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 машинного обучения

Студент группы РТ5-61 _____ Абраменков Г.Г.
(Группа) (Подпись, дата) (И.О.Фамилия)

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

Москва, 2020 г.

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

УТВЕРЖДАЮ
Заведующий кафедрой _____
(Индекс)
_____ _____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»

Студент группы РТ5-61

_____Абраменков Георгий Григорьевич_____
(Фамилия, имя, отчество)

Тема курсового проекта: “Бинарная классификация”
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____

Источник тематики (кафедра, предприятие, НИР) _____

Задание решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на __35___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____

Дата выдачи задания «__ » _____ 202_ г.

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

Студент _____ Абраменков Г.Г.
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Задание

В данной курсовой работе необходимо предпринять следующие шаги:

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 постройение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

Введение

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

Курсовая работа опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работ по дисциплине.

В рамках данной курсовой работы необходимо применить навыки, полученные в течение курса «Технологии машинного обучения», и обосновать полученные результаты.

Основная часть

Постановка задачи

В данной курсовой работе ставится задача определения принадлежности звезды к классу пульсаров по различным параметрам с помощью методов машинного обучения: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest". С помощью различных метрик выбор метода, который наиболее эффективно и качественно определяет значение целевого признака.

Решение поставленной задачи

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**

Описание выбранного датасета

Выбранный набор данных описывает выборку из 600 песен, собранных с популярного музыкального сервера Spotify.
Однако данная подборка была найдена на сервисе Kaggle. Сервис организован, как публичная веб-платформа, на которой пользователи и организации могут публиковать наборы данных, исследовать и создавать модели, взаимодействовать с другими специалистами по данным и инженерами по машинному обучению, организовывать конкурсы по исследованию данных и участвовать в них. В системе размещены наборы открытых данных, предоставляются облачные инструменты для обработки данных и машинного обучения

В рассматриваемом примере будем решать задачу классификации:

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

Импорт библиотек

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold, RepeatedKFold, ShuffleSplit, StratifiedKFold
from sklearn.model_selection import learning_curve, validation_curve

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Загрузка данных.

Загрузим файл датасета в помощью библиотеки Pandas.

```
#Загружаем данные и выводим первые 5 строк
data=pd.read_csv('data.csv', sep=",")
data.head()
```

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

Основные характеристики датасета

genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	key	mode
21st century classical	0.983000	0.218500	1.496130e+05	0.018350	0.874000	0.112800	-37.256000	0.038750	69.526500	0.062900	40.500000	1	1
432hz	0.485070	0.330000	1.059273e+06	0.463084	0.480393	0.118862	-17.099000	0.086288	125.227125	0.217675	52.125000	6	1
[]	0.686395	0.516830	2.305397e+05	0.397491	0.202883	0.221324	-12.773475	0.109871	111.933224	0.513905	21.556669	7	1
a cappella	0.666036	0.576732	1.961439e+05	0.334535	0.028486	0.128292	-13.011177	0.106782	112.461108	0.502521	38.786415	11	1
abstract	0.352395	0.489100	3.429772e+05	0.509300	0.788033	0.122317	-13.812100	0.044157	124.176500	0.354130	41.600000	1	1

```
#Размер датасета
data.shape
```

```
(599, 14)
```

```
#Список колонок
data.columns
```

```
Index(['genres', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'popularity', 'key', 'mode'],
      dtype='object')
```

```
#Список колонок с типами данных
data.dtypes
```

```
genres             object
acousticness      float64
danceability      float64
duration_ms       float64
energy            float64
instrumentalness  float64
liveness          float64
loudness          float64
speechiness       float64
tempo             float64
valence           float64
popularity        float64
key                 int64
mode                int64
dtype: object
```

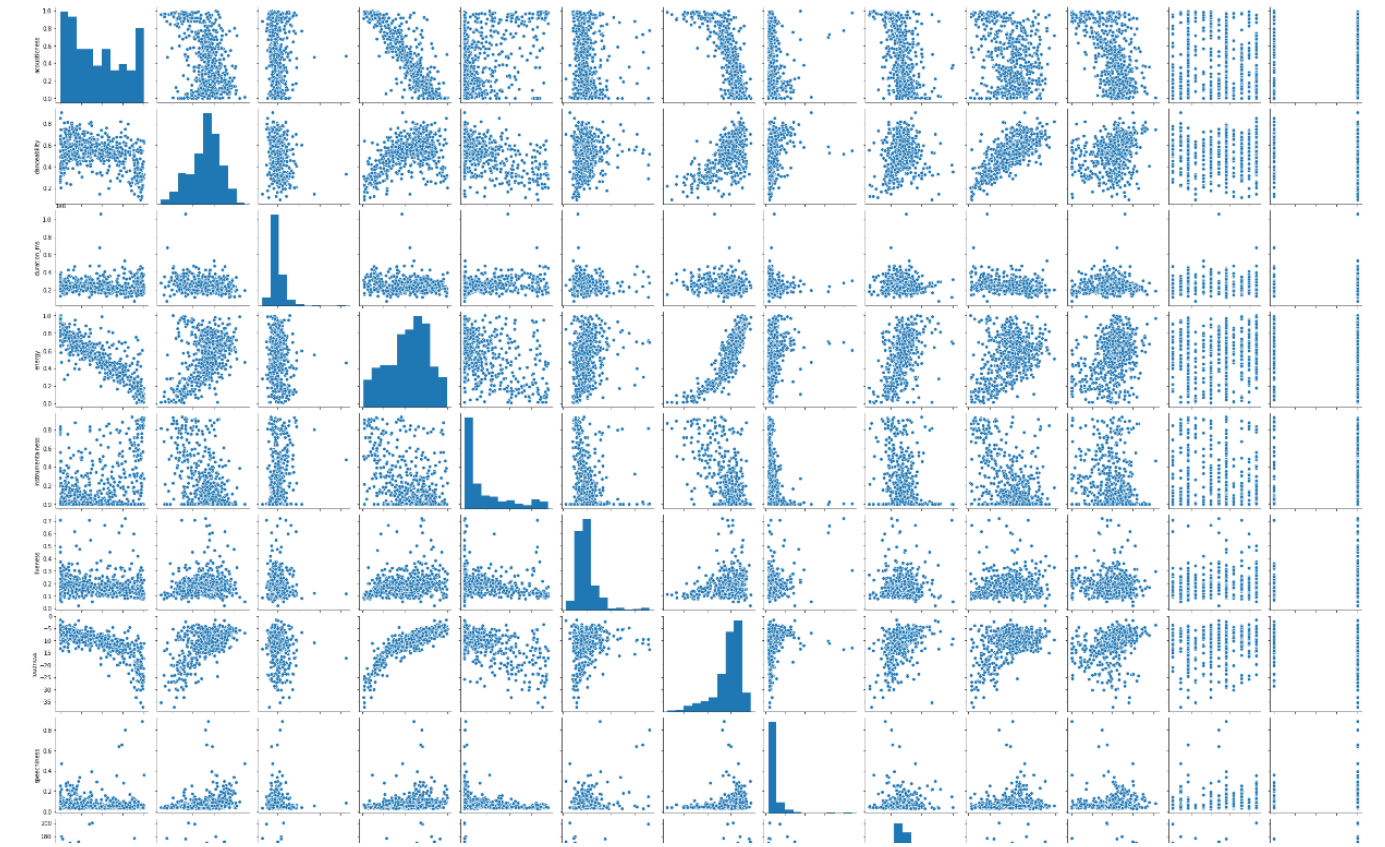
Посмотрим заполненность датасета. Возможно есть пропуски.

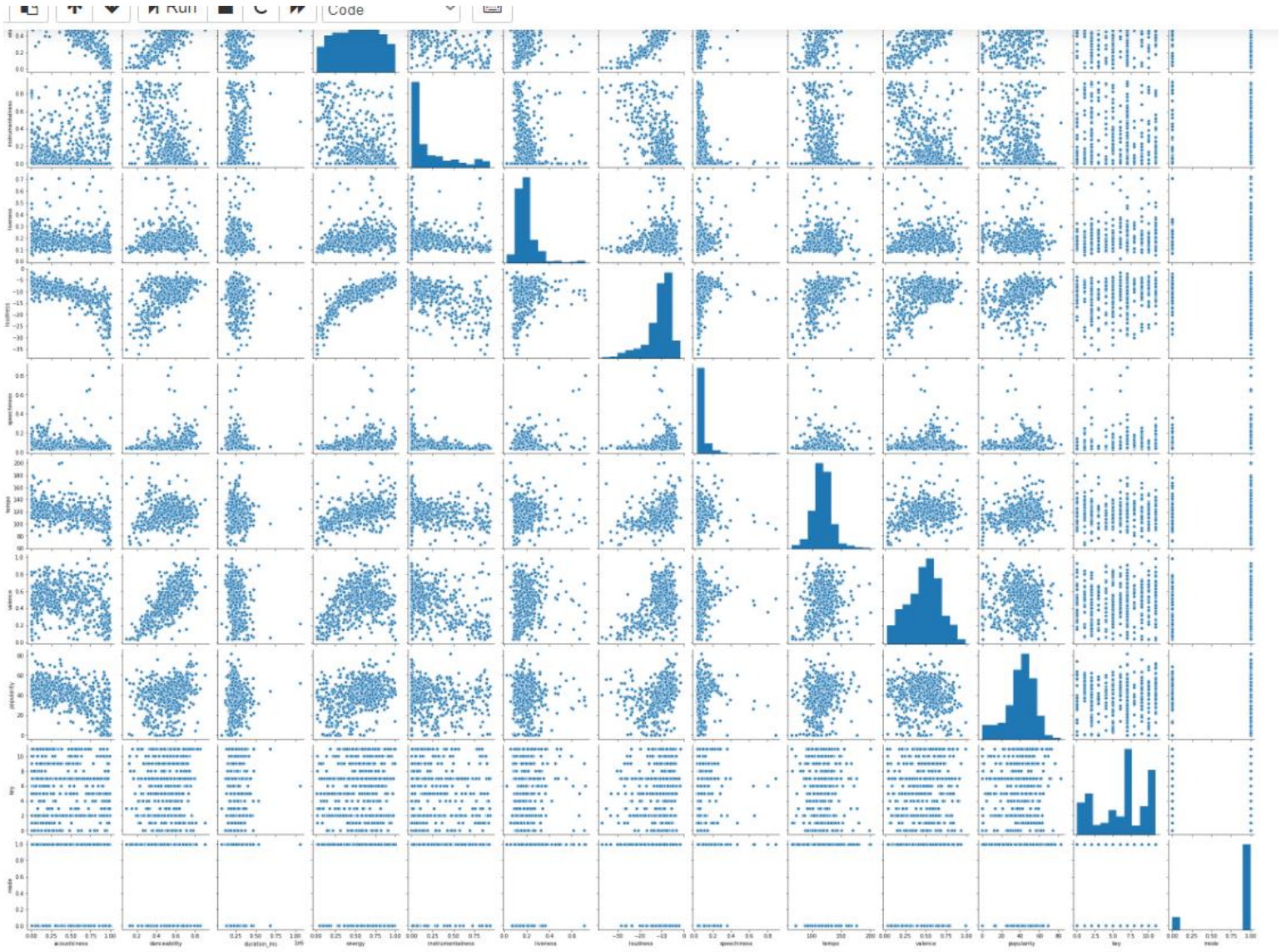
```
In [24]: data.isnull().sum()

Out[24]: genres              0
         acousticness        0
         danceability        0
         duration_ms         0
         energy              0
         instrumentalness    0
         liveness            0
         loudness            0
         speechiness         0
         tempo               0
         valence             0
         popularity          0
         key                 0
         mode                0
         dtype: int64
```

Можем видеть, что пропуски данных отсутствуют

Построим некоторые графики для понимания структуры данных.

```
<seaborn.axisgrid.PairGrid at 0x1b887be7a00>
```

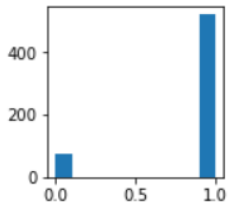



Run
Code
acousticness
danceability
duration_ms
energy
instrumentalness
liveness
loudness
speechiness
tempo
valence
popularity
key
mode

In [11]:
```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
data['mode'].unique()
```

Out[11]:
```
array([1, 0], dtype=int64)
```

In [12]:
```
# Оценим дисбаланс классов для целевого признака
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['mode'])
plt.show()
```



In [13]:
```
data['mode'].value_counts()
```

Out[13]:
```
1    522
0     77
Name: mode, dtype: int64
```

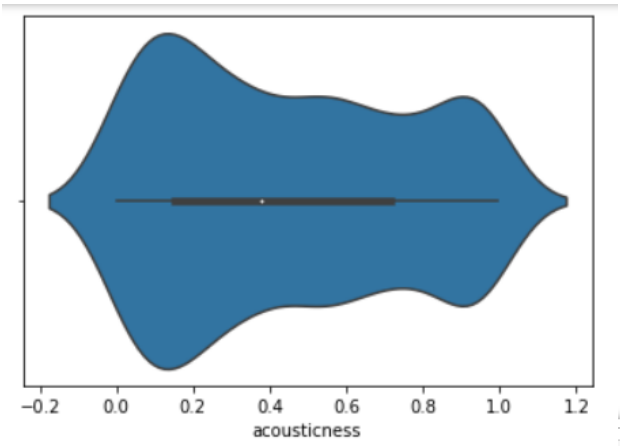
In [14]:
```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['mode'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

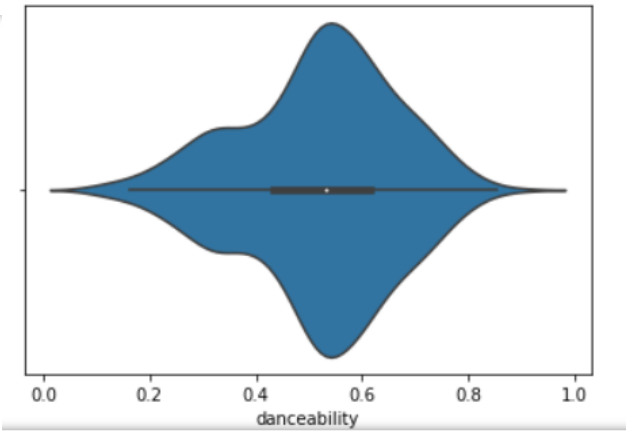
```
Класс 0 составляет 87.15%, а класс 1 составляет 12.85%.
```

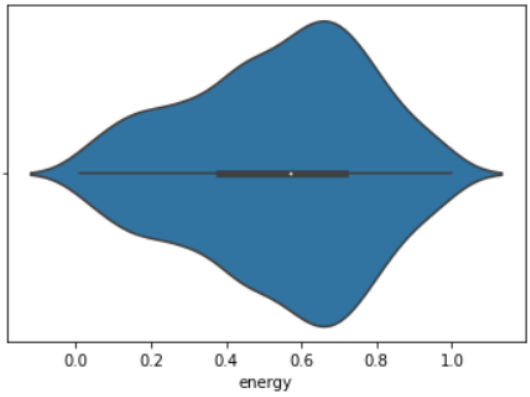
Можем наблюдать явно выраженный дисбаланс классов. Но он является приемлемым.

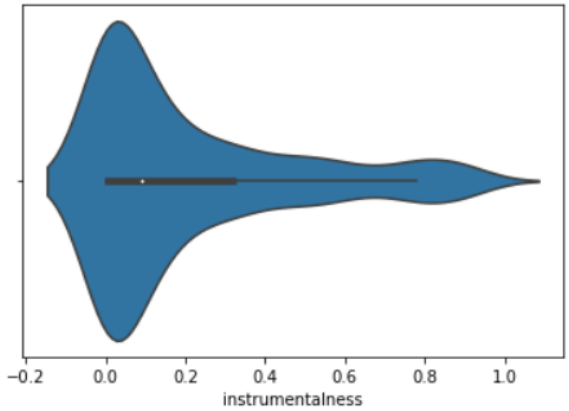
```
Index(['genres', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'popularity', 'key', 'mode'],
      dtype='object')
```

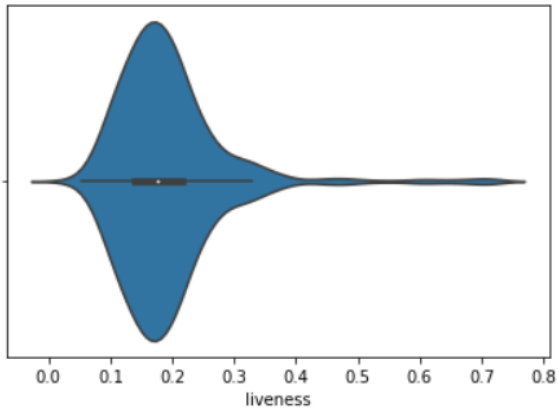
```
# Скрипичные диаграммы для числовых колонок
for col in ['acousticness',
       'danceability',
       'energy',
       'instrumentalness', 'liveness',
       'loudness',
       'speechiness', 'tempo',
       'valence', 'popularity']:
    sns.violinplot(x=data[col])
    plt.show()
```

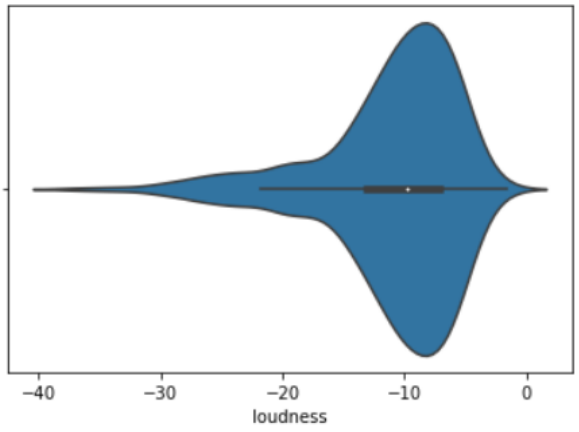

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
acousticness

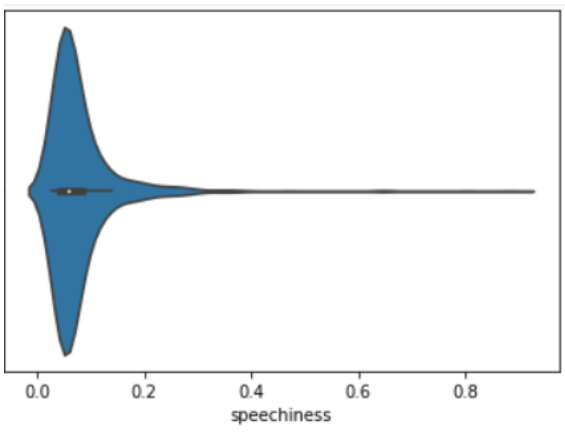

0.0
0.2
0.4
0.6
0.8
1.0
danceability

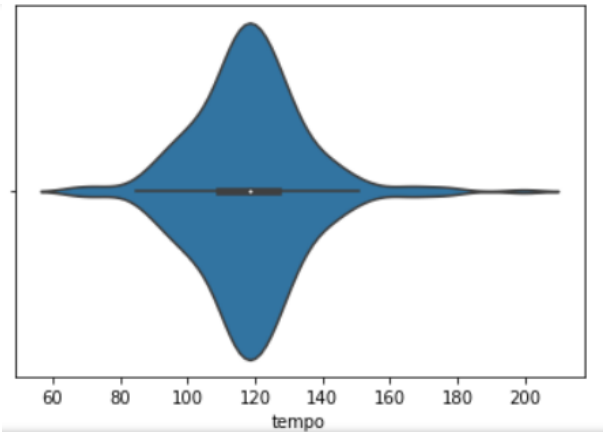

0.0
0.2
0.4
0.6
0.8
1.0
energy

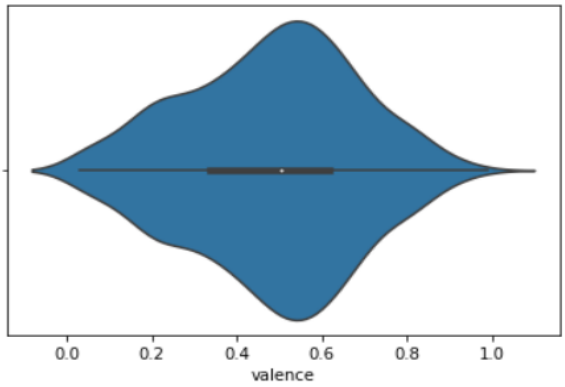

−0.2
0.0
0.2
0.4
0.6
0.8
1.0
instrumentalness

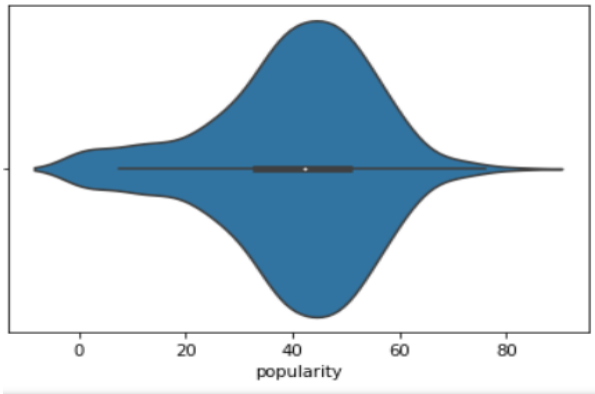

0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
liveness


−40
−30
−20
−10
0
loudness









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

```
: data.dtypes

: genres              object
  acousticness       float64
  danceability       float64
  duration_ms        float64
  energy             float64
  instrumentalness   float64
  liveness           float64
  loudness           float64
  speechiness        float64
  tempo              float64
  valence            float64
  popularity         float64
  key                  int64
  mode                 int64
  dtype: object
```

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирования не требуется. Исключением является признак mode, но в представленном датасете он уже закодирован на основе подхода LabelEncoding.

Вспомогательные признаки для улучшения качества моделей строить не будем.

Выполним масштабирование данных.

```
data.head()
```

	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	valence	popularity	key	m
0	21st century classical	0.983000	0.218500	1.496130e+05	0.018350	0.874000	0.112800	-37.256000	0.038750	69.526500	0.062900	40.500000	1	
1	432hz	0.485070	0.330000	1.059273e+06	0.463084	0.480393	0.118862	-17.099000	0.086288	125.227125	0.217675	52.125000	6	
2	[]	0.686395	0.516830	2.305397e+05	0.397491	0.202883	0.221324	-12.773475	0.109871	111.933224	0.513905	21.556669	7	
3	a cappella	0.666036	0.576732	1.961439e+05	0.334535	0.028486	0.128292	-13.011177	0.106782	112.461108	0.502521	38.786415	11	
4	abstract	0.352395	0.489100	3.429772e+05	0.509300	0.788033	0.122317	-13.812100	0.044157	124.176500	0.354130	41.600000	1	

```
# Числовые колонки для масштабирования
scale_cols = ['acousticness','danceability', 'energy', 'instrumentalness', 'liveness','loudness', 'speechiness', 'tempo',
'valence', 'popularity']
```

```
from sklearn.preprocessing import MinMaxScaler
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

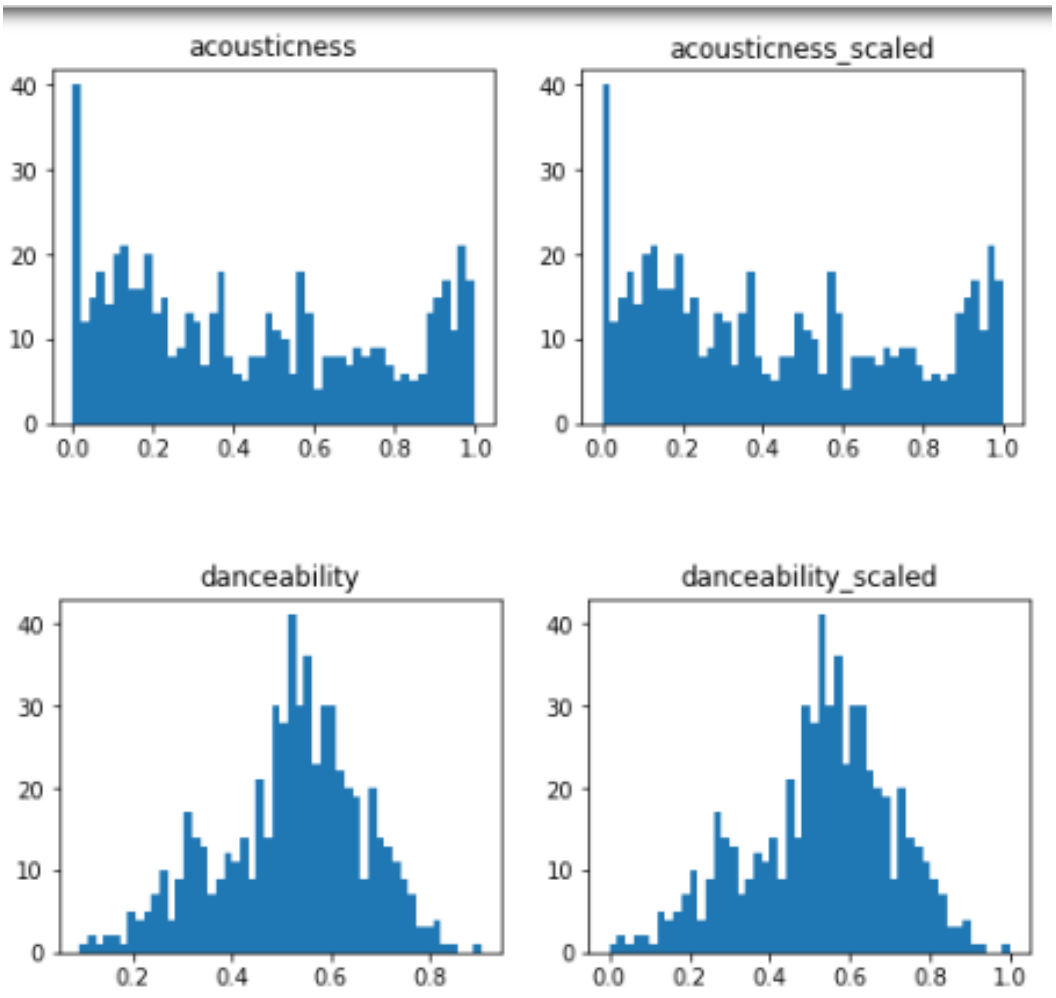
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

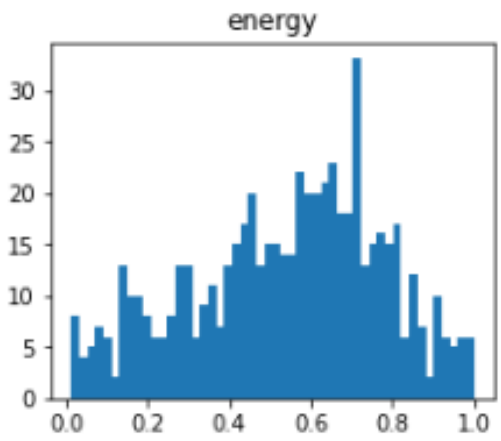
```
data.head()
```

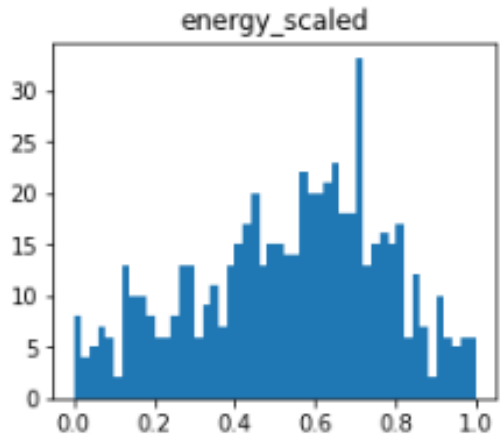
	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	...	acousticness_scaled	dan
0	21st century classical	0.983000	0.218500	1.496130e+05	0.018350	0.874000	0.112800	-37.256000	0.038750	69.526500	...	0.986948	
1	432hz	0.485070	0.330000	1.059273e+06	0.463084	0.480393	0.118862	-17.099000	0.086288	125.227125	...	0.487016	
2	[]	0.686395	0.516830	2.305397e+05	0.397491	0.202883	0.221324	-12.773475	0.109871	111.933224	...	0.689151	
3	a cappella	0.666036	0.576732	1.961439e+05	0.334535	0.028486	0.128292	-13.011177	0.106782	112.461108	...	0.668710	
4	abstract	0.352395	0.489100	3.429772e+05	0.509300	0.788033	0.122317	-13.812100	0.044157	124.176500	...	0.353808	

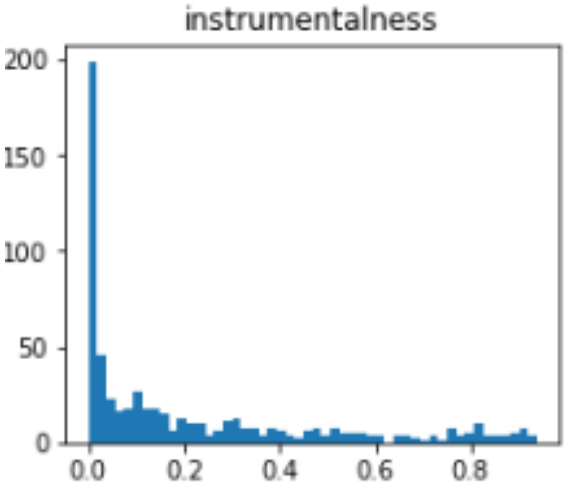
5 rows × 24 columns

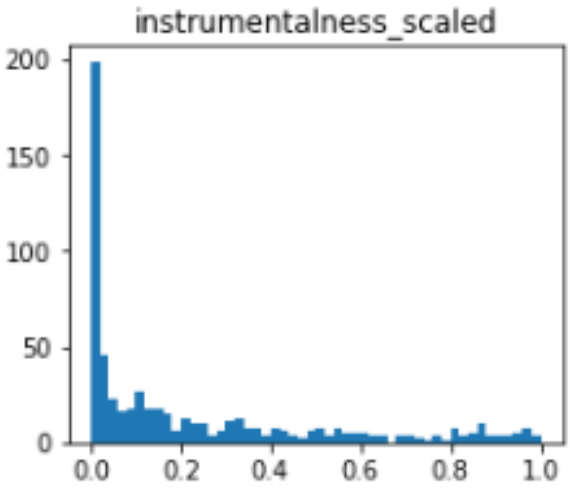
```
: # Проверим, что масштабирование не повлияло на распределение данных
for col in scale_cols:
    col_scaled = col + '_scaled'

    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()
```

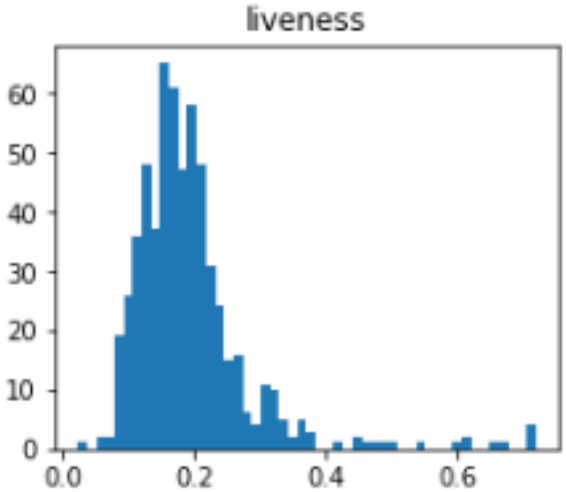


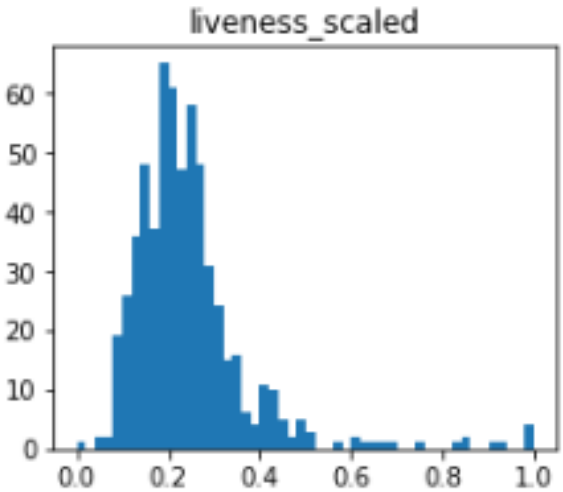

energy
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0

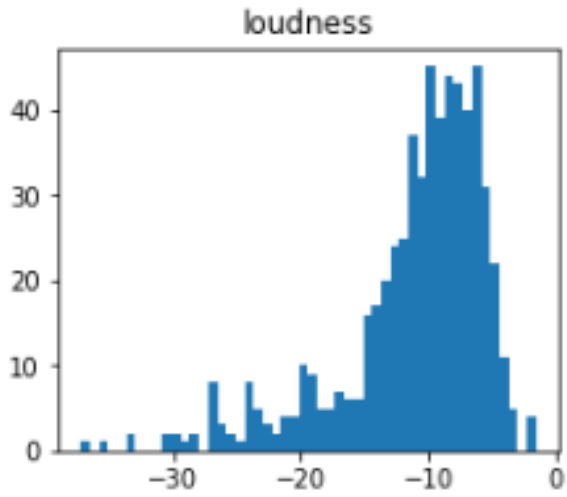

energy_scaled
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0

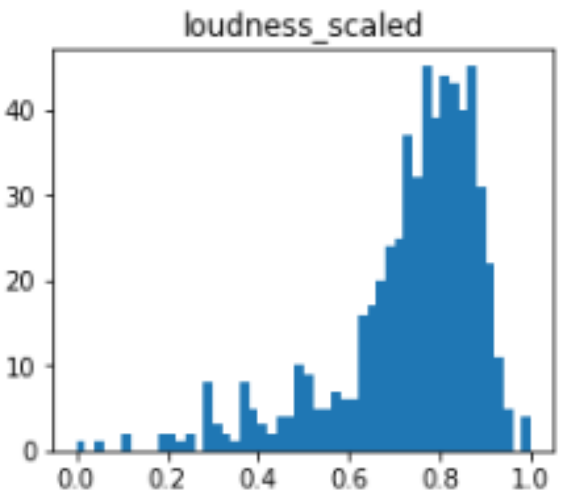

instrumentalness
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8

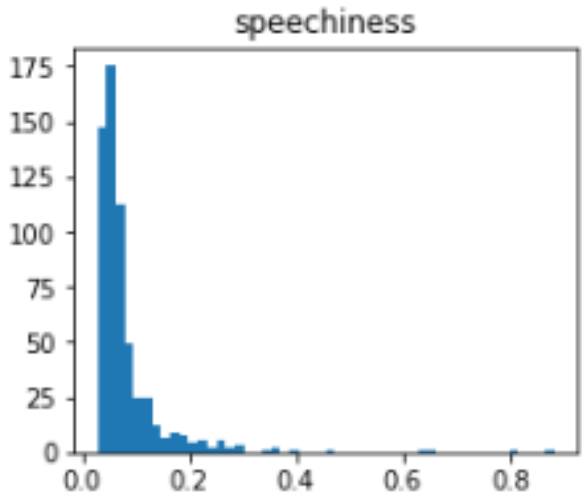

instrumentalness_scaled
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8
1.0

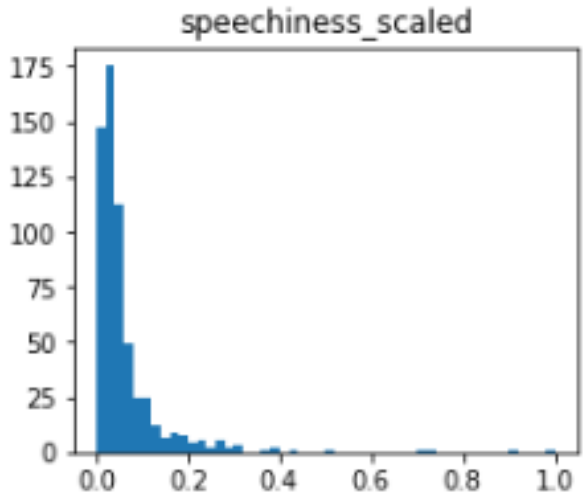

liveness
0
10
20
30
40
50
60
0.0
0.2
0.4
0.6

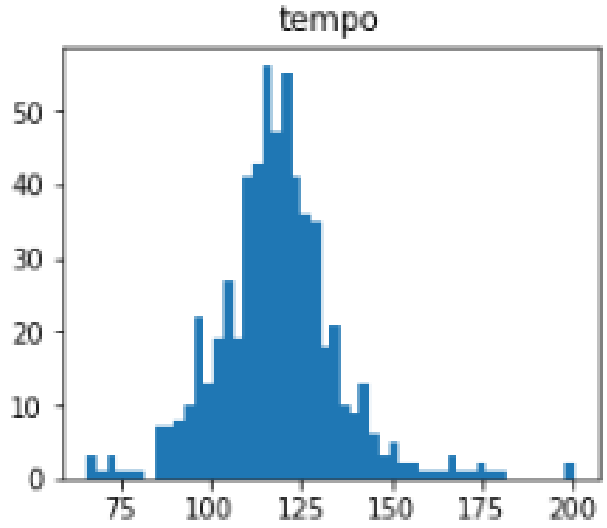

liveness_scaled
0
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
1.0

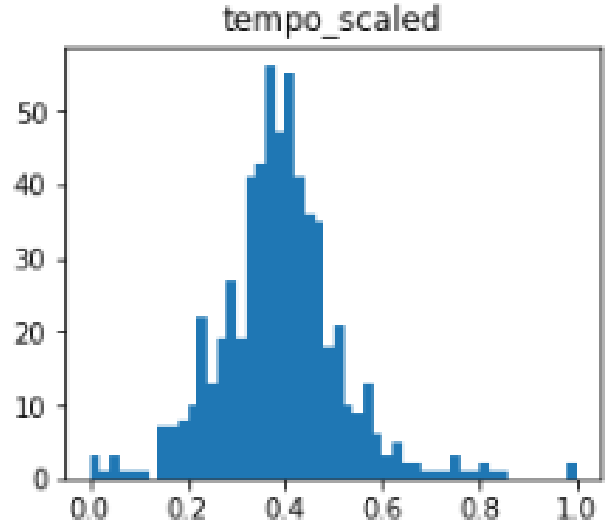

loudness
40
30
20
10
0
−30
−20
−10
0

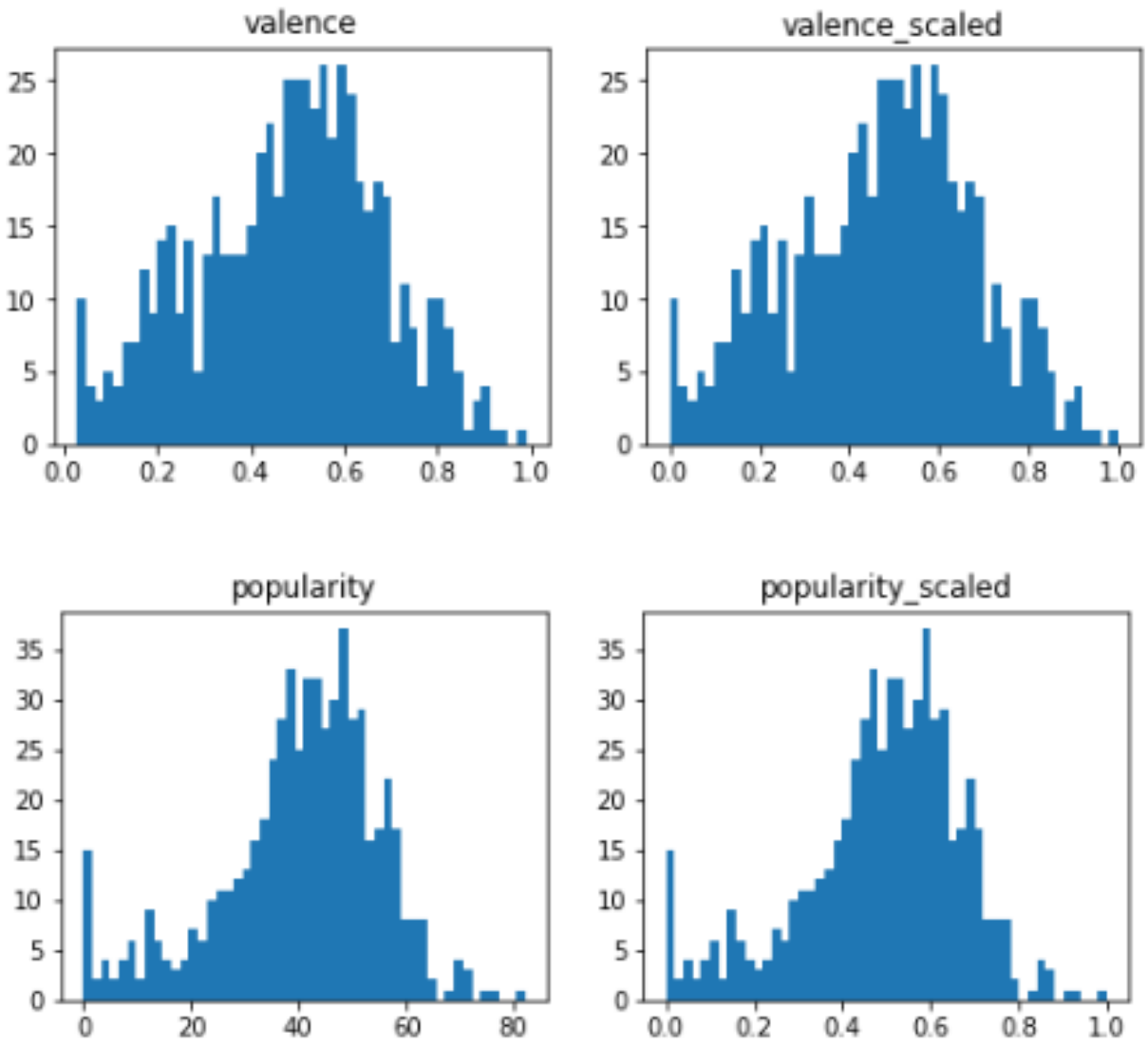

loudness_scaled
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0


speechiness
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8


speechiness_scaled
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0


tempo
50
40
30
20
10
0
75
100
125
150
175
200


tempo_scaled
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0



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

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

```
: corr_cols_1 = scale_cols + ['mode']
  corr_cols_1
```

```
: ['acousticness',
   'danceability',
   'energy',
   'instrumentalness',
   'liveness',
   'loudness',
   'speechiness',
   'tempo',
   'valence',
   'popularity',
   'mode']
```

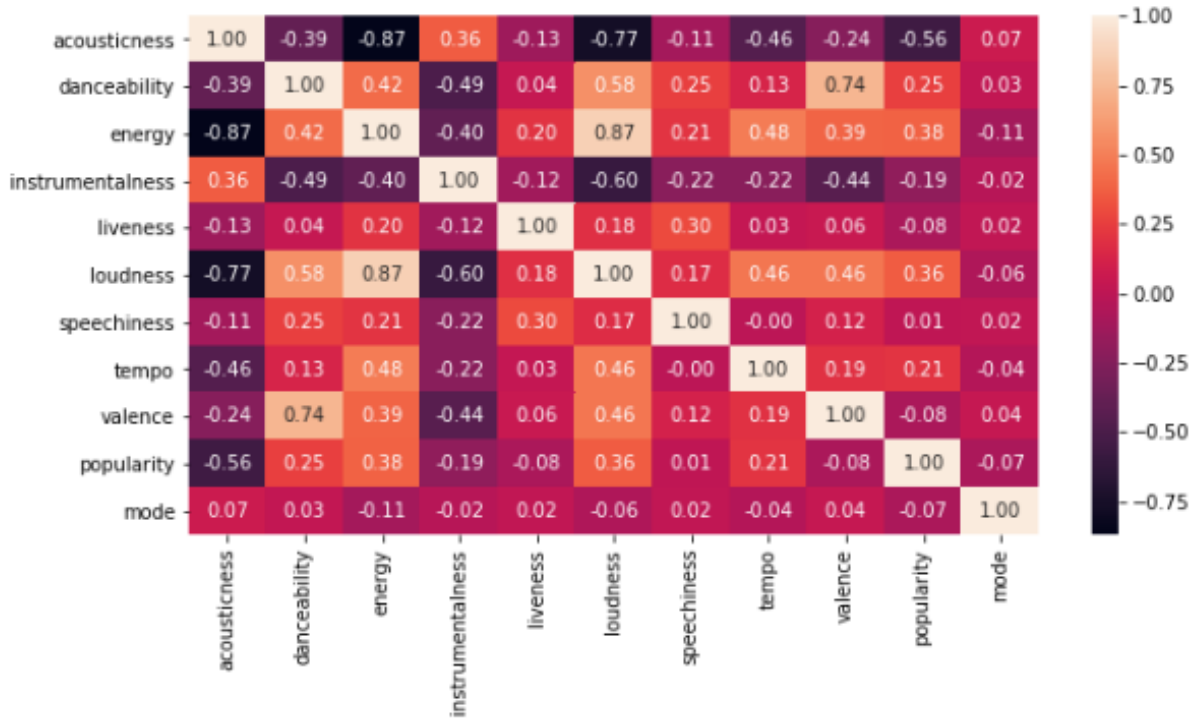
```
: scale_cols_postfix = [x+'_scaled' for x in scale_cols]
  corr_cols_2 = scale_cols_postfix + ['mode']
  corr_cols_2
```

```
: ['acousticness_scaled',
   'danceability_scaled',
   'energy_scaled',
   'instrumentalness_scaled',
   'liveness_scaled',
   'loudness_scaled',
   'speechiness_scaled',
   'tempo_scaled',
   'valence_scaled',
   'popularity_scaled',
   'mode']
```

```
: fig, ax = plt.subplots(figsize=(10,5))
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b8bb7cb4c0>
```




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

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b8bc2a5a30>
```

На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабированных данных совпадают.

2. Достаточно большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

5. Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

1. Метрика precision:

$$precision = \frac{TP}{TP+FP}$$

Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Используется функция *precision_score.*

2. Метрика recall (полнота):

$$recall = \frac{TP}{TP+FN}$$

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция *recall_score*.

3. Метрика F1-мера

Для того, чтобы объединить precision и recall в единую метрику используется F_β-мера, которая вычисляется как среднее гармоническое от precision и recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{precision + recall}$$

где β определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при β=1:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Для вычисления используется функция *f1_score*.

4. Метрика ROC AUC

Используется для оценки качества бинарной классификации. Основана на вычислении следующих характеристик:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

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.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция *roc_auc_score*.

Сохранение и визуализация метрик

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

```
]: #Сохранение и визуализация метрик
```

```
]: 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)].
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()
```

6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия

Метод, используемый для решения задачи бинарной классификации.
Метод выдает вероятность принадлежности объекта к нулевому/единичному классам.
Используется класс *LogisticRegression*.

- Машина опорных векторов

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам

гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

Для решения задачи классификации используем класс:
SVC - основной классификатор на основе SVM. Поддерживает различные ядра.

- Решающее дерево

Для текущего выбранного признака (колонки) из N признаков построить все варианты ветвления по значениям (для категориальных признаков) или по диапазонам значений (для числовых признаков).
Если подвыборке соответствует единственное значение целевого признака, то в дерево добавляется терминальный лист, который соответствует предсказанному значению.
Если в подвыборке больше одного значения целевого признака, то предыдущие пункты выполняются рекурсивно для подвыборки.

Для решения задачи классификации используется класс *DecisionTreeClassifier*.

- Случайный лес (ансамблевая)

Случайный лес можно рассматривать как алгоритмом бэггинга над решающими деревьями.

Но при этом каждое решающее дерево строится на случайно выбранном подмножестве признаков. Эта особенность называется "feature bagging" и основана на методе случайных подпространств.

Случайный лес для задача классификации реализуется в scikit-learn с помощью класса *RandomForestClassifier*.

Задание параметра n_jobs=-1 распараллеливает алгоритм на максимально возможное количество процессоров.

- Градиентный бустинг (ансамблевая)

В отличие от методов бэггинга и случайного леса, которые ориентированы прежде всего на минимизизацию дисперсии (Variance), методы бустинга ориентированы прежде всего на минимизацию смещения (Bias) и, отчасти, на минимизизацию дисперсии.

Исторически первым полноценным алгоритмом бустинга считается алгоритм AdaBoost.

AdaBoost реализуется в scikit-learn с помощью класса *AdaBoostClassifier* для задач классификации.

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

Имеем масштабированные данные:

```
target = data['mode']
data = data.drop('mode', axis = 1)
```

```
data.columns
```

```
Index(['genres', 'acousticness', 'danceability', 'duration_ms', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'popularity', 'key', 'acousticness_scaled',
       'danceability_scaled', 'energy_scaled', 'instrumentalness_scaled',
       'liveness_scaled', 'loudness_scaled', 'speechiness_scaled',
       'tempo_scaled', 'valence_scaled', 'popularity_scaled'],
      dtype='object')
```

```
data.head()
```

	genres	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	tempo	...	acousticness_scaled	dan
0	21st century classical	0.983000	0.218500	1.496130e+05	0.018350	0.874000	0.112800	-37.256000	0.038750	69.526500	...	0.986948	
1	432hz	0.485070	0.330000	1.059273e+06	0.463084	0.480393	0.118862	-17.099000	0.086288	125.227125	...	0.487016	
2	[]	0.686395	0.516830	2.305397e+05	0.397491	0.202883	0.221324	-12.773475	0.109871	111.933224	...	0.689151	
3	a cappella	0.666036	0.576732	1.961439e+05	0.334535	0.028486	0.128292	-13.011177	0.106782	112.461108	...	0.668710	
4	abstract	0.352395	0.489100	3.429772e+05	0.509300	0.788033	0.122317	-13.812100	0.044157	124.176500	...	0.353808	

На основе масштабированных данных выделим обучающую и тестовую выборки:

```
# Признаки для задачи классификации
task_clas_cols = ['liveness',
       'danceability',
       'energy',
       'tempo']
```

```
# Выборки для задачи классификации
clas_data = data[task_clas_cols]
```

```
clas_data.head()
```

	liveness	danceability	energy	tempo
0	0.112800	0.218500	0.018350	69.526500
1	0.118862	0.330000	0.463084	125.227125
2	0.221324	0.516830	0.397491	111.933224
3	0.128292	0.576732	0.334535	112.461108
4	0.122317	0.489100	0.509300	124.176500

```
#деление на тестовую и обучающую выборку
clas_X_train, clas_X_test, clas_Y_train, clas_Y_test = train_test_split(
    clas_data, target, test_size=0.2, random_state=1)
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

```
((479, 4), (120, 4), (479,), (120,))
```

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.**

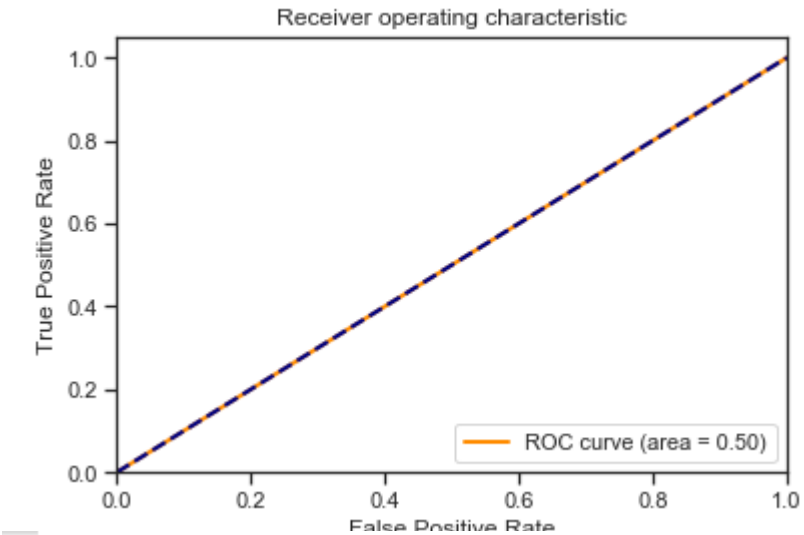
```
from sklearn.linear_model import LogisticRegression
# Модели
clas_models = {'LogR': LogisticRegression(),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

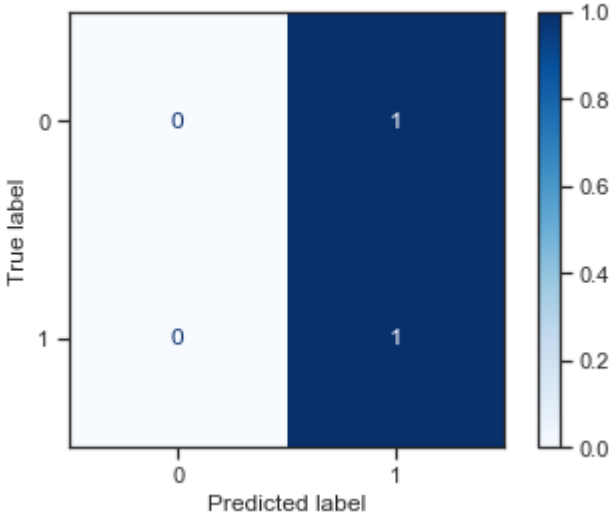
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

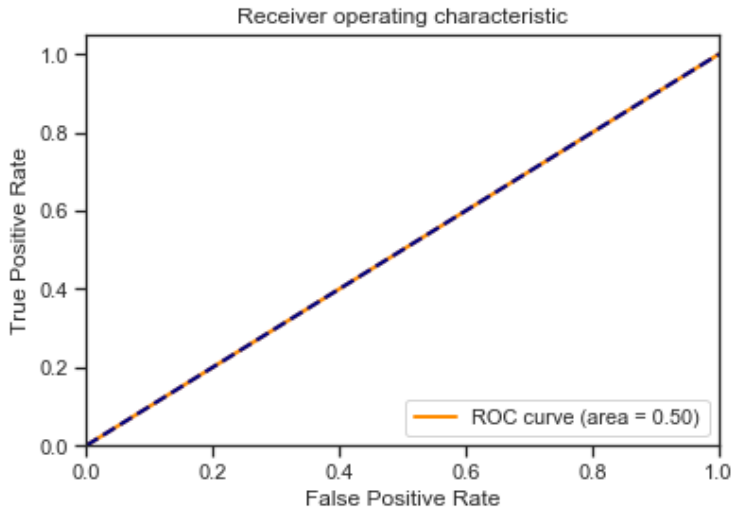
```
def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(clas_X_train, clas_Y_train)
    Y_pred = model.predict(clas_X_test)
    precision = precision_score(clas_Y_test.values, Y_pred)
    recall = recall_score(clas_Y_test.values, Y_pred)
    f1 = f1_score(clas_Y_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y_test.values, Y_pred)

    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    clasMetricLogger.add('roc_auc', model_name, roc_auc)

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

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

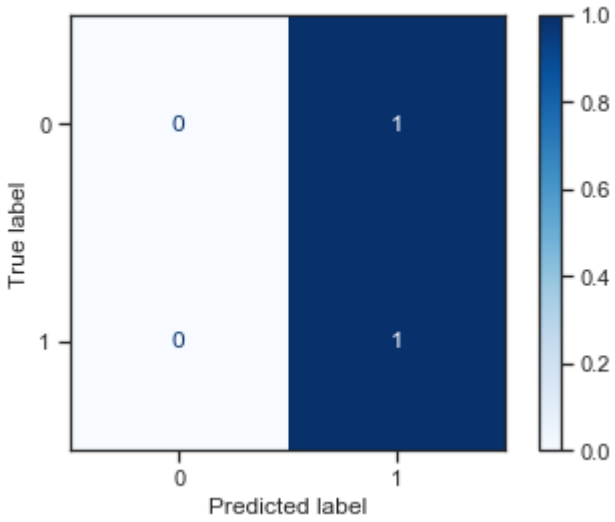
```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

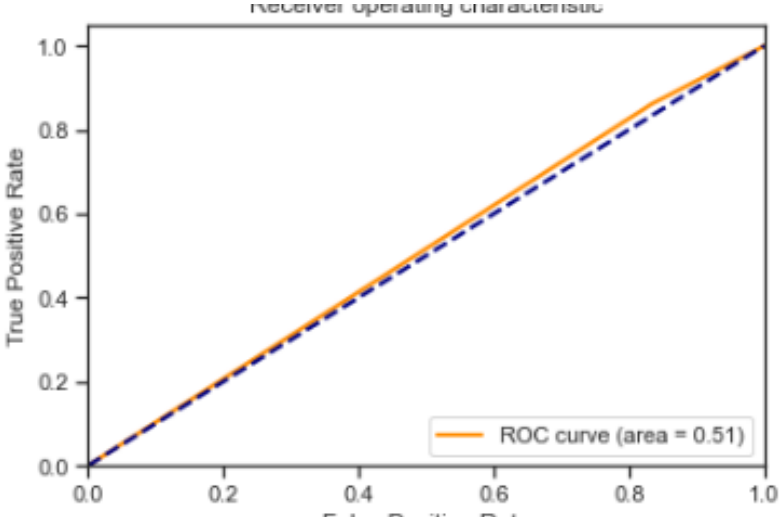
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

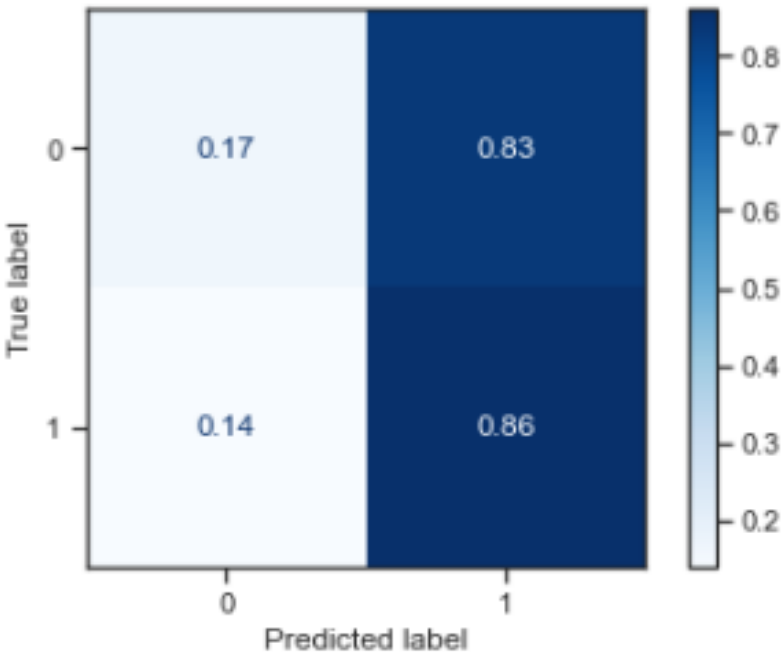

Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.50)
0.0
0.2
0.4
0.6
0.8
1.0

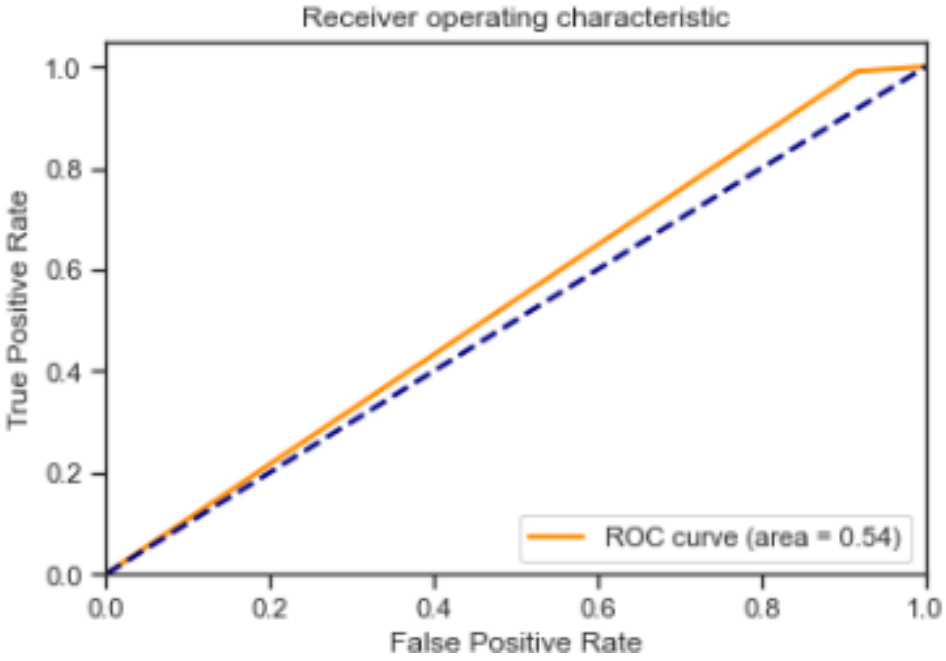

True label
Predicted label
0
1
0
1
0
1
0.0
0.2
0.4
0.6
0.8
1.0

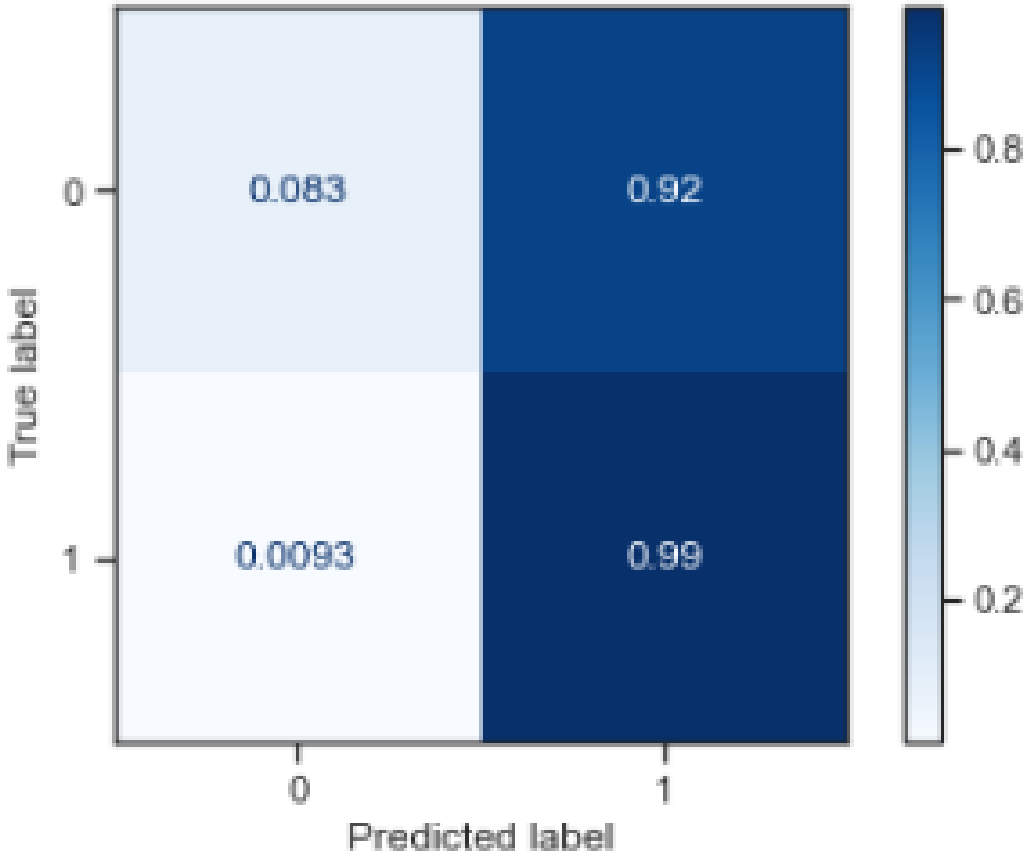

Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.50)
0.0
0.2
0.4
0.6
0.8
1.0

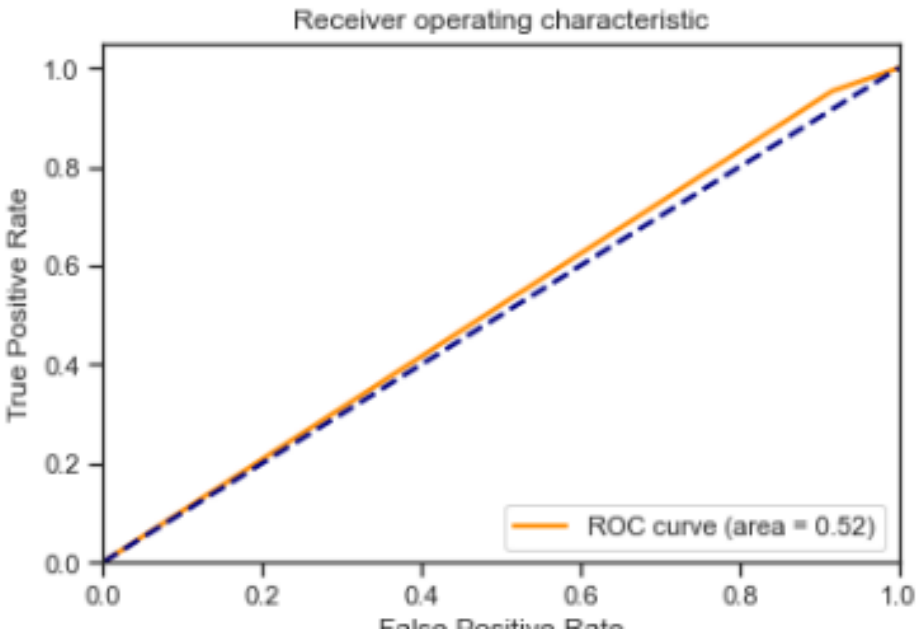

0
1
True label
0
1
0
1
0
1
Predicted label
1.0
0.8
0.6
0.4
0.2
0.0

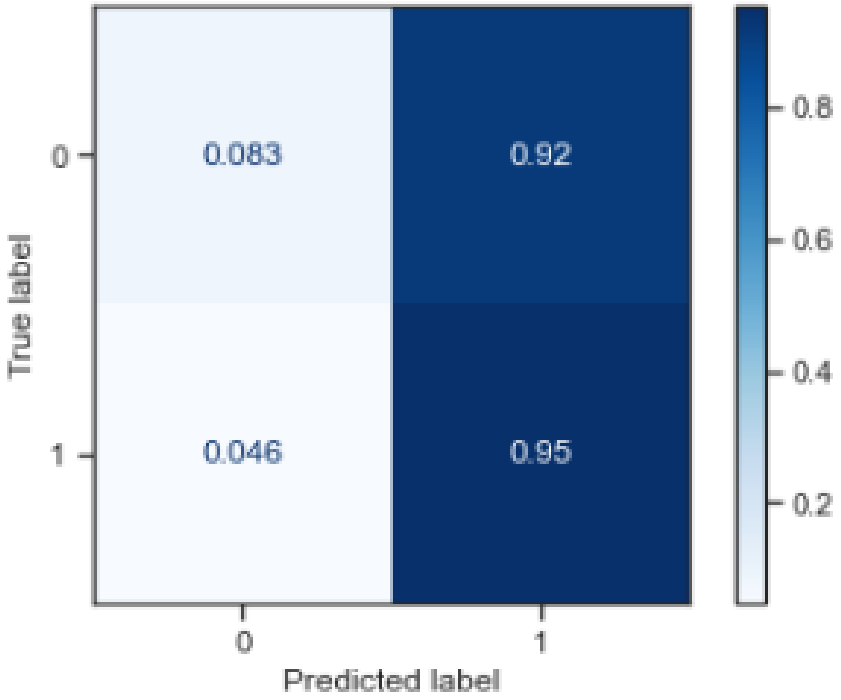

True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.51)


0
1
True label
0.17
0.83
0.14
0.86
0
1
Predicted label
0.8
0.7
0.6
0.5
0.4
0.3
0.2


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.54)
0.0
0.2
0.4
0.6
0.8
1.0


0.083
0.92
0.0093
0.99
True label
Predicted label
0
1
0.8
0.6
0.4
0.2





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.**

Кросс-валидация

```
clas_X_train.shape
```

```
(479, 4)
```

```
#Кроссвалидация
scores_log = cross_val_score(LogisticRegression(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_log, np.mean(scores_log)
```

```
(array([0.8625    , 0.86610879]), 0.8643043933054393)
```

```
scores_svc = cross_val_score(SVC(gamma='auto'),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_svc, np.mean(scores_svc)
```

```
(array([0.85416667, 0.87029289]), 0.8622297768479776)
```

```
scores_tree = cross_val_score(DecisionTreeClassifier(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_tree, np.mean(scores_tree)
```

```
(array([0.79166667, 0.79916318]), 0.7954149232914923)
```

```
scores_rand_tree = cross_val_score(RandomForestClassifier(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_rand_tree, np.mean(scores_rand_tree)
```

```
(array([0.8625    , 0.83682008]), 0.8496600418410042)
```

```
scores_boost = cross_val_score(GradientBoostingClassifier(),
                             clas_X_train, clas_Y_train, cv=2)
# Значение метрики accuracy для 2 фолдов
scores_boost, np.mean(scores_boost)
```

Подбор гиперпараметров:

```
parameters = {'gamma':[140,130,120,110,100,70,50]}
clf_gs_svm_svc = GridSearchCV(SVC(), parameters, cv=5, scoring='accuracy')
clf_gs_svm_svc.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'gamma': [140, 130, 120, 110, 100, 70, 50]},
             scoring='accuracy')
```

```
# Лучшая модель
clf_gs_svm_svc.best_estimator_
```

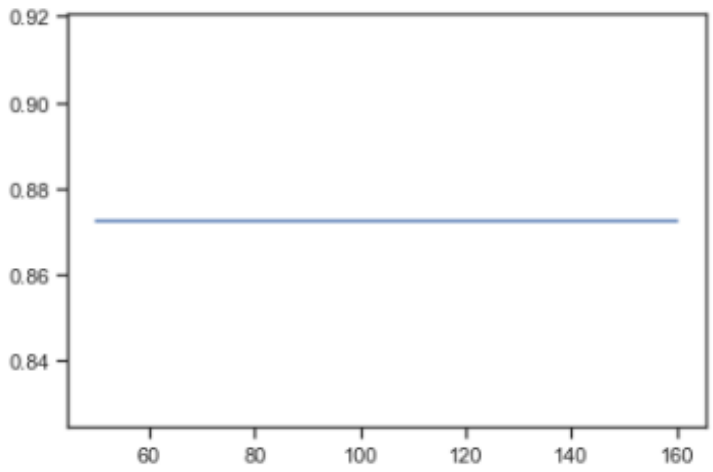
```
SVC(gamma=140)
```

```
# Лучшее значение параметров
clf_gs_svm_svc.best_params_
```

```
{'gamma': 140}
```

```
# Изменение качества на тестовой выборке в зависимости от параметра
n_range = np.array([160,130,100,70,65,60,50])
plt.plot(n_range, clf_gs_svm_svc.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x1b8bb34b5b0>]
```



```
parameters = {'max_depth':[20,15,10,6,5,4,3], 'min_samples_split':[10,8,6,5,4,3,2]}
clf_gs_decision_tree = GridSearchCV(DecisionTreeClassifier(), parameters, cv=5, scoring='accuracy')
clf_gs_decision_tree.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [20, 15, 10, 6, 5, 4, 3],
                         'min_samples_split': [10, 8, 6, 5, 4, 3, 2]},
             scoring='accuracy')
```

```
# Лучшая модель
clf_gs_decision_tree.best_estimator_
```

```
DecisionTreeClassifier(max_depth=3, min_samples_split=10)
```

```
# Лучшее значение параметров
clf_gs_decision_tree.best_params_
```

```
{'max_depth': 3, 'min_samples_split': 10}
```

```
parameters_random_forest = {'n_estimators':[1, 3, 5, 6, 7, 8, 10],
                            'max_depth':[1, 3, 5, 6, 7, 8, 10],
                            'random_state':[0, 2, 4, 6, 8, 10, 15]}
best_random_forest = GridSearchCV(RandomForestClassifier(), parameters_random_forest, cv=5, scoring='accuracy')
best_random_forest.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 3, 5, 6, 7, 8, 10],
                         'n_estimators': [1, 3, 5, 6, 7, 8, 10],
                         'random_state': [0, 2, 4, 6, 8, 10, 15]},
             scoring='accuracy')
```

```
# Лучшая модель
best_random_forest.best_estimator_
```

```
RandomForestClassifier(max_depth=5, n_estimators=10, random_state=8)
```

```
best_random_forest.best_params_
```

```
{'max_depth': 5, 'n_estimators': 10, 'random_state': 8}
```

```
parameters_gradient_boosting = {'n_estimators':[3, 5, 7, 10, 15, 20],
                                'max_depth':[3, 5, 7, 9, 10, 15]}
best_gradient_boosting = GridSearchCV(GradientBoostingClassifier(), parameters_gradient_boosting, cv=5, scoring='accuracy')
best_gradient_boosting.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [3, 5, 7, 9, 10, 15],
                         'n_estimators': [3, 5, 7, 10, 15, 20]},
             scoring='accuracy')
```

```
# Лучшая модель
best_gradient_boosting.best_estimator_
```

```
: GradientBoostingClassifier(n_estimators=7)
```

```
: best_gradient_boosting.best_params_
```

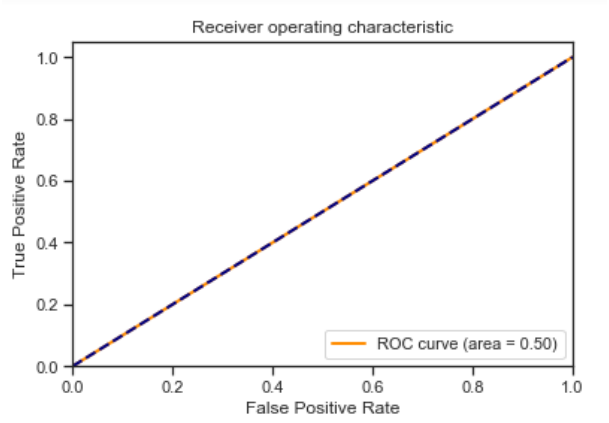
```
: {'max_depth': 3, 'n_estimators': 7}
```

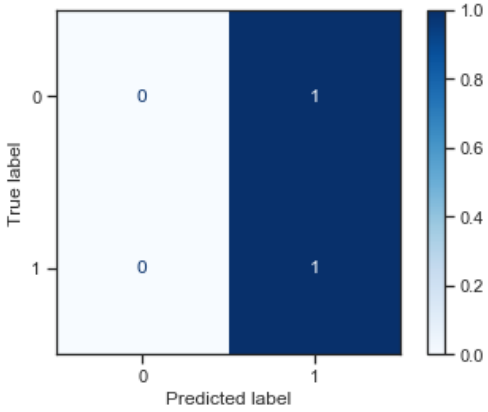
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

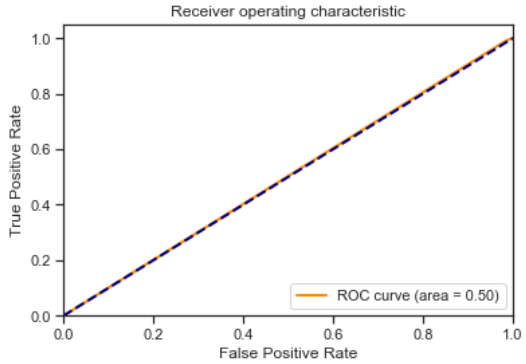
```
#Повторение пункта 8 для найденных оптимальных значений гиперпараметров.
#Сравнение качества полученных моделей с качеством baseline-моделей.
```

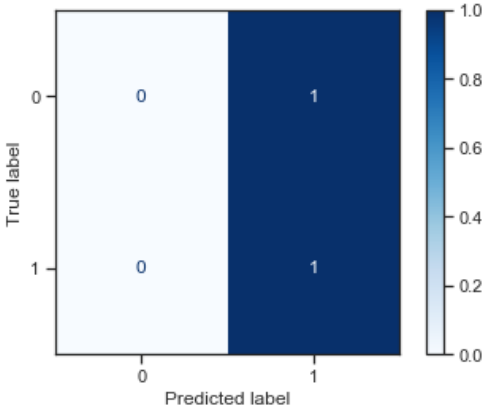
```
# Новые модели с подобранными гиперпараматерами
clas_models_grid = {'LogR':clf_gs_log.best_estimator_,
                    'SVC':clf_gs_svm_svc.best_estimator_,
                    'Tree':clf_gs_decision_tree.best_estimator_,
                    'RF':best_random_forest.best_estimator_,
                    'GB':best_gradient_boosting.best_estimator_}
```

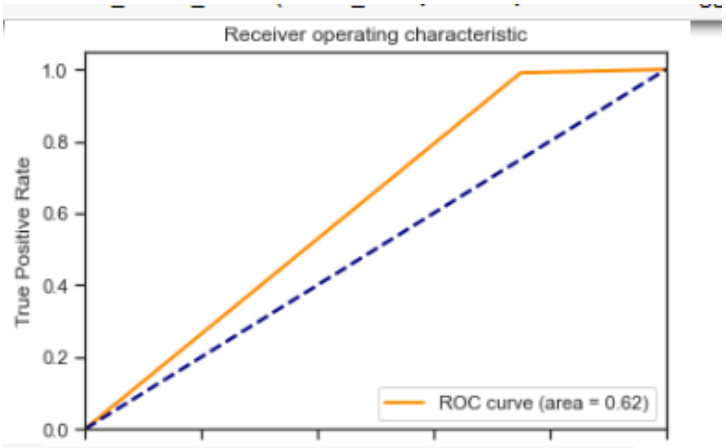
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

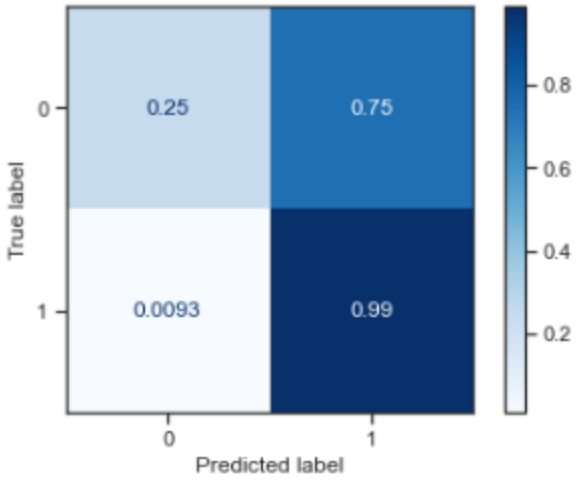


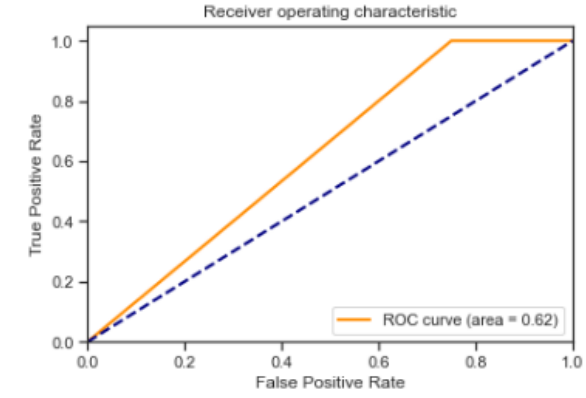


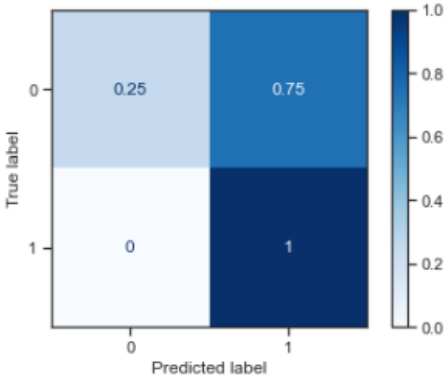


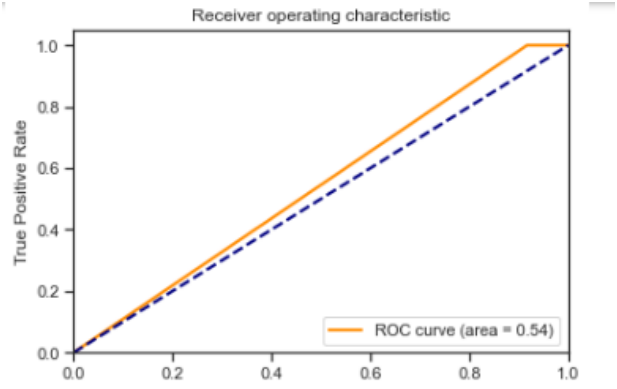


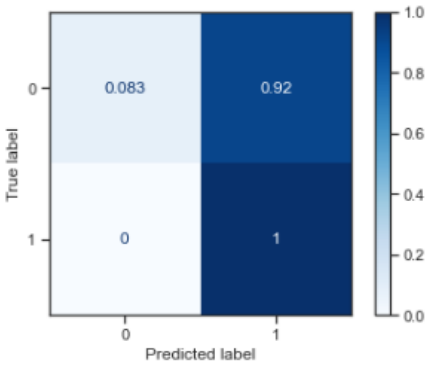

Receiver operating characteristic
True Positive Rate
ROC curve (area = 0.62)


0.25
0.75
0.0093
0.99
True label
Predicted label


Receiver operating characteristic
True Positive Rate
False Positive Rate
ROC curve (area = 0.62)


0.25
0.75
0
1
True label
Predicted label


Receiver operating characteristic
True Positive Rate
ROC curve (area = 0.54)

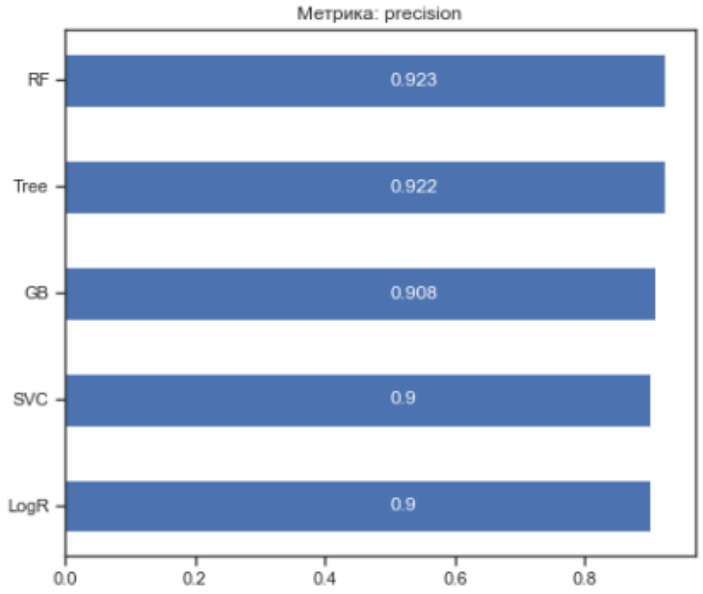

0.083
0.92
0
1
True label
Predicted label

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.

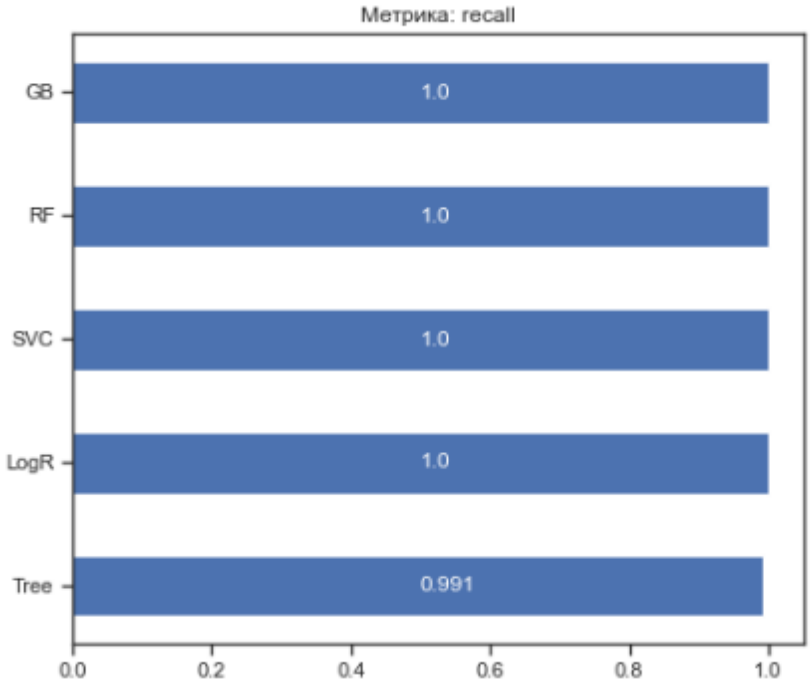
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

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

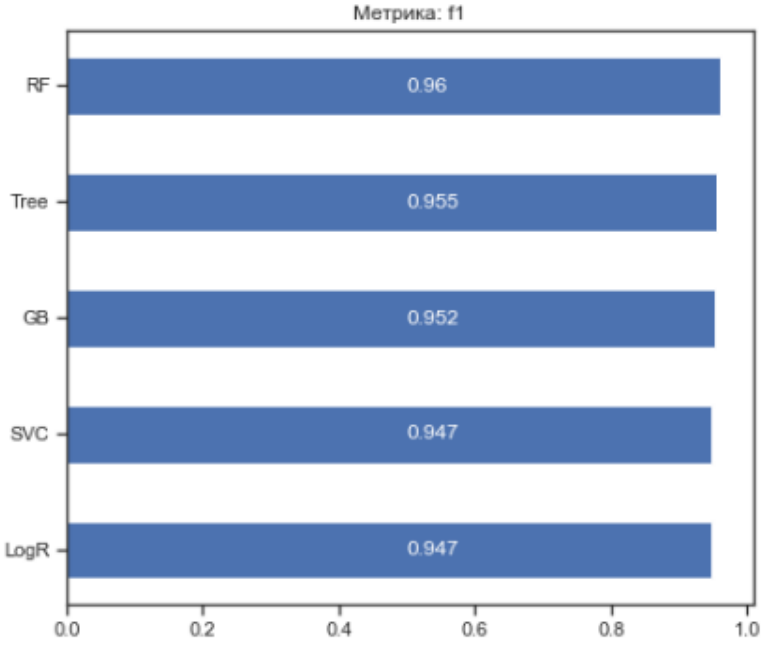
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



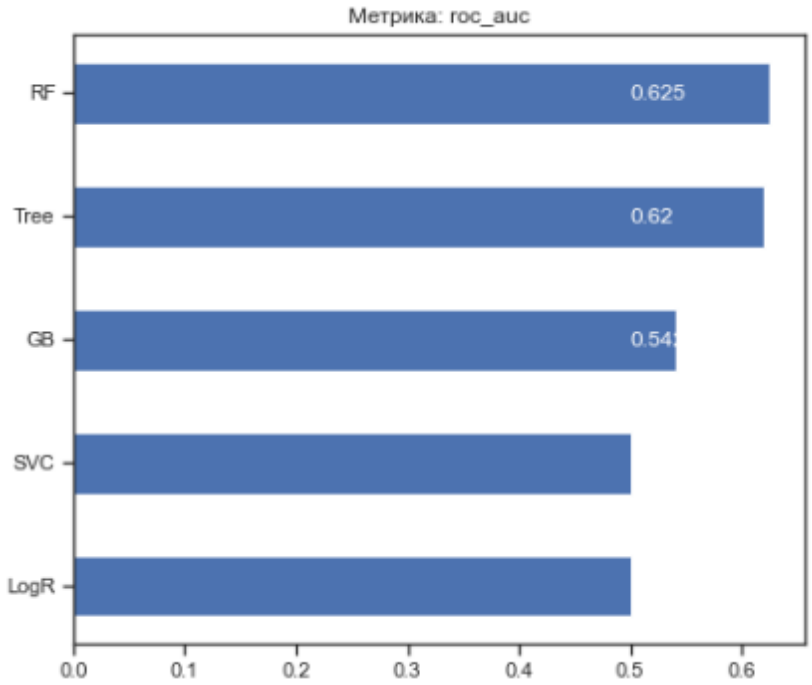
Лучшая модель по метрике precision: Логическая регрессия



Лучшая модель по метрике recall: Метод опорных векторов



Лучшая модель по метрике f1: Логическая регрессия



Лучшая модель по метрике ROC AUC: Gradient Boosting

Вывод: на основании двух метрик из четырех используемых, лучшей оказалась модель Логической регрессии.

Заключение

Из всех рассмотренных алгоритмов: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest" для модели классификации типов песен наиболее эффективным оказался алгоритм случайного леса, т.е. "Random forest".

Список литературы

1. Репозиторий курса "Технологии машинного обучения", бакалавриат, 6 семестр. Лекции по теории машинного обучения. Ю.Е. Гапанюк [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_TMO (дата обращения: 31.05.2020)
2. Spotify Dataset 1921-2020 [Электронный ресурс]. – Электрон. дан. - URL: https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks/data (дата обращения: 01.06.2020)
3. Машинное обучение (часть 1). А.М.Миронов [Электронный ресурс]. – Электрон. дан. - URL: http://www.intsys.msu.ru/staff/mironov/machine_learning_vol1.pdf (дата обращения: 31.05.2020)
4. Scikit learn[Электронный ресурс]. – Электрон. дан. - URL: https://scikit-learn.org/stable/index.html (дата обращения: 01.06.2020)