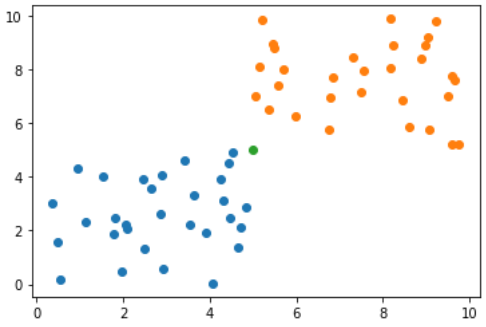
```
import pandas as pd
import numpy as np
import random
```

Generujemy zbiór danych:

```
X_A,X_B,Y_A,Y_B = [],[],[],[]
for j in range(30):
  X_A.append(random.random()*4.9)
  X_B.append(random.random()*5.1+5)
  Y_A.append(random.random()*5.1)
  Y_B.append(random.random()*4.9+5)
```

Punkty podzielone są na dwie klasy. Interesuje nas przynależność punktu (5,5) - punkt zielony.

```
import matplotlib.pyplot as plt
plt.scatter(X_A,Y_A)
plt.scatter(X_B,Y_B)
plt.scatter(5,5)
plt.show()
```



Łączymy współrzędne w krotki i zapisujemy je w tablicy. Etykiety umieszczamy w tablicy.

```
X_A = np.array(X_A)
X_B = np.array(X_B)
Y_A = np.array(Y_A)
Y_B = np.array(Y_B)
X = np.concatenate([X_A,X_B])
Y = np.concatenate([Y_A,Y_B])

data = list(zip(X,Y))

L_0 = np.full((30,),0)
L_1 = np.full((30,),1)
L = np.concatenate([L_0,L_1])
dataC =list(zip(X,Y,L))
data,L
```

```
       (2.4471498932396347, 3.8975996628323597),
       (4.311599756934288, 3.129323409271743),
       (4.257125258226131, 3.89952705973703),
       (4.43599112892591, 4.518740859242036),
       (1.8220902915238648, 2.4747476770202286),
       (0.48011749888611044, 1.5950836360447287),
       (1.9575214310636806, 0.4862335908652296),
       (4.717934841757632, 2.102052609772762),
       (1.7746107124397301, 1.8499194331116253),
       (1.1333498173554815, 2.3347145954504644),
       (3.403507232440234, 4.611648567034556),
       (4.669485479855284, 1.3877811377888385),
       (0.9614129751620495, 4.338546679344032),
       (0.36139220899579344, 3.027937126090144),
       (2.05921618048528, 2.2412649255861643),
       (3.9124331875778844, 1.941654017226313),
       (2.496031140028135, 1.3250880980373778),
       (2.892930551329341, 4.059955755870323),
       (4.521359432552599, 4.908145198662925),
       (2.9219796796113635, 0.5956498114846187),
       (4.848492790671941, 2.885183719019854),
```

```
 (5.452150618003403, 8.943410170398534),
 (9.607656624538825, 7.753990390897641),
 (5.99759475761825, 6.250855132239232),
 (5.139323594870438, 8.077235817208553),
 (5.199363869238435, 9.863317140124739),
 (8.465180458455343, 6.833179868122967),
 (5.481686330320197, 8.793736089327489),
 (8.902352452645825, 8.382733574845977),
 (9.219034390084188, 9.812173831440472),
 (5.3607783674732365, 6.530126997934978),
 (6.747837327221626, 5.7756769439679125),
 (5.061256881126596, 7.026458107728223),
 (9.671263357862546, 7.577443244252294),
 (6.782292797611046, 6.932406089740718),
 (7.301706710932772, 8.45972317314279),
 (8.609795011982657, 5.864722340782496),
 (5.592483534274709, 7.422562451843758),
 (7.5474865900603, 7.9352000873166055),
 (8.170762643478968, 9.879446186991792),
 (9.041561156674113, 9.195256927975326),
 (6.846567115659311, 7.697455517030395),
 (9.086651923797076, 5.774686614519051),
 (8.178049514835504, 8.056109820687318),
 (9.592041372042846, 5.209930647676828),
 (9.76491230826419, 5.221052863949283),
 (8.981644011495556, 8.919295584408747),
 (5.69102951709372, 8.017852825441715),
 (8.239667146380736, 8.907145291870894),
 (7.511043441041924, 7.177174002221364)],
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]))
```

Klasyfikacja dla k=3:

```
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(data, L)

point = np.array([[2,2]])

print(neigh.predict(point))
```

```
[0]
```

K nieparzyste

```
import math


#NewData = list(zip(data,L))#np.dstack((dataC,L))

print(dataC)
```

```
[(0.5538771260457623, 0.17692494683198737, 0), (2.104135867180641, 2.0829092599517893, 0), (1.5444941934255163, 4.003294021660498,
```

```
point_x = 2
point_y = 2
num_neighbors=3
distances = list()
neighbors = list()
for point in dataC:
  print(point[0],point[1])
  i = -1
  j= -1
  #
  #for cent in Centers:
  #  print("Cent",cent[0],cent[1])
  #  i= i+1
  euclidesian=math.sqrt(((point_y-point[1])*(point_y-point[1]))+((point_x-point[0])*(point_x-point[0])))
  #if(i==0):
     # min=euclidesian
     # j=i
  #if(min>euclidesian):
     # min=euclidesian
    #  j=i

  print(euclidesian)
  print("    ")
  #print("Max"," i ", min,i,j)
```

```
    #point[2]=j

    #distances = list()
   # for train_row in train:
      #dist = euclidean_distance(test_row, train_row)
    distances.append((point, euclidesian))
    distances.sort(key=lambda tup: tup[1])

     #neighbors = list()
  for test in distances:
    print(test,"test")
  for i in range(num_neighbors):
     neighbors.append(distances[i][0])

  print(distances)
  print("******")
  print(neighbors)
```

```
      2.5727154523984184

      4.257125258226131 3.89952705973703
      2.9500538100169607

      4.43599112892591 4.518740859242036
      3.504013170098115

      1.8220902915238648 2.4747476770202286
      0.5069883836994361

      0.48011749888611044 1.5950836360447287
      1.5728954443925922

      1.9575214310636806 0.4862335908652296
      1.5143622982112488

      4.717934841757632 2.102052609772762
      2.719850094987133

      1.7746107124397301 1.8499194331116253
      0.27078498389767097

      1.1333498173554815 2.3347145954504644
      0.9290405801068894

      3.403507232440234 4.611648567034556
      2.964884616507984

      4.669485479855284 1.3877811377888385
      2.7387889043161624

      0.9614129751620495 4.338546679344032
      2.5588011997091025

      0.36139220899579344 3.027937126090144
      1.9343449092481284

      2.05921618048528 2.2412649255861643
      0.24842568375544966

      3.9124331875778844 1.941654017226313
      1.913323012628849

      2.496031140028135 1.3250880980373778
      0.8375875878309179

      2.892930551329341 4.059955755870323
      2.245159835210089

      4.521359432552599 4.908145198662925
      3.8489689378622156

      2.9219796796113635 0.5956498114846187
      1.6799541605649422

      4.848492790671941 2.885183719019854
      2.9828612765141864
```

```
  classesArr = np.array([])
  for classes in range(num_neighbors):#neighbors:
    print(neighbors[classes][2])
    classesArr·=·np.append(classesArr,neighbors[classes][2])
```

```
      0
      0
      0
```

```
classesArr = np.append(classesArr,2)


classesArr = classesArr.astype(int)
counts = np.bincount(classesArr)
print(np.argmax(counts))
```

```
0
```


✓ 0 s ukończono o 14:37