Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №5
по дисциплине
«Методы машинного обучения»
на тему
«Линейные модели, SVM и деревья решений.»

Выполнил:
студент группы ИУ5-64Б
Береговая Д.

_____

Москва — 2019 г.

1. Цель лабораторной работы

Изучение линейных моделей, SVM и деревьев решений. # Задание 1. Выберите набор данных (датасет) для решения задачи классификации или регрессии. 2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков. 3. С использованием метода train_test_split разделите выборку на обучающую и тестовую. 4. Обучите следующие модели: - одну из линейных моделей; - SVM; - дерево решений. 5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

2. Ход выполнения работы

2.1. 1) Набор данных для решения задачи классификации или регрессии

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

Датасет состоит из одного файла: - wine.csv

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)

2.1.2. Импортируем библиотеки

[1]:
```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
 ↪mean_squared_log_error, median_absolute_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")
```

2.1.3. Отрисовка ROC-кривой

[2]:
```
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()
```

2.2. 2) Разделение выборки на обучающую и тестовую

[3]:
```
def split(filehandler, delimiter=';', row_limit=3500,
          output_name_template='wine%s.csv', output_path='.',␣
 ↪keep_headers=True):
    import csv
    reader = csv.reader(filehandler, delimiter=delimiter)
    current_piece = 1
    current_out_path = os.path.join(
        output_path,
        output_name_template % current_piece
    )
    current_out_writer = csv.writer(open(current_out_path, 'w'),␣
 ↪delimiter=delimiter)
    current_limit = row_limit
    if keep_headers:
        headers = next(reader)
        current_out_writer.writerow(headers)
    for i, row in enumerate(reader):
        if i + 1 > current_limit:
            current_piece += 1
            current_limit = row_limit * current_piece
            current_out_path = os.path.join(
```

```
                output_path,
                output_name_template % current_piece
            )
            current_out_writer = csv.writer(open(current_out_path, 'w'),␣
↪delimiter=delimiter)
            if keep_headers:
                current_out_writer.writerow(headers)
        current_out_writer.writerow(row)
```

[4]:
```
split(open('wine.csv', 'r'));
```

[5]:
```
os.rename('wine1.csv', 'wine_Train.csv')
os.rename('wine2.csv', 'wine_Test.csv')
```

[6]:
```
#           :
train = pd.read_csv('wine_Train.csv', sep=";")
#           :
test = pd.read_csv('wine_Test.csv', sep=";")
```

Проверим правильность создания обучающей и тестовой выборок

[7]:
```
train.head()
```

[7]:

	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

[8]:
```
test.head()
```

[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides
0	6.0	0.28	0.27	15.5	0.036
1	6.7	0.24	0.36	8.4	0.042
2	6.7	0.29	0.45	14.3	0.054
3	6.9	0.33	0.31	4.2	0.040
4	6.5	0.16	0.34	1.4	0.029

```
   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 31.0                 134.0  0.99408  3.19       0.44
1                 42.0                 123.0  0.99473  3.34       0.52
2                 30.0                 181.0  0.99869  3.14       0.57
3                 21.0                  93.0  0.98960  3.18       0.48
4                 29.0                 133.0  0.99108  3.33       0.64

   alcohol  quality
0     13.0        7
1     10.9        6
2      9.1        5
3     13.4        7
4     11.5        7
```

2.3. 3) Проведение разведочного анализа данных

[9]:
```
train.shape, test.shape
```

[9]:
```
((3500, 12), (1398, 12))
```

2.3.1. Проверим, одинаковы ли типы данных в столбцах обучающего и тестового датасета

[12]:
```
train.dtypes
```

[12]:
```
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
```

[13]:
```
test.dtypes
```

[13]:
```
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
```

Проверяем датасеты на наличие пустых значений:

[14]:
```
train.isnull().sum()
```

[14]:
```
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
```

[15]:
```
test.isnull().sum()
```

[15]:
```
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
```

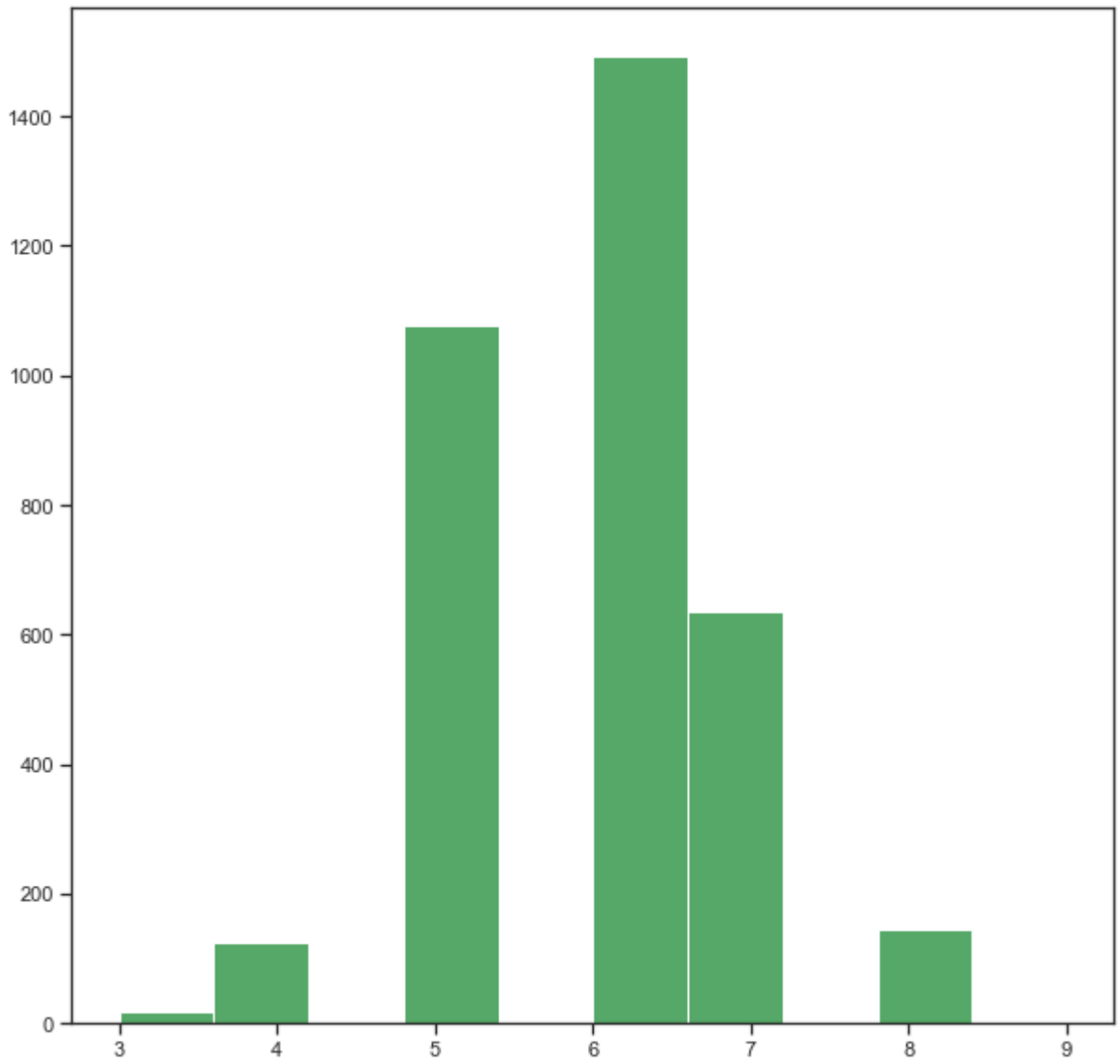
2.3.2. Уникальные значения целевого признака

[16]:
```
train['quality'].unique()
```

[16]:
```
array([6, 5, 7, 8, 4, 3, 9])
```

2.3.3. Рассмотри распределение цеелвых значений в обучающей и тестовой выборках

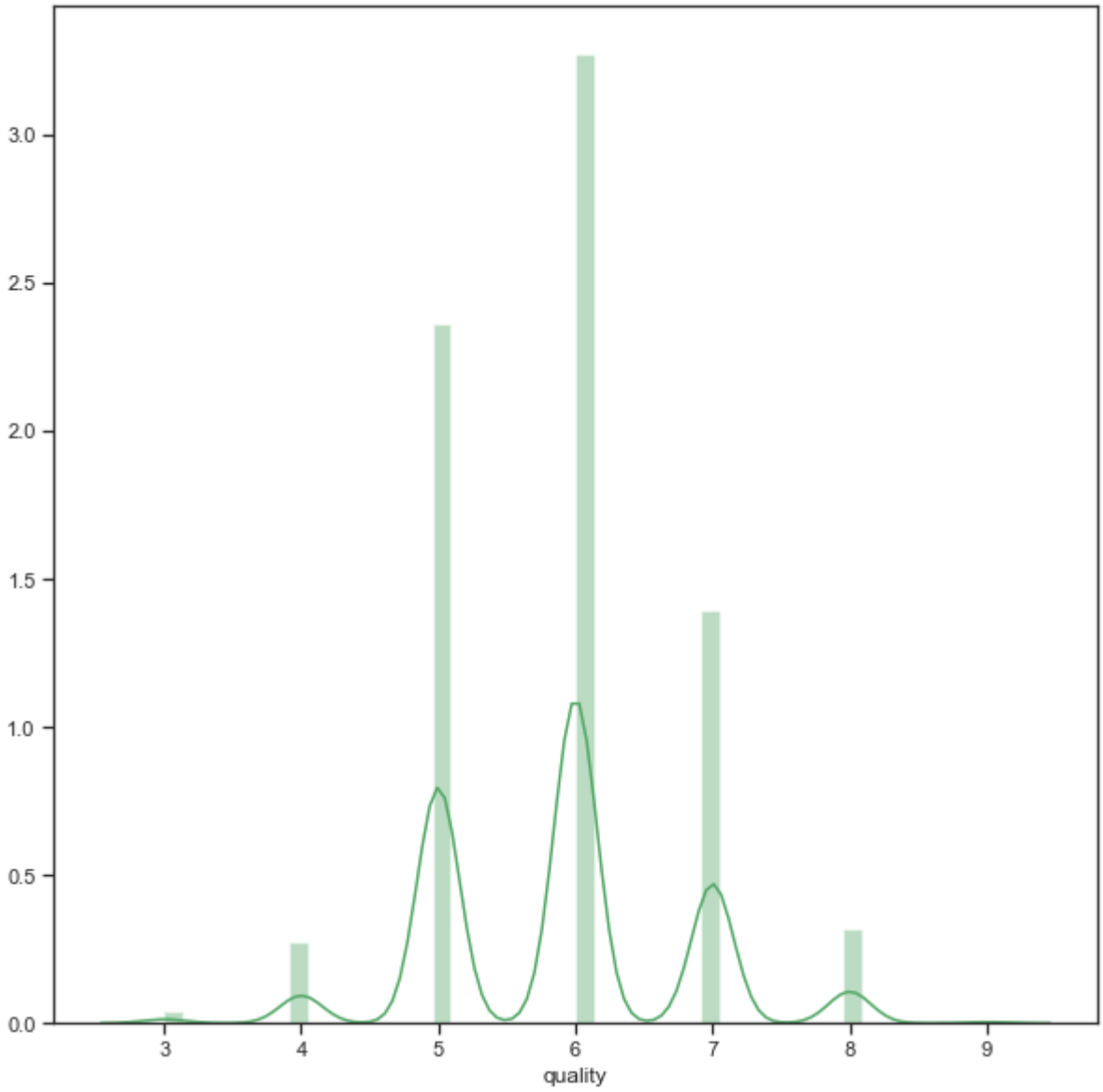
[17]:
```
fig, ax = plt.subplots(figsize=(10,10))
plt.hist(train['quality'], color="g")
plt.show()
```



Оценим здесь же плотность вероятности распределения:

[18]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.distplot(train['quality'], color="g")
```

[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a26037b90>
```



2.3.4. Подсчитаем дисбаланс классов для обучающей выборки

[19]:
```
#
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        30.769999999999996%,
6        42.69%,
7        18.17%,
8        4.109999999999999%,
9        0.13999999999999999%.
```

[20]:
```
train['quality'].value_counts()
```

[20]:
```
6    1494
5    1077
7     636
8     144
4     126
3      18
9       5
Name: quality, dtype: int64
```

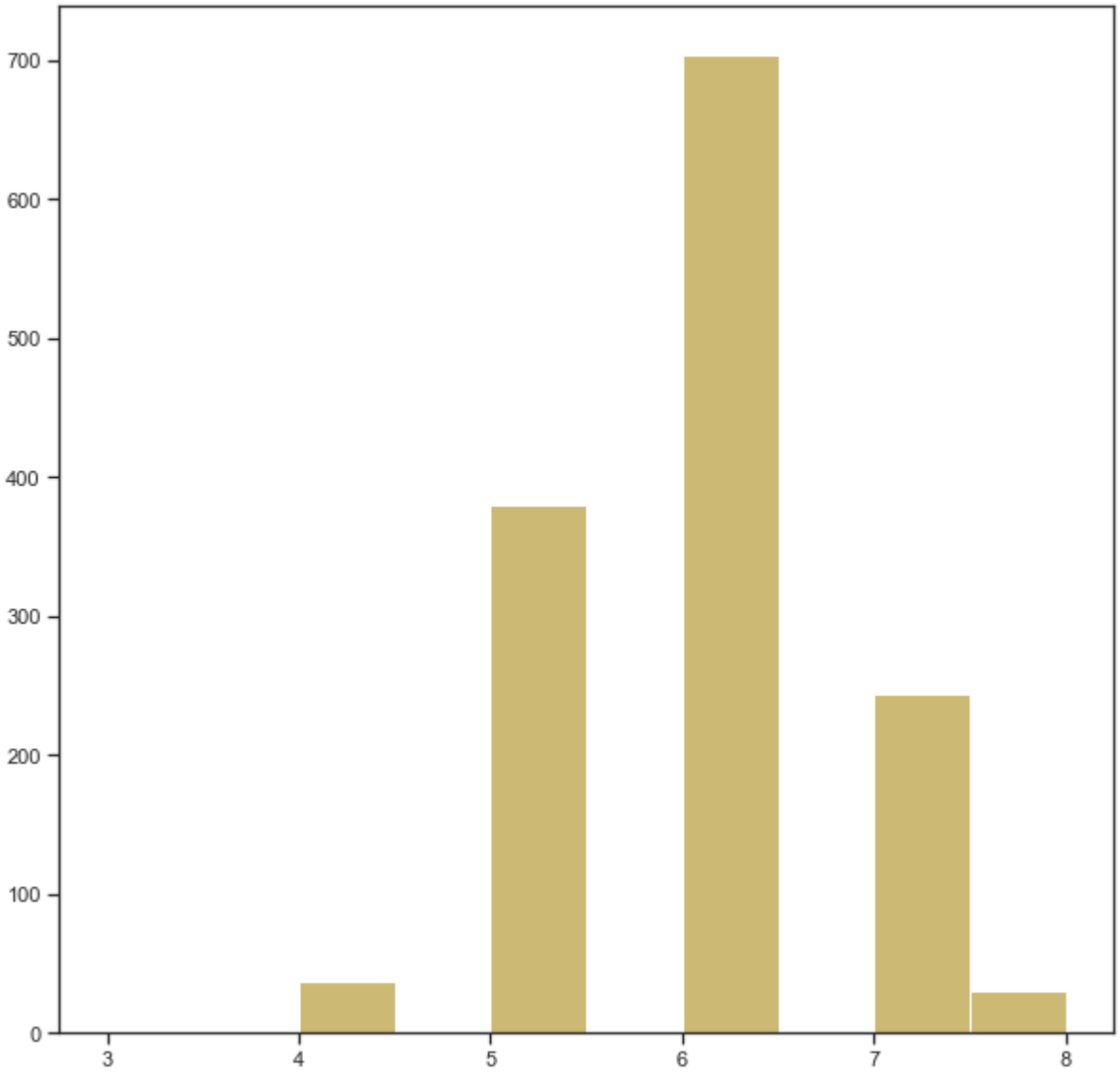
2.3.5. Подсчитаем дисбаланс классов для тестовой выборки

[21]:
```
#
total = test.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        1.29%,
4        9.01%,
5        77.03999999999999%,
6        106.87%,
7        45.49%,
8        10.299999999999999%,
9        0.36%.
```

2.3.6. Распределенеи классов в тестовой выборке

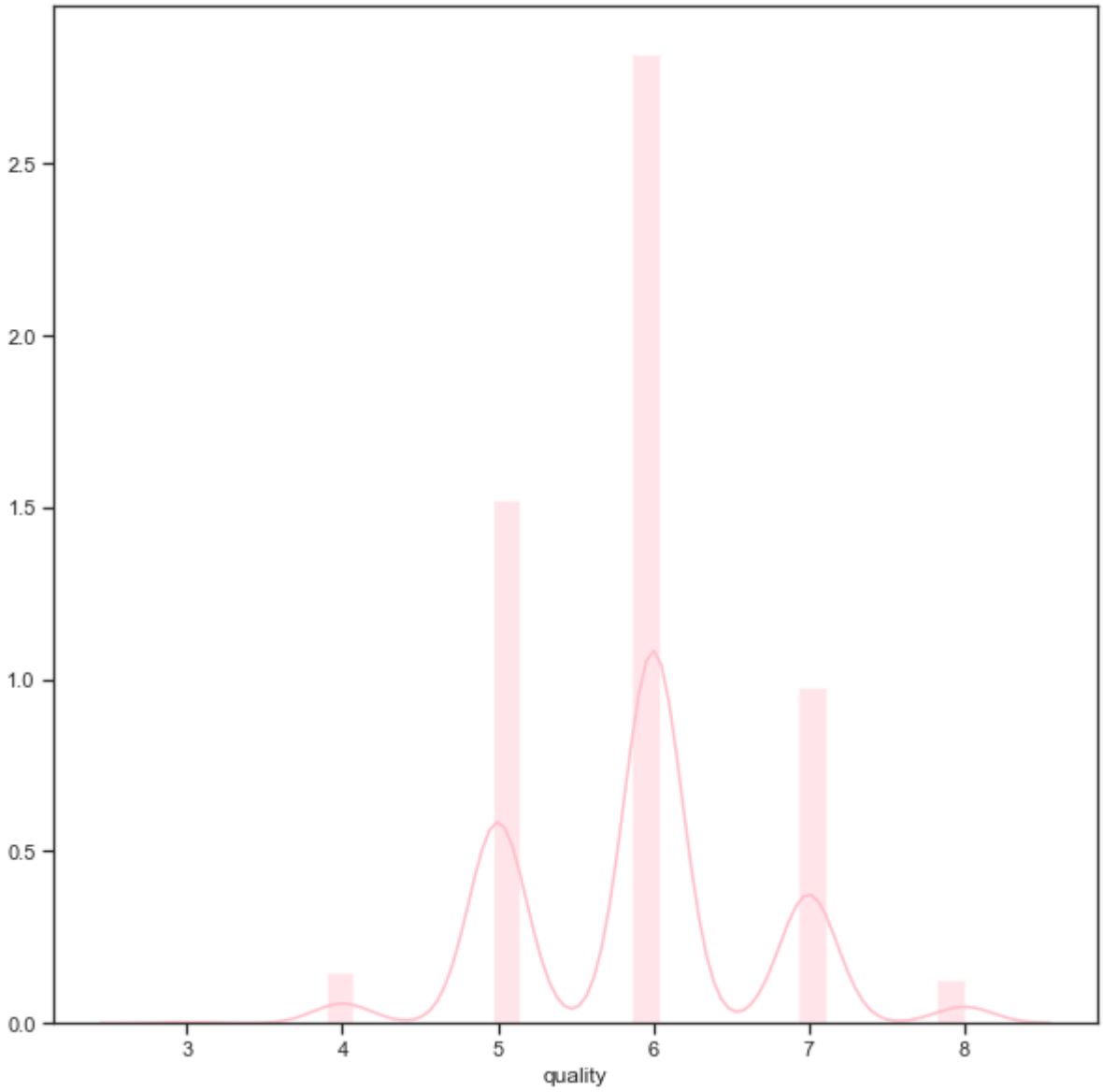
[22]:
```
fig, ax = plt.subplots(figsize=(10,10))
plt.hist(test['quality'], color="y")
plt.show()
```



2.3.7. Оценим плотность вероятности распределения

[23]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.distplot(test['quality'], color="pink")
```

[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a261718d0>
```



2.3.8. 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.

[24]:
```
train.dtypes
```

[24]:
```
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
```

Кодирование признаков не требуется, поскольку все данные представлены в числовом виде. Для построения моделей будем использовать все признаки. Объединим обучающую и тестовую выборки для масштабирования данных. Для начала создадим вспомогательные колонки для возможности дальнейшего разделения целого датасета

[25]:
```
train['dataset'] = 'TRAIN'
test['dataset'] = 'TEST'
```

2.3.9. Выберем столбцы для объединения датасетов

[26]:
```
#
join_cols = ['dataset', 'fixed acidity', 'volatile acidity', 'citric acid',␣
↪'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',␣
↪'density',
       'pH', 'sulphates', 'alcohol', 'quality']
```

[27]:
```
data_all = pd.concat([train[join_cols], test[join_cols]])
```

2.3.10. Проверяем корректность объединения

[28]:
```
assert data_all.shape[0] == train.shape[0]+test.shape[0]
```

[29]:
```
data_all.head()
```

[29]:

	dataset	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
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			

```
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
```

2.3.11. Выберем столбцы для масштабирования

[30]:
```
#
scale_cols = ['fixed acidity', 'volatile acidity', 'citric acid', 'residual␣
↪sugar',
      'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',␣
↪'density',
      'pH', 'sulphates', 'alcohol']
```

[31]:
```
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data_all[scale_cols])
```

2.3.12. Добавляем масштабированные данные в наш датасет

[32]:
```
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data_all[new_col_name] = sc1_data[:,i]
```

2.3.13. Проверяем корректность

[33]:
```
data_all.head()
```

[33]:

	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	

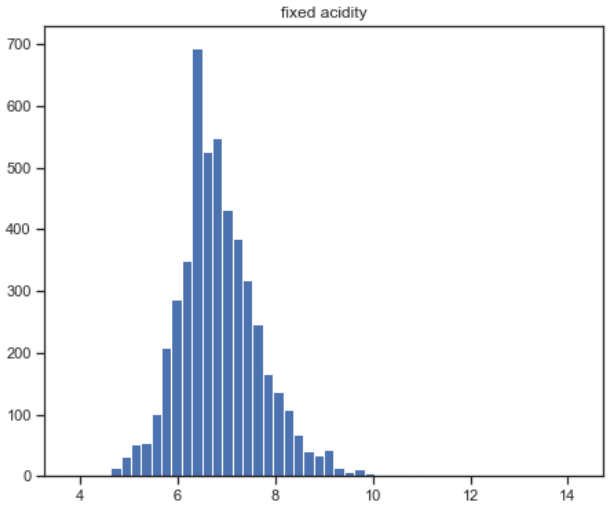
	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	…	\
0	0.045	45.0	170.0	1.0010	3.00	…	
1	0.049	14.0	132.0	0.9940	3.30	…	
2	0.050	30.0	97.0	0.9951	3.26	…	
3	0.058	47.0	186.0	0.9956	3.19	…	
4	0.058	47.0	186.0	0.9956	3.19	…	

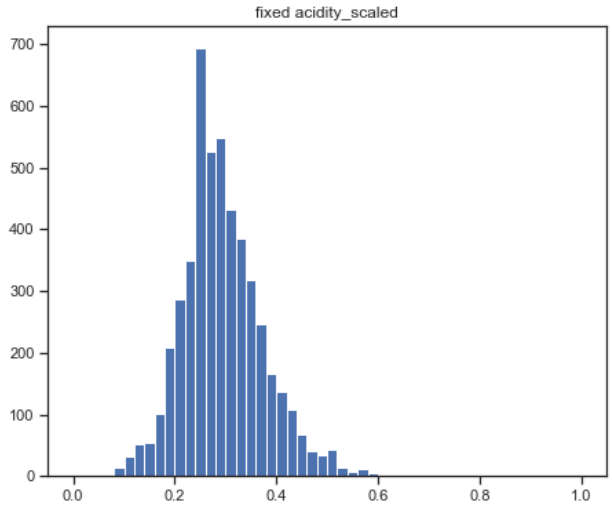
	volatile acidity_scaled	citric acid_scaled	residual sugar_scaled	\
0	0.186275	0.216867	0.308282	
1	0.215686	0.204819	0.015337	
2	0.196078	0.240964	0.096626	
3	0.147059	0.192771	0.121166	

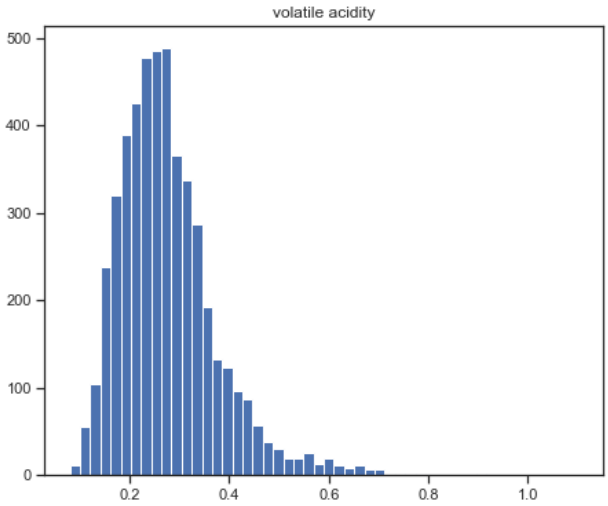
```
4                 0.147059             0.192771                0.121166

   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]
```

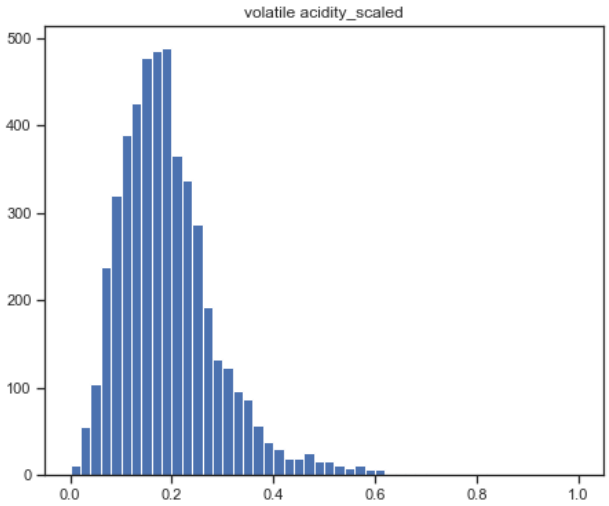
2.3.14. Посмотрим, повлияло ли масштабирование на распределение данных

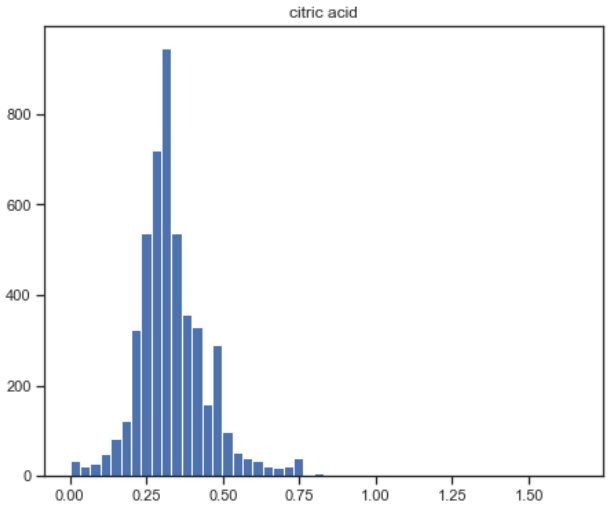
[34]:
```
for col in scale_cols:
    col_scaled = col + '_scaled'
    fig, ax = plt.subplots(1, 2, figsize=(16,6))
    ax[0].hist(data_all[col], 50)
    ax[1].hist(data_all[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```

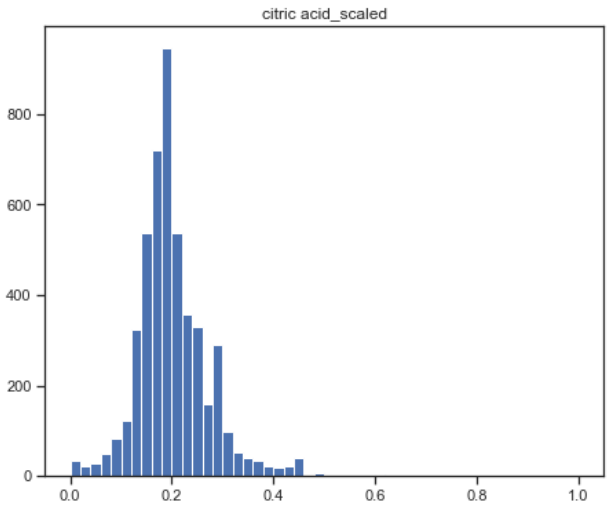


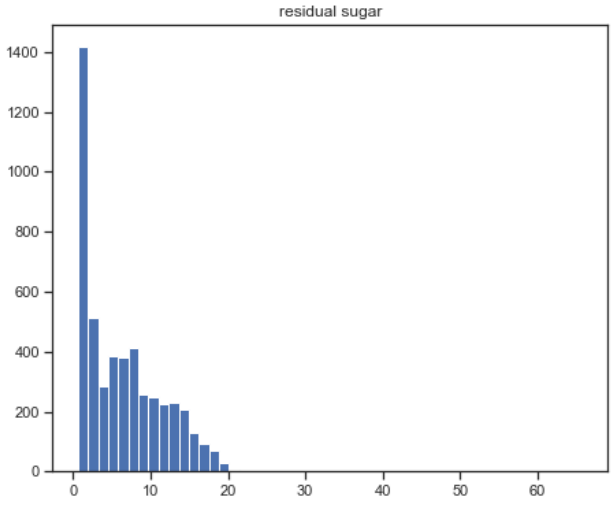


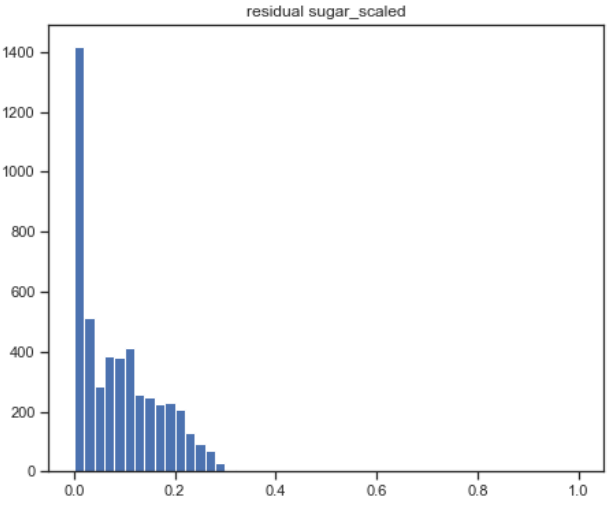

volatile acidity
500
400
300
200
100
0
0.2
0.4
0.6
0.8
1.0

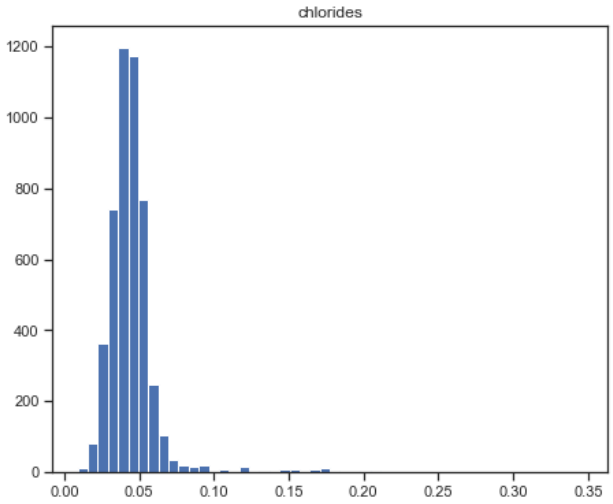

volatile acidity_scaled
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

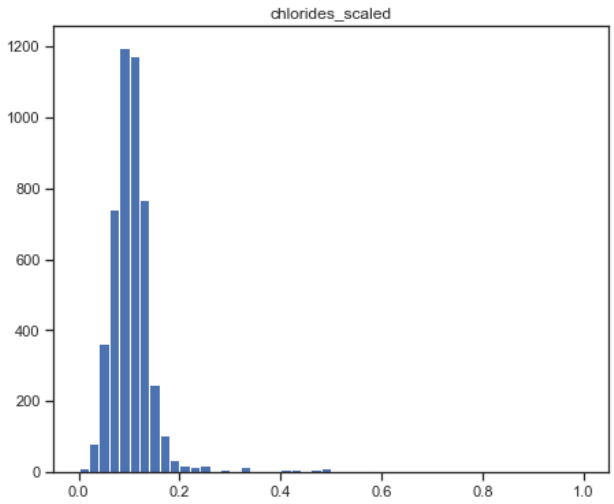

citric acid
800
600
400
200
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50

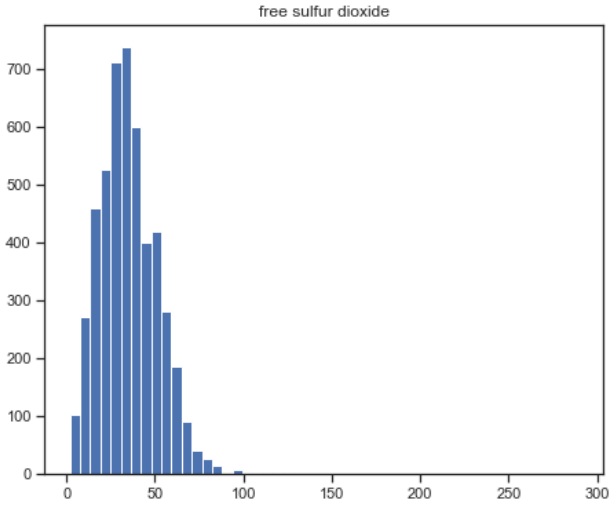

citric acid_scaled
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

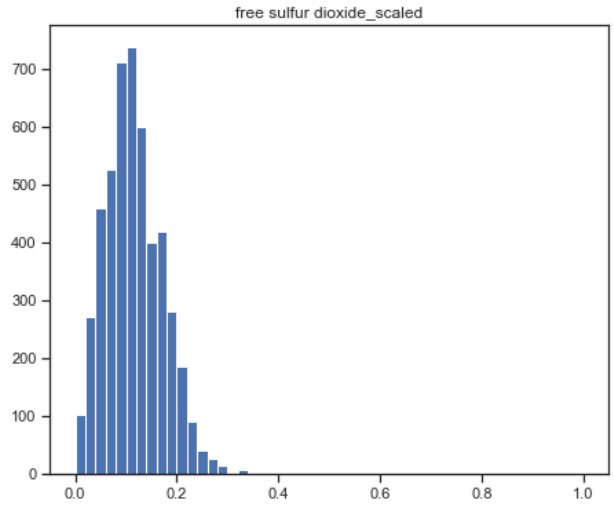

residual sugar
1400
1200
1000
800
600
400
200
0
0
10
20
30
40
50
60

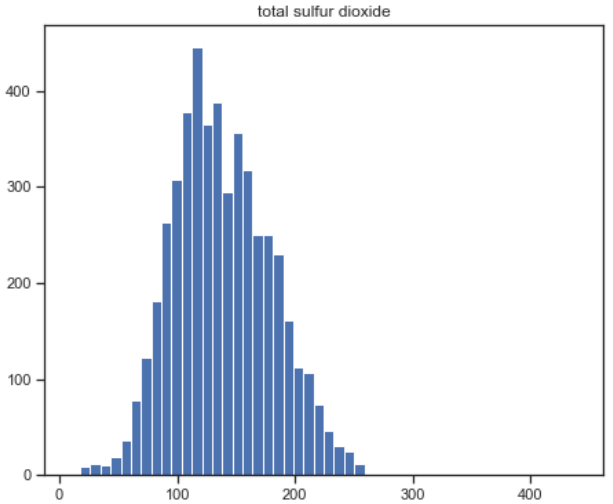

residual sugar_scaled
1400
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

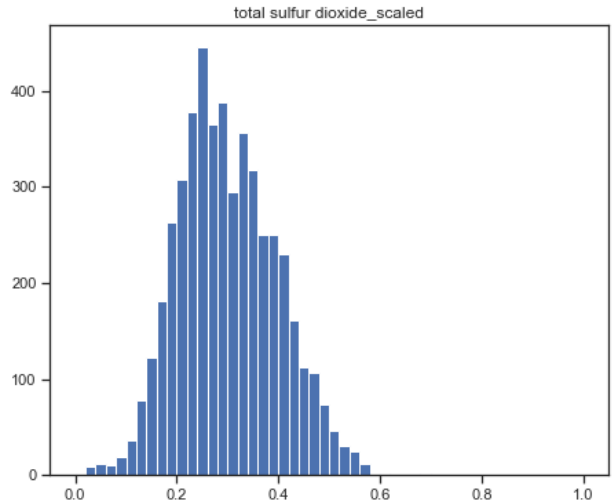

chlorides
1200
1000
800
600
400
200
0
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35

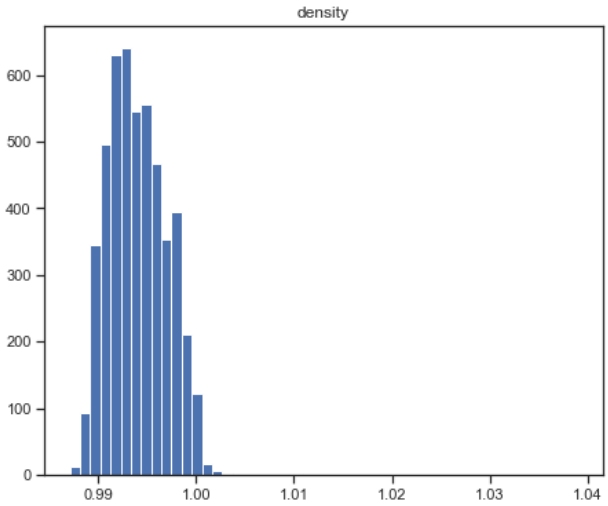

chlorides_scaled
1200
1000
800
600
400
200
0
0.0
0.2
0.4
0.6
0.8
1.0

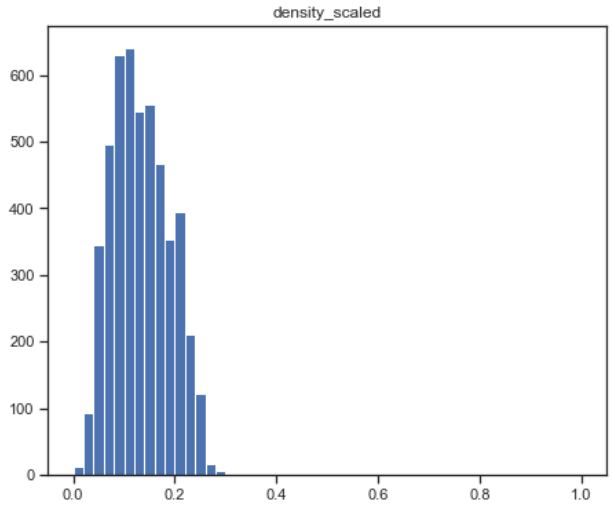

free sulfur dioxide
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300

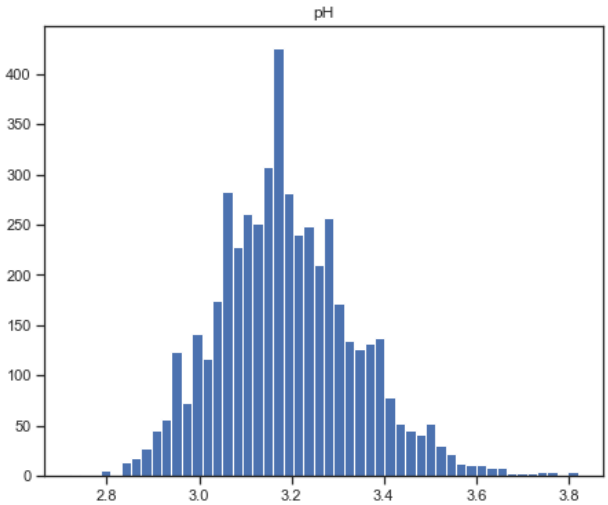

free sulfur dioxide_scaled
700
600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

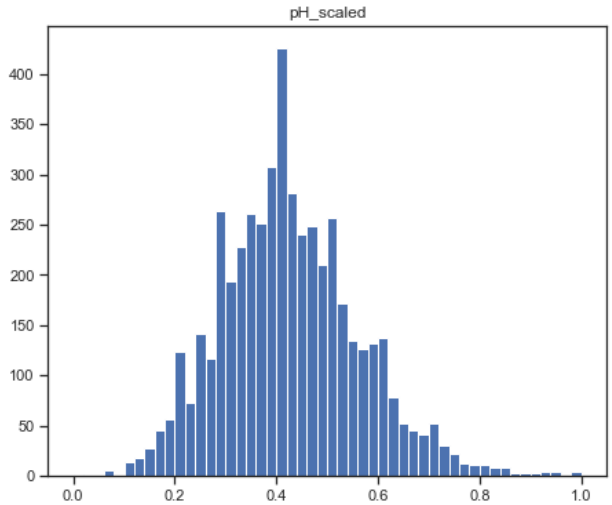

total sulfur dioxide
400
300
200
100
0
0
100
200
300
400

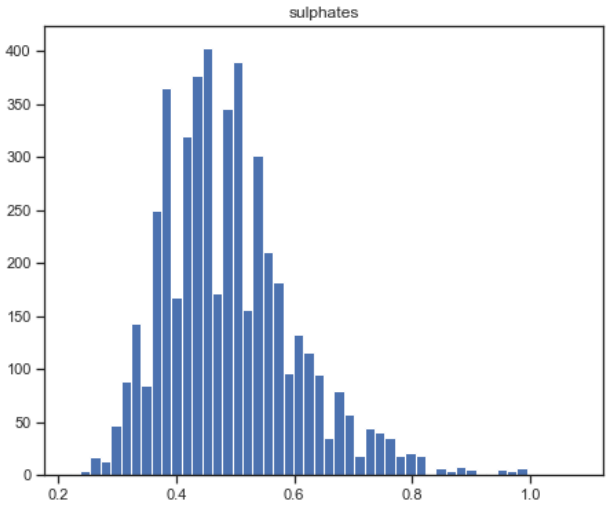

total sulfur dioxide_scaled
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

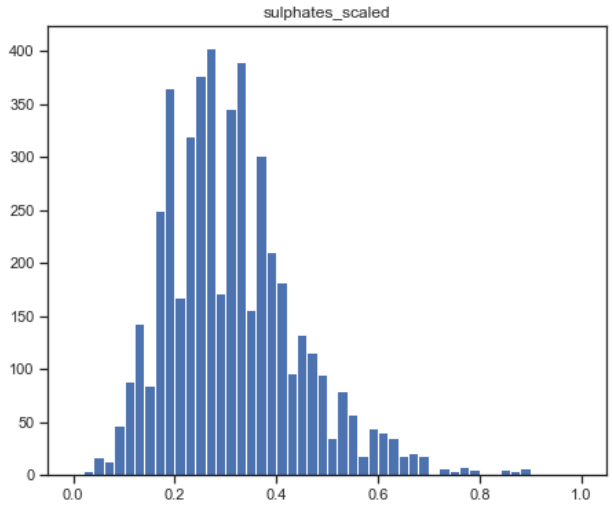

density
600
500
400
300
200
100
0
0.99
1.00
1.01
1.02
1.03
1.04

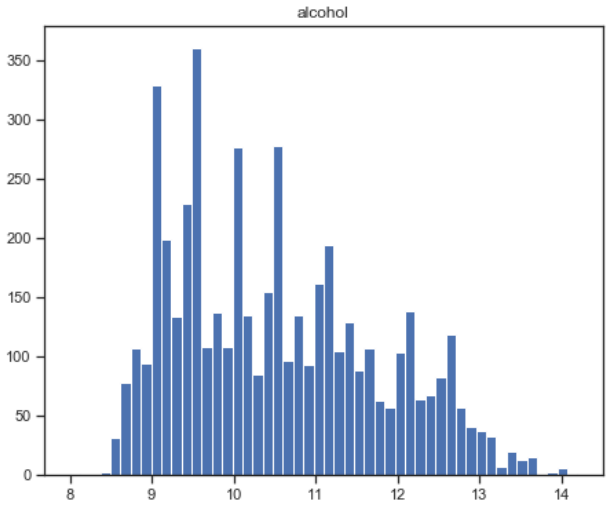

density_scaled
600
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0

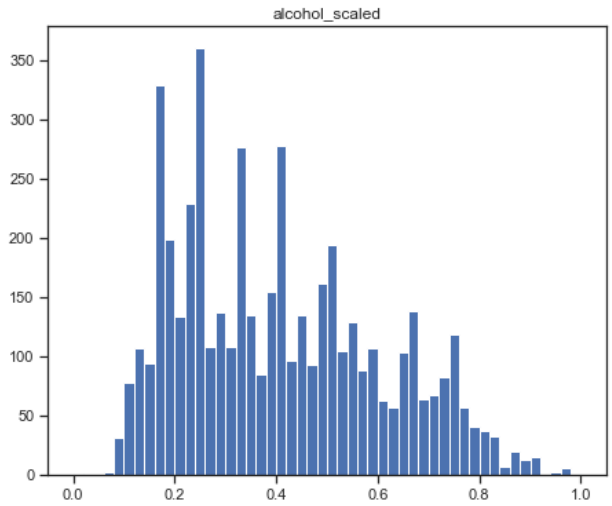

pH
400
350
300
250
200
150
100
50
0
2.8
3.0
3.2
3.4
3.6
3.8


pH_scaled
400
350
300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0


sulphates
400
350
300
250
200
150
100
50
0
0.2
0.4
0.6
0.8
1.0


sulphates_scaled
400
350
300
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0





Масштабирование данных не повлияло на их распределение

2.3.15. Включим тестовую выборку в корреляционную матрицу

[35]:
```
corr_cols_1 = scale_cols + ['quality']
corr_cols_1
```

[35]:
```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

[36]:
```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['quality']
corr_cols_2
```

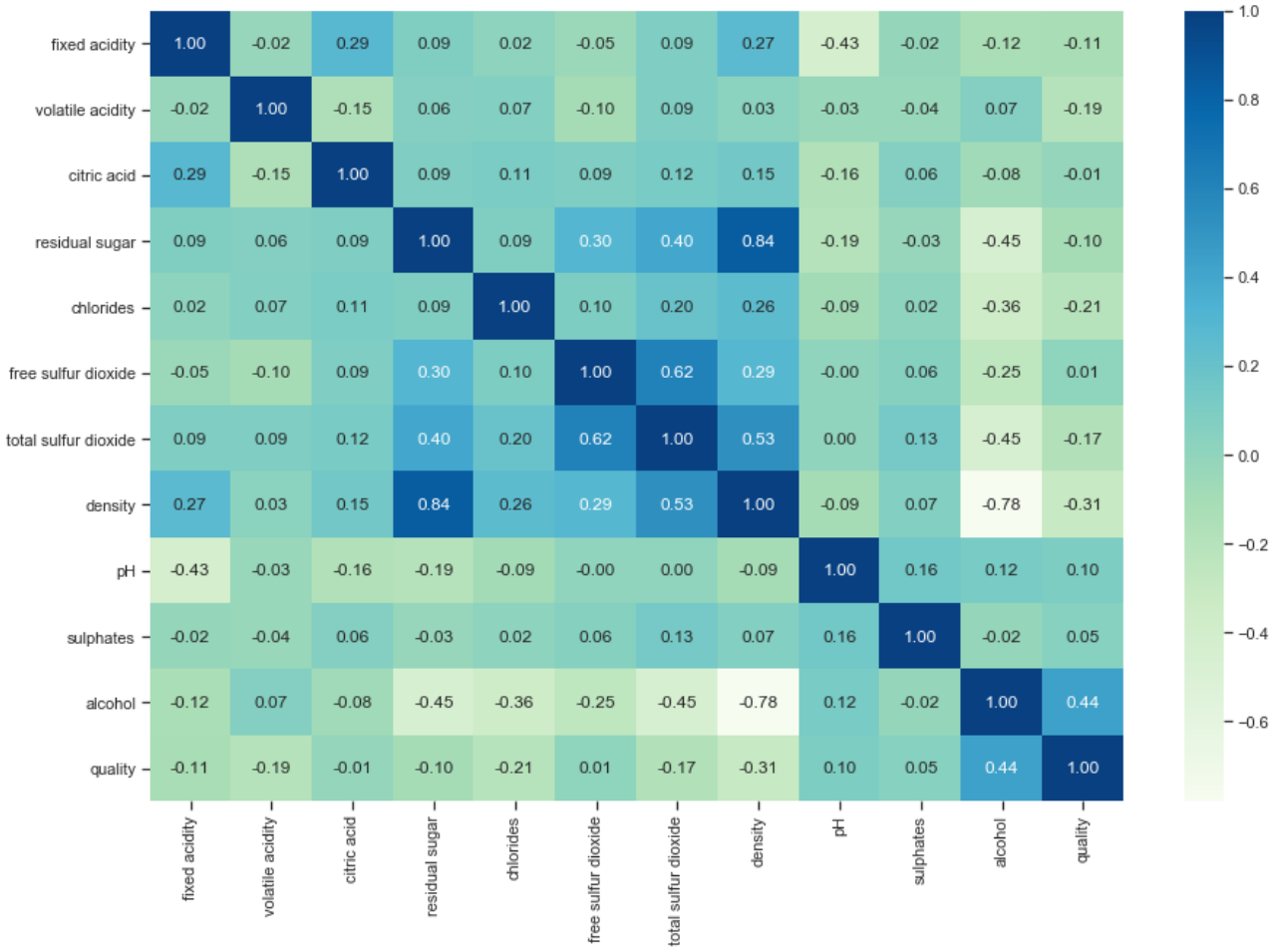
[36]:
```
['fixed acidity_scaled',
 'volatile acidity_scaled',
 'citric acid_scaled',
 'residual sugar_scaled',
 'chlorides_scaled',
 'free sulfur dioxide_scaled',
 'total sulfur dioxide_scaled',
 'density_scaled',
 'pH_scaled',
 'sulphates_scaled',
 'alcohol_scaled',
```

```
'quality']
```

2.3.16. Построим корреляционную матрицу

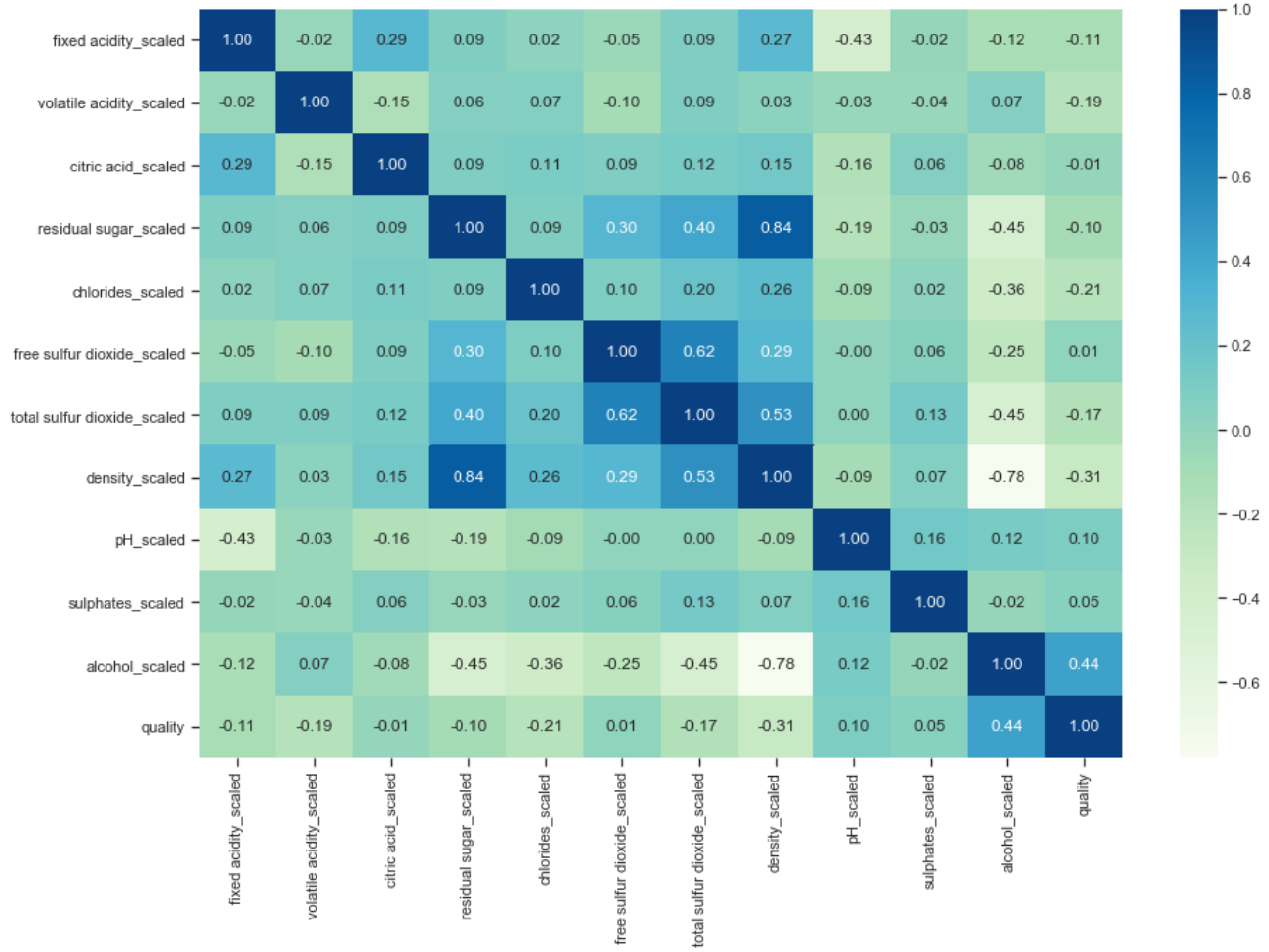
[37]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data_all[corr_cols_1].corr(), annot=True, fmt='.2f',␣
↪cmap='GnBu')
```

[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a27563190>
```



[38]:
```
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(data_all[corr_cols_2].corr(), annot=True, fmt='.2f',␣
↪cmap="GnBu")
```

[38]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a267570d0>
```



Корреляционные матрицы для исходных и масштабированных данных полностью совпадают

2.4. 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.

2.4.1. 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”.

[39]:
```
qual = pd.concat([train['quality'], test['quality']])
```

[40]:
```
def code_myohe(data, column):
    for i in data[column].unique():
        data[column + '=' + str(i)] = (data[column] == i).astype(int)
```

[41]:
```
code_myohe(data_all, 'quality')
data_all.head()
```

[41]:

	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]

[42]:
```
data_all['quality'] = qual
```

2.5. 4) Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать: - Метрика precision: $precision = \frac{TP}{TP+FP}$ - Метрика recall (полнота): $recall = \frac{TP}{TP+FN}$ - Метрика F_1-мера: $F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{precision+recall}$, где β определяет вес точности в метрике. - Метрика ROC AUC: $TPR = \frac{TP}{TP+FN}$ - True Positive Rate, откладывается по оси ординат. Совпадает с recall. $FPR = \frac{FP}{FP+TN}$ - False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

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

[43]:
```
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()
```

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

2.6.1. Выделим обучающую и тестовую выборки на основе масштабированных данных с помощью фильтра

[44]:
```
train_data_all = data_all[data_all['dataset']=='TRAIN']
test_data_all = data_all[data_all['dataset']=='TEST']
train_data_all.shape, test_data_all.shape
```

[44]:
```
((3500, 31), (1398, 31))
```

2.6.2. Определим признаки для задачи классификации

[45]:
```
#
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']
```

2.6.3. Определим выборки для задачи классификации

[46]:
```
#
clas_X_train = train_data_all[task_clas_cols]
clas_X_test = test_data_all[task_clas_cols]

clas_Y6_train = train_data_all['quality=6']
clas_Y6_test = test_data_all['quality=6']

clas_Y7_train = train_data_all['quality=7']
clas_Y7_test = test_data_all['quality=7']

clas_X_train.shape, clas_X_test.shape, clas_Y6_train.shape, clas_Y6_test.
 ↪shape
```

[46]:
```
((3500, 9), (1398, 9), (3500,), (1398,))
```

2.7. Построение базового решения

2.7.1. Определим модель

[49]:
```
clas_models = {'LogR': LogisticRegression(),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier()}
```

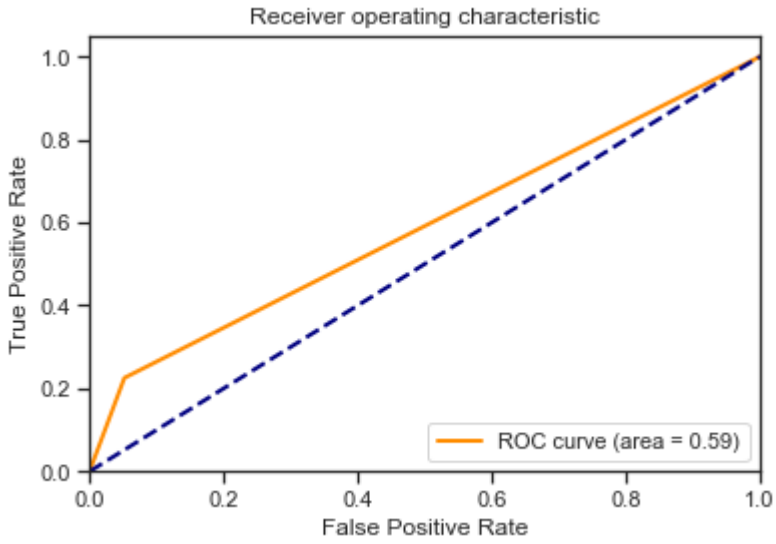
2.7.2. Сохранение метрик

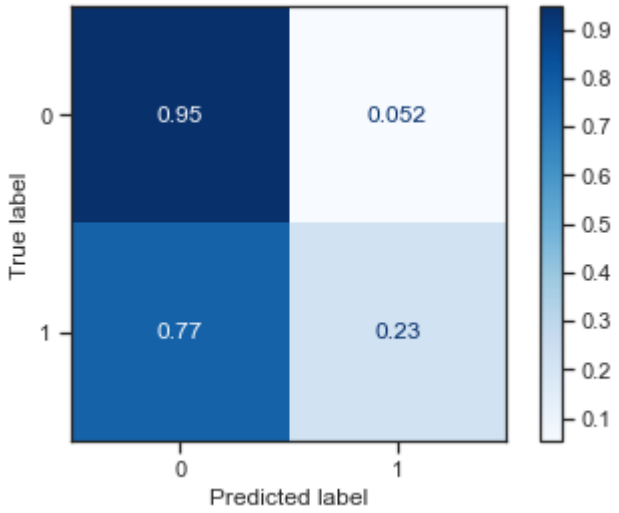
[50]:
```
clasMetricLogger = MetricLogger()
```

[51]:
```
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()
```

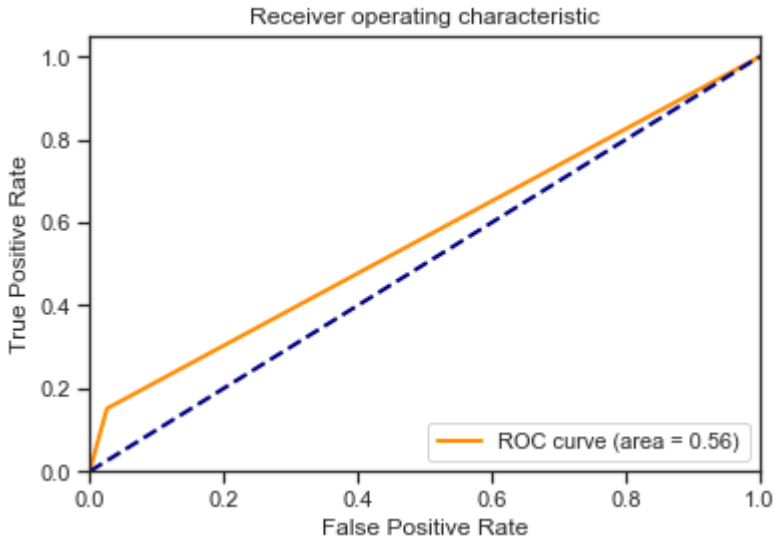
[52]:
```
for model_name, model in clas_models.items():
    clas_train_model7(model_name, model, clasMetricLogger)
```

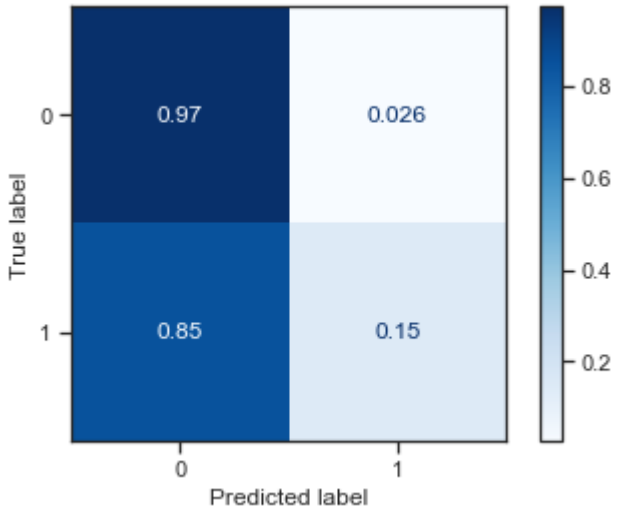
```
****************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None,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



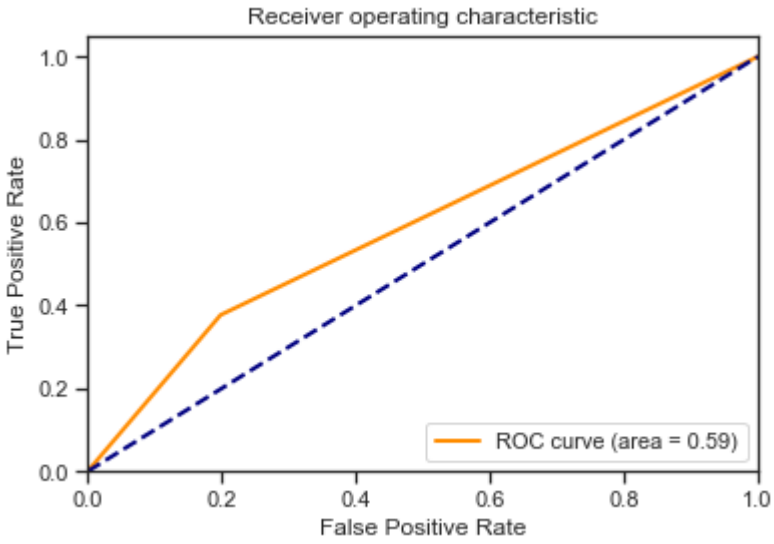


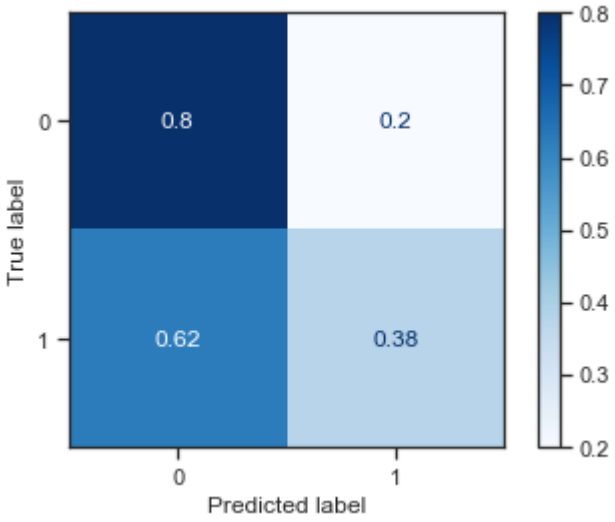

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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






```
******************************************************
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')
******************************************************
```





2.8. 7) Подбор гиперпараметров

[76]:
```
clas_X_train.shape
```

[76]: (3500, 9)

[53]:
```
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)
```

[53]:
```
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)
```

2.8.1. Лучшая модель

[54]:
```
clf_gs.best_estimator_
```

[54]:
```
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')
```

2.8.2. Лучшее значение параметров

[55]:
```
clf_gs.best_params_
```

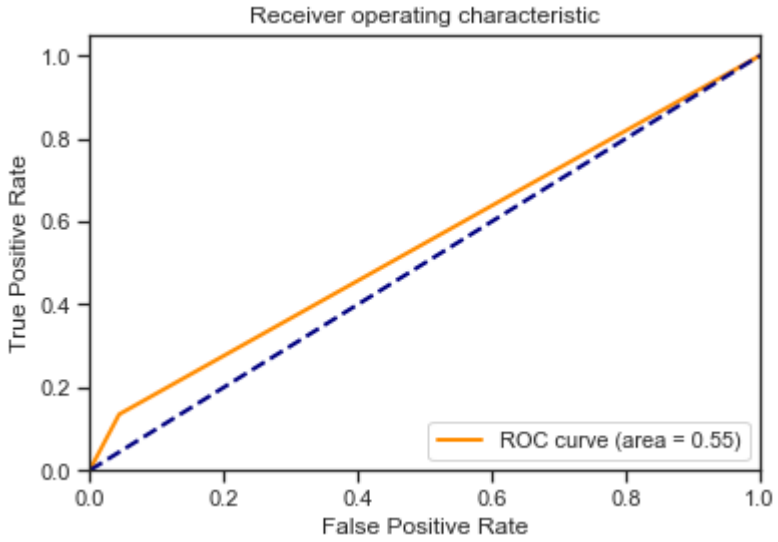
[55]:
```
{'criterion': 'gini', 'max_depth': 5, 'splitter': 'random'}
```

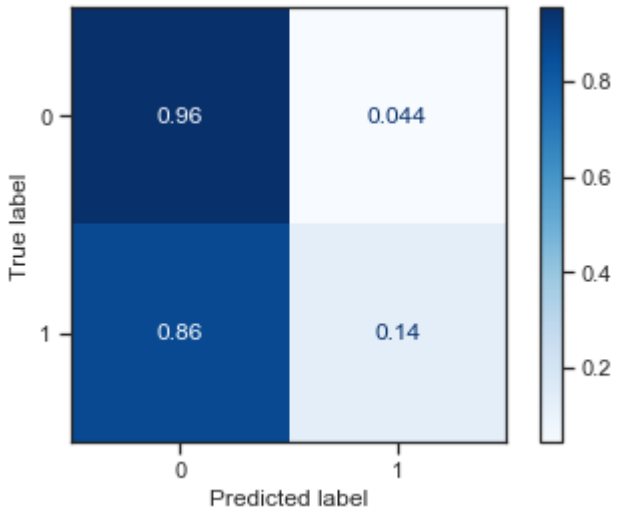
2.9. 8) Сравнение качества полученных моделей с качеством baseline-моделей.

[57]:
```
clas_models_grid = {'best_tree':clf_gs.best_estimator_}
```

[58]:
```
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')
******************************************************
```





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

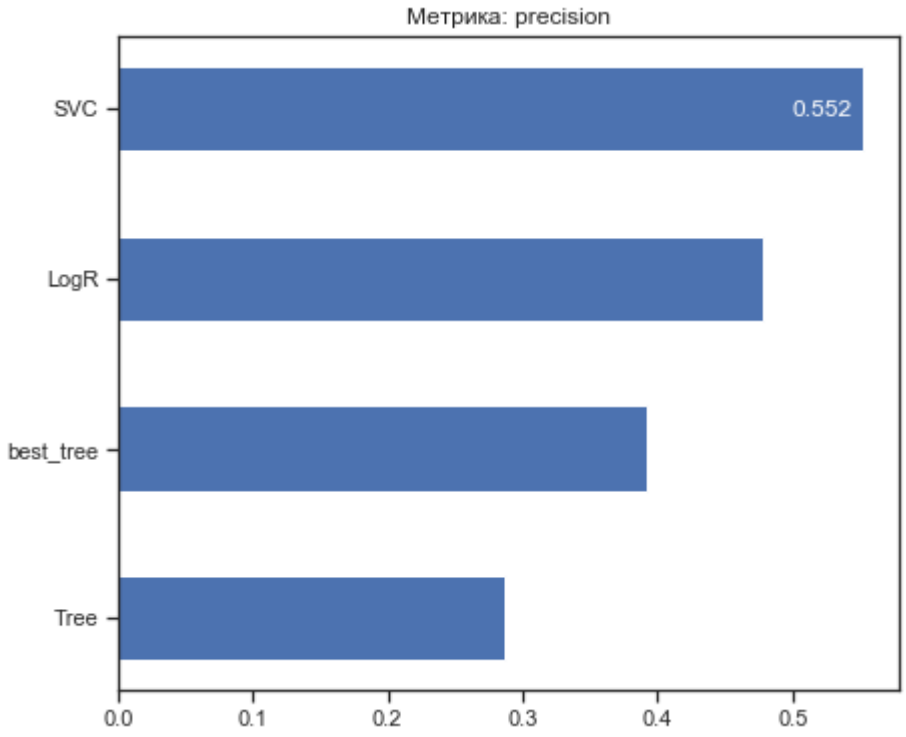
2.10.1. Метрики качества модели

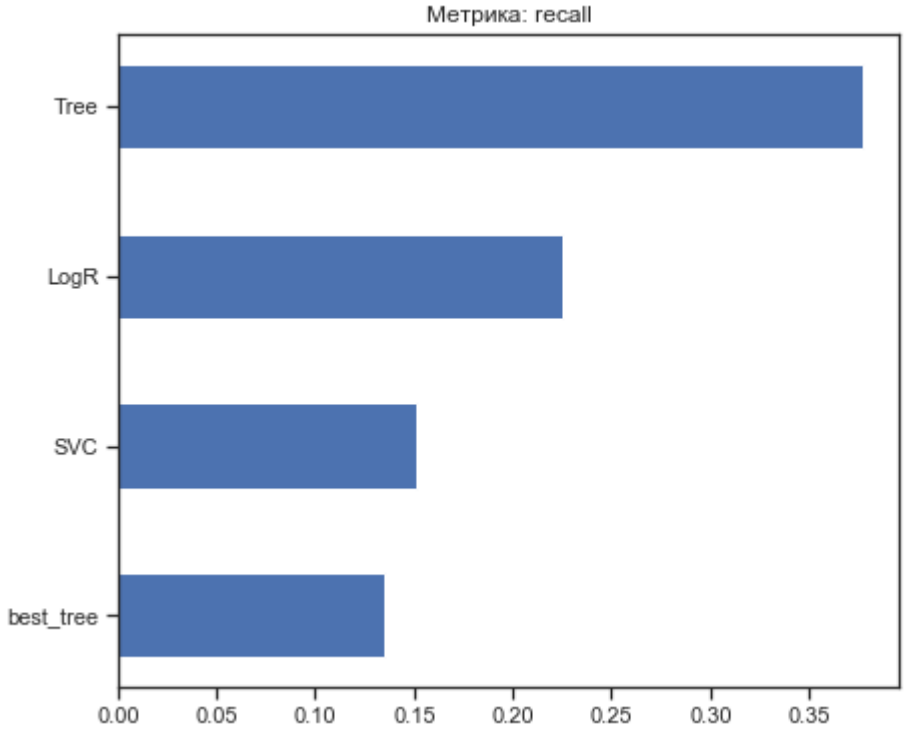
[59]:
```
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

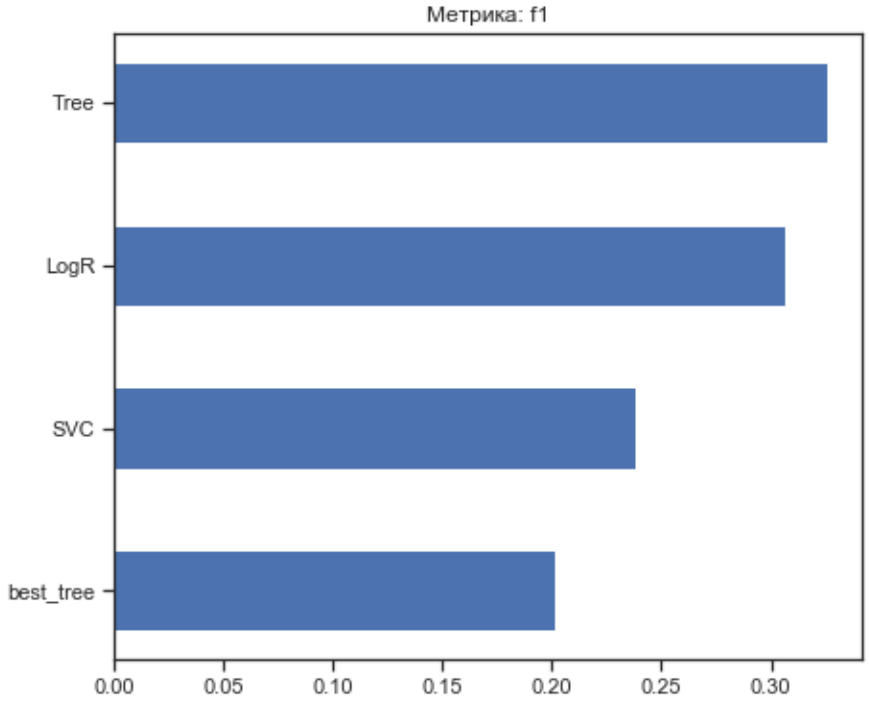
[59]:
```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

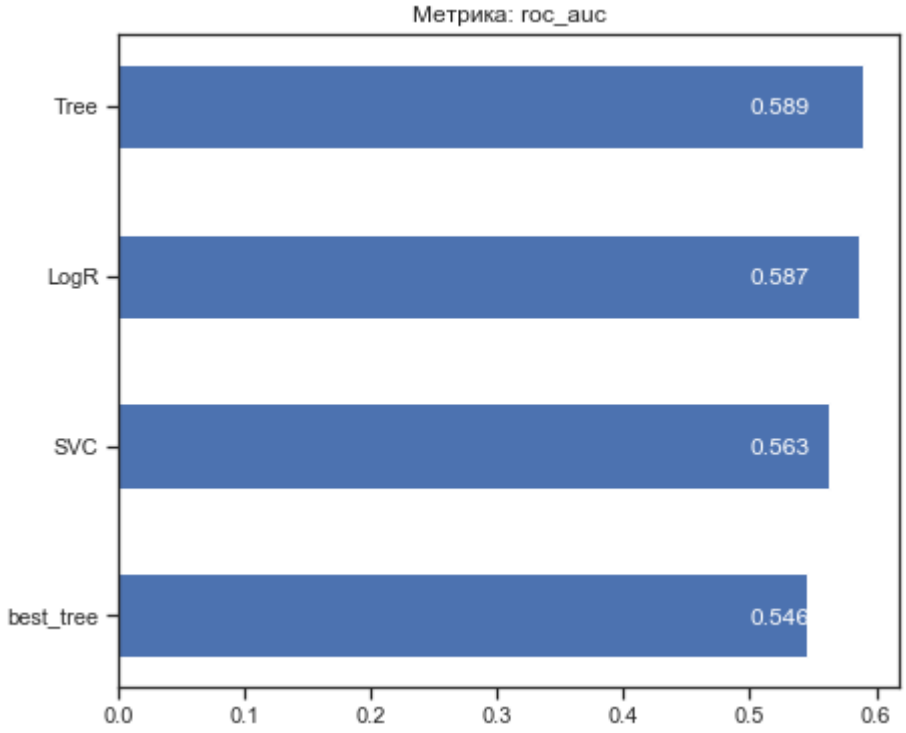
2.10.2. Графики метрик качества модели

[60]:
```
for metric in clas_metrics:
    clasMetricLogger.plot('    : ' + metric, metric, figsize=(7, 6))
```




Метрика: recall
Tree
LogR
SVC
best_tree
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35


Метрика: f1
Tree
LogR
SVC
best_tree
0.00
0.05
0.10
0.15
0.20
0.25
0.30



2.11. Вывод

Без гиперпараметров точность расчетов оказалась точнее

[ ]: