Regression by Divyani Pathak

Life Cycle of machine learning Project

a. Understand the problem statement b. Data collection c. Data cleaning d. EDA e. Preprocessing [F.E] f. model Training g. Choose Model[validation]

1. Understand the Problem statement

In [ ]:

```python
if user can predict that the forest will catch fire or not based on input features.
prediction result can be used to take prevention to avoid it in future.
```

2. Data Collection:

The dataset is collected from website named, UCI Machine Learning Repository. The dataset consists of 15 columns and 244 rows

2.1 Import required packages

In [1]:

```python
#importing pandas, numpy, matplotlib, seaborn and warnings Library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings

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

2.2 Load csv files

In [54]:

```python
df=pd.read_csv("Algerian_forest_fires_dataset_UPDATE.csv")
```

In [55]:

```python
df
```

Out[55]:

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

247 rows × 1 columns

In [56]:

```python
df=pd.read_csv("Algerian_forest_fires_dataset_UPDATE.csv",header=1)
```

In [11]:

```python
df
```

Out[11]:

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

246 rows × 14 columns

3. DATA cleaning

Removing unnecessary rows from dataset

In [57]:

```python
##remove unnecessary rows from dataset
df.drop(index=[122,123], inplace=True)
df.reset_index(inplace=True)
df.drop('index',axis=1, inplace=True)   #axis=1 represent column
```

In [58]:

```python
df
```

Out[58]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire
240	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire
241	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire
242	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
243	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire

244 rows × 14 columns

Adding New Feature Named Region in a dataset

In [59]:

```python
##adding new feature named 'Region' in a dataset
df.loc[:122, 'region'] = 'bejaia'
df.loc[122:, 'region'] = 'Sidi-Bel Abbes'
```

In [60]:

```python
df.head()
```

Out[60]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	bejaia
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire	bejaia
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	bejaia
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire	bejaia
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire	bejaia

In [61]:

```python
##all column names
df.columns
```

Out[61]:

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

Stripping the names of Columns

In [62]:

```python
###some column names has spaces which needs to be removed
df.columns=[i.strip() for i in df.columns]
```

In [63]:

```python
df.columns
```

Out[63]:

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

Stripping the Features data

In [64]:

```python
########only categorical data need to check
df['region'].unique()
```

Out[64]:

```
array(['bejaia', 'Sidi-Bel Abbes'], dtype=object)
```

In [65]:

```python
##classes column have values which has spaces
df['Classes'].unique()
```

Out[65]:

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

In [66]:

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

Out[66]:

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

In [67]:

```python
df.head()
```

Out[67]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	bejaia
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire	bejaia
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	bejaia
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire	bejaia
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire	bejaia

EDA

4.1 Profile of the DATA

In [68]:

```python
##getting shape n size of data
df.shape   ##OBServ.: it has 242 rows and 15 column
```

Out[68]:

```
(244, 15)
```

In [69]:

```python
##getting column name of dataset
df.columns
```

Out[69]:

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

In [70]:

```python
##top 5 rows
df.head()
```

Out[70]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	bejaia
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire	bejaia
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	bejaia
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire	bejaia
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire	bejaia

In [71]:

```python
##last 5 rows
df.tail()
```

Out[71]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region
239	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire	Sidi-Bel Abbes
240	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire	Sidi-Bel Abbes
241	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire	Sidi-Bel Abbes
242	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire	Sidi-Bel Abbes
243	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire	Sidi-Bel Abbes

In [72]:

```python
##checking missing value present or not
df.isnull().sum()  ##observation Classes has 1 Null value
```

Out[72]:

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

In [73]:

```python
##unique value of classes feature
df['Classes'].unique()
```

Out[73]:

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

In [74]:

```python
##handling categorical features [Classes]
df['Classes']=df['Classes'].map({ 'not fire':0 ,'fire':1 } )
```

In [75]:

```python
df.head()
```

Out[75]:

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

In [76]:

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

Out[76]:

```
array([ 0.,  1., nan])
```

Focus on replacing Null Values

In [78]:

```python
#focus on replacing null values.
#the best way to replce null value by using mode

df['Classes'].mode() [0]
```

Out[78]:

```
1.0
```

In [80]:

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

In [82]:

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

Out[82]:

```
array([0., 1.])
```

In [84]:

```python
df.isnull().sum()   ###now we zero null value in dataset
```

Out[84]:

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

In [86]:

```python
##check feature/col. datatype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          244 non-null    object 
 1   month        244 non-null    object 
 2   year         244 non-null    object 
 3   Temperature  244 non-null    object 
 4   RH           244 non-null    object 
 5   Ws           244 non-null    object 
 6   Rain         244 non-null    object 
 7   FFMC         244 non-null    object 
 8   DMC          244 non-null    object 
 9   DC           244 non-null    object 
 10  ISI          244 non-null    object 
 11  BUI          244 non-null    object 
 12  FWI          244 non-null    object 
 13  Classes      244 non-null    float64
 14  region       244 non-null    object 
dtypes: float64(1), object(14)
memory usage: 28.7+ KB
```

In [88]:

```python
##changing datatype
df['day']=df['day'].astype(int)
df['month']=df['month'].astype(int)
df['year']=df['year'].astype(int)
df['Temperature']=df['Temperature'].astype(int)
df['RH']=df['RH'].astype(int)
df['Rain']=df['Rain'].astype(float)
df['FFMC']=df['FFMC'].astype(float)
df['DMC']=df['DMC'].astype(float)
df['ISI']=df['ISI'].astype(float)
df['BUI']=df['BUI'].astype(float)
df['Ws']=df['Ws'].astype(int)
```

In [90]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
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    int32  
 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    object 
 10  ISI          244 non-null    float64
 11  BUI          244 non-null    float64
 12  FWI          244 non-null    object 
 13  Classes      244 non-null    float64
 14  region       244 non-null    object 
dtypes: float64(6), int32(6), object(3)
memory usage: 23.0+ KB
```

observation for datatype:

1. there is 244 rows n 15 column
2. there are 3 data type : float64 , int32, object
3. datatype included: float64-6 column int32 - 6 column object - 3 column

In [91]:

```python
##checking usage of menory
df.memory_usage()
```

Out[91]:

```
Index           128
day             976
month           976
year            976
Temperature     976
RH              976
Ws              976
Rain           1952
FFMC           1952
DMC            1952
DC             1952
ISI            1952
BUI            1952
FWI            1952
Classes        1952
region         1952
dtype: int64
```

[A] Numerical and categorical features:

---divinding the columns as numeriacal n categorical

Numerical Features

In [92]:

```python
#1. getting numerical features from dataset
#2. creating Numerical dataset
num_col=[i for i in df.columns if df[i].dtype != 'O']

#printing num_col
print('We have {} numerical feature : {}'.format(len(num_col), num_col))
```

```
We have 12 numerical feature : ['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'IS
I', 'BUI', 'Classes']
```

Categorical Features

In [93]:

```python
#1. getting categorical features from dataset
#2. creating categorical dataset
cat_col=[i for i in df.columns if df[i].dtype == 'O']

#printing cat_col
print('We have {} categorical feature : {}'.format(len(cat_col), cat_col))
```

```
We have 3 categorical feature : ['DC', 'FWI', 'region']
```

In [94]:

```python
df.Ws
```

Out[94]:

```
0      18
1      13
2      22
3      13
4      16
       ..
239    14
240    15
241    29
242    18
243    15
Name: Ws, Length: 244, dtype: int32
```

[B]: Features information

In [95]:

```python
df.head(3)
```

Out[95]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0.0	bejaia
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0.0	bejaia
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0.0	bejaia

Date observation

1. Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012)

Weather data observation:

1. Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42
2. RH : Relative Humidity in %: 21 to 90
3. Ws :Wind speed in km/h: 6 to 29
4. Rain: total day in mm: 0 to 16.8

FWI Component

1. Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5
2. Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9
3. Drought Code (DC) index from the FWI system: 7 to 220.4
4. Initial Spread Index (ISI) index from the FWI system: 0 to 18.5
5. Buildup Index (BUI) index from the FWI system: 1.1 to 68
6. Fire Weather Index (FWI) Index: 0 to 31.1
7. Classes: two classes, namely Fire and not fire

4.2 Statistics Analysis

[A]: univariant analysis

:it refers to the analysis of one variable prefix "uni" means one. the purpose is to understand the distribution of values for a single variable.

In [97]:

```python
df.var() # to check variance of numeriacal features
```

Out[97]:

```
day             77.881670
month            1.238683
year             0.000000
Temperature     13.204817
RH             221.539415
Ws               7.897102
Rain             3.997623
FFMC           205.565939
DMC            152.968382
ISI             17.433281
BUI            201.777024
Classes          0.246711
dtype: float64
```

In [98]:

```python
df.describe()
```

Out[98]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	BUI	Cla
count	244.000000	244.000000	244.0	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.000000	244.00
mean	15.754098	7.500000	2012.0	32.172131	61.938525	15.504098	0.760656	77.887705	14.673361	4.774180	16.664754	0.56
std	8.825059	1.112961	0.0	3.633843	14.884200	2.810178	1.999406	14.337571	12.368039	4.175318	14.204824	0.49
min	1.000000	6.000000	2012.0	22.000000	21.000000	6.000000	0.000000	28.600000	0.700000	0.000000	1.100000	0.00
25%	8.000000	7.000000	2012.0	30.000000	52.000000	14.000000	0.000000	72.075000	5.800000	1.400000	6.000000	0.00
50%	16.000000	7.500000	2012.0	32.000000	63.000000	15.000000	0.000000	83.500000	11.300000	3.500000	12.250000	1.00
75%	23.000000	8.000000	2012.0	35.000000	73.250000	17.000000	0.500000	88.300000	20.750000	7.300000	22.525000	1.00
max	31.000000	9.000000	2012.0	42.000000	90.000000	29.000000	16.800000	96.000000	65.900000	19.000000	68.000000	1.00

In [99]:

```python
df.cov()
```

Out[99]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	B
day	7.788167e+01	4.641920e-16	0.0	3.071308	-9.747689	1.165621	-1.980908	28.346758	53.654328	6.548769	64.839
month	4.641920e-16	1.238683e+00	0.0	-0.238683	-0.627572	-0.129630	0.078601	0.248560	0.938477	0.286626	1.3567
year	0.000000e+00	0.000000e+00	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
Temperature	3.071308e+00	-2.386831e-01	0.0	13.204817	-35.396782	-2.840215	-2.374270	35.297598	21.712423	9.218043	23.512
RH	-9.747689e+00	-6.275720e-01	0.0	-35.396782	221.539415	9.874739	6.635431	-137.785533	-74.580245	-42.920524	-73.709
Ws	1.165621e+00	-1.296296e-01	0.0	-2.840215	9.874739	7.897102	0.956129	-6.577727	-0.043306	0.178913	1.187
Rain	-1.980908e+00	7.860082e-02	0.0	-2.374270	6.635431	0.956129	3.997623	-15.595918	-7.135415	-2.897687	-8.496
FFMC	2.834676e+01	2.485597e-01	0.0	35.297598	-137.785533	-6.577727	-15.595918	205.565939	106.820535	44.283138	120.090
DMC	5.365433e+01	9.384774e-01	0.0	21.712423	-74.580245	-0.043306	-7.135415	106.820535	152.968382	34.831449	172.536
ISI	6.548769e+00	2.866255e-01	0.0	9.218043	-42.920524	0.178913	-2.897687	44.283138	34.831449	17.433281	37.714
BUI	6.483903e+01	1.356790e+00	0.0	23.512265	-73.700941	1.187799	-8.496825	120.090018	172.536341	37.714477	201.777
Classes	8.845038e-01	1.234568e-02	0.0	0.935168	-3.216117	-0.092862	-0.376833	5.484349	3.588791	1.525363	4.119

[A.1]KDE plot and scatter plot ->numerical feature analysis [uni-variant analysis]

In [100…

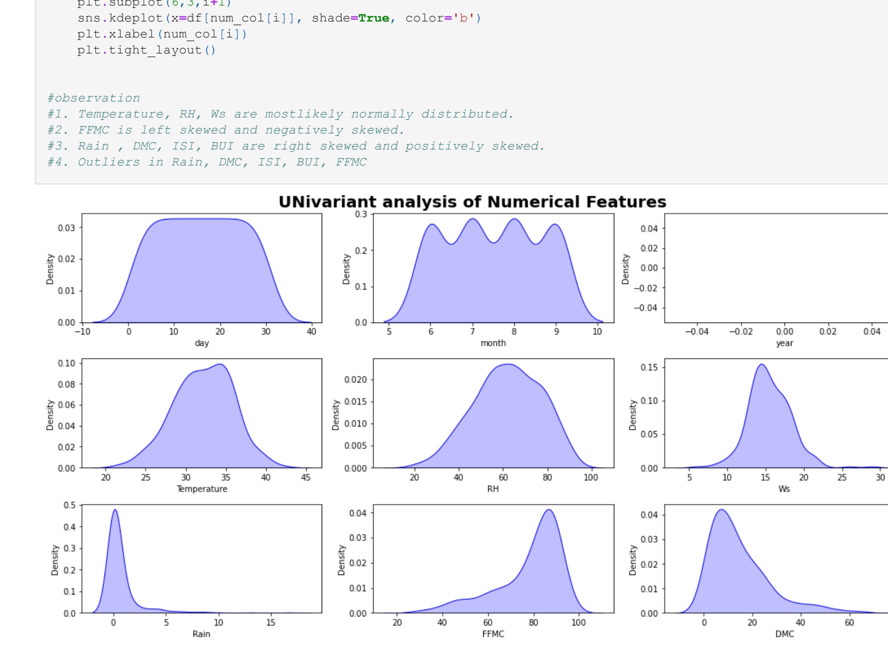
```python
num_col
```

Out[100…

```
['day',
 'month',
 'year',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'ISI',
 'BUI',
 'Classes']
```

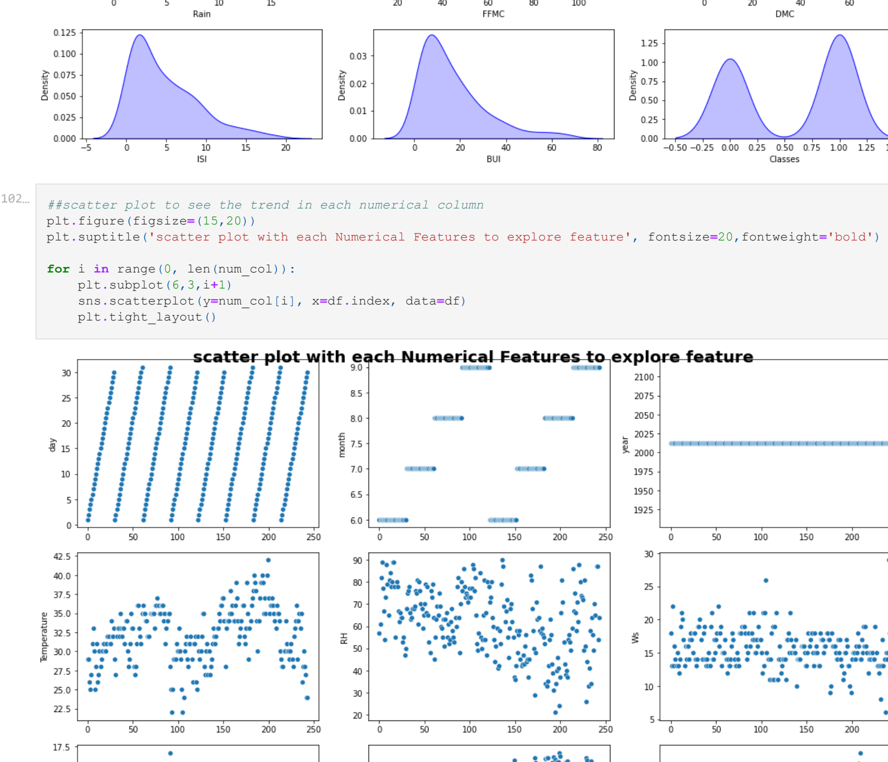
In [101…

```python
plt.figure(figsize=(15,15))
plt.suptitle('UNivariant analysis of Numerical Features', fontsize=20,fontweight='bold')

for i in range(0, len(num_col)):
    plt.subplot(6,3,i+1)
    sns.kdeplot(x=df[num_col[i]], shade=True, color='b')
    plt.xlabel(num_col[i])
    plt.tight_layout()


#observation
#1. Temperature, RH, Ws are mostlikely normally distributed.
#2. FFMC is left skewed and negatively skewed.
#3. Rain , DMC, ISI, BUI are right skewed and positively skewed.
#4. Outliers in Rain, DMC, ISI, BUI, FFMC
```



In [102…

```python
##scatter plot to see the trend in each numerical column
plt.figure(figsize=(15,20))
plt.suptitle('scatter plot with each Numerical Features to explore feature', fontsize=20,fontweight='bold')

for i in range(0, len(num_col)):
    plt.subplot(6,3,i+1)
    sns.scatterplot(y=num_col[i], x=df.index, data=df)
    plt.tight_layout()
```



[A.2] count plot ->categorical feature analysis [univariant analysis]

In [103…

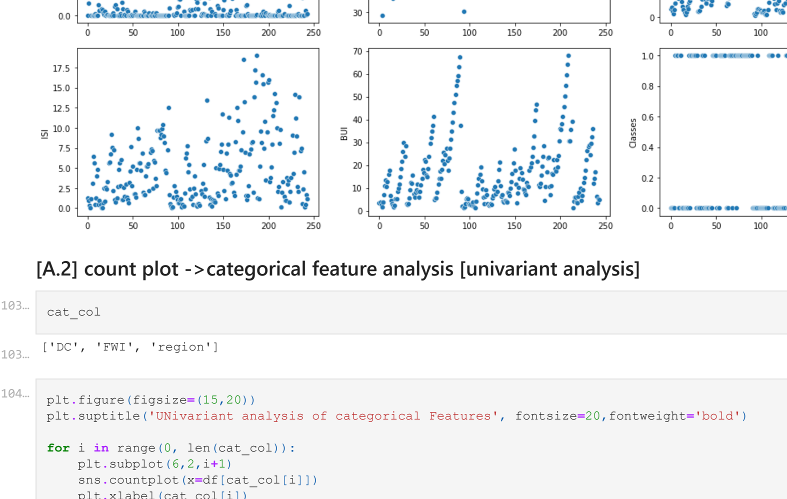
```python
cat_col
```

Out[103…

```
['DC', 'FWI', 'region']
```

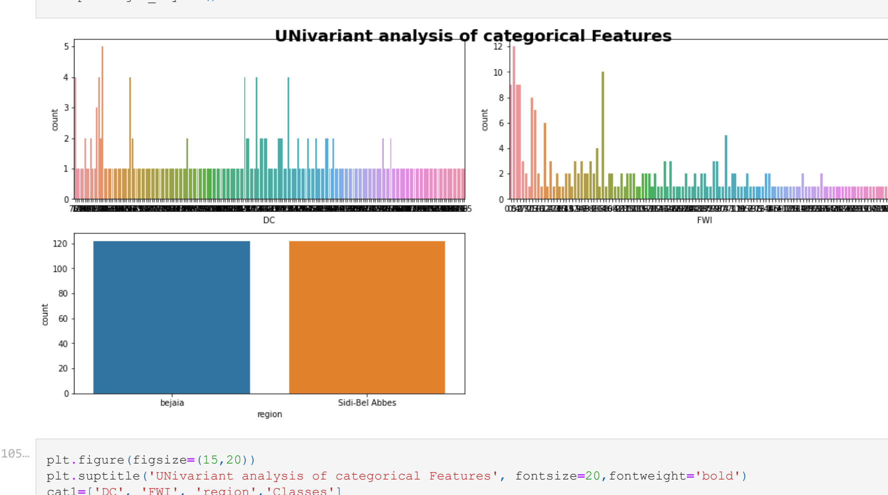
In [104…

```python
plt.figure(figsize=(15,20))
plt.suptitle('UNivariant analysis of categorical Features', fontsize=20,fontweight='bold')

for i in range(0, len(cat_col)):
    plt.subplot(6,2,i+1)
    sns.countplot(x=df[cat_col[i]])
    plt.xlabel(cat_col[i])
    plt.tight_layout()
```



In [105…

```python
plt.figure(figsize=(15,20))
plt.suptitle('UNivariant analysis of categorical Features', fontsize=20,fontweight='bold')
cat1=['DC', 'FWI', 'region','Classes']
for i in range(0, len(cat1)):
    plt.subplot(6,2,i+1)
    sns.countplot(x=df[cat1[i]])
    plt.xlabel(cat1[i])
    plt.tight_layout()

#observation
#1. fire has occured more time
```

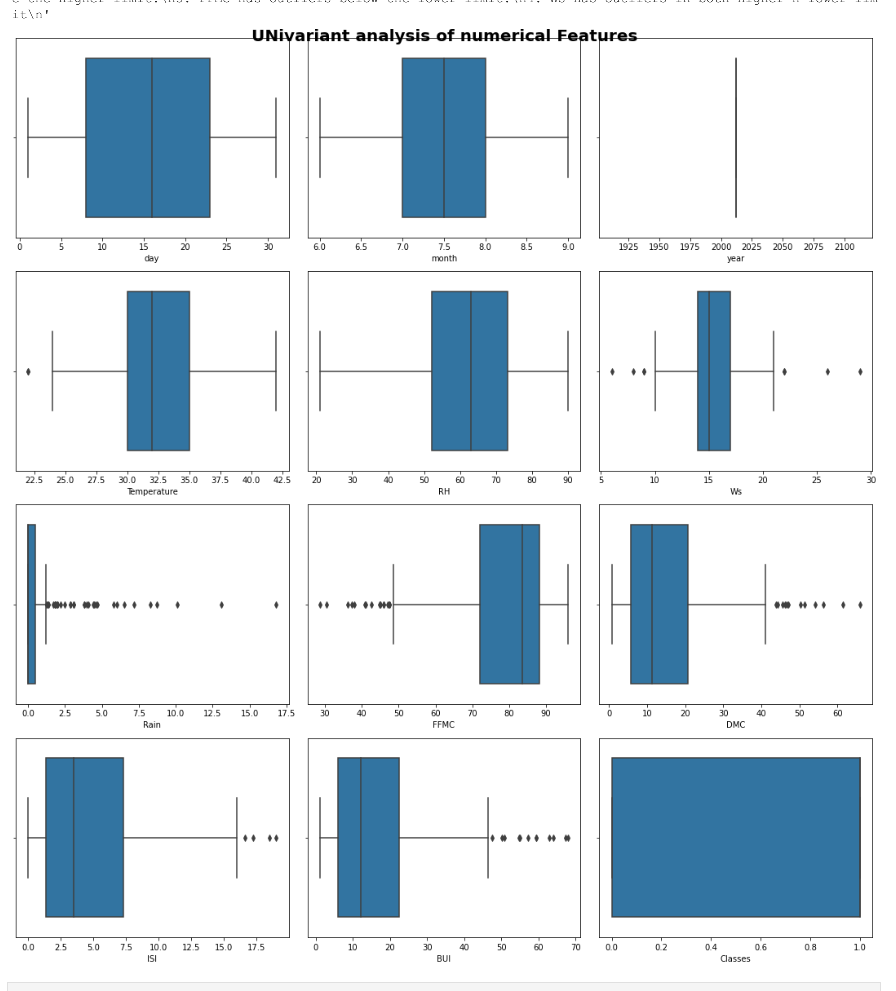


[A.3] Box plot => to find outlier

In [106…

```python
#box plot to find outliers in the features
plt.figure(figsize=(15,20))
plt.suptitle('UNivariant analysis of numerical Features', fontsize=20,fontweight='bold')

for i in range(0, len(num_col)):
    plt.subplot(5,3,i+1)
    sns.boxplot(x=df[num_col[i]])
    plt.xlabel(num_col[i])
    plt.tight_layout()

### observations:
'''
1. exceot RH, every numerical variable has the outlier.
2. Rain, BUI, ISI, FWI, DC ahs outliers in the above the higher limit.
3. FFMC has outliers below the lower limit.
4. Ws has outliers in both higher n lower limit
'''
```

Out[106…
'\n1. exceot RH, every numerical variable has the outlier.\n2. Rain, BUI, ISI, FWI, DC ahs outliers in the above the higher limit.\n3. FFMC has outliers below the lower limit.\n4. Ws has outliers in both higher n lower limit\n'



In [107…
```
# stripplot (categorical vs numeriacal) [bivariant]
# scatterplot/pairplot (numerical vs numerical) (check correaltion) [uni]
# count plot ( categorical vs count) [uni]
# box plot (outliers) [uni]
# heatmap (correlation) and for all column [multivariant]
# lineplot ( trend in numerical feature with time)
```

Multivariant analysis

it describe the analysis of multiple variables

In [108…
```
df.columns
```
Out[108…
```
Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'region'],
      dtype='object')
```

In [109…
```
num_col
```
Out[109…
```
['day',
 'month',
 'year',
 'Temperature',
 'RH',
 'Ws',
 'Rain',
 'FFMC',
 'DMC',
 'ISI',
 'BUI',
 'Classes']
```

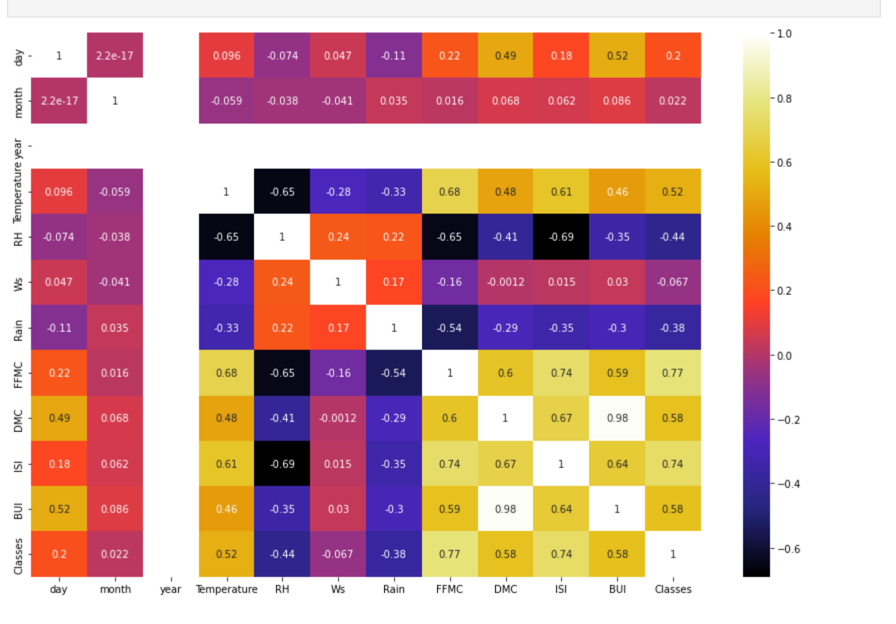
In [110…
```
cat_col
```
Out[110…
```
['DC', 'FWI', 'region']
```

In [111…
```
df.corr()
```
Out[111…

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	BUI	Cla
day	1.000000e+00	2.232788e-17	NaN	0.095772	-0.074209	0.047001	-0.112265	0.224032	0.491571	0.177727	0.517229	0.201
month	2.232788e-17	1.000000e+00	NaN	-0.059017	-0.037884	-0.041447	0.035322	0.015577	0.068178	0.061680	0.085822	0.022
year	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
Temperature	9.577222e-02	-5.901677e-02	NaN	1.000000	-0.654443	-0.278132	-0.326786	0.677491	0.483105	0.607551	0.455504	0.518
RH	-7.420934e-02	-3.788419e-02	NaN	-0.654443	1.000000	0.236084	0.222968	-0.645658	-0.405133	-0.690637	-0.348587	-0.435
Ws	4.700086e-02	-4.144673e-02	NaN	-0.278132	0.236084	1.000000	0.170169	-0.163255	-0.001246	0.015248	0.029756	-0.066
Rain	-1.122654e-01	3.532207e-02	NaN	-0.326786	0.222968	0.170169	1.000000	-0.544045	-0.288548	-0.347105	-0.299171	-0.379
FFMC	2.240321e-01	1.557668e-02	NaN	0.677491	-0.645658	-0.163255	-0.544045	1.000000	0.602391	0.739730	0.589652	0.770
DMC	4.915710e-01	6.817778e-02	NaN	0.483105	-0.405133	-0.001246	-0.288548	0.602391	1.000000	0.674499	0.982073	0.584
ISI	1.777266e-01	6.168011e-02	NaN	0.607551	-0.690637	0.015248	-0.347105	0.739730	0.674499	1.000000	0.635891	0.735
BUI	5.172292e-01	8.582162e-02	NaN	0.455504	-0.348587	0.029756	-0.299171	0.589652	0.982073	0.635891	1.000000	0.583
Classes	2.017844e-01	2.233266e-02	NaN	0.518119	-0.435023	-0.066529	-0.379449	0.770114	0.584188	0.735511	0.583882	1.000

[A] Heatmap

In [112…
```
##plotting heat map
plt.figure(figsize = (15,10))
sns.heatmap(df.corr(), cmap="CMRmap", annot=True)
plt.show()

##observation
#1. temperature has +ve correlation with FFMC,DMC,ISI, BUI, CLASSES and -ve correlation with Rain, Ws, RH
#2. BUI has more +ve correlated with DMC
```

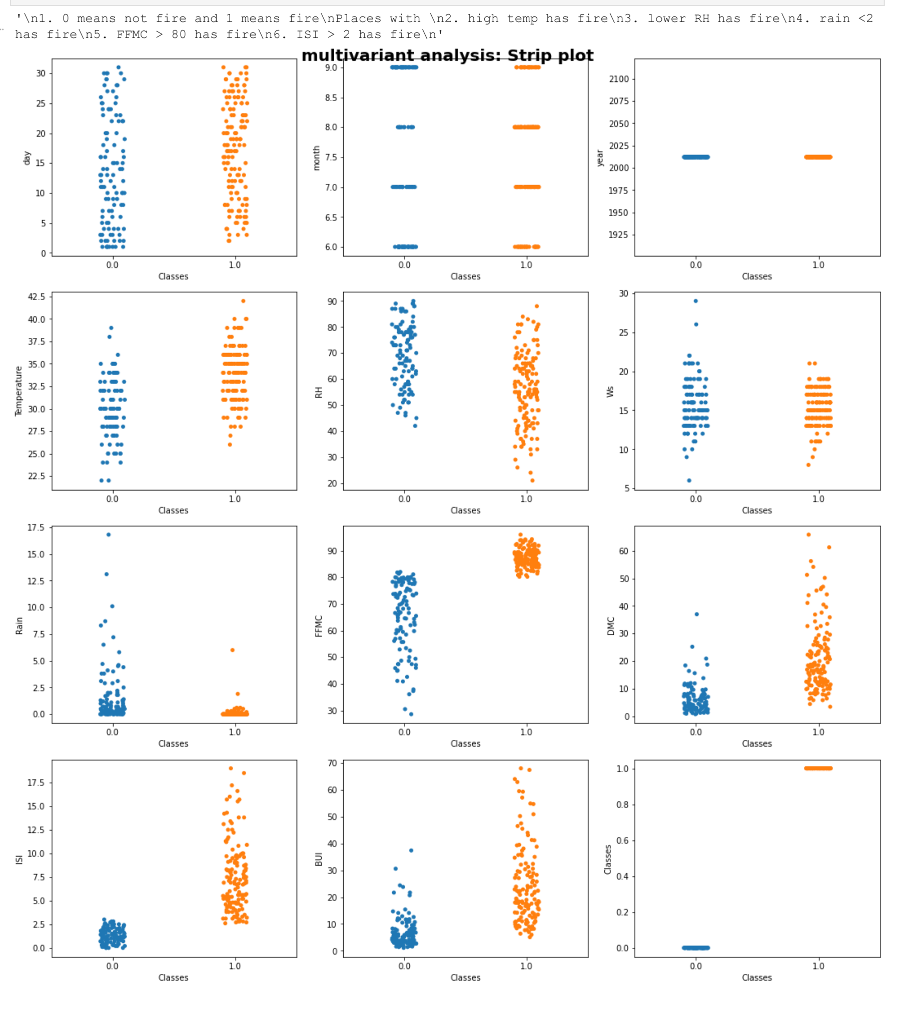


Bivariant analysis

[A] Strip plot

to see relationship between numerical feature and cate. feature

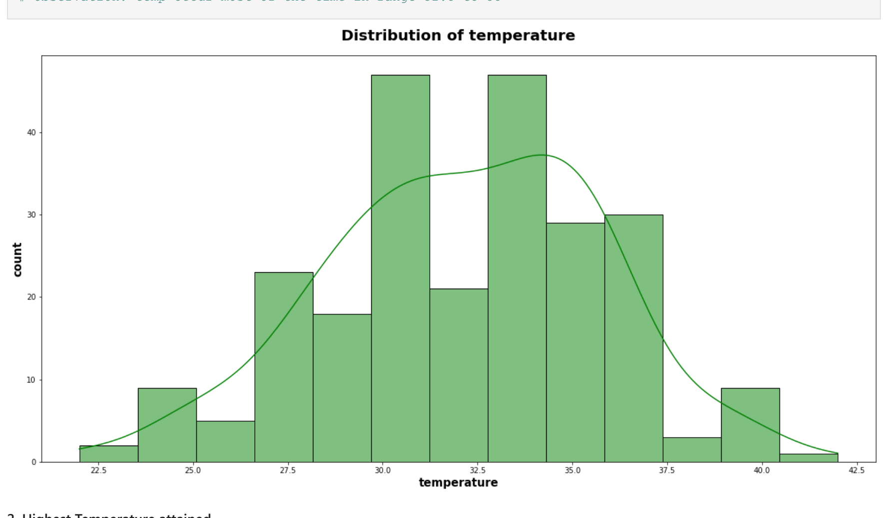
In [113…
```
plt.figure(figsize=(15,15))
plt.suptitle('multivariant analysis: Strip plot', fontsize=20,fontweight='bold')

for i in range(0, len(num_col)):
    plt.subplot(5,3,i+1)
    sns.stripplot(y=num_col[i], x='Classes', data=df)
    plt.tight_layout()

# observation
'''
1. 0 means not fire and 1 means fire
Places with
2. high temp has fire
3. lower RH has fire
4. rain <2 has fire
5. FFMC > 60 has fire
6. ISI > 2 has fire
'''
```
Out[113…
'\n1. 0 means not fire and 1 means fire\nPlaces with \n2. high temp has fire\n3. lower RH has fire\n4. rain <2 has fire\n5. FFMC > 60 has fire\n6. ISI > 2 has fire\n'



Graph Analysis [Visualization]

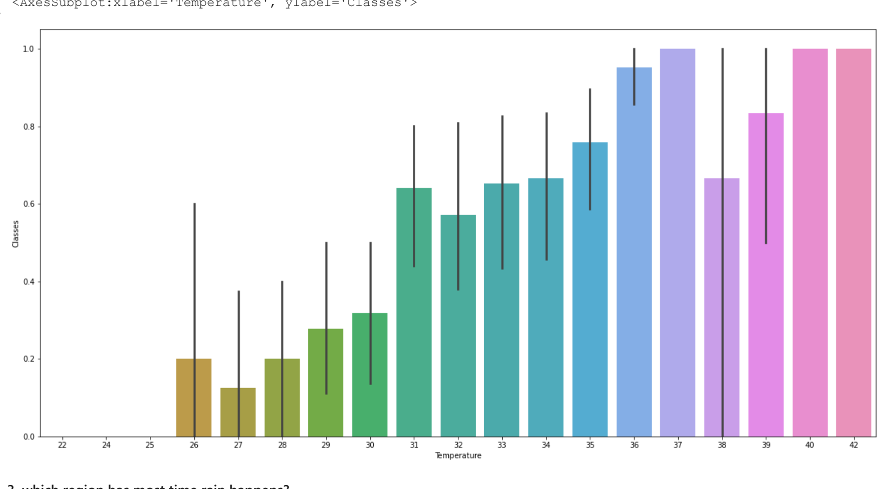
1. Temperature range which is most in common or most of the places?

In [114…
```
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.ylabel("count",weight="bold",fontsize=15)
plt.xlabel("temperature",weight="bold",fontsize=15)
plt.show()

# observation: temp occur most of the time in range 32.5 to 35
```



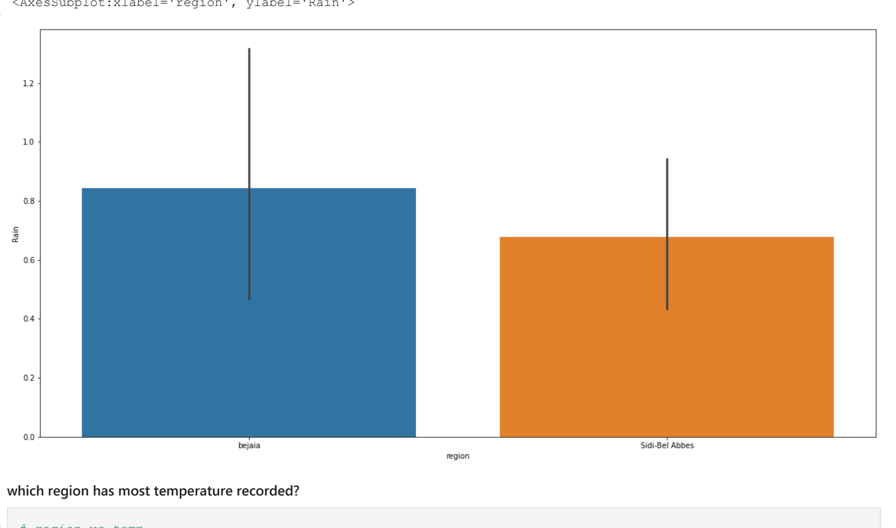
2. Highest Temperature attained

In [115…
```
# temp vs classes
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

sns.barplot(x="Temperature",y="Classes",data=df)

#observation: highest temp. is 40, 42,37
```
Out[115…
<AxesSubplot:xlabel='Temperature', ylabel='Classes'>



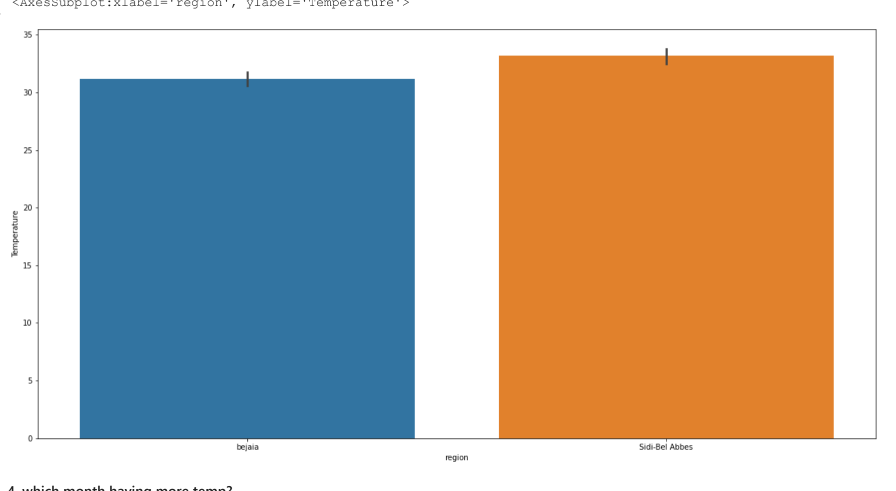
3. which region has most time rain happens?

In [116…
```
#region vs rain
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

sns.barplot(x="region",y="Rain",data=df)

#observation: Bejaiais the region has most of the time rain happen means,
# Sidi-Bel Abbes has less rainfall and it is more prone to catch fire
```
Out[116…
<AxesSubplot:xlabel='region', ylabel='Rain'>



which region has most temperature recorded?

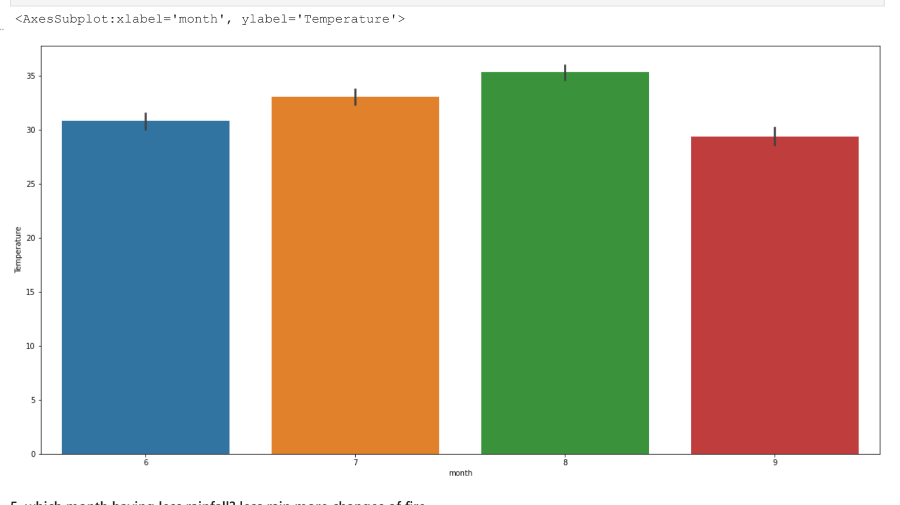
In [117…
```
# region vs temp
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)

sns.barplot(x="region",y="Temperature",data=df)

#observation: Bejaiais the region has less temp recorded
# Sidi-Bel Abbes has more temp and it is more prone to catch fire
```
Out[117…
<AxesSubplot:xlabel='region', ylabel='Temperature'>

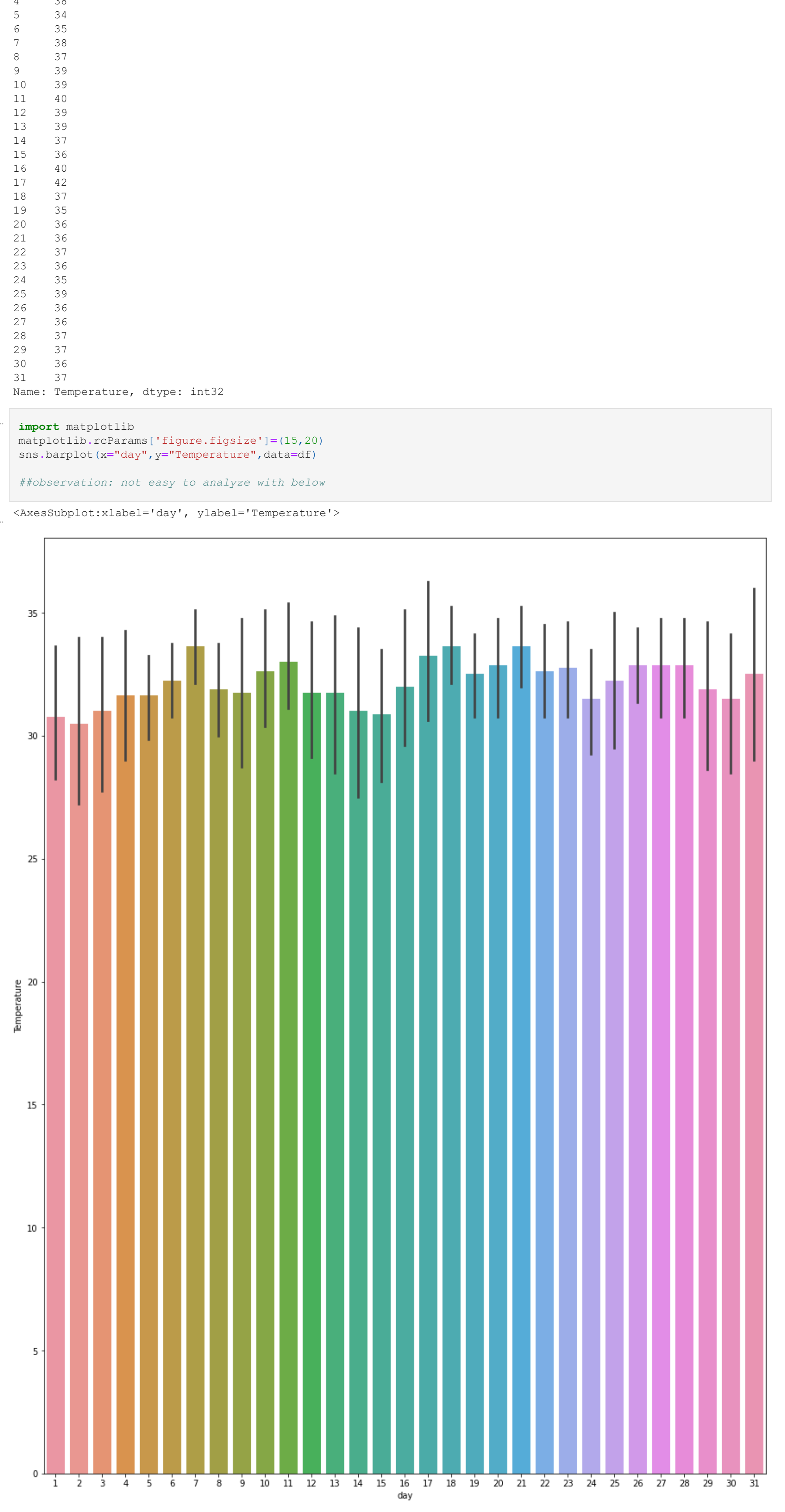


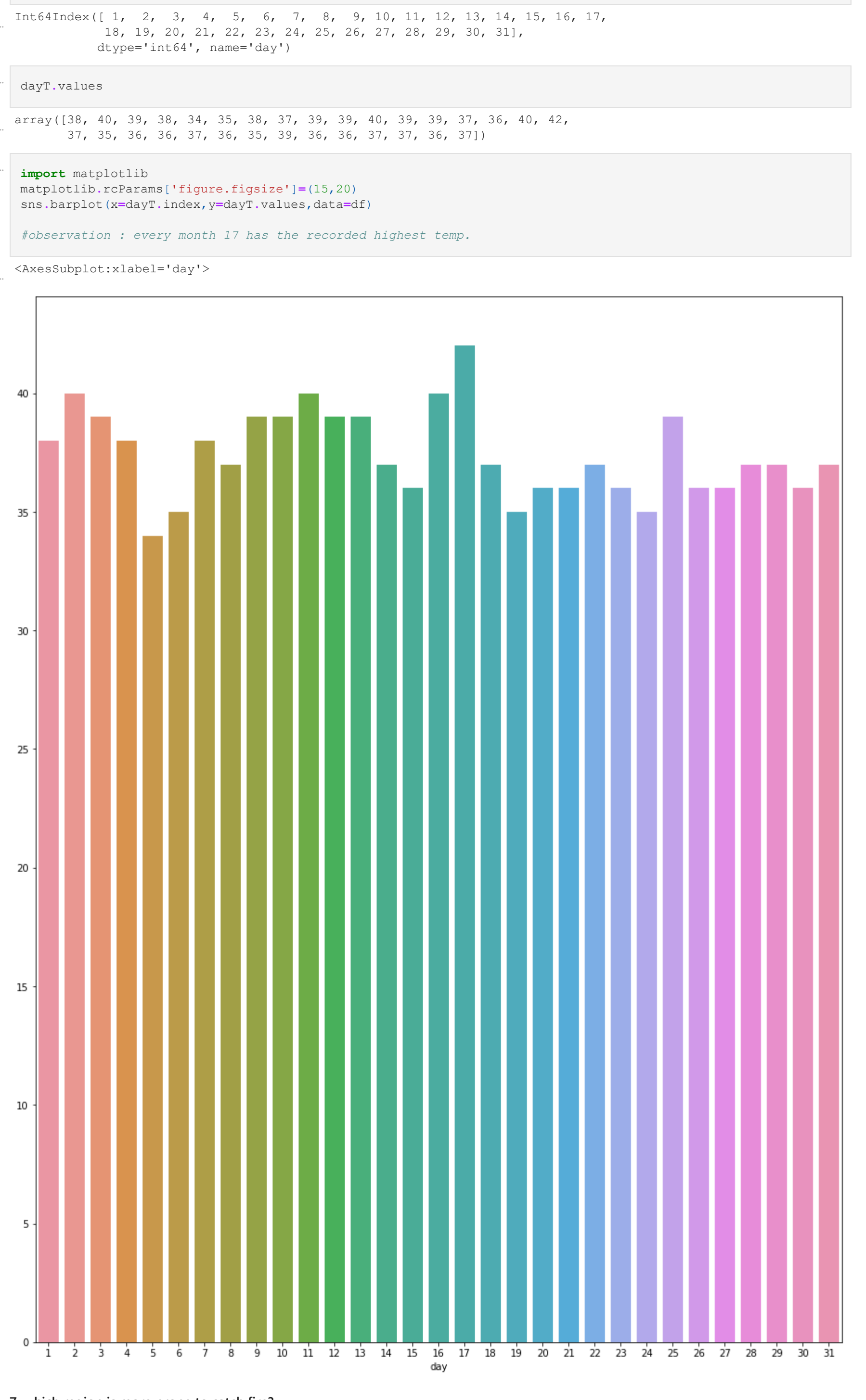
4. which month having more temp?

In [118…
```
# month vs temp
df.groupby('month')['Temperature'].max()
```
Out[118…
```
month
6    37
7    39
8    42
9    36
Name: Temperature, dtype: int32
```

In [119…
```
import matplotlib
matplotlib.rcParams['figure.figsize']=(20,10)
sns.barplot(x="month",y="Temperature",data=df)

##observation : Aug month has highest temperature so the chances of probability for getting forest catching fi
```
Out[119…
<AxesSubplot:xlabel='month', ylabel='Temperature'>







7. which region is more prone to catch fire?

In [128…
```
# region vs classes 0 means not fire and 1 means fire
df.groupby('region')['Classes'].value_counts()

## observation : Sidi-Bel Abbes  is more prone to catch fire
```
Out[128…
```
region          Classes
Sidi-Bel Abbes  1.0        79
                0.0        43
bejaia          0.0        63
                1.0        59
Name: Classes, dtype: int64
```

FINAL Report for EDA

In [ ]:
```
1. SIDI-BEL ABBES has less rainfall recorded so it is more prone to catch fire.
2. SIDI0BEL ABBES has high temp recorded so it is more prone to catch fire.
```

* Model Building steps *

In [233…
```
dataset=df
```

In [234…
```
dataset
```
Out[234…

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
0	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0	0	2012-06-01
1	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	0	0	2012-06-02
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0	0	2012-06-03
3	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	0	0	2012-06-04
4	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0	0	2012-06-05
...	...	...	...	...	...	...	...	...	...	...	...	...	...
239	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	1	1	2012-09-26
240	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	0	1	2012-09-27
241	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	0	1	2012-09-28
242	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	0	1	2012-09-29
243	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	0	1	2012-09-30

244 rows × 13 columns

In [ ]:
```
# adding new Feature named 'Date'
dataset['date']= pd.to_datetime(dataset[['day','month','year']])
dataset.drop(['day','month','year'],axis=1,inplace=True)
```

In [235…
```
dataset.head()
```
Out[235…

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

In [236…
```
dataset.tail()
```
Out[236…

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
239	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	1	1	2012-09-26
240	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	0	1	2012-09-27
241	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	0	1	2012-09-28
242	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	0	1	2012-09-29
243	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	0	1	2012-09-30

In [237…
```
dataset.iloc[165]
```
Out[237…
```
Temperature                     37
RH                              37
Ws                              18
Rain                           0.2
FFMC                          88.9
DMC                           12.9
DC                            14.6
ISI                            9.0
BUI                           12.5
FWI                           10.4
Classes                          1
region                           1
date           2012-07-14 00:00:00
Name: 165, dtype: object
```

In [238…
```
# setting those mismatch positions
dataset.at[165,'DC'] =14.6
dataset.at[165,'ISI'] = 9
dataset.at[165,'BUI'] = 12.5
dataset.at[165,'FWI'] = 10.4
```

In [239…
```
dataset.iloc[165]
```
Out[239…
```
Temperature                     37
RH                              37
Ws                              18
Rain                           0.2
FFMC                          88.9
DMC                           12.9
DC                            14.6
ISI                            9.0
BUI                           12.5
FWI                           10.4
Classes                          1
region                           1
date           2012-07-14 00:00:00
Name: 165, dtype: object
```

In [240]:
```
#encoding new feature named 'Region' in a dataset
dataset.loc[:122, 'region'] = 0
dataset.loc[122:, 'region'] = 1
```

In [241]:
```
dataset.iloc[165]
```

Out[241]:
```
Temperature                     37
RH                              37
Ws                              18
Rain                           0.2
FFMC                          88.9
DMC                           12.9
DC                            14.6
ISI                            9.0
BUI                           12.5
FWI                           10.4
Classes                          1
region                           1
date           2012-07-14 00:00:00
Name: 165, dtype: object
```

In [242]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Temperature  244 non-null    int32
 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      244 non-null    int32
 11  region       244 non-null    int32
 12  date         244 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(7), int32(5)
memory usage: 20.1 KB
```

In [ ]:

In [243]:
```
##changing datatype
#df['date']=df['date'].astype(int)
df['Temperature']=df['Temperature'].astype(int)
df['RH']=df['RH'].astype(int)
df['Rain']=df['Rain'].astype(float)
df['FFMC']=df['FFMC'].astype(float)
df['DMC']=df['DMC'].astype(float)
df['ISI']=df['ISI'].astype(float)
df['BUI']=df['BUI'].astype(float)
df['Ws']=df['Ws'].astype(int)
df['Classes']=df['Classes'].astype(int)
df['region']=df['region'].astype(int)
df['FWI']=df['FWI'].astype(float)
df['DC']=df['DC'].astype(float)
```

In [228]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Temperature  244 non-null    int32
 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      244 non-null    int32
 11  region       244 non-null    int32
 12  date         244 non-null    datetime64[ns]
dtypes: datetime64[ns](1), float64(7), int32(5)
memory usage: 20.1 KB
```

In [244]:
```
dataset.tail()
```

Out[244]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region	date
239	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	1	1	2012-09-26
240	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	0	1	2012-09-27
241	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	0	1	2012-09-28
242	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	0	1	2012-09-29
243	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	0	1	2012-09-30

1. Independent and dependent feature separation

In [245]:
```
# 1. Independent and dependent feature separation // X-> independent features //Y-> dependent feature
X=dataset.iloc[:,1:-1]
y=dataset.iloc[:,0]
```

In [246]:
```
X.head()
```

Out[246]:

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

In [247]:
```
y.head()
```

Out[247]:
```
0    29
1    29
2    26
3    25
4    27
Name: Temperature, dtype: int32
```

2. split the data into trian and test split dataset

In [248]:
```
from sklearn.model_selection import train_test_split
```

In [290]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

In [291]:
```
X_train.head()
```

Out[291]:

	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	region
114	54	11	0.5	73.7	7.9	30.4	1.2	9.6	0.7	0	0
65	65	13	0.0	86.8	11.1	29.7	5.2	11.5	6.1	1	0
132	42	21	0.0	90.6	18.2	30.5	13.4	18.0	16.7	1	1
207	40	18	0.0	92.1	56.3	157.5	14.3	59.5	31.1	1	1
162	56	15	2.9	74.8	7.1	9.5	1.6	6.8	0.8	0	1

In [292]:
```
X_train.shape
```

Out[292]:
```
(163, 11)
```

In [293]:
```
y_train.shape
```

Out[293]:
```
(163,)
```

In [294]:
```
X_test.shape
```

Out[294]:
```
(81, 11)
```

In [295]:
```
y_test.shape
```

Out[295]:
```
(81,)
```

3. Standardize or feature scaling the datasets

In [296]:
```
#only train data
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [297]:
```
scaler
```

Out[297]:
```
StandardScaler()
```

In [298]:
```
X_train=scaler.fit_transform(X_train)
```

In [299]:
```
X_test=scaler.transform(X_test)  #to avoid data leakage use transform
```

In [300]:
```
X_train
```

Out[300]:
```
array([[-0.60257784, -1.68484146, -0.17054229, ..., -0.8196431 ,
        -1.04390785, -0.99388373],
       [ 0.14460201, -0.93856657, -0.39436188, ..., -0.08219052,
         0.95793896, -0.99388373],
       [-1.41768313,  2.04653297, -0.39436188, ...,  1.36540157,
         0.95793896,  1.0061539 ],
       ...,
       [ 0.89178186,  0.5539832 ,  2.82864022, ..., -0.90158227,
        -1.04390785, -0.99388373],
       [-0.39880152,  0.18084575, -0.39436188, ...,  0.31384882,
         0.95793896,  1.0061539 ],
       [ 0.9597073 ,  2.04653297,  0.41138865, ..., -0.87426921,
        -1.04390785, -0.99388373]])
```

In [301]:
```
X_test
```

Out[301]:
```
array([[ 7.66765714e-02, -1.92291688e-01, -3.94361879e-01,
         6.76854493e-01, -3.05224430e-02,  3.01184828e-01,
         2.81509163e-01,  1.19661624e-01,  2.31909648e-01,
         9.57938964e-01, -9.93883735e-01],
       [-6.02577838e-01, -9.38566574e-01, -3.94361879e-01,
         7.79312965e-01, -3.70097033e-01, -3.74297004e-01,
         4.49765818e-01, -3.88384315e-01,  6.80312985e-02,
         9.57938964e-01, -9.93883735e-01],
       [-1.01013048e+00,  1.80845755e-01, -3.94361879e-01,
         7.38329576e-01, -5.28038264e-01, -6.31913198e-01,
         5.45912478e-01, -6.01214370e-01, -4.12209346e-02,
         9.57938964e-01,  1.00615390e+00],
[illegible]
         9.57938964e-01,  1.00615390e+00],
       [ 1.02763274e+00,  5.53983197e-01, -3.94361879e-01,
         2.26037218e-01, -7.88641789e-01, -6.17713880e-01,
        -4.39590788e-01, -7.45389569e-01, -6.83077804e-01,
        -1.04390785e+00, -9.93883735e-01],
       [ 1.16348363e+00,  1.80845755e-01, -2.15306207e-01,
        -1.16056743e+00, -8.83406707e-01, -4.43265118e-01,
        -8.72250758e-01, -7.86582483e-01, -8.60612683e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-9.42205043e-01, -1.68484146e+00, -3.94361879e-01,
         8.61279743e-01, -3.77993810e-01, -3.21556680e-01,
         5.45912478e-01, -3.60922372e-01,  1.36313944e-01,
         9.57938964e-01, -9.93883735e-01],
       [ 3.48378335e-01,  1.30025808e+00,  5.32773012e-02,
        -1.15373687e+00, -9.54480396e-01, -8.18532803e-01,
        -8.24177428e-01, -9.37623168e-01, -8.60612683e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-1.89316122e+00, -1.92291688e-01, -3.94361879e-01,
         1.16182459e+00,  6.64420291e-01, -1.26823179e-01,
         2.68517566e+00,  4.35473965e-01,  1.93897579e+00,
         9.57938964e-01,  1.00615390e+00],
       [ 1.16348363e+00,  1.80845755e-01,  4.11388646e-01,
        -2.00755747e+00, -9.22892090e-01, -8.36789069e-01,
        -1.01647075e+00, -9.30757682e-01, -9.01582270e-01,
        -1.04390785e+00, -9.93883735e-01],
       [ 1.44602012e-01, -5.65429131e-01, -3.94361879e-01,
         5.26582068e-01, -1.64772744e-01,  1.08479801e-01,
        -1.27114143e-01, -5.19755172e-02, -1.50473168e-01,
         9.57938964e-01, -9.93883735e-01],
       [ 8.75113043e-03, -1.92291688e-01, -3.94361879e-01,
         6.97346188e-01,  3.48537230e-01,  7.33249784e-01,
         3.29582493e-01,  5.38456250e-01,  4.77727173e-01,
         9.57938964e-01, -9.93883735e-01],
       [-1.07805592e+00,  9.27120640e-01,  2.29147320e+00,
         2.73851171e-01, -3.77993810e-01, -7.96219589e-01,
        -3.43444128e-01, -4.91366600e-01, -5.05542925e-01,
         9.57938964e-01,  1.00615390e+00],
       [-1.21390681e+00,  9.27120640e-01, -3.94361879e-01,
         9.36415955e-01,  3.24846000e-01, -5.17696424e-02,
         1.62756240e+00,  1.47123567e-01,  1.13324057e+00,
         9.57938964e-01,  1.00615390e+00],
       [-5.34652397e-01, -1.68484146e+00, -3.94361879e-01,
         7.51990706e-01,  1.51110317e-01,  1.81504864e-01,
         2.09399168e-01,  1.81450995e-01,  2.18253119e-01,
         9.57938964e-01, -9.93883735e-01],
       [-3.30876074e-01, -1.31170402e+00,  1.44095876e+00,
        -7.30241852e-01, -8.36024248e-01, -8.22589751e-01,
        -8.48214093e-01, -8.68968311e-01, -8.60612683e-01,
        -1.04390785e+00,  1.00615390e+00],
       [ 7.66765714e-02,  9.27120640e-01, -3.94361879e-01,
         6.83685058e-01,  2.53772312e-01,  4.63462746e-01,
         5.21875813e-01,  3.46222651e-01,  5.32353289e-01,
         9.57938964e-01, -9.93883735e-01],
       [ 7.66765714e-02,  5.53983197e-01, -3.94361879e-01,
         7.11007317e-01,  1.36726010e+00,  1.96250501e+00,
         5.45912478e-01,  1.69185784e+00,  1.22883628e+00,
         9.57938964e-01, -9.93883735e-01],
       [-5.34652397e-01,  1.80845755e-01, -3.49597961e-01,
         2.12376088e-01, -7.96538865e-01, -6.68425729e-01,
        -4.87664118e-01, -7.72851512e-01, -7.24047391e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-1.07805592e+00, -5.65429131e-01, -8.10144529e-02,
         4.84425331e-02, -5.91214876e-01, -8.14475855e-01,
        -6.55920773e-01, -6.69869227e-01, -7.92330037e-01,
        -1.04390785e+00, -9.93883735e-01],
       [ 7.55930981e-01,  1.30025808e+00, -3.49597961e-01,
        -6.76770683e-02, -8.67612554e-01, -3.39812946e-01,
        -5.83810778e-01, -7.52255055e-01, -7.92330037e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-1.14598137e+00, -9.38566574e-01, -2.60070125e-01,
         4.85598679e-01,  1.82698623e-01,  1.06451327e-01,
        -2.47297468e-01,  1.67720024e-01, -1.64129697e-01,
         9.57938964e-01,  1.00615390e+00],
       [-5.91743105e-02, -2.05797890e+00,  3.50009899e+00,
        -7.71225241e-01, -7.88641789e-01, -8.24618225e-01,
        -8.72250758e-01, -8.34640883e-01, -8.60612683e-01,
        -1.04390785e+00,  1.00615390e+00],
       [ 1.63896171e+00,  5.03163251e+00, -1.70542289e-01,
        -2.11001594e+00, -8.75509630e-01, -8.32732121e-01,
        -9.92434083e-01, -9.03295739e-01, -8.87925741e-01,
        -1.04390785e+00,  1.00615390e+00],
       [-5.34652397e-01, -9.38566574e-01, -2.60070125e-01,
         3.83140208e-01,  8.00366283e-02,  8.83356857e-01,
        -1.99224138e-01,  3.73684594e-01, -5.48774637e-02,
         9.57938964e-01, -9.93883735e-01],
       [-6.02577838e-01, -1.68484146e+00, -3.49597961e-01,
         4.71937550e-01, -4.88552881e-01, -4.59492910e-01,
        -3.43444128e-01, -4.98232085e-01, -4.91886396e-01,
         9.57938964e-01,  1.00615390e+00],
       [-8.74279602e-01,  1.30025808e+00, -1.25778371e-01,
         6.89342274e-02, -3.14817198e-01, -1.53193341e-01,
        -5.11700783e-01, -2.51074601e-01, -5.32855984e-01,
        -1.04390785e+00, -9.93883735e-01],
       [ 1.77481259e+00,  1.80845755e-01, -1.25778371e-01,
        -2.69744451e+00, -1.06503947e+00, -8.34760595e-01,
        -1.08858074e+00, -1.02687448e+00, -9.15238799e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-1.55353401e+00, -9.38566574e-01, -3.94361879e-01,
         1.03204386e+00,  6.09140755e-01,  1.32821489e-01,
         1.50737908e+00,  3.94281051e-01,  1.22883628e+00,
         9.57938964e-01,  1.00615390e+00],
       [ 4.16303776e-01,  1.80845755e-01, -3.94361879e-01,
         6.63193364e-01,  7.21395518e-02, -7.14321508e-03,
         2.33435833e-01,  4.41412821e-02,  1.77283532e-01,
         9.57938964e-01, -9.93883735e-01],
       [ 1.23140907e+00,  5.53983197e-01, -3.94361879e-01,
         5.33412633e-01,  5.14375837e-01, -1.34937075e-01,
         4.11425125e-02,  2.98164252e-01,  1.36313944e-01,
         9.57938964e-01,  1.00615390e+00],
       [ 3.48378335e-01, -5.65429131e-01,  2.32332973e-01,
        -6.96089029e-01, -5.43832417e-01, -8.06361959e-01,
        -8.24177428e-01, -6.28676313e-01, -8.33299625e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-1.96108666e+00,  1.80845755e-01, -3.04834043e-01,
         7.86143530e-01,  1.82698623e-01, -7.81398040e-02,
         7.14169133e-01,  6.47377390e-02,  5.18696760e-01,
         9.57938964e-01,  1.00615390e+00],
       [-8.06354161e-01, -9.38566574e-01, -3.94361879e-01,
         8.13465789e-01,  1.11624934e-01,  2.53123684e-02,
         5.69949143e-01,  8.53341960e-02,  4.23101056e-01,
         9.57938964e-01,  1.00615390e+00],
       [-1.75731033e+00,  1.80845755e-01, -3.94361879e-01,
         1.05253556e+00,  3.68900060e+00,  2.39862691e+00,
         2.06022237e+00,  3.25718857e+00,  3.22268953e+00,
         9.57938964e-01,  1.00615390e+00],
       [ 8.23856422e-01,  1.80845755e-01, -3.94361879e-01,
         2.73851171e-01, -8.83406707e-01, -5.06147812e-01,
        -4.15554123e-01, -7.86582483e-01, -6.83077804e-01,
         9.57938964e-01, -9.93883735e-01],
       [ 1.36725995e+00,  5.53983197e-01, -3.94361879e-01,
         5.26582068e-01,  3.80125536e-01, -3.21556680e-01,
         4.11425125e-02,  1.81450995e-01,  8.16878277e-02,
         9.57938964e-01,  1.00615390e+00],
       [ 7.66765714e-02, -5.65429131e-01, -3.94361879e-01,
         8.27126919e-01,  2.04640868e+00,  2.48179435e+00,
         1.07471911e+00,  2.35780995e+00,  1.93897579e+00,
         9.57938964e-01, -9.93883735e-01],
       [-6.70503279e-01,  1.30025808e+00, -3.94361879e-01,
         8.47618613e-01,  1.98492776e-01,  1.00709377e+00,
         1.31508576e+00,  5.04128821e-01,  1.17421016e+00,
         9.57938964e-01, -9.93883735e-01],
       [ 1.02763274e+00,  9.27120640e-01, -3.94361879e-01,
         6.15379410e-01,  2.44915958e+00,  2.87328982e+00,
         4.11425125e-02,  2.78347006e+00,  9.55705693e-01,
         9.57938964e-01, -9.93883735e-01],
       [-2.30071386e+00,  9.27120640e-01, -3.94361879e-01,
         1.16865516e+00,  1.97533499e+00,  6.41968455e-01,
         3.35820228e+00,  1.57514458e+00,  3.18171994e+00,
         9.57938964e-01,  1.00615390e+00],
       [-6.02577838e-01,  9.27120640e-01, -3.49597961e-01,
         1.98714958e-01, -8.12333018e-01, -6.84653521e-01,
        -6.79957438e-01, -7.86582483e-01, -8.19643095e-01,
        -1.04390785e+00,  1.00615390e+00],
       [-3.30876074e-01,  9.27120640e-01,  5.90444318e-01,
        -8.94175407e-01, -8.99200860e-01, -8.20561277e-01,
        -8.00140763e-01, -9.10161225e-01, -8.46956154e-01,
        -1.04390785e+00,  1.00615390e+00],
       [-2.02901210e+00, -9.38566574e-01, -3.49597961e-01,
         9.43246520e-01,  8.85538433e-01,  5.85171184e-01,
         1.07471911e+00,  7.99344705e-01,  1.18786669e+00,
         9.57938964e-01,  1.00615390e+00],
       [-3.30876074e-01,  9.27120640e-01, -3.94361879e-01,
         7.65651836e-01, -7.79049021e-02, -2.46503143e-01,
         8.34352458e-01, -1.68688773e-01,  4.36757585e-01,
         9.57938964e-01,  1.00615390e+00],
       [ 1.44602012e-01,  1.30025808e+00, -1.25778371e-01,
        -5.79969427e-01, -7.17568100e-01, -6.84653521e-01,
        -7.28030768e-01, -7.38524083e-01, -8.19643095e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-4.66726956e-01,  1.80845755e-01, -3.94361879e-01,
         8.27126919e-01,  7.27596903e-01,  1.65277072e-01,
         8.82425788e-01,  4.97263336e-01,  8.87423047e-01,
         9.57938964e-01,  1.00615390e+00],
       [ 6.20080099e-01, -5.65429131e-01, -3.94361879e-01,
         5.06090374e-01, -4.96449958e-01, -4.81806124e-01,
        -1.75187473e-01, -5.11963057e-01, -3.82634163e-01,
         9.57938964e-01,  1.00615390e+00],
       [-1.95025192e-01,  9.27120640e-01, -2.60070125e-01,
         2.11202739e-02, -2.59537662e-01, -3.95987986e-02,
        -5.59774113e-01, -1.68688773e-01, -5.60169042e-01,
        -1.04390785e+00, -9.93883735e-01],
       [ 1.57103627e+00,  2.04653297e+00,  1.66477835e+00,
        -2.45154418e+00, -1.04924531e+00, -8.40846016e-01,
        -1.06454408e+00, -1.01314351e+00, -9.15238799e-01,
        -1.04390785e+00, -9.93883735e-01],
       [-2.62950633e-01,  5.53983197e-01, -3.94361879e-01,
         7.24668447e-01,  1.68600161e-02,  1.63248598e-01,
         5.69949143e-01,  9.21996817e-02,  4.36757585e-01,
         9.57938964e-01, -9.93883735e-01],
       [ 1.43518539e+00,  9.27120640e-01, -3.94361879e-01,
         4.78768115e-01, -8.58019786e-02,  7.05610270e-03,
        -6.93081752e-03, -3.82445459e-02, -5.48774637e-02,
         9.57938964e-01, -9.93883735e-01],
       [-1.68938489e+00, -1.92291688e-01, -3.94361879e-01,
         1.04570499e+00,  2.89929295e+00,  2.00307449e+00,
         1.84389239e+00,  2.63242938e+00,  2.75836754e+00,
         9.57938964e-01,  1.00615390e+00],
       [-1.75731033e+00,  9.27120640e-01, -3.04834043e-01,
         8.27126919e-01, -1.33184438e-01, -6.96824365e-01,
         1.07471911e+00, -2.78536544e-01,  5.05040231e-01,
         9.57938964e-01,  1.00615390e+00],
       [-4.66726956e-01,  5.53983197e-01, -3.49597961e-01,
         5.40243198e-01, -3.85890886e-01, -4.39208171e-01,
         4.11425125e-02, -4.29577229e-01, -2.05099284e-01,
         9.57938964e-01,  1.00615390e+00],
       [ 8.23856422e-01,  1.80845755e-01, -3.94361879e-01,
         3.69479078e-01, -8.04435942e-01, -5.00062390e-01,
        -2.95370798e-01, -7.24793112e-01, -5.73825571e-01,
         9.57938964e-01,  1.00615390e+00],
       [-5.91743105e-02,  1.30025808e+00, -3.94361879e-01,
         8.61279743e-01,  6.80214444e-01,  1.45944346e+00,
         1.24297576e+00,  1.01217476e+00,  1.43368421e+00,
         9.57938964e-01, -9.93883735e-01],
       [-8.74279602e-01, -5.65429131e-01, -3.94361879e-01,
         8.13465789e-01,  6.56523214e-01,  8.89442279e-01,
         6.42059138e-01,  8.06210191e-01,  8.46453459e-01,
         9.57938964e-01, -9.93883735e-01],
       [-1.96108666e+00,  5.53983197e-01, -3.94361879e-01,
         1.05253556e+00,  7.11802750e-01,  9.80723608e-01,
         2.22847903e+00,  8.81730533e-01,  2.03457149e+00,
         9.57938964e-01,  1.00615390e+00],
       [ 1.23140907e+00, -1.92291688e-01, -3.94361879e-01,
         5.33412633e-01, -1.56875667e-01, -1.51164867e-01,
        -5.50041475e-02, -1.54957802e-01, -1.36816639e-01,
         9.57938964e-01,  1.00615390e+00],
       [-2.62950633e-01,  1.80845755e-01, -3.94361879e-01,
         7.72482400e-01,  3.88022612e-01, -3.55418507e-02,
         6.90132468e-01,  2.02047452e-01,  5.73322877e-01,
         9.57938964e-01,  1.00615390e+00],
       [ 2.80452894e-01, -5.65429131e-01, -3.94361879e-01,
         3.96801337e-01, -6.93876871e-01, -5.42660343e-01,
        -3.43444128e-01, -6.56138256e-01, -5.73825571e-01,
         9.57938964e-01, -9.93883735e-01],
       [-1.01013048e+00, -9.38566574e-01, -3.94361879e-01,
         9.22754826e-01,  6.01243679e-01,  1.20791269e+00,
         1.00260911e+00,  8.81730533e-01,  1.17421016e+00,
         9.57938964e-01, -9.93883735e-01],
       [ 7.66765714e-02, -5.65429131e-01, -3.94361879e-01,
         1.78223264e-01, -7.41259330e-01, -6.80596573e-01,
        -5.59774113e-01, -7.52255055e-01, -7.78673508e-01,
        -1.04390785e+00,  1.00615390e+00],
       [-5.34652397e-01, -1.92291688e-01, -3.94361879e-01,
         8.40788048e-01,  4.98581684e-01, -1.14652335e-01,
         8.34352458e-01,  2.91298766e-01,  7.23544697e-01,
         9.57938964e-01,  1.00615390e+00]])
```

A. Linear Regression

A.1 Model Training

In [302]:
```
from sklearn.linear_model import LinearRegression
regression=LinearRegression()
regression
```

Out[302]:
```
LinearRegression()
```

In [303]:
```
regression.fit(X_train,y_train)  #1. to model a data, only training data is needed
```

Out[303]:
```
LinearRegression()
```

coefficients and the intercept

In [304]:
```
## print the coefficients and the intercept
print(regression.coef_)
```

```
[-1.05126674 -0.48159084  0.11938767  1.82417191  0.94467874  0.67540664
  0.17325427 -1.25422021  0.0537007  -0.23927078 -0.00511072]
```

In [305]:
```
print(regression.intercept_)
```

```
31.98159509202454
```

A.2 Prediction

In [306]:
```
reg_pred=regression.predict(X_test)
reg_pred
```

Out[306]:
```
[illegible]
```

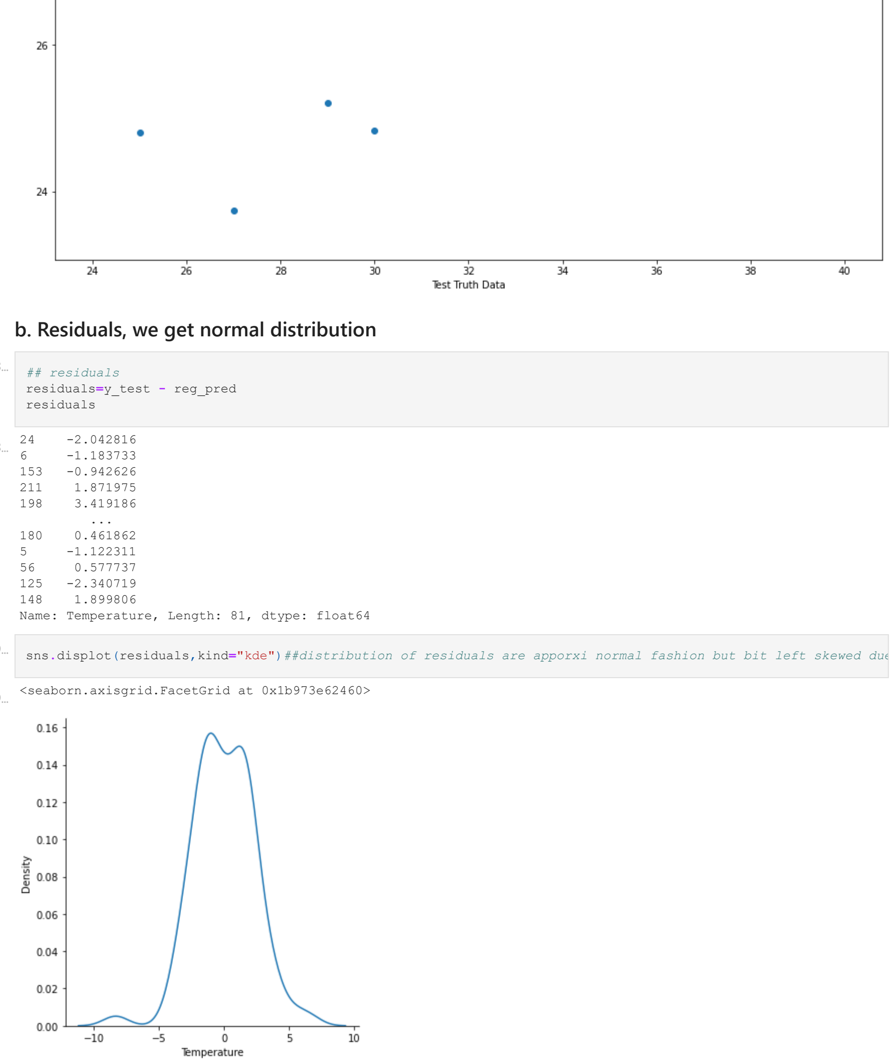
A.3 Assumption

We used to check Model is good or not

a. linear relation between Y_test and reg_pred

In [307]:
```
plt.scatter(y_test,reg_pred)   #reg_pred is prediction of X_test, also if u get a linear manner then it is good
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[307]:
```
Text(0, 0.5, 'Test Predicted Data')
```



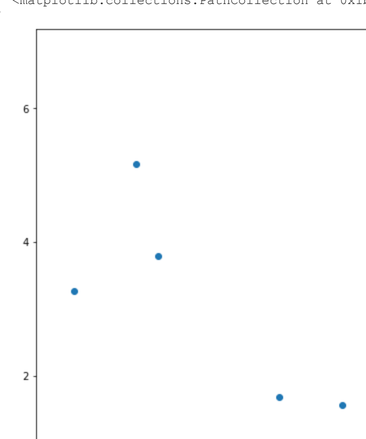
b. Residuals, we get normal distribution

In [308]:
```
## residuals
residuals=y_test - reg_pred
residuals
```

Out[308]:
```
24    -2.042816
6     -1.183733
153   -0.942626
211    1.871975
198    3.419186
         ...
180    0.461862
5     -1.122311
56     0.577737
125   -2.340719
148    1.899806
Name: Temperature, Length: 81, dtype: float64
```

In [309]:
```
sns.displot(residuals,kind="kde")##distribution of residuals are apporxi normal fashion but bit left skewed due
```

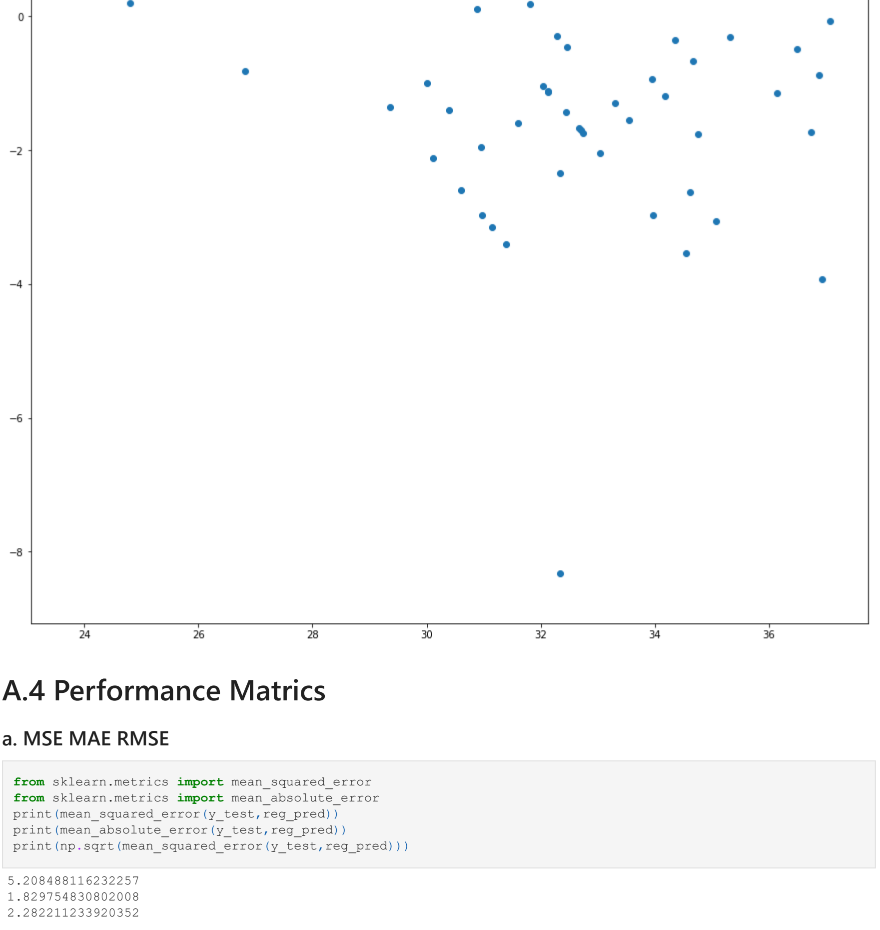
Out[309]:
```
<seaborn.axisgrid.FacetGrid at 0x1b973e62460>
```



c. Uniform Distribution

In [310]:
```
## SCatter plot with predictions and residual
##uniform distribution
plt.scatter(reg_pred,residuals)
```

Out[310]:
```
<matplotlib.collections.PathCollection at 0x1b97474ae50>
```



A.4 Performance Matrics

a. MSE MAE RMSE

In [311]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,reg_pred))
print(mean_absolute_error(y_test,reg_pred))
print(np.sqrt(mean_squared_error(y_test,reg_pred)))
```

```
5.208488116232257
1.829754830802008
2.282211233920353
```

b. R square and adjusted R square¶

In [312]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(score)
```

```
0.5150717960749278
```

In [313]:
```
## Adjusted R square
#display adjusted R-squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[313]:
```
0.437764401246293
```

B. Ridge Regression

B.1 Model Training

In [318]:
```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=1.0)
ridge_reg
```

Out[318]:
```
Ridge()
```

a. to train a model

In [319]:
```
ridge_reg.fit(X_train,y_train)
```

Out[319]:
```
Ridge()
```

b. Coeficient and intercept

In [321]:
```
print(ridge_reg.coef_)
```

```
[-1.06044036 -0.48245502  0.10285448  1.76950837  0.44681754  0.43522539
  0.21730663 -0.48758237 -0.0148011  -0.22972883  0.01004977]
```

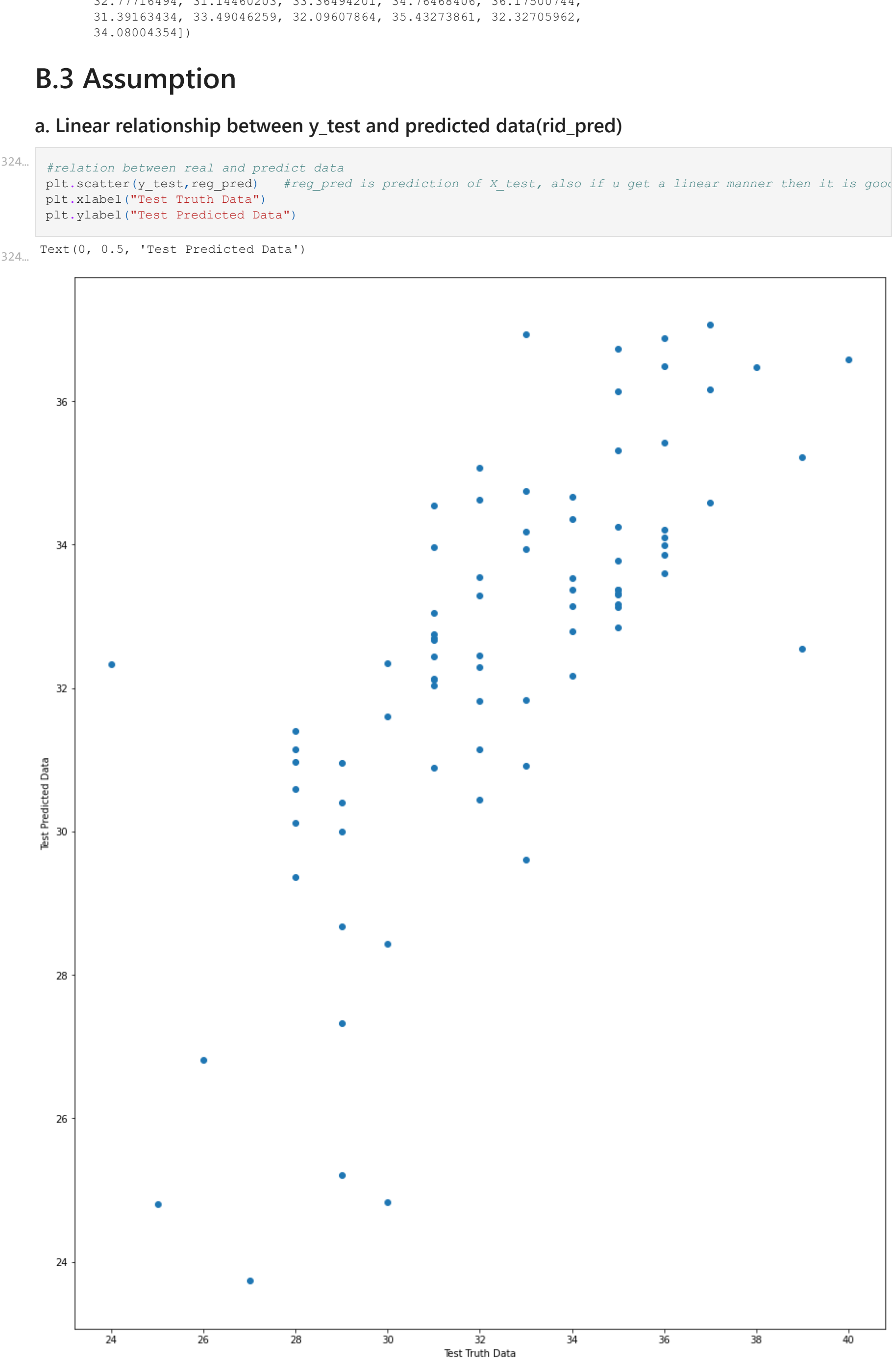
In [322]:
```
print(ridge_reg.intercept_)
```

```
31.98159509202454
```

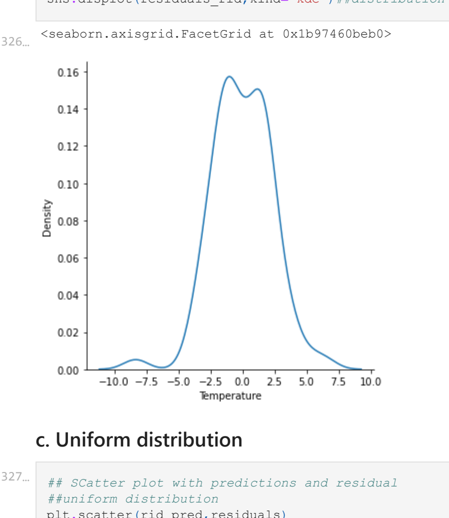
B.2 Prediction

B.3 Assumption

a. Linear relationship between y_test and predicted data(rid_pred)



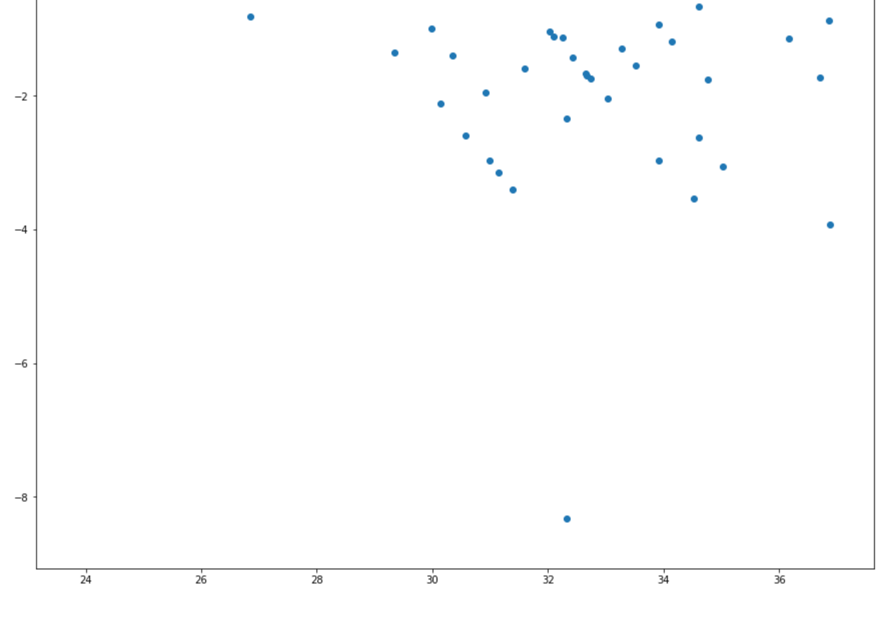
b. Residual we get normal distribution



c. Uniform distribution

```
In [327]: ## SCatter plot with predictions and residual
##uniform distribution
plt.scatter(rid_pred,residuals)
```

```
Out[327]: <matplotlib.collections.PathCollection at 0x1b974ac25e0>
```



B.4 Performance Matrics

a. MSE MAE RMSE

```
In [328]: from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,rid_pred))
print(mean_absolute_error(y_test,rid_pred))
print(np.sqrt(mean_squared_error(y_test,rid_pred)))
```

```
5.160557474677581
1.820352374126694
2.271686042277317
```

b. R square and adjusted R square

```
In [329]: ##R square
from sklearn.metrics import r2_score
score_rid=r2_score(y_test,rid_pred)
print(score_rid)
```

```
0.5195343040817426
```

```
In [330]: ## Adjusted R square
#display adjusted R-squared
1 - (1-score_rid)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
Out[330]: 0.44293832357303486
```

C. Lasso Regression

C.1 Model Training

```
In [331]: from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha=0.1)
lasso_reg
```

```
Out[331]: Lasso(alpha=0.1)
```

a. to train the model

```
In [332]: lasso_reg.fit(X_train,y_train) # to train a model , using training data
```

```
Out[332]: Lasso(alpha=0.1)
```

b. coeficients and intercept

```
In [334]: print(lasso_reg.coef_)
```

```
[-1.1147147  -0.33914294 -0.          1.61403791  0.21606476  0.08233736
  0.          0.          0.          0.          0.        ]
```

```
In [335]: print(lasso_reg.intercept_)
```

```
31.98159509202454
```

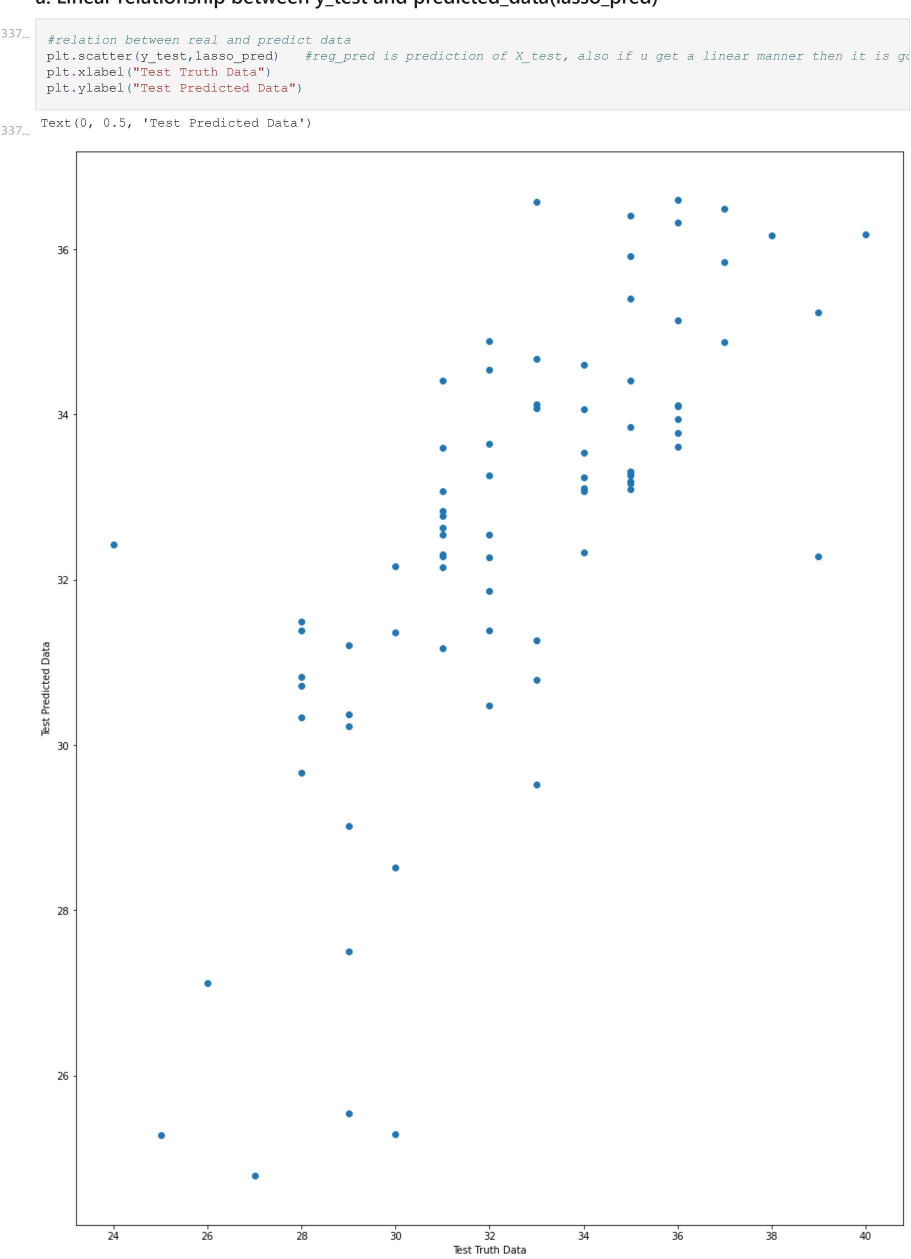
C.2 Prediction

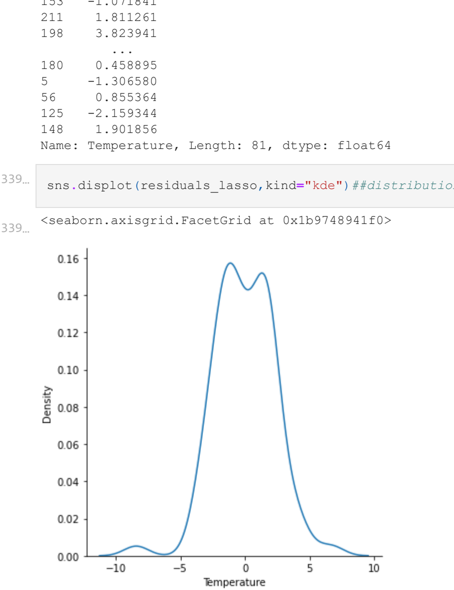
```
In [336]: lasso_pred=lasso_reg.predict(X_test)
lasso_pred
```

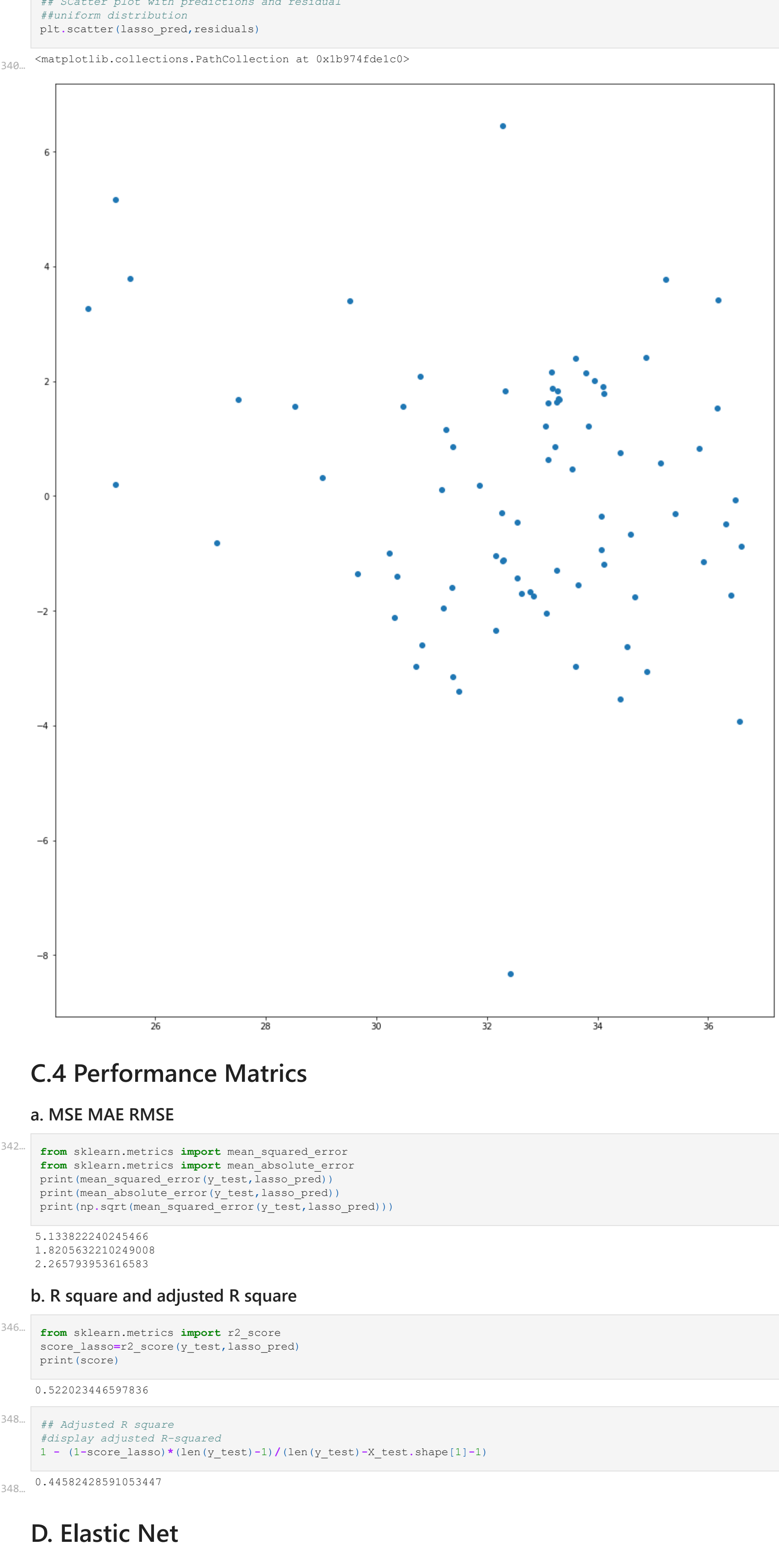
```
Out[336]: array([33.07200971, 34.11866287, 34.07184123, 33.1887394 , 36.17605922,
       32.28643412, 35.23145847, 27.49903808, 30.71811796, 29.51910213,
       29.66280233, 33.10282481, 33.59795808, 33.1012979 , 34.11387509,
       32.32629507, 36.48949003, 25.54647262, 32.27086081, 33.64839285,
       30.79177229, 28.52274007, 34.88527787, 29.01647885, 36.16548855,
       27.11473292, 32.83542018, 33.29827809, 33.16366987, 34.59766448,
       34.41031781, 31.3682794 , 32.77818132, 33.31284006, 32.63189009,
       33.25846677, 30.37330122, 34.40935992, 31.26242522, 24.7848003 ,
       33.60431738, 33.84303215, 32.54582209, 25.28922568, 35.83995845,
       32.54162048, 31.38202216, 30.47760313, 35.40822102, 34.53792514,
       36.57255409, 31.21135591, 31.17518987, 34.06939927, 33.78193891,
       32.2806532 , 36.59772021, 32.42771542, 30.33091963, 36.32362366,
       33.23466326, 30.23192538, 33.94636066, 32.15205706, 31.85931764,
       25.28344746, 33.27355719, 30.82213851, 36.40916201, 34.87493139,
       33.06641534, 31.38326684, 33.26385946, 34.67597918, 35.91314568,
       31.48874607, 33.54110493, 32.30658027, 35.14463579, 32.15934445,
       34.09814389])
```

C.3 Assumption

a. Linear relationship between y_test and predicted_data(lasso_pred)







C.4 Performance Matrics

a. MSE MAE RMSE

```
In [342]: from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,lasso_pred))
print(mean_absolute_error(y_test,lasso_pred))
print(np.sqrt(mean_squared_error(y_test,lasso_pred)))
```

```
5.133822240245466
1.8205632210249008
2.265793953616583
```

b. R square and adjusted R square

```
In [346]: from sklearn.metrics import r2_score
score_lasso=r2_score(y_test,lasso_pred)
print(score)
```

```
0.522023446597836
```

```
In [348]: ## Adjusted R square
#display adjusted R-squared
1 - (1-score_lasso)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
Out[348]: 0.44582428591053447
```

D. Elastic Net

D.1 Model training

```
In [349]: from sklearn.linear_model import ElasticNet
EN_reg=ElasticNet(random_state=0)
EN_reg
```

```
Out[349]: ElasticNet(random_state=0)
```

a. to train the model

```
In [350]: EN_reg.fit(X_train,y_train)  #1. to model a data, only training data is needed
```

```
Out[350]: ElasticNet(random_state=0)
```

b. coefficients and the intercept

```
In [352]: print(EN_reg.coef_)
```

```
[-0.68808933 -0.10544712 -0.00834786  0.85162206  0.10376148  0.
  0.23158765  0.02547021  0.15362153  0.07372069  0.        ]
```

```
In [353]: print(EN_reg.intercept_)
```

```
31.98159509202454
```

D.2 Prediction

```
In [354]: EN_pred=EN_reg.predict(X_test)
EN_pred
```

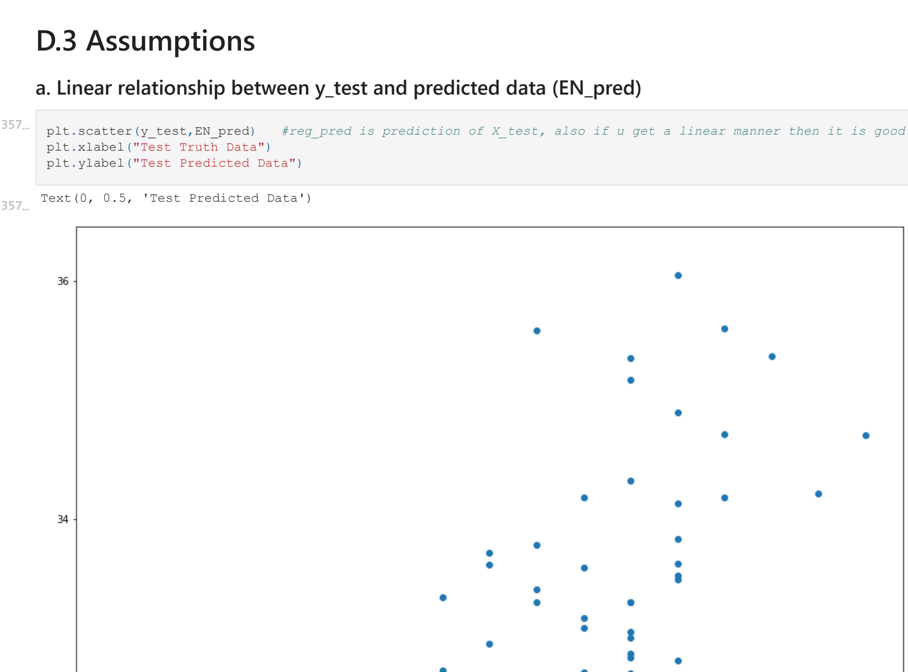
```
Out[354]: array([32.70014869, 33.29910099, 33.41026626, 32.61092932, 34.7047485 ,
       31.58360838, 34.21527053, 29.02563256, 30.73347022, 30.11039166,
       30.41277398, 32.24275851, 32.49689882, 32.11572726, 33.82943086,
       32.26602144, 35.60101706, 27.89264401, 32.12951491, 32.95265792,
       31.02735367, 29.65251053, 33.71392821, 30.09882338, 35.36763797,
       28.87850096, 32.39310489, 32.86323328, 32.70140428, 34.17896096,
       33.34379291, 31.19918174, 32.72501691, 33.05000354, 32.11073799,
       32.39596102, 30.82655691, 33.29618332, 31.03905163, 27.97071052,
       32.81078878, 32.83085009, 32.18944481, 27.83881794, 34.71347157,
       32.40467835, 31.69547324, 30.73847724, 34.32027173, 33.61283179,
       35.5799204 , 31.38989764, 31.57088417, 33.58592157, 33.61999323,
       32.2560815 , 36.04767586, 32.00594323, 30.83622131, 34.89715148,
       33.0854042 , 30.78738109, 33.49083492, 31.9555529 , 31.71431021,
       27.98432489, 32.99835463, 31.35802614, 35.34742765, 34.17614646,
       32.70498873, 31.52572086, 33.29706361, 33.77686044, 35.16781422,
       31.6287495 , 33.16854289, 32.00367904, 34.13279424, 31.72123893,
       33.52323673])
```

D.3 Assumptions

a. Linear relationship between y_test and predicted data (EN_pred)

```
In [357]: plt.scatter(y_test,EN_pred)   #reg_pred is prediction of X_test, also if u get a linear manner then it is good
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

```
Out[357]: Text(0, 0.5, 'Test Predicted Data')
```



b. Residuals, we get normal distribution

```
In [358]: ## residuals
residuals_EN=y_test-EN_pred
residuals_EN
```

```
Out[358]: 24    -1.700149
6     -0.299101
153   -0.410266
211    2.389071
198    5.295251
         ...
180    0.831457
5     -1.003679
56     1.867206
125   -1.721239
148    2.476763
Name: Temperature, Length: 81, dtype: float64
```

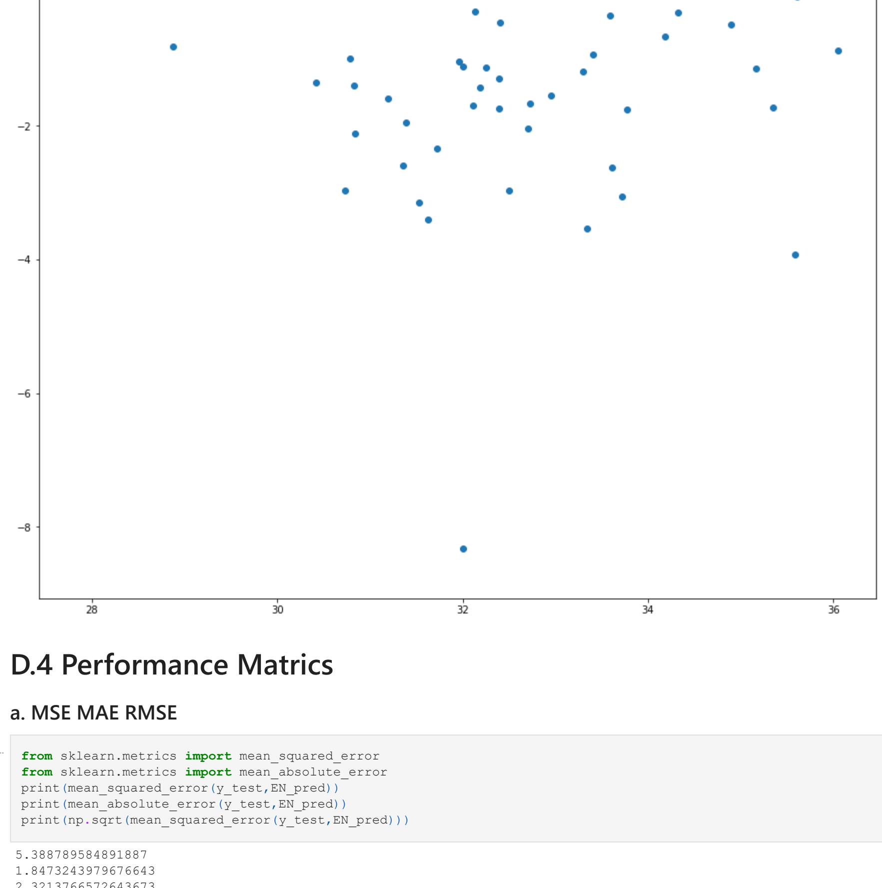
```
In [359]: sns.displot(residuals_EN,kind="kde")##distribution of residuals are apporxi normal fashion but bit right skewed
```

```
Out[359]: <seaborn.axisgrid.FacetGrid at 0x1b974fb30d0>
```

c. Uniform Distribution

```
In [360]: ## SCatter plot with predictions and residual
##uniform distribution
plt.scatter(EN_pred,residuals)
```

```
Out[360]: <matplotlib.collections.PathCollection at 0x1b9769b53a0>
```



D.4 Performance Matrics

a. MSE MAE RMSE

```
In [361]: from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,EN_pred))
print(mean_absolute_error(y_test,EN_pred))
print(np.sqrt(mean_squared_error(y_test,EN_pred)))
```

```
5.388789584891887
1.8473243979676643
2.3213766572643673
```

b. R square and adjusted R square

```
In [364]: from sklearn.metrics import r2_score
score_EN=r2_score(y_test,EN_pred)
print(score_EN)
```

```
0.4982851076135141
```

```
In [365]: ## Adjusted R square
#display adjusted R-squared
1 - (1-score_EN)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

```
Out[365]: 0.41830157404465407
```

.................THANKYOU.................

```
In [ ]:
```