PFE 5

Dr W Palubicki

Statistics

Why statistics?

- English believe 24% of their population are Muslim

Why statistics?

- English believe 24% of their population are Muslim → reality 5%

Why statistics?

- English believe 24% of their population are Muslim → reality 5%
- Saudis believe 28% of their population are overweight

Why statistics?

- English believe 24% of their population are Muslim → reality 5%
- Saudis believe 28% of their population are overweight → reality 71%

Why statistics?

- English believe 24% of their population are Muslim → reality 5%
- Saudis believe 28% of their population are overweight → reality 71%
- Japanese believe 56% of their population live in the countryside

Why statistics?

- English believe 24% of their population are Muslim → reality 5%
- Saudis believe 28% of their population are overweight → reality 71%
- Japanese believe 56% of their population live in the countryside → reality 7%

Source Ipsos MORI

Statistical inference

- Use data from a sampling measurement to infer information which is generally applicable

Example: drug testing

- 50 patients received new pain medication whereas a similar group of 50 patients were treated with older medication. Mean scores of perceived pain were 4.1 for the new medication and 4.5 for the old.

Example: drug testing

- Statistical inference addresses:
  - How can we estimate the difference of the effect of medication?
  - How can we quantify the precision of that estimate?

Statistical inference

- **Effect size**: the quantification of an effect, e.g. in the simplest case a single number
- **Confidence interval** and/or the **standard error**: the precision of the quantification (estimate)

numpy.mean

numpy.**mean**(*a*, *axis=None*, *dtype=None*, *out=None*, *keepdims=False*) [source]

Compute the arithmetic mean along the specified axis.

Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs.

Examples

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.mean(a)
2.5
>>> np.mean(a, axis=0)
array([ 2.,  3.])
>>> np.mean(a, axis=1)
array([ 1.5,  3.5])
```

numpy.std Standard deviation

numpy.std(*a, axis=None, dtype=None, out=None, ddof=0, keepdims=False*) [source]

Compute the standard deviation along the specified axis.

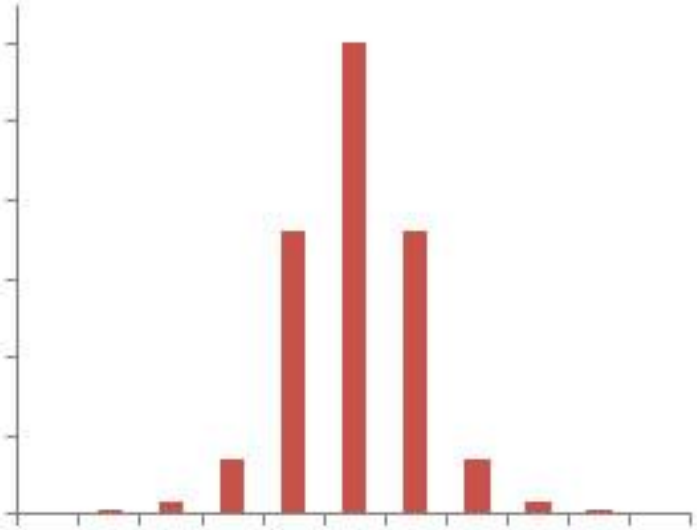
Returns the standard deviation, a measure of the spread of a distribution, of the array elements. The standard deviation is computed for the flattened array by default, otherwise over the specified axis.

Examples

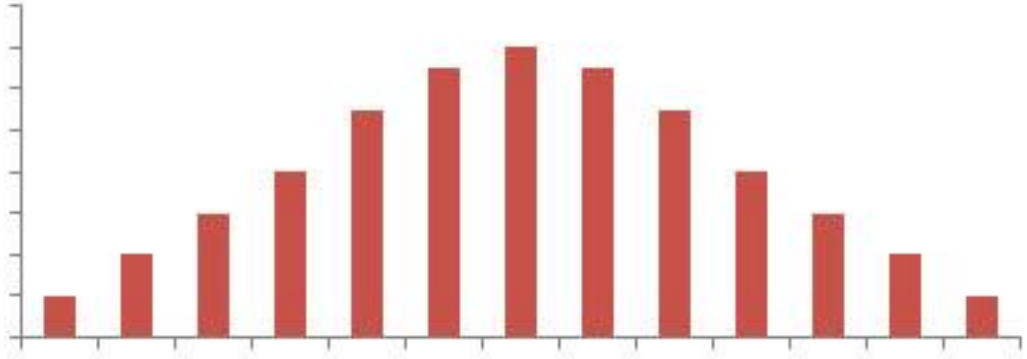
```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1.,  1.])
>>> np.std(a, axis=1)
array([ 0.5,  0.5])
```

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2, \text{ std} = \sigma$$

Assessing variance with standard deviations



Histogram of distribution with small SD



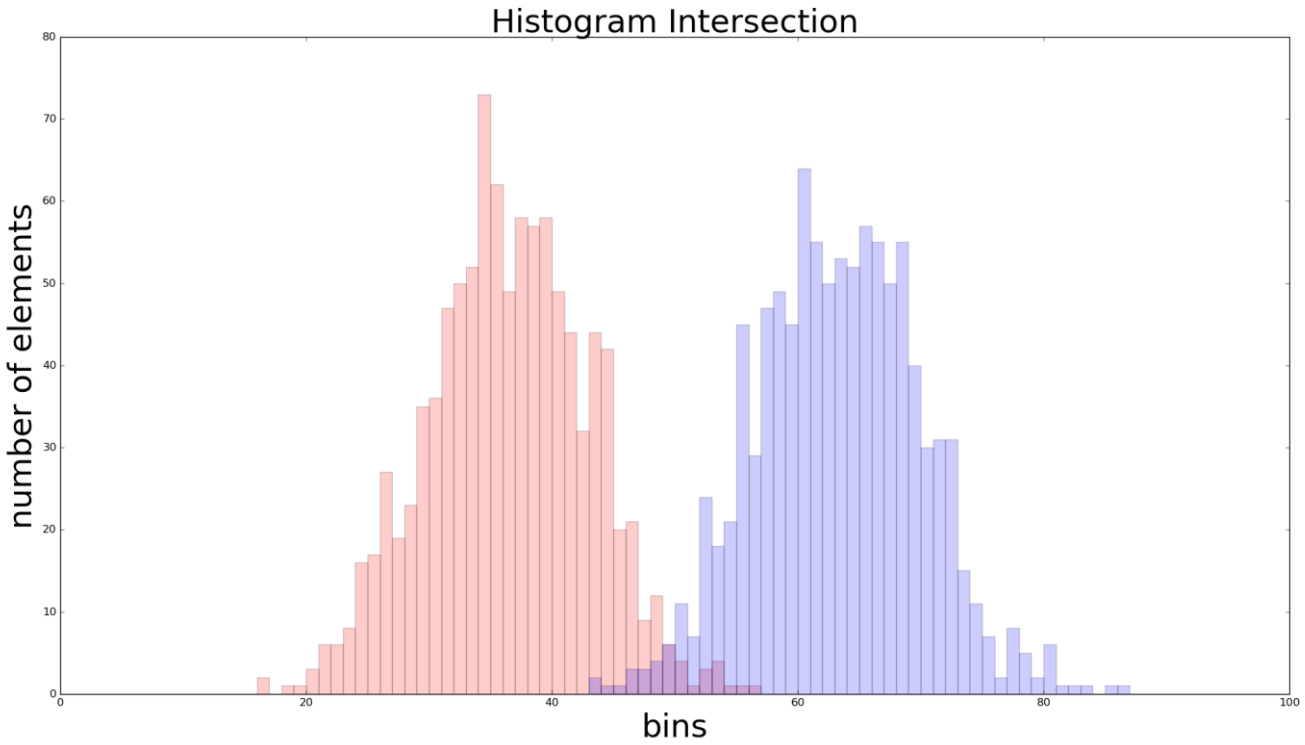
Histogram of distribution with large SD

Comparing distributions

- Difference of means: e.g. mu1 – mu2

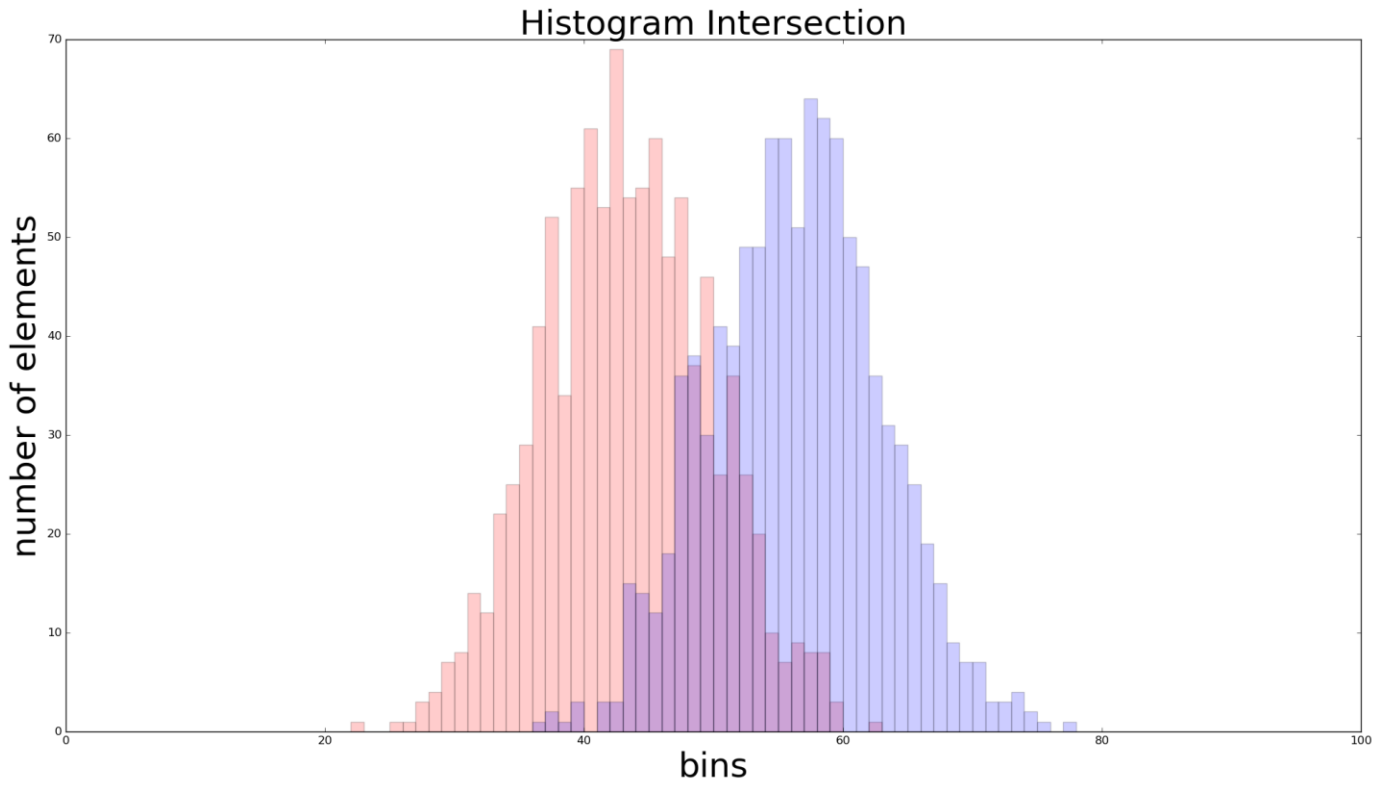
Comparing distributions

- Overlap of distributions



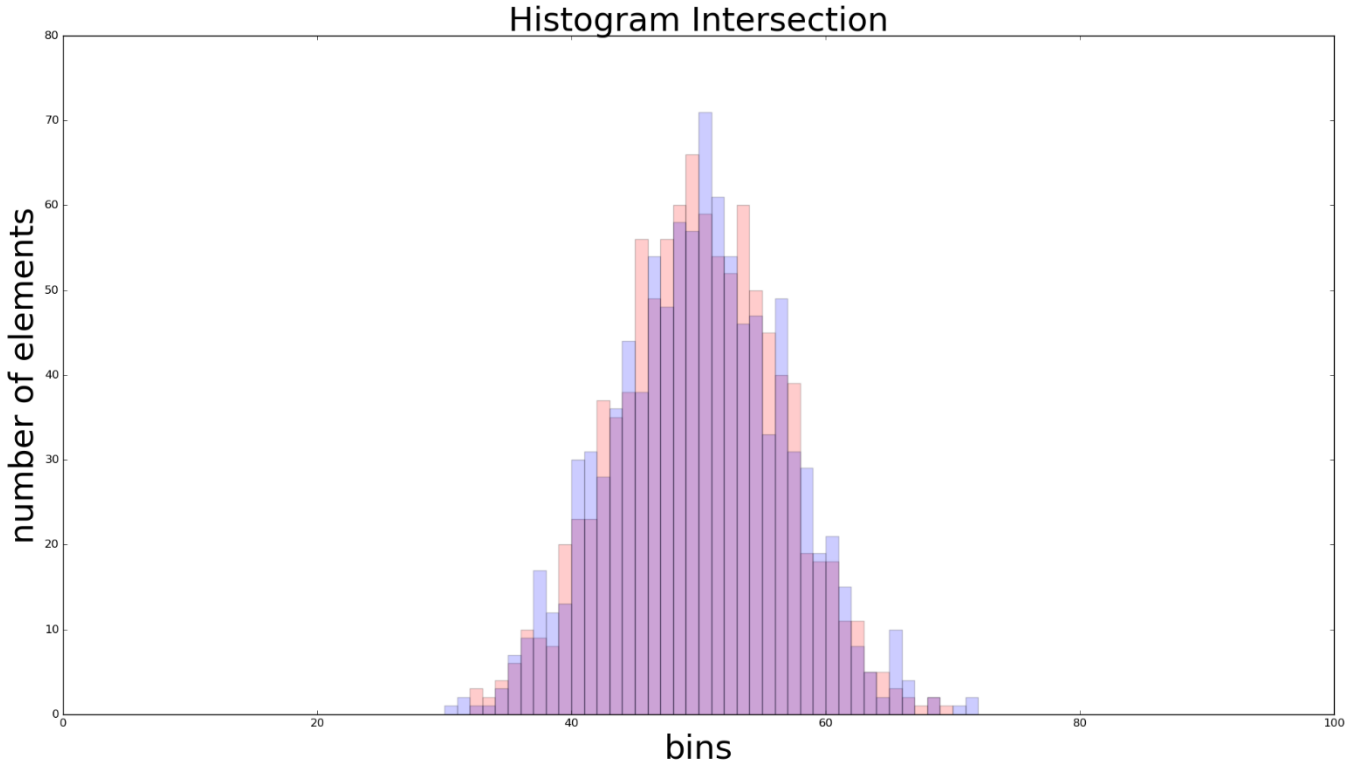
Comparing distributions

- Overlap of distributions



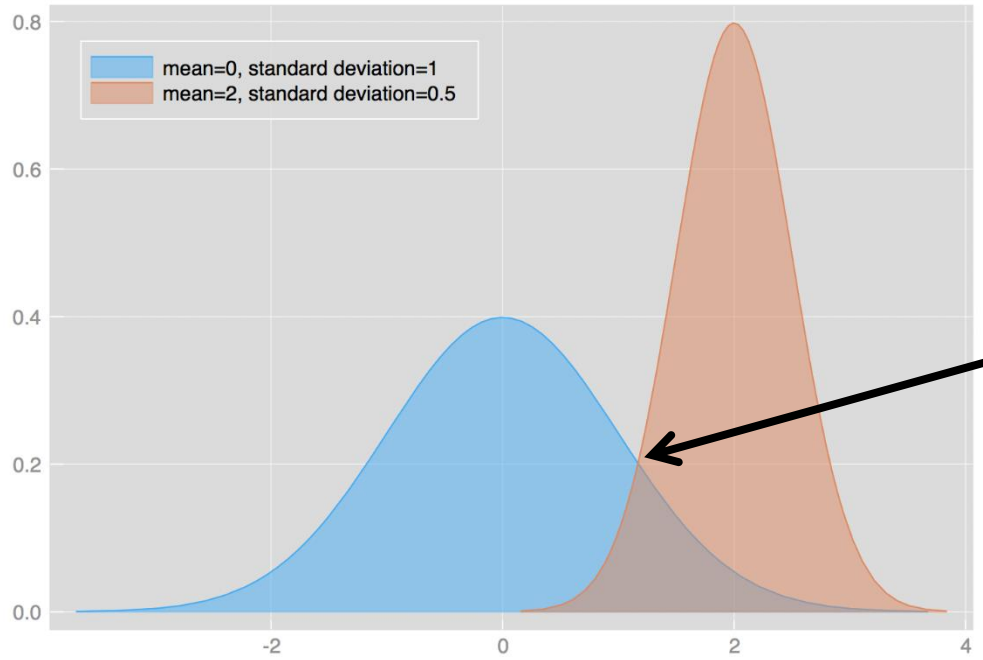
Comparing distributions

- Overlap of distributions



Computing overlap

- Define a threshold between the means of the distributions:
  - `thres = (std1 * mu2 + std2 * mu1) / (std1 + std2)`



threshold

Computing overlap

- Compute number of data points below threshold:

```
sample1_below_thres= sum(sample1 < thres)
sample2_above_thres= sum(sample2 > thres)
```

Computing overlap

- Compute number of data points below threshold:

```
sample1_below_thres= sum(sample1 < thres) e.g. 100
sample2_above_thres= sum(sample2 > thres) e.g. 200
```

Computing overlap

- Compute number of data points below threshold:

```
sample1_below_thres= sum(sample1 < thres) e.g. 100
sample2_above_thres= sum(sample2 > thres) e.g. 200

sample1_overlap = sample1_below_thres / len(sample1)
sample2_overlap = sample2_above_thres / len(sample2)
```

Computing overlap

- Compute number of data points below threshold:

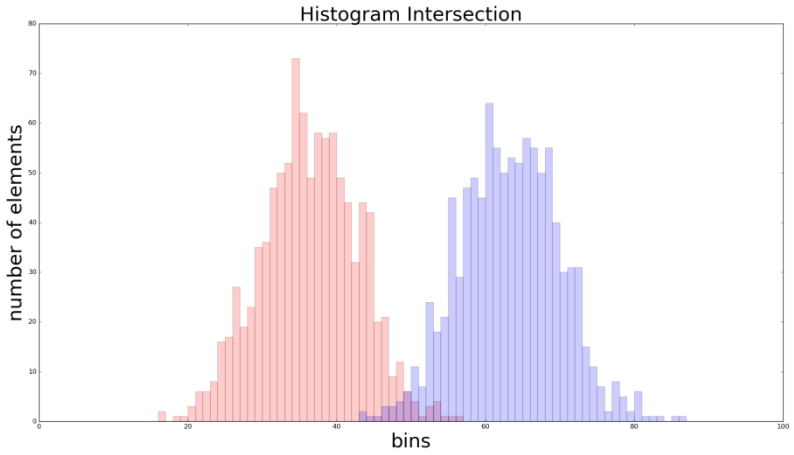
```
sample1_below_thres= sum(sample1 < thres) e.g. 100
sample2_above_thres= sum(sample2 > thres) e.g. 200

sample1_overlap = sample1_below_thres / len(sample1)
sample2_overlap = sample2_above_thres / len(sample2)
e.g. 0.2 and 0.3 (20% and 30%)
```

Computing overlap

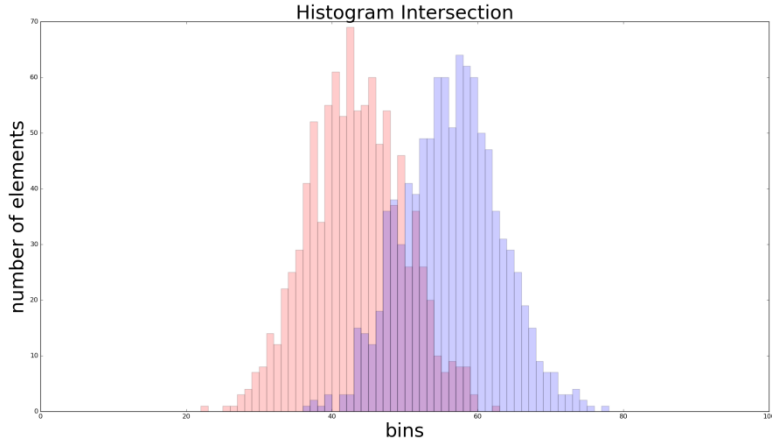
- Compute number of data points below threshold:

```
sample1_below_thres= sum(sample1 < thres)
sample2_above_thres= sum(sample2 > thres)

sample1_overlap = sample1_below_thres / len(sample1)
sample2_overlap = sample2_above_thres / len(sample2)

misclassification_rate = (sample1_overlap +
sample2_overlap) / 2
```

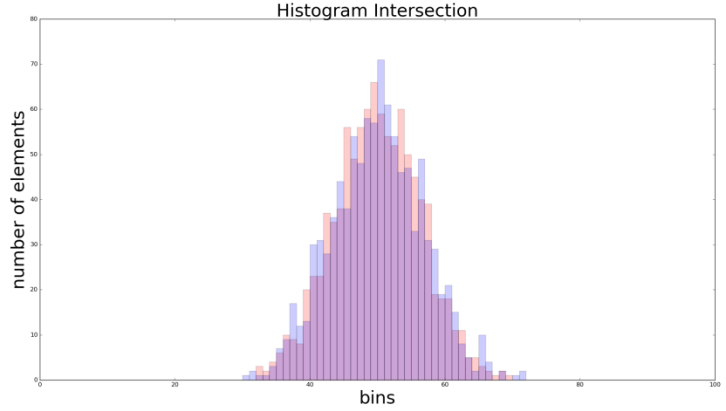
Example misclassification rates



~0.05



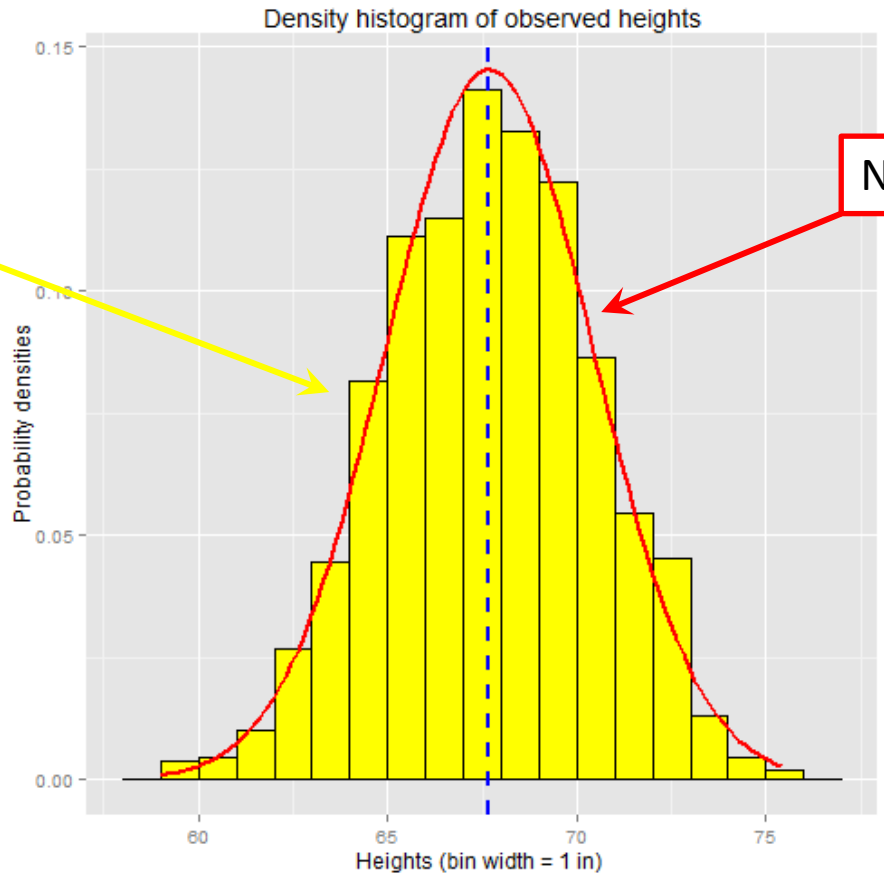
~0.2



~0.99

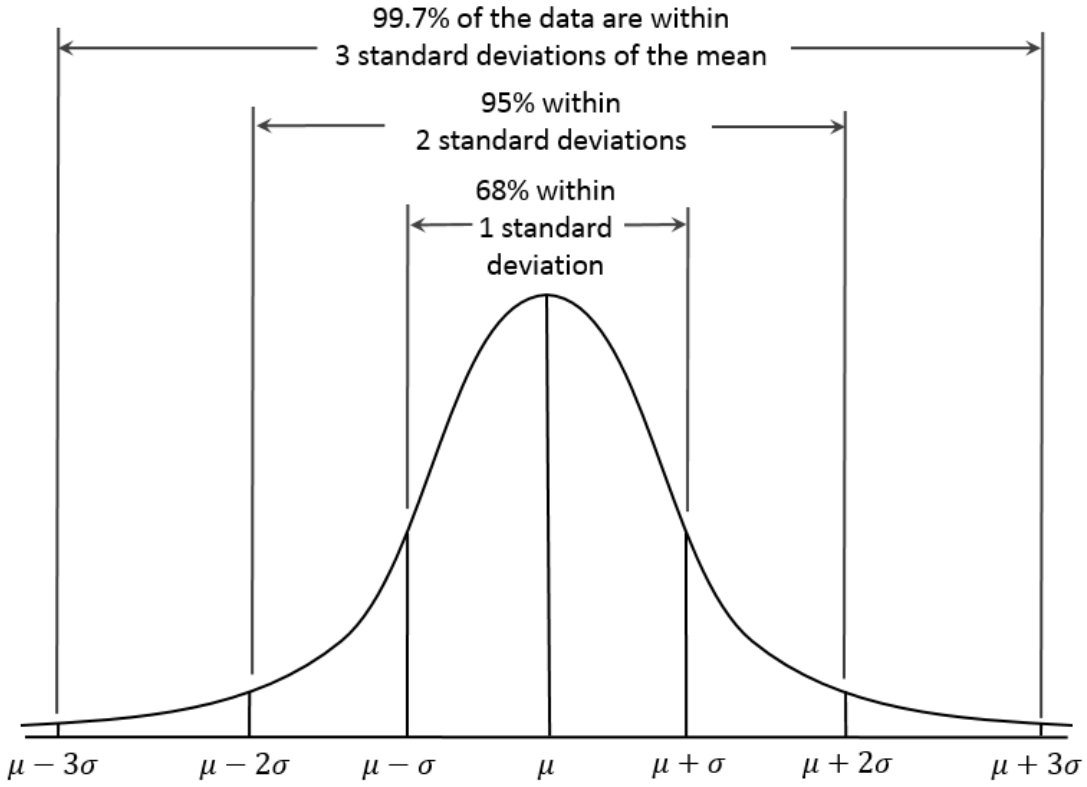
Histogram vs density distribution function

Histogram of data sample



Normal distribution function (mu, std)

Gaussian/Normal Distribution Function

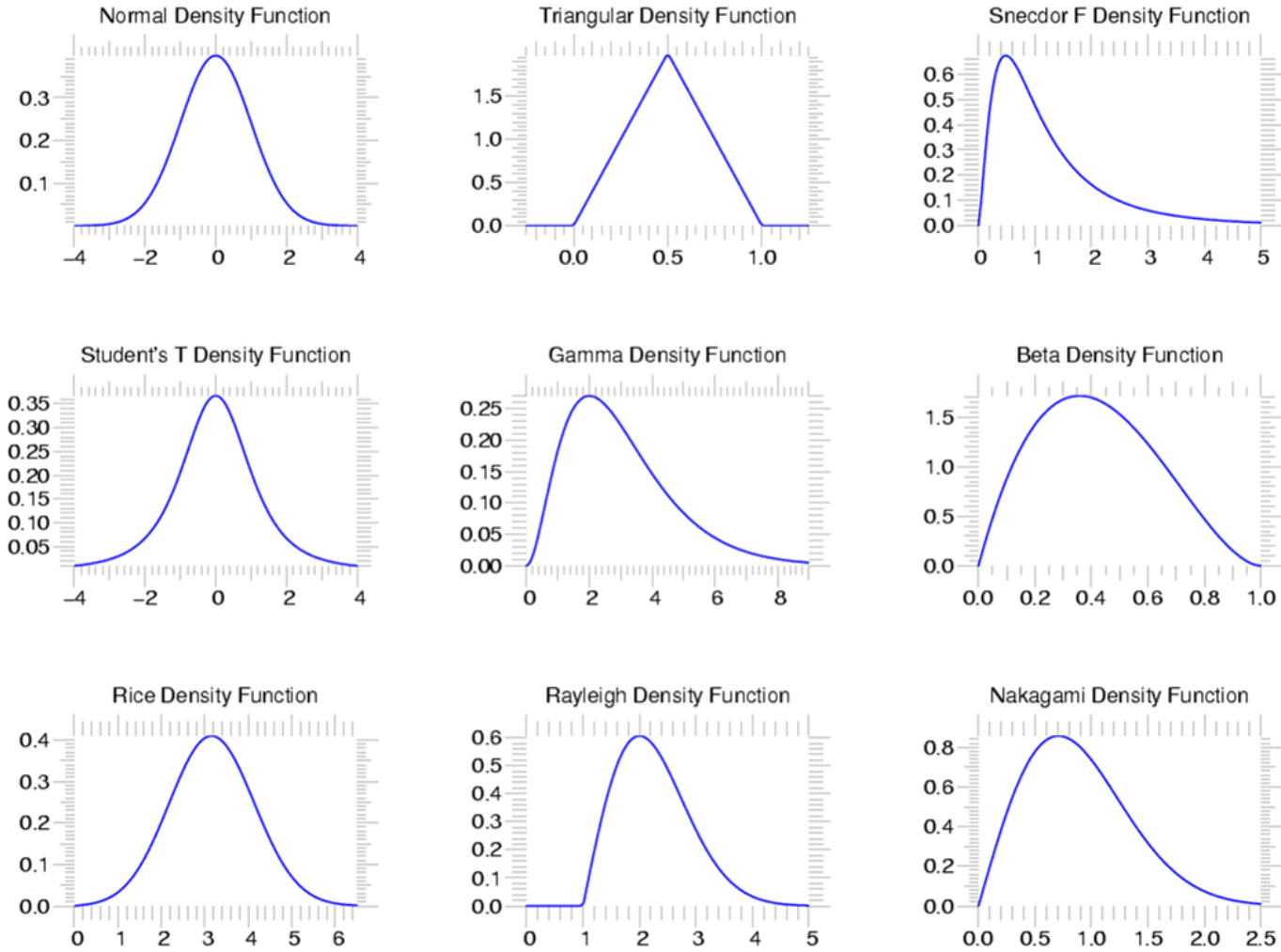


$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Statistics: scipy.stats

- Over 80 continuous distributions

```
pdf
cdf
Rvs
ppf
fit
var
Mean
std
```



Statistics: scipy.stats

- 10 discrete distributions

  `pdf`

  `cdf`

  `Rvs`

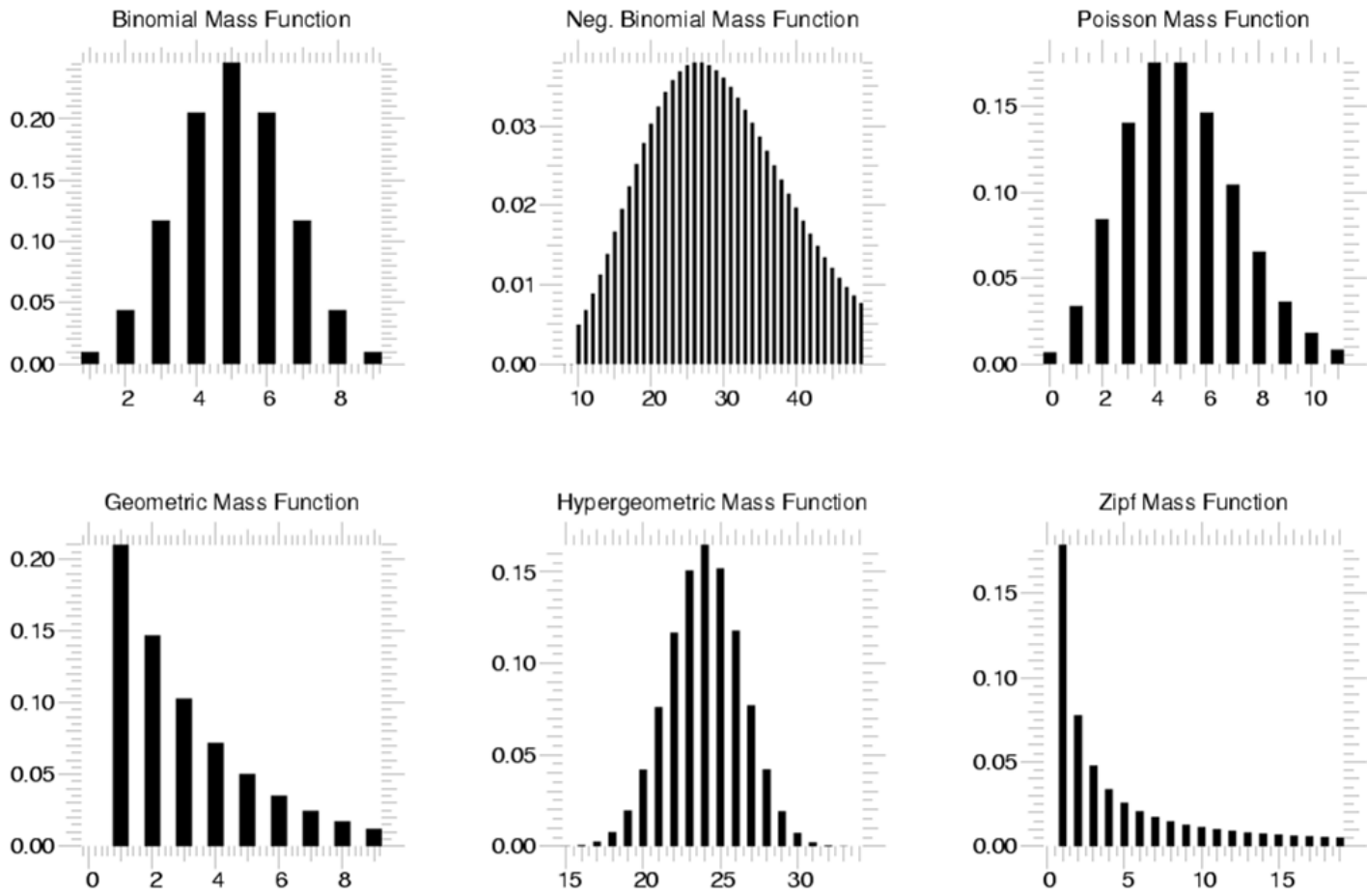
  `ppf`

  `fit`

  `var`

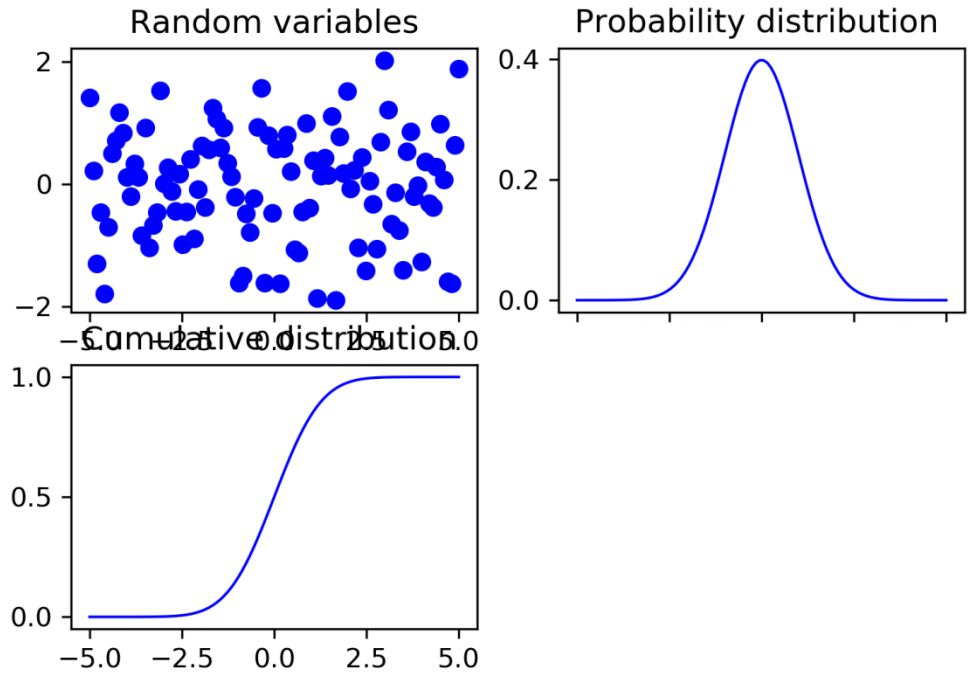
  `Mean`

  `std`



Stats objects

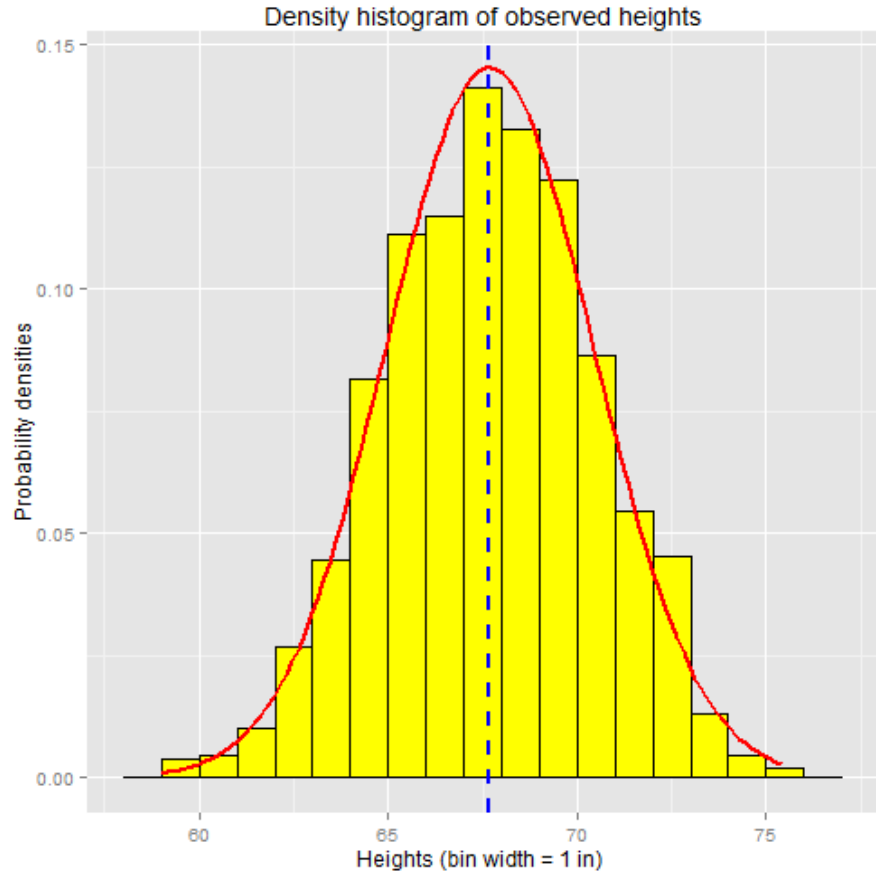
```
from scipy.stats import norm

samp = norm.rvs(size=100)

x = np.linspace(-5, 5, 100)

pdf = norm.pdf(x)

cdf = norm.cdf(x)
```



Stats objects

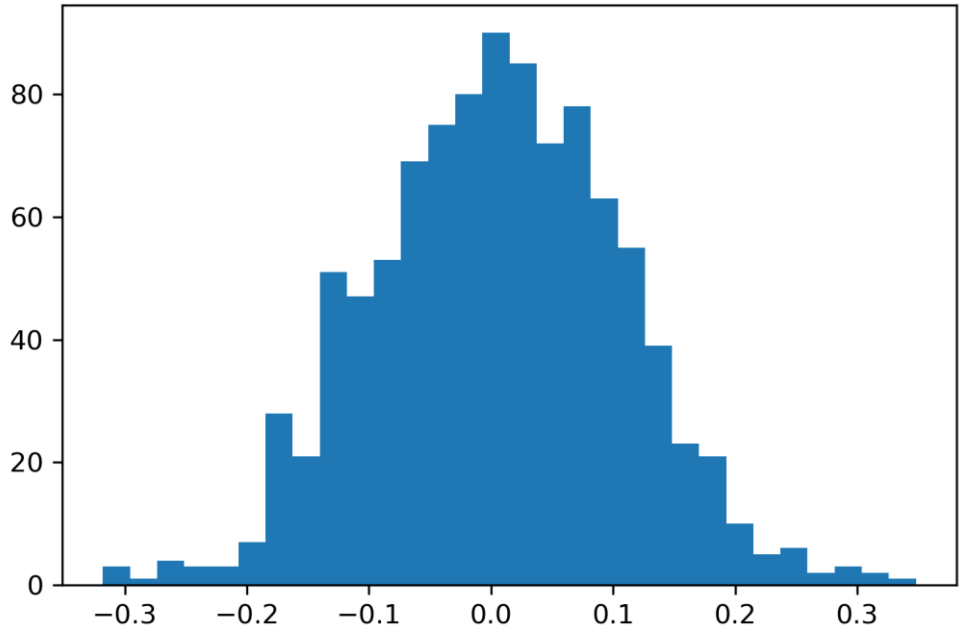
```
from scipy.stats import norm

samp = norm.rvs(size=100)

x = np.linspace(-5, 5, 100)

pdf = norm.pdf(x)

cdf = norm.cdf(x)
```

```
mu, sigma = norm.fit(samp)
-0.14, 1.01
```



Mean of sample means

Sample a 100 random variable array 1000 times and plot the **means** as a histogram

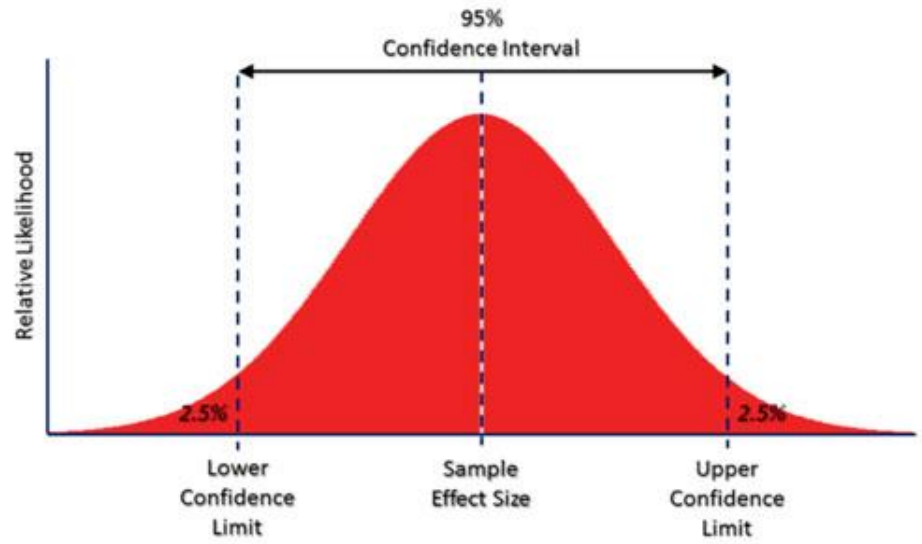


Confidence interval

np.percentile(sample, [2.5, 97.5])

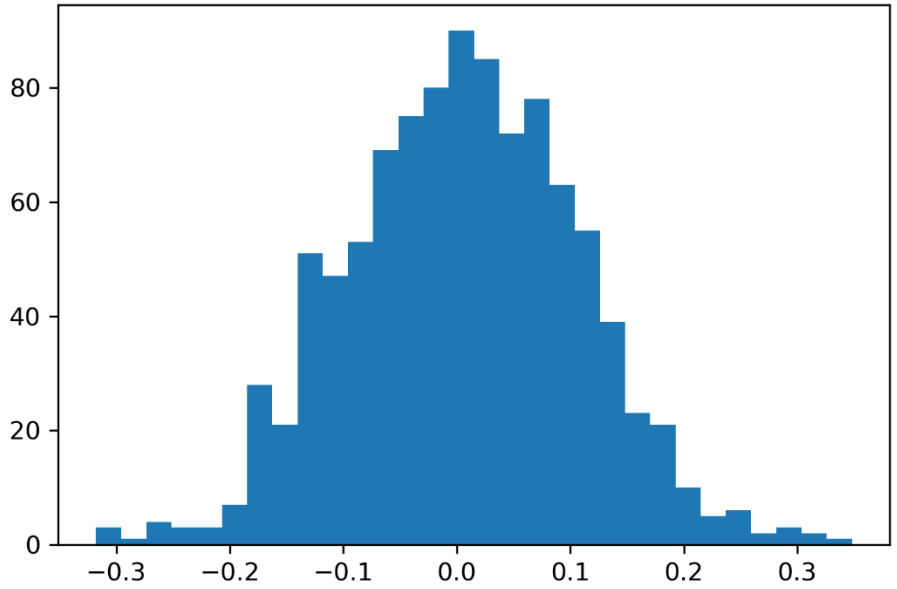
array([-0.0496105 , 0.05035856])

Lower Confidence Limit Upper Confidence Limit

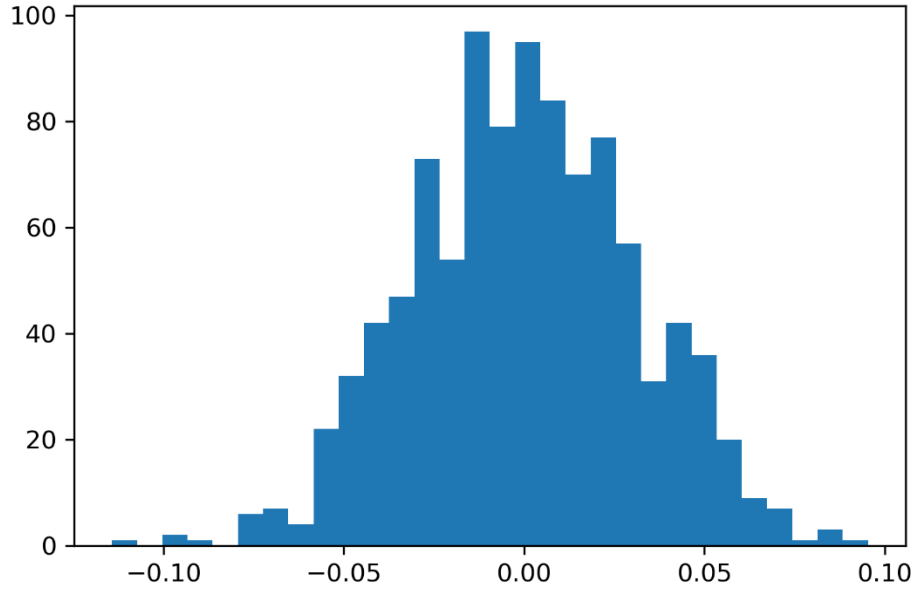


The sample mean std is the standard error

Sample a 1000 random variable array 1000 times and plot the means as a histogram: **standard deviation** ~0.03 (vs. ~0.1 for the 100 samples simulation)



100 samples

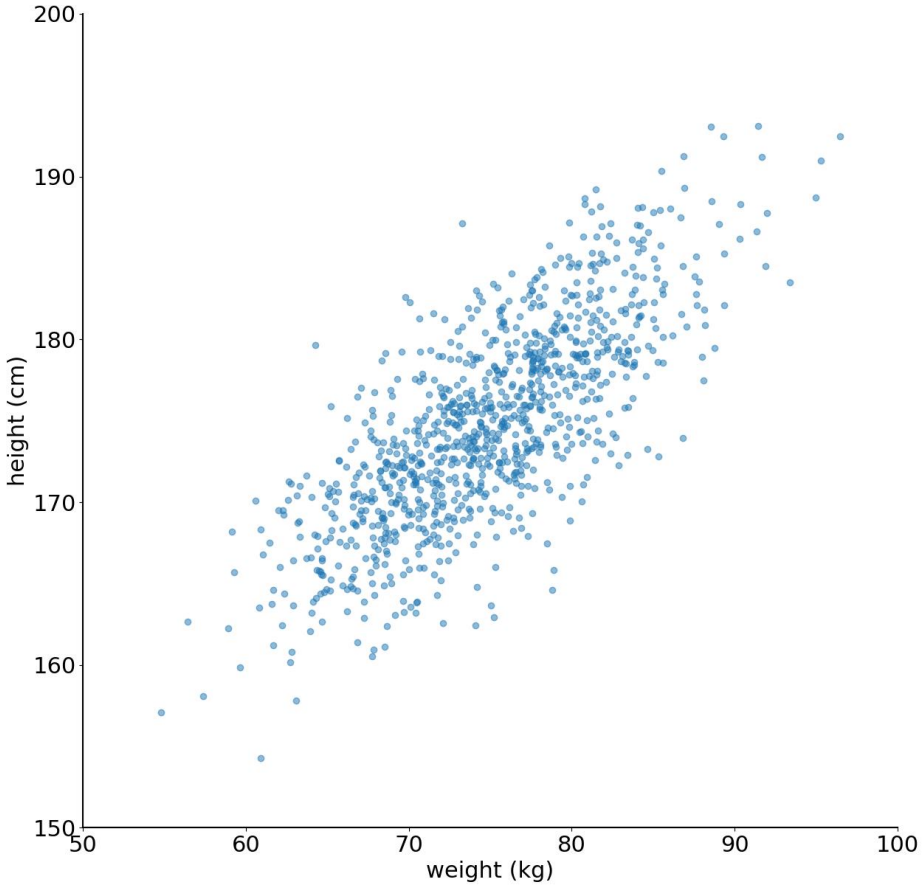


1000 samples

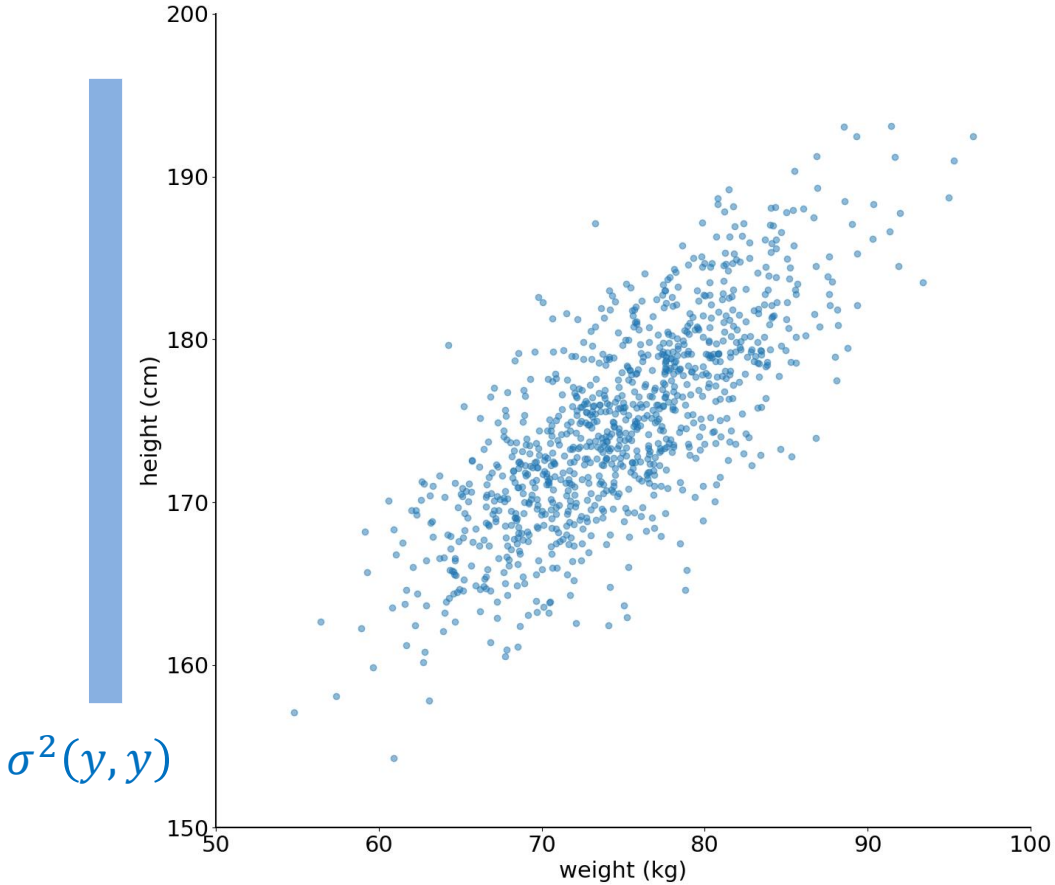
Statistical inference

- **Effect size**: means, standard deviations, overlaps
- **Precision:** Confidence interval and/or the standard error

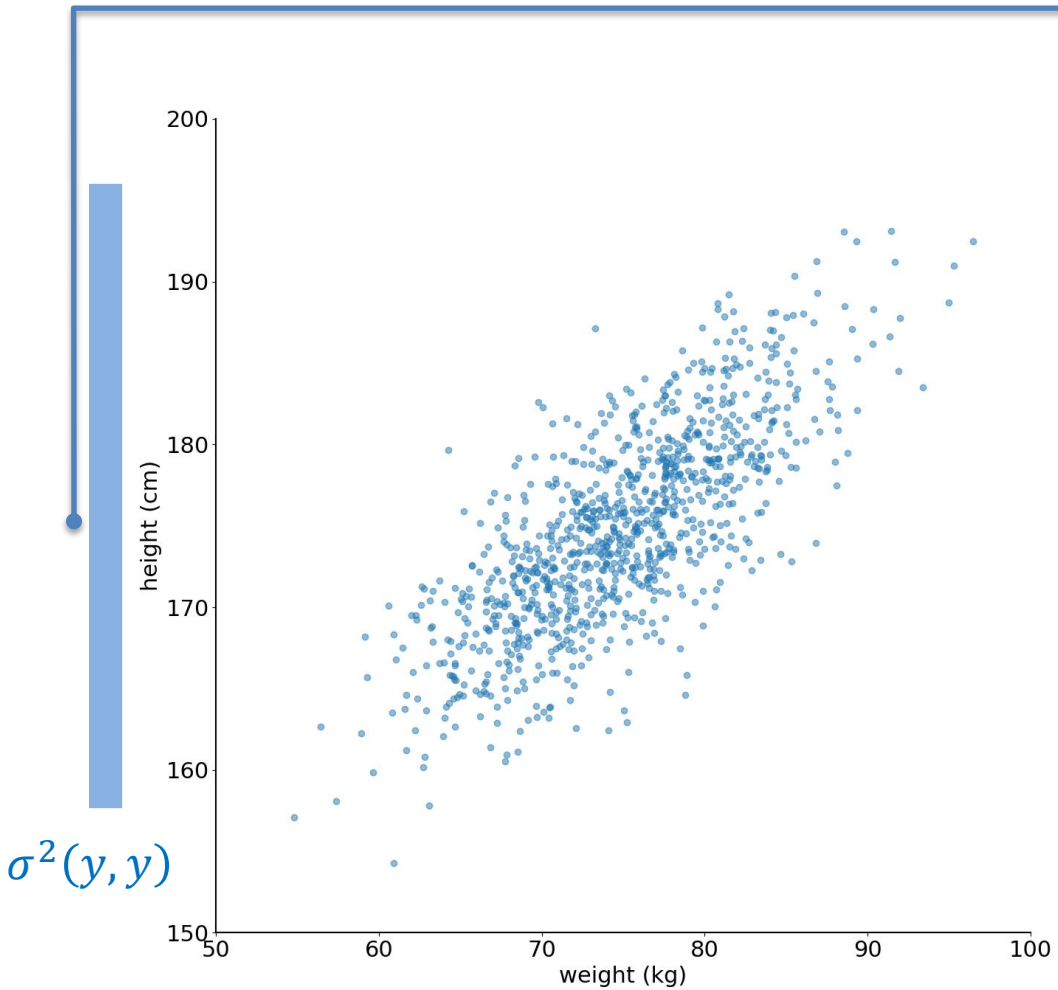
Multivariate data samples (covariance)?

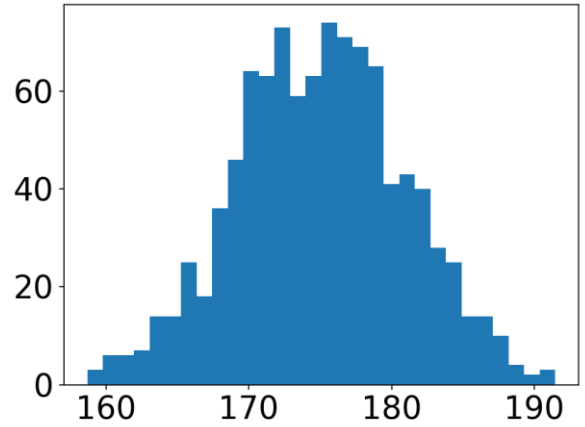


Multivariate data samples (covariance)?



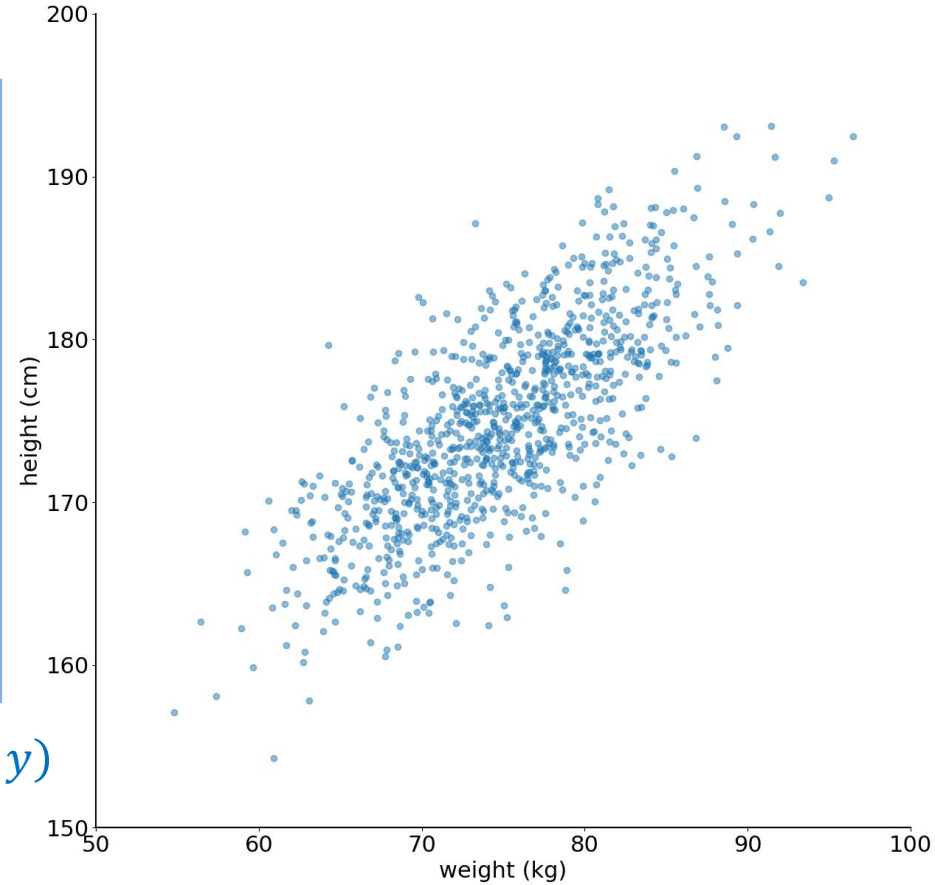
Multivariate data samples (covariance)?





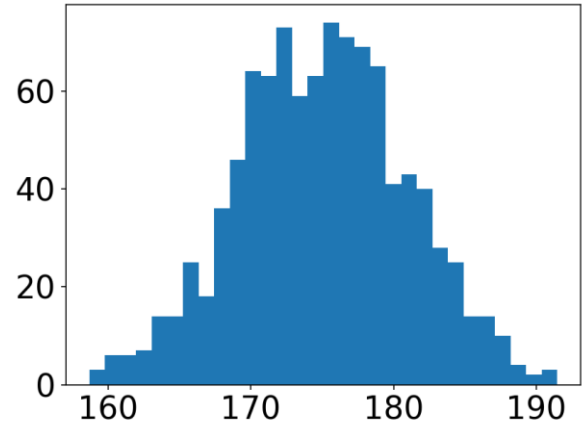
Variance in height = 36.9

Multivariate data samples (covariance)?

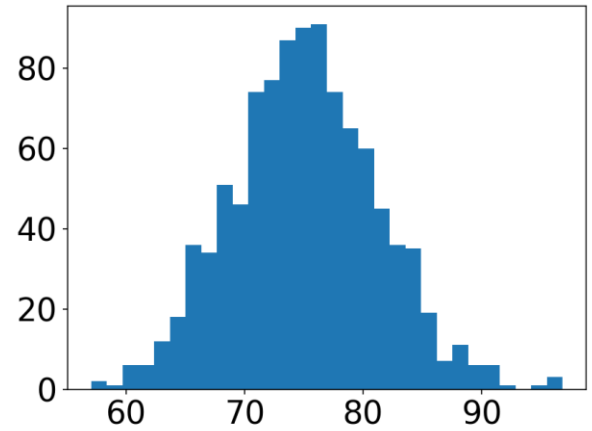


$\sigma^2(y, y)$

$\sigma^2(x, x)$

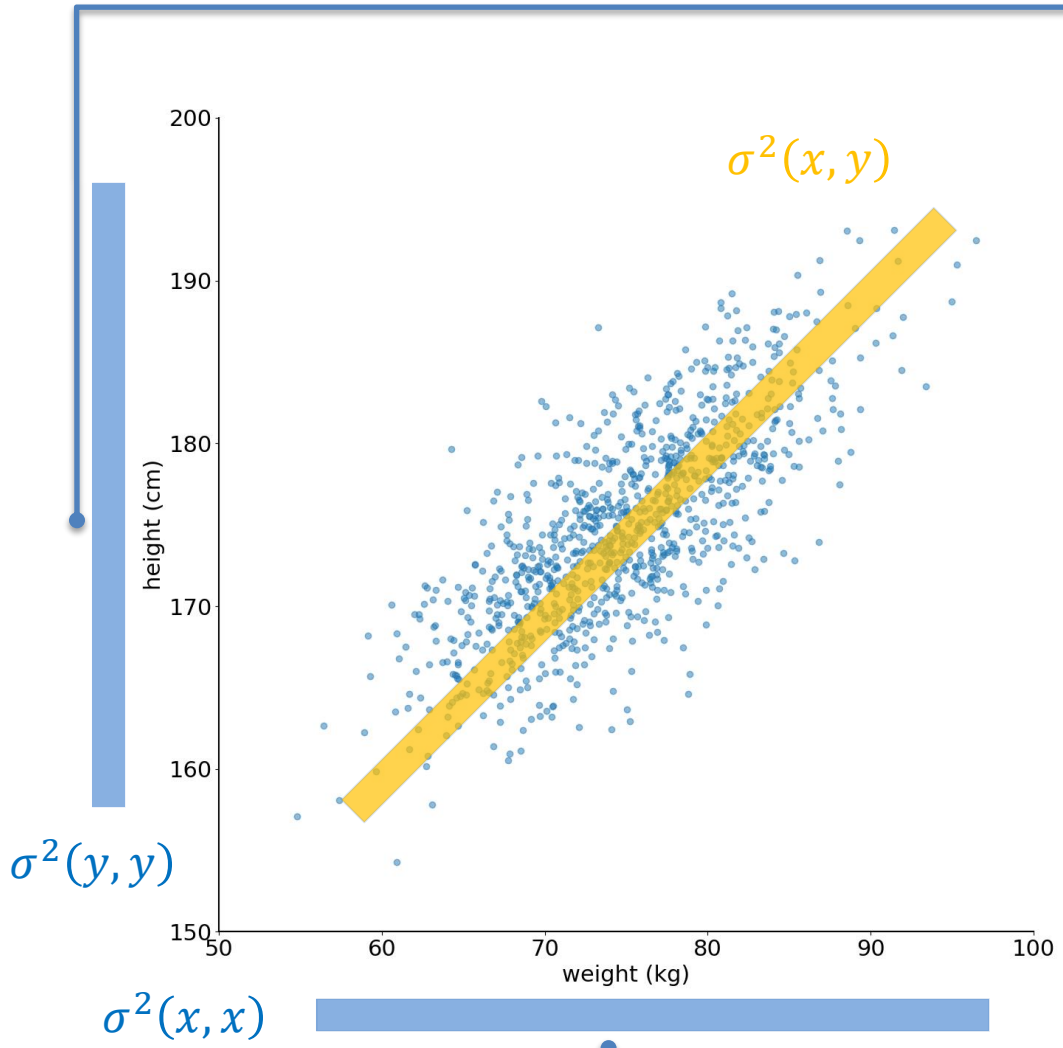


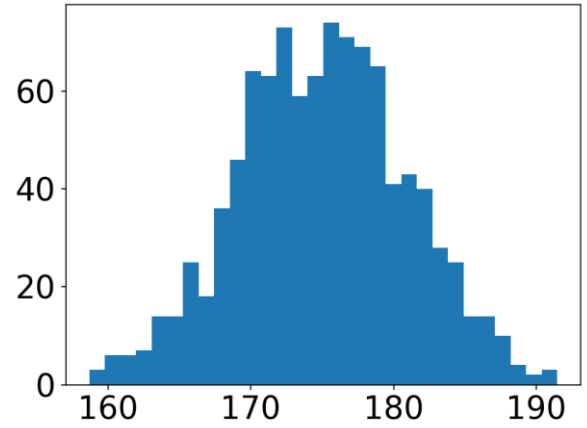
Variance in height = 36.9



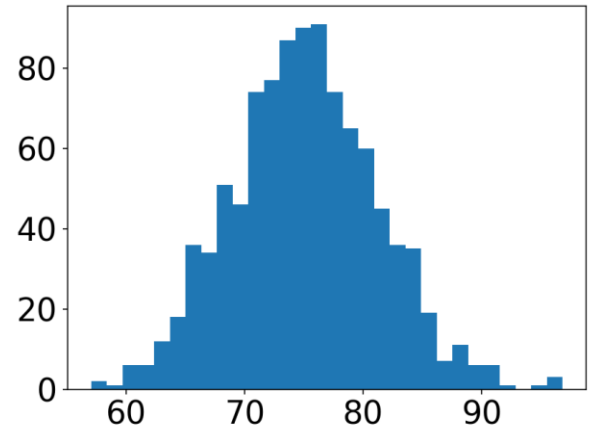
Variance in weight = 35.0

Multivariate data samples (covariance)?





Variance in height = 36.9



Variance in weight = 35.0

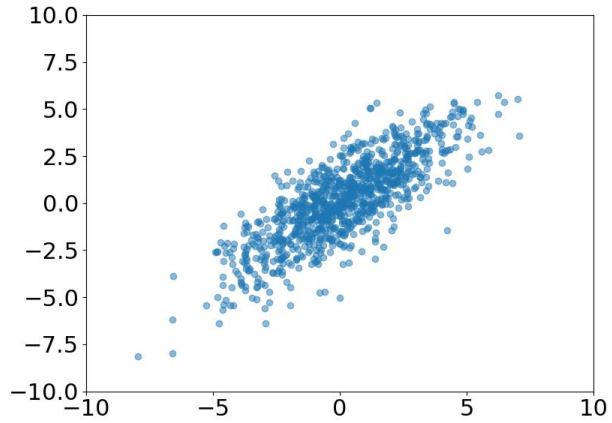
Covariance definition

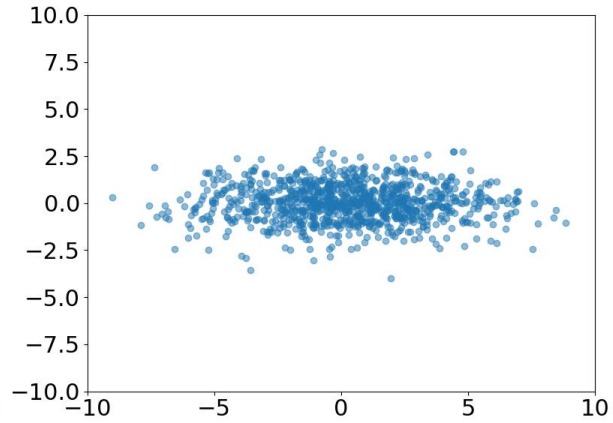
$$\sigma^2(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i \ - \mu_x)(y_i \ - \mu_y)$$

$$\Sigma = \begin{bmatrix} \sigma^2(x,x) & \sigma^2(x,y) \\ \sigma^2(x,y) & \sigma^2(y,y) \end{bmatrix}$$

Example Covariance Matrices

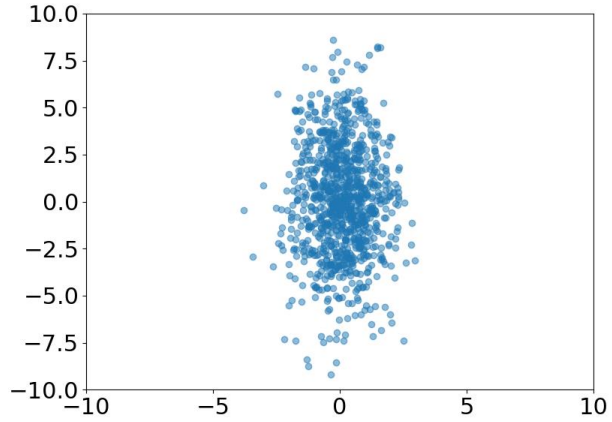
$$\Sigma = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

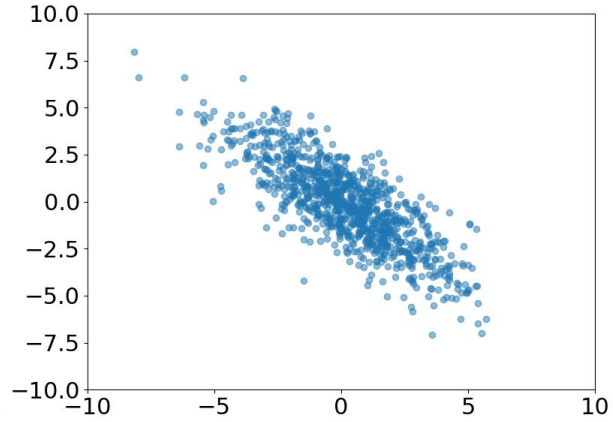




$$\Sigma = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

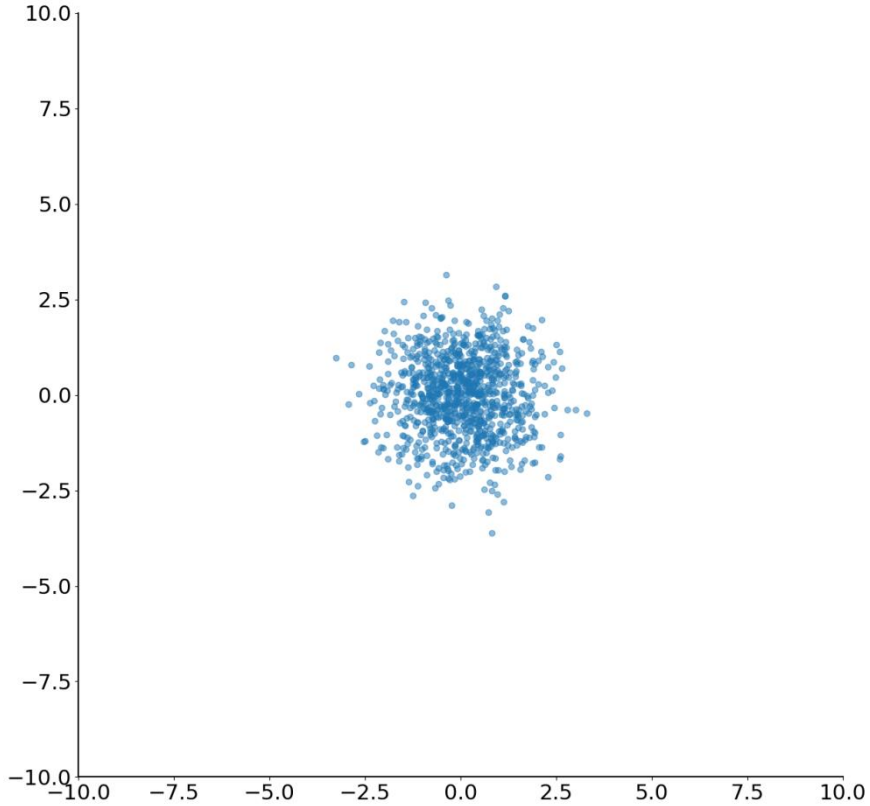
$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}$$





$$\Sigma = \begin{bmatrix} 5 & -4 \\ -4 & 5 \end{bmatrix}$$

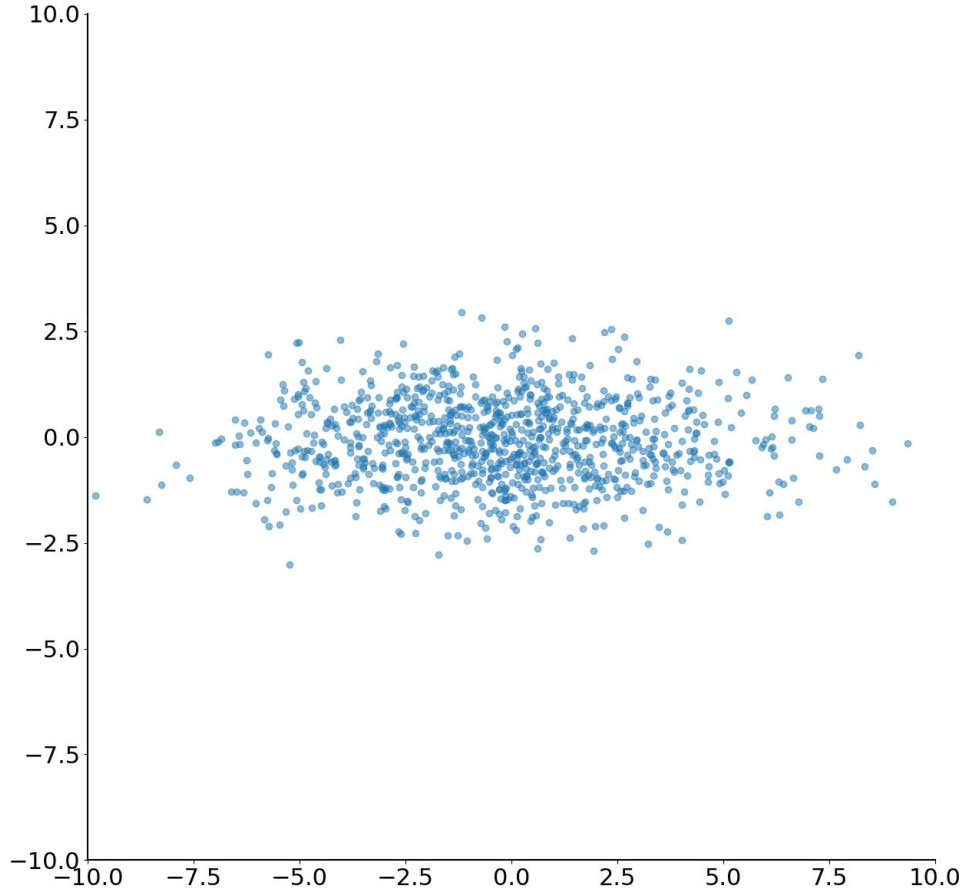
Normal distribution



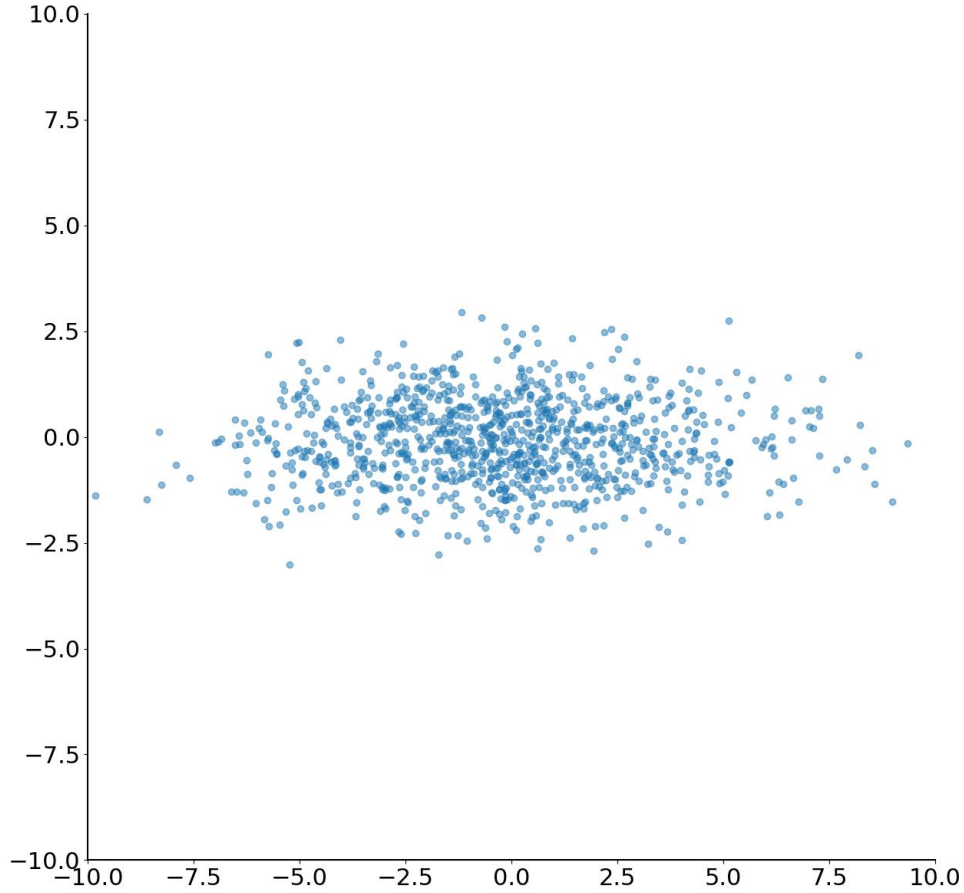
$$\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\sigma^2(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i \ - \mu_x)(y_i \ - \mu_y)$$

Scaling by 3 times in x-direction?

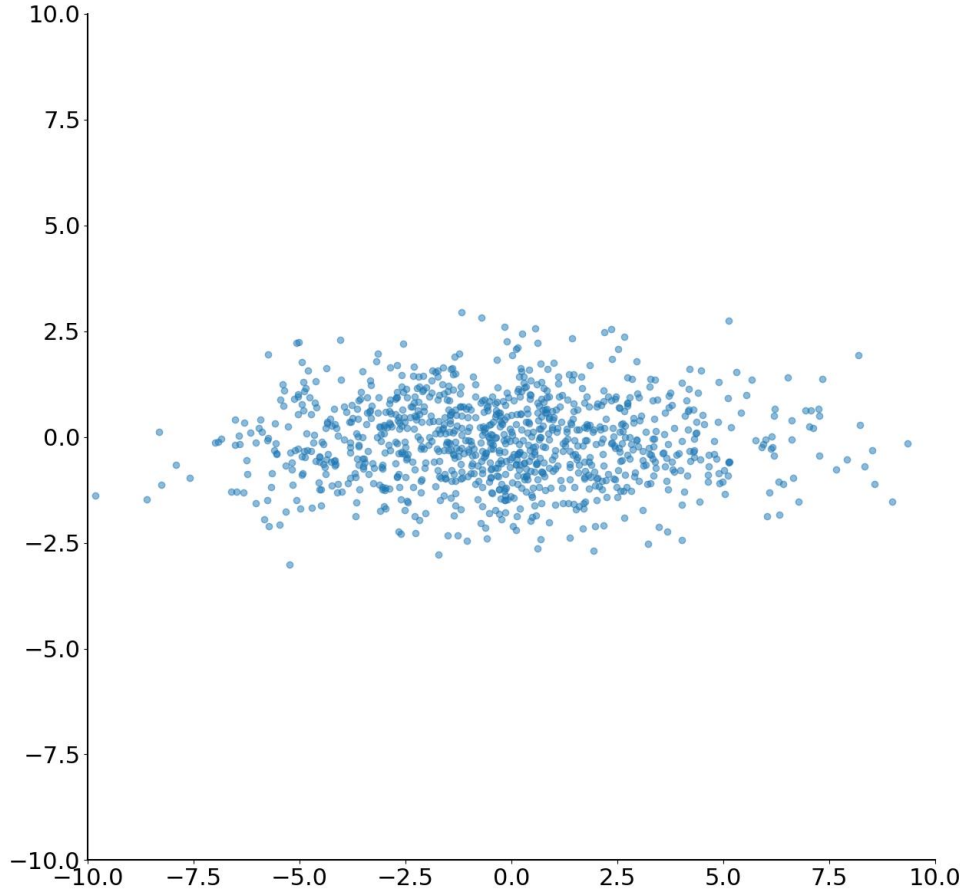


Scaling by 3 times in x-direction?



$$S = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

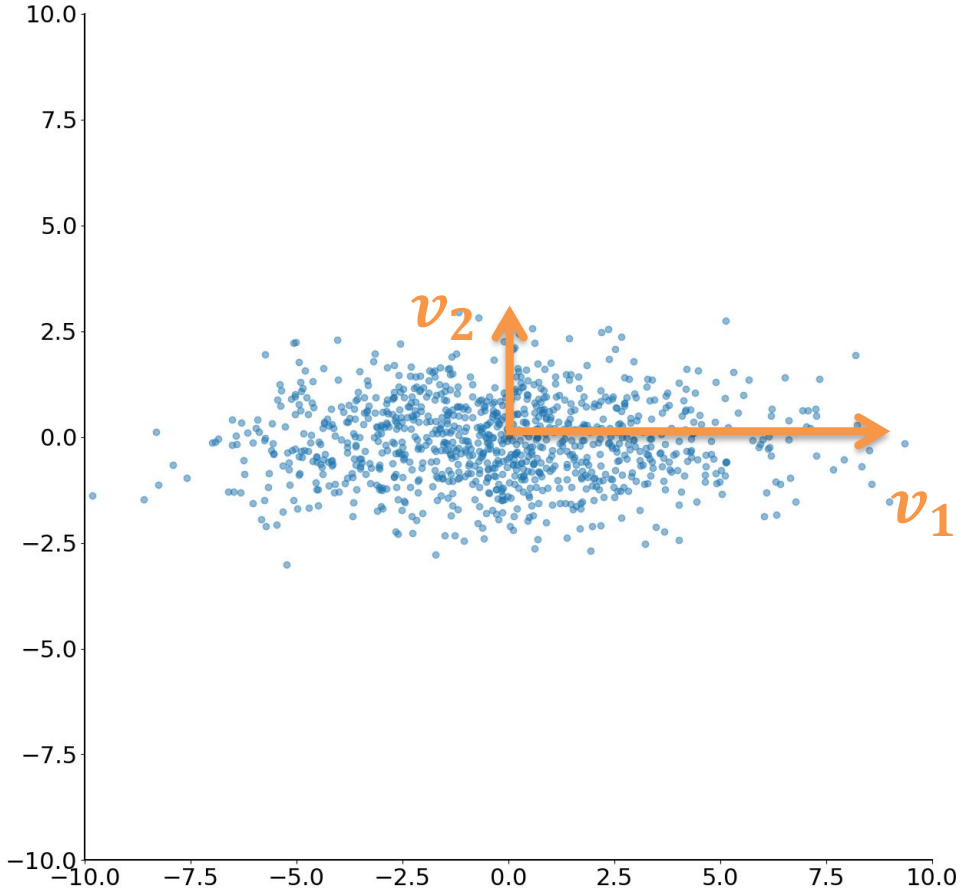
Scaling Transformation Matrix



$$S = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Eigenvectors

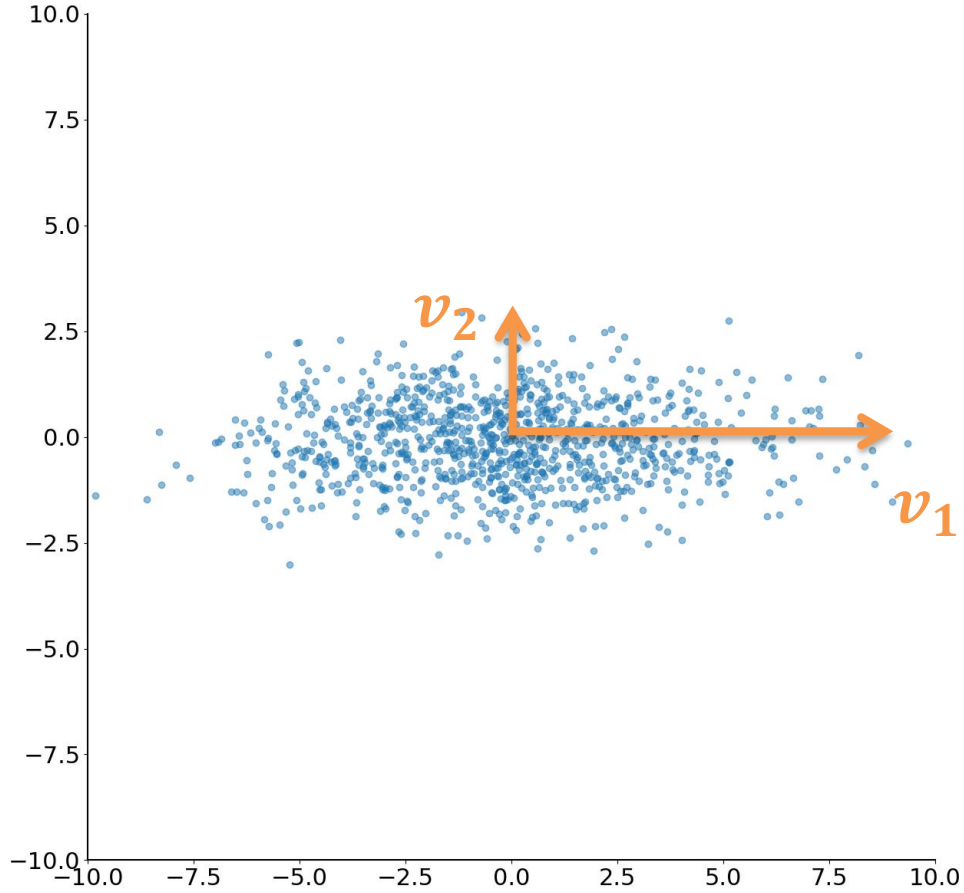


$$S = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Covariance and Data Transformation

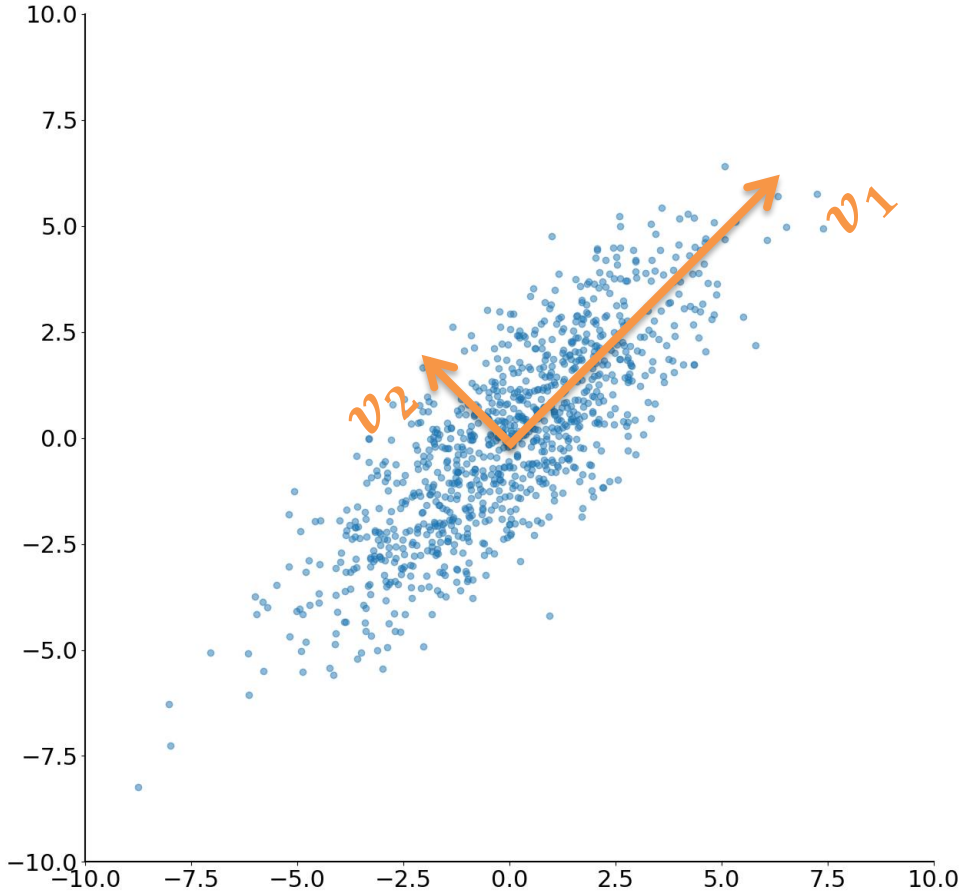
$$\Sigma = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix} = S^2$$



$$S = \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

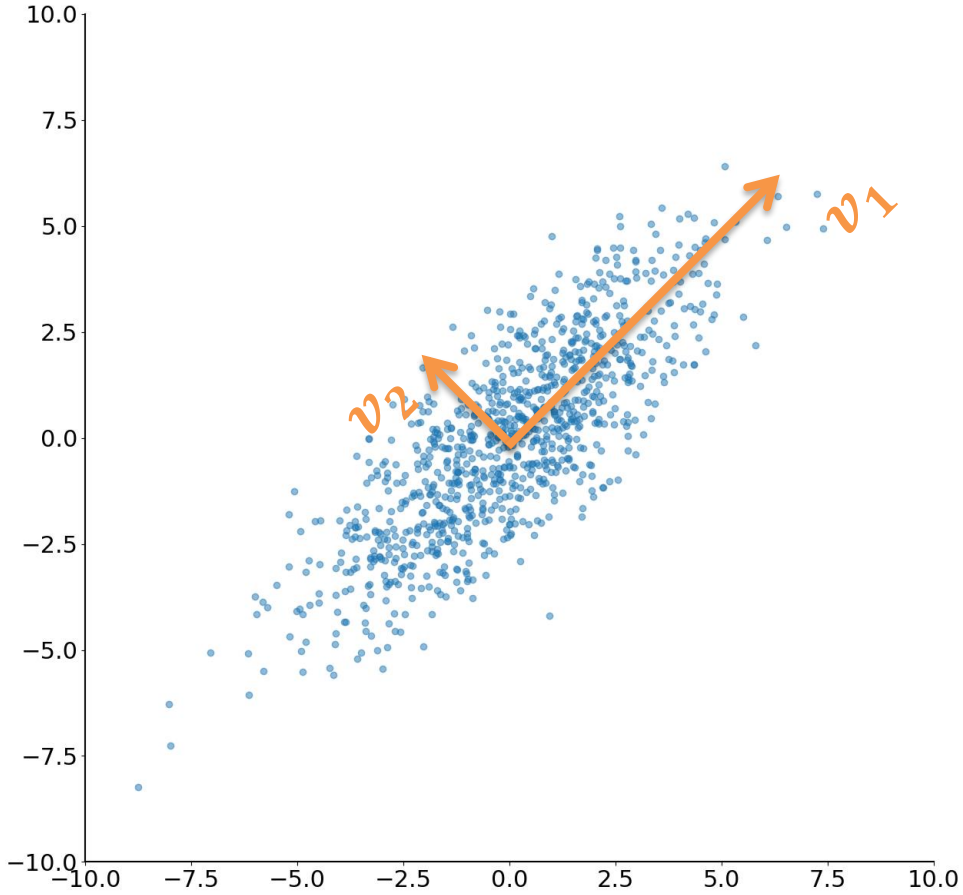
Rotation Transformation



$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta & \cos(\theta) \end{bmatrix}$$

Covariance as Linear Transformations

$$\Sigma v = \lambda v$$

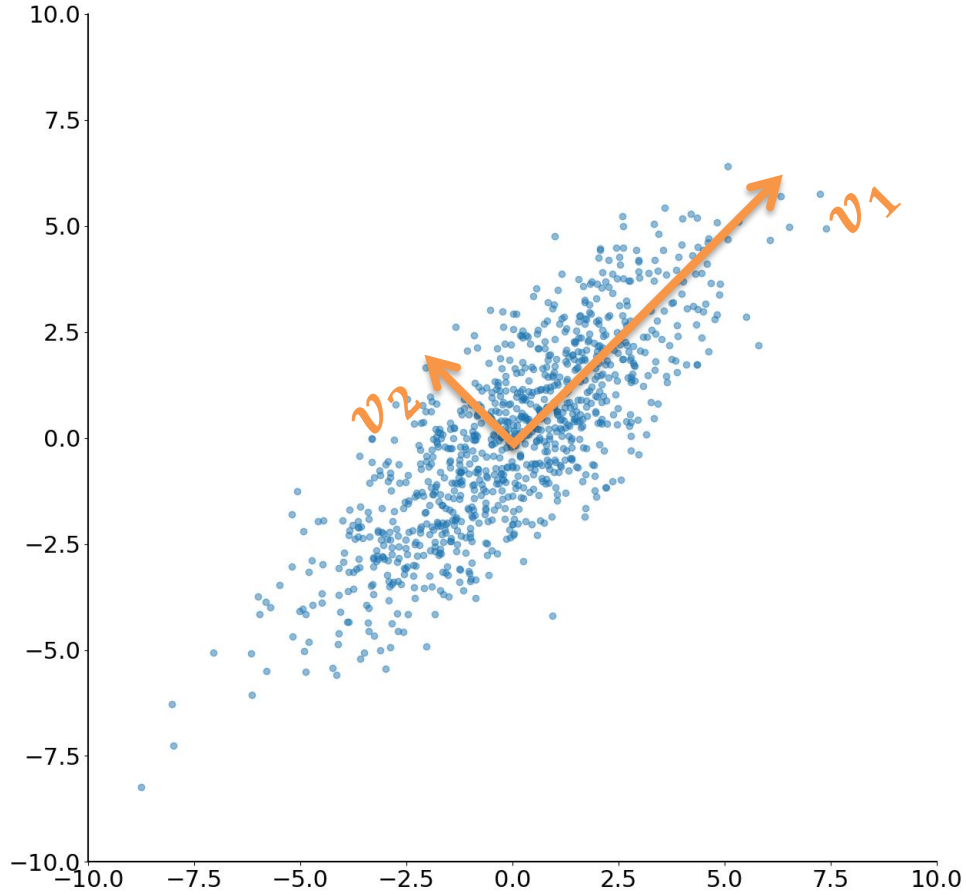


$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta & \cos(\theta) \end{bmatrix}$$

Covariance as Linear Transformations

$$\Sigma v = \lambda v$$

$$\Sigma V = LV$$



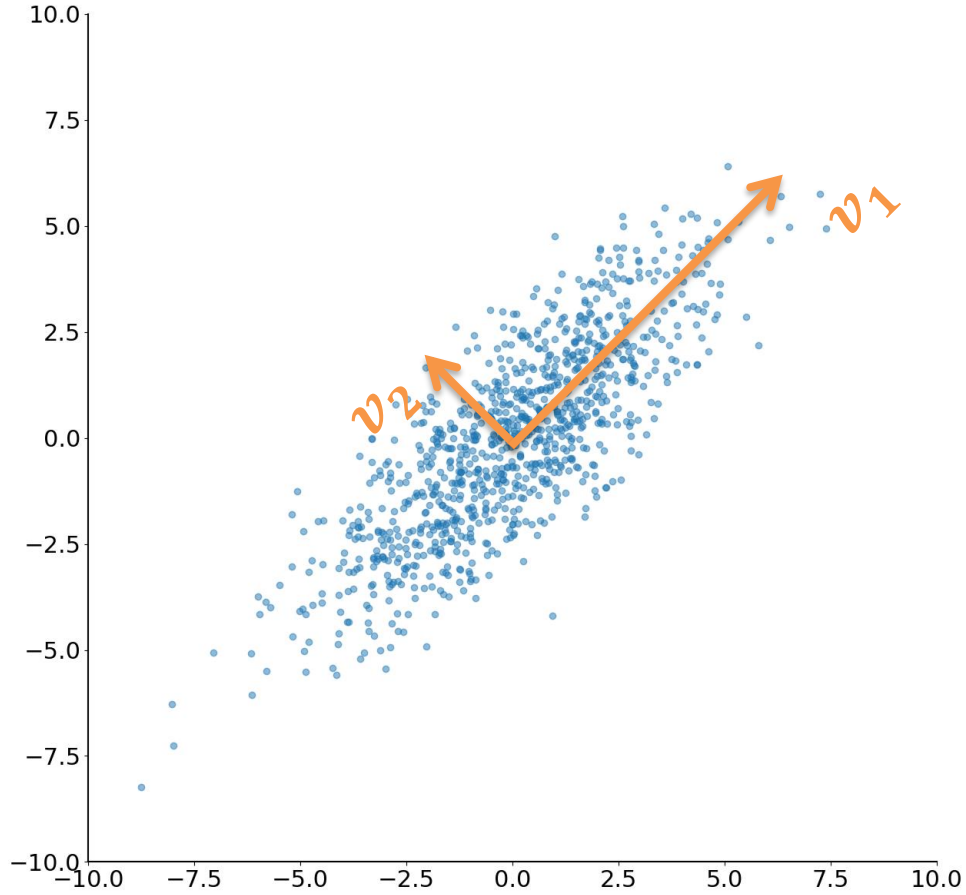
$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta & \cos(\theta) \end{bmatrix}$$

Covariance as Linear Transformations

$$\Sigma v = \lambda v$$

$$\Sigma V = LV$$

$$\Sigma = VLV^{-1}$$



$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta & \cos(\theta) \end{bmatrix}$$

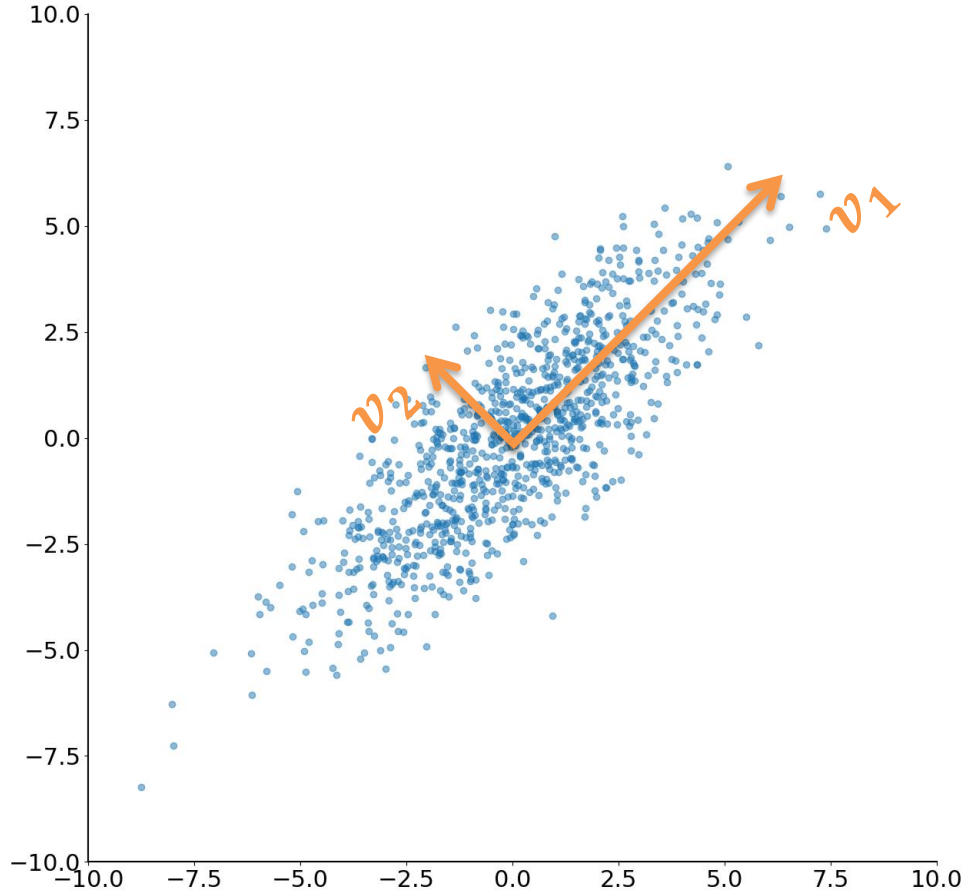
Covariance as Linear Transformations

$$\Sigma v = \lambda v$$

$$\Sigma V = LV$$

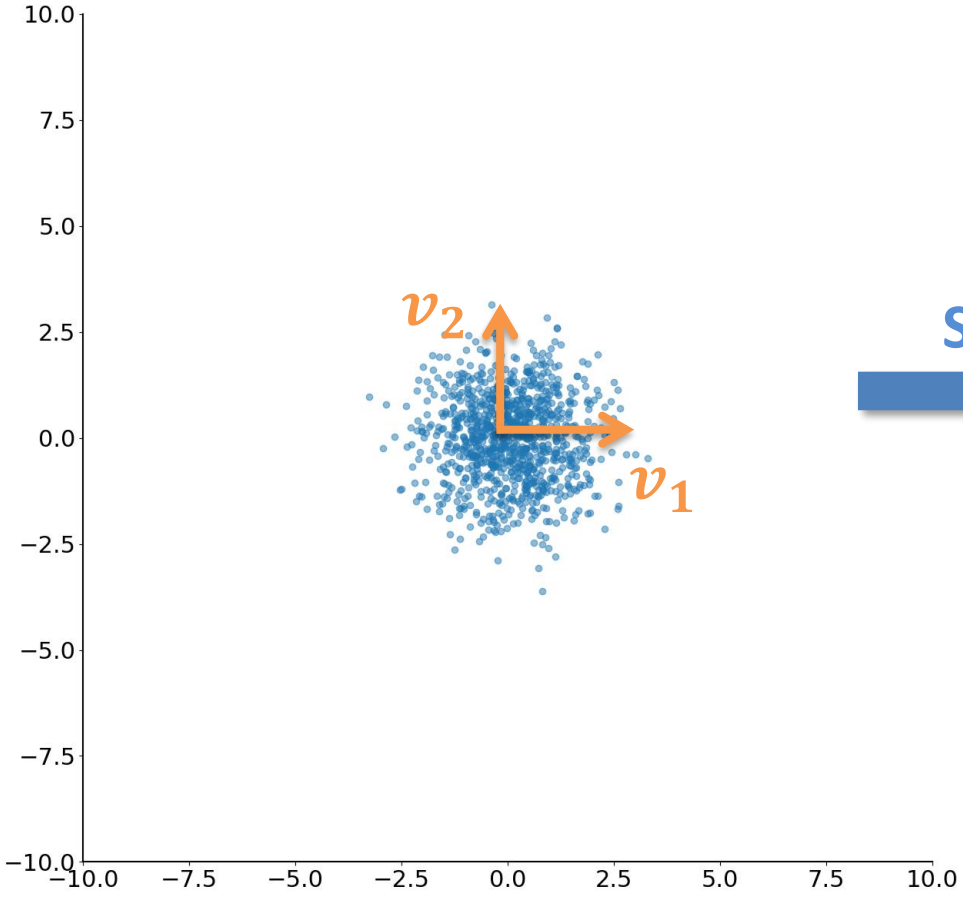
$$\Sigma = VLV^{-1}$$

$$\Sigma = RSSR^{-1}$$



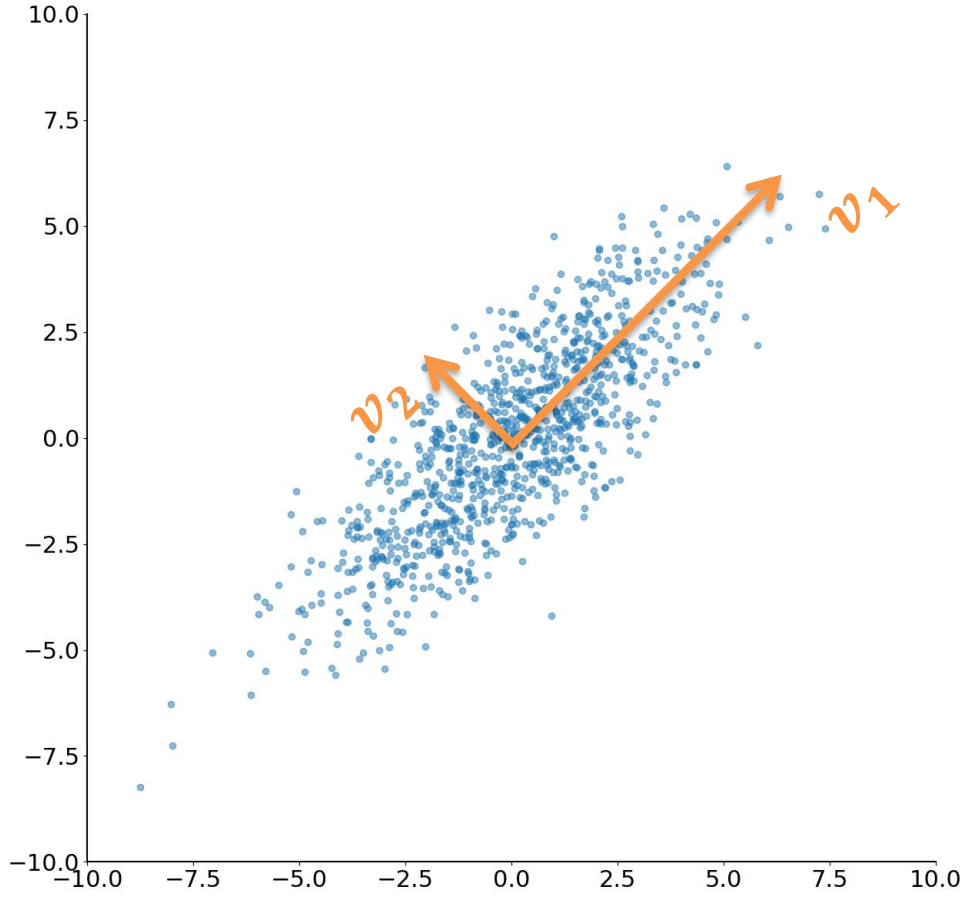
$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta & \cos(\theta) \end{bmatrix}$$

Covariance as Linear Transformations



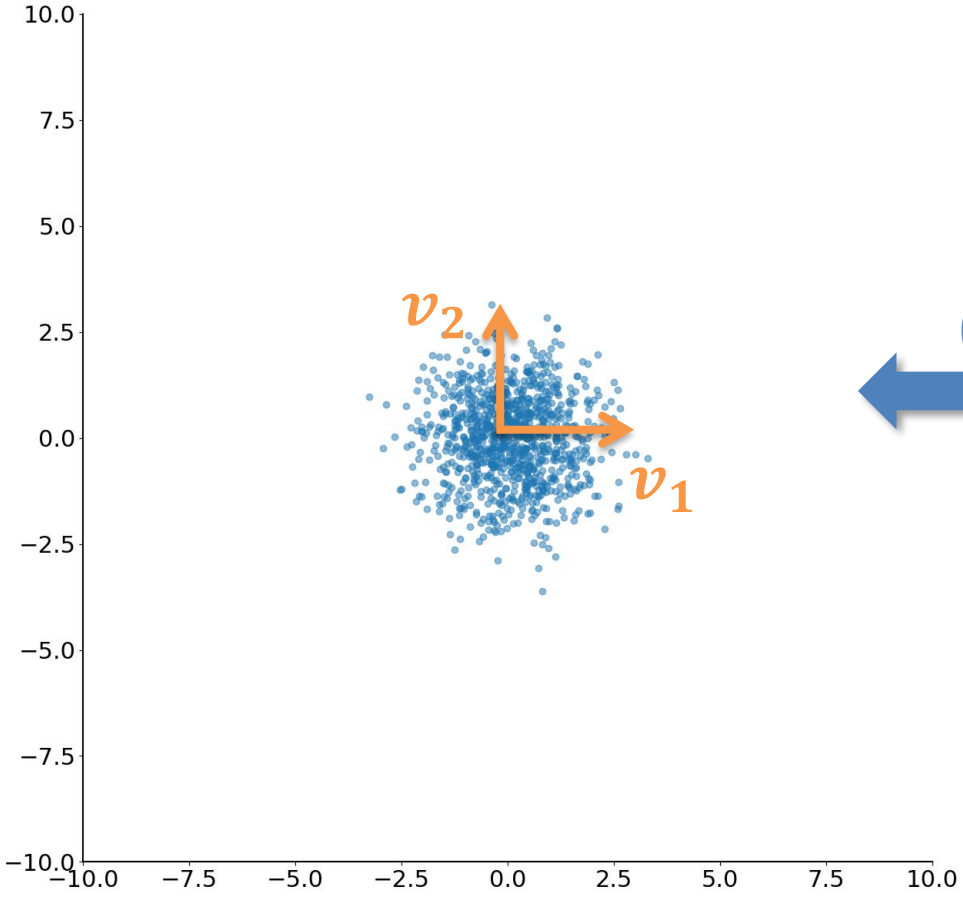
Σ

SR



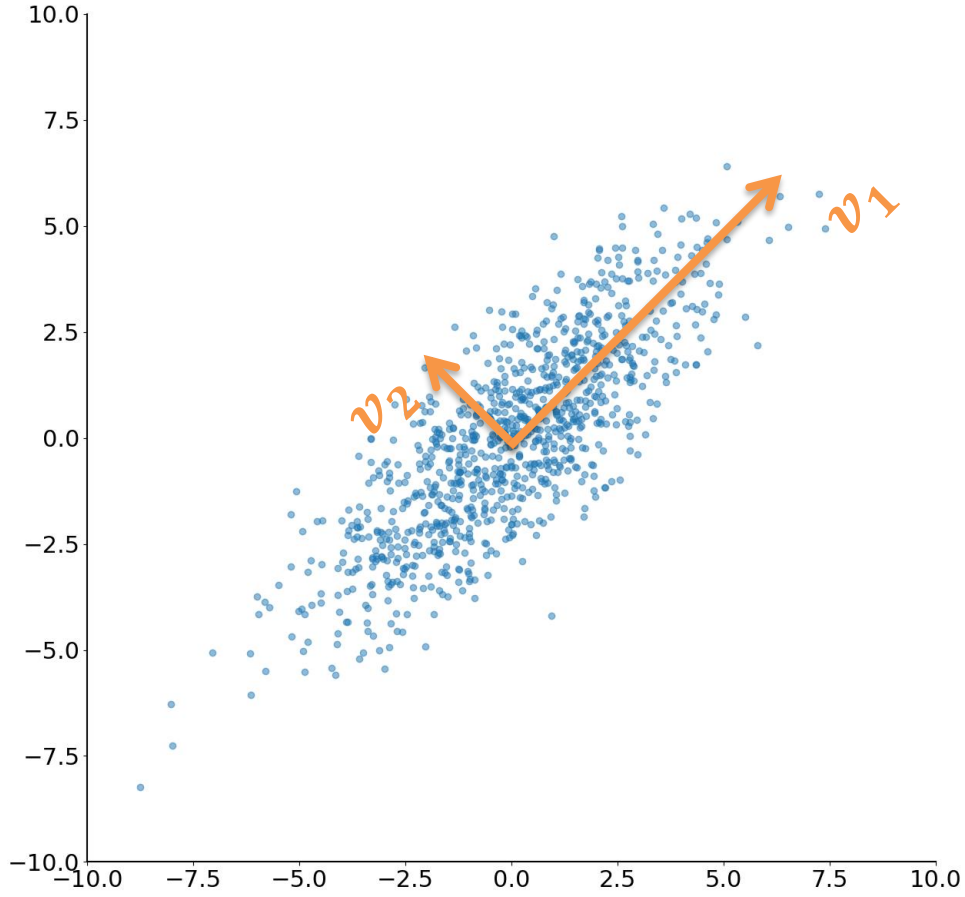
$\Sigma' = RSSR^{-1}$

Covariance as Linear Transformations



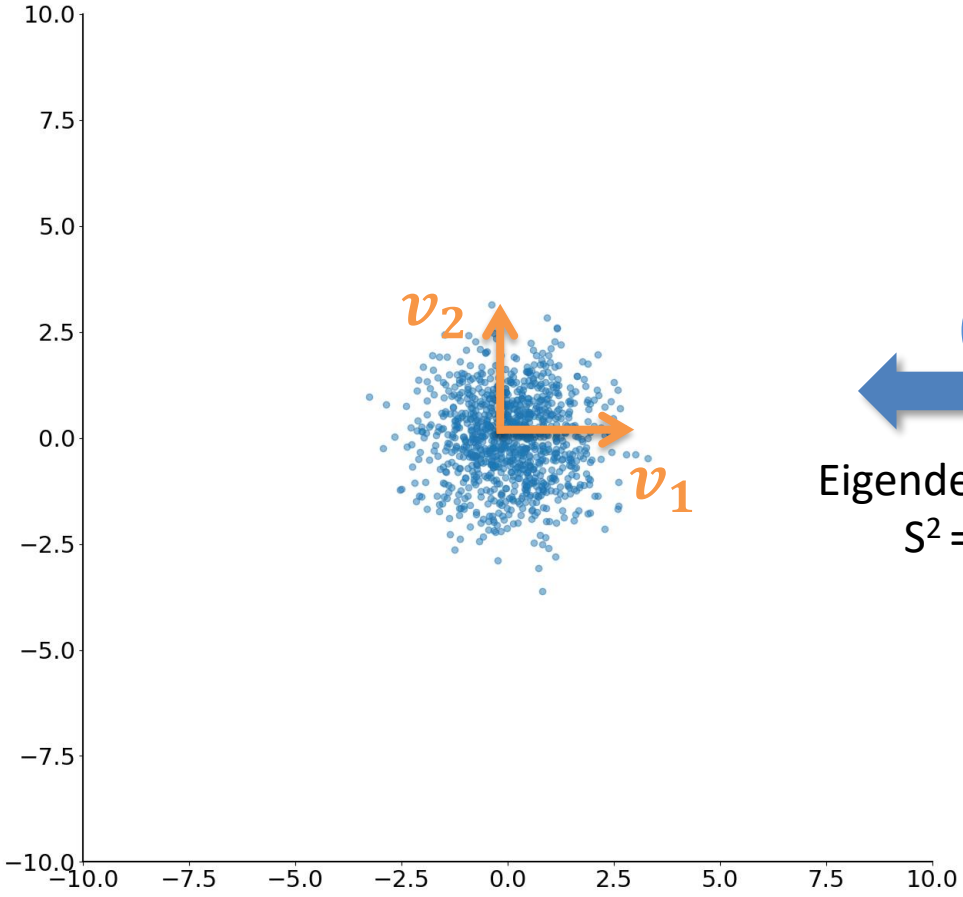
Σ

$(SR)^{-1}$



$\Sigma' = RSSR^{-1}$

Covariance as Linear Transformations

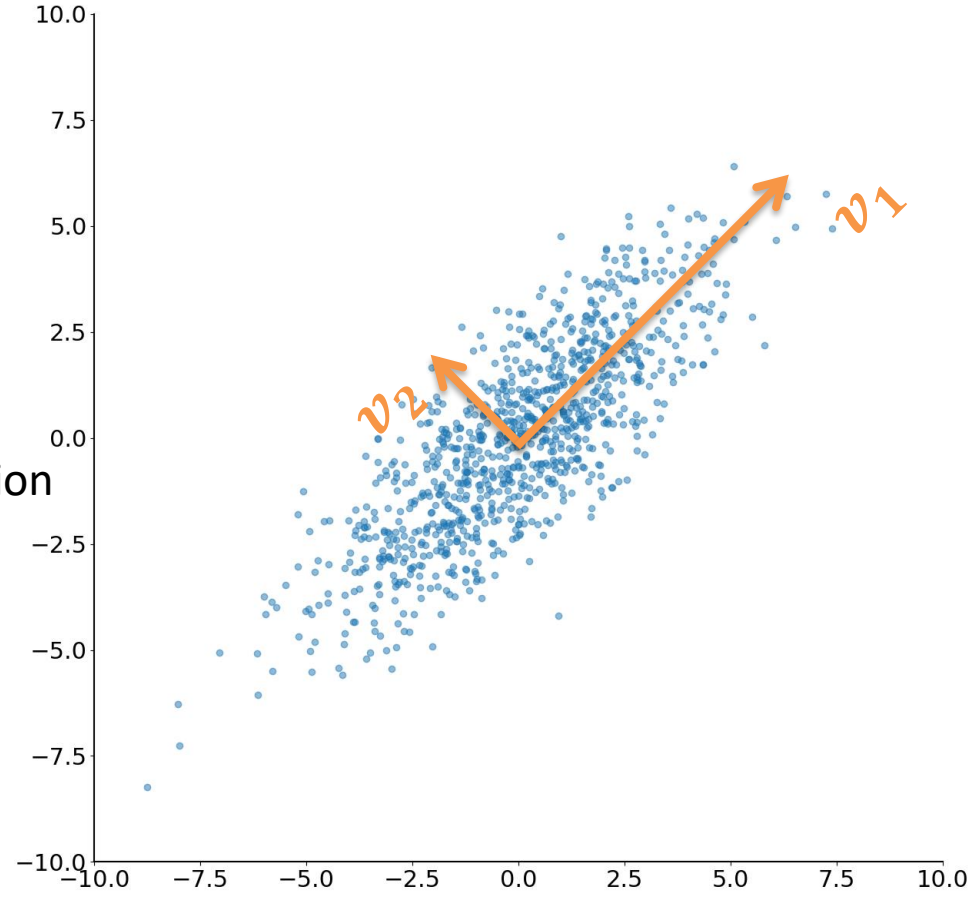


Σ

$(SR)^{-1}$

Eigendecomposition

$S^2 = L$, $R = V$



$\Sigma' = RSSR^{-1}$

numpy.corrcoef

numpy.**corrcoef**(*x*, *y=None*, *rowvar=1*, *bias=<class numpy._NoValue at 0x40a7274c>*, *ddof=<class numpy._NoValue at 0x40a7274c>*) [source]

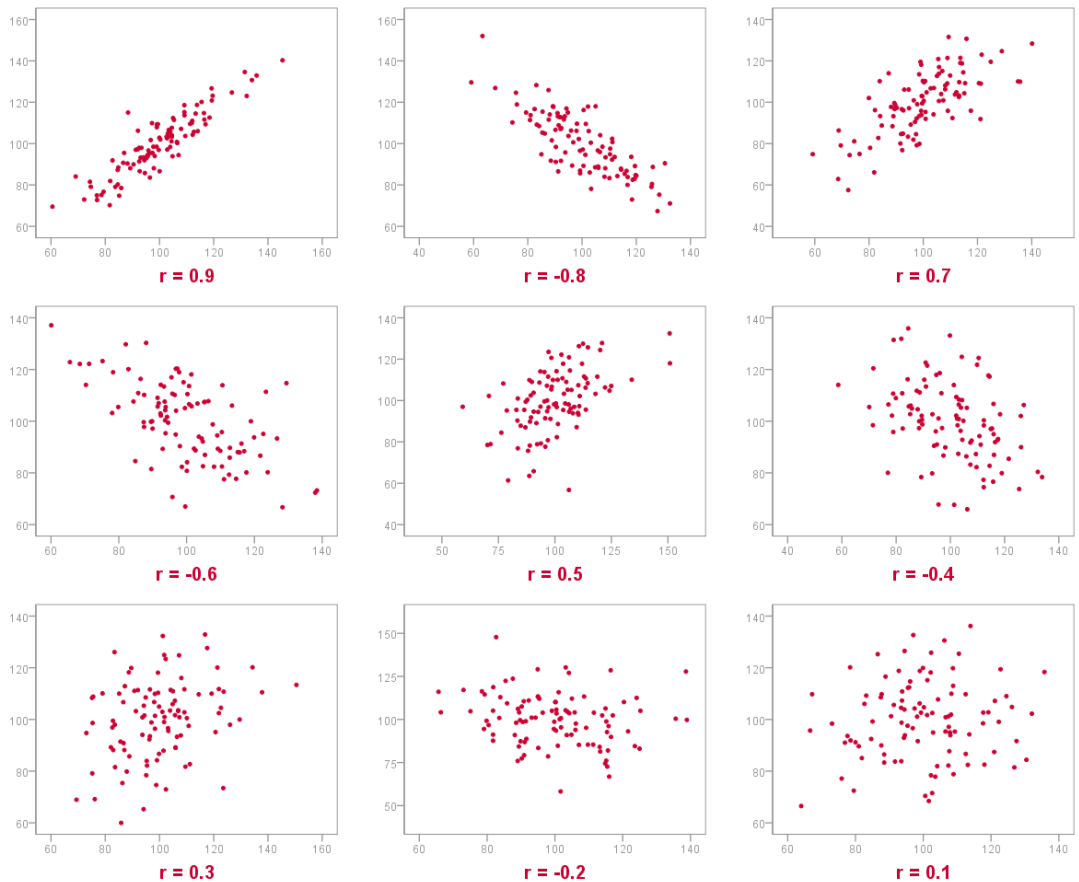
Return Pearson product-moment correlation coefficients.

Please refer to the documentation for cov for more detail. The relationship between the correlation coefficient matrix, *R*, and the covariance matrix, *C*, is

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}}$$

The values of *R* are between -1 and 1, inclusive.

r value	Strength
0.0 – 0.2	Weak correlation
0.3 – 0.6	Moderate correlation
0.7 – 1.0	Strong correlation



numpy.cov

numpy.cov(*m*, *y=None*, *rowvar=1*, *bias=0*, *ddof=None*, *fweights=None*, *aweights=None*) [source]

Estimate a covariance matrix, given data and weights.

Covariance indicates the level to which two variables vary together. If we examine N-dimensional samples, $X = [x_1, x_2, ...x_N]^T$, then the covariance matrix element C_{ij} is the covariance of x_i and x_j. The element C_{ii} is the variance of x_i.

```
>>> x
array([[0, 1, 2],
       [2, 1, 0]])

>>> np.cov(x)
array([[ 1., -1.],
       [-1.,  1.]])
```

numpy.linalg.eig

numpy.linalg.eig(*a*) [source]

Compute the eigenvalues and right eigenvectors of a square array.

Parameters: **a : *(..., M, M) array***

Matrices for which the eigenvalues and right eigenvectors will be computed

Returns: **w : *(..., M) array***

The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be of complex type, unless the imaginary part is zero in which case it will be cast to a real type. When *a* is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs

v : *(..., M, M) array*

The normalized (unit “length”) eigenvectors, such that the column `v[:,i]` is the eigenvector corresponding to the eigenvalue `w[i]`.

numpy.dot

numpy.dot(*a*, *b*, *out=None***)**

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using **matmul** or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to **multiply** and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where `M>=2`), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

Parameters:

a : *array_like*

First argument.

b : *array_like*

Second argument.

out : *ndarray, optional*

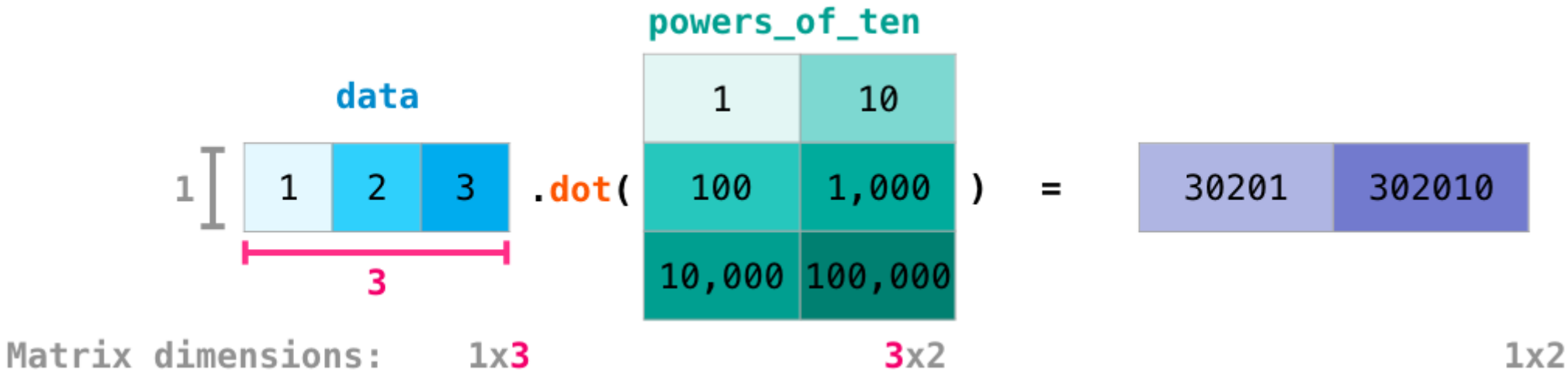
Output argument. This must have the exact kind that would be returned if it was not used. In particular, it must have the right type, must be C-contiguous, and its dtype must be the dtype that would be returned for *dot(a,b)*. This is a performance feature. Therefore, if these conditions are not met, an exception is raised, instead of attempting to be flexible.

Returns:

output : *ndarray*

Returns the dot product of *a* and *b*. If *a* and *b* are both scalars or both 1-D arrays then a scalar is returned; otherwise an array is returned. If *out* is given, then it is returned.

Dot Broadcasting



Exercise

- Generate 100 random points sample from a normal distribution (mu=3.0, sigma=1.0) and visualize them as a histogram.
- Generate 100 random points sample from a normal distribution (mu=1.0, sigma=2.0) and visualize them together with the previous histogram as a histogram.
- Calculate the misclassification rate for these two distributions.
- Fit a normal distribution function to the first data set.
- Generate 100 times 100 random points sample from a normal distribution (mu=3.0, sigma=1.0). Create a histogram of the means and calculate the standard error.
- Repeat the previous point 1000 times instead of 100. How does the standard error differ?

Exercise

- Create a bivariate distribution of 1000 points (normal with mu=1, sigma=1) and visualize it as a scatter plot. You should see a point cloud centered around the origin.
- Apply a transformation along the x-axis via a scaling matrix that stretches the point cloud of data by a factor 3 and visualize the points again.
- Rotate the point cloud by 45 degrees “up” using a rotation matrix.
- How can you apply both transformations (scaling and rotating) in a single transformation?
- Compute the correlation coefficient for the transformed data set.

Exercise

- Calculate the covariance matrix of your transformed bivariate data sample. Compute the eigendecomposition of the covariance matrix. Compute the inverse of the transformation matrix by multiplying eigenvalues with the matrix composed of both eigenvectors (don’t forget to take the square root of the eigenvalues) and transform your data set with it. Plot the transformed data as a scatter plot, you should see a similar point cloud to the original one again (without the stretch and rotation).