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)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Модели машинного обучения

Студент РТ5-61Б _____ А. С. Агеев
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы _____ Ю. Е. Гапанюк
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2023 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 _____
(Индекс)

_____ _____
(И.О.Фамилия)

« _____ » _____ 20 ____ г.

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

по теме Модели машинного обучения

Студент группы РТ5-61Б

Агеев Алексей Сергеевич
(Фамилия, имя, отчество)

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

учебная

Источник тематики (кафедра, предприятие, НИР) НИР

График выполнения НИР: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание Решение задачи машинного обучения на основе материалов дисциплины

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 32 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « 7 » февраля 2023 г.

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

Студент _____ А. С. Агеев
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Введение

В современном мире машинное обучение является одной из наиболее перспективных и актуальных технологий, которая находит свое применение в различных сферах деятельности, начиная от медицины и финансов и заканчивая производством и транспортом. Технологии машинного обучения позволяют компьютерам обучаться на основе большого количества данных и использовать полученные знания для решения сложных задач. В данной научно-исследовательской работе рассмотрены основные принципы и методы машинного обучения. Мы изучим различные алгоритмы обучения, задачи классификации. В результате выполнения данной работы получены необходимые знания и навыки для работы с технологиями машинного обучения, что позволяет успешно применять эти технологии в практической деятельности.

Основная часть

Цель научно-исследовательской работы – разработка эффективной модели машинного обучения для решения задачи классификации на выбранном наборе данных.

Последовательность действий:

1. 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. Масштабирование данных.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей.
5. Выбор метрик для последующей оценки качества моделей.
6. Выбор наиболее подходящих моделей для решения задачи классификации.
7. Формирование обучающей и тестовой выборок на основе исходных данных.
8. Построение базового решения 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 с помощью методов кросс-валидации.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.
12. Создать веб-приложение для демонстрации хотя бы одной модели машинного обучения. У пользователя должна быть возможность изменения хотя бы одного гиперпараметра модели, при изменении гиперпараметра модель должна перестраиваться в веб-интерфейсе.

Заключение

В результате проведенной научно-исследовательской работы была разработана эффективная модель машинного обучения для решения задачи классификации на выбранном наборе данных. В ходе работы были выполнены все поставленные задачи.

Полученные результаты позволяют сделать вывод о том, что построенные модели машинного обучения имеют высокое качество и могут быть использованы для решения задачи классификации на данном наборе данных. Веб-приложение для демонстрации модели машинного обучения позволяет пользователю изменять гиперпараметры модели и наблюдать за изменением ее качества в режиме реального времени.

Таким образом, научно-исследовательская работа по технологиям машинного обучения позволила успешно решить задачу классификации на выбранном наборе данных и создать веб-приложение для демонстрации модели машинного обучения. Полученные результаты могут быть использованы в различных областях, где требуется решение задач классификации на основе данных.

Список использованных источников информации

1. Бурков, В.Н. Методы машинного обучения в задачах классификации / В.Н. Бурков. - М.: ФИЗМАТЛИТ, 2017. - 352 с.
2. Шестаков, А.В. Технологии машинного обучения: учебное пособие / А.В. Шестаков. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2018. - 232 с.
3. Кузнецов, М.П. Машинное обучение и анализ данных: учебное пособие / М.П. Кузнецов, Е.В. Кузнецова. - М.: Изд-во МГУ, 2019. - 432 с.
4. Решетников, И.В. Методы машинного обучения и анализа данных: учебник для вузов / И.В. Решетников, В.К. Курганов, И.Б. Петров. - СПб.: БХВ-Петербург, 2018. - 480 с.
5. Карпов, О.В. Технологии машинного обучения: учебное пособие для студентов вузов / О.В. Карпов, М.В. Чернышев, А.В. Шестаков. - СПб.: Питер, 2019. - 288 с.

Приложение

Ход работы в Jupyter Notebook:

Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации.

Датасет состоит из трех файлов:

- datatraining.txt - выборка

Каждый файл содержит следующие колонки:

- Temperature - температура в Кельвинах.
- Color - Общий цвет спектра.
- L - коэфициент светимости (относительное солнца)
- R - коэфициент радиуса (относительное солнца)
- Spectral class - спектральный класс O,B,A,F,G,K,M.
- Type - целевой признак датасета. Диапазон 0-5.

В рассматриваемом примере будем решать обе задачи - и задачу классификации, и задачу регрессии:

- Для решения **задачи классификации** в качестве целевого признака будем использовать "Type".

Импорт библиотек

Импортируем библиотеки с помощью команды import. Как правило, все команды import размещают в первых ячейках ноутбука.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
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, classifica
tion_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squar
ed_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, LinearS
VR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_
graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import BaggingClassifier
```

```
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegres
sor
%matplotlib inline
sns.set(style="ticks")
```

Загрузка данных

```
# Обучающая выборка
original = pd.read_csv('Stars.csv', sep=",")
original.drop_duplicates()

original.head()
```

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	0
1	3042	0.000500	0.1542	16.60	Red	M	0
2	2600	0.000300	0.1020	18.70	Red	M	0
3	2800	0.000200	0.1600	16.65	Red	M	0
4	1939	0.000138	0.1030	20.06	Red	M	0

Основные характеристики датасетов

```
# Первые 5 строк датасета
original.head()
```

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	0
1	3042	0.000500	0.1542	16.60	Red	M	0
2	2600	0.000300	0.1020	18.70	Red	M	0
3	2800	0.000200	0.1600	16.65	Red	M	0
4	1939	0.000138	0.1030	20.06	Red	M	0

```
original.head()
```

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	Red	M	0
1	3042	0.000500	0.1542	16.60	Red	M	0
2	2600	0.000300	0.1020	18.70	Red	M	0
3	2800	0.000200	0.1600	16.65	Red	M	0
4	1939	0.000138	0.1030	20.06	Red	M	0

```
original.shape
```

```
(240, 7)
```

```
# Список колонок
original.columns
```

```
Index(['Temperature', 'L', 'R', 'A_M', 'Color', 'Spectral_Class', 'Type'], dtyp
e='object')
```

```
# Список колонок с типами данных
# убедимся что типы данных одинаковы в обучающей и тестовых выборках
original.dtypes
```

```
Temperature        int64
L                float64
R                float64
A_M              float64
Color             object
Spectral_Class    object
Type               int64
dtype: object
```

```
original.dtypes
```

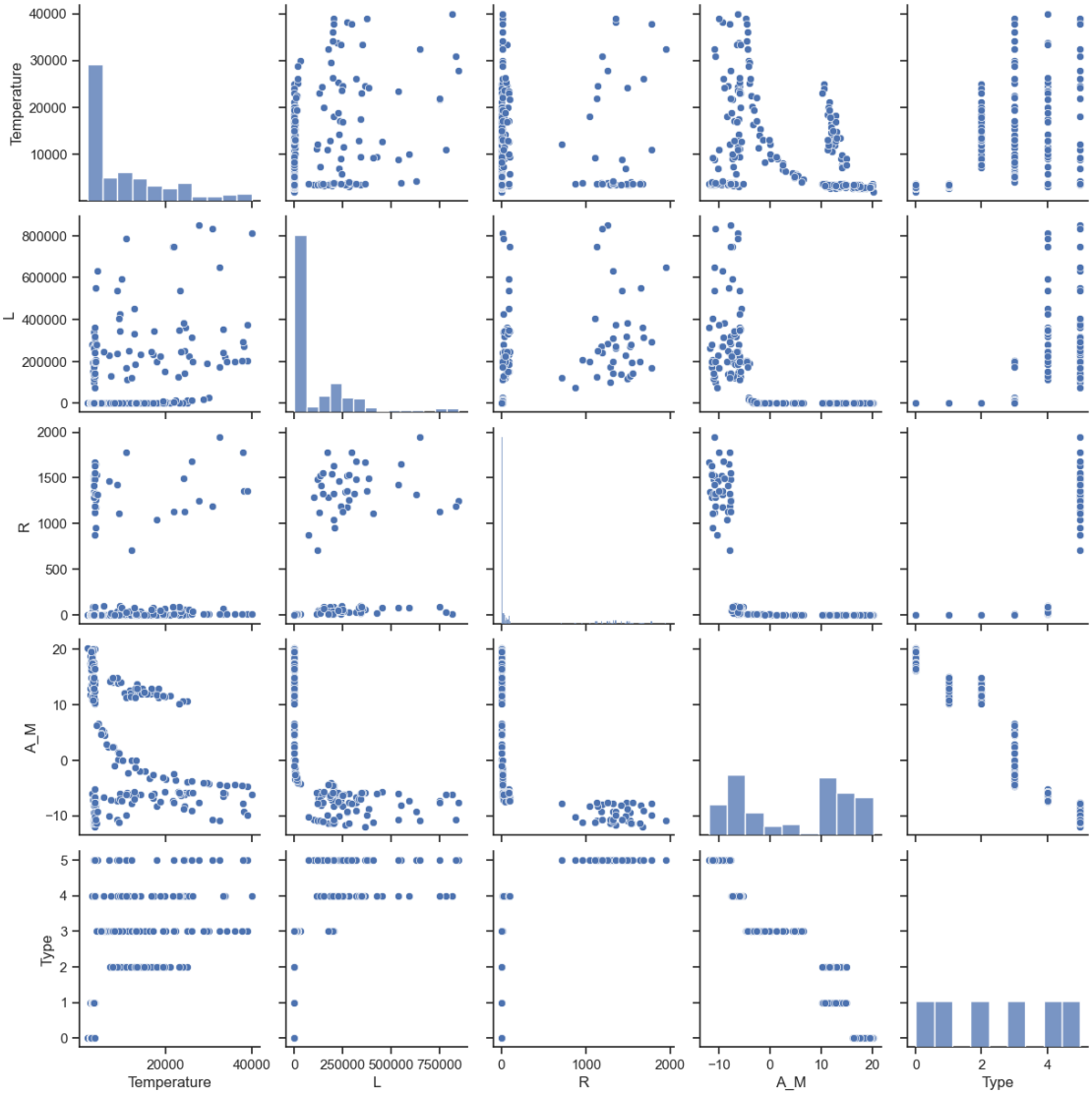
```
Temperature          int64
L                  float64
R                  float64
A_M                float64
Color               object
Spectral_Class      object
Type                 int64
dtype: object
```

```
# Проверим наличие пустых значений
original.isnull().sum()
```

```
Temperature       0
L                 0
R                 0
A_M               0
Color             0
Spectral_Class    0
Type              0
dtype: int64
```

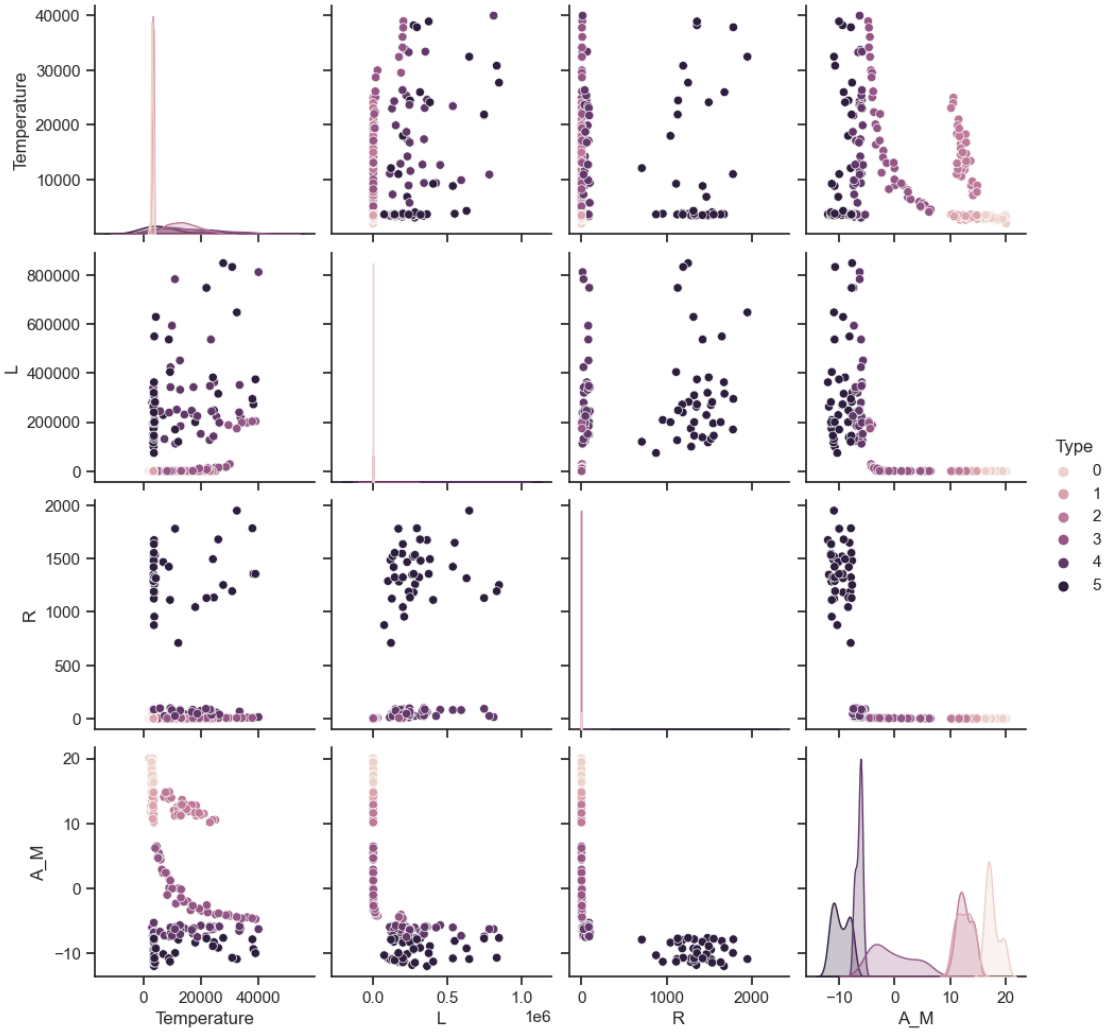
```
# Парные диаграммы
sns.pairplot(original)
```

```
<seaborn.axisgrid.PairGrid at 0x2943c3b08e0>
```



```
sns.pairplot(original, hue="Type")
```

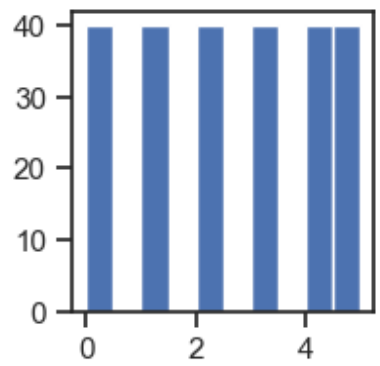
```
<seaborn.axisgrid.PairGrid at 0x29443c798a0>
```




```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
original['Type'].unique()
```

```
array([0, 1, 2, 3, 4, 5], dtype=int64)
```

```
# Оценим дисбаланс классов для Occupancy
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(original['Type'])
plt.show()
```



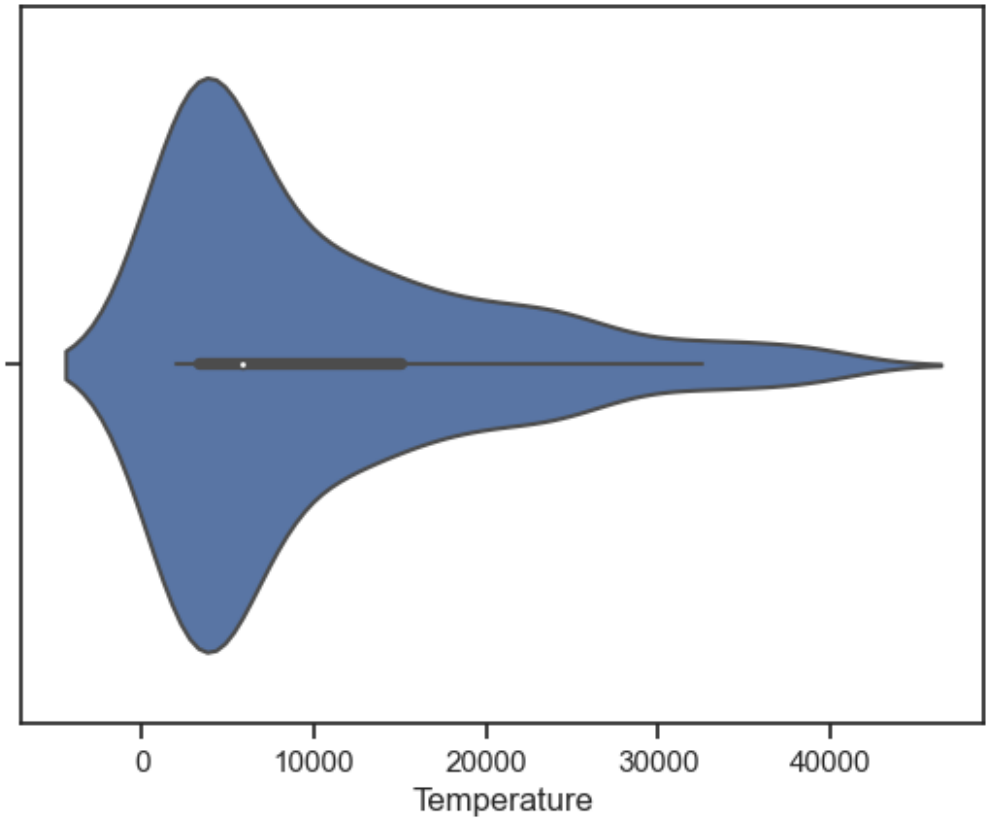
```
original['Type'].value_counts()
```

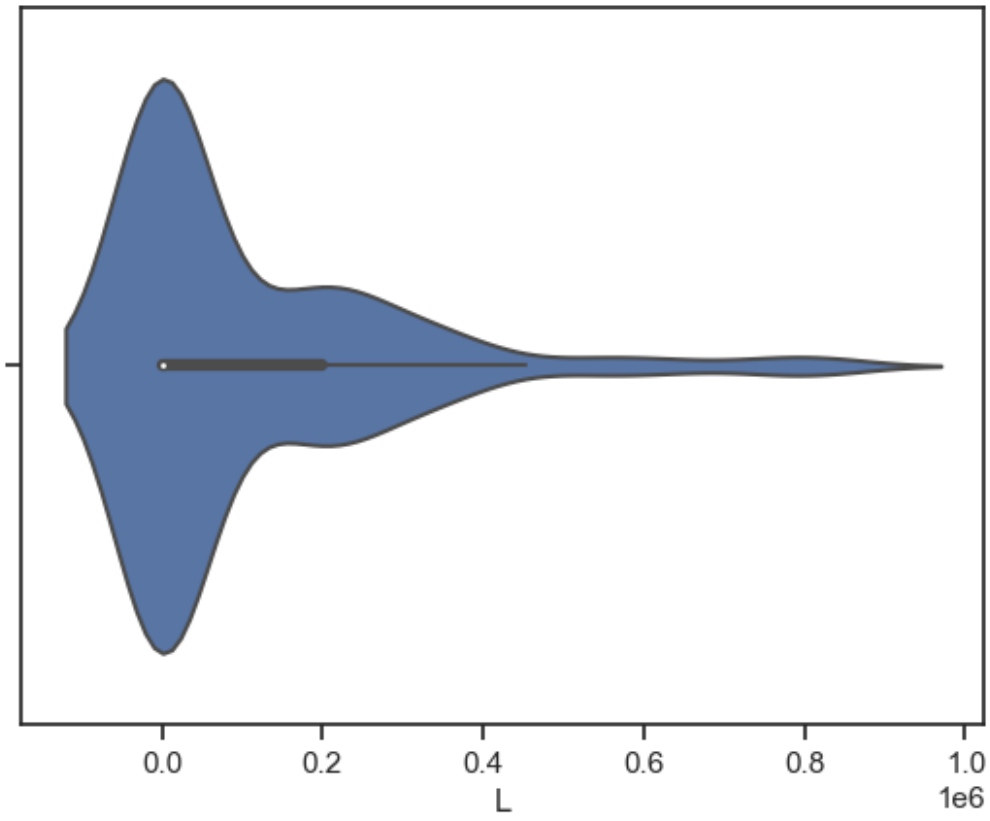
```
0    40
1    40
2    40
3    40
4    40
5    40
Name: Type, dtype: int64
```

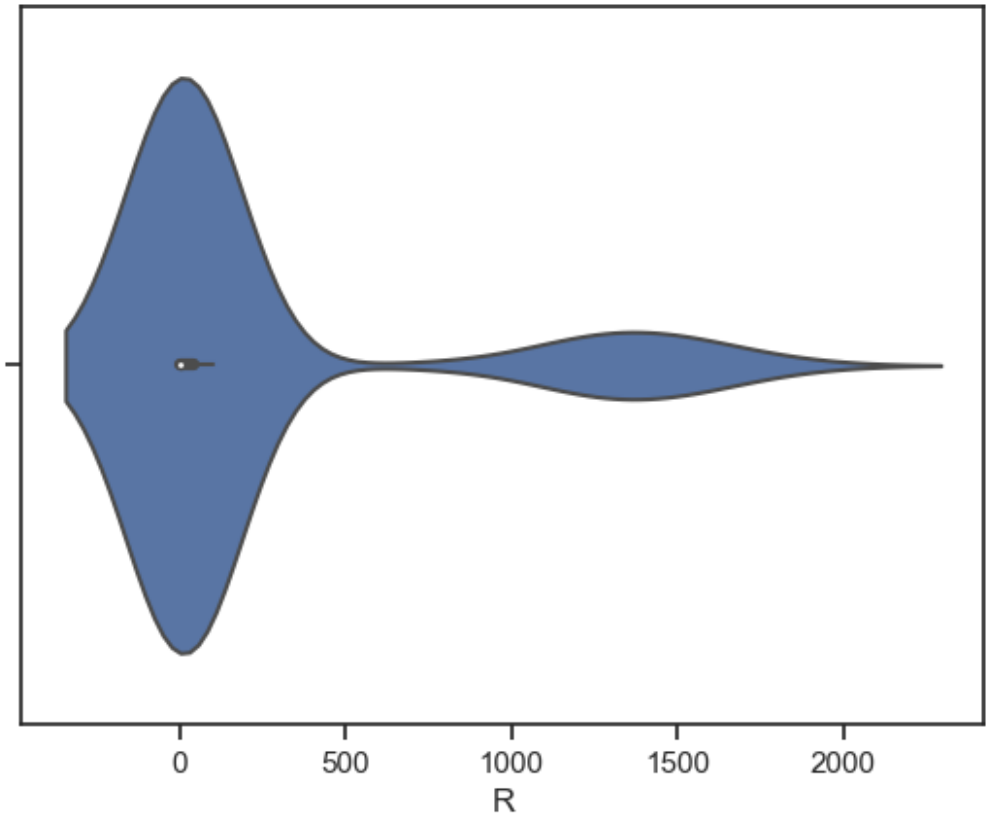
```
original.columns
```

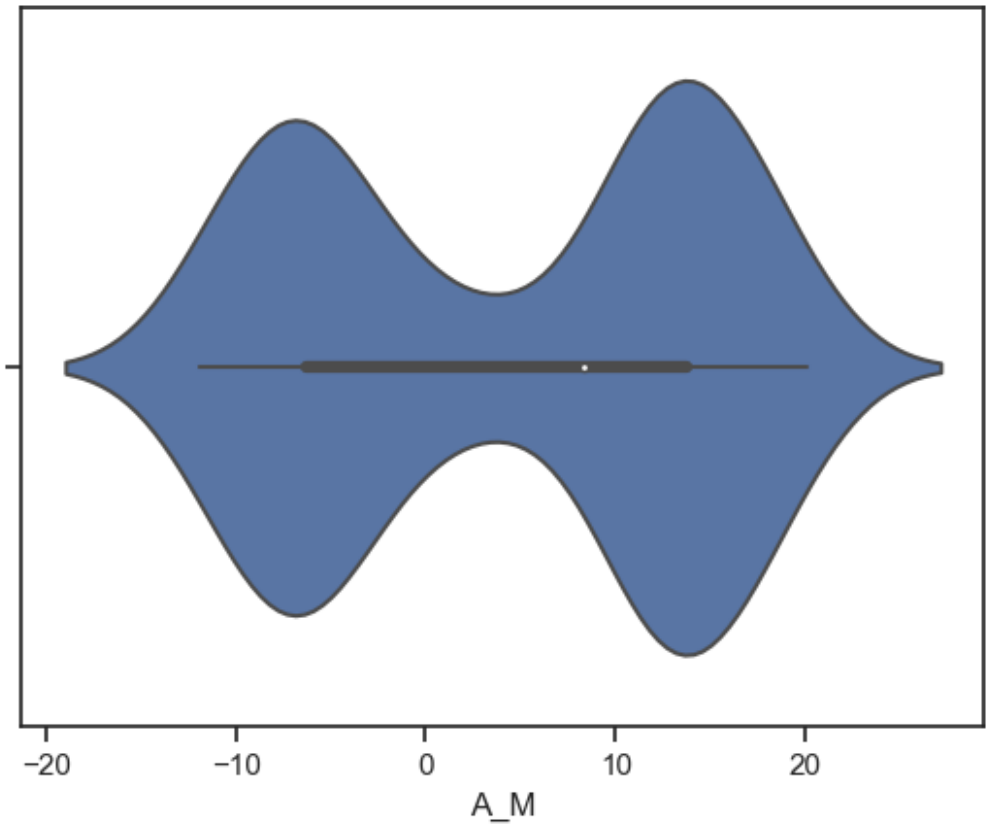
```
Index(['Temperature', 'L', 'R', 'A_M', 'Color', 'Spectral_Class', 'Type'], dtyp
e='object')
```

```
# Скрипичные диаграммы для числовых колонок
for col in ['Temperature', 'L', 'R', 'A_M']:
    sns.violinplot(x=original[col])
    plt.show()
```




0.0
0.2
0.4
0.6
0.8
1.0
L
1e6


0
500
1000
1500
2000
R



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

```
original.dtypes
```

```
Temperature         int64
L                 float64
R                 float64
A_M               float64
Color              object
Spectral_Class     object
Type                int64
dtype: object
```

Для построения моделей будем использовать все признаки.

Необхоимо закодировать категориальные признаки.

Вспомогательные признаки для улучшения качества моделей мы строить не будем.

Выполним масштабирование данных. Для этого необходимо объединить обучающую и тестовые выборки.

```
cat_enc = pd.DataFrame({'c1':original['Color'], 'c2':original['Spectral_Class']
})
cat_enc
```

```
        c1 c2
0      Red  M
1      Red  M
2      Red  M
3      Red  M
4      Red  M
..     ... ..
```

```
235   Blue  O
236   Blue  O
237  White  A
238  White  A
239   Blue  O

[240 rows x 2 columns]
```

```
from sklearn.preprocessing import LabelEncoder

cat_enc['c1'].unique()
```

```
array(['Red', 'Blue White', 'White', 'Yellowish White', 'Blue white',
       'Pale yellow orange', 'Blue', 'Blue-white', 'Whitish',
       'yellow-white', 'Orange', 'White-Yellow', 'white', 'yellowish',
       'Yellowish', 'Orange-Red', 'Blue-White'], dtype=object)
```

```
cat_enc['c2'].unique()
```

```
array(['M', 'B', 'A', 'F', 'O', 'K', 'G'], dtype=object)
```

```
le = LabelEncoder()
cat_enc_c1_le = le.fit_transform(cat_enc['c1'])
original['Color'] = cat_enc_c1_le
print(cat_enc_c1_le)
print(le.classes_)
```

```
[ 8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  1  9  9  9
  1 13  2 13 13  7  0  4  4 11 15 11 15 15 15 15  8  8  8  8  8  8  8  8
  8  0  8  8  8  5  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8
  8  8  8  8  8  8  8  8 10  9 14  1  1  1  1  1 14  0 15 16 16 12  4  4
  6 15  4  0  0  0  0  0  0  0  0  0  0  0  8  8  8  8  8  8  8  8  8  8
  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  1  0  0  1
  2  2  2 14  9  1  4  3  4  4  4  4  4  4  4  0  0  0  0  0  0  0  0  0
  0  0  8  8  5  0  8  8  0  4  4  4  8  8  8  8  8  8  8  8  8  8  8  8
  8  8  8  8  8  8  8  8  0  0  0  0  0  0  0  0  0  0  4  4  4  4  0  0
  4  4  4 15  0  0  0  0  0  0  0  0  0  0  4  0  0  4  4  0  0  9  9  0]
['Blue' 'Blue White' 'Blue white' 'Blue-White' 'Blue-white' 'Orange'
 'Orange-Red' 'Pale yellow orange' 'Red' 'White' 'White-Yellow' 'Whitish'
 'Yellowish' 'Yellowish White' 'white' 'yellow-white' 'yellowish']
```

```
cat_enc_c2_le = le.fit_transform(cat_enc['c2'])
original['Spectral_Class'] = cat_enc_c2_le
print(cat_enc_c2_le)
print(le.classes_)
```

```
[5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 0 2 0 1 2 0 2 2 2 6 1 1 1 2 0 2
 2 2 2 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 2 2 0 1 1 0 1 1 2 1 2 4 4 4 0 0 4 2 0 6 1 6 6 6 6 6 6 6 6 6 5
 5 5 5 5 5 4 5 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 1 1 1 1 0 1 2
 0 1 1 0 0 1 1 1 1 1 0 6 6 6 6 6 6 6 6 6 6 6 5 5 4 1 5 5 6 1 1 1 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 1 1 1 1 1 1 1 1 1 0 0 1 6 6 0 1 1 2 6 6
 6 6 6 6 6 6 6 6 1 6 6 1 1 6 6 0 0 6]
['A' 'B' 'F' 'G' 'K' 'M' 'O']
```

original

	Temperature	L	R	A_M	Color	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	8	5	0
1	3042	0.000500	0.1542	16.60	8	5	0
2	2600	0.000300	0.1020	18.70	8	5	0
3	2800	0.000200	0.1600	16.65	8	5	0

```
4           1939       0.000138     0.1030  20.06      8               5     0
..           ...            ...        ...    ...    ...             ...   ...
235        38940  374830.000000  1356.0000  -9.93      0               6     5
236        30839  834042.000000  1194.0000 -10.63      0               6     5
237         8829  537493.000000  1423.0000 -10.73      9               0     5
238         9235  404940.000000  1112.0000 -11.23      9               0     5
239        37882  294903.000000  1783.0000  -7.80      0               6     5

[240 rows x 7 columns]
```

```
original.head()
```

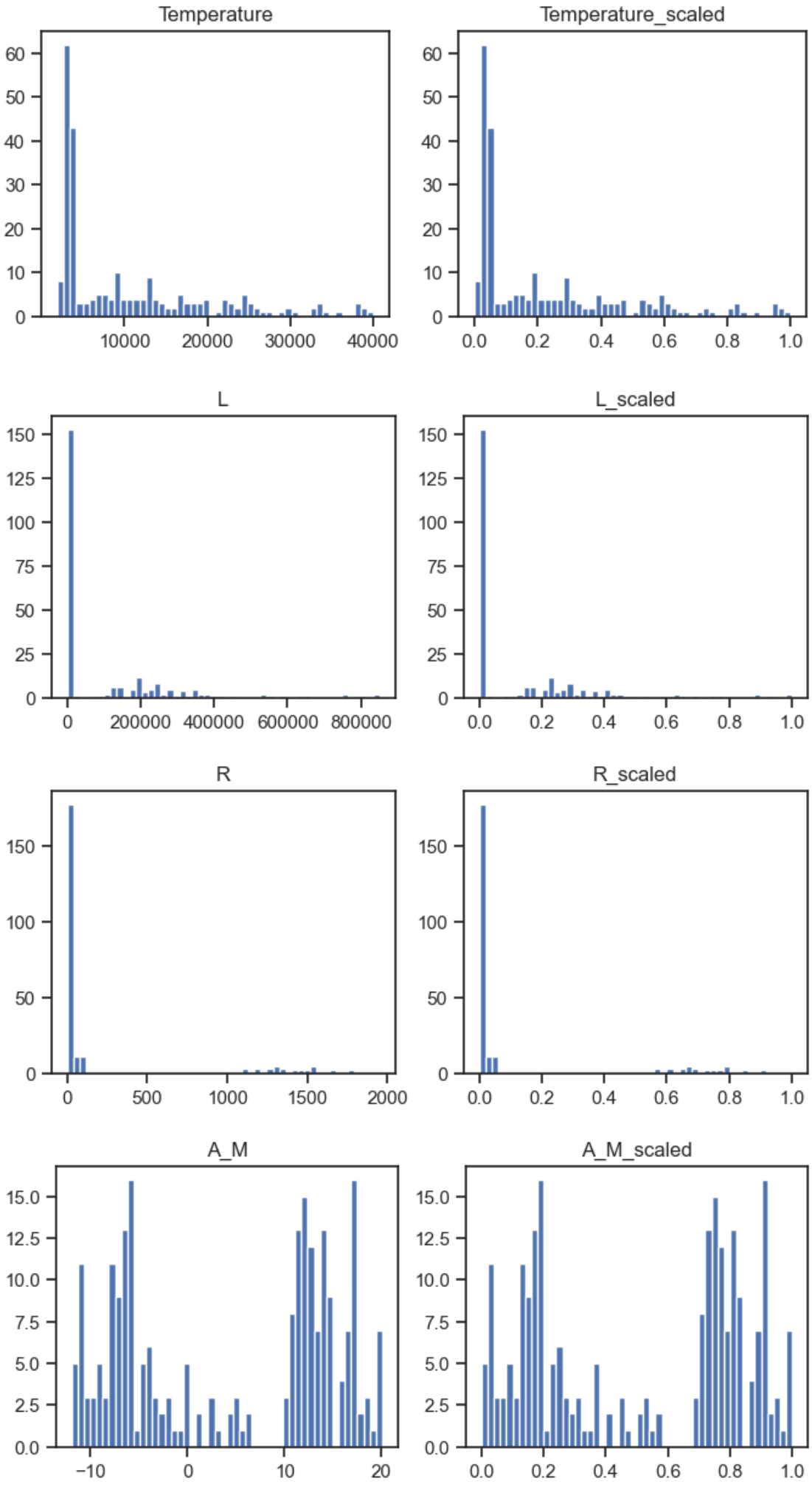
```
   Temperature         L       R    A_M  Color  Spectral_Class  Type
0         3068  0.002400  0.1700  16.12      8               5     0
1         3042  0.000500  0.1542  16.60      8               5     0
2         2600  0.000300  0.1020  18.70      8               5     0
3         2800  0.000200  0.1600  16.65      8               5     0
4         1939  0.000138  0.1030  20.06      8               5     0
```

```
# Числовые колонки для масштабирования
scale_cols = ['Temperature', 'L', 'R', 'A_M']

sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(original[scale_cols])

# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    original[new_col_name] = sc1_data[:,i]
```

```
original.head()
```

```
   Temperature         L       R    A_M  Color  Spectral_Class  Type  \
0         3068  0.002400  0.1700  16.12      8               5     0
1         3042  0.000500  0.1542  16.60      8               5     0
2         2600  0.000300  0.1020  18.70      8               5     0
3         2800  0.000200  0.1600  16.65      8               5     0
4         1939  0.000138  0.1030  20.06      8               5     0

   Temperature_scaled      L_scaled  R_scaled  A_M_scaled
0            0.029663  2.731275e-09  0.000083    0.876798
1            0.028980  4.944550e-10  0.000075    0.891807
2            0.017367  2.590003e-10  0.000048    0.957473
3            0.022622  1.412729e-10  0.000078    0.893371
4            0.000000  6.828189e-11  0.000049    1.000000
```

```
# 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(original[col], 50)
    ax[1].hist(original[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```


Temperature
Temperature_scaled
L
L_scaled
R
R_scaled
A_M
A_M_scaled

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

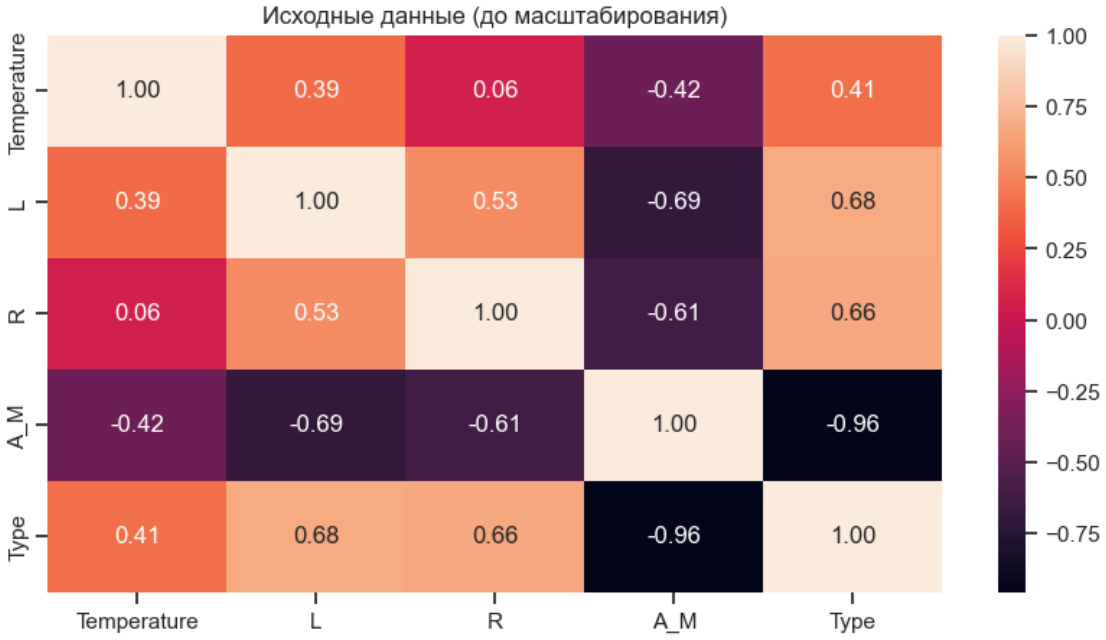
```
# Воспользуемся наличием тестовых выборок,
# включив их в корреляционную матрицу
corr_cols_1 = scale_cols + ['Type']
corr_cols_1
```

```
['Temperature', 'L', 'R', 'A_M', 'Type']
```

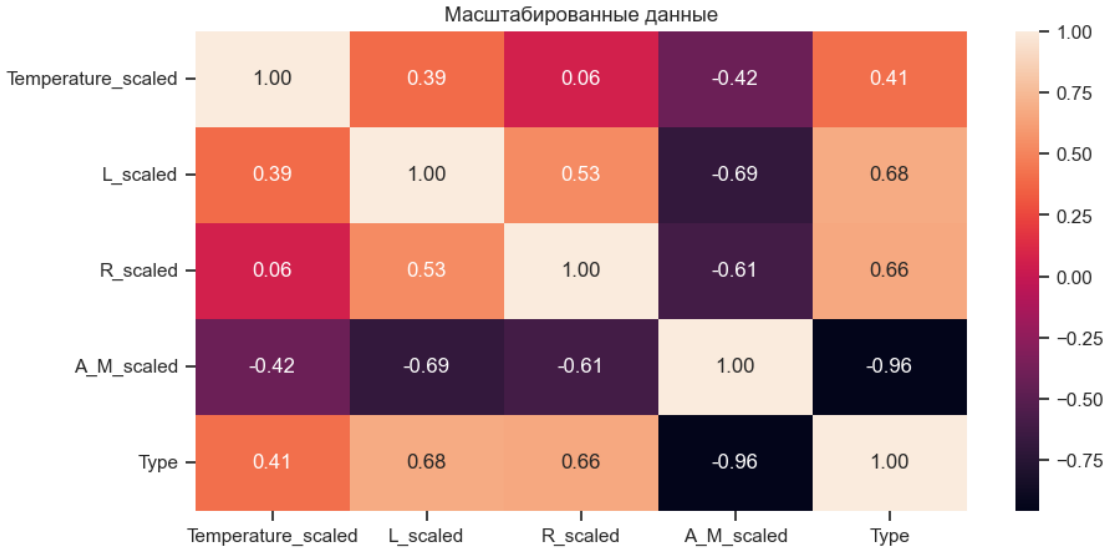
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Type']
corr_cols_2
```

```
['Temperature_scaled', 'L_scaled', 'R_scaled', 'A_M_scaled', 'Type']
```

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(original[corr_cols_1].corr(), annot=True, fmt='.2f')
ax.set_title('Исходные данные (до масштабирования)')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(original[corr_cols_2].corr(), annot=True, fmt='.2f')
ax.set_title('Масштабированные данные')
plt.show()
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак классификации "Type" наиболее сильно коррелирует с L (0.68), R (0.66) и A_M (-0.96). Эти признаки обязательно следует оставить в модели классификации.
- Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

Метрики, формируемые на основе матрицы ошибок:

Метрика precision:

Отображение доли верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Метрика recall (полнота):

Отображение доли верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Метрика ROC AUC

Для определения качества классификатора.

```
class MetricLogger:

    def __init__(self):
        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
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)]
.index, inplace = True)
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending
)
        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)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия
- Метод ближайших соседей
- Решающее дерево
- Случайный лес
- Градиентный бустинг

Формирование обучающей и тестовой выборок на основе исходного набора данных.

```
# На основе масштабированных данных выделим
# обучающую и тестовую выборки с помощью фильтра
X_train, X_test, y_train, y_test = train_test_split(
    original, original['Type'], test_size=0.2, random_state=1)
print(original.columns)
X_train.shape, X_test.shape
```

```
Index(['Temperature', 'L', 'R', 'A_M', 'Color', 'Spectral_Class', 'Type',
       'Temperature_scaled', 'L_scaled', 'R_scaled', 'A_M_scaled'],
      dtype='object')
```

```
((192, 11), (48, 11))
```

```
# Признаки для задачи классификации
task_clas_cols = ['Color', 'Spectral_Class',
        'Temperature_scaled', 'L_scaled', 'R_scaled', 'A_M_scaled']

# Выборки для задачи классификации
clas_X_train = X_train[task_clas_cols]
clas_X_test = X_test[task_clas_cols]
clas_Y_train = y_train
clas_Y_test = y_test
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

```
((192, 6), (48, 6), (192,), (48,))
```

```
clas_Y_test.values
```

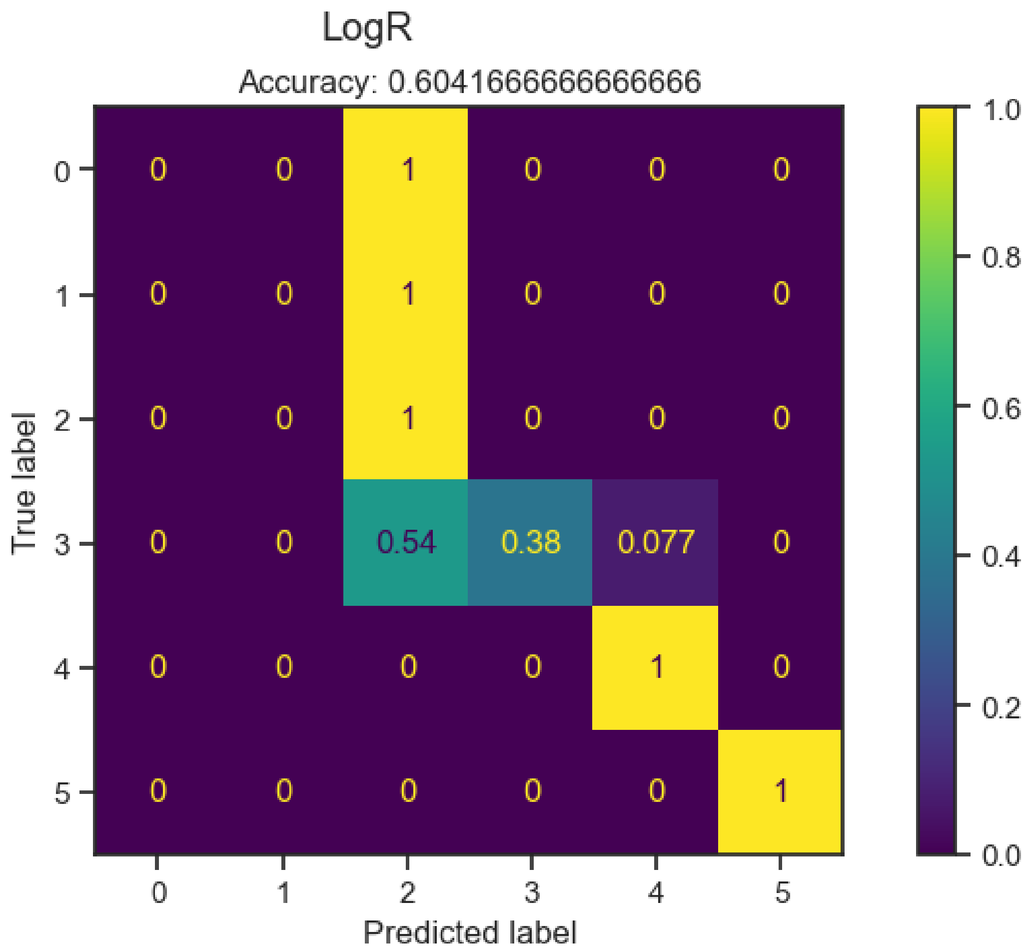
```
array([4, 1, 2, 3, 3, 0, 3, 2, 5, 5, 5, 0, 1, 3, 2, 3, 5, 0, 3, 4, 5, 5,
       4, 3, 5, 5, 0, 0, 3, 2, 2, 4, 2, 2, 4, 3, 3, 1, 4, 5, 1, 3, 5, 3,
       0, 3, 1, 2], dtype=int64)
```

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.

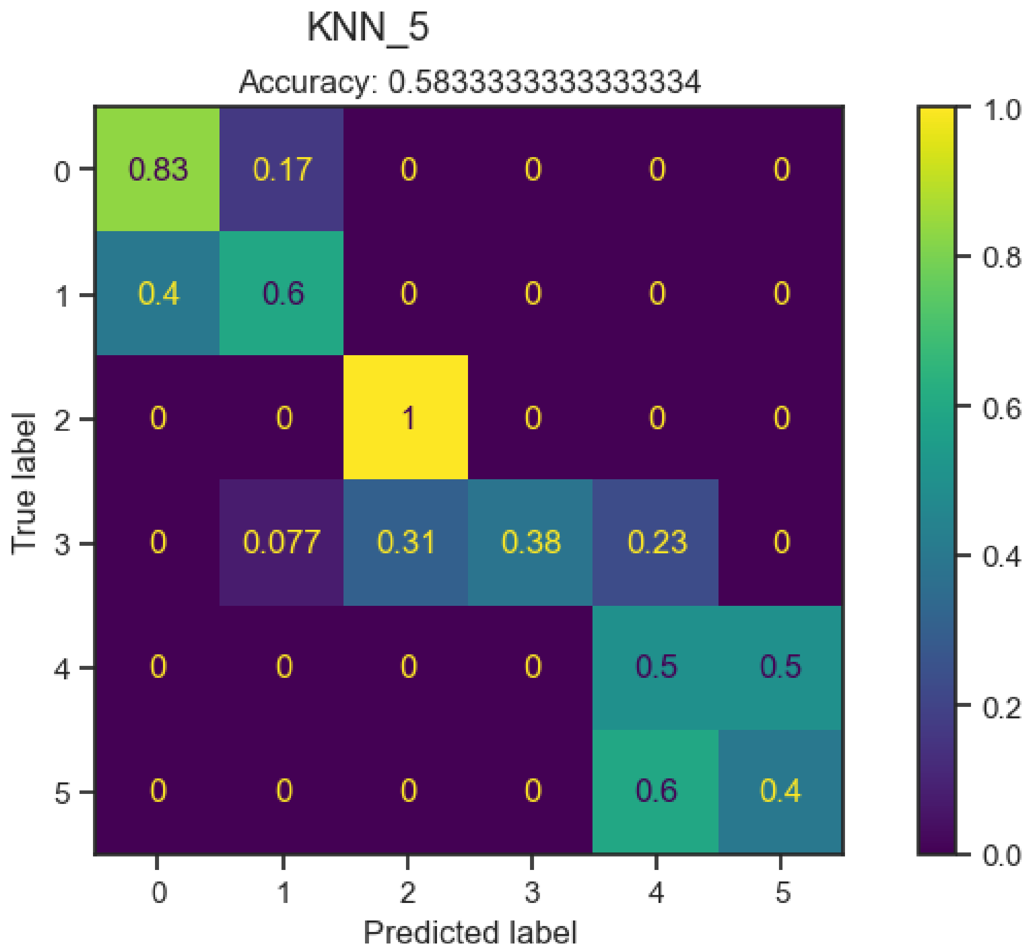
```
# Сохранение метрик
clasMetricLogger = MetricLogger()

# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average, multi_class
=False)
    #plt.figure()
    lw = 2
    ax.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_xlim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic')
    ax.legend(loc="lower right")

def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(X_train, y_train)
    # Предсказание значений
    Y_pred = model.predict(X_test)

    accuracy = accuracy_score(clas_Y_test.values, Y_pred)
    precision = precision_score(clas_Y_test.values, Y_pred, average='micro')
    recall = recall_score(clas_Y_test.values, Y_pred, average='micro')

    clasMetricLogger.add('accuracy', model_name, accuracy)
    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)

    fig, ax = plt.subplots(nrows=1, figsize=(10,5))
```

```
    cm = confusion_matrix(clas_Y_test, Y_pred, labels=np.unique(clas_Y_train),
normalize='true')
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique
(clas_Y_train))
    disp.plot(ax=ax)
    ax.set_title("Accuracy: {}".format(accuracy_score(clas_Y_test.values, Y_pre
d)))

    fig.suptitle(model_name)
    plt.show()


for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

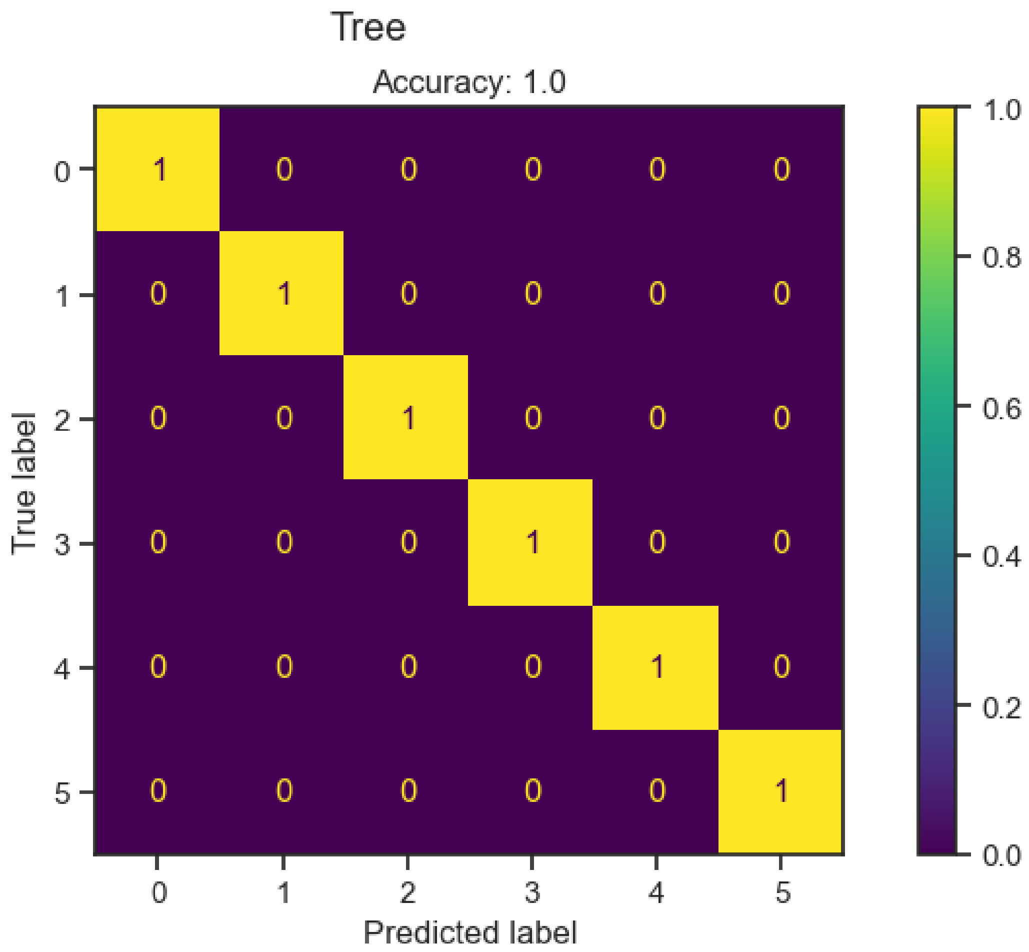
```
C:\Users\prite\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:45
8: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on
  n_iter_i = _check_optimize_result(
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



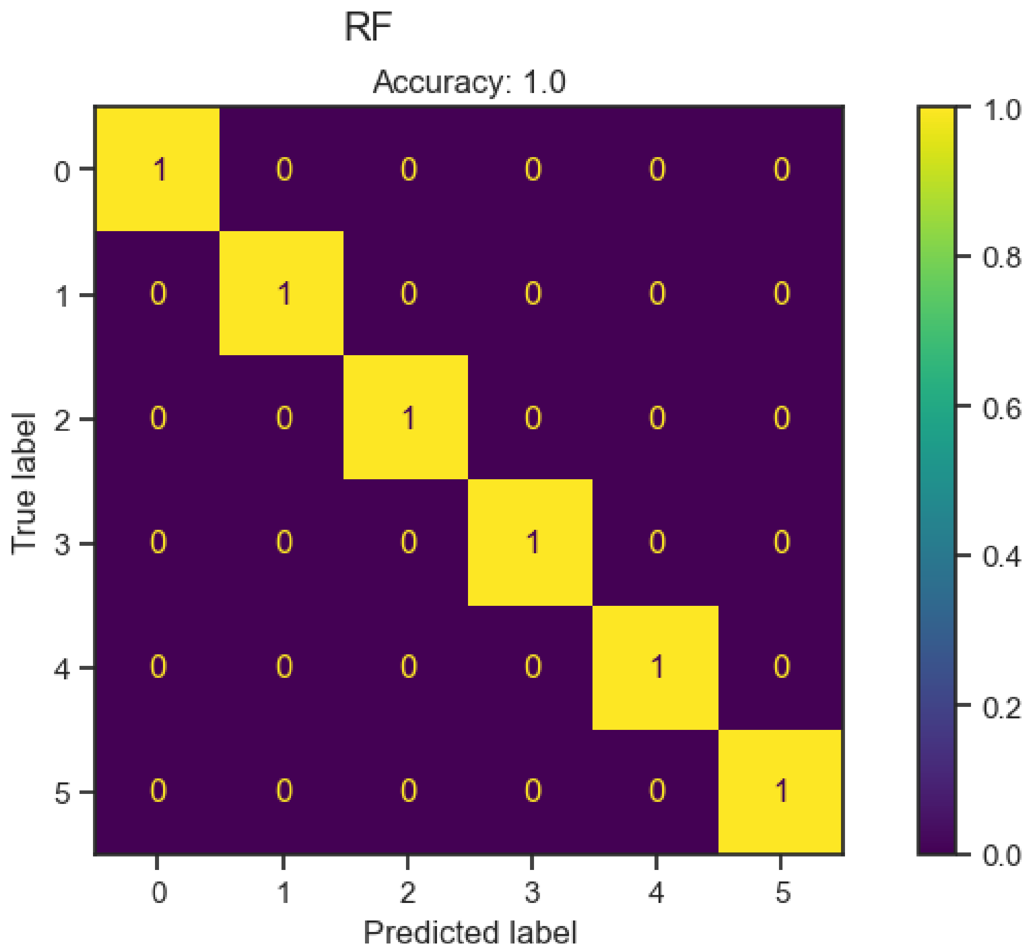

```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



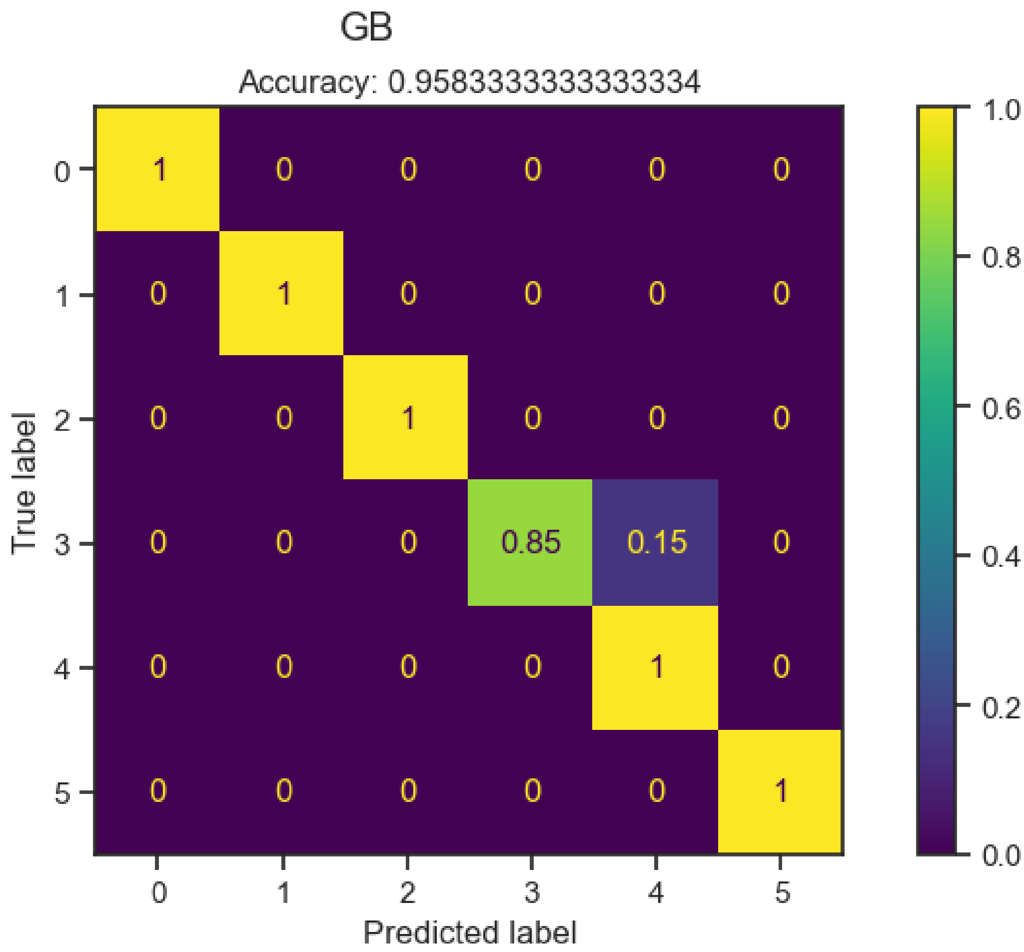

```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



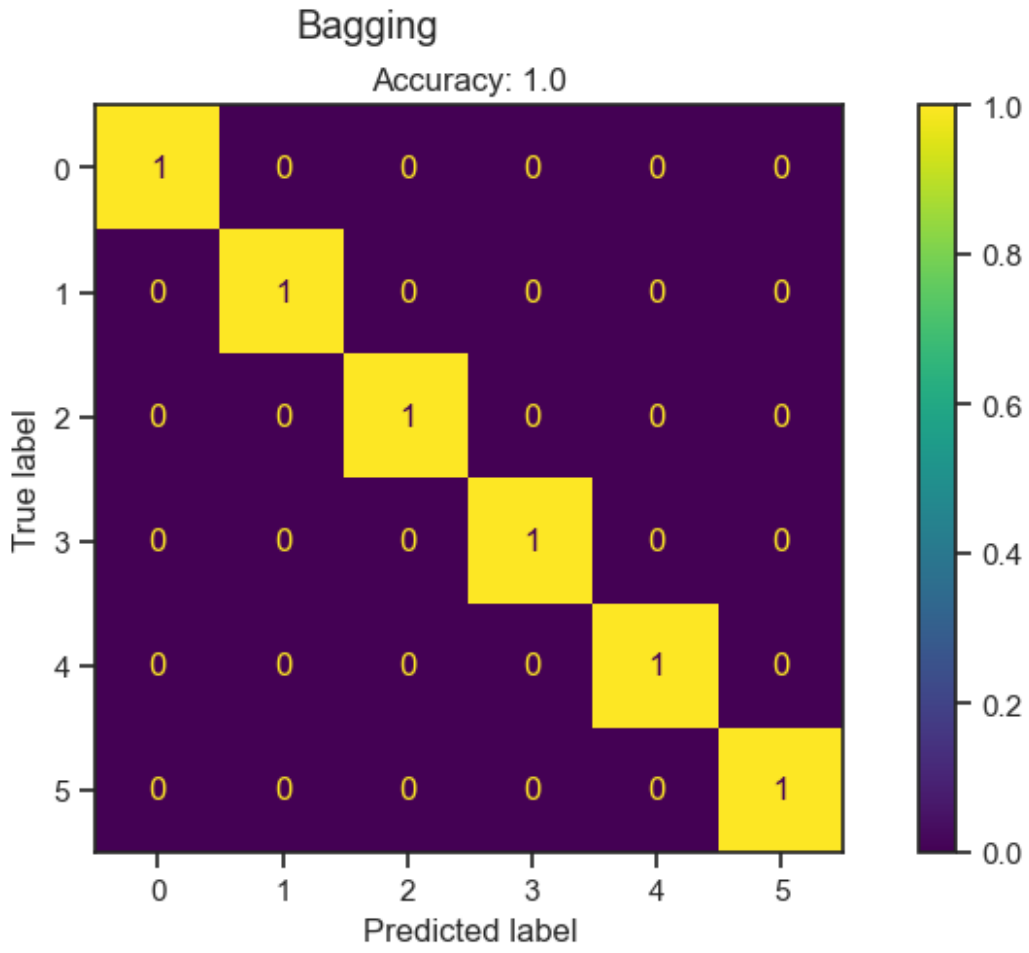

```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```




```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```




```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.

```
clas_X_train.shape
```

```
(192, 6)
```

```
n_range = np.array(range(2,31,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 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])}]
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='
accuracy')
clf_gs.fit(clas_X_train, clas_Y_train)
```

```
CPU times: total: 1.14 s
Wall time: 1.17 s
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 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])}],
             scoring='accuracy')
```

```
# Лучшая модель
clf_gs.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=2)
```

```
# Лучшее значение параметров
clf_gs.best_params_
```

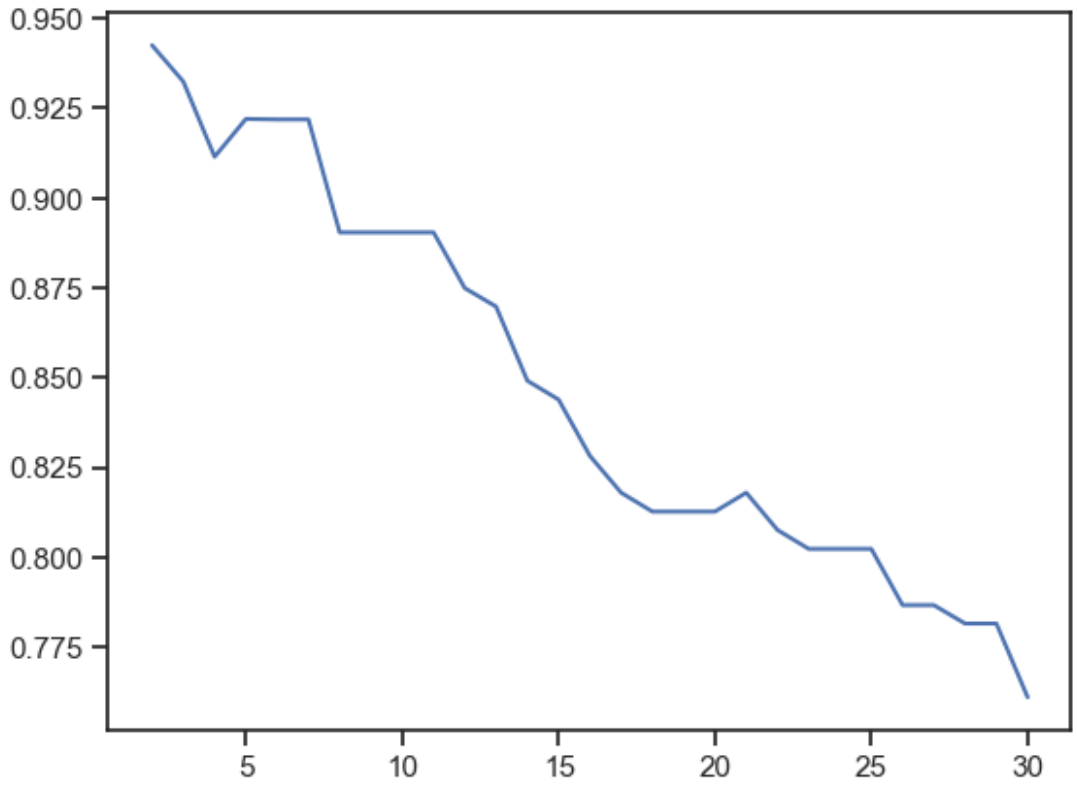
```
{'n_neighbors': 2}
```

```
clf_gs_best_params_txt = str(clf_gs.best_params_['n_neighbors'])
clf_gs_best_params_txt
```

```
'2'
```

```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x2944a1f2410>]
```

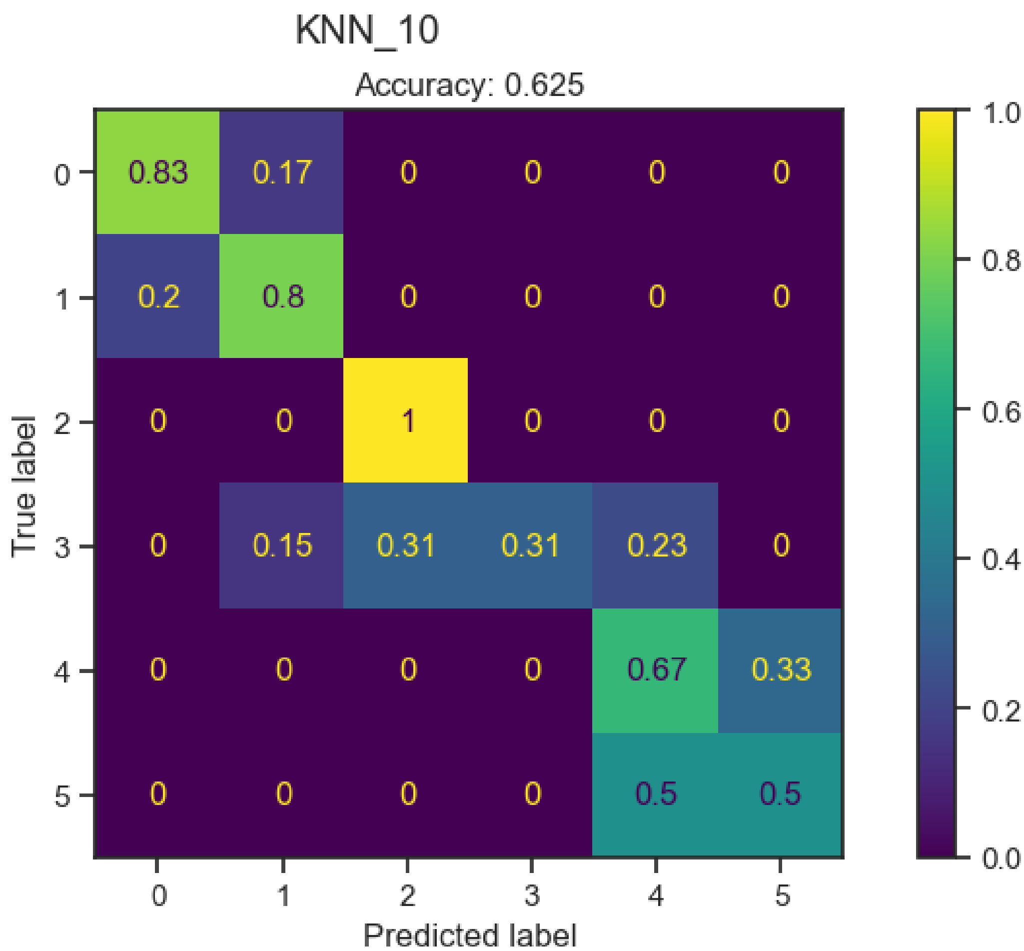


Повторение пункта для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

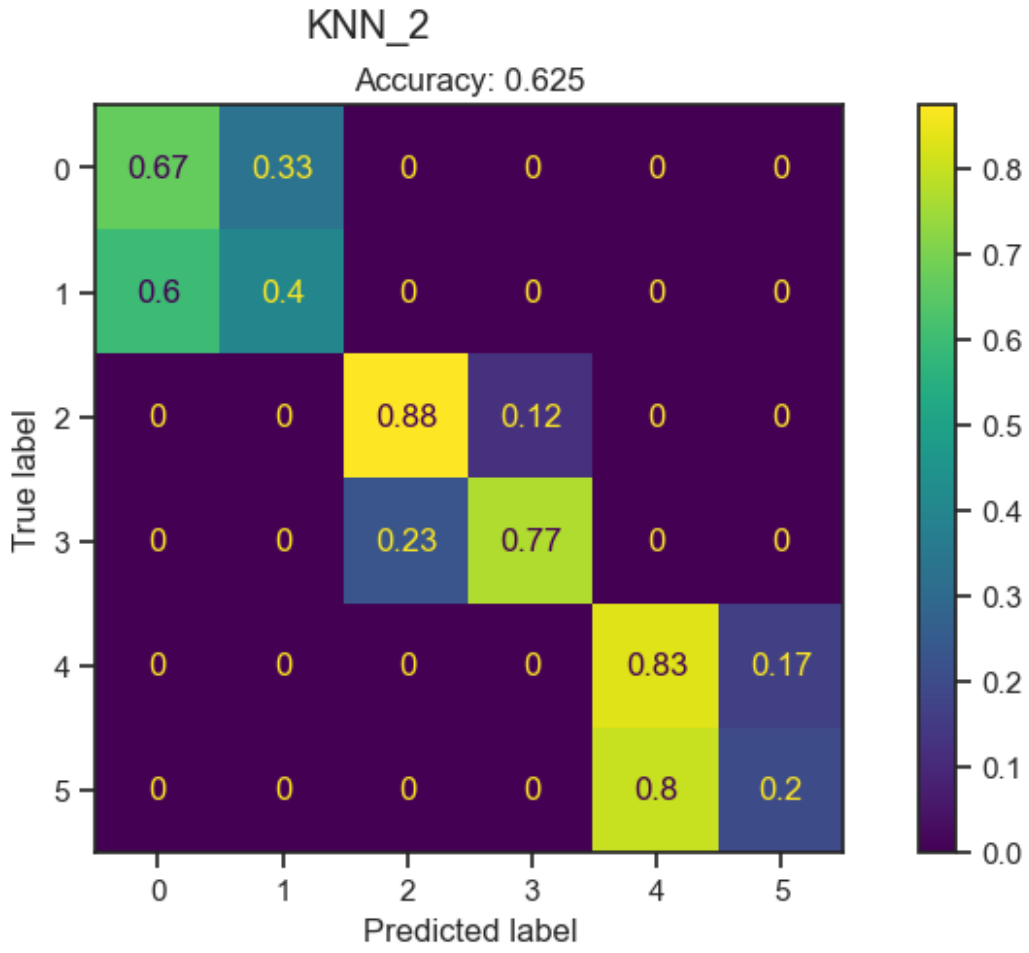
```
clas_models_grid = {'KNN_10':KNeighborsClassifier(n_neighbors=10),
                    str('KNN_' + clf_gs_best_params_txt):clf_gs.best_estimator_
}

for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
```

```
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```




```
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
C:\Users\prite\AppData\Local\Temp\ipykernel_26244\368643467.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a f
uture version. Use pandas.concat instead.
  self.df = self.df.append(temp, ignore_index=True)
```



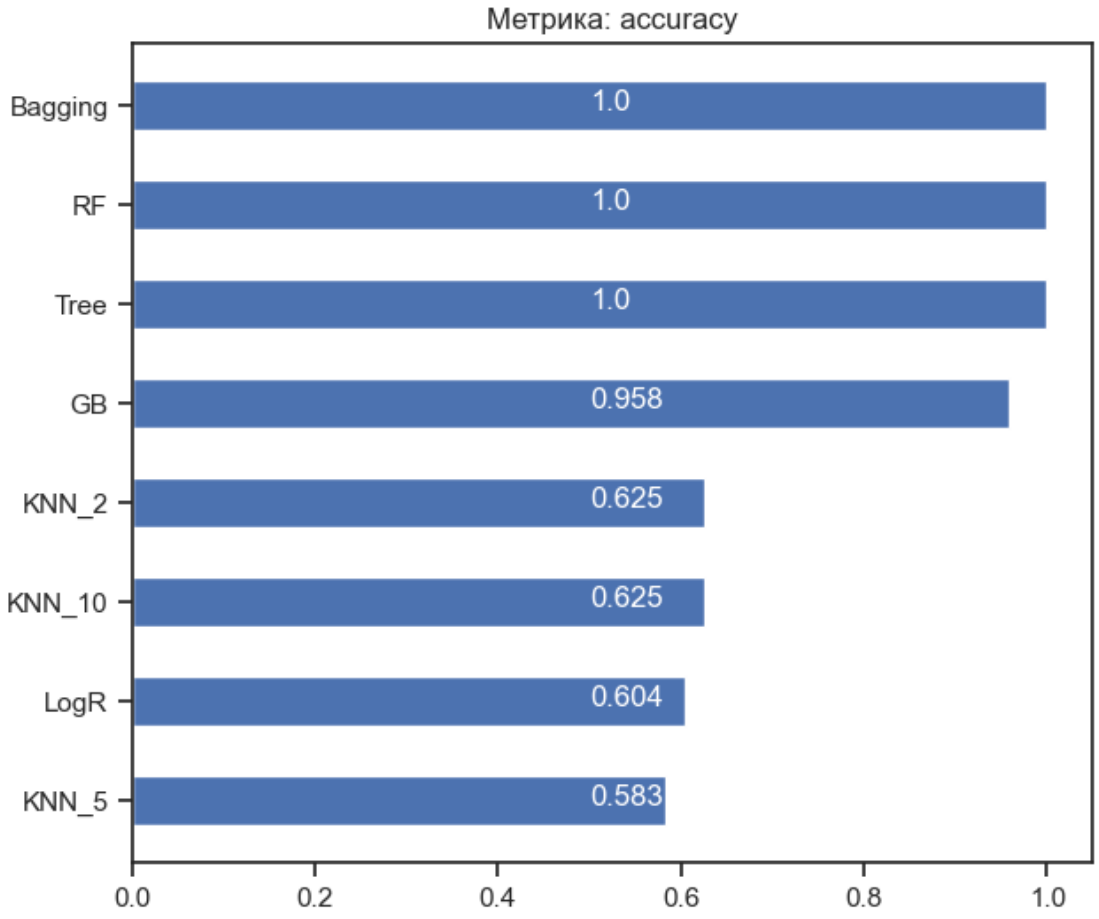
Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

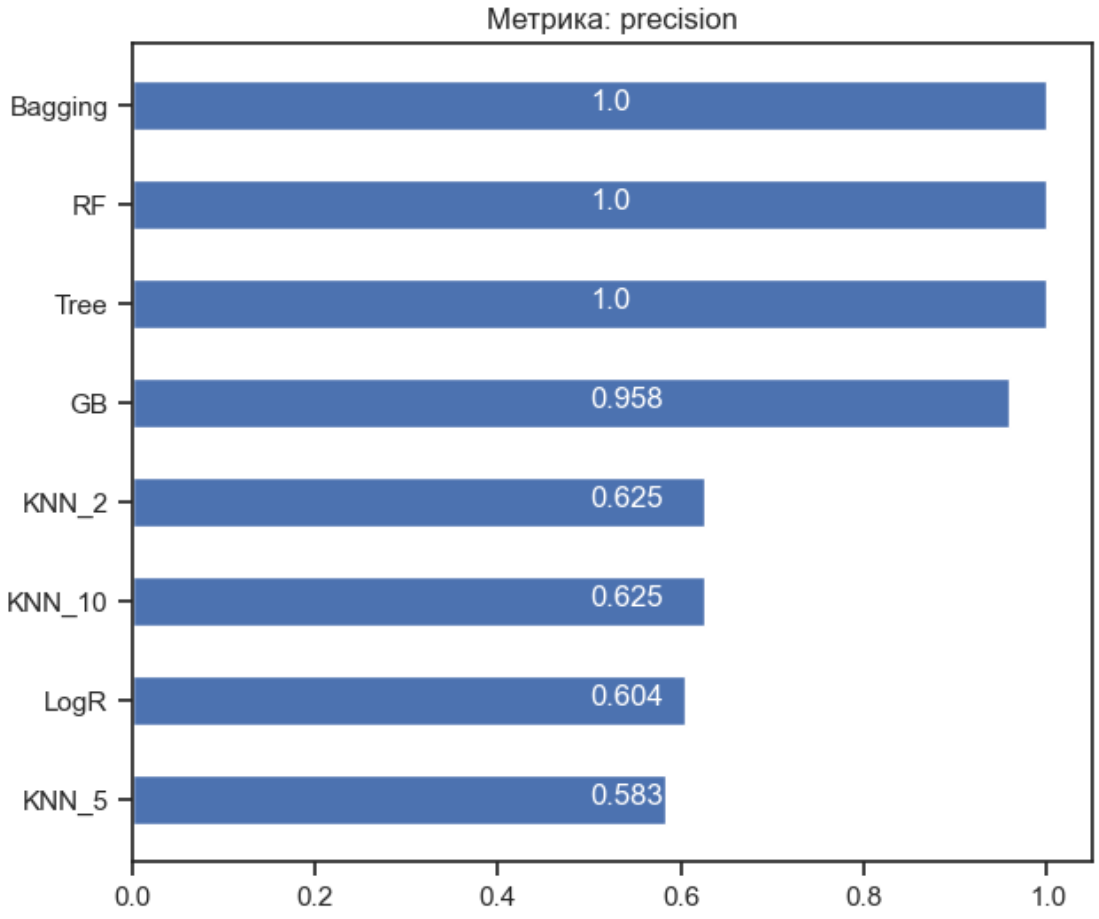
Решение задачи классификации

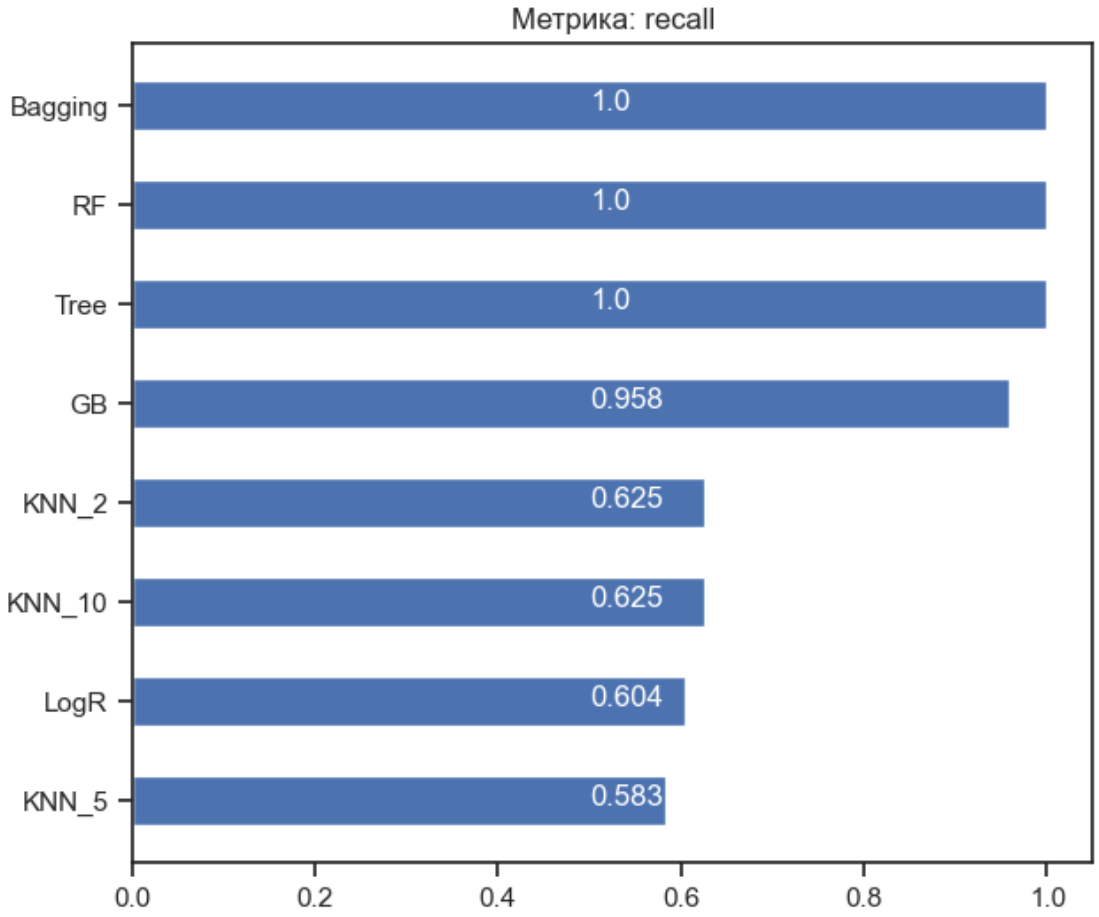
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

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

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


Метрика: accuracy
Bagging
RF
Tree
GB
KNN_2
KNN_10
LogR
KNN_5
1.0
1.0
1.0
0.958
0.625
0.625
0.604
0.583
0.0
0.2
0.4
0.6
0.8
1.0


Метрика: precision
Bagging
RF
Tree
GB
KNN_2
KNN_10
LogR
KNN_5
1.0
1.0
1.0
0.958
0.625
0.625
0.604
0.583
0.0
0.2
0.4
0.6
0.8
1.0



Вывод: на основании трех метрик, лучшей оказалась модель случайного леса и Дерево решений.