Problem Set 1

Prasanna Parasurama

Due: 2/15/19

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.8
## v tidyr   0.8.2     v stringr 1.3.1
## v readr   1.2.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(dplyr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
library(ggplot2)
library(grid)
library(zoo)
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(dynlm)
```

```
## Warning: package 'dynlm' was built under R version 3.5.2
```

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?

   Distribution of Y_1

$$Y_1 = \alpha Y_0 + \epsilon_1$$

Since $Y_0 = 0$,

$$Y_1 = \epsilon_1 \sim N(0, \sigma^2)$$

Distribution of Y_2

$$Y_2 = \alpha Y_1 + \epsilon_2$$
$$Y_1 \sim N(0, \sigma^2)$$
$$\alpha Y_1 \sim N(0, \alpha^2\sigma^2)$$

Since Y_2 is a sum of 2 random normal variables,

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

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

   We can generalize the distribution for Y_t as follows:

   $$Y_t \sim N(0, \sigma^2(1 + \alpha^2 + \alpha^4 + ...\alpha^{2t}))$$

   The power series converges to $\frac{1}{1-\alpha^2}$. Therefore:

   $$Y_t \sim N(0, \sigma^2 \frac{1}{1 - \alpha^2}), t \to \infty$$

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 means that the distribution of Y is invariant to time. In this model, since Y is normally distributed, it is fully characterized by its mean and variance. If the mean and variance are independent of time, then the series is stationary.

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?

   Yes, $\alpha = 1$ means that the variance increases with time.

   Taking the first difference $\Delta Y = Y_t - Y_{t-1} = \epsilon_t$ will make this stationary.

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

   $$\rho_j = corr(Y_t, Y_{t-j}) = \frac{Cov(Y_t, Y_{t-j})}{Var(Y_t)}$$

   Consider Y_t when it is stationary (as t becomes large). From part (2), we know that:

   $$Var(Y_t) = \frac{\sigma^2}{1 - \alpha^2}$$
   $$Cov(Y_t, Y_{t-j}) = E[Y_t Y_{t-j}] - E[Y_t]E[Y_{t-j}] = E[Y_t Y_{t-j}]$$

   Consider a simple case, where $j = 1$

   $$Cov(Y_t, Y_{t-1}) = E[Y_t Y_{t-1}]$$

   Substituting the AR(1) model formula for T_t:

   $$E[Y_t Y_{t-1}] = E[(\alpha Y_{t-1} + \epsilon_t)Y_{t-1}] = E[\alpha Y_{t-1} Y_{t-1}] + E[\epsilon_t Y_{t-1}]$$

Note that $E[\epsilon_t Y_{t-1}] = 0$, and $E[\alpha Y_{t-1} Y_{t-1}]$ is simply $\alpha Var(Y_{t-1})$

Since Y is stationary,

$$Var(Y_{t-1}) = Var(Y_t)$$

$$Cov(Y_t, Y_{t-1}) = \alpha Var(Y_t) = \frac{\alpha\sigma^2}{1-\alpha^2}$$

If we recurse for any j as in part(1), we can get the general formula:

$$Cov(Y_t, Y_{t-j}) = \frac{\alpha^j\sigma^2}{1-\alpha^2}$$

Plugging these back into the original equation:

$$\rho_j = \frac{Cov(Y_t, Y_{t-j})}{Var(Y_t)} = \frac{\alpha^j\sigma^2}{1-\alpha^2}\frac{1}{\frac{\sigma^2}{1-\alpha^2}} = \alpha^j$$

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.

   Estimate the sample correlation coeffecient:

$$\hat{\rho}(j) = \frac{1}{T}\sum_{t=j+1}^{T}(Y_t - \bar{Y})(Y_{t-j} - \bar{Y})$$

   $\hat{\rho}(j)$ will tend to 0, as j increases. That is, current data will be less and less predictive the further we go into the future.

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) 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

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

## Load Air Revenue Data ##
df_rev = read_csv("../data/us_air_rev.csv")
```

```
Parsed with column specification:
cols(
  date = col_character(),
  total_rev = col_double()
)
```

```
## Load Air Load Factor Data ##
df_load = read_csv("../data/us_load_factor.csv")
```

```
Parsed with column specification:
cols(
  date = col_character(),
  load_factor = col_double()
)
```

```
#convert to date
df_rev = df_rev %>% mutate(date=as.Date(date, format="%m/%d/%Y"))
df_load = df_load %>% mutate(date=as.Date(date, format="%m/%d/%Y"))
```

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.

```
print(colnames(df_rev))
print(colnames(df_load))
```

```
## [1] "date"      "total_rev"
## [1] "date"        "load_factor"
```

(2) Print off the first 20 rows.

```
head(df_rev, 20)
head(df_load, 20)
```

```
## # A tibble: 20 x 2
##    date       total_rev
```

```
##    <date>         <dbl>
##  1 2000-01-01  34913749
##  2 2000-02-01  36005847
##  3 2000-03-01  43901516
##  4 2000-04-01  41954244
##  5 2000-05-01  43168824
##  6 2000-06-01  45856363
##  7 2000-07-01  47303803
##  8 2000-08-01  46457758
##  9 2000-09-01  38372644
## 10 2000-10-01  41820446
## 11 2000-11-01  40712967
## 12 2000-12-01  39656140
## 13 2001-01-01  37179562
## 14 2001-02-01  35784235
## 15 2001-03-01  44208143
## 16 2001-04-01  42363481
## 17 2001-05-01  42517548
## 18 2001-06-01  45304289
## 19 2001-07-01  48067498
## 20 2001-08-01  48579798
## # A tibble: 20 x 2
##    date       load_factor
##    <date>           <dbl>
##  1 2000-01-01        62
##  2 2000-02-01        66.3
##  3 2000-03-01        73.3
##  4 2000-04-01        73.1
##  5 2000-05-01        73.3
##  6 2000-06-01        78.7
##  7 2000-07-01        78
##  8 2000-08-01        75.6
##  9 2000-09-01        66.3
## 10 2000-10-01        68.8
## 11 2000-11-01        70.5
## 12 2000-12-01        68.5
## 13 2001-01-01        62.2
## 14 2001-02-01        65.9
## 15 2001-03-01        73.1
## 16 2001-04-01        72
## 17 2001-05-01        69.9
## 18 2001-06-01        75.2
## 19 2001-07-01        75.8
## 20 2001-08-01        75.8
```

(3) Print off the number of rows and columns.

```
dim(df_rev)
dim(df_load)
```

```
## [1] 223   2
## [1] 225   2
```

(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.

```
lapply(df_rev, typeof)
lapply(df_load, typeof)
```

```
## $date
## [1] "double"
##
## $total_rev
## [1] "double"
##
## $date
## [1] "double"
##
## $load_factor
## [1] "double"
```

(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`.

```
print("Avg Yearly Revenue")
avg_yearly_rev = df_rev %>% group_by("year"=year(date)) %>% summarise("avg_revenue"=mean(total_rev))
avg_yearly_rev
df_rev %>% group_by("year"=year(date)) %>%
        summarise("avg_revenue"=sum(total_rev)/length(total_rev))

print("Avg Yearly Load")
avg_yearly_load = df_load %>% group_by("year"=year(date)) %>% summarise("avg_load"=mean(load_factor))
avg_yearly_load
print(df_load %>% group_by("year"=year(date)) %>% summarise("avg_load"=sum(load_factor)/length(load_fac
```

```
[1] "Avg Yearly Revenue"
# A tibble: 19 x 2
    year avg_revenue
   <dbl>       <dbl>
 1  2000   41677025.
 2  2001   39383247.
 3  2002   39163678.
 4  2003   41061093.
 5  2004   45234736
 6  2005   47436304.
 7  2006   47876604.
 8  2007   49360651.
 9  2008   47353907
10  2009   44915178.
```

```
11  2010   46071164.
12  2011   46970585.
13  2012   47390205.
14  2013   48157386.
15  2014   49611603.
16  2015   52554844.
17  2016   54998140.
18  2017   56979170.
19  2018   59997018.
# A tibble: 19 x 2
    year avg_revenue
   <dbl>       <dbl>
 1  2000   41677025.
 2  2001   39383247.
 3  2002   39163678.
 4  2003   41061093.
 5  2004   45234736
 6  2005   47436304.
 7  2006   47876604.
 8  2007   49360651.
 9  2008   47353907
10  2009   44915178.
11  2010   46071164.
12  2011   46970585.
13  2012   47390205.
14  2013   48157386.
15  2014   49611603.
16  2015   52554844.
17  2016   54998140.
18  2017   56979170.
19  2018   59997018.
[1] "Avg Yearly Load"
# A tibble: 19 x 2
    year avg_load
   <dbl>    <dbl>
 1  2000     71.2
 2  2001     68.8
 3  2002     70.2
 4  2003     72.6
 5  2004     74.4
 6  2005     77.1
 7  2006     79.0
 8  2007     79.8
 9  2008     79.6
10  2009     80.9
11  2010     82.0
12  2011     82.7
13  2012     83.2
14  2013     83.4
15  2014     84.4
16  2015     84.9
17  2016     84.5
18  2017     84.5
19  2018     84.4
```

```
# A tibble: 19 x 2
    year avg_load
   <dbl>    <dbl>
 1  2000     71.2
 2  2001     68.8
 3  2002     70.2
 4  2003     72.6
 5  2004     74.4
 6  2005     77.1
 7  2006     79.0
 8  2007     79.8
 9  2008     79.6
10  2009     80.9
11  2010     82.0
12  2011     82.7
13  2012     83.2
14  2013     83.4
15  2014     84.4
16  2015     84.9
17  2016     84.5
18  2017     84.5
19  2018     84.4
```

(2) Calculate the average revenue and load factor, by month. Do this two ways: (1) Using `aggregate` and `mean`, (2) Using `aggregate` and `sum`.

```
print("Avg Monthly Revenue")
avg_monthly_rev = df_rev %>% group_by("month"=month(date)) %>%
  summarise("avg_revenue"=mean(total_rev))
avg_monthly_rev
df_rev %>% group_by("month"=month(date)) %>%
        summarise("avg_revenue"=sum(total_rev)/length(total_rev))


print("Avg Monthly Load")
avg_monthly_load = df_load %>% group_by("month"=month(date)) %>%summarise("avg_load"=mean(load_factor))
avg_monthly_load
df_load %>% group_by("month"=month(date)) %>% summarise("avg_load"=sum(load_factor)/length(load_factor))
```

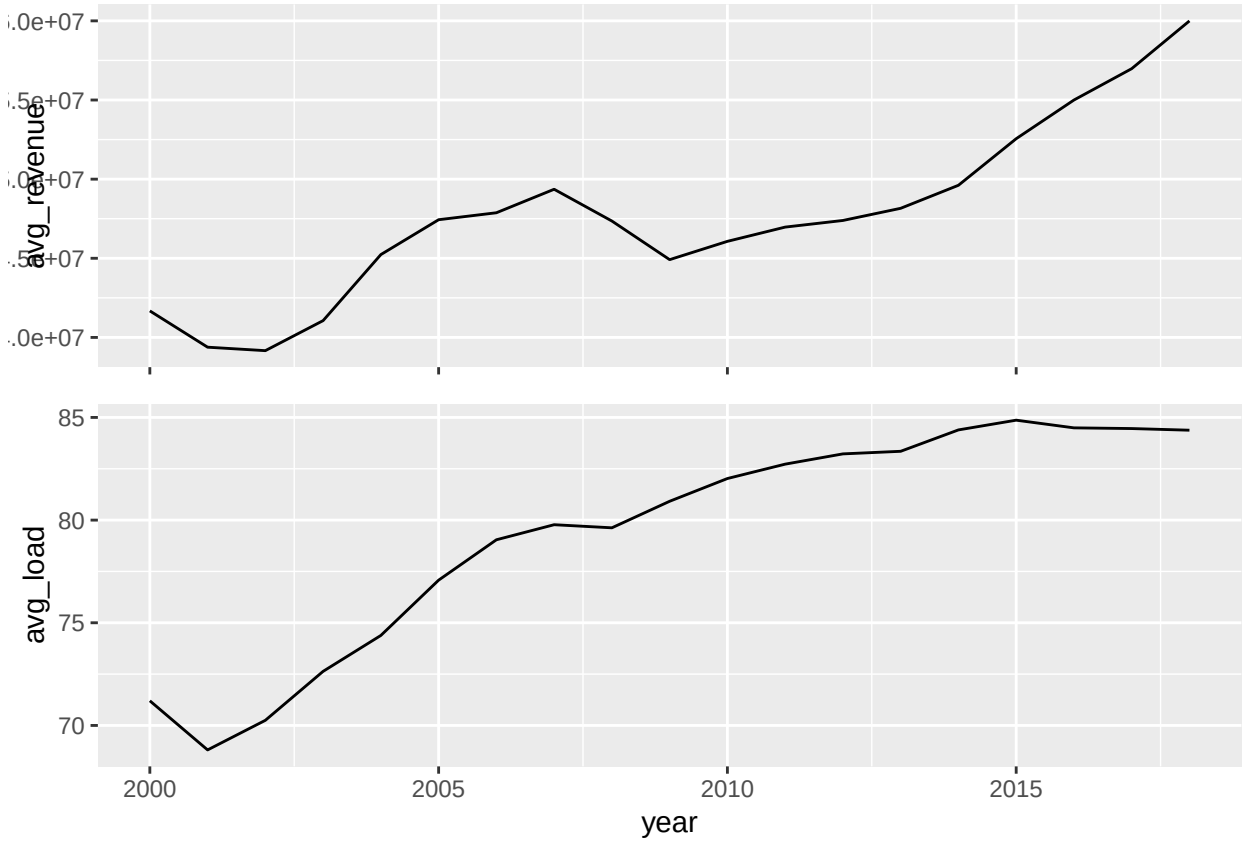
```
## [1] "Avg Monthly Revenue"
## # A tibble: 12 x 2
##    month avg_revenue
##    <dbl>       <dbl>
##  1     1   41887203.
##  2     2   40092415
##  3     3   49960016.
##  4     4   47601541.
##  5     5   49327660.
##  6     6   52372782.
##  7     7   55037605.
##  8     8   52494201.
##  9     9   42685703.
## 10    10   46660990.
## 11    11   44422602.
## 12    12   46237624.
```

```
## # A tibble: 12 x 2
##    month avg_revenue
##    <dbl>       <dbl>
##  1     1   41887203.
##  2     2   40092415
##  3     3   49960016.
##  4     4   47601541.
##  5     5   49327660.
##  6     6   52372782.
##  7     7   55037605.
##  8     8   52494201.
##  9     9   42685703.
## 10    10   46660990.
## 11    11   44422602.
## 12    12   46237624.
## [1] "Avg Monthly Load"
## # A tibble: 12 x 2
##    month avg_load
##    <dbl>    <dbl>
##  1     1     72.9
##  2     2     75.5
##  3     3     81.0
##  4     4     80.0
##  5     5     80.5
##  6     6     83.9
##  7     7     84.5
##  8     8     82.7
##  9     9     75.7
## 10    10     78.7
## 11    11     77.8
## 12    12     78
## # A tibble: 12 x 2
##    month avg_load
##    <dbl>    <dbl>
##  1     1     72.9
##  2     2     75.5
##  3     3     81.0
##  4     4     80.0
##  5     5     80.5
##  6     6     83.9
##  7     7     84.5
##  8     8     82.7
##  9     9     75.7
## 10    10     78.7
## 11    11     77.8
## 12    12     78
```

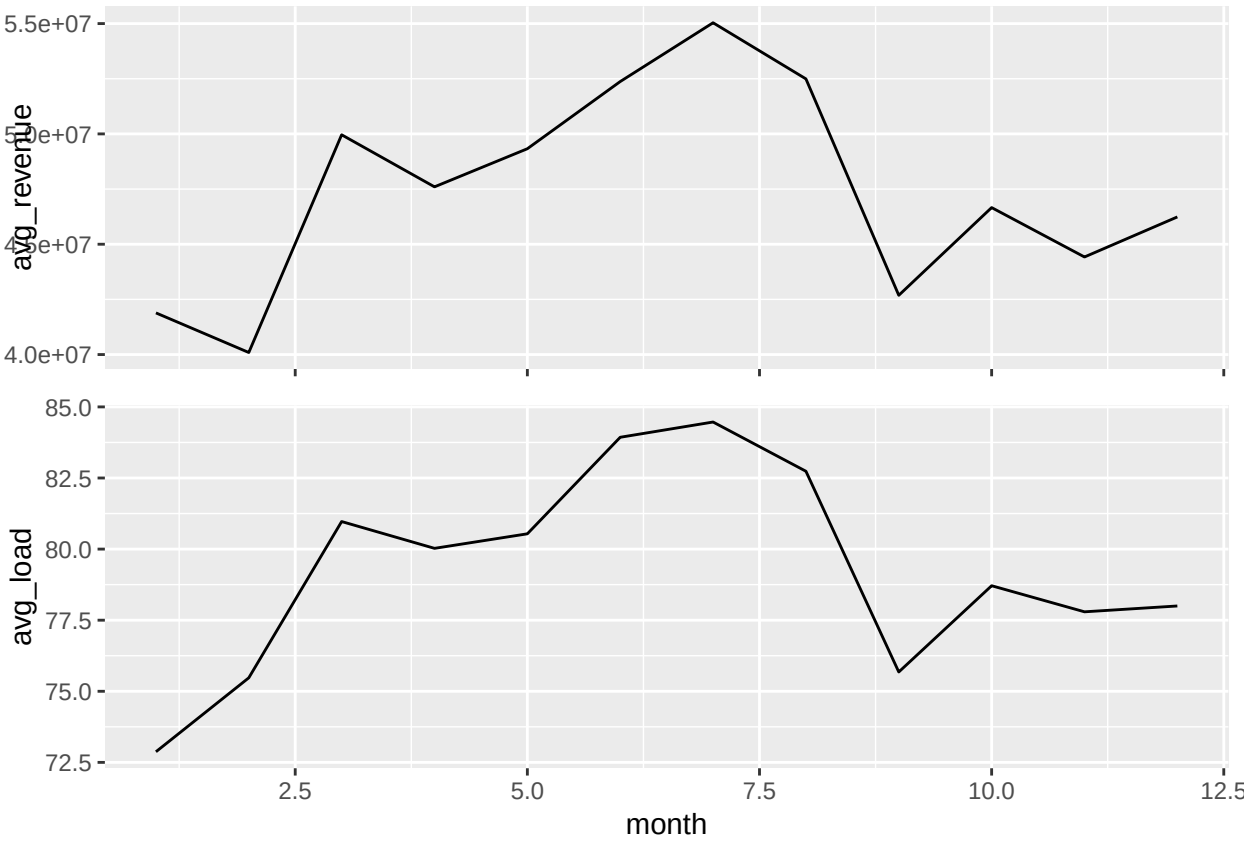
(3) Plot graphs for part (1) and (2) on the same plot, using your favorite plotting function. Note, you can either use the built-in `plot` function or the popular external library `ggplot2`.

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.

```
p1 = ggplot() + geom_line(data=avg_yearly_rev, aes(x=year, y=avg_revenue)) + theme(axis.text.x = elemen
p2 = ggplot() + geom_line(data=avg_yearly_load, aes(x=year, y=avg_load))
grid.newpage()
grid.draw(rbind(ggplotGrob(p1), ggplotGrob(p2), size="last"))
```




```
p3 = ggplot() + geom_line(data=avg_monthly_rev, aes(x=month, y=avg_revenue)) + theme(axis.text.x = eleme
p4 = ggplot() + geom_line(data=avg_monthly_load, aes(x=month, y=avg_load))
grid.newpage()
grid.draw(rbind(ggplotGrob(p3), ggplotGrob(p4), size="last"))
```



(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

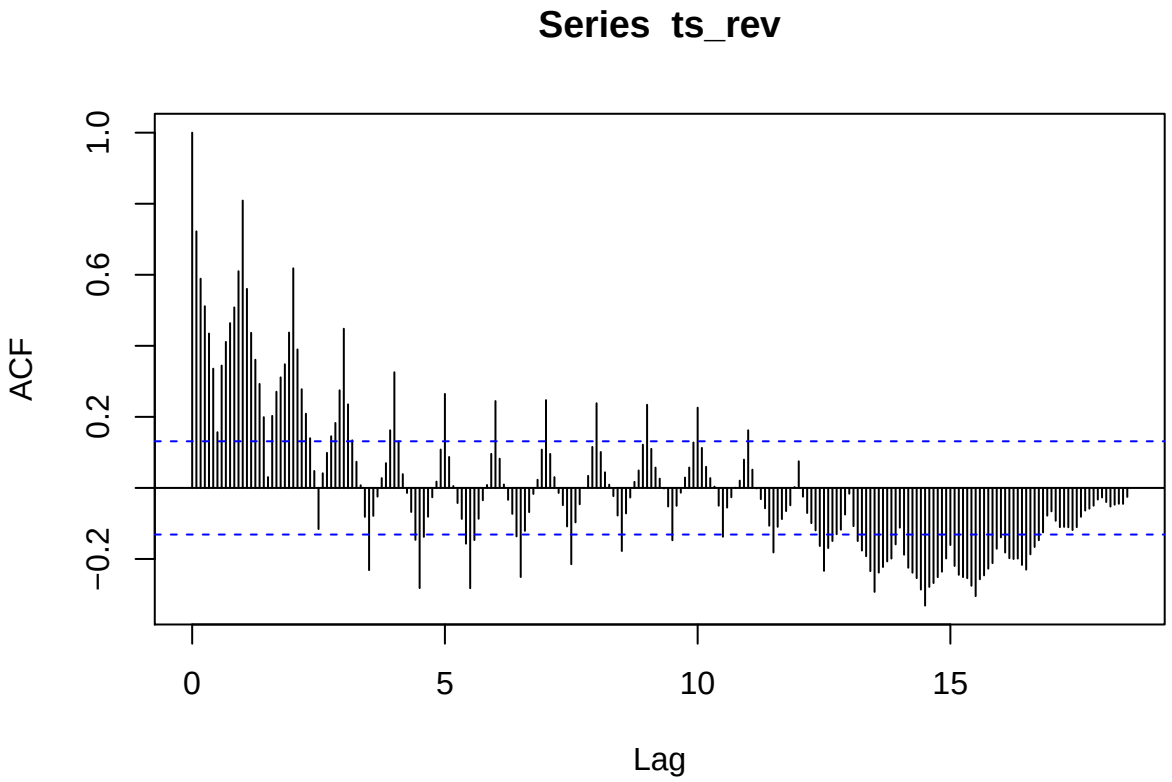
We now want you to do the following:

(1) Create a time series object using the `ts` command for each series. Be sure to specify the correct frequency for the data.

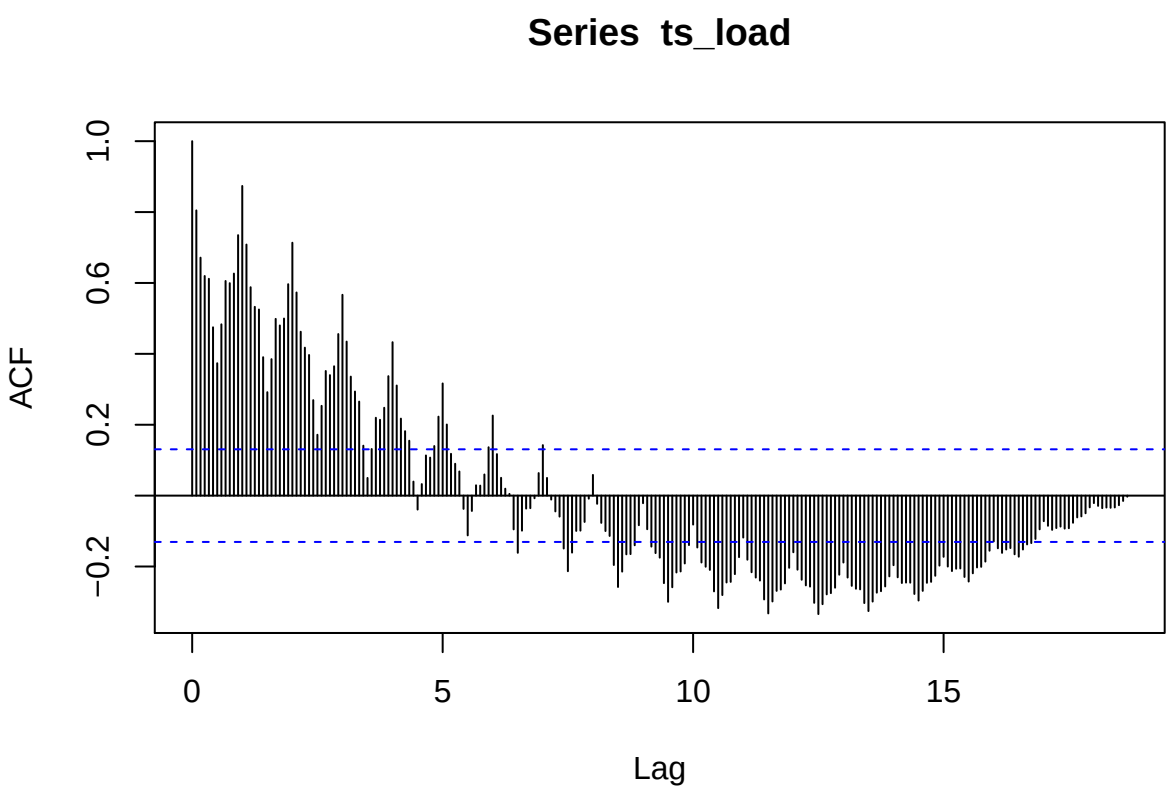
```
# convert to timeseries
ts_rev = ts(df_rev$total_rev, start = c(2000,1), end = c(2018,7), frequency = 12)
ts_load = ts(df_load$load_factor, start = c(2000,1), end = c(2018,9), frequency = 12)
```

(2) Plot an autocorrelation function between our two time series: $\{Rev_t\}_{t \in T}$ and $\{Load_t\}_{t \in T}$.

```
# autocorrelation
acf(ts_rev, lag.max = 300)
```




```
acf(ts_load, lag.max = 300)
```



(3) Run the following linear regression, reporting the coefficients and R^2:

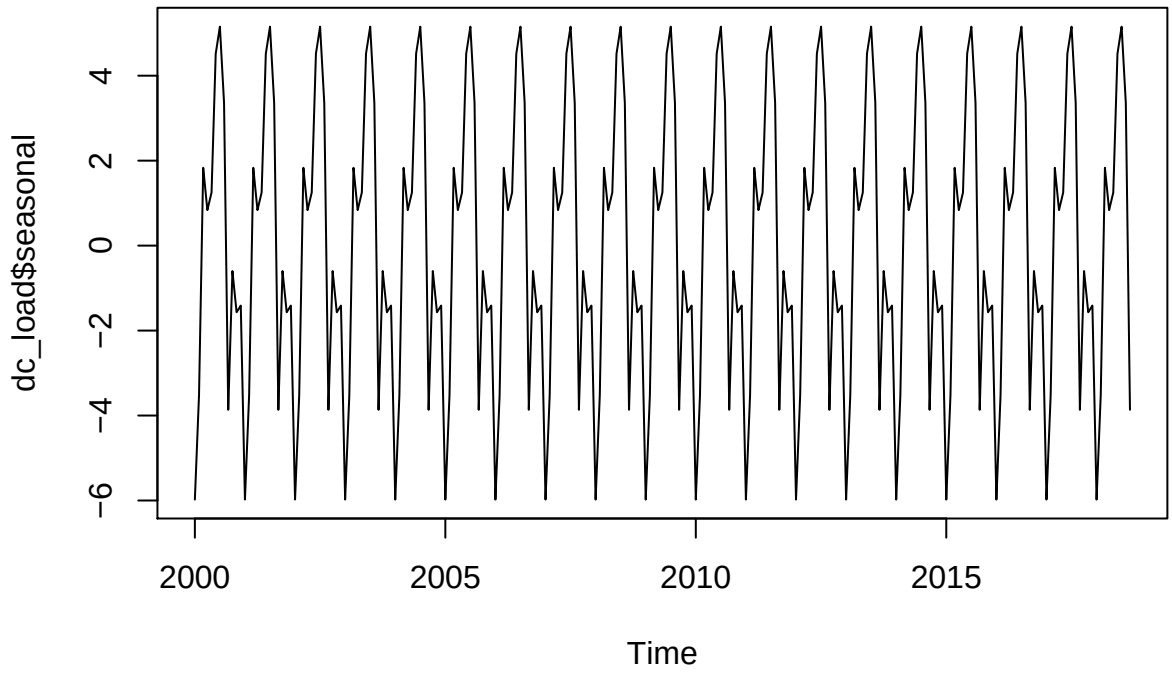
$$Rev_t = \alpha + \beta Load_t + \epsilon_t$$

```
# regression
df = inner_join(df_rev, df_load, by=c("date"))
m.lm = lm(total_rev ~ load_factor, data=df)
print(summary(m.lm))
```

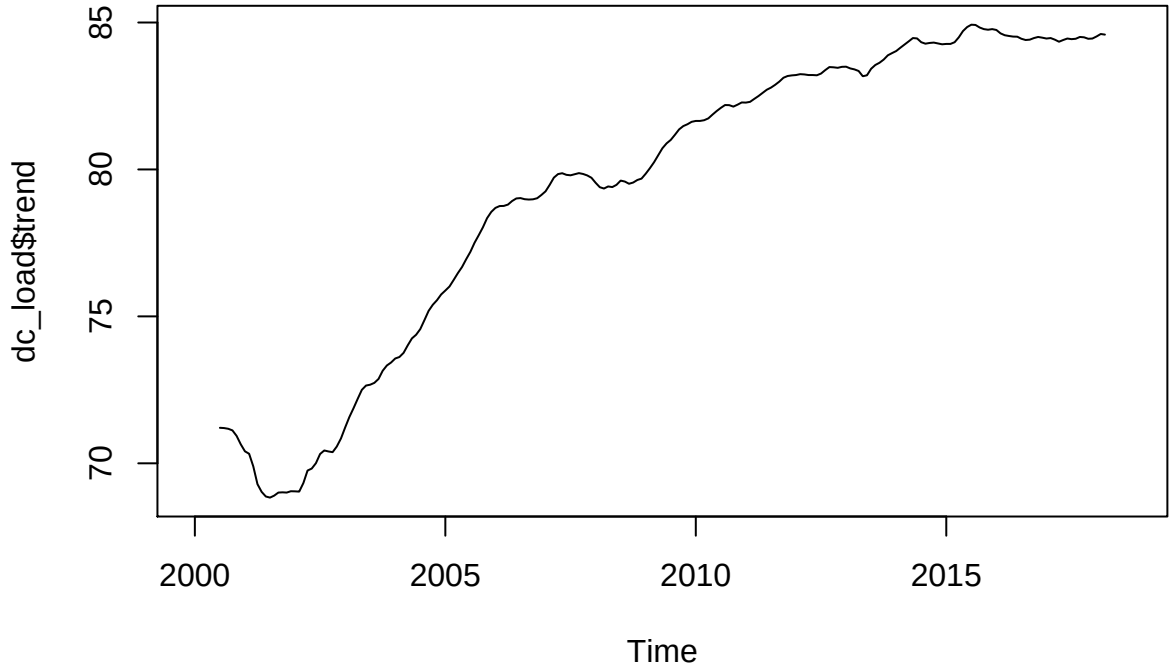
```
##
## Call:
## lm(formula = total_rev ~ load_factor, data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -9488829 -2660839   216944  2089950 13564306
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -25078775    3027552  -8.284 1.15e-14 ***
## load_factor    914872      38080  24.025  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3666000 on 221 degrees of freedom
## Multiple R-squared:  0.7231, Adjusted R-squared:  0.7219
## F-statistic: 577.2 on 1 and 221 DF,  p-value: < 2.2e-16
```

(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.

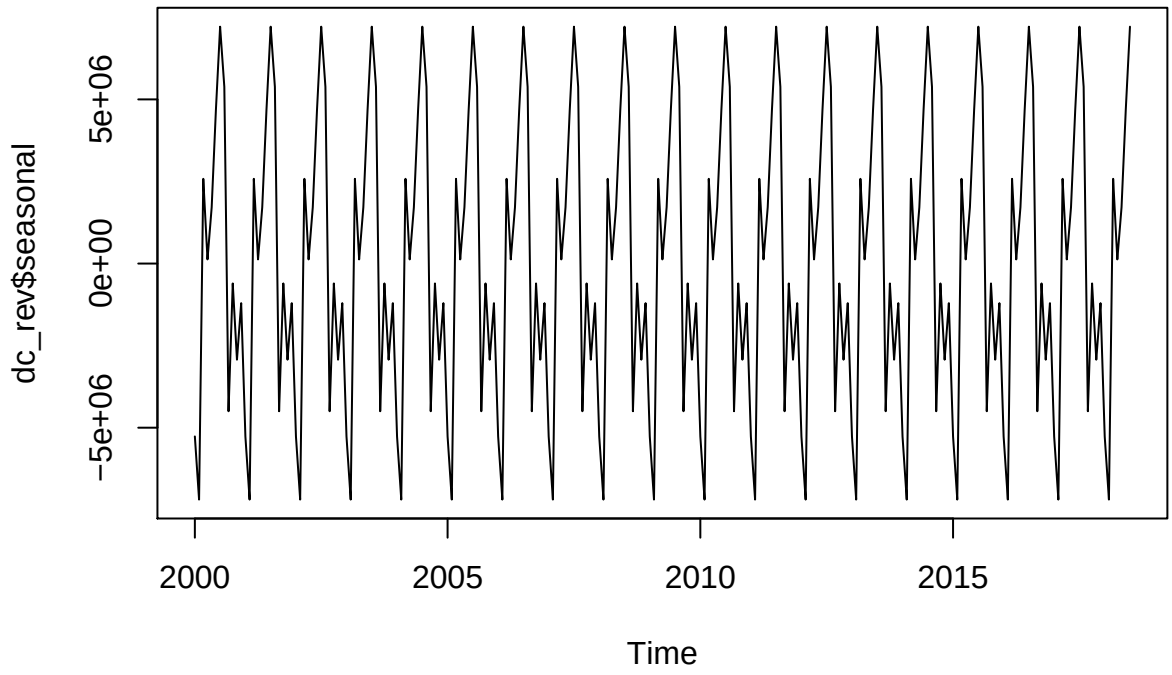
```
# decompose
dc_load = decompose(ts_load)
dc_rev = decompose(ts_rev)

plot(dc_load$seasonal)
```



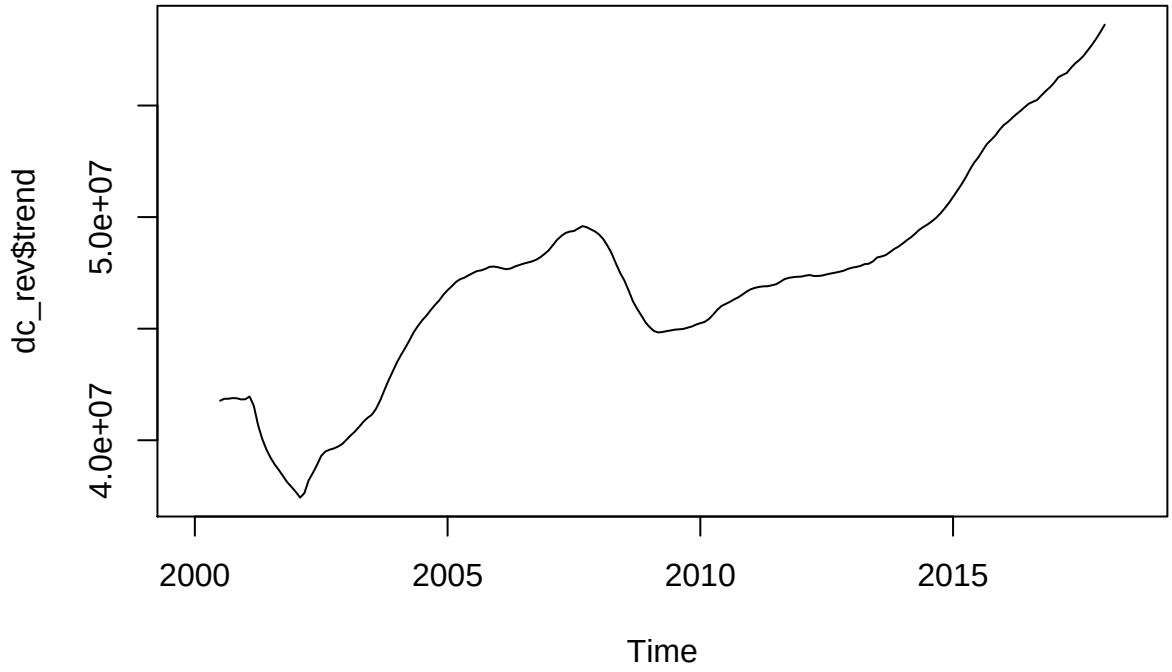

```
plot(dc_load$trend)
```




```
plot(dc_rev$seasonal)
```

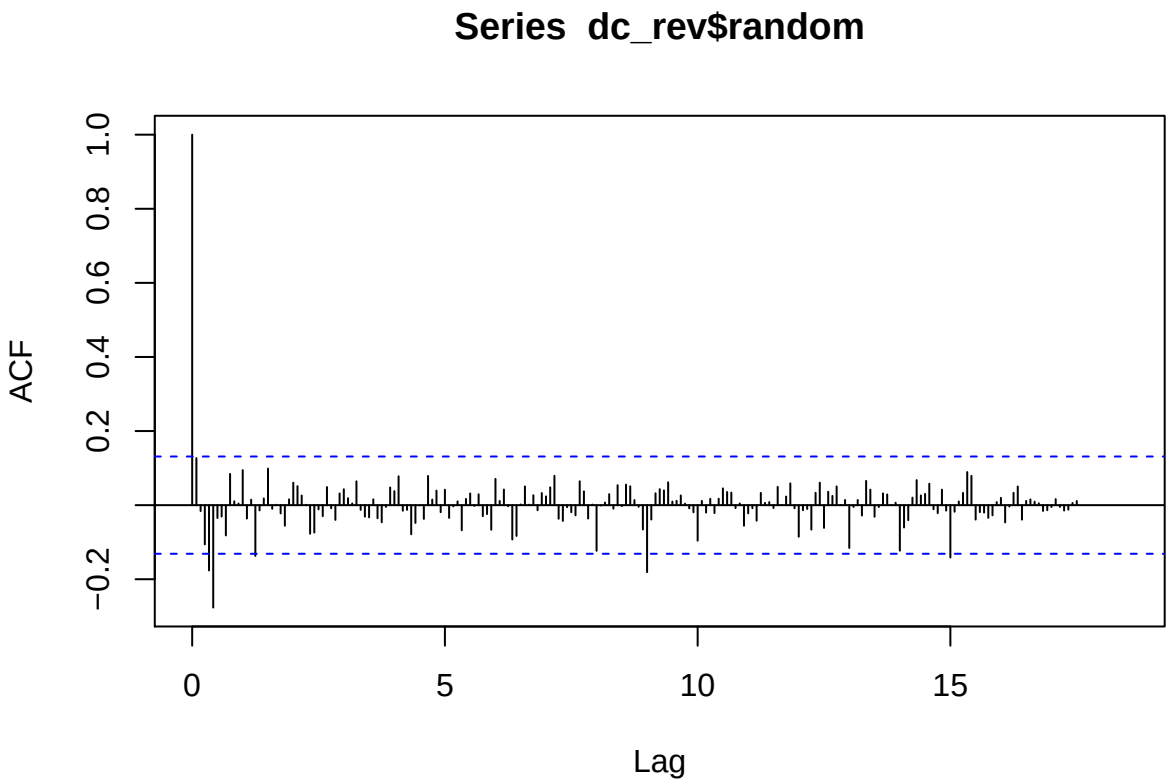



```
plot(dc_rev$trend)
```

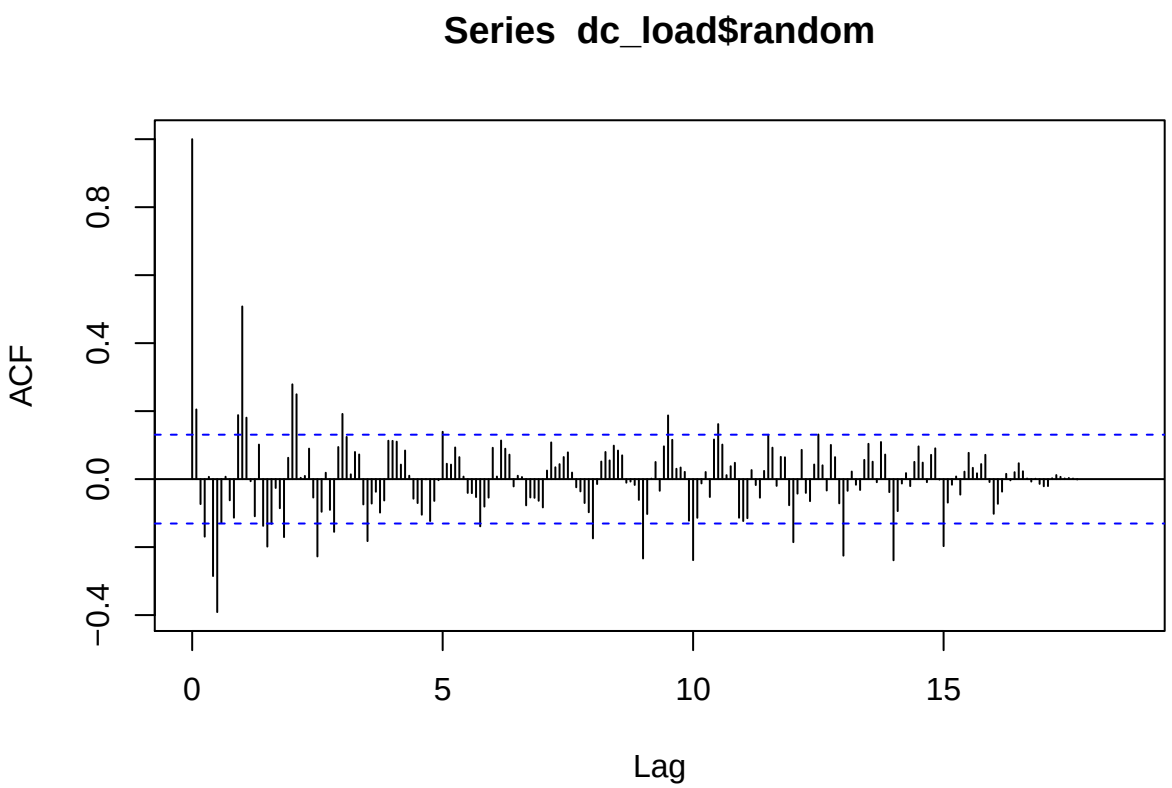


(5) Using the dataframe created in part (3), redo parts (2) and (3). What differs from part (2)? Why? What can we conclude about the impact of capacity utilization changes on revenues?

```
acf(dc_rev$random, lag.max = 300, na.action = na.pass)
```




```
acf(dc_load$random, lag.max=300, na.action = na.pass)
```




```
# join dataframes
rev_random = data.frame(revenue=as.matrix(dc_rev$random), date=time(dc_rev$random))
load_random = data.frame(load=as.matrix(dc_load$random), date=time(dc_load$random))
```

```
df_random = inner_join(rev_random, load_random, by=c("date"))
```

```
## Warning: Column `date` has different attributes on LHS and RHS of join
```

```
m.random_lm = lm(revenue ~ load, data=df_random)
summary(m.random_lm)
```

```
##
## Call:
## lm(formula = revenue ~ load, data = df_random)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4005935  -378889    95501   455232  2757324
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    38345      67637   0.567    0.571
## load          423482      42600   9.941   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 894600 on 173 degrees of freedom
##   (10 observations deleted due to missingness)
## Multiple R-squared:  0.3635, Adjusted R-squared:  0.3599
## F-statistic: 98.82 on 1 and 173 DF,  p-value: < 2.2e-16
```

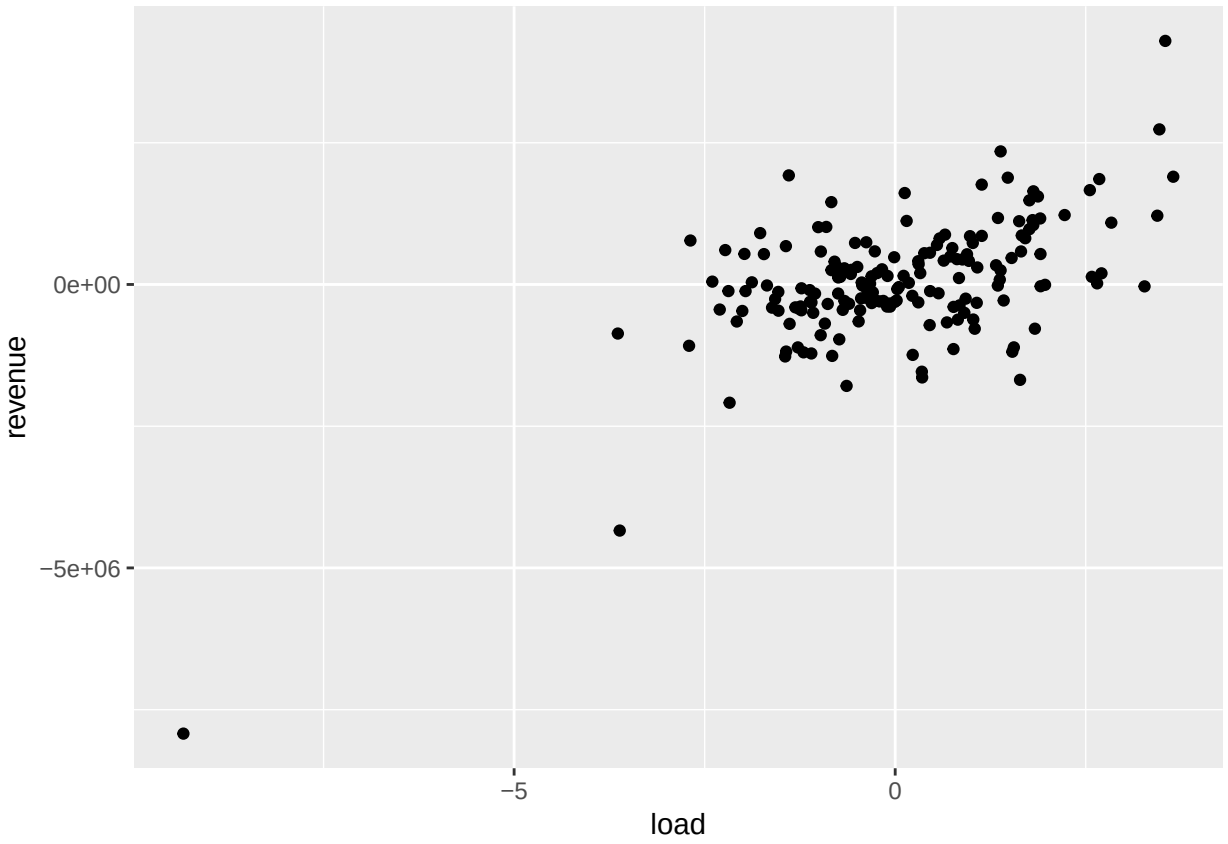
Seasonal and trend effects are mostly gone - i.e most of the bars are within dotted blue lines. In load factors, there's some artifacts of seasonality.

Even after taking out trend and seasonality, load factor is still predictive of revenue miles. Although, the estimate is much lower. A 0.1 increase in load factor corresponds to 42348 increase in revenue miles.

Below is a scatter plot of disturbences of revenue vs. load.

```
df_random %>% ggplot() + geom_point(aes(x=load, y = revenue))
```

```
## Warning: Removed 10 rows containing missing values (geom_point).
```



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".

```
load("../data/ts_simulation.rda")
```

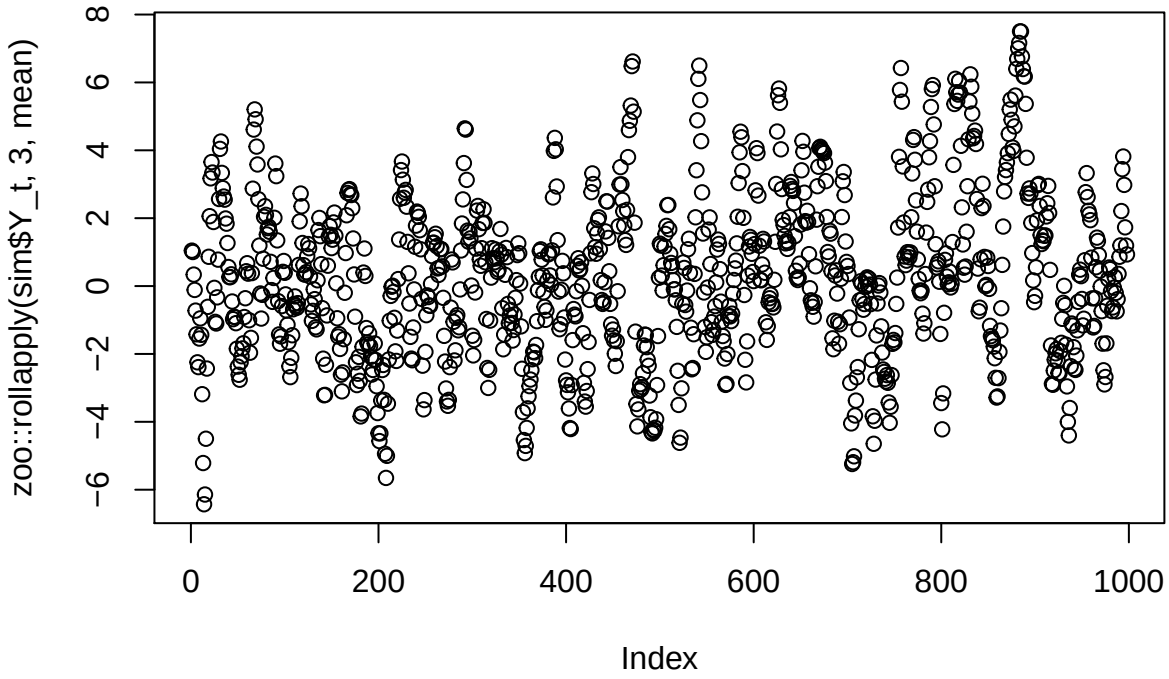
We now want you to do the following:

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

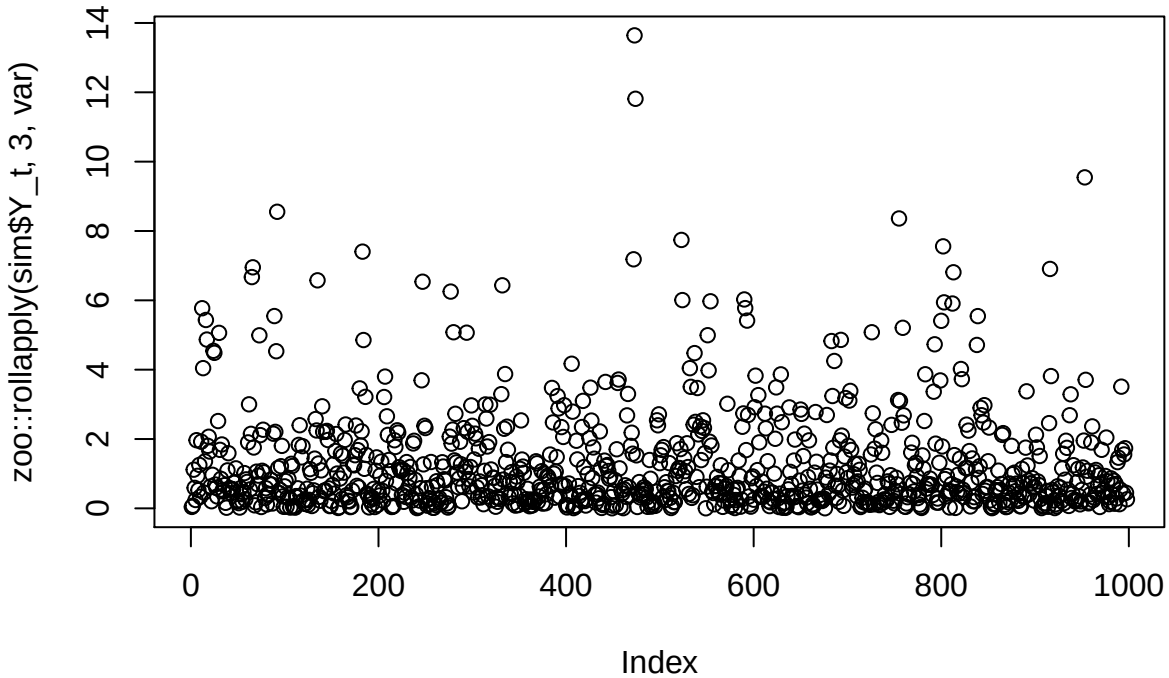
(a) "Heuristic" : show that the first-moment and second-moment do not depend on t.

We can select a window size (say, H=3). The mean and the variance should be randomly dispersed, as this window moves over time:

```
plot(zoo::rollapply(sim$Y_t, 3, mean))
```




```
plot(zoo::rollapply(sim$Y_t, 3, var))
```



We see that the both mean and variance are indeed randomly dispersed.

(b) "Testing" : use a Dickey-Fuller test to test for stationarity. Interpret your results.

We can test the following model, with the null that $\delta = 0$ with a one-sided test

$$\Delta Y_t = \beta_0 + \delta Y_{t-1} + \epsilon_t$$

```
sim_ts = ts(sim$Y_t)
summary(dynlm(d(sim_ts) ~ L(sim_ts,1)))
```

```
##
## Time series regression with "ts" data:
## Start = 2, End = 1000
##
## Call:
## dynlm(formula = d(sim_ts) ~ L(sim_ts, 1))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2227 -0.8199 -0.0240  0.8155  3.4356
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.03277    0.03796   0.863    0.388
## L(sim_ts, 1) -0.11207    0.01457  -7.693 3.45e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.192 on 997 degrees of freedom
## Multiple R-squared:  0.05603,    Adjusted R-squared:  0.05508
## F-statistic: 59.18 on 1 and 997 DF,  p-value: 3.448e-14
```

Since δ is negative and not 0, the series is stationary. (Question for later: We do a one-sided test. Why can delta not be positive?)

(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:

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

Estimate AR(1), AR(3) and AR(6)

```
ar1 = ar(sim_ts, order.max = 1)
ar3 = ar(sim_ts, order.max = 3)
ar6 = ar(sim_ts, order.max = 6, aic=FALSE)
```

MA(2) can be estimated with residuals from AR(1) and AR(2):

```
ar2 = ar(sim_ts, order.max = 2)

ma2 = dynlm(sim_ts ~ ar1$resid + ar2$resid)
summary(ma2)
```

```
##
## Time series regression with "ts" data:
## Start = 3, End = 1000
##
## Call:
## dynlm(formula = sim_ts ~ ar1$resid + ar2$resid)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.4369 -1.6062 -0.0724  1.4189  6.6246
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.2911382  0.0729160   3.993 7.01e-05 ***
## ar1$resid    1.0001798  0.1230854   8.126 1.31e-15 ***
## ar2$resid   -0.0001257  0.1418563  -0.001    0.999
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.303 on 995 degrees of freedom
##   (2 observations deleted due to missingness)
## Multiple R-squared:  0.2116, Adjusted R-squared:   0.21
## F-statistic: 133.6 on 2 and 995 DF,  p-value: < 2.2e-16
```

We can now test the Wold assumptions below.

Correlation assumtion:

```
# correlation assumtion
cor(ar1$resid, data.table::shift(sim_ts, 1), use="complete.obs")
cor(ar3$resid, data.table::shift(sim_ts, 3), use="complete.obs")
cor(ar6$resid, data.table::shift(sim_ts, 6), use="complete.obs")
```

```
## [1] 0.0001104776
## [1] -7.886547e-05
## [1] -0.0001946108
```

Note that they are all close to 0.

Expectation assumtion:

```
mean(ma2$residuals)
```

```
## [1] 9.680817e-17
```

Expectation of residuals is 0.

```
var(ma2$residuals)
```

```
## [1] 5.295459
```

Not sure what σ^2 is supposed to be...

(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).

```
for (p in c(1:6)) {
  arma = arima(sim_ts, order=c(p, 0, 2))
  print(paste("p=", p, " ", "AIC=", arma$aic))
}
```

```
Warning in arima(sim_ts, order = c(p, 0, 2)): possible convergence problem:
optim gave code = 1

Warning in arima(sim_ts, order = c(p, 0, 2)): possible convergence problem:
optim gave code = 1
```

```
[1] "p= 1   AIC= 2879.9433304044"
[1] "p= 2   AIC= 2881.1472257598"
[1] "p= 3   AIC= 2879.899347839"
[1] "p= 4   AIC= 2882.25025237178"
[1] "p= 5   AIC= 2879.55864168521"
[1] "p= 6   AIC= 2879.32794058952"
```

$p = 1$ gives a low AIC.