LearningRegression

April 8, 2021

#The Data ### This data has the following inputs: 1. Frequency, in Hertzs. 2. Angle of attack, in degrees. 3. Chord length, in meters. 4. Free-stream velocity, in meters per second. 5. Suction side displacement thickness, in meters.

0.0.1 The only output is (and Y_data):

6. Scaled sound pressure level, in decibels.

[1]:
```
import pandas as pd;

# Set the display format for the Floating Point numbers
pd.options.display.float_format = "{:,.2f}".format
```

[2]:
```
# To store dataset in a Pandas Dataframe
df_raw = pd.read_csv('airfoil_self_noise.dat.txt', sep='\t', header=None)
df_raw
```

[2]:

	0	1	2	3	4	5
0	800	0.00	0.30	71.30	0.00	126.20
1	1000	0.00	0.30	71.30	0.00	125.20
2	1250	0.00	0.30	71.30	0.00	125.95
3	1600	0.00	0.30	71.30	0.00	127.59
4	2000	0.00	0.30	71.30	0.00	127.46
…	…	…	…	…	…	…
1498	2500	15.60	0.10	39.60	0.05	110.26
1499	3150	15.60	0.10	39.60	0.05	109.25
1500	4000	15.60	0.10	39.60	0.05	106.60
1501	5000	15.60	0.10	39.60	0.05	106.22
1502	6300	15.60	0.10	39.60	0.05	104.20

[1503 rows x 6 columns]

[3]:
```
df = df_raw.rename(columns = {0:'Frequency', \
                              1:'AttackAngle', \
                              2:'ChordLength', \
                              3:'FreeStreamVelocity', \
                              4:'Thickness', \
                              5:'ScaledSoundPressureLevel'}, \
                   inplace=False)
```

```
df.head()
```

[3]:

	Frequency	AttackAngle	…	Thickness	ScaledSoundPressureLevel
0	800	0.00	…	0.00	126.20
1	1000	0.00	…	0.00	125.20
2	1250	0.00	…	0.00	125.95
3	1600	0.00	…	0.00	127.59
4	2000	0.00	…	0.00	127.46

[5 rows x 6 columns]

[4]:
```
df.shape
```

[4]: (1503, 6)

0.1 Let us see the structure of our data

[5]:
```
df.dtypes
```

[5]:
```
Frequency                   int64
AttackAngle                 float64
ChordLength                 float64
FreeStreamVelocity          float64
Thickness                   float64
ScaledSoundPressureLevel    float64
dtype: object
```

0.1.1 Let us check if we have any missing values

[6]:
```
df.isnull().sum()
```

[6]:
```
Frequency                   0
AttackAngle                 0
ChordLength                 0
FreeStreamVelocity          0
Thickness                   0
ScaledSoundPressureLevel    0
dtype: int64
```

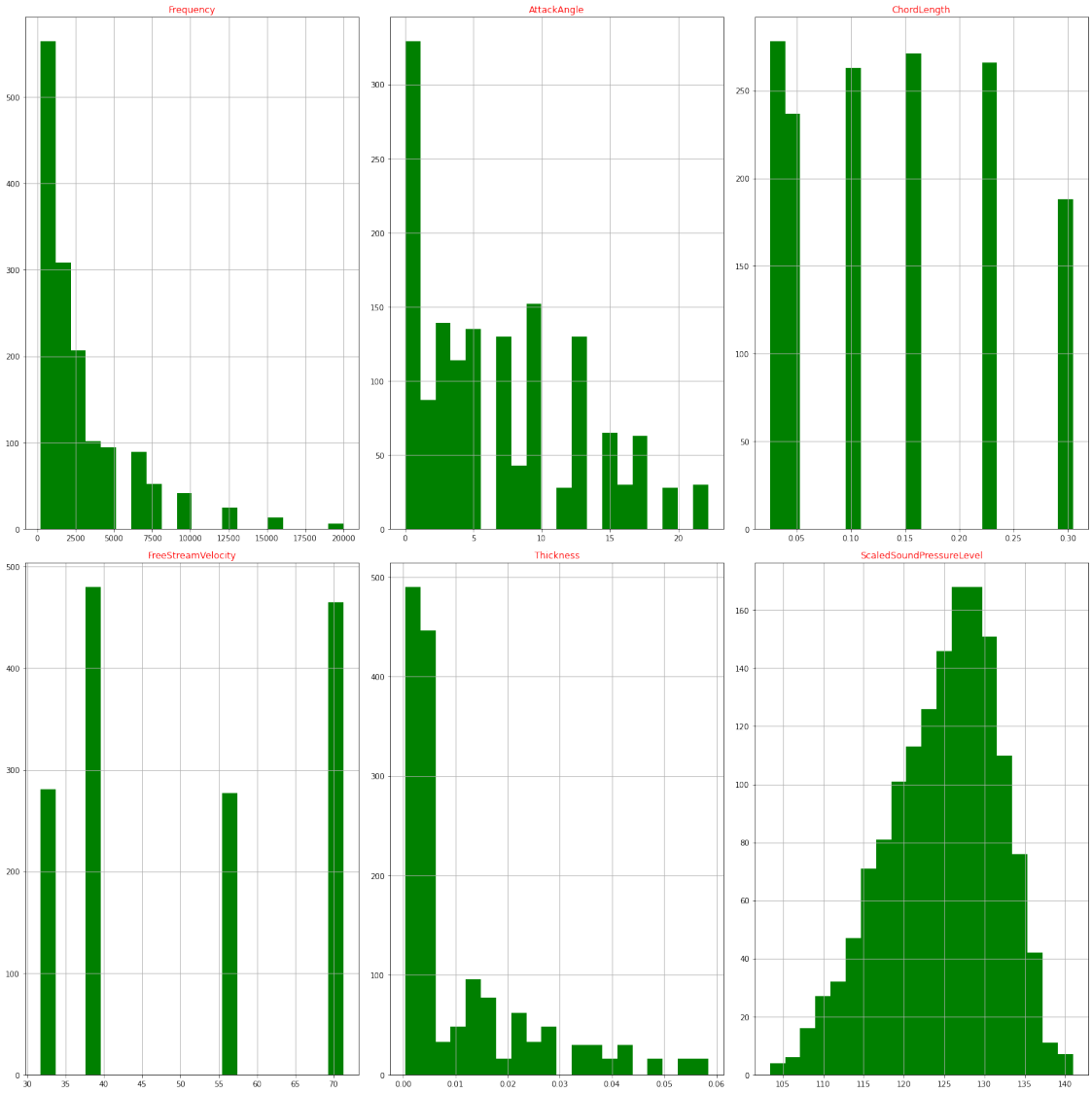
0.1.2 Let us create histograms for each olumn to check the distribution of data in each column.

[7]:
```
import matplotlib.pyplot as plt;

fig = plt.figure(figsize=(20,20))
for i, feature in enumerate(df.columns):
    ax = fig.add_subplot(2, 3, i+1)
    df[feature].hist(bins=20, ax=ax, facecolor='green')
```

```
    ax.set_title(feature, color='red')

fig.tight_layout()
plt.show()
```



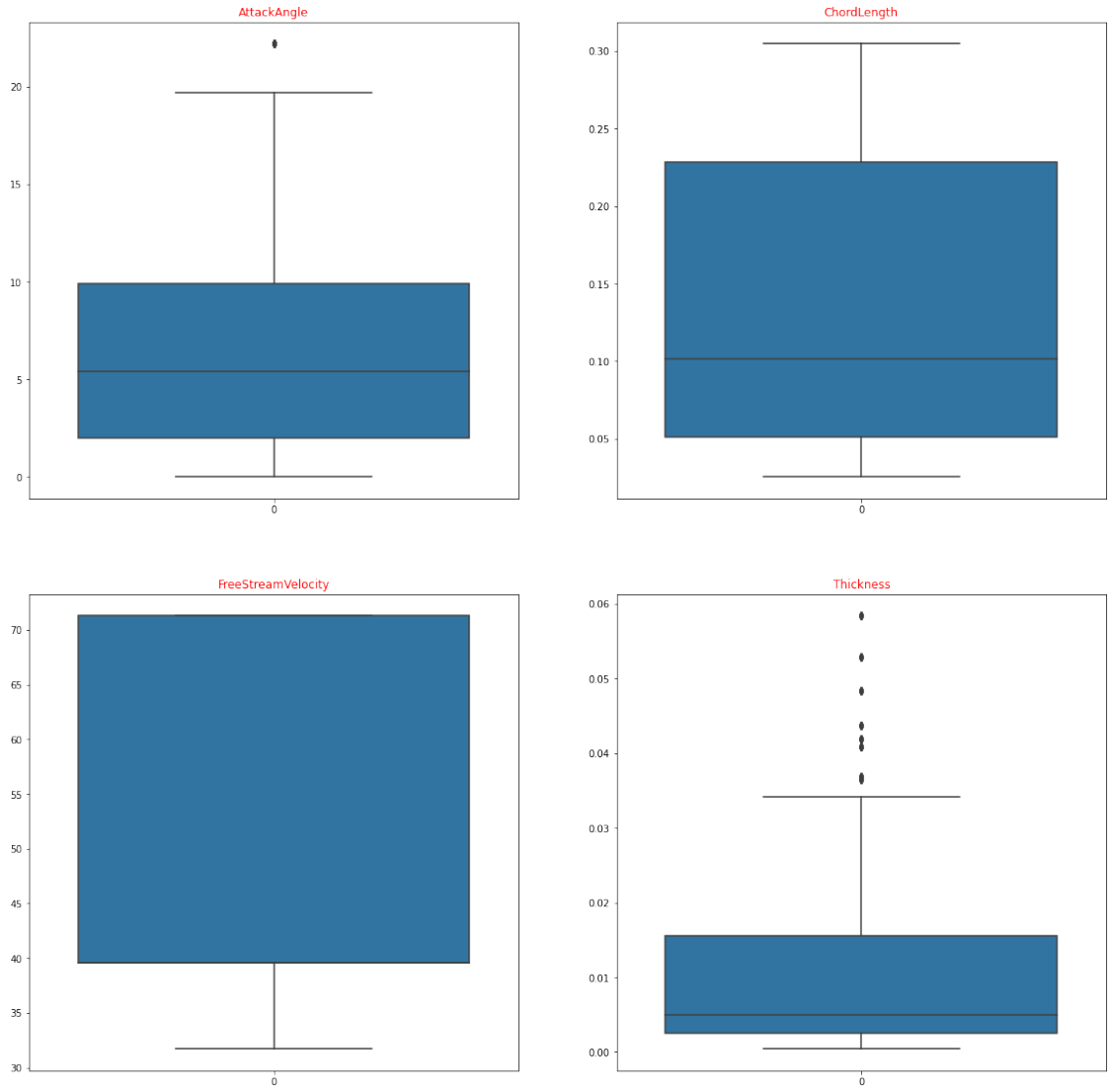
0.1.3 Box Plot

[8]:
```
import seaborn as sb;

fig = plt.figure(figsize=(20,20))
for i in range(5):
    if i > 0:
        ax = fig.add_subplot(2, 2, i)
```

```
        sb.boxplot(data=df.iloc[:,i])
        ax.set_title(df.columns[i], color='red')

plt.show()
```



Check the number of rows where AttackAngle > 20

[9]:
```
len(df.query('AttackAngle > 20'))
```

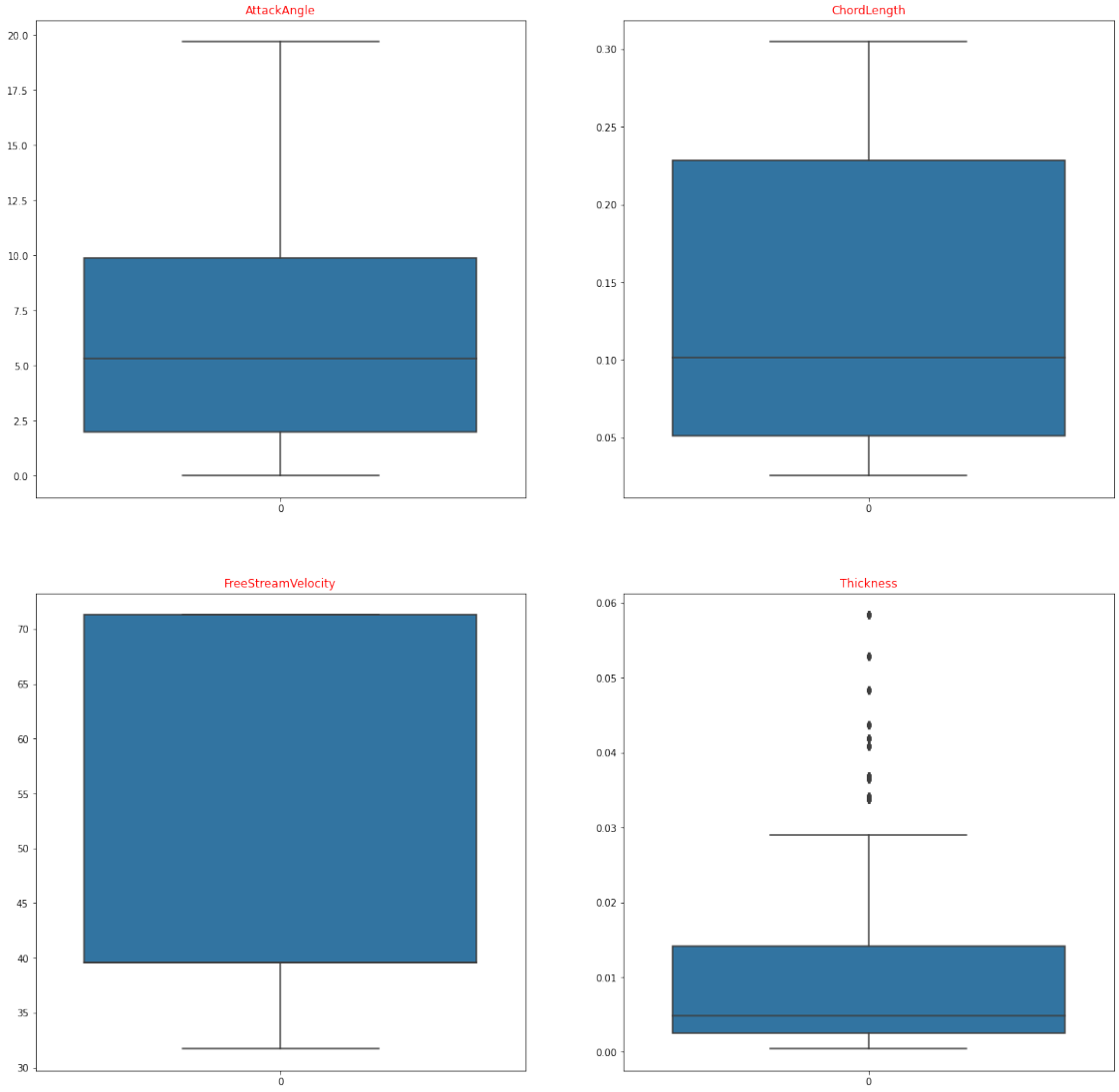
[9]: 30

[10]:
```
df_clean = df.copy()
df_clean = df_clean.query('AttackAngle <= 20')
df_clean.shape
```

[10]: (1473, 6)

[11]:
```
fig = plt.figure(figsize=(20,20))
for i in range(5):
    if i > 0:
        ax = fig.add_subplot(2, 2, i)
        sb.boxplot(data=df_clean.iloc[:,i])
        ax.set_title(df_clean.columns[i], color='red')

plt.show()
```



0.1.4 Correlations between the dependent variable and the independent variables

[12]:
```
# Create an empty Data Frame to store all the Correlations
corrdf = pd.DataFrame(columns = ['Dependent Variable', 'Feature',␣
↪'Correlation']);

# Loop through all the Features in scope
df1 = df['ScaledSoundPressureLevel'];
for j in df_clean.iloc[:,1:5].columns:
    df2 = df[j];
    c = df1.corr(df2);

    # Add the computed Correlation to the Data Frame
    corrdf = corrdf.append({'Dependent Variable':'ScaledSoundPressureLevel',␣
↪'Feature':j, 'Correlation':c}, ignore_index = True);

corrdf
```

[12]:

	Dependent Variable	Feature	Correlation
0	ScaledSoundPressureLevel	AttackAngle	-0.16
1	ScaledSoundPressureLevel	ChordLength	-0.24
2	ScaledSoundPressureLevel	FreeStreamVelocity	0.13
3	ScaledSoundPressureLevel	Thickness	-0.31

0.2 Build our Regression Models

0.2.1 Create the Training and Test Sets

[13]:
```
from sklearn.model_selection import train_test_split

df_train, df_test  = train_test_split(df_clean, train_size=0.7, random_state=44)

y_train = df_train[['ScaledSoundPressureLevel']]
x_train = df_train.drop("ScaledSoundPressureLevel", axis=1)

y_test = df_test[['ScaledSoundPressureLevel']]
x_test = df_test.drop("ScaledSoundPressureLevel", axis=1)
```

[14]:
```
df_train.shape
```

[14]:
```
(1031, 6)
```

[15]:
```
df_test.shape
```

[15]:
```
(442, 6)
```

[16]:
```
y_train.head()
```

[16]:
```
      ScaledSoundPressureLevel
718                     110.78
1079                    128.41
1361                    130.96
1295                    131.43
136                     120.16
```

[17]:
```
x_train.head()
```

[17]:
```
      Frequency  AttackAngle  ChordLength  FreeStreamVelocity  Thickness
718        1600        12.60         0.15                39.60       0.06
1079       5000         9.50         0.03                55.50       0.00
1361        500         6.70         0.10                39.60       0.01
1295       1000         3.30         0.10                71.30       0.00
136        2500         3.00         0.30                39.60       0.00
```

0.2.2 Build the Linear Regression Model

[18]:
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression(normalize = True)
lr.fit(x_train, y_train)
```

[18]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)
```

[19]:
```
y_pred_train = lr.predict(x_train)
y_pred_train
```

[19]:
```
array([[116.05933638],
       [126.1557078 ],
       [129.04766575],
       …,
       [124.39223663],
       [134.25775043],
       [122.82943923]])
```

Check the Metrics

[20]:
```
import sklearn.metrics as sm

print("Mean absolute error =", round(sm.mean_absolute_error(y_train,␣
 ↪y_pred_train), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_train,␣
 ↪y_pred_train), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_train,␣
 ↪y_pred_train), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_train,␣
 ↪y_pred_train), 2))
```

```
print("R2 score =", round(sm.r2_score(y_train, y_pred_train), 2))
```

```
Mean absolute error = 3.57
Mean squared error = 21.52
Median absolute error = 2.84
Explain variance score = 0.54
R2 score = 0.54
```

[21]:
```
y_pred_test = lr.predict(x_test)

print("Mean absolute error =", round(sm.mean_absolute_error(y_test,␣
 ↪y_pred_test), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred_test),␣
 ↪2))
print("Median absolute error =", round(sm.median_absolute_error(y_test,␣
 ↪y_pred_test), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test,␣
 ↪y_pred_test), 2))
print("R2 score =", round(sm.r2_score(y_test, y_pred_test), 2))
```

```
Mean absolute error = 3.95
Mean squared error = 25.19
Median absolute error = 3.24
Explain variance score = 0.5
R2 score = 0.5
```

0.3 Build the Regression Model using Random Forest Algorithm

[22]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

rf = make_pipeline(StandardScaler(), RandomForestRegressor())
rf.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/pipeline.py:354:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  self._final_estimator.fit(Xt, y, **fit_params)
```

[22]:
```
Pipeline(memory=None,
         steps=[('standardscaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('randomforestregressor',
                 RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                       criterion='mse', max_depth=None,
                                       max_features='auto', max_leaf_nodes=None,
```

```
                                                max_samples=None,
                                                min_impurity_decrease=0.0,
                                                min_impurity_split=None,
                                                min_samples_leaf=1, min_samples_split=2,
                                                min_weight_fraction_leaf=0.0,
                                                n_estimators=100, n_jobs=None,
                                                oob_score=False, random_state=None,
                                                verbose=0, warm_start=False))],
             verbose=False)
```

[23]:
```
y_pred_train = rf.predict(x_train)
y_pred_train
```

[23]:
```
array([111.90273, 127.27859, 131.07173, …, 118.71893, 129.00899,
       130.60024])
```

[24]:
```
print("Mean absolute error =", round(sm.mean_absolute_error(y_train,␣
 ↪y_pred_train), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_train,␣
 ↪y_pred_train), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_train,␣
 ↪y_pred_train), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_train,␣
 ↪y_pred_train), 2))
print("R2 score =", round(sm.r2_score(y_train, y_pred_train), 2))
```

```
Mean absolute error = 0.48
Mean squared error = 0.45
Median absolute error = 0.36
Explain variance score = 0.99
R2 score = 0.99
```

[25]:
```
y_pred_test = rf.predict(x_test)

print("Mean absolute error =", round(sm.mean_absolute_error(y_test,␣
 ↪y_pred_test), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred_test),␣
 ↪2))
print("Median absolute error =", round(sm.median_absolute_error(y_test,␣
 ↪y_pred_test), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test,␣
 ↪y_pred_test), 2))
print("R2 score =", round(sm.r2_score(y_test, y_pred_test), 2))
```

```
Mean absolute error = 1.33
Mean squared error = 3.16
Median absolute error = 0.94
Explain variance score = 0.94
```

```
R2 score = 0.94
```

0.4 Build Regression Model using SVM

[29]:
```
from sklearn.svm import SVR

svr = make_pipeline(StandardScaler(), SVR(C = 30.0, epsilon = 0.9))
svr.fit(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[29]:
```
Pipeline(memory=None,
         steps=[('standardscaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('svr',
                 SVR(C=30.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.9,
                     gamma='scale', kernel='rbf', max_iter=-1, shrinking=True,
                     tol=0.001, verbose=False))],
         verbose=False)
```

[30]:
```
y_pred_train = svr.predict(x_train)

print("Mean absolute error =", round(sm.mean_absolute_error(y_train,␣
 ↪y_pred_train), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_train,␣
 ↪y_pred_train), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_train,␣
 ↪y_pred_train), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_train,␣
 ↪y_pred_train), 2))
print("R2 score =", round(sm.r2_score(y_train, y_pred_train), 2))
```

```
Mean absolute error = 1.84
Mean squared error = 6.77
Median absolute error = 1.19
Explain variance score = 0.85
R2 score = 0.85
```

[31]:
```
y_pred_test = svr.predict(x_test)

print("Mean absolute error =", round(sm.mean_absolute_error(y_test,␣
 ↪y_pred_test), 2))
print("Mean squared error =", round(sm.mean_squared_error(y_test, y_pred_test),␣
 ↪2))
```

```
print("Median absolute error =", round(sm.median_absolute_error(y_test,␣
 ↪y_pred_test), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test,␣
 ↪y_pred_test), 2))
print("R2 score =", round(sm.r2_score(y_test, y_pred_test), 2))
```

```
Mean absolute error = 2.16
Mean squared error = 8.73
Median absolute error = 1.59
Explain variance score = 0.83
R2 score = 0.83
```