Algerian forest fire Data Analysis

Life cycle of Machine learning Project

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

1) Problem statement.

- This dataset comprises information regarding forest fier in Algerian forests.
- If user can extract some information the cause of temperature in the forest.

Reading the data

In [1]:
```
### 2.1 Import Data and Required Packages
```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
from six.moves import urllib

warnings.filterwarnings("ignore")

%matplotlib inline
```

In [185]:
```
df = pd.read_csv('Algerian_forest_fires_dataset_UPDATE.csv',header=1)
df.head()
```

Out[185]:

	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

In [186]:
```
df
```

Out[186]:

	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

In [187]:
```
df.iloc[:126]
```

Out[187]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
123	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	
125	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	

126 rows × 14 columns

Remove extra Rows and Add a new column for region

In [188]:
```
for index in range(df.shape[0]):
    if index < 122:
        df.loc[index,'region'] = 0
    else:
        df.loc[index,'region'] = 1
```

In [189]:
```
df.iloc[:125]
```

Out[189]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
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	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
120	29	09	2012	26	80	16	1.8	47.4	2.9	7.7	0.3	3	0.1	
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
123	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	

125 rows × 15 columns

In [190]:
```
df.drop(df.index[122:124], inplace=True)
```

In [191]:
```
df.iloc[:125]
```

Out[191]:

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
120	29	09	2012	26	80	16	1.8	47.4	2.9	7.7	0.3	3	0.1	not fire
121	30	09	2012	25	78	14	1.4	45	1.9	7.5	0.2	2.4	0.1	not fire
124	01	06	2012	32	71	12	0.7	57.1	2.5	8.2	0.6	2.8	0.2	not fire
125	02	06	2012	30	73	13	4	55.7	2.7	7.8	0.6	2.9	0.2	not fire
126	03	06	2012	29	80	14	2	48.7	2.2	7.6	0.3	2.6	0.1	not fire

125 rows × 15 columns

In [192]:
```
df = df.rename(columns = {'Classes  ':'Classes','Rain ':'Rain'})
```

In [194]:
```
#Adding a column fire to indicate 0 for no fire and 1 for fire
df['fire'] = pd.factorize(df.Classes)[0]
```

In [195]:
```
#Cleaned Data Frame
df
```

Out[195]:

	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

244 rows × 16 columns

Basic Details About Data Set

Show top 5 records

In [196]:
```
df.head()
```

Out[196]:

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

Shape of the dataset

In [14]:
```
1 df.tail()
```

Out[14]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classe
241	26	09	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fir
242	27	09	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fir
243	28	09	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fir
244	29	09	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fir
245	30	09	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fir

In [15]:
```
1 df.shape
```

Out[15]:
```
(244, 16)
```

In [ ]:
```
1
```

Summery of the dataset

In [143]:
```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 244 entries, 0 to 245
Data columns (total 16 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      243 non-null    object 
 14  region       244 non-null    float64
 15  fire         244 non-null    int64  
dtypes: float64(1), int64(1), object(14)
memory usage: 32.4+ KB
```

Observation: As a lot of relevant columns are in Object type, we need to convert the type to do basic opearation.

Action:Converting desired columns into float

In [197]:
```
df.columns
```

Out[197]:
```
Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain', 'FFMC',
       'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes', 'region', 'fire'],
      dtype='object')
```

In [198]:
```
df = df.astype({'day':'int','month':'int','year':'int','Temperature':'int','
```

In [199]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 244 entries, 0 to 245
Data columns (total 16 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      243 non-null    object 
 14  region       244 non-null    float64
 15  fire         244 non-null    int64  
dtypes: float64(6), int32(6), int64(1), object(3)
memory usage: 26.7+ KB
```

In [200]:
```
df.describe()
```

Out[200]:

	day	month	year	Temperature	RH	Ws	Rain	FFM
count	244.000000	244.000000	244.0	244.000000	244.000000	244.000000	244.000000	244.00000
mean	15.754098	7.500000	2012.0	32.172131	61.938525	15.504098	0.760656	77.88770
std	8.825059	1.112961	0.0	3.633843	14.884200	2.810178	1.999406	14.33757
min	1.000000	6.000000	2012.0	22.000000	21.000000	6.000000	0.000000	28.60000
25%	8.000000	7.000000	2012.0	30.000000	52.000000	14.000000	0.000000	72.07500
50%	16.000000	7.500000	2012.0	32.000000	63.000000	15.000000	0.000000	83.50000
75%	23.000000	8.000000	2012.0	35.000000	73.250000	17.000000	0.500000	88.30000
max	31.000000	9.000000	2012.0	42.000000	90.000000	29.000000	16.800000	96.00000

In [201]:
```
#Dropping Year and Classes are there are no use of these two columns
#Also dropping fire column for now as we will concentrate on the cause of te
#Dropping Object type columns
df = df.drop(['year','Classes','DC','FWI','region'],axis=1)
```

In [202]:
```
df.head()
```

Out[202]:

	day	month	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	BUI	fire
0	1	6	29	57	18	0.0	65.7	3.4	1.3	3.4	0
1	2	6	29	61	13	1.3	64.4	4.1	1.0	3.9	0
2	3	6	26	82	22	13.1	47.1	2.5	0.3	2.7	0
3	4	6	25	89	13	2.5	28.6	1.3	0.0	1.7	0
4	5	6	27	77	16	0.0	64.8	3.0	1.2	3.9	0

Exploring Data

In [203]:
```
# define numerical & categorical columns
numeric_features = [feature for feature in df.columns if df[feature].dtype !
categorical_features = [feature for feature in df.columns if df[feature].dty

# print columns
print('We have {} numerical features : {}'.format(len(numeric_features), num
print('\nWe have {} categorical features : {}'.format(len(categorical_featur
```

```
We have 11 numerical features : ['day', 'month', 'Temperature', ' RH', ' Ws',
'Rain', 'FFMC', 'DMC', 'ISI', 'BUI', 'fire']

We have 0 categorical features : []
```

Attribute Information

1. Date : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012) Weather data observations
2. Temp : temperature noon (temperature max) in Celsius degrees: 22 to 42
3. RH : Relative Humidity in %: 21 to 90
4. Ws :Wind speed in km/h: 6 to 29
5. Rain: total day in mm: 0 to 16.8 FWI Components
6. Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5
7. Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9
8. Drought Code (DC) index from the FWI system: 7 to 220.4
9. Initial Spread Index (ISI) index from the FWI system: 0 to 18.5
10. Buildup Index (BUI) index from the FWI system: 1.1 to 68
11. Fire Weather Index (FWI) Index: 0 to 31.1
12. Classes: two classes, namely Fire and not Fire
13. Region: 0 Signifies Bejaia region and 1 signifies Sidi Bel-abbes region

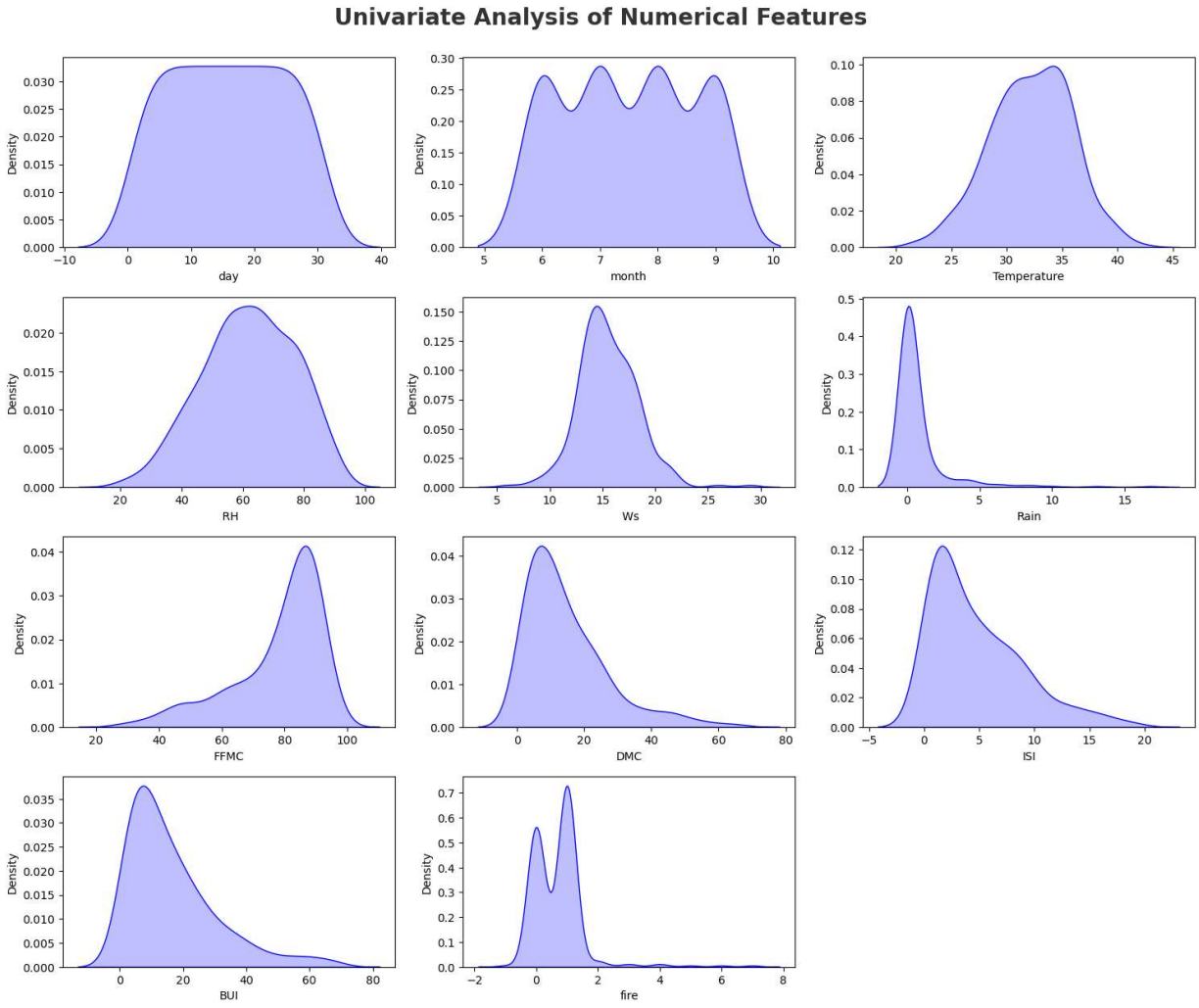
14. Fire: 0 indicstes fire and 1 indicates no fire

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.

In [204]:
```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontw

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



Report

- Rain, DMC, BUI, ISI are right skewed and postively skewed.
- FFMC is left skewed

- Outliers in BUI and DMC power.

Multivariate Analysis

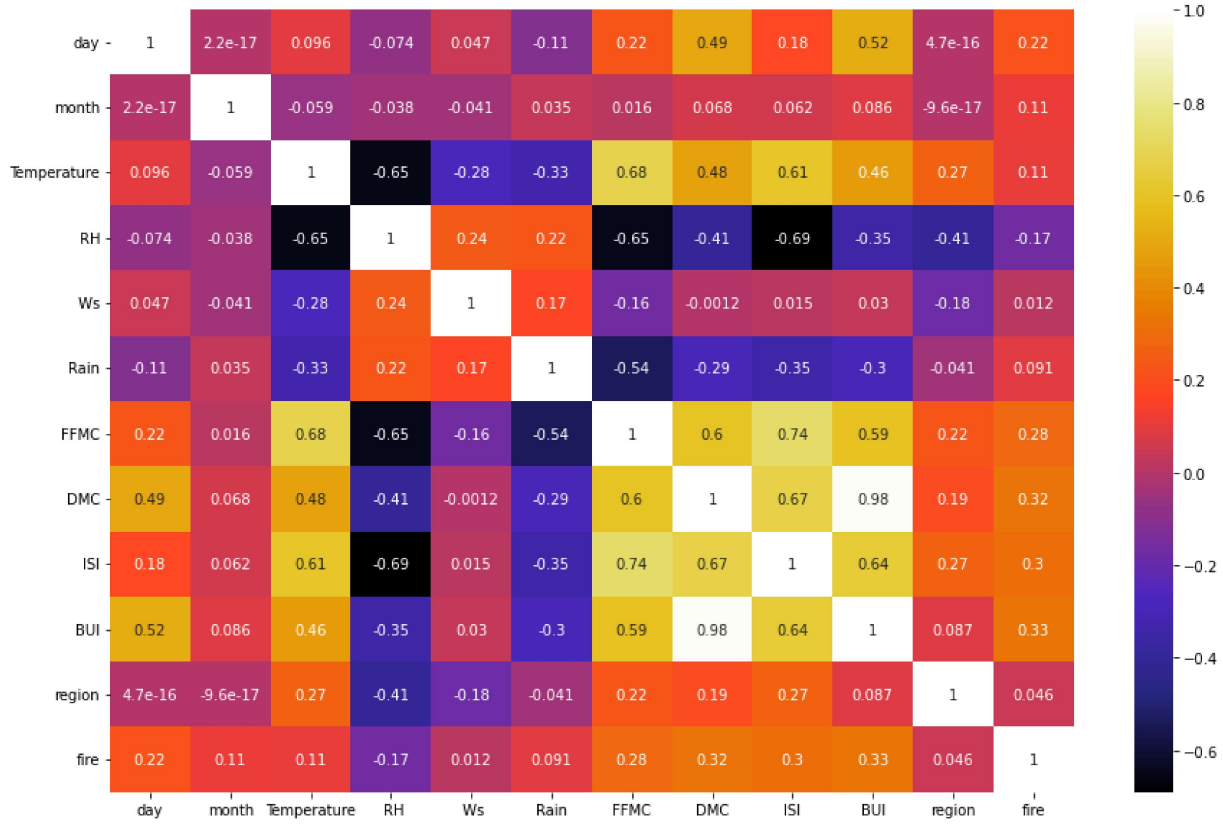
Check Multicollinearity in Numerical features

In [205]:
```
df[(list(df.columns)[1:])].corr()
```

Out[205]:

	month	Temperature	RH	Ws	Rain	FFMC	DMC	I
month	1.000000	-0.059017	-0.037884	-0.041447	0.035322	0.015577	0.068178	0.0616
Temperature	-0.059017	1.000000	-0.654443	-0.278132	-0.326786	0.677491	0.483105	0.6075
RH	-0.037884	-0.654443	1.000000	0.236084	0.222968	-0.645658	-0.405133	-0.6906
Ws	-0.041447	-0.278132	0.236084	1.000000	0.170169	-0.163255	-0.001246	0.0152
Rain	0.035322	-0.326786	0.222968	0.170169	1.000000	-0.544045	-0.288548	-0.3471
FFMC	0.015577	0.677491	-0.645658	-0.163255	-0.544045	1.000000	0.602391	0.7397
DMC	0.068178	0.483105	-0.405133	-0.001246	-0.288548	0.602391	1.000000	0.6744
ISI	0.061680	0.607551	-0.690637	0.015248	-0.347105	0.739730	0.674499	1.0000
BUI	0.085822	0.455504	-0.348587	0.029756	-0.299171	0.589652	0.982073	0.6358
fire	0.108848	0.106087	-0.166084	0.012109	0.090625	0.283148	0.321358	0.3006

In [26]:
```
plt.figure(figsize = (15,10))
sns.heatmap(df.corr(), cmap="CMRmap", annot=True)
plt.show()
```



Report

- RH is negatively corelated with Temperature, FFMC and ISI
- Rain is negatively correlated with Temperature and FFMC,DMC, ISI and BUI

Checking Null Values

In [206]:
```
df.isnull().sum()
```

Out[206]:
```
day            0
month          0
Temperature    0
 RH            0
 Ws            0
Rain           0
FFMC           0
DMC            0
ISI            0
BUI            0
fire           0
dtype: int64
```

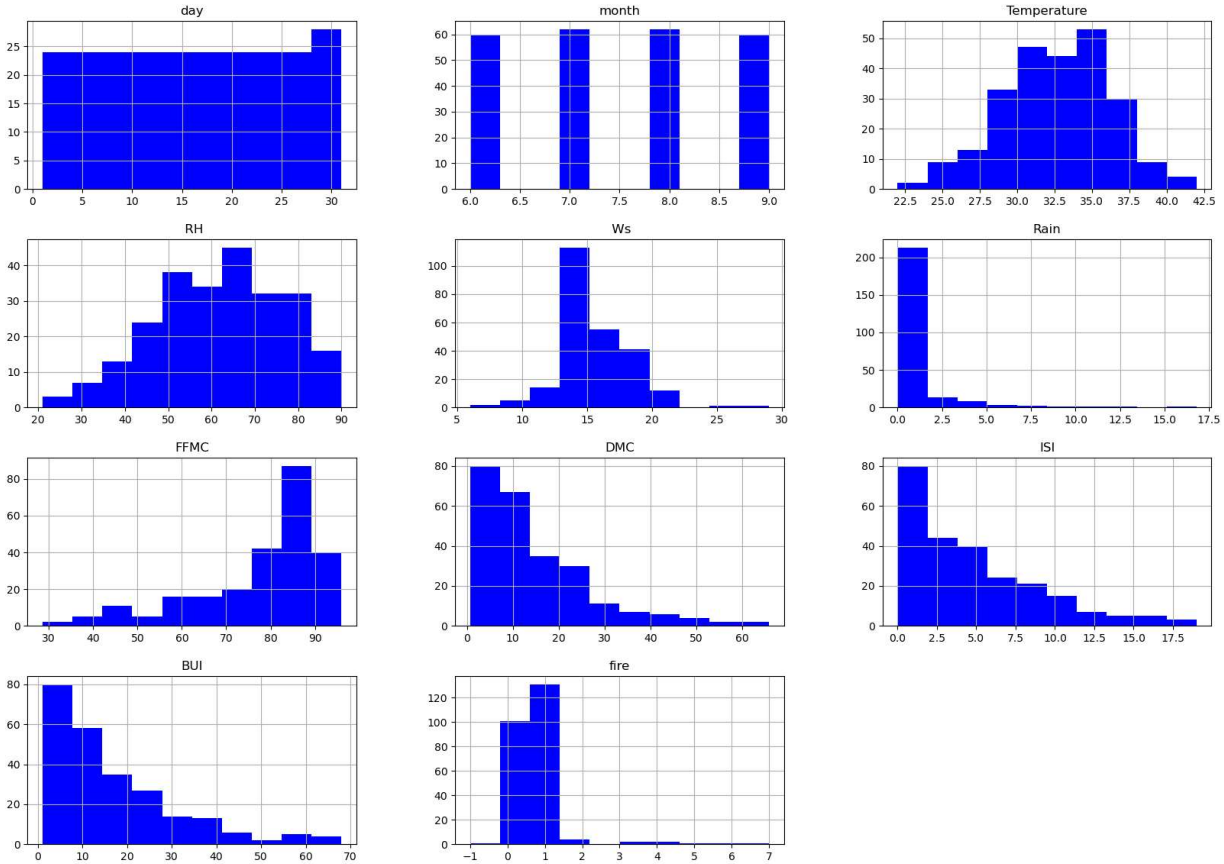
Report: There are no null values

In [207]:
```
continues_features=[feature for feature in numeric_features if len(df[featur
print('Num of continues features :',continues_features)
```

```
Num of continues features : ['day', 'Temperature', ' RH', ' Ws', 'Rain', 'FFM
C', 'DMC', 'ISI', 'BUI']
```

In [208]:
```
#Histograms
df.hist(figsize=(20,14),color='b')
```

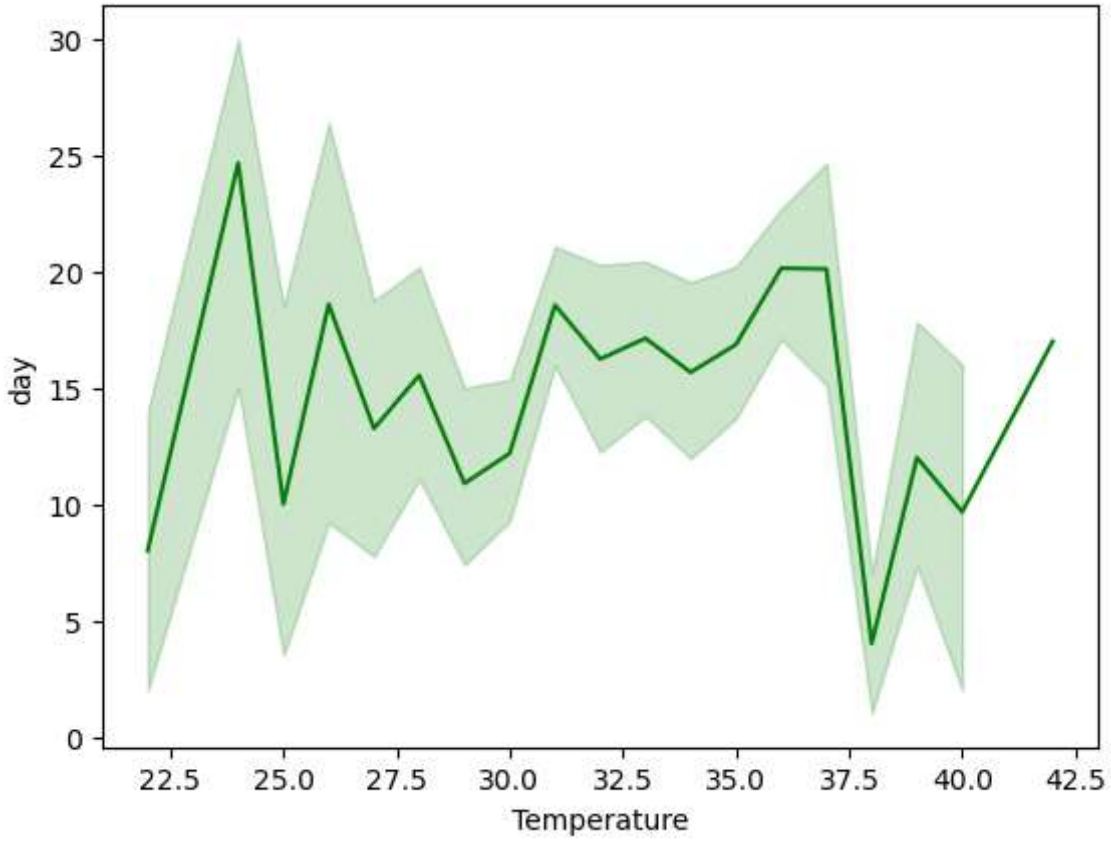
Out[208]:
```
array([[<AxesSubplot:title={'center':'day'}>,
        <AxesSubplot:title={'center':'month'}>,
        <AxesSubplot:title={'center':'Temperature'}>],
       [<AxesSubplot:title={'center':' RH'}>,
        <AxesSubplot:title={'center':' Ws'}>,
        <AxesSubplot:title={'center':'Rain'}>],
       [<AxesSubplot:title={'center':'FFMC'}>,
        <AxesSubplot:title={'center':'DMC'}>,
        <AxesSubplot:title={'center':'ISI'}>],
       [<AxesSubplot:title={'center':'BUI'}>,
        <AxesSubplot:title={'center':'fire'}>, <AxesSubplot:>]],
      dtype=object)
```



Report:

In [209]:
```
#Line Plot
sns.lineplot(x='Temperature',y='day', data=df,color='g')
```

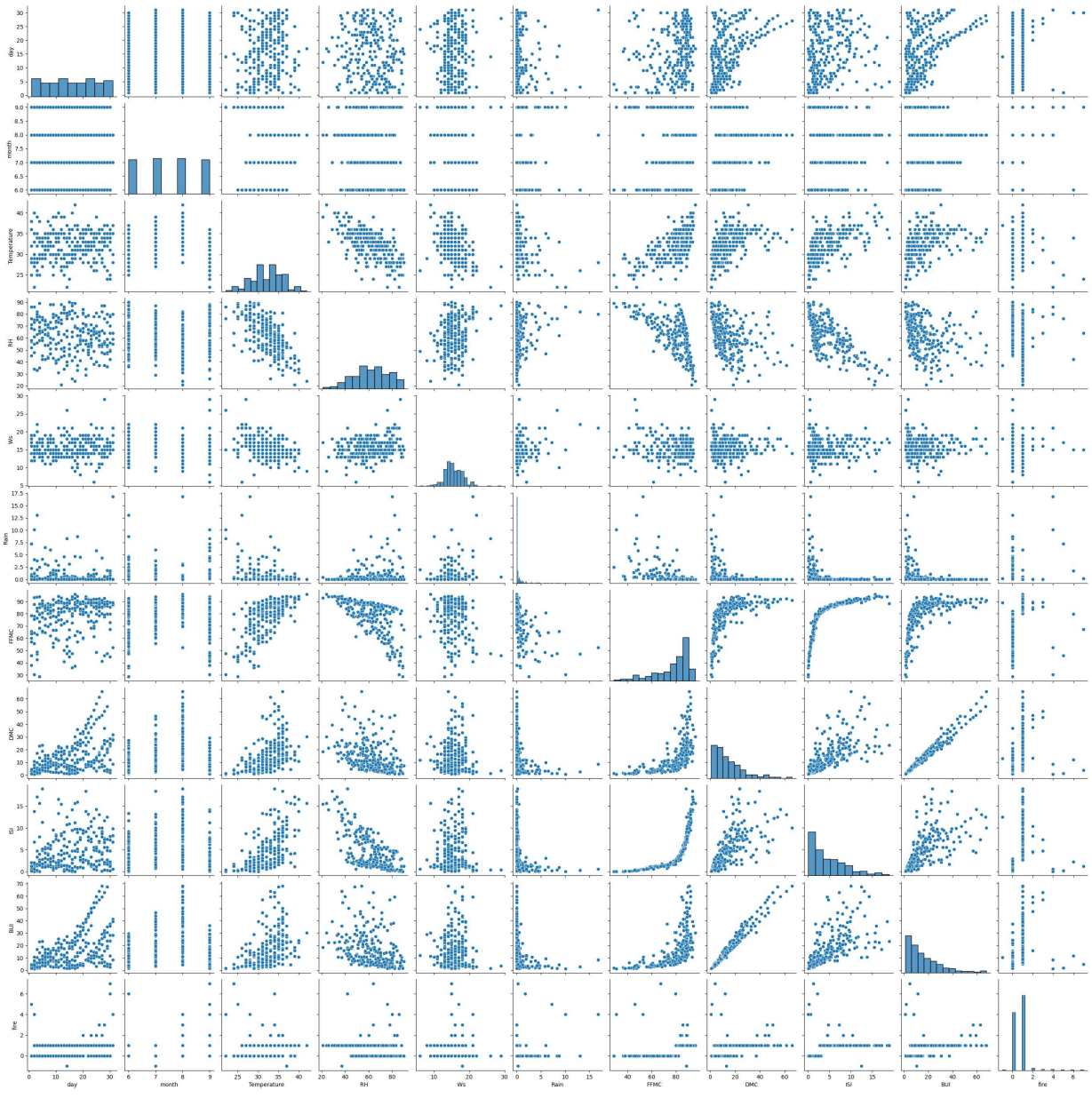
Out[209]: <AxesSubplot:xlabel='Temperature', ylabel='day'>



Report:

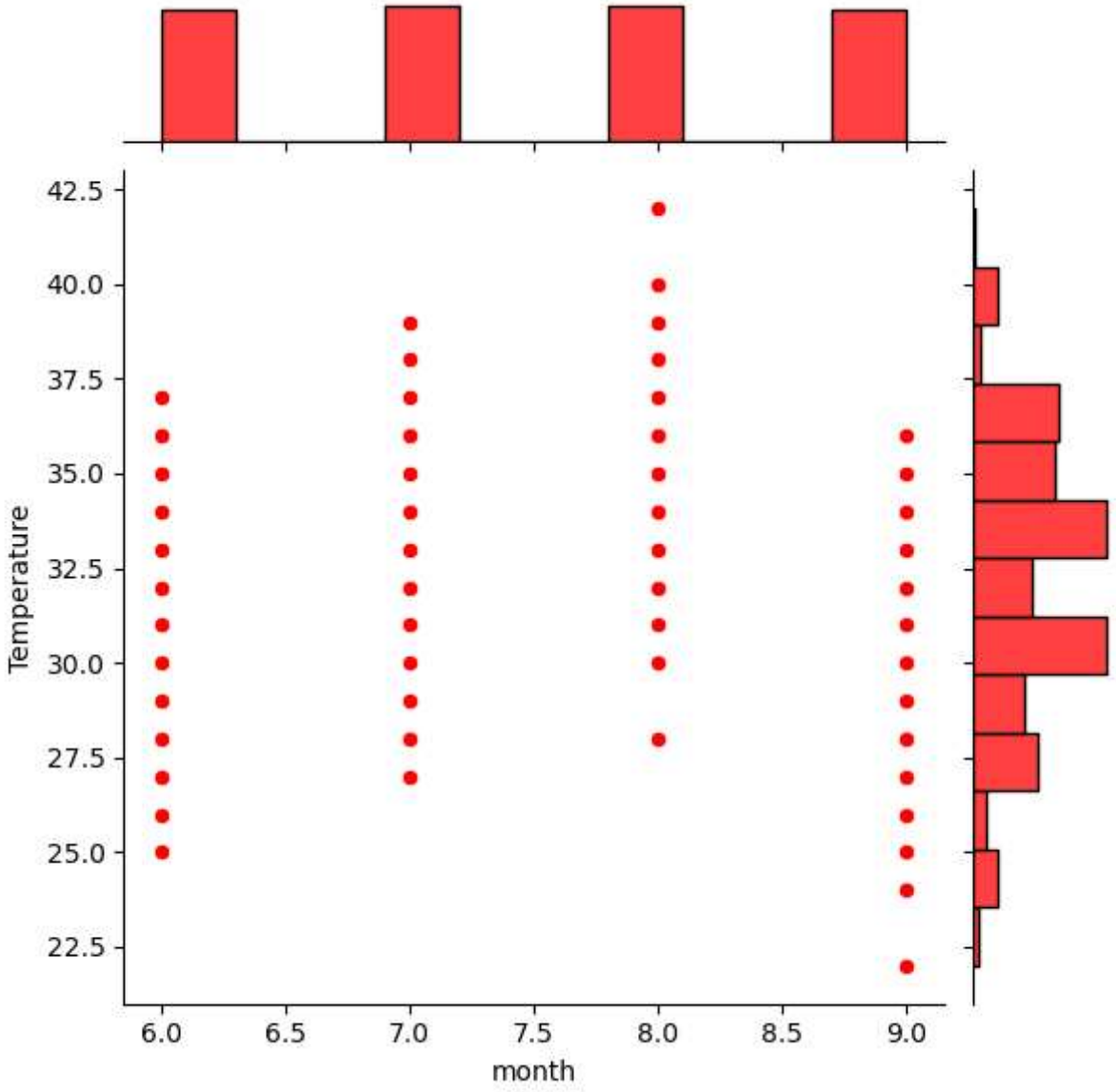
In [210]:
```
#Pairplot
sns.pairplot(df)
```

Out[210]: <seaborn.axisgrid.PairGrid at 0x28cf82f4ac0>

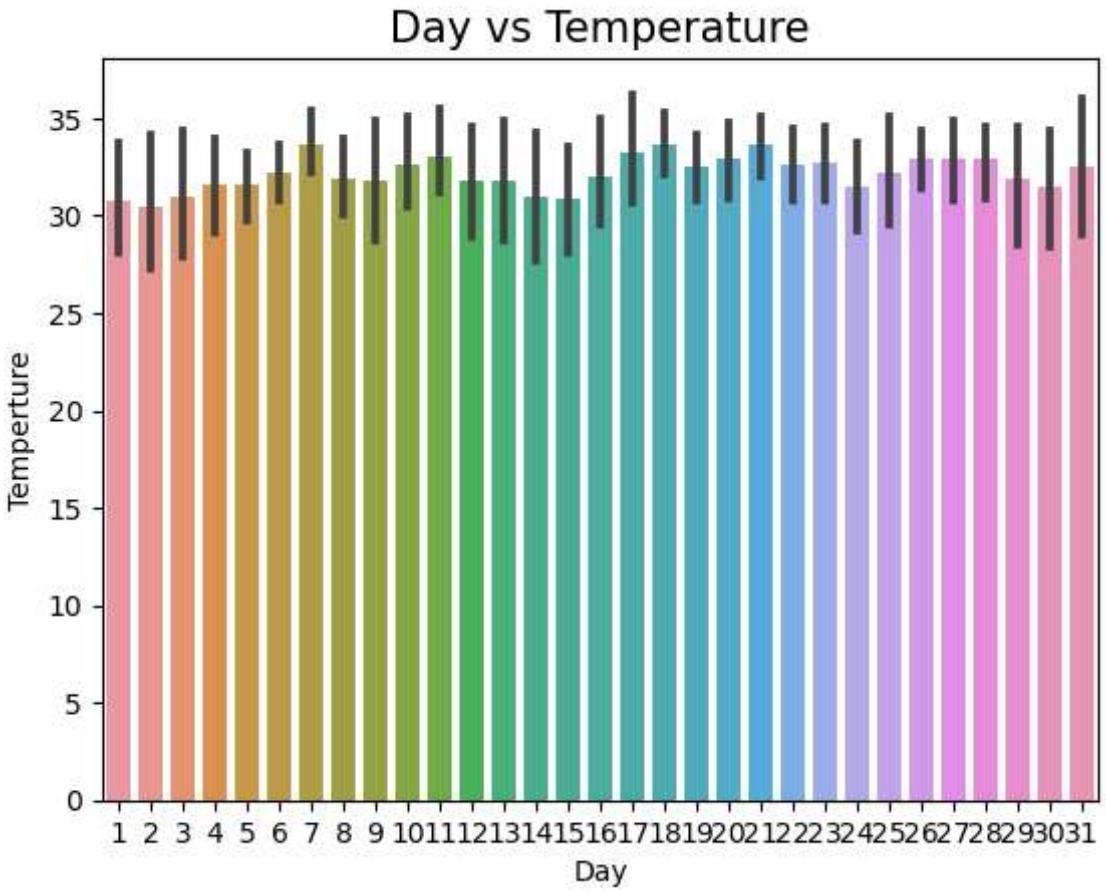


In [211]:
```
#Jointplot
sns.jointplot(x='month',y='Temperature',data=df,color='r')
```

Out[211]: <seaborn.axisgrid.JointGrid at 0x28cf83335e0>



In [156]:
```
#Barplot
plt.style.use('default')
sns.barplot(x='day',y='Temperature',data=df)
plt.title('Day vs Temperature',fontsize=15)
plt.xlabel('Day')
plt.ylabel('Temperture')
plt.show()
```



Basic analysis of data

In [212]:
```
#The highest temperature in the dataset
df.Temperature.max()
```

Out[212]: 42

In [213]:
```
#The lowest temperature in the dataset
df.Temperature.min()
```

Out[213]: 22

In [215]:
```
#When did it rain the most
highest_rain = df.sort_values(by='Rain',ascending=False)[['Rain','day','mont
highest_rain
```

Out[215]:

	Rain	day	month
91	16.8	31	8

Report: On August 31st it rain the most

In [216]:
```
#When did it rain the least
lowest_rain = df.sort_values(by='Rain',ascending=True)[['Rain','day','month'
lowest_rain
```

Out[216]:

	Rain	day	month
0	0.0	1	6

Report: On June 1st it rain the lowest

In [217]:
```
#hotest month
hotest_month = df.sort_values(by='Temperature',ascending=False)[['month']].h
hotest_month
```

Out[217]:

	month
201	8

Report: June is the hotest month

In [218]:
```
#collest month
coolest_month = df.sort_values(by='Temperature',ascending=True)[['month']].h
coolest_month
```

Out[218]:

	month
105	9

Report: September is the collest month

In [220]:
```
#The day which was coolest , id it rain or not , from which region it belong
coolest_temperature = df.sort_values(by='Temperature',ascending=True)[['Temp
coolest_temperature
```

Out[220]:

	Temperature	day	month	Rain
105	22	14	9	8.3

Report: The day was coolest on 14th September, it rain on that day and there was no forest fire on that day.

Splitting Training and Testing data

In [222]:
```
## Independent And Dependent Features
X=df.loc[: , df.columns != 'Temperature']
y=df.iloc[:,2]
```

In [223]:
```
X
```

Out[223]:

	day	month	RH	Ws	Rain	FFMC	DMC	ISI	BUI	fire
0	1	6	57	18	0.0	65.7	3.4	1.3	3.4	0
1	2	6	61	13	1.3	64.4	4.1	1.0	3.9	0
2	3	6	82	22	13.1	47.1	2.5	0.3	2.7	0
3	4	6	89	13	2.5	28.6	1.3	0.0	1.7	0
4	5	6	77	16	0.0	64.8	3.0	1.2	3.9	0
...	...	...	...	...	...	...	...	...	...	...
241	26	9	65	14	0.0	85.4	16.0	4.5	16.9	1
242	27	9	87	15	4.4	41.1	6.5	0.1	6.2	0
243	28	9	87	29	0.5	45.9	3.5	0.4	3.4	0
244	29	9	54	18	0.1	79.7	4.3	1.7	5.1	0
245	30	9	64	15	0.2	67.3	3.8	1.2	4.8	7

244 rows × 10 columns

In [225]:
```
1 y
```

Out[225]:
```
0      29
1      29
2      26
3      25
4      27
       ..
241    30
242    28
243    27
244    24
245    24
Name: Temperature, Length: 244, dtype: int32
```

In [226]:
```
1 from sklearn.model_selection import train_test_split
```

In [227]:
```
1 X_train, X_test, y_train, y_test = train_test_split(
2     X, y, test_size=0.33, random_state=10)
```

In [228]:
```
1 ## Standardize or feature scaling the datasets
2 from sklearn.preprocessing import StandardScaler
3 scaler=StandardScaler()
```

In [229]:
```
1 scaler
```

Out[229]:
```
StandardScaler()
```

In [230]:
```
1 X_train = scaler.fit_transform(X_train)
```

In [167]:
```
1 X_train
```

Out[167]:
```
array([[ 1.0627621 ,  1.33562856, -0.85631108, ..., -0.9695694 ,
        -0.32535487,  0.98176139],
       [ 0.34495731,  0.44338489, -0.52508491, ...,  1.17918645,
         0.76565444, -1.01857744],
       [ 1.30203036, -1.34110244,  0.13736742, ...,  0.4708054 ,
         0.35302912, -1.01857744],
       ...,
       [-0.01394508,  1.33562856, -0.72382061, ...,  0.5180308 ,
         0.08727045,  0.98176139],
       [-1.32992053, -1.34110244,  0.13736742, ..., -0.59176617,
        -0.76595478,  0.98176139],
       [-0.61211574, -1.34110244,  1.13104591, ..., -0.80428049,
        -0.27639932, -1.01857744]])
```

In [231]:
```
1 X_test=scaler.transform(X_test)
```

In [232]:
```
1 X_test
```

```
       -7.65709751e-01],
      [ 5.84225573e-01, -1.34110244e+00,  1.19729114e+00,
        2.72169591e-01, -1.75423310e-01, -1.20698154e+00,
       -8.98449972e-01, -8.98731295e-01, -8.00923028e-01,
       -7.65709751e-01],
      [-9.71018132e-01,  1.33562856e+00,  4.87694966e-03,
       -9.14668296e-02, -4.41414004e-01,  3.98178614e-01,
       -7.06597555e-01, -2.13962947e-01, -7.10005585e-01,
        4.22963482e-01],
      [-1.39450828e-02,  1.33562856e+00,  1.32978161e+00,
       -9.14668296e-02, -1.75423310e-01, -2.22472138e+00,
       -1.09829624e+00, -1.06402021e+00, -1.05968806e+00,
       -7.65709751e-01],
      [ 1.42166449e+00, -4.48858777e-01, -9.22556312e-01,
       -8.18739670e-01, -4.41414004e-01,  8.76311426e-01,
        6.04393966e-01,  9.43059434e-01,  8.98533779e-01,
        4.22963482e-01],
      [ 5.84225573e-01,  4.43384889e-01,  4.02348347e-01,
        1.36307885e+00, -4.41414004e-01,  7.39702051e-01,
```

Model Training

Linear Regression

In [233]:
```
1 from sklearn.linear_model import LinearRegression
```

In [234]:
```
1 regression=LinearRegression()
```

In [235]:
```
1 regression
```

Out[235]:
```
LinearRegression()
```

In [236]:
```
1 regression.fit(X_train,y_train)
```

Out[236]:
```
LinearRegression()
```

In [237]:
```
1 ## print the coefficients and the intercept
2 print(regression.coef_)
```

```
[-0.12763382 -0.26075655 -1.1847671  -0.5754377  -0.08222937  1.17292577
 -0.66319282  0.22933348  1.17189594 -0.22725638]
```

In [238]:
```
1 print(regression.intercept_)
```

```
32.17791411042945
```

In [239]:
```
1 ## PRediction for the test data
2 reg_pred=regression.predict(X_test)
```

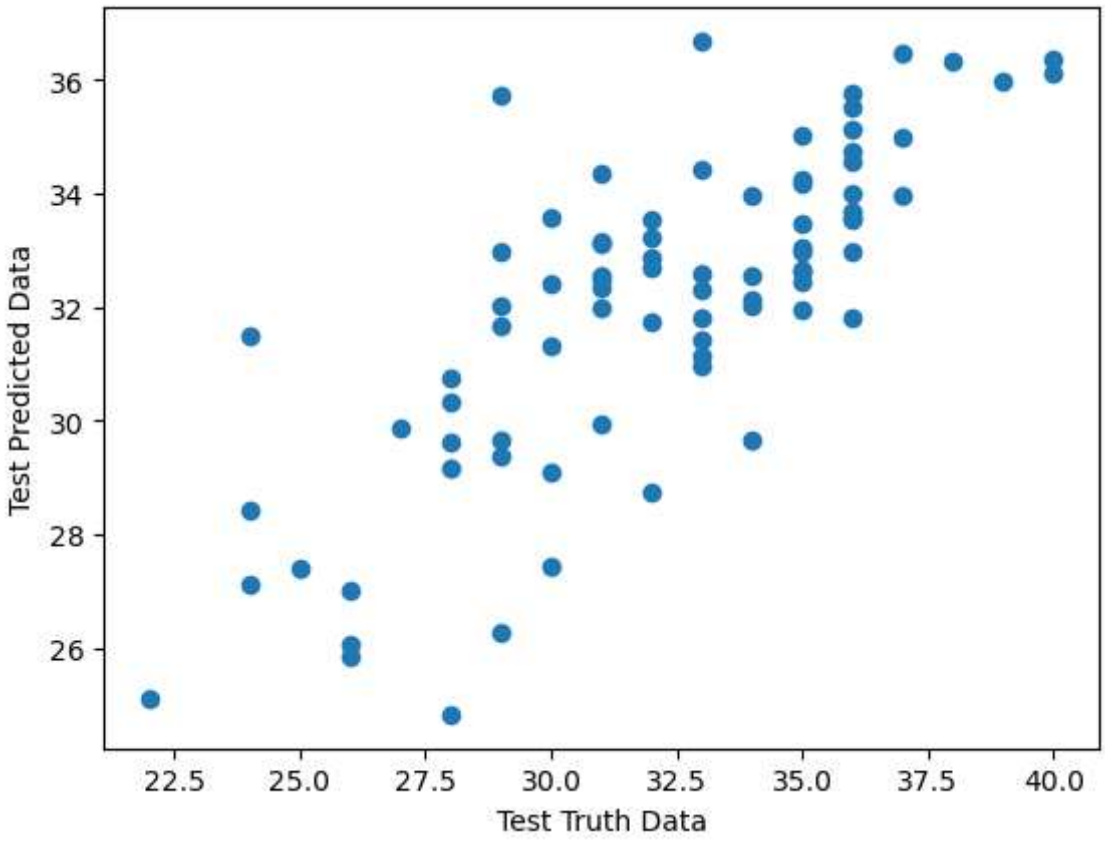
In [240]:
```
1 reg_pred
```

Out[240]:
```
array([32.14558088, 33.46188639, 33.58872421, 31.95440018, 32.97594044,
       34.74720424, 33.68332666, 33.03793263, 32.01092316, 33.22995495,
       33.52964187, 27.41881115, 35.03578801, 29.6650372 , 32.46223944,
       32.0293162 , 34.26176875, 27.40652403, 36.31878358, 34.55316286,
       33.12800137, 33.57946946, 33.96777591, 32.97987204, 35.96677583,
       29.16393442, 31.82732812, 32.68374345, 27.01830529, 32.32925842,
       25.83217787, 26.276322  , 34.42430306, 32.55883754, 33.16655026,
       30.33883365, 29.10179621, 31.99512928, 27.13009898, 35.51411029,
       32.64654066, 33.96981592, 34.34427118, 30.75288268, 36.35322357,
       34.17599001, 24.82516368, 35.14325283, 34.0089041 , 29.38893565,
       31.32098024, 32.54767033, 35.76861722, 32.48038529, 29.62862121,
       29.9331413 , 32.60889327, 36.47997033, 31.40695171, 32.98800746,
       32.2978315 , 32.85904151, 31.68489082, 26.05550531, 31.75160187,
       36.13280259, 29.64126398, 29.85610178, 34.97167514, 33.55028755,
       28.74559695, 31.49483848, 32.42426697, 28.40746611, 30.97166858,
       31.13913039, 32.59641539, 35.73935291, 31.81416645, 36.68568128,
       25.09946125])
```

Assumptions of Linear Regression

In [241]:
```
1 plt.scatter(y_test,reg_pred)
2 plt.xlabel("Test Truth Data")
3 plt.ylabel("Test Predicted Data")
```

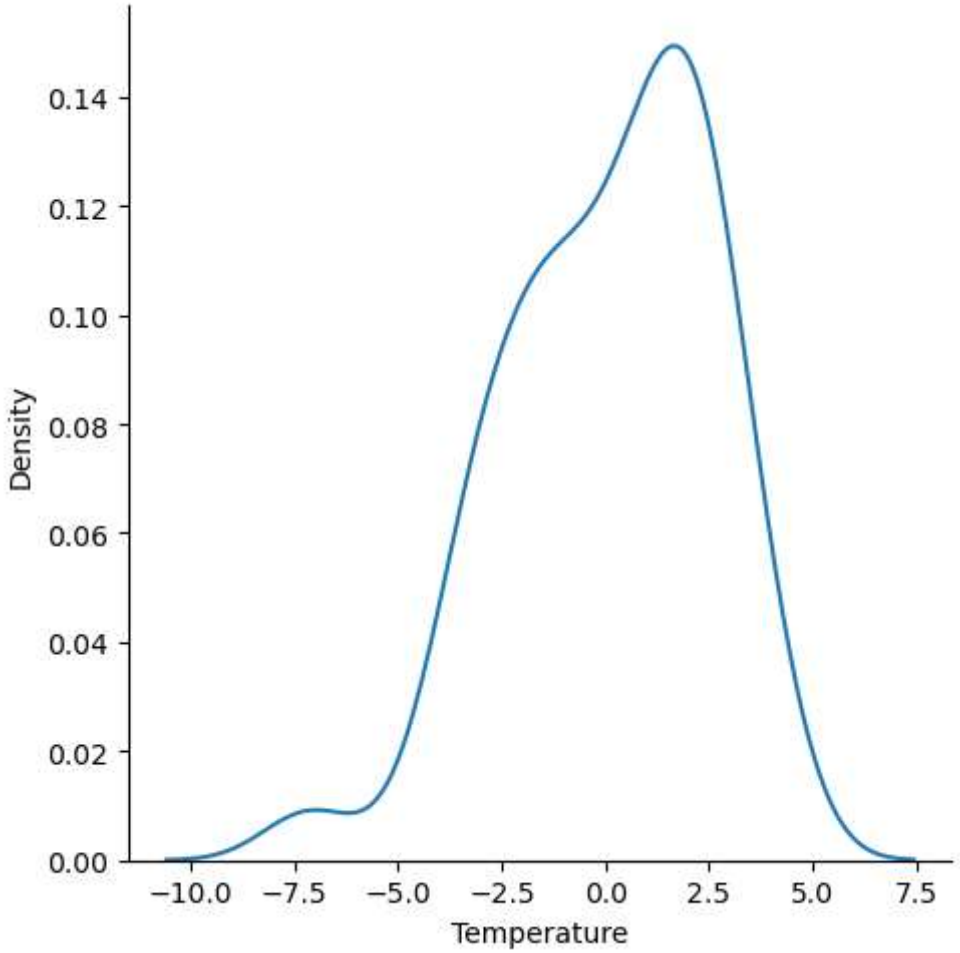
Out[241]:
```
Text(0, 0.5, 'Test Predicted Data')
```



In [242]:
```
## residuals
residuals=y_test-reg_pred
```

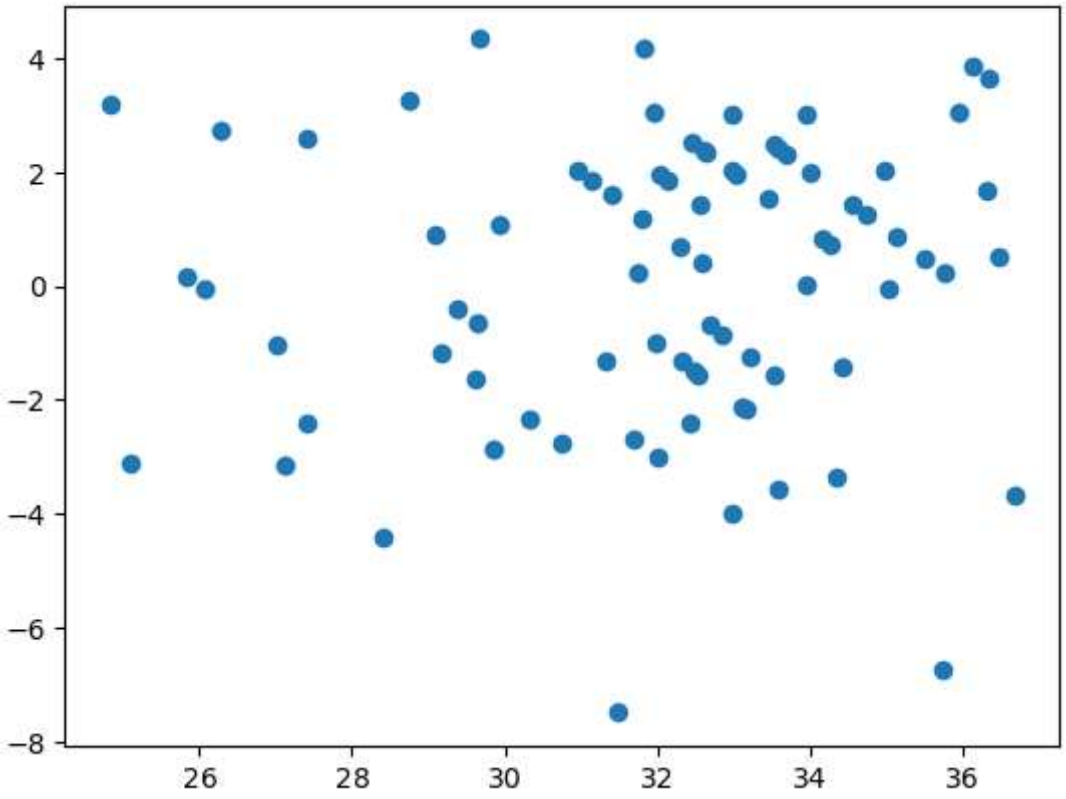
In [243]:
```
sns.displot(residuals,kind="kde")
```

Out[243]: <seaborn.axisgrid.FacetGrid at 0x28c81e0ad30>



In [244]:
```
## SCatter plot with predictions and residual
##uniform distribution
plt.scatter(reg_pred,residuals)
```

Out[244]: <matplotlib.collections.PathCollection at 0x28c8209d940>



In [245]:
```
## Performance Metrics
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)))
```

```
6.124774596218712
2.0616126398983243
2.4748281952933042
```

R square and adjusted R square

In [246]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(score)
```

```
0.5949863318572137
```

In [247]:
```
## Adjusted R square
#display adjusted R-squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[247]:
```
0.5371272364082442
```

Ridge Regression

In [248]:
```
## Ridge
from sklearn.linear_model import Ridge
ridge=Ridge()
```

In [249]:
```
ridge.fit(X_train,y_train)
```

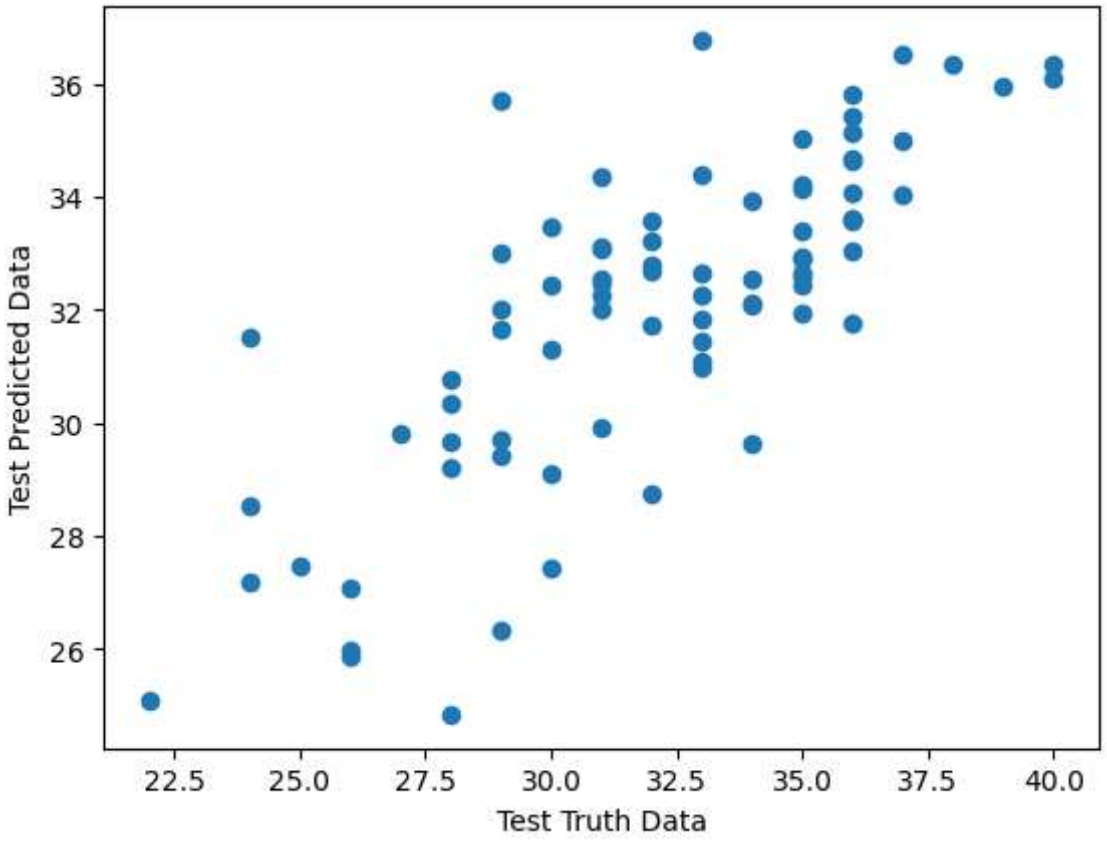
Out[249]:
```
Ridge()
```

In [250]:
```
ridge_pred = ridge.predict(X_test)
```

Assumptions of Ridge Regresson

In [251]:
```
1 plt.scatter(y_test,ridge_pred)
2 plt.xlabel('Test Truth Data')
3 plt.ylabel('Test Predicted Data')
```

Out[251]: Text(0, 0.5, 'Test Predicted Data')

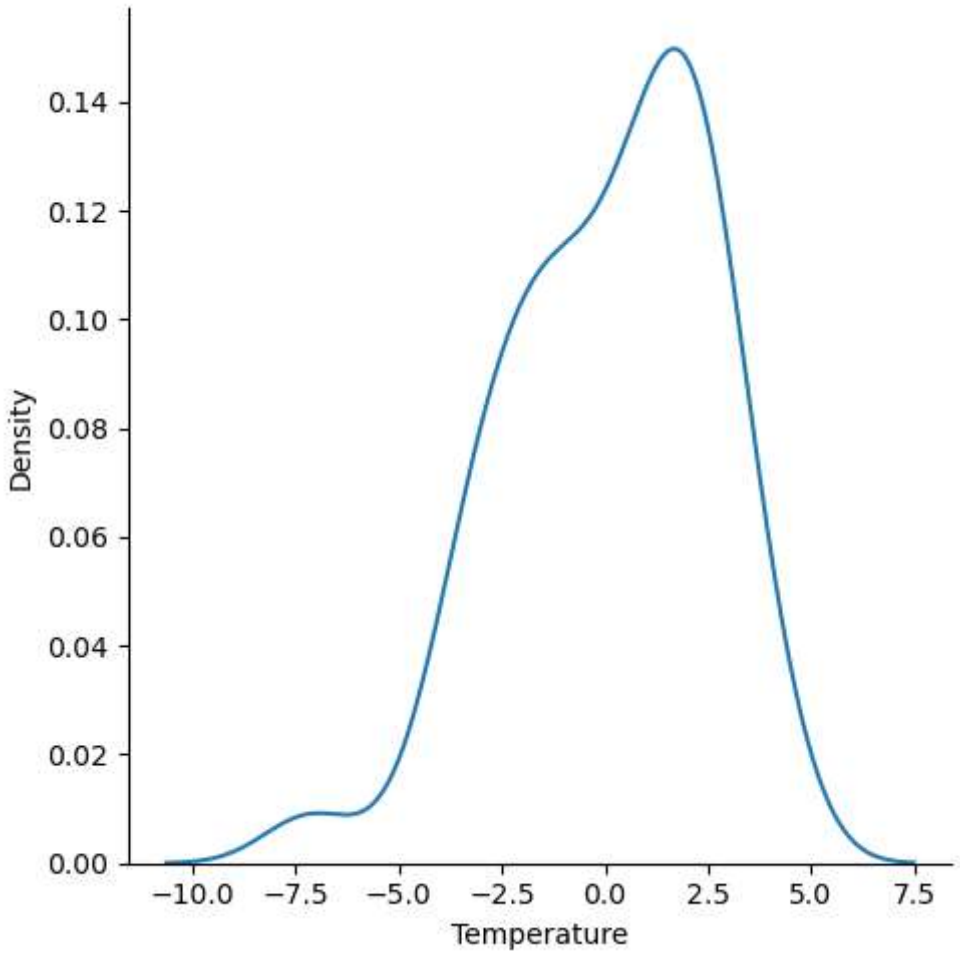


In [252]:
```
## Residual
ridge_residuals = y_test-ridge_pred
ridge_residuals
```

Out[252]:
```
164    1.870739
60     1.611998
61     2.437400
63     3.042947
69     2.046794
         ...
171    0.331989
234   -6.704687
146    1.173161
210   -3.772551
105   -3.094730
Name: Temperature, Length: 81, dtype: float64
```

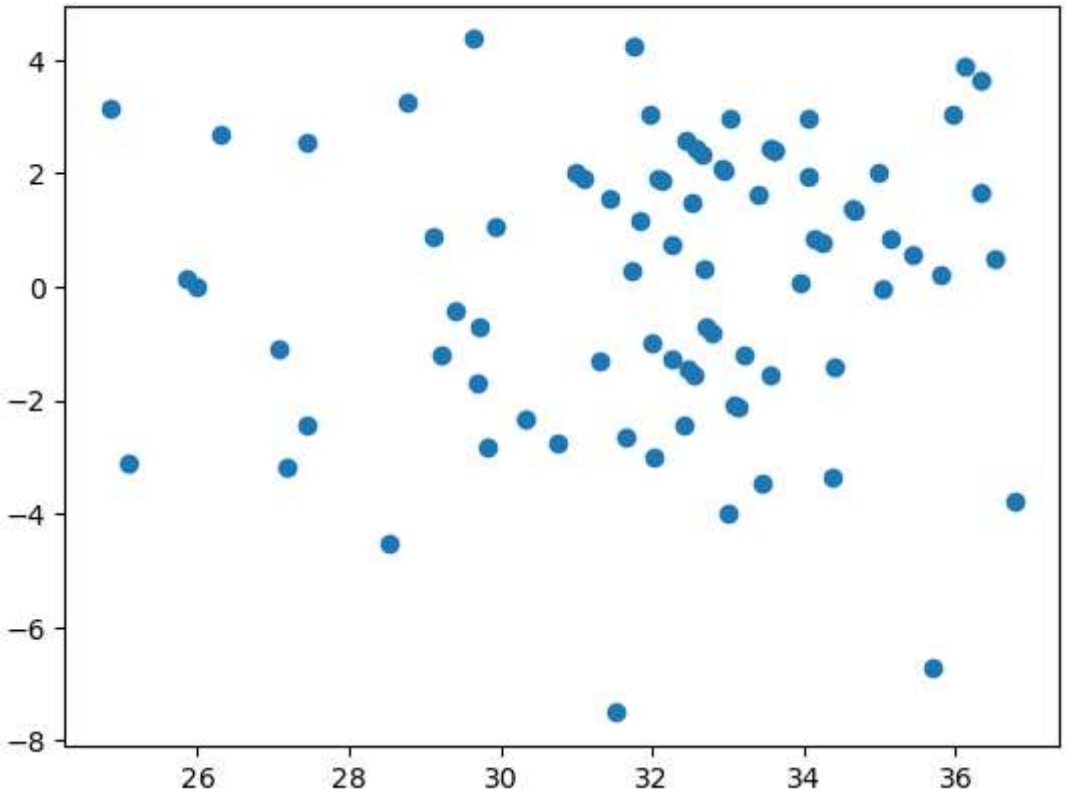
In [253]:
```
##Normal distribution expected
sns.displot(ridge_residuals,kind='kde')
```

Out[253]:
```
<seaborn.axisgrid.FacetGrid at 0x28cf83334c0>
```



In [254]:
```
## Scatter plot with prediction and residual
## If it is Uniform distribution, model is good
plt.scatter(ridge_pred,ridge_residuals)
```

Out[254]: <matplotlib.collections.PathCollection at 0x28c82156370>



In [255]:
```
## Performance Matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,ridge_pred))
print(mean_absolute_error(y_test,ridge_pred))
print(np.sqrt(mean_squared_error(y_test,ridge_pred)))
```

```
6.144470282712657
2.065566411464254
2.4788042041905323
```

In [256]:
```
#R2 score
from sklearn.metrics import r2_score
r_score_ridge=r2_score(y_test,ridge_pred)
print(r_score_ridge)
```

```
0.5936839129504954
```

In [258]:
```
## Adjusted R square
#display adjusted R-squared
1 - (1-r_score_ridge)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[258]: 0.535638757657709

Lasso Regression

In [259]:
```
## Lasso
from sklearn.linear_model import Lasso
lasso=Lasso()
```

In [260]:
```
lasso.fit(X_train,y_train)
```

Out[260]: Lasso()

In [261]:
```
#Lasso Prediction
lasso_pred = lasso.predict(X_test)
```

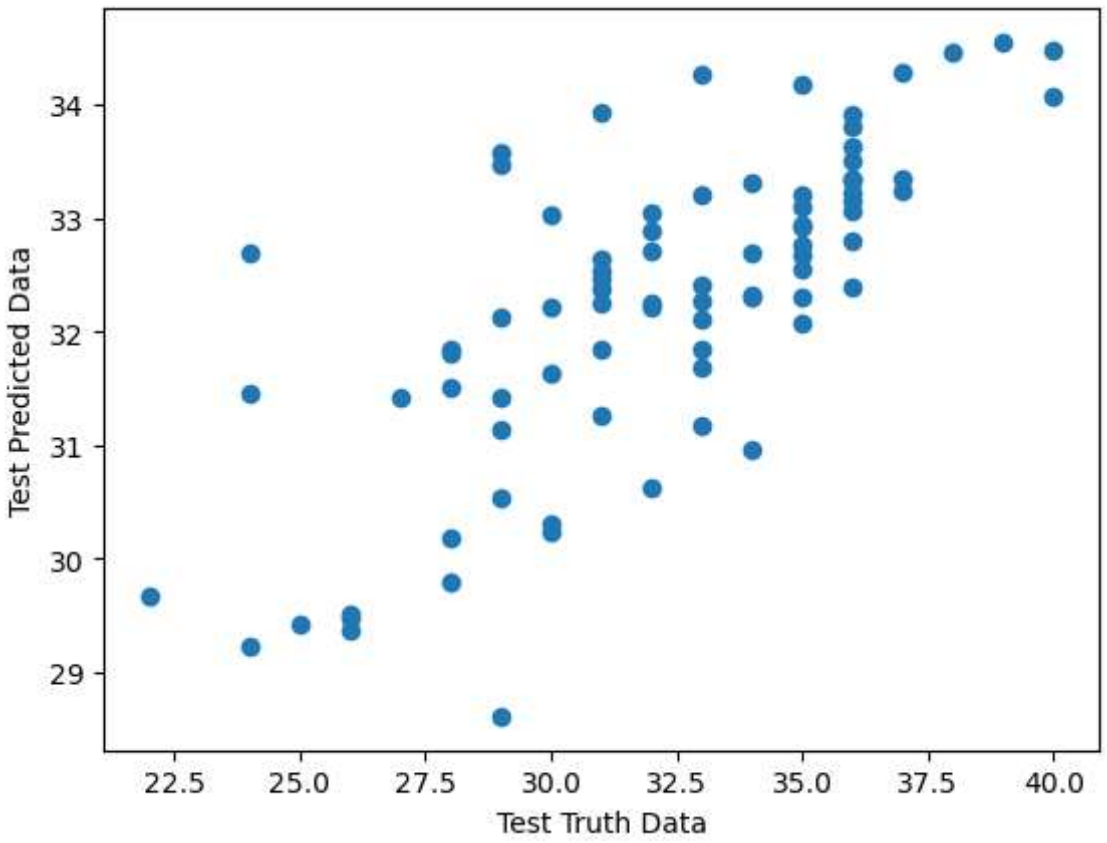
In [262]:
```
lasso_pred
```

Out[262]:
```
array([32.29700076, 32.6744027 , 33.06609539, 32.07346965, 32.92497671,
       33.33947653, 33.32111992, 32.77042154, 32.11916885, 32.70983221,
       33.15976154, 30.29861247, 34.17172792, 30.95174825, 33.0931383 ,
       32.31497272, 32.93691477, 29.42489766, 34.46059856, 33.50695377,
       32.46152593, 33.02899752, 33.30888217, 32.80645043, 34.5498142 ,
       30.18680443, 32.38908351, 32.89121556, 29.47641605, 31.8492542 ,
       29.50217524, 28.6091198 , 33.21226395, 32.70054654, 32.64380834,
       31.80937418, 30.23515603, 32.53110125, 29.22810977, 33.62676377,
       32.55104126, 33.23190428, 33.93112391, 31.84411936, 34.06445535,
       33.20742879, 29.78847846, 33.80519505, 33.21966653, 30.53913152,
       31.62769114, 32.373594  , 33.92016988, 32.24993288, 31.51301599,
       31.26381066, 32.303719  , 34.28571873, 31.84095256, 33.47507571,
       32.27184094, 32.20868418, 31.42230192, 29.36272493, 32.24706577,
       34.47767146, 31.13749714, 31.41648274, 33.33947653, 33.04221928,
       30.62774778, 32.69215994, 32.20868418, 31.45674741, 31.17557904,
       31.67565808, 32.4164261 , 33.56882682, 32.11728577, 34.26736212,
       29.66708507])
```

Assumptions of Lasso Regression

In [263]:
```
plt.scatter(y_test,lasso_pred)
plt.xlabel('Test Truth Data')
plt.ylabel('Test Predicted Data')
```

Out[263]: Text(0, 0.5, 'Test Predicted Data')

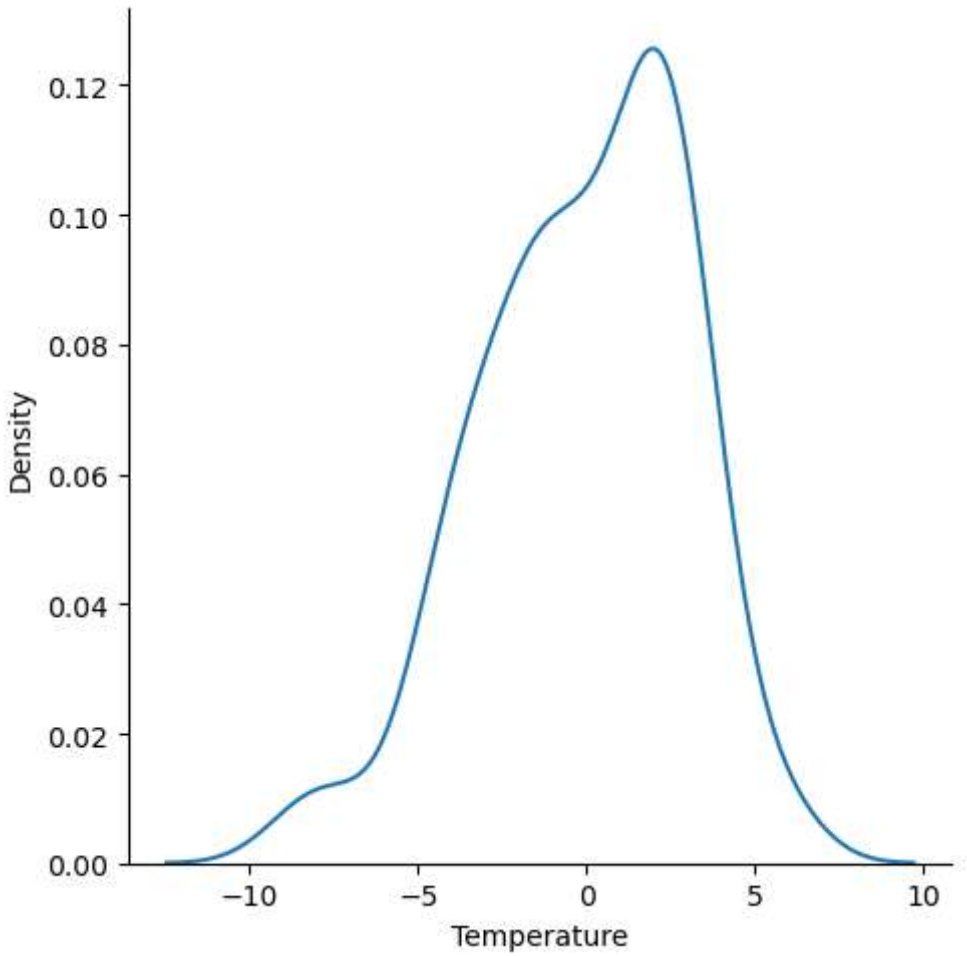


In [264]:
```
## Residual
lasso_residuals = y_test-lasso_pred
lasso_residuals
```

Out[264]:
```
164    1.702999
60     2.325597
61     2.933905
63     2.926530
69     2.075023
         ...
171    0.583574
234   -4.568827
146    0.882714
210   -1.267362
105   -7.667085
Name: Temperature, Length: 81, dtype: float64
```

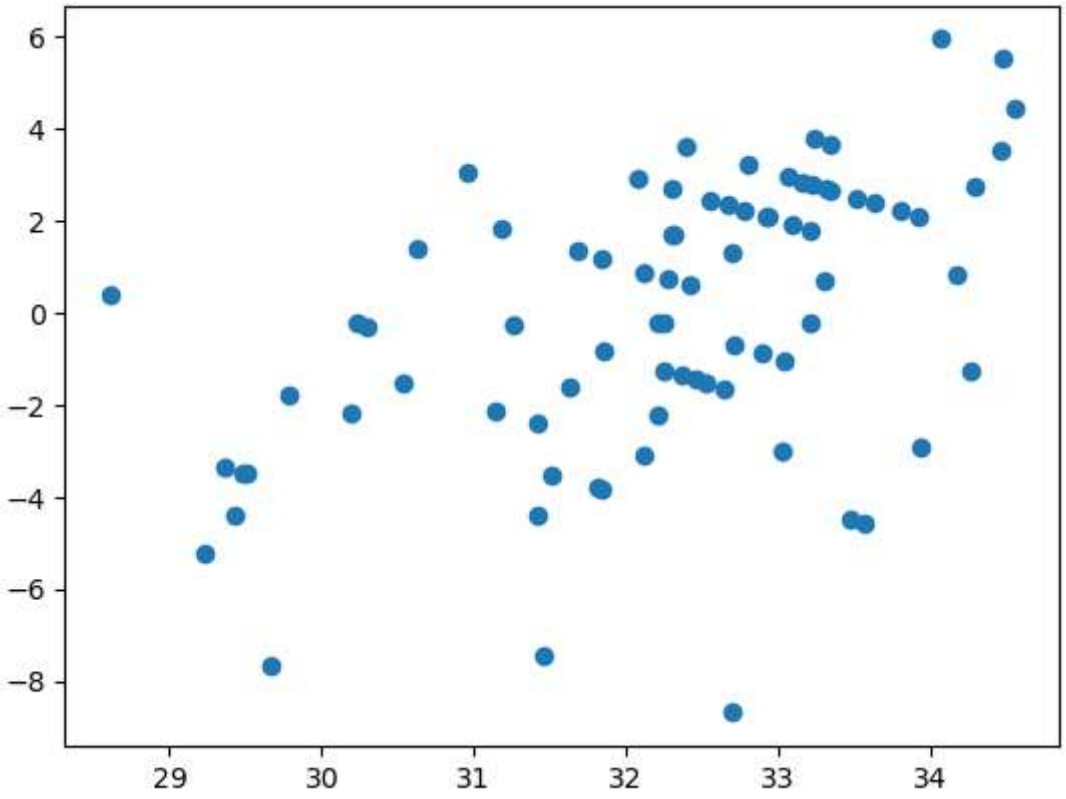
In [265]:
```
##Normal distribution expected
sns.displot(lasso_residuals,kind='kde')
```

Out[265]:
```
<seaborn.axisgrid.FacetGrid at 0x28c821ecaf0>
```



In [266]:
```
## Scatter plot with prediction and residual
## If it is Uniform distribution, model is good
plt.scatter(lasso_pred,lasso_residuals)
```

Out[266]: <matplotlib.collections.PathCollection at 0x28c8271c340>



In [267]:
```
## Performance Matrix
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)))
```

```
9.10609532182792
2.4978660766652734
3.0176307464346794
```

In [268]:
```
#R2 score
from sklearn.metrics import r2_score
r_score_lasso=r2_score(y_test,lasso_pred)
print(r_score_lasso)
```

```
0.39784019626969913
```

In [269]:
```
## Adjusted R square
#display adjusted R-squared
1 - (1-r_score_lasso)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[269]: 0.31181736716537045

Elastic Net Regression

In [270]:
```
## Elastic Net
from sklearn.linear_model import ElasticNet
elasticnet=ElasticNet()
```

In [271]:
```
elasticnet.fit(X_train,y_train)
```

Out[271]: ElasticNet()

In [272]:
```
#ElasticNet Prediction
elasticnet_pred = elasticnet.predict(X_test)
```

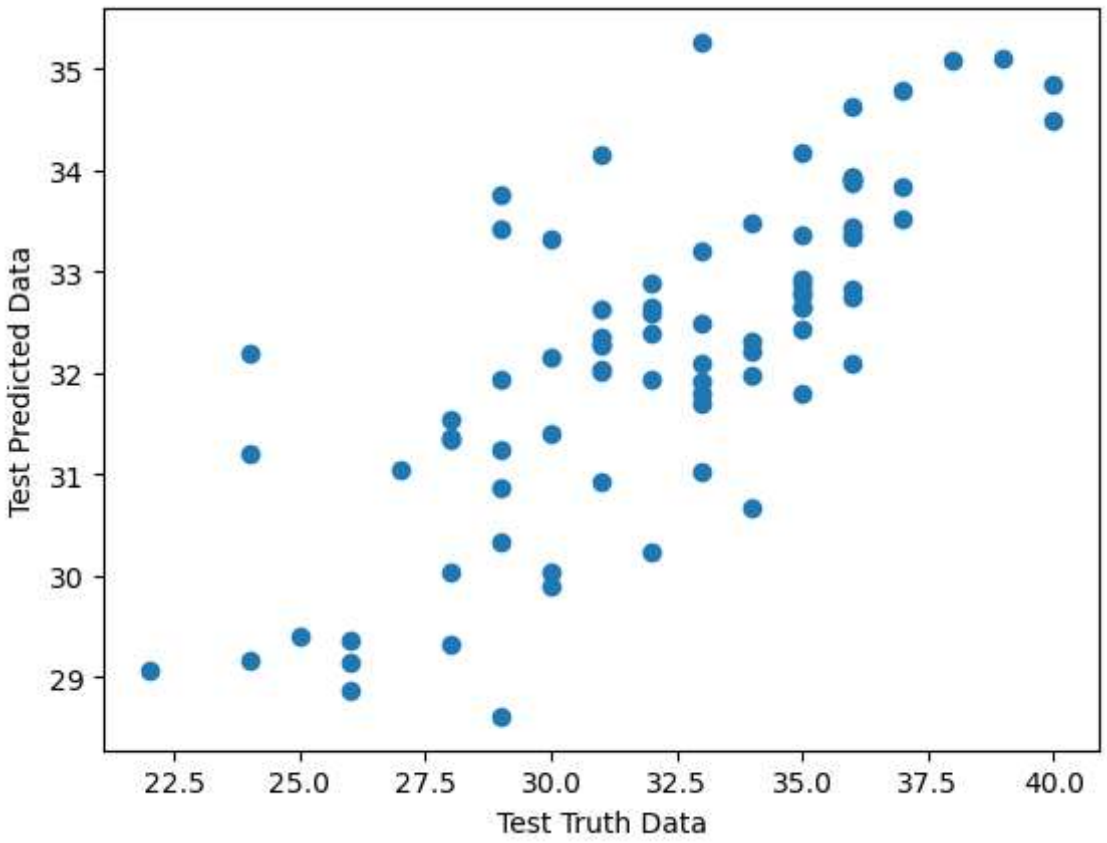
In [273]:
```
elasticnet_pred
```

Out[273]:
```
array([31.96849993, 32.9285877 , 32.74102577, 31.7937955 , 32.89593392,
       33.91383589, 33.44859926, 32.6520612 , 31.94209007, 32.65225696,
       33.34481387, 29.89710641, 34.18210338, 30.67435755, 32.65945852,
       32.21099569, 32.80759978, 29.40256805, 35.08465192, 33.88501852,
       32.36233153, 33.33325695, 33.4722567 , 32.82841736, 35.1137581 ,
       30.03041688, 32.10493921, 32.59468206, 29.36555355, 32.0320653 ,
       29.14369383, 28.6128648 , 33.19892404, 32.32001666, 32.62204464,
       31.35558043, 30.04300005, 32.28018978, 29.1754207 , 33.94380787,
       32.7701881 , 33.51906949, 34.15655052, 31.54246557, 34.48897629,
       33.3548283 , 29.32252254, 33.91327691, 33.38225331, 30.34176683,
       31.40349589, 32.29644759, 34.62152368, 32.00926138, 31.34808463,
       30.93561323, 32.42956124, 34.79260841, 31.7028492 , 33.42969404,
       32.09351218, 32.39852866, 31.24718349, 28.8752595 , 31.93701242,
       34.83906503, 30.87191814, 31.05063201, 33.84559284, 32.89417446,
       30.2418511 , 32.19661981, 32.15129766, 31.20572649, 31.02798791,
       31.90852295, 32.49153324, 33.75443898, 31.79098362, 35.26539323,
       29.06969799])
```

Assumptions of ElasticNet Regression

In [274]:
```
1 plt.scatter(y_test,elasticnet_pred)
2 plt.xlabel('Test Truth Data')
3 plt.ylabel('Test Predicted Data')
```

Out[274]: Text(0, 0.5, 'Test Predicted Data')

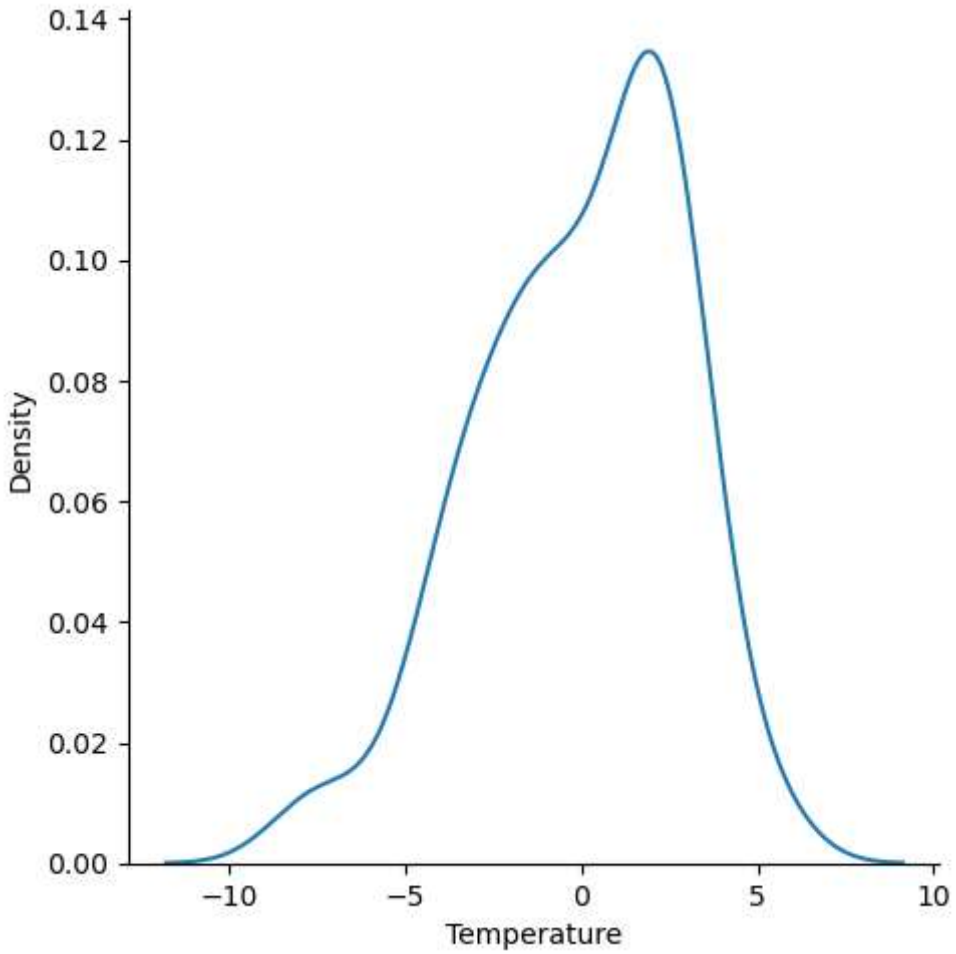


In [275]:
```
## Residual
elasticnet_residuals = y_test-elasticnet_pred
elasticnet_residuals
```

Out[275]:
```
164    2.031500
60     2.071412
61     3.258974
63     3.206205
69     2.104066
         ...
171    0.508467
234   -4.754439
146    1.209016
210   -2.265393
105   -7.069698
Name: Temperature, Length: 81, dtype: float64
```

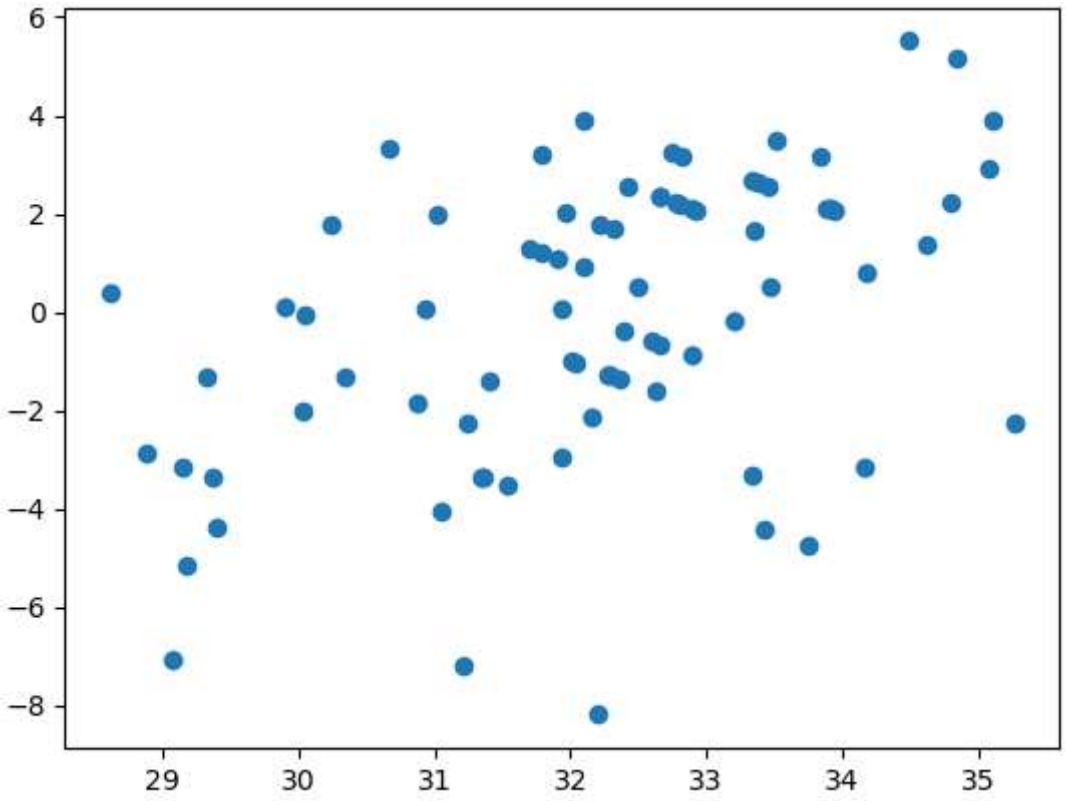
In [276]:
```
##Normal distribution expected
sns.displot(elasticnet_residuals,kind='kde')
```

Out[276]:
```
<seaborn.axisgrid.FacetGrid at 0x28c826d5250>
```



In [277]:
```
## Scatter plot with prediction and residual
## If it is Uniform distribution, model is good
plt.scatter(elasticnet_pred,elasticnet_residuals)
```

Out[277]: <matplotlib.collections.PathCollection at 0x28c8288a730>



In [278]:
```
## Performance Matrix
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test,elasticnet_pred))
print(mean_absolute_error(y_test,elasticnet_pred))
print(np.sqrt(mean_squared_error(y_test,elasticnet_pred)))
```

```
8.3291749910145
2.392517522029474
2.886031010057671
```

In [279]:
```
#R2 score
from sklearn.metrics import r2_score
r_score_elasticnet=r2_score(y_test,elasticnet_pred)
print(r_score_elasticnet)
```

```
0.4492156955789661
```

In [280]:
```
## Adjusted R square
#display adjusted R-squared
1 - (1-r_score_elasticnet)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[280]: 0.3705322235188184