Шанаурина Е.Г. ИУ5Ц-84Б

25 + 4 = 29 вариант РК-1

Номер задачи – 4, номер набора данных – 5.

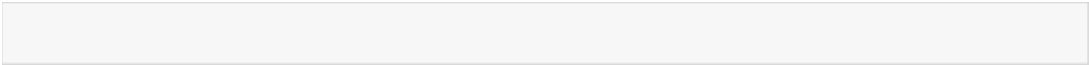
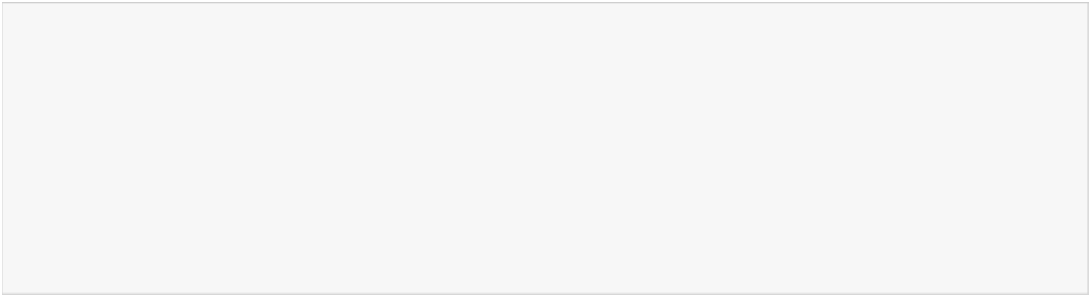
Для студентов группы ИУ5-64Б, ИУ5Ц-84Б - для произвольной колонки данных построить график "Скрипичная диаграмма (violin plot)".

Задача №4.

Для заданного набора данных постройте основные графики, входящие в этап разведочного анализа данных. В случае наличия пропусков в данных удалите строки или колонки, содержащие пропуски. Какие графики Вы построили и почему? Какие выводы о наборе данных Вы можете сделать на основании построенных графиков?

Используемый набор данных: Heart Disease Dataset | Kaggle

# Подгружаем необходимые библиотеки и датасет:



In [1]:

#Подключение библиотек import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV from sklearn.neighbors import KNeighborsRegressor

from sklearn.preprocessing import MinMaxScaler, StandardScaler from matplotlib import pyplot as plt

import seaborn as sns

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score from warnings import simplefilter

simplefilter('ignore')

In [2]:

## #Подгружаем DataSet

heart\_dataset = pd.read\_csv('heart.csv', encoding='latin-1')

In [3]:

heart\_dataset Out[3]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **age** | **sex** | **cp** | **trestbps** | **chol** | **fbs** | **restecg** | **thalach** | **exang** | **oldpeak** | **slope** | **ca** | **thal** | **target** |
| **0** | 52 | 1 | 0 | 125 | 212 | 0 | 1 | 168 | 0 | 1.0 | 2 | 2 | 3 | 0 |
| **1** | 53 | 1 | 0 | 140 | 203 | 1 | 0 | 155 | 1 | 3.1 | 0 | 0 | 3 | 0 |
| **2** | 70 | 1 | 0 | 145 | 174 | 0 | 1 | 125 | 1 | 2.6 | 0 | 0 | 3 | 0 |
| **3** | 61 | 1 | 0 | 148 | 203 | 0 | 1 | 161 | 0 | 0.0 | 2 | 1 | 3 | 0 |
| **4** | 62 | 0 | 0 | 138 | 294 | 1 | 1 | 106 | 0 | 1.9 | 1 | 3 | 2 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1020** | 59 | 1 | 1 | 140 | 221 | 0 | 1 | 164 | 1 | 0.0 | 2 | 0 | 2 | 1 |
| **1021** | 60 | 1 | 0 | 125 | 258 | 0 | 0 | 141 | 1 | 2.8 | 1 | 1 | 3 | 0 |
| **1022** | 47 | 1 | 0 | 110 | 275 | 0 | 0 | 118 | 1 | 1.0 | 1 | 1 | 2 | 0 |
| **1023** | 50 | 0 | 0 | 110 | 254 | 0 | 0 | 159 | 0 | 0.0 | 2 | 0 | 2 | 1 |
| **1024** | 54 | 1 | 0 | 120 | 188 | 0 | 1 | 113 | 0 | 1.4 | 1 | 1 | 3 | 0 |

1025 rows × 14 columns

# Выводим информацию о столбцах датасета:

In [4]:

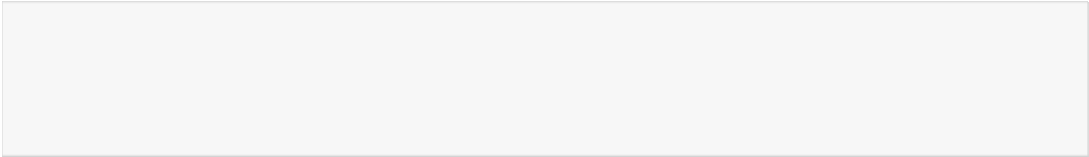
heart\_dataset.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 1025 entries, 0 to 1024 Data columns (total 14 columns):

# Column Non-Null Count Dtype

1. age 1025 non-null int64
2. sex 1025 non-null int64
3. cp 1025 non-null int64
4. trestbps 1025 non-null int64
5. chol 1025 non-null int64
6. fbs 1025 non-null int64
7. restecg 1025 non-null int64
8. thalach 1025 non-null int64
9. exang 1025 non-null int64
10. oldpeak 1025 non-null float64
11. slope 1025 non-null int64
12. ca 1025 non-null int64
13. thal 1025 non-null int64
14. target 1025 non-null int64 dtypes: float64(1), int64(13) memory usage: 112.2 KB

In [6]:



#категоральный признаков в датасете слишком много, они будут мешать делать масштабировани е данных.

#Оставим только те столбцы, с которопыми потом будем работать

heart\_dataset\_new = heart\_dataset[['sex', 'age', 'trestbps', "chol", "thalach", "oldpeak ", "slope", "thal"]].copy()

heart\_dataset\_new

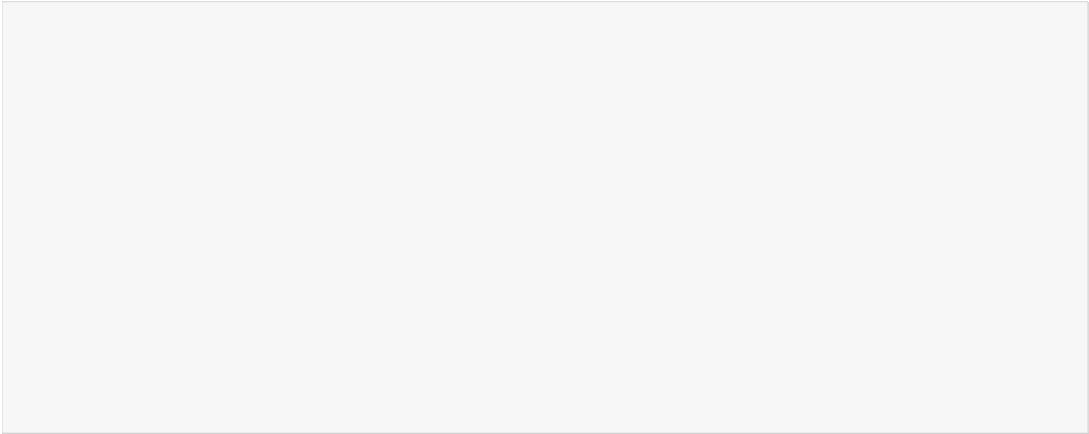
Out[6]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **sex** | **age** | **trestbps** | **chol** | **thalach** | **oldpeak** | **slope** | **thal** |
| **0** | 1 | 52 | 125 | 212 | 168 | 1.0 | 2 | 3 |
| **1** | 1 | 53 | 140 | 203 | 155 | 3.1 | 0 | 3 |
| **2** | 1 | 70 | 145 | 174 | 125 | 2.6 | 0 | 3 |
| **3** | 1 | 61 | 148 | 203 | 161 | 0.0 | 2 | 3 |
| **4** | 0 | 62 | 138 | 294 | 106 | 1.9 | 1 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **1020** | 1 | 59 | 140 | 221 | 164 | 0.0 | 2 | 2 |
| **1021** | 1 | 60 | 125 | 258 | 141 | 2.8 | 1 | 3 |
| **1022** | 1 | 47 | 110 | 275 | 118 | 1.0 | 1 | 2 |
| **1023** | 0 | 50 | 110 | 254 | 159 | 0.0 | 2 | 2 |
| **1024** | 1 | 54 | 120 | 188 | 113 | 1.4 | 1 | 3 |

1025 rows × 8 columns

# Кодируем категориальные признаки

In [7]:



#Кодирование категориальных признаков

heart\_dataset\_new["trestbps"].value\_counts()

heart\_dataset\_new["trestbps"] = heart\_dataset\_new["trestbps"].astype('category')

heart\_dataset\_new["chol"] = heart\_dataset\_new["chol"].astype('category') heart\_dataset\_new["thalach"] = heart\_dataset\_new["thalach"].astype('category') heart\_dataset\_new["oldpeak"] = heart\_dataset\_new["oldpeak"].astype('category')

#Назначить закодированную переменную новосу столбцу с помощью метода доступа heart\_dataset\_new["trestbps\_cat"] = heart\_dataset\_new["trestbps"].cat.codes heart\_dataset\_new["chol\_cat"] = heart\_dataset\_new["chol"].cat.codes heart\_dataset\_new["thalach\_cat"] = heart\_dataset\_new["thalach"].cat.codes heart\_dataset\_new["oldpeak\_cat"] = heart\_dataset\_new["oldpeak"].cat.codes

heart\_dataset\_cat = heart\_dataset\_new.drop(['trestbps', 'chol', 'thalach', 'oldpeak'], a xis=1, inplace=True)

heart\_dataset\_new

Out[7]:

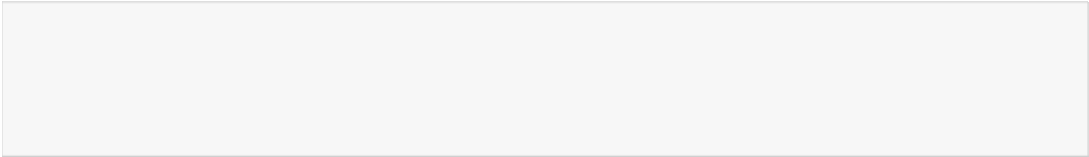


|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **sex** | **age** | **slope** | **thal** | **trestbps\_cat** | **chol\_cat** | **thalach\_cat** | **oldpeak\_cat** |
| **0** | 1 | 52 | 2 | 3 | 18 | 43 | 67 | 10 |
| **1** | 1 | 53 | 0 | 3 | 28 | 34 | 54 | 29 |
| **2** | 1 | 70 | 0 | 3 | 31 | 12 | 25 | 25 |
| **3** | 1 | 61 | 2 | 3 | 33 | 34 | 60 | 0 |
| **4** | 0 | 62 | 1 | 2 | 27 | 116 | 9 | 18 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **1020** | 1 | 59 | 2 | 2 | 28 | 52 | 63 | 0 |
| **1021** | 1 | 60 | 1 | 3 | 18 | 87 | 40 | 26 |
| **1022** | 1 | 47 | 1 | 2 | 8 | 103 | 19 | 10 |
| **1023** | 0 | 50 | 2 | 2 | 8 | 83 | 58 | 0 |
| **1024** | 1 | 54 | 1 | 3 | 14 | 24 | 14 | 14 |

1025 rows × 8 columns

# Разделяем выборки

In [8]:



#разделение выборки

from sklearn.model\_selection import train\_test\_split y = heart\_dataset\_new['age']

X = heart\_dataset\_new.drop('age', axis=1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=3) x\_train

Out[8]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **sex** | **slope** | **thal** | **trestbps\_cat** | **chol\_cat** | **thalach\_cat** | **oldpeak\_cat** |
| **730** | 1 | 2 | 2 | 14 | 51 | 69 | 0 |
| **549** | 1 | 2 | 3 | 13 | 105 | 50 | 10 |
| **518** | 1 | 1 | 3 | 14 | 24 | 38 | 19 |
| **617** | 0 | 2 | 2 | 19 | 125 | 62 | 0 |
| **261** | 1 | 2 | 3 | 34 | 63 | 64 | 16 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **659** | 1 | 2 | 2 | 27 | 100 | 80 | 0 |
| **789** | 1 | 1 | 3 | 14 | 107 | 7 | 14 |
| **256** | 0 | 2 | 2 | 27 | 19 | 80 | 14 |
| **968** | 1 | 0 | 3 | 28 | 34 | 54 | 29 |
| **952** | 0 | 2 | 2 | 25 | 123 | 69 | 0 |

717 rows × 7 columns

In [9]:



y\_train

Out[9]:

|  |  |
| --- | --- |
| 730 | 44 |
| 549 | 68 |
| 518 | 49 |
| 617 | 41 |
| 261 | 54 |
|  | .. |
| 659 | 59 |

789 62

256 35

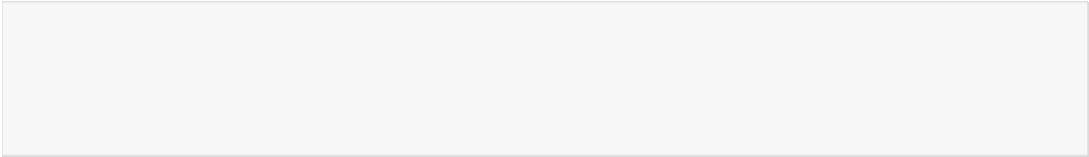
968 53

952 54

Name: age, Length: 717, dtype: int64

# Масштабирование данных

In [10]:



#Масштбирование данных

from sklearn.preprocessing import MinMaxScaler scaler = MinMaxScaler().fit(x\_train)

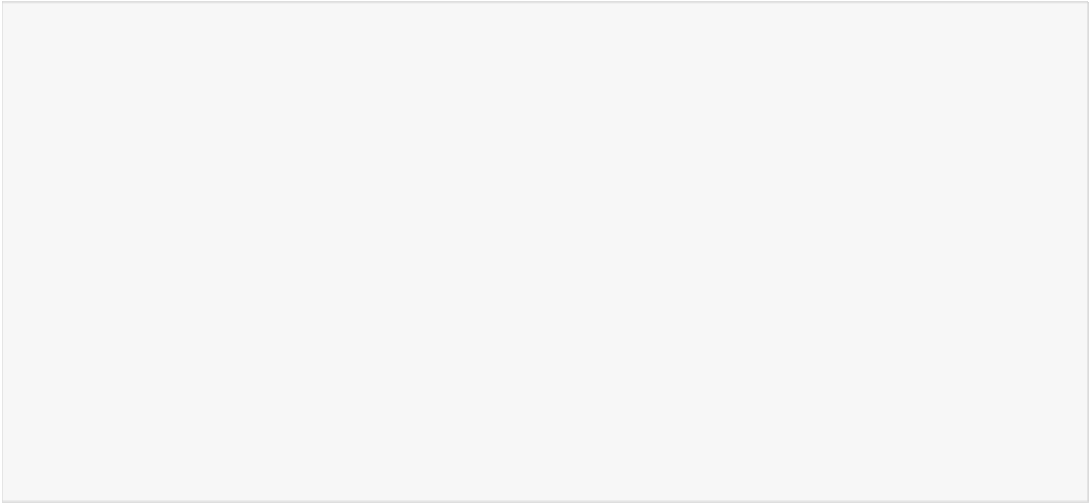
x\_train = pd.DataFrame(scaler.transform(x\_train), columns = x\_train.columns) x\_test = pd.DataFrame(scaler.transform(x\_test), columns = x\_train.columns) x\_train.describe()

Out[10]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **sex** | **slope** | **thal** | **trestbps\_cat** | **chol\_cat** | **thalach\_cat** | **oldpeak\_cat** |
| **count** | 717.000000 | 717.000000 | 717.000000 | 717.000000 | 717.000000 | 717.000000 | 717.000000 |
| **mean** | 0.695955 | 0.682706 | 0.776383 | 0.453278 | 0.478205 | 0.540586 | 0.263777 |
| **std** | 0.460323 | 0.312821 | 0.208569 | 0.218063 | 0.260037 | 0.235252 | 0.268882 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 0.000000 | 0.500000 | 0.666667 | 0.291667 | 0.260000 | 0.366667 | 0.000000 |
| **50%** | 1.000000 | 0.500000 | 0.666667 | 0.458333 | 0.453333 | 0.566667 | 0.205128 |
| **75%** | 1.000000 | 1.000000 | 1.000000 | 0.583333 | 0.686667 | 0.722222 | 0.435897 |
| **max** | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

# Обучение KNN с производным k

In [11]:



#Обучение KNN с производным k

simplefilter('ignore')

def print\_metrics(y\_test, y\_pred): print(f"R^2: {r2\_score(y\_test, y\_pred)}")

print(f"MSE: {mean\_squared\_error(y\_test, y\_pred)}") print(f"MAE: {mean\_absolute\_error(y\_test, y\_pred)}")

def print\_cv\_result(cv\_model, x\_test, y\_test):

print(f'Оптимизация метрики {cv\_model.scoring}: {cv\_model.best\_score\_}') print(f'Лучший параметр: {cv\_model.best\_params\_}')

print('Метрики на тестовом наборе') print\_metrics(y\_test, cv\_model.predict(x\_test)) print()

base\_k = 7

base\_knn = KNeighborsRegressor(n\_neighbors=base\_k) base\_knn.fit(x\_train, y\_train)

y\_pred\_base = base\_knn.predict(x\_test) print(f'Test metrics for KNN with k={base\_k}\n') print\_metrics(y\_test, y\_pred\_base)

Test metrics for KNN with k=7 R^2: 0.4400118414555648

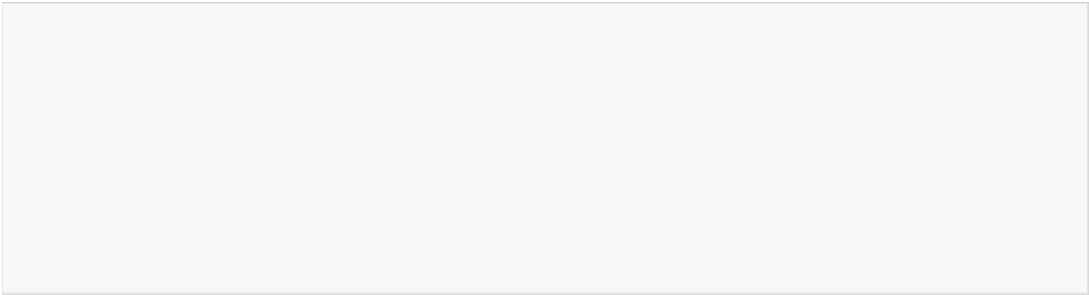
MSE: 44.78028094354626

MAE: 5.355287569573284

# Кросс-валидация



In [12]:



#Кросс валидация

metrics = ['r2', 'neg\_mean\_squared\_error', 'neg\_mean\_absolute\_error'] cv\_values = [5, 10]

for cv in cv\_values:

print(f'Результаты кросс-валидации при cv={cv}\n') for metric in metrics:

params = {'n\_neighbors': range(1, 30)}

knn\_cv = GridSearchCV(KNeighborsRegressor(), params, cv=cv, scoring=metric, n\_jo

bs=-1)

knn\_cv.fit(x\_train, y\_train) print\_cv\_result(knn\_cv, x\_test, y\_test)

Результаты кросс-валидации при cv=5

Оптимизация метрики r2: 0.8414599182336975 Лучший параметр: {'n\_neighbors': 1} Метрики на тестовом наборе

R^2: 0.9501820473138464

MSE: 3.9837662337662336

MAE: 0.41883116883116883

Оптимизация метрики neg\_mean\_squared\_error: -12.75434149184149 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе R^2: 0.9501820473138464

MSE: 3.9837662337662336

MAE: 0.41883116883116883

Оптимизация метрики neg\_mean\_absolute\_error: -0.9810508935508937 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе R^2: 0.9501820473138464

MSE: 3.9837662337662336

MAE: 0.41883116883116883

Результаты кросс-валидации при cv=10

Оптимизация метрики r2: 0.8947027785180053 Лучший параметр: {'n\_neighbors': 1} Метрики на тестовом наборе

R^2: 0.9501820473138464

MSE: 3.9837662337662336

MAE: 0.41883116883116883

Оптимизация метрики neg\_mean\_squared\_error: -8.46725352112676 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе R^2: 0.9501820473138464

MSE: 3.9837662337662336

MAE: 0.41883116883116883

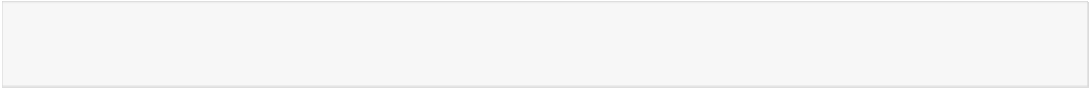
Оптимизация метрики neg\_mean\_absolute\_error: -0.6595070422535211 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе R^2: 0.9501820473138464

MSE: 3.9837662337662336

MAE: 0.41883116883116883

In [13]:



best\_k = 9

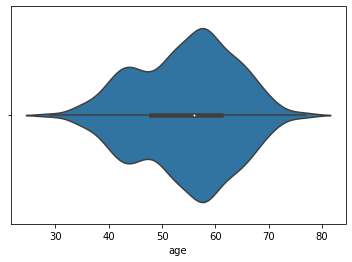
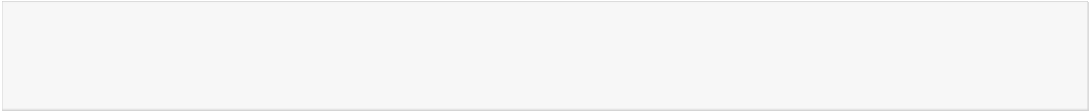
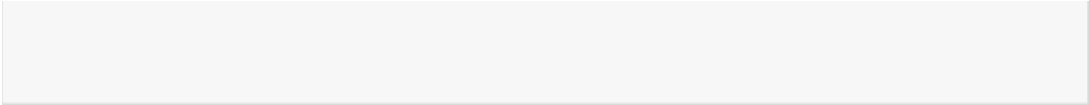
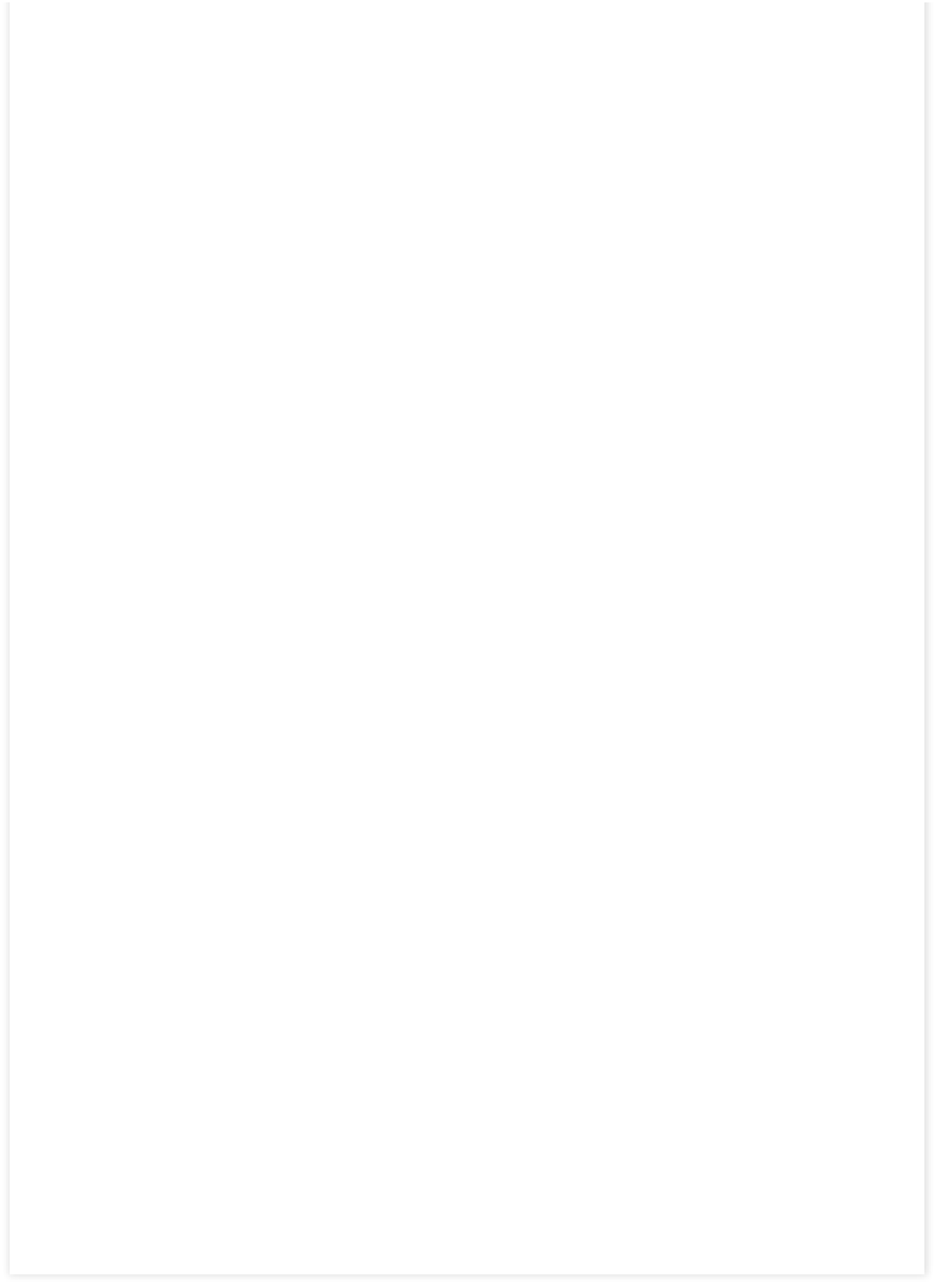
y\_pred\_best = KNeighborsRegressor(n\_neighbors=best\_k).fit(x\_train, y\_train).predict(x\_te st)

In [14]:



#Сравнение исходной и оптимальной модели

print('Исходная модель\n') print\_metrics(y\_test, y\_pred\_base) print('\nОптимальная модель\n') print\_metrics(y\_test, y\_pred\_best)



Исходная модель

R^2: 0.4400118414555648

MSE: 44.78028094354626

MAE: 5.355287569573284

Оптимальная модель

R^2: 0.39001366302838936

MSE: 48.77845919512586

MAE: 5.6273448773448775

In [15]:

## #Визуализация результатов

res = pd.DataFrame({'y\_test': y\_test, 'y\_pred\_best': y\_pred\_best}).sort\_values(by='y\_tes t')

res.head() Out[15]:

|  |  |  |
| --- | --- | --- |
|  | **y\_test** | **y\_pred\_best** |
| **118** | 29 | 44.222222 |
| **64** | 29 | 44.222222 |
| **143** | 34 | 37.555556 |
| **201** | 34 | 37.555556 |
| **846** | 35 | 42.333333 |

# Скрипичная диаграмма по столбцу "age"

In [16]:

sns.violinplot(x=heart\_dataset\_new['age']) Out[16]:

<AxesSubplot:xlabel='age'>

In [ ]: