|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления

**Отчёты по лабораторной работе №5**

По дисциплине:

«Технологии машинного обучения»

Выполнил:

Студент группыИУ5-62 **\_\_\_ \_ \_ \_\_\_\_ \_ Ким Р.И. \_\_**

(Подпись, дата) (Фамилия И.О.)

Проверил:

**\_\_\_\_\_\_\_\_\_ \_ Гапанюк Ю. Е.\_**

(Подпись, дата) (Фамилия И.О.)

Москва, 2020

# Лабораторная работа №5

**Линейные модели, SVM и деревья решений**

## Цель лабораторной работы

Изучение линейных моделей, SVM и деревьев решений.

## Задание

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train\_test\_split разделите выборку на обучающую и тестовую.
4. Обучите следующие модели: одну из линейных моделей; SVM;

дерево решений.

1. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

## Дополнительные задания

Проведите эксперименты с важностью признаков в дереве решений. Визуализируйте дерево решений.

# Ход выполнения лабораторной работы

In [1]:

**import pandas as pd import seaborn as sns import numpy as np**

**from typing import** Tuple, Dict **import matplotlib.pyplot as plt from operator import** itemgetter

**from sklearn.model\_selection import** train\_test\_split **from sklearn.linear\_model import** LinearRegression **from sklearn.preprocessing import** StandardScaler

**from sklearn.metrics import** f1\_score, r2\_score, mean\_squared\_error, mean\_absolute\_error, accuracy\_s core, precision\_score

**from sklearn.svm import** LinearSVR, SVR

**from sklearn.tree import** DecisionTreeRegressor, plot\_tree

%**matplotlib** inline sns.set(style="ticks")

In [2]:

data = pd.read\_csv('pulsar\_stars.csv')

In [3]:

data.head()

Out[3]:

**Mean of the integrated**

**profile**

**Standard deviation of the**

**integrated**

**profile**

**Excess kurtosis of the**

**integrated**

**profile**

**Skewness of**

**the integrated**

**profile**

**Mean of the DM- SNR curve**

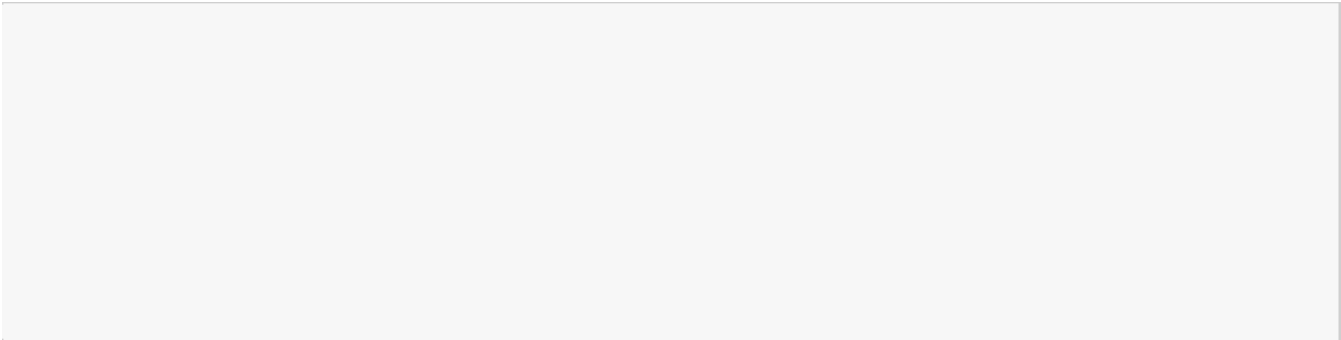
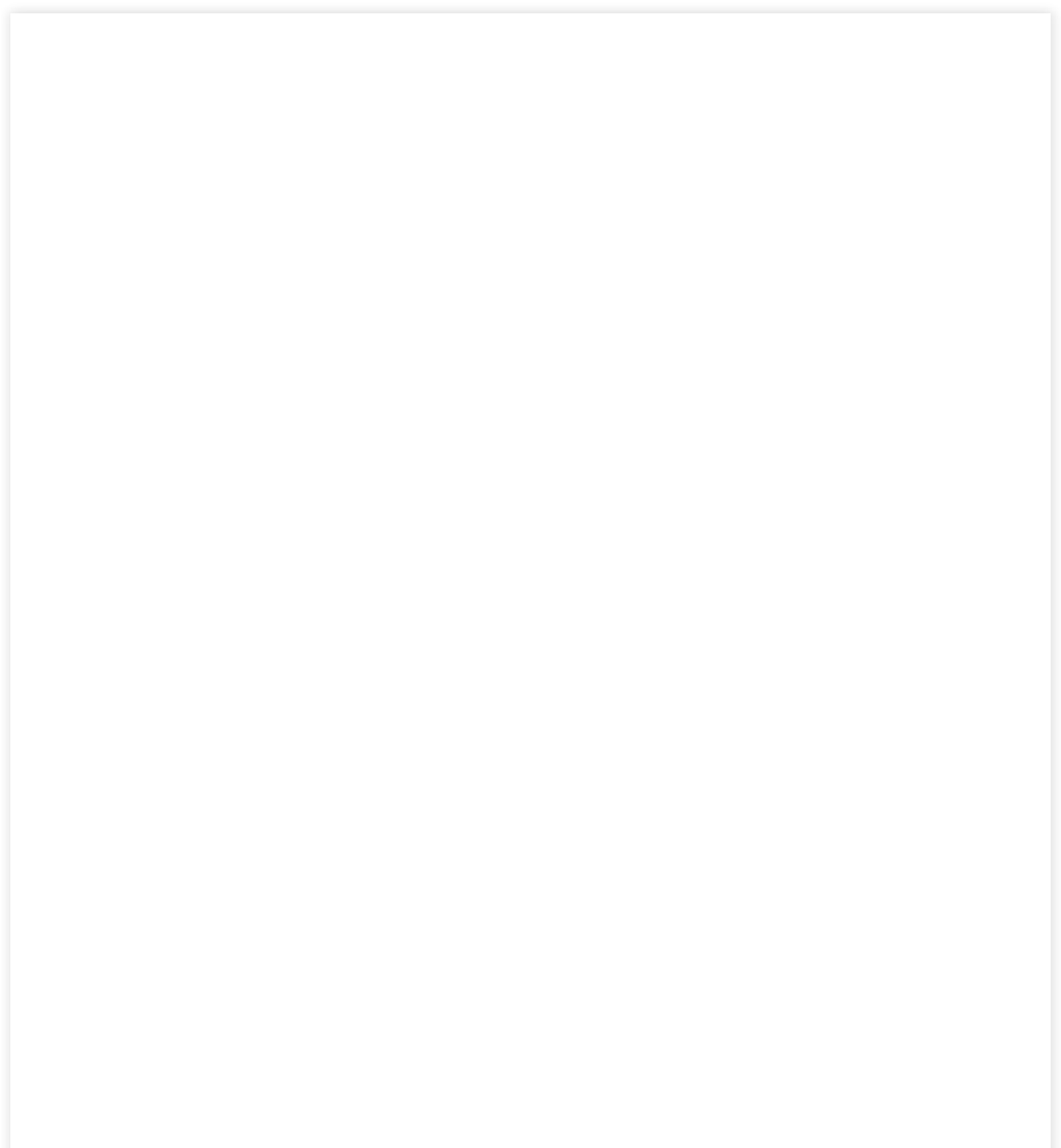
**Standard deviation of the DM-SNR curve**

**Excess kurtosis of the DM-SNR curve**

**Skewness of the DM-SNR**

**curve**

**target\_class**



**0** 140.562500 55.683782 -0.234571 -0.699648 3.199833 19.110426 7.975532 74.242225 0

**1** 102.507812 5**S**8**t**.8**a**8**n**2**d**4**a**3**rd**0 0.**E**46**x**5**c**3**e**1**s**8**s**

**Mean of the 2** 1**i**0**n**3**te**.0**g**1**r**5**a**6**te**2**d**5

**profile**

**deviation of the kurtosis of the**

**Mean of**

**Standard**

**Excess Skewness of**

**i**3**n**9**t**.**e**3**g**4**r**1**a**6**te**4**d**9

**in**0**te**.3**g**2**r**3**a**3**te**2**d**8

**in**1**te**.0**g**5**r**1**a**1**te**6**d**4

**the**

3**t**.**h**12**e**1**D**2**M**37**- deviat**2**io**1**n**.74**o**4**f** 6**th**6**e**9 **kurtos**7**i**.**s**73**o**5**f** 8**th**2**e**2 **the**63**D**.1**M**7**-**1**S**9**N**0**R**9 **target\_clas**0**s**

**SNR curve DM-SNR curve DM-SNR curve curve**

**Ske**-**w**0.**n**5**e**1**s**5**s**08**o**8**f**

1.677258 14.860146 10.576487 127.393580 0



57.178449 -0.068415 -0.636238

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **3** | 136.750000 | **profile** | **profile** | **profile** | 3.642977 | 20.959280 | 6.896499 | 53.593661 | 0 |
|  |  |  |  |  |  |  |  |  |  |
| **4** | 88.726562 | 40.672225 | 0.600866 | 1.123492 | 1.178930 | 11.468720 | 14.269573 | 252.567306 | 0 |

In [4]:



data.isnull().sum()

Out[4]:

Mean of the integrated profile 0

Standard deviation of the integrated profile 0

Excess kurtosis of the integrated profile 0

Skewness of the integrated profile 0

Mean of the DM-SNR curve 0

Standard deviation of the DM-SNR curve 0

Excess kurtosis of the DM-SNR curve 0

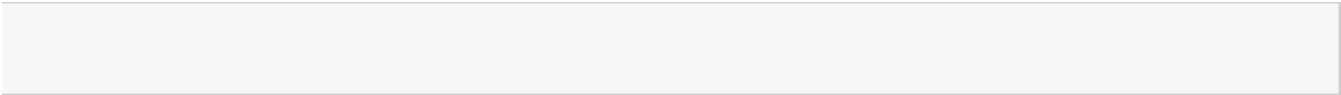
Skewness of the DM-SNR curve 0

target\_class 0

dtype: int64

Таким образом, пропуски данных отсутствуют. Разделим выборку на обучающую и тестовую:

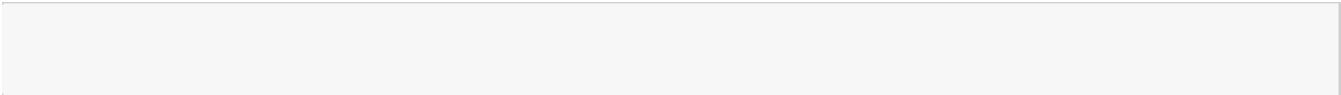
In [5]:



*# Разделим данные на целевой столбец и признаки*

X = data.drop("target\_class", axis=1) Y = data["target\_class"]

In [6]:



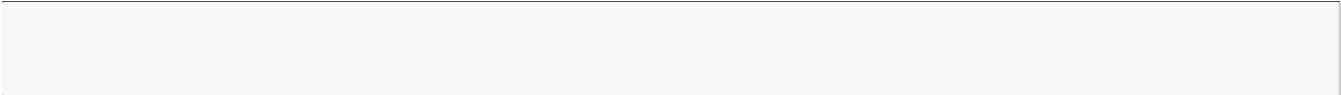
*# С использованием метода train\_test\_split разделим выборку на обучающую и тестовую* X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.25, random\_state=1) X\_train.shape, X\_test.shape, Y\_train.shape, Y\_test.shape

Out[6]:

((13423, 8), (4475, 8), (13423,), (4475,))

## Линейная модель

In [7]:

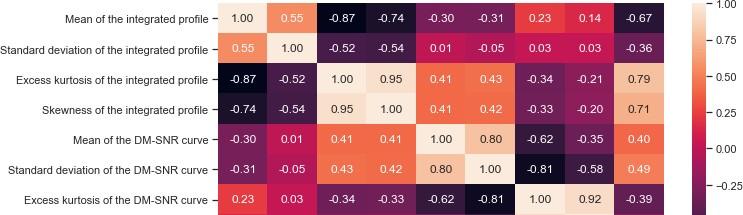


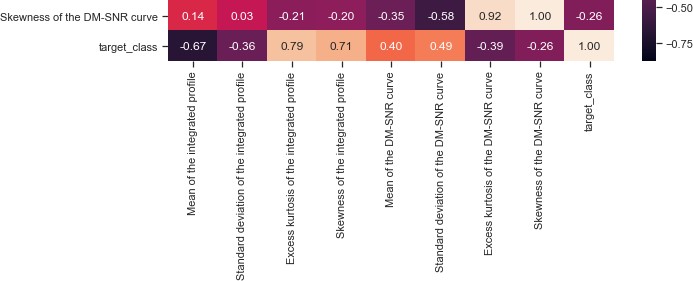
*#Построим корреляционную матрицу*

fig, ax = plt.subplots(figsize=(10,5)) sns.heatmap(data.corr(method='pearson'), ax=ax, annot=**True**, fmt='.2f')

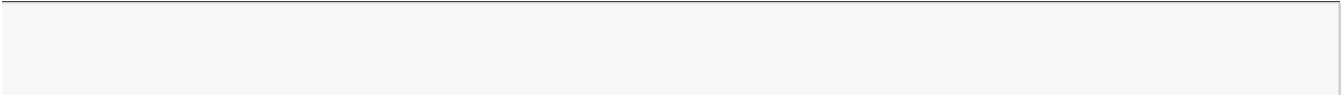
Out[7]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x11463670>





In [8]:

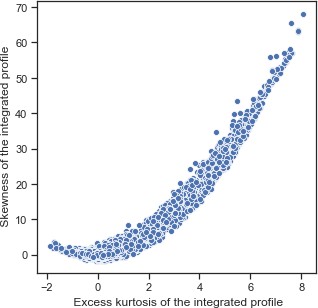


fig, ax = plt.subplots(figsize=(5,5))

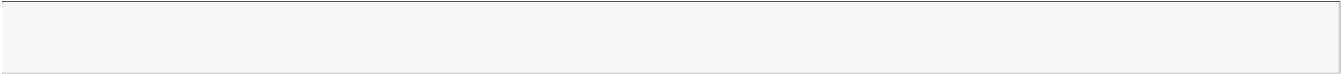
sns.scatterplot(ax=ax, x=' Excess kurtosis of the integrated profile', y=' Skewness of the integrated profile', data=data)

Out[8]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x12898370>

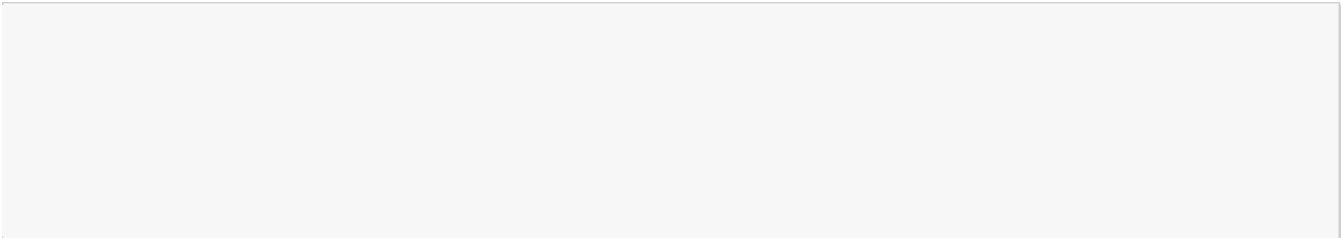


In [9]:



x\_array = data[' Excess kurtosis of the integrated profile'].values y\_array = data[' Skewness of the integrated profile'].values

In [10]:



*# Аналитическое вычисление коэффициентов регрессии*

**def** analytic\_regr\_coef(x\_array : np.ndarray,

y\_array : np.ndarray) -> Tuple[float, float]: x\_mean = np.mean(x\_array)

y\_mean = np.mean(y\_array)

var1 = np.sum([(x-x\_mean)\*\*2 **for** x **in** x\_array])

cov1 = np.sum([(x-x\_mean)\*(y-x\_mean) **for** x, y **in** zip(x\_array, y\_array)]) b1 = cov1 / var1

b0 = y\_mean - b1\*x\_mean

**return** b0, b1

In [11]:

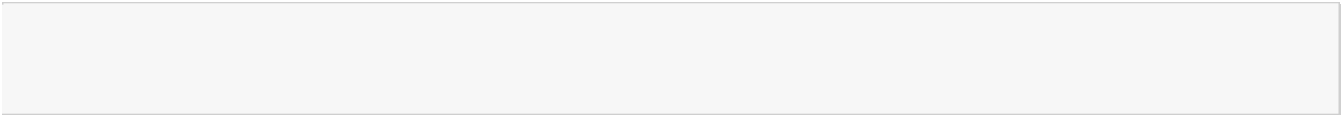


b0, b1 = analytic\_regr\_coef(x\_array, y\_array) b0, b1

Out[11]:

(-0.8493834635342365, 5.482102483932346)

In [12]:



*# Вычисление значений y на основе x для регрессии*

**def** y\_regr(x\_array : np.ndarray, b0: float, b1: float) -> np.ndarray: res = [b1\*x+b0 **for** x **in** x\_array]

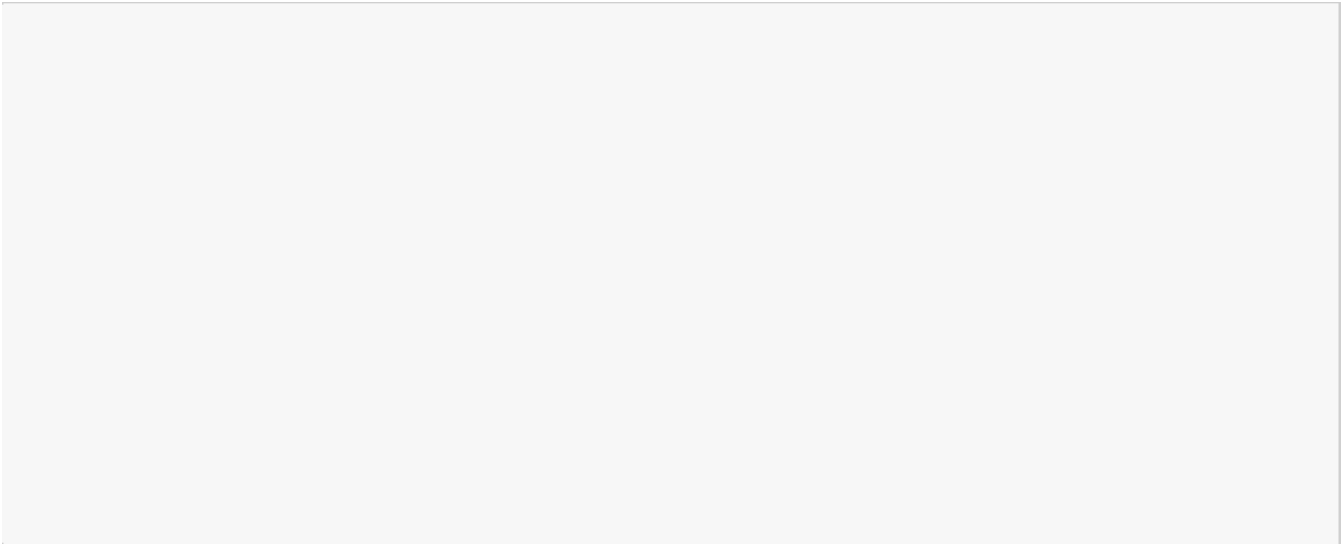
**return** res

In [13]:



y\_array\_regr = y\_regr(x\_array, b0, b1)

In [14]:



*# Простейшая реализация градиентного спуска*

**def** gradient\_descent(x\_array : np.ndarray,

y\_array : np.ndarray, b0\_0 : float,

b1\_0 : float, epochs : int,

learning\_rate : float = 0.001

) -> Tuple[float, float]: *# Значения для коэффициентов по умолчанию* b0, b1 = b0\_0, b1\_0

k = float(len(x\_array))

**for** i **in** range(epochs):

*# Вычисление новых предсказанных значений*

*# используется векторизованное умножение и сложение для вектора и константы*

y\_pred = b1 \* x\_array + b0

*# Расчет градиентов*

*# np.multiply - поэлементное умножение векторов*

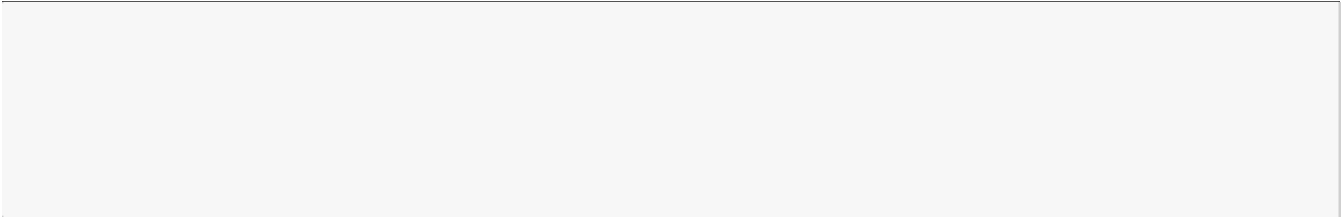
dL\_db1 = (-2/k) \* np.sum(np.multiply(x\_array, (y\_array - y\_pred))) dL\_db0 = (-2/k) \* np.sum(y\_array - y\_pred)

*# Изменение значений коэффициентов:*

b1 = b1 - learning\_rate \* dL\_db1 b0 = b0 - learning\_rate \* dL\_db0

*# Результирующие значения* y\_pred = b1 \* x\_array + b0 **return** b0, b1, y\_pred

In [15]:

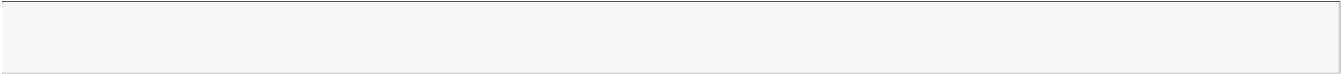


**def** show\_gradient\_descent(epochs, b0\_0, b1\_0):

grad\_b0, grad\_b1, grad\_y\_pred = gradient\_descent(x\_array, y\_array, b0\_0, b1\_0, epochs) print('b0 = **{}** - (теоретический), **{}** - (градиентный спуск)'.format(b0, grad\_b0)) print('b1 = **{}** - (теоретический), **{}** - (градиентный спуск)'.format(b1, grad\_b1)) print('MSE = **{}**'.format(mean\_squared\_error(y\_array\_regr, grad\_y\_pred))) plt.plot(x\_array, y\_array, 'g.')

plt.plot(x\_array, y\_array\_regr, 'b', linewidth=2.0) plt.plot(x\_array, grad\_y\_pred, 'r', linewidth=2.0) plt.show()

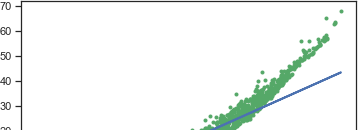
In [16]:

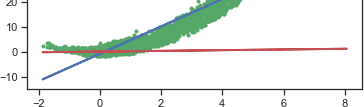


*# Примеры использования градиентного спуска*

show\_gradient\_descent(10, 0, 0)

b0 = -0.8493834635342365 - (теоретический), 0.03448999261498756 - (градиентный спуск) b1 = 5.482102483932346 - (теоретический), 0.13918310376478424 - (градиентный спуск) MSE = 35.104804413419025





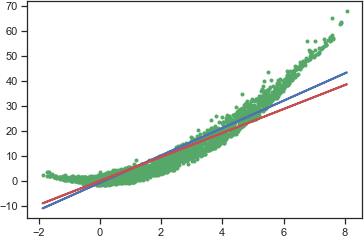
In [17]:



show\_gradient\_descent(1000, 0, 0)

b0 = -0.8493834635342365 - (теоретический), -0.1046956764421704 - (градиентный спуск) b1 = 5.482102483932346 - (теоретический), 4.8016950489981 - (градиентный спуск)

MSE = 0.700140861729278

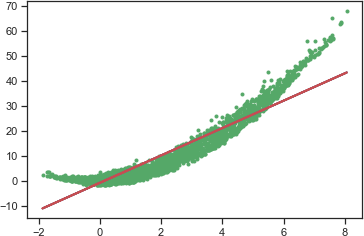


In [18]:



show\_gradient\_descent(10000, 0, 0)

b0 = -0.8493834635342365 - (теоретический), -0.8493787037369952 - (градиентный спуск) b1 = 5.482102483932346 - (теоретический), 5.482099192076996 - (градиентный спуск) MSE = 2.2423416471452288e-11



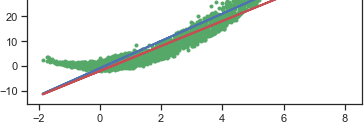
In [19]:



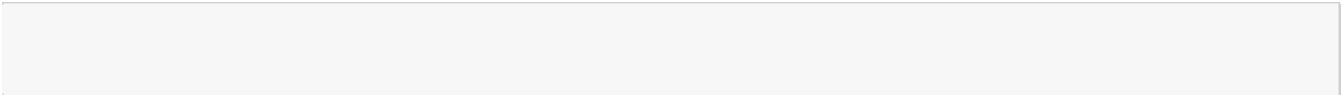
show\_gradient\_descent(10, -2, 5)

b0 = -0.8493834635342365 - (теоретический), -1.972729515823501 - (градиентный спуск) b1 = 5.482102483932346 - (теоретический), 5.023704112532512 - (градиентный спуск) MSE = 2.039915190146421





In [20]:



*# Обучим линейную регрессию и сравним коэффициенты с рассчитанными ранее* reg1 = LinearRegression().fit(x\_array.reshape(-1, 1), y\_array.reshape(-1, 1)) (b1, reg1.coef\_), (b0, reg1.intercept\_)

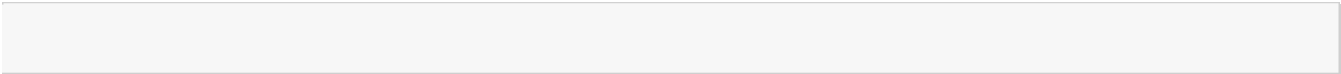
Out[20]:

((5.482102483932346, array([[5.48210248]])),

(-0.8493834635342365, array([-0.84938346])))

## SVM

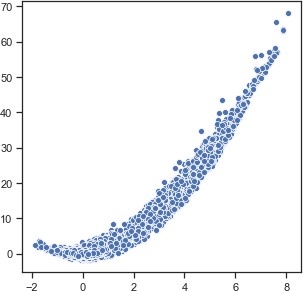
In [21]:



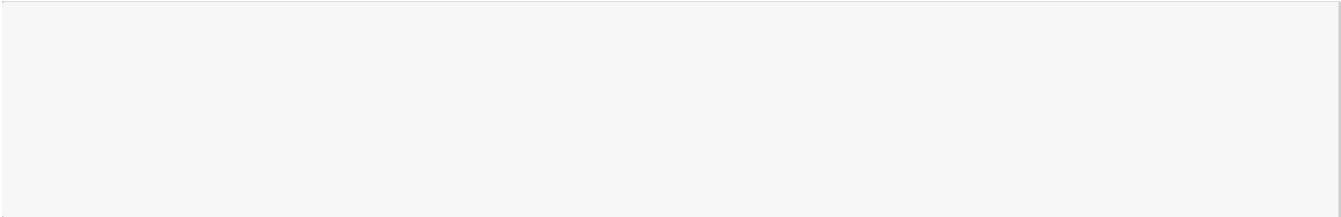
fig, ax = plt.subplots(figsize=(5,5)) sns.scatterplot(ax=ax, x=x\_array, y=y\_array)

Out[21]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x14146d50>



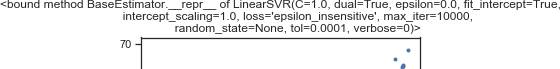
In [22]:



**def** plot\_regr(clf): title = clf. repr

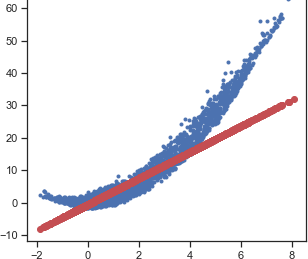
clf.fit(x\_array.reshape(-1, 1), y\_array) y\_pred = clf.predict(x\_array.reshape(-1, 1)) fig, ax = plt.subplots(figsize=(5,5)) ax.set\_title(title)

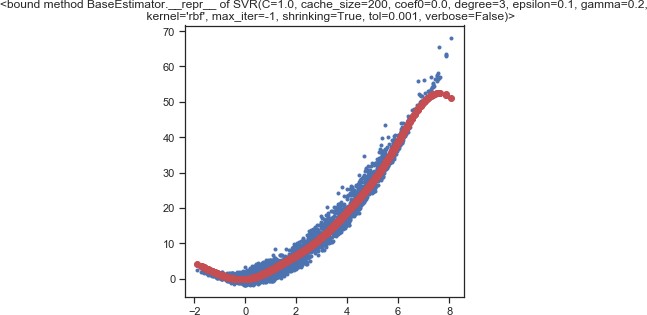
ax.plot(x\_array, y\_array, 'b.') ax.plot(x\_array, y\_pred, 'ro') plt.show()

In [23]:



plot\_regr(LinearSVR(C=1.0, max\_iter=10000))



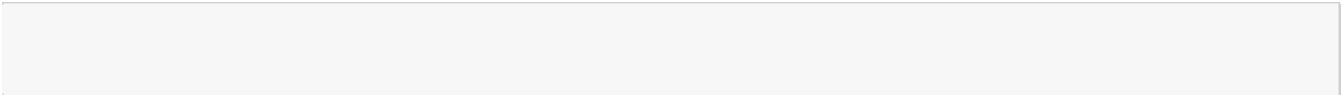
In [24]:



plot\_regr(SVR(kernel='rbf', gamma=0.2, C=1.0))

## Дерево решений

In [25]:



*# Обучим дерево на всех признаках*

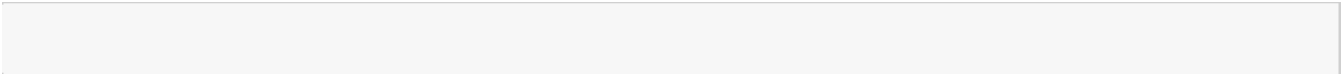
tree = DecisionTreeRegressor(random\_state=1) tree.fit(X\_train, Y\_train)

Out[25]:

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=1, splitter='best')

In [26]:



*# Важность признаков*

list(zip(X\_train.columns.values, tree.feature\_importances\_))

Out[26]:

[(' Mean of the integrated profile', 0.02764437664705047),

(' Standard deviation of the integrated profile', 0.029344493324682287), (' Excess kurtosis of the integrated profile', 0.7986346670595349),

(' Skewness of the integrated profile', 0.02697304742290089), (' Mean of the DM-SNR curve', 0.020993135609091208),

(' Standard deviation of the DM-SNR curve', 0.05870193902245078),

(' Excess kurtosis of the DM-SNR curve', 0.022090683667882426), (' Skewness of the DM-SNR curve', 0.015617657246407004)]

In [27]:

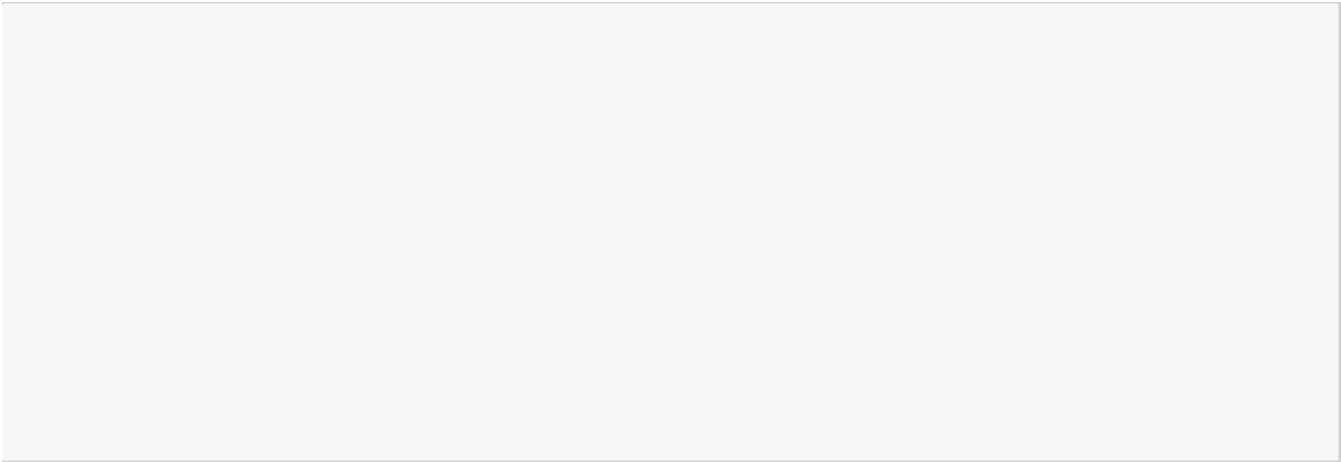


*# Важность признаков в сумме дает единицу*

sum(tree.feature\_importances\_)

Out[27]: 1.0

In [28]:



**def** draw\_feature\_importances(tree\_model, X\_dataset, figsize=(10,5)):

*"""*

*Вывод важности признаков в виде графика """*

*# Сортировка значений важности признаков по убыванию*

list\_to\_sort = list(zip(X\_dataset.columns.values, tree\_model.feature\_importances\_)) sorted\_list = sorted(list\_to\_sort, key=itemgetter(1), reverse = **True**)

*# Названия признаков*

labels = [x **for** x,\_ **in** sorted\_list]

*# Важности признаков*

data = [x **for** \_,x **in** sorted\_list]

*# Вывод графика*

fig, ax = plt.subplots(figsize=figsize) ind = np.arange(len(labels)) plt.bar(ind, data)

plt.xticks(ind, labels, rotation='vertical')

*# Вывод значений*

**for** a,b **in** zip(ind, data):

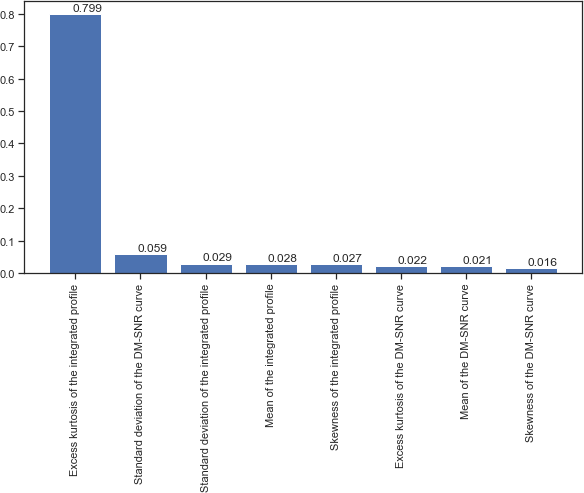
plt.text(a-0.05, b+0.01, str(round(b,3))) plt.show()

**return** labels, data

In [29]:

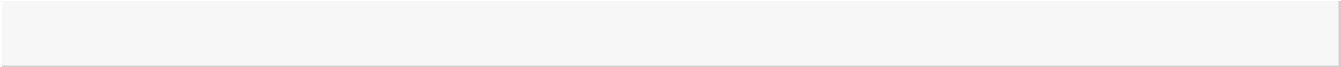


tree\_fl, tree\_fd = draw\_feature\_importances(tree, X\_train)



In [30]:





*# Список признаков, отсортированный на основе важности, и значения важности*

tree\_fl, tree\_fd

Out[30]:

([' Excess kurtosis of the integrated profile', ' Standard deviation of the DM-SNR curve',

* Standard deviation of the integrated profile', ' Mean of the integrated profile',
* Skewness of the integrated profile', ' Excess kurtosis of the DM-SNR curve', ' Mean of the DM-SNR curve',
* Skewness of the DM-SNR curve'], [0.7986346670595349,

0.05870193902245078,

0.029344493324682287,

0.02764437664705047,

0.02697304742290089,

0.022090683667882426,

0.020993135609091208,

0.015617657246407004])

In [31]:

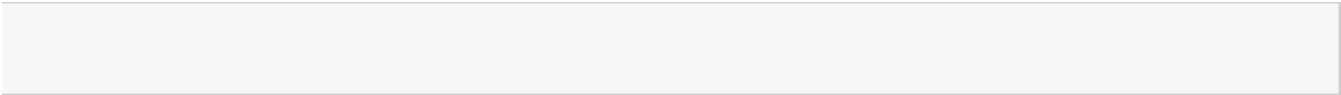


X\_train.head()

Out[31]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Mean of the integrated**  **profile** | **Standard deviation of the integrated profile** | **Excess kurtosis of the integrated**  **profile** | **Skewness of the integrated**  **profile** | **Mean of the DM- SNR curve** | **Standard deviation of the DM-SNR curve** | **Excess kurtosis of the DM-SNR**  **curve** | **Skewness of the DM-SNR**  **curve** |
| **5744** | 79.429688 | 48.745943 | 1.594305 | 3.010275 | 19.621237 | 52.699148 | 2.758187 | 6.734941 |
| **12724** | 26.421875 | 28.354786 | 6.414866 | 45.383641 | 59.007525 | 58.278946 | 1.452548 | 1.756035 |
| **1010** | 138.476562 | 58.650843 | -0.251328 | -0.632375 | 2.656355 | 20.022599 | 8.482193 | 76.954850 |
| **15844** | 94.117188 | 41.160247 | 0.608247 | 1.171382 | 2.255853 | 17.461752 | 10.027291 | 112.565713 |
| **4163** | 130.039062 | 51.597663 | -0.033977 | -0.202117 | 17.432274 | 47.884699 | 2.799555 | 7.008160 |

In [32]:



*# Пересортируем признаки на основе важности* X\_train\_sorted = X\_train[tree\_fl] X\_train\_sorted.head()

Out[32]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Excess kurtosis of the integrated**  **profile** | **Standard deviation of the DM-SNR curve** | **Standard deviation of the integrated profile** | **Mean of the integrated**  **profile** | **Skewness of the integrated**  **profile** | **Excess kurtosis of the DM-SNR**  **curve** | **Mean of the DM- SNR curve** | **Skewness of the DM-SNR**  **curve** |
| **5744** | 1.594305 | 52.699148 | 48.745943 | 79.429688 | 3.010275 | 2.758187 | 19.621237 | 6.734941 |
| **12724** | 6.414866 | 58.278946 | 28.354786 | 26.421875 | 45.383641 | 1.452548 | 59.007525 | 1.756035 |
| **1010** | -0.251328 | 20.022599 | 58.650843 | 138.476562 | -0.632375 | 8.482193 | 2.656355 | 76.954850 |
| **15844** | 0.608247 | 17.461752 | 41.160247 | 94.117188 | 1.171382 | 10.027291 | 2.255853 | 112.565713 |
| **4163** | -0.033977 | 47.884699 | 51.597663 | 130.039062 | -0.202117 | 2.799555 | 17.432274 | 7.008160 |

In [33]:



Y\_test\_predict = tree.predict(X\_test)

In [34]:

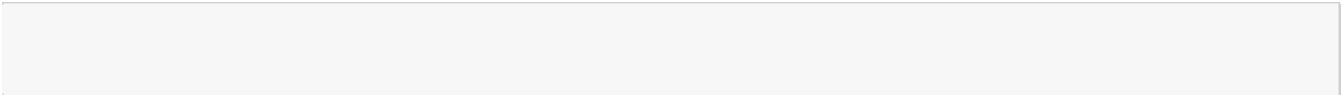


mean\_absolute\_error(Y\_test, Y\_test\_predict)

Out[34]: 0.03217877094972067



In [35]:



*# Обучим дерево и предскажем результаты на пяти лучших признаках*

tree\_2 = DecisionTreeRegressor(random\_state=1).fit(X\_train[tree\_fl[0:5]], Y\_train) Y\_test\_predict\_2 = tree\_2.predict(X\_test[tree\_fl[0:5]])

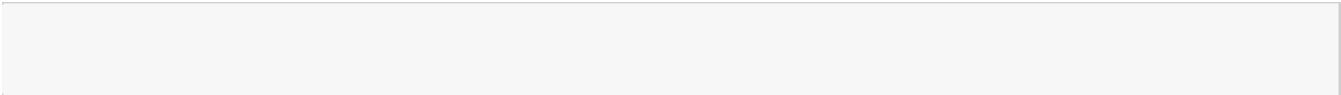
In [36]:



mean\_absolute\_error(Y\_test, Y\_test\_predict\_2)

Out[36]: 0.033072625698324025

In [37]:



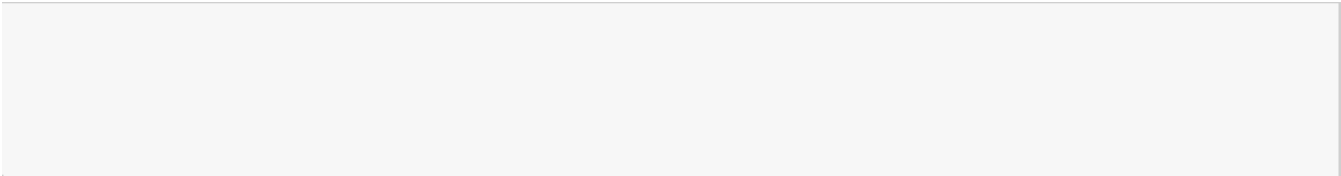
*# Исследуем, как изменяется ошибка при добавлении признаков в порядке значимости*

X\_range = list(range(1, len(X\_train.columns)+1)) X\_range

Out[37]:

[1, 2, 3, 4, 5, 6, 7, 8]

In [38]:



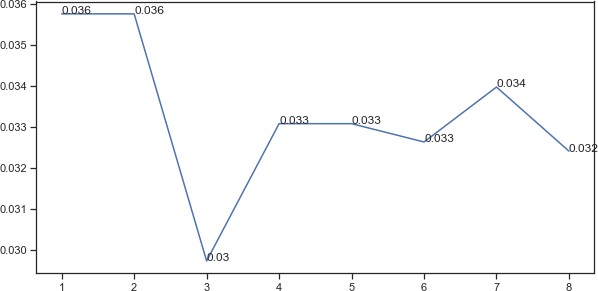
mae\_list = []

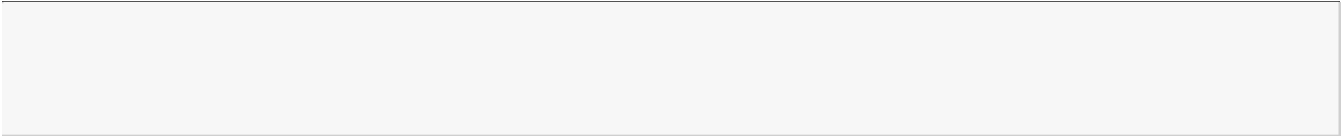
**for** i **in** X\_range:

*# Обучим дерево и предскажем результаты на заданном количестве признаков*

tree\_3 = DecisionTreeRegressor(random\_state=1).fit(X\_train[tree\_fl[0:i]], Y\_train) Y\_test\_predict\_3 = tree\_3.predict(X\_test[tree\_fl[0:i]])

temp\_mae = mean\_absolute\_error(Y\_test, Y\_test\_predict\_3) mae\_list.append(temp\_mae)

In [39]:



plt.subplots(figsize=(10,5)) plt.plot(X\_range, mae\_list)

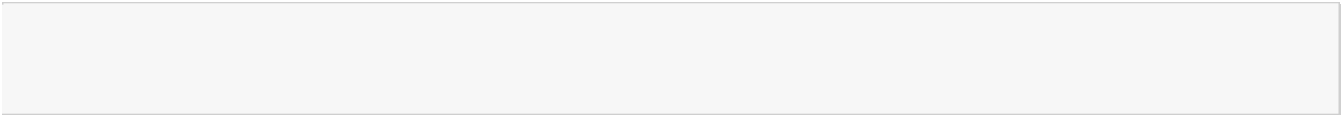
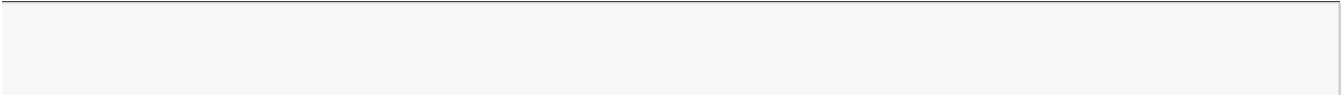
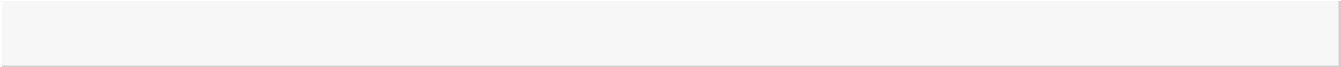
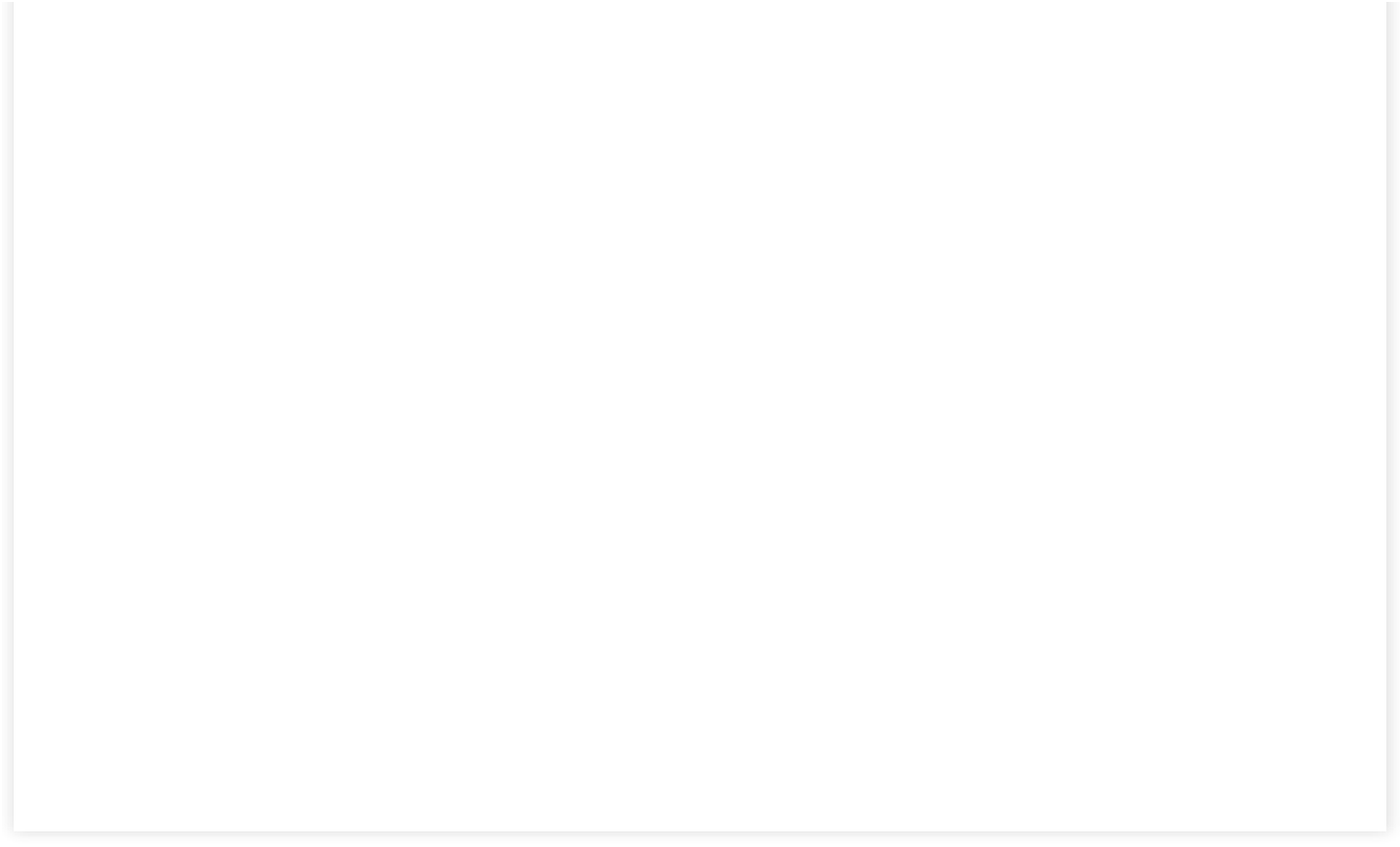
**for** a,b **in** zip(X\_range, mae\_list): plt.text(a, b, str(round(b,3)))

plt.show()

## Оценка качества моделей

Дерево решений

In [40]:



print("r2\_score:", r2\_score(Y\_test, tree.predict(X\_test))) print("mean\_squared\_error:", mean\_squared\_error(Y\_test, tree.predict(X\_test)))

r2\_score: 0.608192211999387

mean\_squared\_error: 0.03217877094972067

Линейная регрессия

In [41]:

pred = reg1.predict(x\_array.reshape(-1, 1)) print("r2\_score:", r2\_score(y\_array, pred))

print("mean\_squared\_error", mean\_squared\_error(y\_array, pred))

r2\_score: 0.8944034470145041

mean\_squared\_error 4.017001458201285

Метод опорных векторов

In [42]:

svr = SVR(kernel='rbf') svr.fit(X\_train, Y\_train)

print("r2\_score:", r2\_score(Y\_test, svr.predict(X\_test))) print("mean\_squared\_error", mean\_squared\_error(Y\_test, svr.predict(X\_test)))

r2\_score: 0.7018891842122201

mean\_squared\_error 0.02448353491854106

Можно сказать, что все три модели являются приемлемыми, т.к. коэффициент детерминации для всех трех моделей больше 50%.

Если учитывать показатели обеих метрик, наилучший результат показал метод опорных векторов.