Laboratory 4 ARIMA MODEL

ANA 535 Forecasting

Name KOHEI NISHITANI Student ID# 1122867

Date May 8 , 2025

# Table of Contents

* Introduction - 3
* Background - 3
* Data - 3
* Methods and Procedures - 4
* Results - 11
* Conclusions - 11
* References - 12
* Attachment or Appendices - 13

# Introduction

Lab 4 takes Amtrak-forecasting one step further from Lab 3’s deterministic smoothing to a complete ARIMA model workflow. Major simulations that demonstrate how AR(0), AR(1 ± drift), and MA(1) models form ACF/PACF profiles are followed by post-practicing sessions with earthquake counts and Google closing prices, Then Amtrak passenger-miles is the emphasis at step2. ACF plot, Augmented Dickey–Fuller and KPSS tests reveal the non-stationary feature which is elimination of the same through differencing and seasonal adjustment. This exercise ends up with choosing the best ARIMA (p, d, q) that reflects the more precise forecasting.

**Background**

Previously labs used linear regression, moving average’s filters, and exponential-smoothing families’ (ETS, Holt–Winters) turn the seasonally adjusted Amtrak passenger-mile series into forecasts. While those deterministic techniques addressed the issue of trend and seasonality, residual correlograms uncovered significant autocorrelation, indicating that serial dependence was unaccounted for. Lab 4 fills this gap by ARIMA model and auto.arima(), thus making it possible for the random component of the series to be systematically dependent. Once stationarity is obtained, candidate AR, MA, and ARIMA specifications are estimated over 1991-2016 and their forecast accuracy is evaluated against the deterministic models used previously.

**Data**

Besides of Amtrak data which is used in earlier Lab, Two real-world benchmark sets complement the simulations at Lab4. The **earthquake dataset** consists of 297 data points representing distinct seismic events recorded between 1933 and 1976; it is read from a cleaned text file and converted to an equally-spaced time-series object before analysis.

Finally, the **Google daily closing-price dataset** spans roughly five calendar years of trading activity.

**Methods and Procedures**

Step 1 begins by generating three benchmark processes—white noise, AR (1), and MA (1)—and then appends a drifted AR (1) variant before turning to two real-world series. White noise (Figures 1–2) provides the reference case: the series fluctuates constantly, the ACF shows no significant lags, the Augmented Dickey–Fuller (ADF) test yields p ≈ 0.01, and the KPSS test returns p > 0.10—mutually supporting stationarity despite their opposite null hypotheses.

A graph showing a sound wave

AI-generated content may be incorrect.A graph with lines and numbers

AI-generated content may be incorrect.

Figure 1 Figure 2

In the AR (1) simulation (e2) the ACF decays quickly while the PACF truncates after lag 1, exactly reflecting phi = 0.6. Unit-root tests again confirm stationarity. The MA (1) series (e3) exhibits the complementary pattern—one dominant ACF spike and negligible PACF lags (Figures 3)—and is likewise stationary.

A group of graphs showing different types of data

AI-generated content may be incorrect.

Figure 3

Introducing a model with drift term produces ambiguity(Figure 4-1): the ADF test still rejects null-hypothesis(p-val = 0.01), it means this is stationary. Whereas KPSS now rejects null hypothesis, it results accept alternative hypothesis, it means non-stationarity(Table 4-2).

A graph of different types of waves

AI-generated content may be incorrect.A computer screen shot of a program

AI-generated content may be incorrect.

Figure 4-1 Table 4-2

This exercise demonstrates how real-world data behave at time-series analysis. The earthquake magnitudes (Figure 5) illustrate challenges posed by irregular sampling. Decomposition fails without a regular frequency. For Google closing prices (Figure 6), it demonstrates contradicted findings. ADF p-value of 0.63 and it indicates non-stationarity, however PACF drops to insignificance after lag 1, it indicates AR(1) may fits well.

A graph of a graph of a graph

AI-generated content may be incorrect.A graph of a stock market

AI-generated content may be incorrect.

Figure 5 Figure 6

To test whether auto ARIMA model return the same model as manually developed model. As a result e2 the stepwise and searches both return ARIMA (1, 0, 0); for e3 even ACF and PACF shows huge down spike at lag1, ARIMA model didn’t catch that; it demonstrates the importance of double-checking (Figures 7–8).

A graph of a graph of a graph

AI-generated content may be incorrect.A graph of a graph of a graph

AI-generated content may be incorrect.

Figure 7 Figure 8

ARIMA model with earthquake dataset; the auto ARIMA favors stepwise ARIMA (0, 0, 3) and ARIMA(0,1,1) over a simple AR (1) (Figure 9). Regarding Google stock dataset, Auto-selection produces a three-way tie—ARIMA (3, 1, 0), ARIMA (2, 1, 3), and ARIMA (2, 1, 4) all share the minimum AICc = 5861, while the simpler ARIMA (1, 1, 1) is only one point higher (delta AICc = 1). Because all model has similar performance, any model can be acceptable.

A screenshot of a computer code

AI-generated content may be incorrect.

Table 9

A screenshot of a computer code

AI-generated content may be incorrect.

Table 10

At Lab4 step2, Amtrak data is handled by basic data transformation and subtract and decompose seasonal, trend component at the beginning(Figure-11). From earlier lab, this data was not stationary, but 3 month MA is. This lab exercises ADF test to determine whether this original data is stationary, surprisingly the result is p-value 0.01, it is stationary over 1991-2016(Table 12).

A graph of different types of time

AI-generated content may be incorrect.

Figure 11

A blue screen with white text

AI-generated content may be incorrect.

Table 12

An initial Augmented Dickey–Fuller (ADF) test produced p = 0.01, rejecting the unit-root null, whereas the KPSS trend test returned p = 0.01, rejecting trend-stationarity. This contradicted outcome that unremoved trend or seasonal structure still exists the data, a conclusion reinforced by unitroot\_ndiffs(), which advised one difference (Table 13). A single seasonal difference (lag = 12) largely suppressed the annual cycle: the ADF still rejected non-stationarity, yet KPSS remained borderline at p ≈ 0.04 and unitroot\_ndiffs() continued to request an additional difference.

A blue background with yellow text

AI-generated content may be incorrect.

Table 13

Applying one ordinary difference resolved the conflict; KPSS rose to p ≥ 0.10, the ADF p value became 0.01, and unitroot\_ndiffs()dropped to zero, indicating consensus that the twice-differenced series is stationary.

A graph of different types of time

AI-generated content may be incorrect.

Figure 14

After confirming stationarity with two differences, the twice-differenced Passenger-Miles series shows surprising swings around 2014 and 2015, even still previous study tells data is stationary(Figure 15). At step 2, it introduced one example ways of ARIMA model, Forecast package. An automatic specification was first obtained with auto.arima(), it yields ARIMA(5, 0, 4) (Table 16). After this model selection, ARIMA(5, 0, 4) forecasted the next 5years with forecast() (Figure 17).

A graph showing a graph of a graph

AI-generated content may be incorrect.

Figure 15

A screenshot of a computer

AI-generated content may be incorrect.

Table 16

A graph of a wave

AI-generated content may be incorrect.

Figure 17

To illustrate that ARIMA can be estimated without explicit pre-differencing, the original 1991-2016 series was re-analysed with auto.arima, returning a seasonal model ARIMA(0, 1, 3)(2, 1, 1) and forecasting. (Table 18 & Figure 19)

A computer screen shot of numbers and letters

AI-generated content may be incorrect.

Table 18

A graph of a graph showing the weather

AI-generated content may be incorrect.

Figure 19

Results

Differencing first and modelling later led auto.arima() to an ARIMA (5, 0, 4): because the data were already made stationary (one seasonal plus one regular difference taken outside the model), all remaining dependence had to be soaked up by short-memory terms, hence five AR and four MA lags.

Running auto.arima on the undifferenced series let the algorithm decide its own integration. It applied one regular and one seasonal difference internally and then needed only a modest remainder, selecting ARIMA (0, 1, 3)(2, 1, 1)[12]. The built-in differencing removes trend and annual seasonality, so three monthly MA lags plus two seasonal AR and one seasonal MA term suffice.

**Conclusions**

Lab 4 Step 1 used simulated and real series to link ACF/PACF patterns with ADF and KPSS results, showing that disagreement between the two tests flags hidden trend or seasonality and that agreement confirms stationarity. Step 2 applied this logic to Amtrak Passenger-Miles. ADF and KPSS conflicted on the raw data; one seasonal plus one regular difference resolved the issue, giving a stationary series. Auto-arima on this twice-differenced input chose ARIMA (5, 0, 4) because all persistence had to be captured by AR and MA terms. Running auto-arima on the undifferenced data instead let the algorithm do its own differencing and produced a leaner ARIMA (0, 1, 3)(2, 1, 1)[12].

The exercise illustrates that stationarity can be secured either before modelling or inside the ARIMA framework—so long as diagnostics are respected and automatic selections are cross-checked.

**References**

**Galit Shmueli. (2016) Practical Time Series Forecasting with R: A Hands-On Guide**

**Hyndman, R.J., & Athanasopoulos, G. (2021) Forecasting: principles and practice, 3rd edition, OTexts: Melbourne, Australia. OTexts.com/fpp3  
Quick Way to Find P, D, and Q Values for ARIMA Analytics India Magazine. https://analyticsindiamag.com/ai-trends/quick-way-to-find-p-d-and-q-values-for-arima/**

**Script**

#

#Script for ANA 535 Laboratory #4 Step 1

#

#Written by Marvine Hamner April 2025

#

library(xlsx)

library(fpp3)

library(dplyr)

library(tidyverse)

library(ggplot2)

library(tsibble)

library(tsibbledata)

library(fable)

library(feasts)

library(forecast)

library(lubridate)

library(zoo)

library(tseries)

# install.packages("pak") # if you don’t have it yet

# pak::pak("KevinKotze/tsm")

library(tsm)

library(ggpubr)

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1a \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

#

#First generate the data for white noise. Use the rnorm() function

#to do that. Use mean = 0, variance = 4.0, standard deviation = 2.0

#Generate 1,000 data points to start with.

n = 1000

e = rnorm(n, mean = 0.0, sd = 4.0)

par(mfrow = c(1,1))

plot(e, type = "l")

#

#How do we handle time in this case? Well, we can assume this is

#a signal we are sampling and just establish a sampling frequency

#to get time. But, we can do this an easier way when we setup the

#time series object in R. We just need to get the sampling frequency

#right. If we say we are sampling at 10 Hertz then we would have 100

#seconds of data. Let's try that.

#

e.ts <- ts(e, frequency = 10)

str(e.ts)

plot(e.ts, xlab = "Time (seconds)", ylab = "Magnitude", bty = "l")

autoplot(e.ts) +

labs(title = "White-noise series e",

x = "Time (seconds)",

y = "Magnitude")

ggsave("Fig01\_white\_noise\_ts.png", dpi = 300, width = 4, height = 3)

#

#Now, let's consider the ACF. If the ACF has no significant spikes

#that would indicate that we should be able to fit an AR(0) model

#to these data.

#

ggAcf(e.ts) +

ggtitle("Sample ACF for e sim data")

ggsave("Fig02\_white\_noise\_acf.png", dpi = 300, width = 4, height = 3)

#

#This looks really good for white noise. There are no significant

#spikes in the ACF. This looks good for an AR(0) model. What other

#diagnostic tests should we run? Well... we should run a unit root

#test. Section 9.7 in out textbook shows graphically the steps in

#time series analysis and forecasting.

#

e.ts.adf <- adf.test(e.ts)

e.ts.adf

#

#Here we have used the Augmented Dickey-Fuller (unit root) test.

#Since the p-value is less than 0.05 we can reject the null

#hypothesis and conclude that the data are stationary. Let's

#try the KPSS test and see if, in this case, it is consistent

#with the ADF test.

#

e.ts.kpss <- kpss.test(e.ts)

e.ts.kpss

#

#The p-value for the kpss test is greater than 0.05 so we reject

#the null hypothesis and, yes, the result is consistent between

#the ADF test and the KPSS test. When you are using these unit

#root tests remember that the null hypotheses are the opposite.

#For the ADF test the null hypothesis is that the data are non-

#stationary. For the KPSS test the null hypothesis is that the

#data are stationary. For the ADF test the p-value is 0.01 so we

#reject the null hypothesis that the data are non-stationary, i.e

#we accept that there is evidence that the data are stationary.

#For the KPSS test the p-value is 0.10 or greater than 0.05 and

#we cannot reject the null hypothesis that the data are stationary.

#

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1b \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

#

#Let's look at another (artificial) dataset using the rnorm()

#function to generate data. We want to generate a data that we

#can fit an AR(1) model to.

#

y <- numeric(1000)

e2 <- rnorm(1000)

for(i in 2:1000)

y[i] <- 0.6\*y[i-1] + e[i]

e2.tsb <- tsibble(idx = seq\_len(1000), y = y, index = idx)

e2.tsb

e2.ts <- ts(e2.tsb[,2], frequency = 10)

autoplot(e2.ts)

str(e2.ts)

#

#Now let's look at the ACF, and if necessary the PACF

#

e2Acf <- ggAcf(e2.ts) +

ggtitle("Sample ACF for e2 sim data")

plot(e2Acf)

e2Pacf<- ggPacf(e2.ts) +

ggtitle("Sample PACF for e2 sim data")

plot(e2Pacf)

#

#The ACF does quickly taper off to insignificant spikes. This is

#the behavior we want to see in the ACF. This tells us that an AR(1)

#model may be a good fit for these data. Double-checking this result

#with the PACF we do see that only the first spike is significant.

#Therefore, I would try to fit an AR(1) model to these data first.

#

#

#What about stationarity? Let's run the ADF and KPSS tests.

#

e2.ts.adf <- adf.test(e2.ts)

e2.ts.adf

e2.ts.kpss <- kpss.test(e2.ts)

e2.ts.kpss

#

#Once again our data appears to be stationary. The difference now

#is that we have an AR(1) or a lag=1 simulation rather than

#white noise.

#

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1d \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Note that this step and step 1c

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*are reversed in order in this script

#

#

#Moving along to MA(q) models. What do you think we need to do to

#generate data that an MA(q) model would fit well?

#

#We can write a function for this that makes it easier to change

#theta to see how the plot changes. To consider how you want to

#change theta look at the bottom of the page, Section 9.4.

#First set theta equal to 0.6.

#

e3.ts <- arima.sim(model = list(ma = 0.6), n = 1000)

plot.ts(e3.ts)

str(e3.ts)

#

#Compare to the AR(1) model with phi equal to 0.6.

#

e3Acf <- ggAcf(e3.ts) +

ggtitle("Sample ACF for e3 sim data")

plot(e3Acf)

e3Pacf <- ggPacf(e3.ts) +

ggtitle("Sample PACF for e3 sim data")

plot(e3Pacf)

#

#Don't forget to run the unit root tests!

#

e3.ts.adf <- adf.test(e3.ts)

e3.ts.adf

e3.ts.kpss <- kpss.test(e3.ts)

e3.ts.kpss

#

#Now let's compare the ACF and PACF for the simulated data e2 and

#e3. To compare the plots its easier if they are all part of one

#figure. Because the plots were all generated with ggplot we cannot

#just use par(mfrow). To do this plotting I've installed the

#ggpubr package and am using the ggarrange() command. But...

#just to make sure that the simulation generating the data is doing

#what we think it is let's use the tsm package to generate data

#for the AR(1) model too.

#

install.packages("ggpubr") # only the first time

library(ggpubr) # every new session

e2b.ts <- arima.sim(model = list(ar = 0.6), n = 1000)

plot.ts(e2b.ts)

str(e2b.ts)

e2bAcf <- ggAcf(e2b.ts) +

ggtitle("Sample ACF for e2 sim data")

plot(e2bAcf)

e2bPacf<- ggPacf(e2b.ts) +

ggtitle("Sample PACF for e2 sim data")

plot(e2bPacf)

#

#Now put the plots together for a comparison

#

fig3 <- ggarrange(e2bAcf, e2bPacf, e3Acf, e3Pacf, ncol = 2, nrow = 2)

fig3

# ggsave("Fig03\_ar\_vs\_ma.png", plot = fig3, dpi = 500, width = 4, height = 3)

#

#What I see is what I expected to see. The ACF falls off quickly

#for the simulated data for AR(1) while the ACF for the simulated

#data for MA(1) only has 1 large spike at lag = 1. So, it seems

#that Kevin Kotzel has made a really nice package. What I wanted

#was the ability to easily generated simulated data for a variety

#of situations; AR(0), AR(1), and MA(1). This we have now.

#

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

#

#Now let's go back and pick up the AR(1) with drift model. Remember

#that the drift is added as a constant to the equation. We'll add

#c = 10.0 for the drift. I played around with several different values

#for c but couldn't really see a lot of effect in the data or trend.

#It seems to me that adding a positive constant should just raise

#the trend line overall. To really understand the effect of drift will

#take some more specialized study...

#

c = 10.0

y <- numeric(1000)

e2wd <- rnorm(1000)

for(i in 2:1000)

y[i] <- c + 0.6\*y[i-1] + e[i]

e2wd.tsb <- tsibble(idx = seq\_len(1000), y = y, index = idx)

e2wd.tsb

e2wd.ts <- ts(e2wd.tsb[,2], frequency = 10)

autoplot(e2wd.ts)

str(e2wd.ts)

e2wd.comp <- decompose(e2wd.ts)

autoplot(e2wd.comp)

#

#Now let's look at the ACF, and if necessary the PACF

#

e2wdAcf <- ggAcf(e2.ts) +

ggtitle("Sample ACF for e2 sim data")

plot(e2Acf)

e2wdPacf<- ggPacf(e2.ts) +

ggtitle("Sample PACF for e2 sim data")

plot(e2Pacf)

#

#The ACF does quickly taper off to insignificant spikes. This is

#the behavior we want to see in the ACF. This tells us that an AR(1)

#model may be a good fit for these data. Double-checking this result

#with the PACF we do see that only the first spike is significant.

#Therefore, I would try to fit an AR(1) model to these data first.

#

#

#Now put the AR(1) and AR(1) with drift plots together for a

#comparison.

#

ggarrange(e2Acf, e2Pacf, e2wdAcf, e2wdPacf, ncol = 2, nrow = 2)

#

#What about stationarity? Let's run the ADF and KPSS tests.

#

e2wd.ts.adf <- adf.test(e2wd.ts)

e2.ts.adf

e2wd.ts.kpss <- kpss.test(e2wd.ts)

e2wd.ts.kpss

#

#The thing I really see changing is the results of the unit root

#tests. These are very different than the results before without

#drift. Now, here is the case where the ADF test result is

#inconsistent with the KPSS test result. The ADF test says that we

#reject the null hypothesis (p-value = 0.01) and say there is

#sufficient evidence to accept the alternate hypothesis. That is,

#that the data are stationary. But, the KPSS test result also says

#reject the null hypothesis. Since these null hypotheses are

#opposite we would need to get opposite test results to be

#consistent. In the KPSS test result we would accept the alternate

#hypothesis and say the data are non-stationary. This means we

#need more testing!

#

#

#Here is a bit more practice. Let's look at the results of some

#real-world data; earthquakes and Google stock closing prices.

#

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1e quakes data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

#

#Let's work a bit with the Earthquake data. I've made the Excel

#spreadsheet I downloaded for earthquakes from 1933 to to 1994.

#There are almost 16,000 observations in this dataset. But many of

#them are related to the same earthquake. I started by taking the

#first observation of every earthquake that was obviously different

#than the one before it. I got 297 observations from 1933 to 1976

#and stopped due to the time available. It is an interesting

#dataset. Let's see what we can tell from it.

#

quakes.v2.txt <- read.csv("/Users/nkohei/Workspace/McDaniel-Repository/535/lab4/quakes.csv")

quakes <- quakes.v2.txt

View(quakes)

colnames(quakes) <- c("Year", "Mag", "Name", "State", "Country")

plot(quakes[,2], type = "l")

quakes.ts <- ts(quakes[,2])

str(quakes.ts)

#

#There isn't any data in the first record so we'll eliminate it.

#

quakes.ts <- ts(quakes.ts[2:297])

str(quakes.ts)

plot(quakes.ts, bty = "l")

par(mfrow=c(1,2))

quakes\_tslm <- tslm(quakes.ts ~ trend)

quakes\_tslm

plot(quakes.ts, ylab = "Magnitude", bty = "l")

lines(quakes\_tslm$fitted, lwd = 2, col = "red")

quakes\_tslm2 <- tslm(quakes.ts ~ trend + I(trend^2))

plot(quakes.ts, ylab = "Magnitude", bty = "l")

lines(quakes\_tslm2$fitted, lwd = 2, col = "blue")

df\_quakes <- tibble(idx = seq\_along(quakes.ts),

mag = as.numeric(quakes.ts),

lin\_fit = fitted(quakes\_tslm),

quad\_fit = fitted(quakes\_tslm2))

plot\_lin <- ggplot(df\_quakes, aes(idx, mag)) +

geom\_line() +

geom\_line(aes(y = lin\_fit), colour = "red", linewidth = 1) +

labs(title = "Linear trend fit", y = "Magnitude") +

theme\_bw()

plot\_quad <- ggplot(df\_quakes, aes(idx, mag)) +

geom\_line() +

geom\_line(aes(y = quad\_fit), colour = "blue", linewidth = 1) +

labs(title = "Quadratic trend fit", y = "Magnitude") +

theme\_bw()

plot\_acf <- ggAcf(quakes.ts) + ggtitle("Sample ACF for Quake Magnitudes")

plot\_pacf <- ggPacf(quakes.ts) + ggtitle("Sample PACF for Quake Magnitudes")

fig5 <- ggarrange(plot\_lin, plot\_quad, plot\_acf, plot\_pacf,

ncol = 2, nrow = 2)

fig5

#A really big problem with these data are that the time interval

#is not the same between each earthquake, nor should it be. But

#our codes are made to do time series analyses on times that are

#equal between data points. So, we'll see what we can get

#but I wouldn't put too much into this.

#

ggAcf(quakes.ts) +

ggtitle("Sample ACF for Quake Magnitudes")

ggPacf(quakes.ts) +

ggtitle("Sample PACF for Quake Magnitudes")

#

#This looks an awful lot like the e2 simulated data, i.e. the

#data we can fit an AR(1) model with and without drift to. Let's

#see if we can continue to blissfully ignore this big time problem

#and decompose these data. And, no! We run into trouble because of

#the time problem here. We cannot specify a correct frequency.

#Without that there is no way for the code to compute a trend.

#

quakes.ts.comp <- decompose(quakes.ts) #intentionally failed?

autoplot(quakes.ts.comp)

#

#Penn State's online course STAT 510 does go ahead and make some

#observations from just looking at the time plot of the data. I

#don't know if I would do that or not. I cannot really see if

#there is any periodicity in these data. And, as Penn State says

#there do tend to be swarms of earthquakes so that sometimes there

#is some autocorrelation in the data.

#

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1f Google data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

#

#Let's start looking at another dataset, i.e. the data for Google's

#closing stock prices for the last 5 years. I've also uploaded that

#data file for you on Blackboard. Unfortunately, we are going to have

#some of the same trouble with the Google data that we had with the

#earthquake data. That is, the time intervals are not equal. Stock

#prices are recorded daily for every day the stock market is open.

#But, the stock market is not open every day of the year. Let's see

#how far we can get.

#

GoogleStockData.5yr.v2 <- read.csv("/Users/nkohei/Workspace/McDaniel-Repository/535/lab4/GoogleStockData-5yr.csv")

google <- GoogleStockData.5yr.v2

View(google)

colnames(google) <- c("Date", "Close", "Volume", "Open", "High", "Low")

str(google)

#

#Change the Date variable from a char format to a Date format. This

#is going to be a bit difficult because the time intervals in the

#data are not equal. The stock market doesn't operate 7 days per

#week. So, we'll have to see how to best handle that.

#

google$Date <- mdy(google$Date)

str(google)

head(google)

google.ts <- ts(google$Close)

str(google.ts)

par(mfrow = c(1,1))

plot(google.ts, ylab = "Google Closing Price", xlab = "Time", bty = "l")

#

#Let's see what the Augmented Dickey-Fuller (unit root) test tells us.

#

google.adf <- adf.test(google.ts)

google.adf

#

#The p-value is 0.6336 which is greater than 0.05 we cannot reject the

#null hypothesis which in the ADF test is that the time series is non-

#stationary. So, we will need to do something to make the time series

#stationary before we can proceed with our time series analysis. What

#about the ACF?

#

ggAcf(google.ts) +

ggtitle("Sample ACF for Google Stock Closing Prices")

ggPacf(google.ts) +

ggtitle("Sample PACF for Google Stock Closing Prices")

#

#This is somewhat inconsistent. The ACF plot indicates that the data

#are non-stationary which agrees with the ADF test results. But, the

#PACF test indicates that an AR(1) model may fit the data well.

#But, we know that to use an AR(p) model the data MUST be stationary!

#

#

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Step 1g \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

#

#Let's see the the auto ARIMA() function returns the same model

#parameters we have decided to use in the first parts of Step 1. We

#know the simulated data for "e" is white noise so we won't work

#on that particular case. Let's start with e2.

#

#

#First we'll plot the data again.

#

e2.tsb |>

autoplot() +

labs(title = "Simulated data 'e2'", y = "Magnitude of sim")

#

#The data goes up and down. If we add a trendline we can see that more

#easily. At any rate, it does seem to be stationary. Let's go to the

#next step and look at the difference and the ACF and PACF plots.

#

e2.tsb |>

gg\_tsdisplay(difference(y), plot\_type = 'partial')

e2\_fit <- e2.tsb |>

model(stepwise = ARIMA(y),

search = ARIMA(y, stepwise = FALSE))

e2\_fit

e2.tsb |>

gg\_tsdisplay(difference(y), plot\_type = "partial")

# auto-ARIMA, stepwise and exhaustive

e2\_fit <- e2.tsb |>

model(stepwise = ARIMA(y),

search = ARIMA(y, stepwise = FALSE))

report(e2\_fit) # confirms ARIMA(1,0,0)

#

#When you print out the ARIMA automatic models the parameters p, d, and

#q are the same as those we determined when we first examined the

#ACF and PACF for e2. Let's try the MA(1) model data now, the e3 data.

#

e3.tsb <- tsibble(

idx = seq\_len(length(e3.ts)),

y = as.numeric(e3.ts),

index = idx

)

e3.tsb |>

autoplot() +

labs(title = "Simulated data 'e3'", y = "Magnitude of sim")

e3.tsb |>

gg\_tsdisplay(difference(y), plot\_type = 'partial')

e3\_fit <- e3.tsb |>

model(arima001 = ARIMA(y ~ pdq(0,0,1)),

stepwise = ARIMA(y),

search = ARIMA(y, stepwise=FALSE))

report(e3\_fit)

#

#This is interesting. The ACF plot shows the same large (but negative)

#spike at lag = 1. However, the automatic ARIMA code doesn't pick this up

#as an MA(1) model. It looks like all the models built are very close in

#terms of accuracy. ARIMA(0,0,0) is again AR(0) or white noise. So,

#caveat emptor! When using automatic codes it is always best to double-

#check their output against something you know is true.

#

#

#Let's look at the quakes data using the automatic ARMIA code. Remember

#that we had determined the quakes data were best fit with an AR(1)

#model.

#

quakes.tsb <- as\_tsibble(quakes.ts)

quakes.tsb |>

autoplot() +

labs(title = "Differences for Earthquake Data", y = "")

#

#The data goes up and down. If we add a trendline we can see that more

#easily. At any rate, it does seem to be stationary. Let's go to the

#next step and look at the difference and the ACF and PACF plots. Note

#that the variables in the tsibble are indx and value. So use value in

#the commands in the code this time. Also, for some reason the first

#observation value is "NA" so I just started from the 2nd observation.

#

quakes.tsb |>

gg\_tsdisplay(difference(value[2:297]), plot\_type = 'partial')

quakes\_fit <- quakes.tsb |>

model(arima100 = ARIMA(value[2:297] ~ pdq(1,1,0)),

stepwise = ARIMA(value[2:297]),

search = ARIMA(value[2:297], stepwise = FALSE, approximation = FALSE))

report(quakes\_fit)

#

#We get the same answer whether we run a stepwise solution or just do

#a search. Both the stepwise and the search return ARIMA models that are

#better than the AR(1) model or the AR(1) with 1 diference, i.e. setting

#d = 1 in the command. It looks like we've got an ARIMA(0,0,3) model as

#best. Try playing around a bit with this and see what you can find! Now

#let's try the Google data. Remember that we found the Google data were

#best fit with an AR(1) model. However, because it was non-stationary

#we will likely have to take at least 1 difference.

#

google.tsb <- as\_tsibble(google.ts)

google.tsb |>

autoplot() +

labs(title = "Differences for Google Closing Data", y = "")

#

#The data goes up and down. If we add a trendline we can see that more

#easily. At any rate, it does seem to be stationary. Let's go to the

#next step and look at the difference and the ACF and PACF plots.

#

google.tsb |>

gg\_tsdisplay(difference(value), plot\_type = 'partial')

google\_fit <- google.tsb |>

model(arima110 = ARIMA(value ~ pdq(1,1,0)),

arima111 = ARIMA(value ~ pdq(1,1,1)),

arima310 = ARIMA(value ~ pdq(3,1,0)),

stepwise = ARIMA(value),

search = ARIMA(value, stepwise = FALSE, approximation = FALSE))

report(google\_fit)

#

#I tried this taking 1 difference and then 2 differences and both gave

#the same result ARIMA(1,1 or 2, 0). Since both the ACF and PACF show a

#significant lag at 3, I tried ARIMA(3,1,0) which was worse and actually

#both the automatic answers were worse too. I don't know why they would

#have changed. But it looks like the ARIMA(1,1,1) has the best AICc.

#

#

#Laboratory 4 ARIMA Models

#

#Written by Marvine Hamner April 2024 updated April/May 2025

#

#

#Load all the necessary libraries

#

library(xlsx)

library(fpp3)

library(dplyr)

library(tidyverse)

library(ggplot2)

library(tsibble)

library(tsibbledata)

library(fable)

library(feasts)

library(forecast)

library(lubridate)

library(zoo)

#

#Set the working directory and read in the data - if you are not continuing with

#the data already loaded!

#

Amtrak <- read.csv("/Users/nkohei/Workspace/McDaniel-Repository/535/lab4/Amtrak1991-2024.csv")

colnames(Amtrak) <- c("Month", "Ridership", "PassengerMiles", "RidersReported")

View(Amtrak)

str(Amtrak)

Amtrak

#

#Take care of the date format and data type

#

Amtrak$Month <- mdy(Amtrak$Month)

Amtrak

#

#To make things a little easier in the lab let's divide the passenger miles by

#1000000. so that we are working with a number of Passenger Miles in the 1M's

#

Amtrak$PassengerMiles <- Amtrak$PassengerMiles/1000000.

Amtrak

Amtrak.ts.91.16 <- ts(Amtrak$PassengerMiles, start = c(1991, 1), end = c(2017, 1), frequency = 12)

Amtrak.tsb.91.16 <- as\_tsibble(Amtrak.ts.91.16)

#

#Generate a 3-month centered moving average model for the Amtrak data (all years).

#Do we need to deseasonalize the data? Consider the ACF

#

Amtrak.tsb.91.16 |>

ACF() |> autoplot() +

labs(subtitle = "Amtrak Passenger Miles")

#

#But first we'll remove the seasonality in the data. But we only want to use the

#years 1991 through and including 2016.

#

Amtrak.ts.91.16 <- ts(Amtrak$PassengerMiles, start = c(1991, 1), end = c(2017, 1), frequency = 12)

Amtrak.comp.91.16 <- decompose(Amtrak.ts.91.16)

autoplot(Amtrak.comp.91.16)

str(Amtrak.comp.91.16)

#

#Now detrend and/or deseason the data as required

#

#Subtract the seasonal component

Amtrak.ts.91.16\_desea <- Amtrak.comp.91.16$x - Amtrak.comp.91.16$seasonal

#Subtract the trend component

Amtrak.ts.91.16\_detren <- Amtrak.ts.91.16\_desea - Amtrak.comp.91.16$trend

#Decompose and plot the remaining data

Amtrak.ts.91.16\_de <- decompose(Amtrak.ts.91.16\_detren)

autoplot(Amtrak.ts.91.16\_de)

#

#Convert to a tsibble and get the ACF and PACF.

#

Amtrak.tsb.91.16\_de <- as\_tsibble(Amtrak.ts.91.16\_detren)

Amtrak.tsb.91.16\_de |>

ACF() |> autoplot() +

labs(subtitle = "Amtrak Passenger Miles")

Amtrak.tsb.91.16\_de |>

PACF() |> autoplot() +

labs(subtitle = "Amtrak Passenger Miles")

#

#Consider the components of the Amtrak data after we've taken two

#differences to detrend and deseasonalize the data. If you just

#quickly look at the seasonal component it looks like it is all still

#there. However, the magnitude of the values is 25x less than

#before. So, if you look at the ACF you can see a bit of seasonality

#but it quickly becomes insignificant. The same thing with the

#trend. Originally it went from 450 to 600 but no only ranges in the

#-5 to 10 magnitude. From the ACF and PACF plots what do you think would be

#a good model to try to fit to these data now? Or, do you think we

#should still take another difference?

#

#

#Unfortunately, no, it still looks like there are multiple spikes in the

#ACF and PACF. We'll actually take two differences a little further

#along.

#

#

#Let's consider moving averages for a little bit. We've looked at them

#before. We know that to use a moving average the data have to be

#stationary. That is, remember that to use a moving average you can't

#have any trend or seasonality. For the moving averages use the

#deseasonalized data. But don't forget to go back to the regular

#data after we've looked at the moving averages!!!

#

#

#We went from the original data and took a difference to "desea" i.e.

#deseasonalize it. Then, we went to "detren" to detrend the data.

#In the end we just used ".de" to make a little shorter name for the

#data object converted to a tsibble. So, we'll start from there.

#

Amtrak\_3MA <- Amtrak.tsb.91.16\_de

#

#Now work on the moving average smoothing beginning with the centered moving

#average. For some reason if you run the entire script all at once it

#doesn't complete a plot for the centered moving average. But you can

#run the script, highlighting from the beginning until the section for the

#trailing moving average, and the plot of the centered moving average does

#plot correctly.

#

Amtrak\_3MA.19.16 <- Amtrak\_3MA |>

mutate(

'3-MA' = slider::slide\_dbl(value, mean,

.before = 1, .after = 1, .complete = TRUE)

)

Amtrak\_3MA.19.16 |>

autoplot(Amtrak\_3MA$value) +

geom\_line(aes(y = `3-MA`), colour = "#D55E00") +

labs(y = "Passenger Miles",

title = "Amtrak Passenger Miles by Month and Centered Moving Average")

#

#Let's zoom into a few years to see exactly how the 3-month moving average

#is handling the fit. To zoom in we'll need to go back to the time

#series data object, i.e. the ".detren" data object.

#

str(Amtrak.ts.91.16\_detren)

Amtrak\_MAZoom\_2001\_03 <- window(Amtrak.ts.91.16\_detren, start = c(2001,1),

end = c(2004,1), frequency = 12)

str(Amtrak\_MAZoom\_2001\_03)

Amtrak.tsb\_2001\_03 <- as\_tsibble(Amtrak\_MAZoom\_2001\_03)

Amtrak\_3MA\_2001\_03 <- Amtrak.tsb\_2001\_03 |>

mutate(

'3-MA' = slider::slide\_dbl(value, mean,

.before = 1, .after = 1, .complete = TRUE)

)

Amtrak\_3MA\_2001\_03 |>

autoplot(Amtrak.tsb\_2001\_03$value) +

geom\_line(aes(y = Amtrak\_3MA\_2001\_03$`3-MA`), colour = "#D55E00") +

labs(y = "Passenger Miles",

title = "Amtrak Passenger Miles by Month and Centered Moving Average")

#

#Generate a 3-month trailing moving average model for the Amtrak data (all years).

#

Amtrak\_3MATR <- Amtrak\_3MA\_2001\_03 |>

mutate(

'3-MA' = slider::slide\_dbl(value, mean,

.before = 2, .after = 0, .complete = TRUE)

)

Amtrak\_3MATR |>

autoplot(Amtrak.tsb\_2001\_03$value) +

geom\_line(aes(y = Amtrak\_3MATR$`3-MA`), colour = "#D55E00") +

labs(y = "Passenger Miles",

title = "Amtrak Passenger Miles by Month and Trailing Moving Average")

#

#Going back to the original data, let's try the Augmented

#Dickey-Fuller test next.

#

library(tseries)

Amtrak\_adf <- adf.test(Amtrak.ts.91.16)

Amtrak\_adf

#

#The ADF test tells us that the data are not stationary. Let's

#work on this from another perspective, this is an example from

#Shmueli and Lichtendahl's textbook, Practical Time Series Forecasting

#with R. This reproduces Figure 5.2 on page 82.

#

#This example uses "Ridership" rather than "Passenger Miles" so first we'll

#read in the data.

#

#Take care of the date format and data type

#

Amtrak$Month <- mdy(Amtrak$Month)

Amtrak

#

#Now get the same months as we have been using, 1991 - 2016 as a time

#series and a tsibble.

#

AmtrakRiders.ts.91.16 <- ts(Amtrak$Ridership, start = c(1991, 1), end = c(2017, 1),

frequency = 12)

AmtrakRiders.tsb.91.16 <- as\_tsibble(AmtrakRiders.ts.91.16)

library(zoo)

ma.trailing <- rollmean(AmtrakRiders.ts.91.16, k=12, align = "right")

ma.centered <- ma(AmtrakRiders.ts.91.16, order = 12)

plot(AmtrakRiders.ts.91.16, ylab = "Ridership", xlab = "Time", bty = "l", xant = "n")

#axis(1, at = seq(1991, 2004.25, 1), labels = format(seq(1991, 2004.25, 1)))

lines(ma.centered, col = "steelblue", lwd = 2) # centred MA → blue

lines(ma.trailing, col = "firebrick", lwd = 2, lty = 2) # trailing MA → red, dashed

legend("topleft",

legend = c("Ridership", "Centered MA", "Trailing MA"),

col = c("black", "steelblue", "firebrick"),

lty = c(1, 1, 2),

lwd = c(1, 2, 2),

bty = "n")

#

#Now let's zoom into the same years we zoomed into before.

#

AmtrakRiders.ts.01.03 <- ts(Amtrak$Ridership, start = c(2001, 1), end = c(2004, 1),

frequency = 12)

ma.trailing <- rollmean(AmtrakRiders.ts.01.03, k=12, align = "right")

ma.centered <- ma(AmtrakRiders.ts.01.03, order = 12)

plot(AmtrakRiders.ts.01.03, ylab = "Ridership", xlab = "Time", bty = "l", xant = "n")

#axis(1, at = seq(2001.5, 2004.25, 1), labels = format(seq(2001.5, 2004.25, 1)))

lines(ma.centered, col = "steelblue", lwd = 2) # centred MA → blue

lines(ma.trailing, col = "firebrick", lwd = 2, lty = 2) # trailing MA → red, dashed

legend("topleft",

legend = c("Ridership", "Centered MA", "Trailing MA"),

col = c("black", "steelblue", "firebrick"),

lty = c(1, 1, 2),

lwd = c(1, 2, 2),

bty = "n")

ma.trailing <- rollmean(AmtrakRiders.ts.01.03, k=3, align = "right")

ma.centered <- ma(AmtrakRiders.ts.01.03, order = 3)

plot(AmtrakRiders.ts.01.03, ylab = "Ridership", xlab = "Time", bty = "l", xant = "n")

#axis(1, at = seq(2001.5, 2004.25, 1), labels = format(seq(2001.5, 2004.25, 1)))

lines(ma.centered, col = "steelblue", lwd = 2) # centred MA → blue

lines(ma.trailing, col = "firebrick", lwd = 2, lty = 2) # trailing MA → red, dashed

legend("topleft",

legend = c("Ridership", "Centered MA", "Trailing MA"),

col = c("black", "steelblue", "firebrick"),

lty = c(1, 1, 2),

lwd = c(1, 2, 2),

bty = "n")

#

#We'll go on from moving averages now to AR(p), MA(q), and ARIMA models.

#If you are like me you have a whole bunch of Amtrak data subsets. Let's clean

#up things a little so we do not make any mistakes by using the wrong data.

#First we'll remove the Amtrak data and then read it back in again.

#

# rm(list = ls(pattern = "Amtrak\*"))

#

# setwd("I:/My Passport Documents/McDaniel/DataAnalytics/ANA535/NewANA535/Laboratories")

Amtrak <- read.csv("Amtrak1991-2024.csv")

colnames(Amtrak) <- c("Month", "Ridership", "PassengerMiles", "RidersReported")

View(Amtrak)

str(Amtrak)

#Amtrak

#

#Take care of the date format and data type

#

Amtrak$Month <- mdy(Amtrak$Month)

#Amtrak

#

#To make things a little easier in the lab let's divide the passenger miles by

#1000000. so that we are working with a number of Passenger Miles in the 1M's

#

Amtrak$PassengerMiles <- Amtrak$PassengerMiles/1000000.

#Amtrak

Amtrak.ts.91.16 <- ts(Amtrak$PassengerMiles, start = c(1991, 1), end = c(2017, 1),

frequency = 12)

Amtrak.tsb.91.16 <- as\_tsibble(Amtrak.ts.91.16)

#

#Now we have our Amtrak data read in, our subset for 1991-2016 setup and

#a corresponding tsibble generated.

#

#

#We will develop models using a couple different packages in R. One from

#the textbook and another from other online sources and Schmueli's book.

#We didn't load the package tseries at the beginning so we need to load that

#now. If you haven't installed the urca package in R you will need to

#do that too. We'll also load that with the library() command now.

#

library(tseries)

library(urca)

Amtrak\_kpss <- kpss.test(Amtrak.ts.91.16, null = "Trend")

Amtrak\_kpss

Amtrak\_adf <- adf.test(Amtrak.ts.91.16)

Amtrak\_adf

#

#This is an interesting result because it gives us conflicting results. This

#is actually not that uncommon. Now let's consider the unit root test to see

#what it tells us about the number of differences we should take.

#

Amtrak\_unitroot <- unitroot\_ndiffs(Amtrak.ts.91.16)

Amtrak\_unitroot

Amtrak\_seasdiff <- Amtrak.tsb.91.16 |>

mutate(Amtrak.ts.91.16 = difference(Amtrak.ts.91.16), 12) |>

features(Amtrak.tsb.91.16, unitroot\_ndiffs)

Amtrak\_seasdiff

#

#Since we have monthly data we are going to use lag = 12 to get a seasonal

#difference.

#

Amtrak.comp.91.16b <- decompose(Amtrak.ts.91.16)

autoplot(Amtrak.comp.91.16b)

Amtrak\_seasdiff\_1 <- diff(Amtrak.ts.91.16, lag = 12)

Amtrak.comp.seasdiff.1 <- decompose(Amtrak\_seasdiff\_1)

autoplot(Amtrak.comp.seasdiff.1)

Amtrak\_seasdiff\_ndiffs\_1 <- unitroot\_ndiffs(Amtrak\_seasdiff\_1)

Amtrak\_seasdiff\_ndiffs\_1

#

#We've taken one difference as the unitroot\_ndiffs() command

#tells us. But are the data stationary now? Let's run the

#unit root tests again and see.

#

Amtrak\_kpss <- kpss.test(Amtrak\_seasdiff\_1, null = "Trend")

Amtrak\_kpss

Amtrak\_adf <- adf.test(Amtrak\_seasdiff\_1)

Amtrak\_adf

#

#We've taken 1 difference to remove seasonality. Out of curiousity

#let's take a 2nd difference to remove seasonality to see what

#difference it makes in the number of differences it tells us to

#take.

#

Amtrak\_seasdiff\_2 <- diff(Amtrak\_seasdiff\_1, lag = 1)

Amtrak.comp.seasdiff.2 <- decompose(Amtrak\_seasdiff\_2)

autoplot(Amtrak.comp.seasdiff.2)

Amtrak\_seasdiff\_ndiffs\_2 <- unitroot\_ndiffs(Amtrak\_seasdiff\_2)

Amtrak\_seasdiff\_ndiffs\_2

Amtrak\_kpss <- kpss.test(Amtrak\_seasdiff\_2, null = "Trend")

Amtrak\_kpss

Amtrak\_adf <- adf.test(Amtrak\_seasdiff\_2)

Amtrak\_adf

#

#So, it did continue to reduce the magnitude of the values somewhat

#and now it tells us we don't need to take any more differences.

#And, the unit root tests tell us the data are now stationary.

#

#

#The last step of all this is to get the ARIMA model for the Amtrak

#data. We can do this without differencing but, again for curiosity,

#let's do it for both the twice-differenced data and for the original

#data and see what we get.

#

Amtrak\_2.ts <- ts(Amtrak\_seasdiff\_2)

Amtrak\_2.tsb <- as\_tsibble(Amtrak\_seasdiff\_2)

Amtrak\_2.tsb |>

autoplot() +

labs(title = "Amtrak Data", y = "")

#

#First let's look at a time plot of the Amtrak twice-differenced data.

#That looks interesting. There is something there I didn't see in the

#data around the end of 2014/beginning of 2015 that is causing some

#wild swings in the data. I'm not sure what that is but it looks like

#we've got stationary data (no discernable trend or periodicity) now.

#

#

#There are lots of ways to get an ARIMA model. I'll use 2 of them here.

#One is from the forecast package. The other is from the fable package.

#I ran the forecast package first and got an answer. Then, I used that

#answer in the fable package. It produced some errors and wouldn't

#work. The answers the fable package did come up with are slightly

#more accurate than the one from the forecast package. Here are the

#commands.

#

library(forecast)

model1 <- auto.arima(Amtrak\_2.ts)

summary(model1)

Amtrak\_2.tsb |>

gg\_tsdisplay(difference(value), plot\_type = 'partial')

#

#I'll use the model built by the auto.arima() command, a ARIMA(5,0,4)

#model as one of the models tested by the fable package. I'll also add

#another commone model, the ARIMA(1,0,1) model to see what that does.

#And, I'll let fable automatically build 2 models with different

#parameters set.

Amtrak\_2\_fit <- Amtrak\_2.tsb |>

model(arima504 = ARIMA(value ~ pdq(5,0,4)),

arima101 = ARIMA(value ~pdq(1,0,1)),

stepwise = ARIMA(value),

search = ARIMA(value, stepwise = FALSE, approximation = FALSE))

report(Amtrak\_2\_fit)

#

#You can see the summary for the models built using this command.

#It looks like the code won't run the ARIMA(5,0,4) model. So we cannot

#directly compare the results for that model as built by 2 different

#packages/codes. But, the value of AICc for all the models built

#by both packages is pretty close. From the forecast package AICc is

#2880 and from the fable package the AICc's are in the 2760-2770 range.

#

#

#Now let's produce a forecast using these results.

#

f

#

#First the forecast results. I'll just do a forecast for 5 years.

#

f <- forecast(model1, level = c(95), h=5\*12)

plot(f)

#

#This actually looks pretty good compared with the time plot of the

#time plot of the data. But, can you tell what's wrong with

#it? Remember that I said whatever we did during processing the data

#and analysis we would have to undo to get real-world values??? This

#is where that shows up.

#

#

#Let's go back to the original time series from 1991 to 2016 and do all

#this again using the forecast package again. Remember that with an

#ARIMA model we do not have to independently take differences, i.e.

#the data do not need to be stationary for this code to work.

#

Amtrak$PassengerMiles <- Amtrak$PassengerMiles/1000000.

#Amtrak

Amtrak.ts <- ts(Amtrak$PassengerMiles, start = c(1991, 1), end = c(2017, 1), frequency = 12)

AmtrakR.ts <- ts(Amtrak$Ridership, start = c(1991, 1), end = c(2017, 1), frequency = 12)

model2 <- auto.arima(Amtrak.ts)

summary(model2)

model3 <- auto.arima(AmtrakR.ts)

summary(model3)

#

#Here I've built a model for Ridership as well as for Passenger Miles.

#This actually results in a simpler model but is much less accurate.

#The ARIMA model built using the original Amtrak data is

#ARIMA(0,1,3)(2,1,1)[12]. Let's look at the 5-year forecast now.

#

f2 <- forecast(model2, level = c(95), h=5\*12)

plot(f2)

#

#That looks like a forecast. It is pretty flat (because the prior few

#years are pretty flat). It probably isn't that different than a naive

#forecast. Let's see what we get from the fable and feasts packages.

#

#

#Just to make sure I've got the right data objects I'll go back to the

#tsibble of the original data.

#

Amtrak.tsb.91.16 <- as\_tsibble(Amtrak.ts.91.16)

Amtrak.tsb.91.16 |>

gg\_tsdisplay(difference(value), plot\_type = 'partial')

Amtrak\_fit <- Amtrak.tsb.91.16 |>

model(arima013 = ARIMA(value ~ pdq(0,1,3)),

arima101 = ARIMA(value ~pdq(1,0,1)),

stepwise = ARIMA(value),

search = ARIMA(value, stepwise = FALSE, approximation = FALSE))

report(Amtrak\_fit)

forecast(Amtrak\_fit, h=60) |>

filter(.model == 'arima013') |>

autoplot(Amtrak.tsb.91.16) +

labs(title = "Amtrak Data 1991-2016 plus 5-year Forecast")

#

#Another interesting point is that the ARIMA(0,1,3) model is actually

#slightly more accurate using the AICc than the automatically found

#ARIMA model from the fable package. The plots from the forecast

#package and the feasts package appear different but the scales are the

#same.

#

#Another point I need to make is that the further out in the future you

#go the larger the potential error gets. This is straightforward but

#might not be something you always think about but should. You can see

#the effect of this in the plots of forecasts.

#

#

#One of the things I like to do to check my work is to use 2 different

#programs that do the same thing. If you use both with the same data

#and get the same answer I think that's a pretty good conclusion!

#

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###

#

#The last part of Laboratory #4 are skills-building exercises again.

#These are intended to help you get more comfortable with the coding

#that the textbook authors use. It will help you with the final

#exam.

#

#

#Exercise 3. Part A: Turkey GDP

#

turkey <- global\_economy |> filter(Country == "Turkey")

turkey |> autoplot(GDP)

turkey |> autoplot(log(GDP))

#

#No noticeable seasonal anything.

#

turkey |> autoplot(log(GDP) |> difference())

turkey |> features(GDP, guerrero)

turkey\_gdp <- global\_economy %>%

filter(Country == "Turkey")

gg\_tsdisplay(turkey\_gdp, GDP)

#strip df down

turkey\_gdp\_prepped <- dplyr::select(turkey\_gdp, GDP)

#determine lambda

BoxCox.lambda(turkey\_gdp\_prepped$GDP)

#

#Exercise 3. Part b: Accomodation takings in Tasmania

#

# create data object

tasmania\_takings <- aus\_accommodation %>%

filter(State == 'Tasmania')

#display data

gg\_tsdisplay(tasmania\_takings, Takings)

#strip data object down

tasmania\_takings\_prepped <- dplyr::select(tasmania\_takings, Takings)

#determine lambda

BoxCox.lambda(tasmania\_takings\_prepped$Takings)

#perform boxcox and apply to df

tasmania\_takings\_prepped$Takings <- box\_cox(tasmania\_takings\_prepped$Takings, BoxCox.lambda(tasmania\_takings\_prepped$Takings))

#plot transformed data

tasmania\_takings\_prepped %>% gg\_tsdisplay(Takings, plot\_type = 'partial')

ndiffs(tasmania\_takings\_prepped$Takings)

ggtsdisplay(diff(tasmania\_takings\_prepped$Takings))

#

#Here is a second way to do all this

#

tas <- aus\_accommodation |> filter(State == "Tasmania")

tas |>

autoplot(Takings, ) +

geom\_point()

tas |> autoplot(log(Takings))

tas |> autoplot(log(Takings) |> difference(lag = 4))

tas |> autoplot(log(Takings) |> difference(lag = 4) |> difference())

tas |> features(Takings, guerrero)

#

#Exercise 3. Part c: Souvenir sales

#

#display data

gg\_tsdisplay(souvenirs, Sales)

souvenirs\_prepped <- souvenirs

#determine lambda

BoxCox.lambda(souvenirs\_prepped$Sales)

#perform boxcox and apply to df

souvenirs\_prepped$Sales <- box\_cox(souvenirs\_prepped$Sales, BoxCox.lambda(souvenirs\_prepped$Sales))

#plot transformed data

souvenirs\_prepped %>% gg\_tsdisplay(Sales, plot\_type = 'partial')

ndiffs(souvenirs\_prepped$Sales)

ggtsdisplay(diff(souvenirs\_prepped$Sales))

#

#Here is the second way to do all this.

#

souvenirs |> autoplot(Sales)

souvenirs |> autoplot(log(Sales))

souvenirs |> autoplot(log(Sales) |> difference(lag=12))

souvenirs |> autoplot(log(Sales) |> difference(lag=12) |> difference())

souvenirs |> features(Sales, guerrero)

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###

#

#Exercise 8. Part a: United States GDP

#

usa\_gdp <- global\_economy %>%

filter(Country == "United States") %>%

dplyr::select(GDP)

usa\_gdp %>%

autoplot(GDP)

lambda <- usa\_gdp |>

features(GDP, features = guerrero) |>

pull(lambda\_guerrero)

lambda

#perform boxcox and apply to data object

usa\_gdp$GDP <- box\_cox(usa\_gdp$GDP, BoxCox.lambda(usa\_gdp$GDP))

#plot transformed data

usa\_gdp %>%

gg\_tsdisplay(GDP, plot\_type = 'partial')

#

#Part b

#

fit <- usa\_gdp |>

model(ARIMA(GDP))

report(fit)

fit <- usa\_gdp |>

model(

arima010 = ARIMA((GDP) ~ 1 + pdq(0, 1, 0)),

arima011 = ARIMA((GDP) ~ 1 + pdq(0, 1, 1)),

arima012 = ARIMA((GDP) ~ 1 + pdq(0, 1, 2)),

arima013 = ARIMA((GDP) ~ 1 + pdq(0, 1, 3)),

arima110 = ARIMA((GDP) ~ 1 + pdq(1, 1, 0)),

arima111 = ARIMA((GDP) ~ 1 + pdq(1, 1, 1)),

arima112 = ARIMA((GDP) ~ 1 + pdq(1, 1, 2)),

arima113 = ARIMA((GDP) ~ 1 + pdq(1, 1, 3)),

arima210 = ARIMA((GDP) ~ 1 + pdq(2, 1, 0)),

arima211 = ARIMA((GDP) ~ 1 + pdq(2, 1, 1)),

arima212 = ARIMA((GDP) ~ 1 + pdq(2, 1, 2)),

arima213 = ARIMA((GDP) ~ 1 + pdq(2, 1, 3)),

arima310 = ARIMA((GDP) ~ 1 + pdq(3, 1, 0)),

arima311 = ARIMA((GDP) ~ 1 + pdq(3, 1, 1)),

arima312 = ARIMA((GDP) ~ 1 + pdq(3, 1, 2)),

arima313 = ARIMA((GDP) ~ 1 + pdq(3, 1, 3))

)

fit |>

glance() |>

arrange(AICc) |>

select(.model, AICc)

#

#Part d

#

best\_fit <- usa\_gdp |>

model(ARIMA(box\_cox(GDP, lambda) ~ 1 + pdq(1, 1, 0)))

best\_fit |> report()

best\_fit |> gg\_tsresiduals()

#

#Part e

#

fit %>%

forecast(h = 10) %>%

autoplot(usa\_gdp)

best\_fit %>%

forecast(h = 10) %>%

autoplot(usa\_gdp)

#

#Part f

#

ets\_check <- usa\_gdp$GDP %>%

ets()

ets\_check

usa\_gdp |>

model(ETS(GDP)) |>

forecast(h = 10) |>

autoplot(usa\_gdp)

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###

###\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*###

#

#Additional Information for ARIMA models and the Amtrak data

#

library(forecast)

Amtrak.ts.91.16 <- ts(Amtrak$PassengerMiles, start = c(1991, 1), end = c(2017, 1), frequency = 12)

Amtrak.tsb.91.16 <- as\_tsibble(Amtrak.ts.91.16)

Amtrak.ts.91.16 %>%

ggtsdisplay(Amtrak.ts.91.16, plot.type="histogram", points=FALSE,

smooth = FALSE)

fit\_Amt <- auto.arima(Amtrak.ts.91.16)

summary(fit\_Amt)

fit <- Amtrak.tsb.91.16 |>

model(arima = ARIMA())

report(fit)

fit |> gg\_tsresiduals()

#

#Plot the Amtrak data again

#

Amtrak.tsb.91.16 %>%

gg\_tsdisplay(value, plot\_type = 'partial')

AmtrakARIMA <- Amtrak.tsb.91.16 |>

gg\_tsdisplay(difference(difference(value, 12)),

plot\_type = 'partial', lag = 12) +

labs(title = "Seasonally differenced", y="")

AmtrakARIMA

fit <- Amtrak.tsb.91.16 |>

model(

AmtrakAMRIMA = ARIMA(value ~ pdq(0,1,0) + PDQ(1,0,0)),

Auto = ARIMA()

)

glance(fit)

fit |> select(Auto) |> gg\_tsresiduals(lag = 12)

fit |> select(AmtrakAMRIMA) |> gg\_tsresiduals(lag = 12)

#

#Step 6: Cross-validation

#

#

#First let's setup a training and test set and look at the accuracy

#we get using those.

#

AmtrakTraining <- Amtrak.tsb.91.16 |>

filter(year(index) < 2009)

autoplot(AmtrakTraining)

autoplot(Amtrak.tsb.91.16, value) +

autolayer(AmtrakTraining, value, colour = "red")

fit <- AmtrakTraining |>

model(SNAIVE(value))

fit |> gg\_tsresiduals()

fcast <- fit |>

forecast(new\_data = anti\_join(Amtrak.tsb.91.16, AmtrakTraining))

fcast |> autoplot(Amtrak.tsb.91.16)

bind\_rows(

accuracy(fit),

accuracy(fcast, Amtrak.tsb.91.16)

) |>

select(-.model)

#

#Here is cross-validation following https://robjhyndman.com/hyndsight/tscvexample/

#

plot(Amtrak.ts.91.16, ylab = "Passenger Miles", xlab = "Time")

k <- 60 #minimum data length for fitting a model

n <- length(Amtrak.ts.91.16)

mae1 <- mae2 <- mae3 <- matrix(NA,n-k,12)

st <- tsp(Amtrak.ts.91.16)[1]+(k-2)/12

for (i in 1:(n-k))

{

xshort <- window(Amtrak.ts.91.16, end=st + i/12)

xnext <- window(Amtrak.ts.91.16, start = st + (i+1)/12, end = st + (i+12)/12)

fit1 <- tslm(xshort ~ trend + season, lambda = 0)

fcast1 <- forecast(fit1, h=12)

fit2 <- Arima(xshort, order = c(3, 0, 1), seasonal = list(order=c(0,1,1), period=12),

include.drift = TRUE, lambda = 0, method = "ML")

fcast2 <- forecast(fit2, h=12)

fit3 <- ets(xshort, model = "MMM", damped = TRUE)

fcast3 <- forecast(fit3, h=12)

mae1[i, 1:length(xnext)] <- abs(fcast1[['mean']]-xnext)

mae2[i, 1:length(xnext)] <- abs(fcast2[['mean']]-xnext)

mae3[i, 1:length(xnext)] <- abs(fcast3[['mean']]-xnext)

}

mae1Cols <- mae1 %>%

colMeans(na.rm = TRUE)

mae1Cols

mae2Cols <- mae2 %>%

colMeans(na.rm = TRUE)

mae2Cols

mae3Cols <- mae3 %>%

colMeans(na.rm = TRUE)

mae3Cols

mae1Cols <- as.data.frame(mae1Cols)

mae1Cols

mae2Cols <- as.data.frame(mae2Cols)

mae3Cols <- as.data.frame(mae3Cols)

mae3Cols

idn <- seq(1,12,by=1)

maes <- data.frame(idn, mae1Cols, mae2Cols, mae3Cols)

maes

maes\_reshaped <- data.frame(x = idn, y = c(maes$mae1Cols, maes$mae2Cols, maes$mae3Cols),

group = c(rep("mae1", nrow(maes)),

rep("mae2", nrow(maes)),

rep("mae3", nrow(maes))))

library(data.table)

data\_melted <- melt(maes, id.vars="idn")

ggplot(data\_melted, aes(x = idn, y = value, fill = variable)) +

geom\_line(aes(group = variable, colour = variable)) +

xlab("Months 1-12") +

ylab("MAE") +

scale\_color\_manual(labels = c("LM", "ARIMA", "ETS"),

values = c("red", "blue", "green"))