Scale and normalize data

Import Libraries

In [1]:
```
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 the data
kickstarters_2017 = pd.read_csv("../input/kickstarter-projects/ks-projects-201801.csv")

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

Scaling vs. Normalization

In scaling, you're changing the *range* of your data while in normalization you're changing the *shape of the distribution* of your data.

Scaling involves transforming your data to fit within a specific scale, such as 0-100 or 0-1. You should scale data when using methods based on measures of the distance between data points, like support vector machines (SVM) or k-nearest neighbors (KNN). These algorithms treat a change of "1" in any numeric feature as equally important.

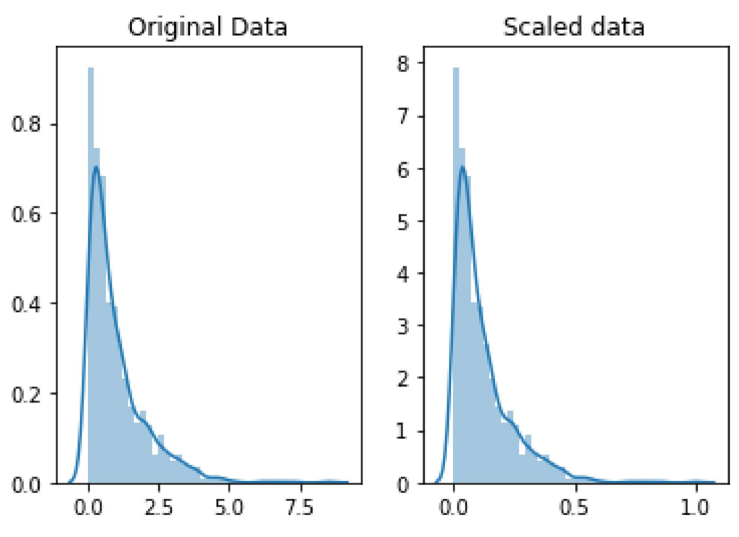
For instance, consider prices of products in both Yen and US Dollars. While one US Dollar is roughly equal to 100 Yen, failure to scale your prices means methods like SVM or KNN would view a 1 Yen difference as significant as a 1 US Dollar difference. Clearly, this doesn't align with our real-world understanding. When dealing with currencies, you can convert between them. However, what about comparisons involving measurements like height and weight? It's not immediately evident how many pounds should equate to one inch (or how many kilograms should correspond to one meter).

In [2]:
```
# 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")
```

Out[2]:
```
Text(0.5,1,'Scaled data')
```



Notice that the shape of the data doesn't change; rather than ranging from 0 to around 8, it now ranges from 0 to 1.

Normalization

Scaling merely alters the range of your data, while normalization represents a more substantial transformation. The purpose of normalization is to align your observations in a manner that resembles a normal distribution.

The normal distribution, commonly known as the 'bell curve,' denotes a specific statistical distribution where observations are approximately equally distributed above and below the mean. Additionally, the mean and median coincide, and there's a higher concentration of observations closer to the mean. This distribution is also referred to as the Gaussian distribution.

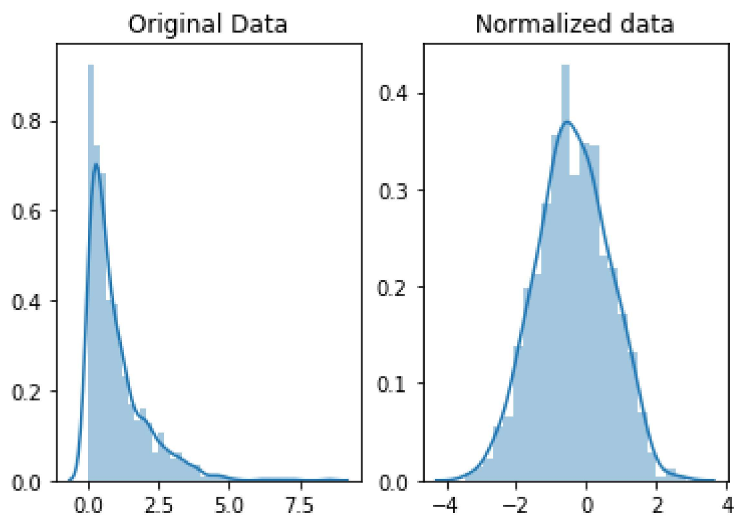
In practice, you'd typically normalize your data if you plan to employ machine learning or statistical techniques assuming a normal distribution. Examples of such techniques include t-tests, ANOVAs, linear regression, linear discriminant analysis (LDA), and Gaussian naive Bayes. (A useful tip: methods bearing 'Gaussian' in their name often presume normality.)

The method we're employing here for normalization is the Box-Cox Transformation. Let's take a quick look at how normalizing some data appears:

In [3]:
```
# 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")
```

Out[3]:
```
Text(0.5,1,'Normalized data')
```



Observe the change in our data's shape. Prior to normalization, it had an almost L-shaped distribution. However, post-normalization, it resembles the outline of a bell curve, hence the term 'bell curve'.

Practice scaling data

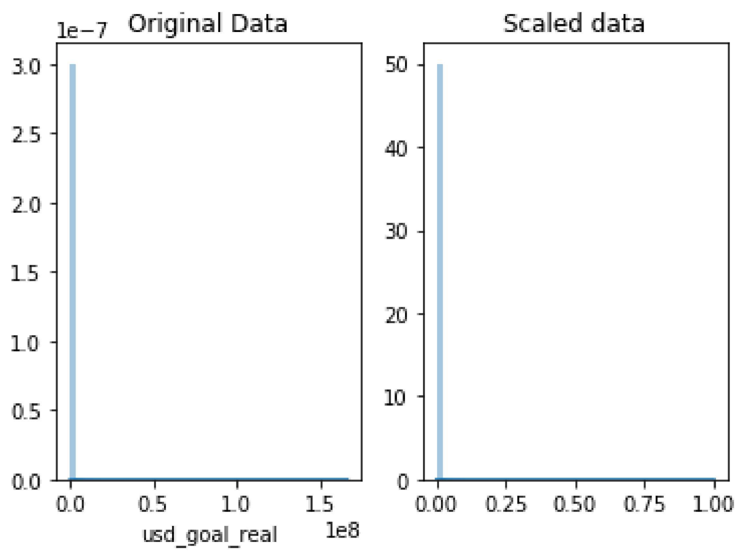
To practice scaling and normalization, we'll utilize a dataset comprising Kickstarter campaigns. (Kickstarter is a platform where individuals seek investments for diverse projects and conceptual products.) Our initial step involves scaling the campaign goals, representing the funding targets for each project.

In [4]:
```
# 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")
```

Out[4]:
```
Text(0.5,1,'Scaled data')
```



You'll notice that scaling significantly altered the scales on the plots, although it didn't affect the fundamental shape of the data. The distribution still showcases that the majority of campaigns have relatively modest goals, while a few exhibit exceedingly large funding targets.

Practice data normalization

We're going to normalize the amount of money pledged to each campaign.

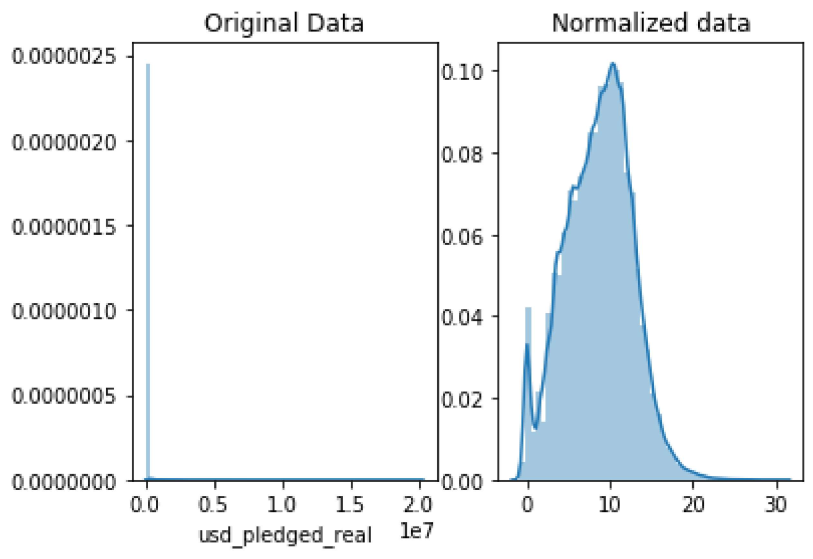
In [6]:
```
# get the index of all positive pledges (Box-Cox only takes postive values)
index_of_positive_pledges = kickstarters_2017.usd_pledged_real > 0

# 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")
```

Out[6]:
```
Text(0.5,1,'Normalized data')
```



It's not perfect (as it seems that many pledges received very few contributions), but the distribution is significantly closer to a normal pattern.