1. Creating features over time

In the final lesson of this chapter, we'll cover some specific features that are useful in timeseries analysis.

2. Extracting features with windows

Remember the rolling window used earlier to smooth our data? We can use the same technique to extract features as they change over time. In this image, we can define multiple functions of each window to extract many features at once.

3. Using .aggregate for feature extraction

In pandas, the dot-aggregate method can be used to calculate many features of a window at once. By passing a list of functions to the method, each function will be called on the window, and collected in the output. Here's an example - we first use the dot-rolling method to define a rolling window, then pass a list of two functions (for the standard deviation, and maximum value). This extracts two features for each column over time.

4. Check the properties of your features!

You can extract many different kinds of features this way. Always plot the features you've extracted over time, as this can give you a clue for how they behave and help you spot noisy data and outliers. Here we can see that the maximum value is much jumpier than the mean.

5. Using partial() in Python

A useful tool when using the dot-aggregate method is the partial function. This is built-in to Python, and lets you create a *new* function from an old one, with some of the parameters pre-configured. Let's see how this works. In this example, we first import partial from functools, then use it to create a mean function that always operates on the first axis. The first argument is the function we want to modify, and subsequent key-value pairs will be pre-set in the output function. After this, we no longer need to configure those values when we call the new function.

6. Percentiles summarize your data

Now, back to feature extraction. A particularly useful tool for feature extraction is the percentile function. This is similar to calculating the mean or median of your data, but it gives you more fine-grained control over what is extracted. The percentile function takes an array as the first input, and an integer between 0 and 100 as the second input. It will return the value in the input array that matches the percentile you've chosen. Here it returns 40, which means that the value "40" is larger than 20% of the input array.

7. Combining np.percentile() with partial functions to calculate a range of percentiles

Here we'll combine the percentile function with partial functions in order to extract several percentiles with the dot-aggregate method. We use a list comprehension to create a list of functions (called percentile_funcs). Then, we loop through the list, calling each function on our data, to return a different percentile of the data. We could pass this list of partial functions to our dot-aggregate method to extract several percentiles for each column.

8. Calculating "date-based" features

Another common feature to consider are "date-based" features. That is, features that take into consideration information like "what time of the year is it?" or "is it a holiday?". Since many datasets involve humans, these pieces of information are often important. For example, if you're trying to predict the number of customers that will visit your store each day, it's important to know if it's the weekend or not! Working with dates and times is straightforward in Pandas, which we'll cover next.

9. datetime features using Pandas

Datetime functionality is most commonly accessed with a DataFrame's index. As you saw in the first chapter, you can use the to_datetime function to ensure dates are treated as datetime objects.

You can also extract many date-specific pieces of information, such as the day of the week, or weekday name as shown here. These could then be treated as features in your model.