Introduction to (some interesting things you can do with) R

Dr. Wolfgang Rolke

Packages / Libraries

- one of the great strength of R
- self contained sets of routines and data that somebody wrote to help with a specific task
- you have an analysis problem? google it and almost always you will find that someone has already done it for you

The main repository for R packages is at https://cran.r-project.org/. As of February 1st there are over 15000!

Basic R paradigm: *if your project has more than three routines, turn it into a package!*

Rmarkdown

- best way to keep everything (text, formulas, data, code, graphs etc.) in one place
- easy syntax for a number of basic objects
- code and output are in the same place and so are always synced
- several output formats (html, latex, word, power point)
- uses html tags or (recommended) latex
- whenever I start a new project I immediately create a corresponding R markdown document
- this workshop is one of them

Case Study: UPR Admissions

Consider the **upr** data set. This is the application data for all the students who applied and were accepted to UPR-Mayaguez between 2003 and 2013.

```
dim(upr)
```

```
## [1] 23666    16
```

- 23666 cases (applications)
- 16 variables (columns)

```r
colnames(upr)
```

```
##  [1] "ID.Code"        "Year"           "Gender"         "Program.Code"
##  [5] "Highschool.GPA" "Aptitud.Verbal" "Aptitud.Matem"  "Aprov.Ingles"
##  [9] "Aprov.Matem"    "Aprov.Espanol"  "IGS"            "Freshmen.GPA"
## [13] "Graduated"      "Year.Grad."     "Grad..GPA"      "Class.Facultad"
```

```r
head(upr, 3)
```

```
##      ID.Code Year Gender Program.Code Highschool.GPA Aptitud.Verbal
## 1 00C2B4EF77 2005      M          502           3.97            647
## 2 00D66CF1BF 2003      M          502           3.80            597
## 3 00AB6118EB 2004      M         1203           4.00            567
##   Aptitud.Matem Aprov.Ingles Aprov.Matem Aprov.Espanol IGS Freshmen.GPA
## 1           621          626         672           551 342         3.67
## 2           726          618         718           575 343         2.75
## 3           691          424         616           609 342         3.62
##   Graduated Year.Grad. Grad..GPA Class.Facultad
## 1        Si       2012      3.33           INGE
## 2        No         NA        NA           INGE
## 3        No         NA        NA       CIENCIAS
```

Note that some columns are numeric, others are not. So data is in the *dataframe* format. This is the standard format in R.

- How many males and females applied?

```r
table(upr$Gender)
```

```
## 
##     F     M 
## 11487 12179
```

Also possible

```r
attach(Gender)
table(Gender)
```

but this is no longer recommended (or needed)

- How did the number of applications change over the years?

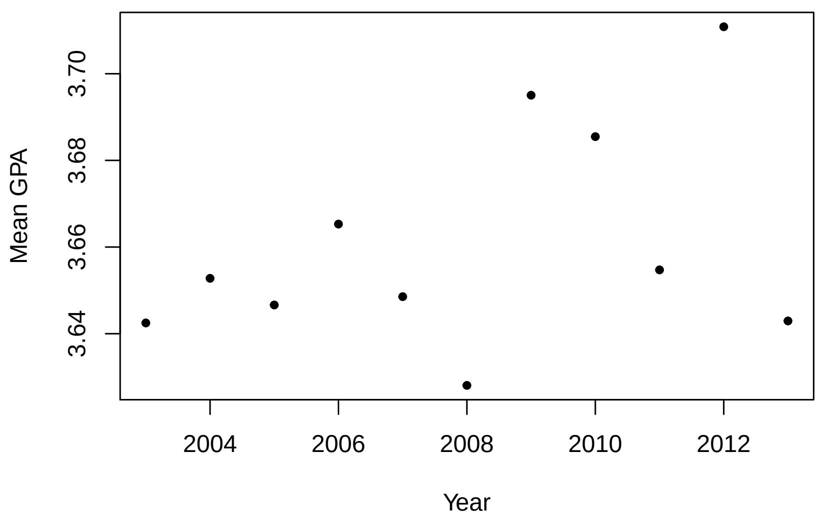
```r
table(upr$Year)
```

```
## 
## 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 
## 2253 2158 2300 2235 2464 2438 2417 2031 1772 1748 1850
```

- Did the high school GPAs change over the years?

```r
yr <- unique(upr$Year)
mean.gpa.year <-
  tapply(upr$Highschool.GPA, upr$Year, mean)
```

```r
plot(yr, mean.gpa.year,
     pch=20,
     xlab="Year",
     ylab="Mean GPA"
    )
```



- How many female students applied between 2009 and 2012 to study Arts and who had a high school GPA over 3.5?

```r
nrow(subset(upr, Gender=="F" &
        Year>=2009 & Year<=2012 &
        Highschool.GPA>3.5 &
        Class.Facultad=="ARTES"))
```

```
## [1] 508
```

Data entry with rio

The package *rio* provides the import and export functions. It figures out from the extension what the file format is. Say we have an EXCEL spreadsheet called salesdata.xlsx:

```r
library(rio)
salesdata <- import("c:/somefolder/salesdata.xlsx")
```

Here is a list of supported file formats:

https://cran.r-project.org/web/packages/rio/vignettes/rio.html

It can also be used to covert data on disk from one format to another. Say we want to turn the EXCEL file into a csv (comma-separated text) file:

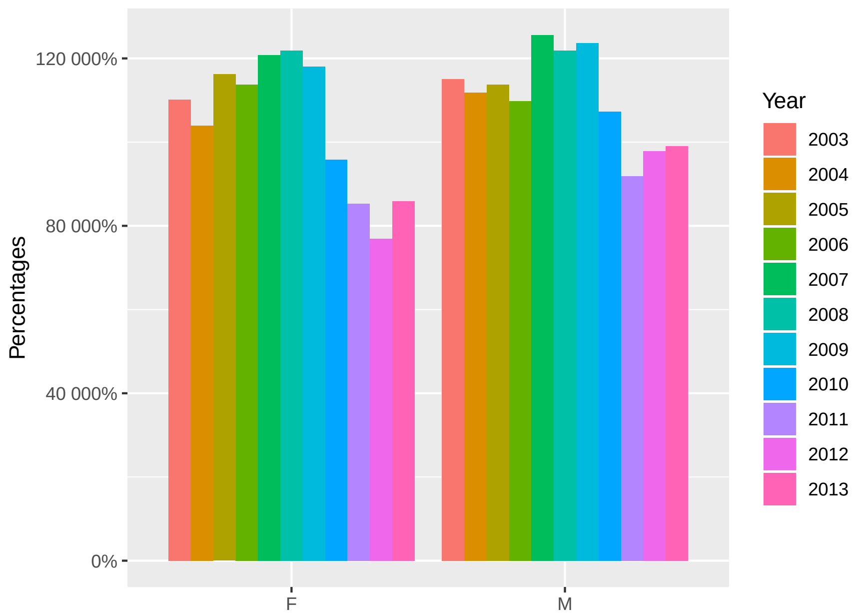
```r
convert("c:/somefolder/salesdata.xlsx",
        "c:/somefolder/salesdata.csv")
```

Graphs with ggplot2

- much nicer looking graphs
- things like proper scaling done automatically
- however, a bit of a learning curve

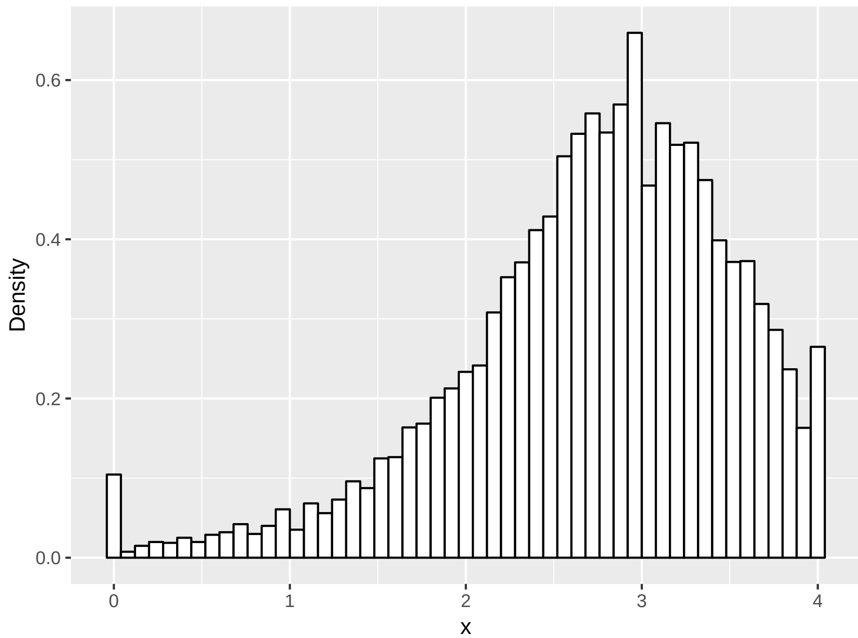
Bar chart

```r
ggplot(upr, aes(x=Gender, fill=factor(Year))) +
    geom_bar(position = "dodge") +
    scale_y_continuous(labels=scales::percent) +
    labs(x="", y="Percentages", fill="Year")
```



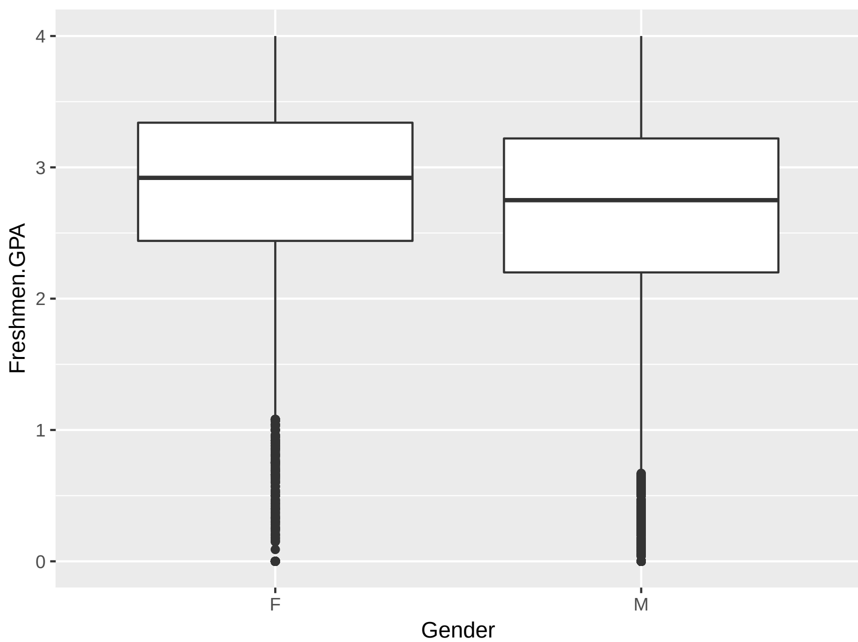
Histogram

```r
bw <- 4/50
ggplot(upr, aes(Freshmen.GPA)) +
  geom_histogram(aes(y = ..density..),
    color = "black",
    fill = "white",
    binwidth = bw) +
    labs(x = "x", y = "Density")
```



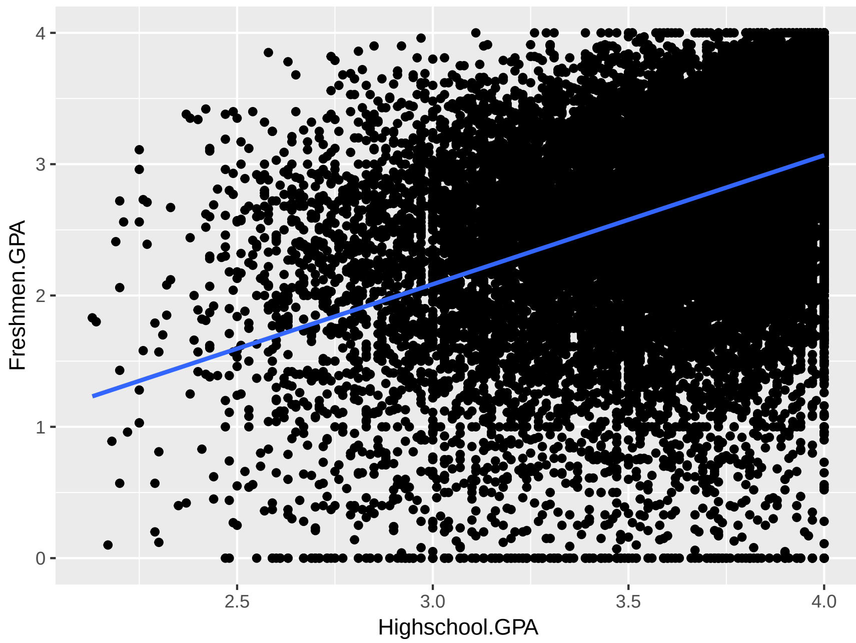
Boxplot

```r
ggplot(upr, aes(Gender, Freshmen.GPA)) +
  geom_boxplot()
```



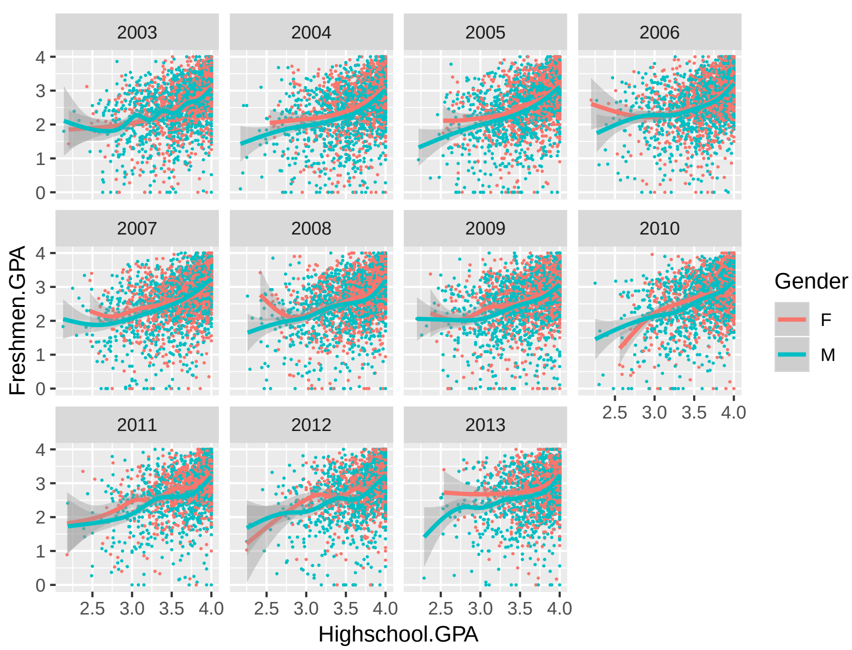
Scatterplot with Least Squares Fit

```r
ggplot(upr, aes(Highschool.GPA, Freshmen.GPA)) +
  geom_point() +
  geom_smooth(method = "lm", se=FALSE)
```



Scatterplot with Loess fit and error bounds, by two Groups

```
ggplot(upr,
       aes(Highschool.GPA, Freshmen.GPA, color=Gender)) +
  facet_wrap(~factor(Year)) +
  geom_point(size=0.1) +
  geom_smooth()
```



Bayesian Analysis with Openbugs

This analysis is based on MCMC simulation, and is computationally quite expensive!

The data set *survey* of the *MASS* library has information on smoking of University students:

```
library(MASS)
tbl <- table(survey$Smoke)
tbl
```

```
##
## Heavy Never Occas Regul
##    11   189    19    17
```

so there are 189 students who never smoked. We want to find a 95% credible interval for the true proportion using a Beta prior. To start we have to define a model:

```
model <- function() {
    # Prior
    p ~ dbeta(1, 1)

    # Likelihood
    y ~ dbin(p, N)
}
```

Notice that this is Openbugs notation, the tilde is not the usual tilde from R.

This model needs to be written to a file on disk:

```
library(R2OpenBUGS)
model.file <- file.path(tempdir(),  "model.txt")
write.model(model, model.file)
```

Next we need to define some objects:

```
N <- as.numeric(sum(tbl)) # Number of Students
y <- N - as.numeric(tbl["Never"]) #Number of Smokers
data <- list("N", "y") #Names of Variables
params <- c("p") #Names of Parameters
inits <- function() { list(p=0.5) }  #Starting Value
```

Next we let Openbugs do the works:

```
out <- bugs(data, inits, params, model.file, n.iter=10000)
```

As a check whether 10000 iterations was enough consider the $\hat{R}$'s. They should all be less than 1.1:

```
all(out$summary[,"Rhat"] < 1.1)
```

```
## [1] TRUE
```

We can get the posterior mean and standard deviation of p from the output.

```r
c(out$mean["p"], out$sd["p"] )
```

```
## $p
## [1] 0.2014755
##
## $p
## [1] 0.02574684
```

The full info is at

```r
print(out, digits=5)
```

```
## Inference for Bugs model at "C:\Users\Wolfgang\AppData\Local\Temp\RtmpKyg2kf/model.tx
## Current: 3 chains, each with 10000 iterations (first 5000 discarded)
## Cumulative: n.sims = 15000 iterations saved
##             mean      sd   2.5%    25%    50%    75%    97.5%    Rhat n.eff
## p        0.20148 0.02575 0.1531 0.1838 0.2009 0.2184  0.25450 1.00129  5100
## deviance 6.45097 1.38373 5.4710 5.5700 5.9070 6.7770 10.47025 1.00119  6900
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = Dbar-Dhat)
## pD = 0.97340 and DIC = 7.42400
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Finally the 95% credible interval:

```r
out$summary[c("p"), c("2.5%", "97.5%")]
```

```
##   2.5%  97.5%
## 0.1531 0.2545
```

ROpenbUGS can be used with Coda for more detailed analyses of the results.

C++ with Rcpp

- speed up computation time by rewriting part of the code in C++
- use available C++ programs in R

```r
library(Rcpp)
library(microbenchmark)
```

Example

we have a data set with points (x, y) and for each point we want to find the nearest neighbor based on Euclidean distance.

Simple R solution:

```r
nn.r <- function(x) {
  n <- nrow(x)
  out <- rep(0, n)
  for(i in 1:n) {
    d <- (x[i, 1]-x[, 1])^2+(x[i, 2]-x[, 2])^2
    d[i] <- max(d)
    out[i] <- which.min(d)
  }
  out
}
```

and here the Rcpp solution:

```cpp
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]

IntegerVector nn_cpp(NumericVector x, NumericVector y) {
  int n=x.length();
  double tmp;
  NumericVector d(n);
  IntegerVector out(n);
  for(int i=0; i<n; ++i) {
    d=(x[i]-x)*(x[i]-x)+(y[i]-y)*(y[i]-y);
    tmp = max(d);
    for(int j=0; j<n; ++j) {
      if((i!=j) & (d[j]<tmp)) {
        out[i] = j+1  ;
        tmp = d[j];
      }
    }
  }
  return out;
}
```

Notice that the R version of C++ is vectorized. It even allows us to use many standard R functions like rnorm etc.!

Let's see that the two routines do the same thing:

```r
x <- matrix(round(rnorm(20), 3), ncol=2 )
cbind(x, nn.r(x), nn_cpp(x[, 1], x[, 2]))
```

```
##         [,1]   [,2] [,3] [,4]
##  [1,]  0.648 -0.077    6    6
##  [2,]  0.110 -1.001    5    5
##  [3,] -2.285 -1.115    4    4
##  [4,] -1.473 -0.205    3    3
```

```
## [5,]  0.060 -1.571    2    2
## [6,]  1.184 -0.475    1    1
## [7,] -1.718  1.565   10   10
## [8,]  0.128 -2.428    5    5
## [9,]  0.439  2.720   10   10
## [10,] -1.162  1.514    7    7
```

So, how fast are they?

```
x <- matrix(round(rnorm(1e3), 3), ncol=2 )
microbenchmark(nn.r(x), nn_cpp(x[, 1], x[, 2]))
```

```
## Unit: microseconds
##                    expr    min       lq     mean  median     uq      max neval
##                 nn.r(x) 2371.8 2412.00 4463.603 2497.20 2751.3 136317.3   100
##  nn_cpp(x[, 1], x[, 2])  718.2  721.35  779.984  734.45  761.3   1631.3   100
##  cld
##    b
##   a
```

so the cpp version is about 4 times faster! In fact, I have seen examples with a speedups of several orders of magnitude.

Parallel and gpu programming with parallel and gpuR

- modern computers have several processor cores.
- many simulation problems are *embarrassingly parallel*

Example say we have 100 data sets and want to find the nearest neighbors for them:

```
x <- as.list(1:100)
for(i in 1:100)
  x[[i]] <- matrix(round(rnorm(1e4), 3), ncol=2 )
```

```
system.time(out <- lapply(x, nn.r))
```

```
##    user  system elapsed
##   22.96    0.00   23.00
```

Now for the parallelized version:

```
library(parallel)
detectCores()
```

```
## [1] 6
```

```
num_cores <- detectCores()-1
cl <- makeCluster(num_cores)
```

```
system.time(out <- parLapply(cl, x, nn.r))
```

```
##    user  system elapsed
##    0.00    0.01    6.75
```

Many computers have a dedicated graphics card (gpu). These are massively parallel processors, but with very limited functionality. R has the gpuR library, mostly useful for matrix manipulations.

Interactive web applications with shiny

shiny is a package that has routines to create interactive web applications that run in a browser.

Here are a number of examples:

- Illustration of different sampling schemes:

```r
library(shiny)
runUrl("http://academic.uprm.edu/wrolke/shiny/sampling.zip")
```

- **Illustration of Integration**

```r
runApp("C:/Users/Wolfgang/Dropbox/R/shiny/integral.zip")
```

These apps run on my laptop. It is also possible to upload (a few) apps to the shinyapps web site. This allows people who have no knowledge of R to run them as well:

- Taylor Polynomials
- Illustration of Bayesian Analysis

Further Reading

- I teach a two semester graduate level course on R. For details go to Computing with R and Computational Statistics with R
- There are literally 100s of books dedicated to R. A short list of those that I have found useful is
  - Learning R
  - Introduction to Data Science with R
  - R Cookbook
  - Advanced R