Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

«Решение задачи классификации»

Студент РТ5-61Б _____ **А.С. Акушко**
(Группа) (Подпись, дата) (И.О.Фамилия)

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

Консультант _____ **Ю.Е. Гапанюк**
(Подпись, дата) (И.О.Фамилия)

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ5
(Индекс)
_____ В.М. Черненький
(И.О.Фамилия)
«_____»_____20____г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»

Студент группы РТ5-61Б

Акушко Антон Сергеевич
(Фамилия, имя, отчество)

Тема курсового проекта «Решение задачи классификации»

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

График выполнения проекта: 25% к 3 нед., 50% к 9 нед., 75% к 12 нед., 100% к 16 нед.

Задание Решение задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Оформление курсового проекта:

Расчетно-пояснительная записка на 36 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

Дата выдачи задания « 7 » февраля 2020 г.

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

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

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

Содержание

Введение

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные в рамках лекций и лабораторных работ по дисциплине.

В рамках курсового проекта было проведено типовое исследование – решение задачи машинного обучения на основе материалов дисциплины.

Основная часть

Задание

Схема типового исследования, проводимого студентом в рамках курсовой работы, содержит выполнение следующих шагов:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
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.
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.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
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.

Приведенная схема исследования является рекомендуемой. В зависимости от решаемой задачи возможны модификации.

Последовательность действий

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

1. Аэродинамические характеристики профиля (Aerodynamic characteristics of the profile)
2. Коэффициент подъёмной силы профиля (Profile lift coefficient)
3. Коэффициент момента тангажа профиля (Profile pitch moment factor)
4. Аэродинамический фокус профиля (Aerodynamic focus profile)
5. Безударное обтекание профиля (Shockless profile flow)
6. Подсасывающая сила (Suction Force)
7. Коэффициент лобового сопротивления профиля (Profile drag coefficient)
8. Сопротивление трения крыла (Friction Resistance Wing)
9. Профильное сопротивление крыла (Wing Profile Resistance)
10. Индуктивное сопротивление крыла (Inductance of the wing)
11. Критическое число Маха (Critical Mach Number)
12. Коэффициент подъёмной силы крыла (The coefficient of lifting force of the wing)
13. Класс (Class_att)

ë

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_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 sklearn.preprocessing import LabelEncoder
from IPython.display import Image
%matplotlib inline
sns.set(style="ticks")
```

In [8]:

```
col_list = ['Aerodynamic_characteristics_of_the_profile',
               'Profile_lift_coefficient',
               'Profile_pitch_moment_factor',
               'Aerodynamic_focus_profile',
               'Shockless_profile_flow',
               'Suction_Force',
               'Profile_drag_coefficient',
               'Friction_Resistance_Wing',
               'Wing_Profile_Resistance',
               'Inductance_of_the_wing',
               'Critical_Mach_Number',
               'The_coefficient_of_lifting_force_of_the_wing',
               'Class_att',
               'To_drop']
data = pd.read_csv('data/Dataset_spine.csv', names=col_list, header=1, sep=",")
data.drop('To_drop', axis=1, inplace=True)
data = pd.read_csv('data/Dataset_weather.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.

In [4]:

```
data.head()
```

Out[4]:

In [5]: data.head()

Out[5]:

	Aerodynamic_characteristics_of_the_profile	Profile_lift_coefficient	Profile_pitch_moment_factor	Aerodynamic_focus_profile	Shockless_profile_flow	Suction_Fo
0	39.05695098	10.060991	25.015378	28.995960	114.405425	4.564
1	68.83202098	22.218482	50.092194	46.613539	105.985135	-3.530
2	69.29700807	24.652878	44.311238	44.644130	101.868495	11.211
3	49.71285934	9.652075	28.317406	40.060784	108.168725	7.918
4	40.25019968	13.921907	25.124950	26.328293	130.327871	2.230

In [5]:

```
data.shape
```

Out[5]:

```
(309, 13)
```

In [6]:

```
data.columns
```

Out[6]:

```
Index(['Aerodynamic_characteristics_of_the_profile',
       'Profile_lift_coefficient', 'Profile_pitch_moment_factor',
       'Aerodynamic_focus_profile', 'Shockless_profile_flow', 'Suction_Force',
       'Profile_drag_coefficient', 'Friction_Resistance_Wing',
       'Wing_Profile_Resistance', 'Inductance_of_the_wing',
       'Critical_Mach_Number', 'The_coefficient_of_lifting_force_of_the_wing',
       'Class_att'],
      dtype='object')
  )
```

In [7]:

```
data.dtypes
```

Out[7]:

```
Aerodynamic_characteristics_of_the_profile       object
Profile_lift_coefficient                        float64
Profile_pitch_moment_factor                     float64
Aerodynamic_focus_profile                       float64
Shockless_profile_flow                          float64
Suction_Force                                   float64
Profile_drag_coefficient                        float64
```

In [8]:

```
Friction_Resistance_Wing                          float64
Wing_Profile_Resistance                           float64
Inductance_of_the_wing                            float64
Critical_Mach_Number                              float64
The_coefficient_of_lifting_force_of_the_wing      float64
Class_att                                          object
dtype: object
```

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

Out[8]:

```
Aerodynamic_characteristics_of_the_profile      0
Profile lift coefficient                        1
Profile_pitch_moment_factor                     1
Aerodynamic_focus_profile                       1
Shockless_profile_flow                          1
Suction_Force                                   1
Profile_drag_coefficient                        1
Friction_Resistance_Wing                        1
Wing_Profile_Resistance                         1
Inductance_of_the_wing                          1
Critical_Mach_Number                            1
The_coefficient_of_lifting_force_of_the_wing    1
Class_att                                       1
dtype: int64
```

In [9]:

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

In [10]:

```
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 [11]:

```
data.head()
```

Out[11]:

	Aerodynamic_characteristics_of_the_profile	Profile_lift_coefficient	Profile_pitch_moment_factor	Aerodynamic_focus_profile	Shockless_profile_flow	Suction_Fo
0	39.05695098	10.060991	25.015378	28.995960	114.405425	4.564
1	68.83202098	22.218482	50.092194	46.613539	105.985135	-3.530
2	69.29700807	24.652878	44.311238	44.644130	101.868495	11.211
3	49.71285934	9.652075	28.317406	40.060784	108.168725	7.918
4	40.25019968	13.921907	25.124950	26.328293	130.327871	2.230

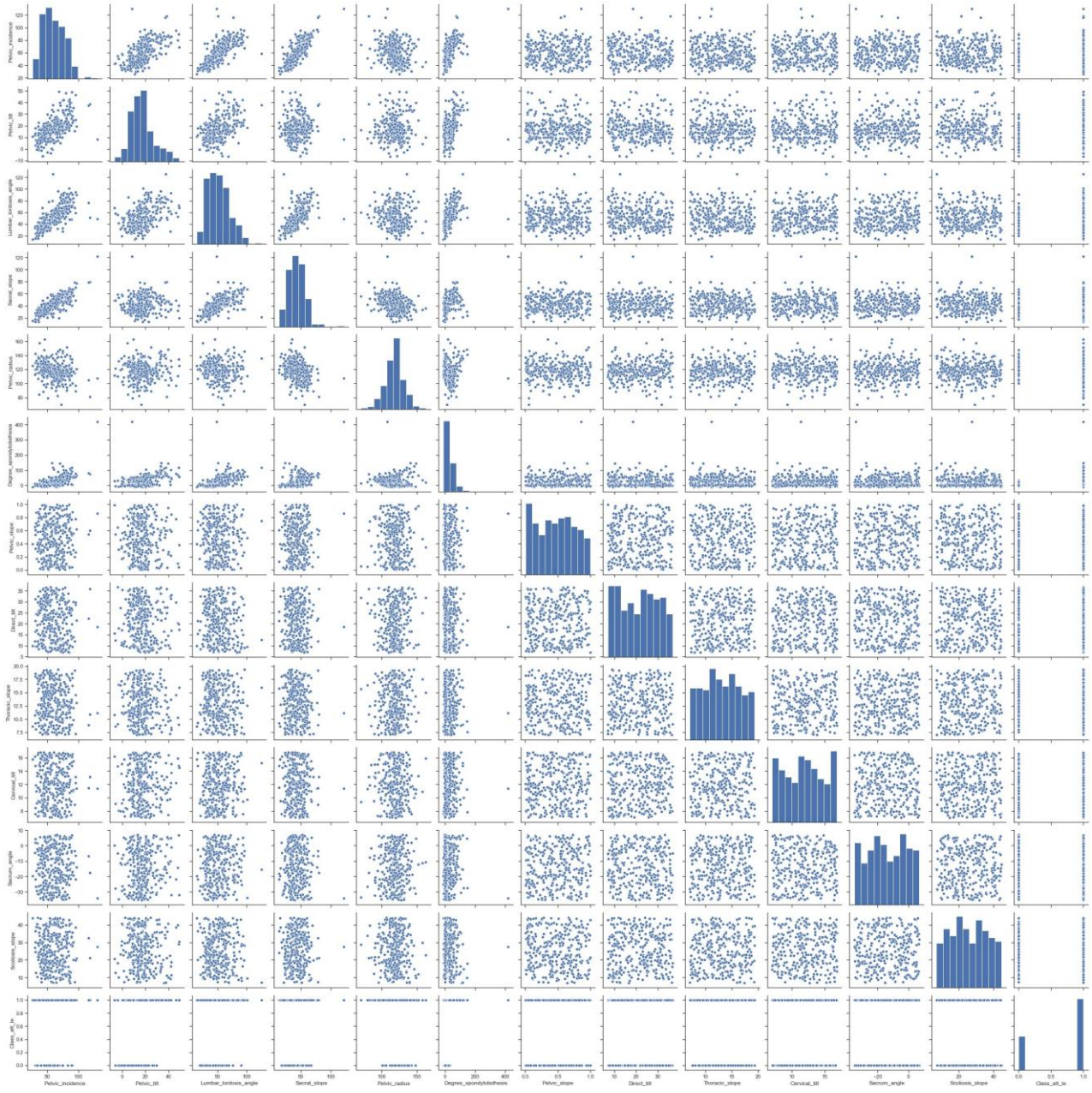
Набор данных не содержит пропусков, категориальные признаки закодированы.

In [12]:

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

Out[12]:

```
<seaborn.axisgrid.PairGrid at 0xca30ff0>
```



In [13]:

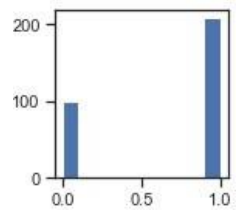
```
sns.pairplot(data, hue="Class_att_le")
```

Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x11ca69d0>
```

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
```

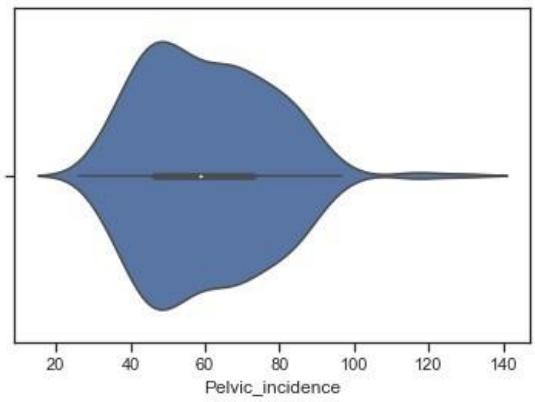
In [16]:

```
# 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)*100))
```

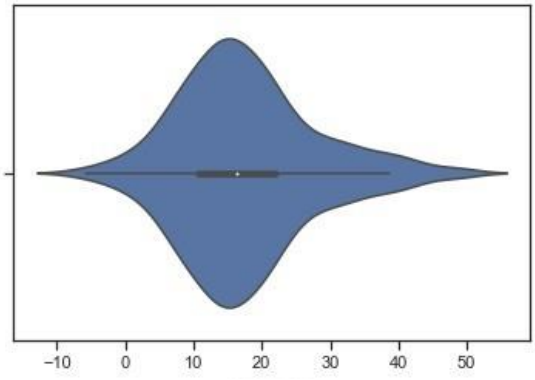
```
Класс 0 составляет 67.64%, а класс 1 составляет 32.36%.
```

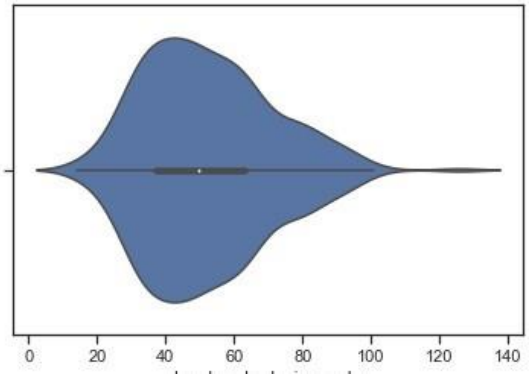
Вывод. Дисбаланс классов присутствует, но является приемлемым.

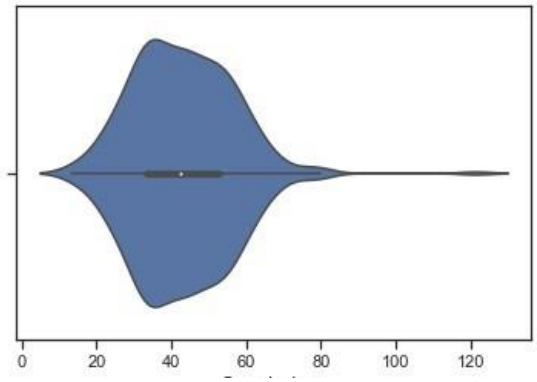
```
# Скрипичные диаграммы для числовых колонок
for col in ['Aerodynamic_characteristics_of_the_profile',
                'Profile_lift_coefficient',
                'Profile_pitch_moment_factor',
                'Aerodynamic_focus_profile',
                'Shockless_profile_flow',
                'Suction_Force',
                'Profile_drag_coefficient',
                'Friction_Resistance_Wing',
                'Wing_Profile_Resistance',
                'Inductance_of_the_wing',
                'Critical_Mach_Number',
                'The_coefficient_of_lifting_force_of_the_wing']:
    sns.violinplot(x=data[col])
    plt.show()
```

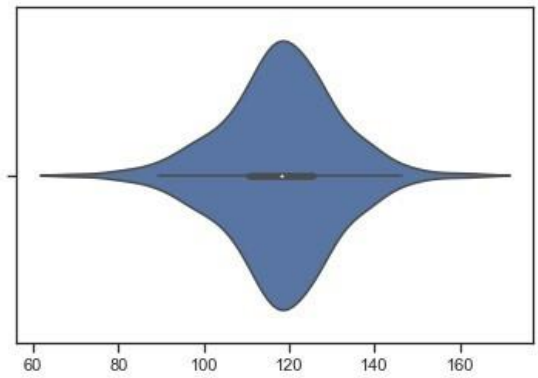


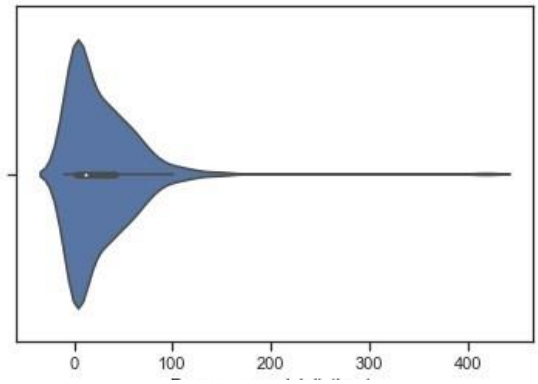
In [17]:

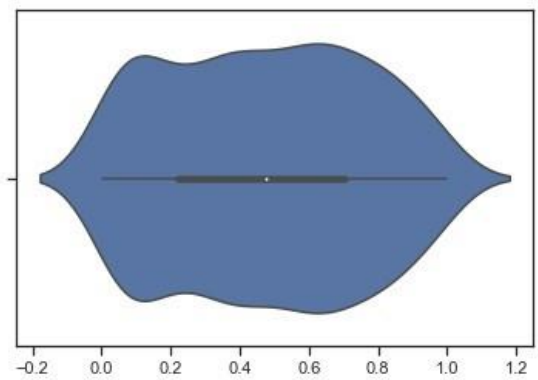


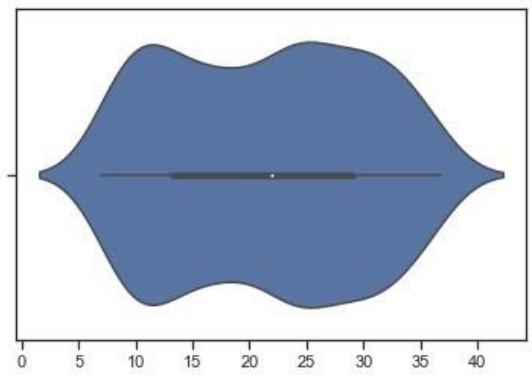


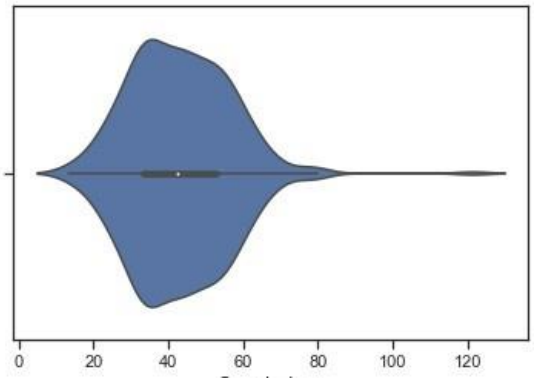

0
20
40
60
80
100
120

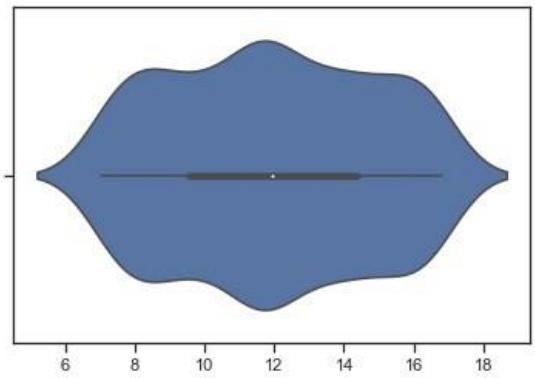

60
80
100
120
140
160

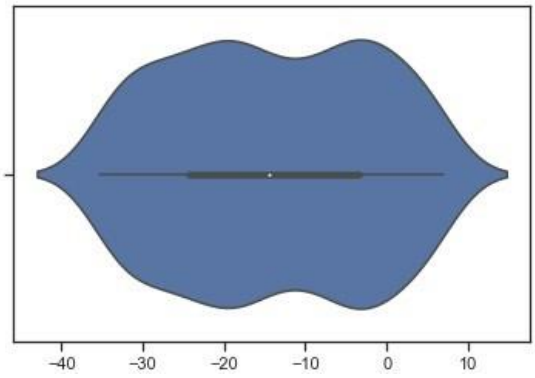

0
100
200
300
400

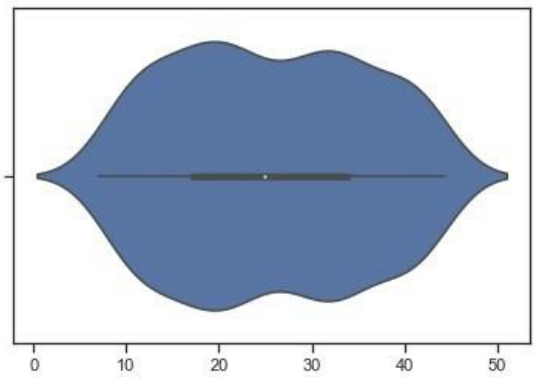

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


0
5
10
15
20
25
30
35
40









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 = ['Aerodynamic_characteristics_of_the_profile',
              'Profile_lift_coefficient',
              'Profile_pitch_moment_factor',
              'Aerodynamic_focus_profile',
              'Shockless_profile_flow',
              'Suction_Force',
              'Profile_drag_coefficient',
              'Friction_Resistance_Wing',
              'Wing_Profile_Resistance',
              'Inductance_of_the_wing',
              'Critical_Mach_Number',
              'The_coefficient_of_lifting_force_of_the_wing']
```

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()
```

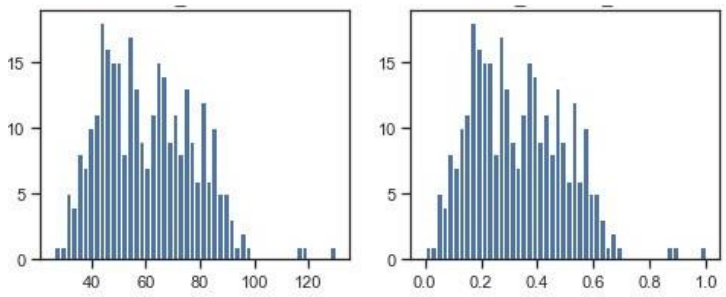
Out[21]:

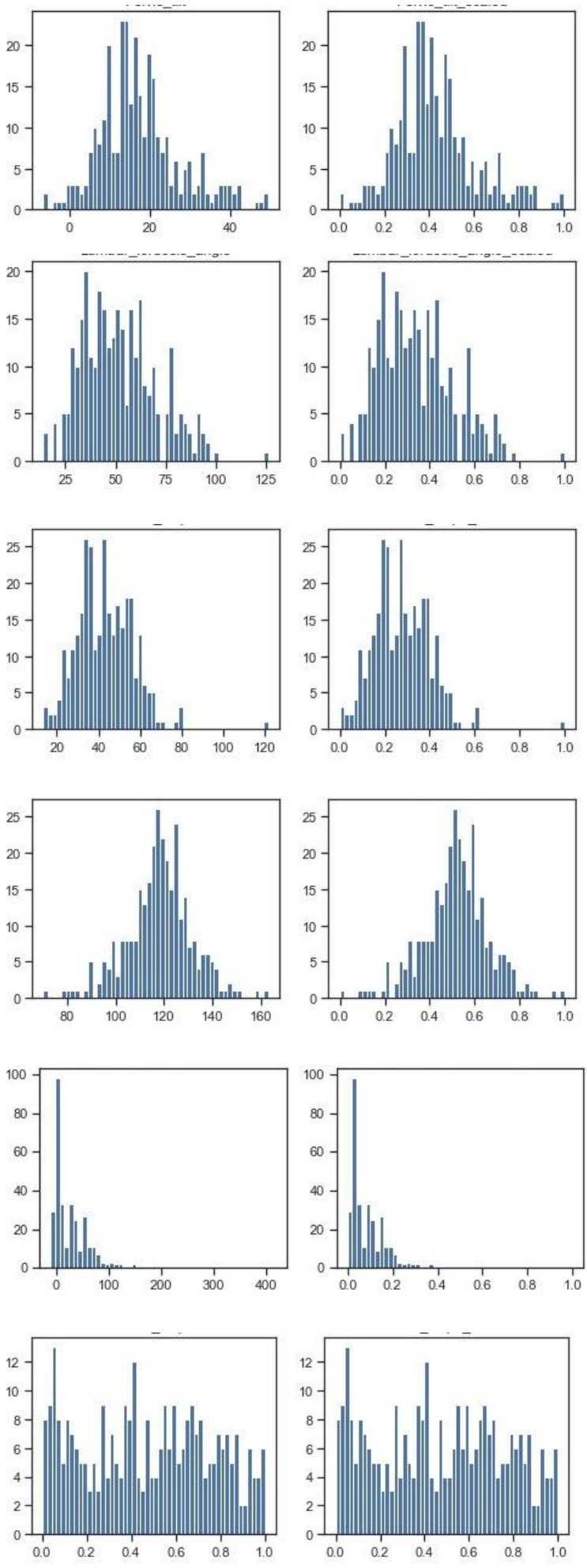
0	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	0.415186	12.8874	17.5323	16.78
1	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	0.474889	26.8343	17.4861	16.65
2	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	0.369345	23.5603	12.7074	11.42
3	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	0.543360	35.4940	15.9546	8.87
4	40.250200	13.921907	25.124950	26.328293	130.327871	2.230652	0.789993	29.3230	12.0036	10.40

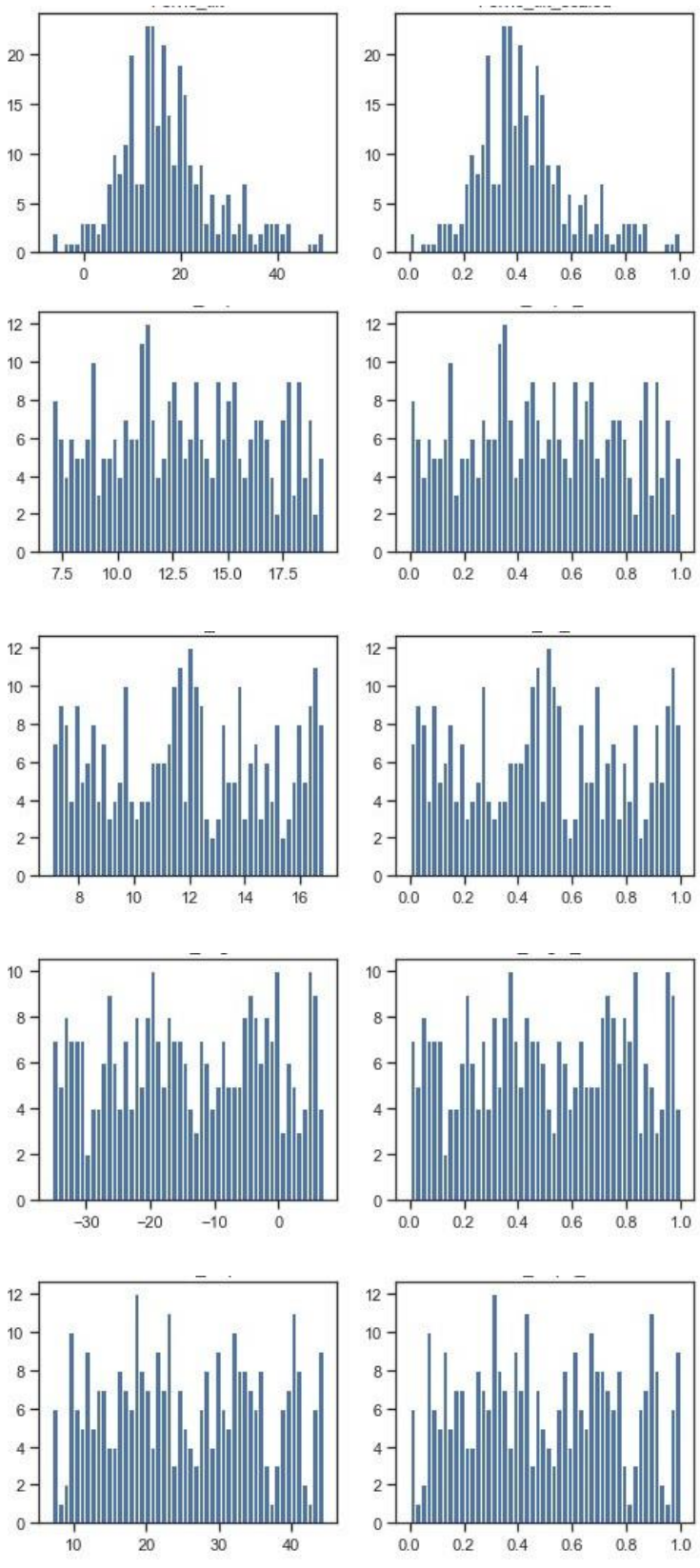
5 rows × 26 columns

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()
```







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]:

```
['Aerodynamic_characteristics
 _of_the_profile',
 'Profile_lift_coefficient',
 'Profile_pitch_moment_factor
 ',
 'Aerodynamic_focus_profile',
 'Shockless_profile_flow',
 'Suction_Force',
 'Profile_drag_coefficient',
 'Friction_Resistance_Wing',
 'Wing_Profile_Resistance',
 'Inductance_of_the_wing',
 'Critical_Mach_Number',
 'The_coefficient_of_lifting_
 force_of_the_wing']
```

In [24]:

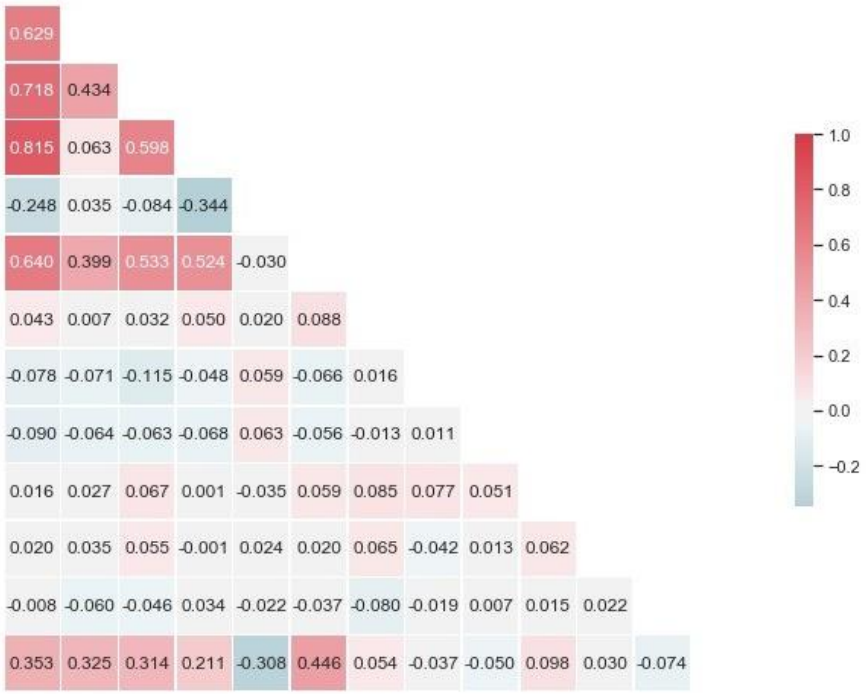
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['Class_att_le']
corr_cols_2
```

Out[24]:

```
['Aerodynamic_characteristics_of_the_pro
 file',
 'Profile_lift_coefficient',
 'Profile_pitch_moment_factor',
 'Aerodynamic_focus_profile',
 'Shockless_profile_flow',
 'Suction_Force',
 'Profile_drag_coefficient',
 'Friction_Resistance_Wing',
 'Wing_Profile_Resistance',
 'Inductance_of_the_wing',
 'Critical_Mach_Number',
 'The_coefficient_of_lifting_force_of_th
 e_wing']
```

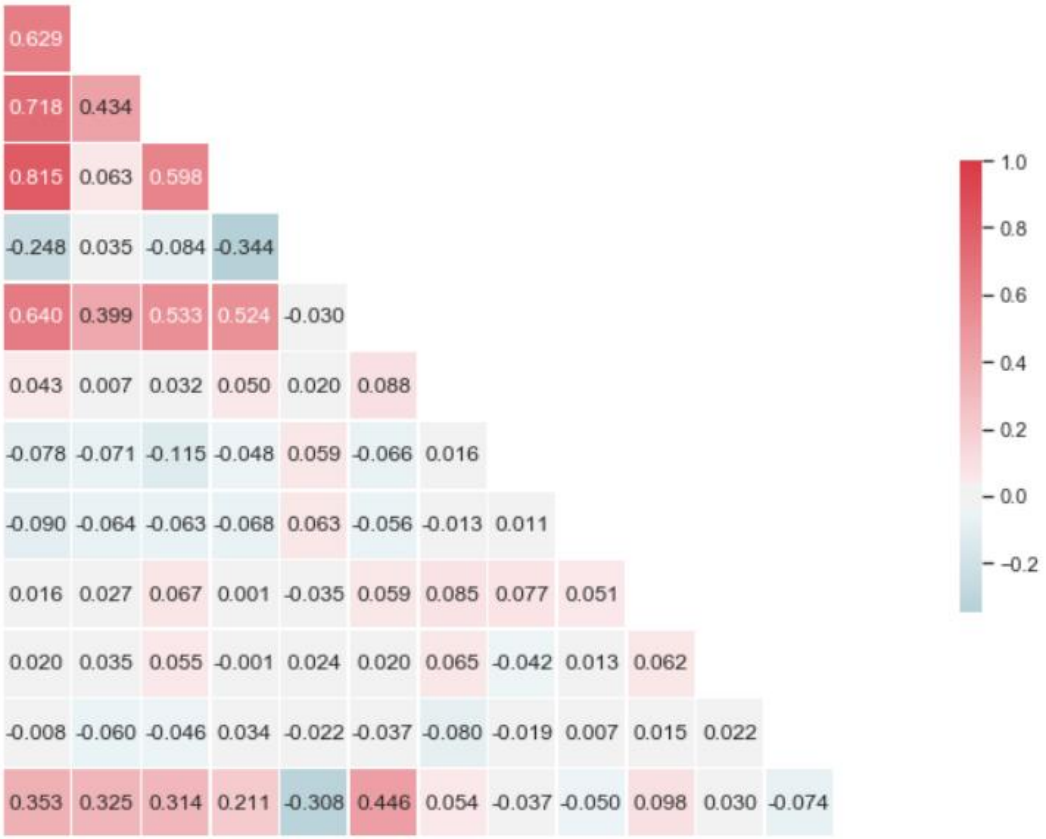
In [25]:

```
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. "Suction_Force" (0.446);
  2. "Aerodynamic_characteristics_of_the_profile" (0.353);
  3. "Profile_lift_coefficient" (0.325)
  4. "Profile_pitch_moment_factor" (0.314) Эти признаки следует оставить в модели классификации.
- Признаки "Aerodynamic_characteristics_of_the_profile" и "The_coefficient_of_lifting_force_of_the_wing" имеют большую корреляцию, поэтому оба признака не следует включать в модель. Будем использовать признак "Aerodynamic_characteristics_of_the_profile".
- 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.
- | -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']==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

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 набораданных

In [29]:

```
# Признаки для задачи классификации
class_cols = ['Aerodynamic_characteristics_of_the_profile,
              'Profile_lift_coefficient,
              'Profile_pitch_moment_facto
              r, 'Suction_Force]
```

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 и тестовую
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=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()
```

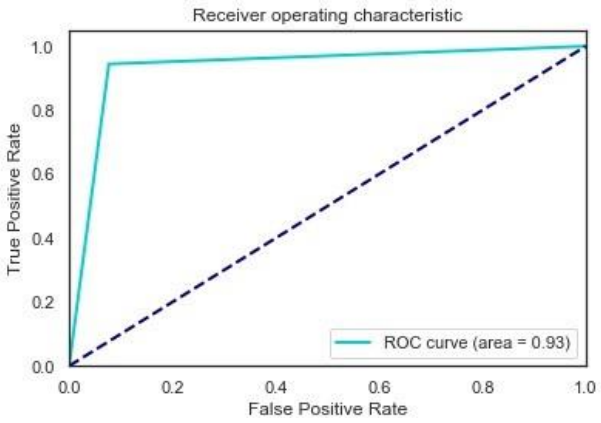
In [35]:

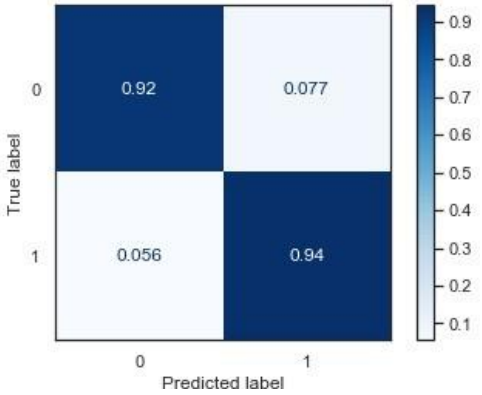
```
def train_model(model_name, model, MetricLogger):
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)

    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]:

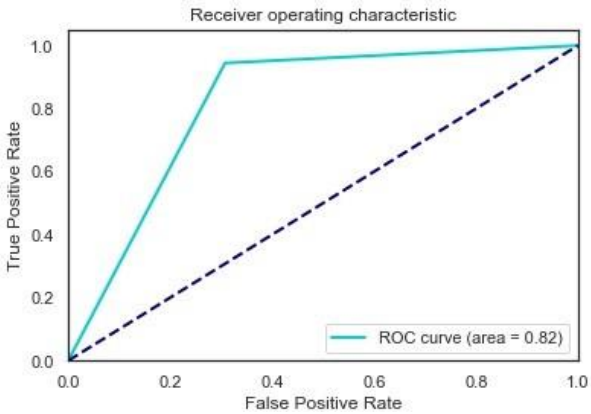
```
for model_name, model in clas_models.items():
    train_model(model_name, model, clasMetricLogger)
```

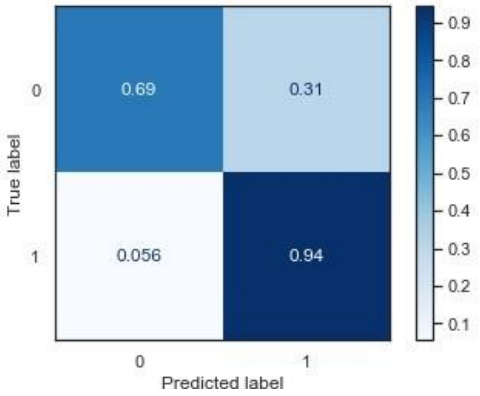
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



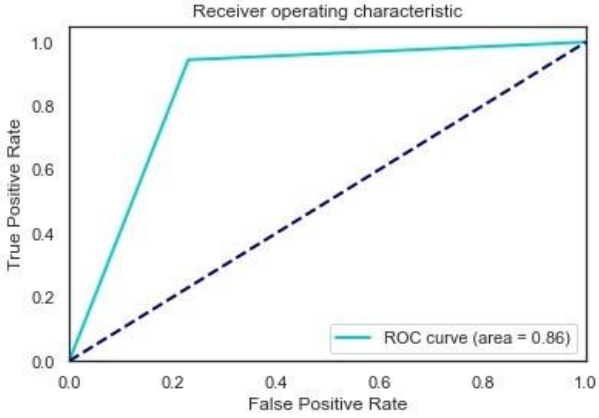


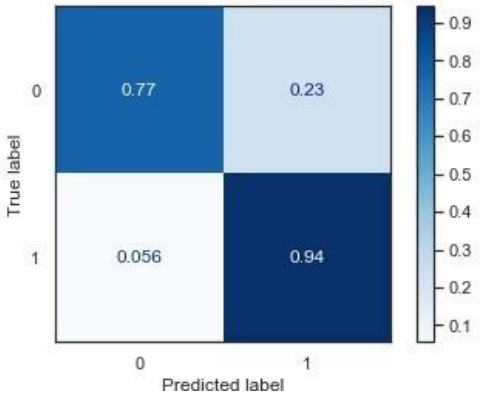

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
**************************************************
```



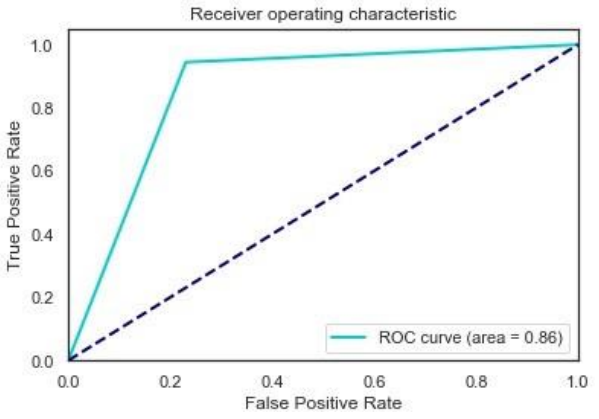


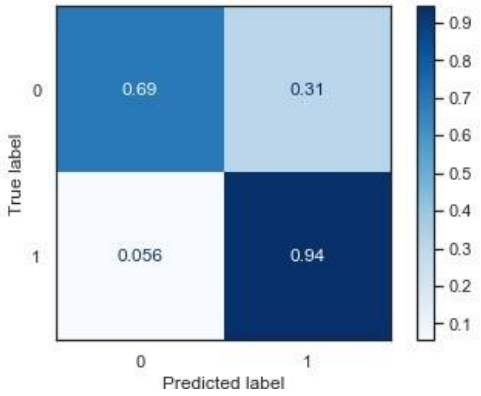

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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



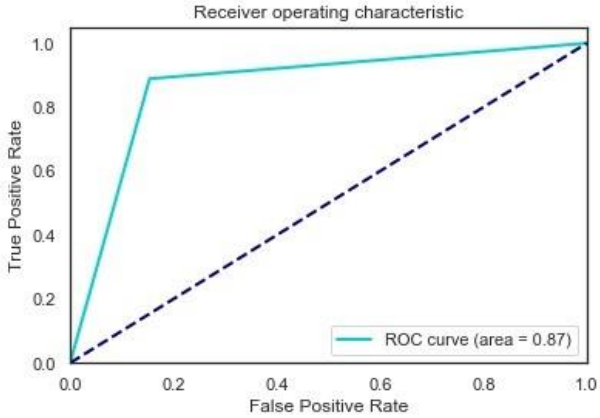


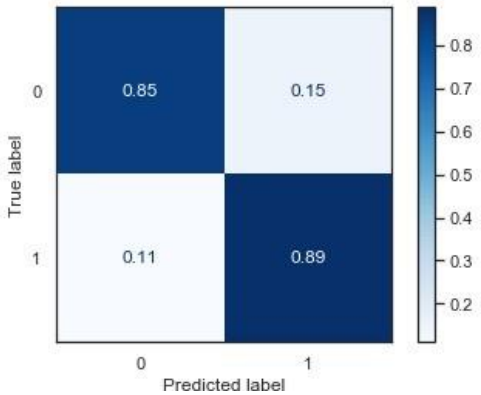

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                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



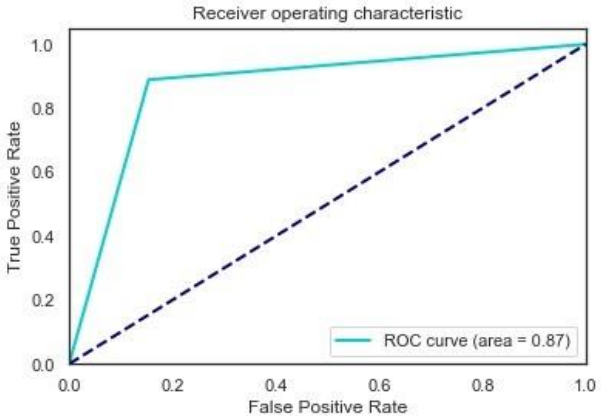


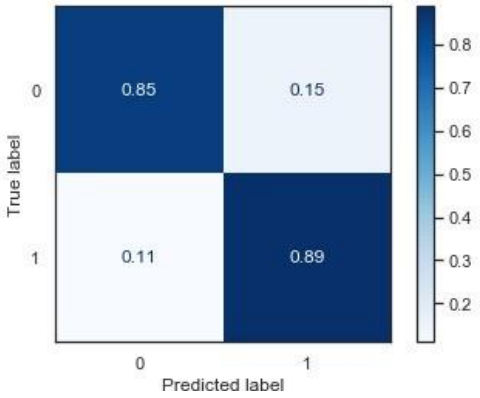

```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0,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






```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```





9) Подбор гиперпараметров для выбранных моделей.

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, scoring='roc_auc')
gs_KNN.fit(X_train, Y_train)
```

Out[38]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             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])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

In [39]:

```
# Лучшая модель
gs_KNN.best_estimator_
```

Out[39]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=19, p=2,
                     weights='uniform')
```

In [40]:

```
# Лучшее значение параметров
gs_KNN.best_params_
```

Out[40]:

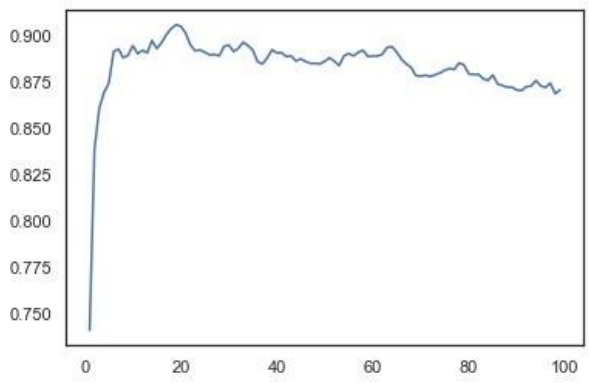
```
{'n_neighbors': 19}
```

In [41]:

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

Out[41]:

```
[<matplotlib.lines.Line2D at 0x1b71ccd0>]
```



Логистическая регрессия

In [42]:

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

Out[42]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.e-03, 1.e+00, 1.e+03])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

In [43]:

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

Out[43]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [44]:

```
# Лучшее значение параметров
gs_LogR.best_params_
```

Out[44]:

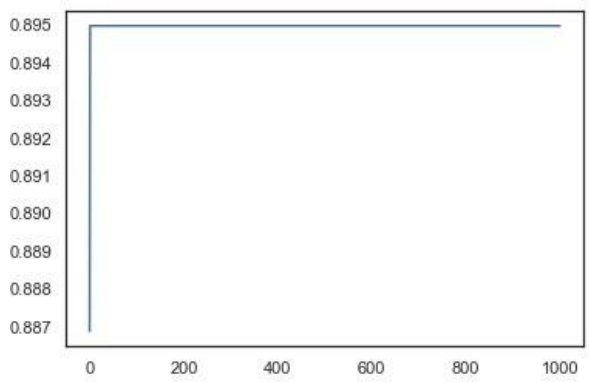
```
{'C': 1.0}
```

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 0x1b74d4b0>]
```



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, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': array([1.00000000e-03, 5.33669923e-03, 2.84803587e-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])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

In [47]:

```
# Лучшая модель
gs_SVC.best_estimator_
```

Out[47]:

```
SVC(C=4.328761281083062, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [48]:

```
# Лучшее значение параметров
gs_SVC.best_params_
```

Out[48]:

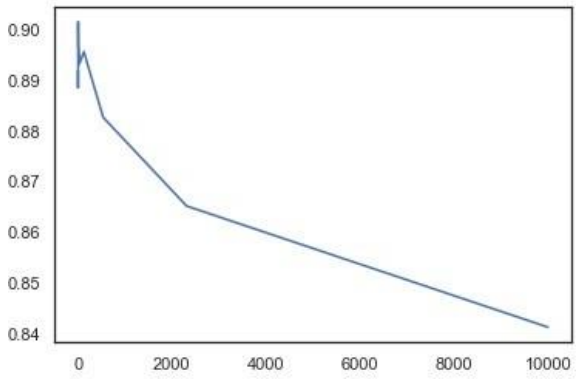
```
{'C': 4.328761281083062}
```

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 0x1b786570>]
```



Решающее дерево

In [50]:

```
tree_params={"max_depth":range(1,20), "max_features":range(1,5)}
gs_Tree = GridSearchCV(DecisionTreeClassifier(), tree_params, cv=5, scoring='precision')
gs_Tree.fit(X_train, Y_train)
```

Out[50]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': range(1, 20),
                         'max_features': range(1, 5)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='precision', verbose=0)
```

In [51]:

```
# Лучшая модель
gs_Tree.best_estimator_
```

Out[51]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=1, max_features=4, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

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='roc_auc')
gs_RF.fit(X_train, Y_train)
```

Out[53]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_leaf_nodes': range(2, 12),
                         'max_samples': range(2, 22)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

In [54]:

```
# Лучшая модель
gs_RF.best_estimator_
```

Out[54]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=8, max_samples=17,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [55]:

```
# Лучшее значение параметров
gs_RF.best_params_
```

Out[55]:

```
{'max_leaf_nodes': 8, 'max_samples': 17}
```

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, scoring='f1')
gs_GB.fit(X_train, Y_train)
```

Out[56]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_mse',
                                                  init=None, learning_rate=0.1,
                                                  loss='deviance', max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=None,
                                                  presort='deprecated',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_features': range(1, 4),
                         'max_leaf_nodes': range(2, 22)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='f1', verbose=0)
```

In [57]:

```
# Лучшая модель
gs_GB.best_estimator_
```

Out[57]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=3, max_leaf_nodes=20,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [58]:

```
# Лучшее значение параметров
gs_GB.best_params_
```

Out[58]:

```
{'max_features': 3, 'max_leaf_nodes': 20}
```

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

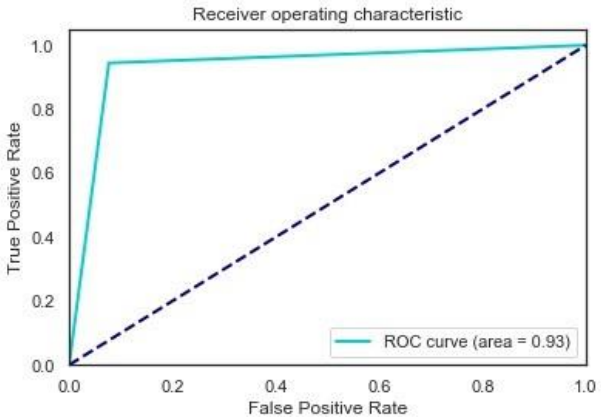
In [59]:

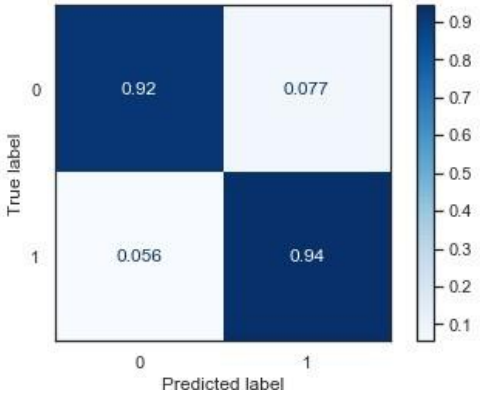
```
models_grid = { 'LogR_new':gs_LogR.best_estimator_,
                'KNN_new':gs_KNN.best_estimator_,
                 'SVC_new':gs_SVC.best_estimator_,
               'Tree_new':gs_Tree.best_estimator_,
                'RF_new':gs_RF.best_estimator_,
               'GB_new':gs_GB.best_estimator_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

In [60]:

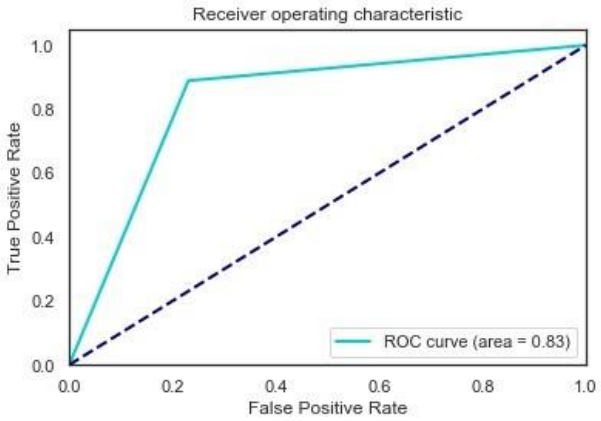
```
for model_name, model in models_grid.items():
    train_model(model_name, model, clasMetricLogger)
```

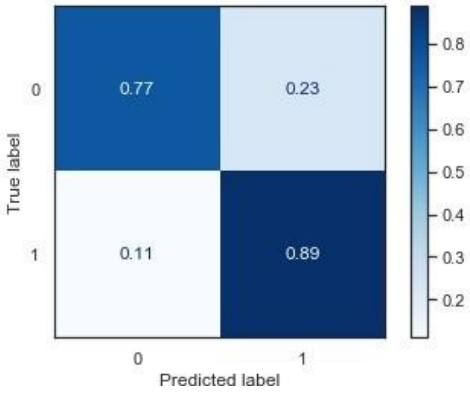
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



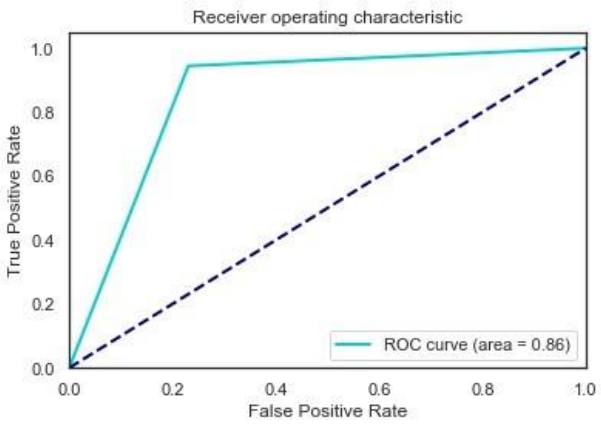


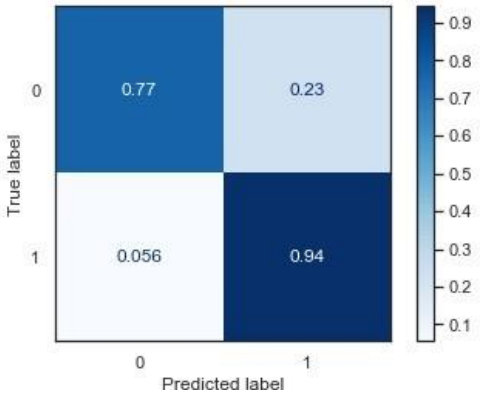

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=19, p=2,
                     weights='uniform')
**************************************************
```



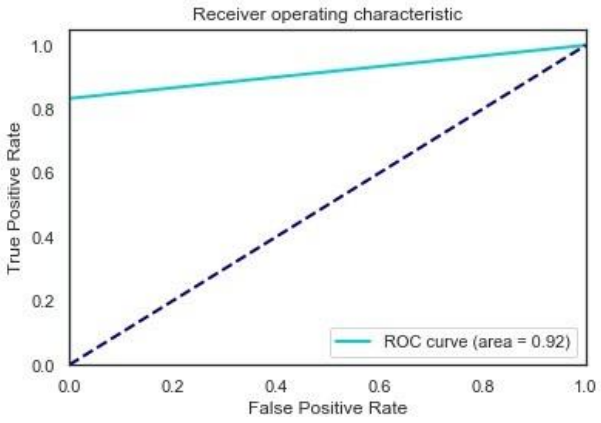


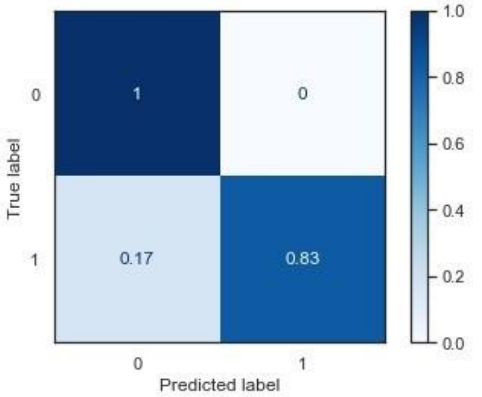
```
**************************************************
SVC(C=4.328761281083062, break_ties=False, cache_size=200, class_weight=None,
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma='scale',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
**************************************************
```



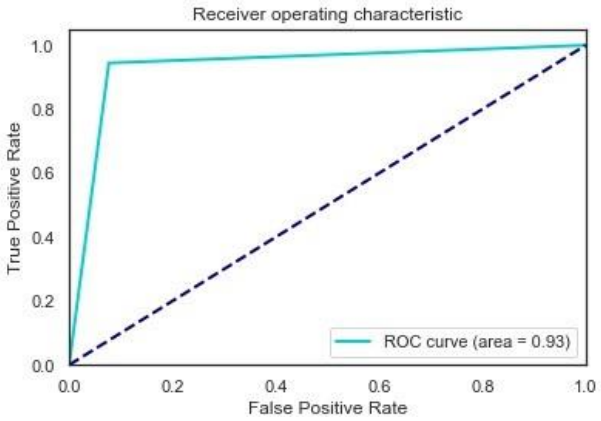


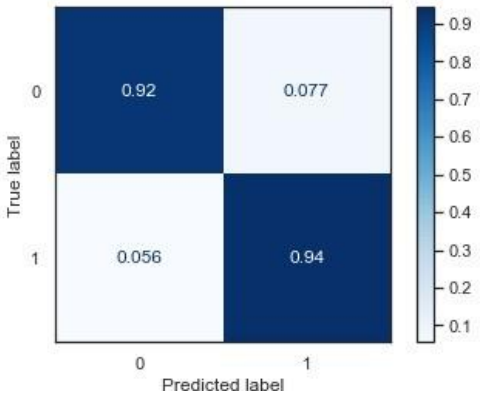

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=1, max_features=4, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```



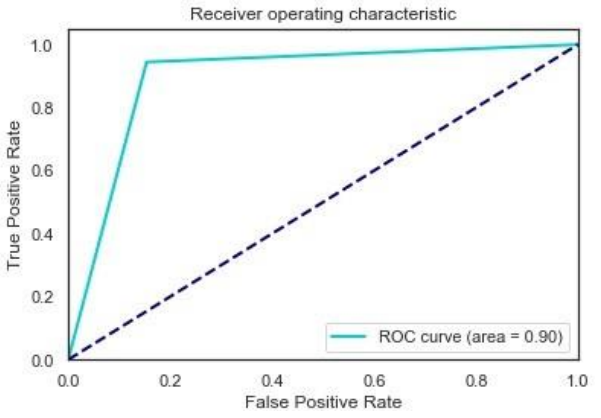


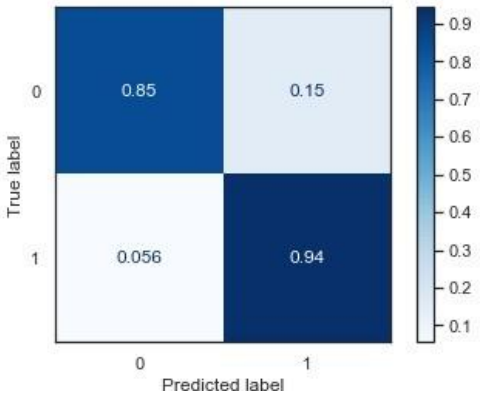
```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=8, max_samples=17,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
**************************************************
```






```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=3, max_leaf_nodes=20,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```





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

In [61]:

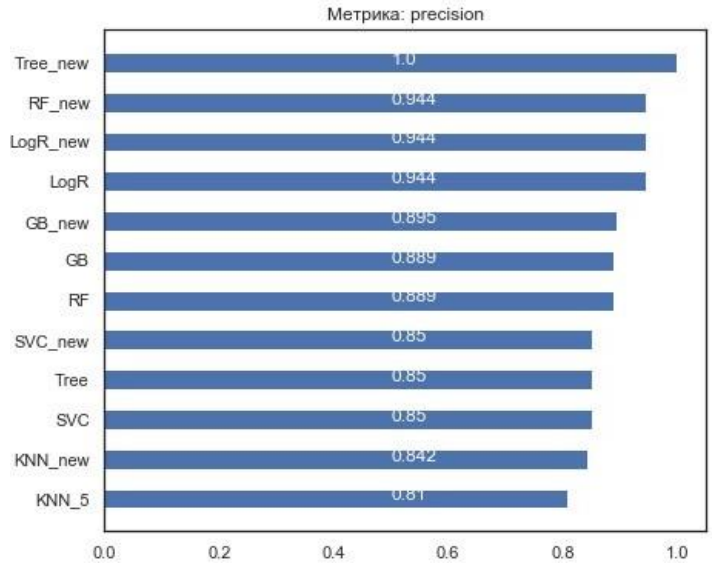
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

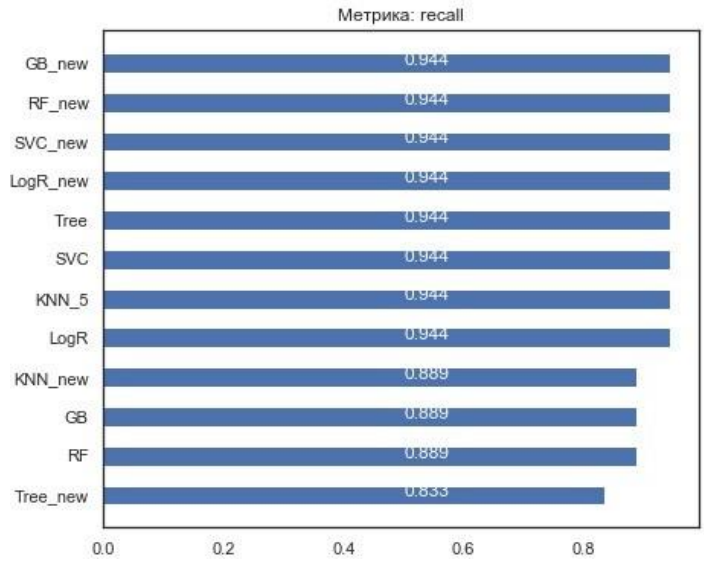
Out[61]:

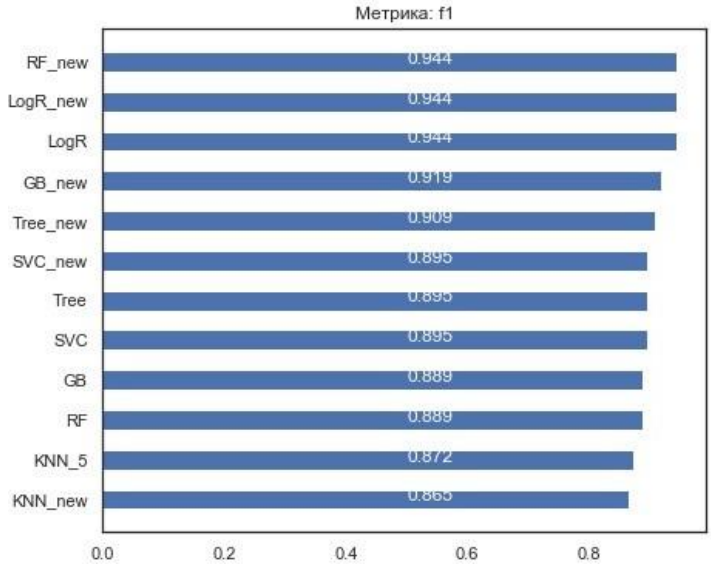
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

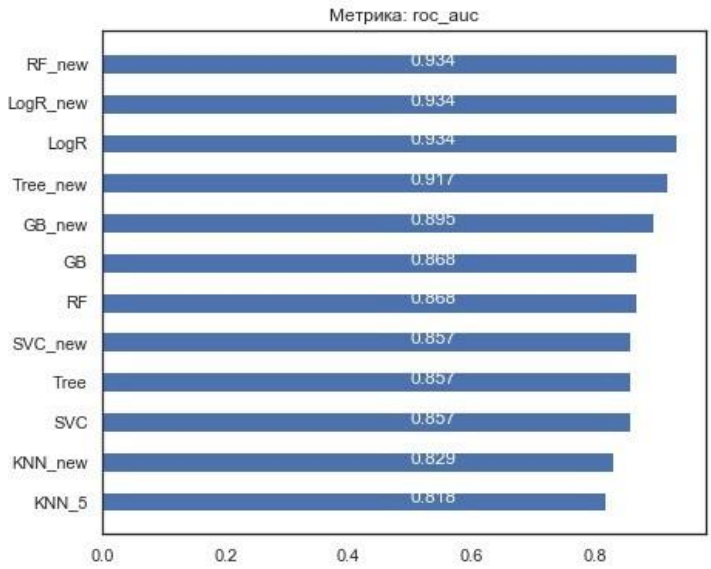
In [62]:

```
# 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.

Заключение

В данном курсовом проекте была решена типовая задача машинного обучения. Был выбран набор данных для построения моделей машинного обучения, проведен разведочный анализ данных и построены графики, необходимые для понимания структуры данных. Были выбраны признаки, подходящие для построения моделей, масштабированы данные и проведен корреляционный анализ данных. Это позволило сформировать промежуточные выводы о возможности построения моделей машинного обучения.

На следующем этапе были выбраны метрики для последующей оценки качества моделей и наиболее подходящие модели для решения задачи классификации. Затем были сформированы обучающая и тестовая выборки на основе исходного набора данных и построено базовое решение для выбранных моделей без подбора гиперпараметров.

Следующим шагом был подбор гиперпараметров для выбранных моделей, после чего мы смогли сравнить качество полученных моделей с качеством baseline-моделей. Большинство моделей, для которых были подобраны оптимальные значения гиперпараметров, показали лучший результат.

В заключение, были сформированы выводы о качестве построенных моделей на основе выбранных метрик. Для наглядности результаты сравнения качества были отображены в виде графиков, а также сделаны выводы в форме текстового описания. Четыре метрики показали, что для выбранного набора данных лучшей моделью оказалась «машина опорных векторов».

Список использованных источников

1. Ю.Е. Гапанюк, Лекции по курсу «Технологии машинного обучения» 2019-2020 учебный год.
2. scikit-learn Machine Learning in Python: [сайт]. URL: https://scikit-learn.org/stable/
3. Lower Back Pain Symptoms Dataset [Электронный ресурс]. URL: https://www.kaggle.com/sammy123/lower-back-pain-symptoms-dataset (дата обращения: 24.05.2020)