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**)

1. 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]:

Out[4]:

In [5]:

Out[5]:

In [6]:

Out[6]:

In [7]:

Out[7]:

data**.**shape

(309, 13)

data**.**columns

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')

data**.**dtypes

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

data**.**isnull()**.**sum()

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 |

In [10]:

Out[10]:

In [11]:

[309 rows x 2 columns]

Abnormal 209

data['Class\_att']**.**value\_counts()

Normal 100

Name: Class\_att, dtype: int64

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 |

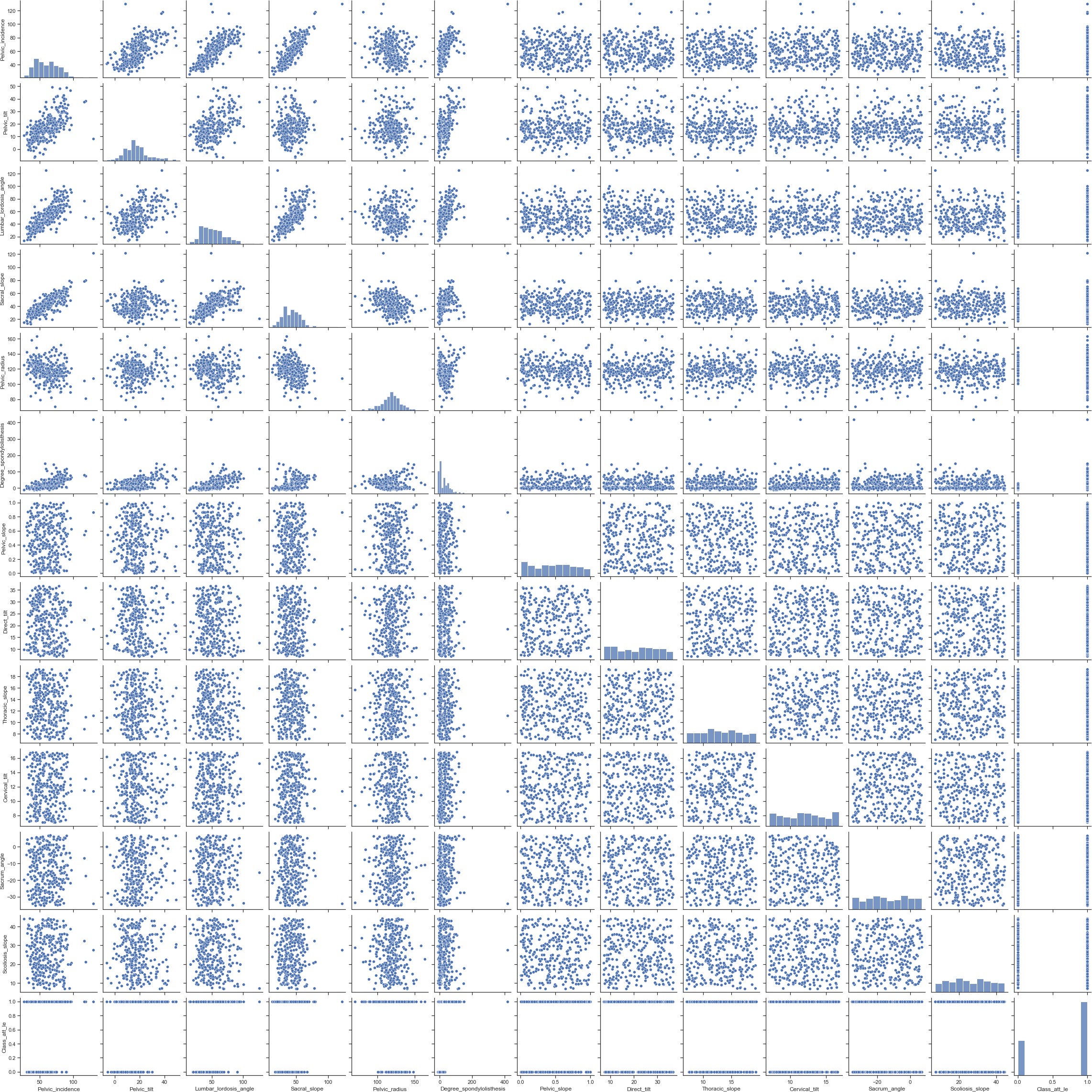
## Набор данных не содержит пропусков, категориальные пизнаки закодированы.

In [12]:

Out[12]:

sns**.**pairplot(data)

<seaborn.axisgrid.PairGrid at 0x7ff5c5940100>



In [13]:

Out[13]:

sns**.**pairplot(data, hue**=**"Class\_att\_le")

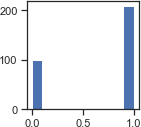
<seaborn.axisgrid.PairGrid at 0x7ff5b39ba9d0>



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]:

Out[15]:

In [16]:

data['Class\_att\_le']**.**value\_counts()

1 209

0 100

Name: Class\_att\_le, dtype: int64

*# 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

In [17]:

*# Скрипичные диаграммы для числовых колонок*

**for** col **in** ['Pelvic\_incidence',

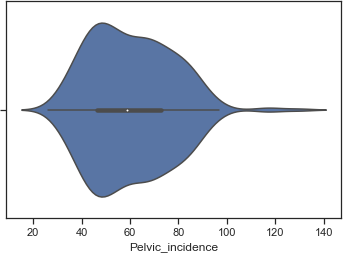
'Pelvic\_tilt', 'Lumbar\_lordosis\_angle', 'Sacral\_slope', 'Pelvic\_radius', 'Degree\_spondylolisthesis', 'Pelvic\_slope', 'Direct\_tilt', 'Thoracic\_slope', 'Cervical\_tilt', 'Sacrum\_angle', 'Scoliosis\_slope']:

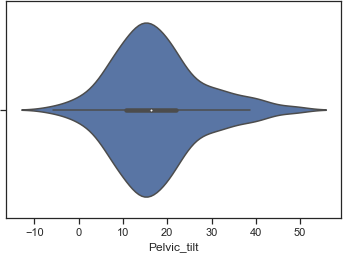
sns**.**violinplot(x**=**data[col])

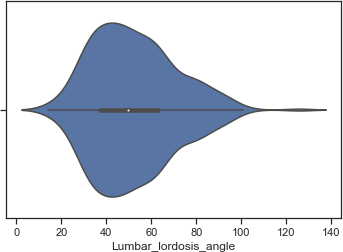
plt**.**show()

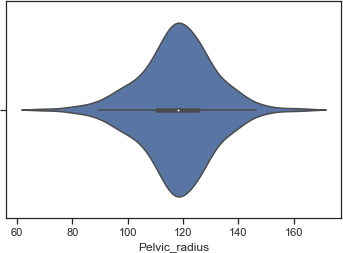
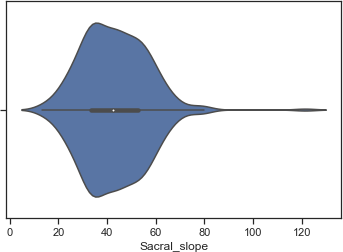
Класс 0 составляет 67.64%, а класс 1 составляет 32.36%.

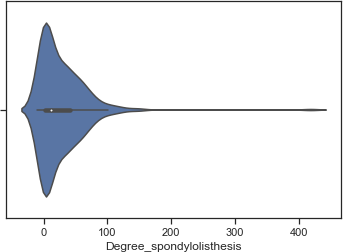
Вывод. Дисбаланс классов присутствует, но является приемлемым.

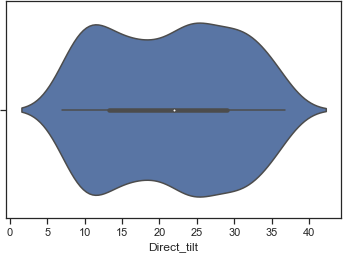
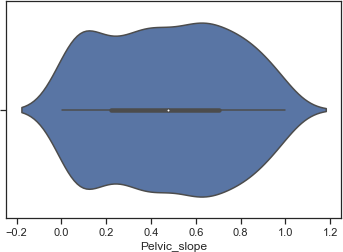


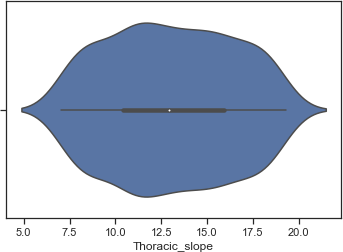


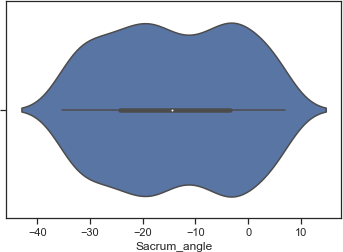
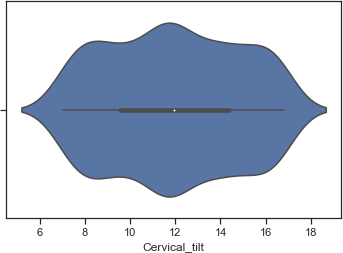


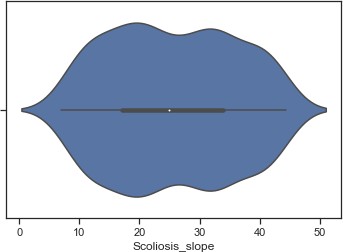












# Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

## 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]:

**1** 68.832021 22.218482 50.092194 46.613539 105.985135

**Pelvic\_incidence Pelvic\_tilt Lumbar\_lordosis\_angle Sacral\_slope Pelvic\_radius Degree\_ 0** 39.056951 10.060991 25.015378 28.995960 114.405425

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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]:

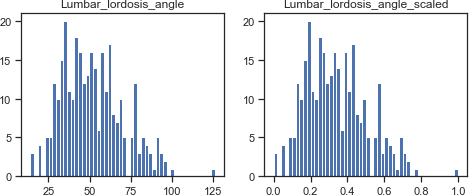
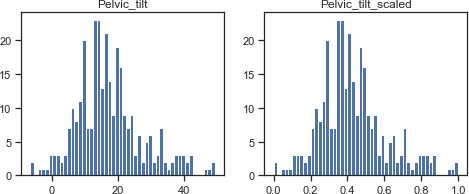
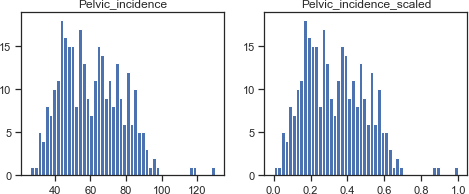
*# Проверим, что масштабирование не повлияло на распределение данных*

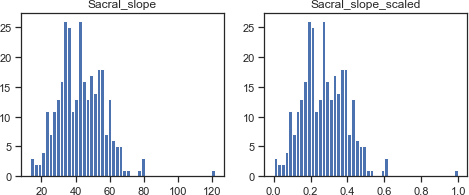
**for** col **in** scale\_cols: col\_scaled **=** col **+** '\_scaled'

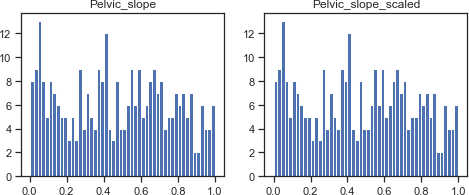
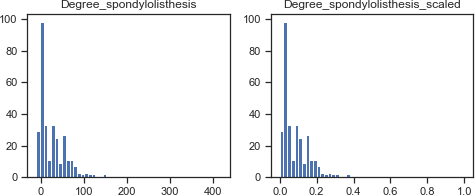
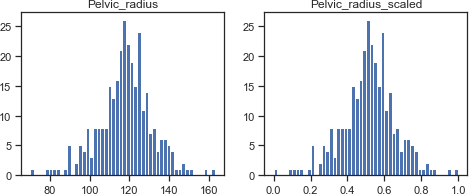
fig, ax **=** plt**.**subplots(1, 2, figsize**=**(8,3)) ax[0]**.**hist(data[col], 50)

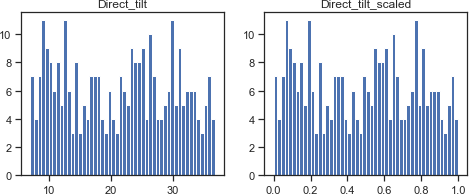
ax[1]**.**hist(data[col\_scaled], 50)

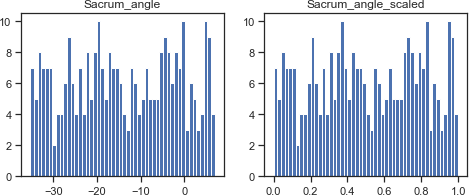
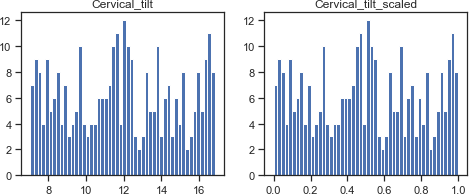
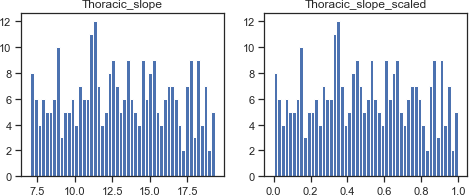
ax[0]**.**title**.**set\_text(col) ax[1]**.**title**.**set\_text(col\_scaled) plt**.**show()

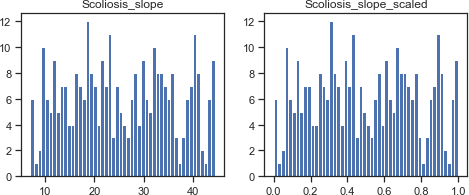












In [23]:

corr\_cols\_1 **=** scale\_cols **+** ['Class\_att\_le'] corr\_cols\_1

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

Out[23]:

In [24]:

scale\_cols\_postfix **=** [x**+**'\_scaled' **for** x **in** scale\_cols] corr\_cols\_2 **=** scale\_cols\_postfix **+** ['Class\_att\_le'] corr\_cols\_2

['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']

Out[24]:

In [25]:

sns**.**set(style**=**"white")

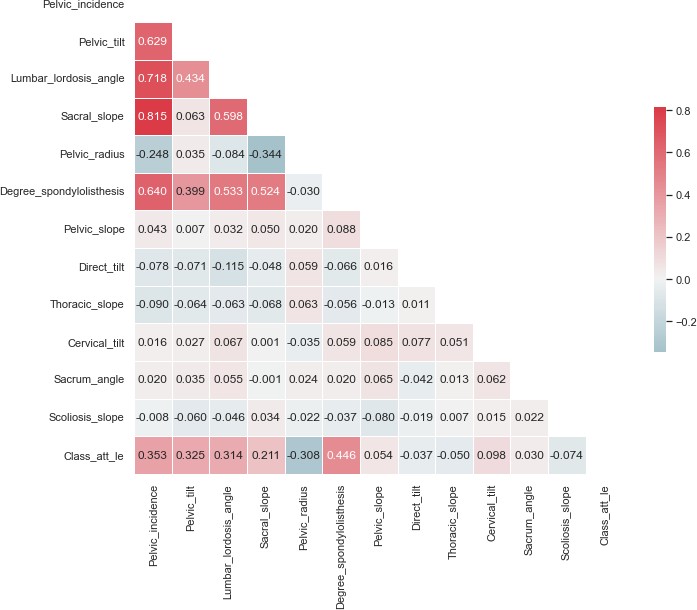
corr **=** data[corr\_cols\_1]**.**corr()

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**)

['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']

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



In [26]:

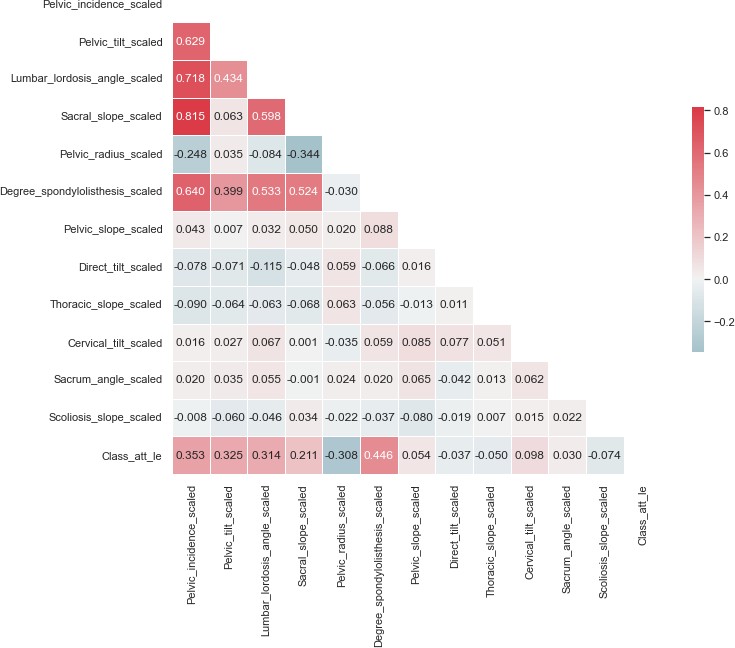
sns**.**set(style**=**"white")

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);

## "Pelvic\_incidence" (0.353);

* 1. "Pelvic\_tilt" (0.325)

## "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.

# Выбор метрик для последующей оценки качества моделей

В качестве метрик для решения задачи классификации будем использовать:

In [27]:

## 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.

*# 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()

# Выбор наиболее подходящих моделей для решения задачи классификации

## Для задачи классификации будем использовать следующие модели:

In [29]:

## Логистическая регрессия Метод ближайших соседей Машина опорных векторов Решающее дерево

Случайный лес Градиентный бустинг

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

*# 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]:

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,

(309, 4)

In [32]:

Out[32]:

X\_train**.**shape, X\_test**.**shape, Y\_train**.**shape, Y\_test**.**shape

((278, 4), (31, 4), (278,), (31,))

# Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится

In [33]:

*# Модели*

clas\_models **=** {'LogR': LogisticRegression(),

'KNN\_5':KNeighborsClassifier(n\_neighbors**=**5), 'SVC':SVC(),

'Tree':DecisionTreeClassifier(), 'RF':RandomForestClassifier(), 'GB':GradientBoostingClassifier()}

# обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.

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')

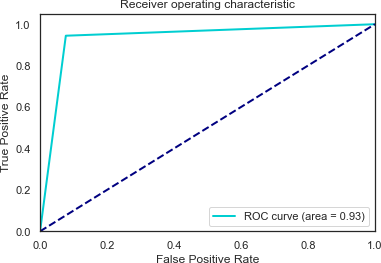
plt**.**show()

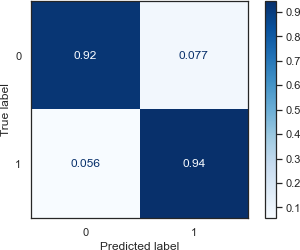
In [36]:

**for** model\_name, model **in** clas\_models**.**items(): train\_model(model\_name, model, clasMetricLogger)

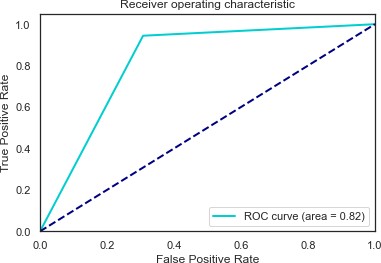
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LogisticRegression()

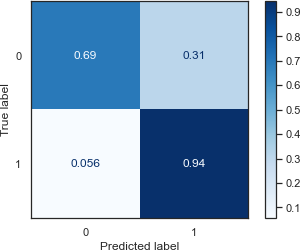
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



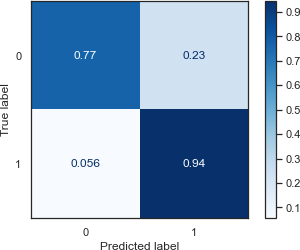
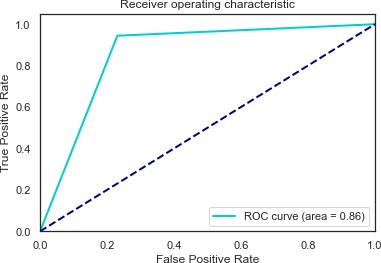


\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KNeighborsClassifier()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

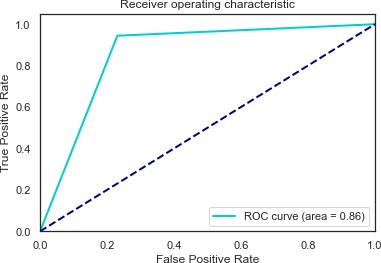


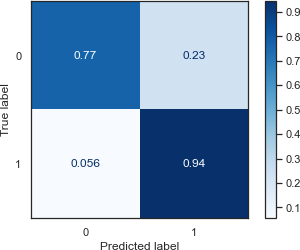
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SVC()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

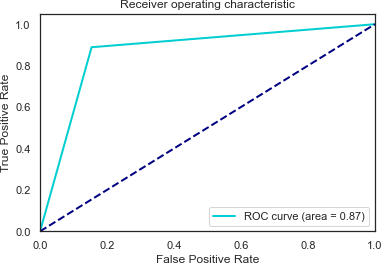
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DecisionTreeClassifier()

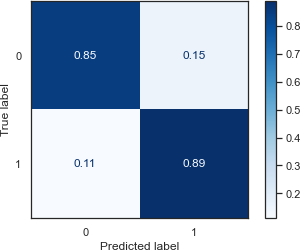
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



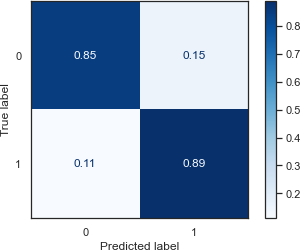
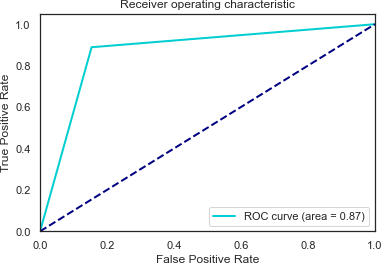


\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RandomForestClassifier()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GradientBoostingClassifier()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In [37]:

n\_range **=** np**.**array(range(1,100,1)) tuned\_parameters **=** [{'n\_neighbors': n\_range}] tuned\_parameters

# Подбор гиперпараметров для выбранных моделей.

## Метод ближайших соседей

Out[37]:

In [38]:

[{'n\_neighbors': array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1

3, 14, 15, 16, 17,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 34, | 18,  35, | 19,  36, | 20,  37, | 21,  38, | 22,  39, | 23,  40, | 24,  41, | 25,  42, | 26,  43, | 27,  44, | 28,  45, | 29,  46, | 30,  47, | 31,  48, | 32,  49, | 33,  50, |
| 51, |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 68, | 52,  69, | 53,  70, | 54,  71, | 55,  72, | 56,  73, | 57,  74, | 58,  75, | 59,  76, | 60,  77, | 61,  78, | 62,  79, | 63,  80, | 64,  81, | 65,  82, | 66,  83, | 67,  84, |
| 85, |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}]

gs\_KNN **=** GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv**=**5, sc gs\_KNN**.**fit(X\_train, Y\_train)

Out[38]:

In [39]:

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, | 3 |
| 4, |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| 35, | 36, | 37, | 38, | 39, | 40, | 41, | 42, | 43, | 44, | 45, | 46, | 47, | 48, 49, | | 50, | 5 |
| 1, |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| 52, | 53, | 54, | 55, | 56, | 57, | 58, | 59, | 60, | 61, | 62, | 63, | 64, | 65, 66, | | 67, | 6 |
| 8, |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| 69, | 70, | 71, | 72, | 73, | 74, | 75, | 76, | 77, | 78, | 79, | 80, | 81, | 82, 83, | | 84, | 8 |
| 5, |  |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |
| 86, | 87, | 88, | 89, | 90, | 91, | 92, | 93, | 94, | 95, | 96, | 97, | 98, | 99])}], | |  |  |

scoring='roc\_auc')

*# Лучшая модель*

gs\_KNN**.**best\_estimator\_

Out[39]:

In [40]:

*# Лучшее значение параметров*

gs\_KNN**.**best\_params\_

KNeighborsClassifier(n\_neighbors=19)

Out[40]:

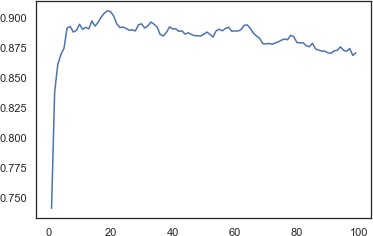
In [41]:

Out[41]:

{'n\_neighbors': 19}

plt**.**plot(n\_range, gs\_KNN**.**cv\_results\_['mean\_test\_score'])

[<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]:

In [43]:

*# Лучшая модель*

gs\_LogR**.**best\_estimator\_

GridSearchCV(cv=5, estimator=LogisticRegression(), param\_grid={'C': array([1.e-03, 1.e+00, 1.e+03])}, scoring='roc\_auc')

Out[43]:

In [44]:

*# Лучшее значение параметров*

gs\_LogR**.**best\_params\_

LogisticRegression()

Out[44]:

In [45]:

*# Изменение качества на тестовой выборке*

plt**.**plot(np**.**logspace(**-**3,3,3), gs\_LogR**.**cv\_results\_['mean\_test\_score'])

{'C': 1.0}

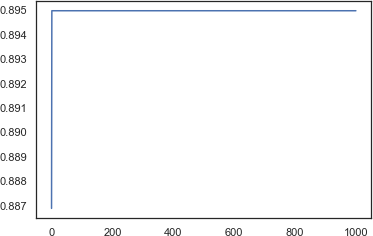
Out[45]:

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)

[<matplotlib.lines.Line2D at 0x7ff5a578f5e0>]



## Машина опорных векторов

Out[46]:

In [47]:

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')

*# Лучшая модель*

gs\_SVC**.**best\_estimator\_

Out[47]:

In [48]:

*# Лучшее значение параметров*

gs\_SVC**.**best\_params\_

SVC(C=4.328761281083062)

Out[48]:

In [49]:

*# Изменение качества на тестовой выборке*

plt**.**plot(np**.**logspace(**-**3,4,12), gs\_SVC**.**cv\_results\_['mean\_test\_score'])

{'C': 4.328761281083062}

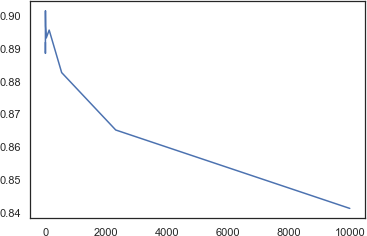
Out[49]:

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)

[<matplotlib.lines.Line2D at 0x7ff5a580cf70>]



## Решающее дерево

Out[50]:

In [51]:

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), param\_grid={'max\_depth': range(1, 20),

'max\_features': range(1, 5)}, scoring='precision')

*# Лучшая модель*

gs\_Tree**.**best\_estimator\_

Out[51]:

In [52]:

*# Лучшее значение параметров*

gs\_Tree**.**best\_params\_

DecisionTreeClassifier(max\_depth=1, max\_features=4)

Out[52]:

In [53]:

{'max\_depth': 1, 'max\_features': 4}

## Случайный лес

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

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)

Out[53]:

In [54]:

param\_grid={'max\_leaf\_nodes': range(2, 12),

'max\_samples': range(2, 22)}, scoring='roc\_auc')

*# Лучшая модель*

gs\_RF**.**best\_estimator\_

Out[54]:

In [55]:

*# Лучшее значение параметров*

gs\_RF**.**best\_params\_

RandomForestClassifier(max\_leaf\_nodes=7, max\_samples=21)

Out[55]:

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)

{'max\_leaf\_nodes': 7, 'max\_samples': 21}

## Градиентный бустинг

Out[56]:

In [57]:

GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), param\_grid={'max\_features': range(1, 4),

'max\_leaf\_nodes': range(2, 22)}, scoring='f1')

*# Лучшая модель*

gs\_GB**.**best\_estimator\_

Out[57]:

In [58]:

*# Лучшее значение параметров*

gs\_GB**.**best\_params\_

GradientBoostingClassifier(max\_features=1, max\_leaf\_nodes=4)

Out[58]:

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\_

}

{'max\_features': 1, 'max\_leaf\_nodes': 4}

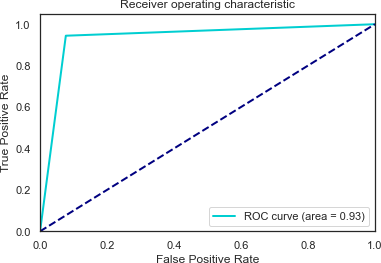
# Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

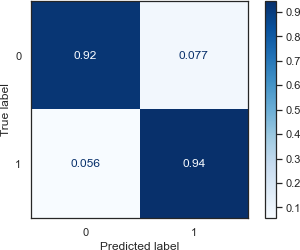
In [60]:

**for** model\_name, model **in** models\_grid**.**items(): train\_model(model\_name, model, clasMetricLogger)

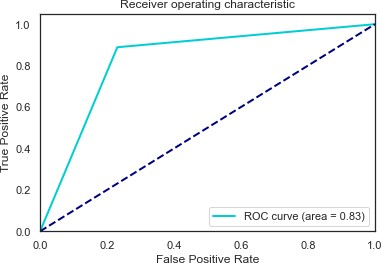
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* LogisticRegression()

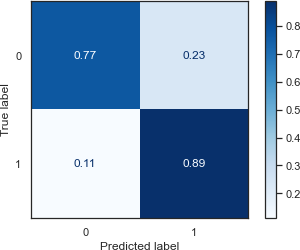
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



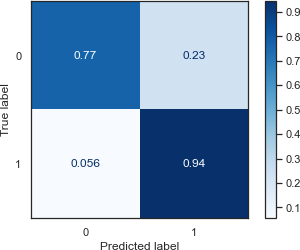
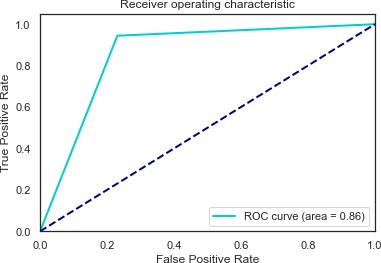


\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* KNeighborsClassifier(n\_neighbors=19)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

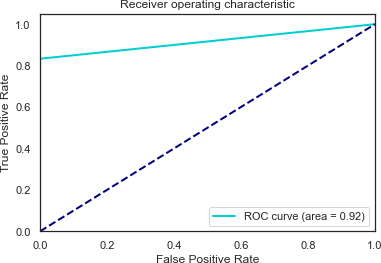


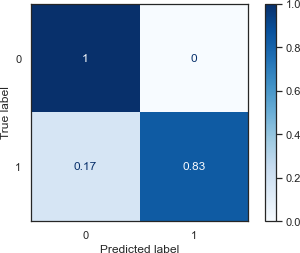
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SVC(C=4.328761281083062)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

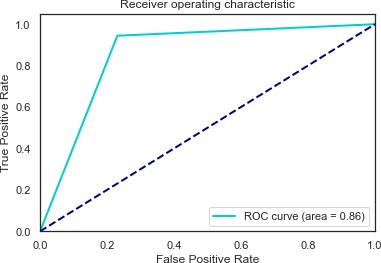
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DecisionTreeClassifier(max\_depth=1, max\_features=4)

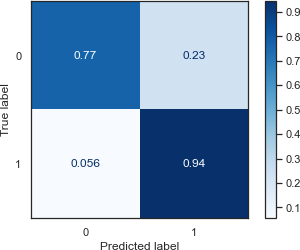
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



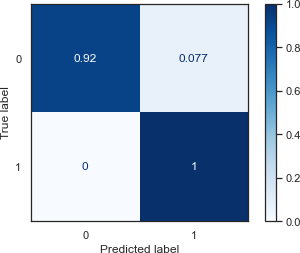
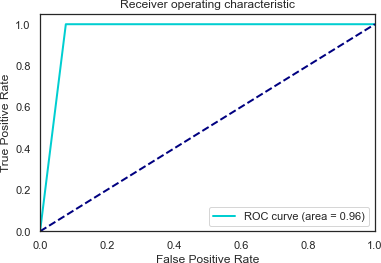


\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* RandomForestClassifier(max\_leaf\_nodes=7, max\_samples=21)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GradientBoostingClassifier(max\_features=1, max\_leaf\_nodes=4)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

In [61]:

*# Метрики качества модели*

clas\_metrics **=** clasMetricLogger**.**df['metric']**.**unique() clas\_metrics

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

Out[61]:

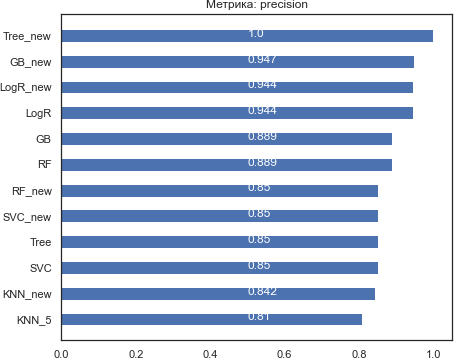
In [62]:

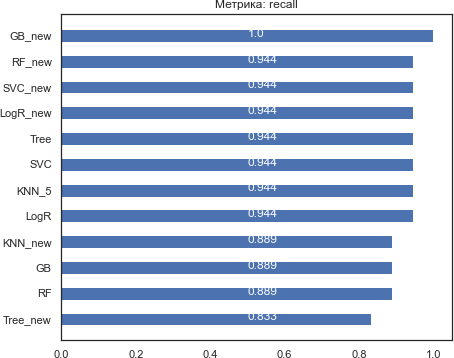
*# Построим графики метрик качества модели*

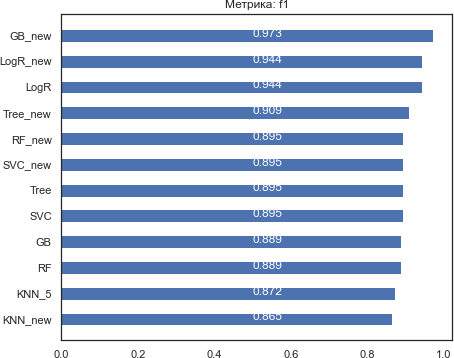
**for** metric **in** clas\_metrics:

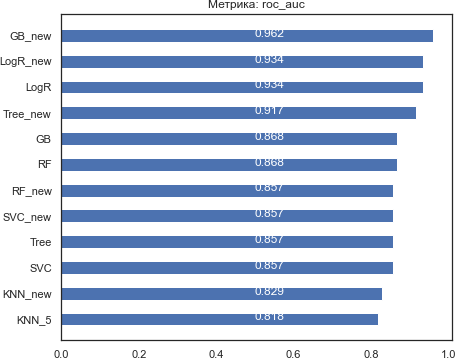
clasMetricLogger**.**plot('Метрика: ' **+** metric, metric, figsize**=**(7, 6))

array(['precision', 'recall', 'f1', 'roc\_auc'], dtype=object)









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

In [ ]:

In [ ]: