code

August 17, 2023

0.1 Assignment 1

0.1.1 Name: Ritesah M

0.1.2 Roll Number: 21CS30042

[1]:
```python
# import all the necessary libraries here
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

[2]:
```python
df = pd.read_csv('../../dataset/linear-regression.csv')
print(df.shape)
```

```
(1599, 12)
```

[3]:
```python
df.isna().any()
```

[3]:
```
fixed acidity           False
volatile acidity        False
citric acid             False
residual sugar          False
chlorides               False
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                      False
sulphates               False
alcohol                 False
quality                 False
dtype: bool
```

[4]:
```python
def print_max(dataframe):
    for column in dataframe.columns:
        print(f"Max for {column}: {dataframe[column].max()}")

def print_min(dataframe):
    for column in dataframe.columns:
        print(f"Min for {column}: {dataframe[column].min()}")
```

[5]:
```
X=df.drop("quality",axis=1)
y=df["quality"]
print(X.shape)
print(y.shape)
```

```
(1599, 11)
(1599,)
```

[6]:
```
print(print_max(df))
print(print_min(df))
```

```
Max for fixed acidity: 15.9
Max for volatile acidity: 1.58
Max for citric acid: 1.0
Max for residual sugar: 15.5
Max for chlorides: 0.611
Max for free sulfur dioxide: 72.0
Max for total sulfur dioxide: 289.0
Max for density: 1.00369
Max for pH: 4.01
Max for sulphates: 2.0
Max for alcohol: 14.9
Max for quality: 8
None
Min for fixed acidity: 4.6
Min for volatile acidity: 0.12
Min for citric acid: 0.0
Min for residual sugar: 0.9
Min for chlorides: 0.012
Min for free sulfur dioxide: 1.0
Min for total sulfur dioxide: 6.0
Min for density: 0.99007
Min for pH: 2.74
Min for sulphates: 0.33
Min for alcohol: 8.4
Min for quality: 3
None
```

[7]:
```
df.columns
```

[7]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

[8]:
```
cols_to_scale=[]
for column in df.columns:
```

```
    if column != "quality":
        cols_to_scale.append(column)
print(cols_to_scale)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH',
'sulphates', 'alcohol']
```

[9]:

```
from sklearn.model_selection import train_test_split
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.5,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.4,␣
 ↪random_state=42)
```

[10]:

```
print(X_train.shape)
print(y_train.shape)
print(X_val.shape)
print(y_val.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(799, 11)
(799,)
(480, 11)
(480,)
(320, 11)
(320,)
```

[11]:

```
X_train=X_train.to_numpy()
ones_column_train= np.ones((X_train.shape[0],1))
y_train=y_train.to_numpy(dtype="float64")
X_train=np.hstack((ones_column_train,X_train))

X_val=X_val.to_numpy()
ones_column_val= np.ones((X_val.shape[0],1))
y_val=y_val.to_numpy(dtype="float64")
X_val=np.hstack((ones_column_val,X_val))

X_test=X_test.to_numpy()
ones_column_test= np.ones((X_test.shape[0],1))
y_test=y_test.to_numpy(dtype="float64")
X_test=np.hstack((ones_column_test,X_test))
```

1 Analytical Solution

[12]:
```
df.columns.shape
```

[12]:
```
(12,)
```

[13]:
```
df
```

[13]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
…               …                …           …              …          …
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
…                    …                   …      …    …        …
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
…       …      …
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6
```

```
[1599 rows x 12 columns]
```

[14]:
```python
X_train.shape
```

[14]:
```
(799, 12)
```

[15]:
```python
print(X_train.shape)
print(y_train)
```

```
(799, 12)
[5. 5. 7. 6. 6. 5. 6. 5. 6. 5. 5. 5. 6. 5. 7. 7. 5. 5. 5. 5. 5. 5. 5. 6.
 6. 3. 5. 5. 6. 6. 5. 6. 7. 5. 5. 7. 6. 6. 6. 6. 7. 6. 6. 5. 7. 7. 7. 5.
 5. 6. 5. 5. 6. 5. 6. 6. 6. 8. 6. 5. 6. 5. 5. 6. 6. 6. 5. 4. 6. 6. 6. 6.
 6. 6. 5. 5. 8. 4. 5. 5. 6. 6. 6. 7. 5. 5. 6. 6. 6. 6. 6. 6. 6. 5. 5. 6.
 7. 6. 4. 5. 5. 5. 5. 5. 6. 6. 5. 4. 7. 6. 5. 7. 7. 5. 5. 5. 6. 6. 5. 6.
 5. 5. 6. 6. 6. 6. 5. 5. 7. 5. 6. 6. 5. 6. 5. 6. 5. 7. 5. 6. 5. 6. 5. 5.
 6. 6. 5. 5. 6. 6. 7. 6. 5. 5. 7. 6. 5. 5. 6. 7. 6. 6. 6. 7. 5. 6. 7. 6.
 6. 4. 7. 5. 5. 5. 5. 6. 6. 7. 6. 6. 5. 7. 6. 6. 6. 5. 6. 5. 7. 6. 5. 6.
 8. 5. 6. 6. 6. 6. 5. 6. 6. 5. 4. 6. 6. 5. 6. 5. 7. 5. 5. 6. 7. 5. 6. 5.
 6. 5. 6. 5. 7. 5. 6. 6. 6. 7. 6. 8. 6. 5. 6. 5. 6. 4. 5. 5. 6. 7. 5. 5.
 5. 6. 6. 5. 5. 7. 7. 5. 6. 7. 7. 5. 5. 5. 7. 6. 5. 6. 5. 5. 6. 6. 5. 7.
 6. 7. 6. 6. 5. 5. 5. 5. 5. 5. 7. 7. 5. 6. 5. 6. 5. 5. 5. 4. 6. 6. 8. 6.
 6. 5. 5. 5. 5. 6. 6. 5. 5. 5. 5. 5. 5. 7. 5. 5. 5. 5. 4. 5. 6. 6. 5. 5.
 6. 6. 5. 8. 5. 5. 5. 7. 3. 5. 7. 5. 5. 6. 6. 6. 5. 7. 6. 6. 5. 6. 6. 6.
 5. 6. 7. 5. 5. 5. 5. 6. 5. 5. 5. 6. 6. 6. 7. 5. 7. 7. 6. 6. 6. 7. 5. 5.
 6. 6. 5. 5. 4. 6. 6. 6. 6. 5. 5. 6. 4. 6. 5. 7. 6. 6. 6. 7. 4. 6. 5. 5.
 5. 7. 6. 5. 6. 5. 6. 5. 5. 5. 4. 6. 6. 5. 7. 5. 5. 6. 6. 7. 7. 5. 7. 5.
 5. 5. 5. 6. 6. 6. 6. 5. 5. 6. 4. 6. 5. 5. 6. 6. 5. 6. 7. 6. 5. 6. 7. 5.
 6. 7. 5. 6. 5. 6. 6. 6. 6. 5. 6. 5. 5. 6. 5. 5. 6. 5. 5. 5. 6. 4. 6. 5.
 5. 6. 6. 5. 6. 7. 5. 6. 6. 3. 6. 6. 3. 5. 5. 5. 5. 6. 7. 6. 6. 6. 6. 5.
 5. 5. 5. 6. 7. 7. 6. 5. 6. 6. 5. 7. 6. 6. 5. 5. 6. 5. 3. 6. 6. 6. 6. 6.
 6. 6. 5. 5. 6. 5. 7. 5. 6. 7. 5. 5. 7. 6. 6. 6. 5. 7. 6. 6. 5. 4. 6. 5.
 5. 5. 6. 5. 7. 6. 7. 6. 6. 5. 6. 7. 5. 5. 6. 6. 5. 6. 5. 6. 6. 6. 5. 5.
 5. 6. 5. 7. 5. 6. 5. 6. 6. 5. 4. 5. 5. 5. 7. 6. 6. 5. 5. 5. 6. 5. 7. 6.
 6. 5. 5. 5. 5. 5. 5. 6. 5. 6. 5. 6. 6. 5. 5. 5. 5. 5. 6. 7. 5. 6. 6. 6.
 6. 5. 5. 6. 6. 6. 5. 6. 7. 6. 5. 6. 7. 5. 6. 5. 5. 7. 6. 5. 5. 6. 5. 7.
 5. 5. 7. 5. 6. 6. 5. 6. 6. 5. 5. 7. 5. 6. 6. 7. 5. 6. 6. 6. 6. 6. 5. 5.
 5. 7. 7. 5. 6. 6. 6. 5. 4. 5. 6. 5. 6. 6. 5. 5. 5. 6. 6. 7. 5. 7. 7. 5.
 7. 6. 5. 5. 6. 7. 6. 7. 6. 6. 5. 7. 7. 4. 5. 6. 6. 5. 5. 6. 6. 7. 6. 5.
 5. 5. 5. 6. 6. 5. 5. 5. 7. 6. 5. 8. 6. 5. 6. 6. 5. 4. 6. 6. 6. 5. 6. 5.
 5. 6. 6. 6. 6. 7. 5. 5. 6. 7. 3. 6. 6. 6. 4. 6. 6. 4. 6. 6. 5. 6. 4. 6.
 6. 5. 7. 6. 5. 6. 6. 6. 6. 6. 5. 6. 5. 6. 5. 5. 6. 6. 5. 6. 6. 6. 5. 5.
 5. 5. 3. 5. 6. 5. 6. 6. 7. 5. 6. 6. 5. 6. 5. 5. 6. 5. 5. 4. 6. 4. 6. 6.
 6. 5. 6. 6. 5. 7. 6.]
```

[16]:
```python
Xt_train=np.transpose(X_train)
print(Xt_train)
```

```
[[ 1.    1.    1.   …  1.    1.    1.  ]
 [ 8.6   7.8   9.8  …  7.2   7.9   5.8 ]
 [ 0.52  0.56  0.5  …  0.62  0.2   0.29]
 …
 [ 3.2   3.19  3.24 …  3.51  3.32  3.39]
 [ 0.52  0.93  0.6  …  0.54  0.8   0.54]
 [ 9.4   9.5   9.7  …  9.5  11.9  13.5 ]]
```

[17]:
```
print(X_train)
```

```
[[ 1.    8.6   0.52 …  3.2   0.52  9.4 ]
 [ 1.    7.8   0.56 …  3.19  0.93  9.5 ]
 [ 1.    9.8   0.5  …  3.24  0.6   9.7 ]
 …
 [ 1.    7.2   0.62 …  3.51  0.54  9.5 ]
 [ 1.    7.9   0.2  …  3.32  0.8  11.9 ]
 [ 1.    5.8   0.29 …  3.39  0.54 13.5 ]]
```

[18]:
```
print(print_max(df))
print(print_min(df))
```

```
Max for fixed acidity: 15.9
Max for volatile acidity: 1.58
Max for citric acid: 1.0
Max for residual sugar: 15.5
Max for chlorides: 0.611
Max for free sulfur dioxide: 72.0
Max for total sulfur dioxide: 289.0
Max for density: 1.00369
Max for pH: 4.01
Max for sulphates: 2.0
Max for alcohol: 14.9
Max for quality: 8
None
Min for fixed acidity: 4.6
Min for volatile acidity: 0.12
Min for citric acid: 0.0
Min for residual sugar: 0.9
Min for chlorides: 0.012
Min for free sulfur dioxide: 1.0
Min for total sulfur dioxide: 6.0
Min for density: 0.99007
Min for pH: 2.74
Min for sulphates: 0.33
Min for alcohol: 8.4
Min for quality: 3
None
```

[19]:

```
XtX_train=np.matmul(Xt_train,X_train)
XtX_inv_train=np.linalg.inv(XtX_train)
print(XtX_inv_train)
print(XtX_inv_train.shape)
```

```
[[ 2.11574955e+03  2.01312462e+00  1.45140448e+00  1.57813370e-01
   9.20446869e-01  3.03217932e+00 -2.62021425e-02  6.21449171e-03
  -2.15934226e+03  1.07472422e+01  2.94881790e+00 -2.00578588e+00]
 [ 2.01312462e+00  3.22026683e-03 -7.49833859e-04 -5.68095055e-03
   8.45858728e-04  1.07236325e-02 -3.95347933e-05  2.13106856e-05
  -2.08656321e+00  1.70945780e-02  2.42460001e-03 -1.89179795e-03]
 [ 1.45140448e+00 -7.49833859e-04  7.61723554e-02  4.86762273e-02
  -1.63186639e-04 -6.86330524e-02  2.10830772e-04 -7.89870899e-05
  -1.50006530e+00 -1.58534290e-04  1.42887160e-02 -5.58737921e-04]
 [ 1.57813370e-01 -5.68095055e-03  4.86762273e-02  9.98141904e-02
  -1.07083492e-03 -7.65867936e-02  2.86912962e-04 -1.31695283e-04
  -1.53067400e-01  5.94837175e-03 -1.30489321e-03 -1.81272642e-03]
 [ 9.20446869e-01  8.45858728e-04 -1.63186639e-04 -1.07083492e-03
   1.06785746e-03 -7.27346859e-05 -2.16787295e-05 -2.26233802e-07
  -9.38606202e-01  4.61728012e-03  1.73878253e-03 -9.90361198e-04]
 [ 3.03217932e+00  1.07236325e-02 -6.86330524e-02 -7.65867936e-02
  -7.27346859e-05  8.64533559e-01 -3.46200057e-04  2.06937659e-04
  -3.41055689e+00  8.24564280e-02 -8.30325562e-02  3.62945602e-03]
 [-2.62021425e-02 -3.95347933e-05  2.10830772e-04  2.86912962e-04
  -2.16787295e-05 -3.46200057e-04  2.25100860e-05 -5.09878419e-06
   2.72403456e-02 -2.85639154e-04 -8.04907241e-05  1.43318450e-05]
 [ 6.21449171e-03  2.13106856e-05 -7.89870899e-05 -1.31695283e-04
  -2.26233802e-07  2.06937659e-04 -5.09878419e-06  2.53519681e-06
  -6.88504971e-03  1.30776574e-04 -5.64659174e-06  6.33502343e-06]
 [-2.15934226e+03 -2.08656321e+00 -1.50006530e+00 -1.53067400e-01
  -9.38606202e-01 -3.41055689e+00  2.72403456e-02 -6.88504971e-03
   2.20568454e+03 -1.14066435e+01 -3.03282330e+00  2.04116816e+00]
 [ 1.07472422e+01  1.70945780e-02 -1.58534290e-04  5.94837175e-03
   4.61728012e-03  8.24564280e-02 -2.85639154e-04  1.30776574e-04
  -1.14066435e+01  1.75223697e-01  1.62852054e-02 -1.27050469e-02]
 [ 2.94881790e+00  2.42460001e-03  1.42887160e-02 -1.30489321e-03
   1.73878253e-03 -8.30325562e-02 -8.04907241e-05 -5.64659174e-06
  -3.03282330e+00  1.62852054e-02  6.02353157e-02 -4.08166660e-03]
 [-2.00578588e+00 -1.89179795e-03 -5.58737921e-04 -1.81272642e-03
  -9.90361198e-04  3.62945602e-03  1.43318450e-05  6.33502343e-06
   2.04116816e+00 -1.27050469e-02 -4.08166660e-03  3.27380605e-03]]
(12, 12)
```

[20]:

```
print(XtX_inv_train.shape)
print(Xt_train.shape)
print(y_train.shape)
```

```
(12, 12)
```

```
(12, 799)
(799,)
```

[21]:
```python
temp=np.matmul(XtX_inv_train,Xt_train)
theta=np.matmul(temp,y_train)
print(theta.shape)
```

```
(12,)
```

[22]:
```python
print(theta)
```

```
[ 1.03434328e+01  4.21803857e-03 -1.28736172e+00 -2.81156682e-01
  1.59731779e-02 -1.78832680e+00  3.08338676e-03 -3.04438457e-03
 -6.74416241e+00 -1.82876873e-01  6.77925417e-01  2.98166487e-01]
```

[23]:
```python
y_testpred= np.matmul(X_test,theta)
print(y_test,y_testpred)
```

```
[5. 6. 6. 6. 6. 4. 6. 6. 6. 7. 5. 5. 5. 5. 6. 6. 6. 5. 5. 5. 6. 6. 5. 6.
 4. 5. 6. 5. 6. 6. 7. 6. 5. 6. 6. 6. 6. 5. 5. 6. 6. 5. 7. 5. 5. 7. 5. 6.
 5. 6. 7. 5. 5. 5. 6. 5. 6. 6. 5. 5. 5. 6. 7. 6. 5. 5. 6. 7. 6. 5. 5. 6.
 5. 5. 5. 7. 6. 7. 4. 5. 7. 5. 6. 5. 5. 6. 5. 5. 5. 5. 5. 7. 5. 6. 4. 6.
 6. 5. 6. 5. 5. 4. 6. 6. 8. 7. 5. 5. 6. 5. 5. 8. 6. 7. 6. 8. 5. 3. 7. 7.
 5. 5. 5. 5. 7. 4. 6. 6. 6. 4. 5. 5. 6. 7. 7. 6. 5. 5. 4. 5. 5. 5. 6. 6.
 5. 6. 5. 6. 6. 5. 6. 6. 7. 5. 5. 5. 5. 5. 3. 5. 7. 5. 6. 5. 7. 6. 6. 5.
 6. 6. 7. 5. 5. 5. 6. 6. 6. 5. 5. 5. 8. 5. 5. 7. 6. 5. 5. 5. 6. 6. 6. 5.
 6. 6. 6. 7. 7. 6. 5. 5. 6. 6. 5. 6. 5. 5. 6. 7. 6. 6. 5. 5. 5. 5. 5. 5.
 6. 7. 6. 6. 5. 5. 5. 6. 7. 6. 5. 5. 5. 6. 7. 6. 5. 6. 5. 6. 7. 5. 5. 6.
 6. 7. 5. 5. 5. 5. 6. 6. 6. 6. 4. 7. 7. 5. 5. 6. 6. 6. 6. 5. 5. 7. 5. 5.
 6. 6. 7. 8. 5. 6. 6. 7. 5. 8. 5. 5. 6. 5. 7. 6. 5. 4. 6. 5. 5. 6. 7. 6.
 5. 6. 7. 6. 6. 5. 7. 6. 6. 5. 5. 6. 4. 6. 6. 6. 5. 5. 6. 3. 5. 5. 7. 6.
 6. 7. 7. 5. 5. 6. 5. 5.] [5.05866571 6.16780205 5.40540303 5.5587164
6.34309566 5.47243099
 6.23112219 6.22778397 5.22632697 6.56664723 5.47137815 5.04981167
 5.08841755 5.34502665 5.66507287 5.85336438 5.13460974 6.03788258
 6.01231499 4.73760647 5.20884616 6.12424315 4.54138508 5.3465968
 5.93024333 5.29598383 6.18249022 5.2839856  6.27806872 6.48103965
 5.85689079 6.18981921 5.24467345 5.15203381 6.25440699 6.47163912
 5.43028685 5.29535933 5.4096561  6.32866565 5.74302167 5.67304882
 5.87650739 5.06381911 6.33594275 6.62716078 5.19866548 6.0074666
 5.57699809 5.51470946 6.27992503 5.08161229 5.38434709 4.9997587
 6.07196624 5.21895798 5.27166371 5.0852831  5.53186062 4.98280495
 5.85287489 4.88812305 6.13947496 5.94926338 5.08696223 5.4113101
 5.60106369 6.52870559 5.25935226 5.32264856 5.66278397 5.61009236
 5.9195928  5.3073922  5.66715885 6.1438282  6.04920607 6.08201398
 5.10517716 5.26094303 5.84138975 5.38582386 5.7976847  5.9195928
 5.32273604 5.60790839 5.36451711 5.65408242 5.41395585 5.18241661
 5.08568262 6.23167082 4.98431127 5.63861754 5.15721123 5.96315626
```

```
6.45006723 5.56211563 5.0379615  4.67909326 6.16433077 5.11701904
5.68213639 6.34083887 6.90697326 6.66878237 5.39383094 4.1880468
5.69332989 5.41699169 5.52291283 6.50088403 5.77369332 6.04579926
5.64288512 6.89487312 5.22974473 5.18753644 5.99815584 6.1438282
5.22200601 5.2837158  5.05262825 5.75308989 6.06925539 5.169148
4.95427843 6.16481292 5.11321693 5.1993043  4.91940753 5.14273371
6.43544309 5.57679563 6.32284723 5.28030362 5.51660094 5.3073922
5.36682984 4.84813203 5.05351499 5.23015223 5.73420014 5.91892274
5.14896726 6.06574572 5.18369058 6.19403055 5.16439175 4.94457414
5.9721172  5.534383   5.41418453 5.78939946 4.96553111 4.76184054
5.02498997 5.39205961 4.29031156 5.31762547 5.85616509 5.74071087
6.16481292 5.98294499 6.22206269 6.56580077 6.03805248 5.37558593
6.44054065 5.4910414  6.68734423 5.38015597 6.33379937 6.06708788
6.55385922 5.7886859  4.95537603 5.2279443  5.2572634  5.2217564
6.16343599 5.10186101 4.94754686 6.67385062 5.32206745 5.67735891
5.09802509 5.58827992 5.14779658 5.79139739 6.41373976 5.3094656
5.38159965 5.849234   6.50770047 6.0410544  6.54397238 6.40373348
5.59363992 5.48523671 5.85668851 5.56849689 5.33220126 5.20976546
5.1964433  5.71679951 6.54524835 6.69908776 5.71090726 5.32267735
5.26881523 5.12683181 5.44940684 5.66805004 5.34886877 5.20740738
5.78559111 6.58518276 5.05658731 5.25924647 5.34318731 5.07197065
5.20738301 5.56165485 5.878091   6.05324232 5.07861559 5.1247638
5.49410713 5.94854628 6.47195298 6.59120605 5.19923269 5.4316484
5.81061245 5.12564497 5.9857701  5.02345741 5.11722292 6.25440699
5.39387465 6.18451178 5.33970195 5.33296715 5.28761188 4.76263204
6.29456807 5.70239947 6.02626143 7.03348091 5.49650457 5.77047795
6.51579649 5.73465739 5.67945232 5.20069553 5.39124599 5.2473494
6.67192742 5.39504536 4.93592009 6.68734423 5.14749707 5.25538739
5.48817315 5.60184627 5.85678724 6.52412433 5.16168764 5.31946323
5.73330713 5.96282129 5.25183403 6.57923107 5.70356959 5.54830594
5.12437265 5.08140482 6.00343281 5.68791321 5.4377205  5.16747436
4.95537603 5.21196137 5.10314296 6.02348341 6.29944255 5.70489069
4.95271918 6.57720474 5.61318076 5.61699339 6.23112219 5.40454309
5.89682178 5.42788502 5.77689781 5.43630053 5.98294499 6.34760929
5.28134244 5.87951821 5.11040479 6.1264466  5.79057013 5.10487856
6.32944717 5.28635644 5.25880501 5.16053471 6.13387606 5.2936595
6.05569245 6.13042551 6.0237445  5.40720179 5.22200601 6.33441606
5.40834416 5.14910353]
```

[24]:
```
def print_comparision(y_test,y_testpred):
    for i in range(y_test.shape[0]):
        print(f"Actual value:{y_test[i]}, Prediction: {y_testpred[i]}")
        print("\n")
```

[25]:
```
def calculate_rmse(y_actual, y_pred):
    squared_errors = (y_actual - y_pred) ** 2
    mean_squared_error = np.mean(squared_errors)
```

```
    rmse = np.sqrt(mean_squared_error)
    return rmse

def calculate_r_squared(y_actual, y_pred):
    total_variance = np.sum((y_actual - np.mean(y_actual)) ** 2)
    explained_variance = np.sum((y_pred - y_actual) ** 2)
    r_squared = 1 - (explained_variance / total_variance)
    return r_squared
```

[26]:
```
# from sklearn.metrics import mean_squared_error, r2_score
rmse= calculate_rmse(y_test,y_testpred)
r_squared=calculate_r_squared(y_test,y_testpred)
print(f"mean squared error:{rmse**2}")
print(f"rmse:{rmse}")
print(f"r_squared:{r_squared}")
```

```
mean squared error:0.4431719294227915
rmse:0.6657115962808455
r_squared:0.3967805083955571
```

2 Gradient Descent Solution

Here I am considering theta as w just for the sake of conveneince

[27]:
```
X_train
```

[27]:
```
array([[ 1.  ,  8.6 ,  0.52, …,  3.2 ,  0.52,  9.4 ],
       [ 1.  ,  7.8 ,  0.56, …,  3.19,  0.93,  9.5 ],
       [ 1.  ,  9.8 ,  0.5 , …,  3.24,  0.6 ,  9.7 ],
       …,
       [ 1.  ,  7.2 ,  0.62, …,  3.51,  0.54,  9.5 ],
       [ 1.  ,  7.9 ,  0.2 , …,  3.32,  0.8 , 11.9 ],
       [ 1.  ,  5.8 ,  0.29, …,  3.39,  0.54, 13.5 ]])
```

[28]:
```
X_train.shape
```

[28]:
```
(799, 12)
```

[29]:
```
y_train.shape
```

[29]:
```
(799,)
```

[30]:
```
def mean_squared(y_train,y_pred):
    cost=0
    m= y_train.shape[0]
    for i in range(m):
        cost += (y_train[i]-y_pred[i])**2
```

```
    error = cost/(2*m)
    return error
```

[31]:
```
def gradient_descent(X,y,X_val,y_val,w,alpha,train_losses,val_losses):
        m=X.shape[0]
        n=X.shape[1]
        pred=np.dot(X,w)
        error=pred-y
        gradient= np.dot(X.T,error)/m
        w-=alpha*gradient

        val_pred=np.dot(X_val,w)
        # print("the feature constants are",w)
        # print("\n")
        # y_pred= np.matmul(X,w)
        loss=mean_squared(y,pred)
        # y_valpred= np.matmul(X_val,w)
        loss_val= mean_squared(y_val,val_pred)
        train_losses.append(loss)
        val_losses.append(loss_val)
        return w
```

[32]:
```
x=np.array([[0,1,2,3,4,5,5,6],
            [2,4,4,5,6,6,7,4]])
mult=np.array([0,1,2,3,4,5,6,7])
print(x.shape)
print(mult.shape)
print(np.dot(x,mult).shape)
print(x.T)
```

```
(2, 8)
(8,)
(2,)
[[0 2]
 [1 4]
 [2 4]
 [3 5]
 [4 6]
 [5 6]
 [5 7]
 [6 4]]
```

[33]:
```
def linear_regression(X,y,X_val,y_val,w,alpha,steps):
    train_losses=[]
    val_losses=[]
    for iter in range(steps):
        w=gradient_descent(X,y,X_val,y_val,w,alpha,train_losses,val_losses)
```

```
        y_valpred=np.dot(X_val,w)
    return train_losses,val_losses
```

[34]:
```
y_val.shape
```

[34]: (480,)

[35]:
```
def normalize_matrix(X):
    columns_to_normalize = range(1, X.shape[1])
    first_column = X[:, 0]
    normalized_columns = (X[:, columns_to_normalize] - X[:,␣
↪columns_to_normalize].mean(axis=0)) / X[:, columns_to_normalize].std(axis=0)
    X_scaled = np.column_stack((first_column, normalized_columns))

    print("Original Matrix:")
    print(X)

    print("\nNormalized Matrix:")
    print(X_scaled)
    return X_scaled
X_train_scaled=normalize_matrix(X_train)
```

```
Original Matrix:
[[ 1.    8.6   0.52 … 3.2   0.52  9.4 ]
 [ 1.    7.8   0.56 … 3.19  0.93  9.5 ]
 [ 1.    9.8   0.5  … 3.24  0.6   9.7 ]
 …
 [ 1.    7.2   0.62 … 3.51  0.54  9.5 ]
 [ 1.    7.9   0.2  … 3.32  0.8  11.9 ]
 [ 1.    5.8   0.29 … 3.39  0.54 13.5 ]]

Normalized Matrix:
[[ 1.          0.21467856 -0.07076412 … -0.77358368 -0.81095735
  -0.97734658]
 [ 1.         -0.26952234  0.15919229 … -0.84003104  1.54211189
  -0.88353352]
 [ 1.          0.94097992 -0.18574232 … -0.50779423 -0.35182189
  -0.6959074 ]
 …
 [ 1.         -0.63267302  0.5041269  …  1.28628454 -0.69617348
  -0.88353352]
 [ 1.         -0.20899723 -1.91041539 …  0.02378466  0.79601677
   1.36797994]
 [ 1.         -1.48002461 -1.39301347 …  0.4889162  -0.69617348
   2.8689889 ]]
```

3 alpha = 0.1

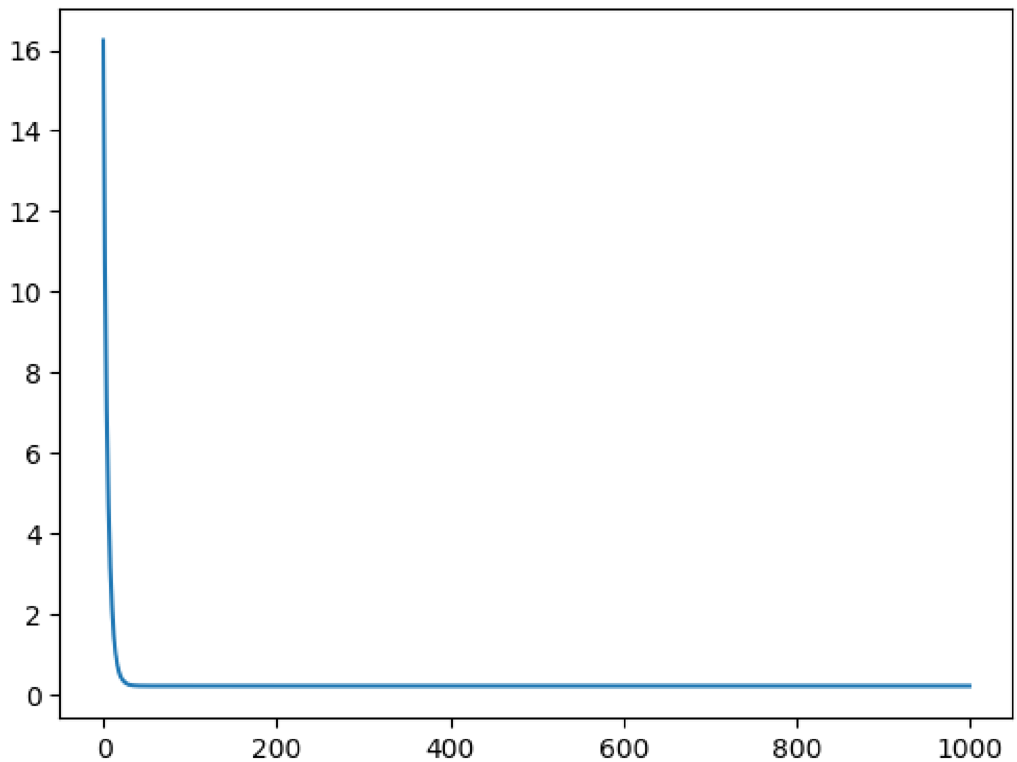
[36]:
```
w=np.zeros((12,))
print(w)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

[37]:
```
train_losses,val_losses=linear_regression(X_train_scaled,y_train,X_val,y_val,w,0.
 ↪1,1000)
```

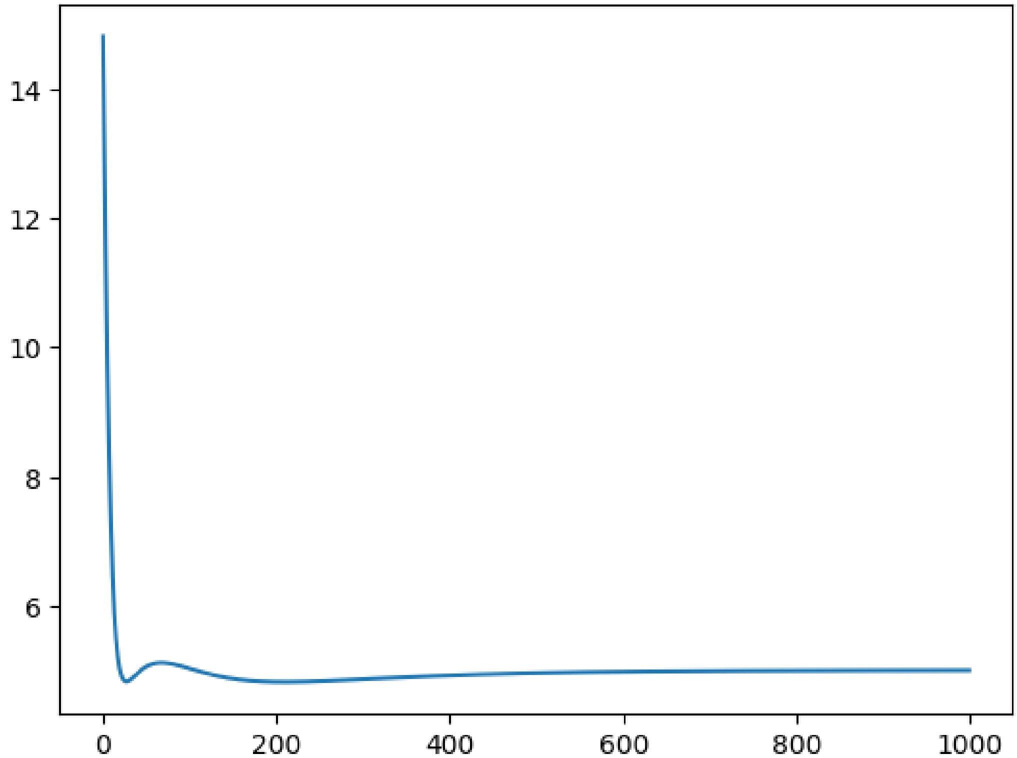
[38]:
```
plt.plot(train_losses)
```

[38]:
```
[<matplotlib.lines.Line2D at 0x203dc3c7f10>]
```



[39]:
```
plt.plot(val_losses)
```

[39]:
```
[<matplotlib.lines.Line2D at 0x203de624640>]
```



4 alpha = 0.01

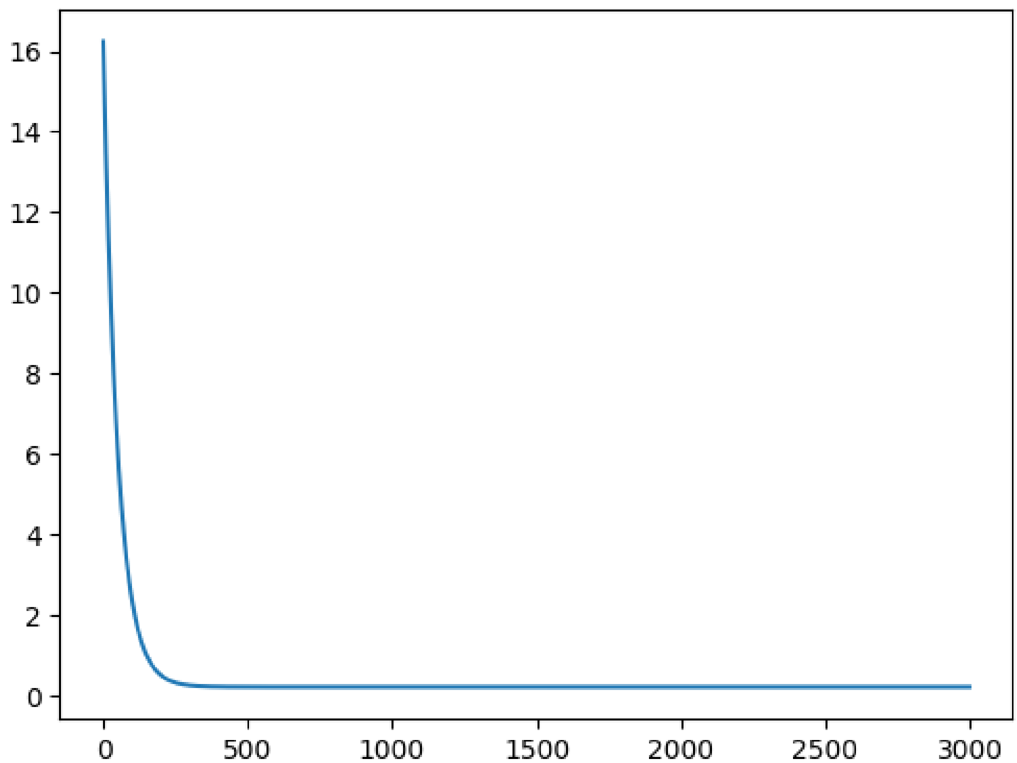
[40]:
```
w=np.zeros((12,))
print(w)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

[41]:
```
train_losses_2,val_losses_2=linear_regression(X_train_scaled,y_train,X_val,y_val,w,0.
    ↪01,3000)
```

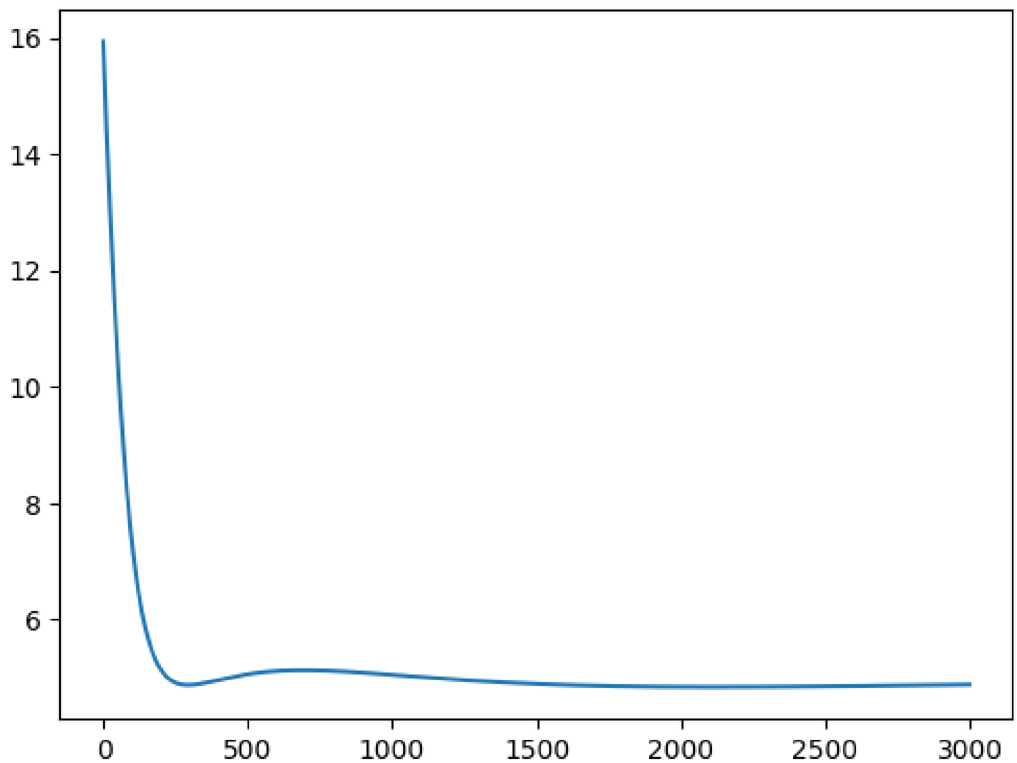
[42]:
```
plt.plot(train_losses_2)
```

[42]:
```
[<matplotlib.lines.Line2D at 0x203de47cfa0>]
```



[43]:
```
plt.plot(val_losses_2)
```

[43]: [<matplotlib.lines.Line2D at 0x203dc402d70>]



5 alpha = 0.001

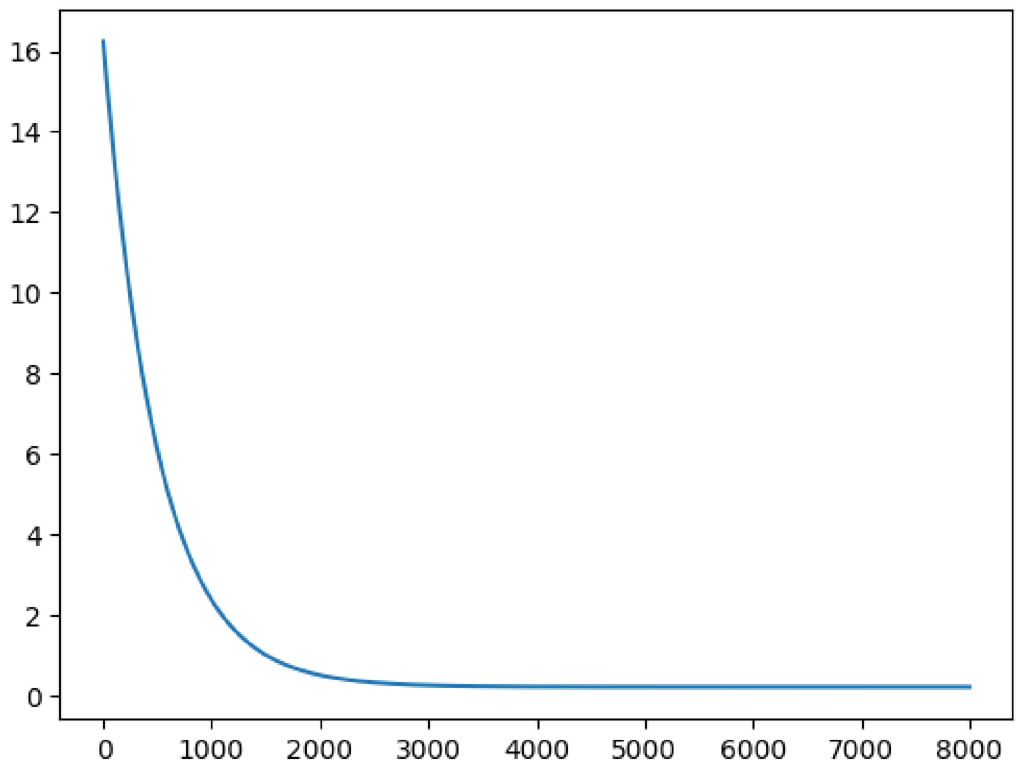
[44]:
```
w=np.zeros((12,))
print(w)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

[45]:
```
train_losses_3,val_losses_3=linear_regression(X_train_scaled,y_train,X_val,y_val,w,0.
↪001,8000)
```

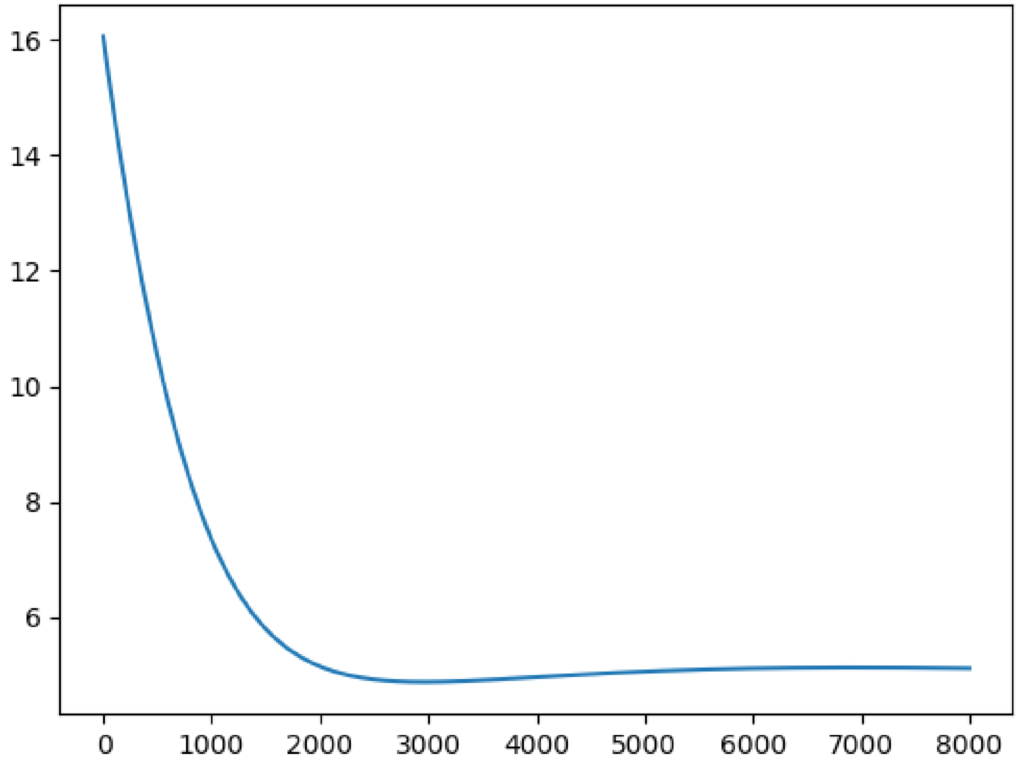
[46]:
```
plt.plot(train_losses_3)
```

[46]: [<matplotlib.lines.Line2D at 0x203de52beb0>]



[47]:
```
plt.plot(val_losses_3)
```

[47]: [<matplotlib.lines.Line2D at 0x203df7c7070>]



[48]:
```
X_test.shape
```

[48]: (320, 12)

[49]:
```
X_test_scaled=normalize_matrix(X_test)

y_testpred=np.dot(X_test_scaled,w)
print(X_test_scaled.shape)
print(w.shape)
y_testpred.shape
```

```
Original Matrix:
[[ 1.    7.9   0.72 …  3.4   0.53  9.5 ]
 [ 1.   10.6   0.31 …  3.26  0.86 10.7 ]
 [ 1.    7.3   0.39 …  3.41  0.54  9.4 ]
 …
 [ 1.   10.    0.35 …  3.23  0.52 12.  ]
 [ 1.    9.9   0.35 …  3.21  0.5   9.5 ]
 [ 1.    7.4   0.66 …  3.51  0.56  9.4 ]]

Normalized Matrix:
[[ 1.         -0.25560867  1.02373839 …  0.5778992  -0.82038121
```

```
  -0.80689022]
 [ 1.          1.19636957 -1.08398192 … -0.30752775  1.42863893
   0.28546248]
 [ 1.         -0.5782705  -0.67271942 …  0.64114398 -0.75222908
  -0.89791961]
 …
 [ 1.          0.87370774 -0.87835067 … -0.4972621  -0.88853333
   1.46884458]
 [ 1.          0.81993077 -0.87835067 … -0.62375167 -1.02483758
  -0.80689022]
 [ 1.         -0.52449353  0.71529151 …  1.27359181 -0.61592483
  -0.89791961]]
(320, 12)
(12,)
```

[49]:
```
(320,)
```

[50]:
```
X_test
```

[50]:
```
array([[ 1.  ,  7.9 ,  0.72, …,  3.4 ,  0.53,  9.5 ],
       [ 1.  , 10.6 ,  0.31, …,  3.26,  0.86, 10.7 ],
       [ 1.  ,  7.3 ,  0.39, …,  3.41,  0.54,  9.4 ],
       …,
       [ 1.  , 10.  ,  0.35, …,  3.23,  0.52, 12.  ],
       [ 1.  ,  9.9 ,  0.35, …,  3.21,  0.5 ,  9.5 ],
       [ 1.  ,  7.4 ,  0.66, …,  3.51,  0.56,  9.4 ]])
```

[51]:
```
print(f"r_squared: {calculate_r_squared(y_test,y_testpred)}")
```

```
r_squared: 0.3991593477424942
```

[52]:
```
print(f"rmse: {calculate_rmse(y_test,y_testpred)}")
```

```
rmse: 0.6643976588767766
```

[53]:
```
print(f"mse:{calculate_rmse(y_test,y_testpred)**2}")
```

```
mse:0.4414242491209416
```