Демьянчук Г.В. ИУ5-22М ДЗ

Задание

Требуется выполнить следующие действия:

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе студент должен построить модели машинного обучения для решения или задачи класси регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для пон Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных п данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов моделей машинного обучения. В зависимости от набора данных, порядок выполнения п изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не мен выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регресси не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпар обучение моделей на основе обучающей выборки и оценка качества моделей на основе
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1 Рекомендуется использовать методы кросс-валидации. В зависимости от используемой применять функцию `GridSearchCV`, использовать перебор параметров в цикле, или испо
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнен моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик

Ход выполнения работы

Выбор набора данных

В качестве набора данных используются датасет с оценкой качества состава португальских в Данный набор данных доступен по следующему адресу: https://www.kaggle.com/uciml/red-win

Текстовое описание набора данных

Набор данных состоит из одного файла `datasets_4458_8204_winequality-red.csv`. Данный фай
колонки:

Входные переменные (на основе физико-химических тестов):

- `fixed acidity` — фиксированная кислотность;
- `volatile acidity` — летучая кислотность;
- `citric acid` — лимонная кислота;
- `residual sugar` — остаточный сахар;
- `chlorides` — хлориды;
- `free sulfur dioxide` — свободный диоксид серы;
- `total sulfur dioxide` — общий диоксид серы;
- `density` — плотность;
- `pH` — pH;
- `sulphates` — сульфаты;
- `alcohol` — содержание алкоголя;

Выходная переменная (на основе сенсорных данных):

- `quality` — качество (оценка от 0 до 10 баллов);

Постановка задачи и предварительный анализ набора данных

Очевидно, что данный набор данных предполагает задачу регрессии, а именно предсказание
сомелье. Остальные колоки предоставляют данные, которые теоретически могут показывать
поставил эксперт и почему.

Проведение разведочного анализа данных

Подключим все необходимые библиотеки:

```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
  import pandas.util.testing as tm
```

Настроим отображение графиков:

```
# Enable inline plots
%matplotlib inline

# Set plot style
sns.set(style="ticks")

# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

Зададим ширину текстового представления данных, чтобы в дальнейшем текст в отчёте влез

```
pd.set_option("display.width", 70)
```

Предварительная подготовка данных

Загрузим описанный выше набор данных:

```
data = pd.read_csv("/content/datasets_4458_8204_winequality-red.csv")
```

Проверим полученные типы:

```
data.dtypes
```

```
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
```

Посмотрим на данные в данном наборе данных:

```
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51

```
df = data.copy()
```

```
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51

```
df.dtypes
```

```
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
```

Проверим размер набора данных:

```
df.shape
```

```
sns.pairplot(df, plot_kws=dict(linewidth=0));
```

```
(1599, 12)
```

Проверим основные статистические характеристики набора данных:

```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	d
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.(
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.(
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.(
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.(
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.(
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.(

Проверим наличие пропусков в данных:

```
df.isnull().sum()
```

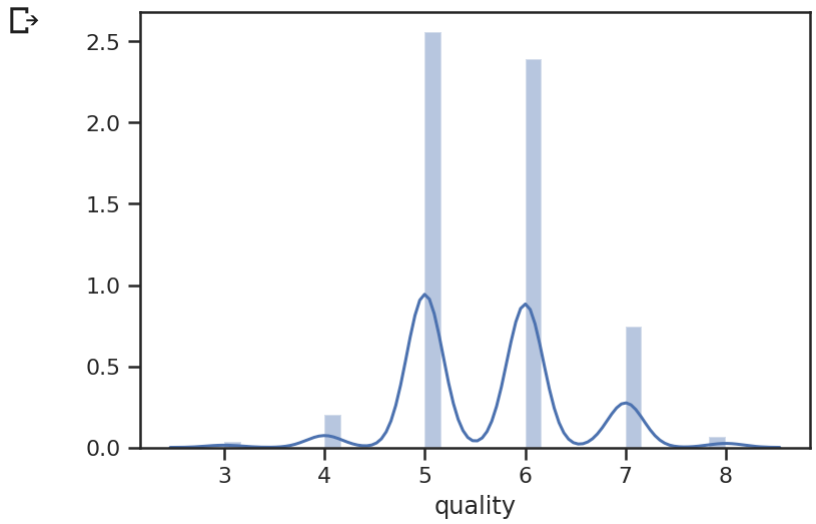
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality   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

Визуальное исследование датасета

Оценим распределение целевого признака — оценки сомелье:

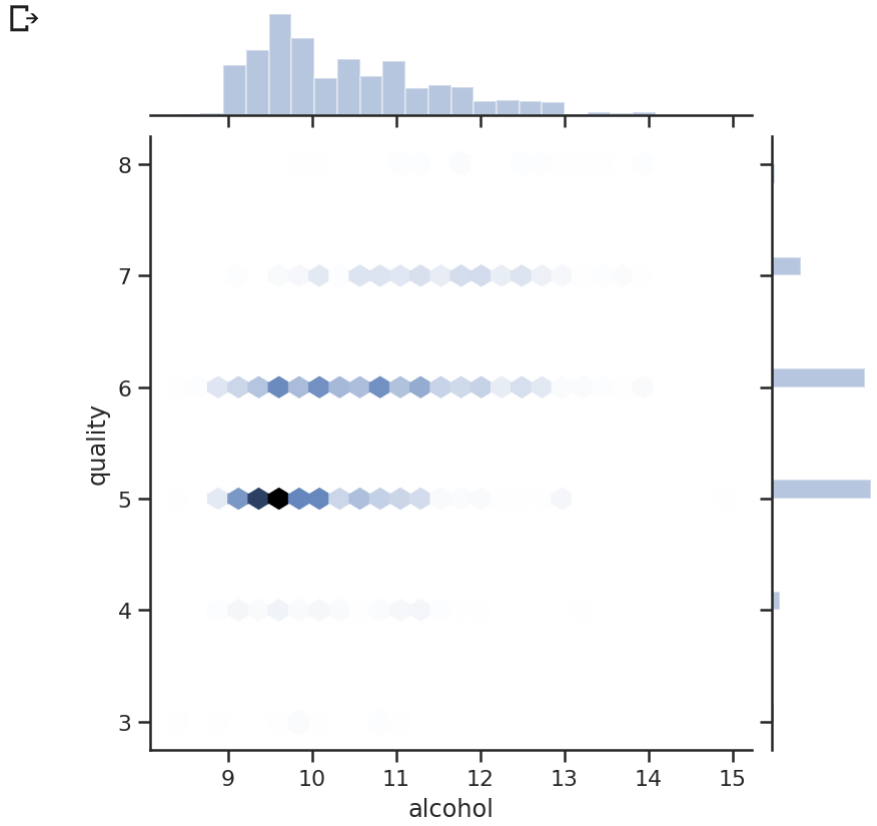
```
di.t.l.t(df["...lit."])
```

```
sns.distplot(df["quality"]);
```



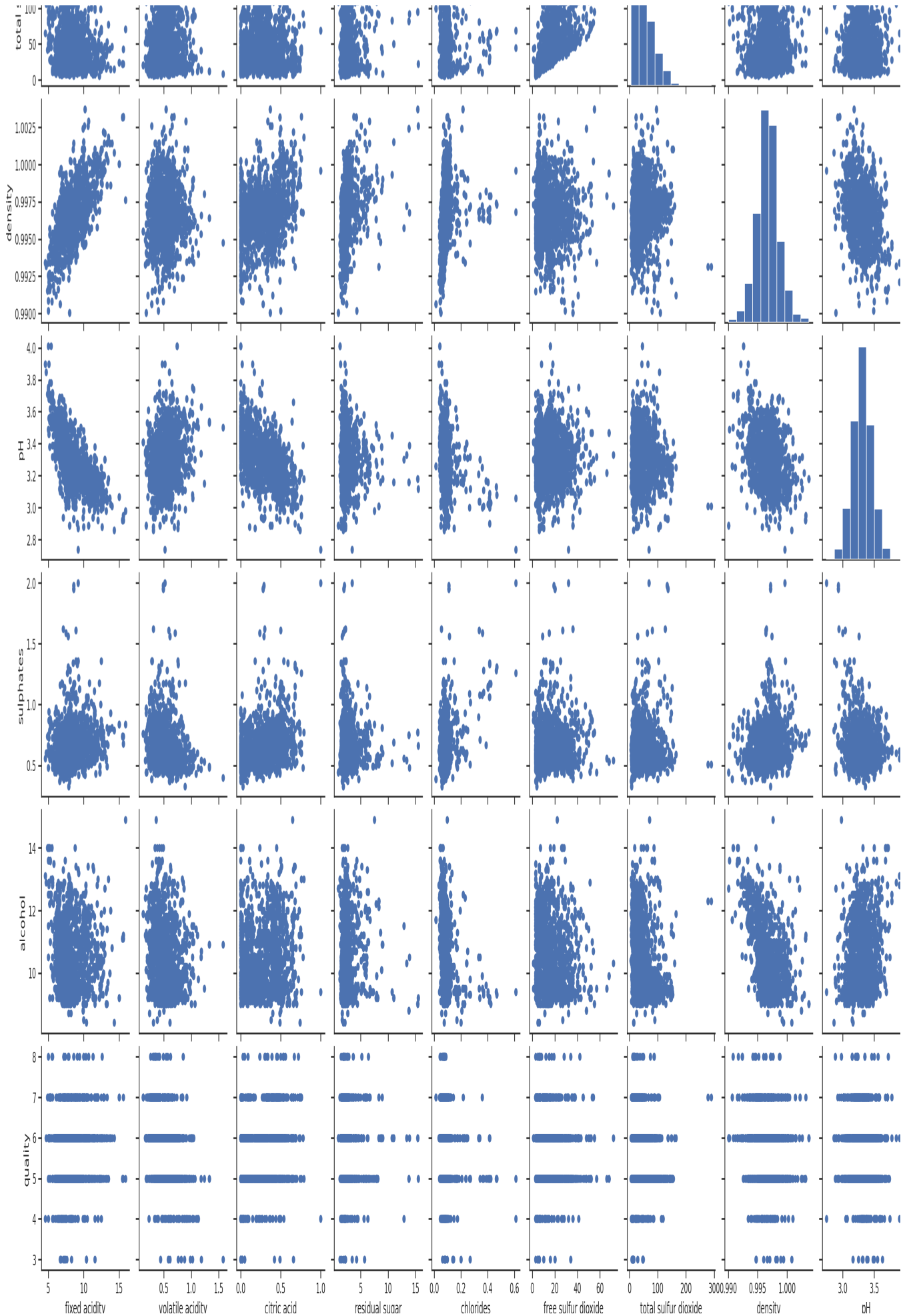
Видно, что оценка большинста вин находится в интервале 5-6. Оценим, насколько оценка зав алкоголя:

```
sns.jointplot(x="alcohol", y="quality", data=df, kind="hex");
```



Видно, что большое содержание алкоголя влияет на оценку не лучшим образом.

Построим парные диаграммы по всем показателям по исходному набору данных:


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

Видно, что зависимости между колонками весьма сложные и в большинстве своём нелинейн
точно определяющего оценку вина, не наблюдается.

Корреляционный анализ

Построим корреляционную матрицу по всему набору данных:

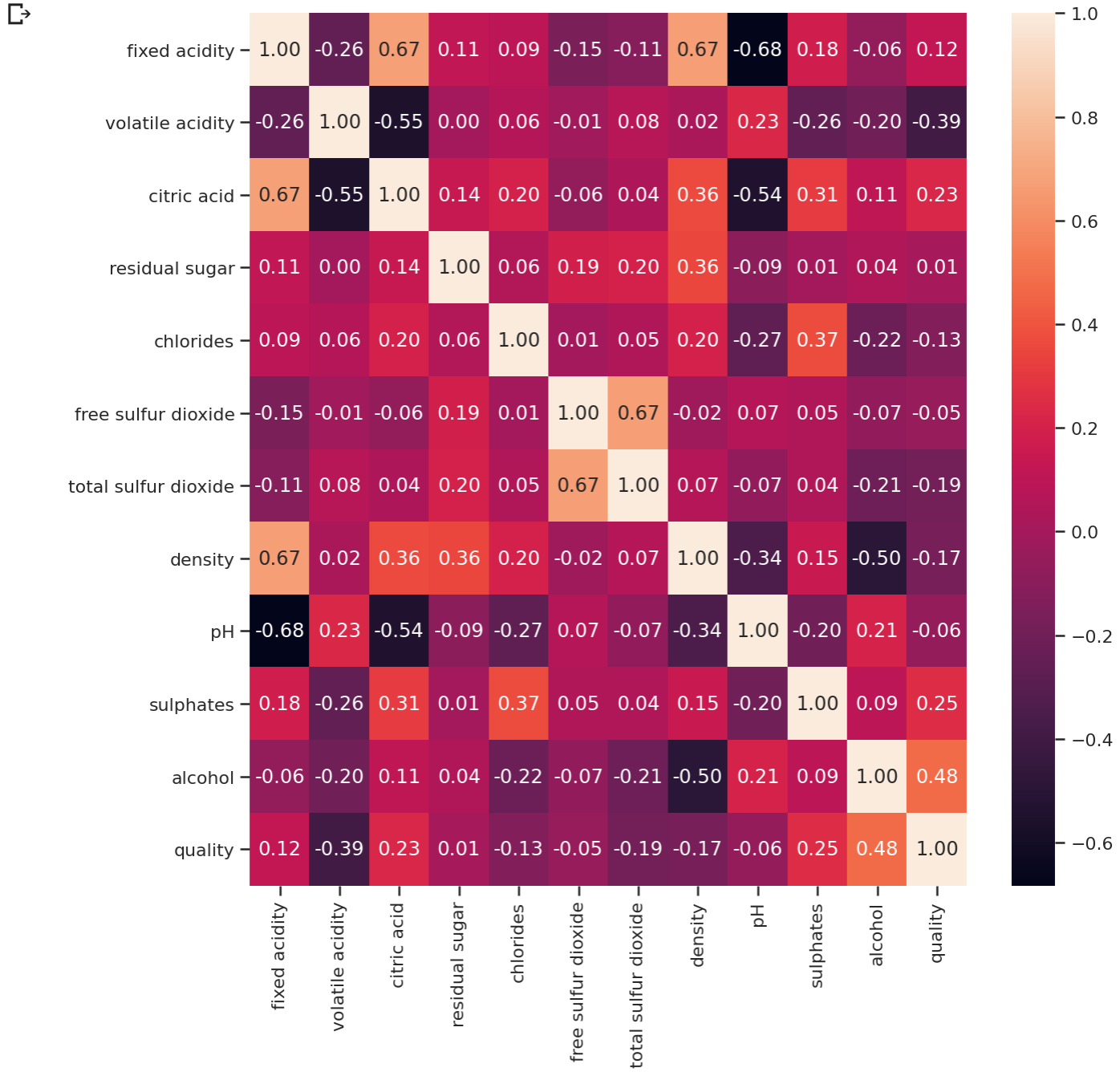
```
df.corr()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	den
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.66
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.02

Визуализируем корреляционную матрицу с помощью тепловой карты:

residual

```
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(df.corr(), annot=True, fmt=".2f",ax=ax);
```



Видно, что оценка заметно коррелирует с содержанием алкоголя, что было показано выше с
Остальные признаки коррелируют друг с другом довольно слабо. Построению моделей маши
не мешает, но насколько хорошо они будут работать — вопрос открытый.

Подготовка данных для обучения моделей

Разделим данные на целевой столбец и признаки:

```
X = df.drop("quality", axis=1)
y = df["quality"]
```

```
print(X.head(), "\n")
print(y.head())
```

```
   fixed acidity  volatile acidity  citric acid  ...    pH  sulphates  alcohol
0            7.4              0.70         0.00  ...  3.51       0.56      9.4
1            7.8              0.88         0.00  ...  3.20       0.68      9.8
2            7.8              0.76         0.04  ...  3.26       0.65      9.8
3           11.2              0.28         0.56  ...  3.16       0.58      9.8
4            7.4              0.70         0.00  ...  3.51       0.56      9.4

[5 rows x 11 columns]

0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

```
print(X.shape)
print(y.shape)
```

```
(1599, 11)
(1599,)
```

Предобработаем данные, чтобы методы работали лучше:

```
from sklearn.preprocessing import StandardScaler

columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sul dio
count	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e
mean	3.435512e-16	1.699704e-16	4.335355e-16	-1.905223e-16	4.838739e-16	1.432042
std	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e
min	-2.137045e+00	-2.278280e+00	-1.391472e+00	-1.162696e+00	-1.603945e+00	-1.422500e
25%	-7.007187e-01	-7.699311e-01	-9.293181e-01	-4.532184e-01	-3.712290e-01	-8.487156
50%	-2.410944e-01	-4.368911e-02	-5.636026e-02	-2.403750e-01	-1.799455e-01	-1.793002
75%	5.057952e-01	6.266881e-01	7.652471e-01	4.341614e-02	5.384542e-02	4.901152
max	4.355149e+00	5.877976e+00	3.743574e+00	9.195681e+00	1.112703e+01	5.367284e

Выбор метрик

Напишем функцию, которая считает метрики построенной модели:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import r2_score

def test_model(model):
    print("mean_absolute_error:",
          mean_absolute_error(y_test, model.predict(X_test)))
    print("median_absolute_error:",
          median_absolute_error(y_test, model.predict(X_test)))
    print("r2_score:",
          r2_score(y_test, model.predict(X_test)))
```

Очевидно, что все эти метрики подходят для задачи регрессии. При этом средняя абсолютна
(`mean_absolute_error`) будет показывать, насколько в среднем мы ошибаемся, медианная аб
(`median_absolute_error`) — насколько мы ошибаемся на половине выборки, а коэффициент де
хорош тем, что он показывает качество модели машинного обучения в задачи регрессии без
моделями.

Выбор моделей

В качестве моделей машинного обучения выберем хорошо показавшие себя в лабораторных

- Метод k ближайших соседей (`KNeighborsRegressor`)
- Дерево решений (`DecisionTreeRegressor`)

- Случайный лес (`RandomForestRegressor`)

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Формирование обучающей и тестовой выборок

Разделим выборку на обучающую и тестовую:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                test_size=0.25, random_state=346705925)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1199, 11)
(400, 11)
(1199,)
(400,)
```

Построение базового решения

Метод k ближайших соседей

Попробуем метод k ближайших соседей с гиперпараметром $k = 5$:

```
knn_5 = KNeighborsRegressor(n_neighbors=5)
knn_5.fit(X_train, y_train)
```

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

Проверим метрики построенной модели:

```
test_model(knn_5)
```

```
mean_absolute_error: 0.506
median_absolute_error: 0.40000000000000036
r2_score: 0.2321892421893451
```

Видно, что данный метод без настройки гиперпараметров показывает неудовлетворительны

Дерево решений

Попробуем дерево решений с неограниченной глубиной дерева:

```
dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)
```

```
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')
```

Проверим метрики построенной модели:

```
test_model(dt_none)
```

```
mean_absolute_error: 0.4325
median_absolute_error: 0.0
r2_score: 0.018470415507346294
```

Видно, что данный метод также без настройки гиперпараметров показывает плохой результа

Случайный лес

Попробуем случайный лес с гиперпараметром $n = 100$:

```
ran_100 = RandomForestRegressor(n_estimators=100)
ran_100.fit(X_train, y_train)
```

```
    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,
```

Проверим метрики построенной модели:

```
                          random_state=None, verbose=0, warm_start=False)
test_model(ran_100)
```

```
    mean_absolute_error: 0.41277500000000006
    median_absolute_error: 0.2699999999999996
    r2_score: 0.44312323535149245
```

Видно, что данный метод даже без настройки гиперпараметров показывает неплохой резуль

Подбор гиперпараметров

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
```

Метод k ближайших соседей

Введем список настраиваемых параметров:

```
param_range = np.arange(1, 50, 2)
tuned_parameters = [{'n_neighbors': param_range}]
tuned_parameters
```

```
    [{'n_neighbors': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31,
                35, 37, 39, 41, 43, 45, 47, 49])}]
```

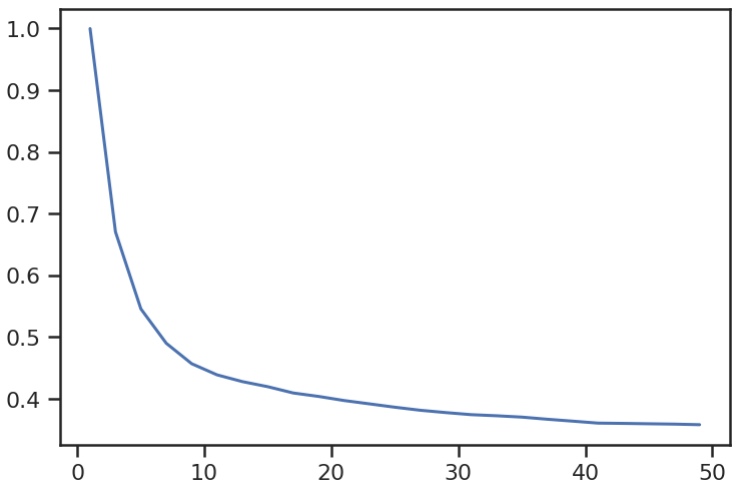
Запустим подбор параметра:

```
gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
    KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                        metric_params=None, n_jobs=None, n_neighbors=47, p=2,
                        weights='uniform')
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данны

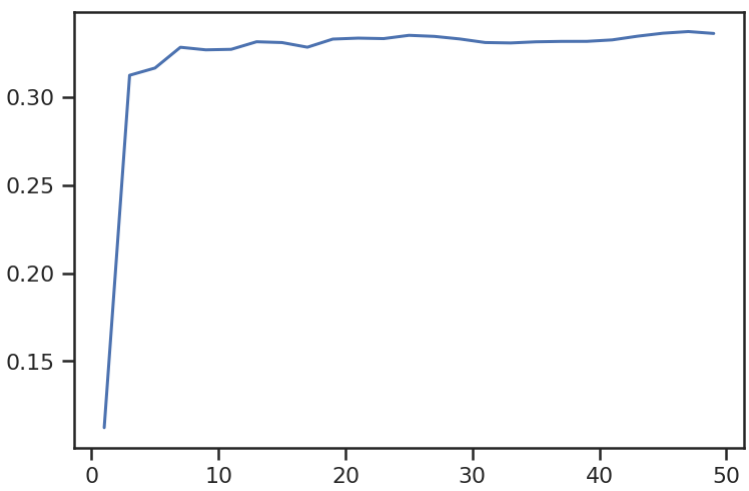
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



В целом результат ожидаемый — чем больше обученных моделей, тем лучше.

На тестовом наборе данных картина похожа:

```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Видно, что наилучший результат достигается при $k = 7$.

```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.5154787234042553
median_absolute_error: 0.4255319148936172
r2_score: 0.32251254867487245
```

Сравним с исходной моделью:

```
test_model(knn_5)
```

```
mean_absolute_error: 0.506
median_absolute_error: 0.40000000000000036
r2_score: 0.2321892421893451
```

Здесь получили улучшение коэффициент детерминации модели.

Дерево решений

Введем список настраиваемых параметров:

```
param_range = np.arange(1, 50, 2)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

```
[{'max_depth': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33
            35, 37, 39, 41, 43, 45, 47, 49])}]
```

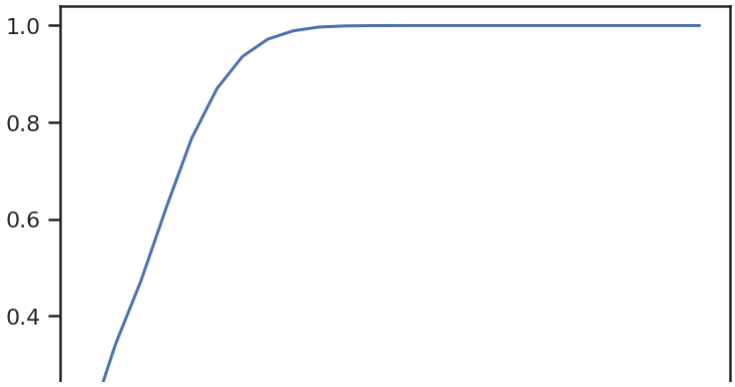
Запустим подбор параметра:

```
gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', 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='best')
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данны

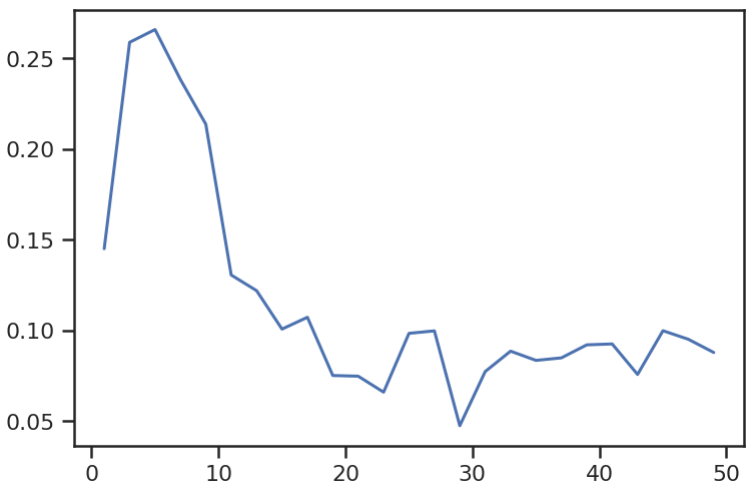
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



В целом результат ожидаемый — чем больше обученных моделей, тем лучше.

На тестовом наборе данных картина похожа:

```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



На графике чётко видно, что модель сначала работает хорошо, а потом начинает переобучать выборке и ухудшается.

```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.5001131329612456
median_absolute_error: 0.4697674418604647
r2_score: 0.2582054476327095
```

Сравним с исходной моделью:

```
test_model(dt_none)
```

```
mean_absolute_error: 0.4325
median_absolute_error: 0.0
r2_score: 0.018470415507346294
```

Конкретно данная модель оказалась заметно лучше, чем исходная.

Случайный лес

Введем список настраиваемых параметров:

```
param_range = np.arange(20, 201, 20)
tuned_parameters = [{'n_estimators': param_range}]
tuned_parameters
```

```
[{'n_estimators': array([ 20,  40,  60,  80, 100, 120, 140, 160, 180, 200])}]
```

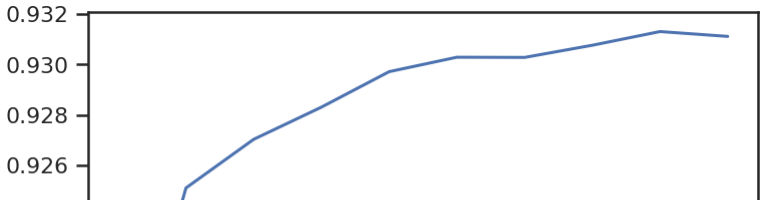
Запустим подбор параметра:

```
gs = GridSearchCV(RandomForestRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

```
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=160, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данны

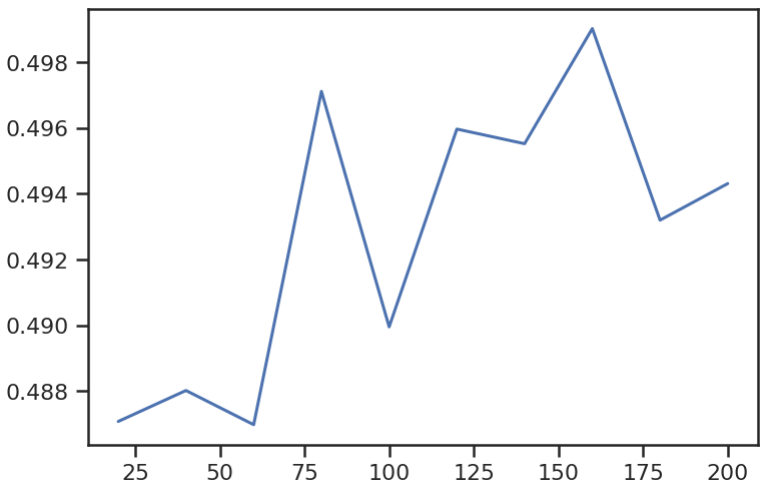
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



В целом результат ожидаемый — чем больше обученных моделей, тем лучше.

На тестовом наборе данных картина похожа:

```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Из-за случайнойсти график немного плавает, но в целом получился чётко выраженный пик с

```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.40973437500000004
median_absolute_error: 0.28125
r2_score: 0.44731770218380085
```

Сравним с исходной моделью:

```
test_model(ran_100)
```

```
mean_absolute_error: 0.41277500000000006
median_absolute_error: 0.2699999999999996
r2_score: 0.44312323535149245
```

Данная модель также оказалась лишь немного лучше, чем исходная.