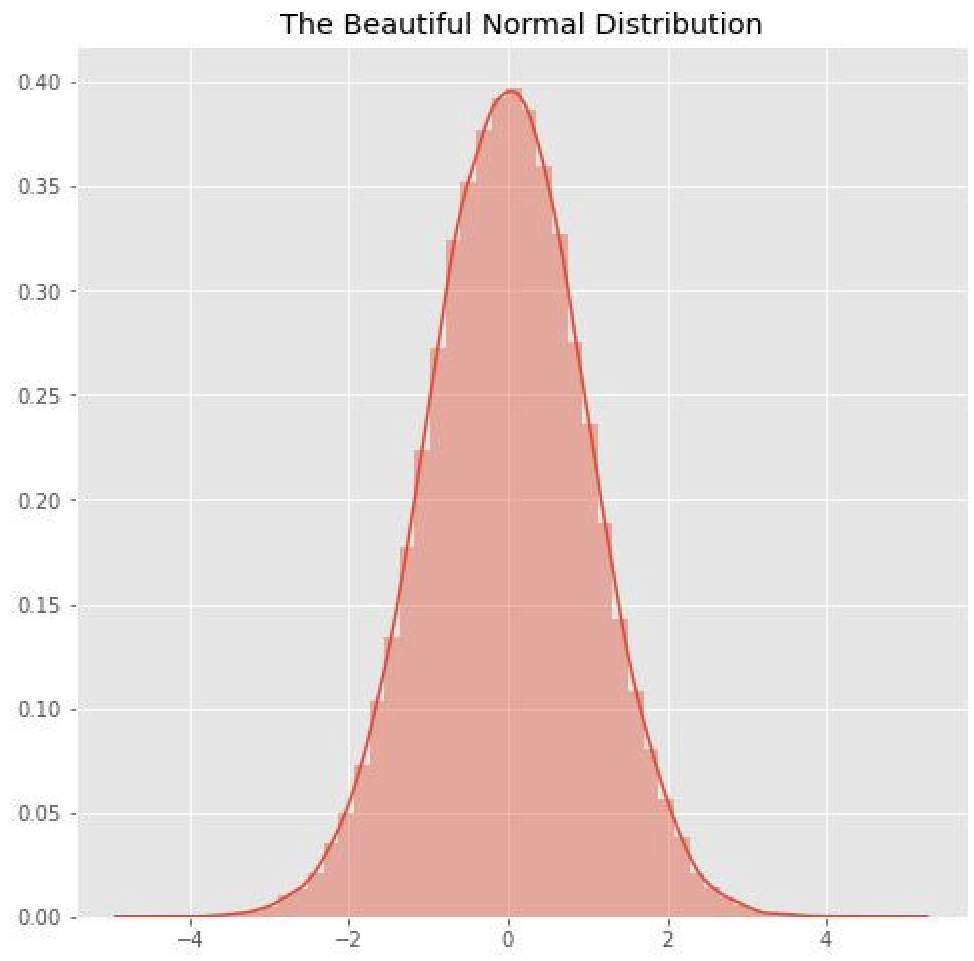
Box-Cox Transformation: Explained

What the Box-Cox transformation is and how to implement it in Python.

Andrew Plummer Sep 2, 2020 · 4 min read ★



The textbook is short, easy for beginners, and a wonderful appendage to the monstrous mess of books, YouTube videos, and MOOCs that aspiring data scientists simply "must" learn.

More importantly, the textbook is free.

The authors of *Forecasting* devote one sub-chapter to transforming data (Section 3.2: "Transformations and Adjustments"), where they go over four types of transformations.

One of these transformations (and the first I was introduced to in undergrad) is the Box-Cox Transformation.

Why Would We Want to Transform Our Data?

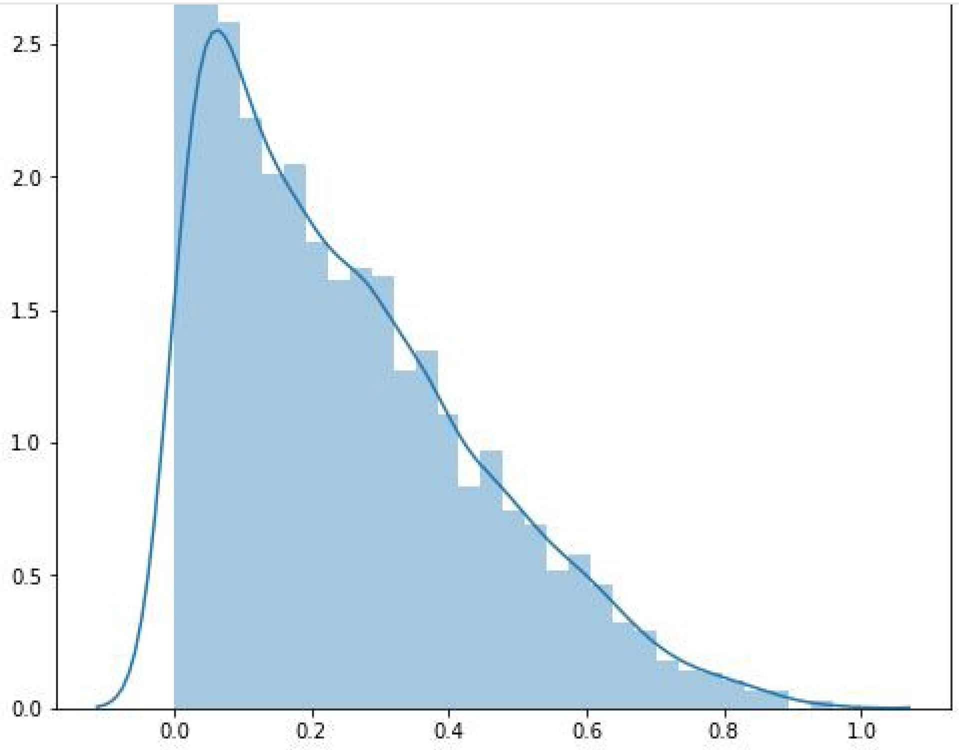
The Box-Cox transformation transforms our data so that it closely resembles a normal distribution.

In many statistical techniques, we assume that the errors are normally distributed. This assumption allows us to construct confidence intervals and conduct hypothesis tests. By transforming your target variable, we can (hopefully) normalize our errors (if they are not already normal).

Additionally, transforming our variables can improve the predictive power of our models because transformations can cut away white noise.

Suppose we had a Beta distribution, where alpha equals 1 and beta equals 3. If we plot this distribution, then it might look something like below:

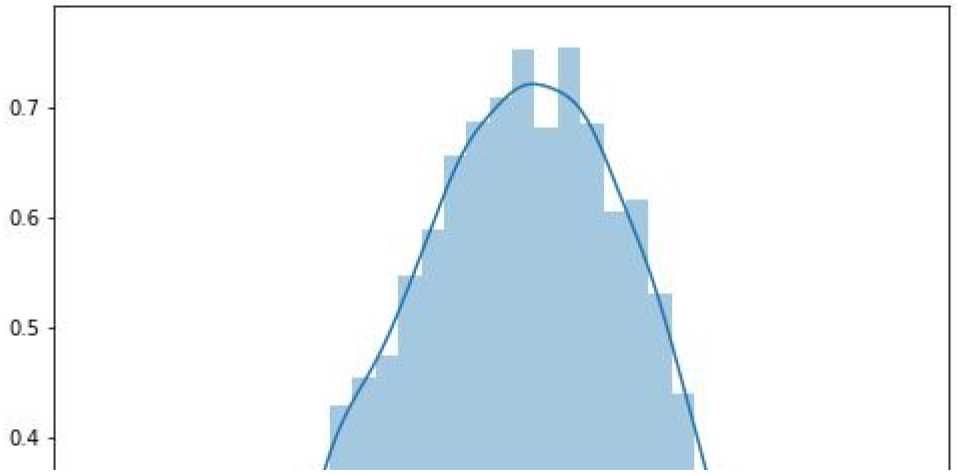
```
plt.figure(figsize = (8, 8))
data = np.random.beta(1, 3, 5000)
sns.distplot(data)
plt.show()
```

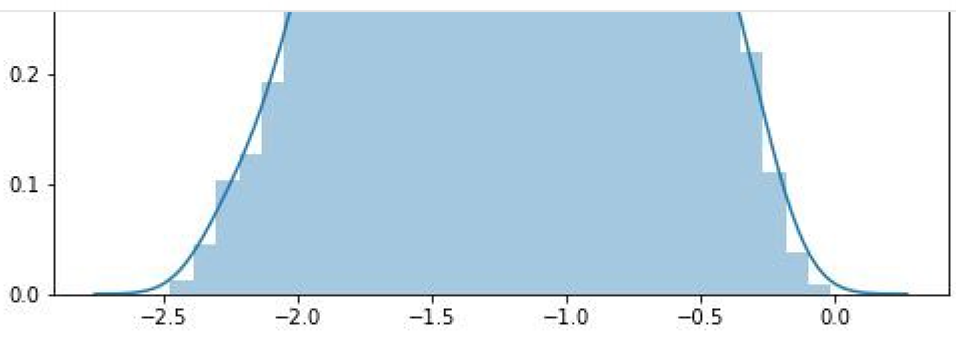


Plotted with Seaborn.

We can use the Box-Cox transformation to transform the above into as close to a normal distribution as the Box-Cox transformation permits.

```
tdata = boxcox(data)[0]
plt.figure(figsize = (8, 8))
sns.distplot(tdata)
plt.show()
```





Plotted with Seaborn

And now our data looks more like a normal distribution.

What is the Box-Cox Transformation?

If *w* is our transformed variable and *y* is our target variable, then

$$w_t = \begin{cases} \log(y_t) & \text{if } \lambda = 0; \\ (y_t^\lambda - 1)/\lambda & \text{otherwise.} \end{cases}$$

Photo from Rob Hyndman's and George Athanasopoulos's "Forecasting".

where *t* is the time period and lambda is the parameter that we choose (you can perform the Box-Cox transformation on non-time series data, also).

Notice what happens when lambda equals 1. In that case, our data shifts down but the shape of the data does not change. Therefore, if the optimal value for lambda is 1, then the data is already normally distributed, and the Box-Cox transformation is unnecessary.

How do we choose lambda?

We choose the value of lambda that provides the best approximation for the normal distribution of our response variable.

Scipy has a boxcox function that will choose the optimal value of lambda for us.

Simply pass a 1-D array into the function and it will return the Box-Cox transformed array and the optimal value for lambda. You can also specify a number, *alpha*, which calculates the confidence interval for that value. (For example, $alpha = 0.05$ gives the 95% confidence interval).

If *llf* is the log-likelihood function, then the confidence interval for lambda can be written as

$$llf(\hat{\lambda}) - llf(\lambda) < \frac{1}{2}\chi^2(1 - \alpha, 1),$$

From Scipy documentation on Box-Cox function.

where X^2 is the chi-squared distribution.

(It may be unnecessary to transform your data if the confidence interval includes 1).

Next, fit your model to the Box-Cox transformed data. However, you must revert your data to its original scale when you are ready to make predictions.

For example, your model might predict that the Box-Cox transformed value, given other features, is 1.5. You need to take that 1.5 and revert it to its original scale (the scale of your target variable).

Thankfully, Scipy has a function for this, also.

```
scipy.special.inv_boxcox(y, lambda)
```

Pass in the data to be transformed, *y*, and the lambda with which you had transformed your data.

What are the limitations of using the Box-Cox Transformation?

difficult to interpret than if we simply applied a log transform.

A second issue is that the Box-Cox transformation usually gives the median of the forecast distribution when we revert the transformed data to its original scale. Occasionally, we want the mean (not the median) and there are ways we can do this, which I may discuss in a later article.

Conclusion

Now you know about the Box-Cox transformation, its implementation in Python, as well as its limitations.

I hope this article was of value to you!

Hey! Like what you just read? Consider using my membership link to join the partner program. (Note: I receive halfof the monthly payments that you make to Medium).

https://andrewmplummer.medium.com/membership

Data Science Statistics Box Cox Normal Distribution Data Transformation