post01-jenny-yang

Jenny Yang

October 28, 2017

It's Time to Talk About Time Series

Import packages:

```
library(tseries)
library(forecast)
```

The ability to make informed decisions is undoubtedly crucial for business and policy decisions. These decisions can have a huge impact on the future health of a company or a group; but how do these informed decisions come about?

Time series modeling allows us to make predictions about the future using past time-based data, and these insights allow decision makers to formulate plans for addressing the future. Exploring and fully understanding time series requires time and considerable effort (there's a whole class dedicated to it within the stats department). Therefore, this post will attempt to only a brief overview of the procedures for time series modeling without delving into some of the deeper mathematical framework (much of which requires mathematical background developed in other classes.)

Although time series analysis is outside the scope of 133, it's definitely an important topic for both statistics majors and for analysts working with data involving a time aspect. R is a great tool to use because it already has many robust packages (my demonstration nowhere near covers all the cool things you can do with R in regards to time series) that enable us to carry out the steps of time series analysis.

Due to the nature of this topic, I did extensive research and pooled together multiple resources to develop my knowledge necessary for understanding time series, and pieced together the concepts I learned to create this post. A comprehensive list of all sources used is listed at the end of this post.

In this post, I will first introduce the basic conditions necessary for performing time series analysis, before discussing the framework on actually conducting the analysis. There are different types of models used to for time series (including VAR), but I'll be focusing on ARIMA models, which is a common model that uses historical data to make predictions.

To illustrate the steps, we'll be using the built-in R dataset AirPassengers, which details the monthly international airline passengers from 1949-1960, measured in thousands. This dataset is often used for demonstrating non-stationary seasonal time series.

Let's examine the data.

```
class(AirPassengers)
```

```
## [1] "ts"
```

This tells us that our data is in a workable time-series format.

```
AirPassengers
```

```
##      Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
## 1949 112 118 132 129 121 135 148 148 136 119 104 118
## 1950 115 126 141 135 125 149 170 170 158 133 114 140
## 1951 145 150 178 163 172 178 199 199 184 162 146 166
## 1952 171 180 193 181 183 218 230 242 209 191 172 194
## 1953 196 196 236 235 229 243 264 272 237 211 180 201
## 1954 204 188 235 227 234 264 302 293 259 229 203 229
## 1955 242 233 267 269 270 315 364 347 312 274 237 278
## 1956 284 277 317 313 318 374 413 405 355 306 271 306
## 1957 315 301 356 348 355 422 465 467 404 347 305 336
## 1958 340 318 362 348 363 435 491 505 404 359 310 337
## 1959 360 342 406 396 420 472 548 559 463 407 362 405
## 1960 417 391 419 461 472 535 622 606 508 461 390 432
```

Stationary Series

In order to conduct time series analysis on data, the data must be stationary. There are three criterion that the data must fulfill in order to be considered stationary.

1. Constant Mean

The mean of the data should be constant throughout time.

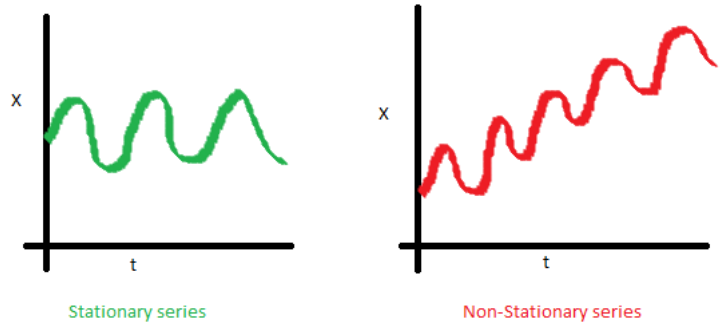


Image courtesy of Analytics Vidhya

2. Homoskedascity, or constant variance.

"Homo" means "same," and "skedastic" means "variance." The data that you're working with should be homoskedastic (an alternative spelling is homoscedastic), which means it should have the same variance throughout time.

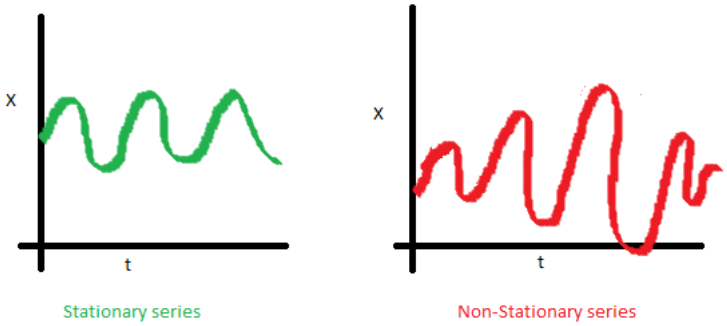


Image courtesy of Analytics Vidhya

3. Covariance

The covariance of the series with itself must be constant for all time periods.

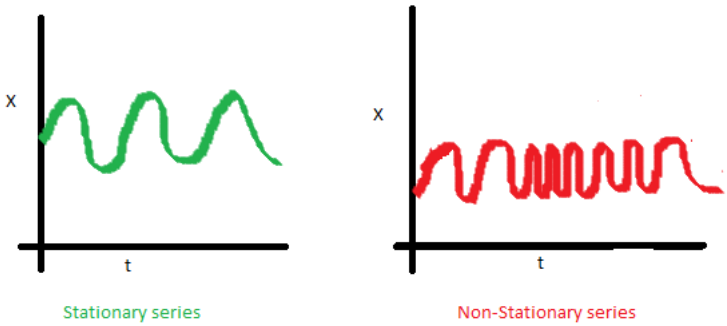


Image courtesy of Analytics Vidhya

In a nutshell, the three conditions require that expectation, variance, and covariance of the data are not dependent on time (they are time invariant, and the mean, variance, and covariance do not depend on the time at which they are observed.)

But what if your data isn't stationary? If one of the three conditions is violated, then you cannot build a time series model. But don't despair! You can stationarize your data. But first, let's examine the model we'll be using.

ARIMA

ARIMA, or auto-regressive integrated moving average, is a model that is specified by three parameters (p, d, q). We can therefore break it down into three components. Since ARIMA relies on past values, it is best used when working with a long and stable series.

Auto-Regressive

The AR component AR(p) uses past values, where p specifies the number of lags.

For example, GDP can be modelled using AR(1), where 1 is the lag (since our current data depends on the previous year's). Intuitively, this makes sense, because this year's GDP depends on last year's. In this case, the auto-regressive equation is formally written as

x(t) = \(\alpha\) * x(t-1) + error(t).

Integrated

The integrated component (I(d)) represents the degree of differencing. Differencing is often used to stationarize a series (more on stationarizing series below.)

Moving Average

The moving average component represents the error of the model, and q determines the number of such terms to be included.

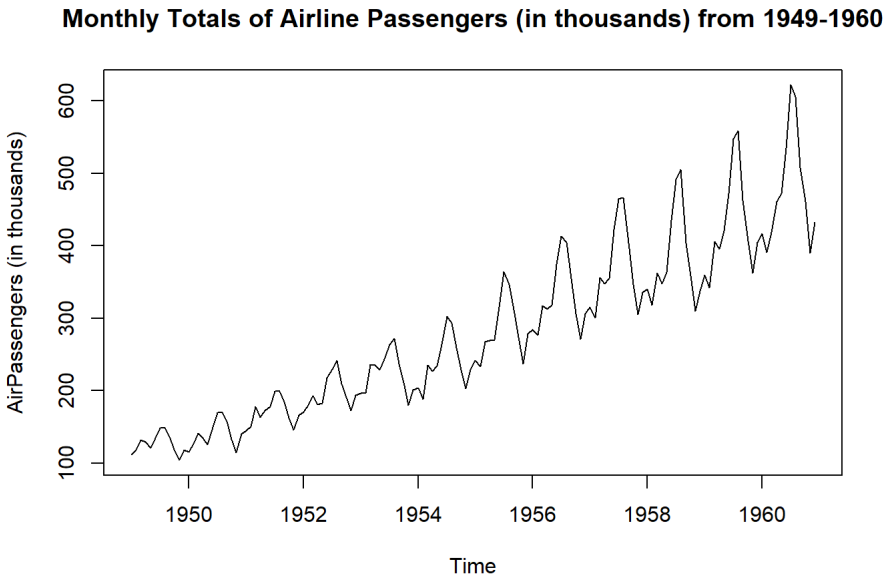
We'll figure out how to estimate the parameters once we start building our model.

The following lists the general steps, with an example, to take when performing a time series analysis.

1) Explore, visualize, and understand your data.

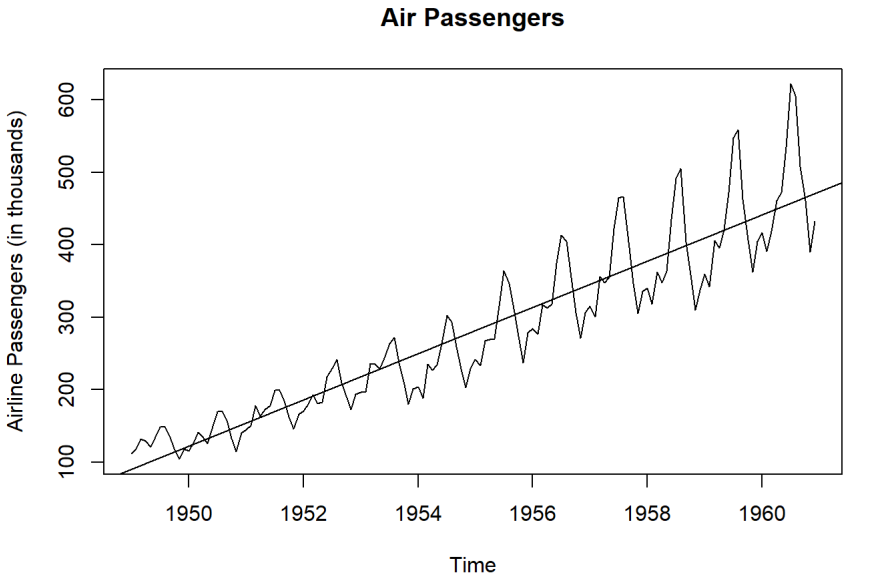
Before we can conduct our analysis, we should understand what our data actually looks like.

```
plot(AirPassengers,
     main = "Monthly Totals of Airline Passengers (in thousands) from 1949-1960",
     ylab = "AirPassengers (in thousands)")
```



Let's also try fitting a line to our data.

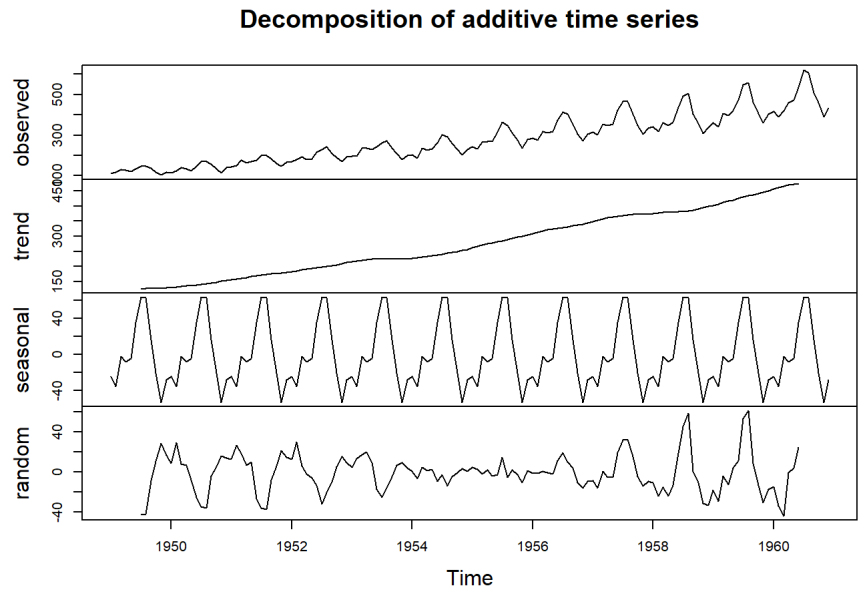
```
plot(AirPassengers,
     main = "Air Passengers",
     ylab = "Airline Passengers (in thousands)") #plots the data, as above

abline(reg=lm(AirPassengers~time(AirPassengers))) #adds the best-fit regression line to our graph
```



We clearly see that both the average and the variance of the data is increasing with time (as time goes on, there is an increasing number of airline passengers, and the volatility, or variance seems to be increasing as well), which violates two of our requirements for stationary data. Furthermore, note that there seems to be a seasonal component of 12 months–a similar trend, or increase/decrease of airline passengers seems to occur regularly every year, probably attributed to the fact that people travel more during certain times of the year (i.e., holidays and during the summer.)

For additional analysis, we can decompose the series to identify the trend, seasonality, and error (noise).

```
#decomposes into trend, seasonality, error
plot(decompose(AirPassengers))
```



Again, we se that there is a clearly an increasing trend as well as a seasonality component.

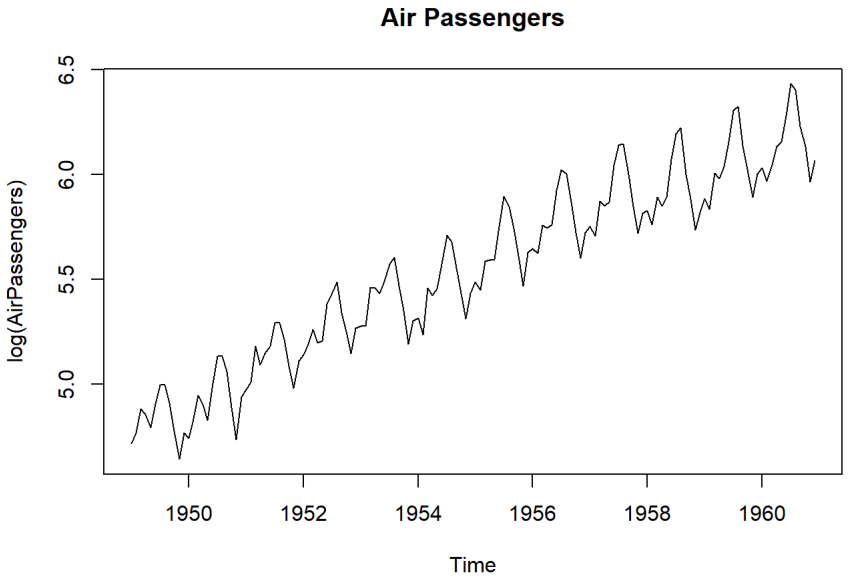
2) Stationarize your data

This is often the most time-consuming and challenging part. If your data is already stationarized, then congrats! You've saved yourself a lot of trouble. Due to the scope of this post, I'll only be describing one of the many methods to stationarize your data, but there are definitely many more not touched on here (e.g., differencing beyond one period, fitting the data to a function, etc.) The method for stationarizing your data depends on the shape of data; differencing is used when the data follows a linear trend.

As we saw from visual inspection, our data is not stationarized.

In data analysis, taking the log of our data is often used to scale data and to adjust for skewedness of large values (on a somewhat unrelated note, taking the log of data is sometimes used because logs allow us to think of the numbers in terms of ratios, which might be more intuitive for a particular analysis you're conducting.)

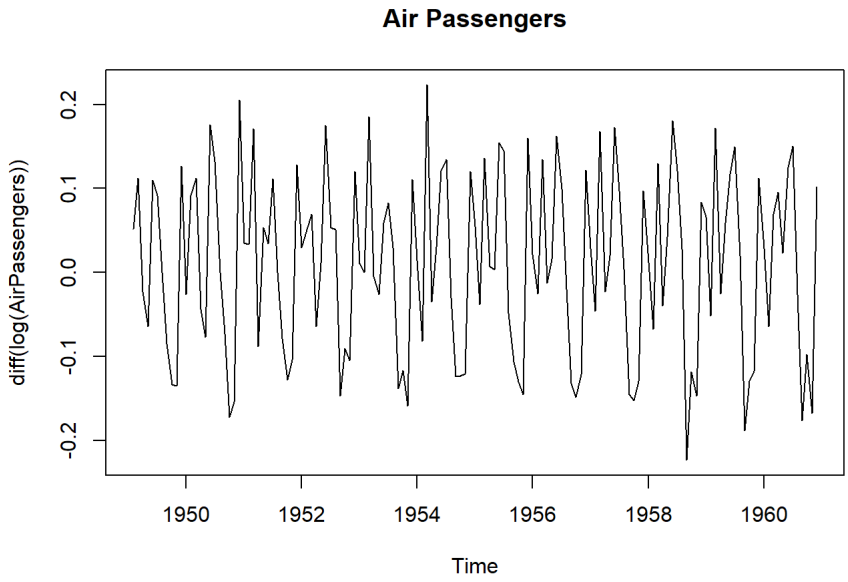
```
plot(log(AirPassengers), main = "Air Passengers")
```



The y-scale is now around 5-6.5 rather than 100-600 (remember, these numbers are in thousands.)

Now, let's try differencing AirPassengers. Differencing simply refers to taking the difference between one time period and the previous year's period (in our case, we're taking the difference on the current year and the previous year.) This technique allows us to both stabilize the increasing variance of the series and remove autocorrelations.

```
plot(diff(log(AirPassengers)), main = "Air Passengers")
```



Miraculously, our data is now stationarized! There is now an oscillating pattern around 0. Since we only differenced the data one time, we can see that d = 1 is a sufficient parameter (We could have included the parameter differences = 2, 3, etc, but it's unnecessary to go beyond d =1 since it works.)

Augmented Dickey-Fuller (ADF) test

There are, of course, more statistically sound methods to check for stationarity than just eyeballing the graph. One (of many) ways to check for stationarity is by using the Augmented Dickey-Fuller, or ADF, test. By using the ADF test, you are essentially conducting a hypothesis test to check for stationarity, with the null hypothesis that the series is non-stationary.

Note that in R, the Augmented Dickey-Fuller test (found in the tseries package) may be misleading since it de-trends (removes the trend component) the data for you, which may bias results. However, for the purposes of this demonstration, I'm showing the adf test for the data that we've already stationarized.

```
#Dickey-Fuller Test, testing on the stationarized data
adf.test(diff(log(AirPassengers)), alternative="stationary", k=0)
```

```
## Warning in adf.test(diff(log(AirPassengers)), alternative = "stationary", :
## p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  diff(log(AirPassengers))
## Dickey-Fuller = -9.6003, Lag order = 0, p-value = 0.01
## alternative hypothesis: stationary
```

We see that the p-value is significantly small (we can reject at 1%) and so our data is now sufficiently stationary to proceed with modeling. Generally, a p-value less than 0.05 is sufficient.

Another test you can use to check whether the data is stationary is the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test, which is kind of the opposite of the Dickey-Fuller test; here, accepting the null means that the series is stationary.

```
kpss.test(diff(log(AirPassengers)))
```

```
## Warning in kpss.test(diff(log(AirPassengers))): p-value greater than
## printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  diff(log(AirPassengers))
## KPSS Level = 0.022017, Truncation lag parameter = 2, p-value = 0.1
```

We see that the p-value is 10%, which is sufficiently large enough to assume stationary series.

3) Find optimal parameters for your model using ACF and PCF

Once we have stationarized our data, we can proceed with building an ARIMA (auto-regressive integration moving average) model. The two key tools we'll use here are the ACF and PCF plots, which check for significant lags in the series. These plots help us determine whether we need AR or MA terms to correct for autocorrelation.

ACF (autocorrelation function) plots the correlation between a series and its lags (the correlation between x(t), x(t-1), x(t-2), etc.) The ACF plot is just a bar chart of the coefficients of correlation between a time series and lags of itself. In simpler terms, the autocorrelation shows whether the previous states of the time series influence the current state. If the autocorrelation bar crosses the blue line, it means that the specific lag is correlated with the current series. ACF plots are used for determining the order of the MA(q) model. The ACF chart cuts off at the qth lag, which

means you have order q of the MA model. If the decay in the ACF chart is very slow, that suggests the data is not stationary enough to proceed with analysis; if the autocorrelation drops very quickly, the time series is stationary.

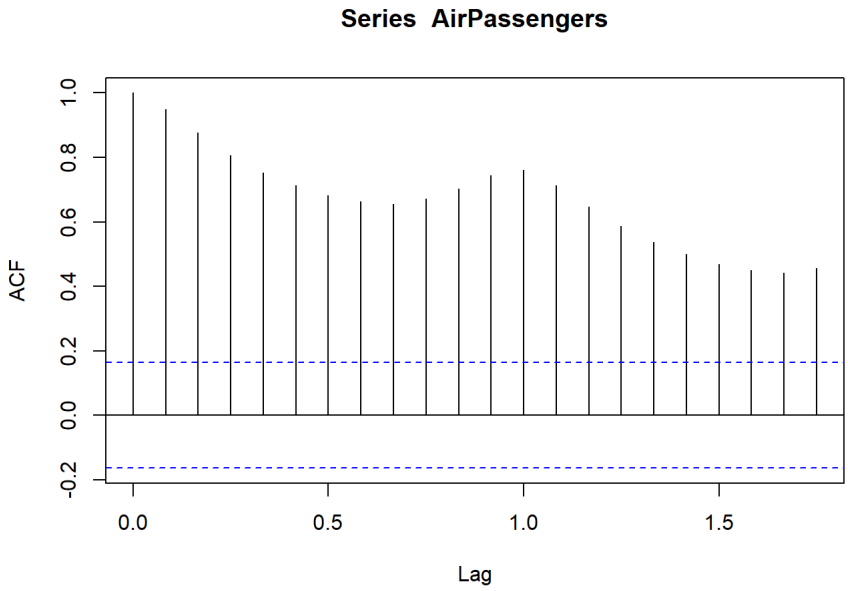
To summarize, ACF tells you 1) whether your data is stationary enough to proceed with analysis, and 2) the order of the MA(q) model.

PACF (partial autocorrelation function) plots are similar to ACF plots, but they show the partial correlation between two variables that are not explained by mutual correlations with other variables. They are useful for determining the order of the AR(q) model.

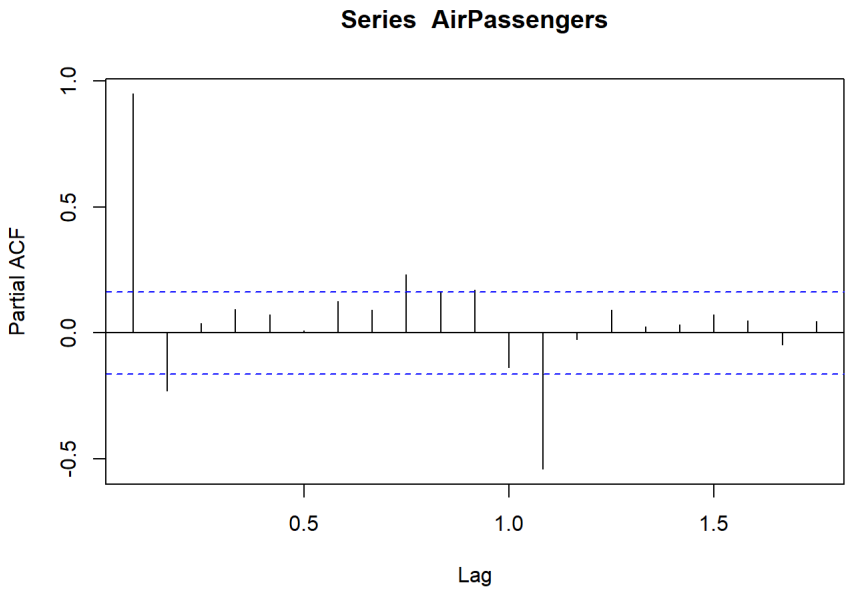
Although these concepts may be confusing at first, they're better understood with a visual.

Let's first see what happens when we do not stationarize our data.

```
acf(AirPassengers)
```

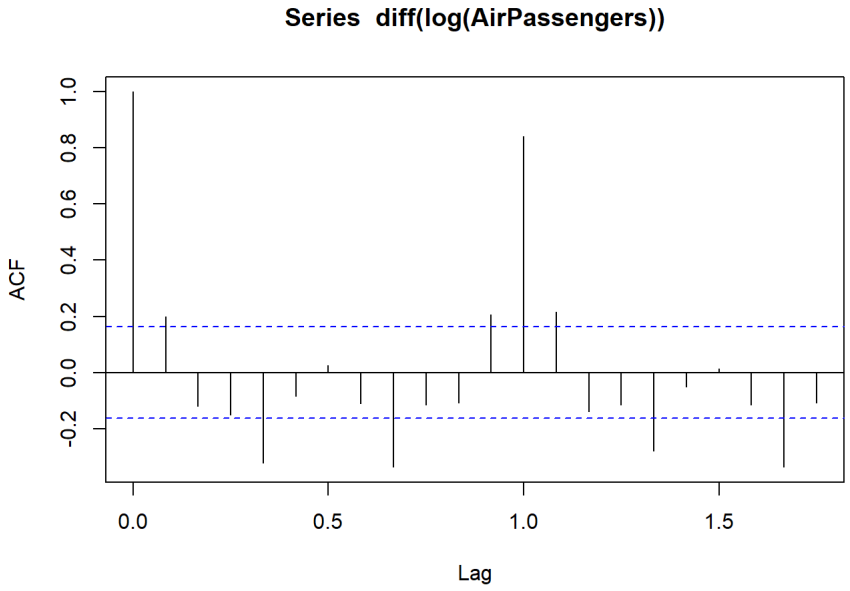



```
pacf(AirPassengers)
```

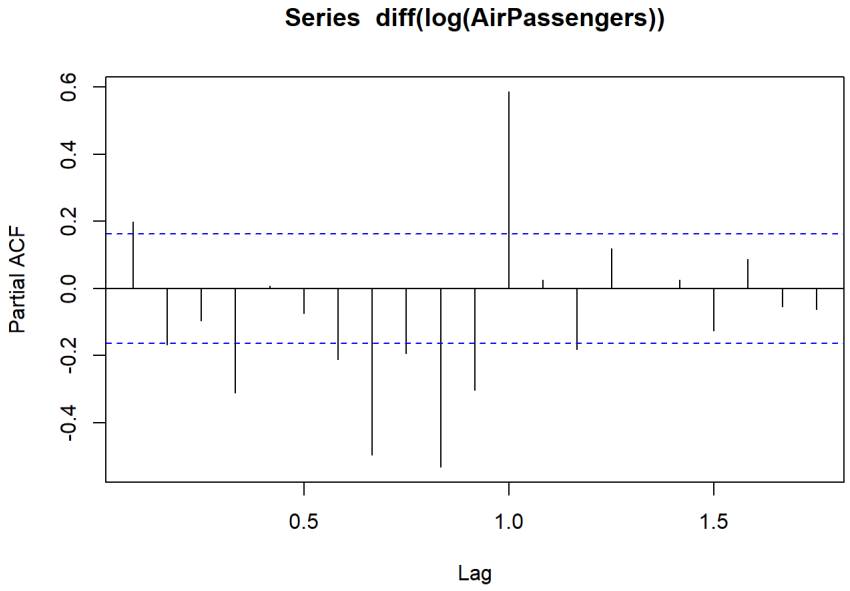


Note that the decay in the ACF chart is very slow and drops very gradually, which means that the data is not stationary. Let's now use our stationarized data.

```
acf(diff(log(AirPassengers)))
```




```
pacf(diff(log(AirPassengers)))
```



The ACF plot cuts off at the first lag. Remember, ACF is good for determining the order of the MA(q) model. This suggests that q = 1.

What about d? Since we had to take the difference of the data only once, (using the diff() function), we see that d = 1. (Had we needed to take the difference a second time, for example, d = 2)

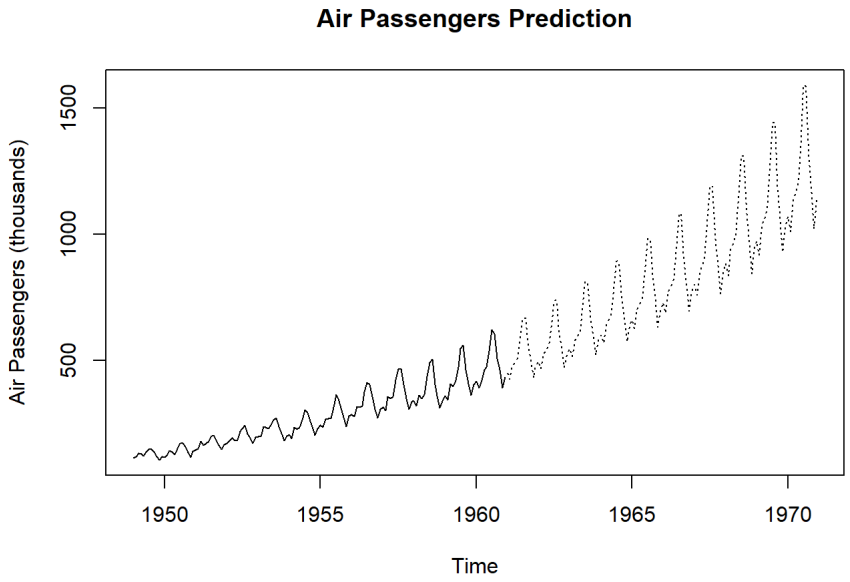
The PACF plot starts off immediately with low lag, so we can say that p = 0. Thus, we can predict that our parameters for (p, d, q) are (0, 1, 1).

Caution: The ARIMA model is what was used here, but we fitted a mixed model. Oftentimes, the strongest model uses either only AR or MA terms, as AR and MA terms might even cancel each other's effects. Thus, choosing the parameters for your model can be quite a tedious and ambiguous process.

4) Build the model and predict

Now that we have our parameters, we're going to try to model the passengers in the next decade.

```
#fits log of Air Passengers with the parameters we found above
fit <- arima(log(AirPassengers), order = c(0, 1, 1), seasonal = list(order = c(0, 1, 1), period = 12))

#predicts 120 months ahead, or 10 years
APforecast <- predict(fit, n.ahead = 10*12)

#ts.plot(allows us to plot several time series onto the same plot)
ts.plot(AirPassengers, #plots original data
        2.718^APforecast$pred, #plots predicted, reverse of log
        lty = c(1,3), #adds visual effect for ease of reading
        main = "Air Passengers Prediction",
        ylab = "Air Passengers (thousands)")
```



Let's break down the code.

fit is a value to hold the fitted ARIMA model. The function arima() takes in a time series set. Order is the (p, d, q) components found earlier, and seasonal is a list of order (as previously found) and period. We set period = 12 because our period is 1 year, or 12 months.

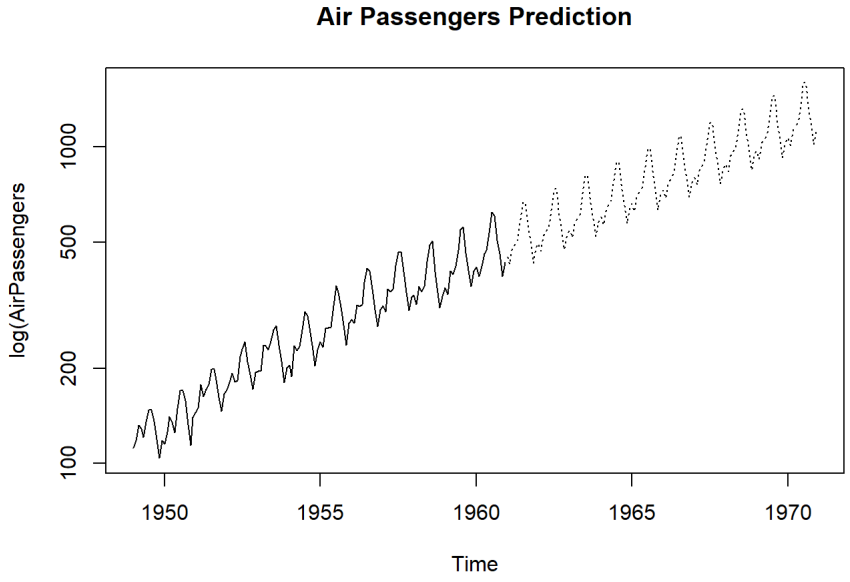
APforecast is a list of predicted values and standard errors. We want to predict the predicted values, so we'll be using APforecast$pred.

In the last piece of code, we want to plot the current data (AirPassengers), along with the predicted data (APforecast$pred). ts.plot() allows us to plot both time series together. Because we did the ARIMA analysis on log(AirPassengers), we need to take the inverse log (take e = 2.718) of the predicted valuesin order to scale the data properly and undo the log when we initially fit() the data.

Why aren't we fitting diff(log(AirPassengers))? The differencing is already accounted for in the three integer components (p, d, q).

The last parameter, lty (or LineTYpe) sets the original time series to a solid line and the predicted time series to a dotted line. This parameter is only included for visualization purposes. If we want to see the values on a log scale, we can just scale the y-axis as follows:

```
ts.plot(AirPassengers,
        2.718^APforecast$pred,
        log = "y",
        lty = c(1,3),
        main = "Air Passengers Prediction",
        ylab = "log(AirPassengers")
```



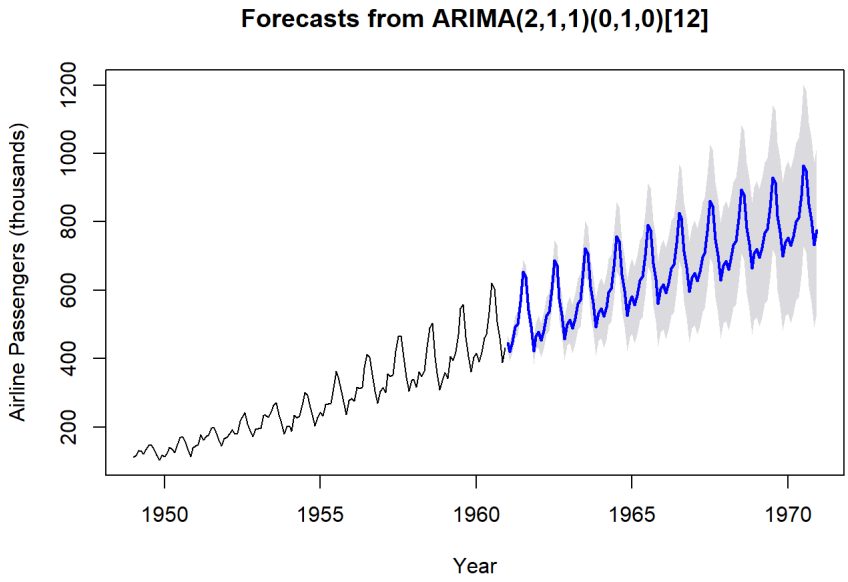
That was a lot of work to create the model. For a faster, more general method, you can use the function auto.arima(), which is part of the forecast package, to find the best ARIMA model (i.e., find the best parameters.)

```
find_best <- auto.arima(AirPassengers)
find_best
```

```
## Series: AirPassengers
## ARIMA(2,1,1)(0,1,0)[12]
##
## Coefficients:
##          ar1     ar2      ma1
##       0.5960  0.2143  -0.9819
## s.e.  0.0888  0.0880   0.0292
##
## sigma^2 estimated as 132.3:  log likelihood=-504.92
## AIC=1017.85   AICc=1018.17   BIC=1029.35
```

Now, let's try plotting it. If we set level = 95, we find the 95% confidence level for the predictions.

```
plot(forecast(find_best, h=12*10, level = 95),
     xlab = "Year",
     ylab = "Airline Passengers (thousands)")
```



We've now seen the basic framework for conducting time series analysis. Of course, there are much more complicated and detailed techniques than I've listed here, but the procedure is essentially the same.

1. Explore your data
2. Stationarize your data
3. Determine parameters for the model
4. Predict and build the model

Conclusion

With these techniques, we're only beginning to enter the realm of time series analysis. Of course, to fully understand the underpinnings of the model, a much deeper statistical background is required, and taking Stats 153 (Time Series) is recommended. However, you should now have a basic understanding of the components and steps necessary to take in order to predict future trends based off of past data. This information is applicable across different fields, from forecasting product sales to predicting the policy implications. At the core, R is our main vehicle and platform for conducting this analysis (though, of course, other data analytics software are capable as well.)

References

Introduction to Forecasting with ARIMA in R

Using R for Time Series Analysis

A Gentle Introduction to Autocorrelation and Partial Autocorrelation

A Complete Tutorial on Time Series Modeling in R

How to check in R if time series is stationary?

Identifying the numbers of AR or MA terms in an ARIMA model

Time Series Analysis

Air Passengers Forecast by Noha Elprince