1) Поиск и выбор набора данных для построения моделей машинного обучения.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import streamlit as st
import matplotlib.pyplot as plt
from catboost import Pool, CatBoostClassifier
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, cla
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mea
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR,
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor,
from sklearn.ensemble import RandomForestClassifier, RandomForestRegress
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostin
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
sns.set(style="ticks")
```

In [2]:
```
col_list = ['Pelvic_incidence',
                'Pelvic_tilt',
                'Lumbar_lordosis_angle',
                'Sacral_slope',
                'Pelvic_radius',
                'Degree_spondylolisthesis',
                'Pelvic_slope',
                'Direct_tilt',
                'Thoracic_slope',
                'Cervical_tilt',
                'Sacrum_angle',
                'Scoliosis_slope',
                'Class_att',
                'To_drop']
data = pd.read_csv('Dataset_spine.csv', names=col_list, header=1, sep=",
data.drop('To_drop', axis=1, inplace=True)
```

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

In [3]:
```
data.head()
```

Out[3]:

	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_radius	Degree_
0	39.056951	10.060991	25.015378	28.995960	114.405425	
1	68.832021	22.218482	50.092194	46.613539	105.985135	
2	69.297008	24.652878	44.311238	44.644130	101.868495	
3	49.712859	9.652075	28.317406	40.060784	108.168725	
4	40.250200	13.921907	25.124950	26.328293	130.327871	

In [4]:
```
data.shape
```

Out[4]:
```
(309, 13)
```

In [5]:
```
data.columns
```

Out[5]:
```
Index(['Pelvic_incidence', 'Pelvic_tilt', 'Lumbar_lordosis_angle',
       'Sacral_slope', 'Pelvic_radius', 'Degree_spondylolisthesis',
       'Pelvic_slope', 'Direct_tilt', 'Thoracic_slope', 'Cervical_tilt',
       'Sacrum_angle', 'Scoliosis_slope', 'Class_att'],
      dtype='object')
```

In [6]:
```
data.dtypes
```

Out[6]:
```
Pelvic_incidence            float64
Pelvic_tilt                 float64
Lumbar_lordosis_angle       float64
Sacral_slope                float64
Pelvic_radius               float64
Degree_spondylolisthesis    float64
Pelvic_slope                float64
Direct_tilt                 float64
Thoracic_slope              float64
Cervical_tilt               float64
Sacrum_angle                float64
Scoliosis_slope             float64
Class_att                    object
dtype: object
```

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
Pelvic_incidence            0
Pelvic_tilt                 0
Lumbar_lordosis_angle       0
Sacral_slope                0
Pelvic_radius               0
Degree_spondylolisthesis    0
Pelvic_slope                0
Direct_tilt                 0
Thoracic_slope              0
Cervical_tilt               0
Sacrum_angle                0
Scoliosis_slope             0
Class_att                   0
dtype: int64
```

In [8]:
```
data['Class_att_le'] = data['Class_att'].map({'Abnormal': 1, 'Normal': 0
```

In [9]:
```
print(data.loc[:, ['Class_att', 'Class_att_le']])
```

```
    Class_att  Class_att_le
0    Abnormal             1
1    Abnormal             1
2    Abnormal             1
3    Abnormal             1
4    Abnormal             1
..        ...           ...
304    Normal             0
305    Normal             0
306    Normal             0
307    Normal             0
308    Normal             0

[309 rows x 2 columns]
```

In [10]:
```
data['Class_att'].value_counts()
```

Out[10]:
```
Abnormal    209
Normal      100
Name: Class_att, dtype: int64
```

In [11]:
```
data.head()
```

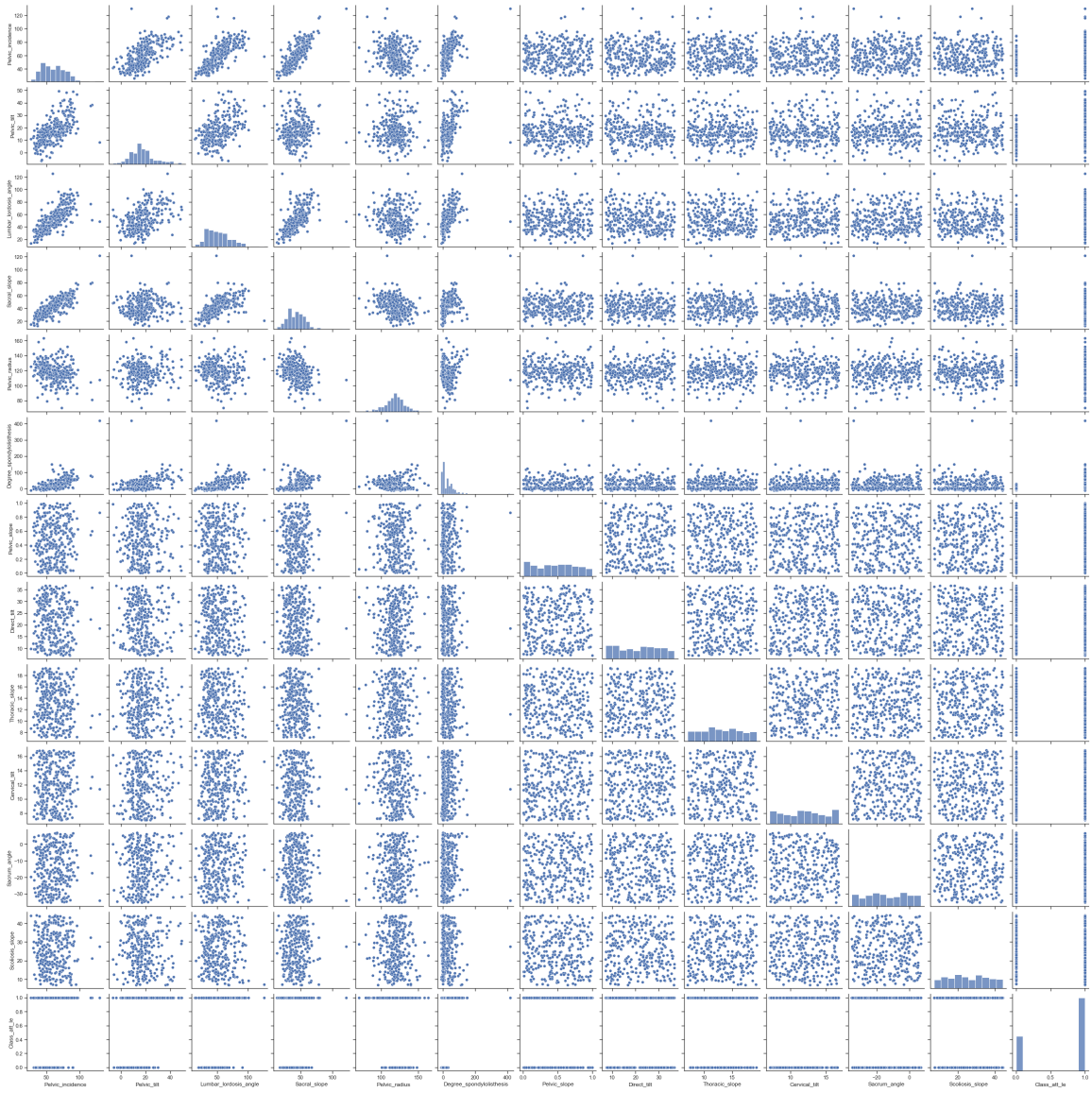
Out[11]:

	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_radius	Degree_
0	39.056951	10.060991	25.015378	28.995960	114.405425	
1	68.832021	22.218482	50.092194	46.613539	105.985135	
2	69.297008	24.652878	44.311238	44.644130	101.868495	
3	49.712859	9.652075	28.317406	40.060784	108.168725	
4	40.250200	13.921907	25.124950	26.328293	130.327871	

Набор данных не содержит пропусков, категориальные пизнаки закодированы.

In [12]:
```
sns.pairplot(data)
```

Out[12]:
```
<seaborn.axisgrid.PairGrid at 0x7ff5c5940100>
```



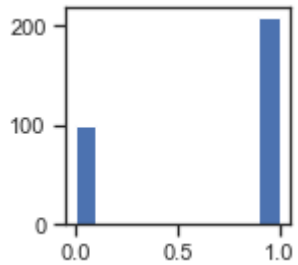
In [13]:

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

Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x7ff5b39ba9d0>
```

In [14]:
```
# Оценим дисбаланс классов для Class_att_le
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['Class_att_le'])
plt.show()
```



In [15]:
```
data['Class_att_le'].value_counts()
```

Out[15]:
```
1    209
0    100
Name: Class_att_le, dtype: int64
```

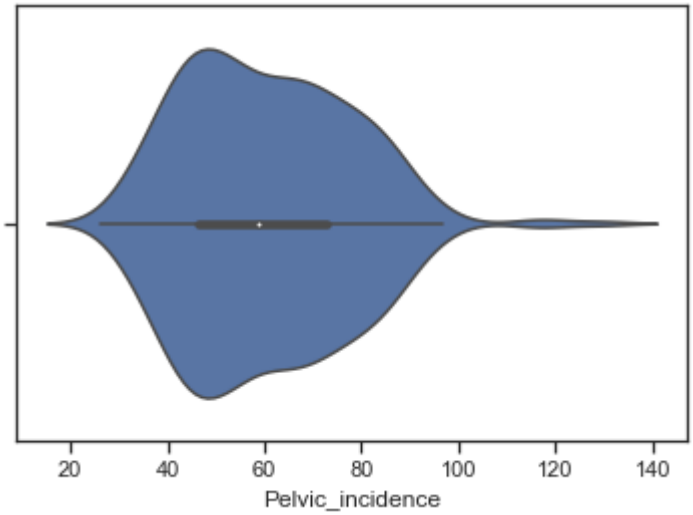
In [16]:
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['Class_att_le'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*1
```

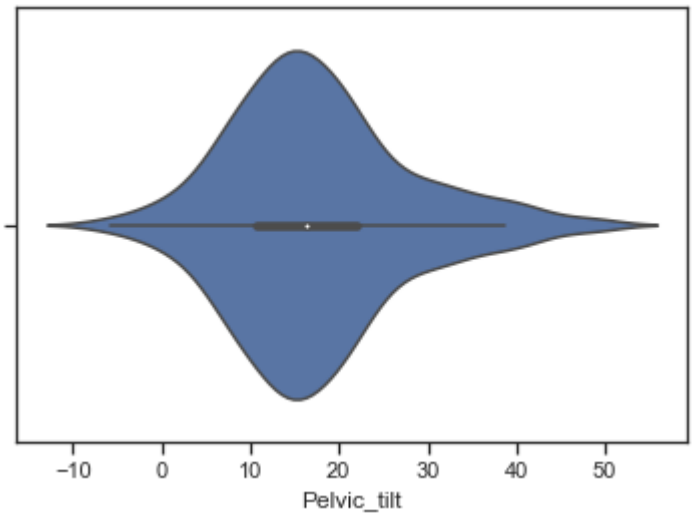
```
Класс 0 составляет 67.64%, а класс 1 составляет 32.36%.
```

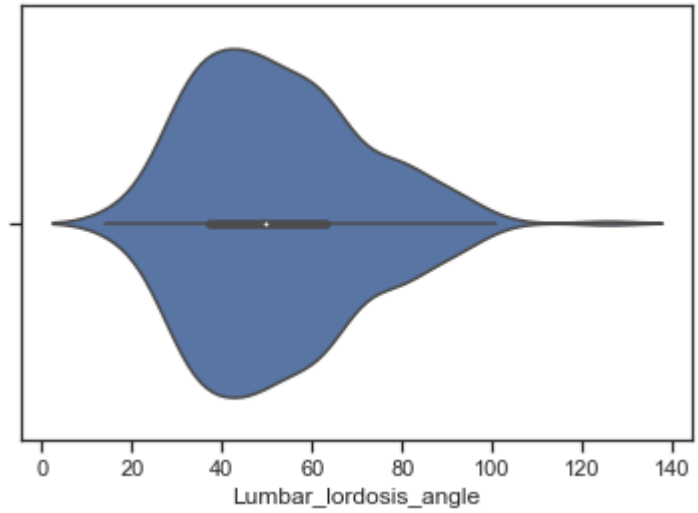
Вывод. Дисбаланс классов присутствует, но является приемлемым.

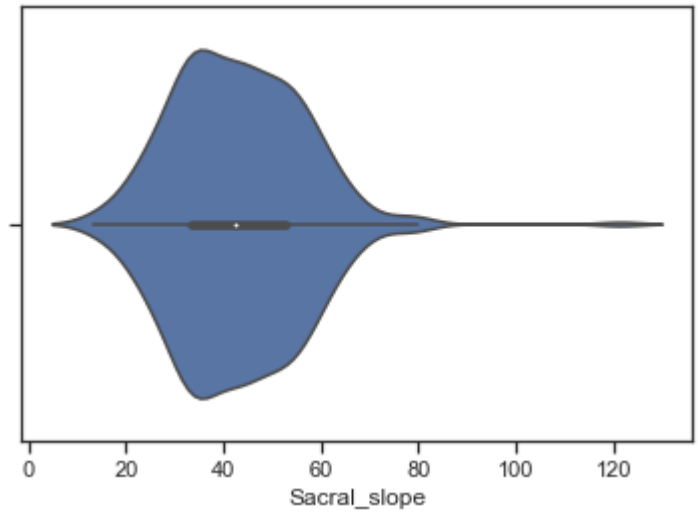
In [17]:

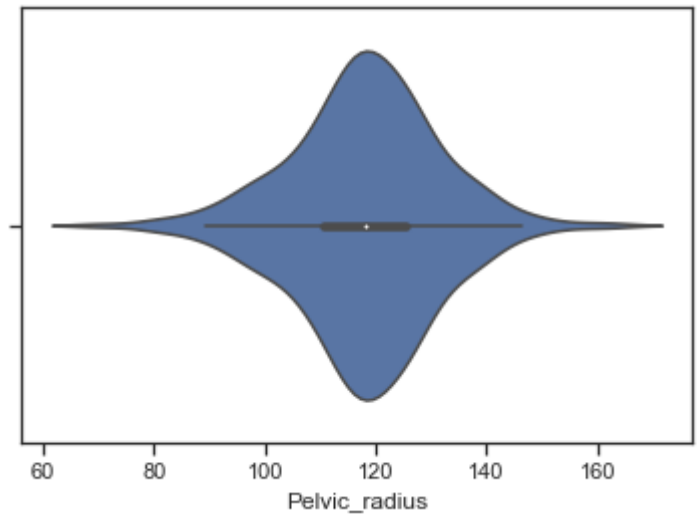
```
# Скрипичные диаграммы для числовых колонок
for col in ['Pelvic_incidence',
                'Pelvic_tilt',
                'Lumbar_lordosis_angle',
                'Sacral_slope',
                'Pelvic_radius',
                'Degree_spondylolisthesis',
                'Pelvic_slope',
                'Direct_tilt',
                'Thoracic_slope',
                'Cervical_tilt',
                'Sacrum_angle',
                'Scoliosis_slope']:
    sns.violinplot(x=data[col])
    plt.show()
```

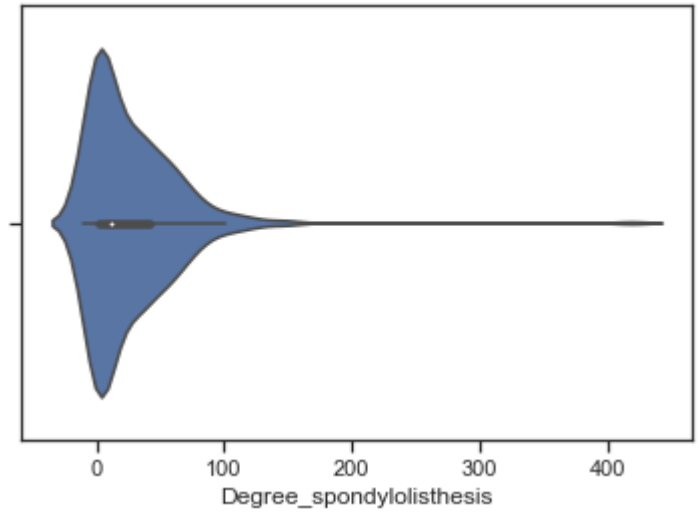


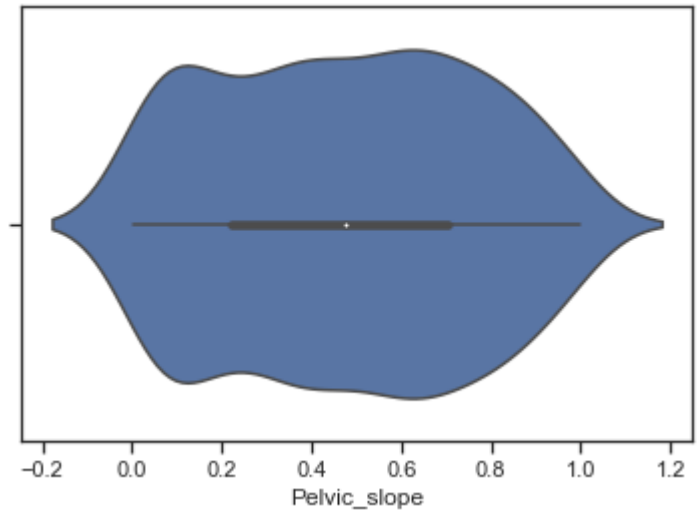


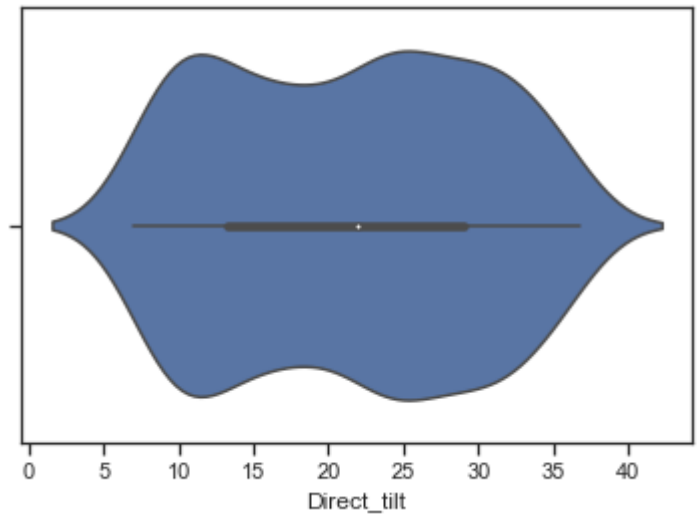

0
20
40
60
80
100
120
140
Lumbar_lordosis_angle

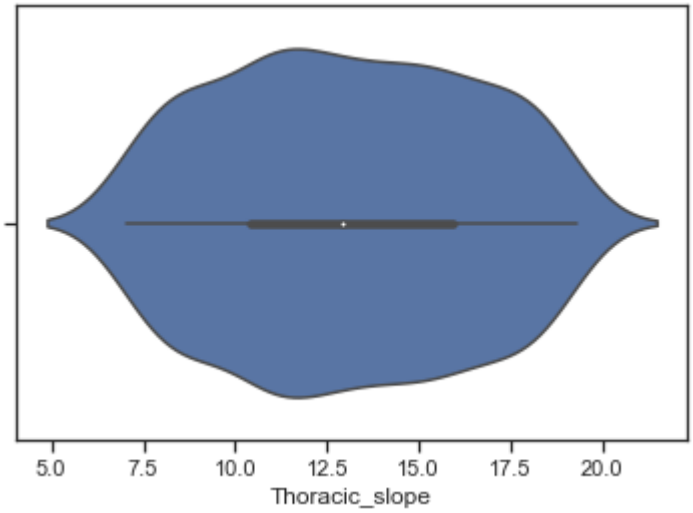

0
20
40
60
80
100
120
Sacral_slope

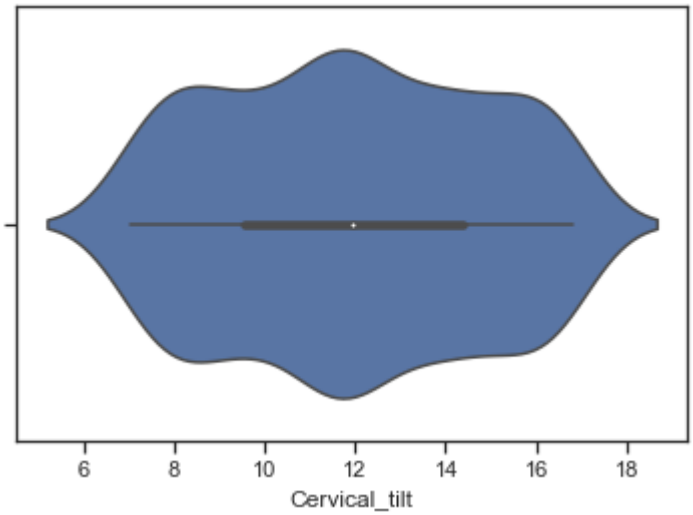

60
80
100
120
140
160
Pelvic_radius

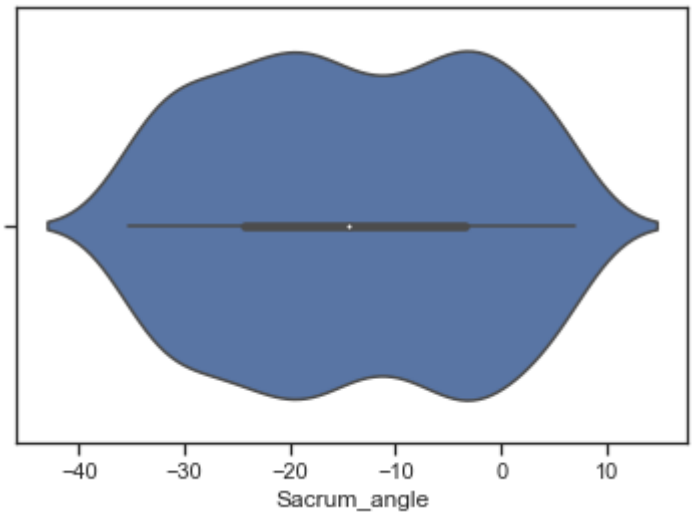

0
100
200
300
400
Degree_spondylolisthesis

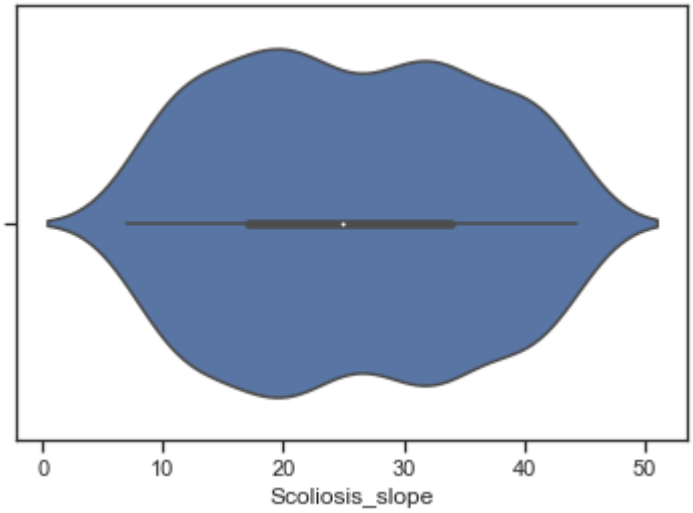

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Pelvic_slope


0
5
10
15
20
25
30
35
40
Direct_tilt


5.0
7.5
10.0
12.5
15.0
17.5
20.0
Thoracic_slope


6
8
10
12
14
16
18
Cervical_tilt


−40
−30
−20
−10
0
10
Sacrum_angle



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 закодированы.Выполним масштабирование данных.

In [18]:
```
# Числовые колонки для масштабирования
scale_cols = ['Pelvic_incidence',
              'Pelvic_tilt',
              'Lumbar_lordosis_angle',
              'Sacral_slope',
              'Pelvic_radius',
              'Degree_spondylolisthesis',
              'Pelvic_slope',
              'Direct_tilt',
              'Thoracic_slope',
              'Cervical_tilt',
              'Sacrum_angle',
              'Scoliosis_slope']
```

In [19]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

In [20]:
```
# 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 [21]:
```
data.head()
```

Out[21]:

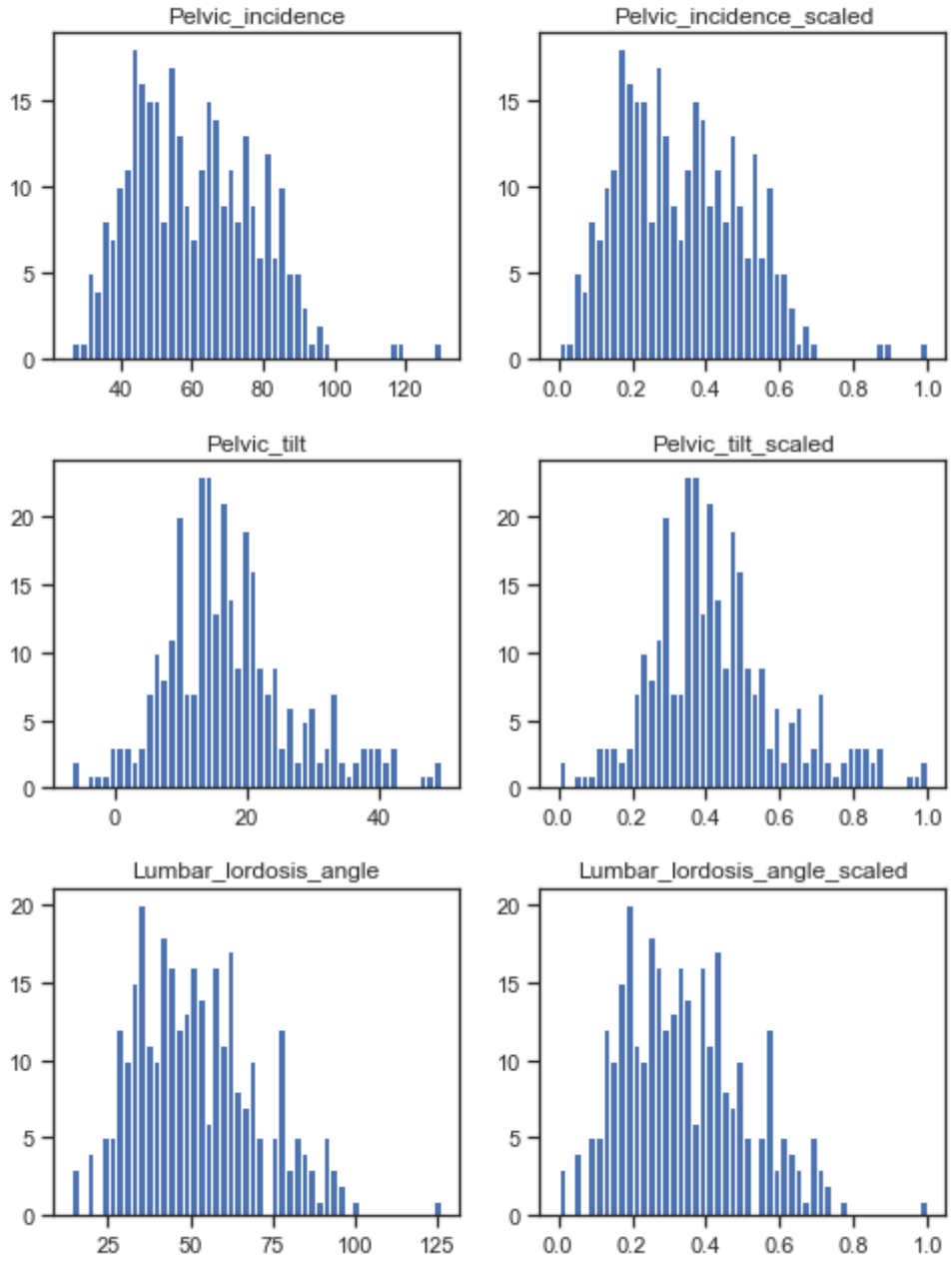
	Pelvic_incidence	Pelvic_tilt	Lumbar_lordosis_angle	Sacral_slope	Pelvic_radius	Degree_
0	39.056951	10.060991	25.015378	28.995960	114.405425	
1	68.832021	22.218482	50.092194	46.613539	105.985135	

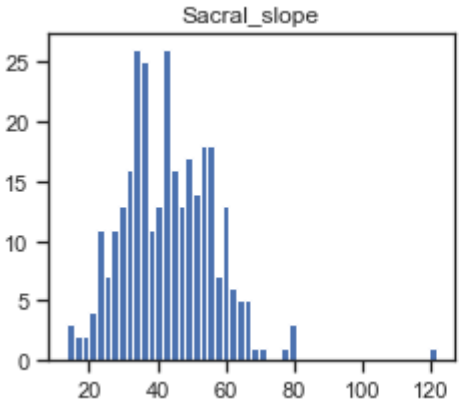
2	69.297008	24.652878	44.311238	44.644130	101.868495
3	49.712859	9.652075	28.317406	40.060784	108.168725
4	40.250200	13.921907	25.124950	26.328293	130.327871

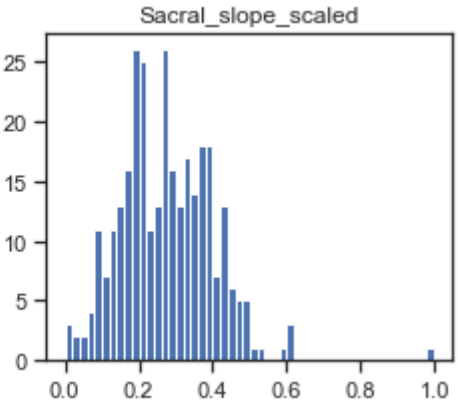
5 rows × 26 columns

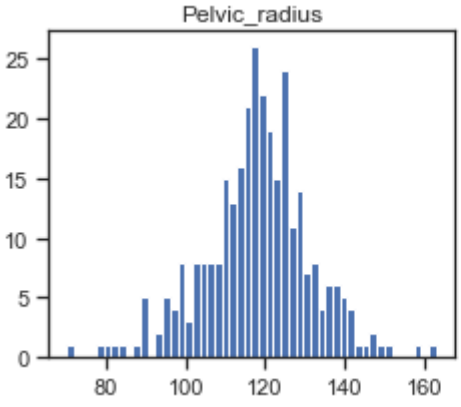
In [22]:

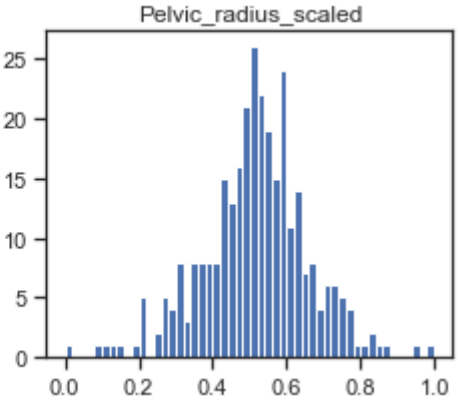
```
# Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    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()
```

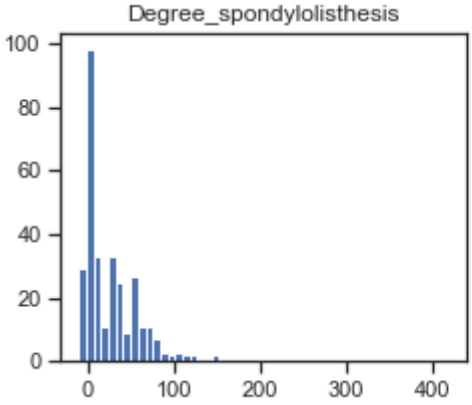


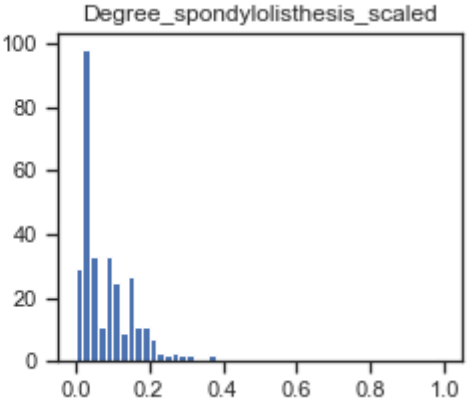

Sacral_slope
0
5
10
15
20
25
20
40
60
80
100
120

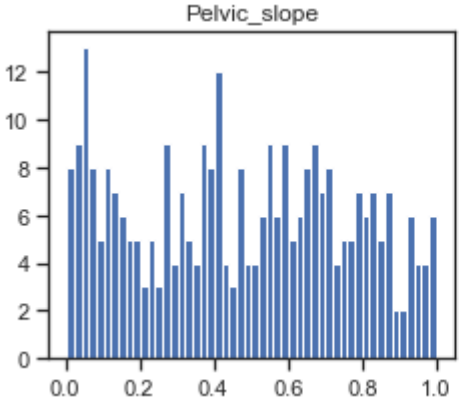

Sacral_slope_scaled
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0

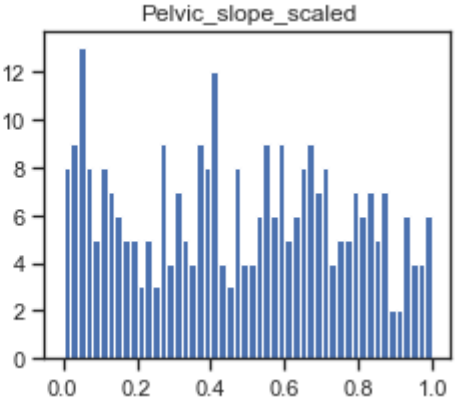

Pelvic_radius
0
5
10
15
20
25
80
100
120
140
160

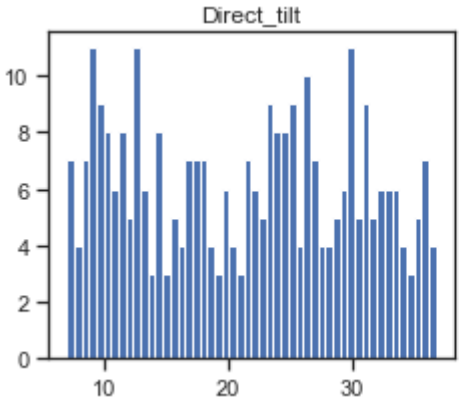

Pelvic_radius_scaled
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0

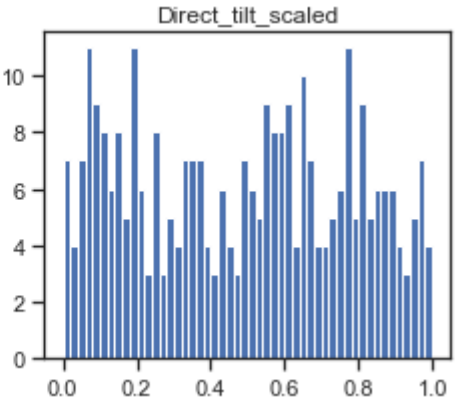

Degree_spondylolisthesis
0
20
40
60
80
100
0
100
200
300
400

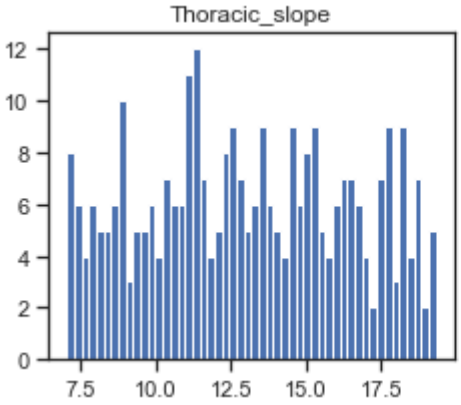

Degree_spondylolisthesis_scaled
0
20
40
60
80
100
0.0
0.2
0.4
0.6
0.8
1.0

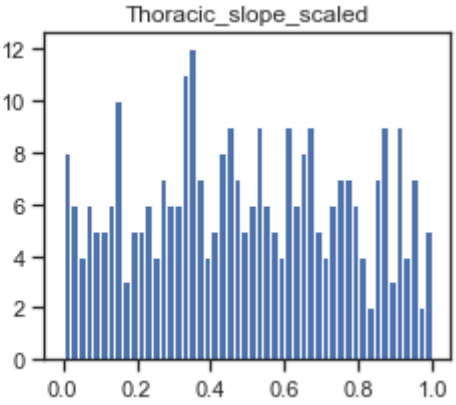

Pelvic_slope
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

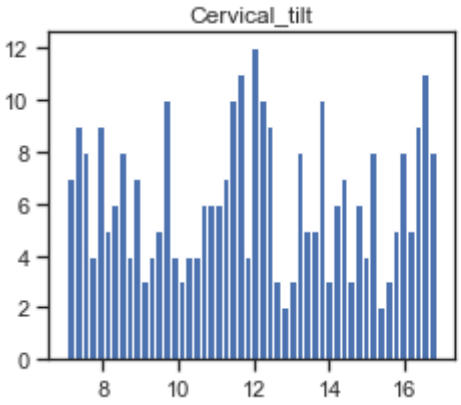

Pelvic_slope_scaled
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

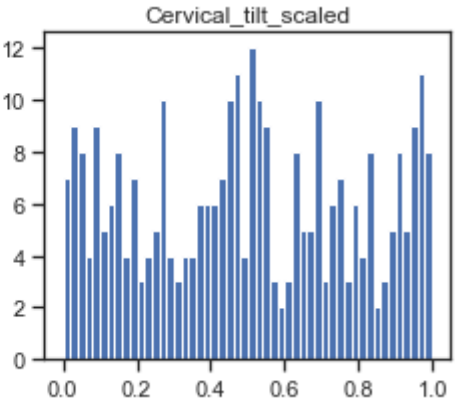

Direct_tilt
10
8
6
4
2
0
10
20
30

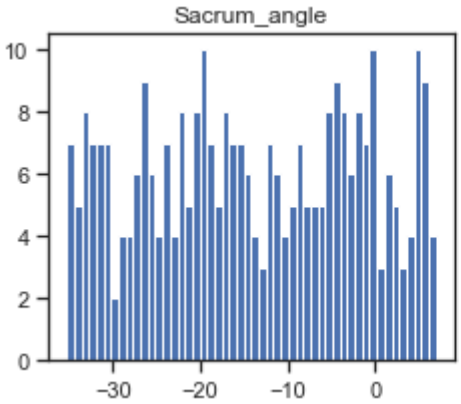

Direct_tilt_scaled
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

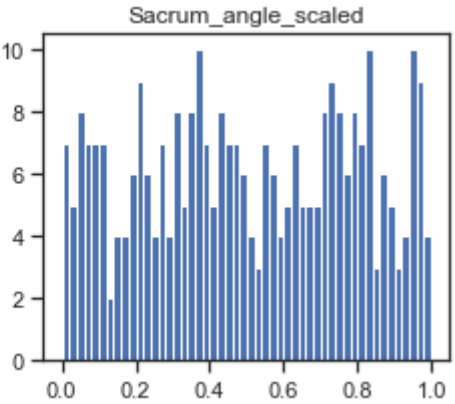

Thoracic_slope
12
10
8
6
4
2
0
7.5
10.0
12.5
15.0
17.5

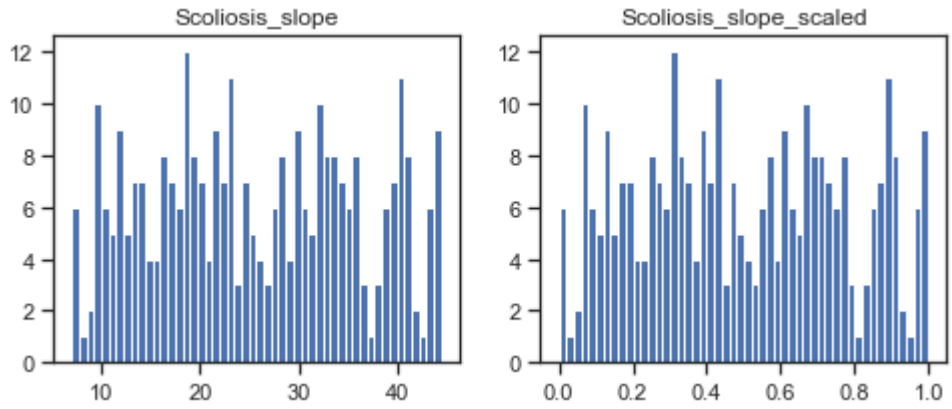

Thoracic_slope_scaled
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0


Cervical_tilt
12
10
8
6
4
2
0
8
10
12
14
16


Cervical_tilt_scaled
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0


Sacrum_angle
10
8
6
4
2
0
−30
−20
−10
0


Sacrum_angle_scaled
10
8
6
4
2
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 [23]:
```
corr_cols_1 = scale_cols + ['Class_att_le']
corr_cols_1
```

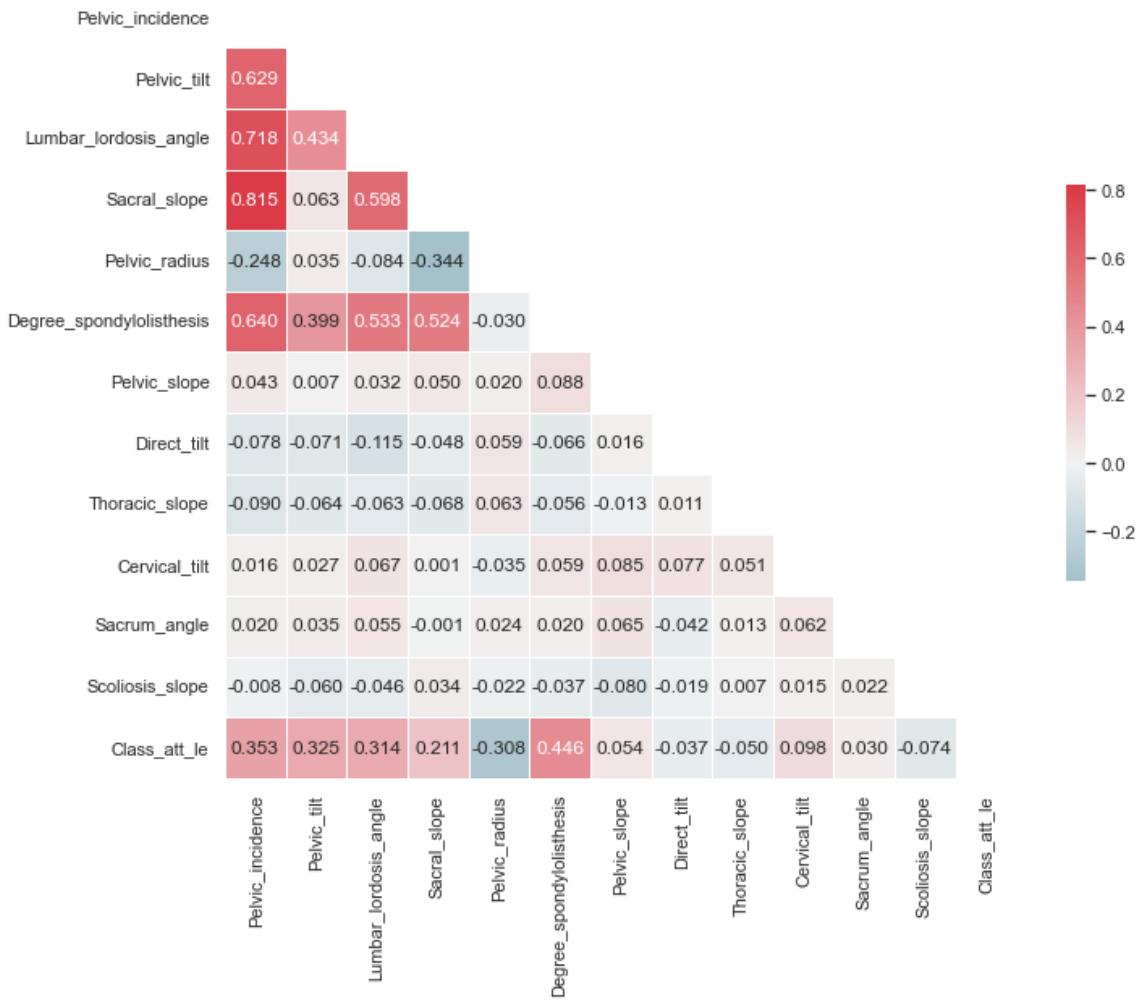
Out[23]:
```
['Pelvic_incidence',
 'Pelvic_tilt',
 'Lumbar_lordosis_angle',
 'Sacral_slope',
 'Pelvic_radius',
 'Degree_spondylolisthesis',
 'Pelvic_slope',
 'Direct_tilt',
 'Thoracic_slope',
 'Cervical_tilt',
 'Sacrum_angle',
 'Scoliosis_slope',
 'Class_att_le']
```

In [24]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Class_att_le']
corr_cols_2
```

Out[24]:
```
['Pelvic_incidence_scaled',
 'Pelvic_tilt_scaled',
 'Lumbar_lordosis_angle_scaled',
 'Sacral_slope_scaled',
 'Pelvic_radius_scaled',
 'Degree_spondylolisthesis_scaled',
 'Pelvic_slope_scaled',
 'Direct_tilt_scaled',
 'Thoracic_slope_scaled',
 'Cervical_tilt_scaled',
 'Sacrum_angle_scaled',
 'Scoliosis_slope_scaled',
 'Class_att_le']
```

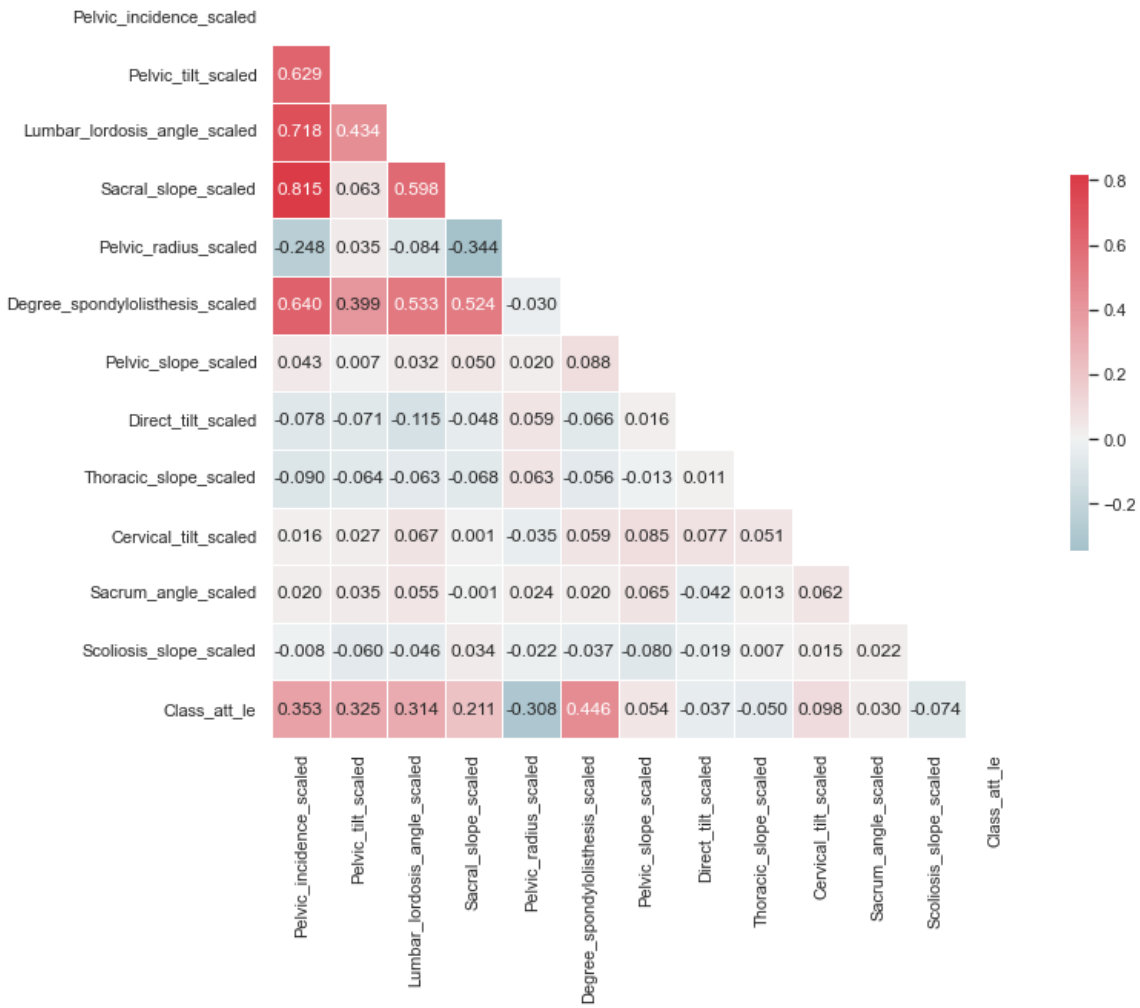
In [25]:
```
sns.set(style="white")
corr = data[corr_cols_1].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
```

```
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



In [26]:

```
sns.set(style="white")
corr = data[corr_cols_2].corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
g=sns.heatmap(corr, mask=mask, cmap=cmap, center=0, annot=True, fmt='.3f
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```



На основе корреляционной матрицы можно сделать следующие выводы:

- Корреляционные матрицы для исходных и масштабированных данных совпадают.
- Целевой признак классификации "Class_att_le" наиболее сильно коррелирует со следующими признаками:
    1. "Degree_spondylolisthesis" (0.446);
    2. "Pelvic_incidence" (0.353);
    3. "Pelvic_tilt" (0.325)
    4. "Lumbar_lordosis_angle" (0.314) Эти признаки следует оставить в модели классификации.
- Признаки "Pelvic_incidence" и "Sacral_slope" имеют большую корреляцию, поэтому оба признака не следует включать в модель. Будем использовать признак "Pelvic_incidence".
- 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.
- F1- 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.

In [27]:
```
# 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='darkturquoise',
             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 [28]:
```
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']=
        # 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=ascend
        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, as
        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

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 [29]:
```
# Признаки для задачи классификации
class_cols = ['Pelvic_incidence',
              'Pelvic_tilt',
              'Lumbar_lordosis_angle',
              'Degree_spondylolisthesis',
              ]
```

In [30]:
```
X = data[class_cols]
Y = data['Class_att_le']
X.shape
```

Out[30]:
```
(309, 4)
```

In [31]:
```
# С использованием метода train_test_split разделим выборку на обучающую
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1,
```

In [32]:
```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[32]:
```
((278, 4), (31, 4), (278,), (31,))
```

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.

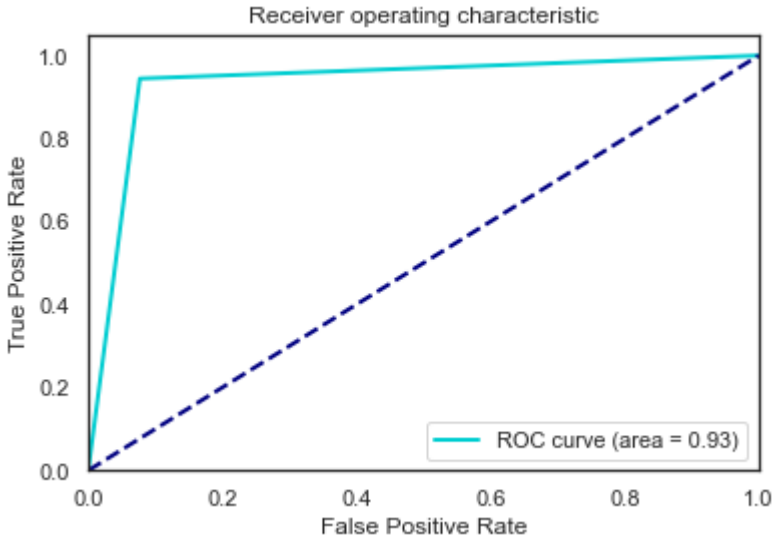
In [33]:
```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

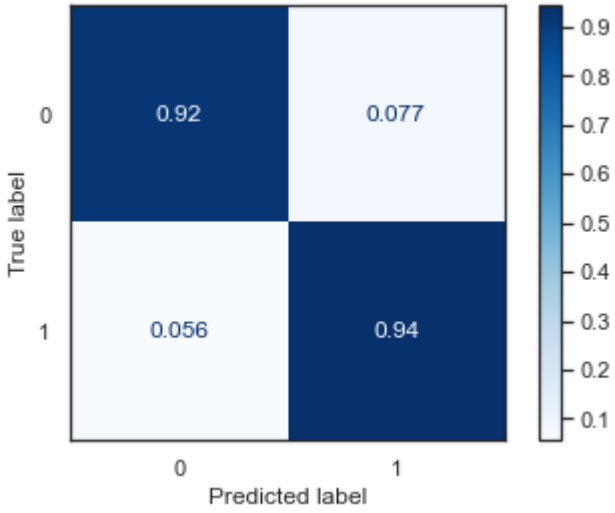
In [34]:
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

In [35]:
```
def train_model(model_name, model, MetricLogger):
    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)

    MetricLogger.add('precision', model_name, precision)
    MetricLogger.add('recall', model_name, recall)
    MetricLogger.add('f1', model_name, f1)
    MetricLogger.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 [36]:
```
for model_name, model in clas_models.items():
    train_model(model_name, model, clasMetricLogger)
```

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

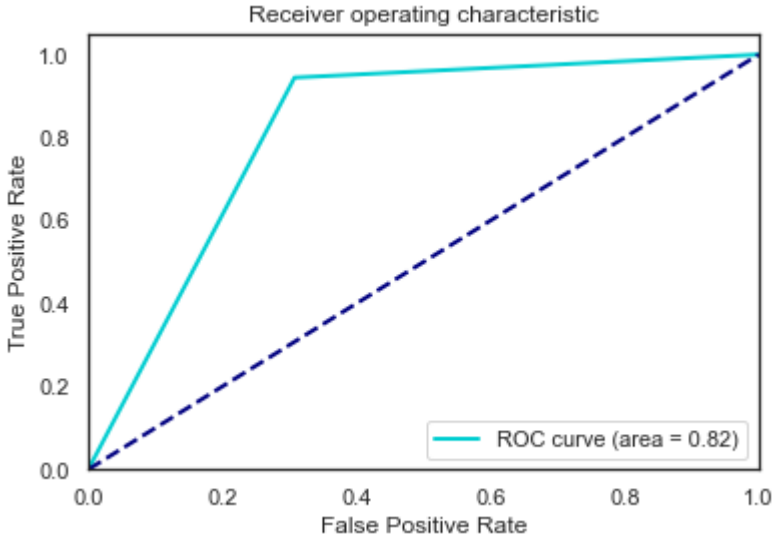


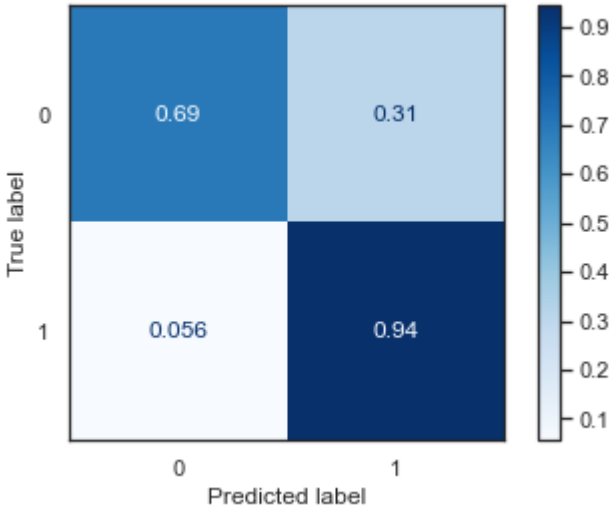


****

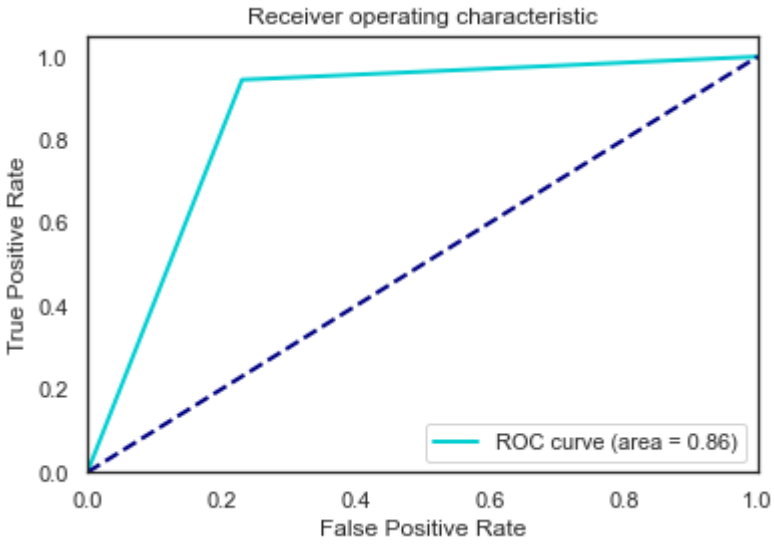
KNeighborsClassifier()

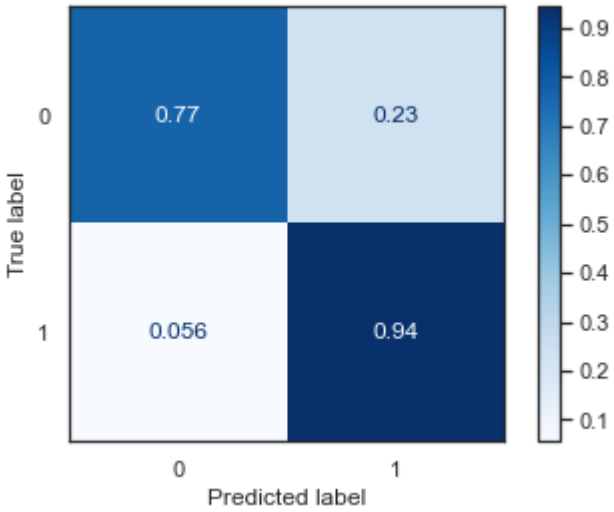
****



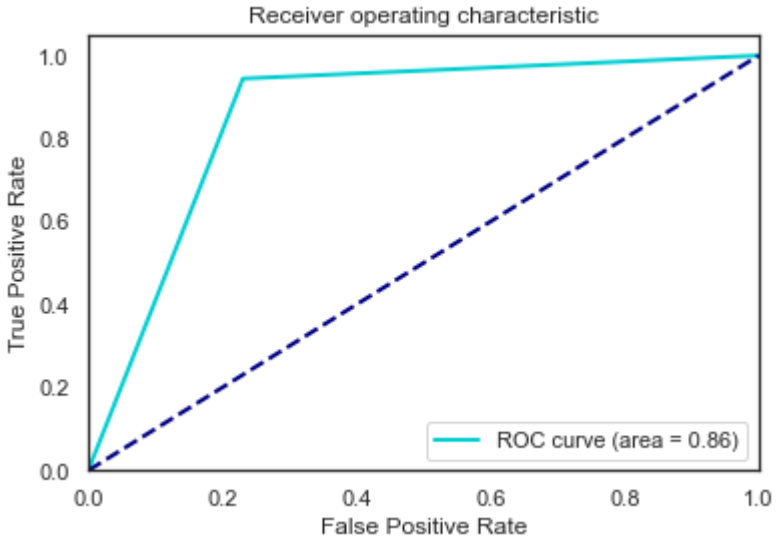


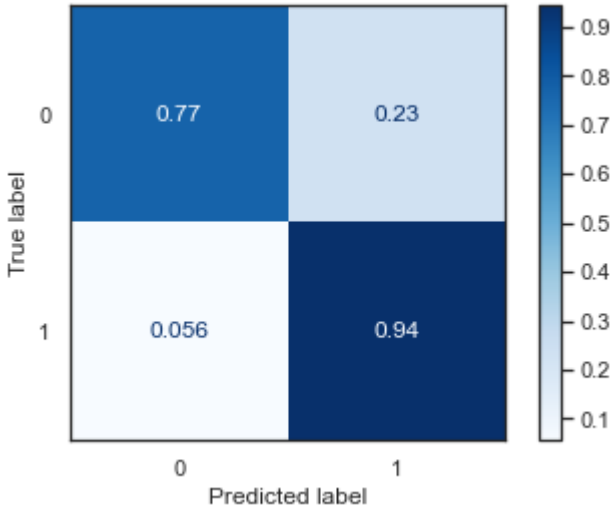
****
SVC()
****





****
DecisionTreeClassifier()
****

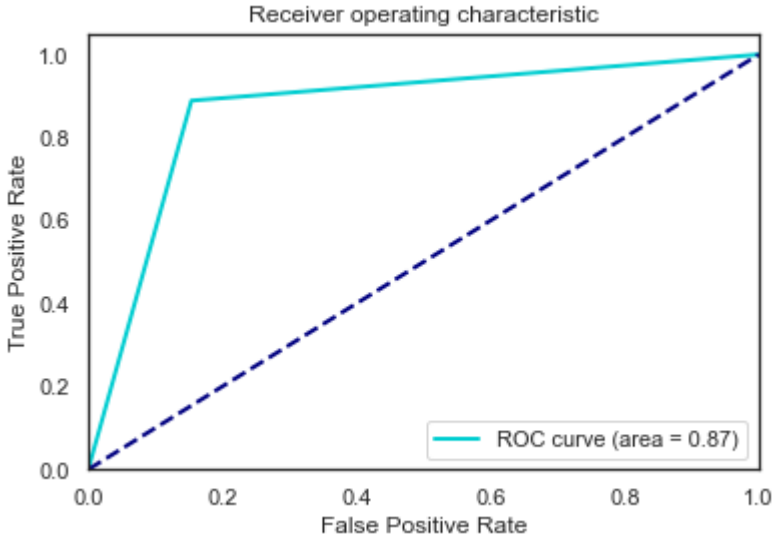


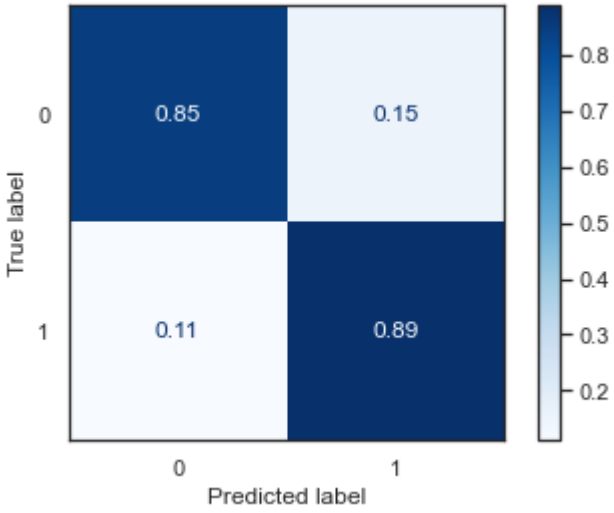


****

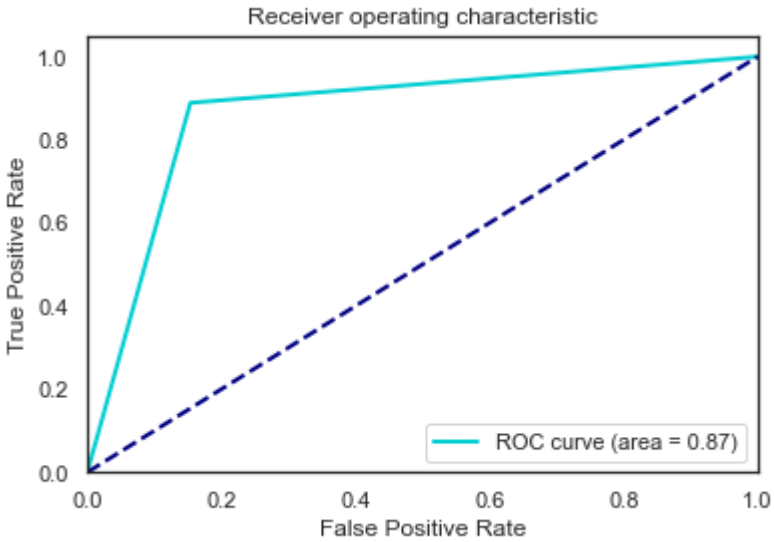
RandomForestClassifier()

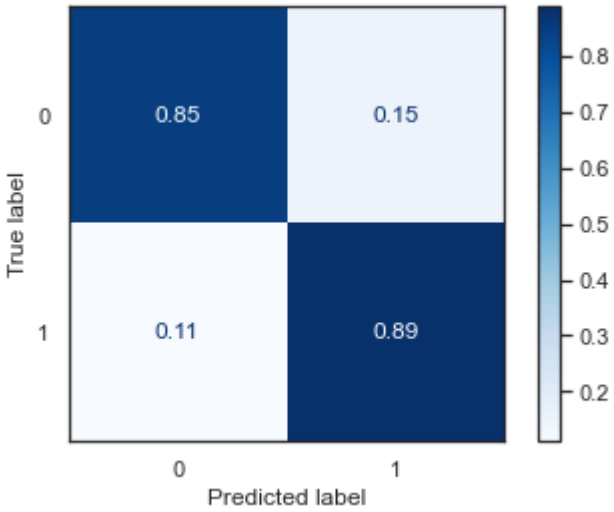
****






```
**************************************************
GradientBoostingClassifier()
**************************************************
```





9) Подбор гиперпараметров для выбранных моделей.

Метод ближайших соседей

In [37]:

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

Out[37]:
```
[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
        35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
        52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
        69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
        86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}]
```

In [38]:
```
gs_KNN = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, sc
gs_KNN.fit(X_train, Y_train)
```

Out[38]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}],
             scoring='roc_auc')
```

In [39]:
```
# Лучшая модель
gs_KNN.best_estimator_
```

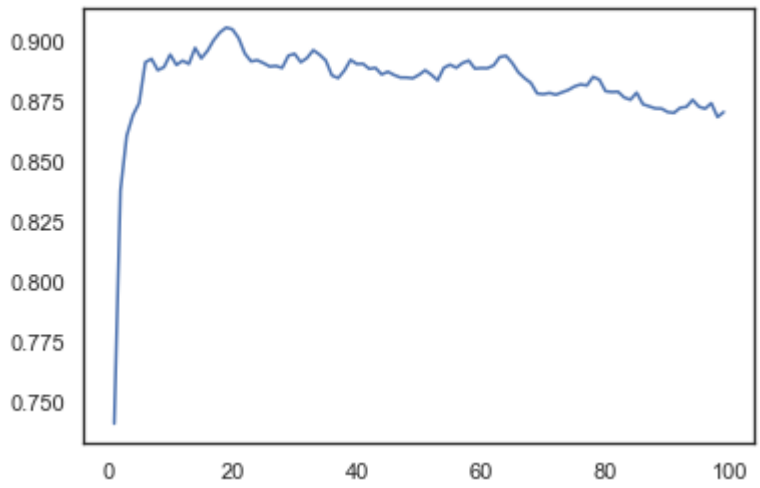
Out[39]:
```
KNeighborsClassifier(n_neighbors=19)
```

In [40]:
```
# Лучшее значение параметров
gs_KNN.best_params_
```

Out[40]:
```
{'n_neighbors': 19}
```

In [41]:
```
plt.plot(n_range, gs_KNN.cv_results_['mean_test_score'])
```

Out[41]:
```
[<matplotlib.lines.Line2D at 0x7ff5a572f310>]
```



Логистическая регрессия

In [42]:
```
grid={"C":np.logspace(-3,3,3)}
gs_LogR = GridSearchCV(LogisticRegression(), grid, cv=5, scoring='roc_au
gs_LogR.fit(X_train, Y_train)
```

Out[42]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': array([1.e-03, 1.e+00, 1.e+03])},
             scoring='roc_auc')
```

In [43]:
```
# Лучшая модель
gs_LogR.best_estimator_
```

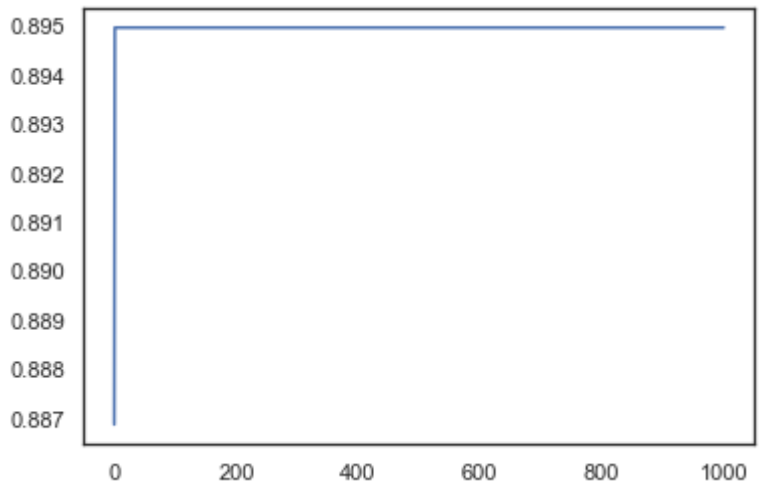
Out[43]:
```
LogisticRegression()
```

In [44]:
```
# Лучшее значение параметров
gs_LogR.best_params_
```

Out[44]:
```
{'C': 1.0}
```

In [45]:
```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,3,3), gs_LogR.cv_results_['mean_test_score'])
```

Out[45]:
```
[<matplotlib.lines.Line2D at 0x7ff5a578f5e0>]
```



Машина опорных векторов

In [46]:
```
SVC_grid={"C":np.logspace(-3,5,12)}
gs_SVC = GridSearchCV(SVC(), SVC_grid, cv=5, scoring='roc_auc')
gs_SVC.fit(X_train, Y_train)
```

Out[46]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 5.33669923e-03, 2.84
803587e-02, 1.51991108e-01,
       8.11130831e-01, 4.32876128e+00, 2.31012970e+01, 1.23284674e+02,
       6.57933225e+02, 3.51119173e+03, 1.87381742e+04, 1.00000000e+05])},
             scoring='roc_auc')
```

In [47]:
```
# Лучшая модель
gs_SVC.best_estimator_
```

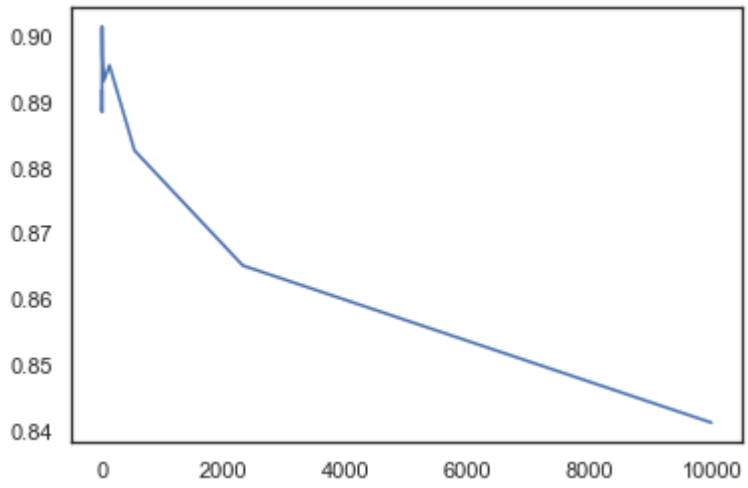
Out[47]:
```
SVC(C=4.328761281083062)
```

In [48]:
```
# Лучшее значение параметров
gs_SVC.best_params_
```

Out[48]:
```
{'C': 4.328761281083062}
```

In [49]:
```
# Изменение качества на тестовой выборке
plt.plot(np.logspace(-3,4,12), gs_SVC.cv_results_['mean_test_score'])
```

Out[49]:
```
[<matplotlib.lines.Line2D at 0x7ff5a580cf70>]
```



Решающее дерево

In [50]:
```
tree_params={"max_depth":range(1,20), "max_features":range(1,5)}
gs_Tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=5, scor
gs_Tree.fit(X_train, Y_train)
```

Out[50]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': range(1, 20),
                         'max_features': range(1, 5)},
             scoring='precision')
```

In [51]:
```
# Лучшая модель
gs_Tree.best_estimator_
```

Out[51]:
```
DecisionTreeClassifier(max_depth=1, max_features=4)
```

In [52]:
```
# Лучшее значение параметров
gs_Tree.best_params_
```

Out[52]:
```
{'max_depth': 1, 'max_features': 4}
```

Случайный лес

In [53]:
```
RF_params={"max_leaf_nodes":range(2,12), "max_samples":range(2,22)}
gs_RF = GridSearchCV(RandomForestClassifier(), RF_params, cv=5, scoring=
gs_RF.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
```

Out[53]:
```
             param_grid={'max_leaf_nodes': range(2, 12),
                         'max_samples': range(2, 22)},
             scoring='roc_auc')
```

In [54]:
```
# Лучшая модель
gs_RF.best_estimator_
```

Out[54]:
```
RandomForestClassifier(max_leaf_nodes=7, max_samples=21)
```

In [55]:
```
# Лучшее значение параметров
gs_RF.best_params_
```

Out[55]:
```
{'max_leaf_nodes': 7, 'max_samples': 21}
```

Градиентный бустинг

In [56]:
```
GB_params={"max_features":range(1,4), "max_leaf_nodes":range(2,22)}
gs_GB = GridSearchCV(GradientBoostingClassifier(), GB_params, cv=5, scor
gs_GB.fit(X_train, Y_train)
```

Out[56]:
```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'max_features': range(1, 4),
                         'max_leaf_nodes': range(2, 22)},
             scoring='f1')
```

In [57]:
```
# Лучшая модель
gs_GB.best_estimator_
```

Out[57]:
```
GradientBoostingClassifier(max_features=1, max_leaf_nodes=4)
```

In [58]:
```
# Лучшее значение параметров
gs_GB.best_params_
```

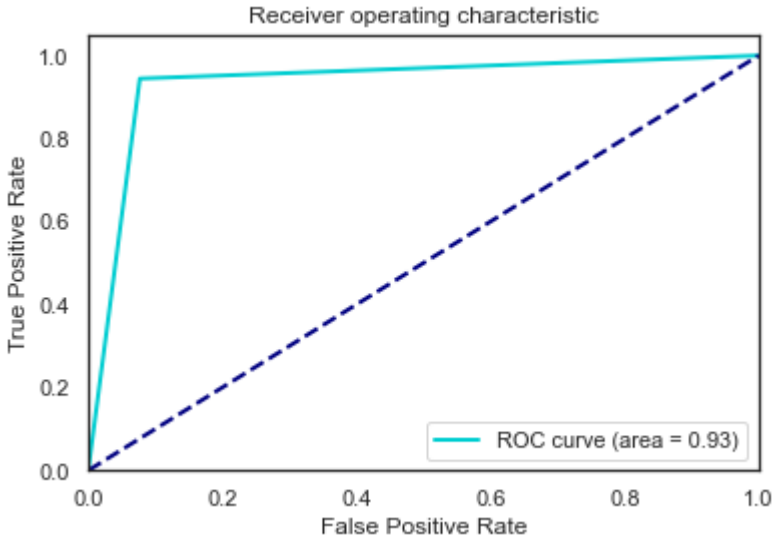
Out[58]:
```
{'max_features': 1, 'max_leaf_nodes': 4}
```

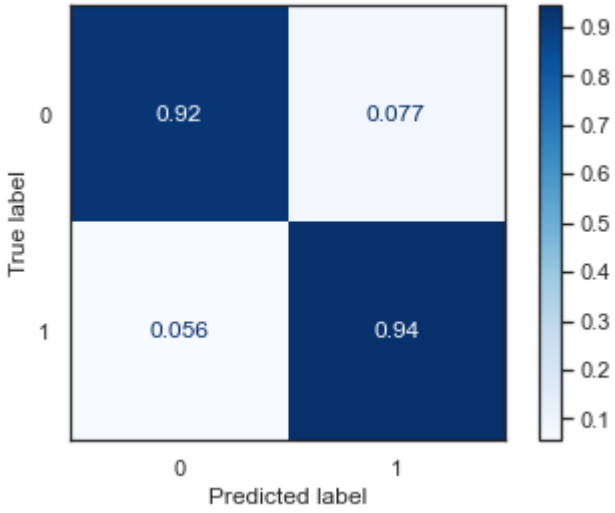
10) Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

In [59]:
```
models_grid = { 'LogR_new':gs_LogR.best_estimator_,
                'KNN_new':gs_KNN.best_estimator_,
                 'SVC_new':gs_SVC.best_estimator_,
               'Tree_new':gs_Tree.best_estimator_,
                'RF_new':gs_RF.best_estimator_,
               'GB_new':gs_GB.best_estimator_
               }
```

In [60]:
```
for model_name, model in models_grid.items():
    train_model(model_name, model, clasMetricLogger)
```

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

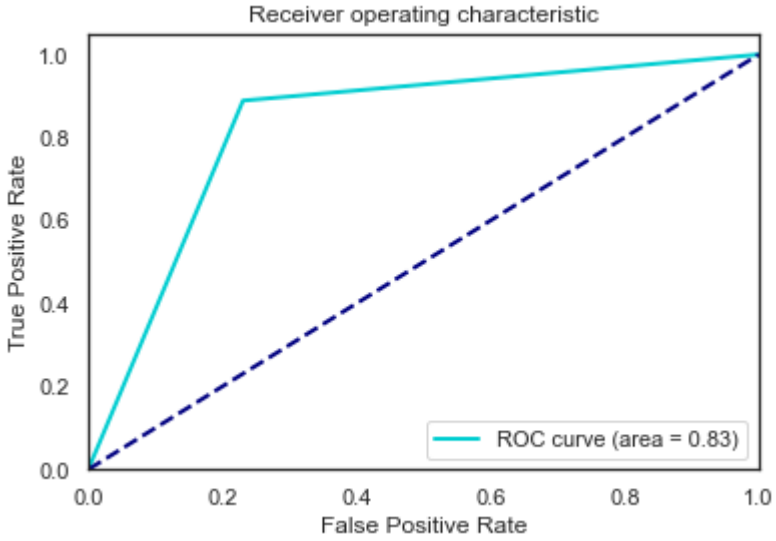


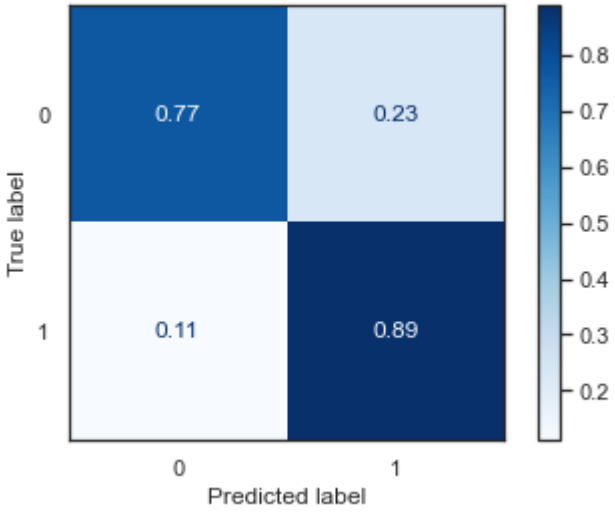


****

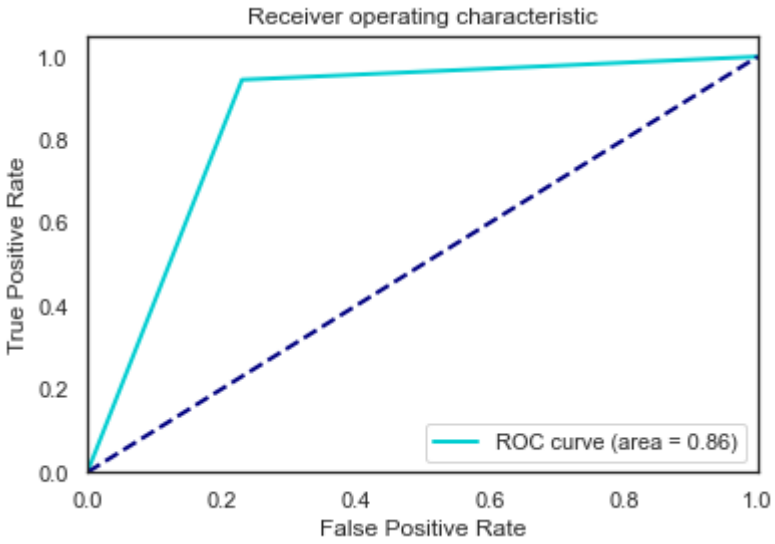
KNeighborsClassifier(n_neighbors=19)

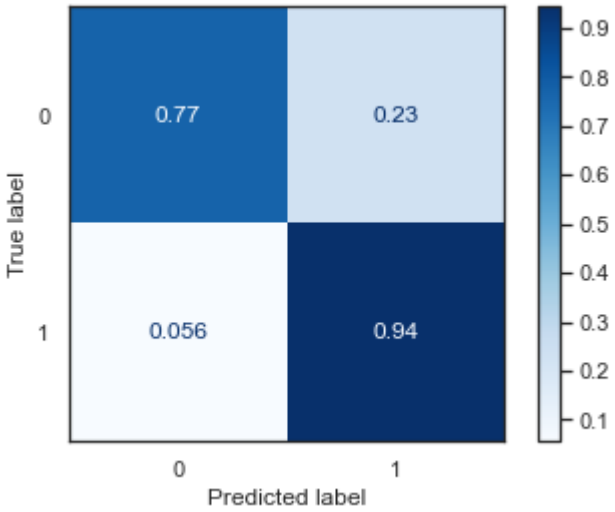
****



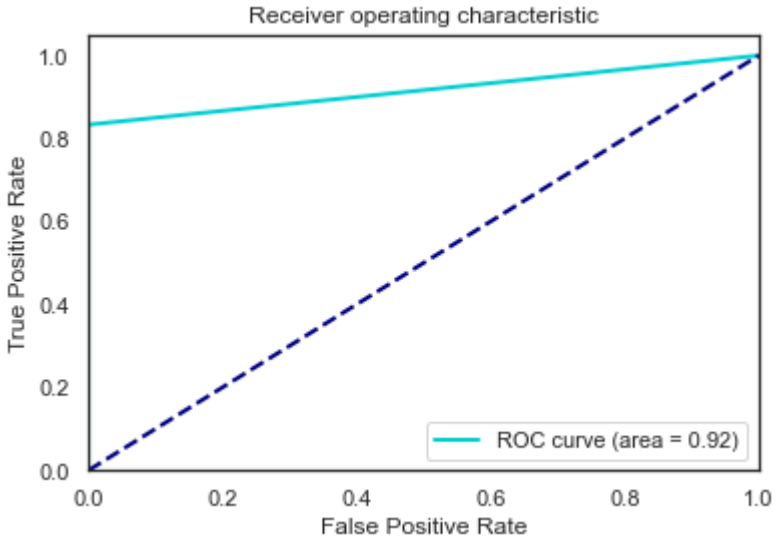


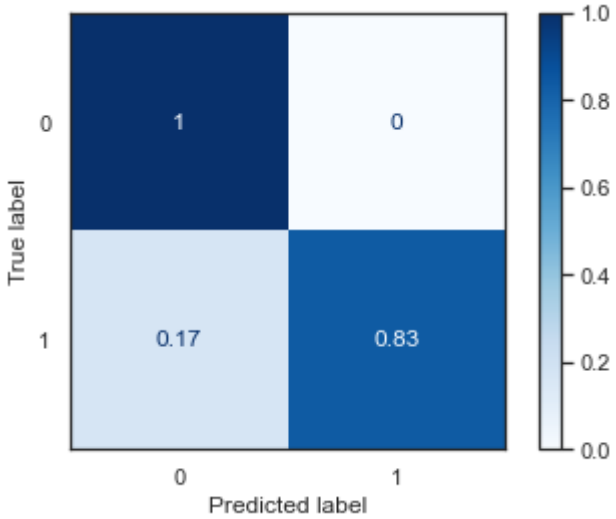
****
SVC(C=4.328761281083062)
****





****
DecisionTreeClassifier(max_depth=1, max_features=4)
****

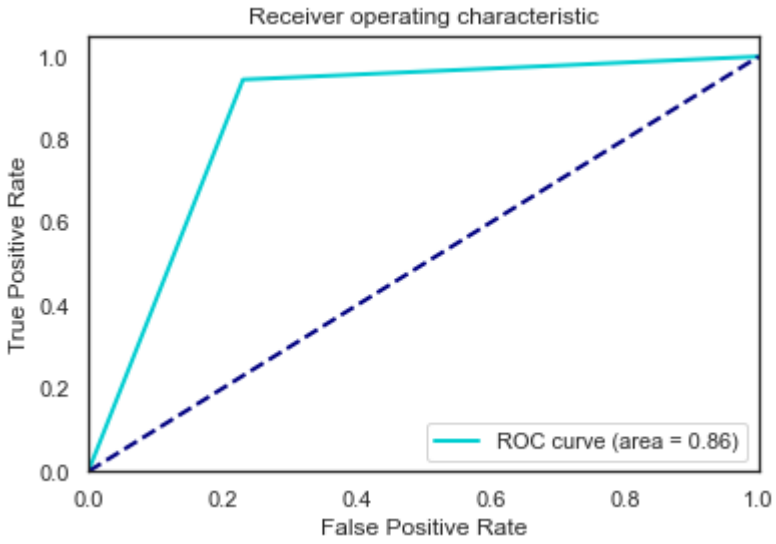


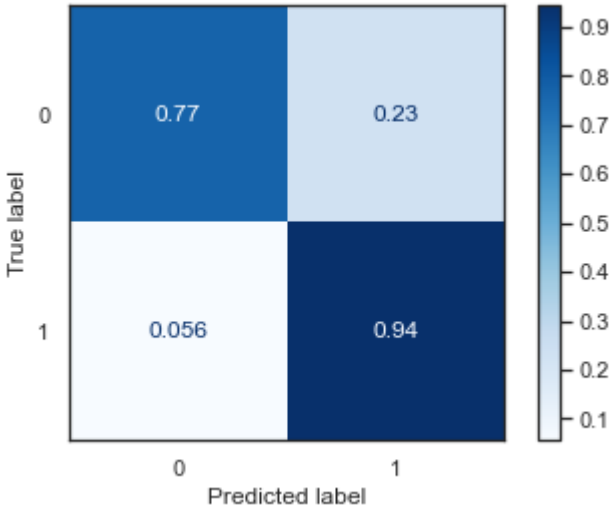


****

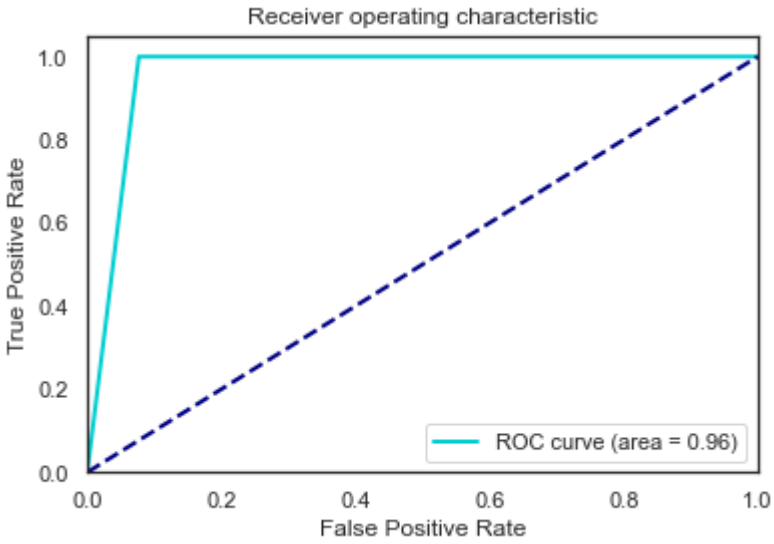
RandomForestClassifier(max_leaf_nodes=7, max_samples=21)

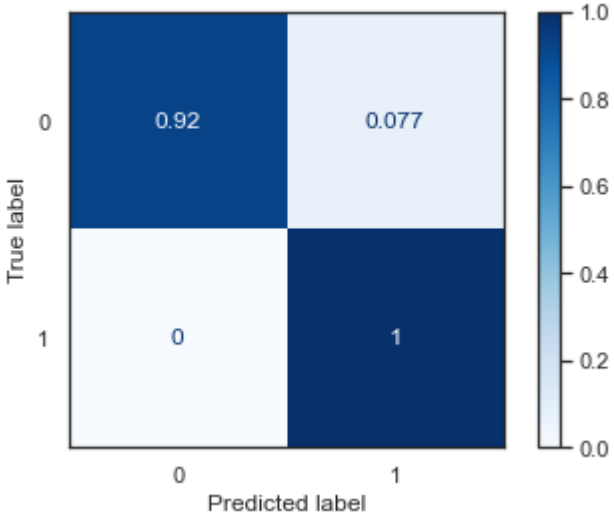
****






```
**************************************************
GradientBoostingClassifier(max_features=1, max_leaf_nodes=4)
**************************************************
```





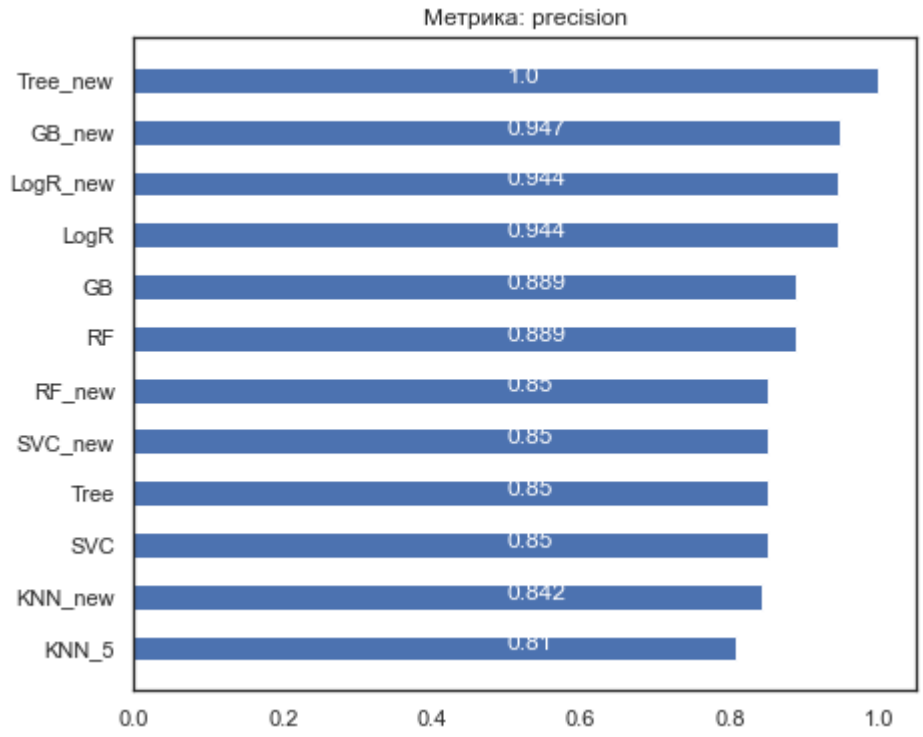
11) Формирование выводов о качестве построенных моделей на основе выбранных метрик.

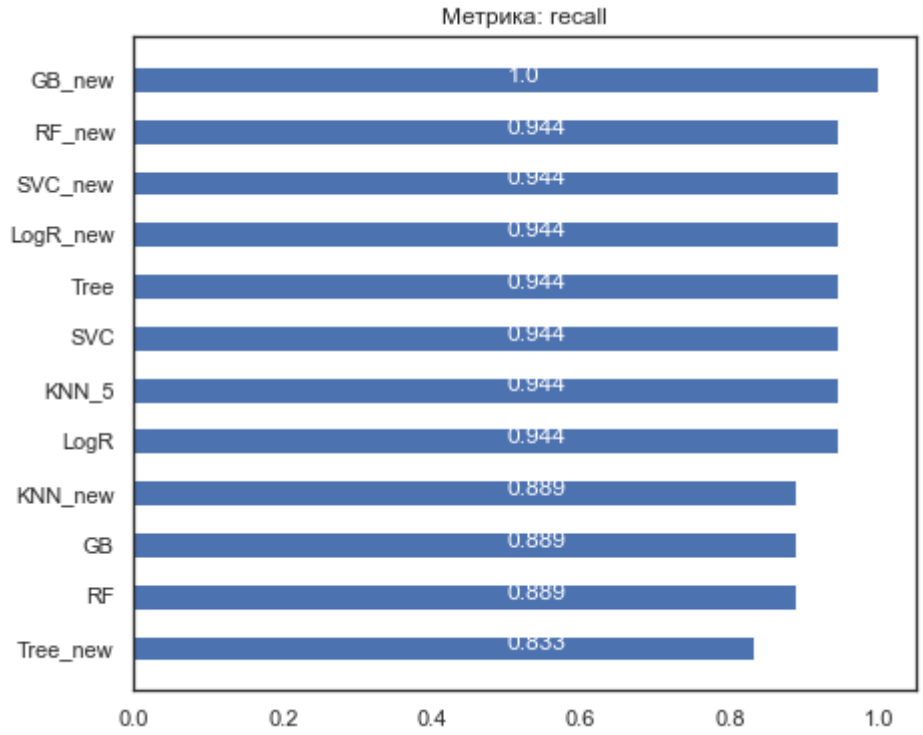
In [61]:

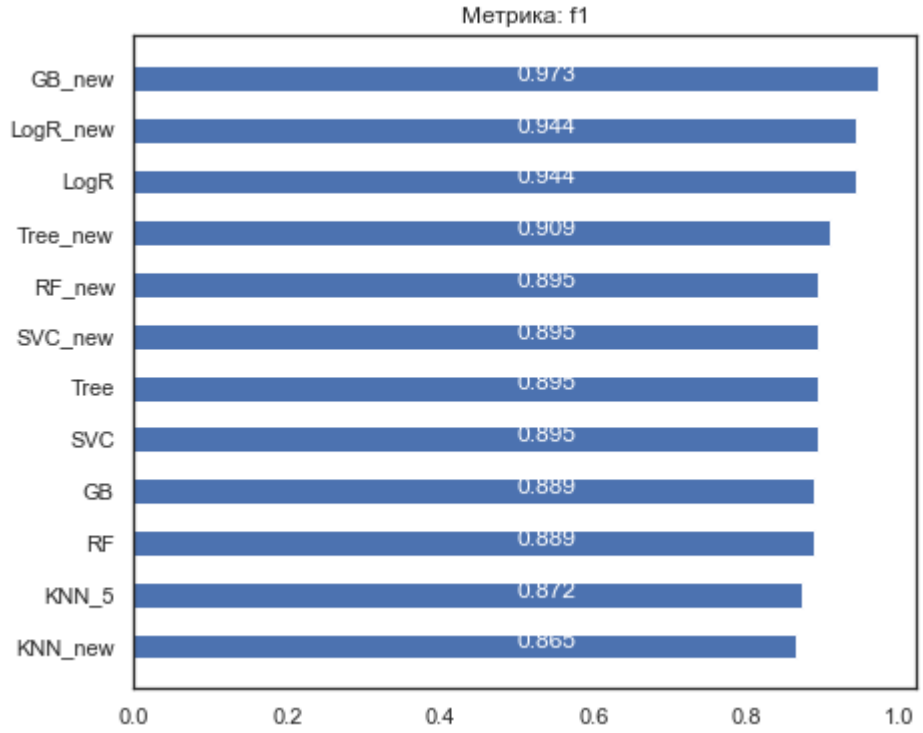
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

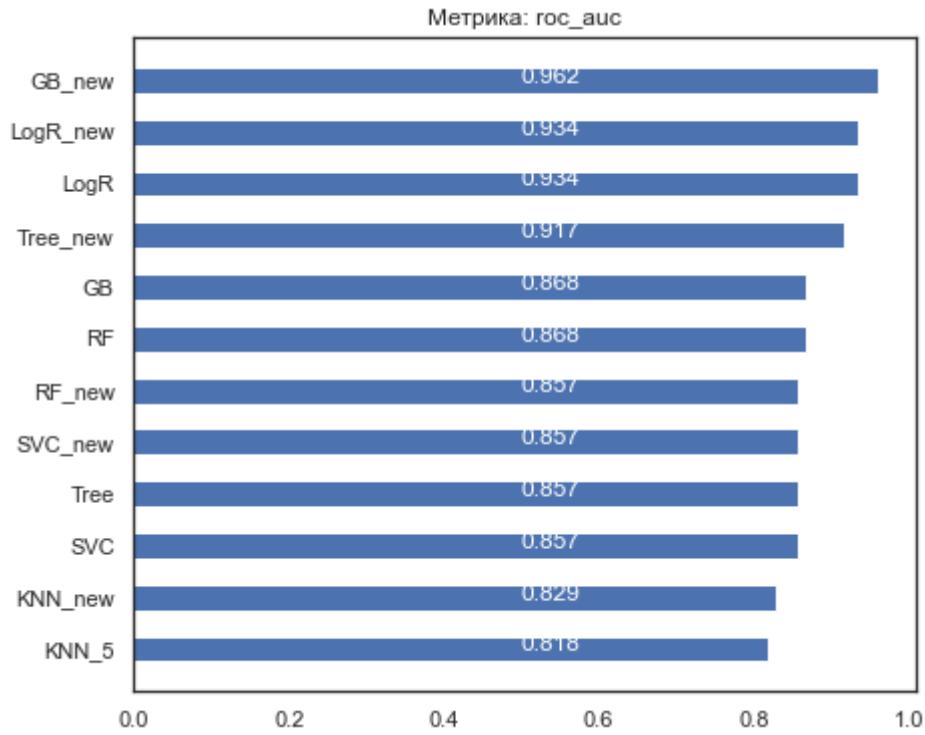
Out[61]:
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

In [62]:
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```









Вывод: на основании трех метрик из четырех, лучшими моделями оказались случайный лес и логистическая регрессия.

In [ ]:

In [ ]: