Link to read only copy of this notebook: https://colab.research.google.com/drive/1XJXMnm_ilFxj4ggKjmEyBcEPbuAIvk8K?usp=sharing

Q1: Calcluate the distance between 2 points of an n-dimensional vectors.

> **AUTOMATED | EASY**

Eg:

```
# Input
v1 = [4, 8, 9, 11]
v2 = [0, 1, -1, 20]

# Output
15.68
```

```
v1 = [4, 8, 9, 11]
v2 = [0, 1, -1, 20]


comp = 0
for i in range(len(v1)):
  comp += (v1[i] - v2[i])**2

print(comp**0.5)
```

```
15.684387141358123
```

```
# Numpy Solution
import numpy as np
np.sum((np.array(v1) - np.array(v2))**2)**0.5
```

```
15.684387141358123
```

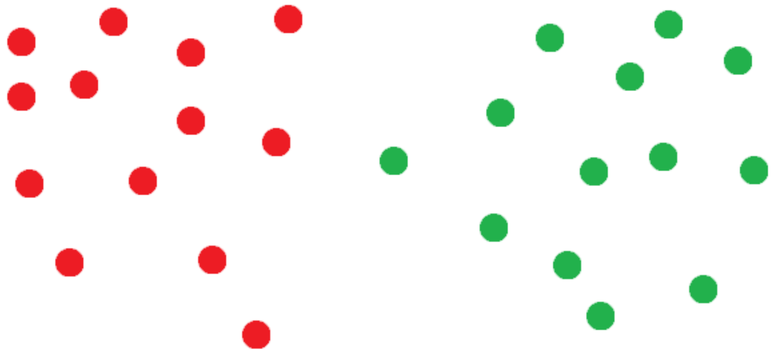
```
# One More way (but this might not be allowed)
np.linalg.norm(np.array(v1)-np.array(v2))
```

```
15.684387141358123
```

Q-2 NumPy for Machine Learning

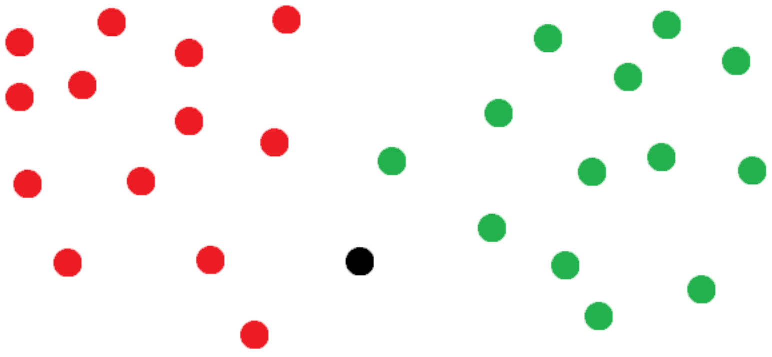
> **LIVE | MEDIUM**

Assume we have some points in our 2D plane, some of which are of red color and the others are green in color.

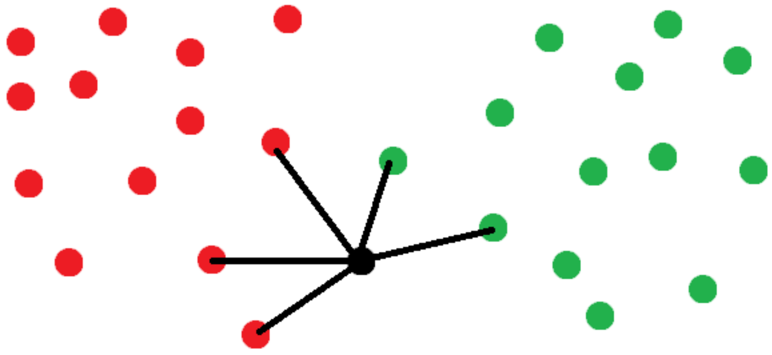


Now we have a new point that does not have any color (black point in the below image)

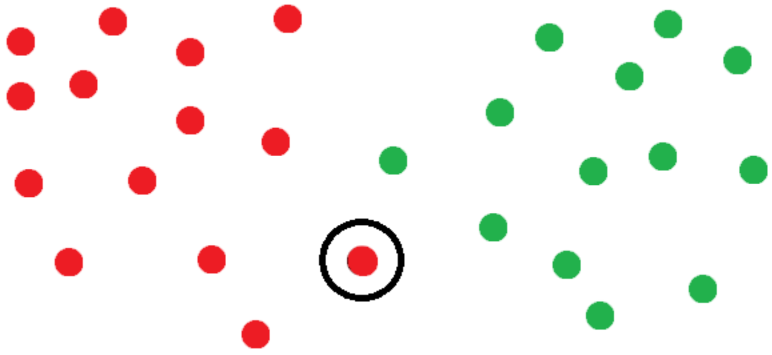
We would like to assign it a color (either red or green) based on the color of it's "K" nearest points.



Let K = 5, that is, we see the nearest 5 points and assign the current point a color which occurs in majority in these 5 points.



As highlighted above, 3 of the 5 nearest points have color as red and hence we assign the current point a red color.



In this process we were required to calculate the distance of current point from all others and pick the nearest K points. This algorithm is known as **K-Nearest-Neighbors** that is a use case for Calculating distance of multiple points from a given point efficiently.

```
# Some points in 4d
all_points = [[1, 2, 3, 4], [2, 3, 0, -4], [-1, 4, 1, 0], [0, 1, 2, 5], [1, 10, 0, 0], [0,0,0,0]]
current_point = [4, 1, 0, 3]
```

Iterating over points and using the above example's optimal code for distance between two points.

```
import numpy as np
distances = []
for point in all_points:
  distances.append(np.sum((np.array(point) - np.array(current_point))**2)**0.5)

print(distances)
```

```
    [4.47213595499958, 7.54983443527075, 6.6332495807108, 4.898979485566356, 9.9498743710662]
```

```
# Minimum distance points
np.argsort(distances)
```

```
    array([0, 3, 2, 1, 4])
```

But do we really need a loop?

Numpy solution : instead of looping we can directly calculate the distance using broadcasting and using the sum function which calculates for all points

```
all_distances = np.sum((np.array(all_points) - np.array(current_point))**2, axis = 1)**0.5
print(all_distances)
```

```
    [4.47213595 7.54983444 6.63324958 4.89897949 9.94987437]
```

```
# Minimum distance points
np.argsort(all_distances)
```

```
    array([0, 3, 2, 1, 4])
```

We can the result is the same for both the solutions but numpy solution would be more optimal.

Q-3: Numpy for Deep Learning (2-Dimensional Average Pooling)

> **LIVE | MEDIUM**

2D Average Pooling: Given a 2 dimensional array of size m x m, and a 2 dimensional window of size d x d, take the average of the elements of the 2D array that fall inside the window while moving the window by (slide=) d units from left-right and top-bottom.

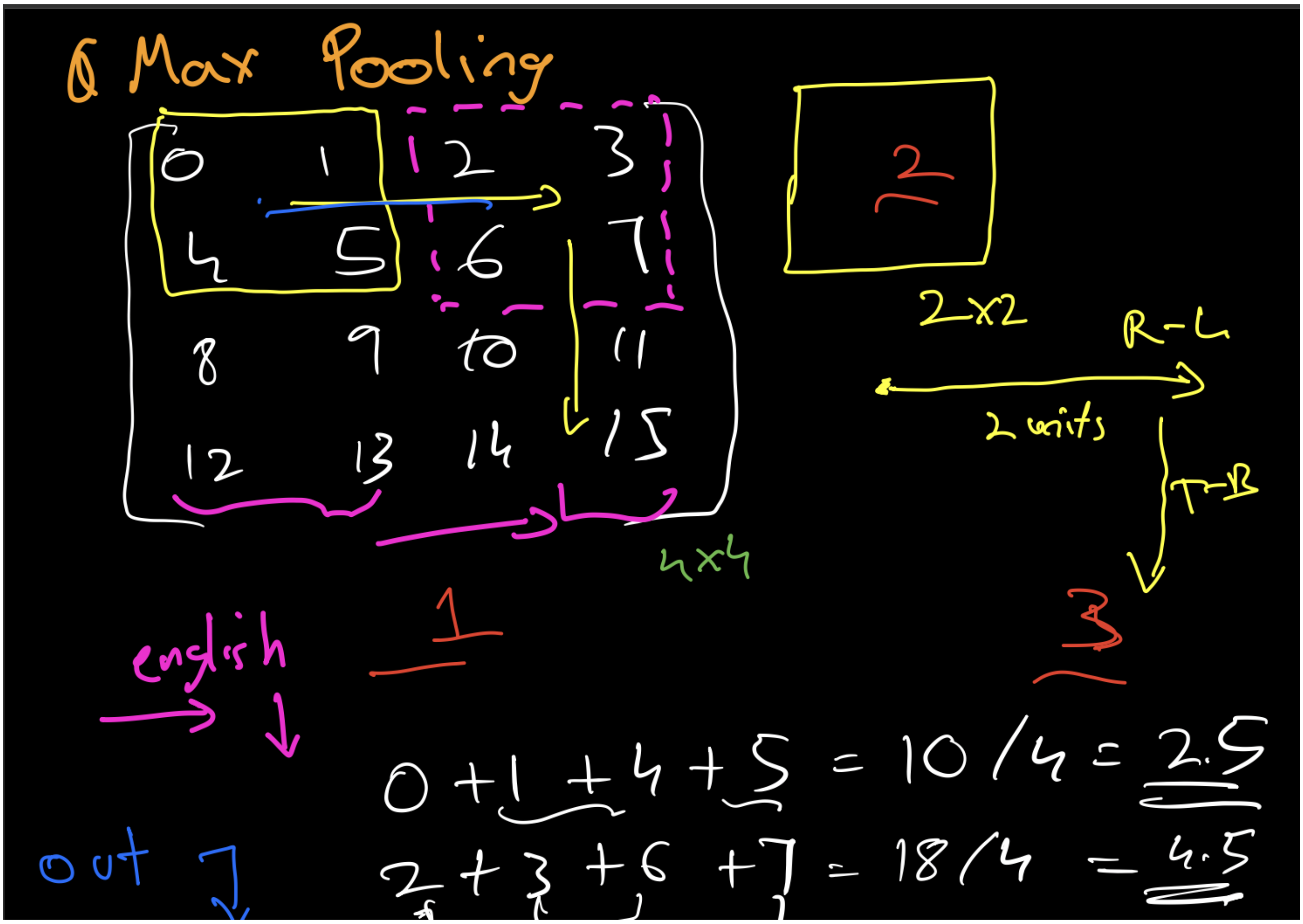
Eg:

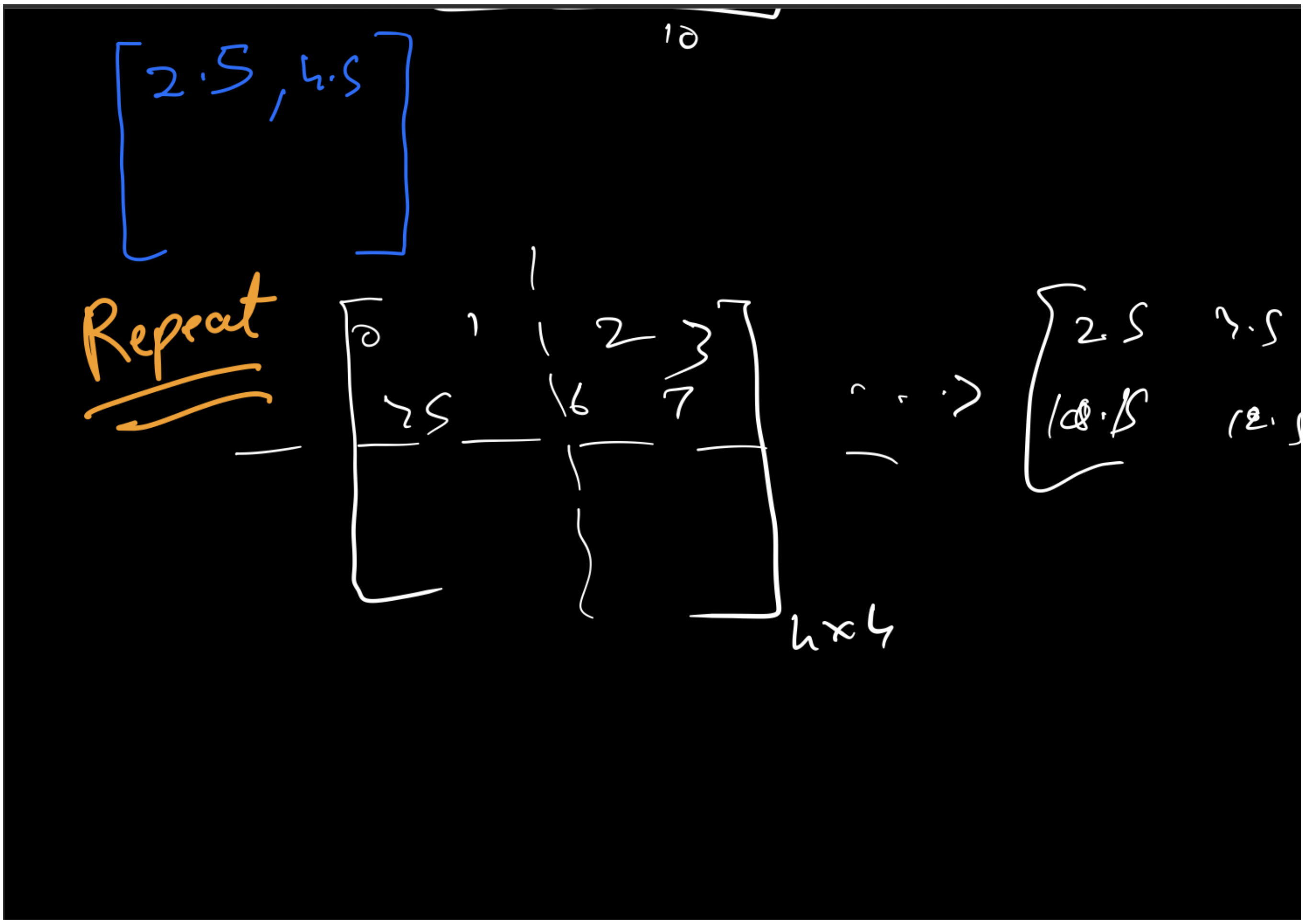
> INPUT

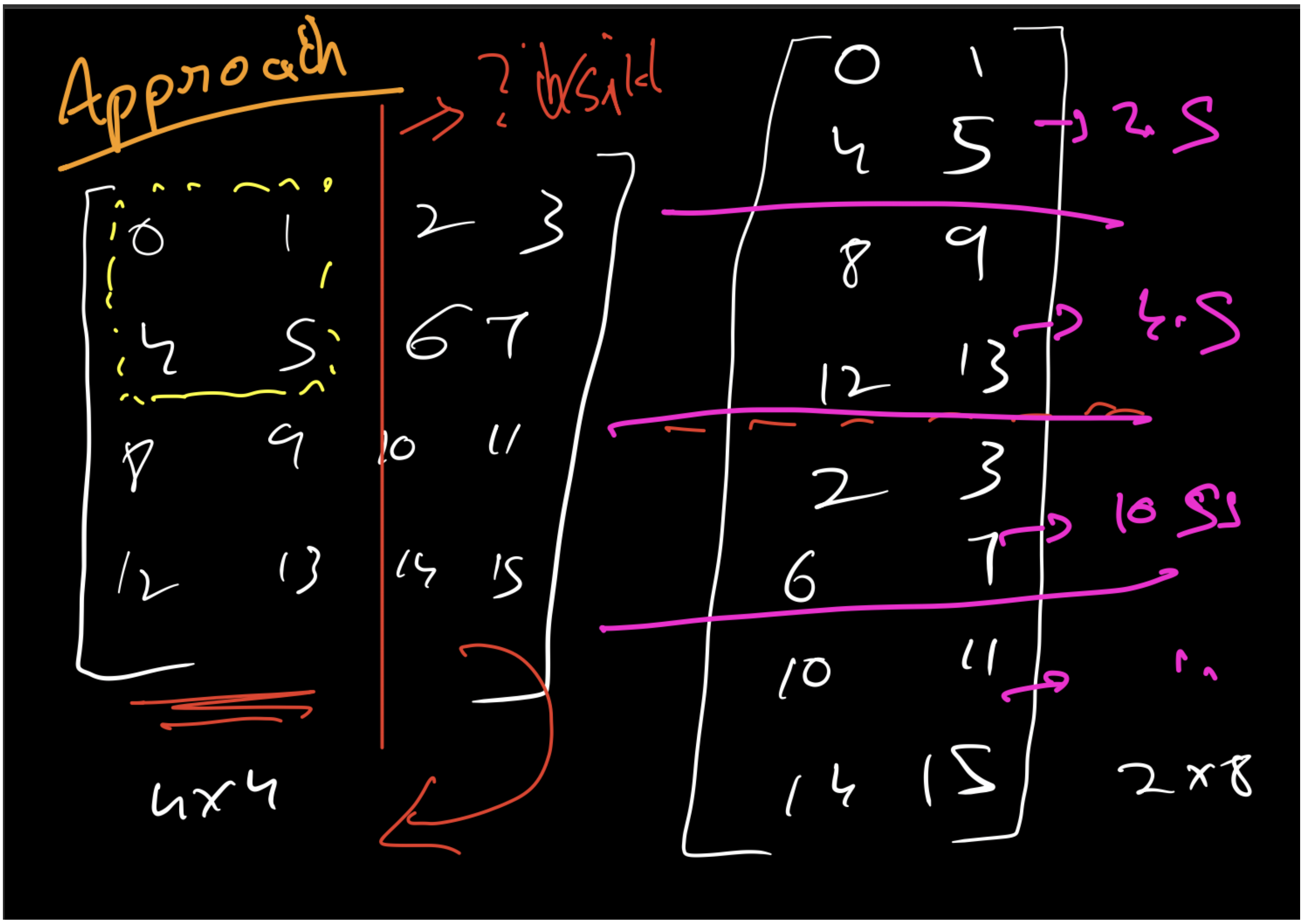
```
# m=4, i.e array size = 4x4
# d=2, i.e window size = 2x2
# s=d, i.e stride=2



[[ 0,  1,  2,  3]
[ 4,  5,  6,  7]
[ 8,  9, 10, 11]
[12, 13, 14, 15]]
```

> OUTPUT

```
[2.5, 4.5]
[10.5, 12.5]
```


Q Max Pooling
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
2
2x2
R-L
2 units
T-B
4x4
english
1
3
0+1+4+5 = 10/4 = 2.5
2+3+6+7 = 18/4 = 4.5
out


10
[2.5, 4.5
Repeat
0 1 2 3
45 6 7
4x4
2.5 3.5
10.5 12.



Setup and Input

```
import numpy as np
```

```
x = np.arange(16).reshape(4,4)
```

```
x
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

Suboptimal Solution (Hardoded)

```
result = np.zeros((2,2))
result
```

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

```
np.hsplit(x, 2)[0]
```

```
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]])
```

```
x1 = np.hsplit(x, 2)[0]
```

```
x2 = np.vsplit(x1, 2)[0]
x2
```

```
array([[0, 1],
       [4, 5]])
```

FINAL RESULT

```
result = np.zeros((2,2))
for i in range(2):
  for j in range(2):
    result[i,j] = np.vsplit(np.hsplit(x, 2)[j], 2)[i].mean()
print(result)
```

```
[[ 2.5  4.5]
 [10.5 12.5]]
```

Best Solution

```
x1 = np.concatenate(np.split(x, 2, axis=1))
x1
```

```
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13],
       [ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15]])
```

```
for a in np.split(x1, 4, axis=0):
  print(a)
  print("============")
```

```
[[0 1]
 [4 5]]
============
[[ 8  9]
 [12 13]]
============
[[2 3]
 [6 7]]
============
[[10 11]
 [14 15]]
============
```

```
x2 = np.array(np.split(x1, 4, axis=0))
x2
```

```
array([[[ 0,  1],
        [ 4,  5]],

       [[ 8,  9],
        [12, 13]],

       [[ 2,  3],
        [ 6,  7]],

       [[10, 11],
        [14, 15]]])
```

```
x2.sum(axis=1).sum(axis=1)/4 # works for but not a clean solution
```

```
array([ 2.5, 10.5,  4.5, 12.5])
```

```
x2.mean(axis=(1,2))
```

```
array([ 2.5, 10.5,  4.5, 12.5])
```

```
x2.reshape(2,2,4)
```

```
array([[[ 0,  1,  4,  5],
        [ 8,  9, 12, 13]],

       [[ 2,  3,  6,  7],
        [10, 11, 14, 15]]])
```

```
x2.reshape(2,2,4).mean(axis=2)
```

```
array([[ 2.5, 10.5],
       [ 4.5, 12.5]])
```

```
x1.reshape(2,2,4)
```

```
array([[[ 0,  1,  4,  5],
        [ 8,  9, 12, 13]],

       [[ 2,  3,  6,  7],
        [10, 11, 14, 15]]])
```

FINAL RESULT

```
def pool(x):
  return np.array(np.split(np.concatenate(np.split(x, 2, axis=1)), 4, axis=0)).reshape(2,2,4).mean(axis=2).T # Goes to product
```

```
pool(x)
```

```
array([[ 2.5,  4.5],
       [10.5, 12.5]])
```

```
def pool(x, window=2):
  n = x.shape[0]
  w = window
  return np.array(np.split(np.concatenate(np.split(x, int(n/w), axis=1)), int(n/w)**2, axis=0)).reshape(w, w, int(n/w)**2).mea
```

```
pool(x, 2)
```

```
array([[ 2.5,  4.5],
       [10.5, 12.5]])
```

```
x = np.arange(256).reshape(16,16)
pool(x, 4)
```

```
array([[ 25.5,  29.5,  33.5,  37.5],
       [ 89.5,  93.5,  97.5, 101.5],
       [153.5, 157.5, 161.5, 165.5],
       [217.5, 221.5, 225.5, 229.5]])
```