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

```
import tensorflow as tf
```

Gradient

Możliwe jest wyliczenie gradientu dowolnego wyrażenia różniczkowalnego. Wykorzystujemy do tego metodę **tf.GradientTape()**

Funkcja **jednej zmiennej**:

```
x = tf.Variable(4.0)
with tf.GradientTape() as tape:
    f = x**3                    #definicja funkcji f(x)=x^3
    df_dx = tape.gradient(f, x)  #gradient 'f' ze względu na zmienną 'x'

df_dx.numpy()
```

```
 48.0
```

Funkcja **dwóch zmiennych**:

```
x = tf.Variable(4.0)
y = tf.Variable(3.0)

with tf.GradientTape() as tape:
    f = x**3+y**2                         #definicja funkcji f(x,y)=x^3+y^2
    df_dx,df_dy = tape.gradient(f,(x,y))  #gradient 'f' ze względu na zmienną 'x' i ze względu na zmienną 'y'

print(df_dx)
print(df_dy)
```

```
 tf.Tensor(48.0, shape=(), dtype=float32)
 tf.Tensor(6.0, shape=(), dtype=float32)
```

Przykład z **prezentacji**:

```
x = tf.Variable(3.0)
y = tf.Variable(2.0)

with tf.GradientTape() as tape:
    f = (x**2)*y                          #definicja funkcji f(x,y)=x^2*y
    df_dx,df_dy = tape.gradient(f,(x,y))  #gradient 'f' ze względu na zmienną 'x' i ze względu na zmienną 'y'

print(df_dx)
print(df_dy)
```

```
 tf.Tensor(12.0, shape=(), dtype=float32)
 tf.Tensor(9.0, shape=(), dtype=float32)
```

Trochę skomplikujemy:

```
x = tf.Variable([3.0,2.0])

with tf.GradientTape() as tape:
    f = (x**3)                  #definicja funkcji f(x)=x^3
    df_dx = tape.gradient(f,x)  #gradient 'f' ze względu na zmienną 'x'

print(df_dx)
```

```
 tf.Tensor([27. 12.], shape=(2,), dtype=float32)
```

I jeszcze trochę skomplikujemy:

```
x = tf.Variable([3.0,2.0])
y = tf.Variable([1.0,0.0])

with tf.GradientTape() as tape:
    f = (x**3)+y**2                       #definicja funkcji f(x)=x^3+y^2
    df_dx,df_dy = tape.gradient(f,(x,y))  #gradient 'f' ze względu na zmienną 'x'
```

```
print(df_dx)
print(df_dy)
```

```
tf.Tensor([27. 12.], shape=(2,), dtype=float32)
tf.Tensor([2. 0.], shape=(2,), dtype=float32)
```

Zmienne mogą być zastąpione przez tensory, wówczas konieczne jest **rejestrowanie wprost** operacji zastosowanych do tych sensorów. Służy do tego metoda **watch()**. W przypkadku zmiennych operacje są rejestrowane automatycznie.

```
x = tf.random.normal([2])
y = tf.random.normal([2])

print(x)
print(y)

with tf.GradientTape() as tape:
    tape.watch(x)
    tape.watch(y)
    f = (x**3)+y**2                       #definicja funkcji f(x)=x^3+y^2
    df_dx,df_dy = tape.gradient(f,(x,y))  #gradient 'f' ze względu na zmienną 'x'

print(df_dx)
print(df_dy)
```

```
tf.Tensor([-1.8064721 -1.0134428], shape=(2,), dtype=float32)
tf.Tensor([-0.59937334  0.14433193], shape=(2,), dtype=float32)
tf.Tensor([9.790023  3.0811987], shape=(2,), dtype=float32)
tf.Tensor([-1.1987467  0.2886639], shape=(2,), dtype=float32)
```

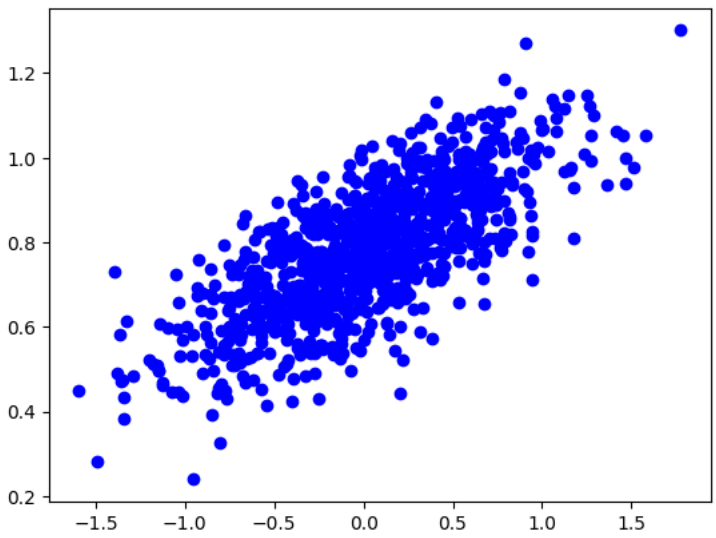
```
import matplotlib.pyplot as plt
import numpy as np


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()
```




```
x_point
```

```
        -0.22097173015298519,
        -0.6176176261806382,
        -0.9911345888529894,
        -0.718252343277229,
        0.18617926675570273,
        -0.09019651762622496,
        -0.31833529156132284,
        -0.37367635951276995,
        -0.06497781280092997,
        -0.19633839201084985,
        -0.39986225245988677,
        0.6610729598649522,
        0.6382882055355888,
        -0.5395753753847415,
        -0.35833272629935425,
        0.35573653359098834,
        -0.6523909275075414,
        -0.1998748006189423,
        -0.23509572038768772,
        -0.2563429336077604,
        -0.1760124467208113,
        -0.047280873810736056,
        -0.008442596198355655,
        -0.32497544262690936,
        0.8622765022311278,
        -0.5956410617238524,
        0.34146251419076423,
        0.40056256453108485,
        -0.3265767089222872,
        -0.4711330232219849,
        0.8204754689132666,
        0.06205149878676606,
        -0.13440790649134712,
        -0.702427805381748,
        0.5049683974856899,
        -0.7544198879919781,
        0.943743938720696,
        0.3873263523083533,
        -0.17226003562304942,
        -0.6543139840638417,
        0.015901182728849918,
        0.6103216850378801,
        0.18016723762603734,
        0.08590945476518931,
        0.5876993978155248,
        0.474641112832452,
        -0.7842583494782777,
        -0.2218673803678507,
        0.28901419215332264,
        0.4531298391961456,
        -0.4617164696107178,
        -0.4210743305090317,
        -0.28249224166838083]


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

Definicja błędu:

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


x = tf.constant([1.0, 2.0, 3.0, 4.0])
tf.reduce_mean(x).numpy()
```

```
2.5
```

```
import random
```

TODO

```
Loss = []
epochs = 1000
learning_rate = 0.001

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
    loss = loss_fn(real_y, pred_y)
```

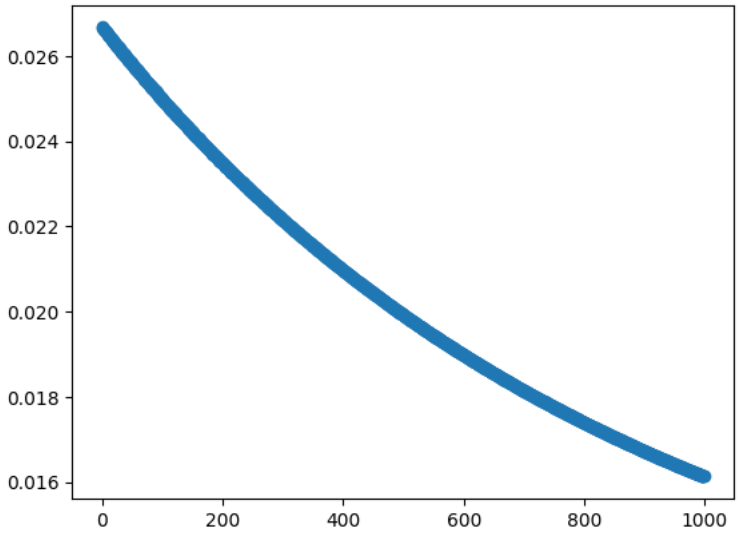
```
    Loss.append(loss.numpy())

  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
```

Wykres zmian błędu:

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




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

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