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 __ИУ5-61Б____ _____ __**А. Д. Головацкий**__
(Группа) (Подпись, дата) (И.О.Фамилия)

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

Консультант _____ _____
(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 _______
(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 _ИУ5-61Б____

_____Головацкий Андрей Дмитриевич_____
(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 _38_ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

_____

_____

_____

_____

Дата выдачи задания « _15_ » ___февраля__ 2022 г.

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

Студент _____ ____А. Д. Головацкий_
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

Введение

Рак молочной железы является наиболее распространенным видом рака среди женщин в мире. На его долю приходится 25% всех случаев рака, и только в 2015 году от него пострадало более 2,1 миллиона человек. Клетки образуют опухоли, которые можно увидеть на рентгеновском снимке или прощупать в виде уплотнений в области молочной железы.

Основные проблемы, связанные с его обнаружением, заключаются в том, как классифицировать опухоли на злокачественные (раковые) или доброкачественные (не раковые). Применяя модели машинного обучения, мы получаем возможность решить данную задачу.

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

```
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

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

# 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.

Основные характеристики датасета

```
data.head()
```

```
  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        17.99         10.38          122.80     1001.0
1         M        20.57         17.77          132.90     1326.0
2         M        19.69         21.25          130.00     1203.0
3         M        11.42         20.38           77.58      386.1
4         M        20.29         14.34          135.10     1297.0
```

```
   smoothness_mean compactness_mean concavity_mean concave
points_mean \
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   symmetry_mean  fractal_dimension_mean
0         0.2419                 0.07871
1         0.1812                 0.05667
2         0.2069                 0.05999
3         0.2597                 0.09744
4         0.1809                 0.05883
```

```
data.shape
```

```
(569, 11)
```

```
# Список колонок
data.columns
```

```
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')
```

```
# Список колонок с типами данных
data.dtypes
```

```
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
```

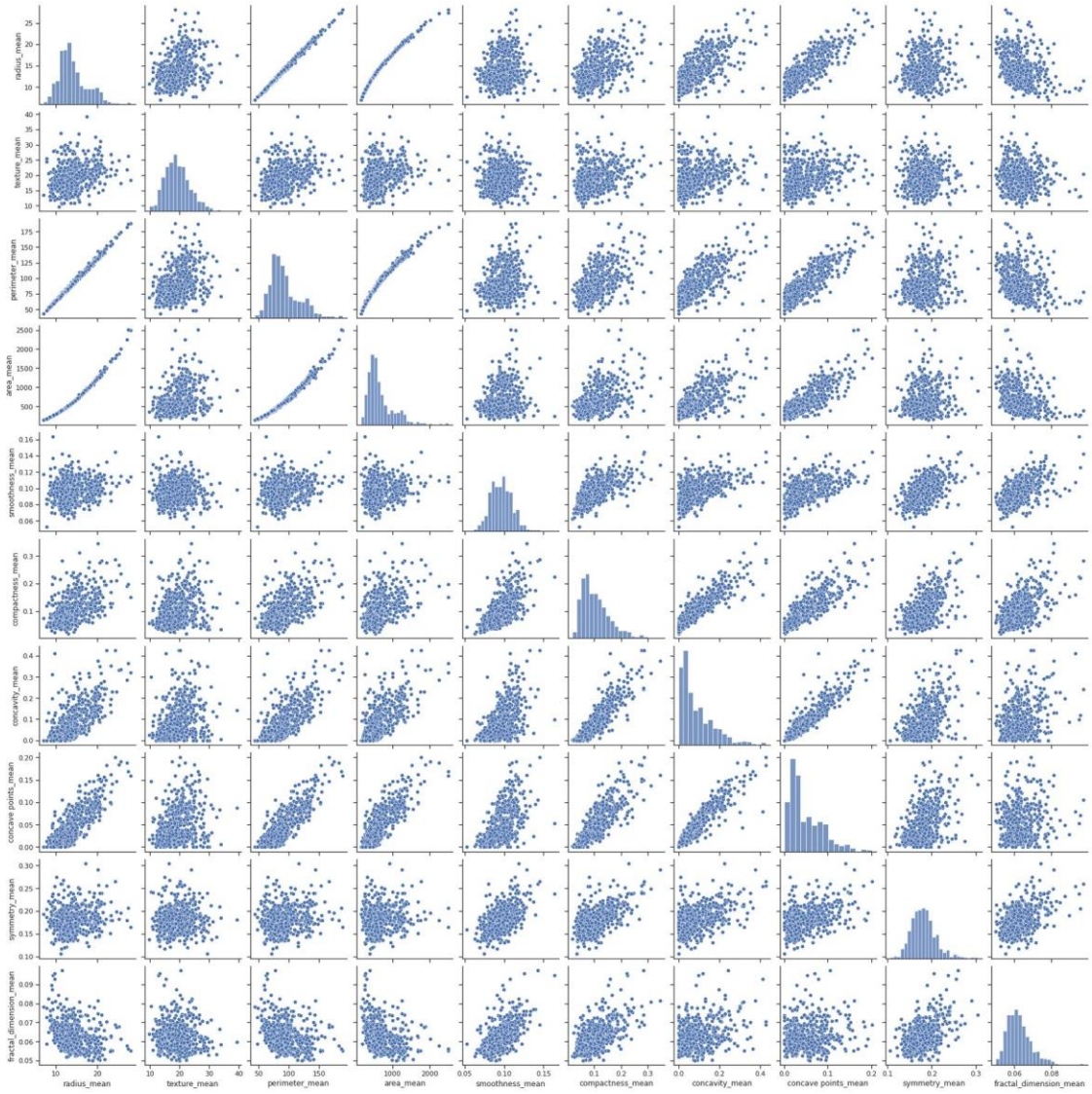
```
# Проверим наличие пустых значений
data.isnull().sum()
```

```
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

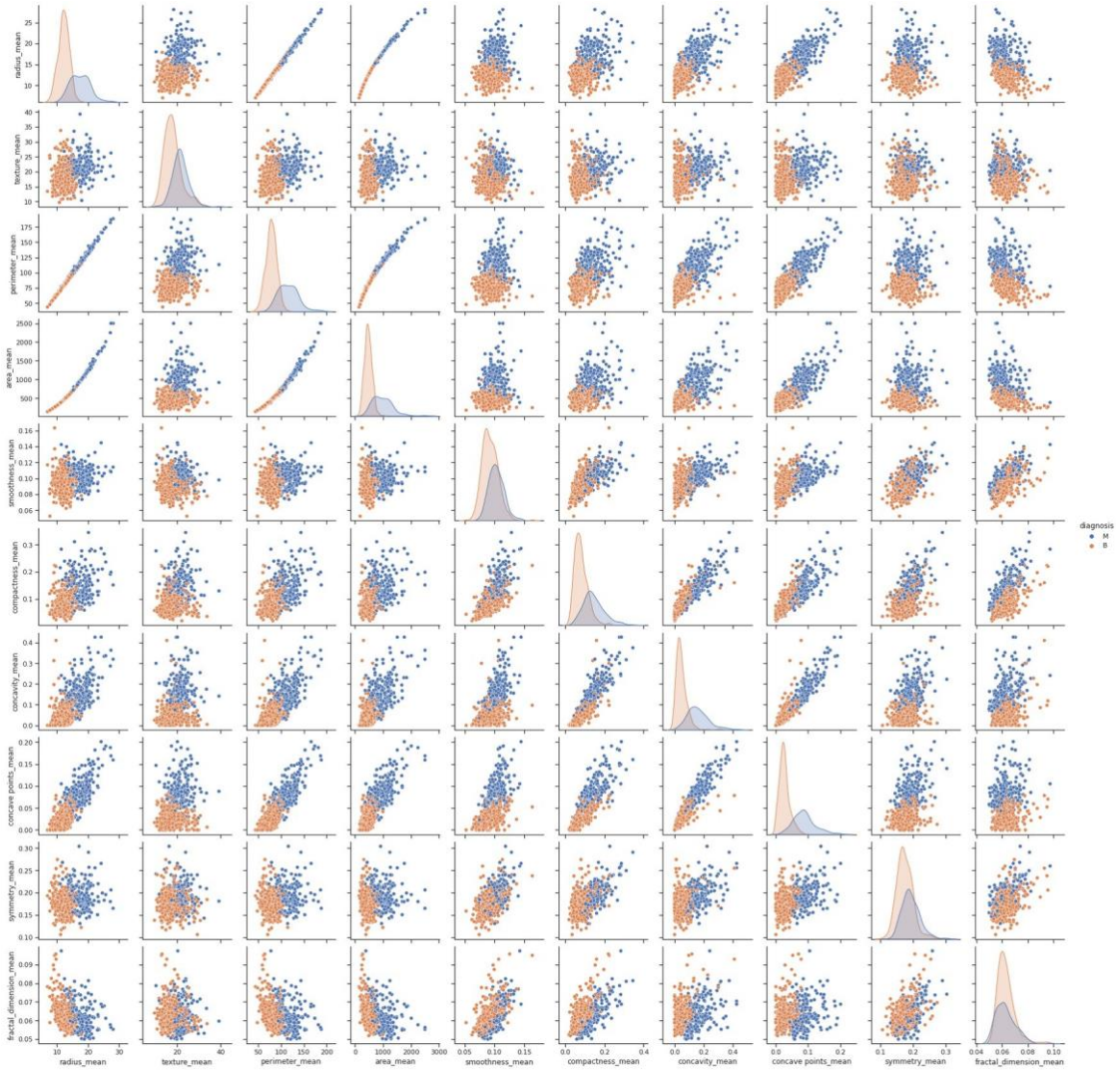
```
# Парные диаграммы
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x7f8fc85e1f70>
```




```
sns.pairplot(data, hue="diagnosis")
```

```
<seaborn.axisgrid.PairGrid at 0x7f8fcba43970>
```




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

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

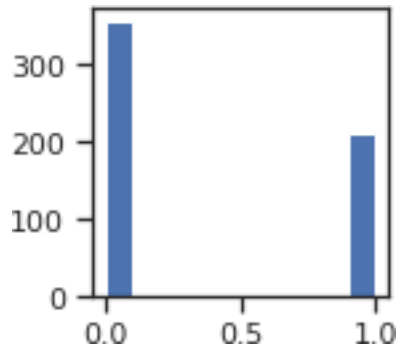
```
diagnosis = LabelEncoder()
code_diagnosis = diagnosis.fit_transform(data["diagnosis"])
data["diagnosis"] = code_diagnosis
data = data.astype({"diagnosis":"int64"})
np.unique(code_diagnosis)
```

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

```
data['diagnosis'].unique()
```

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

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




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

```
0    357
1    212
Name: diagnosis, dtype: int64
```

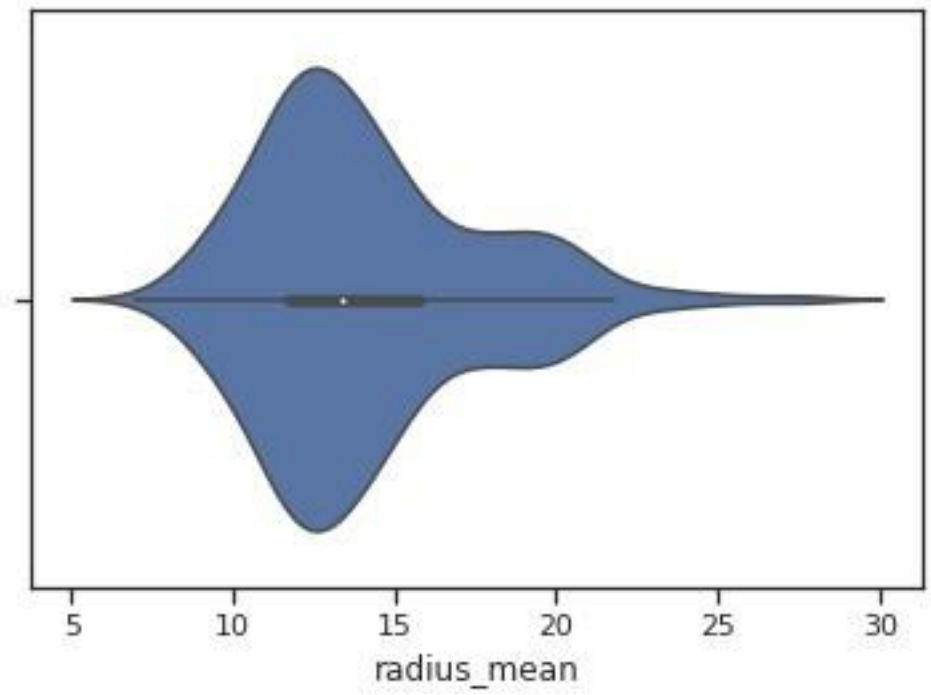
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['diagnosis'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total,
4)*100))
```

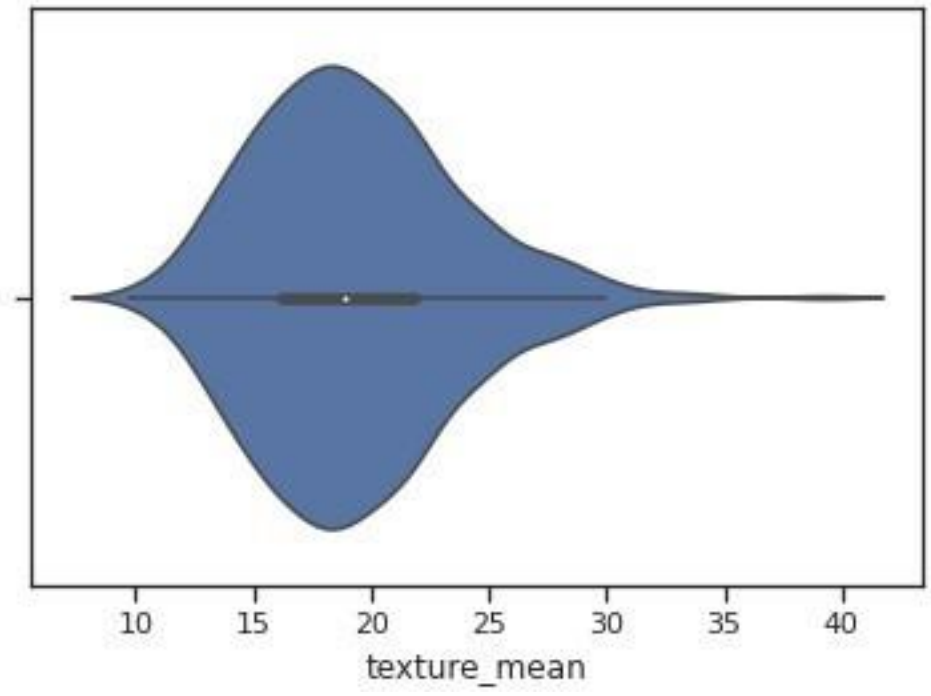
```
Класс 0 составляет 62.739999999999995%, а класс 1 составляет 37.26%.
```

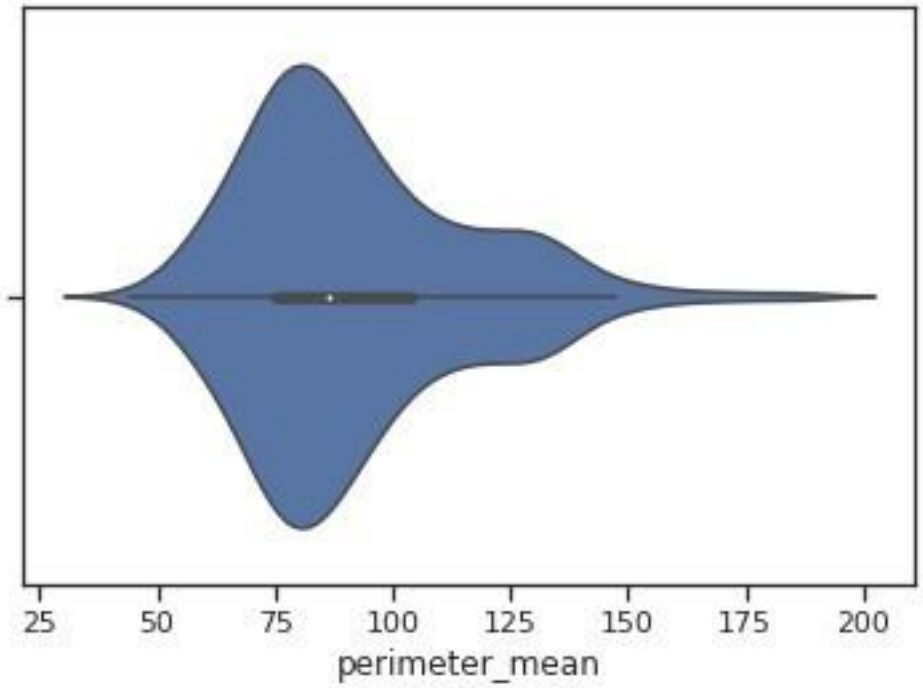
Присутствует незначительный дисбаланс классов.

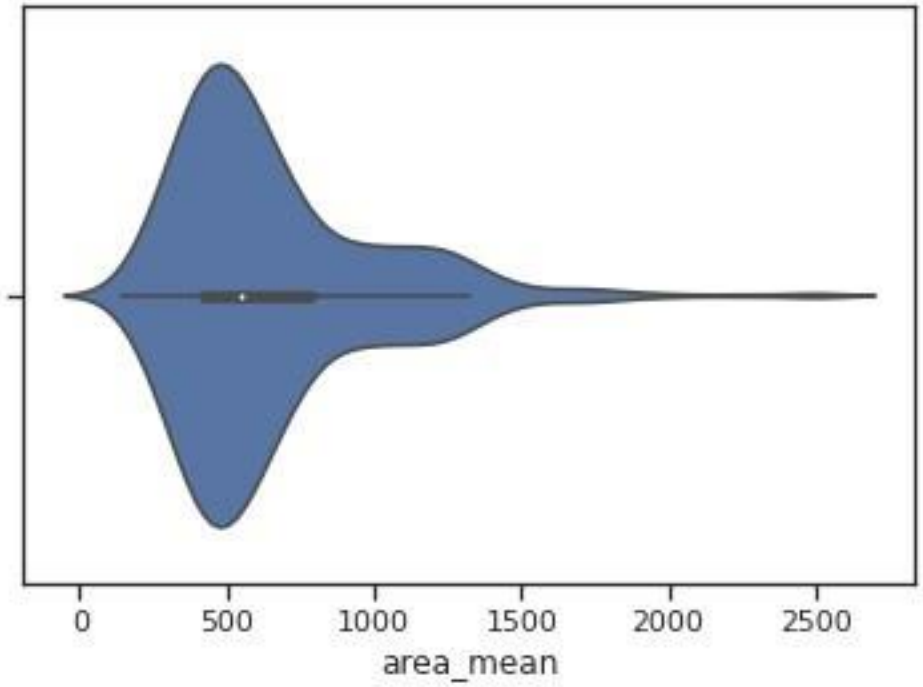
```
def upsample(features, target, repeat):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]

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

    return features_upsampled, target_upsampled

# 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',
```

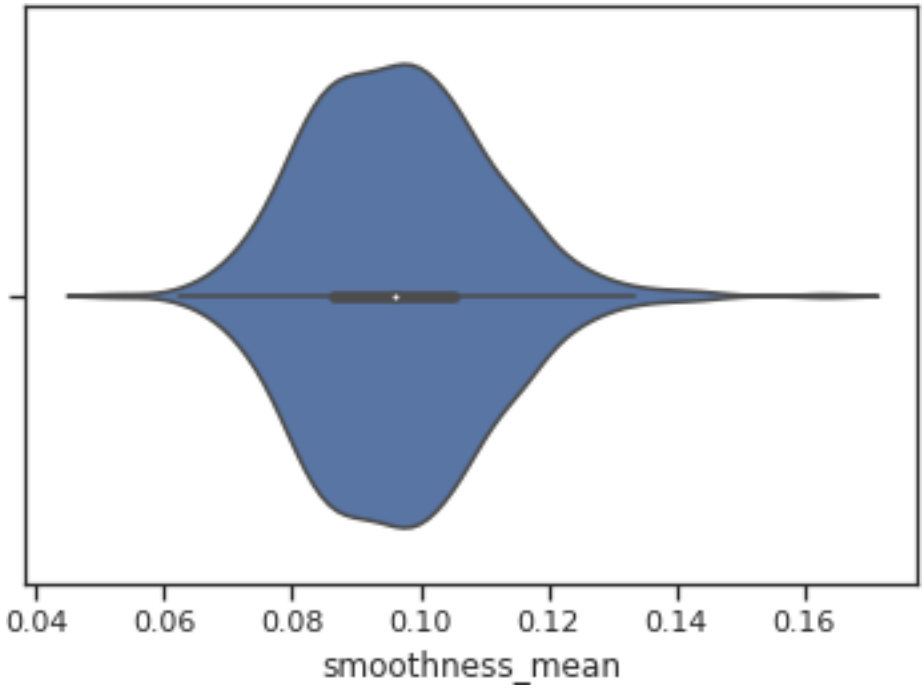
```
'fractal_dimension_mean']:
    sns.violinplot(x=data[col])
    plt.show()
```

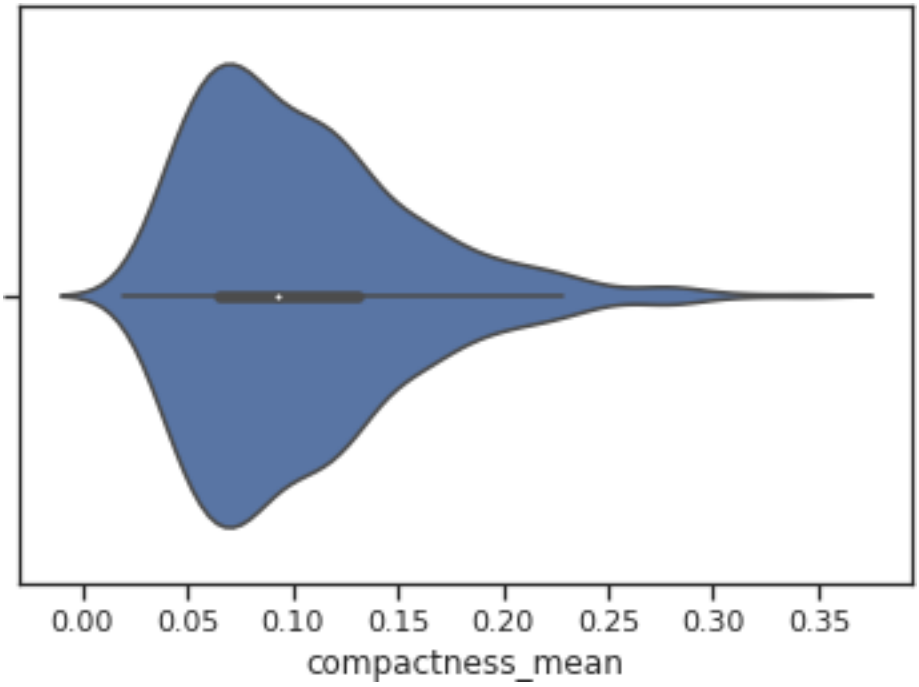


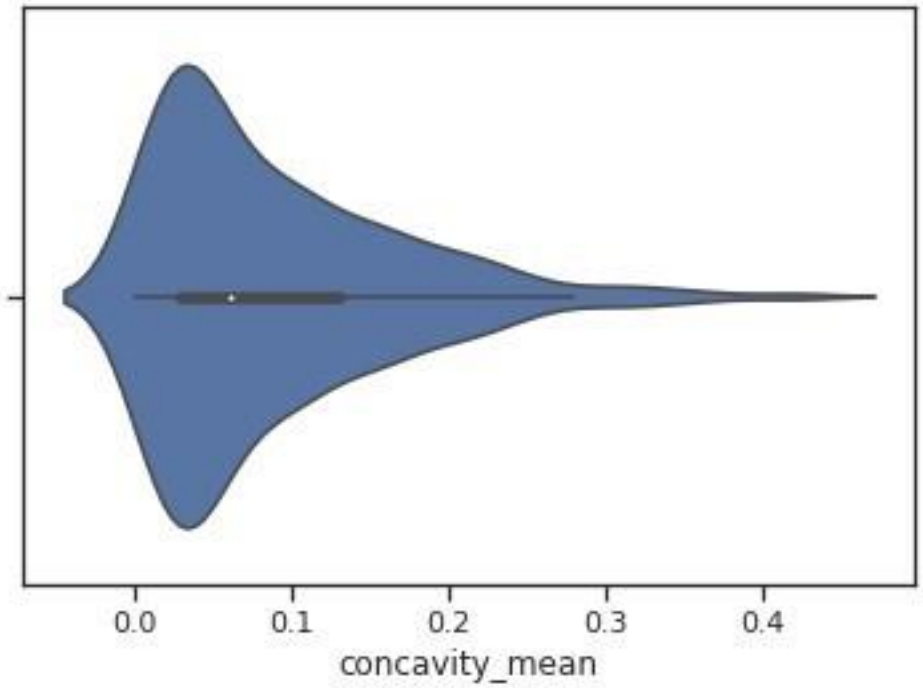


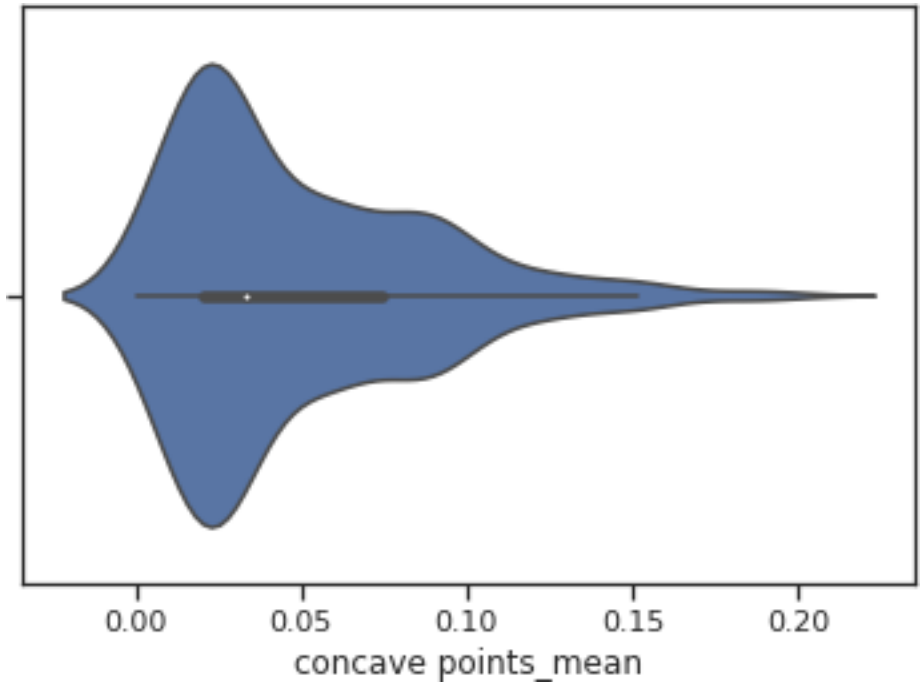

25
50
75
100
125
150
175
200
perimeter_mean

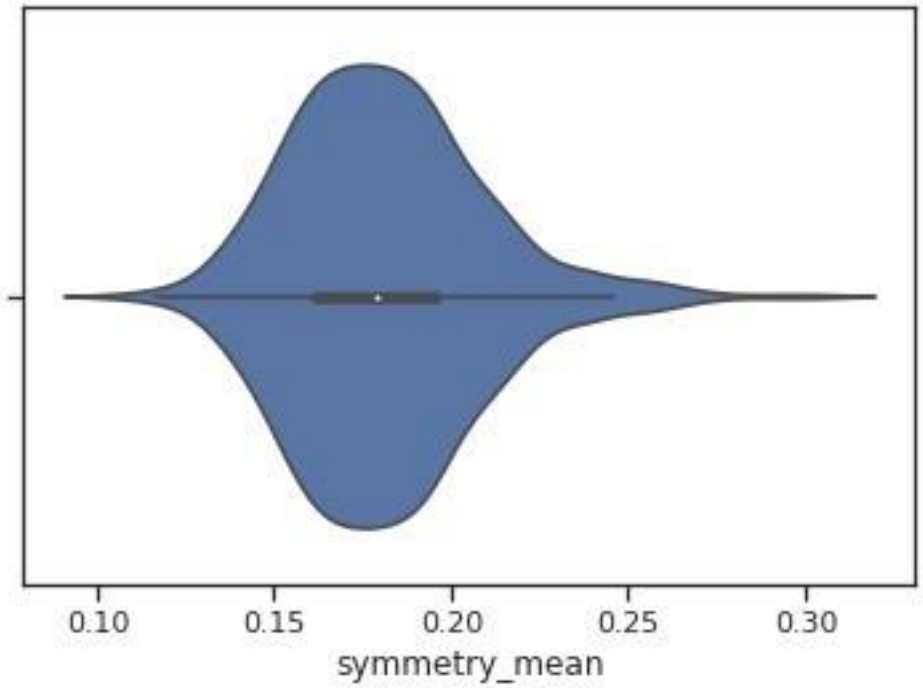

0
500
1000
1500
2000
2500
area_mean

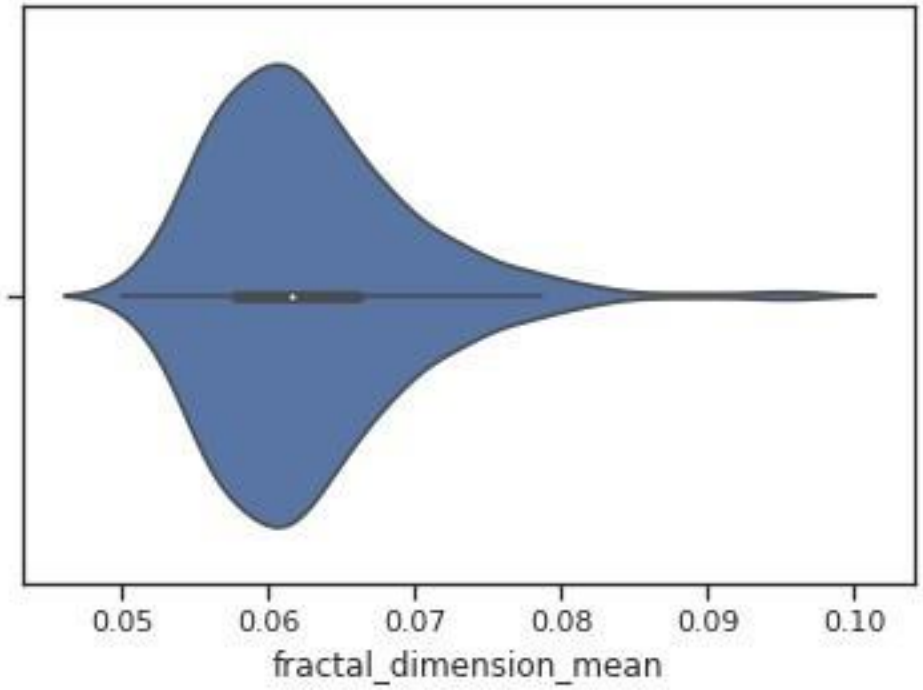

0.04
0.06
0.08
0.10
0.12
0.14
0.16
smoothness_mean


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.

```
data.dtypes
```

```
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.

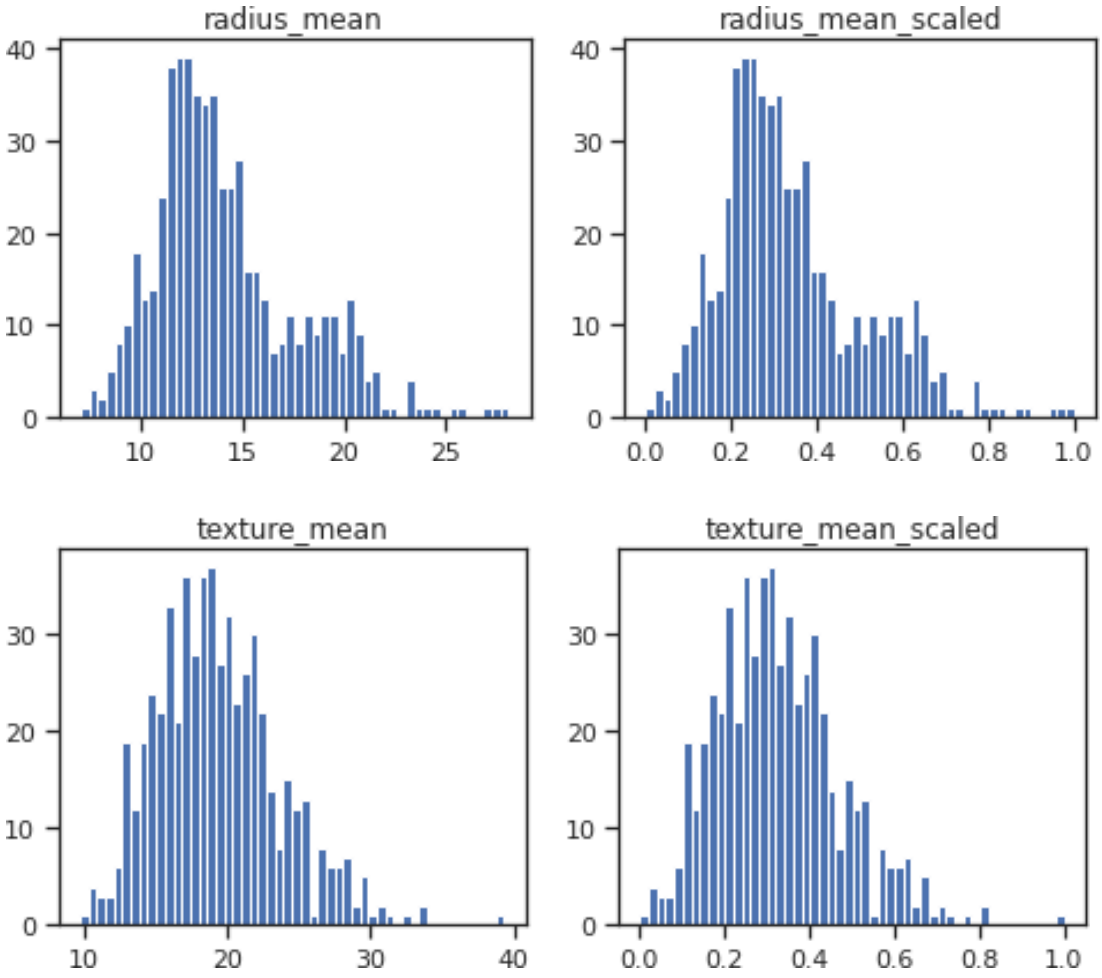
```
# Числовые колонки для масштабирования
scale_cols = ['radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean',
'concavity_mean',
       'concave points_mean', 'symmetry_mean',
'fractal_dimension_mean']

sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[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[new_col_name] = sc1_data[:,i]

data.head()
```

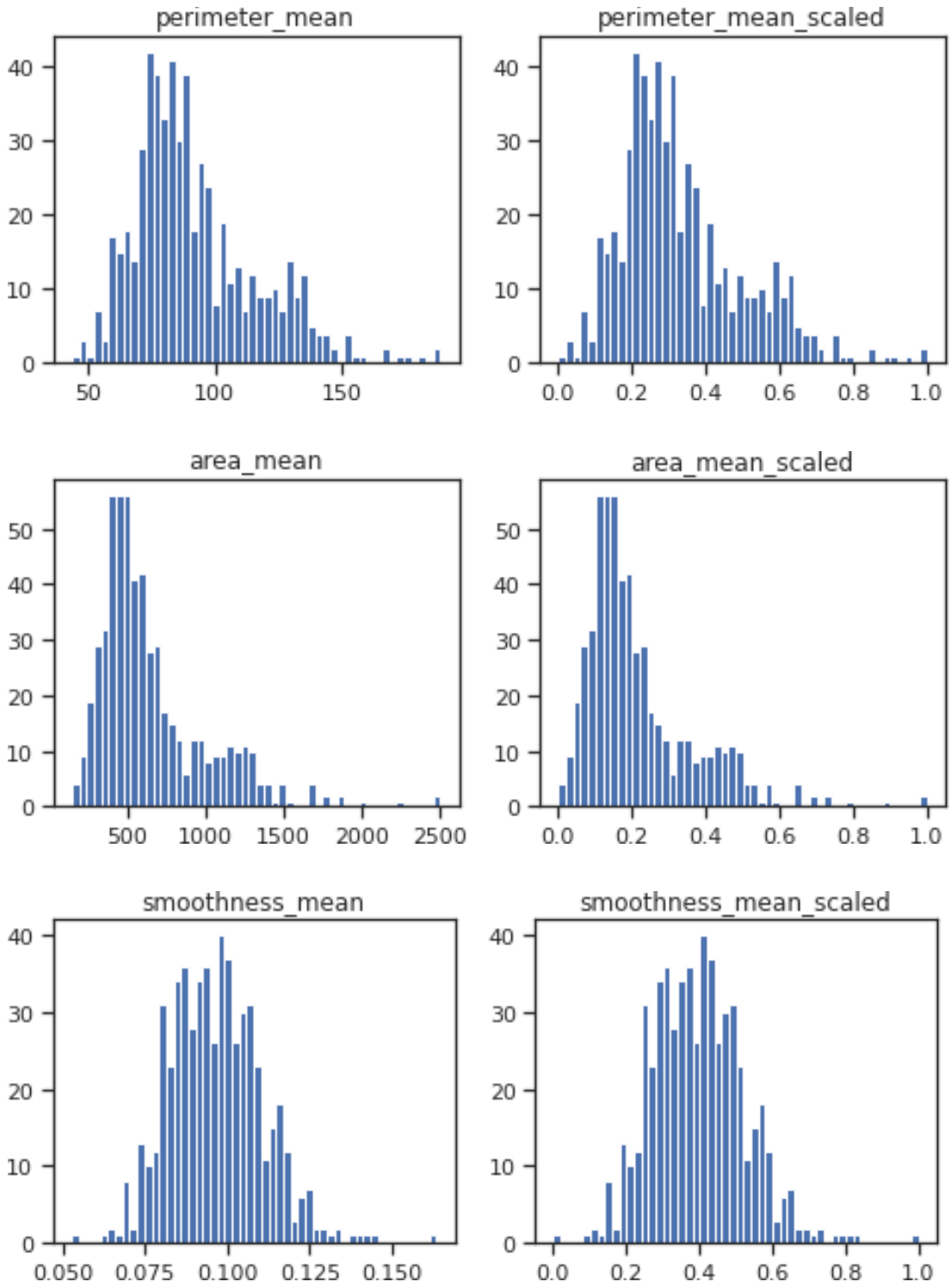
```
   diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0          1        17.99         10.38          122.80     1001.0
1          1        20.57         17.77          132.90     1326.0
2          1        19.69         21.25          130.00     1203.0
3          1        11.42         20.38           77.58      386.1
4          1        20.29         14.34          135.10     1297.0

   smoothness_mean compactness_mean concavity_mean concave
points_mean \
```

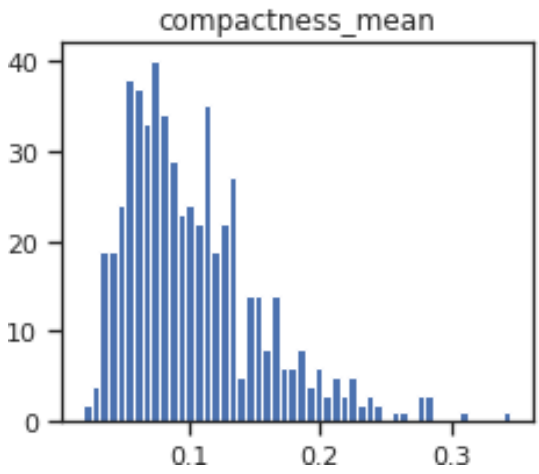
```
0          0.11840           0.27760          0.3001
0.14710
1          0.08474           0.07864          0.0869
0.07017
2          0.10960           0.15990          0.1974
0.12790
3          0.14250           0.28390          0.2414
0.10520
4          0.10030           0.13280          0.1980
0.10430

   symmetry_mean  ...  radius_mean_scaled  texture_mean_scaled  \
0         0.2419  ...            0.521037             0.022658
1         0.1812  ...            0.643144             0.272574
2         0.2069  ...            0.601496             0.390260
3         0.2597  ...            0.210090             0.360839
4         0.1809  ...            0.629893             0.156578

   perimeter_mean_scaled  area_mean_scaled  smoothness_mean_scaled  \
0               0.545989          0.363733                0.593753
1               0.615783          0.501591                0.289880
2               0.595743          0.449417                0.514309
3               0.233501          0.102906                0.811321
4               0.630986          0.489290                0.430351

   compactness_mean_scaled concavity_mean_scaled concave
points_mean_scaled \
0                 0.792037               0.703140
0.731113
1                 0.181768               0.203608
0.348757
2                 0.431017               0.462512
0.635686
3                 0.811361               0.565604
0.522863
4                 0.347893               0.463918
0.518390

   symmetry_mean_scaled  fractal_dimension_mean_scaled
0              0.686364                       0.605518
1              0.379798                       0.141323
2              0.509596                       0.211247
3              0.776263                       1.000000
4              0.378283                       0.186816

[5 rows x 21 columns]
```

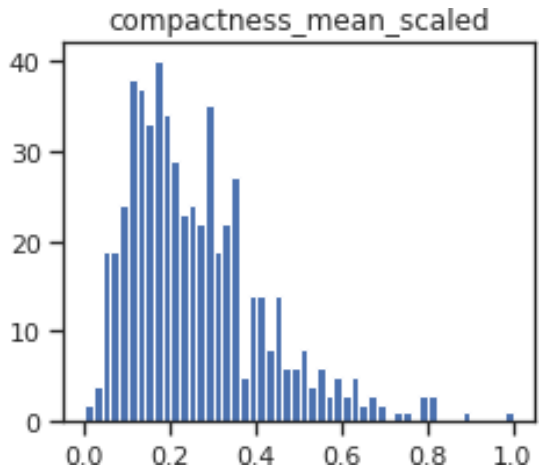
```python
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'
```

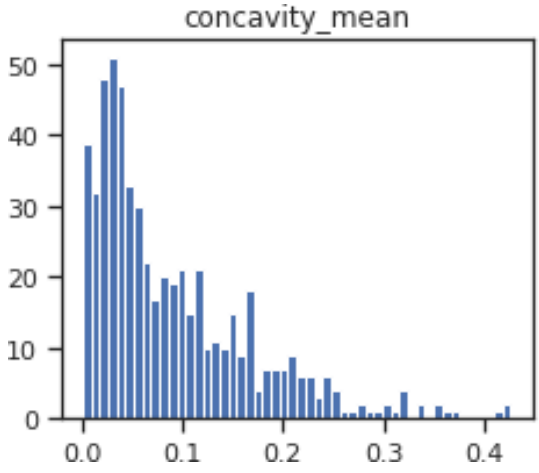
```
fig, ax = plt.subplots(1, 2, figsize=(8,3))
ax[0].hist(data[col], 50)
ax[1].hist(data[col_scaled], 50)
ax[0].title.set_text(col)
ax[1].title.set_text(col_scaled)
plt.show()
```

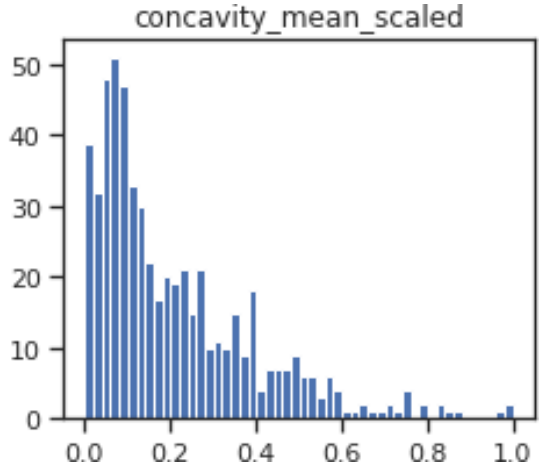


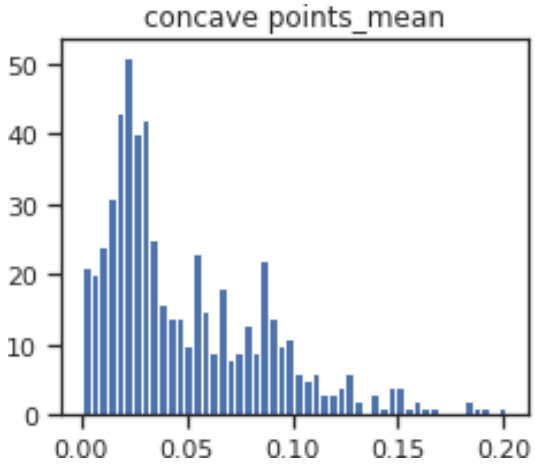

perimeter_mean
perimeter_mean_scaled
area_mean
area_mean_scaled
smoothness_mean
smoothness_mean_scaled

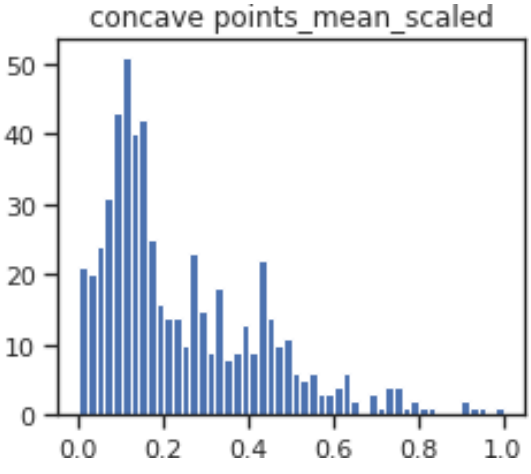

compactness_mean
0
10
20
30
40
0.1
0.2
0.3

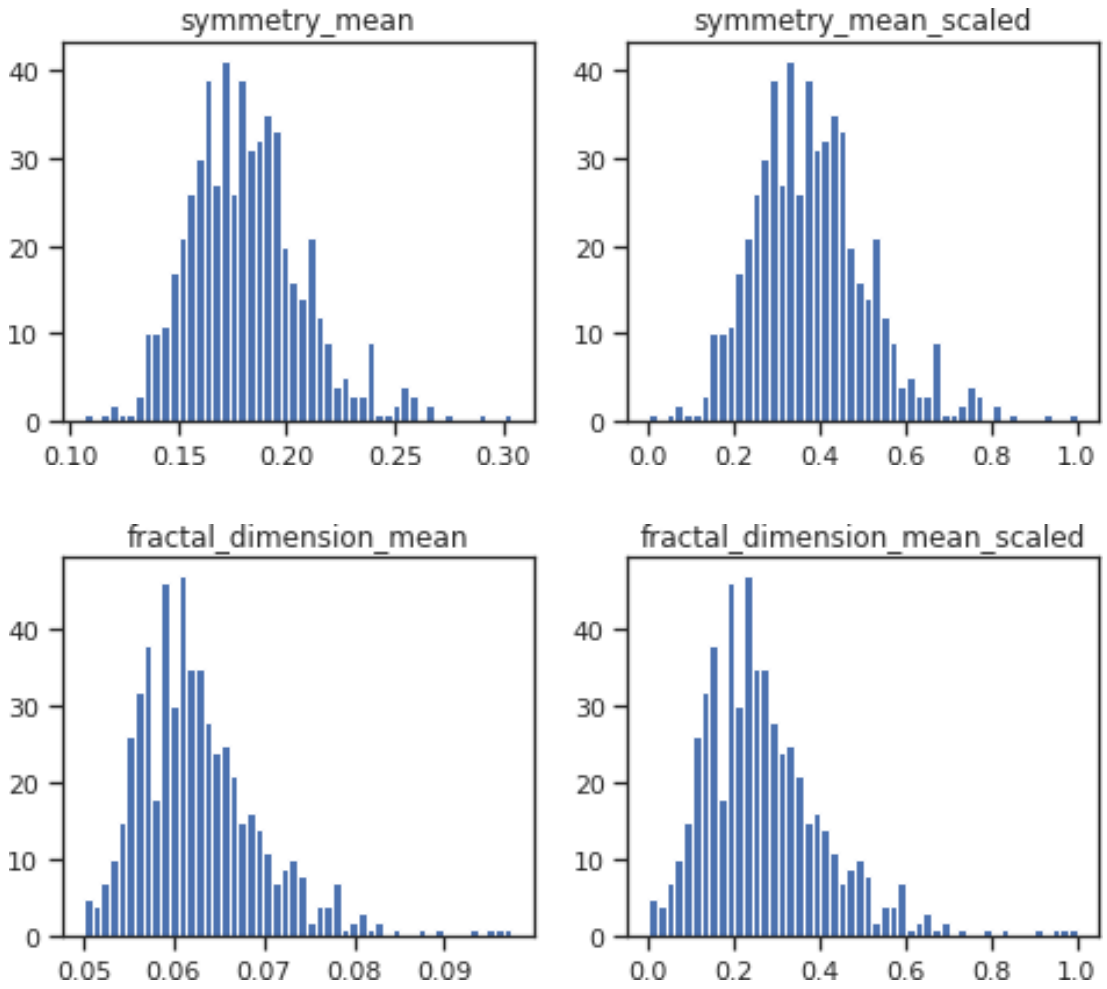

compactness_mean_scaled
0
10
20
30
40
0.0
0.2
0.4
0.6
0.8
1.0


concavity_mean
0
10
20
30
40
50
0.0
0.1
0.2
0.3
0.4


concavity_mean_scaled
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0


concave points_mean
0
10
20
30
40
50
0.00
0.05
0.10
0.15
0.20


concave points_mean_scaled
0
10
20
30
40
50
0.0
0.2
0.4
0.6
0.8
1.0



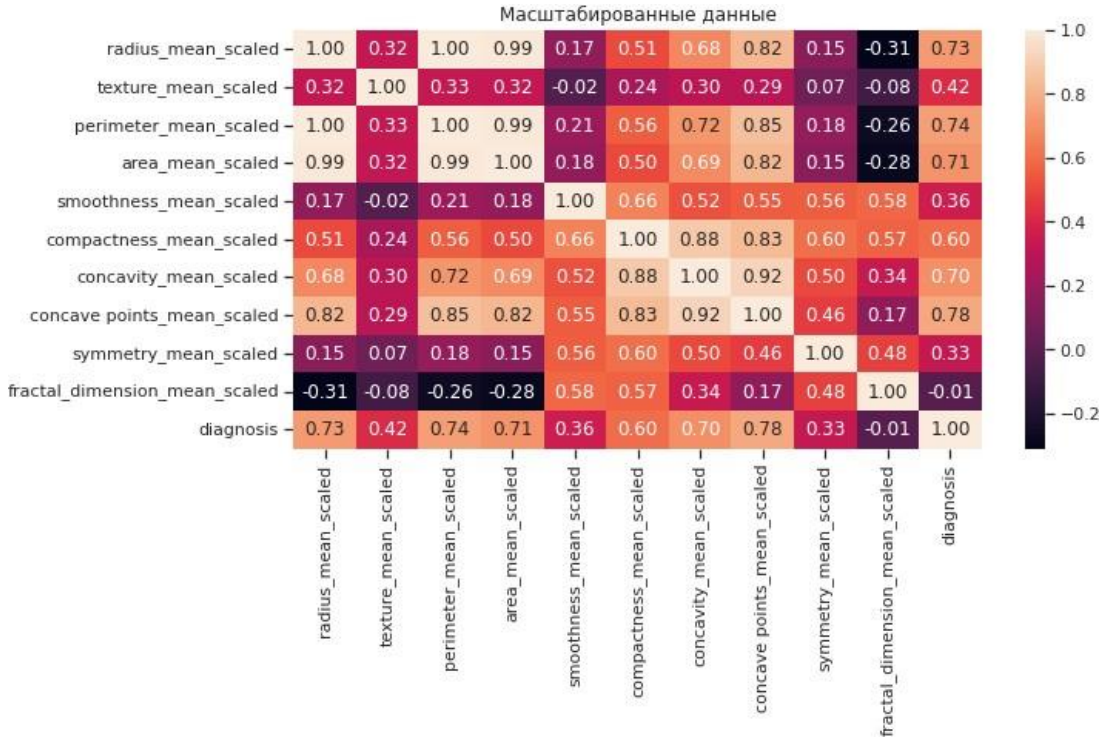
Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

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

```
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean',
 'concave points_mean',
 'symmetry_mean',
```

```
 'fractal_dimension_mean',
 'diagnosis']

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

['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']

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()
```


```
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.

```
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

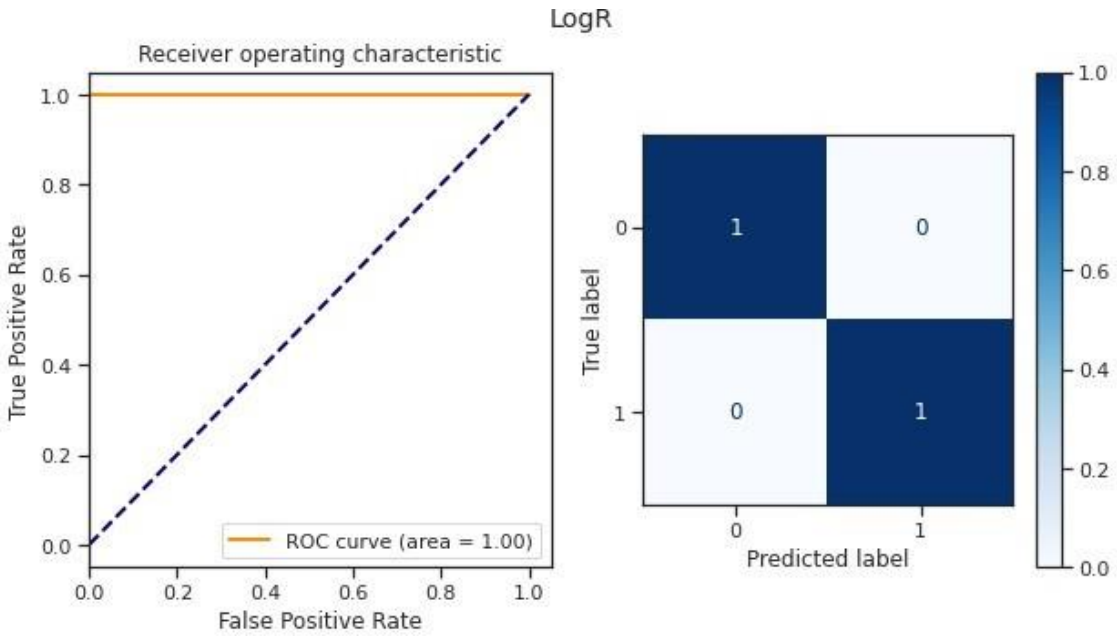
Формирование обучающей и тестовой выборок на основе исходного набора данных.

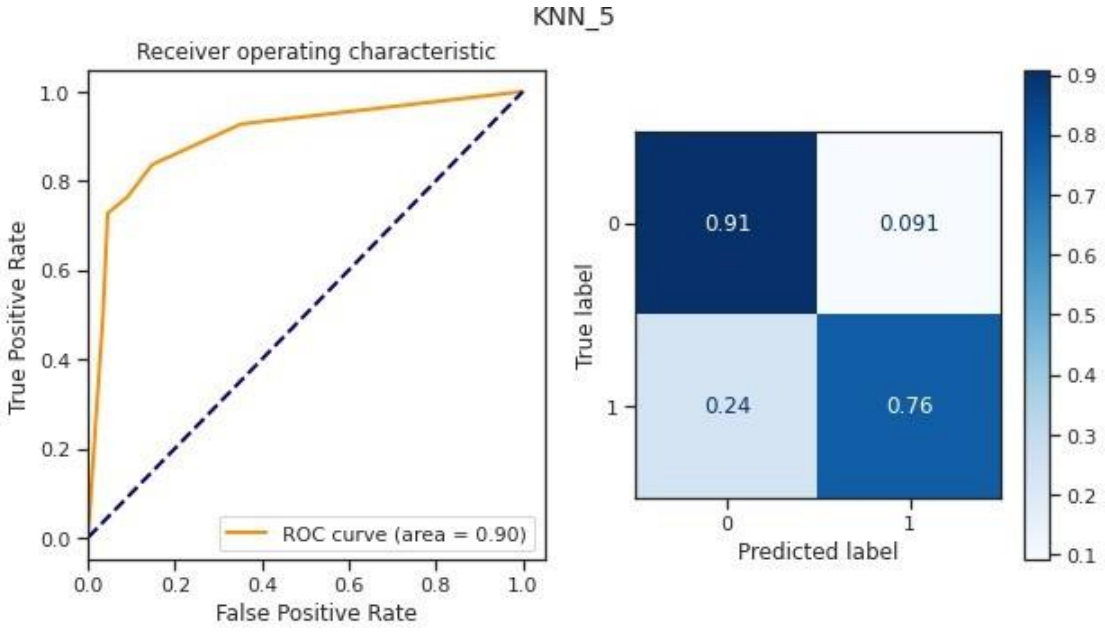
```
X_train, X_test, y_train, y_test = train_test_split(data,
data.diagnosis, random_state=1)

X_train.shape, y_train.shape, X_test.shape, y_test.shape

((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.

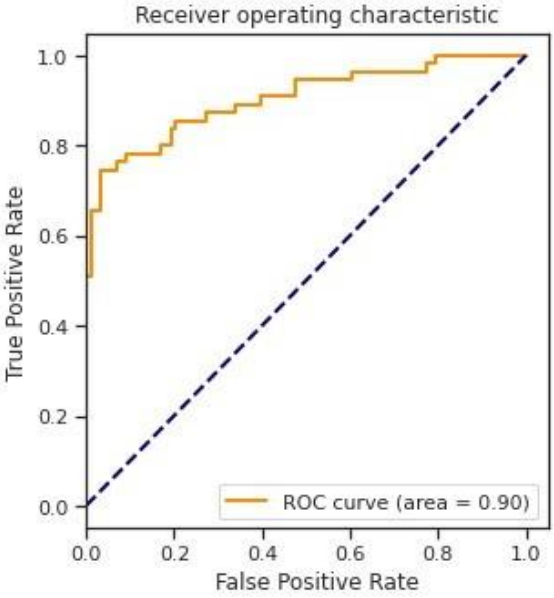
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(probability=True),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}

# 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(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)
```

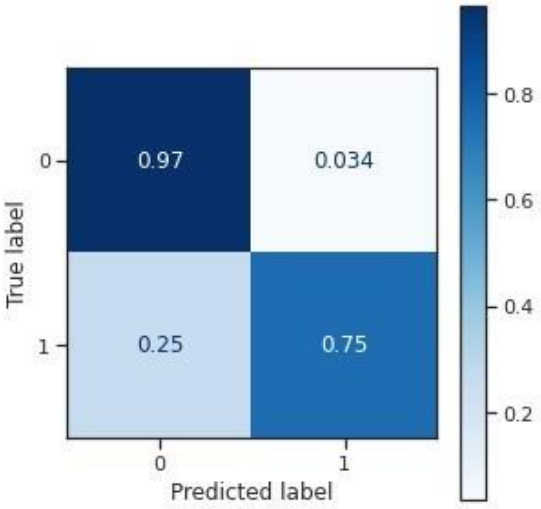
```
    fig, ax = plt.subplots(ncols=2, figsize=(10,5))
    draw_roc_curve(y_test.values, Y_pred_proba, ax[0])
    plot_confusion_matrix(model, X_test, y_test.values, ax=ax[1],
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    fig.suptitle(model_name)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

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



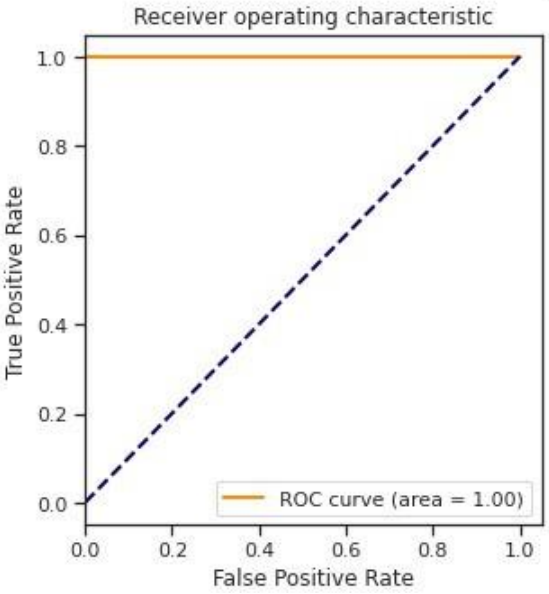


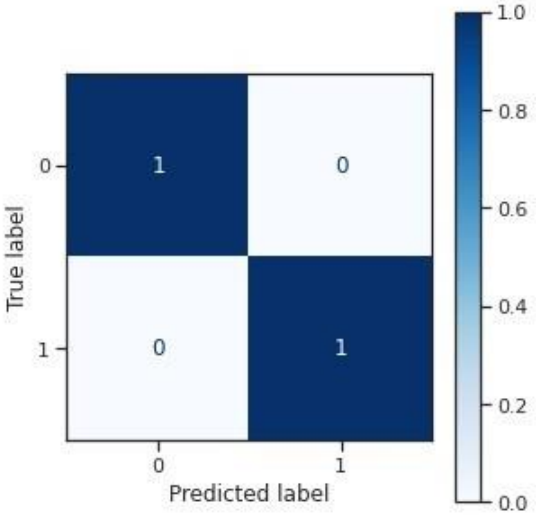
SVC

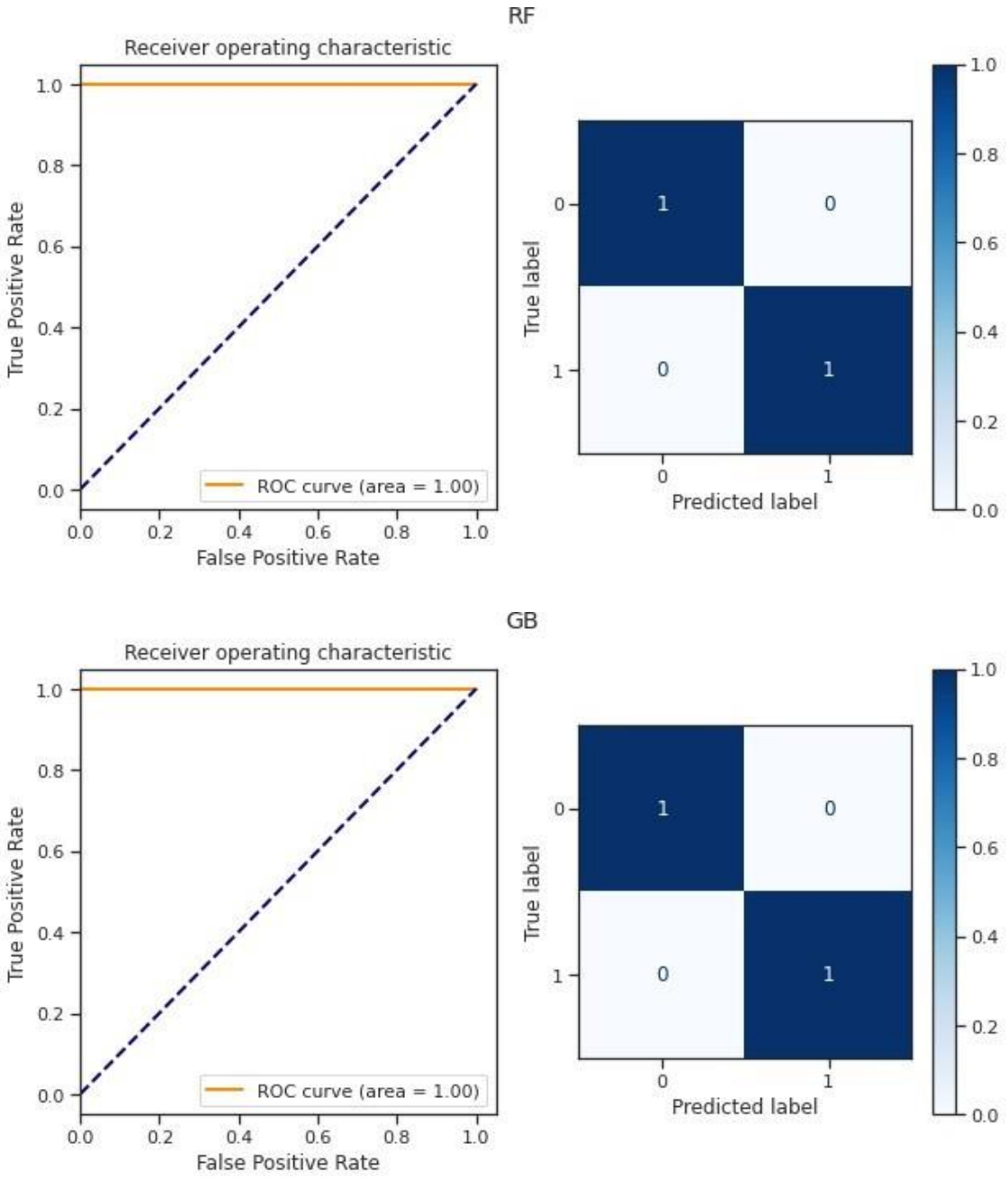

Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0


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



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.

```
X_train.shape
```

```
(426, 21)
```

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

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

```
[{'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])}]
```

```
%%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
```

```
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')
```

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

```
KNeighborsClassifier(n_neighbors=200)
```

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

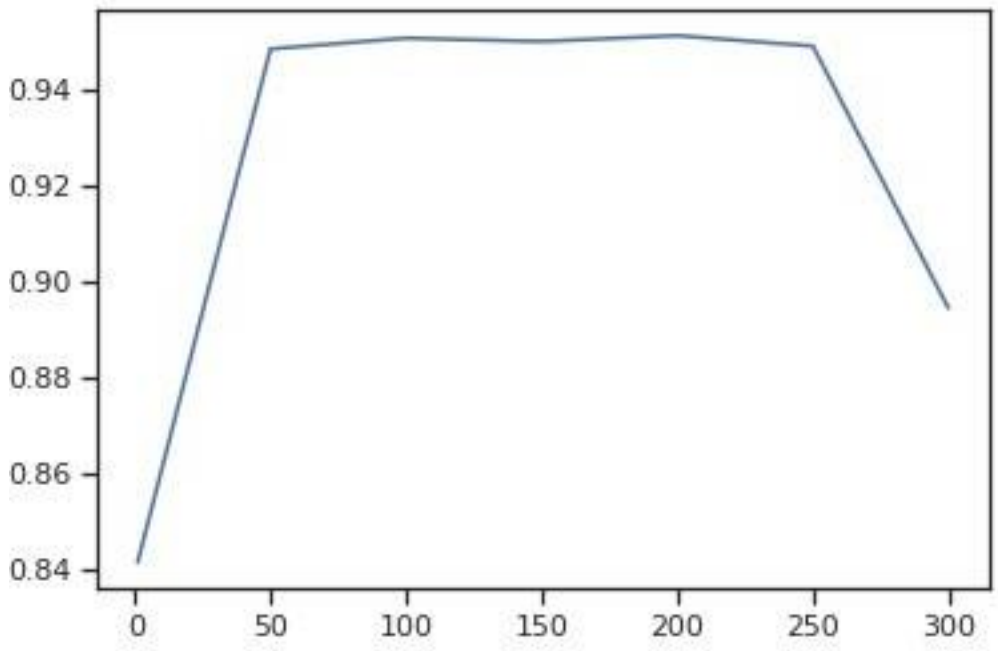
```
{'n_neighbors': 200}
```

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

```
'200'
```

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

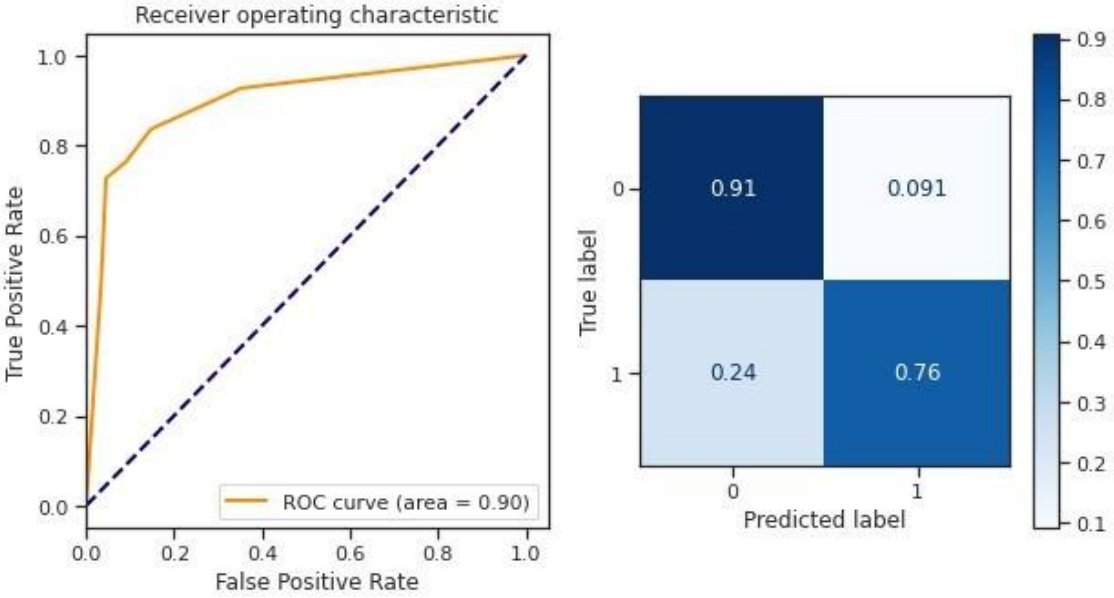
```
[<matplotlib.lines.Line2D at 0x7f8ff0dbe7f0>]
```



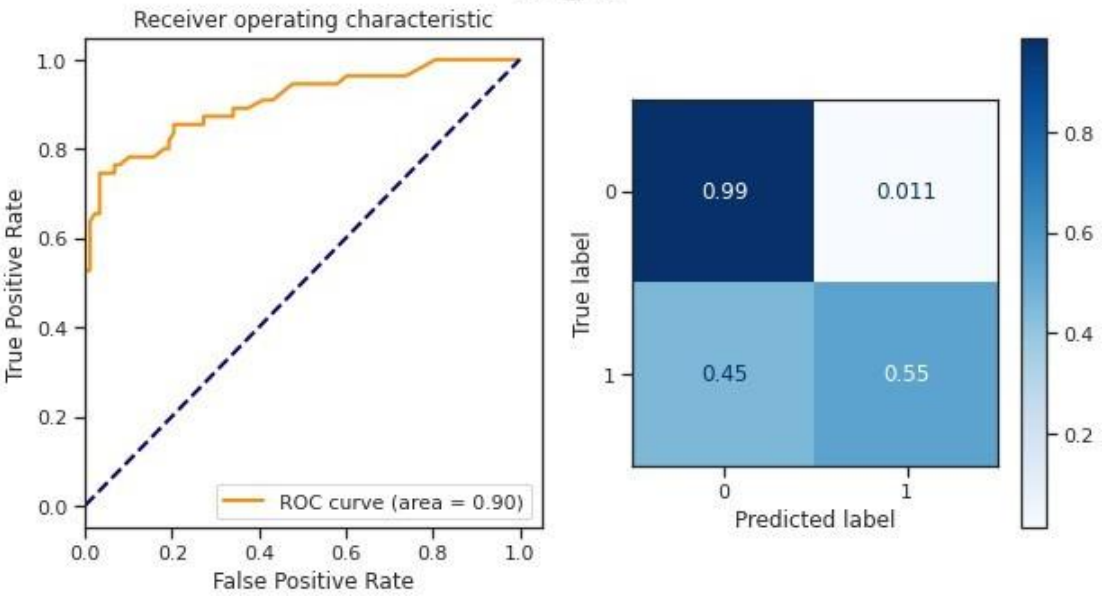
Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

```
clas_models_grid = {'KNN_5':KNeighborsClassifier(n_neighbors=5),
                    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)
```

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
0.91
0.091
0.24
0.76
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_200

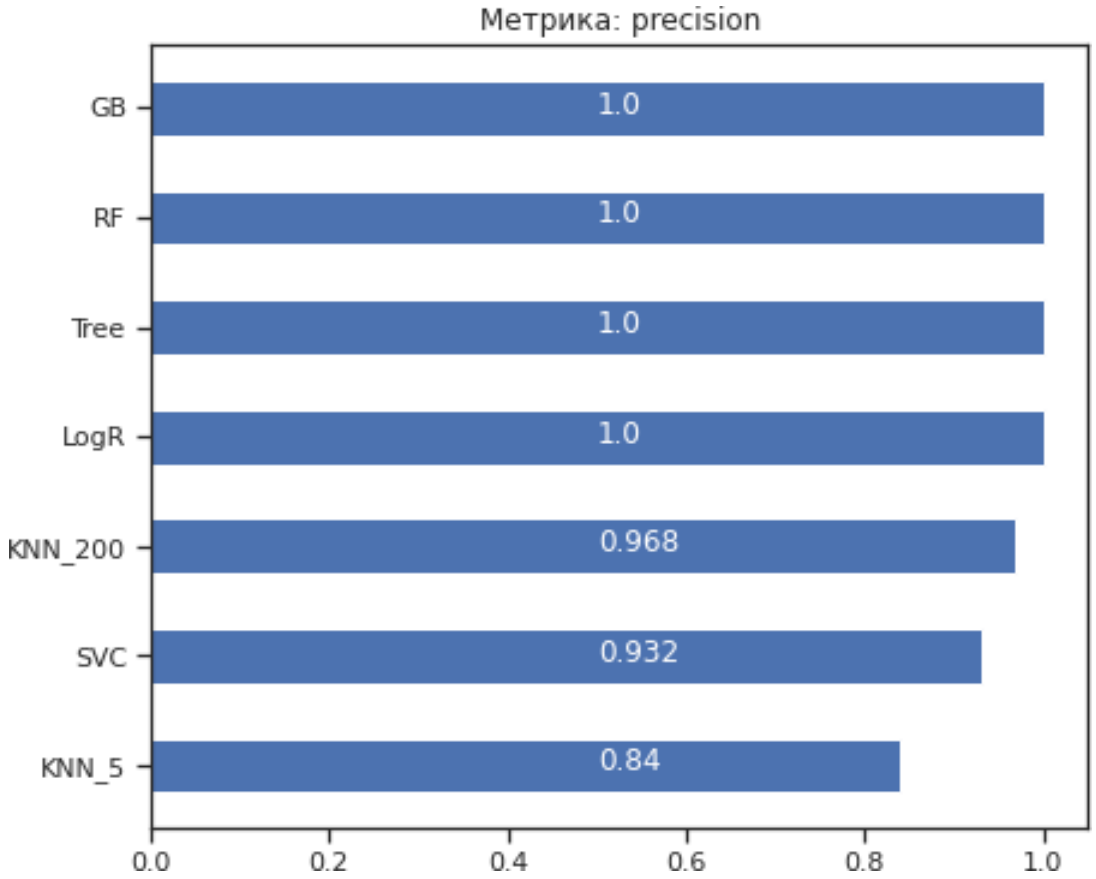

Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.90)
0.0
0.2
0.4
0.6
0.8
1.0
0.99
0.011
0.45
0.55
True label
Predicted label
0
1
0.8
0.6
0.4
0.2

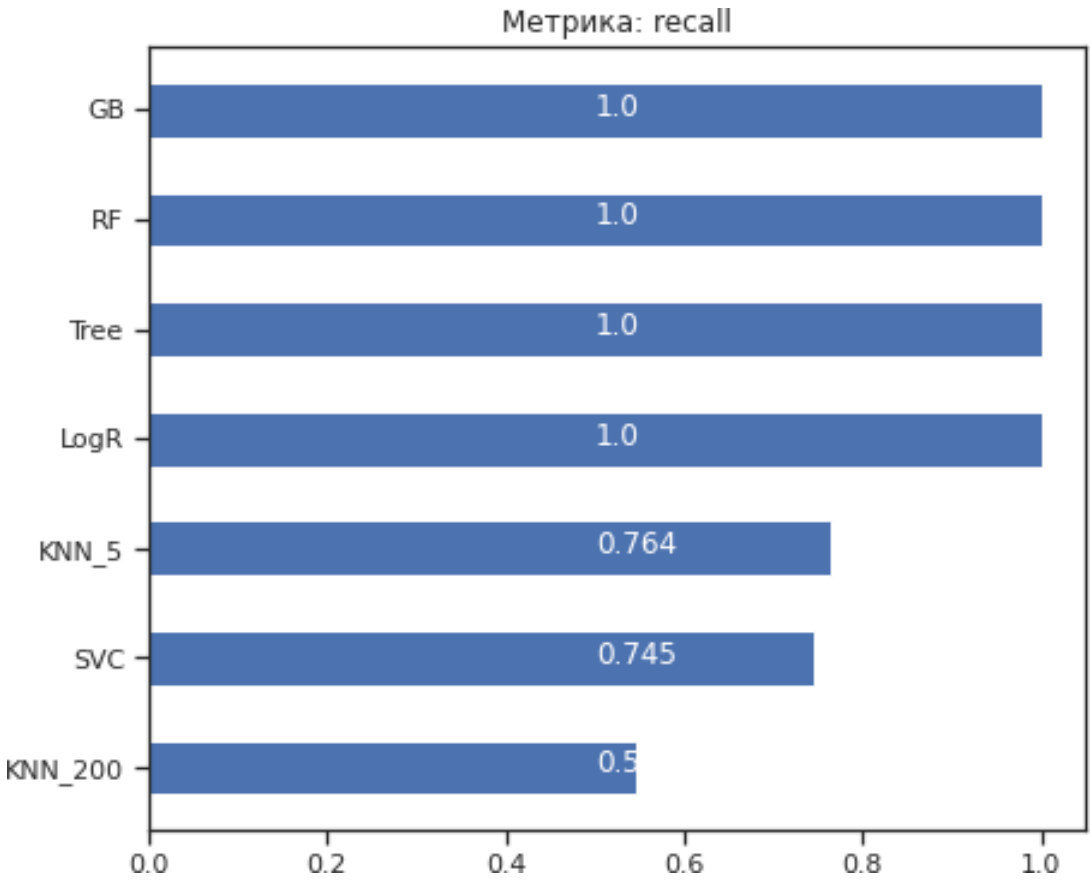
Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей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

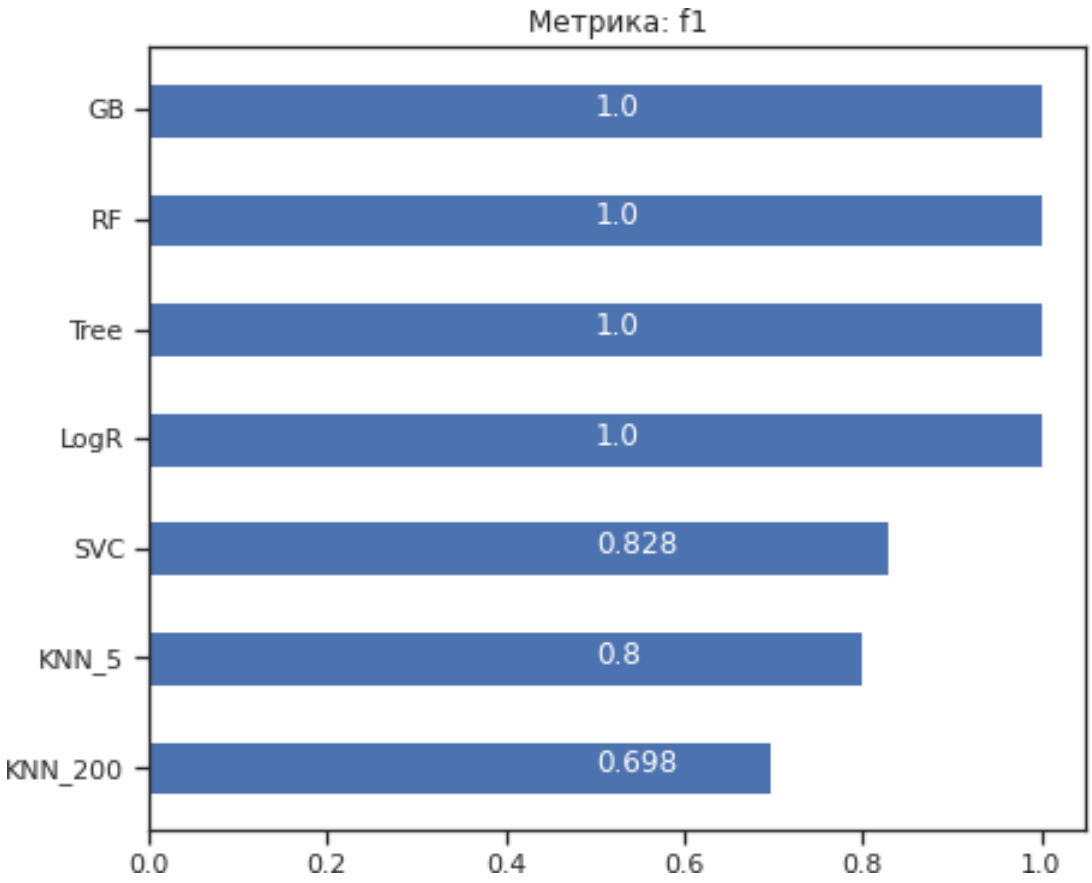
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

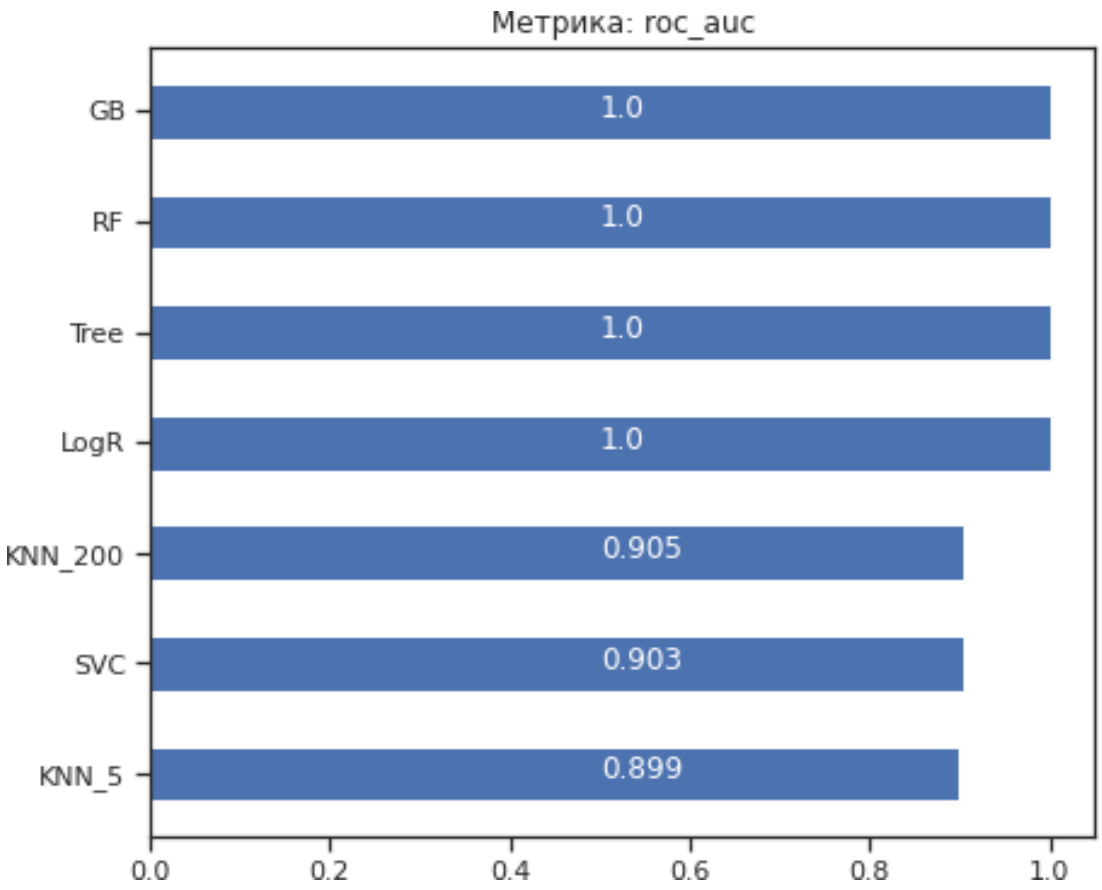
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

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




Метрика: recall
GB
RF
Tree
LogR
KNN_5
SVC
KNN_200
1.0
1.0
1.0
1.0
0.764
0.745
0.5
0.0
0.2
0.4
0.6
0.8
1.0


Метрика: f1
GB
RF
Tree
LogR
SVC
KNN_5
KNN_200
1.0
1.0
1.0
1.0
0.828
0.8
0.698
0.0
0.2
0.4
0.6
0.8
1.0



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

Заключение

В ходе выполнения работы был проведен разведочных анализ данных, исследованы зависимости при помощи корреляционного анализа для дальнейшего выбора оптимальных моделей. Категориальные признаки были закодированы, также проведено масштабирование данных. Построенные модели были проанализированы, определены наилучшие в рамках данной задачи при помощи соответствующих метрик.

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/