

```

import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from scipy.optimize import fmin_tnc
from IPython.display import Image
from sklearn.datasets import load_iris, load_boston
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor,
KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import SGDClassifier
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")

```

```
data = pd.read_csv('melb_data.csv', sep=",")
```

```
data.head()
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson

	Date	Distance	Postcode	...	Bathroom	Car	Landsize
0	3/12/2016	2.5	3067.0	...	1.0	1.0	202.0
1	4/02/2016	2.5	3067.0	...	1.0	0.0	156.0
2	4/03/2017	2.5	3067.0	...	2.0	0.0	134.0

```

3  4/03/2017      2.5    3067.0  ...      2.0  1.0      94.0
NaN
4  4/06/2016      2.5    3067.0  ...      1.0  2.0     120.0
142.0

```

```

      YearBuilt  CouncilArea  Lattitude  Longtitude      Regionname
\
0      NaN      Yarra  -37.7996    144.9984  Northern Metropolitan
1    1900.0      Yarra  -37.8079    144.9934  Northern Metropolitan
2    1900.0      Yarra  -37.8093    144.9944  Northern Metropolitan
3      NaN      Yarra  -37.7969    144.9969  Northern Metropolitan
4    2014.0      Yarra  -37.8072    144.9941  Northern Metropolitan

```

```

      Propertycount
0      4019.0
1      4019.0
2      4019.0
3      4019.0
4      4019.0

```

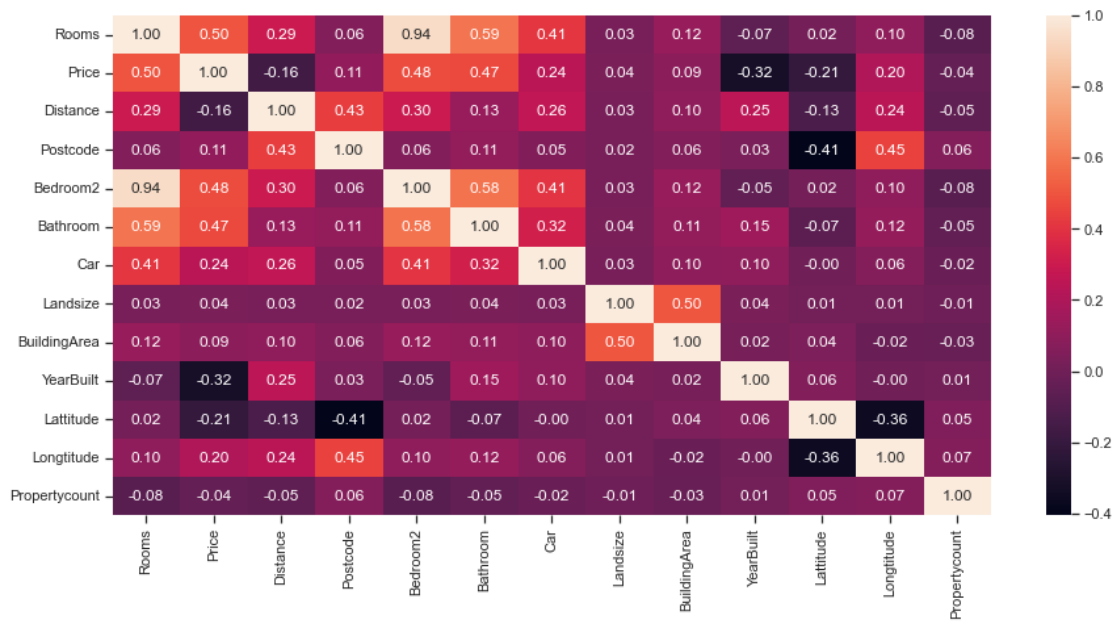
```
[5 rows x 21 columns]
```

```
#Построим корреляционную матрицу
```

```
fig, ax = plt.subplots(figsize=(15,7))
```

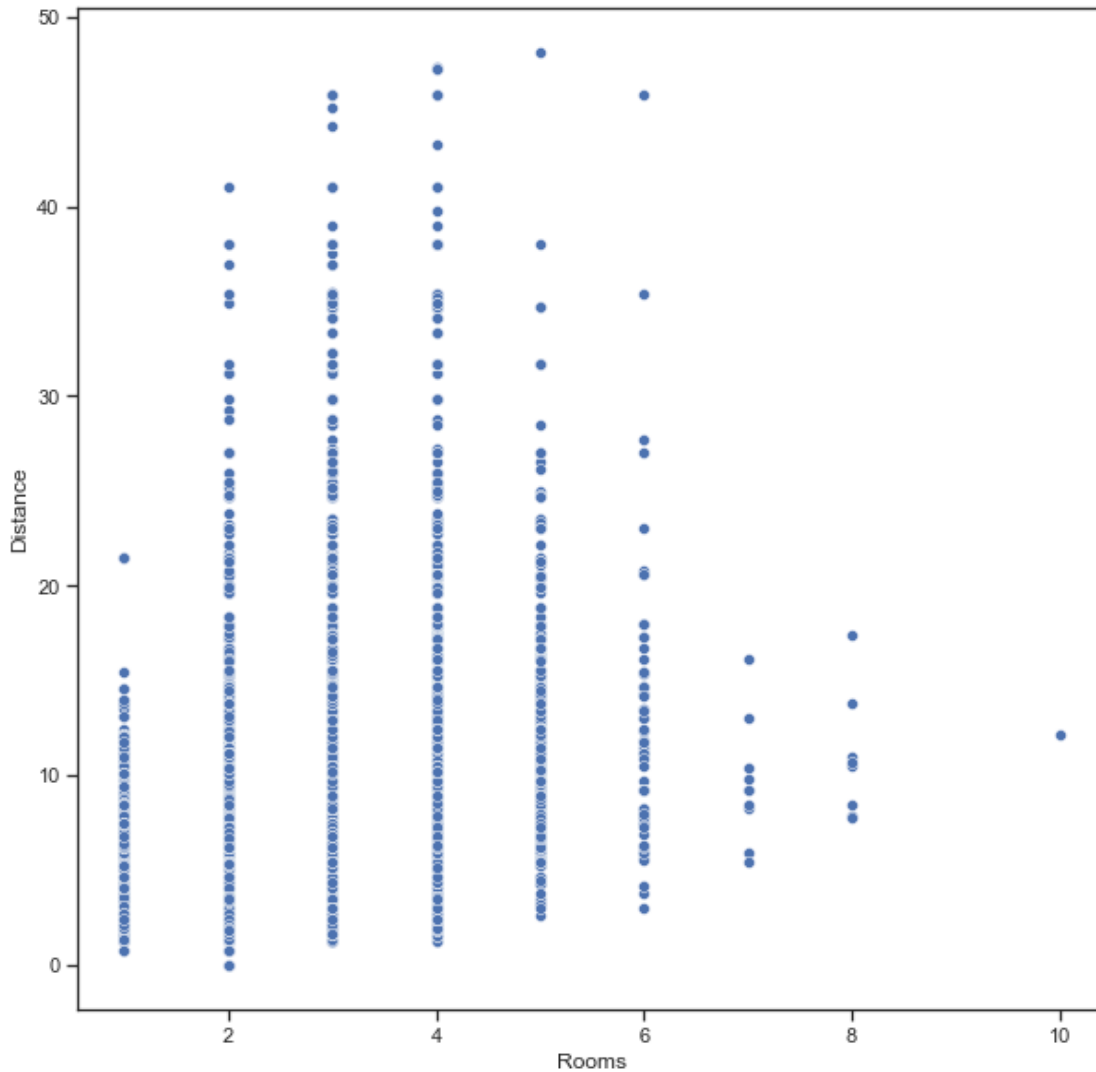
```
sns.heatmap(data.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
<AxesSubplot:>
```



```
fig, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(ax=ax, x='Rooms', y='Distance', data=data)

<AxesSubplot:xlabel='Rooms', ylabel='Distance'>
```



*# 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-y_mean) for x, y in zip(x_array,
y_array)])
    b1 = cov1 / var1
    b0 = y_mean - b1*x_mean
    return b0, b1
```

```
x_array = data['Rooms'].values
y_array = data['Distance'].values
```

```
b0, b1 = analytic_regr_coef(x_array, y_array)
b0, b1
```

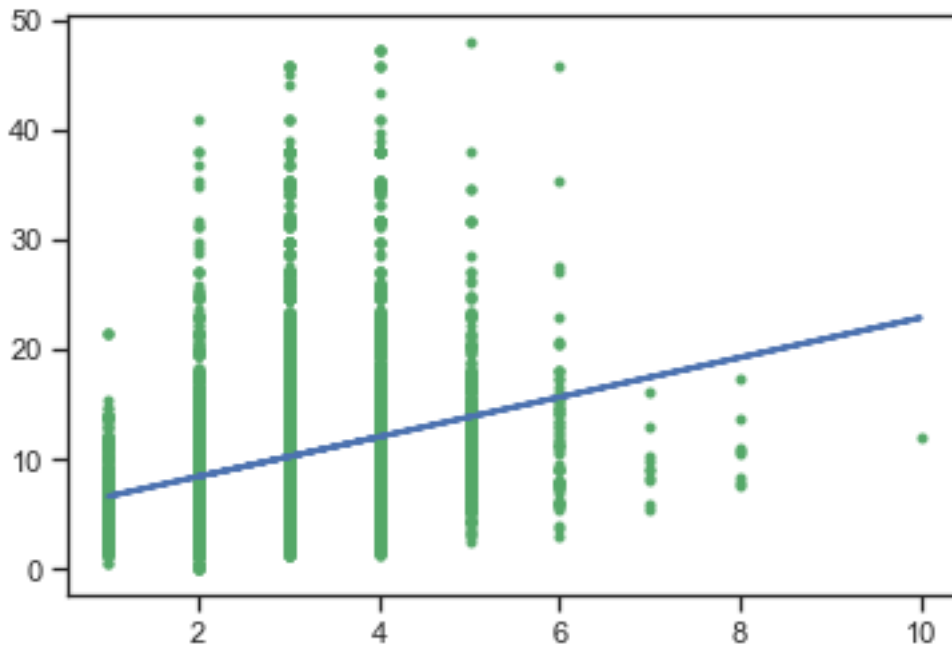
```
(4.83017680419352, 1.8065366434170134)
```

```
# 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
```

```
y_array_regr = y_regr(x_array, b0, b1)
```

```
plt.plot(x_array, y_array, 'g.')  
plt.plot(x_array, y_array_regr, 'b', linewidth=2.0)  
plt.show()
```

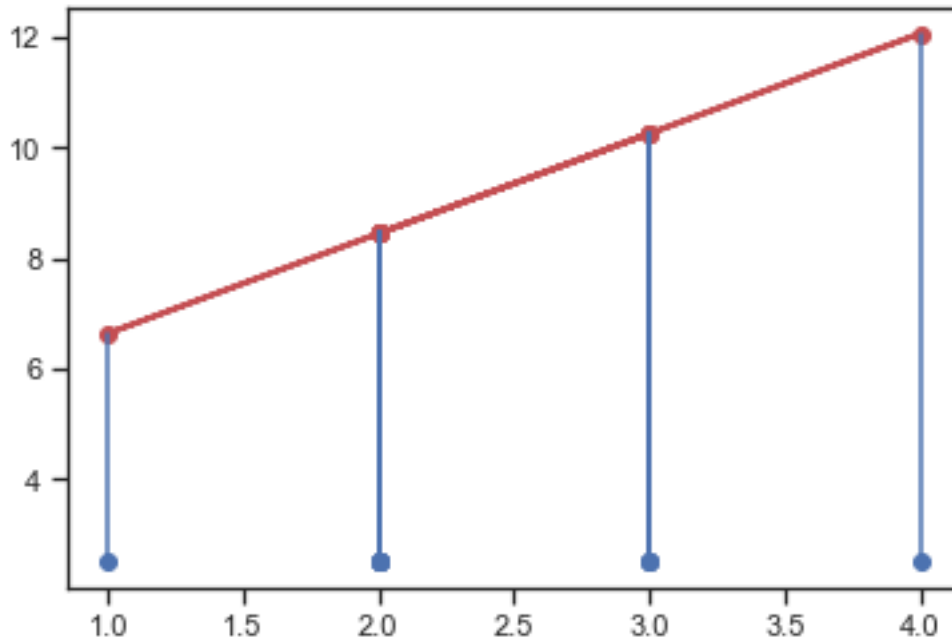


```
# Синими отрезками показаны ошибки между  
# истинными и предсказанными значениями  
K_mnk=10
```

```
plt.plot(x_array[1:K_mnk+1], y_array[1:K_mnk+1], 'bo')  
plt.plot(x_array[1:K_mnk+1], y_array_regr[1:K_mnk+1], '-ro',  
         linewidth=2.0)
```

```
for i in range(len(x_array[1:K_mnk+1])):  
    x1 = x_array[1:K_mnk+1][i]  
    y1 = y_array[1:K_mnk+1][i]  
    y2 = y_array_regr[1:K_mnk+1][i]  
    plt.plot([x1,x1],[y1,y2], 'b-')
```

```
plt.show()
```



*# 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

def show_gradient_descent(epochs, b0_0, b1_0):
    grad_b0, grad_b1, grad_y_pred = gradient_descent(x_array, y_array,
```

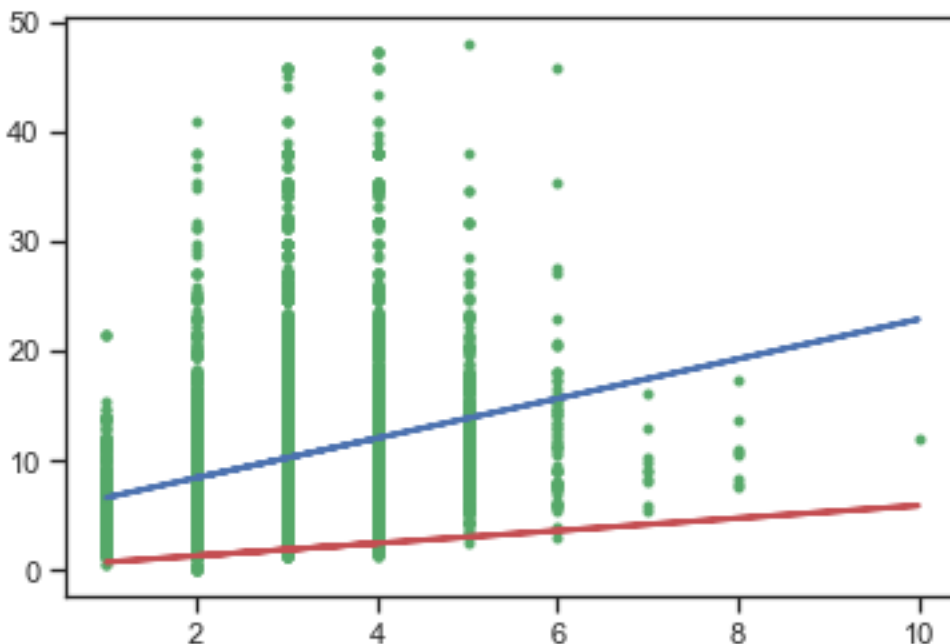
```

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()

```

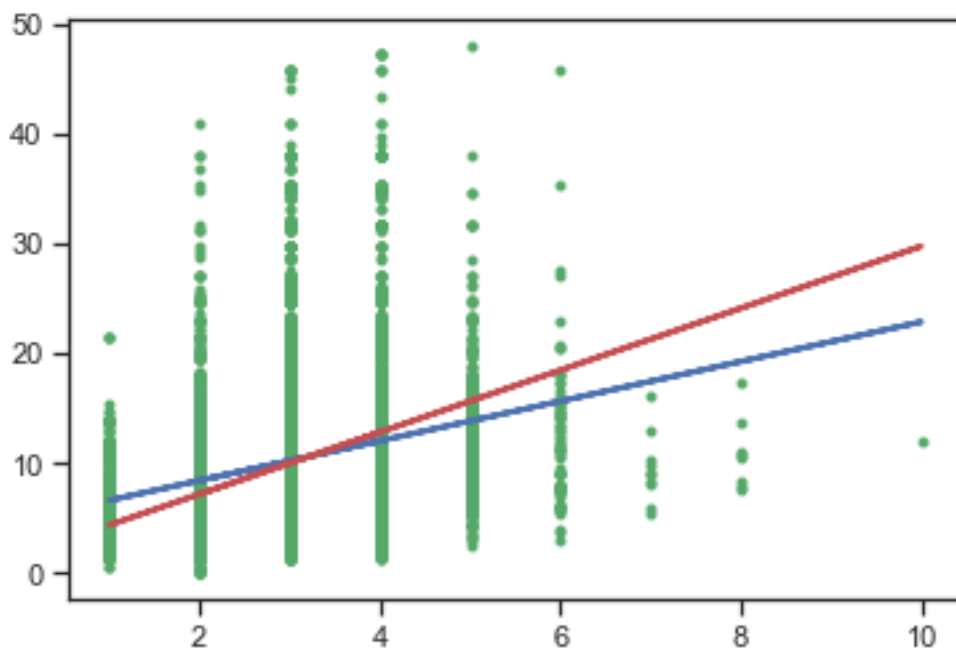
*# Примеры использования градиентного спуска*  
show\_gradient\_descent(10, 0, 0)

b0 = 4.83017680419352 - (теоретический), 0.1852987147824515 -  
(градиентный спуск)  
b1 = 1.8065366434170134 - (теоретический), 0.5725191196015689 -  
(градиентный спуск)  
MSE = 69.79071771536495



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

b0 = 4.83017680419352 - (теоретический), 1.5609200283930669 -  
(градиентный спуск)  
b1 = 1.8065366434170134 - (теоретический), 2.8220996378843677 -  
(градиентный спуск)  
MSE = 1.0235687847709871



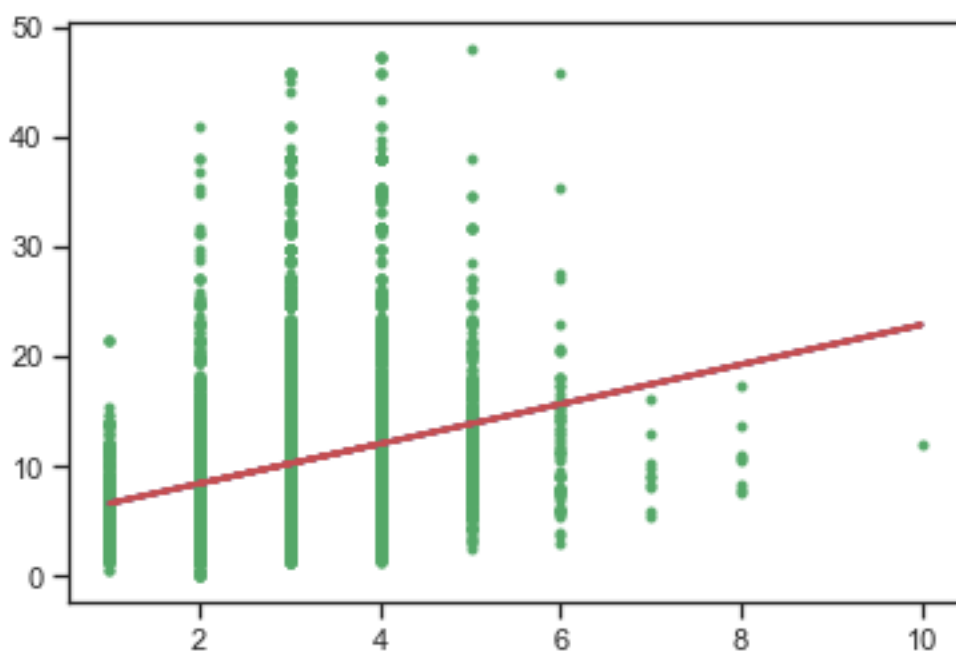
```
%%time
```

```
show_gradient_descent(50000, 0, 0)
```

$b_0 = 4.83017680419352$  - (теоретический),  $4.829550371293804$  - (градиентный спуск)

$b_1 = 1.8065366434170134$  - (теоретический),  $1.8067312387383558$  - (градиентный спуск)

MSE =  $3.758097860837766e-08$



Wall time: 3.6 s



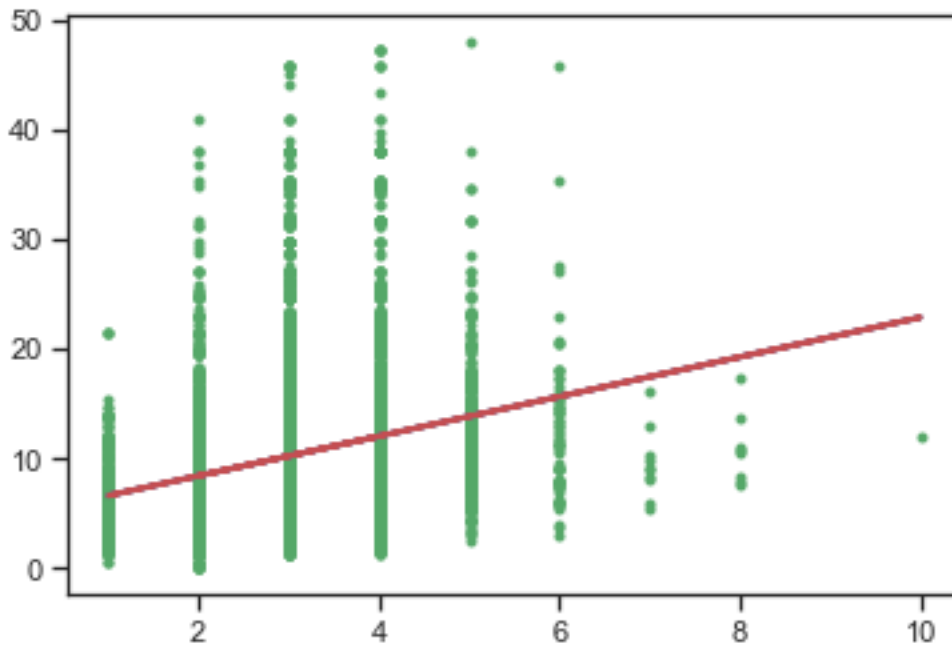
```
%%time
```

```
show_gradient_descent(100000, 0, 0)
```

```
b0 = 4.83017680419352 - (теоретический), 4.830176703400344 -  
(градиентный спуск)
```

```
b1 = 1.8065366434170134 - (теоретический), 1.8065366747274394 -  
(градиентный спуск)
```

```
MSE = 9.729291730126751e-16
```



```
Wall time: 6.96 s
```

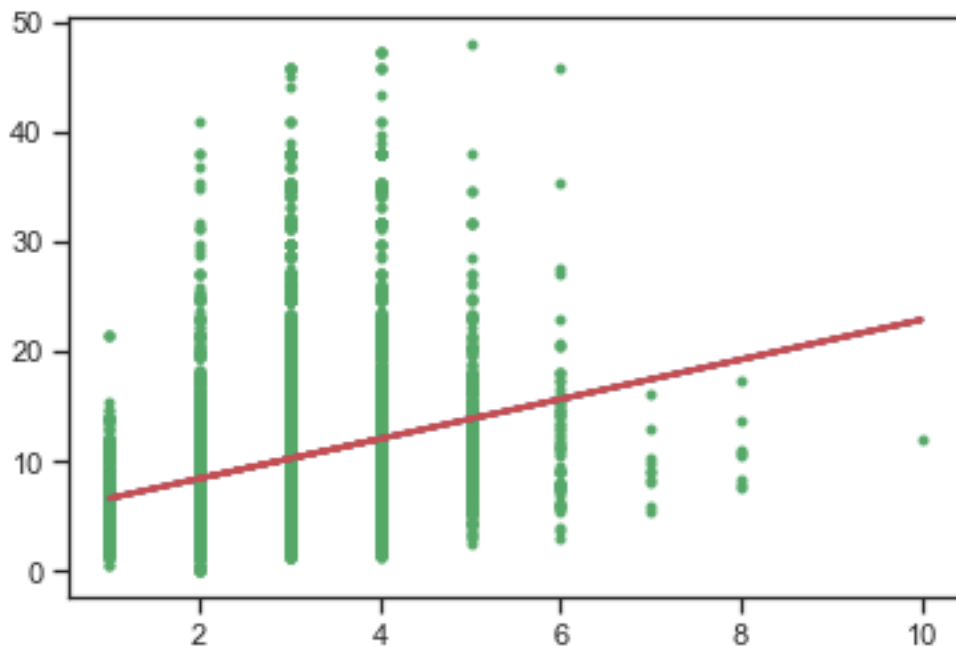
```
%%time
```

```
show_gradient_descent(1000000, 0, 0)
```

```
b0 = 4.83017680419352 - (теоретический), 4.8301768041910265 -  
(градиентный спуск)
```

```
b1 = 1.8065366434170134 - (теоретический), 1.8065366434177865 -  
(градиентный спуск)
```

```
MSE = 5.957213408071935e-25
```



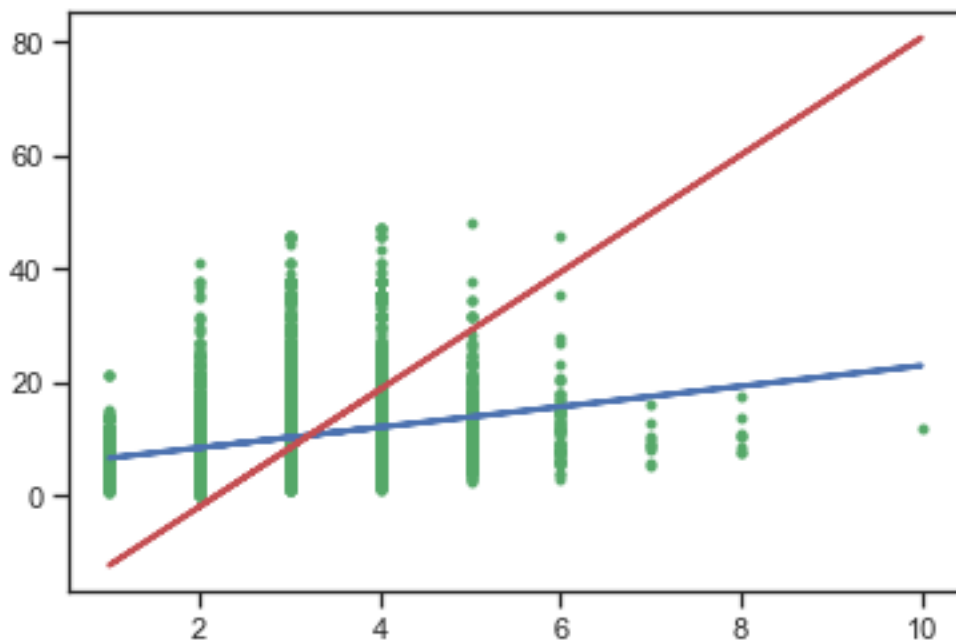
Wall time: 1min 8s

*# Сходимость алгоритма может сильно зависеть от начальных значений*  
`show_gradient_descent(1000, -30, 5)`

$b_0 = 4.83017680419352$  - (теоретический),  $-22.60297578385468$  - (градиентный спуск)

$b_1 = 1.8065366434170134$  - (теоретический),  $10.328380152221749$  - (градиентный спуск)

MSE = 72.07263599382124

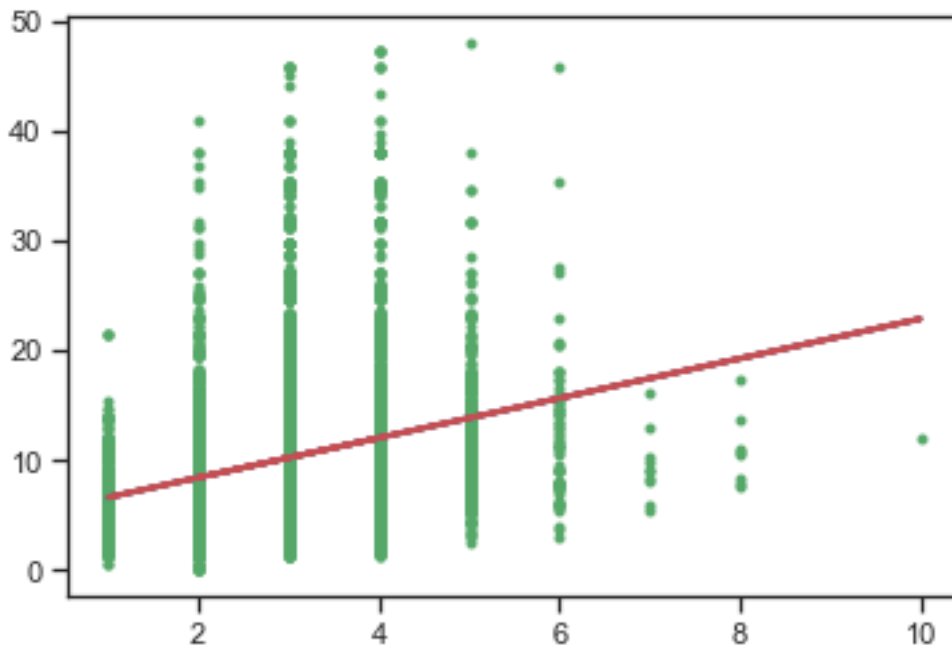


```
show_gradient_descent(100000, -30, 5)
```

b0 = 4.83017680419352 - (теоретический), 4.830175958412565 -  
(градиентный спуск)

b1 = 1.8065366434170134 - (теоретический), 1.8065369061506864 -  
(градиентный спуск)

MSE = 6.850697195193023e-14



*# 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_)
```

```
((1.8065366434170134, array([[1.80653664]])),  
(4.83017680419352, array([4.8301768])))
```

*# Для небольшой выборки качество обучения сильно уступает нестохастическому градиентному спуску.*

```
print('Размер выборки - {}'.format(x_array.shape[0]))  
reg2 = SGDRegressor().fit(x_array.reshape(-1, 1), y_array)  
(b1, reg2.coef_), (b0, reg2.intercept_)
```

Размер выборки - 13580

```
((1.8065366434170134, array([1.86537821])),  
(4.83017680419352, array([4.86889697])))
```

```
from sklearn.linear_model import Lasso
```

```

reg3 = Lasso().fit(x_array.reshape(-1, 1), y_array)
(b1, reg3.coef_), (b0, reg3.intercept_)

((1.8065366434170134, array([0.71171029])),
 (4.83017680419352, 8.046773399101927))

from sklearn.linear_model import Ridge

reg4 = Ridge().fit(x_array.reshape(-1, 1), y_array)
(b1, reg4.coef_), (b0, reg4.intercept_)

((1.8065366434170134, array([1.80639101])),
 (4.83017680419352, 4.830604670995869))

from sklearn.linear_model import ElasticNet

reg5 = ElasticNet().fit(x_array.reshape(-1, 1), y_array)
(b1, reg5.coef_), (b0, reg5.intercept_)

((1.8065366434170134, array([0.81369571])),
 (4.83017680419352, 7.747140532157479))

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

poly_model = Pipeline([('poly', PolynomialFeatures(degree=3)),
                        ('linear',
                         LinearRegression(fit_intercept=False))])

poly_model.fit(x_array.reshape(-1, 1), y_array)

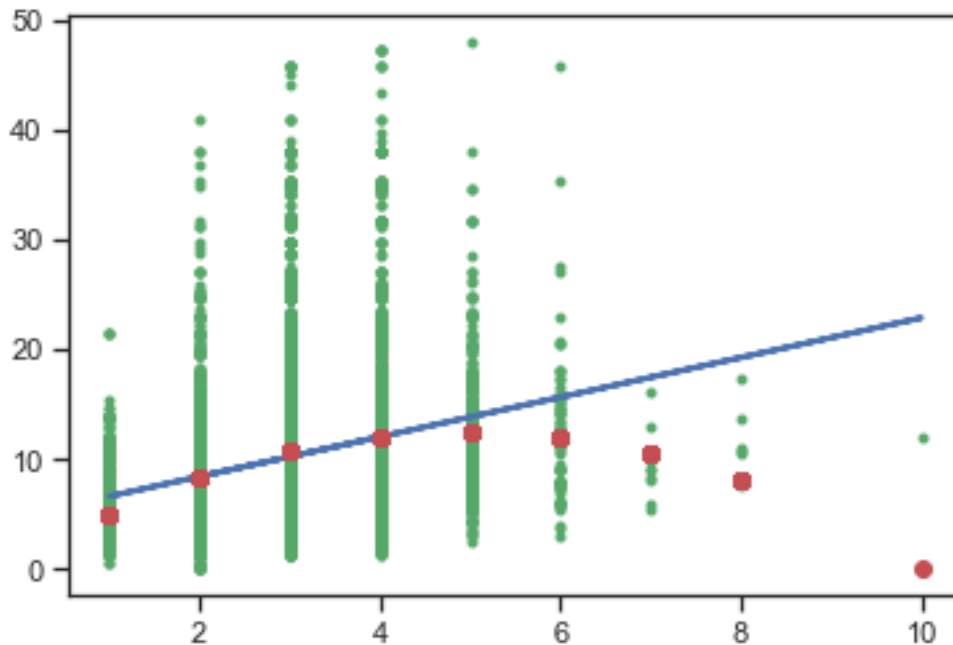
Pipeline(steps=[('poly', PolynomialFeatures(degree=3)),
                 ('linear', LinearRegression(fit_intercept=False))])

poly_model.fit(x_array.reshape(-1, 1), y_array)

Pipeline(steps=[('poly', PolynomialFeatures(degree=3)),
                 ('linear', LinearRegression(fit_intercept=False))])

poly_y_pred = poly_model.predict(x_array.reshape(-1, 1))
plt.plot(x_array, y_array, 'g.')
plt.plot(x_array, y_array_regr, 'b', linewidth=2.0)
plt.plot(x_array, poly_y_pred, 'ro')
plt.show()

```



```
# Степени полинома
```

```
poly_model.named_steps['linear'].coef_,  
poly_model.named_steps['linear'].intercept_
```

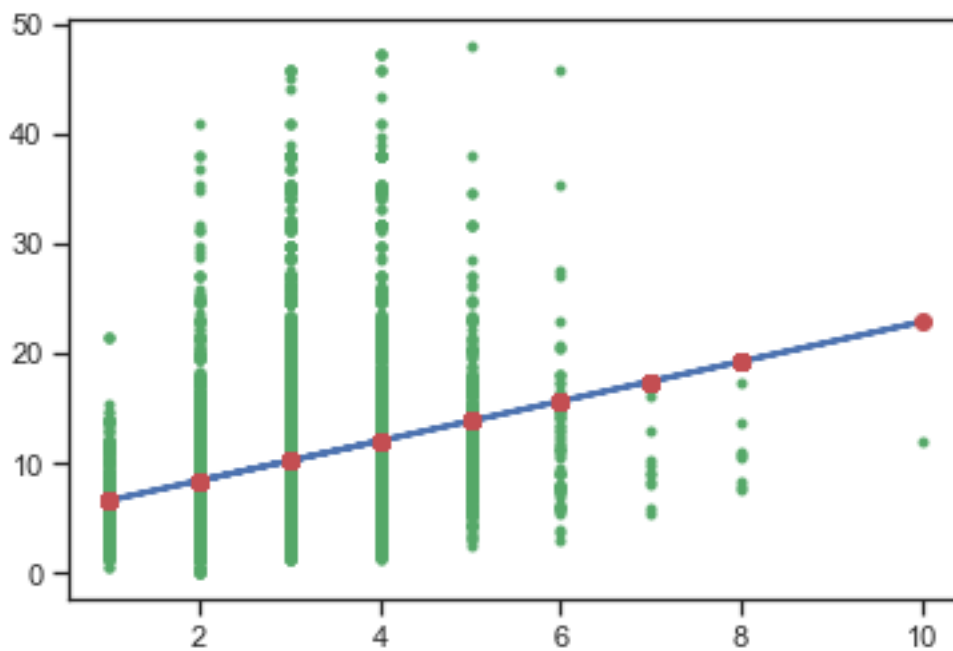
```
(array([ 6.62627100e-01,  4.74434239e+00, -4.60319777e-01, -  
2.07070134e-03]),  
 0.0)
```

```
def test_poly_model(degree=3):  
    poly_model = Pipeline([('poly',  
PolynomialFeatures(degree=degree)),  
        ('linear',  
LinearRegression(fit_intercept=False))])  
    poly_model.fit(x_array.reshape(-1, 1), y_array)  
    poly_y_pred = poly_model.predict(x_array.reshape(-1, 1))
```

```
    plt.plot(x_array, y_array, 'g.')  
    plt.plot(x_array, y_array_regr, 'b', linewidth=2.0)  
    plt.plot(x_array, poly_y_pred, 'ro')  
    plt.show()
```

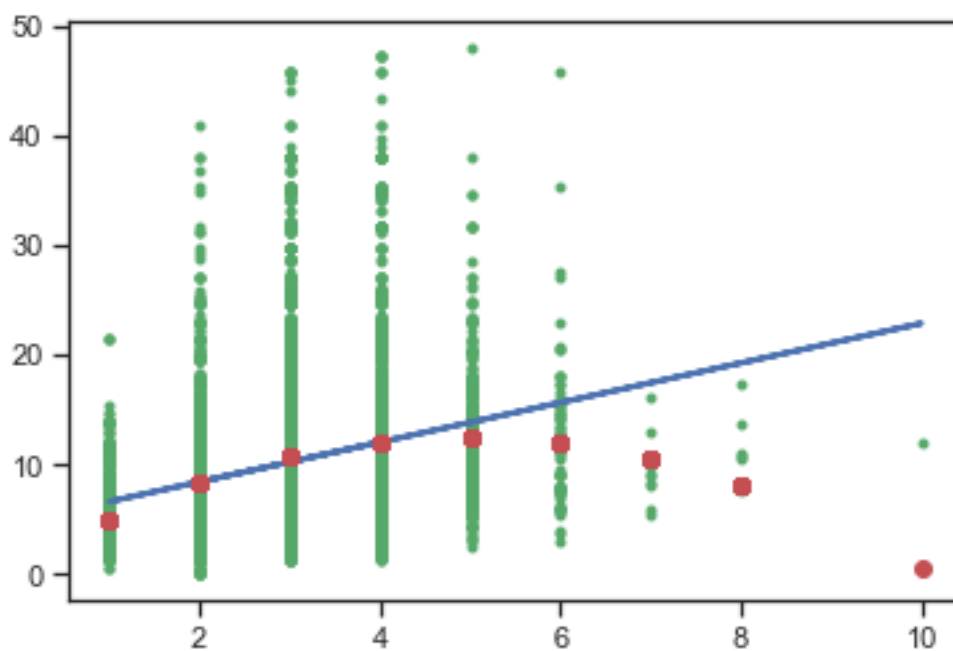
```
    print('Степени полинома -  
{0}'.format(poly_model.named_steps['linear'].coef_))
```

```
test_poly_model(degree=1)
```



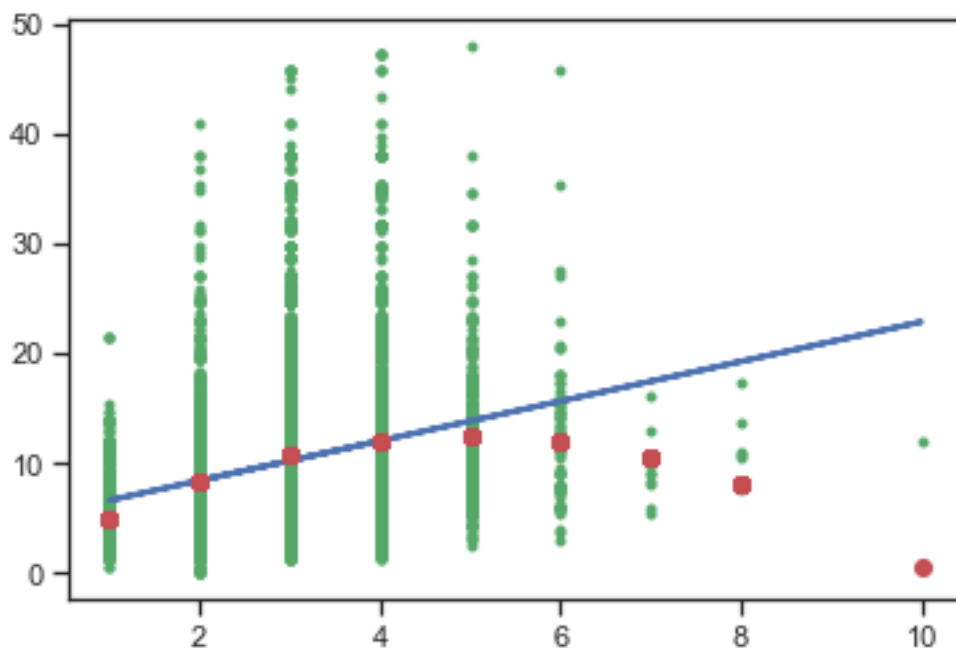
Степени полинома - [4.8301768 1.80653664]

test\_poly\_model(degree=2)



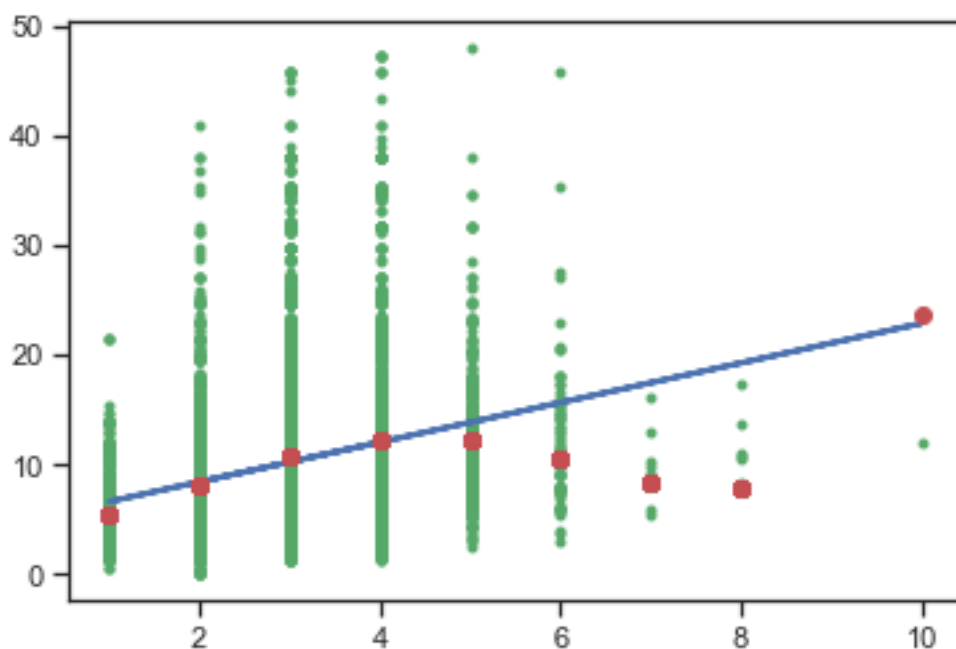
Степени полинома - [ 0.59525262 4.81584462 -0.48258887]

test\_poly\_model(degree=2)



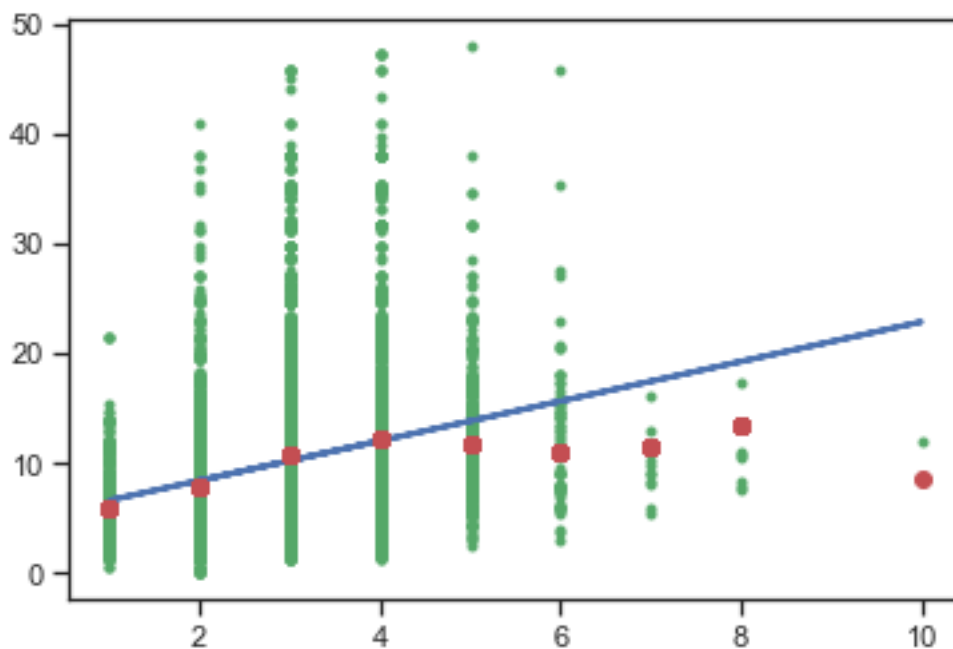
Степени полинома - [ 0.59525262 4.81584462 -0.48258887]

test\_poly\_model(degree=4)



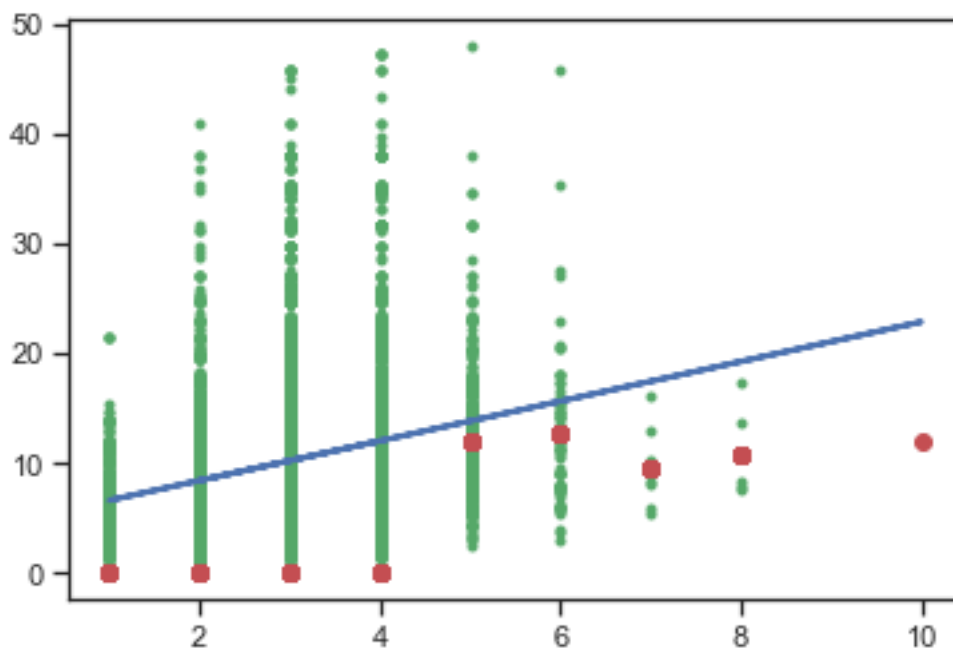
Степени полинома - [ 5.668722 -2.39146926 2.83283556 -0.59633531  
0.0355107 ]

test\_poly\_model(degree=5)



Степени полинома - [ 1.25241705e+01 -1.47843538e+01 1.05762708e+01 -  
2.73729652e+00  
3.00633274e-01 -1.18266141e-02]

test\_poly\_model(degree=35)



Степени полинома - [ 5.15135913e-22 -3.58491391e-23 1.21353054e-26  
7.17358085e-30  
-4.03429661e-33 4.81489351e-35 2.15947621e-34 8.97789572e-34  
3.84432602e-33 1.68652620e-32 7.54545558e-32 3.42881182e-31]



```

1.57700135e-30  7.31801450e-30  3.41669826e-29  1.60083832e-28
7.50838930e-28  3.51682219e-27  1.64087558e-26  7.60610119e-26
3.49230698e-25  1.58276493e-24  7.05096010e-24  3.07127939e-23
1.29914186e-22  5.28748312e-22  2.04383858e-21  7.35961933e-21
2.39377335e-20  6.65883009e-20  1.41110934e-19  1.57956363e-19
-1.13227238e-19  2.44389672e-20  -2.21637950e-21  7.31997130e-23]

```

```
# Определение функции
```

```
# f(0)=0.5
```

```
x = np.linspace(-7, 7, 31)
```

```
y = 1 / (1 + np.exp(-x))
```

```
list(zip(x,y))
```

```

[(-7.0, 0.0009110511944006454),
 (-6.533333333333333, 0.0014520391100099122),
 (-6.066666666666666, 0.0023135251651203942),
 (-5.6, 0.003684239899435989),
 (-5.133333333333333, 0.005862302196338335),
 (-4.666666666666666, 0.009315959345066693),
 (-4.2, 0.014774031693273055),
 (-3.7333333333333334, 0.023354516476977092),
 (-3.2666666666666666, 0.03673259067202974),
 (-2.8, 0.057324175898868755),
 (-2.333333333333333, 0.08839967720705845),
 (-1.8666666666666663, 0.1339278883240737),
 (-1.4000000000000004, 0.1978161114414182),
 (-0.9333333333333336, 0.28224894304225995),
 (-0.4666666666666668, 0.3854055017324505),
 (0.0, 0.5),
 (0.4666666666666668, 0.6145944982675495),
 (0.9333333333333336, 0.71775105695774),
 (1.4000000000000004, 0.8021838885585818),
 (1.8666666666666671, 0.8660721116759263),
 (2.3333333333333334, 0.9116003227929417),
 (2.8000000000000007, 0.9426758241011313),
 (3.2666666666666675, 0.9632674093279703),
 (3.7333333333333343, 0.9766454835230229),
 (4.199999999999999, 0.9852259683067269),
 (4.666666666666666, 0.9906840406549333),
 (5.133333333333333, 0.9941376978036617),
 (5.6, 0.9963157601005641),
 (6.066666666666666, 0.9976864748348797),
 (6.533333333333333, 0.9985479608899902),
 (7.0, 0.9990889488055994)]

```

```
# Вывод графика и осей
```

```
plt.figure(figsize=(10, 5))
```

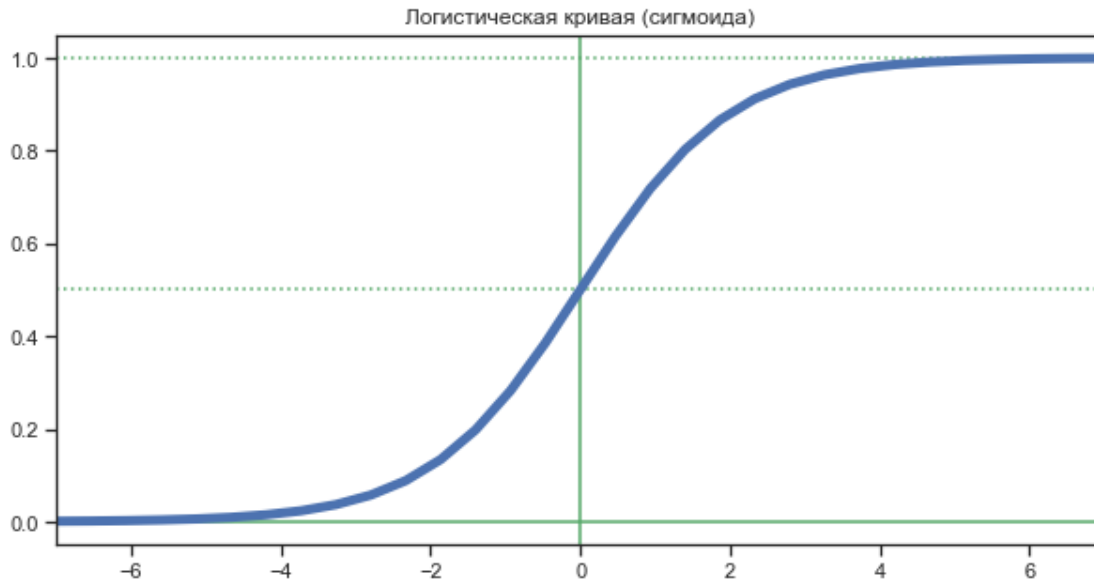
```
plt.plot([-7, 7], [0, 0], "g-")
```

```
plt.plot([-7, 7], [0.5, 0.5], "g:")
```

```
plt.plot([-7, 7], [1, 1], "g:")
```

```
plt.plot([0, 0], [-1.1, 1.1], "g-")
```

```
plt.plot(x, y, "b-", linewidth=5)
plt.axis([-7, 7, -0.05, 1.05])
plt.title('Логистическая кривая (сигмоида)')
plt.show()
```



```
# Подготовка данных
data1 = pd.read_csv('melb_data.csv', sep=",")
iris = load_iris()
iris_x_ds = pd.DataFrame(data=iris['data'],
columns=iris['feature_names'])
iris_x_ds_lr = iris_x_ds[['petal length (cm)', 'sepal length (cm)']]
data1['x0'] = 1
iris_x_ds_lr['target'] = iris.target
data1.head()
```

C:\Users\7272~1\AppData\Local\Temp\ipykernel\_17792\75802874.py:7:  
SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation:  
[https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)  
iris\_x\_ds\_lr['target'] = iris.target

	Suburb	Address	Rooms	Type	Price	Method	SellerG
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin

3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson

	Date	Distance	Postcode	...	Car	Landsize	BuildingArea
YearBuilt \							
0	3/12/2016	2.5	3067.0	...	1.0	202.0	NaN
1	4/02/2016	2.5	3067.0	...	0.0	156.0	79.0
2	4/03/2017	2.5	3067.0	...	0.0	134.0	150.0
3	4/03/2017	2.5	3067.0	...	1.0	94.0	NaN
4	4/06/2016	2.5	3067.0	...	2.0	120.0	142.0

	CouncilArea	Lattitude	Longitude	Regionname
Propertycount x0				
0	Yarra	-37.7996	144.9984	Northern Metropolitan
1	Yarra	-37.8079	144.9934	Northern Metropolitan
2	Yarra	-37.8093	144.9944	Northern Metropolitan
3	Yarra	-37.7969	144.9969	Northern Metropolitan
4	Yarra	-37.8072	144.9941	Northern Metropolitan

[5 rows x 22 columns]

```
def convert_target_to_binary(array:data1['Car'], Bathroom:int) ->
data1['Car']:
```

```
# Если целевой признак совпадает с указанным, то 1 иначе 0
res = [1 if x==Bathroom else 0 for x in array]
return res
```

```
bin_iris_y = convert_target_to_binary(data1['Bathroom'], 1)
```

```
data1['target_bin'] = bin_iris_y
data1.head()
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin

2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson

	Date	Distance	Postcode	...	Landsize	BuildingArea
YearBuilt \						
0	3/12/2016	2.5	3067.0	...	202.0	NaN
1	4/02/2016	2.5	3067.0	...	156.0	79.0
2	4/03/2017	2.5	3067.0	...	134.0	150.0
3	4/03/2017	2.5	3067.0	...	94.0	NaN
4	4/06/2016	2.5	3067.0	...	120.0	142.0

	CouncilArea	Lattitude	Longitude	Regionname
Propertycount \				
0	Yarra	-37.7996	144.9984	Northern Metropolitan
1	Yarra	-37.8079	144.9934	Northern Metropolitan
2	Yarra	-37.8093	144.9944	Northern Metropolitan
3	Yarra	-37.7969	144.9969	Northern Metropolitan
4	Yarra	-37.8072	144.9941	Northern Metropolitan

	x0	target_bin
0	1	1
1	1	1
2	1	0
3	1	0
4	1	1

[5 rows x 23 columns]

# Визуализация данных

colors = "gb"

#X\_viz = iris.data[:, [1,2]]

X\_viz = data1[['Rooms', 'Bathroom']].values

y\_viz = data1['target\_bin'].values

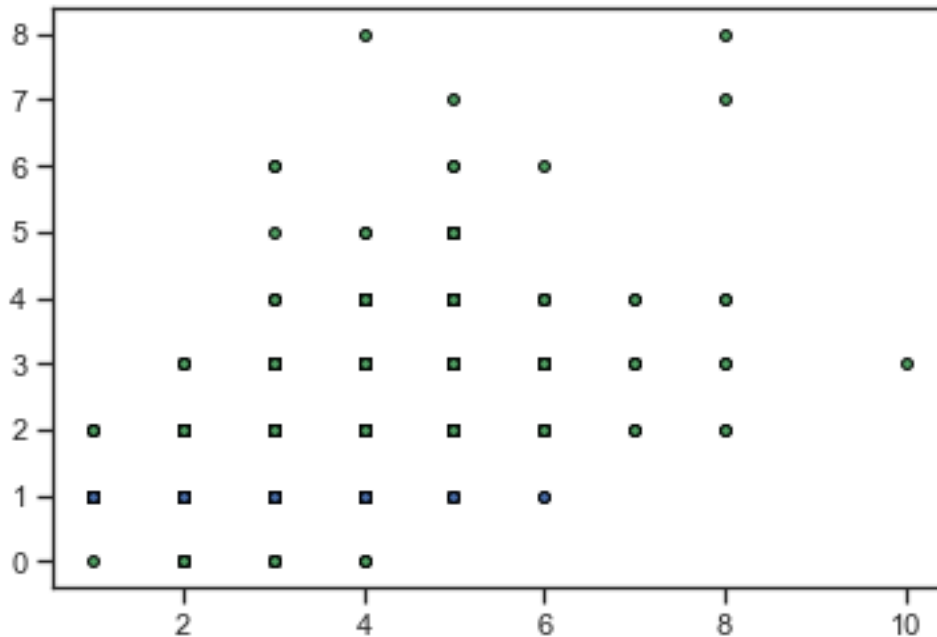
n\_classes = len(np.unique(y\_viz))

for i, color in zip(range(n\_classes), colors):

```

idx = np.where(y_viz == i)
plt.scatter(X_viz[idx, 0], X_viz[idx, 1],
            c=color,
            cmap=plt.cm.RdYlBu,
            edgecolor='black', s=15)
plt.show()

```



```

# Реализация градиентного спуска
def sigmoid(x):
    """
    Функция - сигмоида
    """
    return 1 / (1 + np.exp(-x))

def proba(b, x):
    """
    Вероятность единичного класса
    """
    return sigmoid(np.dot(x,b))

def cost_function(b, x, y):
    """
    Функция потерь
    """
    k = x.shape[0]
    res = -(1 / k) * np.sum(
        y * np.log(proba(b, x))
        + (1 - y) * np.log(1 - proba(b, x)))
    return res

```

```

def gradient(b, x, y):
    """
    Определение градиента
    """
    k = x.shape[0]
    res = (1 / k) * np.dot(
        x.T, (proba(b, x) - y))

def optimize_lr(x, y, b):
    """
    Для оптимизации используется функция
    scipy.optimize.fmin_tnc
    """
    opt_weights = fmin_tnc(
        func=cost_function,
        x0=b,
        fprime=gradient,
        approx_grad=True,
        args=(x, y))
    return opt_weights[0]

opt_x = data1[['x0', 'Rooms', 'Bathroom']].values
opt_x[:5]

array([[1., 2., 1.],
       [1., 2., 1.],
       [1., 3., 2.],
       [1., 3., 2.],
       [1., 4., 1.]])

opt_y = data1['target_bin']
opt_y[:5]

0      1
1      1
2      0
3      0
4      1
Name: target_bin, dtype: int64

b_init = np.zeros(3)
b_init

array([0., 0., 0.])

b_res = optimize_lr(opt_x, opt_y, b_init)
b_res

array([14.89013702, -0.27069669, -9.50234537])

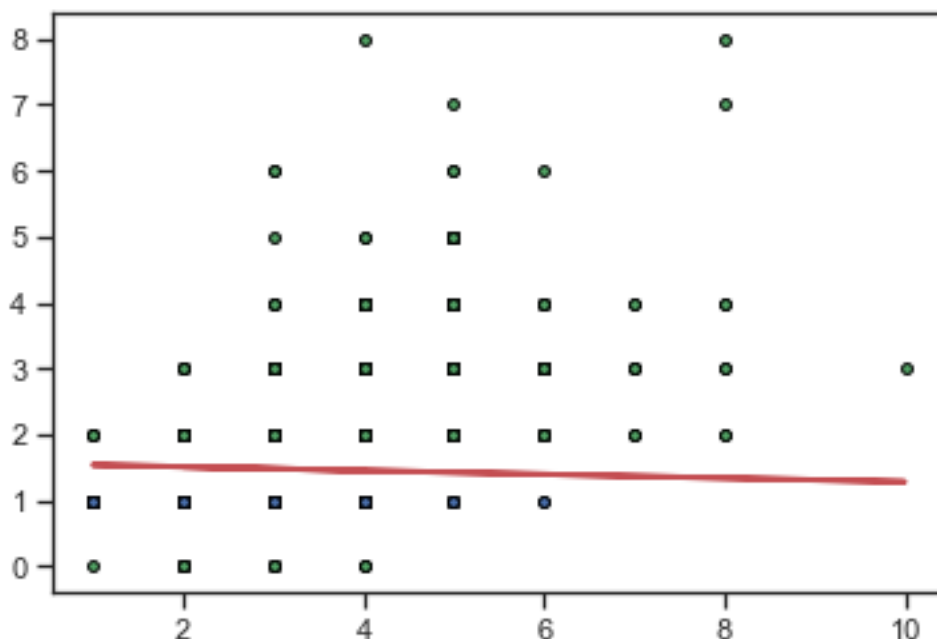
def vis_lr(b):
    """

```

### Визуализация результата

```
'''
colors = "gb"
X_viz = data1[['Rooms', 'Bathroom']].values
y_viz = data1['target_bin'].values
n_classes = len(np.unique(y_viz))
for i, color in zip(range(n_classes), colors):
    idx = np.where(y_viz == i)
    plt.scatter(X_viz[idx, 0], X_viz[idx, 1],
                c=color,
                cmap=plt.cm.RdYlBu,
                edgecolor='black', s=15)
t1 = data1['Rooms'].values
t2 = -((b[0]+np.dot(b[1], t1))/b[2])
plt.plot(t1, t2, 'r', linewidth=2.0)
plt.show()
```

vis\_lr(b\_res)



```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn.datasets import load_iris, load_boston
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import confusion_matrix
```

```

from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.datasets import make_blobs, make_circles
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR,
NuSVR, LinearSVR
import seaborn as sns
from sklearn.neighbors import KNeighborsRegressor,
KNeighborsClassifier
from sklearn.metrics import plot_confusion_matrix
from collections import Counter
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
%matplotlib inline
sns.set(style="ticks")
data = pd.read_csv('melb_data.csv', sep=",")
data_2 = data.dropna(axis=0, how='any')
(data.shape, data_2.shape)

((13580, 21), (6196, 21))

```

*# формирование второго целевого признака для классификации*

```

data1 = {'a': [], 'b': []}
data2 = {'c': []}
df = pd.DataFrame(data1)
df1 = pd.DataFrame(data2)
iris = load_iris()
df['a'] = data_2['Rooms']
df['b'] = data_2['Bathroom']
df1['c'] = data_2['Car']
df = df.astype({'a': int, 'b': int})
df1 = df1.astype({'c': int})
X = df.to_numpy()
y = df1.to_numpy()
data

```

	Suburb	Address	Rooms	Type	Price	
Method \						
0	Abbotsford	85 Turner St	2	h	1480000.0	S
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S
2	Abbotsford	5 Charles St	3	h	1465000.0	SP
3	Abbotsford	40 Federation La	3	h	850000.0	PI
4	Abbotsford	55a Park St	4	h	1600000.0	VB



...	...	...	...	...	...	...
13575	Wheelers Hill	12 Strada Cr	4	h	1245000.0	S
13576	Williamstown	77 Merrett Dr	3	h	1031000.0	SP
13577	Williamstown	83 Power St	3	h	1170000.0	S
13578	Williamstown	96 Verdon St	4	h	2500000.0	PI
13579	Yarraville	6 Agnes St	4	h	1285000.0	SP

	SellerG	Date	Distance	Postcode	...	Bathroom	Car
Landsize \							
0	Biggin	3/12/2016	2.5	3067.0	...	1.0	1.0
202.0							
1	Biggin	4/02/2016	2.5	3067.0	...	1.0	0.0
156.0							
2	Biggin	4/03/2017	2.5	3067.0	...	2.0	0.0
134.0							
3	Biggin	4/03/2017	2.5	3067.0	...	2.0	1.0
94.0							
4	Nelson	4/06/2016	2.5	3067.0	...	1.0	2.0
120.0							
...	...	...	...	...	...	...	...
...							
13575	Barry	26/08/2017	16.7	3150.0	...	2.0	2.0
652.0							
13576	Williams	26/08/2017	6.8	3016.0	...	2.0	2.0
333.0							
13577	Raine	26/08/2017	6.8	3016.0	...	2.0	4.0
436.0							
13578	Sweeney	26/08/2017	6.8	3016.0	...	1.0	5.0
866.0							
13579	Village	26/08/2017	6.3	3013.0	...	1.0	1.0
362.0							

	BuildingArea	YearBuilt	CouncilArea	Lattitude	Longtitude	\
0	NaN	NaN	Yarra	-37.79960	144.99840	
1	79.0	1900.0	Yarra	-37.80790	144.99340	
2	150.0	1900.0	Yarra	-37.80930	144.99440	
3	NaN	NaN	Yarra	-37.79690	144.99690	
4	142.0	2014.0	Yarra	-37.80720	144.99410	
...	...	...	...	...	...	
13575	NaN	1981.0	NaN	-37.90562	145.16761	
13576	133.0	1995.0	NaN	-37.85927	144.87904	
13577	NaN	1997.0	NaN	-37.85274	144.88738	
13578	157.0	1920.0	NaN	-37.85908	144.89299	

```
13579          112.0      1920.0          NaN -37.81188      144.88449
```

```
          Regionname Propertycount
0      Northern Metropolitan      4019.0
1      Northern Metropolitan      4019.0
2      Northern Metropolitan      4019.0
3      Northern Metropolitan      4019.0
4      Northern Metropolitan      4019.0
...
13575  South-Eastern Metropolitan      7392.0
13576      Western Metropolitan      6380.0
13577      Western Metropolitan      6380.0
13578      Western Metropolitan      6380.0
13579      Western Metropolitan      6543.0
```

```
[13580 rows x 21 columns]
```

```
# Разделение выборки на обучающую и тестовую
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.5,
                                                    random_state=1)
```

```
cl1 = LogisticRegression()
```

```
cl1.fit(X_train, y_train)
```

```
C:\Users\Админ\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\utils\validation.py:993: DataConversionWarning: A
column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples, ), for example using ravel().
```

```
    y = column_or_1d(y, warn=True)
```

```
C:\Users\Админ\AppData\Local\Programs\Python\Python39\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
    n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```

```
pred_test = cl1.predict(X_test)
```

```
pred_test
```

```
array([2, 2, 2, ..., 2, 2, 2])
```

```

pred_test_proba = cl1.predict_proba(X_test)
pred_test_proba[:10]

array([[1.32013711e-02, 2.96290842e-01, 6.03483202e-01, 3.96693291e-
02,
        3.39316368e-02, 4.11248106e-03, 1.96059893e-03, 6.99851374e-
03,
        3.52025237e-04],
[2.57712450e-02, 1.55746500e-01, 6.44845065e-01, 8.68156869e-
02,
        7.01437630e-02, 8.18680938e-03, 6.15007736e-03, 1.92599816e-
03,
        4.14855267e-04],
[4.10886501e-02, 3.84717424e-01, 4.91431234e-01, 4.50435637e-
02,
        2.90271381e-02, 3.32958632e-03, 2.84935269e-03, 1.99862602e-
03,
        5.14425780e-04],
[4.10886501e-02, 3.84717424e-01, 4.91431234e-01, 4.50435637e-
02,
        2.90271381e-02, 3.32958632e-03, 2.84935269e-03, 1.99862602e-
03,
        5.14425780e-04],
[2.78185967e-05, 2.31238831e-03, 6.50181267e-01, 4.99576129e-
02,
        1.96945112e-01, 2.96621145e-02, 1.65387858e-03, 6.92445740e-
02,
        1.52346602e-05],
[1.14282381e-01, 6.91603435e-01, 1.70936890e-01, 1.48735030e-
02,
        5.75952787e-03, 6.14497816e-04, 1.07531917e-03, 3.31971672e-
04,
        5.22475421e-04],
[2.57712450e-02, 1.55746500e-01, 6.44845065e-01, 8.68156869e-
02,
        7.01437630e-02, 8.18680938e-03, 6.15007736e-03, 1.92599816e-
03,
        4.14855267e-04],
[2.57712450e-02, 1.55746500e-01, 6.44845065e-01, 8.68156869e-
02,
        7.01437630e-02, 8.18680938e-03, 6.15007736e-03, 1.92599816e-
03,
        4.14855267e-04],
[1.32013711e-02, 2.96290842e-01, 6.03483202e-01, 3.96693291e-
02,
        3.39316368e-02, 4.11248106e-03, 1.96059893e-03, 6.99851374e-
03,
        3.52025237e-04],
[2.57712450e-02, 1.55746500e-01, 6.44845065e-01, 8.68156869e-
02,

```

```
03,
    7.01437630e-02, 8.18680938e-03, 6.15007736e-03, 1.92599816e-
    4.14855267e-04]])
```

```
# Вероятность принадлежности к 0 классу
```

```
[round(x, 4) for x in pred_test_proba[:10,0]]
```

```
[0.0132, 0.0258, 0.0411, 0.0411, 0.0, 0.1143, 0.0258, 0.0258, 0.0132,
0.0258]
```

```
# Вероятность принадлежности к 1 классу
```

```
[round(x, 4) for x in pred_test_proba[:10,1]]
```

```
[0.2963,
0.1557,
0.3847,
0.3847,
0.0023,
0.6916,
0.1557,
0.1557,
0.2963,
0.1557]
```

```
# Сумма вероятностей равна 1
```

```
pred_test_proba[:10,0] + pred_test_proba[:10,1]
```

```
array([0.30949221, 0.18151775, 0.42580607, 0.42580607, 0.00234021,
       0.80588582, 0.18151775, 0.18151775, 0.30949221, 0.18151775])
```

```
accuracy_score(y_test, pred_test)
```

```
0.6007101355713363
```

```
def accuracy_score_for_classes(
```

```
    y_true: np.ndarray,
```

```
    y_pred: np.ndarray) -> Dict[int, float]:
```

```
    """
```

```
    Вычисление метрики ассигасу для каждого класса
```

```
    y_true - истинные значения классов
```

```
    y_pred - предсказанные значения классов
```

```
    Возвращает словарь: ключ - метка класса,
```

```
    значение - Ассигасу для данного класса
```

```
    """
```

```
    # Для удобства фильтрации сформируем Pandas DataFrame
```

```
    d = {'t': y_true, 'p': y_pred}
```

```
    df = pd.DataFrame(data=d)
```

```
    # Метки классов
```

```
    classes = np.unique(y_true)
```

```
    # Результирующий словарь
```

```
    res = dict()
```

```
    # Перебор меток классов
```

```

for c in classes:
    # отфильтруем данные, которые соответствуют
    # текущей метке класса в истинных значениях
    temp_data_flt = df[df['t']==c]
    # расчет accuracy для заданной метки класса
    temp_acc = accuracy_score(
        temp_data_flt['t'].values,
        temp_data_flt['p'].values)
    # сохранение результата в словарь
    res[c] = temp_acc
return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))

```