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:

_____ *Предсказание наличия сердечных*___ _____ _____*Заболеваний*___ _____ _____

Студент ___ИУ5-63б_____ _____ __**Бахрамов Н.А.**_____
(Группа) (Подпись, дата)
(И.О.Фамилия)

Руководитель _____ ____**Гапанюк Ю.Е.**___
(Подпись, дата) (И.О.Фамилия)

Консультант _____ ____**Гапанюк Ю.Е.**___
(Подпись, дата) (И.О.Фамилия)

2022 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.М.Черненький __
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

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

по теме ___ предсказание наличия сердечных заболеваний_____

_____

_____

Студент группы _____ ИУ5-63б ____

___Миронова Александра Романовна_____
(Фамилия, имя, отчество)

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

_____исследовательская_____

Источник тематики (кафедра, предприятие, НИР) _____НИР_____

График выполнения НИР: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Техническое задание ___*разработать и обучить модель предсказывающую наличие сердечных заболеваний у пациентов*

Оформление научно-исследовательской работы:

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

_____

Дата выдачи задания « 20 » _февраля___ 2022_ г.

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

Студент _____ __ **Бахрамов Н.А** ____
(Подпись, дата) (И.О.Фамилия)

Содержание

Схема типового исследования

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

1. 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, или задачи регрессии.
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. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
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. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
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. 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.

На основе датасета Heart будем предсказывать наличие сердечно-сосудистых заболеваний

```
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, classificat
ion_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_square
d_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, LinearSV
R
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_g
raphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegress
or
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
%matplotlib inline
sns.set(style="ticks")
```

Описание колонок датасета

- age
- sex
- chest pain type (4 values)
- resting blood pressure
- serum cholestoral in mg/dl
- fasting blood sugar > 120 mg/dl
- resting electrocardiographic results (values 0,1,2)
- maximum heart rate achieved
- exercise induced angina
- oldpeak = ST depression induced by exercise relative to rest
- the slope of the peak exercise ST segment
- number of major vessels (0-3) colored by flourosopy
- thal: 0 = normal; 1 = fixed defect; 2 = reversable defect

```
# Обучающая выборка
original_train = pd.read_csv('Heart_train.csv')
# Тестовая выборка
original_test = pd.read_csv('Heart_test.csv')

# Удалим дубликаты записей, если они присутствуют
train = original_train.drop_duplicates()
test = original_test.drop_duplicates()



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

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	58	1	0	114	318	0	2	140	0	4.4	0	
1	52	1	2	172	199	1	1	162	0	0.5	2	
2	53	1	0	142	226	0	0	111	1	0.0	2	
3	68	1	2	118	277	0	1	151	0	1.0	2	
4	62	1	2	130	231	0	1	146	0	1.8	1	

	ca	thal	target
0	3	1	0
1	0	3	1
2	0	3	1
3	1	3	1
4	3	3	1

```
test.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	55	1	0	132	353	0	1	132	1	1.2	1	
1	53	0	0	130	264	0	0	143	0	0.4	1	
2	45	0	1	130	234	0	0	175	0	0.6	1	
3	57	1	1	154	232	0	0	164	0	0.0	2	
4	59	1	3	160	273	0	0	125	0	0.0	2	

	ca	thal	target
0	1	3	0
1	0	2	1
2	0	2	1
3	1	2	0
4	0	2	0

```
# Размер обучающего датасета - 1895 строк, 34 колонок
train.shape, test.shape
```

```
((243, 14), (60, 14))
```

```
train.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

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

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

```
test.dtypes
```

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

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

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
```

```
thal        0
target      0
dtype: int64
```

```
test.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

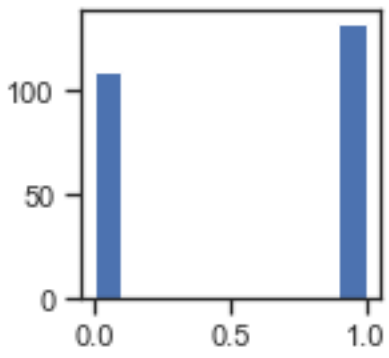
Вывод. Представленный набор данных не содержит пропусков ни в обучающей, ни в тестовой выборках.

Построение графиков для понимания структуры данных

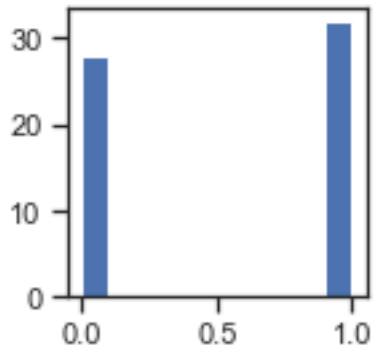
```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
train['target'].unique()
```

```
array([0, 1])
```

```
# Оценим дисбаланс классов для Target
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(train['target'])
plt.show()
```




```
# Оценим дисбаланс классов для Target
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(test['target'])
plt.show()
```




```
train['target'].value_counts()
```

```
1    133
0    110
Name: target, dtype: int64
```

```
test['target'].value_counts()
```

```
1    32
0    28
Name: target, dtype: int64
```

```
# посчитаем дисбаланс классов
total = train.shape[0]
class_0, class_1 = train['target'].value_counts()
print('Класс 1 составляет {}%, а класс 0 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

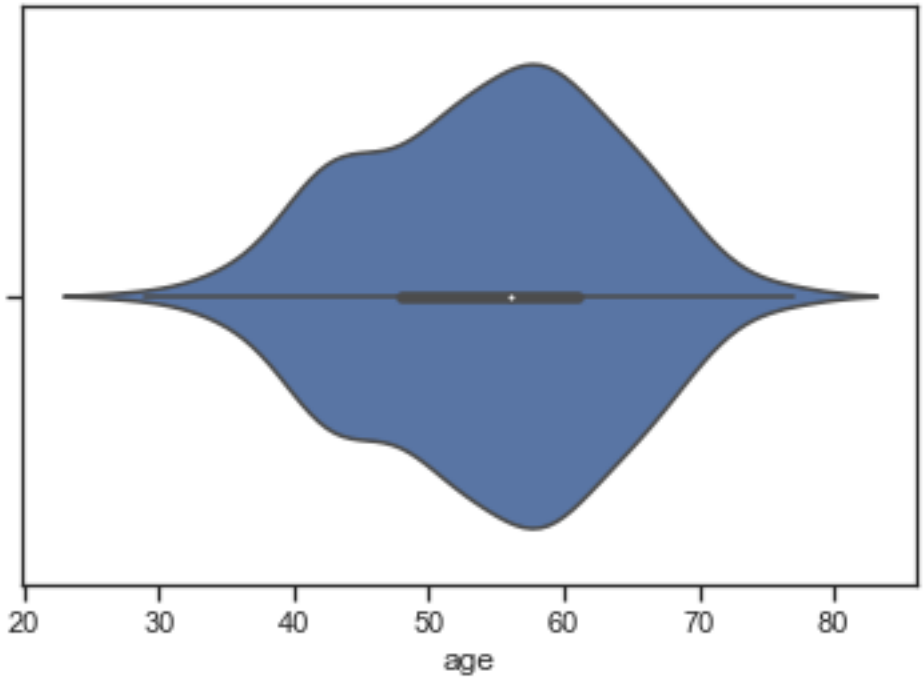
```
Класс 1 составляет 54.730000000000004%, а класс 0 составляет 45.269999999999996%
.
```

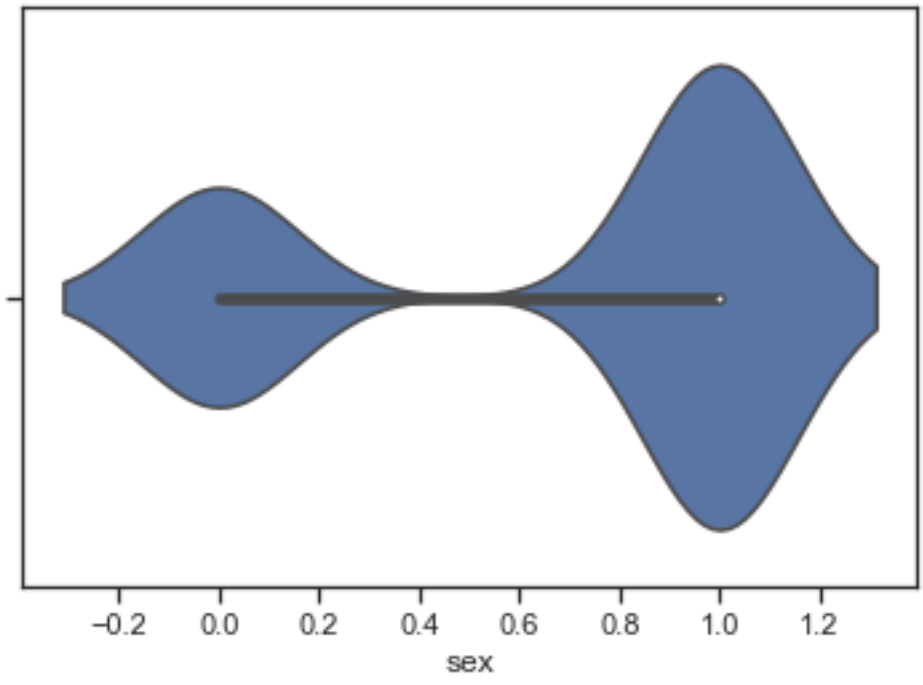
Вывод: Дисбаланс классов практически отсутствует

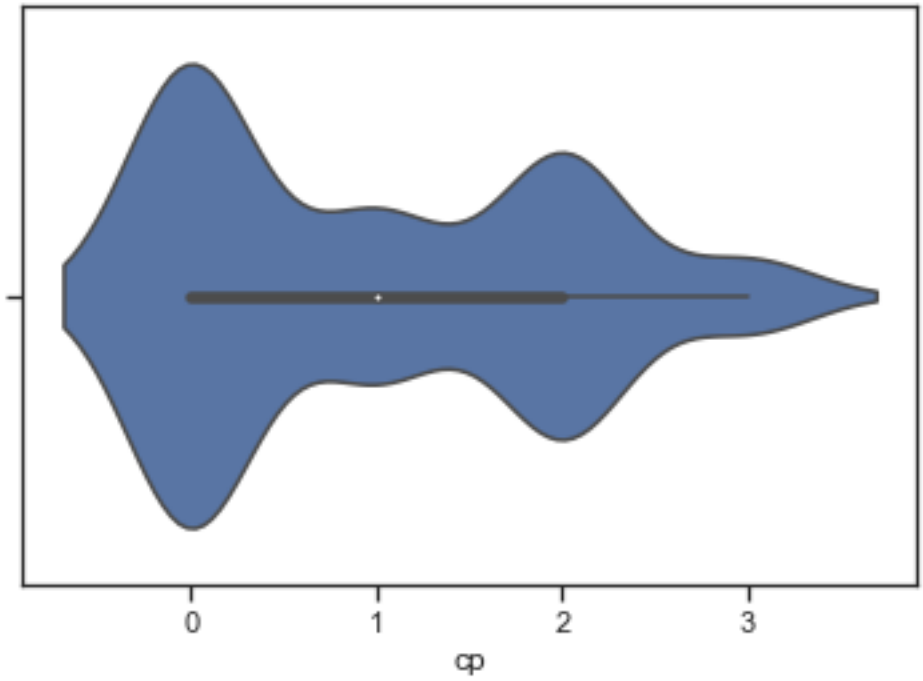
```
train.columns
```

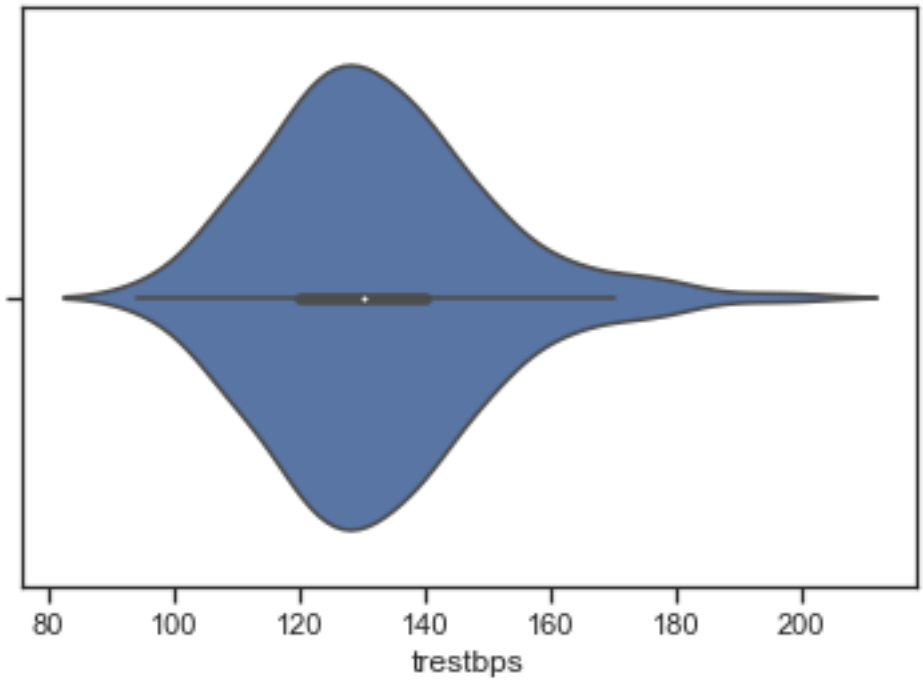
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

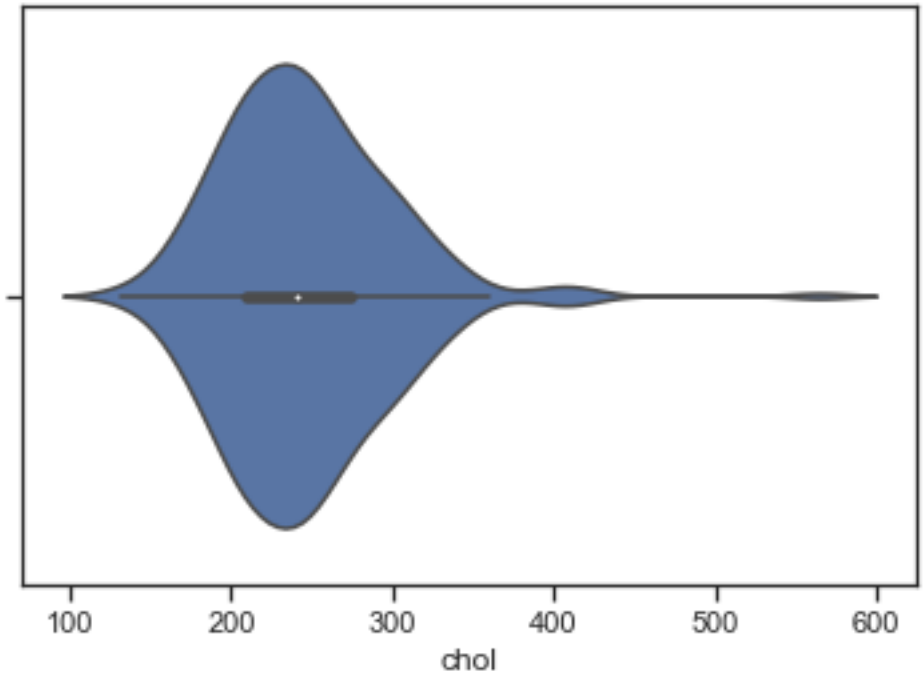
```
# Скрипичные диаграммы для числовых колонок
for col in ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']:
    sns.violinplot(x=train[col])
    plt.show()
```

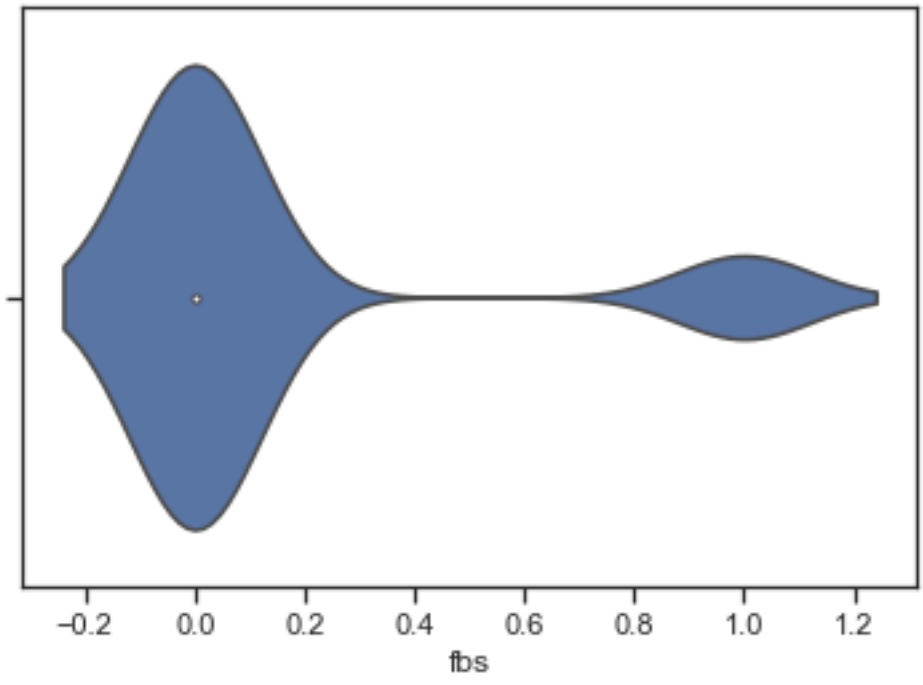

20
30
40
50
60
70
80
age

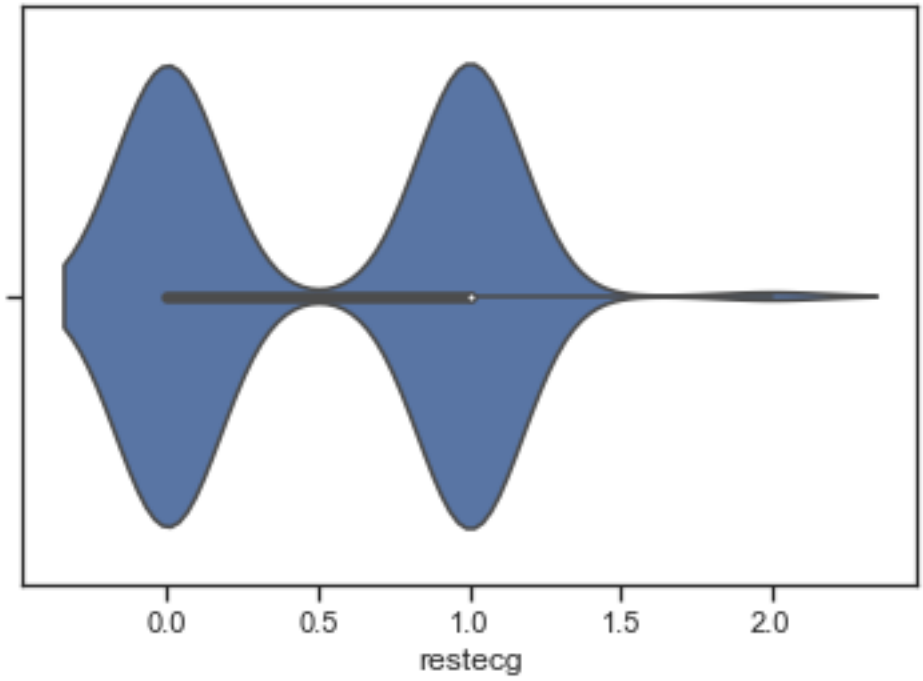

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
sex

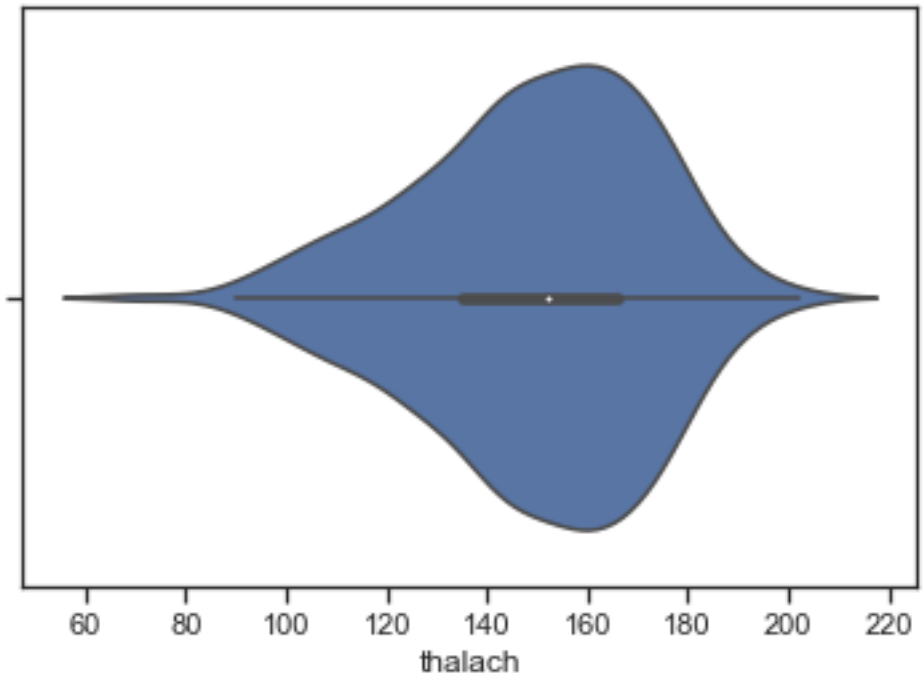

0
1
2
3
cp

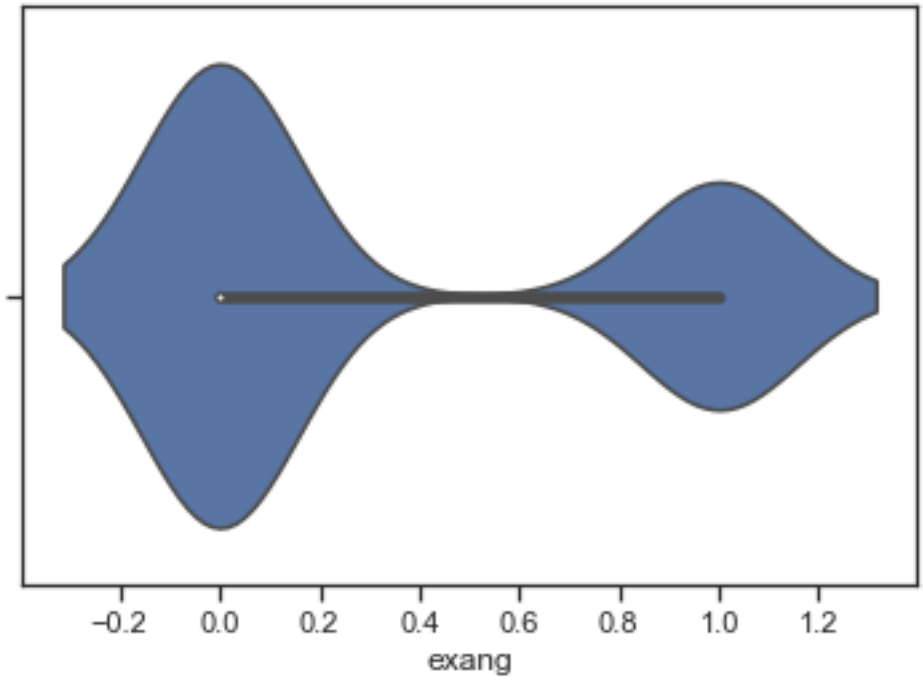

80
100
120
140
160
180
200
trestbps

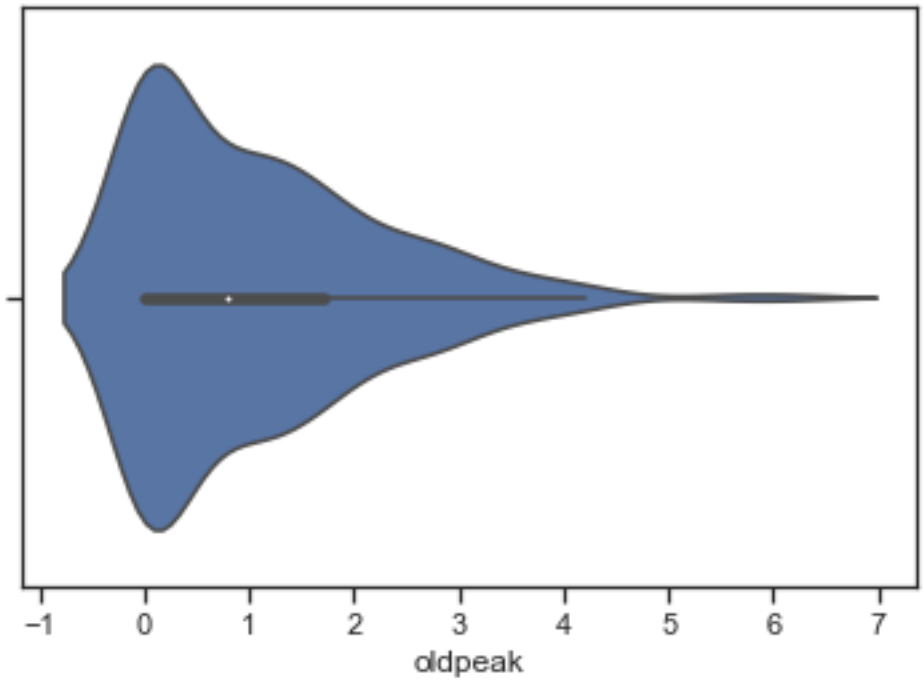

100
200
300
400
500
600
chol

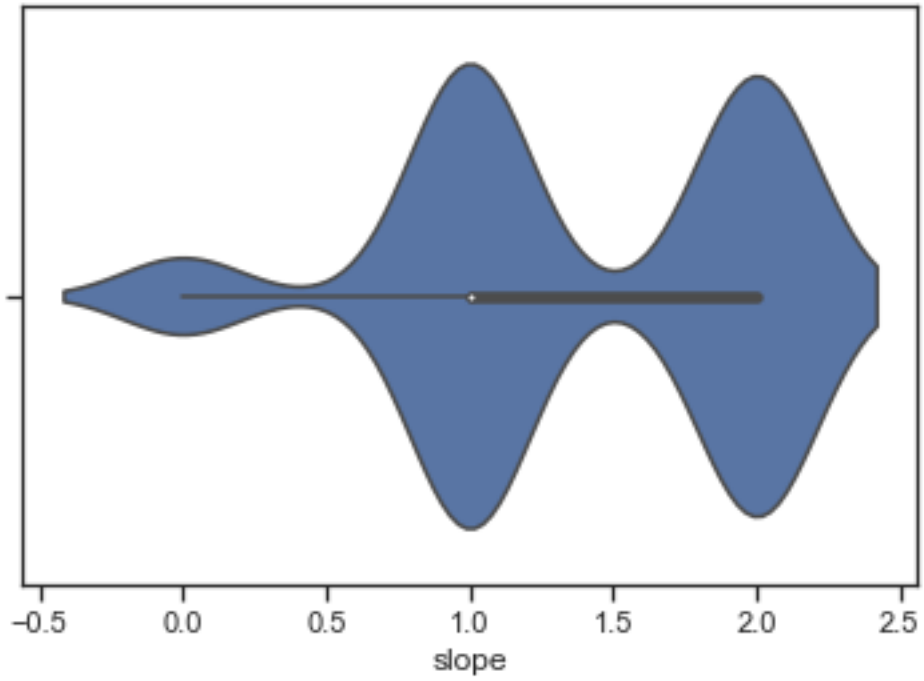

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
fbs

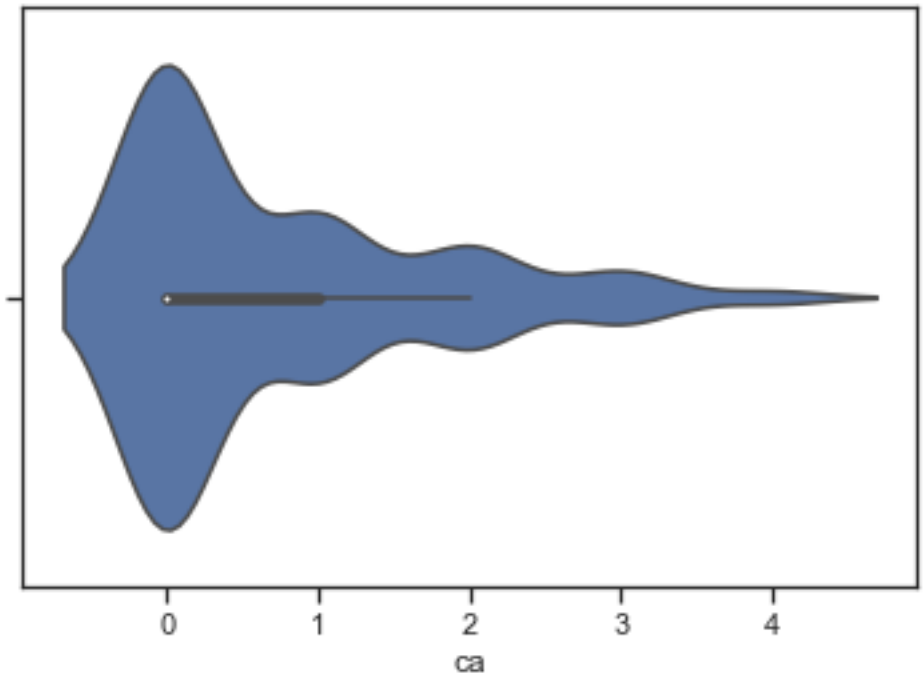

0.0
0.5
1.0
1.5
2.0
restecg

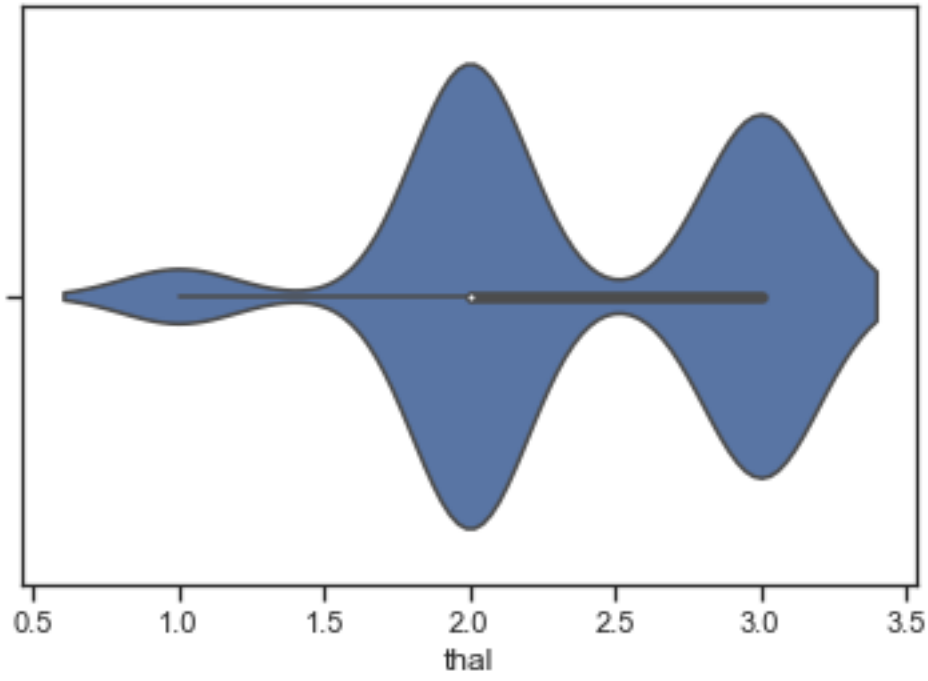

60
80
100
120
140
160
180
200
220
thalach

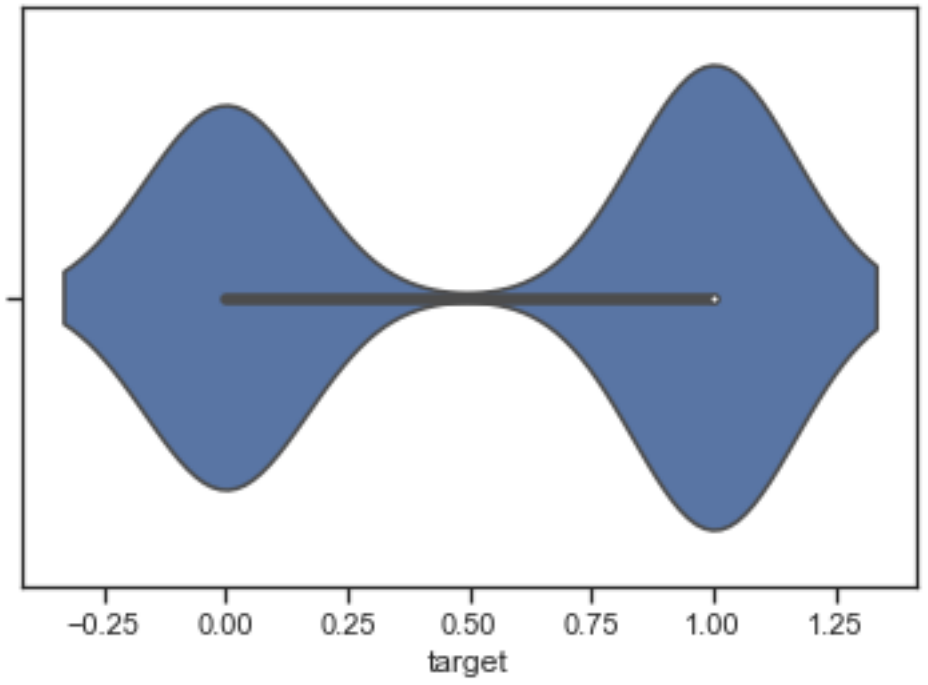

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
exang


−1
0
1
2
3
4
5
6
7
oldpeak


−0.5
0.0
0.5
1.0
1.5
2.0
2.5
slope


0
1
2
3
4
ca





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

```
train.dtypes
```

```
age            int64
sex            int64
cp             int64
trestbps       int64
```

```
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей в данном примере мы строить не будем.

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

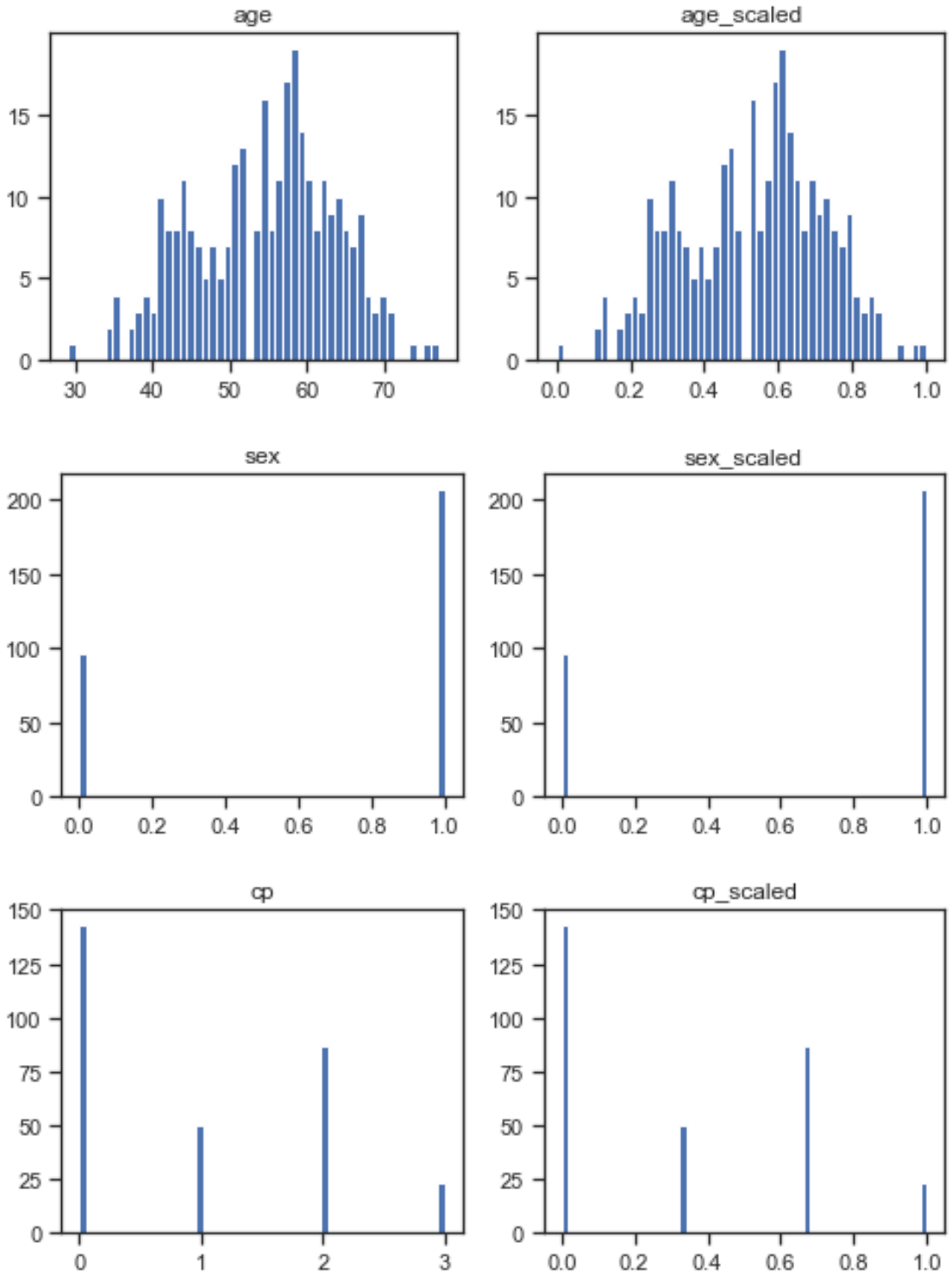
```
# Создадим вспомогательные колонки,
# чтобы наборы данных можно было разделить.
train['dataset'] = 'TRAIN'
test['dataset'] = 'TEST'

# Колонки для объединения
join_cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach'
,
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target','dataset']

data_all = pd.concat([train[join_cols], test[join_cols]])

# Проверим корректность объединения
assert data_all.shape[0] == train.shape[0]+test.shape[0]

data_all.head()
```

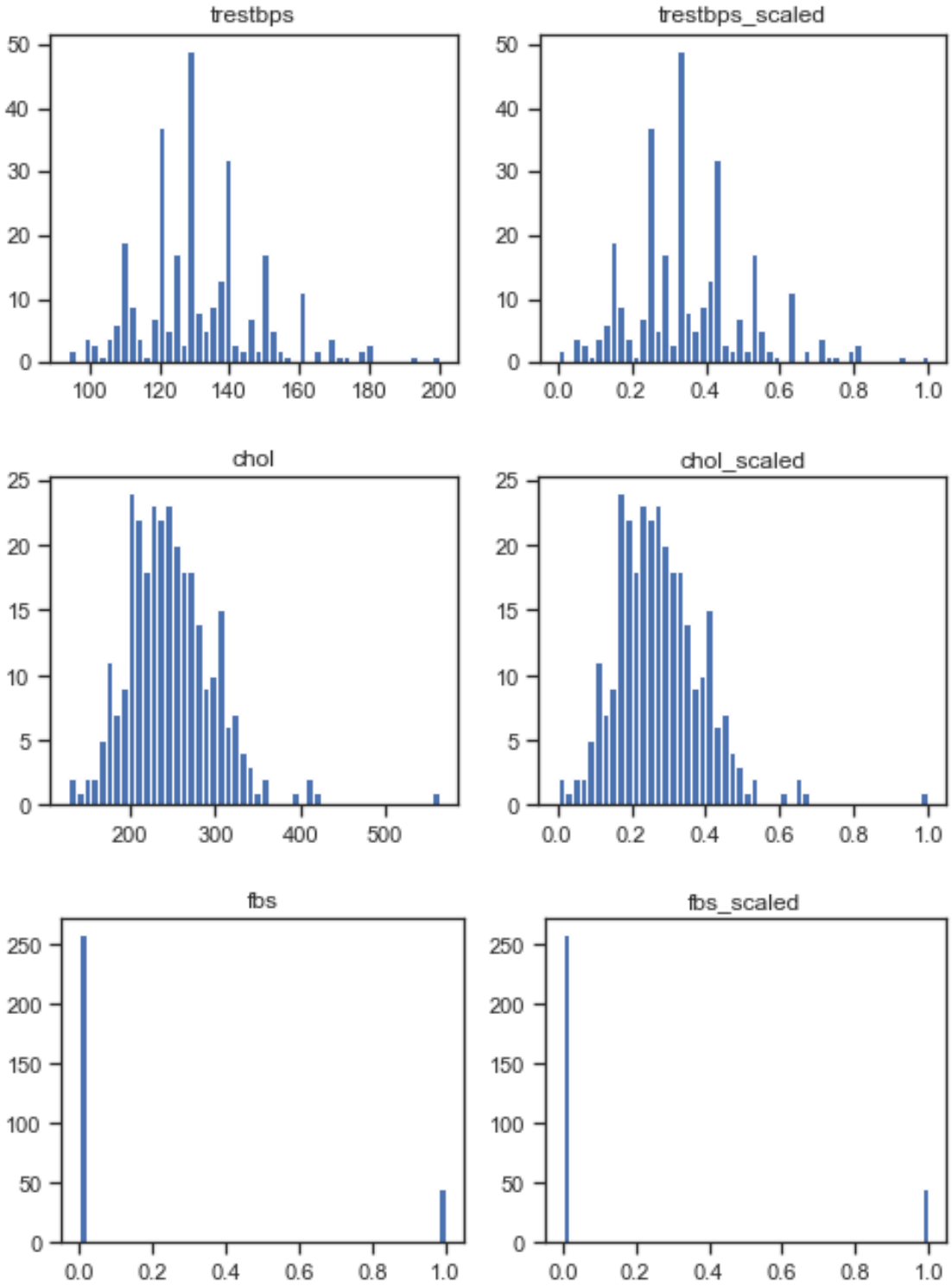
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   58    1   0       114   318    0        2      140      0      4.4      0
1   52    1   2       172   199    1        1      162      0      0.5      2
2   53    1   0       142   226    0        0      111      1      0.0      2
3   68    1   2       118   277    0        1      151      0      1.0      2
4   62    1   2       130   231    0        1      146      0      1.8      1

   ca  thal  target dataset
0   3     1       0   TRAIN
1   0     3       1   TRAIN
2   0     3       1   TRAIN
3   1     3       1   TRAIN
4   3     3       1   TRAIN
```

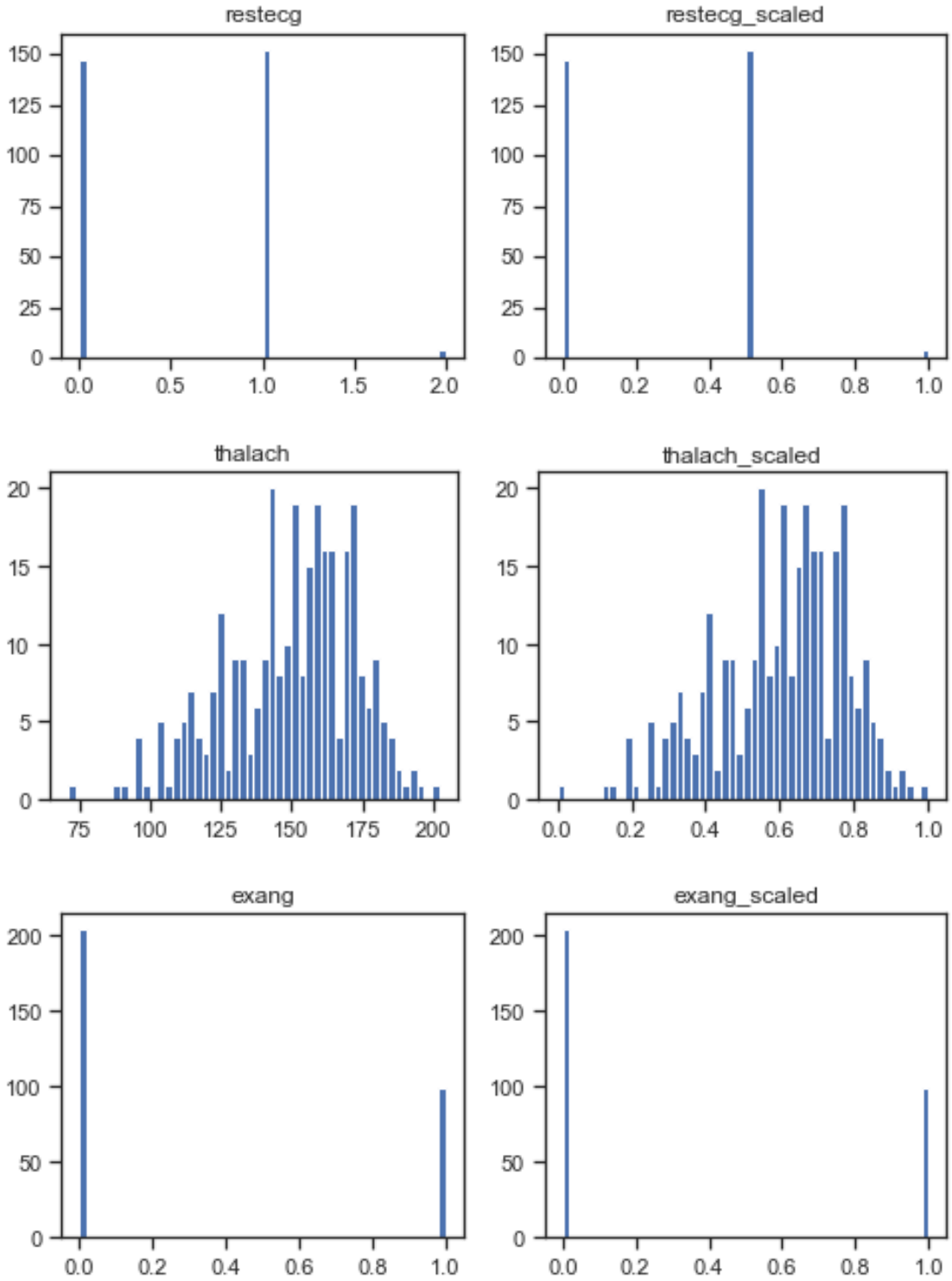
```
# Числовые колонки для масштабирования
scale_cols = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach
',
       'exang', 'oldpeak', 'slope', 'ca', 'thal']

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

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

data_all.head()
```

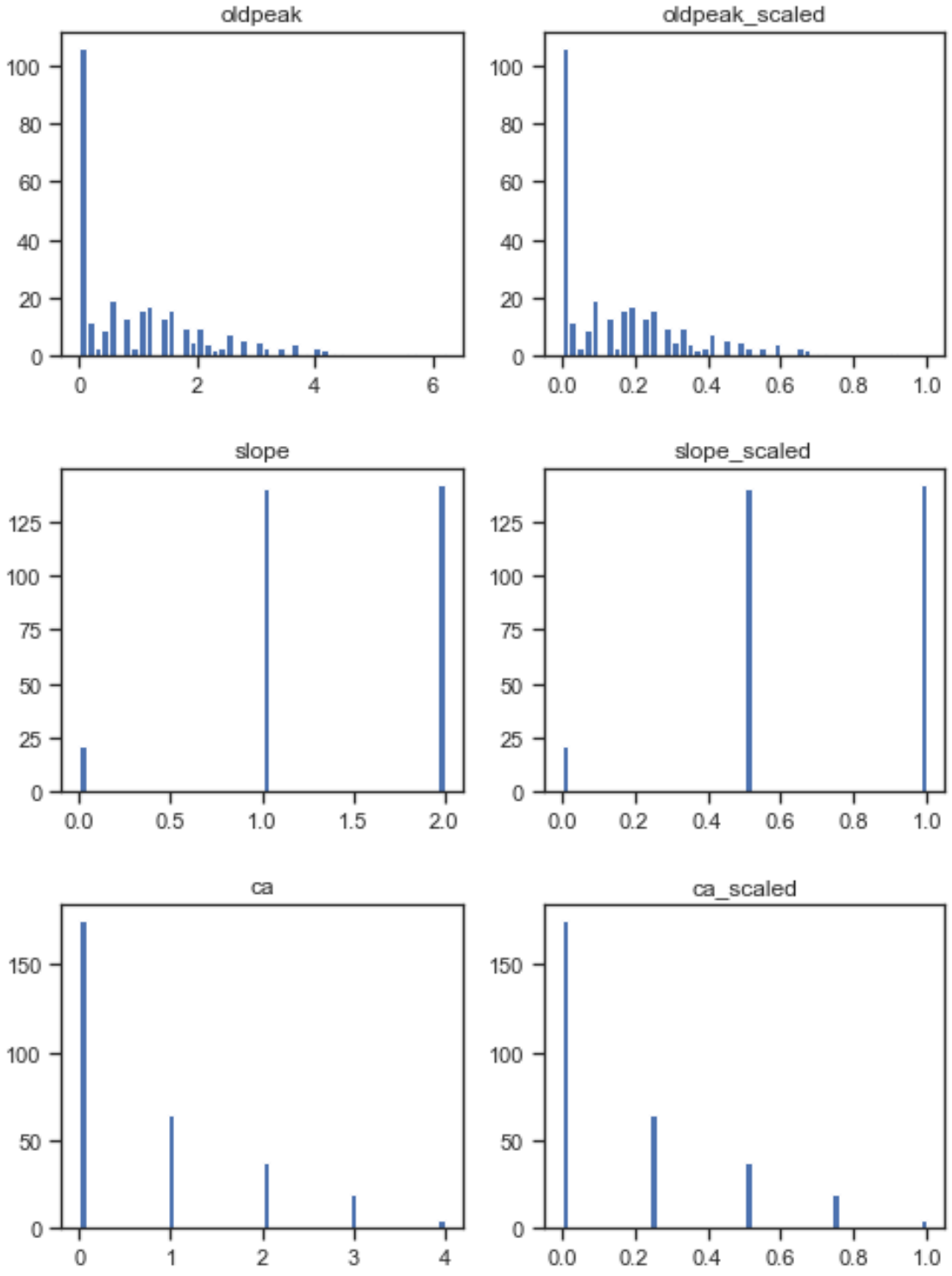
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  ...  \
0   58    1   0       114   318    0        2      140      0      4.4  ...
1   52    1   2       172   199    1        1      162      0      0.5  ...
2   53    1   0       142   226    0        0      111      1      0.0  ...
3   68    1   2       118   277    0        1      151      0      1.0  ...
4   62    1   2       130   231    0        1      146      0      1.8  ...

   trestbps_scaled  chol_scaled  fbs_scaled  restecg_scaled thalach_scaled  \
0         0.188679     0.438356         0.0             1.0       0.526718
1         0.735849     0.166667         1.0             0.5       0.694656
2         0.452830     0.228311         0.0             0.0       0.305344
3         0.226415     0.344749         0.0             0.5       0.610687
4         0.339623     0.239726         0.0             0.5       0.572519

   exang_scaled  oldpeak_scaled  slope_scaled  ca_scaled  thal_scaled
0           0.0        0.709677           0.0       0.75     0.333333
1           0.0        0.080645           1.0       0.00     1.000000
2           1.0        0.000000           1.0       0.00     1.000000
3           0.0        0.161290           1.0       0.25     1.000000
4           0.0        0.290323           0.5       0.75     1.000000

[5 rows x 28 columns]
```

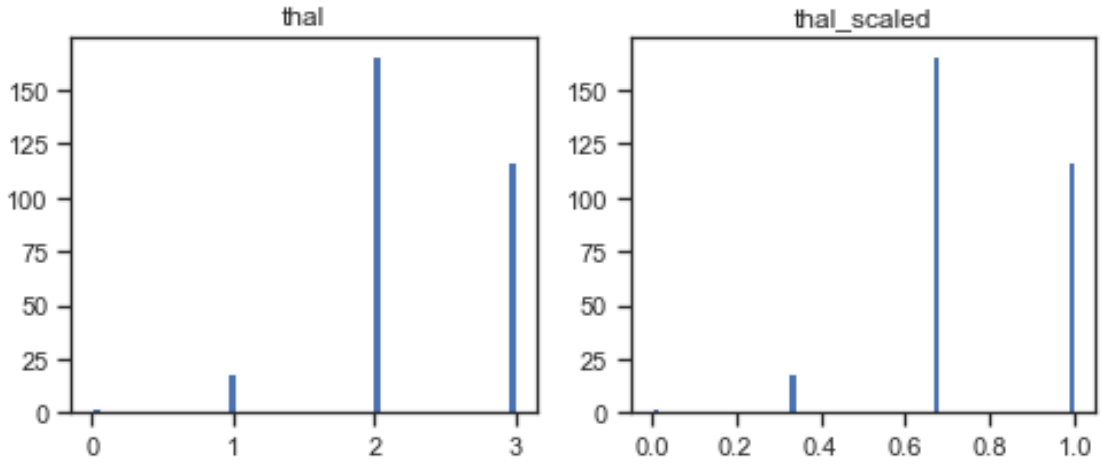
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

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


age
age_scaled
sex
sex_scaled
cp
cp_scaled


trestbps
trestbps_scaled
chol
chol_scaled
fbs
fbs_scaled


restecg
restecg_scaled
thalach
thalach_scaled
exang
exang_scaled


oldpeak
oldpeak_scaled
slope
slope_scaled
ca
ca_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 + ['target']
corr_cols_1
```

```
['age',
 'sex',
 'cp',
 'trestbps',
 'chol',
 'fbs',
 'restecg',
 'thalach',
 'exang',
 'oldpeak',
 'slope',
 'ca',
 'thal',
 'target']
```

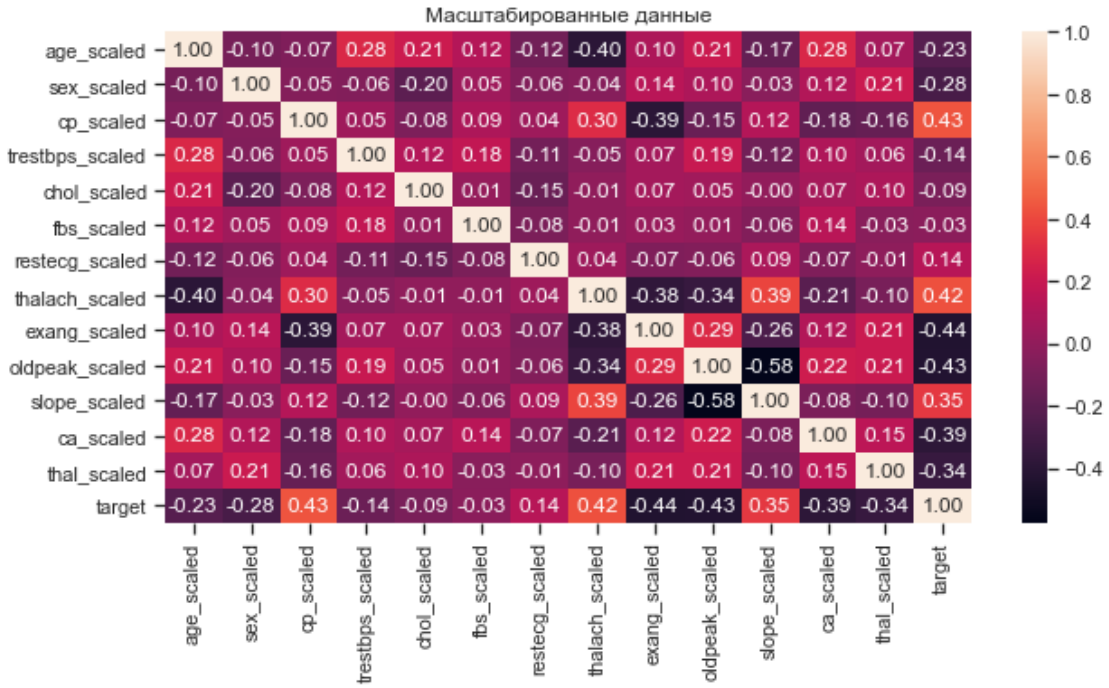
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['target']
corr_cols_2
```

```
['age_scaled',
 'sex_scaled',
 'cp_scaled',
 'trestbps_scaled',
 'chol_scaled',
 'fbs_scaled',
 'restecg_scaled',
 'thalach_scaled',
 'exang_scaled',
 'oldpeak_scaled',
```

```
 'slope_scaled',
 'ca_scaled',
 'thal_scaled',
 'target']
```

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data_all[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(data_all[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 бинарной классификации "target" наиболее сильно коррелирует с "cp" (0.43), "thalach" (0.42) и "slope" (0.35), а также 'exang' (-0.44), 'oldpeak'(-0.43), 'ca'(-0.39), 'thal'(-0.39). Эти признаки обязательно следует оставить в модели классификации.

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:

Можно переводить как точность, но такой перевод совпадает с переводом метрики "accuracy".

precision=TPTP+FP Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Используется функция precision_score.

Метрика recall (полнота):

recall=TPTP+FN Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция recall_score.

Метрика F1-мера

Для того, чтобы объединить precision и recall в единую метрику используется Fβ-мера, которая вычисляется как среднее гармоническое от precision и recall:

Fβ=(1+β2)·precision·recallprecision+recall где β определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при β=1:

F1=2·precision·recallprecision+recall Для вычисления используется функция f1_score.

Метрика ROC AUC

Основана на вычислении следующих характеристик:

TPR=TPTP+FN - True Positive Rate, откладывается по оси ординат. Совпадает с recall.

FPR=FPFP+TN - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

Идеальная ROC-кривая проходит через точки (0,0)-(0,1)-(1,1), то есть через верхний левый угол графика.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция roc_auc_score.

Сохранение и визуализация метрик

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества.

```
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()
```

Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.

Для задачи классификации будем использовать следующие модели:

Логистическая регрессия

Метод ближайших соседей

Машина опорных векторов

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
train_data_all = data_all[data_all['dataset']=='TRAIN']
test_data_all = data_all[data_all['dataset']=='TEST']
train_data_all.shape, test_data_all.shape
```

```
((243, 28), (60, 28))
```

```
# Признаки для задачи классификации
task_clas_cols = ['cp', 'thalach',
                  'slope','exang', 'oldpeak', 'ca', 'thal']

# Выборки для задачи классификации
clas_X_train = train_data_all[task_clas_cols]
clas_X_test = test_data_all[task_clas_cols]
clas_Y_train = train_data_all['target']
clas_Y_test = test_data_all['target']
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

```
((243, 7), (60, 7), (243,), (60,))
```

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.

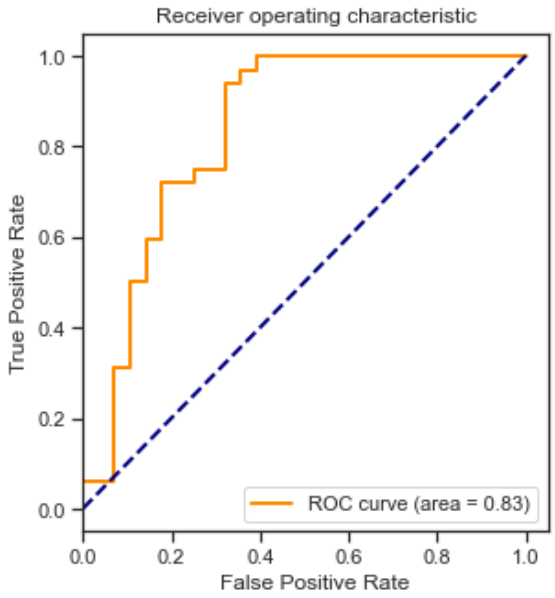
Решение задачи классификации

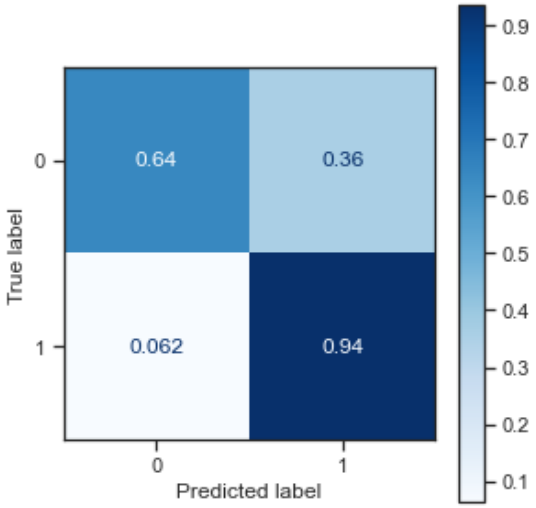
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_10':KNeighborsClassifier(n_neighbors=10),
               'SVC':SVC(probability=True),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}

# 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)
    #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(clas_X_train, clas_Y_train)
    # Предсказание значений
    Y_pred = model.predict(clas_X_test)
    # Предсказание вероятности класса "1" для roc auc
```

```
    Y_pred_proba_temp = model.predict_proba(clas_X_test)
    Y_pred_proba = Y_pred_proba_temp[:,1]

    precision = precision_score(clas_Y_test.values, Y_pred)
    recall = recall_score(clas_Y_test.values, Y_pred)
    f1 = f1_score(clas_Y_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y_test.values, Y_pred_proba)

    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    clasMetricLogger.add('roc_auc', model_name, roc_auc)

    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(clas_Y_test.values, Y_pred_proba, ax[0])
    plot_confusion_matrix(model, clas_X_test, clas_Y_test.values, ax=ax[1],
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    fig.suptitle(model_name)
    plt.show()

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

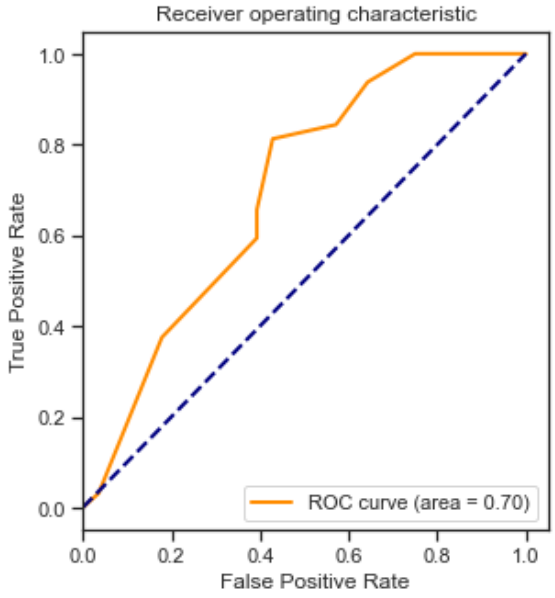
```
/home/zeus/anaconda3/envs/tml_env/lib/python3.9/site-packages/sklearn/linear_mod
el/_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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-regressio
n
  n_iter_i = _check_optimize_result(
```

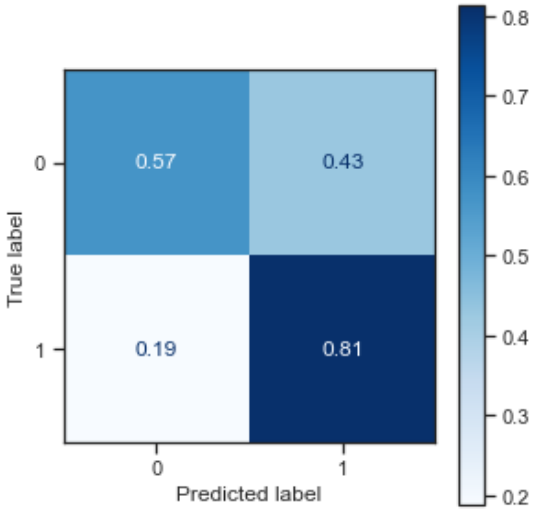
LogR


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.83)
0.0
0.2
0.4
0.6
0.8
1.0

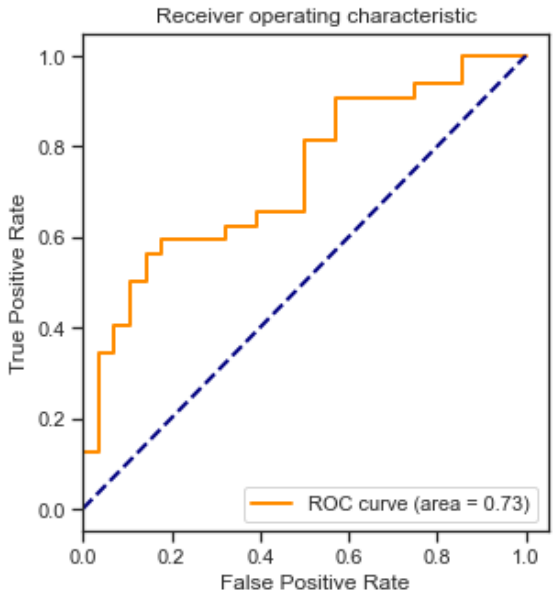

0.64
0.36
0.062
0.94
True label
Predicted label
0
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

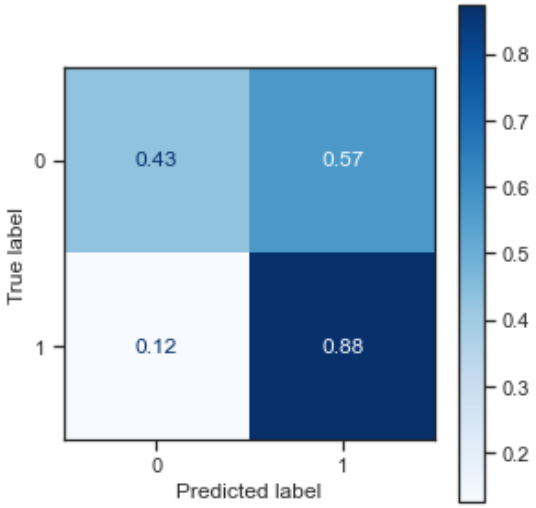
KNN_10


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.70)
0.0
0.2
0.4
0.6
0.8
1.0

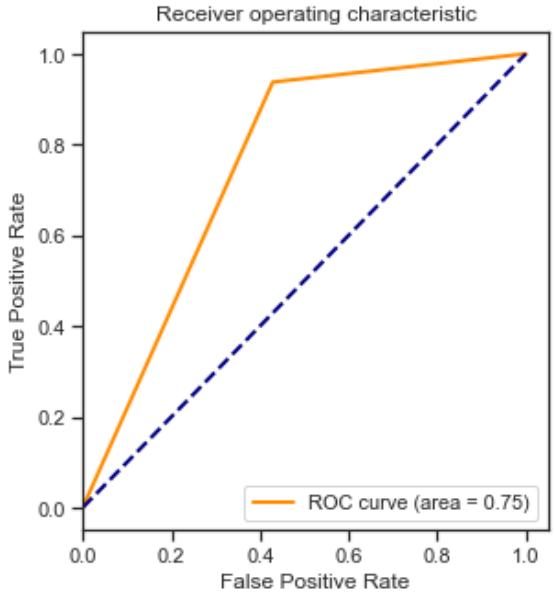

0.57
0.43
0.19
0.81
True label
Predicted label
0
1
0.8
0.7
0.6
0.5
0.4
0.3
0.2

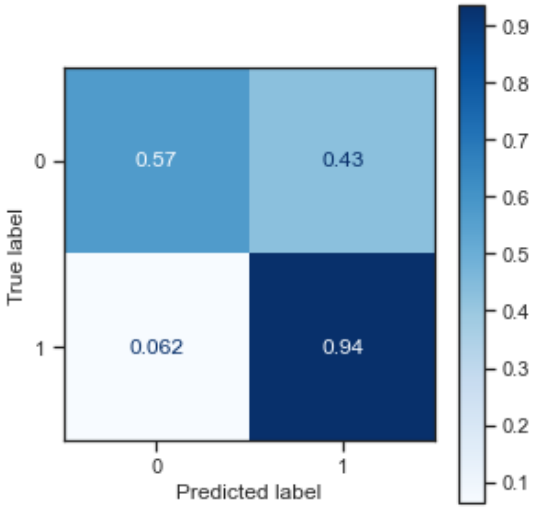
SVC


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.73)
0.0
0.2
0.4
0.6
0.8
1.0

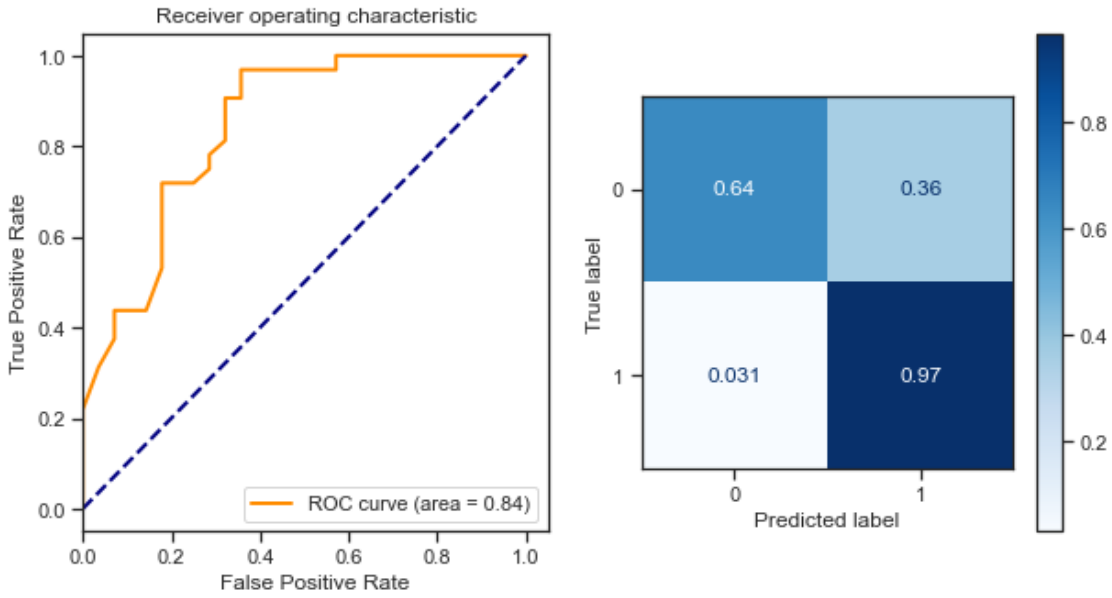

0.43
0.57
0.12
0.88
True label
Predicted label
0
1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

Tree

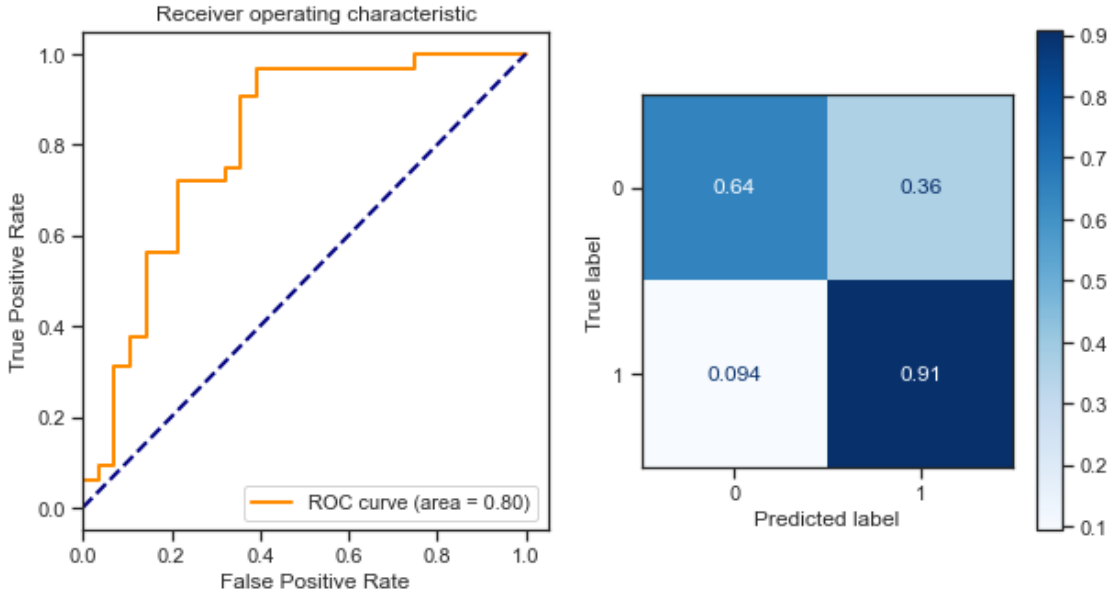

Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.75)
0.0
0.2
0.4
0.6
0.8
1.0


0.57
0.43
0.062
0.94
True label
Predicted label
0
1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

RF



GB



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
```

```
(243, 7)
```

```
n_range_list = list(range(0,50,2))
n_range_list[0] = 1
```

```
n_range = np.array(n_range_list)
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 1,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26,
28, 30, 32,
        34, 36, 38, 40, 42, 44, 46, 48])}]
```

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

```
CPU times: user 822 ms, sys: 0 ns, total: 822 ms
Wall time: 820 ms
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 1,  2,  4,  6,  8, 10, 12, 14,
16, 18, 20, 22, 24, 26, 28, 30, 32,
       34, 36, 38, 40, 42, 44, 46, 48])}],
             scoring='roc_auc')
```

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

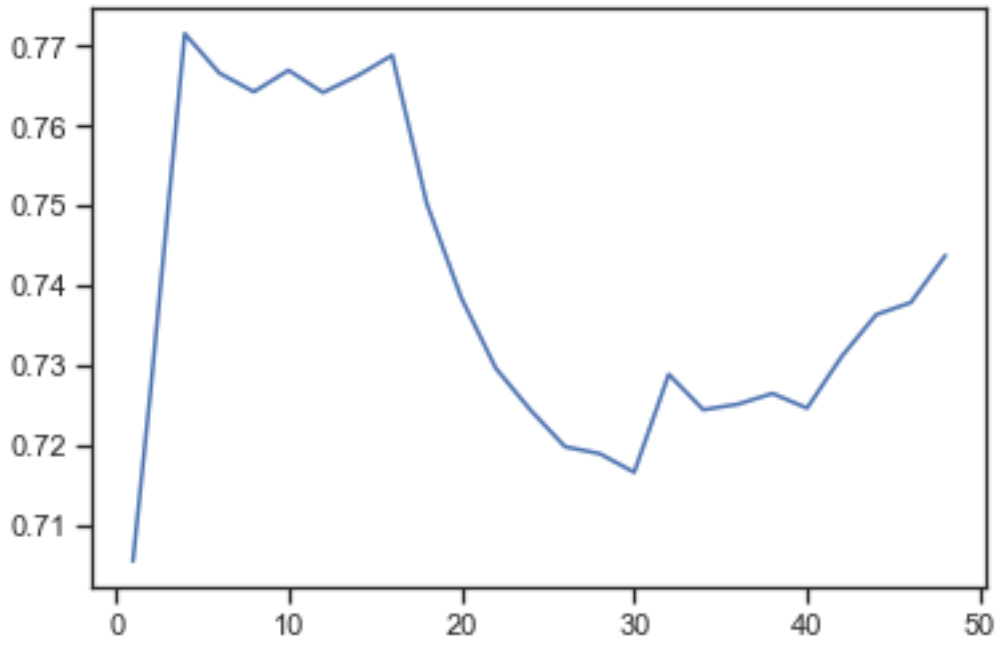
```
KNeighborsClassifier(n_neighbors=4)
```

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

```
'4'
```

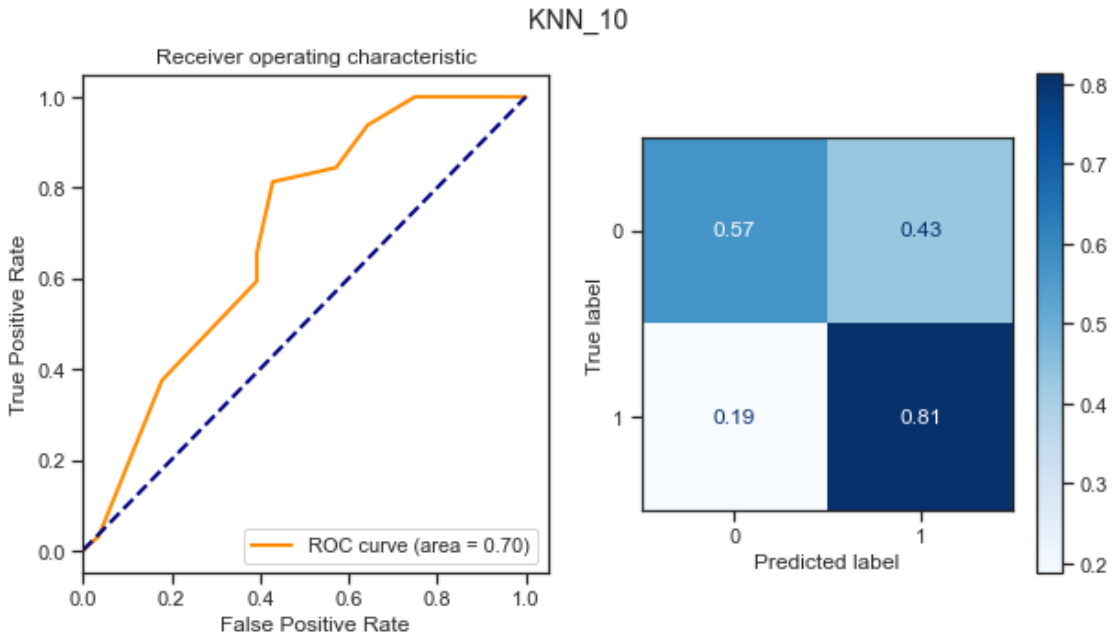
```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

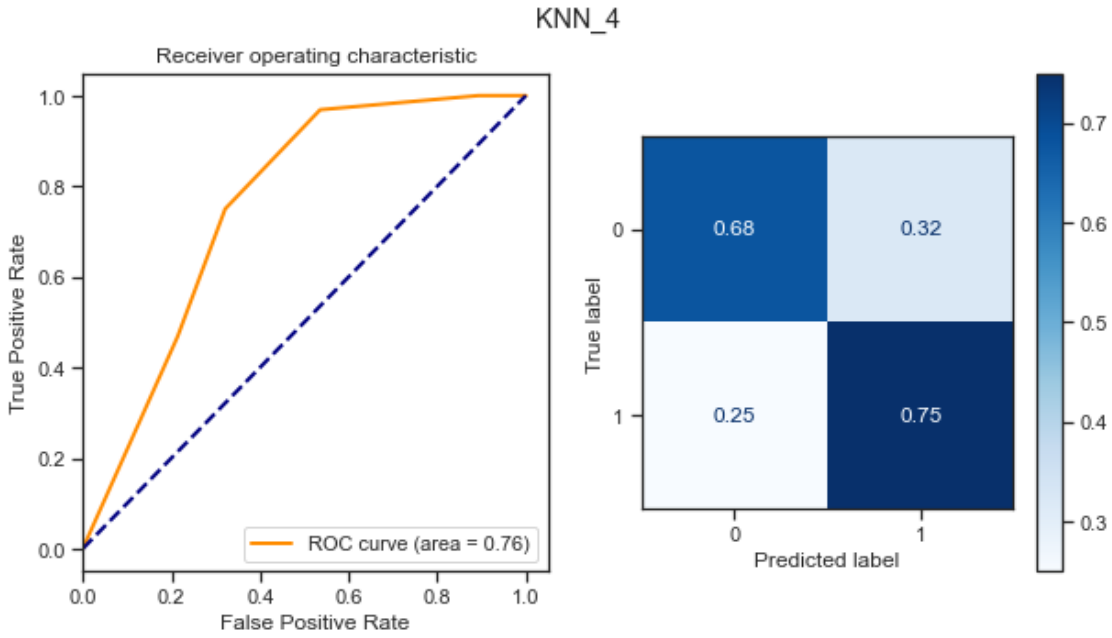
```
[<matplotlib.lines.Line2D at 0x7f8f0541ba00>]
```



Повторение пункта 8 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)
```



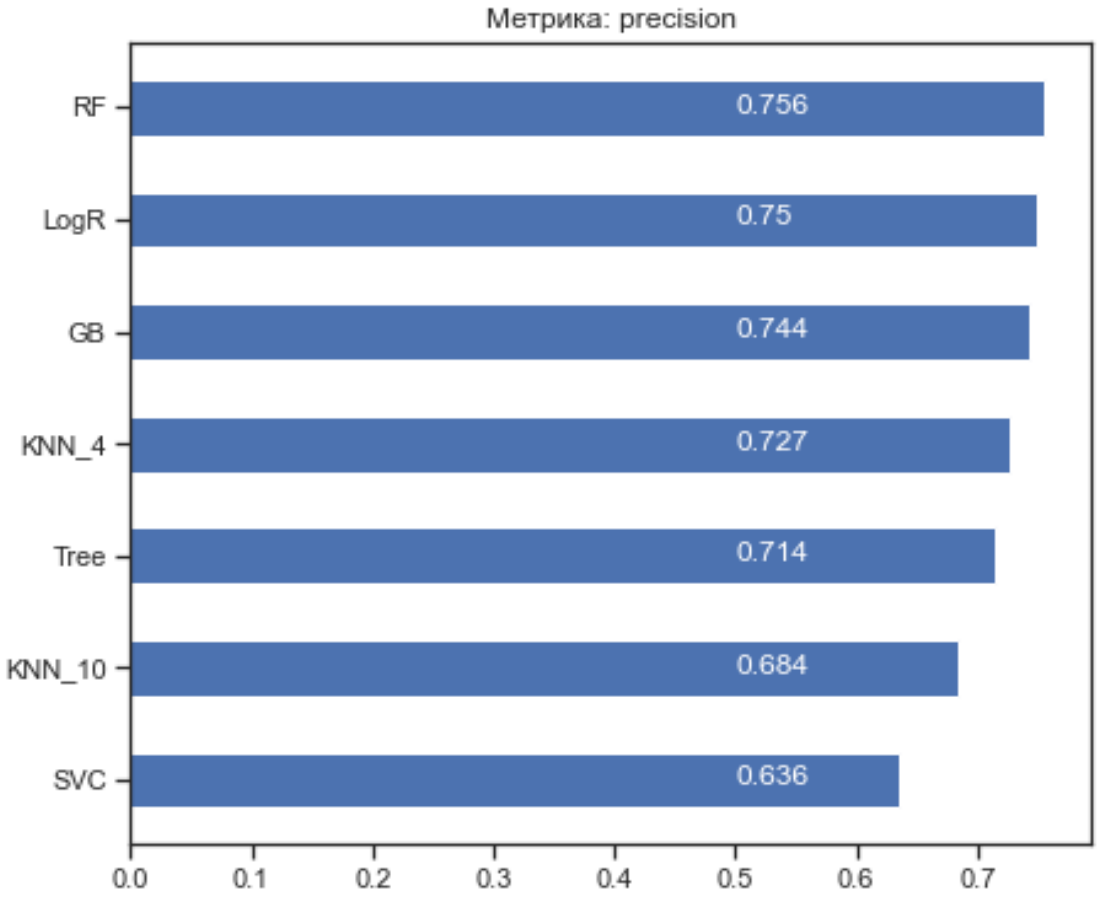


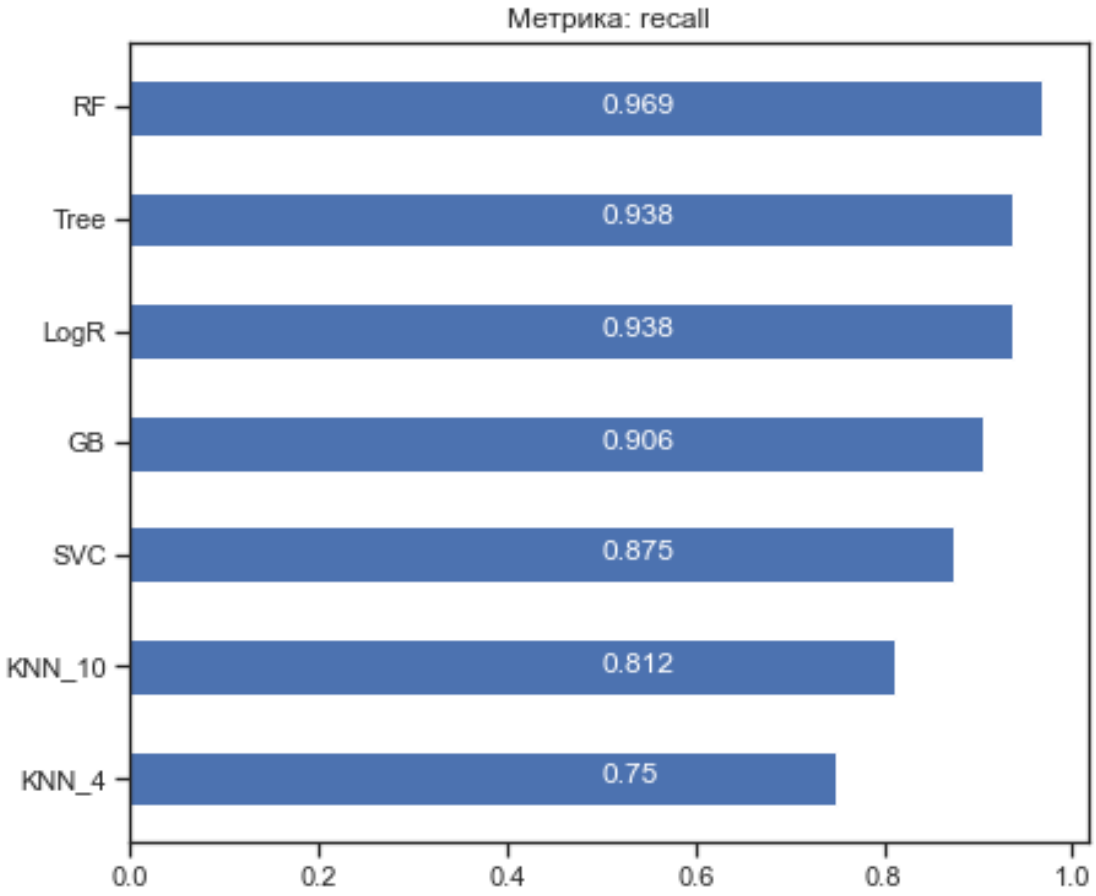
Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей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

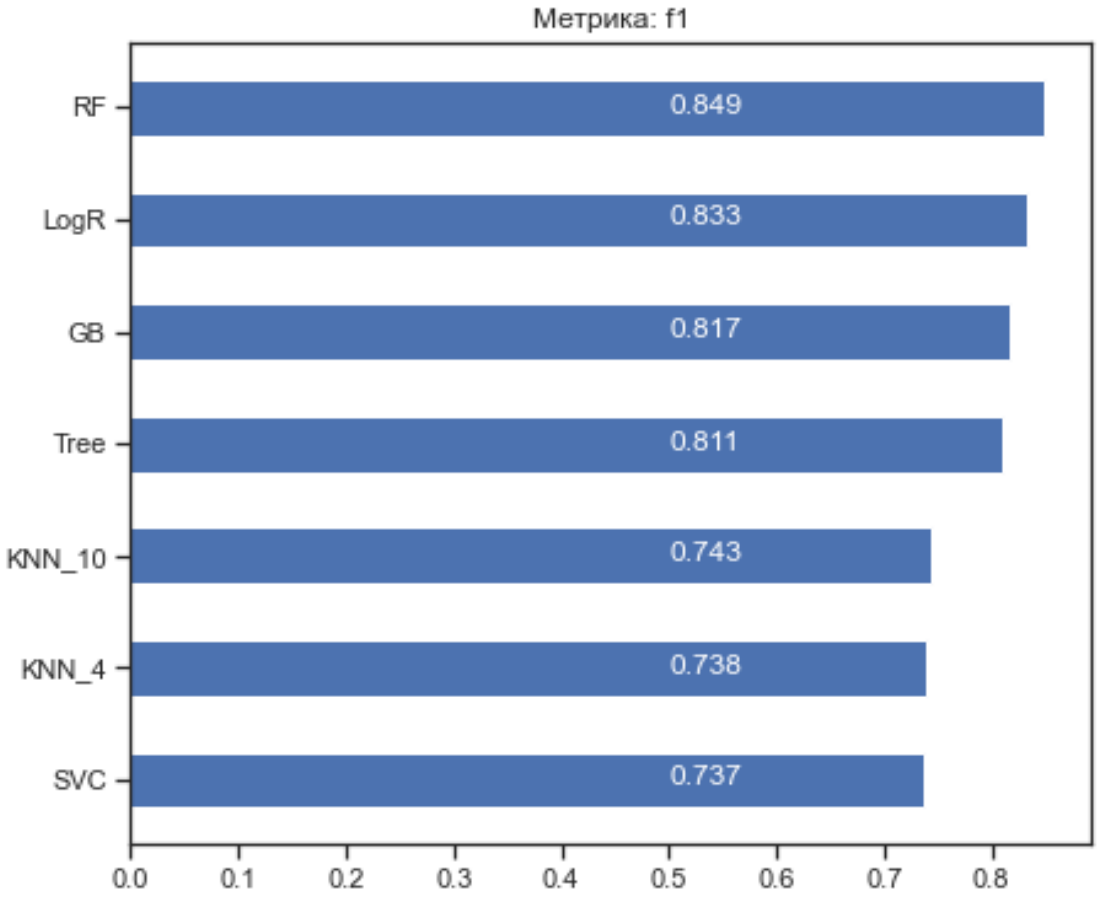
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

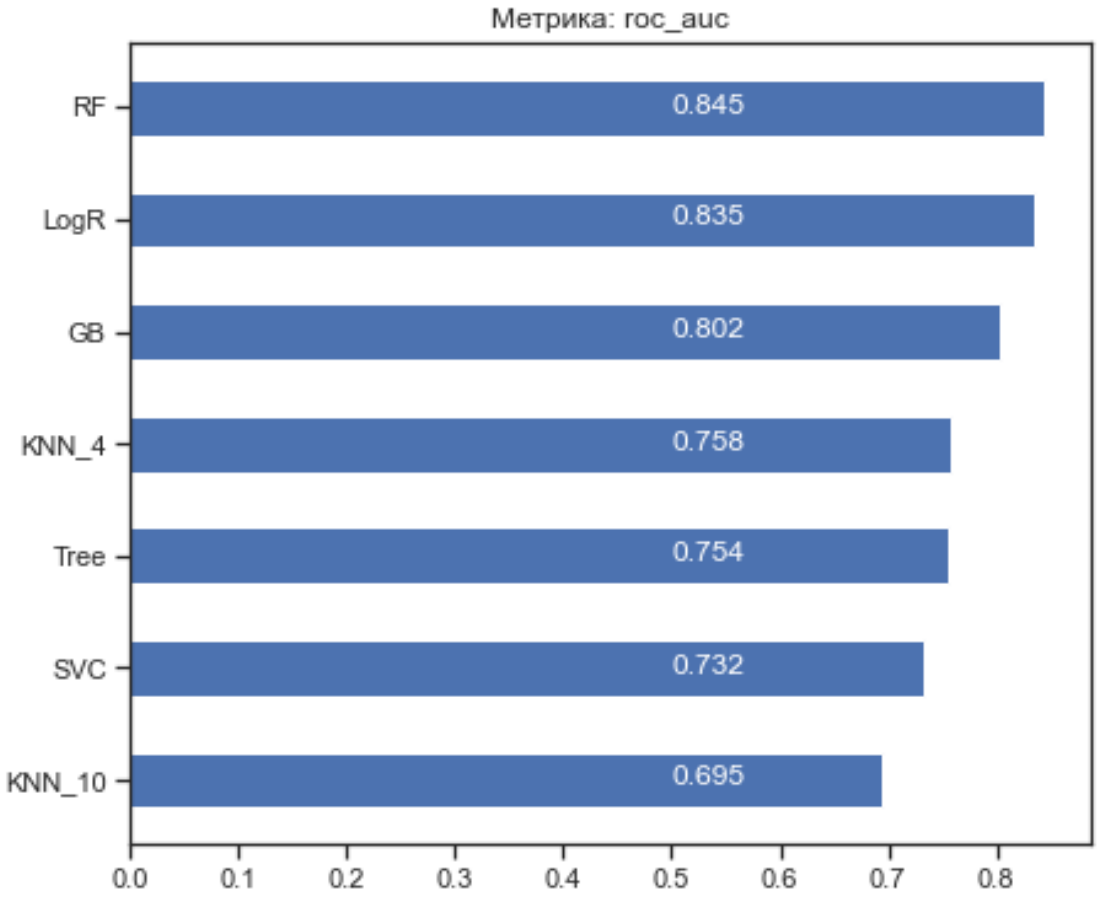
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

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


Метрика: precision
RF
LogR
GB
KNN_4
Tree
KNN_10
SVC
0.756
0.75
0.744
0.727
0.714
0.684
0.636
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Метрика: recall
RF
Tree
LogR
GB
SVC
KNN_10
KNN_4
0.969
0.938
0.938
0.906
0.875
0.812
0.75
0.0
0.2
0.4
0.6
0.8
1.0


Метрика: f1
RF
LogR
GB
Tree
KNN_10
KNN_4
SVC
0.849
0.833
0.817
0.811
0.743
0.738
0.737
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8



Вывод: на основании трех метрик из четырех используемых, лучшей оказалась модель случайного леса