Домашнее задание №1 - Метод K-ближайших соседей (K-neariest neighbors)

Сегодня мы с вами реализуем наш первый алгоритм машинного обучения, метод K-ближайших соседей. Мы попытаемся решить с помощью него задачи:

- бинарной классификации (то есть, только двум классам)
- многоклассовой классификации (то есть, нескольким классам)
- регрессии (когда зависимая переменная - натуральное число)

Так как методу необходим гиперпараметр (hyperparameter) - количество соседей, то нам нужно научиться подбирать этот параметр. Мы постараемся научиться пользовать numpy для векторизованных вычислений, а также посмотрим на несколько метрик, которые используются в задачах классификации и регрессии.

Перед выполнением задания:

- установите все необходимые библиотеки, запустив `pip install -r requirements.txt`

Если вы раньше не работали с numpy или позабыли его, то можно вспомнить здесь:
http://cs231n.github.io/python-numpy-tutorial/

In [1]:
```
import time
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import random
import pandas as pd


from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from knn import KNNClassifier
from metrics import binary_classification_metrics, multiclass_accuracy
```

In [2]:
```
# plt.rcParams["figure.figsize"] = 12, 9
sns.set_style("whitegrid")

SEED = 111
random.seed(SEED)
np.random.seed(SEED)
```

In [3]:
```
%load_ext autoreload
%autoreload 2
```

Задание 1. KNN на датасете Fashion-MNIST (10 баллов)

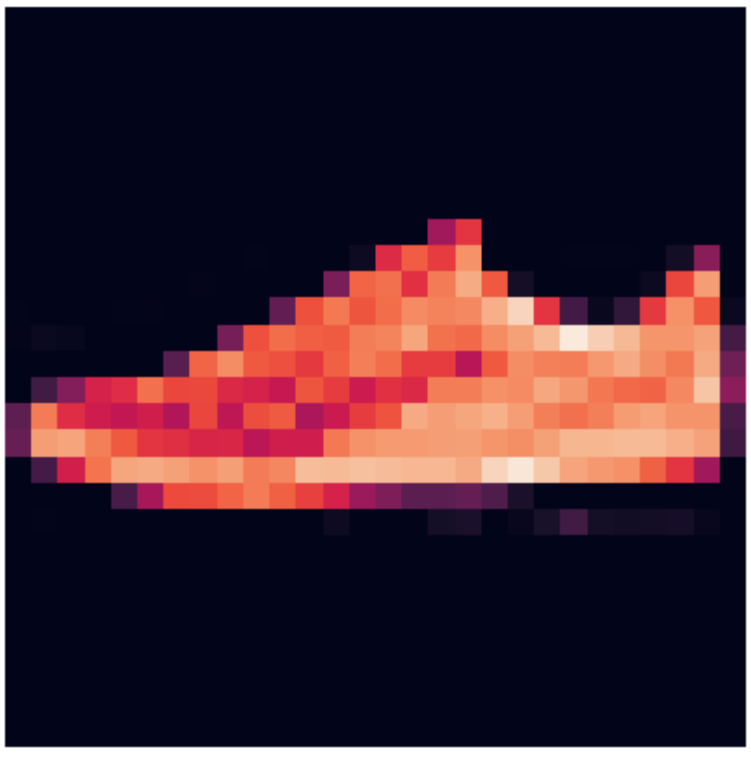
В этом задании вам предстоит поработать с картинками одежды, среди которых можно выделить 10 классов. Данные уже загружены за вас: в переменной X лежат 70000 картинок размером 28 на 28 пикселей, вытянутые в вектор размерностью 784 (28 * 28). Так как данных довольно много, а наш KNN будет весьма медленный, то возьмем случайно 1000 наблюдений (в реальности в зависимости от вашей реализации можно будет взять больше, но если будет не зватать ОЗУ, то берите меньше).

In [4]:
```
X, y = fetch_openml(name="Fashion-MNIST", return_X_y=True, as_frame=False)
```

In [5]:
```
idx_to_stay = np.random.choice(np.arange(X.shape[0]), replace=False, size=1000)
X = X[idx_to_stay]
y = y[idx_to_stay]
```

Давайте посмотрим на какое-нибудь изображение из наших данных:

In [6]:
```
# возьмем случайную картинку и сделаем reshape
# 28, 28, 1 = H, W, C (число каналов, в данном случае 1)
image = X[np.random.choice(np.arange(X.shape[0]))].reshape(28, 28, 1)
plt.imshow(image)
plt.axis("off");
```



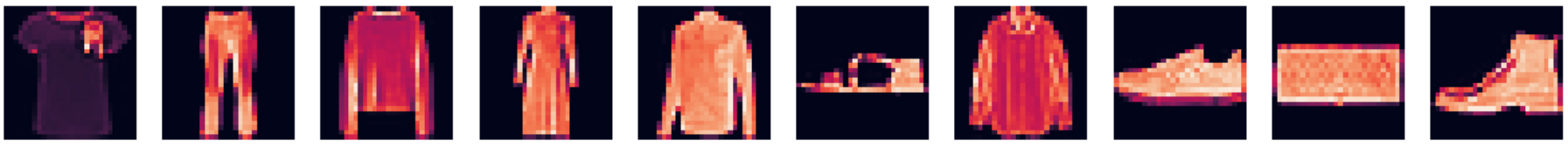
1.1. Посмотрим на все классы (0.5 баллов)

Возьмите по одной картинке каждого класса и изобразите их (например, сделайте subplots 5 на 2).

In [7]:
```
classes = np.unique(y)
```

In [8]:
```
Y_series = pd.Series(y)

indexes = []
for i in range(10):
    indexes.append(Y_series.loc[Y_series == str(i)].index[0])
```

In [9]:
```
images = []
for i in indexes:
    images.append(X[i])

fig, axs = plt.subplots(1, 10, figsize=(15, 5))

for i in range(0, 10):
    axs[i].imshow(images[i].reshape(28,28,1))
    axs[i].axis('off')
```



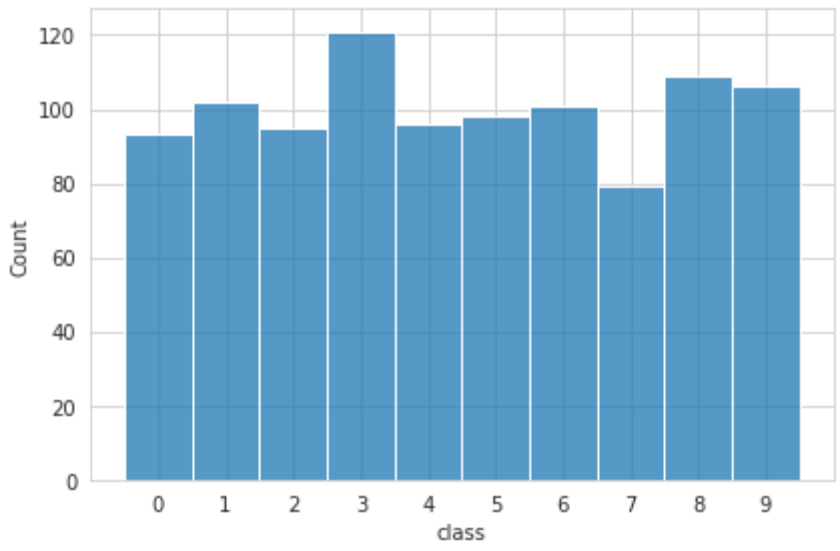
1.2. Сделайте небольшой EDA (1 балл)

Посмотрите на баланс классов. В дальнейших домашках делайте EDA, когда считаете нужным, он нужен почти всегда, но оцениваться это уже не будет, если не будет указано иное. Делайте EDA, чтобы узнать что-то новое о данных!

In [14]:
```
pd.Series(y).describe()
```

Out[14]:
```
count     1000
unique      10
top          3
freq       121
dtype: object
```

In [15]:
```
fig, axes = plt.subplots(figsize=(6, 4), tight_layout=True);
sns.histplot(sorted(y));
plt.xlabel('class');
```



1.3. Разделите данные на train и test (0.5 баллов)

Разделите данные на тренировочную и тестовую выборки, размеры тестовой выборки выберите сами. Здесь вам может помочь функция `train_test_split`

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

1.4. KNN для бинарной классификации (6 баллов)

Давайте возьмем для задачи бинарной классификации только объекты с метками классов 0 и 1.

In [17]:
```
binary_train_y = y_train[(y_train == '1') | (y_train == '0')]
binary_train_X = X_train[:binary_train_y.shape[0]]

binary_test_y = y_test[(y_test == '1') | (y_test == '0')]
binary_test_X = X_test[:binary_test_y.shape[0]]
```

И вот мы подготовили данные, но модели у нас пока что нет. В нескольких занятиях нашего курса вам придется самостоятельно реализовывать какие-то алгоритмы машинного обучения, а потом сравнивать их с готовыми библиотечными решениями. В остальных заданиях реализовывать алгоритмы будет не обязательно, но может быть полезно, поэтому часто это будут задания на дополнительные баллы, но главное не это, а понимание работы алгоритма после его реализации с нуля на простом numpy. Также это все потом можно оформить в виде репозитория ml_from_scratch и хвастаться перед друзьями.

In [18]:
```
knn_classifier = KNNClassifier(k=1)
knn_classifier.fit(binary_train_X, binary_train_y)
```

Настало время писать код!

В KNN нам нужно для каждого тестового примера найти расстояния до всех точек обучающей выборки. Допустим у нас 1000 примеров в train'е и 100 в test'е, тогда в итоге мы бы хотели получить матрицу попарных расстояний (например, размерностью 100 на 1000). Это можно сделать несколькими способами, и кому-то наверняка, в голову приходит идея с двумя вложенными циклами (надеюсь, что не больше:). Так можно делать, то можно и эффективнее. Вообще, в реальном KNN используется структура данных k-d-tree, которая позволяет производить поиск за log(N), а не за N, как будем делать мы (по сути это такое расширение бинарного поиска на многомерное пространство).

Вам нужно будет последовательно реализовать методы `compute_distances_two_loops`, `compute_distances_one_loop` и `compute_distances_no_loops` класса `KNN` в файле `knn.py`.

Эти функции строят массив расстояний между всеми векторами в тестовом наборе и в тренировочном наборе.
В результате они должны построить массив размера `(num_test, num_train)`, где координата `[i][j]` соотвествует расстоянию между i-м вектором в test (`test[i]`) и j-м вектором в train (`train[j]`).

Обратите внимание Для простоты реализации мы будем использовать в качестве расстояния меру L1 (ее еще называют Manhattan distance).

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|$$

В начале я буду иногда писать разные assert'ы, чтобы можно было проверить правильность реализации, в дальнейшем вам нужно будет их писать самим, если нужно будет проверять корректность каких-то вычислений.

In [19]:
```
dists = knn_classifier.compute_distances_two_loops(binary_test_X)
assert np.isclose(dists[0, 100], np.sum(np.abs(binary_test_X[0] - binary_train_X[100])))
```

In [20]:
```
dists = knn_classifier.compute_distances_one_loop(binary_test_X)
assert np.isclose(dists[0, 100], np.sum(np.abs(binary_test_X[0] - binary_train_X[100])))
```

In [21]:
```
dists = knn_classifier.compute_distances_no_loops(binary_test_X)
assert np.isclose(dists[0, 100], np.sum(np.abs(binary_test_X[0] - binary_train_X[100])))
```

Проверим скорость работы реализованных методов

In [22]:
```
%timeit knn_classifier.compute_distances_two_loops(binary_test_X)
%timeit knn_classifier.compute_distances_one_loop(binary_test_X)
%timeit knn_classifier.compute_distances_no_loops(binary_test_X)
```

```
591 ms ± 28.6 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
11 ms ± 2.7 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
56 ms ± 17.1 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Реализуем метод для предсказания меток класса

In [23]:
```
prediction = knn_classifier.predict(binary_test_X)
prediction
```

Out[23]:
```
array([1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0])
```

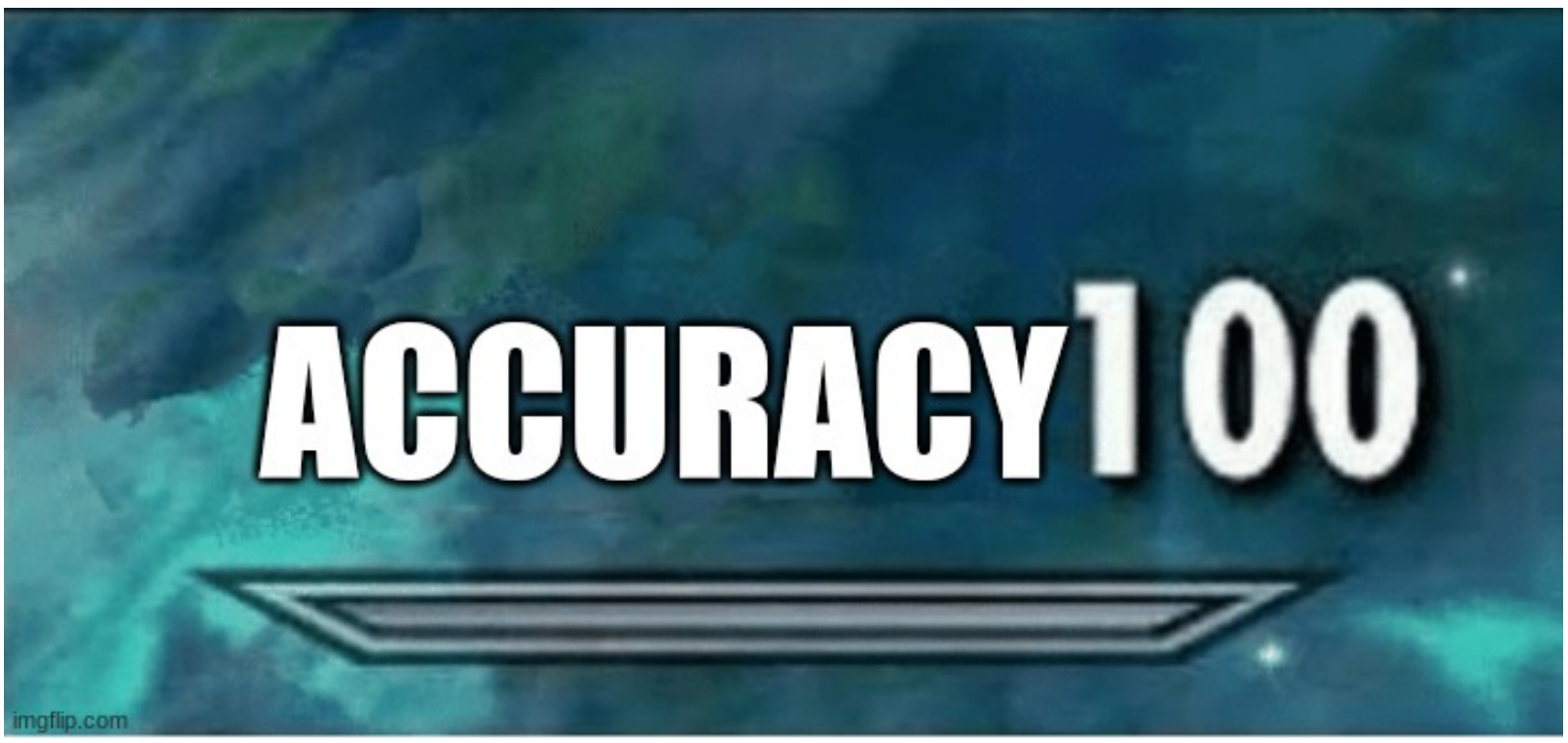
Метрика

Теперь нужно реализовать несколько метрик для бинарной классификации. Не забудьте подумать о численной нестабильности (деление на 0).

In [24]:
```
binary_classification_metrics(prediction, binary_test_y.astype('int64'))
```

```
Accuracy = 0.6428571428571429
precision = 0.7142857142857143
recall = 0.625
f1 = 0.6666666666666666
```

Все ли хорошо с моделью? Можно проверить свою реализацию с функциями из библиотеки `sklearn`:



In [25]:
```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
```

In [26]:
```
accuracy_score(binary_test_y.astype('int64'), prediction)
```

Out[26]:
```
0.6428571428571429
```

In [27]:
```
precision_score(binary_test_y.astype('int64'), prediction)
```

Out[27]:
```
0.7142857142857143
```

In [28]:
```
recall_score(binary_test_y.astype('int64'), prediction)
```

Out[28]:
```
0.625
```

In [30]:
```
f1_score(binary_test_y.astype('int64'), prediction)
```

Out[30]:
```
0.6666666666666666
```

СОВПАДАЕТ!

Подбор оптимального k

Чтобы подрбрать оптимальное значение параметра k можно сделать следующее: задать область допустимых значений k, например, `[1, 3, 5, 10]`. Дальше для каждого k обучить модель на тренировочных данных, сделать предсказания на тестовых и посчитать какую-нибудь метрику (метрику выберите сами исходя из задачи, но постарайтесь обосновать выбор). В конце нужно посмотреть на зависимость метрики на train'e и test'e от k и выбрать подходящее значение.

Реализуйте функцию `choose_best_k` прямо в ноутбуке.

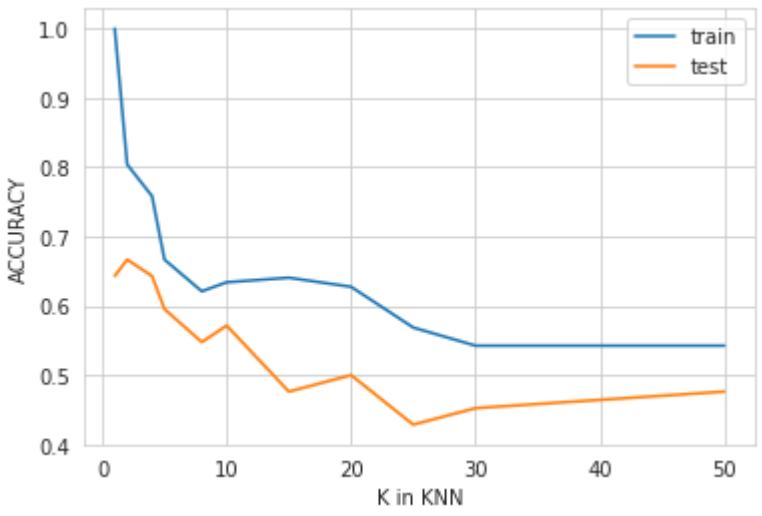
In [49]:
```python
def find_best_k(X_train, y_train, X_test, y_test, params, metric):
    """
    Choose the best k for KKNClassifier
    Arguments:
    X_train, np array (num_train_samples, num_features) - train data
    y_train, np array (num_train_samples) - train labels
    X_test, np array (num_test_samples, num_features) - test data
    y_test, np array (num_test_samples) - test labels
    params, list of hyperparameters for KNN, here it is list of k values
    metric, function for metric calculation
    Returns:
    train_metrics the list of metric values on train data set for each k in params
    test_metrics the list of metric values on test data set for each k in params
    """
    train_metrics, test_metrics = [], []

    for k in params:
        knn_classifier = KNNClassifier(k=k)
        knn_classifier.fit(X_train, y_train)

        y_pred_train = knn_classifier.predict(X_train)
        y_pred_test = knn_classifier.predict(X_test)

        train_metrics.append(metric(y_train.astype('int64'), y_pred_train.astype('int64')))
        test_metrics.append(metric(y_test.astype('int64'), y_pred_test.astype('int64')))

    return train_metrics, test_metrics
```

Я выбрал метрику accuracy, потому что здесь нет "позитивных и отрицательных" исходов, здесь 2 равнозначных класса

То есть True Negative и True Positive имеют одинаковый вклад

In [50]:
```python
params = [1, 2, 4, 5, 8, 10, 15, 20, 25, 30, 50]
train_metrics, test_metrics = find_best_k(binary_train_X, binary_train_y, binary_test_X, binary_test_y, params, acc
```

In [51]:
```python
plt.plot(params, train_metrics, label="train")
plt.plot(params, test_metrics, label="test")
plt.legend()
plt.xlabel("K in KNN")
plt.ylabel("ACCURACY");
```



k = 5 - вполне нормально

На самом деле, это не самый лучший способ подбирать гиперпараметры, но способы получше мы рассмотрим в следующий раз, а пока что выберите оптимальное значение k, сделайте предсказания и посмотрите, насколько хорошо ваша модель предсказывает каждый из классов.

1.5. Многоклассоввая классификация (2 балла)

Теперь нужно научиться предсказывать все 10 классов. Для этого в начале напишем соответствующий метод у нашего классификатора.

In [52]:
```python
knn_classifier = KNNClassifier(k=2)
knn_classifier.fit(X_train, y_train)
```

In [53]:
```
predictions = knn_classifier.predict(X_test)
predictions
```

Out[53]:
```
array(['8', '1', '3', '6', '9', '5', '5', '7', '0', '0', '0', '6', '0',
       '8', '2', '3', '9', '1', '1', '2', '8', '4', '2', '5', '6', '0',
       '2', '1', '2', '7', '0', '7', '4', '0', '4', '8', '4', '9', '7',
       '3', '0', '5', '1', '0', '8', '2', '0', '3', '0', '7', '7', '1',
       '9', '7', '1', '4', '3', '8', '4', '1', '2', '9', '1', '0', '1',
       '8', '9', '5', '5', '6', '6', '1', '2', '4', '7', '3', '3', '9',
       '8', '1', '2', '2', '8', '3', '8', '9', '2', '3', '0', '2', '5',
       '8', '8', '0', '7', '3', '6', '8', '1', '1', '0', '3', '8', '2',
       '7', '0', '9', '5', '1', '3', '3', '9', '4', '4', '2', '3', '4',
       '4', '0', '7', '2', '5', '4', '3', '6', '0', '1', '2', '8', '0',
       '4', '5', '1', '8', '4', '2', '2', '1', '7', '3', '2', '8', '0',
       '2', '9', '2', '4', '1', '4', '3', '0', '8', '0', '5', '9', '9',
       '0', '8', '3', '4', '3', '0', '7', '7', '7', '7', '3', '2', '1',
       '3', '4', '9', '1', '9', '9', '1', '0', '1', '2', '4', '0', '0',
       '7', '7', '3', '2', '3', '1', '3', '2', '8', '5', '5', '5', '9',
       '3', '3', '3', '6', '1'], dtype=object)
```

Осталось реализовать метрику качества для многоклассовой классификации, для этого реализуйте функцию `multiclass_accuracy` в `metrics.py`.

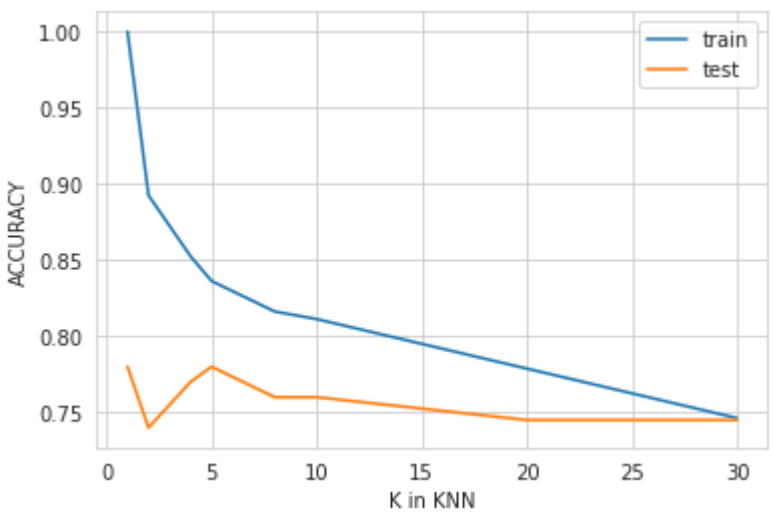
In [43]:
```
multiclass_accuracy(predictions, y_test)
```

Out[43]:
```
0.74
```

Снова выберите оптимальное значение K как мы делали для бинарной классификации.

In [54]:
```
params = [1, 2, 4, 5, 8, 10, 20, 30]
train_metrics, test_metrics = find_best_k(X_train, y_train, X_test, y_test, params, accuracy_score)
```

In [55]:
```
plt.plot(params, train_metrics, label="train")
plt.plot(params, test_metrics, label="test")
plt.legend()
plt.xlabel("K in KNN")
plt.ylabel("ACCURACY");
```



k= 5 - вполне нормально

Задание 2. KNN на датасете diabetes (10 баллов)

Теперь попробуем применить KNN к задаче регрессии. Будем работать с данными о диабете. В этом задании будем использовать класс `KNeighborsRegressor` из библиотеки `sklearn`. Загрузим необходимые библиотеки:

In [56]:
```
from sklearn.datasets import load_diabetes
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor

from metrics import r_squared, mse, mae
```

In [57]:
```
X, y = load_diabetes(as_frame=True, return_X_y=True)
```

In [58]:
```
new_colnames = dict(zip(X.columns[4:],["tc", "ldl", "hdl","tch", "ltg", "glu"]))
X = X.rename(columns=new_colnames)
X.head()
```

Out[58]:

	age	sex	bmi	bp	tc	ldl	hdl	tch	ltg	glu
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641

2.1. EDA (2 обязательных балла + 2 доп. балла за Pipeline)

Сделайте EDA, предобработайте данные так, как считаете нужным, нужна ли в данном случае стандартизация и почему? Не забудте, что если вы стандартизуете данные, то нужно считать среднее и сдандартное отклонение на тренировочной части и с помощью них трансформировать и train, и test (**если не поняли это предложение, то обязательно разберитесь**).

Дополнительно: Попробуйте разобраться с `Pipeline`, чтобы можно было создать класс, который сразу проводит стандартизацию и обучает модель (или делает предсказание). Пайплайны очень удобны, когда нужно применять различные методы предобработки данных (в том числе и к разным столбцам), а также они позволяют правильно интегрировать предобработку данных в различные классы для поиска наилучших гиперпараметров модели (например, `GridSearchCV`).

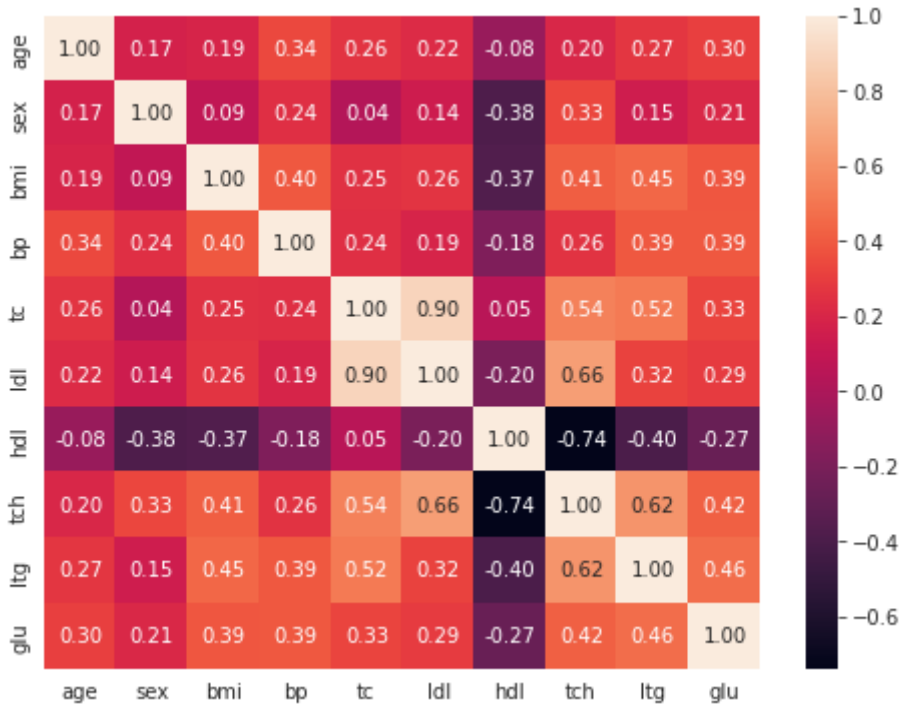
In [59]:
```
from sklearn.pipeline import Pipeline
```

In [60]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   tc      442 non-null    float64
 5   ldl     442 non-null    float64
 6   hdl     442 non-null    float64
 7   tch     442 non-null    float64
 8   ltg     442 non-null    float64
 9   glu     442 non-null    float64
dtypes: float64(10)
memory usage: 34.7 KB
```

Мультиколинеарность

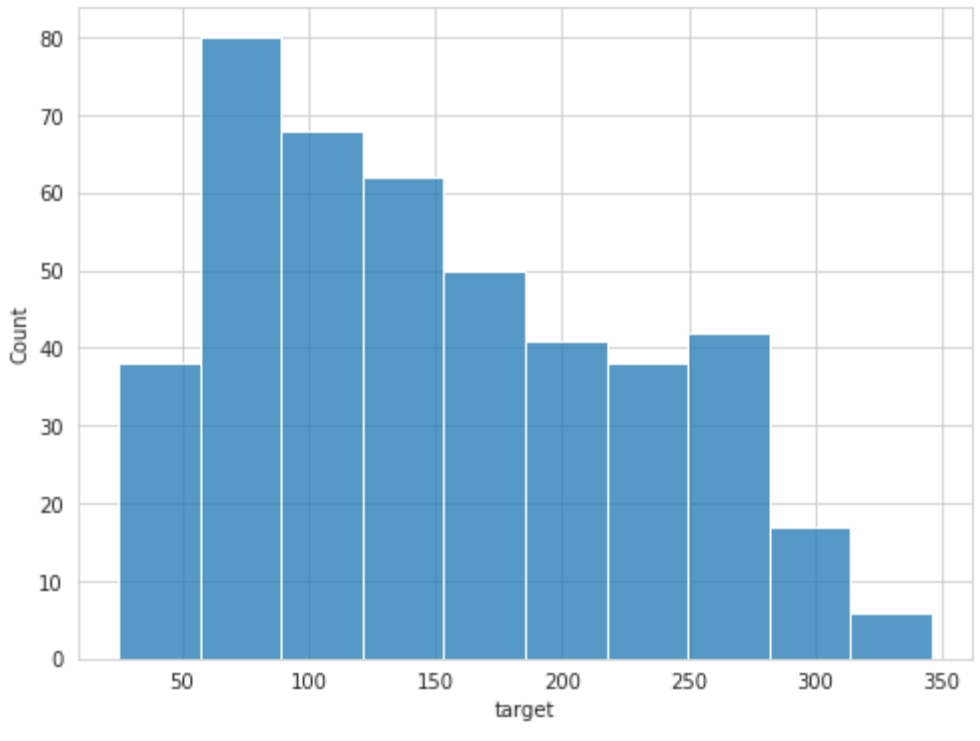
In [61]:
```
plt.rcParams['figure.figsize'] = (8,6)
sns.heatmap(X.corr(), annot=True, fmt='.2f');
```



tc, hdl, ldl, tch сильно коррелированы друг с другом, так как обозначают схожые параметры

Распределение целевой переменной

In [62]:
```
sns.histplot(y);
```



В описании данных написано:

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of n_samples (i.e. the sum of squares of each column totals 1).

Поэтому стандартизация уже не нужна

НО! Я все равно сделаю, чтобы потренироваться в пайплайнах))

Пусть k=5

In [63]:
```
knn_pipeline = Pipeline(steps=[
    ("scaler", StandardScaler()),
    ("knn", KNeighborsRegressor(n_neighbors=5))
])
```

2.2. Регрессионная модель (1 балл)

Создайте модель `KNeighborsRegressor`, обучите ее на тренировочных данных и сделайте предсказания.

In [64]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [65]:
```
# knn_regressor = KNeighborsRegressor(n_neighbors=5);
# knn_regressor.fit(X_train, y_train);
knn_pipeline.fit(X_train, y_train);
```

In [66]:
```
y_pred = knn_pipeline.predict(X_test)
# y_pred = knn_regressor.predict(X_test)
```

2.3. Метрики регресии (3 балла)

Реализуйте метрики R^2, MSE и MAE в `metrics.py`. Примените их для оценки качества полученной модели. Все ли хорошо?

Напомню, что:

$$R^2 = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \bar{y})^2}$$

$$MSE = \frac{1}{n}\sum_i^n (y_i - \hat{y}_i)^2$$

$$MAE = \frac{1}{n}\sum_i^n |y_i - \hat{y}_i|$$

Для k = 5

In [67]:
```
r_squared(y_pred, y_test)
```

Out[67]:
```
0.4628811790427807
```

In [68]:
```
r2_score(y_test, y_pred)
```

Out[68]:
```
0.4628811790427807
```

In [69]:
```
mse(y_pred, y_test)
```

Out[69]:
```
3776.9114606741573
```

In [70]:
```
mean_squared_error(y_test, y_pred)
```

Out[70]: 3776.9114606741573

In [71]:
```
mae(y_pred, y_test)
```

Out[71]: 48.997752808988764

In [72]:
```
mean_absolute_error(y_test, y_pred)
```

Out[72]: 48.997752808988764

2.4. Подбор оптимального числа соседей (2 балла)

Мы почти дошли до конца. Теперь осталось при помощи реализованных нами метрик выбрать лучшее количество соседей для нашей модели.

!!! Обратите внимание на то, что значат наши метрики, для некоторых хорошо, когда они уменьшаются, для других наоборот.

In [73]:
```
from metrics import r_squared, mse, mae
```

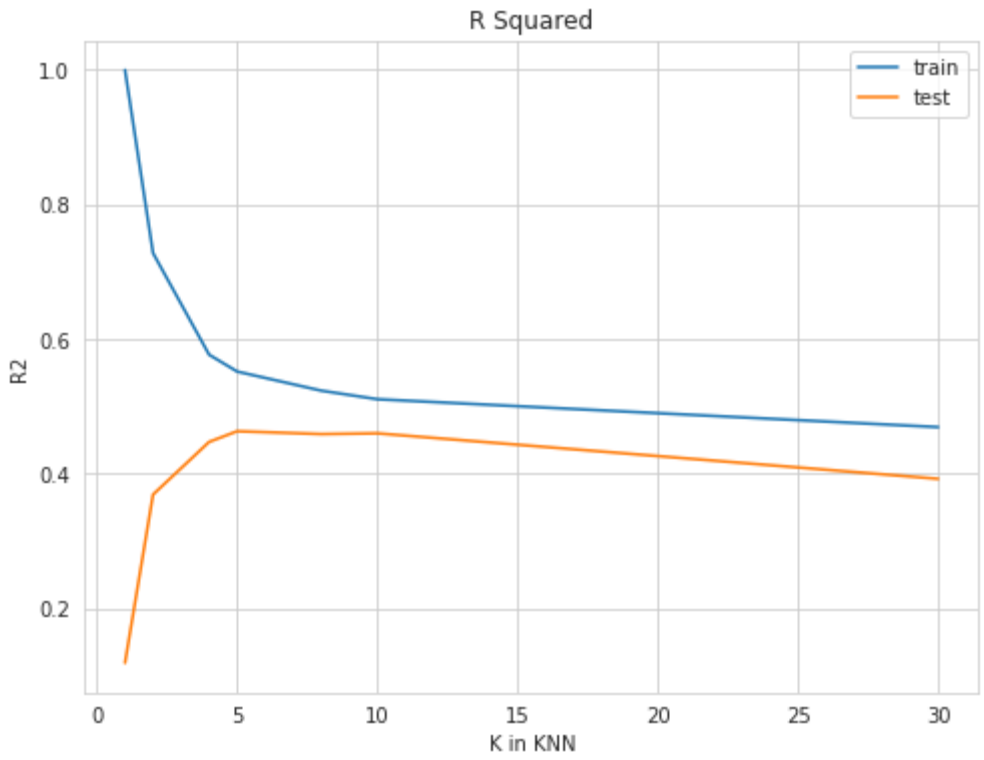
In [74]:
```
def find_best_k(X_train, y_train, X_test, y_test, params, r2=r_squared, mae=mae, mse=mse):
    """
    Choose the best k for KNeighborsRegressor

    Arguments:
    X_train, np array (num_train_samples, num_features) - train data
    y_train, np array (num_train_samples) - train labels
    X_test, np array (num_test_samples, num_features) - test data
    y_test, np array (num_test_samples) - test labels
    params, list of hyperparameters for KNN, here it is list of k values
    r2, function calculates the R squared value, r2(y_pred, y_test)
    mse, function calculates the Mean squared error value, mse(y_pred, y_test)
    mae, function calculates the Mean absolute error value, mae(y_pred, y_test)

    Returns:
    None
    """
    train_r2, test_r2, train_mse, test_mse, train_mae, test_mae = [], [], [], [], [], []

    for k in params:
        knn_pipeline = Pipeline(steps=[("scaler", StandardScaler()),
                                       ("knn", KNeighborsRegressor(n_neighbors=k))])

        knn_pipeline.fit(X_train, y_train)

        y_pred_train = knn_pipeline.predict(X_train)
        y_pred_test = knn_pipeline.predict(X_test)

        r_sq_train = r2(y_pred_train, y_train)
        r_sq_test = r2(y_pred_test, y_test)

        mse_train = mse(y_pred_train, y_train)
        mse_test = mse(y_pred_test, y_test)

        mae_train = mae(y_pred_train, y_train)
        mae_test = mae(y_pred_test, y_test)

        print(
            f'''
            <------- for k = {k} ------>:

            R2 for train data = {r_sq_train}
            R2 for test data = {r_sq_test}

            MSE for train data = {mse_train}
            MSE for test data = {mse_test}

            MAE for train data = {mae_train}
            MAE for test data = {mae_test}'''
             )

        train_r2.append(r_sq_train)
        test_r2.append(r_sq_test)

        train_mse.append(mse_train)
        test_mse.append(mse_test)

        train_mae.append(mae_train)
        test_mae.append(mae_test)

    return train_r2, test_r2, train_mse, test_mse, train_mae, test_mae
```

Для поиска лучшего k вы можете воспользоваться функцией `find_best_k`, которую вы реализовали выше.

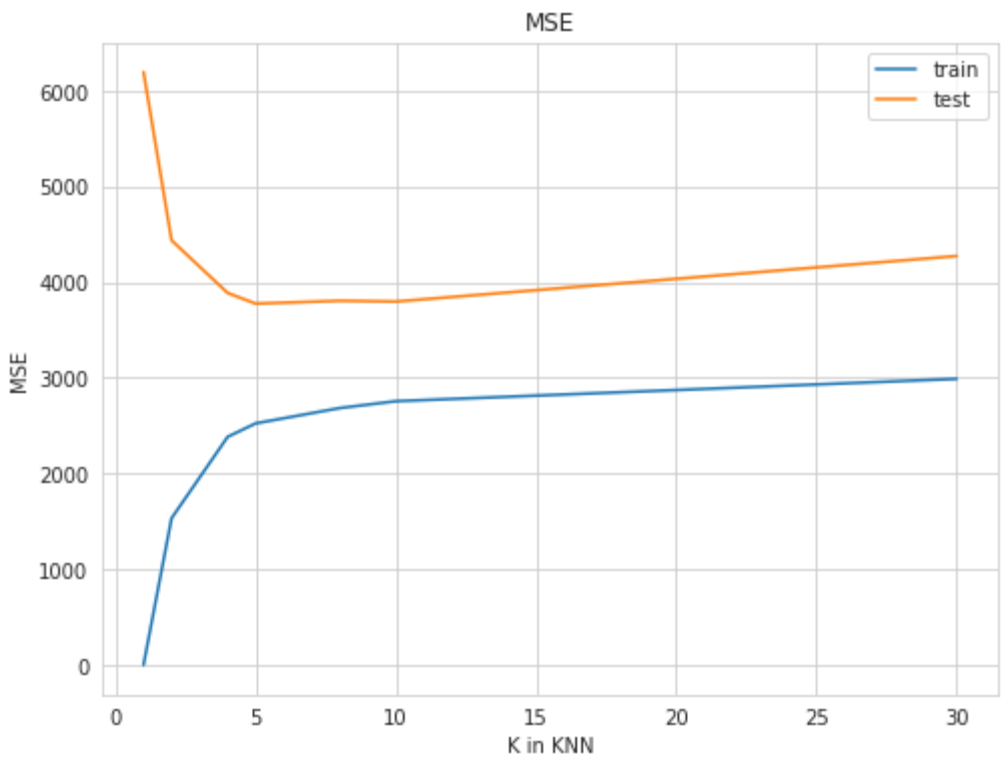
In [75]:
```
params = [1, 2, 4, 5, 8, 10, 30]
train_r2, test_r2, train_mse, test_mse, train_mae, test_mae = find_best_k(X_train, y_train, X_test, y_test, params)
```

```
<------- for k = 1 ------>:

R2 for train data = 1.0
R2 for test data = 0.11841567077886195

MSE for train data = 0.0
MSE for test data = 6199.123595505618

MAE for train data = 0.0
MAE for test data = 58.247191011235955

<------- for k = 2 ------>:

R2 for train data = 0.7277141388907948
R2 for test data = 0.3687555833709971

MSE for train data = 1533.520538243626
MSE for test data = 4438.783707865168

MAE for train data = 29.491501416430594
MAE for test data = 49.97191011235955

<------- for k = 4 ------>:

R2 for train data = 0.5766528842402701
R2 for test data = 0.4469956692843974

MSE for train data = 2384.301168555241
MSE for test data = 3888.6151685393256

MAE for train data = 37.57577903682719
MAE for test data = 48.89887640449438

<------- for k = 5 ------>:

R2 for train data = 0.5516096095483172
R2 for test data = 0.4628811790427807

MSE for train data = 2525.3454957507083
MSE for test data = 3776.9114606741573

MAE for train data = 39.563739376770535
MAE for test data = 48.997752808988764

<------- for k = 8 ------>:

R2 for train data = 0.5231923051222986
R2 for test data = 0.45866432163588644

MSE for train data = 2685.3924398016998
MSE for test data = 3806.5635533707864

MAE for train data = 40.830736543909346
MAE for test data = 49.94101123595506

<------- for k = 10 ------>:

R2 for train data = 0.5104609529465775
R2 for test data = 0.45975930933474096

MSE for train data = 2757.0957223796036
MSE for test data = 3798.8638202247193

MAE for train data = 41.54249291784703
MAE for test data = 50.59775280898876

<------- for k = 30 ------>:

R2 for train data = 0.4690172615539585
R2 for test data = 0.39211302177796503

MSE for train data = 2990.507592067989
MSE for test data = 4274.538901373284

MAE for train data = 44.446836638338056
MAE for test data = 55.09213483146067
```

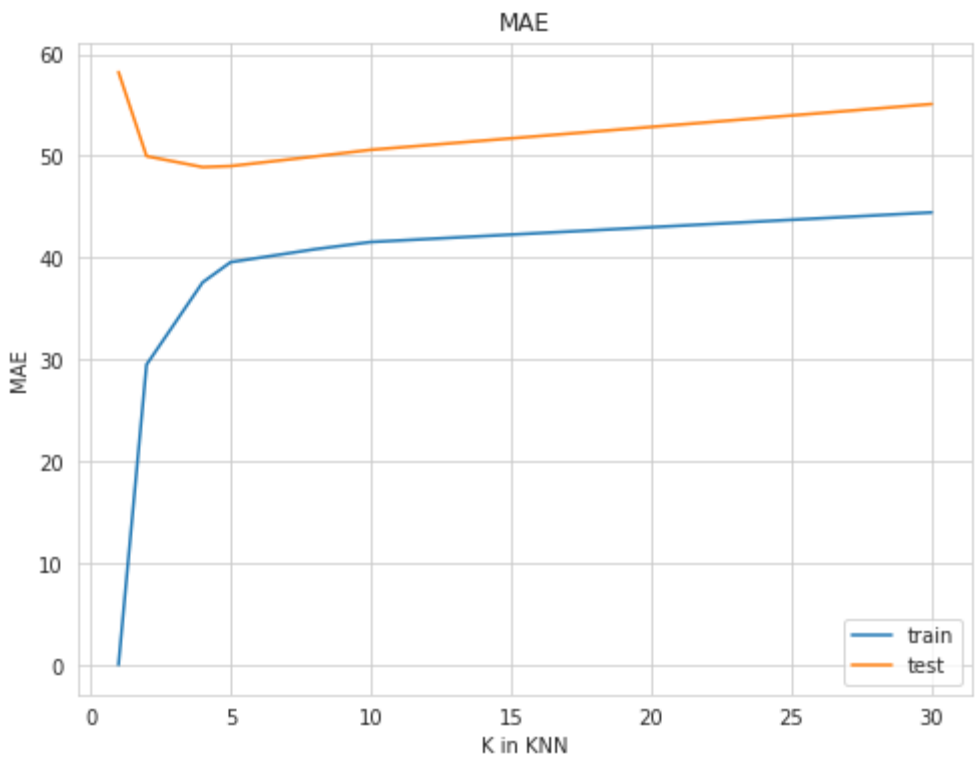
In [76]:
```
plt.plot(params, train_r2, label="train")
plt.plot(params, test_r2, label="test")
plt.legend()
plt.xlabel("K in KNN")
plt.ylabel("R2");
plt.title('R Squared');
```



In [77]:
```
plt.plot(params, train_mse, label="train")
plt.plot(params, test_mse, label="test")
plt.legend()
plt.xlabel("K in KNN")
plt.ylabel("MSE");
plt.title('MSE');
```



In [78]:
```
plt.plot(params, train_mae, label="train")
plt.plot(params, test_mae, label="test")
plt.legend()
plt.xlabel("K in KNN")
plt.ylabel("MAE");
plt.title('MAE');
```



Кажется, значение k = 10 будет вполне приемлимо

3. Социализация (0.5 доп. балла)

Так как у нас теперь большая группа, то было бы здорово всем познакомиться получше (так как выпускной не за горами). Соберитесь с одногруппниками в зуме (желательно, чтобы были люди и с Онлайна, и с Питера), познакомьтесь, а сюда прикрепите скриншот с камерами всех участников.

Therapy time

Напишите здесь ваши впечатления о задании: было ли интересно, было ли слишком легко или наоборот сложно и тд. Также сюда можно написать свои идеи по улучшению заданий, а также предложить данные, на основе которых вы бы хотели построить следующие дз.

Ваши мысли:

Спасибо, что отвечаете на вопросы!