In [1]:

```
import pandas as pd
```

Загрузим данные

In [2]:

```
import pandas as pd
import numpy as np

# Измененный вариант загрузки т.к. не поддерживается датасет в склеарн
#  по причине "этической проблемы"
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]
```

In [3]:

```
print(raw_df)
```

```
              0      1      2    3      4      5     6       7    8      9   \
0       0.00632  18.00   2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0
1     396.90000   4.98  24.00  NaN    NaN    NaN   NaN     NaN  NaN    NaN
2       0.02731   0.00   7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0
3     396.90000   9.14  21.60  NaN    NaN    NaN   NaN     NaN  NaN    NaN
4       0.02729   0.00   7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0
...         ...    ...    ...  ...    ...    ...   ...     ...  ...    ...
1007  396.90000   5.64  23.90  NaN    NaN    NaN   NaN     NaN  NaN    NaN
1008    0.10959   0.00  11.93  0.0  0.573  6.794  89.3  2.3889  1.0  273.0
1009  393.45000   6.48  22.00  NaN    NaN    NaN   NaN     NaN  NaN    NaN
1010    0.04741   0.00  11.93  0.0  0.573  6.030  80.8  2.5050  1.0  273.0
1011  396.90000   7.88  11.90  NaN    NaN    NaN   NaN     NaN  NaN    NaN

        10
0     15.3
1      NaN
2     17.8
3      NaN
4     17.8
...    ...
1007   NaN
1008  21.0
1009   NaN
1010  21.0
1011   NaN

[1012 rows x 11 columns]
```

In [4]:

```
target
```

Out[4]:

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
```

```
 19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
 20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
 23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
 33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,

 21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
 20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
 23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
 15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
 17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
 25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
 23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
 32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
 34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
 20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
 26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
 31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
 22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
 42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
 36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
 32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
 20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
 20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
 22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
 21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
 19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
 32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
 18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
 16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
 13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
  7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
 12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
 27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
  8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
  9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
 10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
 15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
 19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
 29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
 20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
 23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

In [5]:

```
# Имена колонок для набора данных
column_names = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD",
                "TAX", "PTRATIO", "B", "LSTAT"]

# Преобразование data в DataFrame
df = pd.DataFrame(data, columns=column_names)
df.head()

X, y = df, target
```

In [6]:

```
X.head()
```

Out[6]:

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	9.14
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	4.03
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	2.94
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	5.33

Зафиксируем генератор случайных чисел для воспроизводимости:

In [7]:

```
SEED = 22
np.random.seed = SEED
```

Разделим данные на условно обучающую и отложенную выборки:

In [8]:

```
from sklearn.model_selection import train_test_split
```

In [9]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=SE
ED)
```

In [10]:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[10]:

```
((404, 13), (404,), (102, 13), (102,))
```

Измерять качество будем с помощью метрики среднеквадратичной ошибки:

In [11]:

```
from sklearn.metrics import mean_squared_error
```

Задача 1.

Обучите **LinearRegression** из пакета **sklearn.linear_model** на обучающей выборке (*X_train, y_train*) и измерьте качество на *X_test*.

P.s. Ошибка должна быть в районе 20.

In [12]:

```
from sklearn.linear_model import LinearRegression

# Создаем и обучаем модель линейной регрессии
lr = LinearRegression()
lr.fit(X_train, y_train)

# Предсказываем значения на тестовой выборке
y_pred = lr.predict(X_test)

# Вычисляем среднеквадратичную ошибку
mse = mean_squared_error(y_test, y_pred)
mse
```

Out[12]:

```
20.7706847842701
```

Задача 2.

Обучите **SGDRegressor** из пакета **sklearn.linear_model** на обучающей выборке (*X_train, y_train*) и измерьте качество на *X_test*.

In [13]:

```
from sklearn.linear_model import SGDRegressor

# Создаем и обучаем модель SGDRegressor
sgd = SGDRegressor()
sgd.fit(X_train, y_train)

# Предсказываем значения на тестовой выборке
y_pred_sgd = sgd.predict(X_test)

# Вычисляем среднеквадратичную ошибку
mse_sgd = mean_squared_error(y_test, y_pred_sgd)
mse_sgd
```

Out[13]:

```
4.338084361876963e+28
```

Задача 3.

Попробуйте все остальные классы:

- Ridge
- Lasso
- ElasticNet

В них, как вам уже известно, используются параметры регуляризации **alpha**. Настройте его как с помощью

In [14]:

```
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)
```

Ridge

In [15]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

# Создаем экземпляр модели и параметры для поиска
ridge = Ridge(max_iter=10000)
# Создаем список alpha значений от 1e-20 до 1e0
alphas = [10**(-x) for x in range(20, -1, -1)]
parameters = {'alpha': alphas}
ridge_regressor = GridSearchCV(ridge, parameters, scoring='neg_mean_squared_error', cv=5
)
ridge_regressor.fit(X_train, y_train)

print(ridge_regressor.best_params_)
print(ridge_regressor.best_score_)
```

```
{'alpha': 1e-20}
-25.24687428066025
```

In [16]:

```
from sklearn.linear_model import RidgeCV

ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
```

```
ridge_cv.fit(X_train, y_train)

print(ridge_cv.alpha_)
```

```
0.01
```

Lasso

In [17]:

```
from sklearn.linear_model import Lasso

lasso = Lasso(max_iter=100000)
lasso_regressor = GridSearchCV(lasso, parameters, scoring='neg_mean_squared_error', cv=5
)
lasso_regressor.fit(X_train, y_train)

print(lasso_regressor.best_params_)
print(lasso_regressor.best_score_)
```

```
{'alpha': 1e-14}
-25.246874280660307
```

In [18]:

```
from sklearn.linear_model import LassoCV

lasso_cv = LassoCV(alphas=alphas, cv=5)
lasso_cv.fit(X_train, y_train)

print(lasso_cv.alpha_)
```

```
1e-20
```

ElasticNet

In [19]:

```
from sklearn.linear_model import ElasticNet

elastic = ElasticNet(max_iter=10000)
elastic_regressor = GridSearchCV(elastic, parameters, scoring='neg_mean_squared_error',
cv=5)
elastic_regressor.fit(X_train, y_train)

print(elastic_regressor.best_params_)
print(elastic_regressor.best_score_)
```

```
{'alpha': 1e-20}
-25.246874280660304
```

In [20]:

```
from sklearn.linear_model import ElasticNetCV

elastic_cv = ElasticNetCV(alphas=alphas, cv=5)
elastic_cv.fit(X_train, y_train)

print(elastic_cv.alpha_)
```

```
1e-20
```

In [21]:

```
best_alpha_ridge = ridge_regressor.best_params_['alpha']
best_alpha_lasso = lasso_regressor.best_params_['alpha']
best_alpha_elastic = elastic_regressor.best_params_['alpha']

print("Best alpha for Ridge:", best_alpha_ridge)
print("Best alpha for Lasso:", best_alpha_lasso)
print("Best alpha for ElasticNet:", best_alpha_elastic)

best_alpha_ridge_cv = ridge_cv.alpha_
best_alpha_lasso_cv = lasso_cv.alpha_
best_alpha_elastic_cv = elastic_cv.alpha_

print("Best alpha for RidgeCV:", best_alpha_ridge_cv)
print("Best alpha for LassoCV:", best_alpha_lasso_cv)
print("Best alpha for ElasticNetCV:", best_alpha_elastic_cv)
```

```
Best alpha for Ridge: 1e-20
Best alpha for Lasso: 1e-14
Best alpha for ElasticNet: 1e-20
```

Задача 4.

Проверять качество правильно на кросс-валидации. Подключаем **cross_val_score** из **sklearn.model_selection**. Параметр **cv** установите равным 5.

Добейтесь **MSE < 27**.

In [22]:

```
from sklearn.model_selection import cross_val_score

# Ridge
ridge_best = Ridge(alpha=ridge_regressor.best_params_['alpha'], max_iter=10000)
ridge_score = cross_val_score(ridge_best, X, y, scoring="neg_mean_squared_error", cv=5)
ridge_mean_mse = -ridge_score.mean()

# Lasso
lasso_best = Lasso(alpha=lasso_regressor.best_params_['alpha'], max_iter=10000)
lasso_score = cross_val_score(lasso_best, X, y, scoring="neg_mean_squared_error", cv=5)
lasso_mean_mse = -lasso_score.mean()

# ElasticNet
elastic_best = ElasticNet(alpha=elastic_regressor.best_params_['alpha'], max_iter=10000)
elastic_score = cross_val_score(elastic_best, X, y, scoring="neg_mean_squared_error", cv
=5)
elastic_mean_mse = -elastic_score.mean()

print("Ridge MSE:", ridge_mean_mse)
print("Lasso MSE:", lasso_mean_mse)
print("ElasticNet MSE:", elastic_mean_mse)
```

```
Ridge MSE: 37.13180746769892
Lasso MSE: 37.131807467697776
ElasticNet MSE: 37.13180746769866
```

Как видим, при лучших альфа значение явно больше 27, поэтому проведем изменение датасета.

In [23]:

```
X["ZN"] = X["ZN"].replace(0, X['ZN'].mean())
```

In [24]:

```
# Inf и -Inf на NaN
X.replace([np.inf, -np.inf], np.nan, inplace=True)

# все NaN (включая те, которые были Inf или -Inf) на среднее значение по колонке
X.fillna(X.mean(), inplace=True)
```

In [25]:

```
features = ['CRIM', 'ZN', 'INDUS', 'NOX', 'AGE', 'DIS', 'TAX', 'PTRATIO', 'B', 'LSTAT']
for feature in features:
    X[feature]=np.log(X[feature])
```

In [26]:

```
X
```

Out[26]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	-5.064036	2.890372	0.837248	0.0	-0.619897	6.575	4.177459	1.408545	1.0	5.690359	2.727853	5.983684	1.605430
1	-3.600502	2.430418	1.955860	0.0	-0.757153	6.421	4.368181	1.602836	2.0	5.488938	2.879198	5.983684	2.212660
2	-3.601235	2.430418	1.955860	0.0	-0.757153	7.185	4.112512	1.602836	2.0	5.488938	2.879198	5.973377	1.393766
3	-3.430523	2.430418	0.779325	0.0	-0.780886	6.998	3.824284	1.802073	3.0	5.402677	2.928524	5.977949	1.078410
4	-2.672924	2.430418	0.779325	0.0	-0.780886	7.147	3.992681	1.802073	3.0	5.402677	2.928524	5.983684	1.673351
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	-2.770511	2.430418	2.479056	0.0	-0.556870	6.593	4.235555	0.907694	1.0	5.609472	3.044522	5.971236	2.269028
502	-3.095111	2.430418	2.479056	0.0	-0.556870	6.120	4.339902	0.827460	1.0	5.609472	3.044522	5.983684	2.206074
503	-2.800824	2.430418	2.479056	0.0	-0.556870	6.976	4.510860	0.773574	1.0	5.609472	3.044522	5.983684	1.729884
504	-2.211009	2.430418	2.479056	0.0	-0.556870	6.794	4.492001	0.870833	1.0	5.609472	3.044522	5.974954	1.868721

505	-3.048922	2.430418	2.479056	0.0	-0.556870	6.030	4.391977	0.918289	1.0	5.609472	3.044522	5.983684	2.064328

506 rows × 13 columns

In [27]:

```
from sklearn.model_selection import cross_val_score

# Ridge
ridge_best = Ridge(alpha=ridge_regressor.best_params_['alpha'], max_iter=10000)
ridge_score = cross_val_score(ridge_best, X, y, scoring="neg_mean_squared_error", cv=5)
ridge_mean_mse = -ridge_score.mean()

# Lasso
lasso_best = Lasso(alpha=lasso_regressor.best_params_['alpha'], max_iter=10000)
lasso_score = cross_val_score(lasso_best, X, y, scoring="neg_mean_squared_error", cv=5)
lasso_mean_mse = -lasso_score.mean()

# ElasticNet
elastic_best = ElasticNet(alpha=elastic_regressor.best_params_['alpha'], max_iter=10000)
elastic_score = cross_val_score(elastic_best, X, y, scoring="neg_mean_squared_error", cv
=5)
elastic_mean_mse = -elastic_score.mean()

print("Ridge MSE:", ridge_mean_mse)
print("Lasso MSE:", lasso_mean_mse)
print("ElasticNet MSE:", elastic_mean_mse)
```

```
Ridge MSE: 25.2485062597585
Lasso MSE: 25.24850625975862
ElasticNet MSE: 25.248506259758194
```