Занятие 2

Градиентный спуск и обработка данных

Идеальный рецепт лимонада

- Мы – начинающие предприниматели и хотим создать **идеальный лимонад**. Отзывы людей показали, что вкус лимонада зависит от **трёх параметров.** Наша задача найти такие x1, x2 и x3, чтобы предугадывать оценки пользователей с наибольшей точностью.

Признак	***Обозначение***	***Диапазон***
Лимонный сок (мл)	X1	20–70 мл
Сахар (г)	X2	5–50 г
Газированность (баллы)	X3	0–10 (насколько он шипит)

Ход работы

- Загрузите датасет с 100000 данными по ссылке: https://clck.ru/3N9z4B
- Откройте Google Collaboratory по ссылке: https://clck.ru/3N9ypd
- Сохраните себе на диск файл Google Collaboratory

Задание 1. Импорт библиотек и загрузка датасета

```
[2]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     from google.colab import files
```

```
[3]  # Загружаем датасет
     uploaded = files.upload()

     df = pd.read_csv("lemonade_3d_dataset.csv")
     df.head()
```

Выбрать файлы lemonade…et (1).csv

- **lemonade_3d_dataset (1).csv**(text/csv) - 2150520 bytes, last modified: 17.07.2025 - 100% done

```
Saving lemonade_3d_dataset (1).csv to lemonade_3d_dataset (1).csv
```

	lemon_juice_ml	sugar_g	fizziness	rating
0	38.73	31.14	2.83	6.80
1	67.54	28.71	4.59	8.27
2	56.60	20.80	0.99	5.87
3	49.93	27.19	4.47	8.14
4	27.80	21.43	2.03	5.22

Задание 2. Подготовка данных

```
X1 = df["lemon_juice_ml"].values  # x1 - лимонный сок
X2 = df["название"].values         # x2 - сахар
X3 = df["название"].values       # x3 - газированность
Y = df["название"].values           # y - рейтинг
n = len(Y)                        # n - количество примеров
```

Задание 3. Модель линейной регрессии и функция ошибки

Модель представляет собой **линейную комбинацию признаков**:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + b$$

где:

- $\hat{y}_i$ — предсказанное значение оценки;
- x_{i1}, x_{i2}, x_{i3} — признаки i-го примера;
- w_1, w_2, w_3 — веса модели;
- b — **bias** (свободный член), позволяющий сдвинуть гиперплоскость;
- $i = 1, \ldots, n$, где n — число наблюдений.

Задание 3. Модель линейной регрессии и функция ошибки

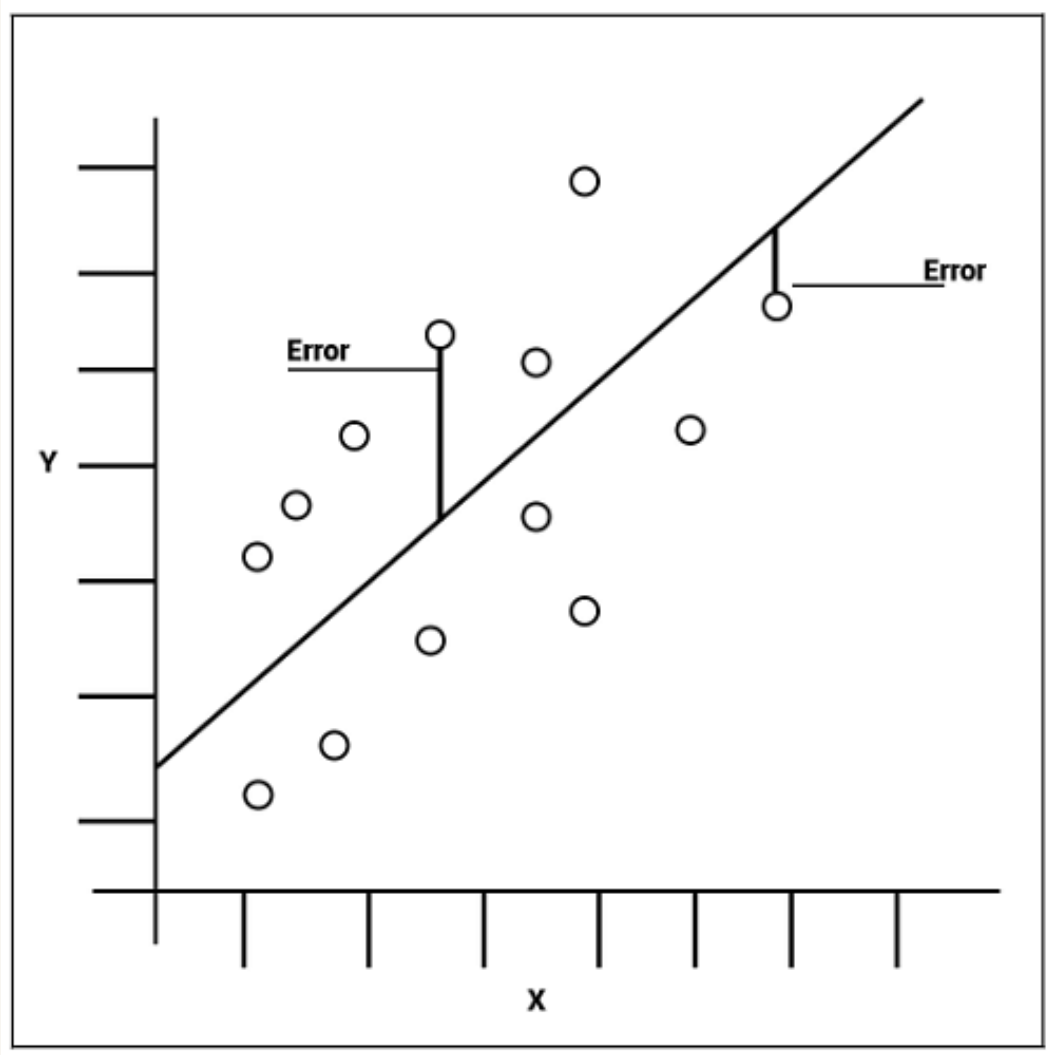
Чтобы оценить, насколько предсказания модели отличаются от реальных оценок, используем **MSE**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Чем меньше MSE, тем лучше предсказания.
- Наша задача — **минимизировать эту ошибку**.

- y_i — реальное значение,
- $\hat{y}_i$ — предсказанное значение,
- n — количество примеров.

Задание 3. Модель линейной регрессии и функция ошибки



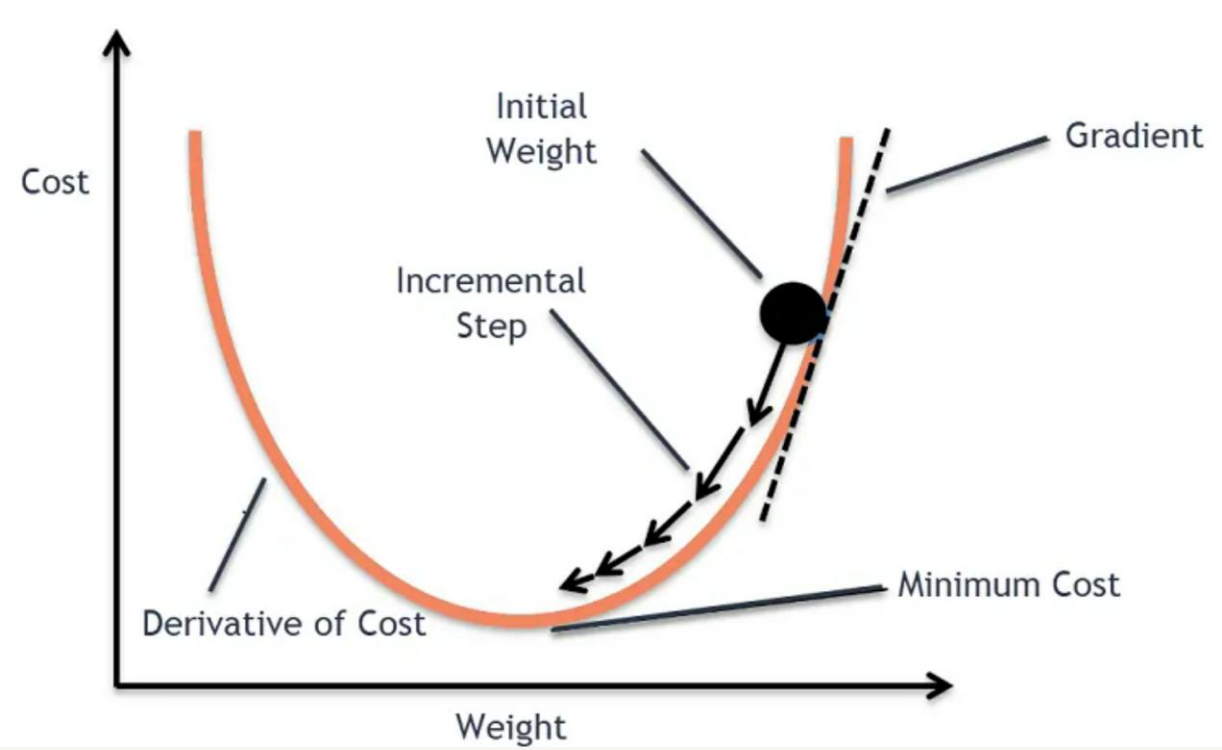
Задание 3. Модель линейной регрессии и функция ошибки

```
def mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2))
```

- Вычисляется разность между каждым истинным и предсказанным значением.
- Эта разность возводится в квадрат, чтобы:
  - убрать знак (ошибки с разными знаками не сокращались);
  - **сильнее "штрафовать" большие ошибки**.

Задание 4. Градиентный спуск

- Чтобы найти такие параметры w1,w2,w3,b, при которых ошибка минимальна, используем **градиентный спуск**.



Задание 4. Градиент

- Градиент – это вектор, который указывает направление наибольшего роста функции.
- В машинном обучении мы хотим уменьшить функцию потерь, значит:
  - Градиент показывает в какую сторону и насколько нужно изменить параметры, чтобы уменьшить ошибку.

есть функция потерь $L(w)$, где w — параметры модели.

Градиент — это производная:

$$\frac{dL}{dw}$$

- Если градиент положительный → функция растёт → нужно **уменьшить** w.
- Если градиент отрицательный → функция убывает → нужно **увеличить** w.

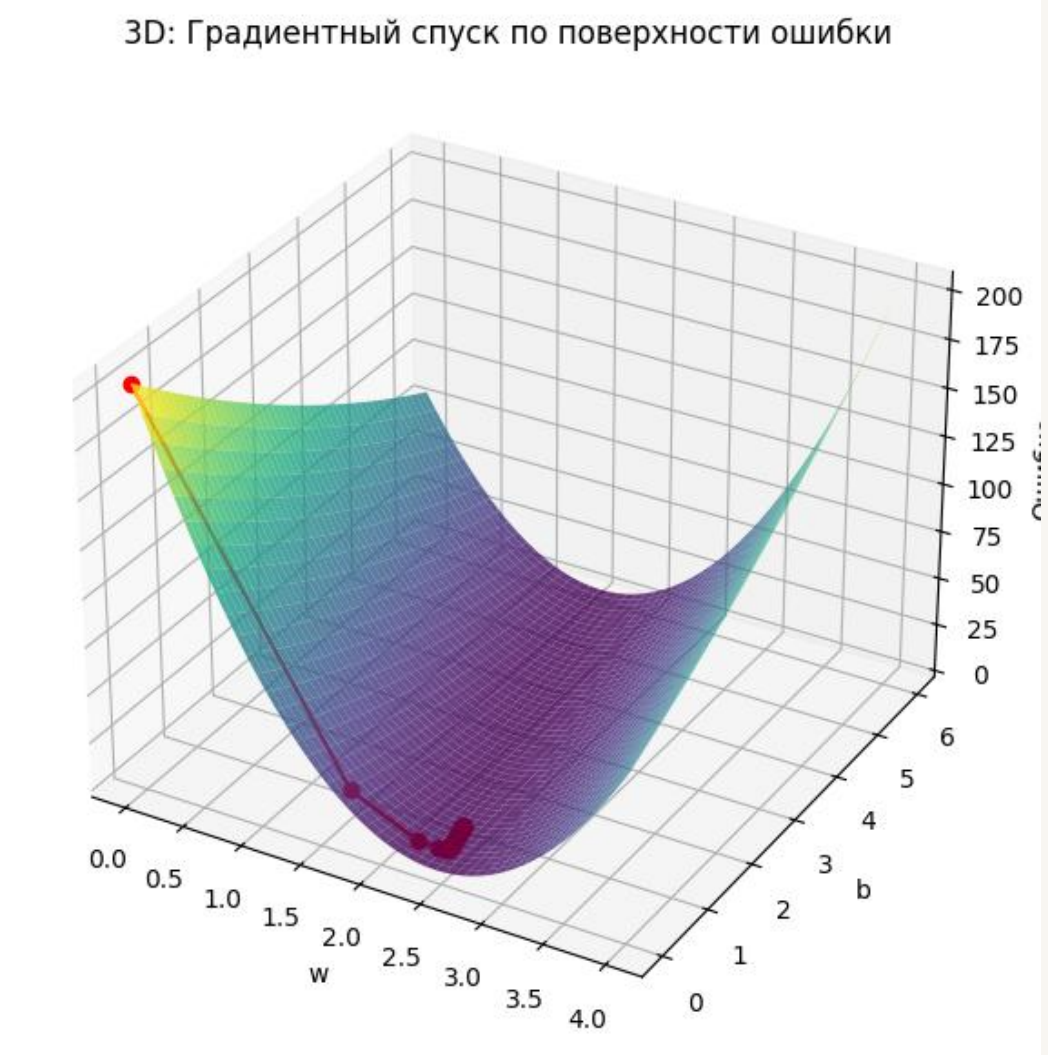
Вот почему в **градиентном спуске**:

$$w := w - \alpha \cdot \frac{dL}{dw}$$

- α — шаг обучения.
- $\frac{dL}{dw}$ — градиент (направление, куда идти).
- Мы **шагаем в сторону уменьшения ошибки**.

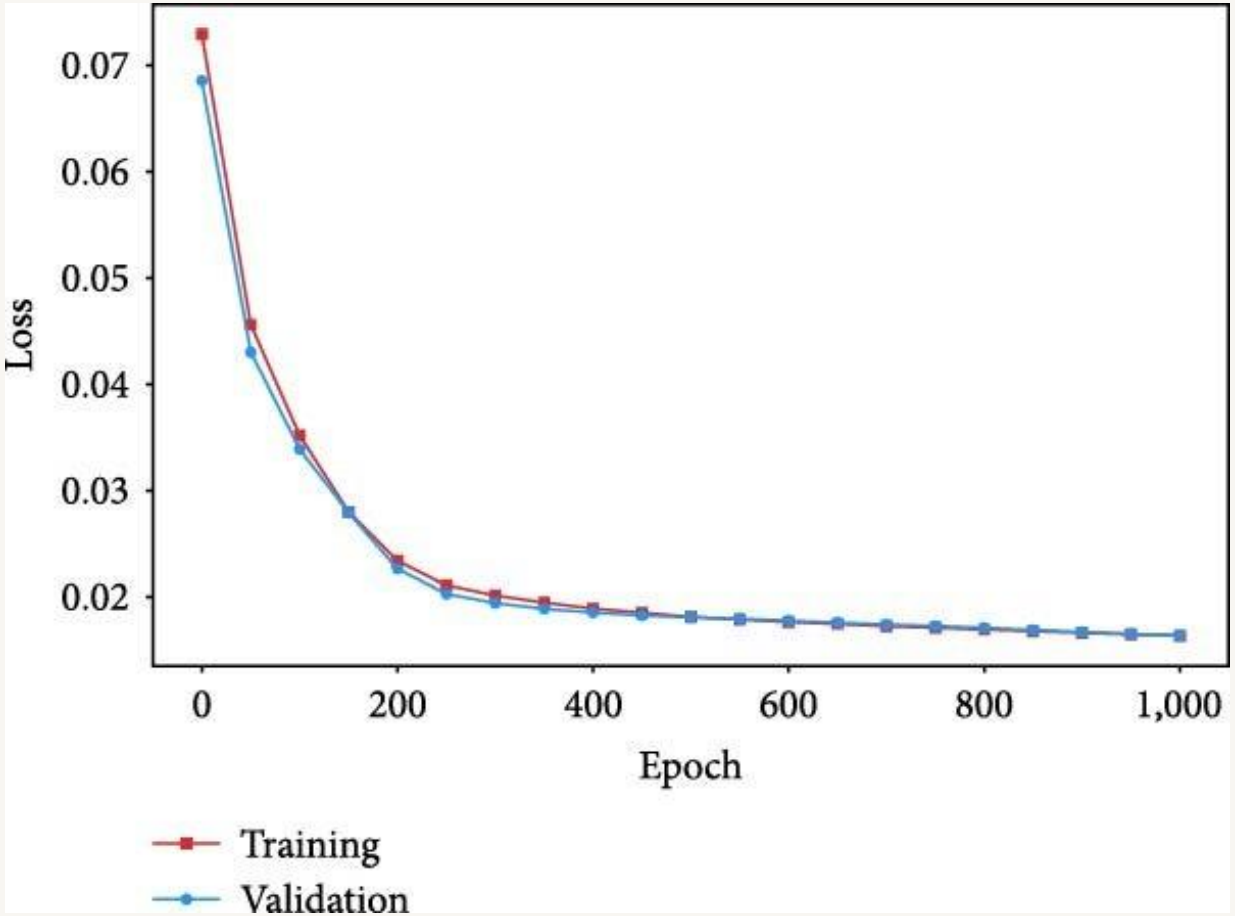
Задание 4. Градиентный спуск

- Чтобы найти такие параметры w1,w2,w3,b, при которых ошибка минимальна, используем **градиентный спуск**.



Что такое loss (функция потерь)?

- **Loss** – это число, которое показывает **насколько плохо** модель предсказывает результаты. Это мера ошибки между тем, **что предсказала модель**, и тем, **что должно было быть на самом деле**.



Задание 4. Градиентный спуск

- Чтобы найти такие параметры w1,w2,w3, b, при которых ошибка минимальна, используем **градиентный спуск**.

Обозначим:

$$e_i = \hat{y}_i - y_i$$

Градиенты по каждому параметру:

$$\frac{\partial \text{MSE}}{\partial w_j} = \frac{2}{n} \sum_{i=1}^{n} e_i x_{ij}, \quad j = 1, 2, 3$$

$$\frac{\partial \text{MSE}}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} e_i$$

Задание 4. Правила обновления параметров

$$w_j := w_j - \alpha \cdot \frac{\partial \text{MSE}}{\partial w_j}$$

$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

где α — **шаг обучения (learning rate)**.

Задание 4. Градиентный спуск

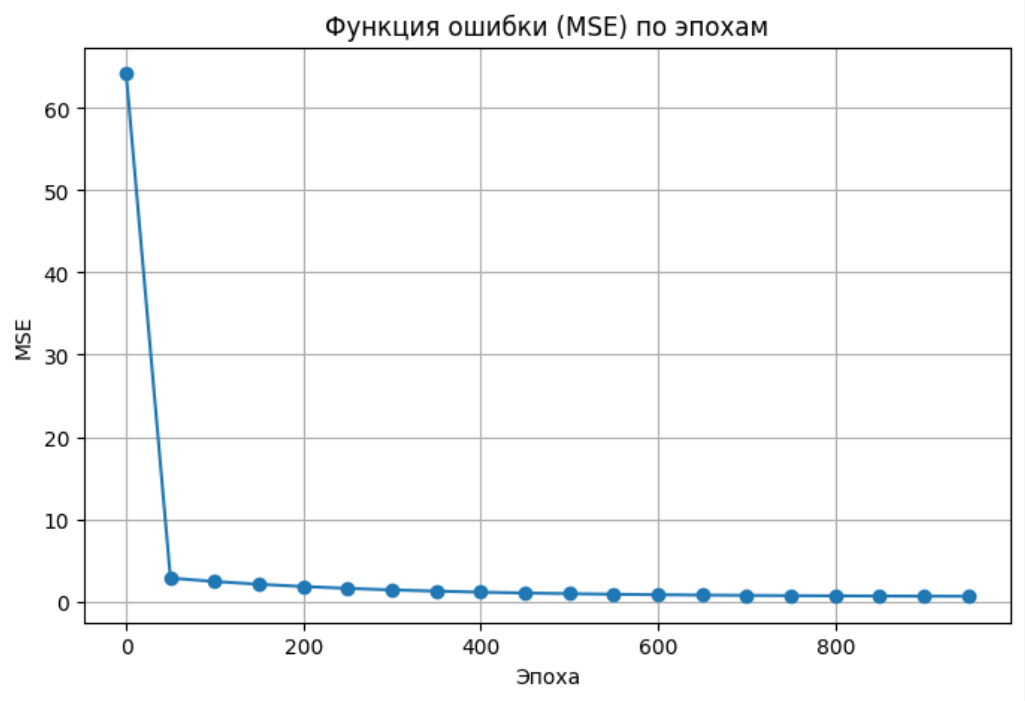
```python
def gradient_descent(X1, X2, X3, Y, alpha=0.0001, epochs=1000):
    w1, w2, w3, b = 0.0, 0.0, 0.0, 0.0
    n = len(Y)
    history = []

    for epoch in range(epochs):
        y_pred = w1 * X1 + w2 * X2 + w3 * X3 + b
        error = y_pred - Y

        dw1 = (2/n) * np.dot(error, X1)
        dw2 = (2/n) * np.dot(error, X2)
        dw3 = (2/n) * np.dot(error, X3)
        db  = (2/n) * np.sum(error)

        w1 -= alpha * dw1
        w2 -= alpha * dw2
        w3 -= alpha * dw3
        b  -= alpha * db

        # Сохраняем ошибку каждые 50 эпох
        if epoch % 50 == 0:
            loss = mse(Y, y_pred)
            history.append(loss)

    return w1, w2, w3, b, history
```

Задание 5. Обучение модели

```
alpha = 0.0001
epochs = 1000

w1, w2, w3, b, history = gradient_descent(X1, X2, X3, Y, alpha=alpha, epochs=epochs)

print(f"Финальные параметры:")
print(f"w1 (лимонный сок): {w1:.4f}")
print(f"w2 (сахар):          {w2:.4f}")
print(f"w3 (газ):            {w3:.4f}")
print(f"b (свободный член): {b:.4f}")
```

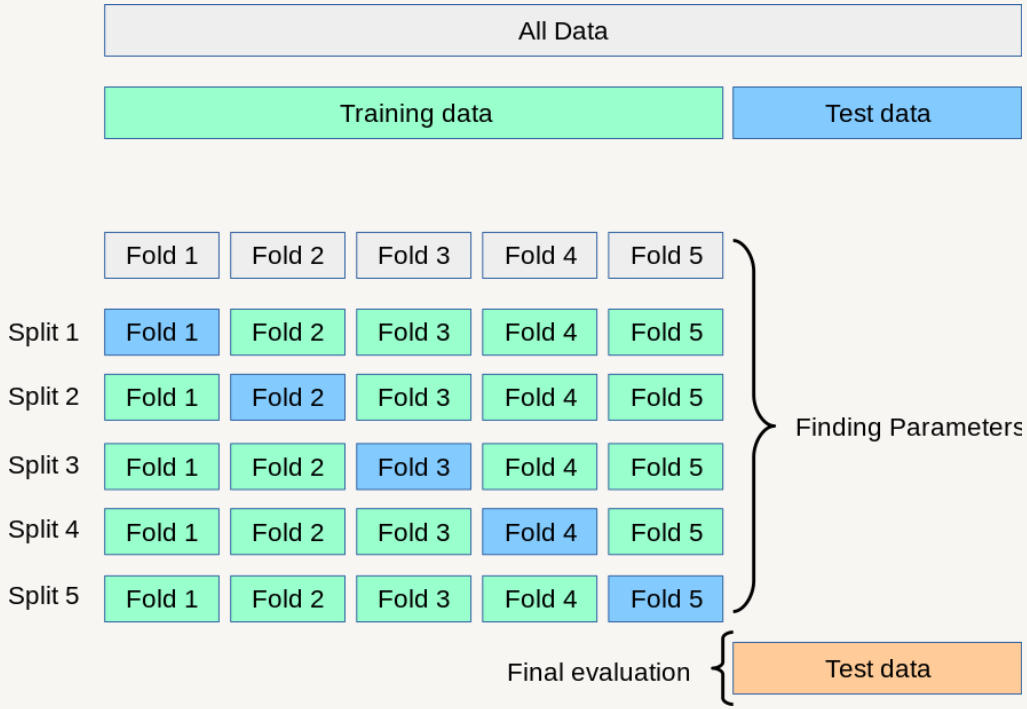
```
Финальные параметры:
w1 (лимонный сок): 0.0698
w2 (сахар):          0.0789
w3 (газ):            0.4650
b (свободный член): 0.0448
```

Задание 6. График изменения ошибки по эпохам



Задание 7. Кросс-валидация для подбора оптимального числа эпох

- **Кросс-валидация** – это метод **оценки качества модели**, при котором:
- данные делятся на **несколько частей (фолдов)**;
- модель обучается на **одних частях** и проверяется на **других**;
- процесс повторяется несколько раз, и считается **средняя ошибка**.



Задание 7. Кросс-валидация для подбора оптимального числа эпох

```python
from sklearn.model_selection import KFold

def cross_val_epochs(X1, X2, X3, Y, epochs_list, alpha=0.0001, k=5):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    results = []

    for ep in epochs_list:
        fold_mse_list = []

        for train_idx, val_idx in kf.split(X1):
            x1_train, x1_val = X1[train_idx], X1[val_idx]
            x2_train, x2_val = X2[train_idx], X2[val_idx]
            x3_train, x3_val = X3[train_idx], X3[val_idx]
            y_train, y_val = Y[train_idx], Y[val_idx]

            w1_cv, w2_cv, w3_cv, b_cv, _ = gradient_descent(x1_train, x2_train, x3_train, y_train, alpha=alpha, epochs=ep)
            y_pred_val = w1_cv * x1_val + w2_cv * x2_val + w3_cv * x3_val + b_cv
            fold_mse_list.append(mse(y_val, y_pred_val))

        avg_mse = np.mean(fold_mse_list)
        results.append((ep, avg_mse))

    return results
```

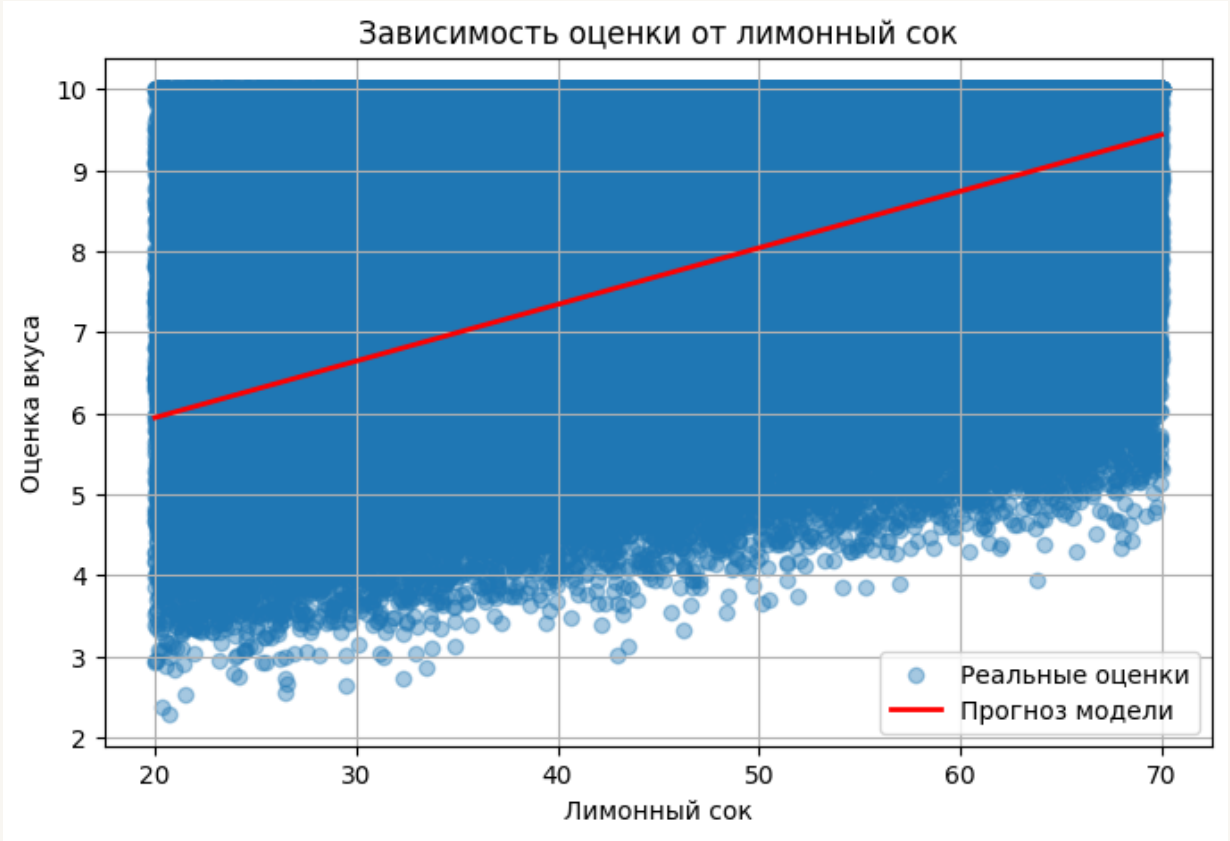
Задание 8. Использование модели для предсказания

```python
def predict(lemon_juice_ml, sugar_g, fizziness):
    return w1 * lemon_juice_ml + w2 * sugar_g + w3 * fizziness + b
```

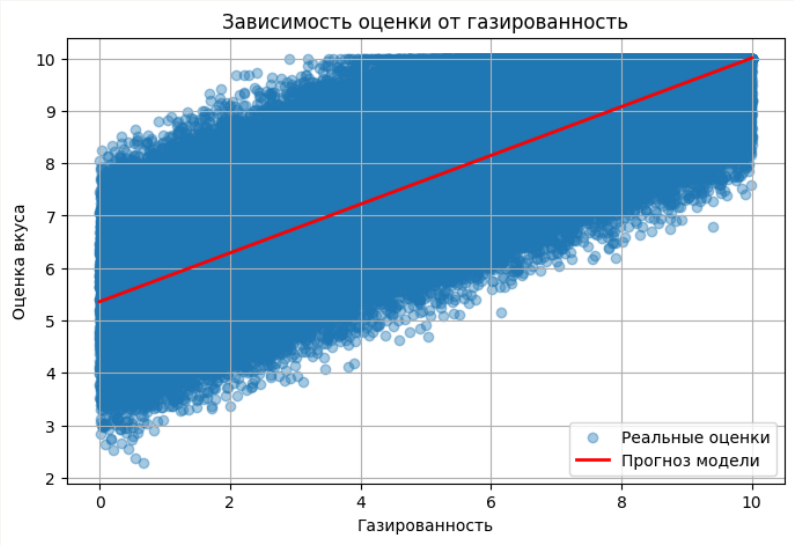
```python
example = {
    "lemon_juice_ml": 45,
    "sugar_g": 25,
    "fizziness": 6
}

predicted_rating = predict(**example)
print(f"Предсказание оценки вкуса при x1={example['lemon_juice_ml']}, x2={example['sugar_g']}, x3={example['fizziness']}: {predicted_rating:.2f} из 10")
```

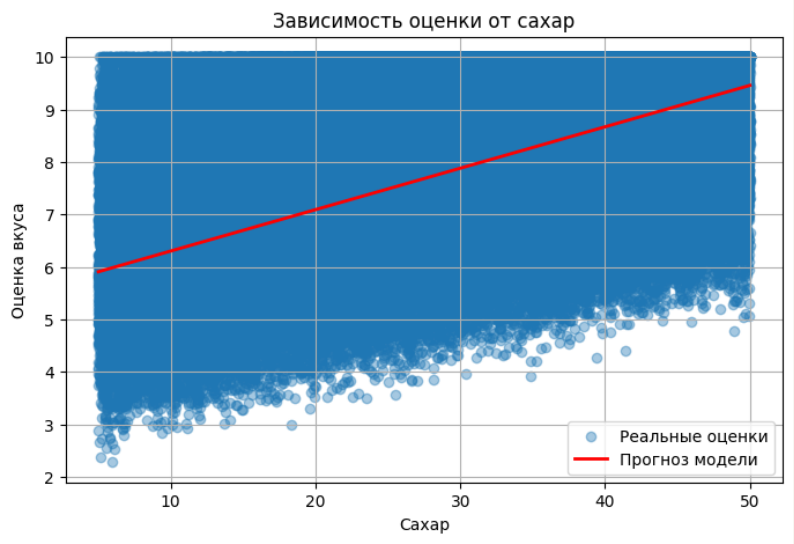
```
Предсказание оценки вкуса при x1=45, x2=25, x3=6: 7.95 из 10
```

Задание 9. Графики



Задание 9. Графики





Задание 10. Графики