TimeSeries Skeleton

Fanny Chow, Michael McCormack

March 12, 2018

Step 1: Get the data.

Download from website https://cdn.rawgit.com/mikejt33/DataViz/246c2026/data/flights.csv.gz Easiest to unzip locally then read in the data as a csv file (hint: read.table() is typically faster than read.csv)

Easiest to unzip locally then read in the data as a csv file (hint: read.table() is typically faster than read.csv)

```
flights.path <- here("flights.csv")
flights <- read.table(flights.path, header = TRUE, sep=',')
```

Step 2: Prepare the data.

*Are there any null values?

Time series data needs to be over a regular time interval. Calculate the average departure delay time and/or average arrival delay time for each day of 2017.

Note the number of NA null values.

```
str(flights)
```

```
## 'data.frame':    895129 obs. of  20 variables:
##  $ DAY_OF_WEEK      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ FL_DATE          : Factor w/ 365 levels "2017-01-01","2017-01-02",..: 359 359 359 359 338 345 338
##  $ UNIQUE_CARRIER   : Factor w/ 11 levels "AA","AS","B6",..: 8 8 8 8 1 1 1 1 1 1 ...
##  $ AIRLINE_ID       : int  20304 20304 20304 20304 19805 19805 19805 19805 19805 19805 ...
##  $ CARRIER          : Factor w/ 11 levels "AA","AS","B6",..: 8 8 8 8 1 1 1 1 1 1 ...
##  $ TAIL_NUM         : Factor w/ 4108 levels "","N001AA","N002AA",..: 2368 2368 226 226 4074 3559 407
##  $ FL_NUM           : int  3522 3522 5404 5426 19 19 19 19 29 29 ...
##  $ ORIGIN           : Factor w/ 103 levels "ABQ","ACY","ALB",..: 4 35 66 70 26 26 56 56 56 56 ...
##  $ DEST             : Factor w/ 102 levels "ABQ","ACY","ALB",..: 35 4 69 65 56 56 26 26 9 9 ...
##  $ DEP_TIME         : int  815 1007 734 1227 855 857 1347 1346 1247 1247 ...
##  $ DEP_DELAY        : num  -5 2 4 2 -5 -3 -3 -4 -3 -3 ...
##  $ DEP_DEL15        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ ARR_DELAY        : num  -17 -6 -3 -3 -19 -11 -10 -14 -13 -7 ...
##  $ ARR_DEL15        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CANCELLED        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CANCELLATION_CODE: Factor w/ 5 levels "","A","B","C",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ DIVERTED         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ AIR_TIME         : num  47 55 158 187 134 125 163 161 159 165 ...
##  $ DISTANCE         : num  300 300 1144 1144 1121 ...
##  $ X                : logi  NA NA NA NA NA NA ...
```

```
# aggregate(DEP_DELAY~DAY_OF_WEEK, data = flights, mean)
# mean.dept.delay <- tapply(flights$FL_DATE, flights$DEP_DELAY, mean, na.rm=TRUE)

mean.dept.delay <- flights %>%
  arrange(FL_DATE) %>%
```

```
  group_by(factor(FL_DATE)) %>%
  summarise(mean(DEP_DELAY, na.rm = TRUE))
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
names(mean.dept.delay) <- c('FL_DATE', 'AVE_DELAY')
```

If you like, compare average delay times for different carriers or different airports by creating multiple time series.

Step 3: Create a ts object of the data.

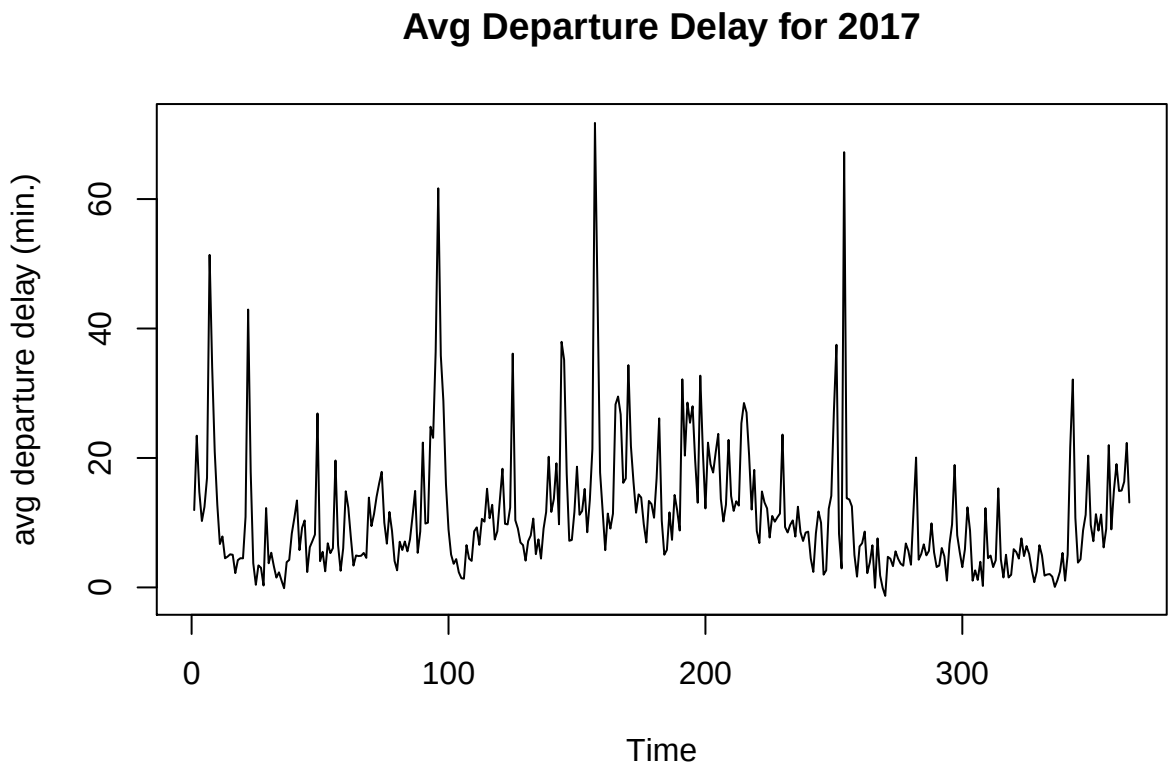
Refer to the slides for tips on how to do this.

```
mean.dept.delay.ts <- ts(mean.dept.delay$AVE_DELAY)
# str(mean.dept.delay)
```

Step 4: Plot the time series using base package and ggplot (advanced).

Create a basic visualization of the time series

```
plot(mean.dept.delay.ts, main="Avg Departure Delay for 2017", ylab="avg departure delay (min.)")
```



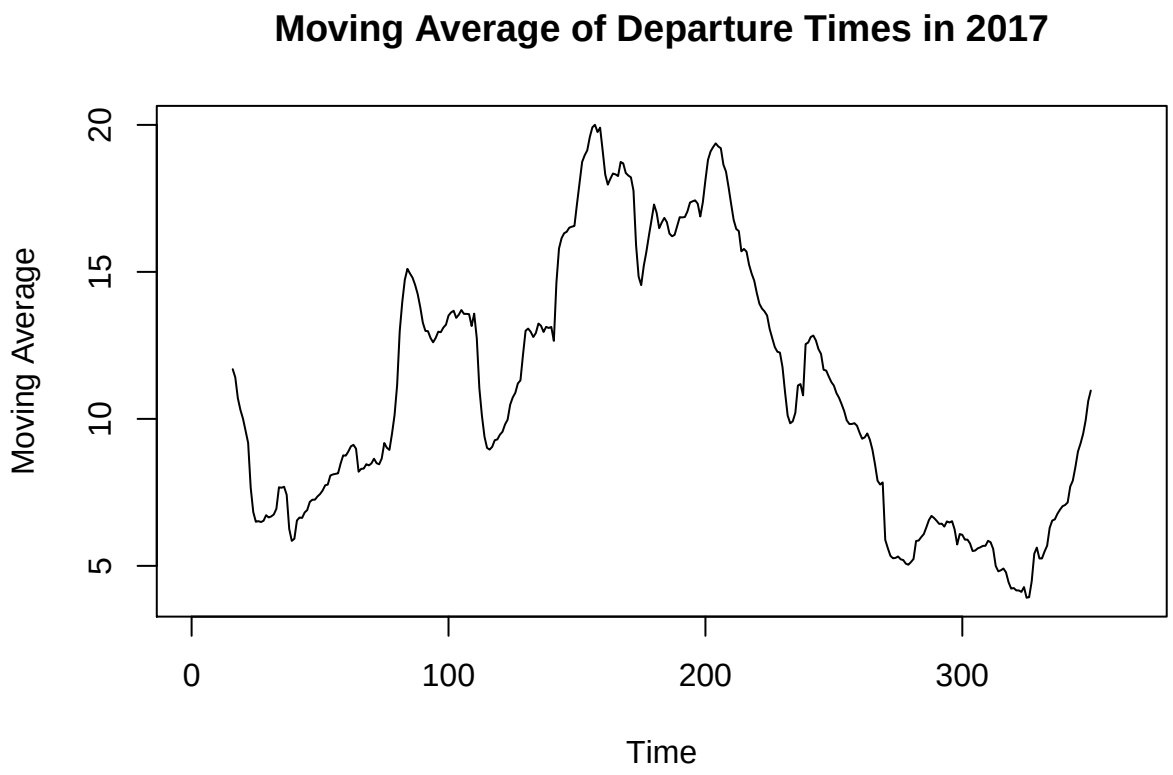
Step 5: Smooth the data to reduce noise and identify trends.

Create your own simple moving average for monthly data. Plot the smoothed data using base package. Plot both the originial and the smoothed data ggplot (advanced).

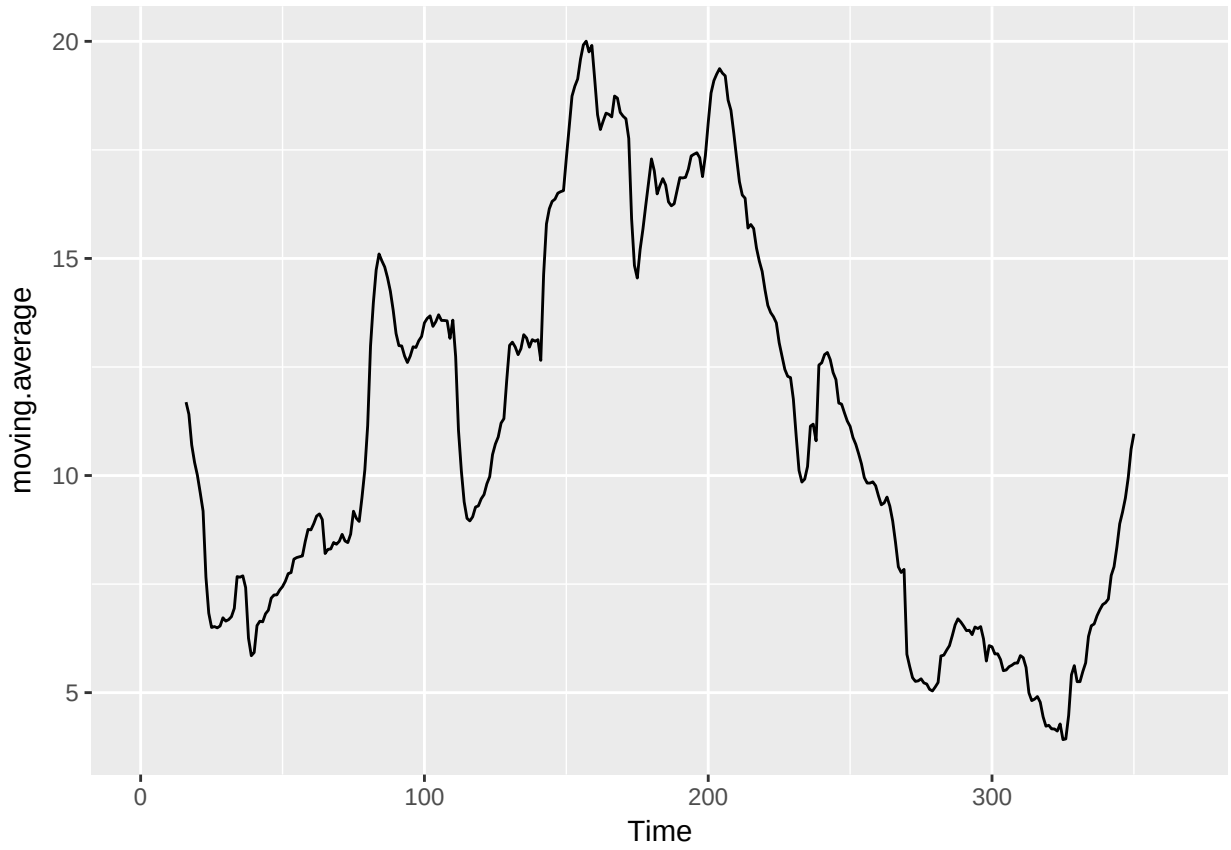
Hints
* good StackOverflow reference for moving average in R: https://stackoverflow.com/questions/743812/calculating-moving-average * watch out for functions that may have been masked by other packages * ggplot: may need to convert data to long format to plot mutliple series

Let's say there are 31 days per month for the window size.
1/n creates intervals. So say if you had quarterly data, we would bin it into 1/4 reps.

```
# n = window size
ma <- function(x,n=5){stats::filter(x,rep(1/n,n), sides=2)}

moving.average <- ma(mean.dept.delay.ts, 31)

plot(moving.average, ylab="Moving Average", main="Moving Average of Departure Times in 2017")
```




```
autoplot(moving.average)
```



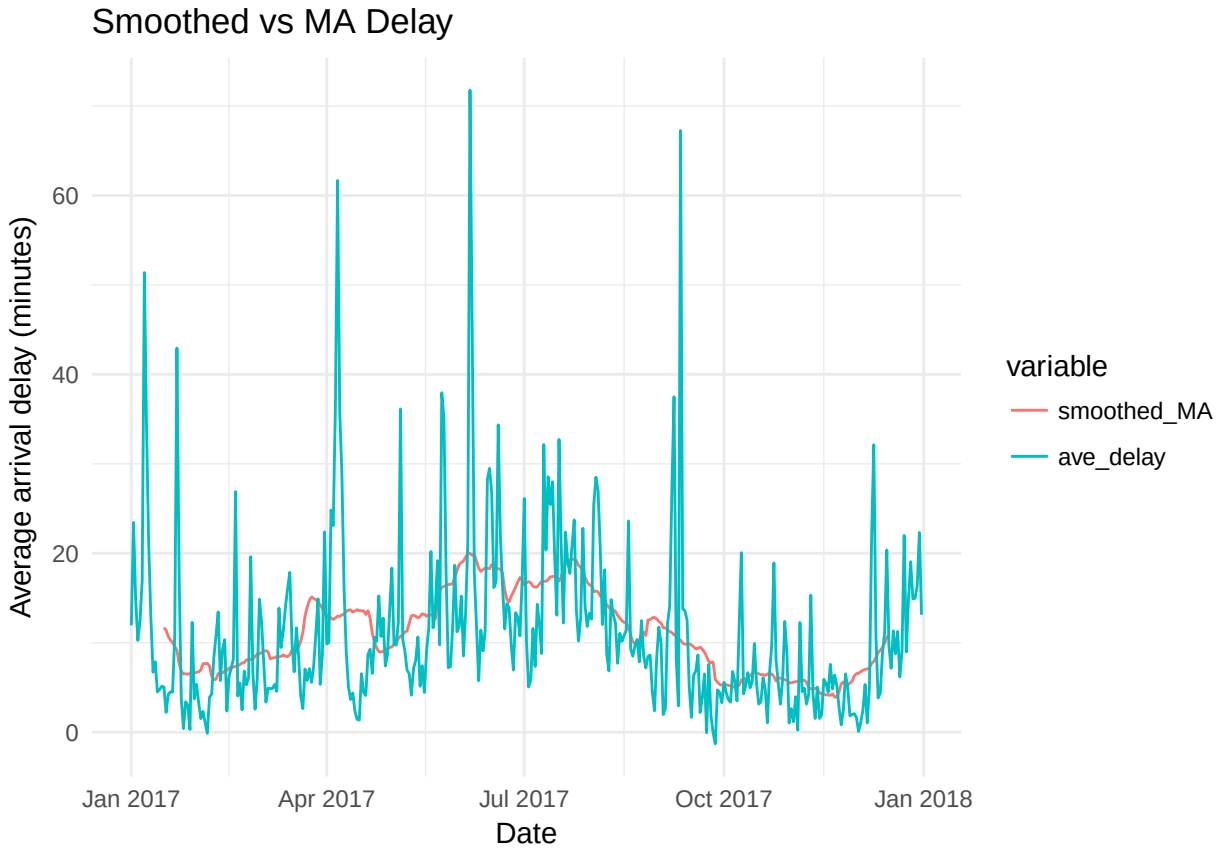

```
# generate dates to make plots more legible
dates <- seq(as.Date('2017-01-01'), as.Date('2017-12-31'), by = 'day') # create sequence of dates
plot.ts <- data.frame(smoothed_MA = moving.average, ave_delay=as.numeric(mean.dept.delay.ts),date=dates)
```

```
# melt into long format
# plot both the original and smoothed data can be easily plotted on the same axes
plot.ts.melted <- melt(plot.ts, id='date')
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
ggplot(plot.ts.melted) + geom_line(aes(x = date, y  = value, col = variable)) + labs(x = 'Date', y = 'A
```

```
## Warning: Removed 30 rows containing missing values (geom_path).
```



Questions

1. How does the neighborhood size, i.e. the number of points in each localized subset, affect the amount of smoothing?

When you increase the window size, you have more smoothing.

2. What happened to endpoints of the smoothed data?
   You lose the first and last endpoints. You cannot intialize a subset of calculating the MA at those points, so they're set to NA.

Advanced: Smooth the same data using Local Regression (loess). Plot smoothed data using base package. Plot all three series (original, smoothed by MA, and smoothed by loess) using ggplot (advanced).

Hint

* loess() librarys all predictors to be numerical so dates cannot be used

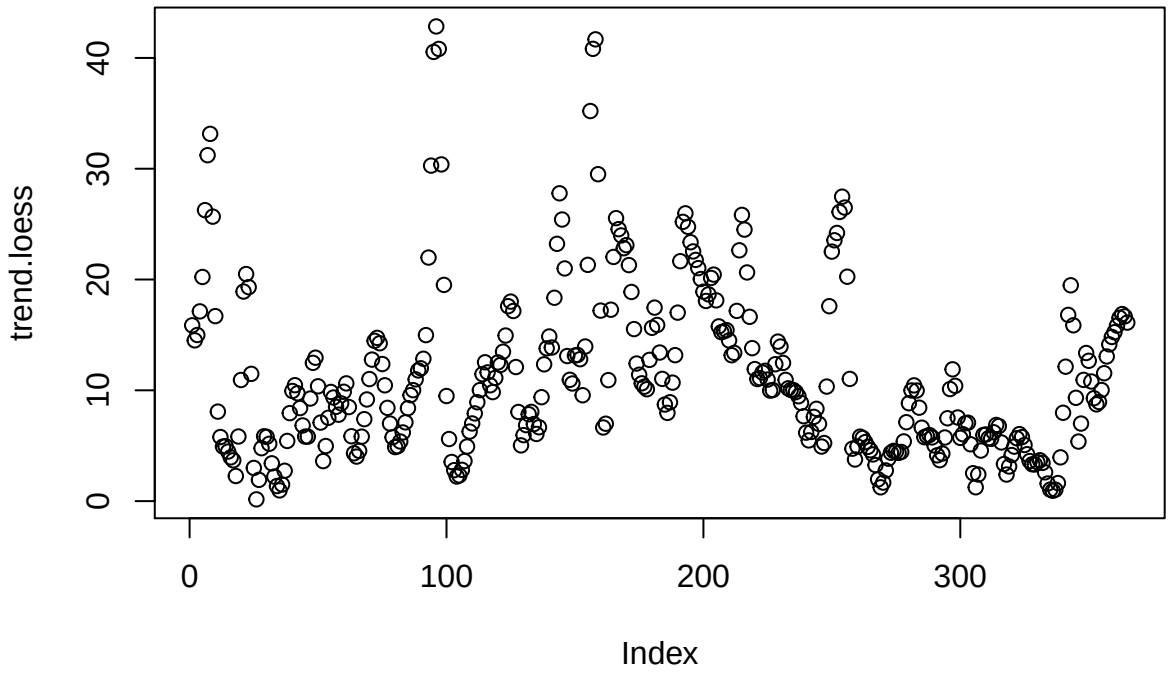
Try different values for the span argument and see how it affects the amount of smoothing.

```
# names(mean.dept.delay) <- c('FL_DATE', 'average.delay')

# lowess.index <- seq(1:nrow(mean.dept.delay))
mean.dept.delay$index <- 1:nrow(mean.dept.delay)

# mean.dept.delay <- cbind(mean.dept.delay, lowess.index)

loess.model <- loess(AVE_DELAY~index, data=mean.dept.delay, span = 1/31)
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
```

```
## parametric, : k-d tree limited by memory. ncmax= 365
```

```
# smooth the data by using the predict() function
trend.loess <- predict(loess.model)

# plot the smoothed data using base package
plot(trend.loess)
```

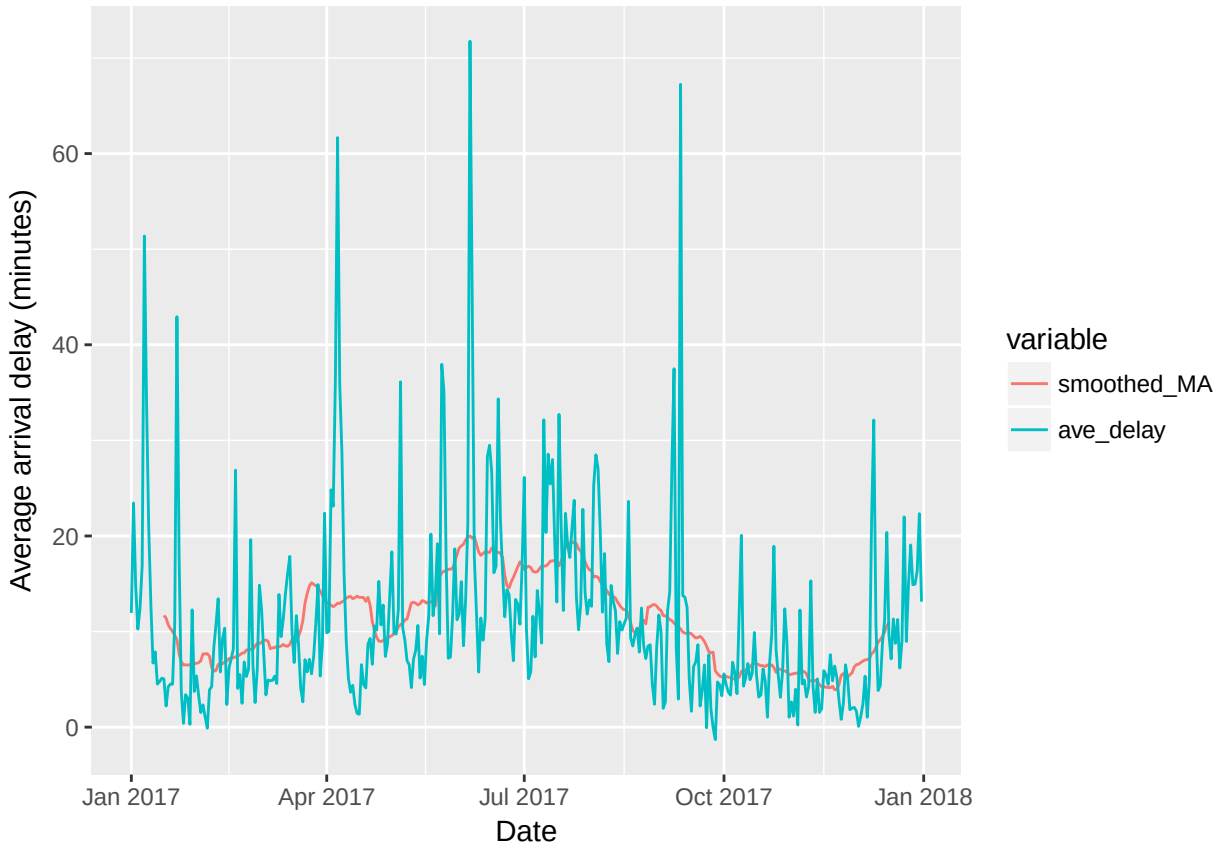



```
# plot using ggplot

# creating data frame with dates to improve readability of plots
plot_ts <- cbind(plot.ts, smoothed.loess = trend.loess)

# melt into long format so both the origininal and smoothed data can be plotted on the same axes
plot.ts.melted <- melt(plot.ts, id='date')
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
ggplot(plot.ts.melted) + geom_line(aes(x = date, y  = value, col = variable)) + labs(x = 'Date', y = 'A
```

```
## Warning: Removed 30 rows containing missing values (geom_path).
```



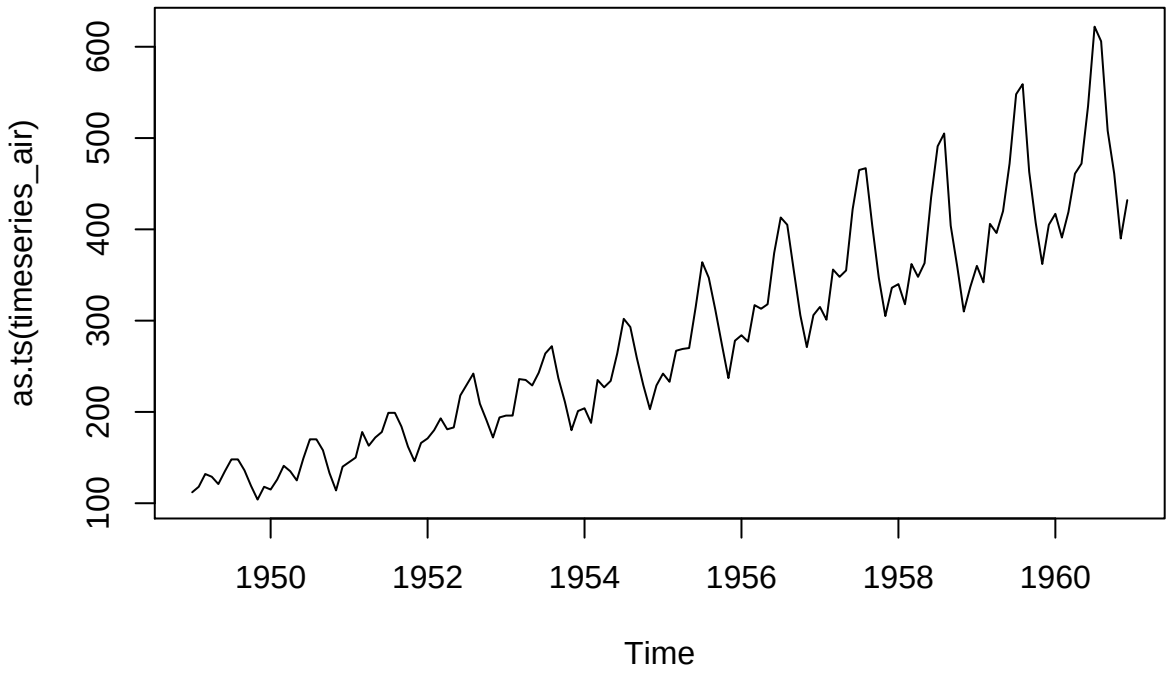
Dive in Deeper to TimeSeries

For this portion of our lab we will be using data from the AirPassengers Dataset

```
data(AirPassengers)
```

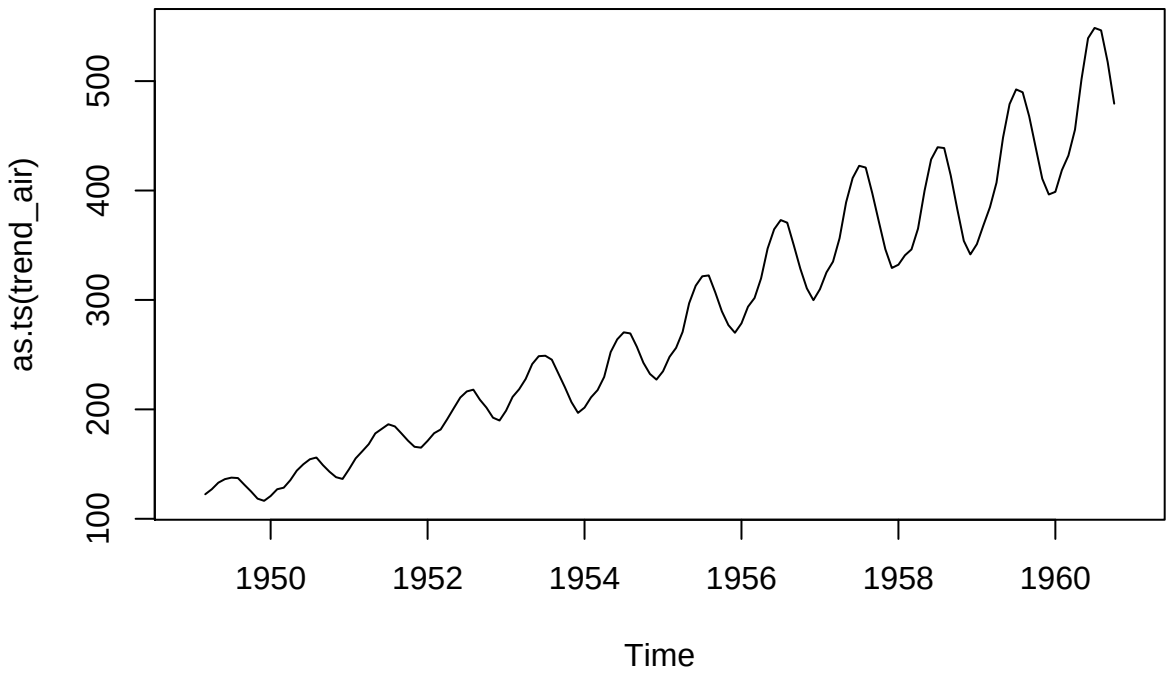
Step 6: Make an inital TimeSeries Visual of the data

```
timeseries_air = AirPassengers
plot(as.ts(timeseries_air))
```



Step 7: Compute the Moving Average of this data using forecast package and vizualize this

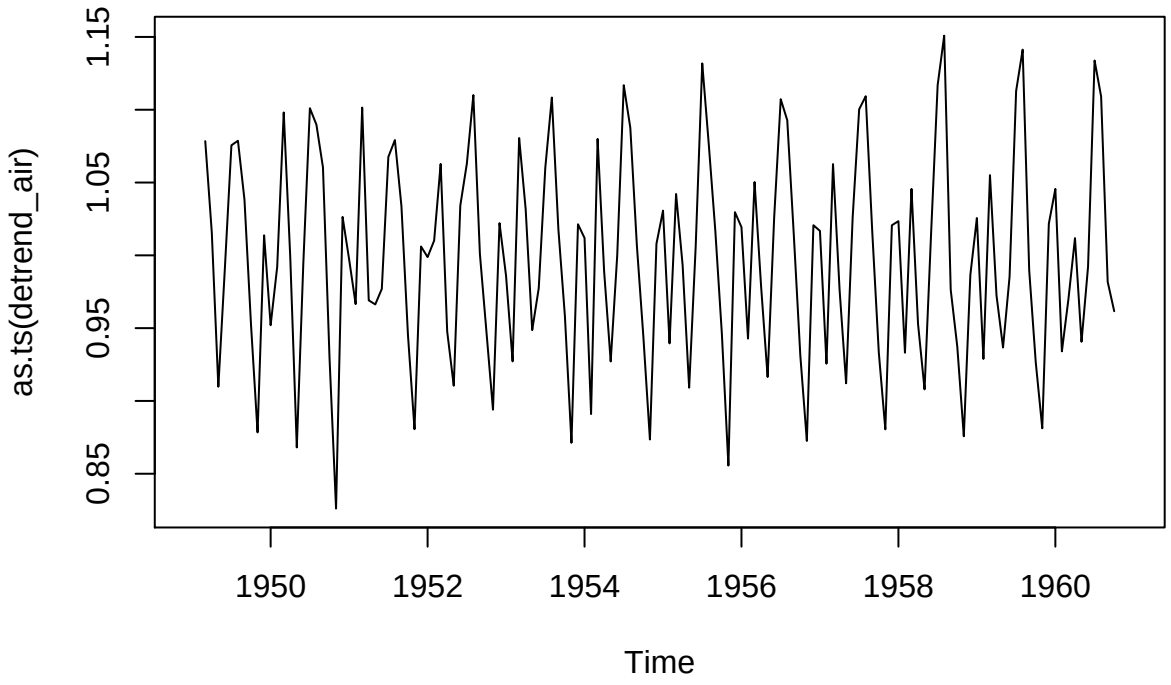
```
trend_air = ma(timeseries_air)
              # , order = 12, centre = T)
#lines(trend_air)
plot(as.ts(trend_air))
```



##

Step 8: Remove the Trend from the data and Visualize this

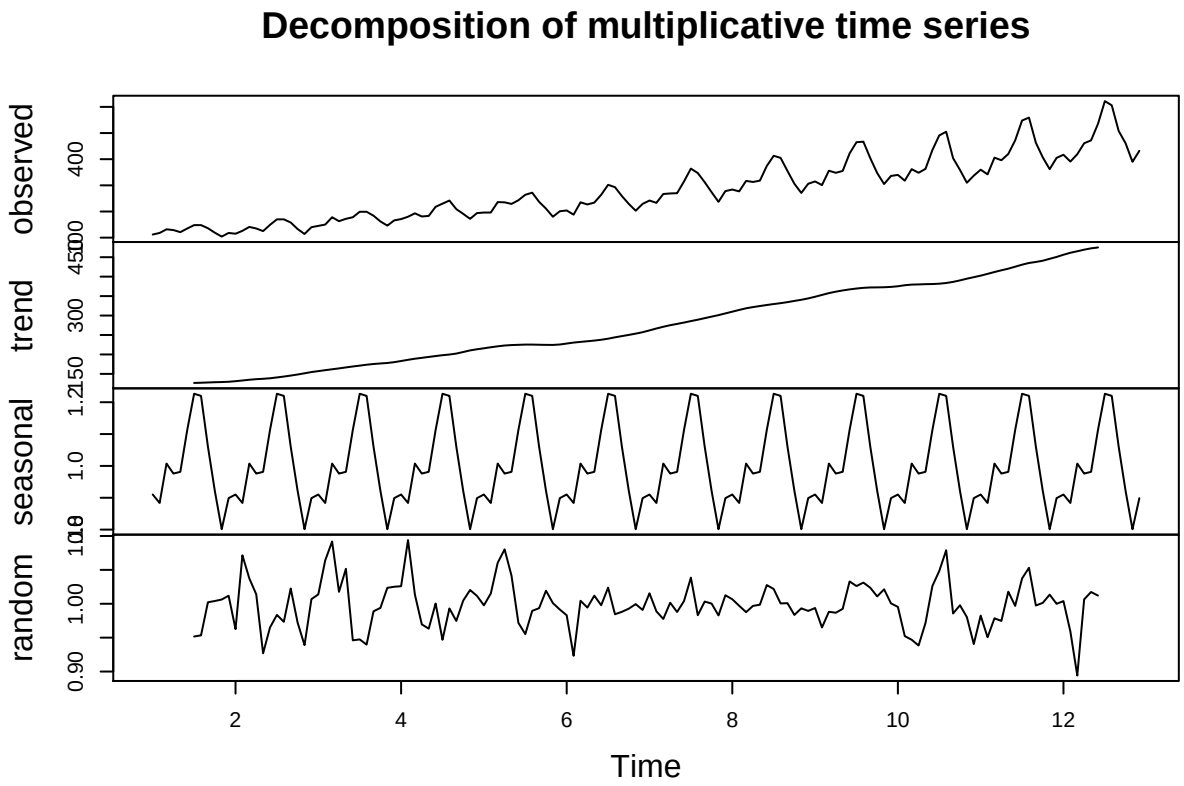
```
detrend_air = timeseries_air / trend_air
plot(as.ts(detrend_air))
```



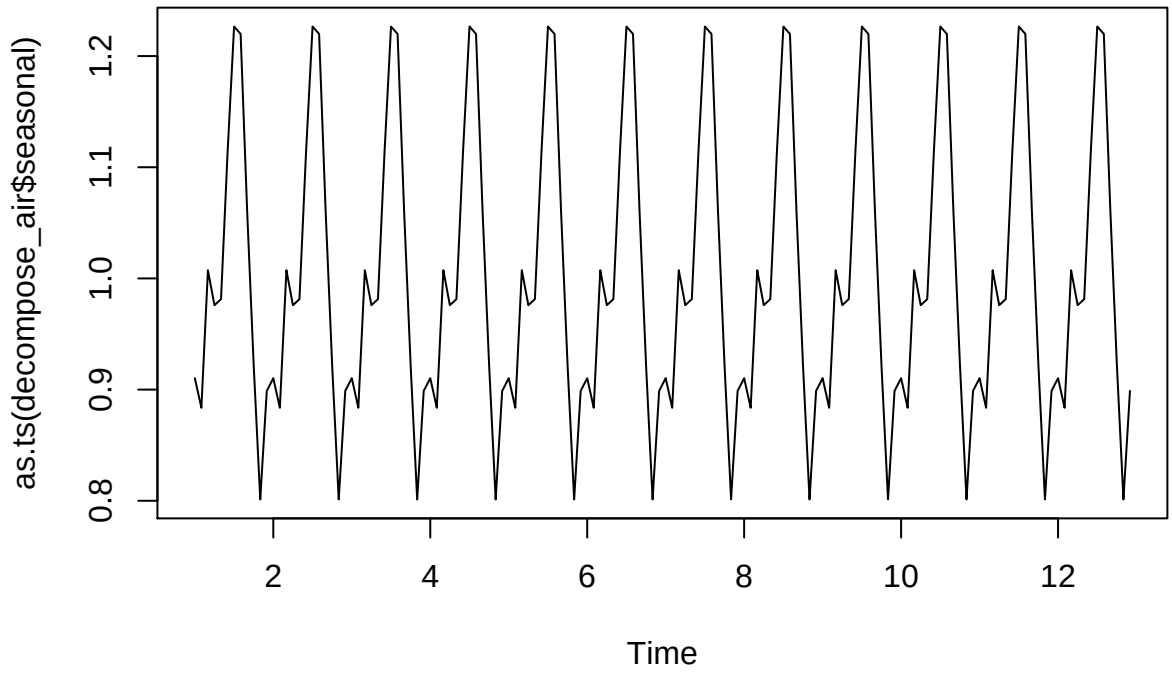
##

Step 9: Create a decomposition of the data by month
– Hint (Frequency = 12)

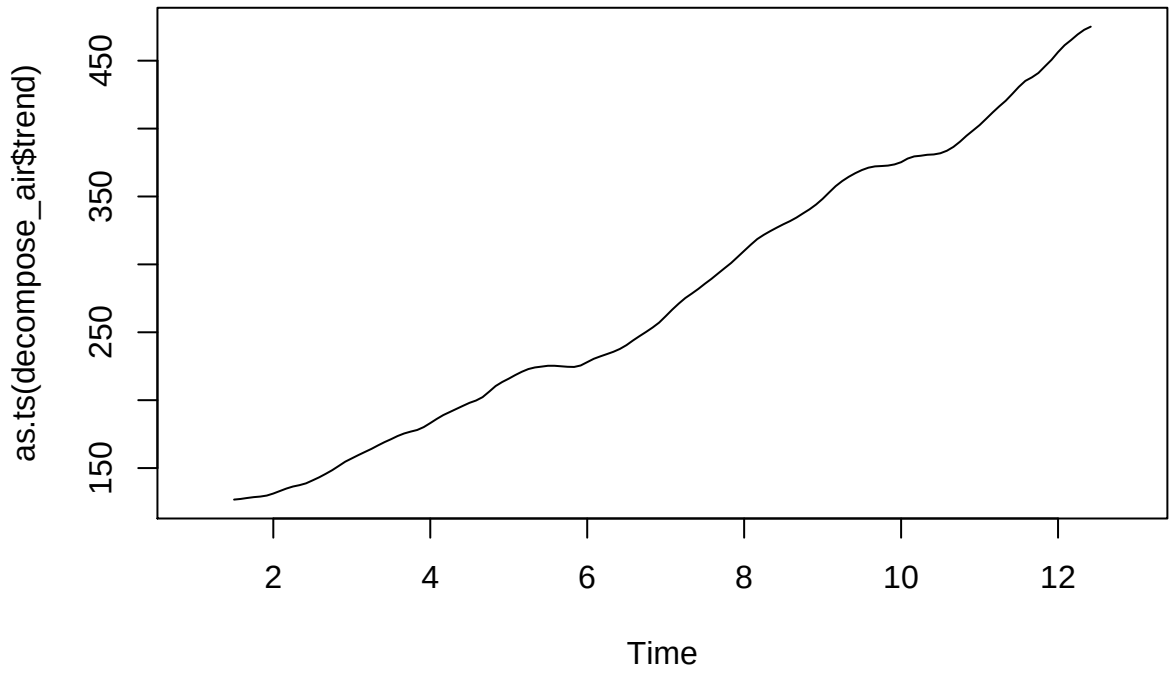
```
ts_air = ts(timeseries_air, frequency = 12)
decompose_air = decompose(ts_air, "multiplicative")

plot(decompose_air)
```




```
### Advanced answers
plot(as.ts(decompose_air$seasonal))
```




```
plot(as.ts(decompose_air$trend))
```




```
plot(as.ts(decompose_air$random))
```