. Cleaning and improving your data

Now that we've covered some simple visualizations and model fitting with continuous timeseries, let's see what happens when we look at more real-world data.

2. Data is messy

Real-world data is always messy, and requires preparing and cleaning the data before fitting models. In timeseries, messy data often happens due to failing sensors or human error in logging the data. Let's cover some specific ways to spot and fix messy data with timeseries.

3. What messy data looks like

First, let's look at some messy-looking data. Here, we're showing the value of the company AIG over the last several years. There seem to be two periods of time where no data was produced, as well as some periods of time where the data doesn't fluctuate at all. Both look like they're aberrations, so let's see how we can correct for them. Before moving forward, note that it is not always clear whether patterns in the data are "aberrations" or not. You should always investigate to understand the source of strange patterns in the data.

4. Interpolation: using time to fill in missing data

First, let's fill in the missing data using other datapoints we do have. We'll use a technique called interpolation, which uses the values on either end of a missing window of time to infer what's in-between.

5. Interpolation in Pandas

In this example, we'll first create a boolean mask that we'll use to mark where the missing values are. Next, we call the dot-interpolate method to fill in the missing values. We'll use the first argument to signal we want linear interpolation. Finally, we'll plot the interpolated values.

6. Visualizing the interpolated data

You can see the results of interpolation in red. In this case, we used the "linear" argument so the interpolated values are a line between the start and stop point of the missing window. Other arguments to the dot-interpolate method will result in different behavior.

7. Using a rolling window to transform data

Another common technique to clean data is transforming it so that it is more well-behaved. To do this, we'll use the same rolling window technique covered in Chapter 2.

8. Transforming data to standardize variance

Using a rolling window, we'll calculate each timepoint's percent change over the mean of a window of previous timepoints. This standardizes the variance of our data and reduces long-term drift.

9. Transforming to percent change with Pandas

In this function, we first separate out the final value of the input array. Then, we calculate the mean of all but the last datapoint. Finally, we subtract the mean from the final datapoint, and divide by the mean. The result is the percent change for the final value.

10. Applying this to our data

We can apply this to our data using the dot-aggregate method, passing our function as an input. On the right, the data is now roughly centered at zero, and periods of high and low changes are easier to spot.

11. Finding outliers in your data

We'll use this transformation to detect outliers. Outliers are datapoints that are statistically different from the dataset as a whole. A common definition is any datapoint that is more than three standard deviations away from the mean of the dataset.

12. Plotting a threshold on our data

Here we'll visualize our definition of an outlier. We calculate the mean and standard deviation of each dataset, then plot outlier "thresholds" (three times the standard deviation from the mean) on the raw and transformed data.

13. Visualizing outlier thresholds

Here is the result. Any datapoint outside these bounds could be an outlier. Note that the datapoints deemed an outlier depend on the transformation of the data. On the right, we see a few outlier datapoints that were *not* outliers in the raw data.

14. Replacing outliers using the threshold

Next, we replace outliers with the median of the remaining values. We first center the data by subtracting its mean, and calculate the standard deviation. Finally, we calculate the absolute value of each datapoint, and mark any that lie outside of three standard deviations from the mean. We then replace these using the nanmedian function, which calculates the median without being hindered by missing values.

15. Visualize the results

As you can see, once we've replaced the outliers, there don't seem to be as many extreme datapoints. This should help our model find the patterns we want.