Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по практической работе №2**

По дисциплине: «Анализ больших данных»

Тема: «**Работа с библиотекой streamlit и FastAPI**»

|  |
| --- |
| Выполнил:  студент группы  Б.ПИН.РИС - 21.06  Леонов А. М. |
| Проверила:  старший преподаватель  кафедры ПО  Корнеева Е.И. |

Тверь 2025

**Оглавление**

[**Введение** 3](#_Toc194242093)

[**Цель:** 3](#_Toc194242094)

[**Формулировка задачи:** 3](#_Toc194242095)

[**Задача 1. Разбор данных из Kaggle** 3](#_Toc194242096)

[**Ссылка на репозиторий с программной реализацией:** 4](#_Toc194242097)

[**Описание проделанной работы** 5](#_Toc194242098)

[**Описание программы:** 5](#_Toc194242099)

[**Описание алгоритмов:** 5](#_Toc194242100)

[**Описание данных:** 8](#_Toc194242101)

[**Методы анализа данных и предобработки:** 8](#_Toc194242102)

[**Обучение и оценка моделей:** 8](#_Toc194242103)

[**Визуализация:** 8](#_Toc194242104)

[**Результаты** 10](#_Toc194242105)

[**Общее задание** 10](#_Toc194242106)

[**Описание базы данных – признаки, целевая переменная:** 10](#_Toc194242107)

[**Исследовательский анализ данных:** 11](#_Toc194242108)

[**Вывод по анализу:** 12](#_Toc194242109)

[**Визуализация результата:** 13](#_Toc194242110)

[**Критерии качества:** 15](#_Toc194242111)

[**Вывод по результату и критериям:** 15](#_Toc194242112)

[**Плюсы и минусы streamlit** 15](#_Toc194242113)

[**Плюсы и минусы FastApi** 16](#_Toc194242114)

[**Веб интерфейс** 17](#_Toc194242115)

[**Вывод** 22](#_Toc194242116)

[**Использованные материалы** 23](#_Toc194242117)

[**Листинг кода** 24](#_Toc194242118)

[**Streamlit:** 24](#_Toc194242119)

[**FastApi:** 27](#_Toc194242120)

# **Введение**

## **Цель:**

Разобраться с представлением отчетов в веб-версии с помощью библиотеки Streamlit. Реализовать отчет по одной из готовых лабораторных работ в формате веб-приложения.

## **Формулировка задачи:**

## **Задача 1. Разбор данных из Kaggle**

Построить минимум две модели классификации с использованием алгоритмов KNN, логистическая регрессия, SVM. Алгоритмы допустимо использовать из библиотеки scikit-learn.

Модели строятся для задачи бинарной классификации.

Затем сравните их качество, используя метрики, такие как A, P, R, E, confusion\_matrix, ROC\_AUC.

на тестовых данных.

Подробное описание:

1. Загрузить один из наборов данных. Загружаются все файлы по ссылке.
2. Провести разведочный анализ данных, ответив на следующие вопросы:
3. Сколько строк в датафрейме, сколько столбцов
4. Сколько места занимает датафрейм в оперативной памяти
5. Для каждой интервальной переменной подсчитать следующее - мин, медиана, среднее, макс и персентили 25, 75
6. Для каждой категориальной переменной рассчитать моду и сколько раз мода встречается в данных
7. Подготовка датасета к построению моделей ML
8. Провести анализ и обработку пропусков (либо заменить, либо удалить)
9. Провести анализ и обработку выбросов (либо заменить, либо удалить)
10. Провести анализ и обработку категориальных переменных (сколько таких переменных, закодируйте категориальные переменные одним из методов ( one hot encoding, mean target, frequence encoding)
11. Разделить датасет на трейн и тест
12. Построить следующие классификационные алгоритмы:
13. Knn
14. Logistic regression
15. SVM
16. Оценить качество алгоритмов, выбрать самый оптимальный алгоритм
17. Возможно ли улучшить алгоритм, предложите идеи?

**Вариант 9**

БД- Playground Series - Season 3, Episode 12

Ссылка на БД- https://www.kaggle.com/competitions/playground-series-s3e12/data?select=train.csv

## **Ссылка на репозиторий с программной реализацией:**

GitHub- https://github.com/akkafe1ix/BigData\_Practice\_2

# **Описание проделанной работы**

## **Описание программы:**

Программа предназначена для анализа данных и машинного обучения. Реализована веб-версия отчета с использованием Streamlit и FastAPI. В приложении выполняются всё это реализовано на FastAPI и подтягивается http get запросами со Streamlit:

* загрузка и анализ данных;
* предварительная обработка;
* обучение моделей (KNN, логистическая регрессия, SVM);
* оценка качества моделей;
* визуализация границ решений.

Приложение включает несколько вкладок для удобного представления информации реализованных на streamlit:

1. Об программе – описание проекта.
2. Датасет – представление загруженного набора данных.
3. EDA (Exploratory Data Analysis) – исследовательский анализ данных.
4. Результаты моделей – сравнительный анализ точности моделей.
5. Визуализация моделей – отображение границ решений.

## **Описание алгоритмов:**

**1. K-Nearest Neighbors (KNN)**

**KNN** — это алгоритм классификации (или регрессии), который основывается на принципе, что объекты, расположенные близко друг к другу, имеют схожие характеристики. Это один из самых простых алгоритмов для классификации и регрессии, но с высокими требованиями к вычислительным ресурсам на больших данных.

**Основные принципы работы:**

* Алгоритм работает следующим образом: для каждого нового объекта алгоритм находит **K ближайших соседей** в обучающем наборе данных и на основе их меток классифицирует этот объект.
* **Метрика расстояния**: Расстояние между объектами чаще всего измеряется с использованием евклидова расстояния, хотя могут быть использованы и другие метрики (например, манхэттенское расстояние, косинусное сходство).

**Шаги алгоритма:**

1. Выбирается значение **K** (количество ближайших соседей).
2. Для каждого нового примера (классифицируемого объекта) вычисляется расстояние до всех объектов обучающего набора.
3. Выбираются **K** объектов с наименьшими расстояниями.
4. Для классификации выбирается наиболее частая метка среди **K** ближайших соседей (для регрессии — среднее значение).

**2. Logistic Regression**

**Логистическая регрессия** — это алгоритм машинного обучения, используемый для бинарной классификации. Несмотря на название, это не регрессия в традиционном смысле, а именно классификационный метод, который использует функцию логистической регрессии для предсказания вероятности того, что объект принадлежит к одному из двух классов.

**Основные принципы работы:**

* Логистическая регрессия вычисляет вероятность принадлежности объекта к положительному классу (например, 1 в задаче бинарной классификации) с помощью логистической функции (сигмоиды).
* Функция сигмоиды преобразует линейную комбинацию признаков в диапазон от 0 до 1, что и интерпретируется как вероятность:

Изображение выглядит как текст, Шрифт, линия, чек

Контент, сгенерированный ИИ, может содержать ошибки.

* Алгоритм минимизирует **логистическую ошибку** с использованием градиентного спуска для оптимизации коэффициентов www.

**Шаги алгоритма:**

1. Модель логистической регрессии находит оптимальные коэффициенты www, которые минимизируют ошибку в предсказаниях.
2. Эти коэффициенты затем используются для вычисления вероятности того, что объект принадлежит к положительному классу.
3. Если вероятность больше 0.5, объект классифицируется как положительный (1), иначе как отрицательный (0).

**3. Support Vector Machine (SVM)**

**SVM** — это алгоритм машинного обучения, используемый для классификации и регрессии. Он находит оптимальную гиперплоскость, которая максимально разделяет объекты разных классов в пространстве признаков.

**Основные принципы работы:**

* Алгоритм ищет гиперплоскость, которая создает максимальный зазор (margin) между объектами разных классов.
* Объекты, которые находятся ближе всего к гиперплоскости, называются опорными векторами, и именно они определяют положение разделяющей границы.
* В случае, если данные неразделимы линейно, используется ядровая функция, которая переводит данные в пространство большей размерности, где они становятся линейно разделимыми.

**Шаги алгоритма:**

1. Определяется оптимальная гиперплоскость, которая разделяет классы с максимальным зазором.
2. Если данные не линейно разделимы, применяется ядровая функция (например, полиномиальное ядро или ядро Гаусса).
3. Новые объекты классифицируются на основе их положения относительно гиперплоскости.

SVM особенно эффективен на небольших наборах данных с высокой размерностью, но может требовать высокой вычислительной мощности при использовании сложных ядерных функций.

## **Описание данных:**

Программа работает с датасетом, содержащим числовые признаки:

* gravity – гравитация
* ph – кислотность
* osmo – осмотическое давление
* cond – проводимость
* urea – уровень мочевины
* calc – уровень кальция
* target – целевая переменная (бинарная классификация)

Данные загружаются из файлов train.csv и test.csv. Отображается информация о количестве строк, столбцов и размера датасета.

## **Методы анализа данных и предобработки:**

Перед обучением моделей выполняется анализ данных:

* Вычисление статистических характеристик (минимум, максимум, среднее, медиана, квартильные значения);
* Анализ категориальных признаков (если присутствуют);
* Масштабирование признаков с помощью StandardScaler.

## **Обучение и оценка моделей:**

В программе используются три модели машинного обучения:

1. KNN (K-Nearest Neighbors) – реализация метода ближайших соседей вручную и с использованием библиотеки sklearn.
2. Логистическая регрессия – базовая линейная модель классификации.
3. SVM (Support Vector Machine) – метод опорных векторов.

Для оценки моделей применяются метрики:

* Accuracy (точность);
* Precision (точность положительного класса);
* Recall (полнота);
* F1-score (среднее гармоническое precision и recall);
* ROC-AUC (качество разделения классов, если применимо).

На основе результатов вычисляется лучшая модель.

## **Визуализация:**

Для наглядного представления классификации строятся границы решений моделей. Используется цветовая карта ListedColormap для отображения принадлежности точек к классам.

Объекты данных отображаются в виде таблиц и диаграмм, что позволяет пользователю легко интерпретировать информацию. Вкладка "Датасет" показывает исходные данные в табличном виде с возможностью прокрутки и фильтрации. В "EDA" строятся гистограммы распределения признаков, боксплоты и корреляционные матрицы. Визуализация моделей включает интерактивные графики границ классификации, что упрощает понимание работы алгоритмов.

# **Результаты**

## **Общее задание**

### **Описание базы данных – признаки, целевая переменная:**

Набор данных **Playground Series - Season 3, Episode 12** с платформы Kaggle содержит информацию о физических и химических свойствах образцов, используемых для бинарной классификации.

**Признаки (Features):**

Датасет включает 7 числовых признаков, описывающих различные характеристики образцов:

1. gravity – плотность образца.
2. ph – уровень pH.
3. osmo – осмолярность (концентрация растворённых частиц).
4. cond – электрическая проводимость.
5. urea – концентрация мочевины.
6. calc – уровень кальция.
7. id – уникальный идентификатор (не используется в моделировании).

Все признаки являются числовыми и измеряются в соответствующих единицах.

**Целевая переменная (Target):**

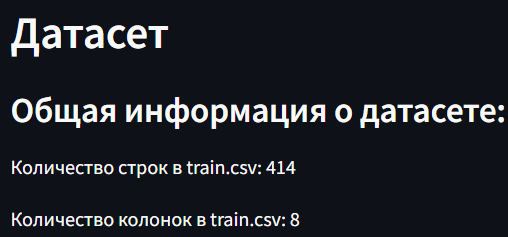
Целевая переменная **target** представляет собой бинарную классификацию:

* 0: Отрицательный результат.
* 1: Положительный результат.

Задача классификации заключается в предсказании значения target на основе представленных признаков.

### **Исследовательский анализ данных:**

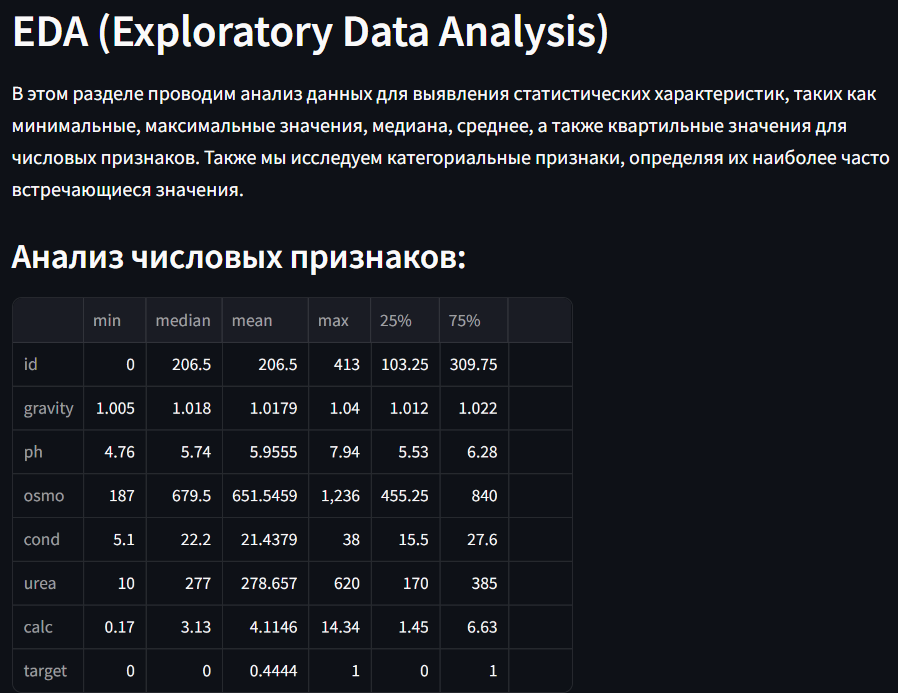
A. Сколько строк в датафрейме, сколько столбцов



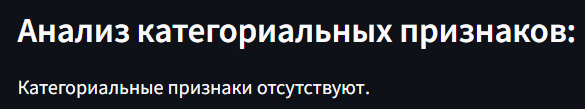
B. Сколько места занимает датафрейм в оперативной памяти



C. Для каждой интервальной переменной подсчитать следующее - мин, медиана, среднее, макс и персентили 25, 75



D. Для каждой категориальной переменной рассчитать моду и сколько раз мода встречается в данных



### **Вывод по анализу:**

**Размерность датафрейма:**

* В датафрейме **414 строк и 8 столбцов**.
* Семь столбцов содержат числовые признаки, а один столбец (**target**) является **целевой переменной** (0 или 1).

**Место, занимаемое в памяти:**

* Датафрейм занимает **около 26 KB** памяти.
* Малый объем данных позволяет **эффективно анализировать и обрабатывать** их без значительных вычислительных затрат.

**Статистики для числовых признаков:**

* **Все числовые признаки находятся в разумных пределах.**
* **Средние и медианные значения** показывают, что данные распределены **без значительных выбросов**.

**Категориальные признаки:**

* В датасете **отсутствуют категориальные признаки** – все данные числовые.

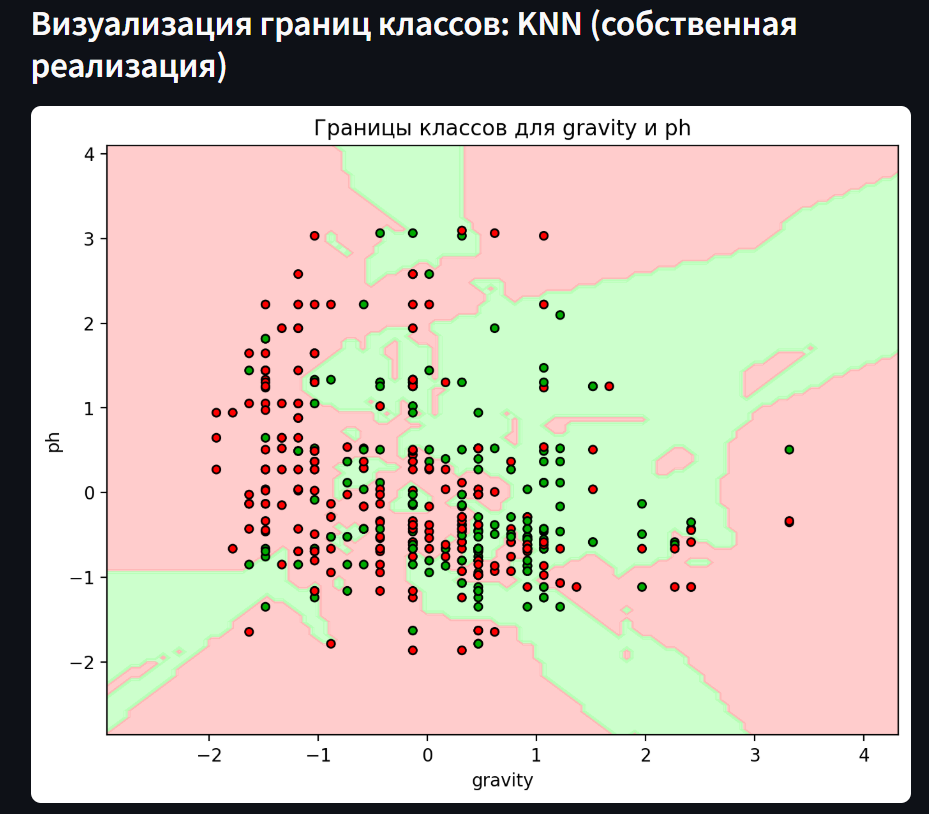
**Пропущенные значения:**

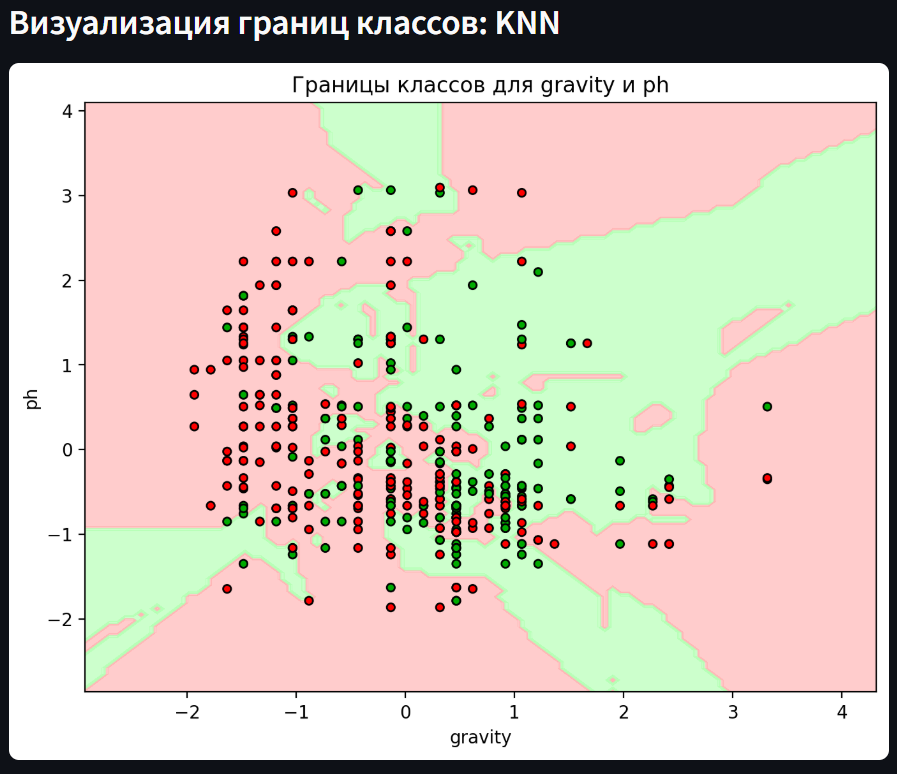
* **Пропущенные значения отсутствуют**, данные полностью заполнены.

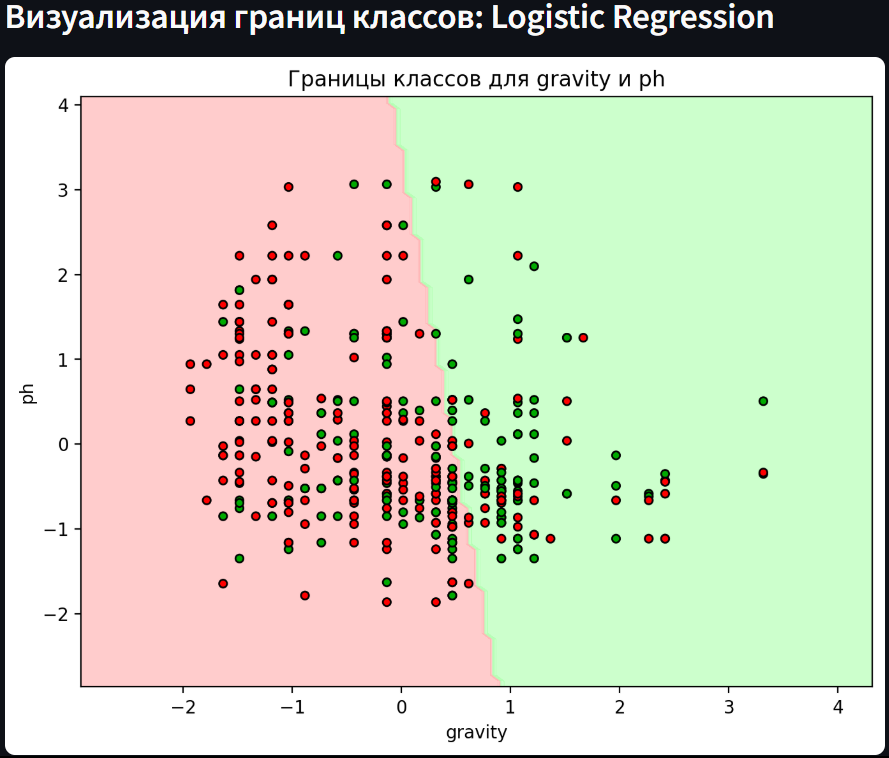
**Мода для целевой переменной:**

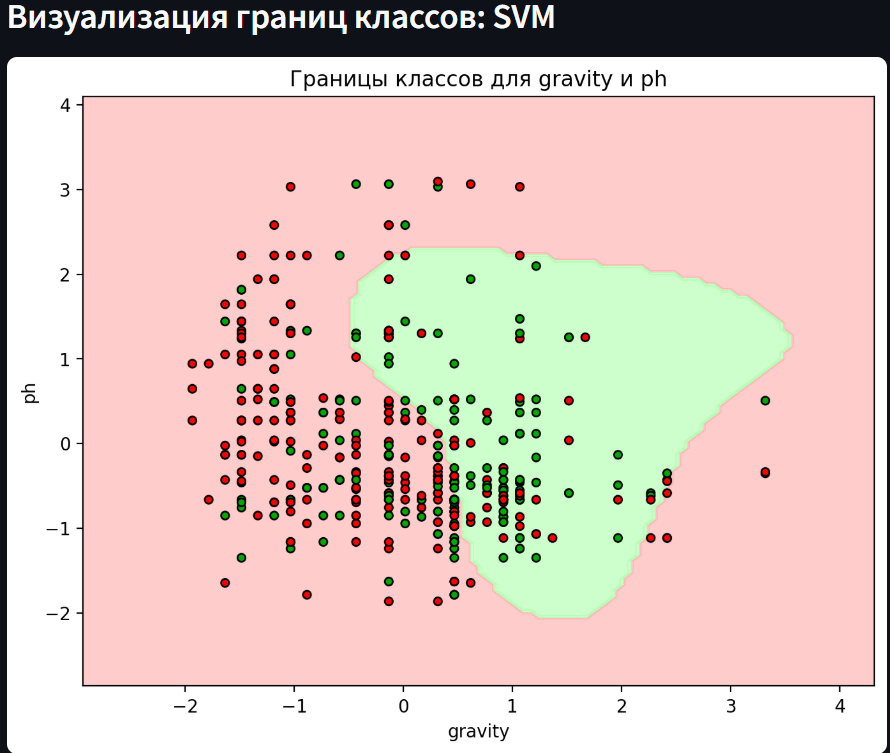
* Для целевой переменной **target** мода — **0**, что означает, что **класс 0 встречается чаще всего**.

### **Визуализация результата:**

****







### **Критерии качества:**



### **Вывод по результату и критериям:**

SVM оказалась более точной, чем KNN из библиотеки, КНН собственной реализации и чем логическая регрессия.

## **Плюсы и минусы streamlit**

Плюсы:

* Простота использования – позволяет быстро создавать веб-приложения без глубоких знаний веб-разработки.
* Быстрота разработки – код, написанный в Streamlit, минималистичен и легко модифицируется.
* Интерактивность – поддержка виджетов (st.slider, st.selectbox и др.) делает приложения удобными.
* Интеграция с Python – поддерживает работу с pandas, matplotlib, seaborn, sklearn и другими библиотеками.
* Развертывание без сложных настроек – можно запускать локально или развертывать в облаке с минимальными усилиями.
* Автоматическое обновление интерфейса – изменения в коде моментально отображаются в веб-приложении.

Минусы:

* Ограниченные возможности кастомизации интерфейса – сложнее настроить внешний вид по сравнению с Dash или Flask.
* Поддержка только Python – нельзя использовать JavaScript или другие языки для расширенной функциональности.
* Медленная обработка больших объемов данных – приложения могут тормозить при работе с большими датасетами.
* Не подходит для сложных веб-приложений – ограничен в плане создания сложных пользовательских взаимодействий.
* Зависимость от сервера – при развертывании в облаке требует постоянного соединения, что может увеличивать затраты.

## **Плюсы и минусы FastApi**

Плюсы:

* Очень высокая производительность  
  Основан на Starlette и Pydantic, работает асинхронно (async/await), производительность сравнима с Node.js и Go.
* Автоматическая документация API  
  Автоматически создаёт Swagger UI и Redoc из аннотаций типов — удобно для тестирования и работы с API.
* Поддержка асинхронности  
  Позволяет обрабатывать тысячи запросов одновременно, особенно полезно при работе с БД, внешними API, очередями и т.д.
* Аннотации типов = валидация и автогенерация  
  Использование Pydantic позволяет валидировать входные данные и генерировать схемы OpenAPI без лишнего кода.
* Простой и современный синтаксис  
  Хорошо читаемый и "питоничный" код, особенно подходит для разработчиков, знакомых с Python 3.6+.
* Интеграция с другими библиотеками  
  Хорошо сочетается с SQLAlchemy, Tortoise ORM, Redis, Celery, и другими инструментами backend-разработки.
* Хорошо документирован  
  Официальная документация понятная, с примерами и объяснениями.

Минусы:

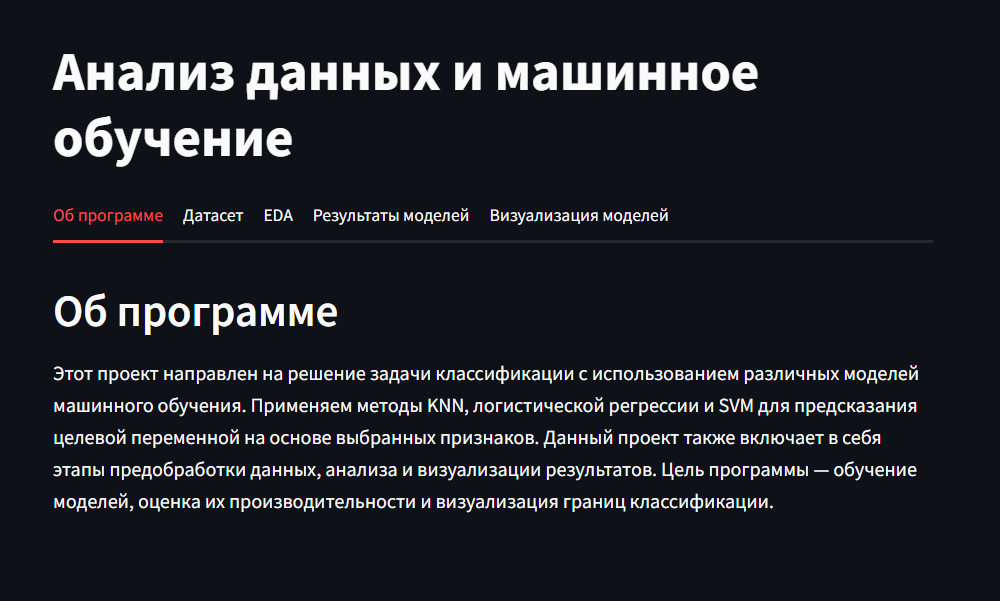
* Меньшее комьюнити по сравнению с Flask/Django  
  Хотя FastAPI быстро развивается, у него пока меньше готовых решений и пакетов, чем у Django или Flask.
* Сложности с асинхронностью  
  Если использовать сторонние библиотеки, не поддерживающие async, можно получить проблемы (например, с SQLAlchemy без async).
* Pydantic может быть медленным при больших объёмах данных  
  Особенно это касается сложных вложенных структур и сериализации/десериализации.

Менее гибкий, чем Flask, в некоторых аспектах  
Например, нестандартные подходы к middleware или ошибкам иногда требуют больше усилий, чем во Flask.

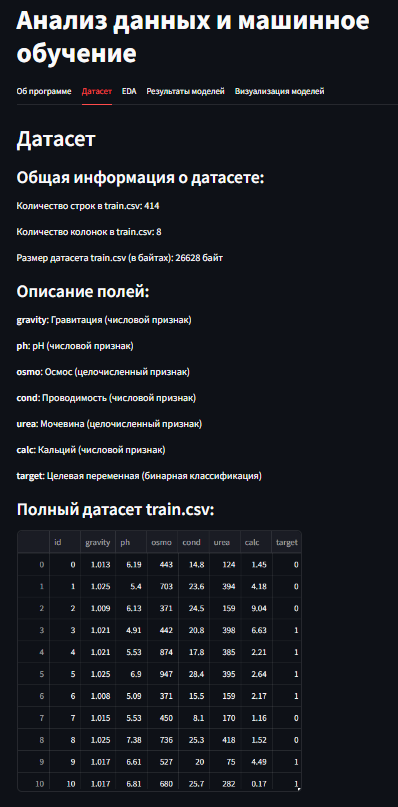
* Не всегда очевиден порядок выполнения зависимостей (Depends)  
  В больших проектах может стать трудно читаемым.

## **Веб интерфейс**

Начальная вкладка:

****

Описание датасета:

****

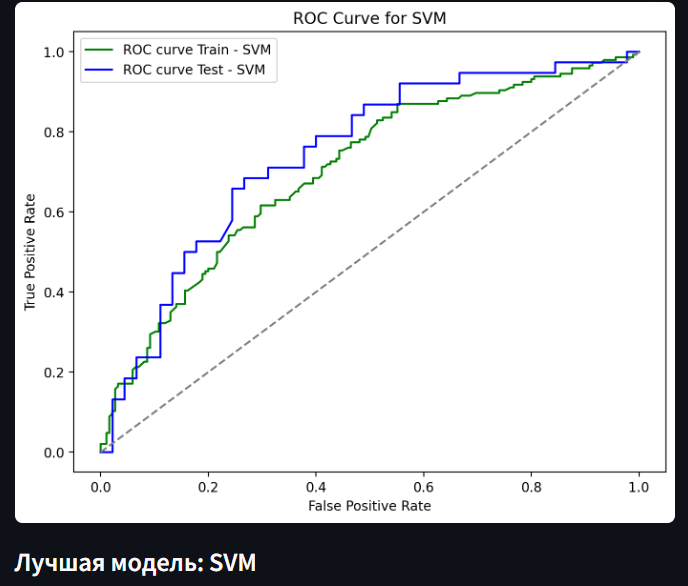
EDA:



Результаты моделей (сначала идут ROC-кривые):



А затем после ROC-кривых по каждой модели идёт вывод по лучшей модели:



Визуализация моделей:



Далее идут другие интерпретации границ классов, а они не уместились в скриншот.

# **Вывод**

В результате работы реализовано веб-приложение, позволяющее загружать данные, анализировать их, обучать модели классификации, сравнивать их качество и визуализировать границы решений.  
**Streamlit** предоставляет удобный и интуитивно понятный интерфейс для представления отчётов в интерактивной форме. Однако, несмотря на его простоту, при работе с большими объёмами данных и сложной логикой может потребоваться разделение задач на frontend и backend.

Для решения этих задач в проекте была использована технология **FastAPI**, которая выступает в роли backend-сервера, обрабатывающего все вычисления и передающего результаты в Streamlit по API-запросам. FastAPI обладает рядом преимуществ — высокой производительностью, асинхронной обработкой, встроенной валидацией данных и автоматической генерацией документации. Это делает его отличным выбором для построения быстрых, надёжных и масштабируемых REST-сервисов в Python.

Такой подход с разделением логики и интерфейса делает приложение гибким, масштабируемым и пригодным для дальнейшего расширения и интеграции в более крупные системы.

# **Использованные материалы**

1. Введение в библиотеку Pandas // PythonRu URL: https://pythonru.com/biblioteki/pandas-vvedenie (дата обращения: 24.02.2025).
2. Основы визуализации данных в Matplotlib и Seaborn // Habr URL: https://habr.com/ru/post/485370 (дата обращения: 24.02.2025).
3. Полное руководство по SQLAlchemy // PythonRu URL: <https://pythonru.com/biblioteki/vvedenie-v-sqlalchemy> (дата обращения: 24.02.2025).
4. Статистический анализ данных в Python с использованием SciPy // DataScienceBlog URL: https://datascienceblog.net/post/statistics/scipy (дата обращения: 24.02.2025).
5. Реализация градиентного спуска в Python // Towards Data Science URL: https://towardsdatascience.com/gradient-descent (дата обращения: 24.02.2025).
6. Разведочный анализ данных (EDA) в Python // AnalyticsVidhya URL: https://www.analyticsvidhya.com/blog/2020/02/exploratory-data-analysis-in-python (дата обращения: 24.02.2025).
7. Работа с датасетом mpg в Seaborn // Seaborn Documentation URL: https://seaborn.pydata.org/generated/seaborn.load\_dataset.html (дата обращения: 24.02.2025).
8. Метод опорных векторов (SVM) // GeeksforGeeks URL: https://www.geeksforgeeks.org/support-vector-machine-introduction/ (дата обращения: 24.02.2025).
9. Логистическая регрессия в машинном обучении // AnalyticsVidhya URL: https://www.analyticsvidhya.com/blog/2021/06/logistic-regression/ (дата обращения: 24.02.2025).
10. Основы KNN-алгоритма // Towards Data Science URL:https://towardsdatascience.com/k-nearest-neighbors (дата обращения: 24.02.2025).
11. Основы Streamlit // Towards Data Science URL: https://towardsdatascience.com/streamlit (дата обращения: 24.02.2025).
12. Введение в FastAPI — современный фреймворк для создания API на Python // FastAPI Official Docs URL: <https://fastapi.tiangolo.com/ru/> (дата обращения: 30.03.2025)

# **Листинг кода**

## **Streamlit:**

#python -m streamlit run app.py --server.port 8001

import streamlit as st

import pandas as pd

import numpy as np

import requests

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

API\_URL = "http://127.0.0.1:8001"

#st.set\_page\_config(layout="wide")

st.title("Анализ данных и машинное обучение")

tab1, tab2, tab3, tab4, tab5 = st.tabs(

    ["Об программе", "Датасет", "EDA", "Результаты моделей", "Визуализация моделей"]

)

# 1. Об программе

with tab1:

    st.header("Об программе")

    st.markdown("""

    Это Streamlit-приложение выполняет анализ данных и классификацию с использованием нескольких моделей:

    - Собственная реализация KNN (CustomKNN)

    - KNeighborsClassifier из sklearn

    - Логистическая регрессия

    - Support Vector Machine

    Данные: признаки `gravity`, `ph` и целевая переменная `target`.

    Все вычисления (EDA, обучение, метрики, графики) выполняются через FastAPI.

    """)

# 2. Датасет

with tab2:

    st.header("Датасет")

    try:

        info = requests.get(f"{API\_URL}/data\_info").json()

        st.subheader("Общая информация:")

        st.write(f"Количество строк: {info['n\_rows']}")

        st.write(f"Количество колонок: {info['n\_cols']}")

        st.write(f"Объем в памяти: {info['memory\_bytes']} байт")

        # Описание каждого поля

        st.write("### Описание полей:")

        description = {

        "gravity": "Гравитация (числовой признак)",

        "ph": "pH (числовой признак)",

        "osmo": "Осмос (целочисленный признак)",

        "cond": "Проводимость (числовой признак)",

        "urea": "Мочевина (целочисленный признак)",

        "calc": "Кальций (числовой признак)",

        "target": "Целевая переменная (бинарная классификация)",

        }

        for col, desc in description.items():

            st.write(f"\*\*{col}\*\*: {desc}")

        st.subheader("Первые 10 строк:")

        df = pd.DataFrame(info["head"])

        st.dataframe(df)

    except Exception as e:

        st.error(f"Ошибка получения информации о датасете: {e}")

# 3. EDA

with tab3:

    st.header("Exploratory Data Analysis (EDA)")

    st.write("""

        В этом разделе проводим анализ данных для выявления статистических характеристик, таких как минимальные,

        максимальные значения, медиана, среднее, а также квартильные значения для числовых признаков.

        Также мы исследуем категориальные признаки, определяя их наиболее часто встречающиеся значения.

    """)

    try:

        eda = requests.get(f"{API\_URL}/eda").json()

        st.subheader("Числовые признаки:")

        st.dataframe(pd.DataFrame(eda["numerical"]).set\_index("feature"))

        st.subheader("Категориальные признаки:")

        if eda["categorical"]:

            for col, stats in eda["categorical"].items():

                st.write(f"\*\*{col}\*\*: most frequent = {stats['mode']}, count = {stats['frequency']}")

        else:

            st.info("Категориальных признаков нет.")

    except Exception as e:

        st.error(f"Ошибка получения EDA: {e}")

# 4. Результаты моделей

with tab4:

    st.header("Результаты моделей")

    try:

        metrics = pd.DataFrame(requests.get(f"{API\_URL}/metrics").json()).set\_index("name")

        st.subheader("Метрики моделей:")

        st.dataframe(metrics.style.highlight\_max(axis=0, color='green'))

        st.subheader("ROC-кривые:")

        roc\_data = requests.get(f"{API\_URL}/roc").json()

        for model in roc\_data:

            fpr = model["fpr"]

            tpr = model["tpr"]

            name = model["name"]

            fig, ax = plt.subplots(figsize=(5, 4))

            ax.plot(fpr, tpr, label=f"{name}")

            ax.plot([0, 1], [0, 1], linestyle="--", color="gray")

            ax.set\_title(f"ROC-кривая для модели: {name}")

            ax.set\_xlabel("False Positive Rate")

            ax.set\_ylabel("True Positive Rate")

            ax.legend()

            st.pyplot(fig, bbox\_inches='tight', dpi=100)

    except Exception as e:

        st.error(f"Ошибка получения метрик или ROC: {e}")

# 5. Визуализация моделей

with tab5:

    st.header("Визуализация границ решений моделей")

    try:

        data = requests.get(f"{API\_URL}/boundaries").json()

        X = np.array(data["X"])

        y = np.array(data["y"])

        for name, surface in data["boundaries"].items():

            xx = np.array(surface["xx"])

            yy = np.array(surface["yy"])

            Z = np.array(surface["Z"])

            fig, ax = plt.subplots(figsize=(5, 4))

            ax.contourf(xx, yy, Z, cmap=ListedColormap(["#FFAAAA", "#AAFFAA"]), alpha=0.6)

            ax.scatter(X[:, 0], X[:, 1], c=y, cmap=ListedColormap(["#FF0000", "#00AA00"]), edgecolor="k")

            ax.set\_title(f"Границы решений модели: {name}")

            ax.set\_xlabel("gravity (scaled)")

            ax.set\_ylabel("ph (scaled)")

            st.pyplot(fig, bbox\_inches='tight', dpi=200)

    except Exception as e:

        st.error(f"Ошибка визуализации границ: {e}")

## **FastApi:**

#python -m uvicorn API:app --reload --host 127.0.0.1 --port 8001

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, roc\_curve

from collections import Counter

app = FastAPI()

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=["\*"],

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

)

# --- Custom KNN ---

class CustomKNN:

    def \_\_init\_\_(self, k=5):

        self.k = k

    def fit(self, X, y):

        self.X\_train = np.array(X)

        self.y\_train = np.array(y)

    def predict(self, X):

        return np.array([self.\_predict(x) for x in X])

    def \_predict(self, x):

        distances = [np.sqrt(np.sum((x - x\_train)\*\*2)) for x\_train in self.X\_train]

        k\_indices = np.argsort(distances)[:self.k]

        k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]

        return Counter(k\_nearest\_labels).most\_common(1)[0][0]

# --- Глобальные переменные ---

df = None

models = {}

X\_train = X\_val = y\_train = y\_val = None

X\_scaled = y\_all = None

selected\_features = ["gravity", "ph"]

@app.on\_event("startup")

def train\_models():

    global df, X\_train, X\_val, y\_train, y\_val, X\_scaled, y\_all, models

    df = pd.read\_csv("data/train.csv")

    X = df[selected\_features].values

    y = df["target"].values

    y\_all = y

    scaler = StandardScaler()

    X\_scaled = scaler.fit\_transform(X)

    X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

    models = {

        "CustomKNN": CustomKNN(k=5),

        "KNN": KNeighborsClassifier(n\_neighbors=5),

        "LogisticRegression": LogisticRegression(),

        "SVM": SVC(probability=True)

    }

    for model in models.values():

        model.fit(X\_train, y\_train)

@app.get("/data\_info")

def get\_data\_info():

    return {

        "n\_rows": int(df.shape[0]),

        "n\_cols": int(df.shape[1]),

        "memory\_bytes": int(df.memory\_usage(deep=True).sum()),

        "head": df.head(10).to\_dict(orient="records")

    }

@app.get("/eda")

def get\_eda():

    numerical = df.select\_dtypes(include=["int64", "float64"])

    stats = numerical.agg(["min", "median", "mean", "max"]).T

    stats["25%"] = numerical.apply(lambda x: np.percentile(x, 25))

    stats["75%"] = numerical.apply(lambda x: np.percentile(x, 75))

    stats = stats.round(3).reset\_index().rename(columns={"index": "feature"})

    cat\_features = df.select\_dtypes(include=["object"]).columns.tolist()

    cat\_data = {}

    for col in cat\_features:

        top\_val = df[col].mode()[0] if not df[col].mode().empty else None

        freq = df[col].value\_counts().iloc[0] if not df[col].value\_counts().empty else 0

        cat\_data[col] = {"mode": top\_val, "frequency": freq}

    return {

        "numerical": stats.to\_dict(orient="records"),

        "categorical": cat\_data

    }

@app.get("/metrics")

def get\_metrics():

    result = []

    for name, model in models.items():

        y\_pred = model.predict(X\_val)

        try:

            roc\_auc = roc\_auc\_score(y\_val, model.predict\_proba(X\_val)[:, 1]) if hasattr(model, "predict\_proba") else None

        except:

            roc\_auc = None

        result.append({

            "name": name,

            "accuracy": accuracy\_score(y\_val, y\_pred),

            "precision": precision\_score(y\_val, y\_pred),

            "recall": recall\_score(y\_val, y\_pred),

            "f1": f1\_score(y\_val, y\_pred),

            "roc\_auc": roc\_auc

        })

    return result

@app.get("/roc")

def get\_roc():

    roc\_data = []

    for name, model in models.items():

        if hasattr(model, "predict\_proba"):

            probs = model.predict\_proba(X\_val)[:, 1]

        else:

            probs = model.predict(X\_val)

        fpr, tpr, \_ = roc\_curve(y\_val, probs)

        roc\_data.append({

            "name": name,

            "fpr": fpr.tolist(),

            "tpr": tpr.tolist()

        })

    return roc\_data

@app.get("/boundaries")

def get\_boundaries():

    from matplotlib.colors import ListedColormap

    boundaries = {}

    x\_min, x\_max = X\_scaled[:, 0].min() - 1, X\_scaled[:, 0].max() + 1

    y\_min, y\_max = X\_scaled[:, 1].min() - 1, X\_scaled[:, 1].max() + 1

    xx, yy = np.meshgrid(np.linspace(x\_min, x\_max, 200),

                         np.linspace(y\_min, y\_max, 200))

    grid = np.c\_[xx.ravel(), yy.ravel()]

    for name, model in models.items():

        Z = model.predict(grid)

        Z = Z.reshape(xx.shape)

        boundaries[name] = {

            "xx": xx.tolist(),

            "yy": yy.tolist(),

            "Z": Z.tolist()

        }

    return {

        "boundaries": boundaries,

        "X": X\_scaled.tolist(),

        "y": y\_all.tolist()

    }