Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет»

Институт компьютерных наук и технологий

Отчет
по лабораторной работе № 6

по дисциплине
«Введение в анализ данных»

Студент Власова Елизавета Александровна

Группа ИТ–14–2023

Пермь 2025

Оглавление

Основная часть (5 баллов)

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

1. Загрузите данные из файла "boston.csv" о недвижимости в различных районах Бостона. Столбцы (признаки) имеют следующий смысл:
    a. CRIM – уровень преступности;
    b. ZN – доля жилых земель, разделенных на участки площадью более 25 000 кв.футов;
    c. INDUS – доля площадей, не связанных с розничной торговлей;
    d. CHAS – наличие реки (1, если граничит с рекой; 0 в противном случае);
    e. NOX – качество воздуха (концентрация оксидов азота);
    f. RM – среднее количество комнат в доме;
    g. AGE – доля жилых помещений, построенных владельцами до 1940 года;
    h. DIS – взвешенные расстояния до пяти бостонских центров занятости;
    i. RAD – транспортная доступность (индекс доступности радиальных автомагистралей);
    j. TAX – налоги (ставка налога на 10 000 долларов США);
    k. PTRATIO – соотношение количества учеников и учителей;
    l. B – нормированное значение доли афроамериканцев среди жителей;
    m. LSTAT – процент населения с низким социальным статусом;
    n. MEDV – медианная цена недвижимости (тыс. $) – это и будет целевой признак.

Решение:

pd.read_csv(...) — читает CSV-файл и возвращает pandas.DataFrame.

df.head(n=5) — метод DataFrame, возвращает верхние n строк (по умолчанию 5).

```
# 1
# читаем данные из boston.csv
df = pd.read_csv('boston.csv')
print("Первые строки набора данных:\n")
print(df.head())  # вывод первых 5 строк
```

Результаты работы программы:

```
Первые строки набора данных:

      CRIM    ZN  INDUS  CHAS    NOX  ...    TAX  PTRATIO       B  LSTAT  MEDV
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
```

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

Решение:

df.dtypes — атрибут (Series), который показывает тип данных каждого столбца в DataFrame.

```
# 2
# проверка, что все признаки числового типа
print("\n -- 2. Типы данных по столбцам: --")
print(df.dtypes)
```

Результаты работы программы:

```
 -- 2. Типы данных по столбцам: --
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
```

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

Решение:

df.isna() (или df.isnull()) — возвращает DataFrame того же размера, заполненный булевыми значениями: True где значение NaN/NA, False в противном случае.

.sum() примененное к DataFrame по умолчанию суммирует по столбцам, при этом True считается как 1. В результате получаем Series: для каждого столбца — количество пропусков.

```python
# 3
# проверка пропусков и заполнение медианой
print("\n -- 3. Количество пропусков в каждом столбце: --")
print(df.isna().sum())
df = df.fillna(df.median(numeric_only=True))
```

df.median(numeric_only=True) — вычисляет медиану для каждого числового столбца: Возвращает Series, индекс — имена столбцов, значения — медианы. numeric_only=True указывает: учитывать только числовые столбцы (иначе pandas может пытаться взять медиану от нечисловых и выдавать предупреждение). Если в столбце все значения NaN — медиана будет NaN.

df.fillna(...) — заполняет пропуски (NaN) значениями из переданной структуры. В данном случае заполняем пропуски в числовых столбцах (они все числовые) их медианами.

Результаты работы программы:

```
 -- 3. Количество пропусков 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
```

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

Решение:

df.corr(numeric_only=True) — вычисляет матрицу попарных коэффициентов корреляции между столбцами. По умолчанию считается корреляция Пирсона (линейная корреляция). Возвращает DataFrame: строки и столбцы — имена признаков; значения в ячейках — коэффициент корреляции от -1 до 1.

numeric_only=True ограничивает вычисление только числовыми столбцами.

Корреляция = 1 означает полную положительную линейную зависимость, -1 — полную отрицательную, 0 — отсутствие линейной связи.

```
# 4
# вычисление корреляционной матрицы
print("\n -- 4. Корреляционная матрица: --")
corr = df.corr(numeric_only=True) # для всех пар
print(corr)
```

Результаты работы программы:

```
 -- 4. Корреляционная матрица: --
             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
```

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

Решение:

plt.figure(figsize=(10,8)) — создаёт новый рисунок matplotlib (figure) с заданным размером в дюймах (ширина=10, высота=8).

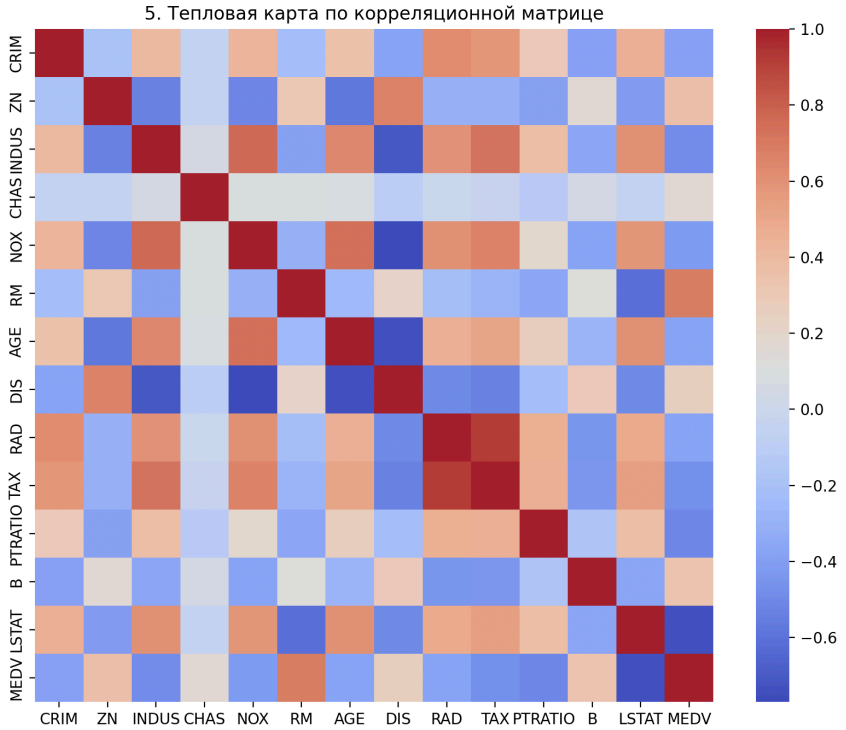
sns.heatmap(...) — функция из seaborn для визуализации матриц в виде "тепловой карты". Первый аргумент: 2D-структура (обычно DataFrame или 2D numpy array).

cmap='coolwarm' — палитра цветов, где обычно холодные (синие) — отрицательные значения, тёплые (красные) — положительные.

annot=False — параметр, управляющий аннотацией ячеек: если True, в каждой ячейке будет подписано числовое значение.

```
# 5
# тепловая карта корреляций
plt.figure(figsize=(10,8))
sns.heatmap(corr, cmap='coolwarm', annot=False)
plt.title("5. Тепловая карта по корреляционной матрице")
plt.show()
```

Результаты работы программы:



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.

Решение:

corr['MEDV']

- corr — это DataFrame (корреляционная матрица), у которого строки и столбцы — имена признаков.
- corr['MEDV'] — обращение к столбцу MEDV в этой матрице; возвращается pandas.Series, где индекс = имена признаков, значение = коэффициент корреляции между каждым признаком и MEDV.
- Тип возвращаемого: pandas.Series с индексом признаков и числовыми значениями в диапазоне [-1, 1].

.abs()

- Метод Series. Возвращает новую Series, где каждое значение заменено на его абсолютное значение (модуль).
- Причина: мы хотим выбрать признаки по силе линейной связи, независимо от знака (положительная или отрицательная корреляция).

.sort_values(ascending=False)

- Сортирует Series по значениям в порядке убывания (от наибольшей корреляции по модулю к наименьшей).
- Возвращает новую Series, индекс остаётся названия признаков, порядок переупорядочен.

.head(7)

- Берет верхние 7 элементов Series (первые 7 по отсортированному порядку).
- Возвращает Series длины ≤7 (если признаков меньше — вернет все).
- Здесь взяли 7, потому что в следующей строке удаляют сам MEDV — таким образом остаётся топ-6 признаков, наиболее коррелирующих с MEDV.

```
# 6
# выбор признаков с наибольшей корреляцией с MEDV
top6 = corr['MEDV'].abs().sort_values(ascending=False).head(7)
print("\n -- 6. Топ-6 признаков по корреляции с MEDV: --")
print(top6)
top6 = top6.index.drop('MEDV').tolist()
```

top6.index

- Для Series top6 атрибут .index — это Index (список меток), содержащий имена признаков, соответствующие верхним 7 элементам.

.drop('MEDV')

- Метод Index. Удаляет метку 'MEDV' из индекса. Это нужно, потому что корреляция признака MEDV с самим собой = 1, и он будет наверху списка; мы хотим признаки *внешние* по отношению к целевой переменной.

- Возвращает новый Index без 'MEDV'. Если 'MEDV' по какой-то причине отсутствует — .drop() выбросит KeyError; альтернативно можно было бы использовать top6.index.difference(['MEDV']).

.tolist()

- Преобразует Index в обычный питоновский список строк (имен признаков).
- Результат: top6 теперь — список (list) с 6 именами признаков, наиболее сильно (по модулю) коррелирующих с MEDV.

Результаты работы программы:

```
 -- 6. Топ-6 признаков по корреляции с MEDV: --
MEDV       1.000000
LSTAT      0.737663
RM         0.695360
PTRATIO    0.507787
INDUS      0.483725
TAX        0.468536
NOX        0.427321
Name: MEDV, dtype: float64
```

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

Решение:

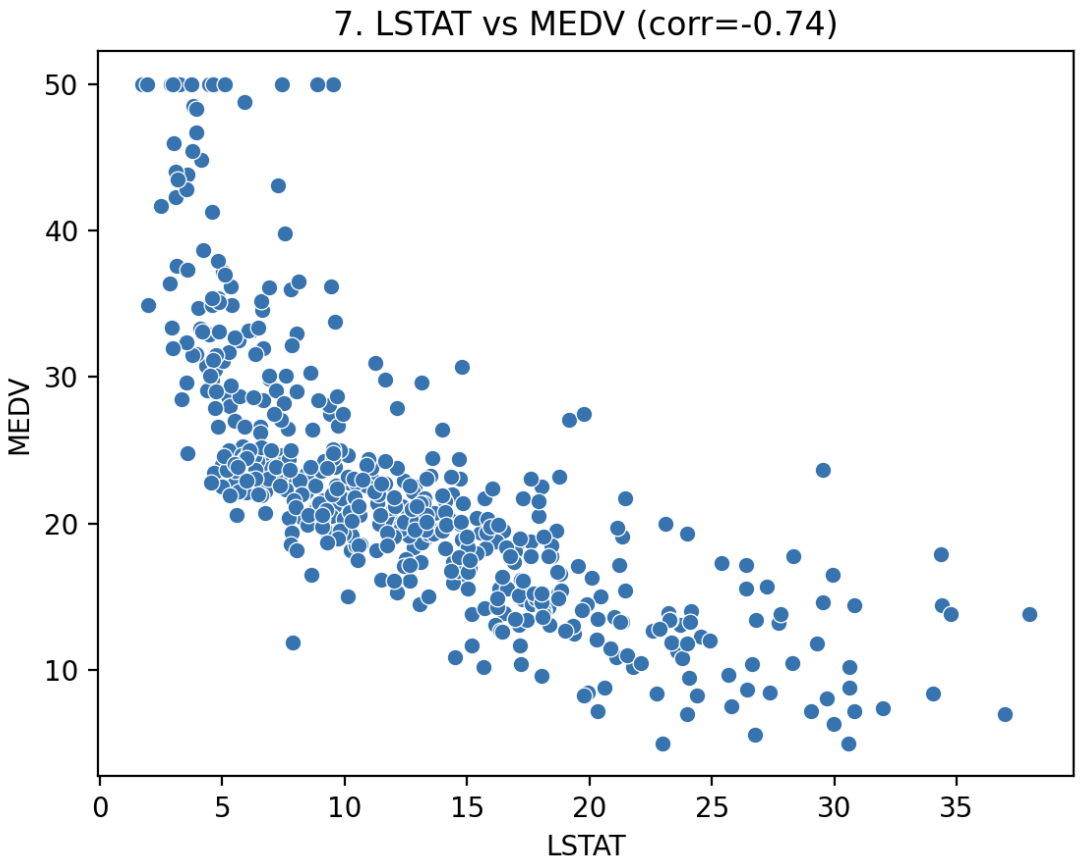
sns.scatterplot(x=df[f], y=df['MEDV'])

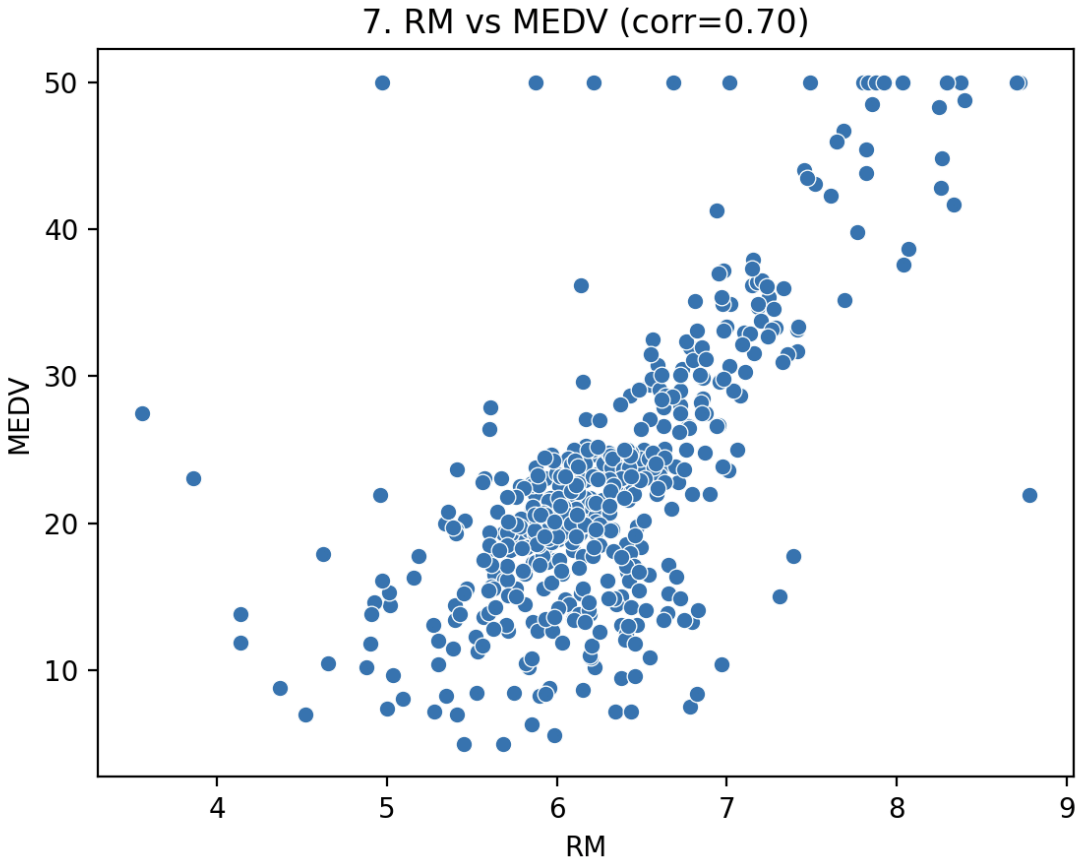
- sns.scatterplot — функция библиотеки **seaborn**, строит диаграмму рассеяния (scatter plot).
- Параметры:
    - x=df[f] — передаётся серия/массив значений признака f из DataFrame df. df[f] — pandas.Series.

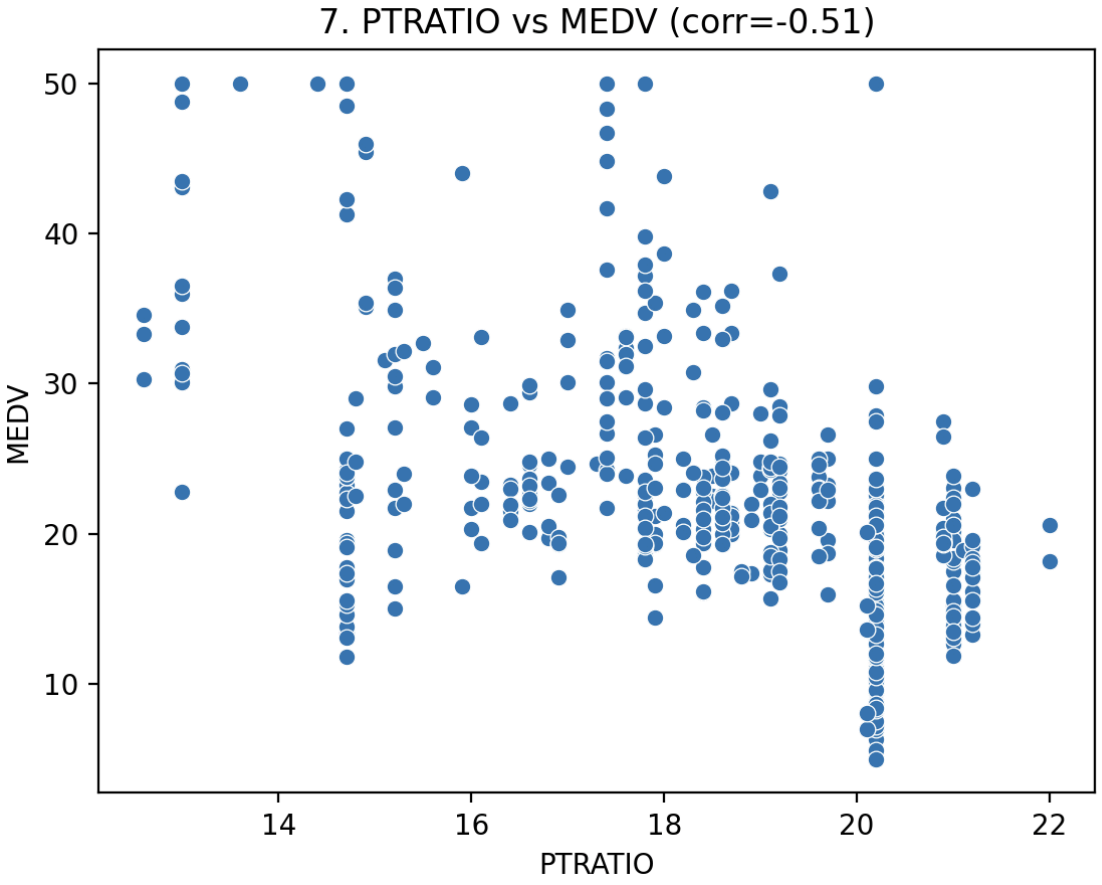
    - y=df['MEDV'] — серия целевой переменной.
- Внутреннее: seaborn использует matplotlib для отрисовки точек; по умолчанию добавляет эстетики (стили), может группировать по hue, size, style и т. д.
- Возвращает объект Axes matplotlib, но в коде он не сохраняется — используется для отрисовки на текущей фигуре.
- Зачем: визуально оценить форму связи (линейная/нелинейная), наличие выбросов, гетероскедастичность, кластеров.

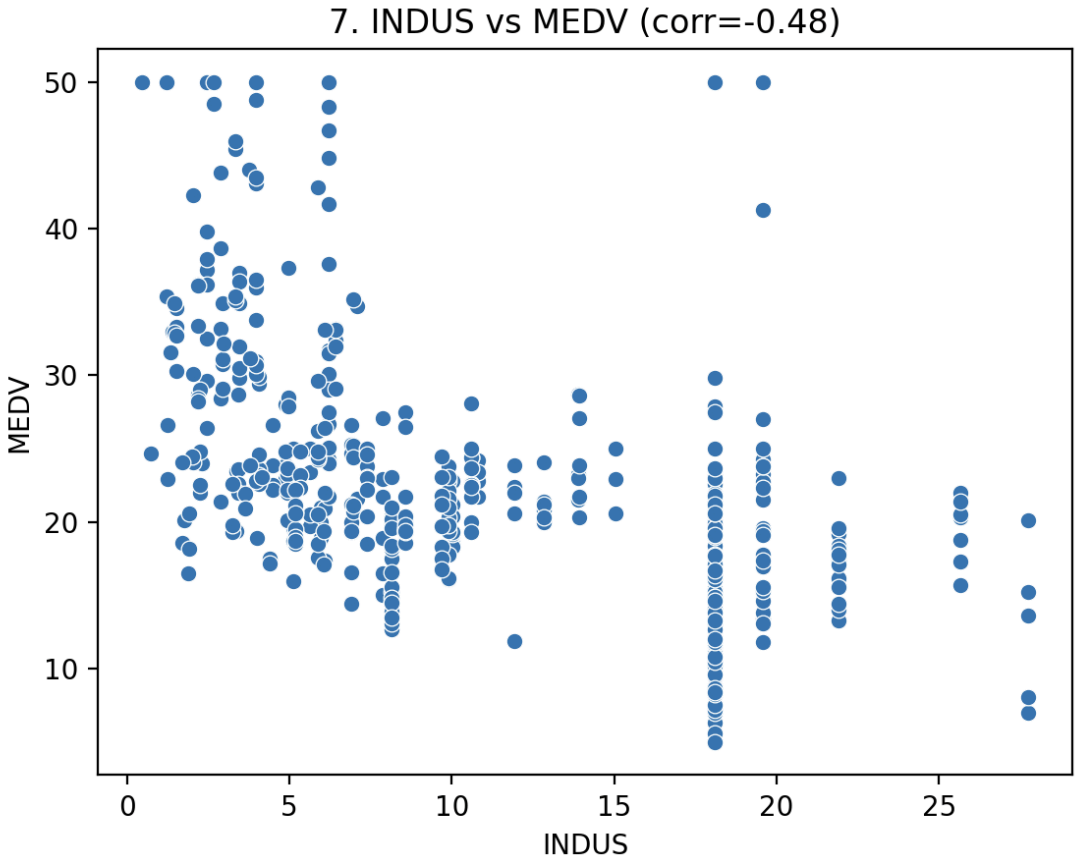
```
# 7
# диаграммы рассеяния для выбранных признаков
for f in top6:
    sns.scatterplot(x=df[f], y=df['MEDV'])
    plt.title(f"7. {f} vs MEDV (corr={corr.loc[f,'MEDV']:.2f})")
    plt.xlabel(f)
    plt.ylabel("MEDV")
    plt.show()
```

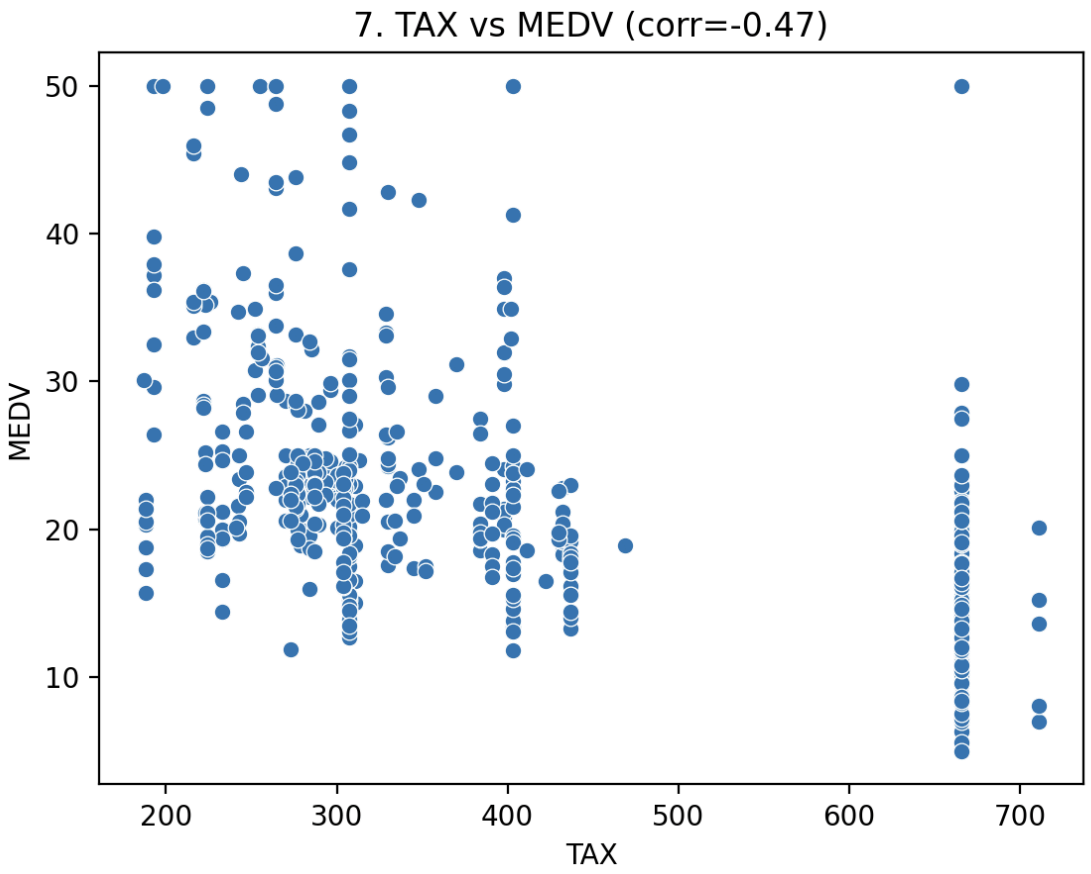
Результаты работы программы:

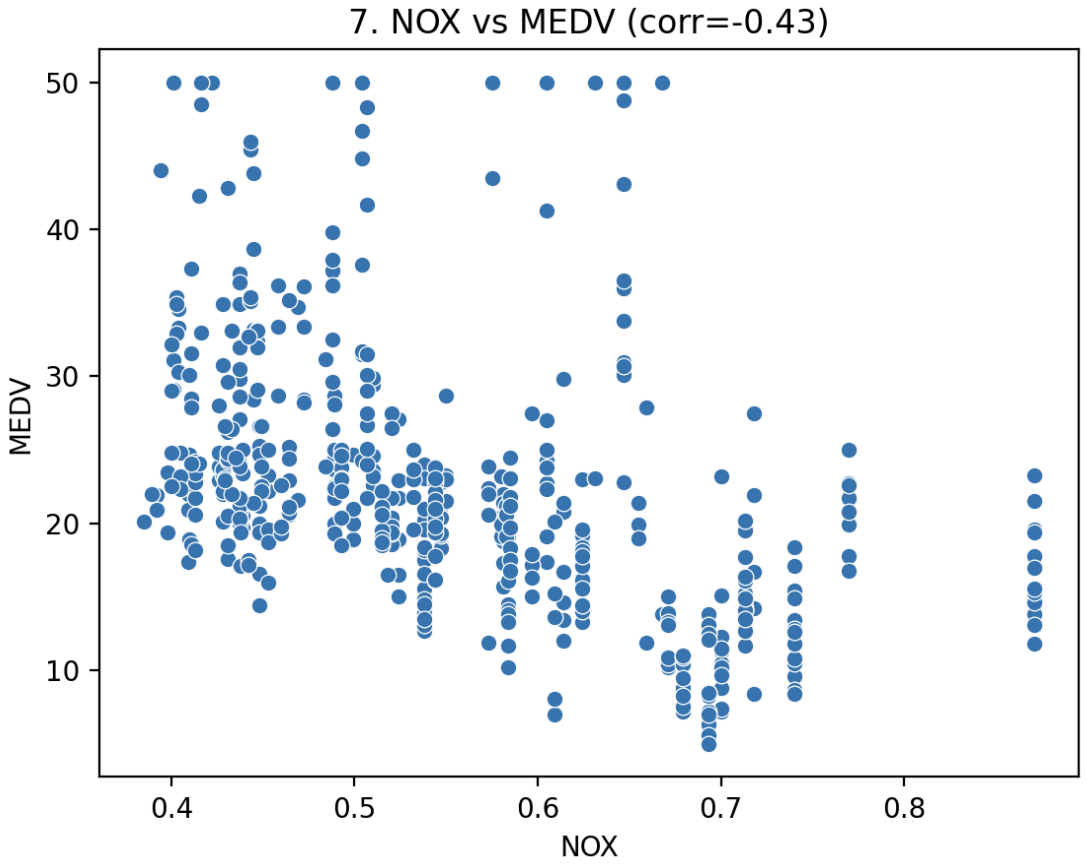

7. LSTAT vs MEDV (corr=-0.74)
MEDV
LSTAT
50
40
30
20
10
0
5
10
15
20
25
30
35


7. RM vs MEDV (corr=0.70)
MEDV
RM
50
40
30
20
10
4
5
6
7
8
9


7. PTRATIO vs MEDV (corr=-0.51)
MEDV
PTRATIO
50
40
30
20
10
14
16
18
20
22


7. INDUS vs MEDV (corr=-0.48)
MEDV
INDUS
50
40
30
20
10
0
5
10
15
20
25


7. TAX vs MEDV (corr=-0.47)
MEDV
TAX
50
40
30
20
10
200
300
400
500
600
700


7. NOX vs MEDV (corr=-0.43)
MEDV
NOX
50
40
30
20
10
0.4
0.5
0.6
0.7
0.8

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).

Зависимость прослеживается в каждом случае, следовательно оставляем все выбранные.

```
# 8
# визуальная проверка связи
# В данном случае оставляем все выбранные
```

Мы оставляем все выбранные признаки, потому что:

- они показали высокую (по модулю) корреляцию с целевой переменной;
- визуальный анализ (scatter plots) не выявил явных проблем (сильных выбросов, явной нелинейности, разделения на кластеры), которые требовали бы исключения.

9. Сформируйте список факторных признаков и целевую переменную.

Решение:

df[top6] — обращение к DataFrame pandas по списку колонок top6.

df['MEDV'] — обращение к столбцу DataFrame; возвращает pandas.Series длины n_samples. y — целевая переменная (в регрессии числовая).

```
# 9
# формируем набор признаков (X) и целевую переменную (y)
X = df[top6] # матрица признаков
y = df['MEDV'] # вектор целевой переменной
```

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.

Решение:

train_test_split — функция из sklearn.model_selection для рандомного разбиения данных.

Аргументы:

- X: признаки (DataFrame или ndarray)
- y: метки/целевая переменная (Series или ndarray)
- test_size=0.2: доля выборки, отведённая под тест — 20% данных пойдут в X_test/y_test. Можно указать как float (доля) или как int (число объектов).
- random_state=42: фиксирует состояние генератора случайных чисел, чтобы разбиение было воспроизводимым. Если не задать, каждый запуск будет давать новое разбиение.
- shuffle=True: перед разделением данные будут перемешаны (по умолчанию True).

Возвращаемые значения (в указанном порядке):

- X_train: признаки для обучения (80% строк)
- X_test: признаки для теста (20% строк)

- y_train: целевая для обучения
- y_test: целевая для теста

```
# 10
# разделение данных на обучающую и тестовую выборки (80/20)
X_train, X_test, y_train, y_test = train_test_split(X,  y, test_size=0.2, random_state=42, shuffle=True)
```

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

Решение:

LinearRegression — класс из sklearn.linear_model, реализует метод наименьших квадратов (OLS).

.fit(X, y) — подгоняет параметры модели по обучающим данным. Для LinearRegression выполняется решение нормальных уравнений (или SVD) для минимизации суммы квадратов ошибок: min_w ||X w + b - y||^2

```
# 11
# обучение линейной регрессии
lin = LinearRegression() # создаем объект модели линейной регрессии
lin.fit(X_train, y_train) # обучаем модель
```

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

Решение:

lin.predict(X) — выдаёт прогнозы модели для переданных признаков X: Ожидает X в виде массива формы (n_samples, n_features). Внутри выполняется линейная комбинация: y_pred = X @ coef_ + intercept_

Результат: numpy.ndarray длины n_samples (тип float).

То же для X_test — прогнозы для отложенной выборки. Эти предсказания используются для оценки обобщающей способности.

```
# 12
# прогнозирование на обучающей и тестовой выборках
y_pred_train = lin.predict(X_train) # для обучающего набора
y_pred_test = lin.predict(X_test) # для тестового набора
```

13. Посчитайте коэффициент детерминации (R^2) и корень из среднеквадратичной ошибки (RMSE) на обучающей и на тестовой выборках.

Решение:

r2_score(y_true, y_pred) — функция из sklearn.metrics, вычисляет коэффициент детерминации R^2.

Аналогично для тестовой выборки. Обычно сравнивают r2_train и r2_test, чтобы оценить переобучение/недообучение.

mean_squared_error(y_true, y_pred) — вычисляет MSE = mean((y_true - y_pred)^2). В коде берем квадратный корень: RMSE = sqrt(MSE), это возвращает ошибку в тех же единицах, что и y (удобнее интерпретировать).

RMSE на тесте — ключевая метрика качества регрессии: показывает среднюю величину ошибки предсказания.

Форматируем вывод с 3 знаками после запятой для компактности.

```
# 13
# вычисление метрик R^2 и RMSE
r2_train = r2_score(y_train, y_pred_train) # коэффициент детерминации (насколько хорошо модель объясняет данные)
r2_test = r2_score(y_test, y_pred_test)
rmse_train = mean_squared_error(y_train, y_pred_train)**0.5 # корень из среднеквадратичной ошибки (на сколько модель ошибается)
rmse_test = mean_squared_error(y_test, y_pred_test)**0.5

# выводим результаты
print("\n -- 13. Линейная регрессия: --")
print(f"Train R^2={r2_train:.3f}, RMSE={rmse_train:.3f}")
print(f"Test  R^2={r2_test:.3f}, RMSE={rmse_test:.3f}")
```

Результаты работы программы:

```
 -- 13. Линейная регрессия: --
Train R^2=0.691, RMSE=5.184
Test  R^2=0.621, RMSE=5.272
```

Дополнительные задания (4 балла)

14. (0.5 балла) 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.

Решение:

sns.boxplot(x=df['MEDV'])

- Использует библиотеку Seaborn (sns) для построения boxplot (ящик с усами) для столбца MEDV датафрейма df.
- x=df['MEDV'] указывает, что строим график для одного числового признака MEDV.
- Boxplot позволяет визуально оценить:
    - Медиану (центральное значение данных)
    - Первый квартиль (Q1) — 25-й процентиль
    - Третий квартиль (Q3) — 75-й процентиль
    - Межквартильный размах (IQR) — расстояние между Q3 и Q1
    - Выбросы — значения, которые значительно отличаются от основной массы данных

df[df['MEDV'] < 36.5]

- Берёт только те строки df, где значение MEDV меньше 36.5.
- Другими словами, отсекаются слишком большие значения, которые считаются выбросами.

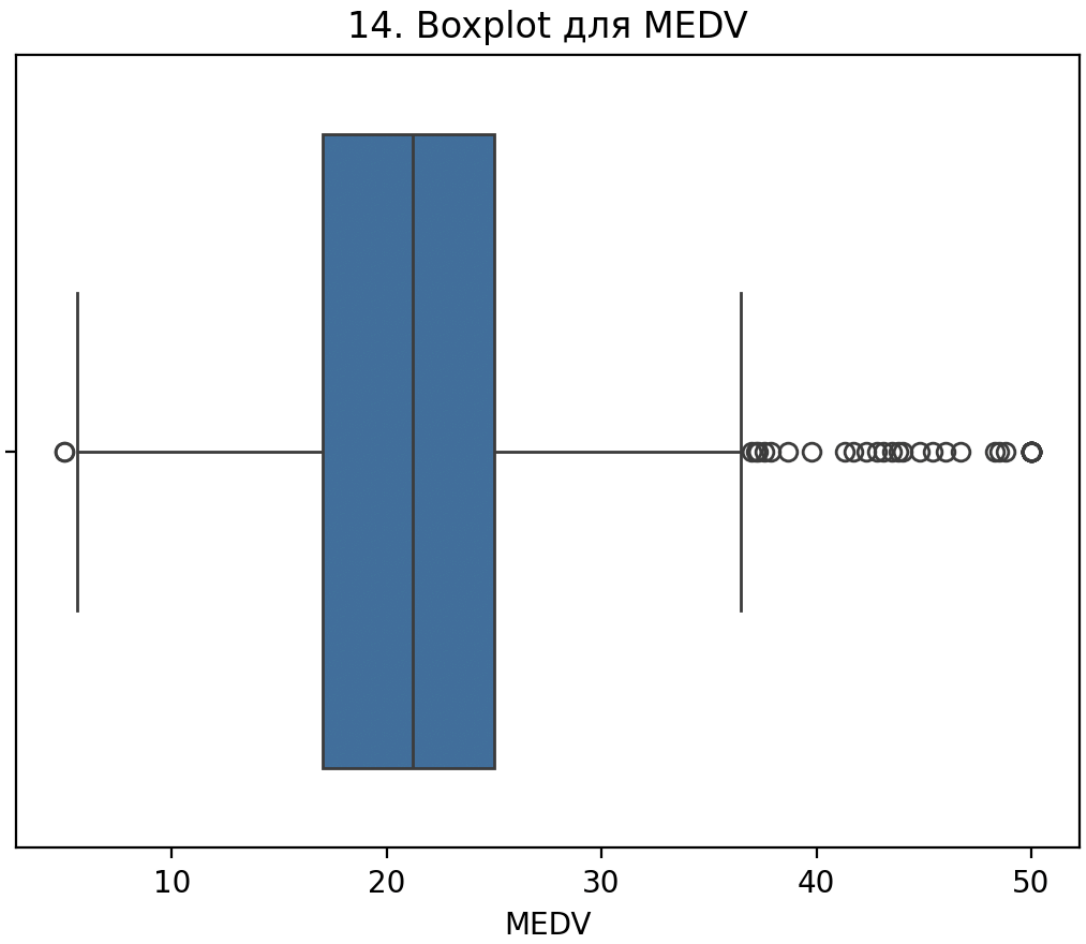
df[df['MEDV'] > 5.5]

- Берет только те строки df, где значение MEDV больше 5.5.
- Таким образом, отсекаются слишком маленькие значения, тоже считающиеся выбросами.

В итоге в датафрейме остаются значения MEDV в диапазоне [5.5; 36.5], что снижает влияние экстремальных выбросов на анализ.

```
# 14
# Boxplot для целевого признака MEDV (поиск выбросов)
sns.boxplot(x=df['MEDV'])
plt.title("14. Boxplot для MEDV")
plt.show()

# считаем выбросами MEDV вне [5.5; 36.5]
df = df[df['MEDV'] < 36.5]
df = df[df['MEDV'] > 5.5]
```

Результаты работы программы:



15. (1 балл) Отфильтруйте исходные данные, удалив выбросы. Пересоздайте тестовую и обучающую выборки, переобучите модель. Посчитайте показатели R^2 и RMSE. Как они изменились? О чем это говорит?

Решение:

X — это матрица признаков, которую модель использует для обучения. df[top6] создаёт датафрейм X только с этими признаками.

y — это целевой признак, который модель будет прогнозировать (значения цены домов MEDV).

Функция train_test_split (из sklearn.model_selection) делает следующее:

- X, y — входные данные и целевая переменная.

- test_size=0.2 — 20% данных выделяется для тестирования, 80% для обучения.
- random_state=42 — фиксирует случайное разбиение, чтобы результаты были воспроизводимыми.
- shuffle=True — перемешивание данных перед разбиением, чтобы не было смещений (например, временной зависимости).

Результат:

- X_train и y_train — данные для обучения модели
- X_test и y_test — данные для оценки качества модели

Обучение модели:
lin — это объект линейной модели (например, LinearRegression()).
.fit(X_train, y_train) обучает модель на тренировочных данных:

- вычисляет коэффициенты для каждого признака в top6, чтобы минимизировать ошибку между предсказанными и фактическими MEDV.

.predict() делает прогноз модели:

- y_pred_train — предсказания на тренировочных данных
- y_pred_test — предсказания на тестовых данных

```python
# 15
# повторное обучение после удаления выбросов
X = df[top6] # заново формируем X и y
y = df['MEDV']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=True
)

lin.fit(X_train, y_train) # переобучаем линейную модель

# делаем новые прогнозы
y_pred_train = lin.predict(X_train)
y_pred_test = lin.predict(X_test)
r2_train_clean = r2_score(y_train, y_pred_train)
r2_test_clean = r2_score(y_test, y_pred_test)
rmse_train_clean = mean_squared_error(y_train, y_pred_train) ** 0.5
rmse_test_clean = mean_squared_error(y_test, y_pred_test) ** 0.5

print("\n -- 15. После удаления выбросов: --")
print(f"Train R^2 = {r2_train_clean:.3f}, RMSE = {rmse_train_clean:.3f}")
print(f"Test  R^2 = {r2_test_clean:.3f}, RMSE = {rmse_test_clean:.3f}")

# сравнение с результатами до очистки
# R² (коэффициент детерминации)
# показывает насколько хорошо модель объясняет дисперсию данных
# (доля объяснённой вариации). Чем больше, тем лучше (идеально = 1).
if 'r2_test' in locals():
    if r2_test_clean > r2_test:
        print("--> Качество модели улучшилось после удаления выбросов")
    elif r2_test_clean < r2_test:
        print("--> Качество модели ухудшилось после удаления выбросов")
    else:
        print("--> Удаление выбросов не повлияло на качество модели")
```

Результаты работы программы:

```
 -- 15. После удаления выбросов: --
Train R^2 = 0.704, RMSE = 3.448
Test  R^2 = 0.708, RMSE = 3.604
--> Качество модели улучшилось после удаления выбросов
```

16. (1 балл) Из набора линейных моделей библиотеки sklearn возьмите гребневую регрессию (Ridge). Обучите модель. Посчитайте показатели R^2 и RMSE.

Решение:

```python
# 16
# гребневая (Ridge) регрессия
# линейная регрессия, но плюсом Ridge добавляет штраф (регуляризацию) за большие коэффициенты
ridge = Ridge(alpha=1.0) # создаем модель
ridge.fit(X_train, y_train) # обучаем модель

y_pred_train_ridge = ridge.predict(X_train)  # прогноз на обучающей
y_pred_test_ridge = ridge.predict(X_test)    # прогноз на тестовой

# для обучающей выборки
r2_train_ridge = r2_score(y_train, y_pred_train_ridge)
rmse_train_ridge = mean_squared_error(y_train, y_pred_train_ridge) ** 0.5

# для тестовой выборки
r2_test_ridge = r2_score(y_test, y_pred_test_ridge)
rmse_test_ridge = mean_squared_error(y_test, y_pred_test_ridge) ** 0.5

# выводим результаты
print("\n -- 16. Гребневая регрессия (Ridge) --")
print(f"Train R^2 = {r2_train_ridge:.3f}, RMSE = {rmse_train_ridge:.3f}")
print(f"Test  R^2 = {r2_test_ridge:.3f}, RMSE = {rmse_test_ridge:.3f}")

# выходит предупреждение от SciPy, что матрица признаков (X) или ее производная (XᵀX) имеет плохую численную обусловленность
```

Результаты работы программы:

```
 -- 16. Гребневая регрессия (Ridge) --
Train R^2 = 0.703, RMSE = 3.453
Test  R^2 = 0.710, RMSE = 3.590
/Users/lisa/Downloads/lab6_vad/venv/lib/python3.14/site-packages/scipy/_lib/_u
til.py:1233: LinAlgWarning: Ill-conditioned matrix (rcond=1.41633e-19): result
 may not be accurate.
  return f(*arrays, *other_args, **kwargs)
```

17. (1.5 балла) Постройте полиномиальную регрессию с использованием полинома 3й степени. Посчитайте показатели R^2 и RMSE. Сравните все полученные результаты.

Решение:

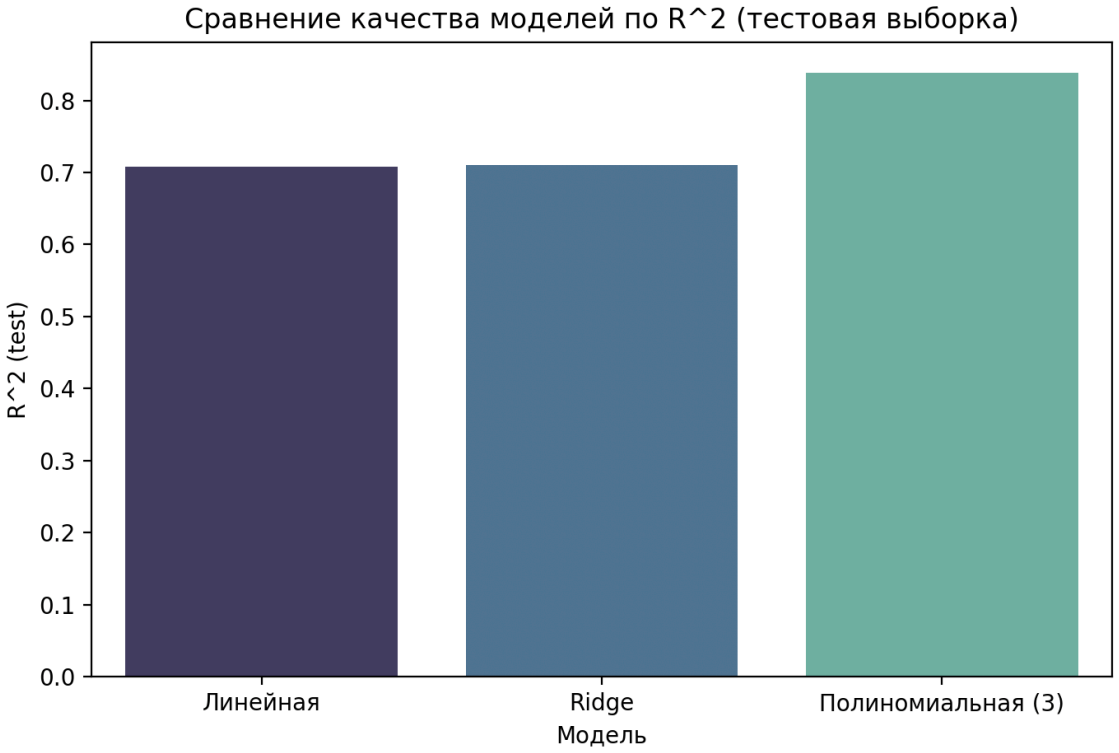
```python
# 17
# полиномиальная регрессия
# создаем конвейер: полином 3-й степени + гребневая регрессия
poly = make_pipeline(
    PolynomialFeatures(degree=3, include_bias=False),
    Ridge(alpha=1.0)
)
poly.fit(X_train, y_train) # обучаем на тренировочных данных

# делаем прогноз на обеих выборках
y_pred_train_poly = poly.predict(X_train)
y_pred_test_poly = poly.predict(X_test)

# для обучающей выборки
r2_train_poly = r2_score(y_train, y_pred_train_poly)
rmse_train_poly = mean_squared_error(y_train, y_pred_train_poly) ** 0.5

# для тестовой выборки
r2_test_poly = r2_score(y_test, y_pred_test_poly)
rmse_test_poly = mean_squared_error(y_test, y_pred_test_poly) ** 0.5

# выводим результаты
print("\n -- 17. Полиномиальная регрессия --")
print(f"Train R^2 = {r2_train_poly:.3f}, RMSE = {rmse_train_poly:.3f}")
print(f"Test  R^2 = {r2_test_poly:.3f}, RMSE = {rmse_test_poly:.3f}")
```

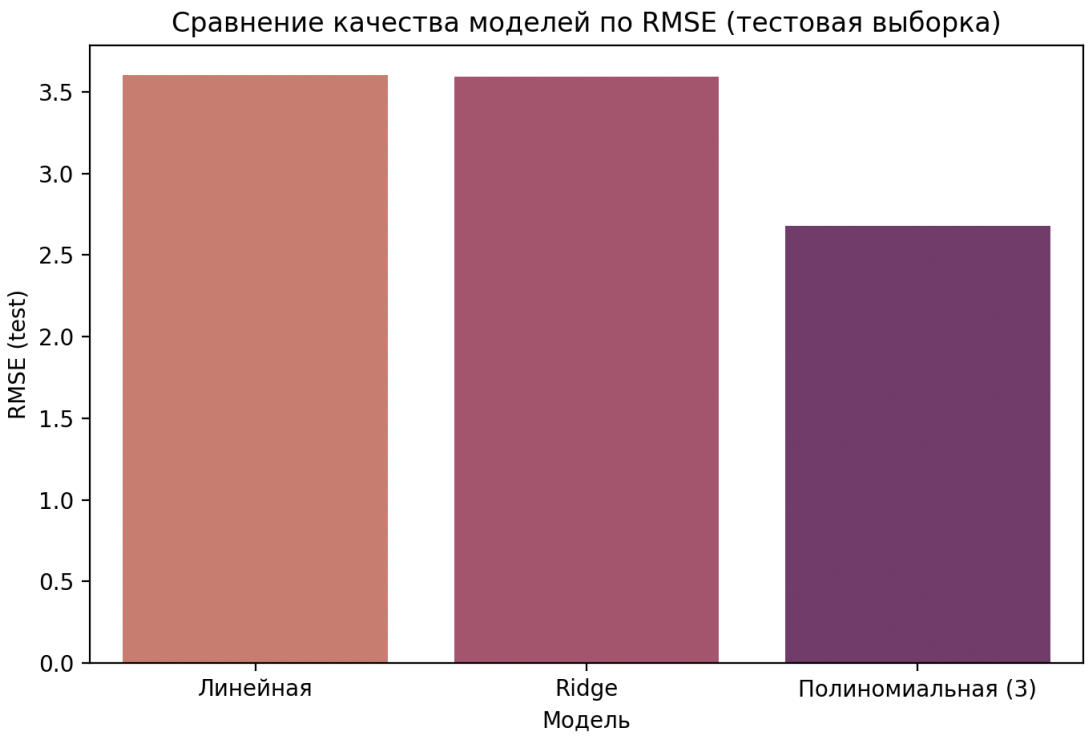
```python
# ---------------- Итоговое сравнение всех моделей --------------------

# создаем таблицу с результатами всех 3 моделей
results = pd.DataFrame({
    'Модель': ['Линейная', 'Ridge', 'Полиномиальная (3)'],
    'R^2 (train)': [r2_train_clean, r2_train_ridge, r2_train_poly],
    'R^2 (test)':  [r2_test_clean,  r2_test_ridge,  r2_test_poly],
    'RMSE (train)': [rmse_train_clean, rmse_train_ridge, rmse_train_poly],
    'RMSE (test)':  [rmse_test_clean,  rmse_test_ridge,  rmse_test_poly]
})

# выводим таблицу
print("\n --- Итоговое сравнение моделей ---")
print(results.round(3))

# R^2
# чем выше столбец, тем лучше модель объясняет вариацию данных
# (то есть точнее предсказывает цены)
plt.figure(figsize=(8, 5))
sns.barplot(data=results, x='Модель', y='R^2 (test)', hue='Модель', palette='mako', legend=False)
plt.title("Сравнение качества моделей по R^2 (тестовая выборка)")
plt.ylabel("R^2 (test)")
plt.show()

# RMSE
# чем ниже столбец, тем меньше средняя ошибка предсказания,
# то есть модель делает более точные прогнозы
plt.figure(figsize=(8, 5))
sns.barplot(data=results, x='Модель', y='RMSE (test)', hue='Модель', palette='flare', legend=False)
plt.title("Сравнение качества моделей по RMSE (тестовая выборка)")
plt.ylabel("RMSE (test)")
plt.show()
```

Результаты работы программы:

```
-- 17. Полиномиальная регрессия --
Train R^2 = 0.868, RMSE = 2.304
Test  R^2 = 0.839, RMSE = 2.676
```

```
 --- Итоговое сравнение моделей ---
               Модель  R^2 (train)  R^2 (test)  RMSE (train)  RMSE (test)
0            Линейная        0.704       0.708         3.448        3.604
1              Ridge         0.703       0.710         3.453        3.590
2  Полиномиальная (3)        0.868       0.839         2.304        2.676
```


Сравнение качества моделей по R^2 (тестовая выборка)
R^2 (test)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Линейная
Ridge
Полиномиальная (3)
Модель


Сравнение качества моделей по RMSE (тестовая выборка)
RMSE (test)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Линейная
Ridge
Полиномиальная (3)
Модель