ГУАП

КАФЕДРА № 41

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

Старший преподаватель		В.В. Боженко
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

РЕГРЕССИОННЫЙ АНАЛИЗ 2024

по курсу: ВВЕДЕНИЕ В АНАЛИЗ ДАННЫХ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4217		Д.М. Никитин
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

1. **Цель работы:** изучение алгоритмов и методов регрессии на практике.

2. **Вариант 4 и задание:**

1. Работа выполняется в https://colab.research.google.com/

2. Выполнить регрессию по вариантам (номер варианта определяется по номеру в списке группы).

Часть 1 - Простая линейная регрессия

1. Обучить модель простой (парной) линейной регрессии, используя для обучения значения x1 и y.

2. Выполнить предсказание.

3. Создать датафрейм с истинными и предсказанными значениями. Вывести его.

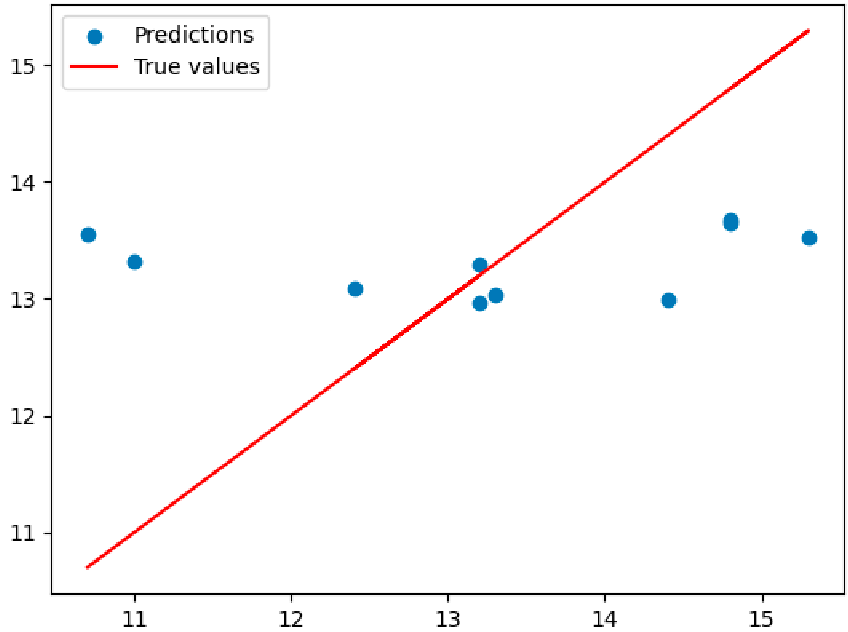
4. Подсчитать и вывести метрики качества регрессии (MSE, MAE, RMSE, R2).

5. Вывести значение коэффициентов a и b.

6. Выполнить визуализацию регрессии: точки (scatter plot) и линия регрессии.

7. Построить график с разницей предсказанного и истинного значения по каждой точке.

8. Построить график следующего вида:



9. Сделать выводы.

Часть 2 - Полиномиальная регрессия

1. Использовать PolynomialFeatures для реализации модели полиномиальной регрессии. Выбрать степень полинома самостоятельно.

2. Обучить модель полиномиальной регрессии.

3. Выполнить предсказание.

4. Подсчитать и вывести метрики качества регрессии (MAE, R2).

5. Выполнить визуализацию регрессии: точки и линия регрессия.

6. Повторить пункты 1-5 минимум для ещё одной степени полинома (degree).

7. Сделать выводы.

Часть 3 - Решение задачи регрессии различными методами

1. Загрузить набор данных car_price.csv.

2. Выделить целевую переменную, которую необходимо предсказать (важно не ошибиться с выбором целевой переменной). Выполнить для целевой переменной визуализацию - построить гистограмму и boxplot.

3. Построить матрицу диаграмм рассеяния.

4. Разделить данные на обучающую и валидационные выборки.

5. Нормализовать числовые данные с помощью StandardScaler.

6. Обучить модель линейной регрессии с помощью LinearRegression.

7. Применить обученную модель на тестовой выборке и оценить её качество с помощью метрик (минимум 4 метрики).

8. Создать датафрейм с истинными и предсказанными значениями. Вывести его.

9. Создать датафрейм с признаками и значением коэффициентов для каждого признака. Сделать выводы относительно важности признаков.

10. Выполнить визуализацию. Отобразить на графике фактическое и предсказанное значение.

11. Для получения оценки 5 - реализовать регрессию методом k-ближайших соседей или деревом решений.

12. Для метода, реализованного в пункте 11 подсчитать метрики, выполнить визуализацию фактического и предсказанного значения. Сравнить результаты, полученные всеми методами. Для этого может потребоваться визуализировать истинные и предсказанные значения на одном графике для разных методов.

13. Сделать выводы по работе. Описать, какой метод целесообразнее использовать.

3. **Ход работы:**

Часть 1 - Простая линейная регрессия

Сначала будут импортированы библиотеки, а далее выполнено задание. Реализацию можно увидеть на рисунках 1 и 2, а на рисунке 3 – результаты выполнения. Описание последует далее.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from math import sqrt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, root_mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures, StandardScaler

# x2: list[int] = [int(x) for x in "9 8 9 7 4 5 3".split()]

# Перенос данных из методических материалов
x1: list[int] = [x for x in list(map(int, "3 3 6 6 7 8 9".split()))]
y: list[float] = list(map(float, "26.5 26.4 28.2 27.6 26.9 25.2 26.6".split()))

# Перевод в массив numpy
X1: np.array = np.array(x1).reshape(-1, 1)
print("X:", X1, end="\n\n")
Y: np.array = np.array(y)
print("Y:", Y, end="\n\n")

# Обучение модели простой линейной регрессии
LinearModel: LinearRegression = LinearRegression()
LinearModel.fit(X1, Y)

# Предсказание значений
```

Рисунок 1 – Реализация начала первого этапа часть 1

```python
Y_predicted: np.array = LinearModel.predict(X1)
print(f"Предсказанные значения: {Y_predicted}")

# Создание датафрейма с реальными и предсказанными значениями
df_x1_y_predicted: pd.DataFrame = pd.DataFrame({"X": [x[0] for x in X1], "Actual": Y, "Predicted": Y_predicted}).reset_index(drop=True)
df_x1_y_predicted
```

✓ [22] 21ms

```
X: [[3]
 [3]
 [6]
 [6]
 [7]
 [8]
 [9]]

Y: [26.5 26.4 28.2 27.6 26.9 25.2 26.6]

Предсказанные значения: [26.92142857 26.92142857 26.77142857 26.77142857 26.72142857 26.67142857
 26.62142857]
```

7 rows × 3 columns

	X	Actual	Predicted
0	3	26.5	26.921429
1	3	26.4	26.921429
2	6	28.2	26.771429
3	6	27.6	26.771429
4	7	26.9	26.721429
5	8	25.2	26.671429
6	9	26.6	26.621429

Рисунок 2 – Реализация начала первого этапа часть 2 и результаты выполнения

Была создана и обучена линейная модель данных. Далее был создан датафрейм с реальными значениями и предсказанными и был выведен на экран. Для лучшего понимания был также добавлен столбец X. Можно заметить, что в некоторых точках значения совпадают, но в некоторых довольно сильно расходятся. Это наблюдается из-за усреднения результата регрессией.

Далее будут выведены между реальными и предсказанными данными

MSE (Mean Squared Error) - среднеквадратическая ошибка, её работа похожа на работу сигма квадратэто мера разброса ошибок предсказаний модели относительно истинных значений (другой выборки, которая представляет реальность).

MAE (Mean Absolute Error) - средняя абсолютная ошибка, её работа похожа на работу S, это средняя абсолютная разница между предсказаниями модели и истинными значениями.

RMSE (Root Mean Squared Error) - корень из среднеквадратической ошибки работа похожа на работу сигмы, используется для измерения среднеквадратичной ошибки модели — то есть, насколько в среднем предсказания отклоняются от реальных значений.

R2 (Coefficient of Determination) - коэффициент детерминации его работа похожа на работу дисперсии, характеризует разброс данных самих по себе.

Значения метрик можно увидеть на рисунке 4.

```python
# Вычисление метрик
mse: np.float64 = mean_squared_error(df_x1_y_predicted.Actual, df_x1_y_predicted.Predicted)
mae: np.float64 = mean_absolute_error(df_x1_y_predicted.Actual, df_x1_y_predicted.Predicted)
rmse: np.float64 = np.float64(sqrt(mse))
r2: np.float64 = r2_score(df_x1_y_predicted.Actual, df_x1_y_predicted.Predicted)

# Вывод метрик
print(f"MSE (среднеквадратическая ошибка): {mse}")
print(f"RMSE (корень из среднеквадратической ошибки): {rmse}")
print(f"MAE (средняя абсолютная ошибка): {mae}")
print(f"R2 (коэффициент детерминации): {r2}")
```
✓ [23] 11ms
```
MSE (среднеквадратическая ошибка): 0.7677551020408181
RMSE (корень из среднеквадратической ошибки): 0.876216355725467
MAE (средняя абсолютная ошибка): 0.6959183673469391
R2 (коэффициент детерминации): 0.014667365112622677
```

Рисунок 4 – Значения метрик для первого этапа

Значение MSE, RMSE и MAE являются довольно высокими. Это означает, что предсказания модели имеют невысокую точность и допускает значительные ошибки. Значение R^2 близится к нулю, это означает, что модель может объяснить только 1.47% вариативности, остальные 98+% остаются необъяснёнными, иными словами, модель предсказывает также плохо, как если бы она брала среднее значение.

Соответственно, по этим метрикам можно сказать, что модель очень неточна. Для улучшения результата можно попробовать взять другую модель, взять другие данные для обучения (отсутствуют).

Далее будут выведены значения коэффициентов a и b. Их можно увидеть на рисунке 5.

```python
a = LinearModel.coef_[0]
b = LinearModel.intercept_
print(f"a = {a}\nb = {b}")
```
✓ [24] 12ms
```
a = -0.04999999999999836
b = 27.07142857142857
```

Рисунок 5 – Коэффициенты a и b

По коэффициентам ожидается падающий график с пересечением оси y в точке b.

Далее будет выполнена визуализация регрессии. Можно увидеть на рисунке 6.

```
# Построение графика
plt.figure(figsize=(8, 5))
plt.title("Предсказанные и реальные значения с помощью линейной регрессии")
plt.scatter(df_x1_y_predicted.X, df_x1_y_predicted.Actual, label="Истиные значения", color="olive")
plt.plot(df_x1_y_predicted.X, df_x1_y_predicted.Predicted, label="Линия регрессии", linewidth=2, color="red")
plt.xlabel('X1')
plt.ylabel('Y')
plt.legend()
plt.show()
```

✓ [43] 208ms

Рисунок 6 – Линейная регрессия этапа 1

По графику также видно, что график недостаточно точно отображает истинные значения. Некоторые точки находятся очень далеко от линии регрессии.

Далее будет построен график разницы, её можно увидеть на рисунках 7 и 8.

```python
# Вычисление разницы
df_x1_y_predicted['Difference'] = (df_x1_y_predicted.Actual - df_x1_y_predicted.Predicted)
print(df_x1_y_predicted)


# Построение графика
plt.figure(figsize=(8, 6))
plt.bar(range(len(df_x1_y_predicted.Difference)), df_x1_y_predicted.Difference, color="green")
plt.title("Разности предсказанных и истинных значений")
plt.xlabel('Номер')
plt.ylabel('Отклонение предсказания от истинного значения')
plt.axhline(y=0, color='black', linestyle='--')
plt.show()
```

✓ [26] 183ms

```
   X  Actual  Predicted  Difference
0  3    26.5  26.921429   -0.421429
1  3    26.4  26.921429   -0.521429
2  6    28.2  26.771429    1.428571
3  6    27.6  26.771429    0.828571
4  7    26.9  26.721429    0.178571
5  8    25.2  26.671429   -1.471429
6  9    26.6  26.621429   -0.021429
```

Рисунок 7 – Реализация визуализации разницы

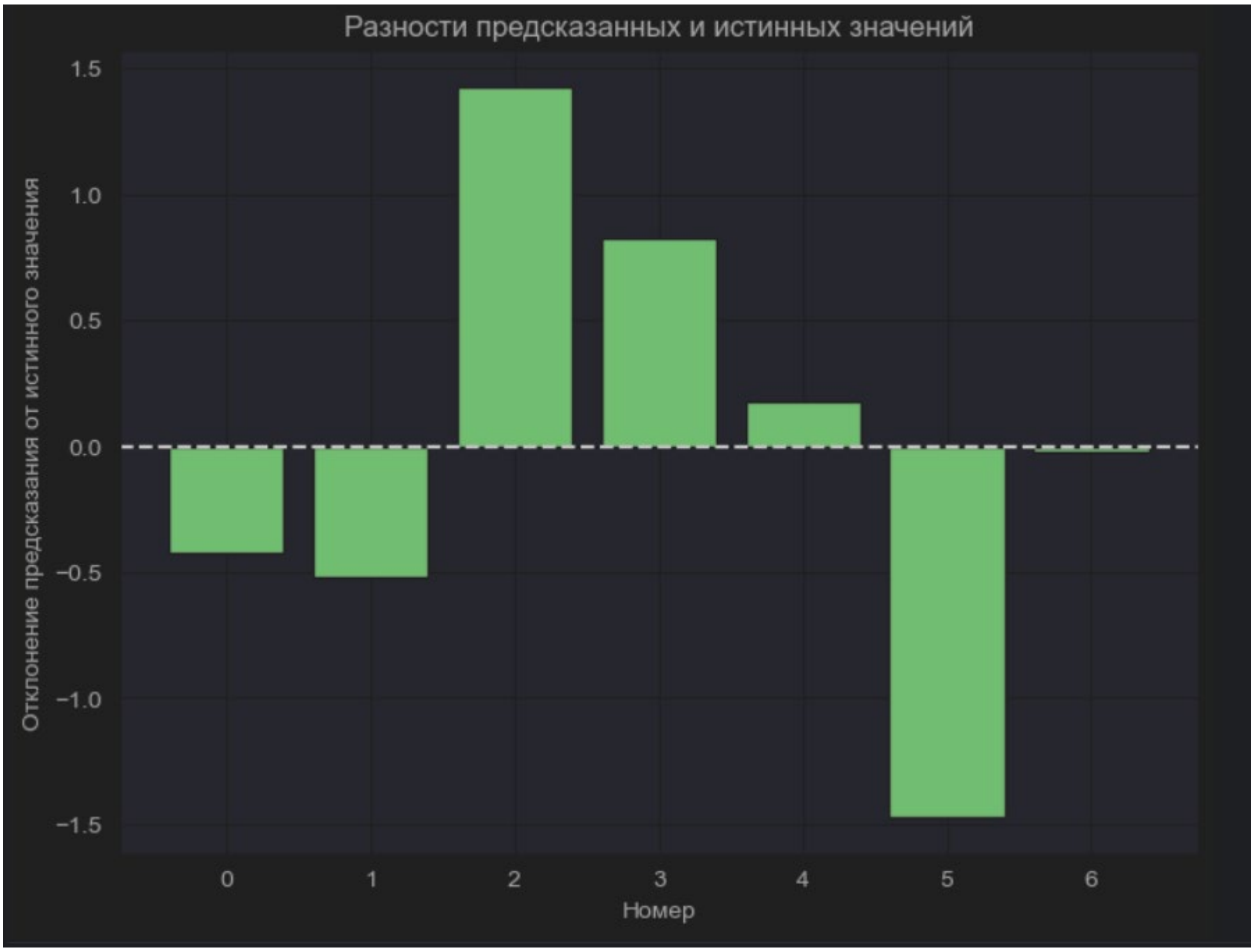


Рисунок 8 – Визуализация разницы

На графике видны отклонения больше, чем в 1 от реальных значений. Это достаточно большая ошибка. Далее будет построен график реальных значений против предсказанных. Его можно увидеть на рисунке 9.

```python
# Построение графика
plt.figure(figsize=(8, 5))
plt.title("Истинные значения и предсказания")
plt.scatter(df_x1_y_predicted.Actual, df_x1_y_predicted.Predicted, label='Предсказанные значения', color='blue')
plt.plot([min(df_x1_y_predicted.Actual), max(df_x1_y_predicted.Actual)], [min(df_x1_y_predicted.Actual), max(df_x1_y_predicted.Actual)], color='red', label='Истинные значения')

plt.show()
```

✓ [45] 186ms

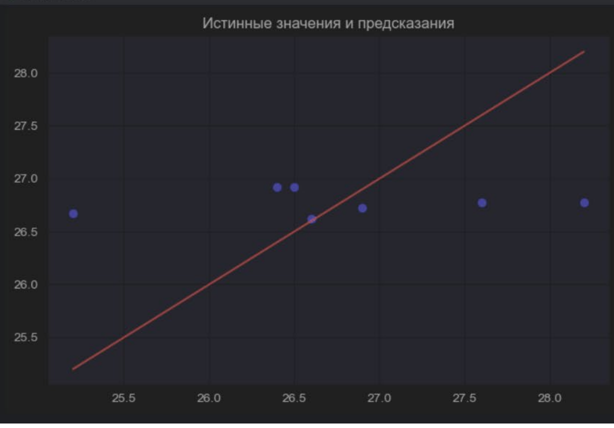


Рисунок 9 – График реальных против предсказанных значений

Значения отходят от линии очень далеко и практически не следуют за ней. Это значит, что модель предсказывает значения только в общем и целом. Точность прогнозирования очень низкая.

Часть 2 - Полиномиальная регрессия

Далее будет обучена модель полиномиальной регрессии, выполнено предсказание, получены метрики регрессии и выведен её график для регрессий 2 и 3 степени. Это всё можно увидеть на рисунках 10, 11, 12 и 13.

```python
# Функция для полиномиальной регрессии
def make_regression_great_again(x: np.array, y: np.array, degree: int) -> None:

    # Создание полиномиальных значений
    poly_features: PolynomialFeatures = PolynomialFeatures(degree=degree)
    x_poly: np.ndarray[int] = poly_features.fit_transform(x)

    # Создание модели
    poly_linear_model: LinearRegression = LinearRegression()

    # Обучение модели полиномиальной регрессии
    poly_linear_model.fit(x_poly, y)

    # Предсказание
    y_poly: np.ndarray[float] = poly_linear_model.predict(x_poly)


    # Получение и вывод MAE и R2
    mae: float = mean_absolute_error(y, y_poly)
    r2: float = r2_score(y, y_poly)
    print(f"MAE: {mae}")
    print(f"R2: {r2}")

    # Построение графика
    plt.figure(figsize=(8, 6))
    plt.title(f"Истинные и предсказанные значения с помощью полиномиальной регрессии {degree} степени")
    plt.scatter(x, y, label="Истиные значения", color="olive")
    plt.plot(x, y_poly, label="Линия регрессии", linewidth=2, color="red")
    plt.xlabel('X')
    plt.ylabel('Y')
```

Рисунок 10 – Часть реализации регрессии этапа 2

```python
    plt.legend()
    plt.show()


new_x = np.array([int(x) for x in "0 1 2 3 4 5 6".split()]).reshape(-1, 1)
new_y: list[float] = list(map(float, "25.2 46.2 56.7 77.6 91.5 112.3 106.2".split()))
make_regression_great_again(new_x, new_y, 2)
make_regression_great_again(new_x, new_y, 3)

✓ [28] 400ms
```

Рисунок 11 – Вторая часть реализации регрессии этапа 2

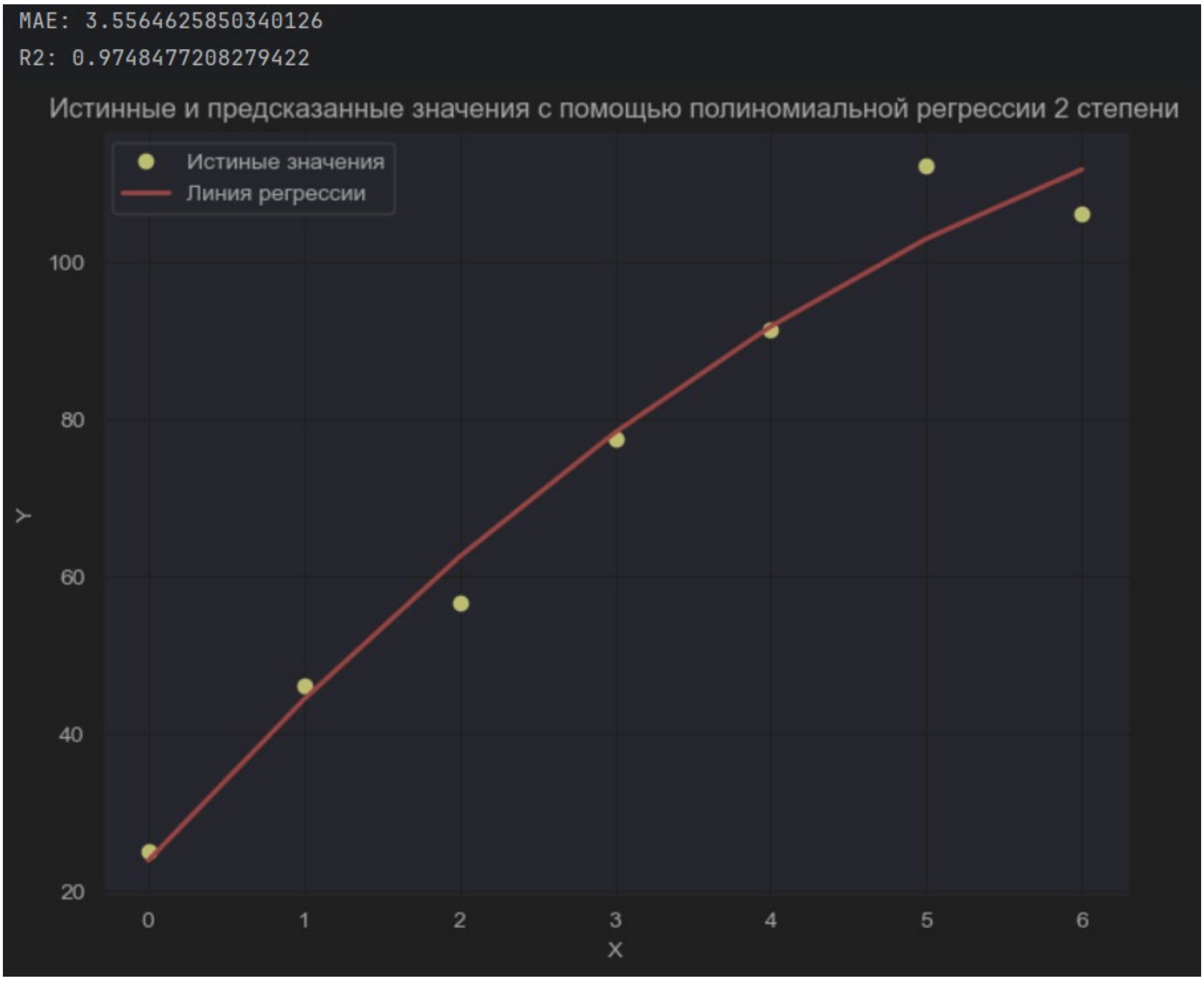


Рисунок 12 – Результат предсказания полинома 2 степени

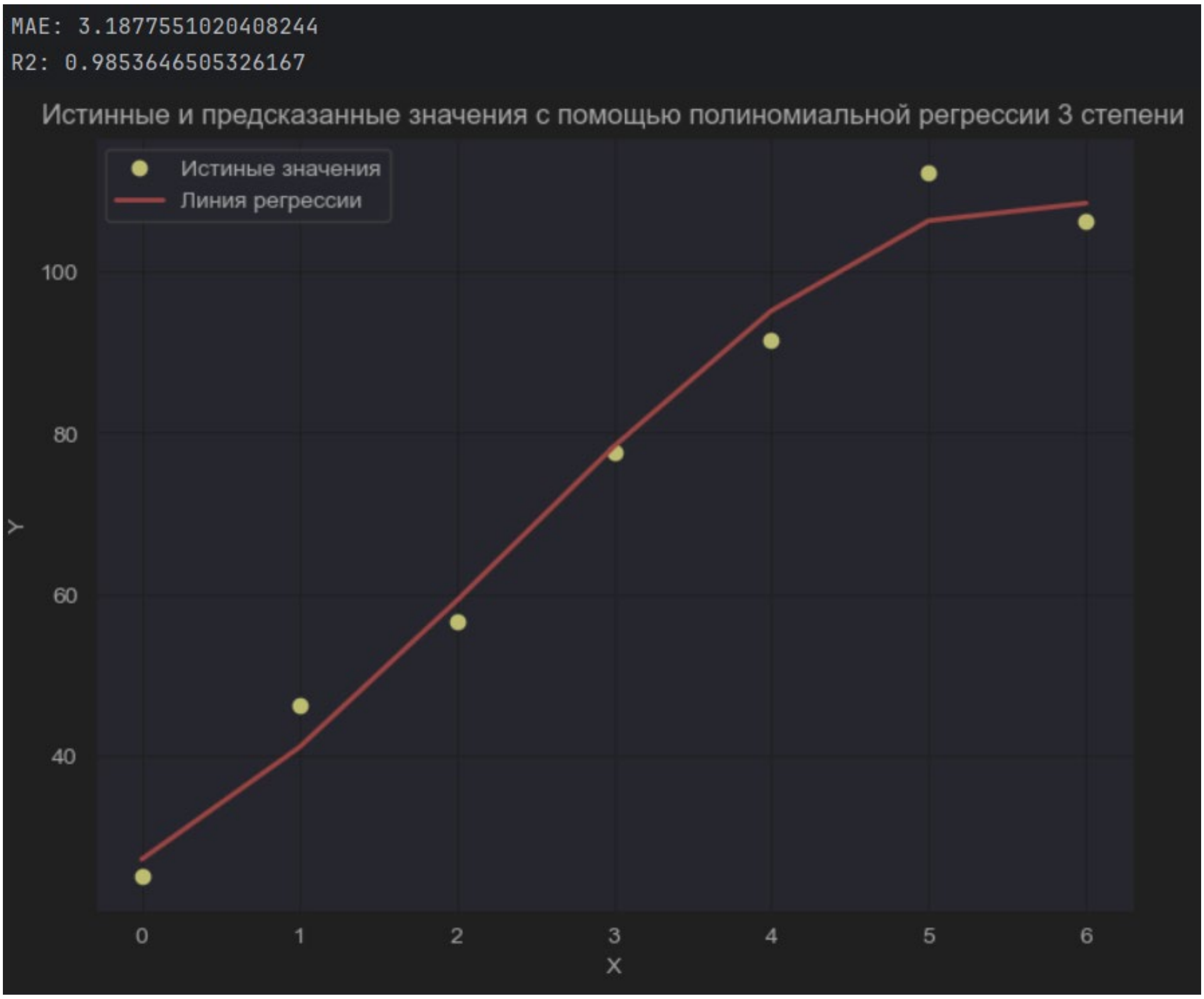


Рисунок 13 – Результат предсказания полинома 3 степени

Данные виды регрессий объясняют значения намного лучше, чем линейная. Это не удивительно, потому что полиномиальная регрессия является расширенной версией линейной. Для этих данных этот метод отлично подходит и прекрасно описывает данные, понять это можно, если посмотреть на значения MAE и R2. Значение MAE около 3, для таких больших значений это значение ошибки очень хорошо. Также R^2 стремится к единице, это говорит о высокой точности предсказания. Также стоит заметить, что при повышении степени точность предсказания увеличиваться. Однако не стоит забывать, что степени полинома показывают не точность, а количество изломов, которые хотелось бы увидеть на графике. Поэтому степень нужно выбирать по форме распределения точек на графике. Например, на этом графике хорошо работала бы даже обыкновенная линейная регрессия, хотя вторая и третья степени также улучшают точность из-за неровности расположения точек.

Часть 3 - Решение задачи регрессии различными методами

Сначала был открыт файл car_price.csv. Результат можно увидеть на рисунке 14.

```
car_price = pd.read_csv("car_price.csv")
car_price
```

[29] 24ms

1-10 205 rows × 26 columns

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front
5	6	2	audi fox	gas	std	two	sedan	fwd	front
6	7	1	audi 100ls	gas	std	four	sedan	fwd	front
7	8	1	audi 5000	gas	std	four	wagon	fwd	front
8	9	1	audi 4000	gas	turbo	four	sedan	fwd	front
9	10	0	audi 5000s (diesel)	gas	turbo	two	hatchback	4wd	front

Рисунок 14 – Открытие датафрейма

Далее он бы проверен на наличие явных дубликатов и пропусков. Результат на рисунке 15.

```
print("Дубликатов:", car_price.duplicated().sum())
print("Пропусков:", car_price.isna().sum().sum())
car_price.info()    car_price
✓ [30] 13ms
```

```
Дубликатов: 0
Пропусков: 0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
```

Рисунок 15 – Проверка на пропуски и явные дубликаты

Явных дубликатов и пропусков в данных нет, данные соответствуют типам. В таком случае можно предположить, что неявных дубликатов также нет, так как задание строится не на этом, а также это учебные данные.

В качестве целевой переменной была выбрана цена на автомобиль. Так как зачастую именно она является решающим фактором покупки или не покупки автомобиля. Далее были построены диаграмма и boxplot. Их можно увидеть на рисунках 16, 17 и 18.

```python
plt.figure(figsize=(8, 6))
plt.title("Гистограмма цен на автомобили")
plt.xlabel("Цена")
plt.ylabel("Количество")
plt.hist(car_price.price, color="green", edgecolor="black", linewidth=2, bins=15)
plt.show()

plt.figure(figsize=(8, 6))
plt.title("Boxplot диаграмма цен на автомобили")
plt.ylabel("Цена")
plt.boxplot(car_price.price)
plt.show()
✓ [31] 330ms
```

Рисунок 16 – Реализация создания графиков

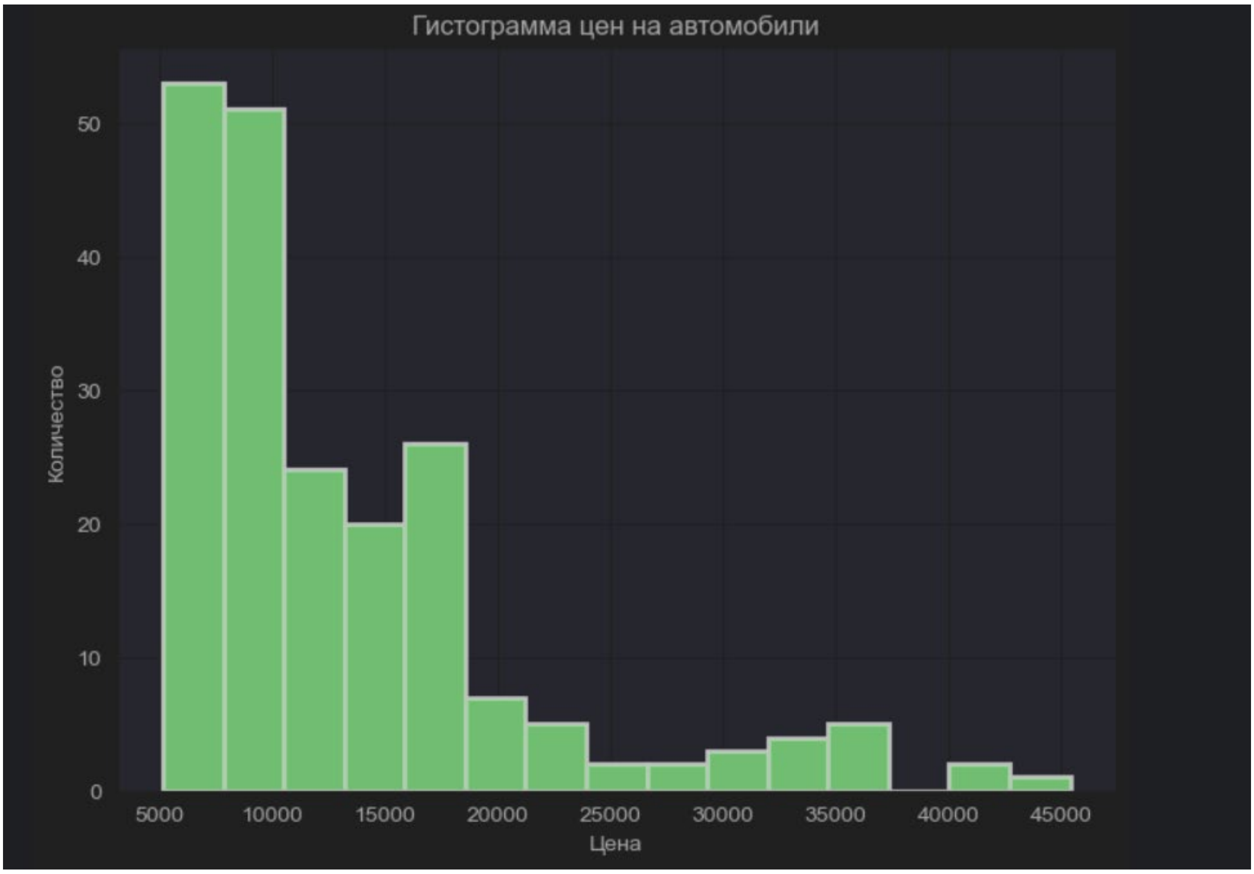


Рисунок 17 – Гистограмма цен на автомобили

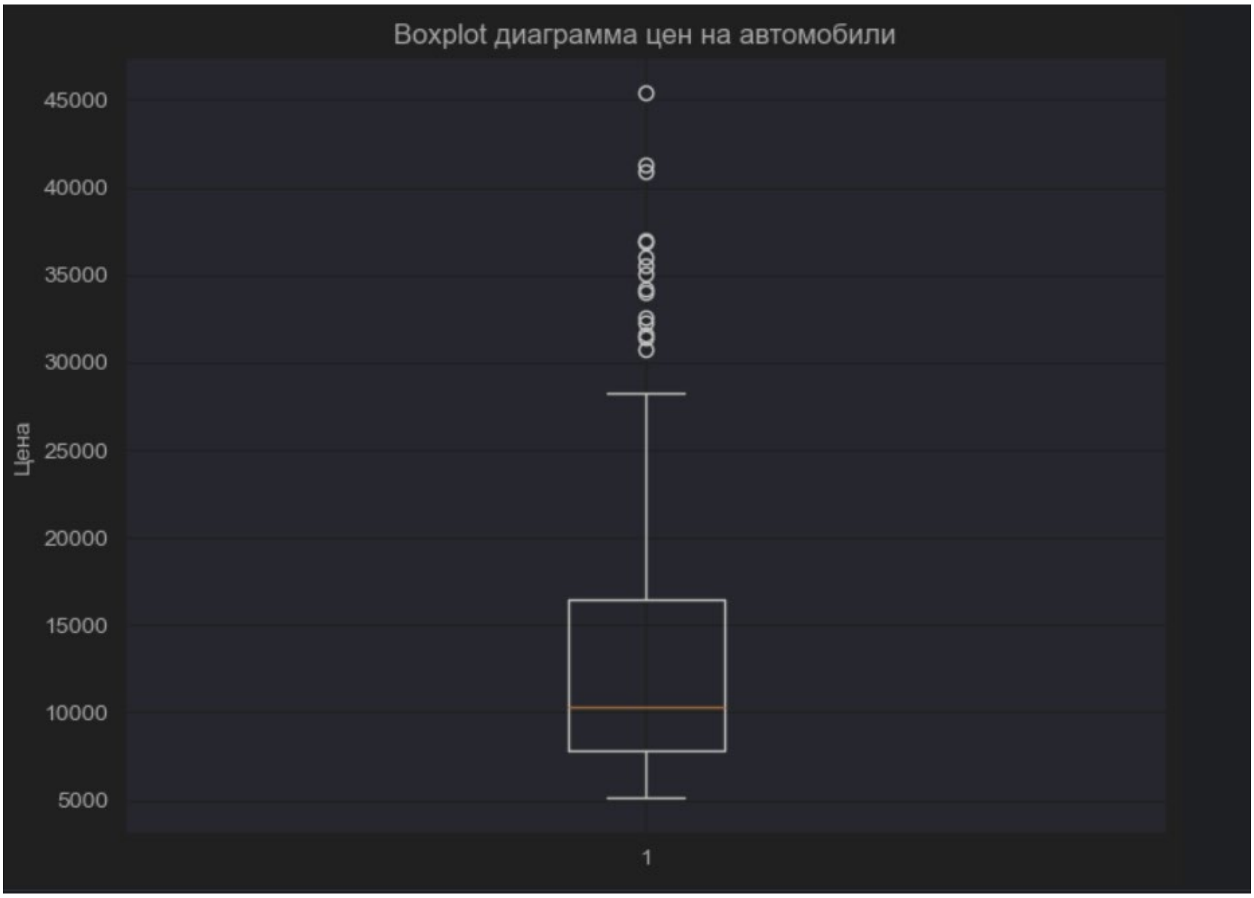


Рисунок 18 – Диаграмма boxplot цен на автомобили

По данным диаграммам заметно, что значительное количество значений смещено в сторону самых дешёвых. Следовательно, большинство автомобилей в выборке бюджетные. Построим матрицу диаграмм рассеяния.

Матрицу и реализацию можно увидеть на рисунке 19 и 20.

```
columns_research: list[str] = ["citympg", "highwaympg", "enginesize", "price"]

plt.figure(figsize=(6, 6))
sns.pairplot(car_price[columns_research])
plt.savefig('pairplot.png', format='png')
plt.show()
✓ [49] 4s 369ms
```

Рисунок 19 – Реализация матрицы рассеяния

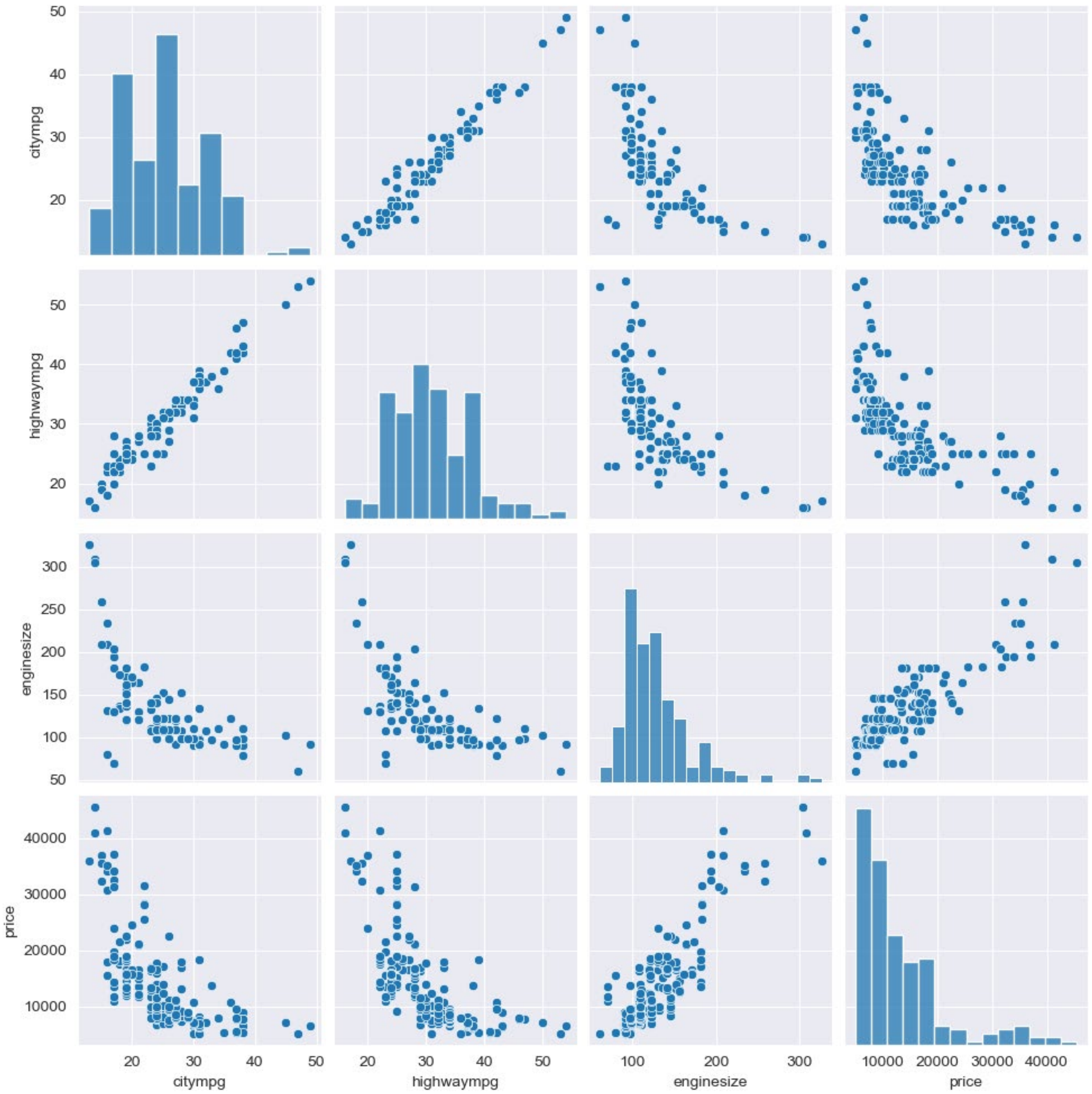


Рисунок 20 – Матрица рассеяния

Между каждой из этих характеристик автомобиля имеется зависимость. Из полученной матрицы можно сделать вывод. Цена снижается от большого расхода по шоссе или городу, а также от низкого объёма двигателя. Второе происходит вследствие того, что на дешёвые автомобили обычно снабжают двигателями с низким объёмом, а дорогие - с большим. Также заметна зависимость между объёмом двигателя и расхода автомобиля. Всё это из-за того, что пробег на двигателе с большим объёмом значительно сильнее бьёт по карману.

Далее для исследования и разделения на валидационную и обучающую выборки были выбраны столбцы:

horsepower: Мощность в лошадиных силах,

enginesize: объём двигателя,

carlength: Длина машины,

citympg: расход по городу,

highwaympg: расход по шоссе.

Далее был произведён выбор данных. Его можно увидеть на рисунке 21.

```
features = car_price[['horsepower', 'enginesize', 'carlength', 'citympg', 'highwaympg']]
target_values = car_price['price']
print(features.info())
features
✓ [33] 21ms
RangeIndex: 205 entries, 0 to 204
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   horsepower  205 non-null    int64
 1   enginesize  205 non-null    int64
 2   carlength   205 non-null    float64
 3   citympg     205 non-null    int64
 4   highwaympg  205 non-null    int64
dtypes: float64(1), int64(4)
memory usage: 8.1 KB
None
```

1-10 205 rows × 5 columns

	horsepower	enginesize	carlength	citympg	highwaympg
0	111	130	168.8	21	27
1	111	130	168.8	21	27
2	154	152	171.2	19	26
3	102	109	176.6	24	30
4	115	136	176.6	18	22
5	110	136	177.3	19	25
6	110	136	192.7	19	25

Рисунок 21 – Выбор данных

Данные были выбраны. Теперь приступим к резделению данных. Это можно увидеть на рисунке 22.

```
x_training, x_check, y_training, y_check = train_test_split(features, target_values, random_state=0, test_size=0.25)

print(x_training.shape)
print(y_training.shape)
print(x_check.shape)
print(y_check.shape)
✓ [34] 22ms
(153, 5)
(153,)
(52, 5)
(52,)
```

Рисунок 22 – Разделение данных

Данные были разделены на тренировочные и валидационные в пропорции 75/25.

Далее данные будут нормализованы с помощью StandardScaler. Обратите внимание, что y не нормализуют, потому что целевая переменная может быть в разных единицах (например, цена в долларах, килограммах, времени и т.д.), и изменение её масштаба может исказить интерпретацию модели. Реализация на рисунке 23.

```
scaler = StandardScaler()

x_training_scaled = scaler.fit_transform(x_training)
x_checking_scaled = scaler.transform(x_check)
✓ [35] 10ms
```

Рисунок 23 – Реализация нормализации

Данные были нормализованы, далее будет проведено обучение модели линейной регрессии. На рисунке 24.

```
CarLinearModel: LinearRegression = LinearRegression()
CarLinearModel.fit(x_training_scaled, y_training)
✓ [36] < 10 ms

LinearRegression()
```

Рисунок 24 – Обучение модели

Далее будет проведена проверка точности прогнозирования модели не менее, чем по 4 метрикам. На рисунке 25.

```
y_predicted = CarLinearModel.predict(x_checking_scaled)
mae: float = mean_absolute_error(y_check, y_predicted)
mse: np.float64 = mean_squared_error(y_check, y_predicted)
rmse: np.float64 = np.float64(sqrt(mse))
r2: np.float64 = r2_score(y_check, y_predicted)
print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2: {r2}")
print(max(target_values), sum(target_values)/len(target_values))
✓ [37] < 10 ms
```

```
MAE: 2773.6634966448287
MSE: 13685489.245559555
RMSE: 3699.3903883693533
R2: 0.816469740345569
45400.0 13276.710570731706
```

Рисунок 25 – Проверка точности

Данные метрики показывают, что в этот раз модель работает достаточно хорошо. При предсказании цены модель ошибается в среднем для каждого автомобиля на 2773, при квадратичном подходе модель ошибается на 3699, что при максимальной цене автомобиля в 45400 и средней в 13276 является хорошим показателем. Также значение R^2 говорит нам о том, что модель объясняет около 81% всех данных, что является также высоким показателем. Модель имеет такие характеристики, так как была обучена на достаточно большом объёме данных.

Далее будет создан датафрейм с истинными и предсказанными значениями. Это показано на рисунке 26.

```
car_df: pd.DataFrame = pd.DataFrame({"Истинные": target_values, "Предсказанные": CarLinearModel.predict(scaler.transform(features))}).reset_index(drop=True)
car_df
```

✓ [38] 14ms

1-10 205 rows × 2 columns

	Истинные	Предсказанные
0	13495.000	13288.611791
1	16500.000	13288.611791
2	16500.000	18421.095359
3	13950.000	11591.911123
4	17450.000	14912.267735
5	15250.000	14632.474724
6	17710.000	16231.675582
7	18920.000	16231.675582
8	23875.000	17804.464345
9	17859.167	17586.208178

Рисунок 26 – Создание датафрейма с нужными значениями

Истинные и предсказанные значения по всему датафрему были помещены в новый датафрейм. Далее будет выполнено получение датафрейма с коэффициентами. Оно показано на рисунке 27.

```
# Извлечение коэффициентов
coefficients = CarLinearModel.coef_

# Создание датафрейма с признаками и коэффициентами
coeff_df = pd.DataFrame({
    'Признак': features.columns,
    'Коэффициент': coefficients
}).sort_values(by="Коэффициент", ascending=False).reset_index(drop=True)
coeff_df
```

✓ [39] 14ms

5 rows 5 rows × 2 columns

	Признак	Коэффициент
0	enginesize	3809.969625
1	horsepower	2573.987040
2	carlength	1252.514949
3	citympg	173.898452
4	highwaympg	-97.201600

Рисунок 27 – Получение датафрейма с коэффициентами

Данные коэффициенты нам говорят о том, что самыми важными признаками при формировании цены из представленных является объём двигателя и его мощность. За ними следует длина автомобиля. Наименее влияющим на цену в положительную сторону является расход топлива в городе и отрицательно на рост цены влияет расход топлива на шоссе.

Это можно объяснить тем, что для повышения мощности двигателя можно увеличить его объём, таким образом мощность двигателя и его объём влияют больше всего, так как значительно увеличивают расход топлива. Поэтому их ставят в дорогие автомобили для богатых. Длина автомобиля также является признаком премиальности автомобиля, а также увеличивает его цену за счёт большего расхода материалов на его создание. В свою очередь увеличение расхода в городе практически не влечёт повышения цены автомобиля, потому что на него придётся много тратить, но всё равно имеет положительный показатель из-за того, что премиальные автомобили с мощными двигателями имеют большой расход. А увеличение расхода на трассе вовсе удешевляет автомобиль по понятным причинам. Далее полученные данные будут выведены на график. Это можно увидеть на рисунке 28.

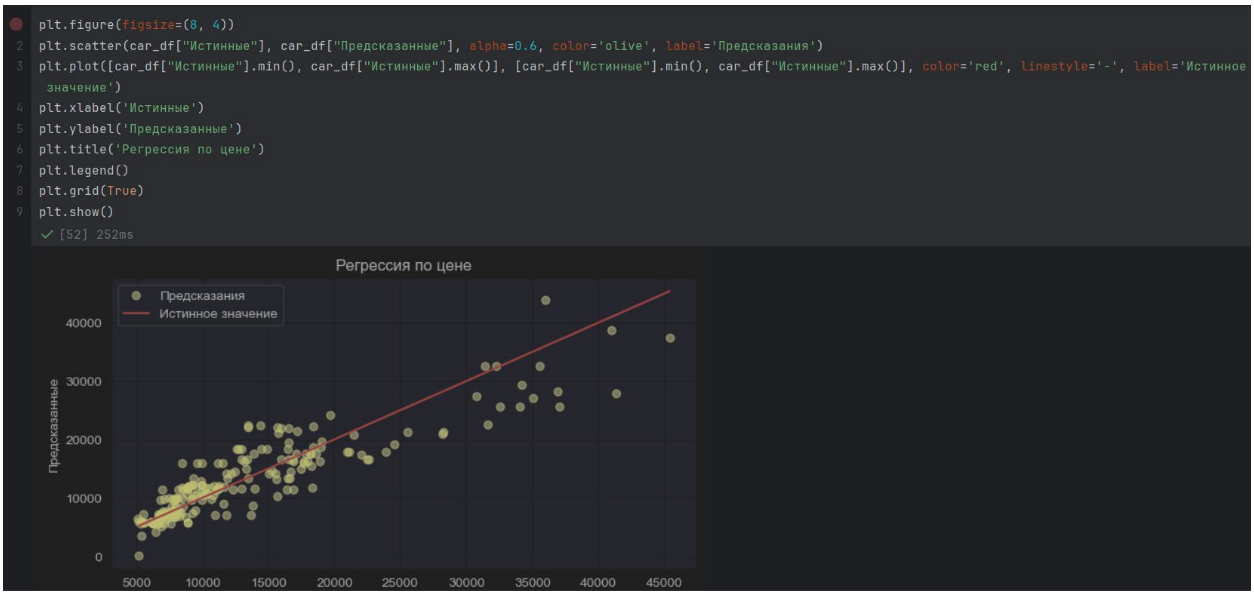


Рисунок 28 – Перенос данных на график

Данный график снова указывает на точность регрессии. Предсказанные значения довольно близко находятся к истинным значениям по заданным параметрам.

Для получения оценки 5

Реализация регрессии методом k-ближайших соседей или деревом решений. Её можно увидеть на рисунках 29 и 30.

```python
# Будут проведены необходимые действия для работы с данными заново, чтобы не запутаться.
# Разделение данных на признаки (X) и целевую переменную (y)
X = car_price[['horsepower', 'enginesize', 'carlength', 'citympg', 'highwaympg']]
y = car_price['price']

# Разделение на обучающие и тестовые выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)

# Масштабирование данных
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Создание и обучение модели KNN (k=2 - число ближайших соседей)
knn_model = KNeighborsRegressor(n_neighbors=2)
knn_model.fit(X_train_scaled, y_train)

# Прогнозирование на тестовых данных
y_pred = knn_model.predict(X_test_scaled)

# Оценка модели
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = root_mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
print(f"R2: {r2}")
```

Рисунок 29 – Реализация метода k-соседей

```
MAE: 1698.125
MSE: 7048818.216346154
RMSE: 2654.9610574067096
R2: 0.9054713050961885
```

Рисунок 30 – Результат выполнения

Эти значения были получены при линейной регрессии:

MAE: 2773.6634966448287

MSE: 13685489.245559555

RMSE: 3699.3903883693533

R2: 0.816469740345569.

Можно заметить, что MAE, MSE и RMSE меньше у регрессии k-соседей, а R2 наоборот больше. Это означает, что данные регрессия методом k-соседей более точна и допускает меньше ошибок, чем линейная. Далее полученные данные будут визуализированы на графике. Это можно увидеть на рисунке 31.

Рисунок 31 – Сравнение линейной регрессии и метода k-соседей при значении 2

Этот график также подтверждает, что метод k-соседей при значении 5 более эффективен, чем метод линейной регрессии. Нередко предсказанные значения находятся почти на линии реальных значений, также большинство предсказанных значений лежит значительно ближе к оси истинных значений, чем предсказание линейной регрессии. Однако метод k-соседей также имеет и недостатки по отношению к линейной регрессии: метод k-соседей теряет эффективность при высокой размерности, необходима правильно выбирать параметр k, который значительно влияет на производительность и точность регрессии, а также модель теряет эффективность при высоком значении k. В нашём случае значительно лучше подходит регрессия методом k-соседей, при этом k = 2, однако при увеличении размерности стоило бы перейти на линейную регрессию из-за снижения эффективности. Также модель k-соседей было бы лучше использовать и при нелинейности распределения данных, так как линейная регрессия лучше работает с линейными данными. То есть, использование одной из данных моделей целесообразно в зависимости от конкретного случая.

4. **Ссылка на Jupiter Notebook:**

https://colab.research.google.com/drive/1vSdkaYXS9TOVOfyuKykYhvDS8RxZB6Z4?usp=sharing

5. **Вывод:**

Выполненная работа охватывает несколько аспектов задачи регрессии на различных этапах. На первом этапе была проведена простая линейная регрессия, где использовались значения x1 и y. В процессе обучения модели была создана таблица с истинными и предсказанными значениями, а также вычислены метрики качества модели, такие как MSE, RMSE, MAE и R^2. На основе этих показателей было сделано заключение о низкой точности модели, так как значения ошибок оказались достаточно высокими, а коэффициент детерминации близким к нулю. Для улучшения результатов можно рассматривать другие модели.

Во втором этапе была выполнена полиномиальная регрессия с использованием различных степеней полинома. Результаты показали, что полиномиальная регрессия с более высокими степенями значительно улучшает точность предсказаний по сравнению с линейной регрессией, что продемонстрировано через снижение значения MAE и увеличение R^2. Однако выбор степени полинома должен зависеть от распределения данных, так как слишком высокая степень может привести к переобучению.

На третьем этапе была использована модель регрессии для предсказания цен на автомобили. В процессе были исследованы данные, выбрана целевая переменная (цена автомобиля), выполнены визуализации (гистограммы, boxplot) и построена матрица диаграмм рассеяния. Также данные были разделены на обучающую и валидационную выборки, нормализованы и использованы для обучения модели линейной регрессии. Метрики качества модели были вычислены, и на основе полученных результатов сделаны выводы о возможных способах улучшения точности предсказаний.

В конце было выполнено задание на 5. Метод k-соседей оказался эффективным инструментом для предсказания цен на автомобили, однако для улучшения производительности при работе с большими наборами данных можно было бы рассмотреть использование более сложных методов или оптимизацию гиперпараметров модели.