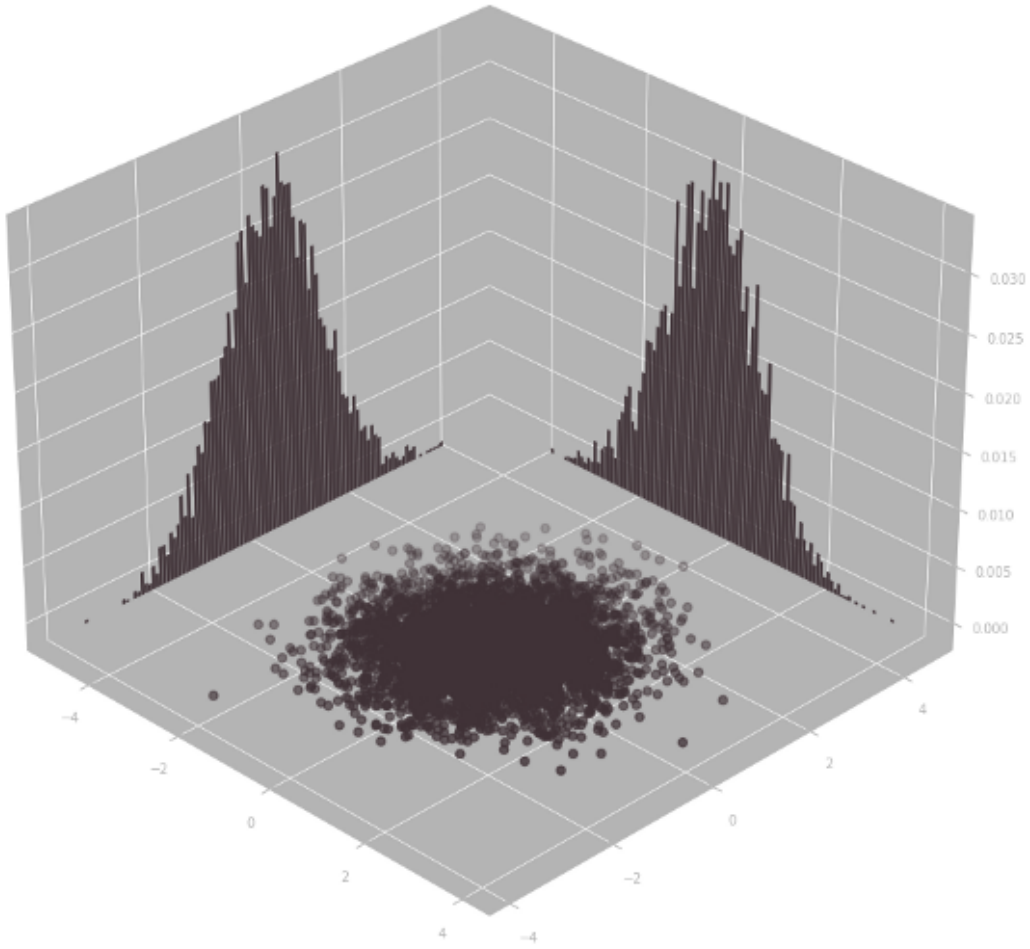
Published in Towards Data Science · Follow

Shuo Wang · Follow

Nov 1, 2020 · 6 min read ★

Multivariate Normal Distribution

What about multivariate normal distributions? Are they basically a few normally distributed variables bundled together? Let's take a look.



Normal distribution, also called gaussian distribution, is one of the most widely encountered distributions. One of the main reasons is that the normalized sum of independent random variables tends toward a normal distribution, regardless of the distribution of the individual variables (for example you can add a bunch of random samples that only takes on values -1 and 1, yet the sum itself actually becomes normally distributed as the number of sample you have becomes larger). This is known as the central limit theorem. But when you have several normal distributions, the situation becomes a little more complicated (don't worry, not that much more)




Normal Distribution

Let's start with a single normal distribution.

Suppose I have a random variable (say the amount of time it takes me to finish my lunch…), I sample it 10000 times (keeping record every day for 28 years…), what is the result going to look like?

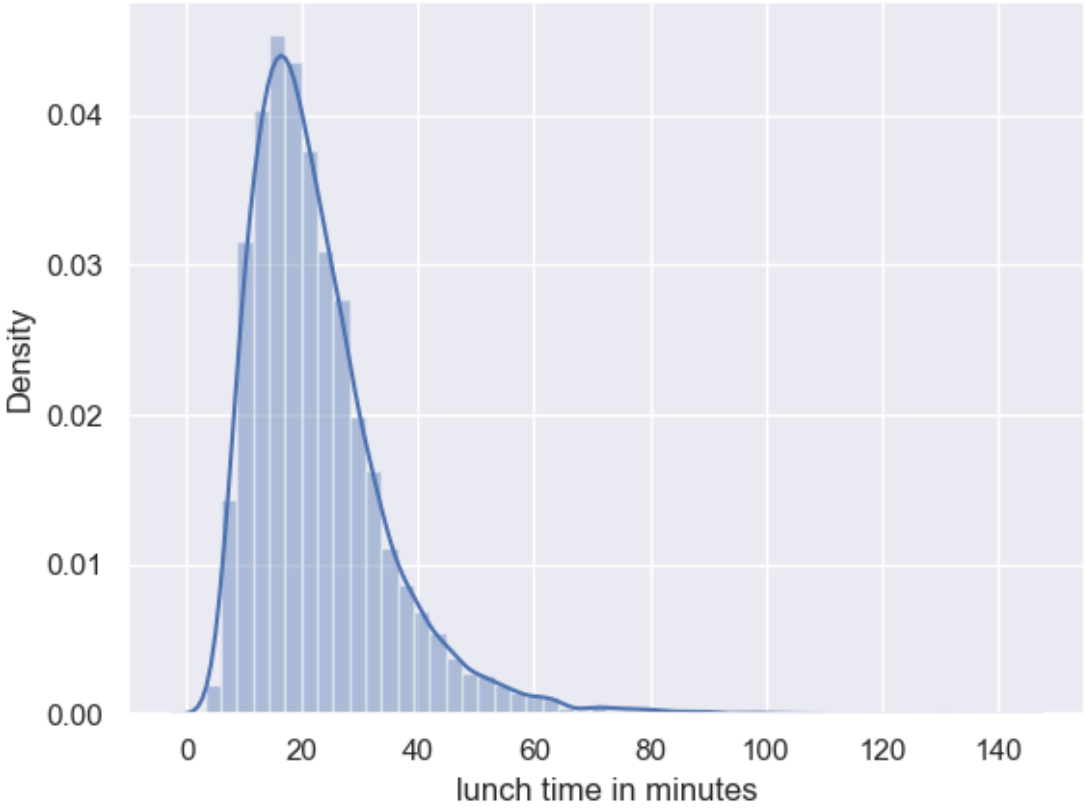


Fig 1

For me it would probably look something like the above. It's going to be higher than 0 minute, for obvious reasons, and it's going to peak around 20 minutes. Sometimes I take longer to finish when I don't have much to do and sometimes I might just eat at my desk really fast so I can get to work.

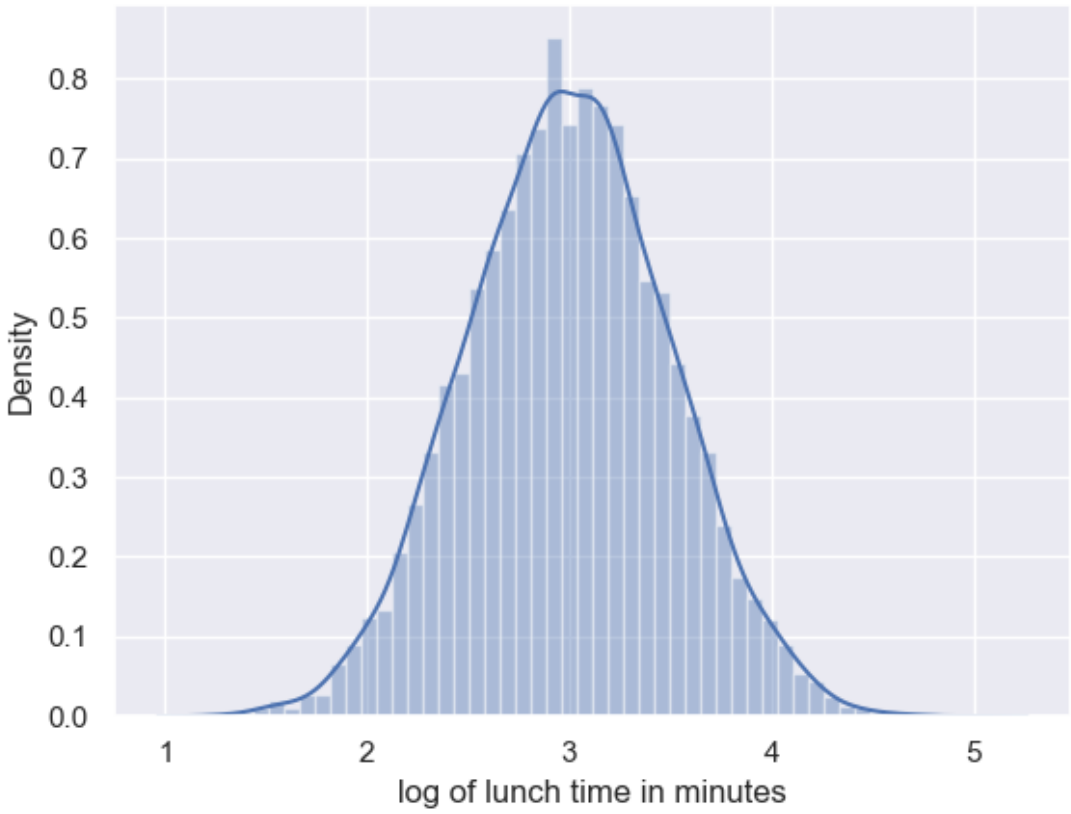


Fig 2

Notice:

1. It's symmetric.
2. It has a mean of 3.
3. Its standard deviation is about 0.5 (I eye-balled it, believe me)

The **probability density function** can be expressed as:

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

This is the famous normal distribution, notice the bell shape! (by the way, fig. 1 is called

Couple things that seem random but are actually defining characteristics of normal distribution:

- A sample has a 68.3% probability of being within 1 standard deviation of the mean(or 31.7% probability of being outside).
- A kurtosis of 3.

Below is python code to generate them:

```
import numpy as np
import pandas as pd
from scipy.stats import norm

num_samples = 10000
samples = norm.rvs(loc=3, scale=.5, size=(1, num_samples))[0]

lunch_time = pd.Series(np.exp(samples),
                       name='lunch time in minutes')

log_lunch_time = pd.Series(samples,
                           name='log of lunch time in minutes')
```

Uncorrelated Normal Distributions

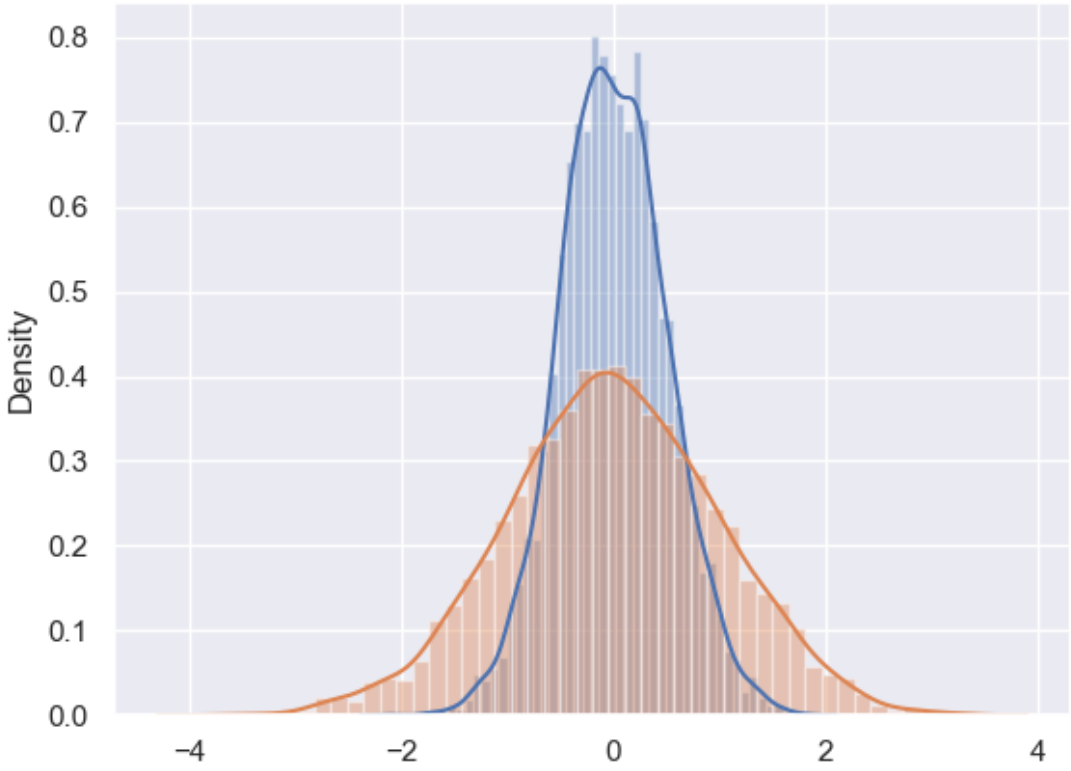
Now that we have had a refresher of normal distribution, what is a multi-variate normal distribution?

First thing that comes to mind is two or more normally distributed variables, and that is true. Let's say I generate samples two normally distributed variables, 5000 sample each:

```
import numpy as np
from scipy.stats import norm
```

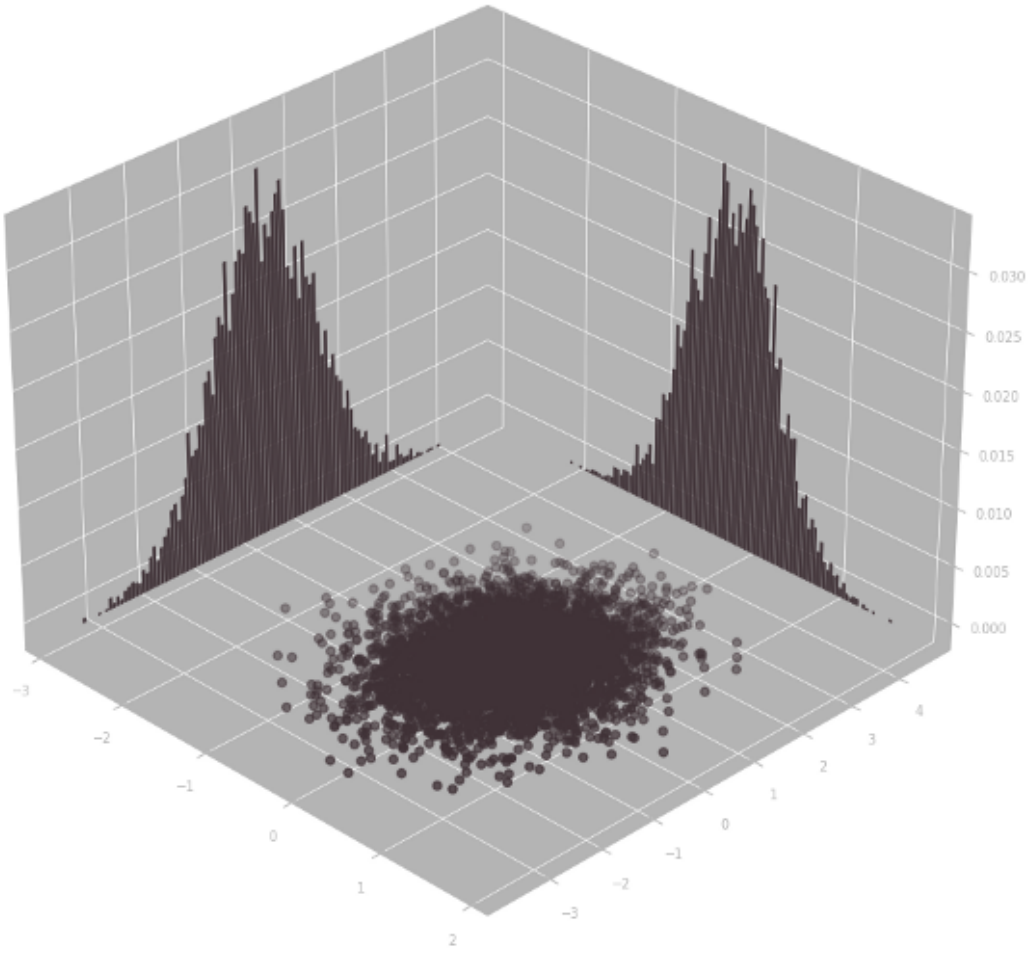



signal01 and signal02 are certainly normally distributed:



never mind the weirdness at the top of the blue histogram, the number of samples isn't high enough

But, there is something more to it, let's plot them in a scatter plot to see:



Do you see how the scatter plot of the two distributions are symmetric about the x-axis and the y-axis? It's because the two distributions are completely uncorrelated:

```
np.corrcoef(signal01, signal02)
'''
Out[1]:
array([[1.        , 0.00384419],
       [0.00384419, 1.        ]])
'''
```

correlations with each other.

Multivariate Normal Distribution

Let's generate some correlated bi-variate normal distributions.

How do you go about doing that, you ask?

First step is to generate 2 standard normal vector of samples:

```
import numpy as np
from scipy.stats import norm

num_samples = 5000

signal01 = norm.rvs(loc=0, scale=1, size=(1, num_samples))[0]
signal02 = norm.rvs(loc=0, scale=1, size=(1, num_samples))[0]
```

Create the desired **variance-covariance(vc) matrix**:

```
# specify desired std
std01 = 11.2
std02 = 0.5

std_m = np.array([
    [std01, 0],
    [0, std02]
])

# specify desired correlation
corr_m = np.array([
    [1, .75],
    [.75, 1]
])

# calc desired covariance (vc matrix)
```



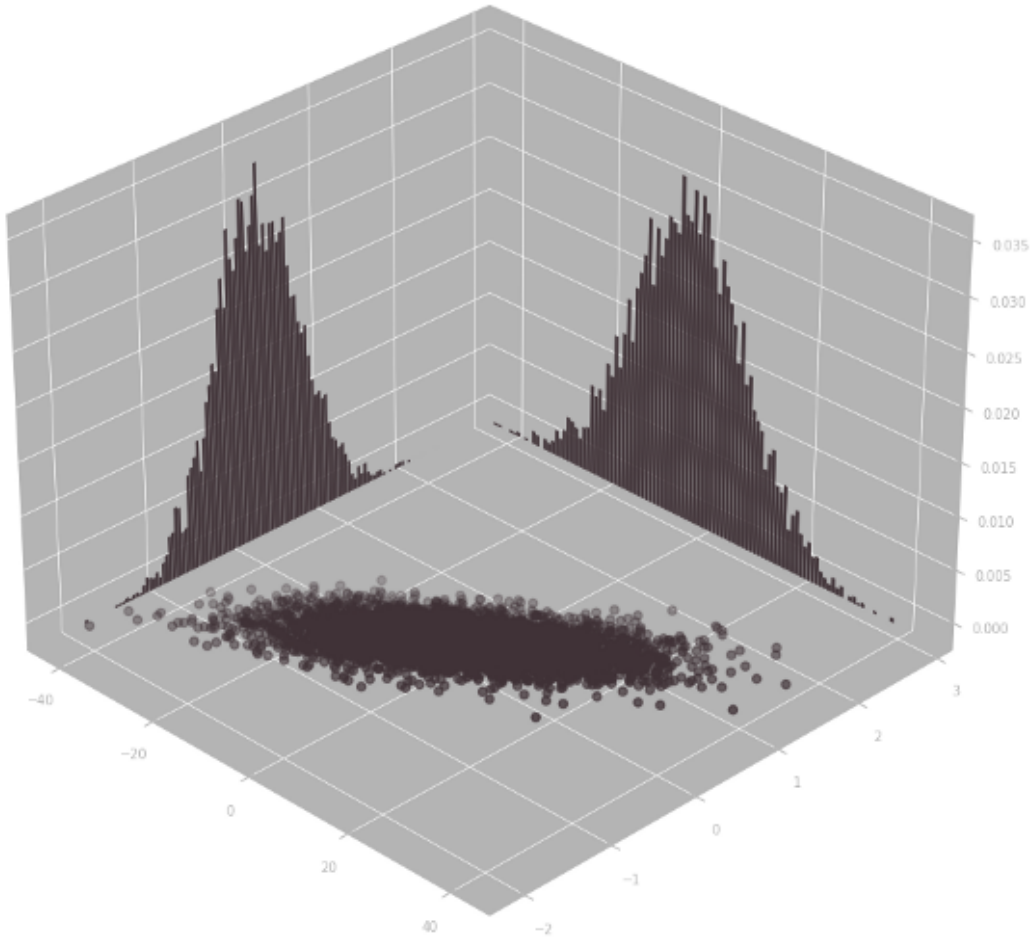

```
from scipy.linalg import cholesky
cky = cholesky(cov_m, lower=True)
```

Now just multiply this matrix to the uncorrelated signals to get the correlated signals:

```
corr_data = np.dot(cky, [signal01, signal02])

signal01 = corr_data[0]
signal02 = corr_data[1]

np.corrcoef(signal01, signal02)

'''
Out[1]:
array([[1.        , 0.75279088],
       [0.75279088, 1.        ]])
'''
```

Let’s take a look at the resulting scatterplot:





See how the scatterplot is not symmetric about the x-axis or the y-axis anymore, and it's becoming more like a line? This is the effect of correlation.

Derivation

Why does this process work?

$$X = \begin{pmatrix} signal01 \\ signal02 \end{pmatrix}$$

the variance covariance of the vector is defined as:

$$VC(X) = \mathbb{E}((X - \mathbb{E}(X))(X - \mathbb{E}(X))^T)$$

if we multiply X by a matrix C, then the variance covariance of the resulting vector is:

$$VC(CX) = \mathbb{E}((CX - \mathbb{E}(CX))(CX - \mathbb{E}(CX))^T) \quad (1)$$
$$VC(CX) = \mathbb{E}((CX - C\mathbb{E}(X))(CX - C\mathbb{E}(X))^T) \quad (2)$$
$$VC(CX) = \mathbb{E}(C(X - \mathbb{E}(X))C^T(X - \mathbb{E}(X))^T) \quad (3)$$
$$VC(CX) = CC^T\mathbb{E}((X - \mathbb{E}(X))(X - \mathbb{E}(X))^T) \quad (4)$$

You see, since the components of our original X vector are uncorrelated, the variance covariance matrix is just equal to:

$$VC(CX) = CC^T$$

This is why we used Cholesky's decomposition! We defined a desired variance covariance matrix of:

```
# calc desired covariance (vc matrix)
cov_m = np.dot(std_m, np.dot(corr_m, std_m))
```





Well, for one thing, if the random variable components in the vector are not normally distribut themselves, the result is definitely not multivariate normally distributed.

Also the covariance matrix has to be positive semidefinite, and that means it has to be symmetr

```
# specify desired correlation
corr_m = np.array([
    [1, .75],
    [.75, 1]
])
```

if you change it to something like:

```
# specify desired correlation
corr_m = np.array([
    [1, .75],
    [.11, 1]
])
```

then the result you get is definitely not a multivariate normal distribution either, since this wou mean that the correlation of signal01 and signal02 is different from the correlation of signal02 and signal01…

Oh yeah, you can actually just use numpy's built-in function: multivariate_normal:

```
mean = [0, 0]
cov = [[1, .5], [.5, 1]]
s1, s2 = np.random.multivariate_normal(mean, cov, 5000).T
```