Scipy an External Library

This lesson introduces an external scipy library by discussing in detail how scipy provides support to handle statistics and probabilistic functionalities.

WE'LL COVER THE FOLLOWING

Calculating correlations

Scipy is a Python library for scientific computing. Scipy and Numpy are the core libraries that Pandas is built upon. We will discuss Pandas later in the course, but having an understanding of Scipy and Numpy before discussing Pandas is useful.

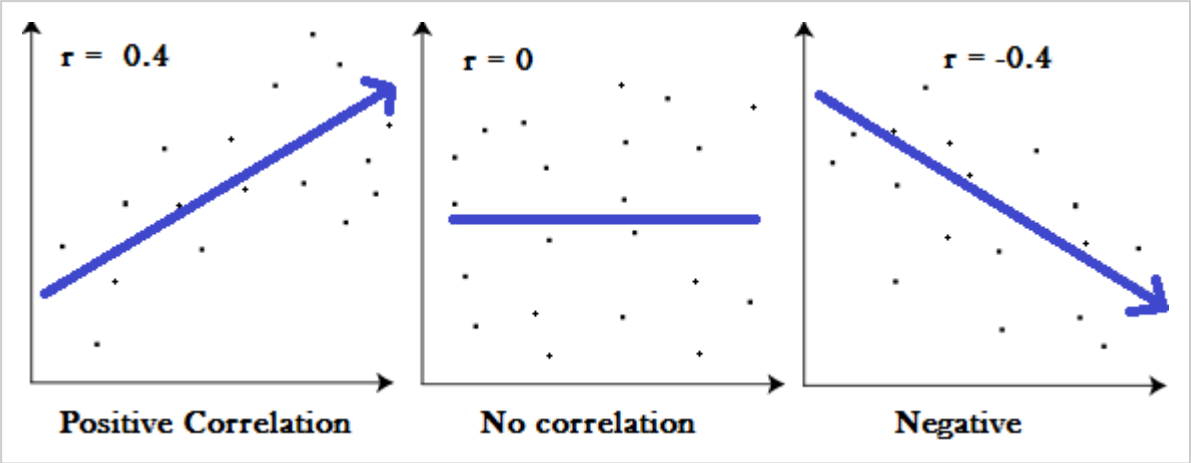
A **correlation** is a numerical measure of the statistical relationship between two variables. For us, those variables will usually be two columns of data, for example, the temperature outside and the likelihood of rain.

One way to calculate the correlation between two vectors of data is with **Pearson's r-value**. This value ranges between -1 and 1. Where -1 means there is a total negative correlation, 0 means no correlation, and 1 means total positive.

> **Note**: these are all linear correlations.

In the image below you can see a graphical representation of correlation

In the image below, you can see a graphical representation of correlation. Source.



An example of a positive correlation might be height and weight as generally taller people weight more. For a negative correlation, the temperature outside and heating costs since as it is warmer outside you run your heater less. For no correlation, the number of shoes you own and your IQ (clearly two completely unrelated entities).

You can find the mathematical formula on the Wikipedia page.

Here is how you would do it with `scipy`:

```
from scipy import stats
import numpy as np

array_1 = np.array([1,2,3,4,5,6])  # Create a numpy array from a list
array_2 = array_1  # Create another array with the same values

print(stats.pearsonr(array_1, array_2))  # Calculate the correlation which will be 1 since th
```

Since the arrays are the same, we expect there to be a perfectly positive correlation. The result is a tuple containing two values. The first value is the value of correlation. It's **1**, which means it's a positive correlation. Where the second value in the tuple is the *p-value*.

Generating samples from distributions

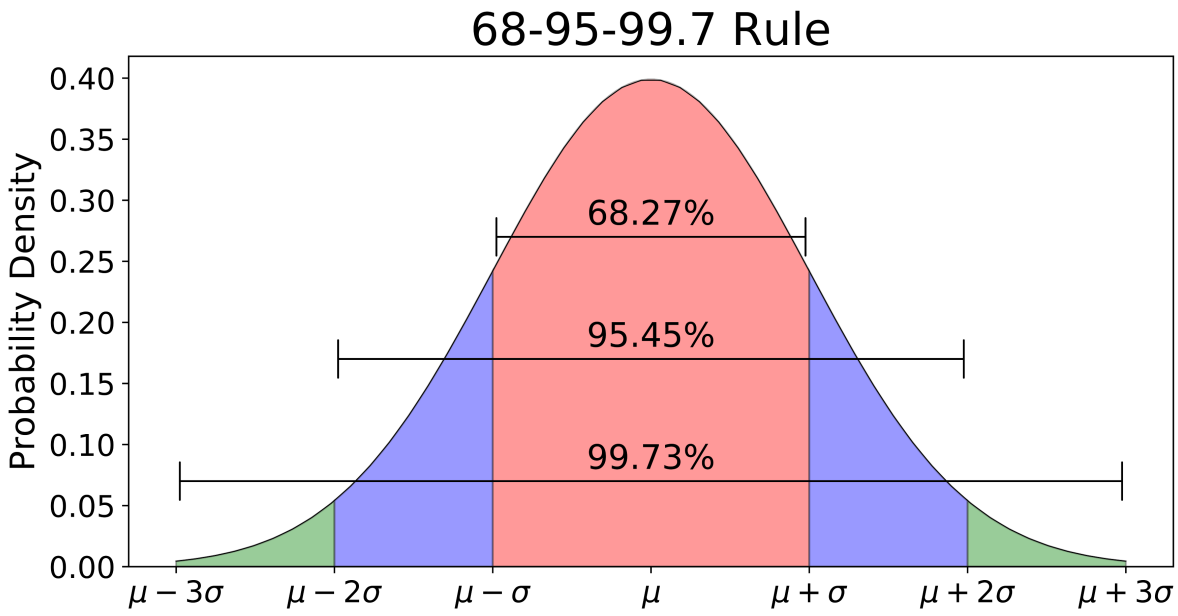
Scipy supports many distributions, we will look at the normal distribution.

Normal distribution

The normal distribution is one that is symmetric around the mean and for which values closer to the mean are more common. It is your standard “bell curve” distribution. See the image below (source).

You can see in the image the probabilities of being within 1, 2, and 3 standard deviations from the mean.

Many natural phenomena have been found to be normally distributed. For example, height.



Say you needed to generate data from a normal distribution with a mean of 0 and a standard deviation of 10; here is how you would do that:

```
from scipy import stats

x = stats.norm.rvs(loc=0, scale=10, size=10)  # Generate 10 values randomly sampled from a no

print(x)
```

The code above uses the `loc` parameter as the *mean*, the `scale` as the *standard deviation*, and the `size` as the *number of samples* to return. If you sampled enough data points and plotted the results, you would see a normal distribution centered around 0 with a standard deviation of 10.

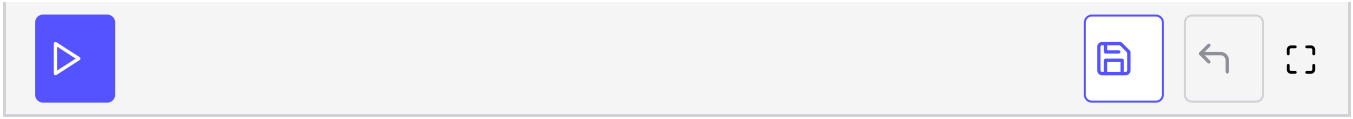
Probability density function

Another common operation on distributions is to calculate the **probability density function**. This function will give you the *relative* likelihood that you would sample a particular value. Let's look at a few:

```
from scipy import stats

p1 = stats.norm.pdf(x=-100, loc=0, scale=10)  # Get probability of sampling a value of -100
p2 = stats.norm.pdf(x=0, loc=0, scale=10)     # Get probability of sampling a value of 0

print(p1)
print(p2)
```



Above we see that relatively it is much more unlikely to sample the value of **-100** (the parameter `x`) at **line 2**, from our distribution than a value of **0** (at **line 3**). This makes sense as our normal distribution is centered on 0 and has a standard deviation of 10.

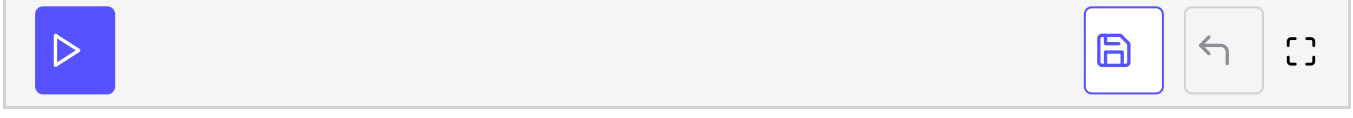
Cumulative distribution function

Another common calculation is the **cumulative distribution function**, the probability of sampling a value less than or equal to `x`.

```
from scipy import stats

p1 = stats.norm.cdf(x=0, loc=0, scale=10)  # Get probability of sampling a value less than or

print(p1)
```



We can see that the *cumulative distribution function* with `x=0` is **0.5** because **0** is the *mean* and with a normal distribution, half of the data is less than or equal to the mean.

Calculating descriptive statistics

Lastly, if you have an array of values in Scipy, you can use the `describe()` function to calculate multiple descriptive statistics for your array. Consider

the following program:

```
from scipy import stats

print(stats.describe(stats.norm.rvs(loc=0, scale=1, size=500)))  # Calculate descriptive stat
```

As you can see above, you get the `number of observations`, the `min`, the `max`, the `mean`, the `variance`, the `skewness`, and the `kurtosis` of your array.

> **Note**: The `variance` is the square of the standard deviation. The `skewness` is a measure of the asymmetry of the distribution. The `kurtosis` is a measure of the "tailedness" of the distribution. A large value usually means there are more outliers.

Now that you have enough information on `numpy` and `scipy` library, the next lesson brings you a challenge to solve.