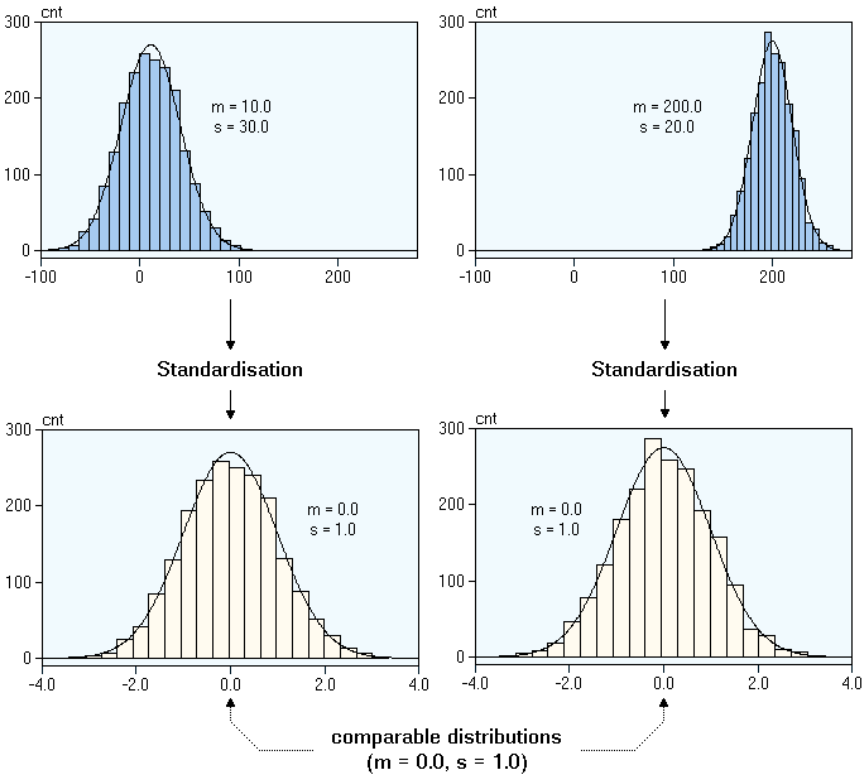
Standardize or Normalize? — Examples in Python

Robert R.F. DeFilippi Follow
Apr 29, 2018 · 6 min read

A common misconception is between what it is—and when to—`standardize` data versus `normalize` date.



What Happens During Standardization

Let's start with `normalization`.

Here your data `Z` is rescaled such that any specific `z` will now be `0 ≤ z ≤ 1`, and is done through this formula:

$$z = \frac{x \ - \min(x)}{\max(x) \ - \min(x)}$$

How to Normalize

Let's to do this with python on a dataset you can quickly access.

```
from sklearn import preprocessing
import numpy as np
```

```
# Get dataset
df =
pd.read_csv("https://storage.googleapis.com/mledudatasets/ca
lifornia_housing_train.csv", sep=",")
```

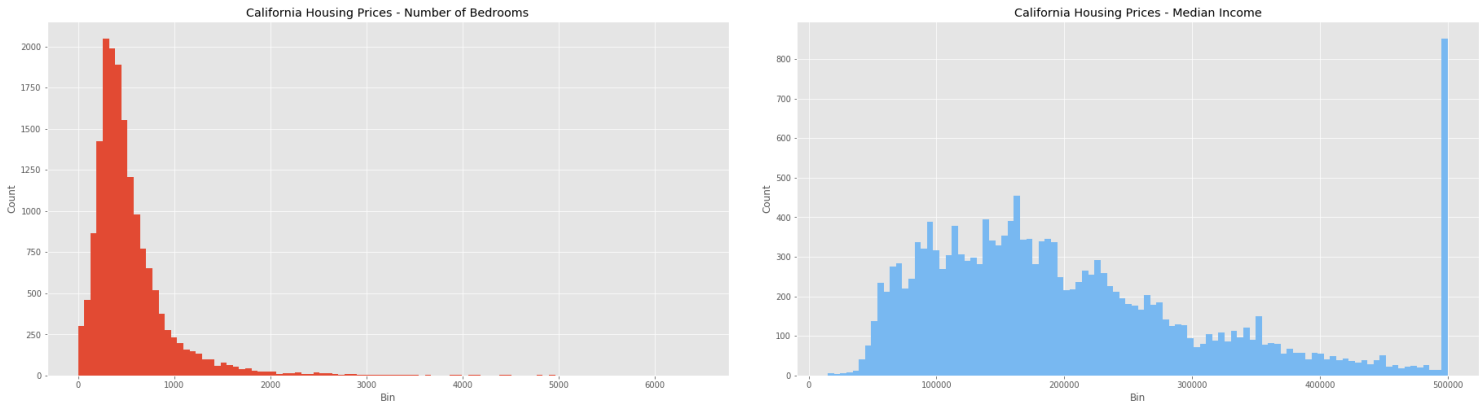
```
# Normalize total_bedrooms column
x_array = np.array(df['total_bedrooms'])
normalized_X = preprocessing.normalize([x_array])
```

Why would we normalize in the first place?

> *1. Normalization makes training less sensitive to the scale of features, so we can better solve for coefficients.*

Consider the dataset above of housing prices in California, which have features such as the `number of bedrooms` and the `median household income` . Each have different units and scales, so consider these feature attributes when going through this article.

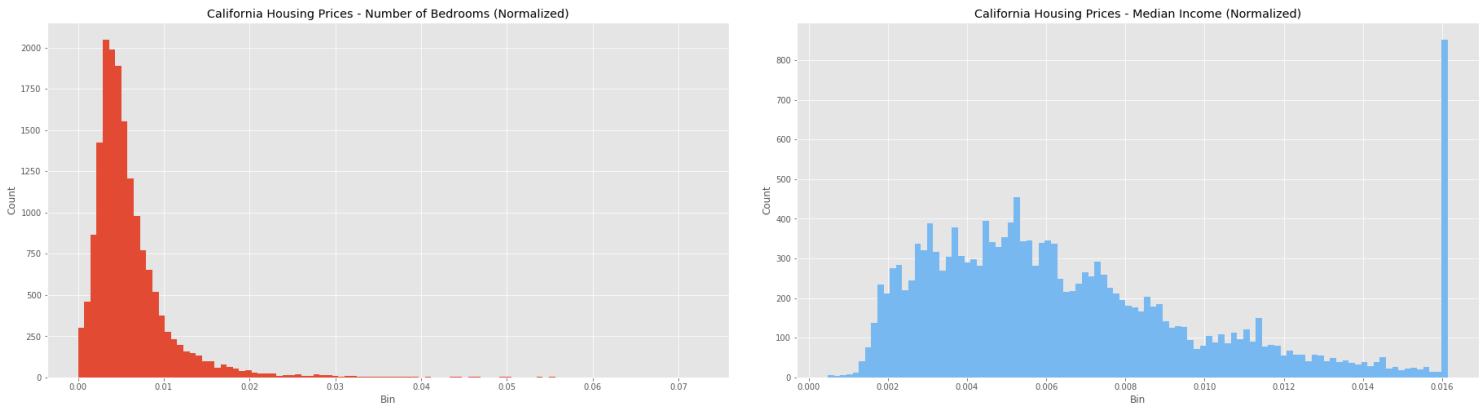
Let's start by looking at both features without normalization.



Raw Values

We can see that there are some odd behaviours with both features (how can we have the number of `bedrooms` be over 1000?) as well as massive outliers and binning issues. We also have a clustering of `income` at $500,000 so the dataset probably puts anyone over that bracket into that bin. It's going to be hard to equate both these features as they are right now.

Let see what `normalization` does.



Normalized Values

All the values are all now between 0 and 1, and the outliers are gone, but still remain visible within our normalized data. However, our features are now more consistent with each other, which will allow us to evaluate the output of our future models better.

> *2. The use of a normalization method will improve analysis from multiple models.*

Additionally, if we were to use any algorithms on this data set before we normalized it would be hard (potentially not possible) to converge the vectors because of the scaling issues. Normalization makes the data better conditioned for convergence.

> *3. Normalizing will ensure that a convergence problem does not have a massive variance, making optimization feasible.*

But wait … sometimes you might not want to `normalize` your data.

The data provided is proportional, so `normalizing` might not provide correct estimators. Or, the scale between your data features does matters so you want to keep in your dataset. You need to think about your data, and understand if the transformations you're applying are in line with the outcomes you're searching for.

Keep in mind, there is some debate stating it is better to have the input values centred around 0—standardization—rather than between 0 and 1. So doing your research is important as well, so you understand what type of data is needed by your model.

. . .

So now that we know how and why to normalize, let's move on to standardization.

Here your data `Z` is rescaled such that $\mu = 0$ and $\sigma = 1$, and is done through this formula:

$$z = \frac{x_i - \mu}{\sigma}$$

Standardization Formula

Why would we do this?

> *1. Compare features that have different units or scales.*

Consider our data above with `housing` and `income`, both have different scales and units. We can start to compare these features and use them in our models once we have `standardized` them.

Later, when you're running models (logistic regression, SVMs, perceptrons, neural networks etc.) the estimated weights will update similarly rather than at different rates during the build process. This will give you more accurate results when the data has been first `standardized`.

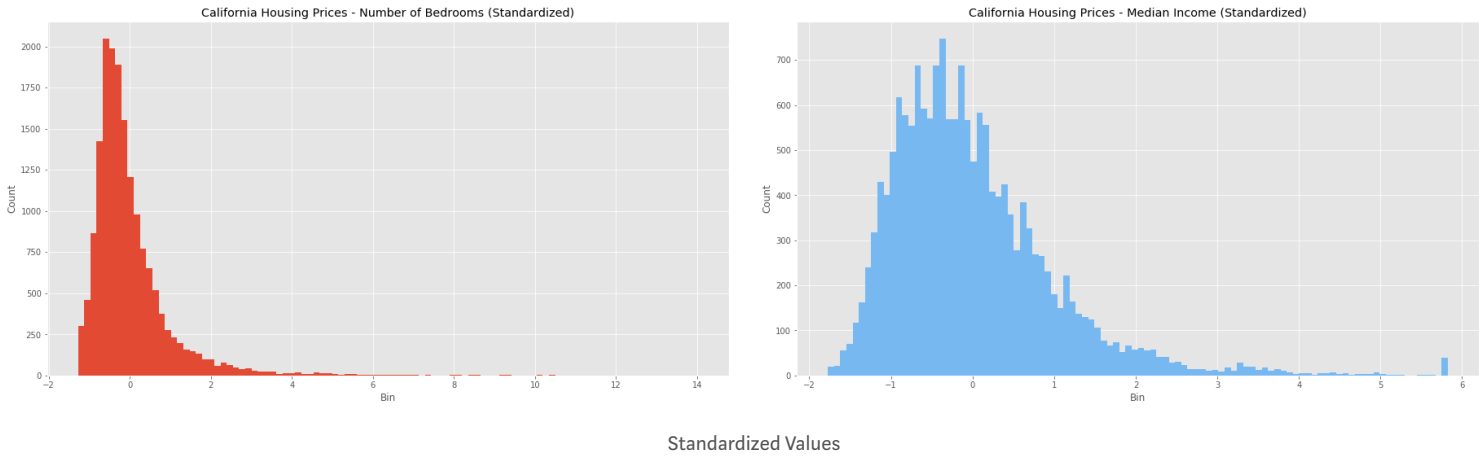
Let see it in python:

```
from sklearn import preprocessing

# Get column names first
names = df.columns

# Create the Scaler object
scaler = preprocessing.StandardScaler()

# Fit your data on the scaler object
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=names)
```



Looks like we've adjusted for all the outlier values in `bedrooms` and `income`, and we have a much more normal distribution for each feature. It's not perfect, but the data is in much better shape than it was when we ran our `normalization`. It seems because of the large difference in scales and units, `standardizing` is a better transformation for this data set.

> *2. Standardizing tends to make the training process well behaved because the numerical condition of the optimization problems is improved.*

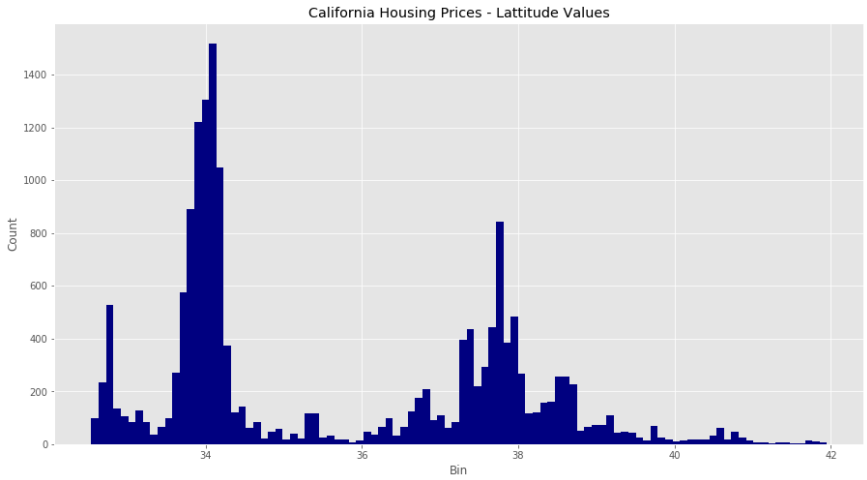
Consider if you're doing PCA, the output can only be interpreted correctly when the features have first been centred around their means. Again, understanding what you want to achieve and the model you'll be using, are necessary conditions to understanding different transformations decisions.

However, if you do standardize your data be warned you might be discarding some information. If that information is not needed, the process can be helpful else it will impede your results.

Bonus Thing: Binning

Let's take a look at one more thing before we leave … `binning` values.

Consider the `latitude` feature in our dataset, which has a geo point of the area in question. How would we think about `standardizing` or `normalizing` this? We could do either, but there is a third alternative: `binning`.



We're going to made new columns for each `latitude` range, and encode each value in our dataset with a `0 or 1` to see if it is within that `latitude` range.

```
# Create range for your new columns
lat_range = zip(xrange(32, 44), xrange(33, 45))

new_df = pd.DataFrame()

# Iterate and create new columns, with the 0 and 1 encoding
for r in lat_range
        new_df["latitude_%d_to_%d" % r] =
df["latitude"].apply(
            lambda l: 1.0 if l >= r[0] and l < r[1] else
0.0)

new_df
```

	latitude_32_to_33	latitude_33_to_34	latitude_34_to_35	latitude_35_to_36	latitude_36_to_37	latitude_37_to_38	latitude_38_to_39	latitude_39_to_40	latitude_
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	
16995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

Values 0 and 1, are between 34 and 35. Values 2, 3, and 4, are between 33 and 34.

Now that we can `binned` values, we have a binary value for each `latitude` in California. With this additional approach, you have another way to clean your data and get it ready for modelling.

As always, I hoped this cleared up a few things and provided some concrete examples for you to work on.

Cheers

Additional Reading:

What's the difference between Normalization and Standardization?

At work we were discussing this as my boss has never heard of normalization. In Linear Algebra,...

stats.stackexchange.com

http://www.dataminingblog.com/standardization-vs-normalization/

https://www.quora.com/What-is-the-difference-between-normalization-standardization-and-regularization-for-data

http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-16.html