Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки «Анализ данных»

**ЗАДАЧА БИНАРНОЙ КЛАССИФИКАЦИИ ДЛЯ ОПРЕДЕЛЕНИЯ БОЛЬНЫХ ДИАБЕТОМ**

Разработчики проекта:

Баркова Маргарита Олеговна,

Караваева Ксения Алексеевна,

Черткова Софья Михайловна

Пермь, 2024

**Оглавление**

[ПАСПОРТ ПРОЕКТА 3](#_heading=h.30j0zll)

[СОДЕРЖАНИЕ ПРОЕКТА 4](#_heading=h.1fob9te)

[Анализ проблемы исследования 4](#_heading=h.3znysh7)

[Исходные данные](#_heading=h.2et92p0) 6

[Реализация проекта 8](#_heading=h.tyjcwt)

[Этап 1. Подготовка данных к анализу 8](#_heading=h.3dy6vkm)

[Этап 2. Предварительный анализ данных 1](#_heading=h.1t3h5sf)0

[Этап 3. Корреляционный анализ данных](#_heading=h.4d34og8) 19

[Этап 4. Моделирование 2](#_heading=h.2s8eyo1)2

[Этап 5. Прогнозирование](#_heading=h.17dp8vu) 37

[Заключение](#_heading=h.3rdcrjn) 38

[Список использованных источников и литературы 3](#_heading=h.26in1rg)9

[Приложения](#_heading=h.lnxbz9) 41

**ПАСПОРТ ПРОЕКТА**

**Название проекта:**Задача бинарной классификации для определения больных диабетом.

**Сведения об авторах:** Баркова Маргарита Олеговна, Караваева Ксения Алексеевна, Черткова Софья Михайловна

**Цель:**выполнить анализ данных и построить модель для определения больных диабетом и здоровых людей с помощью бинарной классификации.

**Задачи:**

1. Выполнить анализ проблемы, обосновать ее актуальность.
2. Осуществить загрузку данных и подготовку их к анализу количественными методами, включая устранение пропущенных значений.
3. Выполнить предварительный анализ данных, в том числе выявление и обработку выбросов, проверку распределения данных на нормальность, корреляционный анализ.
4. Осуществить моделирование зависимости целевого признака от факторных методом бинарной классификации, в том числе подобрать наилучшую модель, оценить ее качество и выполнить прогнозирование.
5. Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

**Краткое описание проекта:**

Требуется проанализировать данные по заболеванию диабетом и определить, возможно ли описать зависимость данного заболевания от имеющихся факторных переменных методом бинарной классификации. Дать интерпретацию полученным результатам. Сделать выводы.

**Конкретные ожидаемые результаты:**

Построенная модель бинарной классификации.

**СОДЕРЖАНИЕ ПРОЕКТА**

**Анализ проблемы исследования**

Анализ заболевания диабетом является важным направлением медицины.

Диабет является одной из наиболее острых медицинских и социальных проблем современности. По данным Всемирной организации здравоохранения (ВОЗ), в 2021 году количество взрослых с диабетом в мире превысило 530 миллионов человек, и прогнозы показывают, что к 2045 году это число может вырасти до 700 миллионов. Эта динамика обусловлена увеличением числа людей с ожирением, малоподвижным образом жизни и недостаточной информированностью населения о профилактике заболевания.

Диабет сопровождается серьезными осложнениями, такими как сердечно-сосудистые заболевания, поражение почек, потеря зрения и нейропатии. Эти состояния значительно снижают качество жизни пациентов и приводят к повышенной смертности. По оценкам, диабет стал одной из ведущих причин смертности, напрямую или косвенно связанной с болезнью.

Методы машинного обучения (ML) предоставляют уникальные возможности для повышения точности и скорости диагностики диабета. Использование алгоритмов ML позволяет:

* Анализировать большие объемы данных пациентов, включая медицинские карты, результаты анализов и данные о наследственности.
* Разрабатывать модели для прогнозирования риска развития диабета, основываясь на различных факторах риска.
* Обеспечивать персонализированные подходы к лечению и профилактике заболевания.

Машинное обучение также имеет потенциал для снижения финансовой нагрузки на систему здравоохранения. Например, автоматизация процесса диагностики позволяет сократить затраты на ручной анализ данных врачами, а ранняя диагностика может предотвратить развитие дорогостоящих осложнений.

Учитывая масштабы проблемы и финансовые затраты, поиск эффективных решений для раннего выявления диабета имеет критическую важность. Применение машинного обучения в этой области позволяет не только улучшить качество диагностики, но и оптимизировать ресурсы здравоохранения. Внедрение подобных технологий способствует повышению доступности медицинской помощи и снижению смертности от осложнений диабета. Таким образом, исследование применения методов машинного обучения для прогнозирования диабета является важным и перспективным направлением.

Анализ данной ситуации и определяет актуальность темы исследования. Таким образом, необходимо проанализировать данные по заболеванию диабетом и определить, возможно ли описать зависимость данного заболевания от имеющихся факторных переменных.

*Цель:* выполнить анализ данных и построить модель для определения больных диабетом и здоровых людей с помощью бинарной классификации.

*Задачи:*

* Выполнить анализ проблемы, обосновать ее актуальность.
* Осуществить загрузку данных и подготовку их к анализу количественными методами, включая устранение пропущенных значений.
* Выполнить предварительный анализ данных, в том числе выявление и обработку выбросов, проверку распределения данных на нормальность, корреляционный анализ.
* Осуществить моделирование зависимости целевого признака от факторных методом бинарной классификации, в том числе подобрать наилучшую модель, оценить ее качество и выполнить прогнозирование.
* Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

**Исходные данные**

В настоящей работе анализируется список факторов, влияющих на диабет. Данные содержат 70692 наблюдения.

Список колонок анализируемого набора данных:

* + - 1. **Diabetes\_binary** – таргет
      2. **HighBP** - артериальное давление
      3. **HighChol** - холестерин в крови
      4. **CholCheck** - проверка уровня холестина в крови за последние 5 лет
      5. **BMI** - индекс массы тела
      6. **Smoker** - курит или нет
      7. **Stroke** - был инсульт или нет
      8. **HeartDiseaseorAttack** - были ли сердечные заболевания или приступы
      9. **PhysActivity** - занимается спортом или нет
      10. **Fruits** - употребляете ли фрукты 1 или более раз в день
      11. **Veggies** - употребляете ли овощи 1 или более раз в день
      12. **HvyAlcoholConsump** - употребляете ли вы алкоголь
      13. **AnyHealthcare** - есть медицинское страхование или нет
      14. **NoDocbcCost** - были бы случаи, когда вы не могли обратиться к врачу из-за дороговизны
      15. **GenHlth** - оценка вашего здоровья от 1 до 5
      16. **MentHlth** - сколько дней в месяц вы испытывали плохое психическое состояние (от 0 до 30)
      17. **PhysHlth** - сколько дней в месяц вы испытывали плохое физическое состояние (от 0 до 30)
      18. **DiffWalk** - тяжело ли вам ходить, подниматься по лестнице
      19. **Sex** – пол
      20. **Age** – возраст
      21. **Education** - какой у вас уровень образования
      22. **Income** - годовой доход семьи по шкале от 1 до 8

Необходимо проанализировать данные по заболеванию диабетом и определить, возможно ли описать зависимость данного заболевания от имеющихся факторных переменных методом бинарной классификации.

Выдвинем гипотезу исследования: заболевание диабетом зависит от 21 фактора, представленных в выбранных данных, и может быть предсказано с помощью этих показателей.

**Реализация проекта**

**Этап 1. Подготовка данных к анализу**

Загрузим данные в датафрейм и подключим необходимые библиотеки:

import seaborn as sns

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import StandardScaler

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

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

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

import matplotlib.pyplot as plt

Загрузим данные:

data = pd.read\_csv("/content/diabetes\_binary\_5050split\_health\_indicators\_BRFSS2015.csv")

data.head(3)

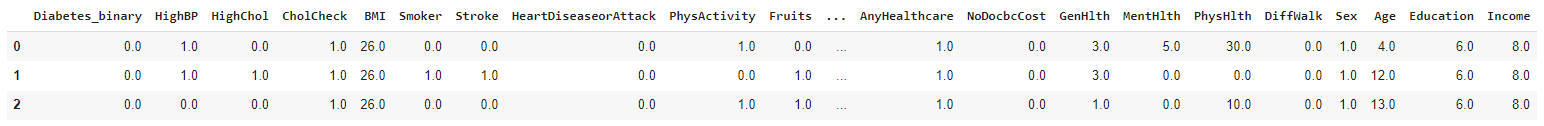


Рисунок 1. Исходный датафрейм

Убедимся, что все количественные столбцы имеют числовой тип. Если это не так, выполним преобразование типа столбца к числовому. Также проверим, есть ли пропуски.

data.info()

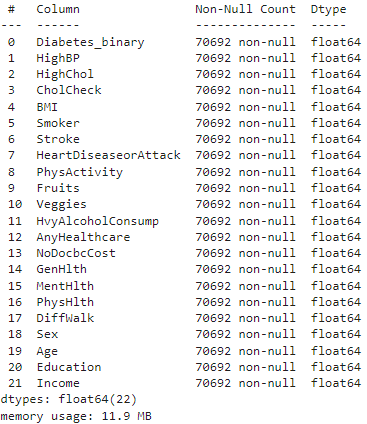


Рисунок 2. Типы данных колонок и количество непропущенных значений

Все столбцы имеют численный тип, пропуски отсутствуют.

Итак, результат первого этапа – это готовый к анализу набор данных в виде датафрейма.

**Этап 2. Предварительный анализ данных**

Вычислим описательные статистики по колонкам (среднее, моду, медиану, стандартное отклонение, квартили).

***Среднее арифметическое*** *равно сумме значений всех вариант выборки, деленной на объем выборки:*

.

Здесь *п* − объем выборки, а *xi* − варианты выборки.

***Модой*** называется значение признака, встречающееся в выборке наиболее часто. Условимся использовать для обозначения моды символы *Mo*.

В случае несгруппированных данных для нахождения медианы необходимо ранжировать выборку, т. е. расположить данные в порядке их возрастания или убывания. Медианой будет являться значение признака, находящееся в середине ранжированного ряда. Медиана находится по формуле

Выборочная дисперсия находится по формуле *.*

Используется также другая формула для вычисления дисперсии: , где *.*

Дисперсия имеет размерность квадрата размерности случайной величины, что затрудняет ее интерпретацию и делает не очень наглядной. Для более наглядного описания рассеяния удобнее пользоваться характеристикой, размерность которой совпадает с размерностью исследуемого признака. С этой целью вводится понятие ***стандартного отклонения*** (или ***среднего квадратичного отклонения***).

***Стандартным отклонением*** называется положительный квадратный корень из дисперсии:

.

Стандартное отклонение имеет те же единицы измерения, что и результаты измерения исследуемого признака, и, таким образом, оно характеризует степень отклонения признака от среднего арифметического. Иными словами, оно показывает, как расположена основная часть вариант относительно среднего арифметического.

data.describe()

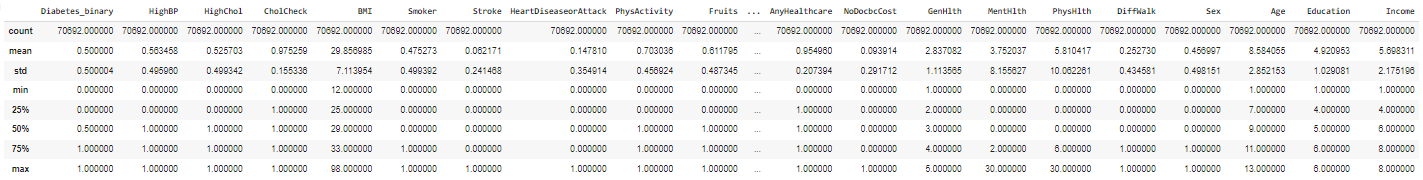


Рисунок 3. Описательные статистики по колонкам

***Аномальными наблюдениями*** (*выбросами*, англ. *Outliers, Extreme values*) называют такие значения уровня временного ряда, которые значительно отличаются от остальных. При выявлении подобных «выбросов» возникают серьезные вопросы: являются ли отклоняющиеся данные действительно ошибками (например, регистрации) или это реальные значения и как получить адекватные оценки для параметров изучаемой совокупности.

Проверим данные на наличие выбросов, для этого можно использовать диаграмму «ящик с усами» (boxplot).

График ***«ящик с усами»,*** или ***«ящичковая диаграмма»***, или ***диаграмма размаха*** − график, используемый в описательной статистике и компактно изображающий одномерное [распределение вероятностей](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9). Такой вид диаграммы в удобной форме показывает медиану, нижний и верхний квартили, минимальное и максимальное значения выборки и [выбросы](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%B1%D1%80%D0%BE%D1%81_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0)).

Расстояния между различными частями ящика позволяют определить степень распространения (дисперсии) и асимметрии в данных и выявить выбросы.

Границами ящика служат первый и третий [квартили](https://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D1%80%D1%82%D0%B8%D0%BB%D1%8C) (25-й и 75-й [процентили](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%BD%D1%82%D0%B8%D0%BB%D1%8C) соответственно), линия в середине ящика — медиана (50-й процентиль). Концы усов — края статистически значимой выборки (без [выбросов](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%B1%D1%80%D0%BE%D1%81_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0))) могут определяться несколькими способами. В общем виде эта формула имеет вид

*.*

*X*н — нижняя граница уса, *X*в — верхняя граница уса, *Q*1 — первый квартиль ,*Q*3 — третий квартиль, *k* — коэффициент, наиболее часто употребляемое значение которого равно 1,5. Данные, выходящие за границы усов ([выбросы](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D0%B1%D1%80%D0%BE%D1%81_(%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0))), отображаются на графике в виде точек, маленьких кружков или звёздочек. Иногда на графике отмечают среднее арифметическое и его [доверительный интервал](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%B2%D0%B5%D1%80%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%B2%D0%B0%D0%BB_%D0%B4%D0%BB%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B6%D0%B8%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B9_%D0%B2%D1%8B%D0%B1%D0%BE%D1%80%D0%BA%D0%B8) («зарубка» на ящике). На рис. изображен график «ящик с усами».

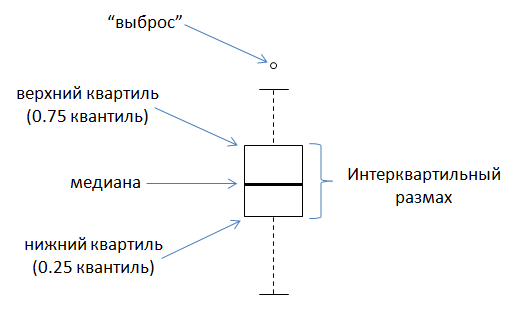


Рисунок 4. «Ящик с усами»

Посмотрим на диаграммы boxplot всех числовых колонок датафрейма:

data.boxplot();

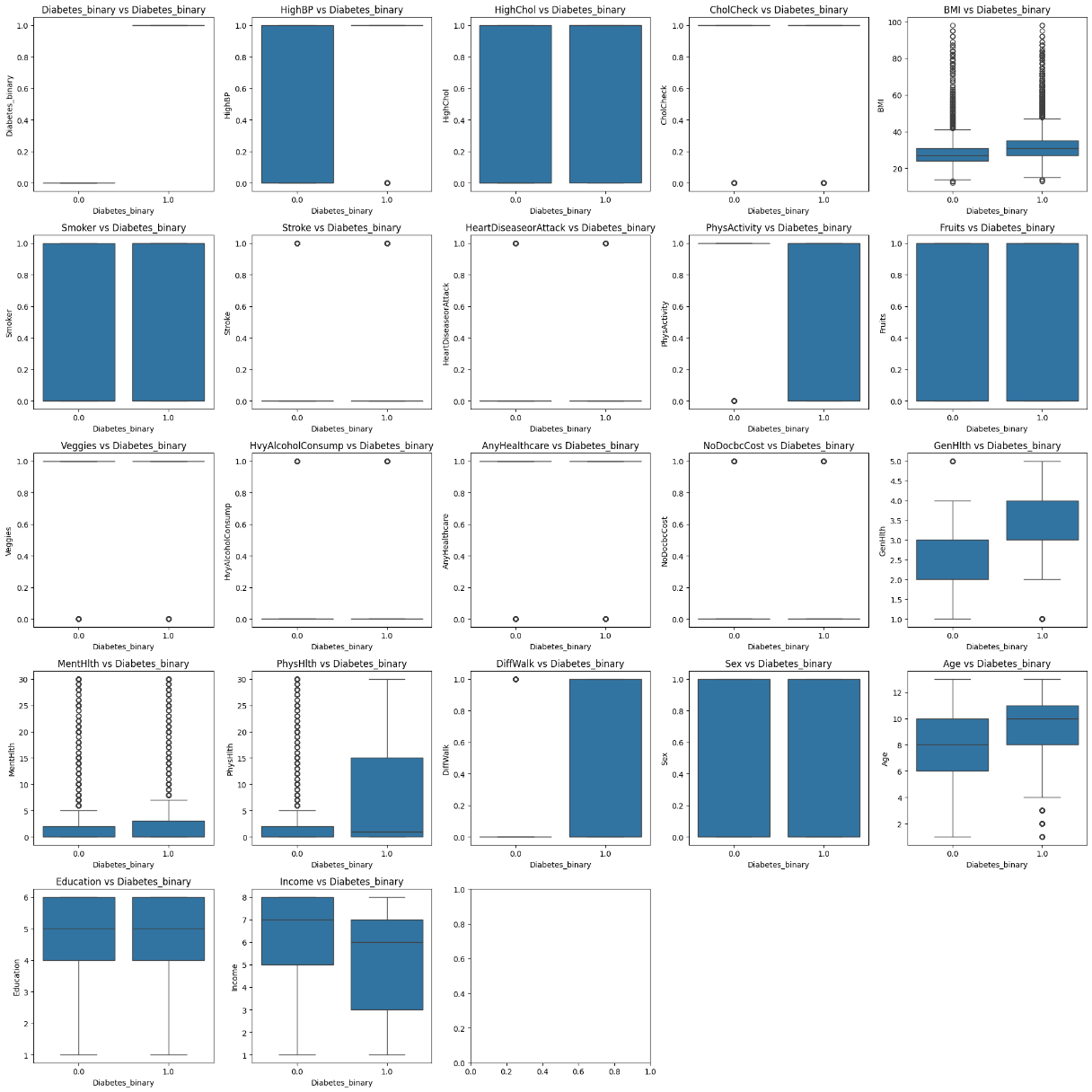


Рисунок 5. Диаграммы boxplot всех колонок

Заметно, что есть явные выбросы в трех колонках – *BMI, MentHlth, PhysHlth*. Обработаем выбросы:

def replace\_outliers\_iqr(df, col, multiplier = 1.5):

Q1 = df[col].quantile(0.25)

Q3 = df[col].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - multiplier \* IQR

upper\_bound = Q3 + multiplier \* IQR

df[col] = np.where(df[col] < lower\_bound, lower\_bound, df[col])

df[col] = np.where(df[col] > upper\_bound, upper\_bound, df[col])

return df

for col in data.columns:

data = replace\_outliers\_iqr(data, 'BMI', multiplier=1.5)

data = replace\_outliers\_iqr(data, 'MentHlth', multiplier=1.5)

data = replace\_outliers\_iqr(data, 'PhysHlth', multiplier=1.5)

После обработки снова посмотрим на выбросы:

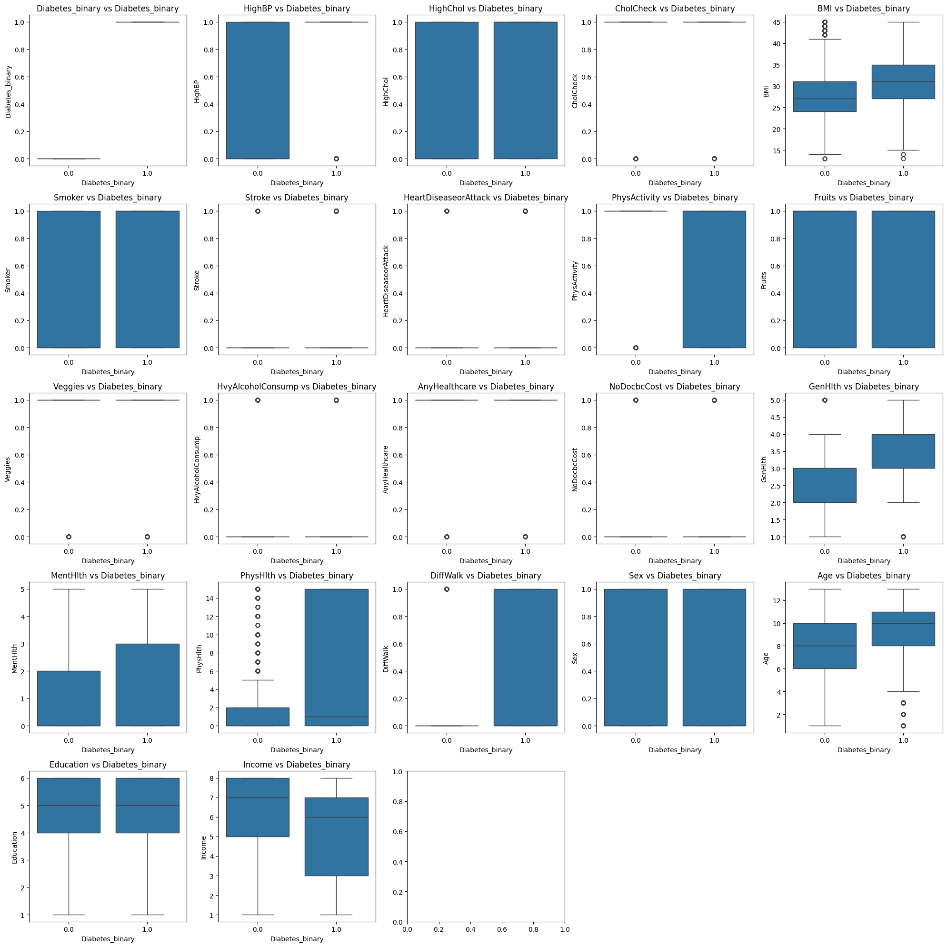


Рисунок 6. Диаграммы boxplot всех колонок

От наиболее заметных выбросов удалось избавиться.

Проверим данные на нормальность распределения двумя способами:

* 1. Построим гистограмму и сделать предположение о том, являются ли данные нормально распределенными.
  2. Выполним статистический тест на нормальность и убедимся, что выдвинутое ранее предположение о нормальности верно или ошибочно.

***Гистограмма****,* представляющая собой совокупность примыкающих друг к другу прямоугольников, основание каждого из которых равно ширине интервала группировки, а площадь − частости этого интервала.

Гистограмма строится в декартовой (прямоугольной) системе координат следующим образом. По оси абсцисс откладываются отрезки, отображающие интервалы группировки, а затем на каждом из них строится прямоугольник, площадь которого равна частости данного интервала. В случае если все интервалы группировки имеют одинаковую ширину, высоты прямоугольников пропорциональны соответствующим частостям.

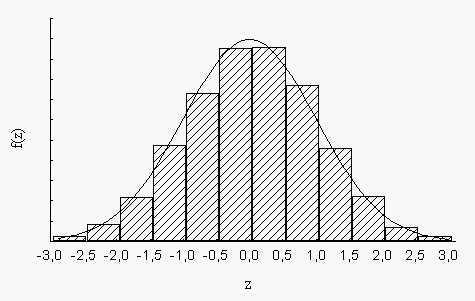


Рисунок 7. Гистограмма

Сначала построим гистограммы для всех колонок:

data.hist(figsize=(20,15))

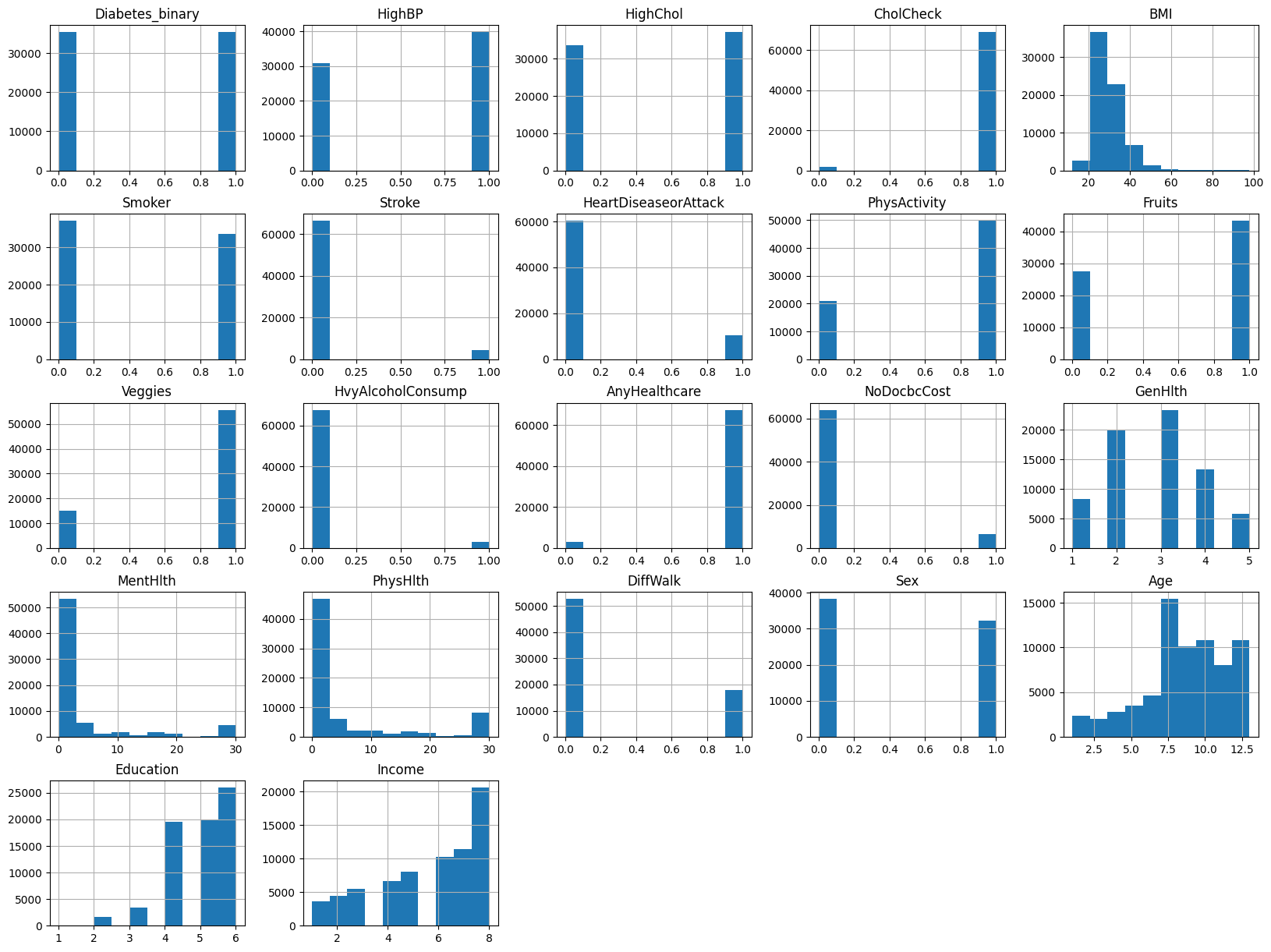


Рисунок 8. Гистограммы для всех колонок

Визуально не видно, что тут есть нормально распределенные колонки, поэтому применим статистические тесты (критерии согласия).

***Критерии согласия з***аключаются в проверке предположения о том, что результаты наблюдений могут быть описаны с помощью определенного закона распределения (в нашем случае нормального распределения). **П**ри малом числе измерений часто используется критерий Шапиро−Уилка.

На формальном языке проверяется гипотеза:

H0: наши данные согласуются с нормальным распределением.

Если p-value меньше заданного уровня значимости (обычно 0,05 или 0,01), то основная гипотезу отвергается.

Используем тест Шапиро-Уилка:

import scipy.stats as stats

for col in df.columns[:-1]:

print(col, stats.shapiro(df[col]))

Результаты теста для колонок:

Diabetes\_binary ShapiroResult(statistic=0.6366224260750695, pvalue=1.935045727506944e-141)

HighBP ShapiroResult(statistic=0.6307424654727146, pvalue=6.219384256707285e-142)

HighChol ShapiroResult(statistic=0.6356614311673741, pvalue=1.6056227416636577e-141)

CholCheck ShapiroResult(statistic=0.13910956884100023, pvalue=3.6958770442485656e-169)

BMI ShapiroResult(statistic=0.8994835344464236, pvalue=8.561231464319347e-105)

Smoker ShapiroResult(statistic=0.6357330813248738, pvalue=1.6280949282298157e-141)

Stroke ShapiroResult(statistic=0.25727821590899824, pvalue=3.0948160301083957e-164)

HeartDiseaseorAttack ShapiroResult(statistic=0.42318075864977234, pvalue=5.606554452775773e-156)

PhysActivity ShapiroResult(statistic=0.5737081100670253, pvalue=2.209418210296378e-146)

Fruits ShapiroResult(statistic=0.6181959145262963, pvalue=5.819128217220532e-143)

Veggies ShapiroResult(statistic=0.5019478666672019, pvalue=2.7931259716782168e-151)

HvyAlcoholConsump ShapiroResult(statistic=0.20201688523758565, pvalue=1.2792219386198115e-166)

AnyHealthcare ShapiroResult(statistic=0.20918256470314933, pvalue=2.555687943354081e-166)

NoDocbcCost ShapiroResult(statistic=0.330066779655831, pvalue=7.692950081514783e-161)

GenHlth ShapiroResult(statistic=0.9148903266677716, pvalue=1.861457091661787e-100)

MentHlth ShapiroResult(statistic=0.5173703217600031, pvalue=2.7732320496942164e-150)

PhysHlth ShapiroResult(statistic=0.6130174518593317, pvalue=2.2333266606087133e-143)

DiffWalk ShapiroResult(statistic=0.5408747609502548, pvalue=1.0433751749517824e-148)

Sex ShapiroResult(statistic=0.6339287988543469, pvalue=1.1481472198842641e-141)

Age ShapiroResult(statistic=0.9567705212720113, pvalue=9.246336311933777e-84)

Education ShapiroResult(statistic=0.8456924272313249, pvalue=2.132924235160923e-116)

Как видим, p-value везде меньше 0,01, поэтому на уровне значимости 1% гипотеза о нормальности (H0) отвергается для всех колонок, что соответствует предположениям, сделанным после визуального анализа.

**Этап 3. Корреляционный анализ данных**

***Корреляционный анализ*** – это совокупность методов оценивания степени тесноты статистической связи между анализируемыми переменными.

Выполним корреляционный анализ данных с помощью матрицы корреляции. В случае нормальности всех данных следует использовать коэффициент корреляции Пирсона, в противном случае – ранговые коэффициенты корреляции Спирмена.

**Парный коэффициент** корреляции характеризует взаимосвязь двух переменных на фоне действия остальных показателей и является самым распространенным показателем тесноты связи при статистическом анализе данных.

Парный коэффициент корреляции между количественными случайными переменными и носит название *выборочного коэффициента корреляции* Пирсона (*sample correlation coefficient*) (или просто коэффициента корреляции) и находится по формуле

где и  — *выборочные дисперсии (sample variances*) переменных  и , а — *выборочная ковариация* или выборочный ковариационный момент, и соответствующие *средние (means)* определяются по формулам



Коэффициент корреляции обладает следующими свойствами:

1. Принимает значения от –1 до +1.
2. Если , то связь между переменными  и  считается сильной. Если , то связь слабая.
3. Если , то корреляционное поле наблюдений представляет собой совокупность точек, которые можно расположить на одной прямой. Знак «+» свидетельствует о прямой линейной зависимости между переменными и , а знак «—» − об обратной линейной зависимости.
4. При  линейная корреляционная связь отсутствует.

Метод ранговой корреляции Спирмена позволяет определить тесноту (силу) и направление корреляционной связи между двумя признаками (как количественными, так и качественными). Коэффициент ранговой корреляции имеет границы изменения от –1 до +1. Полное совпадение рангов означает максимально тесную прямую связь, полная противоположность рангов – максимально тесную обратную связь. Формула расчета ***коэффициента корреляции рангов Ч. Спирмена:***

где  – ранг  в выборке .

Матрицу корреляции отобразим с помощью диаграммы «тепловая карта» (heatmap). Данные всех колонок не имеют нормального распределения, поэтому используем ранговый коэффициент Спирмена.

plt.figure(figsize = (20,10))

corr\_matrix = data.corr(method='spearman')

sns.heatmap(corr\_matrix, annot=True , cmap ='YlOrRd' )

plt.title("Heatmap")

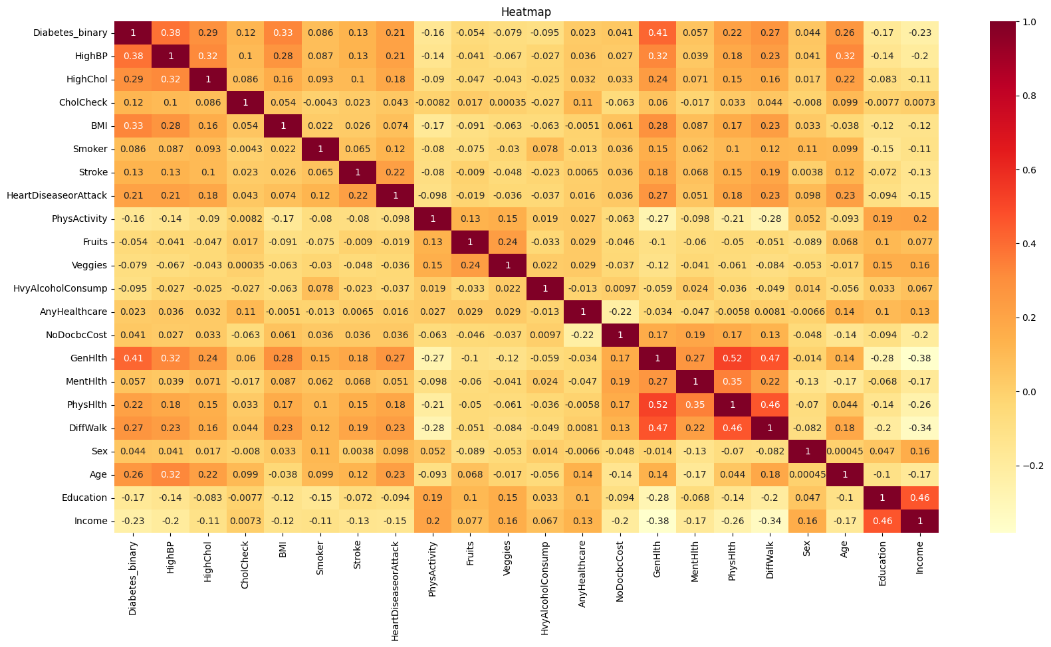


Рисунок 9. Тепловая карта матрицы корреляции

Определим наиболее сильные корреляции между признаками, между таргетом и признаками:

strong\_corrs = corr\_matrix[(corr\_matrix.abs() > 0.465) & (corr\_matrix != 1)]

strong\_corr\_pairs = strong\_corrs.stack().index.tolist()

print("Наиболее сильные корреляции между признаками (положительные):")

for feature1, feature2 in strong\_corr\_pairs:

print(f"{feature1} - {feature2}: {corr\_matrix.loc[feature1, feature2]}")

strong\_corrs = corr\_matrix[(corr\_matrix < -0.3) & (corr\_matrix != 1)]

strong\_corr\_pairs = strong\_corrs.stack().index.tolist()

print("\nНаиболее сильные корреляции между признаками (отрицательные):")

for feature1, feature2 in strong\_corr\_pairs:

print(f"{feature1} - {feature2}: {corr\_matrix.loc[feature1, feature2]}")

corr\_with\_target = corr\_matrix['Diabetes\_binary'].drop('Diabetes\_binary')

sorted\_corr\_with\_target = corr\_with\_target.abs().sort\_values(ascending=False)

print("\nНаиболее сильные корреляции между таргетом и признаками:")

for feature, corr\_value in list(sorted\_corr\_with\_target.items())[:3]:

print(f"{feature}: {corr\_matrix.loc[feature, 'Diabetes\_binary']}")

Получим следующие результаты:

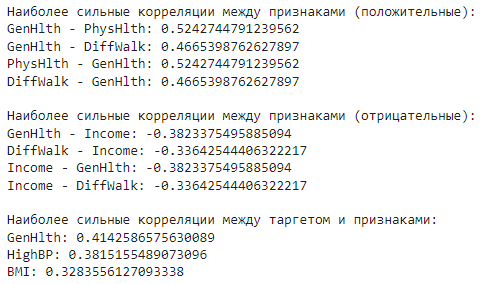


Рисунок 10. Наиболее сильные корреляции

**Этап 4. Моделирование**

Масштабируем данные, используя MinMaxScaler:

y = data.pop('Diabetes\_binary')

df = data

scaler = StandardScaler()

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

Разобьём данные на обучающую и тестовую выборки, используя соотношение 7 к 3:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df\_scaled, y, test\_size=0.3, random\_state=17, stratify=y)

Применяем следующие алгоритмы:

* ***Logistic Regression (логистическая регрессия)***

Простой, интерпретируемый алгоритм для бинарной классификации, использующий сигмоидную функцию для оценки вероятности принадлежности объекта к одному из двух классов. Он моделирует зависимость между признаками и вероятностью с помощью линейной комбинации, что делает его эффективным для задач с линейно разделимыми данными.

* ***Random Forest (случайный лес)***

Ансамблевый метод, использующий множество решающих деревьев, обученных на случайных подвыборках данных и признаков. Это позволяет ему хорошо справляться с нелинейными зависимостями и предотвращать переобучение, что делает его универсальным решением для классификации и регрессии.

* ***Support Vector Machine (машина опорных векторов, SVM)***

Мощный алгоритм для классификации, который строит гиперплоскость, максимально разделяющую классы. SVM использует ядровые функции для работы с нелинейно разделимыми данными и эффективен в задачах классификации с высокой размерностью.

* ***K-Nearest Neighbors (k ближайших соседей, KNN)***

Простой алгоритм классификации (регрессии), основанный на оценивании сходства объектов. Суть метода проста: объект относится к тому классу, к которому принадлежит большинство из его ближайших k соседей.

* ***Gradient Boosting (градиентный бустинг)***

Ансамблевый алгоритм, строящий модель итеративно, добавляя новые слабые классификаторы, которые обучаются на ошибках предыдущих. Это обеспечивает высокую точность, но требует настройки гиперпараметров и вычислительных ресурсов, делая его мощным, но сложным решением для различных задач.

Для оценки качества моделей рассматриваем следующие метрики:

* ***precision (точность)*:** доля верно классифицированных объектов данного класса среди всех объектов, отнесенных к этому классу.

Формула: TP / (TP + FP), где TP - True Positive (истинно положительные), FP - False Positive (ложно положительные).

* ***recall (полнота)*:** доля верно классифицированных объектов данного класса среди всех объектов этого класса.

Формула: TP / (TP + FN), где FN - False Negative (ложно отрицательные).

* ***f1-score (F1-мера)*:** гармоническое среднее между точностью и полнотой.

Формула: 2 \* (precision \* recall) / (precision + recall)

* ***accuracy (общая точность)*:** доля верно классифицированных объектов среди всех объектов.

Формула: (TP + TN) / (TP + TN + FP + FN), где TN - True Negative (истинно отрицательные).

Кроме этого, будем строить ***матрицу ошибок*** и ***ROC-кривую***.

Матрица ошибок (Confusion Matrix) — это инструмент, используемый в машинном обучении для оценки производительности модели классификации, особенно в задачах, где нужно предсказывать, к какому из нескольких классов относится тот или иной объект. Она представляет собой таблицу, в которой отражаются результаты работы модели, сравнивая истинные метки классов с предсказанными.

Матрица ошибок показывает, где именно модель допускает ошибки, а не просто общее количество правильных и неправильных предсказаний. Это особенно полезно при работе с несбалансированными классами, когда общая точность (accuracy) может ввести в заблуждение.

ROC-кривая (Receiver Operating Characteristic curve) и площадь под кривой (AUC, Area Under the Curve) являются важными инструментами для оценки качества моделей классификации, особенно в задачах, связанных с бинарной классификацией.

ROC-кривая представляет собой график, который отображает соотношение между истинными положительными результатами (True Positive Rate, TPR) и ложными положительными результатами (False Positive Rate, FPR) при различных порогах классификации.

График ROC-кривой строится путем изменения порога классификации и вычисления TPR и FPR для каждого порога. В результате получается кривая, которая показывает, как изменяются эти метрики в зависимости от выбранного порога.

Площадь под ROC-кривой (AUC) — это числовая метрика, которая определяет качество модели. AUC принимает значения от 0 до 1:

• AUC = 0.5: модель не лучше случайного выбора. Это означает, что модель не может различить положительные и отрицательные классы;

• AUC < 0.5: модель работает хуже случайного выбора, что указывает на необходимость доработки модели;

• AUC = 1.0: модель идеально различает классы, правильно классифицируя все положительные и отрицательные случаи.

Чем выше значение AUC, тем лучше модель справляется с задачей классификации.

Начнем с логистической регрессии. Обучаем модель на тренировочных данных и предсказываем на тестовых.

logistic\_model = LogisticRegression() #Инизиализация модели

logistic\_params = {"C": [0.1, 1, 10], "penalty": ["l2"], "solver": ["lbfgs"]} #Параметры для GridSearchCV

logistic\_grid = GridSearchCV(logistic\_model, logistic\_params, cv=5, scoring="accuracy", n\_jobs=-1) #Использование GridSearchCV

logistic\_grid.fit(X\_train, y\_train)

best\_logistic = logistic\_grid.best\_estimator\_ #Обучаем модель с лучшими параметрами

y\_pred\_logistic = best\_logistic.predict(X\_test) #Предскзываем на тесте

Выводим метрики по классам:

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_logistic, target\_names=target\_names))

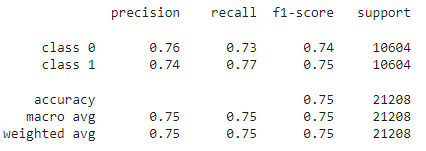


Рисунок 11. Метрики по классам (логистическая регрессия)

Значения метрик для class 0 и class 1 близки друг к другу, что означает, что модель классифицирует оба класса примерно с одинаковой эффективностью. F1-мера объединяет точность и полноту. 0.74 и 0.75 указывают на сбалансированную производительность модели в отношении обоих классов.

Выводим метрики для модели:

print("Logistic Regression Metrics:")

print(f"Best Parameters: {logistic\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_logistic):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_logistic):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_logistic):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_logistic):.4f}")

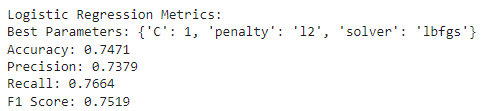


Рисунок 12. Метрики модели (логистическая регрессия)

Accuracy (Точность): модель правильно классифицирует примерно 75% всех объектов.

Precision (Точность): из предсказанных положительных примерно 74% верны.

Recall (Полнота): модель находит примерно 77% истинно положительных объектов.

F1 Score: 75.19%. Сбалансированная оценка точности и полноты.

Модель показывает умеренную производительность. Все метрики находятся в районе 75%, что указывает на то, что модель достаточно неплохо справляется с предсказанием.



Рисунок 13. ROC-кривая (логистическая регрессия)

ROC-кривая показывает хорошую работу модели логистической регрессии. Значение AUC равно 0.8237, что говорит о хорошей способности модели различать классы.

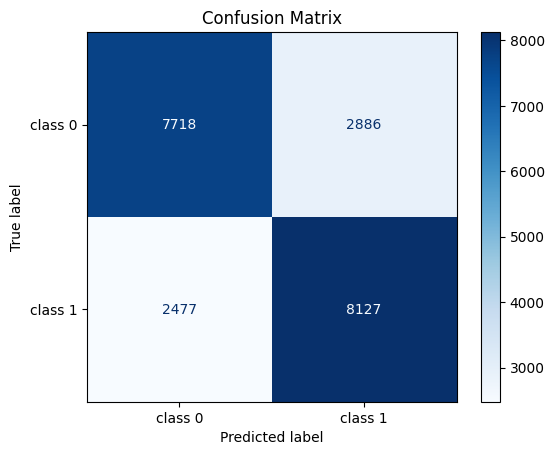


Рисунок 14. Матрица ошибок (логистическая регрессия)

Модель верно классифицирует большую часть объектов из обоих классов (7718 для class 0, и 8127 для class 1). Она делает больше ошибок типа False Positives (2886), чем False Negatives (2477).

Теперь рассмотрим случайный лес. Обучаем модель на тренировочных данных и предсказываем на тестовых.

rf\_model = RandomForestClassifier()

rf\_params = {"n\_estimators": [100, 200], "max\_depth": [10, 20, None], "min\_samples\_split": [2, 5]}

rf\_grid = GridSearchCV(rf\_model, rf\_params, cv=5, scoring="accuracy", n\_jobs=-1)

rf\_grid.fit(X\_train, y\_train)

best\_rf = rf\_grid.best\_estimator\_

y\_pred\_rf = best\_rf.predict(X\_test)

Выводим метрики по классам:

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_rf, target\_names=target\_names))

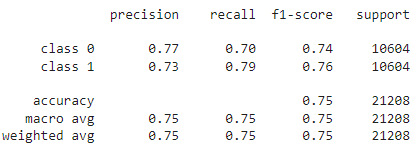


Рисунок 15. Метрики по классам (случайный лес)

Для class 0 точность выше, чем для class 1, полнота, наоборот. F1-мера для class 1 стала выше в сравнении с логистической регрессией.

Выводим метрики для модели:

print("Random Forest Metrics:")

print(f"Best Parameters: {rf\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_rf):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_rf):.4f}")

print(f"ыRecall: {recall\_score(y\_test, y\_pred\_rf):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_rf):.4f}")

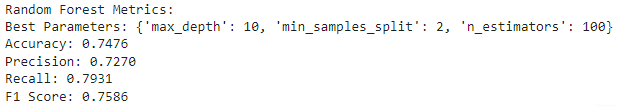


Рисунок 16. Метрики модели (случайный лес)

Все метрики, кроме precision, выросли в сравнении с логистической регрессией. Модель показывает умеренную производительность и достаточно неплохо справляется с предсказанием.

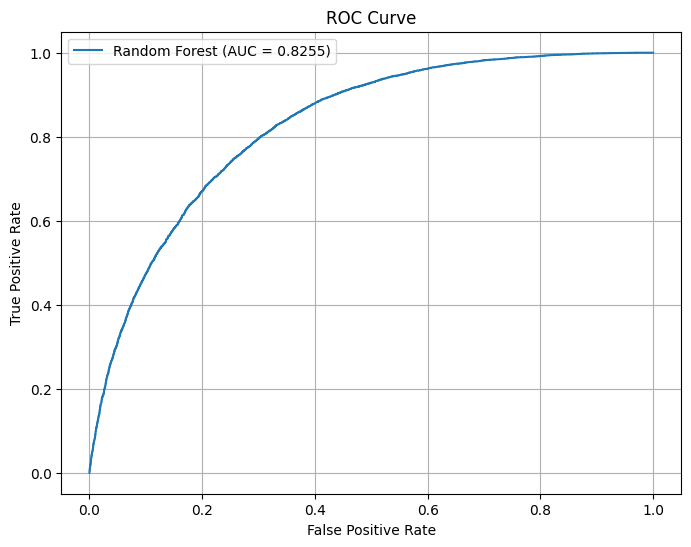


Рисунок 17. ROC-кривая (случайный лес)

ROC-кривая показывает хорошую работу модели случайного леса. Значение AUC выросло в сравнении с логистической регрессией.

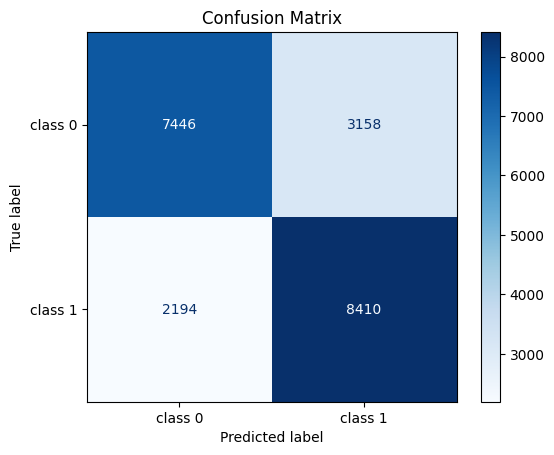


Рисунок 18. Матрица ошибок (случайный лес)

Для class 0 число верно классифицируемых уменьшилось, для class 1 – увеличилось в сравнении с логистической регрессией. Количество FP – увеличилось, FN – уменьшилось.

Перейдем к методу опорных векторов. Обучаем модель на тренировочных данных и предсказываем на тестовых.

svm\_model = SVC()

svm\_params = {"C": [0.1, 1, 10], "kernel": ["linear", "rbf"]}

svm\_grid = GridSearchCV(svm\_model, svm\_params, cv=5, scoring="accuracy", n\_jobs=-1)

svm\_grid.fit(X\_train, y\_train)

best\_svm = svm\_grid.best\_estimator\_

y\_pred\_svm = best\_svm.predict(X\_test)

Выводим метрики по классам:

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_svm, target\_names=target\_names))

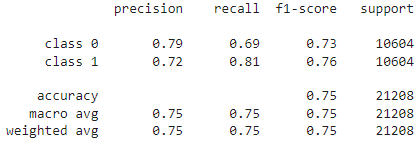


Рисунок 19. Метрики по классам (метод опорных векторов)

Для class 0 точность выше, чем для class 1, полнота, наоборот. F1-мера для class 1 выше, чем для class 0.

Выводим метрики для модели:

print("SVM Metrics:")

print(f"Best Parameters: {svm\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_svm):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_svm):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_svm):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_svm):.4f}")

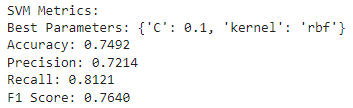


Рисунок 20. Метрики модели (метод опорных векторов)

Все метрики, кроме precision, выросли в сравнении с ранее рассмотренным моделями. Модель показывает умеренную производительность и достаточно неплохо справляется с предсказанием.

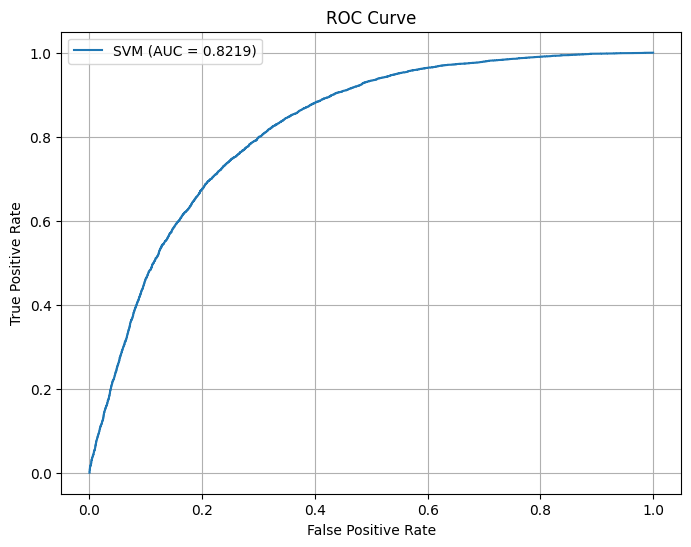


Рисунок 21. ROC-кривая (метод опорных векторов)

ROC-кривая показывает хорошую работу модели метода опорных векторов. Значение AUC уменьшилось в сравнении с предыдущими моделями.

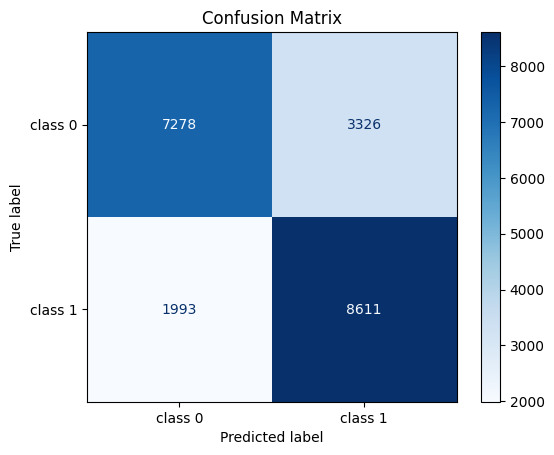


Рисунок 22. Матрица ошибок (метод опорных векторов)

Для class 0 число верно классифицируемых уменьшилось, для class 1 – увеличилось в сравнении с предыдущими моделями. Количество FP – увеличилось, FN – уменьшилось.

Перейдем к методу k-ближайших соседей. Обучаем модель на тренировочных данных и предсказываем на тестовых.

knn\_model = KNeighborsClassifier()

knn\_params = {"n\_neighbors": [3, 5, 7], "weights": ["uniform", "distance"]}

knn\_grid = GridSearchCV(knn\_model, knn\_params, cv=5, scoring="accuracy", n\_jobs=-1)

knn\_grid.fit(X\_train, y\_train)

best\_knn = knn\_grid.best\_estimator\_

y\_pred\_knn = best\_knn.predict(X\_test)

Выводим метрики по классам:

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_knn, target\_names=target\_names))

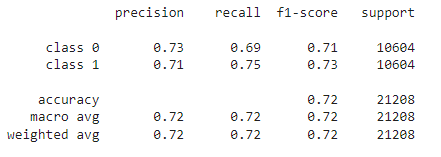


Рисунок 23. Метрики по классам (kNN)

Для class 0 точность выше, чем для class 1, полнота, наоборот. F1-мера для class 1 выше, чем для class 0. Значения ухудшились в сравнении с предыдущими моделями.

Выводим метрики для модели:

print("KNN Metrics:")

print(f"Best Parameters: {knn\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_knn):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_knn):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_knn):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_knn):.4f}")

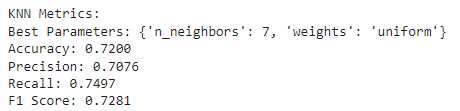


Рисунок 24. Метрики модели (kNN)

Все метрики уменьшились в сравнении с ранее рассмотренным моделями.

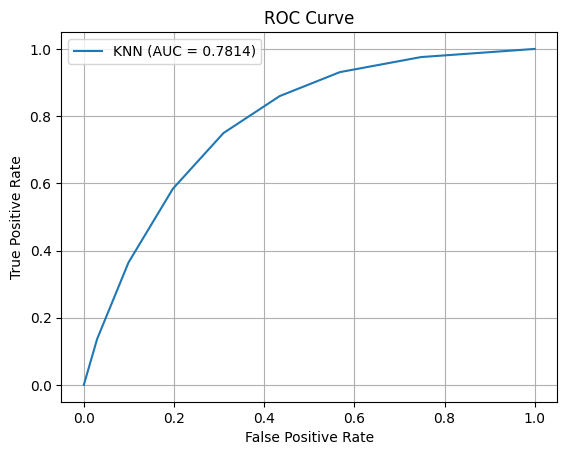


Рисунок 25. ROC-кривая (kNN)

ROC-кривая показывает неплохую работу модели kNN. Значение AUC уменьшилось в сравнении с предыдущими моделями.

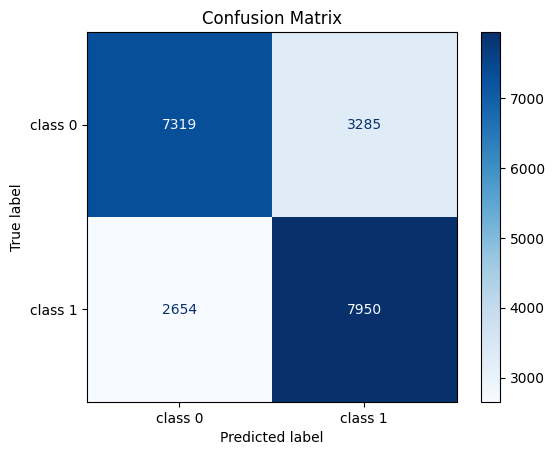


Рисунок 26. Матрица ошибок (kNN)

Модель верно классифицирует большую часть объектов из обоих классов (7319 для class 0, и 7950 для class 1). Она делает больше ошибок типа False Positives (3285), чем False Negatives (2654).

Рассмотри градиентный бустинг. Обучаем модель на тренировочных данных и предсказываем на тестовых.

gb\_model = GradientBoostingClassifier()

gb\_params = {"n\_estimators": [100, 200], "learning\_rate": [0.01, 0.1, 0.2], "max\_depth": [3, 5]}

gb\_grid = GridSearchCV(gb\_model, gb\_params, cv=5, scoring="accuracy", n\_jobs=-1)

gb\_grid.fit(X\_train, y\_train)

best\_gb = gb\_grid.best\_estimator\_

y\_pred\_gb = best\_gb.predict(X\_test)

Выводим метрики по классам:

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_gb, target\_names=target\_names))

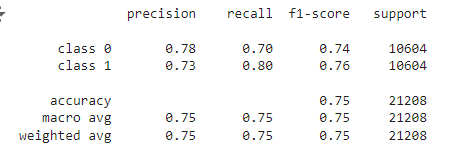


Рисунок 27. Метрики по классам (градиентный бустинг)

Для class 0 точность выше, чем для class 1, полнота, наоборот. F1-мера для class 1 выше, чем для class 0.

Выводим метрики для модели:

print("Gradient Boosting Metrics:")

print(f"Best Parameters: {gb\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_gb):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_gb):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_gb):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_gb):.4f}")

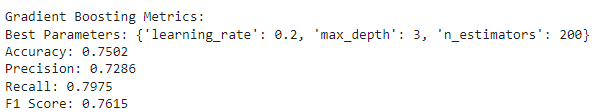


Рисунок 28. Метрики модели (градиентный бустинг)

Метрики обладают хорошими значениями: точность (accuracy) лучшая среди всех моделей, F1-мера близка к лучшей. Модель показывает умеренную производительность и достаточно неплохо справляется с предсказанием.

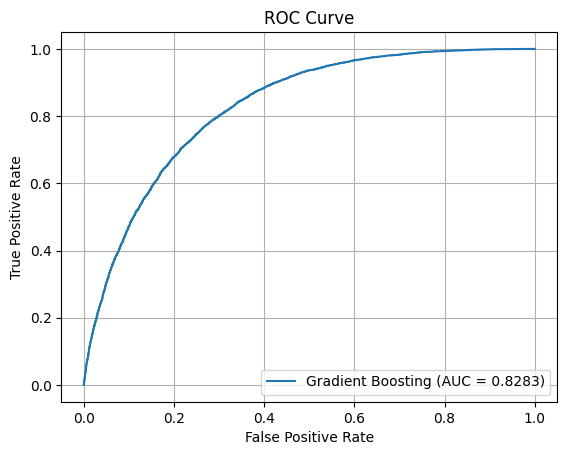


Рисунок 29. ROC-кривая (градиентный бустинг)

ROC-кривая показывает хорошую работу модели метода опорных векторов. Значение AUC увеличилось в сравнении с предыдущими моделями.

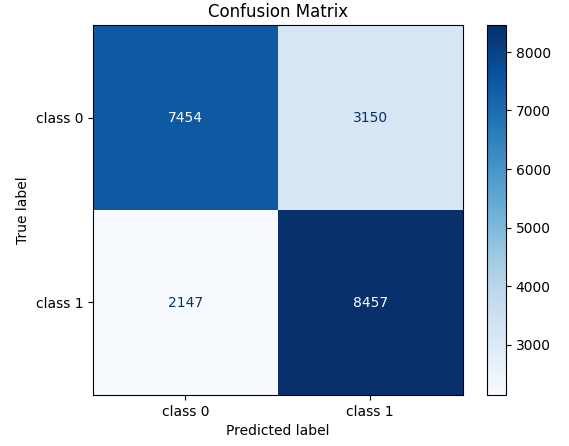


Рисунок 30. Матрица ошибок (градиентный бустинг)

Модель верно классифицирует большую часть объектов из обоих классов (7454 для class 0, и 8457 для class 1). Она делает больше ошибок типа False Positives (3150), чем False Negatives (2147).

Соберем все результаты:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Модель**  **Метрика** | **LR** | **RF** | **SVM** | **KNN** | **GB** |
| **Accuracy** | 0.7471 | 0.7476 | 0.7492 | 0.7200 | 0.7502 |
| **Precision** | 0.7379 | 0.7270 | 0.7214 | 0.7076 | 0.7286 |
| **Recall** | 0.7664 | 0.7931 | 0.8121 | 0.7497 | 0.7975 |
| **F1-мера** | 0.7519 | 0.7586 | 0.7640 | 0.7281 | 0.7615 |
| **AUC** | 0.8237 | 0.8255 | 0.8219 | 0.7814 | 0.8283 |

*Градиентный бустинг (GB)*, является наиболее эффективной моделью среди протестированных, показав лучшие результаты по точности и AUC.

*Модель SVM* демонстрирует очень хорошие полноту и F1-меру, что может быть важным, если нужно найти как можно больше положительных примеров.

*Метод k-ближайших соседей (KNN)* показал наихудшие результаты по всем метрикам, возможно из-за сложности задачи или неподходящих параметров.

Разница между моделями не очень большая. Для того, чтобы выбрать наилучшую, следует сфокусироваться на том, что важнее (точность или полнота).

**Этап 5. Прогнозирование**

Этап прогнозирования был выполнен на этапе моделирования. Все метрики, ROC-кривые и матрицы ошибок получены на тестовой выборке.

**Заключение**

На основе проведенного анализа можно сделать вывод, что выдвинутая гипотеза подтвердилась: заболевание диабетом зависит от 21 фактора, представленных в выбранных данных, и может быть предсказано с помощью этих показателей.

Градиентный бустинг и метод опорных векторов показали лучшие результаты. Градиентный бустинг лидирует по точности и AUC, метод опорных векторов – по полноте и F1-мере.

Таким образом, поставленная цель была достигнута. Был выполнен анализ данных и построить модель для определения больных диабетом и здоровых людей с помощью бинарной классификации.

Для достижения цели были решены следующие задачи:

* Выполнить анализ проблемы, обосновать ее актуальность.
* Осуществить загрузку данных и подготовку их к анализу количественными методами, включая устранение пропущенных значений.
* Выполнить предварительный анализ данных, в том числе выявление и обработку выбросов, проверку распределения данных на нормальность, корреляционный анализ.
* Осуществить моделирование зависимости целевого признака от факторных методом бинарной классификации, в том числе подобрать наилучшую модель, оценить ее качество и выполнить прогнозирование.
* Выполнить интерпретацию полученных результатов и сделать выводы о достижении цели.

Построенная модель бинарной классификации описывает зависимость заболевания диабетом и может применяться для его прогнозирования, что может быть полезно в рамках улучшения диагностики, профилактики и управления заболеванием.

**Список использованных источников и литературы**

1. Амоа К.А. Разработка программных пакетов на языке Python: учебное пособие / К.А. Амоа, Н.А. Рындин, Ю.С. Скворцов. – Воронеж: Воронежский государственный технический университет, ЭБС АСВ, 2020. – 61 c. // Электронно-библиотечная система IPR BOOKS: [сайт].
2. Вандер П. Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.
3. Васильев, А.Н. Программирование на Python в примерах и задачах / А,Н. Васильев. — Москва : Эксмо, 2021. — 616 с.
4. Волкова В.М., Программные системы статистического анализа. Обнаружение закономерностей в данных с использованием системы R и языка Python [Электронный ресурс]: учебное пособие / Волкова В.М. - Новосибирск : Изд-во НГТУ, 2017. - 74 с. - ISBN 978-5-7782-3183-2 - Режим доступа: http://www.studentlibrary.ru/book/ISBN9785778231832.html
5. Гуриков С. Р. Основы алгоритмизации и программирования на Python : учеб. пособие / С.Р. Гуриков. ? М. : ФОРУМ : ИНФРА-М, 2018. ? 343 с. ? (Высшее образование: Бакалав-риат). - Режим доступа: http://znanium.com/catalog/product/924699
6. Криволапов С.Я. Математика на Python : учебник / С.Я. Криволапов, М.Б. Хрипунова. — Москва: КНОРУС, 2022. — 456 с.
7. Маккини У. Python и анализ данных / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 540 с.
8. Саммерфилд М., Python на практике [Электронный ресурс] / Марк Саммерфилд - М. : ДМК Пресс, 2014. - 338 с. - ISBN 978-5-97060-095-5 - Режим доступа: http://www.studentlibrary.ru/book/ISBN9785970600955.html
9. Сузи Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. – 3-е изд. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. – 350 c. // Электронно-библиотечная система IPR BOOKS: [сайт].
10. Федин, Ф. О. Анализ данных. Часть 1. Подготовка данных к анализу : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический универ-ситет, 2012. — 204 c. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
11. Федин, Ф. О. Анализ данных. Часть 2. Инструменты DataMining : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический университет, 2012. — 308 c. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
12. Федоров Д.Ю. Программирование на языке высокого уровня Python: учебное пособие / Д.Ю. Федоров. – 2-е изд.– М.: Юрайт, 2020. – 161 с.
13. Филлипс Т. Управление на основе данных. Как интерпретировать цифры и принимать качественные решения в бизнесе. – М.:Манн, Иванов и Фербер, 2017. - 117 с.
14. Фрэнкс, Б. Революция в аналитике: как в эпоху BigData улучшить ваш бизнес с помощью операционной аналитики / Б. Фрэнкс; Пер. с англ. И. Евстигнеевой; Ред. В. Мылов. – М.: Альпина Паблишер, 2016. – 315 с.

**Приложения**

Приложение 1

Программный код

<https://colab.research.google.com/drive/1XVU9G2Ai5-FrgMMc7wirzymcxTeQArRu>

**Импорт библиотек**

[ ]

import seaborn as sns  
import numpy as np  
import pandas as pd  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_curve, roc\_auc\_score, confusion\_matrix, ConfusionMatrixDisplay  
import matplotlib.pyplot as plt

**Предварительная обработка данных**

1. Скачали датасет
2. Загрузили его в фрейм данных

[ ]

data = pd.read\_csv("diabetes\_binary\_5050split\_health\_indicators\_BRFSS2015.csv")  
data.head(3)

1. Проверим, есть ли в данных пропущенные значения? - Нет

[ ]

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 70692 entries, 0 to 70691

Data columns (total 22 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Diabetes\_binary 70692 non-null float64

1 HighBP 70692 non-null float64

2 HighChol 70692 non-null float64

3 CholCheck 70692 non-null float64

4 BMI 70692 non-null float64

5 Smoker 70692 non-null float64

6 Stroke 70692 non-null float64

7 HeartDiseaseorAttack 70692 non-null float64

8 PhysActivity 70692 non-null float64

9 Fruits 70692 non-null float64

10 Veggies 70692 non-null float64

11 HvyAlcoholConsump 70692 non-null float64

12 AnyHealthcare 70692 non-null float64

13 NoDocbcCost 70692 non-null float64

14 GenHlth 70692 non-null float64

15 MentHlth 70692 non-null float64

16 PhysHlth 70692 non-null float64

17 DiffWalk 70692 non-null float64

18 Sex 70692 non-null float64

19 Age 70692 non-null float64

20 Education 70692 non-null float64

21 Income 70692 non-null float64

dtypes: float64(22)

memory usage: 11.9 MB

1. Смотрим статистическую информацию о данных

[ ]

data.describe()

1. Проверим названия фичей

[ ]

data.columns

Index(['Diabetes\_binary', 'HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker',

'Stroke', 'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',

'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',

'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education',

'Income'],

dtype='object')

1. Проверим размерность датасета

[ ]

data.shape

(70692, 22)

1. Проверим распределение классов - идеальное

[ ]

data['Diabetes\_binary'].value\_counts()

**Разведочный анализ данных**

1. Построим корреляционную матрицу, на ее основе построим тепловую карту

[ ]

plt.figure(figsize = (20,10))  
corr\_matrix = data.corr(method='spearman')  
sns.heatmap(corr\_matrix, annot=True , cmap ='YlOrRd' )  
plt.title("Heatmap")

[ ]

strong\_corrs = corr\_matrix[(corr\_matrix.abs() > 0.465) & (corr\_matrix != 1)]  
strong\_corr\_pairs = strong\_corrs.stack().index.tolist()  
print("Наиболее сильные корреляции между признаками (положительные):")  
for feature1, feature2 in strong\_corr\_pairs:  
    print(f"{feature1} - {feature2}: {corr\_matrix.loc[feature1, feature2]}")  
  
strong\_corrs = corr\_matrix[(corr\_matrix < -0.3) & (corr\_matrix != 1)]  
strong\_corr\_pairs = strong\_corrs.stack().index.tolist()  
print("\nНаиболее сильные корреляции между признаками (отрицательные):")  
for feature1, feature2 in strong\_corr\_pairs:  
    print(f"{feature1} - {feature2}: {corr\_matrix.loc[feature1, feature2]}")  
  
corr\_with\_target = corr\_matrix['Diabetes\_binary'].drop('Diabetes\_binary')  
sorted\_corr\_with\_target = corr\_with\_target.abs().sort\_values(ascending=False)  
print("\nНаиболее сильные корреляции между таргетом и признаками:")  
for feature, corr\_value in list(sorted\_corr\_with\_target.items())[:3]:  
    print(f"{feature}: {corr\_matrix.loc[feature, 'Diabetes\_binary']}")

Наиболее сильные корреляции между признаками (положительные):

GenHlth - PhysHlth: 0.5242744791239562

GenHlth - DiffWalk: 0.4665398762627897

PhysHlth - GenHlth: 0.5242744791239562

DiffWalk - GenHlth: 0.4665398762627897

Наиболее сильные корреляции между признаками (отрицательные):

GenHlth - Income: -0.3823375495885094

DiffWalk - Income: -0.33642544406322217

Income - GenHlth: -0.3823375495885094

Income - DiffWalk: -0.33642544406322217

Наиболее сильные корреляции между таргетом и признаками:

GenHlth: 0.4142586575630089

HighBP: 0.3815155489073096

BMI: 0.3283556127093338

**Наиболее сильная положительная корреляция у данных пар фичей:**

GenHlth - PhysHlth: 0.5242744791239562

GenHlth - DiffWalk: 0.4665398762627897

**Наиболее сильная отрицательная корреляция у данных пар фичей:**

GenHlth - Income: -0.3823375495885094

DiffWalk - Income: -0.33642544406322217

**Наиболее сильная корреляция с таргетом у фичей:**

GenHlth

HighBP

BMI

1. Построим гистрограммы

[ ]

data.hist(figsize=(20,15))

[ ]

import scipy.stats as stats  
for col in data.columns[:-1]:  
  print(col, stats.shapiro(data[col]))

Diabetes\_binary ShapiroResult(statistic=0.6366224260750695, pvalue=1.935045727506944e-141)

HighBP ShapiroResult(statistic=0.6307424654727146, pvalue=6.219384256707285e-142)

HighChol ShapiroResult(statistic=0.6356614311673741, pvalue=1.6056227416636577e-141)

CholCheck ShapiroResult(statistic=0.13910956884100023, pvalue=3.6958770442485656e-169)

BMI ShapiroResult(statistic=0.8994835344464236, pvalue=8.561231464319347e-105)

Smoker ShapiroResult(statistic=0.6357330813248738, pvalue=1.6280949282298157e-141)

Stroke ShapiroResult(statistic=0.25727821590899824, pvalue=3.0948160301083957e-164)

HeartDiseaseorAttack ShapiroResult(statistic=0.42318075864977234, pvalue=5.606554452775773e-156)

PhysActivity ShapiroResult(statistic=0.5737081100670253, pvalue=2.209418210296378e-146)

Fruits ShapiroResult(statistic=0.6181959145262963, pvalue=5.819128217220532e-143)

Veggies ShapiroResult(statistic=0.5019478666672019, pvalue=2.7931259716782168e-151)

HvyAlcoholConsump ShapiroResult(statistic=0.20201688523758565, pvalue=1.2792219386198115e-166)

AnyHealthcare ShapiroResult(statistic=0.20918256470314933, pvalue=2.555687943354081e-166)

NoDocbcCost ShapiroResult(statistic=0.330066779655831, pvalue=7.692950081514783e-161)

GenHlth ShapiroResult(statistic=0.9148903266677716, pvalue=1.861457091661787e-100)

MentHlth ShapiroResult(statistic=0.5173703217600031, pvalue=2.7732320496942164e-150)

PhysHlth ShapiroResult(statistic=0.6130174518593317, pvalue=2.2333266606087133e-143)

DiffWalk ShapiroResult(statistic=0.5408747609502548, pvalue=1.0433751749517824e-148)

Sex ShapiroResult(statistic=0.6339287988543469, pvalue=1.1481472198842641e-141)

Age ShapiroResult(statistic=0.9567705212720113, pvalue=9.246336311933777e-84)

Education ShapiroResult(statistic=0.8456924272313249, pvalue=2.132924235160923e-116)

1. Построим графики для просмотра зависимостей между таргетом и наиболее коррелирующими с ним фичами

[ ]

fig, axes = plt.subplots(2, 2, figsize=(20, 15))  
  
sns.countplot(data=data, x="Diabetes\_binary", hue='HighBP', ax=axes[0, 0])  
axes[0, 0].set\_title("Diabetes\_binary vs HighBP")  
  
sns.countplot(data=data, x="Diabetes\_binary", hue='GenHlth', ax=axes[0, 1])  
axes[0, 1].set\_title("Diabetes\_binary vs GenHlth")  
  
sns.countplot(data=data, x="Diabetes\_binary", hue='HighChol', ax=axes[1, 0])  
axes[1, 0].set\_title("Diabetes\_binary vs HighChol")  
  
sns.countplot(data=data, x="Diabetes\_binary", hue='BMI', ax=axes[1, 1])  
axes[1, 1].set\_title("Diabetes\_binary vs BMI")

1. Посмотрим выбросы

[ ]

num\_cols = 5  
num\_rows = (len(data.columns) + num\_cols - 1) // num\_cols  
fig, axes = plt.subplots(num\_rows, num\_cols, figsize=(20, 4 \* num\_rows))  
axes = axes.flatten()  
i=0  
for column in data.columns:  
    sns.boxplot(x='Diabetes\_binary', y=column, data=data, ax=axes[i])  
    axes[i].set\_title(f'{column} vs Diabetes\_binary')  
    i += 1  
  
for j in range(i + 1, len(axes)):  
    fig.delaxes(axes[j])  
  
plt.tight\_layout()  
plt.show()

11.1 Удалим выбросы

[ ]

def replace\_outliers\_iqr(df, col, multiplier = 1.5):  
    Q1 = df[col].quantile(0.25)  
    Q3 = df[col].quantile(0.75)  
    IQR = Q3 - Q1  
    lower\_bound = Q1 - multiplier \* IQR  
    upper\_bound = Q3 + multiplier \* IQR  
    df[col] = np.where(df[col] < lower\_bound, lower\_bound, df[col])  
    df[col] = np.where(df[col] > upper\_bound, upper\_bound, df[col])  
    return df  
  
for col in data.columns:  
    data = replace\_outliers\_iqr(data, 'BMI', multiplier=1.5)  
    data = replace\_outliers\_iqr(data, 'MentHlth', multiplier=1.5)  
    data = replace\_outliers\_iqr(data, 'PhysHlth', multiplier=1.5)

[ ]

num\_cols = 5  
num\_rows = (len(data.columns) + num\_cols - 1) // num\_cols  
fig, axes = plt.subplots(num\_rows, num\_cols, figsize=(20, 4 \* num\_rows))  
axes = axes.flatten()  
i=0  
for column in data.columns:  
    sns.boxplot(x='Diabetes\_binary', y=column, data=data, ax=axes[i])  
    axes[i].set\_title(f'{column} vs Diabetes\_binary')  
    i += 1  
  
for j in range(i + 1, len(axes)):  
    fig.delaxes(axes[j])  
  
plt.tight\_layout()  
plt.show()

**Обучение модели**

1. Таргет бинарные - кодировать не нужно
2. Удаляем таргет из исходного датасета

[ ]

y = data.pop('Diabetes\_binary')

df = data

1. Масштабируем данные, используя MinMaxScaler

[ ]

scaler = StandardScaler()  
df\_scaled = scaler.fit\_transform(df)

1. Поделим данные на обучающую и тестовую выборки в соотнощении 70/30

[ ]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
    df\_scaled, y, test\_size=0.3, random\_state=17, stratify=y  
)

1. Применяем следующие алгоритмы:

Logistic Regression (логистическая регрессия)

Random Forest (случайный лес)

Support Vector Machine (машина опорных векторов, SVM)

K-Nearest Neighbors (k ближайших соседей, KNN)

Gradient Boosting (градиентный бустинг)

**Логистическая регрессия**

[ ]

logistic\_model = LogisticRegression() #Инифиализация модели  
logistic\_params = {"C": [0.1, 1, 10], "penalty": ["l2"], "solver": ["lbfgs"]} #Параметры для GridSearchCV  
logistic\_grid = GridSearchCV(logistic\_model, logistic\_params, cv=5, scoring="accuracy", n\_jobs=-1) #Использование GridSearchCV  
logistic\_grid.fit(X\_train, y\_train)  
best\_logistic = logistic\_grid.best\_estimator\_ #Обучаем модель с лучшими параметрами  
y\_pred\_logistic = best\_logistic.predict(X\_test) #Предскзываем на тесте

[ ]

from sklearn.metrics import classification\_report  
target\_names = ['class 0', 'class 1']  
print(classification\_report(y\_test, y\_pred\_logistic, target\_names=target\_names))

precision recall f1-score support

class 0 0.76 0.73 0.74 10604

class 1 0.74 0.77 0.75 10604

accuracy 0.75 21208

macro avg 0.75 0.75 0.75 21208

weighted avg 0.75 0.75 0.75 21208

[ ]

print("Logistic Regression Metrics:")

print(f"Best Parameters: {logistic\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_logistic):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_logistic):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_logistic):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_logistic):.4f}")

Logistic Regression Metrics:

Best Parameters: {'C': 1, 'penalty': 'l2', 'solver': 'lbfgs'}

Accuracy: 0.7471

Precision: 0.7379

Recall: 0.7664

F1 Score: 0.7519

[ ]

if hasattr(best\_logistic, "predict\_proba"):  
    y\_proba\_logistic = best\_logistic.predict\_proba(X\_test)[:, 1]  
else:  
    y\_proba\_logistic = best\_logistic.decision\_function(X\_test)  
  
  
fpr\_logistic, tpr\_logistic, \_ = roc\_curve(y\_test, y\_proba\_logistic)  
roc\_auc\_logistic = roc\_auc\_score(y\_test, y\_proba\_logistic)  
plt.figure(figsize=(8, 6))  
plt.plot(fpr\_logistic, tpr\_logistic, label=f'Logistic Regression (AUC = {roc\_auc\_logistic:.4f})')  
plt.xlabel("False Positive Rate")  
plt.ylabel("True Positive Rate")  
plt.title("ROC Curve")  
plt.legend(loc="best")  
plt.grid()  
plt.show()

[ ]

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_logistic)  
disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=target\_names)  
disp.plot(cmap="Blues")  
plt.title("Confusion Matrix")  
plt.show()

**Случайный лес**

[ ]

rf\_model = RandomForestClassifier()  
rf\_params = {"n\_estimators": [100, 200], "max\_depth": [10, 20, None], "min\_samples\_split": [2, 5]}  
rf\_grid = GridSearchCV(rf\_model, rf\_params, cv=5, scoring="accuracy", n\_jobs=-1)  
rf\_grid.fit(X\_train, y\_train)  
best\_rf = rf\_grid.best\_estimator\_  
y\_pred\_rf = best\_rf.predict(X\_test)

/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process\_executor.py:752: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.

warnings.warn(

[ ]

from sklearn.metrics import classification\_report  
target\_names = ['class 0', 'class 1']  
print(classification\_report(y\_test, y\_pred\_rf, target\_names=target\_names))

precision recall f1-score support

class 0 0.77 0.70 0.74 10604

class 1 0.73 0.79 0.76 10604

accuracy 0.75 21208

macro avg 0.75 0.75 0.75 21208

weighted avg 0.75 0.75 0.75 21208

[ ]

print("Random Forest Metrics:")  
print(f"Best Parameters: {rf\_grid.best\_params\_}")  
print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_rf):.4f}")  
print(f"Precision: {precision\_score(y\_test, y\_pred\_rf):.4f}")  
print(f"Recall: {recall\_score(y\_test, y\_pred\_rf):.4f}")  
print(f"F1 Score: {f1\_score(y\_test, y\_pred\_rf):.4f}")

Random Forest Metrics:

Best Parameters: {'max\_depth': 10, 'min\_samples\_split': 2, 'n\_estimators': 100}

Accuracy: 0.7476

Precision: 0.7270

Recall: 0.7931

F1 Score: 0.7586

[ ]

if hasattr(best\_rf, "predict\_proba"):  
    y\_proba\_rf = best\_rf.predict\_proba(X\_test)[:, 1]  
else:  
    y\_proba\_rf = best\_rf.decision\_function(X\_test)  
  
fpr\_rf, tpr\_rf, \_ = roc\_curve(y\_test, y\_proba\_rf)  
roc\_auc\_rf = roc\_auc\_score(y\_test, y\_proba\_rf)  
plt.figure(figsize=(8, 6))  
plt.plot(fpr\_rf, tpr\_rf, label=f'Random Forest (AUC = {roc\_auc\_rf:.4f})')  
plt.xlabel("False Positive Rate")  
plt.ylabel("True Positive Rate")  
plt.title("ROC Curve")  
plt.legend(loc="best")  
plt.grid()  
plt.show()

[ ]

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_rf)  
disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=target\_names)  
disp.plot(cmap="Blues")  
plt.title("Confusion Matrix")  
plt.show()

**Метод опорных векторов**

[ ]

svm\_model = SVC()  
svm\_params = {"C": [0.1, 1, 10], "kernel": ["linear", "rbf"]}  
svm\_grid = GridSearchCV(svm\_model, svm\_params, cv=5, scoring="accuracy", n\_jobs=-1)  
svm\_grid.fit(X\_train, y\_train)  
best\_svm = svm\_grid.best\_estimator\_  
y\_pred\_svm = best\_svm.predict(X\_test)

[ ]

from sklearn.metrics import classification\_report  
target\_names = ['class 0', 'class 1']  
print(classification\_report(y\_test, y\_pred\_svm, target\_names=target\_names))

precision recall f1-score support

class 0 0.79 0.69 0.73 10604

class 1 0.72 0.81 0.76 10604

accuracy 0.75 21208

macro avg 0.75 0.75 0.75 21208

weighted avg 0.75 0.75 0.75 21208

[ ]

print("SVM Metrics:")  
print(f"Best Parameters: {svm\_grid.best\_params\_}")  
print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_svm):.4f}")  
print(f"Precision: {precision\_score(y\_test, y\_pred\_svm):.4f}")  
print(f"Recall: {recall\_score(y\_test, y\_pred\_svm):.4f}")  
print(f"F1 Score: {f1\_score(y\_test, y\_pred\_svm):.4f}")

SVM Metrics:

Best Parameters: {'C': 0.1, 'kernel': 'rbf'}

Accuracy: 0.7492

Precision: 0.7214

Recall: 0.8121

F1 Score: 0.7640

[ ]

if hasattr(best\_svm, "predict\_proba"):  
    y\_proba\_svm = best\_svm.predict\_proba(X\_test)[:, 1]  
else:  
    y\_proba\_svm = best\_svm.decision\_function(X\_test)  
  
fpr\_svm, tpr\_svm, \_ = roc\_curve(y\_test, y\_proba\_svm)  
roc\_auc\_svm = roc\_auc\_score(y\_test, y\_proba\_svm)  
plt.figure(figsize=(8, 6))  
plt.plot(fpr\_svm, tpr\_svm, label=f'SVM (AUC = {roc\_auc\_svm:.4f})')  
plt.xlabel("False Positive Rate")  
plt.ylabel("True Positive Rate")  
plt.title("ROC Curve")  
plt.legend(loc="best")  
plt.grid()  
plt.show()

[ ]

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_svm)  
disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=target\_names)  
disp.plot(cmap="Blues")  
plt.title("Confusion Matrix")  
plt.show()

**K-ближайших соседей**

[ ]

knn\_model = KNeighborsClassifier()  
knn\_params = {"n\_neighbors": [3, 5, 7], "weights": ["uniform", "distance"]}  
knn\_grid = GridSearchCV(knn\_model, knn\_params, cv=5, scoring="accuracy", n\_jobs=-1)  
knn\_grid.fit(X\_train, y\_train)  
best\_knn = knn\_grid.best\_estimator\_  
y\_pred\_knn = best\_knn.predict(X\_test)

/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast

\_data = np.array(data, dtype=dtype, copy=copy,

[ ]

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_knn, target\_names=target\_names))

precision recall f1-score support

class 0 0.73 0.69 0.71 10604

class 1 0.71 0.75 0.73 10604

accuracy 0.72 21208

macro avg 0.72 0.72 0.72 21208

weighted avg 0.72 0.72 0.72 21208

[ ]

print("KNN Metrics:")

print(f"Best Parameters: {knn\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_knn):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_knn):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_knn):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_knn):.4f}")

KNN Metrics:

Best Parameters: {'n\_neighbors': 7, 'weights': 'uniform'}

Accuracy: 0.7200

Precision: 0.7076

Recall: 0.7497

F1 Score: 0.7281

[ ]

if hasattr(best\_knn, "predict\_proba"):

    y\_proba\_knn = best\_knn.predict\_proba(X\_test)[:, 1]

else:

    y\_proba\_knn = best\_knn.decision\_function(X\_test)

fpr\_knn, tpr\_knn, \_ = roc\_curve(y\_test, y\_proba\_knn)

roc\_auc\_knn = roc\_auc\_score(y\_test, y\_proba\_knn)

plt.plot(fpr\_knn, tpr\_knn, label=f'KNN (AUC = {roc\_auc\_knn:.4f})')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend(loc="best")

plt.grid()

plt.show()

[ ]

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_knn)

disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=target\_names)

disp.plot(cmap="Blues")

plt.title("Confusion Matrix")

plt.show()

**Градиентный бустинг**

[ ]

gb\_model = GradientBoostingClassifier()

gb\_params = {"n\_estimators": [100, 200], "learning\_rate": [0.01, 0.1, 0.2], "max\_depth": [3, 5]}

gb\_grid = GridSearchCV(gb\_model, gb\_params, cv=5, scoring="accuracy", n\_jobs=-1)

gb\_grid.fit(X\_train, y\_train)

best\_gb = gb\_grid.best\_estimator\_

y\_pred\_gb = best\_gb.predict(X\_test)

[ ]

from sklearn.metrics import classification\_report

target\_names = ['class 0', 'class 1']

print(classification\_report(y\_test, y\_pred\_gb, target\_names=target\_names))

precision recall f1-score support

class 0 0.78 0.70 0.74 10604

class 1 0.73 0.80 0.76 10604

accuracy 0.75 21208

macro avg 0.75 0.75 0.75 21208

weighted avg 0.75 0.75 0.75 21208

[ ]

print("Gradient Boosting Metrics:")

print(f"Best Parameters: {gb\_grid.best\_params\_}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_gb):.4f}")

print(f"Precision: {precision\_score(y\_test, y\_pred\_gb):.4f}")

print(f"Recall: {recall\_score(y\_test, y\_pred\_gb):.4f}")

print(f"F1 Score: {f1\_score(y\_test, y\_pred\_gb):.4f}")

Gradient Boosting Metrics:

Best Parameters: {'learning\_rate': 0.2, 'max\_depth': 3, 'n\_estimators': 200}

Accuracy: 0.7502

Precision: 0.7286

Recall: 0.7975

F1 Score: 0.7615

[ ]

if hasattr(best\_gb, "predict\_proba"):

    y\_proba\_gb = best\_gb.predict\_proba(X\_test)[:, 1]

else:

    y\_proba\_gb = best\_gb.decision\_function(X\_test)

fpr\_gb, tpr\_gb, \_ = roc\_curve(y\_test, y\_proba\_gb)

roc\_auc\_gb = roc\_auc\_score(y\_test, y\_proba\_gb)

plt.plot(fpr\_gb, tpr\_gb, label=f'Gradient Boosting (AUC = {roc\_auc\_gb:.4f})')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend(loc="best")

plt.grid()

plt.show()

[ ]

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_gb)  
disp = ConfusionMatrixDisplay(confusion\_matrix=conf\_matrix, display\_labels=target\_names)  
disp.plot(cmap="Blues")  
plt.title("Confusion Matrix")  
plt.show()