

# Data Cleaning: Scale and Normalize Data



nishant sethi

Follow

Aug 30, 2018 · 5 min read

In my previous post of Data Cleaning, we discussed *How to Handle Missing Values*. In this post, we're going to be looking at how to scale and normalize data. If you are comfortable with this technique, you can skip to *A Machine Learning Classification Model in Python—Part I*.

To get started, Download the required Dataset which I'm going to use in this post. You can download it from my [GitHub](#) link.

Here's what we're going to do today:

- Get our environment set up
- Scaling vs. Normalization: What's the difference?
- Practice scaling
- Practice normalization

Let's get started!

## Get our environment set up

The first thing we'll need to do is load in the libraries and datasets we'll be using.

```
# modules we'll use
import pandas as pd
import numpy as np

# for Box-Cox Transformation
from scipy import stats

# for min_max scaling
from mlxtend.preprocessing import minmax_scaling

# plotting modules
import seaborn as sns
import matplotlib.pyplot as plt

# read in all our data
kickstarters_2017 = pd.read_csv("../input/kickstarter-projects/ks-projects-201801.csv")

# set seed for reproducibility
np.random.seed(0)
```

Now that we're set up, let's learn about scaling & normalization. (If you like, you can take this opportunity to take a look at some of the data.)

## Scaling vs. Normalization: What's the difference?

One of the reasons that it's easy to get confused between scaling and normalization is because the terms are sometimes used interchangeably and, to make it even more confusing, they are very similar! In both cases, you're transforming the values of numeric variables so that the transformed data points have specific helpful properties. The difference is that, in scaling, you're changing the *range* of your data while in normalization you're changing the *shape of the distribution* of your data. Let's talk a little more in-depth about each of these options.

### Scaling

This means that you're transforming your data so that it fits within a specific scale, like 0–100 or 0–1. You want to scale data when you're using methods based on measures of how far apart data points, like support vector machines, or SVM or k-nearest neighbors, or KNN. With these algorithms, a change of “1” in any numeric feature is given the same importance.

For example, you might be looking at the prices of some products in both Yen and US Dollars. One US Dollar is worth about 100 Yen, but if you don't scale your prices methods like SVM or KNN will consider a difference in price of 1 Yen as important as a difference of 1 US Dollar! This clearly doesn't fit with our intuitions of the world. With currency, you can convert between currencies. But what about if you're looking at something like height and weight? It's not entirely clear how many pounds should equal one inch (or how many kilograms should equal one meter).

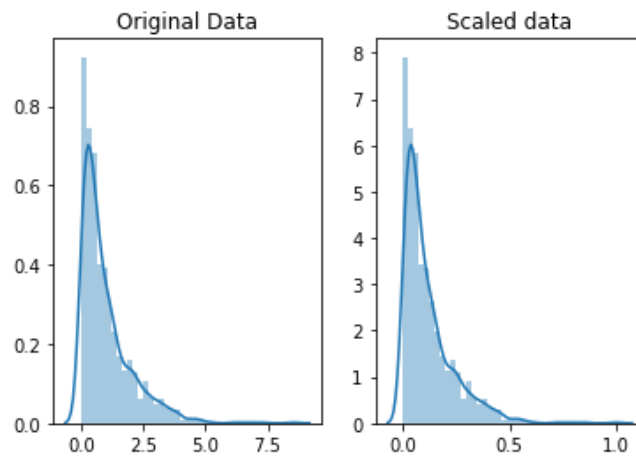
By scaling your variables, you can help compare different variables on equal footing. To help solidify what scaling looks like, let's look at a made-up example. (Don't worry, we'll work with real data in just a second, this is just to help illustrate my point.)

```
# generate 1000 data points randomly drawn from an exponential distribution
original_data = np.random.exponential(size = 1000)

# mix-max scale the data between 0 and 1
scaled_data = minmax_scaling(original_data, columns = [0])

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(original_data, ax=ax[0])
ax[0].set_title("Original Data")
sns.distplot(scaled_data, ax=ax[1])
ax[1].set_title("Scaled data")
```

```
Text(0.5,1,'Scaled data')
```



Notice that the *shape* of the data doesn't change, but that instead of ranging from 0 to 8ish, it now ranges from 0 to 1.

## Normalization

Scaling just changes the range of your data. Normalization is a more radical transformation. The point of normalization is to change your observations so that they can be described as a normal distribution.

**Normal distribution:** Also known as the “bell curve”, this is a specific statistical distribution where a roughly equal observations fall above and below the mean, the mean and the median are the same, and there are more observations closer to the mean. The normal distribution is also known as the Gaussian distribution.

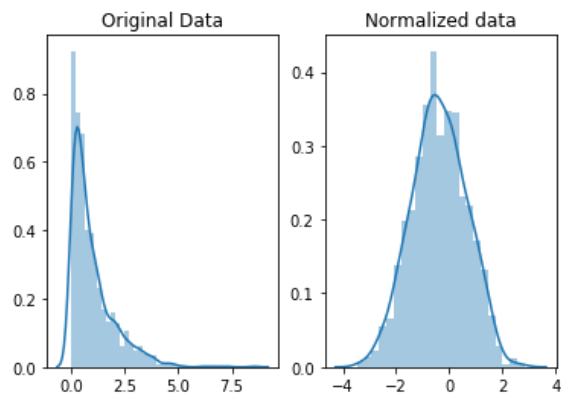
In general, you'll only want to normalize your data if you're going to be using a machine learning or statistics technique that assumes your data is normally distributed. Some examples of these include t-tests, ANOVAs, linear regression, linear discriminant analysis (LDA) and Gaussian naive Bayes. (Pro tip: any method with “Gaussian” in the name probably assumes normality.)

The method we were using to normalize here is called the Box-Cox Transformation. Let's take a quick peek at what normalizing some data looks like:

```
# normalize the exponential data with boxcox
normalized_data = stats.boxcox(original_data)

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(original_data, ax=ax[0])
ax[0].set_title("Original Data")
sns.distplot(normalized_data[0], ax=ax[1])
ax[1].set_title("Normalized data")
```

```
Text(0.5,1,'Normalized data')
```



Notice that the *shape* of our data has changed. Before normalizing it was almost L-shaped. But after normalizing it looks more like the outline of a bell (hence “bell curve”).

## Your turn!

For the following example, decide whether scaling or normalization makes more sense.

- You want to build a linear regression model to predict someone’s grades given how much time they spend on various activities during a normal school week. You notice that your measurements for how much time students spend studying aren’t normally distributed: some students spend almost no time studying and others study for four or more hours every day. Should you scale or normalize this variable?
- You’re still working on your grades study, but you want to include information on how students perform on several fitness tests as well. You have information on how many jumping jacks and push-ups each student can complete in a minute. However, you notice that students perform far more jumping jacks than push-ups: the average for the former is 40 and for the latter only 10. Should you scale or normalize these variables?

## Practice scaling

To practice scaling and normalization, we’re going to be using a dataset of Kickstarter campaigns. (Kickstarter is a website where people can ask people to invest in various projects and concept products.)

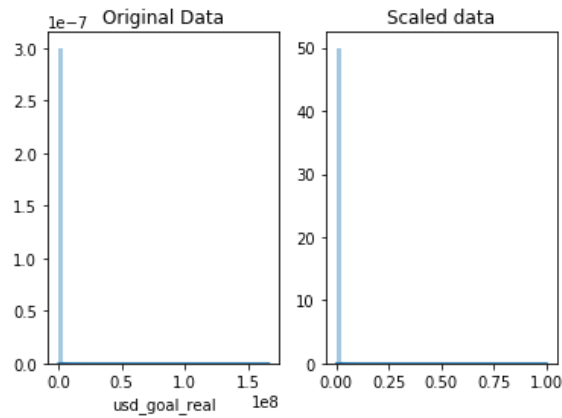
Let’s start by scaling the goals of each campaign, which is how much money they were asking for.

```
# select the usd_goal_real column
usd_goal = kickstarters_2017.usd_goal_real

# scale the goals from 0 to 1
scaled_data = minmax_scaling(usd_goal, columns = [0])

# plot the original & scaled data together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(kickstarters_2017.usd_goal_real, ax=ax[0])
ax[0].set_title("Original Data")
sns.distplot(scaled_data, ax=ax[1])
ax[1].set_title("Scaled data")
```

```
Text(0.5,1,'Scaled data')
```



## Practice normalization

Ok, now let's try practicing normalization. We're going to normalize the amount of money pledged to each campaign.

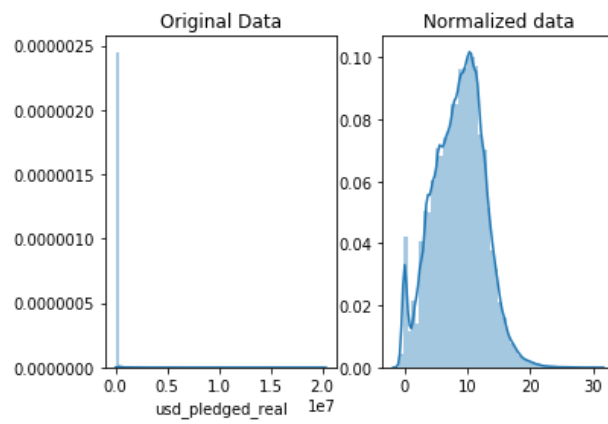
```
# get the index of all positive pledges (Box-Cox only takes positive values)
index_of_positive_pledges = kickstarters_2017.usd_pledged_real > 0

# get only positive pledges (using their indexes)
positive_pledges = kickstarters_2017.usd_pledged_real.loc[index_of_positive_pledges]

# normalize the pledges (w/ Box-Cox)
normalized_pledges = stats.boxcox(positive_pledges)[0]

# plot both together to compare
fig, ax=plt.subplots(1,2)
sns.distplot(positive_pledges, ax=ax[0])
ax[0].set_title("Original Data")
sns.distplot(normalized_pledges, ax=ax[1])
ax[1].set_title("Normalized data")
```

```
Text(0.5,1,'Normalized data')
```



And that's it! In my next Post on Data Cleaning, I'll discuss *How to Parse Dates*.

If you have any questions, be sure to post them in the comments below.

I welcome feedback and constructive criticism and can be reached on Facebook.

Happy coding!!