1. Курсовой проект по ТМО

2. 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.

3. Ход выполнения работы

3.0.1. В качестве набора данных используется набор по исследованию качества белых вин

Датасет состоит из одного файла: - wine.csv

Далее этот файл будет разбит на два — для обучающей и тестовой выборок соответственно.

Файл содержит следующие колонки: 1. fixed acidity — фиксированная кислотность 2. volatile acidity — летучая кислотность 3. citric acid — лимонная кислота 4. residual sugar — остаточный сахар 5. chlorides — хлориды 6. free sulfur dioxide — свободный диоксид серы 7. total sulfur dioxide — общая двуокись серы 8. density — плотность 9. pH — потенциал водорода 10. sulphates — сульфаты 11. alcohol — алкоголь 12. quality — качество алкоголя (выходной параметр)

3.1. Описание каждой из составляющих белого вина

3.1.1. Фиксированная кислотность

Кислоты являются основными свойствами вина и вносят большой вклад во вкус вина. Обычно общая кислотность делится на две группы: летучие кислоты и нелетучие или фиксированные кислоты. Среди основных кислот, которые вы можете найти в винах: винная, яблочная, лимонная и янтарная. Эта переменная выражена в г (винная кислота) / дм³ в наборе данных.

3.1.2. Летучая кислотность

Изменчивая кислотность - это процесс превращения вина в уксус. В США допустимые пределы содержания летучей кислоты составляют 1,1 г / л для белого столового вина. В нашем наборе летучая кислотность выражается в г (уксусная кислота) / дм³

3.1.3. Лимонная кислота

Лимонная кислота является одной из фиксированных кислот, которые вы найдете в винах. Выражается в г / дм3 в датасете.

3.1.4. Остаточный сахар

Этот сахар обычно относится к сахару, остающемуся после прекращения брожения. Выражается в г / дм3 в данных.

3.1.5. Хлориды

Может быть основным фактором солености в вине. Здесь параметр выражен в г (хлорид натрия) / дм3.

3.1.6. Свободный диоксид серы

Говорят, что часть диоксида серы, которая добавляется в вино и которая теряется в нем, связана, а активная часть считается свободной. Винодел всегда будет пытаться получить наибольшую долю свободной серы для связывания. Эта переменная выражена в мг / дм3 в данных.

3.1.7. Общая двуокись серы

Это сумма связанного и свободного диоксида серы (SO2). Здесь она выражено в мг / дм3.

3.1.8. Плотность

Плотность обычно используется в качестве меры превращения сахара в алкоголь. Здесь это выражено в г / см3.

3.1.9. Потенциал водорода

Потенциал водорода представляет собой числовую шкалу для указания кислотности или основности вина. Как вы, возможно, знаете, растворы с pH ниже 7 являются кислыми, а растворы с pH выше 7 являются основными. При pH 7 чистая вода является нейтральной. Большинство вин имеют pH от 2,9 до 3,9 и поэтому являются кислыми.

3.1.10. Сульфаты

Сульфаты в вине, как глютен к еде. Они являются неотъемлемой частью виноделия во всем мире и считаются необходимыми. В данных они выражаются в г (сульфат калия) / дм3.

3.1.11. Алкоголь

Вино - это алкогольный напиток, и, как вы знаете, процент алкоголя может варьироваться от вина к вину. Не следует удивляться, что эта переменная включена в наборы данных, где она выражена в% об.

3.1.12. Качество

Эксперты оценили качество вина между 0 (очень плохо) и 10 (отлично). Окончательное число - это медиана как минимум трех оценок, сделанных теми же винными экспертами.

3.1.13. Импорт библиотек

[93]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import os
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
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_error, 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 gmdhpy import gmdh
%matplotlib inline
sns.set(style="ticks")
```

3.1.14. Разбиение исходного файла на обчующую и тестовую выборку

Файл wine.csv разбит на два файла в соотношении ~ 0.7 : 0.3. Файлы приобретают итерационные числовые названия, начиная с “1.csv”.

[6]:
```
#                  ,
csvfile = open('wine.csv', 'r').readlines()
filename = 1
for i in range(len(csvfile)):
    if i % 3500 == 0:
        open(str(filename) + '.csv', 'w+').writelines(csvfile[i:i+3500])
        filename += 1
```

3.1.15. Создание обучающей выборки

Импортируем файл 1.csv в качестве обучающей выборки.

[7]:
```
#
train = pd.read_csv('1.csv', sep=";")
```

3.1.16. Выводим информацию об обучающей выборке

[8]:
```
#     5
train.head()
```

[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

[9]:
```
#                - 3499    , 12
train.shape
```

[9]:
```
(3499, 12)
```

[10]:
```
#
train.columns
```

[10]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

[11]:
```
#
train.dtypes
```

[11]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

[12]:
```
#
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3499 entries, 0 to 3498
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         3499 non-null   float64
 1   volatile acidity      3499 non-null   float64
 2   citric acid           3499 non-null   float64
 3   residual sugar        3499 non-null   float64
 4   chlorides             3499 non-null   float64
 5   free sulfur dioxide   3499 non-null   float64
 6   total sulfur dioxide  3499 non-null   float64
 7   density               3499 non-null   float64
 8   pH                    3499 non-null   float64
 9   sulphates             3499 non-null   float64
 10  alcohol               3499 non-null   float64
 11  quality               3499 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 328.2 KB
```

3.1.17. Проверка на наличие нулевых значений

Делаем вывод, что их нет.

[13]:
```
#                    (                    )
train.isnull().sum()
```

[13]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

3.1.18. Создание тестовой выборки

Импортируем файл 2.csv в качестве тестовой выборки.

[55]:
```
#
#
test = pd.read_csv('2.csv', sep=";")
```

3.1.19. Выводим информацию о тестовой выборке

[56]:
```
#     5
test.head()
```

[56]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	6.0	0.28	0.27	15.5	0.036	31.0	134.0	0.99408	3.19	0.44	13.0	7
1	6.7	0.24	0.36	8.4	0.042	42.0	123.0	0.99473	3.34	0.52	10.9	6
2	6.7	0.29	0.45	14.3	0.054	30.0	181.0	0.99869	3.14	0.57	9.1	5
3	6.9	0.33	0.31	4.2	0.040	21.0	93.0	0.98960	3.18	0.48	13.4	7
4	6.5	0.16	0.34	1.4	0.029	29.0	133.0	0.99108	3.33	0.64	11.5	7

[57]:
```
#                - 1399    , 12
test.shape
```

[57]:
```
(1398, 12)
```

3.1.20. Проверка на наличие нулевых значений в тестовой выборке

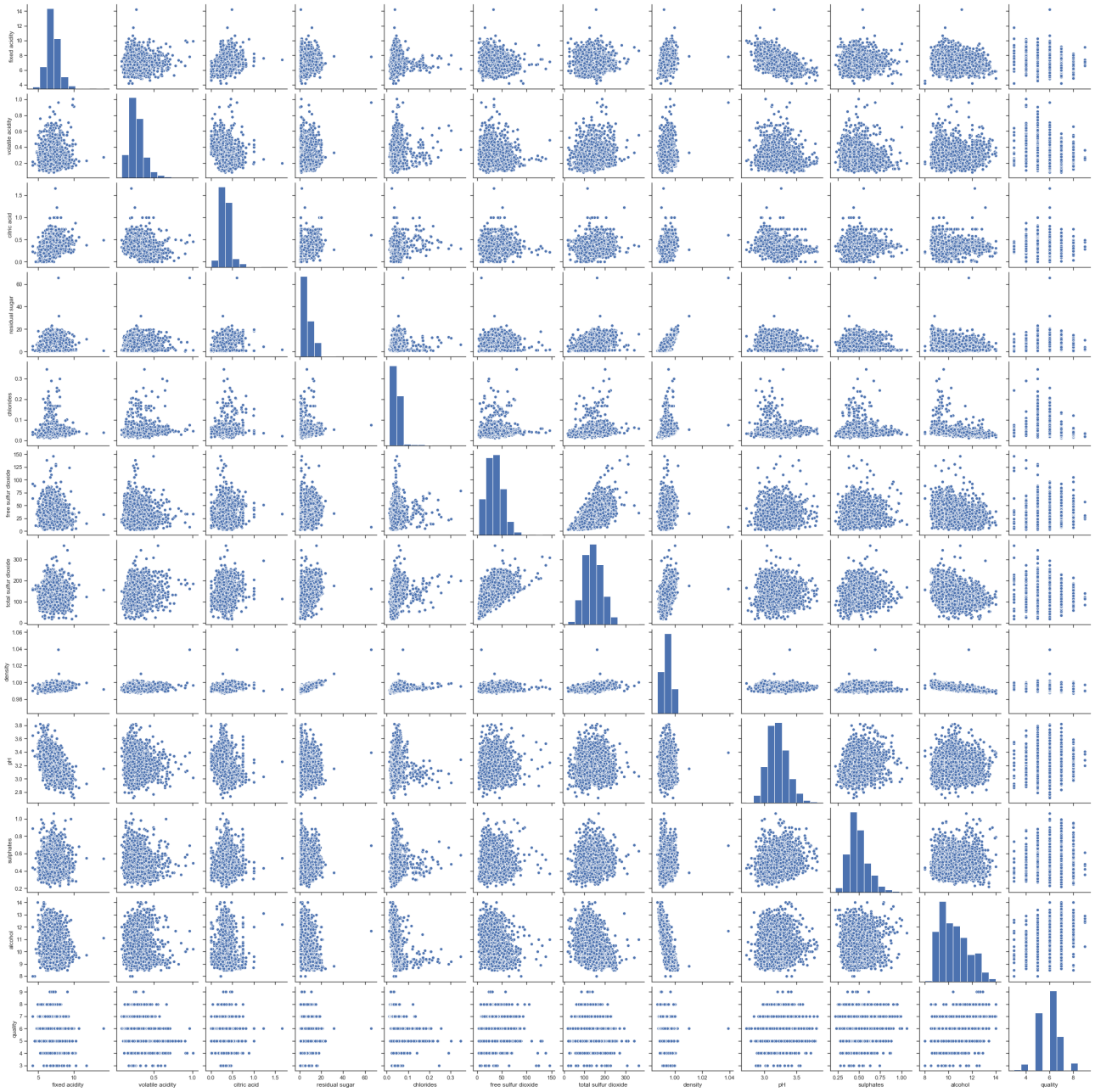
[58]:
```
#
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1398 entries, 0 to 1397
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1398 non-null   float64
 1   volatile acidity      1398 non-null   float64
 2   citric acid           1398 non-null   float64
 3   residual sugar        1398 non-null   float64
 4   chlorides             1398 non-null   float64
 5   free sulfur dioxide   1398 non-null   float64
 6   total sulfur dioxide  1398 non-null   float64
 7   density               1398 non-null   float64
 8   pH                    1398 non-null   float64
 9   sulphates             1398 non-null   float64
 10  alcohol               1398 non-null   float64
 11  quality               1398 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 131.2 KB
```

3.1.21. Построим парные диаграммы для обучающей выборки

[18]:
```
#
sns.pairplot(train)
```

[18]:
```
<seaborn.axisgrid.PairGrid at 0x1a1c295710>
```

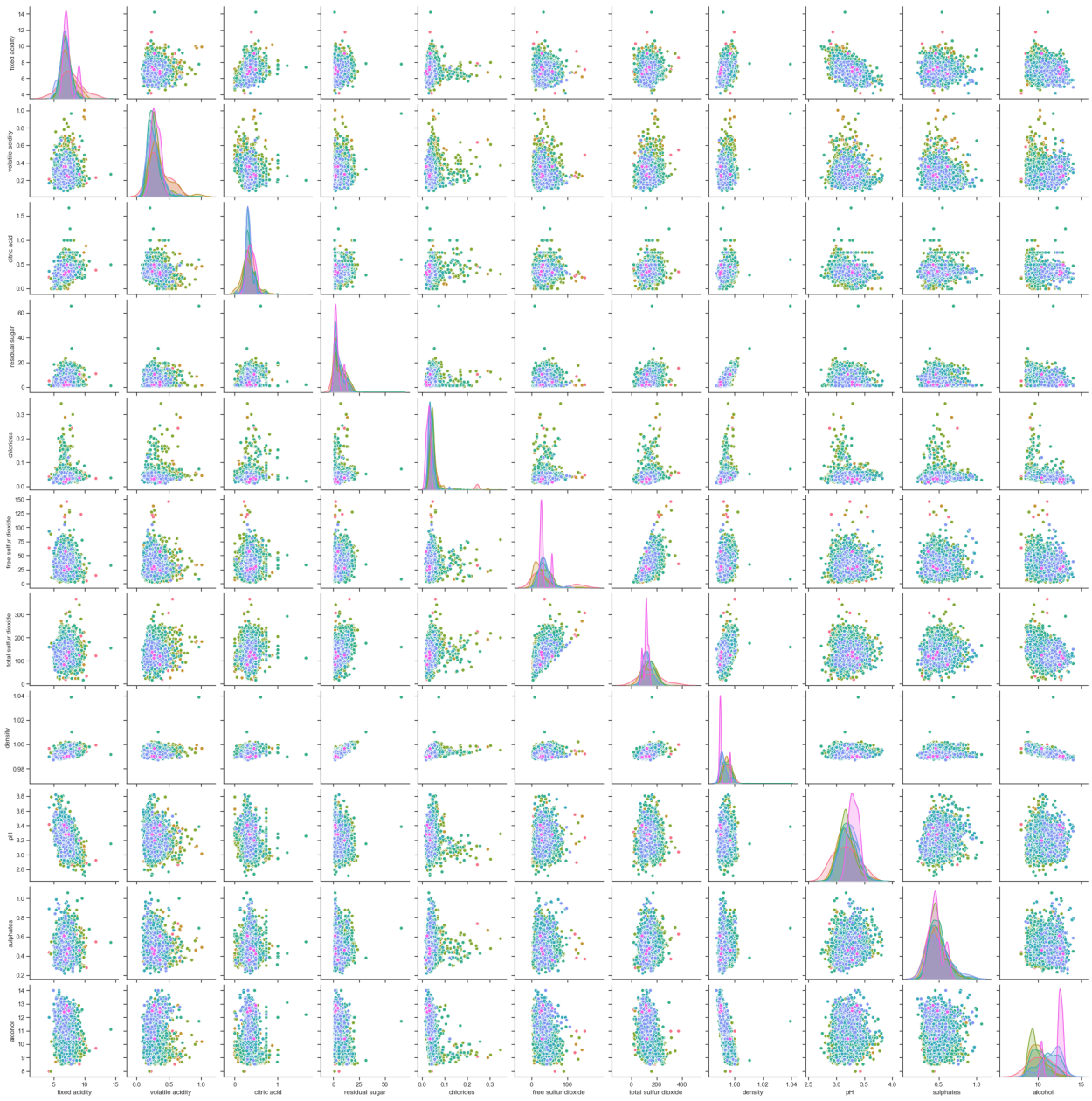


3.1.22. Построим парные диаграммы с использованием целевого признака классификации

Целевой признак для классификации будет ‘quality’ — качество вина.

[317]:
```
sns.pairplot(train, hue="quality", palette = 'husl')
```

[317]:
```
<seaborn.axisgrid.PairGrid at 0x1a2d46f390>
```



3.1.23. Возможные значения целевого признака в обучающей выборке

[20]:
```
#                              ,          ,
np.sort(train['quality'].unique())
```

[20]:
```
array([3, 4, 5, 6, 7, 8, 9])
```

3.1.24. Возможные значения целевого признака в тестовой выборке

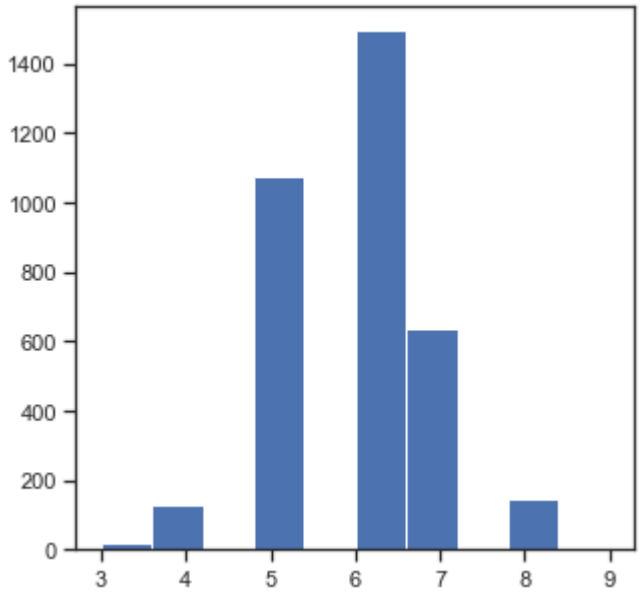
[318]:
```
np.sort(test['quality'].unique())
```

[318]:
```
array([3, 4, 5, 6, 7, 8])
```

Количество уникальных целевых значений отличается для обучающей и тестовый выборки. Здесь наглядно продемонстрирована проблема неравномерного распределения классов по выборкам.

3.1.25. Оценка дисбаланса классов для обучающей выборки

[21]:
```
#                    quality
fig, ax = plt.subplots(figsize=(5,5))
plt.hist(train['quality'])
plt.show()
```



[320]:
```
#                    ,
train['quality'].value_counts()
```

[320]:
```
6    1494
5    1077
7     635
8     144
4     126
3      18
9       5
Name: quality, dtype: int64
```

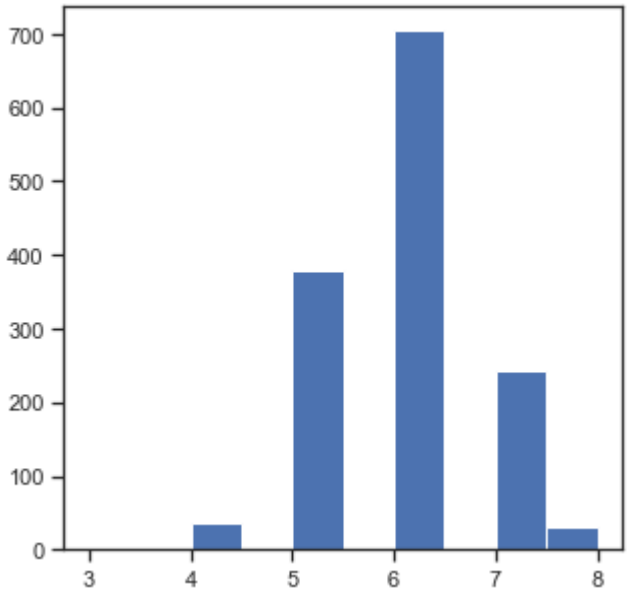
[321]:
```
#
total = train.shape[0]
class_6, class_5, class_7, class_8, class_4, class_3, class_9 =␣
 ↪train['quality'].value_counts()
print('    3       {}%, \n    4       {}%, \n    5       {}%, \n    6       ␣
 ↪{}%, \n    7       {}%, \n    8       {}%, \n    9       {}%.'
      .format(round(class_3 / total, 4)*100,
              round(class_4 / total, 4)*100,
```

```
            round(class_5 / total, 4)*100,
            round(class_6 / total, 4)*100,
            round(class_7 / total, 4)*100,
            round(class_8 / total, 4)*100,
            round(class_9 / total, 4)*100))
```

```
3       0.51%,
4       3.5999999999999996%,
5       30.78%,
6       42.699999999999996%,
7       18.15%,
8       4.12%,
9       0.13999999999999999%.
```

3.1.26. Оценка дисбаланса классов для тестовой выборки

[319]:
```
#                   quality
fig, ax = plt.subplots(figsize=(5,5))
plt.hist(test['quality'])
plt.show()
```



[322]:
```
#                    ,
test['quality'].value_counts()
```

[322]:
```
6    704
5    380
7    244
4     37
8     31
3      2
```

```
Name: quality, dtype: int64
```

[323]:
```
#
total = test.shape[0]
class_6, class_5, class_7, class_4, class_8, class_3 = test['quality'].
 ↪value_counts()
print('    3       {}%, \n    4       {}%, \n    5       {}%, \n    6       ␣
 ↪{}%, \n    7       {}%, \n    8       {}%.'
      .format(round(class_3 / total, 4)*100,
              round(class_4 / total, 4)*100,
              round(class_5 / total, 4)*100,
              round(class_6 / total, 4)*100,
              round(class_7 / total, 4)*100,
              round(class_8 / total, 4)*100))
```

```
    3       0.13999999999999999%,
    4       2.65%,
    5       27.18%,
    6       50.36000000000001%,
    7       17.45%,
    8       2.22%.
```

3.1.27. Выводы об оценке дисбаланса классов

Дисбаланс классов неравномерен к рамках обучающей и тестовой выборках по отдельности.

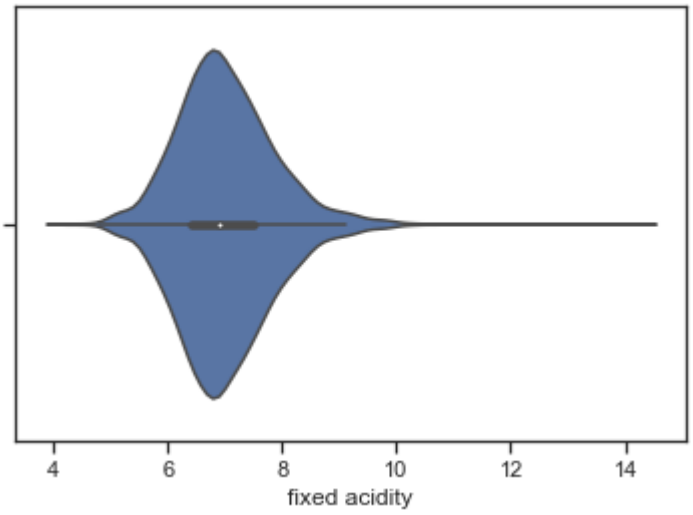
Также сложилась такая ситуация, что количество уникальных значений целевого признака в тестовой выборке меньше. Это следствие дисбаланса распределения классов.

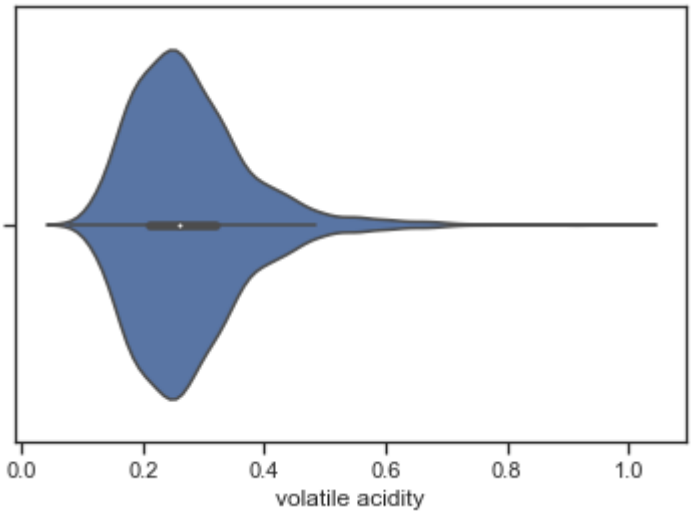
Было выявлено, что что для задачи классификации подходят не все классы (нам не подходят классы, которые встречаются < 10% раз).

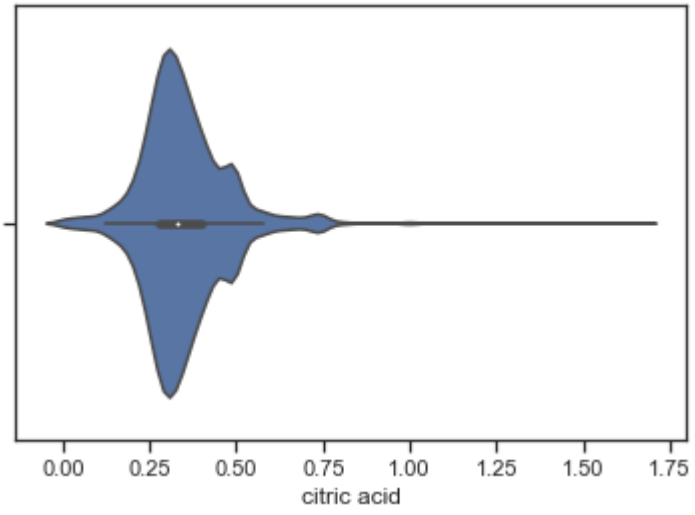
Поэтому для задачи классификации у нас будет только 2 класса: - оценка качества 6; - оценка качества 7.

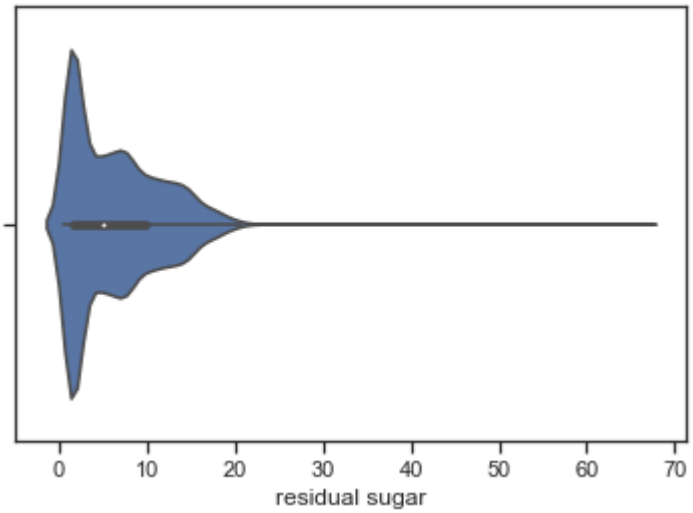
3.1.28. Построение скрипичных диаграмм для обучающей выборки

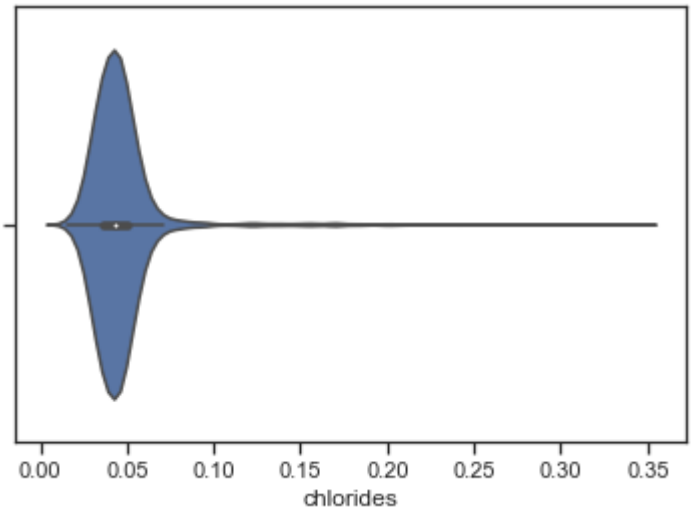
[24]:
```
#
for col in ['fixed acidity', 'volatile acidity', 'citric acid', 'residual␣
 ↪sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',␣
 ↪'density',
       'pH', 'sulphates', 'alcohol', 'quality']:
    sns.violinplot(x=train[col])
    plt.show()
```

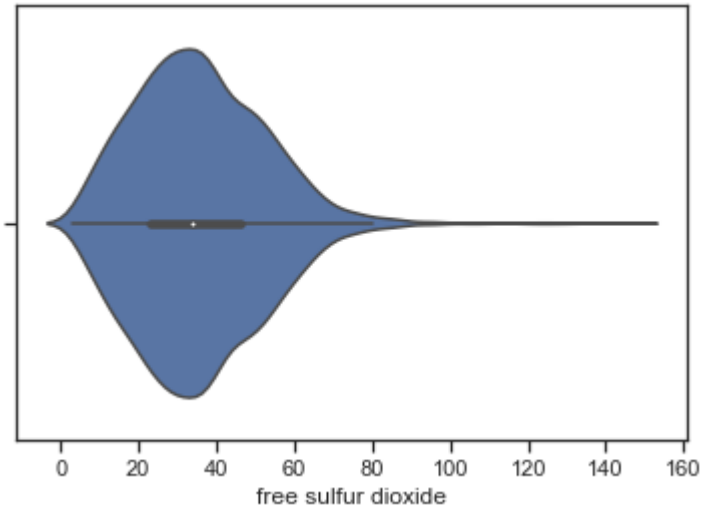

4
6
8
10
12
14
fixed acidity

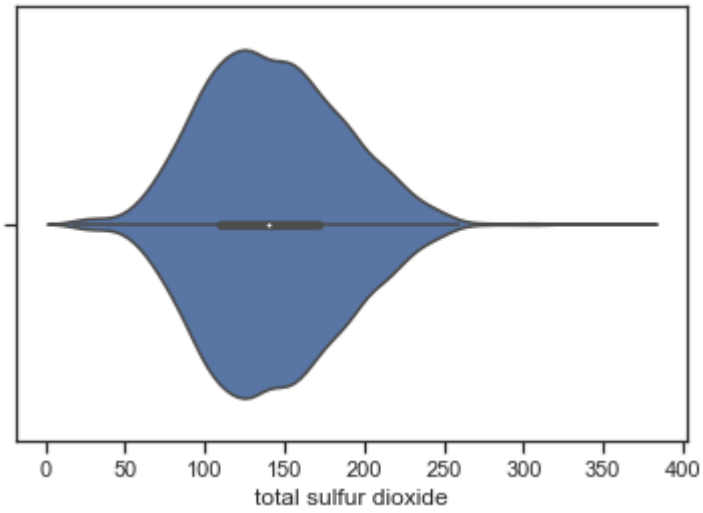

0.0
0.2
0.4
0.6
0.8
1.0
volatile acidity

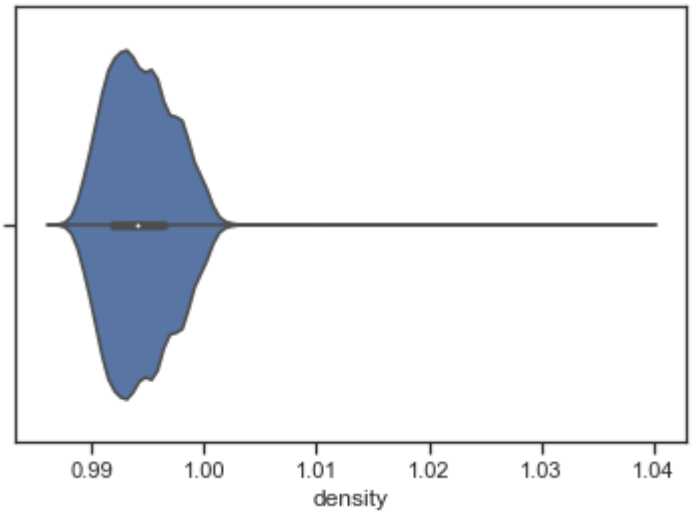

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
citric acid

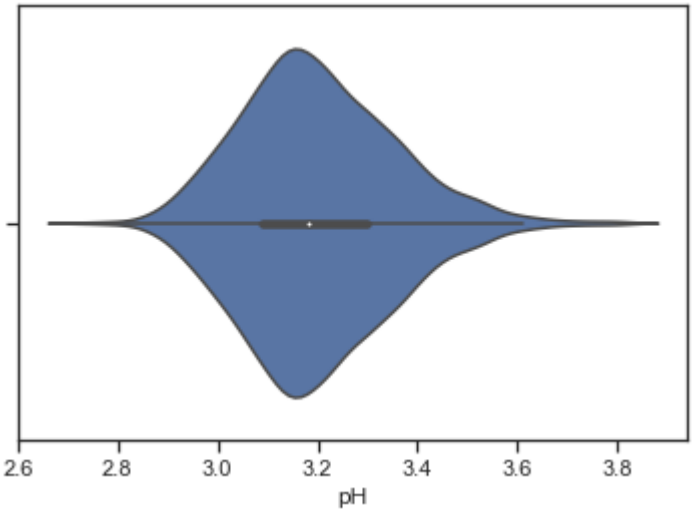

0
10
20
30
40
50
60
70
residual sugar

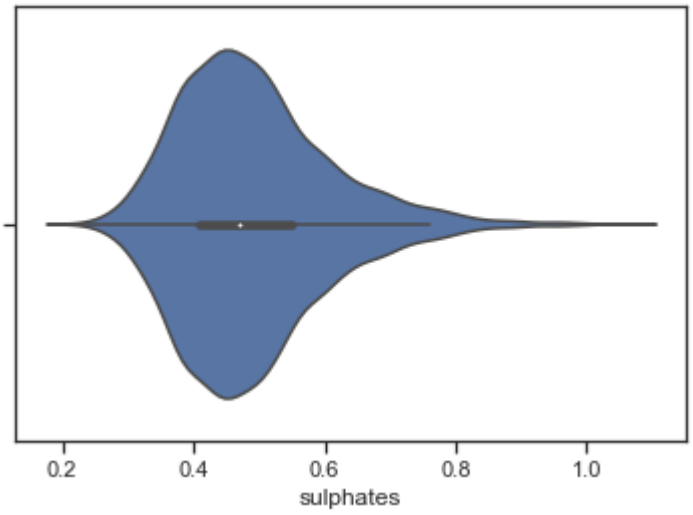

0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
chlorides

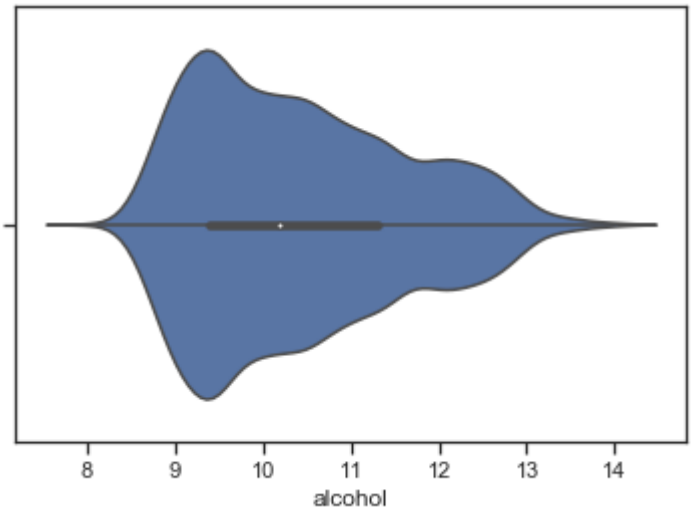

0
20
40
60
80
100
120
140
160
free sulfur dioxide

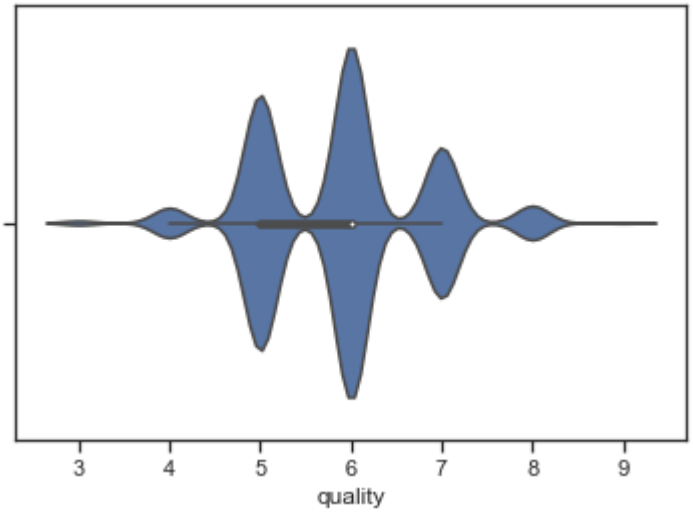

0
50
100
150
200
250
300
350
400
total sulfur dioxide


0.99
1.00
1.01
1.02
1.03
1.04
density


2.6
2.8
3.0
3.2
3.4
3.6
3.8
pH


0.2
0.4
0.6
0.8
1.0
sulphates


8
9
10
11
12
13
14
alcohol


3
4
5
6
7
8
9
quality

3.1.29. Убеждаемся, что все типы числовые

[25]:
```
train.dtypes
```

[25]:
```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

[59]:
```
#                    ,
train['dataset'] = 'TRAIN'
test['dataset'] = 'TEST'
```

[60]:
```
#
join_cols = ['dataset', 'fixed acidity', 'volatile acidity', 'citric acid',␣
↪'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',␣
↪'density',
       'pH', 'sulphates', 'alcohol', 'quality']
```

3.1.30. Склеиваем обучающую и тестовую выборку с дополнительным полем-флагом

[152]:
```
data = pd.concat([train[join_cols], test[join_cols]])
```

[153]:
```
#
assert data.shape[0] == train.shape[0]+test.shape[0]
```

[154]:
```
data.head()
```

[154]:

	dataset	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	TRAIN	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00
1	TRAIN	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30
2	TRAIN	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26
3	TRAIN	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19
4	TRAIN	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19

```
   sulphates  alcohol  quality
0       0.45      8.8        6
1       0.49      9.5        6
2       0.44     10.1        6
3       0.40      9.9        6
4       0.40      9.9        6
```

3.1.31. На всякий случай отмасштабируем все признаки, кроме целевого

[155]:
```
#
scale_cols = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual␣
 ↪sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',␣
 ↪'density',
       'pH', 'sulphates', 'alcohol']
```

[156]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

[157]:
```
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

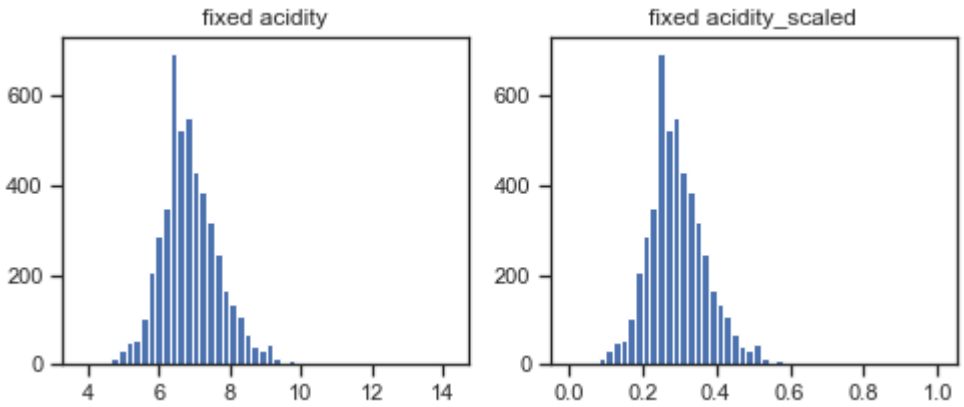
[158]:
```
data.head()
```

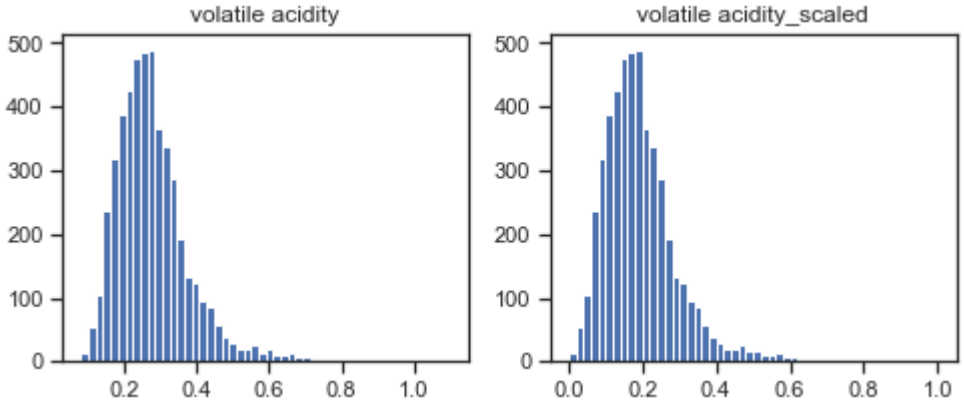
[158]:
```
  dataset  fixed acidity  volatile acidity  citric acid  residual sugar  \
0   TRAIN            7.0              0.27         0.36            20.7
1   TRAIN            6.3              0.30         0.34             1.6
2   TRAIN            8.1              0.28         0.40             6.9
3   TRAIN            7.2              0.23         0.32             8.5
4   TRAIN            7.2              0.23         0.32             8.5

   chlorides  free sulfur dioxide  total sulfur dioxide  density    pH  … \
0      0.045                 45.0                 170.0   1.0010  3.00  …
1      0.049                 14.0                 132.0   0.9940  3.30  …
2      0.050                 30.0                  97.0   0.9951  3.26  …
3      0.058                 47.0                 186.0   0.9956  3.19  …
4      0.058                 47.0                 186.0   0.9956  3.19  …

   volatile acidity_scaled  citric acid_scaled  residual sugar_scaled  \
0                 0.186275            0.216867               0.308282
1                 0.215686            0.204819               0.015337
2                 0.196078            0.240964               0.096626
3                 0.147059            0.192771               0.121166
4                 0.147059            0.192771               0.121166

   chlorides_scaled  free sulfur dioxide_scaled  total sulfur dioxide_scaled␣
 ↪ \
```

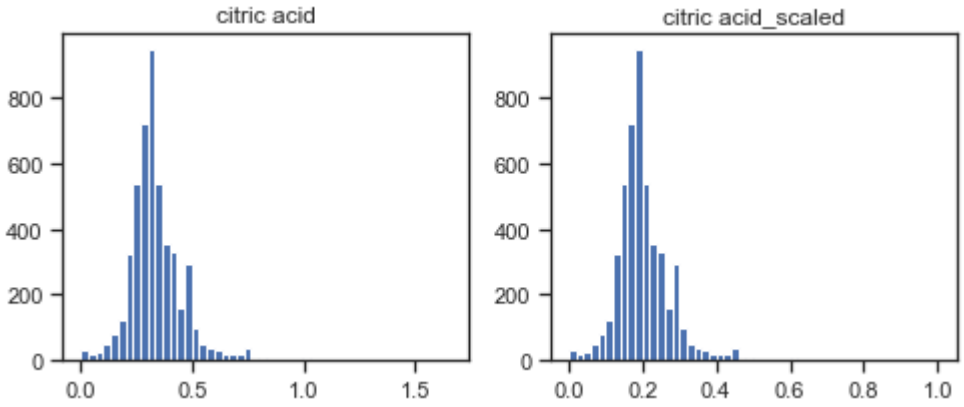
```
0          0.106825                    0.149826                    0.373550
1          0.118694                    0.041812                    0.285383
2          0.121662                    0.097561                    0.204176
3          0.145401                    0.156794                    0.410673
4          0.145401                    0.156794                    0.410673

   density_scaled  pH_scaled  sulphates_scaled  alcohol_scaled
0        0.267785   0.254545          0.267442        0.129032
1        0.132832   0.527273          0.313953        0.241935
2        0.154039   0.490909          0.255814        0.338710
3        0.163678   0.427273          0.209302        0.306452
4        0.163678   0.427273          0.209302        0.306452

[5 rows x 24 columns]
```

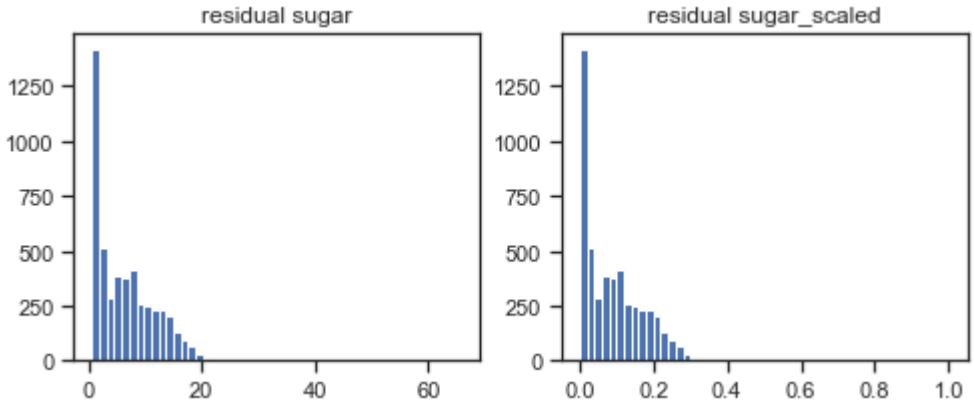
3.1.32. Убедимся, что масштабирование не повлияло на распределение данных

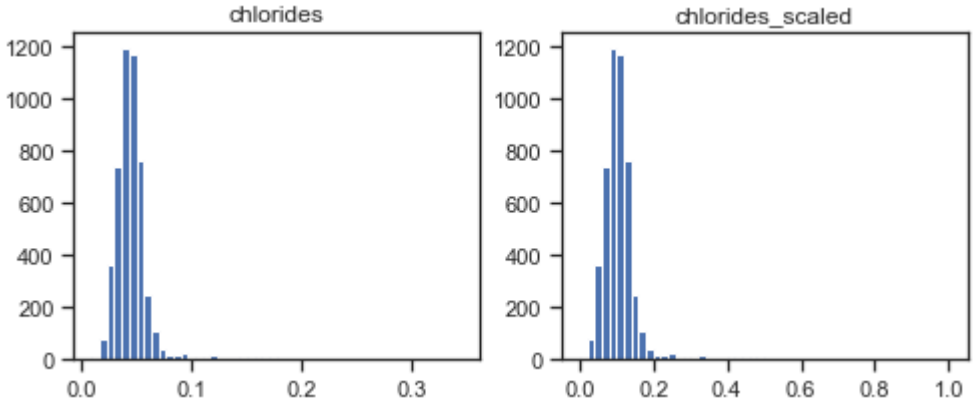
[159]:
```
#      ,
for col in scale_cols:
    col_scaled = col + '_scaled'

    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()
```



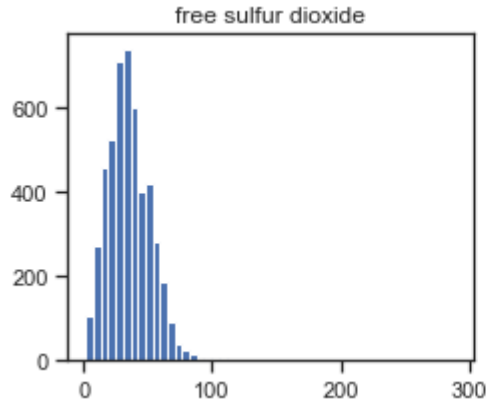

volatile acidity
volatile acidity_scaled

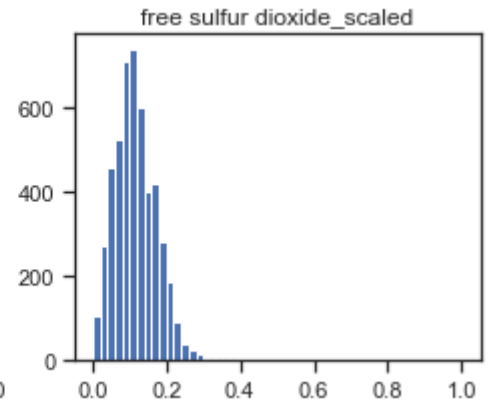

citric acid
citric acid_scaled

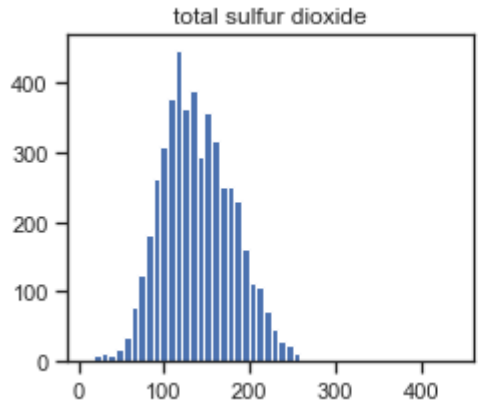

residual sugar
residual sugar_scaled

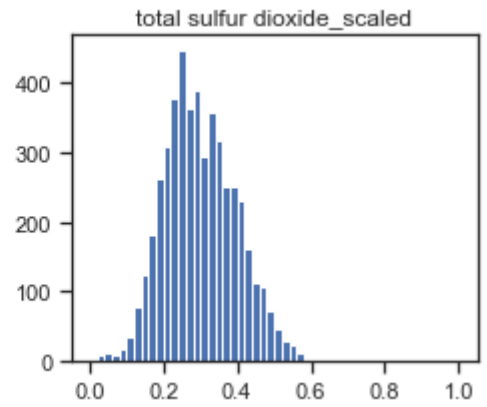

chlorides
1200
1000
800
600
400
200
0
0.0
0.1
0.2
0.3

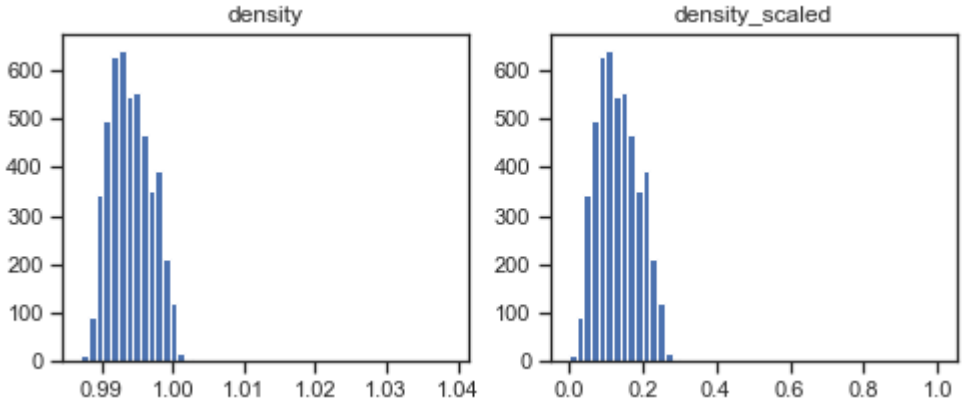
chlorides_scaled
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0


free sulfur dioxide
600
400
200
0
0
100
200
300

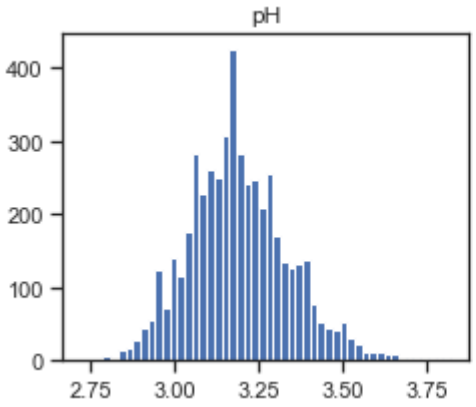

free sulfur dioxide_scaled
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

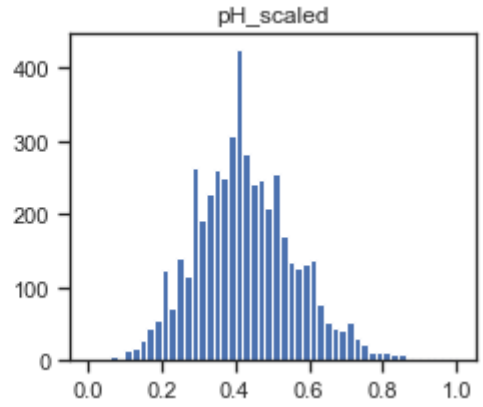

total sulfur dioxide
400
300
200
100
0
0
100
200
300
400

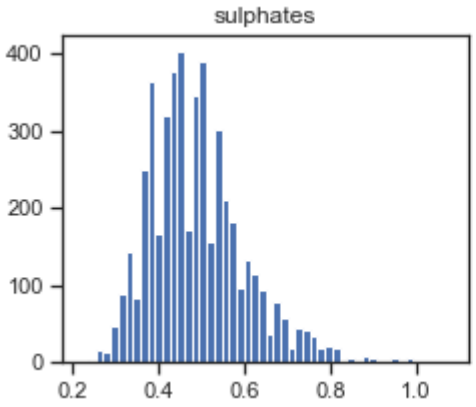

total sulfur dioxide_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

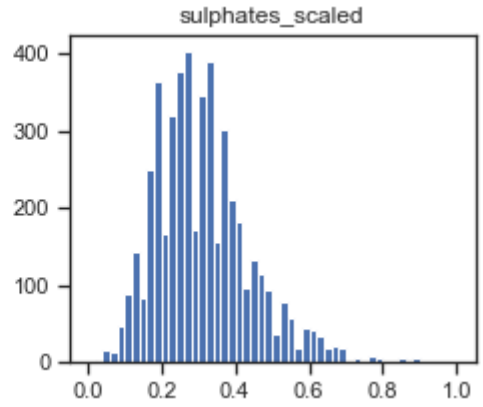

density
600
500
400
300
200
100
0
0.99
1.00
1.01
1.02
1.03
1.04

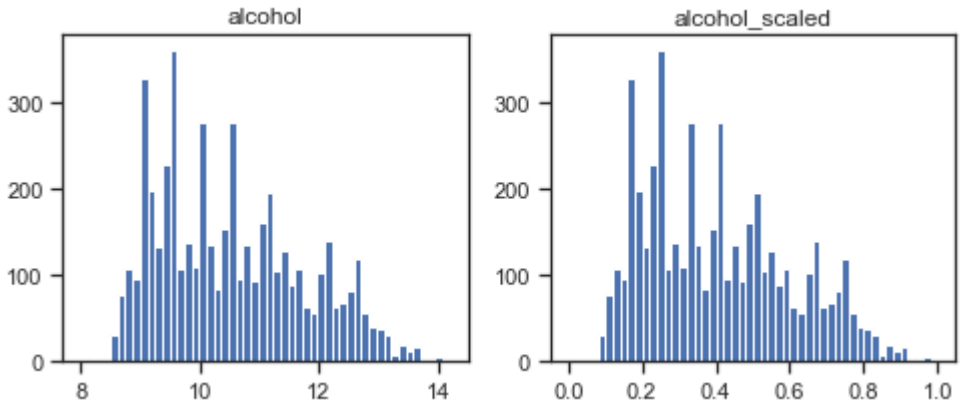
density_scaled
600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0


pH
400
300
200
100
0
2.75
3.00
3.25
3.50
3.75


pH_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0


sulphates
400
300
200
100
0
0.2
0.4
0.6
0.8
1.0


sulphates_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0



[165]:
```
data.head(10)
```

[165]:

	dataset	fixed acidity	volatile acidity	citric acid	residual sugar	\
0	TRAIN	7.0	0.27	0.36	20.7	
1	TRAIN	6.3	0.30	0.34	1.6	
2	TRAIN	8.1	0.28	0.40	6.9	
3	TRAIN	7.2	0.23	0.32	8.5	
4	TRAIN	7.2	0.23	0.32	8.5	
5	TRAIN	8.1	0.28	0.40	6.9	
6	TRAIN	6.2	0.32	0.16	7.0	
7	TRAIN	7.0	0.27	0.36	20.7	
8	TRAIN	6.3	0.30	0.34	1.6	
9	TRAIN	8.1	0.22	0.43	1.5	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	…	\
0	0.045	45.0	170.0	1.0010	3.00	…	
1	0.049	14.0	132.0	0.9940	3.30	…	
2	0.050	30.0	97.0	0.9951	3.26	…	
3	0.058	47.0	186.0	0.9956	3.19	…	
4	0.058	47.0	186.0	0.9956	3.19	…	
5	0.050	30.0	97.0	0.9951	3.26	…	
6	0.045	30.0	136.0	0.9949	3.18	…	
7	0.045	45.0	170.0	1.0010	3.00	…	
8	0.049	14.0	132.0	0.9940	3.30	…	
9	0.044	28.0	129.0	0.9938	3.22	…	

	volatile acidity_scaled	citric acid_scaled	residual sugar_scaled	\
0	0.186275	0.216867	0.308282	
1	0.215686	0.204819	0.015337	
2	0.196078	0.240964	0.096626	
3	0.147059	0.192771	0.121166	
4	0.147059	0.192771	0.121166	
5	0.196078	0.240964	0.096626	
6	0.235294	0.096386	0.098160	
7	0.186275	0.216867	0.308282	
8	0.215686	0.204819	0.015337	

```
9                0.137255            0.259036                0.013804

   chlorides_scaled  free sulfur dioxide_scaled  total sulfur dioxide_scaled␣
↪ \
0          0.106825                    0.149826                     0.373550
1          0.118694                    0.041812                     0.285383
2          0.121662                    0.097561                     0.204176
3          0.145401                    0.156794                     0.410673
4          0.145401                    0.156794                     0.410673
5          0.121662                    0.097561                     0.204176
6          0.106825                    0.097561                     0.294664
7          0.106825                    0.149826                     0.373550
8          0.118694                    0.041812                     0.285383
9          0.103858                    0.090592                     0.278422

   density_scaled  pH_scaled  sulphates_scaled  alcohol_scaled
0        0.267785   0.254545          0.267442        0.129032
1        0.132832   0.527273          0.313953        0.241935
2        0.154039   0.490909          0.255814        0.338710
3        0.163678   0.427273          0.209302        0.306452
4        0.163678   0.427273          0.209302        0.306452
5        0.154039   0.490909          0.255814        0.338710
6        0.150183   0.418182          0.290698        0.258065
7        0.267785   0.254545          0.267442        0.129032
8        0.132832   0.527273          0.313953        0.241935
9        0.128976   0.454545          0.267442        0.483871

[10 rows x 24 columns]
```

3.1.33. Вернем в набор данных целевой признак

[166]:
```
corr_cols_1 = scale_cols + ['quality']
corr_cols_1
```

[166]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

[167]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['quality']
```

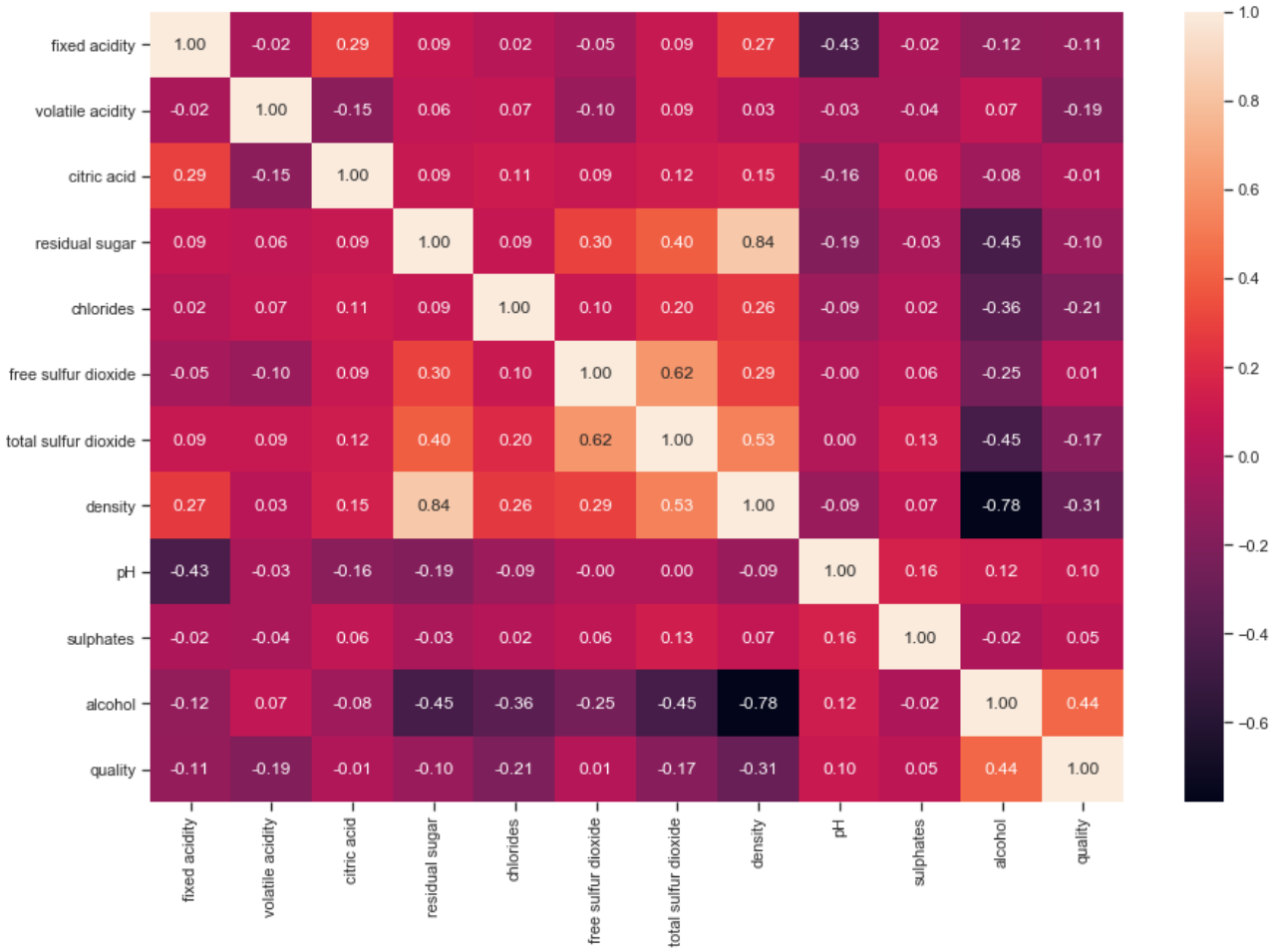
```
corr_cols_2
```

[167]:
```
['fixed acidity_scaled',
 'volatile acidity_scaled',
 'citric acid_scaled',
 'residual sugar_scaled',
 'chlorides_scaled',
 'free sulfur dioxide_scaled',
 'total sulfur dioxide_scaled',
 'density_scaled',
 'pH_scaled',
 'sulphates_scaled',
 'alcohol_scaled',
 'quality']
```

3.1.34. Построим корреляционную матрицу для обычных данных

[168]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

[168]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2a2957d0>
```

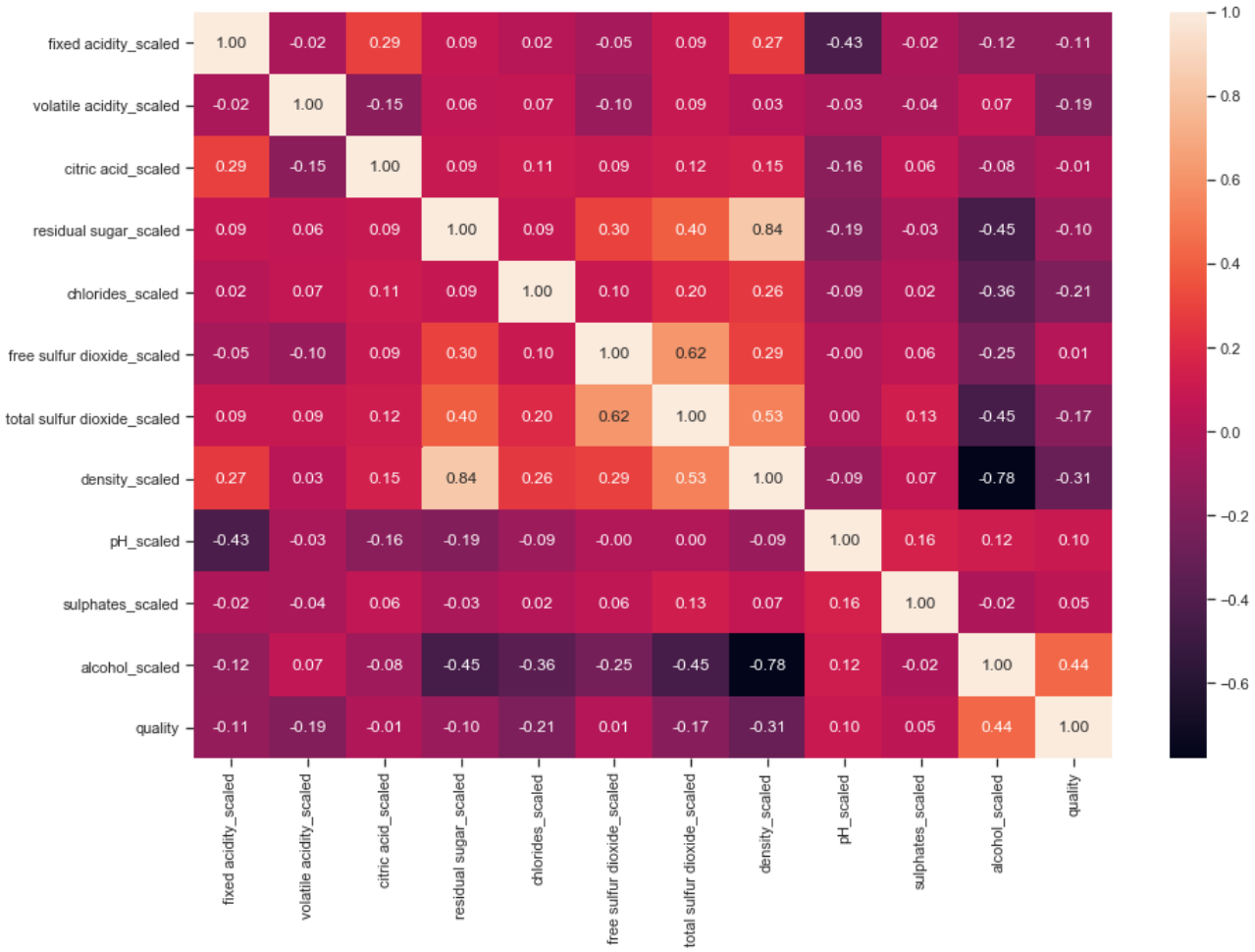


3.1.35. Построим корреляционную матрицу для масштабированных данных

Коэффициенты корреляции не изменились, так как распределение данных осталось таким же

[169]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

[169]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2d2677d0>



3.2. Выводы о коррелирующих признаках

1. Коэффициенты корреляции в данном наборе достаточно низкие. Этот факт будет иметь непосредственное влияние на качество наших моделей (в сторону ухудшения, к сожалению).
2. Если рассуждать чисто логически, то все представленные входные параметры влияют на качество алкоголя, так как они определяют его химический состав. С этой точки зрения для построения моделей мы можем использовать все 11 признаков. Однако, для улучшения качества моделей исключим признаки, которые могут быть зависимы друг от друга.
3. ‘alcohol’ и ‘density’ лучше всего коррелируют с целевым признаком, однако они очень сильно коррелируют друг с другом (|0.78|), что может означать зависимость между ними и

плохо влиять на построение моделей. ‘alcohol’ лучше коррелирует с целквым признаком, поэтому оставим его, а ‘density’ уберем.

4. ‘free sulfur’ и ‘total sulfur’ довольно неплохо коррелируют друг с другом (|0.62|), что логично, так как общий дикосид серы является сумма связной и свободной серы. У них прослеживается явная заивисмость. Уберем ‘free sulfur’ из признаков для построения модели.

3.2.1. Бинаризация данных

Так как наш целевой признак ‘quality’ включает в себя 7 значений, бинарная классификация невозможна.

Чтобы бинаризировать 7 различных значений целевого признака, мы вместо одного целевого столбца ‘quality’ создаем 7 столбцов (каждый столбец соответствует определенному значению выходного параметра ‘quality’).

Каждый из семи столбцов является бинарным, то есть принимает значение “1”, когда вино имеет оценку качества, соответствующую столбцу, и “0” — во всех остальных случаях.

Все семь столбцов мы создали для наглядности и удобства. Как уже было скзаано выше, для задачи классификации мы будем использовать только оценку “6” и “7”.

[190]:
```
qual = pd.concat([train['quality'], test['quality']])
```

[191]:
```
def code_myohe(data, column):
    for i in data[column].unique():
        data[column + '=' + str(i)] = (data[column] == i).astype(int)
```

[192]:
```
code_myohe(data, 'quality')
data.head()
```

[192]:

	dataset	fixed acidity	volatile acidity	citric acid	residual sugar	\
0	TRAIN	7.0	0.27	0.36	20.7	
1	TRAIN	6.3	0.30	0.34	1.6	
2	TRAIN	8.1	0.28	0.40	6.9	
3	TRAIN	7.2	0.23	0.32	8.5	
4	TRAIN	7.2	0.23	0.32	8.5	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	…	\
0	0.045	45.0	170.0	1.0010	3.00	…	
1	0.049	14.0	132.0	0.9940	3.30	…	
2	0.050	30.0	97.0	0.9951	3.26	…	
3	0.058	47.0	186.0	0.9956	3.19	…	
4	0.058	47.0	186.0	0.9956	3.19	…	

	pH_scaled	sulphates_scaled	alcohol_scaled	quality=6	quality=5	\
0	0.254545	0.267442	0.129032	1	0	
1	0.527273	0.313953	0.241935	1	0	
2	0.490909	0.255814	0.338710	1	0	
3	0.427273	0.209302	0.306452	1	0	
4	0.427273	0.209302	0.306452	1	0	

	quality=7	quality=8	quality=4	quality=3	quality=9
0	0	0	0	0	0

```
1          0          0          0          0          0
2          0          0          0          0          0
3          0          0          0          0          0
4          0          0          0          0          0

[5 rows x 31 columns]
```

[193]:
```
data['quality'] = qual
```

3.2.2. Создадим класс для сохранения и визуализации метрик

[194]:
```
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):
        """

        """
        #
        self.df.drop(self.df[(self.df['metric']==metric)&(self.
↪df['alg']==alg)].index, inplace = True)
        #
        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):
        """

        """
        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)):
        """

        """
        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()
```

3.2.3. Снова произведем разделение обучающей и тестовой выборки

Теперь данные в этих выборках масштабированные

[195]:
```
#
train_data = data[data['dataset']=='TRAIN']
test_data = data[data['dataset']=='TEST']
train_data.shape, test_data.shape
```

[195]:
```
((3499, 31), (1398, 31))
```

3.3. Решаем задачу классификации

3.3.1. Определим признаки для задачи классификации

[372]:
```
#
task_clas_cols = ['fixed acidity_scaled', 'volatile acidity_scaled',␣
 ↪'citric acid_scaled', 'residual sugar_scaled',
       'chlorides_scaled', 'total sulfur dioxide_scaled',
       'pH_scaled', 'sulphates_scaled', 'alcohol_scaled']
```

3.3.2. Создадим выборки для задачи классификации

Будем осуществлять классификайию для класса качества “6” и “7”.

[373]:
```
#
clas_X_train = train_data[task_clas_cols]
clas_X_test = test_data[task_clas_cols]

clas_Y6_train = train_data['quality=6']
clas_Y6_test = test_data['quality=6']

clas_Y7_train = train_data['quality=7']
clas_Y7_test = test_data['quality=7']

clas_X_train.shape, clas_X_test.shape, clas_Y3_train.shape, clas_Y3_test.
 ↪shape
```

[373]:
```
((3499, 9), (1398, 9), (3499,), (1398,))
```

3.3.3. Создаем словарь моделей, которые будем строить

[482]:
```
#
clas_models = {'LogR': LogisticRegression(),
               'KNN_5':KNeighborsClassifier(n_neighbors=5),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

[483]:
```
#
clasMetricLogger = MetricLogger()
```

3.3.4. Создаем функцию для отрисовки ROC-кривой наших моделей

[484]:
```
#       ROC-
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()
```

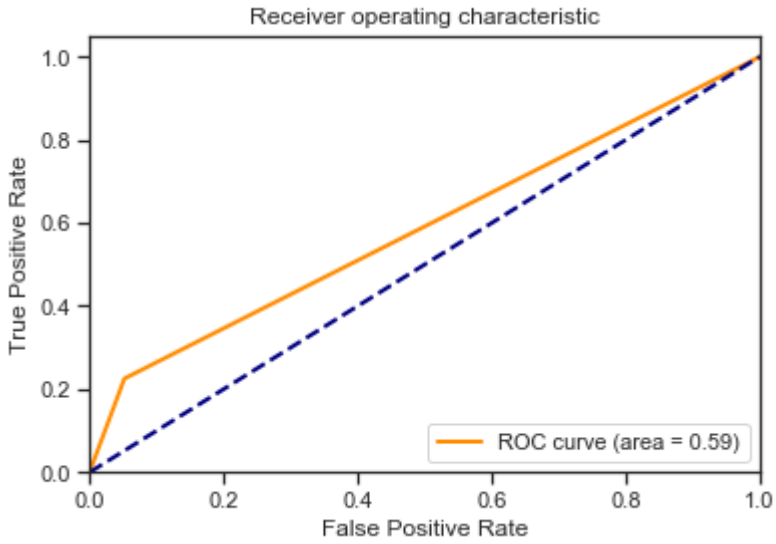
3.3.5. Построим модели для класса качества “7”

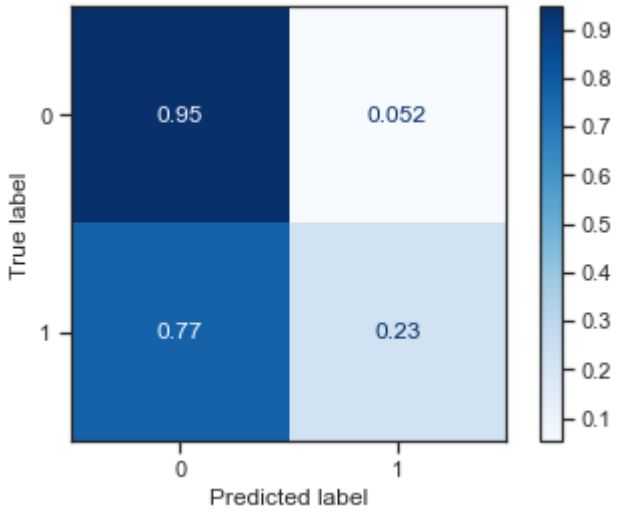
[485]:
```
def clas_train_model7(model_name, model, clasMetricLogger):
    model.fit(clas_X_train, clas_Y7_train)
    Y_pred = model.predict(clas_X_test)
    precision = precision_score(clas_Y7_test.values, Y_pred)
    recall = recall_score(clas_Y7_test.values, Y_pred)
    f1 = f1_score(clas_Y7_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y7_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_Y7_test.values, Y_pred)

    plot_confusion_matrix(model, clas_X_test, clas_Y7_test.values,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

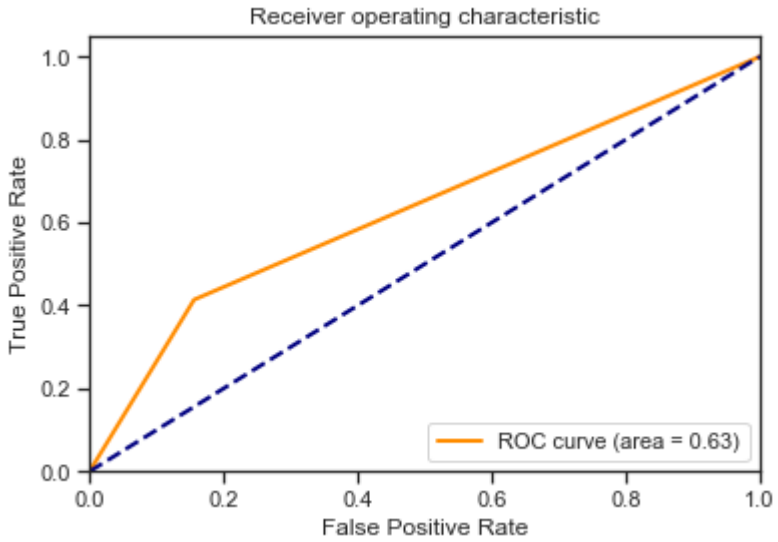
[486]:
```
for model_name, model in clas_models.items():
    clas_train_model7(model_name, model, clasMetricLogger)
```

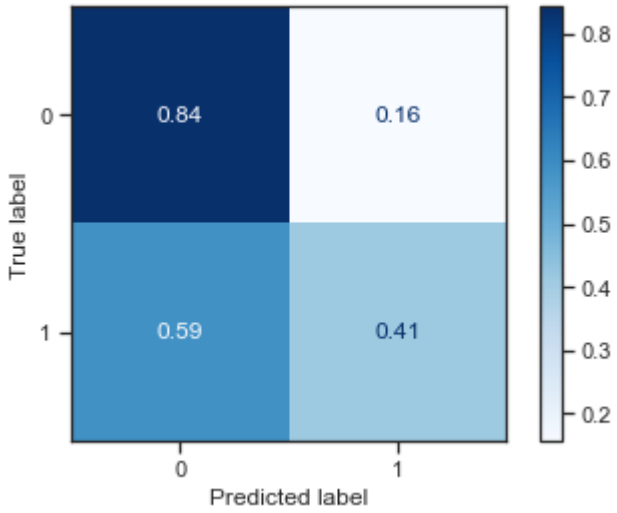
```
*****************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None,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



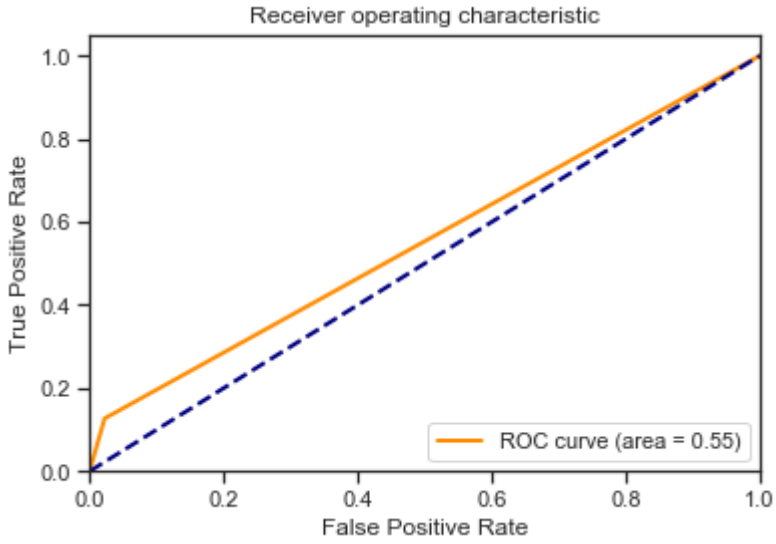


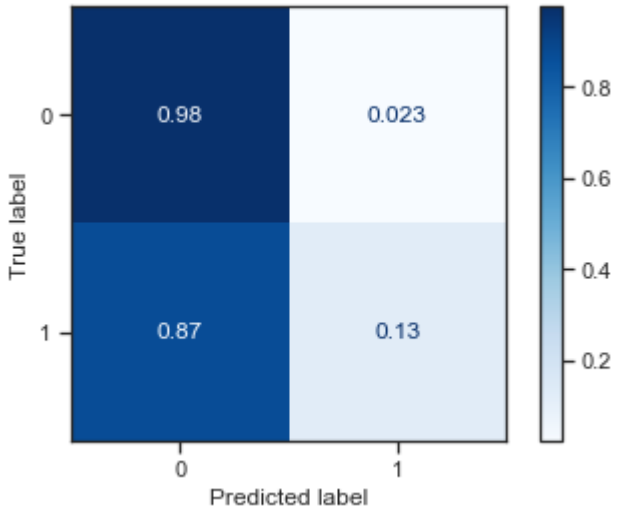

```
**************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
**************************************************
```



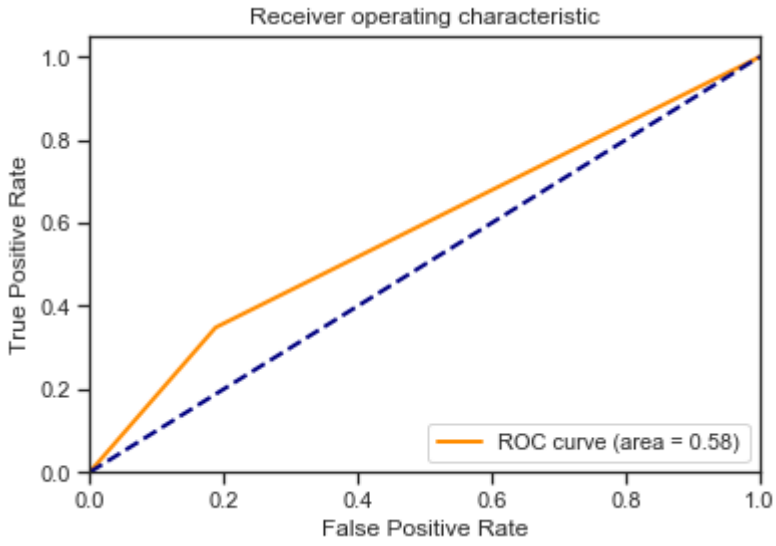


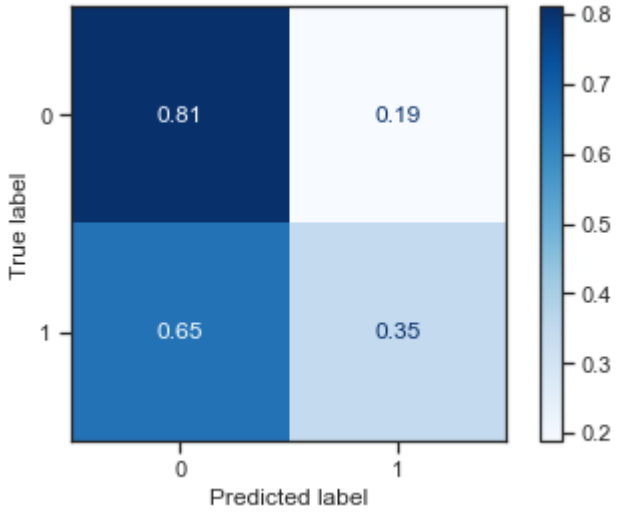

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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



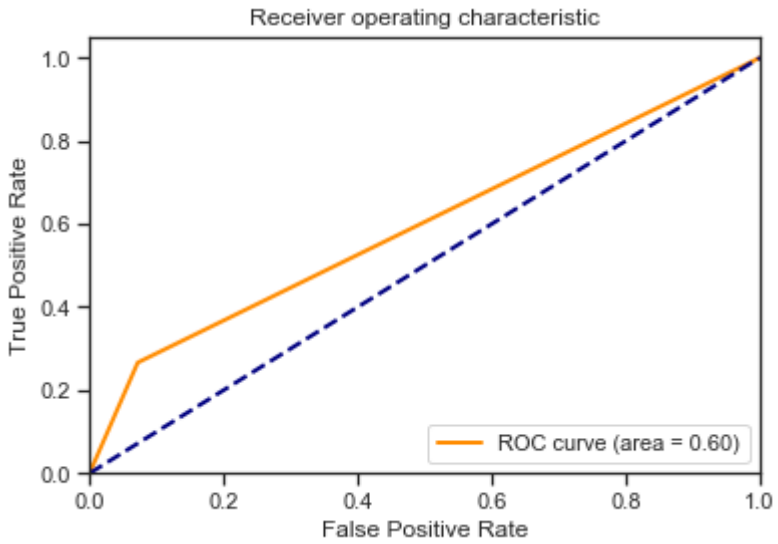


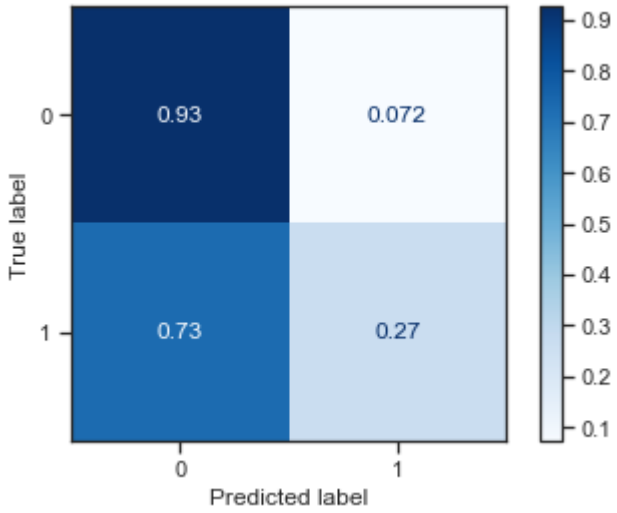

```
*******************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
           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



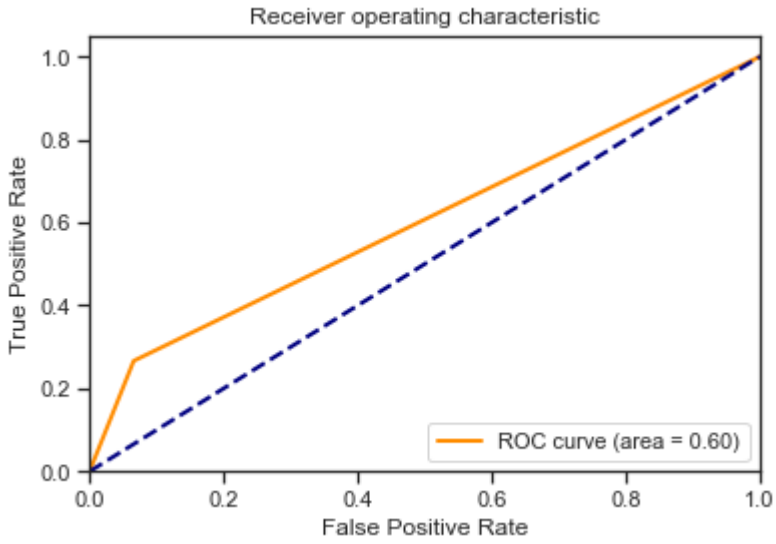


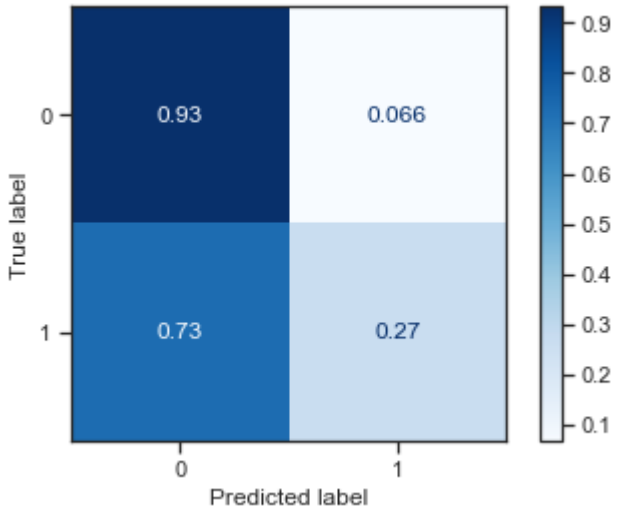

```
*******************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, 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)
**********************************************************
```





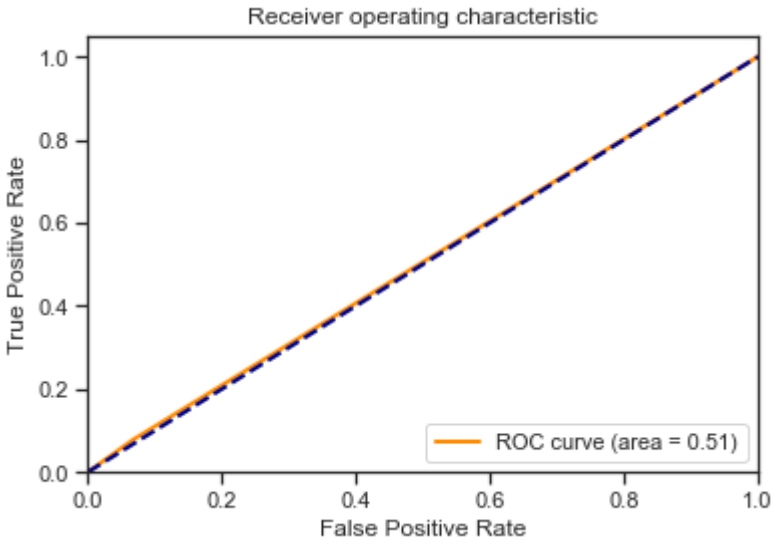
3.3.6. Построим модели для класса качества “6”

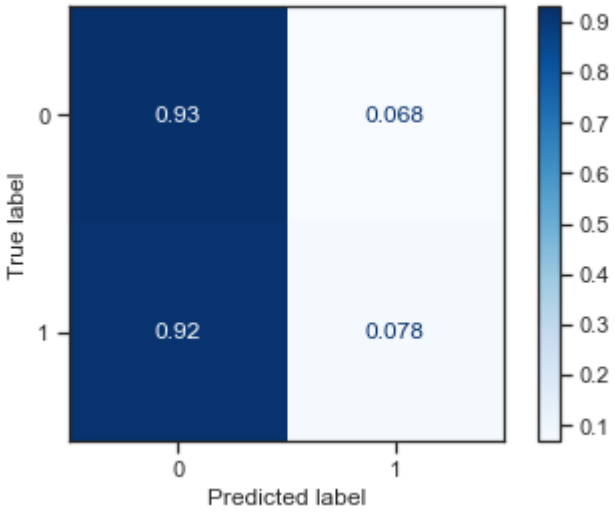
[487]:
```
def clas_train_model6(model_name, model, clasMetricLogger):
    model.fit(clas_X_train, clas_Y6_train)
    Y_pred = model.predict(clas_X_test)
    precision = precision_score(clas_Y6_test.values, Y_pred)
    recall = recall_score(clas_Y6_test.values, Y_pred)
    f1 = f1_score(clas_Y6_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y6_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_Y6_test.values, Y_pred)

    plot_confusion_matrix(model, clas_X_test, clas_Y6_test.values,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

[488]:
```
for model_name, model in clas_models.items():
    clas_train_model6(model_name, model, clasMetricLogger)
```

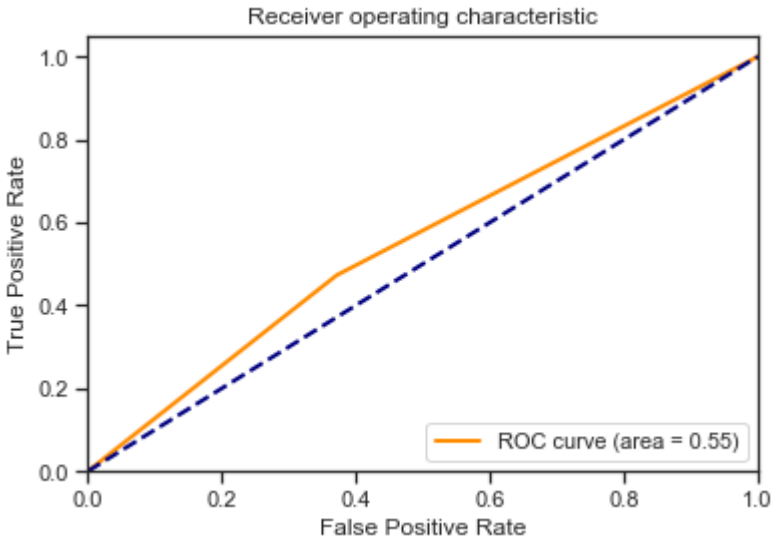
```
************************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
```

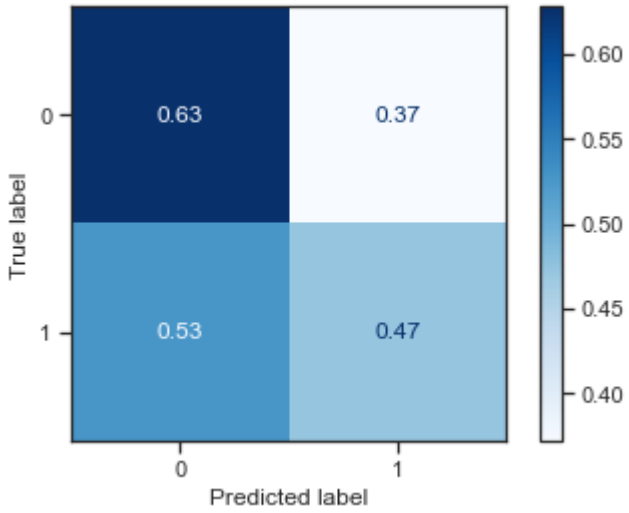
```
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
******************************************************
```



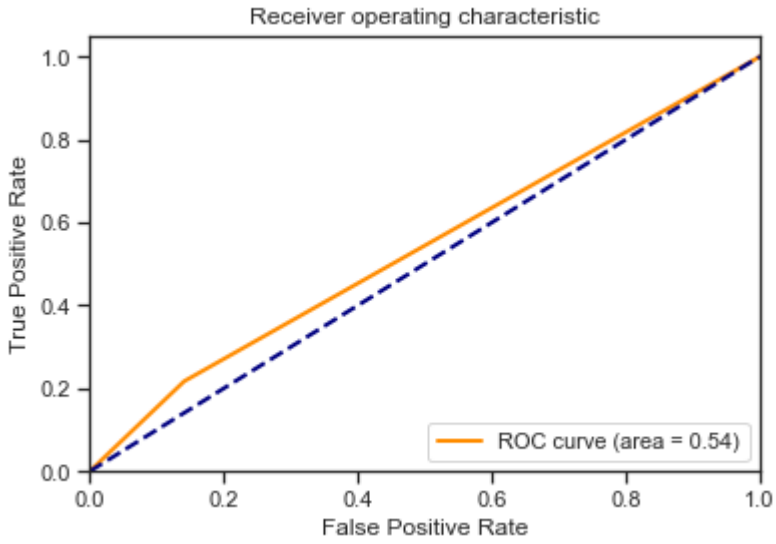


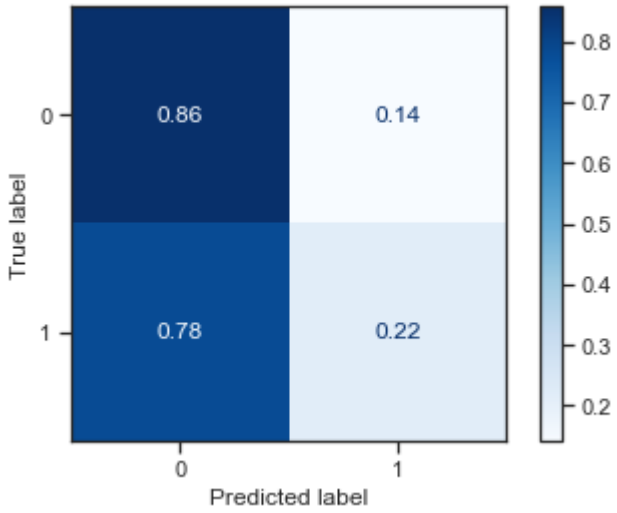

```
******************************************************
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
******************************************************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



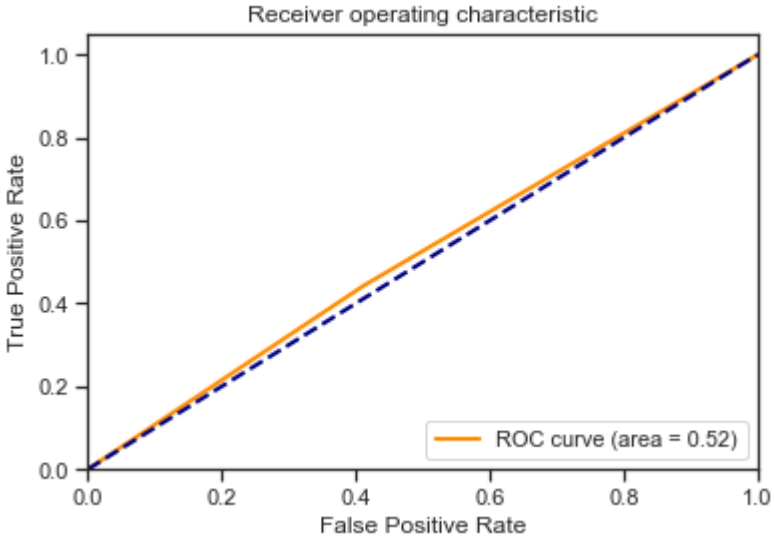


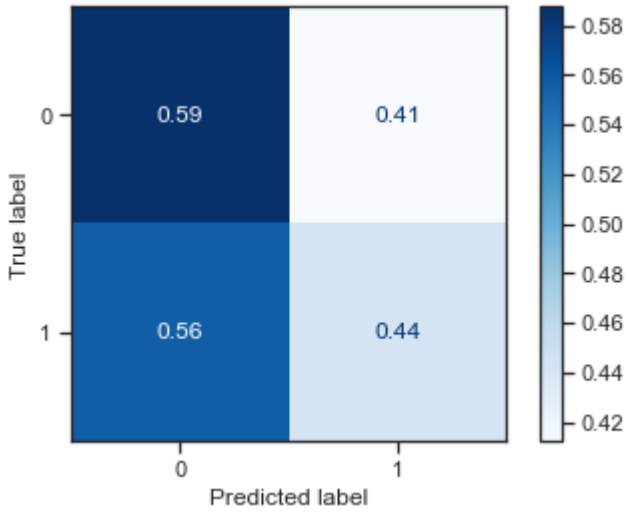

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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



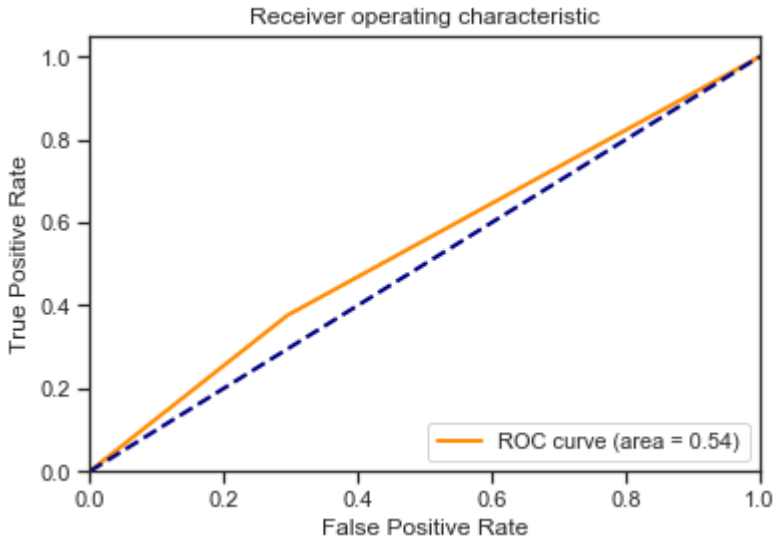


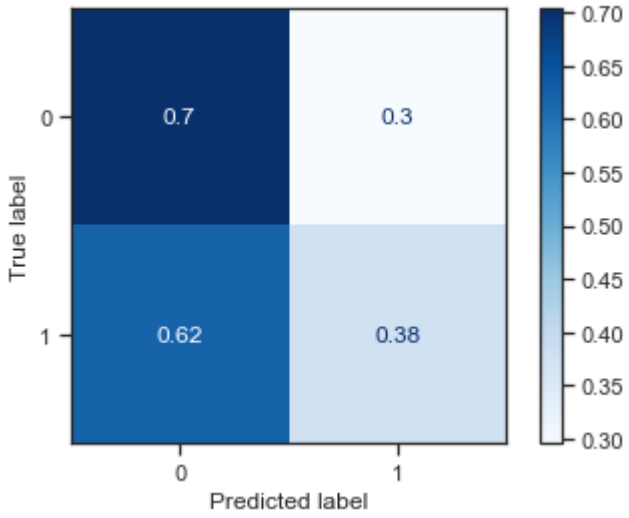

```
******************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
            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



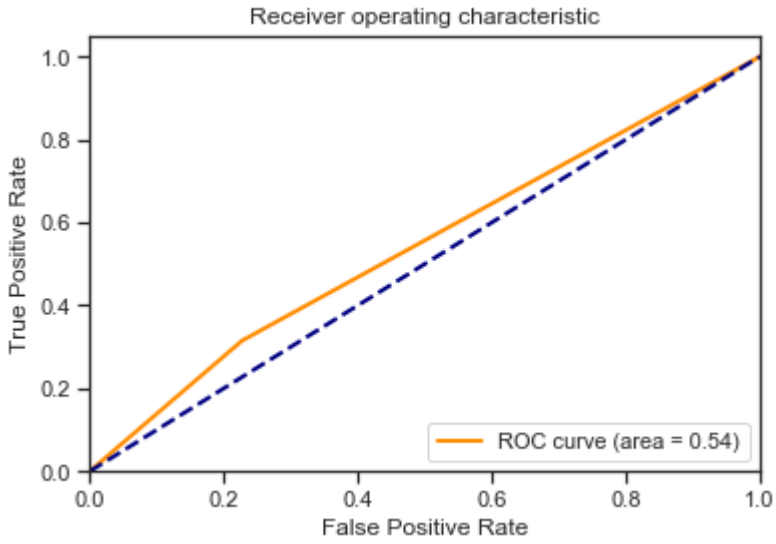


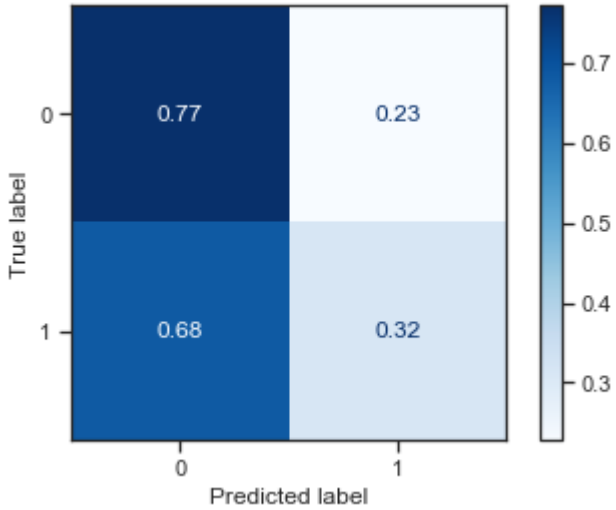

```
**************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, 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)
*******************************************************
```





3.4. Решаем задачу регрессии

3.4.1. Выбор признаков для задачи регрессии

Целевым признаком будет ‘alcohol’. По факту, этот признак, как и качества, зависит от всех остальных характеристик химического состава вина.

Для решения задачи регресси мы все же возьмем наиболее коррелирующие признаки с целевым. Их всего четыре — ‘quality’, ‘density’, ‘residual sugar’, ‘total sulfur’.

Посмотрим, насколько эти признаки коррелируют между собой. Признак ‘density’ отлично коррелирует со всеми другими, что может означать зависимость. Поэтому его мы уберем из признаков для построения моделей.

[489]:
```
#
task_regr_cols = ['residual sugar_scaled', 'total sulfur dioxide_scaled',␣
 ↪'quality']
```

[490]:
```
#
regr_X_train = train_data[task_regr_cols]
regr_X_test = test_data[task_regr_cols]
regr_Y_train = train_data['alcohol']
regr_Y_test = test_data['alcohol']
regr_X_train.shape, regr_X_test.shape, regr_Y_train.shape, regr_Y_test.shape
```

[490]:
```
((3499, 3), (1398, 3), (3499,), (1398,))
```

[491]:
```
#
regr_models = {'LR': LinearRegression(),
               'KNN_5':KNeighborsRegressor(n_neighbors=5),
               'SVR':SVR(),
               'Tree':DecisionTreeRegressor(),
               'RF':RandomForestRegressor(),
               'GB':GradientBoostingRegressor()}
```

[492]:
```
#
regrMetricLogger = MetricLogger()
```

[493]:
```
def regr_train_model(model_name, model, regrMetricLogger):
    model.fit(regr_X_train, regr_Y_train)
    Y_pred = model.predict(regr_X_test)

    mae = mean_absolute_error(regr_Y_test, Y_pred)
    mse = mean_squared_error(regr_Y_test, Y_pred)
    r2 = r2_score(regr_Y_test, Y_pred)

    regrMetricLogger.add('MAE', model_name, mae)
    regrMetricLogger.add('MSE', model_name, mse)
    regrMetricLogger.add('R2', model_name, r2)

    print('************************************************************')
    print(model)
    print()
    print('MAE={}, MSE={}, R2={}'.format(
        round(mae, 3), round(mse, 3), round(r2, 3)))
    print('************************************************************')
```

[494]:
```
for model_name, model in regr_models.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,␣
 ↪normalize=False)

MAE=0.822, MSE=1.096, R2=0.35
**************************************************
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
```

```
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')

MAE=0.854, MSE=1.202, R2=0.287
****************************************************
****************************************************
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)

MAE=0.879, MSE=1.267, R2=0.248
****************************************************
****************************************************
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', 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')

MAE=1.039, MSE=1.853, R2=-0.099
****************************************************
****************************************************
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      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)

MAE=0.868, MSE=1.24, R2=0.264
****************************************************
****************************************************
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls',␣
 ↪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)

MAE=0.783, MSE=1.01, R2=0.401
****************************************************
```

3.5. Подбор гиперпараметров для выбранных моделей

3.5.1. Задача классификации

Возьмем класс качества “7”

[495]:
```
clas_X_train.shape
```

[495]:
```
(3499, 9)
```

[496]:
```
tree_param = {'criterion':['gini','entropy'], 'splitter': ['best',␣
↪'random'], 'max_depth':
↪[4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150, 250, 500, 1000]}

clf_gs = GridSearchCV(DecisionTreeClassifier(), tree_param, cv=5)
clf_gs.fit(clas_X_train, clas_Y7_train)
```

[496]:
```
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={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20,␣
↪30,
                                       40, 50, 70, 90, 120, 150, 250, 500,
                                       1000],
                         'splitter': ['best', 'random']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

[497]:
```
clf_gs.best_estimator_
```

[497]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, 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='random')
```

[498]:
```
clf_gs.best_params_
```

[498]: {'criterion': 'gini', 'max_depth': 5, 'splitter': 'random'}

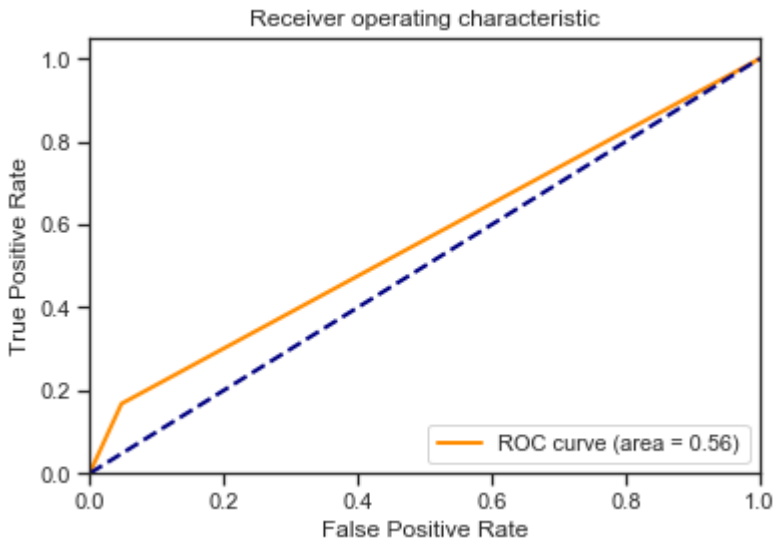
[499]:
```
clas_models_grid = {'entropy':clf_gs.best_estimator_}
```

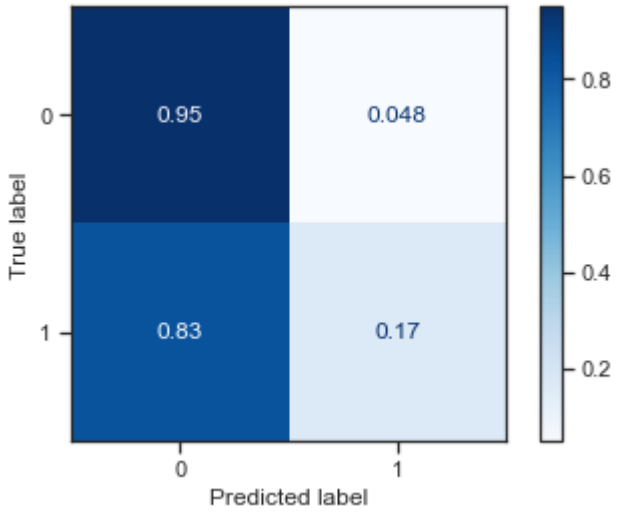
3.5.2. Сравнение качества исходных моделей с качеством моделей при найденных гиперпараметрах

3.5.3. Модель с подбором гиперпараметров

[500]:
```
for model_name, model in clas_models_grid.items():
    clas_train_model7(model_name, model, clasMetricLogger)
```

```
************************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, 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='random')
************************************************************
```





3.6. Подбор гиперпараметров для выбранных моделей

3.6.1. Задача регрессии

[501]:
```
n_range = np.array(range(1,1000,50))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

[501]:
```
[{'n_neighbors': array([  1,  51, 101, 151, 201, 251, 301, 351, 401, 451,␣
 ↪501,
551, 601,
         651, 701, 751, 801, 851, 901, 951])}]
```

[502]:
```
%%time
regr_gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters, cv=5,␣
 ↪scoring='neg_mean_squared_error')
regr_gs.fit(regr_X_train, regr_Y_train)
```

```
CPU times: user 5.67 s, sys: 167 ms, total: 5.84 s
Wall time: 5.87 s
```

[502]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsRegressor(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,  51, 101, 151, 201, 251,
301, 351, 401, 451, 501, 551, 601,
       651, 701, 751, 801, 851, 901, 951])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
```

```
                    scoring='neg_mean_squared_error', verbose=0)
```

[503]:
```
#
regr_gs.best_estimator_
```

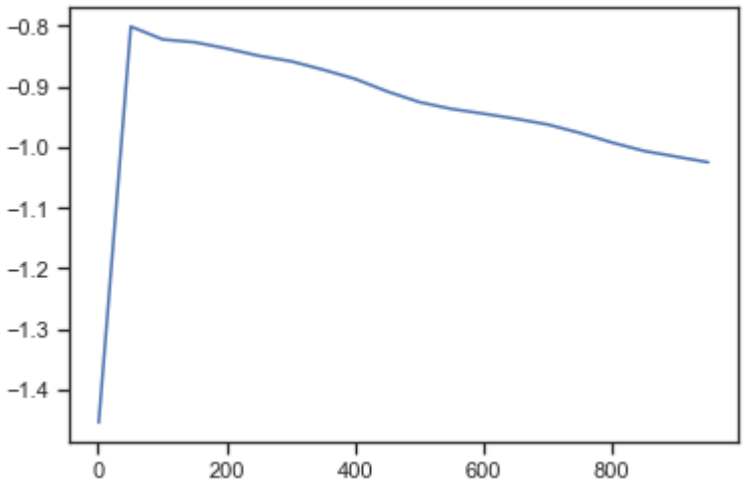
[503]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=51, p=2,
                    weights='uniform')
```

[504]:
```
#
regr_gs.best_params_
```

[504]:
```
{'n_neighbors': 51}
```

[505]:
```
#                                       -
plt.plot(n_range, regr_gs.cv_results_['mean_test_score'])
```

[505]:
```
[<matplotlib.lines.Line2D at 0x1a3606b7d0>]
```



3.6.2. Сравнение качества исходных моделей с качеством моделей при найденных гиперпараметрах

3.6.3. Модель с подбором гиперпараметров

[506]:
```
regr_models_grid = {'KNN_51':regr_gs.best_estimator_}
```

[507]:
```
for model_name, model in regr_models_grid.items():
    regr_train_model(model_name, model, regrMetricLogger)
```

```
**************************************************
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=51, p=2,
                    weights='uniform')

MAE=0.789, MSE=1.031, R2=0.389
**************************************************
```

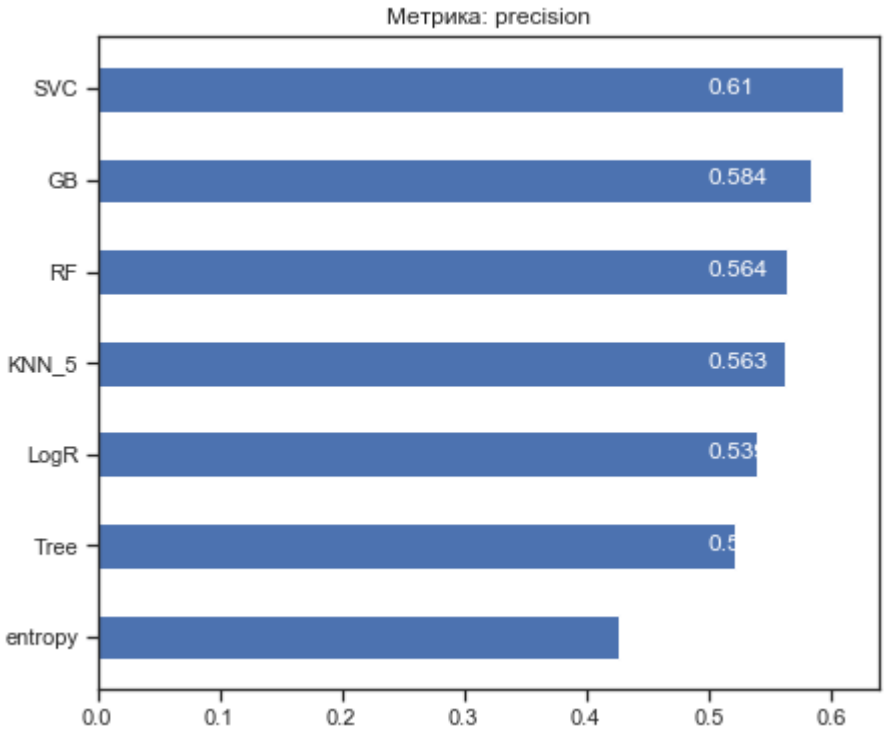
3.7. Формирование выводов о качестве построенных моделей на основе выбранных метрик

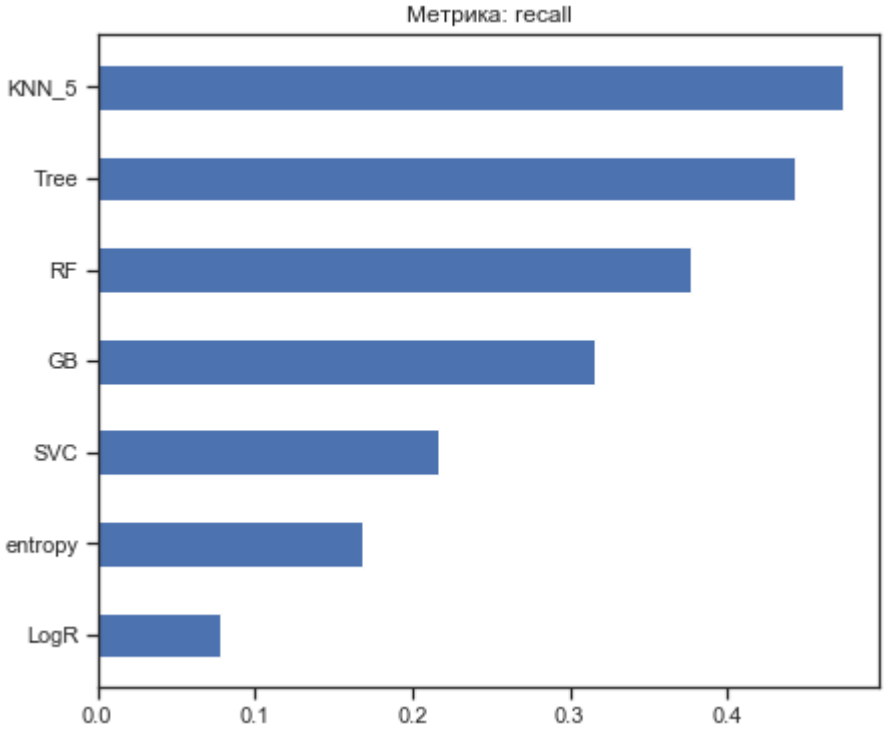
3.7.1. Задача классификации

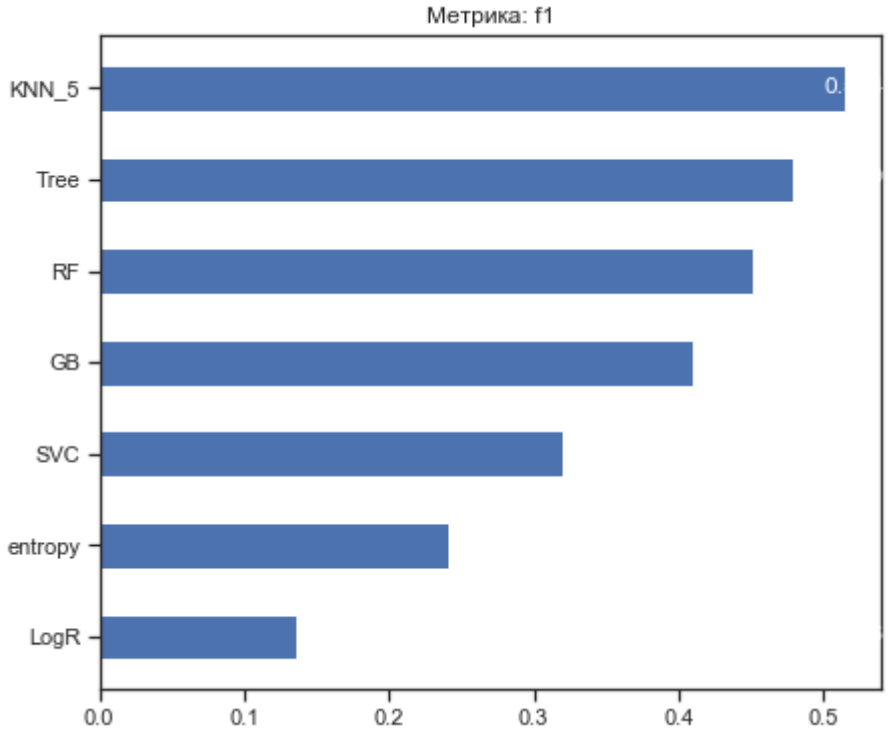
[508]:
```
#
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

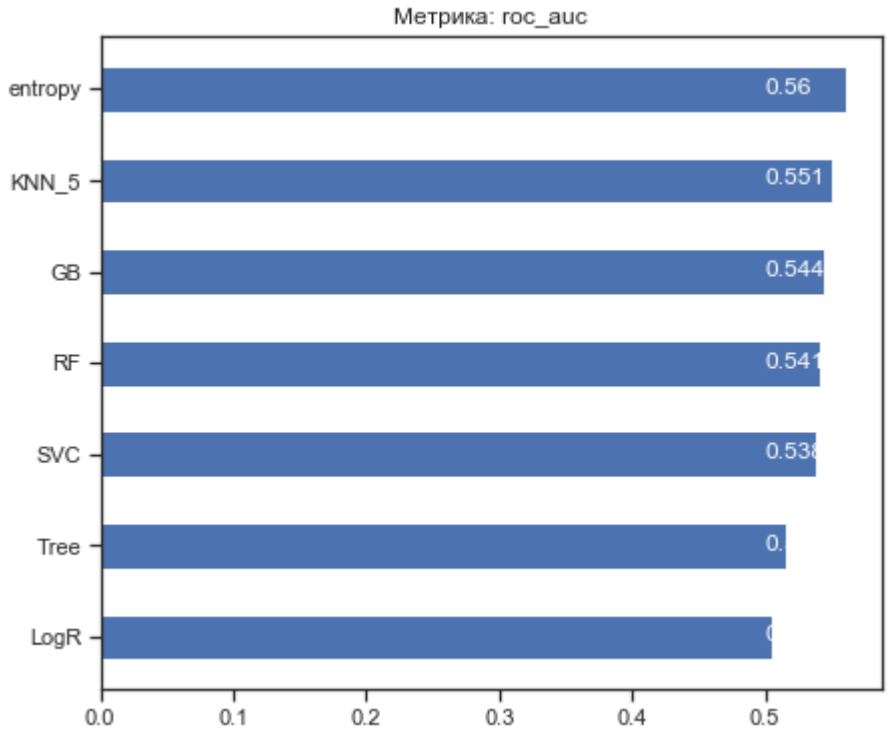
[508]:
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

[509]:
```
#
for metric in clas_metrics:
    clasMetricLogger.plot('    : ' + metric, metric, figsize=(7, 6))
```




Метрика: recall
KNN_5
Tree
RF
GB
SVC
entropy
LogR
0.0
0.1
0.2
0.3
0.4


Метрика: f1
KNN_5
Tree
RF
GB
SVC
entropy
LogR
0.0
0.1
0.2
0.3
0.4
0.5



3.8. Вывод

На основе двух из четырех метрик лучшим оказался метод построения модели на основе К-ближайших соседей (в нашем случае — 5-ти). Однако, подобранный гиперпараметр для метода построения модели решающим деревом оказался далеко не идеальным. Можно подобрать лучше. Но вряд ли способ построения методом решающих деревьев окажется наиболее точным.

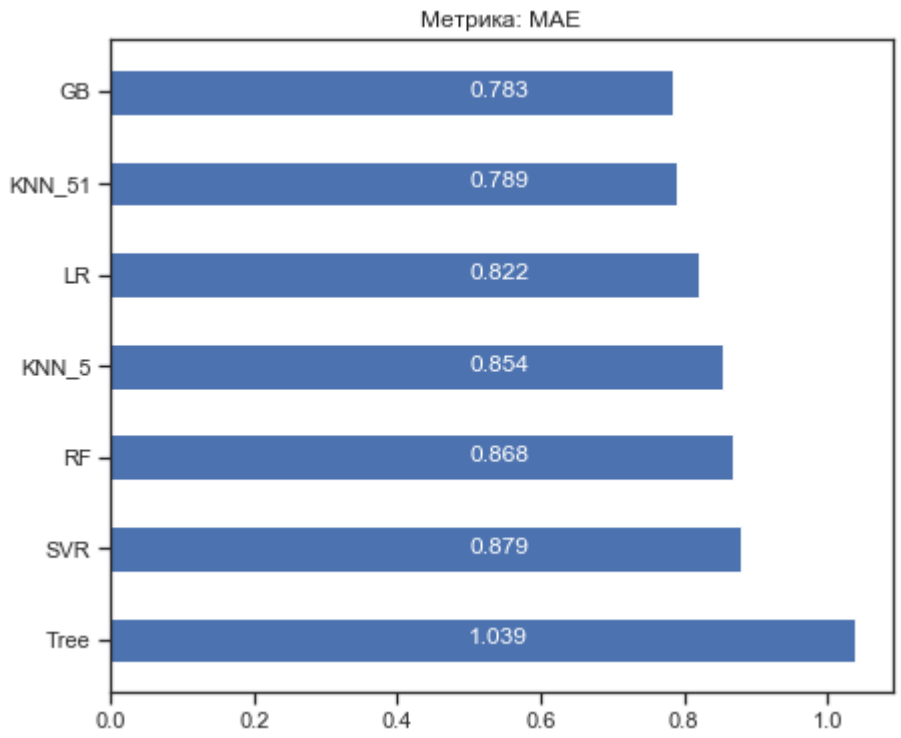
3.9. Формирование выводов о качестве построенных моделей на основе выбранных метрик

3.9.1. Задача регрессии

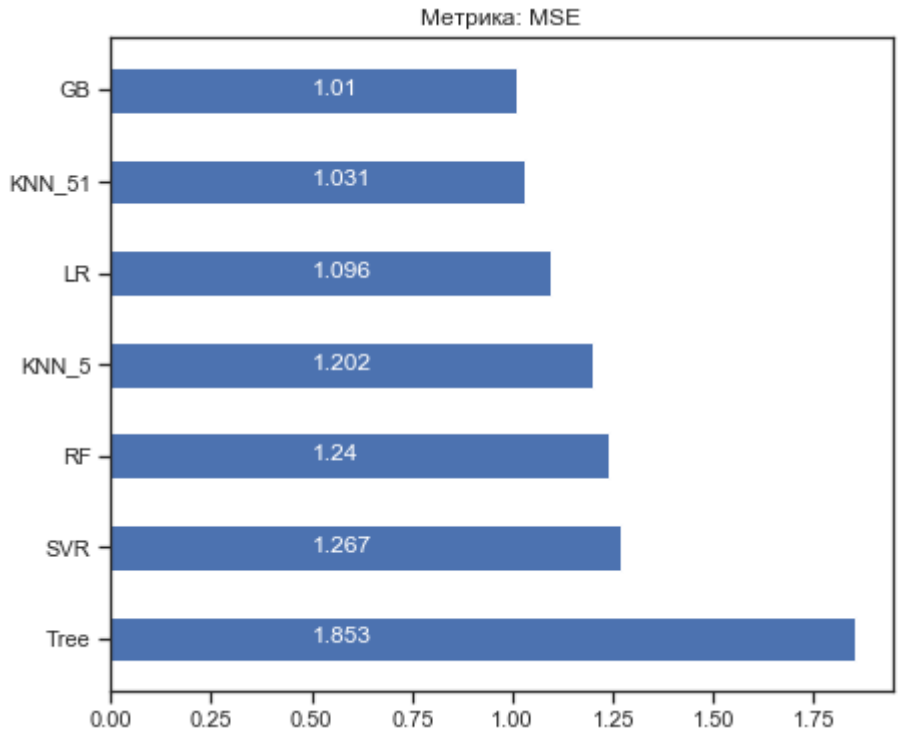
[510]:
```
#
regr_metrics = regrMetricLogger.df['metric'].unique()
regr_metrics
```

[510]:
```
array(['MAE', 'MSE', 'R2'], dtype=object)
```

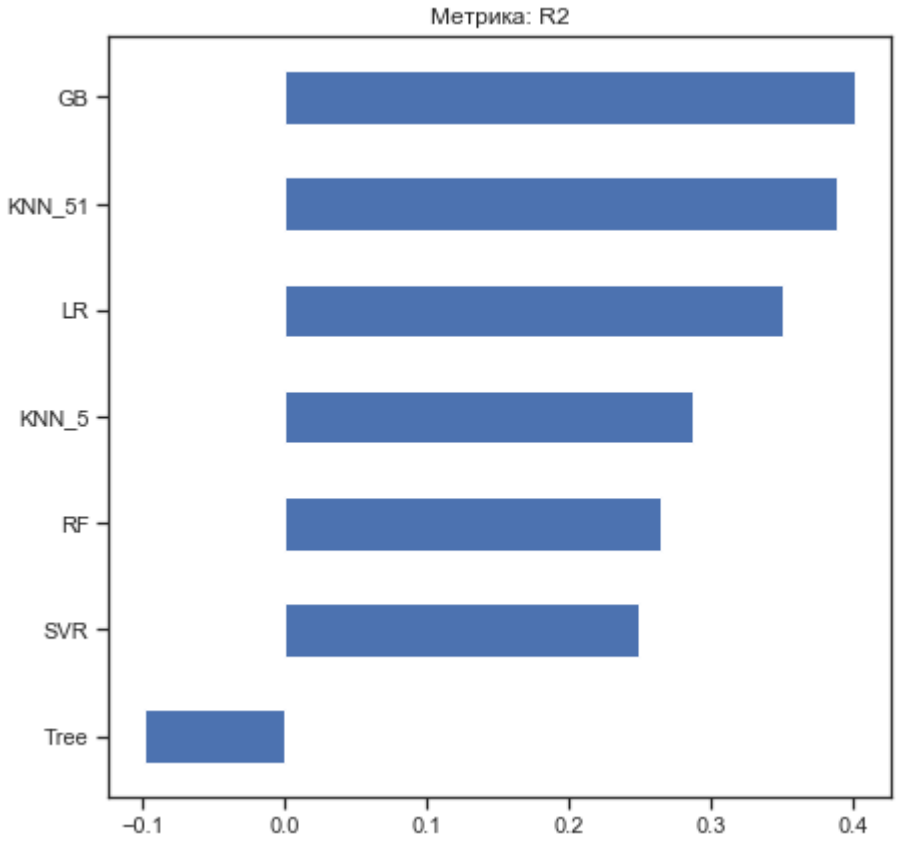
[511]:
```
regrMetricLogger.plot('    : ' + 'MAE', 'MAE', ascending=False, figsize=(7,␣
↪6))
```



[512]:
```
regrMetricLogger.plot('    : ' + 'MSE', 'MSE', ascending=False, figsize=(7,␣
↪6))
```



[513]:
```
regrMetricLogger.plot('    : ' + 'R2', 'R2', ascending=True, figsize=(7, 7))
```



3.10. Вывод

На основе всех трех метрик лучшими оказались модели на основе градиентного бустинга. При этом при более точном подборе гиперпараметров для метода K-ближайших соседей можно добиться +- такой же точности.