boston

In [ ]:

```
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [2]:

```
boston_data=load_boston()
df=pd.DataFrame(boston_data.data,columns=boston_data.feature_names)
df
```

```
C:\Users\CSE WPT\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:8
7: FutureWarning: Function load_boston is deprecated; `load_boston` is depre
cated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer
to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of th
is
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9(
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9(
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.8:
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.6:
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.9(
...	...	...	...	...	...	...	...	...	...	...	...	..
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.9!

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.9
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.4
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.9

In [25]:

```
df['PRICE']=boston_data.target
x=df.drop('PRICE',axis=1)
y=df['PRICE']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=5)
df
```

Out[25]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	24.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	21.6	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	34.7	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	33.4	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	36.2	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90
...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	22.4	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99
502	0.04527	0.0	11.93	20.6	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90
503	0.06076	0.0	11.93	23.9	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90
504	0.10959	0.0	11.93	22.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45
505	0.04741	0.0	11.93	11.9	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90

506 rows × 14 columns

In [4]:

```
model=LinearRegression()
```

In [5]:

```
model.fit(x_train,y_train)
```

Out[5]:

```
LinearRegression()
```

In [6]:

```
prediction=model.predict(x_test)
prediction
```

Out[6]:

```
array([37.56311787, 32.14445143, 27.06573629,  5.67080633, 35.09982577,
        5.85803701, 27.53708506, 31.81019188, 26.35634771, 22.77208748,
       31.91183048, 21.50224061, 23.70119983, 33.3622504 , 28.51633591,
       14.39456899,  0.19284025, 18.66247155, 13.71004139, 14.13408635,
        2.03263952, 19.7280831 , 38.18657429, 24.19760058, 31.30247973,
       11.14144544, 25.03636951, 23.27970871, 22.49420127, 20.52972594,
       15.16513744,  6.92553586, 18.3557733 , 22.37179804, 28.91287973,
       19.02980786, 30.19357214,  8.74384915, 40.86691522, 34.53763591,
       20.70224878,  2.59618963, 29.99590282, 12.15704798, 27.10186397,
       30.8052437 , -6.24169079, 19.84885777, 20.92973441, 12.43523958,
       20.4949947 , 19.19231742, 23.69073157, 12.67998473, 17.14252424,
       25.04649176, 34.77758126, 15.23294903, 28.22306193, 21.08745388,
       20.39506129, 25.79476888, 14.72463673, 33.18635032, 23.17771307,
       13.11057248, 19.23154617, 24.61162961, 21.50327036, 22.00419172,
       20.5900874 , 27.19709085, 16.86361523, 18.92610238, 20.62344917,
       25.73255665, 22.03855586, 14.51899949, 34.3918044 , 18.5369776 ,
       23.38945015, 41.36132839, 23.27134886, 15.62340913, 25.69729854,
       17.16406313, 18.5066679 , 10.04976469, 18.99779955, 17.02528993,
       35.707325  , 17.50855206, 22.16184894, 19.26215663, 24.16777784,
       27.80472748, 12.42828948, 21.91295599, 22.39477399, 13.19335364,
       23.96991103, 21.19914699])
```

In [20]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error,r2_score
import numpy as np
print("MAE:",mean_absolute_error(y_test,prediction))
print("MSE:",mean_squared_error(y_test,prediction))
print("RMSE:",np.sqrt(mean_absolute_error(y_test,prediction)))
print("r2_score:",r2_score(y_test,prediction))
```

```
MAE: 3.213270495842398
MSE: 20.86929218377099
RMSE: 1.7925597607450632
r2_score: 0.7334492147453053
```

In [21]:

```
from sklearn.metrics import accuracy_score
print("Accuracy",model.score(x_test,y_test))
```

```
Accuracy 0.7334492147453053
```

iris

In [21]:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [22]:

```
iris_data=load_iris()
df=pd.DataFrame(iris_data.data,columns=iris_data.feature_names)
df
```

Out[22]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [23]:

```

x=iris_data['data']
y=iris_data['target']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=5)
```

In [24]:

```
model=LinearRegression()
```

In [25]:

```
model.fit(x_train,y_train)
```

Out[25]:

```
LinearRegression()
```

In [26]:

```
prediction=model.predict(x_test)
prediction
```

Out[26]:

```
array([ 1.04528614,  1.50372236,  2.14140386,  0.01886528,  2.19888523,
        0.9240494 , -0.04321461,  1.64367392, -0.01635344,  1.45201555,
        1.4187976 ,  1.43217512,  1.89140987,  1.73601921,  0.12433869,
        0.19865573,  1.59704788,  1.74513104,  0.08714977, -0.09618162,
        1.0871553 ,  1.94169741, -0.08522015,  1.38754985,  1.19270534,
        2.07290743,  1.10291863,  1.19744639,  1.30926184,  2.01127339])
```

In [27]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error,r2_score
import numpy as np
print("MAE:",mean_absolute_error(y_test,prediction))
print("MSE:",mean_squared_error(y_test,prediction))
print("RMSE:",np.sqrt(mean_absolute_error(y_test,prediction)))
print("r2_score:",r2_score(y_test,prediction))
```

```
MAE: 0.20533358254335438
MSE: 0.07231433152478547
RMSE: 0.45313748746197813
r2_score: 0.8839877034361731
```

In [28]:

```
from sklearn.metrics import accuracy_score
print("Accuracy",model.score(x_test,y_test))
```

```
Accuracy 0.8839877034361731
```