Задание

DONE

```
1) Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного наб
2) Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структур
3) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масшт
4) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности пос
5) Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и
6) Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо ис
7) Формирование обучающей и тестовой выборок на основе исходного набора данных.
8) Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Произ
9) Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации
10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества пол
11) Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сра
```

1. Выбор и подготовка набора данных

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

В качестве набора данных мы будем использовать набор данных по прогнозирование сердечной недостаточности https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data

Эта задача является очень актуальной для создания т.к. сердечной недостаточности являются причиной смерти номер 1 в мире, унося, по оценкам, 17,9 миллиона жизней каждый год, что составляет 31% всех смертей в мире.

Датасет состоит из файла:

```
heart_failure_clinical_records_dataset.txt - выборка
```

Каждый файл содержит следующие колонки:

```
age - возраст (года)
```

```
creatinine_phosphokinase - Уровень фермента КФК в крови (мкг/л)
diabetes - Если у пациента диабет (логическое значение)
ejection_fraction - Процент крови, покидающей сердце при каждом сокращении (в процентах)
high_blood_pressure - Если у пациента гипертония (логическое значение)
platelets - Тромбоциты в крови (килограмм тромбоцитов/мл)
serum_creatinine - Уровень сывороточного креатинина в крови (мг/дл)
serum_sodium - Уровень натрия в сыворотке крови (мэкв/л)
sex - Woman or man (binary)
smoking - Если пациент курит или нет (логическое значение)
time - Период наблюдения (дни)
DEATH_EVENT - Если пациент умер в течение периода наблюдения (логическое значение)
```

В рассматриваемом примере будем решать обе задачу классификации:

```
Для решения задачи классификации в качестве целевого признака будем использовать "DEATH_EVENT". По
```

Импорт библиотек

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

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.io as pio
import plotly.express as px
import plotly.figure_factory as ff
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, confusion_matrix, plo
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from scipy.stats import randint, uniform
```

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, ConfusionMatri

# скроем предупреждения о возможных ошибках для лучшей читаемости
import warnings
warnings.filterwarnings('ignore')
```

Дополнительные библиотеки для дополнения отчета

Голова таблицы

```
data = pd.read_csv('./heart_failure_clinical_records_dataset.csv')
data.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pr
0	75.0	0	582	0	20	
1	55.0	0	7861	0	38	
2	65.0	0	146	0	20	
3	50.0	1	111	0	20	
4	65.0	1	160	1	20	

```
data.describe()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction
count	299.000000	299.000000	299.000000	299.000000	299.000000
mean	60.833893	0.431438	581.839465	0.418060	38.083612
std	11.894809	0.496107	970.287881	0.494067	11.834841
min	40.000000	0.000000	23.000000	0.000000	14.000000
25%	51.000000	0.000000	116.500000	0.000000	30.000000
50%	60.000000	0.000000	250.000000	0.000000	38.000000
75%	70.000000	1.000000	582.000000	1.000000	45.000000
max	95.000000	1.000000	7861.000000	1.000000	80.000000

```
print("Размер набора:")
print(f'В датасете {data.shape[0]} строк и {data.shape[1]} колонок.')
```

```
Размер набора:
В датасете 299 строк и 13 колонок.
```

Проведем удаление лишних столбцов. Просмотрев все столбцы, оказалось, что они все нужны.

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

Проверим пропуски:

```
data.isnull().sum()
```

```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

Как видим, пропуски отсутствуют

Проверим датасет на сбалансированность.

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

```
0    203
1     96
Name: DEATH_EVENT, dtype: int64
```

```
total = data.shape[0]
class_0, class_1 = data['DEATH_EVENT'].value_counts()
print('Сердечная недостаточность у {}%, а ее отсутствие составляет {}% в представленной вы
```

```
print('Сердечная недостаточность у {}% а ее отсутствие составляет {}% в представленно
      .format(round(class_1 / total, 4)*100, round(class_0 / total, 4)*100))
```

```
Сердечная недостаточность у 32.11%, а ее отсутствие составляет 67.89% в представленно
```

Вывод. Дисбаланс классов присутствует, но является приемлемым.

Построение графиков для понимания структуры данных

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

```
<seaborn.axisgrid.PairGrid at 0x18ca3572a30>
```

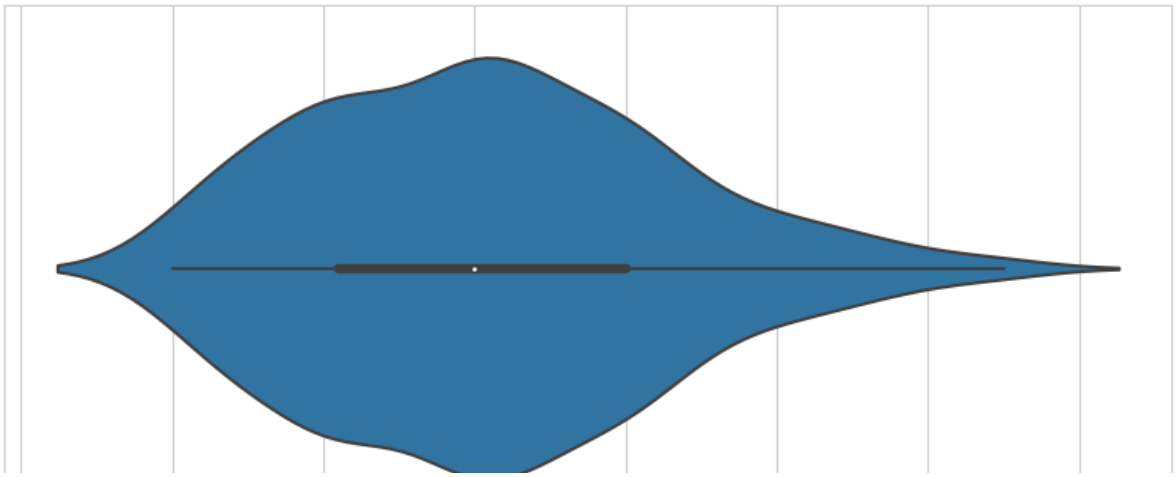
Убедимся, что целевой признак для задачи бинарной классификации содержит только 0 и 1

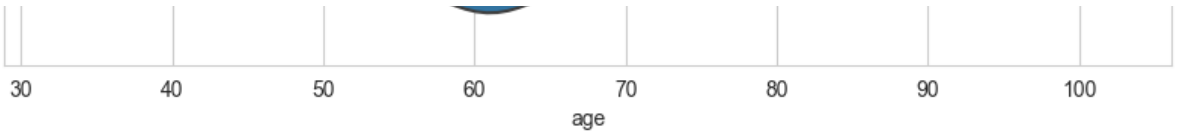
```
data['DEATH_EVENT'].unique()
```

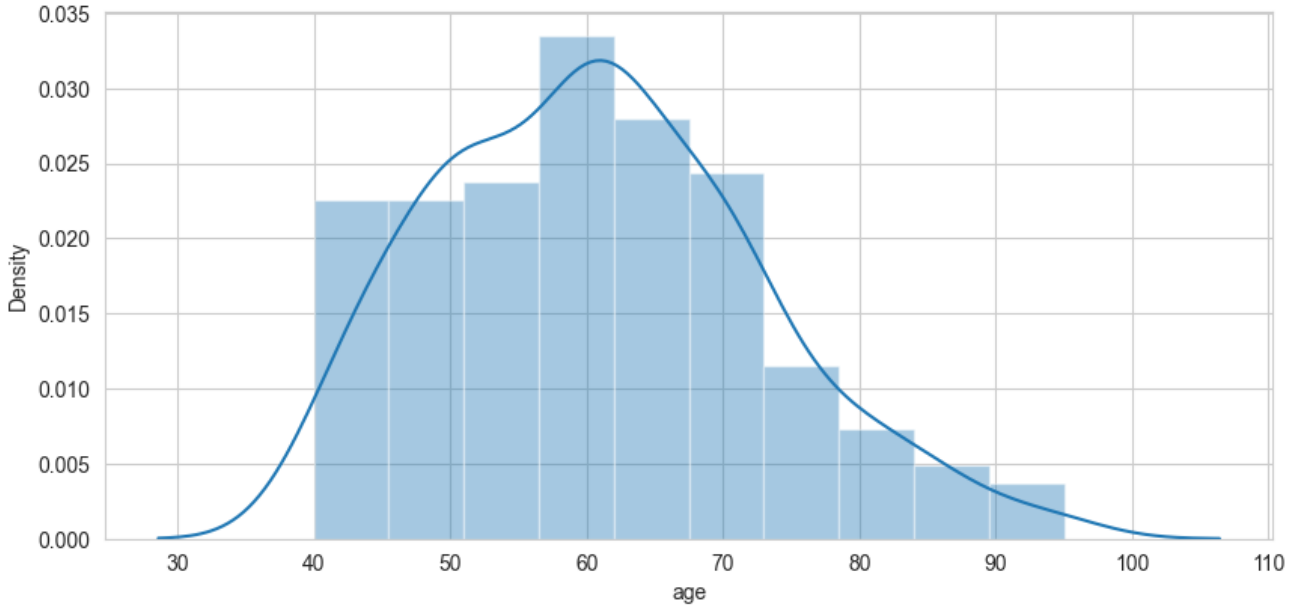
```
array([1, 0], dtype=int64)
```

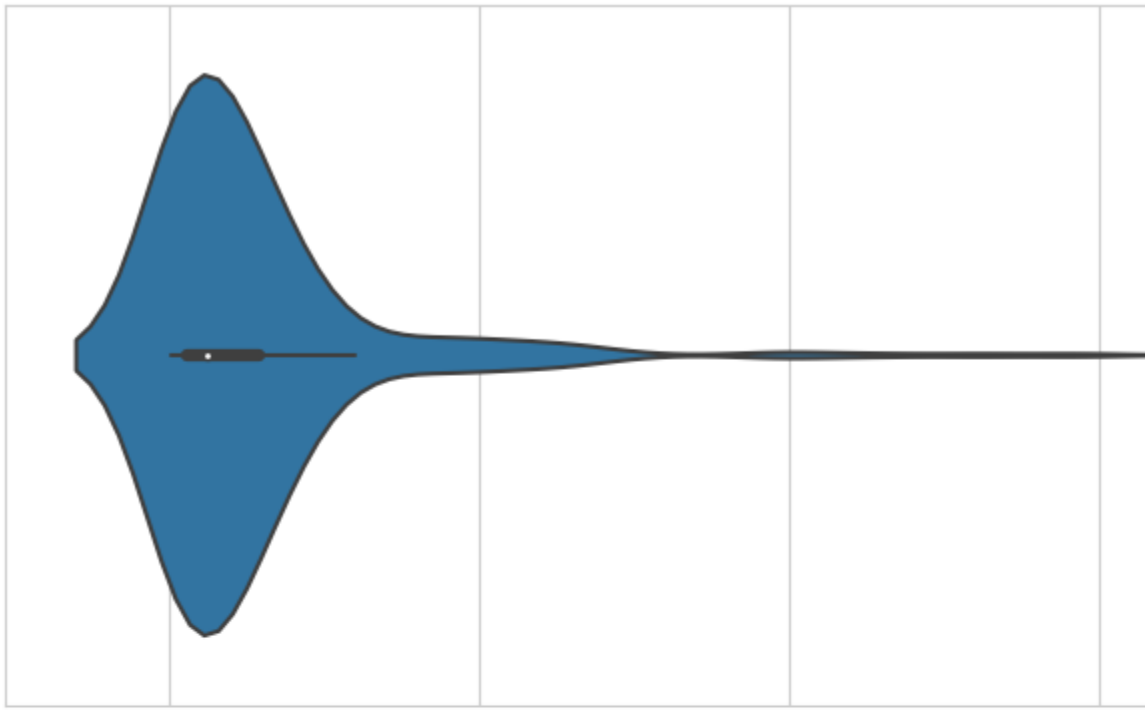
Скрипичные диаграммы для числовых колонок

```
for col in ['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_cre

    fig, ax = plt.subplots(2, 1, figsize=(10,10))
    sns.violinplot(ax=ax[0], x=data[col])
    sns.distplot(data[col], ax=ax[1])
```







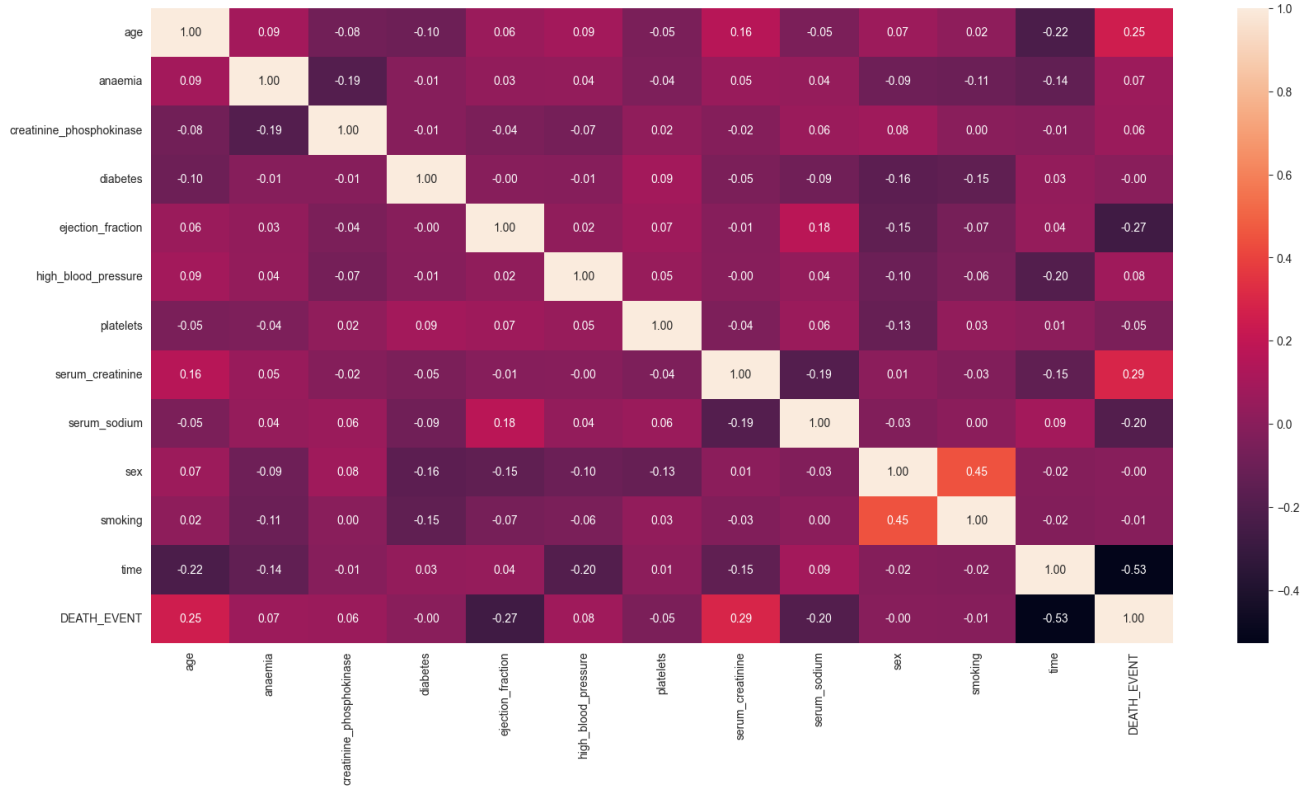


4. (Перед пунктом 3) Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

```
fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(20,10))
```

```
sns.heatmap(data.corr(), annot=True, fmt='.2f')
```

```
<AxesSubplot:>
```



Мы не удалили некоторые категориальные параметры, которые могут влиять на общую картину.

Проверим их корелляцию с другими параметрами.

```
fig2 = px.violin(data, y="age", x="sex", color="DEATH_EVENT", box=True, points="all", hover
fig2.update_layout(title_text="Analysis in Age and Sex on Survival Status")

fig2.show()
```

```
fig2 = px.violin(data, y="age", x="smoking", color="DEATH_EVENT", box=True, points="all", h
fig2.update_layout(title_text="Analysis in Age and Smoking on Survival Status")

fig2.show()
```

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

```
fig2 = px.violin(data, y="age", x="diabetes", color="DEATH_EVENT", box=True, points="all",
fig2.update_layout(title_text="Analysis in Age and Diabetes on Survival Status")

fig2.show()
```

На основе корреляционной матрицы и других графиков можно сделать следующие выводы:

```
1) Целевой признак классификации "DEATH_EVENT" наиболее сильно коррелирует с возрастом (0,25); про

2) У нас нет признаков которые имеют корреляцию, близкую по модулю к 1, поэтому никакие признаки н

3) Однако у нас существуют признаки, которые близки по модулю к 0 по отношению к целевому параметр
    a) diabetes - Если у пациента диабет                                (0.00)
    b) sex - Human or man (binary)                                      (0.00)
```

```
    b) sex - woman or man (binary)                                  (0.00)
    c) smoking - Если пациент курит или нет (логическое значение)   (0.01)

 4) Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между

 5) В исследовании учавствовало меньше женщин, чем мужчин.
```

3. (После пункта 4) Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

```
data.dtypes
```

```
    age                         float64
    anaemia                       int64
    creatinine_phosphokinase      int64
    diabetes                      int64
    ejection_fraction             int64
    high_blood_pressure           int64
    platelets                   float64
    serum_creatinine            float64
    serum_sodium                  int64
    sex                           int64
    smoking                       int64
    time                          int64
    DEATH_EVENT                   int64
    dtype: object
```

Вспомогательные признаки для улучшения качества моделей мы строить не будем.

```
# Числовые колонки для масштабирования
scale_cols = ['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_c
```

```
data_scaled = data
```

```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_scaled[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_scaled[new_col_name] = sc1_data[:,i]
```

```
data_scaled.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pr
0	75.0	0	582	0	20	
1	55.0	0	7861	0	38	
2	65.0	0	146	0	20	
3	50.0	1	111	0	20	
4	65.0	1	160	1	20	

Теперь удалим ненормализованные колонки

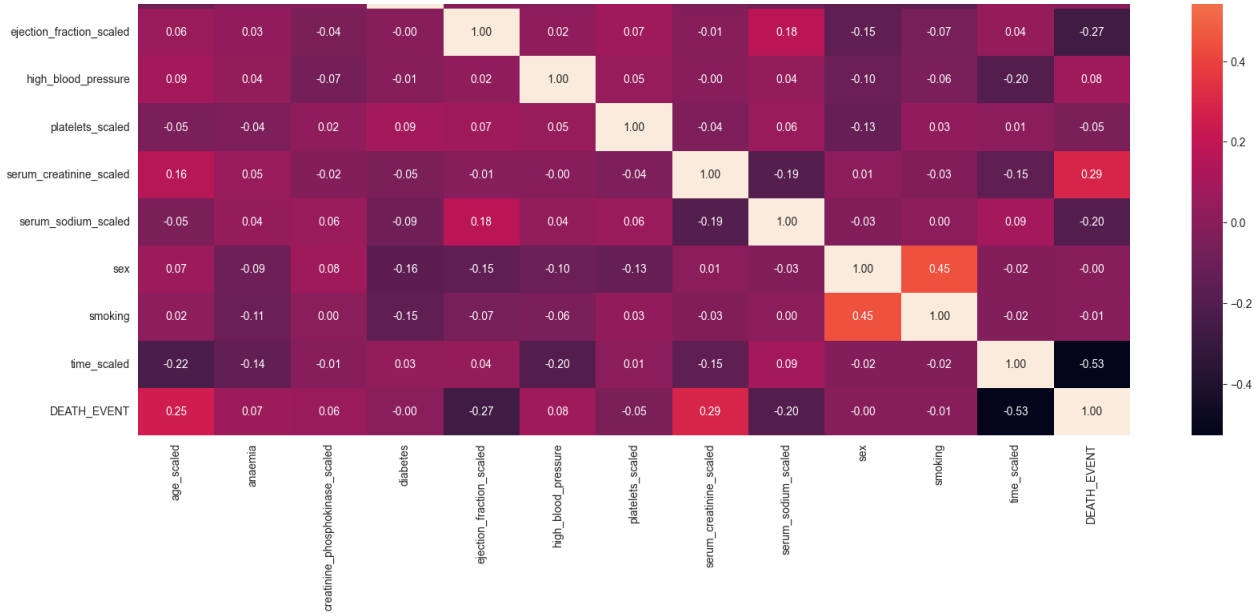
```
data_scaled = data_scaled.drop(scale_cols, axis=1)
data_scaled.head()
```

	anaemia	diabetes	high_blood_pressure	sex	smoking	DEATH_EVENT	age_scaled	cre
0	0	0	1	1	0	1	0.636364	
1	0	0	0	1	0	1	0.272727	
2	0	0	0	1	1	1	0.454545	
3	1	0	0	1	0	1	0.181818	
4	1	1	0	0	0	1	0.454545	

```
data_scaled = data_scaled.reindex(columns=['age_scaled','anaemia','creatinine_phosphokinase
```

```
fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(20,10))
sns.heatmap(data_scaled.corr(), annot=True, fmt='.2f')
```

```
<AxesSubplot:>
```



Так как таблица корелляции не была изменена после нормализации, мы можем продолжить работу.

5. 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".

$precision = \frac{TP}{TP+FP}$

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

Используется функция precision_score.

Метрика recall (полнота):

$recall = \frac{TP}{TP+FN}$

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция recall_score.

Метрика F_1-мера

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

$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{precision + recall}$

где β определяет вес точности в метрике.

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

$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

Для вычисления используется функция f1_score.

Метрика ROC AUC

Метрика ROC AUC

Основана на вычислении следующих характеристик:

$TPR = \frac{TP}{TP+FN}$ - True Positive Rate, откладывается по оси ординат. Совпадает с recall.

$FPR = \frac{FP}{FP+TN}$ - 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, inp
        # 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()
```

6. Выбор наиболее подходящих моделей для решения задачи классификации

Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.

Для задачи классификации будем использовать следующие модели:

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

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

```
X = data_scaled.drop('DEATH_EVENT', axis=1)
Y = data_scaled['DEATH_EVENT']
```

```
[    data_scaled[ 'Class_train' ]
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=2)
print('{}, {}'.format(X_train.shape, X_test.shape))
print('{}, {}'.format(Y_train.shape, Y_test.shape))
```

```
     (209, 12), (90, 12)
     (209,), (90,)
```

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.

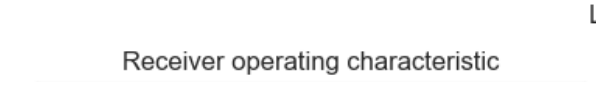
```
models =    {'LogR': LogisticRegression(),
             'KNN_3':KNeighborsClassifier(n_neighbors=3),
             'Tree':DecisionTreeClassifier(),
             'RF':RandomForestClassifier(),
             'GB':GradientBoostingClassifier()}
```

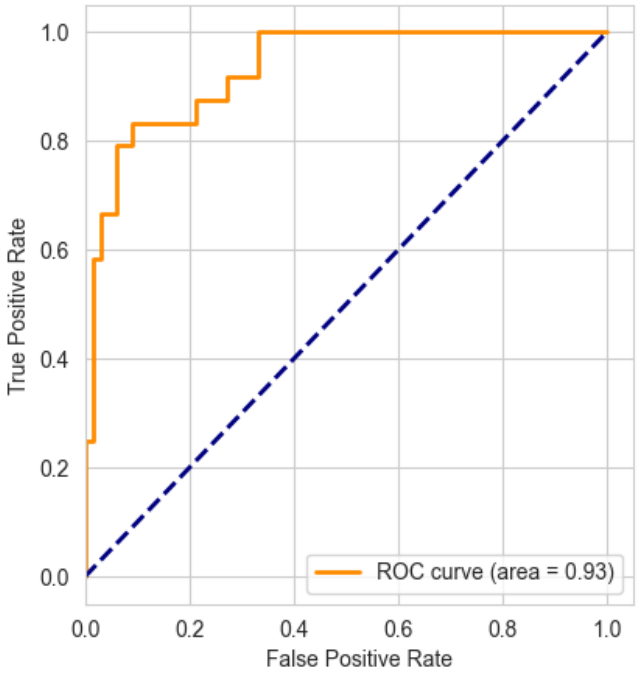
```
clasMetricLogger = MetricLogger()
accuracies = {}
```

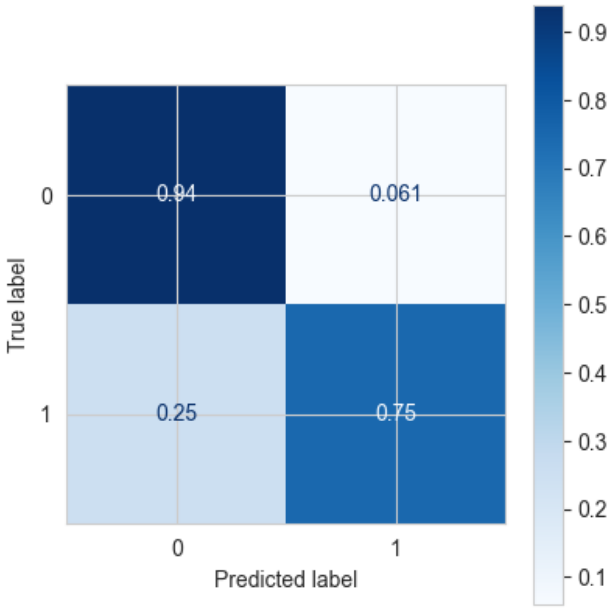
```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, ax, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    #plt.figure()
    lw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

```
clas_X_train = X_train
clas_Y_train = Y_train
```

```
clas_X_test = X_test
clas_Y_test = Y_test


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 models.items():
    model.fit(X_train,Y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,1]
    cm = confusion_matrix(Y_test, y_pred)


    clas_train_model(model_name, model, clasMetricLogger)


    print(classification_report(Y_test, y_pred))
    print(f'ROC AUC score: {roc_auc_score(Y_test, y_prob)}')
    print('Accuracy Score: ',accuracy_score(Y_test, y_pred))

    acc = accuracy_score(Y_test, y_pred)*100
    accuracies[model_name] = acc
```



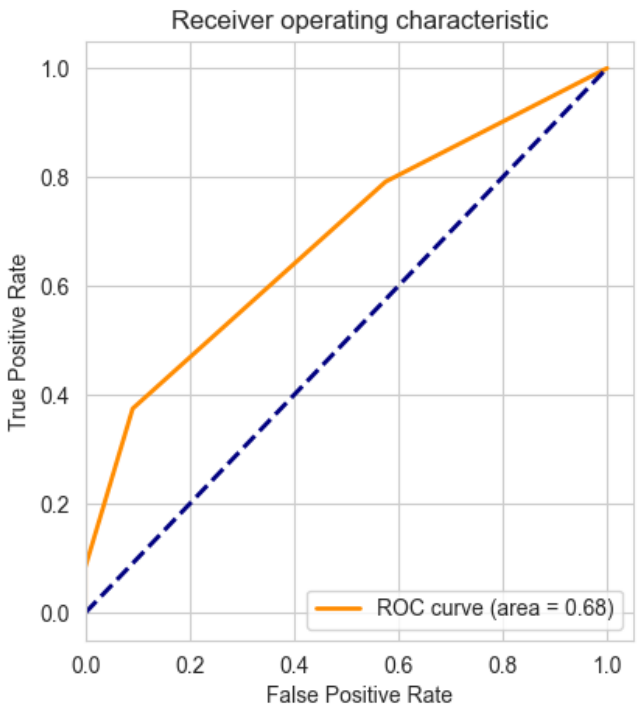


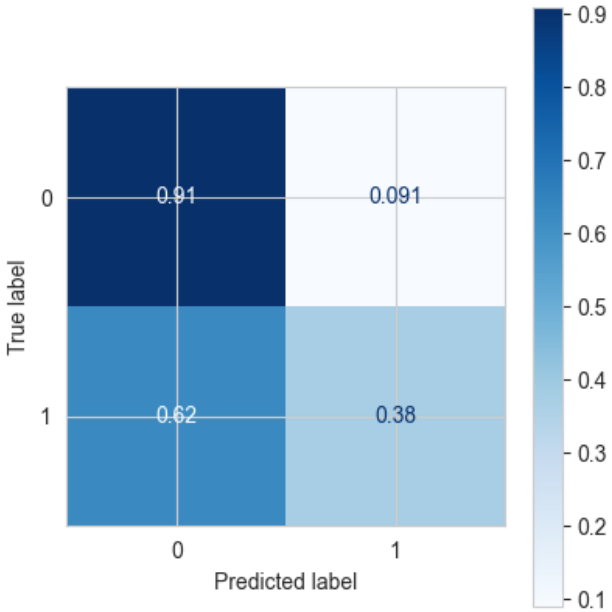


	precision	recall	f1-score	support
0	0.91	0.94	0.93	66
1	0.82	0.75	0.78	24
accuracy			0.89	90
macro avg	0.86	0.84	0.85	90
weighted avg	0.89	0.89	0.89	90

```
ROC AUC score: 0.9330808080808082
Accuracy Score:  0.8888888888888888
```

KNN_3





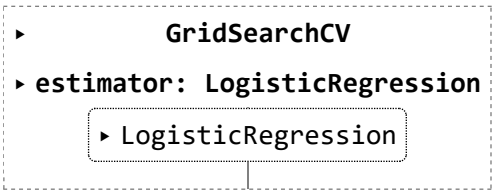
	precision	recall	f1-score	support

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.

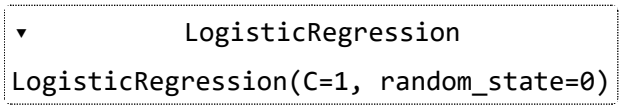
```
n_range_list = list(range(0,300,10))
n_range_list[0] = 1

KN_range_list = list(range(0,150,1))
KN_range_list[0] = 1

thirty_range_list = list(range(10,50,1))


grid_models = [(LogisticRegression(),[{'C':[0.25,0.5,0.75,1],'random_state':[0]}]),
               (KNeighborsClassifier(),[{'n_neighbors':KN_range_list}]),
               (DecisionTreeClassifier(),[{'criterion':['gini','entropy'],'random_state':[0
               (RandomForestClassifier(),[{'n_estimators':n_range_list,'criterion':['gini',
               (GradientBoostingClassifier(),[{'n_estimators':n_range_list,'criterion':['fr
```

Подбор по примеру курса

```
first_grid = GridSearchCV(LogisticRegression(),param_grid = [{'C':[0.25,0.5,0.75,1],'randor
first_grid.fit(clas_X_train, clas_Y_train)
```



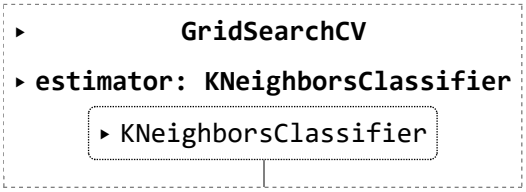

```
first_grid.best_estimator_
```



```
first_grid.best_score_
```

```
0.789198606271777
```

```
second_grid = GridSearchCV(KNeighborsClassifier(),param_grid = [{'n_neighbors':KN_range_lis
second_grid.fit(clas_X_train, clas_Y_train)
```



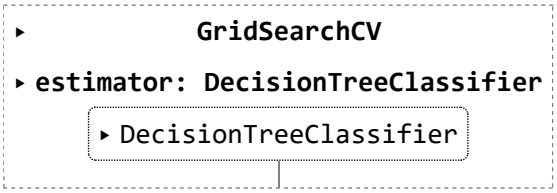

```
second_grid.best_estimator_
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=11)
```

```
second_grid.best_score_
```

```
0.7227642276422765
```

```
third_grid = GridSearchCV(DecisionTreeClassifier(),param_grid = [{'criterion':['gini','entr
third_grid.fit(clas_X_train, clas_Y_train)
```




```
third_grid.best_estimator_
```

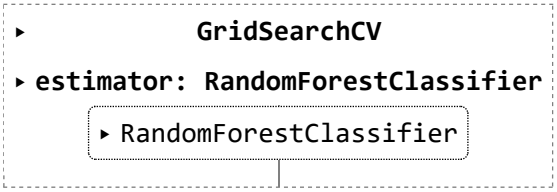
```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

```
third_grid.best_score_
```

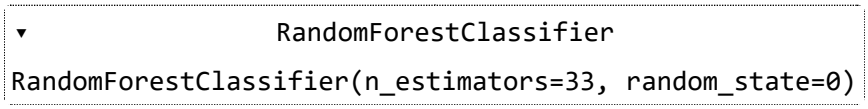
```
0.770267131242741
```

```
fourth_grid = GridSearchCV(RandomForestClassifier(),param_grid = [{'n_estimators':thirty_ra
```

```
fourth_grid.fit(clas_X_train, clas_Y_train)
```



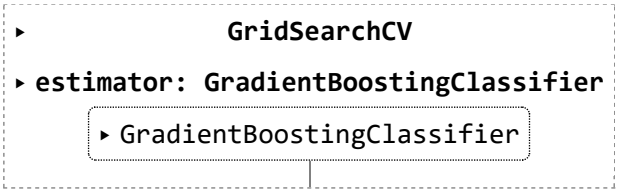

```
fourth_grid.best_estimator_
```



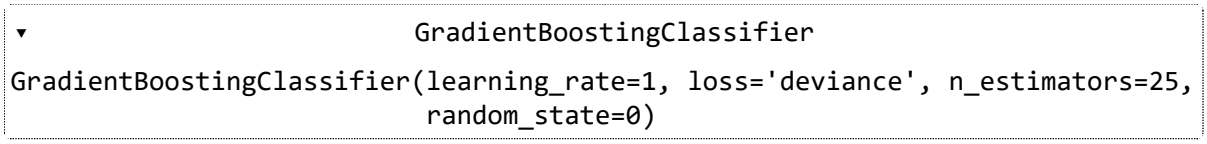

```
fourth_grid.best_score_
```

```
0.851800232288037
```

```
# GradientBoostingClassifier(),[{'n_estimators':n_range_list,'criterion':['friedman_mse','r
fifth_grid = GridSearchCV(GradientBoostingClassifier(),param_grid = [{'n_estimators':thirty
fifth_grid.fit(clas_X_train, clas_Y_train)
```




```
fifth_grid.best_estimator_
```




```
fifth_grid.best_score_
```

```
0.827526132404181
```

Продолжим поиск по моему методу (через цикл)

```
for i,j in grid_models:
    grid = GridSearchCV(estimator=i,param_grid = j, scoring = 'accuracy',cv=5)
    grid.fit(X_train, Y_train)
```

```
    best_accuracy = grid.best_score_
    best_param = grid.best_params_
    print('{}:\nBest Accuracy : {:.2f}%'.format(i,best_accuracy*100))
    print('Best Parameters : ',best_param)
    print('')
    print('----------------')
    print('')
```

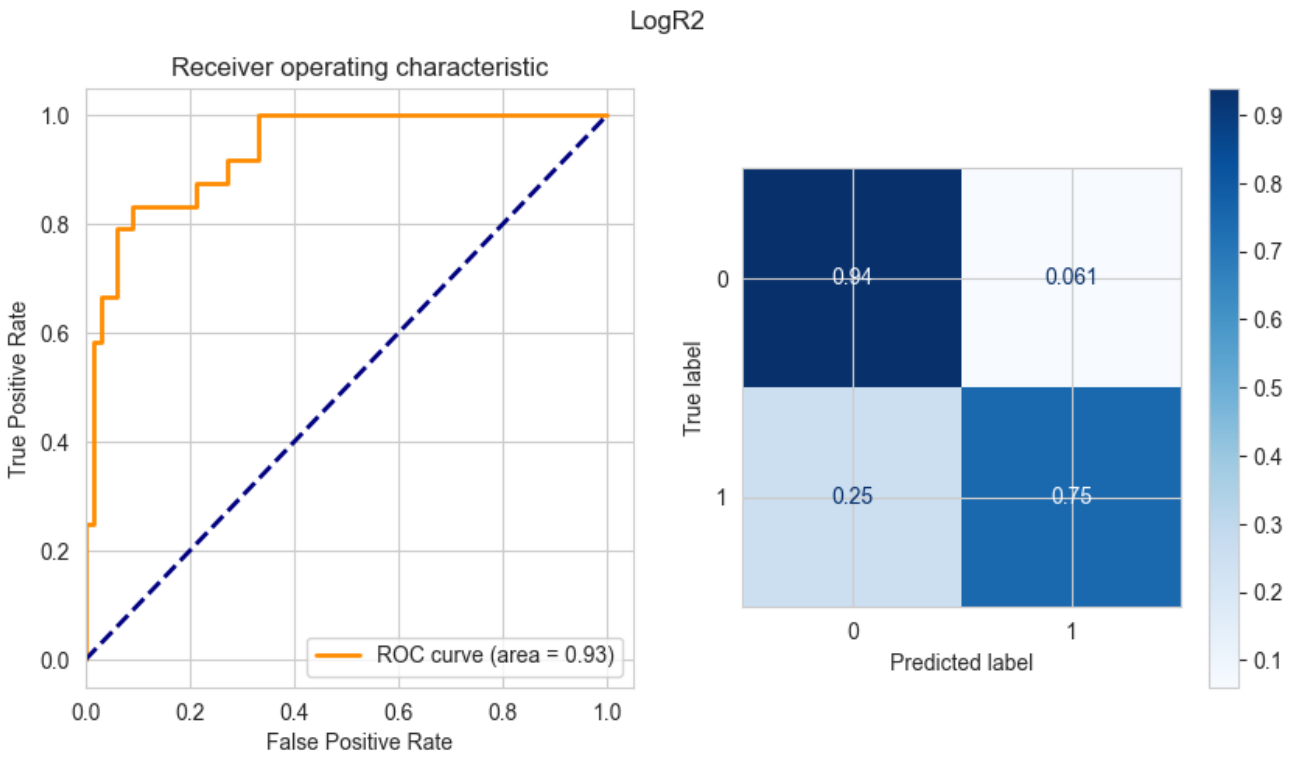
```
 LogisticRegression():
 Best Accuracy : 78.92%
 Best Parameters :  {'C': 1, 'random_state': 0}

 ----------------

 KNeighborsClassifier():
 Best Accuracy : 72.28%
 Best Parameters :  {'n_neighbors': 11}

 ----------------

 DecisionTreeClassifier():
 Best Accuracy : 77.03%
 Best Parameters :  {'criterion': 'gini', 'random_state': 0}

 ----------------

 RandomForestClassifier():
 Best Accuracy : 84.22%
 Best Parameters :  {'criterion': 'gini', 'n_estimators': 30, 'random_state': 0}

 ----------------

 GradientBoostingClassifier():
 Best Accuracy : 82.28%
 Best Parameters :  {'criterion': 'friedman_mse', 'learning_rate': 1, 'loss': 'exponen

 ----------------
```

10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров.

Сравнение качества полученных моделей с качеством baseline-моделей.

```
models_params =    {'LogR2': LogisticRegression(C = 1, random_state = 0),
                    'KNN_14':KNeighborsClassifier(n_neighbors = 14),
                    'Tree0':DecisionTreeClassifier(criterion = 'gini', random_state = 0),
                    'RF33':RandomForestClassifier(criterion = 'gini', n_estimators = 33, ra
```

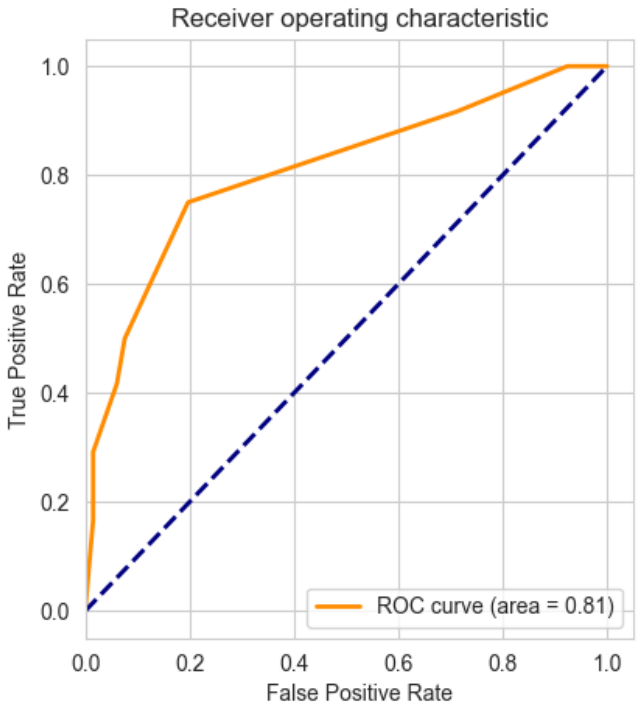
```
                    'GB25':GradientBoostingClassifier(criterion = 'friedman_mse', learning_
params_accuracies = {}


for model_name, model in models_params.items():
    model.fit(X_train,Y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,1]
    cm = confusion_matrix(Y_test, y_pred)


    clas_train_model(model_name, model, clasMetricLogger)


    print(classification_report(Y_test, y_pred))
    print(f'ROC AUC score: {roc_auc_score(Y_test, y_prob)}')
    print('Accuracy Score: ',accuracy_score(Y_test, y_pred))

    param_acc = accuracy_score(Y_test, y_pred)*100
    params_accuracies[model_name] = param_acc
```

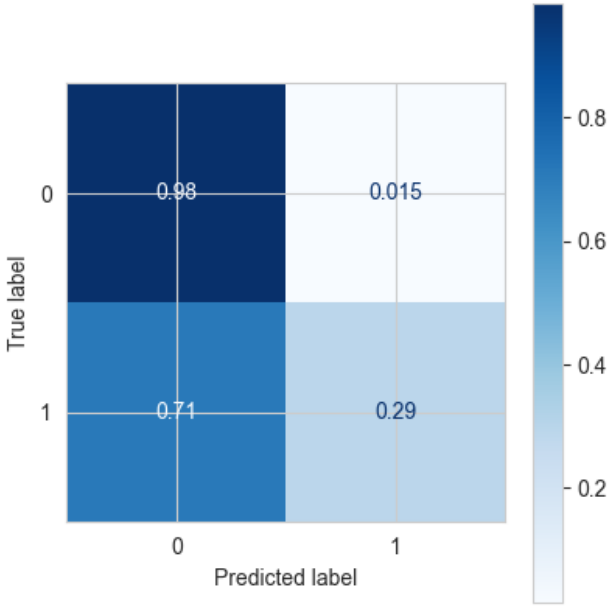



```
              precision    recall  f1-score   support

           0       0.91      0.94      0.93        66
           1       0.82      0.75      0.78        24

    accuracy                           0.89        90
   macro avg       0.86      0.84      0.85        90
weighted avg       0.89      0.89      0.89        90
```

```
ROC AUC score: 0.9330808080808082
Accuracy Score:  0.8888888888888888
```

KNN_14



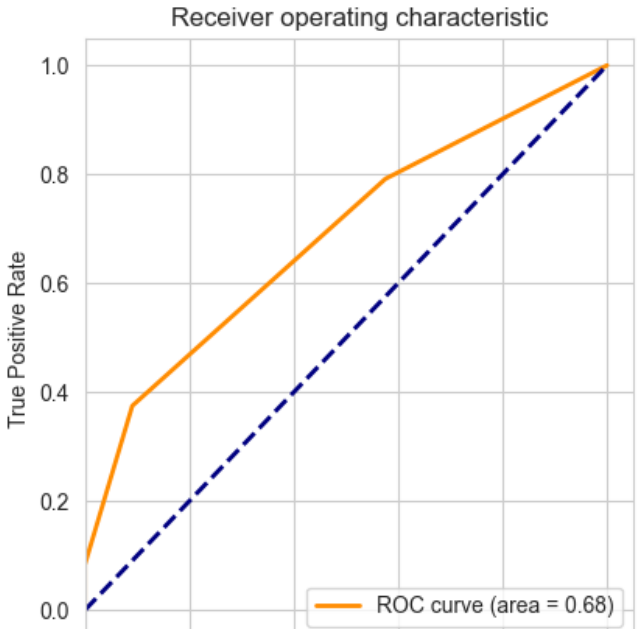


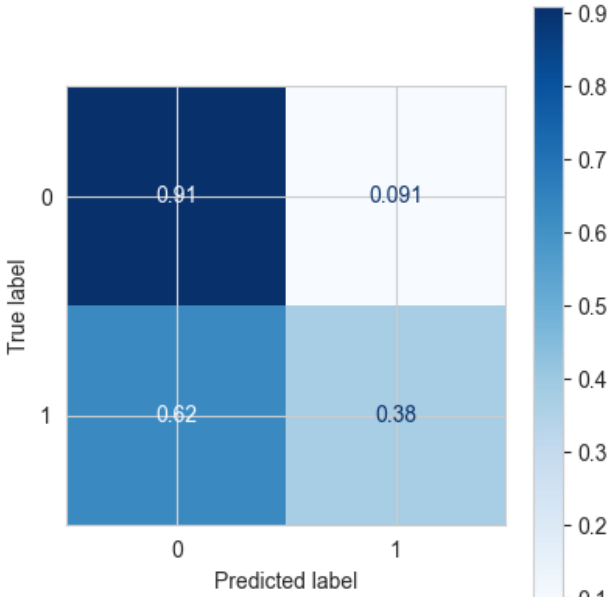

```
              precision    recall  f1-score   support
```

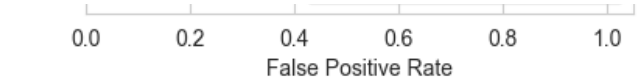
```
clas_models_grid = {'KNN_3':KNeighborsClassifier(n_neighbors=3),
                    str('KNN_' + '11'):second_grid.best_estimator_}
```

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

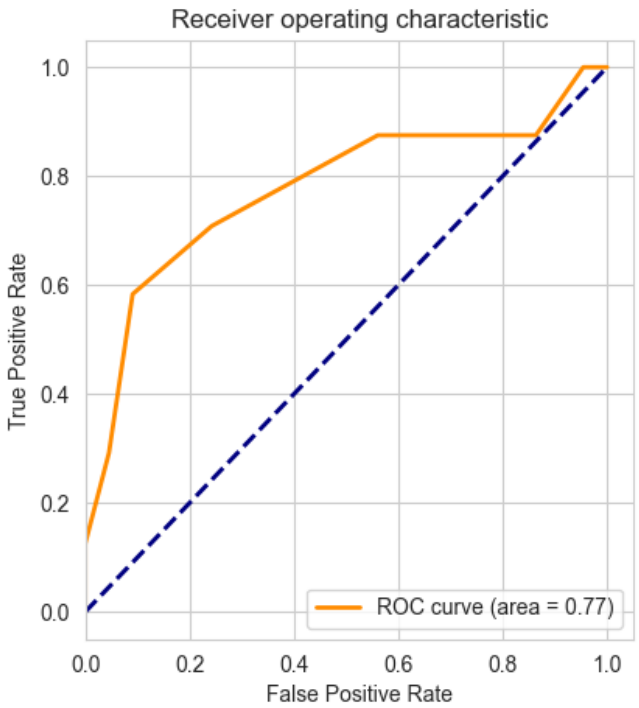
KNN_3

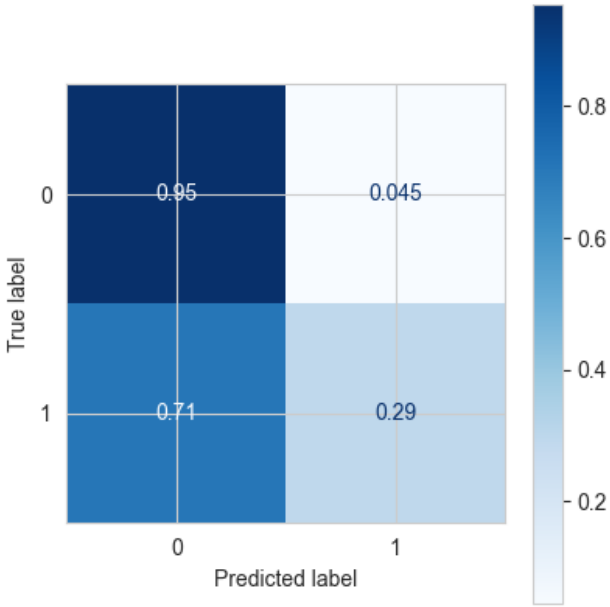






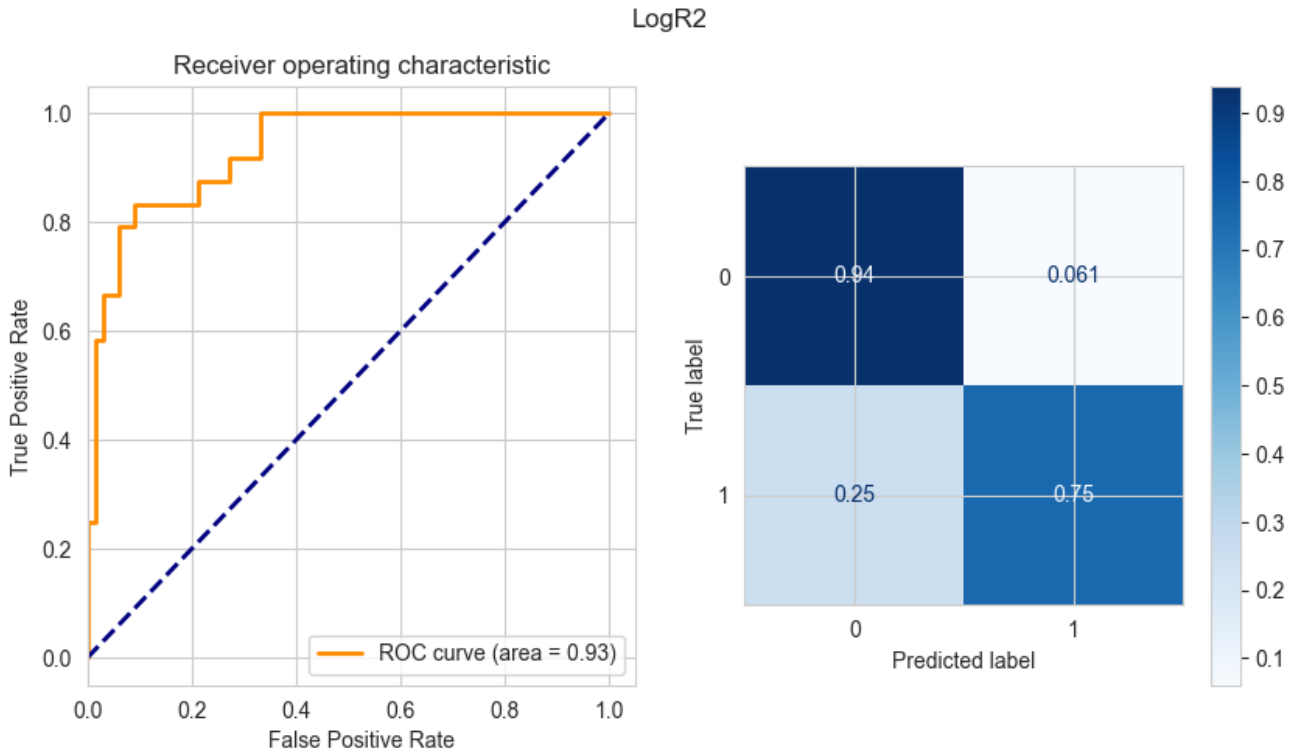
KNN_11



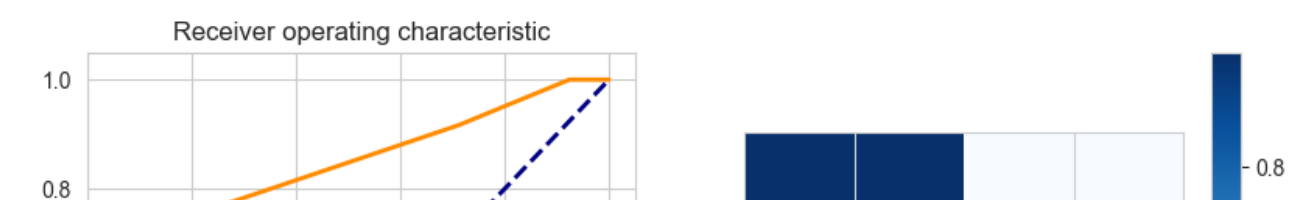


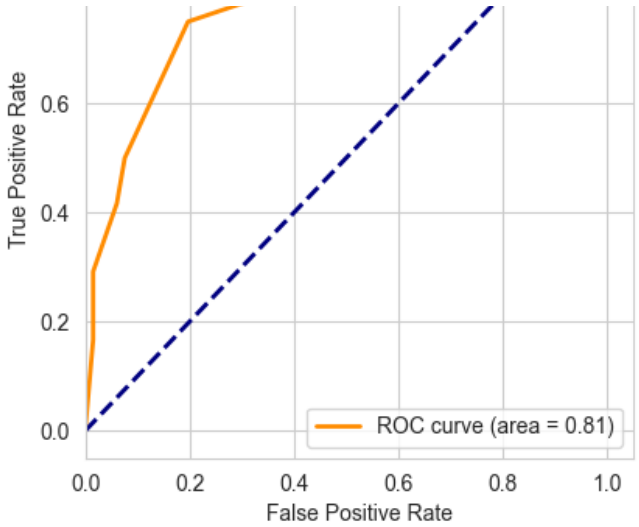

```
models_params =    {'LogR2': LogisticRegression(C = 1, random_state = 0),
                    'KNN_14':KNeighborsClassifier(n_neighbors = 14),
                    'Tree0':DecisionTreeClassifier(criterion = 'gini', random_state = 0),
                    'RF33':RandomForestClassifier(criterion = 'gini', n_estimators = 33, ra
                    'GB25':GradientBoostingClassifier(criterion = 'friedman_mse', learning_
params_accuracies = {}


clasMetricLogger_params = MetricLogger()

for model_name, model in models_params.items():
    model.fit(X_train,Y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:,1]
```

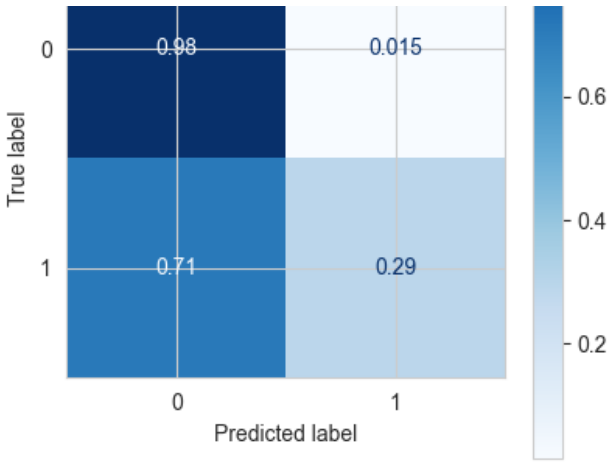
```
cm = confusion_matrix(Y_test, y_pred)


clas_train_model(model_name, model, clasMetricLogger_params)


print(classification_report(Y_test, y_pred))
print(f'ROC AUC score: {roc_auc_score(Y_test, y_prob)}')
print('Accuracy Score: ',accuracy_score(Y_test, y_pred))

param_acc = accuracy_score(Y_test, y_pred)*100
params_accuracies[model_name] = param_acc
```




```
              precision    recall  f1-score   support

           0       0.91      0.94      0.93        66
           1       0.82      0.75      0.78        24

    accuracy                           0.89        90
   macro avg       0.86      0.84      0.85        90
weighted avg       0.89      0.89      0.89        90

ROC AUC score: 0.9330808080808082
Accuracy Score:  0.8888888888888888
```








```
precision    recall  f1-score   support
```

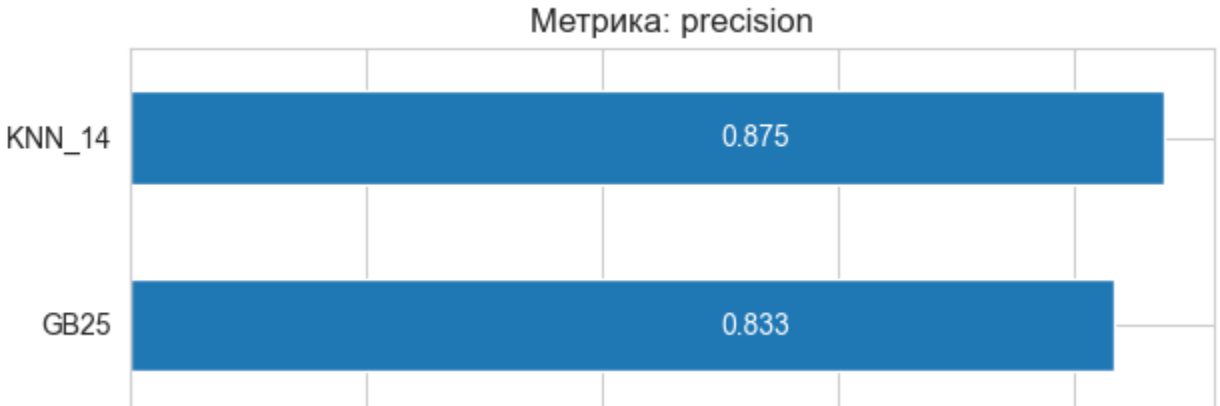
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

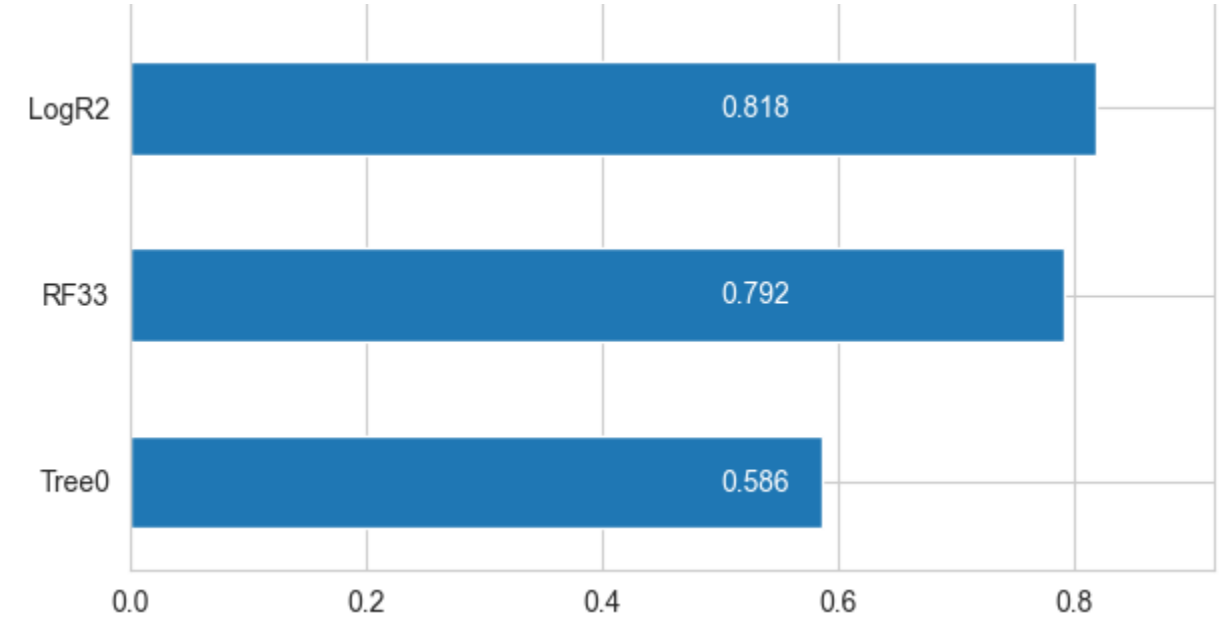
Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется построение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

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

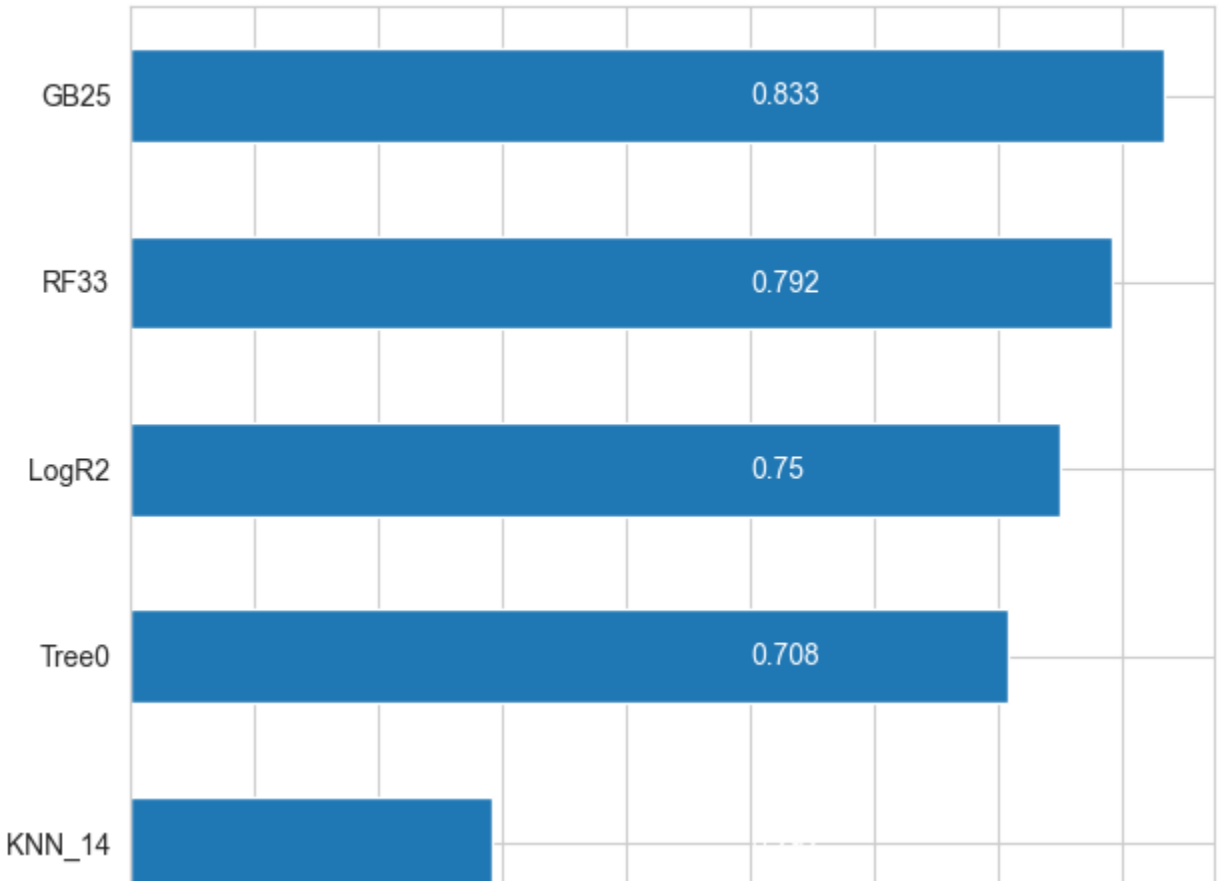
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

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





Метрика: recall



Вывод: на основании трех метрик из четырех используемых, лучшей оказалась модель "Случайный лес".