# НИРС по дисциплине "Технологии машинного обучения"

## Поиск и выбор набора данных для построения моделей машинного обучения

В качестве набора данных используется набор данных химического анализа вин - <https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

Данные являются результатом химического анализа вин, выращенных в одном и том же регионе Италии тремя разными культиваторами. Существует тринадцать различных измерений различных компонентов, содержащихся в трех типах вина.

Набор данных содержит следующие параметры: Alcohol - Алкоголь; Acid - Яблочная кислота; Ash - Пепел; Alcalinity of Ash - Щелочность пепла; Magnesium - Магний; Total Phenols - Всего фенолов; Flavanoids - Флавоноиды; Nonflavanoid Phenols - Нефлаваноидные фенолы; Proanthocyanins - Проантоцианы; Colour Intensity - Интенсивность цвета; Hue - Оттенок; OD280/OD315 of diluted wines - OD280/OD315 разбавленных вин; Proline - Пролин.

### Импорт библиотек и загрузка датасета

import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.datasets import \*  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.model\_selection import GridSearchCV  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.linear\_model import LogisticRegression  
# from sklearn import svm, tree  
from sklearn.tree import DecisionTreeClassifier  
# from sklearn.ensemble import RandomForestClassifier  
from sklearn.ensemble import BaggingClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, balanced\_accuracy\_score  
from sklearn.metrics import roc\_curve, roc\_auc\_score  
from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report  
# from sklearn.metrics import mean\_absolute\_error  
from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay  
from operator import itemgetter  
  
def make\_dataframe(ds\_function):  
 ds = ds\_function()  
 df = pd.DataFrame(data= np.c\_[ds['data'], ds['target']],  
 columns= list(ds['feature\_names']) + ['target'])  
 return df  
  
wine = load\_wine()  
  
df = make\_dataframe(load\_wine)

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

### Основные характеристики датасета

# Первые 5 строк датасета  
df.head()

alcohol malic\_acid ash alcalinity\_of\_ash magnesium total\_phenols \  
0 14.23 1.71 2.43 15.6 127.0 2.80   
1 13.20 1.78 2.14 11.2 100.0 2.65   
2 13.16 2.36 2.67 18.6 101.0 2.80   
3 14.37 1.95 2.50 16.8 113.0 3.85   
4 13.24 2.59 2.87 21.0 118.0 2.80   
  
 flavanoids nonflavanoid\_phenols proanthocyanins color\_intensity hue \  
0 3.06 0.28 2.29 5.64 1.04   
1 2.76 0.26 1.28 4.38 1.05   
2 3.24 0.30 2.81 5.68 1.03   
3 3.49 0.24 2.18 7.80 0.86   
4 2.69 0.39 1.82 4.32 1.04   
  
 od280/od315\_of\_diluted\_wines proline target   
0 3.92 1065.0 0.0   
1 3.40 1050.0 0.0   
2 3.17 1185.0 0.0   
3 3.45 1480.0 0.0   
4 2.93 735.0 0.0

# Размер датасета - 178 строк, 14 колонок  
df.shape

(178, 14)

# Список колонок  
df.columns

Index(['alcohol', 'malic\_acid', 'ash', 'alcalinity\_of\_ash', 'magnesium',  
 'total\_phenols', 'flavanoids', 'nonflavanoid\_phenols',  
 'proanthocyanins', 'color\_intensity', 'hue',  
 'od280/od315\_of\_diluted\_wines', 'proline', 'target'],  
 dtype='object')

# Список колонок с типами данных  
df.dtypes

alcohol float64  
malic\_acid float64  
ash float64  
alcalinity\_of\_ash float64  
magnesium float64  
total\_phenols float64  
flavanoids float64  
nonflavanoid\_phenols float64  
proanthocyanins float64  
color\_intensity float64  
hue float64  
od280/od315\_of\_diluted\_wines float64  
proline float64  
target float64  
dtype: object

# Проверим наличие пустых значений  
# Цикл по колонкам датасета  
for col in df.columns:  
 # Количество пустых значений - все значения заполнены  
 temp\_null\_count = df[df[col].isnull()].shape[0]  
 print('{} - {}'.format(col, temp\_null\_count))

alcohol - 0  
malic\_acid - 0  
ash - 0  
alcalinity\_of\_ash - 0  
magnesium - 0  
total\_phenols - 0  
flavanoids - 0  
nonflavanoid\_phenols - 0  
proanthocyanins - 0  
color\_intensity - 0  
hue - 0  
od280/od315\_of\_diluted\_wines - 0  
proline - 0  
target - 0

# Основные статистические характеристки набора данных  
df.describe()

alcohol malic\_acid ash alcalinity\_of\_ash magnesium \  
count 178.000000 178.000000 178.000000 178.000000 178.000000   
mean 13.000618 2.336348 2.366517 19.494944 99.741573   
std 0.811827 1.117146 0.274344 3.339564 14.282484   
min 11.030000 0.740000 1.360000 10.600000 70.000000   
25% 12.362500 1.602500 2.210000 17.200000 88.000000   
50% 13.050000 1.865000 2.360000 19.500000 98.000000   
75% 13.677500 3.082500 2.557500 21.500000 107.000000   
max 14.830000 5.800000 3.230000 30.000000 162.000000   
  
 total\_phenols flavanoids nonflavanoid\_phenols proanthocyanins \  
count 178.000000 178.000000 178.000000 178.000000   
mean 2.295112 2.029270 0.361854 1.590899   
std 0.625851 0.998859 0.124453 0.572359   
min 0.980000 0.340000 0.130000 0.410000   
25% 1.742500 1.205000 0.270000 1.250000   
50% 2.355000 2.135000 0.340000 1.555000   
75% 2.800000 2.875000 0.437500 1.950000   
max 3.880000 5.080000 0.660000 3.580000   
  
 color\_intensity hue od280/od315\_of\_diluted\_wines proline \  
count 178.000000 178.000000 178.000000 178.000000   
mean 5.058090 0.957449 2.611685 746.893258   
std 2.318286 0.228572 0.709990 314.907474   
min 1.280000 0.480000 1.270000 278.000000   
25% 3.220000 0.782500 1.937500 500.500000   
50% 4.690000 0.965000 2.780000 673.500000   
75% 6.200000 1.120000 3.170000 985.000000   
max 13.000000 1.710000 4.000000 1680.000000   
  
 target   
count 178.000000   
mean 0.938202   
std 0.775035   
min 0.000000   
25% 0.000000   
50% 1.000000   
75% 2.000000   
max 2.000000

Выводы:

1. Все значения в датасете являются числовыми.
2. Представленный набор данных не содержит пропусков.

### Построение графиков для понимания структуры данных

sns.pairplot(df)

<seaborn.axisgrid.PairGrid at 0x7f9100859fd0>

# Группировка по целевому признаку  
sns.pairplot(df, hue="target")

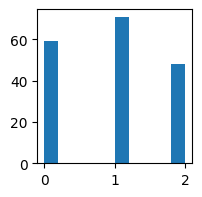
<seaborn.axisgrid.PairGrid at 0x7f9123550dc0>

# Убедимся, что целевой признак подходит для задачи классификации  
df['target'].unique()

array([0., 1., 2.])

# Оценим дисбаланс классов для target  
fig, ax = plt.subplots(figsize=(2,2))   
plt.hist(df['target'])  
plt.show()





df['target'].value\_counts()

1.0 71  
0.0 59  
2.0 48  
Name: target, dtype: int64

# посчитаем дисбаланс классов  
total = df.shape[0]  
class\_0, class\_1, class\_2 = df['target'].value\_counts()  
print('Класс 0 составляет {}%, класс 1 составляет {}%, а класс 2 составляет {}%'  
 .format(round(class\_0 / total, 4)\*100, round(class\_1 / total, 4)\*100, round(class\_2 / total, 4)\*100))

Класс 0 составляет 39.89%, класс 1 составляет 33.15%, а класс 2 составляет 26.97%

Вывод. Дисбаланс классов присутствует, но является приемлемым.

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

df.dtypes

alcohol float64  
malic\_acid float64  
ash float64  
alcalinity\_of\_ash float64  
magnesium float64  
total\_phenols float64  
flavanoids float64  
nonflavanoid\_phenols float64  
proanthocyanins float64  
color\_intensity float64  
hue float64  
od280/od315\_of\_diluted\_wines float64  
proline float64  
target float64  
dtype: object

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей в данном примере мы строить не будем.

Выполним масштабирование данных.

df.columns

Index(['alcohol', 'malic\_acid', 'ash', 'alcalinity\_of\_ash', 'magnesium',  
 'total\_phenols', 'flavanoids', 'nonflavanoid\_phenols',  
 'proanthocyanins', 'color\_intensity', 'hue',  
 'od280/od315\_of\_diluted\_wines', 'proline', 'target'],  
 dtype='object')

data\_all=df

# колонки для масштабирования  
scale\_cols = ['alcohol', 'malic\_acid', 'ash', 'alcalinity\_of\_ash', 'magnesium',  
 'total\_phenols', 'flavanoids', 'nonflavanoid\_phenols',  
 'proanthocyanins', 'color\_intensity', 'hue',  
 'od280/od315\_of\_diluted\_wines', 'proline']

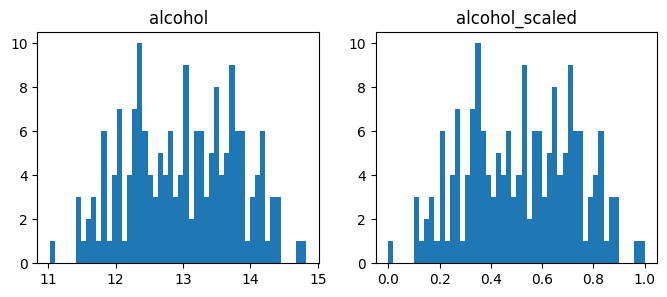
sc1 = MinMaxScaler()  
sc1\_data = sc1.fit\_transform(data\_all[scale\_cols])

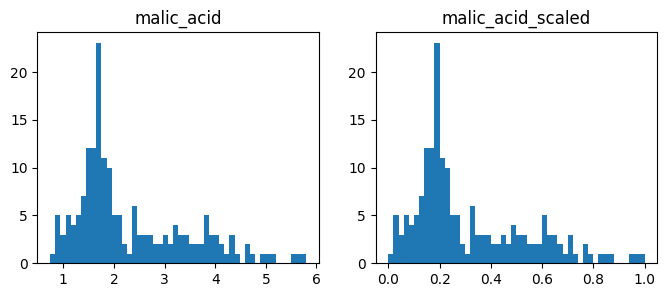
# Добавим масштабированные данные в набор данных  
for i in range(len(scale\_cols)):  
 col = scale\_cols[i]  
 new\_col\_name = col + '\_scaled'  
 data\_all[new\_col\_name] = sc1\_data[:,i]

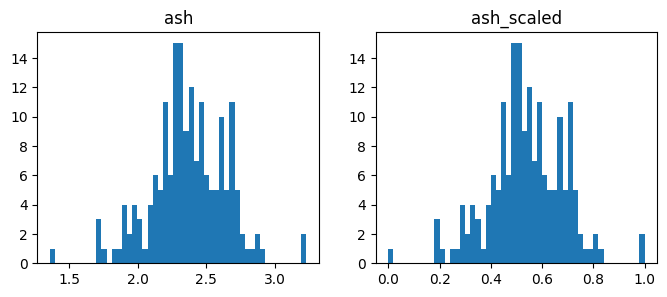
data\_all.head()

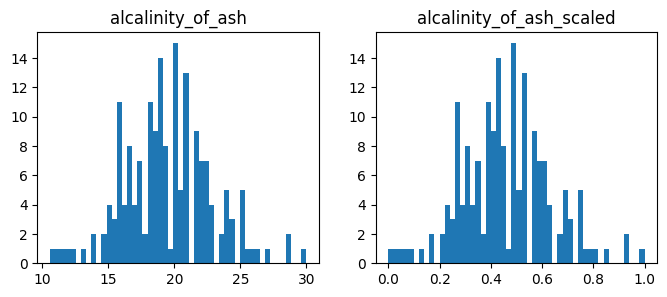
alcohol malic\_acid ash alcalinity\_of\_ash magnesium total\_phenols \  
0 14.23 1.71 2.43 15.6 127.0 2.80   
1 13.20 1.78 2.14 11.2 100.0 2.65   
2 13.16 2.36 2.67 18.6 101.0 2.80   
3 14.37 1.95 2.50 16.8 113.0 3.85   
4 13.24 2.59 2.87 21.0 118.0 2.80   
  
 flavanoids nonflavanoid\_phenols proanthocyanins color\_intensity ... \  
0 3.06 0.28 2.29 5.64 ...   
1 2.76 0.26 1.28 4.38 ...   
2 3.24 0.30 2.81 5.68 ...   
3 3.49 0.24 2.18 7.80 ...   
4 2.69 0.39 1.82 4.32 ...   
  
 alcalinity\_of\_ash\_scaled magnesium\_scaled total\_phenols\_scaled \  
0 0.257732 0.619565 0.627586   
1 0.030928 0.326087 0.575862   
2 0.412371 0.336957 0.627586   
3 0.319588 0.467391 0.989655   
4 0.536082 0.521739 0.627586   
  
 flavanoids\_scaled nonflavanoid\_phenols\_scaled proanthocyanins\_scaled \  
0 0.573840 0.283019 0.593060   
1 0.510549 0.245283 0.274448   
2 0.611814 0.320755 0.757098   
3 0.664557 0.207547 0.558360   
4 0.495781 0.490566 0.444795   
  
 color\_intensity\_scaled hue\_scaled od280/od315\_of\_diluted\_wines\_scaled \  
0 0.372014 0.455285 0.970696   
1 0.264505 0.463415 0.780220   
2 0.375427 0.447154 0.695971   
3 0.556314 0.308943 0.798535   
4 0.259386 0.455285 0.608059   
  
 proline\_scaled   
0 0.561341   
1 0.550642   
2 0.646933   
3 0.857347   
4 0.325963   
  
[5 rows x 27 columns]

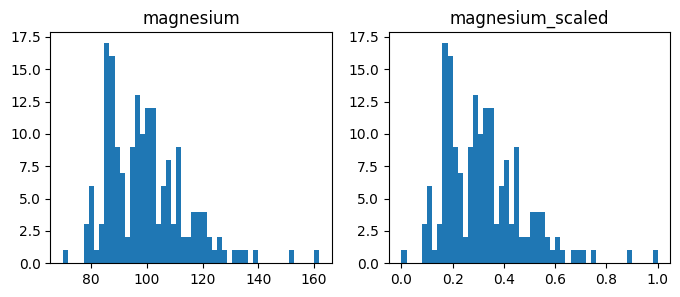
# Проверим, что масштабирование не повлияло на распределение данных  
for col in scale\_cols:  
 col\_scaled = col + '\_scaled'  
  
 fig, ax = plt.subplots(1, 2, figsize=(8,3))  
 ax[0].hist(data\_all[col], 50)  
 ax[1].hist(data\_all[col\_scaled], 50)  
 ax[0].title.set\_text(col)  
 ax[1].title.set\_text(col\_scaled)  
 plt.show()

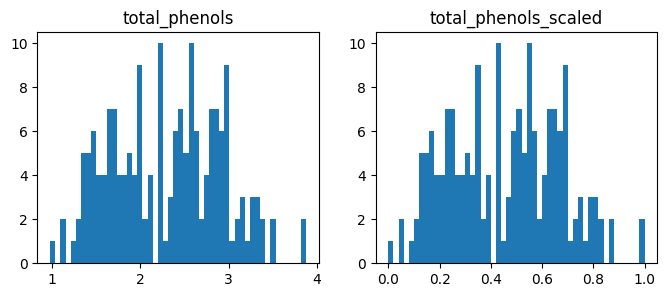


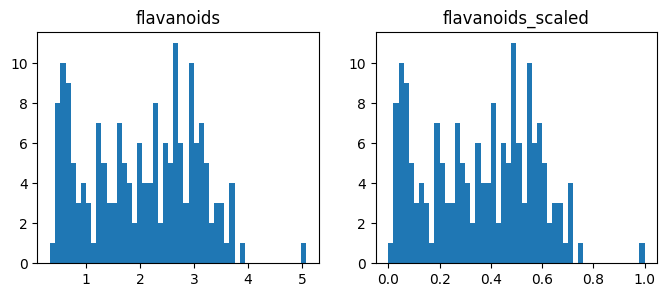


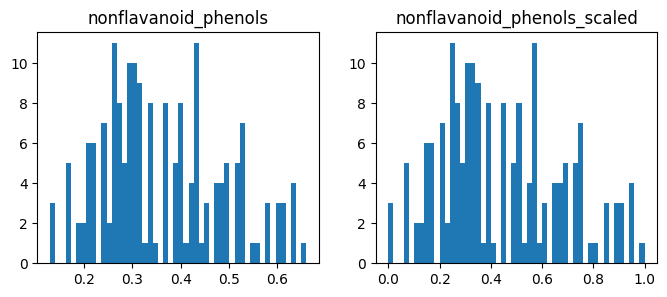


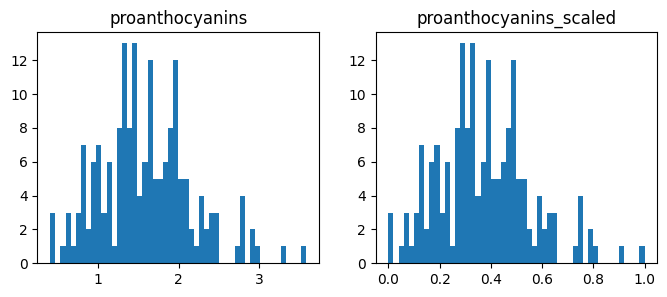


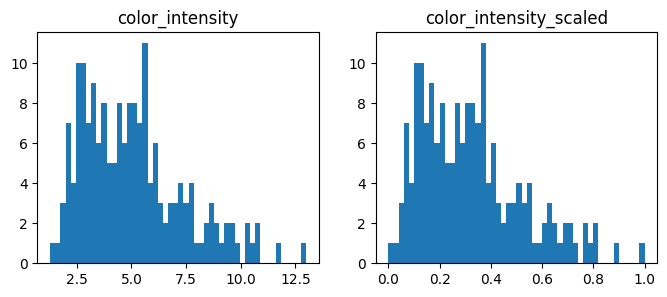


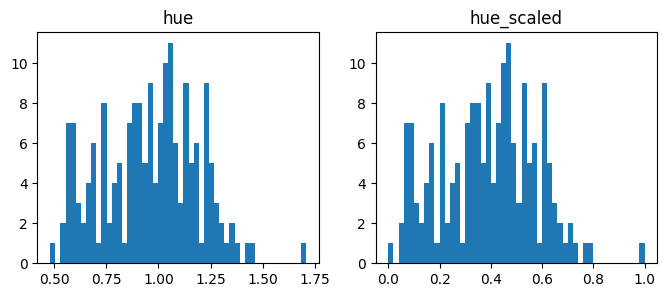


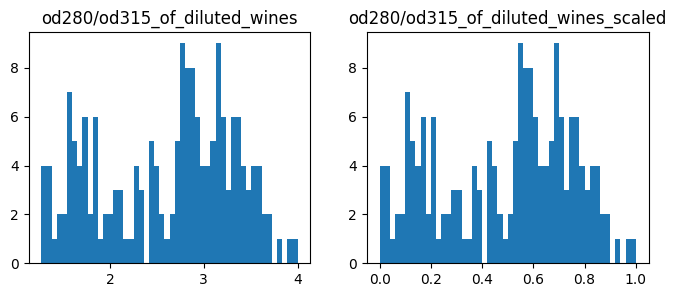


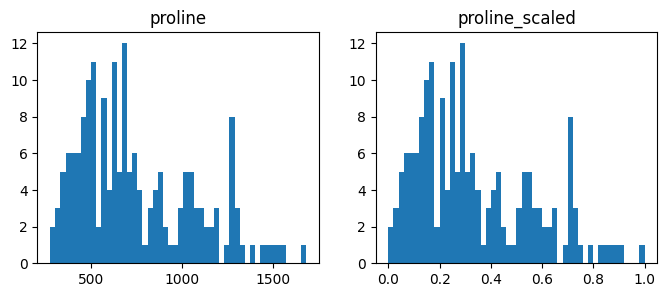












## Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

corr\_cols\_1 = scale\_cols + ['target']  
corr\_cols\_1

['alcohol',  
 'malic\_acid',  
 'ash',  
 'alcalinity\_of\_ash',  
 'magnesium',  
 'total\_phenols',  
 'flavanoids',  
 'nonflavanoid\_phenols',  
 'proanthocyanins',  
 'color\_intensity',  
 'hue',  
 'od280/od315\_of\_diluted\_wines',  
 'proline',  
 'target']

df\_not\_scaled = data\_all[corr\_cols\_1]

df\_not\_scaled.head()

alcohol malic\_acid ash alcalinity\_of\_ash magnesium total\_phenols \  
0 14.23 1.71 2.43 15.6 127.0 2.80   
1 13.20 1.78 2.14 11.2 100.0 2.65   
2 13.16 2.36 2.67 18.6 101.0 2.80   
3 14.37 1.95 2.50 16.8 113.0 3.85   
4 13.24 2.59 2.87 21.0 118.0 2.80   
  
 flavanoids nonflavanoid\_phenols proanthocyanins color\_intensity hue \  
0 3.06 0.28 2.29 5.64 1.04   
1 2.76 0.26 1.28 4.38 1.05   
2 3.24 0.30 2.81 5.68 1.03   
3 3.49 0.24 2.18 7.80 0.86   
4 2.69 0.39 1.82 4.32 1.04   
  
 od280/od315\_of\_diluted\_wines proline target   
0 3.92 1065.0 0.0   
1 3.40 1050.0 0.0   
2 3.17 1185.0 0.0   
3 3.45 1480.0 0.0   
4 2.93 735.0 0.0

scale\_cols\_postfix = [x+'\_scaled' for x in scale\_cols]  
corr\_cols\_2 = scale\_cols\_postfix + ['target']  
corr\_cols\_2

['alcohol\_scaled',  
 'malic\_acid\_scaled',  
 'ash\_scaled',  
 'alcalinity\_of\_ash\_scaled',  
 'magnesium\_scaled',  
 'total\_phenols\_scaled',  
 'flavanoids\_scaled',  
 'nonflavanoid\_phenols\_scaled',  
 'proanthocyanins\_scaled',  
 'color\_intensity\_scaled',  
 'hue\_scaled',  
 'od280/od315\_of\_diluted\_wines\_scaled',  
 'proline\_scaled',  
 'target']

df\_scaled = data\_all[corr\_cols\_2]

df\_scaled.head()

alcohol\_scaled malic\_acid\_scaled ash\_scaled alcalinity\_of\_ash\_scaled \  
0 0.842105 0.191700 0.572193 0.257732   
1 0.571053 0.205534 0.417112 0.030928   
2 0.560526 0.320158 0.700535 0.412371   
3 0.878947 0.239130 0.609626 0.319588   
4 0.581579 0.365613 0.807487 0.536082   
  
 magnesium\_scaled total\_phenols\_scaled flavanoids\_scaled \  
0 0.619565 0.627586 0.573840   
1 0.326087 0.575862 0.510549   
2 0.336957 0.627586 0.611814   
3 0.467391 0.989655 0.664557   
4 0.521739 0.627586 0.495781   
  
 nonflavanoid\_phenols\_scaled proanthocyanins\_scaled \  
0 0.283019 0.593060   
1 0.245283 0.274448   
2 0.320755 0.757098   
3 0.207547 0.558360   
4 0.490566 0.444795   
  
 color\_intensity\_scaled hue\_scaled od280/od315\_of\_diluted\_wines\_scaled \  
0 0.372014 0.455285 0.970696   
1 0.264505 0.463415 0.780220   
2 0.375427 0.447154 0.695971   
3 0.556314 0.308943 0.798535   
4 0.259386 0.455285 0.608059   
  
 proline\_scaled target   
0 0.561341 0.0   
1 0.550642 0.0   
2 0.646933 0.0   
3 0.857347 0.0   
4 0.325963 0.0

fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(15,5))  
fig.suptitle('Корреляционная матрица (до масштабирования)')  
sns.heatmap(df\_not\_scaled.corr(), ax=ax, annot=True, fmt='.3f')

<Axes: >

fig, ax = plt.subplots(1, 1, sharex='col', sharey='row', figsize=(15,5))  
fig.suptitle('Корреляционная матрица (после масштабирования)')  
sns.heatmap(df\_scaled.corr(), ax=ax, annot=True, fmt='.3f')

<Axes: >

Корреляционная матрица содержит коэффициенты корреляции между всеми парами признаков.

Корреляционная матрица симметрична относительно главной диагонали. На главной диагонали расположены единицы (корреляция признака самого с собой).

На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабированных данных совпадают.
2. Целевой признак наиболее сильно коррелирует с щелочностью пепла (0.52) и отрицательно коррелирует с флаваноидами (-0.85). Эти признаки обязательно следует оставить в модели.
3. Целевой признак слабо коррелирует с пеплом (-0.05). Скорее всего, этот признак стоит исключить из модели, возможно, он только ухудшит качество модели.
4. Целевой признак отчасти коррелирует с температурой (0.54). Этот признак стоит также оставить в модели.
5. Остальные признаки отчасти коррелируют как между собой, так и с целевым признаком. Их стои оставить в модели.
6. Большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

## Выбор метрик для последующей оценки качества моделей

Для задачи классификации будем использовать следующие модели:

1. Логистическая регрессия
2. Метод ближайших соседей
3. Машина опорных векторов
4. Решающее дерево
5. Бэггинг
6. Градиентный бустинг

## Формирование обучающей и тестовой выборок на основе исходного набора данных

# На основе масштабированных данных выделим обучающую и тестовую выборки  
  
y = df\_scaled['target']  
x = df\_scaled.drop('target', axis = 1).drop('ash\_scaled', axis = 1)  
  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 7)  
  
x\_train.shape, x\_test.shape

((106, 12), (72, 12))

## 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.

# Модели  
clas\_models = {'LogisticRegression': LogisticRegression(),   
 'KNN\_10':KNeighborsClassifier(n\_neighbors=10),  
 'SVC':SVC(probability=True),  
 'DecisionTree':DecisionTreeClassifier(),  
 'Bagging':BaggingClassifier(),  
 'GradientBoosting':GradientBoostingClassifier()}

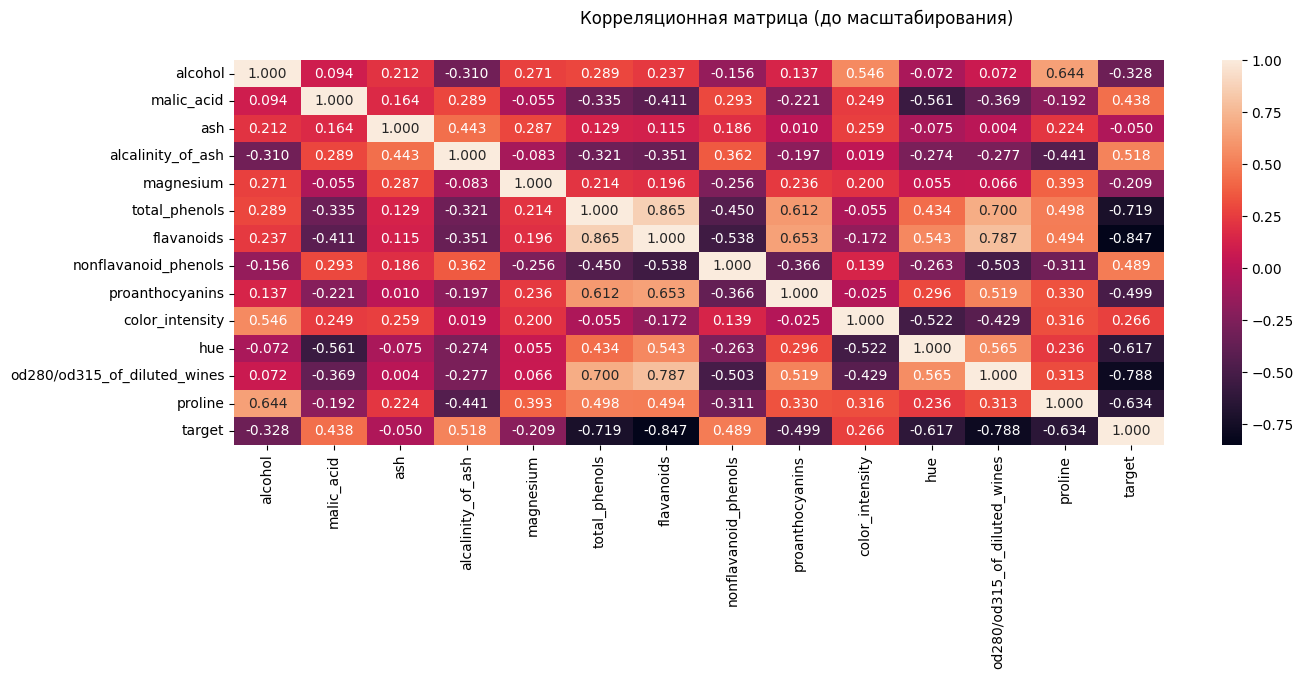
class MetricLogger:  
   
 def \_\_init\_\_(self):  
 self.df = pd.DataFrame(  
 {'metric': pd.Series([], dtype='str'),  
 'alg': pd.Series([], dtype='str'),  
 'value': pd.Series([], dtype='float')})  
  
 def add(self, metric, alg, value):  
 """  
 Добавление значения  
 """  
 # Удаление значения если оно уже было ранее добавлено  
 self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)  
 # Добавление нового значения  
 temp = [{'metric':metric, 'alg':alg, 'value':value}]  
 self.df = self.df.append(temp, ignore\_index=True)  
  
 def get\_data\_for\_metric(self, metric, ascending=True):  
 """  
 Формирование данных с фильтром по метрике  
 """  
 temp\_data = self.df[self.df['metric']==metric]  
 temp\_data\_2 = temp\_data.sort\_values(by='value', ascending=ascending)  
 return temp\_data\_2['alg'].values, temp\_data\_2['value'].values  
   
 def plot(self, str\_header, metric, ascending=True, figsize=(5, 5)):  
 """  
 Вывод графика  
 """  
 array\_labels, array\_metric = self.get\_data\_for\_metric(metric, ascending)  
 fig, ax1 = plt.subplots(figsize=figsize)  
 pos = np.arange(len(array\_metric))  
 rects = ax1.barh(pos, array\_metric,  
 align='center',  
 height=0.5,   
 tick\_label=array\_labels)  
 ax1.set\_title(str\_header)  
 for a,b in zip(pos, array\_metric):  
 plt.text(0.5, a-0.05, str(round(b,3)), color='white')  
 plt.show()

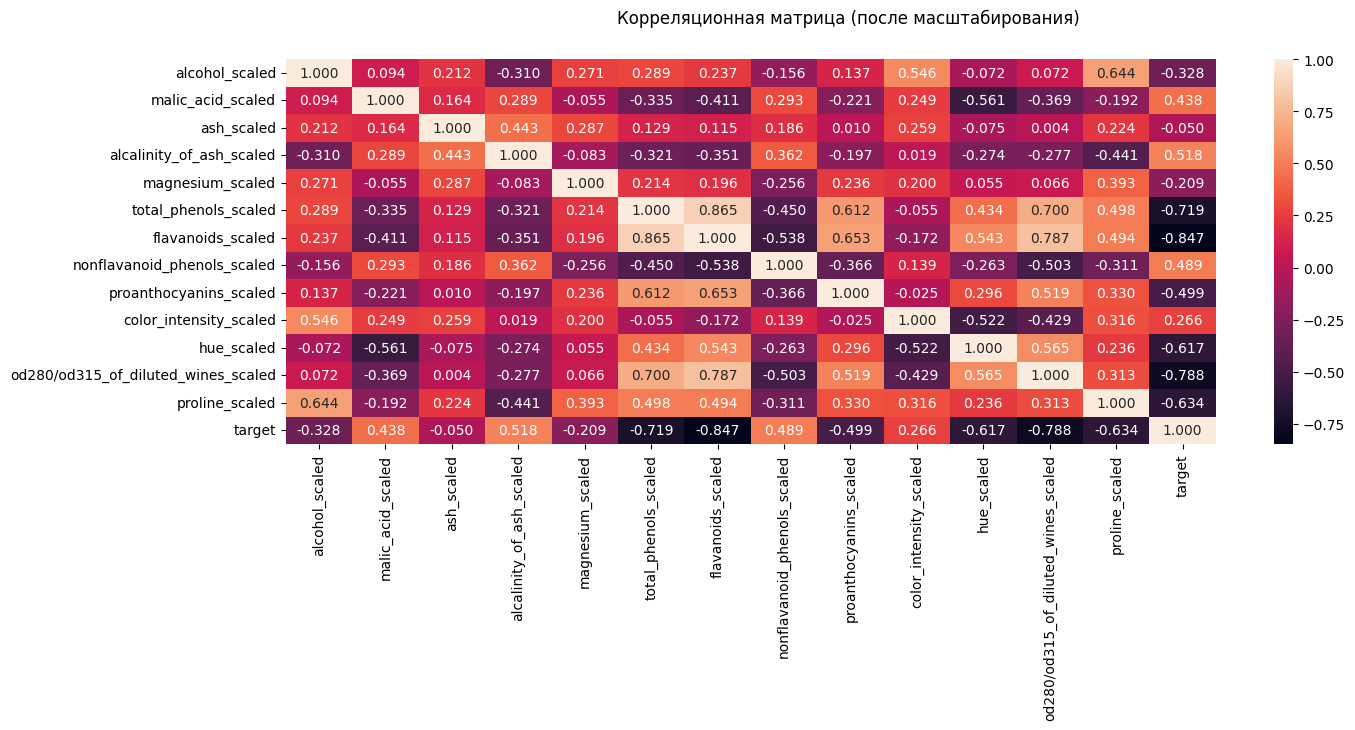
# Сохранение метрик  
clasMetricLogger = MetricLogger()

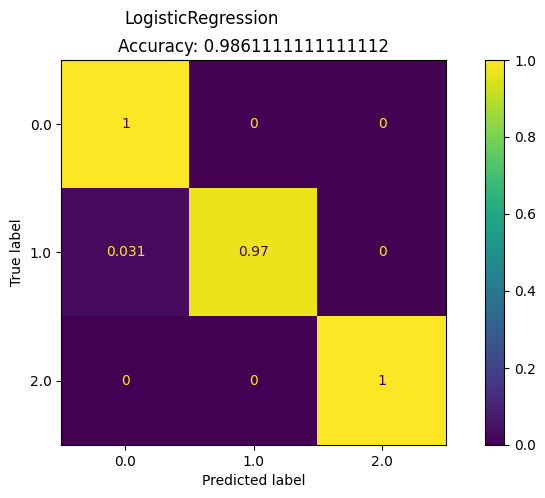
def clas\_train\_model(model\_name, model, clasMetricLogger):  
 model.fit(x\_train, y\_train)  
 # Предсказание значений  
 Y\_pred = model.predict(x\_test)  
  
 accuracy = accuracy\_score(y\_test.values, Y\_pred)  
  
 clasMetricLogger.add('accuracy', model\_name, accuracy)  
   
 fig, ax = plt.subplots(nrows=1, figsize=(10,5))   
  
 cm = confusion\_matrix(y\_test, Y\_pred, labels=np.unique(df\_scaled.target), normalize='true')  
 disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(df\_scaled.target))  
 disp.plot(ax=ax)  
 ax.set\_title("Accuracy: {}".format(accuracy\_score(y\_test.values, Y\_pred)))  
  
 fig.suptitle(model\_name)  
 plt.show()

for model\_name, model in clas\_models.items():  
 clas\_train\_model(model\_name, model, clasMetricLogger)

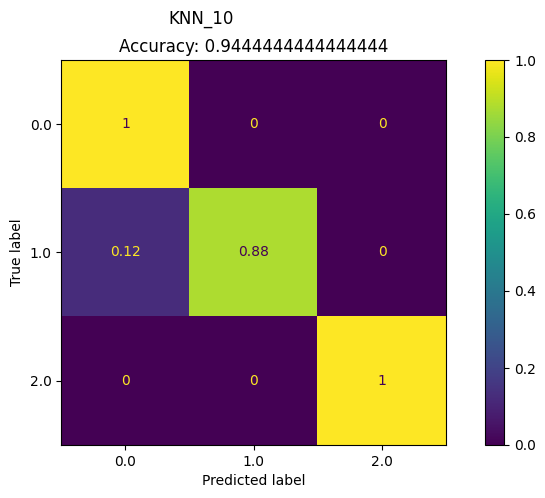
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)



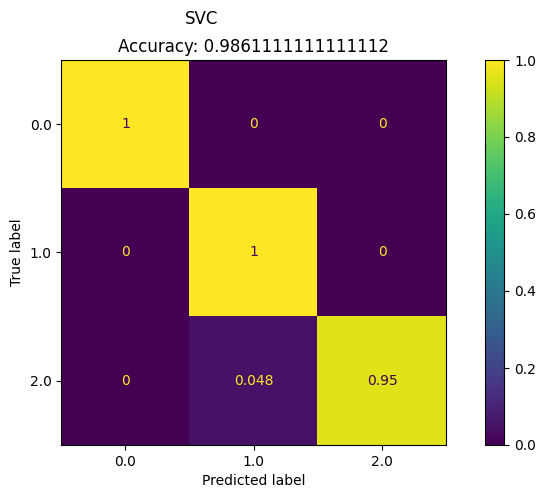




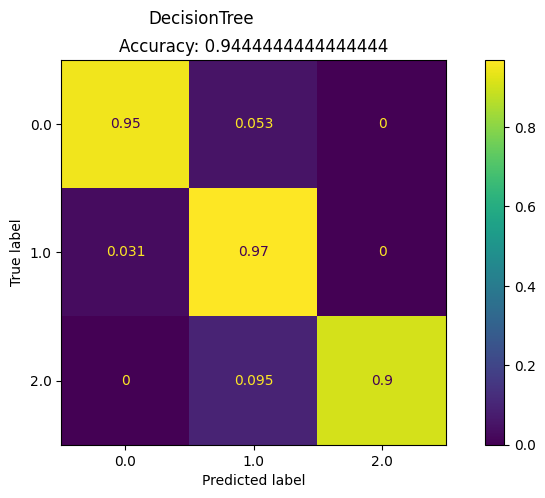
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)



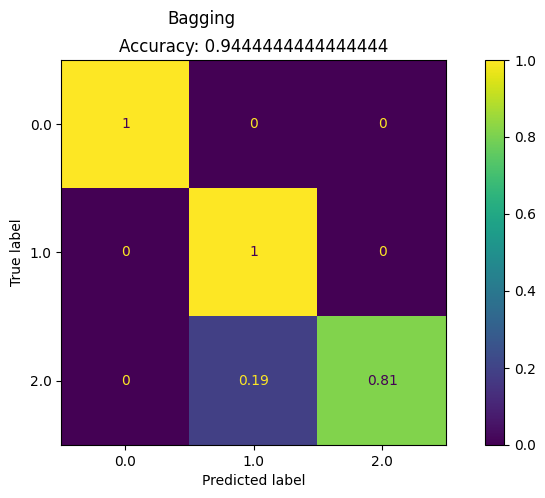
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)



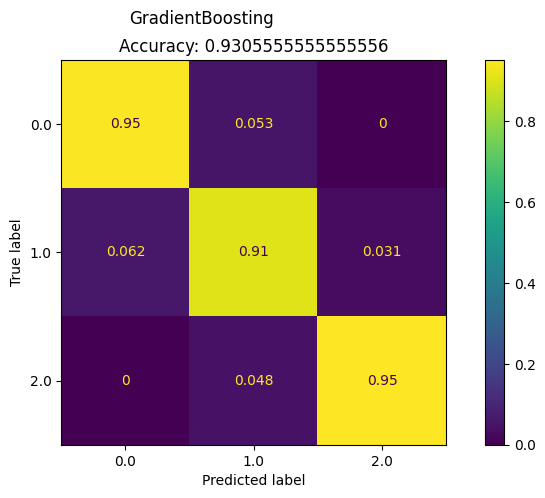
/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)



/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)



/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)



## 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.

x\_train.shape

(106, 12)

n\_range = np.array(range(2,31,1))  
tuned\_parameters = [{'n\_neighbors': n\_range}]  
tuned\_parameters

[{'n\_neighbors': array([ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])}]

%%time  
clf\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=5, scoring='accuracy')  
clf\_gs.fit(x\_train, y\_train)

CPU times: user 641 ms, sys: 62.8 ms, total: 704 ms  
Wall time: 388 ms

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
 param\_grid=[{'n\_neighbors': array([ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,  
 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])}],  
 scoring='accuracy')

# Лучшая модель  
clf\_gs.best\_estimator\_

KNeighborsClassifier(n\_neighbors=9)

clf\_gs\_best\_params\_txt = str(clf\_gs.best\_params\_['n\_neighbors'])  
clf\_gs\_best\_params\_txt

'9'

# Изменение качества на тестовой выборке в зависимости от К-соседей  
plt.plot(n\_range, clf\_gs.cv\_results\_['mean\_test\_score'])

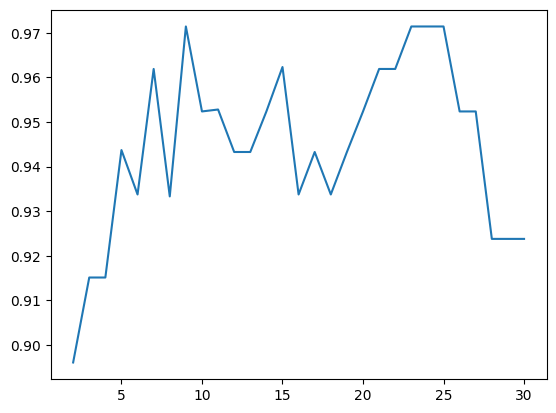
[<matplotlib.lines.Line2D at 0x7f91257a0b20>]

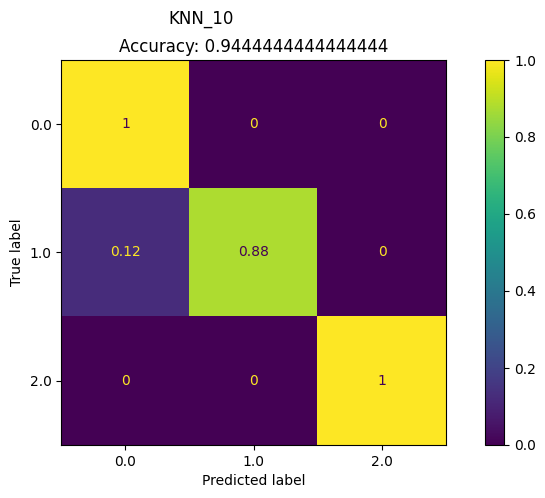
## Повторение для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

clas\_models\_grid = {'KNN\_10':KNeighborsClassifier(n\_neighbors=10),   
 str('KNN\_' + clf\_gs\_best\_params\_txt):clf\_gs.best\_estimator\_}

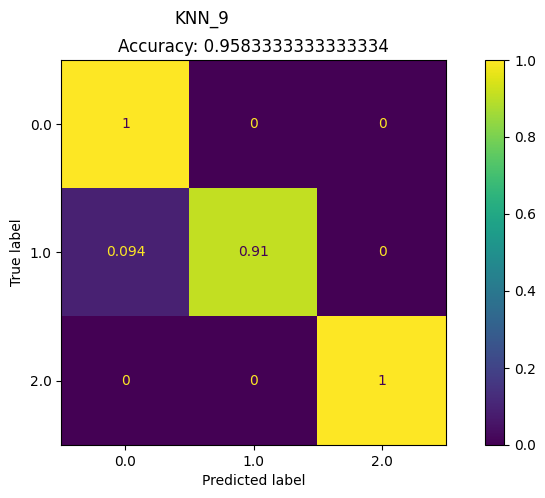
for model\_name, model in clas\_models\_grid.items():  
 clas\_train\_model(model\_name, model, clasMetricLogger)

/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)





/var/folders/jr/9bbh58ys2356w7ntm3zfqcqw0000gn/T/ipykernel\_61763/368643467.py:17: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.  
 self.df = self.df.append(temp, ignore\_index=True)

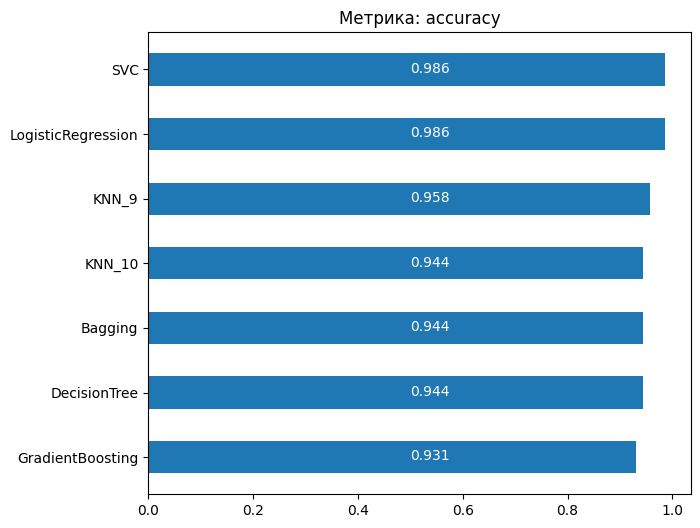


## Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания.

# Метрики качества модели  
clas\_metrics = clasMetricLogger.df['metric'].unique()  
clas\_metrics

array(['accuracy'], dtype=object)

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



Вывод: лучшими оказались модели на основе метода опорных векторов и логистической регрессии.