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

```
from google.colab import files
uploaded = files.upload()
```

Choose Files Cleaned_Al...Dataset.csv

- **Cleaned_Algerian_Forest_Fires_Dataset.csv**(text/csv) - 15094 bytes, last modified: 27/6/2025 - 100% done

Saving Cleaned_Algerian_Forest_Fires_Dataset.csv to Cleaned_Algerian_Forest_Fires_Dataset (3).csv

```
df=pd.read_csv('Cleaned_Algerian_Forest_Fires_Dataset.csv')
```

```
df.head()
```

	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	not fire	0
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire	0
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	0
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	not fire	0
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.columns
```

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

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

```
df.head()
```

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
0	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	0
1	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire	0
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	0
3	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	not fire	0
4	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

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

Classes	count
fire	131
not fire	101
fire	4
fire	2
not fire	2
not fire	1
not fire	1
not fire	1

dtype: int64

```
df['Classes']=np.where(df['Classes'].str.contains("not fire"),0,1)
```

```
df.head()
```

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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.tail()
```

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
238	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	1	1
239	28	87	15	4.4	41.1	6.5	8.0	0.1	6.2	0.0	0	1
240	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	0	1
241	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	0	1
242	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	0	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243 entries, 0 to 242
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Temperature  243 non-null    int64  
 1   RH           243 non-null    int64  
 2   Ws           243 non-null    int64  
 3   Rain         243 non-null    float64
 4   FFMC         243 non-null    float64
 5   DMC          243 non-null    float64
 6   DC           243 non-null    float64
 7   ISI          243 non-null    float64
 8   BUI          243 non-null    float64
 9   FWI          243 non-null    float64
 10  Classes      243 non-null    int64  
 11  Region       243 non-null    int64  
dtypes: float64(7), int64(5)
memory usage: 22.9 KB
```

```
df.describe()
```

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Class
count	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.0000
mean	32.152263	62.041152	15.493827	0.762963	77.842387	14.680658	49.430864	4.742387	16.690535	7.035391	0.5637
std	3.628039	14.828160	2.811385	2.003207	14.349641	12.393040	47.665606	4.154234	14.228421	7.440568	0.4969
min	22.000000	21.000000	6.000000	0.000000	28.600000	0.700000	6.900000	0.000000	1.100000	0.000000	0.0000
25%	30.000000	52.500000	14.000000	0.000000	71.850000	5.800000	12.350000	1.400000	6.000000	0.700000	0.0000
50%	32.000000	63.000000	15.000000	0.000000	83.300000	11.300000	33.100000	3.500000	12.400000	4.200000	1.0000
75%	35.000000	73.500000	17.000000	0.500000	88.300000	20.800000	69.100000	7.250000	22.650000	11.450000	1.0000
max	42.000000	90.000000	29.000000	16.800000	96.000000	65.900000	220.400000	19.000000	68.000000	31.100000	1.0000

```
## Independent And dependent features
X=df.drop('FWI',axis=1)
y=df['FWI']


X.head()
```

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	Classes	Region
0	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0	0
1	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0	0
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0	0
3	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0	0
4	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0	0

Next steps: Generate code with X | View recommended plots | New interactive sheet

y

	FWI
0	0.5
1	0.4
2	0.1
3	0.0
4	0.5
...	...
238	6.5
239	0.0
240	0.2
241	0.7
242	0.5

243 rows × 1 columns

dtype: float64

```
#Train Test Split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=42)
```

```
X_train.shape,X_test.shape
```

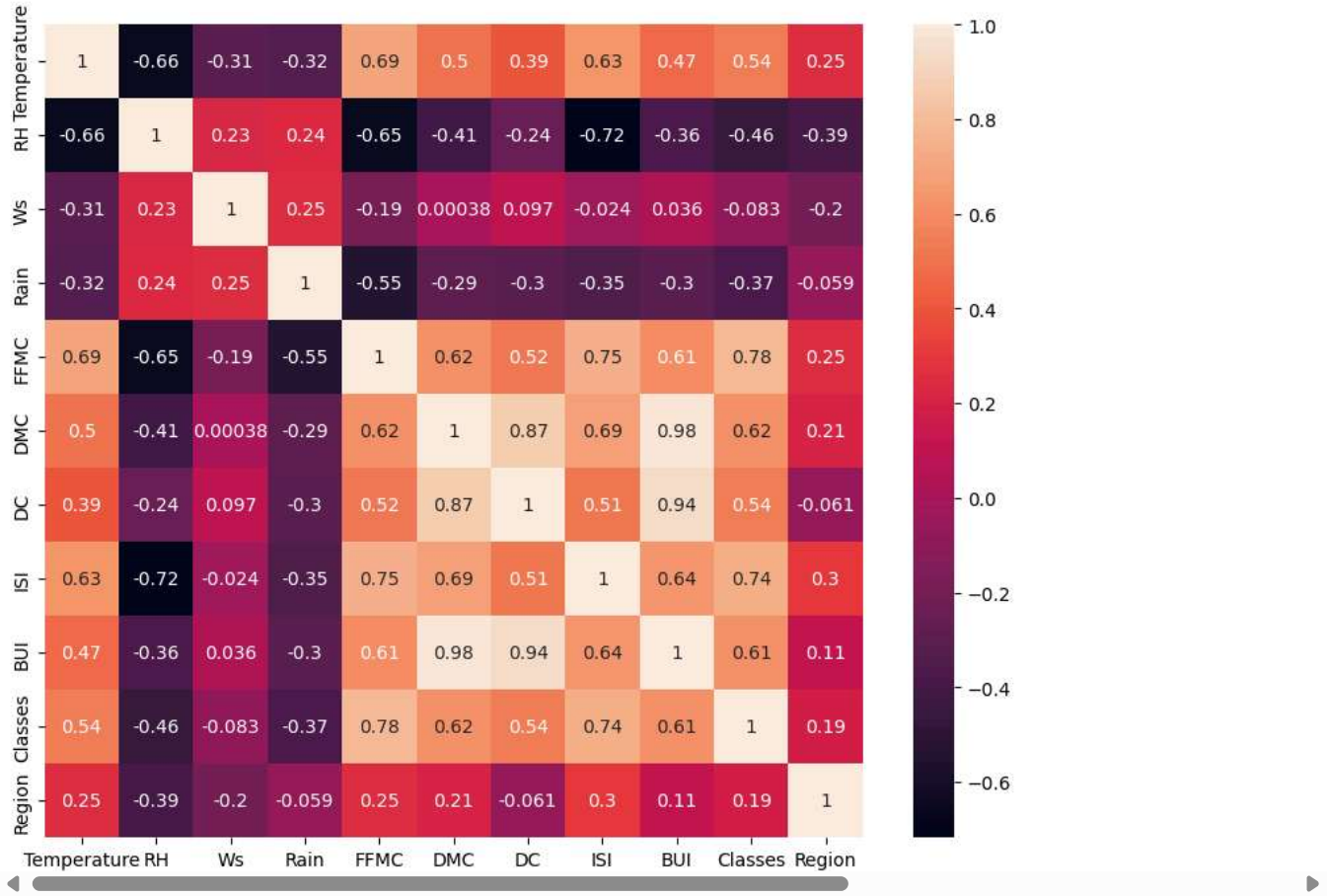
```
((182, 11), (61, 11))
```

```
## Feature Selection based on correlaltion
X_train.corr()
```

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	Classes	Region
Temperature	1.000000	-0.656095	-0.305977	-0.317512	0.694768	0.498173	0.390684	0.629848	0.473609	0.542141	0.254549
RH	-0.656095	1.000000	0.225736	0.241656	-0.653023	-0.414601	-0.236078	-0.717804	-0.362317	-0.456876	-0.394665
Ws	-0.305977	0.225736	1.000000	0.251932	-0.190076	0.000379	0.096576	-0.023558	0.035633	-0.082570	-0.199969
Rain	-0.317512	0.241656	0.251932	1.000000	-0.545491	-0.289754	-0.302341	-0.345707	-0.300964	-0.369357	-0.059022
FFMC	0.694768	-0.653023	-0.190076	-0.545491	1.000000	0.620807	0.524101	0.750799	0.607210	0.781259	0.249514
DMC	0.498173	-0.414601	0.000379	-0.289754	0.620807	1.000000	0.868647	0.685656	0.983175	0.617273	0.212582
DC	0.390684	-0.236078	0.096576	-0.302341	0.524101	0.868647	1.000000	0.513701	0.942414	0.543581	-0.060838
ISI	0.629848	-0.717804	-0.023558	-0.345707	0.750799	0.685656	0.513701	1.000000	0.643818	0.742977	0.296441
BUI	0.473609	-0.362317	0.035633	-0.300964	0.607210	0.983175	0.942414	0.643818	1.000000	0.612239	0.114897
Classes	0.542141	-0.456876	-0.082570	-0.369357	0.781259	0.617273	0.543581	0.742977	0.612239	1.000000	0.188837
Region	0.254549	-0.394665	-0.199969	-0.059022	0.249514	0.212582	-0.060838	0.296441	0.114897	0.188837	1.000000

```
## Checking for multicollinearity
plt.figure(figsize=(10,8))
corr=X_train.corr()
sns.heatmap(corr,annot=True)
```

<Axes: >




```
def correlation(dataset, threshold):
  col_corr=set()
  corr_matrix=dataset.corr()
  for i in range(len(corr_matrix.columns)):
    for j in range(i):
      if abs(corr_matrix.iloc[i,j])>threshold:
        colname=corr_matrix.columns[i]
        col_corr.add(colname)
  return col_corr


## Threshold Value (Determined by domain expert)
corr_features=correlation(X_train,0.85)


## Dropping those features with correlation more than 0.85
X_train.drop(corr_features,axis=1,inplace=True)
X_test.drop(corr_features,axis=1,inplace=True)
X_train.shape,X_test.shape
```

((182, 9), (61, 9))

```
X_train.columns,X_test.columns
```

```
(Index(['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'ISI', 'Classes',
        'Region'],
       dtype='object'),
 Index(['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'ISI', 'Classes',
        'Region'],
       dtype='object'))
```

Feature Scaling (Standardization)

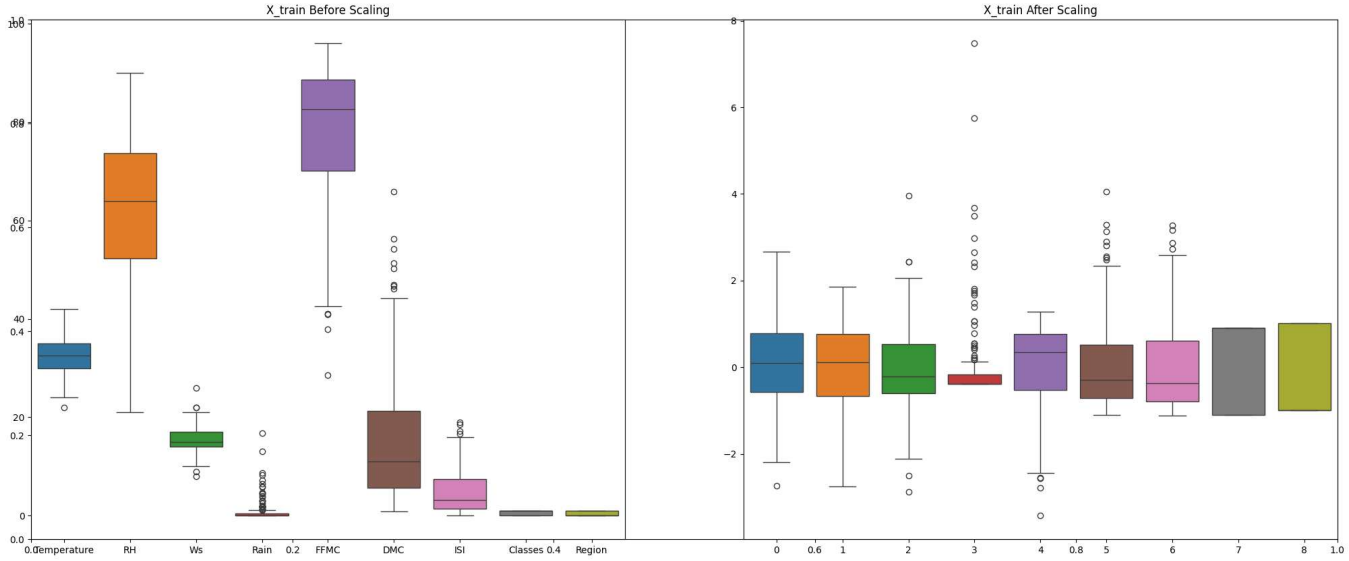
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)


X_train_scaled
```

```
array([[-0.84284248,  0.78307967,  1.29972026, ..., -0.62963326,
        -1.10431526, -0.98907071],
       [-0.30175842,  0.64950844, -0.59874754, ..., -0.93058524,
        -1.10431526,  1.01105006],
       [ 2.13311985, -2.08870172, -0.21905398, ...,  2.7271388 ,
         0.90553851,  1.01105006],
       ...,
       [-1.9250106 ,  0.9166509 ,  0.54033314, ..., -1.06948615,
        -1.10431526, -0.98907071],
       [ 0.50986767, -0.21870454,  0.16063958, ...,  0.5973248 ,
         0.90553851,  1.01105006],
       [-0.57230045,  0.98343651,  2.05910739, ..., -0.86113478,
        -1.10431526, -0.98907071]])
```

Box Plots To understand the Effect Of Standard Scaler

```
plt.subplots(figsize=(25, 10))
plt.subplot(1, 2, 1)
sns.boxplot(data=X_train)
plt.title('X_train Before Scaling')
plt.subplot(1, 2, 2)
sns.boxplot(data=X_train_scaled)
plt.title('X_train After Scaling')
```

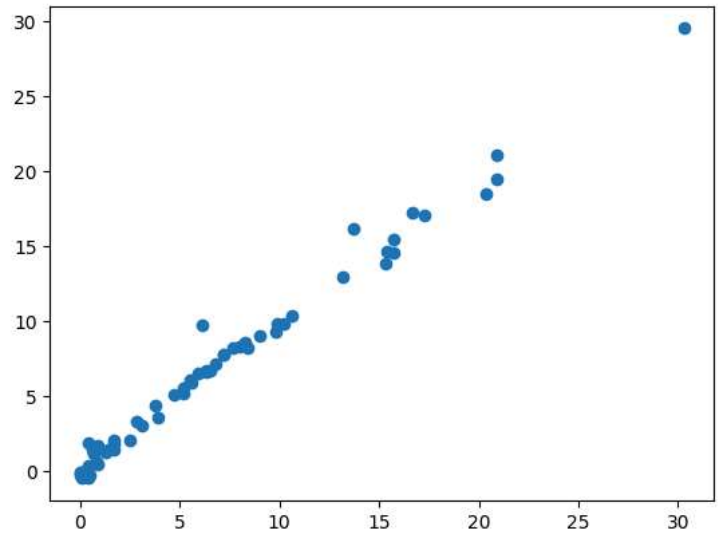
```
Text(0.5, 1.0, 'X_train After Scaling')
```



Linear Regression Model

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
lin_reg=LinearRegression()
lin_reg.fit(X_train_scaled,y_train)
y_pred=lin_reg.predict(X_test_scaled)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
print("Mean absolute error", mae)
print("R2 Score", score)
plt.scatter(y_test,y_pred)
```

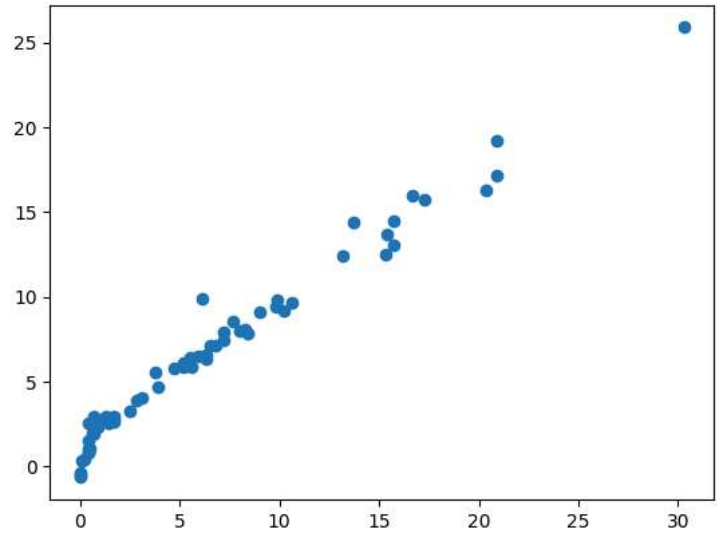
```
Mean absolute error 0.5468236465249986
R2 Score 0.9847657384266951
<matplotlib.collections.PathCollection at 0x7aadeaffb810>
```



Lasso Regression

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
lasso_reg=Lasso()
lasso_reg.fit(X_train_scaled,y_train)
y_pred=lasso_reg.predict(X_test_scaled)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
print("Mean absolute error", mae)
print("R2 Score", score)
plt.scatter(y_test,y_pred)
```

```
Mean absolute error 1.133175994914409
R2 Score 0.9492020263112388
<matplotlib.collections.PathCollection at 0x7aadeb7f0950>
```



Cross Validation Lasso

```
from sklearn.linear_model import LassoCV
lassocv=LassoCV(cv=5)
lassocv.fit(X_train_scaled,y_train)
```

```
LassoCV
LassoCV(cv=5)
```

```
lassocv.alpha_
```

```
np.float64(0.05725391318234408)
```

```
lassocv.alphas_
```

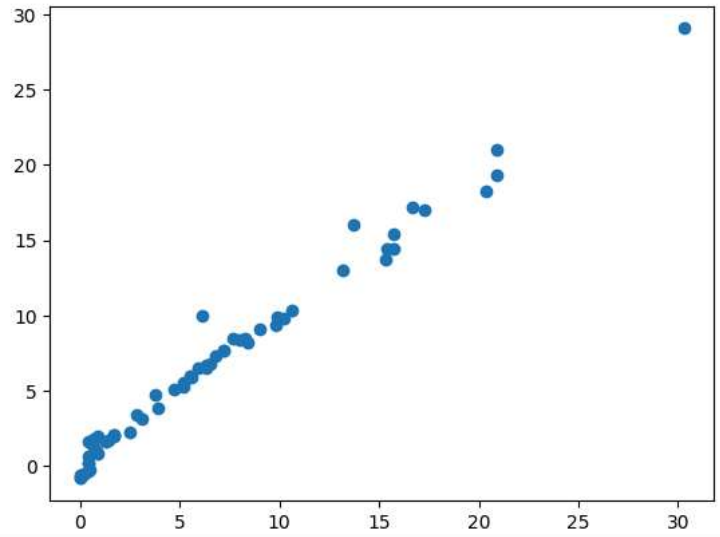
```
array([7.05853002, 6.58280872, 6.13914944, 5.72539132, 5.33951911,
       4.97965339, 4.64404142, 4.33104857, 4.03915039, 3.76692517,
       3.51304702, 3.27627941, 3.05546914, 2.84954075, 2.65749124,
       2.47838523, 2.31135036, 2.15557308, 2.01029467, 1.87480753,
       1.74845178, 1.63061198, 1.52071419, 1.41822315, 1.32263965,
       1.23349817, 1.15036452, 1.0728338 , 1.00052839, 0.93309613,
       0.87020857, 0.81155943, 0.75686304, 0.705853  , 0.65828087,
       0.61391494, 0.57253913, 0.53395191, 0.49796534, 0.46440414,
       0.43310486, 0.40391504, 0.37669252, 0.3513047 , 0.32762794,
       0.30554691, 0.28495408, 0.26574912, 0.24783852, 0.23113504,
       0.21555731, 0.20102947, 0.18748075, 0.17484518, 0.1630612 ,
       0.15207142, 0.14182231, 0.13226397, 0.12334982, 0.11503645,
       0.10728338, 0.10005284, 0.09330961, 0.08702086, 0.08115594,
       0.0756863 , 0.0705853 , 0.06582809, 0.06139149, 0.05725391,
       0.05339519, 0.04979653, 0.04644041, 0.04331049, 0.0403915 ,
       0.03766925, 0.03513047, 0.03276279, 0.03055469, 0.02849541,
       0.02657491, 0.02478385, 0.0231135 , 0.02155573, 0.02010295,
       0.01874808, 0.01748452, 0.01630612, 0.01520714, 0.01418223,
       0.0132264 , 0.01233498, 0.01150365, 0.01072834, 0.01000528,
       0.00933096, 0.00870209, 0.00811559, 0.00756863, 0.00705853])
```

```
lassocv.mse_path_
```

```
[ 1.29190414,  2.35586563,  6.24396657,  2.88962691,  0.94446421],
[ 1.26432775,  2.30322922,  6.29839177,  2.84193587,  0.94587808],
[ 1.23978001,  2.25555322,  6.35084178,  2.79951289,  0.94833447],
[ 1.21631899,  2.21233117,  6.39529714,  2.76173546,  0.9516149 ],
[ 1.18452807,  2.16000042,  6.42643862,  2.72805794,  0.95553494],
[ 1.15701368,  2.11158527,  6.45636883,  2.69761451,  0.95595979],
[ 1.13320995,  2.06862134,  6.48638494,  2.67077799,  0.93982106],
[ 1.11260173,  2.03047905,  6.51570208,  2.6467804 ,  0.9267394 ],
[ 1.09481028,  1.9965441 ,  6.54418839,  2.62528895,  0.92156967],
[ 1.07943936,  1.96633625,  6.57175064,  2.60556754,  0.91819477],
[ 1.06616655,  1.93941688,  6.60120289,  2.58826543,  0.91600498],
[ 1.05471212,  1.91540122,  6.66074506,  2.53939631,  0.91492536],
[ 1.04483316,  1.89395167,  6.72040081,  2.49354558,  0.91475751],
[ 1.03631885,  1.87477186,  6.77985049,  2.45183158,  0.91533073],
[ 1.02898619,  1.85760147,  6.8386118 ,  2.41402473,  0.91650002],
[ 1.02267637,  1.84221172,  6.89546904,  2.37952566,  0.91817465],
[ 1.0172516 ,  1.81986019,  6.95182997,  2.34943959,  0.92100746],
[ 1.01259234,  1.7874912 ,  7.00657253,  2.30905785,  0.91090128],
[ 0.99291676,  1.75813753,  7.05952508,  2.26689771,  0.88812743],
[ 0.96711245,  1.73133215,  7.11055395,  2.22965179,  0.86893338],
[ 0.94404465,  1.70754321,  7.15957739,  2.19646   ,  0.85251259],
[ 0.91746069,  1.68586828,  7.21115863,  2.16644165,  0.83841802],
[ 0.89121876,  1.66666838,  7.26823916,  2.14003416,  0.82646203],
[ 0.86783937,  1.64937312,  7.32193772,  2.11642121,  0.81629395],
[ 0.84703112,  1.6337788 ,  7.37194387,  2.09528441,  0.80766048],
[ 0.82845196,  1.619701  ,  7.42070575,  2.07634166,  0.80034774],
[ 0.81184328,  1.6069769 ,  7.46783924,  2.05934486,  0.79417047],
[ 0.79697877,  1.59523036,  7.51171241,  2.04379341,  0.78898574],
[ 0.78366252,  1.58481658,  7.5533042 ,  2.03007893,  0.78514158],
[ 0.77340653,  1.57536934,  7.59178479,  2.01773193,  0.78410497],
[ 0.76437368,  1.56730639,  7.62890427,  2.00633629,  0.78327866],
[ 0.75641103,  1.56014926,  7.66385201,  1.99569195,  0.78309295],
[ 0.74929762,  1.55377904,  7.69675973,  1.98581272,  0.78325254],
[ 0.7431075 ,  1.54808751,  7.72772336,  1.97708583,  0.78348718],
[ 0.73764056,  1.5428574 ,  7.75701245,  1.9690422 ,  0.78415382],
[ 0.73271889,  1.5383076 ,  7.78098988,  1.96195515,  0.78479522],
[ 0.72844826,  1.53422868,  7.80009362,  1.95555728,  0.78577592],
[ 0.72457927,  1.53042136,  7.81782859,  1.94960372,  0.78686385],
[ 0.72121402,  1.5271394 ,  7.83584096,  1.94420011,  0.78783843],
[ 0.71854269,  1.52403047,  7.8521645 ,  1.93945512,  0.78886011],
[ 0.71624922,  1.52137747,  7.86797141,  1.93532188,  0.79008917],
[ 0.71419505,  1.51882628,  7.8824946 ,  1.93156393,  0.7910736 ],
[ 0.71283686,  1.51649634,  7.89597341,  1.92813104,  0.79328236],
[ 0.7117556 ,  1.51454548,  7.90862683,  1.92492966,  0.7959553 ],
[ 0.71078691,  1.5128162 ,  7.92077339,  1.92207644,  0.79869912],
[ 0.71003406,  1.51137977,  7.93211766,  1.9195065 ,  0.80158876],
[ 0.7094272 ,  1.51017923,  7.94254787,  1.9171673 ,  0.80451499],
[ 0.70893209,  1.50910355,  7.95231005,  1.91555613,  0.80717091],
[ 0.70847636,  1.50819995,  7.96151575,  1.914521  ,  0.8098638 ],
[ 0.70814046,  1.50740984,  7.97034636,  1.91358558,  0.81227152],
[ 0.70789298,  1.5065737 ,  7.97838619,  1.91277526,  0.81468439],
[ 0.70770357,  1.50591279,  7.98587605,  1.9120262 ,  0.8170304 ],
[ 0.70752166,  1.50536216,  7.99241057,  1.91138883,  0.81925406],
[ 0.70734296,  1.50487616,  7.99849196,  1.91084915,  0.82119901],
[ 0.70724307,  1.50444309,  8.00451482,  1.91033293,  0.82327046],
[ 0.70719344,  1.50391791,  8.01011355,  1.9098903 ,  0.8250587 ],
[ 0.70714379,  1.50342997,  8.01481494,  1.90951275,  0.826765  ],
[ 0.70711086,  1.50300182,  8.01992921,  1.90919915,  0.82842365]])
```

```
y_pred=lassocv.predict(X_test_scaled)
plt.scatter(y_test,y_pred)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
print("Mean absolute error", mae)
print("R2 Score", score)
```

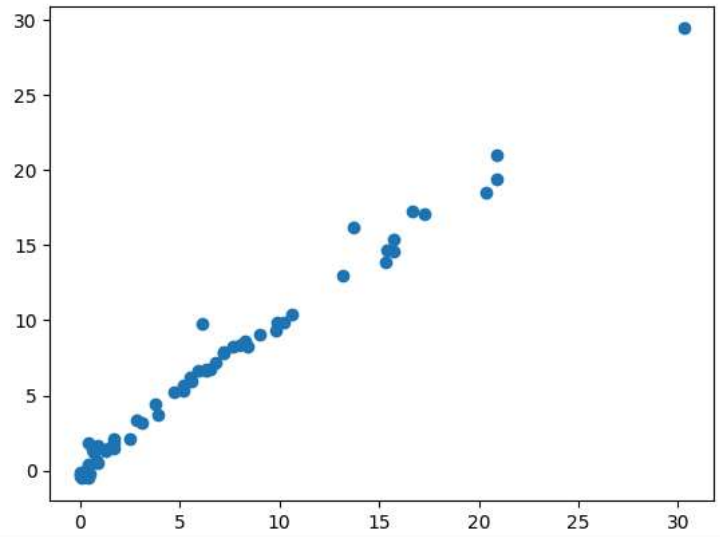
```
Mean absolute error 0.619970115826343
R2 Score 0.9820946715928275
```



Ridge Regression

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
ridge=Ridge()
ridge.fit(X_train_scaled,y_train)
y_pred=ridge.predict(X_test_scaled)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
print("Mean absolute error", mae)
print("R2 Score", score)
plt.scatter(y_test,y_pred)
```

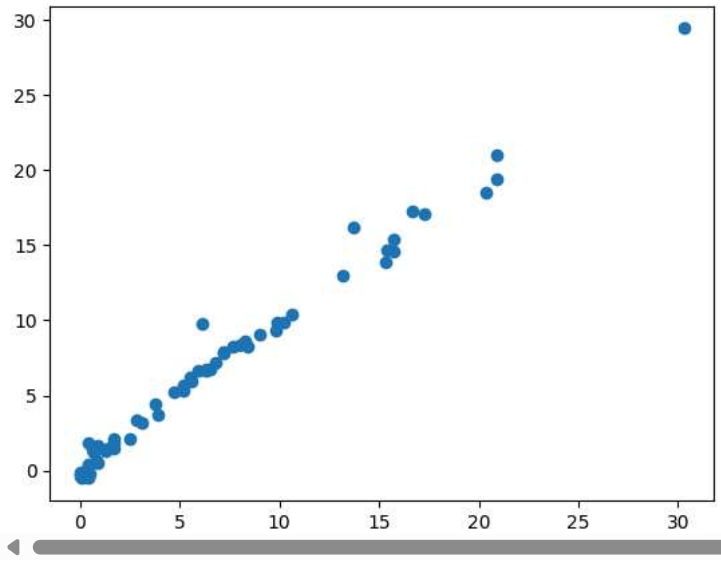
```
Mean absolute error 0.5642305340105692
R2 Score 0.9842993364555513
<matplotlib.collections.PathCollection at 0x7aadeab555d0>
```




```
from sklearn.linear_model import RidgeCV
ridgecv=RidgeCV(cv=5)
ridgecv.fit(X_train_scaled,y_train)
y_pred=ridgecv.predict(X_test_scaled)
plt.scatter(y_test,y_pred)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
```

```
print("Mean absolute error", mae)
print("R2 Score", score)
```

```
Mean absolute error 0.5642305340105692
R2 Score 0.9842993364555513
```




```
print(ridgecv.alpha_)
```

```
1.0
```

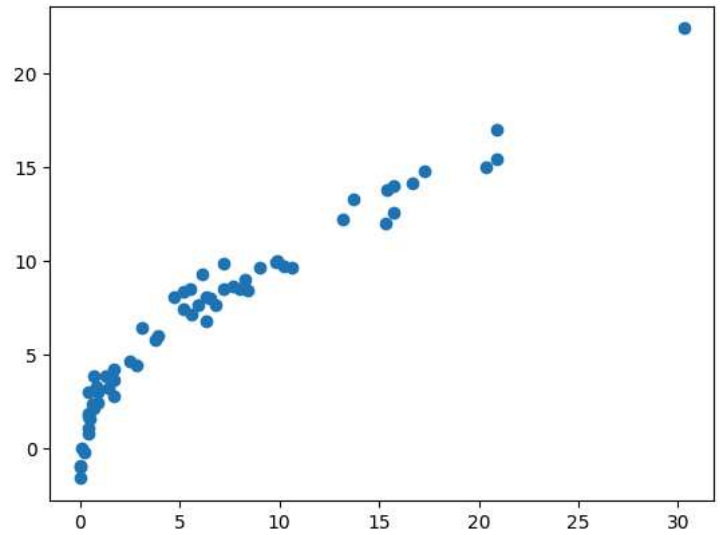
```
ridgecv.get_params()
```

```
{'alpha_per_target': False,
 'alphas': (0.1, 1.0, 10.0),
 'cv': 5,
 'fit_intercept': True,
 'gcv_mode': None,
 'scoring': None,
 'store_cv_results': None,
 'store_cv_values': 'deprecated'}
```

ElasticNet Regression

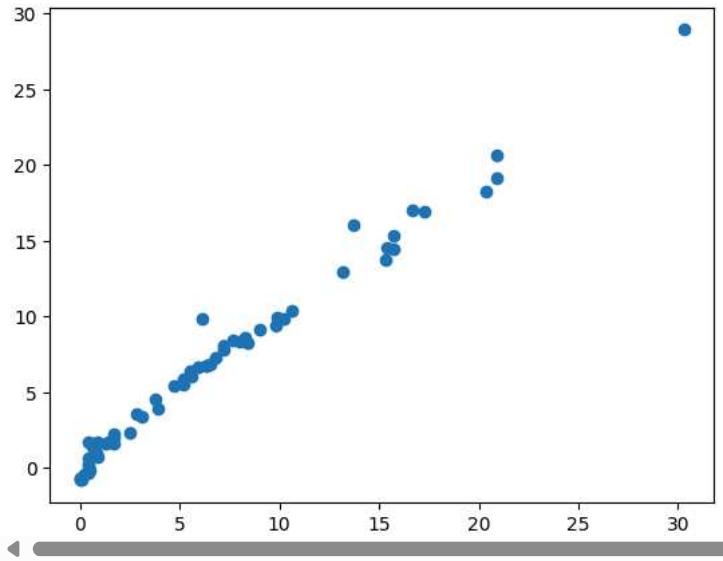
```
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
elastic=ElasticNet()
elastic.fit(X_train_scaled,y_train)
y_pred=elastic.predict(X_test_scaled)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
print("Mean absolute error", mae)
print("R2 Score", score)
plt.scatter(y_test,y_pred)
```

```
Mean absolute error 1.8822353634896005
R2 Score 0.8753460589519703
<matplotlib.collections.PathCollection at 0x7aadeacb1d50>
```




```
from sklearn.linear_model import ElasticNetCV
elasticcv=ElasticNetCV(cv=5)
elasticcv.fit(X_train_scaled,y_train)
y_pred=elasticcv.predict(X_test_scaled)
plt.scatter(y_test,y_pred)
mae=mean_absolute_error(y_test,y_pred)
score=r2_score(y_test,y_pred)
print("Mean absolute error", mae)
print("R2 Score", score)
```

```
Mean absolute error 0.6575946731430898
R2 Score 0.9814217587854941
```




```
elasticcv.alphas_
```

```
array([14.11706004, 13.16561744, 12.27829889, 11.45078264, 10.67903821,
        9.95930678,  9.28808283,  8.66209714,  8.07830078,  7.53385034,
        7.02609405,  6.55255882,  6.11093829,  5.6990815 ,  5.31498248,
        4.95677045,  4.62270071,  4.31114616,  4.02058933,  3.74961507,
        3.49690356,  3.26122397,  3.04142839,  2.83644629,  2.64527931,
        2.46699633,  2.30072904,  2.1456676 ,  2.00105679,  1.86619226,
        1.74041714,  1.62311885,  1.51372607,  1.411706  ,  1.31656174,
        1.22782989,  1.14507826,  1.06790382,  0.99593068,  0.92880828,
        0.86620971,  0.80783008,  0.75338503,  0.7026094 ,  0.65525588,
        0.61109383,  0.56990815,  0.53149825,  0.49567705,  0.46227007,
        0.43111462,  0.40205893,  0.37496151,  0.34969036,  0.3261224 ,
        0.30414284,  0.28364463,  0.26452793,  0.24669963,  0.2300729 ,
        0.21456676,  0.20010568,  0.18661923,  0.17404171,  0.16231189,
        0.15137261,  0.1411706 ,  0.13165617,  0.12278299,  0.11450783,
        0.10679038,  0.09959307,  0.09288083,  0.08662097,  0.08078301,
        0.0753385 ,  0.07026094,  0.06552559,  0.06110938,  0.05699082,
        0.05314982,  0.0495677 ,  0.04622701,  0.04311146,  0.04020589,
        0.03749615,  0.03496904,  0.03261224,  0.03041428,  0.02836446,
        0.02645279,  0.02466996,  0.02300729,  0.02145668,  0.02001057,
        0.01866192,  0.01740417,  0.01623119,  0.01513726,  0.01411706])
```

Pickling the machine learning model(Ridge Regressor), Preprocessing Model StandardScaler

```
scaler
```

```
StandardScaler
StandardScaler()
```