Любой исследовательский ноутбук должен начинаться с того, что мы устанавливаем необходимые пакеты для дальнейшей работы - не забывайте этого делать при создании ноутбуков для лабораторных! Делается это вот так:

import matplotlib

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

TEXT\_COLOR = 'black'

matplotlib.rcParams['figure.figsize'] = (15, 10)

matplotlib.rcParams['text.color'] = 'black'

matplotlib.rcParams['font.size'] = 14

matplotlib.rcParams['axes.labelcolor'] = TEXT\_COLOR

matplotlib.rcParams['xtick.color'] = TEXT\_COLOR

matplotlib.rcParams['ytick.color'] = TEXT\_COLOR

RANDOM\_STATE = 42

np.random.seed(RANDOM\_STATE)

from sklearn.datasets import load\_boston

boston\_data = load\_boston()

print(type(boston\_data))

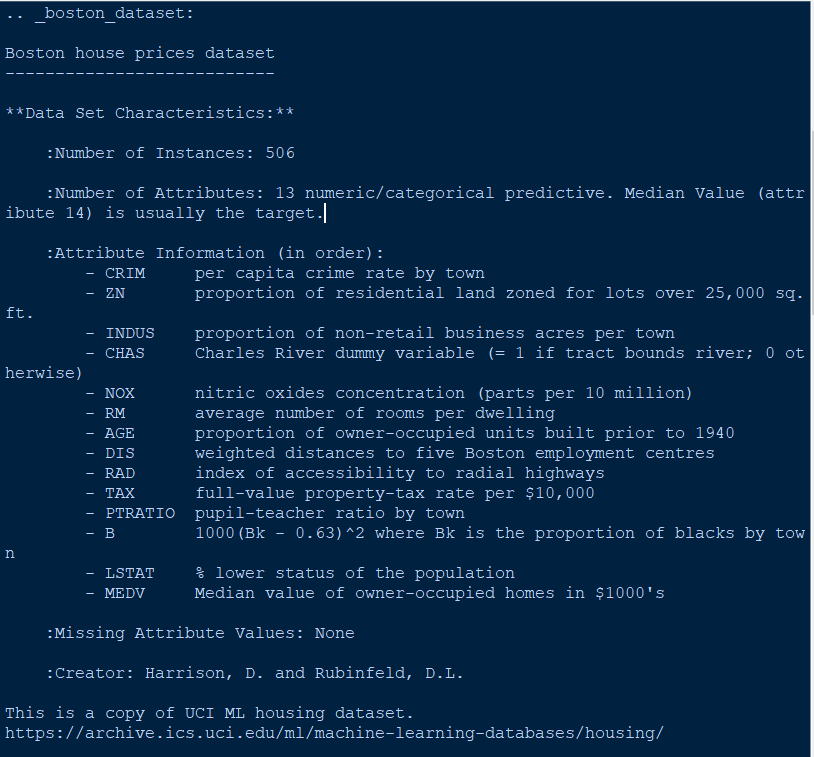
Результат：

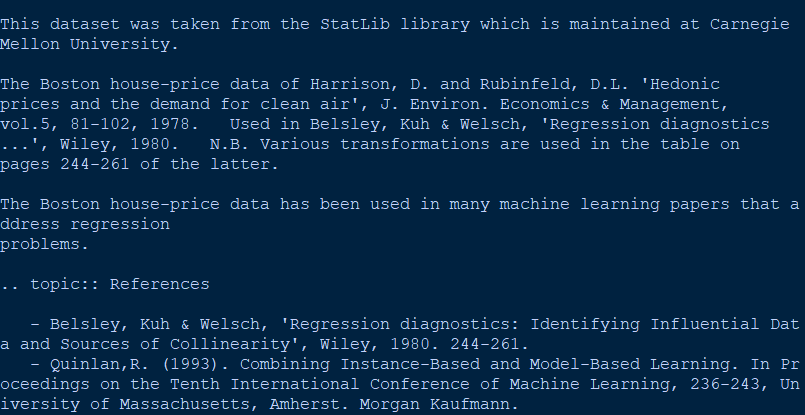


Начнем с получения описания набора данных.

print(boston\_data['DESCR'])

Результат：





Проверим размерности матрицы признаков и вектора предсказываемых значений. Так как они имеет тип ndarray (массив фреймворка numpy), то мы можем воспользоваться членом класса .shape, который является кортежем с элементами в виде размерности массива. Также, проверим тип данных через встроенную функцию Python - type().

feature\_data = boston\_data['data']

feature\_names = boston\_data['feature\_names']

target\_data = boston\_data['target']

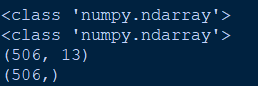
print(type(feature\_data))

print(type(target\_data))

print(feature\_data.shape)

print(target\_data.shape)

Результат：



Отлично, размерности и тип соответствуют описанию. Теперь посмотрим, что содержит последний ключ.

print(feature\_names)

Результат：

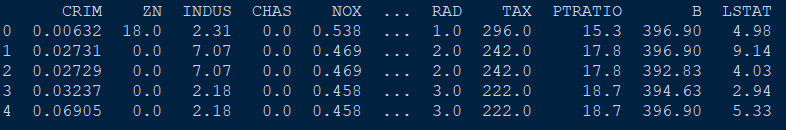


Для дальнейшей работы с данными рекомендуется перевести данные в формат фреймворка pandas, так как данные имеют матричный формат и работа в данном фреймворке позволит удобно организовать анализ.

df = pd.DataFrame(feature\_data, columns=boston\_data['feature\_names'])

print(df.head())

Результат：

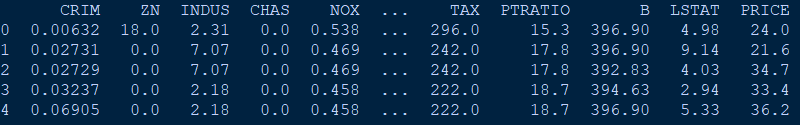


Для дальнейшего анализа включим значения целевых переменных в DataFrame под собственным именем.

df['PRICE'] = target\_data

print(df.head())

Результат：

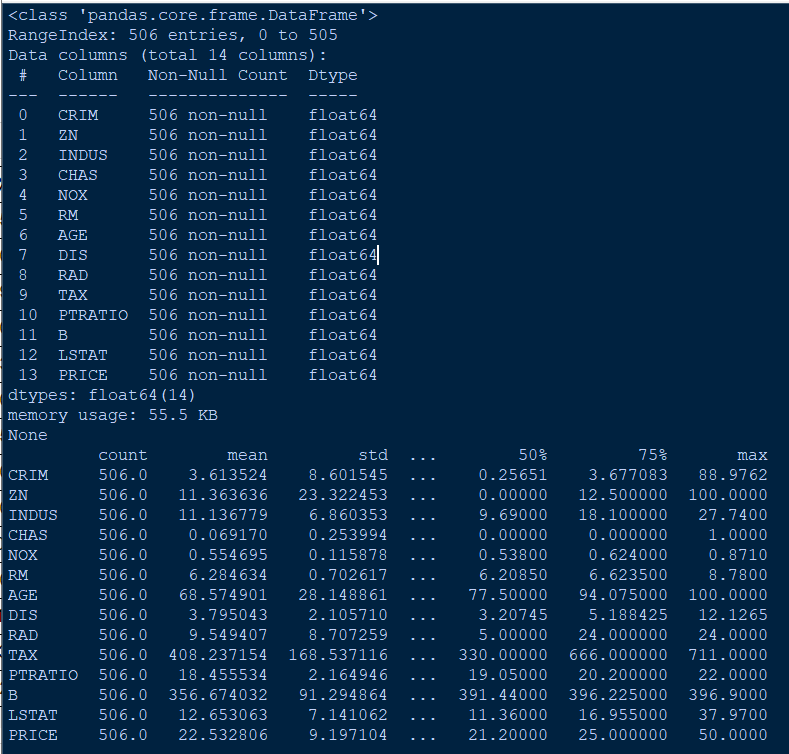


Выведем базовую информацию по данным. С помощью методов info() и describe().

print(df.info())

print(df.describe().T)

Результат：



Одним из важных этапов является проверка данных на наличие одинаковых строк (записей). Поэчему это важно? Ответ прост! При разделении данные на обучение/тест или других делениях дубликаты могут попасть как в одну кучу, так и в другую. Таким образом происходит leak информации, то есть модель обучается на данных, которые есть в тесте. Это очень плохо и такое надо всегда исключать! Проверим наши:

# Данный метод возвращает маску дубликатов

# Если запись - дубликат, то в маске она будет True

# Получить DataFrame без дубликатов можно утем применения инвертированной маски

dup\_mask = df.duplicated()

# А мы просто посмотрим сумму элементов и так убедимся, что дубликатов нет!

print(dup\_mask.sum())

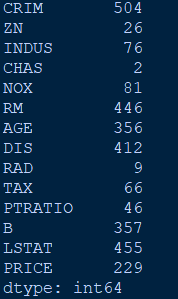
Результат：



Метод df.info() показал, что данные не имеют строк, которые сразу принимаются за категориальные признаки. На данный момент все признаки являются числовыми.

print(df.nunique())

Результат：



Признак CHAS имеет всего два значения - проверим, какие это значения:

print(df['CHAS'].value\_counts())

Результат：



Анализ распределений заключается в том, чтобы просмотреть распределения численных признаков, понять их характер и необходимость коррекции смещения.

Под нормальным (Гауссовым) распределением понимается распределение следующего вида:

df['CHAS'] = df['CHAS'].astype(int).astype(dtype='category')

print(df.info())

plt.figure(figsize=[10, 7])

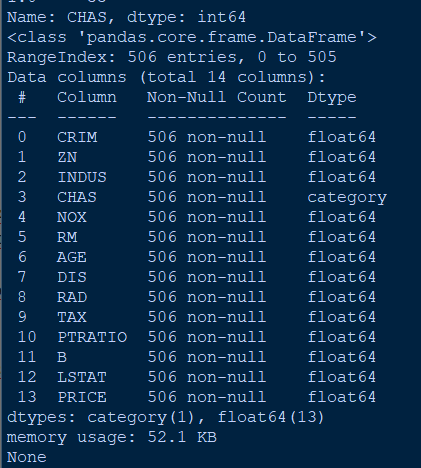
sns.distplot(df['RM'], bins=20)

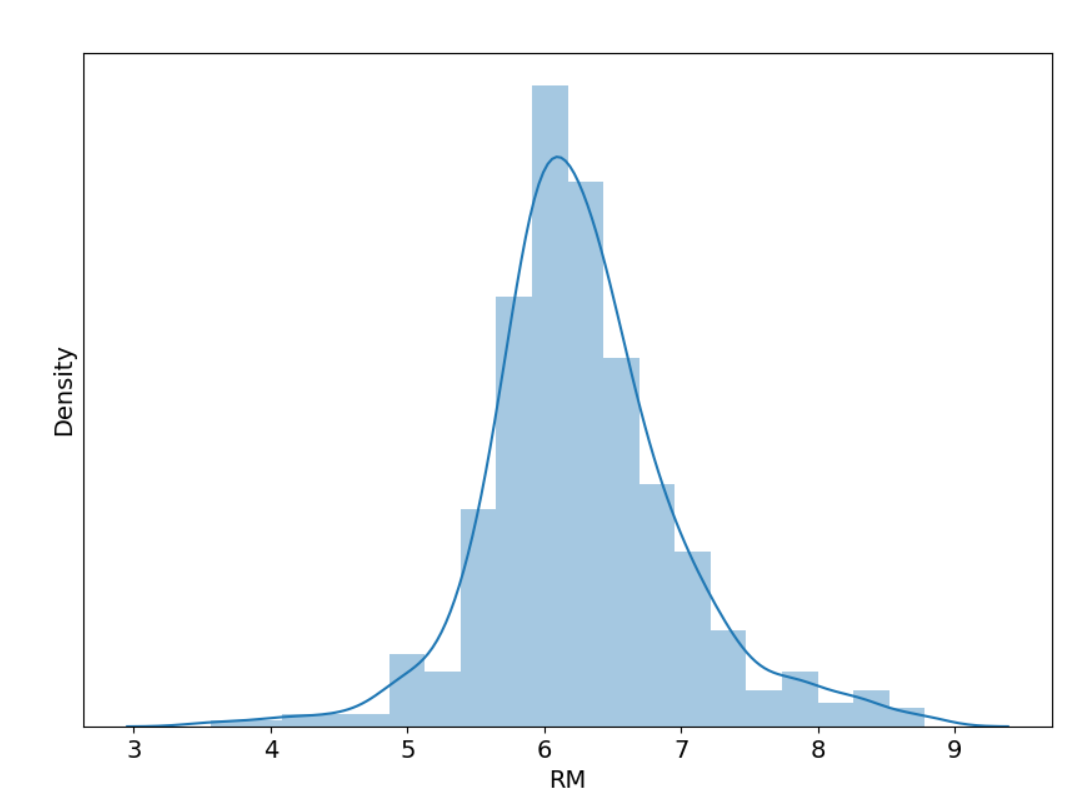
# Отключение меток на оси Y

plt.yticks([])

plt.show()

Результат：





Смещение нормального распределения могут быть как левые (хвост находится справа), так и правое (хвост слева):

\_, ax = plt.subplots(nrows=1, ncols=2, figsize=[20, 7])

sns.distplot(df['DIS'], ax=ax[0], bins=20)

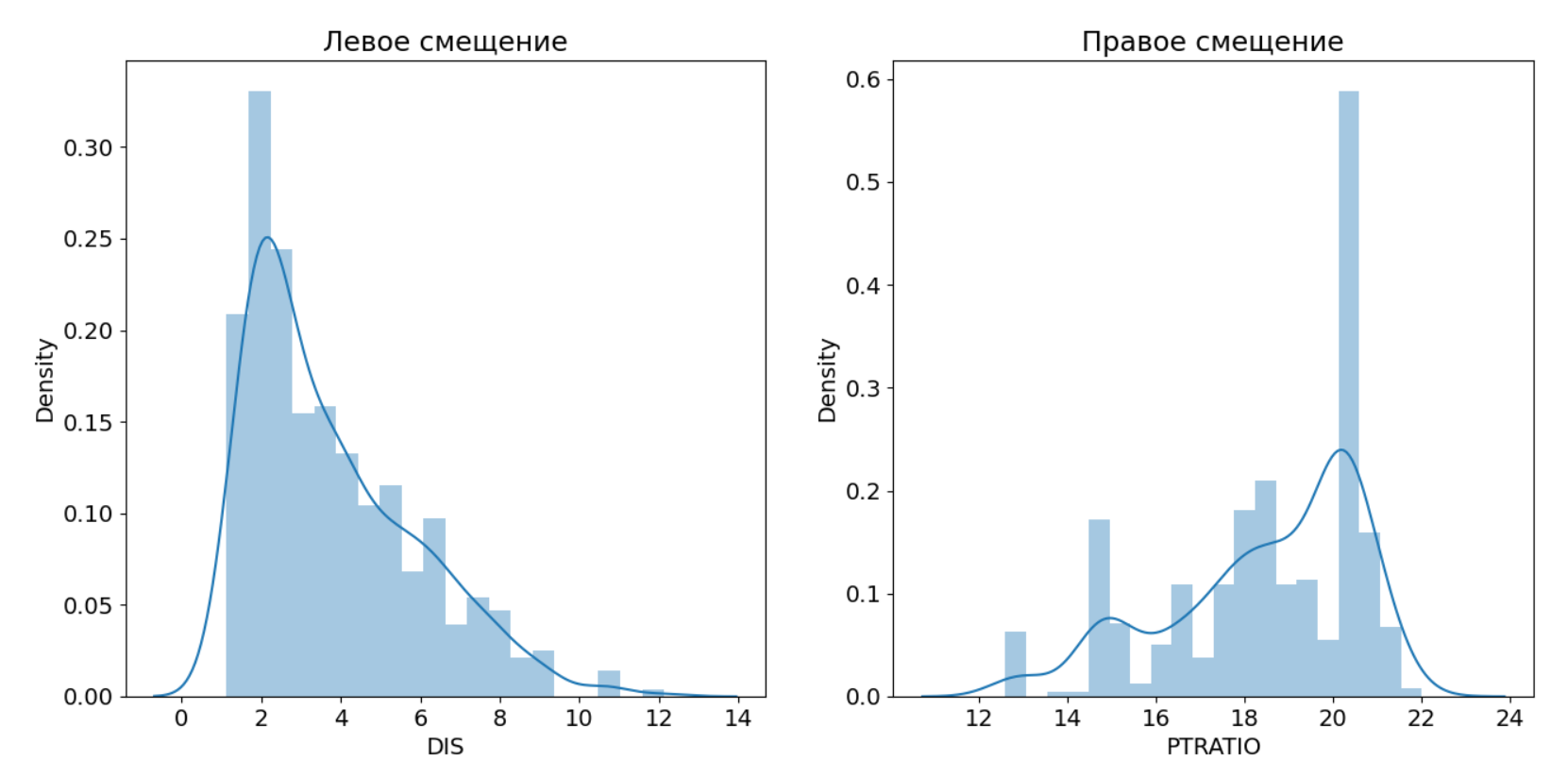
ax[0].title.set\_text('Левое смещение')

sns.distplot(df['PTRATIO'], ax=ax[1], bins=20)

ax[1].title.set\_text('Правое смещение')

plt.show()

Результат：

  
Распределения могут иметь и бимодальный характер:

plt.figure(figsize=[10, 7])

sns.distplot(df['RAD'], bins=20)

# Отключение меток на оси Y

plt.yticks([])

plt.show()

Результат：



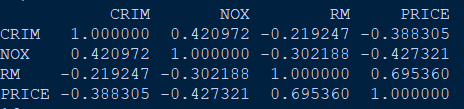
Корреляционный анализ важен для оценки того, как данные взаимосвязаны. В этом поможет функция отображения коэффициентов корреляции DataFrame.corr(). Корреляционный анализ позволит понять, какие переменные имеют связь с целевой переменной, что полезно для выбора признаков для обучения.

features = ['CRIM', 'NOX', 'RM', 'PRICE']

correlation\_mtrx = df[features].corr()

correlation\_mtrx

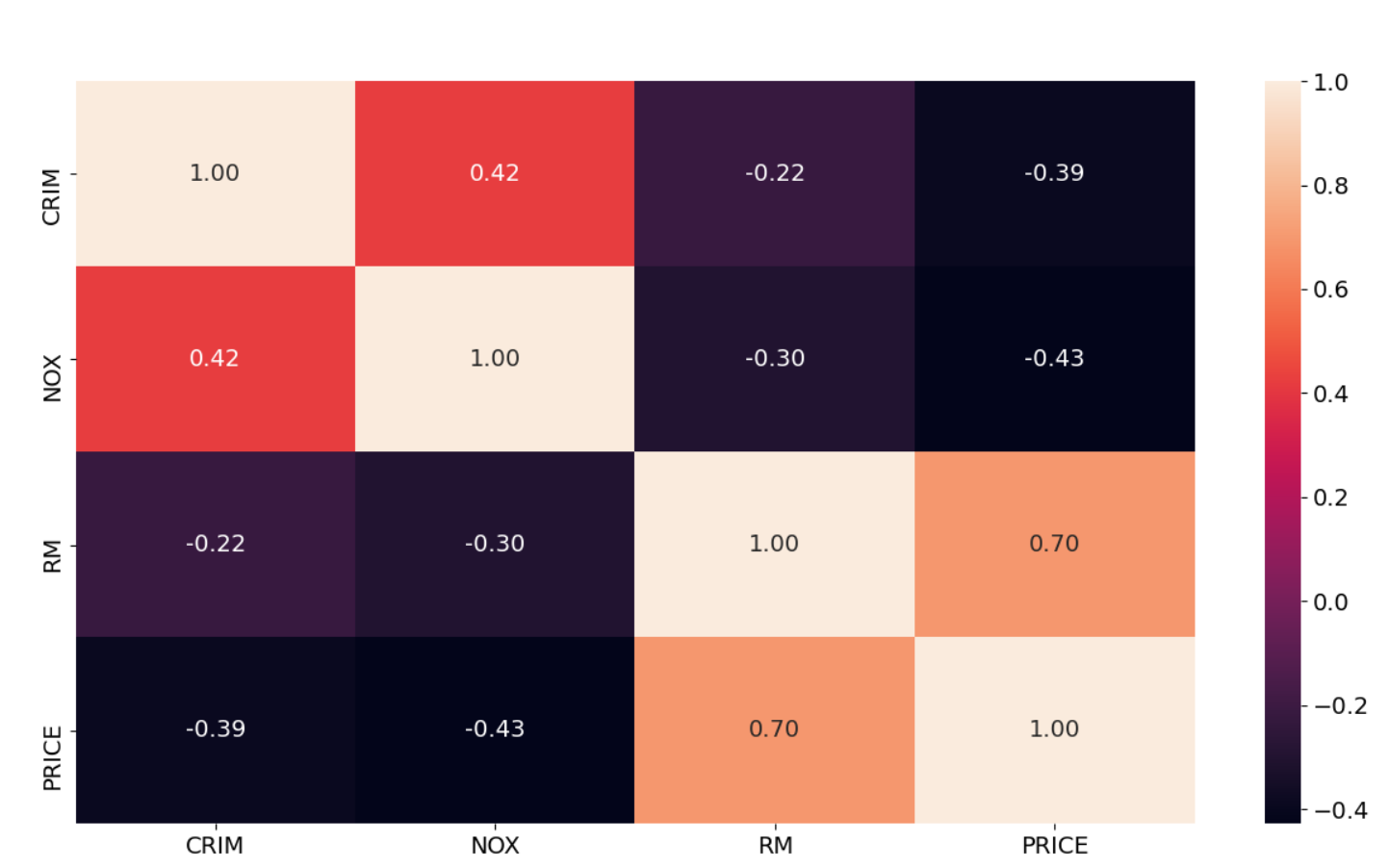
Результат：



При этом отображение численных значений часто сложно для восприятия, поэтому проще отобразить данные в виде представления тепловой карты с помощью функции heatmap().

sns.heatmap(correlation\_mtrx, annot=True, fmt='.2f')

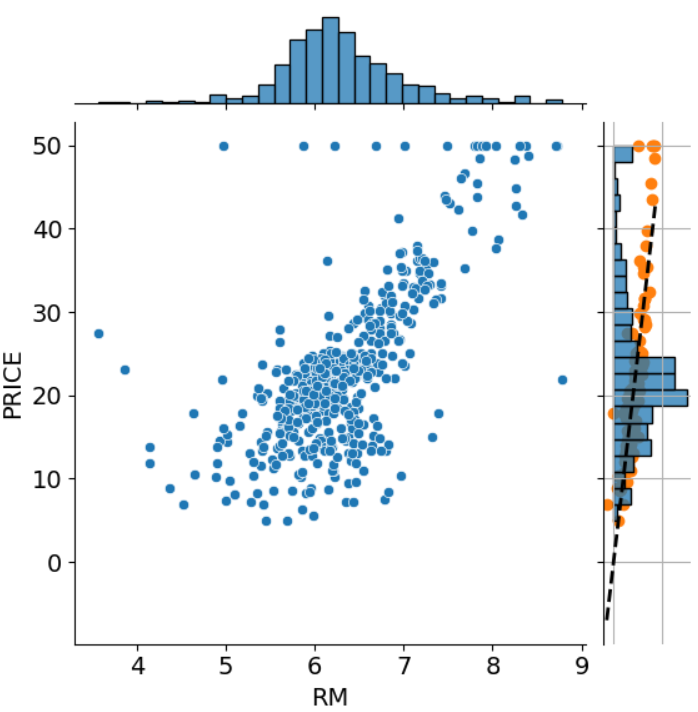
Результат：



После определения признаков, которые имеют высокую корреляцию, важно также посмотреть на распределение данных, так как показатель корреляции не сообщает о характере зависимости или о том, как данные распределены:

sns.jointplot(x="RM", y="PRICE", data=df)

Результат：



Для оценки количества возможных выбросов выберем данные, которые соответсвуют PRICE = 50.

outliers\_count = df[df['PRICE'] == 50].shape[0]

print(outliers\_count)

print(outliers\_count/df.shape[0]\*100)

Результат：



Перед тем, как переходить к модели, требуется сделать разделение обраотанных данных на выборки для обучения и тестирования. Для этого воспользуется функцией train\_test\_split().

from sklearn.model\_selection import train\_test\_split

# 70% - для обучения, 30% - для тестов

TRAIN\_RATIO = 0.7

# X - DataFrame только с признаками (для примера берем все признаки)

X = df[feature\_names]

# y - истинные значения (разметка)

y = df['PRICE']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

# Передаются наборы, которые будут разделены в соответсвии с параметрами

X, y,

# Задается размер обучающей выборки

train\_size=TRAIN\_RATIO,

# Фиксируется состояние генератора случайных чисел

# для сохранения повторяемости

random\_state=RANDOM\_STATE

)

# Отображение размеров выборок

print(X\_train.shape, y\_train.shape)

print(X\_test.shape, y\_test.shape)

Результат：



Начнем с создания модели линейной регрессии и ее обучения с помощью метода fit().

from sklearn.linear\_model import LinearRegression

reg = LinearRegression()

reg.fit(X\_train, y\_train)

def show\_linear\_model\_weights(model, feature\_names=None):

'''

Отображение весов

Аргументы:

model - модель линейной регрессии

feature\_names - (опционально) если передано, отображение весов вместе с именами признаков

'''

print('\n----- Weights -----')

weights = model.coef\_

bias = model.intercept\_

if feature\_names is None:

print(f'Weights: {weights}')

else:

# Функция zip() делает из двух списков один список пар значений

pairs = zip(feature\_names, weights)

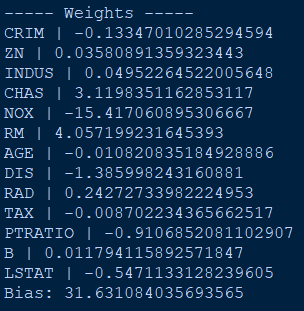
for pair in pairs:

print(f'{pair[0]} | {pair[1]}')

print(f'Bias: {bias}')

show\_linear\_model\_weights(reg, X\_train.columns)

Результат：



Для начала отобразим график предсказаний и посмотрим, как распределяются на линии пресказания. Для получения предсказания модели используется метод predict().

def draw\_predictions\_plot(y\_pred, y\_true):

plt.scatter(y\_pred, y\_true)

line\_pnts = np.linspace(np.amin(y\_pred)-0.1, np.amax(y\_pred)+0.1, 1000)

plt.plot(line\_pnts, line\_pnts, 'k--', lw=2)

plt.xlabel('Predicted')

plt.ylabel('True')

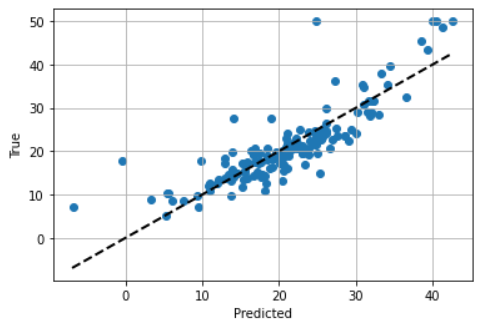
plt.grid()

y\_pred = reg.predict(X\_test)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат



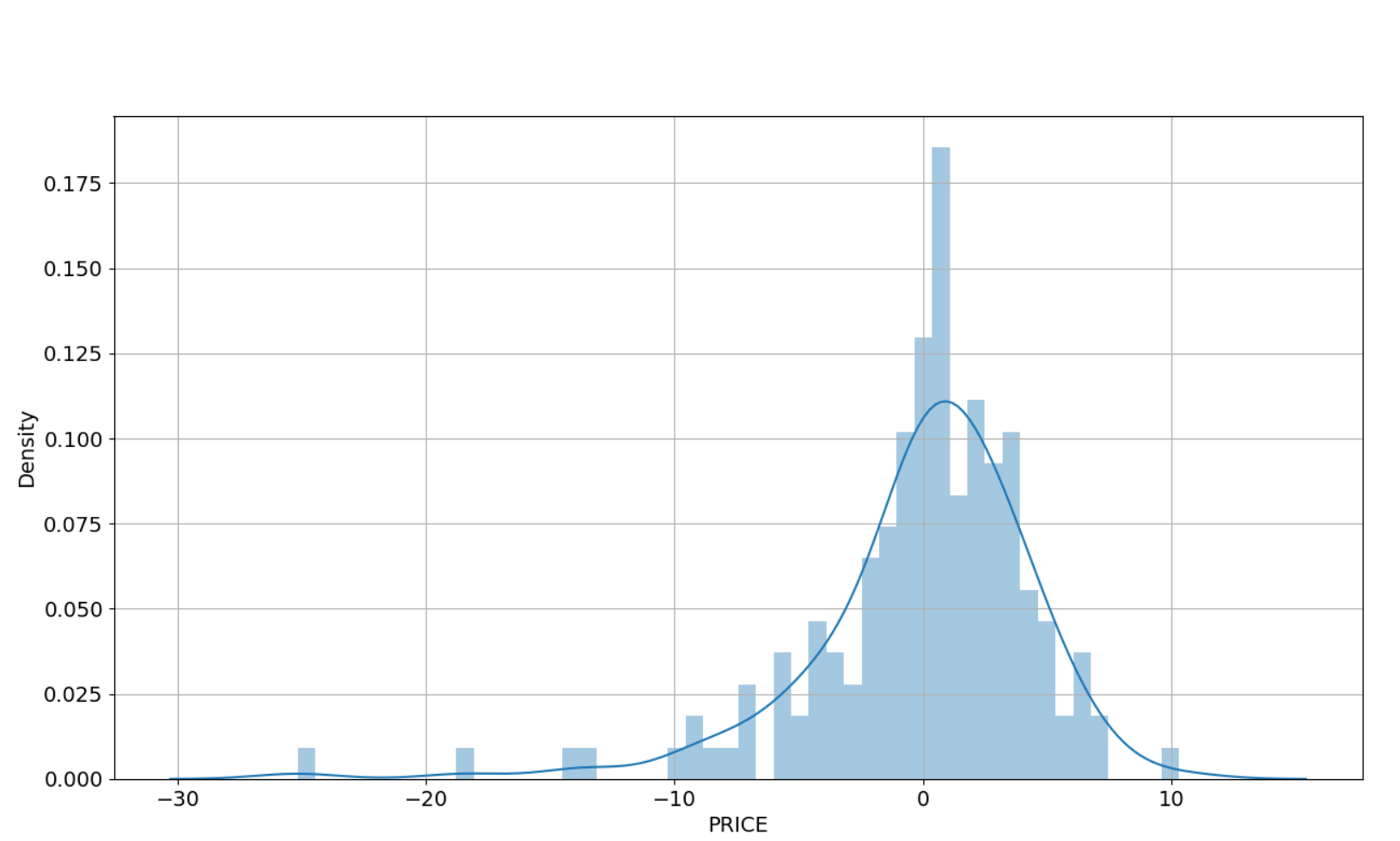
Помимо отображения точек предсказаний важно также оценить распределение ошибок, что может сказать о возможном смещении результатов.

y\_residuals = y\_pred-y\_test

sns.distplot(y\_residuals, bins=50)

plt.grid()

plt.show()

Результат

При использовании стандартизации происходит определение необходимых параметров (среднее, стандартное отклонение) методом fit() и дальнейшая предобработка данных методом transform().

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

# При повторении кода лучшей практикой является создание функции

# для исключения дублирований

# Дублирование кода -> ошибки

def evaluate\_regressor(reg, X, y, mode):

y\_pred = reg.predict(X)

r2\_score\_value = r2\_score(y, y\_pred)

rmse\_score = np.sqrt(mean\_squared\_error(y, y\_pred))

mae\_score = mean\_absolute\_error(y, y\_pred)

print(f'\n----- Evaluation for {mode} -----')

print(f' R2: {r2\_score\_value}')

print(f' RMSE: {rmse\_score}')

print(f' MAE: {mae\_score}')

evaluate\_regressor(reg, X\_train, y\_train, 'train')

evaluate\_regressor(reg, X\_test, y\_test, 'test')

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

reg = LinearRegression()

# "Обучение" трансформера - определение параметров средних и отклонений

scaler.fit(X\_train)

# Стандартизация данных

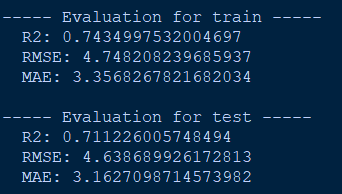
X\_train\_scaled = scaler.transform(X\_train)

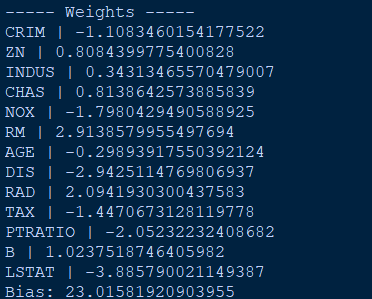
X\_test\_scaled = scaler.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

Результат





Важно обратить внимание на порядок весов модели. Использование стандартизации делает данные более центрированными и теперь веса больше отражают влияние на результат.

means = scaler.mean\_

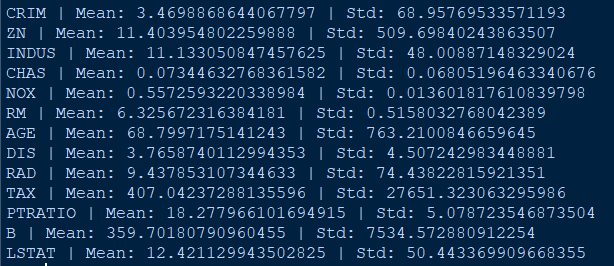
stds = scaler.var\_

pairs = zip(X\_test.columns, means, stds)

for pair in pairs:

print(f'{pair[0]} | Mean: {pair[1]} | Std: {pair[2]}')

Результат



# **Задание**

Требуется создать ноутбук, в котором будут проведены эксперименты с инструментами sklearn.

Задачи:

* Произведите базовую подготовку данных;
* Создайте baseline модель линейной регрессии;

Произведите расширенный анализ данных:

Оцените корреляции признаков;

Оцените распределения признаков;

Оцените характер зависимости целевой переменной от каждого признака;

Напишите свои выводы по результатам анализа.

Изучите работу инструментов нормализации данных:

Примените трансформер sklearn.preprocessing.StandardScaler к данным, отобразите распределения после обработки, обучите модель на обработанных данных и оцените показатели;

Примените трансформер sklearn.preprocessing.MinMaxScaler к данным, отобразите распределения после обработки, обучите модель на обработанных данных и оцените показатели;

Сравните результаты и сделайте выводы.

Изучите работу инструментов выравнивания распределений:

Примените трансформер sklearn.preprocessing.PowerTransformer к данным, отобразите распределения после обработки, обучите модель на обработанных данных и оцените показатели;

Примените трансформер sklearn.preprocessing.QuantileTransformer(output\_distribution='normal') к данным, отобразите распределения после обработки, обучите модель на обработанных данных и оцените показатели;

Сравните результаты и сделайте выводы.

Исключите выбросы на уровне PRICE == 50 из данных, обучите модель и оцените показатели;

Оцените работу модели при исключении каждого признака [13 экспериментов] (результаты представить в виде таблицы DataFrame с указание исключенного признака и показателей метрик);

Примените трансформер sklearn.preprocessing.PolynomialFeatures для генерации признака LSTAT в квадрате, обучите модель на данных с новым признаком, оцените работу модели;

Создайте базовую модель Ридж регрессии <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html>;

Испытайте работу модели при использовании различных методов нормализации и выравнивания распределений;

Оцените работу модели при различных IMG_256 (изучить не менее 7 различных значений) и отобразите на графике (y-axis ~ R2, x-axis ~ log(alpha));

Разработайте лучшую модель с необходимой предобработкой для решения задачи;

Написать выводы по результатам.

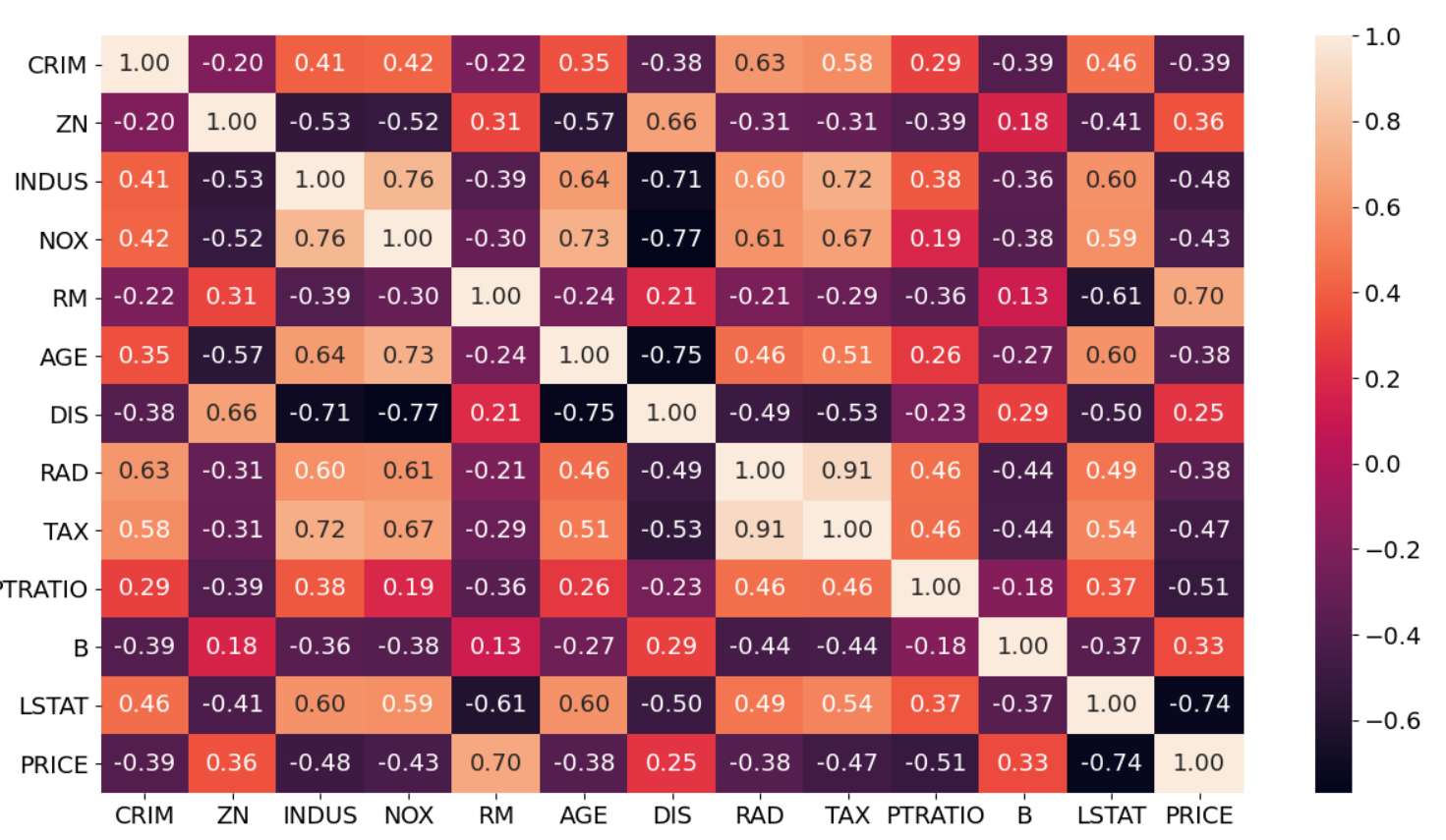
features = ['CRIM', 'ZN','INDUS', 'CHAS','NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT','PRICE']

correlation\_mtrx = df[features].corr()

sns.heatmap(correlation\_mtrx, annot=True, fmt='.2f')

plt.show()

Результат



for i in features:

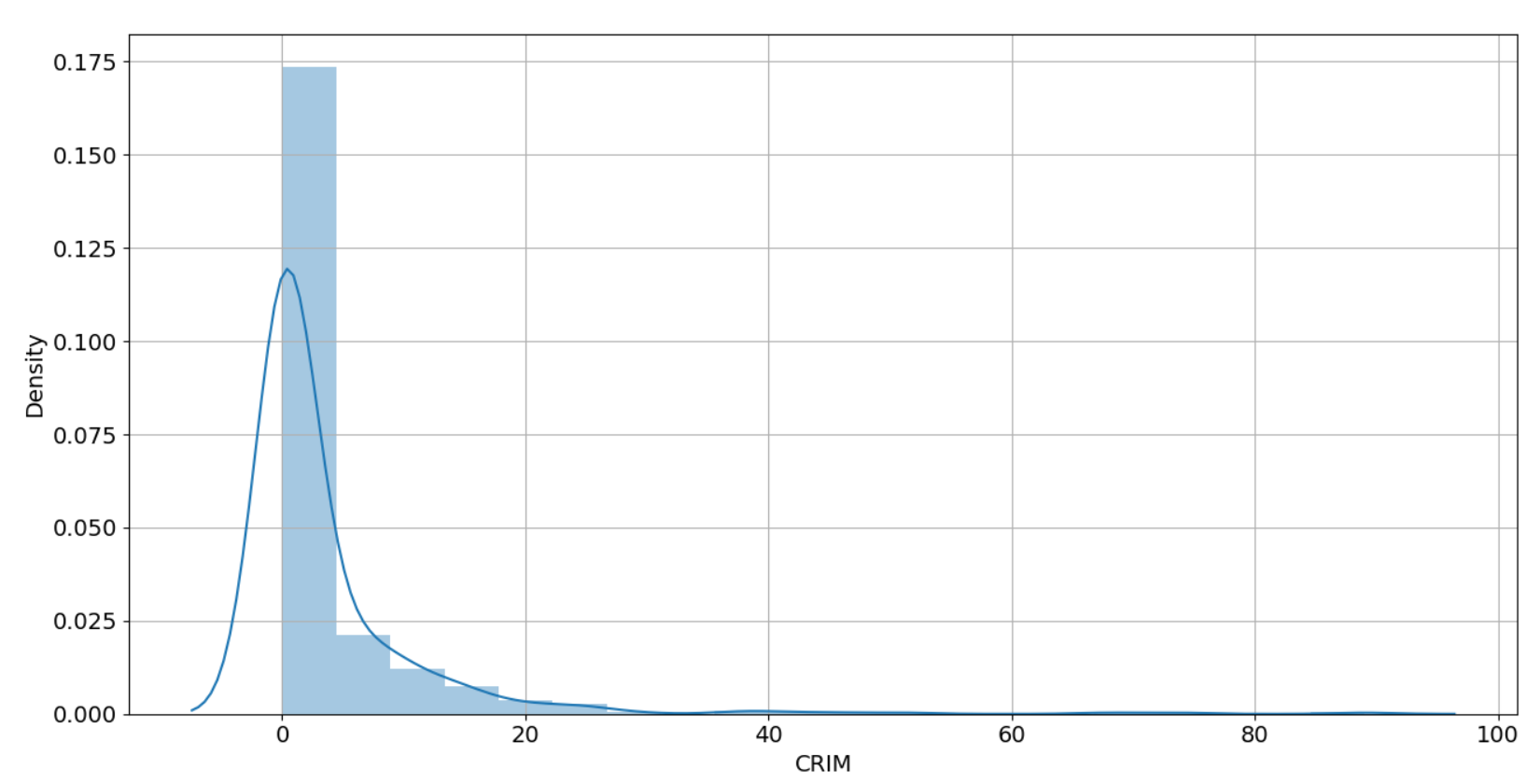
plt.figure(figsize=[5, 3])

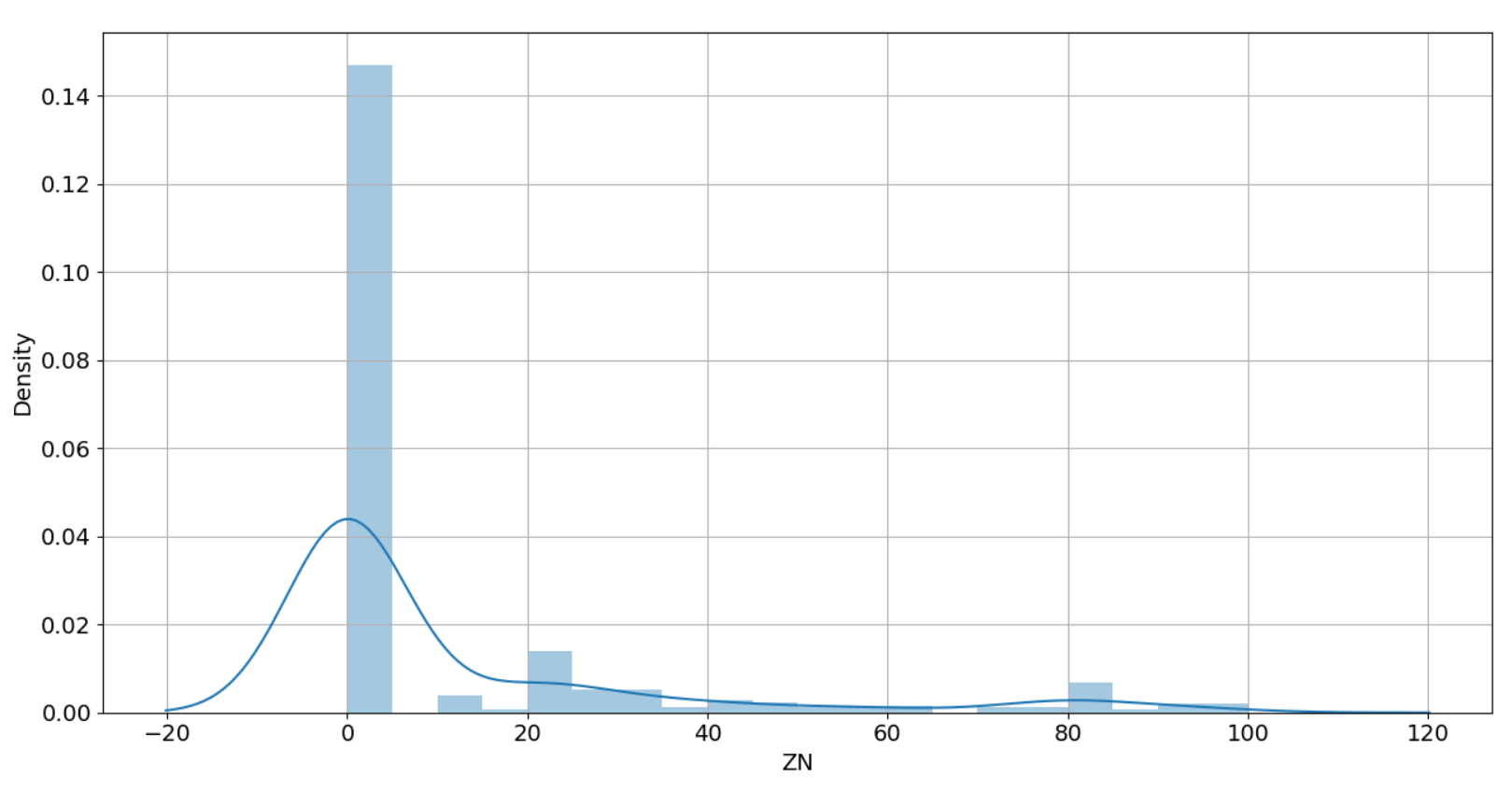
sns.distplot(df[i], bins=20)

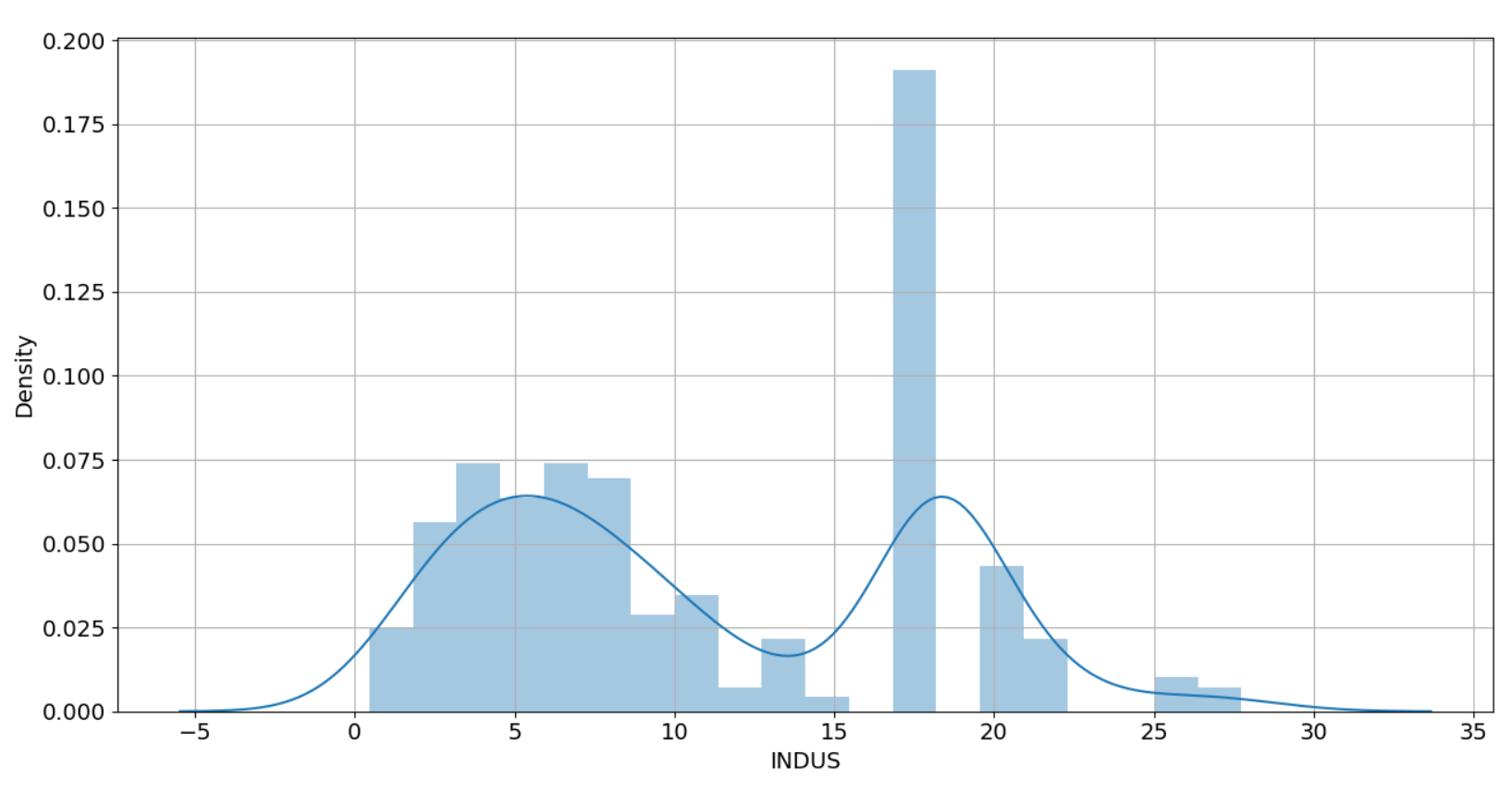
plt.grid()

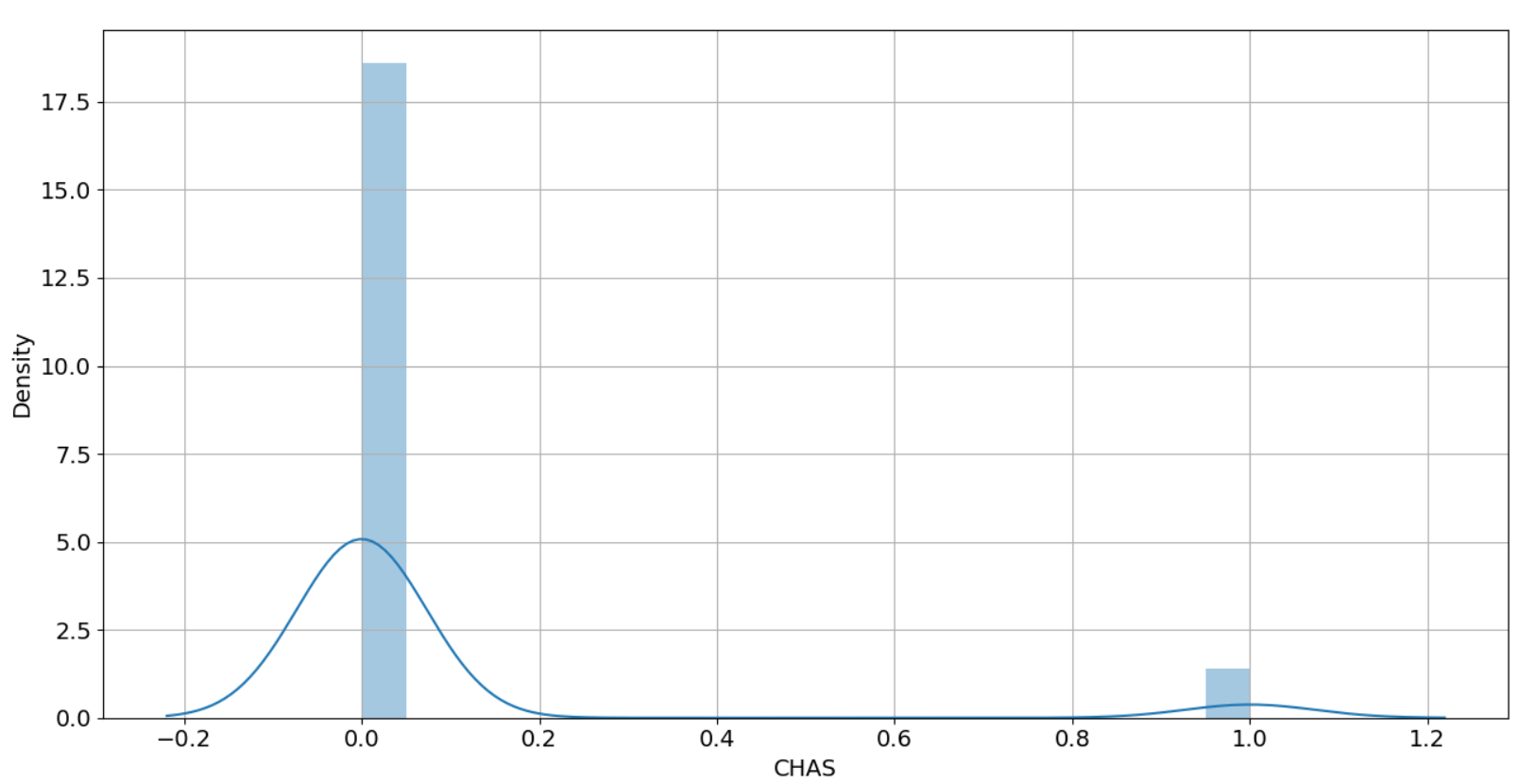
plt.show()

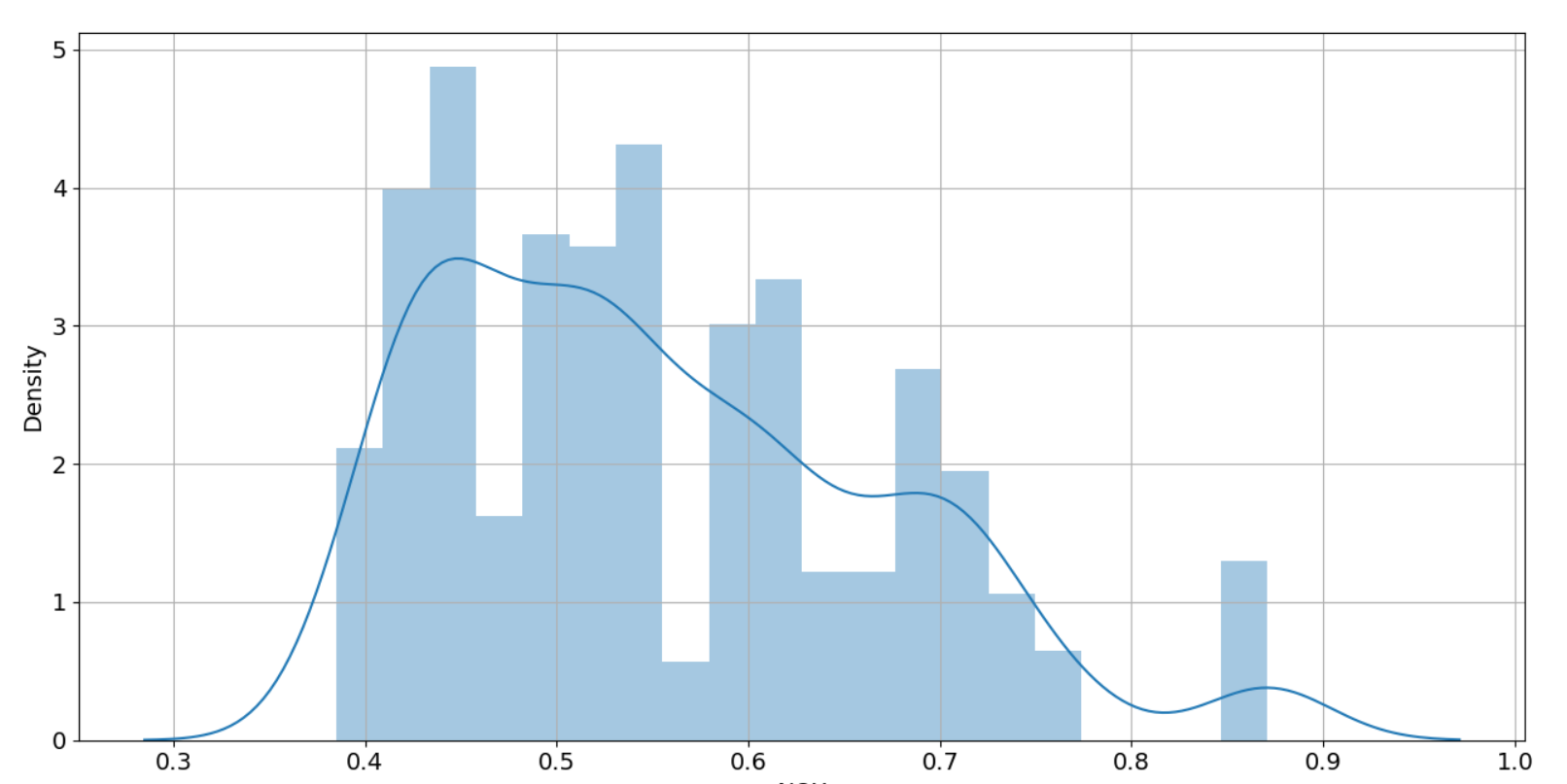
Результат

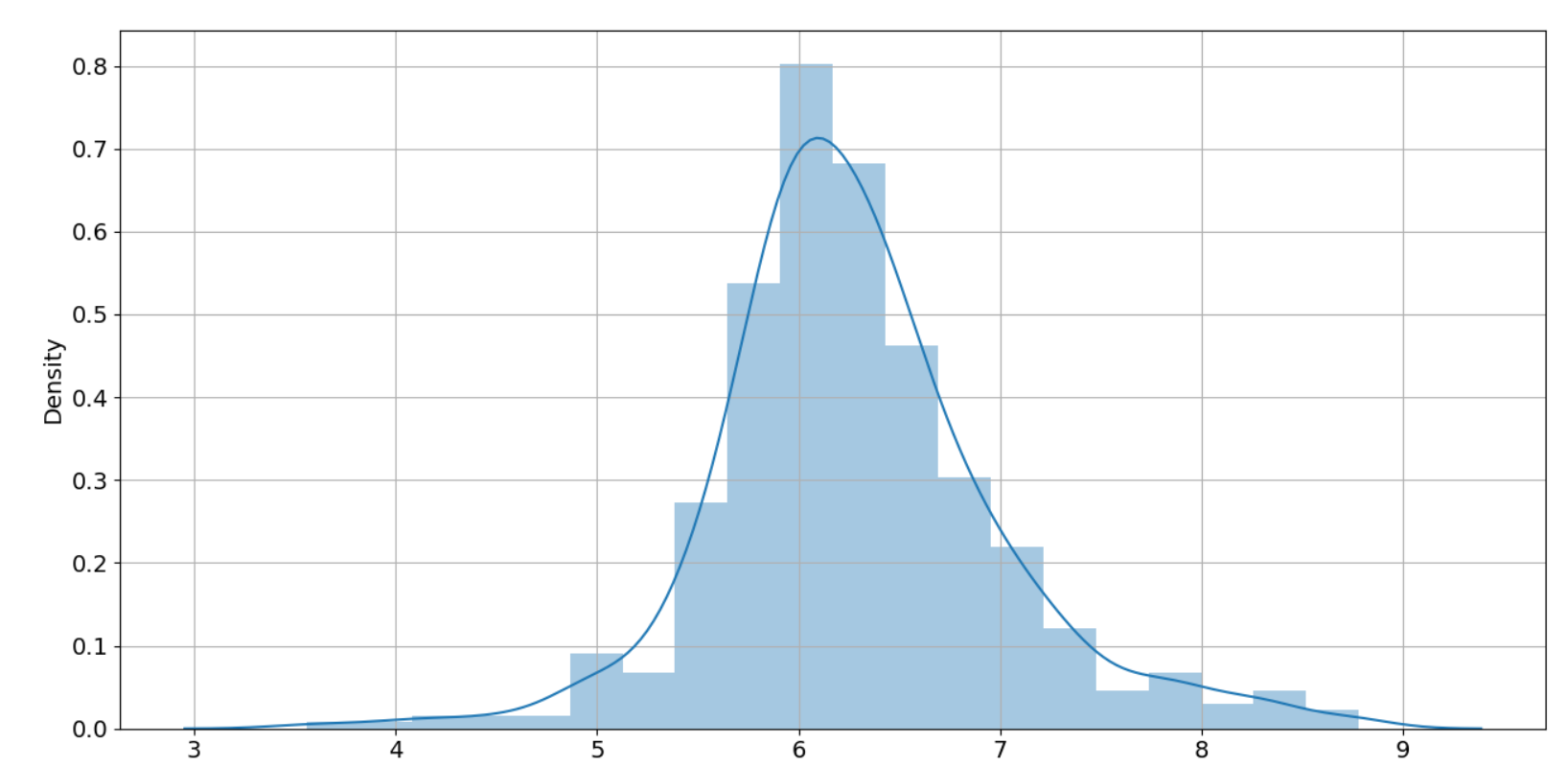


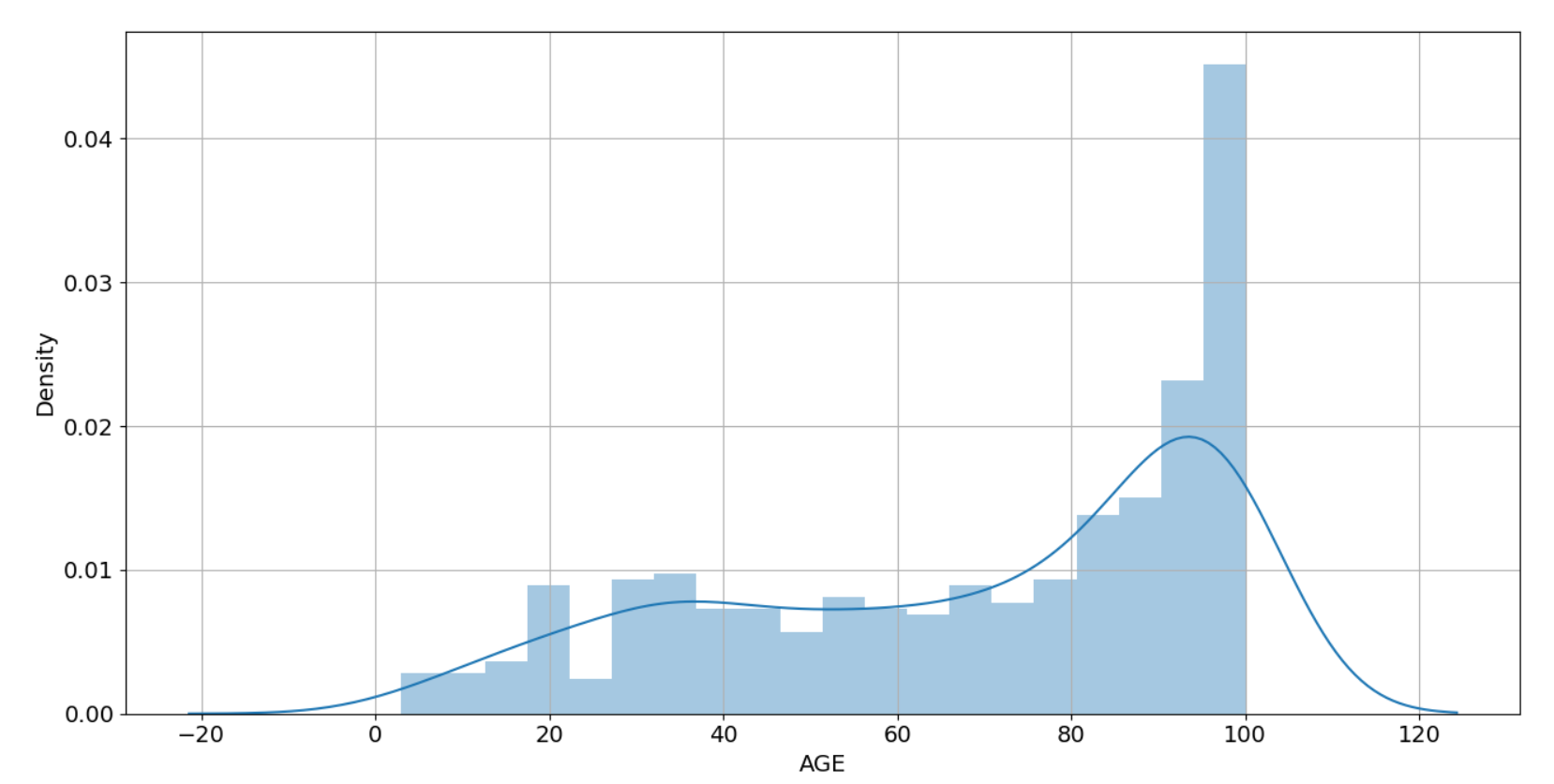


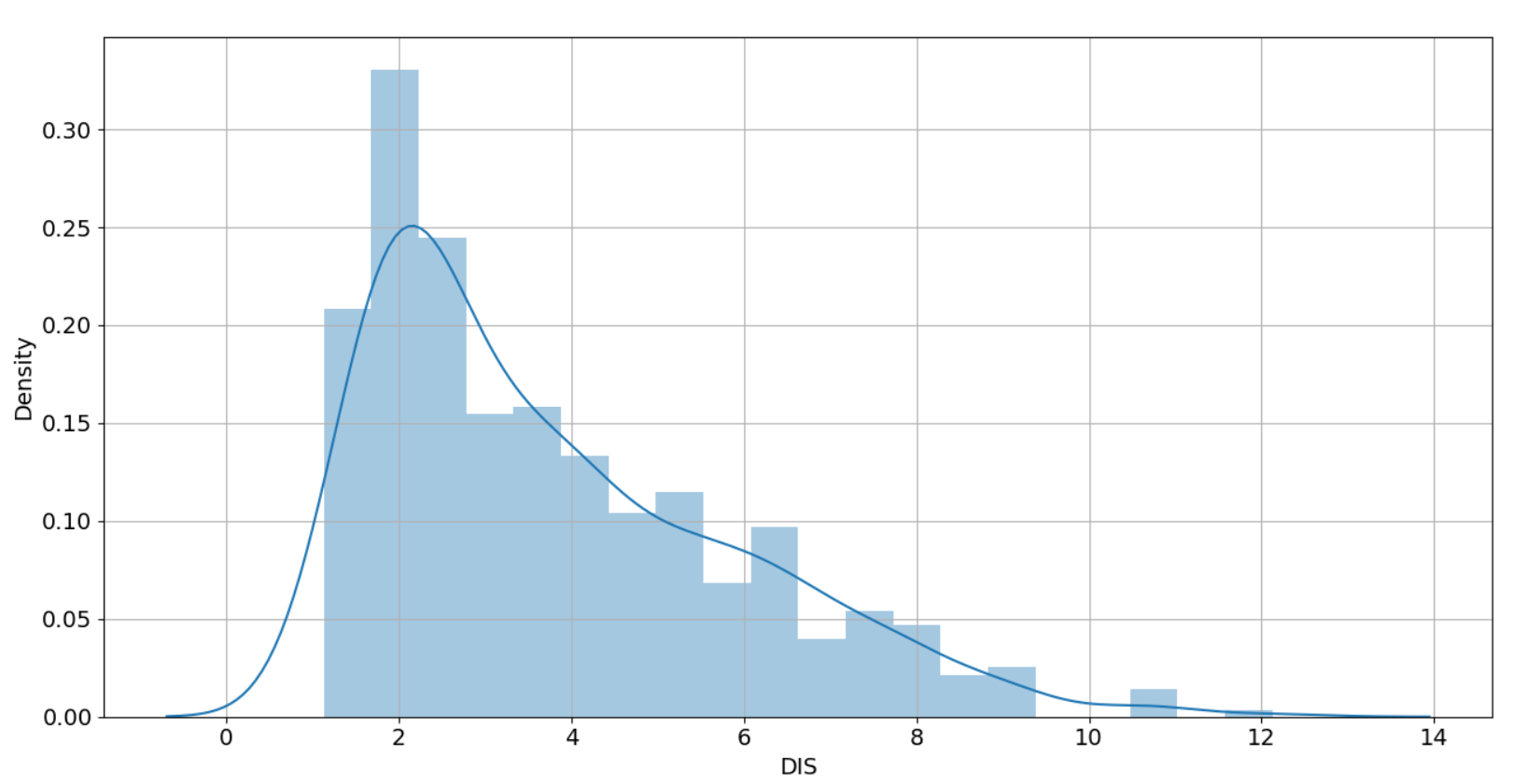


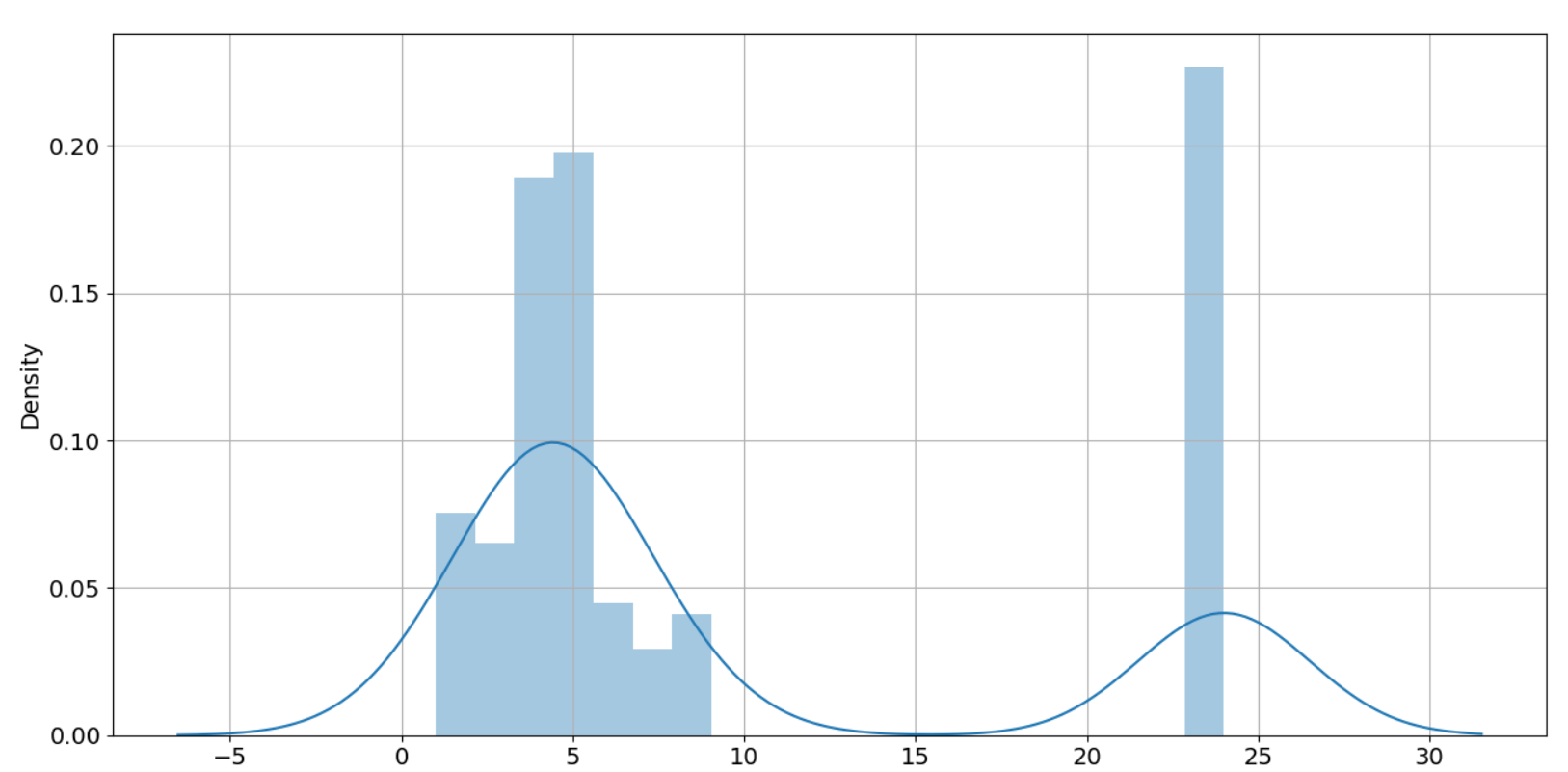


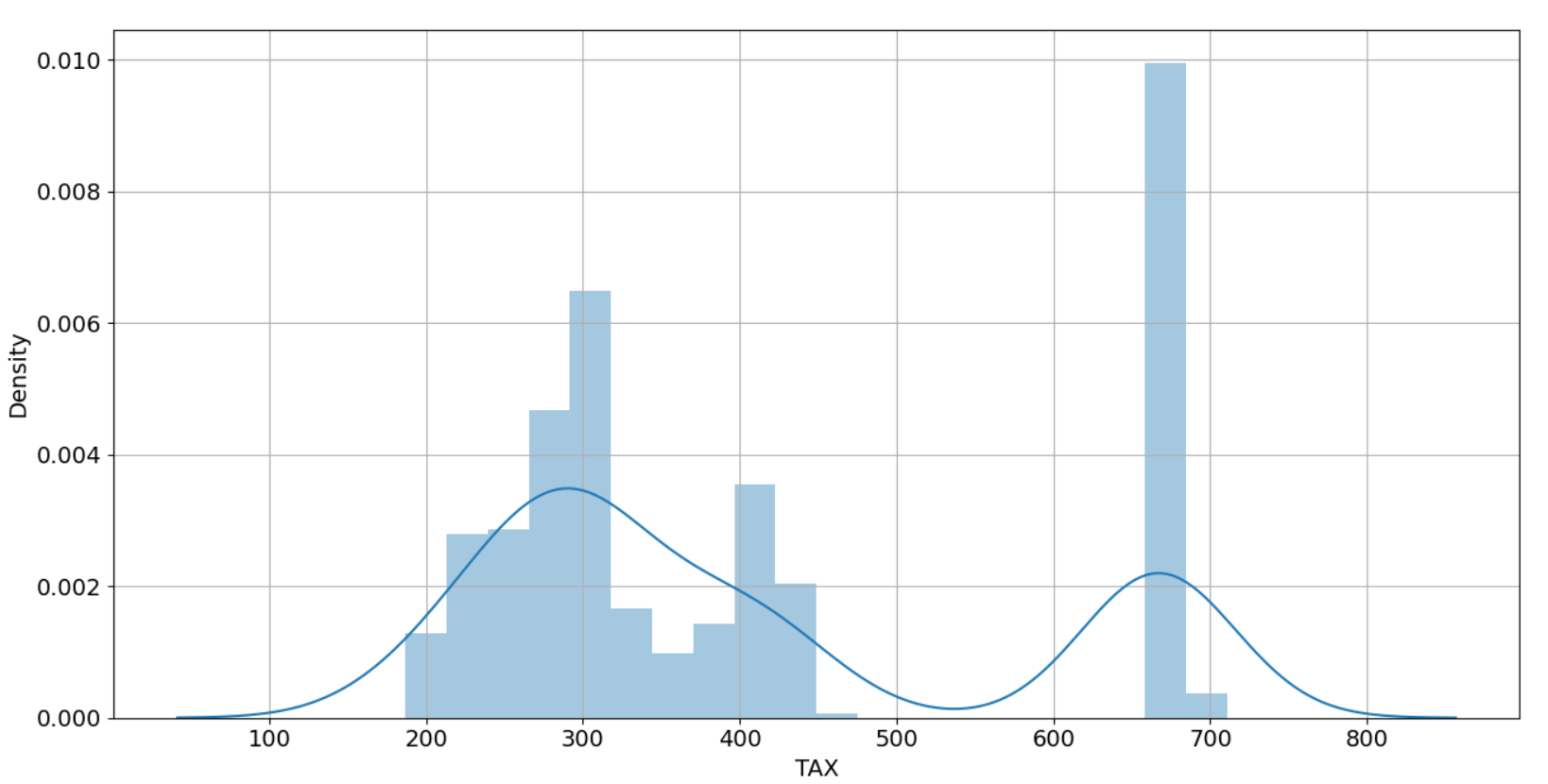


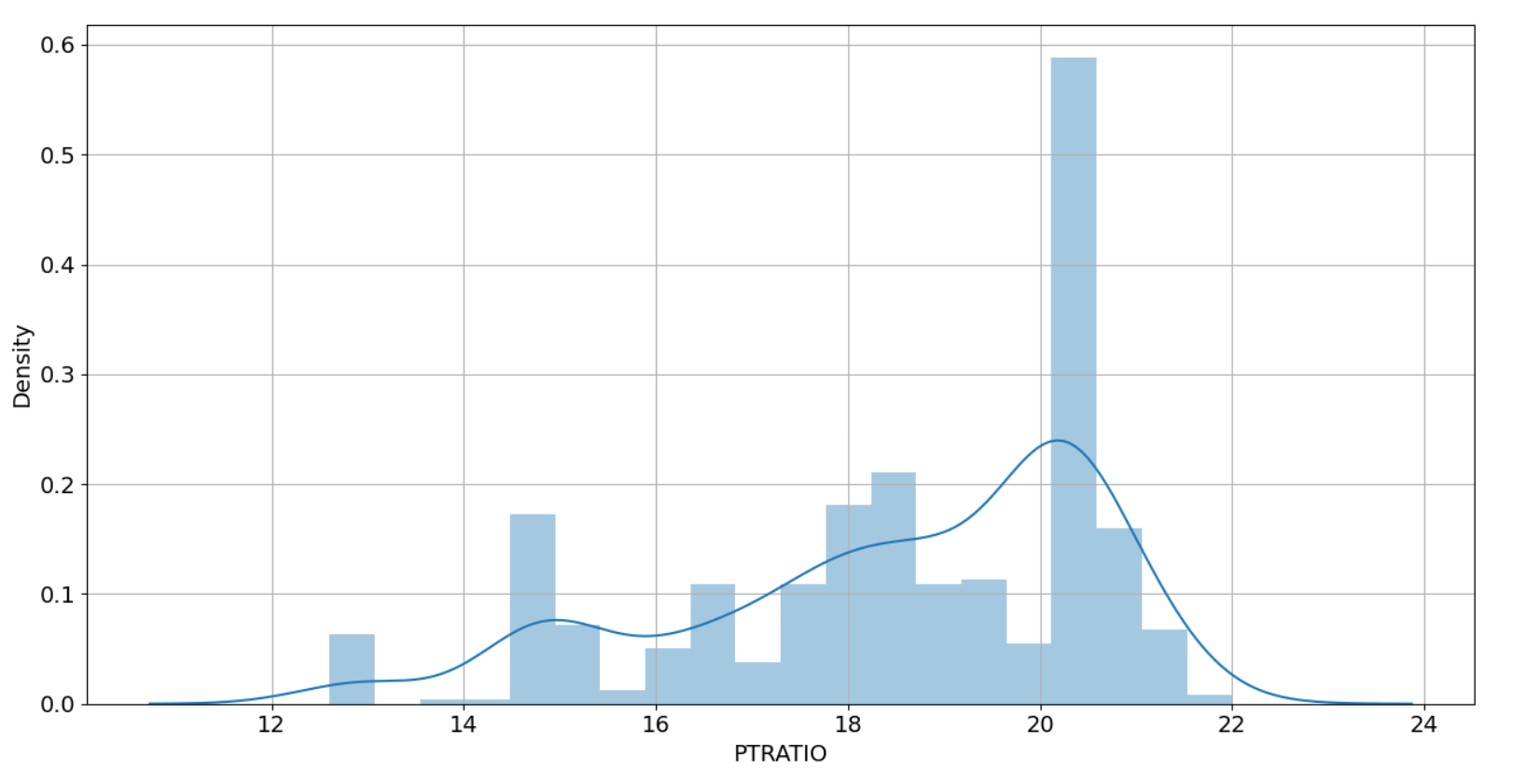


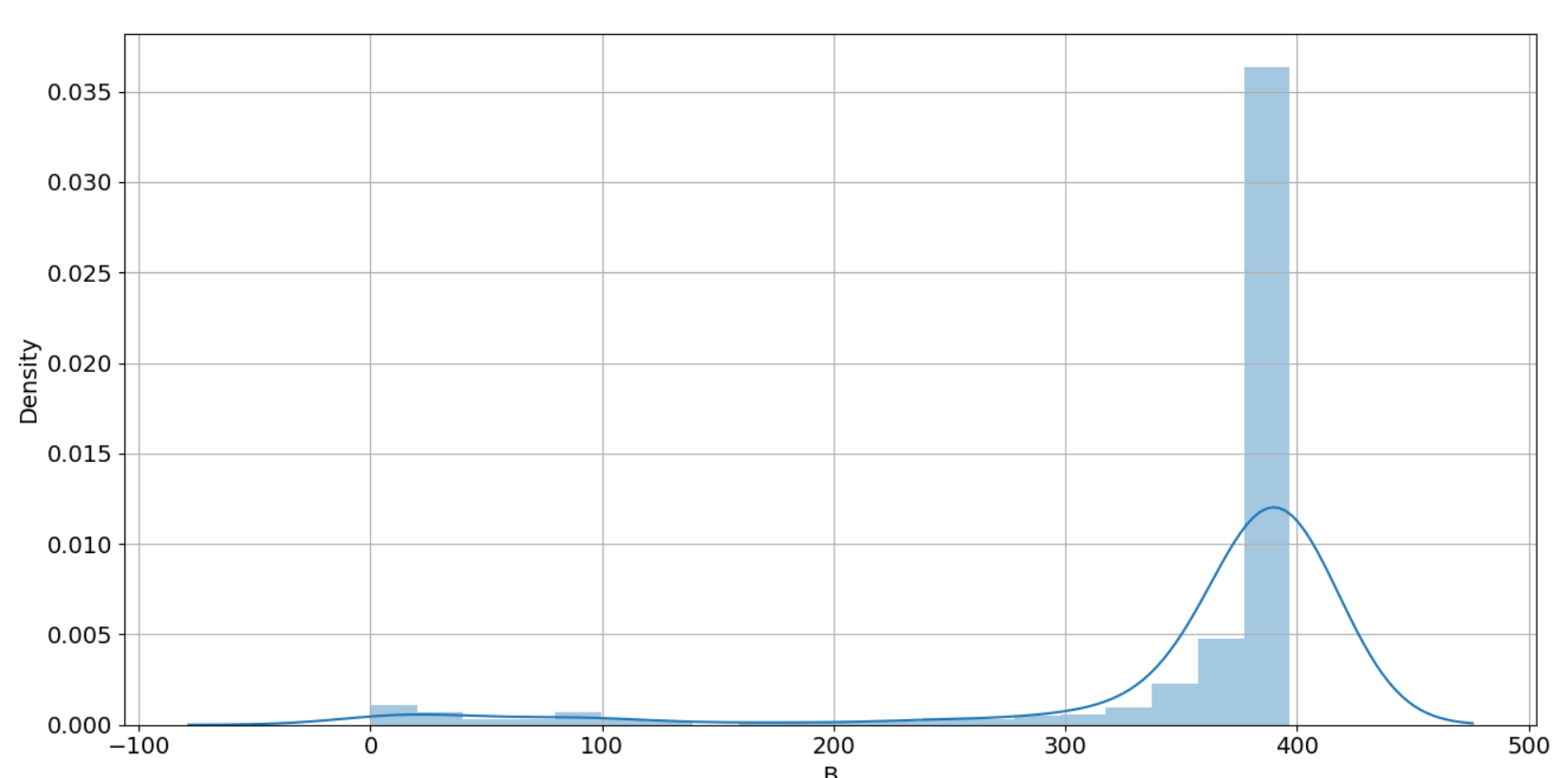


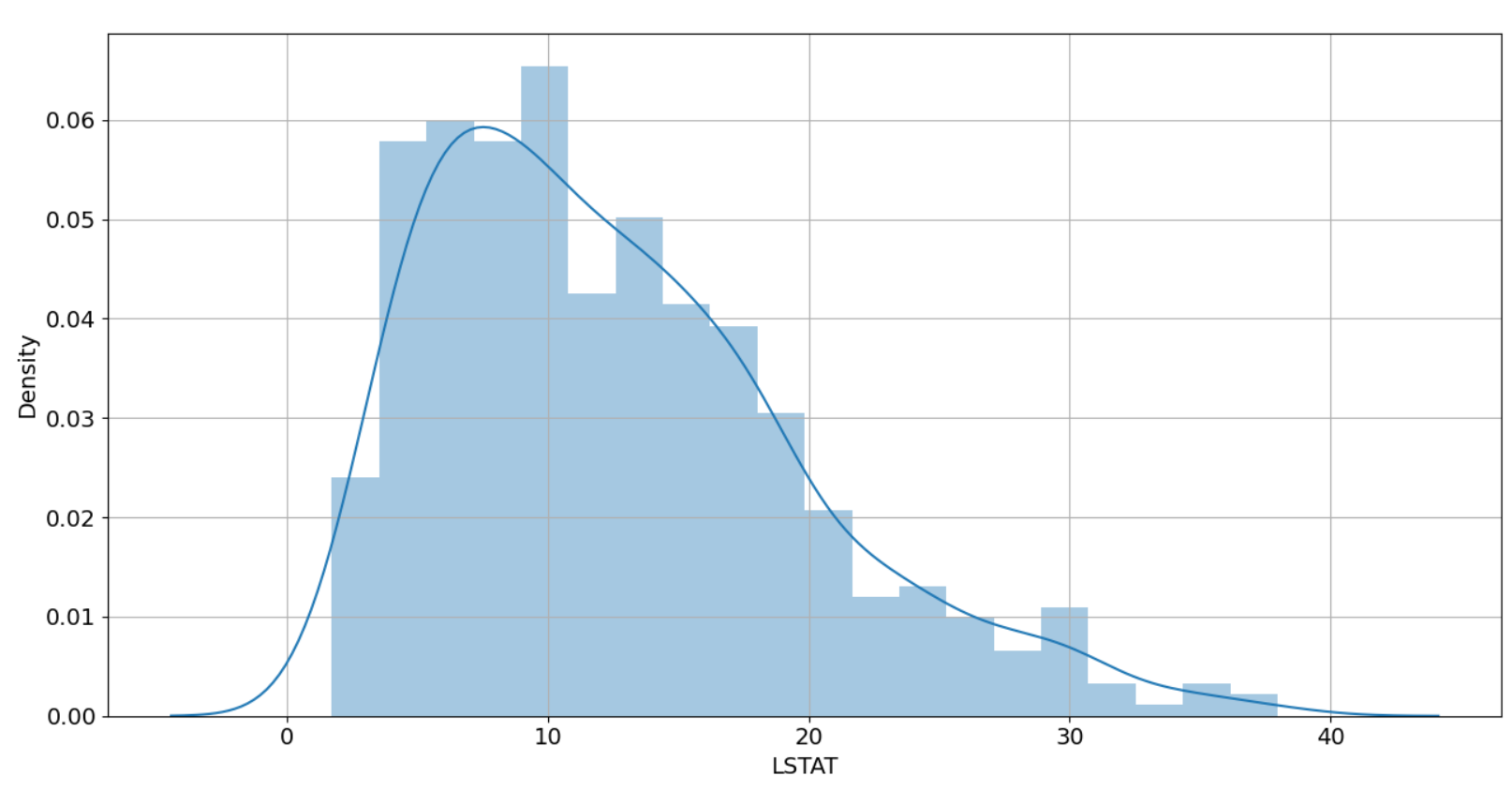


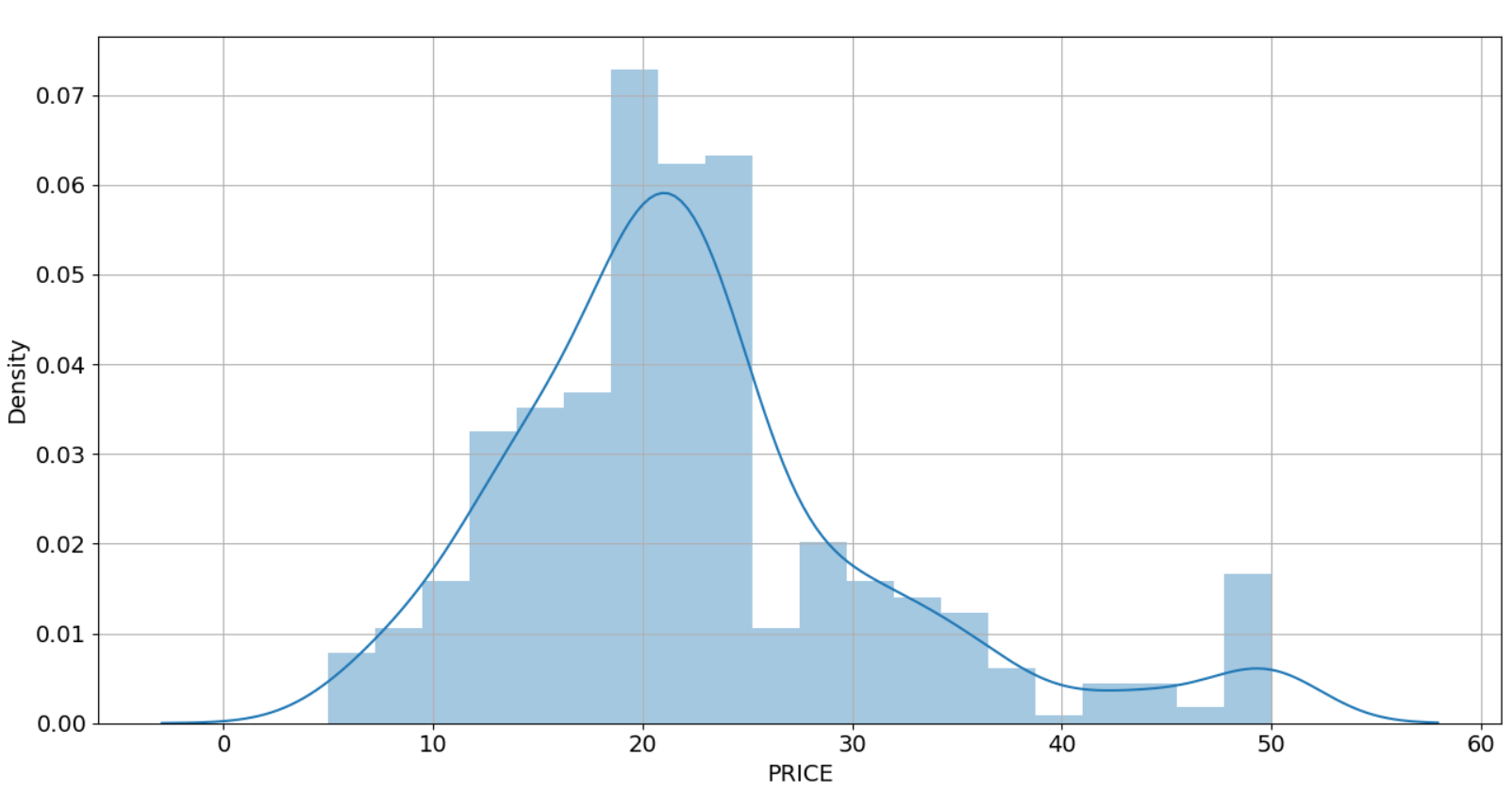












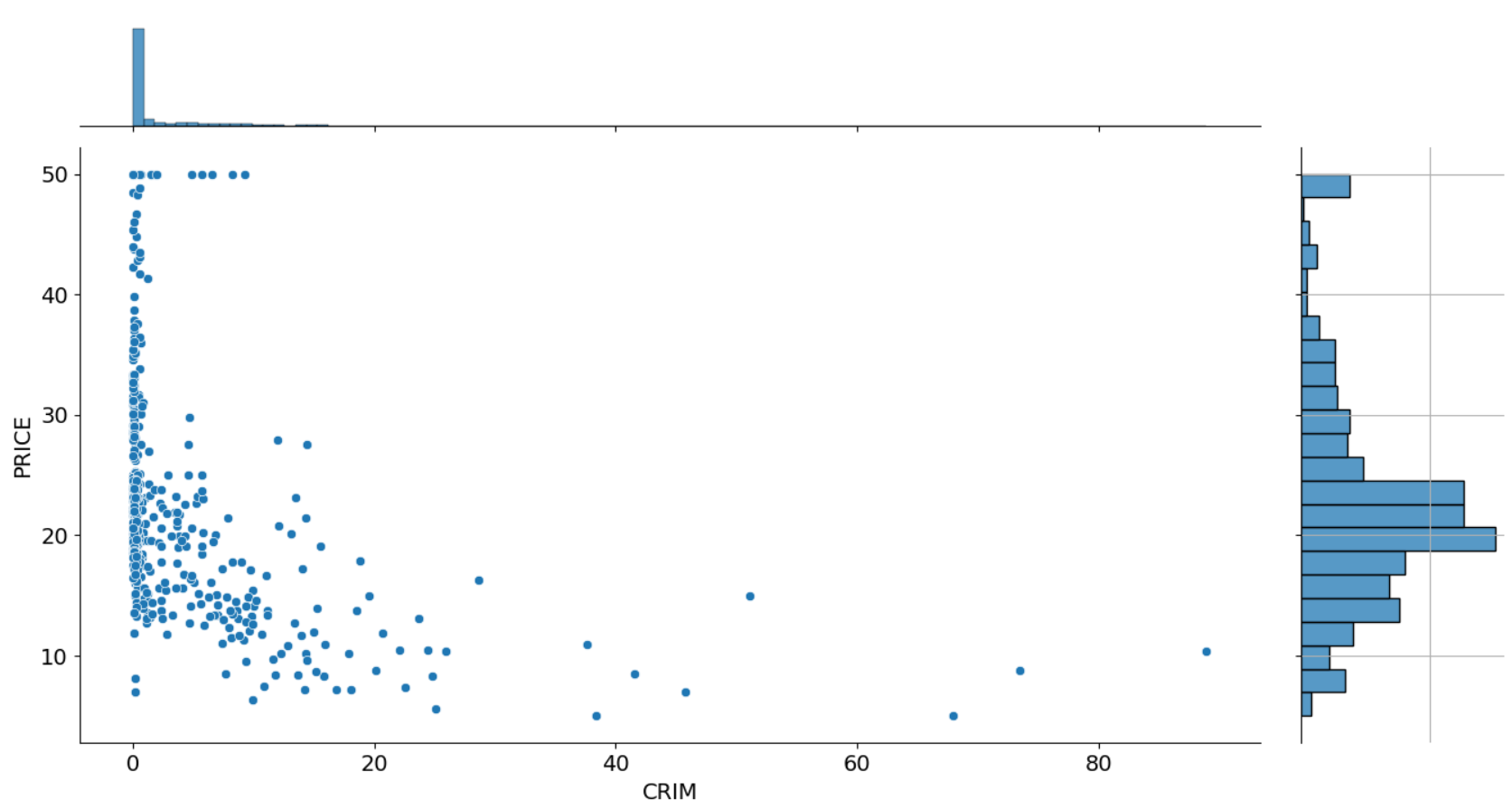
for i in features:

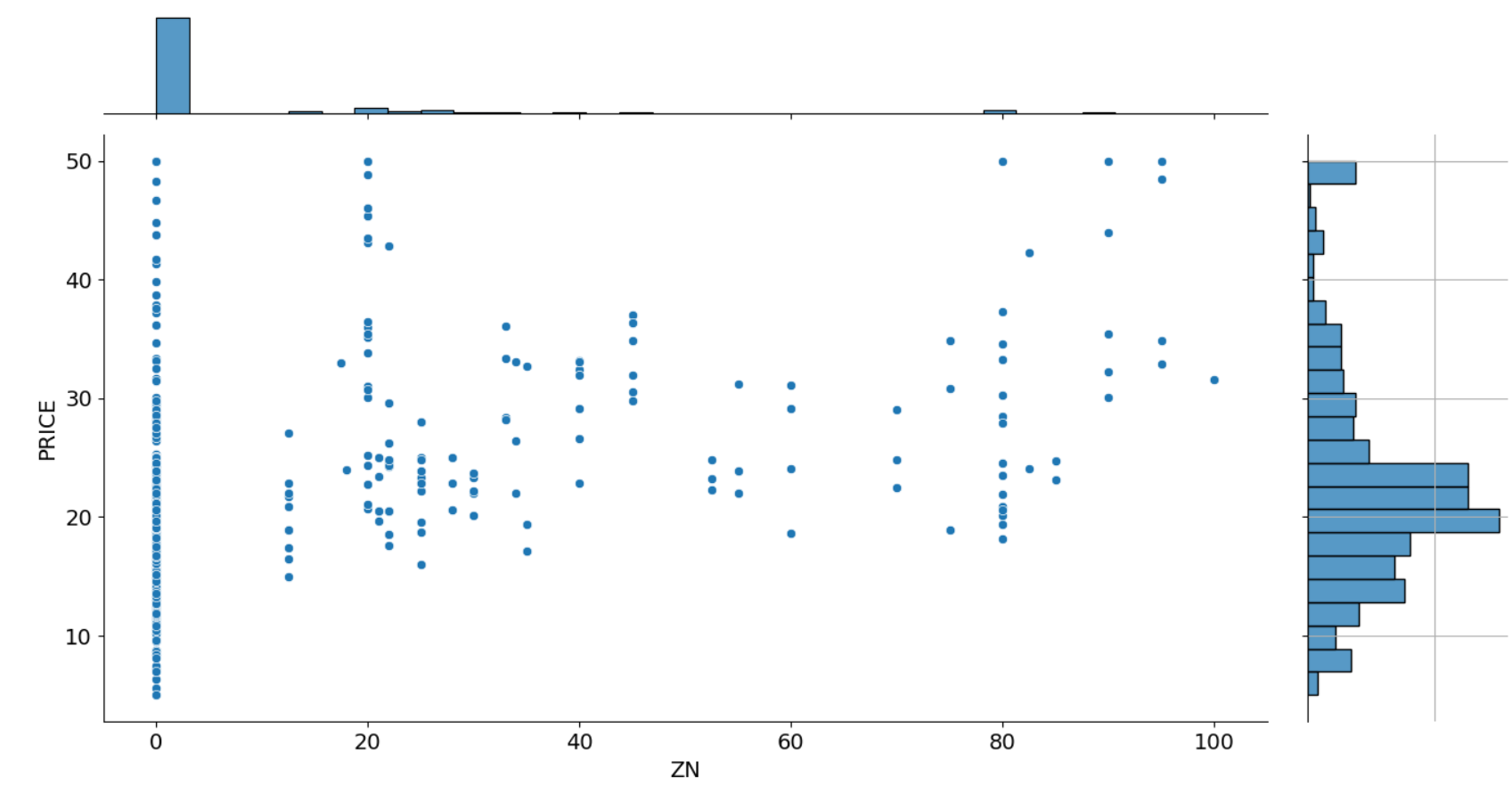
sns.jointplot(x=i, y="PRICE", data=df)

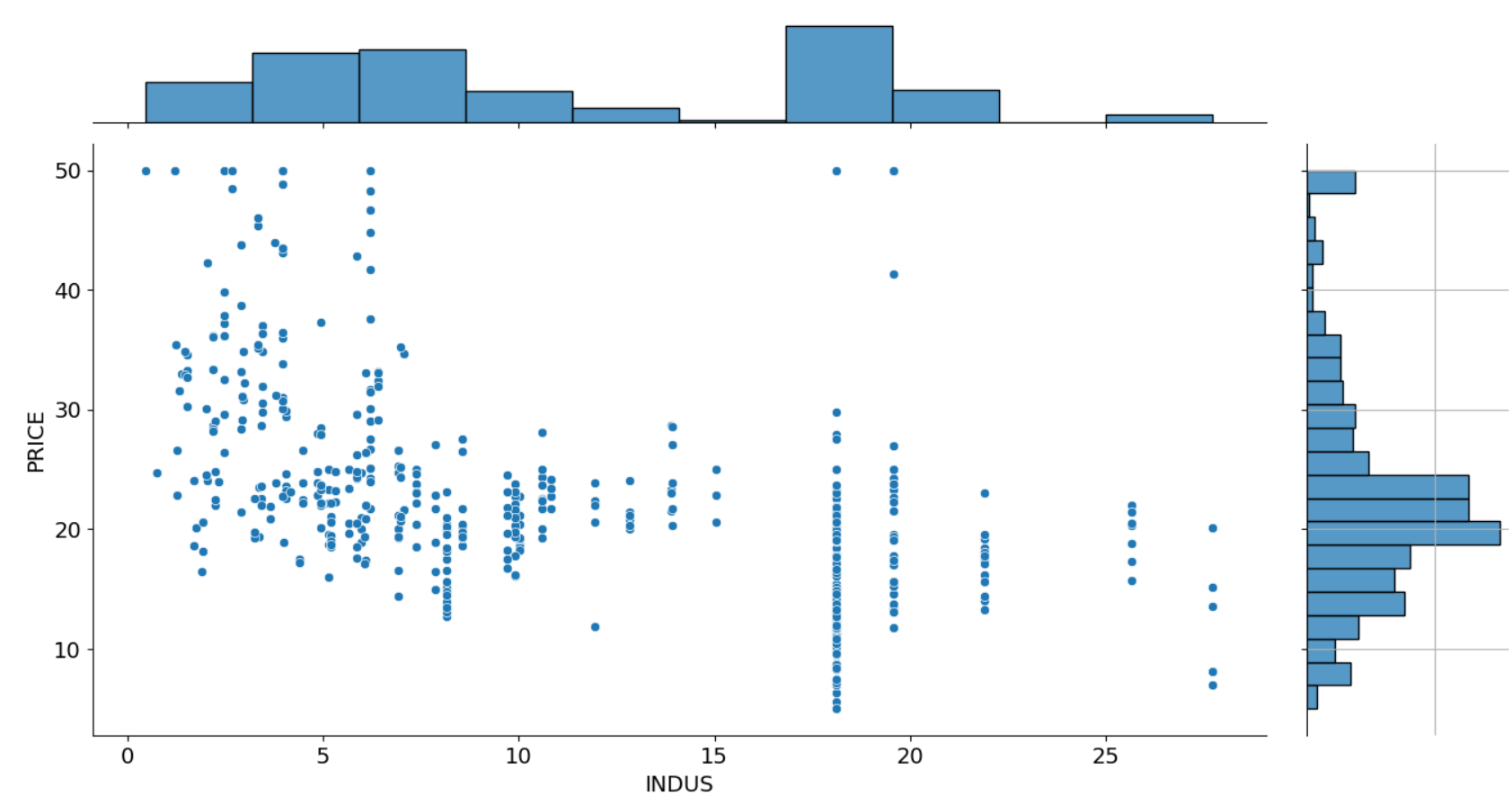
plt.grid()

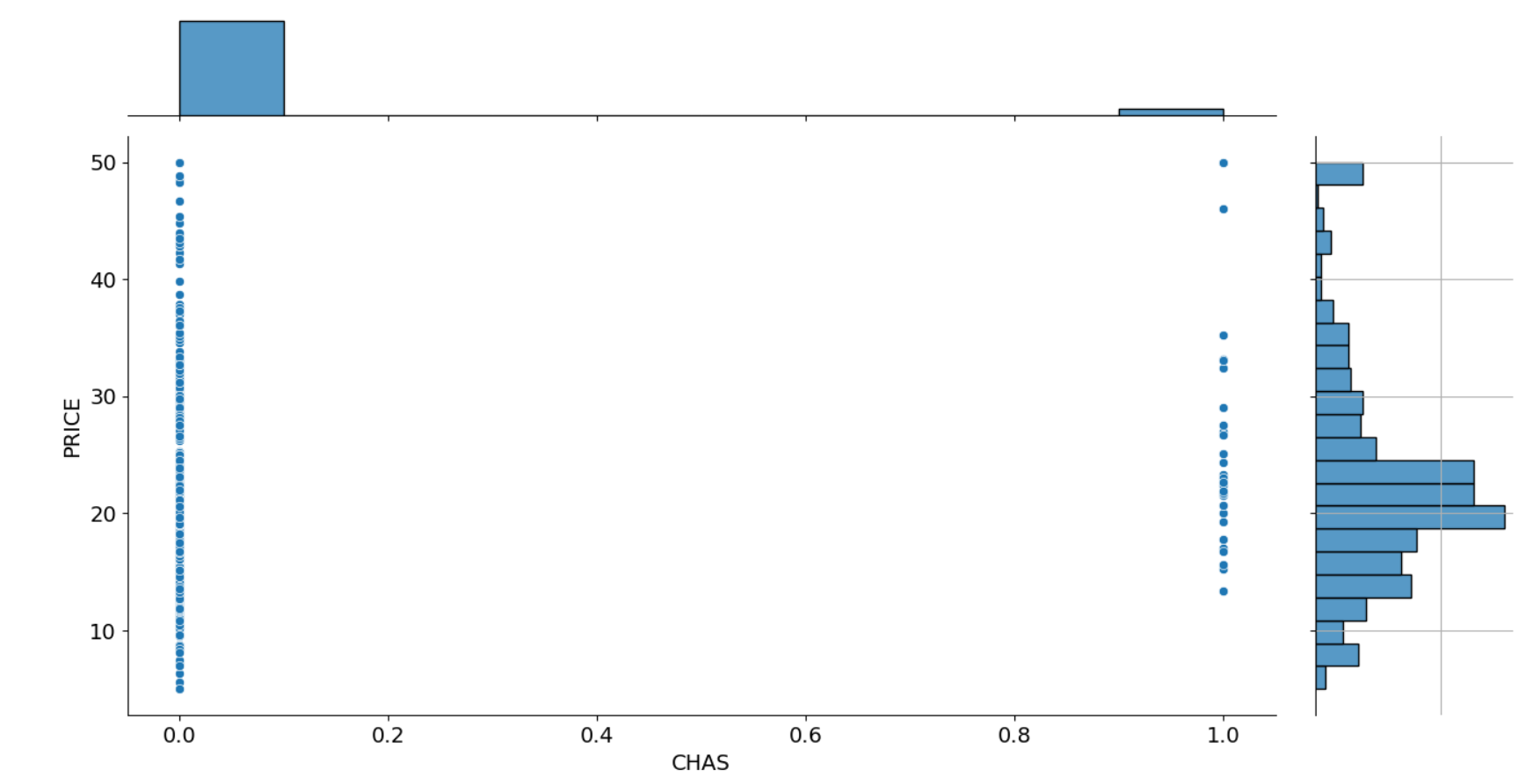
plt.show()

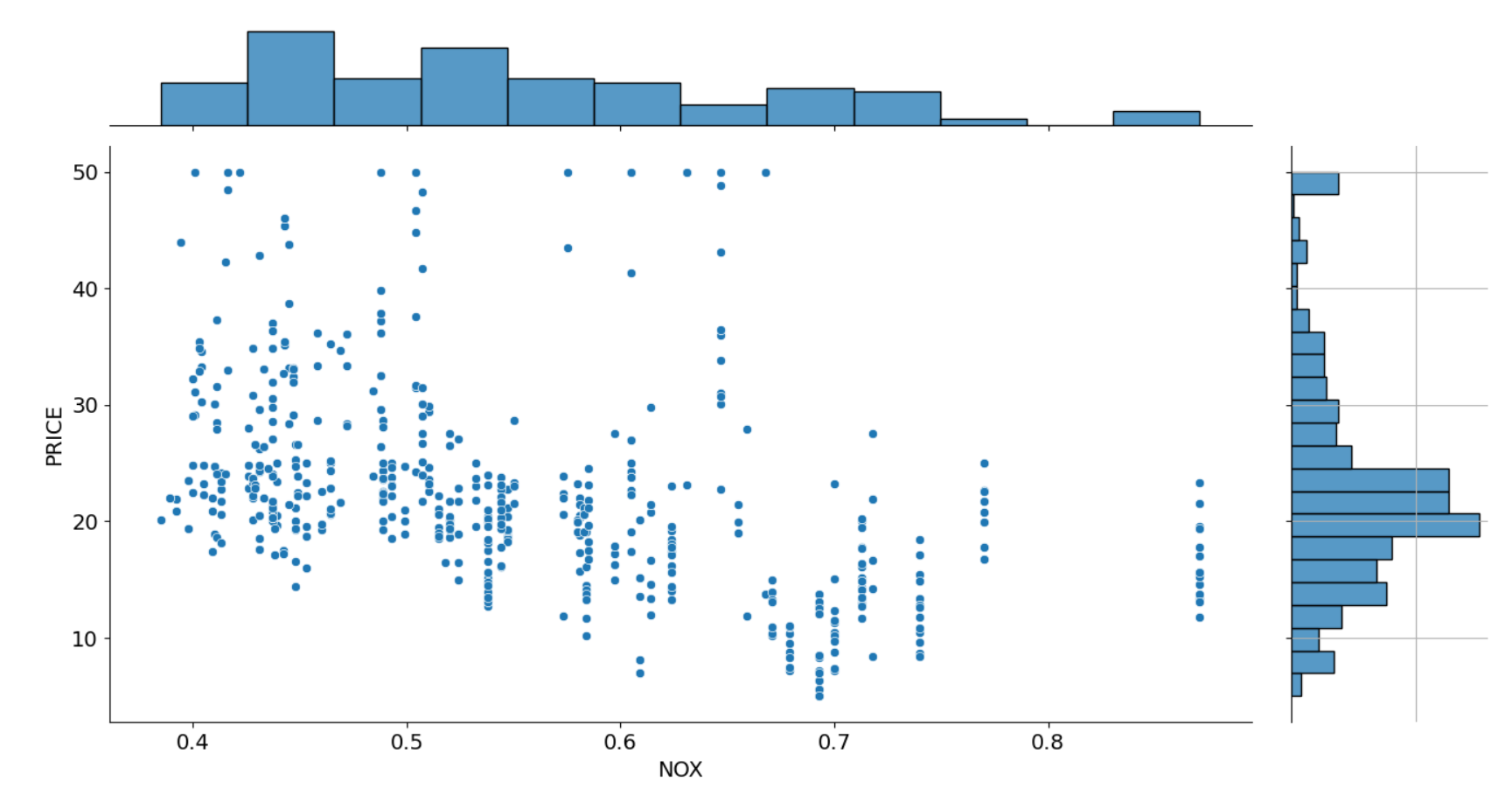
Результат

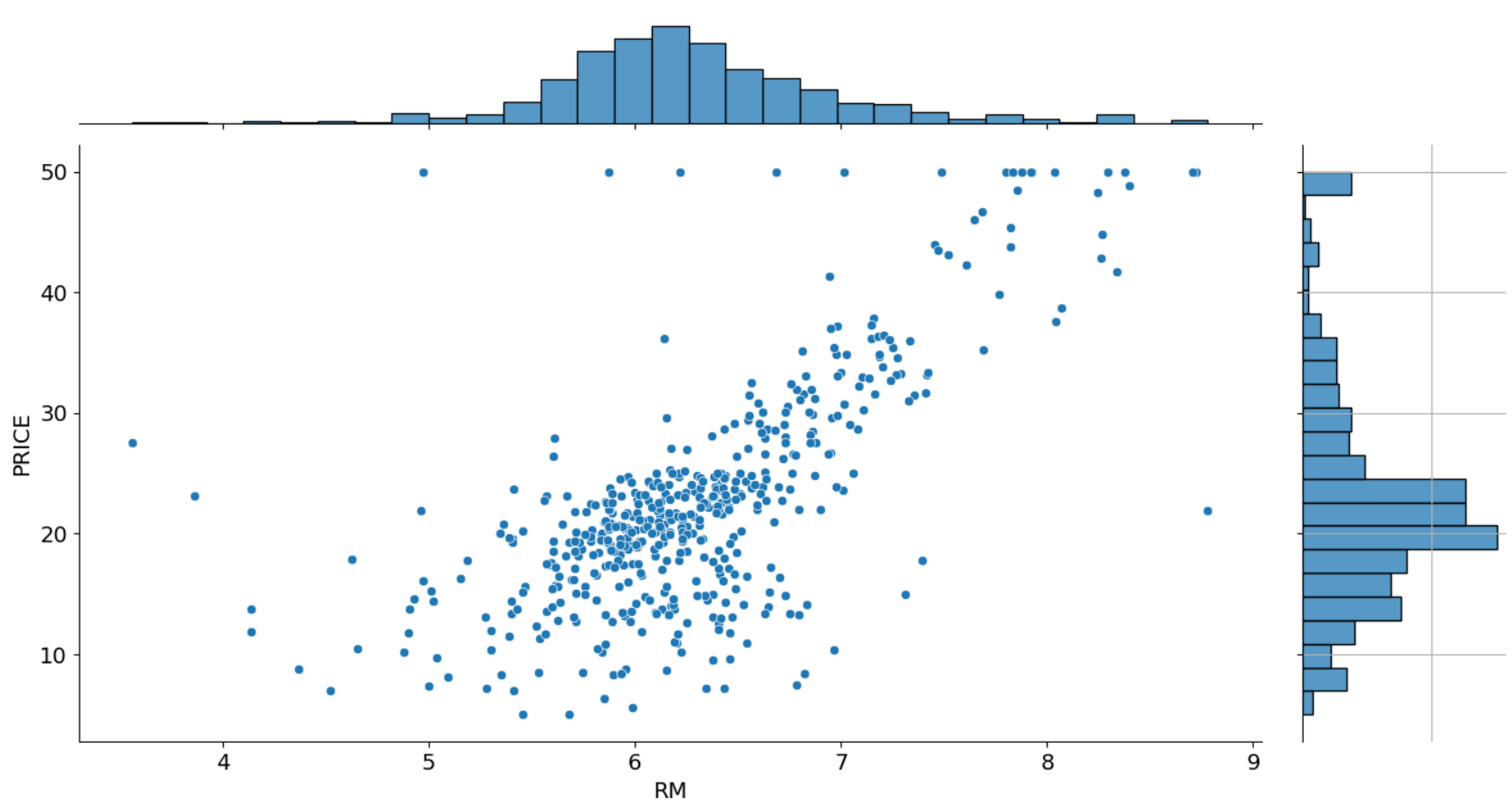


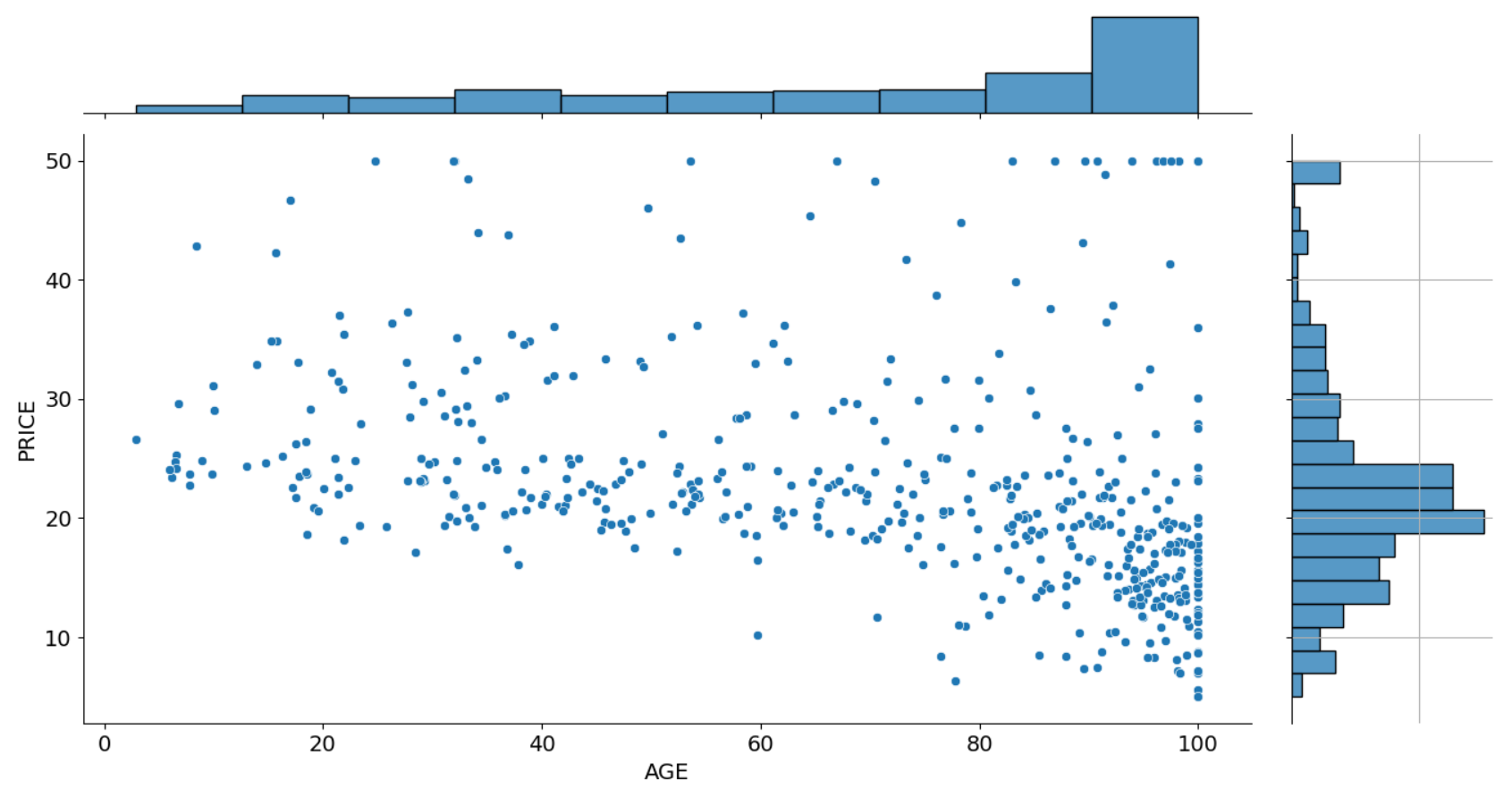


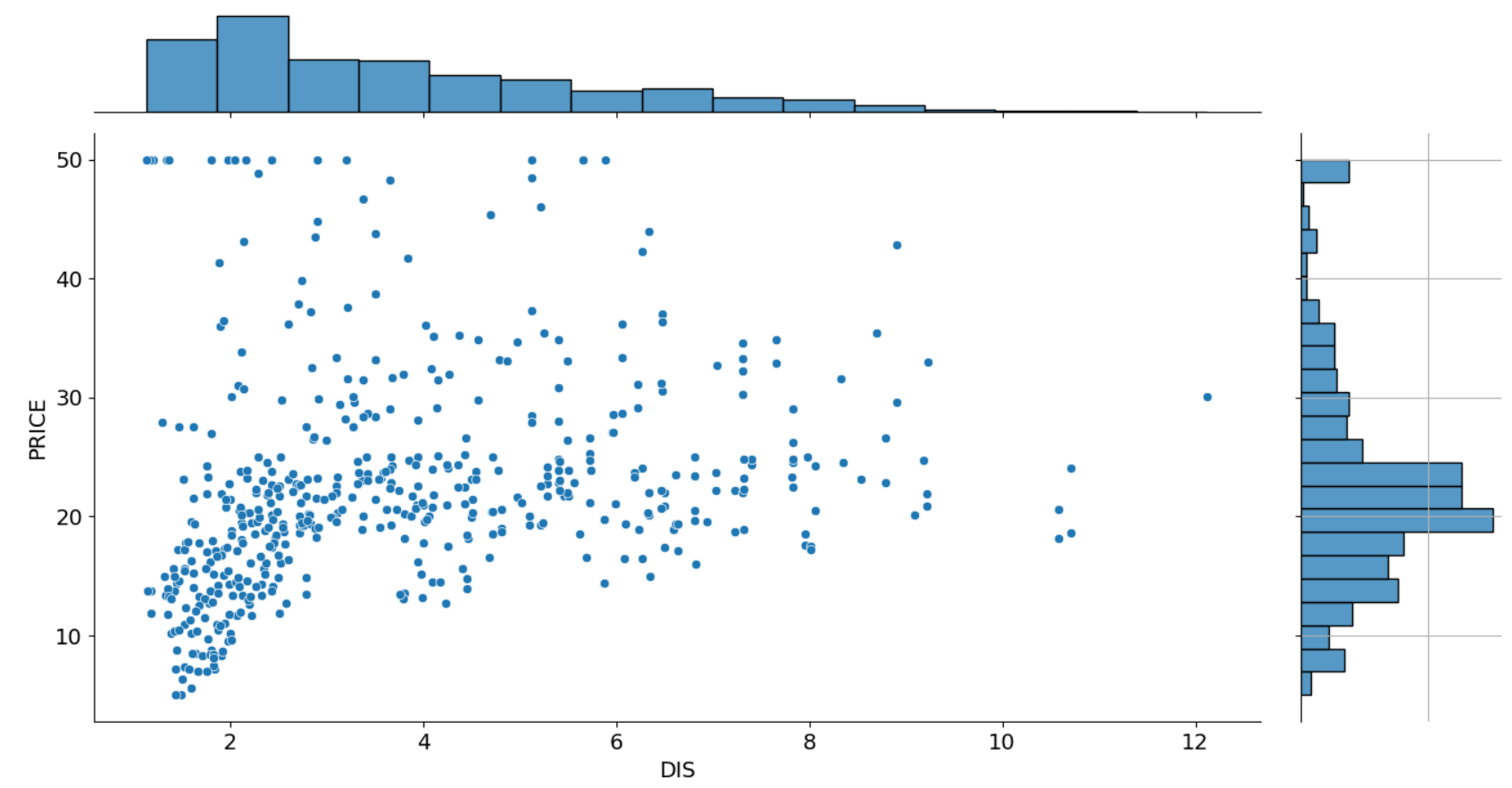


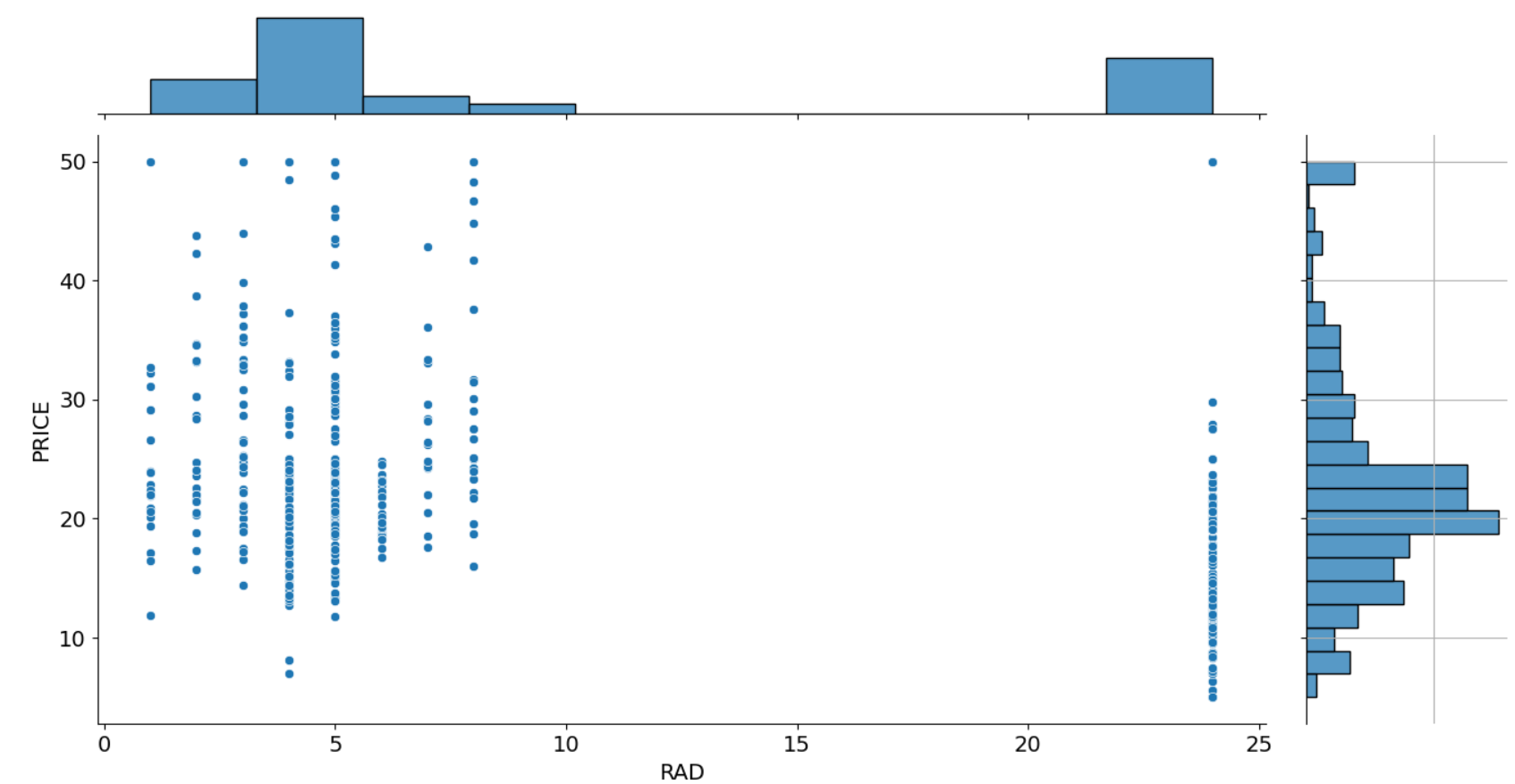


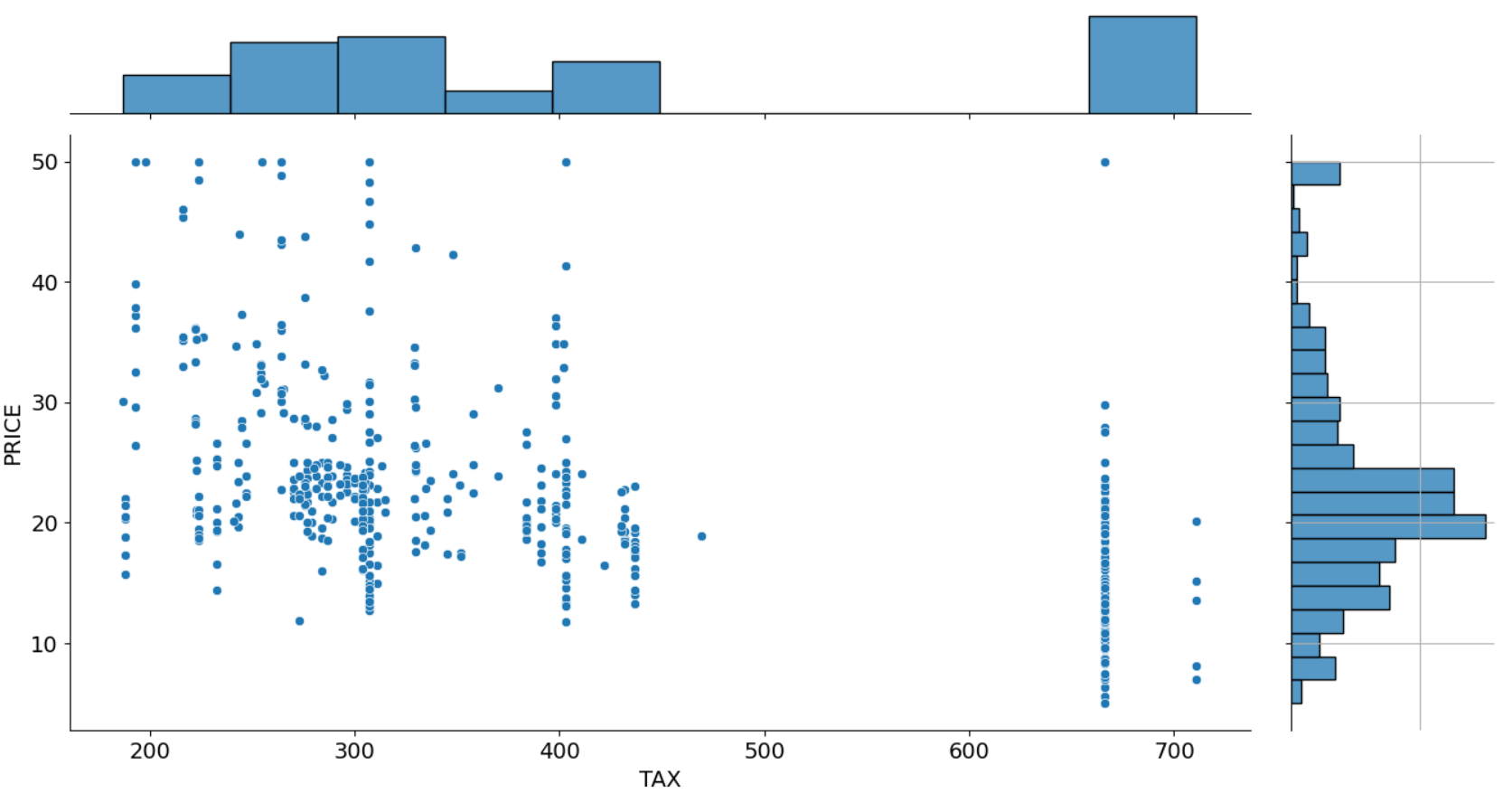


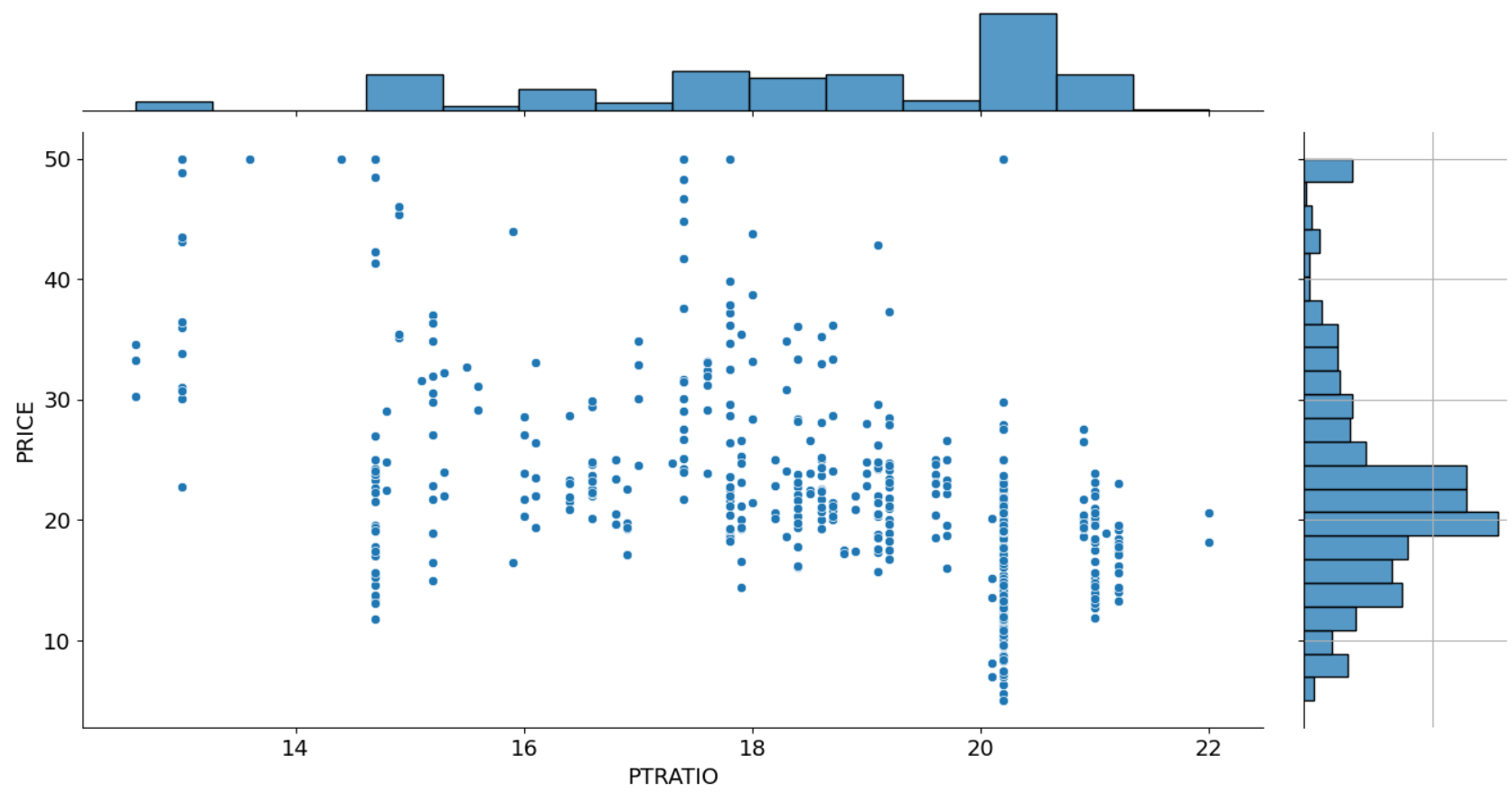


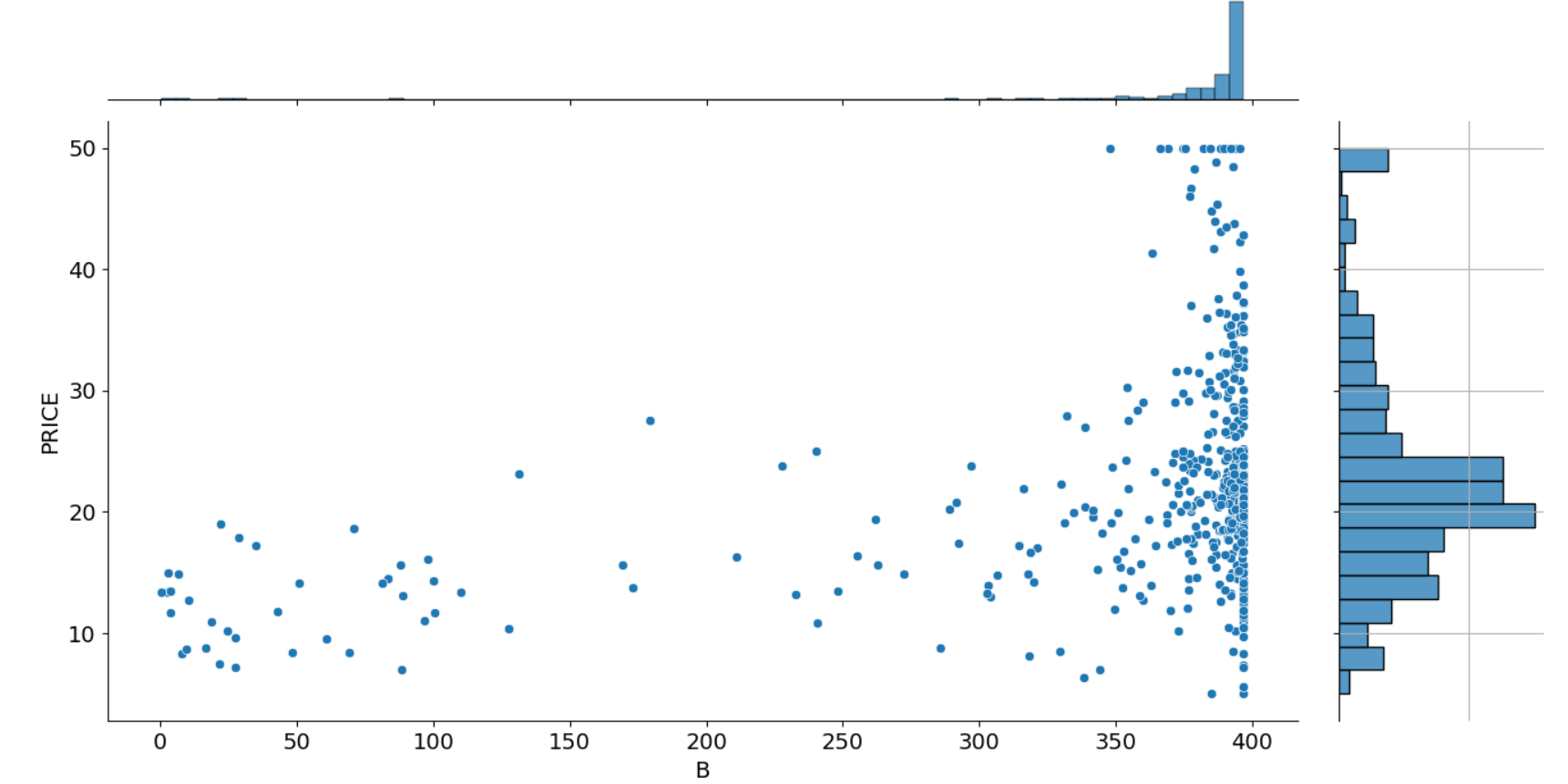


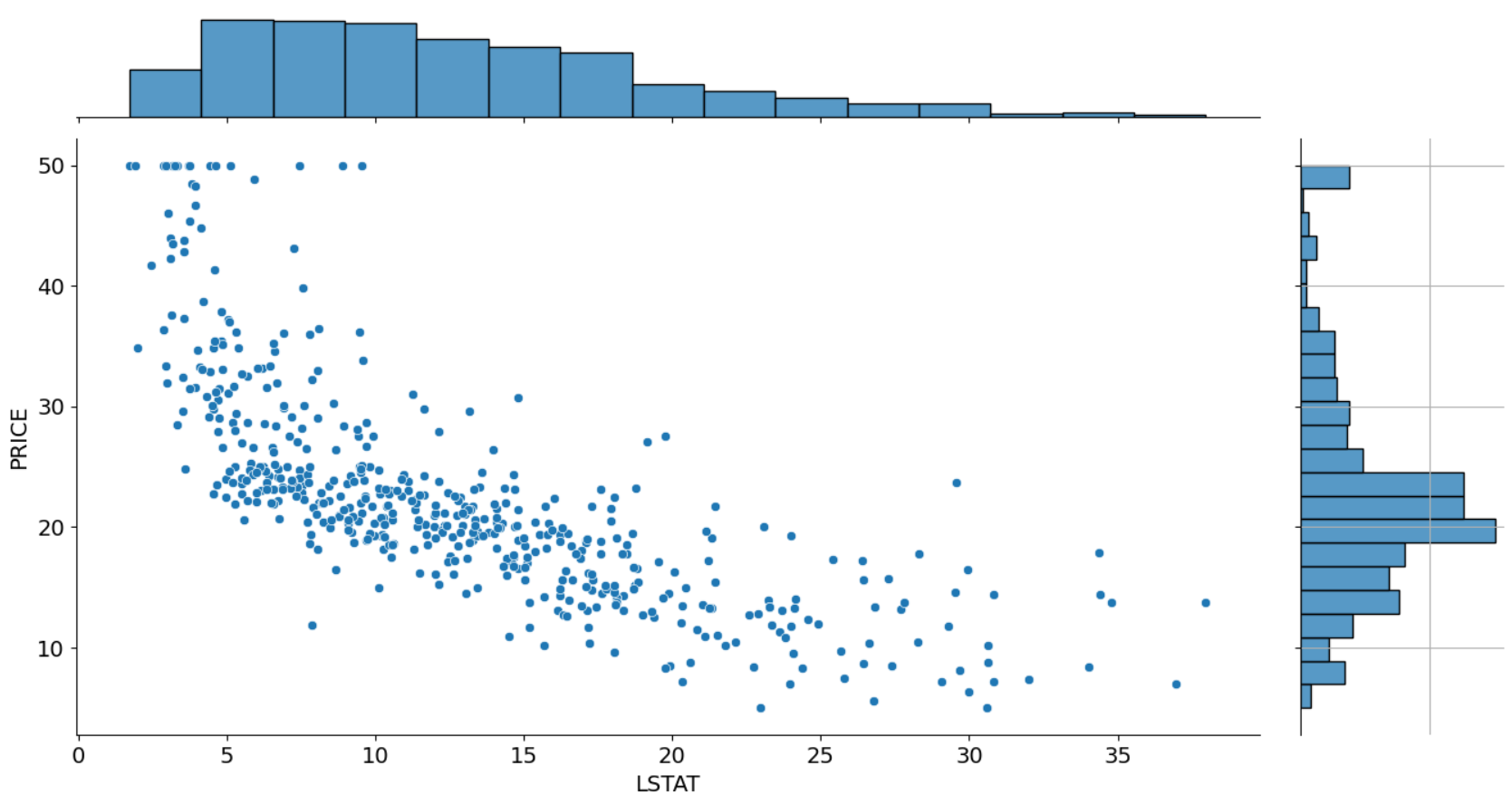


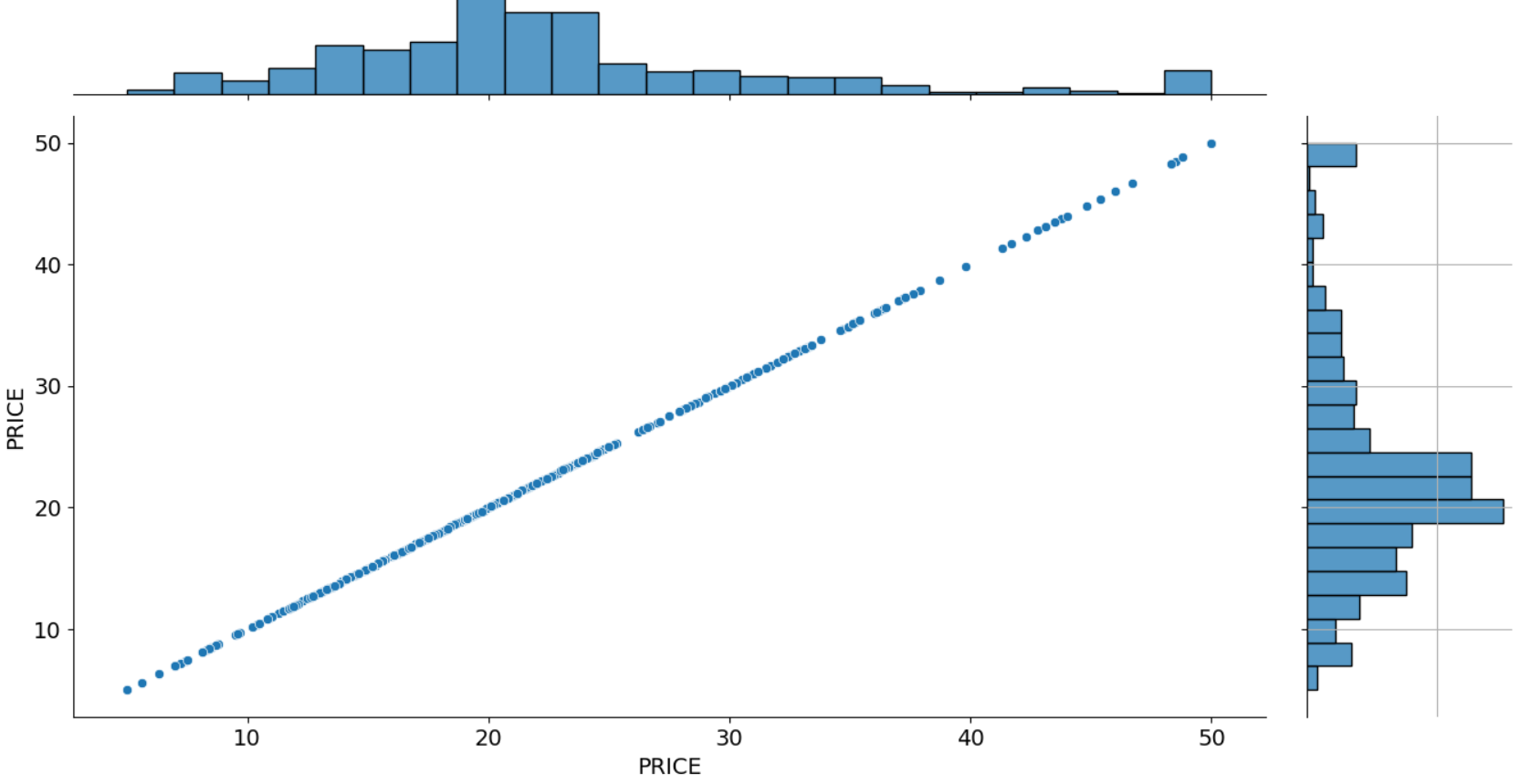












from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

reg = LinearRegression()

scaler.fit(X\_train)

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

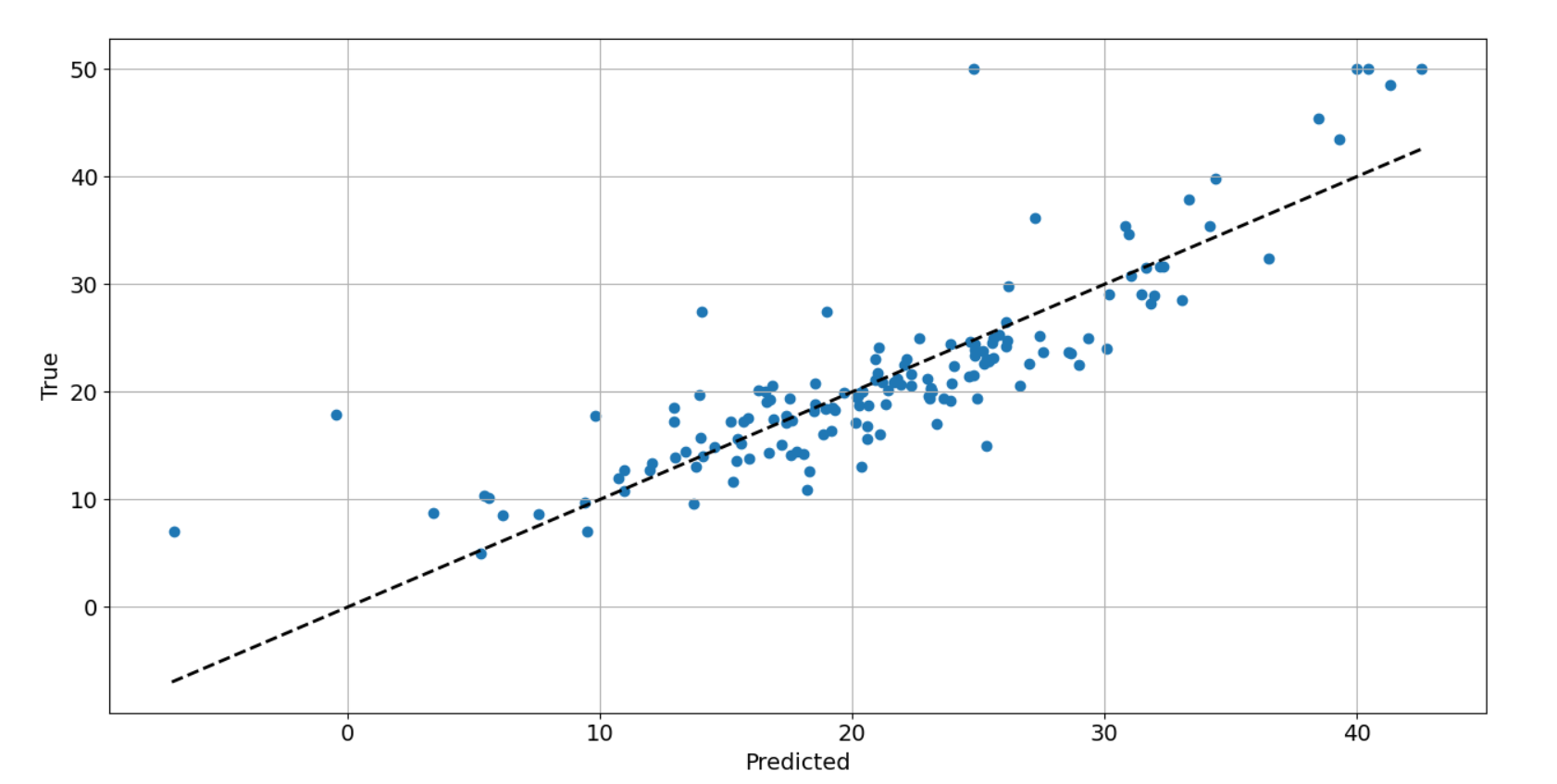
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

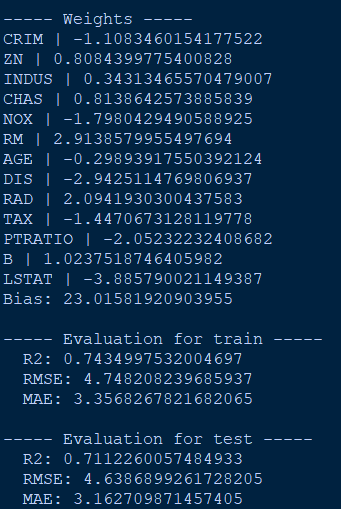
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

reg = LinearRegression()

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

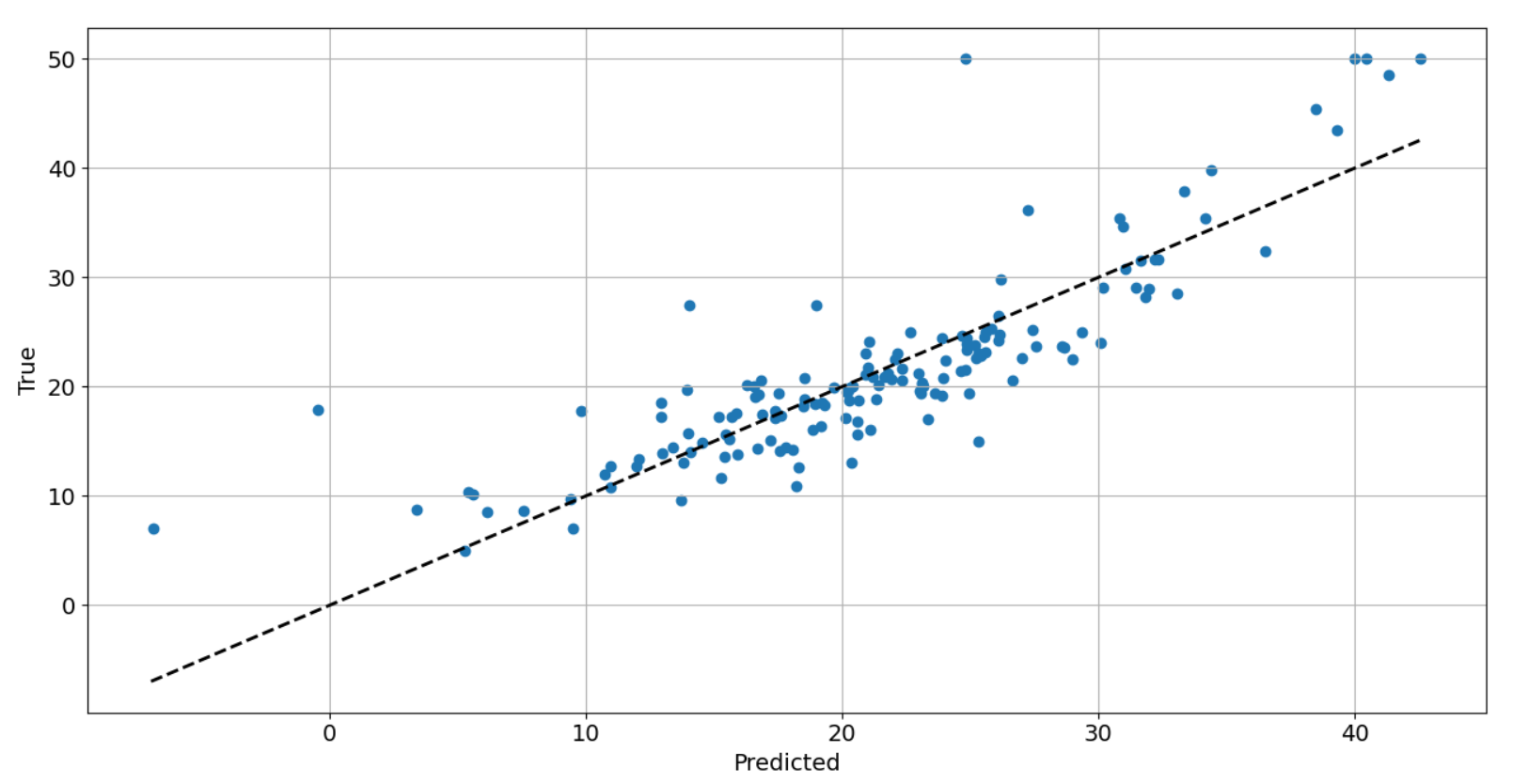
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

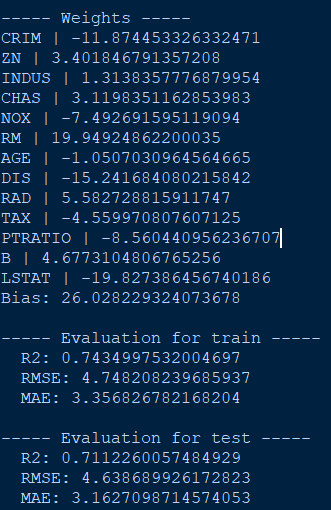
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.preprocessing import PowerTransformer

pt=PowerTransformer()

reg = LinearRegression()

pt.fit(X\_train)

X\_train\_scaled = pt.transform(X\_train)

X\_test\_scaled = pt.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

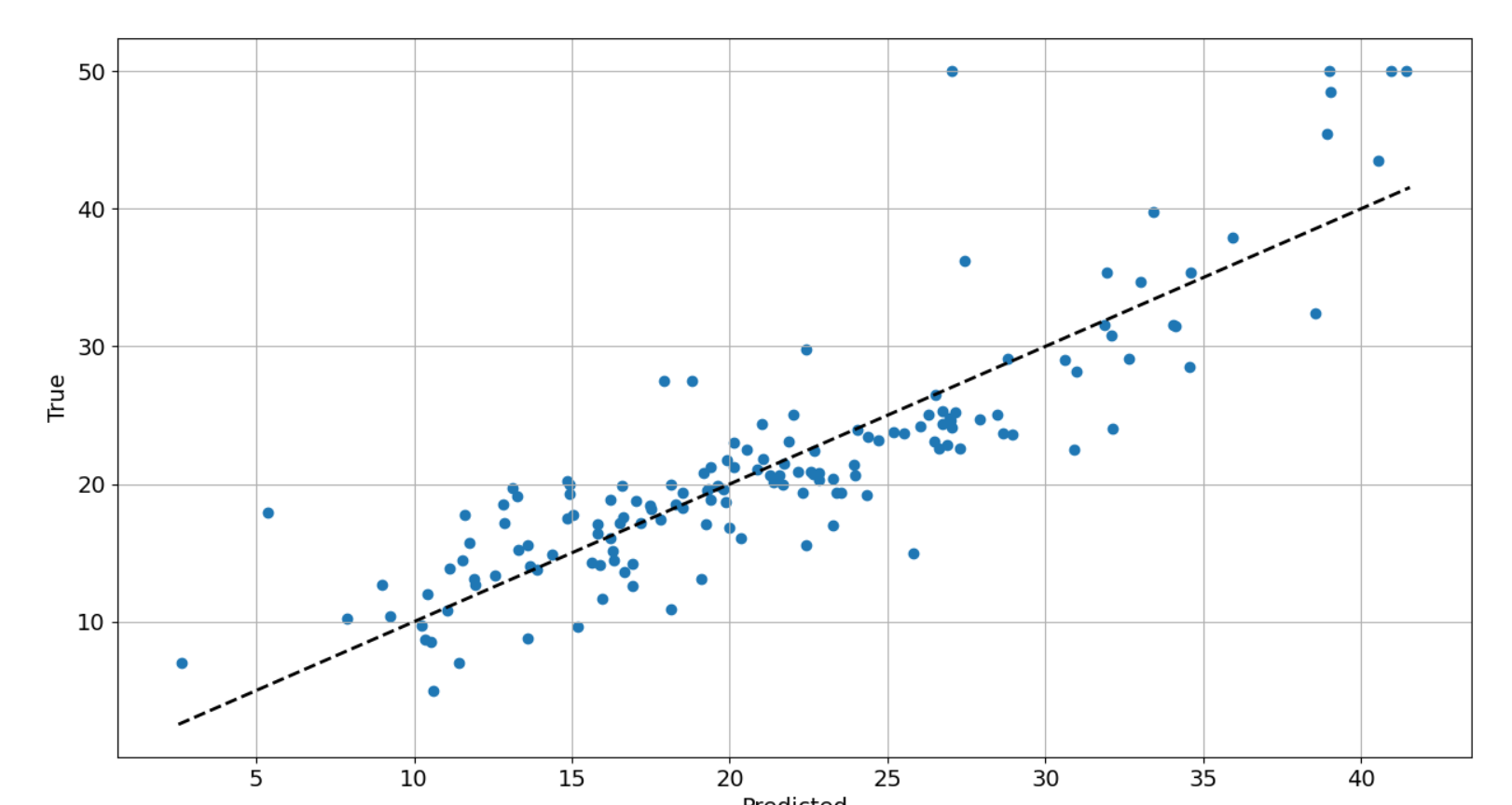
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

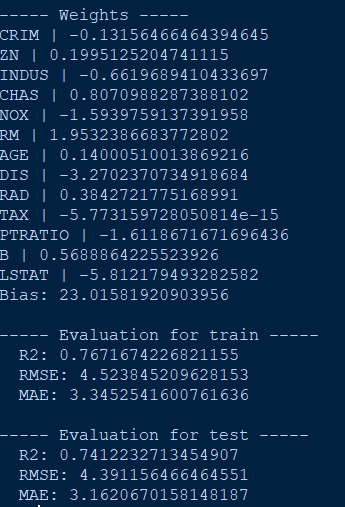
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.preprocessing import QuantileTransformer

pt=QuantileTransformer(output\_distribution='normal')

reg = LinearRegression()

pt.fit(X\_train)

X\_train\_scaled = pt.transform(X\_train)

X\_test\_scaled = pt.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

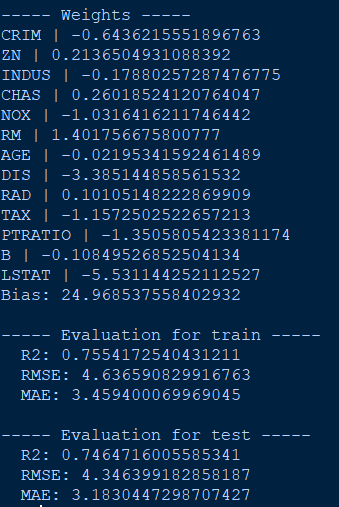
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

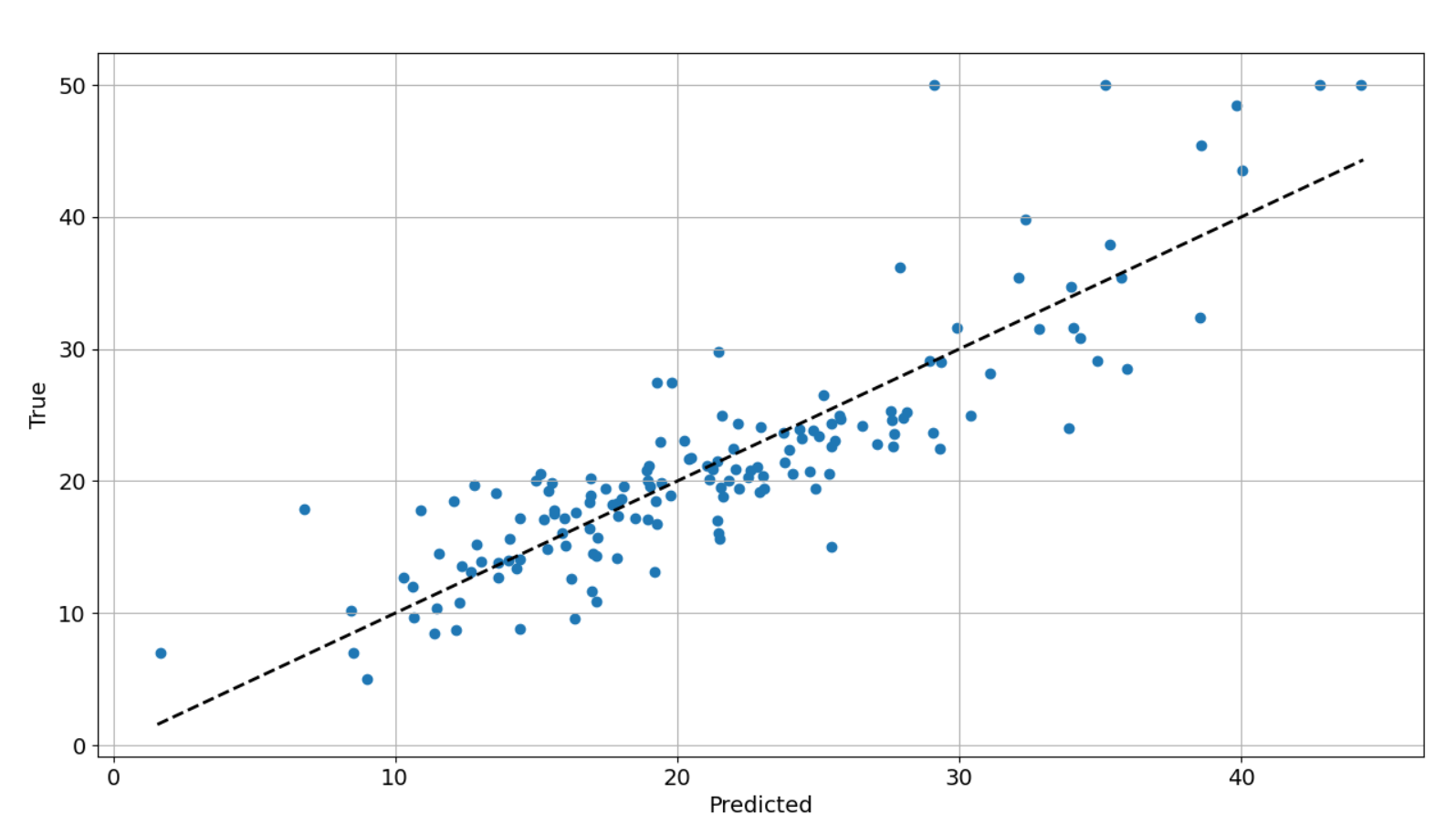
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





counter=df[df['PRICE']==50].index

df1=df.drop(counter,axis=0)

TRAIN\_RATIO = 0.7

X = df1[feature\_names]

y = df1['PRICE']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

train\_size=TRAIN\_RATIO,

random\_state=RANDOM\_STATE

)

reg.fit(X\_train, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train, y\_train, 'train')

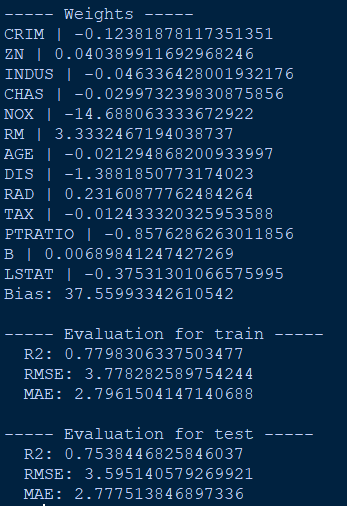
evaluate\_regressor(reg, X\_test, y\_test, 'test')

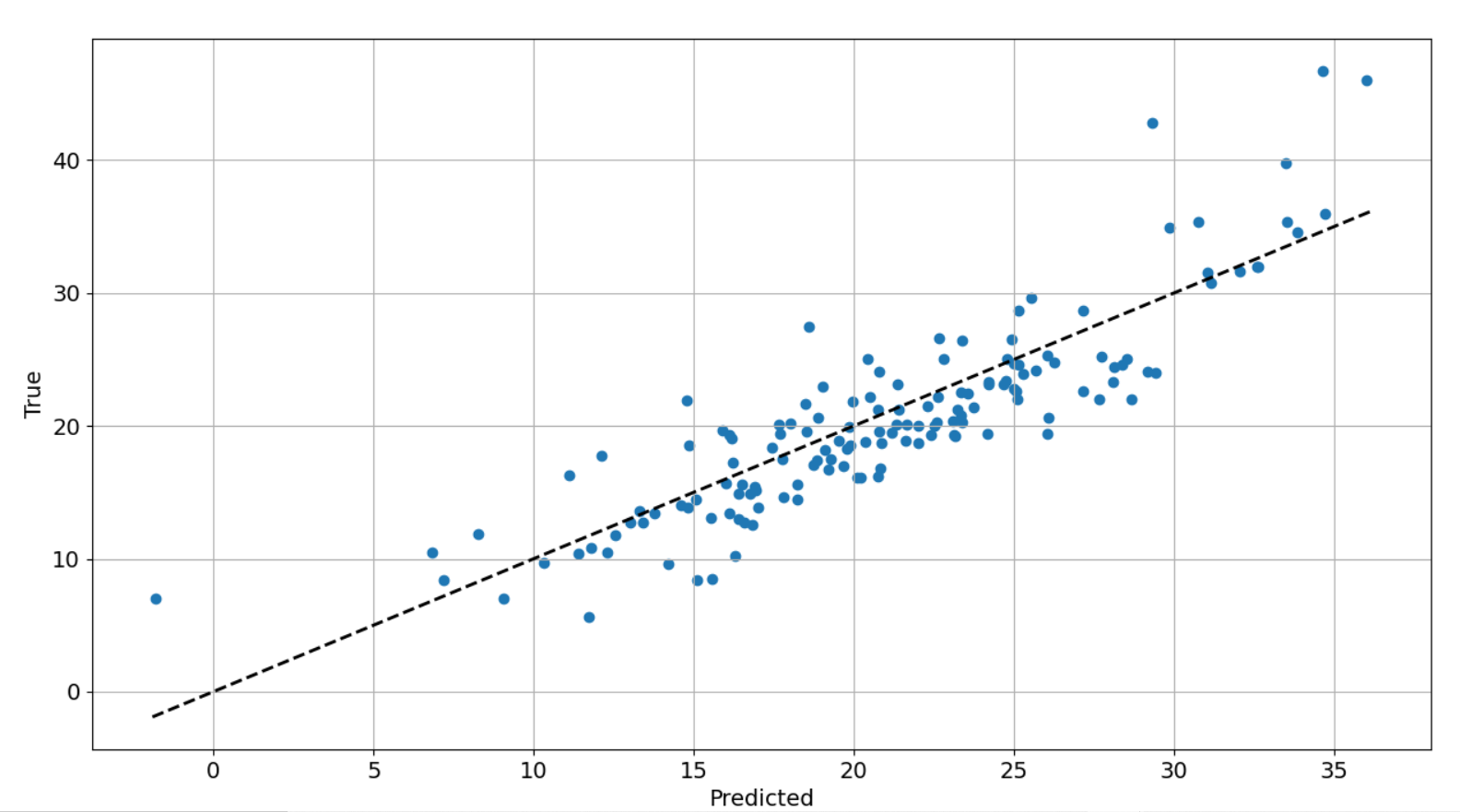
y\_pred = reg.predict(X\_test)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





def evaluate\_regressor1(reg, X, y):

y\_pred = reg.predict(X)

r2\_score\_value = r2\_score(y, y\_pred)

rmse\_score = np.sqrt(mean\_squared\_error(y, y\_pred))

mae\_score = mean\_absolute\_error(y, y\_pred)

return r2\_score\_value, rmse\_score, mae\_score

new\_features=['CRIM','ZN','INDUS','CHAS','NOX','RM','AGE','DIS','RAD','TAX','PTRATIO','B','LSTAT']

y=df['PRICE']

data\_frame=pd.DataFrame(columns=['R2\_train','RMSE\_train','MAE\_train','R2\_test','RMSE\_test','MAE\_test'])

for i in new\_features:

X=df.drop(columns=[i,'PRICE'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

train\_size=TRAIN\_RATIO,

random\_state=RANDOM\_STATE)

reg=LinearRegression()

reg.fit(X\_train, y\_train)

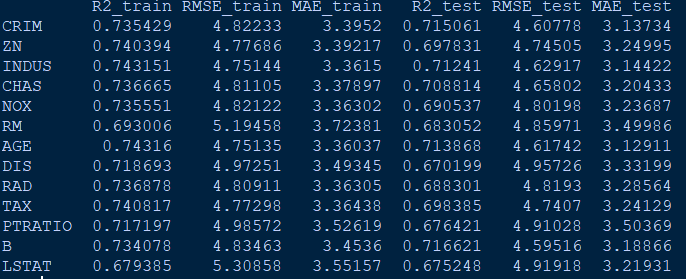
y\_pred = reg.predict(X\_test)

data\_frame.loc[i,['R2\_train','RMSE\_train','MAE\_train']] = evaluate\_regressor1(reg, X\_train, y\_train)

data\_frame.loc[i,['R2\_test','RMSE\_test','MAE\_test']] = evaluate\_regressor1(reg, X\_test, y\_test)

print(data\_frame)

Результат



from sklearn.preprocessing import PolynomialFeatures

TRAIN\_RATIO = 0.7

X = df[feature\_names]

y = df['PRICE']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=TRAIN\_RATIO, random\_state=RANDOM\_STATE)

pf = PolynomialFeatures(2)

reg = LinearRegression();

X\_train\_new=np.array(X\_train['LSTAT']).reshape(-1,1)

X\_test\_new=np.array(X\_test['LSTAT']).reshape(-1,1)

pf.fit(X\_train\_new)

X\_train\_new=np.concatenate((np.array(X\_train.drop(columns=['LSTAT'])),X\_train\_new),axis=1)

X\_test\_new=np.concatenate((np.array(X\_test.drop(columns=['LSTAT'])),X\_test\_new),axis=1)

reg.fit(X\_train\_new, y\_train)

evaluate\_regressor(reg, X\_train\_new, y\_train, 'train')

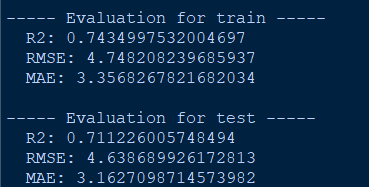
evaluate\_regressor(reg, X\_test\_new, y\_test, 'test')

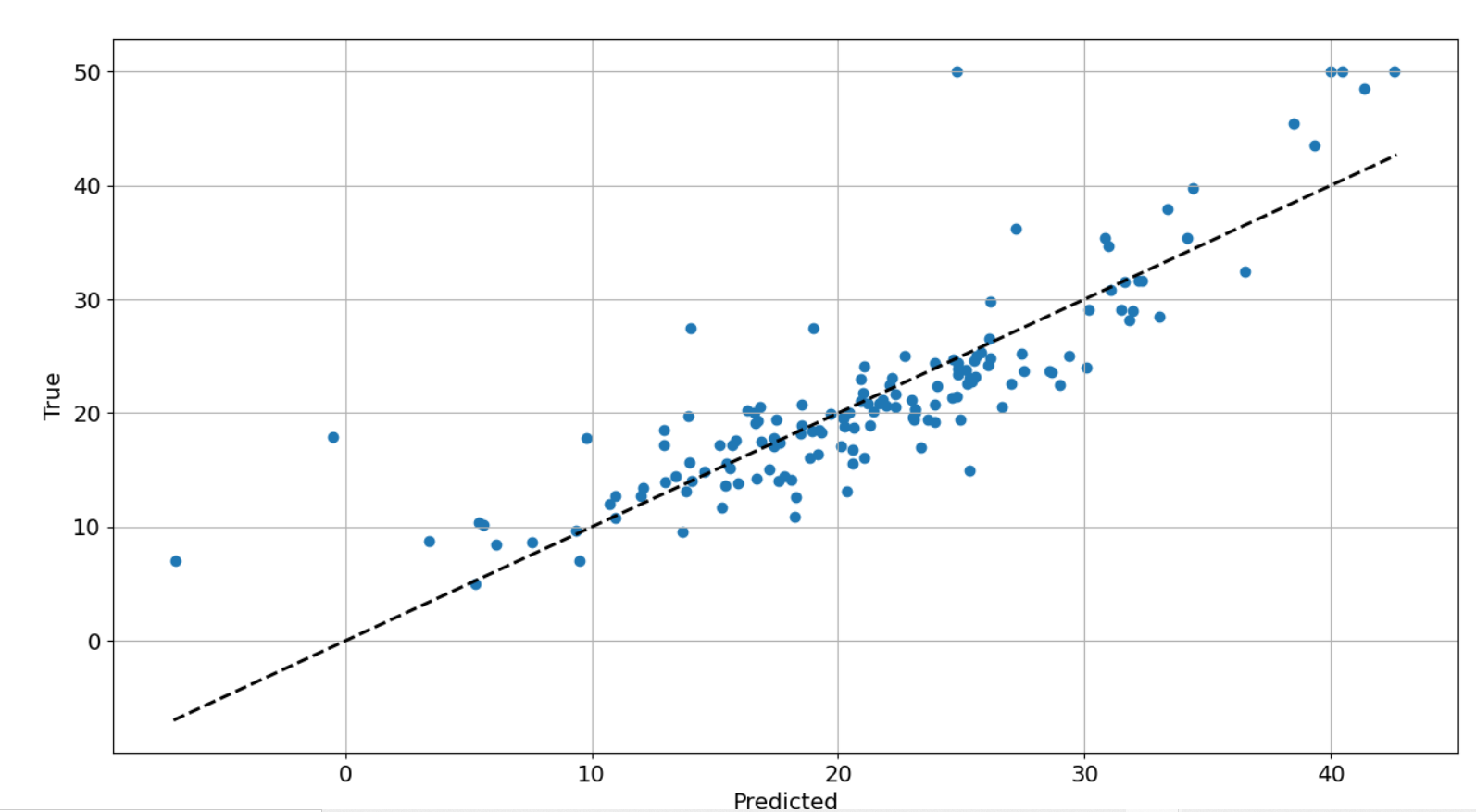
y\_pred = reg.predict(X\_test\_new)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.metrics import r2\_score

def R2\_metric (reg, X, y, mode):

y\_pred = rd.predict(X)

r2= r2\_score(y, y\_pred)

return r2

from sklearn.linear\_model import Ridge

a=np.logspace(-1,2,50)

rdge=[]

for i in a:

rd=Ridge(alpha=i)

rd.fit(X\_train, y\_train)

y\_pred=rd.predict(X\_test)

rdge.append(R2\_metric(reg,X\_train,y\_train,'train'))

plt.plot(a,rdge)

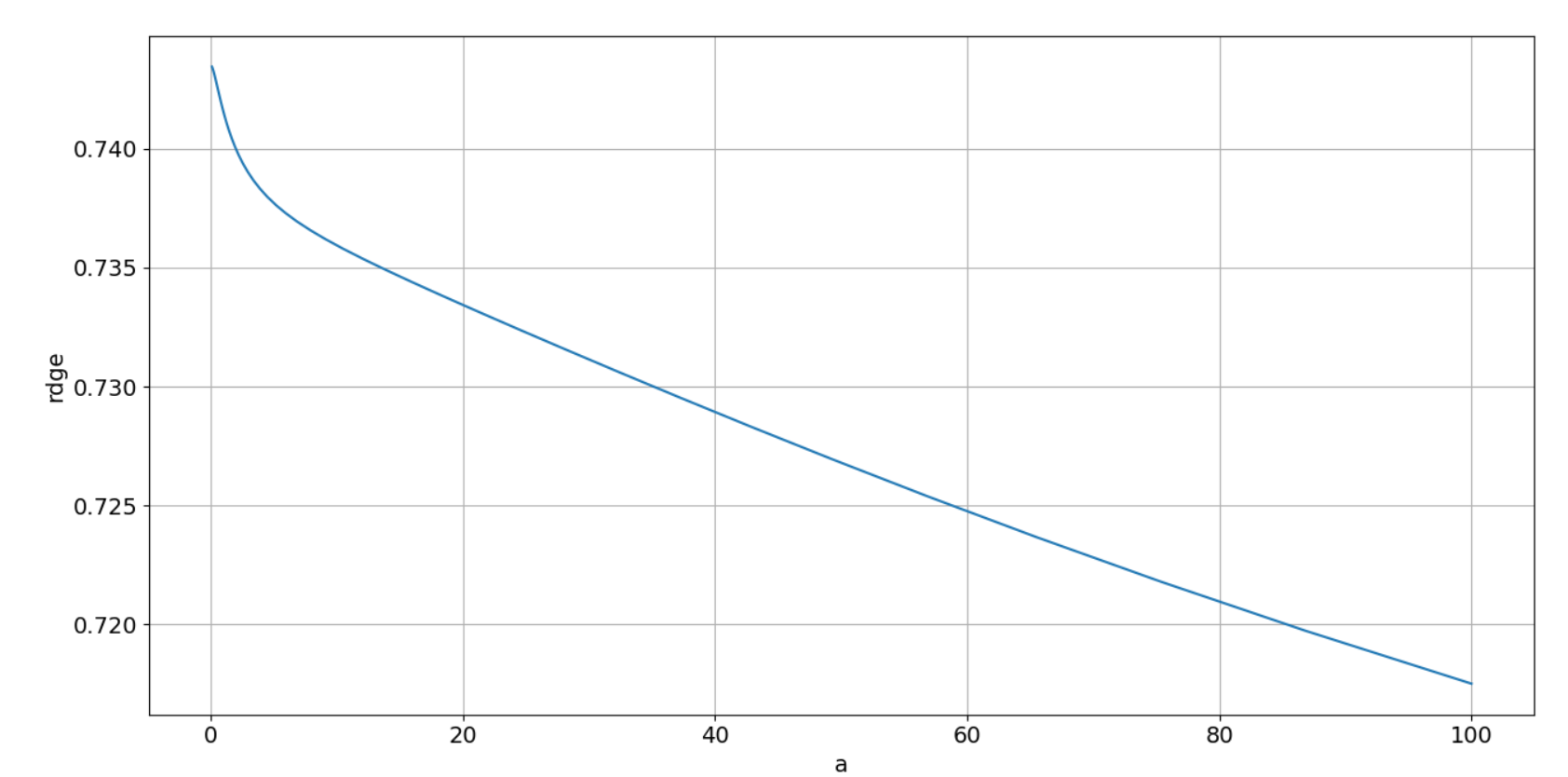
plt.xlabel('a')

plt.ylabel('rdge')

plt.grid()

plt.show()

Результат



from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

reg = Ridge(alpha=1.0)

scaler.fit(X\_train)

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

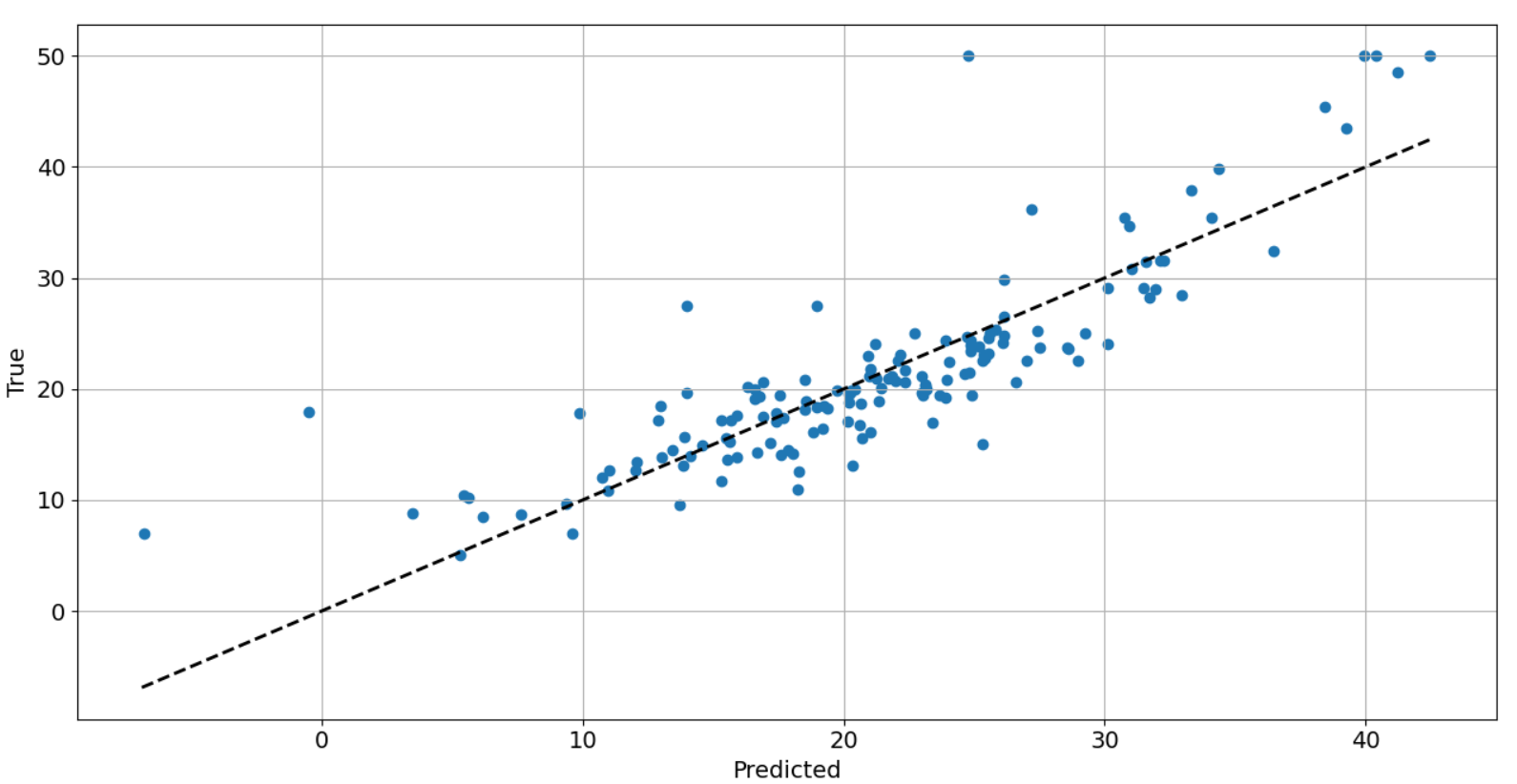
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

reg =Ridge()

scaler.fit(X\_train)

X\_train\_scaled = scaler.transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

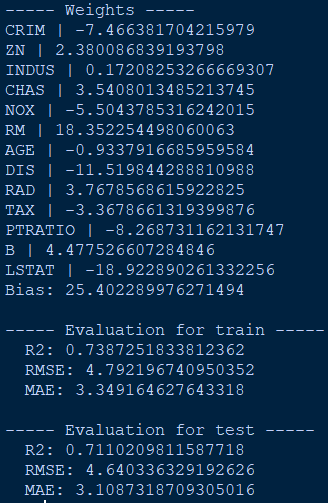
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

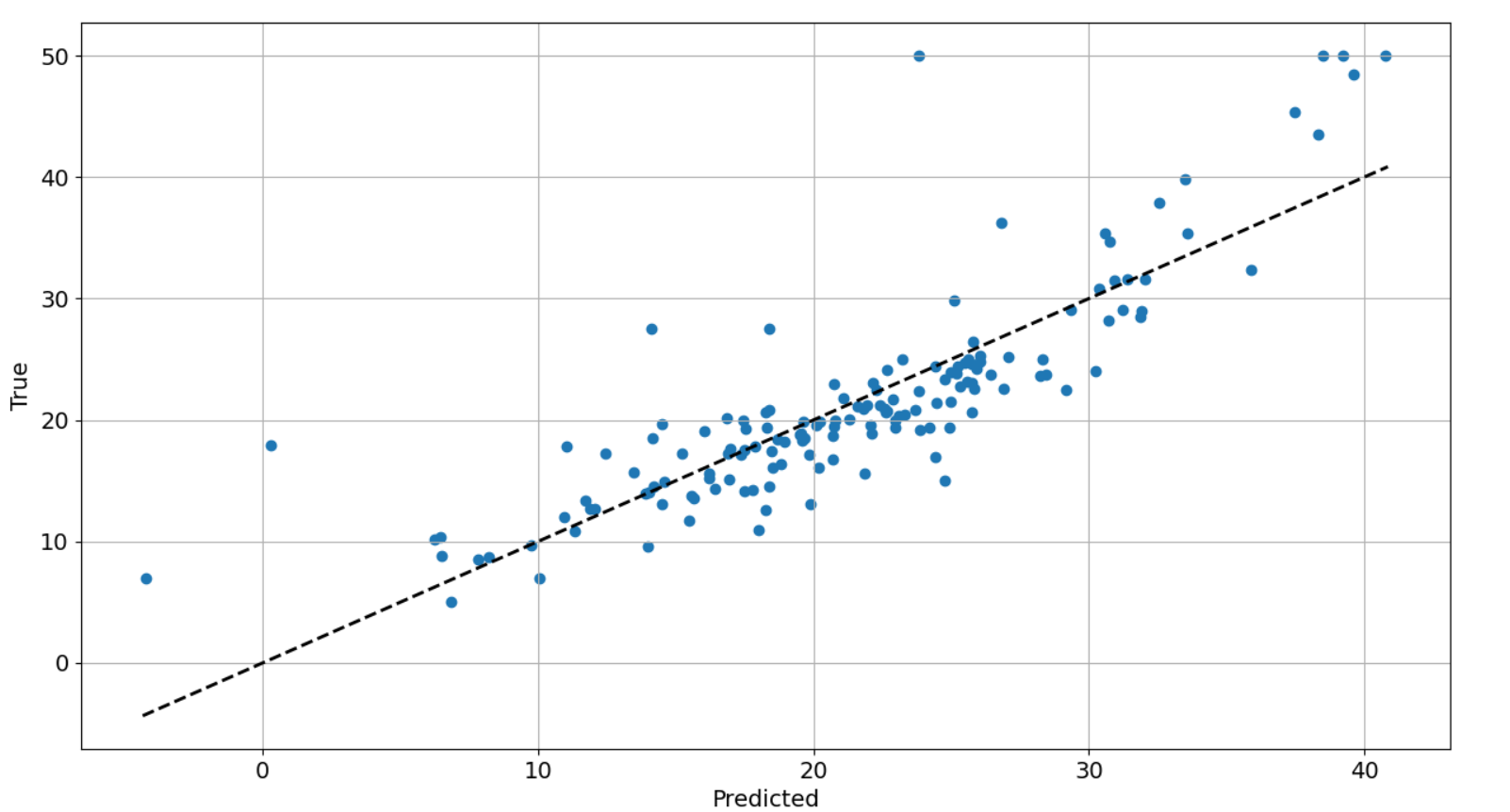
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.preprocessing import PowerTransformer

pt=PowerTransformer()

reg =Ridge()

pt.fit(X\_train)

X\_train\_scaled = pt.transform(X\_train)

X\_test\_scaled = pt.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

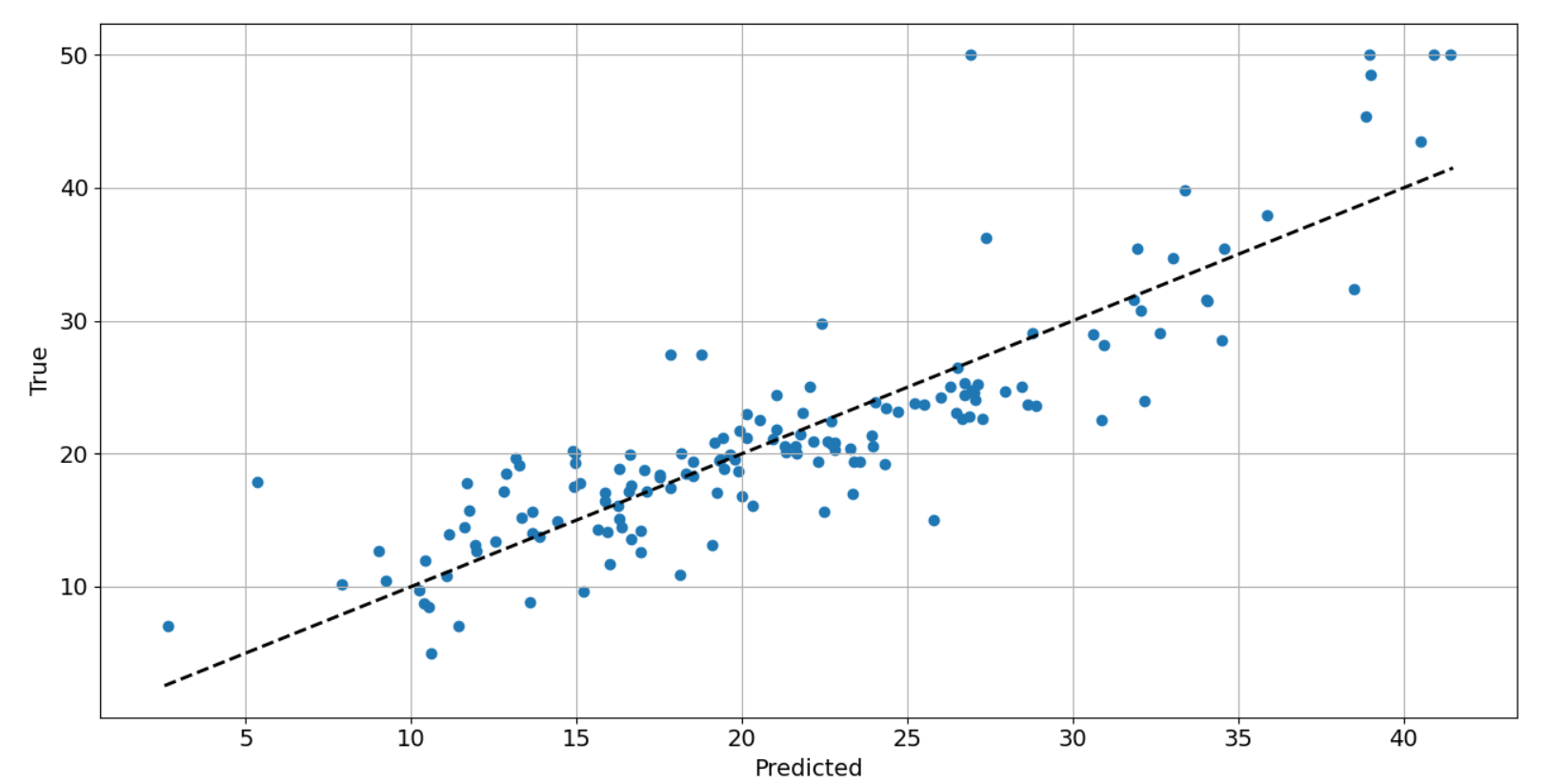
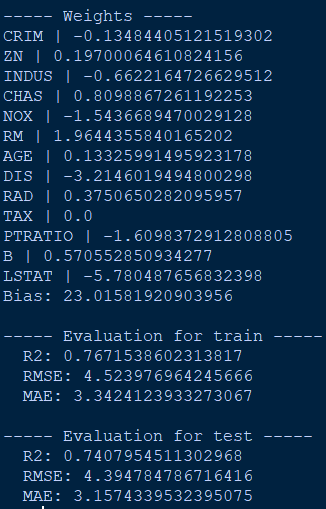
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат



from sklearn.preprocessing import QuantileTransformer

pt=QuantileTransformer(output\_distribution='normal')

reg = Ridge()

pt.fit(X\_train)

X\_train\_scaled = pt.transform(X\_train)

X\_test\_scaled = pt.transform(X\_test)

reg.fit(X\_train\_scaled, y\_train)

show\_linear\_model\_weights(reg, X\_train.columns)

evaluate\_regressor(reg, X\_train\_scaled, y\_train, 'train')

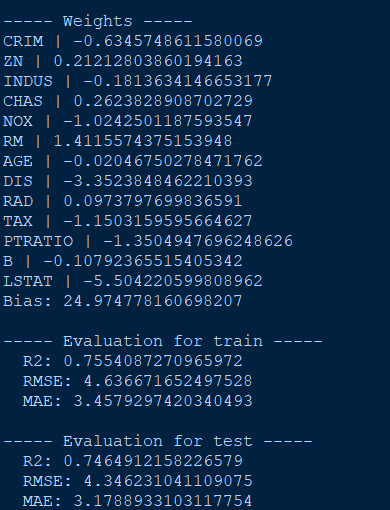
evaluate\_regressor(reg, X\_test\_scaled, y\_test, 'test')

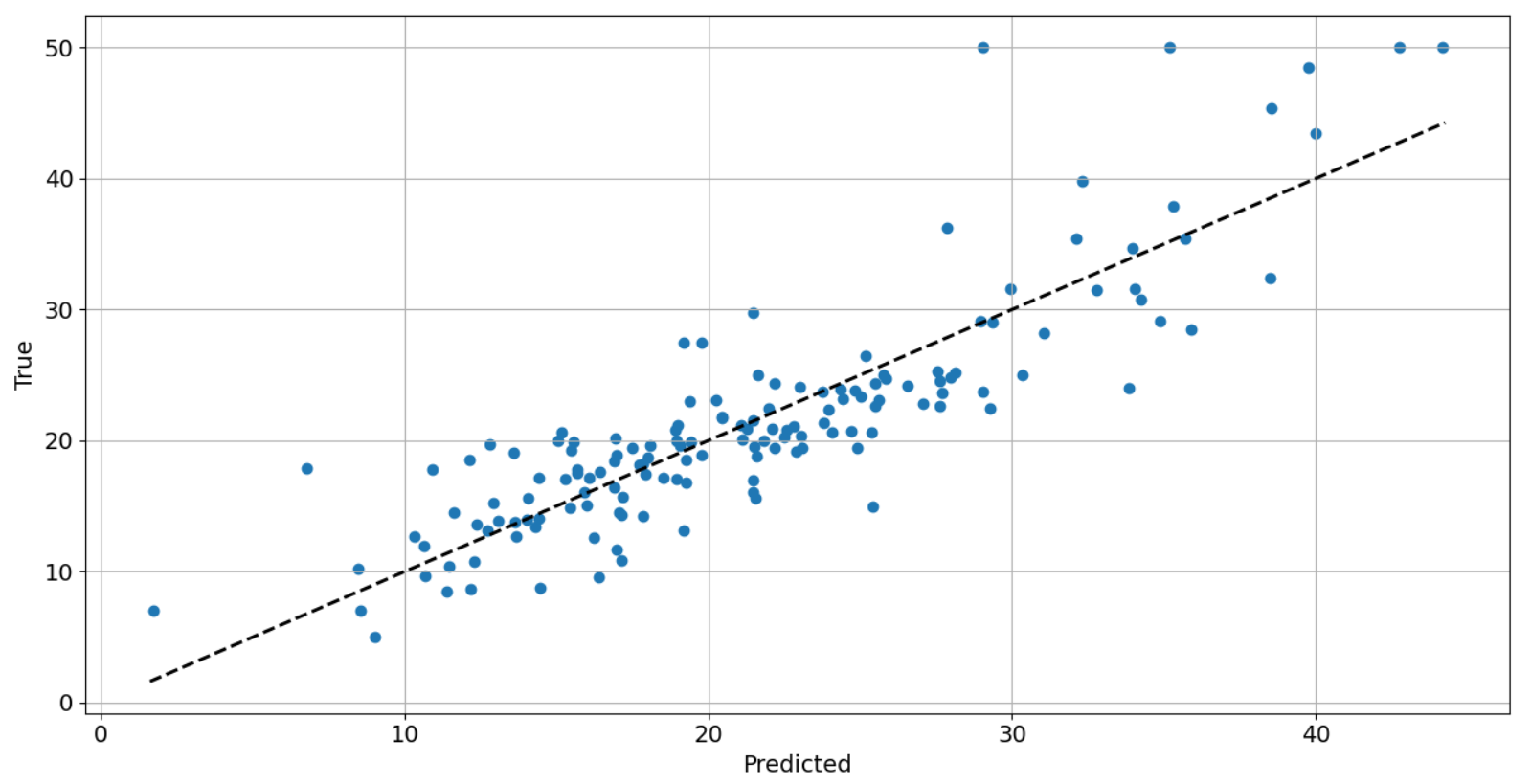
y\_pred = reg.predict(X\_test\_scaled)

draw\_predictions\_plot(y\_pred, y\_test)

plt.show()

Результат





from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import Ridge

from sklearn.preprocessing import PowerTransformer

counter=df[df['PRICE']==50].index

df1=df.drop(counter,axis=0)

TRAIN\_RATIO = 0.7

X = df1[feature\_names]

y = df1['PRICE']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

train\_size=TRAIN\_RATIO,

random\_state=RANDOM\_STATE

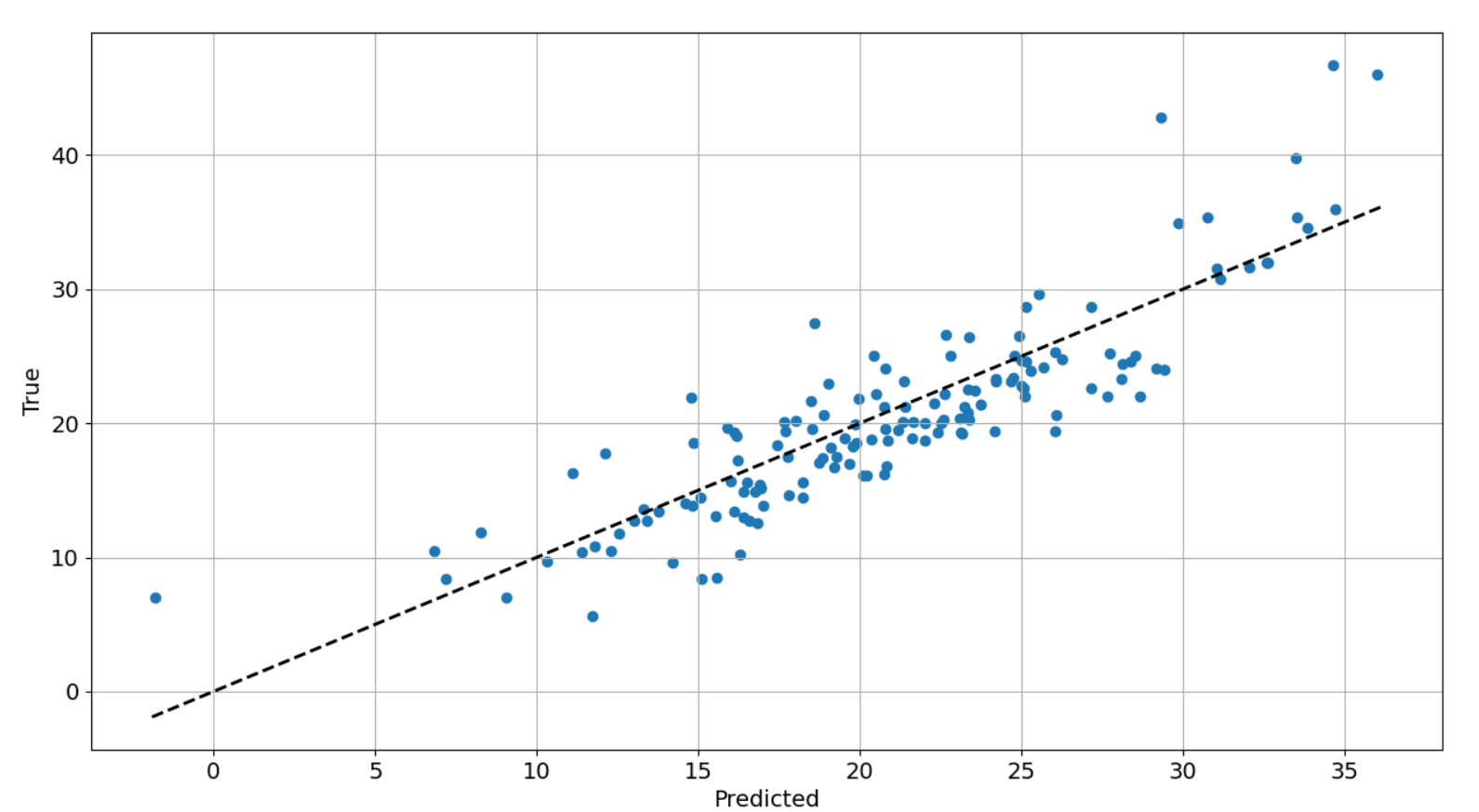
)

pf = PolynomialFeatures(2)

reg = Ridge(0.0001)

scaler=PowerTransformer()

Результат

 ****Выводы по настройке****

В результате создания лучшей модели, используя Ridge регрессию, трансформеры PowerTransformer и .PolynomialFeatures для генерации признака LSTAT в квадрате, а также исключая выбросы на уровне PRICE=50 получилось добиться наилучшего результата работы модели среди всех испытаний. получились следующие показатели: R2: 0.8100073250479757, RMSE: 3.5098165667888663 и MAE: 2.6335281765717444 для тренировочной выборки и R2: 0.7872966298697655, RMSE: 3.3419384860156516, MAE: 2.5694595533827016 для тестовой выборки. При использовании других методов нормализации результаты остаются такими же, соответственно можно выбрать любой.

****Ответы на вопросы****

Какую задачу выполняет QuantileTransformer?

Трансформер QuantileTransformer является одим из методов нормализации и выравнивания распределения, уменьшает количество выбросов. Использует информацию о квантилях.

За что отвечает аргумент конструктора output\_distribution класса QuantileTransformer?

output\_distribution отвечает за Предельное распределение для преобразованных данных. Возможно равномерное и нормальное распределение.

В чем отличие Ридж регрессии от линейной регрессии?

Ридж грегрессия это один из методов понижения размерности. Применяется для борьбы с избыточностью данных, когда независимые переменные коррелируют друг с другом, вследствие чего проявляется неустойчивость оценок коэффициентов многомерной линейной регрессии. Линейная регрессия неэффективна в случае высокой коллинеарности, у ридж регрессии этот недостаток исключен.

Что такое регуляризация?

Регуляризация это метод добавления некоторых дополнительных ограничений к условию с целью решить некорректно поставленную задачу или предотвратить переобучение.

Как сказывается станартизация данных на результатах обучения с использованием регуляризации?

При использовании StandartScaler практически не сказывается, при использовании MinMaxScaler наблюдается ухудшение

В каких случаях нужно применять StandardScaler, а в каких MinmaxScaler?

StandardScaler - когда хотим получить распределение со стандартным отклонением, равным 1.Т.е. масштабирование до единичной дисперсии. MinmaxScaler - трансформирует признаки в заданном диапазоне (исходный минимум и исходный максимум). После применения StandardScaler диапазон значений больше, чем после MinmaxScaler.

Как сказывается применение QuantileTransformer на результатах обучения с использованием регуляризации?

Наблюдается улучшение показателя R2 относительно использования StandartScaler и MinMaxScaler

Как влияет значение IMG_256 на результаты обучения Ридж регрессии?

Согласно полученному графику, чем ближе к 0 значение IMG_257, тем выше показатель R2