Time Series Lab

Noah Johnson
Michael McCormack

March 15, 2018

Step 1: Get the data.

Download from website.

```
flights <- read.csv('flights.csv.gz')
```

Step 2: Prepare the data.

```
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 ...
```

```
unique(flights$X)
```

```
## [1] NA
```

The X column is just NAs, so we can remove it.

```
flights <- flights %>% select(-c(X))
```

FL_DATE clearly should be a Date object not a factor.

```
flights <- flights %>% mutate(FL_DATE = as.Date(FL_DATE, format = "%Y-%m-%d"))
```

Are there any null values?

```
sapply(flights, function(y) sum(is.na(y)))
```

```
##       DAY_OF_WEEK           FL_DATE    UNIQUE_CARRIER        AIRLINE_ID
##                 0                 0                 0                 0
##           CARRIER         TAIL_NUM            FL_NUM            ORIGIN
##                 0                 0                 0                 0
##              DEST         DEP_TIME         DEP_DELAY         DEP_DEL15
##                 0             19247             19259             19259
##         ARR_DELAY         ARR_DEL15         CANCELLED CANCELLATION_CODE
##             21802             21802                 0                 0
##          DIVERTED          AIR_TIME          DISTANCE
##                 0             21802                 0
```

If departure or arrival delay is NA, does that mean there was just 0 delay?

```
head(flights %>% filter(is.na(ARR_DELAY)))
```

```
##   DAY_OF_WEEK    FL_DATE UNIQUE_CARRIER AIRLINE_ID CARRIER TAIL_NUM FL_NUM
## 1           1 2017-12-25             AA      19805      AA   N3DLAA    134
## 2           1 2017-12-18             AA      19805      AA   N004AA    139
## 3           1 2017-12-25             AA      19805      AA   N820AA   1815
## 4           1 2017-12-18             AA      19805      AA   N558AA   1820
## 5           1 2017-12-18             AA      19805      AA   N025AA    958
## 6           1 2017-12-04             NK      20416      NK   N630NK    179
##   ORIGIN DEST DEP_TIME DEP_DELAY DEP_DEL15 ARR_DELAY ARR_DEL15 CANCELLED
## 1    MIA  SEA     1958        -2         0        NA         NA         0
## 2    ATL  MIA       NA        NA        NA        NA         NA         1
## 3    MIA  SEA      831         1         0        NA         NA         0
## 4    CLT  RSW      724        -6         0        NA         NA         0
## 5    ATL  MIA       NA        NA        NA        NA         NA         1
## 6    ORD  PBI       NA        NA        NA        NA         NA         1
##   CANCELLATION_CODE DIVERTED AIR_TIME DISTANCE
## 1                           1       NA     2724
## 2                 B         0       NA      594
## 3                           1       NA     2724
## 4                           1       NA      600
## 5                 B         0       NA      594
## 6                 B         0       NA     1144
```

It looks like there are rows that have a flight date, and airline carrier, but NAs for delays and departure time. I'm not confident we can assume there was no delay. Maybe it just means the data recorder knew the date of the flight but not how long it took, nor if there were delays. So rather than converting these NAs to zeroes, let's just drop these rows.

```
flights <- flights %>% drop_na()
```

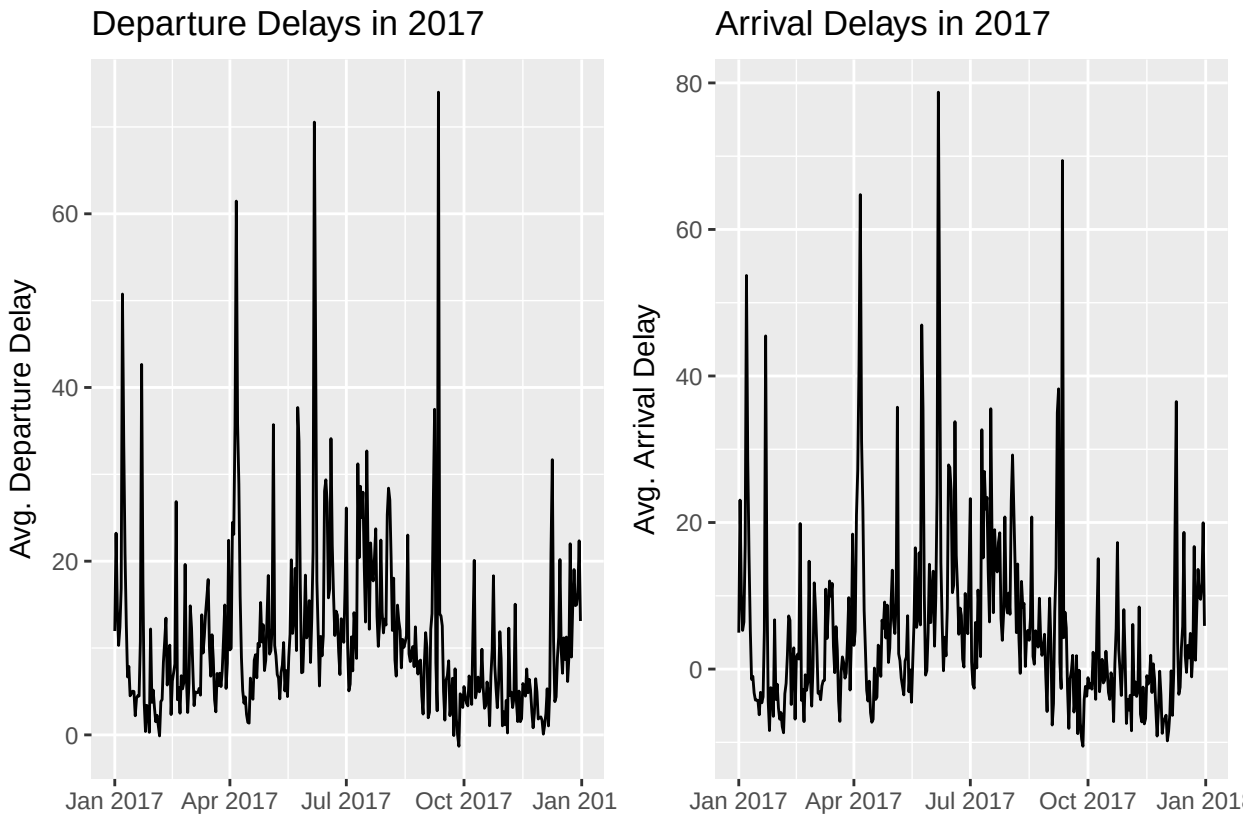
Time series data needs to be over a regular time interval. Calculate the average departure delay time and/or average arrival delay time for each day of 2017.

```
delays <- flights %>% group_by(FL_DATE) %>%
  summarise(avgDepDelay=mean(DEP_DELAY), avgArrivDelay=mean(ARR_DELAY))

departure.plot <- delays %>% ggplot(aes(FL_DATE, avgDepDelay)) +
  geom_line() +
  labs(title = 'Departure Delays in 2017', x = '', y = 'Avg. Departure Delay')

arrival.plot <- delays %>% ggplot(aes(FL_DATE, avgArrivDelay)) +
  geom_line() +
  labs(title = 'Arrival Delays in 2017', x = '', y = 'Avg. Arrival Delay')
```

```
grid.arrange(departure.plot, arrival.plot, ncol=2)
```



Compare average delay times for different carriers or different airports by creating multiple time series.

Let's look at Spirit Airlines (cheap) vs Virgin America (expensive)

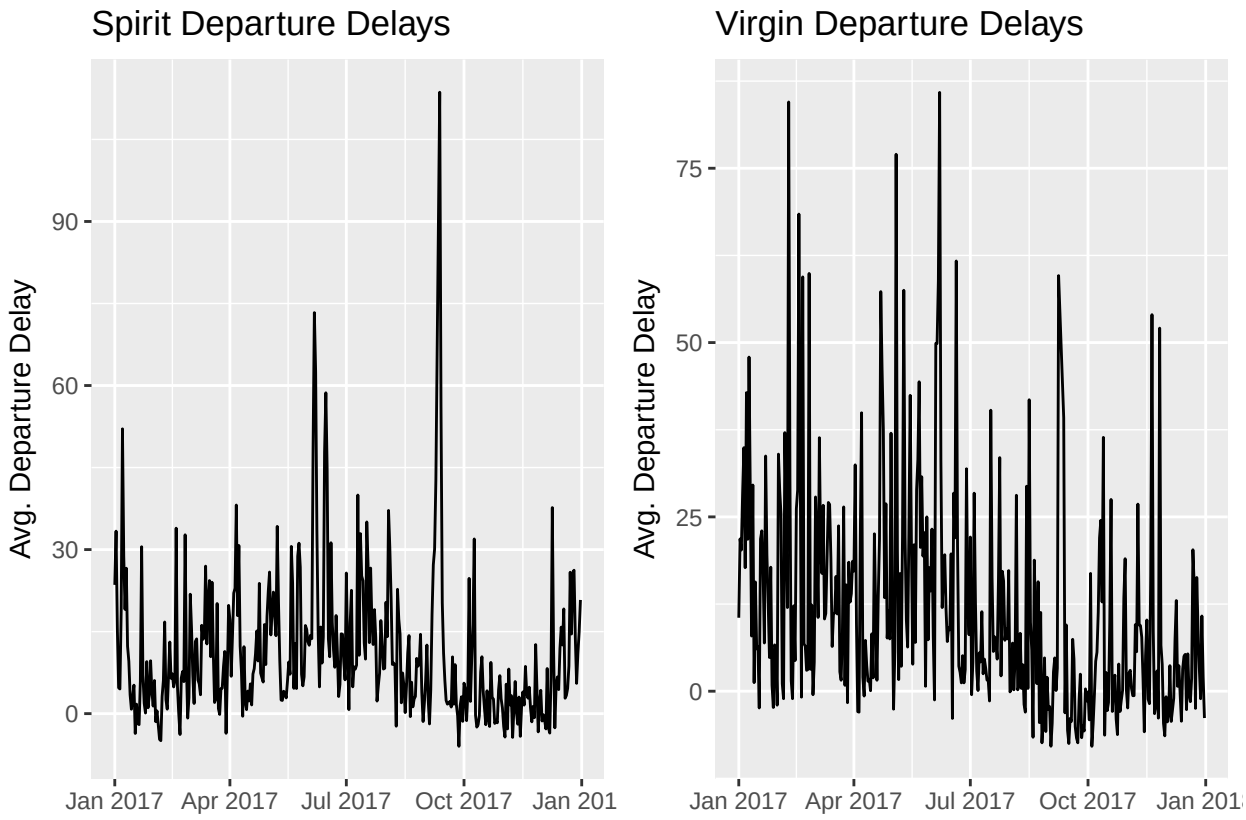
```
spirit.delays <- flights %>% filter(CARRIER=='NK') %>% group_by(FL_DATE) %>%
  summarise(avgDepDelay=mean(DEP_DELAY))

virgin.delays <- flights %>% filter(CARRIER=='VX') %>% group_by(FL_DATE) %>%
  summarise(avgDepDelay=mean(DEP_DELAY))

spirit.plot <- spirit.delays %>% ggplot(aes(FL_DATE, avgDepDelay)) +
  geom_line() +
  labs(title = 'Spirit Departure Delays', x = '', y = 'Avg. Departure Delay')

virgin.plot <- virgin.delays %>% ggplot(aes(FL_DATE, avgDepDelay)) +
  geom_line() +
  labs(title = 'Virgin Departure Delays', x = '', y = 'Avg. Departure Delay')

grid.arrange(spirit.plot, virgin.plot, ncol=2)
```



Step 3: Create a ts object of the data.

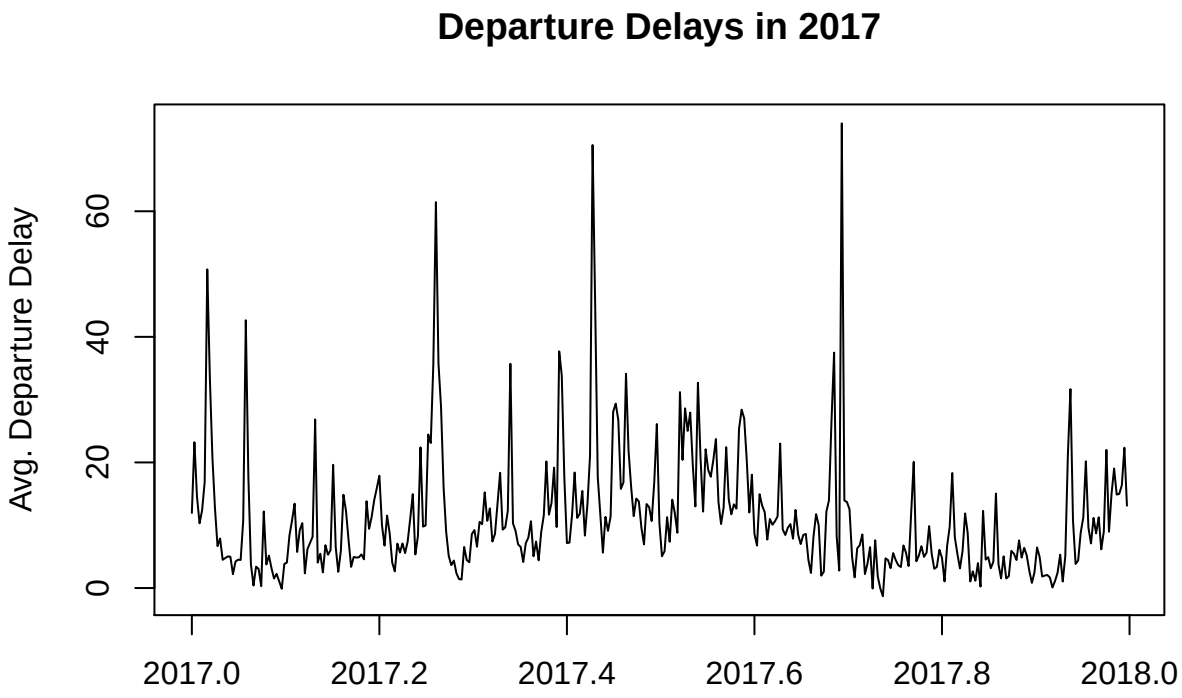
We're going to choose to look at a time series of the average departure delays per day.

```
delays.ts <- ts(delays$avgDepDelay, start = c(2017,1), frequency=365)
```

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

We already plotted departure delays in ggplot earlier. Here is the same plot, using base R.

```
plot(delays.ts, main = 'Departure Delays in 2017', xlab = '', ylab = 'Avg. Departure Delay')
```



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

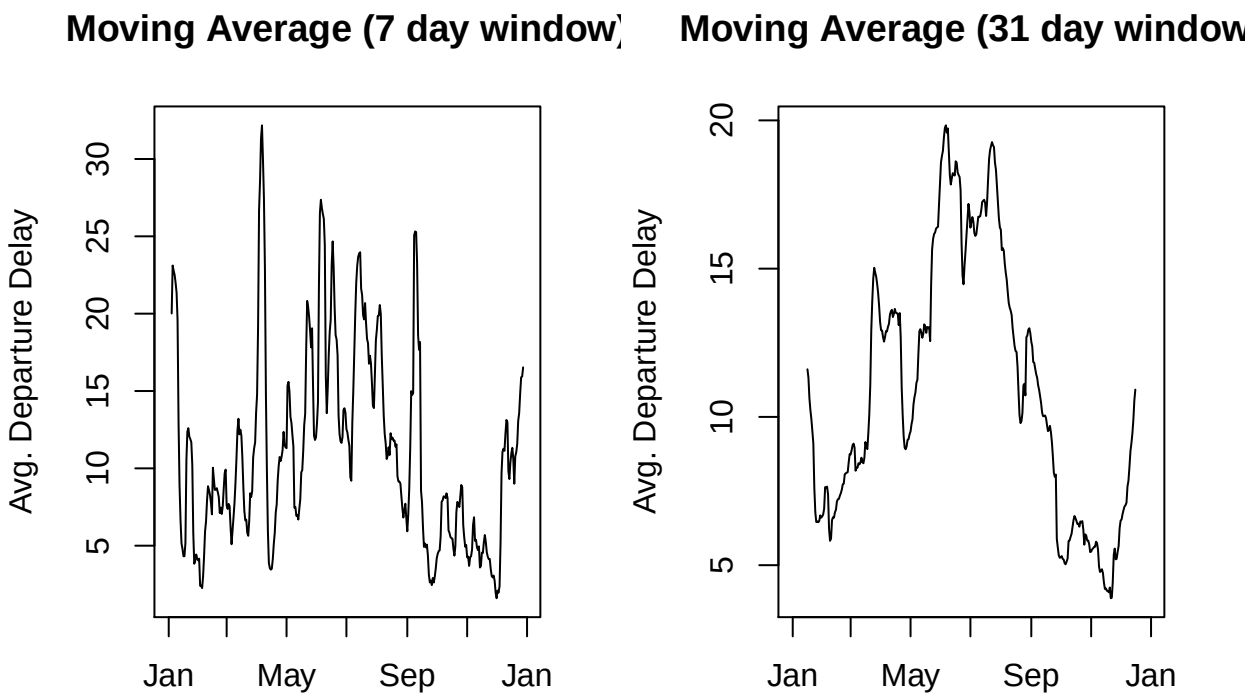
We can use the *ma* function from the forecast package to compute a simple moving average.

```
sm.7 <- ma(delays.ts, order = 7) # average a week
sm.31 <- ma(delays.ts, order = 31) # average a month

par(mfrow=c(1,2))

plot(delays$FL_DATE, sm.7, type='l', main = 'Moving Average (7 day window)',
     xlab = '', ylab = 'Avg. Departure Delay')

plot(delays$FL_DATE, sm.31, type='l', main = 'Moving Average (31 day window)',
     xlab = '', ylab = 'Avg. Departure Delay')
```



Looks like there's an increase in average departure delays over the summer months.

Create your own simple moving average for monthly data. Plot both the original and the smoothed data with 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 multiple series

```
# Size of the sliding window used in this simple moving average filter
window_size <- 31

# The cumulative delay values
cum_delays <- c(0, cumsum(delays.ts))

manual.sm <- (cum_delays[(window_size+1):length(cum_delays)] -
                cum_delays[1:(length(cum_delays) - window_size)]) / window_size

# pad with NAs on the edges
manual.sm <- c(rep(NA, (window_size-1)/2), manual.sm, rep(NA, (window_size-1)/2))

# Check if our manually computed smoothed data is the same as the output from ma()
sum(manual.sm - sm.31, na.rm=T)
```

```
## [1] 1.203482e-13
```

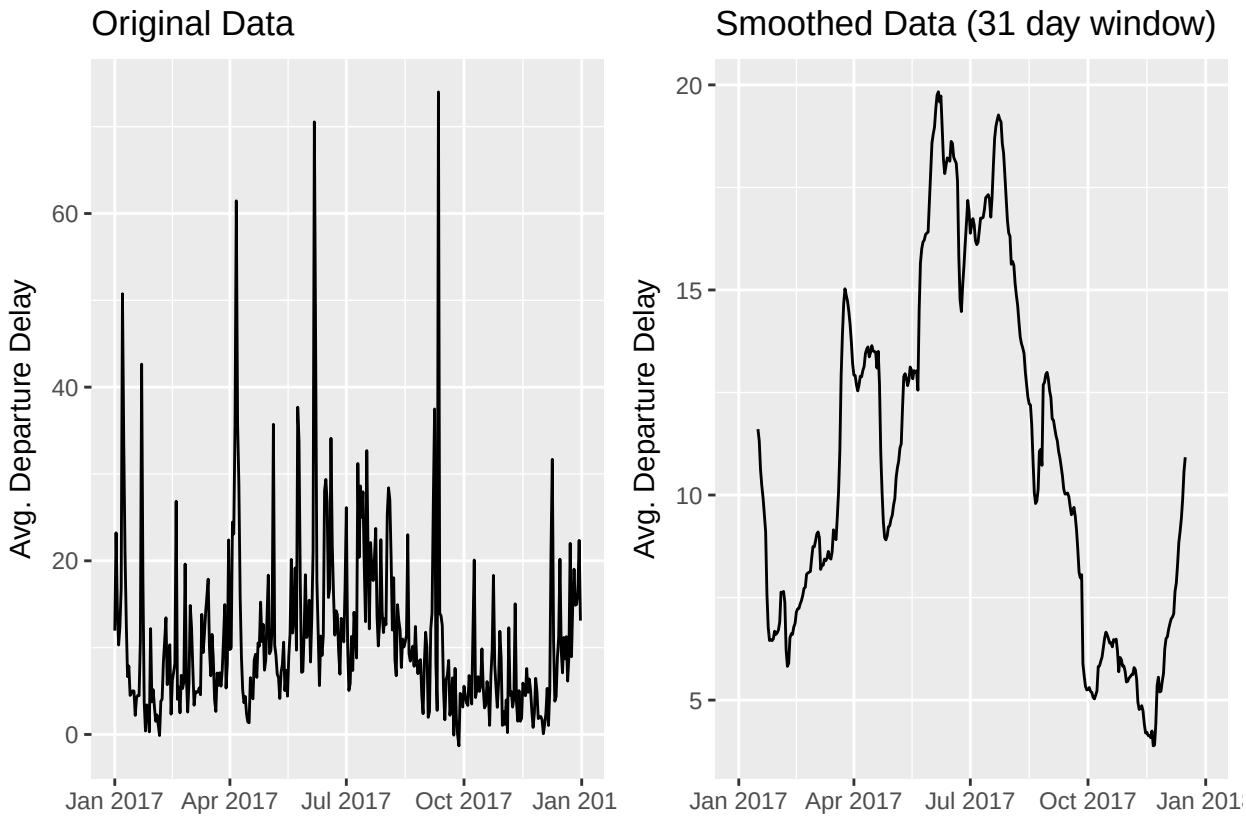
Success! Now we'll plot the original data next to the filter output using ggplot.

```
delays$sm <- manual.sm

ma.plot <- delays %>% ggplot(aes(FL_DATE, sm)) +
  geom_line() +
  labs(title='Smoothed Data (31 day window)', x='', y='Avg. Departure Delay')

grid.arrange(departure.plot + ggtitle('Original Data'), ma.plot, ncol=2)
```

```
## 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?

A larger neighborhood size leads to more smoothing.

2. What happened to endpoints of the smoothed data?

Endpoints are chopped off, since the filter cannot estimate them (not enough data around each point).

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).

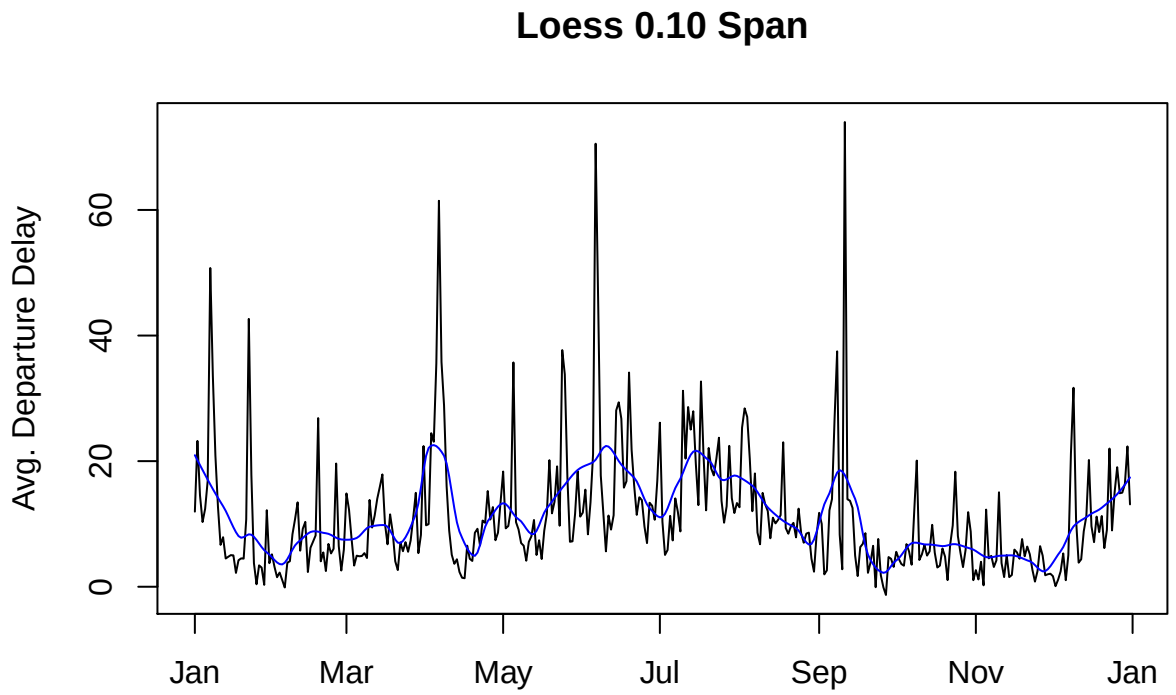
Hint

- loess() requires 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.

The span argument controls the sliding window size. A larger span parameter means a larger window is averaged, giving a smoother result.

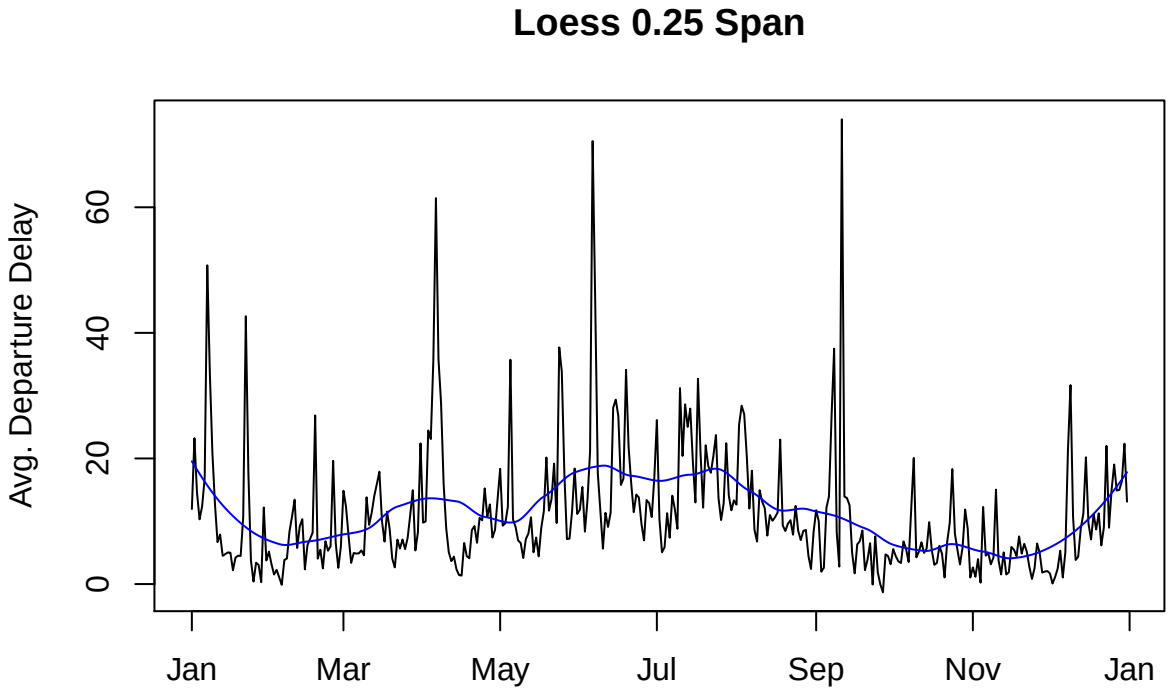
```
loess10.delays <- loess(avgDepDelay ~ c(1:nrow(delays)), span=0.1, data=delays)
loess10.predictions <- predict(loess10.delays)

plot(delays$FL_DATE, delays$avgDepDelay, type = 'l', main = 'Loess 0.10 Span',
     xlab = '', ylab = 'Avg. Departure Delay')
lines(loess10.predictions, x=delays$FL_DATE, col='blue')
```



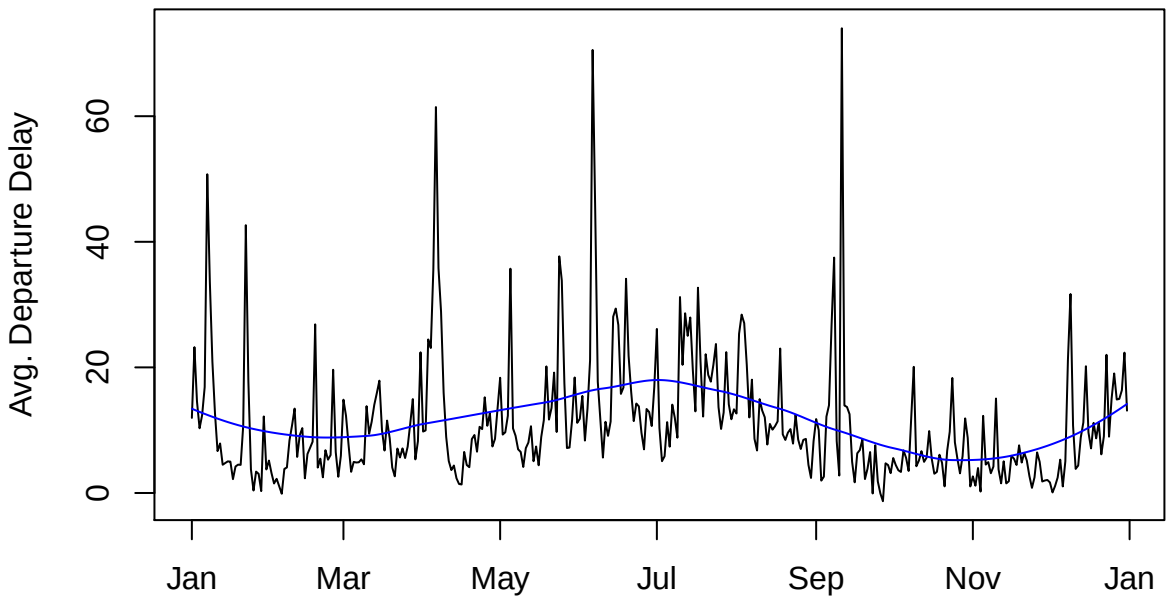

```
loess25.delays <- loess(avgDepDelay ~ c(1:nrow(delays)), span=0.25, data=delays)
loess25.predictions <- predict(loess25.delays)

plot(delays$FL_DATE, delays$avgDepDelay, type = 'l', main = 'Loess 0.25 Span',
     xlab = '', ylab = 'Avg. Departure Delay')
lines(loess25.predictions, x=delays$FL_DATE, col='blue')
```



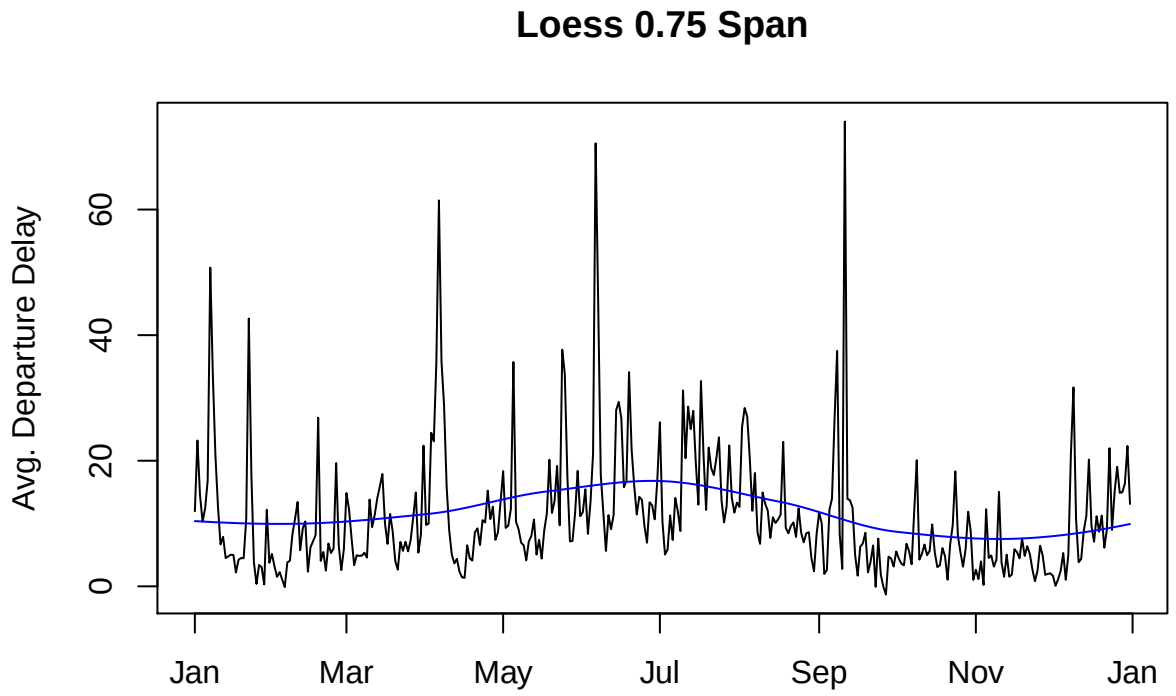

```
loess50.delays <- loess(avgDepDelay ~ c(1:nrow(delays)), span=0.5, data=delays)
loess50.predictions <- predict(loess50.delays)

plot(delays$FL_DATE, delays$avgDepDelay, type = 'l', main = 'Loess 0.50 Span',
     xlab = '', ylab = 'Avg. Departure Delay')
lines(loess50.predictions, x=delays$FL_DATE, col='blue')
```




```
loess75.delays <- loess(avgDepDelay ~ c(1:nrow(delays)), span=0.75, data=delays)
loess75.predictions <- predict(loess75.delays)
```

```
plot(delays$FL_DATE, delays$avgDepDelay, type = 'l', main = 'Loess 0.75 Span',
     xlab = '', ylab = 'Avg. Departure Delay')
lines(loess75.predictions, x=delays$FL_DATE, col='blue')
```



We can see that the loess output is still very noisy with a span of 0.1, as expected, and smooths out as the span is increased. The output doesn't change much for spans greater than 0.5, so we'll use the 0.5 curve as our loess trend estimate.

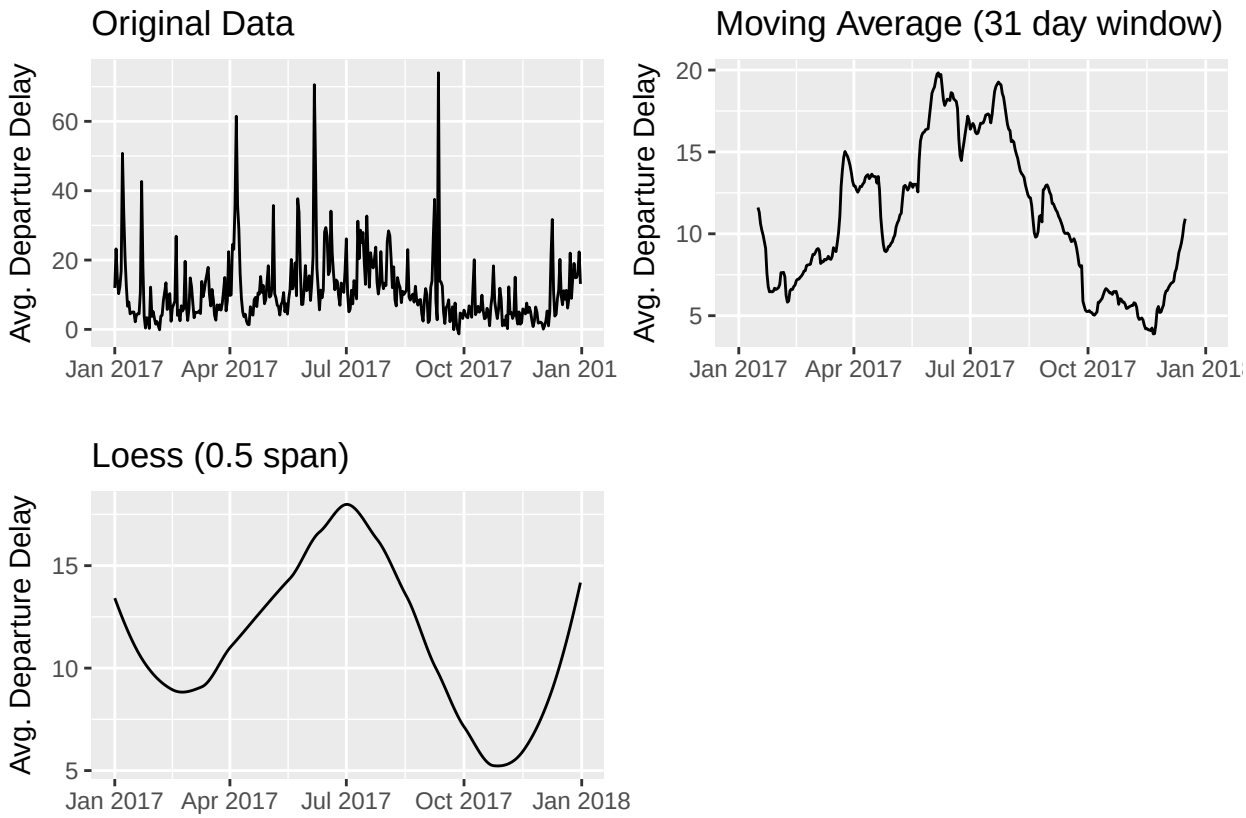
Now let's plot all three series using ggplot:

```
delays$sm.loess <- loess50.predictions

loess.plot <- delays %>% ggplot(aes(FL_DATE, sm.loess)) +
  geom_line() +
  labs(title='Loess (0.5 span)', x='', y='Avg. Departure Delay')

grid.arrange(departure.plot + ggtitle('Original Data'),
             ma.plot + ggtitle('Moving Average (31 day window)'),
             loess.plot, ncol=2)
```

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



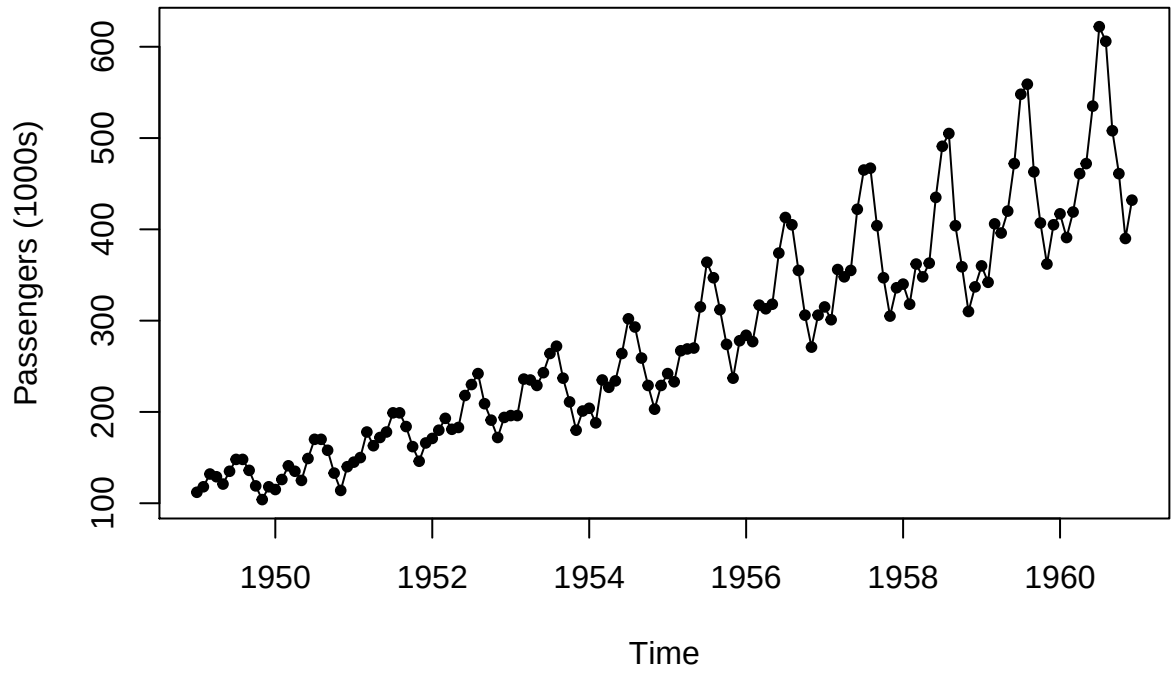
Dive in Deeper to TimeSeries

For this portion of our lab we will be using data from the AirPassengers Dataset. This classic example dataset lists monthly totals of international airline passengers in thousands from 1949 to 1960.

```
data(AirPassengers)
```

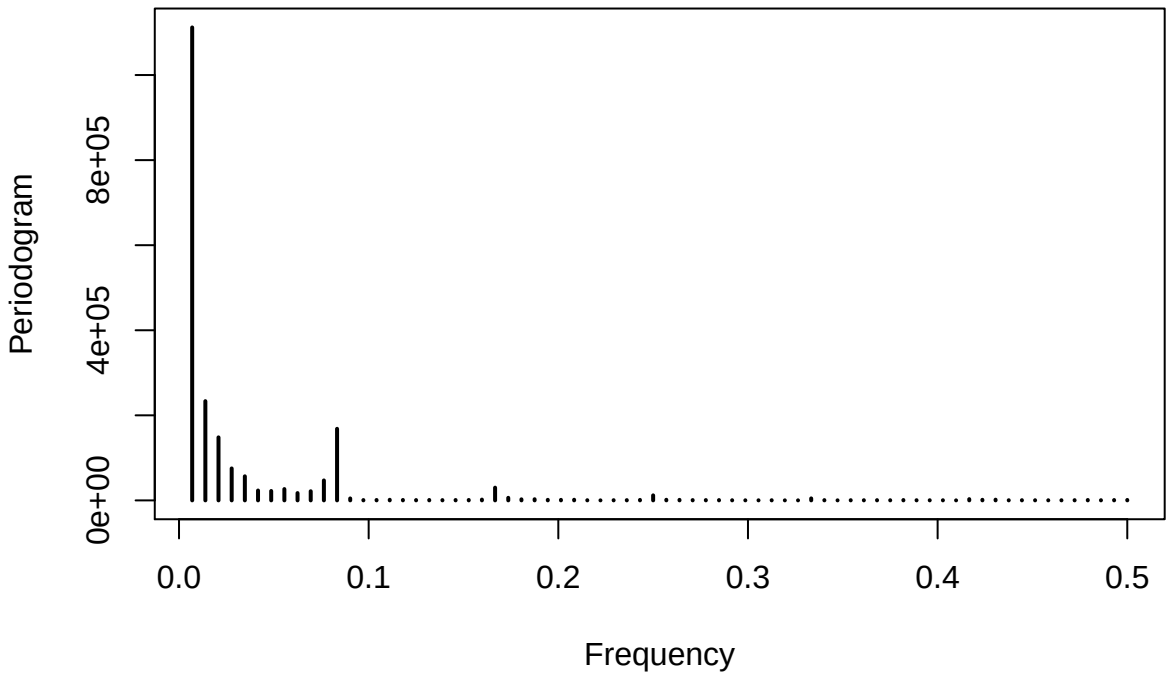
Step 6: Make an inital TimeSeries Visual of the data

```
plot(AirPassengers, type="o", pch=20, ylab='Passengers (1000s)')
```



We can see that this time series is multiplicative, as the seasonal swings get larger and larger as the years go by. But exactly how long is one season? We can use the Fourier Transform to map our signal into the frequency domain, and figure this out.

```
p <- periodogram(AirPassengers)
```



Looks like there is a large spike at a small frequency very close to zero. We can calculate the top frequencies and periods.

```
topF <- data.frame(freq=p$freq, spec=p$spec) %>% top_n(5, spec) %>% arrange(desc(spec))
topF
```

```
##          freq       spec
## 1 0.006944444 1111915.05
## 2 0.013888889  233355.97
## 3 0.083333333  168356.50
## 4 0.020833333  148241.33
## 5 0.027777778   75210.71
```

```
periods <- 1 / topF$freq
periods
```

```
## [1] 144  72  12  48  36
```

Ok, great! But what unit of time is a period associated with? That depends on how frequently our data was sampled per year.

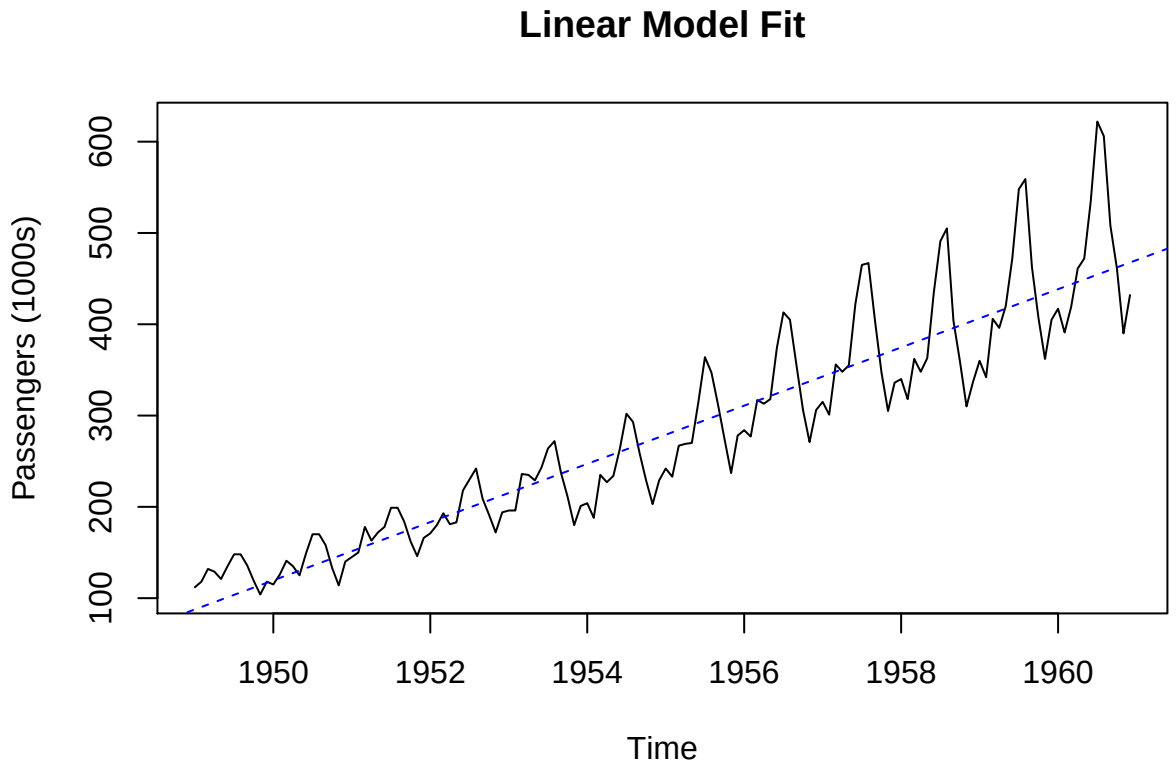
```
frequency(AirPassengers)
```

```
## [1] 12
```

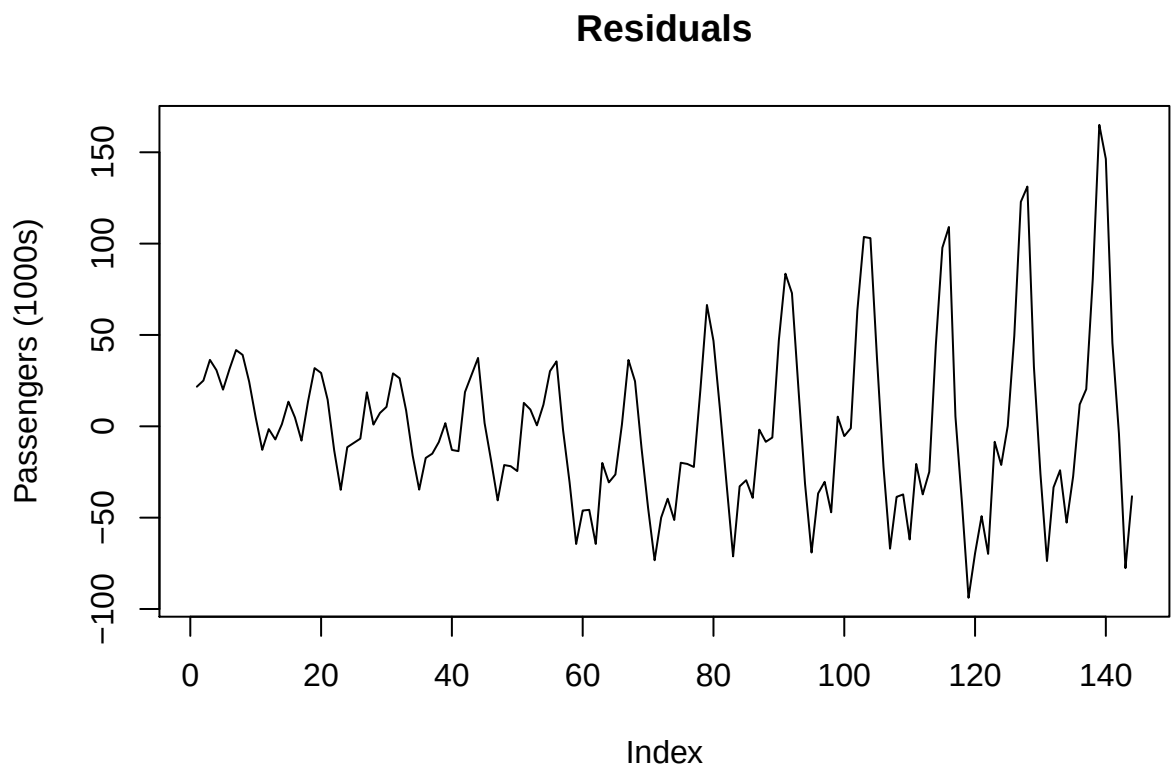
Ok, so our airline passenger time series has 12 samples per year, i.e. it is monthly. So the periods correspond to months, and the top three periods detected are 12, 6, and 1 year. But wait, 12 years covers all of our data! And a 6-year season doesn't match what we saw visually in the plot. An annual pattern would make more sense. It turns out this occurs because of the general upwards trend in the data which is obscuring the smaller seasonal variation. This trend appears linear to our eye, so if we fit a linear model to our data and subtract the estimate from our signal, we should be able to recompute the periodogram with better results.

```
trend <- lm(AirPassengers ~ c(1:length(AirPassengers)))

plot(AirPassengers, main='Linear Model Fit', ylab='Passengers (1000s)')
abline(trend$coefficients[[1]] - trend$coefficients[[2]]*12*1949,
       trend$coefficients[[2]]*12, lty='dashed', col='blue')
```

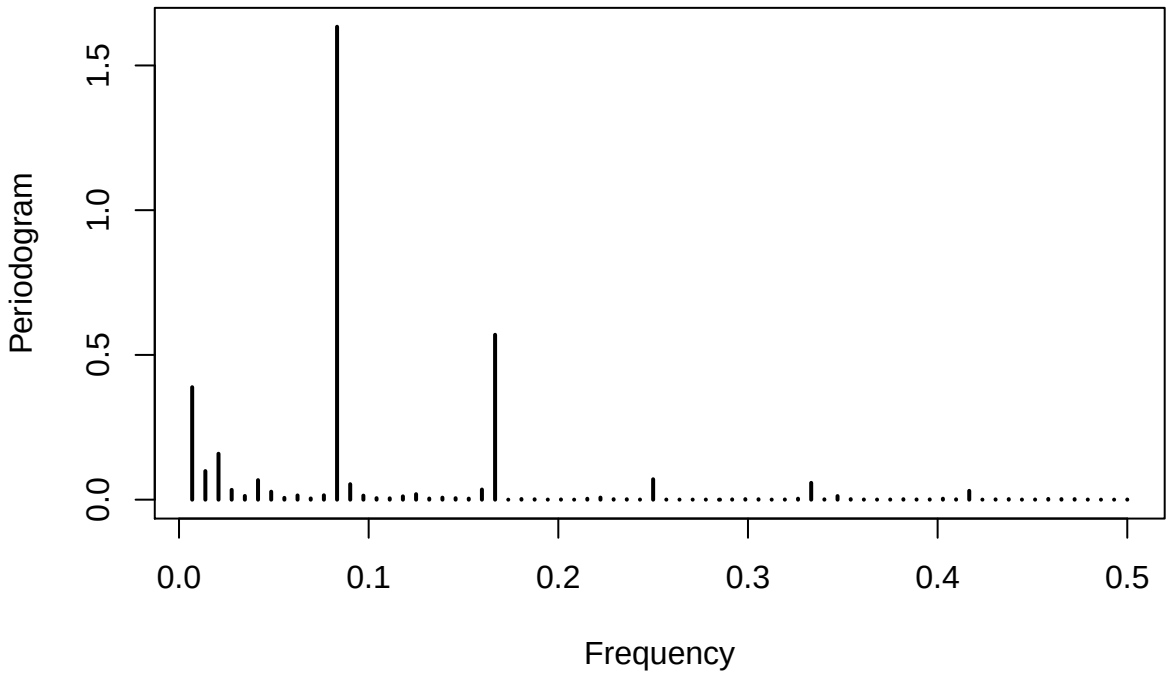


```
plot(resid(trend), type="l", main='Residuals', ylab='Passengers (1000s)')
```



Since we're modelling our time series as a multiplicative case, we remove the trend by dividing by the fit values.

```
p <- periodogram(AirPassengers / trend$fitted.values)
```




```
topF <- data.frame(freq=p$freq, spec=p$spec) %>% top_n(5, spec) %>% arrange(desc(spec))
topF
```

```
##          freq       spec
## 1 0.083333333 1.63358676
## 2 0.166666667 0.56952422
## 3 0.006944444 0.38841984
## 4 0.020833333 0.15859037
## 5 0.013888889 0.09876847
```

```
periods <- 1 / topF$freq
periods
```

```
## [1]  12   6 144  48  72
```

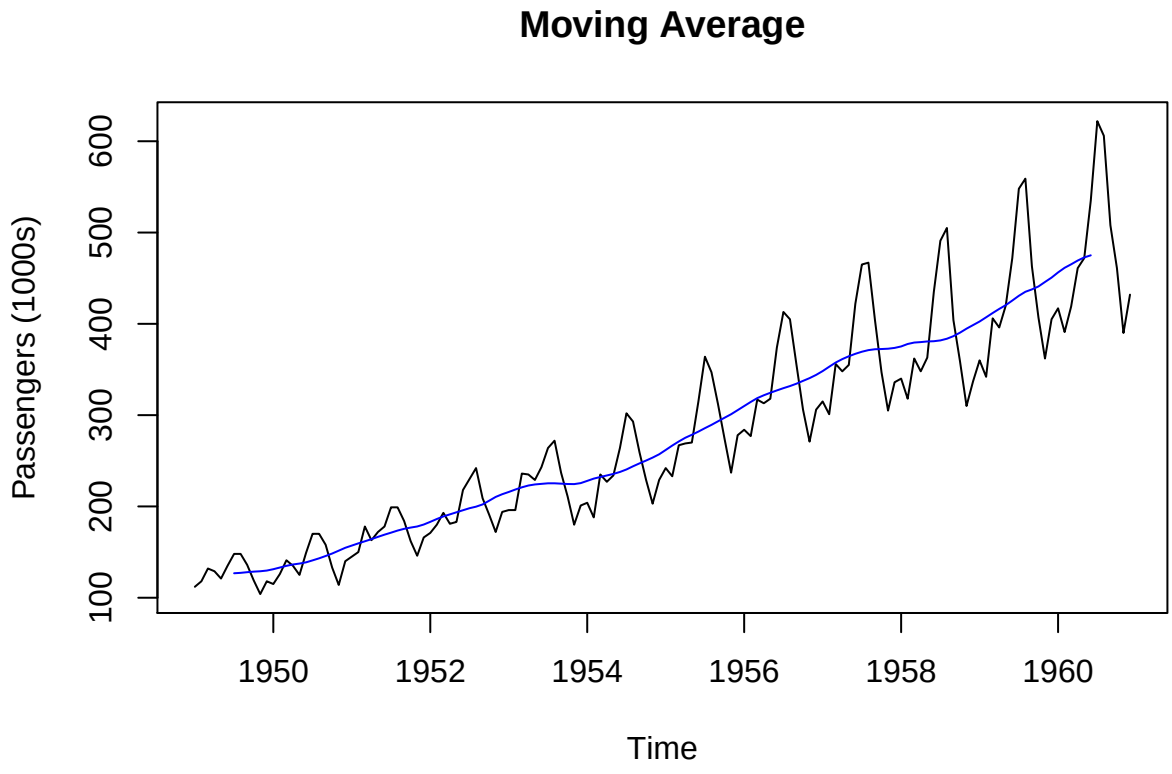
Great, now the main seasonality detected is clearly 12 months. This also fits well with what we see graphically. We can use this knowledge in the next step.

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

The previous linear model was a rough fit. It was adequate to uncover the annual pattern in the data. Now we can get another estimate of the underlying trend in the data using a centered moving average, with a sliding window size exactly equal to the seasonality.

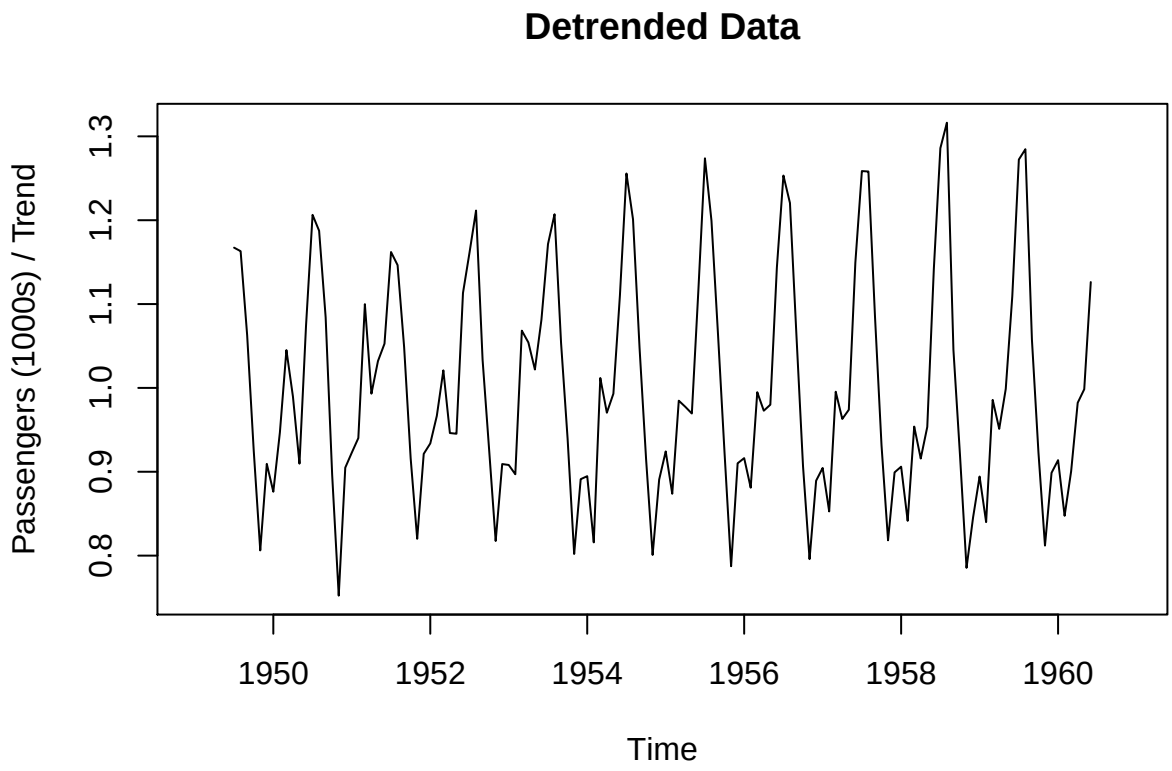
```
# compute moving average with a window of 12 (the seasonal trend is annual)
trend <- ma(AirPassengers, order = 12, centre = TRUE)

plot(AirPassengers, main='Moving Average', ylab='Passengers (1000s)')
lines(trend, col='blue')
```



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

```
AirPassengers.detrend <- AirPassengers / trend
plot(AirPassengers.detrend, main='Detrended Data', ylab='Passengers (1000s) / Trend')
```

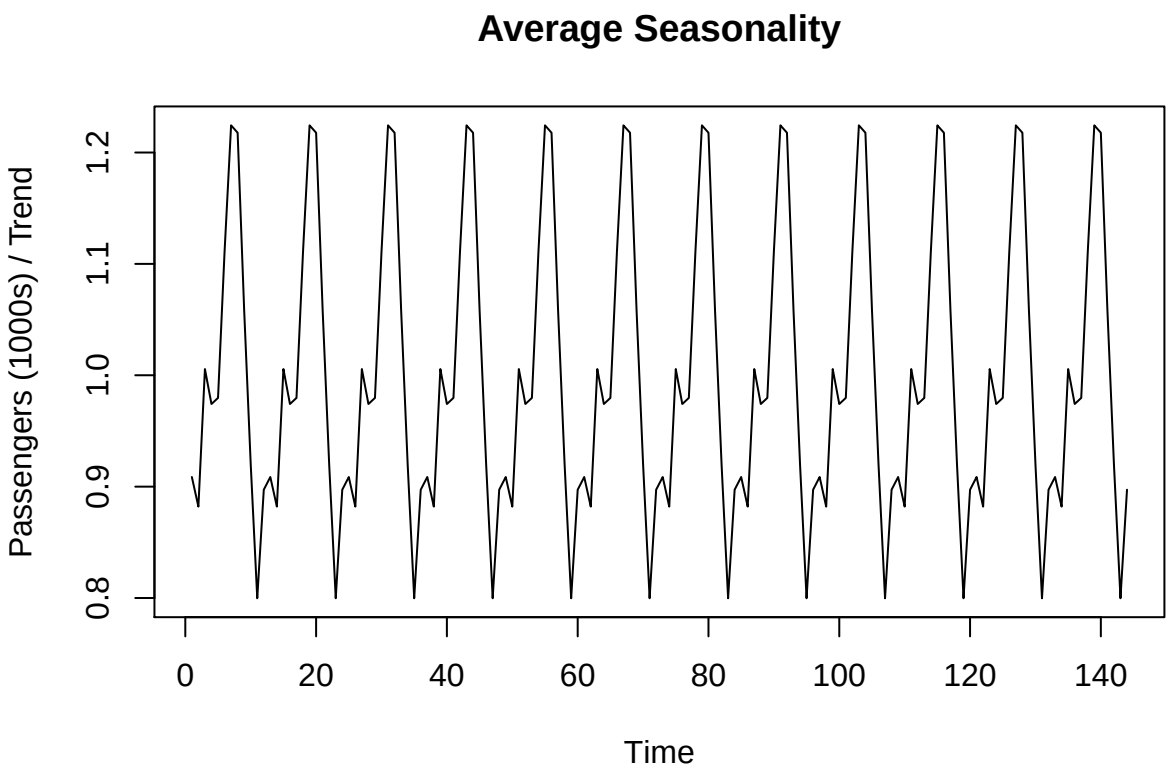


Step 9: Create a decomposition of the data by month

We've already decomposed the data into a trend and remainder using the moving average filter. To complete the decomposition we will decompose this remainder into seasonal and random components.

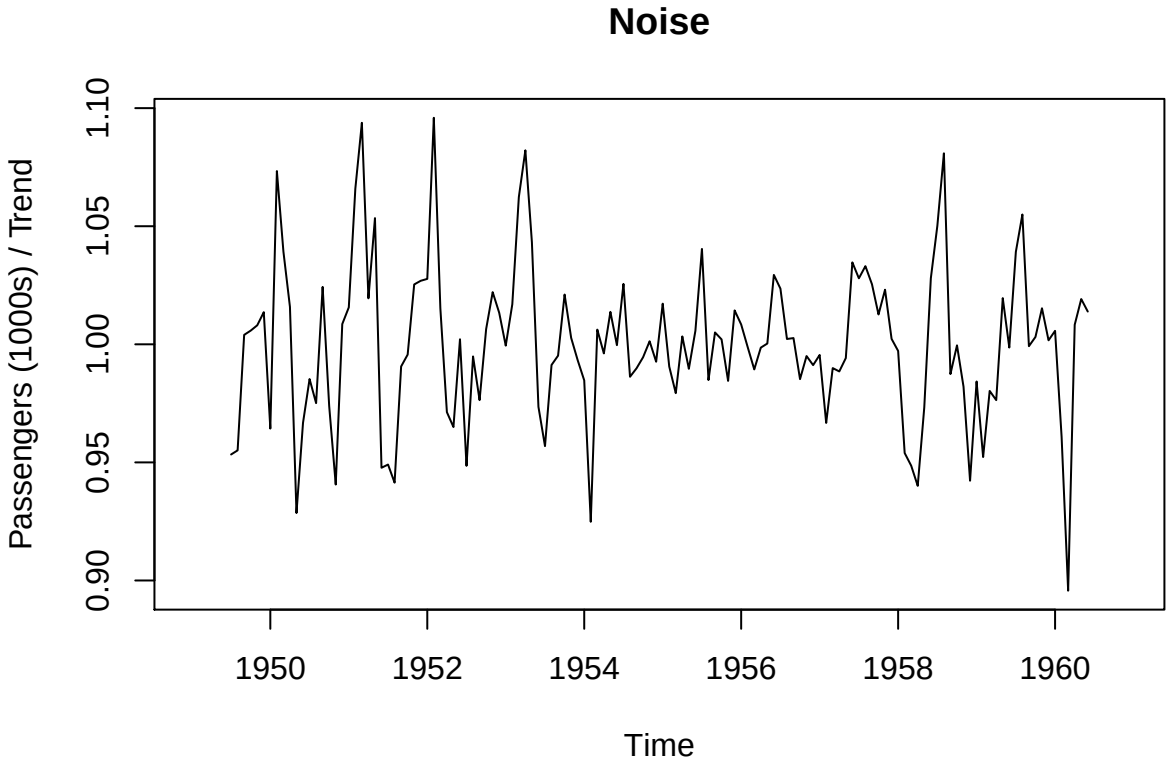
First, to find the seasonal component, we average all the observed seasonal swings. Remembering that the seasonal window is 12 months, we put our data into a matrix with twelve columns, one per month, and take the average of each column. This average seasonal variation is taken as the "true" variation every year due to the month, which gets multiplied by the overall trend.

```
m <- t(matrix(data=AirPassengers.detrend, nrow=12))
avgSeasonality <- colMeans(m, na.rm=TRUE)
plot(as.ts(rep(avgSeasonality, 12)), main='Average Seasonality', ylab='Passengers (1000s) / Trend')
```



Lastly, there is the noise. By the multiplicative model, $Passengers = Trend * Seasonal * Noise$. So $Noise = \frac{Passengers}{Trend * Seasonal}$.

```
noise <- AirPassengers / (trend * avgSeasonality)
plot(noise, main='Noise', ylab='Passengers (1000s) / Trend')
```

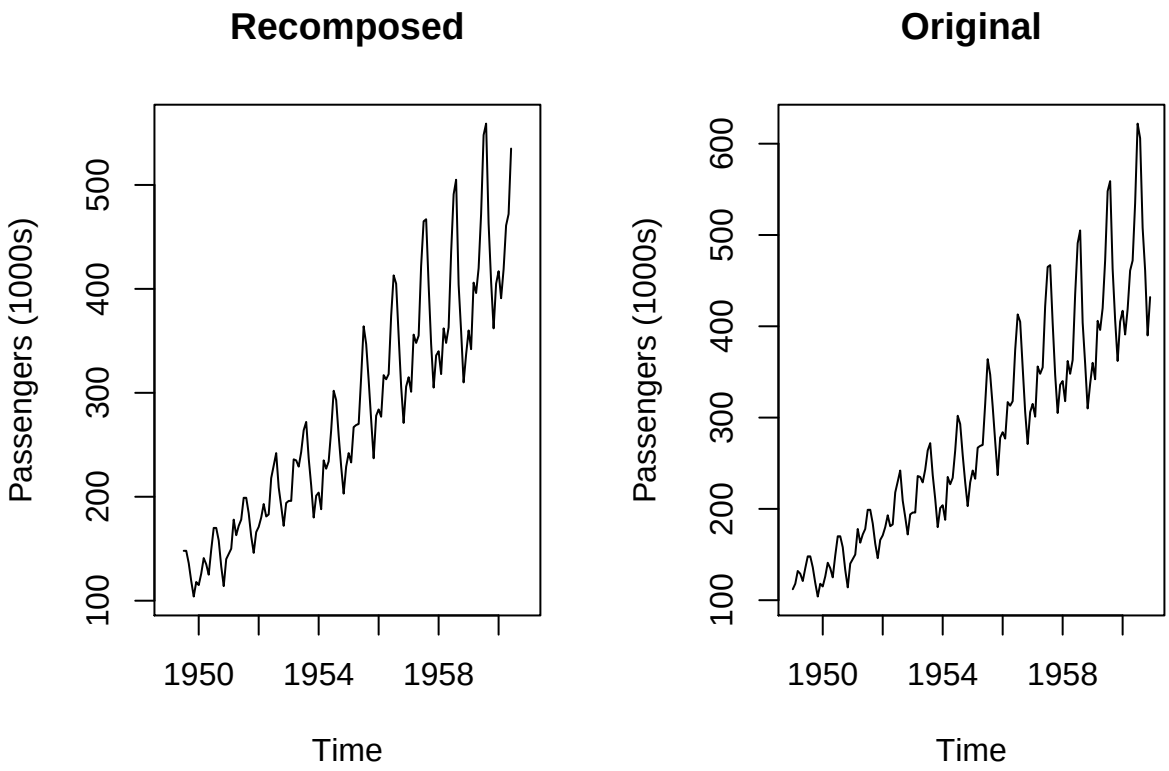


Using these three pieces, we can recompose the original signal.

```
AirPassengers.recomposed <- trend * avgSeasonality * noise

par(mfrow=c(1,2))
plot(AirPassengers.recomposed, main='Recomposed', ylab='Passengers (1000s)')

plot(AirPassengers, main='Original', ylab='Passengers (1000s)')
```



These plots look the same, except for the ends. Some data in the recomposed series is missing at the ends, because the trend used to recompose it was computed using a moving average filter, which doesn't produce an estimate if there isn't enough data on either side of an observation.

We can check that these are the same time series:

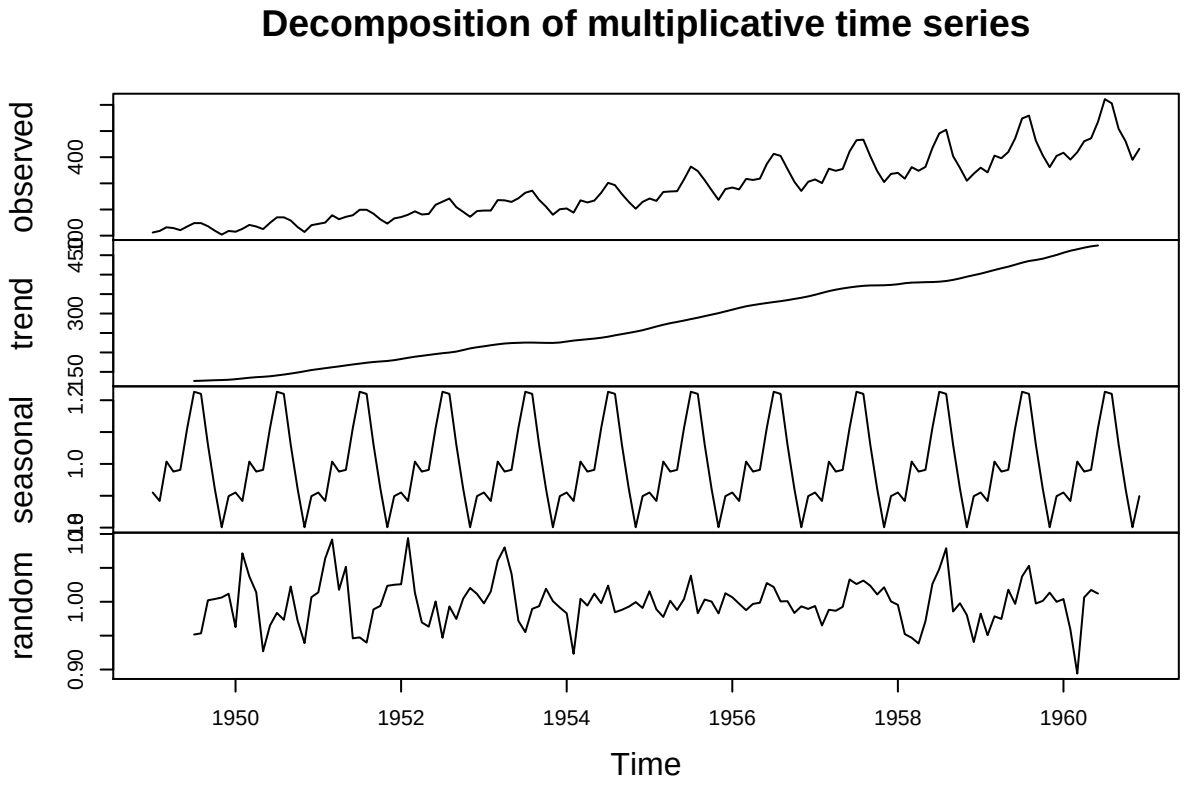
```
sum(AirPassengers.recomposed - AirPassengers, na.rm = TRUE)
```

```
## [1] -5.400125e-13
```

Yep, nice!

We can also use the convenient decompose function to decompose our time series for us.

```
decomposition <- decompose(AirPassengers, "multiplicative")
plot(decomposition)
```


Decomposition of multiplicative time series
observed
trend
seasonal
random
400
300
1.25
1.0
1.00
0.90
1950
1952
1954
1956
1958
1960
Time