AMS 597: Statistical Computing

Pei-Fen Kuan (c)

Applied Math and Stats, Stony Brook University

Session Management

- The workplace: All variables created in R are stored in a common workspace. To see which variables are defined in the workspace, you can use the function `ls` (list). you can delete some of the objects and this is done using `rm` (remove).
- The current working directory can be obtained by `getwd()` and changed by `setwd(mydir)`, where `mydir` is a character string.

Session Management

```
getwd()
```

```
## [1] "/Users/peifenkuan/Documents/PFKWorkspace/StonyBrookTea
```

```
setwd("/Users/peifenkuan/Documents/PFKWorkspace/StonyBrookTeac
getwd()
```

```
## [1] "/Users/peifenkuan/Documents/PFKWorkspace/StonyBrookTea
```

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.8, 1.65, 1.9, 1.74, 1.91)
ls()
```

```
## [1] "height" "weight"
```

```
objects()
```

```
## [1] "height" "weight"
```

Session Management

```r
rm(height, weight)
rm(list = ls())
save.image()
```

Session Management

- Textual output: It is important to note that the workspace consists only of R objects, not of any of the output that you have generated during a session
- If you want to save your output, use "Save to File" from the File menu in Windows or use standard cut-and-paste facilities
- An alternative way of diverting output to a file is to use the `sink()` function

```
sink("myfile1.txt")
print("hello world")
sink()

sink("myfile2.txt")
cat("hello world")
sink()
```

Session Management

- Scripting: Beyond a certain level of complexity, you will not want to work with R on a line-by-line basis. In such cases, it is better to work with R scripts, collections of lines of R code stored either in a file.
- You could use the `source()` function, which takes the input (i.e., the commands from a file) and runs them.

Session Management

- Getting help

```
help.search("kendal")
```

- Packages: An R installation contains one or more libraries of packages. Some of these packages are part of the basic installation.
- Others can be downloaded from CRAN, which currently hosts over 1000 packages for various purposes. Another source is Bioconductor. You can even create your own packages.

Session Management

- A package is loaded into R using the library command, so to load the `survival` package you should enter

```
library(survival)
```

- Built-in data: Many packages, both inside and outside the standard R distribution, come with built-in data sets. Such data sets can be rather large, so it is not a good idea to keep them all in computer memory at all times.

```
library(ISwR)
data(thuesen)
str(thuesen)
```

Session Management

- Subset and transform functions

```r
thue2 <- subset(thuesen, blood.glucose < 7)
thue2
```

```
##    blood.glucose short.velocity
## 6            5.3           1.49
## 11           6.7           1.25
## 12           5.2           1.19
## 15           6.7           1.52
## 17           4.2           1.12
## 22           4.9           1.03
```

Session Management

```r
thue3 <- transform(thuesen, log.gluc = log(blood.glucose))
thue3[1:5, ]
```

```
##   blood.glucose short.velocity log.gluc
## 1          15.3           1.76 2.727853
## 2          10.8           1.34 2.379546
## 3           8.1           1.27 2.091864
## 4          19.5           1.47 2.970414
## 5           7.2           1.27 1.974081
```

Wide *versus* Long data format

- A wide format contains values that do not repeat in the first column

```
d1 <- data.frame(ID = LETTERS[1:4], T1 = 1:4,
    T2 = c(1:4) * 2, T3 = c(1:4) * 3)
d1
```

```
##   ID T1 T2 T3
## 1  A  1  2  3
## 2  B  2  4  6
## 3  C  3  6  9
## 4  D  4  8 12
```

Wide *versus* Long data format

- A long format contains values that do repeat in the first column

```
d2 <- data.frame(ID = rep(LETTERS[1:4], each = 3),
    Time = rep(paste("T", 1:3, sep = ""),
        4), Value = rep(c(1:3), 4) * rep(1:4,
        each = 3))
d2
```

```
##    ID Time Value
## 1   A   T1     1
## 2   A   T2     2
## 3   A   T3     3
## 4   B   T1     2
## 5   B   T2     4
## 6   B   T3     6
## 7   C   T1     3
## 8   C   T2     6
## 9   C   T3     9
## 10  D   T1     4
```

Wide *versus* Long data format

- Converting wide to long data format

```r
reshape(d1, varying = c("T1", "T2", "T3"),
    v.names = "Variable", timevar = "Time",
    times = c("T1", "T2", "T3"), idvar = "ID",
    direction = "long")
```

```
##      ID Time Variable
## A.T1  A   T1        1
## B.T1  B   T1        2
## C.T1  C   T1        3
## D.T1  D   T1        4
## A.T2  A   T2        2
## B.T2  B   T2        4
## C.T2  C   T2        6
## D.T2  D   T2        8
## A.T3  A   T3        3
## B.T3  B   T3        6
```

Wide *versus* Long data format

- Converting long to wide data format

```
reshape(d2, timevar = "Time", idvar = c("ID"),
    direction = "wide")
```

```
##    ID Value.T1 Value.T2 Value.T3
## 1   A        1        2        3
## 4   B        2        4        6
## 7   C        3        6        9
## 10  D        4        8       12
```

Introduction to `dplyr`

- The `dplyr` package contains functions for performing data manipulation more elegantly.
- Alternatively, you can install `tidyverse` which is a collection of R packages including `ggplot2` and `dplyr`.
- Commonly used functions in `dplyr`:
  - `filter()`: subset the rows of a data set
  - `arrange()`: reorder the rows
  - `select()`: subset the columns of a data set
  - `mutate()`: add new columns that are based on calculations on data in other columns
  - `summarise()`: perform summary calculations (mean, max, etc.) on a set of data

Introduction to dplyr

```r
library(dplyr)
mydat <- data.frame(month = rep(12:1, each = 30),
    day = rep(1:30, 12), name = rep(c("Bob",
        "Rob", "Deb", "Ann"), each = 90),
    mscore = runif(360))
mytib <- as_tibble(mydat)
mytib
```

```
## # A tibble: 360 x 4
##    month   day name  mscore
##    <int> <int> <chr>  <dbl>
##  1    12     1 Bob    0.742
##  2    12     2 Bob    0.921
##  3    12     3 Bob    0.418
##  4    12     4 Bob    0.822
##  5    12     5 Bob    0.940
##  6    12     6 Bob    0.144
##  7    12     7 Bob    0.327
```

Introduction to dplyr

```r
filter(mytib, name == "Bob")
```

```
## # A tibble: 90 x 4
##    month   day name  mscore
##    <int> <int> <chr>  <dbl>
##  1    12     1 Bob    0.742
##  2    12     2 Bob    0.921
##  3    12     3 Bob    0.418
##  4    12     4 Bob    0.822
##  5    12     5 Bob    0.940
##  6    12     6 Bob    0.144
##  7    12     7 Bob    0.327
##  8    12     8 Bob    0.792
##  9    12     9 Bob    0.717
## 10    12    10 Bob    0.719
## # ... with 80 more rows
```

Introduction to dplyr

```r
arrange(mytib, month, name)
```

```
## # A tibble: 360 x 4
##    month   day name  mscore
##    <int> <int> <chr>  <dbl>
##  1     1     1 Ann   0.277
##  2     1     2 Ann   0.840
##  3     1     3 Ann   0.645
##  4     1     4 Ann   0.254
##  5     1     5 Ann   0.428
##  6     1     6 Ann   0.123
##  7     1     7 Ann   0.500
##  8     1     8 Ann   0.0155
##  9     1     9 Ann   0.716
## 10     1    10 Ann   0.489
## # ... with 350 more rows
```

Introduction to dplyr

```r
select(mytib, month, day)
```

```
## # A tibble: 360 x 2
##    month   day
##    <int> <int>
##  1    12     1
##  2    12     2
##  3    12     3
##  4    12     4
##  5    12     5
##  6    12     6
##  7    12     7
##  8    12     8
##  9    12     9
## 10    12    10
## # ... with 350 more rows
```

Introduction to dplyr

- Exercise: How to select all columns except “name’ ’?
- How about column which starts with “m’ ’?

Introduction to dplyr

```r
mutate(mytib, monthday = paste(month, day, sep = "-"))
```

```
## # A tibble: 360 x 5
##    month   day name  mscore monthday
##    <int> <int> <chr>  <dbl> <chr>
##  1    12     1 Bob    0.742 12-1
##  2    12     2 Bob    0.921 12-2
##  3    12     3 Bob    0.418 12-3
##  4    12     4 Bob    0.822 12-4
##  5    12     5 Bob    0.940 12-5
##  6    12     6 Bob    0.144 12-6
##  7    12     7 Bob    0.327 12-7
##  8    12     8 Bob    0.792 12-8
##  9    12     9 Bob    0.717 12-9
## 10    12    10 Bob    0.719 12-10
## # ... with 350 more rows
```

Introduction to dplyr

```
by_name <- group_by(mytib, name)
summarise(by_name, mean_mscore = mean(mscore, na.rm = TRUE))
```

```
## # A tibble: 4 x 2
##   name  mean_mscore
##   <chr>       <dbl>
## 1 Ann         0.473
## 2 Bob         0.503
## 3 Deb         0.553
## 4 Rob         0.536
```

Introduction to dplyr

- The pipe %>%: for shortening and simplying the code.

```
mytib %>% group_by(name) %>% summarise(mean_mscore = mean(msco
    na.rm = TRUE))
```

```
## # A tibble: 4 x 2
##   name  mean_mscore
##   <chr>       <dbl>
## 1 Ann         0.473
## 2 Bob         0.503
## 3 Deb         0.553
## 4 Rob         0.536
```

Graphics subsystem

- Plot layout: A standard x-y plot has an x and a y title label generated from the expressions being plotted.

```
x <- runif(50, 0, 2)
y <- runif(50, 0, 2)
plot(x, y, main = "Main title", sub = "subtitle",
    xlab = "x-label", ylab = "y-label")
```

- Inside the plotting region, you can place points and lines that are either specified in the plot call or added later with points and lines. You can also place a text with:

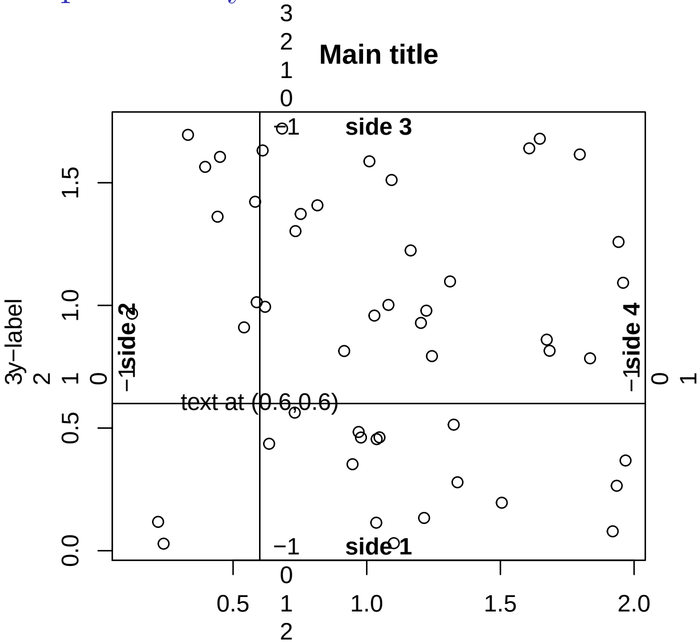
```
plot(x, y, main = "Main title", sub = "subtitle",
    xlab = "x-label", ylab = "y-label")
text(0.6, 0.6, "text at (0.6,0.6)")
abline(h = 0.6, v = 0.6)
```

Graphics subsystem

- The margin coordinates are used by the `mtext()` function. They can be demonstrated as follows:

```
plot(x, y, main = "Main title", sub = "subtitle",
    xlab = "x-label", ylab = "y-label")
text(0.6, 0.6, "text at (0.6,0.6)")
abline(h = 0.6, v = 0.6)
for (side in 1:4) mtext(-1:4, side = side,
    at = 0.7, line = -1:4)
mtext(paste("side", 1:4), side = 1:4, line = -1,
    font = 2)
```

Graphics subsystem



Building a plot from pieces

- High-level plots are composed of elements, each of which can also be drawn separately.

```
plot(x, y, type = "n", xlab = "", ylab = "")
```

```
plot(x, y, type = "n", xlab = "", ylab = "",
    axes = F)
```

Building a plot from pieces

- To add the plot elements, evaluate the following:

```r
plot(x, y, type = "n", xlab = "", ylab = "")
```

```r
plot(x, y, type = "n", xlab = "", ylab = "",
    axes = F)
points(x, y)
axis(1)
axis(2, at = seq(0.2, 1.8, 0.2))
box()
title(main = "Main title", sub = "subtitle",
    xlab = "x-label", ylab = "y-label")
```

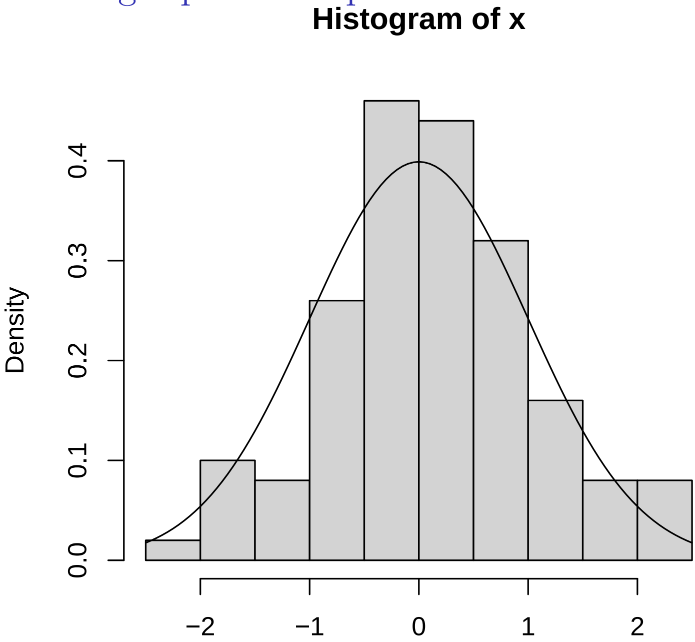
Building a plot from pieces

- Combining plots: Consider overlaying a histogram with a normal density

```
x <- rnorm(100)
hist(x, freq = F)
curve(dnorm(x), add = T)
```

```
h <- hist(x, plot = F)
ylim <- range(0, h$density, dnorm(0))
hist(x, freq = F, ylim = ylim)
curve(dnorm(x), add = T)
```

Building a plot from pieces

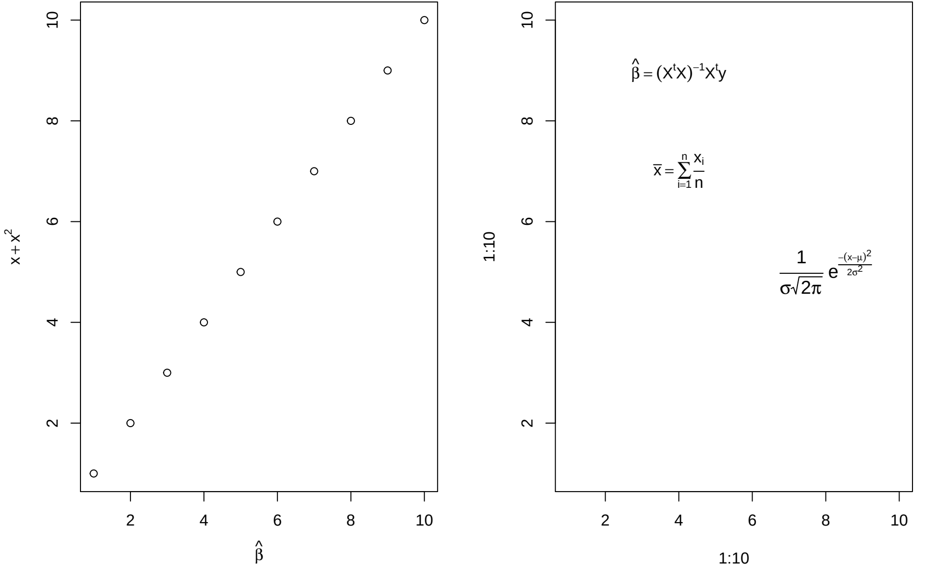


Building a plot from pieces

- R graphics also allows one to add mathematical annotations

```
par(mfrow = c(1, 2))
plot(1:10, 1:10, xlab = expression(hat(beta)),
    ylab = expression(x + x^2))
plot(1:10, 1:10, type = "n")
text(4, 9, expression(hat(beta) == (X^t *
    X)^{
    -1
} * X^t * y))
text(4, 7, expression(bar(x) == sum(frac(x[i],
    n), i == 1, n)))
text(8, 5, expression(paste(frac(1, sigma *
    sqrt(2 * pi)), " ", plain(e)^{
    frac(-(x - mu)^2, 2 * sigma^2)
})), cex = 1.2)
```

Building a plot from pieces



Building a plot from pieces

- More examples can be found at `?plotmath`
- You may save the plots in R in various formats: `jpeg()`, `bmp()`, `png()`, `tiff()`, `postscript()`, `pdf()`
- E.g. `pdf('myboxplot.pdf',width=5,height=10)`

R: Basic graphic functions

Method	in (graphics)	in (package)
Scatter plot	plot	
Add regression line to plot	abline	
Add reference line to plot	abline	
Reference curve	curve	
Histogram	hist	truehist (MASS)
Bar plot	barplot	
Plot empirical CDF	plot.ecdf	
QQ Plot	qqplot	qqmath (lattice)
Normal QQ plot	qqnorm	
QQ normal ref. line	qqline	
Box plot	boxplot	
Stem plot	stem	

R programming

- It is possible to write your own R functions.

```
functionName <- function(argList) {
    functionBody
    return(retList)
}
sumXY <- function(x, y) {
    z <- x + y
    return(list(sumxy = z, x = x, y = y))
}
sumXY(2, 3)
```

```
## $sumxy
## [1] 5
##
## $x
## [1] 2
##
## $
```

R programming

- Example of a function generating the plot from page 30:

```
hist.with.normal <- function(x,
xlab=deparse(substitute(x)),...){
  h <- hist(x, plot=F, ...)
  s <- sd(x)
  m <- mean(x)
  ylim <- range(0,h$density,dnorm(0,sd=s))
  hist(x, freq=F, ylim=ylim, xlab=xlab, ...)
  curve(dnorm(x,m,s), add=T)
}
```

Flow control

- Consider the following code that implements a version of Newton's method for calculating the square root of y.
- `while(condition)` expression construction.

```
y <- 12345
x <- y/2
while (abs(x * x - y) > 1e-10) x <- (x +
    y/x)/2
x
```

```
## [1] 111.1081
```

```
x^2
```

```
## [1] 12345
```

Flow control

- `repeat()` construction.

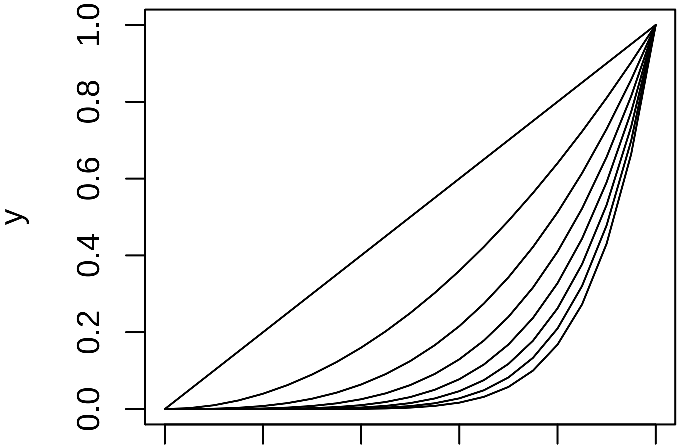
```
x <- y/2
repeat {
    x <- (x + y/x)/2
    if (abs(x * x - y) < 1e-10)
        break
}
x
```

```
## [1] 111.1081
```

Flow control

- for loop

```
x <- seq(0, 1, 0.05)
plot(x, x, ylab = "y", type = "l")
for (j in 2:8) lines(x, x^j)
```



Exercise

- Write an R function to create a Fibonacci sequence of length N.
- Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34,...

Classes and generic functions

- Object-oriented programming is about creating coherent systems of data and methods that work upon them. A prototype example is the print method: It makes sense to print many kinds of data objects, but the print layout will depend on what the data object is.

```
weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.8, 1.65, 1.9, 1.74, 1.91)
bmi <- weight/height^2
t.test(bmi, mu = 22.5)$p.value
```

```
## [1] 0.7442183
```

Reading from and writing to a file

- The most convenient way of reading data into R is via the function called `read.table()`.
- Data url: "http://www.ams.sunysb.edu/~pfkuan/Teaching/AMS597/Data/m_logret_10stocks.txt"

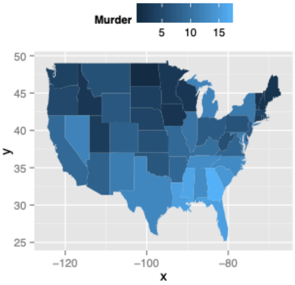
```
logret <- read.table("http://www.ams.sunysb.edu/~pfkuan/Teachi
logret
```

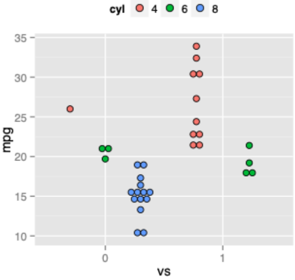
```
##          Date        AAPL        ADBE         ADP
## 1    1/3/1994  0.048913361  0.135255005 -0.022895763  0.062
## 2    2/1/1994  0.048848568 -0.014934233 -0.009507324  0.020
## 3    3/1/1994 -0.040760913 -0.078928087  0.000872954  0.162
## 4    4/4/1994 -0.044394486  0.041944521  0.002174739 -0.073
## 5    5/2/1994 -0.009571616  0.031889144  0.015763276  0.010
## 6    6/1/1994 -0.042543456 -0.022343827 -0.000837598 -0.033
## 7    7/1/1994  0.103924521  0.056810193 -0.013626043  0.033
## 8    8/1/1994  0.032859345  0.009365203  0.021518747  0.031
## 9    9/1/1994 -0.031286762  0.011429462  0.018140226  0.011
```

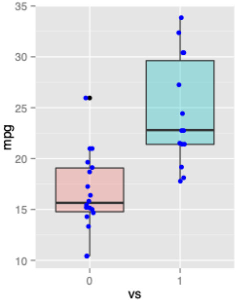
Reading from and writing to a file

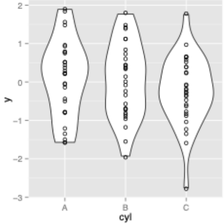
- One may also read a csv file using `read.csv()`
- Writing a data.frame into a file can be done using `write.table()` or `write.csv()`
- E.g., `write.table(df2,file="Myfile.txt",row.names=F, sep='\t',quote=F)`

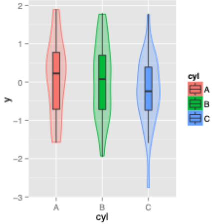
Advanced graphics in R (ggplot2)

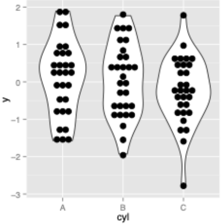












Advanced graphics in R (ggplot2)

- `ggplot2` produces elegant graphics
- To install, `install.packages("ggplot2")`
- One initializes a `ggplot2` object with `ggplot()`, and adds on layers (e.g., `geom_point()`, `geom_line()`, `geom_histogram()`) on a `data.frame` object.

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
x <- sort(rnorm(100))
mydat <- data.frame(myx = x, myy = x + rnorm(100),
    mygroup = factor(rep(1:5, each = 20)))
ggplot(mydat, aes(myx, myy, colour = mygroup)) +
    geom_point()
```

```
ggplot(mydat, aes(myx, myy, colour = mygroup)) +
    geom_line()
```

```
ggplot(mydat, aes(myx, myy, colour = mygroup)) +
```

Advanced graphics in R (ggplot2)

- Examples of customizing plotting symbol size, legends, etc.

```
df2 <- data.frame(group = rep(c("Discovery",
    "Replication"), c(100, 50)),
    y = c(rnorm(100, 1), rnorm(50,
        2)))
ggplot(df2, aes(group, y)) + geom_boxplot(aes(fill = factor(gr
    ggtitle("My Plot") + theme(text = element_text(size = 20),
    legend.text = element_text(size = 20),
    legend.title = element_blank(),
    plot.title = element_text(hjust = 0.5))
```

Advanced graphics in R (ggplot2)

- To create multiple plots on the same window

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```
p1 <- ggplot(mydat, aes(myx, myy, colour = mygroup)) +
    geom_point()
p2 <- ggplot(mydat, aes(myx, myy, colour = mygroup)) +
    geom_line()
p3 <- ggplot(mydat, aes(myx, myy, colour = mygroup)) +
    geom_point() + geom_line()
grid.arrange(p1, p2, p3, ncol = 2)
```

Advanced graphics in R (ggplot2)

- More examples are given in http://ggplot2.tidyverse.org/reference/
- Cheatsheet for ggplot2 https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-visualization.pdf

Reading from a file (large datasets)

- R packages `readr` and `data.table` are two packages for reading and manipulating large datasets (e.g., 100 GB RAM).
- Efficiency: up to 10x faster than read.table or data.frame in the base package.
- http://r4ds.had.co.nz/data-import.html
- http://readr.tidyverse.org/
- https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

Introduction to R Markdown

- R Markdown is widely used to generate reproducible research reports.
- R Markdown is a plaintext file with extension `.Rmd`
- First install the `rmarkdown` package.
- After creating the R Markdown file (e.g., myfirstRMD.Rmd), you will type `render(myfirstRMD.Rmd)` to generate report from the file.
- If you use R studio, you can alternatively click on the "Knit'' button to generate the report.

Introduction to R Markdown

- The common reports/outputs that R Markdown can generate include `html_document`, `pdf_document`, `beamer_presentation` and `ioslides_presentation`.
- Other outputs supported by R Markdown is available https://rmarkdown.rstudio.com/lesson-9.html
- R Markdown complete reference: https://rmarkdown.rstudio.com/
- R Markdown reference guide: https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf
- R Markdown cheatsheet: https://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf
- Demo