Homework 1

CUNY DATA 624|2023 Summer I

Group 3: Daniel Sullivan, Jeff Parks, Lwin Shwe, Matthew Katz

Contents

# Exercise Key

**KJ** - *Applied Predictive Modeling* (Kuhn, Johnson)

**HA** - *Forecasting: Principles and Practice, 2nd Ed.* (Hyndman, Athanasopoulos)

# Required R Libraries

library(tidyverse)  
library(fpp2)  
library(corrplot)  
library(mlbench)  
library(naniar)  
library(readxl)  
library(seasonal)

# HA Exercise 2.1

**Use the help function to explore what the series gold, woolyrnq and gas represent.**

help(gold)  
help(woolyrnq)  
help(gas)

* The series gold is a time-series dataset of daily morning gold prices in US dollars, from Jan 1, 1985 through Mar 31, 1989.
* The series woolyrng is a time-series dataset of quarterly production of woolen yarn in Australia, in metric tons (tonnes), from Mar 1965 through Sep 1994.
* The series gas is a time-series dataset of monthly gas production in Australia, units unknown, from 1956 through 1995.

1. **Use autoplot() to plot each of these in separate plots.**

autoplot(gold) +   
 ggtitle("Plot for Gold Time Series") +   
 xlab("Day") +   
 ylab("Price $")  
#  
autoplot(woolyrnq) +   
 ggtitle("Plot for Wool Time Series") +   
 xlab("Year") + ylab("Weight in tonnes")  
#  
autoplot(gas) +   
 ggtitle("Plot for Gas Time Series") +   
 xlab("Year") + ylab("Amount Produced")

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |

1. **What is the frequency of each series? Hint: apply the frequency() function.**

* The frequency of the gold series is 1 (daily), the frequency of the woolyrnq series is 4 (quarterly), and the frequency of the gas series is 12 (yearly).

frequency(gold)  
frequency(woolyrnq)  
frequency(gas)

1. **Use which.max() to spot the outlier in the gold series. Which observation was it?**

* The outlier value in the gold series is observation 770 with a value of 593.7.

which.max(gold)  
gold[which.max(gold)]

# HA Exercise 2.3

**Download some monthly Australian retail data from the book website. These represent retail sales in various categories for different Australian states, and are stored in a MS-Excel file.**

retaildata <- readxl::read\_excel("data/retail.xlsx", skip=1)

1. **Select one of the time series … explore your chosen retail time series using the following functions: autoplot(), ggseasonplot(), ggsubseriesplot(), gglagplot(), ggAcf()**

ts\_ausretail <- ts(retaildata[,"A3349338X"], frequency=12, start=c(1982,4))  
#  
autoplot(ts\_ausretail) +  
 ggtitle('Monthly Product Sales: A3349338X') +  
 xlab('Year') + ylab('Sales')  
#  
ggseasonplot(ts\_ausretail) +   
 ggtitle('Seasonal Plot: A3349338X') +  
 xlab('Month') + ylab('Sales')  
#  
ggsubseriesplot(ts\_ausretail) +  
 ggtitle('Subseries Plot: A3349338X') +  
 xlab('Month') + ylab('Sales')  
#  
gglagplot(ts\_ausretail) +  
 ggtitle('Lag Plot: A3349338X')  
#  
ggAcf(ts\_ausretail) +  
 ggtitle('ACFPlot: A3349338X') +  
 xlab('Month') + ylab('Sales')  
#  
autoplot(decompose(ts\_ausretail))

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

1. **Can you spot any seasonality, cyclicity and trend? What do you learn about the series?**

* We’ve selected product A3349338X, which appears to display a gradually increasing trend over the 30 years with strong annual seasonality in the month of December, when sales spike dramatically and then quickly drop. There may also be a cyclic pattern present that follows macroeconomic trends, with broad sales slowdowns observed from 2001-2003 and again from 2011-2013.

# HA Exercise 6.2

**The plastics data set consists of the monthly sales (in thousands) of product A for a plastics manufacturer for five years.**

1. **Plot the time series of sales of product A. Can you identify seasonal fluctuations and/or a trend-cycle?**

* We can observe a rising trend-cycle with a seasonal pattern of sales increases beginning in April/May, peaking in August/September and declining for the winter months with the lowest sales month in February.

autoplot(plastics) +   
 ggtitle("Plastics: Sales per Month") +  
 xlab("Year") + ylab("Sales (K)")  
#  
ggseasonplot(plastics) +   
 ggtitle("Plastics: Classical Multiplicative Decomp") +  
 xlab("Year") + ylab("Sales (K)")

|  |  |
| --- | --- |
|  |  |

1. Use a classical multiplicative decomposition to calculate the trend-cycle and seasonal indices.

plastics\_d <- plastics %>%   
 decompose(type="multiplicative")  
#  
autoplot(plastics\_d) +   
 ggtitle("Plastics: Classical Multiplicative Decomp") +  
 xlab("Year")

|  |
| --- |
|  |

1. **Do the results support the graphical interpretation from part a?**

* The results largely support the graphical interpretation of a rising trend-cycle with a clear pattern of seasonality. However one issue we notice is a fair amount of remainder / residual that seems to also follow that seasonal pattern, suggesting that the model is not fully capturing all of the seasonal effects.

1. **Compute and plot the seasonally adjusted data.**

* See below.

1. **Change one observation to be an outlier (e.g., add 500 to one observation), and recompute the seasonally adjusted data. What is the effect of the outlier?**

* The presence of a large outlier early in the time series carried forward into future modeled seasonal periods in the decomposition, even though the outlier only occurred once.

plastics\_seas <-(plastics / plastics\_d$seasonal)  
#  
autoplot(plastics\_seas, ylim=c(800,1500)) +   
 ggtitle("Plastics: Seasonally Adjusted") +  
 xlab("Year") + ylab("Sales (K)")  
#  
plastics\_out\_10 <- plastics  
plastics\_out\_10[10] <- plastics\_out\_10[10]+500  
plastics\_out\_10\_d <- plastics\_out\_10 %>%   
 decompose(type="multiplicative")  
#  
plastics\_out\_10\_seas <- (plastics\_out\_10/plastics\_out\_10\_d$seasonal)  
#  
autoplot(plastics\_out\_10\_seas, ylim=c(800,1500)) +   
 ggtitle("Plastics + Early Outlier: Seasonally Adjusted") +  
 xlab("Year") + ylab("Sales (K)")  
#  
# decomp plots  
autoplot(plastics\_d) +   
 ggtitle("Plastics: Classical Multiplicative Decomp") +  
 xlab("Year")  
#  
autoplot(plastics\_out\_10\_d) +  
 ggtitle("Plastics + Early Outlier: Classical Multiplicative Decomp") + xlab("Year")

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

1. **Does it make any difference if the outlier is near the end rather than in the middle of the time series?**

* It seems the positioning of the outlier can have a significant impact on how the model interprets seasonality - the earlier an outlier is present in the time series, the longer it will impact the model. As the outlier moves to later positions in the time series, it has less of an effect on future periods.

# outlier at index 30  
plastics\_out\_30 <- plastics  
plastics\_out\_30[30] <- plastics\_out\_30[30]+500  
#  
plastics\_out\_30\_d <- plastics\_out\_30 %>%   
 decompose(type="multiplicative")  
#  
plastics\_out\_30\_seas <- (plastics\_out\_30/plastics\_out\_30\_d$seasonal)  
#  
autoplot(plastics\_out\_30\_seas, ylim=c(800,2000)) +   
 ggtitle("Plastics + Middle Outlier: Seasonally Adjusted") +  
 xlab("Year")   
#  
# outlier at index 50  
plastics\_out\_50 <- plastics  
#  
plastics\_out\_50[50] <- plastics\_out\_50[50]+500  
#  
plastics\_out\_50\_d <- plastics\_out\_50 %>%   
 decompose(type="multiplicative")  
#  
plastics\_out\_50\_seas <- (plastics\_out\_50/plastics\_out\_50\_d$seasonal)  
#  
autoplot(plastics\_out\_50\_seas, ylim=c(800,2000)) +   
 ggtitle("Plastics + Late Outlier: Seasonally Adjusted") +  
 xlab("Year")   
#  
# decomp plots  
autoplot(plastics\_out\_30\_d) +   
 ggtitle("Plastics + Middle Outlier: Classical Multiplicative Decomp") +  
 xlab("Year")  
autoplot(plastics\_out\_50\_d) +   
 ggtitle("Plastics + Late Outlier: Classical Multiplicative Decomp") +  
 xlab("Year")

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

# KJ Exercise 3.1

**The UC Irvine Machine Learning Repository contains a data set related to glass identification. The data consist of 214 glass samples labeled as one of seven class categories. There are nine predictors, including the refractive index and percentages of eight elements: Na, Mg, Al, Si, K, Ca, Ba, and Fe.**

1. **Using visualizations, explore the predictor variables to understand their distributions as well as the relationships between predictors.**
2. **Do there appear to be any outliers in the data? Are any predictors skewed?**
3. **Are there any relevant transformations of one or more predictors that might improve the classification model?**

* We can observe that:
  + Al is slightly right skewed
  + Ba is right skewed and mostly centered around 0
  + Ca is right skewed
  + Fe is right skewed and mostly centered around 0
  + K is right skewed \* Mg is left skewed and bimodal
  + Na is almost normal with a slight right tail
  + RI is right skewed \* Si is left skewed
  + Type is mostly centered around Types 1,2, and 7
* There also seems to be a strong positive correlation between RI and Ca. There are also notable negative correlations between RI and Si, Al and Mg, Ca and Mg, and Ba and Mg. There is also notable positive correlations between Ba and Al.
* There seem to be outliers present in Ba, K, RI, Ca, Fe, and possibly Na.
* Since Be, Fe, and K have a strong right skewness with a concentrations of points with low values, they may benefit from log transformations. Mg may also be log-transformed since it is left skewed. RI can be inverse squared while Si can be squared, and Al can be square rooted. It would also be interesting to see how the model performs without Ca as it has correlations with other variables.

library(mlbench)  
library(corrplot)  
data(Glass)  
  
Glass %>%  
 keep(is.numeric) %>%  
 gather() %>%  
 ggplot(aes(value)) +   
 geom\_histogram(bins = 15) +   
 facet\_wrap(~key, scales = 'free') +  
 ggtitle("Histograms of Numerical Predictors")  
#  
Glass %>%  
 keep(is.numeric) %>%  
 gather() %>%  
 ggplot(aes(value)) +   
 geom\_boxplot() +   
 facet\_wrap(~key, scales = 'free') +  
 ggtitle("Boxplots of Numerical Predictors")  
#  
Glass %>%  
 keep(is.numeric) %>%  
 cor() %>%  
 corrplot()   
#  
Glass %>%  
 ggplot() +  
 geom\_bar(aes(x = Type)) +  
 ggtitle("Distribution of Types of Glass")

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

# KJ Exercise 3.2

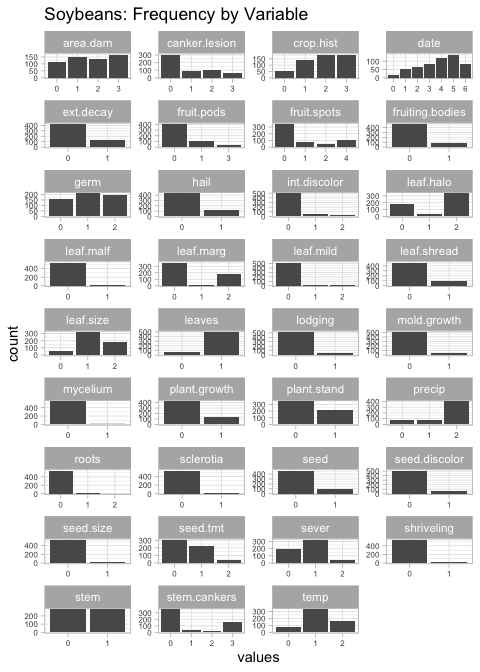
**The UC Irvine Machine Learning Repository contains a data set related to soybeans. Data were collected to predict disease in 683 soybeans. The 35 predictors are mostly categorical and include information on the environmental conditions (e.g., temperature, precipitation) and plant conditions (e.g., left spots, mold growth). The outcome labels consist of 19 distinct classes.**

1. **Investigate the frequency distributions for the categorical predictors. Are any of the distributions degenerate in the ways discussed earlier in this chapter?**
2. **Roughly 18 % of the data are missing. Are there particular predictors that are more likely to be missing? Is the pattern of missing data related to the classes?**
3. **Develop a strategy for handling missing data, either by eliminating predictors or imputation.**

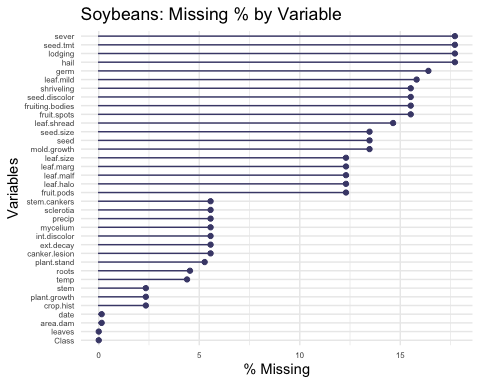
* By examining bar plots of the frequencies, we can see that all variables have more than one possible outcome, so they are not ‘degenerate distributions’ in the strictest sense. However, several only have two possible outcomes with a huge imbalance (such as shriveling and leaf.maif), so these might be considered to fall into this category.
* Using the naniar package to visualize missing data, we can see that certain predictors (e.g. sever, seed.tmt, lodging and hail) are missing data for over 15% of the observations. However can also see this missingness is restricted to just five soybean Classes.
* On closer examination, we can see that several classes (2-4-d-injury, cyst-nematode, herbicide-injury) are missing 100% of data for most of the predictor variables. Altogether these only represent 38 observations or about 6% of the total. Unless there is a compelling reason to keep these three classes, we might consider dropping them from the model, and using a robust imputation method (such as MICE or KNN) on the two other classes for which we have partial data.

library(naniar)  
library(mlbench)  
data(Soybean)  
df\_soybean <- Soybean  
  
# factors: ordered to regular; drop NAs and 'Class'  
df\_soybean\_freq <- df\_soybean %>%  
 dplyr::select(!Class) %>%  
 drop\_na() %>%  
 mutate\_if(is.ordered, ~ factor(., ordered = FALSE))

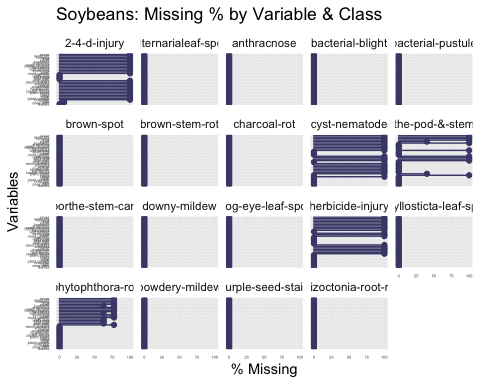
# matrixed bar plots  
df\_soybean\_freq %>%  
 pivot\_longer(everything(), names\_to = c('variables'), values\_to = c('values')) %>%  
 ggplot() + geom\_bar(aes(x=values)) +  
 facet\_wrap(vars(variables), scales="free", ncol=4) +  
 ggtitle('Soybeans: Frequency by Variable') +  
 theme(axis.text = element\_text(size = 6))



# missing data by predictor  
gg\_miss\_var(df\_soybean, show\_pct=TRUE) +  
 ggtitle('Soybeans: Missing % by Variable') +  
 theme(axis.text = element\_text(size = 6))



# missing data by predictor and class  
gg\_miss\_var(df\_soybean, facet = Class, show\_pct=TRUE) +  
 ggtitle('Soybeans: Missing % by Variable & Class') +  
 theme(axis.text = element\_text(size = 3))

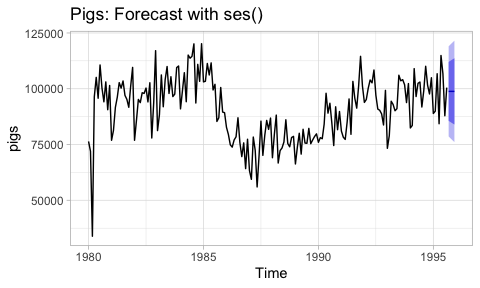


# HA Exercise 7.1

**Consider the pigs series — the number of pigs slaughtered in Victoria each month.**

1. **Use the ses() function in R to find the optimal values of** **and** **, and generate forecasts for the next four months.**

data(pigs)  
pigs\_ses <- ses(pigs, h=4)  
#  
autoplot(forecast(pigs\_ses)) +  
 ggtitle('Pigs: Forecast with ses()')



round(pigs\_ses$model$par['alpha'],3) # 0.297  
round(pigs\_ses$model$par['l'],2) # 77260.06

The optimal parameter values determined by the ets model are 0.30 for , and 77260.06 for .

1. **Compute a 95% prediction interval for the first forecast using** **where s is the standard deviation of the residuals. Compare your interval with the interval produced by R.**

pigs\_ses$mean[1] # point forecast 98816.41  
  
# 95% PI from model  
pigs\_ses$lower[,'95%'][1] # 78611.97 lower  
pigs\_ses$upper[,'95%'][1] # 119020.8 upper  
  
# 95% PI estimates .. more accurate PIs provided in the model  
pigs\_ses\_sd <- sd(pigs\_ses$residuals)  
pigs\_ses$mean[1] - pigs\_ses\_sd # 88542.71 lower  
pigs\_ses$mean[1] + pigs\_ses\_sd # 109090.1 upper

Our point forecast for the first period is 98816.41 with an estimated standard deviation of 10273.69, producing a 95% confidence interval from 88542.71 to 109090.10.

The model calculates a more accurate estimate from 78611.97 lower bound to 78611.97 upper bound.

# HA Exercise 7.2

**Write your own function to implement simple exponential smoothing. The function should take arguments y (the time series), alpha (the smoothing parameter** **) and level(the initial level** **). It should return the forecast of the next observation in the series. Does it give the same forecast as ses()?**

my\_ses <- function(y, alpha, l0){  
 y\_hat <- l0  
 for(index in 1:length(y)){  
 y\_hat <- alpha\*y[index] + (1 - alpha)\*y\_hat   
 }  
 return(y\_hat)  
}  
  
ses\_pigs <- ses(pigs, h = 4)  
  
my\_ses\_pigs <- my\_ses(pigs,   
 alpha = ses\_pigs$model$par[1],   
 l0 = ses\_pigs$model$par[2])

ses\_pigs$mean[1] # 98816.41  
my\_ses\_pigs # 98816.41

Using the pigs time series and identical parameters, we get the same one-step forecast from our custom function as we do from the ses() function.

* ses() forecast: 98816.41
* custom forecast: 98816.41

# HA Exercise 7.3

**Modify your function from the previous exercise to return the sum of squared errors rather than the forecast of the next observation. Then use the optim() function to find the optimal values of** **and** **. Do you get the same values as the ses() function?**

my\_sse <- function(pars = c(alpha, l0), y){  
 error <- 0  
 SSE <- 0  
 alpha <- pars[1]  
 l0 <- pars[2]  
 y\_hat <- l0  
   
 for(index in 1:length(y)){  
 error <- y[index] - y\_hat  
 SSE <- SSE + error^2  
   
 y\_hat <- alpha\*y[index] + (1 - alpha)\*y\_hat   
 }  
 return(SSE)  
}  
  
my\_sse\_pigs <- optim(par = c(0.5, pigs[1]), y = pigs, fn = my\_sse)

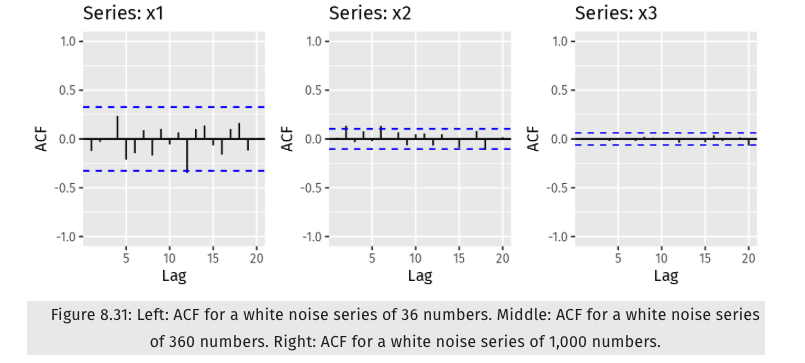
ses\_pigs$model$par[1] # 0.2971488   
my\_sse\_pigs$par[1] # 0.2990081  
  
ses\_pigs$model$par[2] # 77260.06  
my\_sse\_pigs$par[2] # 76379.27

The resulting parameters are very close to the ses() model, but not exact.

* ses() parameters: :0.30, :76379.27
* custom parameters: :0.30, :77260.06

# HA Exercise 8.1

**Figure** [**8.31**](https://otexts.com/fpp2/arima-exercises.html#fig:wnacfplus) **shows the ACFs for 36 random numbers, 360 random numbers and 1,000 random numbers.**



1. **Explain the differences among these figures. Do they all indicate that the data are white noise?**
2. **Why are the critical values at different distances from the mean of zero? Why are the autocorrelations different in each figure when they each refer to white noise?**

* Since all three datasets are generated from random (white noise) distributions, we would expect there to be no autocorrelation detected, and thus an ACF value of zero throughout.
* In actual practice, random variation does occur in these randomly-generated samples and so the datasets with a low display some ‘false positives’ of autocorrelation detected at random intervals.
* However as number of samples grows larger, these random errors tend to cancel each other out and our ACF values begin to converge to zero.

# HA Exercise 8.2

**A classic example of a non-stationary series is the daily closing IBM stock price series (data set ibmclose).**

1. **Use R to plot the daily closing prices for IBM stock and the ACF and PACF. Explain how each plot shows that the series is non-stationary and should be differenced.**

* While there is not a great deal of seasonality apparent in this dataset, the time series plot of daily closing prices demonstrates a strong cyclic trend and changes in levels. In order to make this non-stationary dataset stationary, we might consider plotting the daily differences in prices instead.
* We’ve converted this time series to a more stationary dataset using differencing, which can also be demonstrated with the ACF and PACF graphs - the results look like a white noise series with only two autocorrelations exceeding the 95% limits.

data(ibmclose)  
  
autoplot(ibmclose, series='orig') +   
 autolayer(diff(ibmclose), series='diff') +  
 ggtitle('IBM Daily Closing Prices: Original and Differenced') +  
 xlab('Days') + ylab('Closing Price')



ggAcf(ibmclose) +  
 ggtitle('IBM Prices: ACF')  
#  
ggAcf(diff(ibmclose)) +  
 ggtitle('IBM Prices (Differenced): ACF')  
#  
ggPacf(ibmclose) +  
 ggtitle('IBM Prices: PACF')  
#  
ggPacf(diff(ibmclose)) +  
 ggtitle('IBM Prices (Differenced): PACF')

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

# HA Exercise 8.6

**Use R to simulate and plot some data from simple ARIMA models.**

1. **Use the following R code to generate data from an AR(1) model with** **and** **. The process starts with** **.**

y <- ts(numeric(100))  
e <- rnorm(100)  
  
for(i in 2:100)  
 y[i] <- 0.6\*y[i-1] + e[i]

1. **Produce a time plot for the series. How does the plot change as you change** **?**

* As the value of phi increases, the amount of variance in the time series increases - in this case, much more dramatic cycles are observed instead of the values staying fairly close to the mean.

ar1 <- function(y,e,phi){  
 for(i in 2:100){  
 y[i] <- phi \* y[i-1] + e[i]  
 }  
 autoplot(y, ylim=c(-5,5))  
}  
  
y <- ts(numeric(100))  
e <- rnorm(100)  
  
ar1(y,e,0.2)  
ar1(y,e,0.5)  
ar1(y,e,0.9)

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **Write your own code to generate data from an MA(1) model with** **and** **.**

y <- ts(numeric(100))  
e <- rnorm(100)  
  
for(i in 2:100)  
 y[i] <- y[i-1] + 0.6\*e[i]

1. **Produce a time plot for the series. How does the plot change as you change** **?**

* As the value of theta increases, the amount of variance in the time series increases - and in this case, the trend becomes more pronounced.

ma1 <- function(y,e,phi){  
 for(i in 2:100){  
 y[i] <- y[i-1] + phi \* e[i]  
 }  
 autoplot(y, ylim=c(-20,20))  
}  
  
y <- ts(numeric(100))  
e <- rnorm(100)  
  
ma1(y,e,0.2)  
ma1(y,e,0.5)  
ma1(y,e,0.9)

|  |  |  |
| --- | --- | --- |
|  |  |  |

1. **Generate data from an ARMA(1,1) model with** **,** **and** **.**

arma <- ts(numeric(100))  
e <- rnorm(100)  
  
for(i in 2:100){  
 arma[i] <- 0.6\*arma[i-1] + 0.6\*e[i]  
}

1. **Generate data from an AR(2) model with** **,** **and** **. (Note that these parameters will give a non-stationary series.)**

ar2 <- ts(numeric(100))  
e <- rnorm(100)  
  
for(i in 3:100){  
 ar2[i] <- 0.3\*ar2[i-2] + 0.8\*ar2[i-1] + e[i]  
}

1. **Graph the latter two series and compare them.**

* The ARMA(1,1) plot is relatively stationary with no obvious or predicatble trends, while the AR(2) model is non-stationary with what appears to be an exponential curve which can either increase or decrease infinitely, depending upon the opening conditions of the random sample.

autoplot(arma) +  
 ggtitle('ARMA(1,1)')  
#  
autoplot(ar2) +  
 ggtitle('AR2')

|  |  |
| --- | --- |
|  |  |

# HA Exercise 8.8

**Consider austa, the total international visitors to Australia (in millions) for the period 1980-2015.**

1. **Use auto.arima() to find an appropriate ARIMA model. What model was selected. Check that the residuals look like white noise. Plot forecasts for the next 10 periods.**

* An ARIMA(0,1,1) with Drift model was selected, and the p-value of a Ljung-Box test of the residuals was quite large (0.97) and therefore similar to the distribution of a white noise series.

austa\_fit <- auto.arima(austa)  
# ARIMA(0,1,1) with drift   
  
autoplot(forecast(austa\_fit, h=10), ylim=c(0,10))  
#  
ggAcf(austa\_fit$residuals)  
#  
Box.test(austa\_fit$residuals,   
 lag=10, type="Ljung-Box")   
# small X2 and large p-value: no correlations in the residuals

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
| Box-Ljung test  data: austa\_fit$residuals X-squared = 3.2552, df = 10, p-value = 0.9748 |

1. **Plot forecasts from an ARIMA(0,1,1) model with no drift and compare these to part a. Remove the MA term and plot again.**

* The upward trend of the forecasts went away with this new model, which appears to be similar to a naive forecast model based on the last-observed value. Removing the MA term resulted in slightly narrower bands for the Predictive Intervals.

austa\_011\_fit <- Arima(austa, order=c(0,1,1), include.drift=FALSE)  
autoplot(forecast(austa\_011\_fit), ylim=c(0,10))  
#  
austa\_010\_fit <- Arima(austa, order=c(0,1,0), include.drift=FALSE)  
autoplot(forecast(austa\_010\_fit), ylim=c(0,10))

|  |  |
| --- | --- |
|  |  |

1. **Plot forecasts from an ARIMA(2,1,3) model with drift. Remove the constant and see what happens.**

* The upward trend of the forecasts returned with this new model, but with an apparent ‘damping’ effect resulting in a gradual tapering-off with later forecast values. Removing the constant from this model resulted in an error pointing to a non-stationary condition in the dataset (commented out in the code block below!)

austa\_213\_fit <- Arima(austa, order=c(2,1,3), include.drift=TRUE)  
autoplot(forecast(austa\_213\_fit), ylim=c(0,10))  
#  
# austa\_213\_noc\_fit <- Arima(austa, order=c(2,1,3), include.drift=TRUE,   
# include.constant=FALSE)  
# autoplot(forecast(austa\_213\_noc\_fit), ylim=c(0,10))  
# error: non-stationary AR part from CSS

|  |
| --- |
|  |

1. **Plot forecasts from an ARIMA(0,0,1) model with a constant. Remove the MA term and plot again.**

* These models seemed to approximate a simple average forecast, with MA helping transition from the last-observed value to the mean of the time series. Removing the MA term wiped out this smoothing effect.

austa\_001\_fit <- Arima(austa, order=c(0,0,1), include.constant=TRUE)  
autoplot(forecast(austa\_001\_fit), ylim=c(0,10))  
#  
austa\_000\_fit <- Arima(austa, order=c(0,0,0), include.constant=TRUE)  
autoplot(forecast(austa\_000\_fit), ylim=c(0,10))

|  |  |
| --- | --- |
|  |  |

1. **Plot forecasts from an ARIMA(0,2,1) model with no constant.**

austa\_021\_noc\_fit <- Arima(austa, order=c(0,2,1),   
 include.constant=FALSE)  
#  
autoplot(forecast(austa\_021\_noc\_fit), ylim=c(0,10))

|  |
| --- |
|  |