Problem Set 1

Prof. Conlon

Due: 2/15/19

Introduction

This document was produced by R using RMarkdown. To complete this weeks assignment, we will ask you to complete a series of analytical and coding exercises. The **Analytical Exercises** require no coding, whereas the **Coding Exercises** require you to use R. The nice thing about RMarkdown is you can do both your analytical and coding exercises in the same document. For each part in the **Coding Exercises**, we provide an empty space of code chunks (area highlighted in grey with a header of the form "## ~~ Problem XX ~~ ##"")

To ease your introduction into R, **Problem 2** is a short tutorial into the R programming environment. Hopefully, you have already downloaded RStudio on your computer. If not, please go do that now. You can download the latest version at (https://www.rstudio.com/products/rstudio/download/).

Once you have downloaded RStudio, you will be able to open the R markdown script (assignment_1.rmd file) that created this assignment_1.pdf file. We ask you to fill in your code in the code chunk sections (the areas highlighted in gray bounded by "' marks) in the .rmd file in each of the subparts of the questions. You will see that you can include LaTex code in these document. You are not required to use LaTex to do the analytical exercises (i.e. those without coding), but it is good practice to improve your LaTex skills.

In order to compile a markdown document (i.e. turn your code into a pdf file), you must have a version of LaTex downloaded on your computer. I suggest you download MikTex (https://miktex.org/howto/install-miktex).

If at any time you are confused about R, or not sure what a command does or additional arguments available for each command, there are two tried and true methods to help resolve this issue. In the R console, you can use the `help` command, where you supply the name of the command you are confused about. Alternatively, google is your friend.

Installing MikTex

R Markdown was recently updated, and this update has issues with missing packages with MikTex. In order to deal with these issues, you need to allow MikTex to install any missing packages without asking you first. To do so, when you are installing MikTex, in the 'Settings' screen, it asks **Install missing packages on-the-fly**. Please select **Yes** in this screen. If you have already installed MikTex, you can go to the MikTex console -> `Settings` and the same box appears in that screen.

Packages to Install

Each week, we will list the packages that you need to install into R in order for you to complete the assignments. This also allows you to know a nice resource to view which packages you have learned throughout this course.

The packages used this week are

- stats
- ggplot2 (optional)

Code Setup

```
## This is a code chunk: it is outlined in grey and has R code inside of it
## Note: you can change what is shown in the final .pdf document using arguments
##       inside the curly braces at the top {r, comment='\t\t'}. For example, you
```

```r
##       can turn off print statements showing in the .pdf by adding 'echo=False'
##       i.e. changing the header to {r, comment='\t\t',echo=FALSE}

## ~~~~~~~~~~~~~~ CODE SETUP ~~~~~~~~~~~~~~ ##

# ~ This bit of code will be hidden after Problem Set 1 ~
#
# This section sets up the correct directory structure so that
#  the working directory for your code is always in the data folder

# Retrieve the code working directory
#script_dir = dirname(sys.frame(1)$ofile)
initial_options <- commandArgs(trailingOnly = FALSE)
render_command <- initial_options[grep('render',initial_options)]
script_name <- gsub("'", "",
                    regmatches(render_command,
                               gregexpr("'([^']*)'",
                               render_command))[[1]][1])

# Determine OS (backslash versus forward slash directory system)
sep_vals = c(length(grep('\\\\',script_name))>0,length(grep('/',script_name))>0)
file_sep = c("\\\\","/")[sep_vals]

# Get data directory
split_str = strsplit(script_name,file_sep)[[1]]
len_split = length(split_str) - 2
data_dir = paste(c(split_str[1:len_split],'data',''),collapse=file_sep)

# Check that the data directory contains the files for this weeks assignment
data_files = list.files(data_dir)
if(any(sort(data_files)!=sort(c('us_air_rev.csv','us_load_factor.csv')))){
  cat("ERROR: DATA DIRECTORY NOT CORRECT\n")
  cat(paste("data_dir variable set to: ",data_dir,collapse=""))
}
```

```
        ERROR: DATA DIRECTORY NOT CORRECT
        data_dir variable set to:  C:/Users/ryanl/Dropbox/r_assignments/assignments/assignment_1/data/
```

Problem 1 (Analytical Exercise)

Consider a simple $AR(1)$ model:

$$Y_t = \alpha Y_{t-1} + \varepsilon_t \qquad \text{with } \varepsilon_t \sim N(0, \sigma^2) \text{ for } t = \{1, \ldots, T\} \text{ and } Y_0 = 0$$

1. What is the distribution of Y_1? What is the distribution of Y_2?

$$Y_1 = \varepsilon_1 \sim N(0, \sigma^2)$$
$$Y_2 = \alpha Y_1 + \varepsilon_2 = (1+\alpha)\varepsilon_1 \sim N(0, (1+\alpha)^2\sigma^2)$$

2. What is the distribution of Y_t for $|\alpha| < 1$ as $t \to \infty$.

$$\begin{aligned} Y_\infty &= \sum_{j=0}^{\infty} \alpha^j \varepsilon_1 \\ &= \frac{1}{1-\alpha}\varepsilon_1 \\ &\sim N(0, \Big(\frac{1}{1-\alpha}\Big)^2 \sigma^2) \end{aligned}$$

3. What is the definition of stationarity? Explain why in this model we can check for stationarity by looking at the mean and the variance of the Y_t.

   Stationarity requires all moments of a distribution to be independent of t and finite! The normal distribution is uniquely defined by two parameters, the mean and the variance, meaning these are the only two moments that we need to check.

4. Suppose that $\alpha = 1$. Why does this imply that the model is nonstationary? Can you think of a simple transformation that makes the model stationary?

   If $\alpha = 1$ then we see from part (2) that the variance $\Big(\frac{1}{1-\alpha}\Big)^2 \sigma^2 \to \infty$ as $t \to \infty$. One simple transformation would be to take first differences, then the model is simply white noise.

5. Now suppose that $|\alpha| < 1$. Find a formula for the jth autocorrelation $\rho_j = corr(Y_t, Y_{t-j})$.

   The autocorrelation function ρ_j is the ratio of autocovariances evaluated a lag j over autocovariances evaluated at lag 0:

$$autocov_j = \mathbb{E}\big[(Y_t - \mathbb{E}Y_t)(Y_{t-j} - \mathbb{E}Y_{t-j})\big]$$

   Using back substitution, we see that:

$$Y_t = \alpha^j Y_{t-j} + \sum_{k=0}^{j-1} \alpha^k \varepsilon_{t-k}$$

   Plugging this in above, we get that:

$$\begin{aligned} \mathbb{E}\big[(Y_t - \mathbb{E}Y_t)(Y_{t-j} - \mathbb{E}Y_{t-j})\big] &= \mathbb{E}\big[(Y_{t-j})^2 \alpha^j\big] \\ &= \alpha^j \Big(\sum_{k=0}^{t-j} \alpha^{2k}\sigma^2\Big) \end{aligned}$$

   Note for this to be defined we can let $autocov_0 = 1$.

6. Explain how we could use estimates of ρ_j for $j = 1, 2, \ldots$ to check whether some actual time series data was generated by an AR(1) model like we one described above.

Problem 2 (Coding Exercise)

The problem will take you through a few tasks to familiarize yourself with R, as well as, some basic time series concepts:

(a) Loading data into R

(b) Doing simple data analysis
(c) Doimg time series analysis

For this problem, we have pulled two seperate datasets from the FRED database, maintained by the Federal Reserve Bank of Saint Louis (https://fred.stlouisfed.org/). The datasets cover the aggregate revenue and load factor in domestic US flights from 2000 to 2018. In the last two decades, airlines have begun using sophisticated algorithms to increase capacity utilization of flights (i.e. flights tend to be more full). Furthermore, during the same time period, airline revenues have increased. The point of this exercise will be to understand the role of these productivity increases in "explaining" increased revenues in the airline industry.

The two seperate datasets you will be working with are:

1. US Domestic Air Travel Revenue Passenger Mile (**filename = us_air_rev.csv**) : this dataset contains monthly data detailing the number of miles traveled by paying passengers in domestic US air travel.
2. US Domestic Air Travel Load Factor (**filename = us_load_factor.csv**) : this dataset contains monthly data detailing the percentage of seats filled up (capacity utilitization) in domestic US air travel.

A Small Detour: Brief introduction to print statements

We ask you to print a number of your results in this exercise. In R, there are two different wants to print results:

1. `print`
2. `cat`

There are some deep programmatic differences underlying what each of these does, for our purposes we only care about how easy to read your outputs are.

Printing Strings

Let's say you have a list of numbers, [4,5,6] and I want you to print out the following statement:

The first element of the list is: 4
The second element of the list is: 5
The third element of the list is: 6

Below I show you three ways to do so, the first way simply uses print without any additional arguments. The second way uses print with an additional argument, `quote=False` which gets rid of the quotes around the strings. The third approaching, using `cat`, shows how this combines the second approach and has an additional formatting feature that is useful for printing output.

```r
## Define a list called x with 3 elements
x = c(4,5,6)

## Retrieve 1st, 2nd, 3rd element of list
first_elem  = x[1]  #1st element
second_elem = x[2]  #2nd element
third_elem  = x[3]  #3rd element

## Create strings stating 'The XXXX element of the list is:'
first_str = 'The first element of the list is:'
second_str = 'The second element of the list is:'
third_str = 'The third element of the list is:'

## Concatenate the list elements and the string to create the whole phrase
first_phrase  = paste(first_str,first_elem,sep=' ')
second_phrase = paste(second_str,second_elem,sep=' ')
third_phrase  = paste(third_str,third_elem,sep=' ')
```

```r
## ~~ (1) Print without extra arguments ~~ ##
print('~~ (1) Print without extra arguments ~~')
print(first_phrase)
print(second_phrase)
print(third_phrase)

## ~~ (2) Print with extra argument turning off quotes ~~ ##
print('~~ (2) Print with extra argument turning off quotes ~~',quote=F)
print(first_phrase, quote=F)
print(second_phrase, quote=F)
print(third_phrase, quote=F)

## ~~ (3) Print without quotes and without trailing # ~~ ##
cat("\n")
cat("~~ (3) Print without quotes and without trailing # ~~\n")
cat(paste(first_phrase,"\n",sep=''))
cat(paste(second_phrase,"\n",sep=''))
cat(paste(third_phrase,"\n",sep=''))
```

```
[1] "~~ (1) Print without extra arguments ~~"
[1] "The first element of the list is: 4"
[1] "The second element of the list is: 5"
[1] "The third element of the list is: 6"
[1] ~~ (2) Print with extra argument turning off quotes ~~
[1] The first element of the list is: 4
[1] The second element of the list is: 5
[1] The third element of the list is: 6

~~ (3) Print without quotes and without trailing # ~~
The first element of the list is: 4
The second element of the list is: 5
The third element of the list is: 6
```

Printing Dataframes

The main object you will be working with in R is called a dataframe (think an excel spreadsheet). We will discuss more fully these objects in the following section. However, oftentimes you will be asked to print out dataframes. In this case, using `print` is your best option. This is due to differences between `cat` and `print` that are beyond the scope of this introduction.

(a) Loading Data

The first thing we want you to do is to load both datasets: **us_air_rev.csv** and **us_load_factor.csv** into R.

Please load data in the section below

```r
## ~~ Problem 2: Part (a) Load Data into R ~~ ##

## Change working directory to data directory
setwd(data_dir) # <- This is set in CODE SETUP section
                #    If you are having issues, you can manually
                #     set this variable to the folder where the data is

## Load Air Revenue Data ##
```

```r
us_air_rev = read.csv('us_air_rev.csv')

## Load Air Load Factor Data ##
us_load_factor = read.csv('us_load_factor.csv')
```

There are two ways to view data that you have loaded into memory in R.

1. View only first (or last few rows) using `head` (`tails`) commands
2. View the entire dataset in a seperate window using `View` commands

Note, for very large datasets it is not a good idea to use the `View` command as it is very memory (RAM) intensive.

Other checks you always want to do when loading data includes:

1. Check the column names using `colnames`
2. Check the data types for each column using a loop and `xxx`
3. Check the dimension (number of rows and columns) using the `dim` command

We now want you to run the following checks on both of your loaded datasets:

(1) Print the column names.
(2) Print off the first 20 rows.
(3) Print off the number of rows and columns.
(4) Print the data types of all the columns.

Note, for part (4) I have already built the for loop statement to get all the data types for each of the columns. For those familiar with `for loops` in other environments, R has a built in set of `apply` functions that are optimized for specific objects (`lapply` is optimized for lists, `vapply` is optimized for vectors etc). If you are unfamiliar with `for loops`, give it a google.

```r
## ~~ Problem 2: Part (a) Run Data Checks ~~ ##

# (1) Print the column names #
print(colnames(us_air_rev))

# (2) Print off first 20 rows $
print(us_air_rev[1:20,])

# (3) Print number of rows and columns #
cat(paste('Rows: ',dim(us_air_rev)[1],' Columns: ', dim(us_air_rev)[2],"\n",sep=''))

# (4) Print the data types of columns #
print(sapply(us_air_rev, class))
```

```
[1] "date"      "total_rev"
        date total_rev
1   1/1/2000  34913749
2   2/1/2000  36005847
3   3/1/2000  43901516
4   4/1/2000  41954244
5   5/1/2000  43168824
6   6/1/2000  45856363
7   7/1/2000  47303803
8   8/1/2000  46457758
9   9/1/2000  38372644
10 10/1/2000  41820446
11 11/1/2000  40712967
```

```
12 12/1/2000  39656140
13  1/1/2001  37179562
14  2/1/2001  35784235
15  3/1/2001  44208143
16  4/1/2001  42363481
17  5/1/2001  42517548
18  6/1/2001  45304289
19  7/1/2001  48067498
20  8/1/2001  48579798
Rows: 223 Columns: 2
     date total_rev
 "factor" "integer"
```

(b) Doing simple data analysis

In the next part, we will have you doing some actual time series analysis. But generally we are interested in decomposing time series into trend, seasonal and stochastic components. One clear form of seasonality is month to month variation in the data. An "approximation" for trend components is to look at year to year changes. We will have you investigate these below.

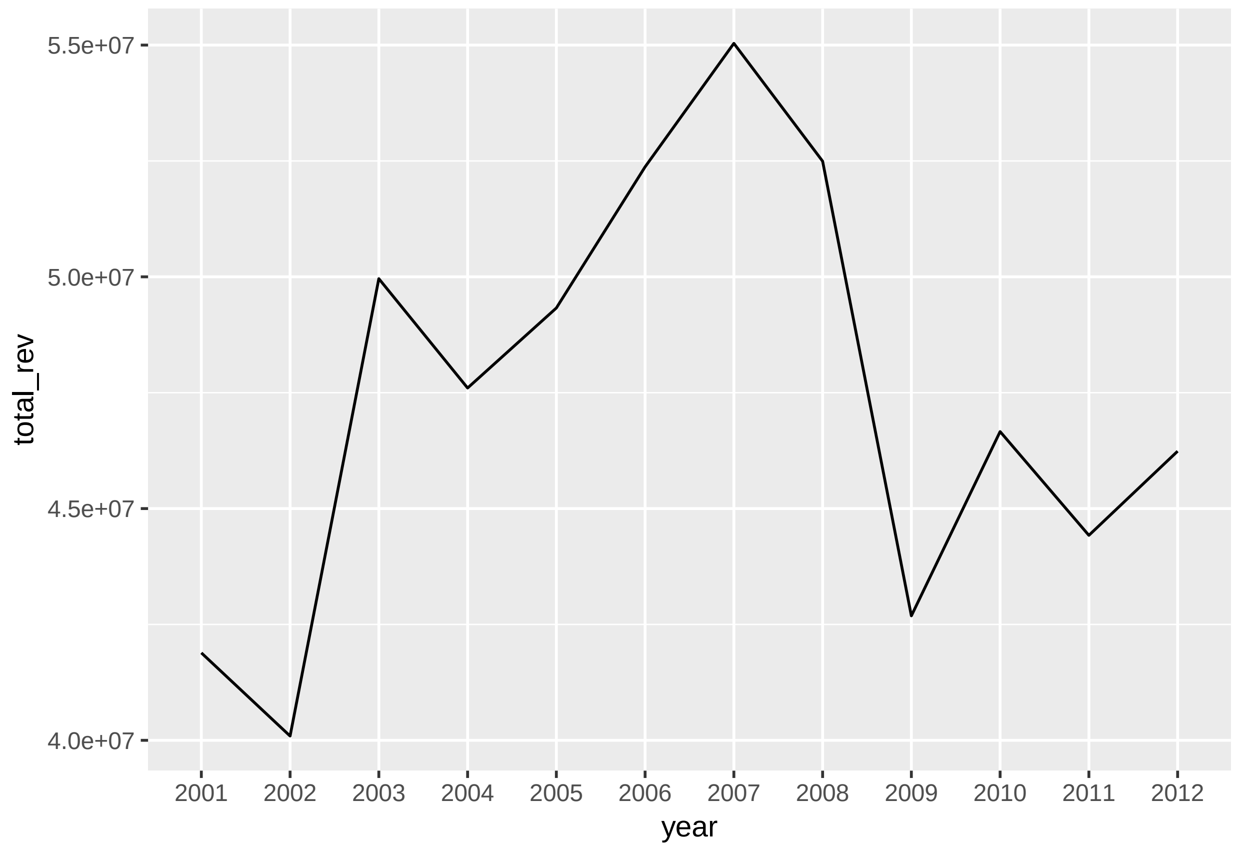
We now want you to do the following:

(1) Calculate the average revenue and load factor, by year. Do this two ways: (1) Using `aggregate` and `mean`, (2) Using `aggregate` and `sum`.
(2) Calculate the average revenue and load factor, by month. Do this two ways: (1) Using `aggregate` and `mean`, (2) Using `aggregate` and `sum`.
(3) Plot graphs for part (1) and (2) on the same plot, using your favorite plotting function. Note, you can either use the built-in `plot` function or the popular external library `ggplot2`.

For parts (1) and (2), I want you to build a better understanding of using R. I am asking you to compute averages using two different methods. In Method (1), you can use the built-in `mean` function to have R do the work for you. In Method (2), you will do the average calculation yourself by summing over observations and dividing by the number of observations.

```r
## ~~ Problem 2: Part (b) Simple Data Analysis ~~ ##

# (0) Turn date column into date format #
us_air_rev$date = as.Date(us_air_rev$date,format='%D')
us_air_rev$month = format(us_air_rev$date,'%m')
us_air_rev$year = format(us_air_rev$date,'%Y')

# (1a) Calculate mean by year using aggregate + mean #
mean_1a = aggregate(.~year,
                    us_air_rev[,colnames(us_air_rev) %in% c('year','total_rev')],
                    FUN=mean,
                    na.rm=T)

# (1b) Calculate mean by year using aggregate + sum #
mean_by_sum = function(x){
  return(sum(x)/length(x))
}

mean_1b = aggregate(.~year,
                    us_air_rev[,colnames(us_air_rev) %in% c('year','total_rev')],
                    FUN=mean_by_sum)
```

```r
# (2a) Calculate mean by month using aggregate + mean #
mean_2a = aggregate(.~month,
                    us_air_rev[,colnames(us_air_rev) %in% c('month','total_rev')],
                    FUN=mean,
                    na.rm=T)

# (2b) Calculate mean by year using aggregate + sum #
mean_by_sum = function(x){
  return(sum(x)/length(x))
}

mean_2b = aggregate(.~month,
                    us_air_rev[,colnames(us_air_rev) %in% c('month','total_rev')],
                    FUN=mean_by_sum)

# (3) Plot year #
ggplot(mean_1a,aes(x=year,y=total_rev,group=1)) + geom_line()
```



(c) Doing time series analysis

In R, there are already built-in functions that allow us to do these seasonality and trend decompositions with much fewer lines of code. To do so, we must convert our data into time series objects. What seperates a normal vector of data from a vector of time series data is that the latter has some time frequency of observations. In our case, the time frequency is monthly.

We want now return to the main question of this section, how much does capacity utilizations explain increases in airline revenue?

Fixing notation, we have:

- $t \in \{01/2000, \ldots, 12/2018\} = T$ is month-year combinations
- Rev_t is revenue for each month-year combination
- $Load_t$ is load factor for each month-year combination

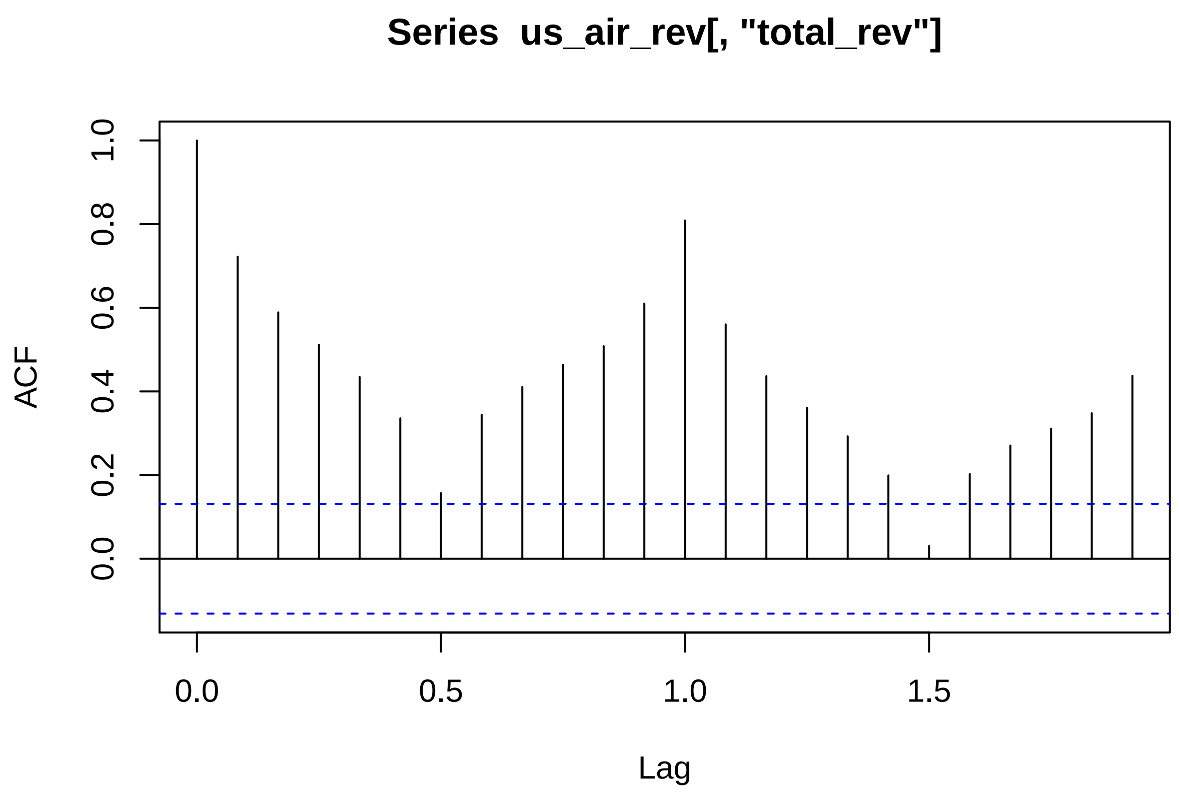
We now want you to do the following:

(1) Create a time series object using the `ts` command for each series. Be sure to specify the correct frequency for the data.
(2) Plot an autocorrelation function between our two time series: $\{Rev_t\}_{t\in T}$ and $\{Load_t\}_{t\in T}$.
(3) Run the following linear regression, reporting the coefficients and R^2:

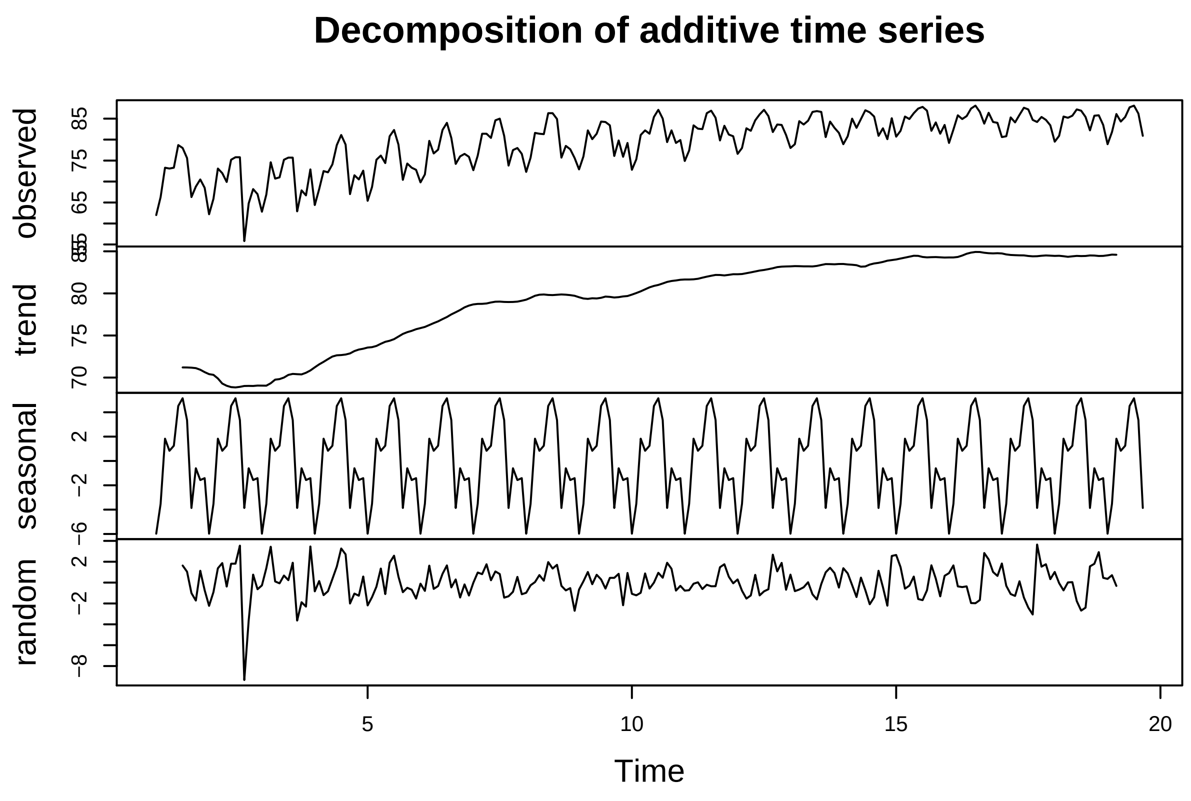
$$Rev_t = \alpha + \beta Load_t + \epsilon_t$$

(4) Decompose both series into cyclical and trend components using the `decompose` command. Plot seperately these cyclical and trend components for each of the series.
(5) Extract the trend component in part 4, redo parts (2) and (3). What differs from part (2)? Why? What can we conclude about the impact of capacity utilization changes on revenues?

```r
## ~~ Problem 2: Part (b) Time Series Analysis ~~ ##
setwd('C:\\Users\\ryanl\\Dropbox\\r_assignments\\assignments\\assignment_1\\data\\')
# (1) Convert to time series #
us_load_factor = read.csv('us_load_factor.csv')
us_air_rev = read.csv('us_air_rev.csv')
us_air_rev = ts(us_air_rev,frequency=12)
us_load_factor = ts(us_load_factor,frequency=12)

# (2) Plot an autocorrelation function #
acf(us_air_rev[,'total_rev'])
```



```
# (3) Run linear regression of revenue on load factor #
lin_reg = merge(us_air_rev,us_load_factor,by=c('date'))
lm(as.formula('total_rev ~ load_factor'),lin_reg)

# (4) Use decompose function #
plot(decompose(us_load_factor[,'load_factor']))
```



```
# (5) Extracting trend components #
load_factor = data.frame(date=us_load_factor[,'date'],
                         load_factor=as.vector(decompose(us_load_factor[,'load_factor'],'additive')$tren
air_rev = data.frame(date=us_air_rev[,'date'],
                     total_rev=as.vector(decompose(us_air_rev[,'total_rev'],'additive')$trend))
lin_reg_trend = merge(load_factor,air_rev,by=c('date'))
lm(as.formula('total_rev ~ load_factor'),lin_reg_trend)
```

```
Call:
lm(formula = as.formula("total_rev ~ load_factor"), data = lin_reg)

Coefficients:
(Intercept)  load_factor
  -23119353       889944


Call:
lm(formula = as.formula("total_rev ~ load_factor"), data = lin_reg_trend)

Coefficients:
(Intercept)  load_factor
  -14654893       780709
```

Problem 3 (Coding Exercise)

In class, you have learned about the Wold decomposition, a fundamental result in time series analysis. This exercise will attempt to walk through Wold's theorem in practice. We have provided simulated time series data, in an .rda file called "ts_simulation.rda", where Y_t is the t^{th} observation from our data. To open this file, use the load command. The name of the dataframe is "sim".

We now want you to do the following:

(1) Verify the stationarity of the process. Do this in two ways:

(a) "Heuristic" : show that the first-moment and second-moment do not depend on t.
(b) "Testing" : use a Dickey-Fuller test to test for stationarity. Interpret your results.

(2) Estimate three seperate autoregressive models: AR(1), AR(3) and AR(6). For each of the seperate models, retrieve the residuals, $\hat{\epsilon}_{\{t,p\}}$, where p is the order of the AR process. Using each set of residuals of the AR process, estimate an MA(2) model, where $\hat{\eta}_{\{t,p,q\}}$ are the residuals of this second step.

Verify whether the assumptions of Wold are violated:

$$
\begin{aligned}
Corr[\hat{\epsilon}_{\{t,p\}} Y_s] &= 0 \text{ such that } s < t \\
E[\eta_{\{t,p,q\}}] &= 0 \\
Var(\eta_{\{t,p,q\}}) &= \sigma^2
\end{aligned}
$$

(3) To find the right ARMA(p,q) process, we add new lags (increase p), estimate our model, use an information criteria to determine the increase in fit and stop once new models do not improve fit. To simplify the problem, assume $q = 2$. Build a series of ARMA(p,q) models, using the Akaike Information Criteria (AIC) to find the right p. (Note: A `for loop` over p would be a good idea).

```
## ~~ Problem 3: Wold-Decomposition ~~ ##
setwd('C:\\Users\\ryanl\\Dropbox\\r_assignments\\assignments\\assignment_1\\data\\')
# (0) Load simulated data #
load('ts_simulation.rda')

# (1a) Verify first and second moments do not depend on t #
cat(paste('Mean obs t = 301 to t = 600 ', mean(sim[301:600,'Y_t']),"\n",sep=''))
cat(paste('Mean obs t = 1 to t = 300 ', mean(sim[1:300,'Y_t']),"\n",sep=''))
cat(paste('Var obs t = 301 to t = 600 ', var(sim[301:600,'Y_t']),"\n",sep=''))
cat(paste('Var obs t = 1 to t = 300 ', var(sim[1:300,'Y_t']),"\n",sep=''))

# (1b) Use dickey-fuller test #
adf.test(sim[,'Y_t'])
```

```
Warning in adf.test(sim[, "Y_t"]): p-value smaller than printed p-value
```

```
# (2) Fit autoregression models #

# Fit ar 1, 3 and 6
ar_1 = arima(sim[,'Y_t'], order = c(1,0,0))
ar_3 = arima(sim[,'Y_t'], order = c(3,0,0))
ar_6 = arima(sim[,'Y_t'], order = c(6,0,0))

# Retrieve residuals
resid_ar1 = ar_1$residuals
resid_ar3 = ar_3$residuals
resid_ar6 = ar_6$residuals
```

```r
# Estimate ma 2
ar1_ma2 = arima(resid_ar1[!is.na(resid_ar1)], order = c(0,0,2))
ar3_ma2 = arima(resid_ar3[!is.na(resid_ar3)], order = c(0,0,2))
ar6_ma2 = arima(resid_ar6[!is.na(resid_ar6)], order = c(0,0,2))

# Verify Wold - we will just look at mean of the residuals
cat(paste('Mean of AR(1) + MA(2) ', mean(ar1_ma2$residuals,na.rm=T),"\n",sep=''))

# (3) Find the right lag order #
last_aic = 10000000
for(p in 1:10){
  # Fit arma model
  arma_model = arima(sim[,'Y_t'],order=c(p,0,2))
  # Find AIC
  aic = AIC(arma_model)
  # If AIC is lower than last evaluation stop iteratiors
  if(aic >= last_aic){
    break
  }
  last_aic = aic
}
cat(paste('Order of p that best fits the model :',p,"\n",sep=''))
```

```
Mean obs t = 301 to t = 600 -0.122834316978193
Mean obs t = 1 to t = 300 -0.136129225673853
Var obs t = 301 to t = 600 6.01700150063216
Var obs t = 1 to t = 300 5.43213094864443

    Augmented Dickey-Fuller Test

data:  sim[, "Y_t"]
Dickey-Fuller = -7.2993, Lag order = 9, p-value = 0.01
alternative hypothesis: stationary

Mean of AR(1) + MA(2) -0.000130552268349102
Order of p that best fits the model :2
```