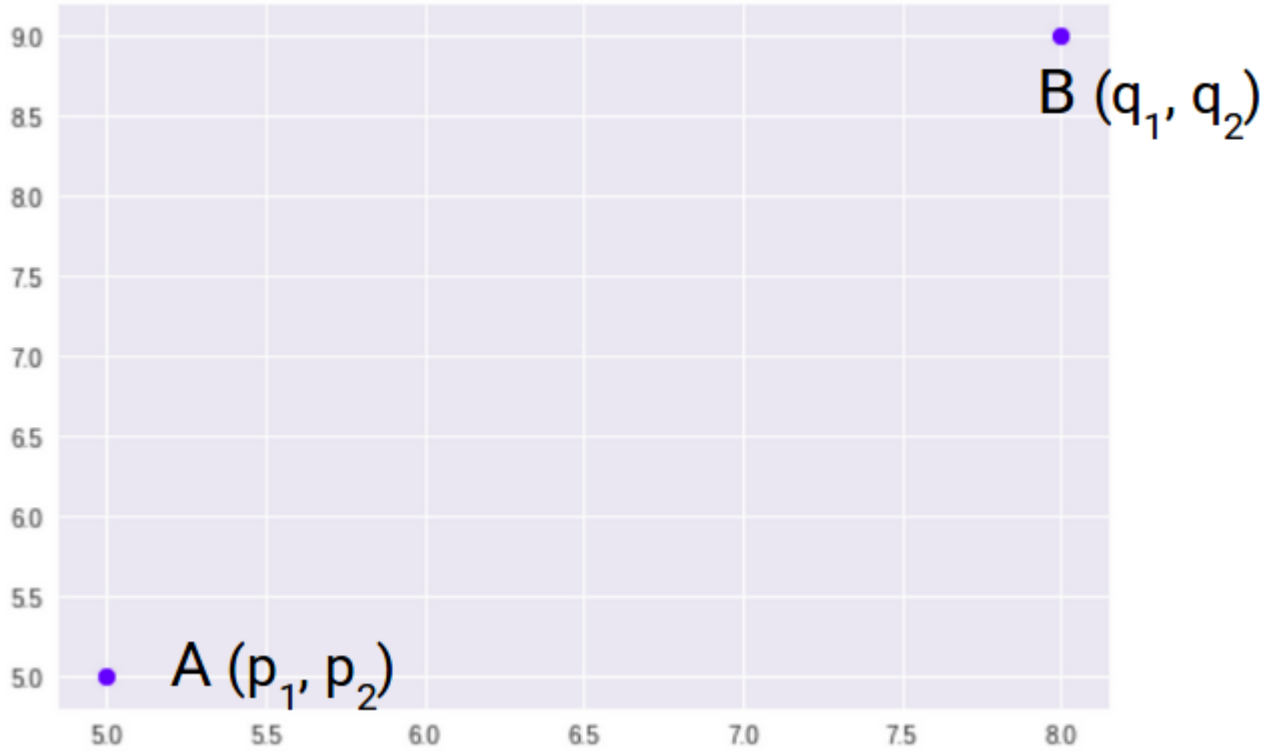
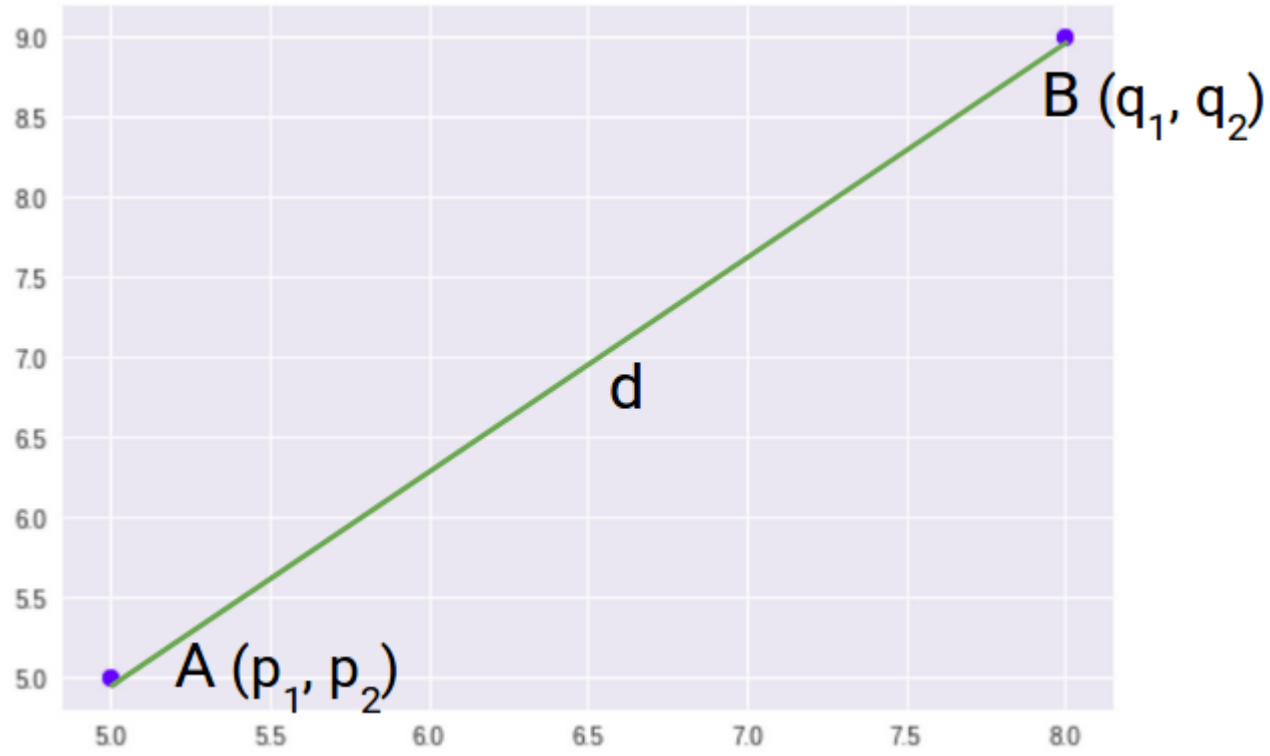
# Euclidean Distance represents the shortest distance between two points.

Most machine learning algorithms including K-Means use this distance metric to measure the similarity between observations.

Let's say we have two points as shown below




# So, the Euclidean Distance between these two points A and B will be:




Here's the formula for Euclidean Distance:

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

Double-click (or enter) to edit

We use this formula when we are dealing with 2 dimensions.

We can generalize this for an n-dimensional space as:

here,

n = number of dimensions

pi, qi = data points

Let’s code Euclidean Distance in

Double-click (or enter) to edit

```
# importing the library
from scipy.spatial import distance

# defining the points
point_1 = (1, 2, 3)
point_2 = (4, 5, 6)
point_1, point_2
```

```
((1, 2, 3), (4, 5, 6))
```

## Examples Using Euclidean Distance Formula

**Example 1:** Find the distance between points P(3, 2) and Q(4, 1).

**Solution:**

Given:

$P(3, 2) = (x_1, y_1)$

$Q(4, 1) = (x_2, y_2)$

Using Euclidean distance formula,

$d = \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$

$PQ = \sqrt{[(4 - 3)^2 + (1 - 2)^2]}$

$PQ = \sqrt{[(1)^2 + (-1)^2]}$

$PQ = \sqrt{2}$ units.

Double-click (or enter) to edit

```
# importing the library
from scipy.spatial import distance

# defining the points
point_1 = (1,2,3)
point_2 = (4,5,6)
point_1, point_2
# computing the euclidean distance
euclidean_distance = distance.euclidean(point_1, point_2)
print('Euclidean Distance b/w', point_1, 'and', point_2, 'is: ', euclidean_distance)
```

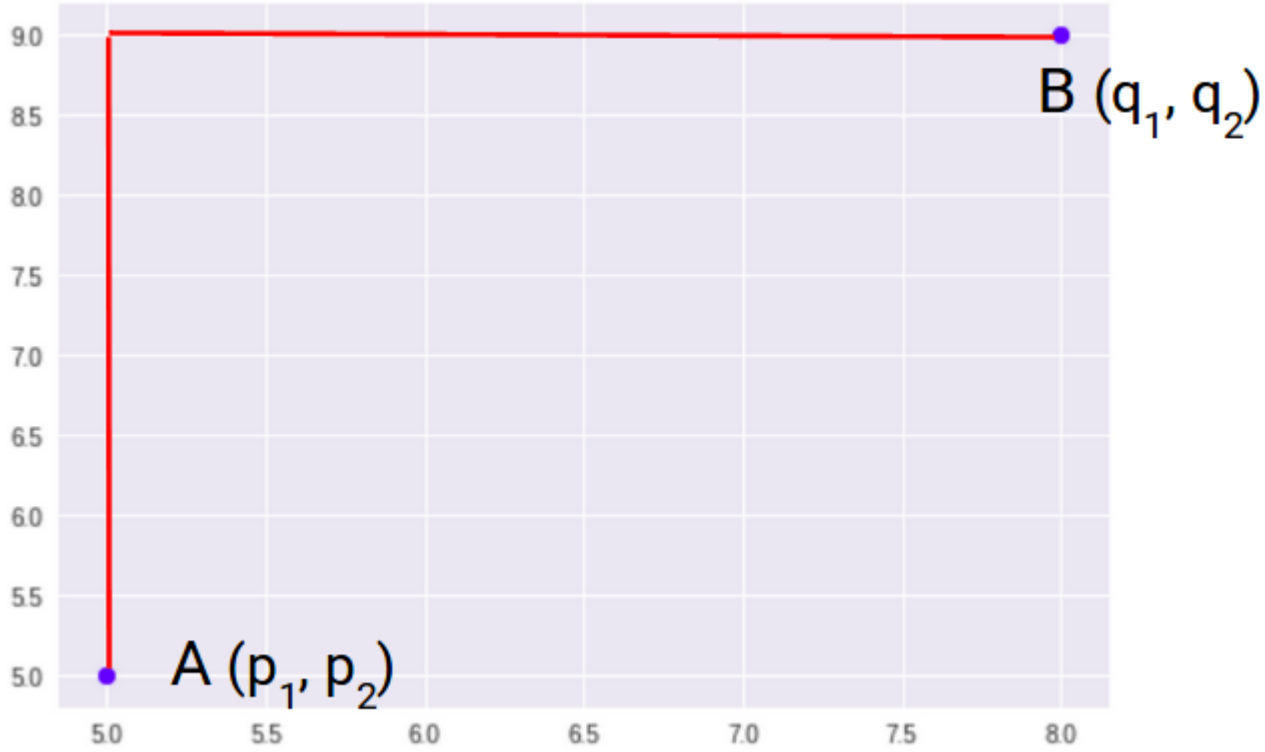
```
Euclidean Distance b/w (1, 2, 3) and (4, 5, 6) is:  5.196152422706632
```

This is how we can calculate the Euclidean Distance between two points in Python. Let’s now understand the second distance metric, Manhattan Distance.

## 2. Manhattan Distance

Manhattan Distance is the sum of absolute differences between points across all the dimensions.

We can represent Manhattan Distance as:




Since the above representation is 2 dimensional, to calculate Manhattan Distance,

we will take the sum of absolute distances in both the x and y directions.

So, the Manhattan distance in a 2-dimensional space is given as:

$$d = |p_1 - q_1| + |p_2 - q_2|$$

## And the generalized formula for an n-dimensional space is given as:

$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

Where,

n = number of dimensions pi, qi = data points Now, we will calculate the Manhattan Distance between the two points:

```
# importing the library
from scipy.spatial import distance

# defining the points
point_1 = (1,2,3)
point_2 = (4,5,6)
point_1, point_2
```

```
((1, 2, 3), (4, 5, 6))
```

```
manhattan_distance = distance.cityblock(point_1, point_2)
print('Manhattan Distance b/w', point_1, 'and', point_2, 'is: ', manhattan_distance)
```

```
Manhattan Distance b/w (1, 2, 3) and (4, 5, 6) is:  9
```

## Note that Manhattan Distance is also known as city block distance.

SciPy has a function called cityblock that returns the Manhattan Distance between two points.

## 3. Minkowski Distance

Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.

The formula for Minkowski Distance is given as:

$$D = \left( \sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$

Here, p represents the order of the norm.

Let's calculate the Minkowski Distance of the order 3:

```
# importing the library
from scipy.spatial import distance

# defining the points
point_1 = (1,2,3)
point_2 = (4,5,6)



# computing the minkowski distance
minkowski_distance = distance.minkowski(point_1, point_2, p=3)
print('Minkowski Distance b/w', point_1, 'and', point_2, 'is: ', minkowski_distance)
```

```
Minkowski Distance b/w (1, 2, 3) and (4, 5, 6) is:  4.3267487109222245
```

The p parameter of the Minkowski Distance metric of SciPy represents the order of the norm.

When the order(p) is 1, it will represent Manhattan Distance and when the order in the above formula is 2, it will represent Euclidean Distance.

Let's verify that in Python:

```
# minkowski and manhattan distance
minkowski_distance_order_1 = distance.minkowski(point_1, point_2, p=1)
print('Minkowski Distance of order 1:',minkowski_distance_order_1, '\nManhattan Distance: ',m
```

```
Minkowski Distance of order 1: 9.0
Manhattan Distance:  9
```

```
# minkowski and manhattan distance
minkowski_distance_order_1 = distance.minkowski(point_1, point_2, p=2)
print('Minkowski Distance of order 2:',minkowski_distance_order_1, '\nManhattan Distance: ',m
```

```
Minkowski Distance of order 2: 5.196152422706632
Manhattan Distance:  9
```

So far, we have covered the distance metrics that are used when we are dealing with continuous or numerical variables.

But what if we have categorical variables? How can we decide the similarity between categorical variables? This is where we can make use of another distance metric called Hamming Distance.

Hamming Distance measures the similarity between two strings of the same length. The Hamming Distance between two strings of the same length is the number of positions at which the corresponding characters are different.

Let's understand the concept using an example. Let's say we have two strings:

“euclidean” and “manhattan”

Since the length of these strings is equal, we can calculate the Hamming Distance. We will go character by character and match the strings.

The first character of both the strings (e and m respectively) is different.

Similarly, the second character of both the strings (u and a) is different. and so on.

Look carefully – seven characters are different whereas two characters (the last two characters) are similar:

euclidean and manhattan

distance metrics Hence, the Hamming Distance here will be 7. Note that larger the Hamming Distance between two strings, more dissimilar will be those strings (and vice versa).

Double-click (or enter) to edit

```
# defining two strings
string_1 = 'euclidean'
string_2 = 'manhattan'
```

These are the two strings “euclidean” and “manhattan” which we have seen in the example as well. Let’s now calculate the Hamming distance between these two strings:

```
# computing the hamming distance
hamming_distance = distance.hamming(list(string_1), list(string_2))*len(string_1)
print('Hamming Distance b/w', string_1, 'and', string_2, 'is: ', hamming_distance)
```

```
Hamming Distance b/w euclidean and manhattan is:  7.0
```

As we saw in the example above, the Hamming Distance between “euclidean” and “manhattan” is 7.

We also saw that Hamming Distance only works when we have strings of the same length.

+ Code + Text

Let’s see what happens when we have strings of different lengths:

```
# strings of different shapes
new_string_1 = 'data'
new_string_2 = 'science'
len(new_string_1), len(new_string_2)
```

```
(4, 7)
```

We can see that the lengths of both the strings are different. Let's see what will happen when we try to calculate the Hamming Distance between these two strings:

```
# computing the hamming distance
hamming_distance = distance.hamming(list(new_string_1), list(new_string_2))
```

This throws an error saying that the lengths of the arrays must be the same. Hence, Hamming distance only works when we have strings or arrays of the same length.

These are some of the similarity measures or the distance matrices that are generally used in Machine Learning.