Module 10: Time Series Analysis and Forecasting

Video Transcripts

Video 1: Introduction

So far, we have built machine learning models for datasets that lacked a notion of time. In this module, we will learn methods that applied to data that present themselves in a sequential manner. Such datasets are very common. They are used to track econometric quantities such as GDP and inflation rates in business for forecasting demand and commodity prices, as well as in science, medicine, finance, engineering, you name it.

The techniques that we will learn leverage the defining property of time series, which is that they are sequential. To analyze time series data, we will need new tools, such as the autocorrelation and partial autocorrelation functions, as well as new concepts such as trends, seasonality, and stationarity. We will learn two widely-used models for representing time series data, the decomposition model and the ARMA or autoregressive moving average model.

This is by no means a comprehensive account of the topic, but it will give you a good understanding of how to handle time series data, as well as two useful modeling tools for making forecasts. So let's get started.

Video 2: Problem Statement

There are many problems involving time series data that we could address. But here we will focus on the most important of these problems, the

forecasting problem. The forecasting problem is stated as follows. Given some historical data, try to predict – or 'forecast' in the time series lingo – what will happen over some future time window? This picture shows yearly data that rises and falls with a period of about a decade.

Let's assume that we are now in the year 2011, and we are tasked with predicting the future evolution of this time series over the next ten years from 2012 to 2021. This is yearly data. However, the methods that we will learn do not depend on the duration of the time step, only on the amount of data that we have. So it will make no difference whether we are making a ten-year prediction with 100 years of data, or a ten-second prediction with 100 seconds of second-by-second data. Because these two situations involve the same number of datapoints, the results will be the same.

Our approach to the forecasting problem involves four steps. First, we will collect some up-to-the-moment historical data. Then, we will use that data to train a model. Then, we will use the model to make a forecast. And finally, once the forecast horizon has passed, we will score it. In other words, we will evaluate the performance of the model. OK. Let's say we made our ten-year forecast back in 2011 and it is now 2021. So it is time to assess how we did. We need some notation. We will denote the value of the quantity at time t with y sub t. The size of the historical dataset is h, and the number of time steps being forecast is f. We collect the historical data into an array, y sub t minus h colon t, which we read as y sub t minus h to t.

In our case, this is y 1987 to 2011. The bold font here indicates that this is a vector, not a scalar. The 20-year forecast is denoted by a bold y with a hat on top and a subscript t to t plus f. In our case, this is y hat sub 2012 to

2021. The actual data for that period is y sub 2012 to 2021. And the prediction error, e sub t, is the difference between the prediction and the actual data for that period. This error, e sub t, is an array so we need to reduce it somehow to a single number. There are two common ways to do this. Notice that simply taking the average of the elements in the array does not work. Because positive and negative elements can cancel each other out, thus making a large error seems small. However, we can take the absolute values of the elements and then make the mean of those. This is known as the mean absolute error, or MAE.

Another possibility is to use the root mean squared error, or RMSE. This is computed by taking the square root of the mean of the squared errors. Here is an example. In blue, we have a time series consisting of 100 samples and we want to make a forecast of ten steps into the future. Our first task is to choose a forecasting model. A forecasting model is any equation or algorithm that generates the forecast from the historical data. Here we see two very simple models. The first, shown in orange, is to say that the next ten time steps will be the same as the last measurement seen. Well, that is pretty simplistic and it also wastes all of the historical data except for the most recent observation. The second option, shown in green, is perhaps a bit better. It extrapolates with a linear regression taken over the latest five datapoints. Again, this discards 95% of the historical data, but it is probably a better model.

To resolve which one of these two models is better, we must compare the scalar prediction errors that they produce. And we must do this continually as we move through time. It may be that one performs better in some cases and the other in other cases. The techniques that we will learn in this

module are quite a bit more sophisticated than these two. However, it is always a good idea to compare the results you get with any model to the predictions from simple models such as these. You may be surprised by the results.

Video 3: Modeling (Part 1)

In this video, we introduce new concepts from probability theory that will help us to analyze time series data. In previous lectures, we have described the data as having been generated by an unseen probability density function. And our task was to infer something about that PDF from the data. Here, again, we take the same approach except that now the random variables are organized in a sequential manner into a stochastic process.

A stochastic process is defined as an ordered sequence of random variables. Here we will denote them by a capital Y sub t, with t varying from one to capital T. Capital T is the length of the stochastic process and it could be infinite. That is, the process could stretch back into the deep past with no discernible beginning. A note on terminology. The term stochastic process applies to a sequence of random variables. A time series is a single sample from the stochastic process. Here we see three different time series, all of which were produced from the same stochastic process.

Next, we will learn about some important properties of stochastic processes. These are stationarity and independence. A process is said to be stationary when its statistical properties – meaning its mean, its variance, and the correlations between different points in time – remain constant over time. That is, if you focus on a time window of any size, then the statistics of the process within that window will be the same no matter

where you place the window in time. In particular, if you choose the window of size one, this means that the marginal distributions at each time are identical.

In mathematical notation, the distribution of the time series at some time t1 is equal to the distribution at any other time t2. The marginal distribution Y sub t is constant over time. A process is independent when all of its constituent random variables are mutually independent. It follows then that the joint probability of the entire stochastic process equals the product of the marginal probabilities across time. An independent process can be difficult to forecast because the values that we have seen in the past do not influence the current or future values. This is expressed mathematically by saying that the distribution of y at time t, given all the past information, is equal to the distribution of Y_t without that information. Stationarity and independence of stochastic processes are distinct from one another.

A process can be stationary and not independent. Or it can be independent and non-stationary. Or it can be neither or both. A process that is both stationary and independent is called IID, or independent and identically distributed. An important special case of an IID process is the Gaussian white noise process. In this process, every y sub t is distributed according to a Gaussian distribution with zero mean and variance equal to one. We will use Gaussian white noise processes later as generators of input noise for ARMA time series models.

Video 4: Modeling (Part 2)

When trying to build a forecasting model, it is useful to estimate the correlations between pairs of random variables. These correlations are

usually organized into a matrix called the autocorrelation matrix. Auto because it is the correlations between two different time instance in the same process.

Let's look closely at the structure of the autocorrelation matrix. Along the diagonal, we have all ones. These are correlations of Y sub t with itself. Then on the diagonals directly above and below that central diagonal, we see autocorrelations between time steps that are one time step apart, Y_1 and Y_2, Y_2 and Y_3, et cetera. Then another diagonal out, our correlations between time steps that are two time steps apart, and so on. Recall that a stationary process is one for which the statistical properties of any window of data are the same no matter where the window is placed in time.

So considering a window of size two, we realize that the correlations between Y_1 and Y_2 should be equal, the correlations for Y_2 and Y_3, et cetera. And thus we conclude that all of the values above and below the central diagonal must be the same. Here we call that value r_1. Similarly with a window of size three, we conclude that all of the entries that are two steps above or below the central diagonals should be equal, and we call them r_2, and so on. We move further away from the central diagonal.

Thus, for stationary processes, the autocorrelation matrix has a very simple structure that is constant along diagonals. This observation motivates the use of the autocorrelation function, or ACF, instead of the autocorrelation matrix for stationary processes. The ACF, shown here at a 45° angle, collects the values along the diagonals and plots them as a function of the lag, or the distance between points in time. This is a typical

autocorrelations plot. The first entry corresponding to a lag of 0, naturally equals 1 since it is the correlation of Y sub t with itself.

For many processes, the ACF will decay with lag, meaning that the data is more strongly correlated with recent measurements than with measurements further back in time. However, this is not always the case. We will see an example of this in an upcoming video. Just as we can compute sample correlations between random variables from data, we can also compute the sample autocorrelation function for a stochastic process from a time series. This is done by first computing the mean, y hat, of the time series, then computing the sample covariances, y sub k, for each lag value, k, with the formula shown here. And then dividing each c_k by c sub 0 so that the first sample autocorrelation, rho hat sub 0, equals 1.

But don't worry, you won't have to code this formula yourself. In the next video, we will learn how to compute this in Python using the statsmodels package. Let's have a look at some example autocorrelation functions. Here on the left, we see an autocorrelation function for a white noise process. Because white noise is independent, there is no correlation with lags greater than 0. The middle plot shows a sample of length 1,000 from this white noise process. We can feed this time series into the formula for the sample autocorrelation and produce the plot on the right, for rho hat of k.

Notice a few things. First, the value of rho and rho hat for k equals zero are both exactly one. This is unsurprising since the formulas guarantee it. Notice also that the sample autocorrelations for larger k are not identically zero, even though the true autocorrelations are zero. This is because these

are merely estimates from the data we have. The blue band around these values is a threshold, within which the sample autocorrelations can be regarded as negligible. This band gets narrower the more samples we have. As it is, the time series of length 1,000 is long enough for us to recognize that it was generated by a white noise process. In this example, the process on the right has a significant factor at a lag of 10. The time series that it produces has a vaguely periodic behavior with a period of about 10 time steps. And when we compute the sample autocorrelations, we can see a factor that stands out significantly from the blue band at a lag of 10. Thus from this sample of 100 points, we can infer that the structure of this stationary process has strong correlations with lag = 10.

Video 5: Autocorrelations

In this video, we will start playing with time series in Python using the statsmodels package. This is the website for statsmodels. Statsmodels provides functionality for doing statistical inference in Python. And it shares some functionality with scikit-learn. But it also adds time series analysis, which scikit-learn does not have. Like scikit-learn, statsmodels comes with a series of datasets that you can use to practice and test different models with. So this is the list of datasets that come included with statsmodels and includes things like the CO2 observations from the Mauna Loa observatory in Hawaii.

Also sea surface temperatures of…related to El Niño, and also yearly sunspots data from 1700 to 2008. So these are all time series data datasets. So we are going to be doing a couple things with statsmodels. We're going to be generating data series, and we are also going to be analyzing data series, or time series, sorry. For generating time series, we

are going to use the ARIMA process module from the TSA, or time series analysis package of statsmodels.
Do that, you have to create an ARMA process object. ARMA here means autoregressive moving average process, and it takes two parameters in the inputs. One is the autoregressive part, AR, and the other is the moving average part, MA. Don't worry about these terms yet. We'll learn about them in future lectures. For now, we're just going to set AR to 1 and negative 0.8, and MA to 1. And see how that works. So now we have a process and we can look at it. Indeed, it's a…an ArmaProcess object. So the first thing we're gonna do with this object, is to take a sample of the process.

And we do that by calling the generate_sample function on the ArmaProcess object. And we pass into that function the size of the sample or the number of datapoints in the sample, which in this case is going to be 100. And so when we run this, we get an array with 100 numbers in it. We can plot that using matplotlib in the usual way. And this is what it looks like. OK? What if we took another sample and plotted it? It would look different. And we could do this several times. And each time it would look different, it would be a different random sample from that process.

Now, the process depends on the parameters that you pass in. And you could, if you're not careful, pass in numbers that would produce an unstable process. So for now we are going to keep this at negative 0.8 so that we have a stable process. OK? The next thing we want to do is to plot the autocorrelations for this process. This…these are the theoretical autocorrelations that come from the process itself and not from the data.

So to do that, what we have to do is call the ACF function on the process. And this will return the autocorrelations. We pass in the number of lags or the number of coefficients that we want, in this case 20. And it will produce a…an array of 20 numbers. And it starts as expected with a 1 and it decays from there. Again, we can plot this using matplotlib. In particular, we're using the stem function of matplotlib. And what we can observe here is that these autocorrelations decay exponentially with the number of lags.

And that is typical of a stationary process. With a stationary process, you'll either see that the autocorrelations are decaying exponentially or maybe they will…you'll have a few that are non-zero and then they'll go to zero and stay at zero. OK, so that observation will help us to identify when a…a time series is stationary or not. To do that, we will have to plot the sample autocorrelations of the time series. So how do we do that? Statsmodels provides a very handy module for doing it called tsaplots. And that module includes a function called plot_acf, that is plot the sample autocorrelation function. And into that we pass our data z. Let's see what that looks like.

Oh, I didn't load this, so I have to load it first. And this is the sample autocorrelation function. We see that it looks more or less like the theoretical autocorrelation function. We've captured the first three. So the one is given and then the first three autocorrelations are greater than zero. But then after that they dip into the blue region where we cannot distinguish them from zero. That is, we cannot reject the hypothesis that the autocorrelations equal zero within this blue region. So let's see how these functions will work on real data. First, we're going to load the sunspots data. And even though it's available from statsmodels, that toy dataset is yearly data and we want to use monthly data. So I'm loading it from a file.

And this is what the file contains. We can see that it has a column called Date that has the date of each observation. So let's look at that. This is the first date. It's first 31st of January, of 1749, and it's a string. One of the very nice things that pandas allows you to do is to convert dates into a timestamp object. So you do that with the to_datetime function. And now the date is in a format called timestamp, which pandas knows how to deal with and has some nice features for. So what we would like to do is to use timestamps for our index. If we do that, then pandas will also be able to say, plot it nicely. So I'm going to use set_index and I'm going to pass in the Date column transformed to timestamps using the to_datetime method.

And that's what it looks like now. We have a date as the index, but we still have these two columns which were not very interested in anymore. So let's call drop(columns) to get rid of them. And we're going to do this inplace. Oops. OK, I just ran this twice. And so I need to start again in order to reload sunspots. And now we've dropped the...the troublesome columns. What we have now is the data that we're interested in. But this column header is really long. I'd like to make it a little bit easier to work with. So I'm going to rename it. OK, that looks a lot better. Now we have a nice dataframe with a...with timestamps in the index and the values in a nicely named column. Let's plot it. The sunspots on the sun surface of the sun increase and decrease in number with a period of about a decade.

So every decade they go down almost to zero. And then a few years later increased to 200 or 300. Let's see. Is this data stationary? Well, probably not because stationary data does not exhibit such oscillations. But let's plot the sample autocorrelation function and see what we get. So we're going to

copy this code, paste it down here, and pass in the sunspots. What we see is that the autocorrelations are decaying very slowly. And maybe we have to generate a few more to see what's really going on.
So I'm gonna plot 200 legs instead of 20. And this is what we get. Wow, this is not decaying at all, it behaves sort of sinusoidally and even after 200 lags is not seeming to decay. So this is a very good indication that the sunspots data is not stationary, which is something that we already knew. So what can we do? If we want to make it stationary, one common thing to do is to take a difference. And by that I mean, instead of plotting each value, plot the difference between each value and the next value. So we can do that by calling the diff function on sunspots. And now in…for each date we have the difference in the number of sunspots between that date and the month before.

But of course, because there was no month before and the first date, then we get a not a number there. So let's drop that on the call, dropna. And that looks a lot better. OK, now we can plot the autocorrelation function for, let's call this diff_sunspots and plug it into here. That's quite a few lags, maybe difficult to see. Let's plot only 20. And what we see here is that we get the one and then there are two lags that are non-zero and then the rest pretty much drop into zero. So that is typical of stationary data. And we have a good indication here that even though the sunspots data is not stationary, maybe the difference of sunspots from one month to the…to the next is stationary.

Video 6: Decomposition (Part 1)

Here are some examples of time series. On the top-left we see a few recent years of the NASDAQ Composite index. This data could be described as

having a clear upward trend, with a few abrupt variations. The plot in the upper right show the number of spots observed on the surface of the sun. In contrast with the stock market, this data rises and falls periodically with no overall increasing or decreasing trend. We say that this data has periodicity, or seasonality. The bottom two plots are more irregular.

On the left we see the rate of unemployment in the United States over the last 70 years. Like the sunspots, this time series exhibits no clear upward or downward trend. However, its variations do not occur periodically. Instead, it exhibits occasional sharp rises followed by a gradual decrease. And these cycles happen more or less randomly and have different durations. The plot on the bottom right is a pure stationary process of the sort that we saw in the last video. Compared to the rest, it seems much noisier and less predictable. And it is true; stationary processes are more difficult to predict than non-stationary processes since they lack a trend, or seasonality, or cycles. However, as we've seen, stationary processes are not necessarily white. They may still contain structure in the form of the autocorrelation function, which we can estimate and use to make short-term predictions.

In this video, we will learn how to model long-term behavior, such as the trend, cycles, and seasonality. And in a future video, we will learn how to extract structure from a stationary process using the ARMA framework. Our approach will be to model the time series y as a composition of four terms, the trend, cycles, seasonality, and the residue. The symbols in this formula are in bold font because they are vectors. Y holds the historical data, which we had previously denoted with a y sub t minus t to h. The trend is the long-term behavior such as the slow rise of the NASDAQ index. The seasonality is the periodic behavior such as we saw in the sunspots dataset.

Its key defining characteristic is that the period is constant and known to the modeler. The cycles are variations that break with the trend, but do not happen with a set period. Any behavior that does not fall into these three categories is considered as a residue, or residual. If the signal, y, is well characterized by this combination of a trend, cycles, and seasonality, then we'll expect the residue to behave like a stationary process. That is, it will exhibit no structure at the time scales of cycles, seasons, and trends.

However, it may still contain hard-to-see structure at the level of a few time steps. Our next task is to extract these components. We could think of many ways of doing this, but what I will show here is a basic and very common approach. And it is what is implemented in the statsmodels package.

Video 7: Decomposition (Part 2)

Our procedure has three steps. First, we will compute the trend and cycles by smoothing the data. We will assume that the cycles are rare and can therefore be lumped in with the trend. Second, we'll compute the seasonal component. And third, we'll check stationarity of the residue. We will work with the sunspots data. Imagine the year is 1984, and you have been tasked with predicting the next 15 years of sunspot activity. You are to base your prediction on the observations of sunspots from 1900 to 1984.

Shown here in a lighter gray color are the sunspots that occurred after 1984, which we know in hindsight but will not use to build the model. In the first step, we extract the trend and cycles by smoothing the data. That is, by running y through a filter, which we call f – f is an array of positive numbers

that add up to one. The length of f should be chosen to be similar but greater than the period of the data. In this case, the period is about 128 months. And so f has a length of about 129. To compute t, at some time, we take a weighted average of nearby values of y with weights given by f.

Because the filter is longer than the period, it will remove the seasonal component completely and leave only trend and cycles, assuming the cycles are longer than the seasonality and not too severe. Here is the resulting trend in red. It looks pretty flat, so at this point we can make a decision to extrapolate it into the future with a horizontal line, as shown here in orange.

Had we observed an increasing or decreasing trend, we might have used a different extrapolation technique. For example, we may have used a linear regression with linear, quadratic, or exponential basis functions. The next step in the process is to compute the seasonal component. To do this, we chop the historical time series into segments of length one period. Then we overlap those segments and take their average. Like many topics in machine learning, time series analysis is a bit of an art. And decisions must be made at every step based on the data at hand. At this point, we notice that the period of the data changes slightly from one season to the next.

And so we cannot define a single overall period. We also notice that there is significant variation in the amplitude of the oscillations. For example, there was a large peak of around 250 sunspots in the late 1950s, while the peak in the 1970s was around 150. However, throughout the historical data, we observe that a), the minimum number of sunspots in every season is very nearly zero. And b), the peak is about twice the trend. So how can we

incorporate these observations into our model? Here is a suggestion. Let's replace our additive decomposition, y hat is equal to t plus s, with a multiplicative model, y hat is equal to t times s, and scale the seasonal component so that it oscillates between zero and two. Then y hat will oscillate between zero and twice the trend as desired.

Video 8: Decomposition (Part 3)

All right, let's continue. We have now divided the time series into seven seasons, each lasting about a decade. Then we scaled each season so that its minimum and maximum values are zero and two. Those seven lines are plotted here in gray. Then we took the average of the seven gray lines. The resulting blue line is a bit too noisy to serve as our seasonal template. To deal with this, we will put the original data through a filter just like we did with to extract the trend. But this time the filter will be much shorter so that it only smooths high-frequency noise without affecting the seasonal behavior. OK, so that looks a lot smoother. But now we can see that there are some outliers in the data. Let's remove those and try again.

This looks much better. We were left with only three seasons, but the average now follows them quite closely. Now we can construct the seasonal signal by repeating the template we've just constructed. And it looks like this. If we now plug this into our formula, y hat is equal to t times s. This is what we get. The red line is the trend. The blue line is the trend multiplied by the seasonal component. And it seems to follow the historical data pretty well. The final step in the process is to produce the residue of the model by subtracting y hat from y. And to check whether this signal is stationary. This is done by checking the autocorrelation coefficients, as we will see in a future video. If the residue is not stationary, then this might

suggest that some long-term trend, or seasonality may have remained unmodeled.
In this case, it may be worthwhile to go back and revisit previous steps. However, if the residue is stationary, then there is not much more structure that can be extracted using decomposition techniques. Let's now use the model we have just built to make a prediction. We do this simply by extending the seasonal component into the future. And in this case, multiplying it by the extrapolation of the trend. The result is shown here in green. Looks pretty good to me.

Now imagine that 17 years have gone by and we have collected observations corresponding to the prediction we made back in 1984. Just as we computed the residue as the historical data minus the model, we can now compute the prediction error as the observed data minus the forecast. Here is the prediction error, plotted in purple. This table shows our two error metrics, mean absolute error and the root mean squared error applied to both the residue and the prediction error. Both of these quantities represent estimates of the error in units of number of sunspots predicted. And they both show something reasonable, which is that the residue is smaller than the prediction error.

To conclude on the topic of time series decomposition, I'd like to stress that the essence of this technique is simply in the identification of trends, cycles, and seasons as common structures in many real-world examples. However, the details of how we decompose the time series into those parts vary widely depending on the application. And it is here that your own ingenuity and experience as a data modeler will come into play.

Video 9: Decomposition (Part 4)

In this video, we will go over the code that was used to perform the time series decomposition of solar sunspots data from the last video. So we will begin by loading the data and it is stored in this Sunspots.csv file. And then we are going to do the same preprocessing as we did before. We are going to set the index to a datetime series of dates. Then we're going to drop a couple of unwanted columns. And finally, we'll rename the last column to: values. And this is what it looks like.

Now we're not going to use all of this data for the prediction. We're only going to use data from 1900 to 1984, to construct the model. And then we're going to make a forecast from 1985 to 2000. So let's store that information into two series called y_hist and y_future. Now look at this nice thing that you can do because the index is a timestamp. You can use this notation to get the rows from 1900 to 1984. And you can imagine that this would not work at all if the index where a string. But it does work here.

So now we can plot the historical data, which appears here in black from 1900 to 1984. And then from 1985 onward is data that is not available to the model training procedure. All right, so the first step in building the model is to extract the trend. We need a period to do that because we need to smooth the model or the…the historical data with a filter, which is the same length as the period. So we know that the period for solar sunspots is about 128 months. And so we can build our filter using that information. The filter is going to have length 129. And it's going to be all ones except for the first and the last element are going to be half that size.

And then finally we divide the filter by the period so that the sum of all of the coefficients in the filter must equal one. And they must equal one, so that the filter does not affect the mean of the historical data. So once we have the filter, we can run the historical data through the filter. And now we have the trend. And if we plot that, this is what it looks like. Now there's one problem here, which is that this trend is not going all the way to the beginning or all the way to the end of the historical period. And we especially need it to reach the end. So we need to extrapolate this so that it reaches the end. And we do that with this function called extrapolate_trend.

And if I run that, now the trend reaches both ends. With detrended data, we can now with...I'm sorry, with the trend, we can now compute the detrended data, which we get by subtracting the trend from the historical data. So let's do that. And we can plot it. The detrended data looks pretty similar to the historical data, except that now the mean is at zero. The next step is going to be to split the historical data into seasons. And as we know, the seasons are of approximately 128 months in length. However, they are not always 128 months, sometimes they are a little less or a little more.

So we need to find, manually, the beginning and end of each season. And the way we can do this is to identify where these curves reach the bottom, which is practically at zero sunspots. So I did that manually and found that the indices for those low points are these numbers. And passing those indices through the index, I can get the dates for the low points. So the first low point is on September 30th of 1901. The second is December 31st of 1912. And those are these two low points. OK, so now we can plot that. I use this axvline to create these vertical lines through the low points.

And you can see here, there is…there are some that may be a little bit larger than others. But essentially we have seven seasons, one, two, three, four, five, six, seven. And the next thing we need to do is to extract the historical data, or the detrended data, for each of these seasons and find an average. At this point, it will be important to have a common period for every season so that we can store the data in a matrix. So we compute that period by taking the difference of the low index, which is the number of months in each season. Taking the average of that, then rounding that number, and casting it as an integer. And that gives us 128. The number of seasons is the number of orange bars, minus one. And that is seven as we saw. And now we're ready to extract the mean of the seasons.

So first we create a matrix that has 128 rows and seven columns. And then we cycle, we iterate through each of the seasons. And we snip out the piece of detrended data that goes from the orange bar and has a length of one period. And then recall that we want these to bottom out at zero and have a height of two. So we normalize that data and multiply it by two, and then we can store it in our matrix. And once we've collected all of those seasons, we can take the mean and that will be our mean seasons.

OK. So let's run that. And this is what it looks like. Each of these gray lines is one of the seasons. So there are seven gray lines here and the blue is the average. Now what's not nice about this is that this average is pretty jagged. And these, this jaggedness is probably not a structural feature. It's not something that we should expect will repeat in the future. So we should smooth it out. And that's our next step. To smooth it out, we're going to pass it again through a filter, a convolution filter. And this filter is going to be much shorter than the previous one. It's not going to be 129.

It's only going to be 9 months long. So here we create the filter and we're basically going to repeat the same as we did here, which is extract the line, but now pass it through the filter with convolution and extrapolate, and then scale it, and save it. And once we have all of them, we can take the mean. Do that. And that does look a lot better. They're not fewer lines here. There's the same number of seven gray lines. And the average looks a lot smoother than before. So I'm satisfied with this. The one thing that's not nice is that we have some outliers. That is, this line here may be bringing our mean down and this mess over here we'd also rather not have.

So our next step is going to be to remove the outliers. And again, with some manual work, we found that the outliers were lines or seasons number zero, one, and then five and six. And so we're keeping two, three, and four. That's what we do here. We only keep seasons two, three, and four. And then take their mean again. That looks a lot better. So this is going to serve as our template for each season. And our next step is going to be to build the complete seasonal template. And we do this simply by repeating that same shape over and over again. And that's pretty simple. The way we do it is we just construct a series whose index is the same as the historical dates.

And we set all the data to zero. And then we will copy into that index the mean_seasonals, as long as we are not in the last case. So as long as it's going to fit, we copy it in. And in the last case, we are going to fall into this clause. It doesn't fit, so we only copy in as much as will fit. And so that gives us the complete seasonal template. Finally, to finalize the model, we take that template, we multiply it by the trend, and we multiply it by two. And this is our complete model. We can plot that here. It shows up in blue and it does look like it's following the data fairly well. It's not a bad model.

So, but let's quantify how good this model is by computing the residue. The residue is the difference between the historical data and the model. So we compute that here and we can plot it. And we would like for this residue to be stationary. Is this stationary? Well, it doesn't look too un-stationary. We'll have to do autocorrelations to see if it's stationary. But first let's look at how big it is, what is the size of the residue? We can quantify that in terms of the mean absolute error or in terms of the root mean squared error.

And we…we compute these, the mean absolute error. We first…we take the absolute values of the residues, and then we take the mean of that. The root mean squared error is computed by first squaring all of the errors, then taking the mean of that, and then taking the square root of that number. And this is what we obtained, 22.72 for the mean absolute error and 29.21 for the root mean squared error. Now back to stationarity. Let's plot the autocorrelation function. Hmm, this doesn't look too stationary.

If it were stationary, we'd expect this to decay or maybe just cut off at some point. But this is not decaying exponentially. It looks like it goes down, and up, then down, and up. And finally it goes to zero. That is indicative of non-stationarity. And maybe it means that we can go back and improve this model somehow until we achieve stationarity of the residue. But we're not going to do that right now. We're going to forge ahead and produce a forecast with this model. So to do that, we first have to project the trend forward. So we do that by creating a series whose index is the future time. And the data will all be equal to the last datapoint in the trend.

That is this point right here. So this is going to project a line. That's the projection of the trend. The seasonal projection, again, we create a series

with the future index. And then we are going to go step-by-step through the future and copy in the value from the seasonal a couple periods ago. And that will produce our seasonal projection. Then our forecast will be applying the model, which is two times the trend times the seasonal. So let's see what that looks like. There is the projection into the future.

Now, let's compute the error for this forecast. The error is the future data minus the model, minus the forecast. And this, of course, we could only do after we reached the year 2001 where we have this data. Let's say that's where we are and we compute the prediction error, we can put it into the same plot as the residuals. It appears here in pink. Now, what is the size of the prediction error? We apply the same metrics, the mean absolute error and the root mean squared error, and we get 30 and 39.

Now if we compare that to the metrics for the residual, which were 22 and 29, we see that they are quite a bit larger and that's very reasonable. It's like saying that the error in the testing dataset is larger than the error in the training dataset. Which is what you expect since the training was done, in a way, to minimize the error. So this is all very reasonable.

Video 10: ARMA (Part 1)

In the previous video, we learned how to create a model of a time series that exhibits a trend and seasonality. Having extracted those components, we were left with a residue, which in the best-case scenario would be stationary. In this video, we will introduce the ARMA family of models. These are designed to capture the time invariant structure exhibited in stationary time series. Once this has been done, we will be left with a final residue of structureless and useless white noise.

Two of the most useful models for representing stationary processes are the moving average, or MA, process and the autoregressive, or AR, process. Here the q and the p are the orders of the moving average and autoregressive process, respectively. So, for example, MA(3) is a moving average process of order three. Both the MA and AR models regard the stationary process as the output, y sub t, of a procedure that is fed with Gaussian white noise, which we denote by a sub t. For a moving average process of order q, the white noise is put through a filter with coefficients theta one through theta q. At each time, t, the output, y sub t, is computed by multiplying the theta coefficients by the past q-values of the input noise and then adding a sub t. This is summarized in the formula for the moving average model. The autoregressive process of order p is a little different.

Instead of thetas here we use the letter phi to represent the p coefficients of the model. And instead of feeding the white noise to the coefficients, we bring back the output y sub t and use that. Hence the name autoregressive, the procedure feeds back on itself. The mathematical formula for the AR(p) process is shown here. Take a moment to observe and understand how each of these formulas correspond to their respective diagrams. Here we see six samples of moving average and autoregressive processes of order 2, 5, and 10. A 0th order MA or AR process is just white noise. And the higher the order, the more structured the signal will become, generally speaking. However, just looking at these samples, it is difficult to guess whether the time series is MA or AR, let alone the order of the process.

We are going to need better tools for analyzing stationary processes. These plots show the autocorrelation functions for each of the processes whose samples we just saw. Notice something interesting about the top row. The

autocorrelation functions for moving average processes begin with a one like all autocorrelation functions, and then is followed by as many non-zero entries as the order q of the process. And beyond that, it is all zeros.

This observation gives us a direct method for identifying the order of a moving average process. Just look at the lag of the largest non-zero entry in the ACF. The bottom row shows that this does not work for autoregressive processes. In this case, the 0th entry is again one. And after that the entries decay more or less exponentially without ever reaching absolute zero. These plots show a different analytical tool called the partial autocorrelation function, or PACF. We will not go into the mathematical details of the PACF as we did with the ACF. However, inspecting the PACF, we notice that the situation is now the reverse. The order p of the autoregressive process shows up as the largest lag with a non-zero entry.

So the AR(2) process has a non-zero PACF for lag two and zeros beyond that. And, similarly, for AR(5) and AR(10). On the other hand, the partial autocorrelation function for the moving average process are not very informative. OK, so it seems now we have two tools that will help us to determine the orders p and q of the autoregressive and moving average processes. The problem is that we do not have these plots. All we have is the data from which we can calculate sample autocorrelation and partial autocorrelation functions. Let's look at those. The top row has sample autocorrelation functions for the MA processes. And the bottom, our sample partial autocorrelation functions for the AR processes. For MA and AR processes of order 2 and 5, counting the number of non-zero coefficients before the plot dips into the blue region provides the correct answer.

However, this is not the case for MA(10) and AR(10). If we were to guess the order of these processes on the basis of these plots, we would have guessed MA(6) and AR(5). For this reason, you will rarely encounter AR and MA models of such high order.

Video 11: ARMA (Part 2)

Having understood the autoregressive and moving average processes, we can now introduce the ARMA process, which is a simple combination of these two elements. Here on the bottom left we see the formula for the autoregressive process of order p. I have put the summation on the left-hand side of the formula in order to gather all of the y's on the left and the a's on the right.

Then on the bottom right, we have the formula for the moving average process of order q. Again, the y's are on the left and the a's are on the right. An ARMA process of order p,q is obtained by taking the left-hand side of the AR(p) and equating it to the right-hand side of the MA(q). Arma(p,q) is a strictly larger class of models than both AR and MA, since we can get any AR model by setting q equal to zero and any MA model by setting p is equal to zero. In fact, the ARMA family of models is very general, as it can replicate any zero mean stationary process to a desired precision by setting p and q large enough and carefully selecting the phi and theta coefficients. Next, we will learn how to do this.

Video 12: ARMA (Part 3)

The steps to modeling and forecasting time series data with ARMA are as follows. First, one must check that the signal is stationary. Remember that

ARMA only applies to stationary processes. The second step is to use the sample autocorrelation and partial autocorrelation functions to select p and q, the orders of the AR and MA pieces, respectively. The third step is to find the values of the theta and phi coefficients of the model. The procedure for doing this is similar to other models we have seen in this course, such as linear regression, in that it is based on an optimization problem.

After the training of the coefficients, we now have a model and we can compute residuals. If the ARMA model has done its job, then the residuals will be white. And the model can then be used to make a forecast. We previously saw the autocorrelation function of the residue from our model of the sunspots data. And we concluded that it was likely non-stationary.

Let's push ahead anyway and try to apply the ARMA model to this data. Here we see both the autocorrelation and partial autocorrelations of the residue. The PACF decays very quickly and suggests that we should consider an AR model of order one. Let's try that. To build the model, we will use the ARIMA class from statsmodels. The constructor for the ARIMA class takes the time series to be modeled and the order of the model. Here we are building a model of order p is equal to 1 and q is equal to 0. The 0 between those two components corresponds to the I in the ARIMA. Setting this to some number d will cause the model to first take d differences of the input data in order to achieve stationarity.

We have seen how differentiating a non-stationary time series can result in a stationary time series. So this is a common trick. However, we will not use it here. Having built the ARIMA object, we can then call the fit method on that object. This invokes the parameter training procedure. In our case, we

have only a single phi parameter to train. The model can now be applied to the training data; that is, the residue from the decomposition model.

Confusingly, perhaps, this is done with the predict method of the model. The result is seen here in orange. And compared to the original residue, in blue, there is a pretty tight correspondence. But don't be fooled. This is because the ARMA model is making only one step prediction from each point in the original residue. If we now subtract the ARMA prediction from the original residue, we get the residue of the ARMA model; shown here in blue.

Looking at the autocorrelation of that, it is now quite close to white noise and hence, there is no further stationary structure to extract from the sunspots time series. We have seen that ARMA significantly reduced the size of the model residues, but will it help with the ultimate goal of prediction? The answer in this case is no. We should keep in mind that the reach of ARMA models into the future is commensurate with their order.

Since the order of this model was p equals one, we only expected to make useful predictions a few steps into the future. This could be very helpful for predicting the number of sunspots a couple months from now, but does not help with our original task of forecasting 15 years into the future.

Video 13: Conclusion

So to summarize, we have seen two methods for forecasting time series data. The decomposition approach attempts to identify long-term behaviors captured in trends and seasonality. The ARMA approach is very good for making short-term predictions.

This has been but an introduction to the large topic of time series forecasting. But I believe you now have the tools to investigate more advanced techniques on your own. So, happy forecasting.