

```
import numpy as np
import pandas as pd
import sklearn
```

```
from sklearn.datasets import load_boston
df=load_boston()
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning

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 this 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)
```

```
df.keys()
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename',
'data_module'])
```

```
print(df.DESCR)
```

```
.. _boston_dataset:
```

Boston house prices dataset

****Data Set Characteristics:****

:Number of Instances: 506

:Number of Attributes: 13 numeric/categorical predictive. Median Value (att

:Attribute Information (in order):

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (= 1 if tract bounds river; 0 o
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five Boston employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per \$10,000
- PTRATIO pupil-teacher ratio by town
- B $1000(B_k - 0.63)^2$ where B_k is the proportion of black people
- LSTAT % lower status of the population
- MEDV Median value of owner-occupied homes in \$1000's

:Missing Attribute Values: None

:Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.

<https://archive.ics.uci.edu/ml/machine-learning-databases/housing/>

This dataset was taken from the StatLib library which is maintained at Carnegie

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that problems.

.. topic:: References

- Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Da
- Quinlan, R. (1993). Combining Instance-Based and Model-Based Learning. In P

```
print(df.data)# information about data
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]]
```

```
[2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
...
[6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
[1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
[4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]
```

```
boston = pd.DataFrame(df.data, columns=df.feature_names)
boston.head()
```

	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	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90



```
boston['MEDV']=df.target
boston.head()
```

	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	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90



```
boston.isnull()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False
...
501	False	False	False	False	False	False	False	False	False	False	False	False	False
502	False	False	False	False	False	False	False	False	False	False	False	False	False
503	False	False	False	False	False	False	False	False	False	False	False	False	False

```
boston.isnull().sum()
```

```
CRIM      0
ZN        0
INDUS     0
CHAS      0
NOX       0
RM        0
AGE       0
DIS       0
RAD       0
TAX       0
PTRATIO   0
B         0
LSTAT     0
MEDV      0
dtype: int64
```

```
from sklearn.model_selection import train_test_split
X=boston.drop('MEDV',axis=1)
Y=boston['MEDV']
```

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.15,random_state=5)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(430, 13)
(76, 13)
(430,)
(76,)
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
lin_model=LinearRegression()  
lin_model.fit(X_train,Y_train)  
  
    LinearRegression()  
  
y_train_predict=lin_model.predict(X_train)  
rmse=(np.sqrt(mean_squared_error(Y_train,y_train_predict)))  
  
print("The model performance for training set")  
print('RMSE is{}'.format(rmse))  
print("\n")  
  
y_test_predict=lin_model.predict(X_test)  
rmse=(np.sqrt(mean_squared_error(Y_test,y_test_predict)))  
  
print("The model for testing set")  
print('RMSE is{}'.format(rmse))  
  
    The model performance for training set  
    RMSE is4.710901797319796  
  
    The model for testing set  
    RMSE is4.687543527902972
```