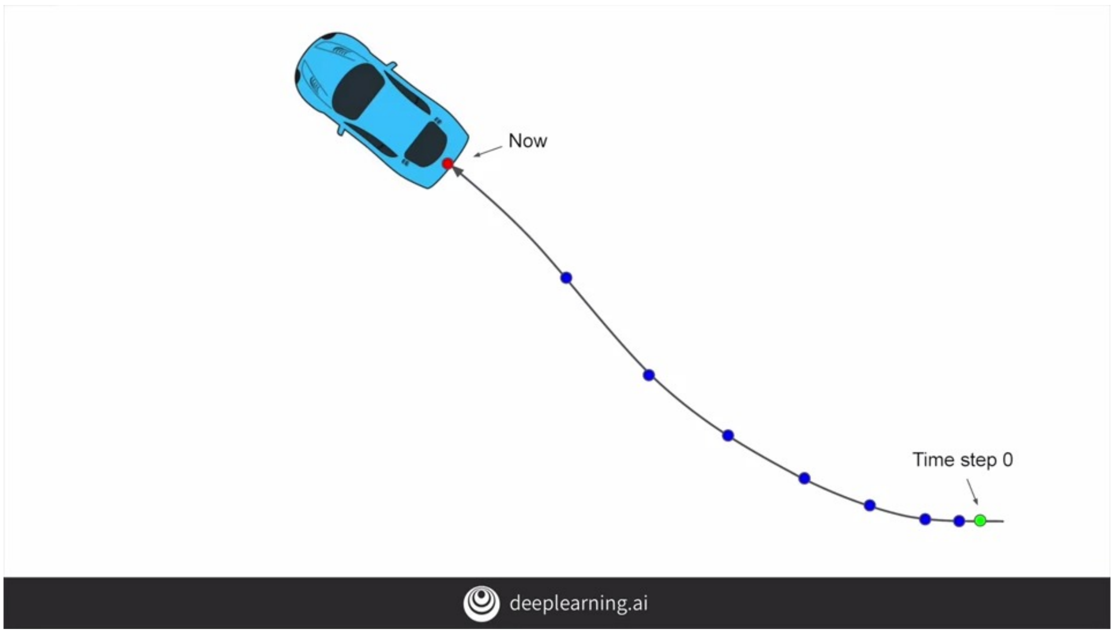
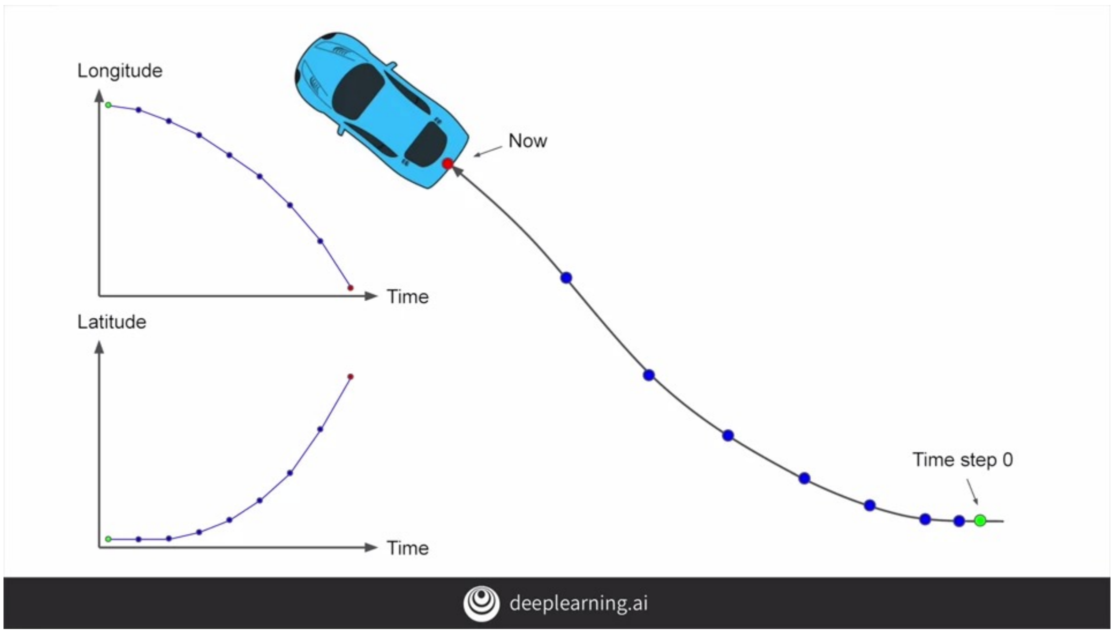
1. Time series examples

Welcome to this course on sequences and prediction, a part of the TensorFlow in practice specialization. In this course, we'll focus on time series, where you'll learn about different types of time series before we go deeper into using time series data. This week, you'll focus on time series themselves. We'll go through some examples of different types of time series, as well as looking at basic forecasting around them. You'll also start preparing time series data for machine learning algorithms. For example, how do you split time series data into training, validation, and testing sets? We'll explore some best practices and tools around that to get you ready for week 2, where you'll start looking at forecasting using a dense model, and how it differs from more naive predictions based on simple numerical analysis of the data. In week 3, we'll get into using recurrent neural networks to forecast time series. We'll see the stateless and stateful approaches, training on windows of data, and you'll also get hands-on in forecasting for yourself. Finally, in week 4, you'll add convolutions to the mix and put everything you've worked on together to start forecasting some real world data, and that's measurements of sunspot activity over the last 250 years.
So let's get started with a look at time series, what they are, and the different types of them that you may encounter. Time series are everywhere. You may have seen them in stock prices, weather forecasts, historical trends, such as Moore's law. Here, I've plotted the number of transistors per square millimeter, where I grouped chip releases by year, and then drew the Moore's law trend line from the first data item, where you can see a correlation. If you want some really fun correlations, here's one from Tyler Vigen's site of Spurious Correlations. This one is a time series correlation of total revenue generated by video game arcades versus computer science doctorates awarded in the United States. While all of these are quite familiar, they begged the question, what exactly is a time series? It's typically defined as an ordered sequence of values that are usually equally spaced over time. So for example, every year in my Moore's law charts or every day in the weather forecast. In each of these examples, there is a single value at each time step, and as a results, the term univariate is used to describe them. You may also encounter time series that have multiple values at each time step. As you might expect, they're called Multivariate Time Series. Multivariate Time Series charts can be useful ways of understanding the impact of related data. For example, consider this chart of births versus deaths in Japan from 1950 to 2008. It clearly shows the two converging, but then deaths begin to outstrip births leading to a population decline. Now, while they could be treated as two separate univariate time series, the real value of the data becomes apparent when we show them together as a multivariate. Also consider this chart showing the global temperature against CO2 concentration. As univariates, they would show a trend, but when combined, the correlation is very easy to see adding further value to the data.



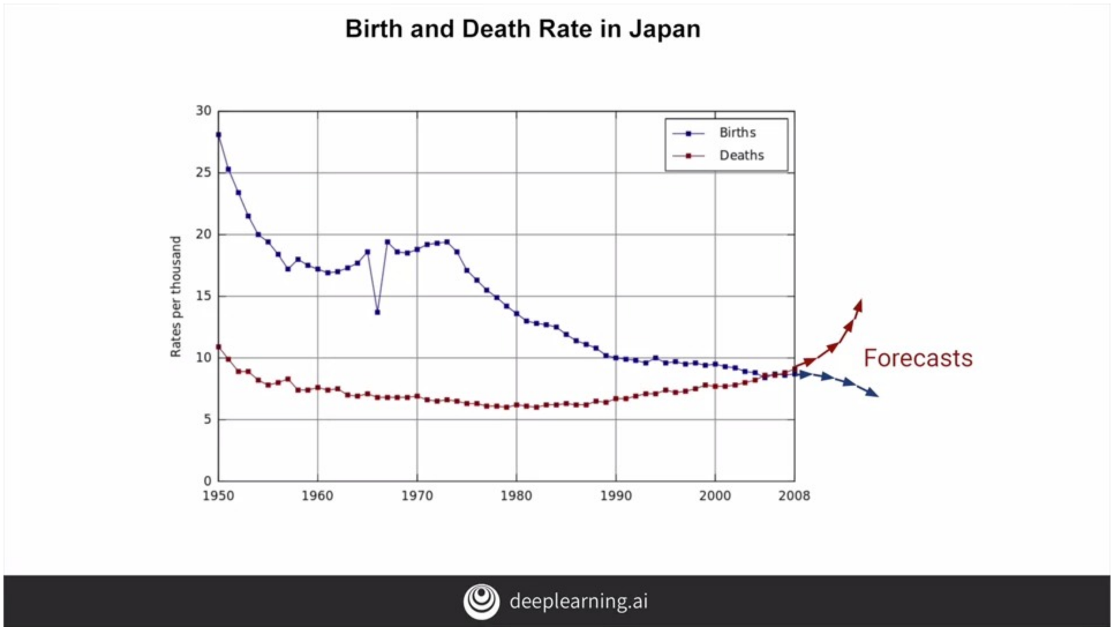
Movement of a body can also be plotted as a series of univariates or as a combined multivariate. Consider, for example, the path of a car as it travels. A time step zero is at a particular latitude and longitude. As subsequent time steps, these values changed based on the path of the car. The acceleration of the car, in other words, it's not moving at a constant speed, means that the spaces between the time steps also change in size, in this case getting larger. But what if we were to plot the direction of the car as univariates.



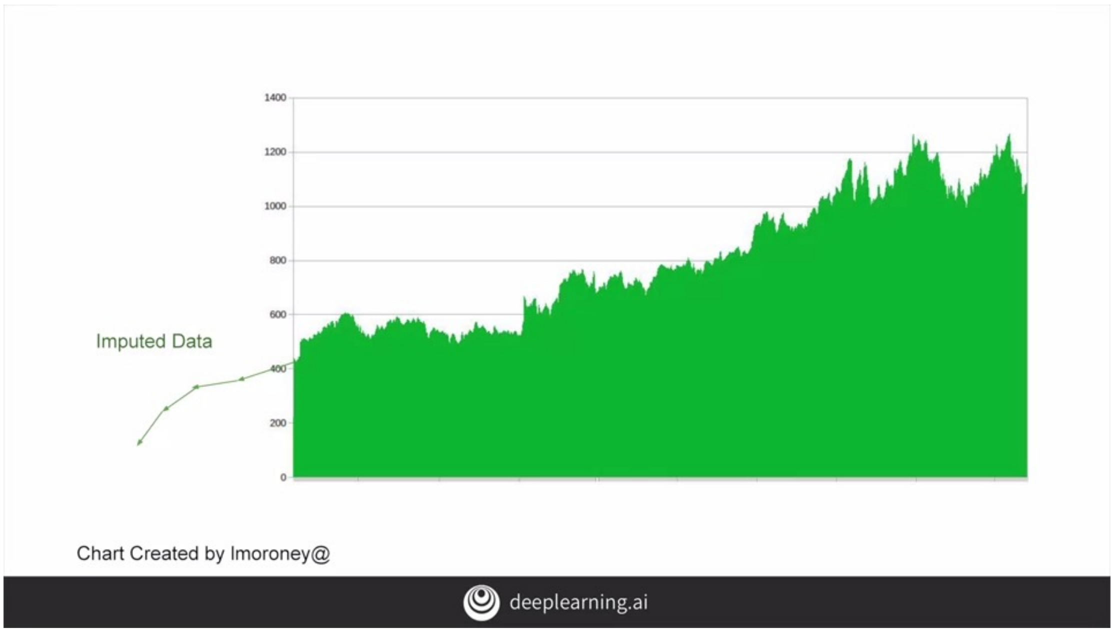
Based on its heading, we could see that the longitude of the car decreases over time, but its latitude increases, and as such you will get charts like these.

2. Machine learning applied to time series

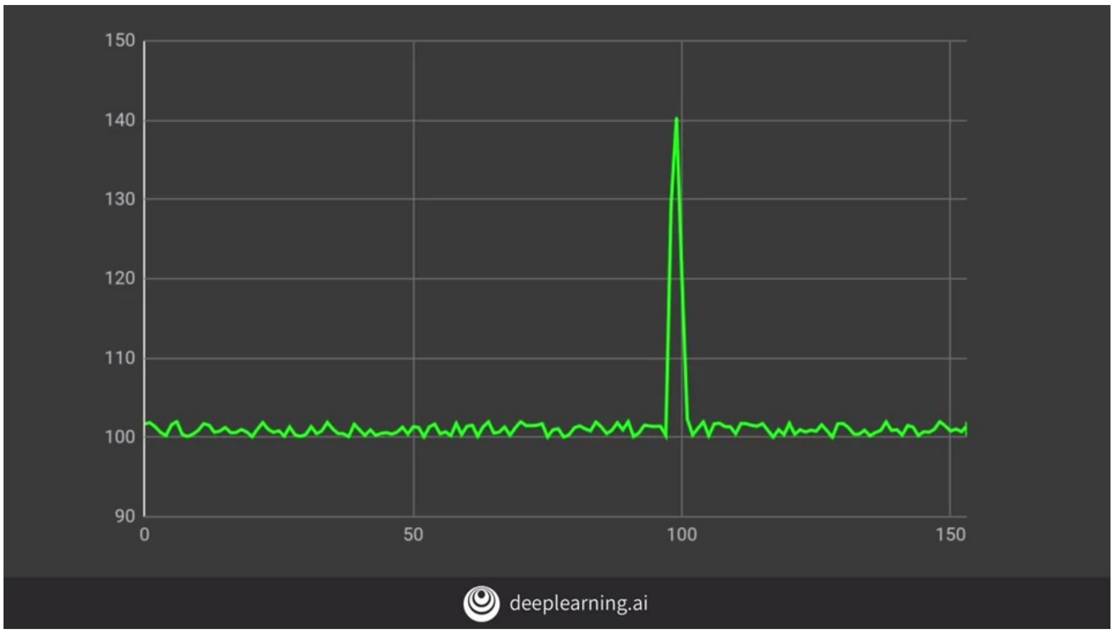
Now these are just a few examples of the types of things that can be analyzed using time series. And just about **anything that has a time factor** in it can be analyzed in this way. So what types of things can we do with machine learning over time series?



The first and most obvious is prediction of forecasting based on the data. So for example with the birth and death rate chart for Japan that we showed earlier. It would be very useful to predict future values so that government agencies can plan for retirement, immigration and other societal impacts of these trends.



In some cases, you might also want to project back into the past to see how we got to where we are now. This process is called **imputation**. Now maybe you want to get an idea for what the data would have been like had you been able to collect it before the data you already have. Or you might simply want to fill in holes in your data for what data doesn't already exist. For example, in our Moore's law chart from earlier. There was no data for some years because there were no chips released in those years, and you can see the gaps here. But with imputation, we can fill them up.

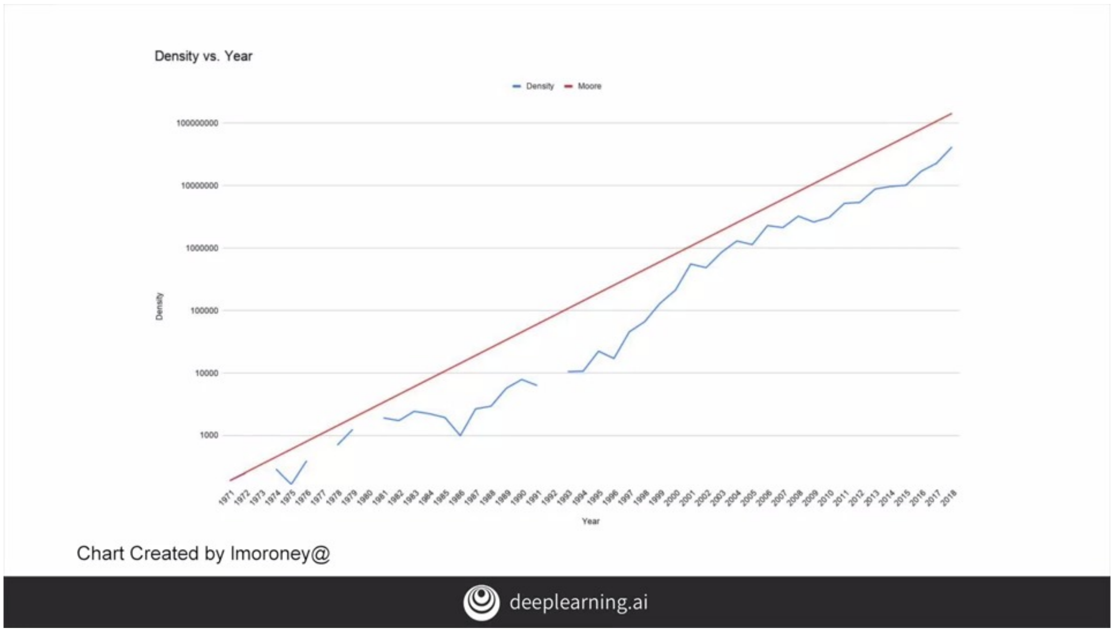


Additionally, time series prediction can be used to detect anomalies. For example, in website logs so that you could see potential denial of service attacks showing up as a spike on the time series like this.

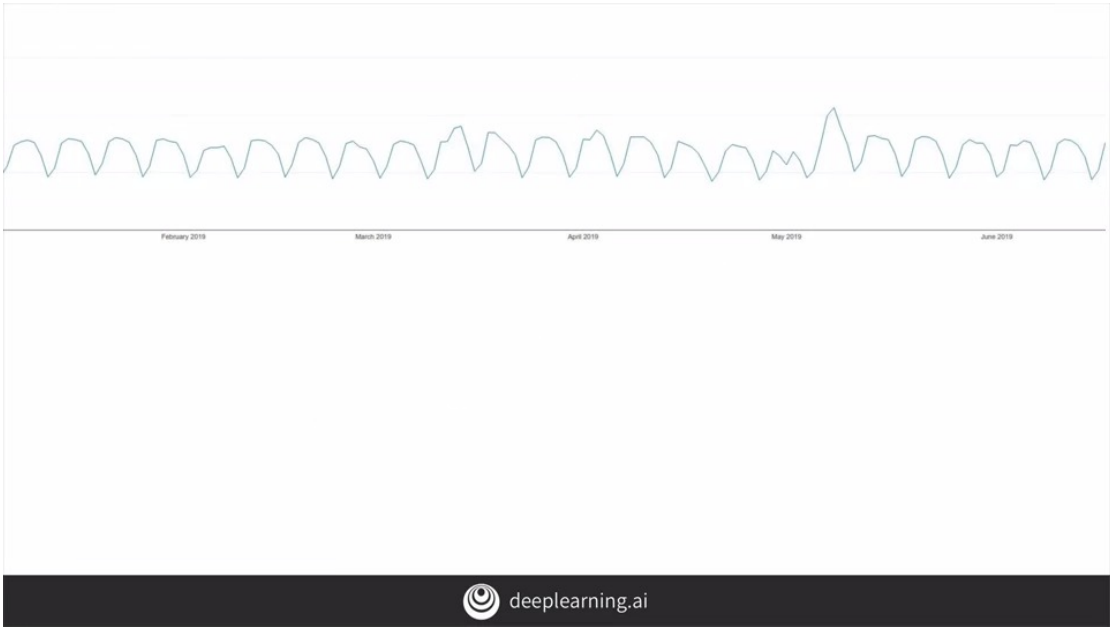
The other option is to analyze the time series to spot patterns in them that determine what generated the series itself. A classic example of this is to analyze sound waves to spot words in them which can be used as a neural network for speech recognition. Here for example, you can see how a sound wave is split into words. Using machine learning, it becomes possible to train a neural network based on the time series to recognize words or sub words.

3. Common patterns in time series

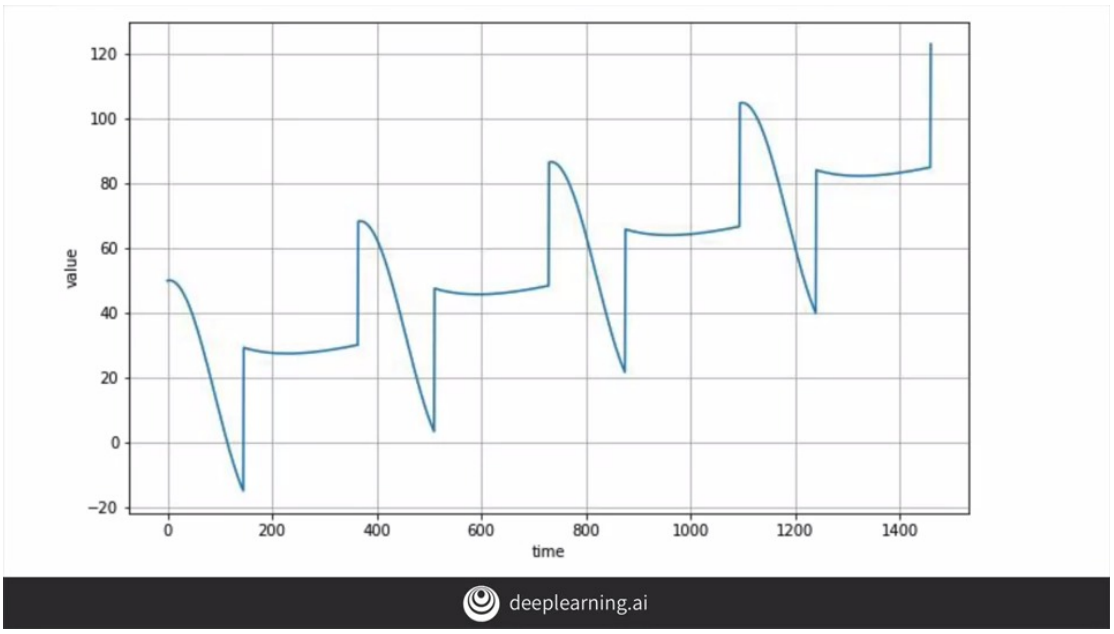
Time-series come in all shapes and sizes, but there are a number of very common patterns. So it's useful to recognize them when you see them. For the next few minutes we'll take a look at some examples.



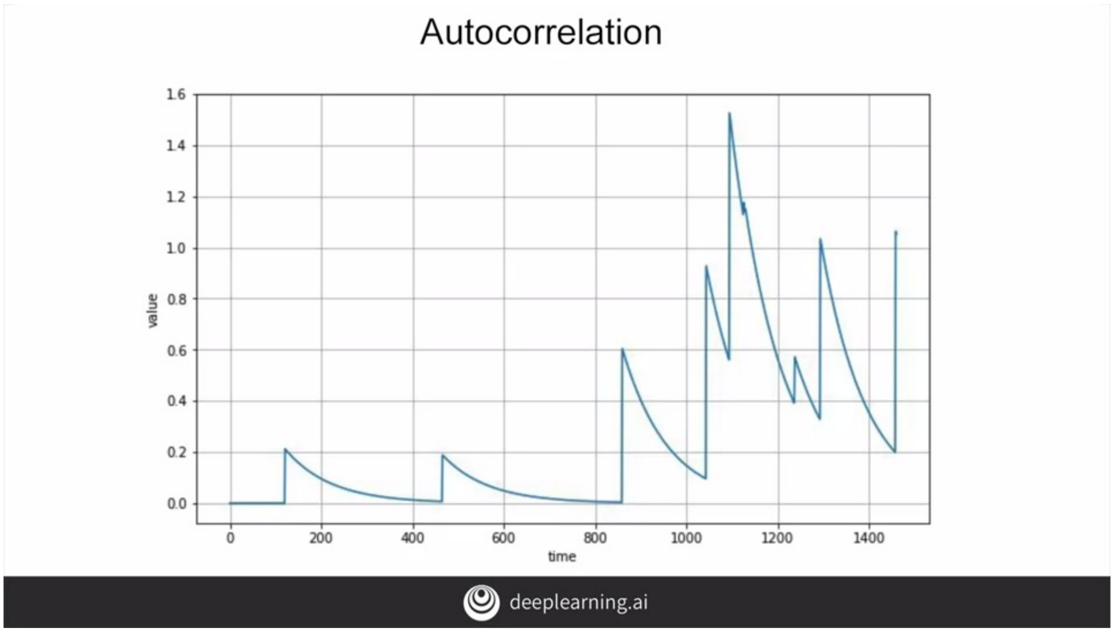
The first is **trend**, where time series have a specific direction that they're moving in. As you can see from the Moore's Law example we showed earlier, this is an upwards facing trend.



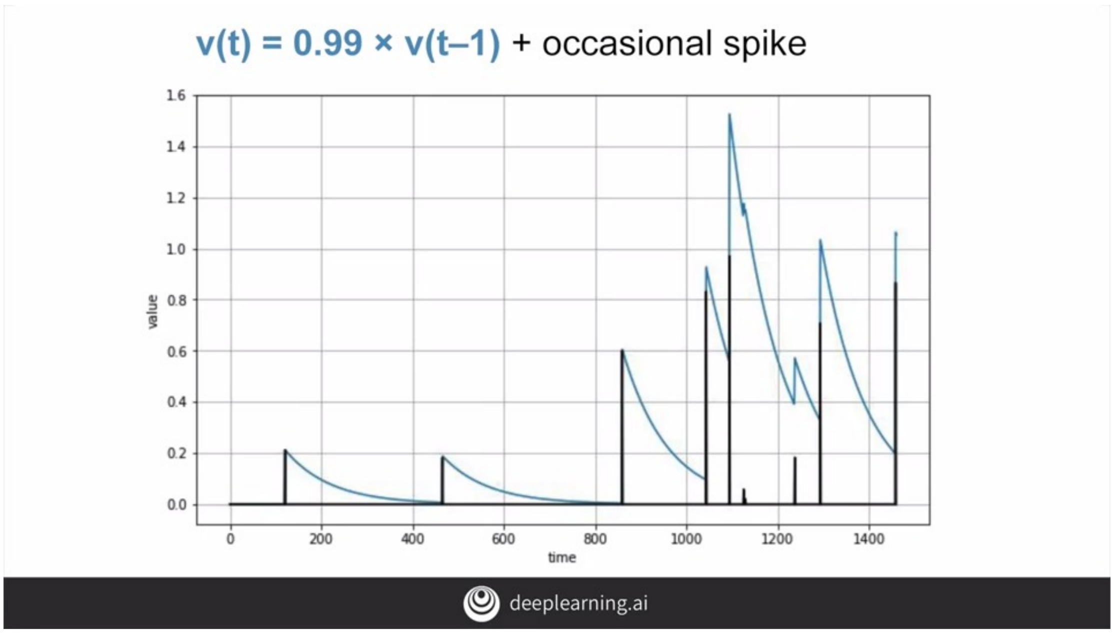
Another concept is **seasonality**, which is seen when patterns repeat at predictable intervals. For example, take a look at this chart showing active users at a website for software developers. It follows a very distinct pattern of regular dips. Can you guess what they are? Well, what if I told you if it was up for five units and then down for two? Then you could tell that it very clearly dips on the weekends when less people are working and thus it shows seasonality. Other seasonal series could be shopping sites that peak on weekends or sport sites that peak at various times throughout the year, like the draft or opening day, the All-Star day playoffs and maybe the championship game.



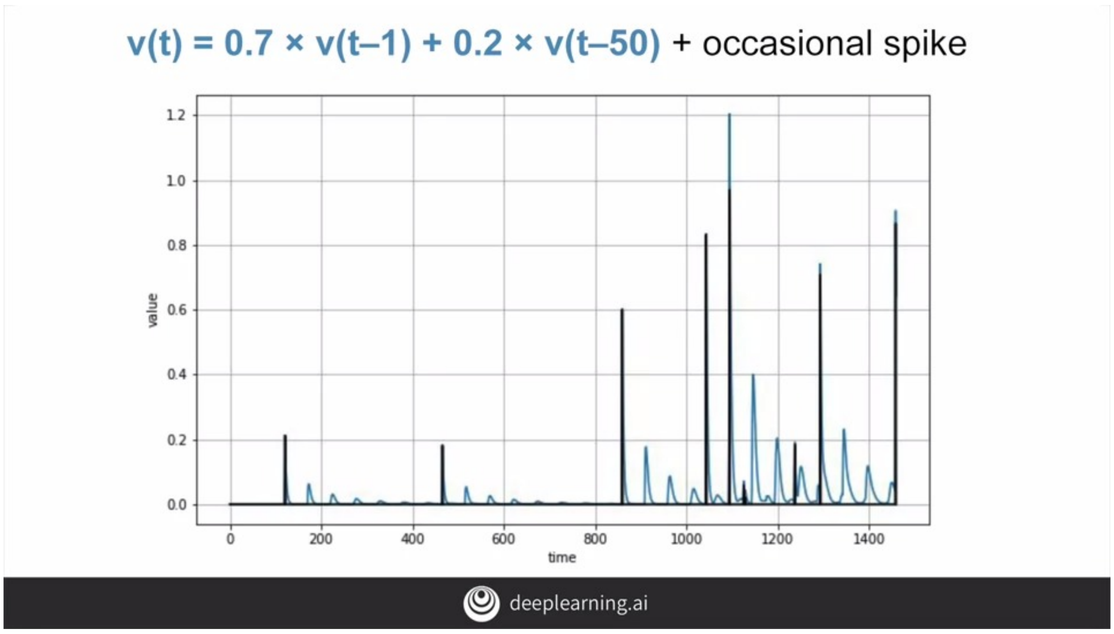
Of course, some time series can have **a combination of both trend and seasonality** as this chart shows. There's an overall upwards trend but there are local peaks and troughs.
But of course, there are also some that are probably **not predictable at all** and just a complete set of random values producing what's typically called white noise. There's not a whole lot you can do with this type of data.



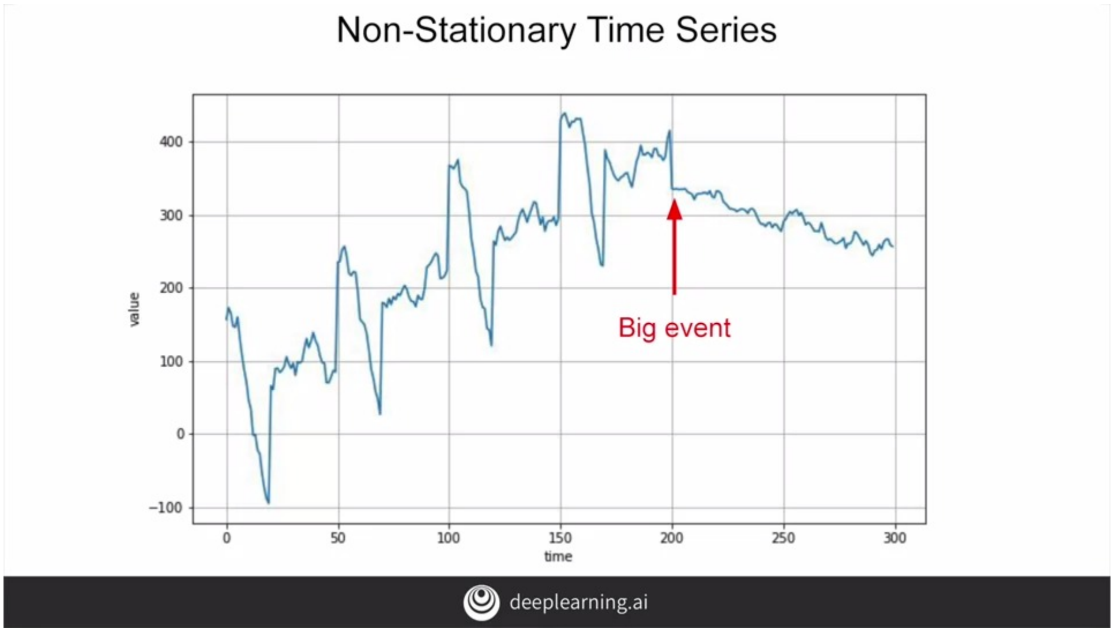
But then consider this time series. There's no trend and there's no seasonality. The spikes appear at random timestamps. You can't predict when that will happen next or how strong they will be. But clearly, the entire series isn't random. Between the spikes there's **a very deterministic type of decay**.



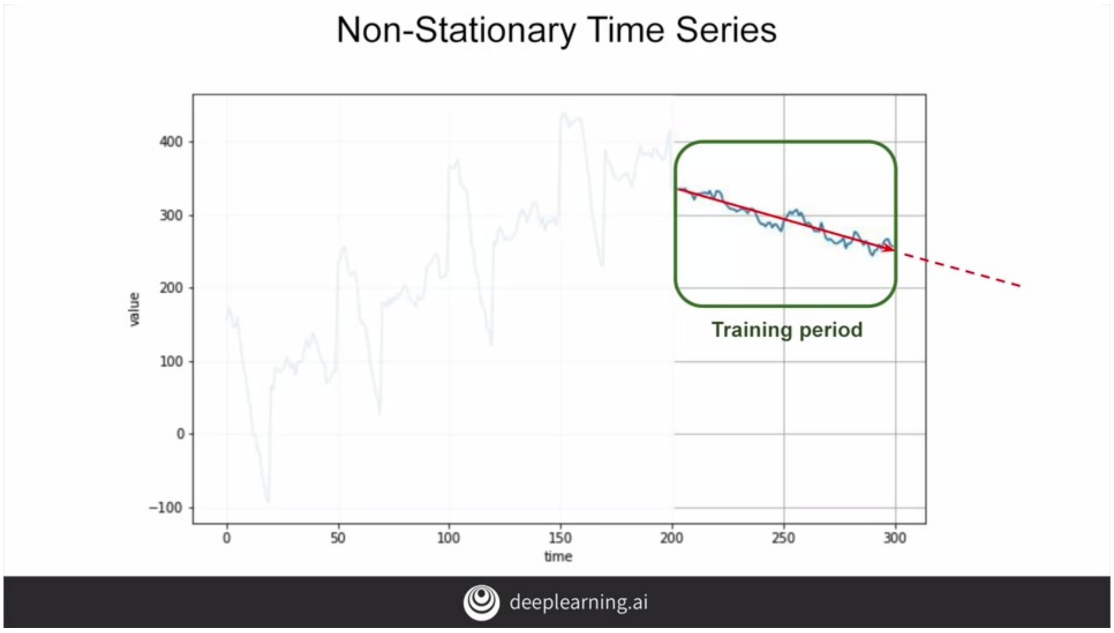
We can see here that the value of each time step is 99 percent of the value of the previous time step plus an occasional spike. This is an **auto correlated time series**. Namely it correlates with **a delayed copy of itself often called a lag**. This example you can see at lag one there's a strong autocorrelation. Often a time series like this is described as having memory as steps are dependent on previous ones. The spikes which are unpredictable are often called **Innovations**. In other words, they cannot be predicted based on past values.



Another example is here where there are **multiple autocorrelations**, in this case, at time steps one and 50. The lag one autocorrelation gives these very quick short-term exponential delays, and the 50 gives the small balance after each spike. Time series you'll encounter in real life probably have a bit of each of these features: trend, seasonality, autocorrelation, and noise. As we've learned a machine-learning model is designed to spot patterns, and when we spot patterns we can make predictions. For the most part this can also work with time series except for the noise which is unpredictable. But we should recognize that this assumes that patterns that existed in the past will of course continue on into the future.

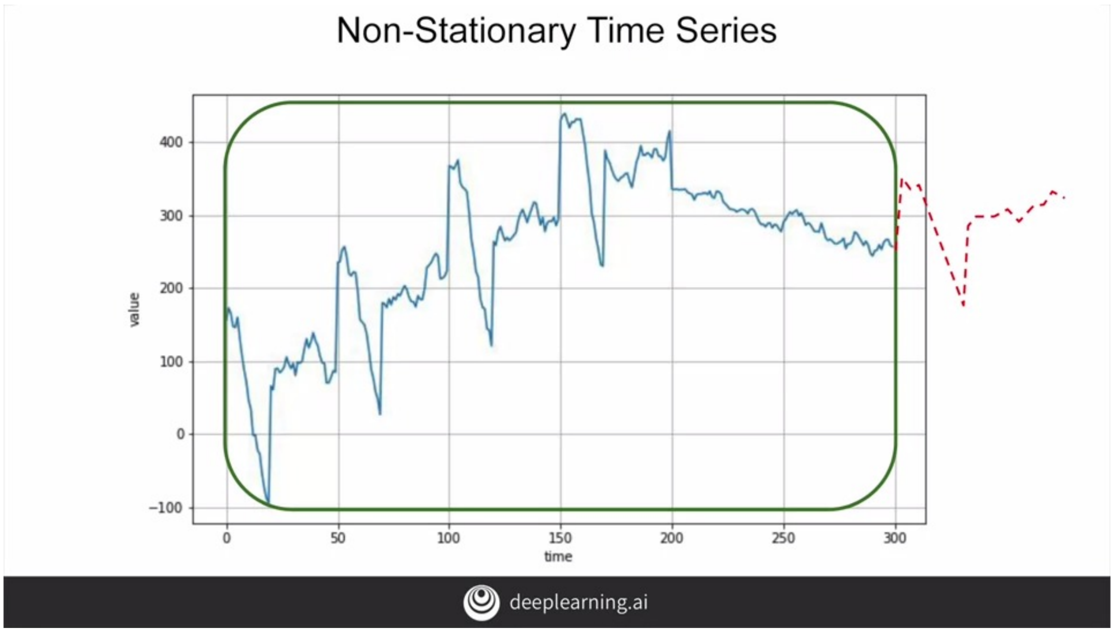


Of course, real life time series are not always that simple. Their behavior can change drastically over time. For example, this time series had a positive trend and a clear seasonality up to time step 200. But then something happened to change its behavior completely. If this were stock, price then maybe it was a big financial crisis or a big scandal or perhaps a disruptive technological breakthrough causing a massive change. After that the time series started to trend downward without any clear seasonality. We'll typically call this a non-stationary time series.



To predict on this we could just train for limited period of time. For example, here where I take just the last 100 steps. You'll probably get a better performance than if you had trained on the entire time series. **But that's breaking the mold for typical machine, learning where we**

always assume that more data is better. But for time series forecasting it really depends on the time series. If it's stationary, meaning its behavior does not change over time, then great. The more data you have the better. But if it's not stationary then the optimal time window that you should use for training will vary. Ideally, we would like to be able to take the whole series into account and generate a prediction for what might happen next.



As you can see, this isn't always as simple as you might think given a drastic change like the one we see here. So that's some of what you're going to be looking at in this course. But let's start by going through a workbook that generates sequences like those you saw in this video. After that we'll then try to predict some of these synthesized sequences as a practice before later we'll move on to real-world data.

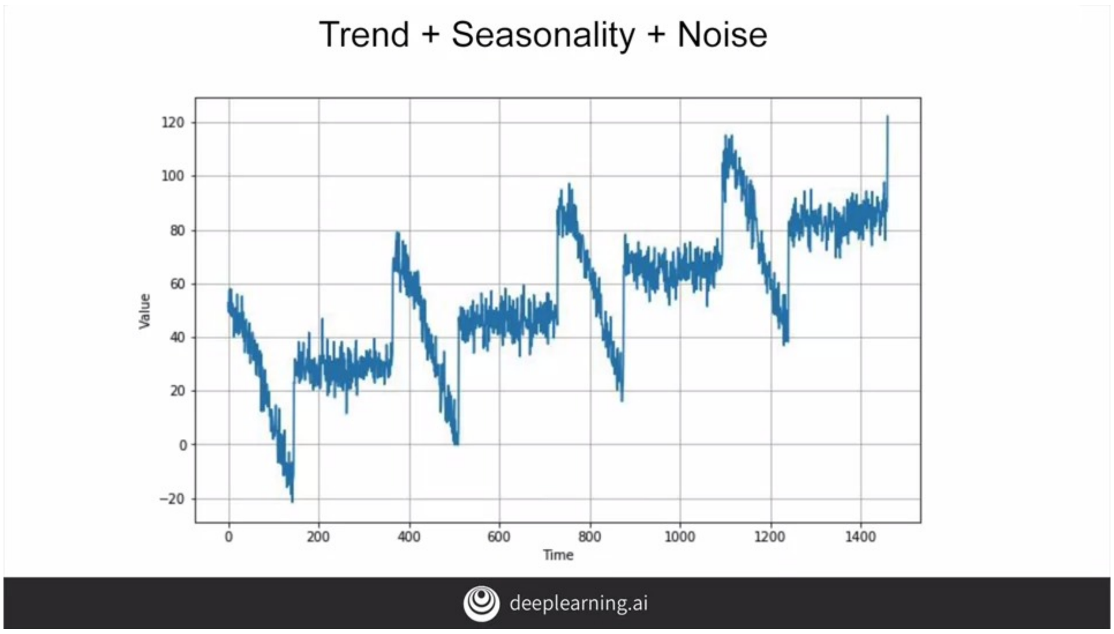
S+P Week 1 - Lesson 1 – Notebook

Let's take a look at time series and the various attributes of time series using Python. This notebook is available as part of the course and I'll provide a link to it. I recommend that you watch this video first and then try the notebook for yourself afterwards. I'll start by running the nodes containing the imports as well as a couple of helper functions. One to plot the series and one to return a trend. So now, let's plot our first very simple time series. Even though it's a straight line, it's also an example of the time series. The x-axis in this case is time and the y value is the value of the function at that time. Next, we'll take a look at adding a seasonal pattern to our time series. These functions contain a seasonal pattern and then seasonality that just uses the same pattern. We'll now plot that. As we investigate the graph, we can see clear peaks and troughs. But in addition to that, there are smaller, regular spikes. This could be seen as a rough simulation of a seasonal value. For example, maybe profits for shop that are negative on the day the store is closed, peaking a little the day after, decaying during the week and then peaking again on the weekend. What if we now add a trend to this so that the seasonal data while still following the pattern increases over time? Maybe simulating a growing business so when we plot it, we'll see the same pattern but with an overall upward trend. What if we now add another feature that's common in time series, noise? Here's a function that add some noise to a series and when we call that and plot the results and their

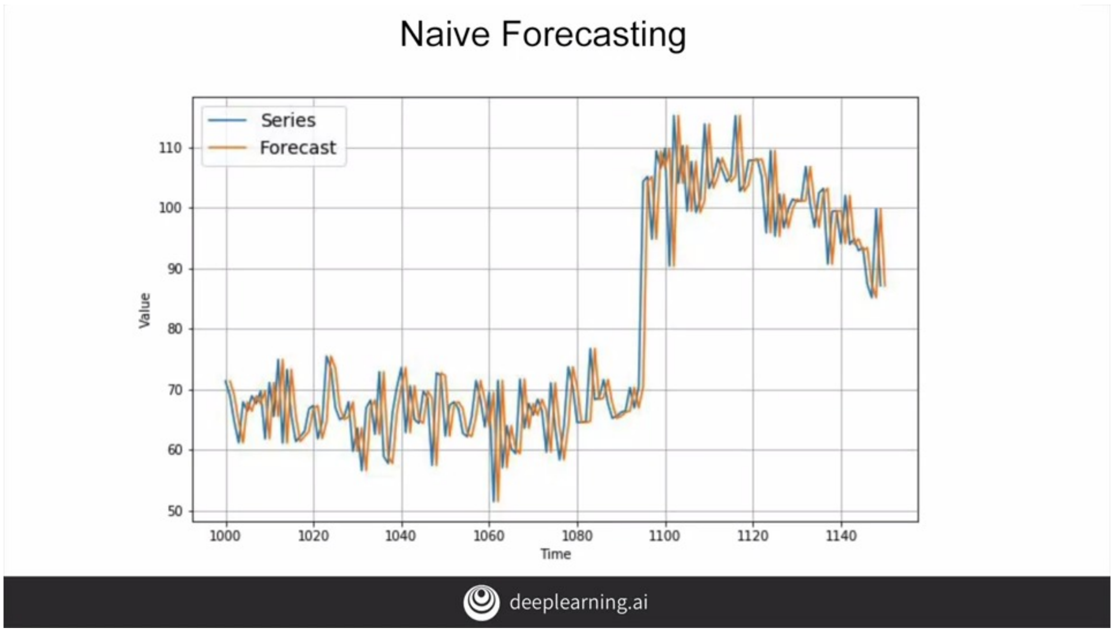
impact on our time series, we now get a very noisy series, but one which follows the same seasonality as we saw earlier. It's interesting because at this point, the human eye may miss a lot of the seasonality data but a computer will hopefully be able to spot it. Next we can explore a little bit of Autocorrelation, but first here are a couple of functions that can add it for you. Here is where we add the autocorrelation to the series and plot it. There are two different autocorrelation functions and I'll plot both so that you can see the effects of each. This one is particularly interesting because you can see the repeated pattern despite different scales. There is a pattern and then a sharp fall off followed by the same pattern on a smaller scale with the same fall off, which is then shrunk et cetera. If I change autocorrelation functions and run it again, we can then see the other function.
Okay. Let's add some noise.
Then we'll try another autocorrelation and another.
Now, let's add them to simulate a seasonal time series that has an impact full of events that changes everything. For example, that might be a financial series that shows seasonality, but then something changes like a failure of the business or big news events.
Now, I'm going to add some impulses and plot them. Nothing too exciting here yet.
But when I start adding some autocorrelations to this, then we'll see some of the behavior that we had discussed earlier where from our pulse we have a decay away from it but the decay could be interrupted by another pulse. This decay could be autocorrelated so that after the pulse it decays but then the decay autocorrelates. So we have these decreasing curves.
Hopefully this exploration of some synthetic data to show some of the attributes of time-series was helpful for you to understand some of the terminology. I have found that synthetic data like this is very useful if you want to learn how to use Machine Learning to understand and predict on data. In the next lesson, you'll take the first steps towards predicting the next values in a synthetic series before later in the course, you'll start applying what you've learned to real-world data.

4. Train, validation and test sets

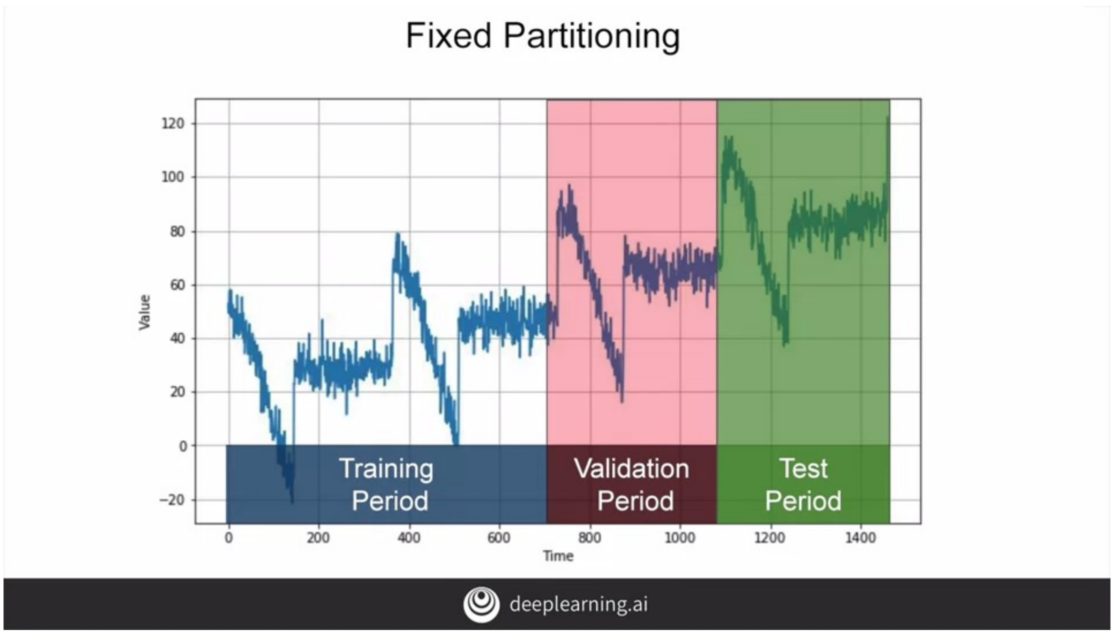
In the previous videos this week, you saw all of the different factors that make up the behavior of a time series. Now we'll start looking at techniques, given what we know, that can be used to then forecast that time series.



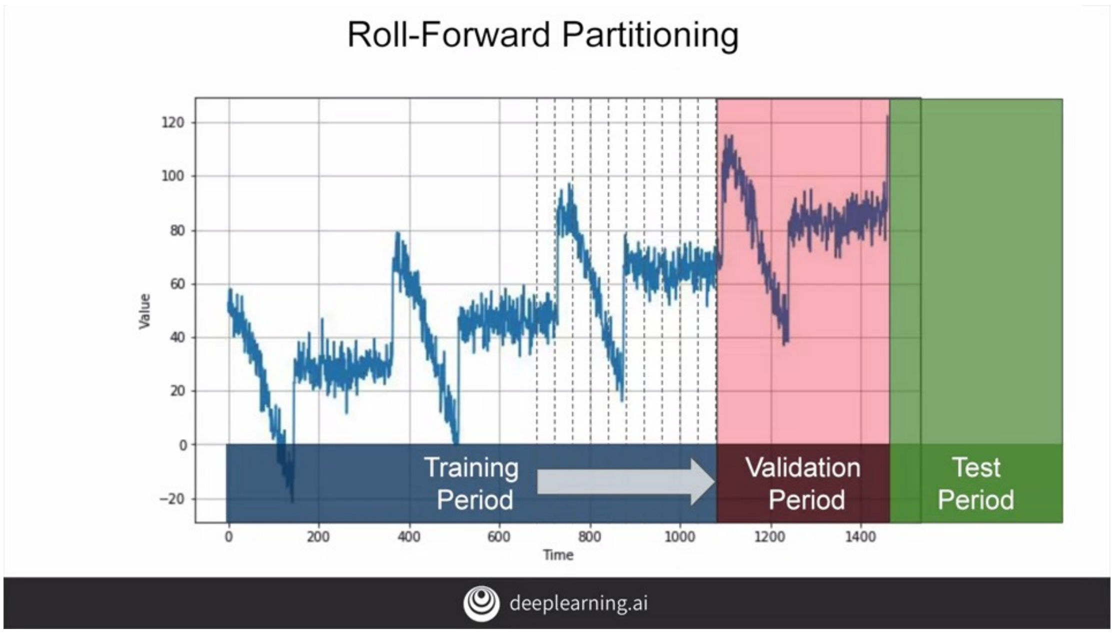
Let's start with this time series containing, trend seasonality, and noise and that's realistic enough for now.



We could, for example, take the last value and assume that the next value will be the same one, and this is called **naive forecasting**. I've zoomed into a part of the data set here to show that in action. We can do that to get a baseline at the very least, and believe it or not, that baseline can be pretty good. But how do you measure performance?

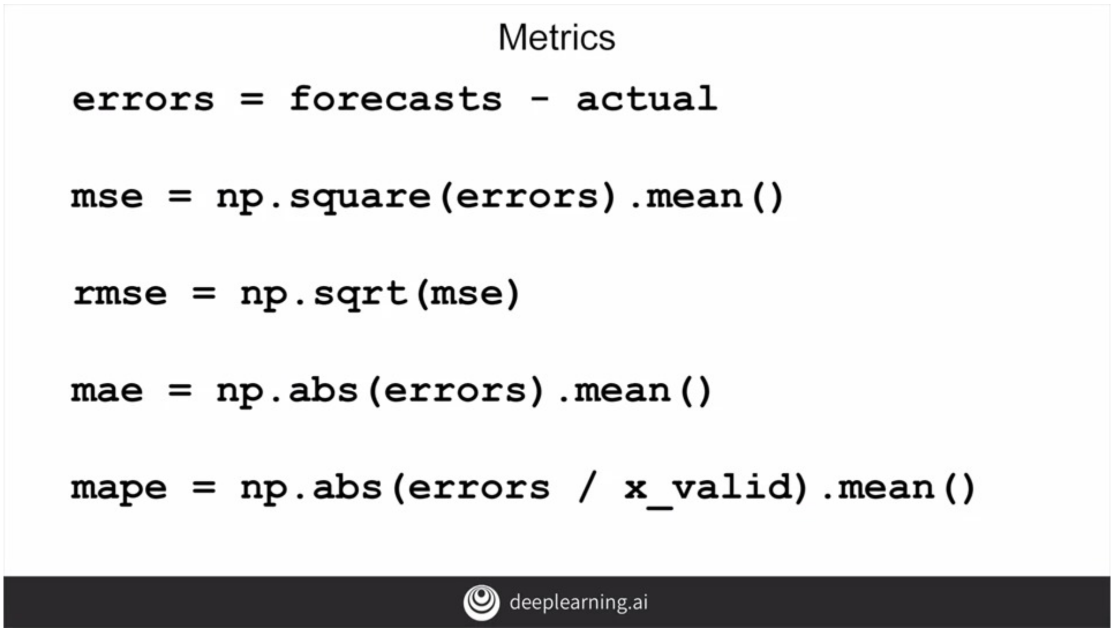


To measure the performance of our forecasting model, We typically want to split the time series into a training period, a validation period and a test period. This is called **fixed partitioning**. If the time series has some seasonality, you generally want to ensure that each period contains a whole number of seasons. For example, one year, or two years, or three years, if the time series has a yearly seasonality. You generally don't want one year and a half, or else some months will be represented more than others. While this might appear a little different from the training validation test, that you might be familiar with from non-time series data sets. Where you just picked random values out of the corpus to make all three, you should see that the impact is effectively the same. Next you'll train your model on the training period, and you'll evaluate it on the validation period. Here's where you can experiment to find the right architecture for training. And work on it and your hyper parameters, until you get the desired performance, measured using the validation set. Often, once you've done that, you can retrain using both the training and validation data. And then test on the test period to see if your model will perform just as well.
And if it does, then you could take the unusual step of retraining again, using also the test data. But why would you do that? Well, it's because the test data is the closest data you have to the current point in time. And as such it's often the strongest signal in determining future values. If your model is not trained using that data, too, then it may not be optimal. Due to this, it's actually quite common to forgo a test set all together. And just train, using a training period and a validation period, and the test set is in the future. We'll follow some of that methodology in this course.



Fixed partitioning like this is very simple and very intuitive, but there's also another way. We start with a short training period, and we gradually increase it, say by one day at a time, or by one week at a time. At each iteration, we train the model on a training period. And we use it to forecast the following day, or the following week, in the validation period. And this is called **roll-forward partitioning**. You could see it as doing fixed partitioning a number of times, and then continually refining the model as such. For the purposes of learning time series prediction in this course, will learn the overall code for doing series prediction. Which you could then apply yourself to a roll-forward scenario, but our focus is going to be on fixed partitioning.

5. Metrics for evaluating performance

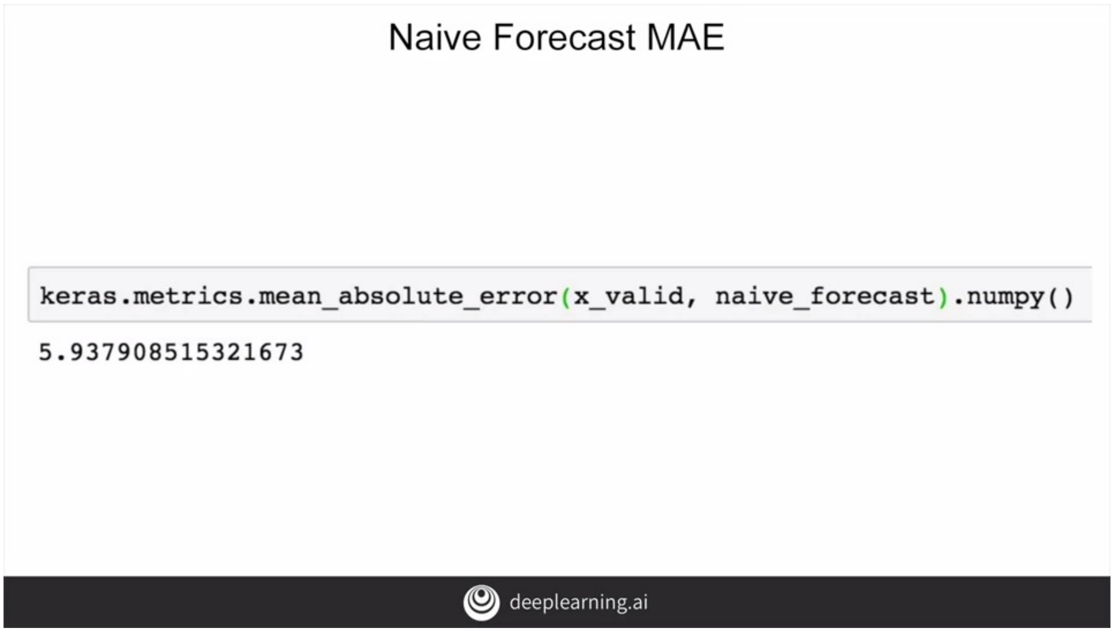


Metrics

```
errors = forecasts - actual

mse = np.square(errors).mean()

rmse = np.sqrt(mse)

mae = np.abs(errors).mean()

mape = np.abs(errors / x_valid).mean()
```

deeplearning.ai

Once we have a model and a period, then we can evaluate the model on it, and we'll need a metric to calculate their performance.
So let's start simply by calculating the errors, which is the difference between the forecasted values from our model and the actual values over the evaluation period.
The most common metric to evaluate the forecasting performance of a model is the mean squared error or mse where we square the errors and then calculate their mean. Well, why would we square it? Well, the reason for this is to get rid of negative values. So, for example, if our error was two above the value, then it will be two, but if it were two below the value, then it will be minus two. These errors could then effectively cancel each other out, which will be wrong because we have two errors and not none. But if we square the error of value before analyzing, then both of these errors would square to four, not canceling each other out and effectively being equal.
And if we want the mean of our errors' calculation to be of the same scale as the original errors, then we just get its square root, giving us a root means squared error or rmse. Another common metric and one of my favorites is the mean absolute error or mae, and it's also called the main absolute deviation or mad. And in this case, instead of squaring to get rid of negatives, it just uses their absolute value. This does not penalize large errors as much as the mse does. Depending on your task, you may prefer the mae or the mse. For example, if large errors are potentially dangerous and they cost you much more than smaller errors, then you may prefer the mse. But if your gain or your loss is just proportional to the size of the error, then the mae may be better.
Also, you can measure the mean absolute percentage error or mape, this is the mean ratio between the absolute error and the absolute value, this gives an idea of the size of the errors compared to the values.



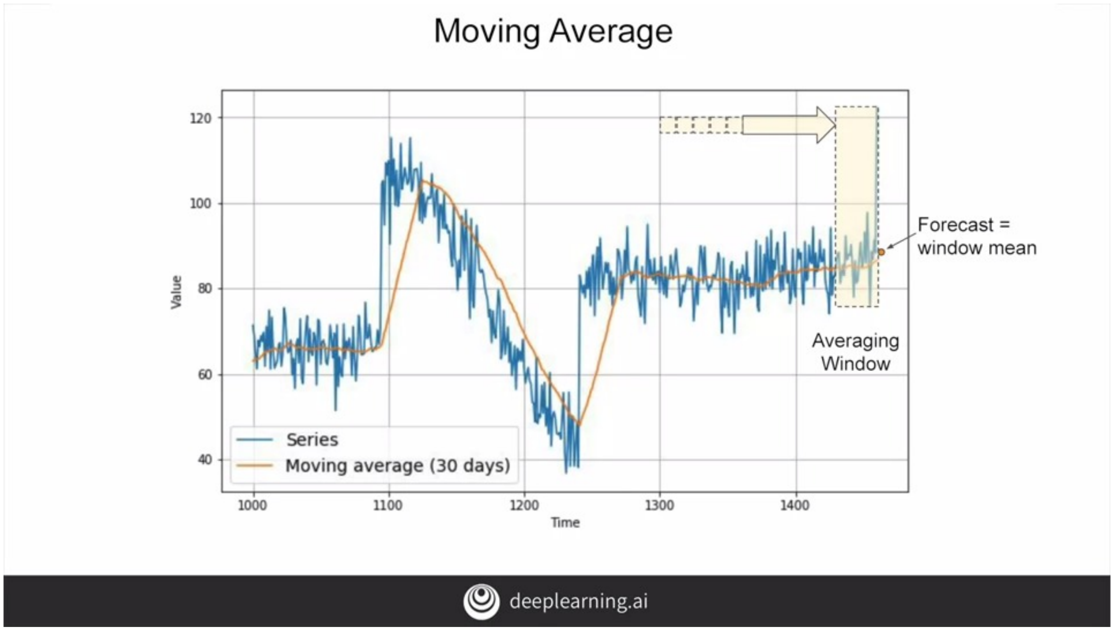
Naive Forecast MAE

```
keras.metrics.mean_absolute_error(x_valid, naive_forecast).numpy()
```

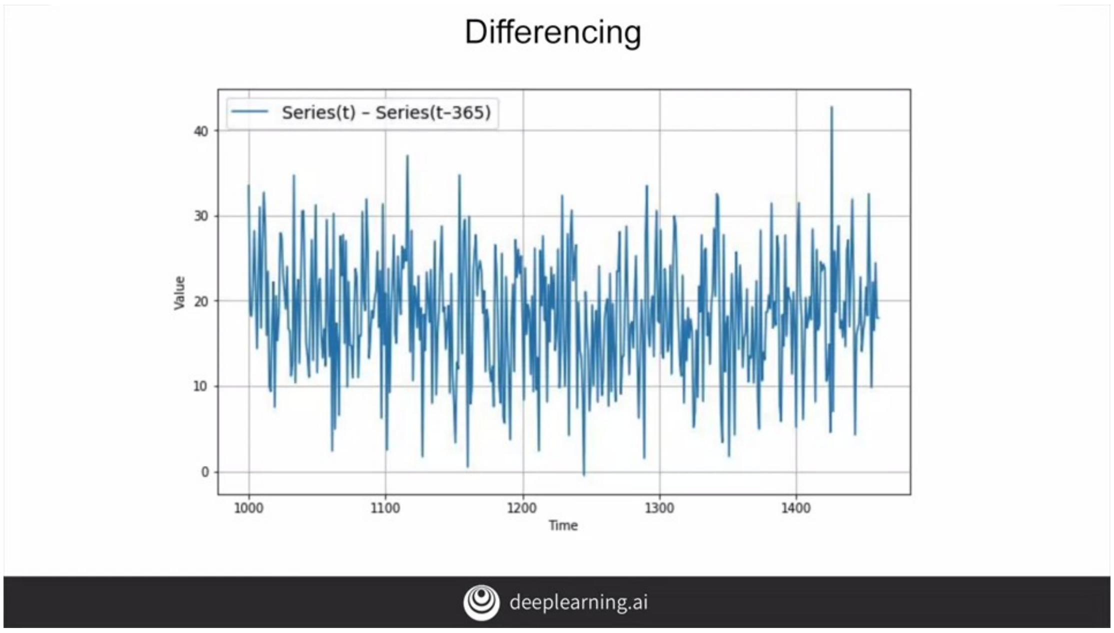
```
5.937908515321673
```

If we look at our data, we can measure the MAE using code like this.
The keras metrics libraries include an MAE that can be called like this. With the synthetic data we showed earlier, we're getting about 5.93, let's consider that our baseline.

6. Moving average and differencing

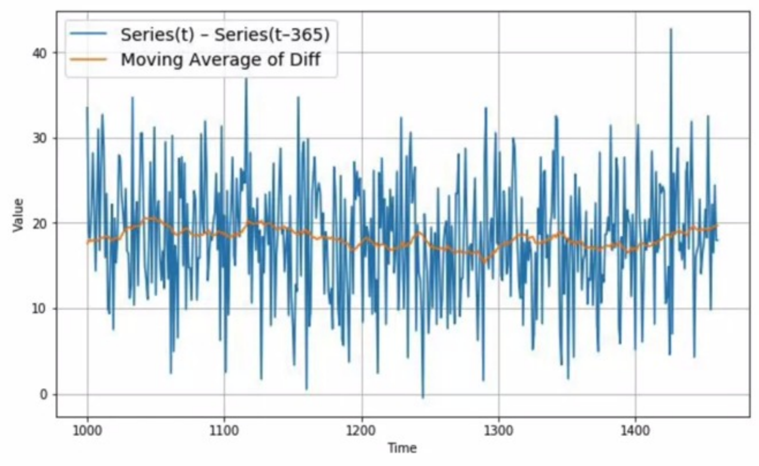


A common and very simple forecasting method is to **calculate a moving average**. The idea here is that the yellow line is a plot of the average of the blue values over a fixed period called an **averaging window**, for example, 30 days. Now this nicely eliminates a lot of the noise and it gives us a curve roughly emulating the original series, **but it does not anticipate trend or seasonality**.
Depending on the current time i.e. the period after which you want to forecast for the future, it can actually end up being worse than a naive forecast. In this case, for example, I got a mean absolute error of about 7.14.



One method to avoid this is to remove the trend and seasonality from the time series with a technique called **differencing**. So instead of studying the time series itself, we study the **difference between the value at time T and the value at an earlier period**. Depending on the time of your data, that period might be a year, a day, a month or whatever.

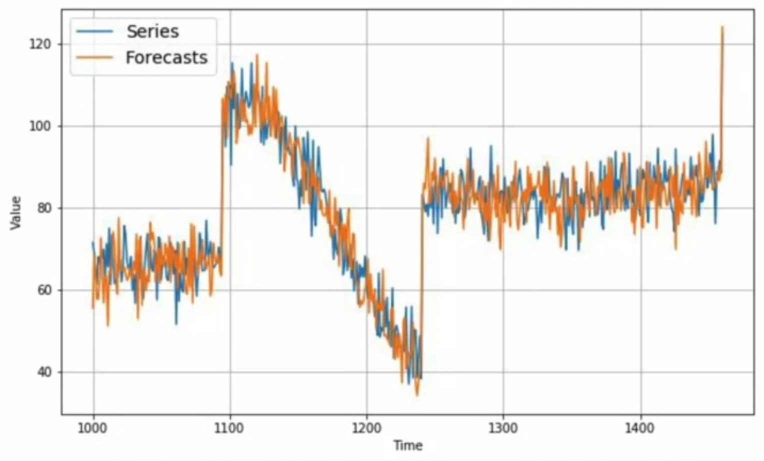
Moving Average on Differenced Time Series



deeplearning.ai

Let's look at a year earlier. So for this data, at time T minus 365, we'll get this difference time series which has no trend and no seasonality. We can then use a moving average to forecast this time series which gives us these forecasts. But these are just forecasts for the difference time series, not the original time series. To get the final forecasts for the original time series, we just need to add back the value at time T minus 365, and we'll get these forecasts.

Restoring the Trend and Seasonality

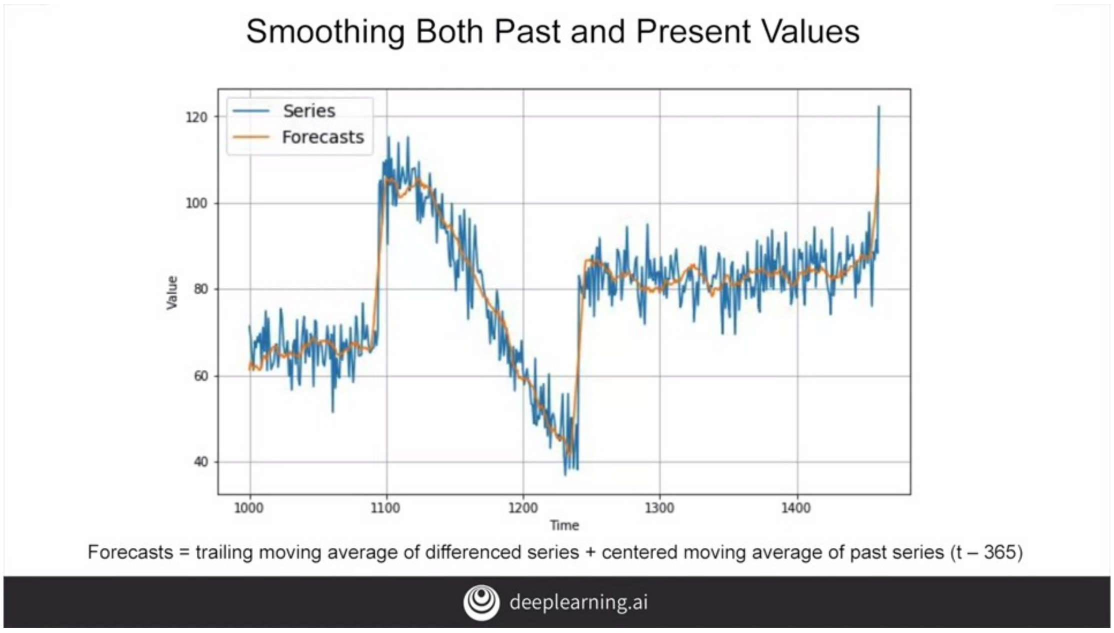


Forecasts = moving average of differenced series + series(t – 365)

deeplearning.ai

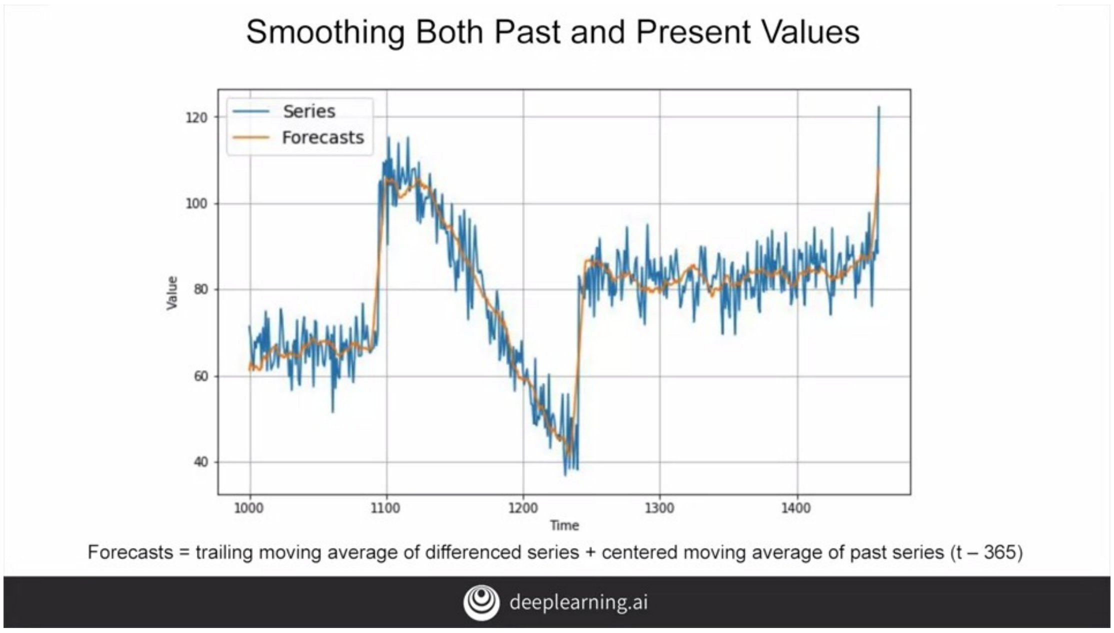
They look much better, don't they? If we measure the mean absolute error on the validation period, we get about 5.8. So it's slightly better than naive forecasting but not tremendously better.

You may have noticed that our moving average removed a lot of noise but our final forecasts are still pretty noisy. **Where does that noise come from?** Well, that's **coming from the past values that we added back into our forecasts**.



So we can improve these forecasts by also removing the past noise using a moving average on that. If we do that, we get much smoother forecasts. In fact, this gives us a mean squared error over the validation period of just about 4.5. Now that's much better than all of the previous methods. In fact, since the series is generated, we can compute that a perfect model will give a mean absolute error of about four due to the noise. Apparently, with this approach, we're not too far from the optimal. Keep this in mind before you rush into deep learning. Simple approaches sometimes can work just fine.

7. Training versus centered windows



Note that when we use the **trailing window** when computing the moving average of present values from t minus 32, t minus one. But when we use a **centered window** to compute the moving average of past values from one year ago, that's t minus one year minus five days, to t minus one year plus five days.

Then moving averages using centered windows can be more accurate than using trailing windows. But we can't use centered windows to smooth present values since we don't know future values. However, to smooth past values we can afford to use centered windows. Okay, so now we've looked at a few statistical methods for predicting the next values in a time series. In the next video, you'll take a look at a screencast of this prediction in action. Once you've done the statistical forecasting, the next step of course will be to apply the machine-learning techniques you've been learning all along in TensorFlow and you'll do that next week.

8. Forecasting

Let's take a look at running some statistical forecasting on the Synthetic Dataset that you've been working with. This should give us some form of baseline that we'll see if we can beat it with Machine Learning. You saw the details in the previous video and you'll go through this workbook in this video. Before you start, make sure you are running Python 3 and you're using an environment that provides a GPU. Some of the code will require TensoFlow 2.0 to be installed. So make sure that you have it. This code will print out your version. If you have something before to 2.0, you'll need to install the latest. To install it, use this code. At the time of recording, TensorfFlow 2.0 was at Beta 1. You can check out the latest install instructions on TensorFlow.org for the updated version so that you can install it. Once it's done, you'll see a message to restart the run-time, makes sure that you do that. Check that you still have a Python 3 GPU run-time and run the script again. You should see that 2.0 is now installed. The next code block condenses a lot of what you saw in the previous lessons. It will create a time series with trend, seasonality, and noise. You can see it in the graph here. Now to create a training validation set split, we'll simply split the array containing the data at index 1,000, and we will chart it. Again, we can see that the seasonality is maintained and it's still trending upwards. It also contains some noise. The validation set is similar and while the charts may appear different, checkout the x-axis. You can see that we've zoomed in quite a bit on it, but it is the same pattern. Now let's start doing some of the naive prediction. The first super simple prediction is to just predict the value at time period plus one. It's the same as the value of the current time period. So we'll create data called Naive Forecasting that simply copies the training data at time t minus 1. When we plot it, we see the original series in blue and the predicted one in orange. It's hard to make it out. So let's zoom in a little. We're looking at the start of the data, and that's sharp climb this C. So when we zoom in, we can see that the orange data is just one time-step after the blue data. This code will then print the mean squared and mean absolute errors. We'll see what they are. We get. 61.8 and 5.9 respectively. We'll call that our baseline. So now let's get a little smarter and try a moving average. In this case, the point in time t will be average of the 30 points prior to it. This gives us a nice smoothing effect. If we print out the error values for this, we'll get values higher than those for the naive prediction. Remember, for errors lower as better. So we can say that this is actually worse than the naive prediction that we made earlier on. So let's try a little trick to improve this. Since the seasonality on this Data is one year or 365 days, let's take a look at the difference between the data at time t and the data from 365 days before that. When we plot that, we can see that the seasonality is gone and we're looking at the raw data plus the noise. So now, if we calculate a moving average on this data, we'll see a relatively smooth moving average not impacted by seasonality. Then if we add back the past values to this moving average, we'll start to see a pretty good prediction. The orange line is quite close to the blue one. If we calculate the errors on this, you'll see that we have a better value than the baseline. We're definitely heading in the right direction. But all we did was just add in the raw historic values which are very noisy. What if, instead, we added in the moving average of the historic values, so we're effectively using two different moving averages? Now, our prediction curve is a lot less noisy and the predictions are looking pretty good. If we measure their overall error, the numbers agree with our visual inspection, the error rate has improved further. That was a pretty simple introduction to using some mathematical methods to analyze a series and get a basic prediction. With a bit of fiddling, we got a pretty decent one too. Next week, you'll look at using what you've learned from Machine Learning to see if you can improve on it.