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 (ИУ 5)

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Прогнозирование рака молочной железы

Студент ИУ5-63Б _____ **С. А. Некрасов**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель _____ **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_____Прогнозирование рака молочной железы_____

_____

_____

Студент группы _ИУ5-63Б____

_____Некрасов Сергей Андреевич_____
(Фамилия, имя, отчество)

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

_____учебная_____

Источник тематики (кафедра, предприятие, НИР)_____кафедра_____

График выполнения НИР: 25% к____нед., 50% к____нед., 75% к___нед., 100% к____нед.

Техническое задание *решить задачу бинарной классификации на основе материалов дисциплины по выбранной предметной области*_____

_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 31 листе формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

_____

_____

_____

Дата выдачи задания « _15_ »____февраля___2023 г.

Руководитель НИР _____ ____Ю. Е. Гапанюк____
(Подпись, дата) (И.О.Фамилия)

Студент _____ ____С. А. Некрасов___
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

Введение

В данном курсовом проекте предстоит выполнить типовую задачу машинного обучения - провести анализ данных, провести некоторые операции с датасетом, подобрать модели, а также подобрать наиболее подходящие гиперпараметры выбранных моделей. Машинное обучение очень актуально в современном мире, оно используется практически во многих сферах. Программист должен подбирать подходящие технологии машинного обучения для достижения наилучших результатов. Чему мы и научимся в этом курсовом проекте. Попробуем не менее пяти видов различных моделей и подберем наилучшую из них на основе выбранных метрик. Также построим вспомогательные графики, которые помогут нам визуально взглянуть на все необходимые показатели.

Задание

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.

Описание датасета

Ссылка на датасет: https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset

1. id: уникальный идентификатор.
2. diagnosis: M - злокачественная B - доброкачественная (целевой признак).
3. radius_mean: радиус долей.
4. texture_mean: среднее значение текстуры поверхности.
5. perimeter_mean: внешний периметр долей.
6. area_mean: средняя площадь долей.
7. smoothness_mean: среднее значение уровней гладкости.
8. compactness_mean: среднее значение компактности.
9. concavity_mean: среднее значение вогнутости.
10. concave points_mean: среднее значение вогнутых точек.
11. symmetry_mean: среднее значение симметрии.
12. fractal_dimension_mean: среднее значение фрактальной размерности.

Использовать будем только вышеперечисленные столбцы.

Импорт библиотек

In [86]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import random
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.metrics import plot_confusion_matrix
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
%matplotlib inline
sns.set(style="ticks")
import warnings
warnings.filterwarnings('ignore')
```

Загрузка данных

In [87]:
```
#first_data = pd.read_csv('healthcare-dataset-stroke-data.csv')
first_data = pd.read_csv('datasets/breast-cancer.csv')
```

In [88]:
```
# Удалим дубликаты записей, если они присутствуют
data = first_data.drop_duplicates()
# Также удалим ненужный столбец-идентификатор
data = data.drop(columns=['id'], axis=1)
# Оставим только медианные значения
data = data.loc[:, 'diagnosis':'fractal_dimension_mean']
```

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

In [89]:
```
data.head()
```

Out[89]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

In [90]:
```
data.shape
```

Out[90]:
```
(569, 11)
```

In [91]:
```
# Список колонок
data.columns
```

Out[91]:
```
Index(['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean'],
      dtype='object')
```

In [92]:
```
# Список колонок с типами данных
data.dtypes
```

Out[92]:
```
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
dtype: object
```

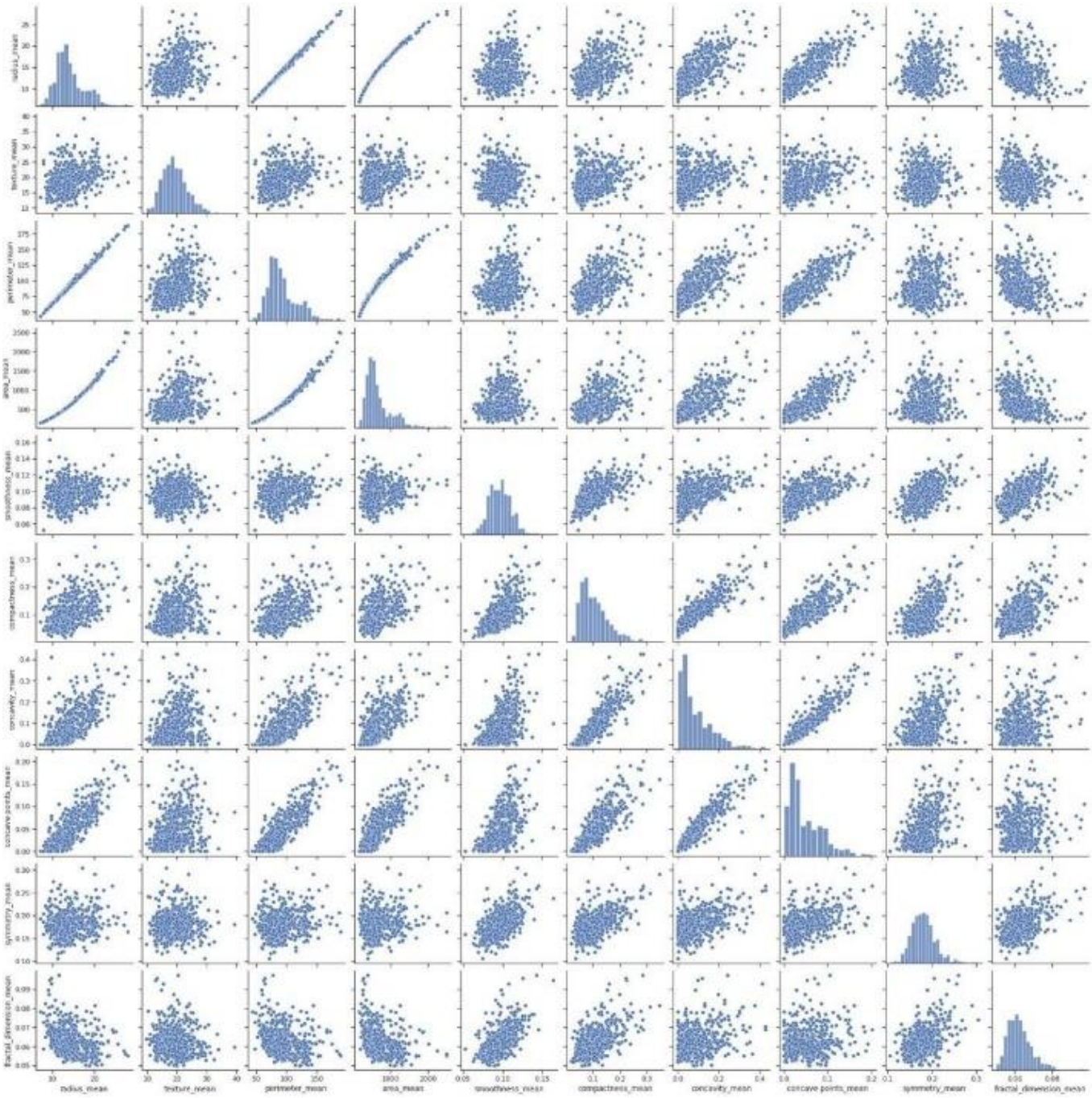
In [93]:
```
# Проверим наличие пустых значений
data.isnull().sum()
```

Out[93]:
```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
dtype: int64
```

Построение графиков для понимания структуры данных

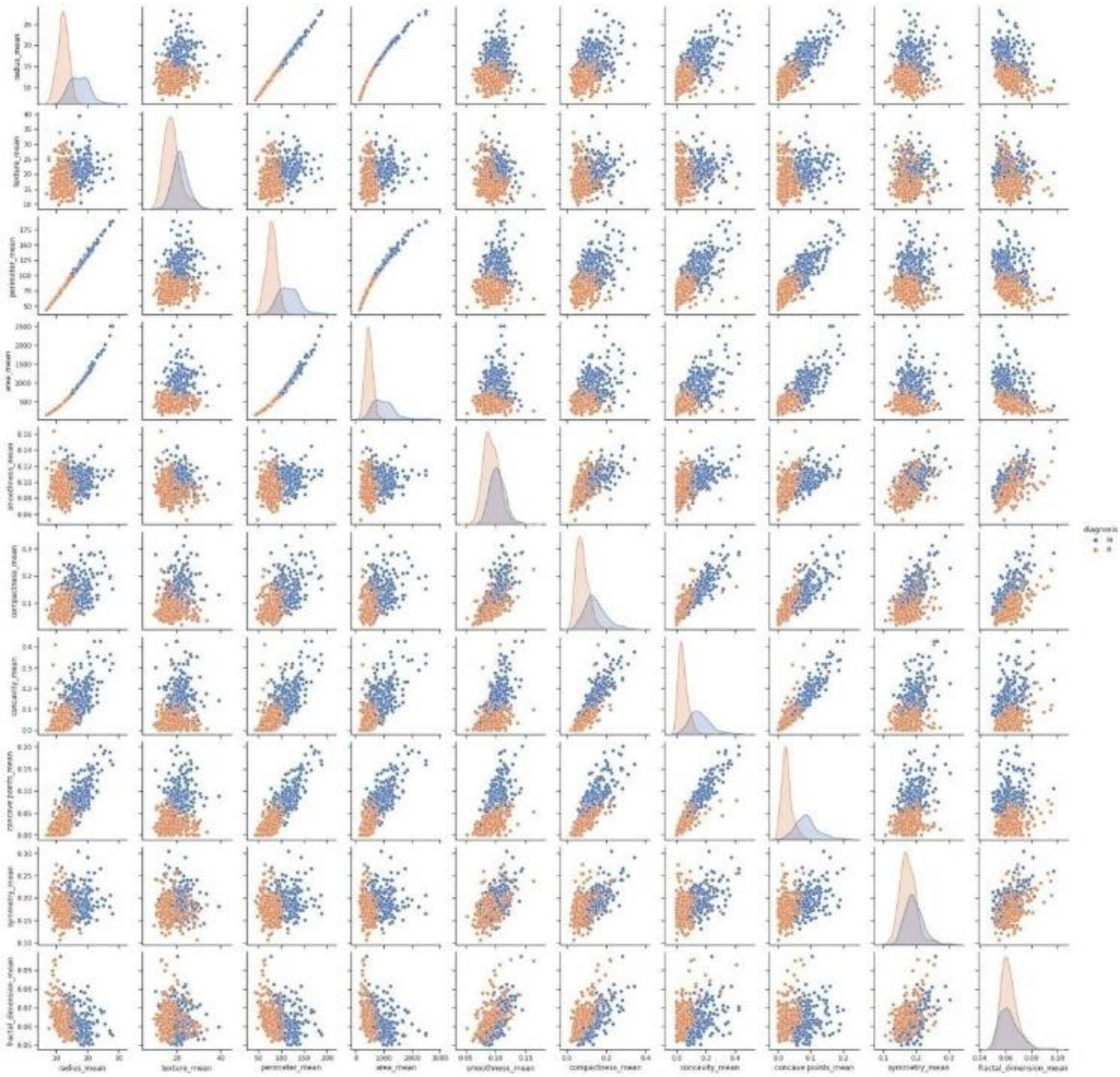
In [94]:
```
# Парные диаграммы
sns.pairplot(data)
```

Out[94]: <seaborn.axisgrid.PairGrid at 0x7f8fc85e1f70>



In [95]:
```
sns.pairplot(data, hue="diagnosis")
```

Out[95]: <seaborn.axisgrid.PairGrid at 0x7f8fcba43970>



In [96]:
```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
data['diagnosis'].unique()
```

Out[96]:
```
array(['M', 'B'], dtype=object)
```

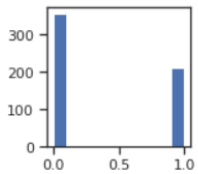
In [97]:
```
diagnosis = LabelEncoder()
code_diagnosis = diagnosis.fit_transform(data["diagnosis"])
data["diagnosis"] = code_diagnosis
data = data.astype({"diagnosis":"int64"})
np.unique(code_diagnosis)
```

Out[97]:
```
array([0, 1])
```

In [98]:
```
data['diagnosis'].unique()
```

Out[98]:
```
array([1, 0])
```

In [99]:
```
# Оценим дисбаланс классов для stroke
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['diagnosis'])
plt.show()
```



In [100]:
```
data['diagnosis'].value_counts()
```

Out[100]:
```
0    357
1    212
Name: diagnosis, dtype: int64
```

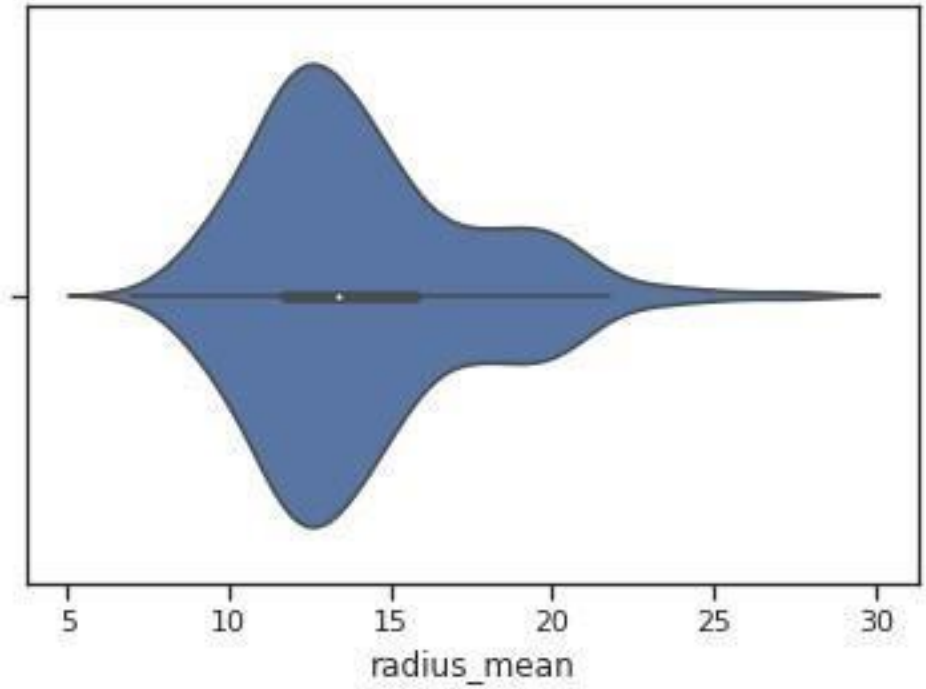
In [101]:
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['diagnosis'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

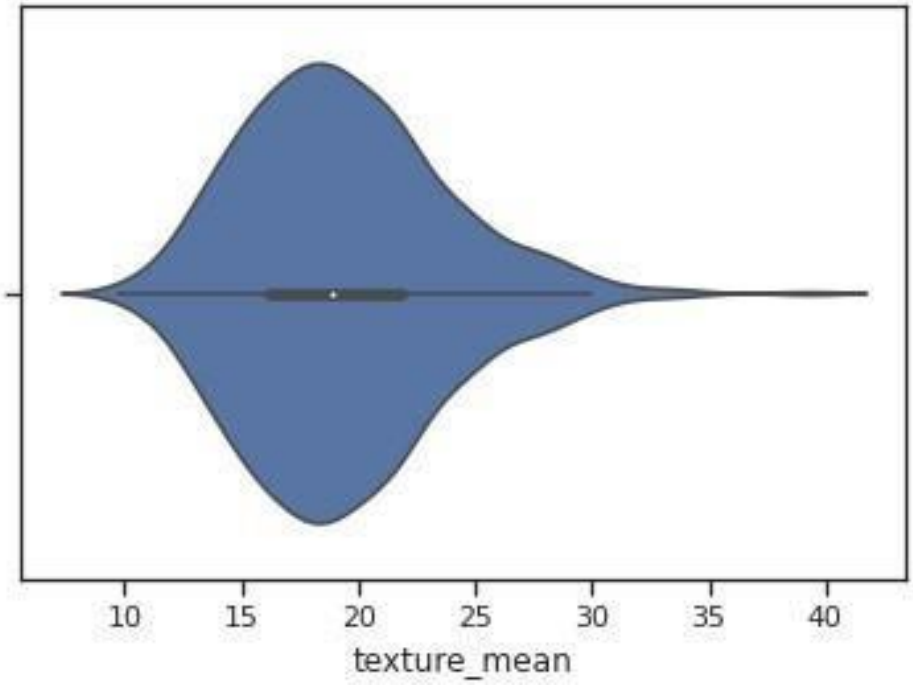
```
Класс 0 составляет 62.739999999999995%, а класс 1 составляет 37.26%.
```

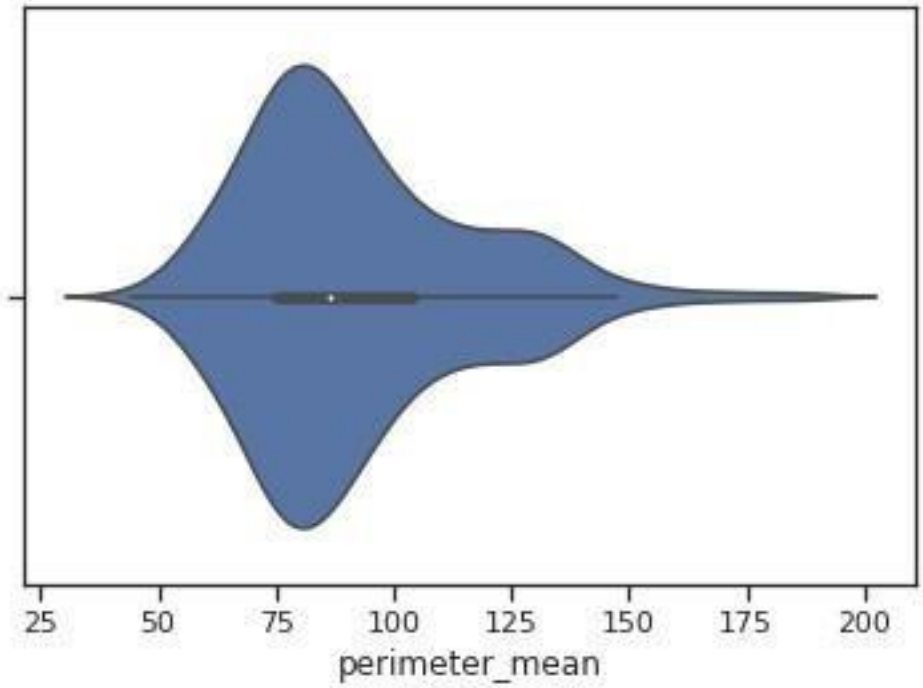
Присутствует незначительный дисбаланс классов.

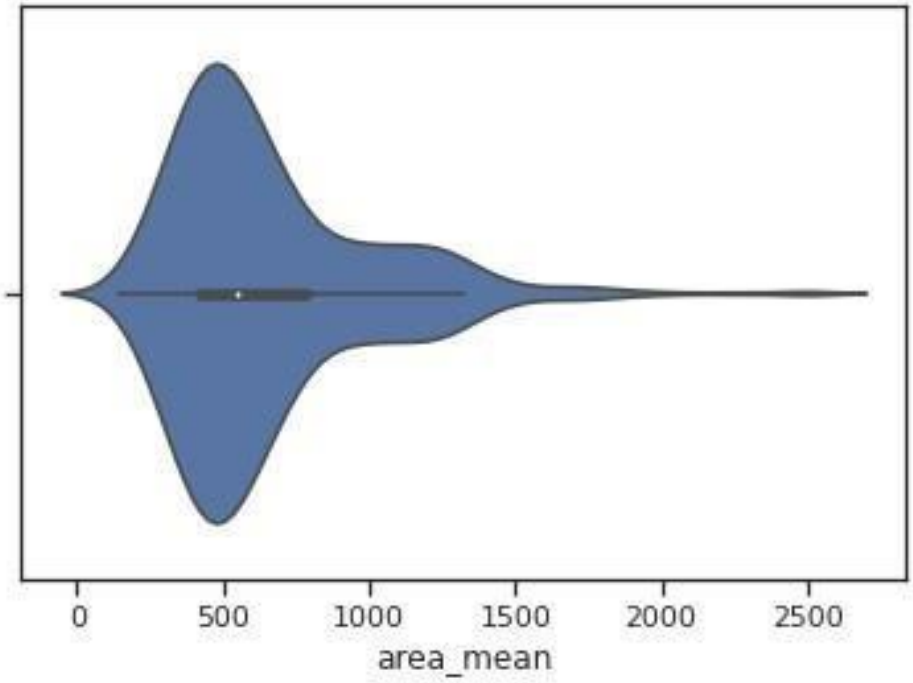
In [104]:
```
def upsample(features, target, repeat):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]

    features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
    target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)

    return features_upsampled, target_upsampled
```

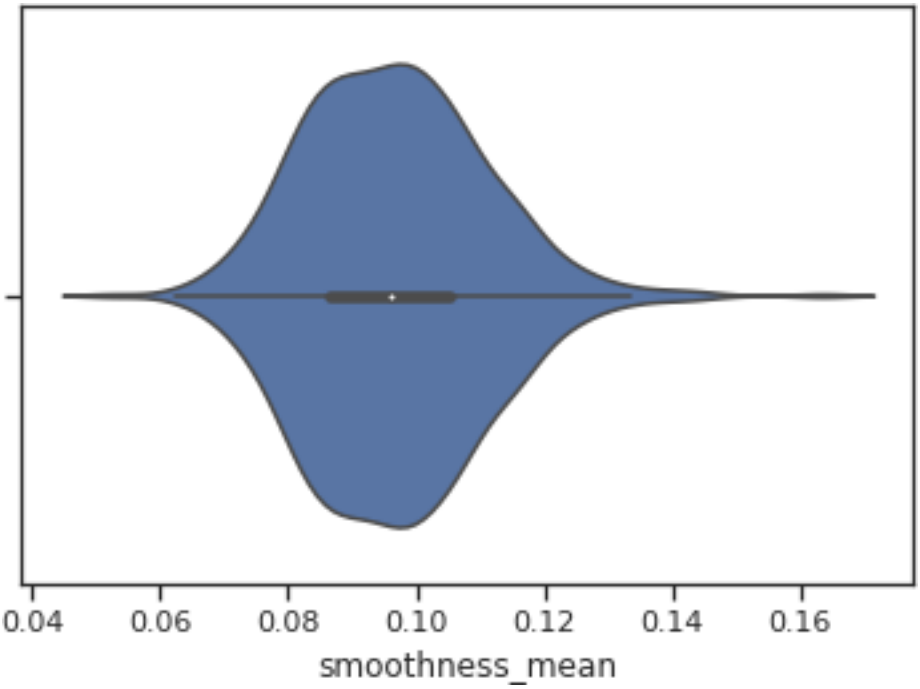
In [105]:
```
# Скрипичные диаграммы для числовых колонок
for col in ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']:
    sns.violinplot(x=data[col])
    plt.show()
```

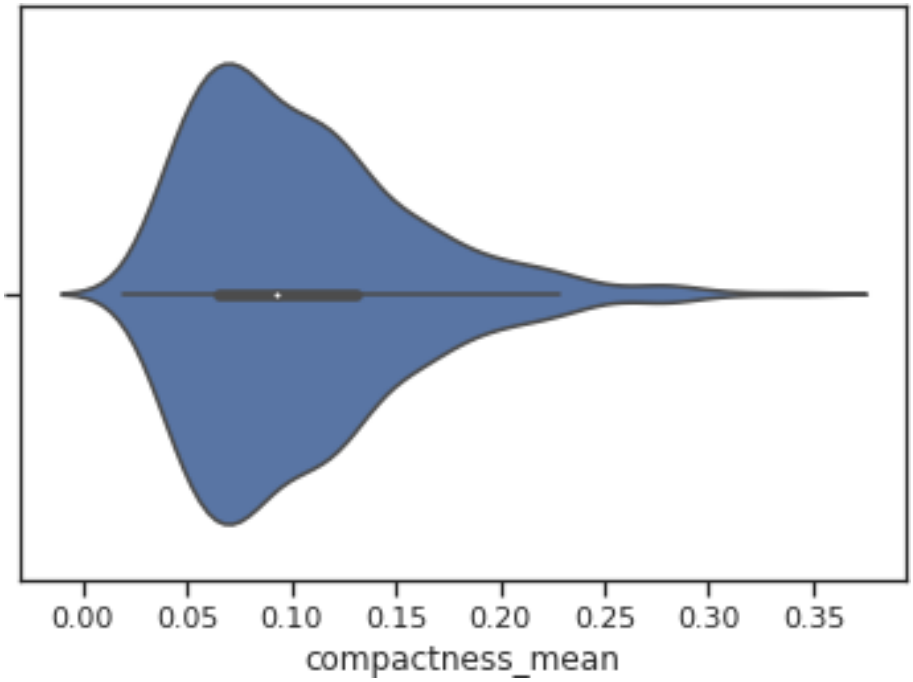

5
10
15
20
25
30
radius_mean

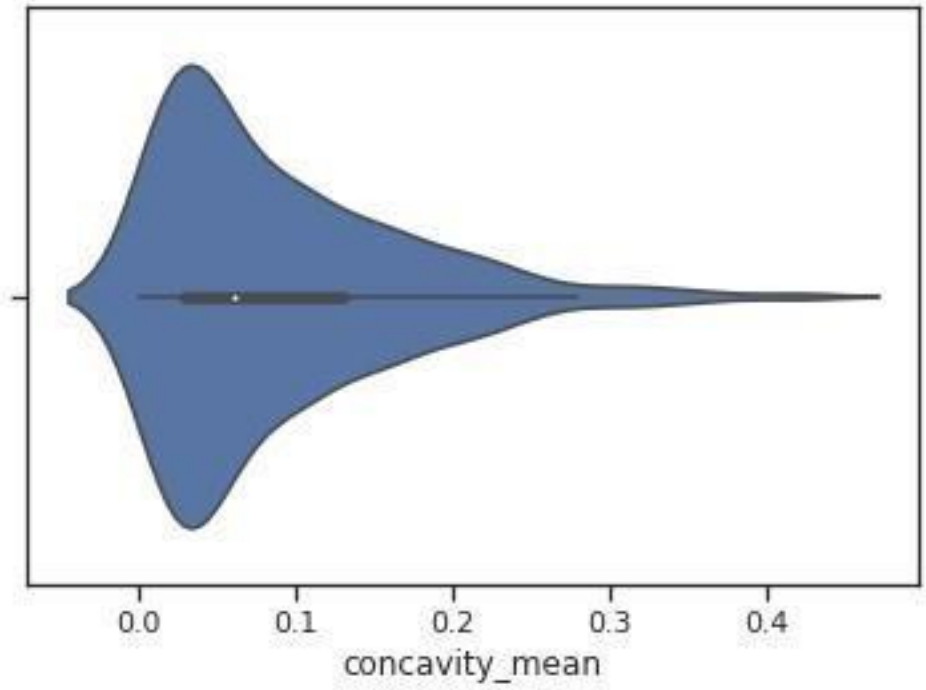

10
15
20
25
30
35
40
texture_mean

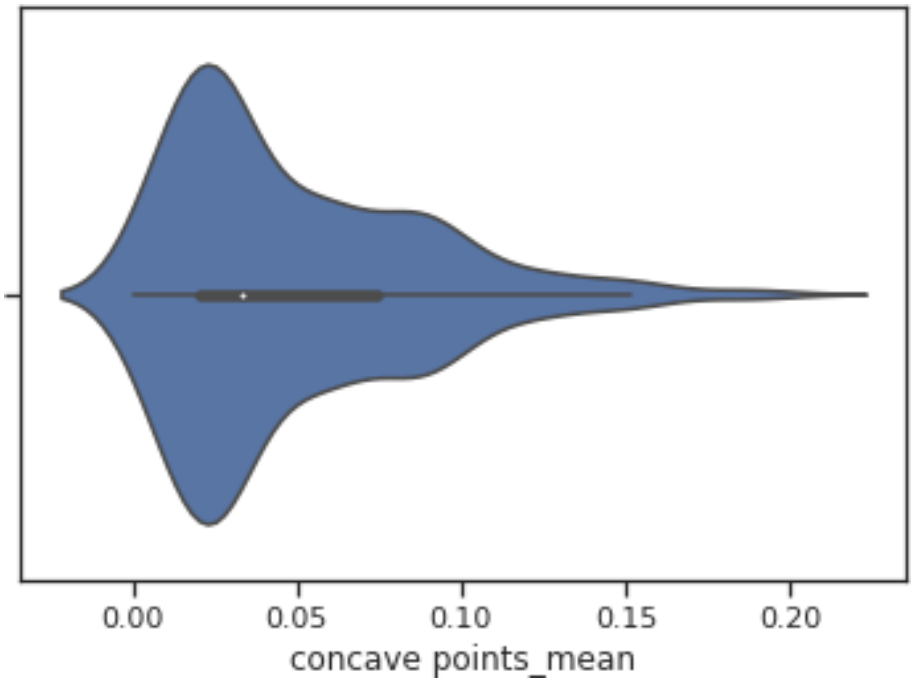

25
50
75
100
125
150
175
200
perimeter_mean

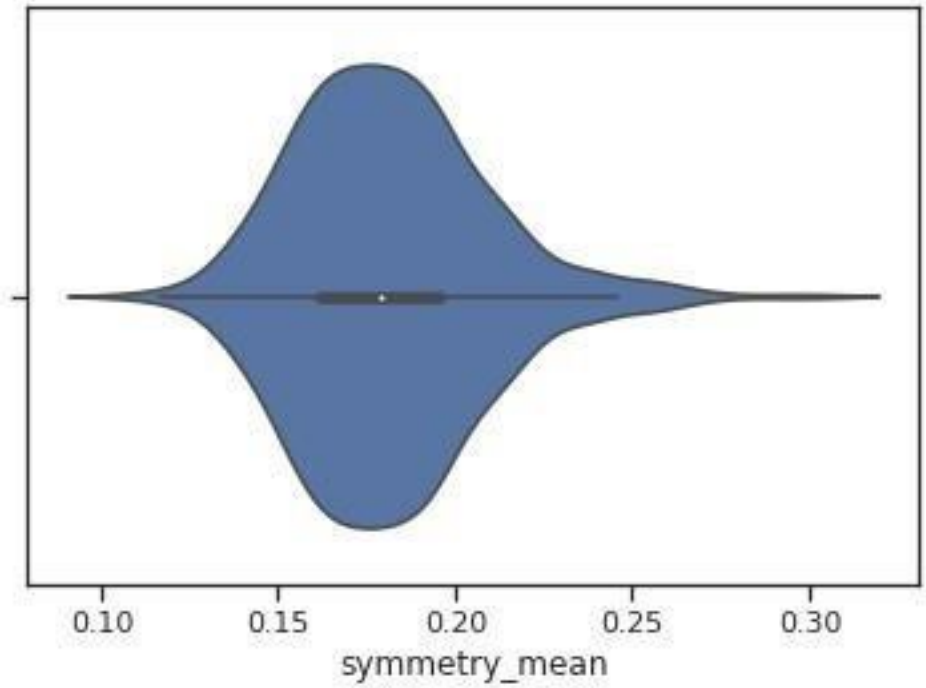

0
500
1000
1500
2000
2500
area_mean

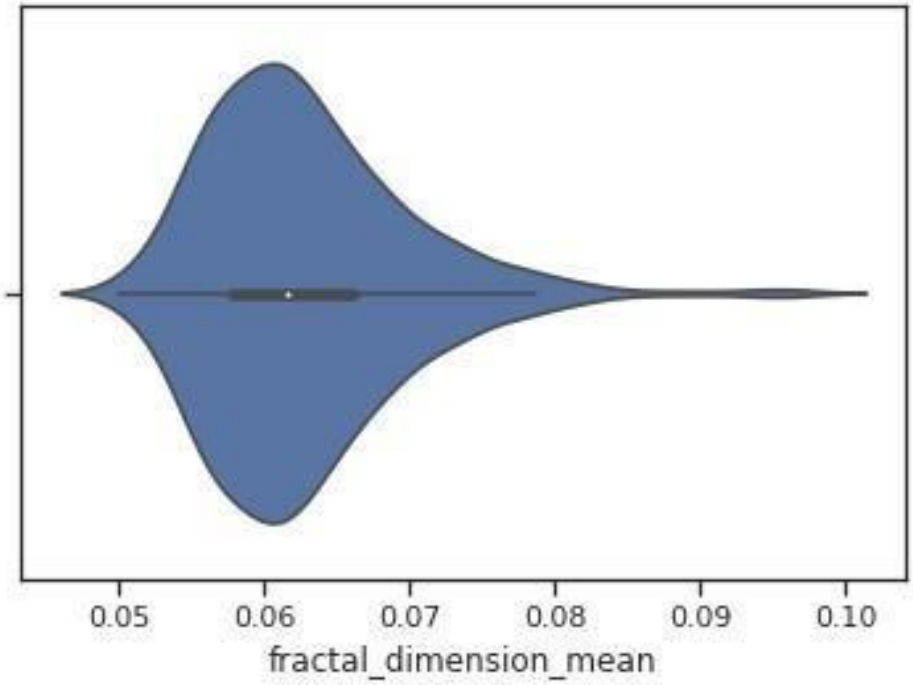

0.04
0.06
0.08
0.10
0.12
0.14
0.16
smoothness_mean


0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
compactness_mean


0.0
0.1
0.2
0.3
0.4
concavity_mean


0.00
0.05
0.10
0.15
0.20
concave points_mean


0.10
0.15
0.20
0.25
0.30
symmetry_mean


0.05
0.06
0.07
0.08
0.09
0.10
fractal_dimension_mean

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

In [106]:
```
data.dtypes
```

Out[106]:
```
diagnosis                 int64
radius_mean             float64
texture_mean            float64
perimeter_mean          float64
area_mean               float64
smoothness_mean         float64
compactness_mean        float64
concavity_mean          float64
concave points_mean     float64
symmetry_mean           float64
fractal_dimension_mean  float64
dtype: object
```

Категориальный признак "diagnosis" был закодирован ранее, другие категориальные признаки отсутствуют.

In [107]:
```
# Числовые колонки для масштабирования
scale_cols = ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']
```

In [108]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

In [109]:
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

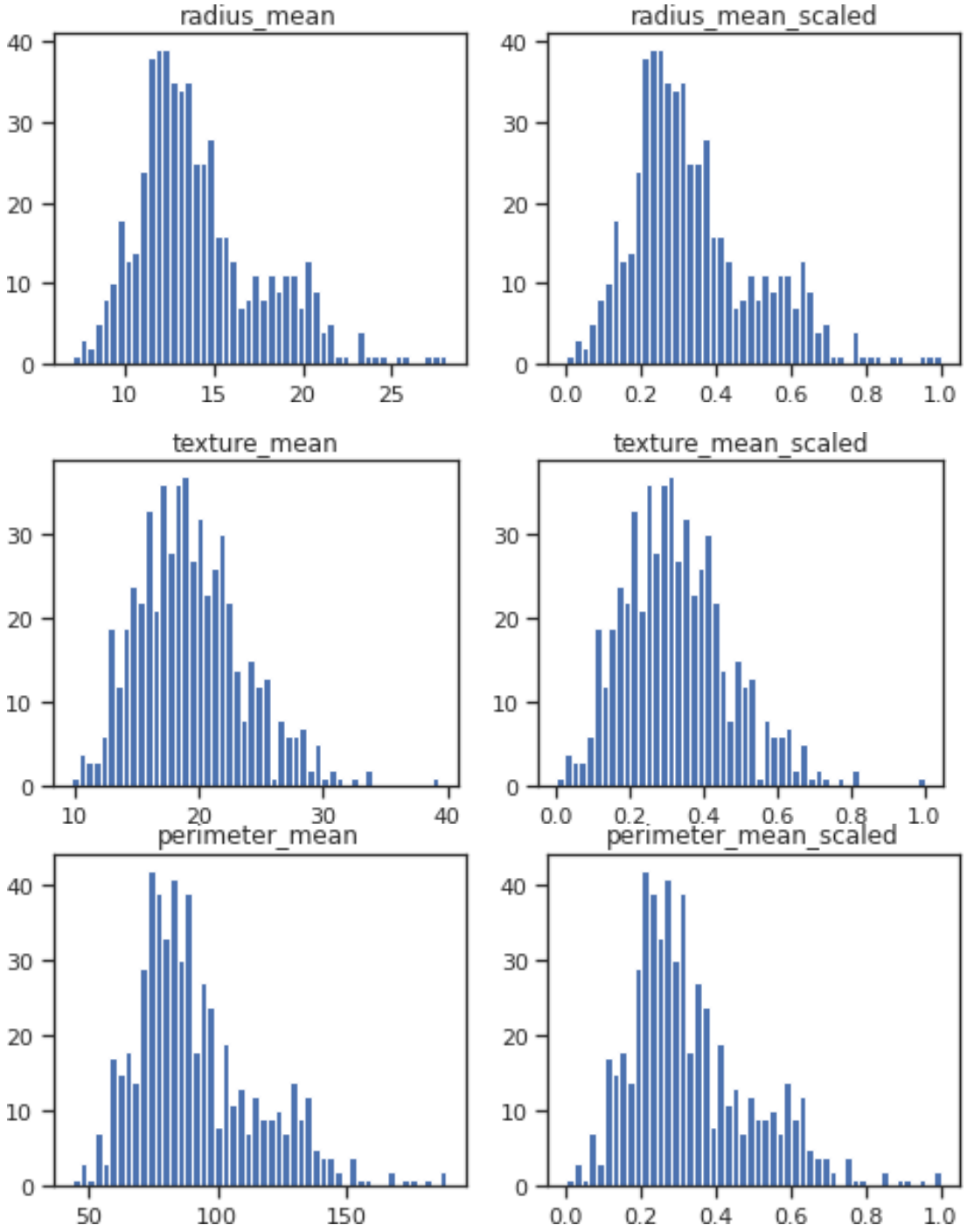
In [110]:
```
data.head()
```

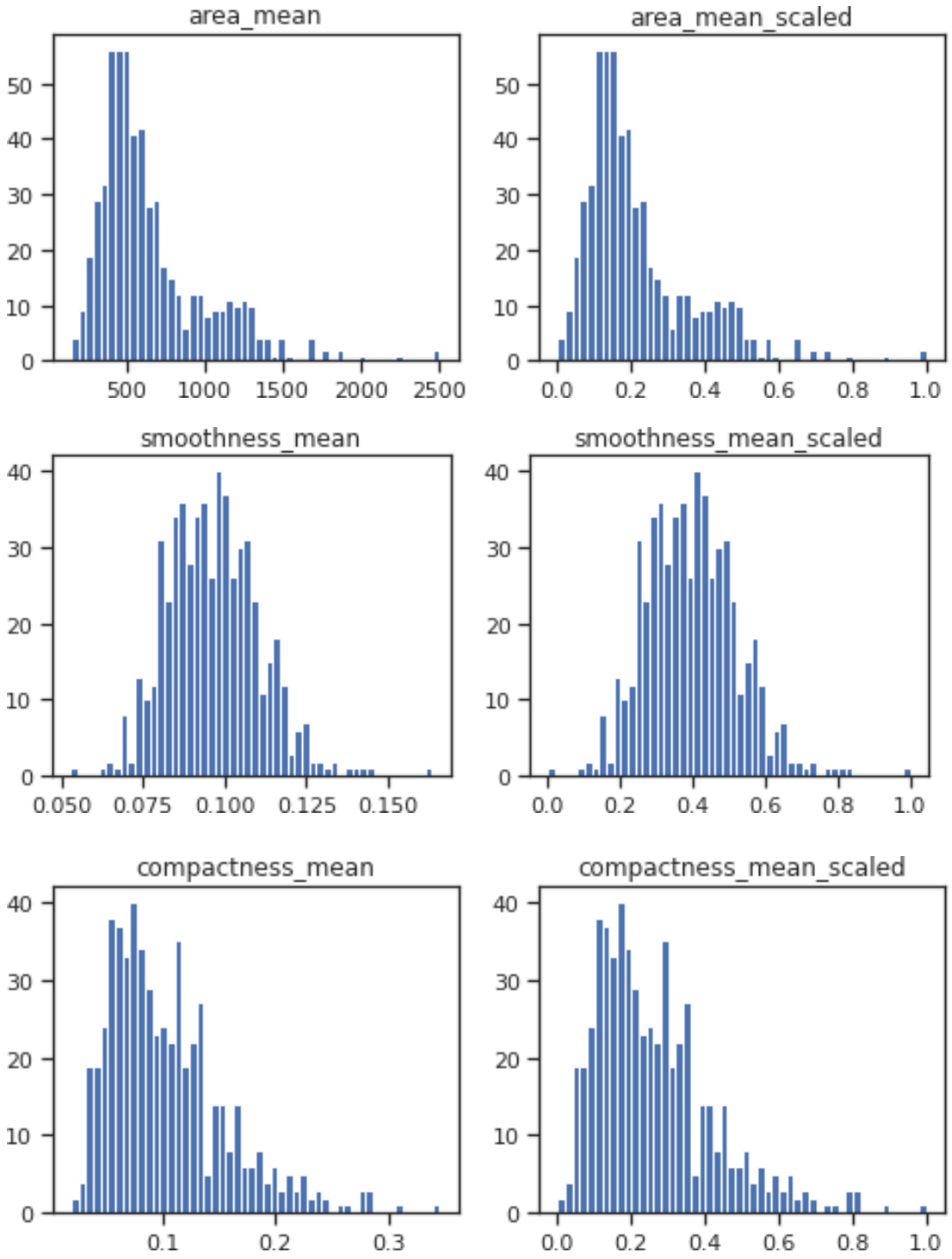
Out[110]:

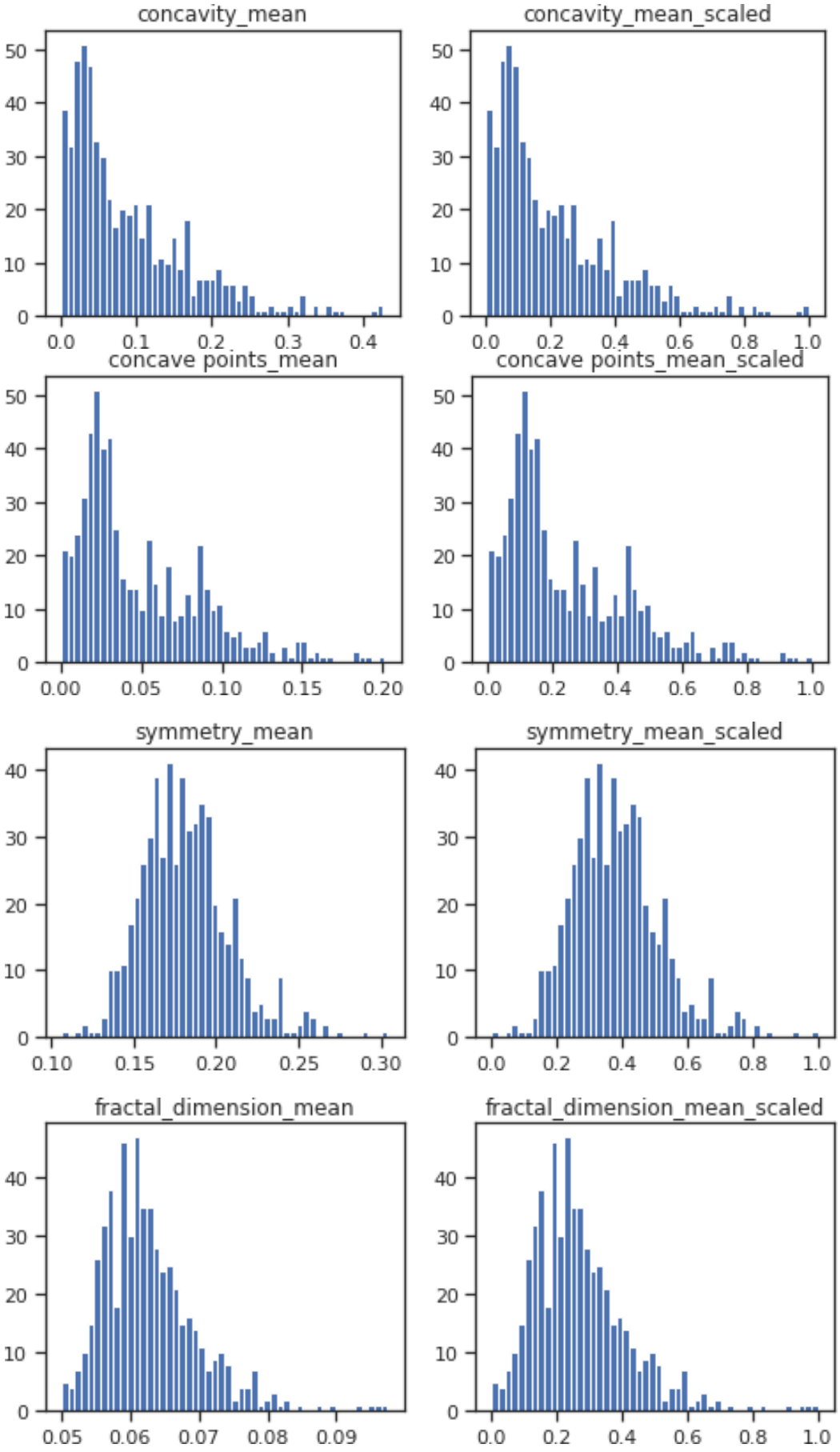
	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 21 columns

In [111]:
```
# 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()
```


radius_mean
radius_mean_scaled
texture_mean
texture_mean_scaled
perimeter_mean
perimeter_mean_scaled


area_mean
area_mean_scaled
smoothness_mean
smoothness_mean_scaled
compactness_mean
compactness_mean_scaled


concavity_mean
concavity_mean_scaled
concave points_mean
concave points_mean_scaled
symmetry_mean
symmetry_mean_scaled
fractal_dimension_mean
fractal_dimension_mean_scaled

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

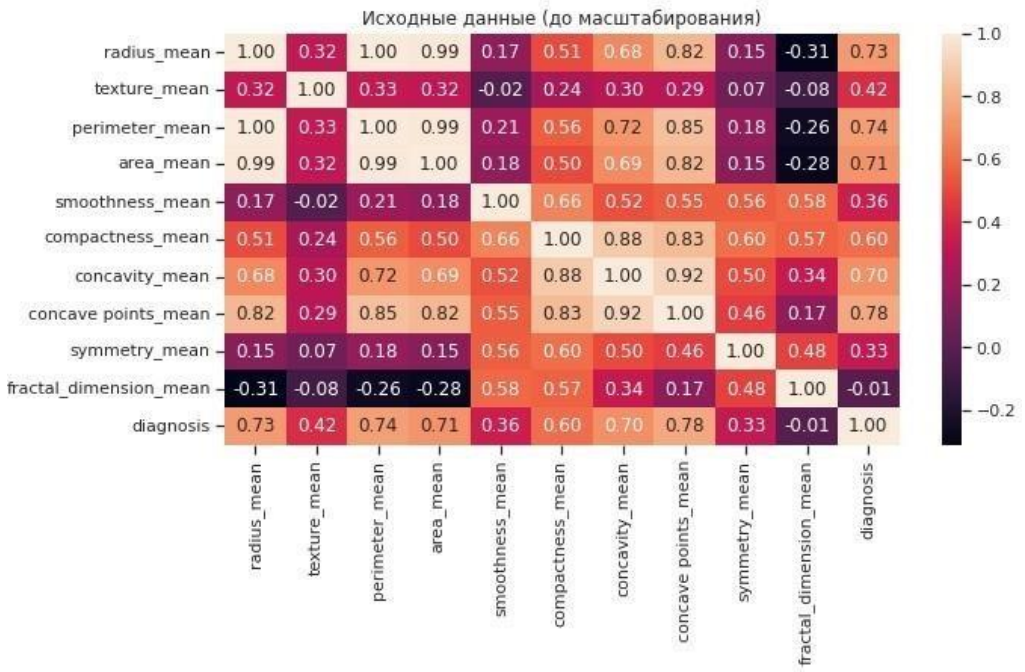
In [112]:
```
# Воспользуемся наличием тестовых выборок,
# включив их в корреляционную матрицу
corr_cols_1 = scale_cols + ['diagnosis']
corr_cols_1
```

Out[112]:
```
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean',
 'concave points_mean',
 'symmetry_mean',
 'fractal_dimension_mean',
 'diagnosis']
```

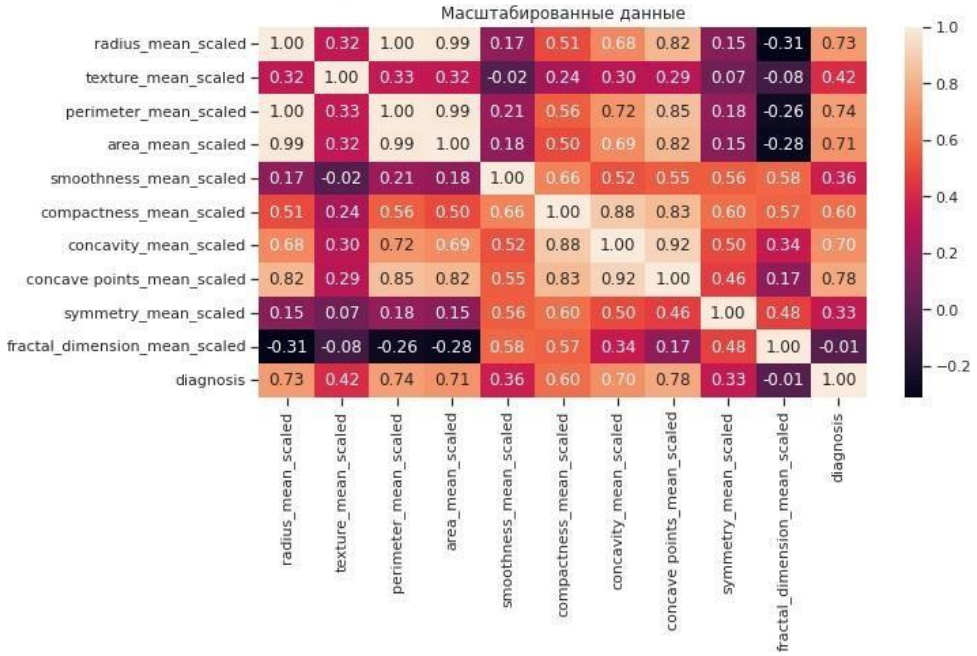
In [113]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['diagnosis']
corr_cols_2
```

Out[113]:
```
['radius_mean_scaled',
 'texture_mean_scaled',
 'perimeter_mean_scaled',
 'area_mean_scaled',
 'smoothness_mean_scaled',
 'compactness_mean_scaled',
 'concavity_mean_scaled',
 'concave points_mean_scaled',
 'symmetry_mean_scaled',
 'fractal_dimension_mean_scaled',
 'diagnosis']
```

In [114]:
```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
ax.set_title('Исходные данные (до масштабирования)')
plt.show()
```



```
In [115]: fig, ax = plt.subplots(figsize=(10,5))
          sns.heatmap(data[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 "diagnosis" наиболее сильно коррелирует с radius_mean (0.73), perimeter_mean (0.74) и concave points_mean (0.78). Эти признаки обязательно следует оставить в модели классификации.

Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:
Метрики, формируемые на основе матрицы ошибок:
Метрика precision: Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.
Метрика recall (полнота): Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.
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:
Метрика ROC AUC:
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.

In [116]:
```
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 задачиклассификации или регрессии.

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия
- Метод ближайших соседей
- Метод опорных векторов
- Дерево решений
- Случайный лес
- Градиентный бустинг

Формирование обучающей и тестовой выборок на основеисходного набора данных.

In [117]:
```
X_train, X_test, y_train, y_test = train_test_split(data, data.diagnosis, random_state=1)
```

In [118]:
```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[118]:
```
((426, 21), (426,), (143, 21), (143,))
```

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.

In [119]:
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(probability=True),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

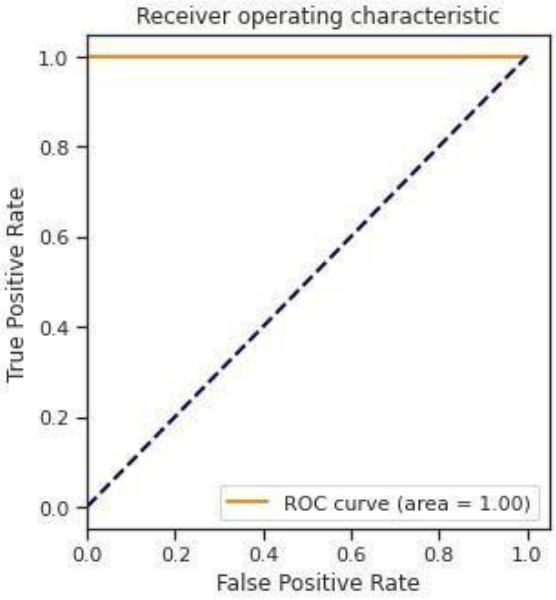
In [120]:
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

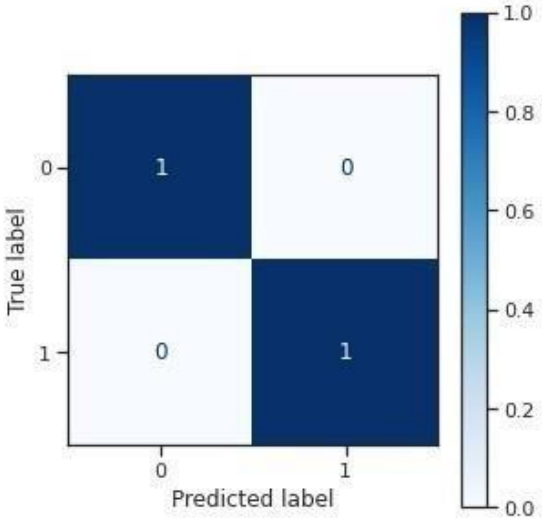
In [121]:
```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    #plt.figure()
    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")
```

In [122]:
```
def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(X_train, y_train)
    # Предсказание значений
    Y_pred = model.predict(X_test)
    # Предсказание вероятности класса "1" для roc auc
    Y_pred_proba_temp = model.predict_proba(X_test)
    Y_pred_proba = Y_pred_proba_temp[:,1]

    precision = precision_score(y_test.values, Y_pred)
    recall = recall_score(y_test.values, Y_pred)
    f1 = f1_score(y_test.values, Y_pred)
    roc_auc = roc_auc_score(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(y_test.values, Y_pred_proba, ax[0])
    plot_confusion_matrix(model, X_test, y_test.values, ax=ax[1],
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    fig.suptitle(model_name)
    plt.show()
```

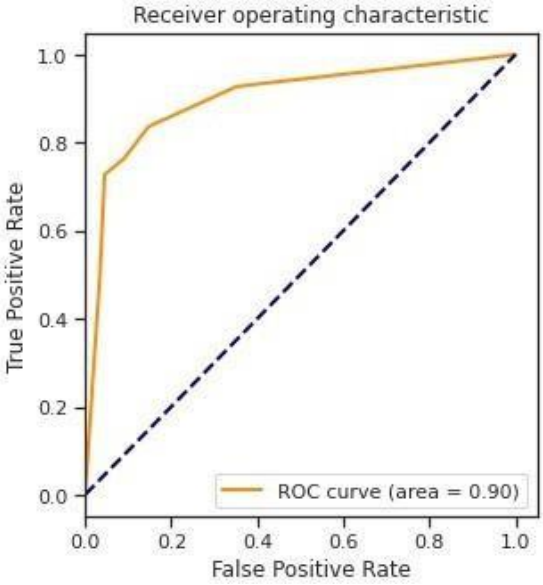
In [123]:
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

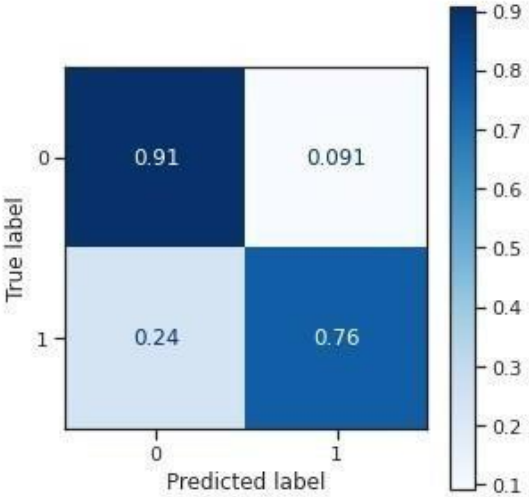
LogR


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

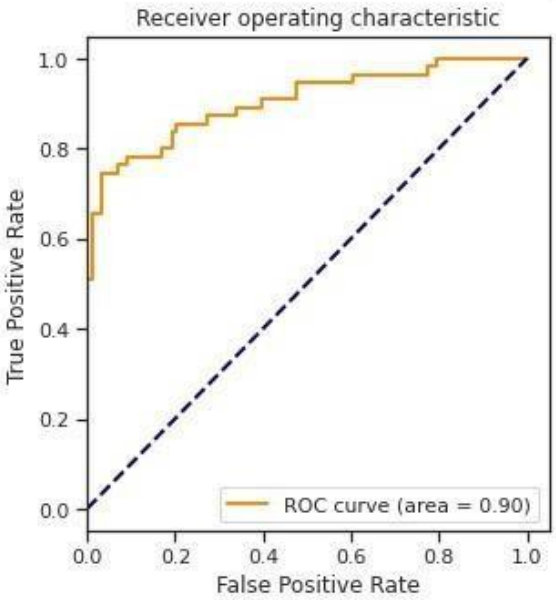

True label
Predicted label
1
0
0
1

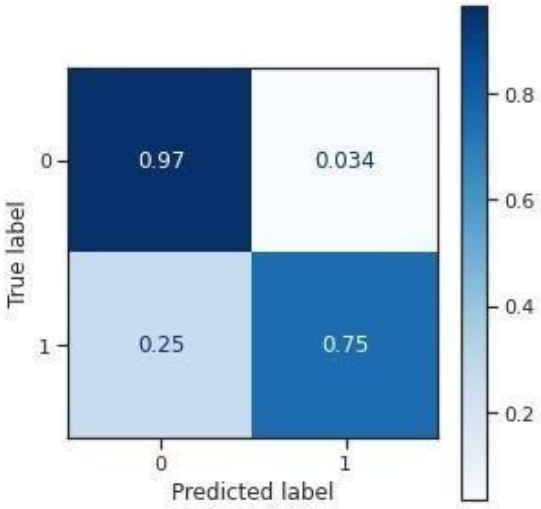
KNN_5


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0

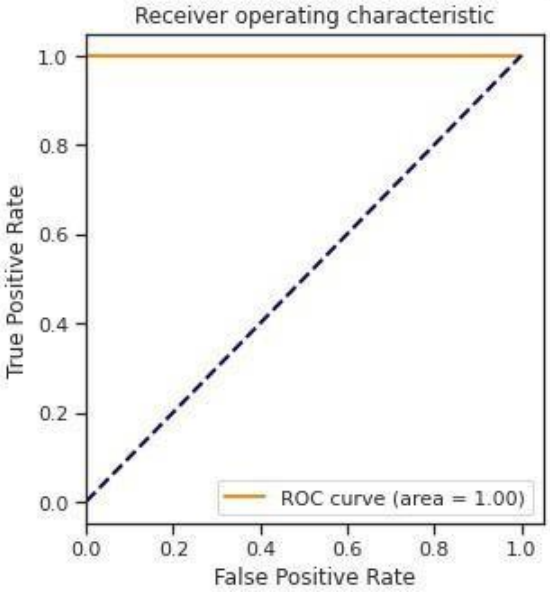

True label
Predicted label
0.91
0.091
0.24
0.76

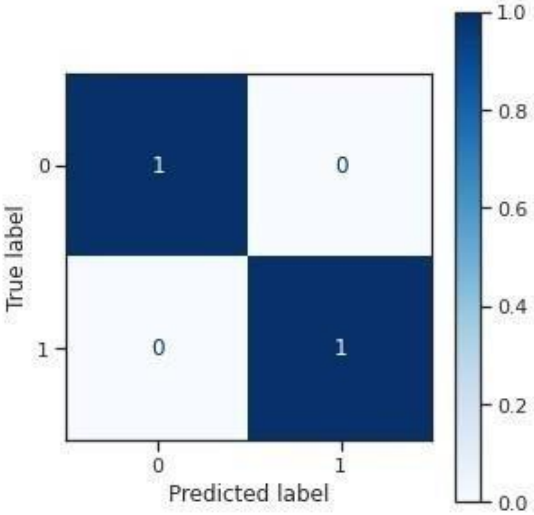
SVC


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0

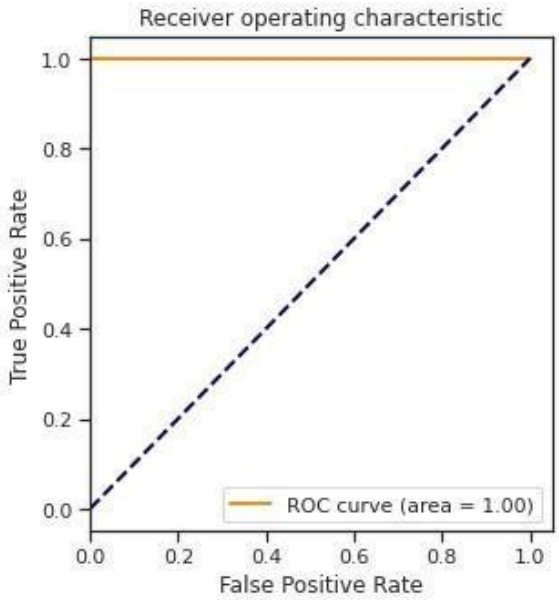

0.97
0.034
0.25
0.75
True label
Predicted label
0
1
0.2
0.4
0.6
0.8

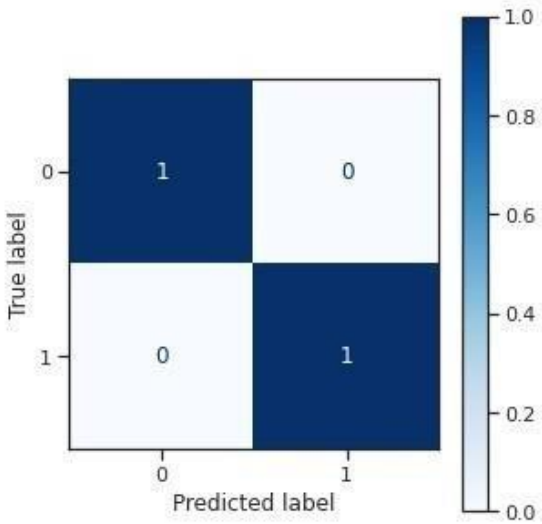
Tree


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

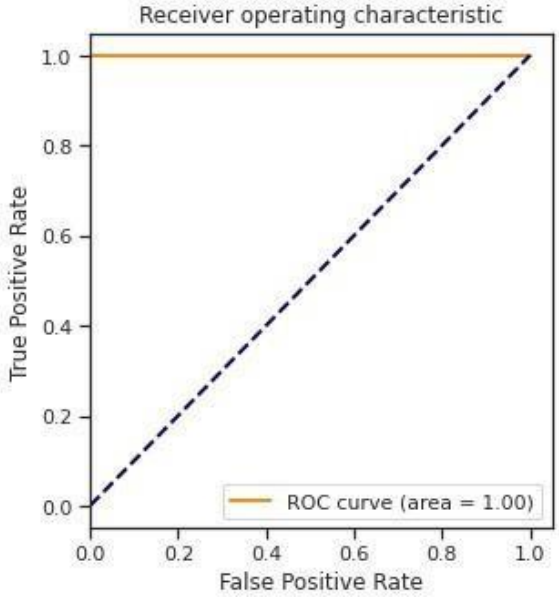

1
0
0
1
True label
Predicted label
0
1
0.0
0.2
0.4
0.6
0.8
1.0

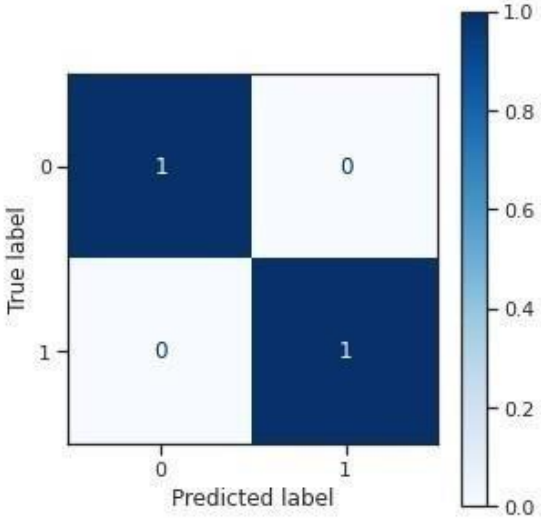
RF


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


True label
Predicted label
1
0
0
1
0
1
1.0
0.8
0.6
0.4
0.2
0.0

GB


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


True label
Predicted label
1
0
0
1
0
1
1.0
0.8
0.6
0.4
0.2
0.0

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.

In [124]:
```
X_train.shape
```

Out[124]:
```
(426, 21)
```

In [125]:
```
n_range_list = list(range(0,1250,50))
n_range_list[0] = 1
```

In [126]:
```
n_range = np.array(n_range_list)
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[126]:
```
[{'n_neighbors': array([   1,   50,  100,  150,  200,  250,  300,  350,  400,  450,  500,
         550,  600,  650,  700,  750,  800,  850,  900,  950, 1000, 1050,
        1100, 1150, 1200])}]
```

In [127]:
```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='roc_auc')
clf_gs.fit(X_train, y_train)
```

```
CPU times: user 1.44 s, sys: 2.54 s, total: 3.98 s
Wall time: 522 ms
```

Out[127]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([   1,   50,  100,  150,  200,  250,  300,  350,  400,  450,  500,
        550,  600,  650,  700,  750,  800,  850,  900,  950, 1000, 1050,
       1100, 1150, 1200])}],
             scoring='roc_auc')
```

In [128]:
```
# Лучшая модель
clf_gs.best_estimator_
```

Out[128]:
```
KNeighborsClassifier(n_neighbors=200)
```

In [129]:
```
# Лучшее значение параметров
clf_gs.best_params_
```

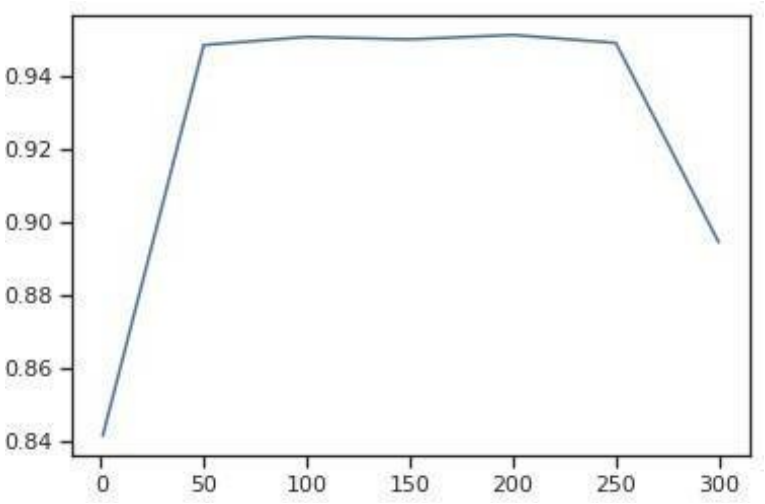
Out[129]:
```
{'n_neighbors': 200}
```

In [130]:
```
clf_gs_best_params_txt = str(clf_gs.best_params_['n_neighbors'])
clf_gs_best_params_txt
```

Out[130]:
```
'200'
```

In [131]:
```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

Out[131]:
```
[<matplotlib.lines.Line2D at 0x7f8ff0dbe7f0>]
```

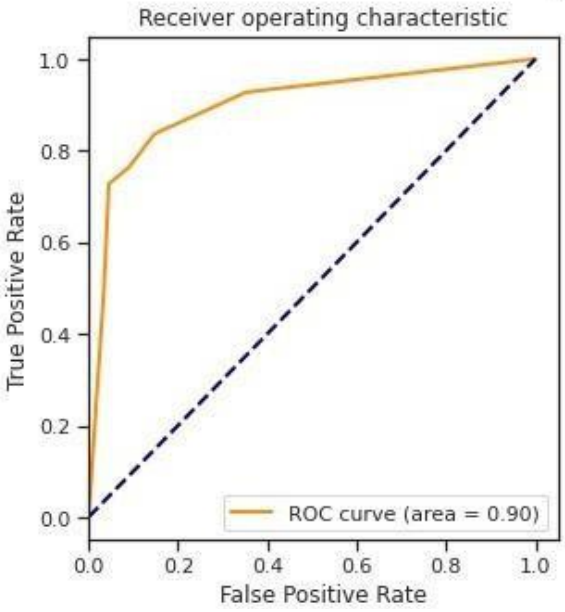


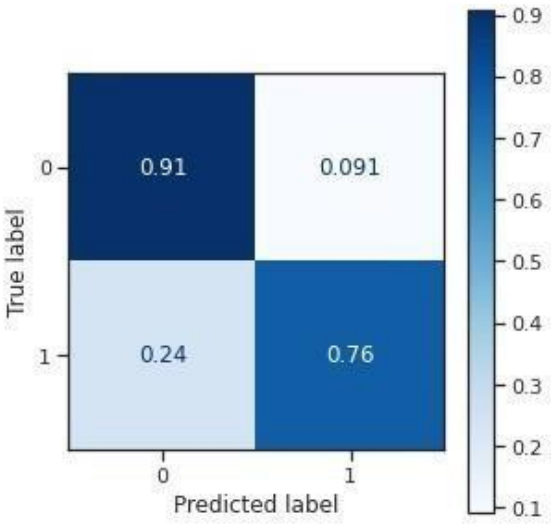
Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей скачеством baseline-моделей.

In [132]:
```
clas_models_grid = {'KNN_5':KNeighborsClassifier(n_neighbors=5),
                    str('KNN_' + clf_gs_best_params_txt):clf_gs.best_estimator_}
```

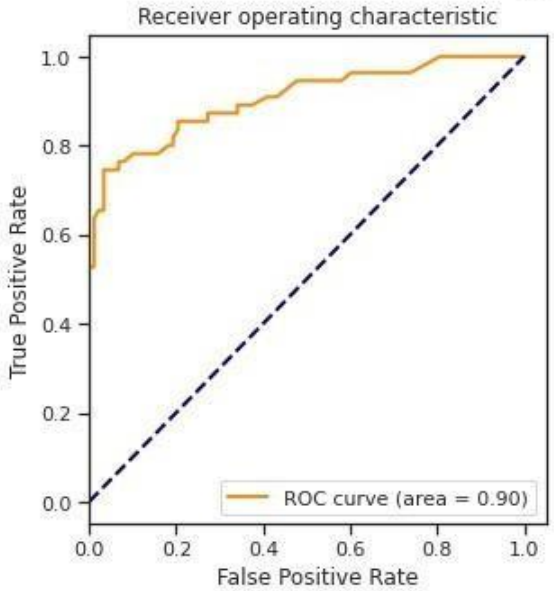
In [133]:
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

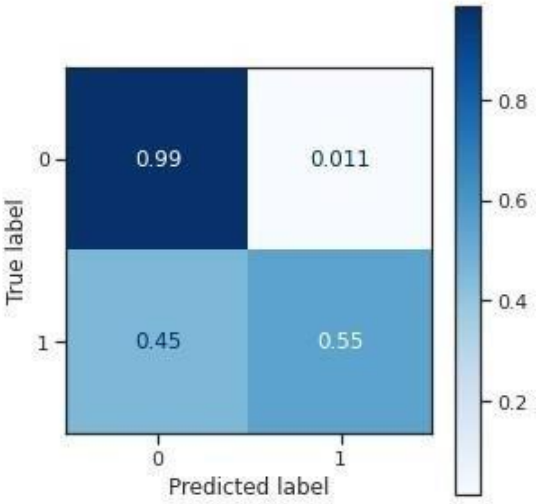
KNN_5





KNN_200



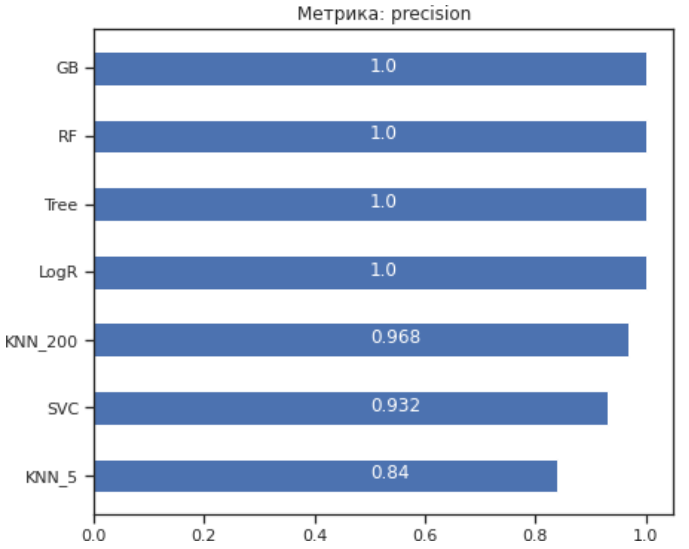


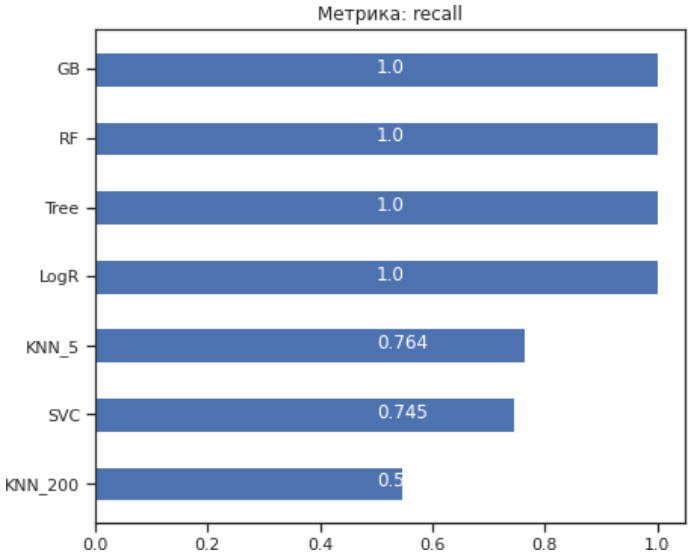
Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпараметров на качество моделей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

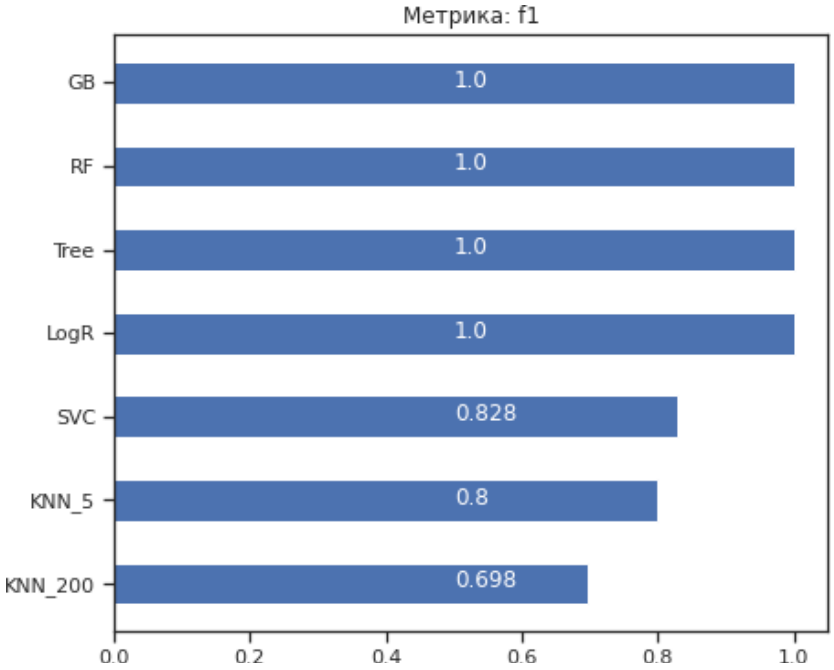
In [134]:
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

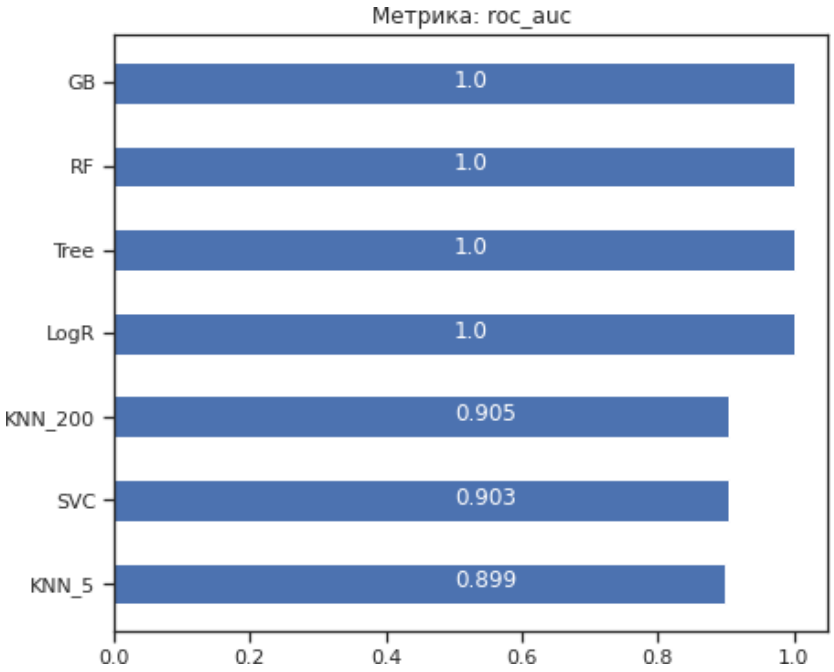
Out[134]:
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

In [135]:
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```









Вывод: Исходя из приведенных метрик, видим, что 4 модели: градиентный бустинг, дерево, логистическая регрессия и случайный лес показывают одинаково высокий результат.

Заключение

В данном курсовом проекте мы выполнили типовую задачу машинного обучения. Провели анализ данных, преобразовали готовый датасет под наши потребности, подобрали модели, а также подобрали наиболее подходящие гиперпараметры.
В данном проекте были закреплены все знания, полученные в курсе лекций и на лабораторных работах. Часть информации была найдена в различных открытых источниках в интернете.
Проделанная работа вызвала интерес к предмету и дальнейшей работе в этой сфере, которая является одной из самых перспективных и актуальных в современном мире.

Список использованных источников информации

1. Документация программной библиотеки seaborn на языке Python [Электронный ресурс]. URL: https://pandas.pydata.org/docs/
2. Документация программной библиотеки Pandas на языке Python [Электронный ресурс]. URL: https://seaborn.pydata.org/
3. Методические указание по разработке НИРС, опорный пример [Электронный ресурс]. URL: https://github.com/ugapanyuk/ml_course_2022/wiki/TMO_NIRS
4. Репозиторий курса “Технологии машинного обучения”, бакалавриат, 6 семестр [Электронный ресурс]. URL: https://github.com/ugapanyuk/ml_course_2022/wiki/COURSE_TMO
5. Kaggle [Электронный ресурс]. URL: https://www.kaggle.com/