NUMERIC

PYTHON

(NUMPY)

Overview

- add-on module for for scientific computing
- built around ndarray()
- (multi) dimensional arrays of objects
- vectorized operations

Arrays

- similar to lists
- but: all elements of the same type
- support indexing/slicing
- multi-dimensional (through re-shaping)
- efficient, vectorized operations

Lists vs. Arrays

```
> from random import random
> import numpy as np
> n = 1000
> x_list = [ random() for i in range(n)]
> y_list = [ random() for i in range(n)]
> timeit z_list = [ x_list[i] + y_list[i]
                              for i in range(n)]
91.8 ms  2.46 ms per loop (mean  std. dev.
of 7 runs, 10000 loops each)
> x_vector = np.array(x)
> y_vector = np.array(y)
> timeit z_vector = x_vector + y_vector
922 ns  12.7 ns per loop (mean  std. dev.
of 7 runs, 1000000 loops each)
```

- vectorized operations are faster
- avoid iterations

Lists vs. Arrays (cont'd)

```
> from random import random
> import numpy as np
> n = 1000
> x_list = [ random() for i in range(n)]
> x_list.__sizeof__()
9000
> x_vector = np.array(x)
> x_vector.__sizeof__()
8096
```

- numpy arrays are more memory efficient

Basic Constructors

```
> import numpy as np
> np.arange(0, 5, 1)
array([ 0,  1,  2,  3,  4])
> np.zeros(5, dtype=float)
array([ 0.,  0.,  0.,  0.,  0.])
> np.linspace(0, 2, 6)
array([ 0. ,  0.4,  0.8,  1.2,  1.6,  2. ])
> np.eye(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
> np.diag([5, 10])
array([[ 5,  0],
       [ 0, 10]])
```

Universal Functions (ufunc)

- ufunc perform element-wise operations
- allow to apply scalar arithmetic to vectors

```
> import numpy as np
> x = np.linspace(0, 1, 5)
array([ 0.  , 0.25, 0.5, 0.75, 1. ])
> y = 2*x
array([ 0.  , 0.5, 1., 1.5, 2.])
> z = x/7
array([ 0.  , 0.0625, 0.125 , 0.1875, 0.25 ])
> w = np.sin(x)
array([ 0., 0.24740396, 0.47942554,
                    0.68163876, 0.84147098])
```

Multidim. Arrays

```
> import numpy as np
> v = np.array([[1,2,3], [4,5,6], [7,8,9]])
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
> v.shape
(3, 3)
> v.ndim
2
> v.dtype
dtype('int32')
```

Broadcasting

- can apply functions to arrays of different sizes

```
> x = np.array([1,2,3])
> y = np.array([4,5,6])
> z = x + y
> z
array([5, 7, 9])
> w = 2 + y
> w
array([6, 7, 8])
```

Numeric Python

- focus on n-dimensional arrays (vectors and matrices)
- all objects of the same type
- vectorized (more efficient than Python objects)
- many math and statistical functions

Creating a Numpy Vector

```
> import numpy as np
> x = np.array([1,2,3,4,5])
> type(x)
numpy.ndarray
> x.ndim
1
> x.shape
(5,)
> x.size
5
```

Creating a Numpy Vector (cont'd)

- implicit typing

```
> import numpy as np
> x = np.array([1,2,3,4,5])
> x.dtype
int32
```

- explicit typing

```
> x = np.array([1,2,3,4,5], dtype=float)
> x.dtype
float64
```

Creating Sequences

- evenly spaced with step 1

```
> x = np.arange(0, 10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- evenly spaced with step 2

```
> x = np.arange(0, 10, step=2)
array([0, 2, 4, 6, 8])
```

- evenly spaced, specified number of elements

```
> x = np.linspace(start=0,stop=10,
                              num=5)
array([0. , 2.5, 5. , 7.5, 10.])
```

Creating Sequences (cont'd)

- repeat value 1 six times

```
> x = np.ones(shape=10)
> array([ 1.,1.,1.,1.,1.,1.])
```

- repeat value 3 ten times

```
> x = np.full(shape=10, fill_value=3)
array([3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

Creating from a List

```
> x = [1,2,3,4,5]
> type(x)
<class 'list'>
> y = np.asarray(x, dtype=float)
> type(y)
<class 'numpy.ndarray'>
```

- can append (like in a list)

```
> x = np.array([1,2,3,4,5]
> x = np.append(x, 6)
> x
array([1,2,3,4,5,6]
```

Arrays Manipulations

- can delete at some position like in a list

```
> x = np.array([1,2,3,4, 5])
> y = np.delete(x,2)
> y
array([1,2,4,5])
```

- add extra zeros

```
> x = np.array([1,2,3,4,5])
> x.resize(new-shape=7)
> x
array[1,2,3,4,5,6,7]
```

Arrays Manipulations (cont'd)

- can concatenate (like lists)

```
> x = np.array([1,2,3,4,5])
> y = np.array([6,7,8,9,10])
> z = np.append(x,y)
> z
array([1,2,3,4,5,6,7,8,9,10])
```

Sorting and Searching

- get maximum value

```
> x = np.array([4,3,5,2,1])
> np.max(x)
5
```

- index of the maximum value

```
> np.argmax(x)
2
```

- sort

```
> np.sort(x)
> array([1, 2, 3, 4, 5])
```

Sorting and Searching (cont'd)

- indices for sorting

```
> np.argsort(x)
array([4, 3, 1, 0, 2], dtype=int64)
```

- get unique elements

```
> y = np.array([1,2,2,3,4,4])
> z = np.unique(y)
> z
array([1, 2, 3, 4])
```

Data Processing With Arrays

- evaluate $f(x) = \sqrt{x^2 + y^2}$
- interested in some range of x and y
- can create a mesh

```
> x_points = np.arange(-2, 3, 1)
array([-2, -1,  0,  1,  2])
> y_points = np.arange(-1, 2, 1)
array([-1,  0,  1])
> xs, ys = np.meshgrid(x_points, y_points)
```

Data Processing With Arrays (cont'd)

```
> xs
array([[-2, -1,  0,  1,  2],
       [-2, -1,  0,  1,  2],
       [-2, -1,  0,  1,  2]])
> ys
array([[-1, -1, -1, -1, -1],
       [ 0,  0,  0,  0,  0],
       [ 1,  1,  1,  1,  1]])
> z = np.round(np.sqrt(xs**2 + ys**2), 2)
array([[ 2.24,  1.41,  1.  ,  1.41,  2.24],
       [ 2.  ,  1.  ,  0.  ,  1.  ,  2.  ],
       [ 2.24,  1.41,  1.  ,  1.41,  2.24]])
```

Multi-Dimensional Arrays

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

```
> import numpy as np
> X = np.arange(1,37).reshape(6,6)
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24],
       [25, 26, 27, 28, 29, 30],
       [31, 32, 33, 34, 35, 36]])
```

Transposing Arrays

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

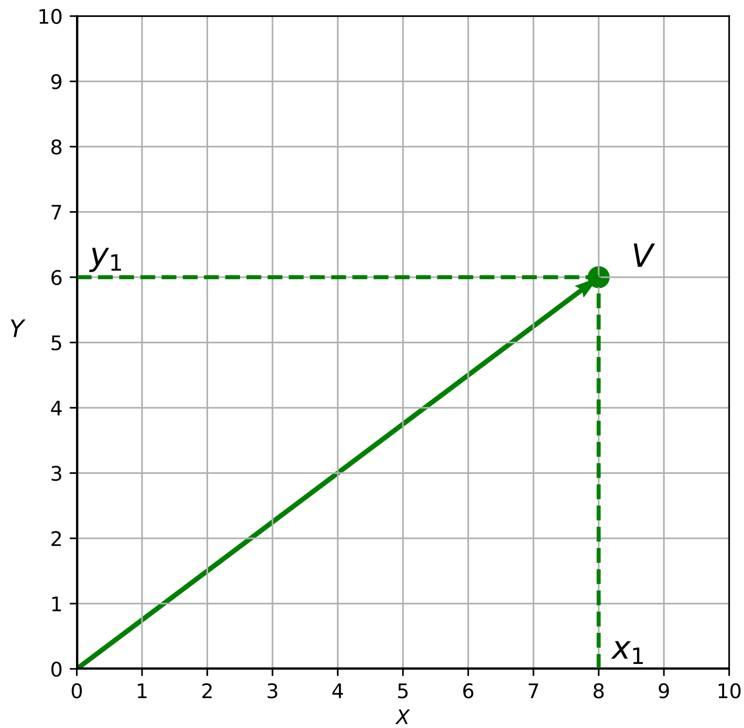
```
> import numpy as np
> X = np.arange(1,37).reshape(6,6)
> Y = X.transpose()
array([[ 1,  7, 13, 19, 25, 31],
       [ 2,  8, 14, 20, 26, 32],
       [ 3,  9, 15, 21, 27, 33],
       [ 4, 10, 16, 22, 28, 34],
       [ 5, 11, 17, 23, 29, 35],
       [ 6, 12, 18, 24, 30, 36]])
```

Reshaping Arrays

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

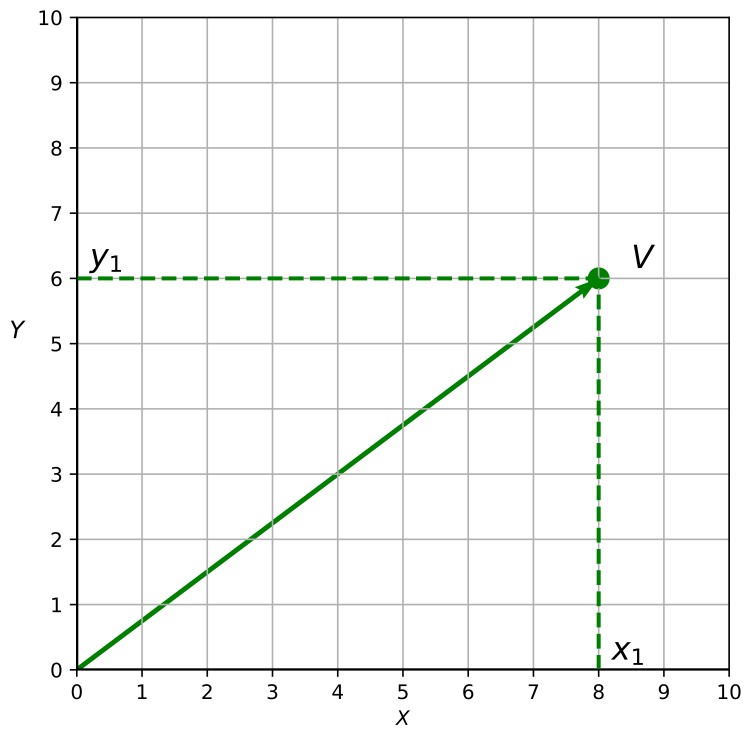
```
> import numpy as np
> X = np.arange(1,37).reshape(6,6)
> Y = X.reshape(4, 9)
array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24, 25, 26, 27],
       [28, 29, 30, 31, 32, 33, 34, 35, 36]])
```

Vectors



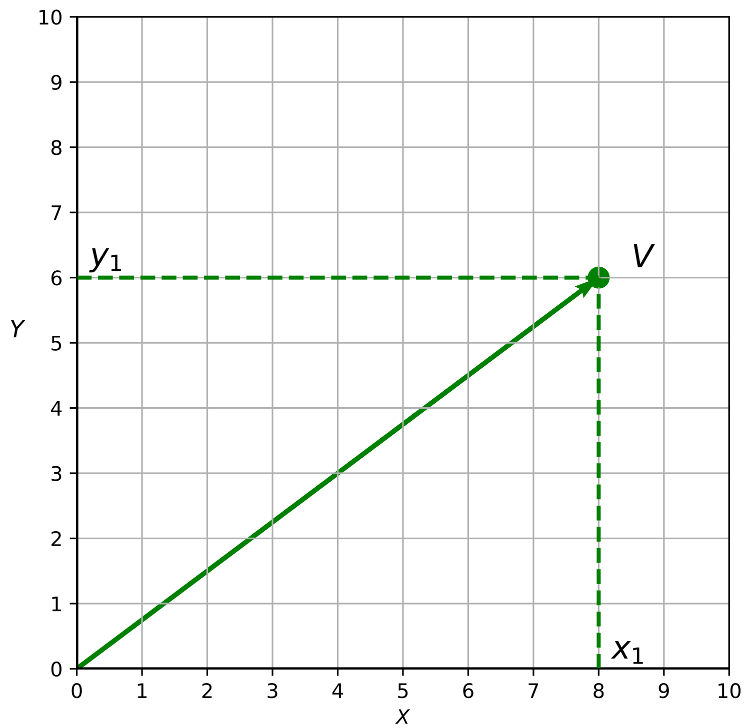
- an object with magnitude and direction

Vectors (cont'd)




```
> import numpy as np
> V = np.array([8,6])
array([8, 6])
```

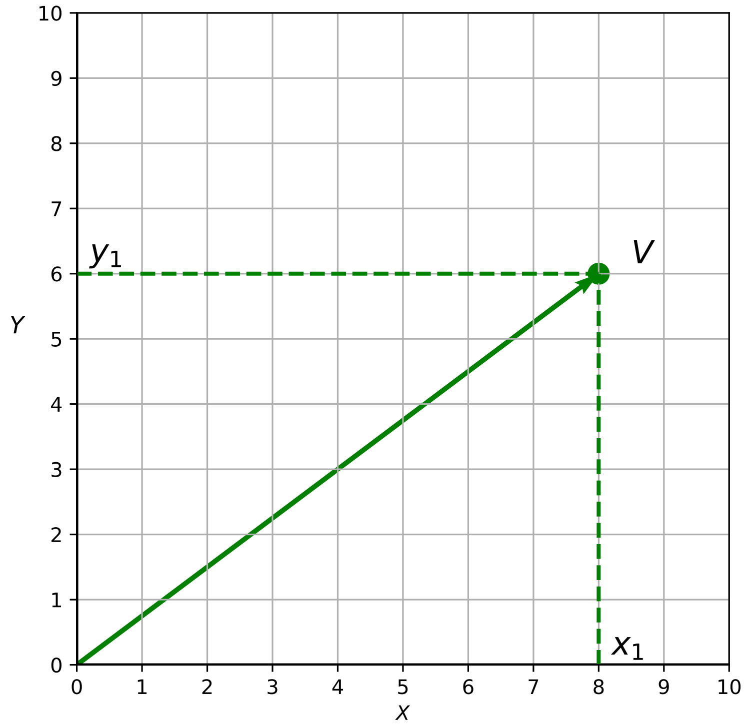
Euclidean Norm



- euclidean (L_2) norm:

$$||V||_2 = \sqrt{x_1^2 + y_1^2}$$

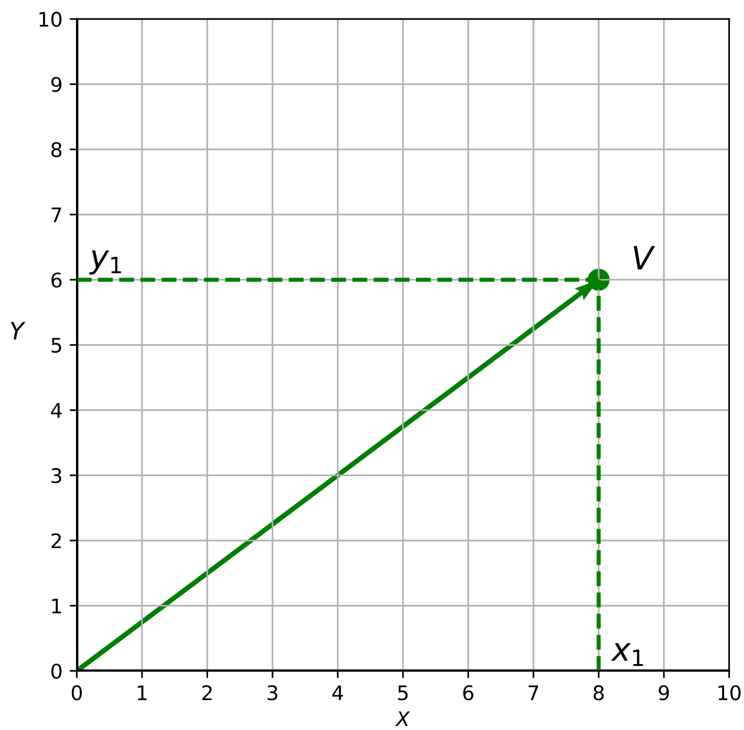
Manhattan Norm



- L_1 (Manhattan, taxicab, street) norm:

$$||V||_1 = |x_1| + |x_2|$$

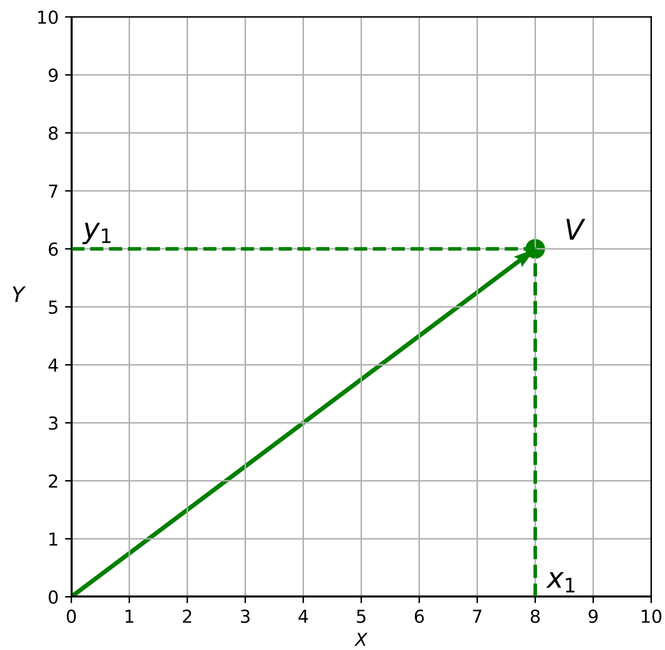
p-Norm



$$||V||_p = \left(|x_1|^p + |x_2|^p\right)^{1/p} \qquad , p \geq 1$$

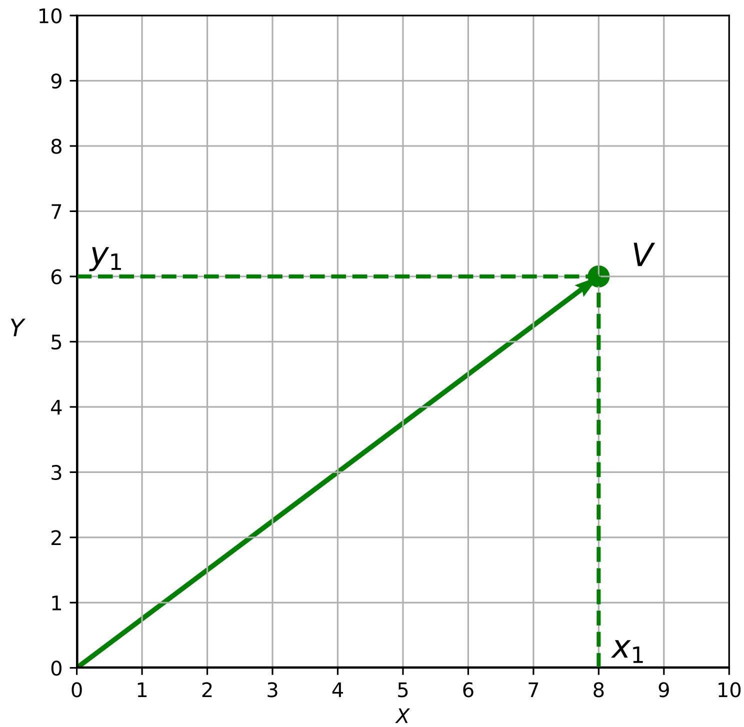
- for $\mapsto \infty$ we get $||V||_\infty = \max_i |x_i|$

Computing Norms



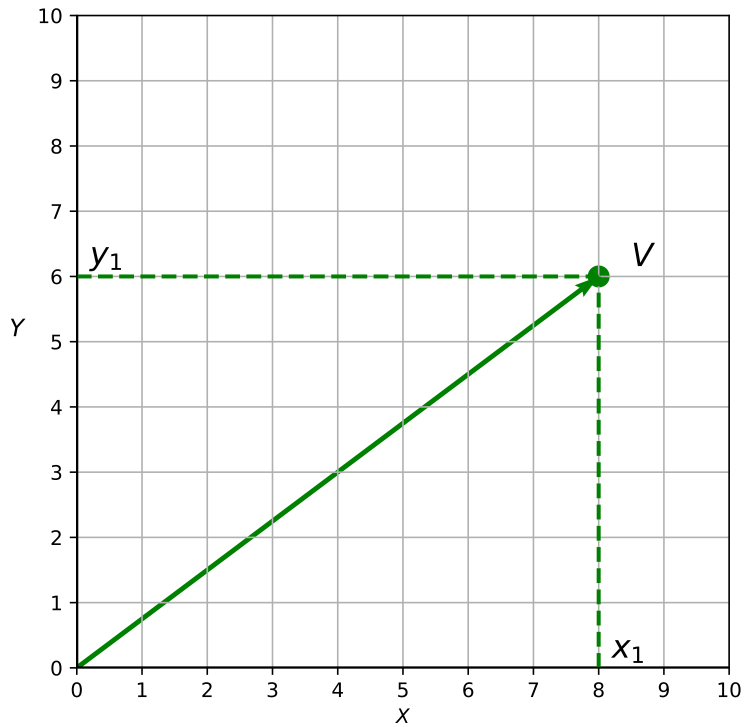

```
> V = np.array([8, 6])
> np.linalg.norm(V)
10.0
> np.linalg.norm(V, ord=1)
14.0
> np.linalg/norm(V, 1.5)
11.168500752960059
```

Computing Norms (cont'd)



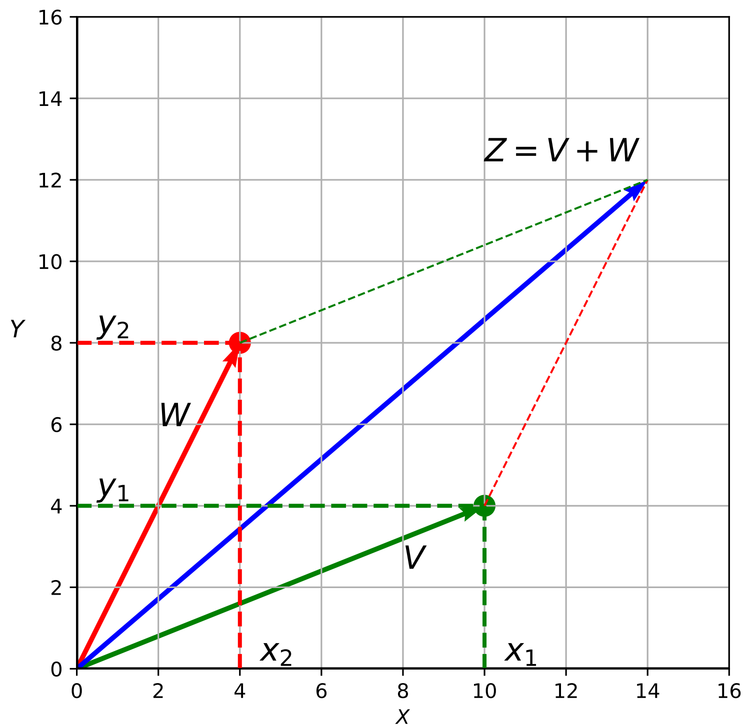

```
> np.linalg.norm(V, ord = 3)
8.9958828905508277
> np.linalg.norm(V, ord = 5)
8.3480553257954924
```

Computing Norms (cont'd)



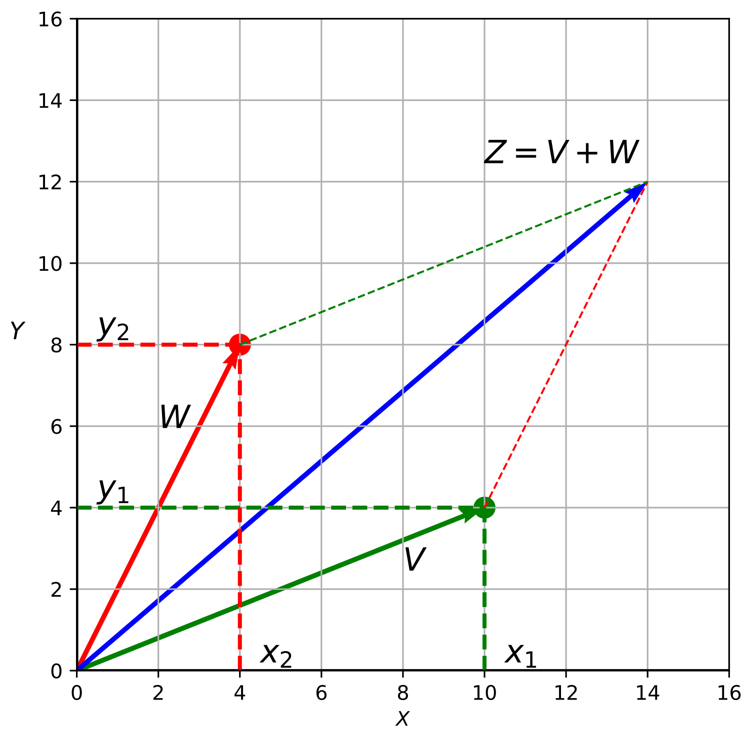

```
> np.linalg.norm(V, ord = 10)
8.043948299644665
> np.linalg.norm(V, ord = 20)
8.001266579538139
```

Vector Addition



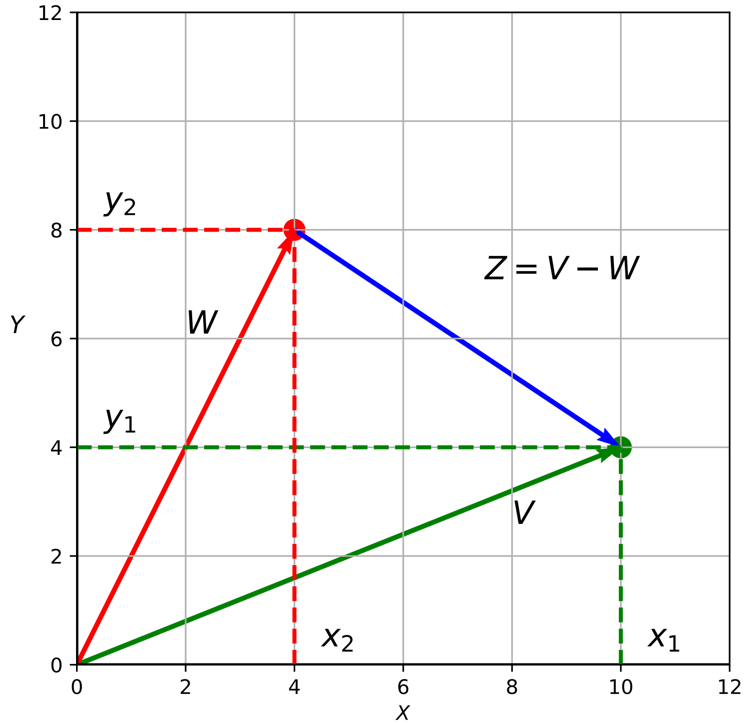
$$Z = V + W = (x_1, y_1) + (x_2, y_2)$$
$$= (x_1 + x_2, y_1 + y_2)$$

Vector Addition (cont'd)



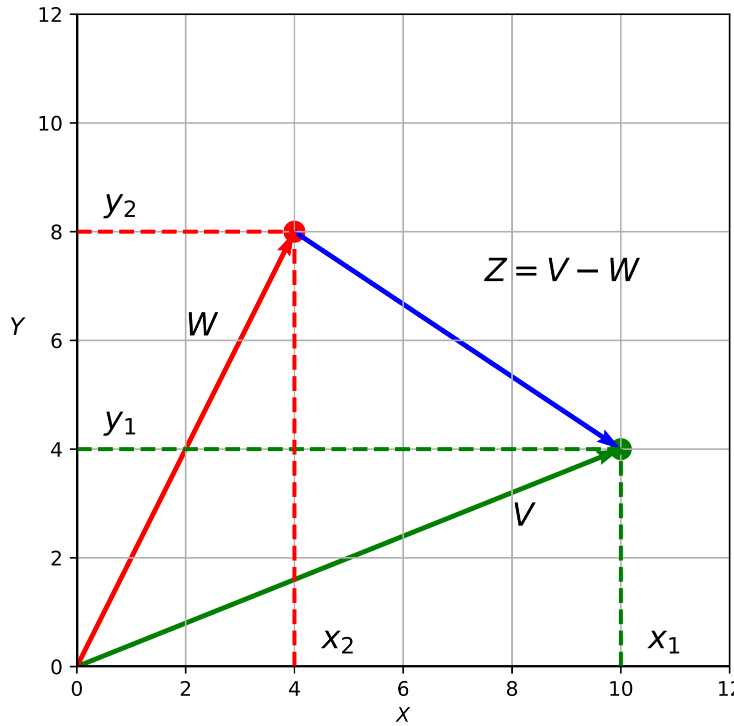

```
> V = np.array([8,6])
> W = np.array[(6, 8])
> Z = V + W
array([14, 14])
```

Vector Difference



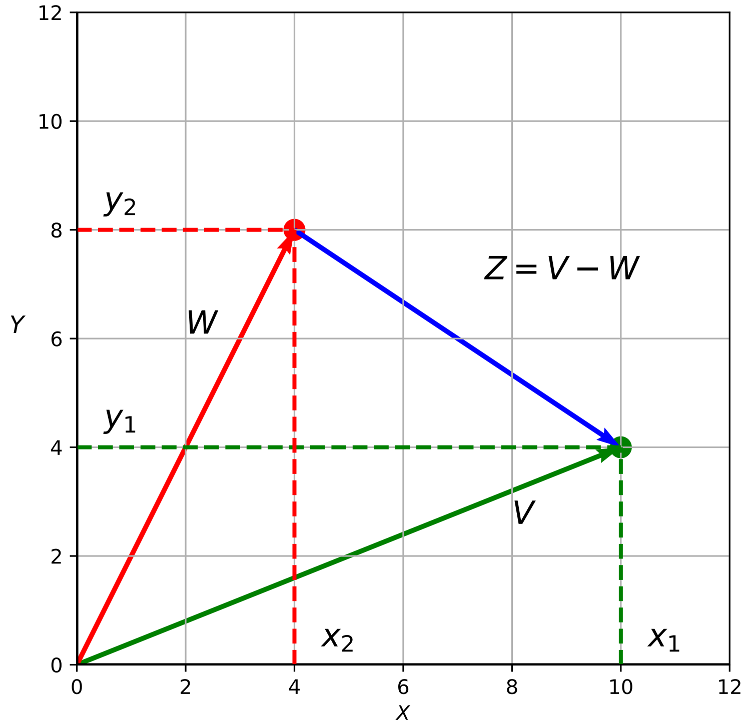
$$Z = V - W = \big(x_1, y_1\big) - \big(x_2, y_2\big)$$
$$= \big(x_1 - x_2, y_1 - y_2\big)$$

Vector Difference (cont'd)




```
> V = np.array([10, 4])
> W = np.array[(4, 8])
> Z = V - W
array([6, -4])
```

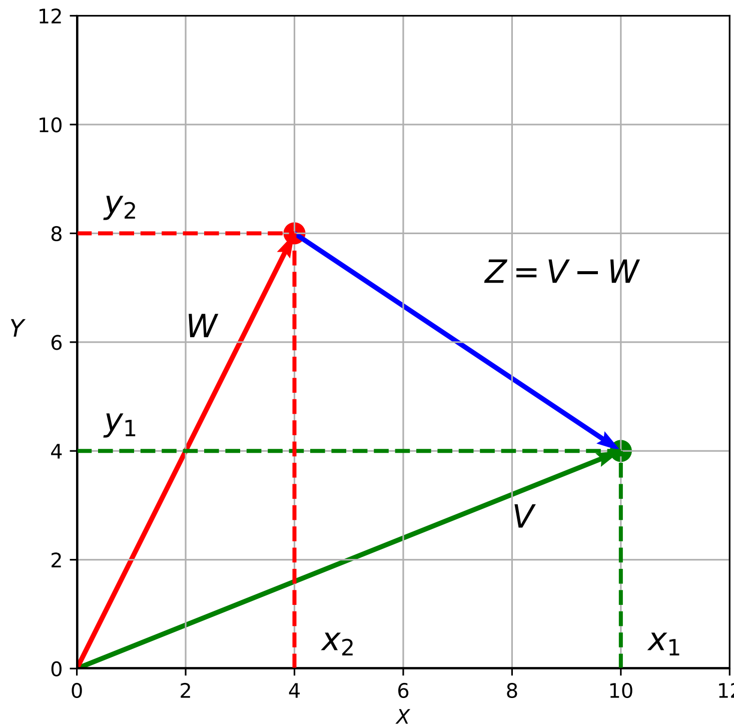
Vector Distances



$$||Z||_2 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$||Z||_1 = |x_1 - x_2| + |y_1 - y_2|$$

Vector Distances




```
> V = np.array([10,4]); W = np.array[(4, 8])
> Z = V - W
> np.linalg.norm(Z)
7.21
> np.linalg.norm(Z, ord=1)
10.0
```

Linear Algebra

```
> x = np.array([[1,2,3], 1,4,2],[1,10, -5]])
array([[ 1,  2,  3],
       [ 1,  4,  2],
       [ 1, 10, -5]])
> x_inv = np.linalg.inv(x)
array([[ 5.   , -5.   ,  1.   ],
       [-0.875,  1.   , -0.125],
       [-0.75 ,  1.   , -0.25 ]])
> y = x.dot(x_inv)  # or  y = np.dot(x, x_inv)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

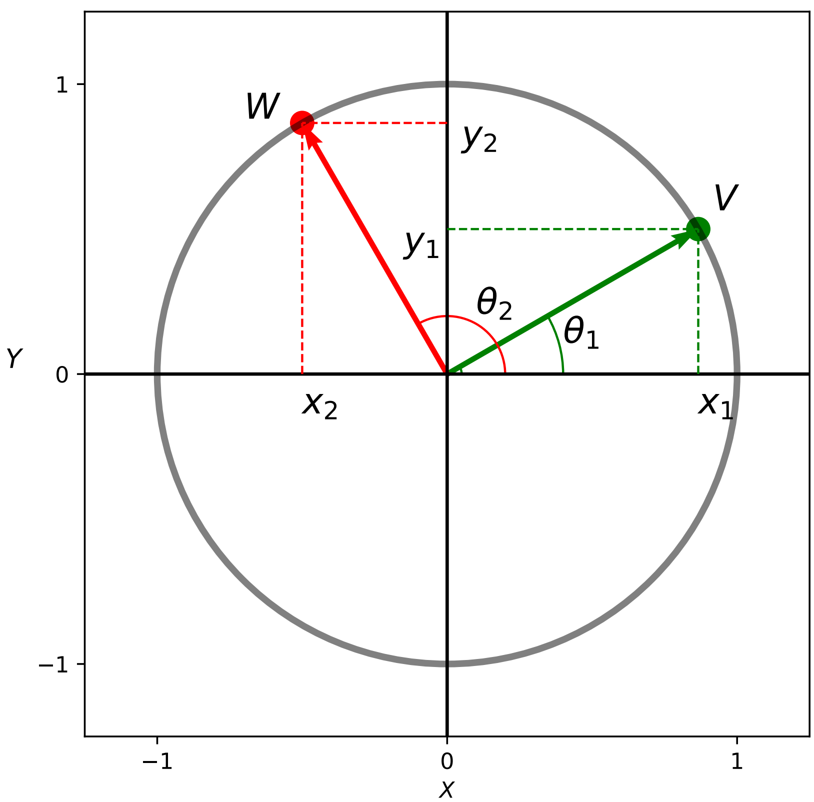
Dot (Inner) Product

$$
\begin{aligned}
&(x_1, x_2, \ldots, x_n) \cdot (y_1, y_2, \ldots, y_n) \\
&\qquad = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n
\end{aligned}
$$

```
> x = np.array([1,2,3])
array([1, 2, 3])
> y = np.array([3,9, 2])
array([3, 9, 2])
> z = np.dot(x, y)
27
```

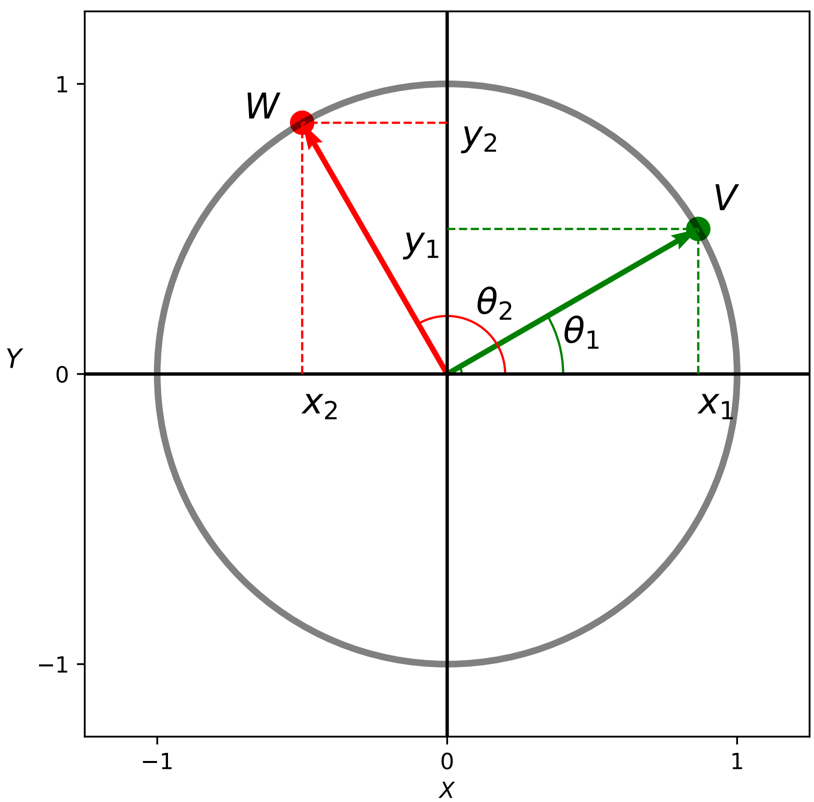
- geometric meaning

Geometric Meaning



$$V \cdot W = (x_1, y_1) \cdot (x_2, y_2) = x_1 x_2 + y_1 y_2$$
$$= \cos\theta_1 \cos\theta_2 + \sin\theta_1 \sin\theta_2$$

Geometric Meaning



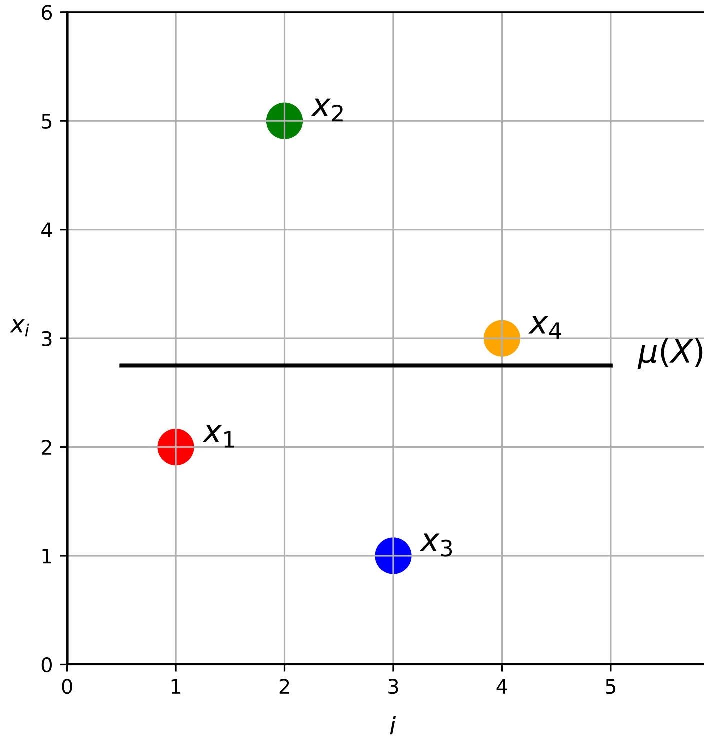
- cosine similarity

$$\cos(\theta_2 - \theta_1) = \cos\theta_1 \cos\theta_2 + \sin\theta_1 \sin\theta_2$$

Applying Numpy Functions to Rows and Columns

```
> x = np.array([[1,2,3], [1,4,2], [1,10, -5]])
array([[ 1,  2,  3],
       [ 1,  4,  2],
       [ 1, 10, -5]])
> sum_columns = np.sum(x, axis = 0)
array([ 3, 16,  0])
> sum_rows = np.sum(x, axis = 1)
array([6, 7, 6])
```

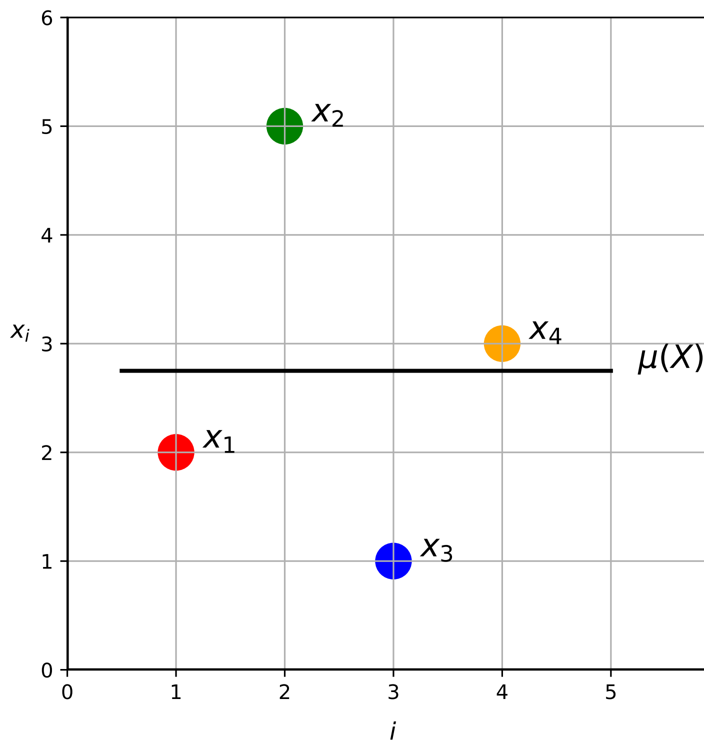
Stat.Functions: Mean



$$X = \left(x_1, \ldots, x_n\right)$$

$$\mu_x = \frac{1}{n} \sum_i x_i$$

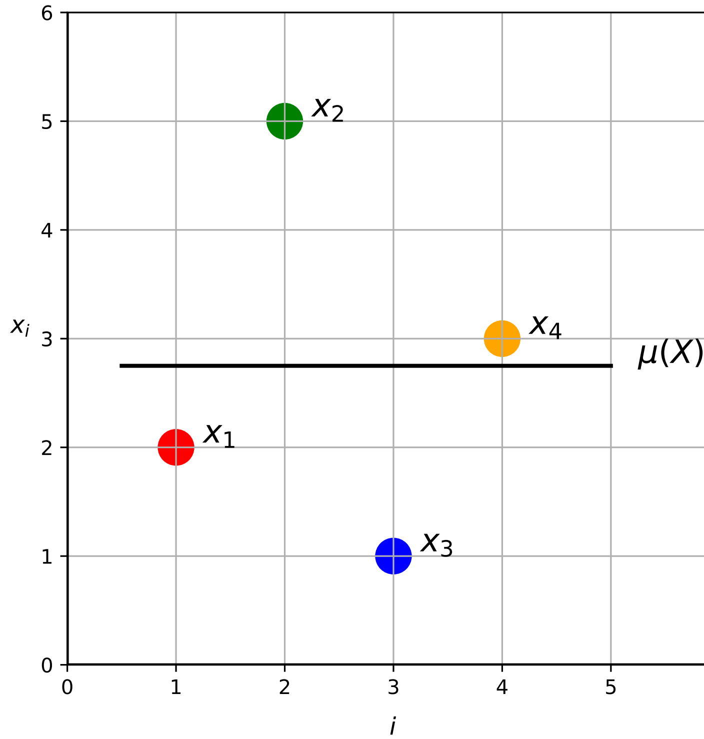
Mean (cont'd)



$$X = (2, 5, 1, 3), \quad n = 4$$

$$\mu_x = \frac{(2+5+1+3)}{4} = 2.75$$

Mean (cont'd)




```
> x = np.array([2, 5, 1, 3])
> mean = np.mean(x)
2.75
```

Properties of the Mean

- effect of shifts

$$\mu\big(X + b\big) = \mu\big(X\big) + b$$

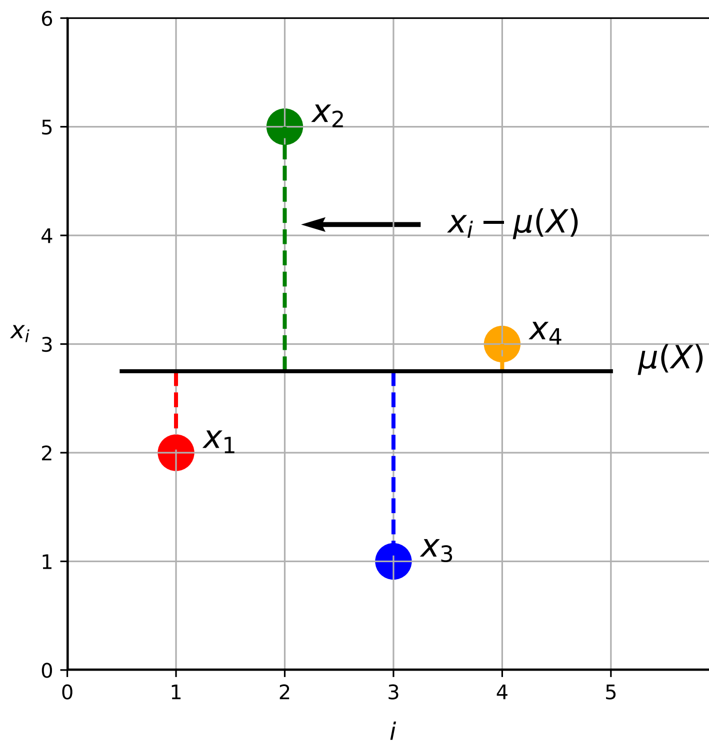
- effect of scaling

$$\mu\big(aX\big) = a\mu\big(X\big)$$

- linear transformation

$$\mu\big(aX + b\big) = a\mu\big(X\big) + b$$

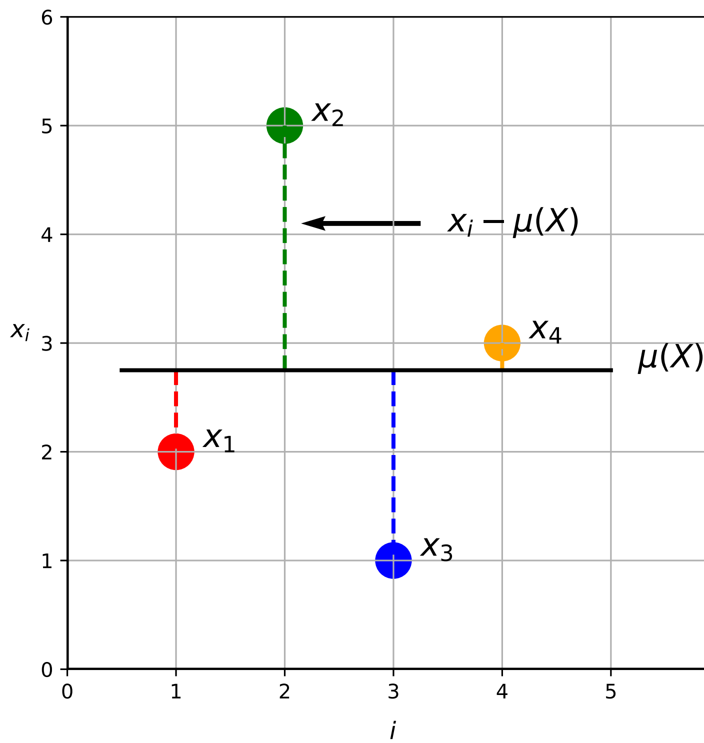
Variance & Deviation



- "spread" around mean

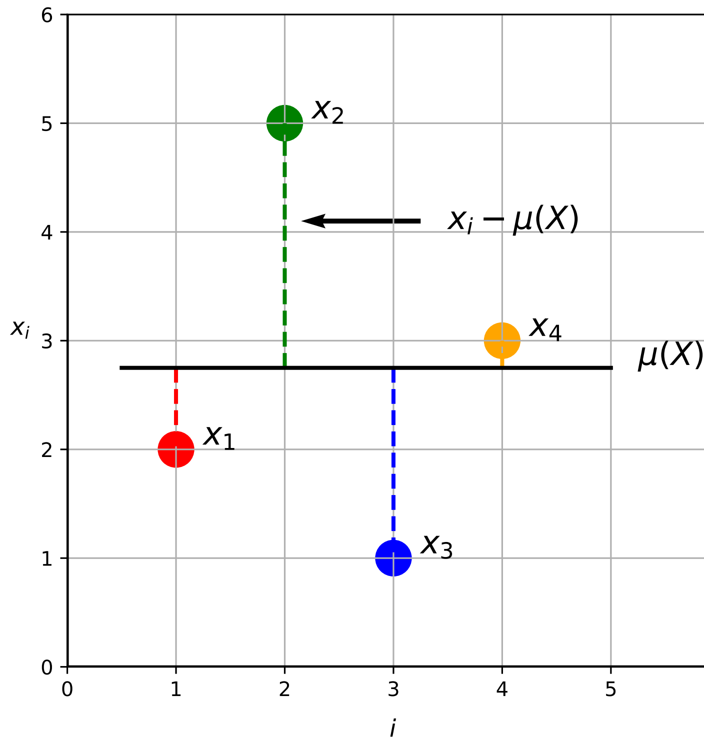
$$\sigma^2(X) = \frac{1}{n}\sum_i (x_i - \mu_x)^2$$

Variance & Deviation



$$\sigma^2(X) = \frac{1}{n}\sum_i (x_i - \mu_x)^2$$

Variance & Deviation

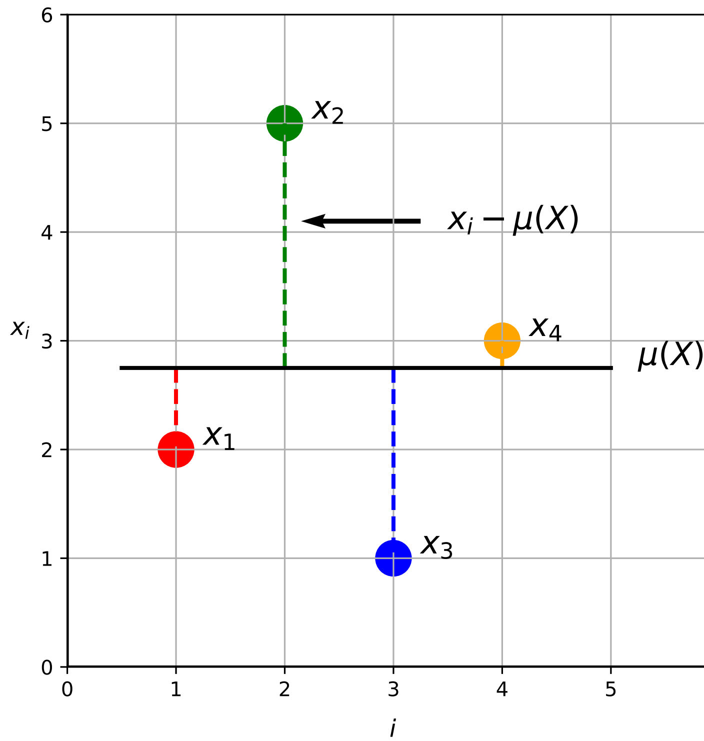


$$X = (2, 5, 1, 3), \quad \mu(X) = 2.75$$

$$\sigma^2(X) = \frac{(2-2.75)^2 + (5-2.75)^2 + (1-2.75)^2 + (3-2.75)^2)}{4}$$

$$= \frac{0.5625 + 5.0625 + 3.0625 + 0.0625}{4} = \frac{8.75}{4} = 2.1875$$

Variance & Deviation




```
> x = np.array([2, 5, 1, 3])
> np.var(x)
2.1875
> np.std(x)
1.4790
```

Properties of the Variance

- invariant under shifts

$$\sigma^2(X + b) = \sigma^2(X)$$

- effect of scaling

$$\sigma^2(aX) = a^2\sigma^2(X)$$

- linear transformation

$$\sigma^2(aX + b) = a^2\sigma^2(X)$$

Miscellaneous Statistical Functions

```
> x = np.array([2, 5, 1, 3]
> median = np.percentile(x, 50)
2.5
> cum_sum = np.cumsum(x)
array([ 2,  7,  8, 11], dtype=int32)
> cum_prod = np.cumprod(x)
array([ 2, 10, 10, 30], dtype=int32)
```

Sum of Two Random Variables

$$Z = X + Y$$

- mean

$$\mu(Z) = \mu(X) + \mu(Y)$$

- variance

$$\sigma^2(Z) = \sigma^2(X) + \sigma^2(Y) + 2 \cdot \mathrm{Cov}(X, Y))$$

Covariance

- measures joint variability of two random variables
- computed as the mean value of the product of deviations from respective means

$$\mathrm{Cov}(\mathrm{X},\mathrm{Y}) = \sum_i \big(x_i - \mu_x\big)\big(y_i - \mu_y\big)$$
$$= \big(X - \mu_x, Y - \mu_y\big)$$

Weighted Sum of n Random Variables

$$Z = w_1 X_1 + w_2 X_2 + \cdots + w_n X_n$$

- mean is weighted sum of means

$$\mu(Z) = w_1 \mu(X_1) + w_2 \mu(X_2) + \cdots + w_n \mu(X_n)$$

- variance

$$\sigma^2(Z) = \sum_{i=1}^{n} w_i^2 \sigma^2(X_i) + \sum_{i,j=1}^{n} w_i w_j \mathrm{Cov}(X_i, X_j)$$

Pearson Correlation

- measures the extend to which variables are linearly related
- is a number form -1 to 1
- invariant to shifts

$$\rho\big(X,Y\big) = \frac{COV\big(X,Y\big)}{\sigma\big(X\big)\sigma\big(Y\big)}$$

$$= \frac{\big(X-\mu_x, Y-\mu_y\big)}{||X-\mu_x||\;||Y-\mu_y||}$$

Cosine Similarity and Pearson Correlation

$$
\begin{aligned}
\rho\Big(X, Y\Big) &= \frac{COV\Big(X, Y\Big)}{\sigma\Big(X\Big)\sigma\Big(Y\Big)} \\
&= \frac{\sum_i\Big(x_i - \mu_x\Big)\Big(y_i - \mu_y\Big)}{\sqrt{\sum_i\Big(x_i - \mu_x\Big)^2}\sqrt{\sum_i\Big(y_i - \mu_y\Big)^2}} \\
&= \frac{\Big(X - \mu_x, Y - \mu_y\Big)}{||X - \mu_x||\;||Y - \mu_y||} \\
&= \frac{\Big(X - \mu_x\Big)\cdot\Big(Y - \mu_y\Big)}{||X - \mu_x||\;||Y - mu_y||} \\
&= \text{CosineSim}\Big(X, Y\Big)
\end{aligned}
$$

A Numerical Dataset

object x_i	Height (H)	Weight (W)	Foot (F)	Label (L)
x_1	5.00	100	6	green
x_2	5.50	150	8	green
x_3	5.33	130	7	green
x_4	5.75	150	9	green
x_5	6.00	180	13	red
x_6	5.92	190	11	red
x_7	5.58	170	12	red
x_8	5.92	165	10	red

- $N = 8$ items
- $M = 3$ (unscaled) attributes

Code for the Dataset

```python
import pandas as pd
data = pd.DataFrame(
        {'id': [ 1,2,3,4,5,6,7,8],
        'Label': ['green','green','green','green',
                   'red','red','red','red'],
        'Height': [5, 5.5, 5.33, 5.75,
                   6.00, 5.92,  5.58, 5.92],
        'Weight': [100, 150, 130, 150,
                   180, 190, 170, 165],
        'Foot': [6, 8, 7, 9, 13, 11, 12, 10]},
         columns = ['id', 'Height', 'Weight',
                      'Foot', 'Label'] )
```

```
ipdb> data
  id Height Weight Foot Label
0  1   5.00    100    6 green
1  2   5.50    150    8 green
2  3   5.33    130    7 green
3  4   5.75    150    9 green
4  5   6.00    180   13   red
5  6   5.92    190   11   red
6  7   5.58    170   12   red
7  8   5.92    165   10   red
```

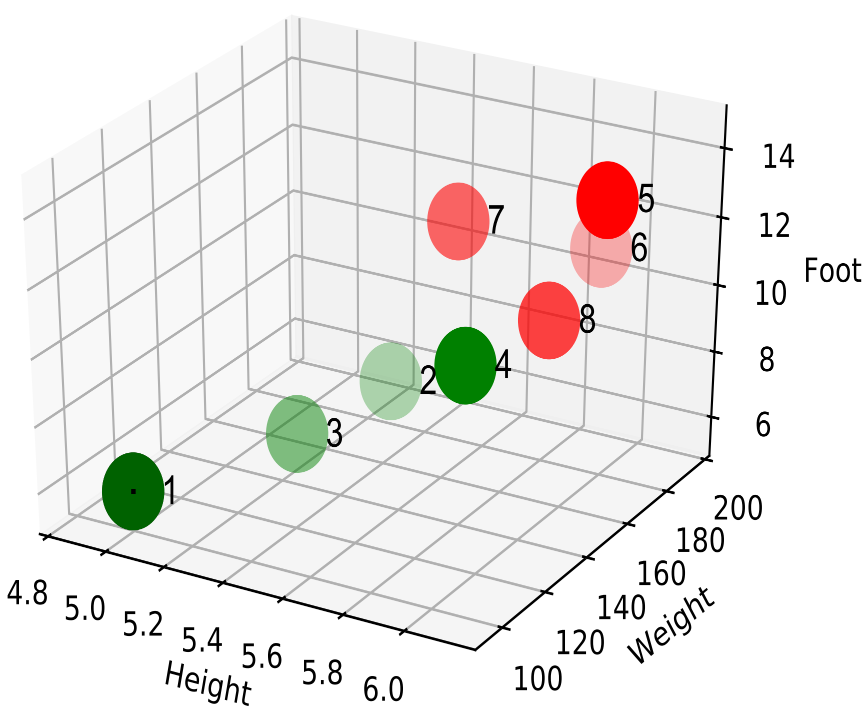
newpage

Desribing the Dataset

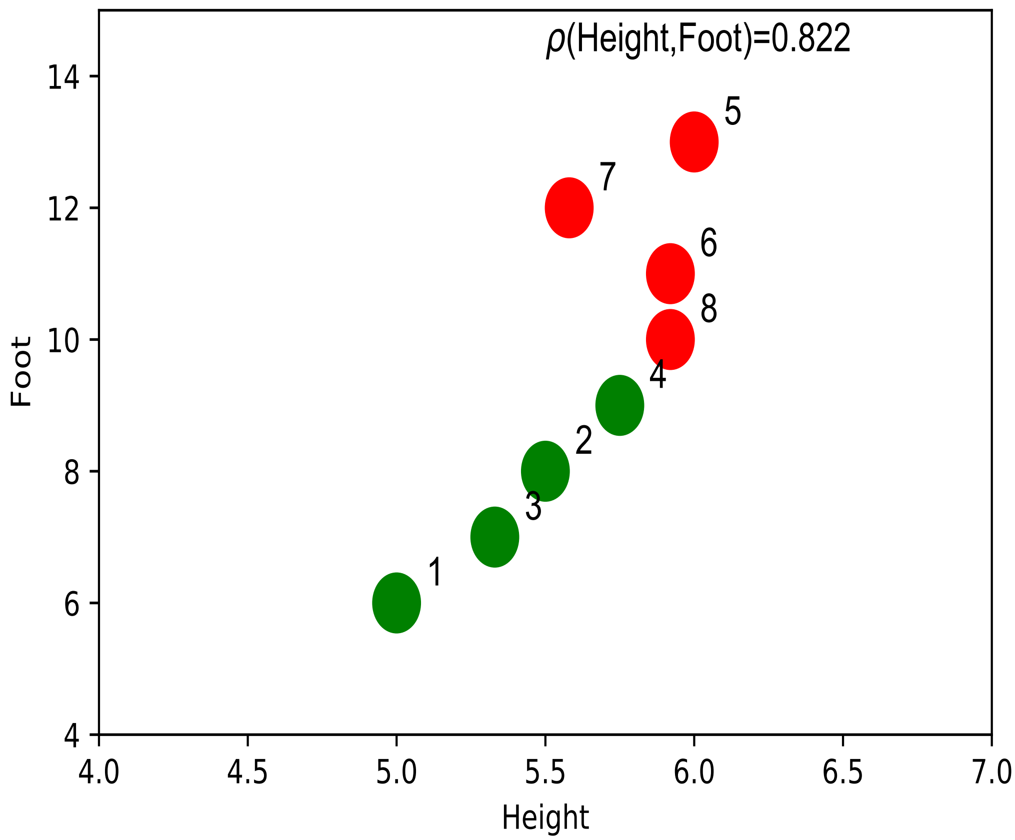
```
import pandas as pd
data = pd.DataFrame(
        {'id': [ 1,2,3,4,5,6,7,8],
        'Label': ['green','green','green','green',
                   'red','red','red','red'],
        'Height': [5, 5.5, 5.33, 5.75,
                   6.00, 5.92,  5.58, 5.92],
        'Weight': [100, 150, 130, 150,
                   180, 190, 170, 165],
        'Foot': [6, 8, 7, 9, 13, 11, 12, 10]},
         columns = ['id', 'Height', 'Weight',
                       'Foot', 'Label'] )
```

```
ipdb> data.describe()
            id    Height      Weight      Foot
count  8.00000  8.000000    8.000000   8.00000
mean   4.50000  5.625000  154.375000   9.50000
std    2.44949  0.343428   28.962722   2.44949
min    1.00000  5.000000  100.000000   6.00000
25%    2.75000  5.457500  145.000000   7.75000
50%    4.50000  5.665000  157.500000   9.50000
75%    6.25000  5.920000  172.500000  11.25000
max    8.00000  6.000000  190.000000  13.00000
```

A Dataset Illustration

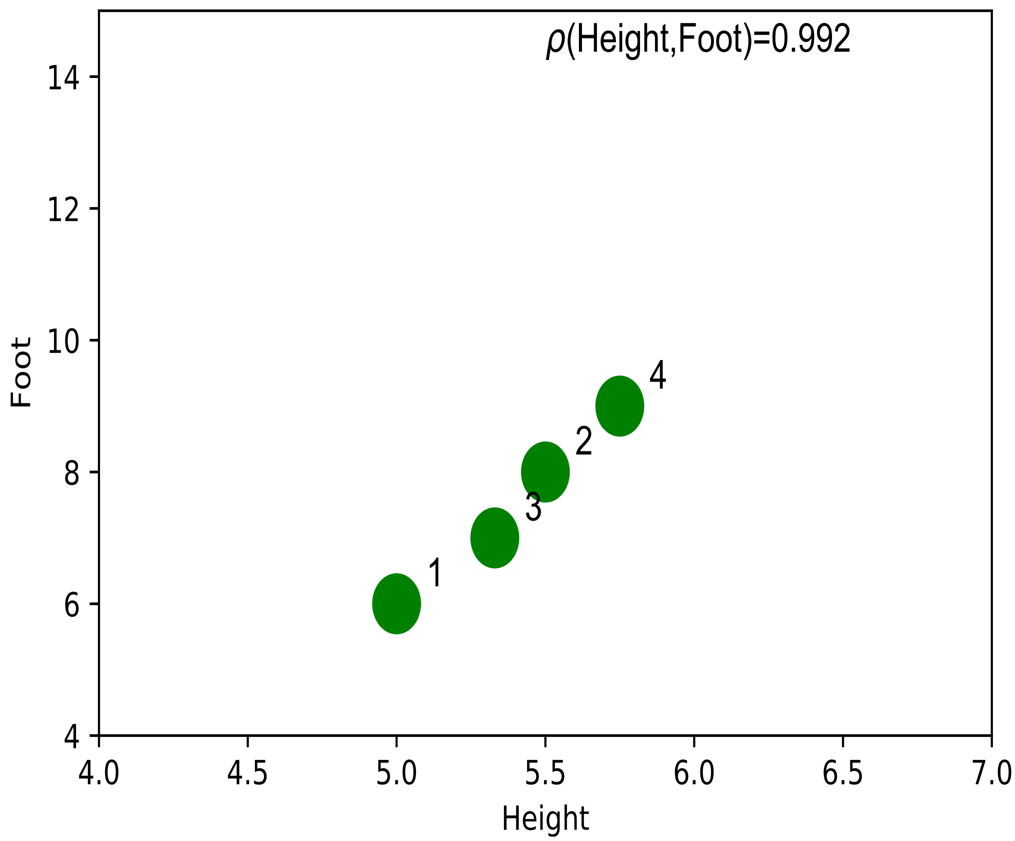


Computing Correlation



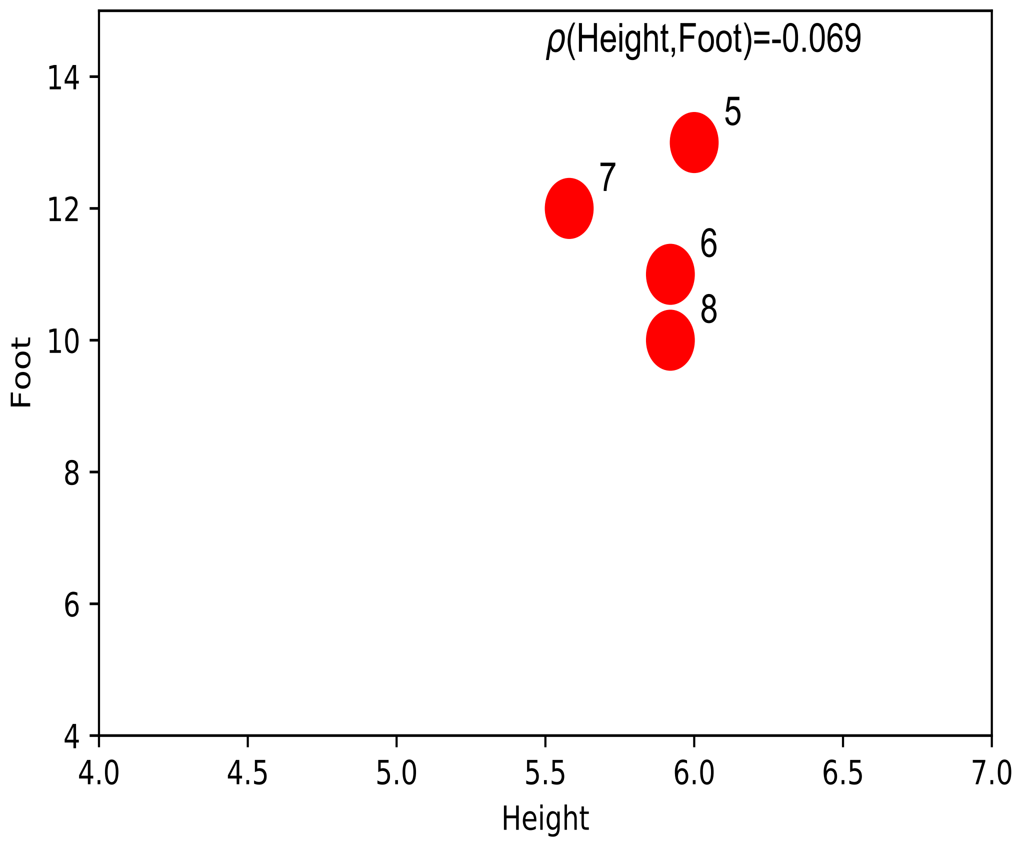

```
> height = np.array([5,5.5,5.33,5.75,6,5.92,5.58,5.92])
> weight = np.array([6,8,7,9,13,11,12,10])
> correlation = np.corrcoef(height,foot)[0][1]
0.822
```

Subset Correlation




```
> height = np.array([5,5.5,5.33,5.75])
> weight = np.array([6,8,7,9])
> correlation = np.corrcoef(height,foot)[0][1]
0.992
```

Subset Correlation




```
> height = np.array([6,5.92,5.58,5.92])
> weight = np.array([13,11,12,10])
> correlation = np.corrcoef(height,foot)[0][1]
0.069
```

Concepts Check:

(a) lists vs. Numpy arrays

(b) universal functions

(c) vectorized computations

(d) broadcasting

(e) sequences with *linspace*() and *arange*()

(f) sorting and searching

(g) vectors and matrices

Concepts Check:

(a) distances (Euclidean, street, Minkowski)

(b) inner products

(c) cosine similarity

(d) statistical functions

(e) properties of mean and variance

(f) covariance and correlation