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

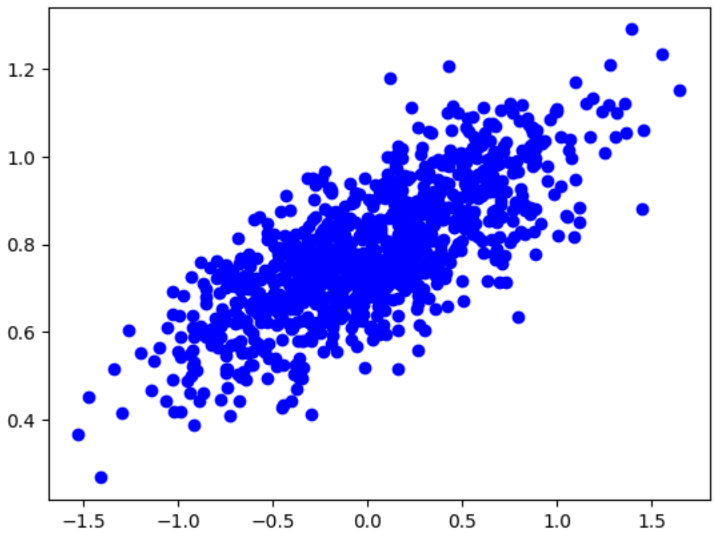
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

```
number_of_points = 1000
x_point = []
y_point = []
```

```
a = 0.22
b = 0.78
```

```
for i in range(number_of_points):
    x = np.random.normal(0.0,0.5)
    y = (a*x+b)+np.random.normal(0.0,0.1)
    x_point.append(x)
    y_point.append(y)
```

```
plt.scatter(x_point,y_point,c='b')
plt.show()
```




```
real_x = np.array(x_point)
real_y = np.array(y_point)
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

Definiujemy model:

```
model = Sequential()
```

Dodajemy **jedną warstwę** (Dense) z **jednym neuronem** (units=1) z **biasem** (use_bias=True) i **liniową funkcją aktywacji** (activation="linear"):

```
model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
```

Definiujemy **optymalizator** i **błąd** (średni błąd kwadratowy - MSE). **Współczynnik uczenia = 0.1**

```
opt = tf.keras.optimizers.SGD(learning_rate=0.1)
```

```
model.compile(loss='MSE',optimizer=opt)
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 1)                 2         
                                                                 
=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Proces **uczenia**:

```
epochs = 1000
h = model.fit(real_x,real_y, verbose=0, epochs=epochs, batch_size=100)
```

```
Loss = h.history['loss']
Loss
```

```
 0.009976331144571304,
 0.009992128238081932,
 0.009996611624956131,
 0.009984729811549187,
 0.009986542165279388,
 0.009986688382923603,
 0.009975828230381012,
 0.009974825195968151,
 0.009980661794543266,
 0.00998423621058464,
 0.009978027082979679,
 0.009987233206629753,
 0.009984341450035572,
 0.00999152660369873,
 0.009998512454330921,
 0.009997275657951832,
 0.009981083683669567,
 0.009973958134651184,
 0.009996677748858929,
 0.009979743510484695,
 0.00997694581747055,
 0.009980532340705395,
 0.009982114657759666,
 0.009985162876546383,
 0.00998216774314642,
 0.009988078847527504,
 0.009986065328121185,
 0.00998198427259922,
 0.009990599006414413,
 0.009999380446970463,
 0.009991451166570187,
 0.009998592548072338,
 0.009995047934353352,
 0.009999320842325687,
 0.00997804757207632,
 0.0099936006590724,
 0.009997382760047913,
 0.009975423105061054,
 0.009991454891860485,
 0.009979229420423508,
 0.009989317506551743,
 0.00997554324567318,
 0.00998847745358944,
 0.00998824741691351,
 0.009990261867642403,
 0.009993656538426876,
 0.009980959817767143,
 0.009979212656617165,
 0.010000030510127544,
 0.010004711337387562,
 0.00998886302113533,
 0.009985696524381638,
 0.009980960749089718,
 0.009991595521569252,
 0.009976213797926903,
 0.01000884547829628,
 0.00997899565845728,
 0.00998386088758707]
```

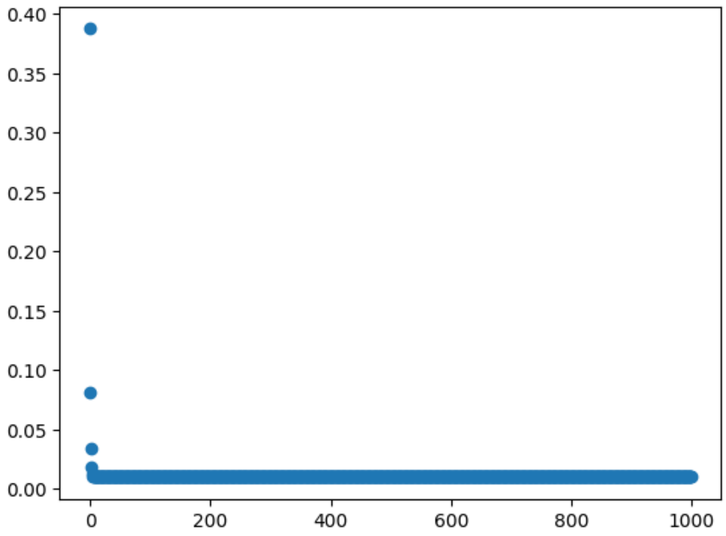
Sprawdźmy jakie są **wartości wag**:

```
weights = model.get_weights()

print(weights[0][0][0])
print(weights[1][0])    #bias
```

```
0.21798924
0.7798659
```

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



Sprawdzenie **modelu**:

```
model.predict([0.6])
```

```
1/1 [==============================] - 0s 94ms/step
array([[0.91065943]], dtype=float32)
```

```
model = Sequential()
```

Verbose 1

```
model = Sequential()
```

```
model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
```

```
opt = tf.keras.optimizers.SGD(learning_rate=0.1)
```

```
model.compile(loss='MSE',optimizer=opt)
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1)                 2

=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 1000
h = model.fit(real_x,real_y, verbose=1, epochs=epochs, batch_size=100)
```

```
Epoch 976/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 977/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 978/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 979/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 980/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 981/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 982/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 983/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 984/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 985/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 986/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 987/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 988/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 989/1000
10/10 [==============================] - 0s 3ms/step - loss: 0.0100
Epoch 990/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 991/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 992/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 993/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 994/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 995/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 996/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 997/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 998/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 999/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
Epoch 1000/1000
10/10 [==============================] - 0s 2ms/step - loss: 0.0100
```

```
Loss = h.history['loss']
Loss
```

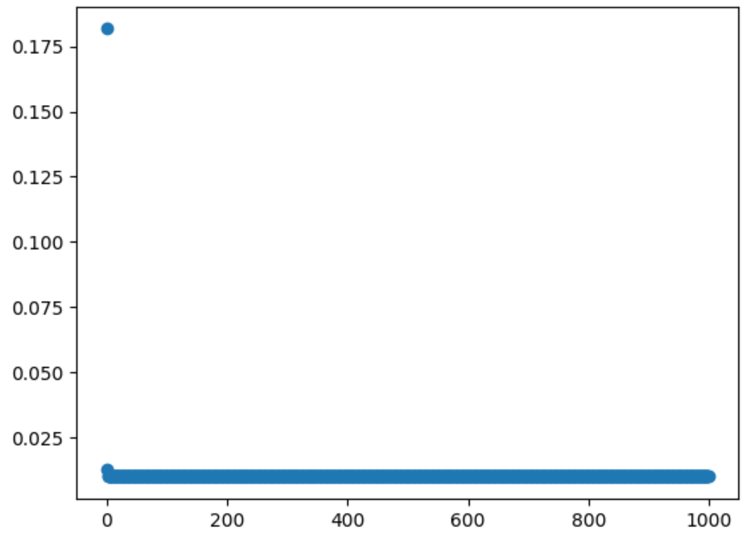
```
0.010005259886384 01,
0.009983773343265057,
0.009987001307308674,
0.009989144280552864,
0.009979978203773499,
0.010005824267864227,
0.009988304227590561,
0.009980930015444756,
0.009987849742174149,
0.009997163899242878,
0.009977073408663273,
0.009979501366615295,
0.009972183033823967,
0.01000935398042202,
0.009984347969293594,
0.010026856325566769,
0.009987297467887402,
0.009978161193430424,
0.009991697035729885,
0.009975031018257141,
0.009981770068407059,
0.009983150288462639,
0.009980679489672184,
```

```
weights = model.get_weights()

print(weights[0][0][0])
print(weights[1][0])    #bias
```

```
0.21753295
0.7798778
```

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




```
model.predict([0.6])
```

```
1/1 [==============================] - 0s 58ms/step
array([[0.9103975]], dtype=float32)
```

```
model = Sequential()
```

Verbose 2

```
model = Sequential()
```

```
model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
```

```
opt = tf.keras.optimizers.SGD(learning_rate=0.1)
```

```
model.compile(loss='MSE',optimizer=opt)
```

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 1)                 2

=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 1000
h = model.fit(real_x,real_y, verbose=2, epochs=epochs, batch_size=100)
```

```
Epoch 1/1000
10/10 - 0s - loss: 0.4559 - 238ms/epoch - 24ms/step
Epoch 2/1000
10/10 - 0s - loss: 0.1056 - 16ms/epoch - 2ms/step
Epoch 3/1000
10/10 - 0s - loss: 0.0413 - 17ms/epoch - 2ms/step
Epoch 4/1000
10/10 - 0s - loss: 0.0205 - 17ms/epoch - 2ms/step
Epoch 5/1000
10/10 - 0s - loss: 0.0135 - 19ms/epoch - 2ms/step
Epoch 6/1000
10/10 - 0s - loss: 0.0112 - 17ms/epoch - 2ms/step
Epoch 7/1000
10/10 - 0s - loss: 0.0104 - 19ms/epoch - 2ms/step
Epoch 8/1000
10/10 - 0s - loss: 0.0101 - 19ms/epoch - 2ms/step
Epoch 9/1000
10/10 - 0s - loss: 0.0100 - 18ms/epoch - 2ms/step
Epoch 10/1000
10/10 - 0s - loss: 0.0100 - 18ms/epoch - 2ms/step
Epoch 11/1000
10/10 - 0s - loss: 0.0100 - 18ms/epoch - 2ms/step
Epoch 12/1000
10/10 - 0s - loss: 0.0100 - 18ms/epoch - 2ms/step
Epoch 13/1000
10/10 - 0s - loss: 0.0100 - 29ms/epoch - 3ms/step
Epoch 14/1000
10/10 - 0s - loss: 0.0100 - 17ms/epoch - 2ms/step
Epoch 15/1000
10/10 - 0s - loss: 0.0100 - 17ms/epoch - 2ms/step
Epoch 16/1000
10/10 - 0s - loss: 0.0100 - 20ms/epoch - 2ms/step
Epoch 17/1000
10/10 - 0s - loss: 0.0100 - 16ms/epoch - 2ms/step
Epoch 18/1000
10/10 - 0s - loss: 0.0100 - 16ms/epoch - 2ms/step
Epoch 19/1000
10/10 - 0s - loss: 0.0100 - 20ms/epoch - 2ms/step
Epoch 20/1000
10/10 - 0s - loss: 0.0100 - 18ms/epoch - 2ms/step
Epoch 21/1000
10/10 - 0s - loss: 0.0100 - 21ms/epoch - 2ms/step
Epoch 22/1000
10/10 - 0s - loss: 0.0100 - 17ms/epoch - 2ms/step
Epoch 23/1000
10/10 - 0s - loss: 0.0100 - 16ms/epoch - 2ms/step
Epoch 24/1000
10/10 - 0s - loss: 0.0100 - 26ms/epoch - 3ms/step
Epoch 25/1000
10/10 - 0s - loss: 0.0100 - 18ms/epoch - 2ms/step
Epoch 26/1000
10/10 - 0s - loss: 0.0100 - 16ms/epoch - 2ms/step
Epoch 27/1000
10/10 - 0s - loss: 0.0100 - 15ms/epoch - 2ms/step
Epoch 28/1000
10/10 - 0s - loss: 0.0100 - 16ms/epoch - 2ms/step
Epoch 29/1000
10/10 - 0s - loss: 0.0100 - 20ms/epoch - 2ms/step
```

```
Loss = h.history['loss']
Loss
```

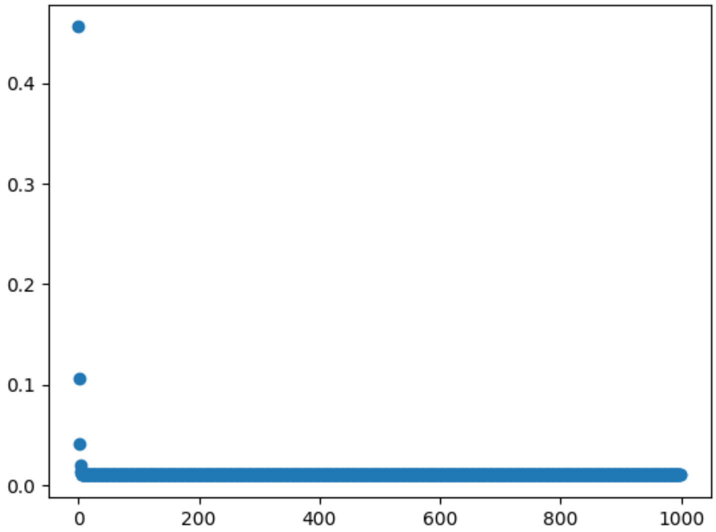
```
    0.009983240626752377,
    0.009986440651118755,
    0.009981588460505009,
    0.009990183636546135,
    0.009995918720960617,
    0.009990015998482704,
    0.009988166391849518,
    0.00999691616743803,
    0.009992764331400394,
    0.009987976402044296,
    0.009995845146477222,
    0.009979686699807644,
    0.009975286200642586,
    0.010006737895309925,
    0.009993739426136017,
    0.009979598224163055,
    0.009985408745706081,
    0.009985476732254028,
    0.01000138372182846,
    0.009978880174458027,
    0.009983888827264309,
    0.009987604804337025,
    0.009979789145290852,
    0.009980215691030025,
    0.00997757725417614,
    0.01000747550278902,
    0.009984835051000118,
    0.009978349320590496,
    0.009982410818338394,
    0.00998592283576727,
    0.009977930225431919,
    0.009994101710617542,
    0.009972793981432915,
    0.009991489350795746,
    0.009978756308555603,
    0.009980859234929085,
    0.00998321920633316,
    0.009992046281695366,
    0.009993370622396469,
    0.009988215751945972,
    0.009968490339815617,
    0.01002125721424818,
    0.0099954754114151,
    0.009979217313230038,
    0.009978643618524075,
    0.00999427493661642,
    0.009976952336728573,
    0.009981672279536724,
    0.009985855780541897,
```

```
weights = model.get_weights()

print(weights[0][0][0])
print(weights[1][0])    #bias
```

```
0.21782869
0.7834272
```

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




```
model.predict([0.6])
```

```
1/1 [==============================] - 0s 59ms/step
array([[0.91412437]], dtype=float32)
```

```
model = Sequential()
```

Podsumowanie verbose

+ Kod + Tekst

Keras verbose definiuje tryb wyświetlania postępu uczenia modelu, można dać wartośći 0, 1 lub 2 (autmatycznie jest 1). W tym trybie 0 jest definiowane jako ciche (nie ma informacji wyświtlanych o uczeniu modelu), w przypadku użycia wartości 1 potęp jest wyświetlany jako pojedyńcza linia z pasekiem postępu na epokę, a wypadku wartości 2 potęp jest wyświetlany jako pojedyncza linia na epokę. PARAMETR TEN NIE MA WPŁYWU NA UCZENIE SIĘ MODELU