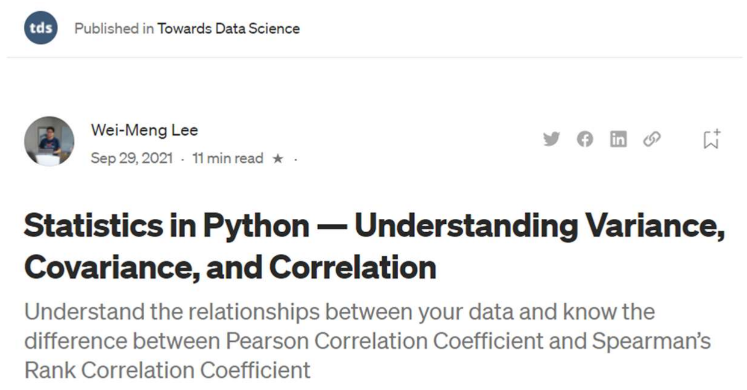

Published in Towards Data Science

Wei-Meng Lee
Sep 29, 2021 · 11 min read ★ ·

Statistics in Python — Understanding Variance, Covariance, and Correlation

Understand the relationships between your data and know the difference between Pearson Correlation Coefficient and Spearman's Rank Correlation Coefficient

One of the topics that a data scientist must understand is the relationships that exist in your dataset. Before you start the machine learning process, it is critical to prepare your data so that only the relevant parts of your dataset is used for training. To understand the relationships in your dataset, you need to understand the following concepts:

- Variance
- Covariance
- Correlation

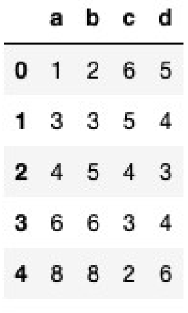
As usual, my aim is to make it easy for you to digest these topics. Let's begin!

Creating the Sample Dataset

To understand relationships in your dataset, let's create a simple one and load in into a Pandas DataFrame:

```
import pandas as pd
import numpy as npdf = pd.DataFrame({
    'a':[1,3,4,6,8],
    'b':[2,3,5,6,8],
    'c':[6,5,4,3,2],
   'd':[5,4,3,4,6]
})
df
```

The dataframe contains five rows and four columns:



	a	b	c	d
0	1	2	6	5
1	3	3	5	4
2	4	5	4	3
3	6	6	3	4
4	8	8	2	6

Variance

Variance is the spread of values in a dataset around its mean value. It tells you how far each number in the dataset is from its mean. The formula for variance (**s²**) is defined as follows:

$$s^2 = \frac{\sum(x_i - \bar{x})^2}{n - 1}$$

Image by author

*For **sample variance**, the denominator is **n-1**. For **population variance**, the denominator is **n**.*

The square root of **variance** (**s²**) is the **standard deviation** (**s**). Variance is calculated by taking the difference of each number in the dataset from the mean, summing all the differences, and finally dividing it by the number of values in the dataset.

A large variance indicates that the numbers in the dataset are far from the mean and far from each other. A small variance, on the other hand, indicates that the numbers are close to the mean and to each other. A variance of 0 indicates that all the numbers in the dataset are the identical. Finally, the valid value of variance is always a positive number (0 or more).

As usual, it is useful to be able to visualize the distribution of numbers in a dataset so that you can better understand the concept of variance.

Using Seaborn, you can plot a strip-plot together with a box-plot to show the distribution of the numbers in columns **a** to **d**:

```
import seaborn as snsg = sns.stripplot(data = df.melt(),
              x = 'variable',
              y = 'value',
              color = 'red')sns.boxplot(data = df.melt(),
          x = 'variable',
          y = 'value',
          color = 'yellow')
```

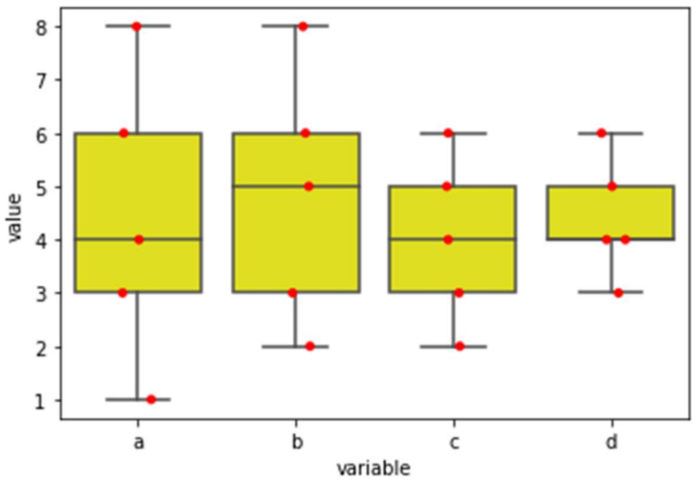


Image by author

As you can see, the values in column **a** are much more dispersed compared to the rest of the columns, and likewise the values in column **b** are more dispersed than **b** and **c**, and so on. The values in **d** are the most closely grouped compared to the rest of the columns. As such, you would expect the variance for **a** would be the largest and the variance for **d** would be the lowest.

Calculating variance using NumPy

Using NumPy, it is easy to calculate the variance for a series of numbers. Here is the statement to calculate the variance for column **a** based on the formula you have seen earlier:

```
(np.square(df['a'] - df['a'].mean())).sum() / (df.shape[0] - 1)
# 7.3
```

However, NumPy also have the **var()** function to calculate the variance of an array. You can directly pass in a dataframe to the **var()** function to calculate the variances of a series of columns in a dataframe:

```
np.var(df[['a','b','c','d']], ddof=1)
# a    7.3
# b    5.7
# c    2.5
# d    1.3
# dtype: float64
```

ddof *stands for* ***Delta Degrees of Freedom****. This value is used in the denominator for the variance calculation (****n — ddof****), where n represents the number of elements. By default ddof is zero (population variance). When* ***ddof*** *is set to* ***1****, you are calculating the sample variance.*

As expected, you can see that column **a** has the largest variance and column **d** has the smallest variance.

Covariance

Now that you have seen the variances of each columns, it is now time to see how columns relate to each other. While **variance** measures the spread of data within its mean value, **covariance** measures the relationalship between *two* random variables.

In statistics, ***covariance*** *is the measure of the directional relationship between two random variables.*

Let’s plot a scatter plot to see how the columns in our dataframe relate to each other. We shall start with the **a** and **b** columns first:

```
import matplotlib.pyplot as plt
plt.scatter(df['a'], df['b'])
plt.xlabel('a')
plt.ylabel('b')
```

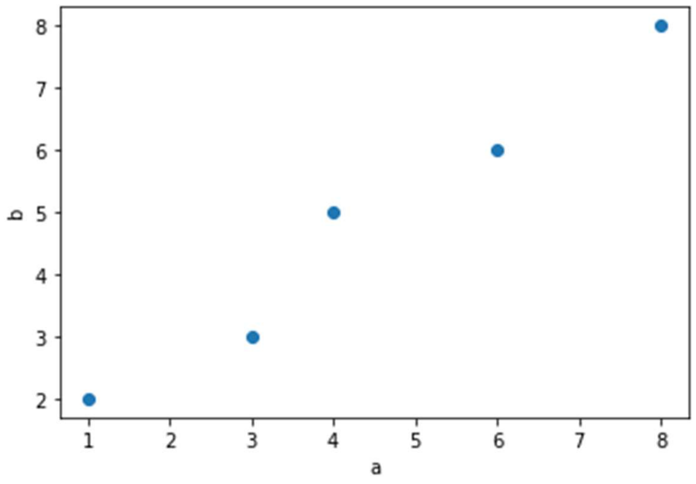


Image by author

As you can see, there seems to be a trend between **a** and **b** — as **a** increases, so does **b**.

*In statistics, **a** and **b** are known to have a **positive** covariance. A positive covariance indicates that both random variables tend to move upward or downward at the same time.*

How about columns **b** and **c**? Let's see:

```
plt.scatter(df['b'], df['c'])
plt.xlabel('b')
plt.ylabel('c')
```

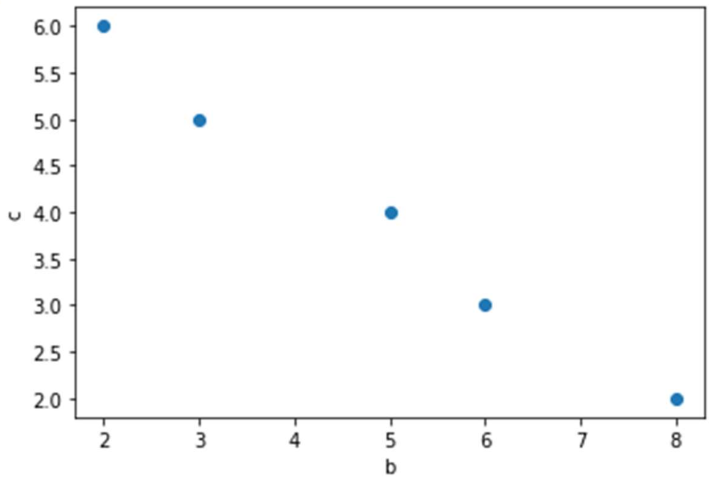


Image by author

This time round, the trend seems to go the other way —
as **b** increases, **c** decreases.

*In statistics, **b** and **c** are known to have a **negative** covariance. A negative covariance indicates that both variables tend to move away from each other — when one moves upward the other moves downward, and vice versa.*

Finally, let's examine columns **c** and **d**:

```
plt.scatter(df['c'], df['d'])
plt.xlabel('c')
plt.ylabel('d')
```

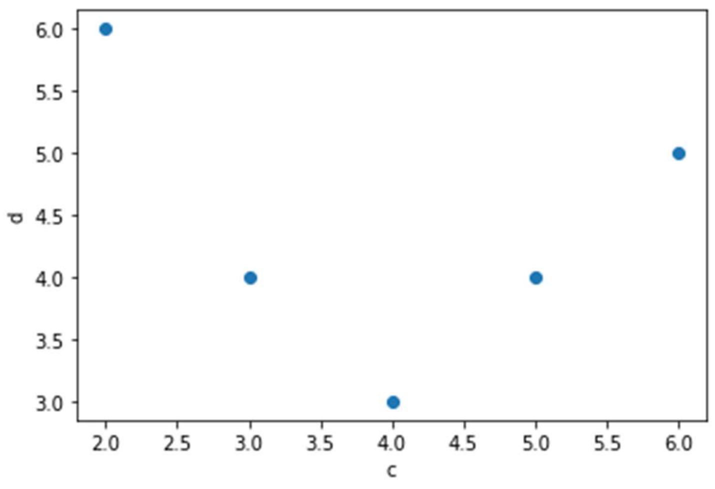


Image by author

There doesn’t seem to exist a direct linear relationship between **c** and **d**.

In statistics, ***c*** *and* ***d*** *are known to have* ***zero*** *covariance (or close to zero). When two random variables are independent, the covariance will be zero.* ***However, the reverse is not necessarily true — a covariance of zero does not mean that 2 random variables are independent (a non-linear relationship can still exist between 2 random variables that has zero covariance). In the above example, you can see that there exists some sort of non-linear v-shape relationship.***

Mathematically, the formula for **covariance** is defined as follows:

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

Image by author

Covariance between 2 random variables is calculated by taking the product of the difference between the value of each random variable and its mean, summing all the products, and finally dividing it by the number of values in the dataset.

As usual, let's calculate the covariance between **a** and **b** manually using NumPy:

```
#---covariance for a and b---
((df['a'] -  df['a'].mean()) * (df['b'] -  df['b'].mean())).sum() / (df.shape[0] - 1)
# 6.35
```

Like variance, NumPy has the **cov()** function to calculate covariance of two random variables directly:

```
np.cov(df['a'],df['b'])
# array([[7.3 , 6.35],
#        [6.35, 5.7 ]])
```

The output of the **cov()** function is a 2D array containing the following values:

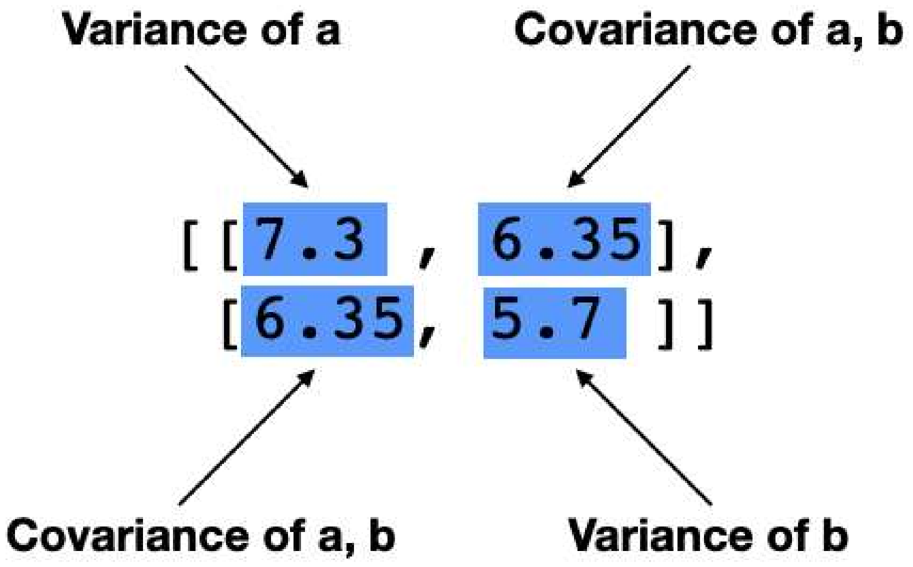


Image by author

In this case, the covariance of a and b is 6.35 (a positive covariance).

Here are the covariance for **b** and **c** (-3.75, a negative covariance):

```
np.cov(df['b'], df['c'])
# array([[ 5.7 , -3.75],
#        [-3.75,  2.5 ]])
```

And the covariance for **c** and **d** (-0.5, a negative covariance):

```
np.cov(df['c'], df['d'])
# array([[ 2.5, -0.5],
#        [-0.5,  1.3]])
```

While the covariance measures the directional relationship between 2 random variables, it does not show the ***strength*** *of the relationship between the 2 random variables. Its value is not constrained, and can be from -infinity to +infinity.*

Also, covariance is dependent on the scale of the values. For example, if you double each value in columns **a** and **b**, you will get a different covariance:

```
np.cov(df['a']*2, df['b']*2)
# array([[29.2, 25.4],
#        [25.4, 22.8]])
```

A much better way to measure the strength of two random variables is *correlation*, which we will discuss next.

Correlation

The correlation between two random variables measures both the *strength* and *direction* of a linear relationship that exists between them. There are two ways to measure correlation:

- **Pearson Correlation Coefficient** — captures the strength and direction of the *linear* association between two continuous variables
- **Spearman's Rank Correlation Coefficient**—determines the strength and direction of the *monotonic* relationship which exists between two ordinal (categorical) or continuous variables.

Pearson Correlation Coefficient

The formula for the **Pearson Correlation Coefficient** is:

$$r = \frac{\mathrm{Cov}(x,y)}{\sigma_x \sigma_y}$$

Image by author

The **Pearson Correlation Coefficient** is defined to be the covariance of x and y divided by the product of each random variable's standard deviation.

Substituting the formula for **convariance** and standard deviation for **x** and **y**, you have:

$$\frac{\frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n-1}}{\sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}\frac{\sum(y_i - \bar{y})^2}{n-1}}}$$

Image by author

Simplifying, the formula now looks like this:

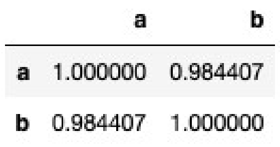
$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Image by author

Pandas have a function **corr()** that calculates the correlation of columns in a dataframe:

```
df[['a','b']].corr()
```

The result is:



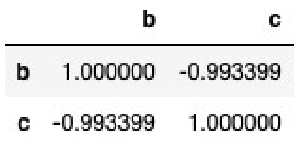
	a	b
a	1.000000	0.984407
b	0.984407	1.000000

Image by author

The diagonal values of 1 indicates the correlation of each column to itself. Obviously, the correlation of **a** to **a** itself is 1, and so is that for column **b**. The value of **0.984407** is the Pearson correlation coefficient of **a** and **b**.

The Pearson correlation coefficient of **b** and **c** is **-0.993399**:

```
df[['b','c']].corr()
```

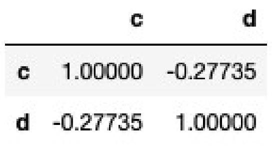


	b	c
b	1.000000	-0.993399
c	-0.993399	1.000000

Image by author

The Pearson correlation coefficient of **c** and **d** is **-0.27735**:

```
df[['c','d']].corr()
```



	c	d
c	1.00000	-0.27735
d	-0.27735	1.00000

Image by author

Like covariance, the sign of the pearson correlation coefficient indicates the direction of the relationship. However, the values of the Pearson correlation coefficient is contrained to be between -1 and 1. Based on the value, you can deduce the following degrees of correlation:

- **Perfect** — values near to ±1
- **High degree** — values between ±0.5 and ±1
- **Moderate degree** — values between ±0.3 and ±0.49
- **Low degree** — values below ±0.29
- **No correlation** — values close to 0

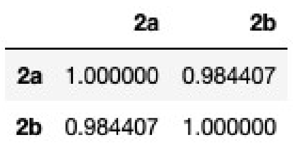
From the above results, you can see that **a,b**, and **b,c** have high degrees of correlation, while **c,d** have very low degree of correlation.

Understanding the correlations between the various columns in your dataset is an important part of the process of preparing your data for machine learning. You want to train your model using the columns that has the highest correlation with the label of your dataset.

Unlike covariance, correlation is not affected by the scale of the values. As an experiment, multiply columns **a** and **b** and you find their correlation:

```
df['2a'] = df['a']*2     # multiply the values in a by 2
df['2b'] = df['b']*2     # multiply the values in b by 2
df[['2a','2b']].corr()   # the result is the same as
                         # df[['a','b']].corr()
```

The result is the same as that of **a** and **b**:



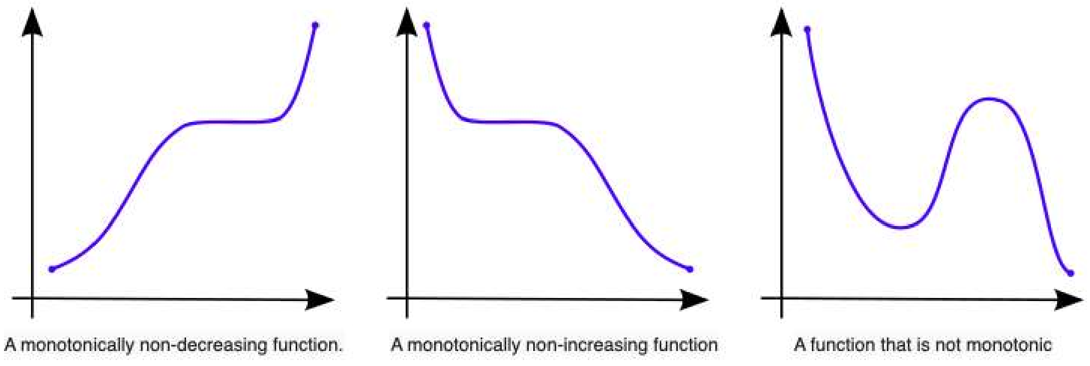
	2a	2b
2a	1.000000	0.984407
2b	0.984407	1.000000

Image by author

Spearman's Rank Correlation Coefficient

If your data is not linearly distributed, you should use **Spearman's Rank Correlation Coefficient** instead of the **Pearson Correlation Coefficient.** The **Spearman's Rank Correlation Coefficient** is designed for distributions that are *monotonic*.

In algebra, a ***montonic function*** *is a function whose gradient never changes sign. In other words, it is a function which is either always increasing or decreasing. The following first two figures are monotonic, while the third is not (since the gradient changes sign a few times going from left to right).*



Source: https://en.wikipedia.org/wiki/Monotonic_function

The formula for **Spearman's Rank Correlation Coefficient** is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2-1)}$$

Image by author

Where **d** is the difference in rank between the 2 random variables. An example will make it clear.

For this example, I will have another dataframe:

```
df = pd.DataFrame({
    'math'   :[78,89,75,67,60,58,71],
    'science':[91,92,90,80,60,56,84]
})
df
```

	math	science
0	78	91
1	89	92
2	75	90
3	67	80
4	60	60
5	58	56
6	71	84

It would be useful to first visualize the data:

```
plt.scatter(df['math'], df['science'])
plt.xlabel('math')
plt.ylabel('science')
```

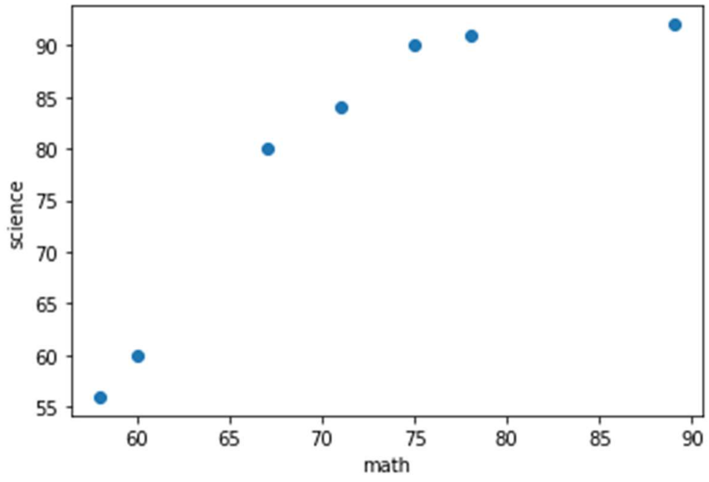
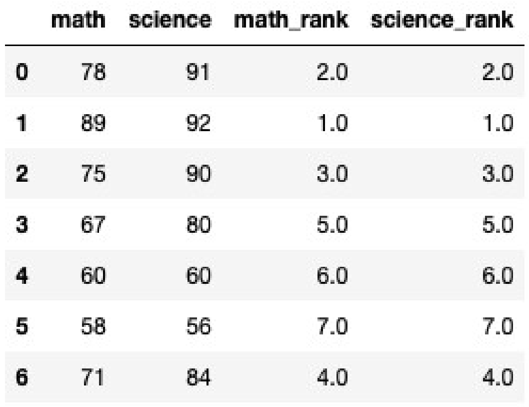


Image by author

And this is looks like a monotonic distribution. The next step is to rank the scores using the **rank()** function in Pandas:

```
df['math_rank'] = df['math'].rank(ascending=False)
df['science_rank'] = df['science'].rank(ascending=False)
df
```

You now have two additional columns containing the ranks for each subject:

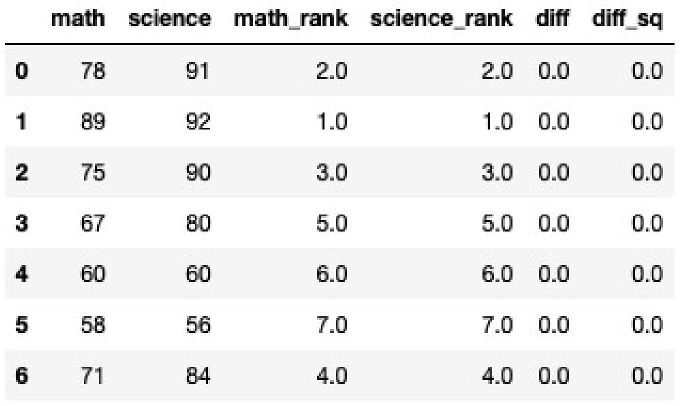


	math	science	math_rank	science_rank
0	78	91	2.0	2.0
1	89	92	1.0	1.0
2	75	90	3.0	3.0
3	67	80	5.0	5.0
4	60	60	6.0	6.0
5	58	56	7.0	7.0
6	71	84	4.0	4.0

Image by author

Let's also create another two new columns to store the differences between the ranks and its squared values:

```
df['diff'] = df['math_rank'] - df['science_rank']
df['diff_sq'] = np.square(df['diff'])
df
```



	math	science	math_rank	science_rank	diff	diff_sq
0	78	91	2.0	2.0	0.0	0.0
1	89	92	1.0	1.0	0.0	0.0
2	75	90	3.0	3.0	0.0	0.0
3	67	80	5.0	5.0	0.0	0.0
4	60	60	6.0	6.0	0.0	0.0
5	58	56	7.0	7.0	0.0	0.0
6	71	84	4.0	4.0	0.0	0.0

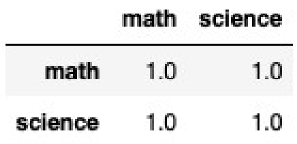
Image by author

You are now ready to calculate the **Spearman's Rank Correlation Coefficient** using the formula defined earlier:

```
n = df.shape[0]
p = 1 - ((6 * df['diff_sq'].sum()) / (n * (n**2 - 1)))
p   # 1.0
```

And you get a perfect 1.0! Of course, to spare you all the effort in calculating the **Spearman's Rank Correlation Coefficient** manually, you can use the **corr()** function and specify '*spearman*' for the **method** parameter:

```
df[['math','science']].corr(method='spearman')
```



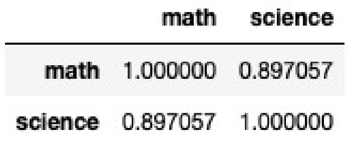
	math	science
math	1.0	1.0
science	1.0	1.0

Image by author

What happens if we calculate the correlation using the Pearson correlation coefficient?

```
df[['math','science']].corr(method='pearson')
```

You get the following:



	math	science
math	1.000000	0.897057
science	0.897057	1.000000

Image by author

Note that the formula for ***Spearman's Rank Correlation Coefficient*** *that I have just listed above is for cases where you have distinct ranks (meaning there is no tie in either math or science scores). In the event of tied ranks, the formula is a little more complicated. For simplicity, I will not go into the formula for tied ranks. The* ***corr()*** *function will automatically handle tied ranks.*

Which method should you use? Pearson or Spearman's

So which method should you use? Remember the following points:

- Pearson correlation describes ***linear*** relationships and spearman correlation describes ***monotonic*** relationships

- A scatter plot would be helpful to visualize the data — if the distribution is linear, use Pearson correlation. If it is monotonic, use Spearman correlation.
- You can also apply both the methods and check which is performing well. For instance if results show spearman rank correlation coefficient is greater than Pearson coefficient, it means your data has monotonic relationships and not linear (just like the above example).

Summary

I hope you now have a much clearer idea of the concept of variance, covariance, and correlation. In particular correlation allows you to know the strength and direction of the relationship between your random variables, and you can make use of either the **Pearson Correlation Coefficient** (for linear relationship) or the **Spearman's Rank Correlation Coefficient** (for monotonic relationship) methods.