**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 _____

КАФЕДРА _____

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОМУ ПРОЕКТУ*

## *НА ТЕМУ:*

_____

_____

_____

_____

_____

_____

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____
(Подпись, дата) (И.О.Фамилия)

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

*2020 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 курсового проекта

по дисциплине _____Технологии машинного обучения_____

Студент группы ____ИУ5-64_____

_____Сафин Рустам Равильевич___
(Фамилия, имя, отчество)

Тема курсового проекта _____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к _4_ нед., 50% к _8_ нед., 75% к 12 нед., 100% к 16 нед.
***Задание***_____
_____
_____
_____
_____

***Оформление курсового проекта:***

Расчетно-пояснительная записка на _35__ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « ___ » _____ 20__ г.

**Руководитель курсового проекта** _____ _____
(Подпись, дата) (И.О.Фамилия)

**Студент** _____ _____
(Подпись, дата) (И.О.Фамилия)

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

СОДЕРЖАНИЕ

**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.

## 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.

В качестве набора данных мы будем использовать набор данных для определения механизмов биоконцентрации.

Файл содержит следующие колонки:

1. CAS - уникальный численный идентификатор химических соединений;
2. SMILES - система упрощённого представления молекул в строке ввода;
3. Set - фильтр разделения обучающей и тестовой выборки;
4. nHM - молекулярный дескриптор;
5. piPC09 - молекулярный дескриптор;
6. PCD - молекулярный дескриптор;
7. X2Av - молекулярный дескриптор;
8. MLOGP - молекулярный дескриптор;
9. ON1V - молекулярный дескриптор;
10. N-072 - молекулярный дескриптор;
11. B02[C-N] - молекулярный дескриптор;
12. F04[C-O] - молекулярный дескриптор;
13. Class - класс биоаккумуляции;
14. logBCF - коэффициент биоконцентрации в логарифмических единицах.

Будем решать задачу классификации. В качестве целевого признака возьмем колонку B02[C-N]. Поскольку она содержит только значения 0 или 1, то это задача бинарной классификации.

**Импорт необходимых библиотек**

In [2]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, precision_score, recall_score,
accuracy_score from sklearn.metrics import plot_confusion_matrix, roc_curve
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

%matplotlib inline

# Устанавливаем тип графиков
sns.set(style="ticks")

# Для лучшего качествоа графиков
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

```
# Устанавливаем ширину экрана для отчета
pd.set_option("display.width", 70)
```

**Загрузка данных**

Загрузим файлы датасета в помощью библиотеки Pandas

In [3]:

```
data = pd.read_csv('C:/users/rusta/desktop/tmo/data.csv')
```

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

In [4]:

```
# Первые пять записей набора
данных data.head()
```

Out[4]:

| | CAS | SMILES | Set | nHM | piPC09 | PCD | X2Av | MLOGP | ON1V | N-072 | B02[C-N] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 100-02-7 | O=[N+](c1ccc(cc1)O)[O-] | Train | 0 | 0.0 | 1.49 | 0.14 | 1.35 | 0.72 | 0 | 1 |
| **1** | 100-17-4 | O=[N+](c1ccc(cc1)OC)[O-] | Train | 0 | 0.0 | 1.47 | 0.14 | 1.70 | 0.88 | 0 | 1 |
| **2** | 100-18-5 | c1cc(ccc1C(C)C)C(C)C | Train | 0 | 0.0 | 1.20 | 0.25 | 4.14 | 2.06 | 0 | 0 |
| **3** | 100-25-4 | O=[N+]([O-])c1ccc(cc1)[N+](=O)[O-] | Train | 0 | 0.0 | 1.69 | 0.13 | 1.89 | 0.79 | 0 | 1 |
| **4** | 100-40-3 | C=CC1CCC=CC1 | Train | 0 | 0.0 | 0.52 | 0.25 | 2.65 | 1.31 | 0 | 0 |

In [5]:

```
# Размер датасета (779 - строк, 14 - колонок)
data.shape
```

Out[5]:

```
(779, 14)
```

In [6]:

```
# Список колонок
data.columns
```

Out[6]:

```
Index(['CAS', 'SMILES', 'Set', 'nHM', 'piPC09', 'PCD', 'X2Av',
       'MLOGP', 'ON1V', 'N-072', 'B02[C-N]', 'F04[C-O]', 'Class',
       'logBCF'],
      dtype='object')
```

In [7]:

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

Out[7]:

```
CAS          object
SMILES       object
Set          object
nHM           int64
piPC09      float64
PCD         float64
X2Av        float64
MLOGP       float64
ON1V        float64
N-072         int64
B02[C-N]      int64
F04[C-O]      int64
Class         int64
logBCF      float64
dtype: object
```

In [8]:

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

Out[8]:

```
CAS         0
SMILES      0
Set         0
nHM         0
piPC09      0
PCD         0
X2Av        0
MLOGP       0
ON1V        0
N-072       0
B02[C-N]    0
F04[C-O]    0
Class       0
logBCF      0
dtype: int64
```

Датасет не содержит пропусков данных

In [15]:

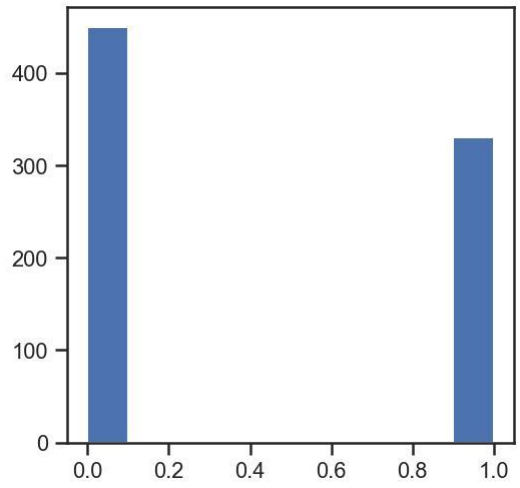
```
# Убедимся, что признак
# для задачи бинарной классификации содержит только 0 и 1
data['B02[C-N]'].unique()
```

Out[15]:

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

In [16]:

```
# Оценим дисбаланс классов для B02[C-N]
fig, ax = plt.subplots(figsize=(4,4))
plt.hist(data['B02[C-N]']) plt.show()
```




In [17]:

```
data['B02[C-N]'].value_counts()
```

Out[17]:

```
0    449
1330
Name: B02[C-N], dtype: int64
```

In [19]:

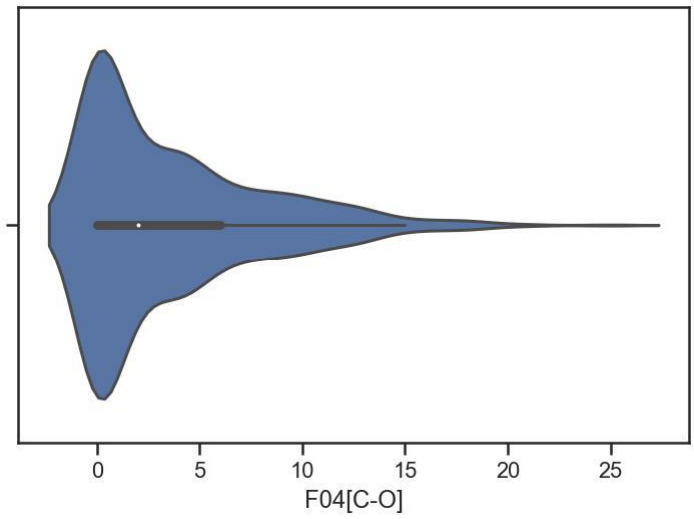
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_1, class_0 = data['B02[C-N]'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_1 / total, 2)*100, round(class_0 / total, 2)*100))
```

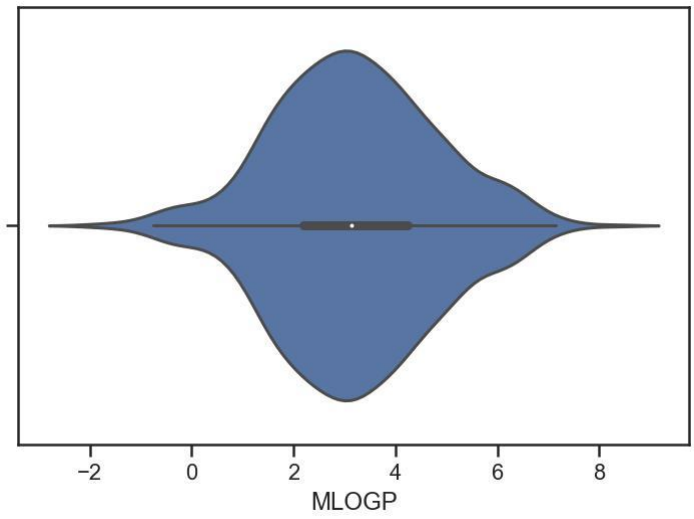
```
Класс 0 составляет 57.99999999999999%, а класс 1 составляет 42.0%.
```

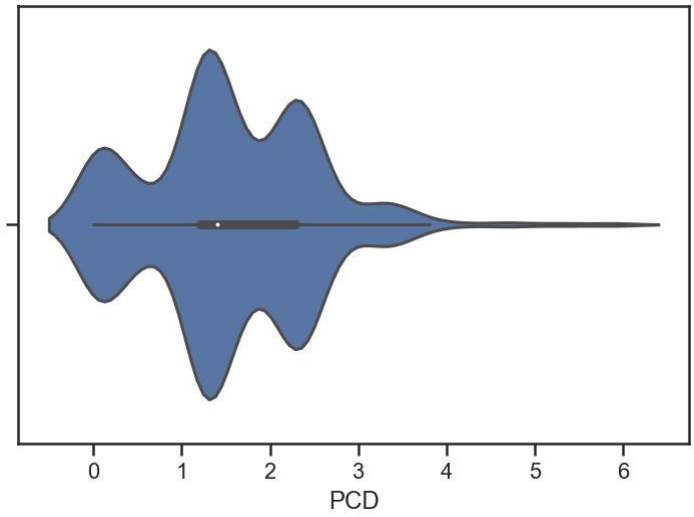
Дисбаланс классов практически отсутствует

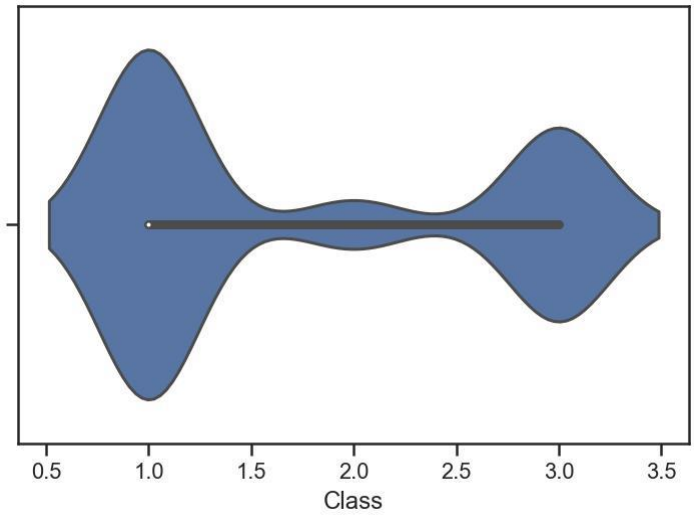
In [20]:

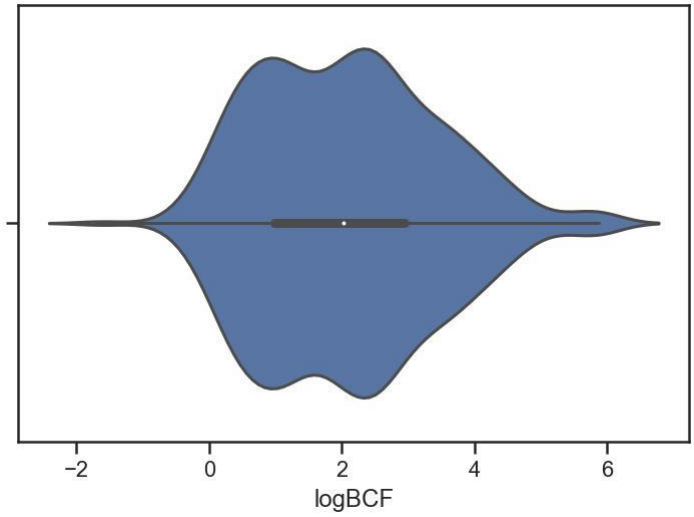
```
# Скрипичные диаграммы для некоторых колонок
for col in ['F04[C-O]', 'MLOGP', 'PCD', 'Class', 'logBCF']:
    sns.violinplot(x=data[col])
    plt.show()
```




0
1
2
3
4
5
6
PCD


0.5
1.0
1.5
2.0
2.5
3.0
3.5
Class




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

Для построения моделей будем использовать все признаки.

Категориальные признаки присутствуют, но их кодирования не требуется, так как все значения в этих строках уникальны и они не окажут влияния.

Вспомогательные признаки для улучшения качества моделей мы строить не будем.

Выполним масштабирование данных.

In [21]:

```
# Числовые колонки для масштабирования
scale_cols = ['nHM', 'piPC09', 'PCD', 'X2Av', 'MLOGP', 'ON1V', 'N-072', 'F04[C-O]',
              'Class', 'logBCF']
```

In [22]:

```
sc = MinMaxScaler()
sc_data = sc.fit_transform(data[scale_cols])
```

In [23]:

```
# 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] = sc_data[:,i]
```

In [24]:

```
data.head()
```

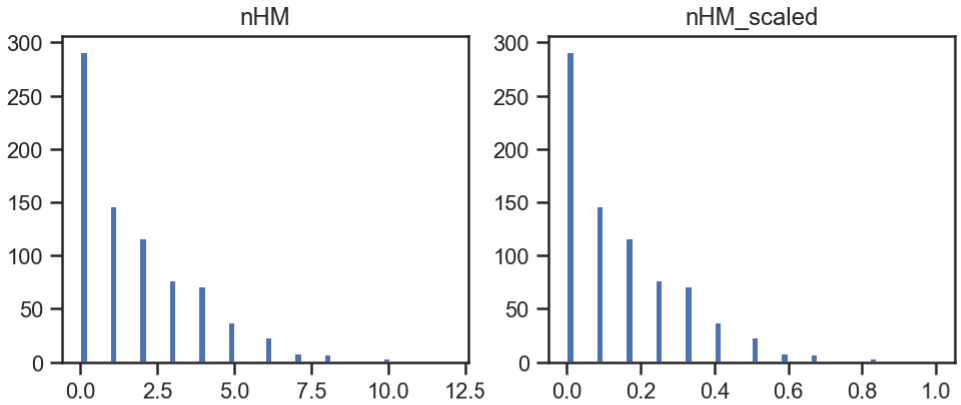
Out[24]:

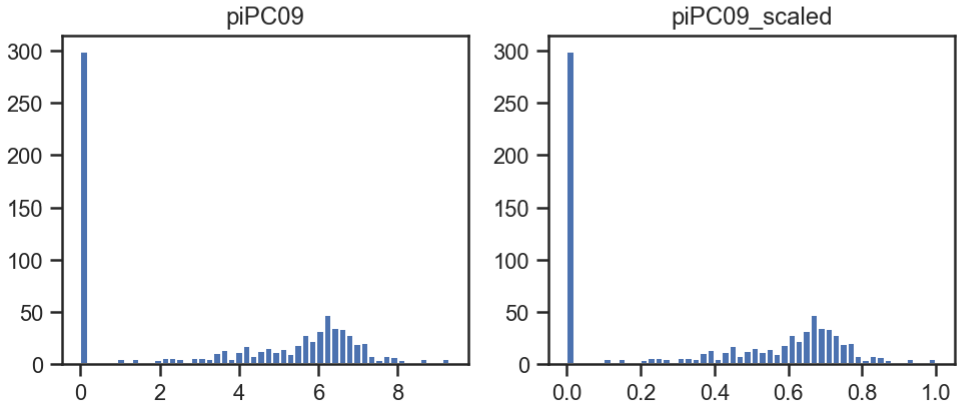
| | CAS | SMILES | Set | nHM | piPC09 | PCD | X2Av | MLOGP | ON1V | N-072 | ... | nHM_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 100-02-7 | O=[N+](c1ccc(cc1)O)[O-] | Train | 0 | 0.0 | 1.49 | 0.14 | 1.35 | 0.72 | 0 | ... | |
| **1** | 100-17-4 | O=[N+](c1ccc(cc1)OC)[O-] | Train | 0 | 0.0 | 1.47 | 0.14 | 1.70 | 0.88 | 0 | ... | |
| **2** | 100-18-5 | c1cc(ccc1C(C)C)C(C)C | Train | 0 | 0.0 | 1.20 | 0.25 | 4.14 | 2.06 | 0 | ... | |
| **3** | 100-25-4 | O=[N+]([O-])c1ccc(cc1)[N+](=O)[O-] | Train | 0 | 0.0 | 1.69 | 0.13 | 1.89 | 0.79 | 0 | ... | |
| **4** | 100-40-3 | C=CC1CCC=CC1 | Train | 0 | 0.0 | 0.52 | 0.25 | 2.65 | 1.31 | 0 | ... | |

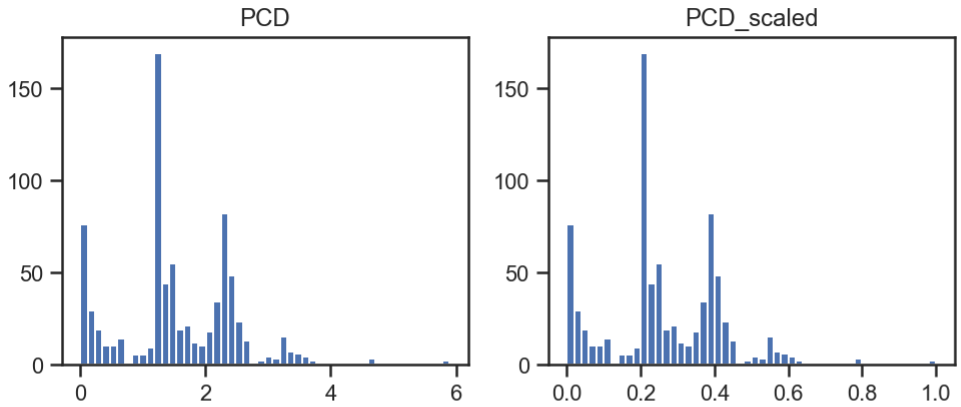
5 rows × 24 columns

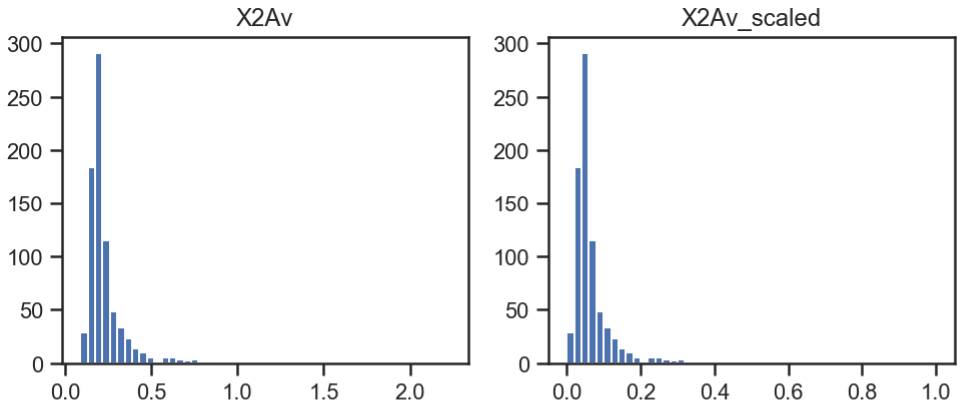
In [25]:

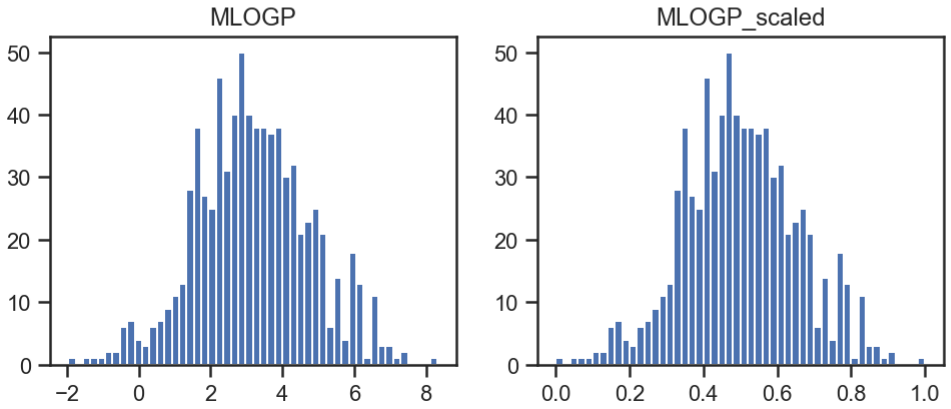
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    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()
```

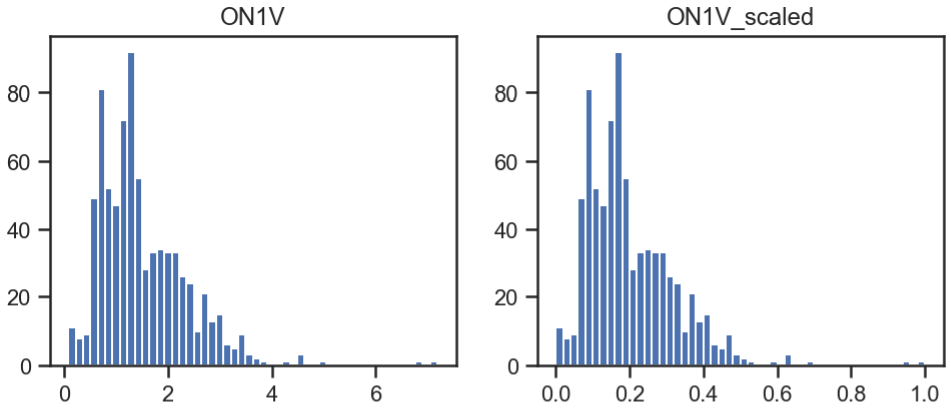


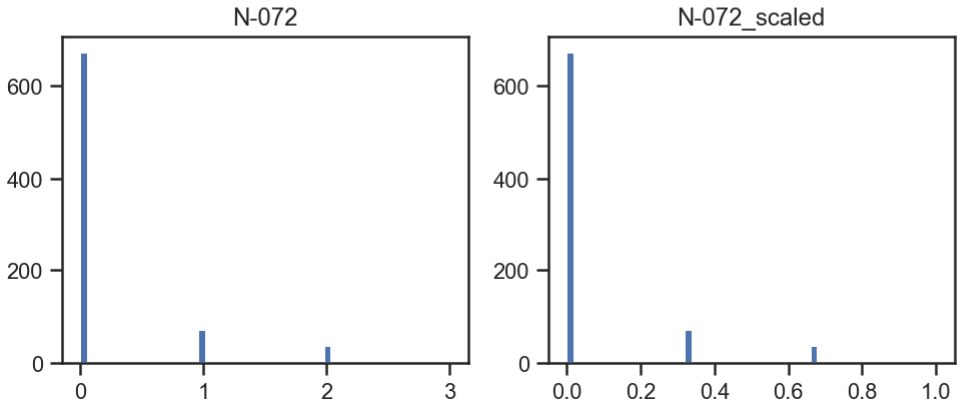



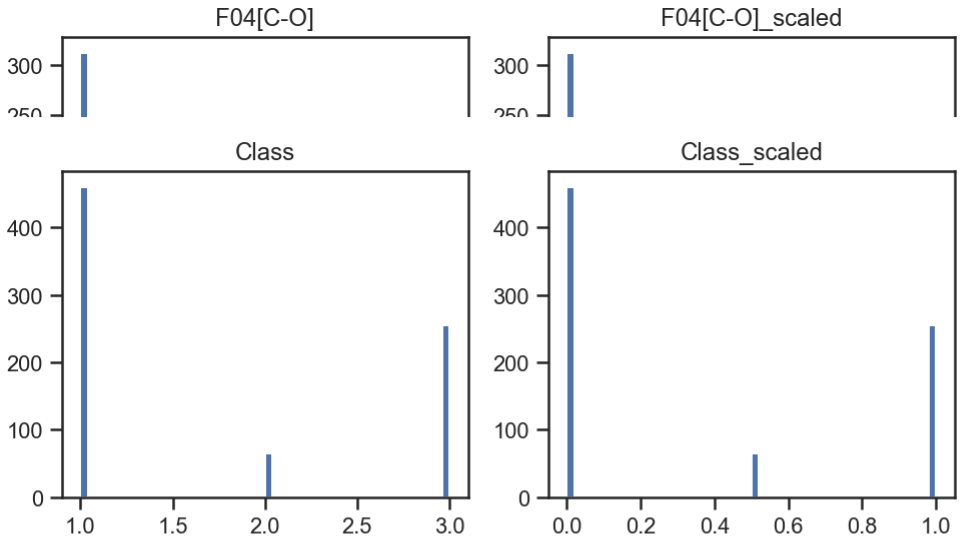



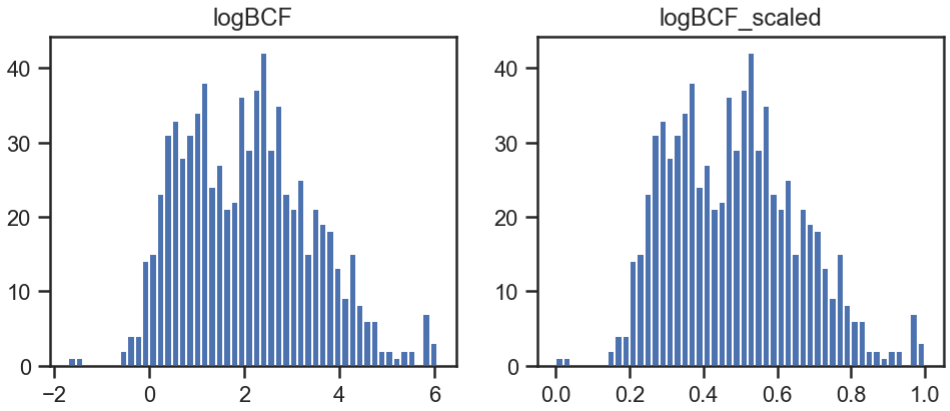

X2Av
X2Av_scaled


MLOGP
MLOGP_scaled


ON1V
ON1V_scaled


N-072
N-072_scaled




Как видим, масштабирование никак не повлияло на распределение данных.

## 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.

In [26]:

```
corr_cols_1 = scale_cols + ['B02[C-N]']
corr_cols_1
```

Out[26]:

```
['nHM',
 'piPC09',
 'PCD',
 'X2Av',
 'MLOGP',
 'ON1V',
 'N-072',
 'F04[C-O]',
 'Class',
 'logBCF',
 'B02[C-N]']
```

In [27]:

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['B02[C-N]']
corr_cols_2
```

Out[27]:

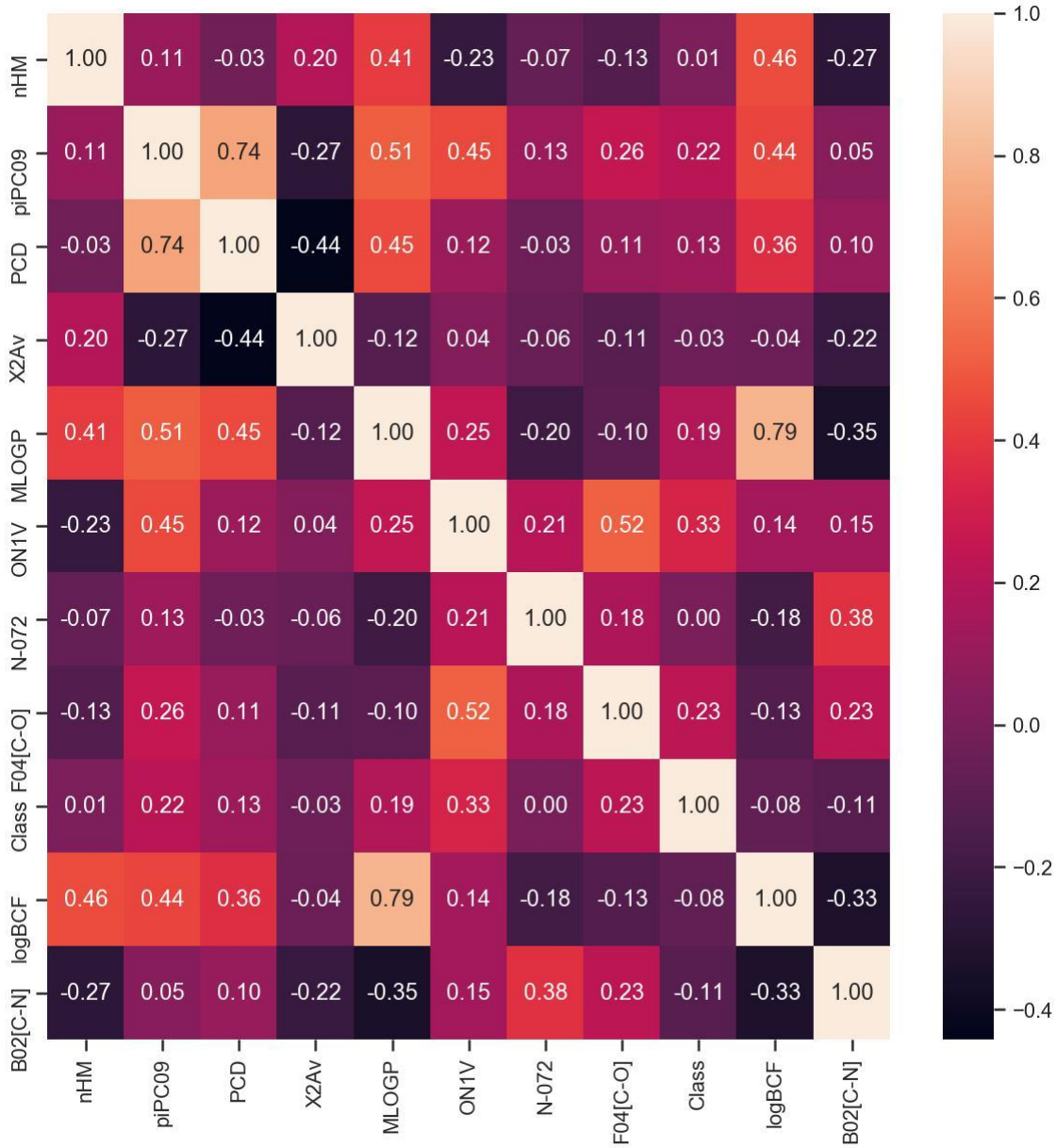
```
['nHM_scaled',
 'piPC09_scaled',
 'PCD_scaled',
 'X2Av_scaled',
 'MLOGP_scaled',
 'ON1V_scaled',
 'N-072_scaled',
 'F04[C-O]_scaled',
 'Class_scaled',
 'logBCF_scaled',
 'B02[C-N]']
```

In [30]:

```
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[30]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20197ef9e48>
```

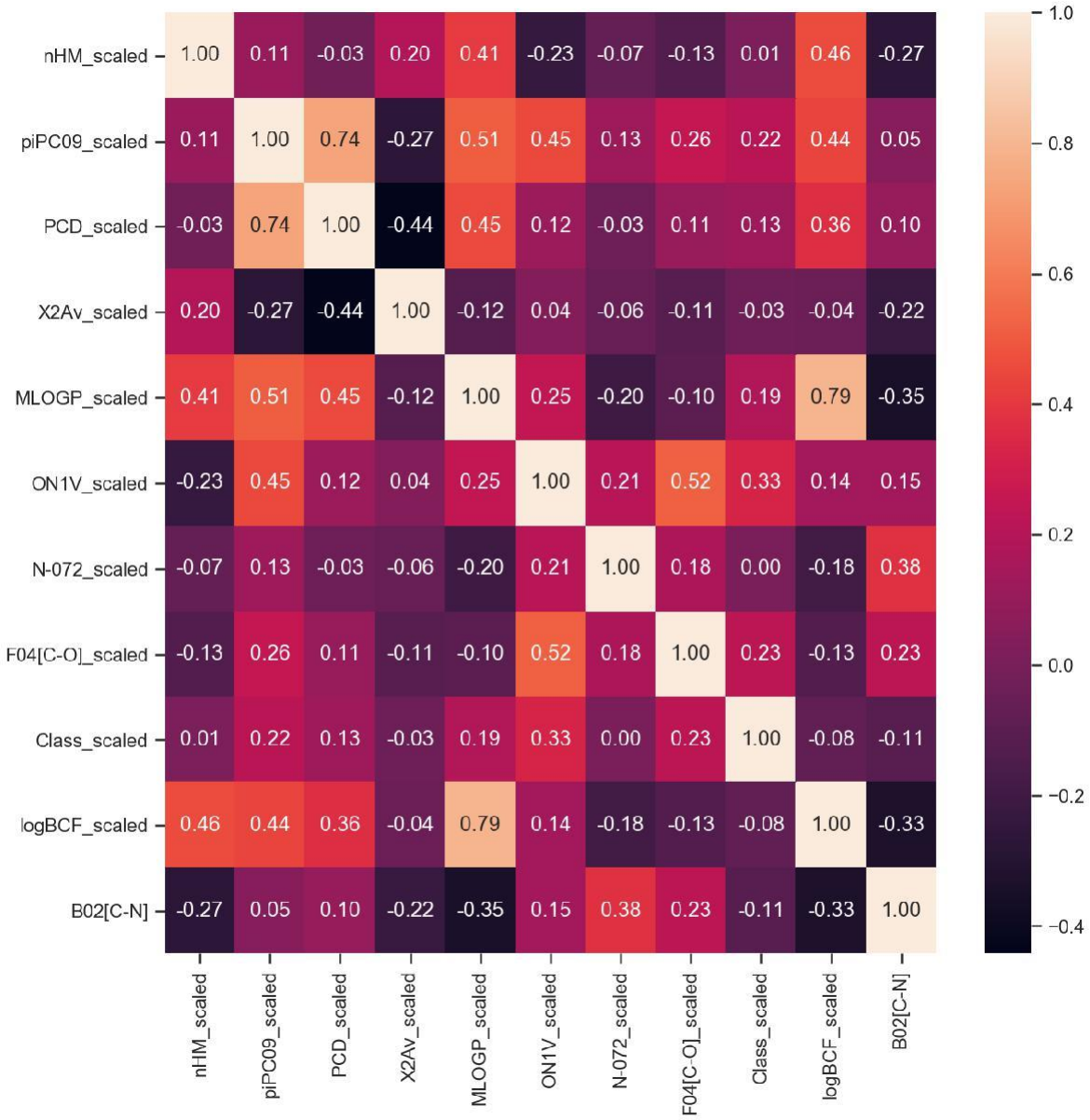


In [31]:

```
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x20198a09648>
```




На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают;
- Видно, что в данном наборе данных небольшие по модулю значения коэффициентов корреляции, это свидетельствуют о незначительной корреляции между исходными признаками и целевым признаком, но некоторая зависимость все равно имеется, поэтому на основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

## 5) Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.

| | TRUE | FALSE |
|---|---|---|
| **POSITIVE** | True-Positive (Rule matched and attack present) | False-Positive (Rule matched and no attack present) |
| **NEGATIVE** | True-Negative (No rule matched and no attack present) | False-Negative (No rule matched and attack present) |

1. метрика accuracy - показывает отношения правильных предсказаний моделью ко всем;
2. метрика precision - это отношение tp / (tp + fp). Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.
3. метрика recall - это отношение tp / (tp + fn). Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.
4. Метрика ROC AUC. Основана на вычислении следующих характеристик:
   - tpr = tp / (tp + fn) - откладывается по оси ординат. Совпадает с recall;
   - fpr = fp / (fp + tn) - 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.

In [32]:

```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, 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
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

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

In [33]:

```
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, inpl
        # 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 или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.

Для задачи классификации будем использовать следующие модели:

1. Метод k ближайших соседей (KNN);
2. Машина опорных векторов (SVM);
3. Дерево решений (Desicion Tree);
4. Случайный лес (Random Forest);
5. Градиентный бустинг (Gradient Boosting).

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

In [39]:

```
train_data = data[data['Set']=='Train']
test_data = data[data['Set']=='Test']
train_data.shape, test_data.shape
```

Out[39]:

```
((584, 24), (195, 24))
```

In [41]:

```
# Признаки для задачи классификации
class_cols = ['nHM_scaled', 'piPC09_scaled', 'PCD_scaled', 'X2Av_scaled', 'MLOGP_scaled',
              ' 'N-072_scaled', 'F04[C-O]_scaled', 'Class_scaled', 'logBCF_scaled']
```

Разделим выборку на обучающую и тестовую

In [43]:

```
X_train = train_data[class_cols]
X_test = test_data[class_cols]
Y_train = train_data['B02[C-N]']
Y_test = test_data['B02[C-N]']
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[43]:

```
((584, 10), (195, 10), (584,), (195,))
```

## 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.

In [44]:

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

In [45]:

```
# Сохранение метрик
metricLogger = MetricLogger()
```

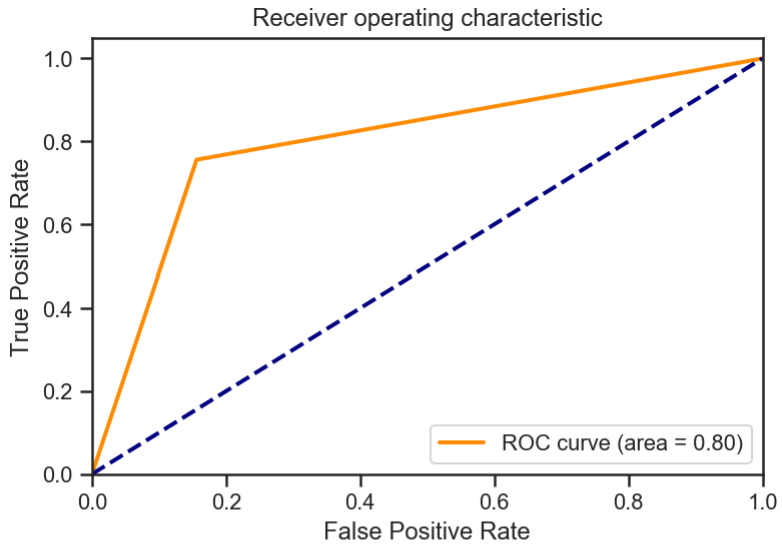
In [48]:

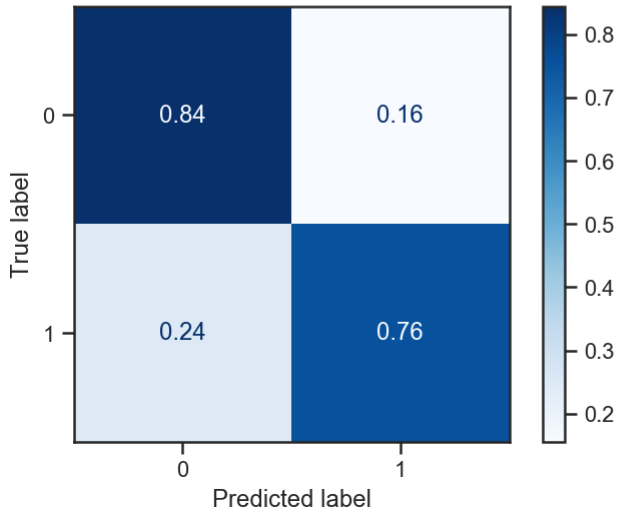
```
def test_model(model_name, model, metricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    accuracy = accuracy_score(Y_test, Y_pred)
    roc_auc = roc_auc_score(Y_test, Y_pred)
    precision = precision_score(Y_test, Y_pred)
    recall = recall_score(Y_test, Y_pred)

    metricLogger.add('precision', model_name, precision)
    metricLogger.add('recall', model_name, recall)
    metricLogger.add('accuracy', model_name, accuracy)
    metricLogger.add('roc_auc', model_name, roc_auc)

    print('**********************************************************')
    print(model)
    print('**********************************************************')
    draw_roc_curve(Y_test, Y_pred)

    plot_confusion_matrix(model, X_test, Y_test,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

In [49]:

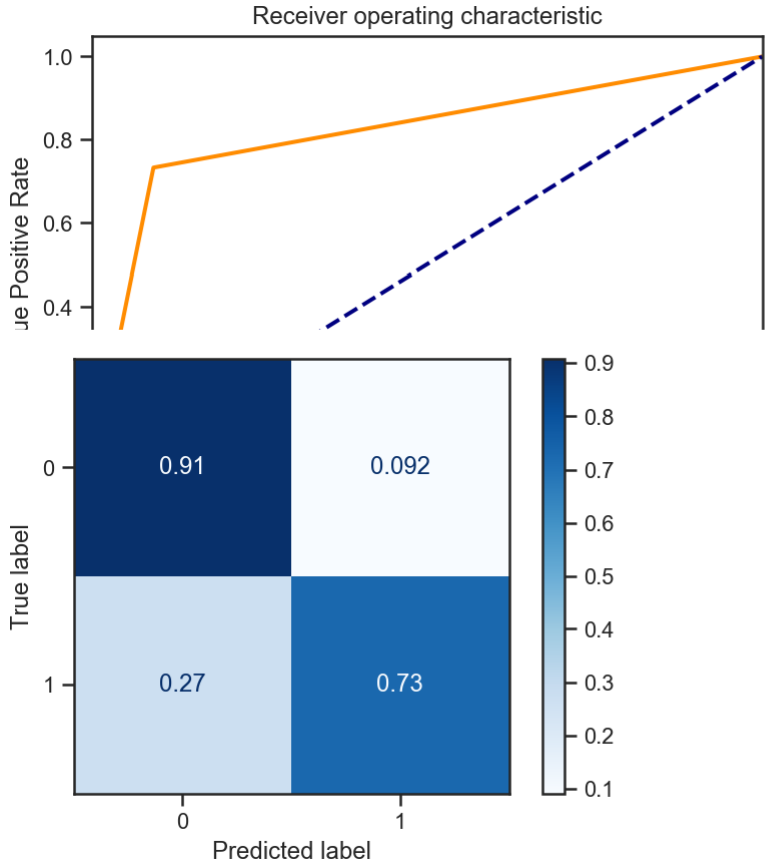
```
for model_name, model in models.items():
    test_model(model_name, model, metricLogger)
```

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3,
                     p=2, weights='uniform')
**************************************************
```

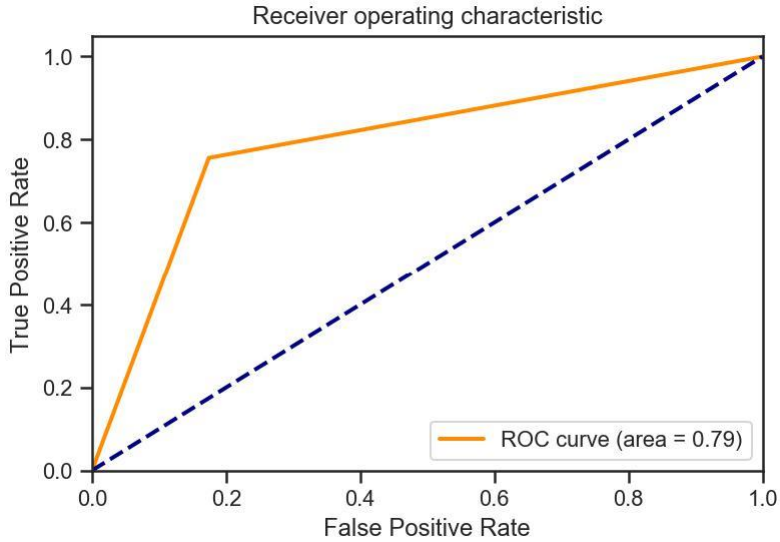


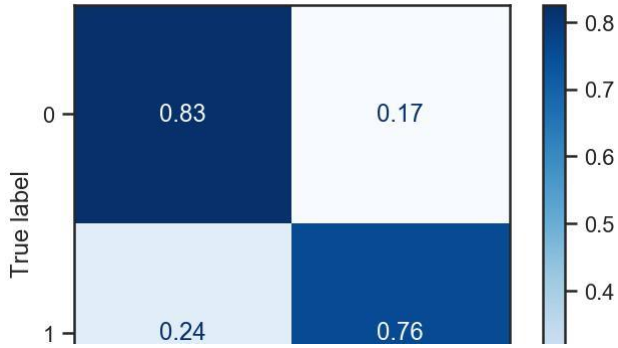




```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
**************************************************
```

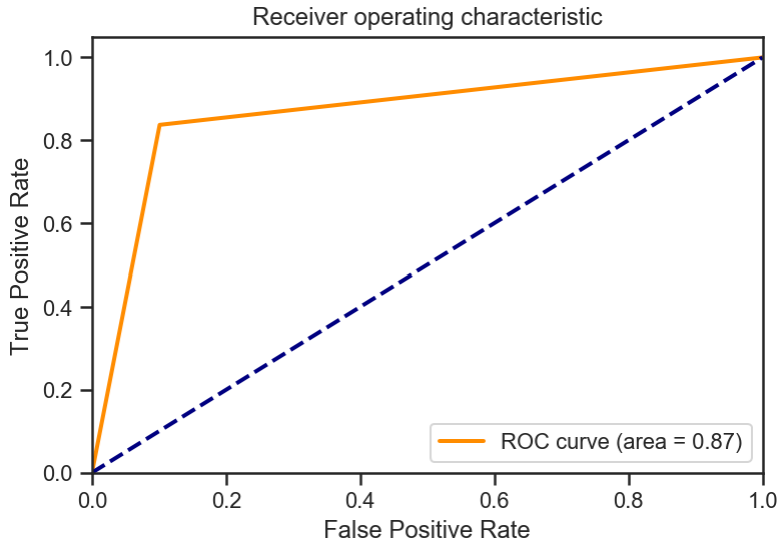


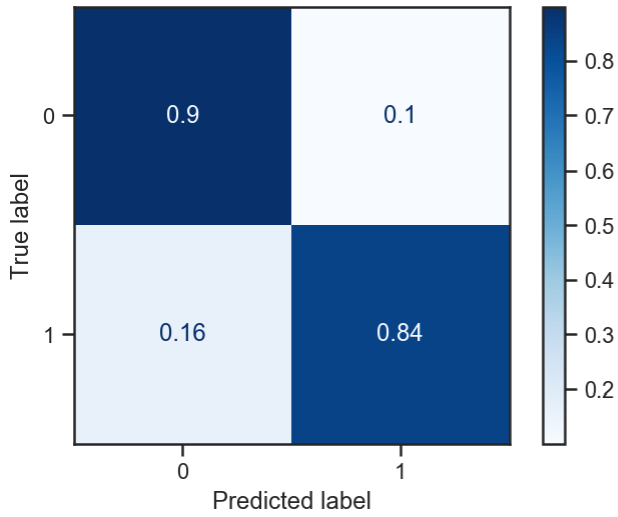

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```





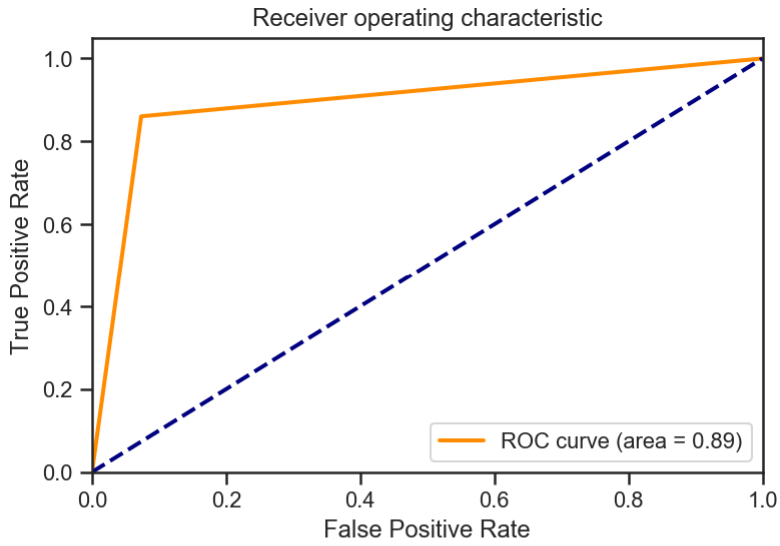

```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='aut
o',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
**************************************************
```

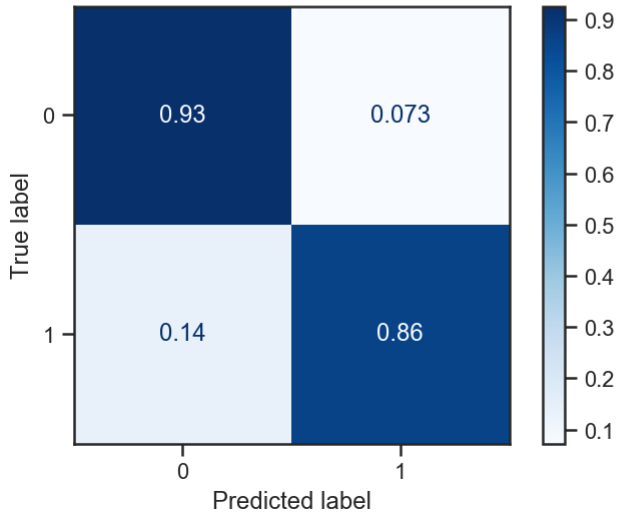



```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse',
init=N one,
                           learning_rate=0.1, loss='deviance', max_depth=
3,
```

```
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=N
one,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```




**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.**

In [103]:

```
n_range = np.array(range(1,410,10))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[103]:

```
[{'n_neighbors': array([ 1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 10 1, 111,
121,
         131, 141, 151, 161, 171, 181, 191, 201, 211, 221, 231, 241, 251,
         261, 271, 281, 291, 301, 311, 321, 331, 341, 351, 361, 371, 381,
         391, 401])}]
```

In [109]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy',
n clf_gs.fit(X_train, Y_train)
```

```
Wall time: 1.44 s
```

Out[109]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=-1,
             param_grid=[{'n_neighbors': array([  1,   11,   21,   31,   41,   5
 1, 61,  71,  81,  91, 101, 111, 121,
        131,141,  151, 161, 171, 181, 191, 201, 211, 221, 231, 241, 251,
        261,271,  281, 291, 301, 311, 321, 331, 341, 351, 361, 371, 381,
        391,401])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

In [110]:

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

Out[110]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1,
                     p=2, weights='uniform')
```

In [111]:

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

Out[111]:

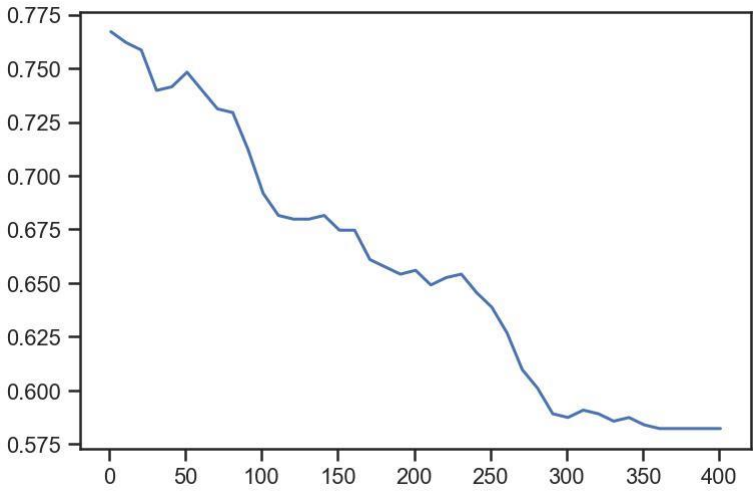
```
{'n_neighbors': 1}
```

In [112]:

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

Out[112]:

```
[<matplotlib.lines.Line2D at 0x20199917208>]
```

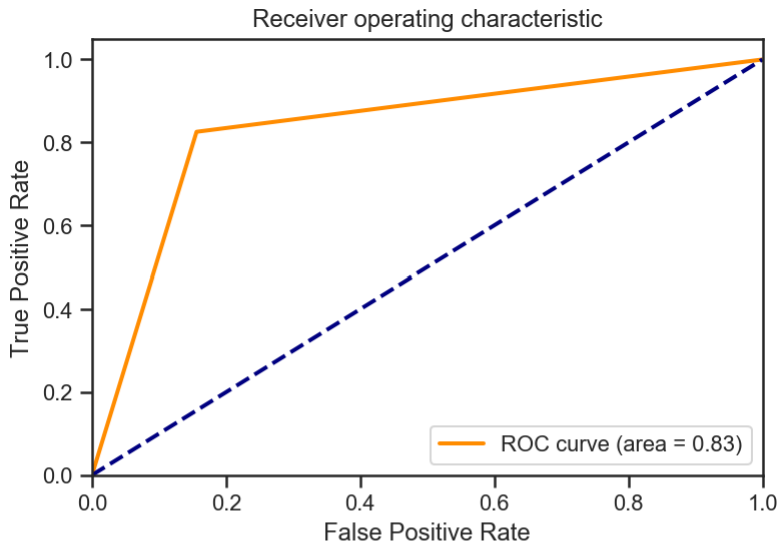


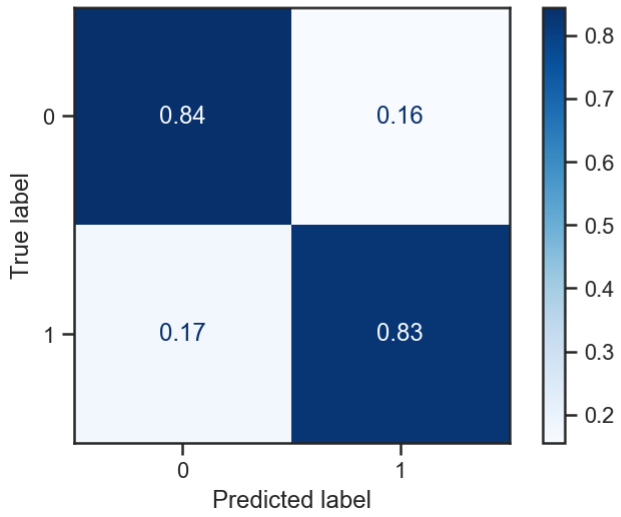

## 10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

In [114]:

```
test_model('KNN_1', KNeighborsClassifier(n_neighbors=1), metricLogger)
```

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1,
                     p=2, weights='uniform')
**************************************************
```




**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.**

In [115]:

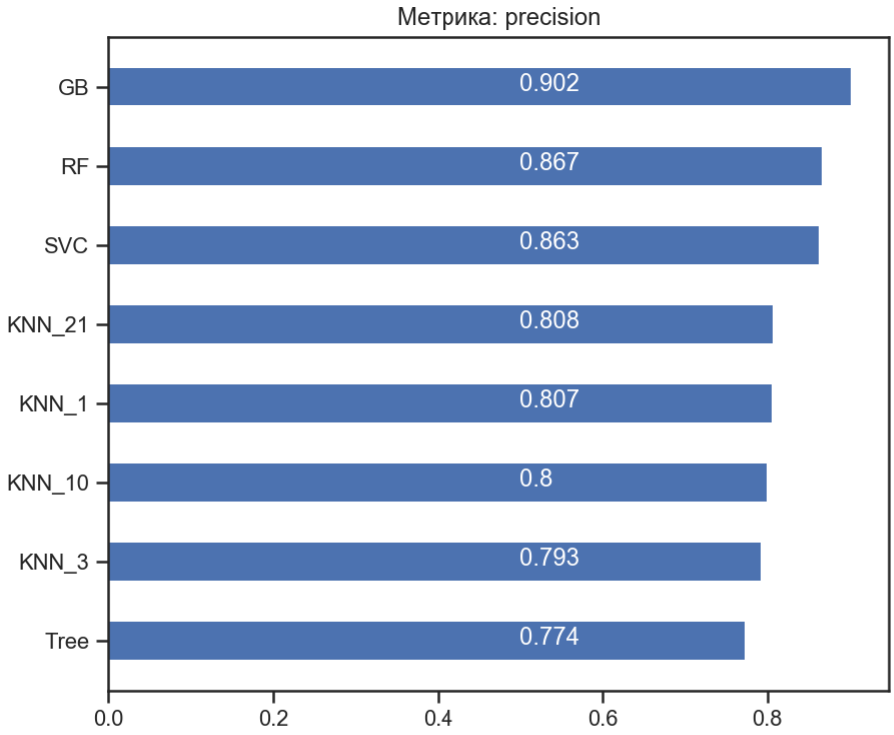
```
# Метрики качества модели
metrics = metricLogger.df['metric'].unique()
metrics
```

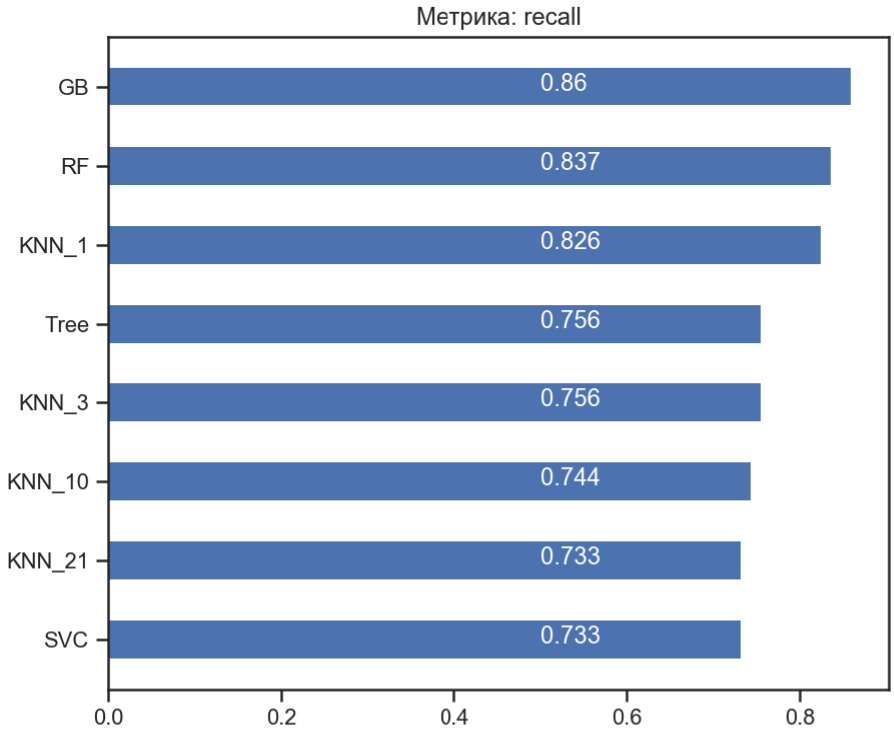
Out[115]:

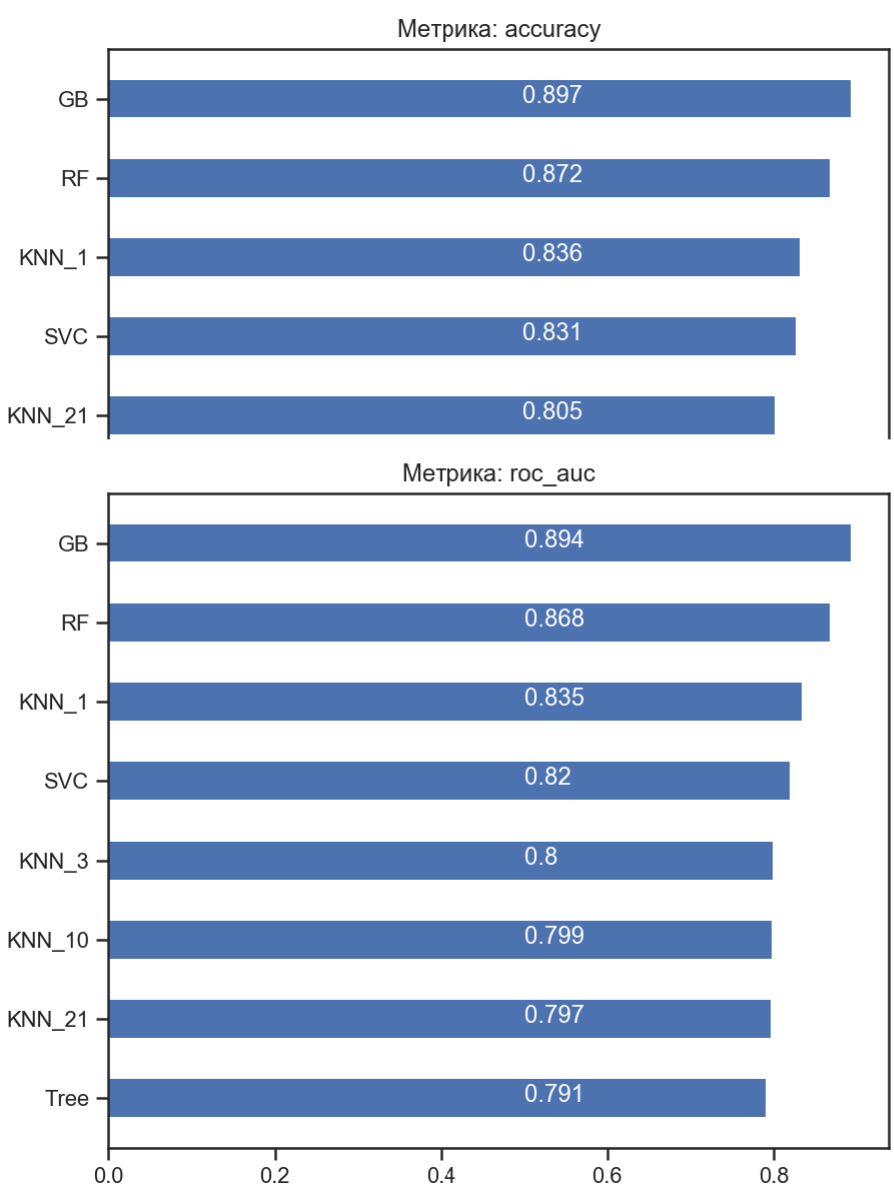
```
array(['precision', 'recall', 'accuracy', 'roc_auc'], dtype=object)
```

In [116]:

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






**Вывод: на основании всех четырёх используемых метрик, лучшей оказалась модель GB - Градиентный бустинг (Gradient Boosting).**

## ЗАКЛЮЧЕНИЕ

В данном курсовом проекте мы выполнили типовую задачу машинного обучения. Провели анализ данных, провели некоторые операции с датасетом, подобрали модели, а также подобрали наиболее подходящие гиперпараметры.

В нашем случае классификатор на основе метода опорных векторов показал лучшие результаты в 75% метрик. В последней метрике немного уступил классификатору на основе случайного леса.

В данном проекте были закреплены все знания, полученные в данном курсе.

Машинное обучение очень актуально в современном мире, оно используется практически во многих сферах. Программист должен подбирать подходящие технологии машинного обучения для достижения наилучших результатов.

**ЛИТЕРАТУРА**


1. Рукописные лекции за 2020 год по дисциплине «Технологии машинного обучения»
2. https://scikit-learn.org/stable/index.html
3. https://www.kaggle.com/datasets
4. http://www.machinelearning.ru/