Практическая работа 1. Линейная регрессия для решения задачи предсказания цены на дома Бостона

На этом практическом занятии мы решаем одну из классических задач машинного обучения -- предсказания цены на дома в Бостоне. Набор данных Boston Housing содержит информацию о 506 домах Бостона: их характеристики (количество комнат, удаленность от центра, уровень преступности и т.д.) и цену.

Наша цель -- построить линейную модель, которая будет по характеристикам дома предсказывать его стоимость.

Сам набор данных доступен по адресу https://archive.ics.uci.edu/ml/datasets/Housing. Однако, поскольку мы собираемся использовать scikit-learn, мы можем загрузить его прямо из самого scikit-learn.

Загрузка данных

Сначала мы импортируем необходимые библиотеки, загрузим сам датасет, посмотрим на него. Далее создадим объект pandas -- таблицу с данными, с которой будем дальше работать.

In [ ]:
```
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
```

Импортируем также все, что нам понадобится из библиотеки scikit-learn.

In [ ]:
```
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Загрузим датасет Boston Housing.

In [ ]:
```
boston = datasets.load_boston()
```

Как видим ниже, набор исходных данных boston.data содержит 506 строк и 13 столбцов. Переменная boston.target пердставляет собой 506 мерный вектор ответов (цены на дома).

In [ ]:
```
print(boston.data.shape)
```

```
(506, 13)
```

In [ ]:
```
print(boston.target.shape)
```

```
(506,)
```

Посмотрим на полное описание набора данных и 13 признаков.

In [ ]:
```
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------
```

```
**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is us
ually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - 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(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - 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 Mellon Universi
ty.

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 address regressi
on
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources o
f Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on th
e Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amh
erst. Morgan Kaufmann.
```

Создадим таблицу bos с помощью пакета pandas, с которой будет удобнее работать дальше.

Посмотрим на ее первые 5 строк.

In [ ]:
```
bos = pd.DataFrame(boston.data, columns = boston.feature_names)
print(bos.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX  ...  RAD    TAX  PTRATIO       B  LSTAT
0  0.00632  18.0   2.31   0.0  0.538  ...  1.0  296.0     15.3  396.90   4.98
1  0.02731   0.0   7.07   0.0  0.469  ...  2.0  242.0     17.8  396.90   9.14
2  0.02729   0.0   7.07   0.0  0.469  ...  2.0  242.0     17.8  392.83   4.03
3  0.03237   0.0   2.18   0.0  0.458  ...  3.0  222.0     18.7  394.63   2.94
4  0.06905   0.0   2.18   0.0  0.458  ...  3.0  222.0     18.7  396.90   5.33

[5 rows x 13 columns]
```

Добавим к ней 14-ый столбец с ценой дома.

In [ ]:
```
bos['PRICE'] = boston.target
print(bos.head())
```

```
      CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  PRICE
0  0.00632  18.0   2.31   0.0  0.538  ...  296.0     15.3  396.90   4.98   24.0
1  0.02731   0.0   7.07   0.0  0.469  ...  242.0     17.8  396.90   9.14   21.6
2  0.02729   0.0   7.07   0.0  0.469  ...  242.0     17.8  392.83   4.03   34.7
3  0.03237   0.0   2.18   0.0  0.458  ...  222.0     18.7  394.63   2.94   33.4
4  0.06905   0.0   2.18   0.0  0.458  ...  222.0     18.7  396.90   5.33   36.2

[5 rows x 14 columns]
```

Разведочный анализ данных

После загрузки данных стоит понять, с чем нам предстоит иметь дело. Мы посмотрим, нет ли в данных пропущенных значений, сильно коррелированных признаков. Мы выясним, что между признаком **RM** (количество комнат) и ценой есть хорошая корреляция и есть смысл решить задачу одномерной линейной регрессии между ними.

Для начала проверим, нет ли в них пропущенных значений. Сделаем это для каждого столбца с помощью функции isnull().

In [ ]:
```
bos.isnull().sum()
```

Out[ ]:
```
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
PRICE      0
dtype: int64
```

Пропусков нет.

Посмотрим теперь на информацию о некоторых статистических характеристиках каждого признака.

In [ ]:
```
bos.describe()
```

Out[ ]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.54940
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.70725
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.00000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.00000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.00000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.00000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.00000

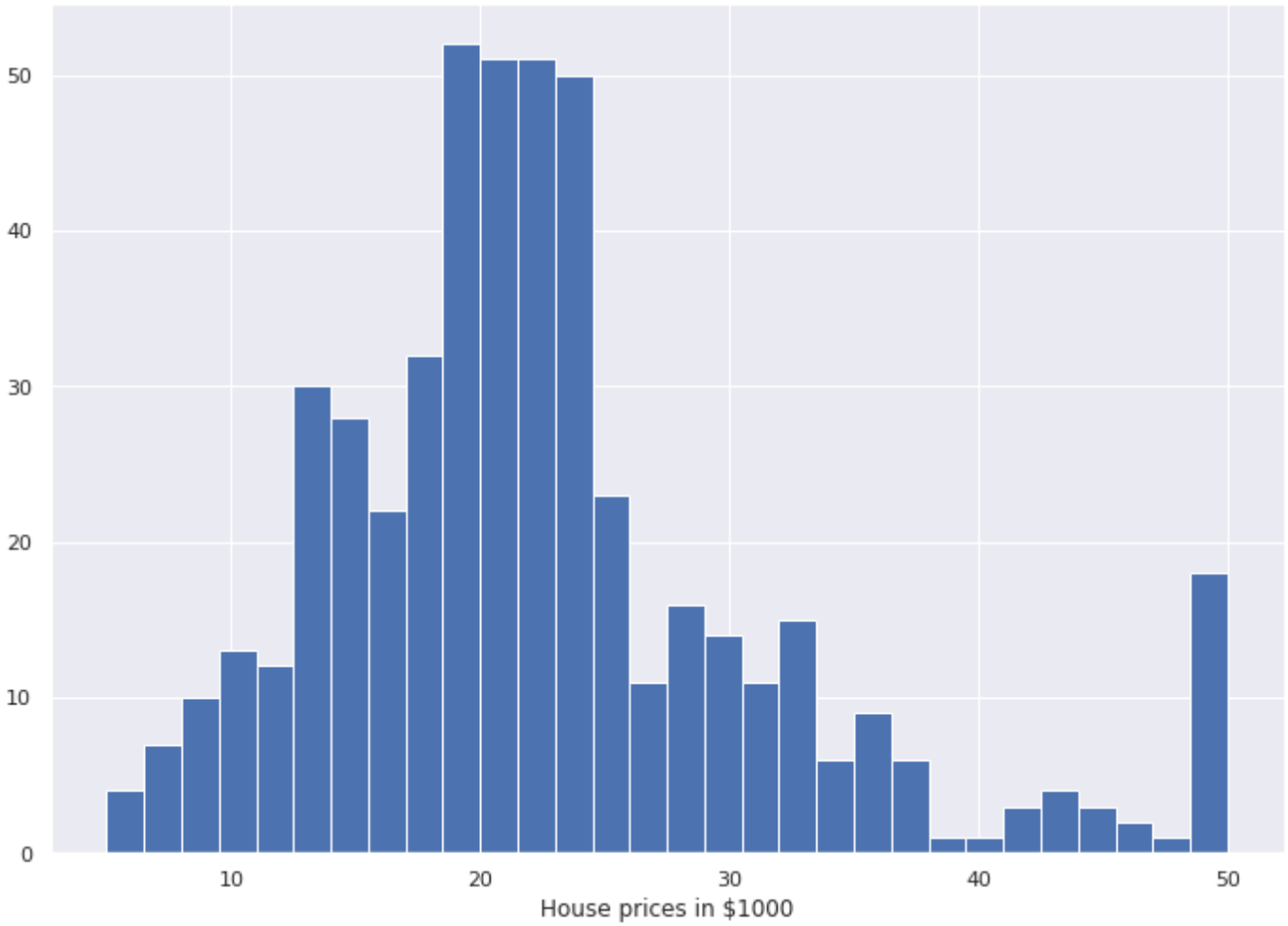
count = 506 говорит о том, что у нас есть по 506 значений в каждом столбце. Можно видеть также среднее значение mean и среднеквадратичное отклонение std каждого признака, их минимальное и

максимальное значения. 25% = 0.08 значит, что 25% ваших данных имеют значение этого признака 0.08 и менее.

Как можно заметить, признаки довольно сильно отличаются по масштабу, имеют разный диапазон изменений. Для дальнейшего обучения модели линейной регрессии на этих данных желательна их нормализация. Метод градиентного спуска будет лучше работать с данными одного масштаба. Заметим однако, что в sklearn реализован метод нормального уравнения для решения задачи линейной регрессии, которому нормализация не важна. Мы, тем не менее, сделаем нормализацию позже, в учебных целях.

Построим теперь гистограмму распределения целевой переменной **Price**. Мы будем использовать функцию distplot из библиотеки seaborn.

In [ ]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
plt.hist(bos['PRICE'], bins=30)
plt.xlabel("House prices in $1000")
plt.show()
```



Из графика видно, что значения **Price** распределены нормально с небольшими выбросами. Большинство цен домов находятся в диапазоне 20-24 (в масштабе 1000 долларов).

Построим теперь матрицу корреляций с помощью функции corr из библиотеки pandas.

In [ ]:
```
correlation_matrix = bos.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f98d11613d0>
```

Коэффициент корреляции варьируется от -1 до 1. Если коэффициент корреляции близок к 1, то между переменными наблюдается положительная корреляция. Иными словами, отмечается высокая степень связи между переменными. В данном случае, если значения переменной x будут возрастать, то и выходная переменная y также будет увеличиваться. Если коэффициент корреляции близок к -1, это означает, что между переменными имеет место сильная отрицательная корреляция. Иными словами, поведение выходной переменной будет противоположным поведению входной. Если значение x будет возрастать, то y будет уменьшаться, и наоборот. Промежуточные значения, близкие к 0, будут указывать на слабую корреляцию между переменными и, соответственно, низкую зависимость.

1. Посмотрев на матрицу корреляции, мы можем увидеть, что признак **RM** имеет сильную положительную корреляцию с признаком **Price**, равную **0,7**, а **LSTAT** имеет сильную отрицательную корреляцию с **Price**, равную **- 0,74**.

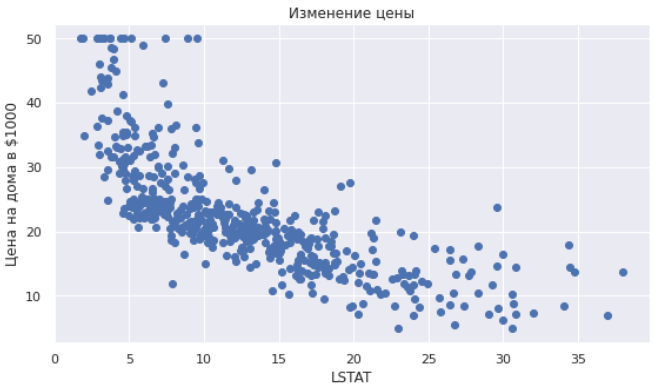
2. Также признаки **RAD**, **TAX** имеют сильную корреляцию **0,91**. То же самое можно сказать о признаках **DIS** и **AGE**, которые имеют корреляцию **-0,75**.

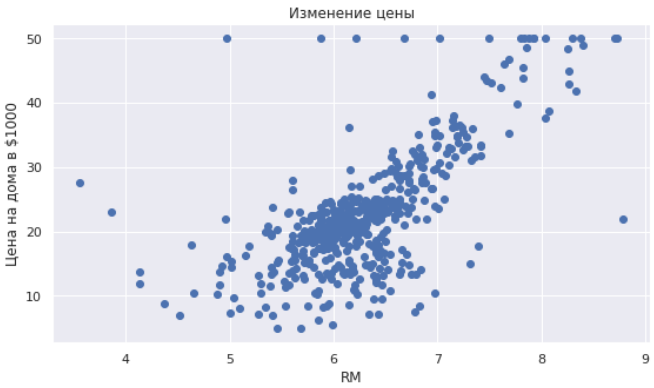
Построим графики зависимости цены от признаков **LSTAT**, **RM**.

In [ ]:
```
plt.figure(figsize=(20, 5))

features = ['LSTAT', 'RM']
target = bos['PRICE']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = bos[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title("Изменение цены")
    plt.xlabel(col)
    plt.ylabel("Цена на дома в $1000")
```





Мы действительно видим зависимость между ценой и этими признаками. Зависимость между **RM** и **Price** похожа на линейную. Будем далее искать ее.

Решение задачи одномерной линейной регрессии

В этой части мы найдем линейную зависимость между **RM** и **Price**. Мы увидим, как правильно разделить данные на обучающий и тестовый набор.

Решим задачу одномерной линейной регрессии. Найдем линейную зависимость между **RM** и **Price**, которая как мы видели, есть. Создадим входной и выходной 506-мерный вектор для нашей линейной модели.

In [ ]:
```
X_rooms = bos.RM
y_price = bos.PRICE

X_rooms = np.array(X_rooms).reshape(-1,1)
y_price = np.array(y_price).reshape(-1,1)

print(X_rooms.shape)
print(y_price.shape)
```

```
(506, 1)
(506, 1)
```

Разделение данных на обучающий и тестовый наборы

Для того, чтобы иметь возможность протестировать нашу модель, мы разделим данные на обучающие и тестовые. Мы обучаем модель на 80% образцов и тестируем на оставшихся 20%. Мы делаем это, чтобы оценить эффективность модели на новых данных, не участвовавших в процессе обучения.

Для разделения данных мы используем функцию train_test_split, предоставленную библиотекой scikit-learn.

In [ ]:
```
X_train_1, X_test_1, Y_train_1, Y_test_1 = train_test_split(X_rooms, y_price, test_size = 0.2,
```

In [ ]:
```
print(X_train_1.shape)
print(X_test_1.shape)
print(Y_train_1.shape)
print(Y_test_1.shape)
```

```
(404, 1)
(102, 1)
(404, 1)
(102, 1)
```

Как мы видим, в обучающей выборке у нас 404 образца, в тестовой 102.

Обучение и тестирование модели

Будем использовать функцию LinearRegression из scikit-learn, чтобы обучить нашу модель линейной регрессии на обучающем наборе данных.

In [ ]:
```
reg_1 = LinearRegression()
reg_1.fit(X_train_1, Y_train_1)
```

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

Оценим эффективность нашей модели на тренировочном наборе данных.

In [ ]:
```
y_train_predict_1 = reg_1.predict(X_train_1)
rmse = (np.sqrt(mean_squared_error(Y_train_1, y_train_predict_1)))
r2 = round(reg_1.score(X_train_1, Y_train_1),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 6.557180458295626
R2 score is 0.51
```

И на тестовом наборе.

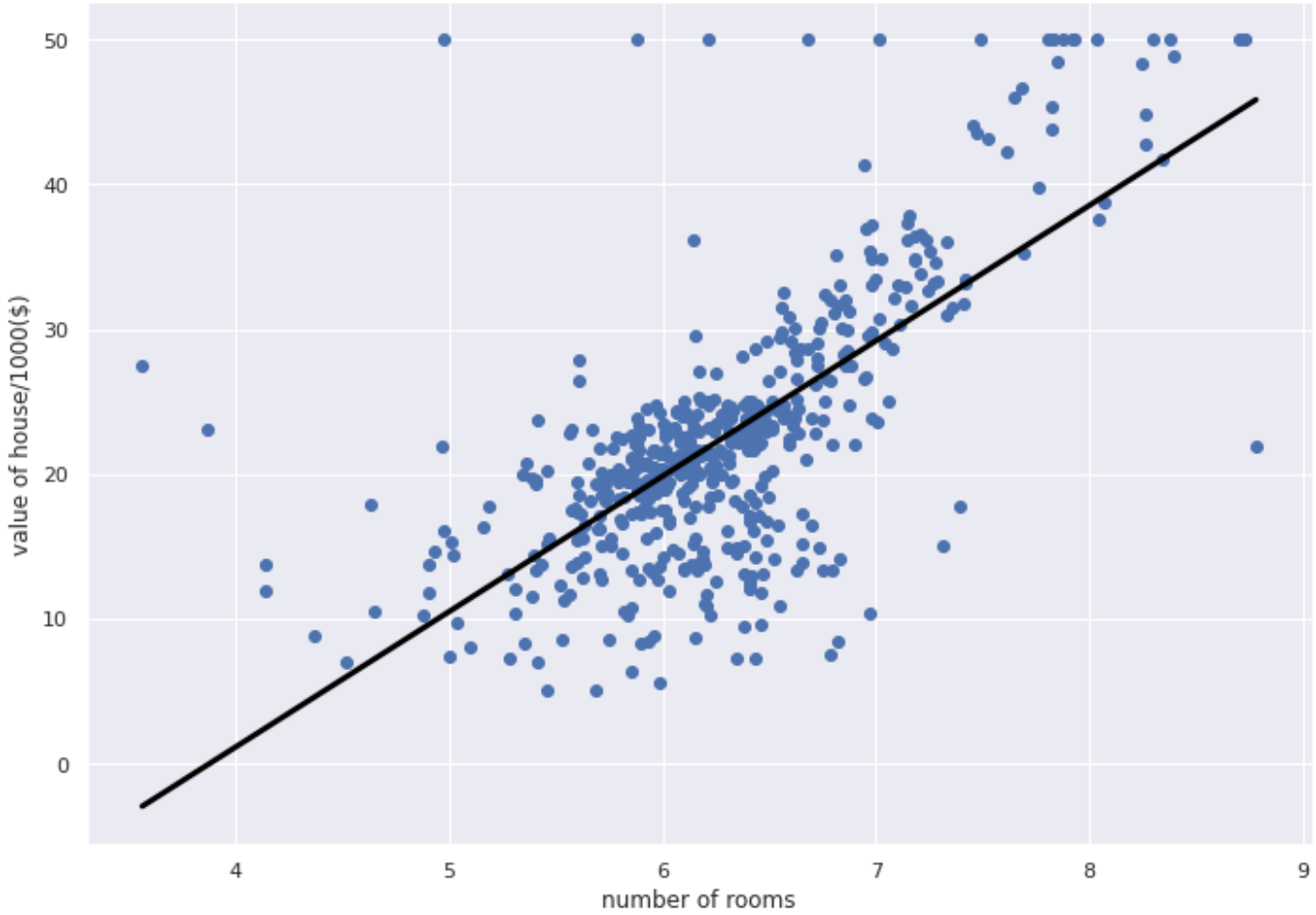
In [ ]:
```
y_pred_1 = reg_1.predict(X_test_1)
rmse = (np.sqrt(mean_squared_error(Y_test_1, y_pred_1)))
r2 = round(reg_1.score(X_test_1, Y_test_1),2)

print("The model performance for test set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
```

```
The model performance for test set
--------------------------------------
Root Mean Squared Error: 6.792994578778734
R^2: 0.37
```

Мы уже ранее заметили линейную зависимость между **RM** и **Price** и фактически искали прямую, проходящую через точки наилучшим образом. Построим найденную прямую.

In [ ]:
```
prediction_space = np.linspace(min(X_rooms), max(X_rooms)).reshape(-1,1)
plt.scatter(X_rooms,y_price)
plt.plot(prediction_space, reg_1.predict(prediction_space), color = 'black', linewidth = 3)
plt.ylabel('value of house/1000($)')
plt.xlabel('number of rooms')
plt.show()
```



Решение задачи множественной линейной регрессии

Построим теперь линейную модель, учитывающую зависимость **Price** не от одной переменной **RM**, а от всех 13 признаков.

В этом случае, разумеется, уже невозможна визуализация модели на плоскости.

Прежде всего сформируем вход модели X и выход y.

In [ ]:
```
X = bos.drop('PRICE', axis = 1)
y = bos['PRICE']
```

Также разобъем данные на тренировочную и тестовую выборки.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Нормализация данных

Следующий этап перед обучением модели -- нормализация данных. Как мы уже видели, признаки меняются в разных диапазонах и их нужно привести к одному масштабу.

Приведем переменные к диапазону изменения [0,1], используя MinMaxScaler.

In [ ]:
```
scaler = MinMaxScaler()
scaler.fit(X_train)   # обучаем scaler только на тренировочном датасете
X_train_norm = scaler.transform(X_train)
X_test_norm = scaler.transform(X_test)
```

И построим модель множественной линейной регрессии.

In [ ]:
```
reg_all = LinearRegression()
```

```
reg_all.fit(X_train_norm, y_train)
```

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

Оценим эффективность модели на тренировочном датасете.

In [ ]:
```
y_train_predict = reg_all.predict(X_train_norm)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = round(reg_all.score(X_train_norm, y_train),2)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 4.6520331848801675
R2 score is 0.75
```

На тестовом.

In [ ]:
```
y_pred = reg_all.predict(X_test_norm)
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
r2 = round(reg_all.score(X_test_norm, y_test),2)

print("The model performance for training set")
print("--------------------------------------")
print("Root Mean Squared Error: {}".format(rmse))
print("R^2: {}".format(r2))
```

```
The model performance for training set
--------------------------------------
Root Mean Squared Error: 4.928602182665338
R^2: 0.67
```

Как можно видеть, используя все 13 признаков, а не один, мы добились большей точности модели.

Заключение

Итак, мы познакомились с библиотекой sklearn языка Python для машинного обучения, загрузили набор данных Boston Housing, проанализировали его, разделили данные на тренировочный и тестовый наборы, построили одномерную и множественную модели линейной регрессии и оценили их эффективность.