Algerian Forest Fire

EDA on Forest Fire in Algeria dataset

The dataset includes 244 instances that regroup a data of two regions of Algeria, namely the Bejaia region located in the northeast of Algeria and the Sidi Bel-abbes region located in the northwest of Algeria.

Each region has 122 instances each. The data is collected from period of June 2012 to September 2012.

The dataset includes 11 attribues and 1 output attribue (class). The 244 instances have been classified into 138 fire classes and 106 not fire classes.

Problem Statement

Based on the given input, a prediction need to be made if there will be fire or not in a particular area

✓ 0s completed at 3:57 PM

Data description

Attribute name	Description	Data range
Day	Date in particular month	1-31
month	month of the year 2012	6-9
year	Year	2012
Temp	Max temperature in a day in Celsius degrees.	It ranges from 22 to 42
RH	Relative Humidity in %	21 to 90
Ws	Wind speed in km/h	6 to 29
Rain	total day in mm	0 to 16.8
Fine Fuel Moisture Code (FFMC)	index from the FWI system	28.6 to 92.5
Duff Moisture Code (DMC)	index from the FWI system	1.1 to 65.9
Drought Code (DC)	index from the FWI system	7 to 220.4
Initial Spread Index (ISI)	index from the FWI system	0 to 18.5
Buildup Index (BUI)	index from the FWI system	1.1 to 68
Fire Weather Index (FWI)	from the FWI system	0 to 31.1
Classes	two classes, that give info about fire	Fire and Not fire

1. Exploring the data

Exploreing data require data to be collected from the the source and load into the python. Data is collected from Algerian Forest Fires Dataset present in UCI Machine Learning repository.

1.1 Importing the required packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
plt.style.use('ggplot')

warnings.filterwarnings("ignore")

%matplotlib inline
```

1.2 Import the dataset and data cleaning

import data in pandas dataframe

```
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DATA Set/Algeria
```

Observations:

1. From index 0 to 122 we had Bejai region
2. From index 123 to 124 we had irrelevant rows
3. From index 125 onwards we had SidiBel region.

```
# Creating the clean dataset
data.drop(index = [122,123,124],inplace = True) # drop the irrelevant rows
data.reset_index(drop = True, inplace = True) # Creating new index
data['region'] = 0
# separating the dataset based on the location
for i in range(len(data)):
    if i < 123:
        data['region'][i] =  1
    else:
        data['region'][i] = 0
```

Cleaning the columns names

```
data.columns
```

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

Observations:

1. Column names has extra spaces at the end

```
# correcting the columns names
for col in data.columns:
    data.rename(columns = {col: col.strip()}, inplace = True)
```

Looking at irregularities in dataset

```
data.iloc[164,:]
```

```
2
```

```
day                14
month              07
year             2012
Temperature        37
RH                 37
Ws                 18
Rain              0.2
FFMC             88.9
DMC              12.9
DC              14.6 9
ISI              12.5
BUI              10.4
FWI             fire
Classes            NaN
region               0
Name: 164, dtype: object
```

Observation:

1. Data at index 164 has

column	Value
DC	14.6 9
ISI	12.5
BUI	10.4
FWI	fire
Class	nan

This means entry has been shifted to left thus creating the problem.

2. The data need to be corrected as

column	Value
DC	14.6
ISI	9
BUI	12.5
FWI	10.4
Class	fire

```
1 # correcting the entries
2 data.at[164, 'DC'] = 14.6
3 data.at[164, 'ISI'] = 9
4 data.at[164, 'BUI'] = 12.5
5 data.at[164,'FWI'] = 10.4
6 data.at[164,'Classes'] = 'fire'
```

```
1 data.head()
2
```

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI

	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

Observation:

1. Year column has ony one type of entry and hence not suitable for any analysis
2. Day column has no impact on the fire.

```
# Dropping the year column
data.drop(columns = ['year', 'day'], inplace = True)
```

Cleaning the "classes" column entries

```
data.Classes.unique()
```

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

Observations

1. The entries have extra spaces that need to be removed

```
# removing the extra spaces from the entries in column Classes
data['Classes'] = [i.strip() for i in data['Classes']]

```

Checking the data types of each columns

```
data.dtypes
```

```
month          object
Temperature    object
RH             object
Ws             object
Rain           object
FFMC           object
DMC            object
DC             object
ISI            object
BUI            object
```

```
BUI             object
FWI             object
Classes         object
region           int64
dtype: object
```

Observations:

1. Columns 'day, 'Temperature', ' RH', ' Ws', 'Rain ','FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI' are having categorical data and they need to be numerical.
2. Columns 'region' need to be categorical.

```
# converting the data types of columns
cols = ['month', 'Temperature', 'RH', 'Ws', 'Rain','FFMC', 'DMC', 'DC', 'ISI
data[cols] = data[cols].apply(pd.to_numeric)

data['region'] = data['region'].apply(str)

```

Shape of dataset

```
data.shape
```

```
(243, 13)
```

Summary of Dataset

```
data.describe()
```

	month	Temperature	RH	Ws	Rain	FI
count	243.000000	243.000000	243.000000	243.000000	243.000000	243.000(
mean	7.506173	32.172840	61.901235	15.518519	0.760905	77.9732
std	1.111065	3.641327	14.903495	2.806918	2.003528	14.3046
min	6.000000	22.000000	21.000000	6.000000	0.000000	28.600(
25%	7.000000	30.000000	52.000000	14.000000	0.000000	72.450(
50%	8.000000	32.000000	63.000000	15.000000	0.000000	83.700(
75%	8.000000	35.000000	73.500000	17.000000	0.500000	88.300(
max	9.000000	42.000000	90.000000	29.000000	16.800000	96.000(

Checking for null values

```
1 data.isna().sum()
```

```
month          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
```

1.3 Exploratory Data Analysis

1.3.1 Feature Information

[ ] ↳ *6 cells hidden*

1.3.2. Univariate analysis

- The term univariate analysis refers to the analysis of one variable prefix “uni” means “one.” The purpose of univariate analysis is to understand the distribution of values for a single variable.

[ ] ↳ *7 cells hidden*

1.3.3 Multivariate analysis

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

[ ] ↳ *3 cells hidden*

1.3.4 Relation between feature and label column

```
1 feature = data.drop(columns = 'Classes')
2 feature.columns
```

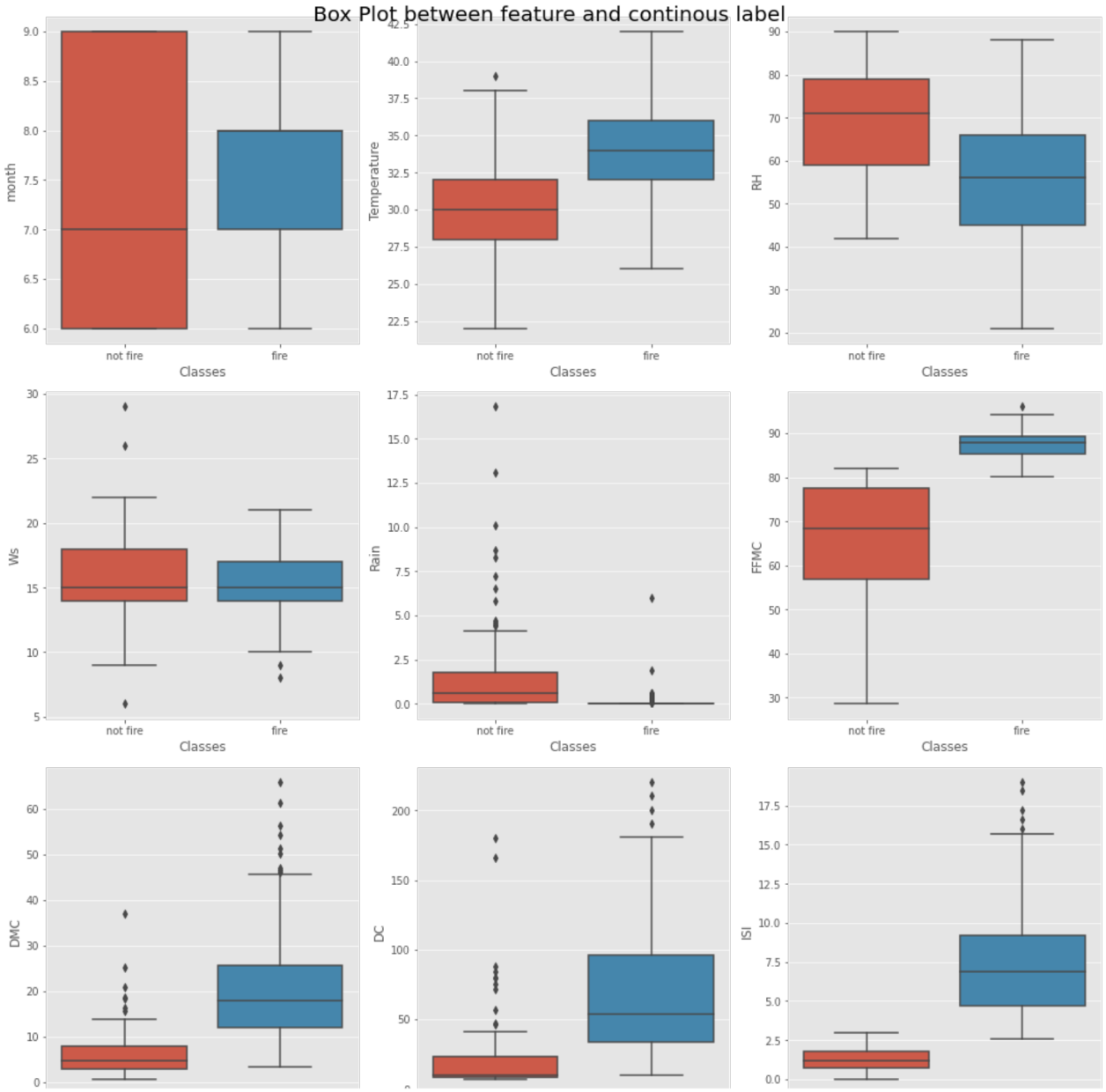
```
Index(['month', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC',
'ISI',
       'BUI', 'FWI', 'region'],
      dtype='object')
```

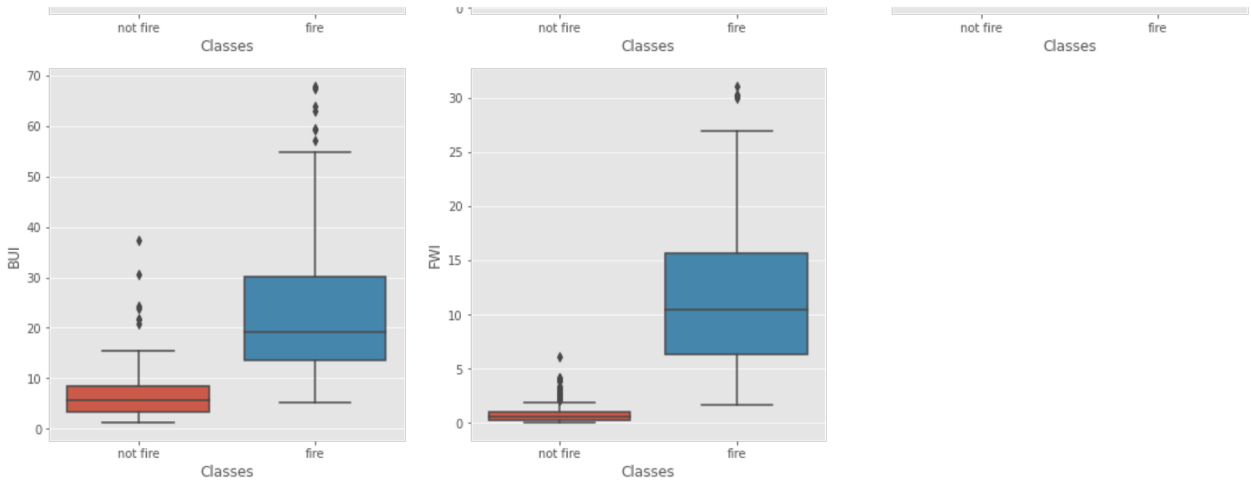
Here we will make an approach to understand the relation between the various columns with the column 'Classes'.

- **Feature columns are:** 'day', 'month', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'region'
- **Label columns:** 'Classes

```
feature_continous = [col for col in feature.columns if data[col].dtypes != 'O
feature_numeric = [col for col in feature.columns if data[col].dtypes == 'O']
```

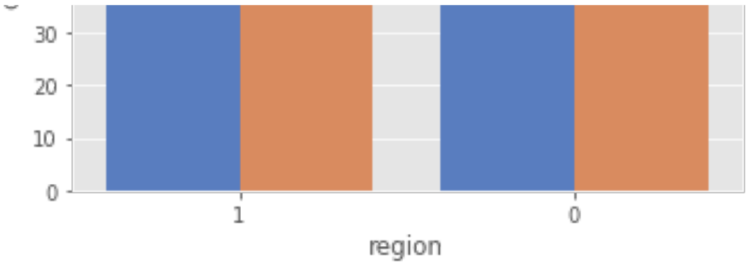
```
fig = plt.figure(figsize=(15, 50))
plt.suptitle('Box Plot between feature and continous label ', fontsize = 20,

for i in range(0, len(feature_continous)):
    ax = plt.subplot(10, 3, i+1)
    sns.boxplot(data = data, x = 'Classes', y = data[feature_continous[i]])
    plt.tight_layout()
```






```
1 # Checking for month column for impact on fire
2 plt.title('Bar graph showing the incident of fire in various month')
3 sns.countplot(data= data, hue = 'Classes', x= 'month', palette = "muted");
```


```
1 # Checking for region column for impact on fire
2 plt.title('Bar graph showing the incident of fire in various region')
3 sns.countplot(data= data, hue = 'Classes', x= 'region', palette = "muted");
4
```



Preliminary Conclusions:

1. If the temperature increases there is increase in the chance of fire.
2. If the humidity increases there is slight decrease in the chance of fire.
3. Wind speed has no impact on fire
4. If rain is increased then chance of fire is significantly decreased
5. If FFMC > 80 chance of fire is very high
6. If DMC > 10 chance of fire is very high
7. If DC > 50 chance of fire is very high
8. If ISI > 5 chance of fire is very high
9. If BUI > 15 chance of fire is very high
10. If FWI > 5 chance of fire is very high
11. Chances of fire in month of june and august is high

```
1 data.columns
2 max(data['Temperature']), min(data['Temperature']), len(data['Temperature']),
```

```
(42, 22, 243, 32.17283950617284, 3.6338264138976832)
```

Model Building

Over here we have to predict the "Temperature" based on the dataset.

```
1 data.head()
2 data['Classes'] = data['Classes'].apply(lambda x: 1 if x == 'fire' else 0)
```

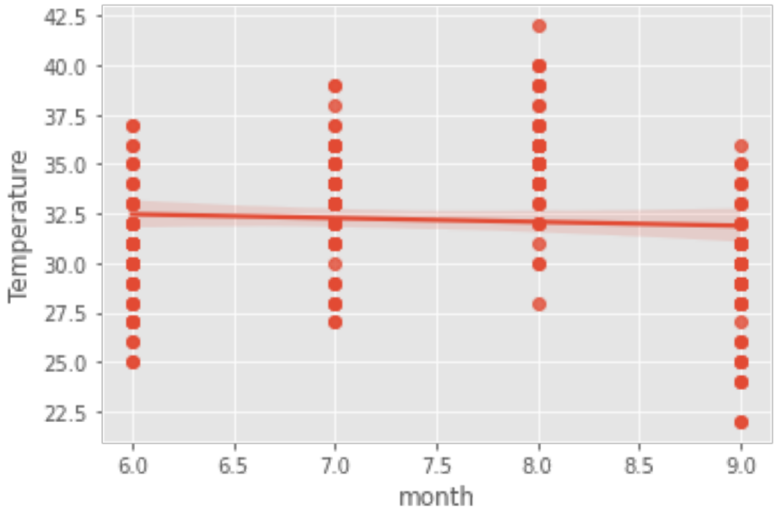
Checking for linear relation among feature and label column in dataset.

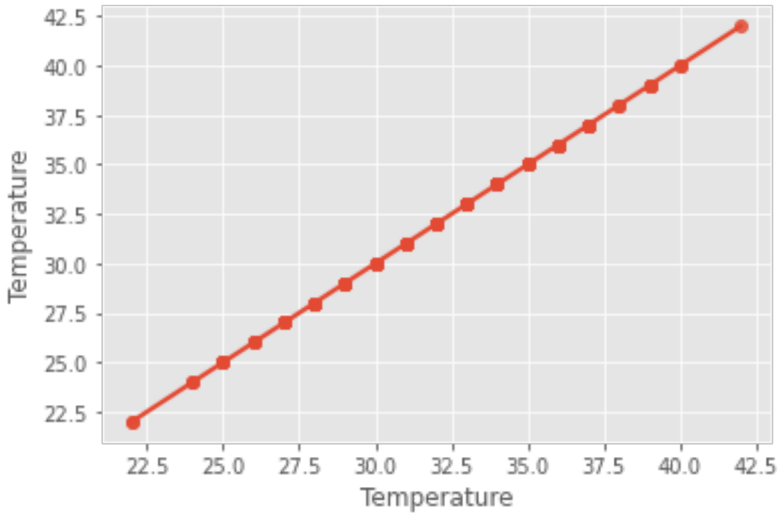
```
1 data.dtypes
```

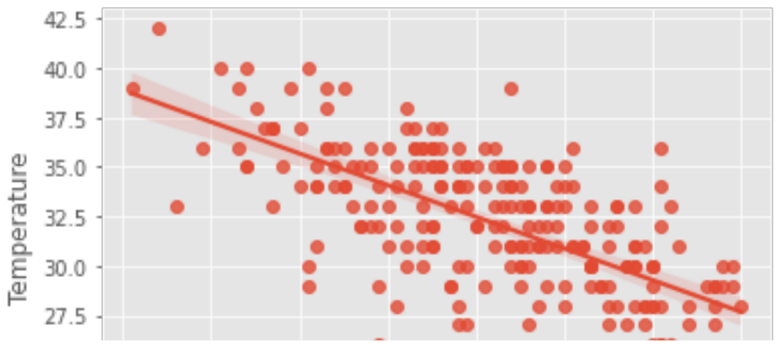
```
month            int64
Temperature      int64
RH               int64
Ws               int64
Rain           float64
FFMC           float64
```

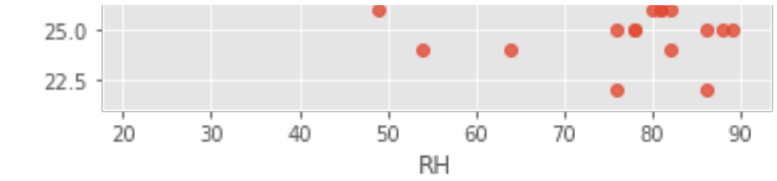
```
DMC             float64
DC              float64
ISI             float64
BUI             float64
FWI             float64
Classes           int64
region           object
dtype: object
```

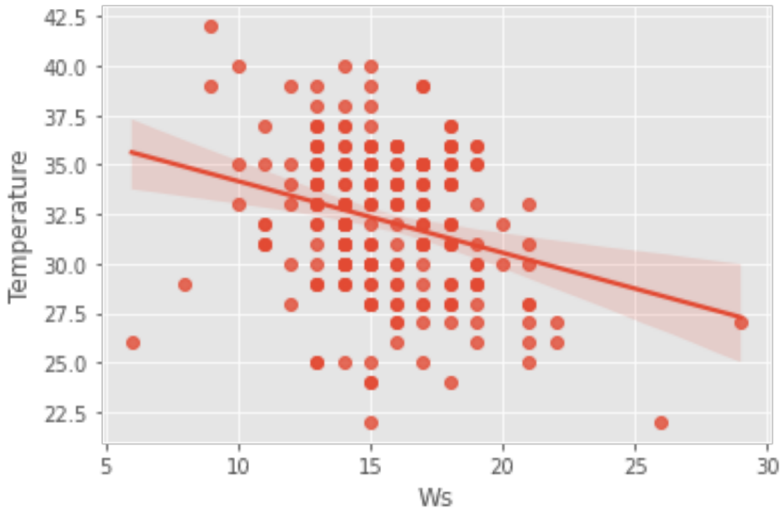
```
for col in data.columns:
    try:
        sns.regplot(x = col, y = 'Temperature', data = data)
        plt.figure()
    except:
        pass

```

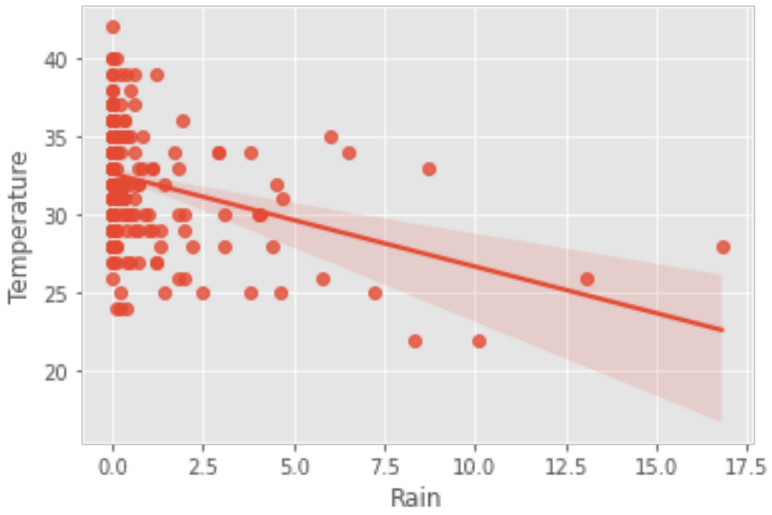


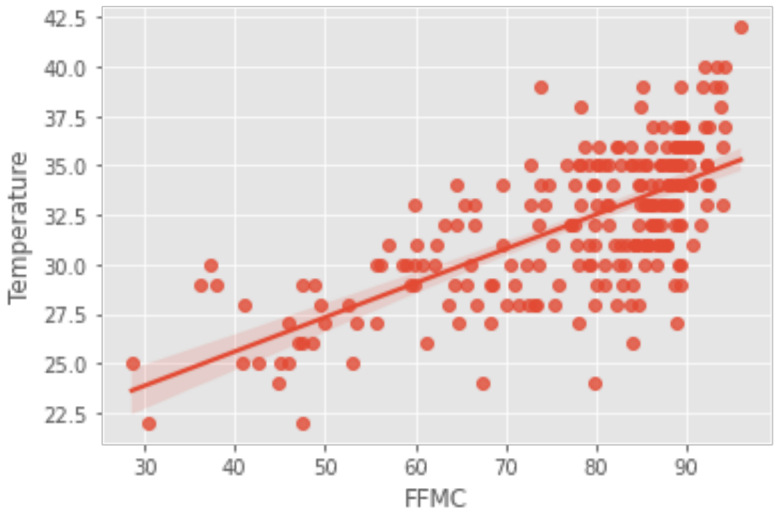


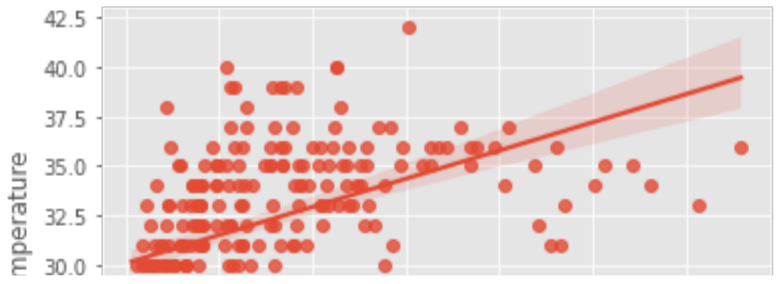


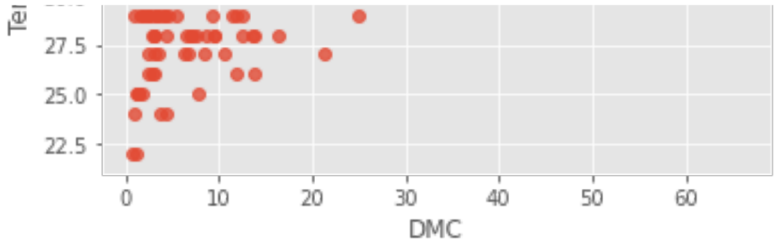

25.0
22.5
20
30
40
50
60
70
80
90
RH

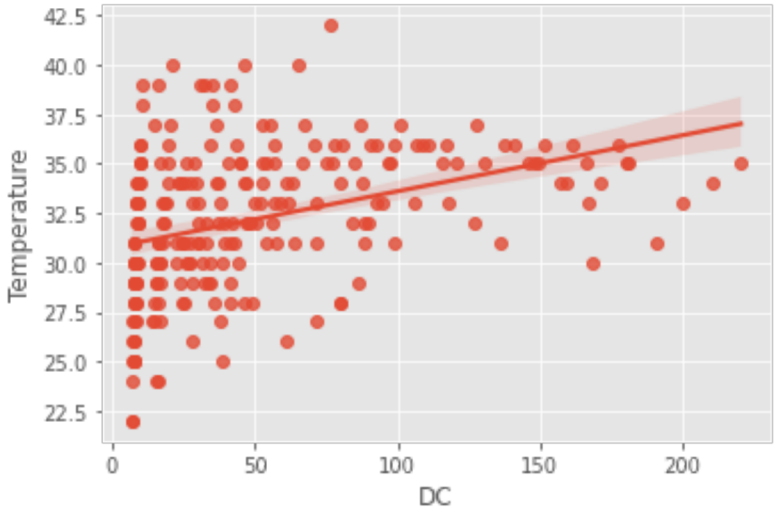

42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
Temperature
5
10
15
20
25
30
Ws

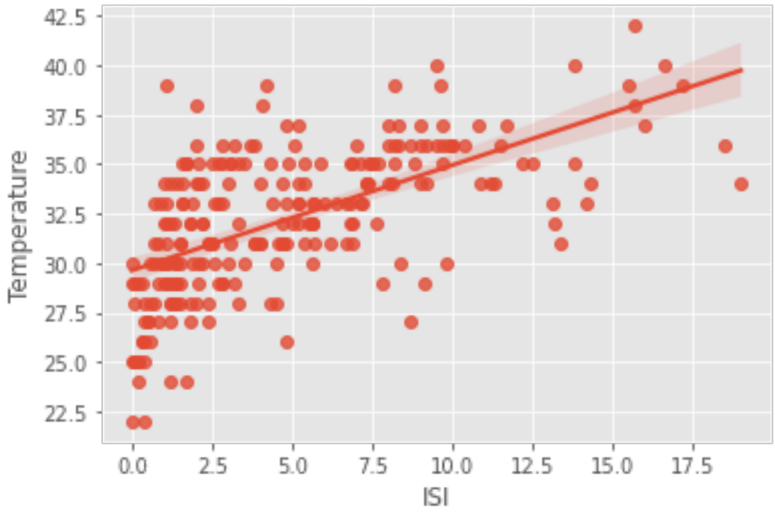

40
35
30
25
20
Temperature
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Rain

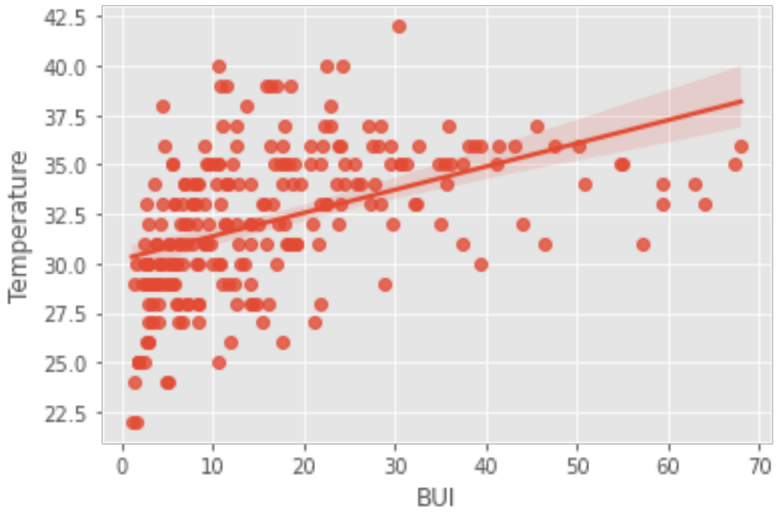

42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
Temperature
30
40
50
60
70
80
90
FFMC

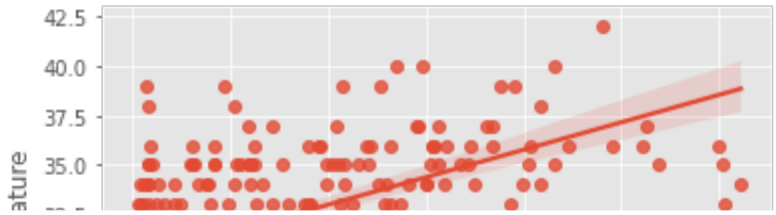

42.5
40.0
37.5
35.0
32.5
30.0
mperature

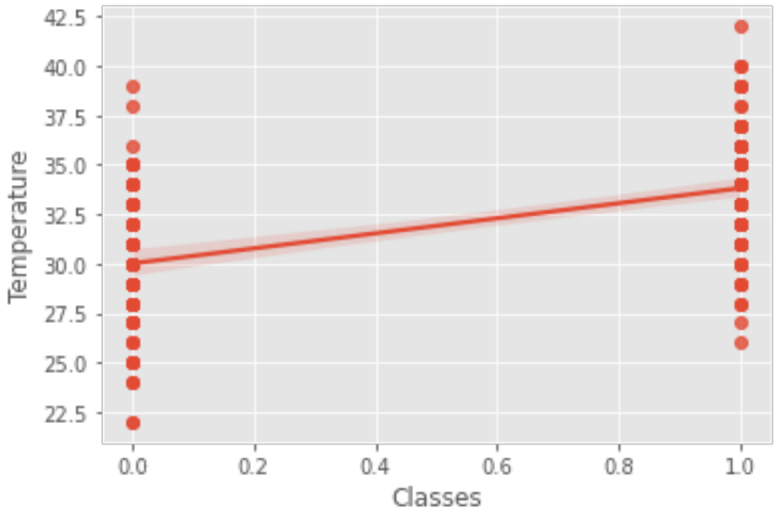

Tei
27.5
25.0
22.5
0
10
20
30
40
50
60
DMC


42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
Temperature
0
50
100
150
200
DC


42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
Temperature
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
ISI


42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
22.5
Temperature
0
10
20
30
40
50
60
70
BUI


42.5
40.0
37.5
35.0
ature



Observation:

1. Month seems irrelevant column
2. Ws, DC column has slight relation with the temperature column

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import StandardScaler
```

```
x = data.drop(columns = ['Temperature'])
y = data['Temperature']
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y , test_size = 35,
```

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Model Training

Linear regression

```
model_lr = LinearRegression()
model_lr.fit(x_train, y_train)

```

```
LinearRegression()
```

```
print('Coeff', end = ' ')
print(model_lr.coef_)
print('Intercept', end = ' ')
print(model_lr.intercept_)

```

```
Coeff [-0.44799206 -0.98565605 -0.70499455  0.02150587  1.02838502  0.19497
  0.9264854   0.45439753 -0.84423222  0.40617484 -0.1649805  -0.13619726]
Intercept 32.27403846153846
```

```
y_pred_lr = model_lr.predict(x_test)
```

Ridge Regression

```
from sklearn.linear_model import Ridge
model_ridge = Ridge()
model_ridge.fit(x_train,y_train)
y_pred_ridge = model_ridge.predict(x_test)
```

```
print('Coeff', end = ' ')
print(model_ridge.coef_, end = '\n\n')
print('Intercept', end = ' ')
print(model_ridge.intercept_)

```

```
3
```

```
Coeff [-0.44045063 -0.99241173 -0.70001123  0.01288741  1.00316896 -0.05678
  0.74235044  0.48417271 -0.3770252   0.33646932 -0.15522368 -0.13550725]

Intercept 32.27403846153846
```

LASSO Regression

```
1 from sklearn.linear_model import Lasso
2 model_lasso = Lasso()
3 model_lasso.fit(x_train, y_train)
4 y_pred_lasso = model_lasso.predict(x_test)
5
```

```
1 print('Coeff', end = ' ')
2 print(model_lasso.coef_)
3 print('Intercept', end = ' ')
4 print(model_lasso.intercept_)
5
```

```
Coeff [-0.         -0.71129503 -0.         -0.          0.83244652  0.
  0.          0.03812382  0.          0.07651554  0.         -0.        ]
Intercept 32.27403846153846
```

NOTE: As expected Lasso regression is used to find the important feature. The same is observed here. As per the Lasso regression, 'RH', 'FFMC' are the important feature

Elastic Net regression

```
1 from sklearn.linear_model import ElasticNet
2 model_elastic = ElasticNet()
3 model_elastic.fit(x_train, y_train)
4 y_pred_elastic = model_elastic.predict(x_test)
5
```

```
1 print('Coeff', end = ' ')
2 print(model_elastic.coef_)
3 print('Intercept', end = ' ')
4 print(model_elastic.intercept_)
5
```

```
Coeff [-0.         -0.61924064 -0.20611375 -0.          0.59778132  0.09704
  0.          0.32109457  0.0401042   0.24897577  0.05896694 -0.        ]
Intercept 32.27403846153846
```

NOTE: Elastic net is combination of both Ridge and lasso. It has pointed out important features as 'RH', 'Ws', 'FFMC', 'DMC', 'ISI', 'FWI', 'Classes'

feature as RH, Ws, FFMC, DMC, ISI, FWI, Classes

```
1 x.columns
```

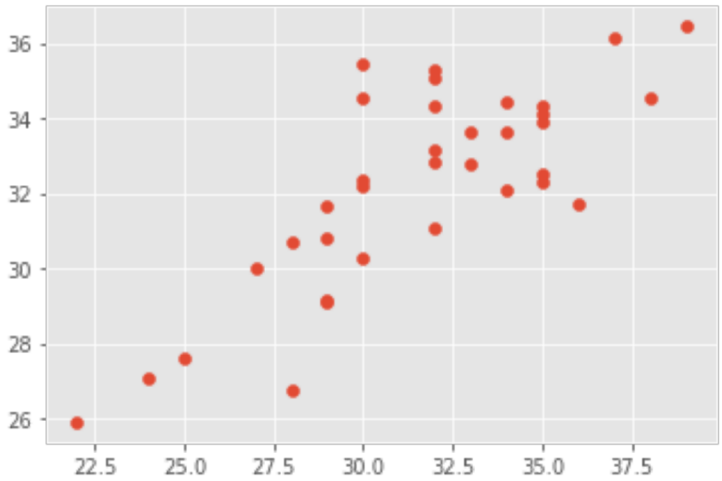
```
Index(['month', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
'FWI',
       'Classes', 'region'],
      dtype='object')
```

Assumptions of Linear Regression

1. True data and predicted data points should show linear relation

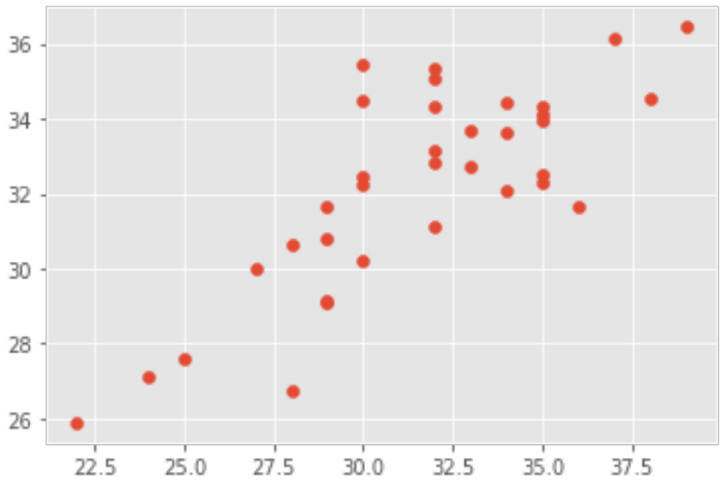
```
1 plt.scatter(y_test, y_pred_lr)
```

```
<matplotlib.collections.PathCollection at 0x7ff8a92c5f10>
```



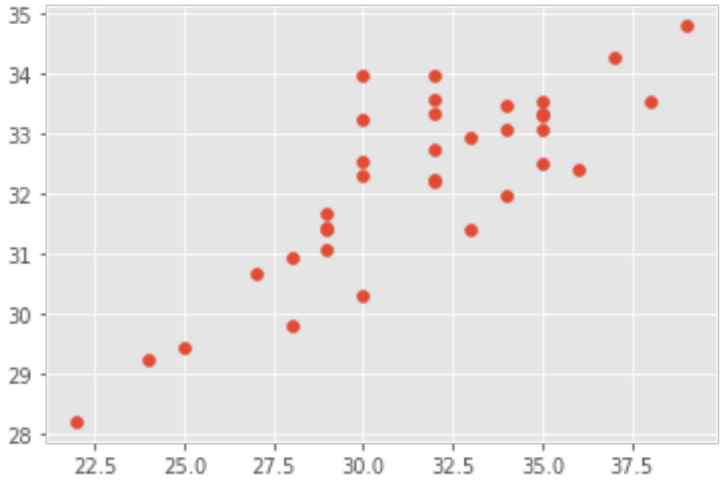

```
1 plt.scatter(y_test, y_pred_ridge)
```

```
<matplotlib.collections.PathCollection at 0x7ff8a929e650>
```



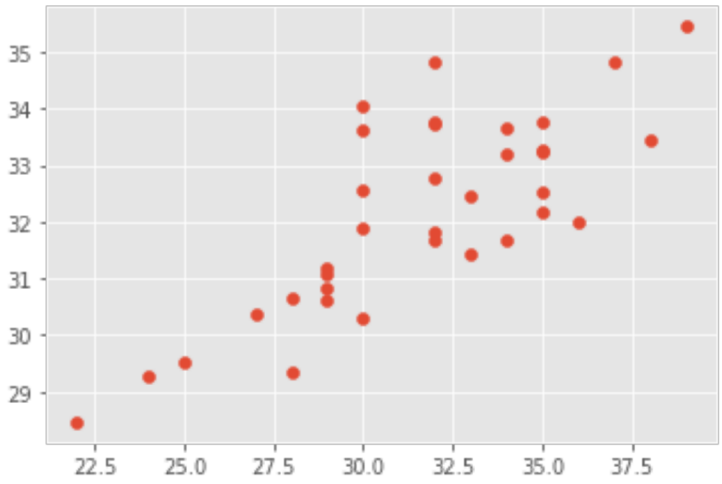

```
1 plt.scatter(y_test, y_pred_lasso)
2
```

```
<matplotlib.collections.PathCollection at 0x7ff8a92b7f10>
```




```
1 plt.scatter(y_test, y_pred_elastic)
2
```

```
<matplotlib.collections.PathCollection at 0x7ff8a91877d0>
```

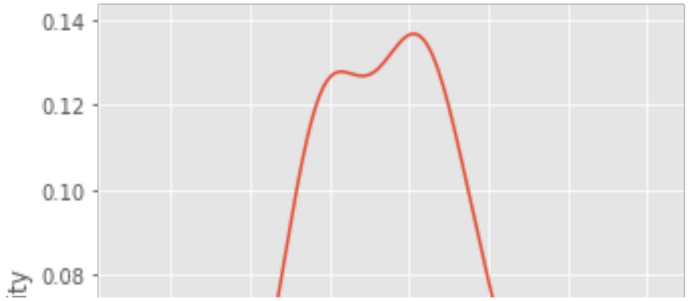


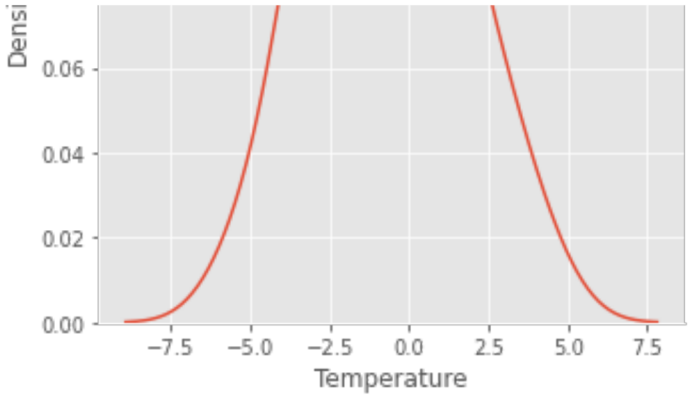
2. Residuals should be normally distributed. (Density plot is used)

Graph is normally distributed

```
1 residual_lr = y_test - y_pred_lr
2 sns.displot(residual_lr, kind = 'kde')
```

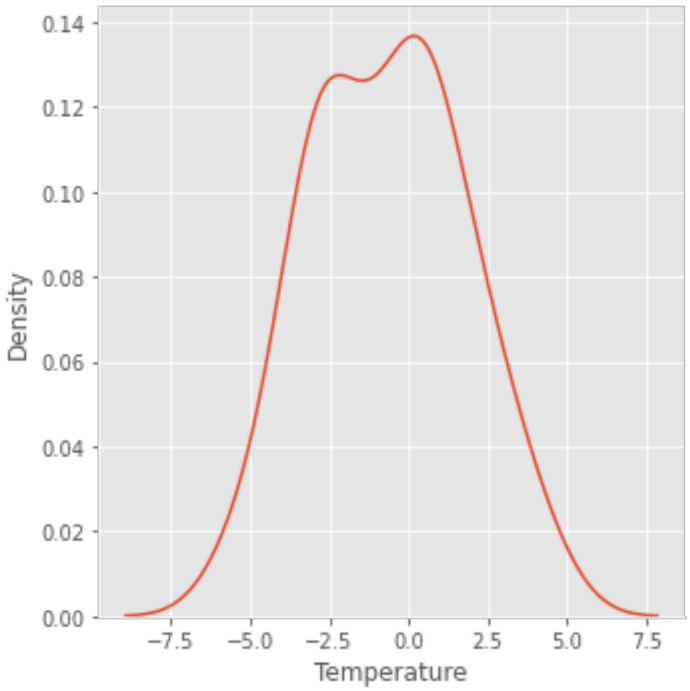
```
<seaborn.axisgrid.FacetGrid at 0x7ff8a931fed0>
```





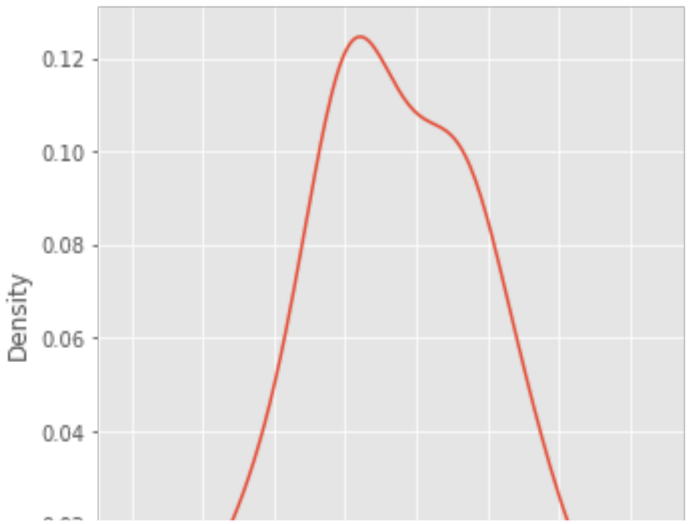

```
1 residual_ridge = y_test - y_pred_ridge
2 sns.displot(residual_ridge, kind = 'kde')
```

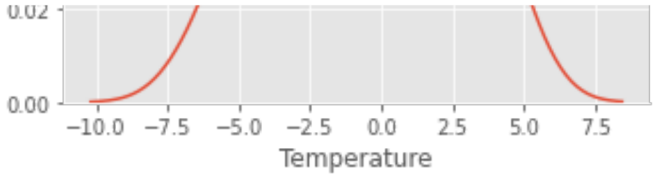
```
<seaborn.axisgrid.FacetGrid at 0x7ff8a91e5dd0>
```




```
1 residual_lasso = y_test - y_pred_lasso
2 sns.displot(residual_lasso, kind = 'kde')
```

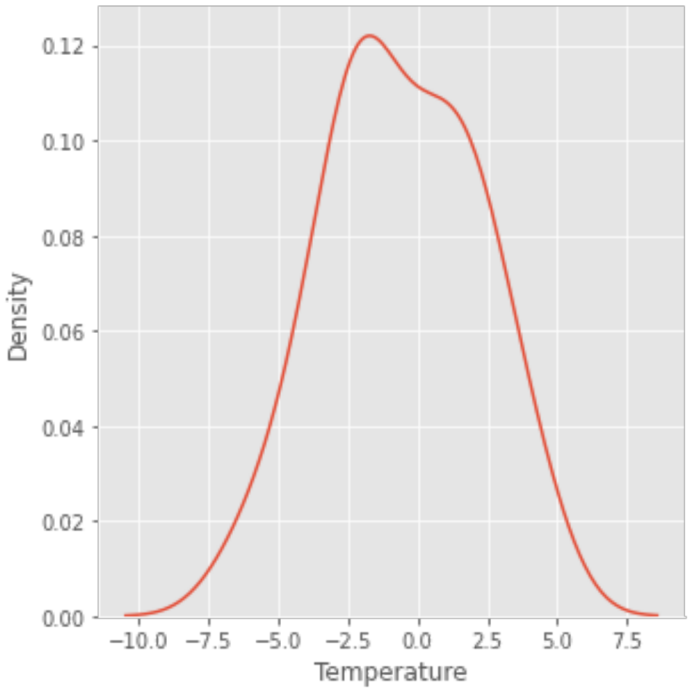
```
<seaborn.axisgrid.FacetGrid at 0x7ff8a90445d0>
```






```
1 residual_elastic = y_test - y_pred_elastic
2 sns.displot(residual_elastic, kind = 'kde')
```

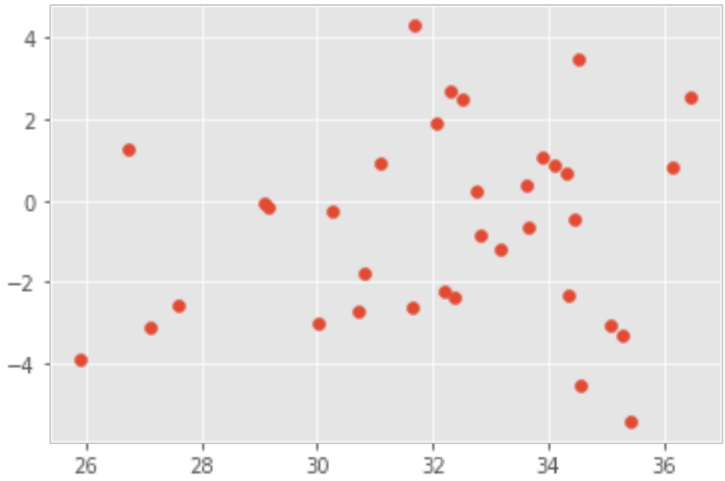
```
<seaborn.axisgrid.FacetGrid at 0x7ff8a902fe10>
```



3. Homosedacity - Scatter plot with predictions and residual.

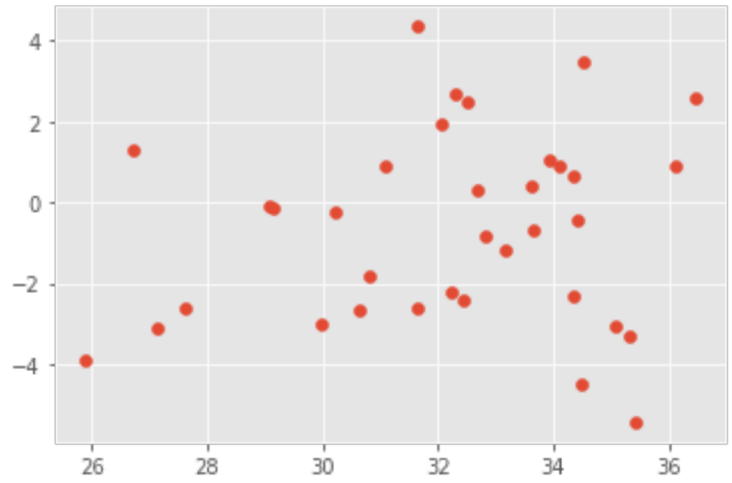
```
1 plt.scatter(y_pred_lr, residual_lr)
```

```
<matplotlib.collections.PathCollection at 0x7ff8a8f2e750>
```



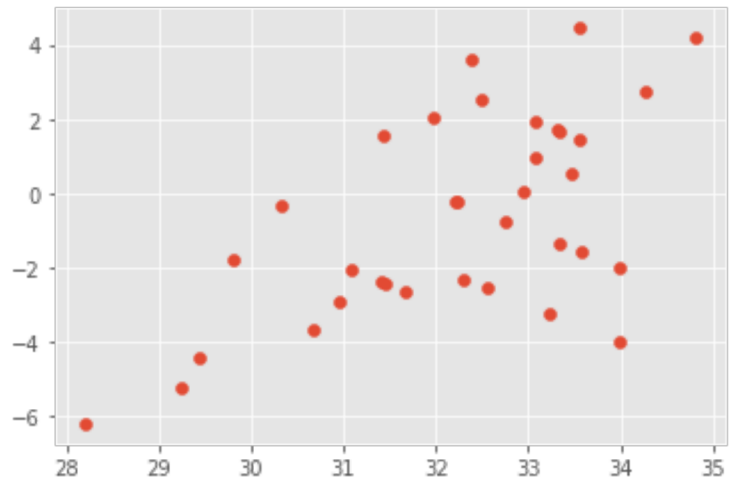

```
1 plt.scatter(y_pred_ridge, residual_ridge)
```

```
<matplotlib.collections.PathCollection at 0x7ff8a8ee3f10>
```



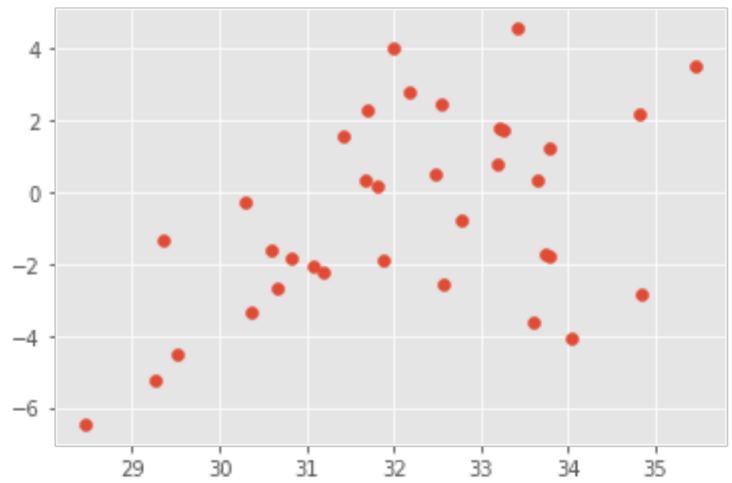
```
1 plt.scatter(y_pred_lasso, residual_lasso)
```

```
<matplotlib.collections.PathCollection at 0x7ff8a8ead1d0>
```



```
1 plt.scatter(y_pred_elastic, residual_elastic)
```

```
<matplotlib.collections.PathCollection at 0x7ff8a8dd9890>
```



Performance metrics

Mean Absolute Error

```
print('Linear Model', end = ' ')
print(mean_absolute_error(y_test,y_pred_lr))
print('Ridge Model' , end = ' ')
print(mean_absolute_error(y_test,y_pred_ridge))
print('LASSO Model' , end = ' ')
print(mean_absolute_error(y_test,y_pred_lasso))
print('Elastic Net Model' , end = ' ')
print(mean_absolute_error(y_test,y_pred_elastic))


```

```
Linear Model 2.008904033398985
Ridge Model 2.010336142677734
LASSO Model 2.3371355676061327
Elastic Net Model 2.3225254757306963
```

Mean square error

```
print('Linear Model', end = ' ')
print(mean_squared_error(y_test, y_pred_lr))
print('Linear Model', end = ' ')
print(mean_squared_error(y_test, y_pred_ridge))
print('Linear Model', end = ' ')
print(mean_squared_error(y_test, y_pred_lasso))
print('Linear Model', end = ' ')
print(mean_squared_error(y_test, y_pred_elastic))


```

```
Linear Model 5.924344644793183
Linear Model 5.922866774145532
Linear Model 7.57713089684745
Linear Model 7.562456058870997
```

```
print('Linear Model', end = ' ')
print(np.sqrt(mean_squared_error(y_test, y_pred_lr)))
print('Linear Model', end = ' ')
print(np.sqrt(mean_squared_error(y_test, y_pred_ridge)))
print('Linear Model', end = ' ')
print(np.sqrt(mean_squared_error(y_test, y_pred_lasso)))
print('Linear Model', end = ' ')

print(np.sqrt(mean_squared_error(y_test, y_pred_elastic)))

```

```
Linear Model 2.4339976673762824
Linear Model 2.433694059273994
Linear Model 2.75265887767581
Linear Model 2.749992010692212
```

R square score

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

```
score_lr = r2_score(y_test, y_pred_lr)
score_ridge = r2_score(y_test, y_pred_ridge)
score_lasso = r2_score(y_test, y_pred_lasso)
score_elastic = r2_score(y_test, y_pred_elastic)

print('R2 score Linear Regression Model', end = ' ')
print(score_lr)
print('R2 score Ridge Model', end = ' ')
print(score_ridge)
print('R2 score Lasso Model', end = ' ')
print(score_lasso)
print('R2 score Elastic Net Model', end = ' ')
print(score_elastic)
```

```
R2 score Linear Regression Model 0.5890531036312769
R2 score Ridge Model 0.5891556173087047
R2 score Lasso Model 0.4744062656490302
R2 score Elastic Net Model 0.47542419750187037
```

Adjusted R square score

```
print('Adjusted r2 score Linear model', end = ' ')
print(1 - (1-score_lr)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('Adjusted r2 score Ridge', end = ' ')
print(1 - (1-score_ridge)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('Adjusted r2 score Lasso', end = ' ')
print(1 - (1-score_lasso)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1))
print('Adjusted r2 score Elastic', end = ' ')
print(1 - (1-score_elastic)*(len(y_test)-1)/(len(y_test)-x_test.shape[1]-1)
```

```
Adjusted r2 score Linear model 0.3649002510665189
Adjusted r2 score Ridge 0.36505868129527086
Adjusted r2 score Lasso 0.18771877418486482
Adjusted r2 score Elastic 0.1892919415937997
```

Colab paid products - Cancel contracts here