Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет»

Институт Компьютерных Наук и Технологий

Лабораторная работа № 6

по дисциплине

«Введение в анализ данных»

Отчет

Студент Панькова Светлана

Группа ИТ-13-2023

Пермь 2025

Задача №1.

В этом задании вам предстоит построить модель для прогнозирования цены недвижимости в зависимости от того, в каком районе Бостона она располагается.

1. Импортируем нужные нам библиотеки

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
```

2. Загрузите данные из файла "boston.csv" о недвижимости в различных районах Бостона.

```
# 1. Загрузите данные из файла "boston.csv"
# о недвижимости в различных районах Бостона.
data = pd.read_csv("boston.csv")
```

3. Проверьте, что у всех загруженных данных числовой тип.

```
# 2. Проверьте, что у всех загруженных данных числовой тип.
print("Типы данных\n")
print(data.dtypes)
```

```
Типы данных

CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

4. Проверьте, есть ли по каким-либо признакам отсутствующие данные. Если отсутствующие данные есть – заполните их медианным значением.

```
# 3. Проверьте, есть ли по каким-либо признакам отсутствующие данные.

# Если отсутствующие данные есть - заполните их медианным значением.

print("\nКоличество отсутствующих данных в каждом столбце:\n")

print(data.isnull().sum())

if data.isnull().values.any():

    # Заполнение пропущенных значений медианой

    # если True, будет изменять исходный DataFrame.

    data.fillna(data.median(), inplace=True)
```

```
Количество отсутствующих данных в каждом столбце:

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
```

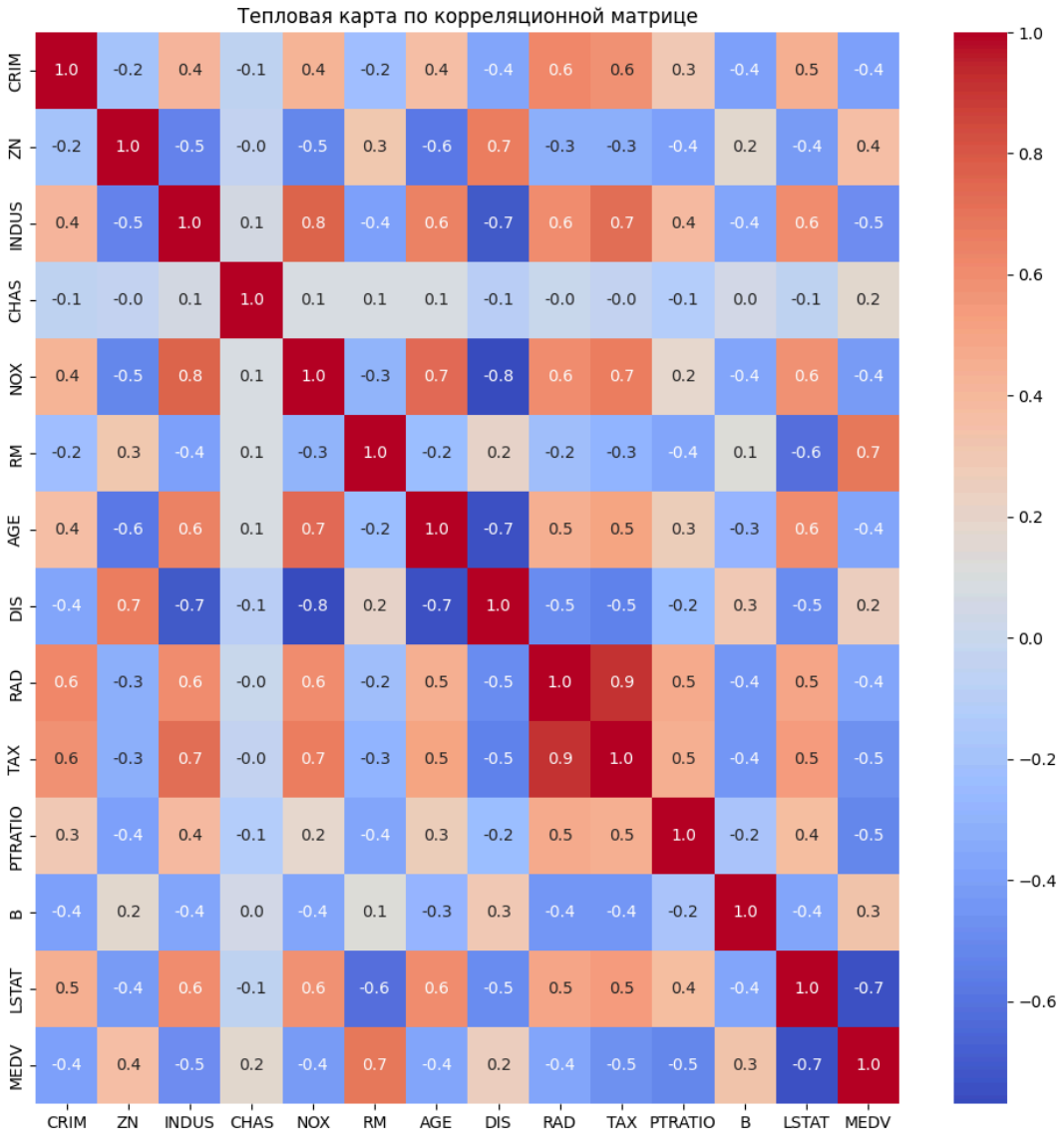
5. Посчитайте коэффициент корреляции для всех пар признаков. Подсказка: воспользуйтесь методом corr() для датафрейма, чтобы получить сразу всю корреляционную матрицу.

```
# 4. Посчитайте коэффициент корреляции для всех пар признаков.

# Подсказка: воспользуйтесь методом corr() для датафрейма,

# чтобы получить сразу всю корреляционную матрицу.

corr_matrix = data.corr()

print("\nКоэффициент корреляции для всех пар признаков:\n")

print(corr_matrix)
```

```
Коэффициент корреляции для всех пар признаков:

             CRIM        ZN     INDUS  ...         B     LSTAT      MEDV
CRIM     1.000000 -0.200469  0.406583  ... -0.385064  0.455621 -0.388305
ZN      -0.200469  1.000000 -0.533828  ...  0.175520 -0.412995  0.360445
INDUS    0.406583 -0.533828  1.000000  ... -0.356977  0.603800 -0.483725
CHAS    -0.055892 -0.042697  0.062938  ...  0.048788 -0.053929  0.175260
NOX      0.420972 -0.516604  0.763651  ... -0.380051  0.590879 -0.427321
RM      -0.219247  0.311991 -0.391676  ...  0.128069 -0.613808  0.695360
AGE      0.352734 -0.569537  0.644779  ... -0.273534  0.602339 -0.376955
DIS     -0.379670  0.664408 -0.708027  ...  0.291512 -0.496996  0.249929
RAD      0.625505 -0.311948  0.595129  ... -0.444413  0.488676 -0.381626
TAX      0.582764 -0.314563  0.720760  ... -0.441808  0.543993 -0.468536
PTRATIO  0.289946 -0.391679  0.383248  ... -0.177383  0.374044 -0.507787
B       -0.385064  0.175520 -0.356977  ...  1.000000 -0.366087  0.333461
LSTAT    0.455621 -0.412995  0.603800  ... -0.366087  1.000000 -0.737663
MEDV    -0.388305  0.360445 -0.483725  ...  0.333461 -0.737663  1.000000

[14 rows x 14 columns]
```

6. С помощью одной из библиотек визуализации постройте тепловую карту (heatmap) по корреляционной матрице.

```python
# 5. С помощью одной из библиотек визуализации постройте тепловую

# карту (heatmap) по корреляционной матрице.

plt.figure(figsize=(12, 12))

# Annot=True: значения в ячейках тепловой карты должны быть отображены на самой карте.

# Coolwarm — палитра в seaborn, которая отображает значения от холодных (синих) к теплым (красным) цветам.

sns.heatmap(corr_matrix, annot=True, fmt=".1f", cmap='coolwarm')

plt.title('Тепловая карта по корреляционной матрице')

plt.show()
```


Тепловая карта по корреляционной матрице
CRIM 1.0 -0.2 0.4 -0.1 0.4 -0.2 0.4 -0.4 0.6 0.6 0.3 -0.4 0.5 -0.4
ZN -0.2 1.0 -0.5 -0.0 -0.5 0.3 -0.6 0.7 -0.3 -0.3 -0.4 0.2 -0.4 0.4
INDUS 0.4 -0.5 1.0 0.1 0.8 -0.4 0.6 -0.7 0.6 0.7 0.4 -0.4 0.6 -0.5
CHAS -0.1 -0.0 0.1 1.0 0.1 0.1 0.1 -0.1 -0.0 -0.0 -0.1 0.0 -0.1 0.2
NOX 0.4 -0.5 0.8 0.1 1.0 -0.3 0.7 -0.8 0.6 0.7 0.2 -0.4 0.6 -0.4
RM -0.2 0.3 -0.4 0.1 -0.3 1.0 -0.2 0.2 -0.2 -0.3 -0.4 0.1 -0.6 0.7
AGE 0.4 -0.6 0.6 0.1 0.7 -0.2 1.0 -0.7 0.5 0.5 0.3 -0.3 0.6 -0.4
DIS -0.4 0.7 -0.7 -0.1 -0.8 0.2 -0.7 1.0 -0.5 -0.5 -0.2 0.3 -0.5 0.2
RAD 0.6 -0.3 0.6 -0.0 0.6 -0.2 0.5 -0.5 1.0 0.9 0.5 -0.4 0.5 -0.4
TAX 0.6 -0.3 0.7 -0.0 0.7 -0.3 0.5 -0.5 0.9 1.0 0.5 -0.4 0.5 -0.5
PTRATIO 0.3 -0.4 0.4 -0.1 0.2 -0.4 0.3 -0.2 0.5 0.5 1.0 -0.2 0.4 -0.5
B -0.4 0.2 -0.4 0.0 -0.4 0.1 -0.3 0.3 -0.4 -0.4 -0.2 1.0 -0.4 0.3
LSTAT 0.5 -0.4 0.6 -0.1 0.6 -0.6 0.6 -0.5 0.5 0.5 0.4 -0.4 1.0 -0.7
MEDV -0.4 0.4 -0.5 0.2 -0.4 0.7 -0.4 0.2 -0.4 -0.5 -0.5 0.3 -0.7 1.0
CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT MEDV
1.0 0.8 0.6 0.4 0.2 0.0 −0.2 −0.4 −0.6

7. Выберите от 4 до 6 признаков (на свое усмотрение), которые в наибольшей степени коррелируют с целевым признаком (ценой недвижимости). Справка. Коэффициент корреляции изменяется от -1 до 1, Значение -1 означает точную обратно-пропорциональную зависимость (чем меньше одна переменная, тем больше вторая, и наоборот). Значение 1 означает точную прямо-пропорциональную зависимость. Значение 0 означает полное отсутствие зависимости. Таким образом, чем ближе модуль коэффициента корреляции к 1, тем сильнее прослеживается зависимость между признаками.

```
# 6. Выберите от 4 до 6 признаков (на свое усмотрение), которые в наибольшей

# степени коррелируют с целевым признаком (ценой недвижимости). Справка.

# Коэффициент корреляции изменяется от -1 до 1, Значение -1 означает точную

# обратно-пропорциональную зависимость (чем меньше одна переменная, тем больше вторая, и наоборот).

# Значение 1 означает точную прямо-пропорциональную зависимость. Значение 0 означает полное

# отсутствие зависимости. Таким образом, чем ближе модуль коэффициента корреляции к 1,

# тем сильнее прослеживается зависимость между признаками.

# MEDV - медианная цена недвижимости (тыс. $) - целевой признак

target = 'MEDV'

#  ascending=False сортировка по убыванию

# выбираем индексы с 1 по 5, чтобы исключить первый индекс (который соответствует

# самой целевой переменной target, так как корреляция с самой собой всегда равна 1)

priznaks = corr_matrix[target].abs().sort_values(ascending=False).index[1:6]  # выбираем 5 признаков

print("Выбранные признаки:", priznaks)
```

```
Выбранные признаки: Index(['LSTAT', 'RM', 'PTRATIO', 'INDUS', 'TAX'], dtype='object')
```

8. Для каждого из выбранных признаков в паре с целевым признаком постройте точечную диаграмму (диаграмму рассеяния).

```
# 7. Для каждого из выбранных признаков в паре с целевым признаком
# постройте точечную диаграмму (диаграмму рассеяния).
for priznak in priznaks:
    plt.figure(figsize=(8, 8))
    # по оси X будут отложены значения текущего признака
    # по оси Y будут отложены значения целевой переменной
    sns.scatterplot(x=data[priznak], y=data[target])
    plt.title(f'Диаграмма рассеяния: {priznak} vs {target}')
    plt.xlabel(priznak)
    plt.ylabel(target)
    plt.show()
```

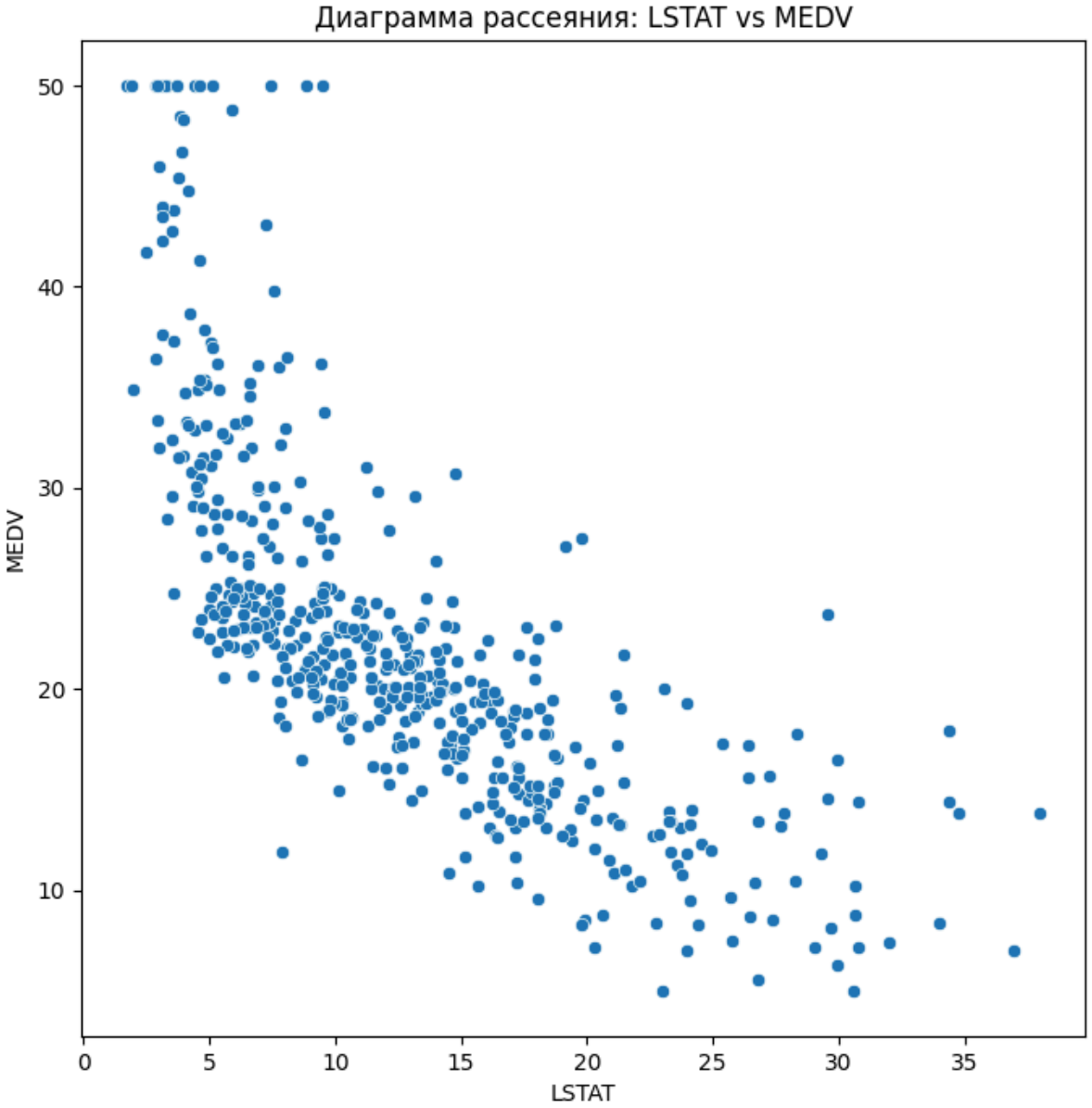


Диаграмма рассеяния: LSTAT vs MEDV
MEDV
LSTAT
50
40
30
20
10
0
5
10
15
20
25
30
35

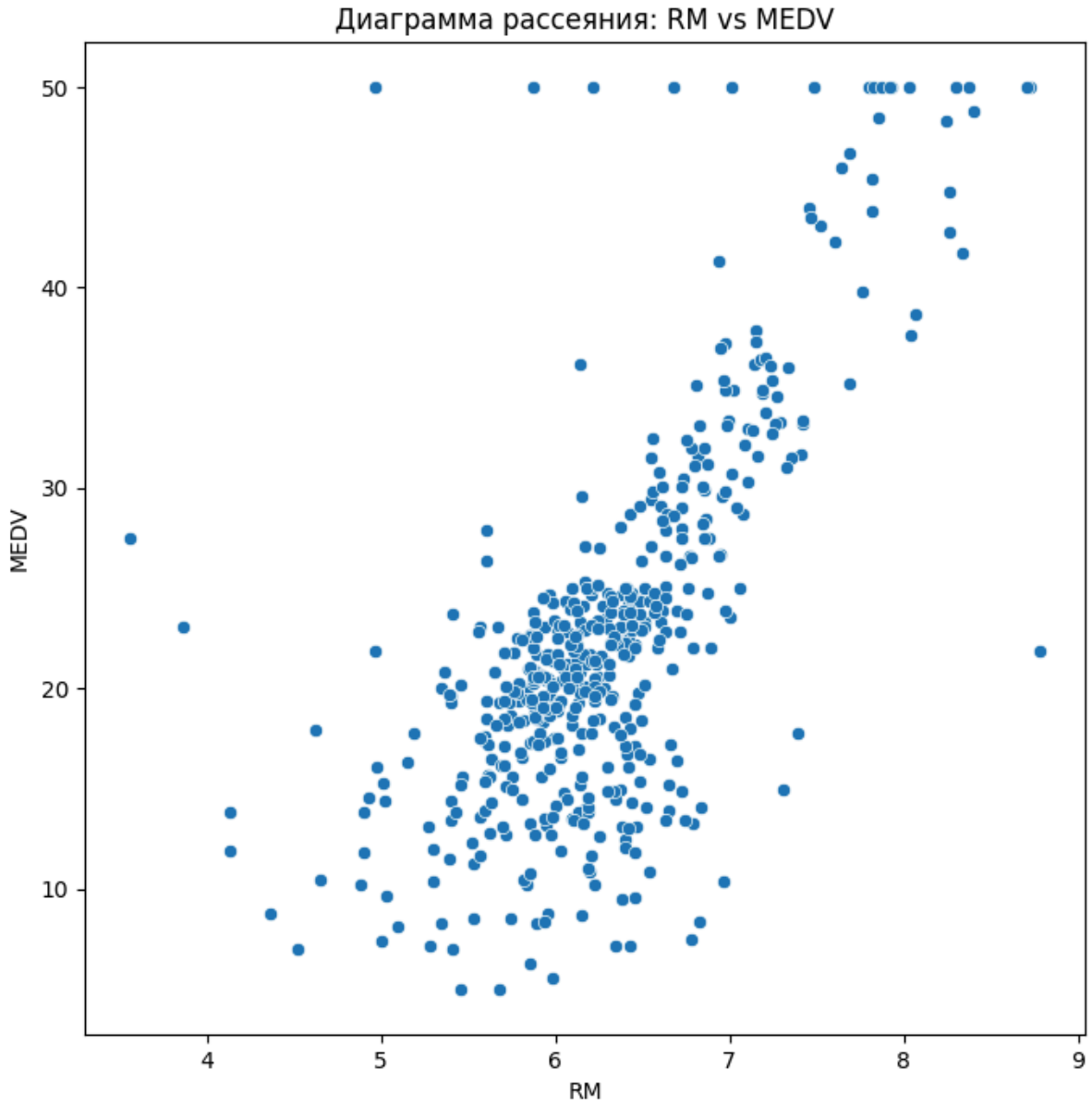


Диаграмма рассеяния: RM vs MEDV
MEDV
RM
50
40
30
20
10
4
5
6
7
8
9

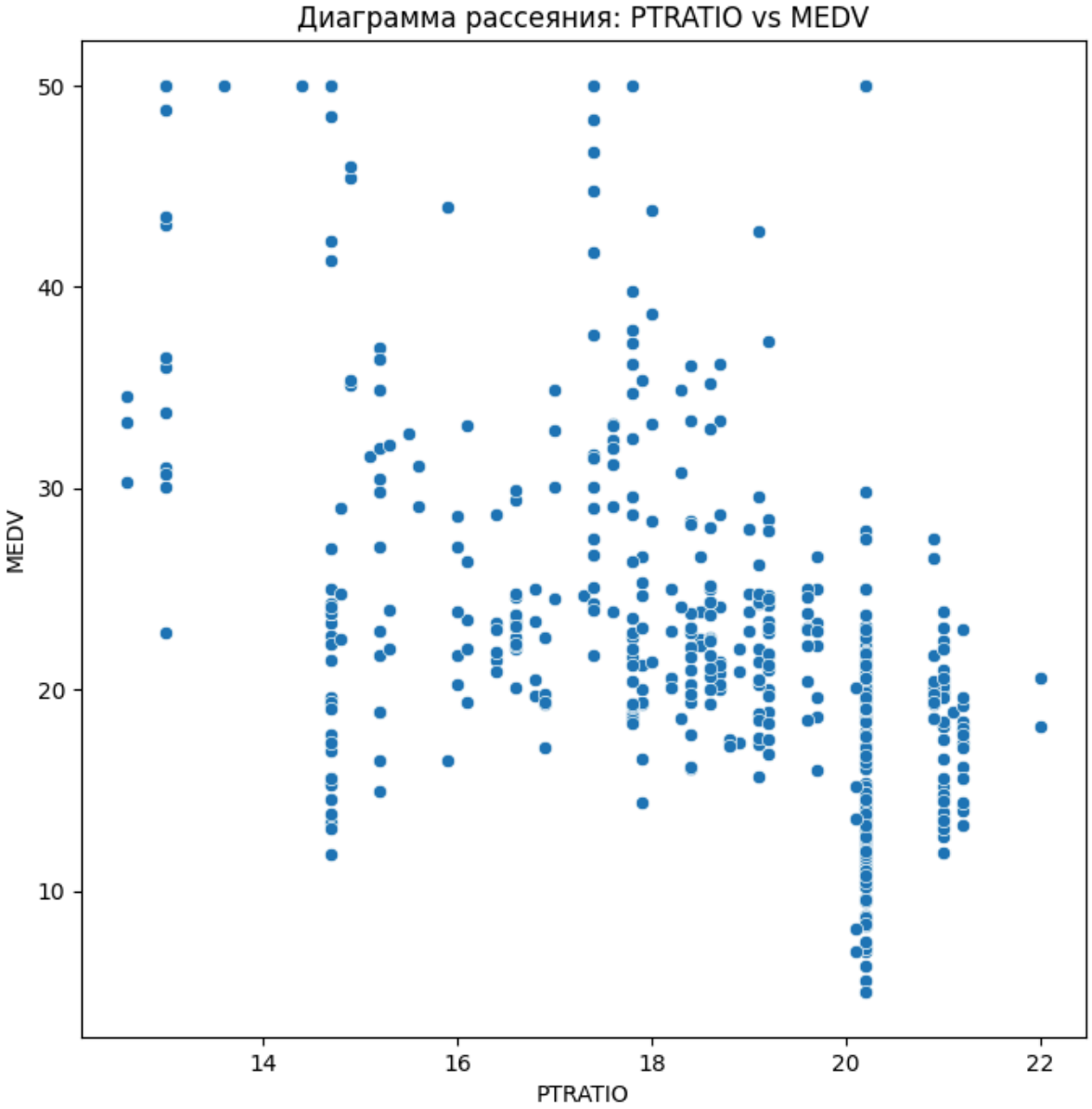


Диаграмма рассеяния: PTRATIO vs MEDV
MEDV
PTRATIO
50
40
30
20
10
14
16
18
20
22

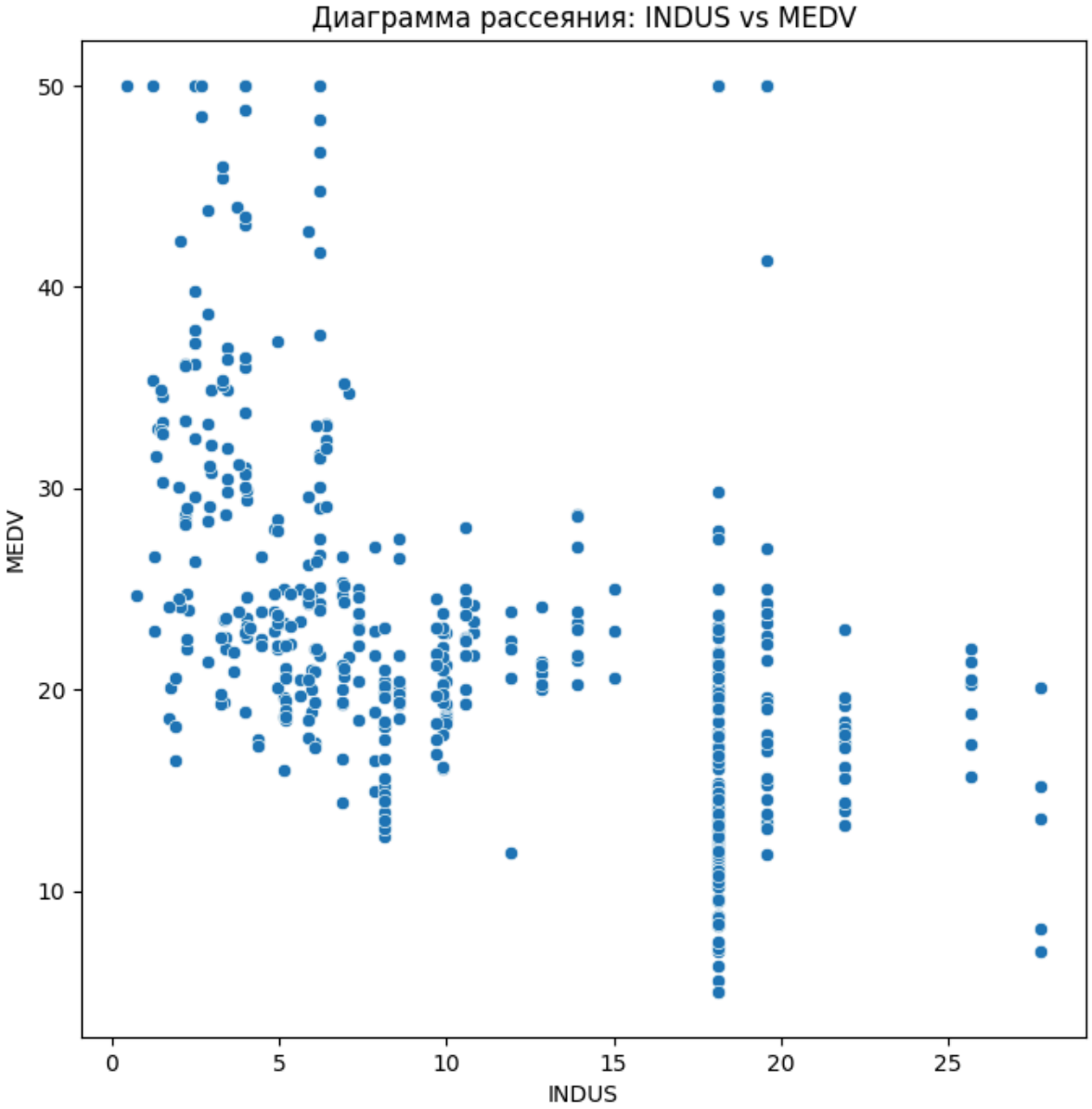
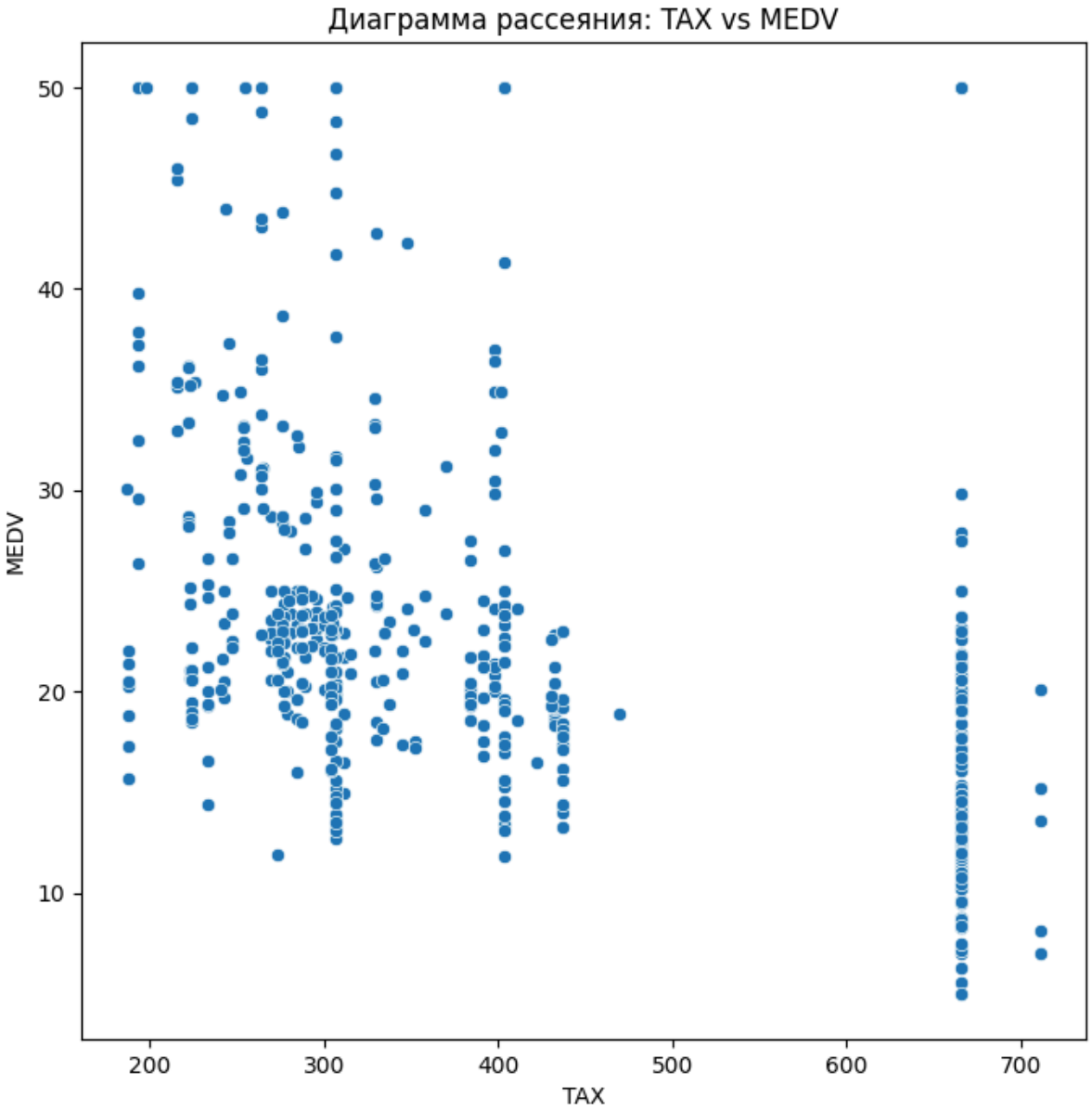


Диаграмма рассеяния: INDUS vs MEDV
MEDV
INDUS
50
40
30
20
10
0
5
10
15
20
25



9. Визуально убедитесь, что связь между выбранным признаком и целевым прослеживается. Если на основе графика считаете, что зависимости нет – исключите этот признак из дальнейшего рассмотрения (но при этом как минимум 3 признака должно остаться в любом случае).

```
# 8. Визуально убедитесь, что связь между выбранным признаком и

# целевым прослеживается. Если на основе графика считаете, что зависимости

# нет – исключите этот признак из дальнейшего рассмотрения (но при этом как
```

```
# минимум 3 признака должно остаться в любом случае).
priznaks = priznaks[0:3]
print("\nОставленные признаки:", priznaks)
```

```
Оставленные признаки: Index(['LSTAT', 'RM', 'PTRATIO'], dtype='object')
```

10. Сформируйте список факторных признаков и целевую переменную.

```
# 9. Сформируйте список факторных признаков и целевую переменную.
X = data[list(priznaks)]
y = data[target]
print("Список факторных признаков:\n", X)
print("Целевая переменная:\n", y)
```

```
Список факторных признаков:
      LSTAT     RM  PTRATIO
0      4.98  6.575     15.3
1      9.14  6.421     17.8
2      4.03  7.185     17.8
3      2.94  6.998     18.7
4      5.33  7.147     18.7
..      ...    ...      ...
501    9.67  6.593     21.0
502    9.08  6.120     21.0
503    5.64  6.976     21.0
504    6.48  6.794     21.0
505    7.88  6.030     21.0
```

```
Целевая переменная:
 0      24.0
1       21.6
2       34.7
3       33.4
4       36.2
       ...
501     22.4
502     20.6
503     23.9
504     22.0
505     11.9
```

11. Выполните разбиение датасета на обучающую и тестовую выборки в соотношении 8:2. При формировании обучающей и тестовой выборок строки из исходного датафрейма должны выбираться в случайном порядке. Подсказка: можно воспользоваться функцией train_test_split из библиотеки sklearn.model_selection.

```
# 10. Выполните разбиение датасета на обучающую и тестовую выборки в соотношении 8:2.

# При формировании обучающей и тестовой выборок строки из исходного датафрейма должны

# выбираться в случайном порядке. Подсказка: можно воспользоваться функцией train_test_split

# из библиотеки sklearn.model_selection.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=30)
```

12. Из набора линейных моделей библиотеки sklearn возьмите линейную регрессию, обучите ее на обучающем наборе.

```
# 11. Из набора линейных моделей библиотеки sklearn возьмите

# линейную регрессию, обучите ее на обучающем наборе.

lin_reg = LinearRegression()

lin_reg.fit(X_train, y_train)
```

13. Получите векторы прогнозных значений целевой переменной на обучающей и на тестовой выборках.

```
# 12. Получите векторы прогнозных значений целевой переменной на обучающей и на тестовой выборках.

y_train_pred = lin_reg.predict(X_train)

y_test_pred = lin_reg.predict(X_test)
```

14. Посчитайте коэффициент детерминации среднеквадратичной ошибки (RMSE) на обучающей и на тестовой выборках.

```
# 13. Посчитайте коэффициент детерминации (R2) и корень из

# среднеквадратичной ошибки (RMSE) на обучающей и на тестовой выборках.

# Функция r2_score вычисляет коэффициент детерминации (R²), который показывает,
```

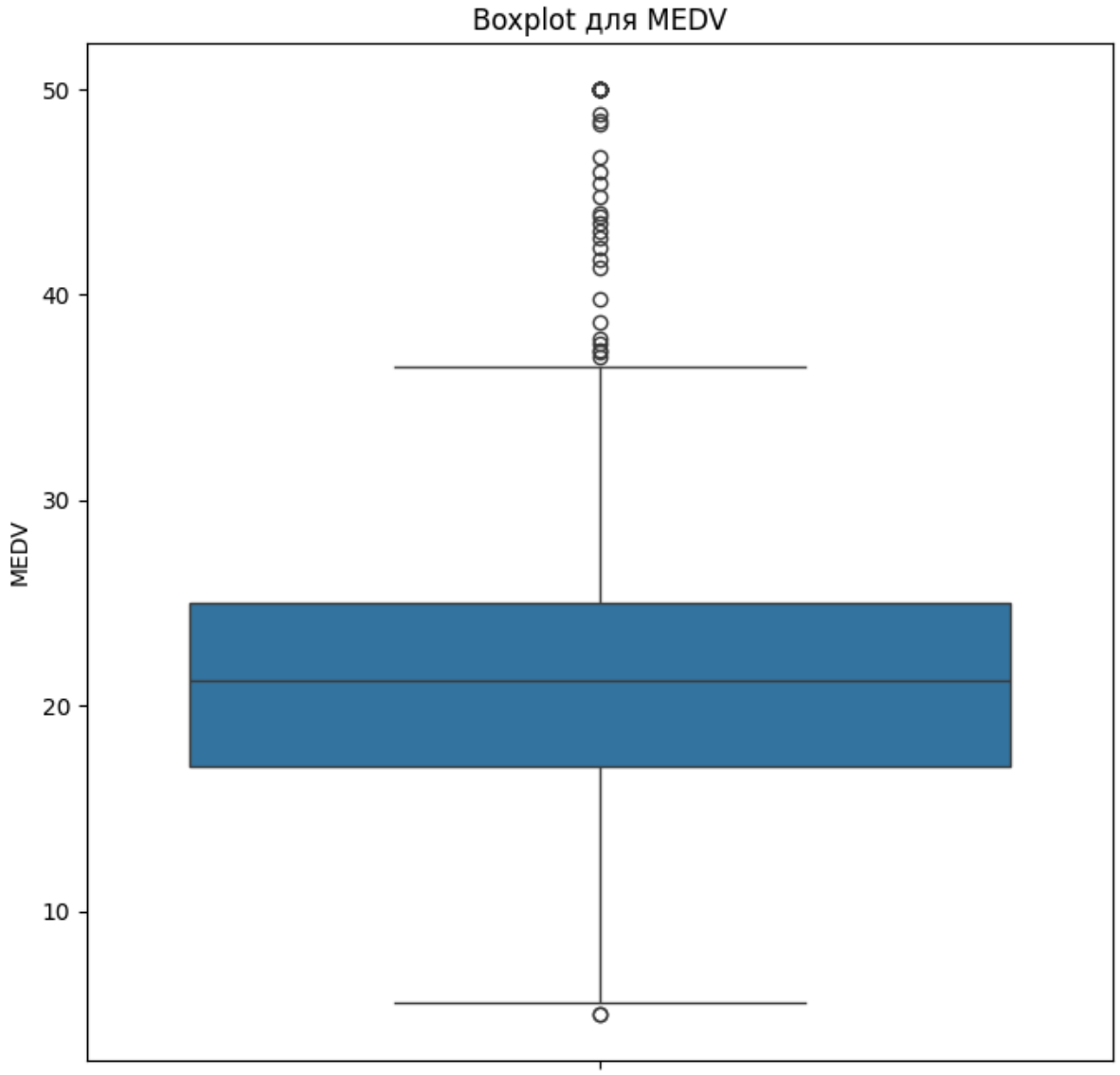
```
# какую долю дисперсии целевой переменной объясняет
модель.

# RMSE — это метрика, оценивающая точность регрессии,
чем меньше, тем лучше

# y_train — это истинные значения целевой переменной для
обучающего набора данных.

# y_train_pred — это предсказанные значения, которые
были получены моделью на том же наборе данных.

r2_train = r2_score(y_train, y_train_pred)

rmse_train = np.sqrt(mean_squared_error(y_train,
y_train_pred))

r2_test = r2_score(y_test, y_test_pred)

rmse_test = np.sqrt(mean_squared_error(y_test,
y_test_pred))


print(f'Обучающая выборка: R2 = {r2_train:}, RMSE =
{rmse_train:}')

print(f'Тестовая выборка: R2 = {r2_test:}, RMSE =
{rmse_test:}')
```

```
Обучающая выборка: R2 = 0.6818661844277927, RMSE = 5.341105889043236
Тестовая выборка: R2 = 0.6542932795600382, RMSE = 4.68971827499922
```

15. Постройте boxplot («ящик с усами») для целевого признака (MEDV). Определите, какие значения можно считать выбросами. Указание. Если по диаграмме выбросы определить не смогли, то для выполнения дальнейших действий считайте выбросами значения MEDV=50.0.

```
# 14. Постройте boxplot («ящик с усами») для целевого
признака (MEDV).

# Определите, какие значения можно считать выбросами.
Указание.

# Если по диаграмме выбросы определить не смогли, то для
выполнения
```

```
# дальнейших действий считайте выбросами значения
MEDV=50.0.

plt.figure(figsize=(8, 8))

sns.boxplot(y=data[target])

plt.title('Boxplot для MEDV')

plt.show()


# Определение выбросов

outs = data[(data[target] < 5.5) | (data[target] >
36.5)]

print("Выбросы:\n", outs)
```


Boxplot для MEDV
MEDV
50
40
30
20
10

```
Выбросы:
        CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  MEDV
97   0.12083   0.0   2.89   0.0  0.4450  ...  276.0     18.0  396.90   4.21  38.7
98   0.08187   0.0   2.89   0.0  0.4450  ...  276.0     18.0  393.53   3.57  43.8
157  1.22358   0.0  19.58   0.0  0.6050  ...  403.0     14.7  363.43   4.59  41.3
161  1.46336   0.0  19.58   0.0  0.6050  ...  403.0     14.7  374.43   1.73  50.0
162  1.83377   0.0  19.58   1.0  0.6050  ...  403.0     14.7  389.61   1.92  50.0
163  1.51902   0.0  19.58   1.0  0.6050  ...  403.0     14.7  388.45   3.32  50.0
166  2.01019   0.0  19.58   0.0  0.6050  ...  403.0     14.7  369.30   3.70  50.0
179  0.05780   0.0   2.46   0.0  0.4880  ...  193.0     17.8  396.90   5.04  37.2
180  0.06588   0.0   2.46   0.0  0.4880  ...  193.0     17.8  395.56   7.56  39.8
182  0.09103   0.0   2.46   0.0  0.4880  ...  193.0     17.8  394.12   4.82  37.9
186  0.05602   0.0   2.46   0.0  0.4880  ...  193.0     17.8  392.63   4.45  50.0
190  0.09068  45.0   3.44   0.0  0.4370  ...  398.0     15.2  377.68   5.10  37.0
195  0.01381  80.0   0.46   0.0  0.4220  ...  255.0     14.4  394.23   2.97  50.0
202  0.02177  82.5   2.03   0.0  0.4150  ...  348.0     14.7  395.38   3.11  42.3
203  0.03510  95.0   2.68   0.0  0.4161  ...  224.0     14.7  392.78   3.81  48.5
204  0.02009  95.0   2.68   0.0  0.4161  ...  224.0     14.7  390.55   2.88  50.0
224  0.31533   0.0   6.20   0.0  0.5040  ...  307.0     17.4  385.05   4.14  44.8
225  0.52693   0.0   6.20   0.0  0.5040  ...  307.0     17.4  382.00   4.63  50.0
226  0.38214   0.0   6.20   0.0  0.5040  ...  307.0     17.4  387.38   3.13  37.6
228  0.29819   0.0   6.20   0.0  0.5040  ...  307.0     17.4  377.51   3.92  46.7
232  0.57529   0.0   6.20   0.0  0.5070  ...  307.0     17.4  385.91   2.47  41.7
233  0.33147   0.0   6.20   0.0  0.5070  ...  307.0     17.4  378.95   3.95  48.3
253  0.36894  22.0   5.86   0.0  0.4310  ...  330.0     19.1  396.90   3.54  42.8
256  0.01538  90.0   3.75   0.0  0.3940  ...  244.0     15.9  386.34   3.11  44.0
```

16. Отфильтруйте исходные данные, удалив выбросы. Пересоздайте тестовую и обучающую выборки, переобучите модель. Посчитайте показатели R2 и RMSE. Как они изменились? О чем это говорит?

```
# 15. Отфильтруйте исходные данные, удалив выбросы.
Пересоздайте тестовую и обучающую выборки, переобучите
модель.

# Посчитайте показатели R2 и RMSE. Как они изменились? О
чем это говорит?

# Фильтруем данные

data_filtered = data[(data[target] > 5.5) &
(data[target] < 36.5)]

# Формируем список факторных признаков и целевую
переменную

X_filtered = data_filtered[list(priznaks)]
```

```
y_filtered = data_filtered[target]


# Разбиение датасета на обучающую и тестовую выборки

X_train_filt, X_test_filt, y_train_filt, y_test_filt =
train_test_split(X_filtered, y_filtered, test_size=0.2,
random_state=30)


# Линейная регрессия и обучение на новом наборе

lin_reg.fit(X_train_filt, y_train_filt)


# Векторы прогнозных значений целевой переменной на
обучающей и на тестовой выборках (новых)

y_train_filt_pred = lin_reg.predict(X_train_filt)

y_test_filt_pred = lin_reg.predict(X_test_filt)


# Коэффициент детерминации (R2) и корень из
среднеквадратичной ошибки (RMSE) на обучающей и на
тестовой выборках.

# y_train_filt — это истинные значения целевой
переменной для обучающего набора данных.

# y_train_filt_pred — это предсказанные значения,
которые были получены моделью на том же наборе данных.

r2_train_filt = r2_score(y_train_filt,
y_train_filt_pred)

rmse_train_filt =
np.sqrt(mean_squared_error(y_train_filt,
y_train_filt_pred))

r2_test_filt = r2_score(y_test_filt, y_test_filt_pred)

rmse_test_filt = np.sqrt(mean_squared_error(y_test_filt,
y_test_filt_pred))
```

```
print("\nСравнение моделей до и после удаления
выбросов")

print(f"R2 (тест до): {r2_test:.4f},  R2 (тест после):
{r2_test_filt:.4f}")

print(f"RMSE (тест до): {rmse_test:.4f},  RMSE (тест
после): {rmse_test_filt:.4f}")


print("\nВывод:")

if r2_test_filt > r2_test and rmse_test_filt <
rmse_test:

   print("После удаления выбросов качество модели
улучшилось: R² увеличился, RMSE уменьшился.")

elif r2_test_filt < r2_test and rmse_test_filt >
rmse_test:

   print("После удаления выбросов качество модели
ухудшилось: R² уменьшился, RMSE увеличился.")

else:

   print("После удаления выбросов качество модели
изменилось незначительно.")
```

```
Сравнение моделей до и после удаления выбросов
R2 (тест до): 0.6543,  R2 (тест после): 0.7557
RMSE (тест до): 4.6897,  RMSE (тест после): 3.2573

Вывод:
После удаления выбросов качество модели улучшилось: R² увеличился, RMSE уменьшился.
```

Показатели R2 увеличились – модель лучше справляется с предсказанием целевой переменной.

Показатели RMSE уменьшились – в модели меньше грубых ошибок.

17. Из набора линейных моделей библиотеки sklearn возьмите гребневую регрессию (Ridge). Обучите модель. Посчитайте показатели R2 и RMSE.

```
# 16. Из набора линейных моделей библиотеки sklearn
возьмите гребневую регрессию (Ridge).
```

```
# Обучите модель. Посчитайте показатели R2 и RMSE.

# Обучаем модель с гребневой регрессией на обучающем наборе

ridge_reg = Ridge()

ridge_reg.fit(X_train_filt, y_train_filt)


# Векторы прогнозных значений целевой переменной

y_train_ridge_pred = ridge_reg.predict(X_train_filt)

y_test_ridge_pred = ridge_reg.predict(X_test_filt)


# Коэффициент детерминации (R2) и корень из среднеквадратичной ошибки (RMSE) на обучающей и на тестовой выборках.

# y_train_filt — это истинные значения целевой переменной для обучающего набора данных.

# y_train_ridge_pred — это предсказанные значения, которые были получены моделью на том же наборе данных.

r2_train_ridge = r2_score(y_train_filt, y_train_ridge_pred)

rmse_train_ridge = np.sqrt(mean_squared_error(y_train_filt, y_train_ridge_pred))

r2_test_ridge = r2_score(y_test_filt, y_test_ridge_pred)

rmse_test_ridge = np.sqrt(mean_squared_error(y_test_filt, y_test_ridge_pred))


print(f'\nГребневая регрессия - Обучающая выборка: R2 = {r2_train_ridge:}, RMSE = {rmse_train_ridge:}')

print(f'Гребневая регрессия - Тестовая выборка: R2 = {r2_test_ridge:}, RMSE = {rmse_test_ridge:}')
```

```
Гребневая регрессия - Обучающая выборка: R2 = 0.6565617165995871, RMSE = 3.7273828467808117
Гребневая регрессия - Тестовая выборка: R2 = 0.7556199116753787, RMSE = 3.2576308654493746
```

18. Постройте полиномиальную регрессию с использованием полинома 3-й степени. Посчитайте показатели R2 и RMSE. Сравните все полученные результаты.

```
# 17. Постройте полиномиальную регрессию с использованием полинома 3-й степени.

# Посчитайте показатели R2 и RMSE. Сравните все полученные результаты.

# Преобразование исходных признаков в полиномиальные

# Полином 3-й степени degree=3

poly_reg = PolynomialFeatures(degree=3)

X_poly = poly_reg.fit_transform(X_filtered)


X_train_poly, X_test_poly, y_train_poly, y_test_poly = train_test_split(X_poly, y_filtered, test_size=0.2, random_state=30)


# Обучение модели полиномиальной регрессии

lin_reg.fit(X_train_poly, y_train_poly)

# Векторы прогнозных значений целевой переменной

y_train_poly_pred = lin_reg.predict(X_train_poly)

y_test_poly_pred = lin_reg.predict(X_test_poly)


# Коэффициент детерминации (R2) и корень из среднеквадратичной ошибки (RMSE) на обучающей и на тестовой выборках.

# y_train_filt — это истинные значения целевой переменной для обучающего набора данных.
```

```
# y_train_poly_pred — это предсказанные значения,
которые были получены моделью на том же наборе данных.

r2_train_poly = r2_score(y_train_poly,
y_train_poly_pred)

rmse_train_poly =
np.sqrt(mean_squared_error(y_train_poly,
y_train_poly_pred))

r2_test_poly = r2_score(y_test_poly, y_test_poly_pred)

rmse_test_poly = np.sqrt(mean_squared_error(y_test_poly,
y_test_poly_pred))



print(f'\nПолиномиальная регрессия - Обучающая выборка:
R2 = {r2_train_poly:}, RMSE = {rmse_train_poly:}')

print(f'Полиномиальная регрессия - Тестовая выборка: R2
= {r2_test_poly:}, RMSE = {rmse_test_poly:}')
```

```
Полиномиальная регрессия - Обучающая выборка: R2 = 0.7699764839618923, RMSE = 3.0504630494022758
Полиномиальная регрессия - Тестовая выборка: R2 = 0.8010749510659014, RMSE = 2.9390955097332774
```

```
# Сравнение моделей



# Создаем таблицу сравнения моделей

print("\nСравнение моделей после удаления выбросов:\n")

print(f"{'Модель':<25}{'R2 train':<12}{'RMSE
train':<12}{'R2 test':<12}{'RMSE test':<12}")

print("-" * 73)

print(f"{'Линейная
регрессия':<25}{r2_train_filt:<12.4f}{rmse_train_filt:<1
2.4f}{r2_test_filt:<12.4f}{rmse_test_filt:<12.4f}")

print(f"{'Гребневая
регрессия':<25}{r2_train_ridge:<12.4f}{rmse_train_ridge:
<12.4f}{r2_test_ridge:<12.4f}{rmse_test_ridge:<12.4f}")
```

```
print(f"{'Полиномиальная
регрессия':<25}{r2_train_poly:<12.4f}{rmse_train_poly:<1
2.4f}{r2_test_poly:<12.4f}{rmse_test_poly:<12.4f}")


# Выводим вывод о лучшей модели

print("\nВывод:")

best_model = ''

if r2_test_poly > r2_test_ridge and r2_test_poly >
r2_test_filt:

   best_model = 'Полиномиальная регрессия'

elif r2_test_ridge > r2_test_filt:

   best_model = 'Гребневая регрессия'

else:

   best_model = 'Линейная регрессия'


print(f"Лучшая модель: {best_model}.")
```

```
Сравнение моделей после удаления выбросов:

Модель                  R2 train    RMSE train  R2 test     RMSE test
-----------------------------------------------------------------------
Линейная регрессия      0.6566      3.7274      0.7557      3.2573
Гребневая регрессия     0.6566      3.7274      0.7556      3.2576
Полиномиальная регрессия 0.7700     3.0505      0.8011      2.9391

Вывод:
Лучшая модель: Полиномиальная регрессия.
```

Результаты в линейной и гребневой регрессии изменяются не значительно. Результаты полиномиальной регрессии отличаются от остальных. Показатели R2 увеличились – модель лучше справляется с предсказанием целевой переменной, Показатели RMSE уменьшились – в модели меньше грубых ошибок.