# Задание

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.

# 1. Выбор и подготовка набора данных

In [78]:

```
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.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score, f1_score, classification_
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
```

В качестве набора данных будем использовать набор данных, состоящий из песен с музыкального сервиса Spotify (https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db)

In [79]:

```
train_data = pd.read_csv('../data/SpotifyFeatures.csv')
train_data.head()
```

Out[79]:

| | genre | artist_name | track_name | track_id | popularity | acousticness | danceab |
|---|---|---|---|---|---|---|---|
| **0** | Movie | Henri Salvador | C'est beau de faire un Show | 0BRjO6ga9RKCKjfDqeFgWV | 0 | 0.611 | 0 |
| **1** | Movie | Martin & les fées | Perdu d'avance (par Gad Elmaleh) | 0BjC1NfoEOOusryehmNudP | 1 | 0.246 | 0 |
| **2** | Movie | Joseph Williams | Don't Let Me Be Lonely Tonight | 0CoSDzoNIKCRs124s9uTVy | 3 | 0.952 | 0 |
| **3** | Movie | Henri Salvador | Dis-moi Monsieur Gordon Cooper | 0Gc6TVm52BwZD07Ki6tIvf | 0 | 0.703 | 0 |
| **4** | Movie | Fabien Nataf | Ouverture | 0IuslXpMROHdEPvSl1fTQK | 4 | 0.950 | 0 |

Размер набора:

In [80]:

```
train_data.shape
```

Out[80]:

```
(232725, 18)
```

Удалим лишние столбцы:

In [81]:

```
train_data = train_data.filter(['genre', 'artist_name', 'track_name', 'energy', 'lou
train_data.head()
```

Out[81]:

| | genre | artist_name | track_name | energy | loudness | speechiness | liveness | popularity | dancea |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Movie | Henri Salvador | C'est beau de faire un Show | 0.910 | -1.828 | 0.0525 | 0.3460 | 0 | 0 |
| **1** | Movie | Martin & les fées | Perdu d'avance (par Gad Elmaleh) | 0.737 | -5.559 | 0.0868 | 0.1510 | 1 | 0 |
| **2** | Movie | Joseph Williams | Don't Let Me Be Lonely Tonight | 0.131 | -13.879 | 0.0362 | 0.1030 | 3 | 0 |
| **3** | Movie | Henri Salvador | Dis-moi Monsieur Gordon Cooper | 0.326 | -12.178 | 0.0395 | 0.0985 | 0 | 0 |
| **4** | Movie | Fabien Nataf | Ouverture | 0.225 | -21.150 | 0.0456 | 0.2020 | 4 | 0 |

Итоговый набор содержит следующие колонки:

- genre - жанр песни
- artist_name - исполнитель песни
- track_name - название песни
- energy - энергичность
- loudness - громкость
- speechiness - показатель количества слов в песне
- liveness - показатель того, что песня была записана при аудитории
- popularity - показатель популярности песни по 10-балльной шкале
- danceability - показатель стабильности песни для танца
- duration_ms - длительность песни (в мс)
- instrumentalness - показатель вокала в песне
- acousticness - акустичность (1.0 - песня в акустической версии)

Новое количество колонок:

In [82]:

```
train_data.shape[1]
```

Out[82]:

```
12
```

Переименуем названия столбцов:

In [83]:

```
train_data.rename(columns={'genre': 'Genre', 'artist_name': 'Artist', 'acousticness'
train_data.head()
```

Out[83]:

| | Genre | Artist | Track | Energy | Loudness | Speechiness | Liveness | Popularity | Danceability |
|---|---|---|---|---|---|---|---|---|---|
| **0** | Movie | Henri Salvador | C'est beau de faire un Show | 0.910 | -1.828 | 0.0525 | 0.3460 | 0 | 0.38 |
| **1** | Movie | Martin & les fées | Perdu d'avance (par Gad Elmaleh) | 0.737 | -5.559 | 0.0868 | 0.1510 | 1 | 0.59( |
| **2** | Movie | Joseph Williams | Don't Let Me Be Lonely Tonight | 0.131 | -13.879 | 0.0362 | 0.1030 | 3 | 0.66 |
| **3** | Movie | Henri Salvador | Dis-moi Monsieur Gordon Cooper | 0.326 | -12.178 | 0.0395 | 0.0985 | 0 | 0.24( |
| **4** | Movie | Fabien Nataf | Ouverture | 0.225 | -21.150 | 0.0456 | 0.2020 | 4 | 0.33 |

# 2. Разведочный анализ

Проверим пропуски:

In [84]:

```
train_data.isnull().sum()
```

Out[84]:

```
Genre                0
Artist               0
Track                0
Energy               0
Loudness             0
Speechiness          0
Liveness             0
Popularity           0
Danceability         0
Duration             0
Instrumentalness     0
Acousticness         0
dtype: int64
```

Как видим, пропуски отсутствуют

Количество уникальных музыкальных жанров:

In [85]:

```
train_data['Genre'].unique().size
```

Out[85]:

```
27
```

Количество песен каждого жанра:

In [86]:

```
popular_genre=train_data.groupby('Genre').size().unique
print(popular_genre)
genre_list=train_data['Genre'].values.tolist()
```

```
<bound method Series.unique of Genre
A Capella             119
Alternative          9263
Anime                8936
Blues                9023
Children's Music     5403
Children’s Music     9353
Classical            9256
Comedy               9681
Country              8664
Dance                8701
Electronic           9377
Folk                 9299
Hip-Hop              9295
Indie                9543
Jazz                 9441
Movie                7806
Opera                8280
Pop                  9386
R&B                  8992
Rap                  9232
Reggae               8771
Reggaeton            8927
Rock                 9272
Ska                  8874
Soul                 9089
Soundtrack           9646
World                9096
dtype: int64>
```

Для решения задачи классификации выберем два жанра - поп (Pop) и рок (Rock):

In [87]:

```
top_genres = ['Pop', 'Rock']
```

In [88]:

```
train_data = train_data[train_data['Genre'].isin(top_genres)]
train_data['Genre'].unique()
```

Out[88]:

```
array(['Pop', 'Rock'], dtype=object)
```

Проверим размер набора:

In [89]:

```
train_data.shape
```

Out[89]:

```
(18658, 12)
```

Подсчитаем количество исполнителей:

In [90]:

```
train_data['Artist'].unique().size
```

Out[90]:

```
3297
```

Выведем топ-5 исполнителей каждого жанра:

In [92]:

```
for g in top_genres:
    print(g + ':')
    print(train_data[train_data['Genre'] == g]['Artist'].value_counts().head(5))
    print('\n')
```

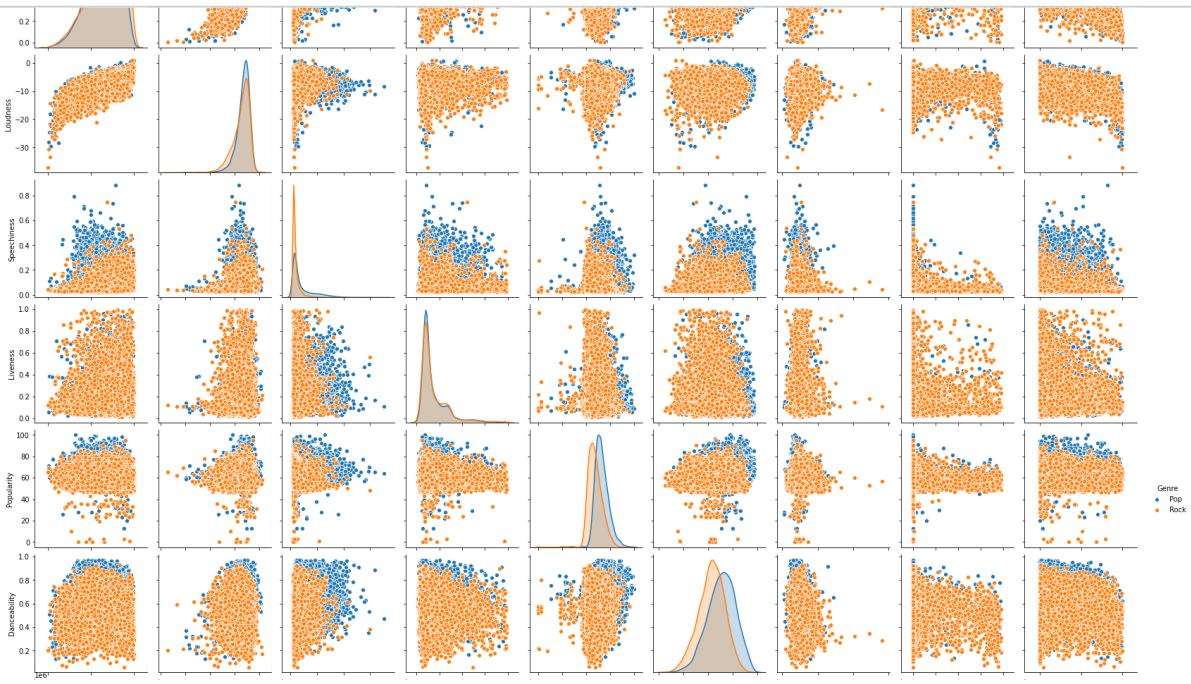
```
Pop:
Drake           154
BTS              76
Kanye West       72
Taylor Swift     67
Future           66
Name: Artist, dtype: int64


Rock:
The Beatles            145
Queen                   97
Led Zeppelin            76
Panic! At The Disco     74
Imagine Dragons         71
Name: Artist, dtype: int64
```

Для понимании структуры набора данных построим графики:

In [15]:

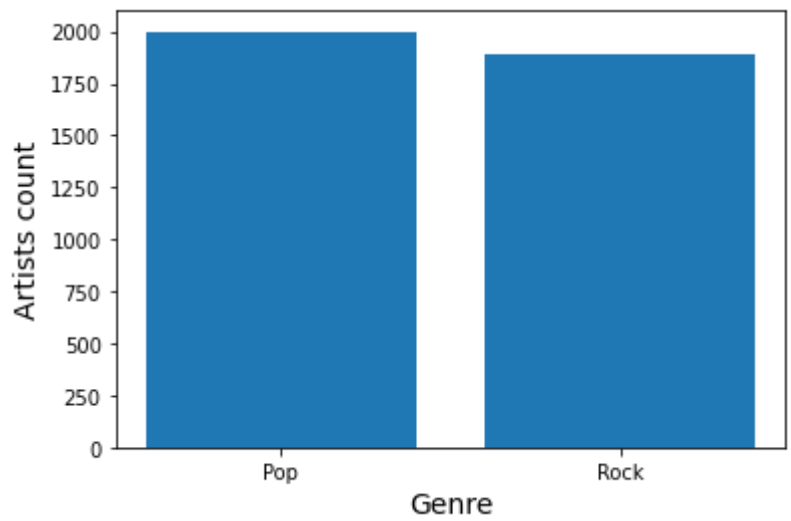
```
sns.pairplot(train_data, hue="Genre")
```




Количество уникальных артистов каждого жанра:

In [93]:

```
x_genres = np.arange(len(top_genres))
y_artists = train_data.groupby('Genre')['Artist'].unique().agg(len)
plt.bar(x_genres, y_artists)
plt.xticks(x_genres, top_genres)
plt.xlabel('Genre', fontsize=14)
plt.ylabel('Artists count', fontsize=14)
plt.show()
```



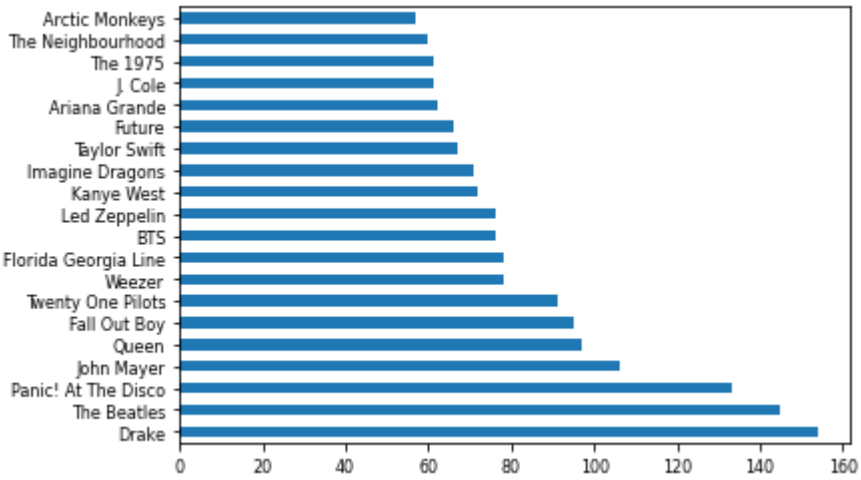
Топ-20 исполнителей по количеству песен:

In [94]:

```
train_data['Artist'].value_counts().head(20).plot(kind='barh', fontsize=8)
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x127684208>
```



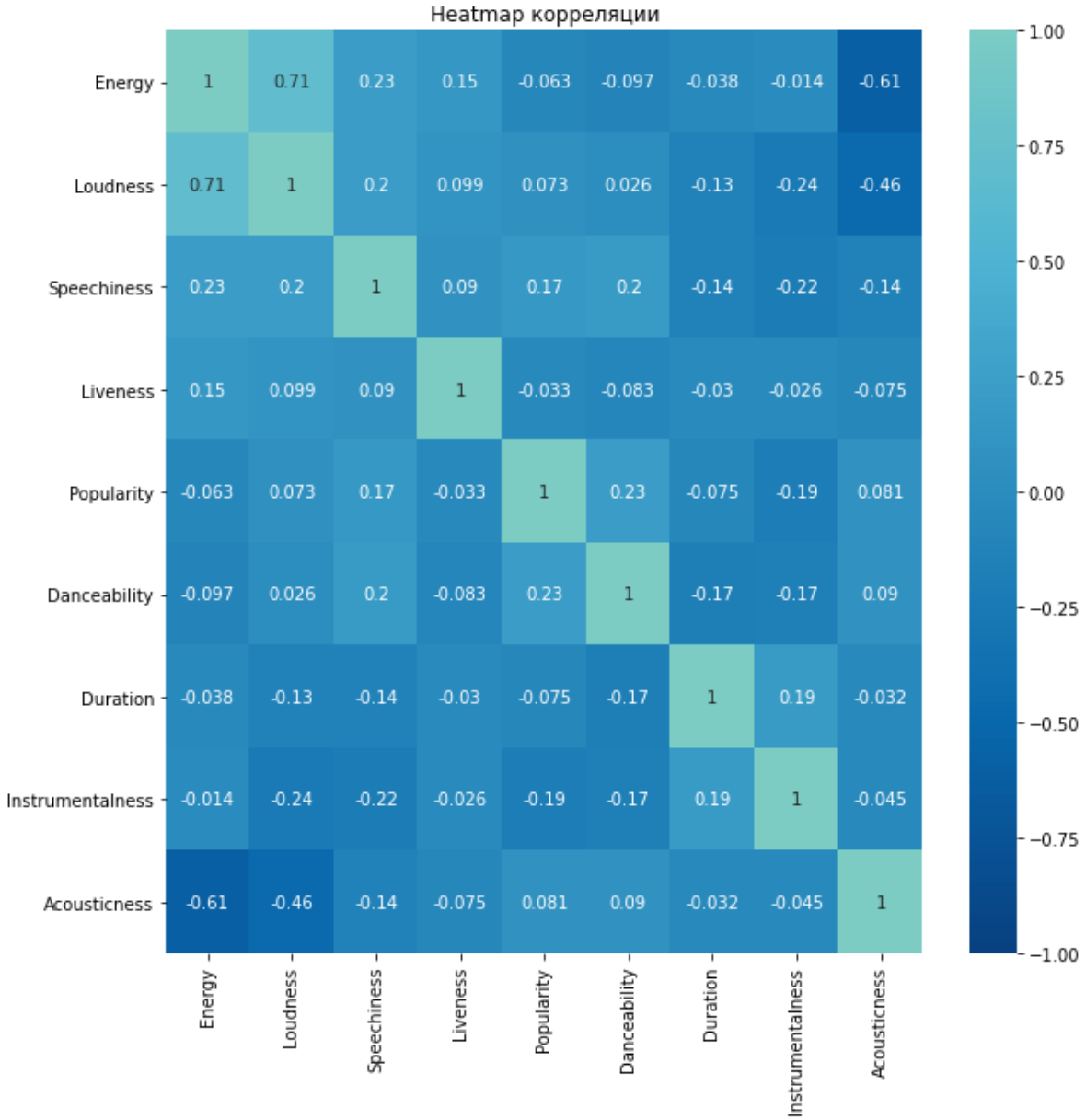
# 3. Корреляционный анализ данных

In [98]:

```
correlation=train_data.corr(method='spearman')
plt.figure(figsize=(10,10))
plt.title('Heatmap корреляции')
sns.heatmap(correlation,annot=True,vmin=-1,vmax=1,cmap="GnBu_r",center=1)
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11f875048>
```



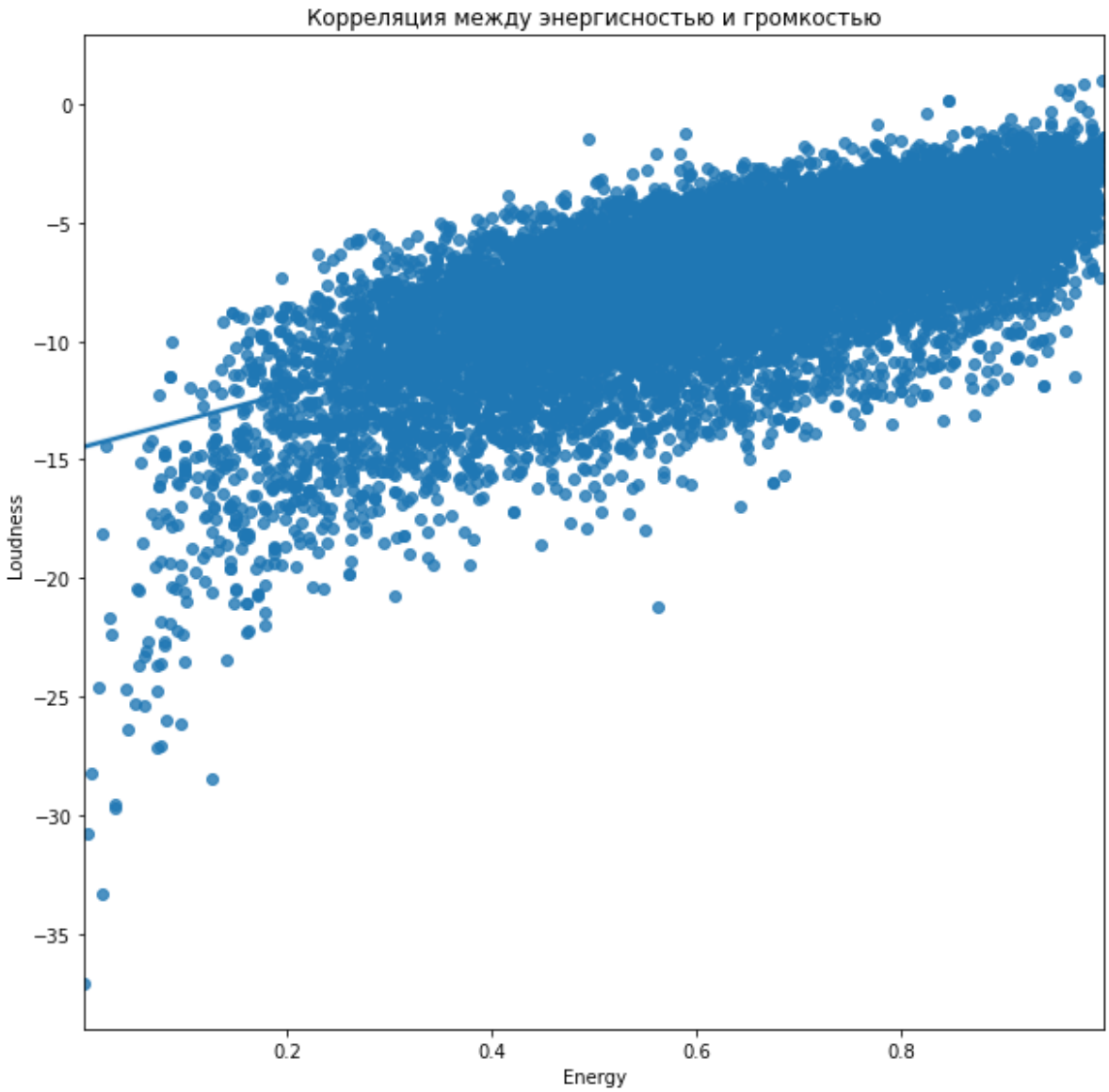
Проверим связь между громкостью и энергичностью:

In [99]:

```
fig=plt.subplots(figsize=(10,10))
plt.title('Корреляция между энергисностью и громкостью')
sns.regplot(x='Energy',y='Loudness',data=train_data)
```

Out[99]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x122c087b8>
```



In [105]:

```
fig=plt.subplots(figsize=(10,10))
plt.title('Корреляция между энергичностью и популярностью')
sns.regplot(x='Energy', y='Popularity',
            ci=None, data=train_data)
sns.kdeplot(train_data['Energy'], train_data['Popularity'])
```

Out[105]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x122212940>
```

# 4. Метрики для оценки качества моделей

In [106]:

```
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)].inde
        # 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()
```

In [107]:

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

In [108]:

```
def test_model(model_name, model, metricLogger):
    model.fit(X_train, Y_train)
    y_pred = model.predict(X_test)

    accuracy = accuracy_score(Y_test, y_pred)
    roc_auc = roc_auc_score(Y_test, y_pred)
    precision = precision_score(Y_test, y_pred)
    recall = recall_score(Y_test, y_pred)

    metricLogger.add('precision', model_name, precision)
    metricLogger.add('recall', model_name, recall)
    metricLogger.add('accuracy', model_name, accuracy)
    metricLogger.add('roc_auc', model_name, roc_auc)

    print('****************************************************')
    print(model)
    print('****************************************************')
    draw_roc_curve(Y_test, y_pred)

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

## 5. Формирование обучающей и тестовой выборок

In [109]:

```
features = ['Genre', 'Acousticness', 'Instrumentalness', 'Energy', 'Loudness', 'Spee
```

In [110]:

```
train_data_enc = train_data.filter(features)
train_data_enc.head()
```

Out[110]:

| | Genre | Acousticness | Instrumentalness | Energy | Loudness | Speechiness | Liveness | Danc |
|---|---|---|---|---|---|---|---|---|
| **107802** | Pop | 0.0421 | 0.000000 | 0.554 | -5.290 | 0.0917 | 0.1060 | |
| **107803** | Pop | 0.1630 | 0.000002 | 0.539 | -7.399 | 0.1780 | 0.1010 | |
| **107804** | Pop | 0.5780 | 0.000000 | 0.321 | -10.744 | 0.3230 | 0.0884 | |
| **107805** | Pop | 0.1490 | 0.000000 | 0.364 | -11.713 | 0.2760 | 0.2710 | |
| **107806** | Pop | 0.5560 | 0.000000 | 0.479 | -5.574 | 0.0466 | 0.0703 | |

Выполним кодирование признака жанра:

In [111]:

```
le = LabelEncoder()
train_data_enc['Genre'] = le.fit_transform(train_data['Genre']);
```

Разделим выборки:

In [112]:

```
X = train_data_enc.drop('Genre', axis=1)
Y = train_data_enc['Genre']
```

In [113]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_sta
print('{}, {}'.format(X_train.shape, X_test.shape))
print('{}, {}'.format(Y_train.shape, Y_test.shape))
```

```
(13993, 9), (4665, 9)
(13993,), (4665,)
```

# 6. Построение базового решения

In [114]:

```
metricLogger = MetricLogger()
```

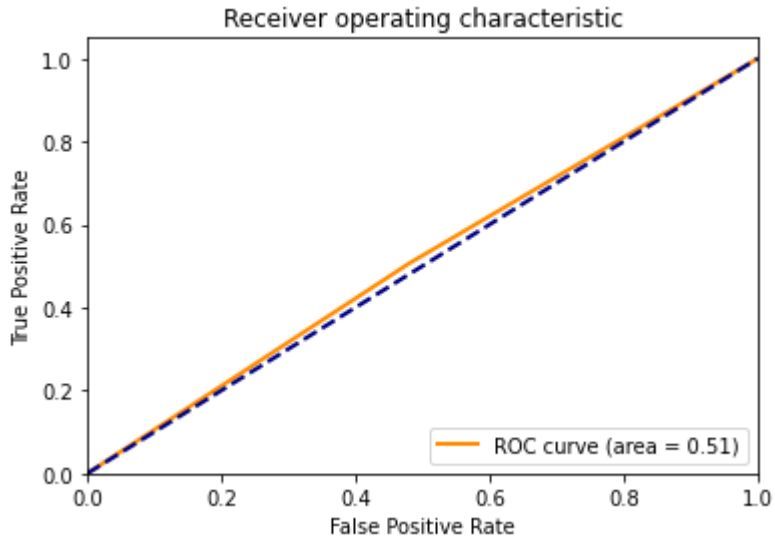
In [115]:

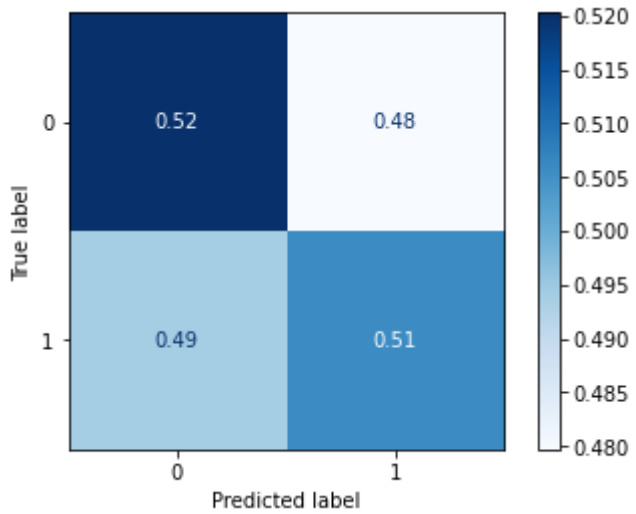
```
models = {'KNN_3':KNeighborsClassifier(n_neighbors=3),
          'SVC':SVC(),
          'Tree':DecisionTreeClassifier(),
          'RF':RandomForestClassifier(),
          'GB':GradientBoostingClassifier()}
```

In [116]:

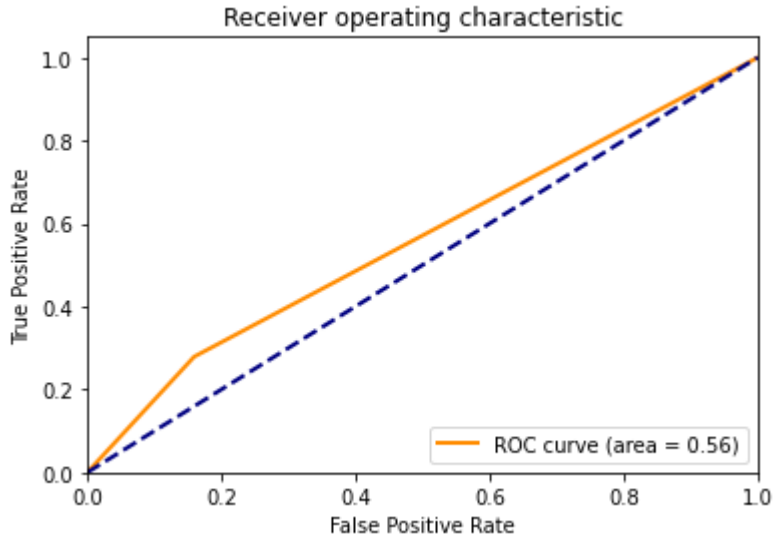
```
for model_name, model in models.items():
    test_model(model_name, model, metricLogger)
```

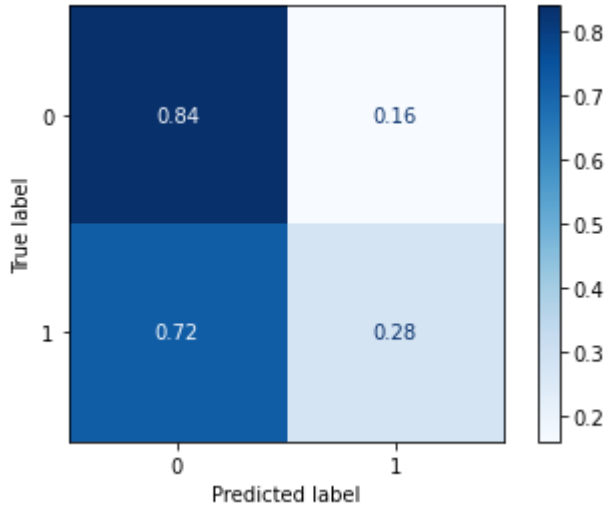
```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=3, p
=2,
                     weights='uniform')
**************************************************
```



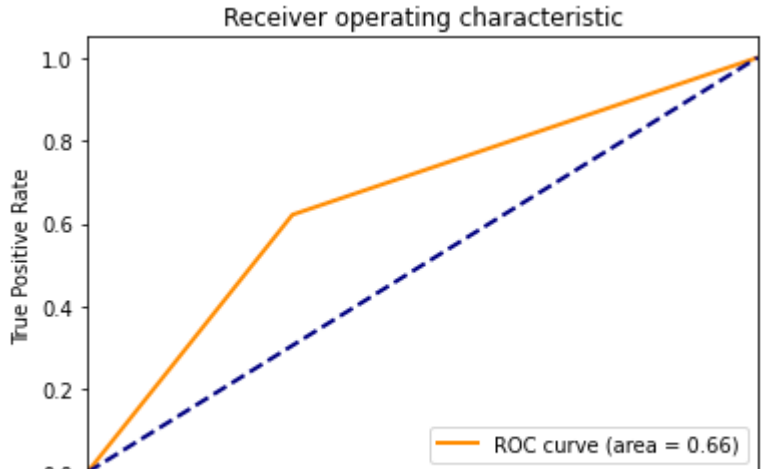



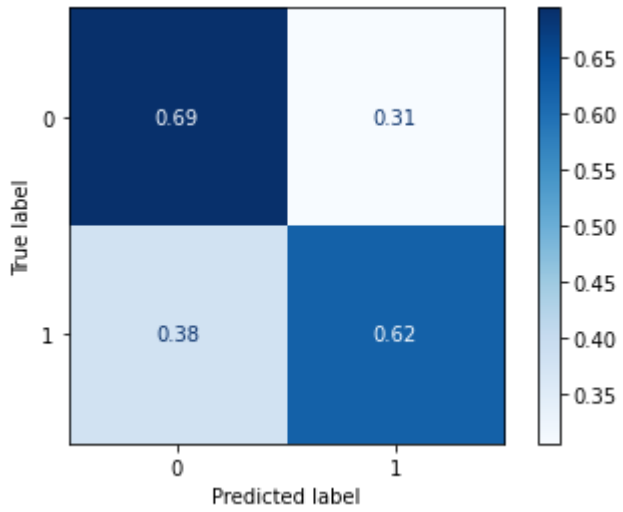

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=
0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rb
f',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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



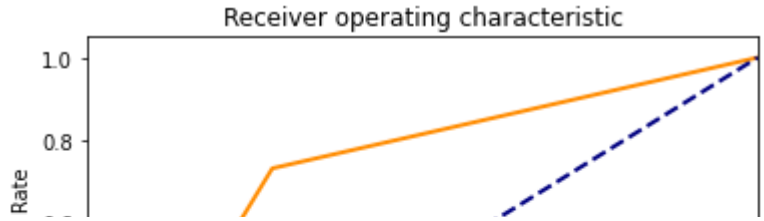



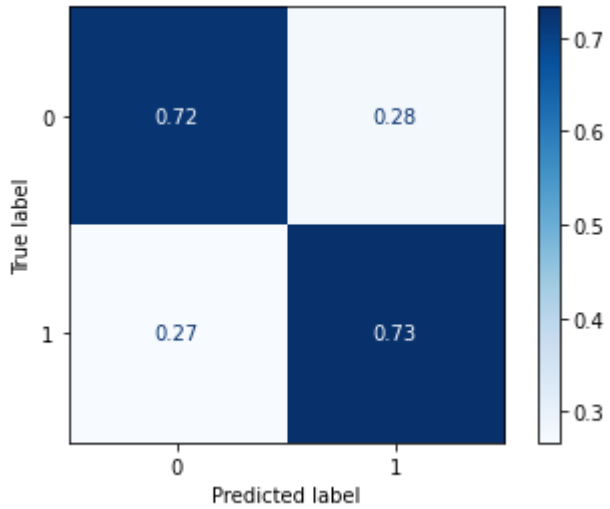

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gi
ni',
                       max_depth=None, max_features=None, max_leaf_nod
es=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=None, splitter='best')
**************************************************
```



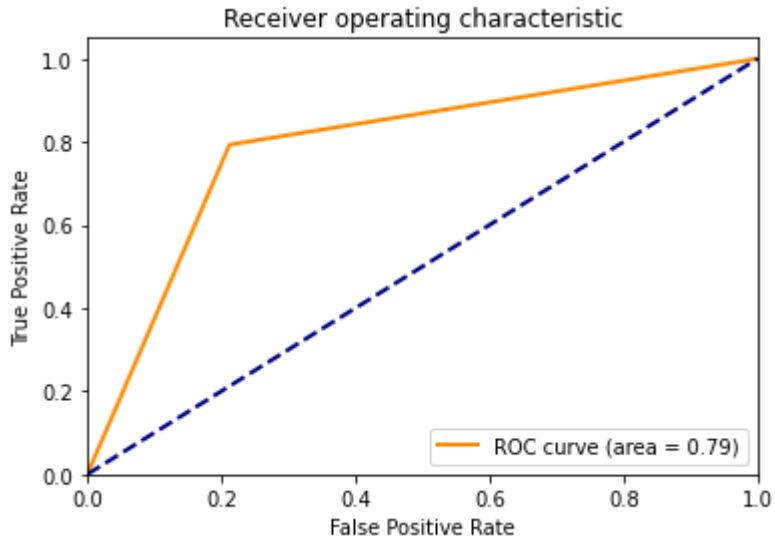



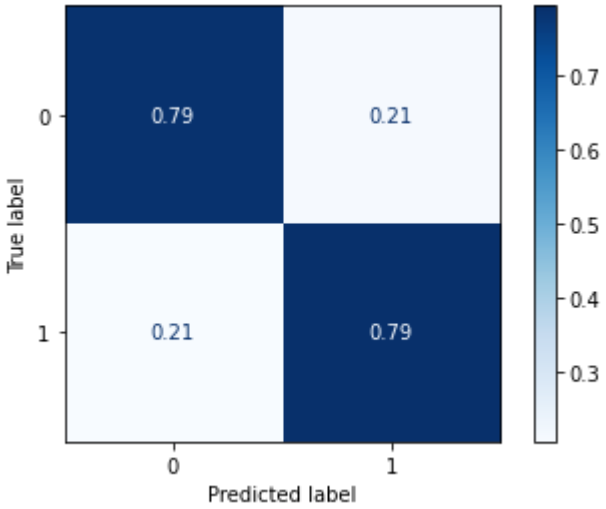

```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=Non
e,
                       criterion='gini', max_depth=None, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       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=Non
e,
                       verbose=0, warm_start=False)
**************************************************
```




```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', in
it=None,
                           learning_rate=0.1, loss='deviance', max_dep
th=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_spl
it=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=
100,
                           n_iter_no_change=None, presort='deprecate
d',
                           random_state=None, subsample=1.0, tol=0.000
1,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```






## 7. Подбор гиперпараметров

In [117]:

```
X_train.shape
```

Out[117]:

```
(13993, 9)
```

In [118]:

```
n_range = np.array(range(10, 500, 10))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[118]:

```
[{'n_neighbors': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 1
00, 110, 120, 130,
        140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 2
60,
        270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370, 380, 3
90,
        400, 410, 420, 430, 440, 450, 460, 470, 480, 490])}]
```

In [119]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accur
clf_gs.fit(X, Y)
```

```
CPU times: user 1.93 s, sys: 251 ms, total: 2.18 s
Wall time: 1min 9s
```

Out[119]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_siz
e=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs
=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=-1,
             param_grid=[{'n_neighbors': array([ 10,  20,  30,  40,  5
0,  60,  70,  80,  90, 100, 110, 120, 130,
       140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 26
0,
       270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370, 380, 39
0,
       400, 410, 420, 430, 440, 450, 460, 470, 480, 490])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring='accuracy', verbose=0)
```

Лучшая модель:

In [120]:

```
clf_gs.best_estimator_
```

Out[120]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                     metric_params=None, n_jobs=None, n_neighbors=380,
p=2,
                     weights='uniform')
```

Лучшее значение параметров:

In [121]:

```
clf_gs.best_params_
```

Out[121]:

```
{'n_neighbors': 380}
```

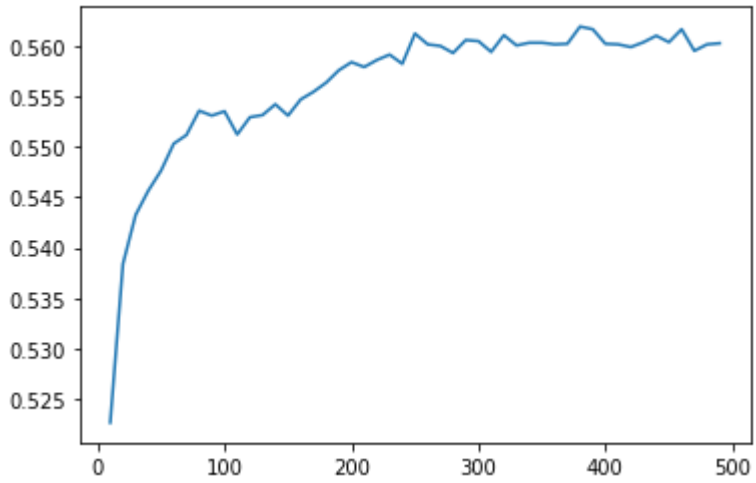
Изменение качества на тестовой выборке в зависимости от K-соседей

In [122]:

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

Out[122]:

```
[<matplotlib.lines.Line2D at 0x1229bf6d8>]
```



In [123]:

```
tree_param = {'criterion':['gini','entropy'], 'splitter': ['best', 'random'], 'max_d

clf_gs = GridSearchCV(DecisionTreeClassifier(), tree_param, cv=5)
clf_gs.fit(X, Y)
```

Out[123]:

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_wei
ght=None,
                                              criterion='gini', max_de
pth=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={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8, 9, 10, 11, 12, 1
5, 20, 30,
                                       40, 50, 70, 90, 120, 150, 100
0],
                         'splitter': ['best', 'random']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=F
alse,
             scoring=None, verbose=0)
```

In [124]:

```
clf_gs.best_estimator_
```

Out[124]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gi
ni',
                       max_depth=4, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=None, splitter='best')
```

In [125]:

```
clf_gs.best_params_
```

Out[125]:

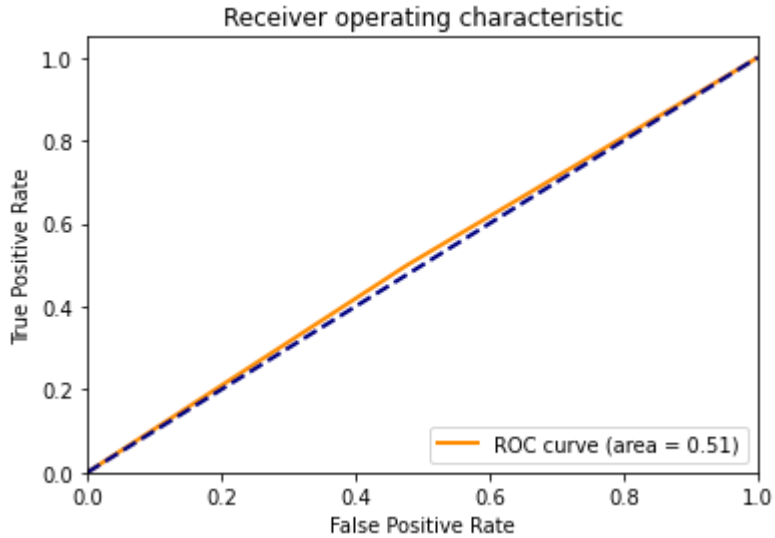
```
{'criterion': 'gini', 'max_depth': 4, 'splitter': 'best'}
```

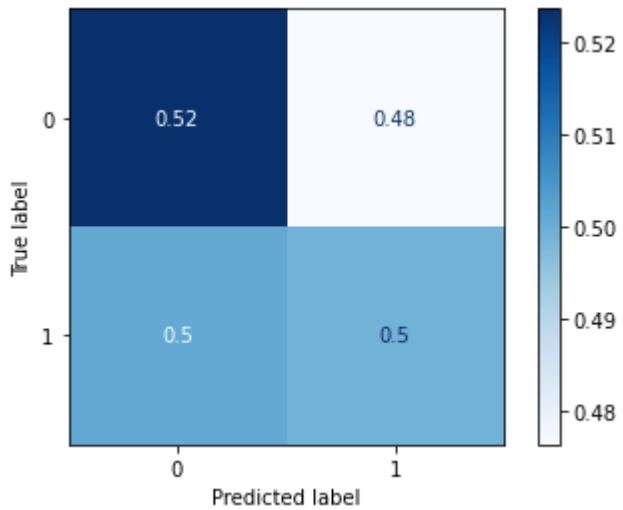
# 8. Сравнение качества полученных моделей с качеством baseline-моделей

In [126]:

```
test_model('KNN_5', KNeighborsClassifier(n_neighbors=5), metricLogger)
```

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

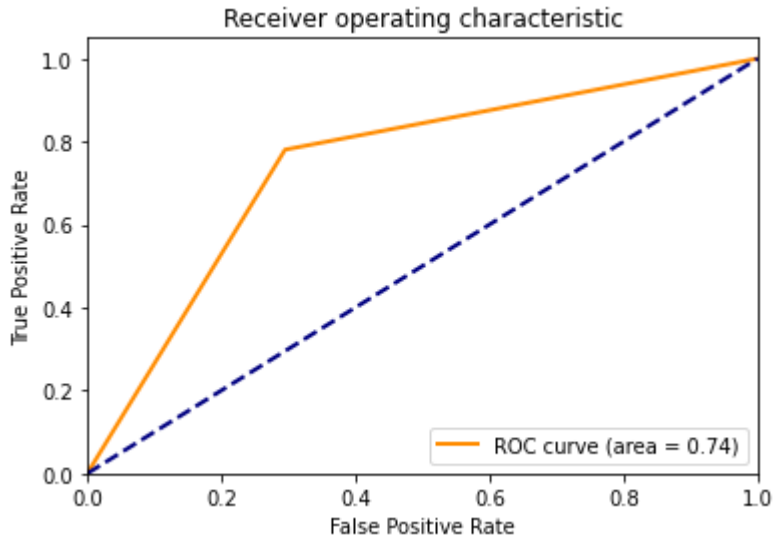


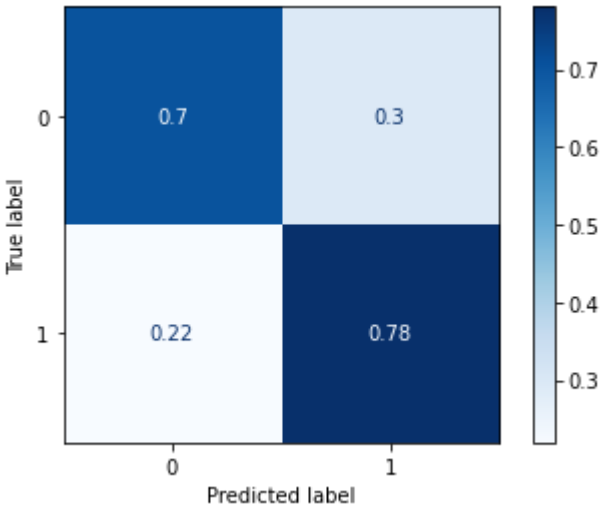



In [127]:

```
test_model('Tree_4', DecisionTreeClassifier(criterion='gini', max_depth=4), metricLc
```

```
**********************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gi
ni',
                       max_depth=4, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=None, splitter='best')
**********************************************************
```

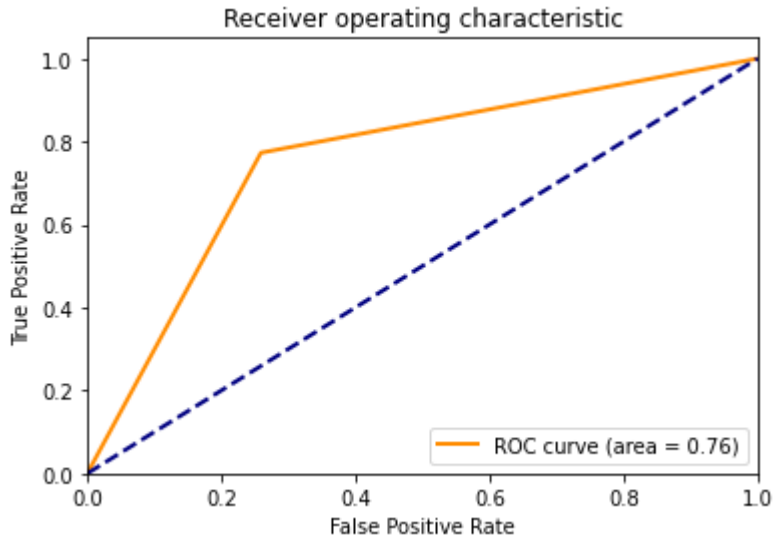


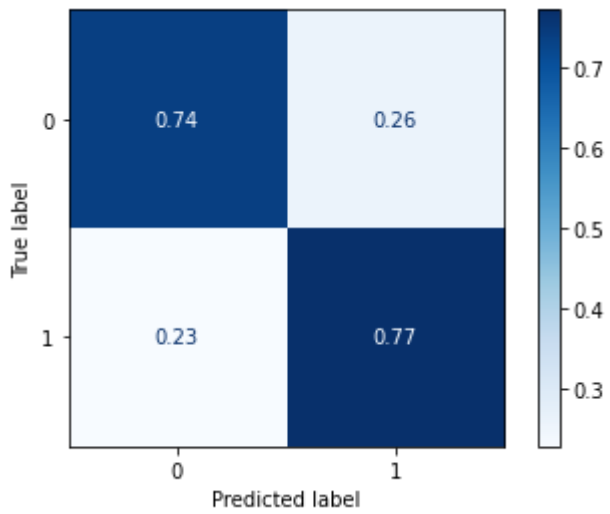

0.7
0.3
0.22
0.78
0
1
True label
0
1
Predicted label
0.7
0.6
0.5
0.4
0.3

In [128]:

```
test_model('Tree_7', DecisionTreeClassifier(criterion='gini', max_depth=7, splitter=
```

```
******************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gi
ni',
                       max_depth=7, max_features=None, max_leaf_nodes=
None,
                       min_impurity_decrease=0.0, min_impurity_split=N
one,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecat
ed',
                       random_state=None, splitter='best')
******************************************************
```




## 9. Выводы о качестве построенных моделей

In [129]:

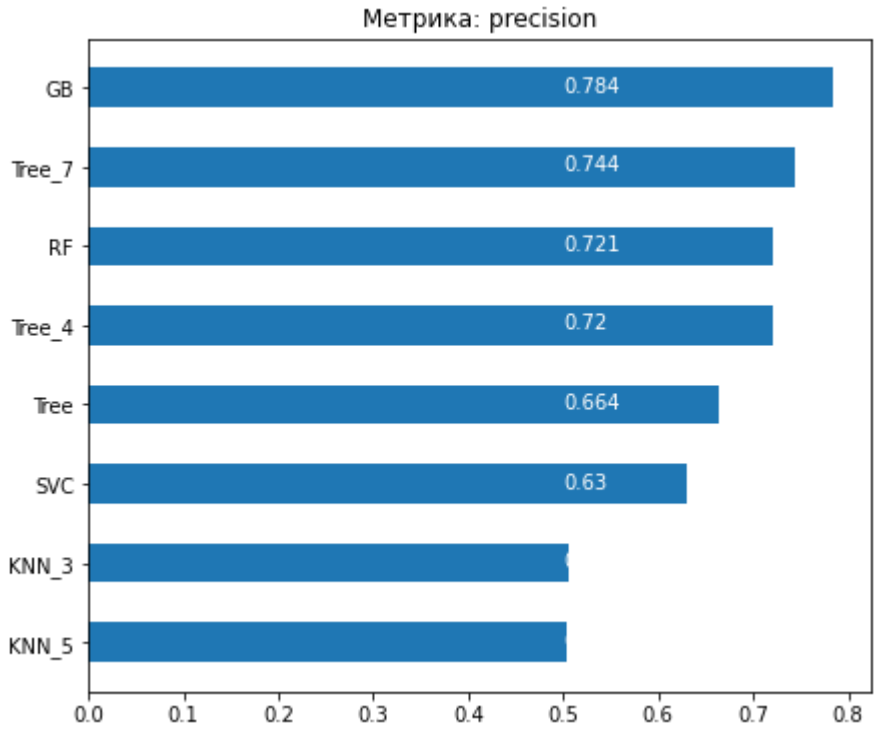
```
metrics = metricLogger.df['metric'].unique()
metrics
```

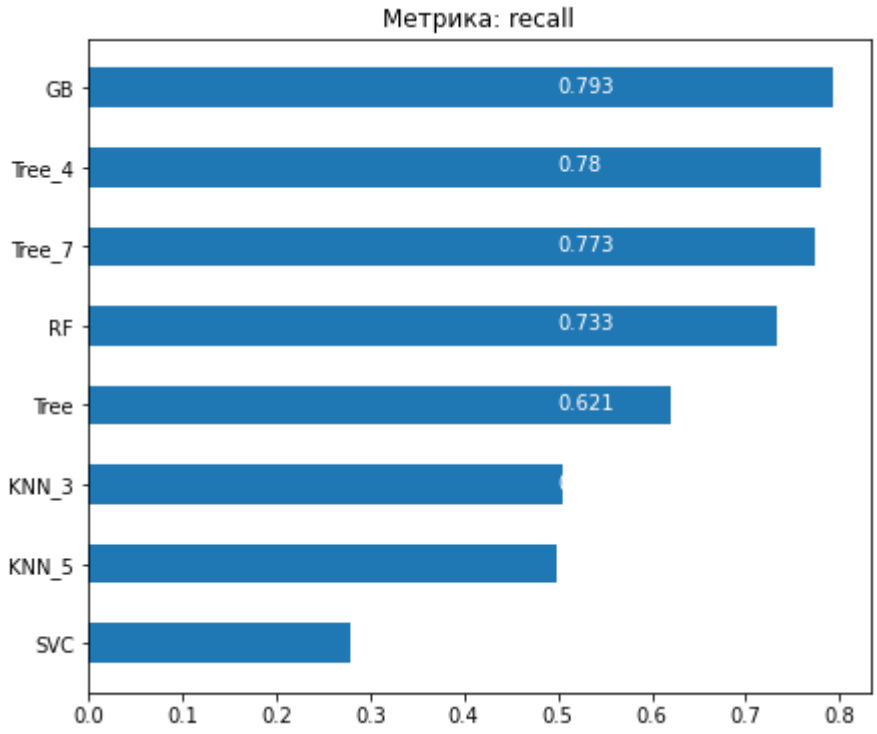
Out[129]:

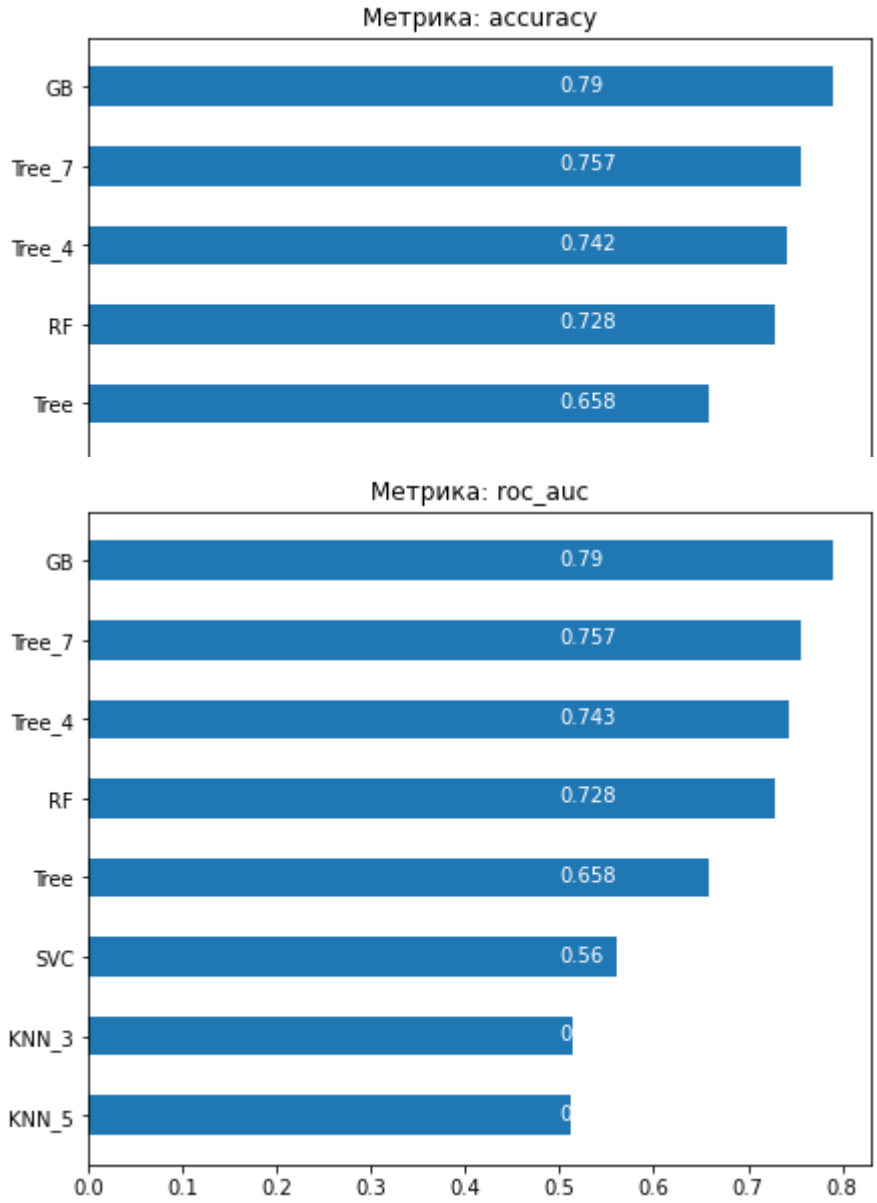
```
array(['precision', 'recall', 'accuracy', 'roc_auc'], dtype=object)
```

In [130]:

```
for metric in metrics:
    metricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```






## 10. Вывод

Было использовано 5 моделей, для двух из них был проведен подбор гиперпараметров. Лучше всего себя показала модель GB, но, к сожалению, для подбора гиперпараметров модели Gradient Boosting не хватило вычислительной мощности.