Today's Class

More data management in R:

- Choosing subsets
- Merging

Plotting in R

Read in data

Syntax:

```
read.table(file_name, header=TRUE)
```

For example:

```
data = read.table("dd_dev_transcriptome.txt", header=TRUE)
```

Alternatively, create a variable for the input file:

```
infile = "dd_dev_transcriptome.txt"
data = read.table(infile, header=TRUE)
```

Note:

`header=TRUE` means that the first row contains the names of the columns. If the file does not contain a header line, then R will call the columns V1, V2, V3, etc. You can rename the columns using `names()`

The header is different from any number of optional comment lines. The default comment character is "#", but this can be configured as an optional argument to read.table(). See ?read.table for more details.

Accessing your data

In a data frame, use square brackets `[i,j]` to specify the item in the i^{th} row and j^{th} column, e.g.,:

```
mydata = read.table("dd_dev_transcriptome.txt, header=TRUE)
mydata[1,6]    # prints the value of row 1, col 6
```

If you leave the row or column blank, it means "all":

```
mydata[1,]    # prints row 1, all columns
mydata[,6]    # prints all rows, column 6
```

Syntax	Meaning
`mydata[,1]`	`all rows, column 1`
`mydata[1,]`	`row 1, all columns`
`mydata[1:10,4:16]`	`rows 1 to 10, columns 4 to 16`
`mydata[c(1,5,6),c(5,1,2)]`	`rows 1,5,6 and columns 5,1,2`
`mydata[-10]`	`all except col 10`

Subsetting based on a pattern match

Use `grep()` to find a match to a string

Syntax:

```
grep("search_pattern", search_in)
```

See ?grep for more information.

Example: Subset the dataset called `data` to genes with names (col 2) that match "act". Keep only columns 1 through 10.

```
act_genes = data[grep("act",data[,2]),1:10]
```

Subset to: Rows where column 2 contains "act" & Columns 1 through 10

General Format:

```
dataset[rows,columns]
```

Note: You are subsetting simply by referring to a subset of the data (and then assigning that subset to a variable, in this case "act_genes"*)*

Choosing a Subset of Your Data

Many ways to subset!

General Format:
`dataset[rows, columns]`

Choose specific columns or rows:

```
data = read.table("dd_dev_transcriptome.txt", header=TRUE)
d2 = data[c(1,5,6), c(5,1,2)]
```

Choose values that meet some criterion:

```
d2 = data[data$tp.hr00>30,]
d2 = data[data$tp.hr00>30 & data$tp.hr04>50,]
```

Note the "logic" indicators:
& = "and"
| = "or"

Can also subset using `subset()` function:

```
d2 = subset(data, data$tp.hr00>30)
```

See help for more details

Adding new rows and columns

```
cbind()     # bind columns
rbind()     # bind rows
```

Example 1:

Generate a 2x2 matrix like this:

```
1   2
3   4
```

```
vec1 = c(1,2)
vec2 = c(3,4)
vec3 = rbind(vec1,vec2)
```

```
vec1=c(1,3)
vec2=c(2,4)
vec3=cbind(vec1,vec2)
```

Example 2:

Create a new column in dd_dev_transcriptome.txt, equal to the change in expression between hours 0 and hours 4 (cols 3 and 4, respectively)

```
# Take the difference
change_0to4hr = data$tp.hr04 – data$tp.hr00

# Append new column
data = cbind(data, change_0to4hr)
```

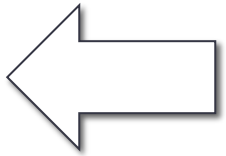
For Loops

Syntax:

```
for (values)  { statements }
```

- Curly braces can go across lines (‘code blocks’)
- Different indentation styles, e.g.:

```
for  ( values ) {
    sample( ...)
    plot( ... )
}
```



Note: You can make plots inside of loops!

```
for (i in 1:10) {
    print(i)
}
```

apply(), sapply() – syntax to avoid loops

```
apply(in_data, dimension, function)
sapply(in_data, function)
```

For example:

```
apply(variablename,1,mean)
```

calculates the mean value of each row (dimension =1) in the data frame "variablename"

```
apply(variablename,2,mean)
```

calculates the mean value of each column (dimension=2) in the data frame "variablename"

sapply (cont'd)

Example 2:

```
> data=read.table('dd_dev_transcriptome.txt', header=TRUE)

> sapply(split(data$tp.hr00, data$type), mean)
         both           dev not_expressed           veg
    5.2186175     1.5525400     0.8391356     4.2730733
```

If your function is more complicated, you can define it on-the-fly from within `sapply` using "`function (..)`"

For example, calculate square of values 1 through 10:

```
> sapply(1:10, function (x) x^2)
 [1]   1   4   9  16  25  36  49  64  81 100
```

by() Calculate a function on data in groups

Syntax

```
by(data, grouping_var, function)
```

Column representing a categorical variable.

```
> by(data$tp.hr00, data$type, mean)
[1] 1994.055
------------------------------------------------------------
data$type: dev
[1] 9.29533
------------------------------------------------------------
data$type: not_expressed
[1] 5.2174
------------------------------------------------------------
data$type: veg
[1] 151.6741
```

by() (cont'd)

Again, if your function is not pre-existing, define it at the time, using function (x):

```
> by(data$tp.hr00, data$type,function(x) mean(log(x+1)))
data$type: both
[1] 5.218617
------------------------------------------------------------
data$type: dev
[1] 1.55254
------------------------------------------------------------
data$type: not_expressed
[1] 0.8391356
------------------------------------------------------------
data$type: veg
[1] 4.273073
```

Or, define it and then use it:

```
> myfunc = function(x) mean(log(x+1))
> by(data$tp.hr00, data$type, myfunc)
```

Writing functions

Syntax:

```
myfunction <- function(arg1, arg2, ... ){
      statements
      return(object)
}
```

See QuickR for more:
http://www.statmethods.net/management/userfunctions.html

Plotting in R

- Refer to a website (like Quick-R) until you grow accustomed to the syntax.

- Basic plotting command is

```
plot()
```

- However, many other plot functions for more specific tasks – for example:

```
barplot()
boxplot()
hist()
heatmap()
```

Simple Plots

The main plot command is `plot()`, but `par()` is used for additional control over appearance

Attributes set using `par()` will be in effect for the rest of the R session (or until you change its values again.)

Alternatively, these same arguments inside `plot()` will limit their scope to that one plot.

For example:

```
x=c(1,2,3,4)
y=c(2,6,5,8)
par(col='red)
plot(x,y)
```

Every plot in this session will be red.

For example:

```
x=c(1,2,3,4)
y=c(2,6,5,8)
plot(x,y,col='red')
```

This plot will be red. Other plots will be in the default color (black).

Easiest to see by example

- Plotting exercises in “plotting_in_r.R”
- We will learn to:
  - Make a single plot
  - Change attributes on the plot: title, axes labels, scale, colors, etc.
  - Make many plots, using variables and a loop.
    - We will save these plots to a single file
    - To many files
  - Overlay multiple plots (necessary to plot in a loop)
  - Place multiple plots on a single page as panels

Colors in R

Source:
http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

color	name	color	name
	white		burlywood4
	aliceblue		cadetblue
	antiquewhite		cadetblue1
	antiquewhite1		cadetblue2
	antiquewhite2		cadetblue3
	antiquewhite3		cadetblue4
	antiquewhite4		chartreuse
	aquamarine		chartreuse1
	aquamarine1		chartreuse2
	aquamarine2		chartreuse3
	aquamarine3		chartreuse4
	aquamarine4		chocolate
	azure		chocolate1
	azure1		chocolate2
	azure2		chocolate3
	azure3		chocolate4
	azure4		coral
	beige		coral1
	bisque		coral2
	bisque1		coral3
	bisque2		coral4
	bisque3		cornflowerblue
	bisque4		cornsilk
	black		cornsilk1
	blanchedalmond		cornsilk2
	blue		cornsilk3
	blue1		cornsilk4
	blue2		cyan
	blue3		cyan1
	blue4		cyan2
	blueviolet		cyan3
	brown		cyan4
	brown1		darkblue
	brown2		darkcyan
	brown3		darkgoldenrod
	brown4		darkgoldenrod1
	burlywood		darkgoldenrod2

Often specified with "`col = `" argument to a plotting function

Colors can specified in many ways, including by:

(1) Name (e.g., see left)

```
col = 'chocolate2'
```

(2) Rgb values

```
col = rgb(0,1,0,1)
```

First three values specify red,green, and blue, respectively. The last value is the degree of transparency.

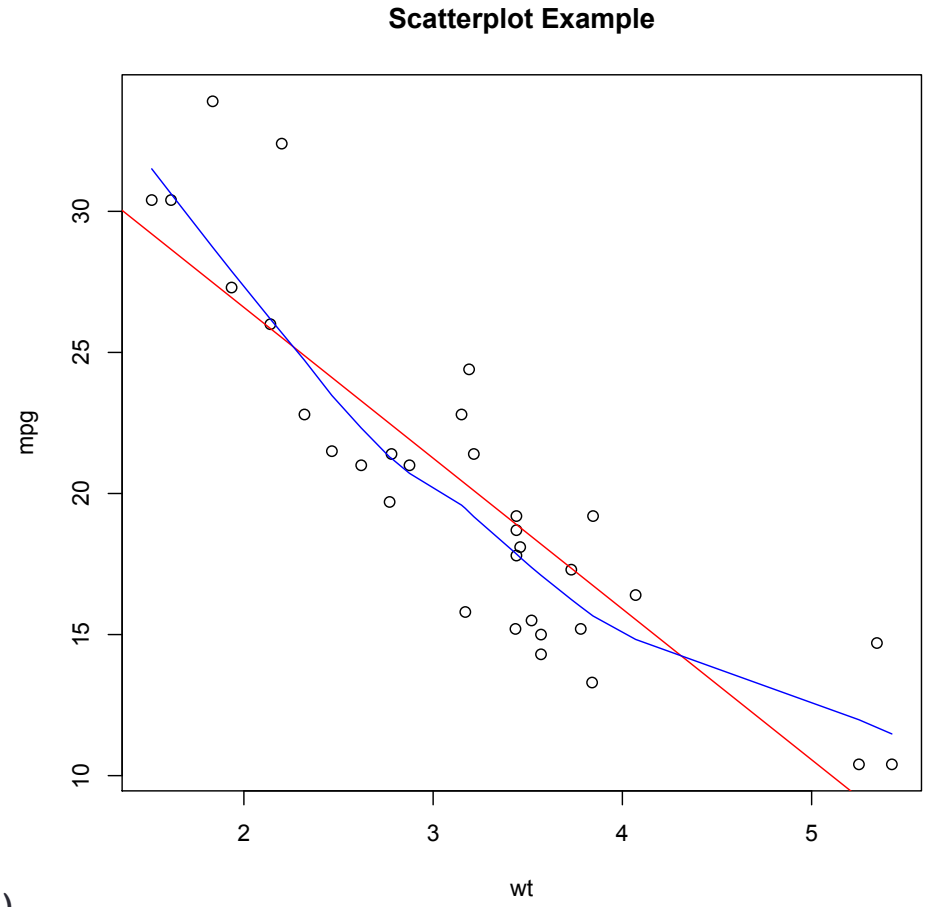
(3) Hexadecimal code

```
col = "#FF0099"
```

(4) Index into a color palette

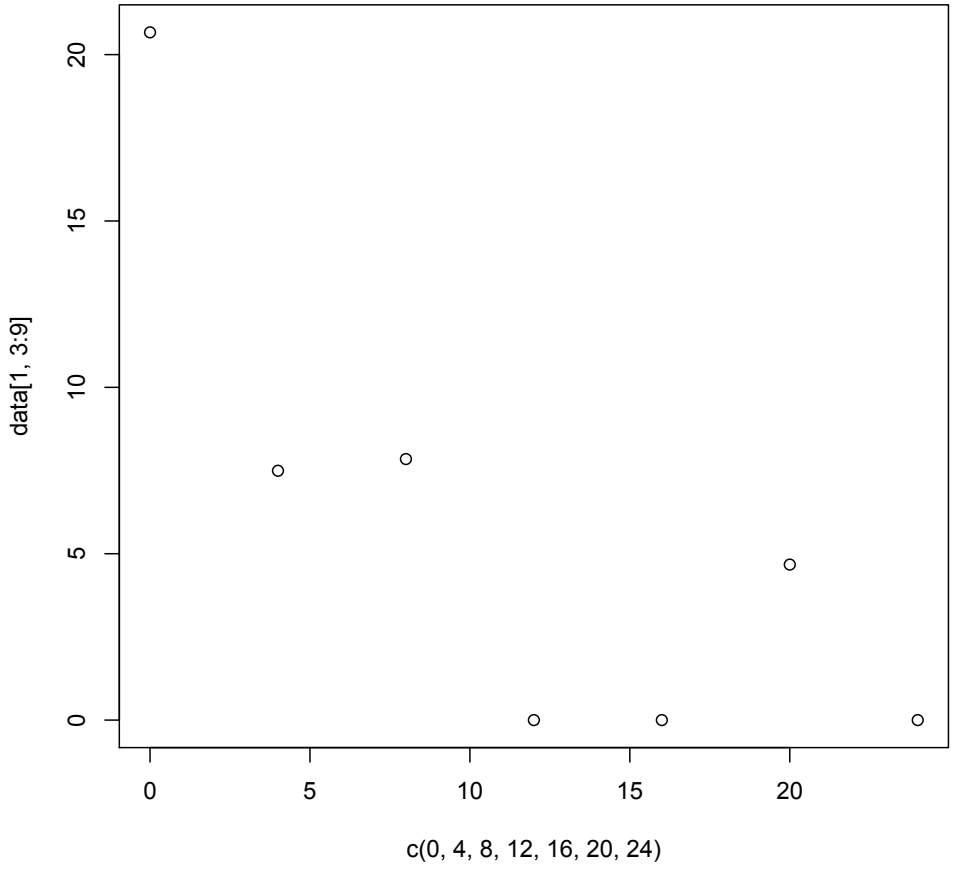
```
colors()[1] (first item in colors)
rainbow(30)[1] (first item in rainbow of 30 shades)
terrain.colors(50)[10]
```

Example from Quick-R:
http://www.statmethods.net/graphs/scatterplot.html




```
attach(mtcars)
plot(wt,mpg,main="Scatterplot Example")
abline(lm(mpg~wt), col="red")              # regression line (y~x)
lines(lowess(wt,mpg), col="blue")          # lowess line (x,y)
```

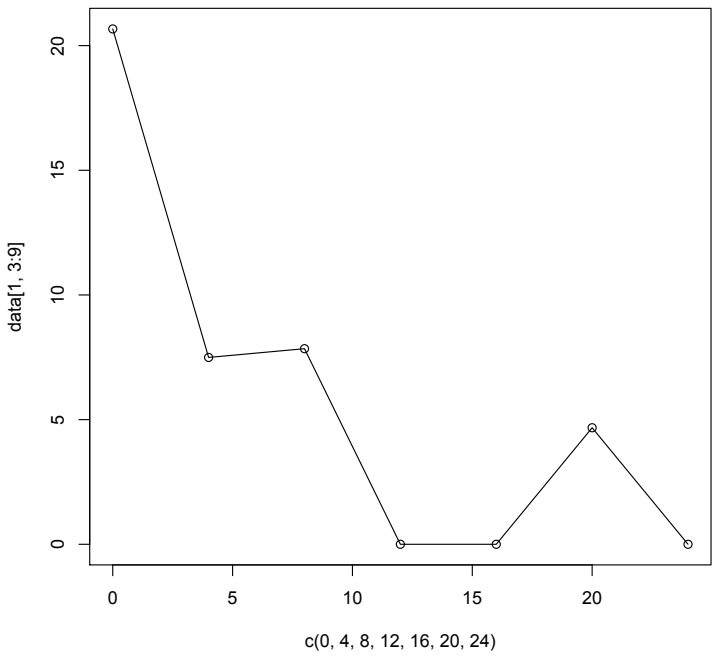
Plotting gene expression time course for one gene:

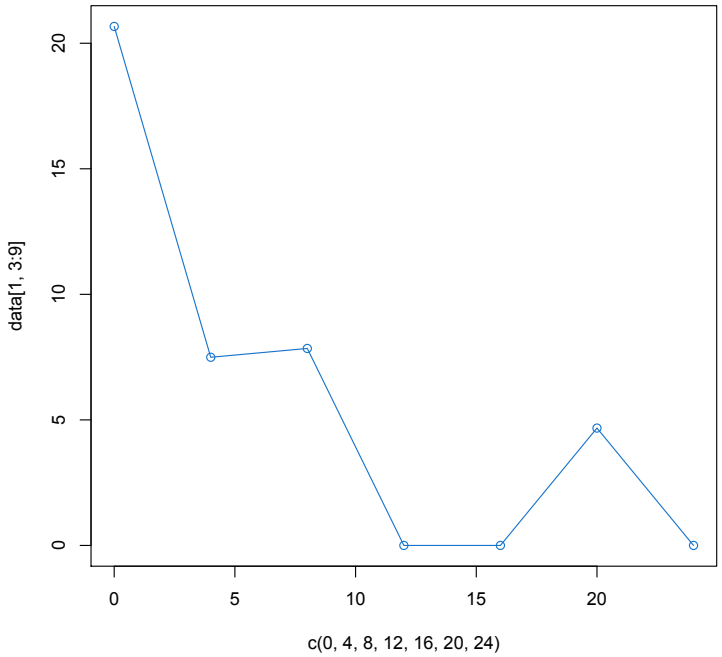



```
infile = ‘dd_dev_transcriptome.txt'
data = read.table(infile, header=TRUE)
summary(data)
plot(c(0,4,8,12,16,20,24),data[1,3:9])
```

```
# Same, but change to a line plot
plot(c(0,4,8,12,16,20,24),data[1,3:9],type='o')


# Same, but make it a nice color
plot(c(0,4,8,12,16,20,24),data[1,3:9],type='o',col='dodgerblue3')
```





```
# Change the x-axis and y-axis labels
plot(c(0,4,8,12,16,20,24),data[1,3:9],type='o',col='dodgerblue3', xlab='Timepoint
(hrs)', ylab='Normalized Number of Reads' )
```