How can I transform data in Python using logarithmic, square root, and cube root functions?

Authored by

stats writer

May 12, 2024

Python is a powerful programming language that allows for various data transformations. One way to transform data in Python is by using logarithmic, square root, and cube root functions. Logarithmic functions can be used to compress large ranges of data into smaller values, while square root and cube root functions can be used to normalize skewed data. These functions are built-in to Python and can be applied to numerical data with ease. By incorporating these functions into data analysis and manipulation, we can effectively transform and manipulate data to better suit our needs.

Transform Data in Python (Log, Square Root, Cube Root)

Many statistical tests make the assumption that datasets are normally distributed. However, this is often not the case in practice.

One way to address this issue is to transform the distribution of values in a dataset using one of the three transformations:

1. Log Transformation: Transform the response variable from y to log(y).

2. Square Root Transformation: Transform the response variable from y to $\sqrt{y}$.

3. Cube Root Transformation: Transform the response variable from y to y1/3.

By performing these transformations, the dataset typically becomes more normally distributed.

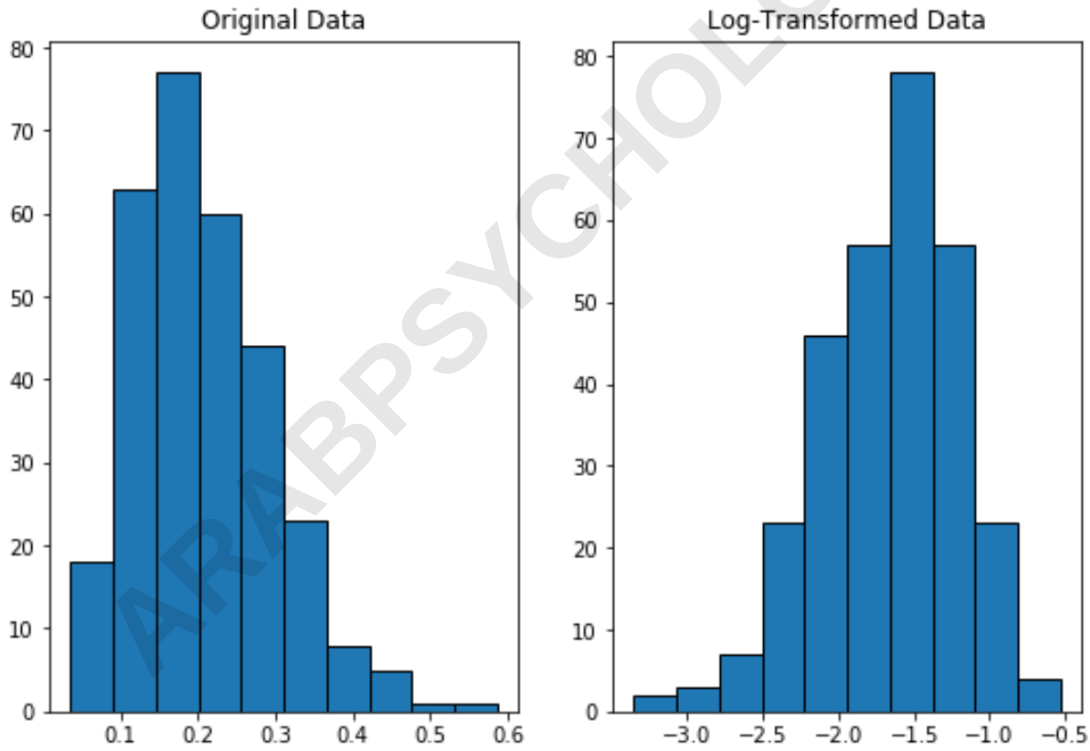
The following examples show how to perform these transformations in Python.

Log Transformation in Python

The following code shows how to perform a log transformation on a variable and create side-by-side plots to view the original distribution and the log-transformed distribution of the data:

```
import numpy as np
import matplotlib.pyplotas plt

#make this example reproducible
np.random.seed(0)

#create beta distributed random variable with 200 values
data = np.random.beta(a=4, b=15, size=300)

#create log-transformed data
data_log = np.log(data)

#define grid of plots
```

```
fig, axs = plt.subplots(nrows=1, ncols=2)

#create histograms
axs.hist(data, edgecolor='black')
axs.hist(data_log, edgecolor='black')

#add title to each histogram
axs.set_title('Original Data')
axs.set_title('Log-Transformed Data')
```

Notice how the log-transformed distribution is more normally distributed compared to the original distribution.

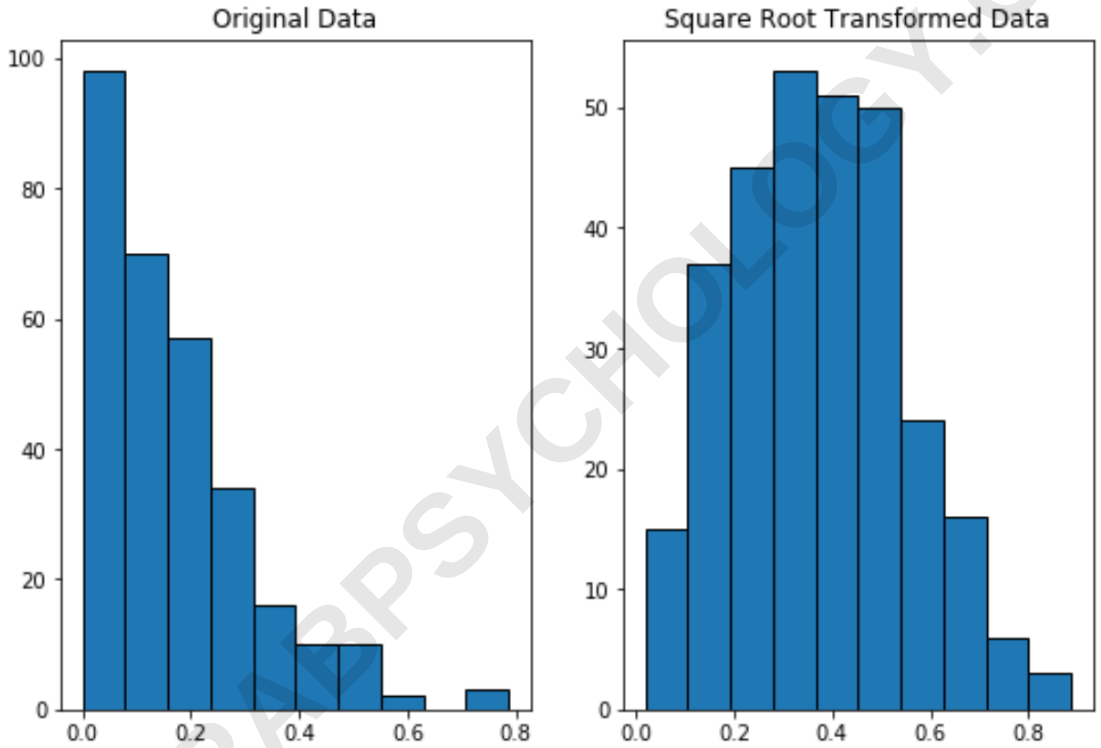
It's still not a perfect "bell shape" but it's closer to a normal distribution that the original distribution.

Square Root Transformation in Python

The following code shows how to perform a square root transformation on a variable and create side-by-side plots to view the original distribution and the square root transformed distribution of the data:

```
import numpy as np
import matplotlib.pyplotas plt

#make this example reproducible
np.random.seed(0)

#create beta distributed random variable with 200 values
data = np.random.beta(a=1, b=5, size=300)

#create log-transformed data
data_log = np.sqrt(data)

#define grid of plots
fig, axs = plt.subplots(nrows=1, ncols=2)

#create histograms
```

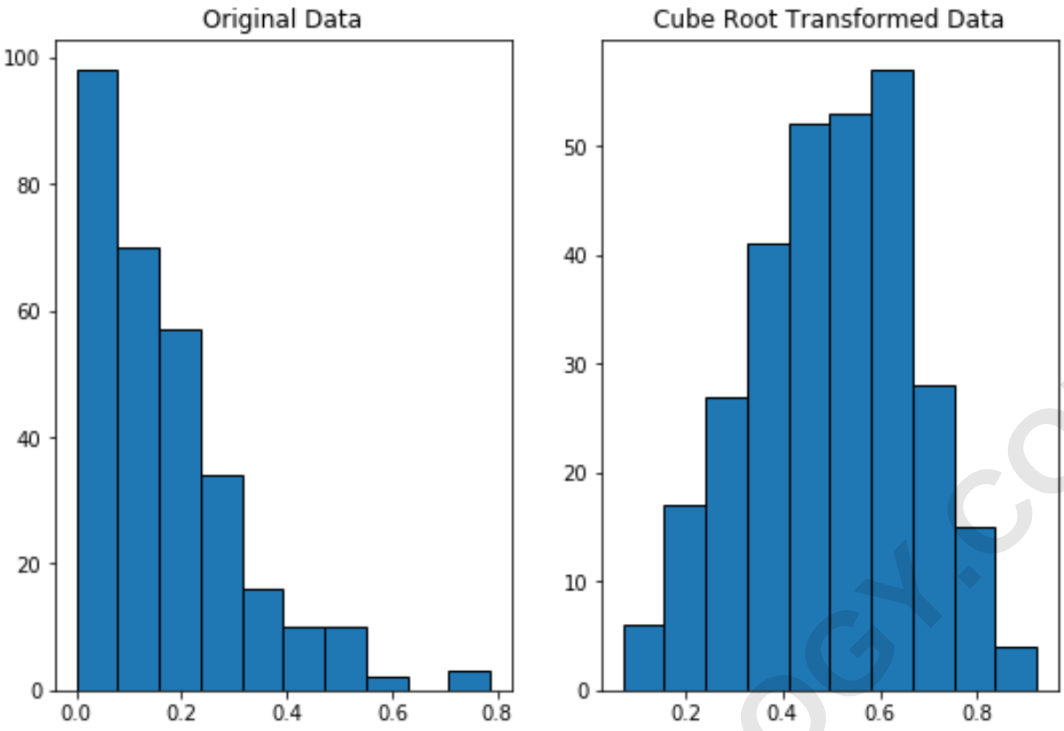
```
axs.hist(data, edgecolor='black')
axs.hist(data_log, edgecolor='black')

#add title to each histogram
axs.set_title('Original Data')
axs.set_title('Square Root Transformed Data')
```



Cube Root Transformation in Python

The following code shows how to perform a cube root transformation on a variable and create side-by-side plots to view the original distribution and the cube root transformed distribution of the data:

```
import numpy as np
import matplotlib.pyplotas plt

#make this example reproducible
np.random.seed(0)

#create beta distributed random variable with 200 values
data = np.random.beta(a=1, b=5, size=300)

#create log-transformed data
data_log = np.cbrt(data)

#define grid of plots
fig, axs = plt.subplots(nrows=1, ncols=2)

#create histograms
axs.hist(data, edgecolor='black')
axs.hist(data_log, edgecolor='black')

#add title to each histogram
axs.set_title('Original Data')
axs.set_title('Cube Root Transformed Data')
```



Notice how the cube root transformed data is much more normally distributed than the original data.