In [5]:

```
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_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_erro
r, 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 gmdhpy import gmdh
import lightgbm
%matplotlib inline
sns.set(style="ticks")
```

In [6]:

```
# 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()
```

In [7]:

```
data = pd.read_csv('/home/perlink/anaconda3/cardio1.csv', sep=",")
data = data[:10000]
```

In [8]:

```
data.head()
```

Out[8]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

In [9]:

```
data.shape
```

Out[9]:

```
(10000, 13)
```

In [10]:

```
data.columns
```

Out[10]:

```
Index(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio'],
      dtype='object')
```

In [11]:

```
data.dtypes
```

Out[11]:

```
id               int64
age              int64
gender           int64
height           int64
weight         float64
ap_hi            int64
ap_lo            int64
cholesterol      int64
gluc             int64
smoke            int64
alco             int64
active           int64
cardio           int64
dtype: object
```

In [12]:

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

Out[12]:

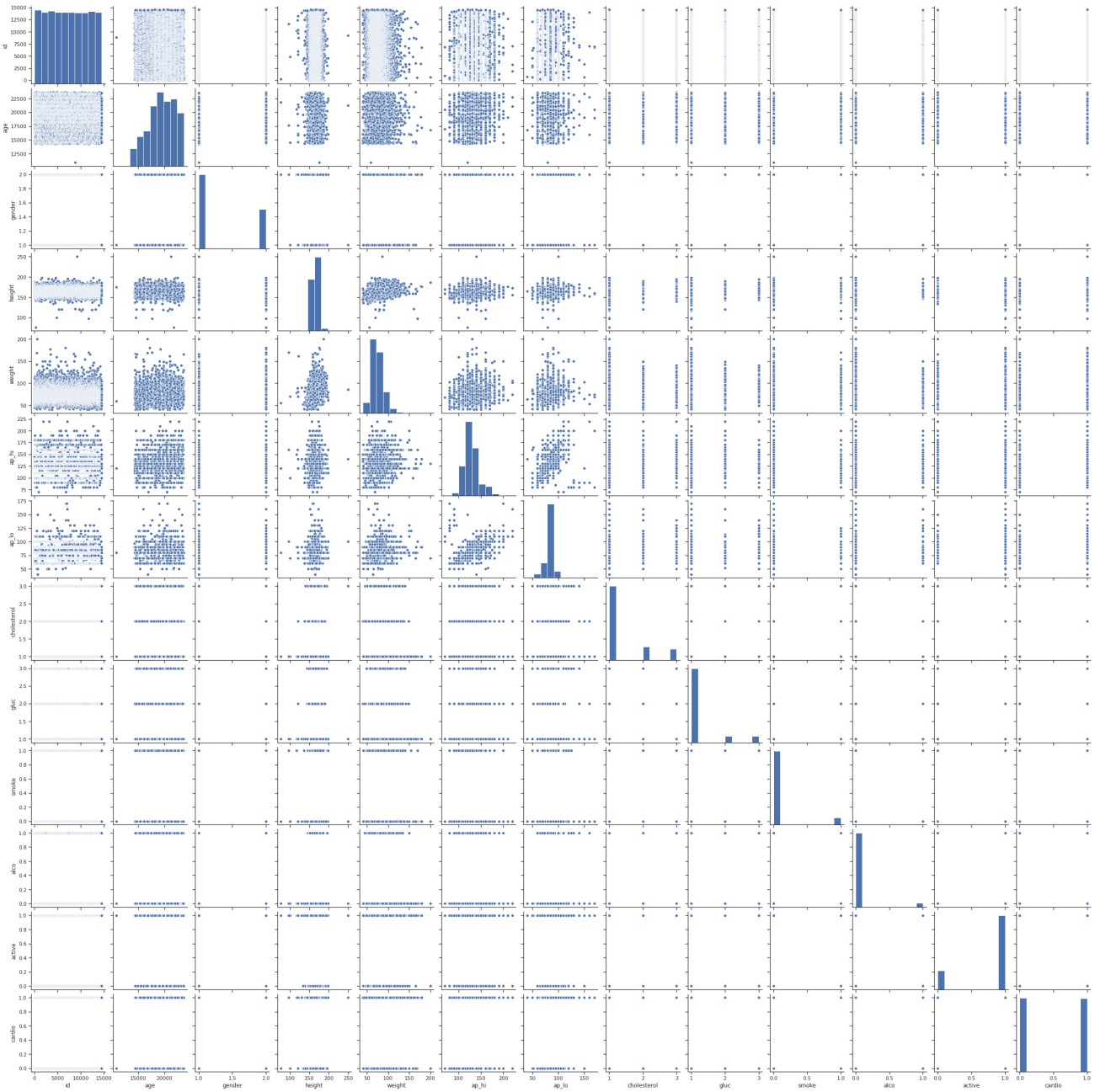
```
id             0
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

In [13]:

```
sns.pairplot(data)
```

Out[13]:

<seaborn.axisgrid.PairGrid at 0x7f228029b590>



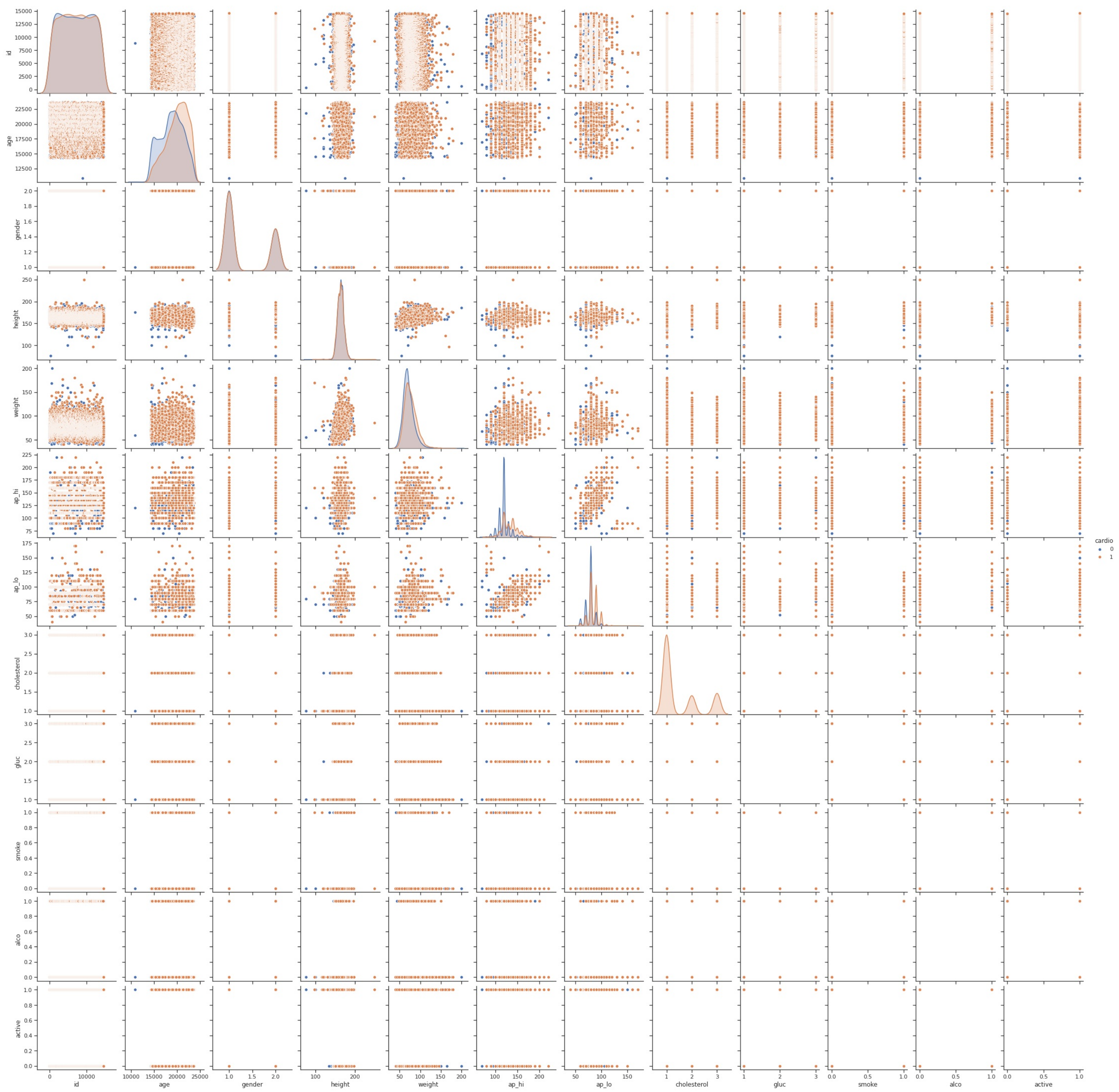
In [98]:

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

```
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/home/perlink/.local/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default
bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```

Out[98]:

```
<seaborn.axisgrid.PairGrid at 0x7f6b93afea10>
```



In [14]:

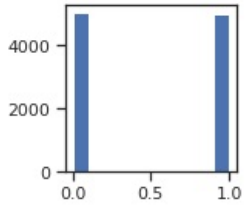
```
data['cardio'].unique()
```

Out[14]:

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

In [15]:

```
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['cardio'])
plt.show()
```



In [16]:

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

Out[16]:

```
0    5028
1    4972
Name: cardio, dtype: int64
```

In [17]:

```
total = data.shape[0]
class_0, class_1 = data['cardio'].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 50.28%, а класс 1 составляет 49.72%.
```

Вывод: Дисбаланс практически отсутсвует.

In [18]:

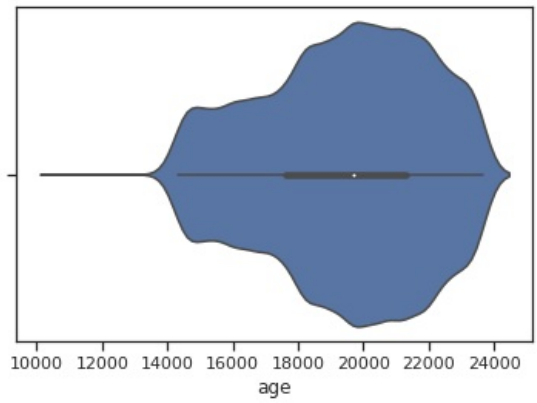
```
data.columns
```

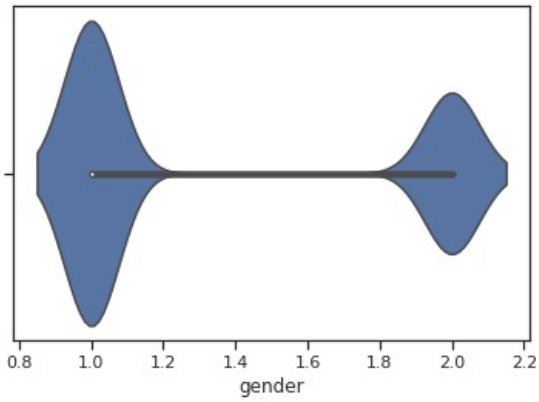
Out[18]:

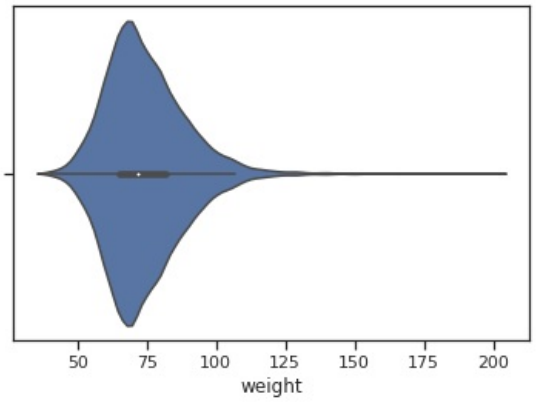
```
Index(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio'],
      dtype='object')
```

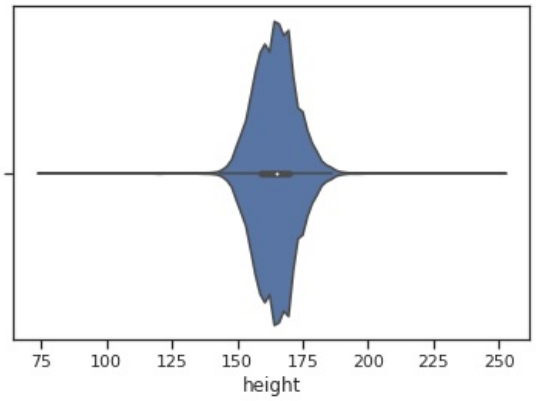
In [19]:

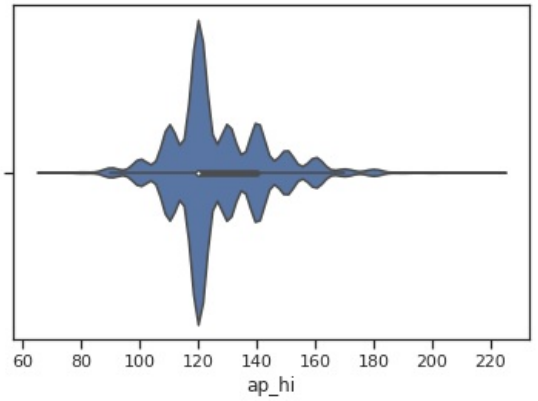
```
for col in ['age', 'gender', 'weight','height','ap_hi','ap_lo', 'cholesterol', 'gluc','smoke','alco', 'active']:
    sns.violinplot(x=data[col])
    plt.show()
```

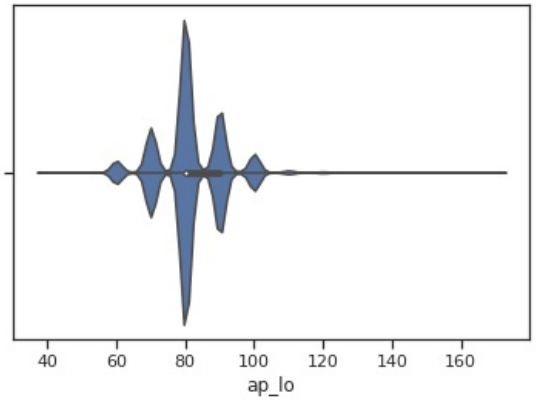


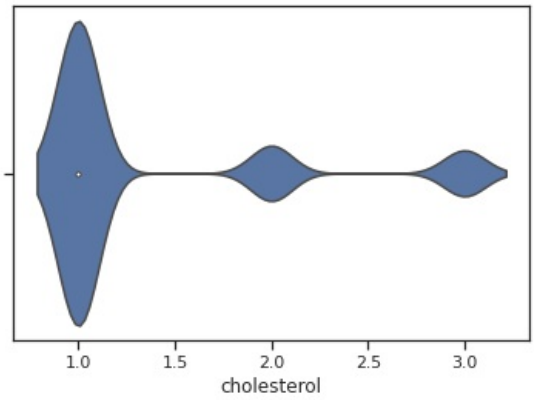

0.8
1.0
1.2
1.4
1.6
1.8
2.0
2.2
gender

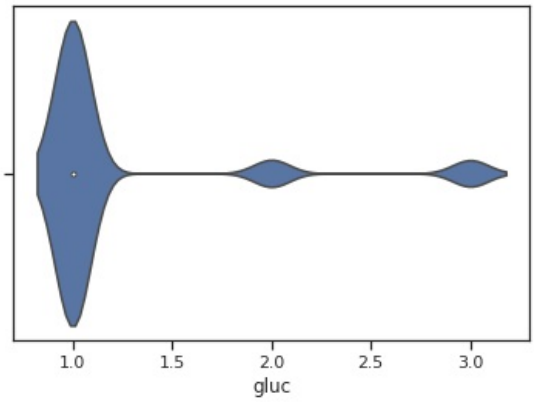

50
75
100
125
150
175
200
weight

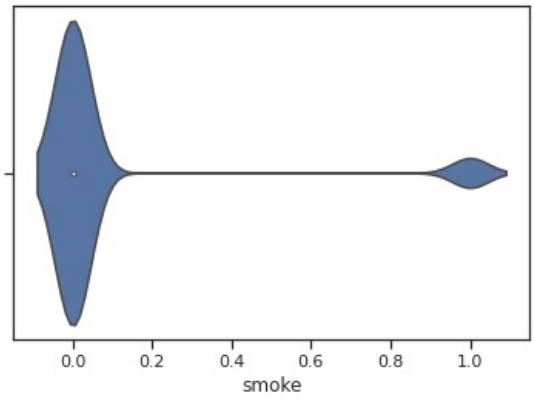

75
100
125
150
175
200
225
250
height

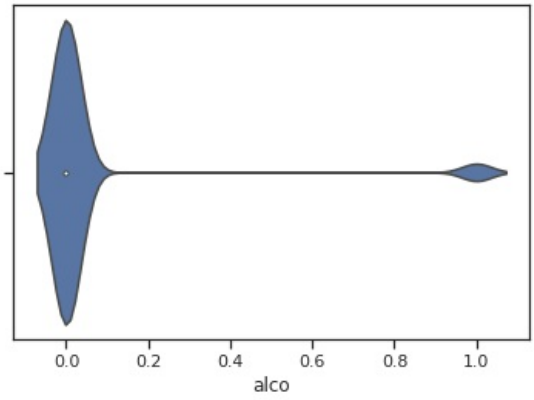

60
80
100
120
140
160
180
200
220
ap_hi

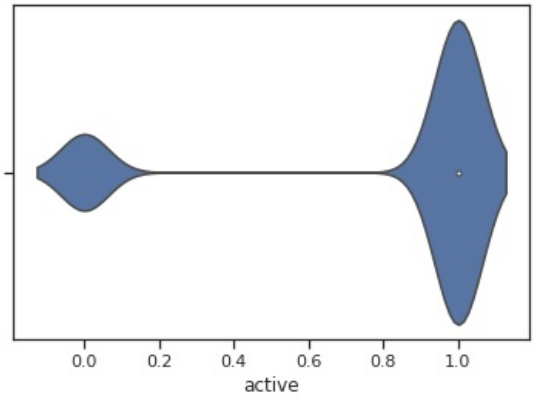

40
60
80
100
120
140
160
ap_lo


1.0
1.5
2.0
2.5
3.0
cholesterol


1.0
1.5
2.0
2.5
3.0
gluc


0.0
0.2
0.4
0.6
0.8
1.0
smoke


0.0
0.2
0.4
0.6
0.8
1.0
alco


0.0
0.2
0.4
0.6
0.8
1.0
active

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

In [20]:

```
data.dtypes
```

Out[20]:

```
id                int64
age               int64
gender            int64
height            int64
weight          float64
ap_hi             int64
ap_lo             int64
cholesterol       int64
gluc              int64
smoke             int64
alco              int64
active            int64
cardio            int64
dtype: object
```

Для построения моделей будем использовать все признаки, кроме признака id.

Категориальные признаки отсутствуют, их кодирования не требуется. Все признаки уже закодированы.

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

Выполним масштабирование данных.

In [21]:

```
scale_cols = ['age', 'gender', 'weight','height','ap_hi','ap_lo', 'cholesterol', 'gluc','smoke','alco', 'active']
```

In [22]:

```
sc1 = MinMaxScaler()
sc1_data = sc1.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] = sc1_data[:,i]
```

In [24]:

```
data.head()
```

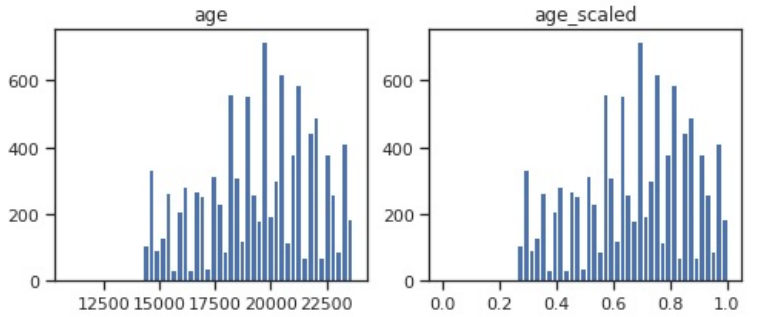
Out[24]:

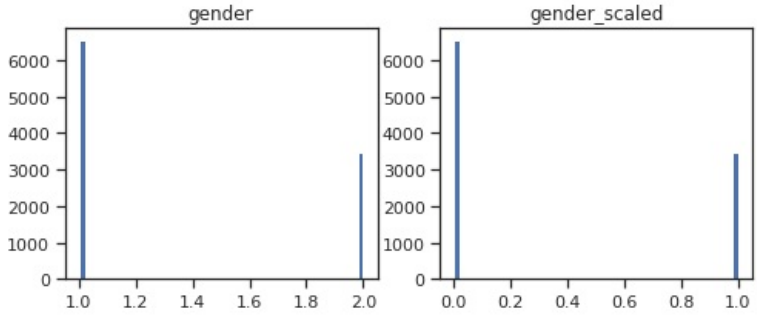
	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	...	gender_scaled	weight_scaled	height_scaled	ap_hi_s
0	0	18393	2	168	62.0	110	80	1	1	0	...	1.0	0.13750	0.528736	0.2
1	1	20228	1	156	85.0	140	90	3	1	0	...	0.0	0.28125	0.459770	0.4
2	2	18857	1	165	64.0	130	70	3	1	0	...	0.0	0.15000	0.511494	0.4
3	3	17623	2	169	82.0	150	100	1	1	0	...	1.0	0.26250	0.534483	0.5
4	4	17474	1	156	56.0	100	60	1	1	0	...	0.0	0.10000	0.459770	0.2

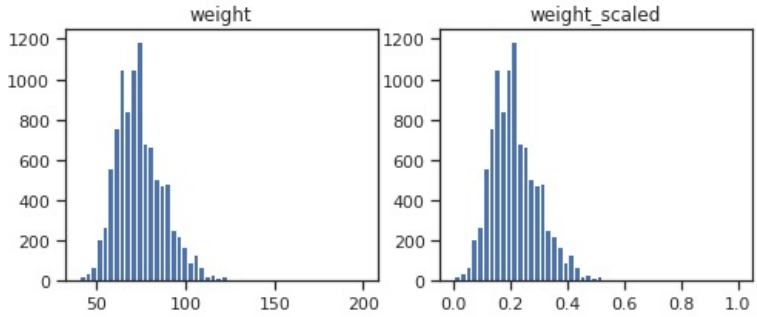
5 rows × 24 columns

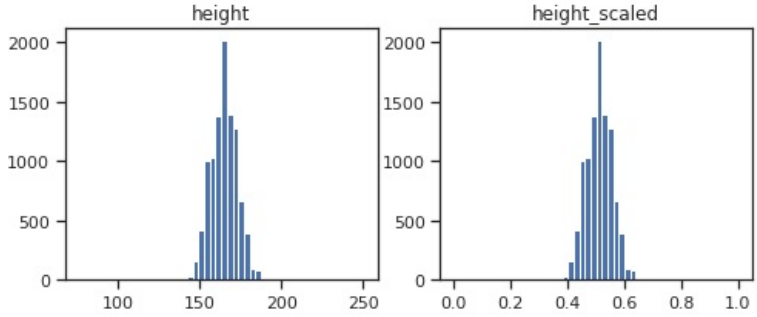
In [25]:

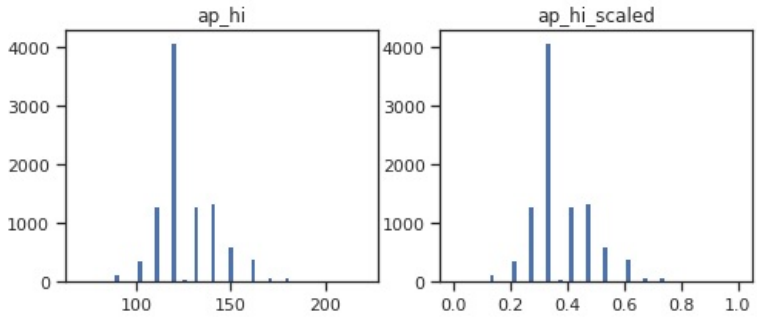
```
for col in scale_cols:
    col_scaled = col + '_scaled'

    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()
```

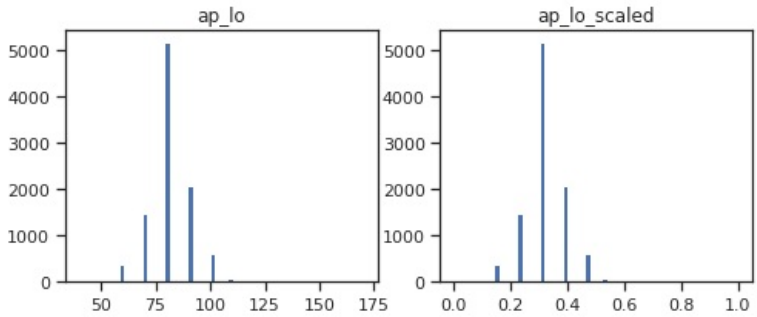

age
age_scaled

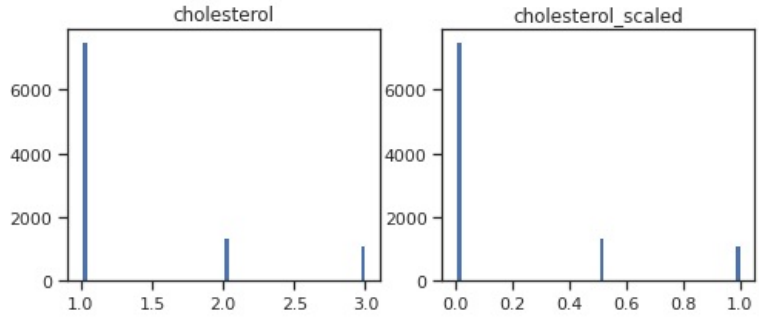

gender
gender_scaled

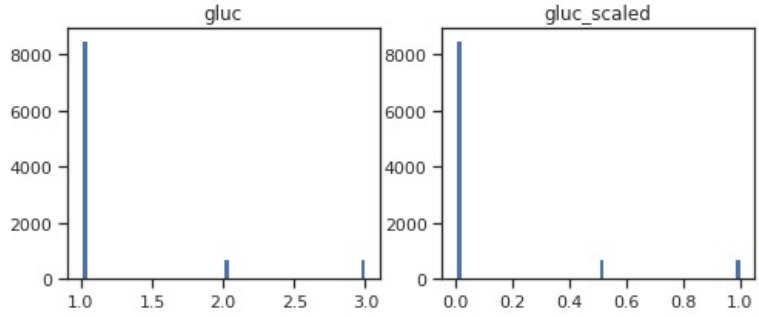

weight
weight_scaled

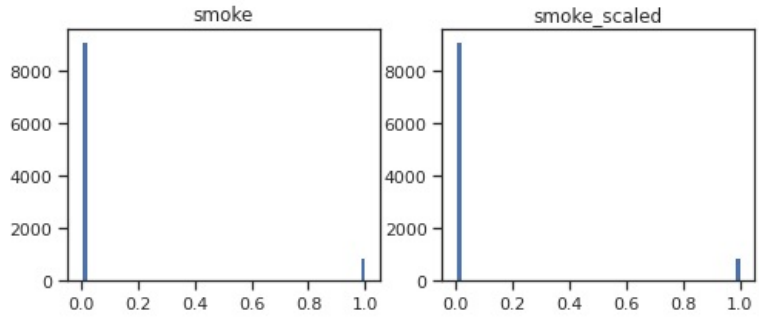

height
height_scaled

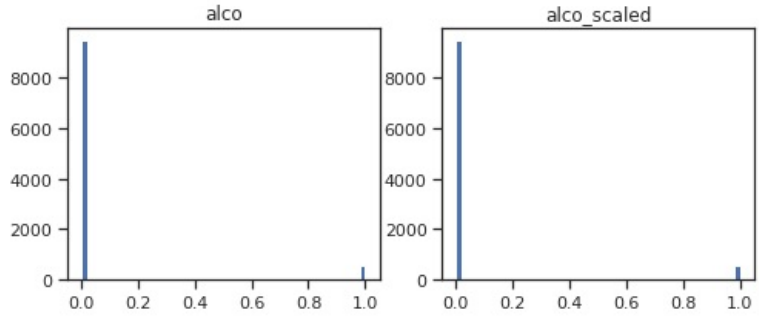

ap_hi
ap_hi_scaled

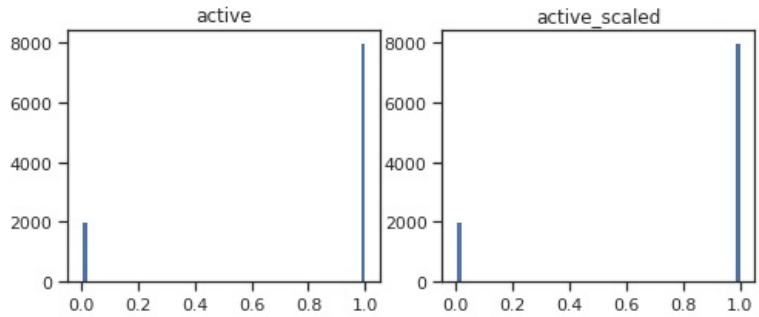

ap_lo
ap_lo_scaled











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

In [26]:

```
corr_cols_1 = scale_cols + ['cardio']
corr_cols_1
```

Out[26]:

```
['age',
 'gender',
 'weight',
 'height',
 'ap_hi',
 'ap_lo',
 'cholesterol',
 'gluc',
 'smoke',
 'alco',
 'active',
 'cardio']
```

In [27]:

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

Out[27]:

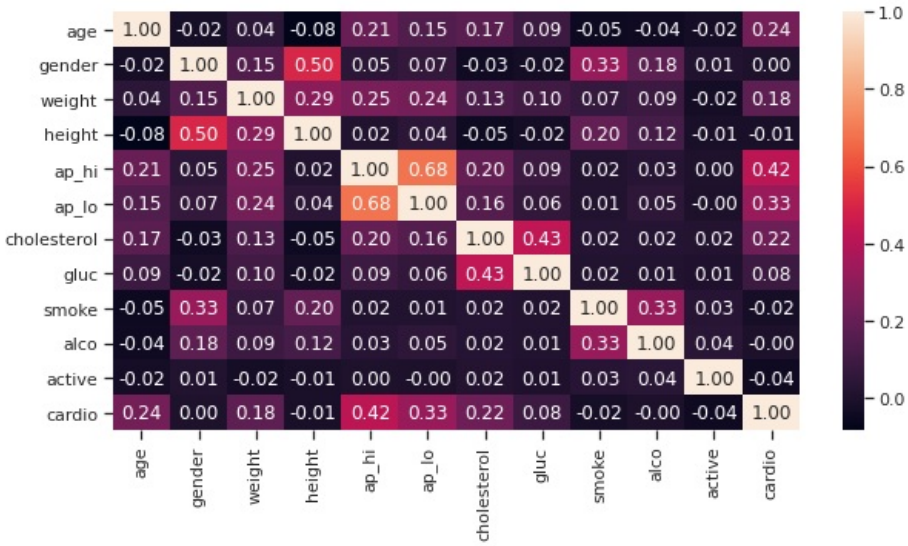
```
['age_scaled',
 'gender_scaled',
 'weight_scaled',
 'height_scaled',
 'ap_hi_scaled',
 'ap_lo_scaled',
 'cholesterol_scaled',
 'gluc_scaled',
 'smoke_scaled',
 'alco_scaled',
 'active_scaled',
 'cardio']
```

In [28]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f222416f650>
```

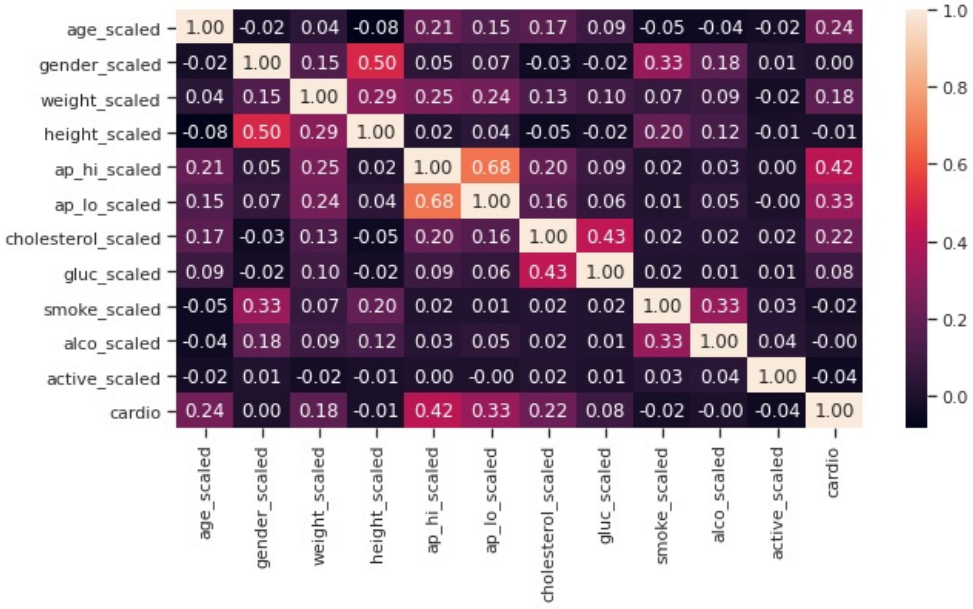


In [29]:

```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22240246d0>
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак классификации "cardio" наиболее сильно коррелирует с давлением верхним (0.42), давлением нижним (0.33), возрастом (0.24) и холестерином (0.22).
- Верхнее и нижнее давления коррелирую достаточно сильно между собой (0.68), поэтому оставим только верхнее, т.к. оно сильнее коррелирует с целевым признаком. Остальные следует оставить в модели классификации.
- 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 качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

- Precision - доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.
- Recall - доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.
- F_1-мера - для объединения precision и recall в единую метрику
- ROC AUC основана на вычислении следующих характеристик:
  - True Positive Rate, откладывается по оси ординат. Совпадает с recall.
  - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

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

In [30]:

```
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 решения задачи классификации или регрессии.

Для задачи классификации будем использовать следующие модели:

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

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

In [31]:

```
# Признаки для задачи классификации
task_clas_cols = ['age_scaled', 'ap_hi_scaled','cholesterol_scaled',
                  'weight_scaled']
```

In [32]:

```
X = data[task_clas_cols]
Y = data['cardio']
X.shape
```

Out[32]:

```
(10000, 4)
```

In [33]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[33]:

```
((8000, 4), (2000, 4), (8000,), (2000,))
```

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 [34]:

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

In [35]:

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

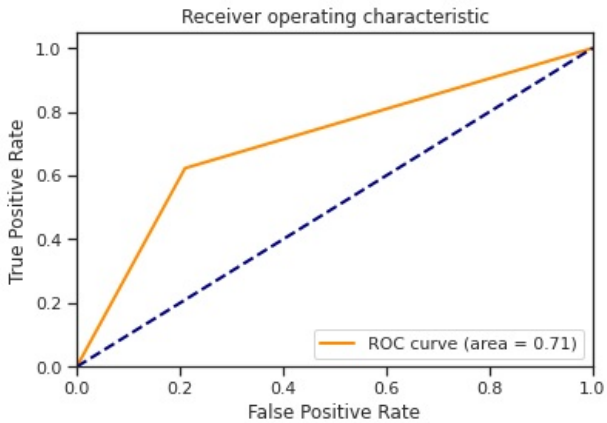
In [36]:

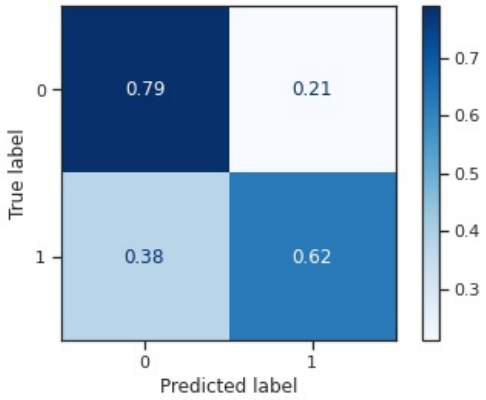
```
def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    precision = precision_score(Y_test.values, Y_pred)
    recall = recall_score(Y_test.values, Y_pred)
    f1 = f1_score(Y_test.values, Y_pred)
    roc_auc = roc_auc_score(Y_test.values, Y_pred)

    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    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 [37]:

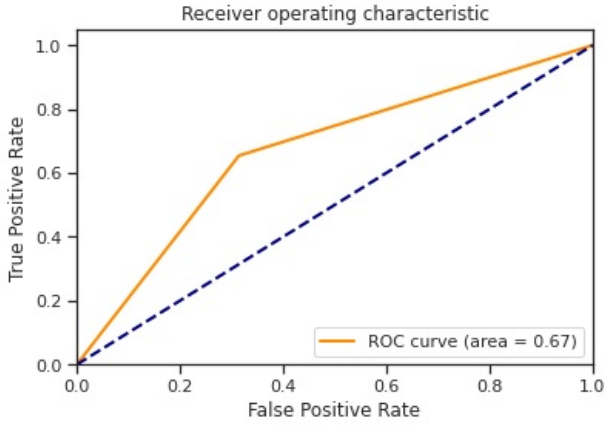
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

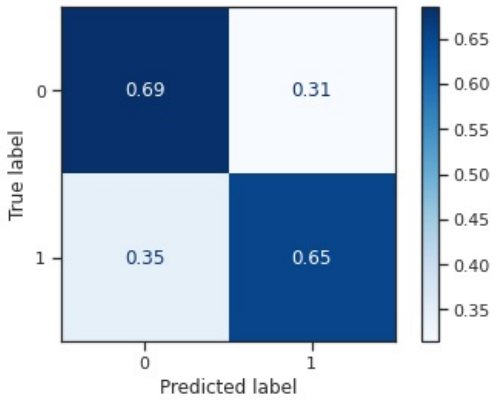
```
*******************************************************
LogisticRegression()
*******************************************************
```



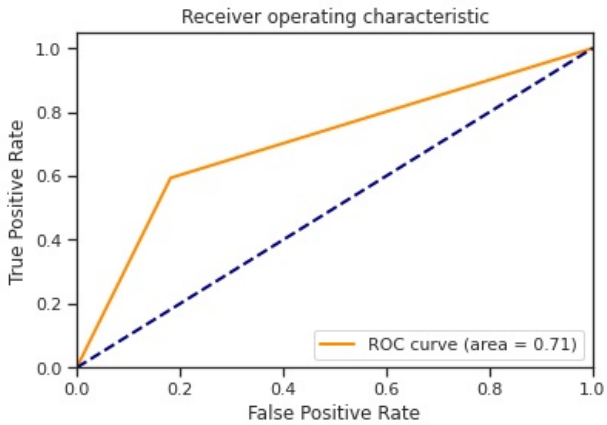


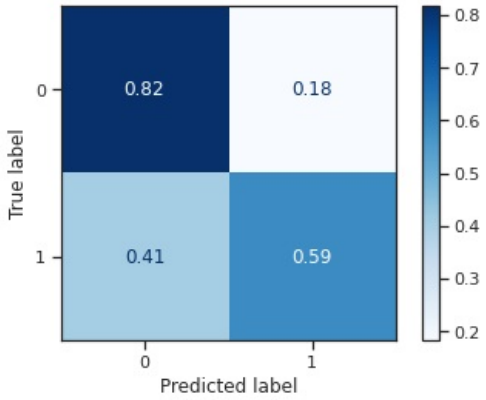
****
KNeighborsClassifier()
****



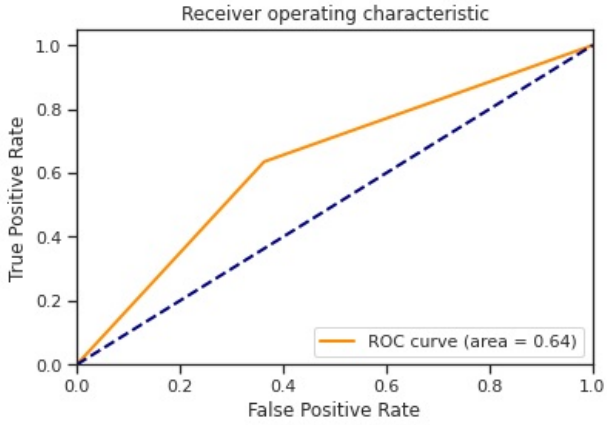


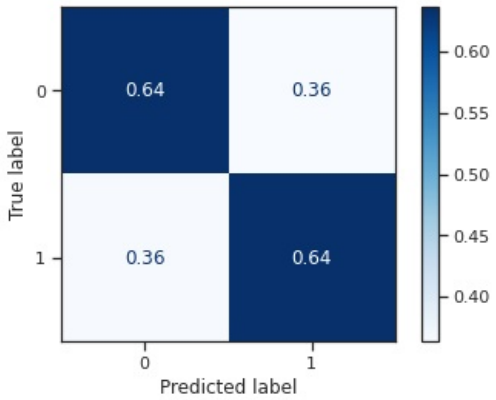
****
SVC()
****



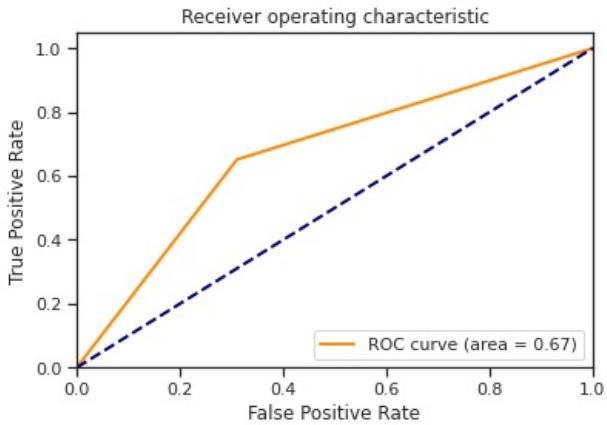


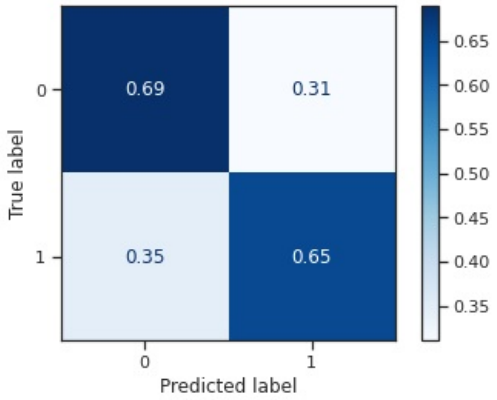
****
DecisionTreeClassifier()
****



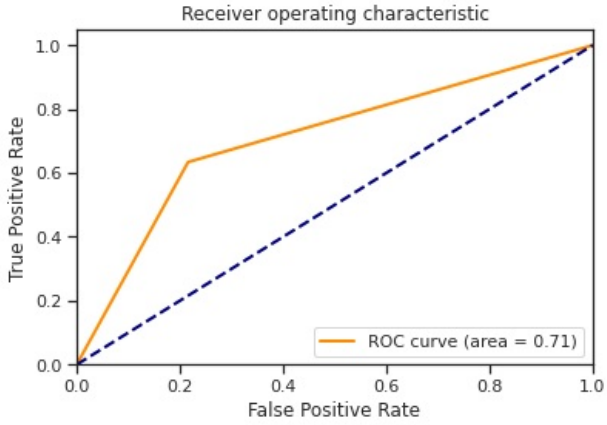


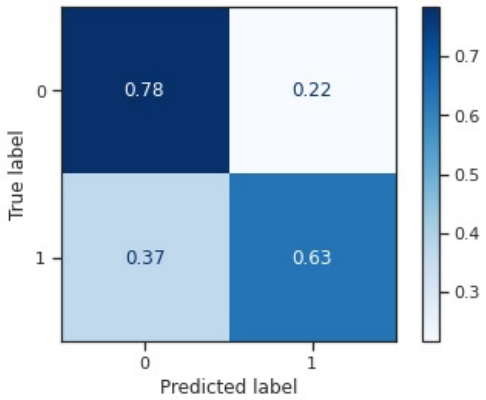
****
RandomForestClassifier()
****



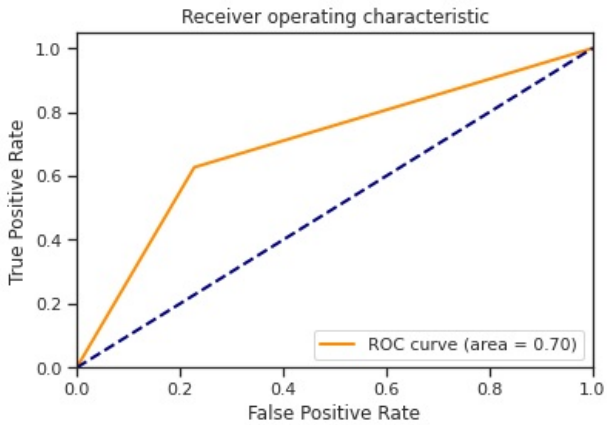


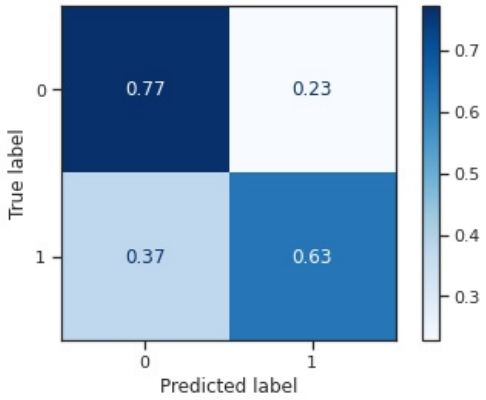
****
GradientBoostingClassifier()
****





****
LGBMClassifier()
****





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 [38]:

```
X_train.shape
```

Out[38]:

```
(8000, 4)
```

In [39]:

```
n_range = np.array(range(1,2000,100))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[39]:

```
[{'n_neighbors': array([   1,  101,  201,  301,  401,  501,  601,  701,  801,  901, 1001,
         1101, 1201, 1301, 1401, 1501, 1601, 1701, 1801, 1901])}]
```

In [40]:

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

```
CPU times: user 29.7 s, sys: 808 ms, total: 30.5 s
Wall time: 30.6 s
```

Out[40]:

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([   1,  101,  201,  301,  401,  501,  601,  701,  801
,  901, 1001,
       1101, 1201, 1301, 1401, 1501, 1601, 1701, 1801, 1901])}],
             scoring='roc_auc')
```

In [41]:

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

Out[41]:

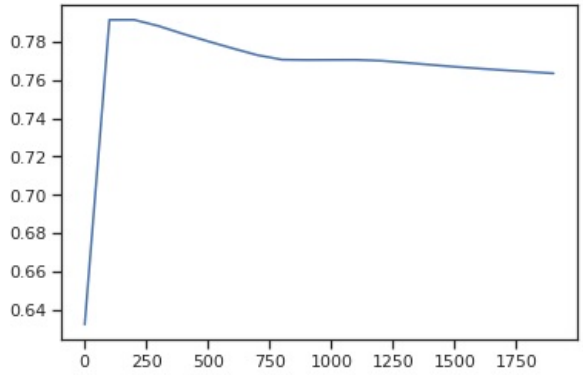
```
KNeighborsClassifier(n_neighbors=201)
```

In [42]:

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

Out[42]:

```
[<matplotlib.lines.Line2D at 0x7f2224e95310>]
```



In [43]:

```
%%time
grid={"C":np.logspace(-3,3,3)}
gs_LogR = GridSearchCV(LogisticRegression(), grid, cv=5, scoring='roc_auc')
gs_LogR.fit(X_train, Y_train)
```

```
CPU times: user 3.17 s, sys: 3.08 s, total: 6.25 s
Wall time: 581 ms
```

Out[43]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e+00, 1.e+03])},
             scoring='roc_auc')
```

In [44]:

```
# Лучшая модель
gs_LogR.best_estimator_
```

Out[44]:

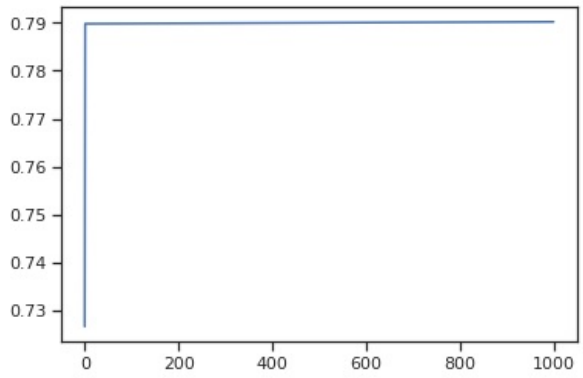
```
LogisticRegression(C=1000.0)
```

In [45]:

```
# Изменение качества на тестовой выборке в зависимости от C
plt.plot(np.logspace(-3,3,3), gs_LogR.cv_results_['mean_test_score'])
```

Out[45]:

```
[<matplotlib.lines.Line2D at 0x7f2224b4cf90>]
```



In [58]:

```
%%time
GB_params={"max_features":range(1,4), "max_leaf_nodes":range(2,22)}
gs_GB = GridSearchCV(GradientBoostingClassifier(), GB_params, cv=5, scoring='recall')
gs_GB.fit(X_train, Y_train)
```

```
CPU times: user 1min 2s, sys: 24.7 ms, total: 1min 2s
Wall time: 1min 2s
```

Out[58]:

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'max_features': range(1, 4),
                         'max_leaf_nodes': range(2, 22)},
             scoring='recall')
```

In [59]:

```
gs_GB.best_estimator_
```

Out[59]:

```
GradientBoostingClassifier(max_features=3, max_leaf_nodes=5)
```

In [52]:

```
%%time
LGBM_params={
    'bagging_fraction': (0.5, 0.8),
    'bagging_frequency': (5, 8),
    'feature_fraction': (0.5, 0.8),
    'max_depth': (10, 13),
    'min_data_in_leaf': (90, 120),
    'num_leaves': (1200, 1550)}
gs_LGBM = GridSearchCV(lightgbm.LGBMClassifier(), LGBM_params, cv=5, scoring='roc_auc')
gs_LGBM.fit(X_train, Y_train)
```

```
CPU times: user 7min 20s, sys: 2.22 s, total: 7min 22s
Wall time: 38.9 s
```

Out[52]:

```
GridSearchCV(cv=5, estimator=LGBMClassifier(),
             param_grid={'bagging_fraction': (0.5, 0.8),
                         'bagging_frequency': (5, 8),
                         'feature_fraction': (0.5, 0.8), 'max_depth': (10, 13),
                         'min_data_in_leaf': (90, 120),
                         'num_leaves': (1200, 1550)},
             scoring='roc_auc')
```

In [53]:

```
gs_LGBM.best_estimator_
```

Out[53]:

```
LGBMClassifier(bagging_fraction=0.5, bagging_frequency=5, feature_fraction=0.8,
               max_depth=10, min_data_in_leaf=120, num_leaves=1200)
```

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

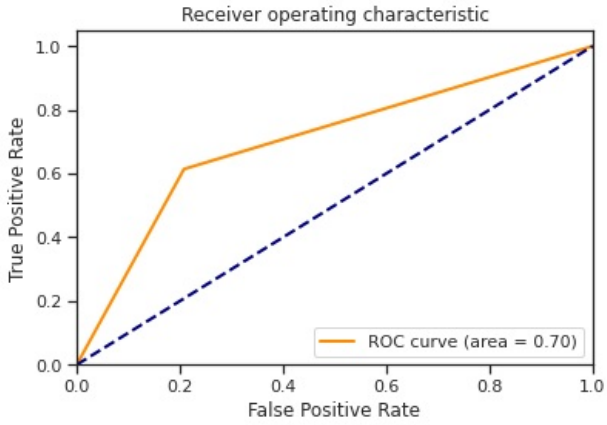
In [60]:

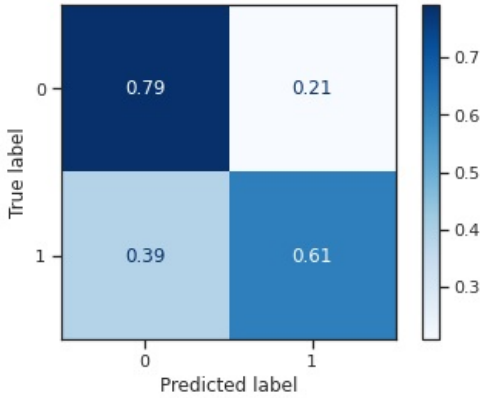
```
clas_models_grid = {'KNN_201':clf_gs.best_estimator_, 'LogR_new':gs_LogR.best_estimator_,
                    'GB_new':gs_GB.best_estimator_, 'LGBM_new':gs_LGBM.best_estimator_}
```

In [61]:

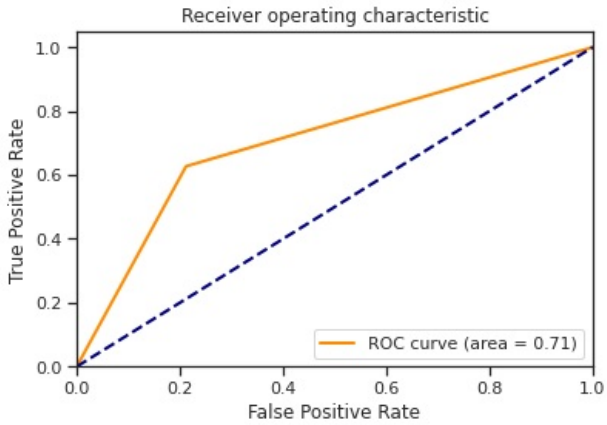
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

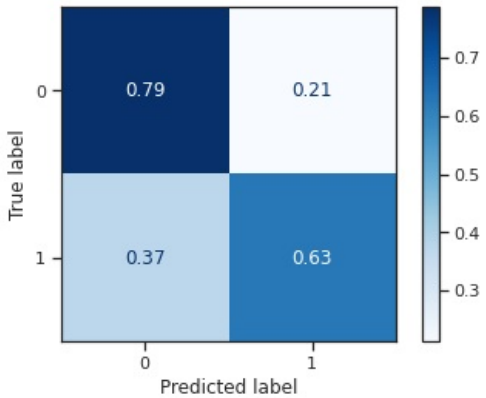
```
**************************************************
KNeighborsClassifier(n_neighbors=201)
**************************************************
```



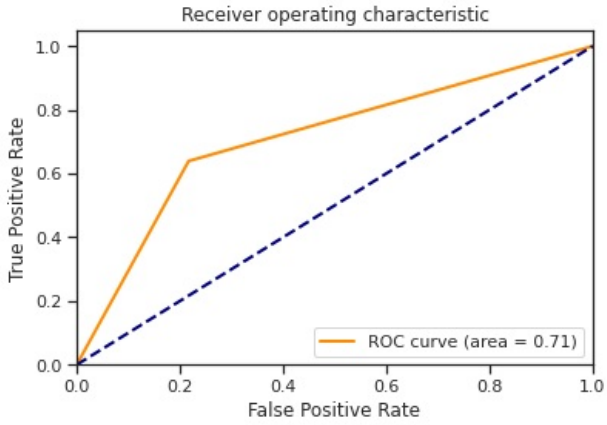


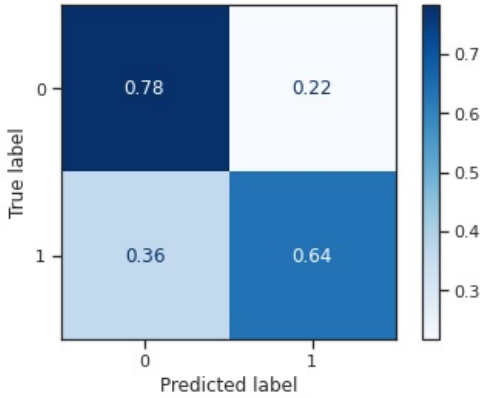
****
LogisticRegression(C=1000.0)
****



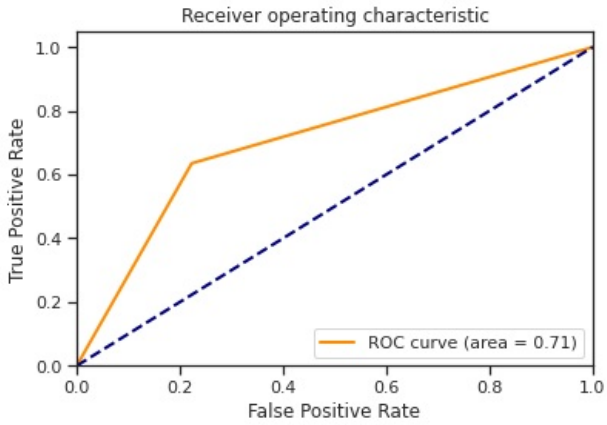


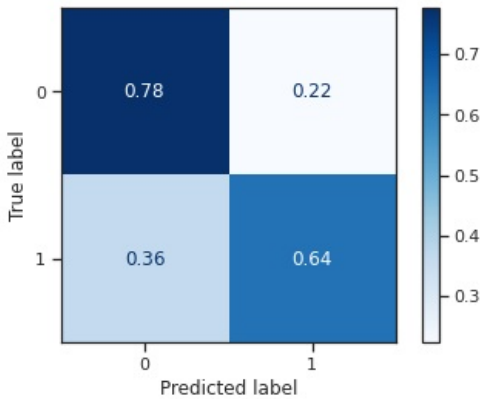
****
GradientBoostingClassifier(max_features=3, max_leaf_nodes=5)
****






```
******************************************************
LGBMClassifier(bagging_fraction=0.5, bagging_frequency=5, feature_fraction=0.8,
               max_depth=10, min_data_in_leaf=120, num_leaves=1200)
******************************************************
```





In [ ]:

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 [62]:

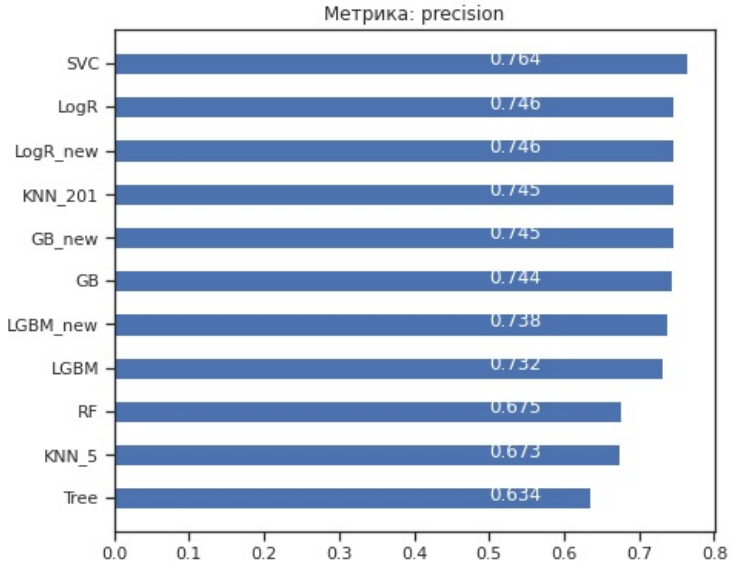
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

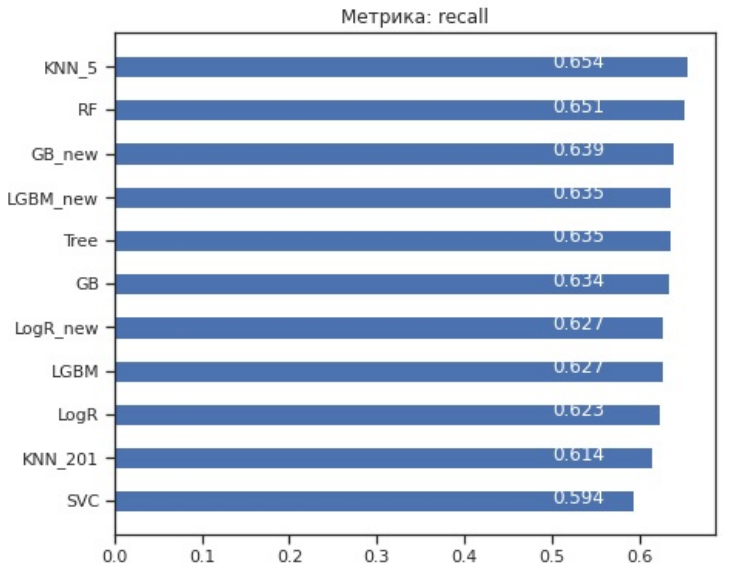
Out[62]:

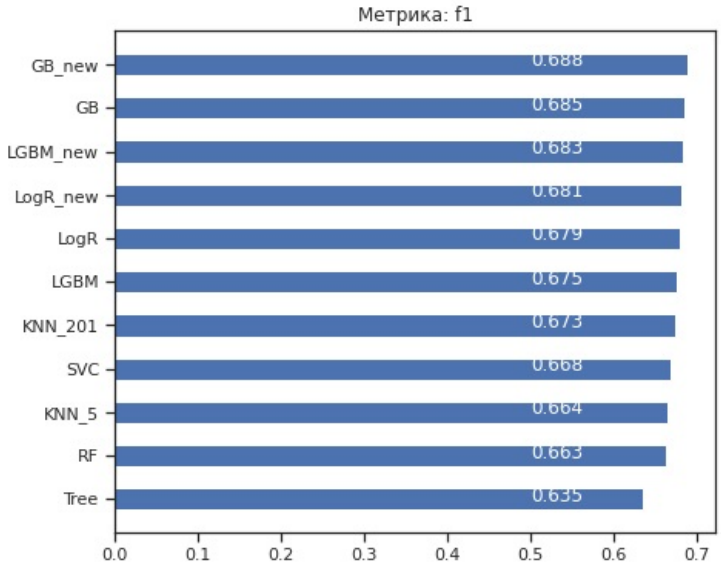
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

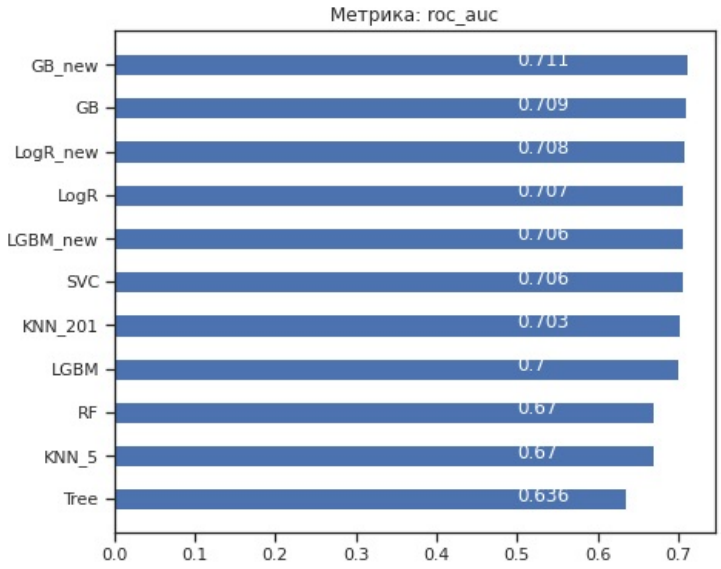
In [63]:

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









Вывод: на основании двух метрик из четырех используемых, лучшей оказалась модель Градиентный бустинг.

In [ ]: