CSE1015 - Machine Learning Essentials

Pinni Venkata Abhiram

20BAI1132

Lab - 7

Prediction of Strain value using linear and polynomial regression

Importing the required modules

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.linear_model import LinearRegression as linearreg
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from tabulate import tabulate
from sklearn import metrics
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
```

Importing of train data and analysing it

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

Ignorning the warnings

```
warnings.filterwarnings('ignore')
```

Preprocessing the data for train set

```
df.columns
```

```
Index(['Load', 'Actuator', 'Time', 'Strain'], dtype='object')
```

```
df.head(10)
```

	Load	Actuator	Time	Strain
0	-1.90042	0.000000	0.101	0.000006
1	-1.90527	0.006944	0.168	0.000002
2	-1.85161	0.006944	0.234	0.000001
3	-1.58909	0.006944	0.301	0.000008
4	-1.36584	0.013889	0.368	0.000009
5	-1.13887	0.020833	0.434	0.000012
6	-0.91174	0.034722	0.501	0.000016
7	-0.65850	0.048611	0.568	0.000013
8	-0.40833	0.062500	0.634	0.000016
9	-0.07415	0.076389	0.701	0.000024

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7368 entries, 0 to 7367
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Load      7368 non-null   float64
 1   Actuator  7368 non-null   float64
 2   Time      7368 non-null   float64
 3   Strain    7368 non-null   float64
dtypes: float64(4)
memory usage: 230.4 KB
```

```
df.describe()
```

	Load	Actuator	Time	Strain
count	7368.000000	7368.000000	7368.000000	7368.000000
mean	129.573679	19.676954	122.940885	0.003217
std	30.083359	12.201710	71.024682	0.012886
min	-2.500420	0.000000	0.101000	-0.035840
25%	105.477175	7.770834	61.486500	0.000379
50%	139.309250	17.447915	122.869000	0.000460
75%	155.505900	30.342015	184.266500	0.003682
max	159.368800	44.465280	251.531000	0.211319

From the above describe function we can get the values of standard deviation , percentile value and many more

Normalisation of data using the max absolute scaling method

```
def maximum_absolute_scaling(dataFrame):
    dataFrame_scaled = dataFrame.copy()
    for column in dataFrame_scaled.columns:
        dataFrame_scaled[column] = dataFrame_scaled[column]  / dataFrame_scaled[column].ab
    return dataFrame_scaled

# call the maximum_absolute_scaling function
df_scaled = maximum_absolute_scaling(df)

df_scaled
```

	Load	Actuator	Time	Strain
0	-0.011925	0.000000	0.000402	0.000028
1	-0.011955	0.000156	0.000668	0.000010
2	-0.011618	0.000156	0.000930	0.000007
3	-0.009971	0.000156	0.001197	0.000036
4	-0.008570	0.000312	0.001463	0.000041
...	...	...	...	...
7363	0.639864	0.998594	0.998939	0.001788
7364	0.638886	0.998907	0.999205	0.001789
7365	0.637644	0.999219	0.999467	0.001805
7366	0.635310	0.999688	0.999734	0.001795
7367	0.631071	1.000000	1.000000	0.001789

7368 rows × 4 columns

```
df_scaled.columns
```

```
Index(['Load', 'Actuator', 'Time', 'Strain'], dtype='object')
```

Checking for null rows , i.e. NaN rows and dropping them if there are any

```
df_scaled.isnull().sum()
```

```
Load        0
Actuator    0
Time        0
Strain      0
dtype: int64
```

Finding the correlation for the train dataset

```
correlation = df_scaled.corr()
correlation['Strain']
```

```
Load       -0.112439
Actuator   -0.195955
Time       -0.162852
Strain      1.000000
Name: Strain, dtype: float64
```

The target var for the question is strain so we find the correlation between the strain and the other variables

In the aboue result there is a negetive correlation but it's significant , the negetive correlation means that means the values of the input variable and the output variable change in the opposite directions.

Therefore we have significant correlations so we don't drop any columns , all columns are important

Plots for the train dataset (Exploratory analysis)

Heatmap for correlation

In this heatmap we can see that the correlation color is quite low and it says that there are parts which have low correlation value and high correlation value regions too so we can't remove any column

```
sns.heatmap(df_scaled.corr() , cmap="Blues")
```

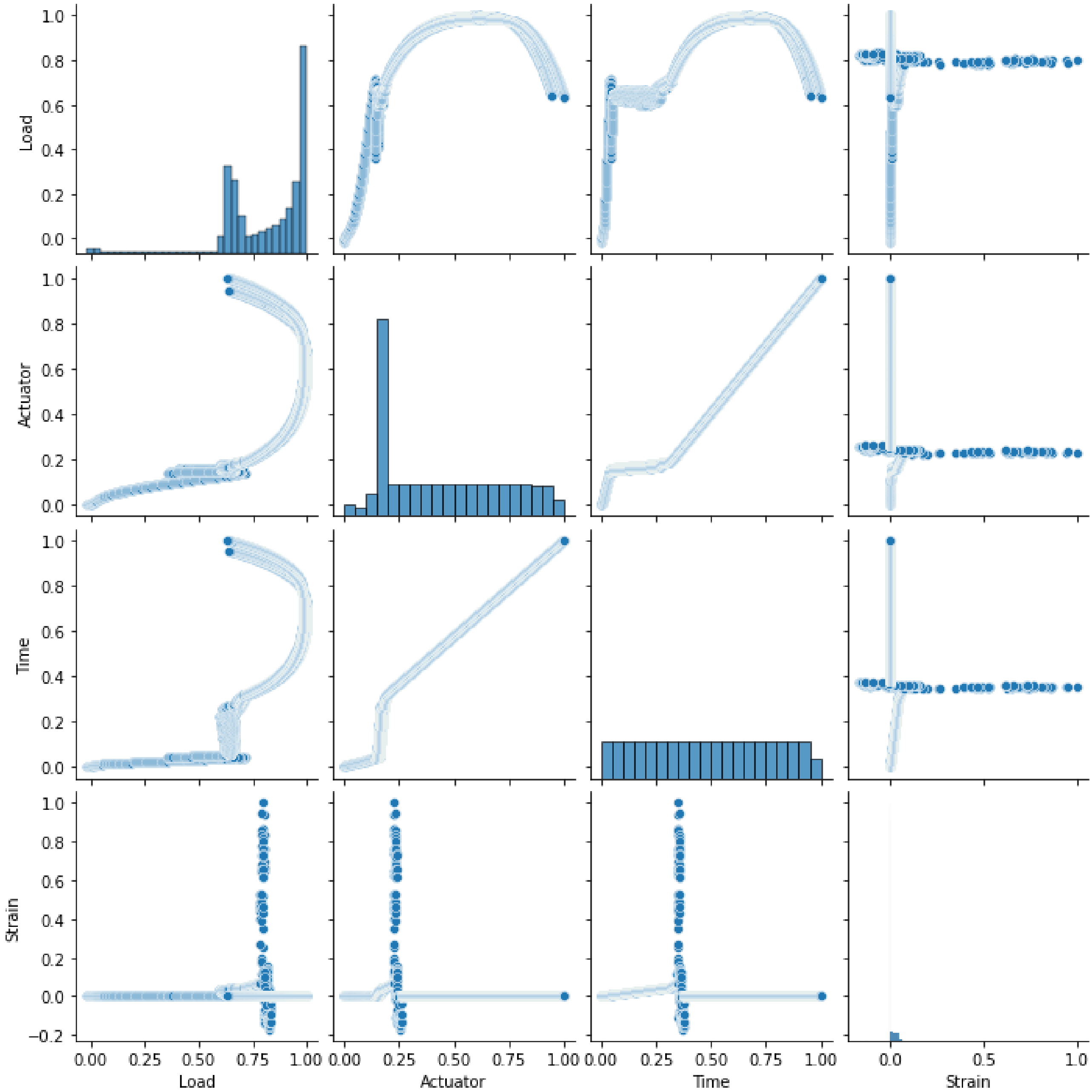
<AxesSubplot:>

Pairplot

Pairplot is usually a grid of plots for each variable in your dataset. Hence you can quickly see how all the variables are related. This can help to infer which variables are useful, which have skewed distribution etc.

```
sns.pairplot(df_scaled)
```

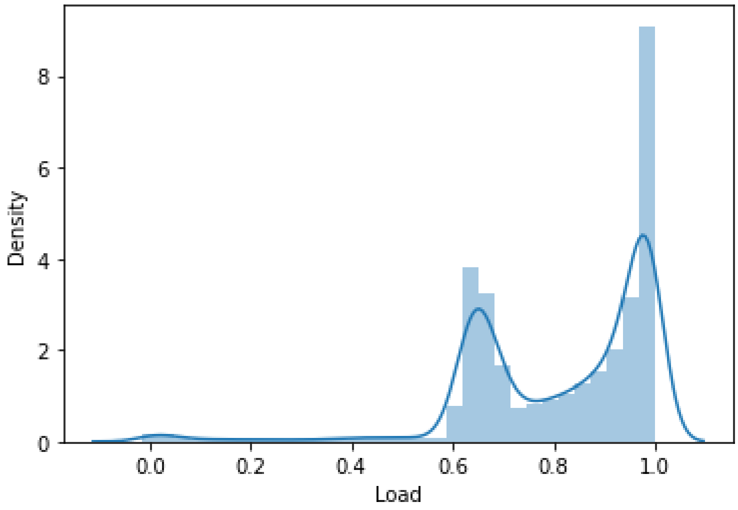
<seaborn.axisgrid.PairGrid at 0x20e41ff8e80>



Distplot

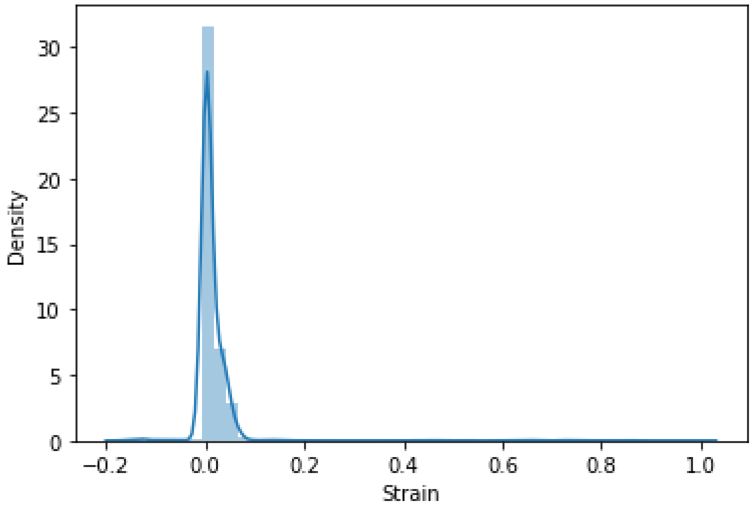
```
sns.distplot(df_scaled['Load'])
```

```
<AxesSubplot:xlabel='Load', ylabel='Density'>
```




```
sns.distplot(df_scaled['Strain'])
```

```
<AxesSubplot:xlabel='Strain', ylabel='Density'>
```



Preprocessing for the test dataset

```
df2 = pd.read_csv('testStrain.csv')
```

```
df2.head(10)
```

	Load	Actuator	Time	Strain
0	-3.06825	0.000000	0.090	0.000001
1	-2.84884	0.000000	0.157	0.000002
2	-2.38585	0.000000	0.223	0.000008
3	-1.91418	0.000000	0.290	0.000012

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3544 entries, 0 to 3543
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Load      3544 non-null   float64
 1   Actuator  3544 non-null   float64
 2   Time      3544 non-null   float64
 3   Strain    3544 non-null   float64
dtypes: float64(4)
memory usage: 110.9 KB
```

```
df2.describe()
```

	Load	Actuator	Time	Strain
count	3544.000000	3544.000000	3544.000000	3544.000000
mean	128.847911	19.503096	118.230345	-0.184297
std	29.834837	11.508828	68.237601	0.197000
min	-3.068250	0.000000	0.090000	-0.386398
25%	108.050800	8.527778	59.140250	-0.386396
50%	136.938750	17.166665	118.251000	-0.013446
75%	154.519600	29.565973	177.300750	0.003765
max	157.671400	41.972220	236.351000	0.240702

Normalisation of the data using maximum absolute scaling

```
df2_scaled = maximum_absolute_scaling(df2)
df2_scaled
```

	Load	Actuator	Time	Strain
0	-0.019460	0.000000	0.000381	0.000003
1	-0.018068	0.000000	0.000664	0.000004
2	-0.015132	0.000000	0.000944	0.000021
3	-0.012140	0.000000	0.001227	0.000031
4	-0.010222	0.000165	0.001510	0.000031
...	...	...	...	...
3539	0.647656	0.998677	0.998870	-0.999995
3540	0.646519	0.999007	0.999154	-0.999995

Data Cleaning for the test dataset

```
df2_scaled.isnull().sum()
```

```
Load        0
Actuator    0
Time        0
Strain      0
dtype: int64
```

There are no null rows so no need to drop the rows

There are no unnessecary columns in our dataset when we analysed for the train dataset so we don't have to drop any columns and directly go for getting the correlation and correlation heatmap

```
corr = df2_scaled.corr()
corr['Strain']
```

```
Load       -0.615589
Actuator   -0.858181
Time       -0.845092
Strain      1.000000
Name: Strain, dtype: float64
```

The target var for the question is strain so we find the correlation between the strain and the other variables

In the aboue result there is a negetive correlation but it's significant , the negetive correlation means that means the values of the input variable and the output variable change in the opposite directions.

Therefore we have significant correlations so we don't drop any columns , all columns are important

```
df2_scaled.columns
```

```
Index(['Load', 'Actuator', 'Time', 'Strain'], dtype='object')
```

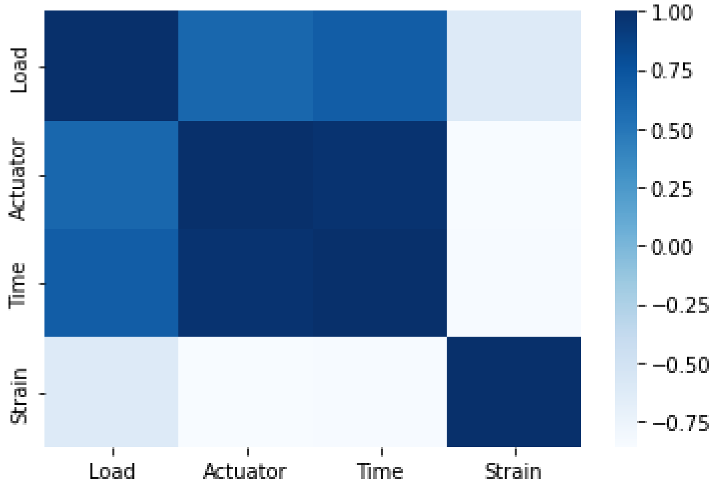
Explanatory analysis i.e. plots for the test dataset

Heatmap for correlation

In this heatmap we can see that the correlation color is quite low but it's not pure white (i.e. 0 correlation) and it says that there are parts which have low correlation value and high correlation value regions too so we can't remove any column

```
sns.heatmap(df2_scaled.corr() , cmap="Blues")
```

```
<AxesSubplot:>
```

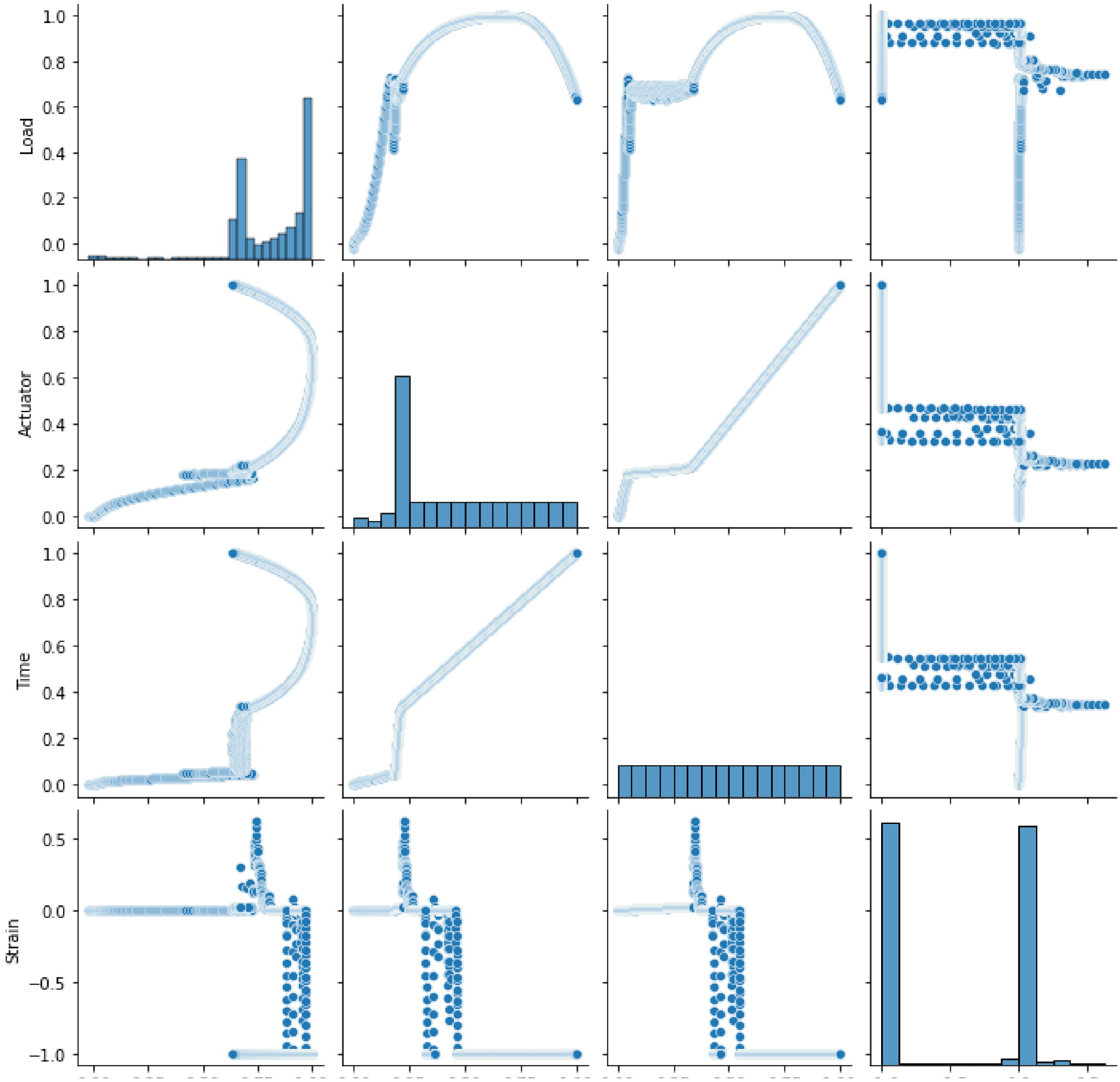


Pairplot

Pairplot is usually a grid of plots for each variable in your dataset. Hence you can quickly see how all the variables are related. This can help to infer which variables are useful, which have skewed distribution etc.

```
sns.pairplot(df2_scaled)
```

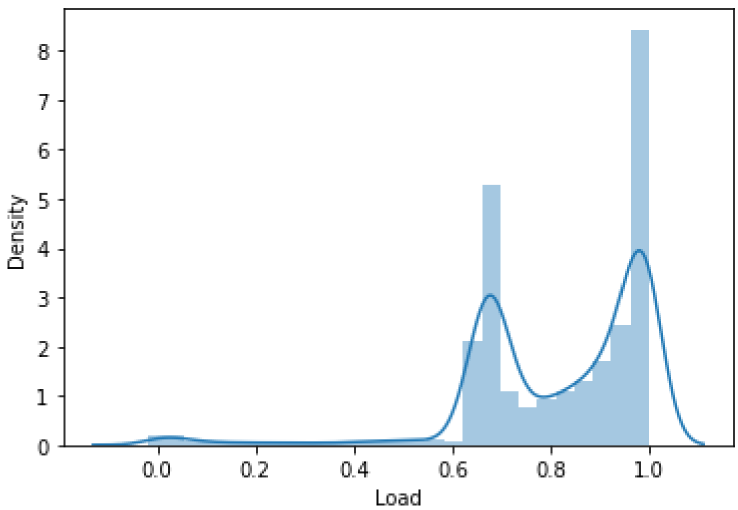
<seaborn.axisgrid.PairGrid at 0x20e66487e80>



Displot

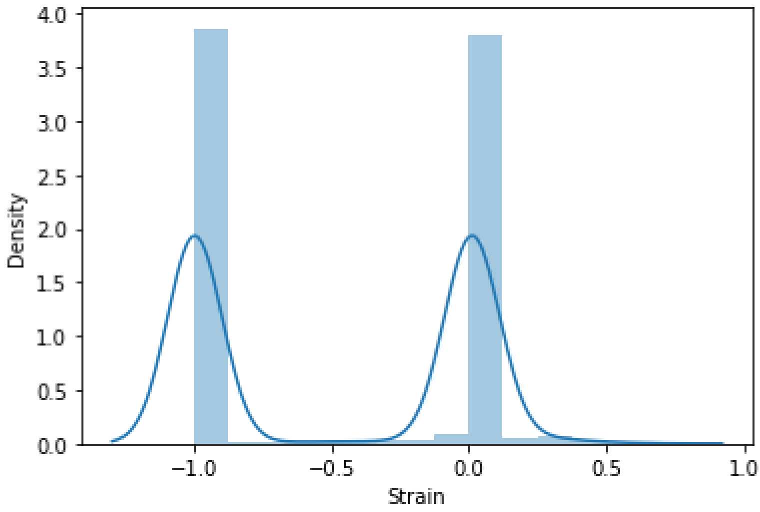
```
sns.distplot(df2_scaled['Load'])
```

<AxesSubplot:xlabel='Load', ylabel='Density'>



```
sns.distplot(df2_scaled['Strain'])
```

```
<AxesSubplot:xlabel='Strain', ylabel='Density'>
```



Linear Regression model

```
df_scaled.columns
```

```
Index(['Load', 'Actuator', 'Time', 'Strain'], dtype='object')
```

Getting the predicted value and train variables in two different dataframes

```
X = df_scaled.drop(['Strain'],axis=1)
Y = df_scaled['Strain']
```

```
X.head
```

```
<bound method NDFrame.head of           Load  Actuator      Time
0    -0.011925  0.000000  0.000402
1    -0.011955  0.000156  0.000668
2    -0.011618  0.000156  0.000930
3    -0.009971  0.000156  0.001197
4    -0.008570  0.000312  0.001463
...        ...       ...       ...
7363  0.639864  0.998594  0.998939
7364  0.638886  0.998907  0.999205
7365  0.637644  0.999219  0.999467
7366  0.635310  0.999688  0.999734
7367  0.631071  1.000000  1.000000

[7368 rows x 3 columns]>
```

```
Y.head
```

```
<bound method NDFrame.head of 0       0.000028
1       0.000010
```

```
2       0.000007
3       0.000036
4       0.000041
          ...
7363    0.001788
7364    0.001789
7365    0.001805
7366    0.001795
7367    0.001789
Name: Strain, Length: 7368, dtype: float64>
```

Importing the linear regression module

```
linear_reg_model = linearreg()
```

Fitting the train dataset in the module

```
linear_reg_model.fit(X,Y)
```

```
LinearRegression()
```

```
intercept = linear_reg_model.intercept_
intercept
```

```
0.05511739260609566
```

Coefficients of the linear regression model

```
coefficients = pd.DataFrame(linear_reg_model.coef_, X.columns, columns = ['coef']).sort_va
coefficients
```

	coef
Time	0.227245
Load	-0.048192
Actuator	-0.252603

The final linear regression equation is will be

```
reg_equation = "Y = " + str(intercept.round(5)) + " + "
reg_equation += "(" + str(coefficients.coef['Time'].round(5)) + ")" + X.columns[0]
for i in range(1, len(X.columns)):
    col = X.columns[i]
    reg_equation += " + (" + str(coefficients.coef[col].round(5)) + ")" + col
print(reg_equation)
```

```
Y = 0.05512 + (0.22724)Load + (-0.2526)Actuator + (0.22724)Time
```

From the regression equation, it is evident that the variables P2, P3, P4 and P8 move in opposite direction as compared to Y since they have the negative correlation coefficient. the variables P6 move in the same direction as compared to Y since they have the positive correlation coefficient.

Summary of result using the statsmodels api

```
result = sm.OLS(Y, X).fit()
print(result.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                 Strain   R-squared (uncentered):                   0.(
Model:                            OLS   Adj. R-squared (uncentered):              0.(
Method:                 Least Squares   F-statistic:                              25(
Date:                Thu, 24 Feb 2022   Prob (F-statistic):                   4.99e-1
Time:                        12:48:29   Log-Likelihood:                          102(
No. Observations:                7368   AIC:                                -2.058e-
Df Residuals:                    7365   BIC:                                -2.056e-
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Load           0.0337      0.002     14.580      0.000       0.029       0.038
Actuator      -0.1740      0.013    -13.142      0.000      -0.200      -0.148
Time           0.1282      0.014      9.369      0.000       0.101       0.155
==============================================================================
Omnibus:                    11326.586   Durbin-Watson:                   0.087
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4413857.749
Skew:                           9.837   Prob(JB):                         0.00
Kurtosis:                     121.281   Cond. No.                         30.1
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

The p statistic values of all columns are 0 so we move on to check the Variance Inflation Factor or VIF value

Based off of the VIF Values, we drop the ones with the highest VIF values.

Drop unnecessary input variables

```
vif  = pd.DataFrame()
vif['variables'] = X.columns
```

```
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif
```

	variables	VIF
0	Load	7.642338
1	Actuator	97.674635
2	Time	122.669998

The time factor in here is having a huge vif value so we drop it

```
X = X.drop(columns='Time')
```

```
vif  = pd.DataFrame()
vif['variables'] = X.columns
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif
```

	variables	VIF
0	Load	5.236635
1	Actuator	5.236635

Now that we have the VIF values sorted out the remaining columns are load and actuator

```
result = sm.OLS(Y, X).fit()
print(result.summary())
```

```
                                 OLS Regression Results
=======================================================================================
Dep. Variable:                 Strain   R-squared (uncentered):                   0.6
Model:                            OLS   Adj. R-squared (uncentered):              0.6
Method:                 Least Squares   F-statistic:                              328
Date:                Thu, 24 Feb 2022   Prob (F-statistic):                   2.03e-1
Time:                        12:52:56   Log-Likelihood:                          1024
No. Observations:                7368   AIC:                                -2.049e+
Df Residuals:                    7366   BIC:                                -2.048e+
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Load           0.0458      0.002     23.824      0.000       0.042       0.050
Actuator      -0.0533      0.003    -17.293      0.000      -0.059      -0.047
==============================================================================
Omnibus:                11449.376   Durbin-Watson:                   0.086
Prob(Omnibus):              0.000   Jarque-Bera (JB):          4597508.607
Skew:                      10.049   Prob(JB):                         0.00
```

```
Kurtosis:                     123.713   Cond. No.                         4.89
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a
[2] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

We calculate the linear regression model again

```
regression_model = linearreg()
regression_model.fit(X,Y)
print("Intercept : ",regression_model.intercept_)
print("Coefficients : ",regression_model.coef_)

x_train = np.column_stack((X['Load'],X['Actuator']))
y_train = Y
x_train = sm.add_constant(x_train)
estimate = sm.OLS(y_train, x_train).fit()
print(estimate.summary())
```

```
Intercept :  0.032805781754241016
Coefficients :  [ 0.0026777  -0.04465247]
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Strain   R-squared:                       0.038
Model:                            OLS   Adj. R-squared:                  0.038
Method:                 Least Squares   F-statistic:                     147.2
Date:                Thu, 24 Feb 2022   Prob (F-statistic):           2.03e-63
Time:                        12:54:38   Log-Likelihood:                 10300.
No. Observations:                7368   AIC:                        -2.059e+04
Df Residuals:                    7365   BIC:                        -2.057e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0328      0.003     10.265      0.000       0.027       0.039
x1             0.0027      0.005      0.580      0.562      -0.006       0.012
x2            -0.0447      0.003    -14.057      0.000      -0.051      -0.038
==============================================================================
Omnibus:                    11602.996   Durbin-Watson:                   0.087
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4916395.469
Skew:                          10.308   Prob(JB):                         0.00
Kurtosis:                     127.857   Cond. No.                         11.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

Final Linear Regression equation will be

```
print("Regression Equation: ")
yx = f"Y = {regression_model.intercept_.round(3)}+ ({regression_model.coef_[0].round(3)})L
print(yx)
```

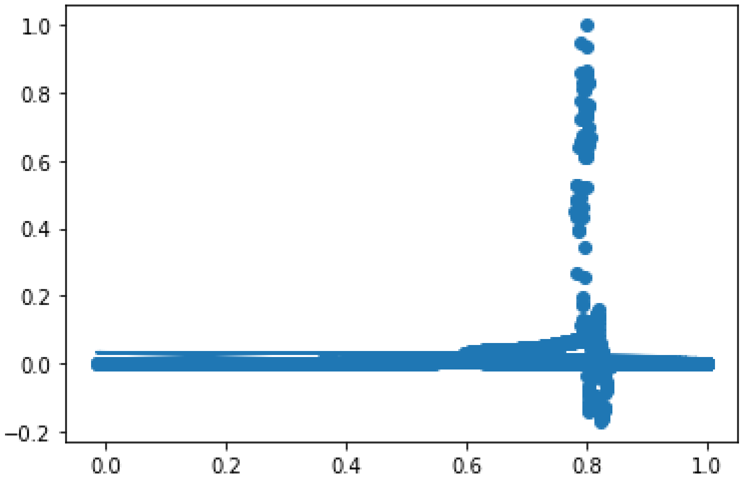
```
Regression Equation:
Y = 0.033+ (0.003)Load + (-0.045)Actuator
```

Performance Metrics for Linear Regression

Scatter plot for load

```
plt.scatter(X['Load'],Y)
eq = X*regression_model.coef_
eq = eq.sum(axis=1)+regression_model.intercept_
plt.plot(X['Load'], eq)
```

```
[<matplotlib.lines.Line2D at 0x20e66def970>]
```



Scatter plot for Actuator

```
plt.scatter(X['Actuator'],Y)
eq = X*regression_model.coef_
eq = eq.sum(axis=1)+regression_model.intercept_
plt.plot(X['Actuator'], eq)
```

```
[<matplotlib.lines.Line2D at 0x20e66e40df0>]
```

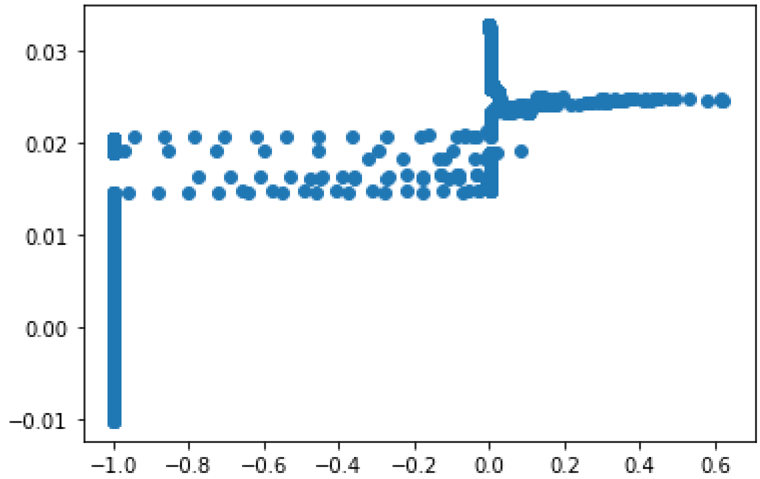
Finding and comparing the prediction values with actual values

[ ] ↳ *4 cells hidden*

Scatterplot of the predicted values

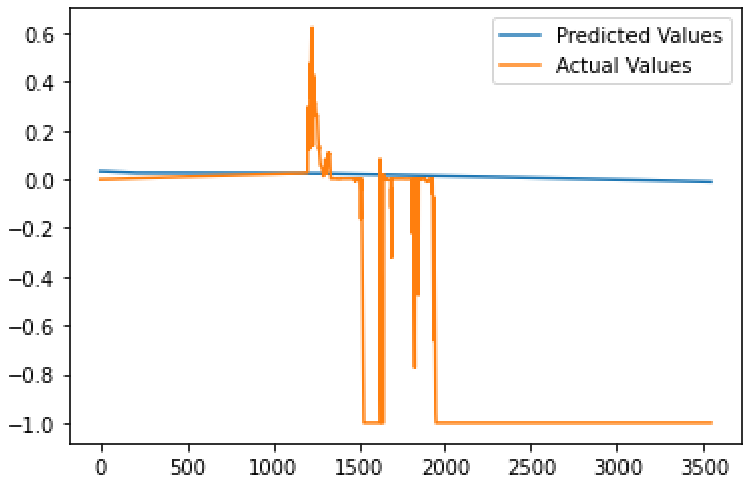
```
plt.scatter(Y_test, predictions)
```

```
<matplotlib.collections.PathCollection at 0x20e66ea4f70>
```




```
plt.plot(predictions, label = 'Predicted Values')
plt.plot(Y_test, label = 'Actual Values')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x20e66efaf70>
```



Comparing the linear regression predicted vs actual values

```
predictions_list = predictions.tolist()
Y_test_list = Y_test.tolist()
```

```
print("Actual\tPredicted")
for i in range(len(Y_test_list)):
    print(Y_test_list[i], "\t" ,predictions_list[i])
```

```
Actual  Predicted
2.6138867885776566e-06   0.03275367434889917
4.477251627959748e-06    0.03275740053998727
2.0988734510262172e-05   0.03276526339588207
3.1314881328504593e-05   0.032773273662280694
3.0538479312095397e-05   0.03277102369396607
4.6842921656688695e-05   0.032761491375983234
4.6325320312415895e-05   0.03275510486779854
4.73605230009615e-05     0.03274315080582426
6.262976265700919e-05    0.0327394975506512
6.4441367361964e-05      0.032727800947575304
6.23709619848728e-05     0.03271724337934869
8.100461037869371e-05    0.032700745315550915
8.592182314928534e-05    0.03268923331526035
8.773342785424015e-05    0.03267190669574129
0.00010455547154310626   0.0326554241942244
0.0001112842890186527    0.03263722719040447
0.00011853070783847195   0.03261847408555368
0.00013664675488802007   0.03259432028875393
0.0001415639676586117    0.032568236348406776
0.00015113959252765856   0.032542418969726455
0.0001772784604134351    0.03251777647653752
0.00018193687251189035   0.032490434021448134
0.0001899596933481188    0.03246459928636978
0.00021454575720107695   0.032430499317350696
0.0002199805713159414    0.03240227787374972
0.00022722699013576065   0.03236597551028613
0.00024430783449676317   0.03233115598706537
0.0002497426486116276    0.032295111439138484
0.0002481898445788092    0.03225859765708479
0.00026212367276663305   0.03221521982814167
0.0002536246586936736    0.032177507792295515
0.0002762593654787233    0.03213981226465383
0.0003112440403381221    0.03209890608602637
0.00031490348184213076   0.0320578929219614
0.0003346965572471228    0.03200958113267072
0.00035655227400904185   0.031969872313805735
0.00036277901818064376   0.031929042082156744
0.0003700306130139057    0.03188153289890418
0.0003970028190639615    0.03184247406186097
0.000407479070272043     0.031795415737370575
0.00042201072801250196   0.031749029132656426
0.00044186332757208536   0.03170300884644084
0.0004477225747892534    0.03165623121934489
0.0004633489593728494    0.0316111449349126
0.0004851140958995208    0.031565864876995434
0.0004998838502583452    0.03151998334006744
0.0005127643597105739    0.031467956155622874
0.0005395761093439051    0.03142329519391344
0.0005477102144691522    0.03136839851456344
0.0005421718800854333    0.031318232324910496
0.0005573168954188555    0.03126106376364996
0.000560704596217121     0.031205582350307192
0.0005698713160241923    0.0311453319873348
0.0005962146364409567    0.031096126361473565
```

```
0.0006065770153532981    0.031045563364125534
0.0006158213753620104    0.030988545418001747
```

Final Evaluation metrics and tabulations for linear regression

```
MAE = metrics.mean_absolute_error(Y_test, predictions)
MSE = metrics.mean_squared_error(Y_test,predictions)
RMSE = np.sqrt(MSE)
R_squared = result.rsquared
adjusted_R_squared = result.rsquared_adj

print("MSE (Mean Squared Error) : ", + MSE)
print("MAE (Mean Absolute Error): ", + MAE)
print("RMSE (Root Mean Squared Error ) : ", + RMSE)
print("Adjusted R squared value : " , adjusted_R_squared)
```

```
MSE (Mean Squared Error) :  0.49089390550979306
MAE (Mean Absolute Error):  0.5030271563547885
RMSE (Root Mean Squared Error ) :  0.7006382129956894
Adjusted R squared value :  0.08166011217622104
```

```
table_testing = [
              ['Input Variable Names', 'Regression', 'MSE', 'MAE', 'RMSE', 'R-Squared',
              [[X.columns[0],X.columns[1]], yx, MSE, MAE, RMSE, R_squared, adjusted_R_sq
              ]
print(tabulate(table_testing, headers='firstrow'))
```

```
Input Variable Names    Regression                                     MSE       MAE
----------------------  ---------------------------------------  --------  --------
['Load', 'Actuator']    Y = 0.033+ (0.003)Load + (-0.045)Actuator  0.490894  0.503027
```

Final Accuracy score will be

```
r2_score = regression_model.score(X_test,Y_test)
print(-(r2_score*100),'%')
```

```
88.90679416276302 %
```

Linear Regression model is successfully implemented with a accuracy of 88.9%

Implementation of the Polynomial Linear Regression model

The data is already scaled up and the data is ready to go and the polynomial linear regression can be implemented with polynomial features and the linear regression model and we assume the degree is equal to 9

```
degree=9
polyreg=make_pipeline(PolynomialFeatures(degree),LinearRegression())
polyreg.fit(X,Y)
```

```
Pipeline(steps=[('polynomialfeatures', PolynomialFeatures(degree=9)),
                ('linearregression', LinearRegression())])
```

Display of polyreg is not possible because of pipline issues

There fore the final regression models are implemented and then the final model is linear regression

```
print("Regression Equation is : " , yx)
```

```
Regression Equation is :  Y = 0.033+ (0.003)Load + (-0.045)Actuator
```