Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ГОЛОВНОЙ УЧЕБНО-ИССЛЕДОВАТЕЛЬСКИЙ И МЕТОДИЧЕСКИЙ ЦЕНТР ПРОФЕССИОНАЛЬНОЙ РЕАБИЛИТАЦИИ ЛИЦ С ОГРАНИЧЕННЫМИ ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ

КАФЕДРА СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Решение задачи машинного обучения

Студент ИУ5Ц-84Б (группа) _____ (подпись, дата) К.Р. Падалко (И.О. Фамилия)

Руководитель НИР _____ (подпись, дата) Ю.Е. Гапанюк (И.О. Фамилия)

2025 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ5
(индекс)
В.И. Терехов
(И.О. Фамилия)
(подпись)
(дата)

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Решение задачи машинного обучения

Студент группы ИУ5Ц-84Б Падалко Ксения Романовна
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)
ИССЛЕДОВАТЕЛЬСКАЯ

Источник тематики (кафедра, предприятие, НИР) КАФЕДРА

График выполнения НИР:
25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 75% к 15 нед

Техническое задание:

Решение задачи машинного обучения. Результатом проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 42 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания «07» февраля 2025 г.

Руководитель НИР _____ Ю.Е. Гапанюк
(подпись, дата) (И.О. Фамилия)

Студент _____ К.Р. Падалко
(подпись, дата) (И.О. Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

СОДЕРЖАНИЕ

Введение

Данное исследование основано на анализе набора данных о качестве вина, который представляет собой ценный ресурс для изучения факторов, влияющих на органолептические характеристики вин. Этот набор данных содержит информацию о различных химических и физико-химических свойствах вина, что позволяет глубже понять, как эти факторы влияют на общее восприятие качества продукта.

Целевым признаком для данного исследования выбрана переменная "quality" — оценка качества вина, основанная на дегустации. Этот выбор обусловлен важностью понимания факторов, влияющих на восприятие качества вина, и их взаимодействия с конкретными химическими характеристиками.

В рамках исследования мы сосредоточимся на анализе структуры данных и основных характеристик набора данных:

- fixed acidity — фиксированная кислотность;
- volatile acidity — летучая кислотность;
- citric acid — лимонная кислота;
- residual sugar — остаточный сахар;
- chlorides — хлористые соединения;
- free sulfur dioxide — свободный двуокись серы;
- total sulfur dioxide — общий диоксид серы;
- density — плотность;
- pH — уровень pH;
- sulphates — сульфаты;
- alcohol — содержание алкоголя;
- quality — оценка качества (от 0 до 10).

Анализ этих характеристик позволит выявить возможные корреляции между химическими параметрами и качеством вина, а также определить наиболее значимые факторы, оказывающие влияние на его восприятие.

Таким образом, данное исследование расширит наше понимание химических аспектов, определяющих качество вина, и предоставит основы для разработки рекомендаций по улучшению производства винодельческой продукции.

Поставим цели, которых мы хотим добиться, выполнив данную научно-исследовательскую работу.

Цели:

1) Поиск и выбор набора данных для построения моделей машинного обучения.
2) Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.
5) Выбор метрик для последующей оценки качества моделей.
6) Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.
7) Формирование обучающей и тестовой выборок на основе исходного набора данных.
8) Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9) Подбор гиперпараметров для выбранных моделей.
10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

11) Формирование выводов о качестве построенных моделей на основе выбранных метрик.

Основная часть

Ссылка на датасет: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

1. **Поиск и выбор набора данных для построения моделей машинного обучения.**

```
# Подключаем библиотеки
import numpy as np
import pandas as pd
import seaborn as sns
import streamlit as st
import matplotlib.pyplot as plt
from catboost import Pool, CatBoostClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split, learning_curve
#from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
sns.set(style="ticks")
```

Здесь я переименовал названия каждого столбца, так как в источнике данных не было конкретных наименований для столбцов.

```
# Список колонок датасета
col_list = ['fixed acidity',
            'volatile acidity',
            'citric acid',
            'residual sugar',
            'chlorides',
            'free sulfur dioxide',
            'total sulfur dioxide',
            'density',
            'pH',
            'sulphates',
            'alcohol',
            'quality']

try:
    # Загрузка датасета с правильным разделителем
    data = pd.read_csv('winequality-red.csv', names=col_list, header=0, sep=",")
    print("Файл успешно загружен.")
except FileNotFoundError:
    print("Файл не найден.")
except pd.errors.ParserError:
    print("Ошибка парсинга: проверьте разделитель или формат файла.")
```

```
Файл успешно загружен.
```

Файл успешно загружен, теперь рассматриваем следующее.

2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.

Выводим информацию о датасете.

```
# Выводим информацию о датасета
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

Выводим первые 5 строк датасета.

```
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Подсчитаем количество строк и столбцов в датасете и выводим результат.

```
# Подсчитаем количество строк и столбцов в датасете
num_rows, num_cols = data.shape
# Выводим результат
print(f"Количество столбцов: {num_cols}, количество строк: {num_rows}")
```

```
Количество столбцов: 12, количество строк: 1599
```

Выводим названия колонок.

```
# Выводим названия колонок
data.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

Выводим типы колонок.

```
# Выводим типы колонок
data.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

Практически в каждом колонке имеется тип float64, это значит, что все колонки имеют вещественные числа.

Проверим датасет на пропуски и на дубликаты.

```
# Проверим датасет на пропуски
data.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```
# Проверим датасет на дубликаты
data.duplicated().sum()
```

```
240
```

Пропусков нет, а дубликаты есть. Удаляем дубликаты.

```
# Удаление дубликатов
data_cleaned = data.drop_duplicates()
print(f'Количество строк после удаления дубликатов: {data_cleaned.shape[0]}')

# Сохранение очищенного датасета
data_cleaned.to_csv('winequality-red-cleaned.csv', sep=';', index=False)
```

```
Количество строк после удаления дубликатов: 1359
```

Подсчитаем уникальные значения в столбце датасета.

```
# Подсчитаем уникальные значения в столбце датасета
data['quality'].value_counts()
```

```
quality
5    681
6    638
7    199
4     53
8     18
3     10
Name: count, dtype: int64
```

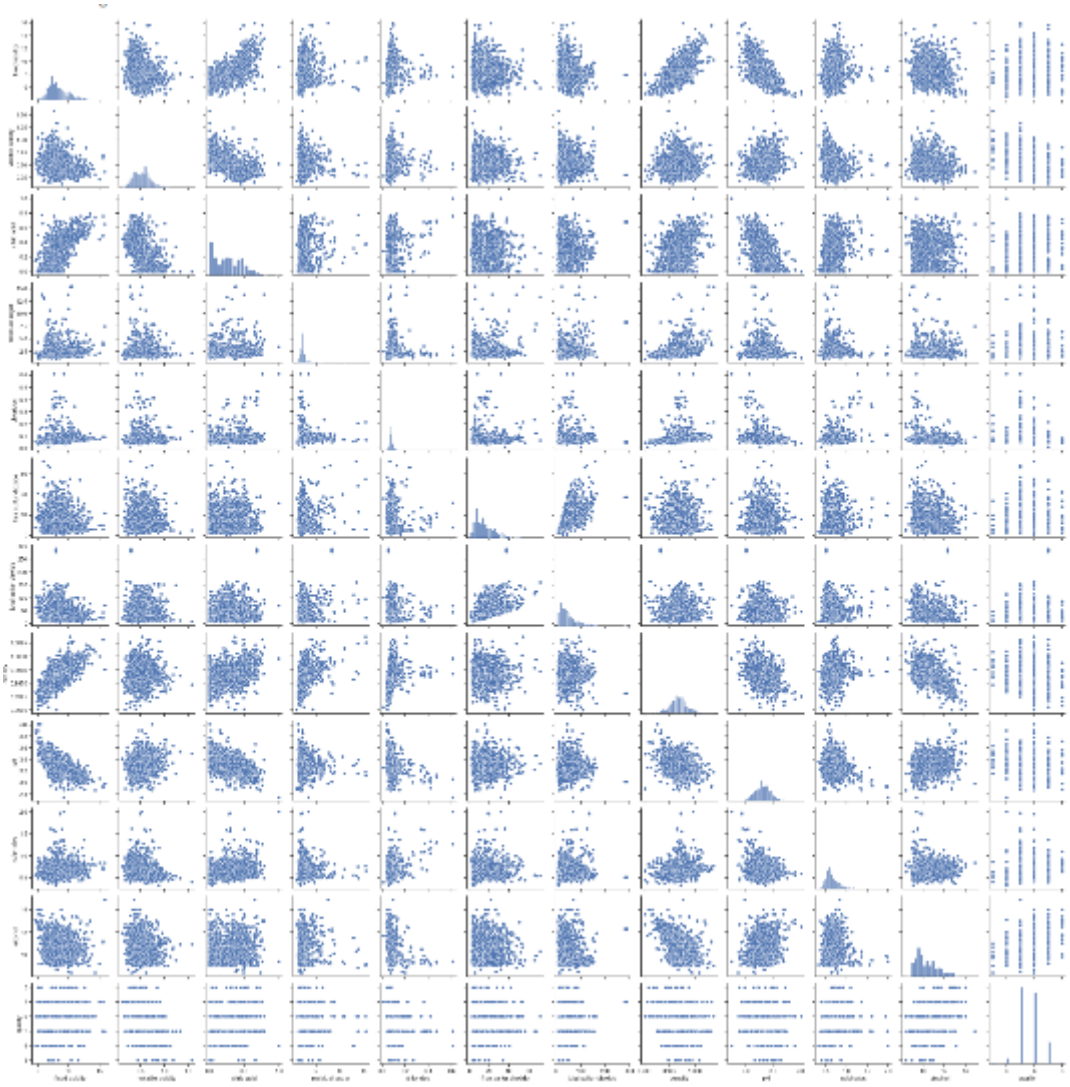
В столбце 'quality' значение '5' встречается 681 раз, а значение '6' - 638 раз, '7' - 199 раз, '4' - 53 раз, '8' - 18 раз, '3' - 10 раз.

Набор данных не содержит пропусков, категориальные признаки

закодированы.

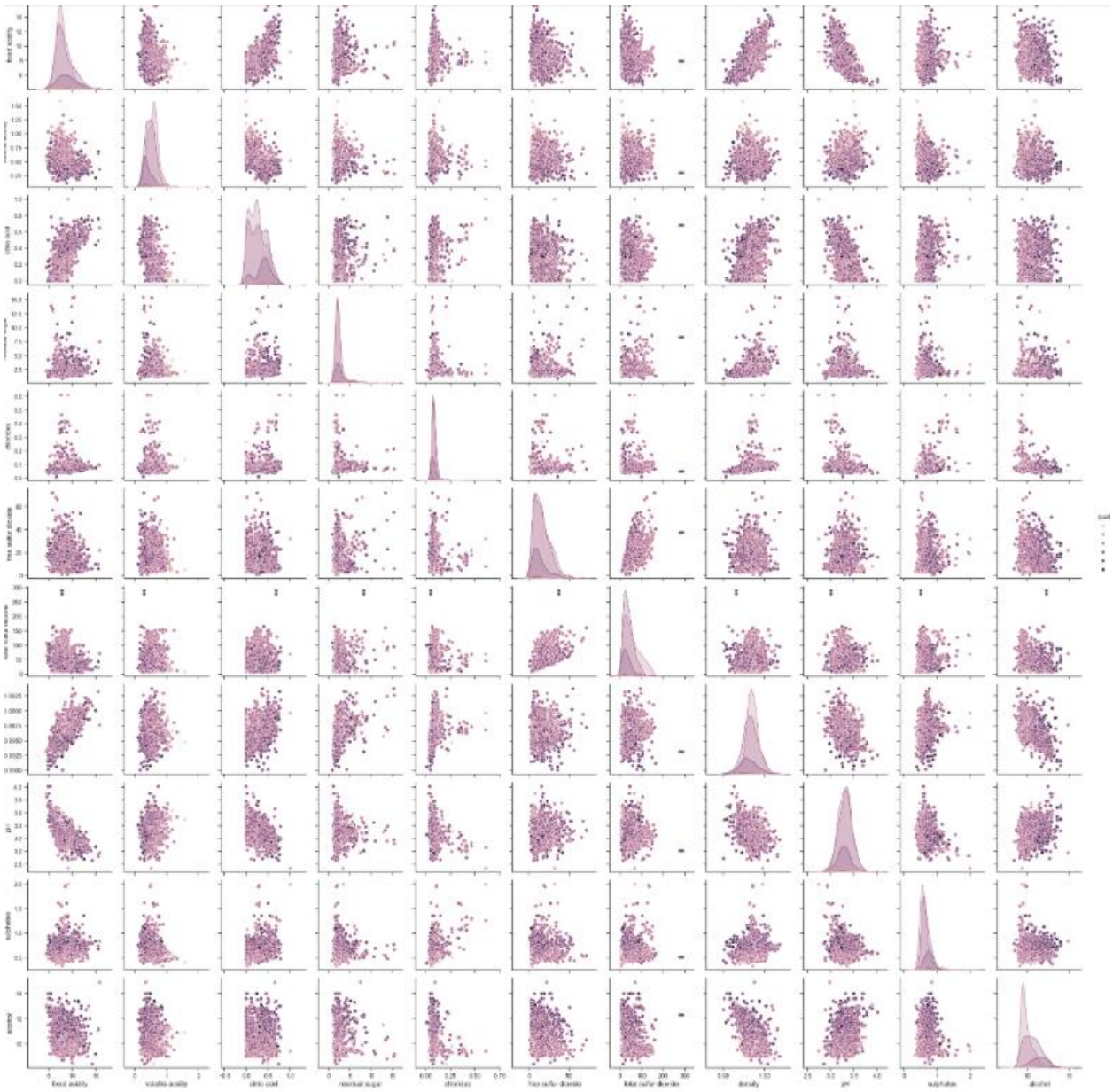
Построим графики рассеяния для всех возможных пар числовых столбцов.

```
# Построим графики рассеяния для всех возможных пар числовых столбцов
sns.pairplot(data)
```



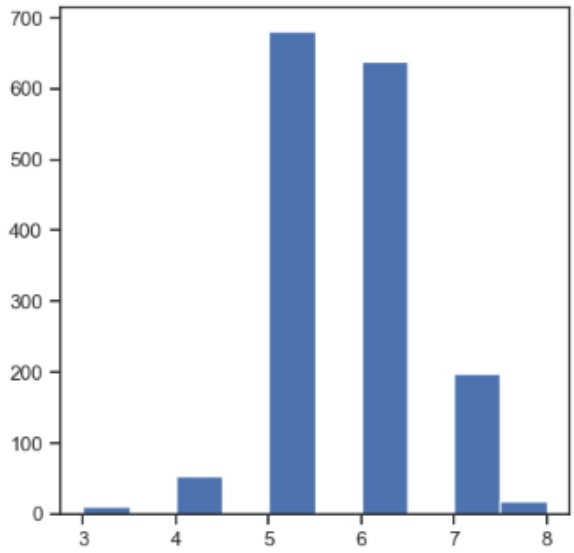
Построим графики рассеяния для всех возможных пар числовых столбцов quality.

```
# Построим графики рассеяния для всех возможных пар числовых столбцов quality
sns.pairplot(data, hue="quality")
```



Оценим дисбаланс классов для quality.

```
# Оценим дисбаланс классов для quality
fig, ax = plt.subplots(figsize=(5,5))
plt.hist(data['quality'])
plt.show()
```




```
data['quality'].value_counts()
```

```
quality
5    681
6    638
7    199
4     53
8     18
3     10
Name: count, dtype: int64
```

Подсчитаем дисбаланс классов.

```
# Подсчитаем дисбаланс качеств
total = data.shape[0]
quality_0, quality_1, quality_2, quality_3, quality_4, quality_5 = data['quality'].value_counts()
print('Качество 8 составляет {}%.\nКачество 7 составляет {}%.\nКачество 6 составляет {}%.\nКачество 5 составляет {}%.\nКачество 4 составляет {}%.\nКаче
      .format(round(quality_0 / total, 4)*100, round(quality_1 / total, 4)*100, round(quality_2 / total, 4)*100, round(quality_3 / total, 4)*100, roun
```

```
Качество 8 составляет 42.59%.
Качество 7 составляет 39.900000000000006%.
Качество 6 составляет 12.45%.
Качество 5 составляет 3.3099999999999996%.
Качество 4 составляет 1.13%.
Качество 3 составляет 0.63%.
```

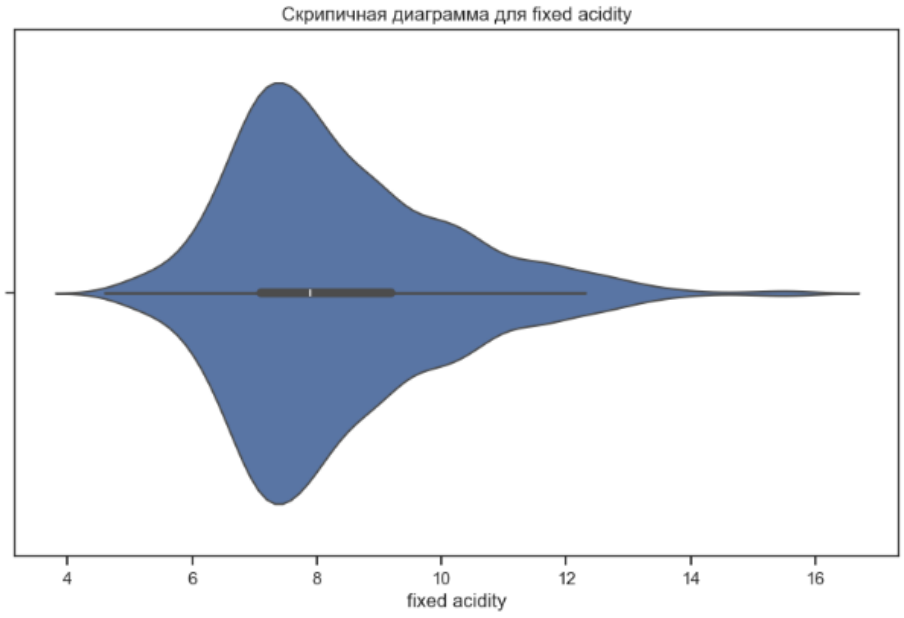
В данном случае, качество 8 составляет примерно 42.59% от общего количества экземпляров, в то время как качество 7 составляет примерно 39.90%. Качество 6 составляет 12.45%, а более низкие качества (5, 4 и 3) составляют значительно меньшую долю — 3.31%, 1.13% и 0.63% соответственно.

Вывод подтверждает наличие дисбаланса качеств в данных. Преобладание качеств 8 и 7 над остальными может оказать влияние на способность модели классификации корректно идентифицировать все уровни качества. В частности, поскольку качества 8 и 7 составляют значительную часть выборки,

модель может быть склонна к неправильному классифицированию экземпляров, относящихся к более низким качествам (5, 4 и 3), как к качествам 8 или 7. Это может привести к снижению общей эффективности модели в предсказании всех уровней качества вина.

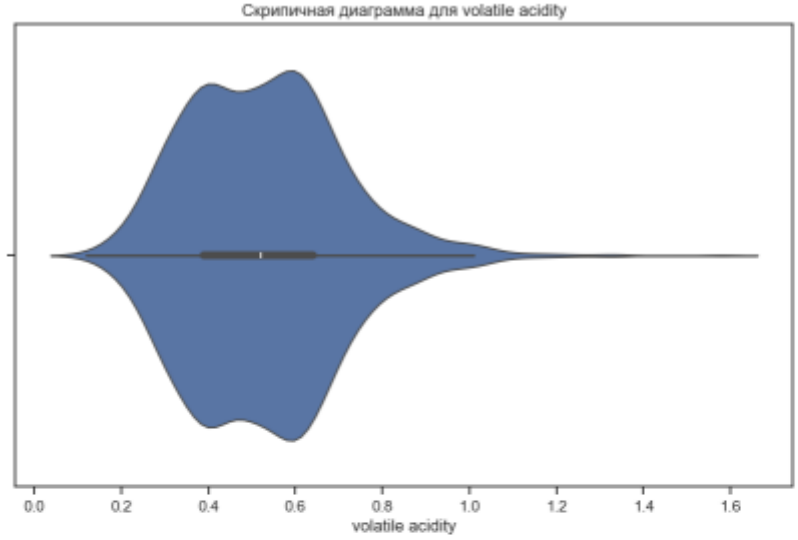
Скрипичные диаграммы для числовых колонок.

```
numeric_columns = [
    'fixed acidity',
    'volatile acidity',
    'citric acid',
    'residual sugar',
    'chlorides',
    'free sulfur dioxide',
    'total sulfur dioxide',
    'density',
    'pH',
    'sulphates',
    'alcohol'
]
# Построение скрипичных диаграмм
for col in numeric_columns:
    plt.figure(figsize=(10, 6))
    sns.violinplot(x=data[col])
    plt.title(f'Скрипичная диаграмма для {col}')
    plt.xlabel(col)
    plt.show()
```

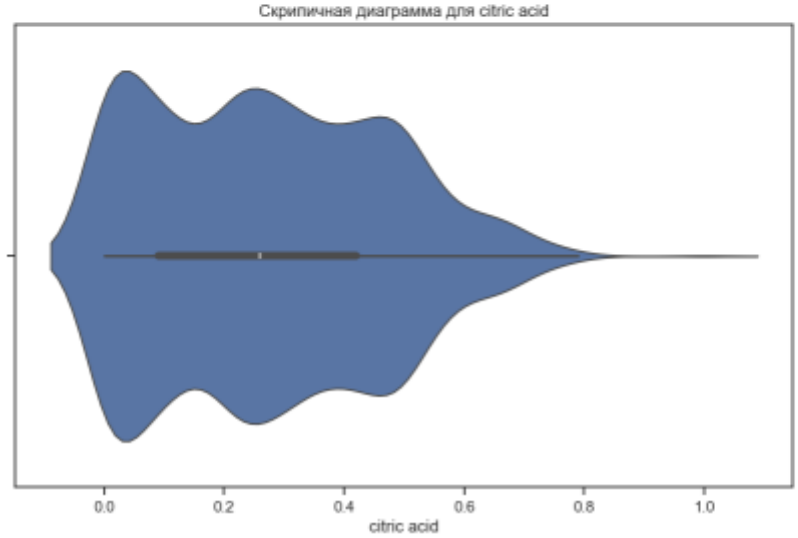


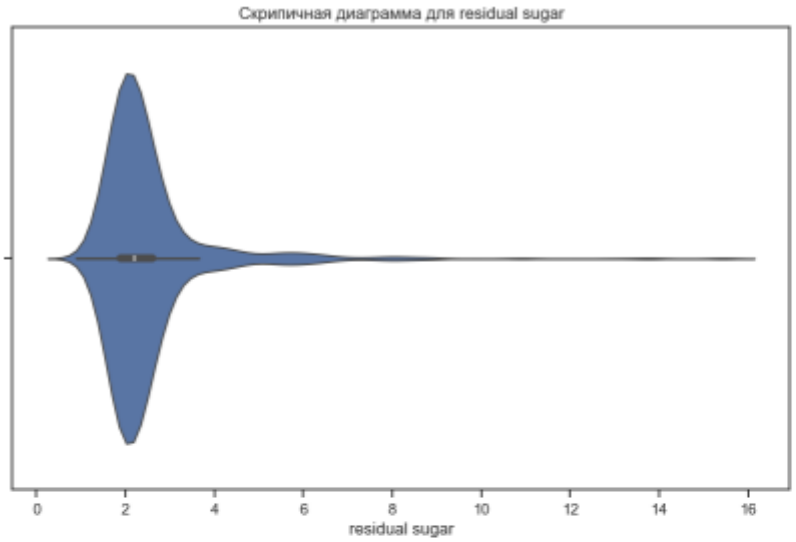
fixed acidity

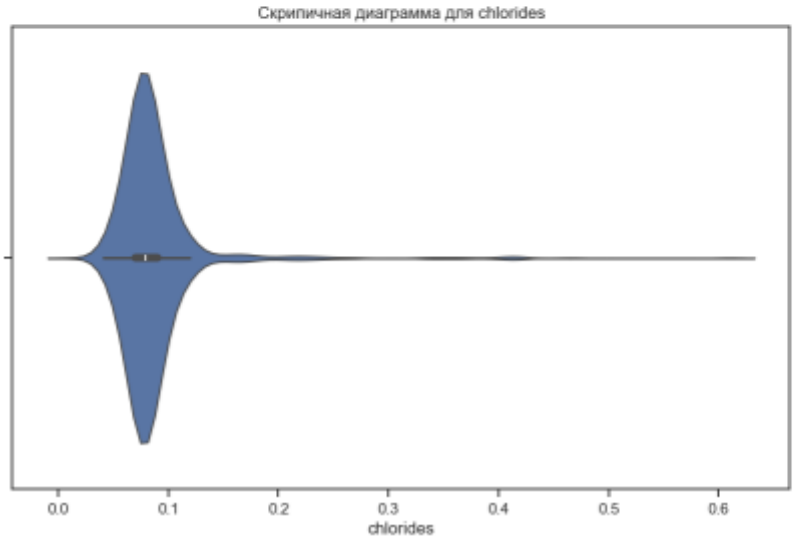
Скрипичная диаграмма для volatile acidity



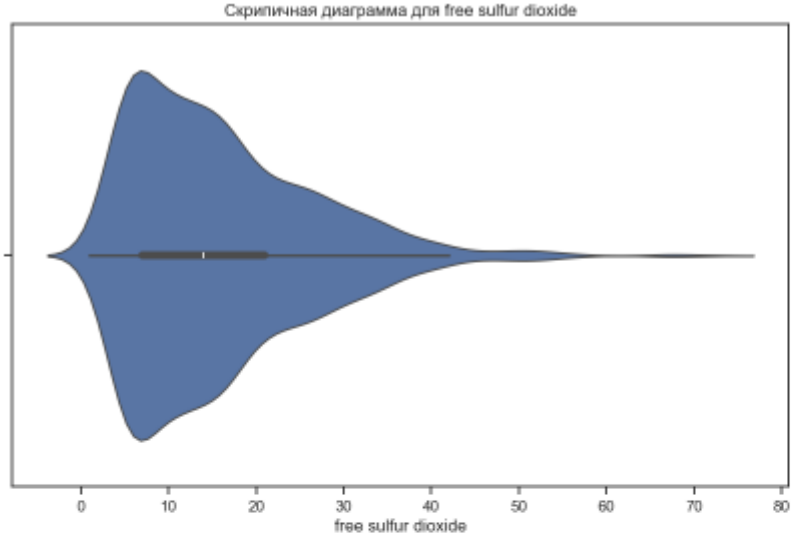
Скрипичная диаграмма для citric acid



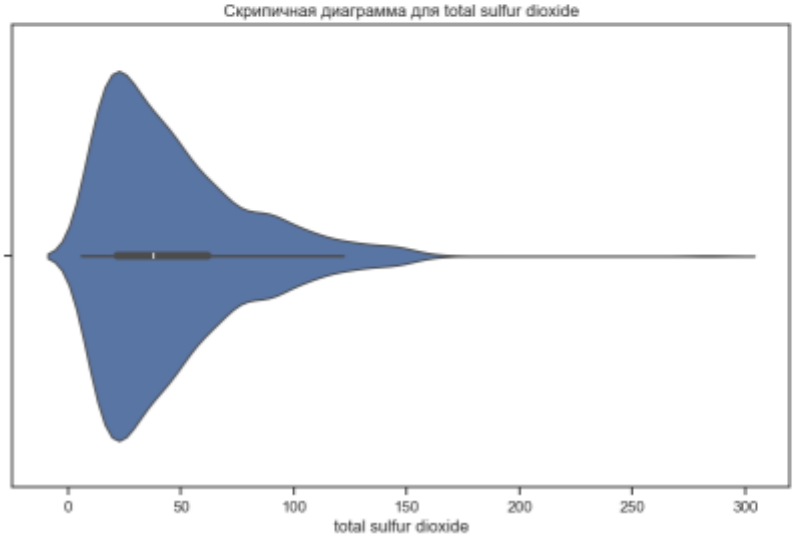

Скрипичная диаграмма для residual sugar
0
2
4
6
8
10
12
14
16
residual sugar

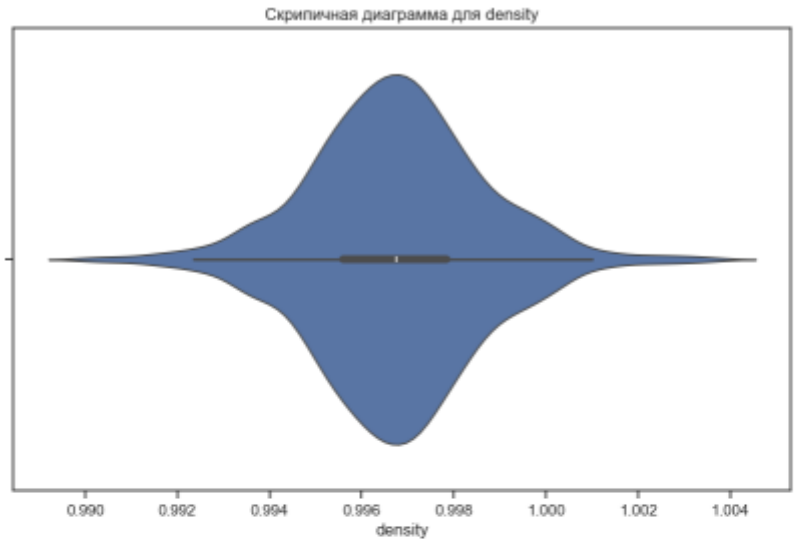

Скрипичная диаграмма для chlorides
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

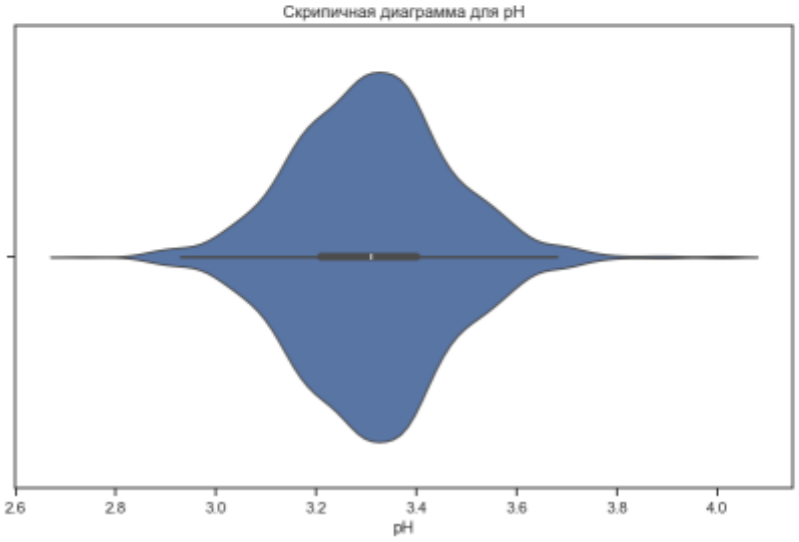
Скрипичная диаграмма для free sulfur dioxide

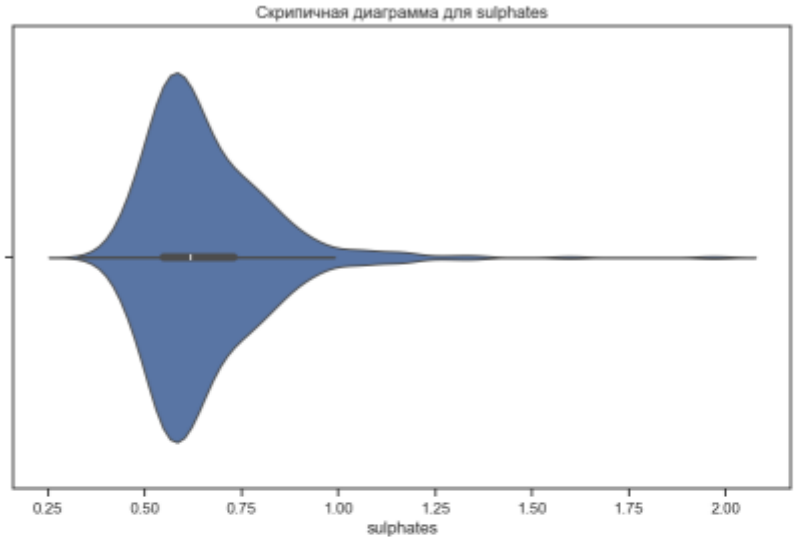


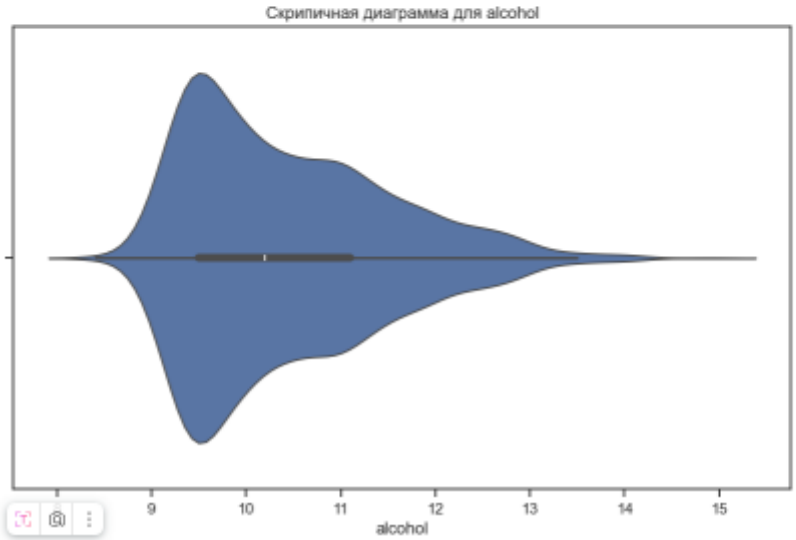
Скрипичная диаграмма для total sulfur dioxide




Скрипичная диаграмма для density
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
density


Скрипичная диаграмма для pH
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH





1. Различие в распределении: Если формы скрипичных диаграмм для разных классов значительно отличаются (например, одна скрипичная диаграмма имеет более широкий разброс значений или имеет более выраженные хвосты), это может указывать на различия в распределении признаков между классами.
2. Выбросы и экстремальные значения: Различия в форме скрипичных диаграмм могут также указывать на наличие выбросов или экстремальных значений в одном из классов.
3. Потенциальная предсказательная сила признаков: Если форма скрипичной диаграммы для одного из классов более выражена или имеет более четкие пики, чем для другого класса, это может указывать

на то, что эти признаки имеют более высокую предсказательную силу для одного из классов.

4. Важность признаков: Различия в форме скрипичных диаграмм могут помочь определить, какие признаки могут быть более важными для разделения классов.

3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

Для построения моделей будем использовать все признаки. Категориальные признаки закодированы, поэтому выполним масштабирование данных.

Числовые колонки для масштабирования данных.

```
scale_cols = [
    'fixed acidity',
    'volatile acidity',
    'citric acid',
    'residual sugar',
    'chlorides',
    'free sulfur dioxide',
    'total sulfur dioxide',
    'density',
    'pH',
    'sulphates',
    'alcohol'
]
```

```
# Создаем экземпляр класса MinMaxScaler
scaler = MinMaxScaler()

# Применяем масштабирование к выбранным колонкам
scaled_data = scaler.fit_transform(data[scale_cols])
```

Добавим масштабированные данные в набор данных.

```
# Добавляем масштабированные данные обратно в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = scaled_data[:, i]
```

Выводим первые 5 строк датасета.

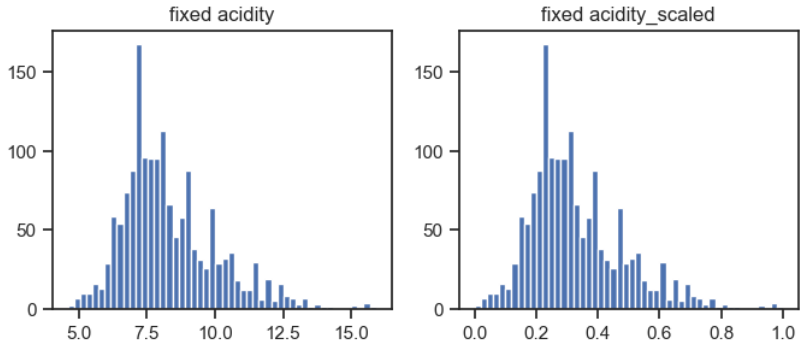
```
# Выводим первые 5 строк датасета
data.head()
```

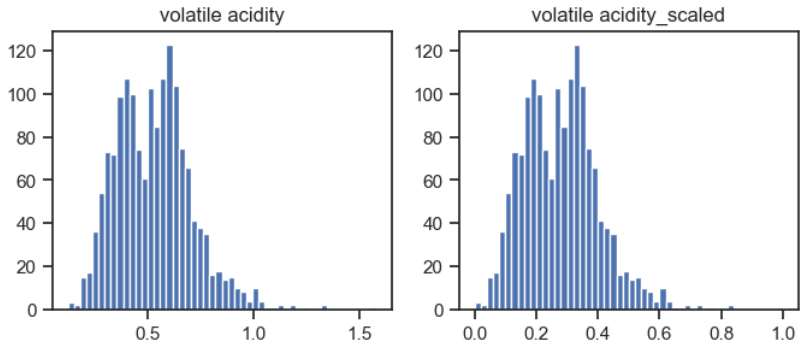
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	...	volatile acidity_scaled	citric acid_scaled	residual sugar_scaled	chlorides_scaled	free sulfur dioxide_scaled
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	...	0.397260	0.00	0.068493	0.106845	0.140845
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	...	0.520548	0.00	0.116438	0.143573	0.338028
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	...	0.438356	0.04	0.095890	0.133556	0.197183
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	...	0.109589	0.56	0.068493	0.105175	0.225352
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	...	0.397260	0.00	0.068493	0.106845	0.140845

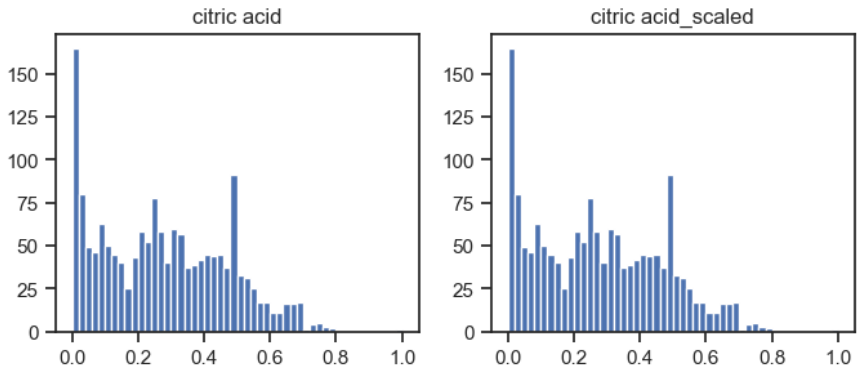
5 rows × 23 columns

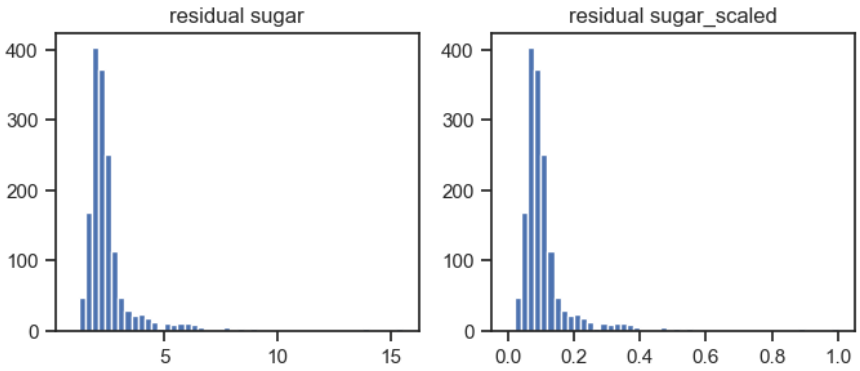
Проверим, что масштабирование не повлияло на распределение данных.

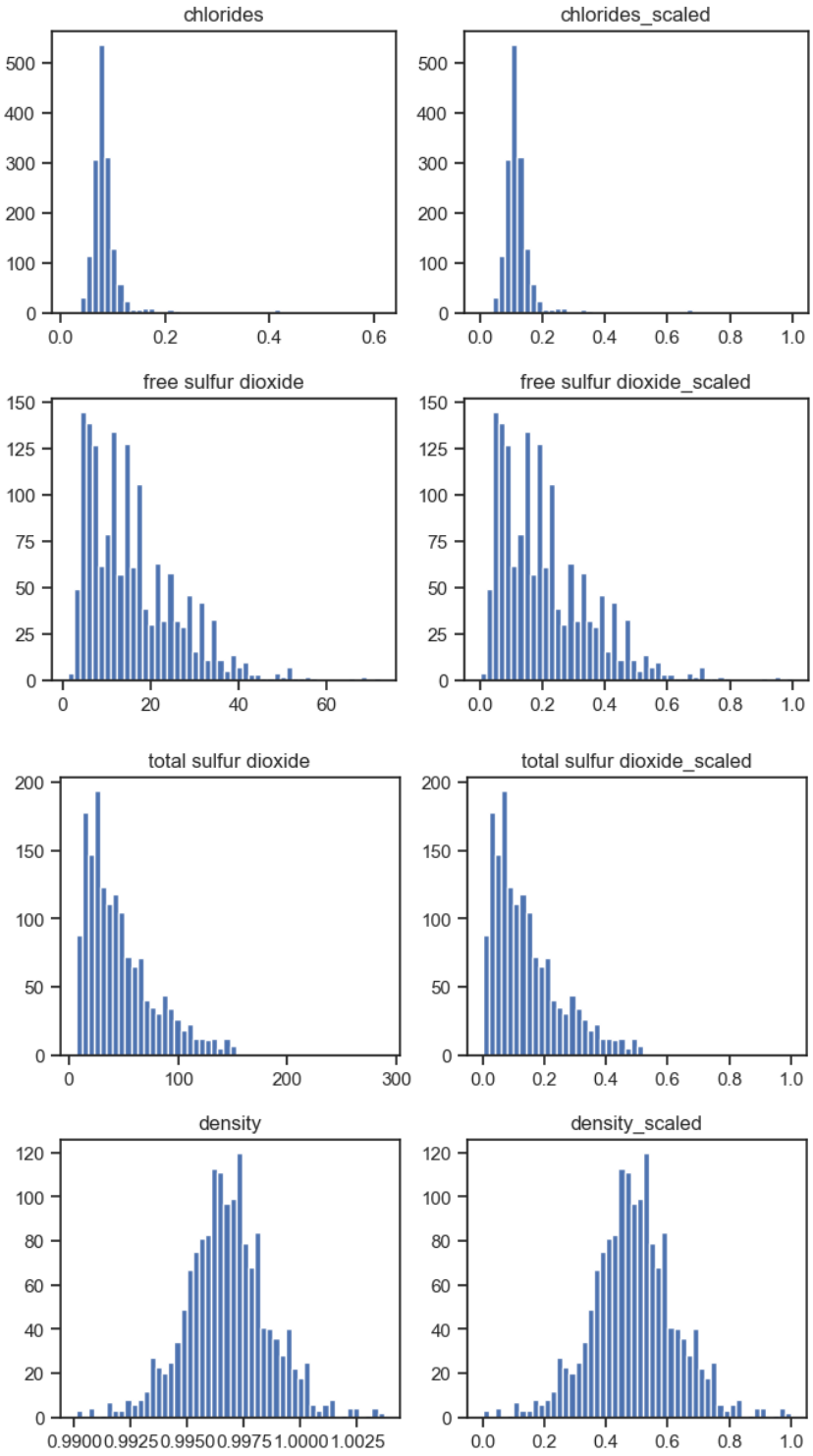
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    fig, ax = plt.subplots(1, 2, figsize=(8,3))
    ax[0].hist(data[col], 50)
    ax[1].hist(data[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

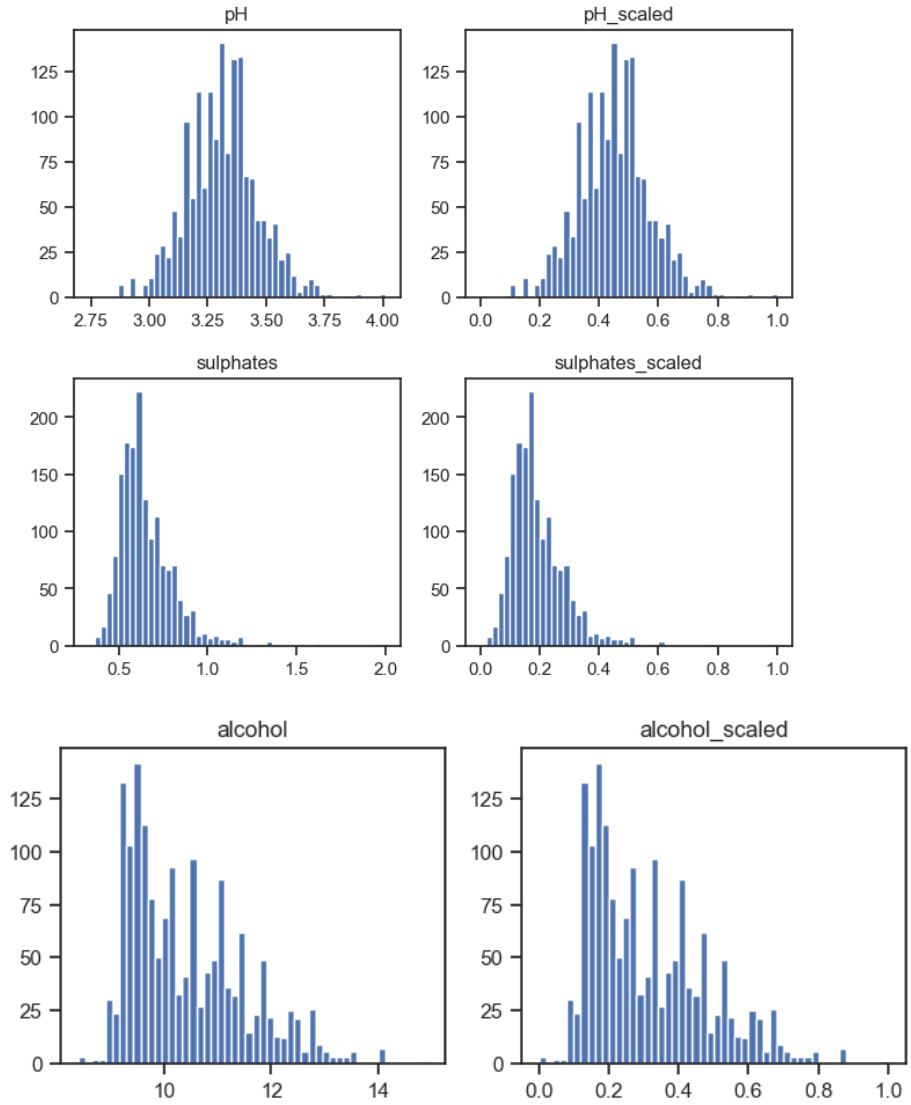









chlorides
chlorides_scaled
free sulfur dioxide
free sulfur dioxide_scaled
total sulfur dioxide
total sulfur dioxide_scaled
density
density_scaled



Все диаграммы имеют нормальный вид распределения.

4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

Создаем исходный список corr_cols_1.

```
# Создаем исходный список corr_cols_1
corr_cols_1 = scale_cols + ['quality']
corr_cols_1
```

```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

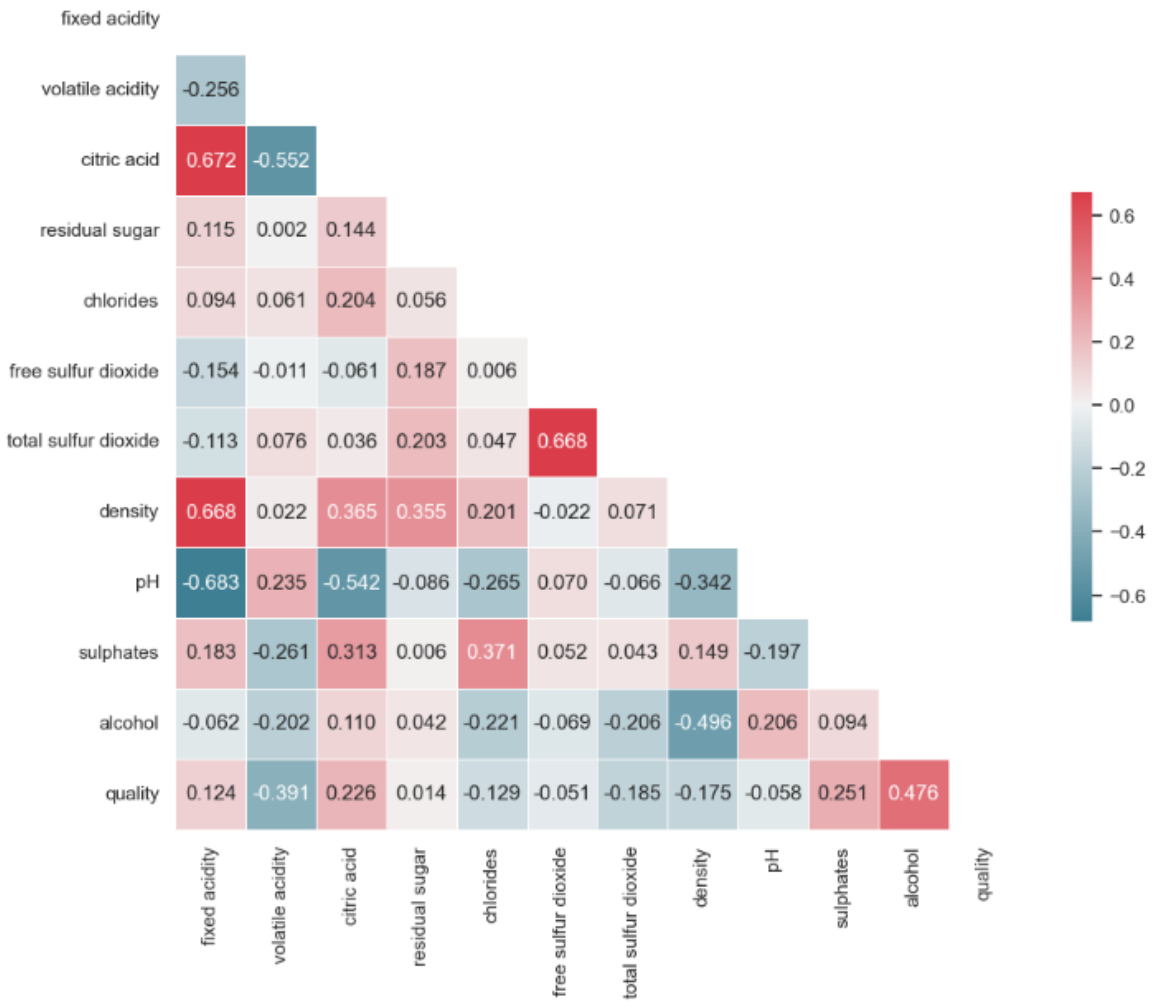
Создаем новый список corr_cols_2, включающий имена столбцов, масштабированных с добавлением суффикса "_scaled".

```
# Создаем новый список corr_cols_2, включающий имена столбцов, масштабированных с добавлением  суффикса "_scaled"
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['quality']
corr_cols_2
```

```
['fixed acidity_scaled',
 'volatile acidity_scaled',
 'citric acid_scaled',
 'residual sugar_scaled',
 'chlorides_scaled',
 'free sulfur dioxide_scaled',
 'total sulfur dioxide_scaled',
 'density_scaled',
 'pH_scaled',
 'sulphates_scaled',
 'alcohol_scaled',
 'quality']
```

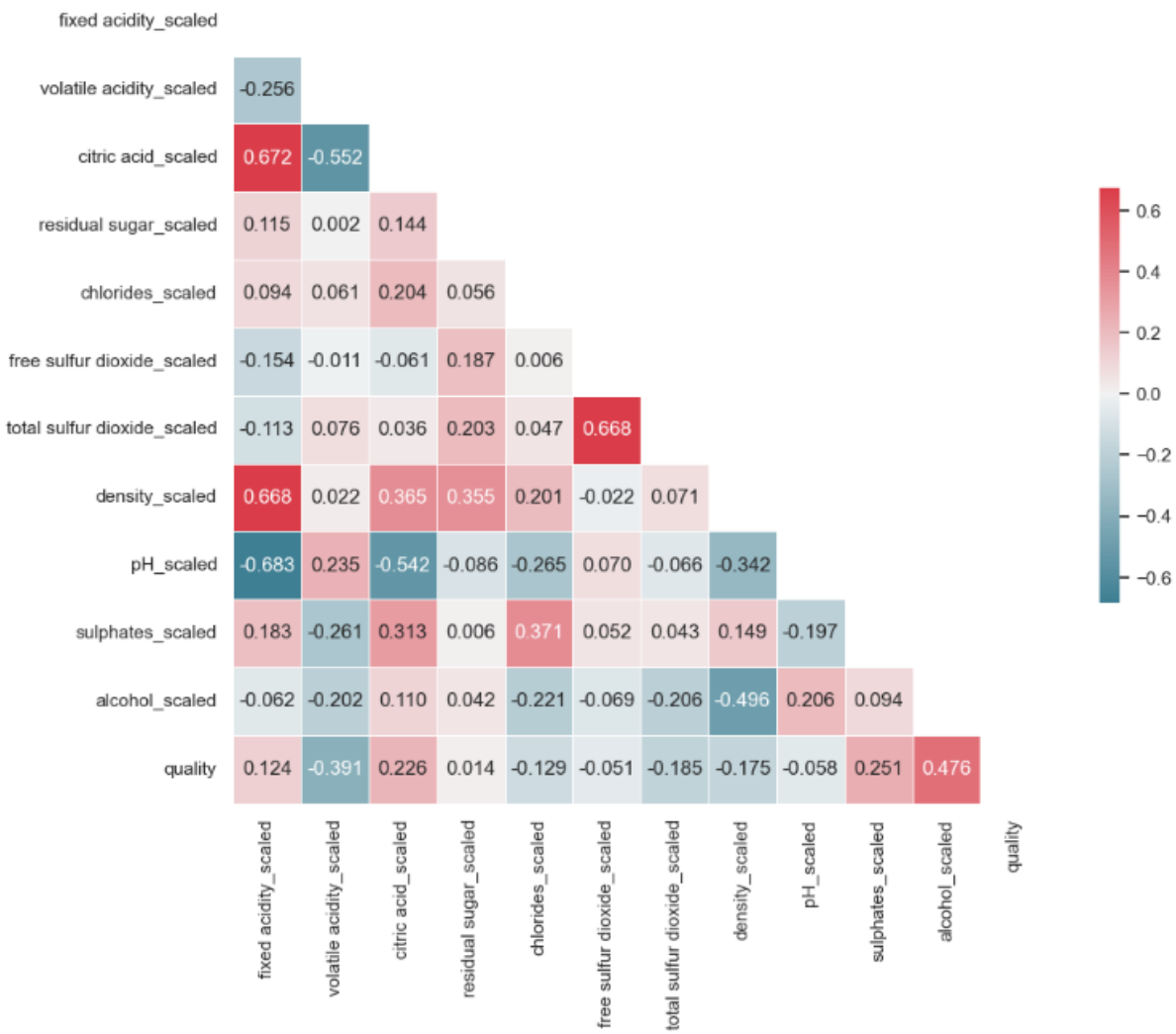
Создаем корреляционную матрицу для corr_cols_1.

```
# Создаем корреляционную матрицу для corr_cols_1
sns.set(style="white")
corr = data[corr_cols_1].corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f',
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



Создаем корреляционную матрицу для corr_cols_2.

```
# Создаем корреляционную матрицу для corr_cols_2
sns.set(style="white")
corr = data[corr_cols_2].corr()
mask = np.zeros_like(corr, dtype=bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f',
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



На основе двух корреляционных матриц для сравнения сделаем следующие выводы:

Корреляционные матрицы для исходных и масштабированных данных совпадают. Целевой признак классификации "quality" наиболее сильно коррелирует со следующими признаками:

- "alcohol" (0.476);
- "sulphates" (0.251);

- "citric acid" (0.226).

На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5. **Выбор метрик для последующей оценки качества моделей.**

В качестве метрик для решения задачи классификации будем использовать:

- Precision - доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор определил как положительные. В контексте анализа качества вина это будет означать, сколько из предсказанных «хороших» вин действительно оказались хорошими.
- Recall - доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов. Для нашей задачи это означает, насколько успешно модель находит все хорошие вина в датасете.
- F1- мера - для объединения precision и recall в единую метрику, обеспечивая баланс между ними.
- ROC AUC основана на вычислении следующих характеристик:
    - True Positive Rate, откладывается по оси ординат. Совпадает с recall.
    - False Positive Rate, откладывается по оси абсцисс. Показывает, какую долю из объектов отрицательного класса алгоритм предсказал неверно.

Отрисовка ROC-кривой.

```python
def draw_roc_curve(y_true, y_score, pos_label=1):

    # Вычисление значений TPR и FPR
    fpr, tpr, thresholds = roc_curve(y_true, y_score, pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score)  # Расчет AUC

    # Настройка графика
    plt.figure(figsize=(8, 6))
    lw = 2
    plt.plot(fpr, tpr, color='darkturquoise',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')  # Линия случайного предсказания
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic')
    plt.legend(loc="lower right")
    plt.grid()
    plt.show()
```

Создаем класс MetricLogger. Это нужно для того, чтобы построить различные решения задач и оценивания качества моделей.

```python
# Создаем класс MetricLogger
class MetricLogger:

    def __init__(self):
        # Инициализация пустого DataFrame с тремя столбцами: 'metric', 'alg', 'value'
        self.df = pd.DataFrame({
            'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')
        })

    def add(self, metric, alg, value):

        #Добавление значения метрики.

        # Создание нового DataFrame из входящих данных
        temp_df = pd.DataFrame({'metric': [metric], 'alg': [alg], 'value': [value]})
        # Конкатенация нового DataFrame с существующим
        self.df = pd.concat([self.df, temp_df], ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):

        # Формирование данных с фильтром по метрике.

        # Получение данных для указанной метрики и сортировка по значению метрики
        filtered_data = self.df[self.df['metric'] == metric]
        sorted_data = filtered_data.sort_values(by='value', ascending=ascending)
        return sorted_data['alg'].values, sorted_data['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):

        # Вывод графика значений метрики.

        # Получение данных для указанной метрики
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)

        # Создание горизонтальной столбчатой диаграммы
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric, align='center', height=0.5, tick_label=array_labels)

        # Настройка заголовка
        ax1.set_title(str_header)

        # Добавление подписей на столбцы
        for a, b in zip(pos, array_metric):
            ax1.text(b, a, f'{b:.3f}', color='black', va='center')

        plt.xlabel('Value')  # Добавляем подпись оси X
        plt.ylabel('Algorithms')  # Добавляем подпись оси Y
        plt.show()
```

6. **Выбор наиболее подходящих моделей для решения задачи классификации.**

Для задачи классификации будем использовать следующие модели:

- Метод ближайших соседей
- Линейная/логистическая регрессия
- Метод опорных векторов
- Дерево решений
- Случайный лес
- Градиентный бустинг

7. **Формирование обучающей и тестовой выборок на основе исходного набора данных.**

Признаки для задачи классификации.

```
# Признаки для задачи классификации вина
class_cols = [
    'quality',              # Качество вина (обычно целочисленная метка)
    'fixed acidity',        # Фиксированная кислотность
    'volatile acidity',     # Летучая кислотность
    'citric acid',          # Цитрусовая кислота
    'residual sugar',       # Остаточный сахар
    'chlorides',            # Хлориды
    'free sulfur dioxide', # Свободный диоксид серы
    'total sulfur dioxide', # Общий диоксид серы
    'density',              # Плотность
    'pH',                   # Уровень pH
    'sulphates',            # Сульфаты
    'alcohol'               # Содержание алкоголя
]
```

```
X = data[class_cols]
Y = data['quality']
print(f"Количество строк (наблюдений) в массиве X: {X.shape[0]}")
print(f"Количество столбцов (признаков) в массиве X: {X.shape[1]}")
```

```
Количество строк (наблюдений) в массиве X: 1599
Количество столбцов (признаков) в массиве X: 12
```

Разделим выборку на обучающую и тестовую.

```
# С использованием метода train_test_split разделим выборку на обучающую и тестовую
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=1)
```

```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

```
((1439, 12), (160, 12), (1439,), (160,))
```

Из итогов ((1439, 12), (160, 12), (1439,), (160,)) можно сделать следующие выводы:

- X_train содержит 1439 наблюдений и 12 признаков.
- X_test содержит 160 наблюдений и 12 признаков.

- Y_train содержит 1439 меток классов.
- Y_test содержит 160 меток классов.

Это означает, что данные были успешно разделены на обучающую и тестовую выборки. Обучающая выборка используется для обучения модели, а тестовая выборка - для оценки ее качества на данных, которые модель ранее не видела.

8. **Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.**

Модели.

```
# Определение моделей
clas_models = {
    'LogR': LogisticRegression(),
    'KNN_5': KNeighborsClassifier(n_neighbors=5),
    'SVC': SVC(probability=True),
    'Tree': DecisionTreeClassifier(),
    'RF': RandomForestClassifier(),
    'GB': GradientBoostingClassifier()
}
```

Сохранение метрик.

```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

Обучающая модель.

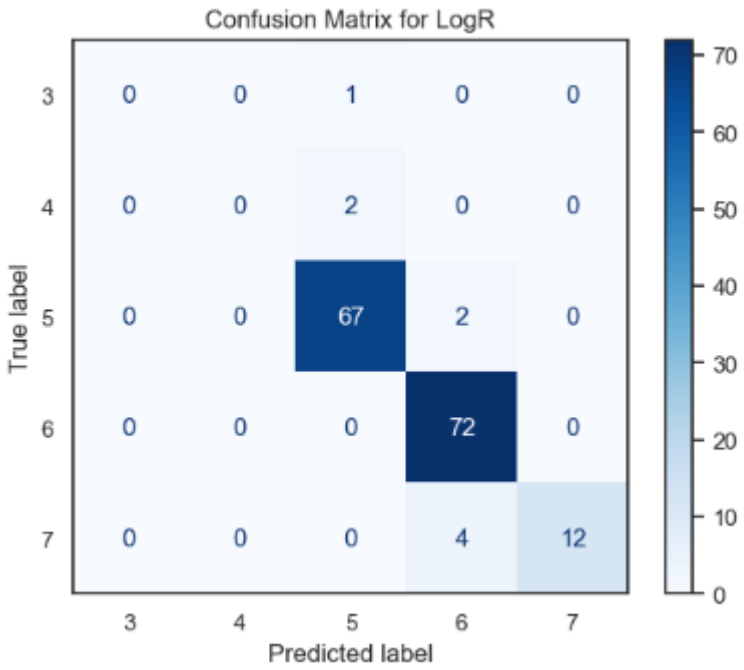
```python
# Обучающая модель
def train_model(model_name, model, MetricLogger):
    # Обучение модели
    model.fit(X_train, Y_train)

    # Получение прогнозов модели на тестовых данных
    Y_pred = model.predict(X_test)

    # Вычисление метрик качества модели
    precision = precision_score(Y_test, Y_pred, average='macro')
    recall = recall_score(Y_test, Y_pred, average='macro')
    f1 = f1_score(Y_test, Y_pred, average='macro')

    try:
        roc_auc = roc_auc_score(Y_test, model.predict_proba(X_test), multi_class='ovr')
    except ValueError:
        roc_auc = None  # Если невозможно посчитать ROC AUC

    # Добавление метрик качества в MetricLogger
    MetricLogger.add('precision', model_name, precision)
    MetricLogger.add('recall', model_name, recall)
    MetricLogger.add('f1', model_name, f1)
    if roc_auc is not None:
        MetricLogger.add('roc_auc', model_name, roc_auc)

    print(model)

    # Построение ROC-кривой (только если есть вероятности)
    if roc_auc is not None:
        draw_roc_curve(Y_test, model.predict_proba(X_test))

    # Получение и отображение матрицы ошибок
    cm = confusion_matrix(Y_test, Y_pred)

    unique_labels = np.unique(Y_test)

    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=unique_labels)
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix for {model_name}')
    plt.show()
```

Выполним обучение каждой модели из словаря clas_models, вычисляем и сохраняем метрики качества и результаты оценки производительности моделей.

```python
# Запуск обучения для каждой из моделей
for model_name, model in clas_models.items():
    train_model(model_name, model, clasMetricLogger)
```

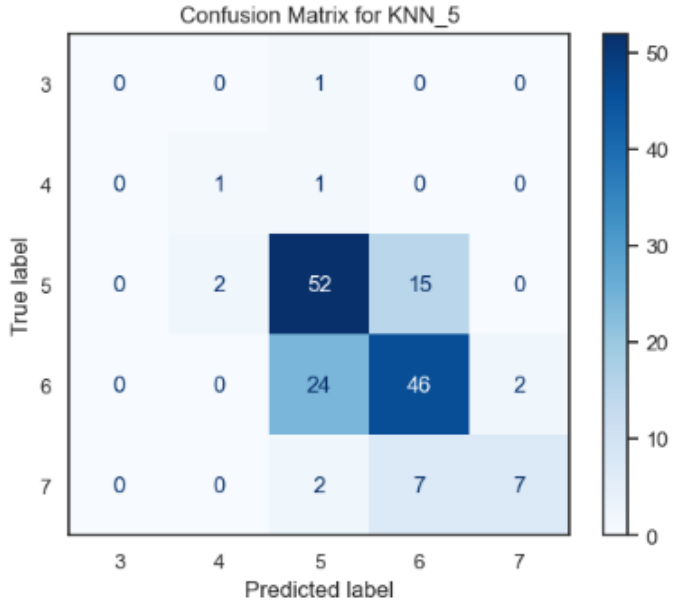
Метод ближайших соседей.

```
LogisticRegression()
```



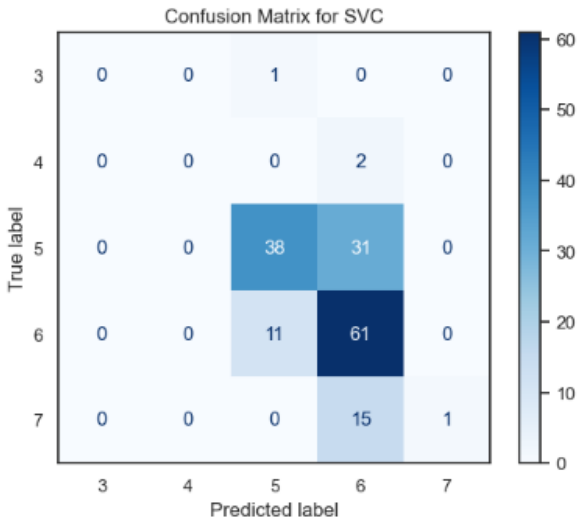
Логическая регрессия.

```
KNeighborsClassifier()
```



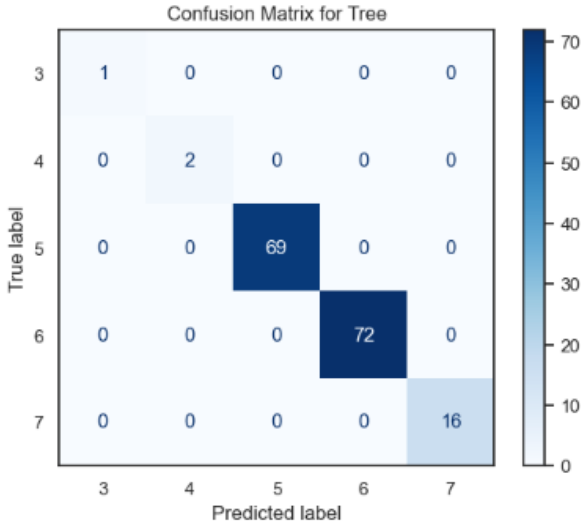
Метод опорных векторов.

SVC(probability=True)



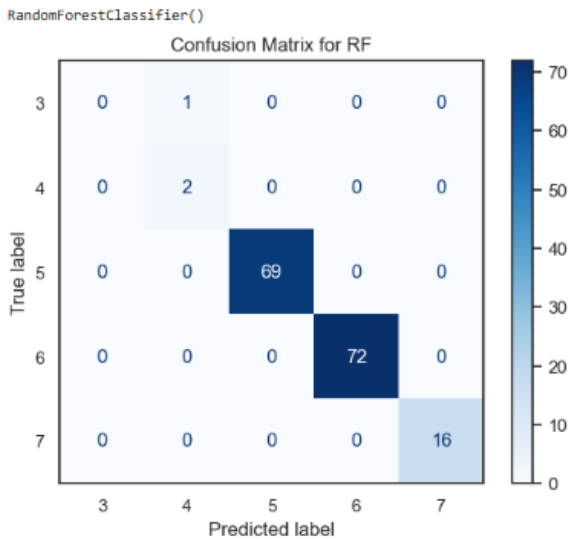
Дерево решений.

DecisionTreeClassifier()



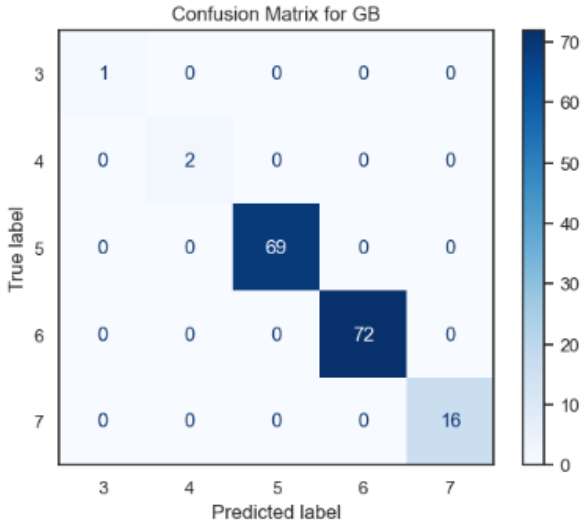
Случайный лес.

RandomForestClassifier()



Градиентный бустинг.

```
GradientBoostingClassifier()
```



9. Подбор гиперпараметров для выбранных моделей.

Метод ближайших соседей.

```
# Определение диапазона значений для n_neighbors
n_range = np.array(range(1, 100, 1))

# Настройка параметров для GridSearchCV
tuned_parameters = [{'n_neighbors': n_range}]
```

```
# Инициализация и запуск GridSearchCV
gs_KNN = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='roc_auc')
gs_KNN.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}],
             scoring='roc_auc')
```

GridSearchCV

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}],
             scoring='roc_auc')
```

best_estimator_: KNeighborsClassifier

```
KNeighborsClassifier(n_neighbors=1)
```

KNeighborsClassifier

Лучшая модель.

```
# Лучшая модель
best_model = gs_KNN.best_estimator_
print("Лучшая модель:", best_model)
```

```
Лучшая модель: KNeighborsClassifier(n_neighbors=1)
```

Лучшее значение параметров.

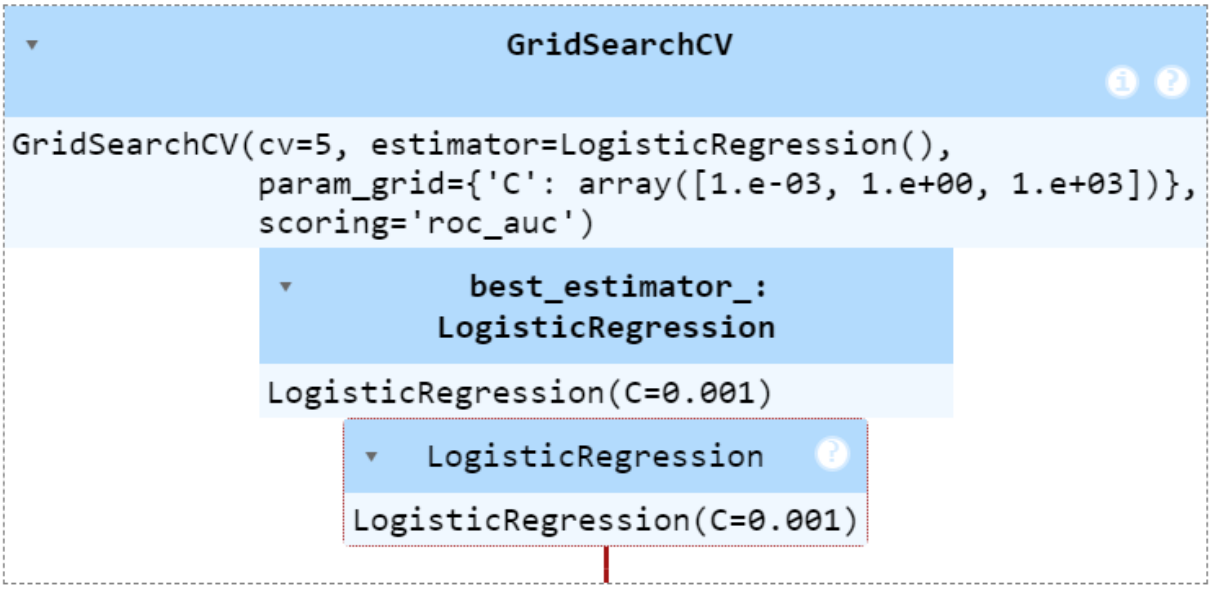
```
# Лучшее значение параметров
best_params = gs_KNN.best_params_
print("Лучшие параметры:", best_params)
```

```
Лучшие параметры: {'n_neighbors': 1}
```

Логистическая регрессия.

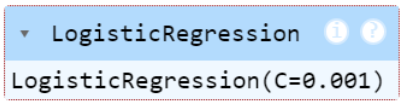
```
grid={"C":np.logspace(-3,3,3)}
gs_LogR = GridSearchCV(LogisticRegression(), grid, cv=5, scoring='roc_auc')
gs_LogR.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e+00, 1.e+03])},
             scoring='roc_auc')
```



Лучшая модель.

```
# Лучшая модель
gs_LogR.best_estimator_
```



Лучшее значение параметров.

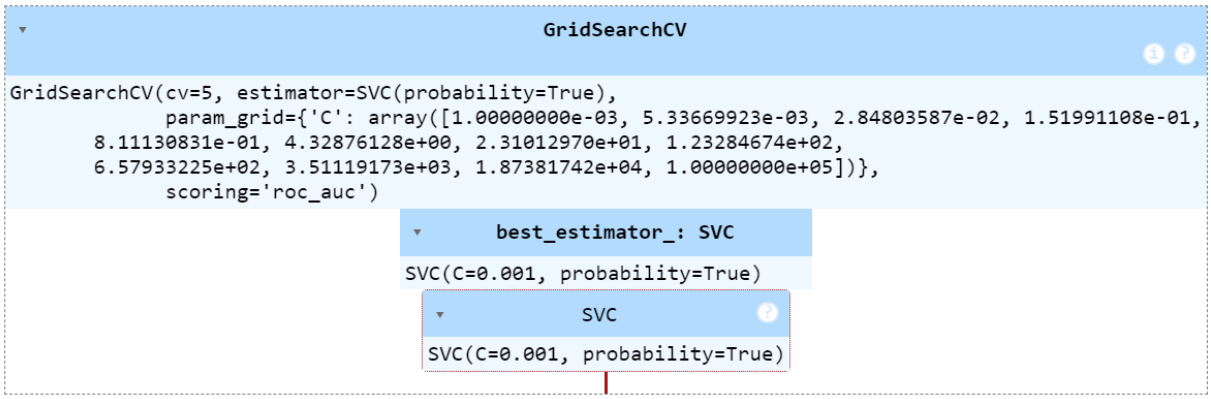
```
# Лучшее значение параметров
gs_LogR.best_params_
```

```
{'C': 0.001}
```

Метод опорных векторов.

```
SVC_grid={"C":np.logspace(-3,5,12)}
gs_SVC = GridSearchCV(SVC(probability=True), SVC_grid, cv=5, scoring='roc_auc')
gs_SVC.fit(X_train, Y_train)
```

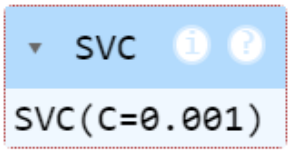
```
GridSearchCV(cv=5, estimator=SVC(probability=True),
             param_grid={'C': array([1.00000000e-03, 5.33669923e-03, 2.84803587e-02, 1.51991108e-01,
       8.11130831e-01, 4.32876128e+00, 2.31012970e+01, 1.23284674e+02,
       6.57933225e+02, 3.51119173e+03, 1.87381742e+04, 1.00000000e+05])},
             scoring='roc_auc')
```



Лучшая модель.

```
# Лучшая модель
gs_SVC.best_estimator_
```

```
SVC(C=0.001)
```



Лучшее значение параметров.

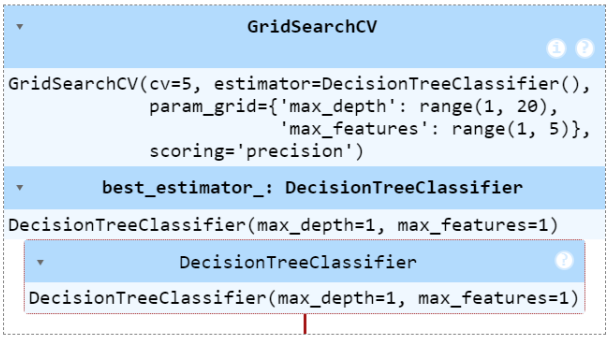
```
# Лучшее значение параметров
gs_SVC.best_params_
```

```
{'C': 0.001}
```

Дерево решений.

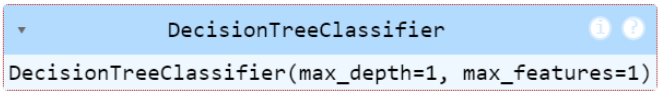
```
tree_params={"max_depth":range(1,20), "max_features":range(1,5)}
gs_Tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=5, scoring='precision')
gs_Tree.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(1, 20),
                         'max_features': range(1, 5)},
             scoring='precision')
```



Лучшая модель.

```
# Лучшая модель
gs_Tree.best_estimator_
```



Лучшее значение параметров.

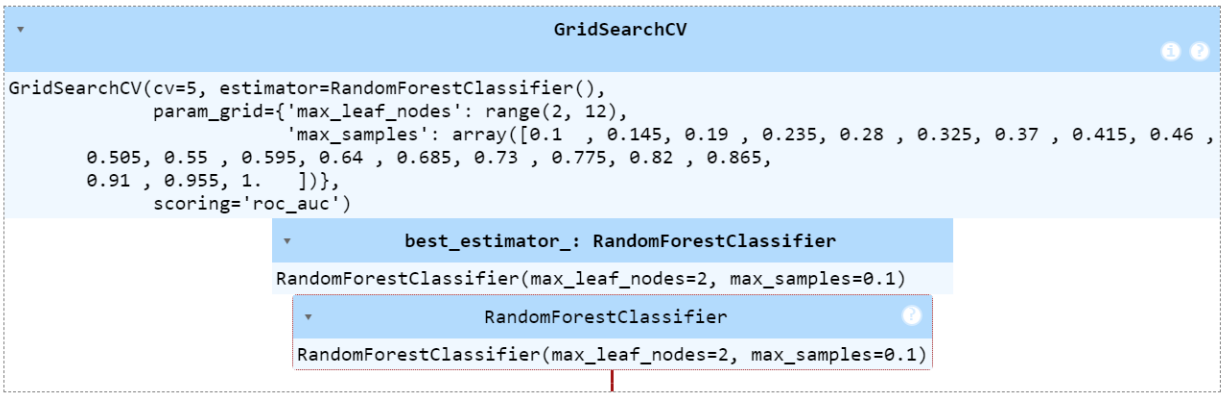
```
# Лучшее значение параметров
gs_Tree.best_params_
```

```
{'max_depth': 1, 'max_features': 1}
```

Случайный лес.

```
# Определение диапазонов параметров для Random Forest
RF_params = {
    "max_leaf_nodes": range(2, 12),  # Максимальное число листьев
    "max_samples": np.linspace(0.1, 1.0, num=21)  # Пробуем от 10% до 100% выборки
}
```

```
# Инициализация и запуск GridSearchCV
gs_RF = GridSearchCV(RandomForestClassifier(), RF_params, cv=5, scoring='roc_auc')
gs_RF.fit(X_train, Y_train)
```



Лучшая модель.

```
# Лучшая модель
best_model_rf = gs_RF.best_estimator_
print("Лучшая модель:", best_model_rf)
```

```
Лучшая модель: RandomForestClassifier(max_leaf_nodes=2, max_samples=0.1)
```

Лучшее значение параметров.

```
# Лучшие параметры
best_params_rf = gs_RF.best_params_
print("Лучшие параметры:", best_params_rf)
```

```
Лучшие параметры: {'max_leaf_nodes': 2, 'max_samples': 0.1}
```

Градиентный бустинг.

```
GB_params={"max_features":range(1,4), "max_leaf_nodes":range(2,22)}
gs_GB = GridSearchCV(GradientBoostingClassifier(), GB_params, cv=5, scoring='f1')
gs_GB.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'max_features': range(1, 4),
                         'max_leaf_nodes': range(2, 22)},
             scoring='f1')
best_estimator_: GradientBoostingClassifier
GradientBoostingClassifier(max_features=1, max_leaf_nodes=2)
GradientBoostingClassifier
GradientBoostingClassifier(max_features=1, max_leaf_nodes=2)
```

Лучшая модель.

```
# Лучшая модель
gs_GB.best_estimator_
```

```
GradientBoostingClassifier
GradientBoostingClassifier(max_features=1, max_leaf_nodes=2)
```

Лучшее значение параметров.

```
# Лучшее значение параметров
gs_GB.best_params_
```

```
{'max_features': 1, 'max_leaf_nodes': 2}
```

10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

Предположительно, LogR_new, KNN_new, SVC_new, Tree_new, RF_new и GB_new - это объекты, полученные в результате перекрестной проверки (GridSearchCV), примененной к моделям логистической регрессии (Logistic Regression), метода k ближайших соседей (K-Nearest Neighbors), метода опорных векторов (Support Vector Classifier), дерева решений (Decision Tree), случайного леса (Random Forest) и градиентного бустинга (Gradient Boosting),

соответственно.

```
models_grid = { 'LogR_new':gs_LogR.best_estimator_,
                'KNN_new':gs_KNN.best_estimator_,
                'SVC_new':gs_SVC.best_estimator_,
                'Tree_new':gs_Tree.best_estimator_,
                'RF_new':gs_RF.best_estimator_,
                'GB_new':gs_GB.best_estimator_
              }
```

Выполним обучение каждой модели из словаря models_grid, вычисляем и сохраняем метрики качества и результаты оценки производительности моделей.

```
for model_name, model in models_grid.items():
    train_model(model_name, model, clasMetricLogger)
```

11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

Метрики качества модели.

```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

```
array(['precision', 'recall', 'f1'], dtype=object)
```

```
for metric in clas_metrics:
    array_labels, array_metric = clasMetricLogger.get_data_for_metric(metric)
    print('Метрика:', metric)
    for label, value in zip(array_labels, array_metric):
        print(f'{label} : {value:.2f}')
    print('\n')
```

```
Метрика: precision
LogR_new : 0.29
LogR_new : 0.29
LogR_new : 0.29
LogR_new : 0.29
LogR_new : 0.29
KNN_new : 0.43
KNN_new : 0.43
KNN_new : 0.43
KNN_new : 0.43
KNN_new : 0.43
SVC : 0.46
KNN_5 : 0.49
KNN_5 : 0.49
KNN_5 : 0.49
KNN_5 : 0.49
KNN_5 : 0.49
LogR : 0.58
LogR : 0.58
LogR : 0.58
LogR : 0.58
LogR : 0.58
LogR : 0.58
LogR : 0.58
RF : 0.73
Tree : 1.00
GB : 1.00
```

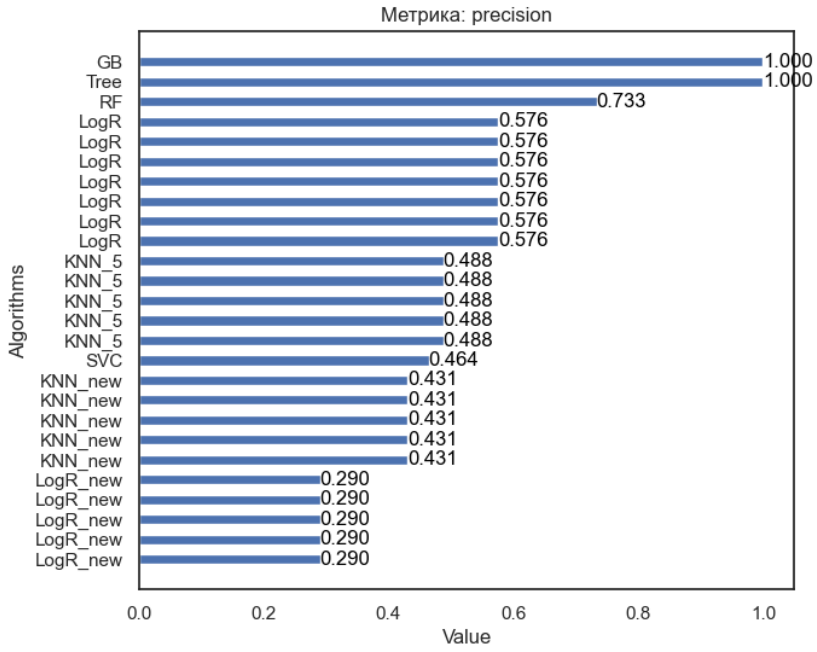
```
Метрика: recall
SVC : 0.29
LogR_new : 0.33
LogR_new : 0.33
LogR_new : 0.33
LogR_new : 0.33
LogR_new : 0.33
KNN_new : 0.42
KNN_new : 0.42
KNN_new : 0.42
KNN_new : 0.42
KNN_new : 0.42
KNN_5 : 0.47
KNN_5 : 0.47
KNN_5 : 0.47
KNN_5 : 0.47
KNN_5 : 0.47
LogR : 0.54
LogR : 0.54
LogR : 0.54
LogR : 0.54
LogR : 0.54
LogR : 0.54
LogR : 0.54
RF : 0.80
Tree : 1.00
GB : 1.00
```

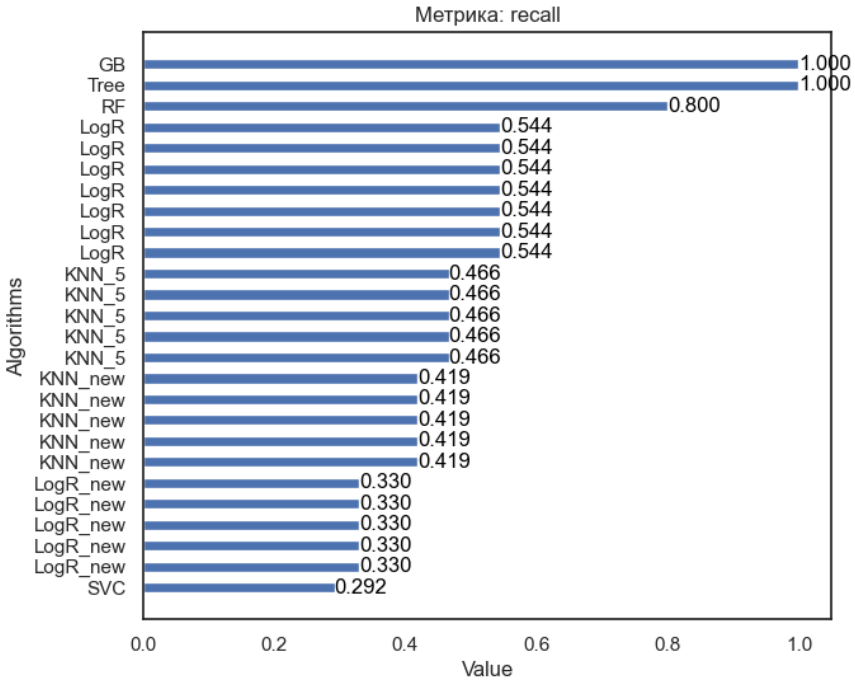
```
Метрика: f1
SVC : 0.29
LogR_new : 0.31
LogR_new : 0.31
LogR_new : 0.31
LogR_new : 0.31
LogR_new : 0.31
KNN_new : 0.42
KNN_new : 0.42
KNN_new : 0.42
KNN_new : 0.42
KNN_new : 0.42
KNN_5 : 0.46
KNN_5 : 0.46
KNN_5 : 0.46
KNN_5 : 0.46
KNN_5 : 0.46
LogR : 0.56
LogR : 0.56
LogR : 0.56
LogR : 0.56
LogR : 0.56
LogR : 0.56
LogR : 0.56
RF : 0.76
Tree : 1.00
GB : 1.00
```

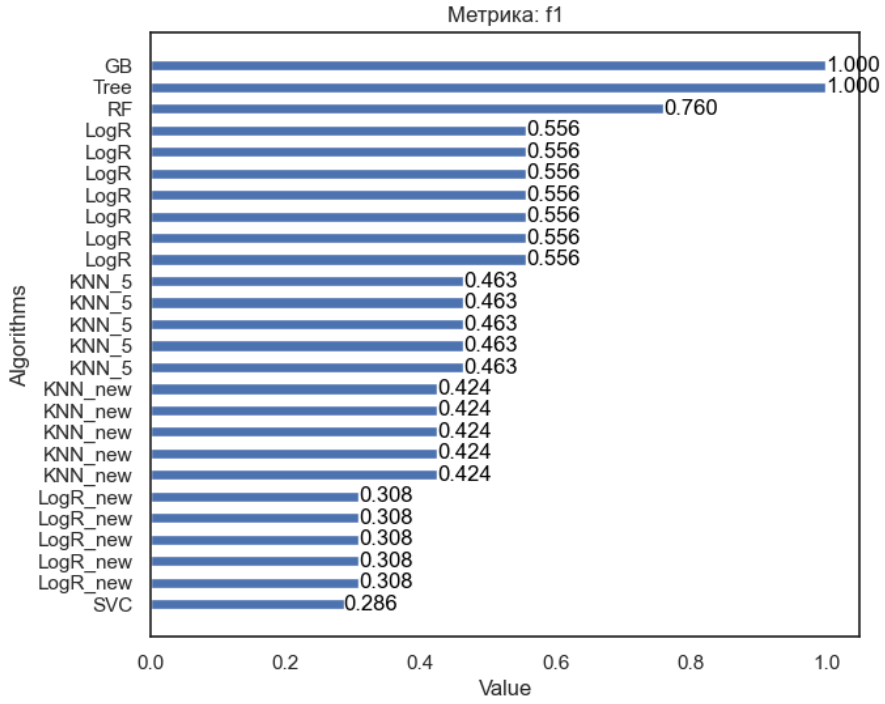
Для наглядности результатов моделей, построим диаграмму.

Построим графики метрик качества модели.

```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```







12.Вывод

Из предоставленных метрик качества моделей можно сделать следующие выводы:

- Точность (precision): Модели GB, Tree имеют самую высокую точность, равную 1. Модель RF также обладает высокой точностью, около 0.733. Модель LogR имеет точность около 0.57, а модель KNN_5 - около 0.48. Модели SVC и KNN_new обладают точностью 0.4. Модель LogR_new обладает точностью 0.29.
- Полнота (recall): Модели GB, Tree имеют самую высокую точность, равную 1. Модель RF также обладает высокой точностью, около 0.8. Модель LogR имеет точность около 0.54, а модель KNN_5 - около 0.46. Модель KNN_new обладает точностью 0.41. Модель LogR_new обладает точностью 0.33. Модель SVC обладает точностью 0.29.
- F1-мера (f1): Модели GB, Tree имеют самую высокую точность, равную 1. Модель RF также обладает высокой точностью, около 0.76. Модель LogR имеет точность около 0.55, а модель KNN_5 - около 0.46. Модель

KNN_new обладает точностью 0.42. Модель LogR_new обладает точностью 0.33. Модель SVC обладает точностью 0.286.

Таким образом, можно сделать вывод, что модели, обученные с использованием логистической регрессии и случайного леса, показывают наилучшее общее качество, выраженное высокими значениями метрик точности, полноты, F1-меры.

Заключение

В ходе научно-исследовательской работы был проведен разведочный анализ данных набора "Биомеханические особенности пациентов-ортопедов". Мы использовали различные инструменты анализа данных, такие как Pandas, Matplotlib, Seaborn, Numpy и Streamlit на платформе Python с помощью программы Jupyter Notebook.

В процессе работы мы провели анализ структуры данных, заполнили пропущенные значения, выбрали признаки для построения моделей, закодировали категориальные признаки, выполняли масштабирование данных и создали дополнительные признаки для улучшения качества моделей.

Мы также провели корреляционный анализ данных и сформулировали промежуточные выводы о возможности построения моделей машинного обучения.

Для оценки качества моделей мы выбрали метрики, такие как точность, полнота, F1-мера, и рассмотрели более пяти различных моделей, включая как базовые, так и ансамблевые методы.

Мы сформировали обучающую и тестовую выборки, построили базовые решения для выбранных моделей и подобрали оптимальные значения гиперпараметров с использованием методов кросс-валидации.

В итоге мы оценили качество построенных моделей и сравнили их с базовыми моделями, сделав выводы о наилучших подходах к решению задачи классификации или регрессии на основе наших выбранных метрик.

Список использованных источников информации

1) Абдрахманов, М. И. Devpractice Team. Pandas. Работа с данными. / М. И. Абдрахманов — 2-е изд. — devpractice.ru. 2020. - 170 с.: [Электронный ресурс]. // URL: https://coderbooks.ru/books/python/pandas_rabota_s_dannymi_abdrahmanov_2020/ (дата обращения: 15.03.2024)
2) Абдрахманов, М. И. Devpractice Team. Python. Визуализация данных. Matplotlib. Seaborn. Mayavi. — devpractice.ru. 2020. - 412 с.: [Электронный ресурс]. // URL: https://coderbooks.ru/books/python/python_vizualizaciya_dannyh_abdrahmanov_2020/ (дата обращения: 21.03.2024)
3) Методические указания по программному обеспечению «Pandas»: [Электронный ресурс]. // URL: https://pandas.pydata.org/ (дата обращения: 20.02.2024)
4) Методические указания по программному обеспечению «Seaborn»: [Электронный ресурс]. // URL: https://seaborn.pydata.org/ (дата обращения: 22.02.2024)
5) Метрики качества. Метрики классификации. Подсчет количества ошибок, доли правильных ответов, точности, полноты. [Электронный ресурс]. // URL: https://data-scientists.ru/metriki-kachestva-metriki-klassifikacii-podschet-kolichestva-oshibok-doli-pravilnyh-otvetov (дата обращения: 01.03.2024)
6) Регрессионные модели в Python: [Электронный ресурс]. // URL: https://nagornyy.me/it/regressionnye-modeli-v-python/ (дата обращения: 11.03.2024)
7) Учебник по машинному обучению: [Электронный ресурс]. // URL: https://education.yandex.ru/handbook/ml (дата обращения: 01.03.2024)