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_____

_____Коваленко Артём Александрович_____
(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 с расшифровкой голосов мужских и женских- https://www.kaggle.com/primaryobjects/voicegender (https://www.kaggle.com/primaryobjects/voicegender) Датасет: • voice.csv

Колонки:

- meanfreq
- sd
- median
- Q25
- Q75
- IQR.
- skew
- kurt
- sp.ent
- sfm
- mode
- centroid
- meanfun
- minfun
- maxfun
- meandom
- mindom
- maxdom
- dfrange
- modindx
- label: основной бинарный параметр
-
- Будем решать задачу классификации

- Для решения **задачи классификации** в качестве целевого признака будем использовать "label". Поскольку признак содержит только значения 0 и 1, то это задача бинарной классификации.

Импорт библиотек

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

In [1]:

```
import numpy as np import
pandas as pd import seaborn as
sns import matplotlib.pyplot as
plt
from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_rep
ort
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_e
rror, 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.model_selection import train_test_split from gmdhpy import gmdh %matplotlib
inline  sns.set(style="ticks")
```

In [2]:

```
# 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()
```

Загрузка данных

In [3]:

```
data = pd.read_csv('voice.csv')
data.head(7)
```

Out[3]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	s
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.89
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.89
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.84

3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.96											
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.97											
5	0.132786	0.079557	0.119090	0.067958	0.209592	0.141634	1.932562	8.308895	0.96											
6	0.150762	0.074463	0.160106	0.092899	0.205718	0.112819	1.530643	5.987498	0.96											

7 rows × 21 columns

In [4]:

```
data.label=[1 if each =="female" else 0 for each in data.label]
data.label.values
```

Out[4]:

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

In [5]:

```
y=data.label.values
x_data=data.drop(['label'],axis=1)
np.min(x_data)
np.max(x_data)
```

Out[5]:

```
meanfreq       0.251124 sd
0.115273 median
0.261224 Q25
0.247347
Q75            0.273469 IQR
0.252225 skew
34.725453 kurt
1309.612887 sp.ent
0.981997 sfm
0.842936 mode
0.280000 centroid
0.251124 meanfun
0.237636 minfun
0.204082 maxfun
0.279114 meandom
2.957682 mindom
0.458984 maxdom
21.867188 dfrange
21.843750 modindx
0.932374 dtype: float64
```

In [6]:

```
data_colls = ['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx']
```

In [7]:

```
x = data[data_colls]
y = data['label']
```

In [8]:

```
#train test split
x_train, x_test, y_train, y_test=train_test_split(x,y, test_size=0.2, random_state=42)
print("x_train shape:",x_train.shape) print("x_test shape:",x_test.shape)
print("y_train shape:",y_train.shape) print("y_test shape:",y_test.shape)
```

```
x_train shape: (2534, 20)
x_test shape: (634, 20)
y_train shape: (2534,) y_test
shape: (634,)
```

In [9]:

```
# Обучающая выборка
train = x_train
# Тестовая выборка
test = x_test
```

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

Основные характеристики датасетов

In [10]:

```
train.head()
```

Out[10]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	s
807	0.180360	0.053766	0.189474	0.141895	0.213474	0.071579	1.473165	4.899974	0.91
2495	0.185643	0.065043	0.203955	0.186017	0.223454	0.037437	2.732155	11.206323	0.89
2529	0.203908	0.045961	0.201869	0.177944	0.239626	0.061682	2.247859	10.481790	0.89
2241	0.115789	0.081103	0.102197	0.038124	0.193181	0.155057	1.620439	6.600749	0.96
2981	0.179889	0.067810	0.163096	0.137244	0.246925	0.109681	2.106748	8.030296	0.93

In [11]:

```
test.head()
```

Out[11]:

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	s

2148	0.186833	0.027472	0.184325	0.173955	0.204731	0.030777	2.655225	10.565846	0.82
1124	0.188879	0.060316	0.195537	0.138072	0.242975	0.104904	1.497393	5.037085	0.90
170	0.150705	0.087127	0.174299	0.069666	0.226082	0.156416	2.603951	22.328899	0.96
3158	0.183667	0.040607	0.182534	0.156480	0.207646	0.051166	2.054138	7.483019	0.89
2229	0.205159	0.039543	0.210805	0.186667	0.228908	0.042241	2.099683	7.562209	0.87

In [12]:

```
# Размер обучающего датасета - 8143 строк, 7 колонок
train.shape, test.shape
```

Out[12]:

```
((2534, 20), (634, 20))
```

In [13]:

```
# Список колонок
train.columns
```

Out[13]:

```
Index(['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx'],
 dtype='object')
```

In [14]:

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

Out[14]:

```
meanfreq    float64 sd
float64 median
float64 Q25
float64
Q75         float64 IQR
float64 skew
float64 kurt
float64 sp.ent
float64 sfm
float64 mode
float64 centroid
float64 meanfun
float64 minfun
float64 maxfun
float64 meandom
float64 mindom
float64 maxdom
float64 dfrange
float64 modindx
float64 dtype: object
```

In [15]:

```
test.dtypes
```

Out[15]:

```
meanfreq    float64
sd          float64
median      float64
Q25         float64
Q75         float64
IQR         float64
skew        float64
kurt        float64
sp.ent      float64
sfm         float64
mode        float64
centroid    float64
meanfun     float64
minfun      float64
maxfun      float64
meandom     float64
mindom      float64
maxdom      float64
dfrange     float64
modindx     float64
dtype: object In
[16]:
```

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

Out[16]:

```
meanfreq    0 sd
0 median      0
Q25         0
Q75         0 IQR
0 skew        0
kurt        0
sp.ent      0 sfm
0 mode        0
centroid    0
meanfun     0
minfun      0
maxfun      0
meandom     0
mindom      0
maxdom      0
dfrange     0
modindx     0
dtype: int64
```

In [17]:

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

Out[17]:

```
meanfreq    0 sd
0 median      0
Q25         0
Q75         0 IQR
0 skew        0
kurt        0
sp.ent      0 sfm
0 mode        0
centroid    0
meanfun     0
minfun      0
maxfun      0
meandom     0
mindom      0
maxdom      0
dfrange     0
modindx     0
dtype: int64
```

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

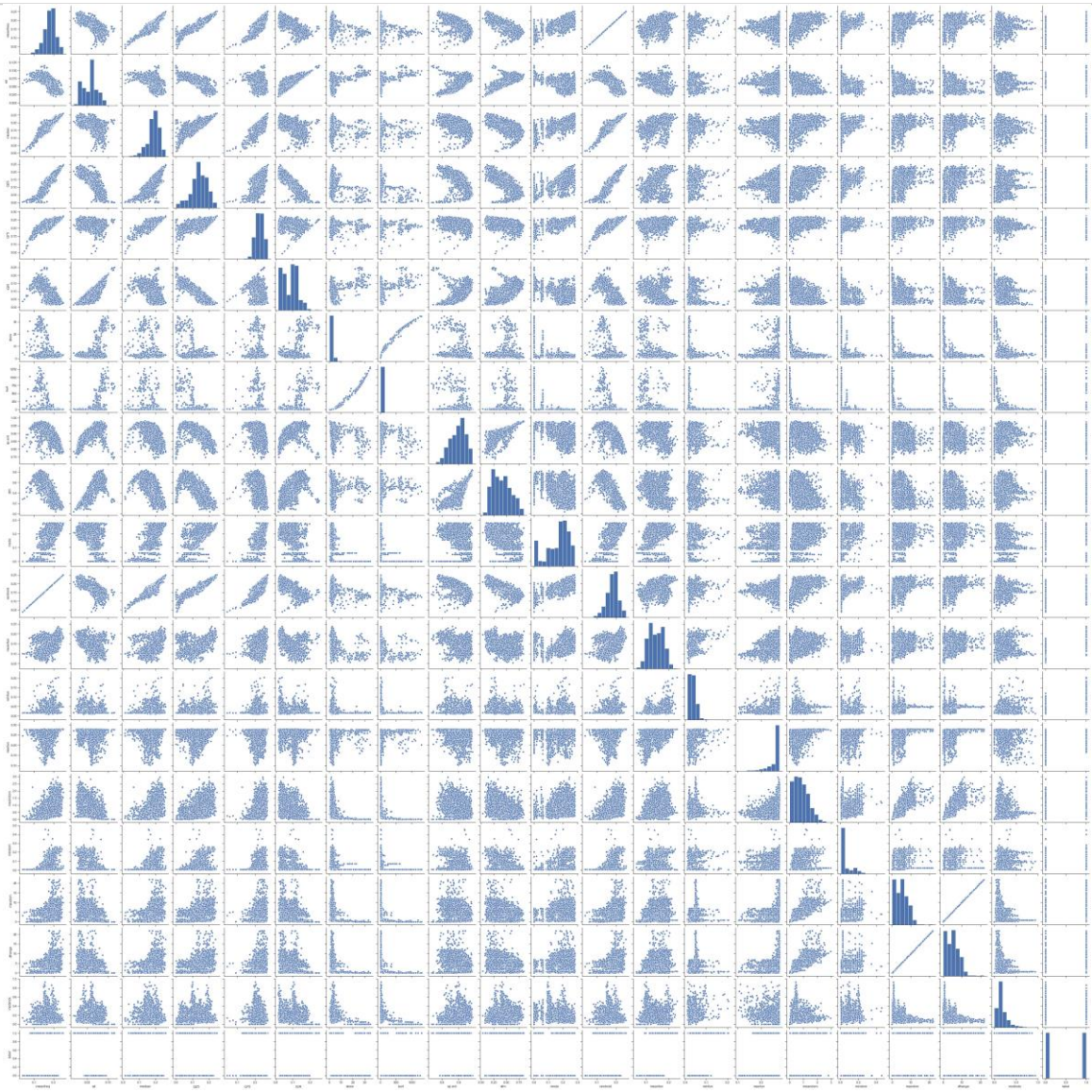
Построим некоторые графики для понимания структуры данных.

In [18]:

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

Out[18]:

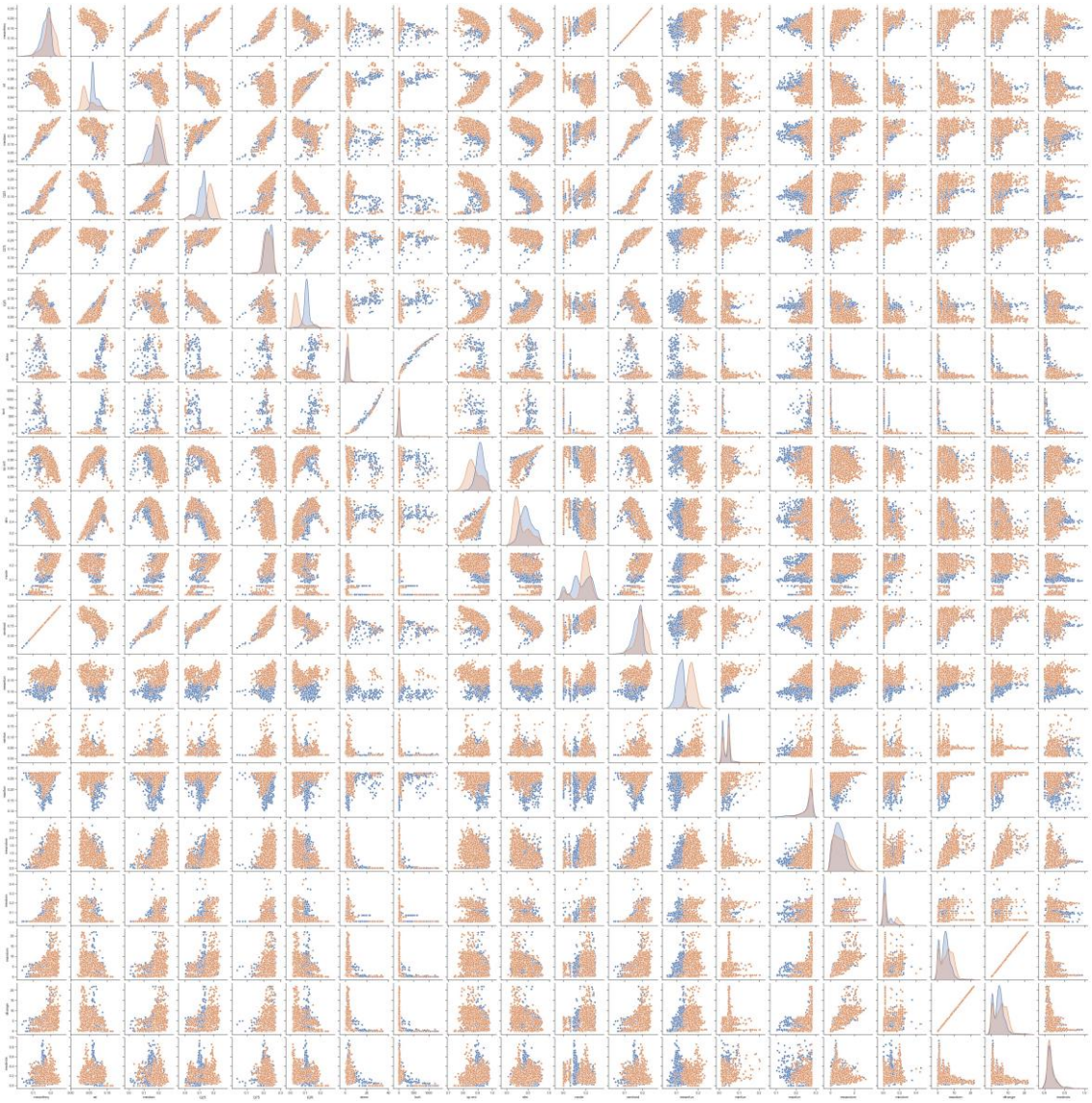
```
<seaborn.axisgrid.PairGrid at 0x27b498efac0>
```



In [19]: sns.pairplot(data,

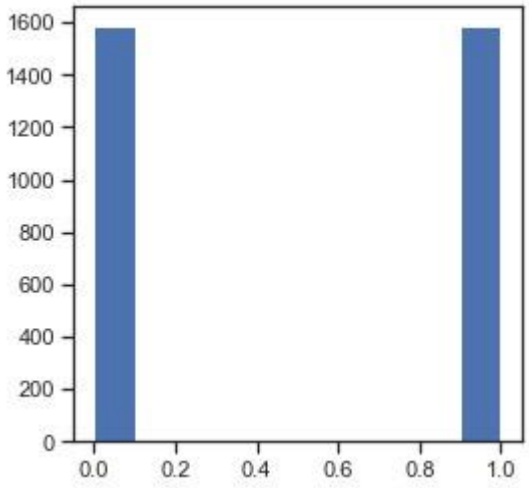
hue="label") Out[19]:

<seaborn.axisgrid.PairGrid at 0x27b5a5239d0>



In [20]:

```
# Оценим дисбаланс классов для Оссирапсу
fig, ax = plt.subplots(figsize=(4,4))
plt.hist(data['label'])
plt.show()
```



In [21]:

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

Out[21]:

```
1    1584
0    1584
Name: label, dtype: int64
```

In [22]:

```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['label'].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 50.0%, а класс 1 составляет 50.0%.
```

Вывод. Дисбаланс классов отсутствует.

In [23]:

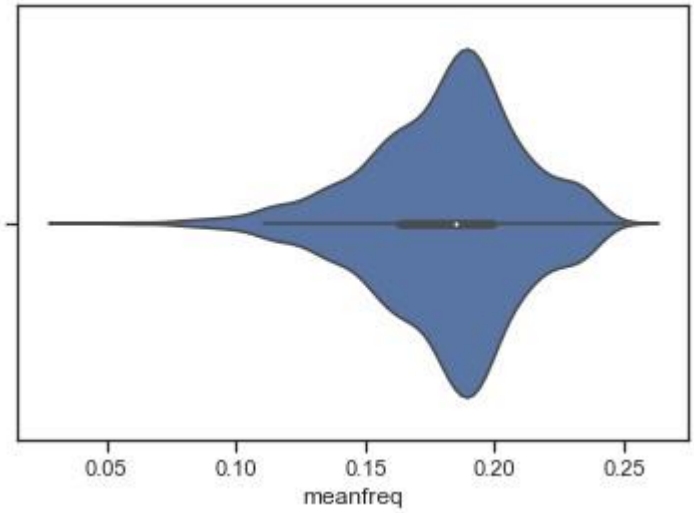
```
data.columns
```

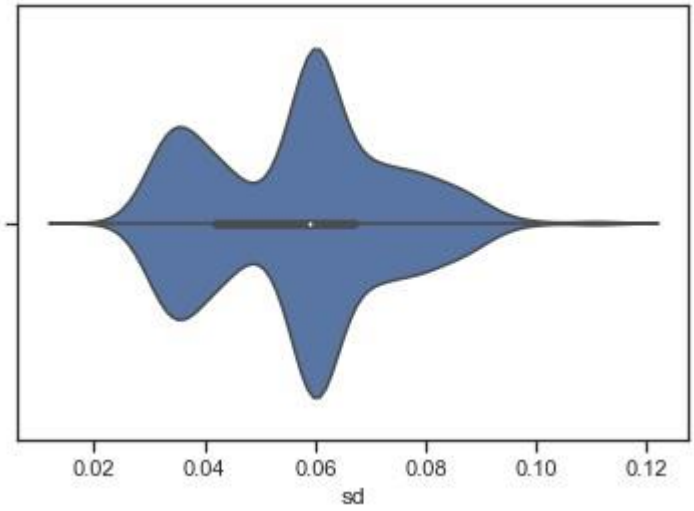
Out[23]:

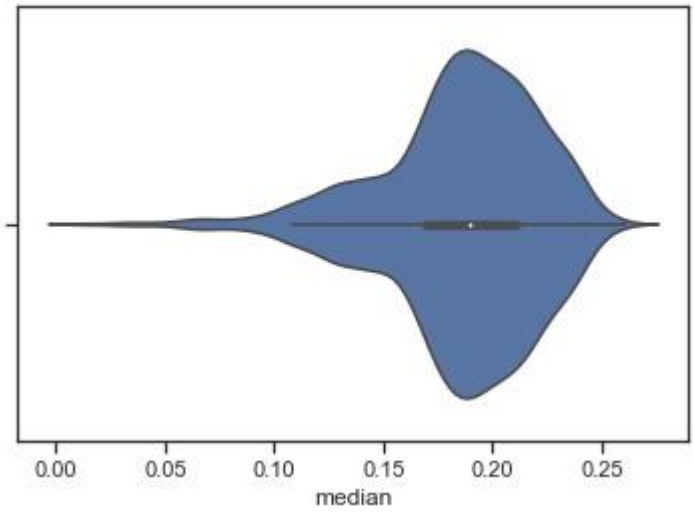
```
Index(['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx', 'label'],
 dtype='object')
```

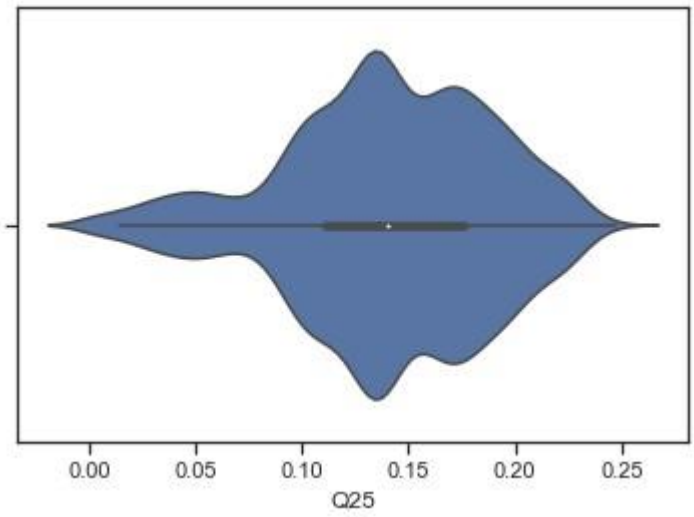
In [24]:

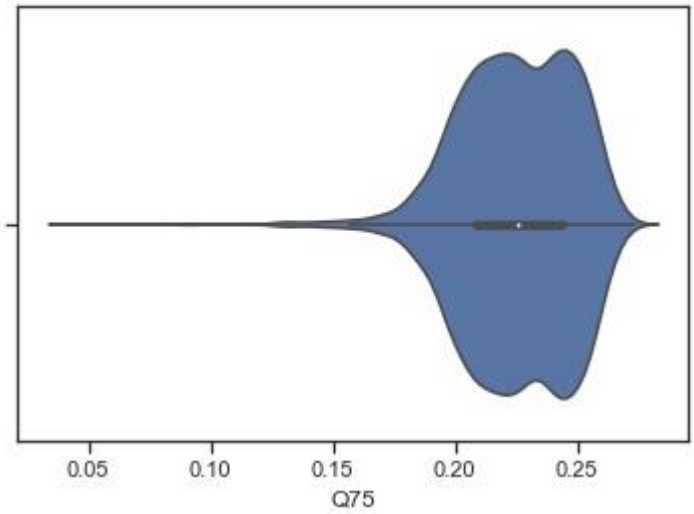
```
# Скрипичные диаграммы для числовых колонок
for col in ['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
 'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx']:
 sns.violinplot(x=data[col])    plt.show()
```

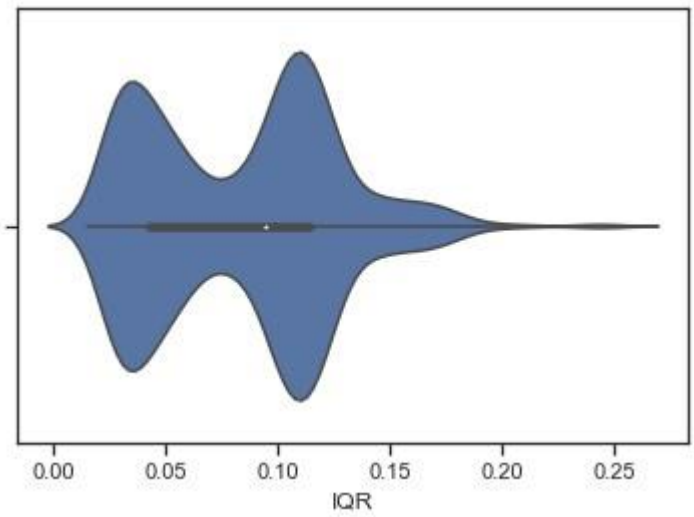

0.05
0.10
0.15
0.20
0.25
meanfreq

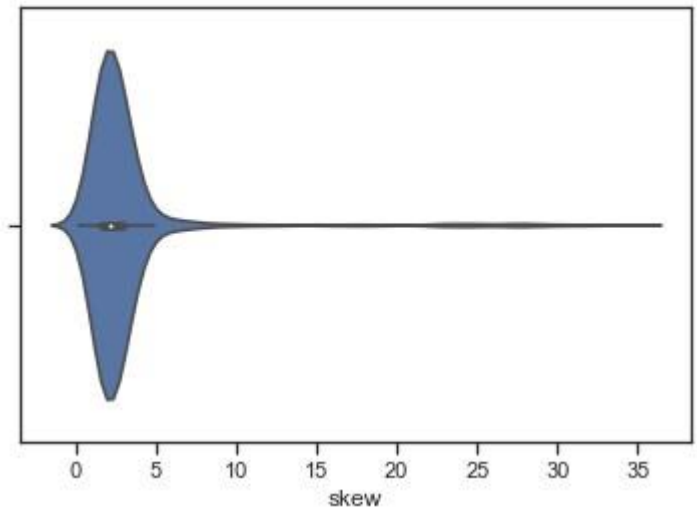

0.02
0.04
0.06
0.08
0.10
0.12
sd

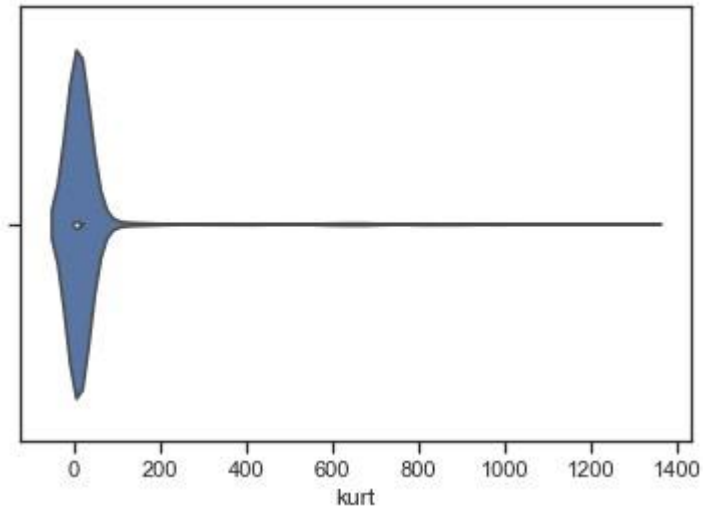

0.00
0.05
0.10
0.15
0.20
0.25
median

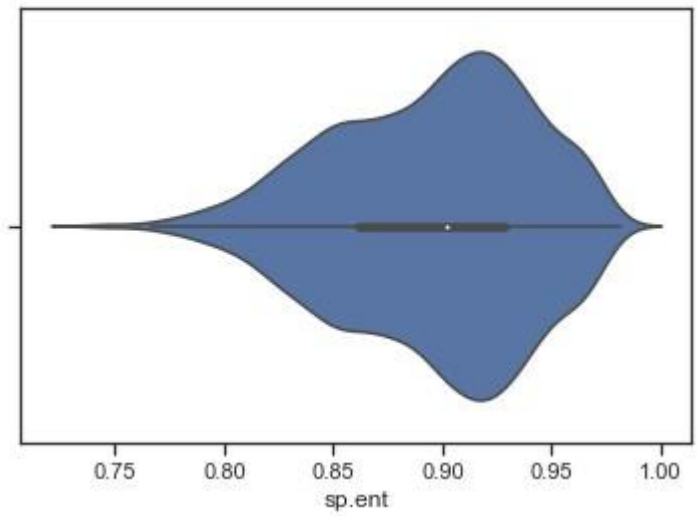

0.00
0.05
0.10
0.15
0.20
0.25
Q25

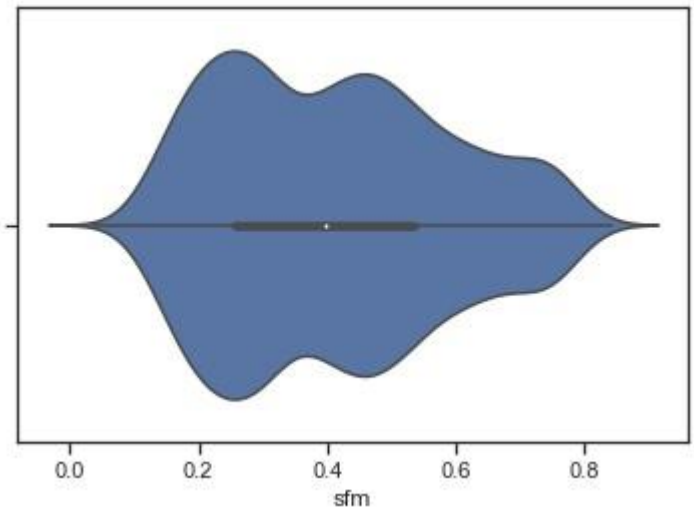

0.05
0.10
0.15
0.20
0.25
Q75

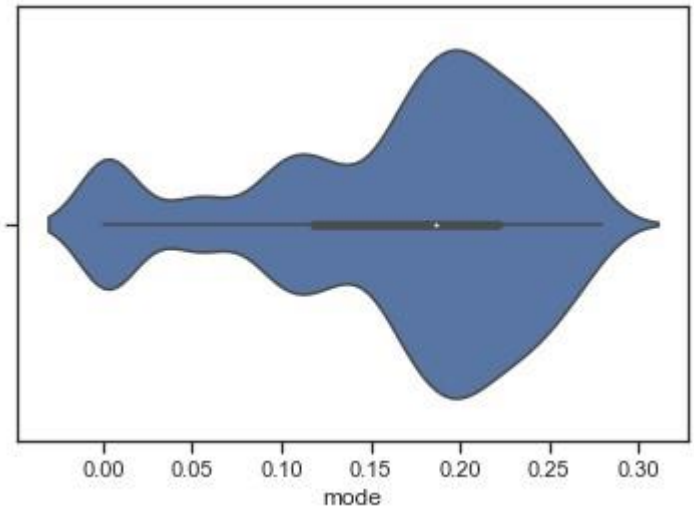

0.00
0.05
0.10
0.15
0.20
0.25
IQR

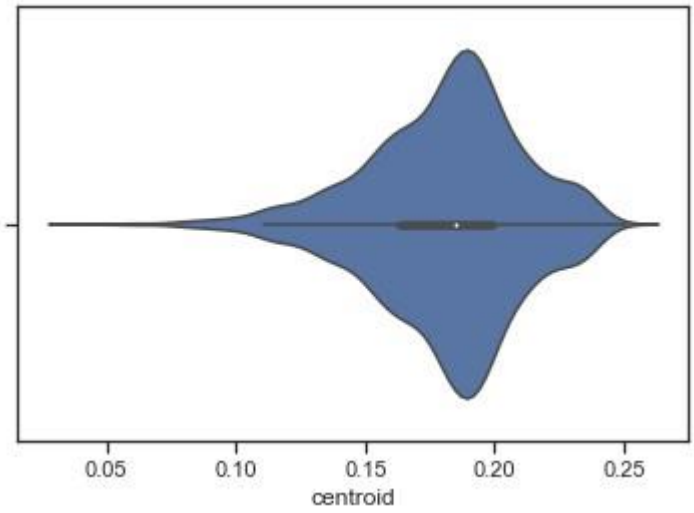

0
5
10
15
20
25
30
35
skew

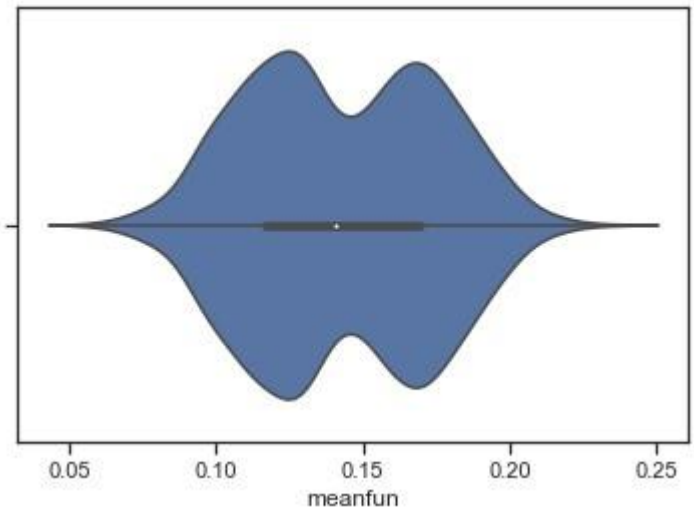

0
200
400
600
800
1000
1200
1400
kurt

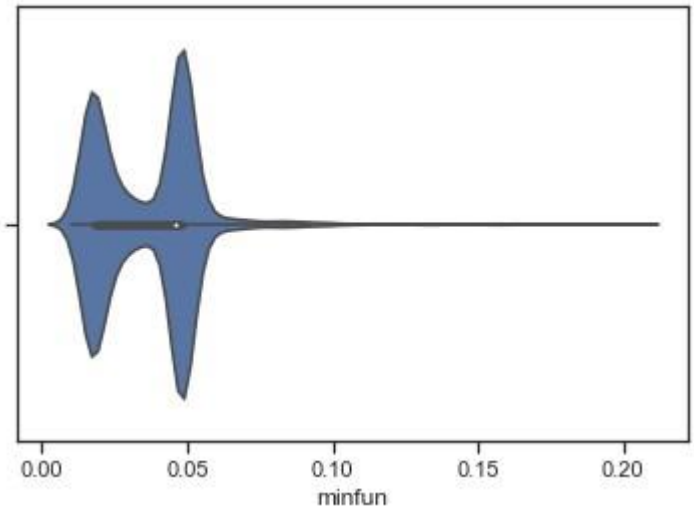

0.75
0.80
0.85
0.90
0.95
1.00
sp.ent

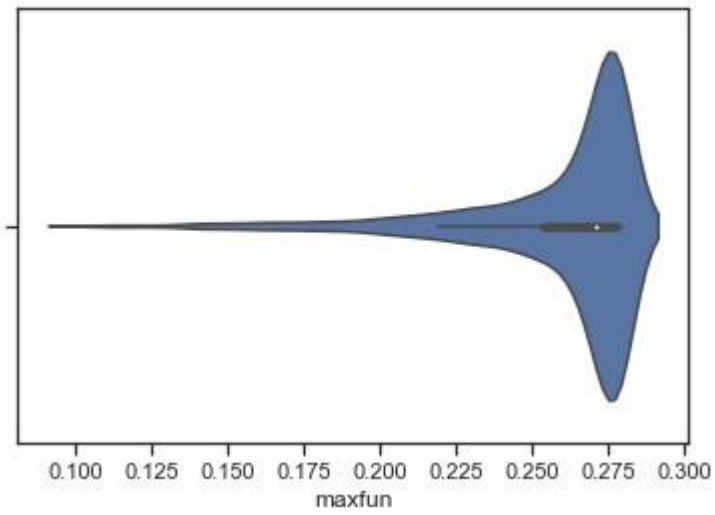

0.0
0.2
0.4
0.6
0.8
sfm

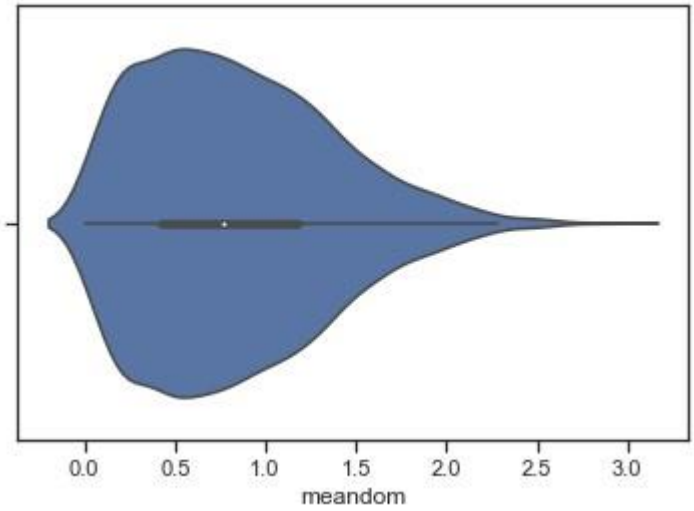

0.00
0.05
0.10
0.15
0.20
0.25
0.30
mode

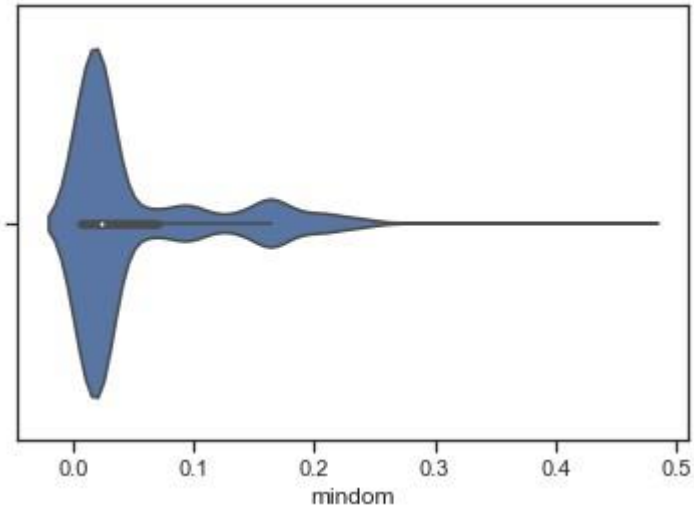

0.05
0.10
0.15
0.20
0.25
centroid

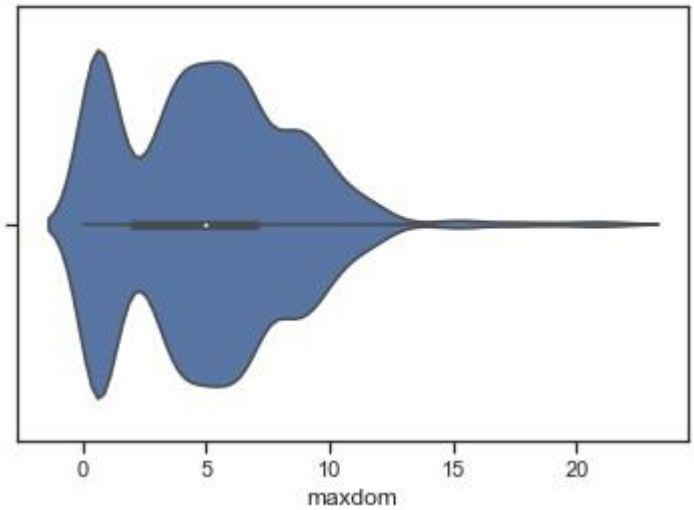

0.05
0.10
0.15
0.20
0.25
meanfun

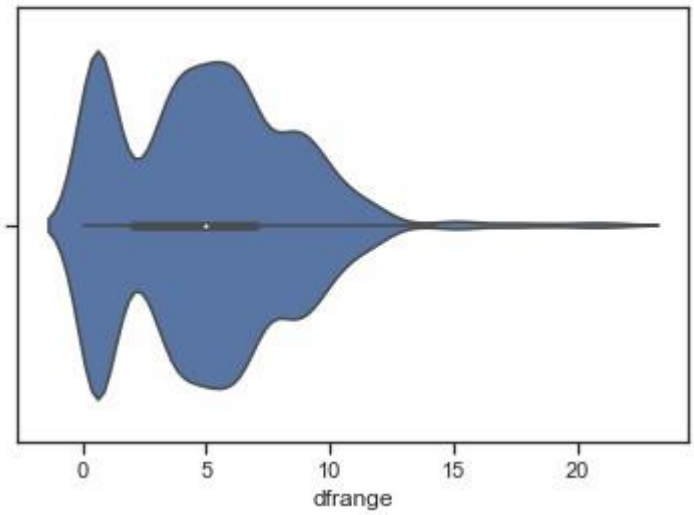

0.00
0.05
0.10
0.15
0.20
minfun

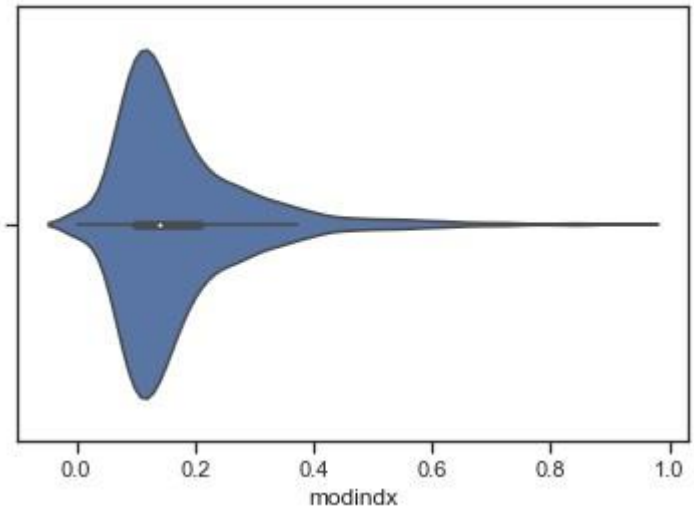

0.100
0.125
0.150
0.175
0.200
0.225
0.250
0.275
0.300
maxfun


0.0
0.5
1.0
1.5
2.0
2.5
3.0
meandom


0.0
0.1
0.2
0.3
0.4
0.5
mindom


0
5
10
15
20
maxdom





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

In [25]:

```
train.dtypes
```

Out[25]:

```
meanfreq    float64 sd
float64 median
float64 Q25
float64
Q75         float64 IQR
float64 skew
float64 kurt
float64 sp.ent
float64 sfm
float64 mode
float64 centroid
float64 meanfun
float64 minfun
```

```
float64 maxfun
float64 meandom
float64 mindom
float64 maxdom
float64 dfrange
float64 modindx
float64 dtype: object
```

Для построения моделей будем использовать все признаки

Категориальные признаки отсутствуют, их кодирования не требуется

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

In [26]:

```
# Числовые колонки для масштабирования
scale_cols = ['meanfreq', 'sd', 'median', 'Q25', 'Q75', 'IQR', 'skew', 'kurt',
       'sp.ent', 'sfm', 'mode', 'centroid', 'meanfun', 'minfun', 'maxfun',
       'meandom', 'mindom', 'maxdom', 'dfrange', 'modindx']
```

In [27]:

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

In [28]:

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

In [29]:

```
data.head()
```

Out[29]:

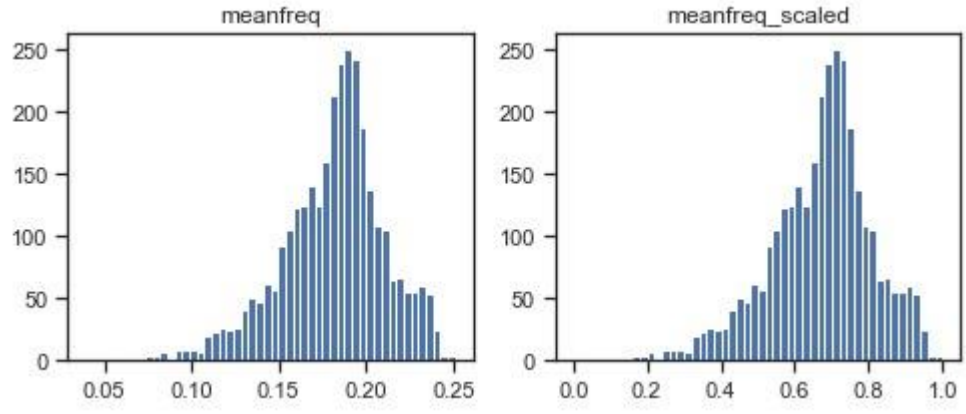
	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	s
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.89
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.89
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.84
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.96

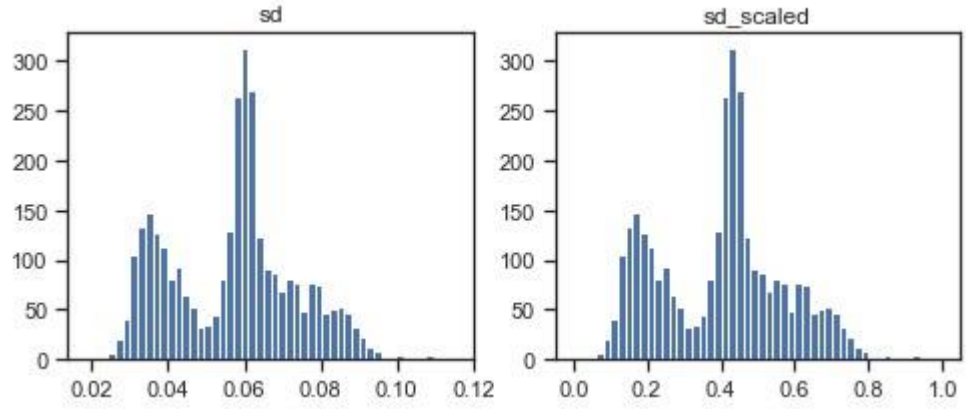
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.97

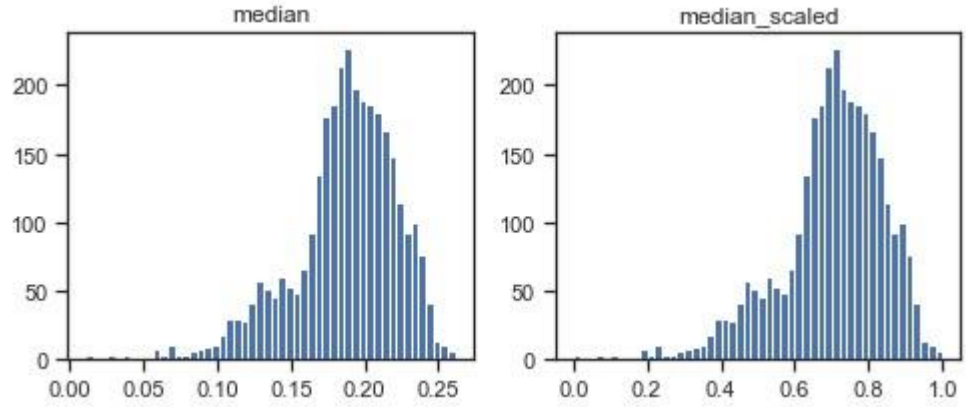
5 rows × 41 columns

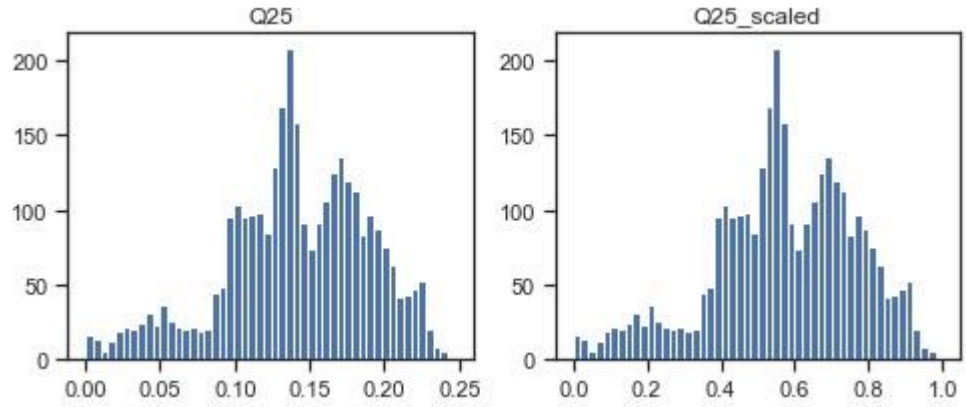
In [30]:

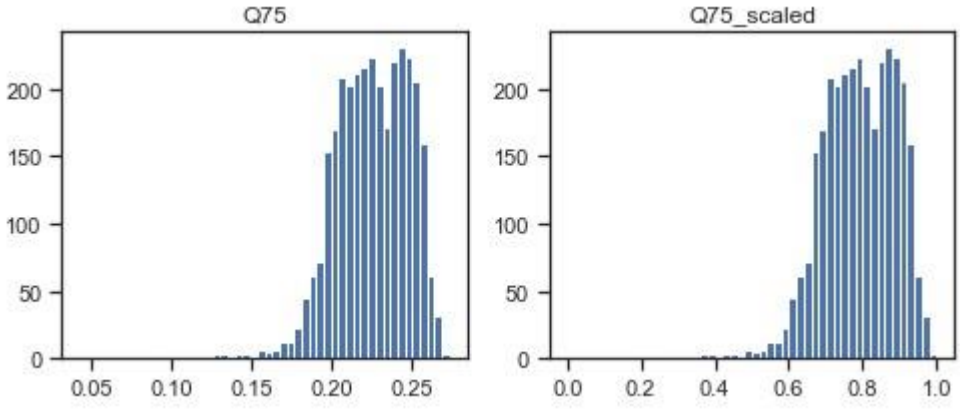
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    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()
```

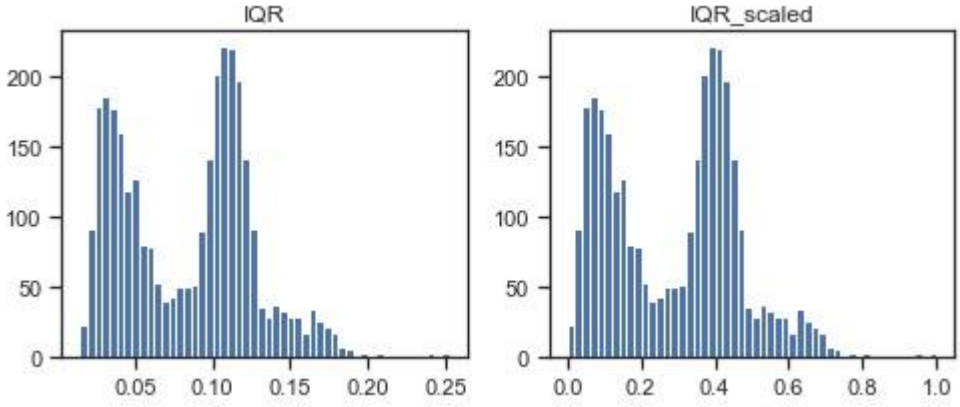

meanfreq
meanfreq_scaled

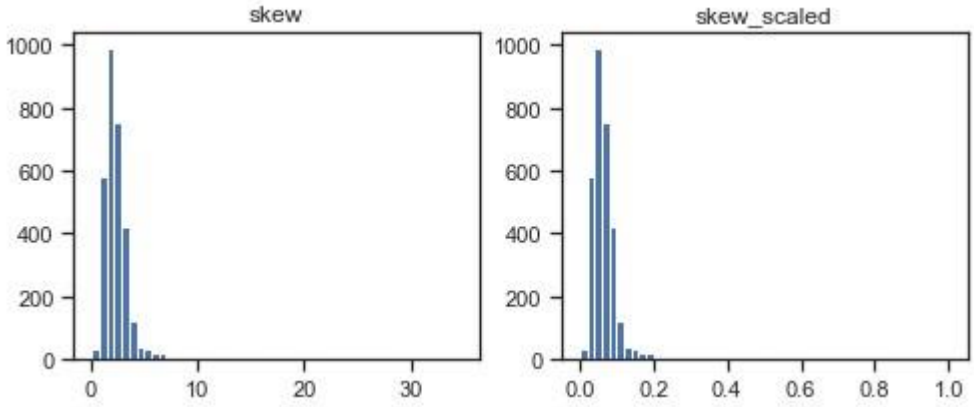

sd
sd_scaled

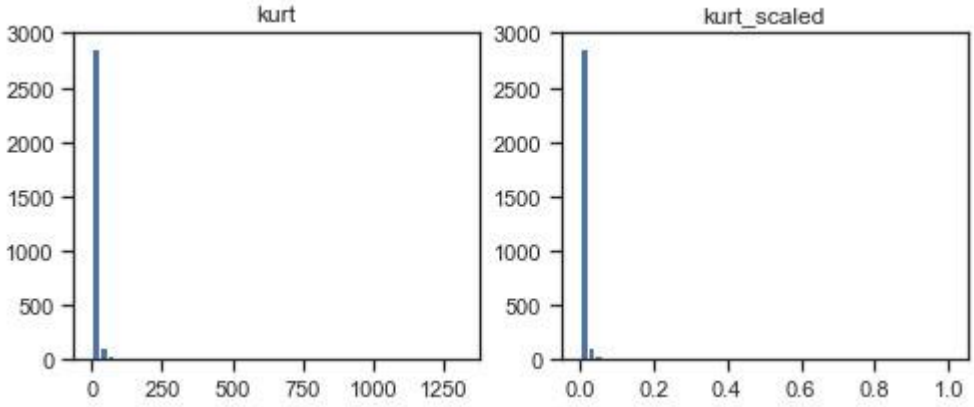

median
median_scaled

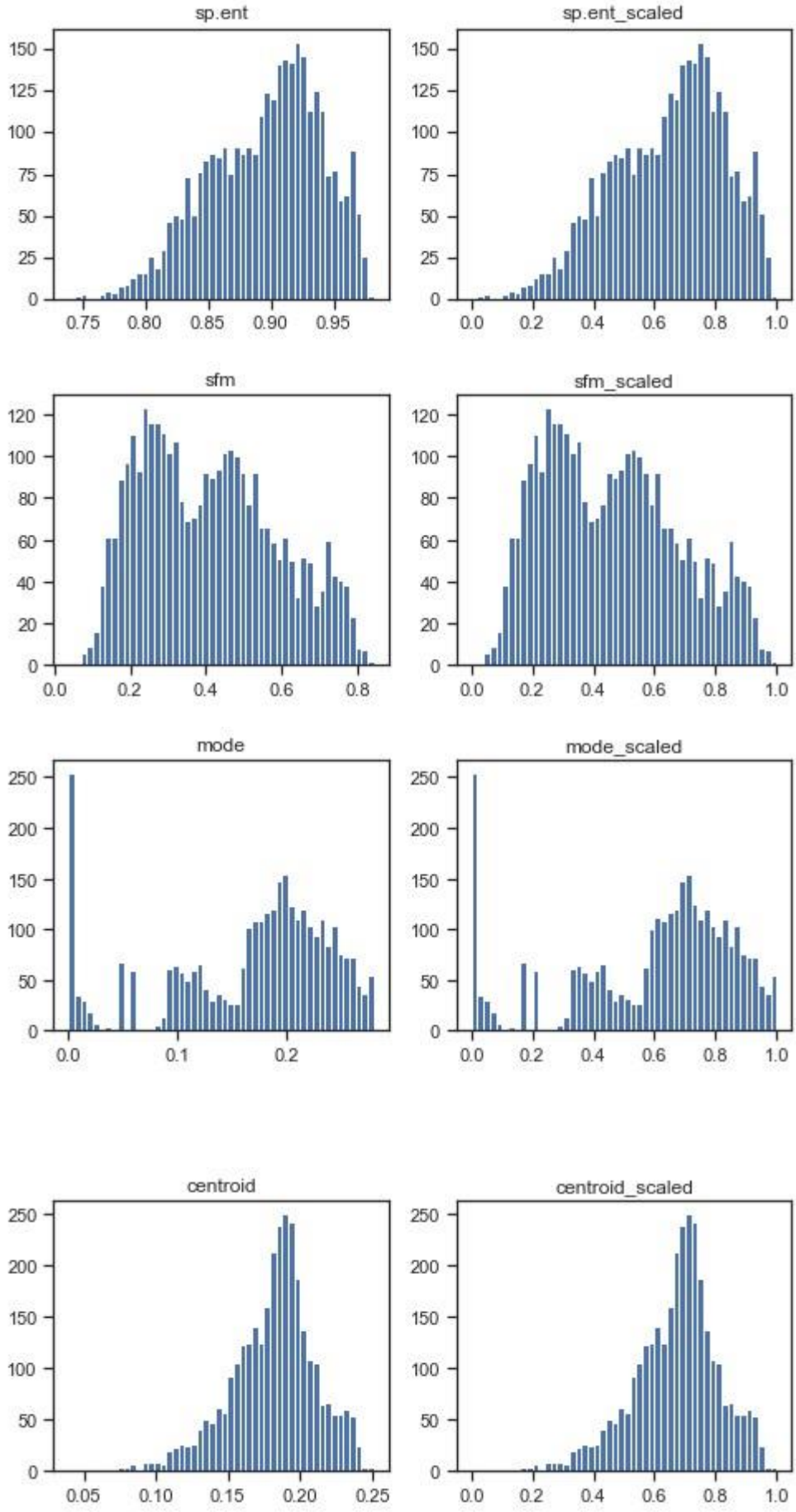

Q25
Q25_scaled

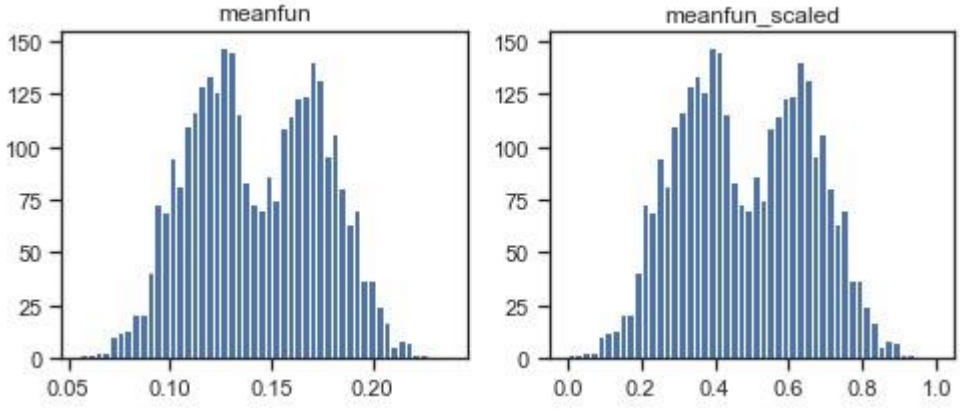

Q75
Q75_scaled

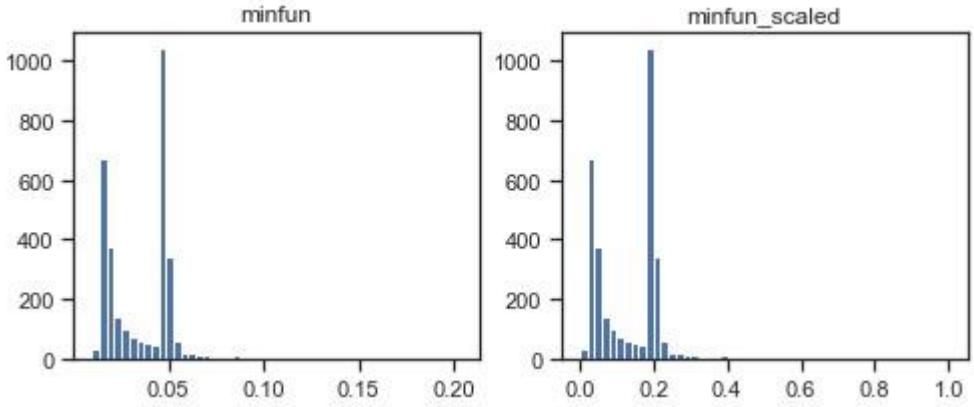

IQR
IQR_scaled

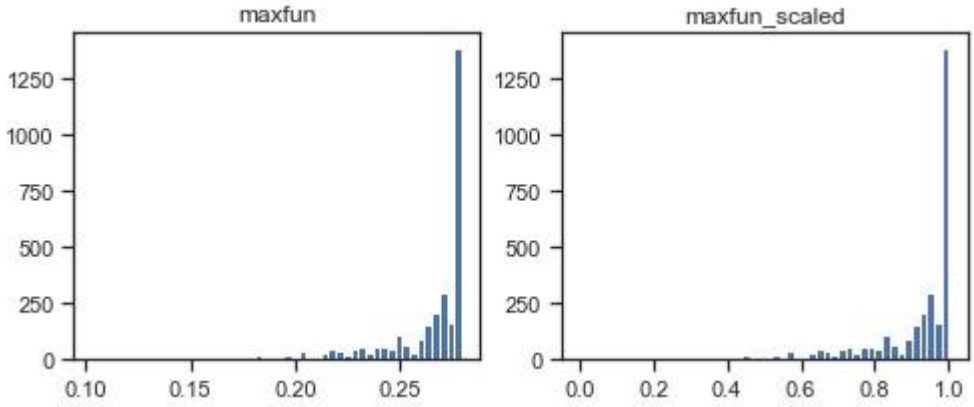

skew
skew_scaled

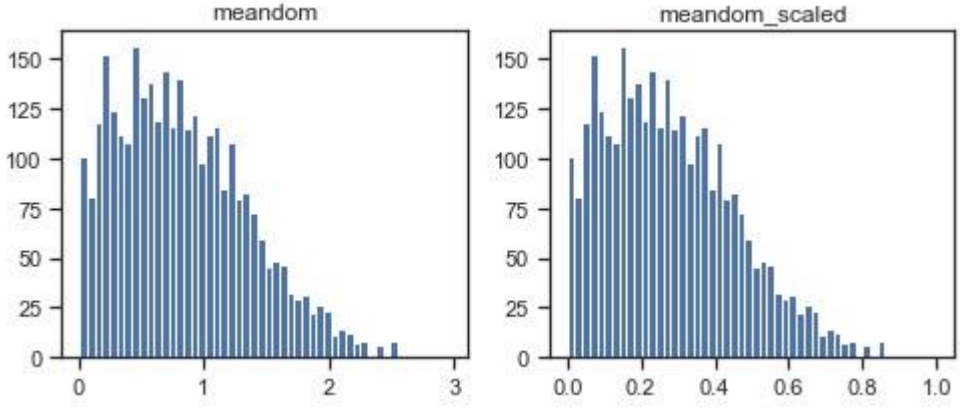

kurt
kurt_scaled

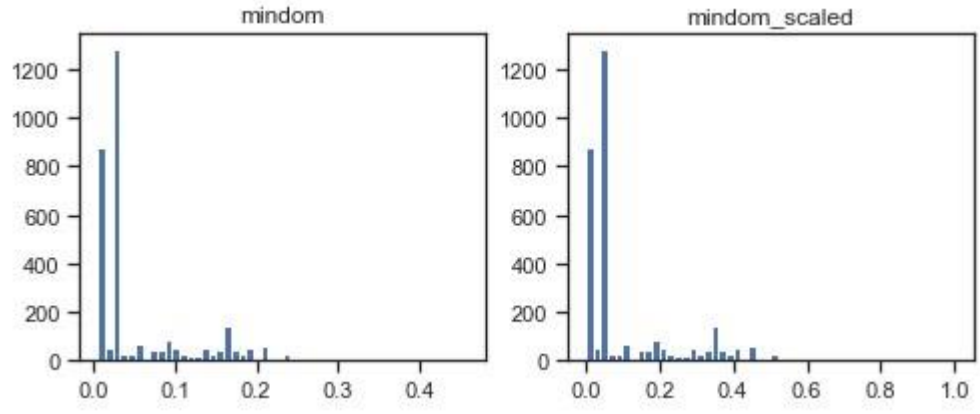

sp.ent
sp.ent_scaled
sfm
sfm_scaled
mode
mode_scaled
centroid
centroid_scaled


meanfun
meanfun_scaled

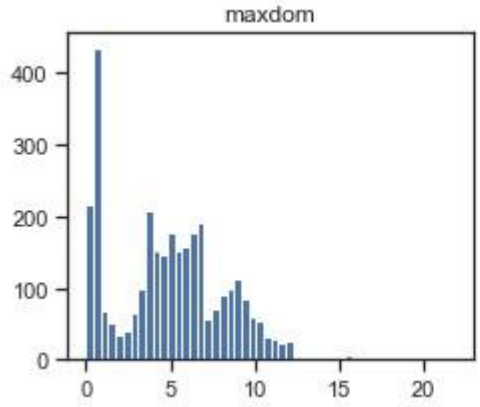

minfun
minfun_scaled

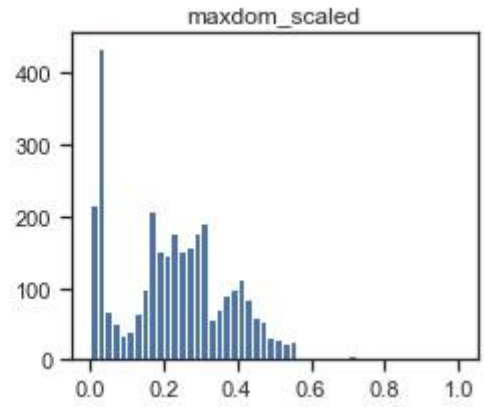

maxfun
maxfun_scaled

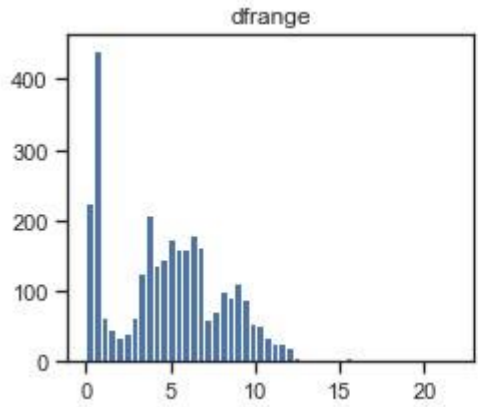

meandom
meandom_scaled

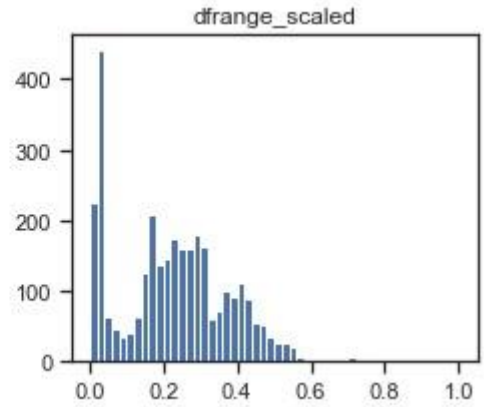

mindom
1200
1000
800
600
400
200
0
0.0
0.1
0.2
0.3
0.4

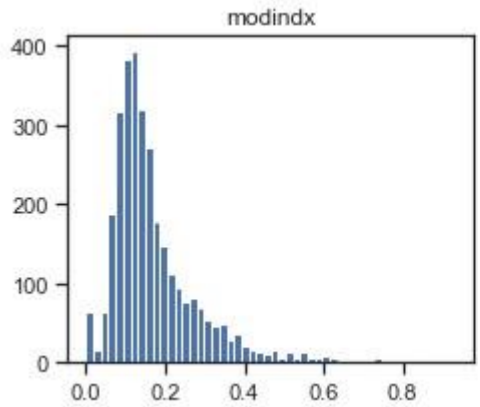
mindom_scaled
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

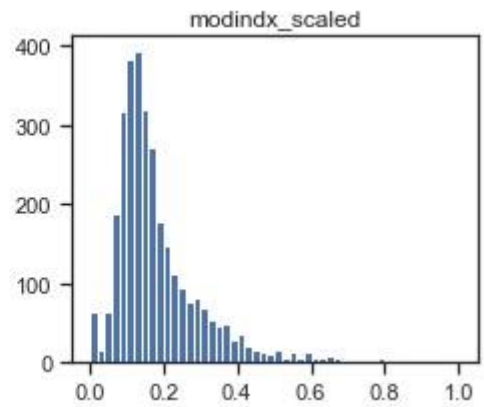

maxdom
400
300
200
100
0
0
5
10
15
20


maxdom_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0


dfrange
400
300
200
100
0
0
5
10
15
20


dfrange_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0


modindx
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8


modindx_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

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

In [31]:

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

Out[31]:

```
['meanfreq',
 'sd',
 'median',
 'Q25',
 'Q75',
 'IQR',
 'skew',
 'kurt',
 'sp.ent',
 'sfm',
 'mode',
 'centroid',
 'meanfun',
 'minfun',
 'maxfun',
 'meandom',
 'mindom',
 'maxdom',
 'dfrange',
 'modindx',
 'label']
```

In [32]:

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

Out[32]:

```
['meanfreq_scaled',
 'sd_scaled',
 'median_scaled',
 'Q25_scaled',
 'Q75_scaled',
 'IQR_scaled',
 'skew_scaled',
 'kurt_scaled',
 'sp.ent_scaled',
 'sfm_scaled',
 'mode_scaled',
 'centroid_scaled',
 'meanfun_scaled',
 'minfun_scaled',
```

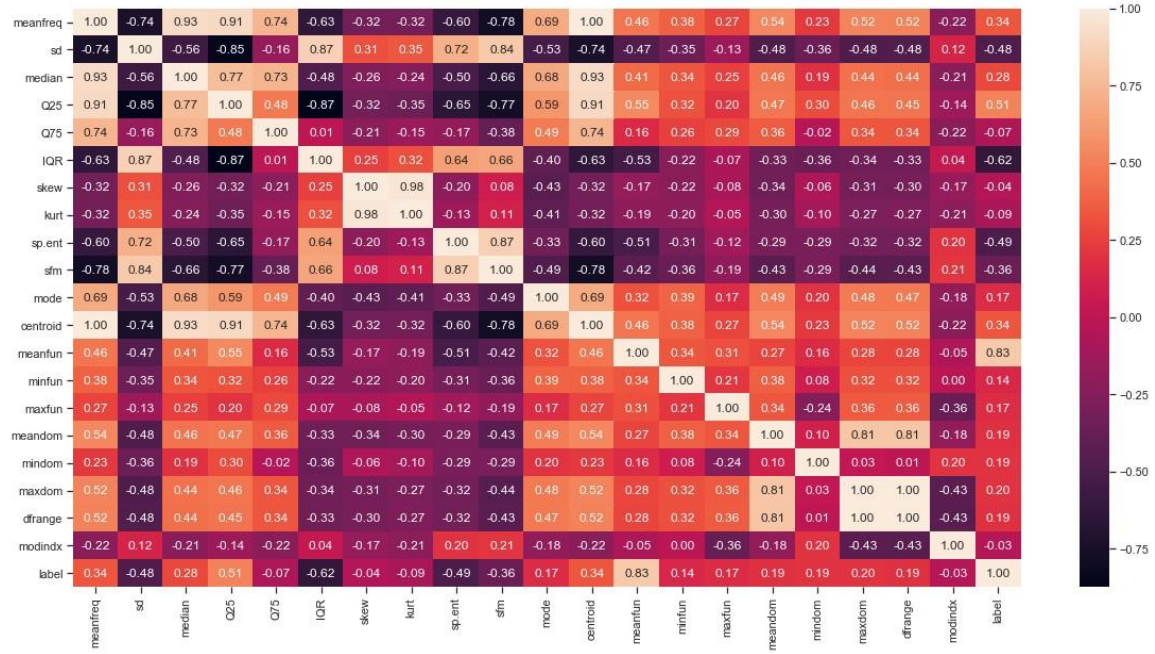
```
 'maxfun_scaled',
 'meandom_scaled',
 'mindom_scaled',
 'maxdom_scaled',
 'dfrange_scaled',
 'modindx_scaled',
 'label']
```

In [33]:

```
fig, ax = plt.subplots(figsize=(20,10))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at

0x27b7707c190>

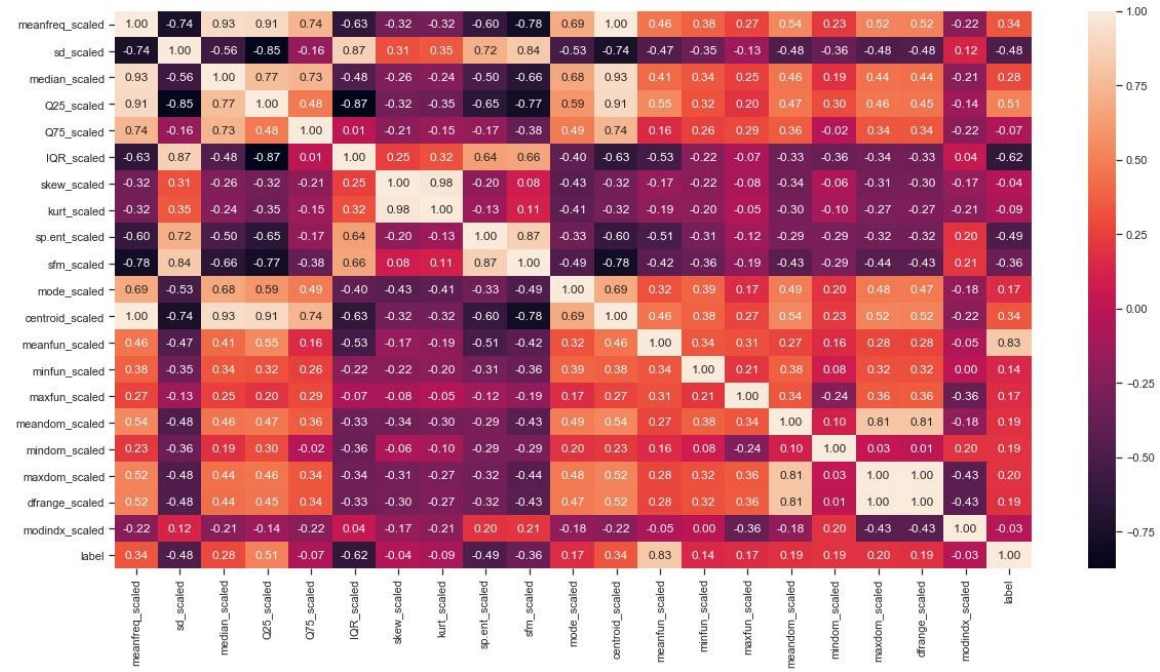


In [34]:

```
fig, ax = plt.subplots(figsize=(20,10))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

Out[34]:

<matplotlib.axes._subplots.AxesSubplot at 0x27b7a82d760>



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак классификации "label" наиболее сильно коррелирует с meanfun (0.83), IQR (-0.62)
- Все признаки в разной степень коррелируют друг с другом, все стоит оставить
- Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5) Выбор метрик для последующей оценки качествамоделей.

5.1) В качестве метрик для решения задачи классификации будем использовать:

Метрики, формируемые на основе матрицы ошибок:

1. Метрика accuracy:

2. Метрика recall (полнота):

3. Метрика precision

4. Метрика ROC AUC

5.2) Сохранение и визуализация метрик

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

In [35]:

```
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()
```

6) Выбор наиболее подходящих моделей для решениязадачи классификации или регрессии.

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия
- Метод ближайших соседей
- Машина опорных векторов
- Решающее дерево
- Случайный лес
- Градиентный бустинг

7) Формирование обучающей и тестовой выборок на основеисходного набора данных.

Test и train уже были сформированы выше

8) Построение базового решения (baseline) для выбранныхмоделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

In [36]:

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

In [37]:

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

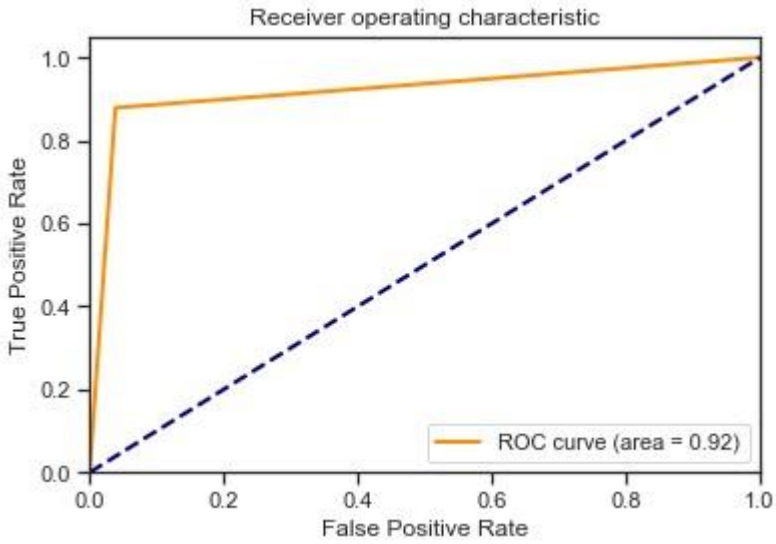
In [38]:

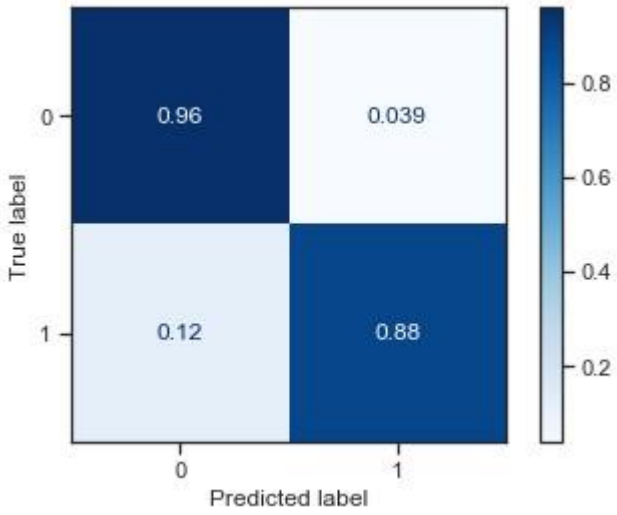
```
def clas_train_model(model_name, model, clasMetricLogger):
 model.fit(x_train, y_train)    Y_pred =
 model.predict(x_test)
    precision = precision_score(y_test.values, Y_pred)
 recall = recall_score(y_test.values, Y_pred)    accuracy
 = accuracy_score(y_test.values, Y_pred)    roc_auc =
 roc_auc_score(y_test.values, Y_pred)

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

    print('************************************************')
 print(model)
    print('************************************************')
 draw_roc_curve(y_test.values, Y_pred)

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

In [39]:

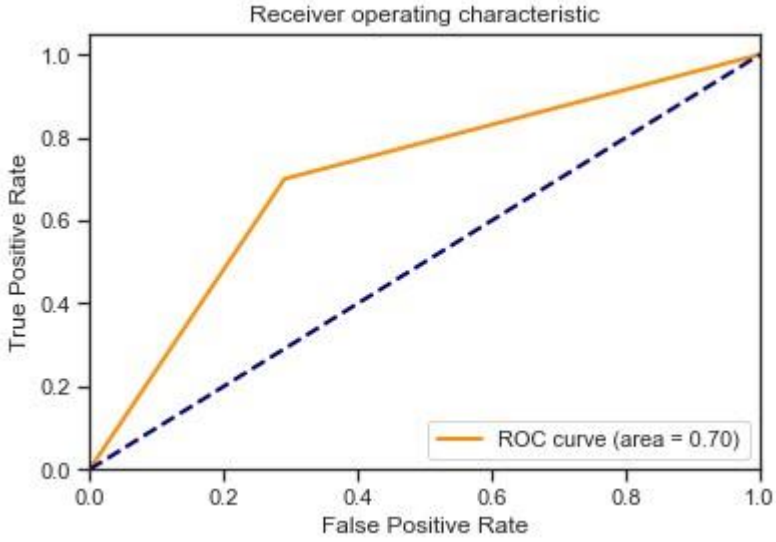
```
for model_name, model in clas_models.items():
    model.max_iter = 10000000

    clas_train_model(model_name, model, clasMetricLogger)
```

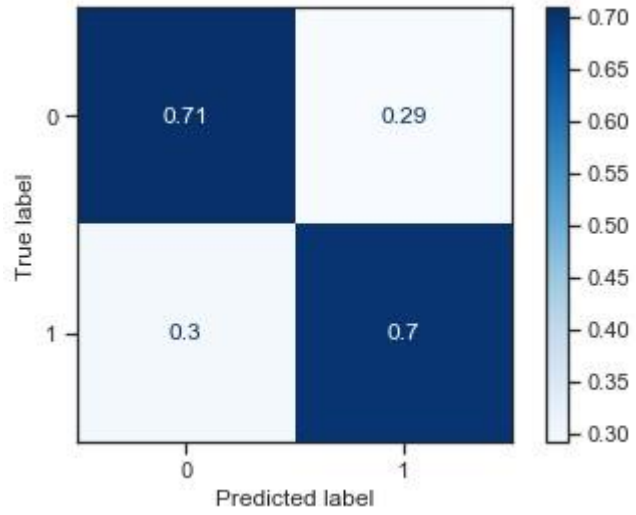
```
**************************************************
LogisticRegression(max_iter=10000000)
**************************************************
```



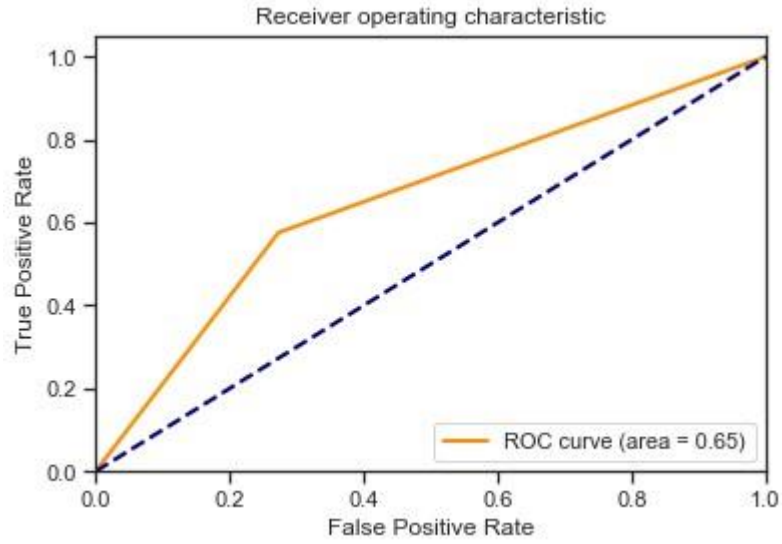


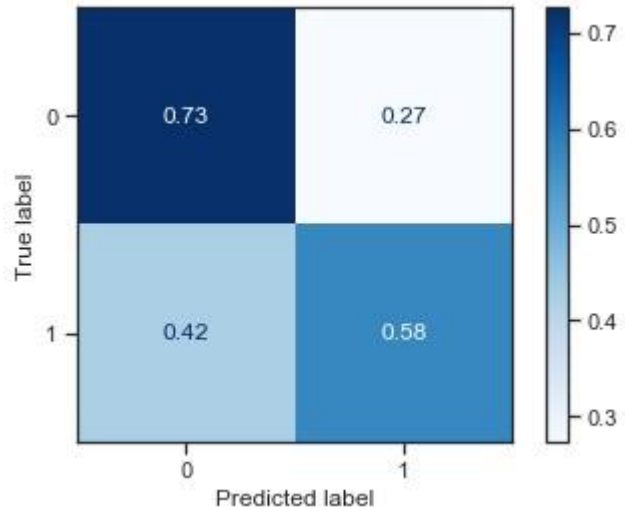

```
************************************************** KNeighborsClassifier()
**************************************************
```



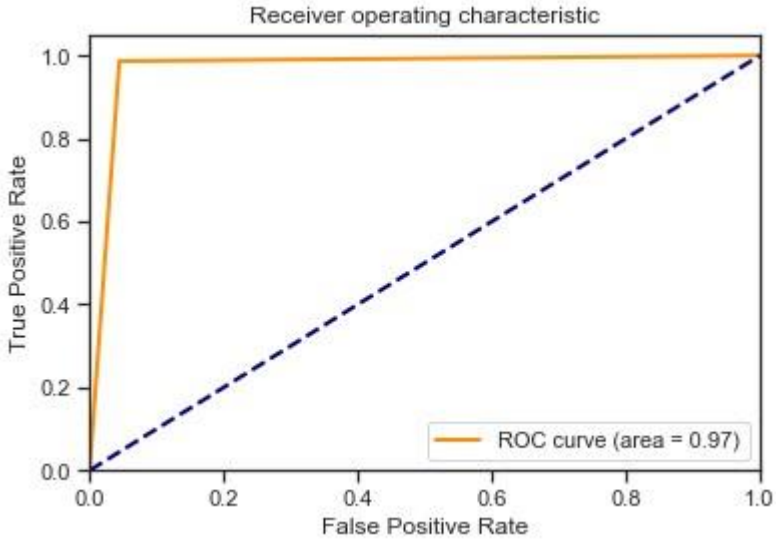


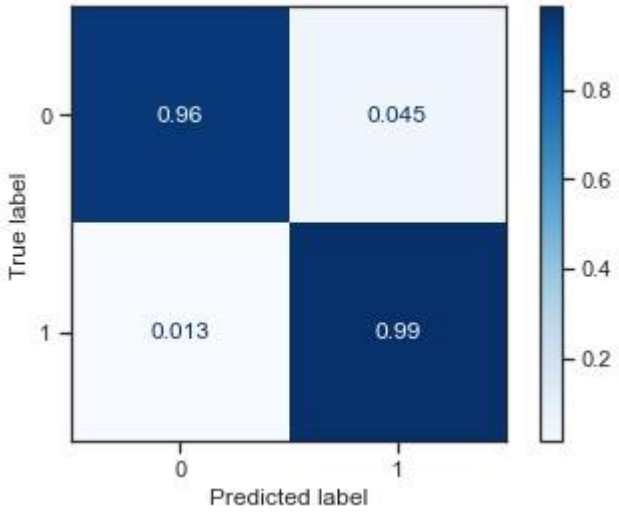
********** SVC(max_iter=10000000)
****





****
DecisionTreeClassifier()
****

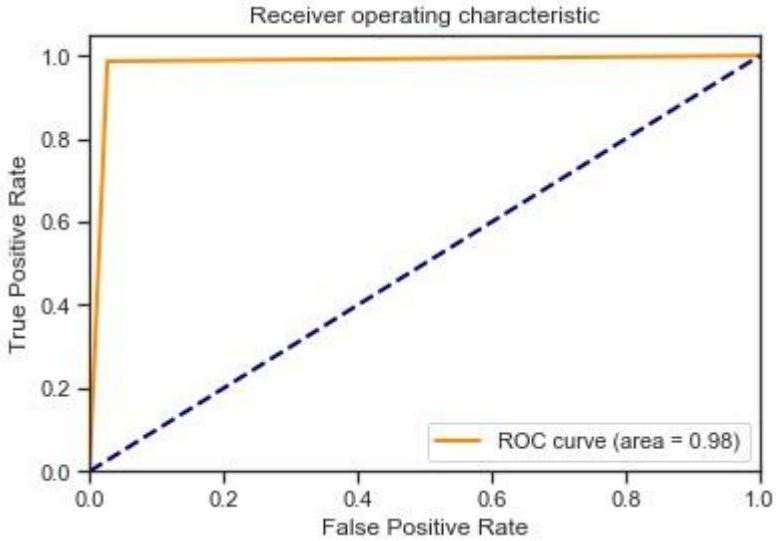


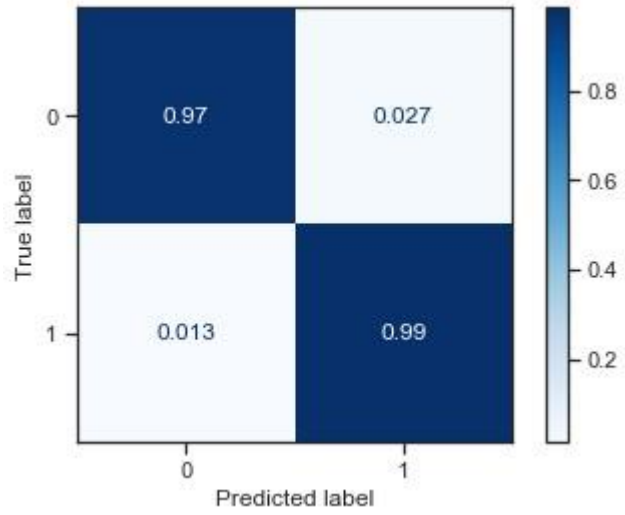


****

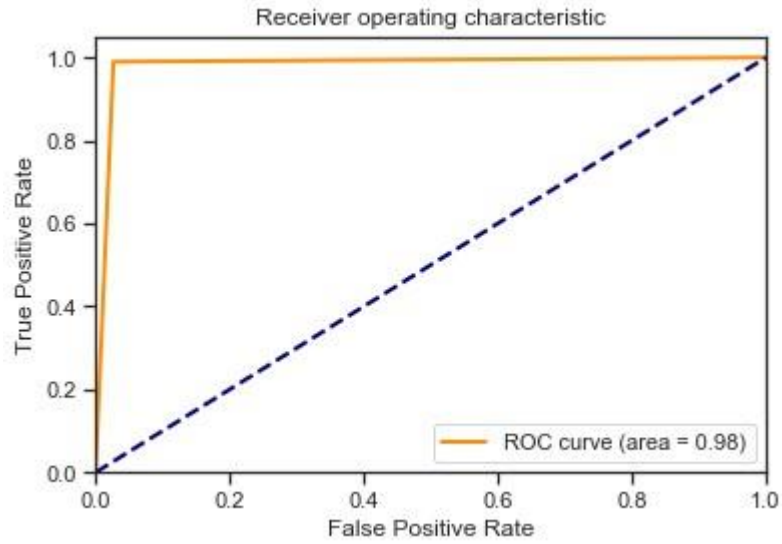
RandomForestClassifier()

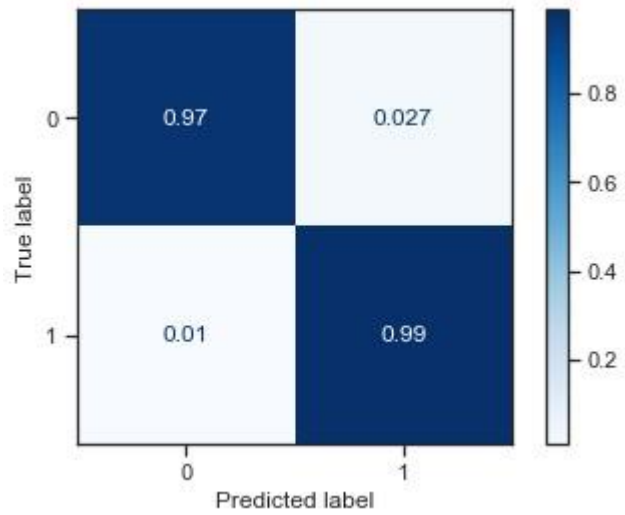
****





****
GradientBoostingClassifier()
****





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 [40]:

```
x_train.shape
```

Out[40]:

```
(2534, 20)
```

In [68]:

```
n_range = np.array(range(1,2000,10))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[68]:

```
[{'n_neighbors': array([   1,   11,   21,   31,   41,   51,   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,  411,  421,  431,
          441,  451,  461,  471,  481,  491,  501,  511,  521,  531,  541,
          551,  561,  571,  581,  591,  601,  611,  621,  631,  641,  651,
          661,  671,  681,  691,  701,  711,  721,  731,  741,  751,  761,
          771,  781,  791,  801,  811,  821,  831,  841,  851,  861,  871,
          881,  891,  901,  911,  921,  931,  941,  951,  961,  971,  981,
          991, 1001, 1011, 1021, 1031, 1041, 1051, 1061, 1071, 1081, 1091,
         1101, 1111, 1121, 1131, 1141, 1151, 1161, 1171, 1181, 1191, 1201,
         1211, 1221, 1231, 1241, 1251, 1261, 1271, 1281, 1291, 1301, 1311,
         1321, 1331, 1341, 1351, 1361, 1371, 1381, 1391, 1401, 1411, 1421,
         1431, 1441, 1451, 1461, 1471, 1481, 1491, 1501, 1511, 1521, 1531,
         1541, 1551, 1561, 1571, 1581, 1591, 1601, 1611, 1621, 1631, 1641,
         1651, 1661, 1671, 1681, 1691, 1701, 1711, 1721, 1731, 1741, 1751,
         1761, 1771, 1781, 1791, 1801, 1811, 1821, 1831, 1841, 1851, 1861,
         1871, 1881, 1891, 1901, 1911, 1921, 1931, 1941, 1951, 1961, 1971,
 1981, 1991])}]
```

In [69]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='roc_auc'
)
clf_gs.fit(x_train, y_train)
```

```
Wall time: 1min 29s
```

Out[69]:

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([   1,   11,   21,   31,
41,   51,   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,  411,  421,  431,
 441,  451,  461,  471,  481,  491,  501,  511,  521,  531,  541,
 551,  561,  571,  581,  591,  601,  611,  621,  631,  641,  651,
 661,  671,  681,...
1321, 1331, 1341, 1351, 1361, 1371, 1381, 1391, 1401, 1411, 1421,
1431, 1441, 1451, 1461, 1471, 1481, 1491, 1501, 1511, 1521, 1531,
1541, 1551, 1561, 1571, 1581, 1591, 1601, 1611, 1621, 1631, 1641,
1651, 1661, 1671, 1681, 1691, 1701, 1711, 1721, 1731, 1741, 1751,
1761, 1771, 1781, 1791, 1801, 1811, 1821, 1831, 1841, 1851, 1861,
1871, 1881, 1891, 1901, 1911, 1921, 1931, 1941, 1951, 1961, 1971,
1981, 1991])}],
```

scoring='roc_auc') In [70]:

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

Out[70]:

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

In [71]:

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

Out[71]:

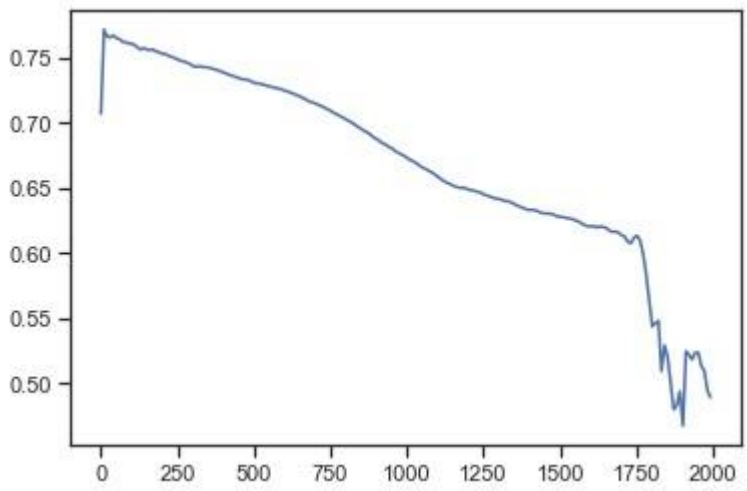
```
{'n_neighbors': 11}
```

In [72]:

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

Out[72]:

```
[<matplotlib.lines.Line2D at 0x27b79df8c40>]
```



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

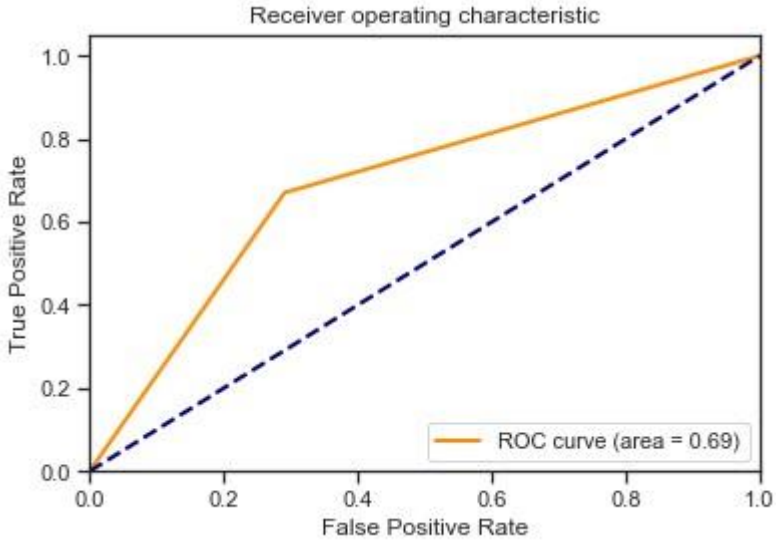
In [73]:

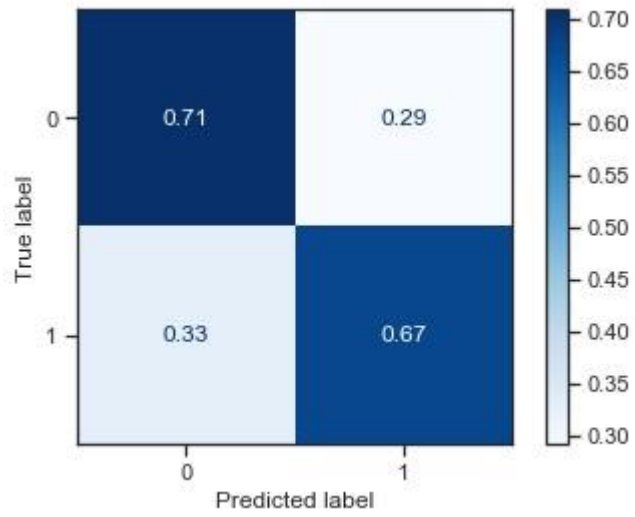
```
clas_models_grid = {'KNN_101':clf_gs.best_estimator_}
```

In [74]:

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

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





11) Формирование выводов о качестве построенныхмоделей на основе выбранных метрик.

In [48]:

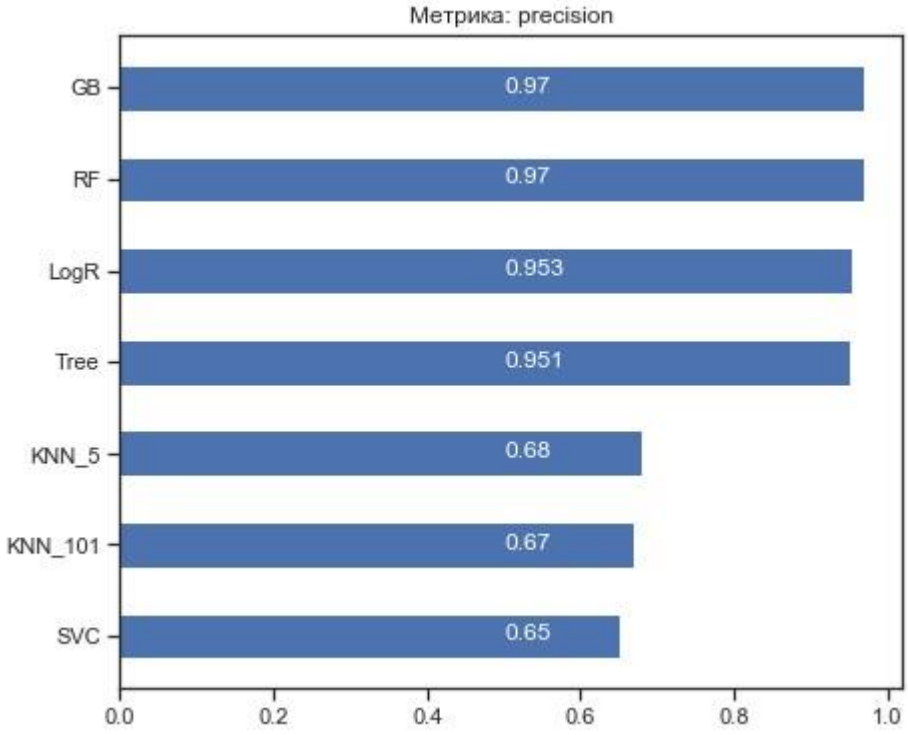
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

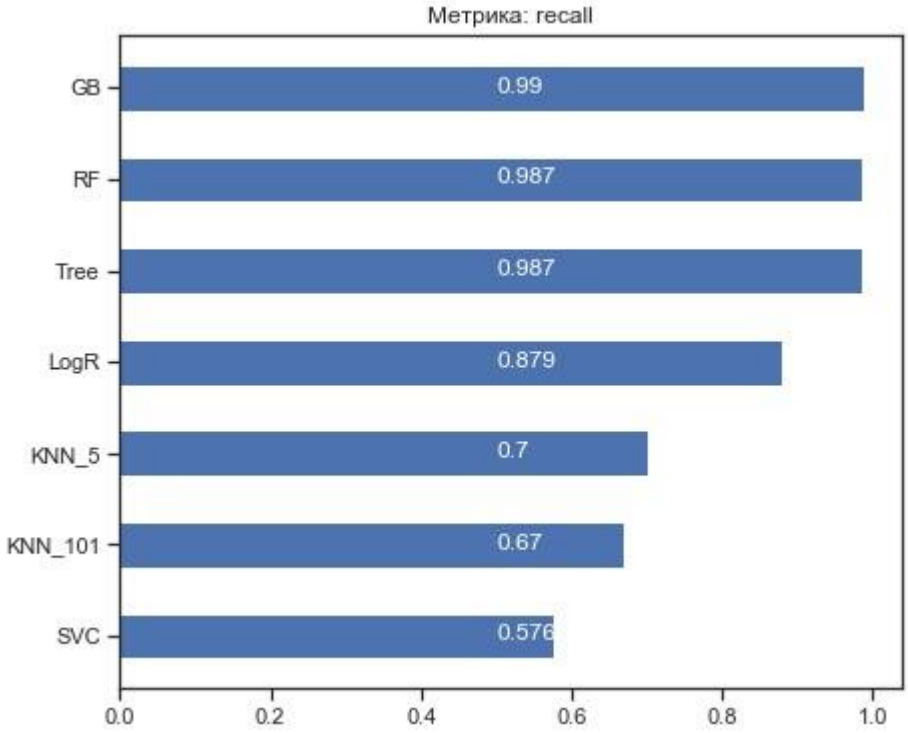
Out[48]:

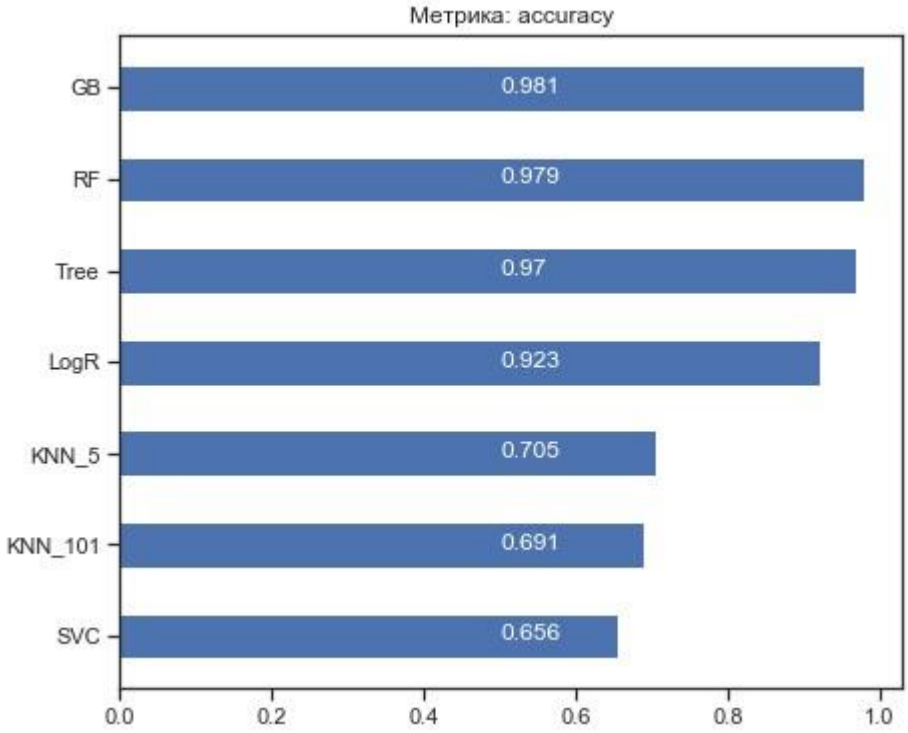
```
array(['precision', 'recall', 'accuracy', 'roc_auc'], dtype=object)
```

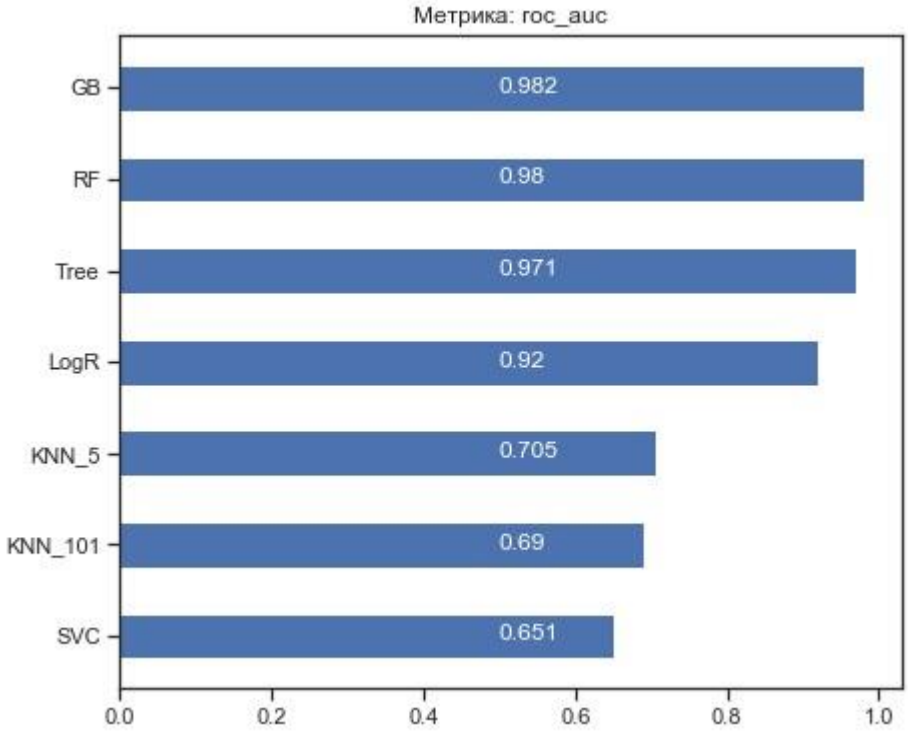
In [49]:

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


Метрика: precision
GB
RF
LogR
Tree
KNN_5
KNN_101
SVC
0.97
0.97
0.953
0.951
0.68
0.67
0.65
0.0
0.2
0.4
0.6
0.8
1.0


Метрика: recall
GB
RF
Tree
LogR
KNN_5
KNN_101
SVC
0.99
0.987
0.987
0.879
0.7
0.67
0.576
0.0
0.2
0.4
0.6
0.8
1.0





Вывод: на основании всех четырех используемых метрик, лучшей оказалась модель градиентный бустинг (GB).

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/