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 «Прикладной анализ данных»

КЛАССИФИКАЦИЯ ОДОБРЕННЫХ КРЕДИТОВ

Разработчики проекта:
Чернояров Матвей Владимирович,
Жильцов Максим Дмитриевич,
Шевцова Татьяна Денисовна

Пермь, 2024

Оглавление

ПАСПОРТ ПРОЕКТА

Название проекта: Классификация одобренных кредитов

Сведения об авторах: Чернояров Матвей Владимирович, Жильцов Максим Дмитриевич, Шевцова Татьяна Денисовна.

Цель: выполнить анализ данных и построить модель, на основе которой можно предсказать одобрят человек кредит или откажут в получении кредита.

Задачи:

1. Выполнить анализ проблемы и аргументировать её актуальность.
2. Загрузить и проанализировать данные.
3. Осуществить предварительный анализ данных.
4. Создать модель для классификации, используя различные методы машинного обучения. Выявить самую лучшую. Оценить метрики для моделей.
5. Сформулировать выводы.

Краткое описание проекта:

Проанализировать данные. На основе полученных данных построить несколько моделей. Отыскать какие признаки будут вносить больший вклад в целевую переменную.

Конкретные ожидаемые результаты:

Построенные модели будут классифицировать людей по одобренным и неодобренным кредитам.

СОДЕРЖАНИЕ ПРОЕКТА

Анализ проблемы исследования

В современном мире практически любой человек в состоянии пойти в тот банк, которым он пользуется, и попросить о такой услуге, как кредит. Для определённости кредит – это сумма денег, которую даёт банк человеку на определённый промежуток времени с определёнными условиями. Банки выдают кредит для того, чтобы получить от заёмщика прибыль, благодаря выплачиваемому вместе с основной суммой долга. Человек, который пользуется этой услугой банка, обязуется вернуть основную сумму и накопившуюся сумму, связанную с набегающими процентами. Однако нередко происходят такие ситуации, в которых человек, получивший кредит, не возвращает его в назначенный или в последующее назначенный промежуток времени по каким-то неожиданным причинам для банка. Именно поэтому банку необходимо уметь отслеживать таких людей по следующим причинам:

1. Минимизирование риска невозврата. Предположим, что банк одобряет и выдаёт кредит человеку, который имеет нестабильный доход или уже присутствуют задолженности перед другими организациями. В таком случае, банк, дающий в долг, терпит убытки в денежном плане, что негативно может сказаться на экономике этого банка в целом. А это в свою очередь может привести к оттоку потенциальных клиентов, способных выплатить и преумножить денежный капитал банка.

2. Поддержание финансовой стабильности. По своей сути каждый выданный банком кредит является его активом, то есть является ресурсом, с помощью которого предприятие распоряжается для получения прибыли. Если по какой-то причине большая часть заёмщиков перестают платить по счетам, то происходит подрыв финансовой устойчивости банка, что может привести даже к банкротству финансовой организации.

3. Соблюдение нормативных требований. Существуют законы и правила, созданные Центральными банками и другими регулирующими органами власти, которые требуют соблюдать ими требования к оценке рисков при выдаче кредитов. Если по каким-либо причинам банк не следует установленным требования, то в таком случае регулирующие органы могут наложить санкции или выписать за эти нарушения довольно большие штрафы. Например, в России Центральный банк (ЦБ) требует от банков соблюдать определенные стандарты оценки платежеспособности клиентов. Если банк переступил черту нормы, то регулятор в состоянии ограничить его деятельность или наложить штрафные санкции.

4. Репутация банка. Немало случалось таких ситуаций, в которых банки выдавали кредиты непроверенным заёмщикам, что приводило к банкротству банков. Выдача кредитов сомнительным людям может подорвать репутацию банка среди клиентов, партнёров и, разумеется, инвесторов. Утрата доверия со стороны социума также может повлиять на отток новых клиентов и депозитов. Кроме того, такие события принуждают людей усомниться в банковской системе в целом.

5. Защита интересов вкладчика. Деньги, которыми манипулирует банк, принадлежат не только организации, на и вкладчикам. Предоставляя возможность взять кредит непроверенным людям, банк подвергает риску средства своих клиентов. В таком положении банк испытывает финансовые трудности. Чтобы хоть как-то компенсировать своё состояние, он должен будет сократить выплаты своим вкладчикам. В самом ужасном случае, банк может объявить о банкротстве и закрыться, что может привести к неодобрению общественности.

Таким образом, банку крайне необходимо уметь отличать людей, которые в состоянии выдержать финансовую ношу, от людей, которые могут потенциально навредить экономической жизни банка и репутации в целом. Для этого банк вводит критерии, по которым можно отличить клиентов.

Основные критерии, по которым можно оценить заёмщика, можно определить следующим образом:

1. Кредитная история клиента. Кредитная история клиента - объективно является одним из важных фактов. По кредитной истории человека можно понять, насколько он ответственно подошёл к этому делу и своим обязательствам. Например, банк с большой вероятностью одобрит кредит человеку, который вовремя возвращал кредиты, не имел просроченных платежей, так как такой человек принесёт меньший ущерб банку.

2. Стабильность и доход клиента. Каждым банком в индивидуальном порядке проверяется достаточно ли доходов для погашения будущего кредита у клиента. Также учитывается продолжительность работы клиента на одном месте. Предположим, что человек работает на какой-то должности в течение 5 лет, имеет стабильный доход, ответственно подходит к своей работе. Если такой человек, придёт за кредитом, то, скорее всего, кредит ему одобрят. С меньшей вероятностью выдадут кредит человеку, который только поменял вид своей деятельности и имеет низкую зарплату на новой должности.

3. Наличие других кредитов, по которым есть долг. Присутствие у человека кредита в другой финансовой организации может понизить шанс получения его в другом банк. Такому человеку, возможно, будет трудно осилить выплату сразу двух кредитов. Однако банк в этом случае может прибегнуть к следующей схеме: если человек по соотношению общих доходов и нынешним кредитам способен взять на себя ещё одни кредит, то, разумеется, банк может одобрить такой кредит. Если заёмщик даже при вышенаписанных обстоятельствах не способен выплачивать проценты, то банку придётся отказать в выдачи кредита.

4. Возраст и семейное положение. Банку также важно оценить возраст человека и его семейное положение в получении кредита. Они стараются меньше давать кредиты молодым семьям с детьми, потому что присутствие одного и более детей увеличивает расходы и снижает эффективность выплачивать кредит.

Организации с меньшим энтузиазмом выдают кредит молодым людям, поскольку у них меньший опыта управления финансами и нестабильная зарплата. Например, человек до 23 лет, не имеющий постоянного дохода и работы, определяется банком как повышенный риск к вовремя не уплачиваемому кредиту. Также банк старается меньше давать кредиты пожилым и старым людям, так как финансовые доходы таких людей по большой части невелики.

5. Наличие активов и имущества. Если у клиента есть, например, автомобиль или другое движимое, недвижимое имущество, то это может стать залогом при получении крупного кредита. Другими словами, чем больше активов у заёмщика, тем ниже риск для банка получить просроченные платежи.

6. Цель кредита. Цель кредита является важным критерием при одобрении кредита. Некоторые цели кредитования по сравнению с другими по своей природе являются более рискованными. Например, потребительские кредиты на бытовую технику или поездку в отпуск по сравнению с ипотечными кредитами оцениваются как более рискованными. Такого рода целям отдают меньший приоритет.

Таким образом, рассмотрели основные критерии, по которым банк может оценить клиента и одобрить или не одобрить кредит. Конечно, существуют более тонкие моменты и критерии.

Если заглянуть в финансовую статистику, которая публикуется Центральным банком РФ, то можно увидеть, что по состоянию на конец марта 2023 года общий долг по кредитам, предоставленный физическим лицам, составил 2,06 трлн. руб. По состоянию на декабрь 2022 год общий долг составил 2,11 трлн. руб.

По официальной информации с января по март 2023 года число неплательщиков по кредитам превысило 14 млн. чел. По сравнению с прошлым годом это на 25% больше.

Знание банка о заёмщике необходимо во избежание рисков.

Цель: выполнить анализ данных и построить модель, на основе которой можно предсказать одобрят человек кредит или откажут в получении кредита.

Задачи:

1. Выполнить анализ проблемы и аргументировать её актуальность.

2. Загрузить и проанализировать данные.

3. Осуществить предварительный анализ данных.

4. Создать модель для классификации, используя различные методы машинного обучения. Выявить самую лучшую модель. Оценить метрики для моделей.

5. Сформулировать выводы.

Исходные данные

Данные представляют из себя синтетическую версию, основанная на оригинальном наборе данных рисках на Kaggle. Датасет содержит 45000 строк 14 столбцов, то есть всего 14 переменных, включая целевую переменную.

Список столбцов:

1. **person_age –** возраст человека.
2. **person_gender –** пол человека.
3. **person_education –** образование человека.
4. **person_income –** годовой доход.
5. **person_emp_exp** – опыт работы.
6. **person_home_ownership** – статус домовладения (например, аренда).
7. **loan_amnt –** запрашиваемая сумма кредиты.
8. **loan_intent** – цель кредита.
9. **loan_int_rate** – Процентная ставка по кредиту.

10. **loan_percent_income** – сумма кредита в процентах от годового дохода.

11. **cb_person_cred_hist_length** – кредитная история в годах.

12. **credit_score** – кредитная рейтинг

13. **previous_loan_defaults_on_file** – дефолт по прошлому кредиту

14. **loan_status** (целевая переменная) – одобрили или не одобрили кредит.

Необходимо проанализировать данные кредитной истории разных лиц и с помощь методов машинного обучения выбрать модели, делающие выбор в пользу выдачи кредитов аналогичным решениям работников банка. Важно сравнить модели и выбрать лучшую модель из предложенных в рассмотрении.

Выдвинем гипотезу исследования: решение о выдаче кредита может быть предсказано с использованием модели машинного обучения.

Реализация проекта

Этап 1. Загрузка данных и их анализ

Представим исходный набор данных в виде датаферма. Подключим библиотеки. (Courier New)

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

Выгрузим csv-файл с данными в среду разработки для Python с помощью следующий строчки кода.

```
data = pd.read_csv('loan_data.csv')
data.head()
```

	person_age	person_gender	person_education	person_income	person_emp_exp	person_home_ownership	loan_amnt
0	22.0	female	Master	71948.0	0	RENT	35000.0
1	21.0	female	High School	12282.0	0	OWN	1000.0
2	25.0	female	High School	12438.0	3	MORTGAGE	5500.0
3	23.0	female	Bachelor	79753.0	0	RENT	35000.0
4	24.0	male	Master	66135.0	1	RENT	35000.0

Рис. а. Исходный датафрейм

loan_intent	loan_int_rate	loan_percent_income	cb_person_cred_hist_length	credit_score	previous_loan_defaults_on_file	loan_status
PERSONAL	16.02	0.49	3.0	561	No	1
EDUCATION	11.14	0.08	2.0	504	Yes	0
MEDICAL	12.87	0.44	3.0	635	No	1
MEDICAL	15.23	0.44	2.0	675	No	1
MEDICAL	14.27	0.53	4.0	586	No	1

Рис. б. Исходный датафрейм

Узнаем тип каждой переменной. Для этого напишем строчку кода.

```
data.info()
```

```
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   person_age                      45000 non-null  float64
 1   person_gender                   45000 non-null  object
 2   person_education                45000 non-null  object
 3   person_income                   45000 non-null  float64
 4   person_emp_exp                  45000 non-null  int64
 5   person_home_ownership           45000 non-null  object
 6   loan_amnt                       45000 non-null  float64
 7   loan_intent                     45000 non-null  object
 8   loan_int_rate                   45000 non-null  float64
 9   loan_percent_income             45000 non-null  float64
 10  cb_person_cred_hist_length      45000 non-null  float64
 11  credit_score                    45000 non-null  int64
 12  previous_loan_defaults_on_file  45000 non-null  object
 13  loan_status                     45000 non-null  int64
dtypes: float64(6), int64(3), object(5)
memory usage: 4.8+ MB
```

Рис. 1. Тип переменной

Числовым типом обладают возраст человека, годовой доход, запрашиваемая сумма кредита, процентная ставка по кредиту, сумма кредитов в процентах от годового дохода, кредитная история, кредитный рейтинг, loan_status. Чтобы получить рабочие признаки необходимо их перевести в числовой формат. Пустых значений нет.

Этап 2. Разведочный анализ

Исследуем числовые признаки на выбросы. Для этого воспользуемся ящиками с усами. Ящик с усами – это способ представления данных, с помощью которого можно понять как распределены данные. Также можно посмотреть 3 основных квартиля и выбросы.

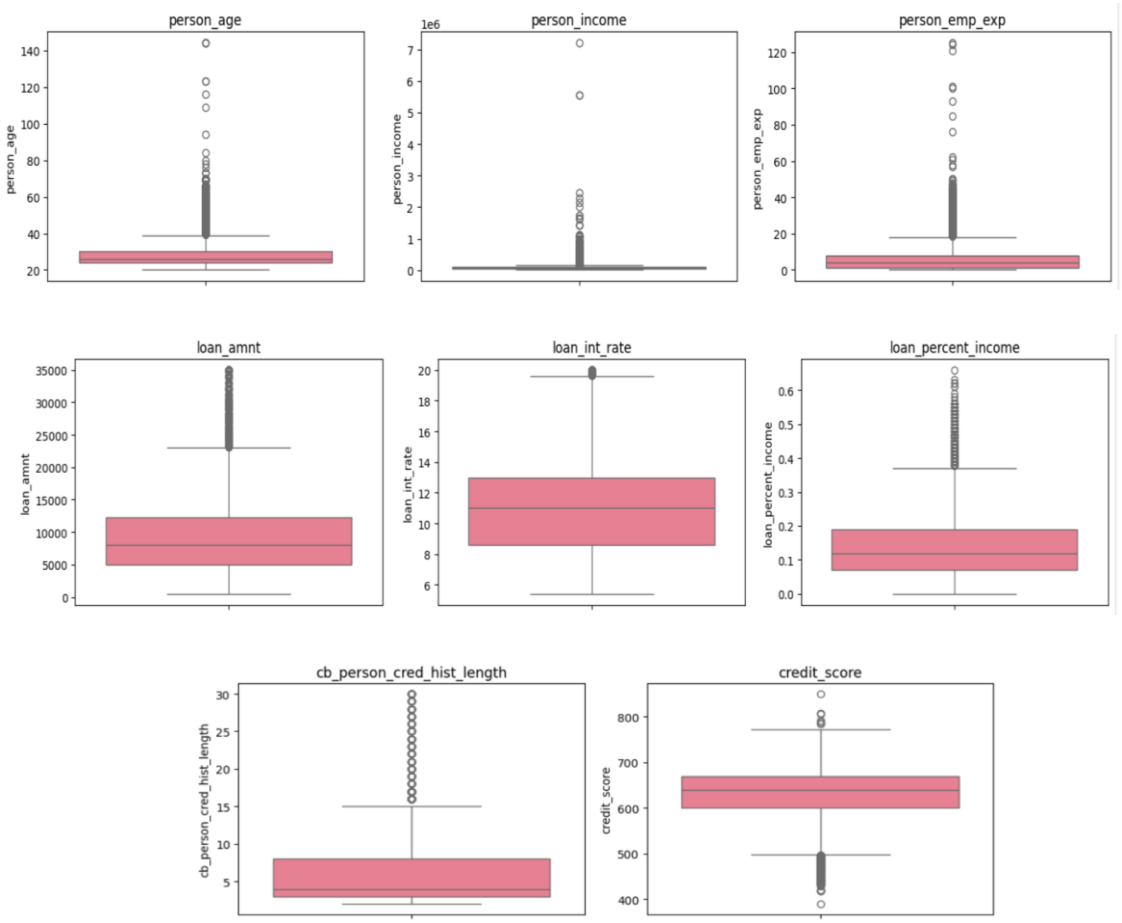

Рис. 2. Диаграммы boxplot для всех числовых признаков

Верхней линии отвечает верхний квартиль – меньшего этого значения 75%. Нижний линии отвечает нижний квартиль – меньше этого значения 25%. Медиана находиться между этими значениями в закрашенной области. Медиана – такое значение, при котором вероятность получить меньшое или большое значения одинаково, то есть 50%.

Как видно из диаграмм присутствует существенное количество признаков, у которых наблюдается много выбросов. От них необходимо избавиться, для того чтобы повысить качество модели. Для этого зададим межквартальный размах следующим образом.

```
Q1 = data[column].quantile(0.25) # Первый квартиль
Q3 = data[column].quantile(0.75) # Третий квартиль
```

Межквартальный размах найдём по формуле:

```
IQR = Q3 - Q1
```

Найдём нижнюю и верхнюю границы:

```
lower_bound = IQR - 1.5 * Q1
upper_bound = IQR - 1.5 * Q1
```

Применим это к каждому числовому признаку и выведем вновь boxplot

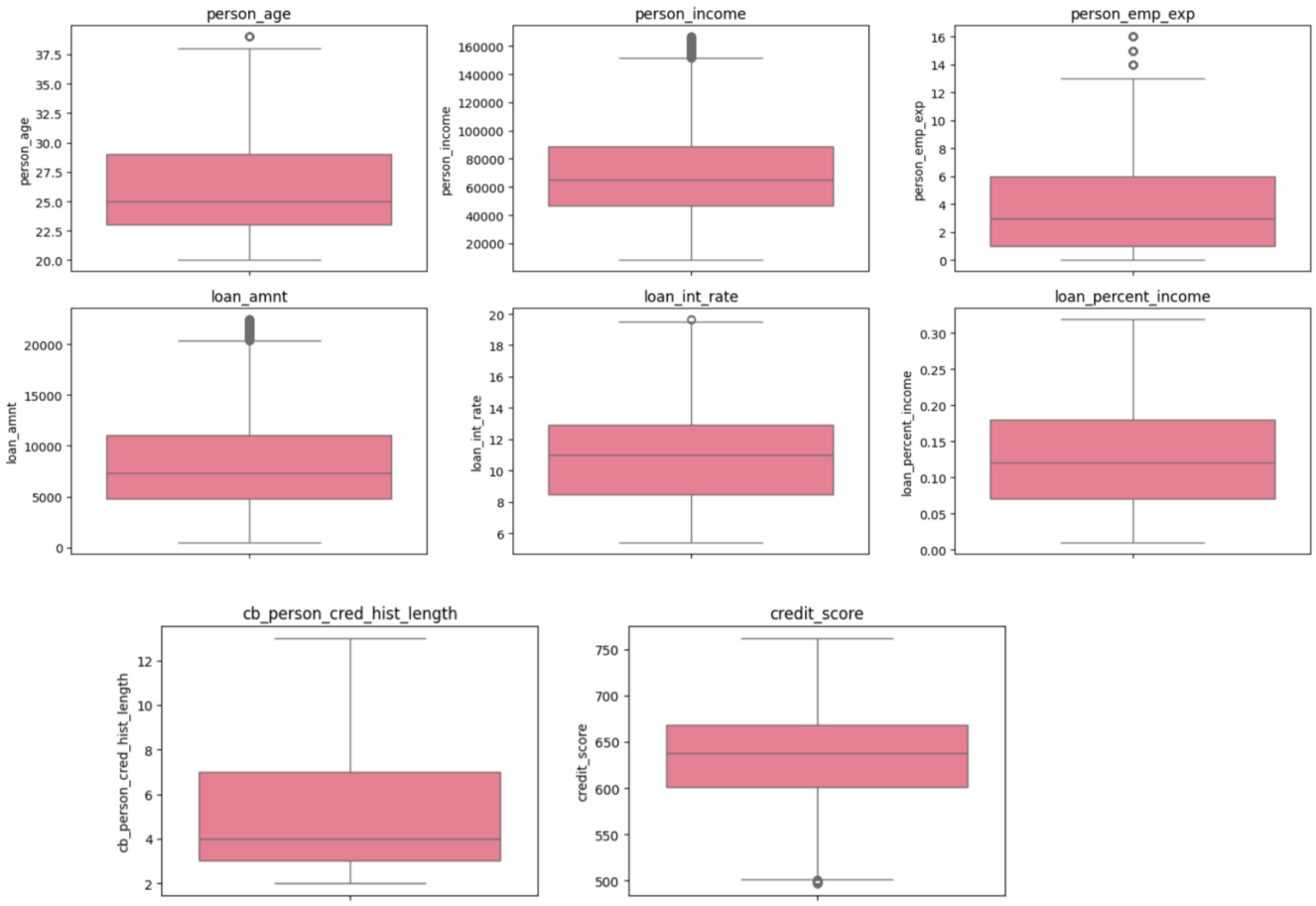

Рис. 3. Диаграммы ящики с усами

После удаления выбросов выведем основные статистические показатель для числовых признаков следующей строчкой кода.

```
data.describe()
```

	person_age	person_income	person_emp_exp	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_hist_length	credit_score	loan_status
count	36065.000000	36065.000000	36065.000000	36065.000000	36065.000000	36065.000000	36065.000000	36065.000000	36065.000000
mean	26.404631	70058.799335	4.071538	8276.245529	10.881279	0.129732	4.998087	632.102703	0.202107
std	3.867338	31562.837789	3.878033	4782.254222	2.927990	0.071587	2.649725	47.684452	0.401577
min	20.000000	8000.000000	0.000000	500.000000	5.420000	0.010000	2.000000	497.000000	0.000000
25%	23.000000	46458.000000	1.000000	4800.000000	8.490000	0.070000	3.000000	601.000000	0.000000
50%	25.000000	64796.000000	3.000000	7350.000000	11.010000	0.120000	4.000000	638.000000	0.000000
75%	29.000000	88578.000000	6.000000	11026.000000	12.910000	0.180000	7.000000	668.000000	0.000000
max	39.000000	166754.000000	16.000000	22500.000000	19.690000	0.320000	13.000000	762.000000	1.000000

Рис. 4. Основные статистические показатели

На предоставленном рисунке можно увидеть следующие параметры для числовой переменной: count – количество наблюдений, mean – среднее значение, std – стандартное отклонение. Стандартное отклонение характеризует средней отклонение от среднего значения. 25%, 50%, 75% - нижний, серединный(медиана) и верхний квартиль, а max – максимальное значения.

Проведём нормализацию данных, потому что у нас параметры разных порядков. Если этого не сделать, то может произойти так, что модель отдаст предпочтение признакам, у которых большие значения. Это может сказаться на предсказательной способности модели и повлиять на метрики.

Перед тем как провести нормализацию данных необходимо убрать категориальные признаки и целевую переменную из датафрейма, предварительно создав копию датафрейма, чтобы не было проблем с пропажей данных. Используем следующую строчку кода:

```
df = pd.DataFrame(data)
df_copy = df.copy()
df_copy_new = df_copy.dropna([все категориальные признаки и целевая переменная], axis = 1)
```

После того как разделили данные необходимо их нормализовать. Будем использовать нормализацию мин-макс. Суть такой нормализации закулючается в том, что мы сначала вычитаем минимальное значения, а затем делим на разность между максимальным и минимальным значением. В таком случае значения будут лежать в диапазоне от 0 до 1. Ниже приведена формула.

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Воспользуемся инструментом библиотеки scikit-learn.preprocessing

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
norm_df = scaler.fit_transform(df_copy_new)
norm_df.head(3)
```

Датафрейм теперь имеет следующий вид:

	person_age	person_income	person_emp_exp	loan_amnt	loan_int_rate	loan_percent_income	cb_person_cred_hist_length\t	credit_score
0	0.052632	0.026973	0.0	0.022727	0.400841	0.225806	0.000000	0.026415
1	0.052632	0.031187	0.0	0.090909	0.120533	0.580645	0.000000	0.132075
2	0.052632	0.029851	0.0	0.050000	0.653118	0.387097	0.090909	0.539623

Рис. 5. Нормализованные числовые признаки

Как видно нормализация данных прошла успешна.

Однако в нашем исходном датафрейме были не только числовые, но и категориальные переменный. Осуществим кодирование категориальных признаков.

Категориальными переменными являются person_gender, person_education, person_home_ownership, loan_intent, previous_loan_defaults_on_file. Необходимо эти переменные перевести в числовой формат.

Создадим снова копию исходного датасета для дальнейшей работы с ней. В этом датафрейме выбираем только категориальные признаки. Также создадим функцию для перевода категориальных значений в числовые.

```
df1 = pd.DataFrame(data)
df1_copy = df1.copy()
df1_obj = df1.select_dtypes(include = ['object'])

def Categorial(Df):
     arr = Df.columns
     for i in arr:
          Df[i] = Df[i].astype('category')
          Df[i] = Df[i].cat.codes
     return Df
df1_obj_int = Categorial(df1_obj)
df1_obj_int.head(3)
```

	person_gender	person_education	person_home_ownership	loan_intent	previous_loan_defaults_on_file
1	0	3	2	1	1
5	0	3	2	5	0
9	0	3	2	5	0

Рис. 6. Категориальные признаки в числовом виде

Объединим два датасета в один для дальнейшей работы с ним.

```
Data = pd.concat([norm_df, df1_obj_int, axis = 1])

Data['loan_status'] #добавляем целевую переменную

Data # итоговый датафрейм
```

Таким образом, подготовили датафрейм к алгоритмам машинного обучения.

Проверим количество одобренных и неодобренных кредитов.

```
Data['loan_status'].value_counts()
```

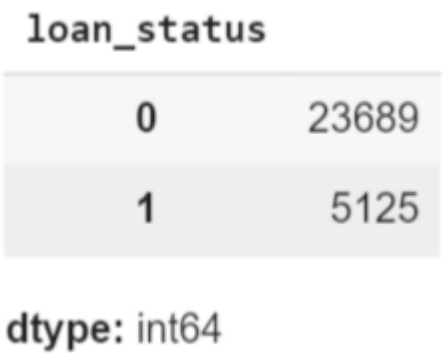

Рис. 7. Количество наблюдений для loan_status

Датасеты оказались несбалансированными. Приведём к одинаковому количеству в каждом классе. Чтобы это сделать, будем удалять наблюдения из доминирующего класса – 0 (отказ в получении кредита).

```
class_0 = Data.query(''loan_status = 0'')
class_0 = Data.query(''loan_status = 0'')
sampled_class_0 = class_0.sample(n = len(class_1), random_state = 42)
balance_df = pd.concat([sampled_class_0, class_1], ignore_index = True)
DF = balance_df.dropna()
```

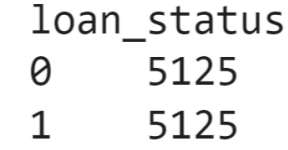

Рис. 8. Распределение отбалансированных классов

Построим тепловою карту для такого набора данных. Тепловая карта – способ визуализации данных, построенной на матрице корреляции между переменными. С помощью такой визуализации можно посмотреть корреляцию между переменными, то есть отследить зависимость между двумя переменными.

```
corr = DF.corr()
plt.figure(figsize = (14, 14))
sns.heatmap(corr, annot = True, cmap ='coolwarm', fmt = '.2f', square = True)
```

Выведем тепловую карту.

```
plt.titile('Матрица корреляций сбалансированной подвыборке', frontsize = 20)
plt.show()
```

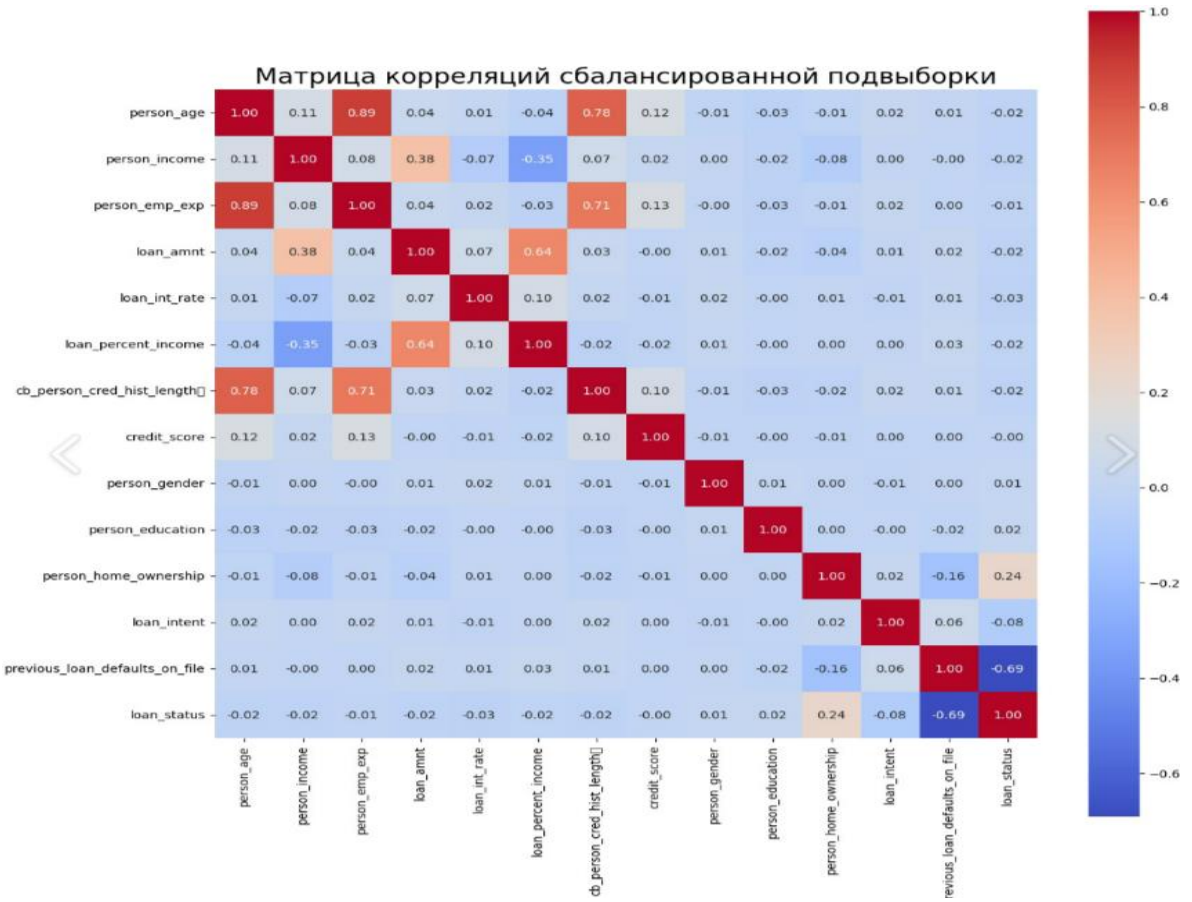

Рис.9. Тепловая карата

Как видно на целевая переменная имеет хорошую отрицательную корреляцию с дефолтом по кредитам.

Этап 3. Моделирование.

Разобьём данные на обучающую и тестовую выборки, где обучающая выборка составляет 80% и тестовая выборка 20%.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Мы будем при машинном обучении пользоваться 8 моделями машинного обучения: логистическая регрессия (LogisticRegression), метод опорных векторов (SVM), деревья решений (Decision Tree), случайный лес (Random Forest), градиентный бустинг (Gradient Boosting), K-ближайших соседей (k-Nearest Neighbors, KNN), гауссовский наивный байес (GNB), нейронные сети (Neural Network). Для применения перечисленных моделей импортируем библиотеки:

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

Классификацию на основе нейронных сетей создаем самостоятельно при помощи библиотеки

```
tensorflow import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorica
```

Для оценки точности пользуемся основными метриками оценивания моделей, а именно accuracy (точность), precision (точность), recall (полнота), F1 score (F1-мера). Каждая модель имеет свой личный набор перечисленных метрик, по которым можно сравнить их между собой (рис. 10).

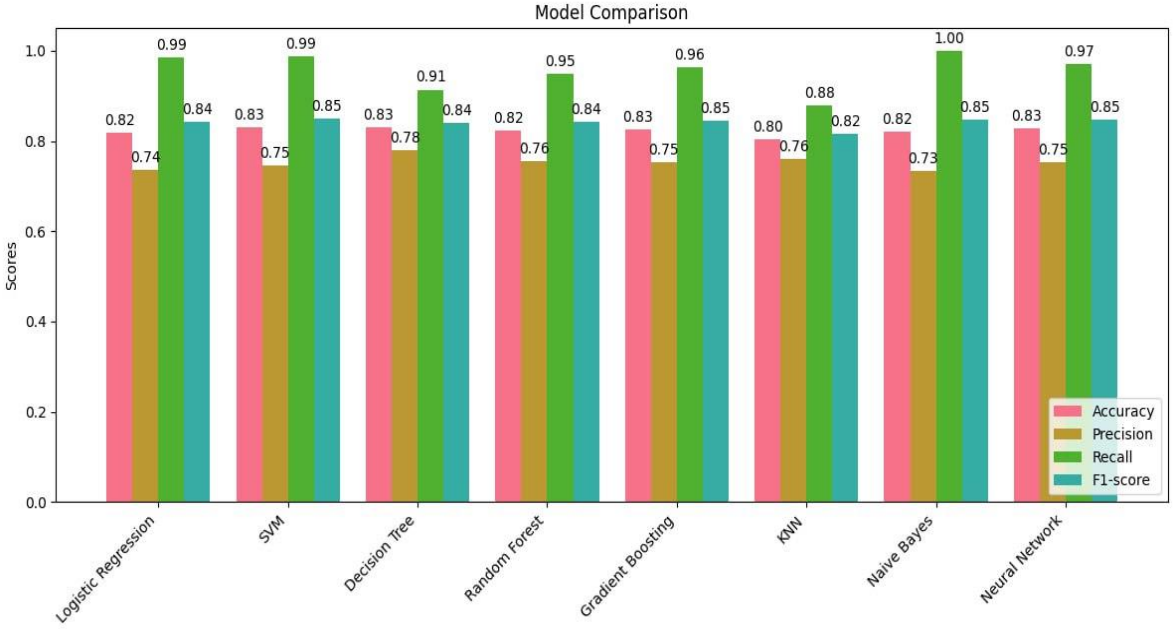

Рис. 10. Диаграмма точностей

Для оценки моделей используем матрицу ошибок. Матрица ошибок – это таблица, которая показывает количество правильных и неправильных прогнозов, сделанных моделью. Матрица ошибок состоит из четырех основных компонент: истинные положительные, истинные отрицательные, ложно положительные, ложно отрицательные. Рассмотрим их более подробно для каждой модели классификации (рис. 11-18).

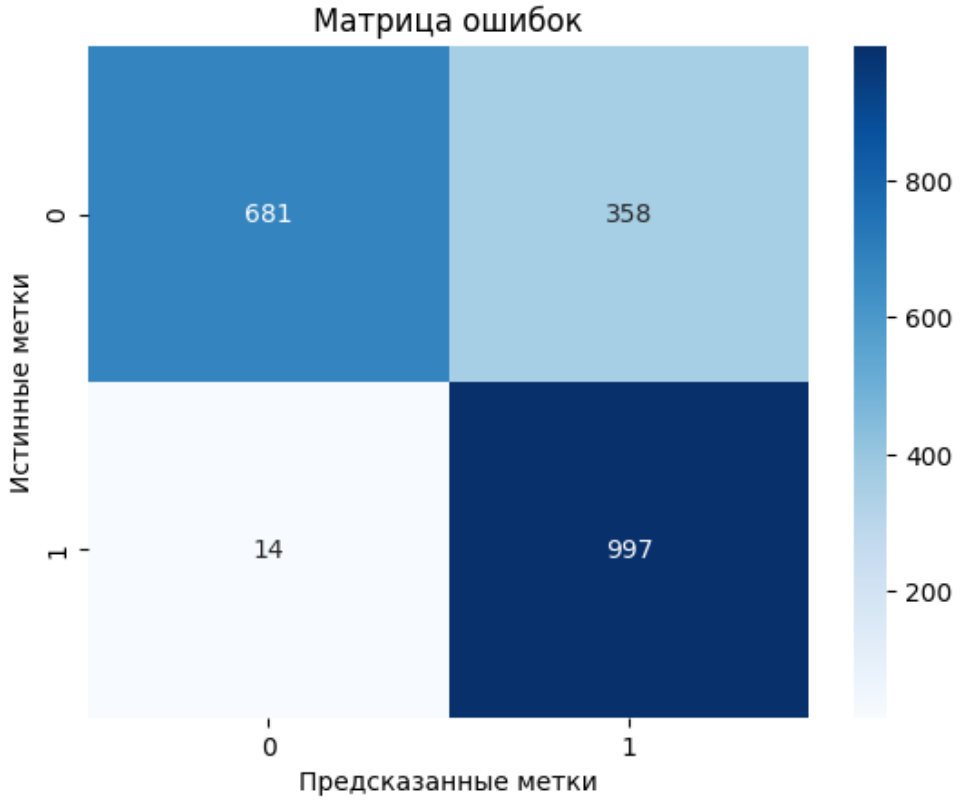

Рис. 11. Логистическая регрессия (LogisticRegression)

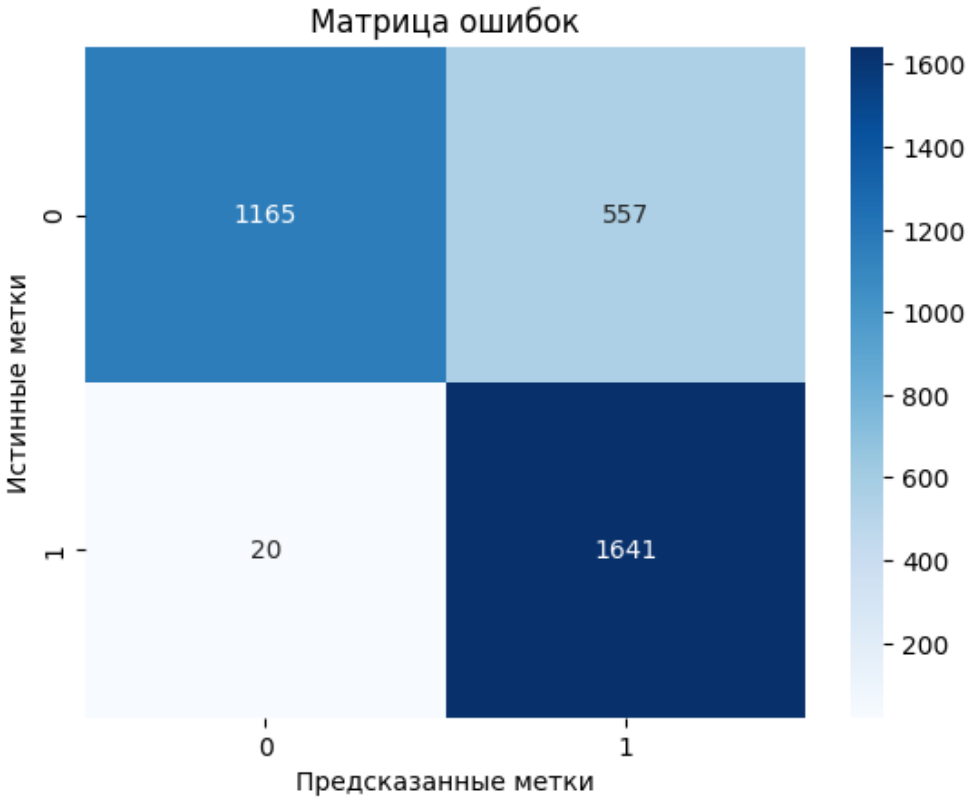

Рис. 12. Метод опорных векторов (SVM)

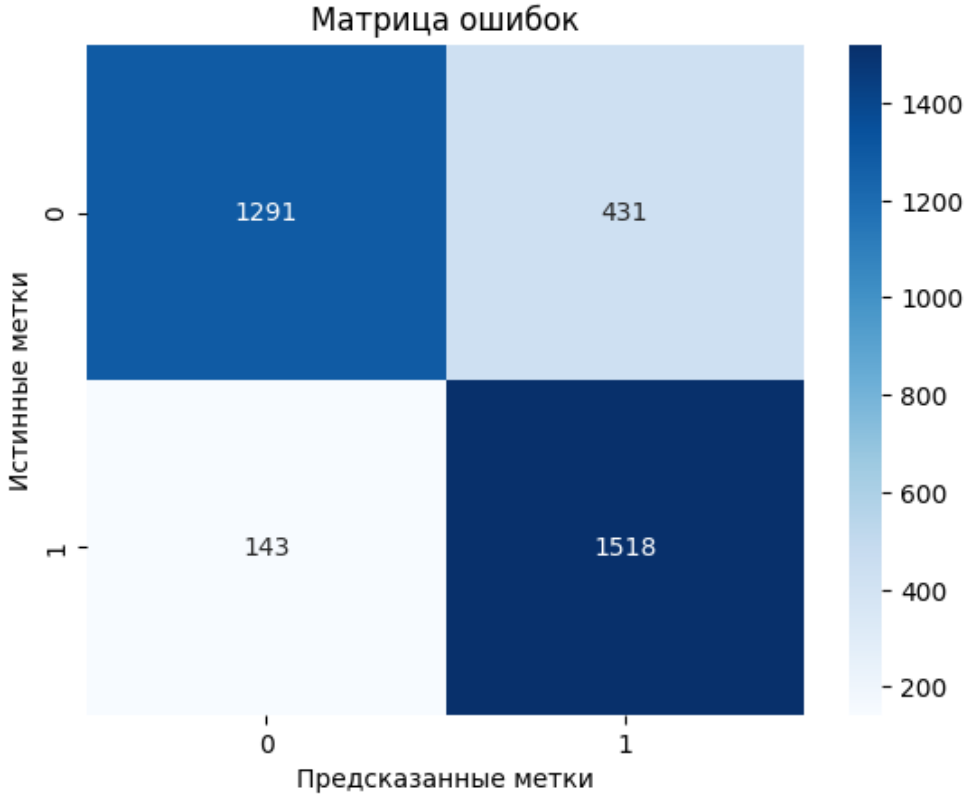

Рис. 13. Деревья решений (Decision Tree)

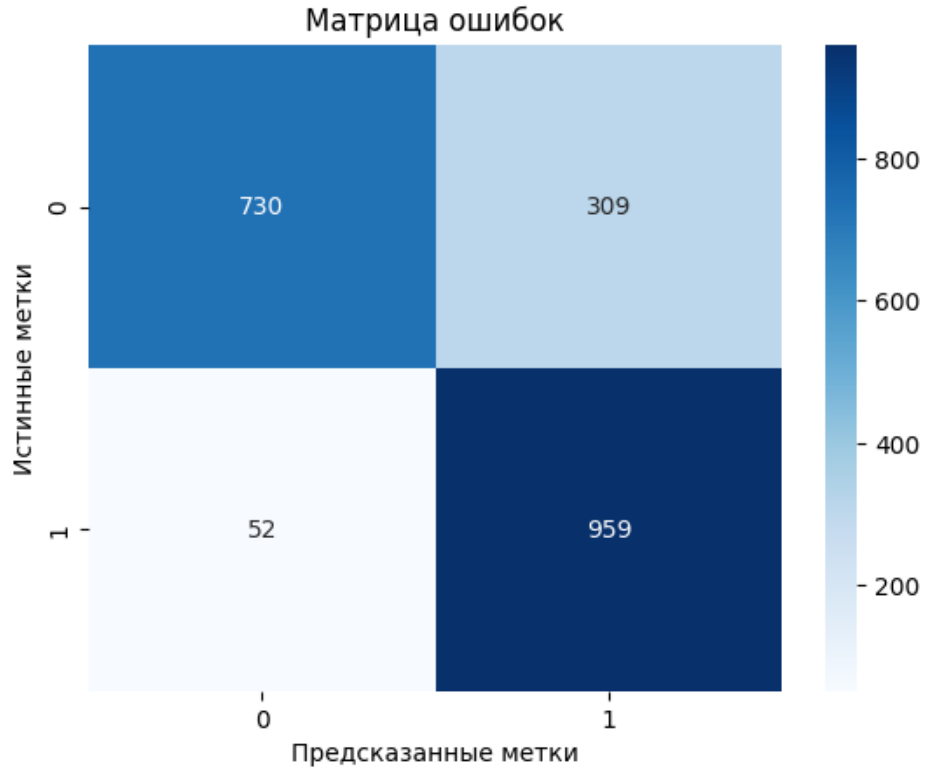

Рис. 14. Случайный лес (Random Forest)

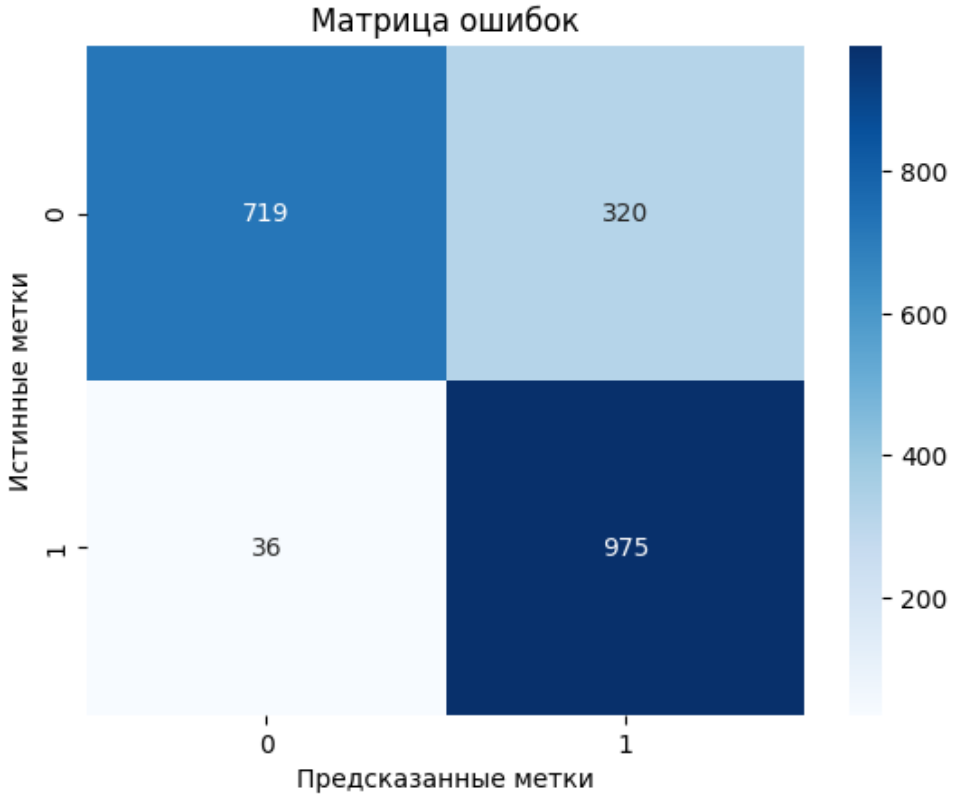

Рис. 15. Градиентный бустинг (Gradient Boosting)

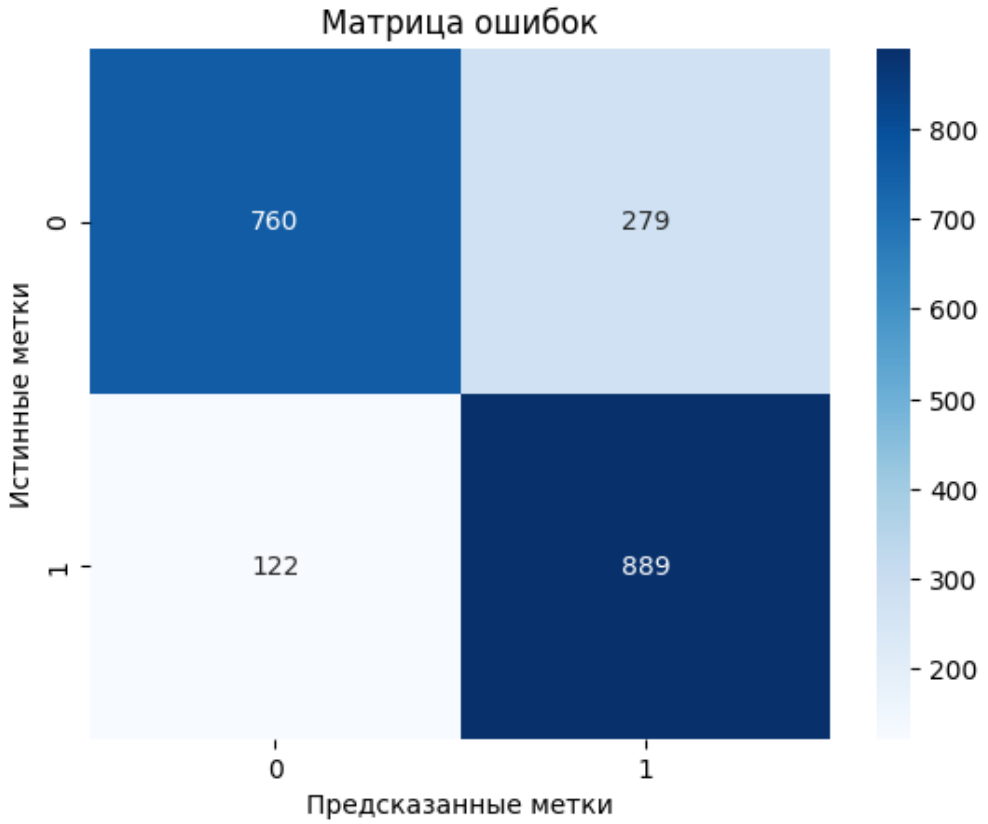

Рис. 16. K-ближайших соседей (k-Nearest Neighbors, KNN)

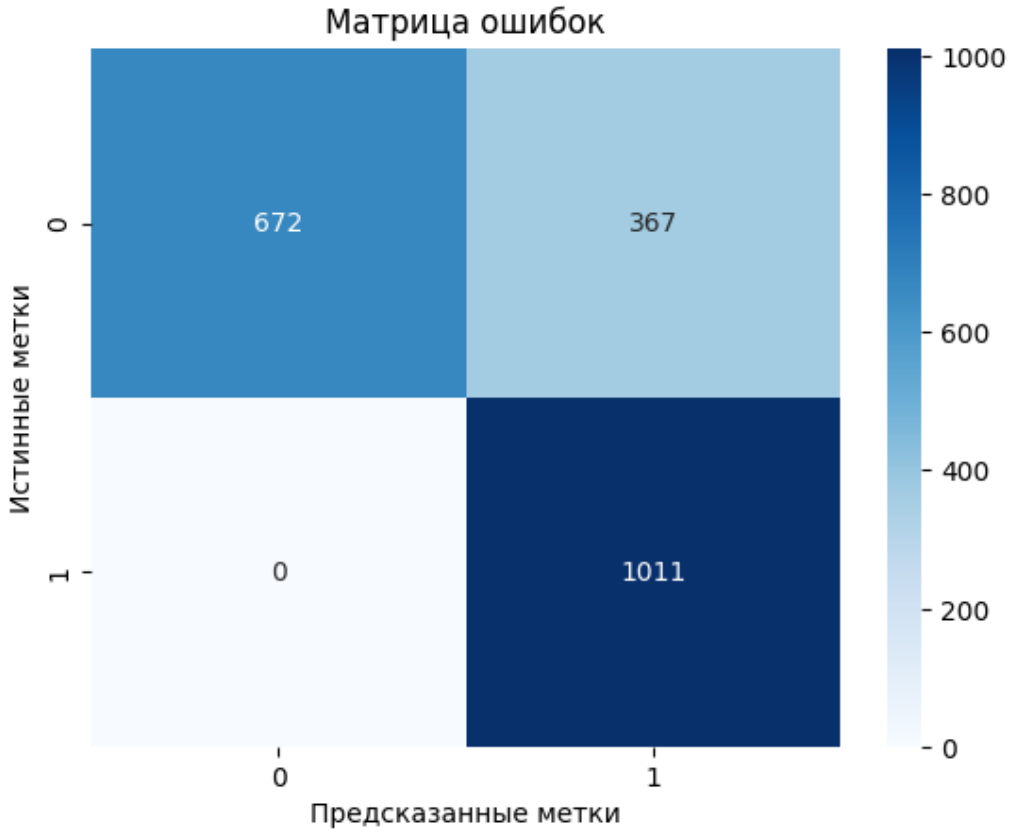

Рис. 17. Гауссовский наивный байес (GNB)

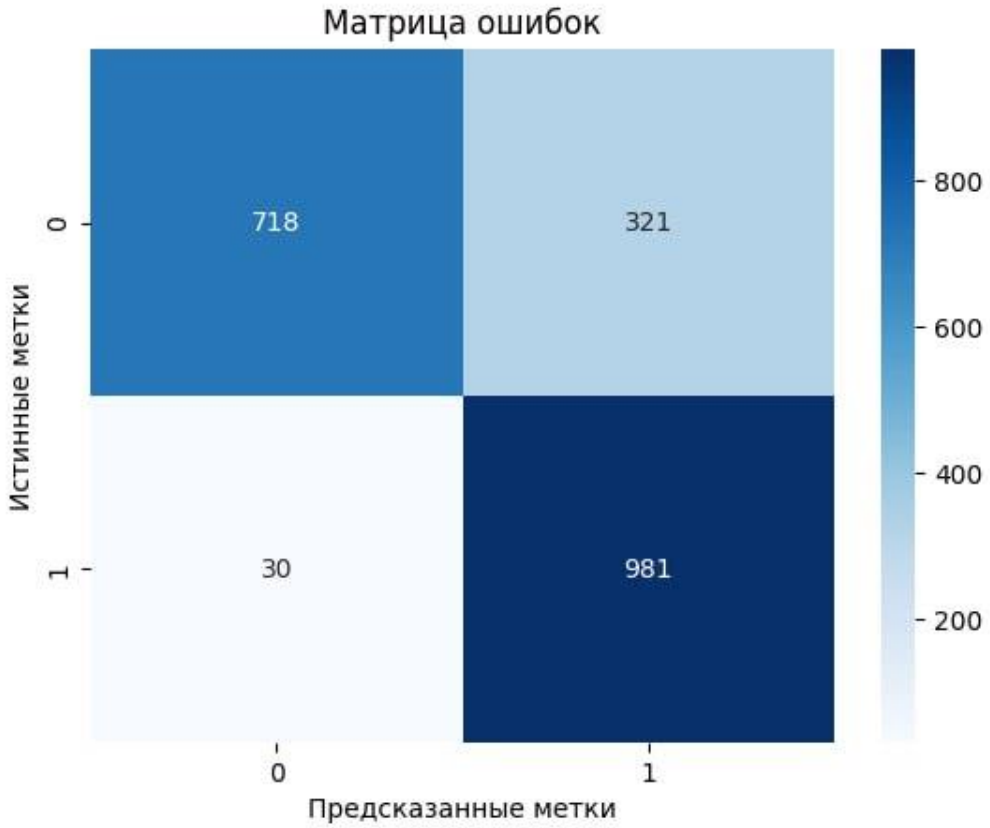

Рис. 18. Нейронные сети (Neural Network)

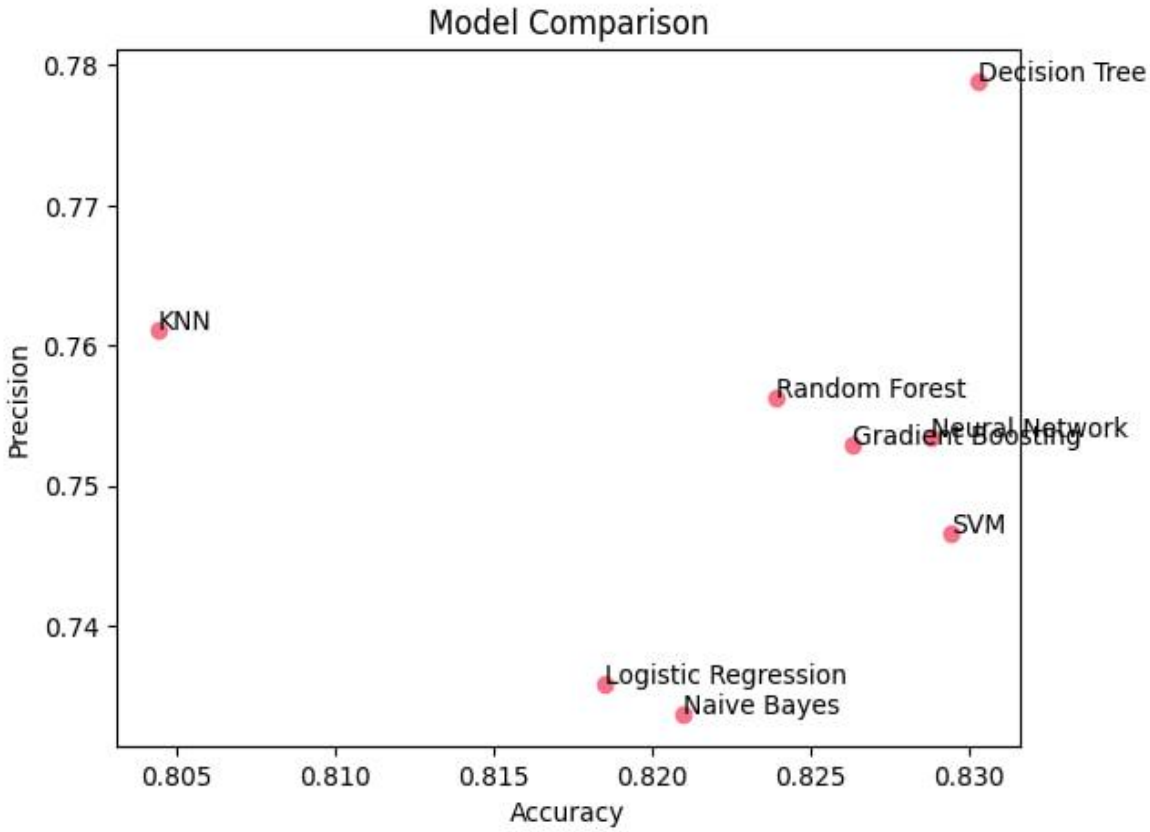

Рис. 19. График сравнения моделей.

Заключение

На основе проведенного анализа можно сделать вывод, что выдвинутая гипотеза подтвердилась: решение о выдаче кредитов может быть предсказано с помощью модели машинного обучения.

Внедрение машинного обучения в процесс принятия решений о выдаче кредитов значительно упрощает оценку кредитоспособности заемщиков. Модели, основанные на анализе данных, могут учитывать множество факторов, включая кредитную историю, финансовое поведение и социально-экономические условия, что повышает уровень финансовой устойчивости как для кредиторов, так и для заемщиков.

Все использованные модели машинного обучения продемонстрировали схожий уровень точности, однако наивысшие результаты были показаны моделью дерева решений. Таким образом, поставленная цель была достигнута: на основе кредитной истории разработаны модели классификации, способные давать прогнозы с высокой степенью точности. Для достижения цели проекта были выполнены следующие задачи:

− Анализ проблемы и обоснование ее значимости;

− Загрузка данных в среду разработки и подготовка к исследованию;

− Проведение анализа данных: выявление и устранение выбросов, балансировка датасета и проверка зависимостей переменных;

− Создание моделей классификации, построение матриц ошибок и определение ключевых метрик для анализа результатов.

Построенная модель может быть применена банками для снижения трудозатрат.

Список использованных источников и литературы

1. Вандер П. Python для сложных задач: наука о данных и машинное обучение. — СПб.: Питер, 2018. — 576 с.
2. Васильев, А.Н. Программирование на Python в примерах и задачах / А,Н. Васильев. — Москва : Эксмо, 2021. — 616 с.
3. Криволапов С.Я. Математика на Python : учебник / С.Я. Криволапов, М.Б. Хрипунова. — Москва: КНОРУС, 2022. — 456 с.
4. Маккини У. Python и анализ данных / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 540 с.
5. Саммерфилд М., Python на практике [Электронный ресурс] / Марк Саммерфилд - М. : ДМК Пресс, 2014. - 338 с. - ISBN 978-5-97060-095-5 - Режим доступа:
6. 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 с. // Электронно-библиотечная система IPR BOOKS: [сайт].
7. Федин, Ф. О. Анализ данных. Часть 1. Подготовка данных к анализу : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический университет, 2012. — 204 с. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
8. Федин, Ф. О. Анализ данных. Часть 2. Инструменты DataMining : учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва : Московский городской педагогический университет, 2012. — 308 с. — ISBN 2227-8397. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
9. 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.

Приложение

Приложение 1

Программный код

```
# Импортируем библиотеки

import numpy as np
import pandas as pd
import seaborn as sns
sns.set_palette('husl')
import matplotlib.pyplot as plt
%matplotlib inline

#Оптимальные параметры моделей
from sklearn.model_selection import GridSearchCV

#оценка точности моделей
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

#рассматриваемые модели классификации
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis
```

```
# открываем датасет
data = pd.read_csv('loan_data.csv')
data.head()

# Выведем название столбцов
data.columns

data.index

# Есть ли пропуски?
data.info()

#Пропусков нет.Также можно посмотреть тип каждого переменной(признака)

# Исследование на выбросы числовые признаки
numerical_features = ['person_age', 'person_income', 'person_emp_exp',
                      'loan_amnt', 'loan_int_rate', 'loan_percent_income',
                      'cb_person_cred_hist_length', 'credit_score']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_features):
    plt.subplot(3, 3, i + 1)
    sns.boxplot(y=data[col])
    plt.title(col)
plt.tight_layout()
plt.show()

data.describe()
```

```
    # здесь можно посмотреть количество наблюдений, среднее значение, стандартное отклонение, квартили,
    # минимальное и максимальное значение для каждгого численного признакак

    # Определение функции для удаления выбросов с использованием межквартильного размаха (IQR)
    def remove_outliers_iqr(data, column):
        Q1 = data[column].quantile(0.25)  # Первый квартиль
        Q3 = data[column].quantile(0.75)  # Третий квартиль
        IQR = Q3 - Q1  # Межквартильный размах
        lower_bound = Q1 - 1.5 * IQR  # Нижняя граница для выбросов
        upper_bound = Q3 + 1.5 * IQR  # Верхняя граница для выбросов
        data_filtered = data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]  # Фильтрация данных
        return data_filtered


    # Применение функции к каждому числовому столбцу
    for col in numerical_features:
        data = remove_outliers_iqr(data, col)

    #Отображение ящичковых диаграмм снова для визуализации исчезновения выбросов
    plt.figure(figsize=(15, 10))
    for i, col in enumerate(numerical_features):
        plt.subplot(3, 3, i + 1)
        sns.boxplot(y=data[col])
        plt.title(col)
    plt.tight_layout()
    plt.show()

    data = data.dropna()

    data.describe()
```

```
    # проведём нормализацию численных признаков, чтобы модель
одинакова учитывала каждый признак
    from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler()

    # создадим копию датасета, чтобы не было изменений в изначальном
датасете
    df = pd.DataFrame(data)
    df_copy = df.copy()

    df_copy_new                    =                    df_copy.drop(
['person_gender','person_education',
                                'person_home_ownership',
'loan_intent',
                                'previous_loan_defaults_on_file',
                                'loan_status' ],  axis = 1)

    norm_df = scaler.fit_transform(df_copy_new)
    norm_df  =  pd.DataFrame(norm_df,  columns  =  ['person_age',
'person_income', 'person_emp_exp', 'loan_amnt',
    'loan_int_rate',                          'loan_percent_income',
'cb_person_cred_hist_length  ', 'credit_score' ])

    norm_df.head(3)
    # получили нормализованные численные данные


    # Категориальные переменными(в том числе и целевая переменная)
- ['person_gender','person_education',
    #
'person_home_ownership', 'loan_intent',
    #
'previous_loan_defaults_on_file',
    #                                                'loan_status'
]
```

```
df1 = pd.DataFrame(data)
df1_copy = df1.copy()

df1_obj = df1.select_dtypes(include = ['object'])


# Функция для преобразования категориальный значений в численные
def Categorial(Df):
  arr = Df.columns

  for i in arr:
    Df[i] = Df[i].astype('category') # преобразует столбец в
категорию
    Df[i] = Df[i].cat.codes # присваем численное значение

  return Df

df1_obj_int = Categorial(df1_obj)
df1_obj_int.head(3)




# Количестов людей по гендеру
df1_obj_int['person_gender'].value_counts()

# 1 - мужчины
# 0 - женщины

# Количество людей по дефолту кредита
df1_obj_int['previous_loan_defaults_on_file'].value_counts()

# 1 - был(и) дефолт(ы)
# 0 - не было дефолтов
```

```
Data = pd.concat([norm_df, df1_obj_int], axis = 1) # объединяем нормальзованные переменные с категориальными
Data = Data.dropna()
Data['loan_status'] = data['loan_status'] # добавляем целевую переменную
Data.head(3) # Итоговый датасет

# Количество одобренных и неодобренных кредитов
Data['loan_status'].value_counts()

# можно применить технику downsampling (уменьшение количества данных)

# Разделение на два датафрейма по классу
class_0 = Data.query("loan_status == 0")
class_1 = Data.query("loan_status == 1")

# Downsampling класса 0 до размера класса 1
sampled_class_0 = class_0.sample(n=len(class_1), random_state=42)

"""np.random.seed(42) устанавливает начальное число генератора случайных чисел,
чтобы результаты были воспроизводимыми."""

# Объединение двух датафреймов
balanced_df = pd.concat([sampled_class_0, class_1], ignore_index=True)

# Проверка нового распределения классов
#print(balanced_df['loan_status'].value_counts())

DF = balanced_df.dropna()
DF.head(3)
```

```
    # тепловая карта

    corr = DF.corr()

    plt.figure(figsize = (14, 14))
    sns.heatmap(corr, annot = True, cmap = 'coolwarm', fmt='.2f',
square=True)
    plt.title('Матрица корреляций сбалансированной подвыборки',
fontsize=20)
    plt.show()

    from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve, auc
    from sklearn.decomposition import PCA



    X = DF.drop(['loan_status'], axis = 1)
    y = DF['loan_status']

    print(f'X shape: {X.shape} | y shape: {y.shape} ')

    # Разделение данных на обучающую и тестовую выборки
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Создание объекта классификатора логистической регрессии
    clf = LogisticRegression(random_state=42)

    # Обучение модели
    clf.fit(X_train, y_train)

    # Прогнозирование на тестовых данных
    y_pred_log = clf.predict(X_test)



    # Вычислим различные метрики
```

```
    accuracy_log = accuracy_score(y_test, y_pred_log)
    precision_log = precision_score(y_test, y_pred_log)
    recall_log = recall_score(y_test, y_pred_log)
    f1_log = f1_score(y_test, y_pred_log)

    # Оценка точности модели
    print(f'Точность модели: {clf.score(X_test, y_test)}')
    print()
    # Выведем результаты
    print("Accuracy:", accuracy_log)
    print("Precision:", precision_log)
    print("Recall:", recall_log)
    print("F1 score:", f1_log)

    # Построение матрицы ошибок
    cm_log = confusion_matrix(y_test, y_pred_log)

    # Визуализация матрицы ошибок
    sns.heatmap(cm_log, annot=True, fmt='g', cmap='Blues')
    plt.xlabel('Предсказанные метки')
    plt.ylabel('Истинные метки')
    plt.title('Матрица ошибок')
    plt.show()


    ################################################## можно ещё
ROC кривую построить


    # Разделение данных на тренировочную и тестовую выборки
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

    # Создание и обучение модели SVM с RBF ядром
    svm = SVC(kernel='rbf', gamma='auto', C=1.0)
    svm.fit(X_train, y_train)
```

```
# Предсказание на тестовой выборке
y_pred_svm = svm.predict(X_test)

# Оценка точности модели
accuracy_svm = accuracy_score(y_test, y_pred_svm)
precision_svm = precision_score(y_test, y_pred_svm)
recall_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)


print(f'Точность модели: {svm.score(X_test, y_test)}')
print()
print("Accuracy:", accuracy_svm)
print("Precision:", precision_svm)
print("Recall:", recall_svm)
print("F1 score:", f1_svm)

# Построение матрицы ошибок
cm_svm = confusion_matrix(y_test, y_pred_svm)

# Визуализация матрицы ошибок
sns.heatmap(cm_svm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Предсказанные метки')
plt.ylabel('Истинные метки')
plt.title('Матрица ошибок')
plt.show()

from sklearn.tree import DecisionTreeClassifier

# Разделение данных на тренировочную и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

# Создание и обучение модели дерева решений
dtc = DecisionTreeClassifier(criterion='gini', max_depth=5)
```

```
dtc.fit(X_train, y_train)

# Предсказание на тестовой выборке
y_pred_tree = dtc.predict(X_test)

# Оценка точности модели
accuracy_tree = accuracy_score(y_test, y_pred_tree)
precision_tree = precision_score(y_test, y_pred_tree)
recall_tree = recall_score(y_test, y_pred_tree)
f1_tree = f1_score(y_test, y_pred_tree)

print(f'Точность модели: {dtc.score(X_test, y_test)}')
print()
print("Accuracy:", accuracy_tree)
print("Precision:", precision_tree)
print("Recall:", recall_tree)
print("F1 score:", f1_tree)

# Построение матрицы ошибок
cm_tree = confusion_matrix(y_test, y_pred_tree)

# Визуализация матрицы ошибок
sns.heatmap(cm_tree, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Предсказанные метки')
plt.ylabel('Истинные метки')
plt.title('Матрица ошибок')
plt.show()

# Разделение данных на обучающие и тестовые наборы
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Создание и обучение модели Random Forest
rf_model = RandomForestClassifier(n_estimators=100,
max_depth=None, random_state=42)
rf_model.fit(X_train, y_train)
```

```
    # Прогнозирование на тестовом наборе
    y_pred_RandF = rf_model.predict(X_test)

    # Оценка точности модели
    accuracy_RandF = accuracy_score(y_test, y_pred_RandF)
    precision_RandF = precision_score(y_test, y_pred_RandF)
    recall_RandF = recall_score(y_test, y_pred_RandF)
    f1_RandF = f1_score(y_test, y_pred_RandF)

    print(f'Точность модели: {rf_model.score(X_test, y_test)}')
    print()
    print("Accuracy:", accuracy_RandF)
    print("Precision:", precision_RandF)
    print("Recall:", recall_RandF)
    print("F1 score:", f1_RandF)

    # Построение матрицы ошибок
    cm_RandF = confusion_matrix(y_test, y_pred_RandF)

    # Визуализация матрицы ошибок
    sns.heatmap(cm_RandF, annot=True, fmt='g', cmap='Blues')
    plt.xlabel('Предсказанные метки')
    plt.ylabel('Истинные метки')
    plt.title('Матрица ошибок')
    plt.show()

    # Разделение данных на обучающие и тестовые наборы
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Создание и обучение модели Gradient Boosting
    gb_model = GradientBoostingClassifier(n_estimators=100,
learning_rate=0.1, max_depth=3, random_state=42)
    gb_model.fit(X_train, y_train)
```

```
# Прогнозирование на тестовом наборе
y_pred_gb = gb_model.predict(X_test)

# Оценка точности модели
accuracy_gb = accuracy_score(y_test, y_pred_gb)
precision_gb = precision_score(y_test, y_pred_gb)
recall_gb = recall_score(y_test, y_pred_gb)
f1_gb = f1_score(y_test, y_pred_gb)

print(f'Точность модели: {gb_model.score(X_test, y_test)}')
print()
print("Accuracy:", accuracy_gb)
print("Precision:", precision_gb)
print("Recall:", recall_gb)
print("F1 score:", f1_gb)

# Построение матрицы ошибок
cm_gb = confusion_matrix(y_test, y_pred_gb)

# Визуализация матрицы ошибок
sns.heatmap(cm_gb, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Предсказанные метки')
plt.ylabel('Истинные метки')
plt.title('Матрица ошибок')
plt.show()


from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Создание и обучение модели kNN
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Прогнозирование на тестовом наборе
```

```
y_pred_knn = knn_model.predict(X_test)

# Оценка точности модели
accuracy_knn = accuracy_score(y_test, y_pred_knn)
precision_knn = precision_score(y_test, y_pred_knn)
recall_knn = recall_score(y_test, y_pred_knn)
f1_knn = f1_score(y_test, y_pred_knn)

print(f'Точность модели: {knn_model.score(X_test, y_test)}')
print()
print("Accuracy:", accuracy_knn)
print("Precision:", precision_knn)
print("Recall:", recall_knn)
print("F1 score:", f1_knn)

# Построение матрицы ошибок
cm_knn = confusion_matrix(y_test, y_pred_knn)

# Визуализация матрицы ошибок
sns.heatmap(cm_knn, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Предсказанные метки')
plt.ylabel('Истинные метки')
plt.title('Матрица ошибок')
plt.show()



from sklearn.naive_bayes import GaussianNB

# Разделение данных на обучающие и тестовые наборы
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Создание и обучение модели Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

```
# Прогнозирование на тестовом наборе
y_pred_NB = nb_model.predict(X_test)

# Оценка точности модели
accuracy_NB = accuracy_score(y_test, y_pred_NB)
precision_NB = precision_score(y_test, y_pred_NB)
recall_NB = recall_score(y_test, y_pred_NB)
f1_NB = f1_score(y_test, y_pred_NB)

print(f'Точность модели: {nb_model.score(X_test, y_test)}')
print()
print("Accuracy:", accuracy_NB)
print("Precision:", precision_NB)
print("Recall:", recall_NB)
print("F1 score:", f1_NB)

# Построение матрицы ошибок
cm_NB = confusion_matrix(y_test, y_pred_NB)

# Визуализация матрицы ошибок
sns.heatmap(cm_NB, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Предсказанные метки')
plt.ylabel('Истинные метки')
plt.title('Матрица ошибок')
plt.show()




import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical

"""
# Преобразование меток в категориальные
```

```
    y_train_cat = to_categorical(y_train)
    y_test_cat = to_categorical(y_test)

    # Определение архитектуры нейронной сети
    model = Sequential([
        Dense(64,                                   activation='relu',
input_shape=(X_train.shape[1],)),
        Dense(32, activation='relu'),
        Dense(16, activation='relu'),
        Dense(2, activation='softmax')
    ])

    # Компилирование модели
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    # Обучение модели
    history = model.fit(X_train, y_train_cat,
                        epochs=10,
                        batch_size=32,
                        validation_data=(X_test, y_test_cat))

    # Прогнозирование на тестовых данных
    y_pred_neuro = model.predict_classes(X_test)

    # Оценка точности модели
    accuracy = accuracy_score(y_test, y_pred_neuro)
    print(f'Accuracy: {accuracy:.2f}')
    """

    # Создание модели
    model_n = Sequential()
    model_n.add(Dense(64,                           activation='relu',
input_shape=(X_train.shape[1],)))
    model_n.add(Dense(32, activation='relu'))
```

```
model_n.add(Dense(16, activation='relu'))
model_n.add(Dense(1, activation='sigmoid'))

# Компиляция модели
model_n.compile(optimizer='adam',
                loss='binary_crossentropy',
                metrics=['accuracy'])

# Обучение модели
history = model_n.fit(X_train, y_train,
                      epochs=10,
                      validation_data=(X_test, y_test))

# Оценка модели
test_loss, test_acc = model_n.evaluate(X_test, y_test)
print()
print(f'Тестовая точность: {test_acc:.2f}')
print()

y_pred_neuro_probs = model_n.predict(X_test)
y_pred_neuro = (y_pred_neuro_probs > 0.5).astype(int)


accuracy_neuro = accuracy_score(y_test, y_pred_neuro)
precision_neuro = precision_score(y_test, y_pred_neuro)
recall_neuro = recall_score(y_test, y_pred_neuro)
f1_neuro = f1_score(y_test, y_pred_neuro)

print(f'Accuracy: {accuracy_neuro:.4f}')
print(f'Precision: {precision_neuro:.4f}')
print(f'Recall: {recall_neuro:.4f}')
print(f'F1 score: {f1_neuro:.4f}')

# Создание матрицы ошибок
cm_neuro = confusion_matrix(y_test, y_pred_neuro)
```

```
    # Визуализация матрицы ошибок
    sns.heatmap(cm_neuro, annot=True, fmt='g', cmap='Blues')
    plt.ylabel('Истинные метки')
    plt.title('Матрица ошибок')
    plt.show()

    from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
    from sklearn.model_selection import train_test_split

    # Разделение данных на обучающую и тестовую выборки
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

    # Словарь для хранения моделей и их имен
    models = {
        "Логистическая                                    регрессия":
LogisticRegression(random_state=42),
        "SVM":     SVC(kernel='rbf',     gamma='auto',     C=1.0,
probability=True),
        "Деревья решений": DecisionTreeClassifier(criterion='gini',
max_depth=5),
        "Случайный лес": RandomForestClassifier(n_estimators=100,
max_depth=None, random_state=42),
        "Градиентный                                      бустинг":
GradientBoostingClassifier(n_estimators=100,     learning_rate=0.1,
max_depth=3, random_state=42),
        "KNN": KNeighborsClassifier(n_neighbors=5),
        "Гаусовский наивный байес": GaussianNB(),
    }

    # Создание пустого списка для хранения результатов
    results = []

    # Обучение и оценка каждой модели
    for model_name, model in models.items():
```

```
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        # Вычисление метрик
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        recall = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        # Добавление результатов в список
        results.append([model_name, accuracy, precision, recall,
f1])

    results.append(["Neural Network", accuracy_neuro,
precision_neuro, recall_neuro, f1_neuro])


    # Создание DataFrame из списка результатов
    results_df = pd.DataFrame(results, columns=["Модель",
"Accuracy", "Precision", "Recall", "F1-мера"])

    # Вывод таблицы результатов
    results_df.to_excel('model_comparison.xlsx', index=False)
    results_df


    # Данные для графика
    models = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random
Forest', 'Gradient Boosting', 'KNN', 'Naive Bayes', 'Neural Network']
    accuracy = [accuracy_log, accuracy_svm, accuracy_tree,
accuracy_RandF, accuracy_gb, accuracy_knn, accuracy_NB,
accuracy_neuro]
    precision = [precision_log, precision_svm, precision_tree,
precision_RandF, precision_gb, precision_knn, precision_NB,
precision_neuro]
```

```
    recall = [recall_log, recall_svm, recall_tree, recall_RandF,
recall_gb, recall_knn, recall_NB, recall_neuro]
    f1 = [f1_log, f1_svm, f1_tree, f1_RandF, f1_gb, f1_knn, f1_NB,
f1_neuro]

    # Создание столбчатой диаграммы
    x = np.arange(len(models))
    width = 0.2

    fig, ax = plt.subplots(figsize=(12, 6))
    rects1 = ax.bar(x - width*1.5, accuracy, width,
label='Accuracy')
    rects2 = ax.bar(x - width/2, precision, width,
label='Precision')
    rects3 = ax.bar(x + width/2, recall, width, label='Recall')
    rects4 = ax.bar(x + width*1.5, f1, width, label='F1-score')

    # Настройка графика
    ax.set_ylabel('Scores')
    ax.set_title('Model Comparison')
    ax.set_xticks(x)
    ax.set_xticklabels(models, rotation=45, ha='right')
    ax.legend(loc='lower right')

    # Добавление значений над столбцами
    def autolabel(rects):
        for rect in rects:
            height = rect.get_height()
            ax.annotate(f'{height:.2f}',
                        xy=(rect.get_x() + rect.get_width() / 2,
height),
                        xytext=(0, 3),
                        textcoords="offset points",
                        ha='center', va='bottom')

    autolabel(rects1)
```

```
    autolabel(rects2)
    autolabel(rects3)
    autolabel(rects4)

    fig.tight_layout()
    plt.show()

    models = ['Logistic Regression', 'SVM', 'Decision Tree', 'Random
Forest', 'Gradient Boosting', 'KNN', 'Naive Bayes', 'Neural Network']
    accuracy = [accuracy_log, accuracy_svm, accuracy_tree,
accuracy_RandF, accuracy_gb, accuracy_knn, accuracy_NB,
accuracy_neuro]
    precision = [precision_log, precision_svm, precision_tree,
precision_RandF, precision_gb, precision_knn, precision_NB,
precision_neuro]

    fig, ax = plt.subplots()
    ax.scatter(accuracy, precision)

    for i, model in enumerate(models):
      ax.annotate(model, (accuracy[i], precision[i]))

    ax.set_xlabel('Accuracy')
    ax.set_ylabel('Precision')
    ax.set_title('Model Comparison')
    plt.show()
```