Import biblioteki **TensorFlow** (https://www.tensorflow.org/) z której będziemy korzystali w **uczeniu maszynowym**:

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
df = pd.read_csv('Boston.csv')
print(df)
```

```
     Unnamed: 0     crim    zn  indus  chas    nox     rm   age     dis  rad  \
0             1  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1
1             2  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2
2             3  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2
3             4  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3
4             5  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3
..          ...      ...   ...    ...   ...    ...    ...   ...     ...  ...
501         502  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1
502         503  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1
503         504  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1
504         505  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1
505         506  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1

     tax  ptratio   black  lstat  medv
0    296     15.3  396.90   4.98  24.0
1    242     17.8  396.90   9.14  21.6
2    242     17.8  392.83   4.03  34.7
3    222     18.7  394.63   2.94  33.4
4    222     18.7  396.90   5.33  36.2
..   ...      ...     ...    ...   ...
501  273     21.0  391.99   9.67  22.4
502  273     21.0  396.90   9.08  20.6
503  273     21.0  396.90   5.64  23.9
504  273     21.0  393.45   6.48  22.0
505  273     21.0  396.90   7.88  11.9

[506 rows x 15 columns]
```

```
df.head()
```

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7

```
nox=df.iloc[:,6]
```

```
nox
```

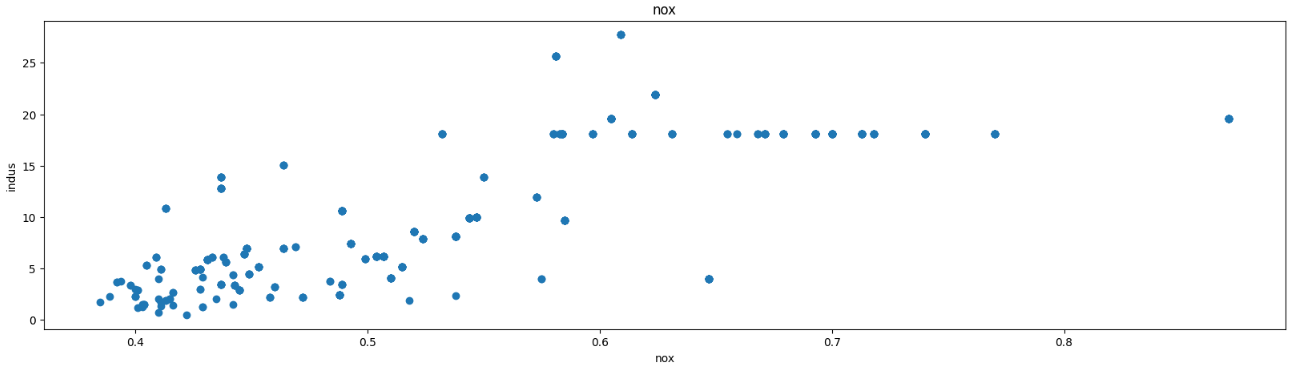
```
0      6.575
1      6.421
2      7.185
3      6.998
4      7.147
       ...
501    6.593
502    6.120
503    6.976
504    6.794
505    6.030
Name: rm, Length: 506, dtype: float64
```

```
indus=df.iloc[:,14]
```

```
indus
```

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: medv, Length: 506, dtype: float64
```

```
plt.figure(figsize=(20, 5))

features = ['nox']
target = df['indus']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('indus')

plt.show()
```




```
nox

     0      6.575
     1      6.421
     2      7.185
     3      6.998
     4      7.147
            ...
     501    6.593
     502    6.120
     503    6.976
     504    6.794
     505    6.030
     Name: rm, Length: 506, dtype: float64

df.corr()
```

	Unnamed: 0	crim	zn	indus	chas	nox	rm	
Unnamed: 0	1.000000	0.407407	-0.103393	0.399439	-0.003759	0.398736	-0.079971	0.
crim	0.407407	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.

```
df.corr()
```

	Unnamed: 0	crim	zn	indus	chas	nox	rm	
Unnamed: 0	1.000000	0.407407	-0.103393	0.399439	-0.003759	0.398736	-0.079971	0.
crim	0.407407	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.
zn	-0.103393	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.
indus	0.399439	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.
chas	-0.003759	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.
nox	0.398736	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.
rm	-0.079971	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.
age	0.203784	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.
dis	-0.302211	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.
rad	0.686002	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.
tax	0.666626	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.
ptratio	0.291074	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.
black	-0.295041	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.
lstat	0.258465	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.

```
real_x = np.array(nox)
real_y = np.array(indus)
```

```
real_x
```

```
array([6.575, 6.421, 7.185, 6.998, 7.147, 6.43 , 6.012, 6.172, 5.631,
       6.004, 6.377, 6.009, 5.889, 5.949, 6.096, 5.834, 5.935, 5.99 ,
       5.456, 5.727, 5.57 , 5.965, 6.142, 5.813, 5.924, 5.599, 5.813,
       6.047, 6.495, 6.674, 5.713, 6.072, 5.95 , 5.701, 6.096, 5.933,
       5.841, 5.85 , 5.966, 6.595, 7.024, 6.77 , 6.169, 6.211, 6.069,
       5.682, 5.786, 6.03 , 5.399, 5.602, 5.963, 6.115, 6.511, 5.998,
       5.888, 7.249, 6.383, 6.816, 6.145, 5.927, 5.741, 5.966, 6.456,
       6.762, 7.104, 6.29 , 5.787, 5.878, 5.594, 5.885, 6.417, 5.961,
       6.065, 6.245, 6.273, 6.286, 6.279, 6.14 , 6.232, 5.874, 6.727,
       6.619, 6.302, 6.167, 6.389, 6.63 , 6.015, 6.121, 7.007, 7.079,
       6.417, 6.405, 6.442, 6.211, 6.249, 6.625, 6.163, 8.069, 7.82 ,
       7.416, 6.727, 6.781, 6.405, 6.137, 6.167, 5.851, 5.836, 6.127,
       6.474, 6.229, 6.195, 6.715, 5.913, 6.092, 6.254, 5.928, 6.176,
```

```
       6.021, 5.872, 5.731, 5.87 , 6.004, 5.961, 5.856, 5.879, 5.986,
       5.613, 5.693, 6.431, 5.637, 6.458, 6.326, 6.372, 5.822, 5.757,
       6.335, 5.942, 6.454, 5.857, 6.151, 6.174, 5.019, 5.403, 5.468,
       4.903, 6.13 , 5.628, 4.926, 5.186, 5.597, 6.122, 5.404, 5.012,
       5.709, 6.129, 6.152, 5.272, 6.943, 6.066, 6.51 , 6.25 , 7.489,
       7.802, 8.375, 5.854, 6.101, 7.929, 5.877, 6.319, 6.402, 5.875,
       5.88 , 5.572, 6.416, 5.859, 6.546, 6.02 , 6.315, 6.86 , 6.98 ,
       7.765, 6.144, 7.155, 6.563, 5.604, 6.153, 7.831, 6.782, 6.556,
       7.185, 6.951, 6.739, 7.178, 6.8  , 6.604, 7.875, 7.287, 7.107,
       7.274, 6.975, 7.135, 6.162, 7.61 , 7.853, 8.034, 5.891, 6.326,
       5.783, 6.064, 5.344, 5.96 , 5.404, 5.807, 6.375, 5.412, 6.182,
       5.888, 6.642, 5.951, 6.373, 6.951, 6.164, 6.879, 6.618, 8.266,
       8.725, 8.04 , 7.163, 7.686, 6.552, 5.981, 7.412, 8.337, 8.247,
       6.726, 6.086, 6.631, 7.358, 6.481, 6.606, 6.897, 6.095, 6.358,
       6.393, 5.593, 5.605, 6.108, 6.226, 6.433, 6.718, 6.487, 6.438,
       6.957, 8.259, 6.108, 5.876, 7.454, 8.704, 7.333, 6.842, 7.203,
       7.52 , 8.398, 7.327, 7.206, 5.56 , 7.014, 8.297, 7.47 , 5.92 ,
       5.856, 6.24 , 6.538, 7.691, 6.758, 6.854, 7.267, 6.826, 6.482,
       6.812, 7.82 , 6.968, 7.645, 7.923, 7.088, 6.453, 6.23 , 6.209,
       6.315, 6.565, 6.861, 7.148, 6.63 , 6.127, 6.009, 6.678, 6.549,
       5.79 , 6.345, 7.041, 6.871, 6.59 , 6.495, 6.982, 7.236, 6.616,
       7.42 , 6.849, 6.635, 5.972, 4.973, 6.122, 6.023, 6.266, 6.567,
       5.705, 5.914, 5.782, 6.382, 6.113, 6.426, 6.376, 6.041, 5.708,
       6.415, 6.431, 6.312, 6.083, 5.868, 6.333, 6.144, 5.706, 6.031,
       6.316, 6.31 , 6.037, 5.869, 5.895, 6.059, 5.985, 5.968, 7.241,
       6.54 , 6.696, 6.874, 6.014, 5.898, 6.516, 6.635, 6.939, 6.49 ,
       6.579, 5.884, 6.728, 5.663, 5.936, 6.212, 6.395, 6.127, 6.112,
       6.398, 6.251, 5.362, 5.803, 8.78 , 3.561, 4.963, 3.863, 4.97 ,
       6.683, 7.016, 6.216, 5.875, 4.906, 4.138, 7.313, 6.649, 6.794,
       6.38 , 6.223, 6.968, 6.545, 5.536, 5.52 , 4.368, 5.277, 4.652,
       5.   , 4.88 , 5.39 , 5.713, 6.051, 5.036, 6.193, 5.887, 6.471,
       6.405, 5.747, 5.453, 5.852, 5.987, 6.343, 6.404, 5.349, 5.531,
       5.683, 4.138, 5.608, 5.617, 6.852, 5.757, 6.657, 4.628, 5.155,
       4.519, 6.434, 6.782, 5.304, 5.957, 6.824, 6.411, 6.006, 5.648,
       6.103, 5.565, 5.896, 5.837, 6.202, 6.193, 6.38 , 6.348, 6.833,
       6.425, 6.436, 6.208, 6.629, 6.461, 6.152, 5.935, 5.627, 5.818,
       6.406, 6.219, 6.485, 5.854, 6.459, 6.341, 6.251, 6.185, 6.417,
       6.749, 6.655, 6.297, 7.393, 6.728, 6.525, 5.976, 5.936, 6.301,
       6.081, 6.701, 6.376, 6.317, 6.513, 6.209, 5.759, 5.952, 6.003,
       5.926, 5.713, 6.167, 6.229, 6.437, 6.98 , 5.427, 6.162, 6.484,
       5.304, 6.185, 6.229, 6.242, 6.75 , 7.061, 5.762, 5.871, 6.312,
       6.114, 5.905, 5.454, 5.414, 5.093, 5.983, 5.983, 5.707, 5.926,
       5.67 , 5.39 , 5.794, 6.019, 5.569, 6.027, 6.593, 6.12 , 6.976,
       6.794, 6.03 ])
```

```
real_y
```

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
```

```
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

Batch Stochastic Gradient Descent - wykorzystujemy **cały zbiór danych**

Definicja błędu:

```
def loss_fn(real_y, pred_y):
    return tf.reduce_mean((real_y - pred_y)**2)
```

```
import random
```

```
Loss = []
epochs = 2000
learning_rate = 0.02

a = tf.Variable(random.random())
b = tf.Variable(random.random())

for _ in range(epochs):

  with tf.GradientTape() as tape:
    pred_y = a * real_x + b
    #print(pred_y)
    loss = loss_fn(real_y, pred_y)
    Loss.append(loss.numpy())
    grad_a, grad_b = tape.gradient(loss,(a, b))

  a.assign_sub(learning_rate*grad_a)
  b.assign_sub(learning_rate*grad_b)
```

```
np.max(Loss),np.min(Loss)
```

```
(318.53137, 45.89997)
```

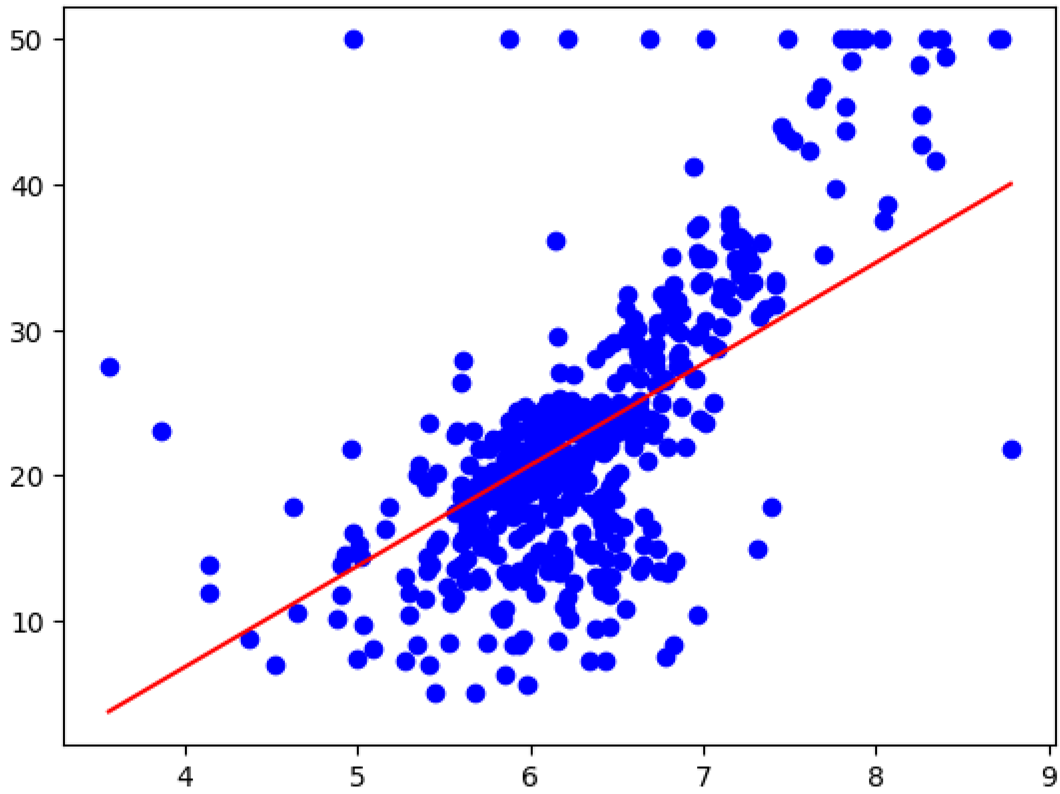
```
print(a.numpy())
print(b.numpy())
```

```
6.955526
-21.015951
```

```
plt.scatter(np.arange(epochs),Loss)
plt.show()
```

```
max = np.max(nox)
min = np.min(nox)
```

```
X = np.linspace(min, max, num=10)
plt.plot(X,a.numpy()*X+b.numpy(),c='r')
plt.scatter(nox,indus,c="b")
plt.show()
```



Mini-batch Stochastic Gradient Descent - wykorzystujemy **część zbióru danych**

Definiujemy tablicę:

```
arr = np.arange(10)
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Mieszamy zawartość tablicy:

```
np.random.shuffle(arr)
arr
```

```
array([6, 8, 4, 0, 1, 2, 5, 7, 9, 3])
```

Funkcja do przetestowania:

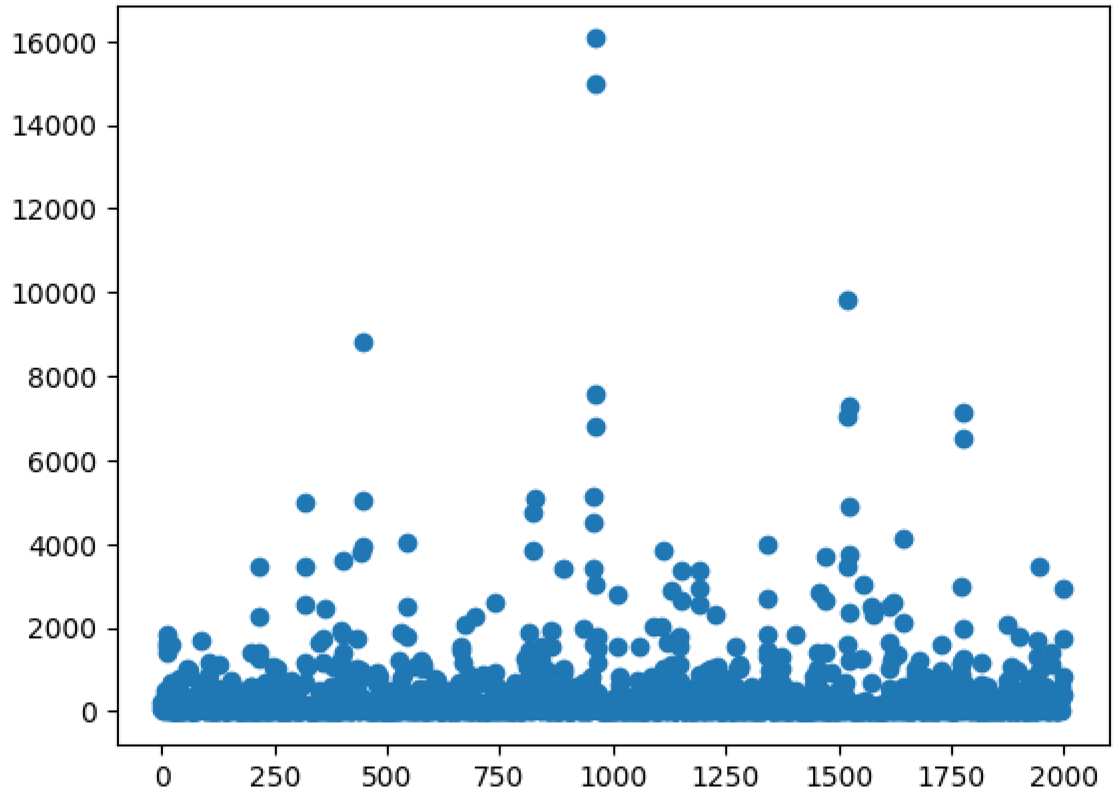
```
def subset_dataset(x_dataset, y_dataset, subset_size):
    arr = np.arange(len(x_dataset))
    np.random.shuffle(arr)
    x_train = x_dataset[arr[0:subset_size]]
    y_train = y_dataset[arr[0:subset_size]]
    return x_train,y_train
```

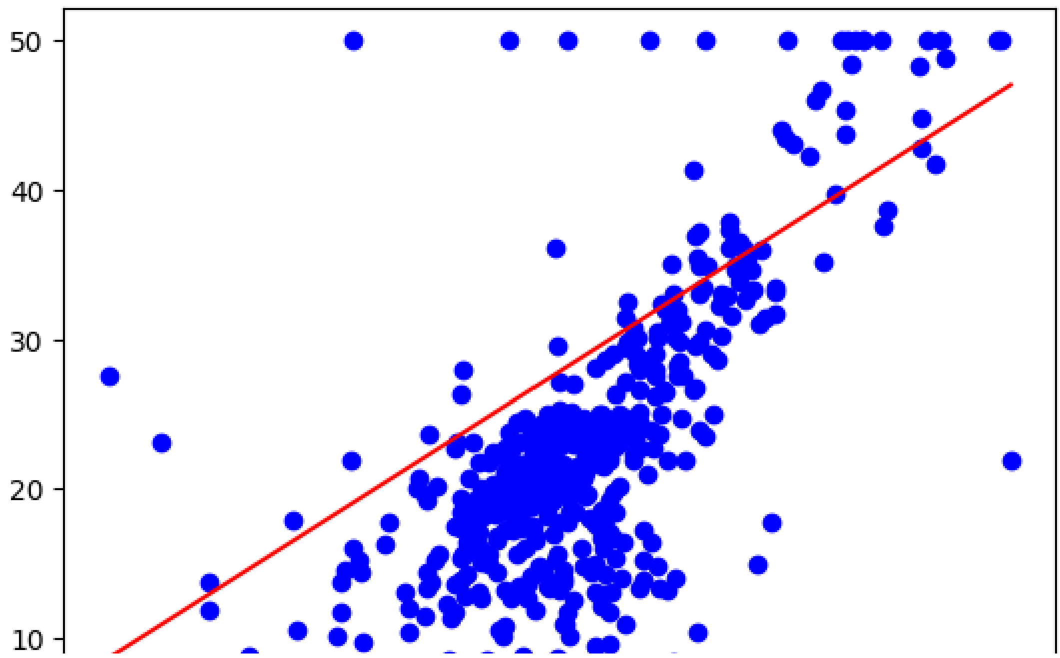
Uzupełnik poniższy kod, tak aby możliwe było testowanie różnych wielkości próbki treningowwej.

```
def mini_batch_stochastic_gradient_descent(batch_size):
  Loss = []
  epochs = 2000
  learning_rate = 0.02
  batch_size = batch_size     #wielkość zbioru wykorzystanego do treningu

  a = tf.Variable(random.random())
  b = tf.Variable(random.random())

  for i in range(epochs):

    real_x_batch, real_y_batch = subset_dataset(real_x,real_y,batch_size)

    with tf.GradientTape() as tape:
      pred_y = a * real_x_batch + b
      loss = loss_fn(real_y_batch, pred_y)
      Loss.append(loss.numpy())

    dloss_da, dloss_db = tape.gradient(loss,(a, b))

    a.assign_sub(learning_rate*dloss_da)  #a = a - alpha*dloss_da
    b.assign_sub(learning_rate*dloss_db)  #b = b - alpha*dloss_db
  print("last one loss", str(loss))

  plt.scatter(np.arange(epochs),Loss)
  plt.show()
  max = np.max(nox)
  min = np.min(nox)
  X = np.linspace(min, max, num=10)
  plt.plot(X,a.numpy()*X+b.numpy(),c='r')
  plt.scatter(nox,indus,c="b")
  plt.show()
```

```
mini_batch_stochastic_gradient_descent(1)
```

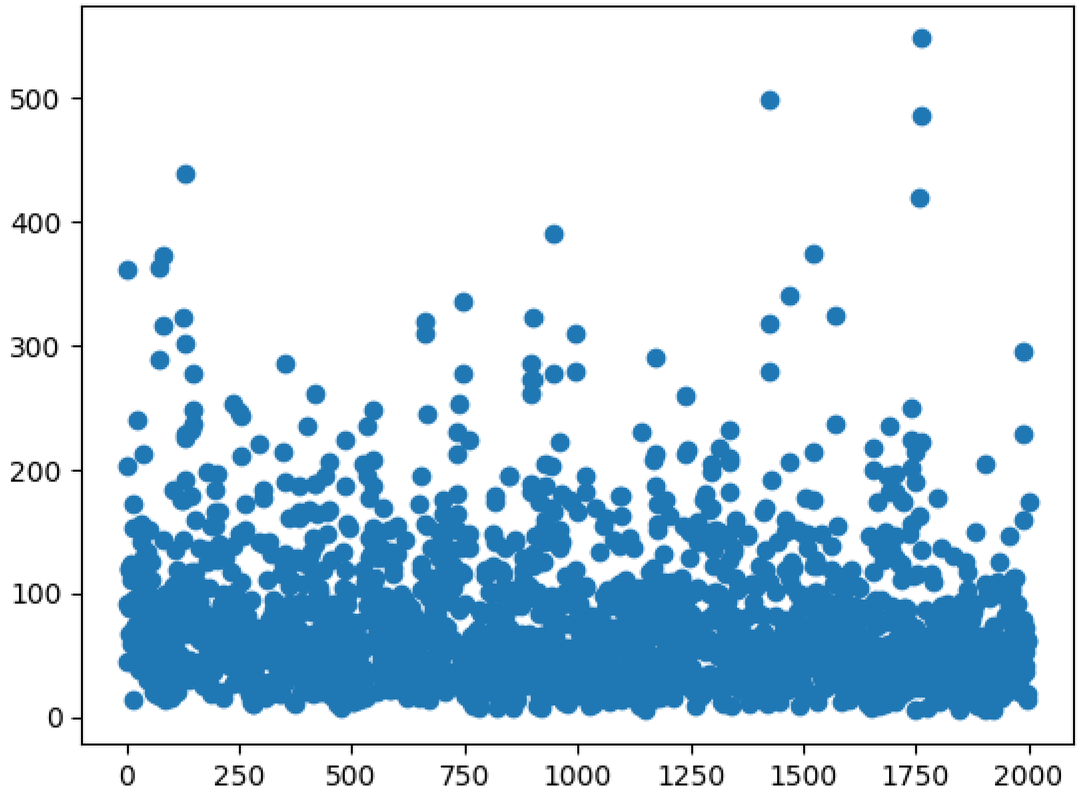
```
last one loss tf.Tensor(428.42133, shape=(), dtype=float32)
```

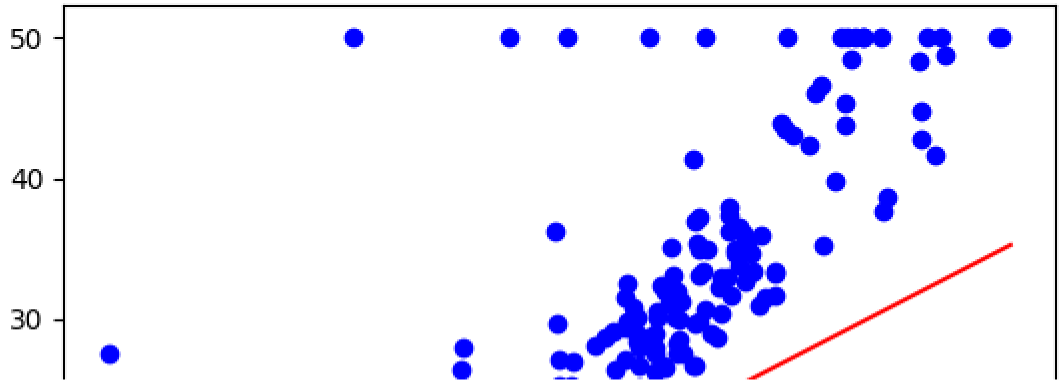





```
mini_batch_stochastic_gradient_descent(10)
```

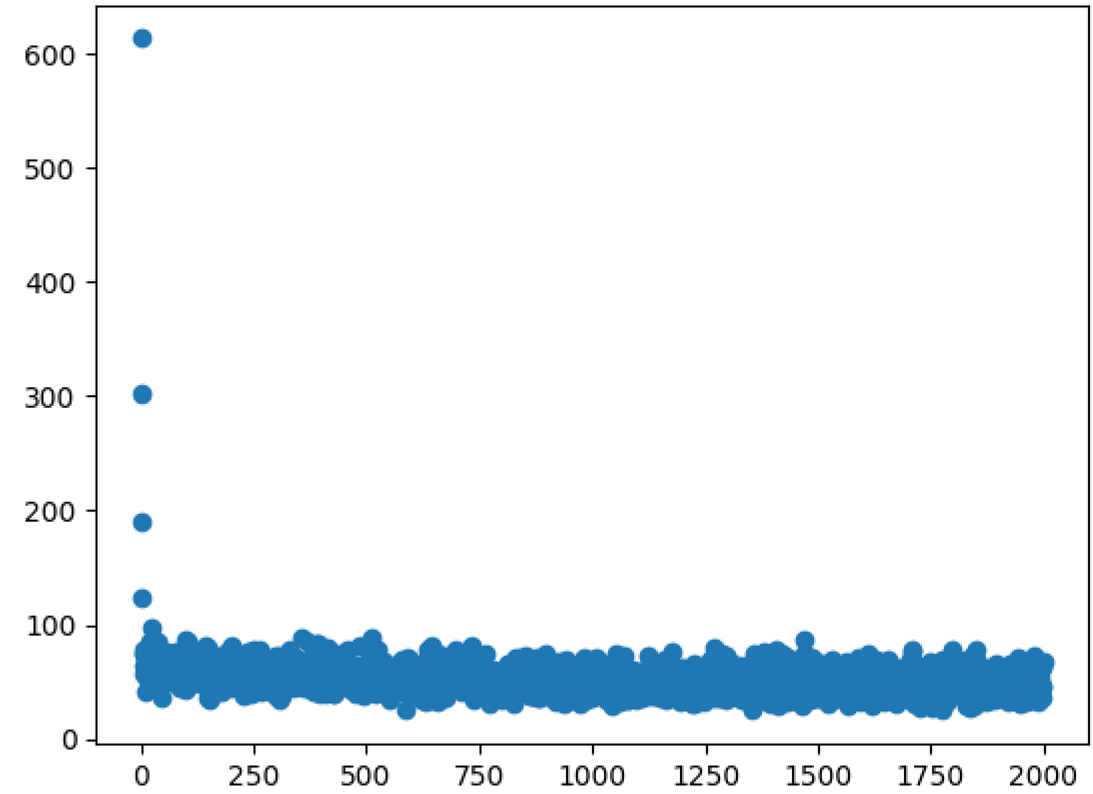
```
last one loss tf.Tensor(173.78526, shape=(), dtype=float32)
```

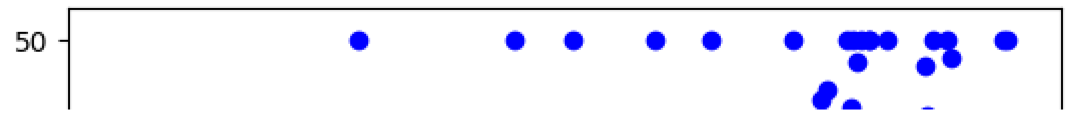





```
mini_batch_stochastic_gradient_descent(100)
```

```
last one loss tf.Tensor(66.96526, shape=(), dtype=float32)
```

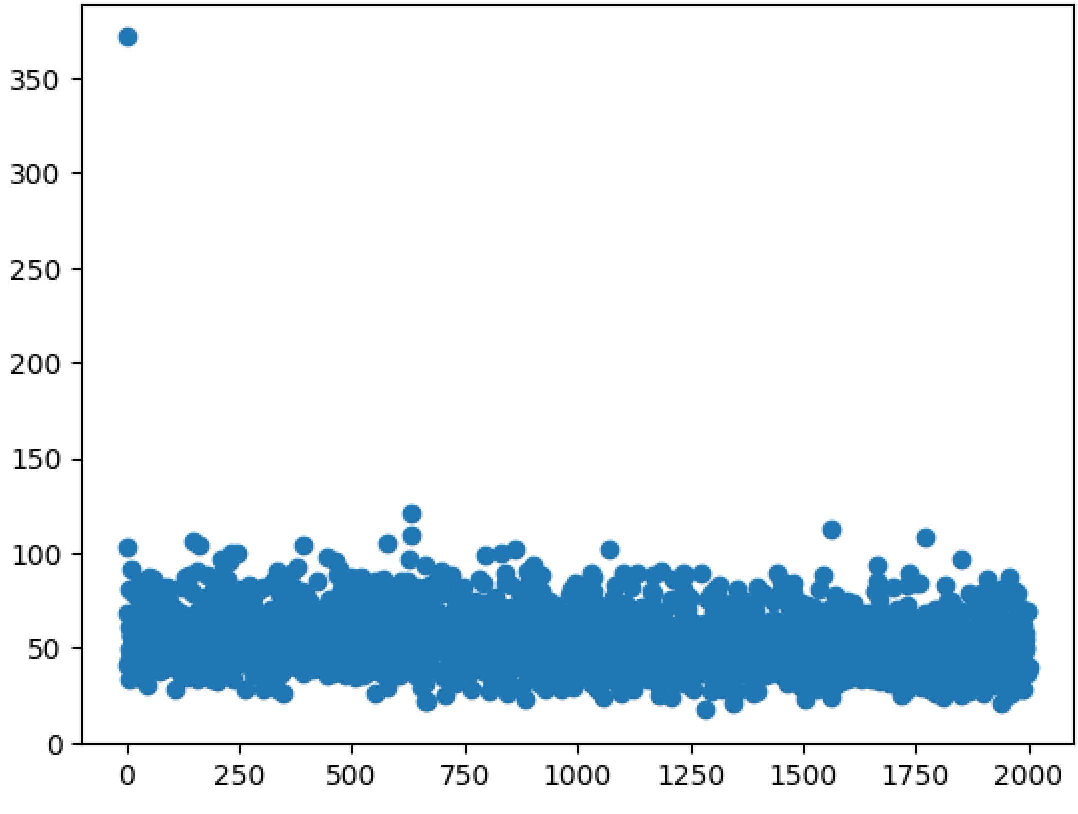




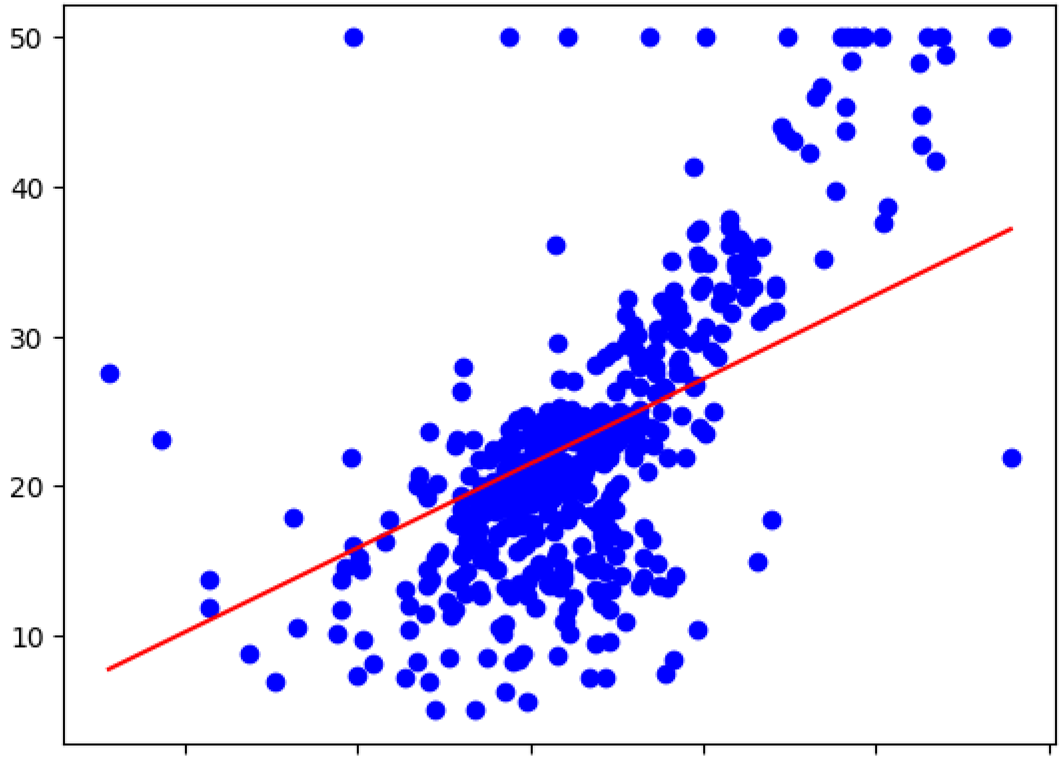
Wykres zmian błędu:

```
#do uzupełnienia

def subset_dataset(x_dataset, y_dataset, subset_size):
    arr = np.arange(len(x_dataset))
    np.random.shuffle(arr)
    x_train = x_dataset[arr[0:subset_size]]
    y_train = y_dataset[arr[0:subset_size]]
    return x_train,y_train

def loss_fn(real_y, pred_y):
    return tf.reduce_mean((real_y - pred_y)**2)
```

```
Loss = []
epochs = 2000
learning_rate = 0.01
batch_size = 50
a = tf.Variable(random.random())
b = tf.Variable(random.random())
for _ in range(epochs):
  real_nox_batch,real_medv_batch = subset_dataset(real_x,real_y,batch_size)
  with tf.GradientTape() as tape:
    pred_medv = a * real_nox_batch + b
    loss = loss_fn(real_medv_batch, pred_medv)
    Loss.append(loss.numpy())

  dloss_da, dloss_db = tape.gradient(loss,(a, b))

  a.assign_sub(learning_rate*dloss_da)  #a = a - alpha*dloss_da
  b.assign_sub(learning_rate*dloss_db)  #b = b - alpha*dloss_db


  plt.scatter(np.arange(epochs),Loss)
  plt.show()
```




```
  max = np.max(nox)
  min = np.min(nox)
  X = np.linspace(min, max, num=10)
  plt.plot(X,a.numpy()*X+b.numpy(),c='r')
  plt.scatter(nox,indus,c="b")
  plt.show()
```



Podsumowanie

Na uczenie modelu ma najwiekszy wpływ użycie batcha (bez batcha jest podawany cały zbiór uczący), małe batche mogą przyspieszyć proces uczenia, ponieważ aktualizacje wag modelu są wykonywane częściej. Dzięki temu wprowadza to pewną losowość w procesie uczenia, pomoga uniknąć utknięcia w minimach lokalnych. Model uczony z minibatchem osiąga lepsze rezultaty jeżeli chodzi o wyniki uczenia(lepiej znaleziona prosta) oraz mniejszy błąd. Model lepiej i szybciej się uczy gdy mini-batch jest większy niż gdy jest on mniejszy. Ponadto na proces uczenia modelu ma wpływ ilość epok. Za mała ilość epok skutkuje niedouczeniem modelu (model nie nauczył się wystarczająco dobrze dostosowywać się do danych treningowych), zaś gdy ilość epok jest zbyt duża następuje przeuczenie modelu (model nieuogulnia zgromadzonej wiedzy tylko "uczy się na pamięć" zbioru treningowego co sprawia, że jest nieskuteczny lub mało skuteczny dla nowych danych). Ostatnim sprawdzonym przeze mnie parametrem jest współczynnik uczenia. Jego zbyt duża wartość rowadzi do skakania wokół minimum globalnego przy czym model go nie osiągnie. W przypadku zastosowania zbyt małej wartości współczynnika uczenia proces uczenia jest bardzo wolny, a model "utyka" w minimach lokalnych.