Using R - MeasureCamp Training

Richard Fergie — E-Analytica

Afternoon Training Session

Contents

1 Introduction and Scope

Topics to cover:

1. Setup and installation
2. Basic R commands
3. GGPlot
4. The Google Analytics API
5. Reproducible research and Knitr
6. Introduction to forecasting

—

Plus, if we have time:

1. Shiny and "what-if" analysis

2 Setup and installation

2.1 RStudio

RStudio is the easiest way to get started with R. You don't *have* to use it (I prefer not to) but it does make things nice and simple to start with.

Install RStudio from `http://www.rstudio.com/products/rstudio/download/` (or from the USB stick that should be going around the room now).

If RStudio opens without any error messages then you have completed the setup portion of this course.

3 Basic R Commands

In RStudio, enter commands in the pane on the left.

Arithmetic works exactly how you would expect:

```
4+4
```

```
8
```

```
4*4
```

```
16
```

```
4/4
```

```
1
```

You can assign a variable using "<-" or "=". "<-" is more commonly used.

```
x <- "Hello everyone"
x
```

```
Hello everyone
```

A very important concept in R is the list (often known as a vector for computer sciency reasons that aren't important here).

You can create a vector like this:

```
c(1,2,3,4,5,6,7,8,9,10)
```

1
2
3
4
5
6
7
8
9
10

Actually, in R, basically everything is a vector.

```
x <- "This is actually a one element vector"
length(x)
```

```
1
```

This is quite handy as it means aggregate functions nearly always work the way you want (as long as the types are correct)

```
x <- 15
sum(x)
```

```
15
```

```
x <- c(15,30,14)
mean(x)
```

```
19.6666666666667
```

I don't find myself working with one dimensional data very often (and when I do, it is frequently boring).

The easiest structure for two dimensional data in R is the *data.frame*.

```
df <- data.frame(meal=c("Breakfast","Breakfast","Breakfast","Lunch","Dinner"),
                 food=c("Bacon","Sausage","Beans","Pork Pie","Raveoli"),
                 amount=c(2,2,387,1,35)
                 )
df
```

Breakfast	Bacon	2
Breakfast	Sausage	2
Breakfast	Beans	387
Lunch	Pork Pie	1
Dinner	Raveoli	35

You can select rows and columns from a data.frame using numbers

```
df[3,]
```

Breakfast	Beans	387

```
unique(df[,1])
```

Breakfast
Lunch
Dinner

```
df[3,3]
```

```
387
```

It is much easier to refer to columns by their names

```
median(df$amount)
```

```
2
```

3.1 Importing from files

read.csv is the function you want here

```
ad <- read.csv("~/text/UK-Admissions.csv")
names(ad)
```

ONSCode
CommissioningRegionCode
CommissioningRegionName
TotalAdmissions
MaleAdmissions
FemaleAdmissions
X
AdmissionsPer100K
MaleAdmissionsPer100K
FemaleAdmissionsPer100K

```
# length(ad) actually returns the number of columns!
nrow(ad)
```

```
211
```

```
head(ad)
```

E38000056	01C	NHS Eastern Cheshire	20	nil	nil	nil	10	nil	nil
E38000151	01R	NHS South Cheshire	43	14	29	nil	24	16	32
E38000189	02D	NHS Vale Royal	23	6	17	nil	23	12	33
E38000194	02E	NHS Warrington	16	nil	nil	nil	8	nil	nil
E38000196	02F	NHS West Cheshire	45	10	35	nil	20	9	30
E38000208	12F	NHS Wirral	23	nil	nil	nil	7	nil	nil

Our quick check with the *head* function shows that we have quite a lot of missing data in our table.

```
sum(ad$MaleAdmissions)
```

```
nil
```

```
sum(ad$MaleAdmissions, na.rm=T)
```

```
2359
```

We can also fix this by filtering the data.frame

```
filtered <- ad[!is.na(ad$MaleAdmissions),]
sum(filtered$MaleAdmissions)
```

```
2359
```

Data.frames can be sorted based on any of the columns

```
alphabetical <- ad[order(ad$CommissioningRegionName),]
head(alphabetical$CommissioningRegionName)
```

```
NHS Airedale, Wharfedale and Craven
NHS Ashford
NHS Aylesbury Vale
NHS Barking & Dagenham
NHS Barnet
NHS Barnsley
```

3.2 Exercises

1. Which Commissioning Region had the most admissions?
2. Which had the least? (hint: the function *tail* shows you the last rows in a data.frame)
3. What proportion of regions have missing data is the FemaleAdmissions column?
4. Which region is the healthiest?
5. Which region has the biggest difference in health between men and women?

6. Write a function that loads the data and answers one of these questions. (`http://www.ats.ucla.edu/stat/r/library/intro_function.htm` will tell you how to create a function)

7. Write a function that takes a number as an argument and returns the answer to that question. If you can make this work when I use '7' as an input then I will buy you a beer/beverage of your choice. (`https://ramnathv.github.io/pycon2014-r/learn/controls.html` might be helpful for this)

4 GGPlot

GGPlot is the best way to draw charts.

First you need to install it

```
install.packages("ggplot2")
```

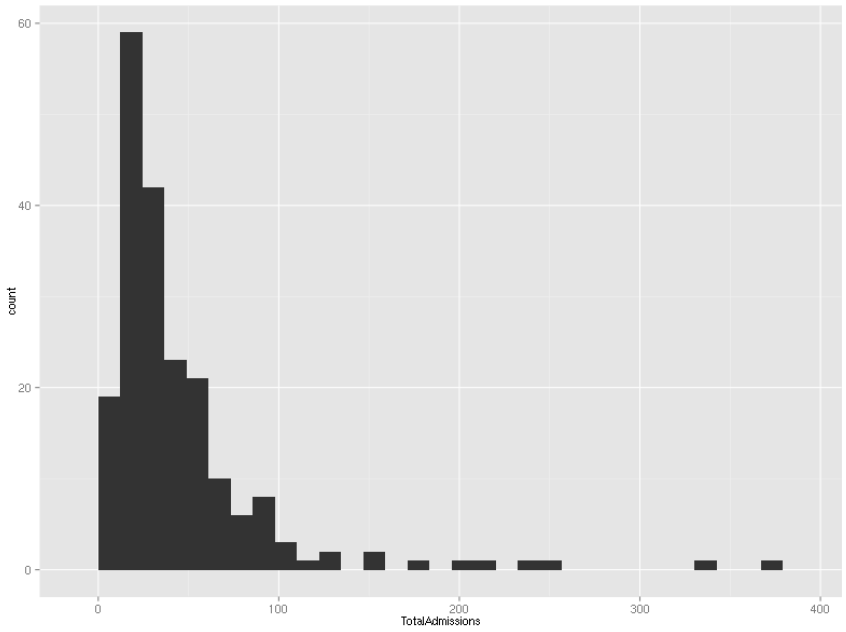
You might be asked to select a download mirror.

Then load the library

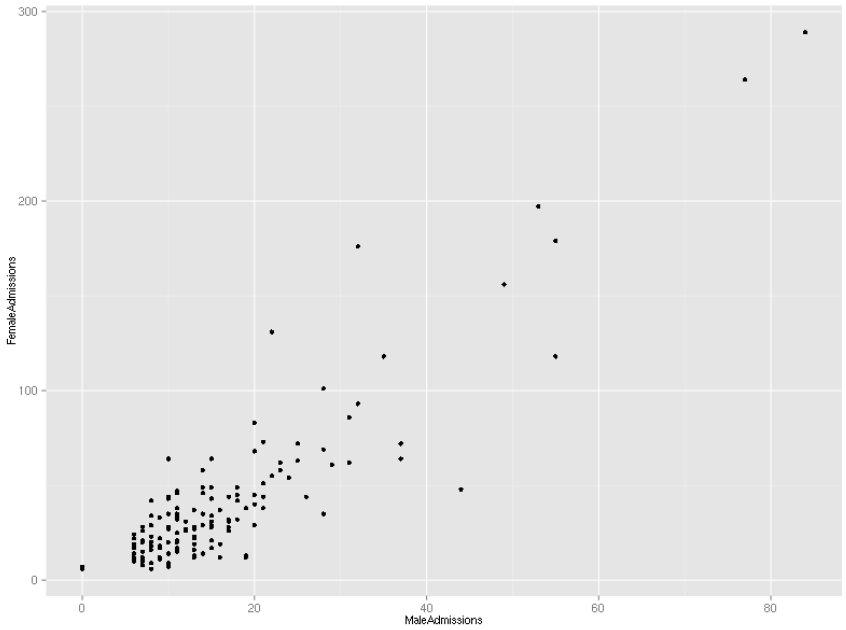
```
library(ggplot2)
```

Now we are ready to plot!

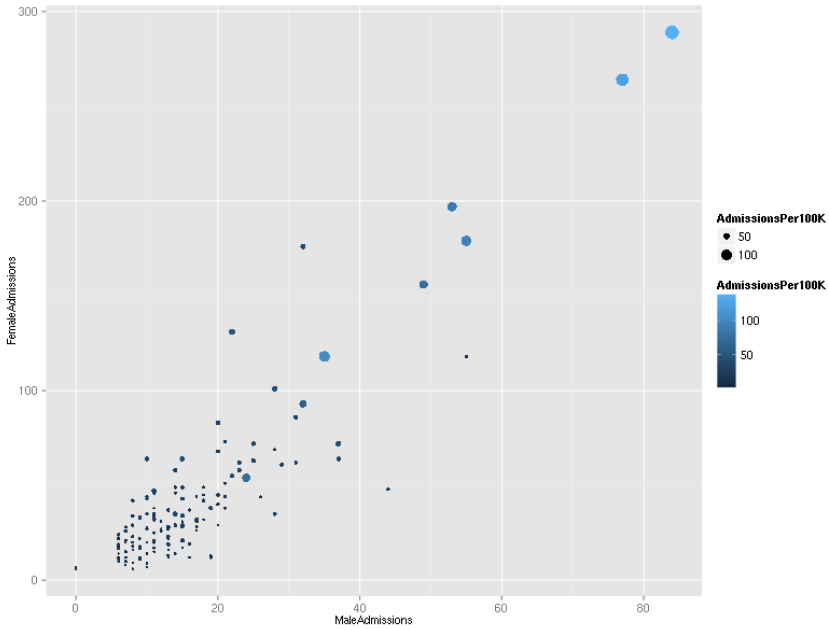
```
ggplot(ad,aes(x=TotalAdmissions)) + geom_histogram()
```




```
ggplot(ad,aes(x=MaleAdmissions,y=FemaleAdmissions)) + geom_point()
```

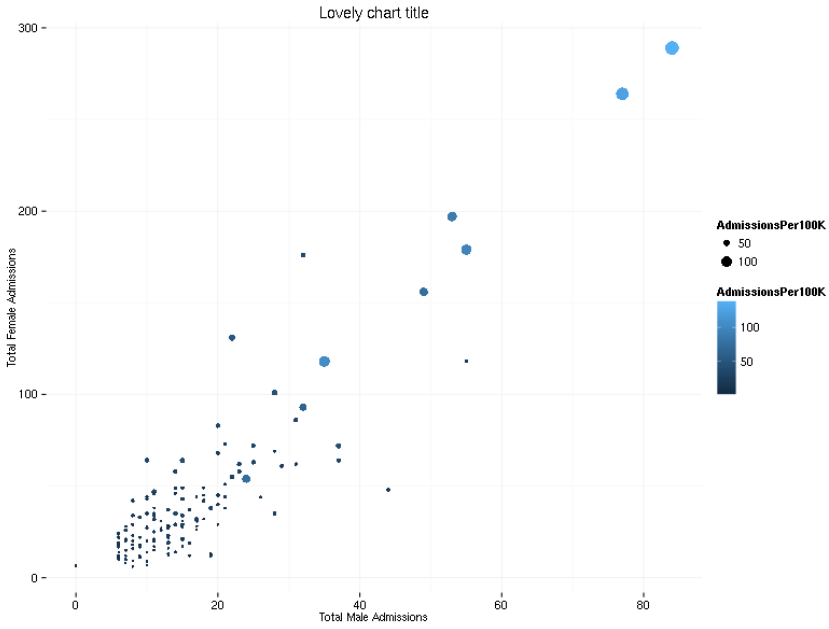



```
ggplot(ad,aes(x=MaleAdmissions,
          y=FemaleAdmissions,
          size=AdmissionsPer100K,
          color=AdmissionsPer100K)) + geom_point()
```



Other chart features can also be 'added' on

```
basechart <- ggplot(ad,aes(x=MaleAdmissions,
                           y=FemaleAdmissions,
                           size=AdmissionsPer100K,
                           color=AdmissionsPer100K)) + geom_point()
basechart + xlab("Total Male Admissions") +
            ylab("Total Female Admissions") +
            ggtitle("Lovely chart title") +
            theme_minimal() # I also like theme_bw()
```



4.1 Exercises

You might have to Google to find out how to do some of these.

1. Experiment with other chart types (bar chart, box plot, pie charts NOT)
2. Add a red coloured horizontal line to a chart (this might come in useful later)
3. Save a chart as a png or svg file.
4. Write a function to load the data and output a chart in a file.

5 The Google Analytics API

5.1 Installing R Packages

The penultimate ingredient in our reporting soup - data from Google Analytics.

There are two libraries for getting data out of Google Analytics and into R:

1. RGoogleAnalytics - used in the last MeasureCamp training. I'm getting errors that I can't be arsed to fix when I try it with a fresh install now.
2. rga - what I normally use and what we will use for this training session.

If you are already comfortable with RGoogleAnalytics then you can continue using it.

```
install.packages("RGA")
```

5.2 Create API access tokens

1. Visit `https://console.developers.google.com/project`
2. Click "Create Project". The AppEngine location stuff (under "advanced options") isn't important for us as we aren't using app engine
3. Wait for the project to be created
4. Click "Enable an API"
5. Enable the Analytics API
6. Click "Consent screen" and fill in the details
7. Click "Credentials" and create a new client id
8. Select "Installed application" and create the credentials
9. Assign the client id and client secret to R variables

```
clientid <- "YOUR CLIENT ID"
clientsecret <- "YOUR CLIENT SECRET"
```

5.3 Authorise rga for access

Firstly load the rga library

```
library(RGA)
```

Now for the tricky bit

```
token <- authorize(client.id = clientid,
                   client.secret = clientsecret
                   cache = "~/.ga.token"
                  )
```

5.4 Getting a profile ID

Do start pulling data you need to be able to tell Google which profile you want to pull the data from.

The easiest way to get an API profile id is through the API Query Explorer which can be found at `https://ga-dev-tools.appspot.com/explorer/`

Once you have authorised this tool you can select a profile and see the API id in the "ids" field. The query explorer is also great for figuring out what dimensions and metrics you want to pull into R.

```
profileid <- "ga:YOUR PROFILE ID"
```

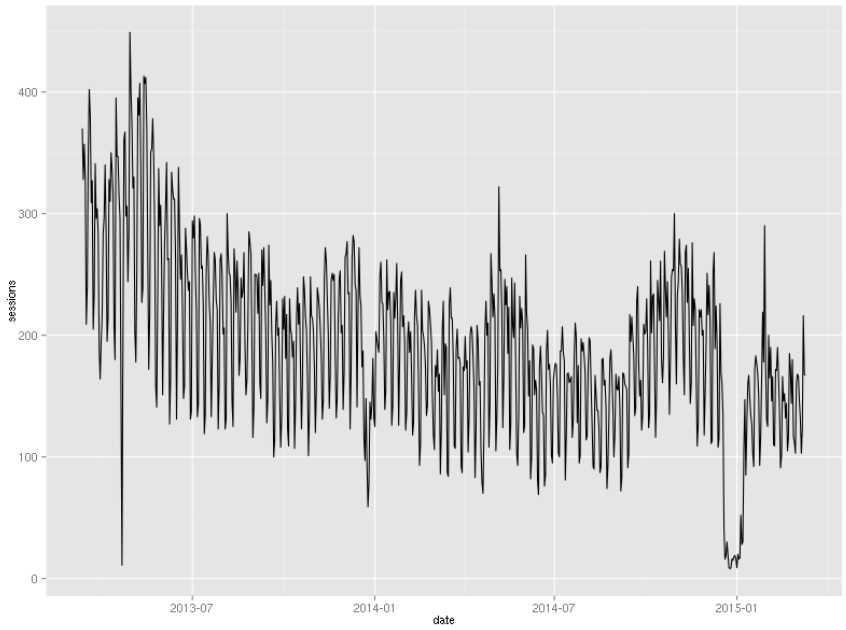
5.5 Get some data!

```
install.packages("lubridate")

library(lubridate)
yesterday <- today() - days(1)
twoyearsago <- today() - days(365*2)

ga_data <- get_ga(profileid,
                  start.date=twoyearsago,
                  end.date=yesterday,
                  dimensions="ga:date",
                  metrics="ga:sessions",
                  )

ggplot(ga_data,aes(x=date,y=sessions)) + geom_line()
```



5.6 Exercises

1. Experiment with different combinations of metrics and dimensions

2. Can you get the "Sampled Data" example from the official documentation (`http://cran.r-project.org/web/packages/RGA/vignettes/reporting-api.html`) working? Be careful to use your own access token

3. Download a tonne of different metrics with the single dimension "ga:date". Which are most highly correlated? (hint: use the *cor* function)

6 Reproducible Research and Knitr

The scientific community has, for years, had problems with the reproducibility of research. Often, raw data (in the rare circumstances) when it is available has been processed in obscure ways before the paper is written up. This means that it is difficult to go from the data to the conclusions in the paper.

Libraries have been developed to enable scientists to embed the processing and analysis of their data into their paper. This helps make their research reproducible.

The best R library for doing this is called Knitr (because it 'knits' together computation and presentation. And because it replaces a library called Sweave).

```
install.package("knitr")
```

Knitr takes an input file that contains a mixture of R code, layout instructions and descriptive text. The R code is evaluated and the results are inserted into the file.

The input file can be written in many formats (markdown, latex etc.) but today we will use HTML as I'm assuming you have some kind of familiarity with it.

Place the following code in document called /knitr-test.Rhtml (the .Rhtml is the important bit).

```
<html>
  <head>
    <title>My first Knitr report</title>
  </head>
  <body>
    <p>Knitr can insert values inline. &pi; is <!--rinline pi --></p>
    <p>And can evaluate whole blocks of code
       <!--begin.rcode
          Sys.Date()
       end.rcode-->
    </p>
  </body>
</html>
```

'Knit' the input file into the output using the following commands

```
library(knitr)
```

```
# this makes life way easier when sharing reports with charts
opts_knit$set(upload.fun = image_uri)
knit("PATH/TO/FILE.Rhtml")
```

The results will be in *YOURFILENAME.html*

There are several clear problems with this output that will be fixed in our next example which will also demonstrate use of the Google Analytics API.

```
<html>
  <head>
    <title>My first Knitr report</title>
  </head>
  <body>
    <!--begin.rcode echo=FALSE, message=FALSE
      library(ggplot2)
      library(RGA)
      library(lubridate)
      t <- readRDS("~/.ga.token")
      token <- t[[1]]
      profileid <- "ga:41734171"

      yesterday <- today() - days(1)
      twoyearsago <- today() - days(365*2)
      ga_data <- get_ga(profileid,
                        start.date=twoyearsago,
                        end.date=yesterday,
                        dimensions="ga:date",
                        metrics="ga:sessions",
                        token=token)
    end.rcode -->

    <!--begin.rcode echo=FALSE
       kable(head(ga_data))
    end.rcode-->
    <p>Here is some insightful commentary about the above table</p>

    <!--begin.rcode echo=FALSE, fig.height=8, fig.width=12
       ggplot(ga_data,aes(x=date,y=sessions)) + geom_line()
    end.rcode-->
```

```
    <p>And here is some commentary about the above chart</p>

  </body>
</html>
```

6.1 Exercises

1. Add another chart and table to the report. Collect some more data from the API to do this
2. Add in a CSS framework to make everything look a bit nicer. I like skeleton.css (`http://getskeleton.com/`) but there are about a billion other frameworks you can use.
3. (Alternative - if you don't like HTML). Produce a PDF report instead
4. Make it so you can automatically produce this report. (hint: create either a function or a script)

Congratulations! You can now create customisable, automated reports. And if you can do a bit of HTML/CSS then you can make them look good too.

7 Introduction to Forecasting

R has good library support for many methods of forecasting. I suspect that it has the best and easiest support out of all the languages you might use for this.

The *forecast* package is the one we will focus on today.

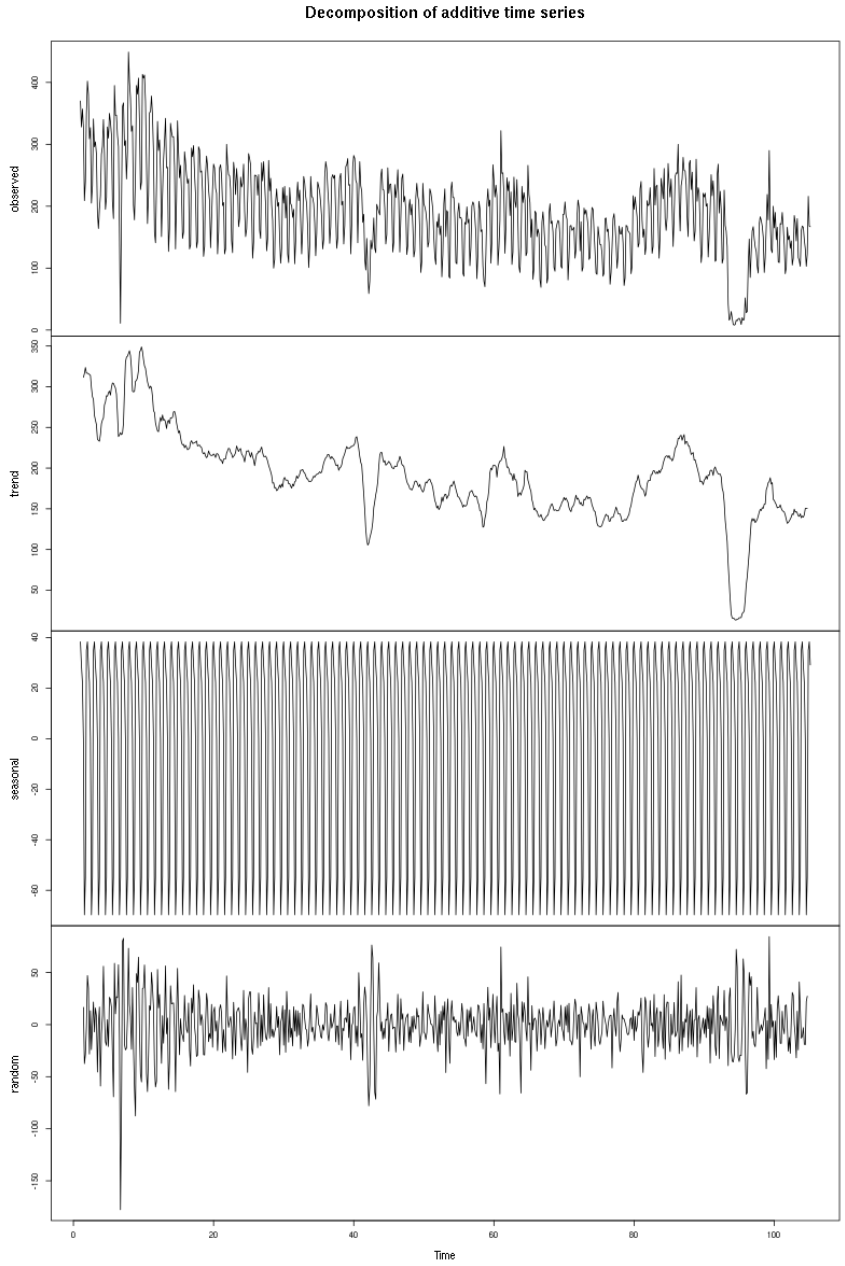
```
install.packages("forecast")
```

To use the *forecast* library the data must first be converted into a time-series. To do this you need to let R know about any periodicity in the data - in this case the data is daily and we are interested in removing the effect of the weekly seasonality.

7.1 Investigating seasonality

```
library(forecast)

timeseries <- ts(ga_data$sessions, frequency=7)
components <- decompose(timeseries)
plot(components)
```



A little explanation of what this plot shows:

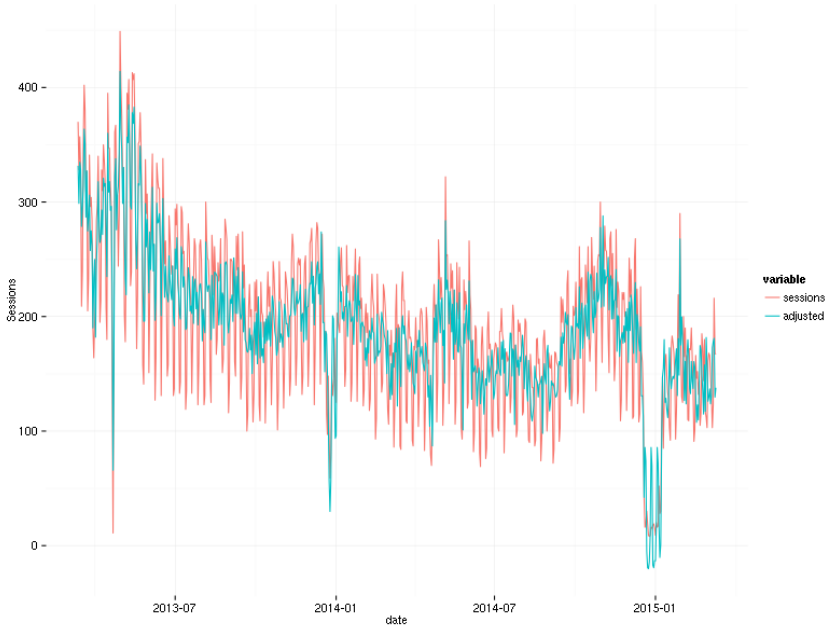
- The top chart is the raw data
- The second chart is the underlying trend (with seasonality removed)

- The third is the seasonal factor
- The fourth is additional 'randomness' in the data - not seasonal and not part of the overall trend either.

We can make a plot of the seasonally adjusted data by subtracting the seasonality numbers from the raw figures

```
# note the way we add a column to a data.frame
ga_data$adjusted <- ga_data$sessions - components$seasonal
library(reshape2)
m<-melt(ga_data, id.vars="date")

ggplot(m,aes(x=date,y=value,color=variable)) + geom_line() +
   ylab("Sessions") + theme_minimal()
```

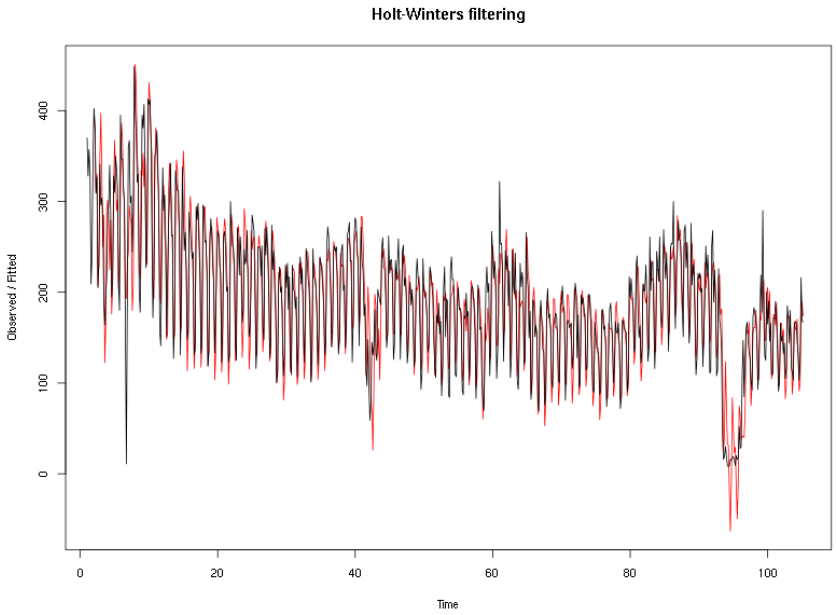


7.2 Making a forecast

For data that has a trend and **additive** (rather than multiplicative) seasonality you can use Holt Winters with exponential smoothing to make a forecast.

```
forecastmodel <- HoltWinters(timeseries)

plot(forecastmodel)
```

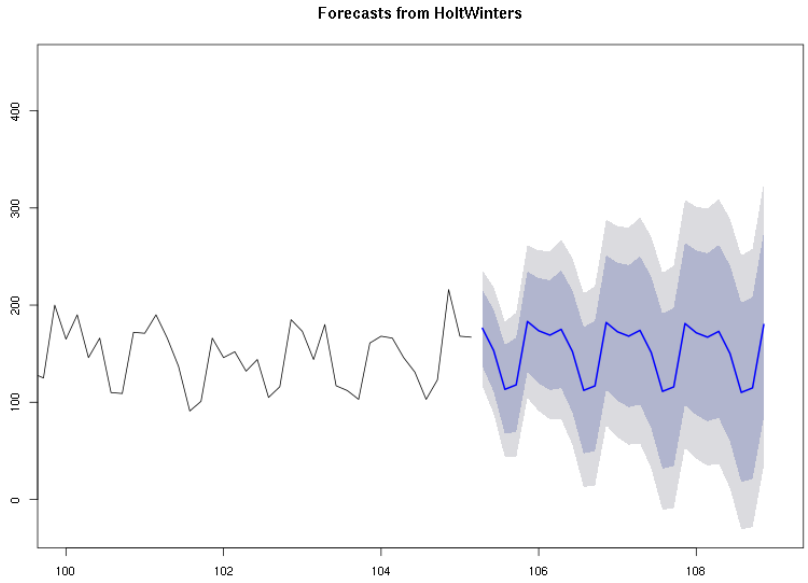


This plot shows that the Holt Winters method aligns fairly closely with the actual numbers (I would be pretty pleased if a forecast I made ever turned out this accurate).

To actually forecast into the future use the *forecast.HoltWinters* function.

```
forecast <- forecast.HoltWinters(forecastmodel, h=26) # 26 weeks in future

plot(forecast, xlim=c(100,109))
```



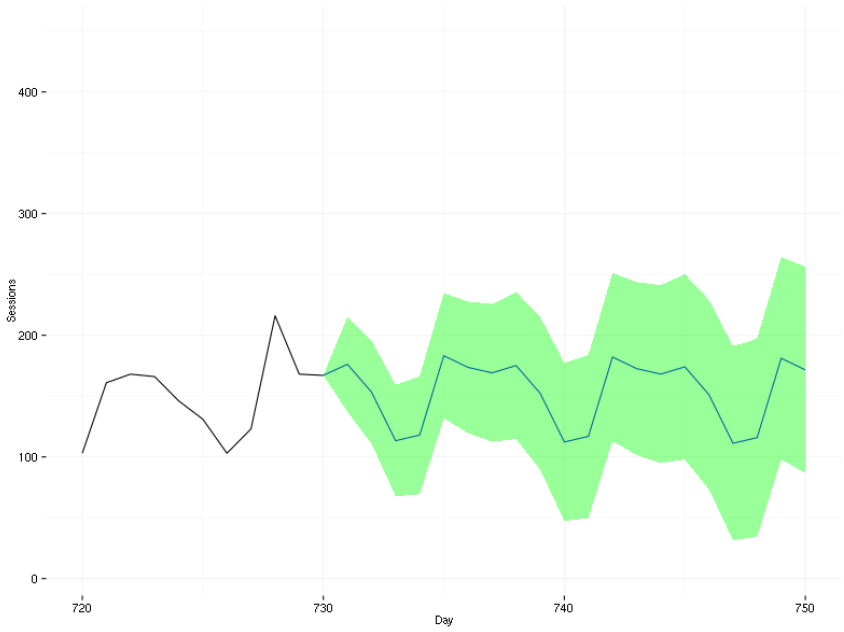
The shaded bands represent 80% and 95% prediction intervals.

With a bit of funkiness we can extract the data from the forecast and plot it with ggplot.

```
forecastdf <- as.data.frame(forecast)
totalrows <- nrow(ga_data) + nrow(forecastdf)

forecastdata <- data.frame(day=c(1:totalrows),
                           actual=c(ga_data$sessions,rep(NA,nrow(forecastdf))),
                           forecast=c(rep(NA,nrow(ga_data)-1),tail(ga_data$sessions,1)
                           forecastupper=c(rep(NA,nrow(ga_data)-1),tail(ga_data$sessio
                           forecastlower=c(rep(NA,nrow(ga_data)-1),tail(ga_data$sessio
                           )

ggplot(forecastdata, aes(x=day)) +
  geom_line(aes(y=actual),color="black") +
  geom_line(aes(y=forecast),color="blue") +
  geom_ribbon(aes(ymin=forecastlower,ymax=forecastupper), alpha=0.4, fill="green") +
  xlim(c(720,750)) +
  xlab("Day") +
  ylab("Sessions") +
  theme_minimal()
```



7.3 Exercises

1. Add a forecast to your Knitr report from earlier.

2. Create a function that produces an alert if a forecast is below a threshold

3. Create a function that produces an alert if the actual value is outside the 80% prediction range for the forecast. i.e. For day n, make a forcast for day n using data from the last n-1 days. Then compare the values for day n with the forecast