Linear Regression on ALgerian Forest

Problem Statement

- To predict the temperature using Algerian Forest Fire Dataset

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

Attribute Information:

1. **Date** : (DD/MM/YYYY) Day, month ('june' to 'september'), year (2012) Weather data observations
2. **Temp** : temperature noon (temperature max) in Celsius degrees: 22 to 42
3. **RH** : Relative Humidity in %: 21 to 90
4. **Ws** :Wind speed in km/h: 6 to 29
5. **Rain**: total day in mm: 0 to 16.8 FWI Components
6. **Fine Fuel Moisture Code (FFMC)** index from the FWI system: 28.6 to 92.5
7. **Duff Moisture Code (DMC)** index from the FWI system: 1.1 to 65.9
8. **Drought Code (DC)** index from the FWI system: 7 to 220.4
9. **Initial Spread Index (ISI)** index from the FWI system: 0 to 18.5
10. **Buildup Index (BUI)** index from the FWI system: 1.1 to 68
11. **Fire Weather Index (FWI)** Index: 0 to 31.1
12. **Classes**: two classes, namely â€œFireâ€ and â€œnot Fireâ€

Importing Libraries

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Komal Diwe

```
import warnings
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.metrics import r2_score
import missingno as msno

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

Data Collection

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

Out[ ]:

	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

Analyzing Data

Checking Null Values

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

Komal Diwe

Out[ ]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
167	14	07	2012	37	37	18	0.2	88.9	12.9	14.69	12.5	10.4	fire

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

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

Drop rows which have null.

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

Out[ ]:

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

Columns

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

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

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

Columns name having extra space

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

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

Remove extra space in column name

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

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

Function to remove extra space in the data

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

Remove extra space from data.

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

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

Out[ ]:

	index	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Cla
0	0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	no
1	1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	no
2	2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	no

Drop extra index column, which was created for reset_index

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

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

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

Check Null values in all the features

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

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

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

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

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

Out[ ]:

	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	0
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	0
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	0
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	0
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	0

Check duplictes values in all the column

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

Out[ ]:
```
0
```

Check data types

In [ ]:
```
df.dtypes
```

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

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

df = df.astype(convert_features)
```

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

Out[ ]:

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

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

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

Check unique values

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

Out[ ]:
```
day            31
month           4
year            1
Temperature    19
RH             62
Ws             18
Rain           39
FFMC          173
DMC           165
DC            197
ISI           106
BUI           173
FWI           125
Classes         2
Region          2
dtype: int64
```

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

Out[ ]:

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

Segregate categorical feature from the dataset

In [ ]:
```
categorical_features = [feature for feature in df.columns if df[feature].dtypes ==
categorical_features
```

Out[ ]:
```
['day', 'month', 'year', 'Classes', 'Region']
```

Check Value_counts() of Classes and Region feature

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

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

Segregate numerical feature from the datasetr

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

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

Segregate discrete feature from the numerical feature

In [ ]:
```
## Discrete features are those whose data is whole number means there is no decima
discrete_features = [x for x in numerical_features if df[x].dtypes == 'int64']
discrete_features
```

Out[ ]:
```
['Temperature', 'RH', 'Ws']
```

Segregate Continuous feature from the numerical feature

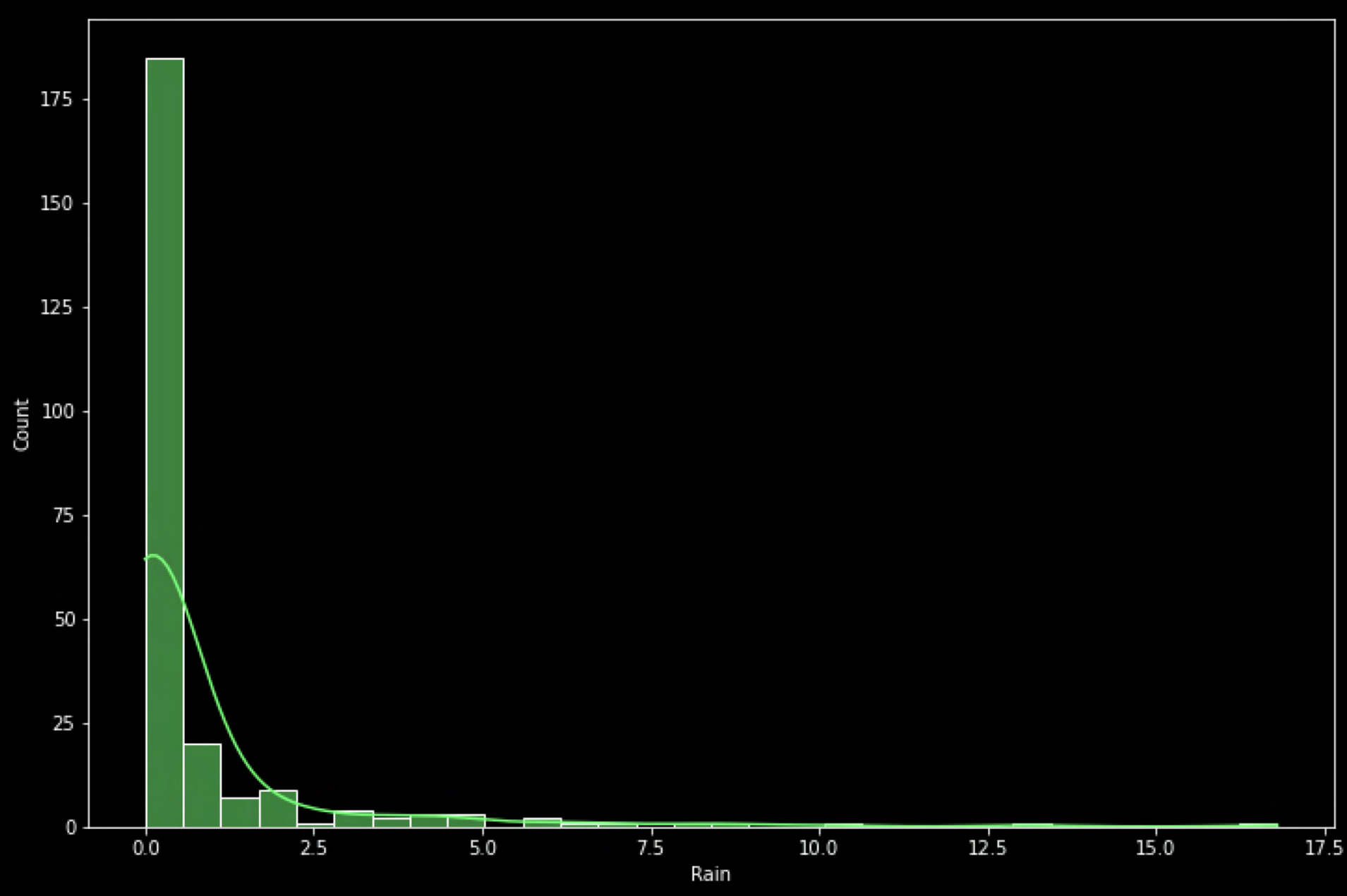
In [ ]:
```
## Continuous features are those features where data has decimal value
continuous_feature = [fea for fea in numerical_features if fea not in discrete_fea
continuous_feature
```

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

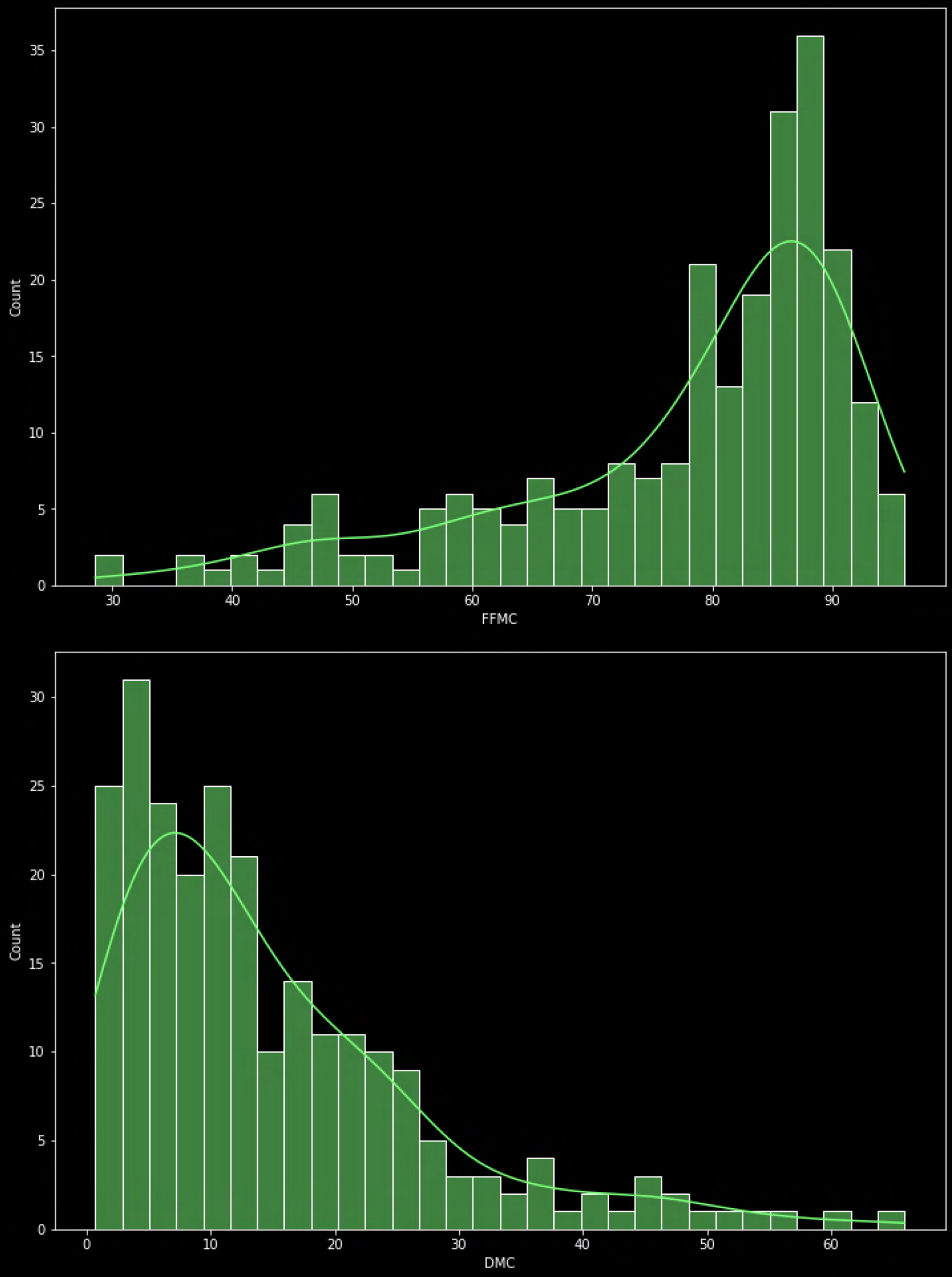
Graphical Analysis

checking distribution of continuous numerical features

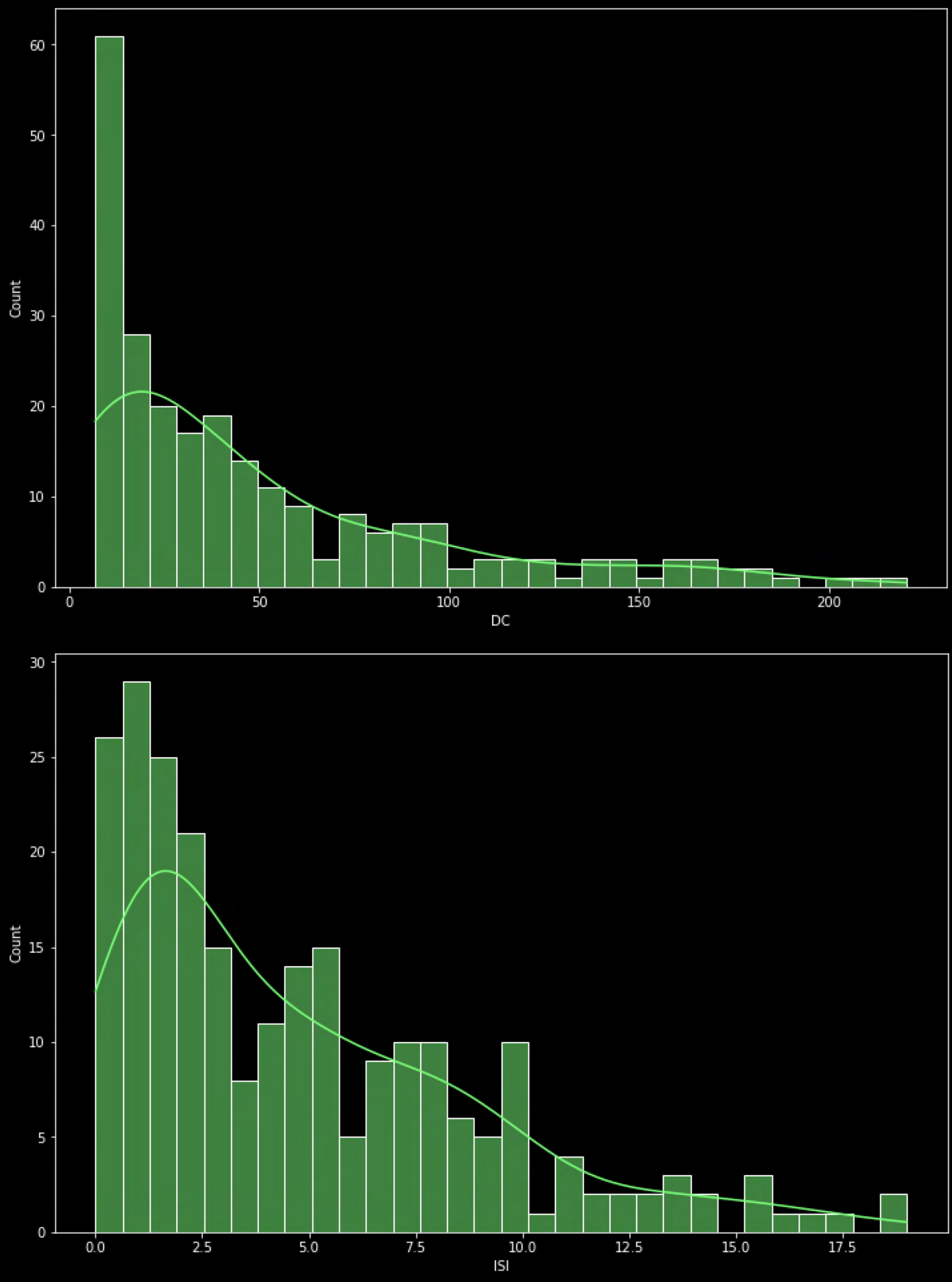
In [ ]:
```
for feature in continuous_feature:
  plt.figure(figsize=(12,8))
  sns.histplot(data=df, x= feature, kde=True, bins = 30, color='purple')
  plt.show()
```



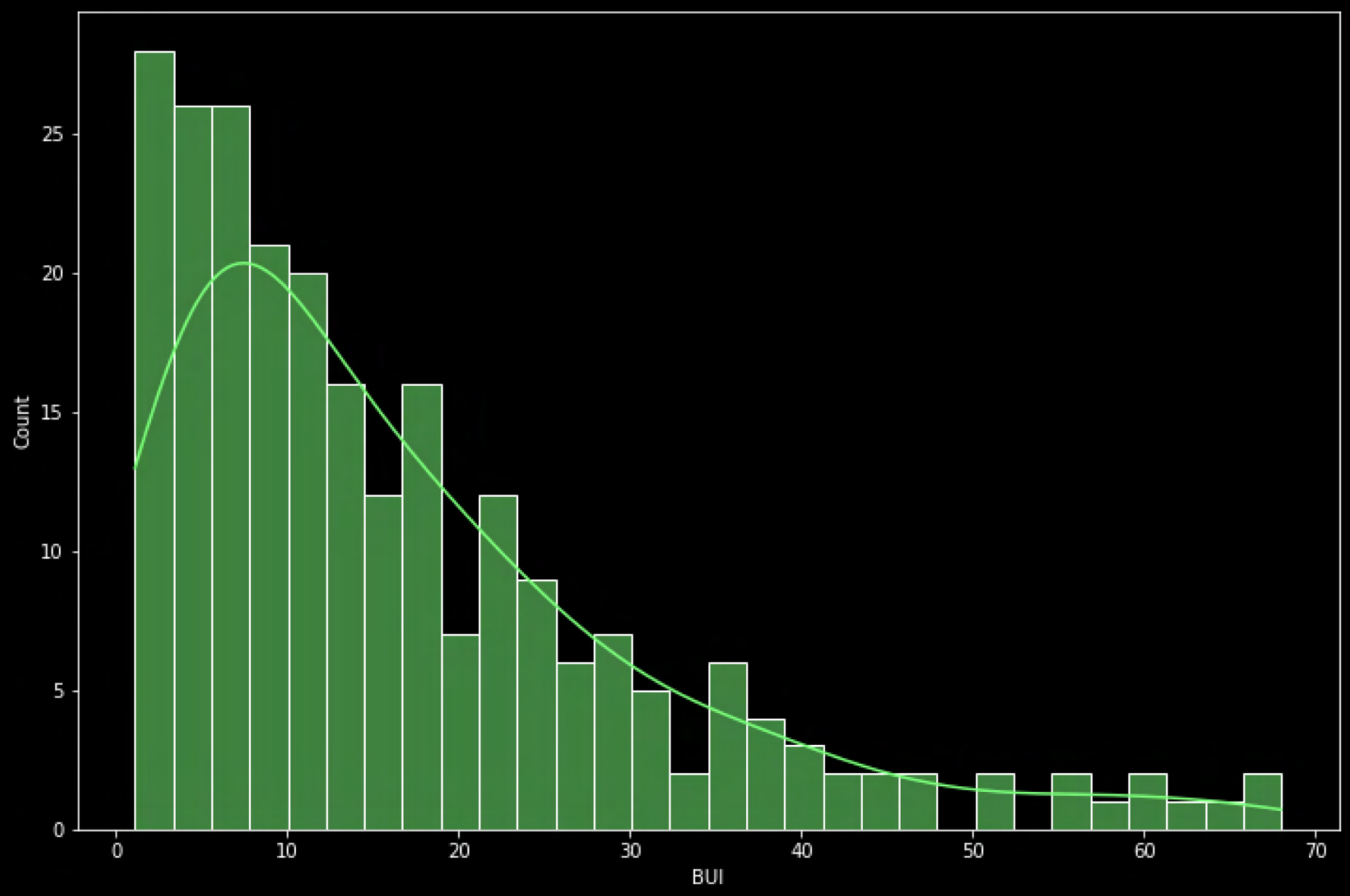
Komal Diwe

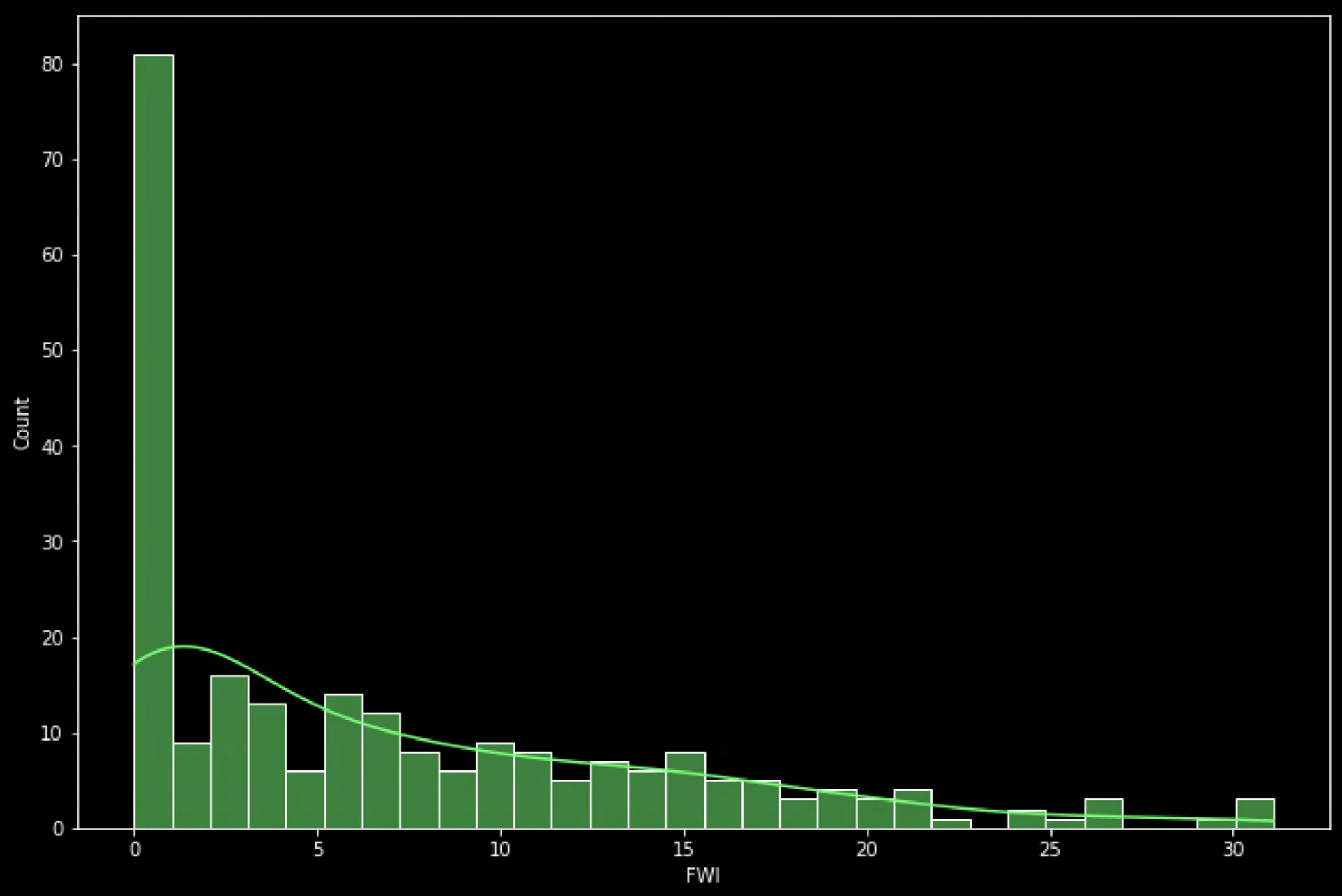

Count
35
30
25
20
15
10
5
0
30
40
50
60
70
80
90
FFMC
Count
30
25
20
15
10
5
0
0
10
20
30
40
50
60
DMC

Komal Diwe


Count
60
50
40
30
20
10
0
0
50
100
150
200
DC
30
25
20
15
10
5
0
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
ISI

Komal Diwe





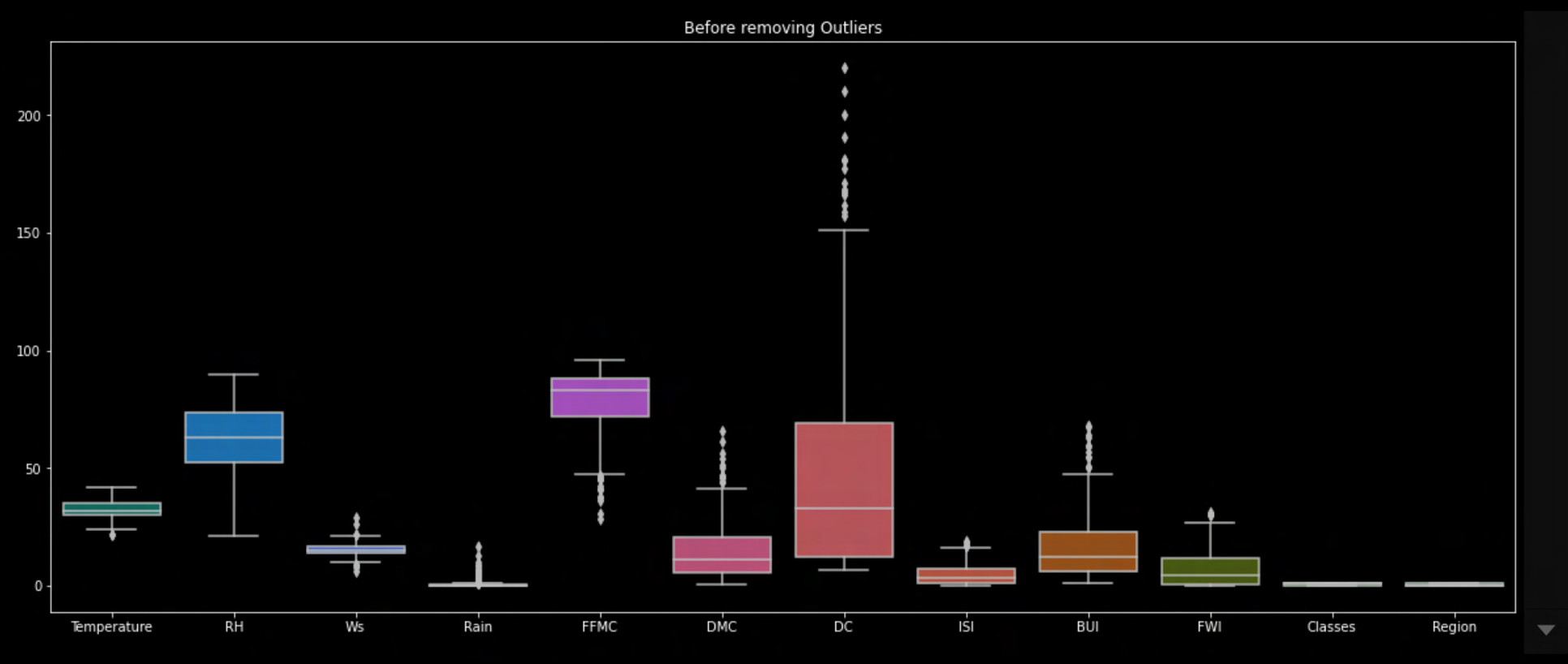
Outliers Handling

Before Removing Outliers

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

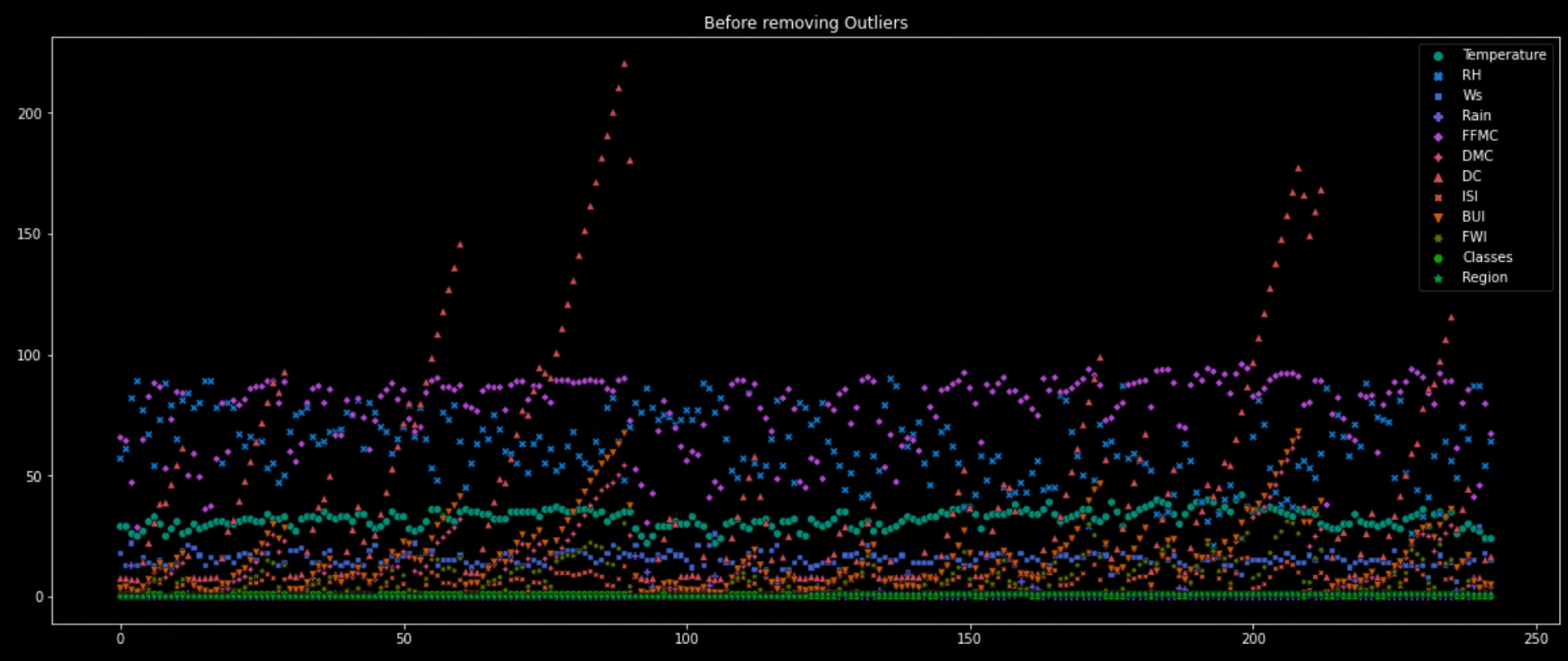
Out[ ]:
```
Text(0.5, 1.0, 'Before removing Outliers')
```

Komal Diwe



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

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



Function to Find upper and lower Boundaries.

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

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

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

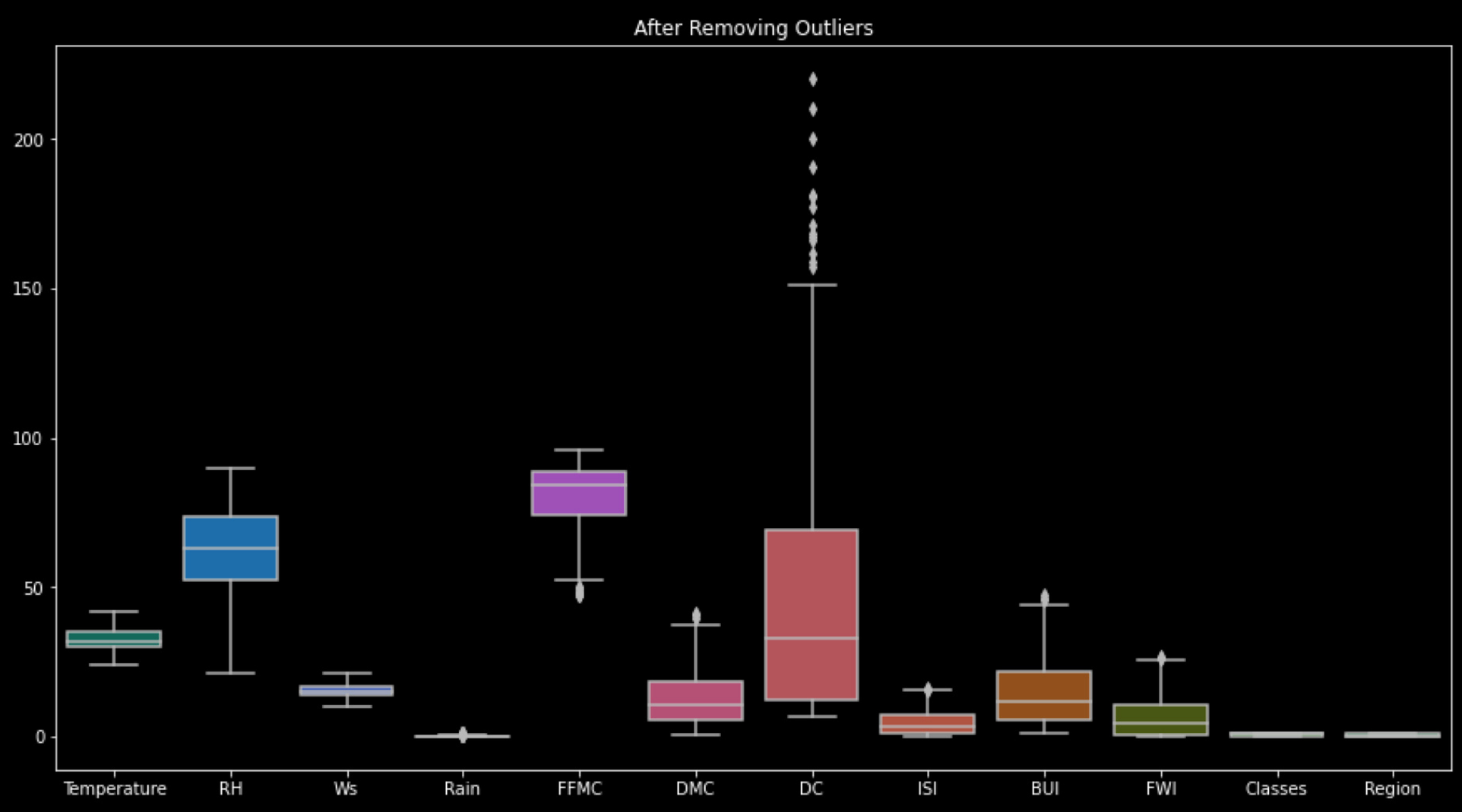
Deletion of Outliers

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

After Removal of Outliers

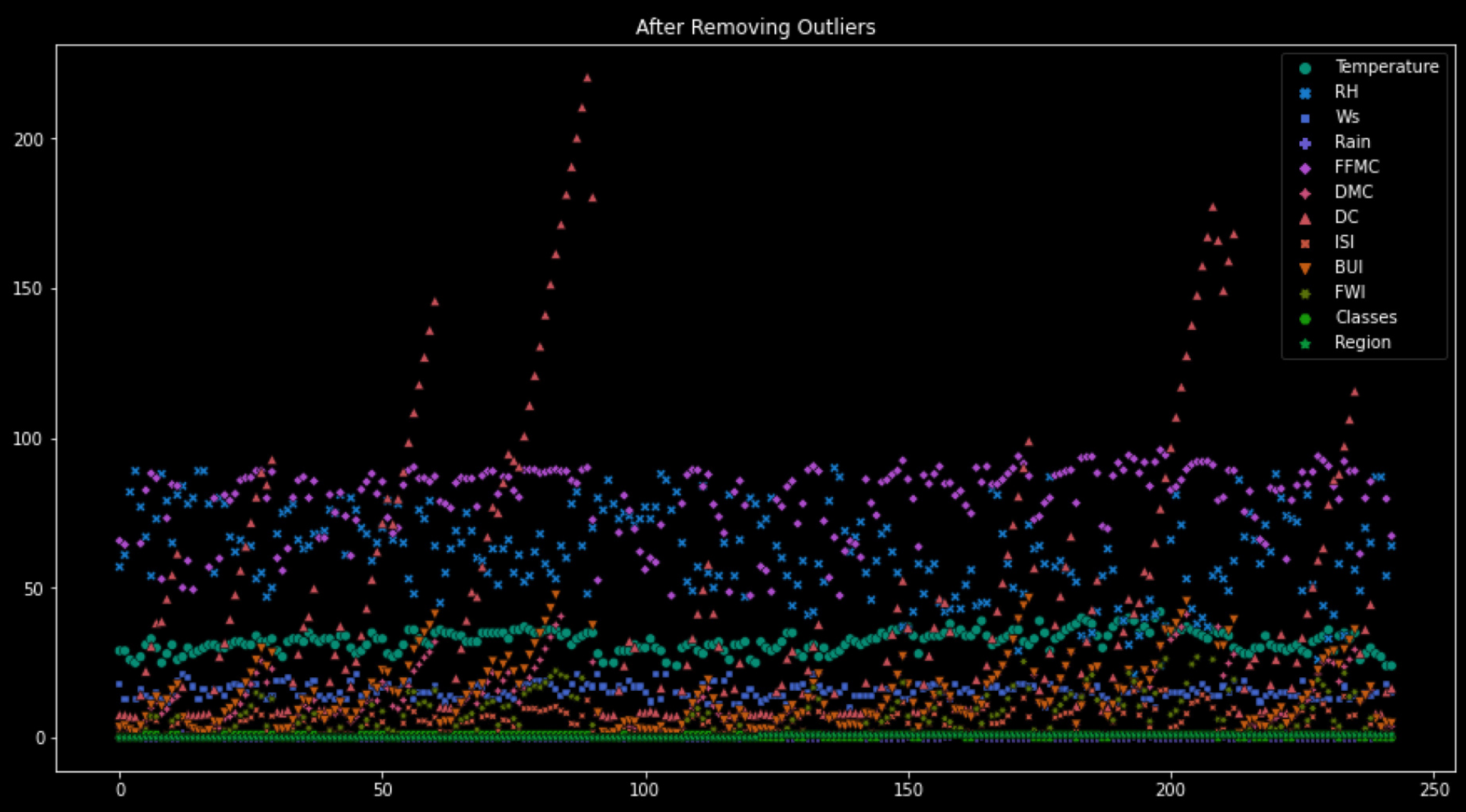
In [ ]:
```
plt.figure(figsize=(15, 8))
sns.boxplot(data=df)
plt.title("After Removing Outliers")
```

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



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

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



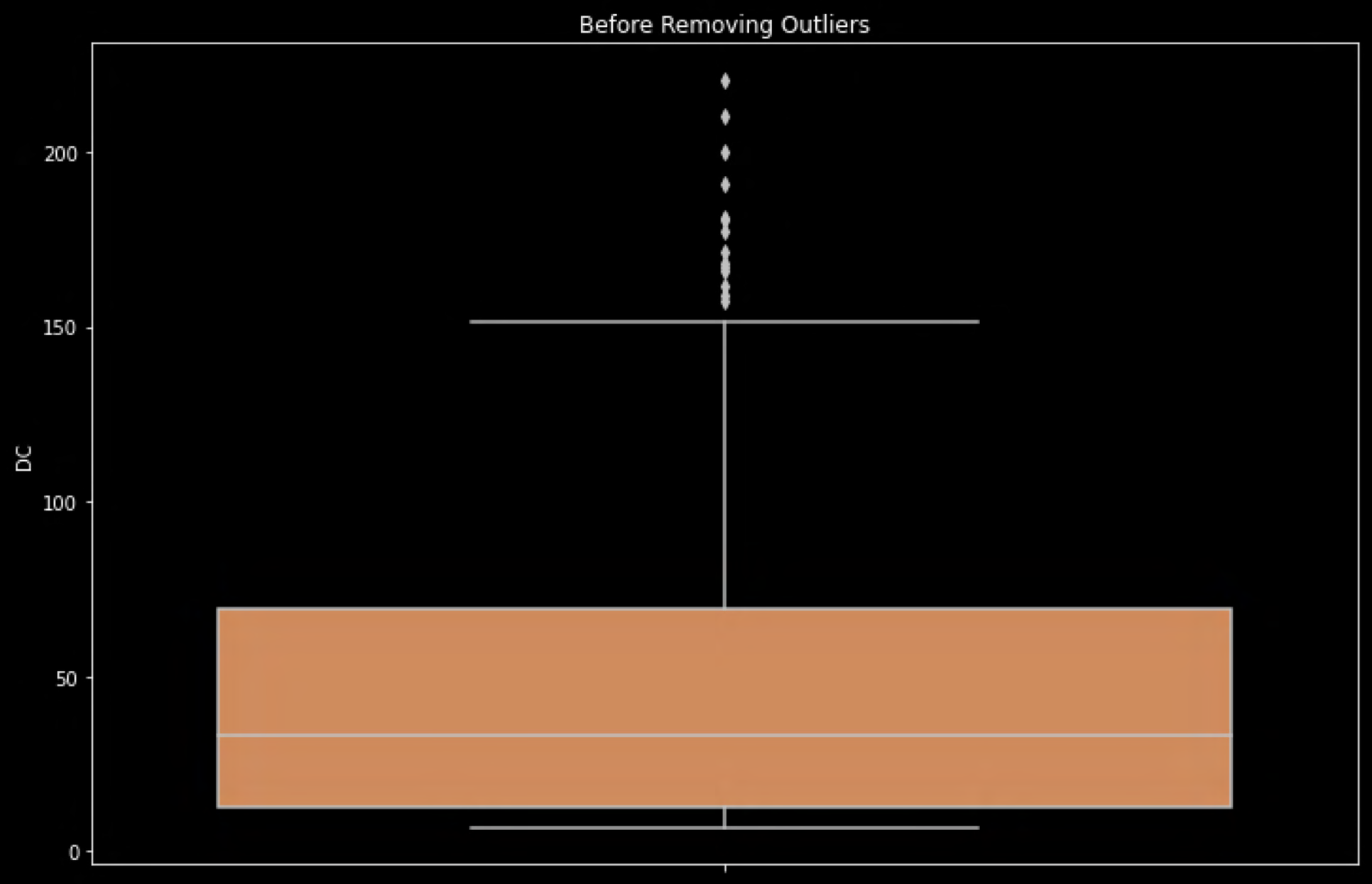
Komal_Diwe

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

Outliers Handling in DC feature

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

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



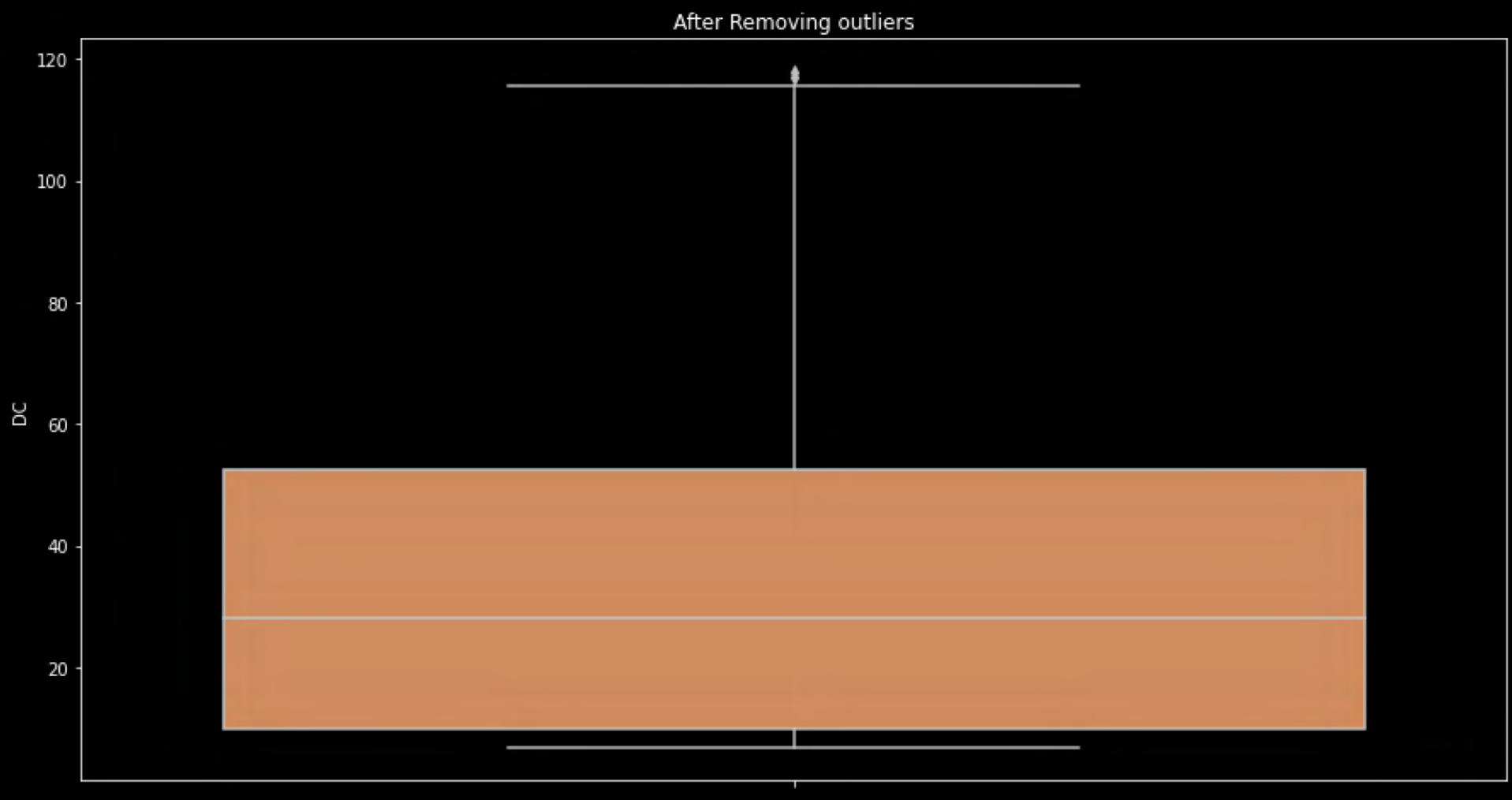
In [ ]:
```
# These are the outliers of DC feature
dc_outliers = df[df['DC'] >= 154]['DC']
dc_outliers
```

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

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

In [ ]:
```
plt.figure(figsize = (15,8))
sns.boxplot(data = df, y = 'DC')
plt.title("After Removing outliers")
```

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



Check null value in each column after removing the outliers

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

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

Fill all the null values with mean

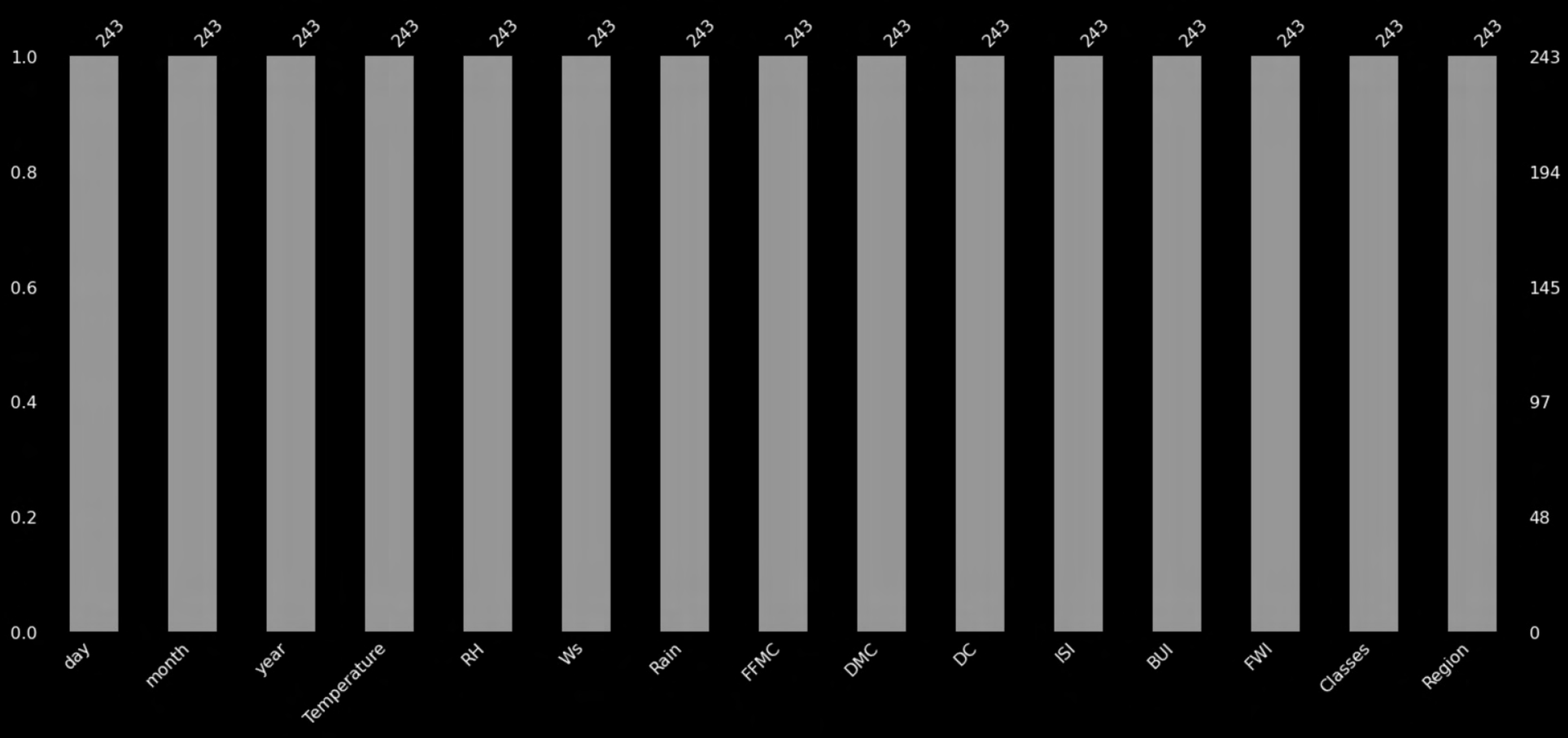
In [ ]:
```
df.fillna(df.mean().round(1), inplace = True)
```

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

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

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



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

Statistical Analysis

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

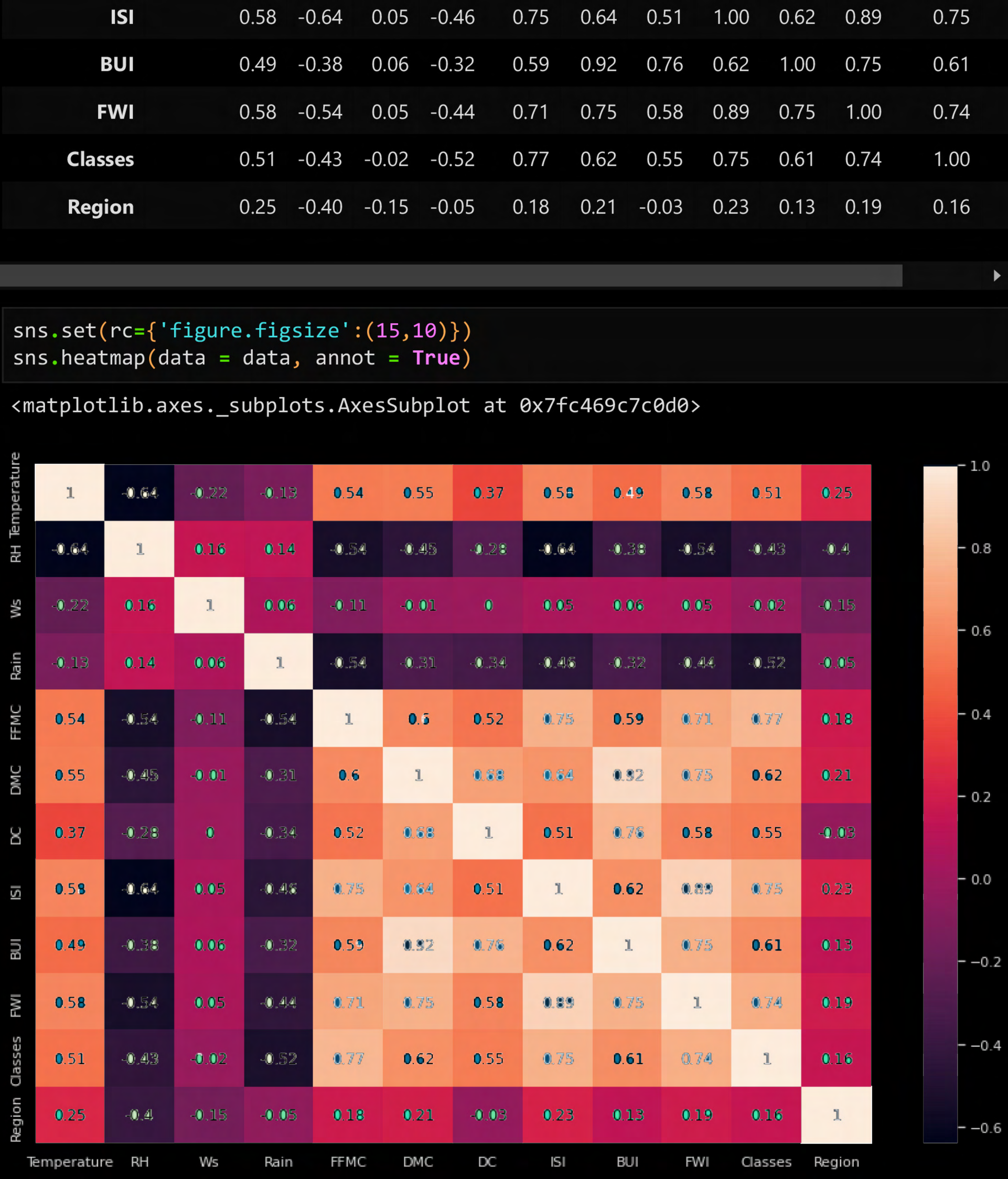
Komal Diwe

Out[ ]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Re
Temperature	1.00	-0.64	-0.22	-0.13	0.54	0.55	0.37	0.58	0.49	0.58	0.51	
RH	-0.64	1.00	0.16	0.14	-0.54	-0.45	-0.28	-0.64	-0.38	-0.54	-0.43	
Ws	-0.22	0.16	1.00	0.06	-0.11	-0.01	0.00	0.05	0.06	0.05	-0.02	
Rain	-0.13	0.14	0.06	1.00	-0.54	-0.31	-0.34	-0.46	-0.32	-0.44	-0.52	
FFMC	0.54	-0.54	-0.11	-0.54	1.00	0.60	0.52	0.75	0.59	0.71	0.77	
DMC	0.55	-0.45	-0.01	-0.31	0.60	1.00	0.68	0.64	0.92	0.75	0.62	
DC	0.37	-0.28	0.00	-0.34	0.52	0.68	1.00	0.51	0.76	0.58	0.55	
ISI	0.58	-0.64	0.05	-0.46	0.75	0.64	0.51	1.00	0.62	0.89	0.75	
BUI	0.49	-0.38	0.06	-0.32	0.59	0.92	0.76	0.62	1.00	0.75	0.61	
FWI	0.58	-0.54	0.05	-0.44	0.71	0.75	0.58	0.89	0.75	1.00	0.74	
Classes	0.51	-0.43	-0.02	-0.52	0.77	0.62	0.55	0.75	0.61	0.74	1.00	
Region	0.25	-0.40	-0.15	-0.05	0.18	0.21	-0.03	0.23	0.13	0.19	0.16	

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

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



Observations:

- BUI and DMC are 92% positively correlated
- FWI and ISI are 89% positively correlated

Komal Diwe

- No features are more than 95% positively correlated, therefore we cannot drop any feature

Model Building

Independent Variable Vs Target Variable distribution.

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

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

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

Out[ ]:

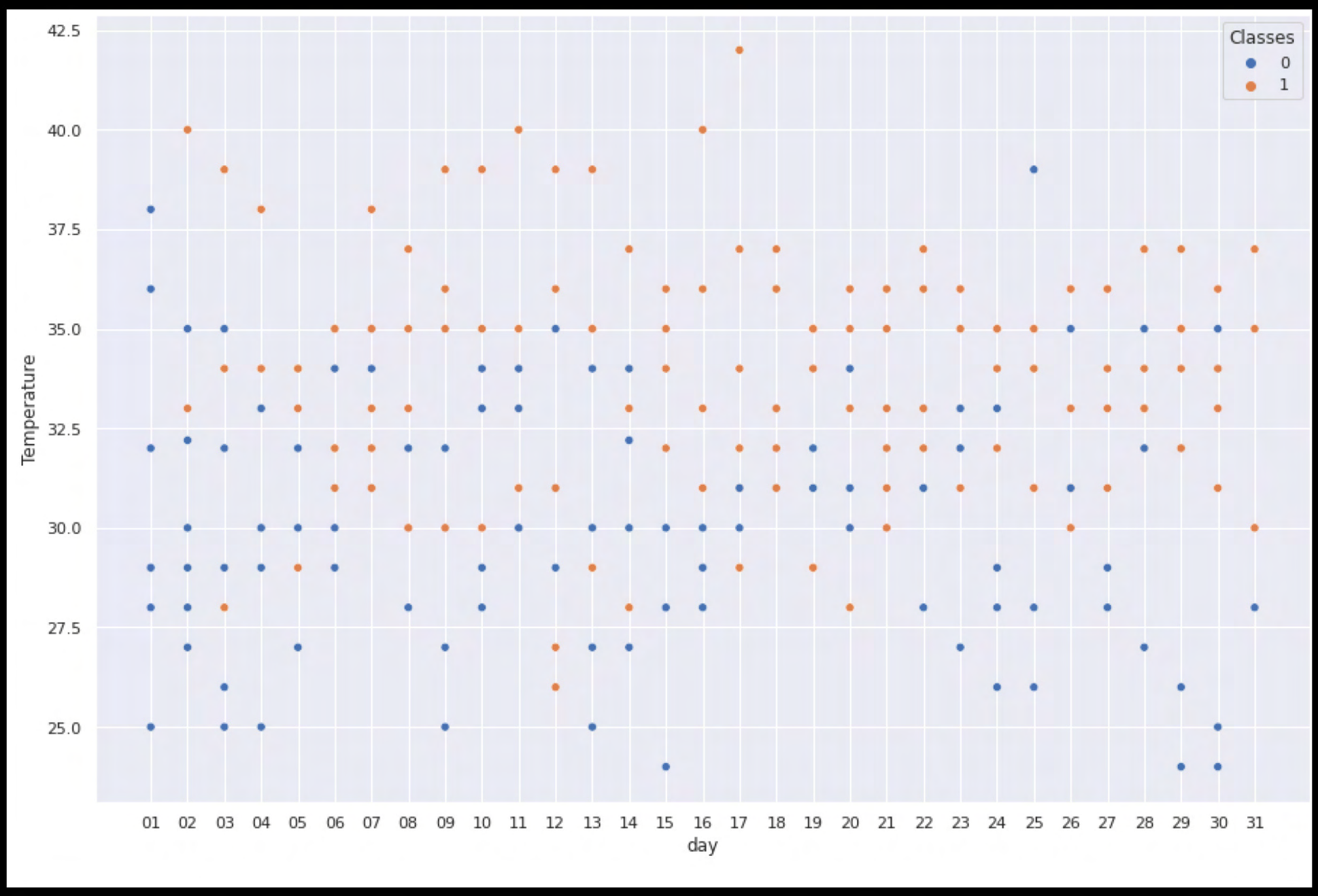
	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29.0	57	18.0	0.0	65.7	3.4	7.6	1.3	3.4	0.5	0
1	02	06	2012	29.0	61	13.0	0.2	64.4	4.1	7.6	1.0	3.9	0.4	0
2	03	06	2012	26.0	82	15.5	0.2	80.0	2.5	7.1	0.3	2.7	0.1	0
3	04	06	2012	25.0	89	13.0	0.2	80.0	1.3	6.9	0.0	1.7	0.0	0
4	05	06	2012	27.0	77	16.0	0.0	64.8	3.0	14.2	1.2	3.9	0.5	0

Scatterplot day Vs temperature

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

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

Komal Diwe



Regression Plot

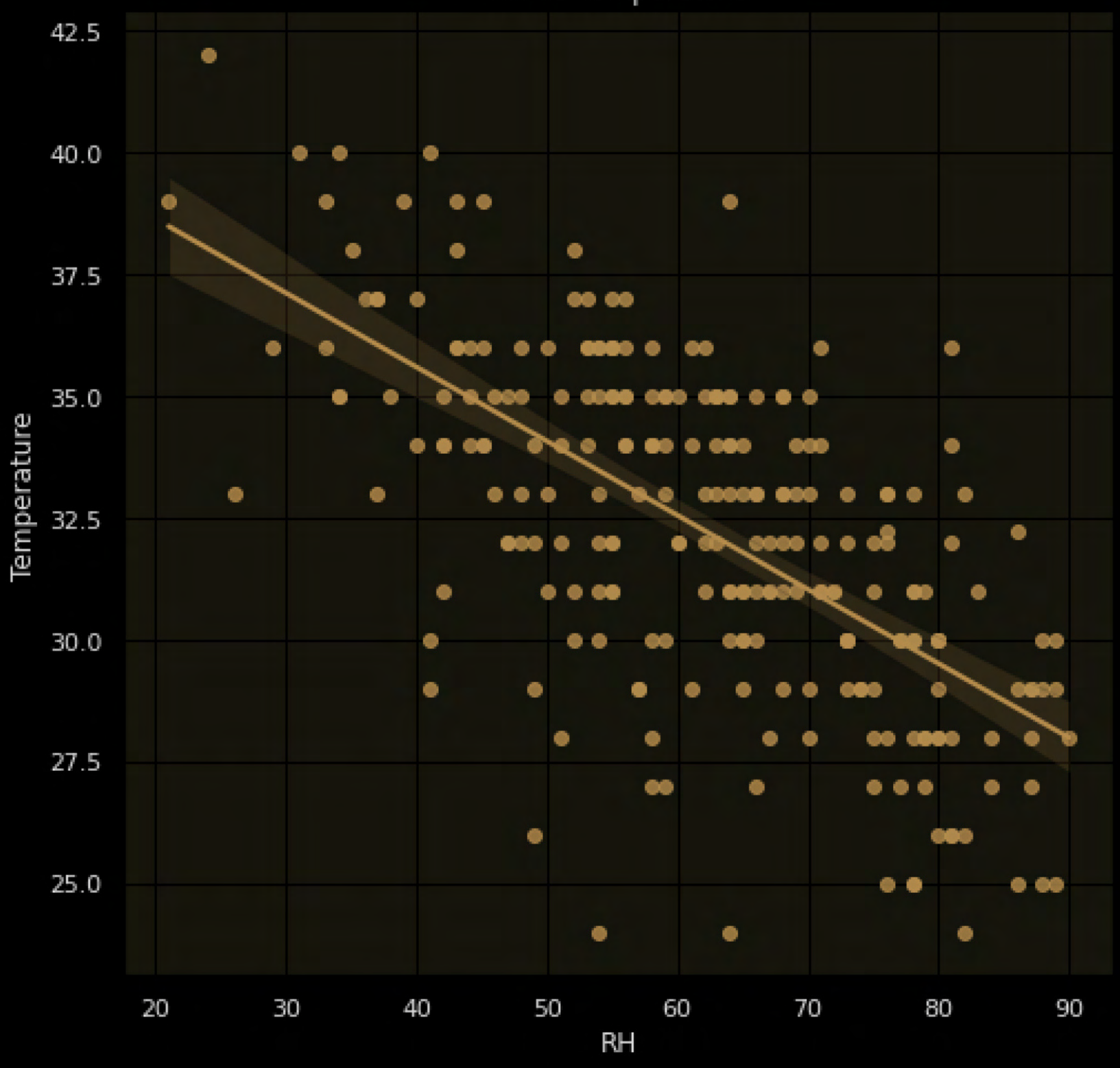
In [ ]:
```
consider_feature = [feature for feature in df.columns if feature not in ['Temperatu
consider_feature
```

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

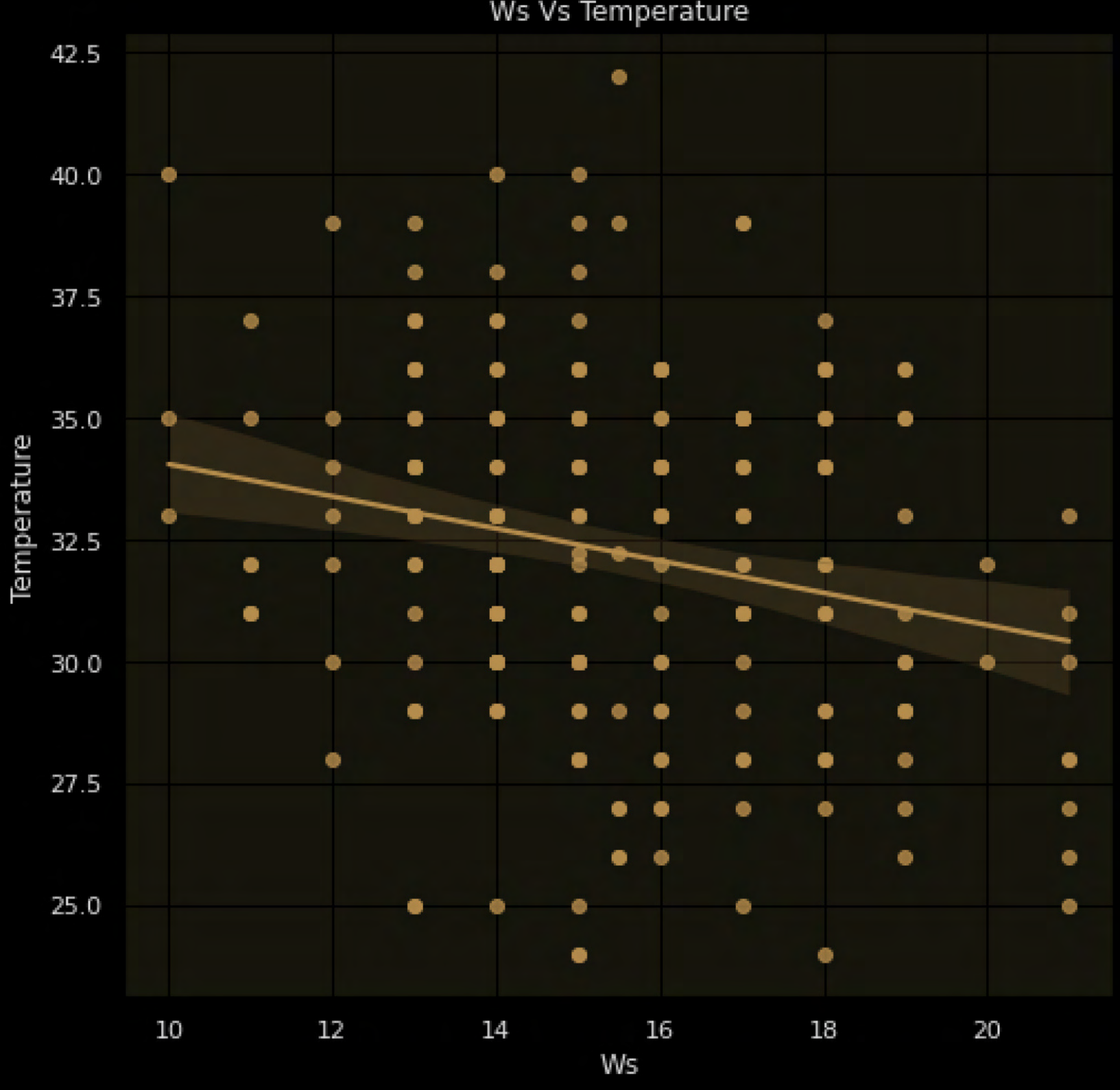
In [ ]:
```
for feature in consider_feature:
  sns.set(rc={'figure.figsize':(8,8)})
  sns.regplot(x = df[feature], y = df['Temperature'])
  plt.xlabel(feature)
  plt.ylabel('Temperature')
  plt.title("{} Vs Temperature".format(feature))
  plt.show()
```

Komal Diwe

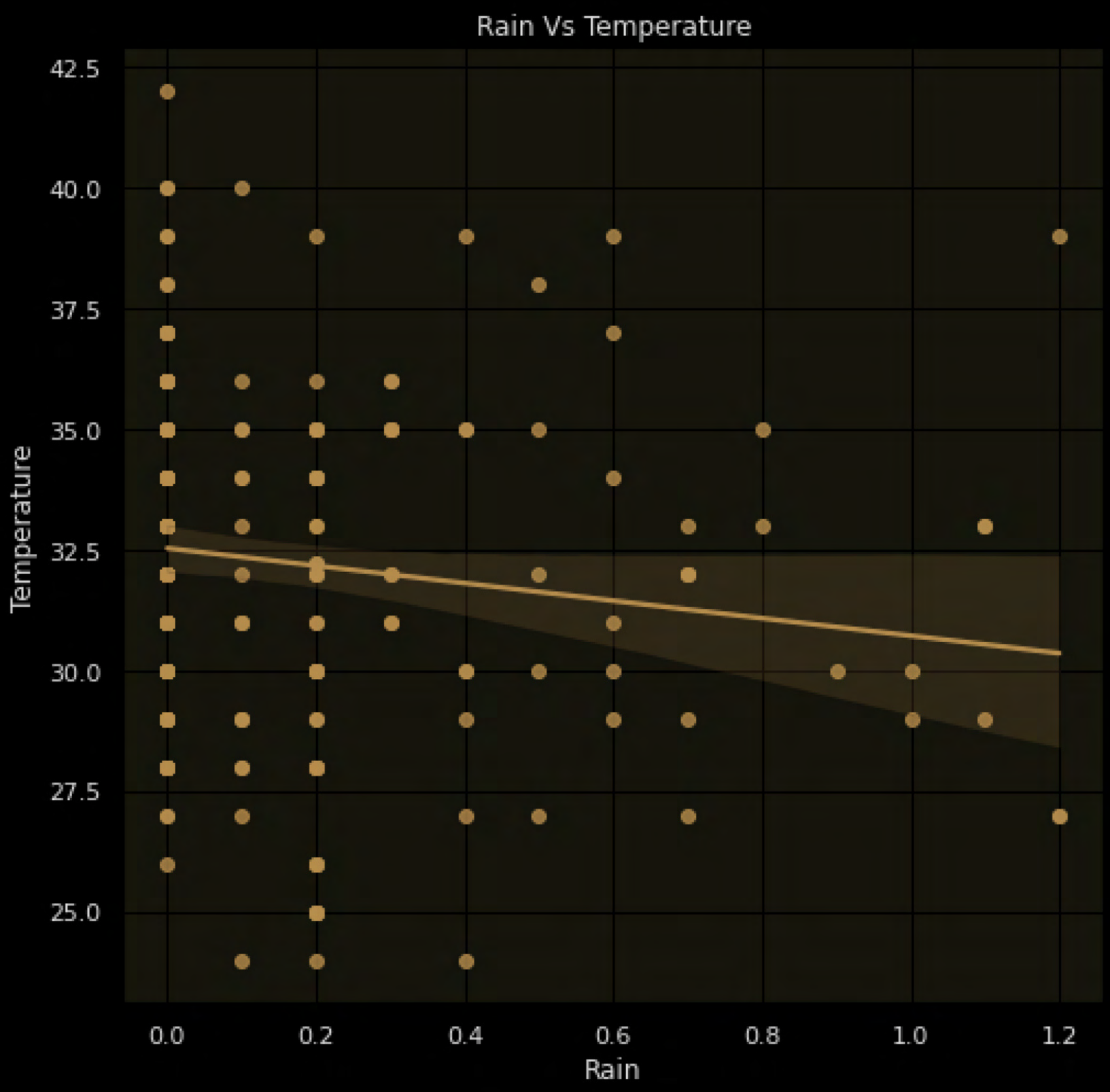
RH Vs Temperature



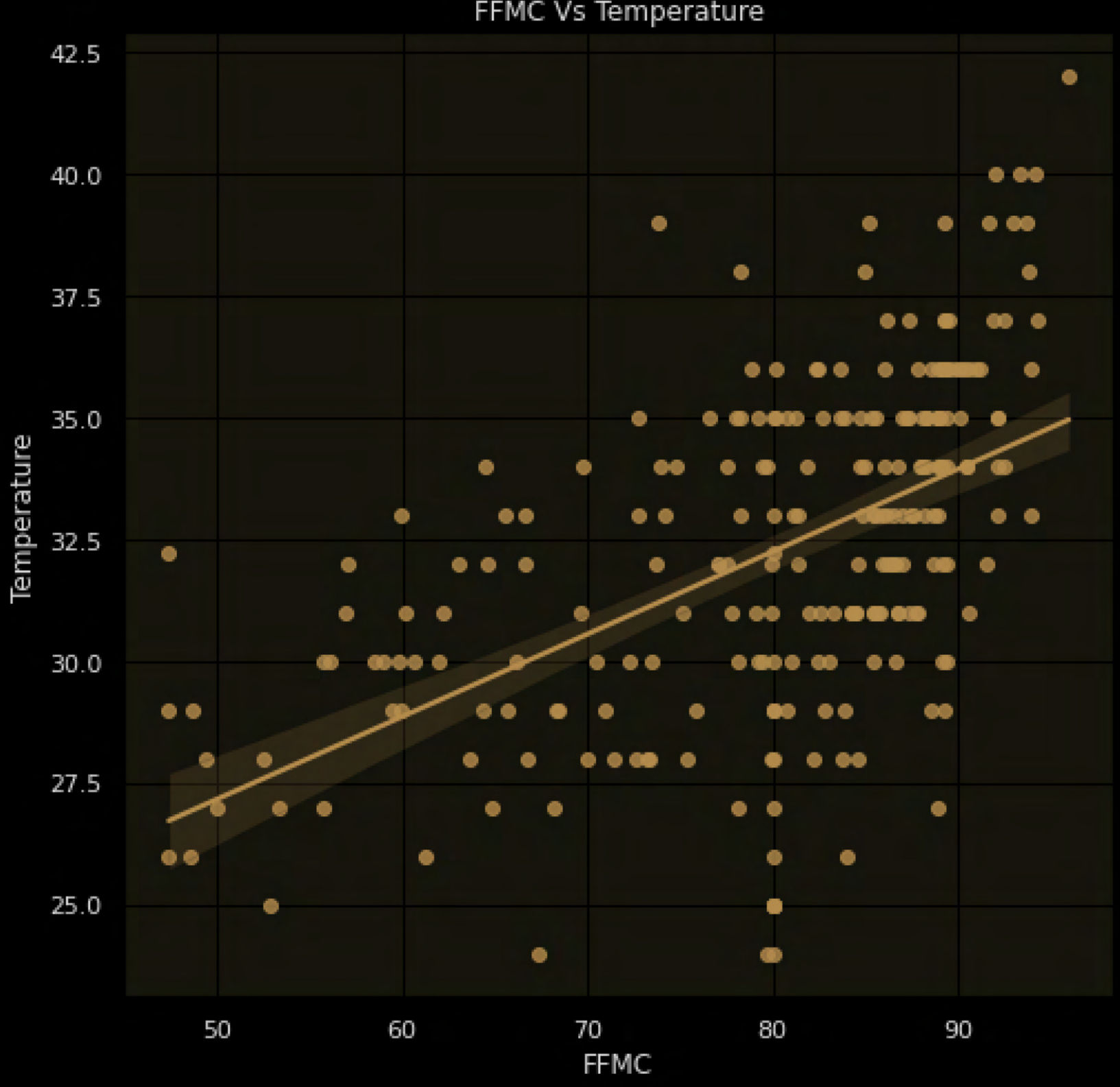
Ws Vs Temperature



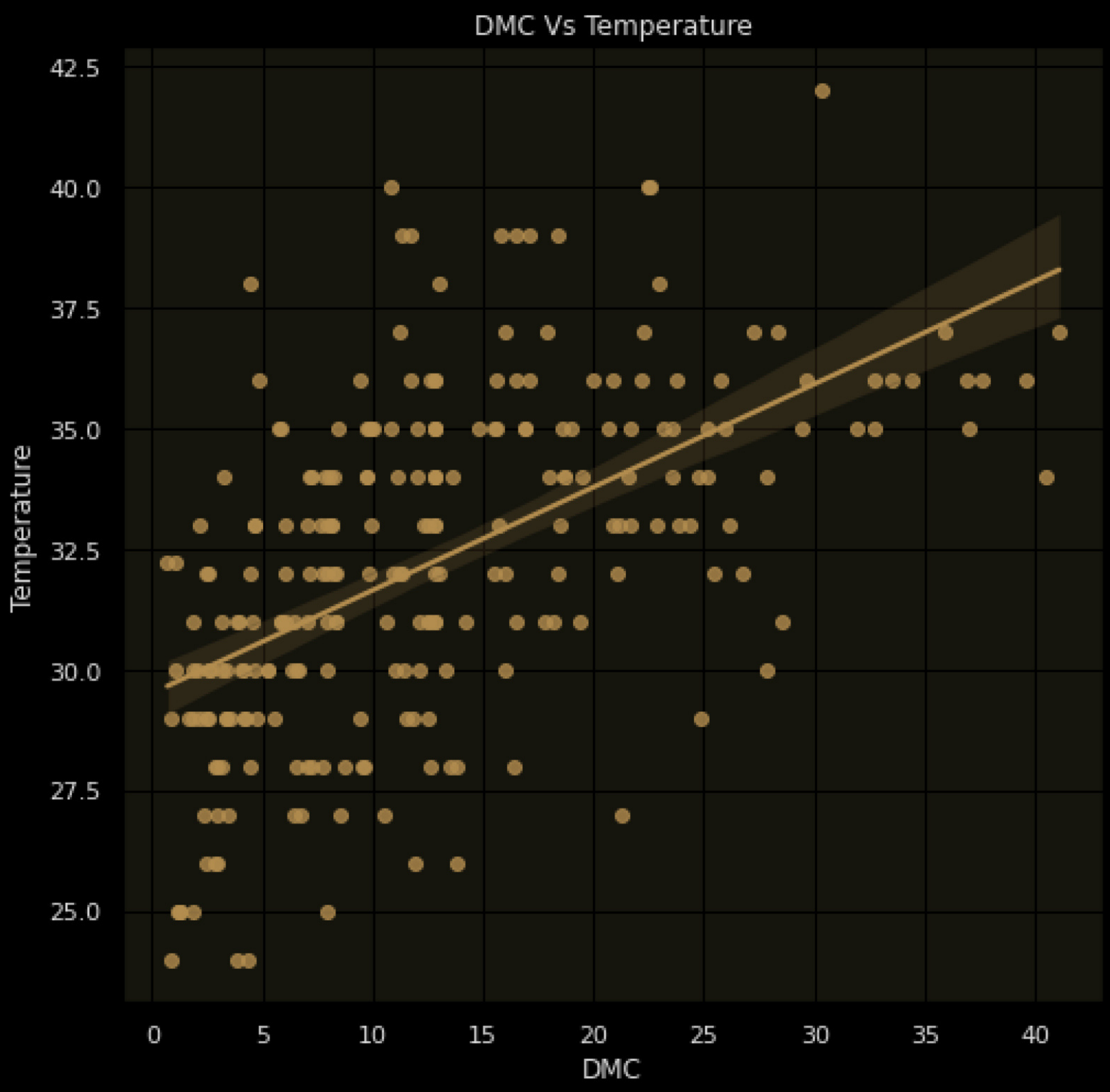
Komal_Diwe



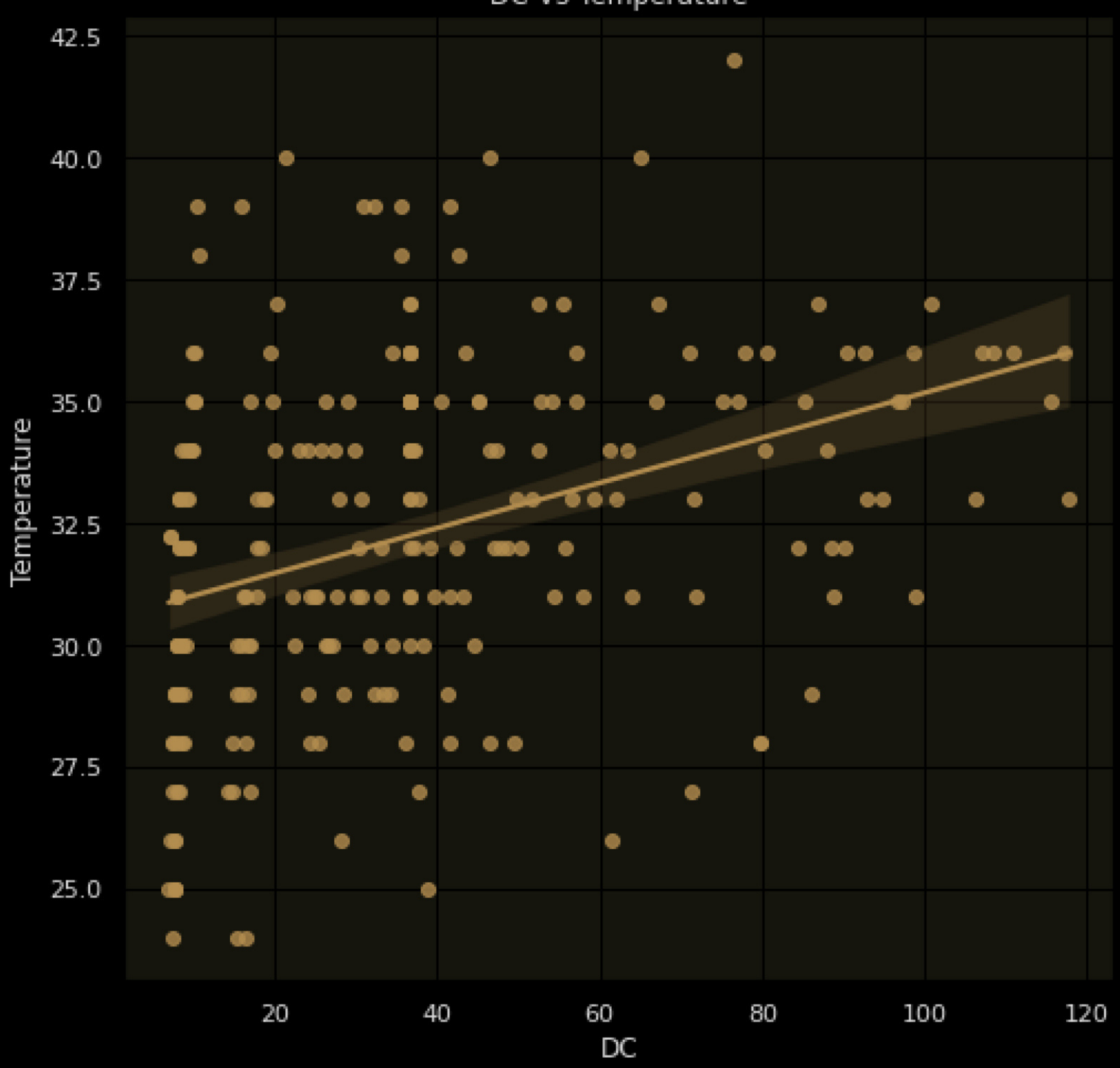
FFMC Vs Temperature



Komal Diwe

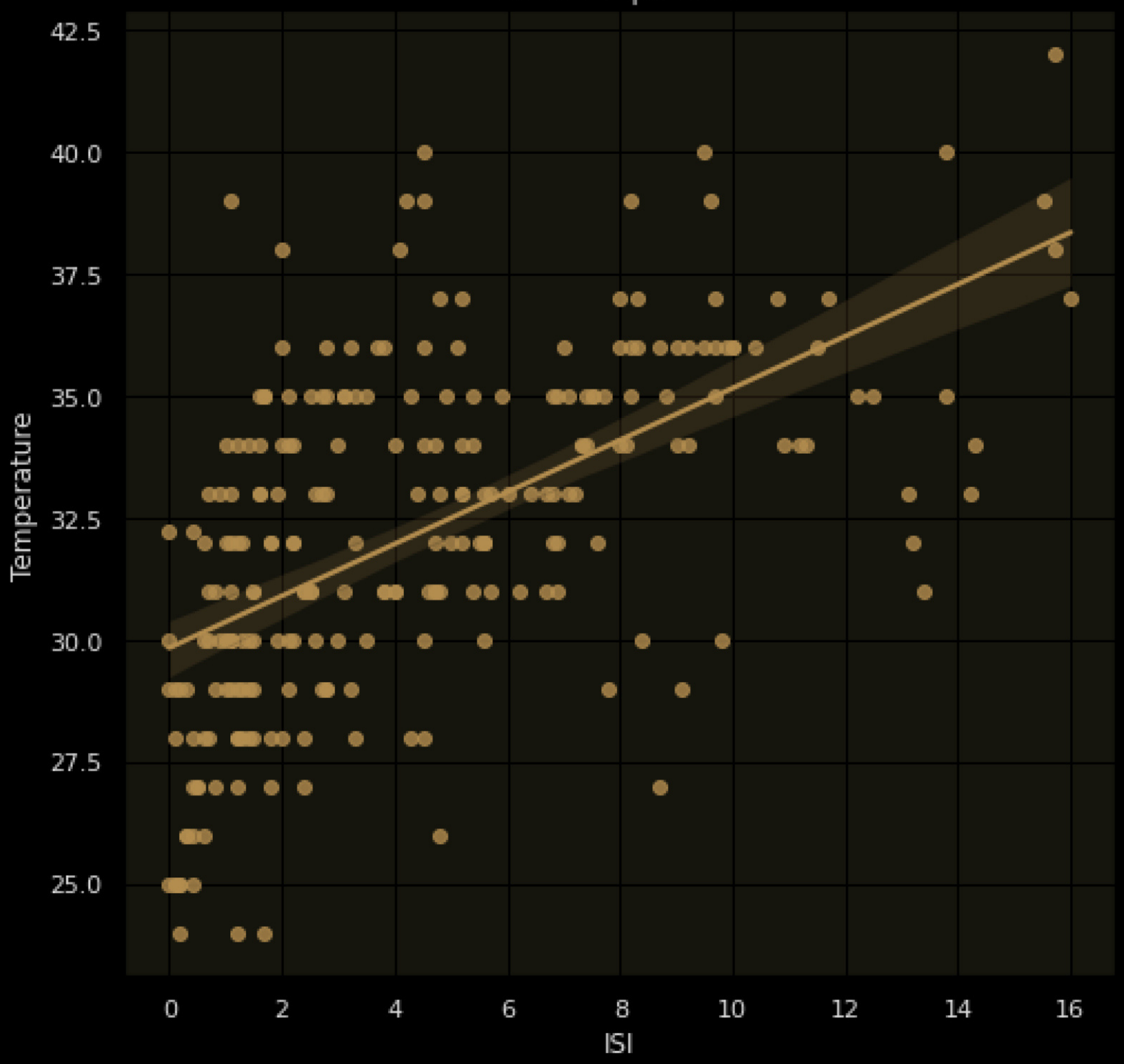


DC Vs Temperature

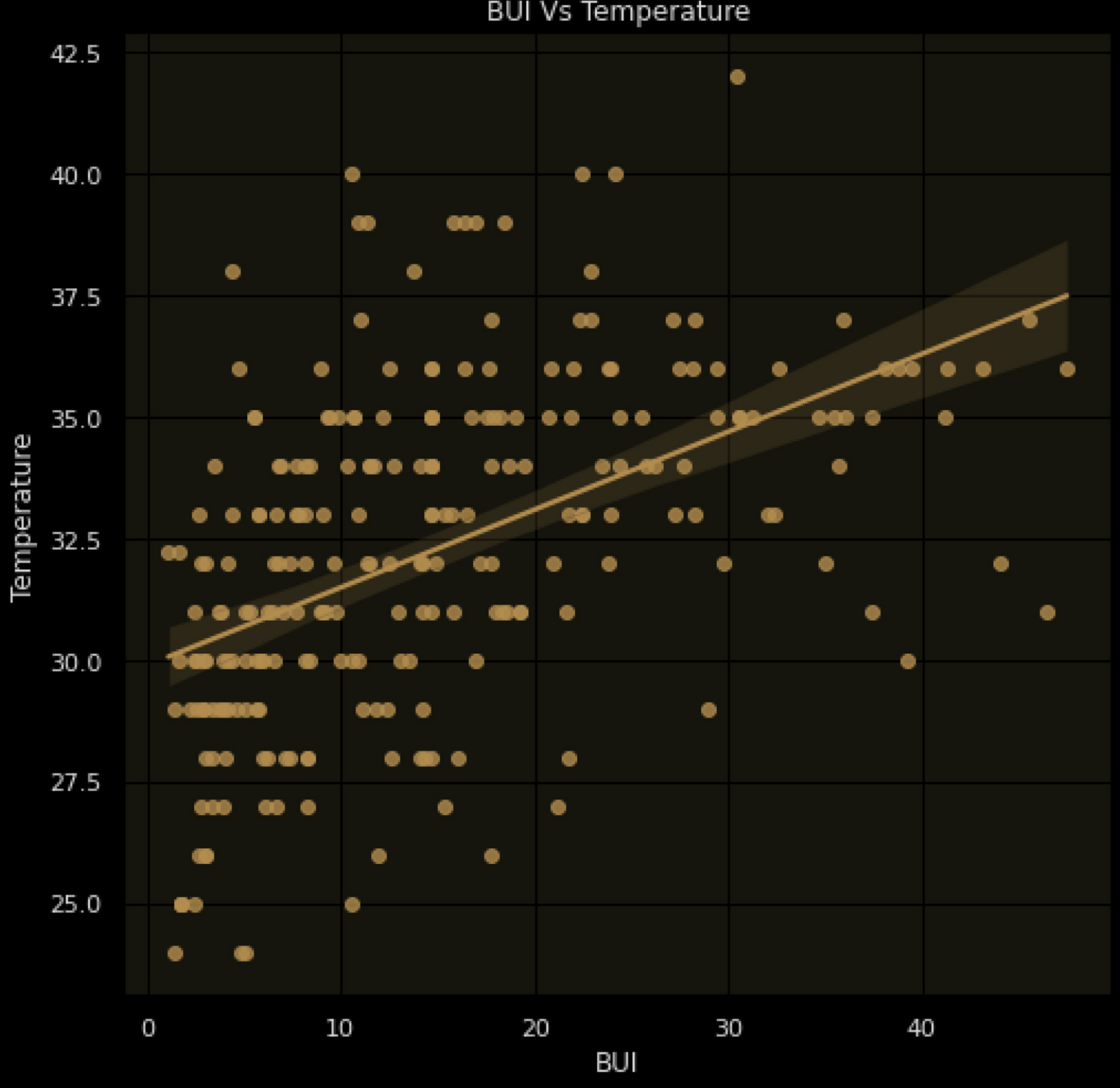


Komal_Diwe

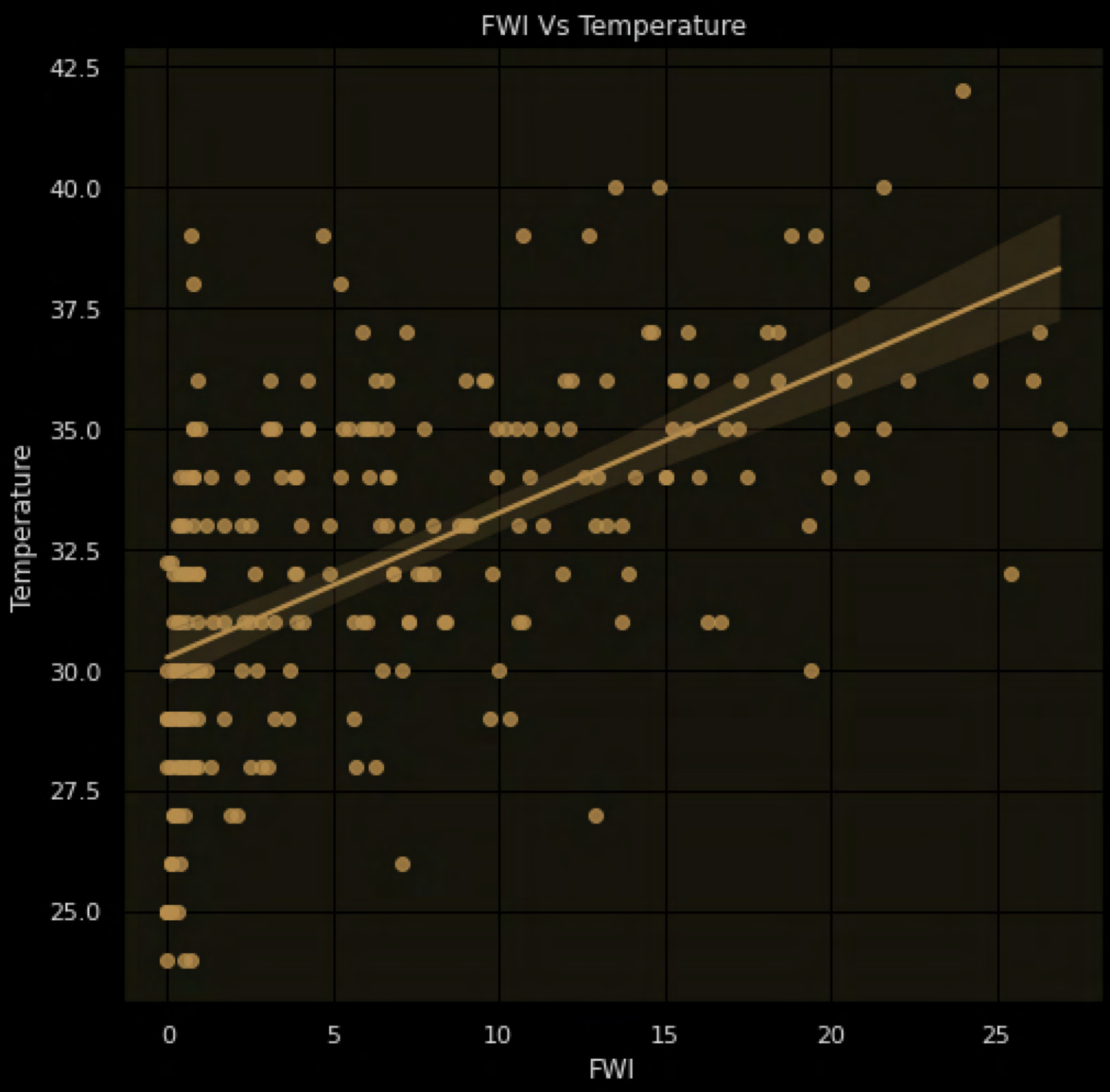
ISI Vs Temperature



BUI Vs Temperature



Komal_Diwe



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

Segregate Dependent and Independent feature

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

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

Out[ ]:

	RH	Ws	Rain	FFMC	DMC	ISI	DC	FWI	Classes	Region
0	57	18.0	0.0	65.7	3.4	1.3	7.6	0.5	0	0.0
1	61	13.0	0.2	64.4	4.1	1.0	7.6	0.4	0	0.0
2	82	15.5	0.2	80.0	2.5	0.3	7.1	0.1	0	0.0
3	89	13.0	0.2	80.0	1.3	0.0	6.9	0.0	0	0.0
4	77	16.0	0.0	64.8	3.0	1.2	14.2	0.5	0	0.0

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

Komal Diwe

Out[ ]:

	Temperature
0	29.0
1	29.0
2	26.0
3	25.0
4	27.0

Split the data into training and testing dataset

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

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

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

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

In [ ]:
```
# here using only transform to avoid data leakage
# (traing mean and training standard deviation will be used for standard isolation

X_test = scaler.transform(X_test)
```

Linear Regression Model

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

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

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

Out[ ]:
```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
9    0
dtype: int64
```

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

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

Komal Dive

Printing co-efficients and intercept of best fit hyperplane

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

```
 Co-efficient of Independent features is [[-1.62572989 -0.60047117  0.28921192 -0.
05938927  0.76367662  0.00760549
  -0.15916525  0.43191456  0.62332559 -0.26791113]]
Intercept of best fit hyper plane is [32.1617284]
```

Prdiction of Test data

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

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

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

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

Out[ ]:

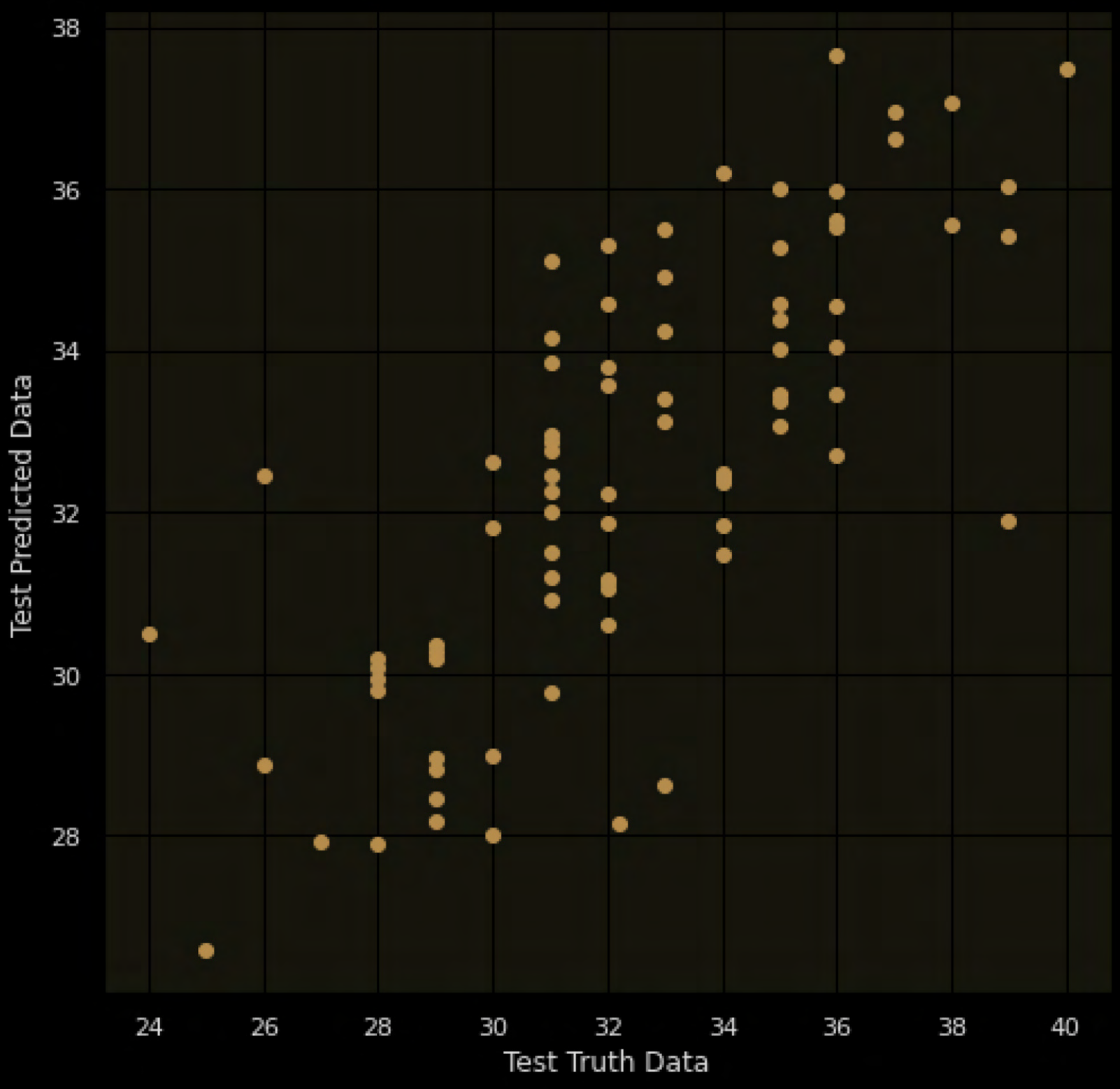
	Temperature
24	-1.870121
6	-1.236611
152	-2.178387
232	1.523195
238	-2.633268

Validation of Linear Regression assumptions

Linear Relationship

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

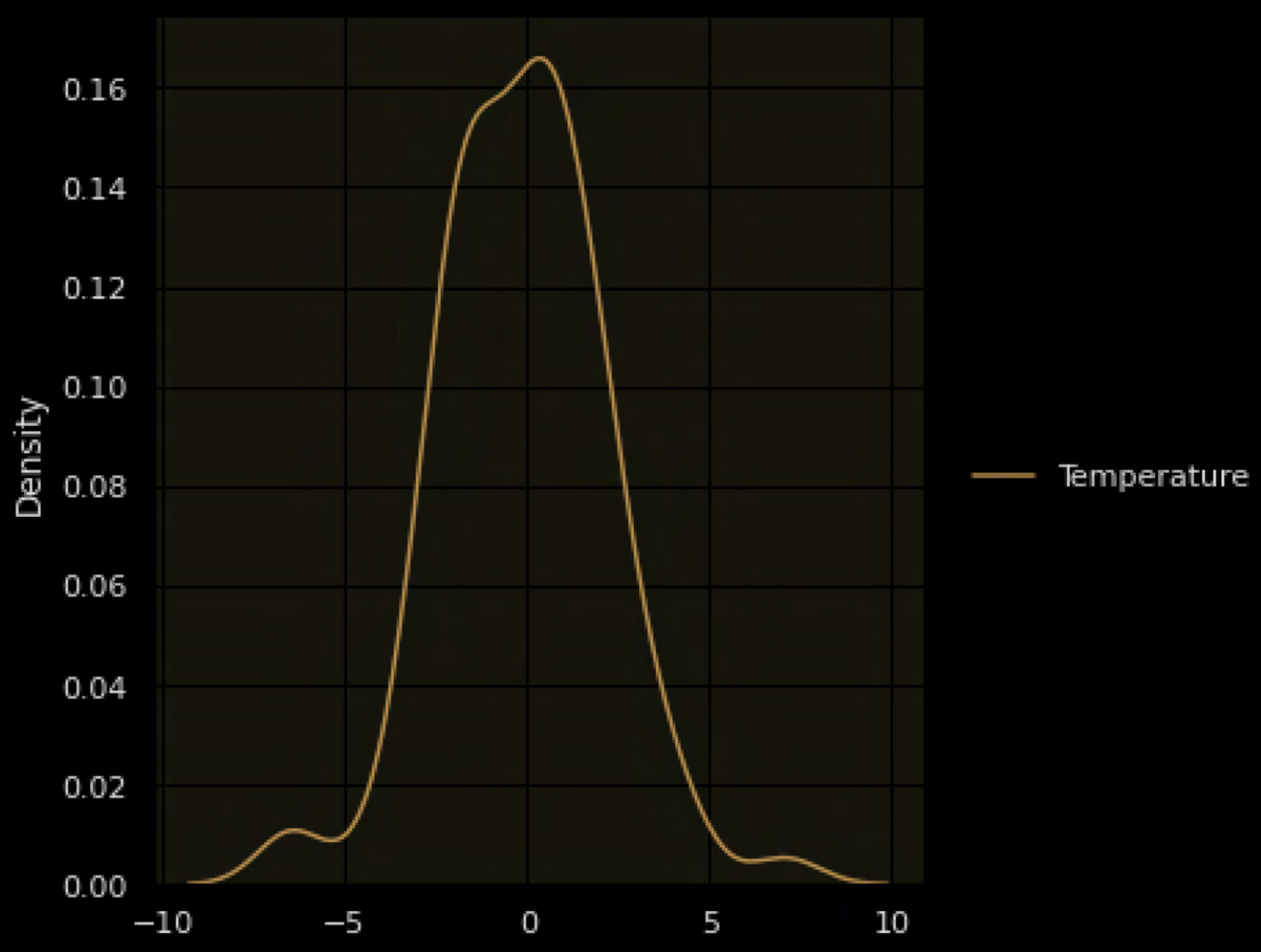
Out[ ]:
```
Text(0, 0.5, 'Test Predicted Data')
```



Residuals should be normally distributed

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

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

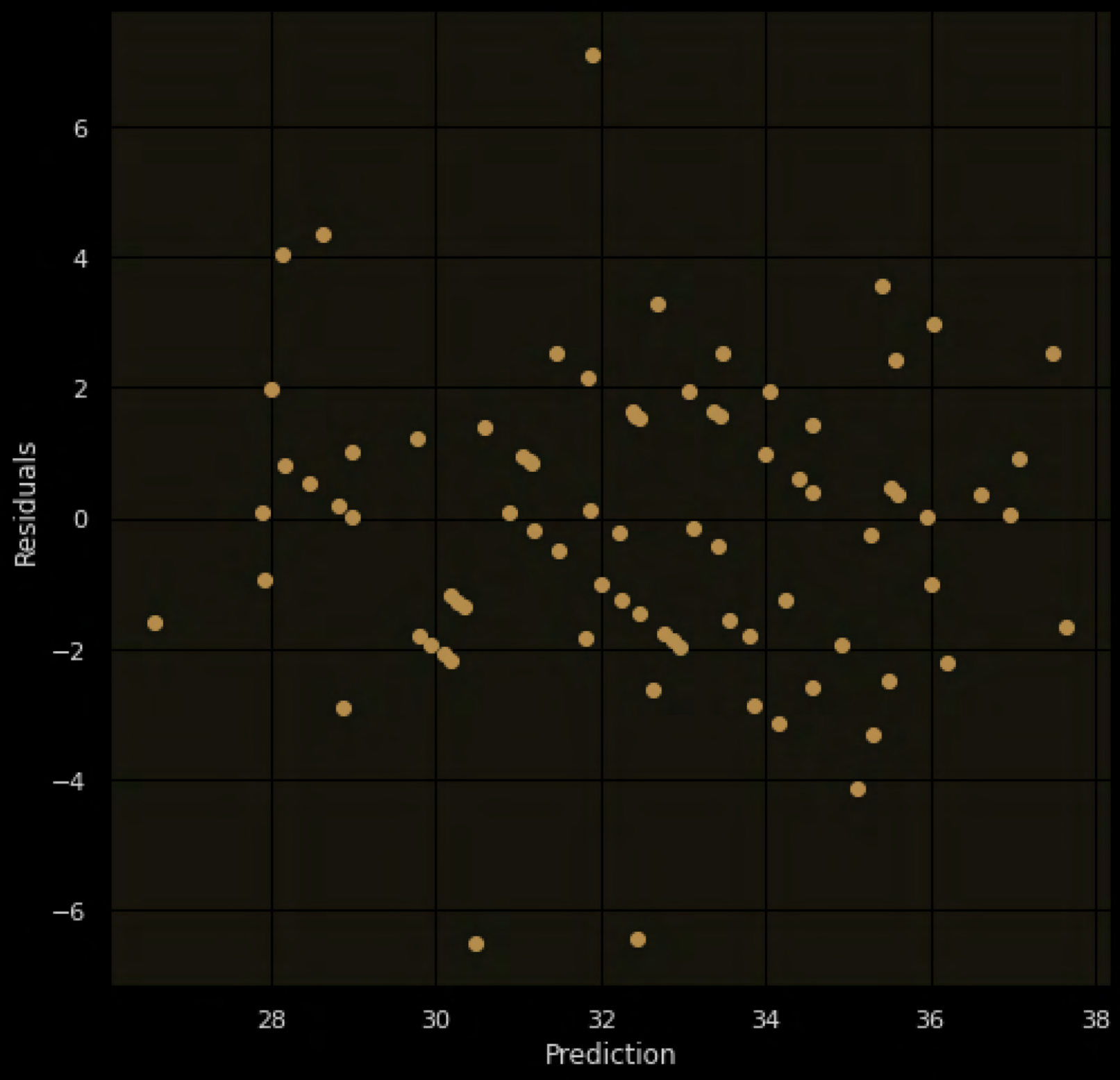


Residual and Predicted values should follow Uniform Distribution.

Komal Dive

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

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



Cost Function

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

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

Performance Metrics

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

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

Ridge Regression Model

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

```
ridge_reg
```

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

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

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

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

```
Co-efficients of Independent features is [[-1.60178318 -0.59652426  0.29122038 -0.
04397446  0.75428229  0.0345659
  -0.15042974  0.41984204  0.606996   -0.25830502]]
Intercept of best fit hyper plane is [32.1617284]
```

Prediction of Test data

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

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

Out[ ]:

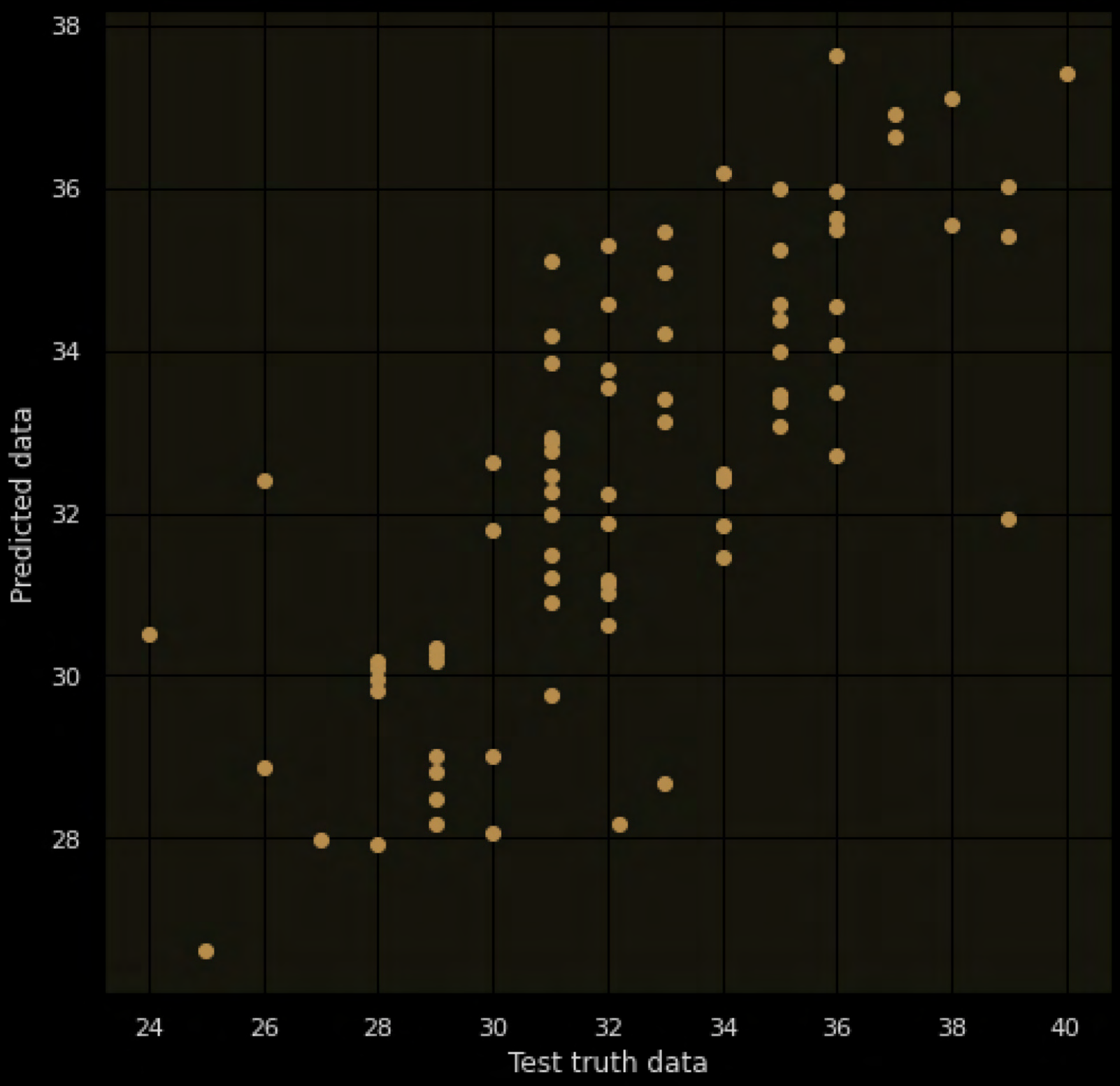
	Temperature
24	-1.866884
6	-1.217160
152	-2.170534
232	1.510022
238	-2.634740

Validation of Ridge Regression Assumptions

Linear Relationship

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

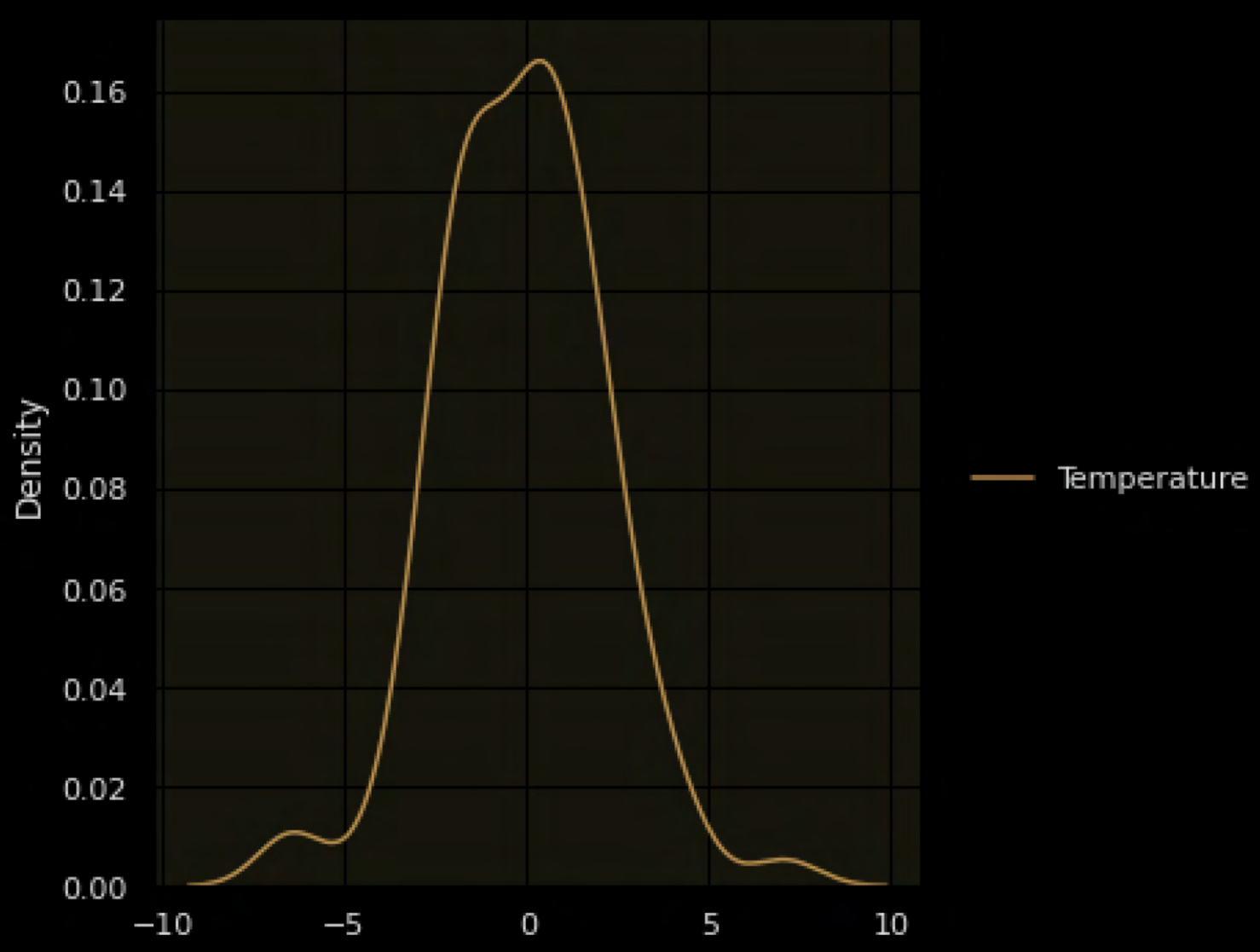
Out[ ]:
```
Text(0, 0.5, 'Predicted data')
```



Residual should be Normally Distributed

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

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



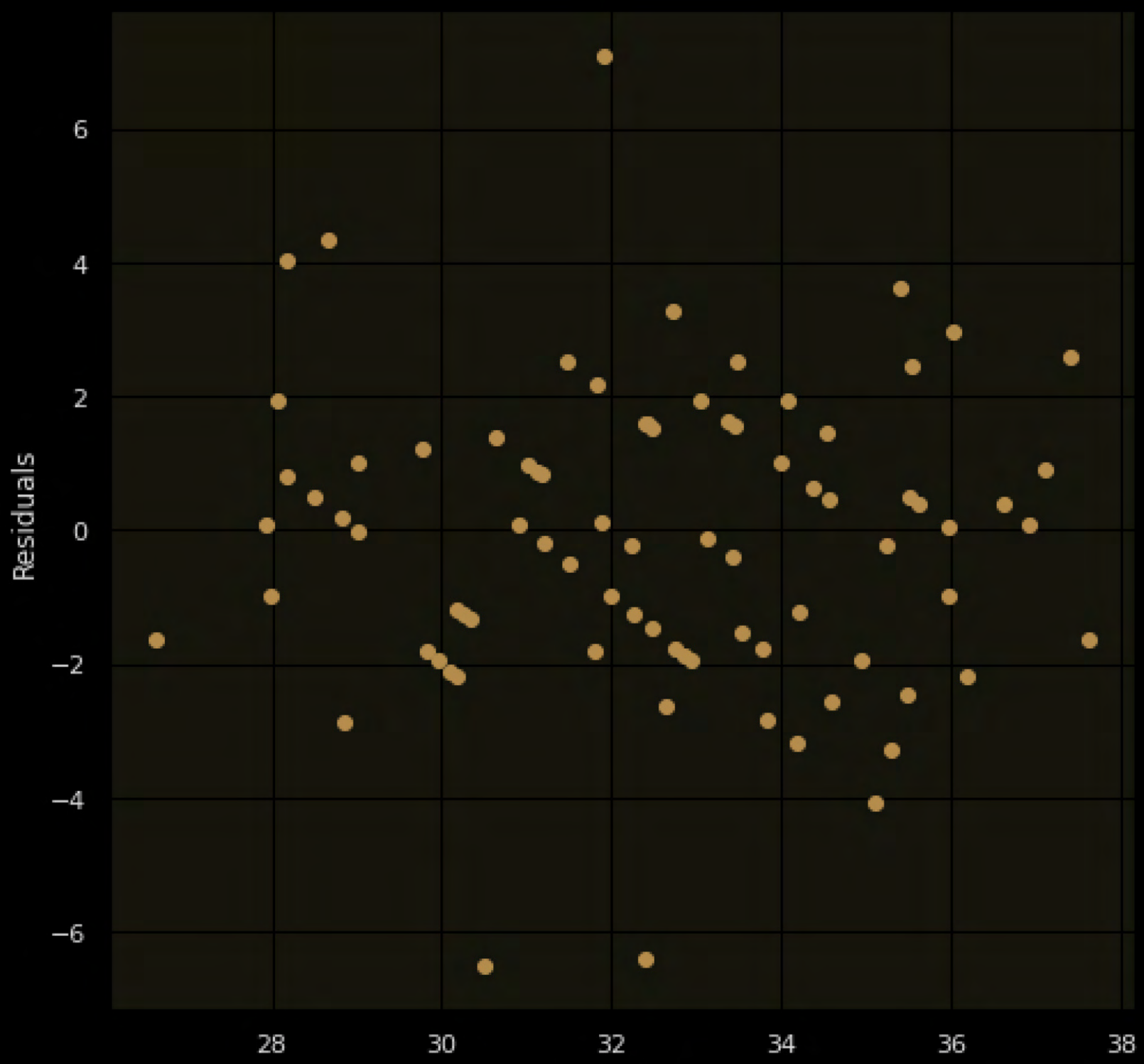
Residual and Predicted values should follow Uniform Distribution

---

Komal Diwe

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

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



Cost Function Values

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

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

Performance Metrics

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

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

Lasso Regression Model

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

```
lasso_reg
```

Out[ ]:
```
Lasso()
```

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

Out[ ]:
```
Lasso()
```

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

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

Prediction of Test data

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

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

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

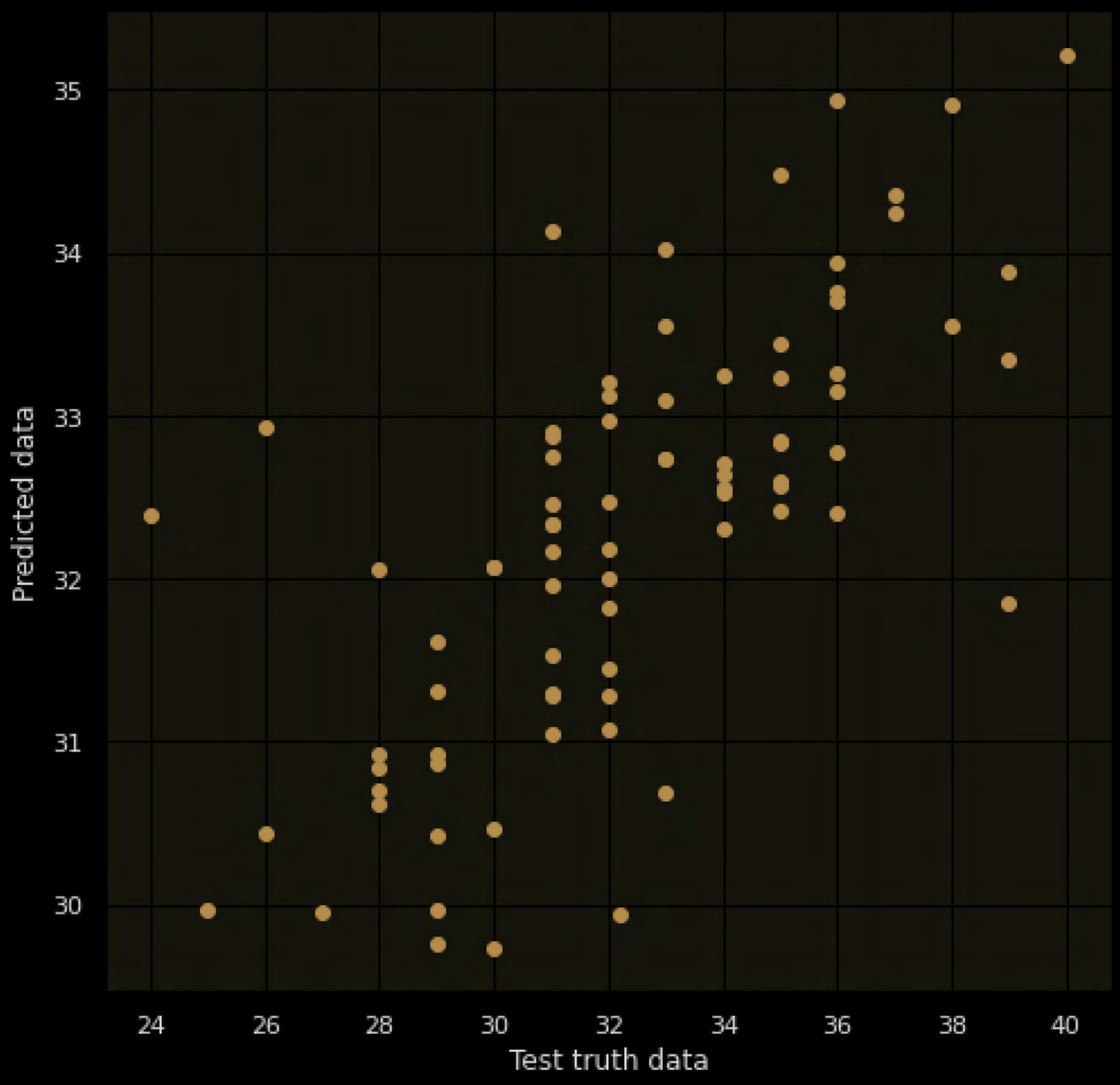
Out[ ]:
```
24    -1.162993
6      0.259013
152   -4.058366
232    1.442790
238   -2.071860
Name: Temperature, dtype: float64
```

Validation of Lasso Regression assumptions

Linear Relationship

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

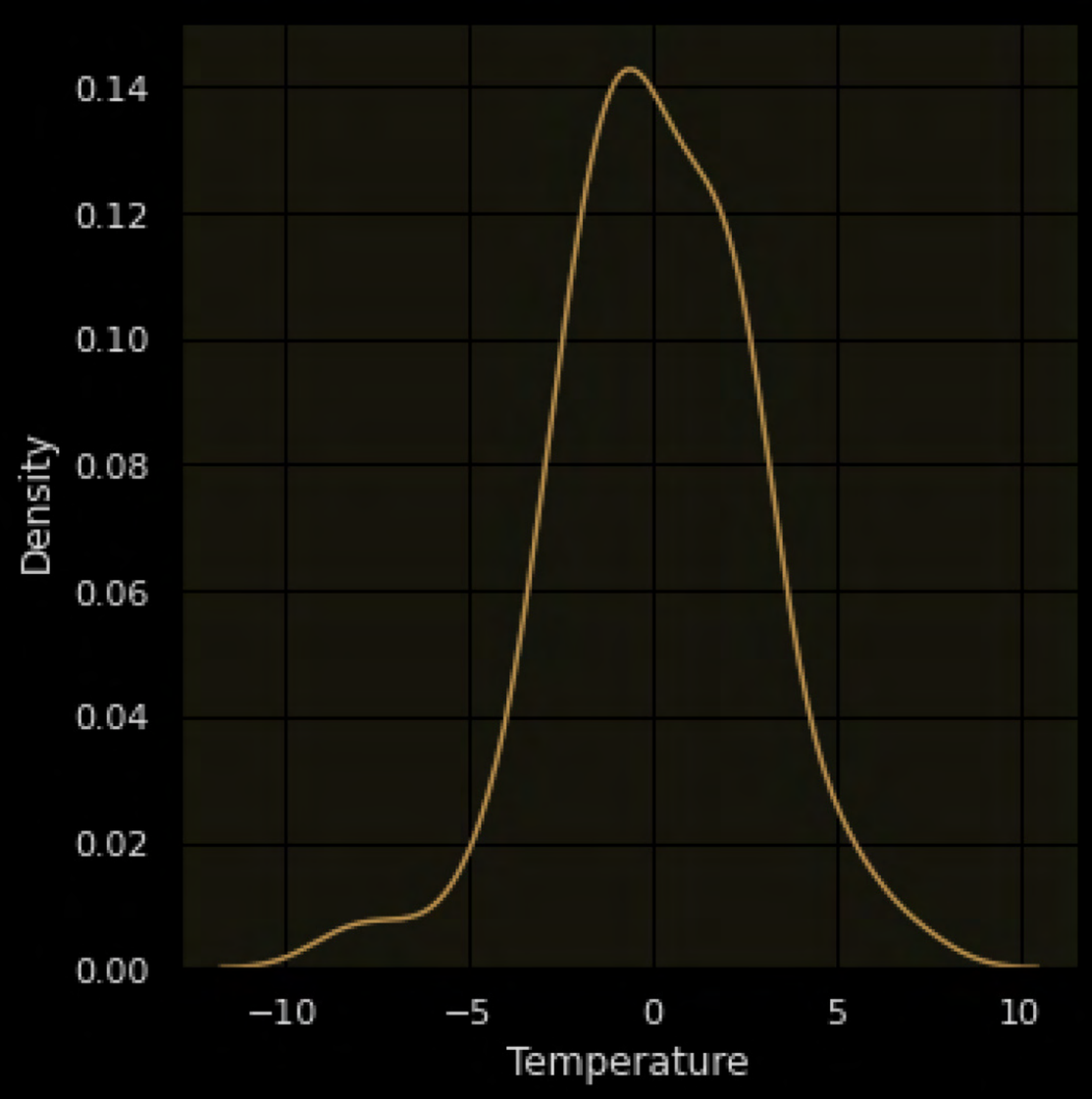
Out[ ]:
```
Text(0, 0.5, 'Predicted data')
```



Residual should be Normally Distributed

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

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



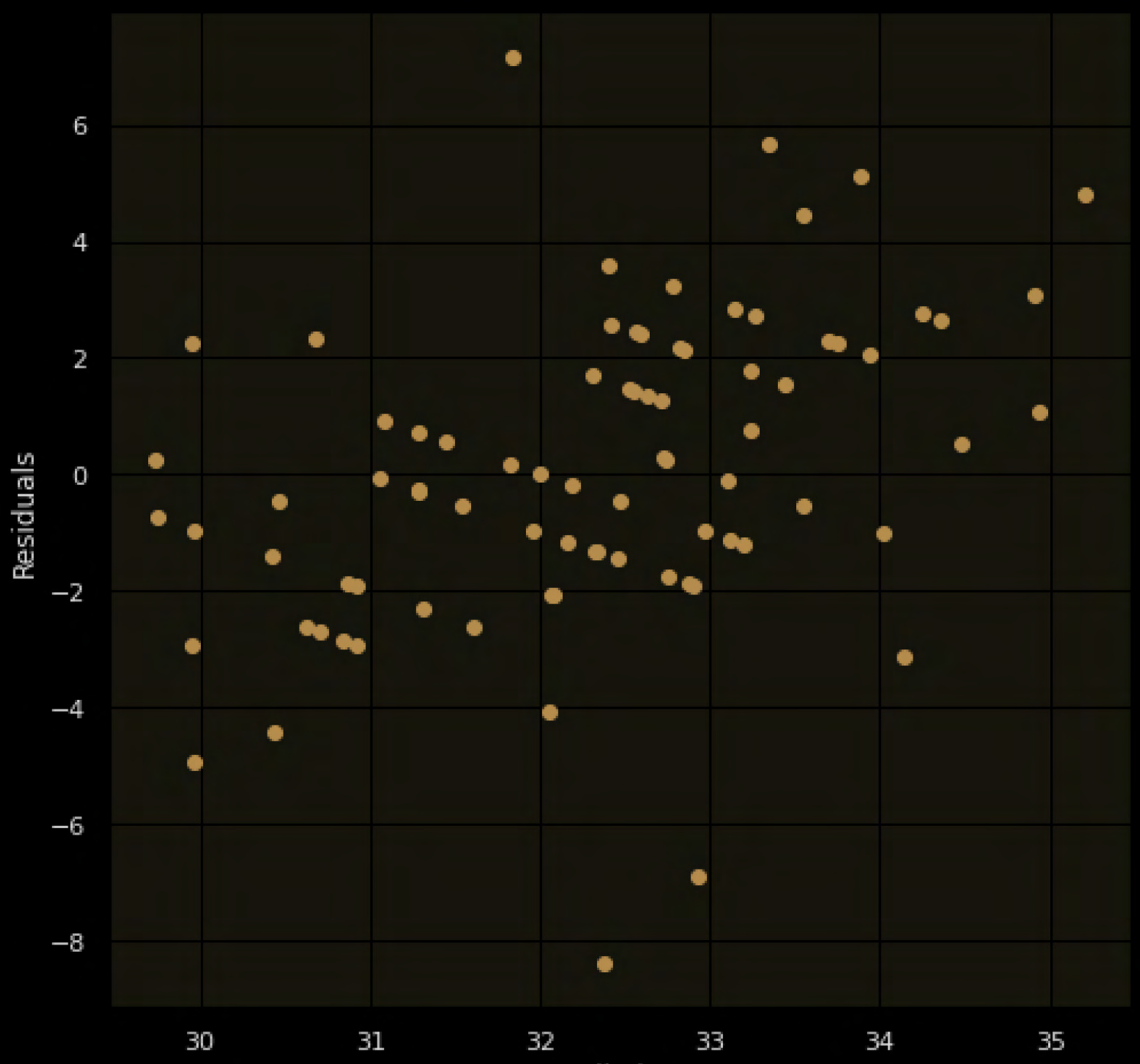
Residual and Predicted values should follow Uniform Distribution

---

Komal Diwe

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

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



Cost Function

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

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

Performance Metrics

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

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

Elastic Net Regression Model

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

```
elastic_reg
```

Out[ ]:
```
ElasticNet()
```

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

Out[ ]:
```
ElasticNet()
```

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

```
Co-efficients of independent features is [-0.79936853 -0.05286721 -0.
5025684  0.32720261  0.25459529
  0.          0.24934961  0.16518801  0.        ]
Intercept of best fit hyper plane is [32.1617284]
```

Prediction of Test data

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

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

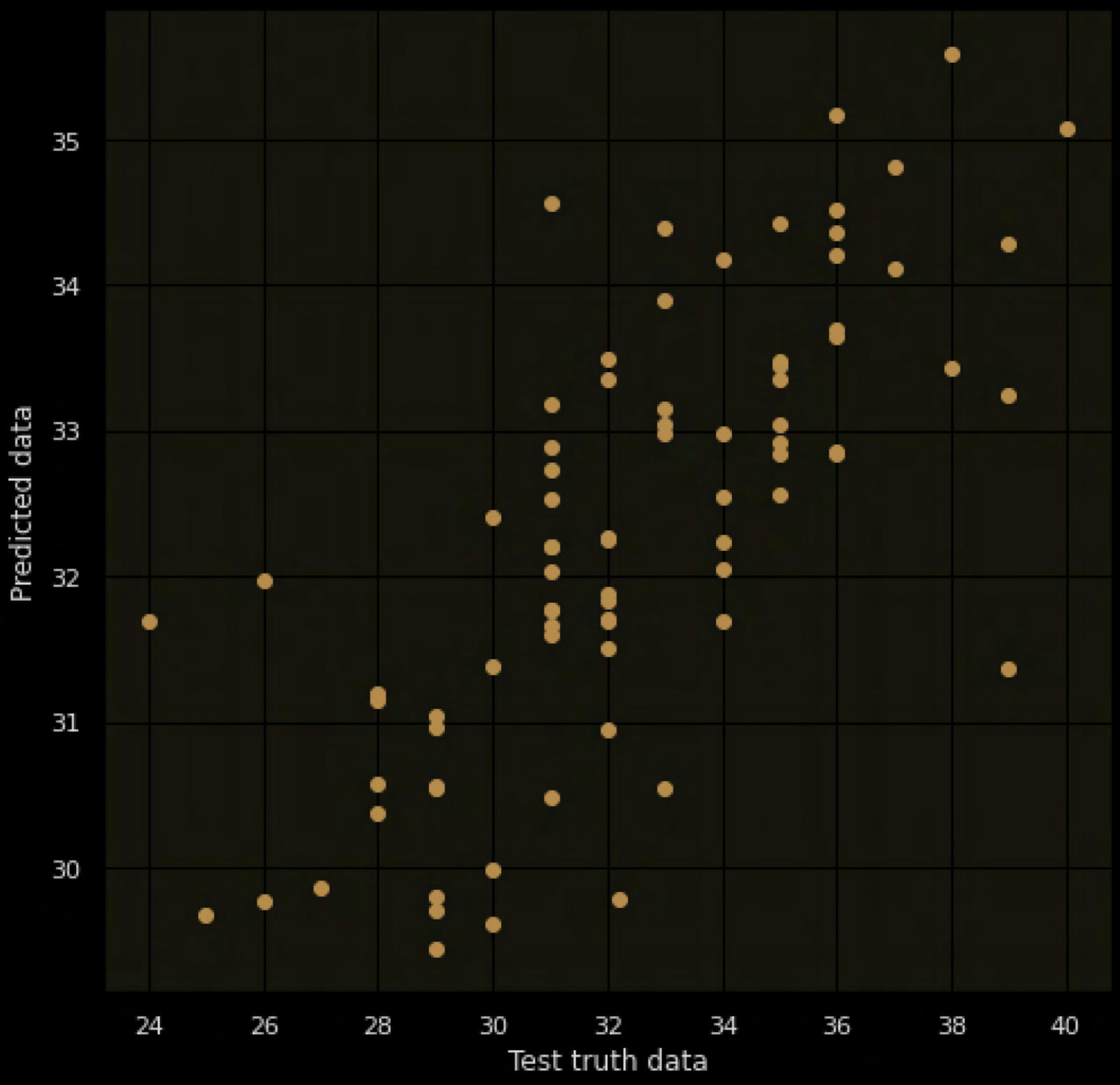
Out[ ]:
```
24    -1.534535
6      0.012176
152   -3.192632
232    1.759639
238   -2.406341
Name: Temperature, dtype: float64
```

Validation of Elastic Regression assumption

Linear Relationship

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

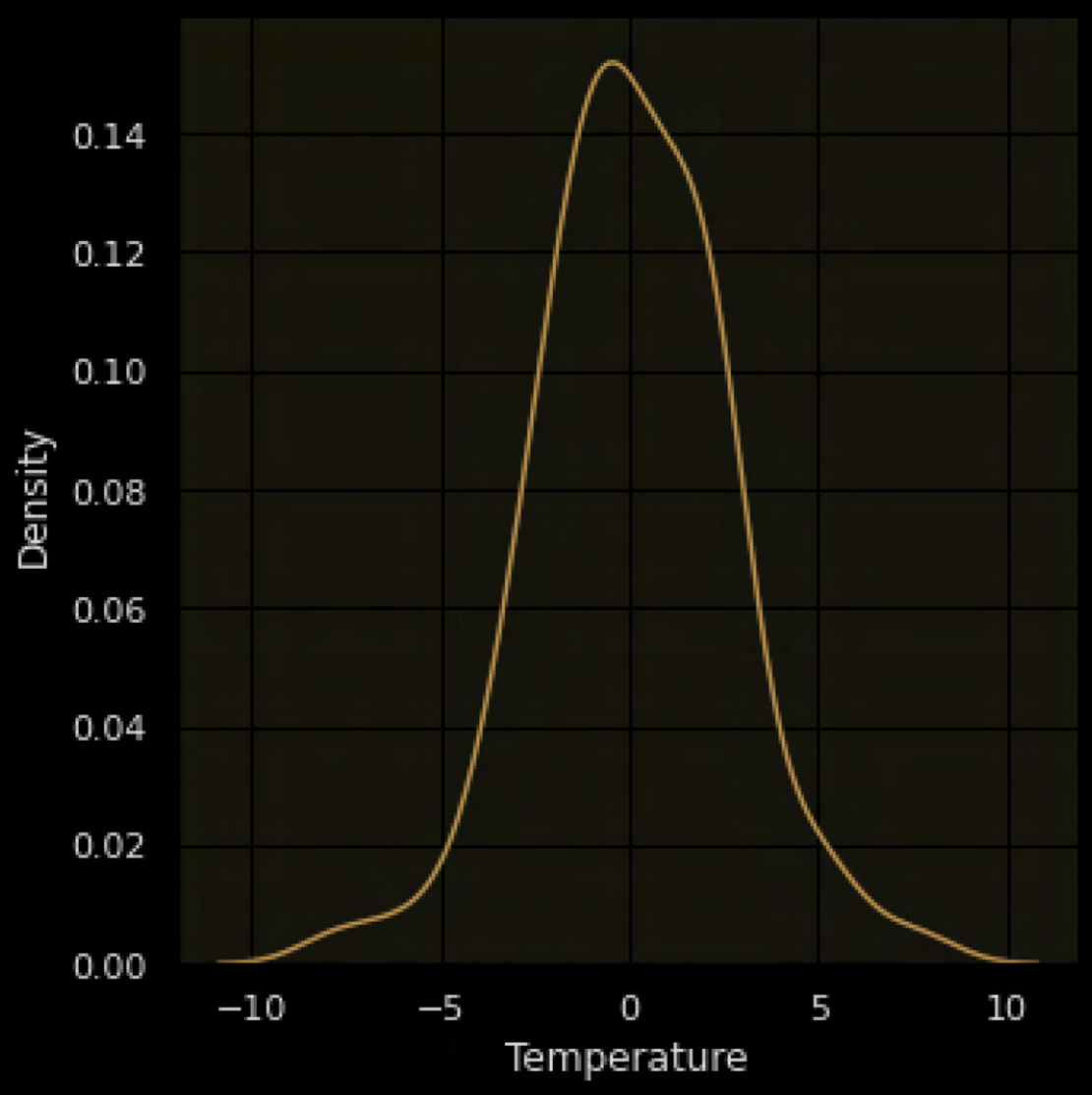
Out[ ]:
```
Text(0, 0.5, 'Predicted data')
```



Residual should be Normally Distributed

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

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



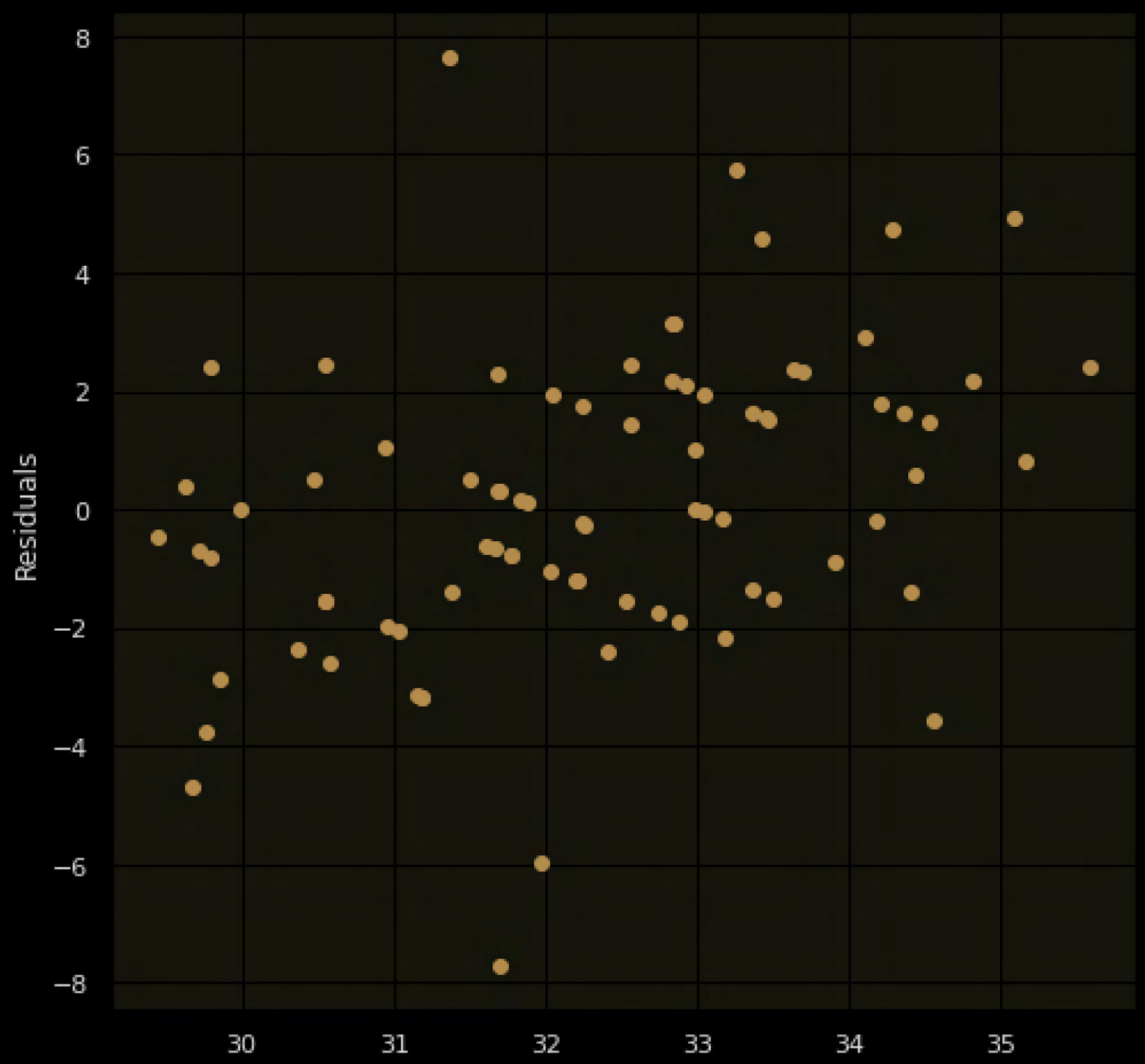
Residual and Predicted values should follow uniform distribution

---

Komal Diwe

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

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



Cost Function Values

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

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

Performance Metrics

In [ ]:
```
Elastic_score = r2_score(y_test,elastic_reg_pred)
print(f"R-Square Accuracy : {round(Elastic_score*100,2)}%")
print(f"Adjusted R-Square Accuracy : {round((1 - (1-Elastic_score)*(len(y_test)-1),
```

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

Comparision of all Models

Models:

Komal Diwe

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

Cost Function Values

In [ ]:
```
print("-----------------------------------------------------")
print(f"MSE:\n1. Linear Regression : {round(mean_squared_error(y_test,linear_reg_pr
print("-----------------------------------------------------")
print(f"MAE:\n1. Linear Regression : {round(mean_absolute_error(y_test,linear_reg_p
print("-----------------------------------------------------")
print(f"RMSE:\n1. Linear Regression : {round(np.sqrt(mean_squared_error(y_test,lin
print("-----------------------------------------------------")
```

```
-----------------------------------------------------
MSE:
1. Linear Regression : 5.01
2. Ridge Regression : 4.99
3. Lasso Regression : 7.06
4. ElasticNet Regression : 6.37
-----------------------------------------------------
MAE:
1. Linear Regression : 1.74
2. Ridge Regression : 1.74
3. Lasso Regression : 2.07
4. ElasticNet Regression : 1.95
-----------------------------------------------------
RMSE:
1. Linear Regression : 2.24
2. Ridge Regression : 2.23
3. Lasso Regression : 2.66
4. ElasticNet Regression : 2.52
-----------------------------------------------------
```

Performance Metrics

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

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

R-Square Accuracy:

----------------------------------------------------

1. Linear Regression : 56.52%

2. Ridge Regression : 56.67%

3. Lasso Regression : 38.7%

4. ElasticNet Regression : 44.71%

----------------------------------------------------

Adjusted R-Square Accuracy:

----------------------------------------------------

Linear Regression : 50.31%

Ridge Regression : 50.48%

Lasso Regression : 29.94%

ElasticNet Regression : 36.81%

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

Conclusion

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

Komal Diwe