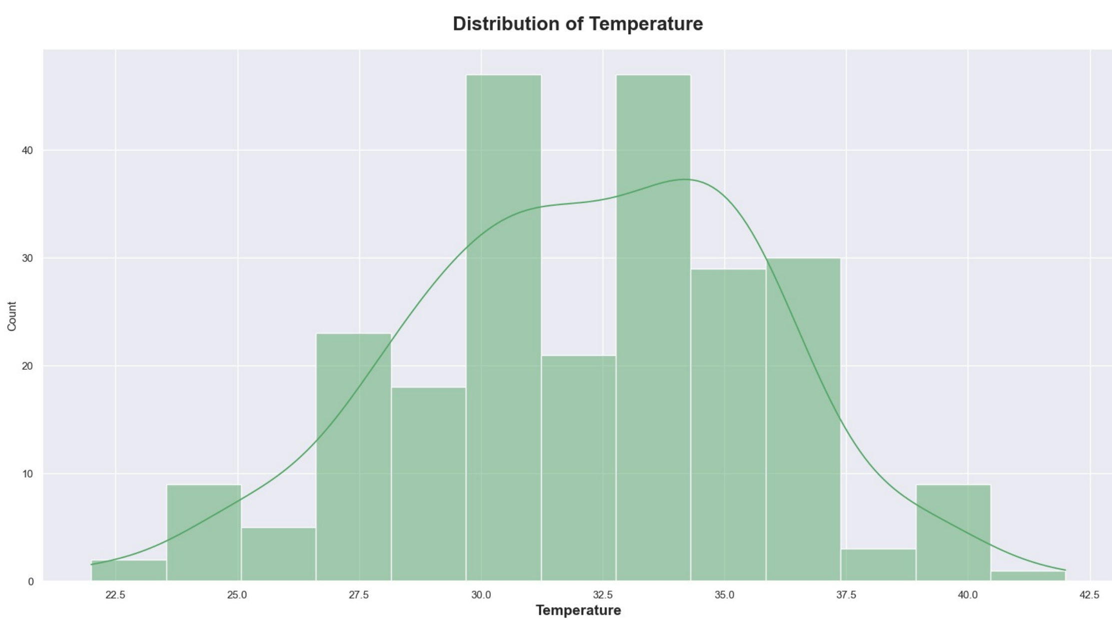
- There is an outlier in the lower side of the `Temperature` feature.
- There are outliers in both side of the `Ws` feature.
- There are outliers only in the upper side of the `Rain`, `DMC`, `DC`, `ISI`, `BUI`, `FWI` features.
- There are outliers only in the lower side of the `FFMC` features.
- It seems the most number of outliers are in `DC` feature.
- There is no outlier in the `RH` feature.

```
plt.figure(figsize=(20,15))
plt.suptitle('Scatter plot for numerical features distribution',
fontsize=20, fontweight='bold', alpha=0.8, y=1.)
for i in range(0, len(numerical_features)):
    plt.subplot(5, 3, i+1)
    sns.scatterplot(y=df[numerical_features[i]], x=df.index, data=df)
    plt.ylabel(numerical_features[i], fontsize=15)
    plt.tight_layout()
```




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

```
Text(0.5, 0, 'Temperature')
```



Observations:

- Most of the teperatures recorded between the range of 30 to 34.

Note:

- As there is only one categorical variable and that too about date so no need for graphical representation.

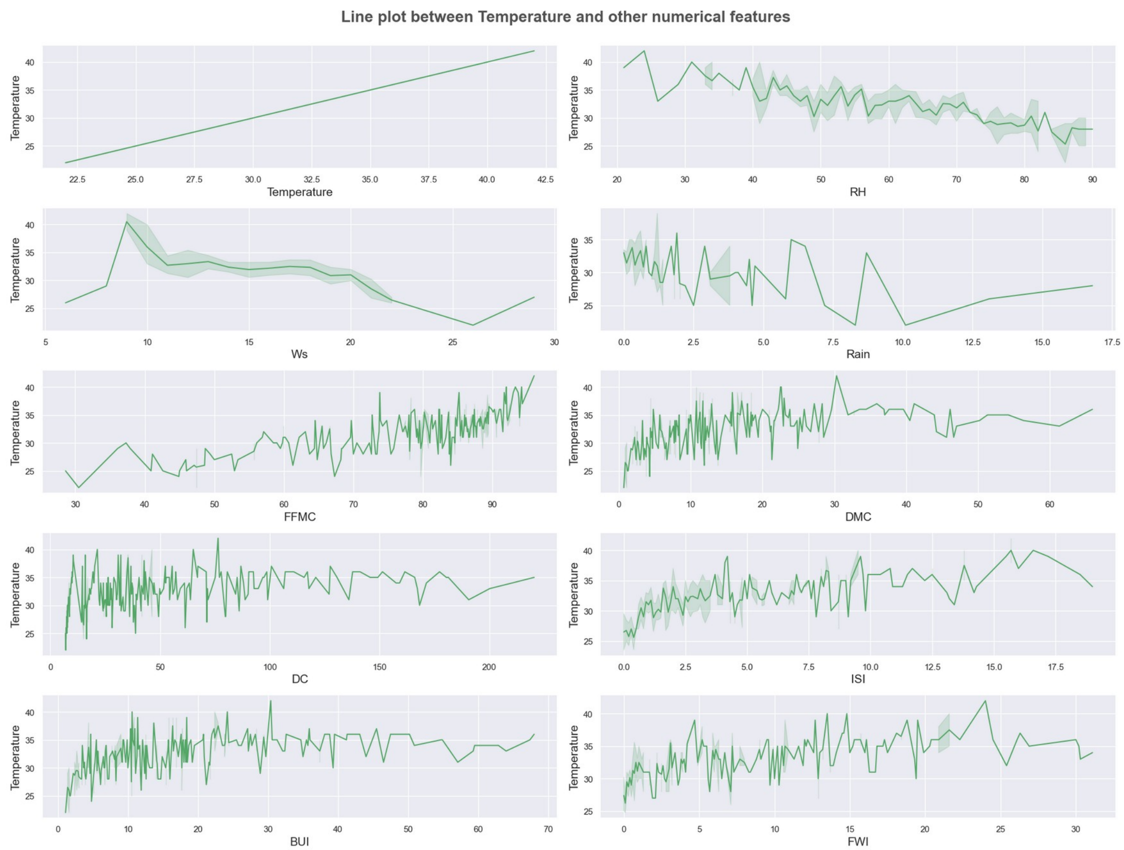
4.3.2 Bivariate Analysis

- Bivariate analysis is the analysis of two variables to find out relationship between them.
- Here we will use lineplot to see the relationship between `Temperature` and other numerical variables leaving `Classes` and `Region`.

```
# Creating a dataframe leaving the two columns 'Classes' and 'Region'

df_numeric = df[numerical_features]
df_numeric = df_numeric.iloc[:, :-2]
df_numeric.head()
```

```
   Temperature  RH  Ws  Rain  FFMC  DMC    DC  ISI  BUI  FWI
0         29.0  57  18   0.0  65.7  3.4   7.6  1.3  3.4  0.5
1         29.0  61  13   1.3  64.4  4.1   7.6  1.0  3.9  0.4
2         26.0  82  22  13.1  47.1  2.5   7.1  0.3  2.7  0.1
3         25.0  89  13   2.5  28.6  1.3   6.9  0.0  1.7  0.0
4         27.0  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5
```

```
plt.figure(figsize=(20,15))
plt.suptitle('Line plot between Temperature and other numerical
features', fontsize=20, fontweight='bold', alpha=0.8, y=1.)

column_names = df_numeric.columns

for i in range(0, len(column_names)):
    plt.subplot(5, 2, i+1)
    sns.lineplot(y=df_numeric['Temperature'], x=df[column_names[i]],
data=df_numeric, color='g')
    plt.ylabel("Temperature", fontsize=15)
    plt.xlabel(column_names[i], fontsize=15)
    plt.tight_layout()
```



Observations:

- The temperature decreases with increase in Relative humidity (RH).
- The temperature increases with increase in Fine Fuel Moisture Code (FFMC).
- The temperature decreases upto a certain point with increase in Wind speed (WS) then it starts to increase.
- The temperature flactuates with amount of Rain then after a certain point it starts to increase. Same happens with Drought Code (DC).
- After a certain point the temperature starts to decrease with increase in Initial Spread Index (ISI). At start it was flactuating.

4.3.2 Multivariate Analysis

- Multivariate analysis is the analysis of more than one variable.

Checking Multicollinearity in the numerical features

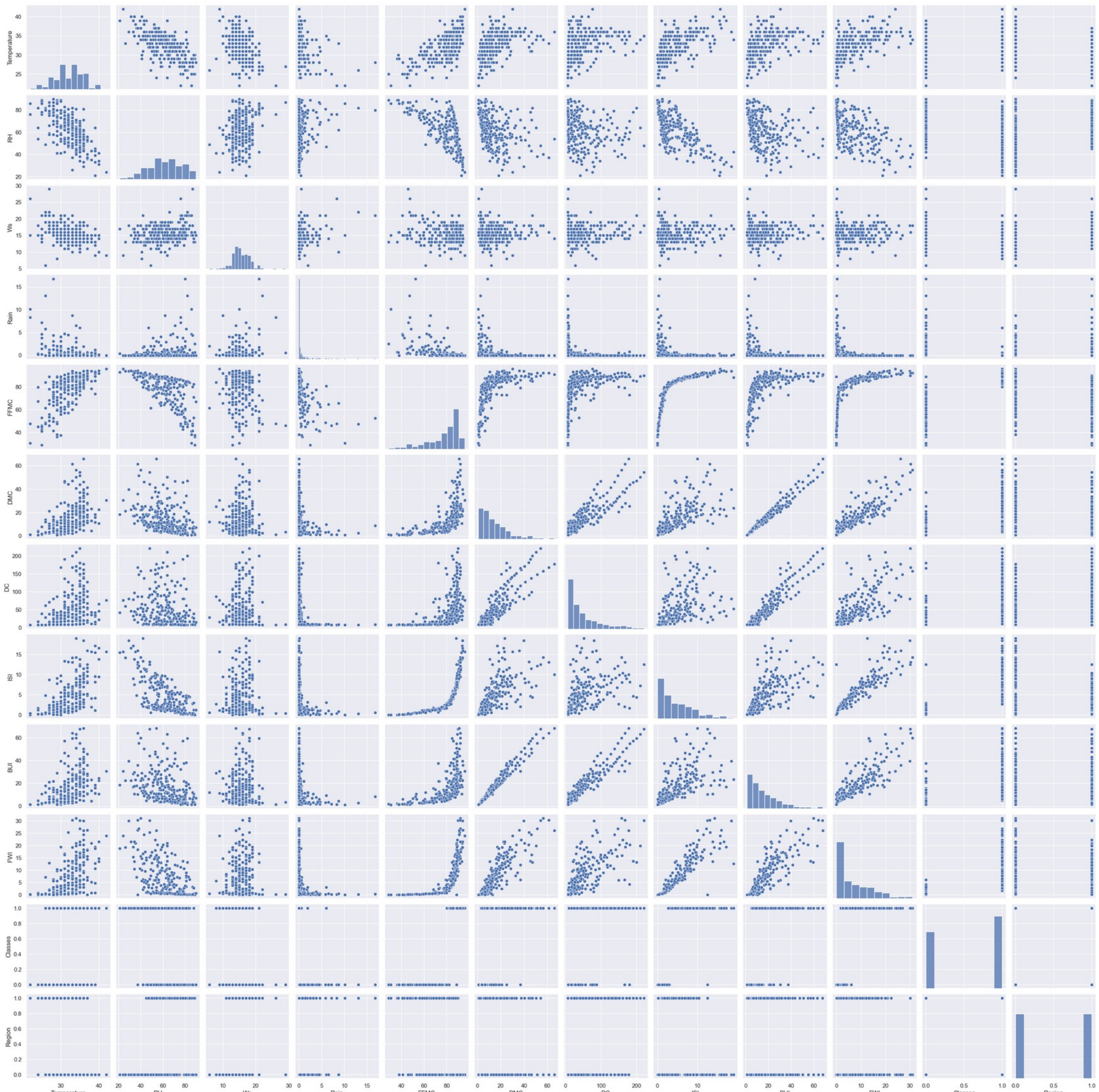
```
df[list(df[numerical_features].columns)].corr()

             Temperature        RH        Ws      Rain      FFMC
DMC  \
Temperature     1.000000 -0.654443 -0.278132 -0.326786  0.677491
0.483105
RH             -0.654443  1.000000  0.236084  0.222968 -0.645658 -
0.405133
```

```
Ws              -0.278132  0.236084  1.000000  0.170169 -0.163255 -
0.001246
Rain            -0.326786  0.222968  0.170169  1.000000 -0.544045 -
0.288548
FFMC             0.677491 -0.645658 -0.163255 -0.544045  1.000000
0.602391
DMC              0.483105 -0.405133 -0.001246 -0.288548  0.602391
1.000000
DC               0.370498 -0.220330  0.076245 -0.296804  0.503910
0.875358
ISI              0.607551 -0.690637  0.015248 -0.347105  0.739730
0.674499
BUI              0.455504 -0.348587  0.029756 -0.299171  0.589652
0.982073
FWI              0.558393 -0.569997  0.028799 -0.322682  0.686033
0.874778
Classes          0.506575 -0.420695 -0.073810 -0.376727  0.762942
0.584757
Region          -0.273496  0.406424  0.176829  0.041080 -0.224680 -
0.191094

                   DC       ISI       BUI       FWI   Classes
Region
Temperature  0.370498  0.607551  0.455504  0.558393  0.506575 -
0.273496
RH          -0.220330 -0.690637 -0.348587 -0.569997 -0.420695
0.406424
Ws           0.076245  0.015248  0.029756  0.028799 -0.073810
0.176829
Rain        -0.296804 -0.347105 -0.299171 -0.322682 -0.376727
0.041080
FFMC         0.503910  0.739730  0.589652  0.686033  0.762942 -
0.224680
DMC          0.875358  0.674499  0.982073  0.874778  0.584757 -
0.191094
DC           1.000000  0.498909  0.941904  0.740189  0.512615
0.081489
ISI          0.498909  1.000000  0.635891  0.907461  0.719419 -
0.268421
BUI          0.941904  0.635891  1.000000  0.857771  0.586915 -
0.087370
FWI          0.740189  0.907461  0.857771  1.000000  0.720398 -
0.192451
Classes      0.512615  0.719419  0.586915  0.720398  1.000000 -
0.156928
Region       0.081489 -0.268421 -0.087370 -0.192451 -0.156928
1.000000
```

Graphical representation

```
sns.pairplot(df[numerical_features])
```

```
<seaborn.axisgrid.PairGrid at 0x23ec4d02040>
```




```
sns.set(rc={'figure.figsize':(15,10)})
sns.heatmap(df[numerical_features].corr(), cmap='CMRmap', annot=True)
```

```
<AxesSubplot:>
```

Observations:

- BUI has high positive correlation with columns DMC, DC and FWI.
- ISI is very highly positively correlated with FWI and negatively correlated with RH and Rain.
- DC and DMC also positively correlated.
- FWI and DMC also positively correlated.
- RH and FFMC has negative correlation.

5. Data Pre-Processing

```
# Seeing the original cleaned dataset

df.head()

   Temperature  RH  Ws  Rain  FFMC  DMC   DC  ISI  BUI  FWI  Classes  
Region  \
0         29.0  57  18   0.0  65.7  3.4  7.6  1.3  3.4  0.5      0.0  
1
1         29.0  61  13   1.3  64.4  4.1  7.6  1.0  3.9  0.4      0.0  
1
2         26.0  82  22  13.1  47.1  2.5  7.1  0.3  2.7  0.1      0.0  
1
3         25.0  89  13   2.5  28.6  1.3  6.9  0.0  1.7  0.0      0.0  
1
```

```
4         27.0  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5      0.0
1

         Date
0  2012-06-01
1  2012-06-02
2  2012-06-03
3  2012-06-04
4  2012-06-05
```

Number of unique values in each column

```
df.nunique()
```

```
Temperature     19
RH              62
Ws              18
Rain            39
FFMC           173
DMC            166
DC             198
ISI            106
BUI            174
FWI            125
Classes          2
Region           2
Date           122
dtype: int64
```

5.1 Separating Different Features

Numerical features

```
num_features = [feature for feature in df.columns if df[feature].dtype
!= 'O']
print(f'Number of Numerical Features is {len(num_features)} and they
are: \n{num_features}')
```

```
Number of Numerical Features is 12 and they are:
['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
'FWI', 'Classes', 'Region']
```

Categorical features

```
cat_features = [feature for feature in df.columns if df[feature].dtype
== 'O']
print(f'Number of Categorical Features is {len(cat_features)} and they
are: \n{cat_features}')
```

```
Number of Categorical Features is 1 and they are:
['Date']
```

Discrete features

```
dis_features = [feature for feature in num_features if
len(df[feature].unique()) <= 10]
print(f'Number of Discrete Features is {len(dis_features)} and they
are: \n{dis_features}')
```

```
Number of Discrete Features is 2 and they are:
['Classes', 'Region']
```

Continuous features

```
con_features = [feature for feature in num_features if feature not in
dis_features]
print(f'Number of Continuous Features is {len(con_features)} and they
are: \n{con_features}')
```

```
Number of Continuous Features is 10 and they are:
['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
'FWI']
```

5.2 Outlier handling

Detecting Outlier and Capping it

- Trimming outliers may result in the removal of a large number of records from this dataset as we have already very less rows so this isn't desirable in this case since columns other than the ones containing the outlier values may contain useful information.
- In this cases, we can use outlier capping to replace the outlier values with a maximum or minimum capped values. Be warned, this manipulates our data but we can replace outlier values by the upper and lower limit calculated using the IQR range.

```
# Creating a function to detect outliers

def detect_outliers(col):
    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)
    print('\n ####', col , '####')
    print("25percentile: ",percentile25)
    print("75percentile: ",percentile75)
    iqr = percentile75 - percentile25
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    print("Upper limit: ",upper_limit)
    print("Lower limit: ",lower_limit)
    df.loc[(df[col]>upper_limit), col]= upper_limit
    df.loc[(df[col]<lower_limit), col]= lower_limit
    return df
```

```python
# Now applying the function on columns

for col in con_features:
        detect_outliers(col)
```

```
 #### Temperature ####
25percentile:  30.0
75percentile:  35.0
Upper limit:  42.5
Lower limit:  22.5

 #### RH ####
25percentile:  52.0
75percentile:  73.25
Upper limit:  105.125
Lower limit:  20.125

 #### Ws ####
25percentile:  14.0
75percentile:  17.0
Upper limit:  21.5
Lower limit:  9.5

 #### Rain ####
25percentile:  0.0
75percentile:  0.5
Upper limit:  1.25
Lower limit:  -0.75

 #### FFMC ####
25percentile:  72.075
75percentile:  88.3
Upper limit:  112.63749999999999
Lower limit:  47.73750000000001

 #### DMC ####
25percentile:  5.8
75percentile:  20.75
Upper limit:  43.175
Lower limit:  -16.624999999999996

 #### DC ####
25percentile:  13.274999999999999
75percentile:  68.15
Upper limit:  150.46250000000003
Lower limit:  -69.03750000000002

 #### ISI ####
```

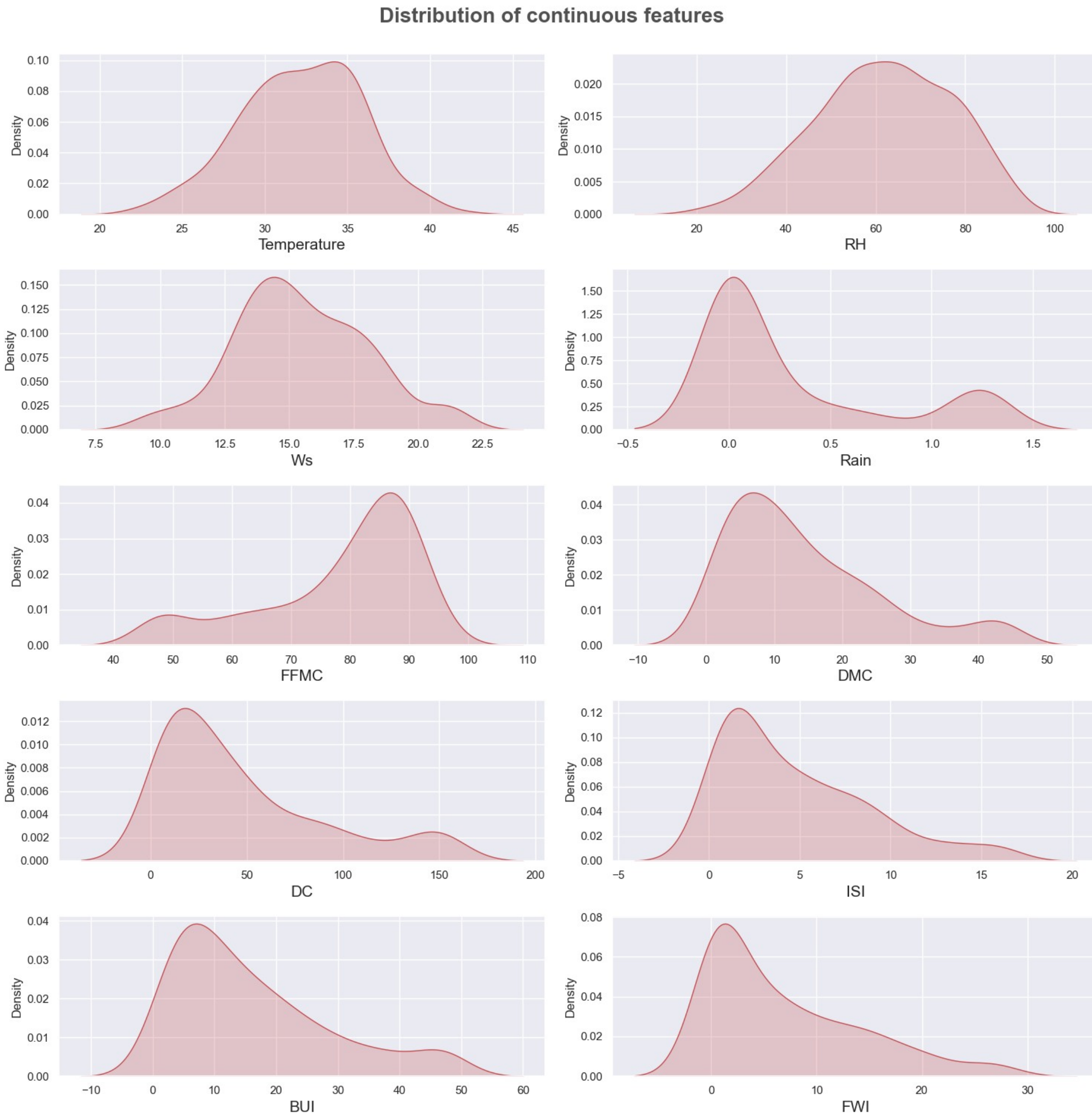
```
25percentile:  1.4
75percentile:  7.3
Upper limit:  16.150000000000002
Lower limit:  -7.450000000000001

 #### BUI ####
25percentile:  6.0
75percentile:  22.525
Upper limit:  47.3125
Lower limit:  -18.787499999999998

 #### FWI ####
25percentile:  0.7
75percentile:  11.375
Upper limit:  27.387500000000003
Lower limit:  -15.312500000000004
```

Checking Skewness after Outlier Capping

```
df[con_features].skew(axis=0, skipna=True)
```

```
Temperature   -0.175783
RH            -0.237964
Ws             0.177613
Rain           1.246290
FFMC          -1.073835
DMC            1.089909
DC             1.159322
ISI            1.021607
BUI            1.021143
FWI            1.057544
dtype: float64
```

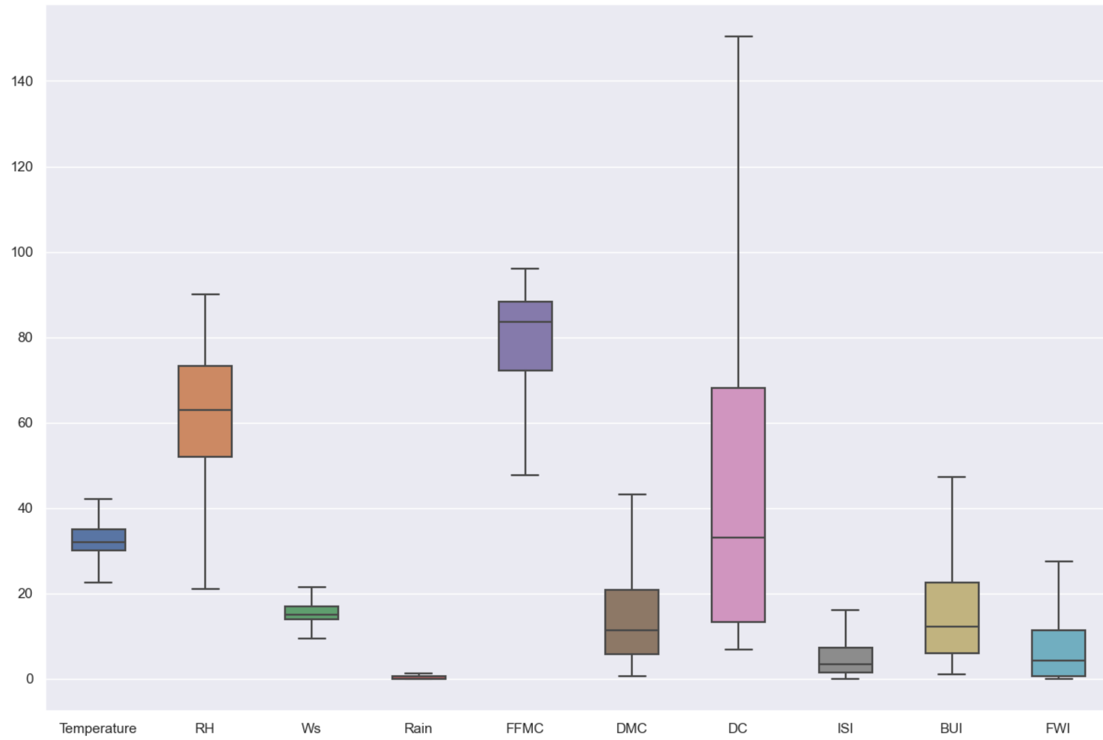
```
# Again For continuous features

plt.figure(figsize=(15, 15))
plt.suptitle('Distribution of continuous features', fontsize=20, 
fontweight='bold', alpha=0.8, y=1.)

for i in range(0, len(con_features)):
    plt.subplot(5, 2, i+1)
    sns.kdeplot(x=df[con_features[i]],shade=True, color='r')
    plt.xlabel(con_features[i], fontsize=15)
    plt.tight_layout()
```




```python
fig, ax = plt.subplots(figsize=(15,10))
plt.suptitle('Finding Outliers in Numerical Features', fontsize=20,
fontweight='bold', alpha=0.8, y=1.)
sns.boxplot(data=df[con_features], width= 0.5, ax=ax, fliersize=3)
```

```
<AxesSubplot:>
```

Finding Outliers in Numerical Features



Observations:

- Now we can see all the outliers are capped.
- Also the distribution remains almost as same as with the outliers.

Adding the continuous and discrete features into the final dataset

```
df_final = pd.concat([df[con_features], df[dis_features]], axis=1)
df_final.head()
```

```
   Temperature    RH    Ws  Rain     FFMC  DMC    DC  ISI  BUI  FWI
Classes  \
0         29.0  57.0  18.0  0.00  65.7000  3.4   7.6  1.3  3.4  0.5
0.0
1         29.0  61.0  13.0  1.25  64.4000  4.1   7.6  1.0  3.9  0.4
0.0
2         26.0  82.0  21.5  1.25  47.7375  2.5   7.1  0.3  2.7  0.1
0.0
3         25.0  89.0  13.0  1.25  47.7375  1.3   6.9  0.0  1.7  0.0
0.0
4         27.0  77.0  16.0  0.00  64.8000  3.0  14.2  1.2  3.9  0.5
0.0

   Region
0       1
```

```
1       1
2       1
3       1
4       1
```

5.3 Creating independent and dependent variables

```
# Here 'X' is independent features and 'y' is dependent feature.

X = df_final.iloc[:, 1:]
y = df_final.iloc[:,0]

X.head()
```

```
     RH    Ws  Rain     FFMC  DMC    DC  ISI  BUI  FWI  Classes  
Region
0  57.0  18.0  0.00  65.7000  3.4   7.6  1.3  3.4  0.5      0.0  
1
1  61.0  13.0  1.25  64.4000  4.1   7.6  1.0  3.9  0.4      0.0  
1
2  82.0  21.5  1.25  47.7375  2.5   7.1  0.3  2.7  0.1      0.0  
1
3  89.0  13.0  1.25  47.7375  1.3   6.9  0.0  1.7  0.0      0.0  
1
4  77.0  16.0  0.00  64.8000  3.0  14.2  1.2  3.9  0.5      0.0  
1
```

```
y.head()
```

```
0    29.0
1    29.0
2    26.0
3    25.0
4    27.0
Name: Temperature, dtype: float64
```

```
# importing library to do test train split

from sklearn.model_selection import train_test_split

# Creating the test and train dataset
# Here 'test_size' is 0.33 means 33%
# Here 'random_state' is 42 so each time we run the code the result 
would be same

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

Let's check the shapes of each datasets

```
X_train.shape
```

```
(163, 11)
```

```
y_train.shape
```

```
(163,)
```

```
X_test.shape
```

```
(81, 11)
```

```
y_test.shape
```

```
(81,)
```

Let's see the datasets

```
X_train.head()
```

```
       RH    Ws  Rain  FFMC     DMC        DC   ISI      BUI      FWI  
\
114  54.0  11.0  0.50  73.7   7.900   30.4000   1.2   9.6000   0.7000   

65   65.0  13.0  0.00  86.8  11.100   29.7000   5.2  11.5000   6.1000   

134  42.0  21.0  0.00  90.6  18.200   30.5000  13.4  18.0000  16.7000   

209  40.0  18.0  0.00  92.1  43.175  150.4625  14.3  47.3125  27.3875   

164  56.0  15.0  1.25  74.8   7.100    9.5000   1.6   6.8000   0.8000   


     Classes  Region  
114      0.0       1  
65       1.0       1  
134      1.0       0  
209      1.0       0  
164      0.0       0  
```

```
y_train.head()
```

```
114    32.0
65     34.0
134    31.0
209    34.0
164    34.0
Name: Temperature, dtype: float64
```

```
X_test.head()
```

```
       RH    Ws  Rain  FFMC   DMC     DC  ISI   BUI   FWI  Classes  
Region  
24   64.0  15.0   0.0  86.7  14.2   63.8  5.7  18.3   8.4      1.0   
1  
6    54.0  13.0   0.0  88.2   9.9   30.5  6.4  10.9   7.2      1.0   
```

```
1
155  48.0  16.0   0.0  87.6   7.9   17.8  6.8   7.8   6.4      1.0
0
213  53.0  17.0   0.5  80.2  20.7  149.2  2.7  30.6   5.9      1.0
0
200  41.0  10.0   0.1  92.0  22.6   65.1  9.5  24.2  14.8      1.0
0
```

```
y_test.head()
```

```
24     31.0
6      33.0
155    33.0
213    35.0
200    40.0
Name: Temperature, dtype: float64
```

Observations:

- Now we have 163 rows for training and 81 for test datasets.

5.4 Standardizing or feature scaling the dataset (Feature Engineering)

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler
```

```
StandardScaler()
```

```
# Now doing fit_transform() the training dataset. Here it will first
fit the data and then transform it.
# That is it will compute the μ of the data points and then use σ to
create new data points on same scale.

X_train = scaler.fit_transform(X_train)
X_train
```

```
array([[-0.60257784, -1.82006847,  0.33656531, ..., -0.82812286,
        -1.04390785,  0.99388373],
       [ 0.14460201, -1.01389684, -0.70640323, ..., -0.07803797,
         0.95793896,  0.99388373],
       [-1.41768313,  2.2107897 , -0.70640323, ...,  1.39435088,
         0.95793896, -1.0061539 ],
       ...,
       [ 0.89178186,  0.59844643,  1.90101811, ..., -0.91146562,
        -1.04390785,  0.99388373],
       [-0.39880152,  0.19536061, -0.70640323, ...,  0.32478539,
         0.95793896, -1.0061539 ],
       [ 0.9597073 ,  2.2107897 ,  1.90101811, ..., -0.8836847 ,
        -1.04390785,  0.99388373]])
```

```python
# Here we do the transform() the test data. It will perform
standardization by centering and scaling.

X_test = scaler.transform(X_test)
X_test
```

```
array([[ 7.66765714e-02, -2.07725206e-01, -7.06403230e-01,
         6.98106296e-01,  1.51079264e-03,  3.83587360e-01,
         2.93741687e-01,  1.77156842e-01,  2.41442627e-01,
         9.57938964e-01,  9.93883735e-01],
       [-6.02577838e-01, -1.01389684e+00, -7.06403230e-01,
         8.08327377e-01, -3.73326220e-01, -3.74243447e-01,
         4.65398161e-01, -3.92522131e-01,  7.47570968e-02,
         9.57938964e-01,  9.93883735e-01],
       [-1.01013048e+00,  1.95360611e-01, -7.06403230e-01,
         7.64238945e-01, -5.47669017e-01, -6.63266007e-01,
         5.63487574e-01, -6.31171430e-01, -3.63665900e-02,
         9.57938964e-01, -1.00615390e+00],
       [-6.70503279e-01,  5.98446428e-01,  3.36565307e-01,
         2.20481608e-01,  5.68124882e-01,  2.32709339e+00,
        -4.41928915e-01,  1.12405568e+00, -1.05818894e-01,
         9.57938964e-01, -1.00615390e+00],
       [-1.48560857e+00, -2.22315429e+00, -4.97809522e-01,
         1.08755412e+00,  7.33750538e-01,  4.13172347e-01,
         1.22559112e+00,  6.31360347e-01,  1.13043212e+00,
         9.57938964e-01, -1.00615390e+00],
       [ 7.66765714e-02, -2.42469720e+00,  1.79672126e+00,
        -2.49795007e-01, -2.16417703e-01, -7.06505603e-01,
        -8.34286569e-01, -3.54030308e-01, -8.28122859e-01,
        -1.04390785e+00, -1.00615390e+00],
       [-1.62145945e+00, -2.07725206e-01, -2.89215815e-01,
         8.89156170e-01,  1.40985030e-01, -2.62730806e-01,
         9.06800522e-01, -1.53022705e-02,  5.60923227e-01,
         9.57938964e-01, -1.00615390e+00],
       [ 1.16348363e+00, -6.10811023e-01,  1.90101811e+00,
        -2.09416111e+00, -1.04454599e+00, -8.95394362e-01,
        -1.03046540e+00, -1.03148638e+00, -9.11465624e-01,
        -1.04390785e+00, -1.00615390e+00],
       [ 1.09555819e+00, -1.41698266e+00, -7.06403230e-01,
        -2.93883439e-01, -4.08194779e-01, -1.46720730e-02,
        -7.85241862e-01, -2.61649934e-01, -8.00341937e-01,
        -1.04390785e+00,  9.93883735e-01],
       [ 6.88005540e-01, -1.41698266e+00,  1.90101811e+00,
        -1.27117703e+00, -1.04454599e+00, -8.65809376e-01,
        -9.32375983e-01, -1.02378802e+00, -8.83684702e-01,
        -1.04390785e+00,  9.93883735e-01],
       [ 8.91781863e-01,  2.21078970e+00, -7.06403230e-01,
        -3.37971872e-01, -6.26123275e-01, -4.88031856e-01,
        -9.32375983e-01, -5.92679608e-01, -8.69794241e-01,
        -1.04390785e+00,  9.93883735e-01],
```

```
[-3.30876074e-01, -1.01389684e+00, -2.89215815e-01,
  1.69045104e-01,  3.93782085e-01,  9.34323262e-01,
 -5.89063035e-01,  6.46757076e-01, -3.97518572e-01,
 -1.04390785e+00, -1.00615390e+00],
[-6.02577838e-01, -1.82006847e+00, -7.06403230e-01,
  3.52746906e-01, -7.13294673e-01, -6.97402530e-01,
 -4.90973621e-01, -7.54345262e-01, -6.89218250e-01,
 -1.04390785e+00,  9.93883735e-01],
[-3.30876074e-01, -2.22315429e+00, -2.89215815e-01,
  8.08682384e-02, -2.94871962e-01, -6.20026411e-01,
 -7.11674802e-01, -4.07918860e-01, -7.86451476e-01,
 -1.04390785e+00, -1.00615390e+00],
[-5.34652397e-01,  1.00153224e+00, -7.06403230e-01,
  8.74460026e-01,  1.68391878e+00,  2.35582497e+00,
  1.32368053e+00,  2.08635124e+00,  1.90829793e+00,
  9.57938964e-01,  9.93883735e-01],
[ 4.84229217e-01,  1.95360611e-01, -7.06403230e-01,
  6.46669791e-01, -1.20529165e-01, -4.85756088e-01,
  2.20174627e-01, -2.53951570e-01,  5.30479256e-03,
  9.57938964e-01, -1.00615390e+00],
[-1.75731033e+00, -6.10811023e-01, -7.06403230e-01,
  1.25655978e+00,  1.89313014e+00,  9.07014044e-01,
  2.81954409e+00,  1.53206899e+00,  2.72783512e+00,
  9.57938964e-01, -1.00615390e+00],
[ 1.77481259e+00, -1.01389684e+00,  7.53752722e-01,
 -2.16488630e+00, -1.08813169e+00, -8.95394362e-01,
 -1.10403246e+00, -1.06227984e+00, -9.25356084e-01,
 -1.04390785e+00,  9.93883735e-01],
[ 8.23856422e-01, -6.10811023e-01, -7.06403230e-01,
  6.76062079e-01, -1.03094885e-01, -1.78527383e-01,
  1.71129920e-01, -1.38476103e-01,  1.91952534e-02,
  9.57938964e-01,  9.93883735e-01],
[-5.34652397e-01, -6.10811023e-01, -7.06403230e-01,
  6.61365935e-01, -5.12800457e-01, -6.49611398e-01,
  1.22085213e-01, -6.00377972e-01, -2.44723503e-01,
  9.57938964e-01, -1.00615390e+00],
[ 1.02763274e+00,  5.98446428e-01, -7.06403230e-01,
  2.13133536e-01, -8.35334631e-01, -6.47335630e-01,
 -4.41928915e-01, -7.92837085e-01, -6.89218250e-01,
 -1.04390785e+00,  9.93883735e-01],
[ 1.16348363e+00,  1.95360611e-01,  1.27971600e-01,
 -1.27852510e+00, -9.39940309e-01, -4.51619565e-01,
 -8.83331276e-01, -8.39027272e-01, -8.69794241e-01,
 -1.04390785e+00,  9.93883735e-01],
[-9.42205043e-01, -1.82006847e+00, -7.06403230e-01,
  8.96504243e-01, -3.82043360e-01, -3.15073474e-01,
  5.63487574e-01, -3.61728673e-01,  1.44209401e-01,
  9.57938964e-01,  9.93883735e-01],
[ 3.48378335e-01,  1.40461806e+00,  1.37953384e+00,
 -1.27117703e+00, -1.01839457e+00, -8.72636681e-01,
```

```
 -8.34286569e-01, -1.00839129e+00, -8.69794241e-01,
 -1.04390785e+00,  9.93883735e-01],
[-1.89316122e+00, -2.07725206e-01, -7.06403230e-01,
  1.21981942e+00,  7.68619098e-01, -9.65997278e-02,
  2.74597703e+00,  5.31281609e-01,  1.97775023e+00,
  9.57938964e-01, -1.00615390e+00],
[ 1.16348363e+00,  1.95360611e-01,  1.90101811e+00,
 -2.16488630e+00, -9.83526008e-01, -8.93118594e-01,
 -1.03046540e+00, -1.00069293e+00, -9.11465624e-01,
 -1.04390785e+00,  9.93883735e-01],
[ 1.44602012e-01, -6.10811023e-01, -7.06403230e-01,
  5.36448709e-01, -1.46680584e-01,  1.67389382e-01,
 -1.23138321e-01, -1.53022705e-02, -1.47490277e-01,
  9.57938964e-01,  9.93883735e-01],
[ 8.75113043e-03, -2.07725206e-01, -7.06403230e-01,
  7.20150512e-01,  4.19933505e-01,  8.68325985e-01,
  3.42786394e-01,  6.46757076e-01,  4.91470922e-01,
  9.57938964e-01,  9.93883735e-01],
[-1.07805592e+00,  1.00153224e+00,  1.90101811e+00,
  2.64570041e-01, -3.82043360e-01, -8.47603230e-01,
 -3.43839501e-01, -5.07997598e-01, -5.08642259e-01,
  9.57938964e-01, -1.00615390e+00],
[-1.21390681e+00,  1.00153224e+00, -7.06403230e-01,
  9.77333036e-01,  3.93782085e-01, -1.23963048e-02,
  1.66699348e+00,  2.07950300e-01,  1.15821304e+00,
  9.57938964e-01, -1.00615390e+00],
[-5.34652397e-01, -1.82006847e+00, -7.06403230e-01,
  7.78935089e-01,  2.02005009e-01,  2.49317037e-01,
  2.20174627e-01,  2.46442122e-01,  2.27552166e-01,
  9.57938964e-01,  9.93883735e-01],
[-3.30876074e-01, -1.41698266e+00,  1.90101811e+00,
 -8.15596559e-01, -8.87637470e-01, -8.77188217e-01,
 -8.58808922e-01, -9.31407646e-01, -8.69794241e-01,
 -1.04390785e+00, -1.00615390e+00],
[ 7.66765714e-02,  1.00153224e+00, -7.06403230e-01,
  7.05454368e-01,  3.15327827e-01,  5.65648815e-01,
  5.38965221e-01,  4.31202870e-01,  5.47032766e-01,
  9.57938964e-01,  9.93883735e-01],
[ 7.66765714e-02,  5.98446428e-01, -7.06403230e-01,
  7.34846656e-01,  1.54444454e+00,  2.24744151e+00,
  5.63487574e-01,  1.94008231e+00,  1.25544627e+00,
  9.57938964e-01,  9.93883735e-01],
[-5.34652397e-01,  1.95360611e-01, -4.97809522e-01,
  1.98437392e-01, -8.44051771e-01, -7.04229834e-01,
 -4.90973621e-01, -8.23630543e-01, -7.30889633e-01,
 -1.04390785e+00,  9.93883735e-01],
[-1.07805592e+00, -6.10811023e-01,  7.53752722e-01,
  2.20836615e-02, -6.17406135e-01, -8.68085144e-01,
 -6.62630095e-01, -7.08155075e-01, -8.00341937e-01,
 -1.04390785e+00,  9.93883735e-01],
```

```
[ 7.55930981e-01,  1.40461806e+00, -4.97809522e-01,
 -1.02833564e-01, -9.22506029e-01, -3.35555388e-01,
 -5.89063035e-01, -8.00535449e-01, -8.00341937e-01,
 -1.04390785e+00,  9.93883735e-01],
[-1.14598137e+00, -1.01389684e+00, -8.06221077e-02,
  4.92360277e-01,  2.36873568e-01,  1.65113614e-01,
 -2.45750088e-01,  2.31045393e-01, -1.61380738e-01,
  9.57938964e-01, -1.00615390e+00],
[-5.91743105e-02, -2.22315429e+00,  1.90101811e+00,
 -8.59684992e-01, -8.35334631e-01, -8.79463985e-01,
 -8.83331276e-01, -8.92915823e-01, -8.69794241e-01,
 -1.04390785e+00, -1.00615390e+00],
[ 1.63896171e+00,  2.41233260e+00,  3.36565307e-01,
 -2.16488630e+00, -9.31223169e-01, -8.88567058e-01,
 -1.00594304e+00, -9.69899468e-01, -8.97575163e-01,
 -1.04390785e+00, -1.00615390e+00],
[-5.34652397e-01, -1.01389684e+00, -8.06221077e-02,
  3.82139195e-01,  1.23550750e-01,  1.03673283e+00,
 -1.96705381e-01,  4.61996328e-01, -5.02570508e-02,
  9.57938964e-01,  9.93883735e-01],
[-6.02577838e-01, -1.82006847e+00, -4.97809522e-01,
  4.77664132e-01, -5.04083318e-01, -4.69825711e-01,
 -3.43839501e-01, -5.15695963e-01, -4.94751798e-01,
  9.57938964e-01, -1.00615390e+00],
[-8.74279602e-01,  1.40461806e+00,  5.45159014e-01,
  4.41278778e-02, -3.12306241e-01, -1.26184714e-01,
 -5.15495975e-01, -2.38554841e-01, -5.36423181e-01,
 -1.04390785e+00,  9.93883735e-01],
[ 1.77481259e+00,  1.95360611e-01,  5.45159014e-01,
 -2.16488630e+00, -1.14043453e+00, -8.90842826e-01,
 -1.10403246e+00, -1.10847003e+00, -9.25356084e-01,
 -1.04390785e+00,  9.93883735e-01],
[-1.55353401e+00, -1.01389684e+00, -7.06403230e-01,
  1.08020605e+00,  7.07599119e-01,  1.94698600e-01,
  1.54438171e+00,  4.85091422e-01,  1.25544627e+00,
  9.57938964e-01, -1.00615390e+00],
[ 4.16303776e-01,  1.95360611e-01, -7.06403230e-01,
  6.83410151e-01,  1.14833610e-01,  3.76705954e-02,
  2.44696980e-01,  9.24748324e-02,  1.85880784e-01,
  9.57938964e-01,  9.93883735e-01],
[ 1.23140907e+00,  5.98446428e-01, -7.06403230e-01,
  5.43796781e-01,  6.02993441e-01, -1.05702801e-01,
  4.85181532e-02,  3.77314319e-01,  1.44209401e-01,
  9.57938964e-01, -1.00615390e+00],
[ 3.48378335e-01, -6.10811023e-01,  1.90101811e+00,
 -7.78856198e-01, -5.65103296e-01, -8.58982071e-01,
 -8.34286569e-01, -6.61964888e-01, -8.42013319e-01,
 -1.04390785e+00,  9.93883735e-01],
[-1.96108666e+00,  1.95360611e-01, -2.89215815e-01,
  8.15675449e-01,  2.36873568e-01, -4.19812913e-02,
```

```
  7.35144048e-01,  1.15569926e-01,  5.33142305e-01,
  9.57938964e-01, -1.00615390e+00],
[-8.06354161e-01, -1.01389684e+00, -7.06403230e-01,
  8.45067738e-01,  1.58419310e-01,  7.40828864e-02,
  5.88009928e-01,  1.38665019e-01,  4.35909079e-01,
  9.57938964e-01, -1.00615390e+00],
[-1.75731033e+00,  1.95360611e-01, -7.06403230e-01,
  1.10225026e+00,  2.52730206e+00,  2.35582497e+00,
  2.10839584e+00,  2.41064484e+00,  2.87889388e+00,
  9.57938964e-01, -1.00615390e+00],
[ 8.23856422e-01,  1.95360611e-01, -7.06403230e-01,
  2.64570041e-01, -9.39940309e-01, -5.22168379e-01,
 -4.17406561e-01, -8.39027272e-01, -6.89218250e-01,
  9.57938964e-01,  9.93883735e-01],
[ 1.36725995e+00,  5.98446428e-01, -7.06403230e-01,
  5.36448709e-01,  4.54802064e-01, -3.15073474e-01,
  4.85181532e-02,  2.46442122e-01,  8.86475577e-02,
  9.57938964e-01, -1.00615390e+00],
[ 7.66765714e-02, -6.10811023e-01, -7.06403230e-01,
  8.59763882e-01,  2.29411857e+00,  2.35582497e+00,
  1.10297935e+00,  2.41064484e+00,  1.97775023e+00,
  9.57938964e-01,  9.93883735e-01],
[-6.70503279e-01,  1.40461806e+00, -7.06403230e-01,
  8.81808098e-01,  2.54307848e-01,  1.17555469e+00,
  1.34820288e+00,  6.08265254e-01,  1.19988443e+00,
  9.57938964e-01,  9.93883735e-01],
[ 1.02763274e+00,  1.00153224e+00, -7.06403230e-01,
  6.31973647e-01,  2.52730206e+00,  2.35582497e+00,
  4.85181532e-02,  2.41064484e+00,  9.77637052e-01,
  9.57938964e-01,  9.93883735e-01],
[-2.30071386e+00,  1.00153224e+00, -7.06403230e-01,
  1.22716749e+00,  2.21566431e+00,  7.65916416e-01,
  2.85632762e+00,  1.80921012e+00,  2.87889388e+00,
  9.57938964e-01, -1.00615390e+00],
[-6.02577838e-01,  1.00153224e+00, -4.97809522e-01,
  1.83741248e-01, -8.61486051e-01, -7.22435980e-01,
 -6.87152449e-01, -8.39027272e-01, -8.28122859e-01,
 -1.04390785e+00, -1.00615390e+00],
[-3.30876074e-01,  1.00153224e+00,  1.90101811e+00,
 -9.91950290e-01, -9.57374589e-01, -8.74912449e-01,
 -8.09764216e-01, -9.77597833e-01, -8.55903780e-01,
 -1.04390785e+00, -1.00615390e+00],
[-2.02901210e+00, -1.01389684e+00, -4.97809522e-01,
  9.84681108e-01,  1.01269901e+00,  7.02194907e-01,
  1.10297935e+00,  9.39294927e-01,  1.21377489e+00,
  9.57938964e-01, -1.00615390e+00],
[-3.30876074e-01,  1.00153224e+00, -7.06403230e-01,
  7.93631233e-01, -5.07920463e-02, -2.30870051e-01,
  8.57755815e-01, -1.46174467e-01,  4.49799540e-01,
  9.57938964e-01, -1.00615390e+00],
```

```
[ 1.44602012e-01,  1.40461806e+00,  5.45159014e-01,
 -6.53938973e-01, -7.56880373e-01, -7.22435980e-01,
 -7.36197155e-01, -7.85138720e-01, -8.28122859e-01,
 -1.04390785e+00,  9.93883735e-01],
[-4.66726956e-01,  1.95360611e-01, -7.06403230e-01,
  8.59763882e-01,  8.38356216e-01,  2.31110892e-01,
  9.06800522e-01,  6.00566889e-01,  9.08184748e-01,
  9.57938964e-01, -1.00615390e+00],
[ 6.20080099e-01, -6.10811023e-01, -7.06403230e-01,
  5.14404493e-01, -5.12800457e-01, -4.94859161e-01,
 -1.72183027e-01, -5.31092692e-01, -3.83628111e-01,
  9.57938964e-01, -1.00615390e+00],
[-1.95025192e-01,  1.00153224e+00, -8.06221077e-02,
 -7.30862694e-03, -2.51286262e-01,  1.25830434e-03,
 -5.64540682e-01, -1.46174467e-01, -5.64204102e-01,
 -1.04390785e+00,  9.93883735e-01],
[ 1.57103627e+00,  2.21078970e+00,  1.90101811e+00,
 -2.16488630e+00, -1.12300025e+00, -8.97670131e-01,
 -1.07951010e+00, -1.09307330e+00, -9.25356084e-01,
 -1.04390785e+00,  9.93883735e-01],
[-2.62950633e-01,  5.98446428e-01, -7.06403230e-01,
  7.49542800e-01,  5.38136316e-02,  2.28835123e-01,
  5.88009928e-01,  1.46363384e-01,  4.49799540e-01,
  9.57938964e-01,  9.93883735e-01],
[ 1.43518539e+00,  1.00153224e+00, -7.06403230e-01,
  4.85012204e-01, -5.95091862e-02,  5.36009727e-02,
 -5.26553600e-04,  9.44584601e-05, -5.02570508e-02,
  9.57938964e-01,  9.93883735e-01],
[-1.68938489e+00, -2.07725206e-01, -7.06403230e-01,
  1.09490219e+00,  2.52730206e+00,  2.29295687e+00,
  1.88769466e+00,  2.41064484e+00,  2.81117788e+00,
  9.57938964e-01, -1.00615390e+00],
[-1.75731033e+00,  1.00153224e+00, -2.89215815e-01,
  8.59763882e-01, -1.11812025e-01, -7.36090589e-01,
  1.96126172e+00, -4.31013953e-01, -9.25356084e-01,
 -1.04390785e+00, -1.00615390e+00],
[-4.66726956e-01,  5.98446428e-01, -4.97809522e-01,
  5.51144853e-01, -3.90760500e-01, -4.47068029e-01,
  4.85181532e-02, -4.38712318e-01, -2.03052120e-01,
  9.57938964e-01, -1.00615390e+00],
[ 8.23856422e-01,  1.95360611e-01, -7.06403230e-01,
  3.67443051e-01, -8.52768911e-01, -5.15341075e-01,
 -2.94794794e-01, -7.69741991e-01, -5.78094563e-01,
  9.57938964e-01, -1.00615390e+00],
[-5.91743105e-02,  1.40461806e+00, -7.06403230e-01,
  8.96504243e-01,  7.86053377e-01,  1.68305100e+00,
  1.27463582e+00,  1.17794423e+00,  1.46380318e+00,
  9.57938964e-01,  9.93883735e-01],
[-8.74279602e-01, -6.10811023e-01, -7.06403230e-01,
  8.45067738e-01,  7.59901958e-01,  1.04356014e+00,
```

```
         6.61576988e-01,  9.46993292e-01,  8.66513365e-01,
         9.57938964e-01,  9.93883735e-01],
       [-1.96108666e+00,  5.98446428e-01, -7.06403230e-01,
         1.10225026e+00,  8.20921937e-01,  1.14596970e+00,
         2.28005231e+00,  1.03167530e+00,  2.07498346e+00,
         9.57938964e-01, -1.00615390e+00],
       [ 1.23140907e+00, -2.07725206e-01, -7.06403230e-01,
         5.43796781e-01, -1.37963445e-01, -1.23908946e-01,
        -4.95712604e-02, -1.30777738e-01, -1.33599816e-01,
         9.57938964e-01, -1.00615390e+00],
       [-2.62950633e-01,  1.95360611e-01, -7.06403230e-01,
         8.00979305e-01,  4.63519204e-01,  5.80984072e-03,
         7.10621695e-01,  2.69537216e-01,  5.88704148e-01,
         9.57938964e-01, -1.00615390e+00],
       [ 2.80452894e-01, -6.10811023e-01, -7.06403230e-01,
         3.96835339e-01, -7.30728953e-01, -5.63132207e-01,
        -3.43839501e-01, -6.92758346e-01, -5.78094563e-01,
         9.57938964e-01,  9.93883735e-01],
       [-1.01013048e+00, -1.01389684e+00, -7.06403230e-01,
         9.62636892e-01,  6.98881979e-01,  1.40085574e+00,
         1.02941229e+00,  1.03167530e+00,  1.19988443e+00,
         9.57938964e-01,  9.93883735e-01],
       [ 7.66765714e-02, -6.10811023e-01, -7.06403230e-01,
         1.61697032e-01, -7.83031792e-01, -7.17884444e-01,
        -5.64540682e-01, -8.00535449e-01, -7.86451476e-01,
        -1.04390785e+00, -1.00615390e+00],
       [-5.34652397e-01, -2.07725206e-01, -7.06403230e-01,
         8.74460026e-01,  5.85559161e-01, -8.29451187e-02,
         8.57755815e-01,  3.69615954e-01,  7.41499218e-01,
         9.57938964e-01, -1.00615390e+00]])
```

6. Model Training

6.1 Simple Linear Regression model

```
from sklearn.linear_model import LinearRegression

regression = LinearRegression()
regression
```

```
LinearRegression()
```

6.1.1 Training the model

```
regression.fit(X_train, y_train)
```

```
LinearRegression()
```

Printing the coefficient

```
print(regression.coef_)
```

```
[-0.73817299 -0.48357275  0.67129951  2.39226728  1.36782176
0.79758278
  0.27688161 -1.85766326  0.12602901 -0.32514707  0.05742277]
```

Printing the intercept

```
print(regression.intercept_)
```

```
31.987730061349694
```

6.1.2 Prediction for the test data

```
reg_pred = regression.predict(X_test)
reg_pred
```

```
array([33.06370081, 34.18627896, 33.66953545, 32.98675299,
36.69985174,
       33.45664154, 35.18228775, 27.34531879, 30.6812043 ,
30.20974793,
       28.86168228, 33.0833322 , 33.72544433, 33.15174196,
34.17494863,
       32.22045997, 37.33415353, 26.23993544, 32.58730468,
33.31519884,
       30.97663452, 28.01679329, 35.0517456 , 28.77656317,
36.5572493 ,
       26.92920597, 32.63969905, 33.29825536, 33.4248167 ,
34.64403936,
       34.50433665, 31.99293704, 32.70544939, 33.28634695,
32.41073544,
       33.52851834, 30.26049893, 34.01654977, 32.06817667,
24.36759725,
       33.55519102, 33.48815856, 32.52688671, 25.53303772,
36.1733408 ,
       32.59875559, 31.3179629 , 31.26987485, 35.07536278,
34.4873544 ,
       36.78817794, 30.84347817, 31.06652713, 34.64807141,
33.86436628,
       32.52243011, 36.72829967, 31.84469026, 30.40908703,
36.28670097,
       33.11489793, 29.9386535 , 34.03204881, 32.00697098,
31.67368275,
       25.61563035, 33.16066526, 30.8303706 , 36.79559519,
35.43195965,
       32.56968083, 31.01865552, 33.53498064, 34.62641719,
35.95256874,
       31.56148045, 33.56895128, 31.96700812, 35.39093576,
32.00884133,
       34.11044326])
```

6.2 Ridge Regression model

```
from sklearn.linear_model import Ridge
```

```
ridge = Ridge()
ridge
```

```
Ridge()
```

6.2.1 Training the model

```
ridge.fit(X_train, y_train)
```

```
Ridge()
```

Printing the coefficient

```
print(ridge.coef_)
```

```
[-0.79486244 -0.48580044  0.60384955  2.25491266  0.70792101
0.45714856
  0.30245189 -0.80895085  0.0470057  -0.30606154  0.04787086]
```

Printing the intercept

```
print(ridge.intercept_)
```

```
31.987730061349694
```

6.2.2 Prediction for the test data

```
ridge_pred = ridge.predict(X_test)
ridge_pred
```

```
array([33.06299963, 34.13621959, 33.63950325, 33.00708305,
36.6803758 ,
       33.41667013, 35.15916138, 27.38789324, 30.7239846 ,
30.1472425 ,
       28.87351895, 33.20700385, 33.69237923, 33.19536392,
34.29715828,
       32.16308518, 37.24325834, 26.3124305 , 32.4919839 ,
33.2839515 ,
       30.88931953, 28.02822622, 35.02554617, 28.7648236 ,
36.569789  ,
       26.95184638, 32.64788315, 33.3312095 , 33.38635294,
34.56546585,
       34.51606178, 31.94271315, 32.68051994, 33.4016699 ,
32.39569905,
       33.51430622, 30.1644501 , 34.08572268, 32.01640168,
24.47872947,
       33.59846986, 33.50431205, 32.5291528 , 25.60143352,
36.10661237,
       32.53723138, 31.27127713, 31.21715899, 35.07731038,
34.49103816,
       36.69621859, 30.77835934, 31.04936127, 34.66752322,
33.84409684,
       32.41386041, 36.78349374, 31.85705488, 30.37465945,
36.34020861,
```

```
        33.04316264, 29.91959998, 33.96485127, 31.95387788,
31.70178279,
        25.60722886, 33.14729069, 30.76167673, 36.72280263,
35.41609203,
        32.54783841, 30.96567625, 33.52906086, 34.70254462,
36.03095557,
        31.49834936, 33.49540666, 31.93870375, 35.45001391,
31.99027625,
        34.08942896])
```

6.3 Lasso Regression model

```
from sklearn.linear_model import Lasso

lasso = Lasso()
lasso
```

```
Lasso()
```

6.3.1 Training the model

```
lasso.fit(X_train, y_train)
```

```
Lasso()
```

Printing the coefficient

```
print(lasso.coef_)
```

```
[-0.62775073 -0.         -0.          1.23221554  0.          0.
  0.          0.          0.          0.         -0.        ]
```

Printing the intercept

```
print(lasso.intercept_)
```

```
31.987730061349694
```

6.3.2 Prediction for the test data

```
lasso_pred = lasso.predict(X_test)
lasso_pred
```

```
array([32.79981371, 33.36203229, 33.56354731, 32.68031985,
34.26042301,
        31.631795  , 34.10123447, 28.67689451, 30.93786487,
29.98946999,
        31.01145915, 32.40373776, 32.80065896, 32.29508486,
33.40088173,
        32.48059138, 34.63923539, 28.20598362, 32.30360779,
33.13830388,
        31.60525931, 29.68193387, 33.68388642, 30.20267122,
34.67924384,
        28.58974582, 32.55797648, 32.86961718, 32.99048777,
33.9540459 ,
```

```
       33.28317442, 31.19044701, 32.80886812, 32.84508576,
32.56787613,
       32.69169229, 31.38648072, 33.31381469, 30.96555957,
28.29126411,
       32.79423635, 32.95458391, 32.59093478, 28.20598362,
34.29400885,
       32.56850367, 31.88478696, 30.8093166 , 34.22389161,
33.53522507,
       34.44909281, 31.79656091, 31.79045206, 32.9990107 ,
33.49521663,
       32.1213606 , 34.94413972, 32.59240756, 30.9731412 ,
34.47478325,
       33.1733625 , 31.09116228, 33.34013266, 32.23233153,
32.10115146,
       28.33390435, 33.0763958 , 31.68443096, 34.39739816,
34.15029733,
       32.9598475 , 31.92332263, 33.12956324, 33.57786532,
34.57701355,
       31.88478696, 33.13977666, 32.30066222, 33.80801635,
32.13884188,
       33.40088173])
```

6.4 ElasticNet Regression model

```
from sklearn.linear_model import ElasticNet

elastic = ElasticNet()
elastic
```

```
ElasticNet()
```

6.4.1 Training the model

```
elastic.fit(X_train, y_train)
```

```
ElasticNet()
```

Printing the coefficient

```
print(elastic.coef_)
```

```
[-0.69537798 -0.10593637 -0.          0.81909116  0.14073666  0.
  0.22627668  0.04484606  0.13919408  0.05340521 -0.        ]
```

Printing the intercept

```
print(elastic.intercept_)
```

```
31.987730061349694
```

6.4.2 Prediction for the test data

```
elastic_pred = elastic.predict(X_test)
elastic_pred
```

```
array([32.68761983, 33.27298145, 33.36365648, 32.6379774 ,
34.76451911,
       31.5805353 , 34.21914232, 28.91901237, 30.72127885,
30.03555877,
       30.35407868, 32.3037364 , 32.49149551, 32.13351757,
33.91653385,
       32.23335956, 35.70766367, 28.45258368, 32.10513619,
32.91156715,
       30.97951418, 29.56413604, 33.69879535, 30.00131768,
35.40514104,
       28.78564781, 32.37272075, 32.87875718, 32.67400158,
34.1806027 ,
       33.36248055, 31.1620342 , 32.71911747, 33.13069877,
32.07705096,
       32.38444168, 30.76286662, 33.31210321, 31.02592752,
28.23637706,
       32.81769139, 32.80123094, 32.18131651, 28.31504697,
34.75692594,
       32.39003109, 31.69745211, 30.70127806, 34.32907808,
33.62144886,
       35.48462934, 31.30171296, 31.56250672, 33.71034604,
33.61377572,
       32.34669296, 35.97782458, 31.96577533, 30.76252086,
34.96694781,
       33.05592649, 30.72340588, 33.52349211, 31.90540321,
31.70731425,
       28.25193426, 32.98208546, 31.31657768, 35.414713  ,
34.03201953,
       32.6595273 , 31.44456266, 33.32118187, 33.82346635,
35.20858015,
       31.59492453, 33.17718143, 31.9414416 , 34.1817811 ,
31.69249853,
       33.54523053])
```

6.5 Assumptions Of the Regression models

- To check whether the Linear model is good or bad.
- 1st to do create a scatter plot between the real test data and predicted test data of the model.
- 2nd get the residuals or errors and then create a distribution plot of those residuals.
- 3rd create an Uniform distribution by using a scatter plot between the predictions and the residuals.

Scatter plots

```
fig, axs = plt.subplots(2, 2, figsize=(20, 10))
plt.suptitle('Scatterplots of different Regression models',
fontsize=20, fontweight='bold', alpha=0.8, y=1.)

plt.subplot(2, 2, 1)
```

```
plt.scatter(y_test, reg_pred)
plt.title("Simple Linear Regression model", fontsize=15,
fontweight='bold')
plt.xlabel("Original Test Data points")
plt.ylabel("Predicted Test Data points")

plt.subplot(2, 2, 2)
plt.scatter(y_test, ridge_pred)
plt.title("Ridge Regression model", fontsize=15, fontweight='bold')
plt.xlabel("Original Test Data points")
plt.ylabel("Predicted Test Data points")

plt.subplot(2, 2, 3)
plt.scatter(y_test, lasso_pred)
plt.title("Lasso Regression model", fontsize=15, fontweight='bold')
plt.xlabel("Original Test Data points")
plt.ylabel("Predicted Test Data points")

plt.subplot(2, 2, 4)
plt.scatter(y_test, elastic_pred)
plt.title("ElasticNet Regression model", fontsize=15,
fontweight='bold')
plt.xlabel("Original Test Data points")
plt.ylabel("Predicted Test Data points")


plt.tight_layout()
plt.show()
```

Observations:

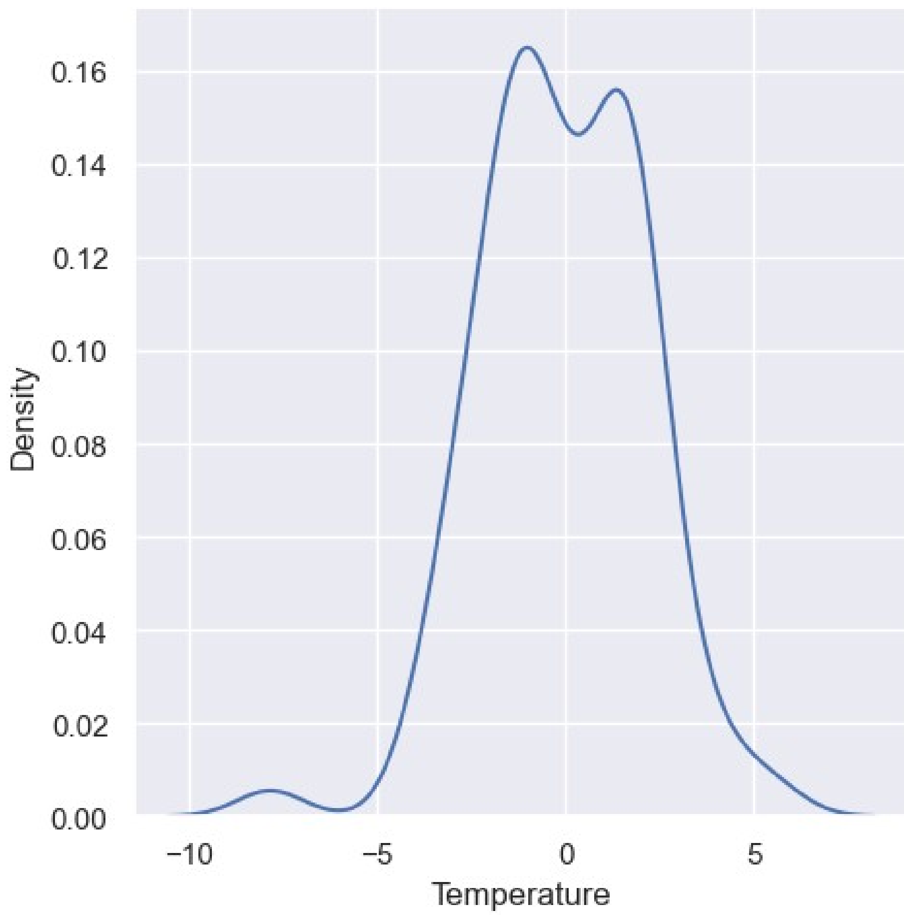
- Here we see not much difference in all the models.

```
# Calculating the residuals, i.e. the errors for different models

simple_residuals = y_test - reg_pred
ridge_residuals = y_test - ridge_pred
lasso_residuals = y_test - lasso_pred
elastic_residuals = y_test - elastic_pred
```

Distribution plots of residuals

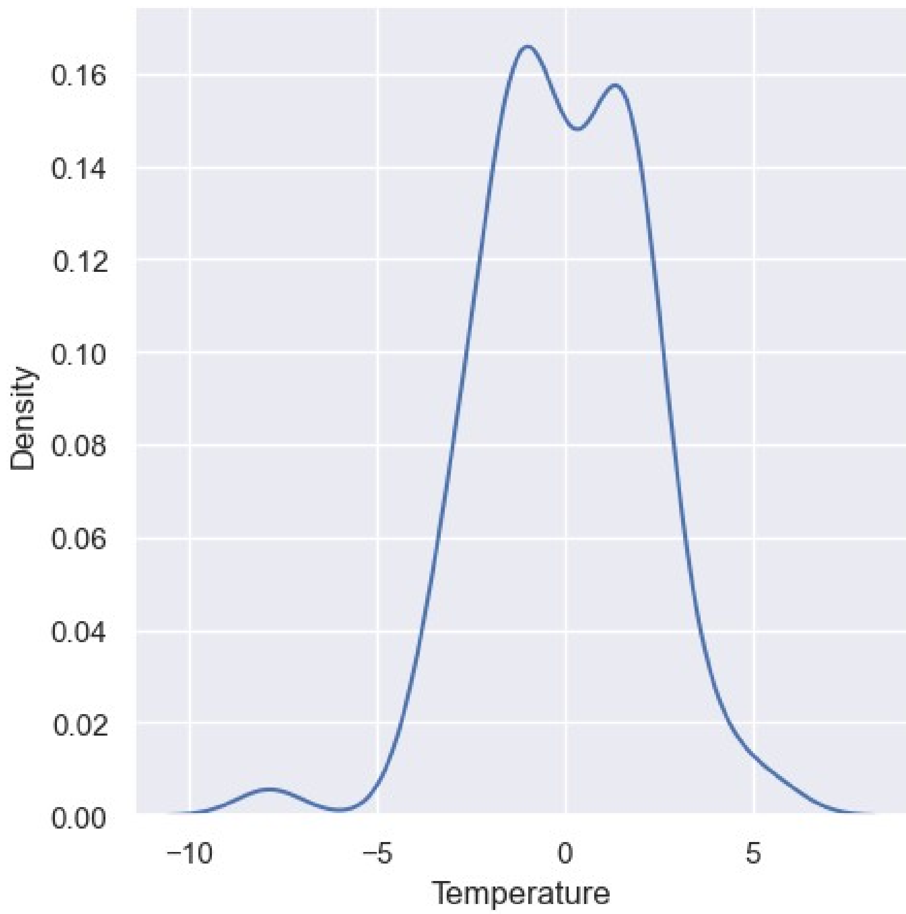
```
sns.displot(simple_residuals, kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x23ed3eccaf0>
```



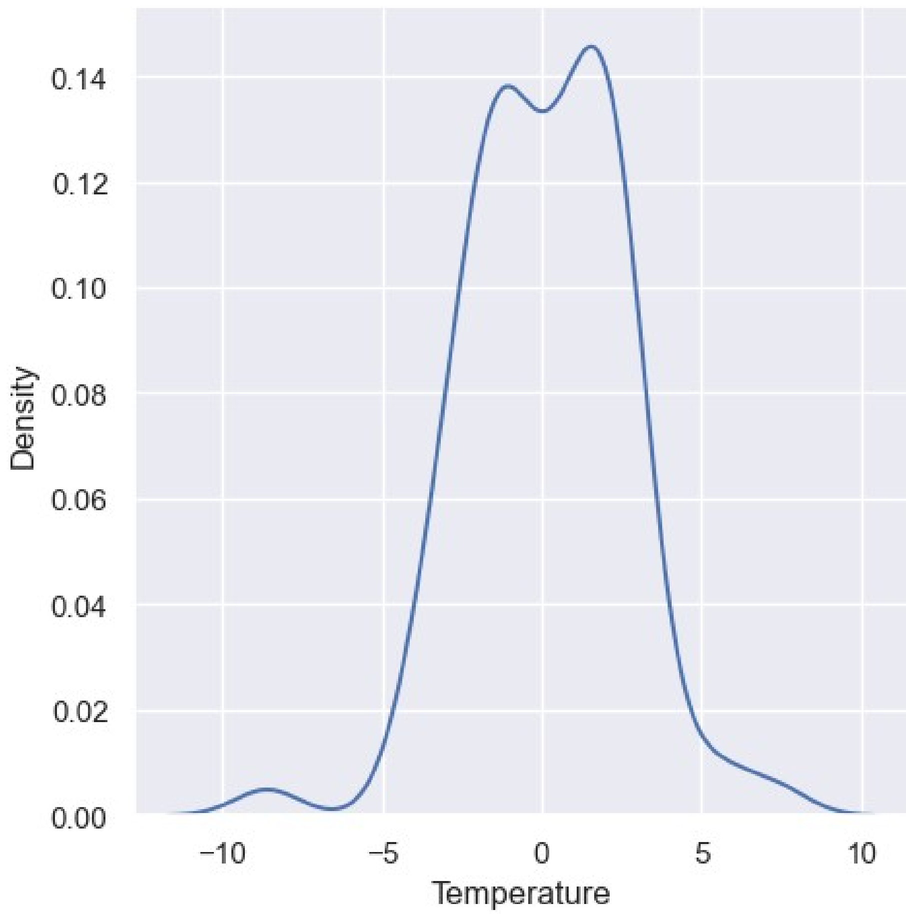

```
sns.displot(ridge_residuals, kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x23ed2cd9b20>
```



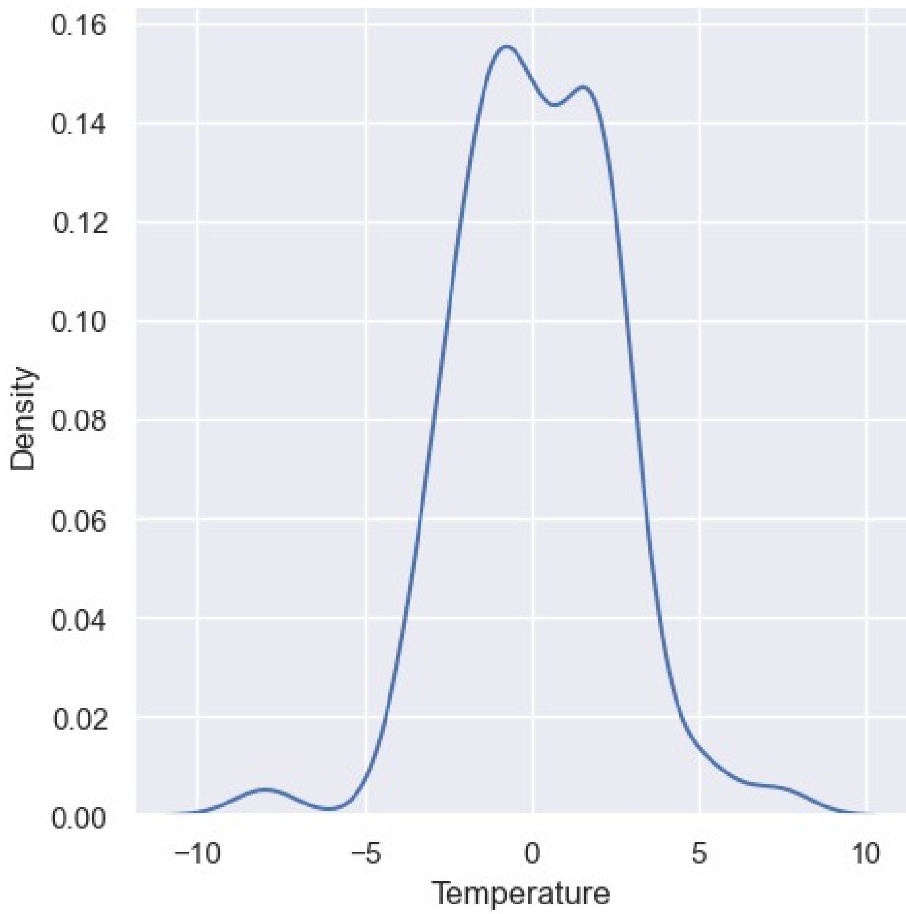

```
sns.displot(lasso_residuals, kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x23ed3b05070>
```




```
sns.displot(elastic_residuals, kind="kde")
```

```
<seaborn.axisgrid.FacetGrid at 0x23ed3aeda00>
```



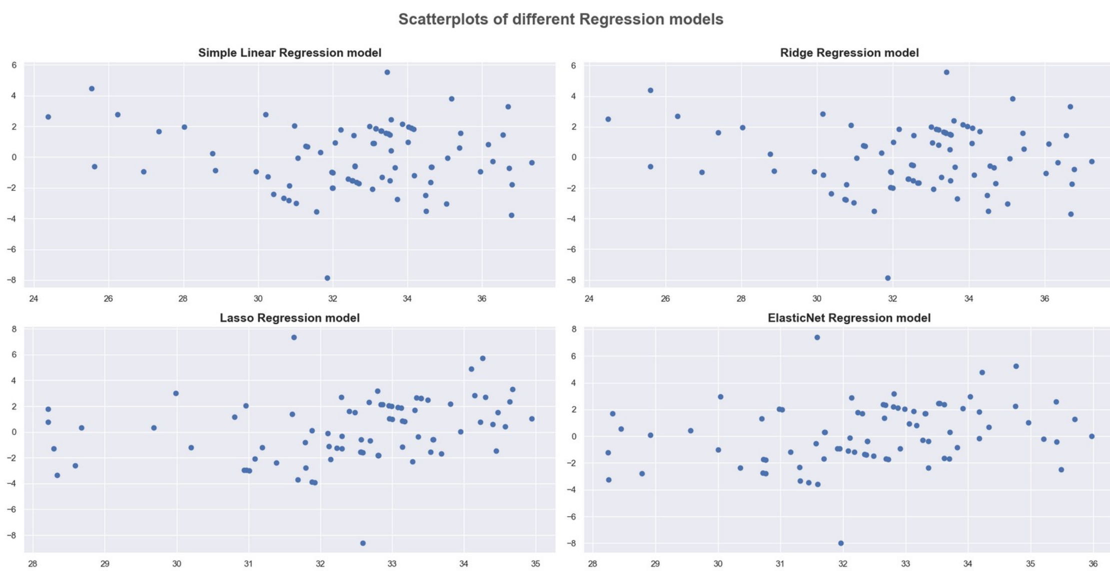
Observations:

- All the plots are slightly left skewed.

Scatter plot for uniform distribution

```
fig, axs = plt.subplots(2, 2, figsize=(20, 10))
plt.suptitle('Scatterplots of different Regression models',
fontsize=20, fontweight='bold', alpha=0.8, y=1.)

plt.subplot(2, 2, 1)
plt.scatter(reg_pred, simple_residuals)
plt.title("Simple Linear Regression model", fontsize=15,
fontweight='bold')


plt.subplot(2, 2, 2)
plt.scatter(ridge_pred, ridge_residuals)
plt.title("Ridge Regression model", fontsize=15, fontweight='bold')


plt.subplot(2, 2, 3)
```

```
plt.scatter(lasso_pred, lasso_residuals)
plt.title("Lasso Regression model", fontsize=15, fontweight='bold')


plt.subplot(2, 2, 4)
plt.scatter(elastic_pred, elastic_residuals)
plt.title("ElasticNet Regression model", fontsize=15,
fontweight='bold')



plt.tight_layout()
plt.show()
```



Observations:

- All the models are showing a negative correlation.

7. Choosing the best model

7.1 Performance metrics

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

# Creting function for all regression models for their errors

def errors(mla, pred_data):
    print(f"For the {mla} algorithm:")
    print("The Mean Squared Error is: ", mean_squared_error(y_test,
pred_data))
    print("The Mean Absolute Error is: ", mean_absolute_error(y_test,
pred_data))
    print("The Root Mean Squared Error is: ",
```

```
np.sqrt(mean_squared_error(y_test, pred_data)))
    print("\n")


errors("Simple Linear Regression", reg_pred)
errors("Ridge Regression", ridge_pred)
errors("Lasso Regression", lasso_pred)
errors("ElasticNet Regression", elastic_pred)
```

```
For the Simple Linear Regression algorithm:
The Mean Squared Error is:  4.6041665086958705
The Mean Absolute Error is:  1.7519691712082832
The Root Mean Squared Error is:  2.145732161453491


For the Ridge Regression algorithm:
The Mean Squared Error is:  4.55373665402209
The Mean Absolute Error is:  1.7380920201509793
The Root Mean Squared Error is:  2.133948606227922


For the Lasso Regression algorithm:
The Mean Squared Error is:  5.999248918641266
The Mean Absolute Error is:  1.974742788359419
The Root Mean Squared Error is:  2.4493364241445614


For the ElasticNet Regression algorithm:
The Mean Squared Error is:  5.3088550720656915
The Mean Absolute Error is:  1.825156200520228
The Root Mean Squared Error is:  2.304095282766251
```

7.2 R Squared and Adjusted R Square

```
from sklearn.metrics import r2_score
```

```
# Creting function for all regreesion models for their r values

def r_value(mla, pred_data):
    print(f"For the {mla} algorithm:")
    score = r2_score(y_test, pred_data)
    print(f"The R Square value of {mla} is: {score}")
    adj = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-
1)
    print(f"The Adjusted R Square value of {mla} is: {adj}")
    print("\n")


r_value("Simple Linear Regression", reg_pred)
```

```
r_value("Ridge Regression", ridge_pred)
r_value("Lasso Regression", lasso_pred)
r_value("ElasticNet Regression", elastic_pred)
```

```
For the Simple Linear Regression algorithm:
The R Square value of Simple Linear Regression is: 0.5713362216041777
The Adjusted R Square value of Simple Linear Regression is:
0.5029985178019452


For the Ridge Regression algorithm:
The R Square value of Ridge Regression is: 0.5760314149703573
The Adjusted R Square value of Ridge Regression is: 0.5084422202554866


For the Lasso Regression algorithm:
The R Square value of Lasso Regression is: 0.44144923860926155
The Adjusted R Square value of Lasso Regression is:
0.35240491432957866


For the ElasticNet Regression algorithm:
The R Square value of ElasticNet Regression is: 0.505727286393884
The Adjusted R Square value of ElasticNet Regression is:
0.42693018712334374
```

Conclusion:

- From the above results we can say that either Simple Linear or Ridge Regression is the best model for this dataset.