Intro to R for Gene Expression Analysis

July 19, 2023

General Introduction - R and RStudio

- What is R?
  - A programming, ‘scripting’ language
- Why use R?
  - Community
  - Free, open-source
  - Useful for statistics and data visualization
- How to use R?
  - We will use RStudio, a user-friendly interface

Install R Studio

- https://www.rstudio.com/products/rstudio/download/#download

- sudo dpkg -i rstudio-2021.09.0-351-amd64.deb

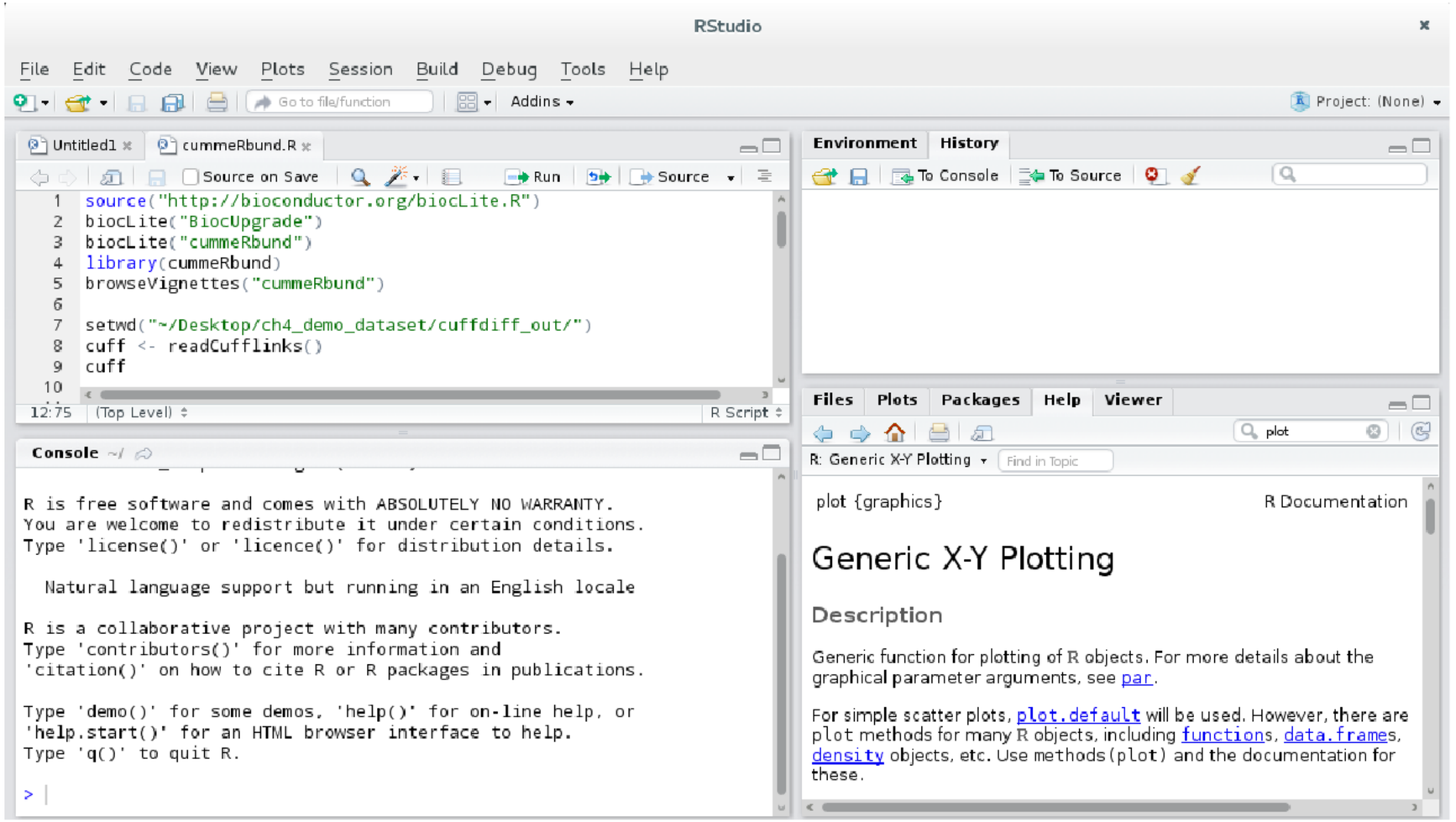
Command line R

- run from a terminal

```
bioinfo@biodebian:~$ R

R version 3.2.5 (2016-04-14) -- "Very, Very Secure Dishes"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> q()
Save workspace image? [y/n/c]: y
bioinfo@biodebian:~$
```

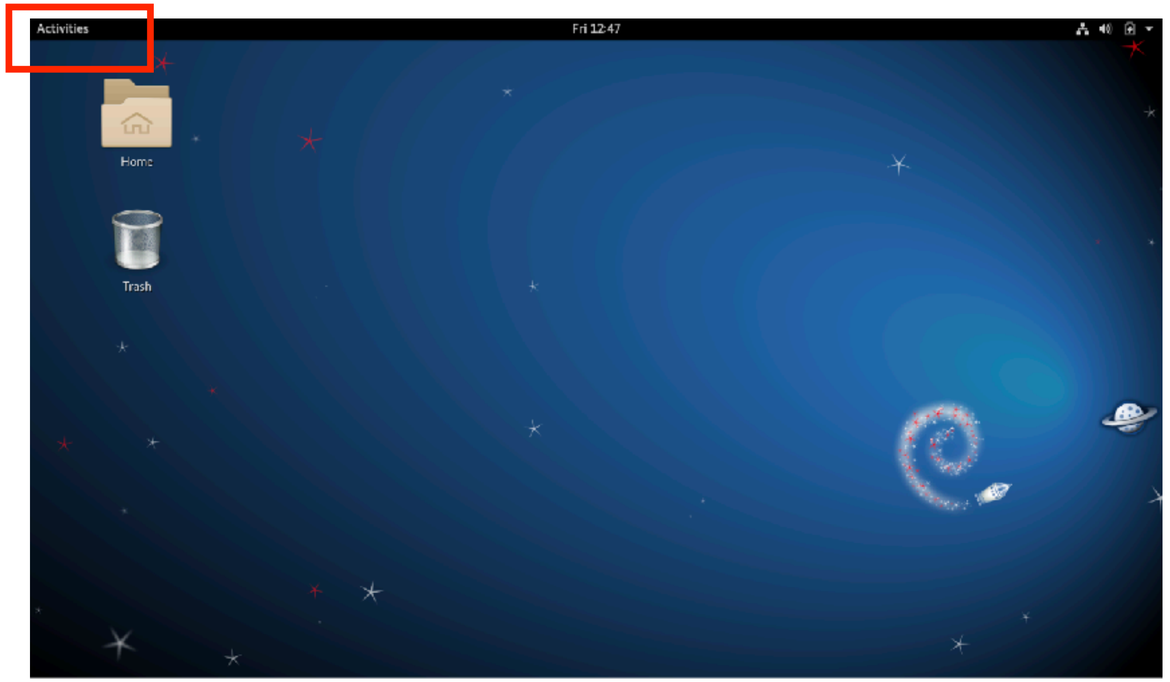
RStudio

- R development environment, GUI, need to install R first.



General Introduction - R and

- Let’s open RStudio

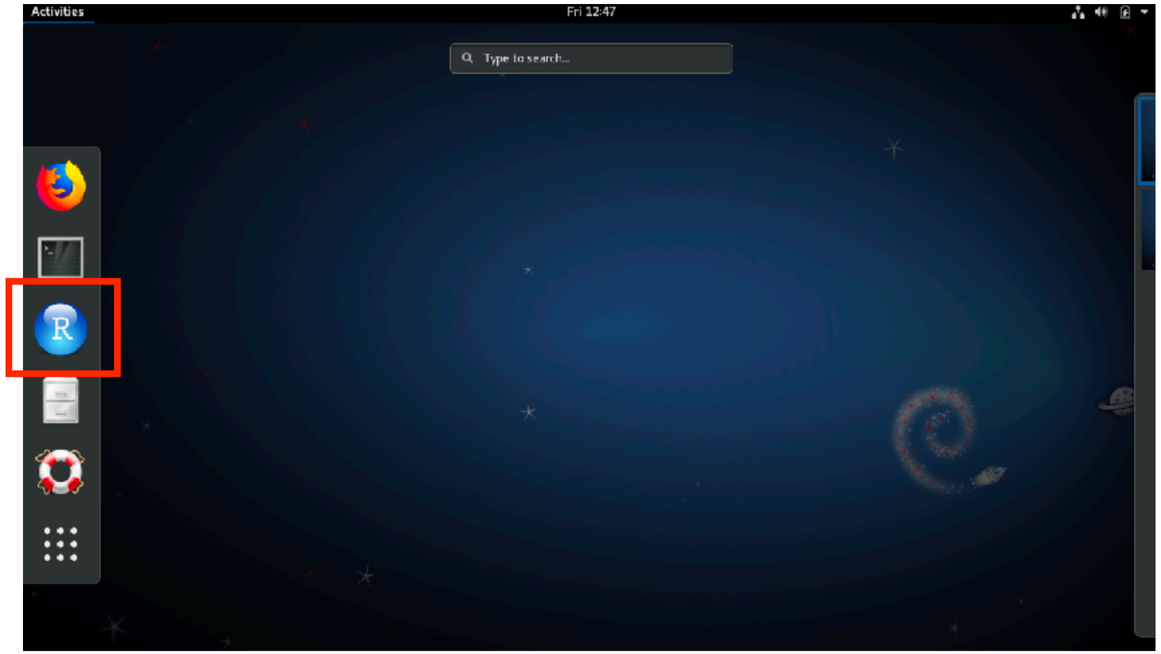


General Introduction - R and

- Let’s open RStudio

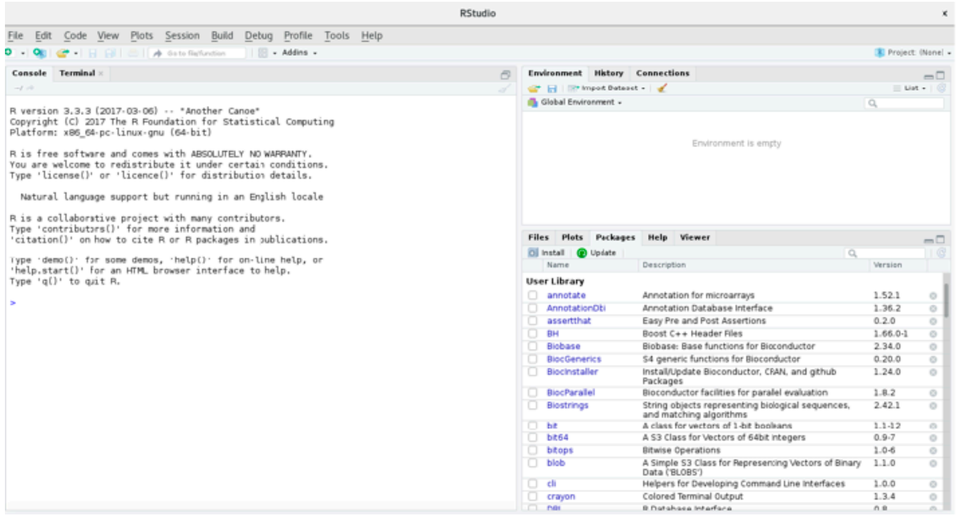
General Introduction - R and

- Let’s open RStudio



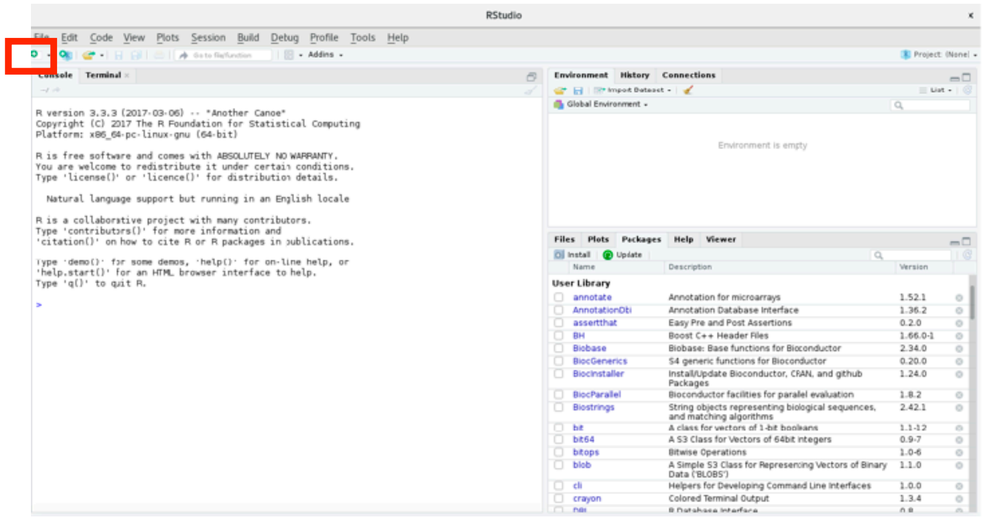
General Introduction - R and

- Getting oriented in RStudio



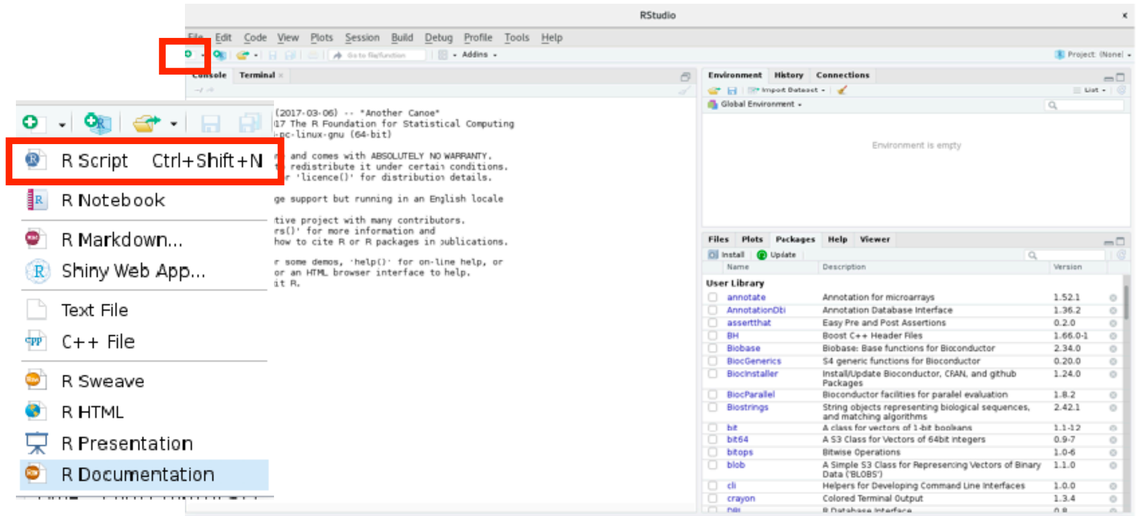
General Introduction - R and

- Getting oriented in RStudio



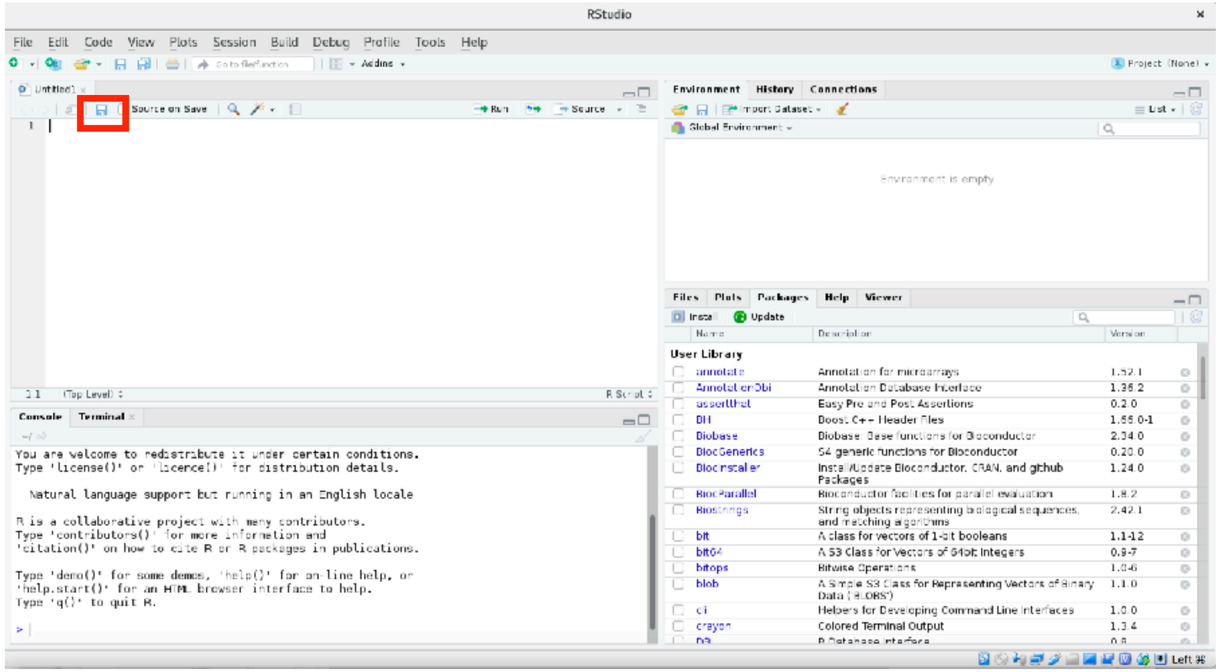
General Introduction - R and

- Getting oriented in RStudio



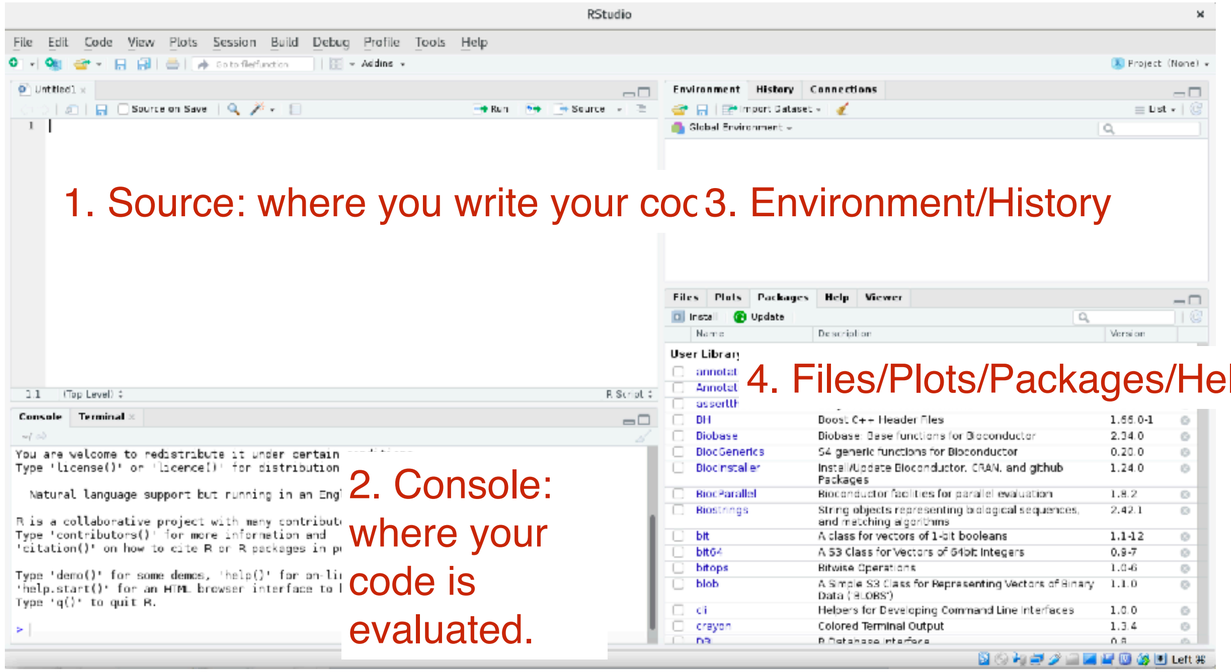
General Introduction - R and

- Getting oriented in RStudio



General Introduction - R and

- Getting oriented in RStudio



Location & Working Directory

To see your working directory (similar to pwd in Terminal):

```
getwd()
```

To set your working directory (similar to cd in Terminal):

```
setwd("./")
```

To list what is in the directory (similar to ls in Terminal):

```
dir()
```

Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

```
56+29
```

Multiplication:

```
3*12
```

Division:

```
40/8
```

Exponents:

```
2^12
```

Try to calculate the following:

```
( (8 / 4) - 4.8 + (12 / (2+1) ) )^3
```

Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

```
56+29
```

Multiplication:

```
3*12
```

Division:

```
40/8
```

Exponents:

```
2^12
```

Try to calculate the following:

```
( (8 / 4) - 4.8 + (12 / (2+1) ) )^3
```

1.728

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

```
number.of.round.progeny.peas <- number.of.total.progeny.peas * (1 - ratio.wrinkled)

number.of.round.progeny.peas
```

Scalar Variables & Data Types

- A few tips
  - Choose meaningful variable names
  - Separate words using dots or underscores (no spaces allowed)
  - Variable names are case-sensitive

Scalar Variables & Data Types

We can use different data types in our variables. To check the type of a variable, use class():

Numeric

```
my_lucky_number <- 7

class(my_lucky_number)

class(number.of.round.progeny.peas)
```

Character

```
cool_fruit <- "tomato"

class(cool_fruit)
```

Scalar Variables & Data Types

Logical

```
my_name_is_Adrian <- TRUE

class(my_name_is_Adrian)
```

Scalar Variables & Data Types

Logical

```
my_name_is_Adrian <- TRUE

class(my_name_is_Adrian)
```

What would class() tell you for the following?

```
tomato_is_the_best_fruit <- "TRUE"

class(tomato_is_the_best_fruit)
```

Functions

- A function is basically just a set of instructions
  - Takes some input
  - Does some things
  - Gives an output

- We can access functions in a few ways…

Functions

- We can define our own custom functions

Let's define and use a simple function:

```
my_awesome_function <- function(input1){
  output <- input1^12 * 5 + 3
  return(output)
}

my_awesome_function(3)

my_awesome_function(5)

my_awesome_new_variable <- my_awesome_function(7)

my_awesome_new_variable
```

Functions

- R also has ‘pre-made’ functions… can you think of any examples that we’ve used?

Functions

- R also has ‘pre-made’ functions… can you think of any examples that we’ve used?

To access R documentation for a function, etc.:

```
?class

?plot
```

Functions

- We can also install packages and load libraries to access sets of additional functions…

Packages

- Let’s try installing/loading a package called ggplot2:

```
install.packages("ggplot2")

library(ggplot2)
```

Data Structures

- Vectors
- Matrices
- Data frames
- Lists

Data Structures: Vectors

```
vector1 <- c(1,2,5.3,6,-2,4)
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector2 <- c("one","two","three")
vector2
```

```
## [1] "one"   "two"   "three"
```

```
vector3 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
vector3
```

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

```
vector4 <- rep(1:3, each=2)
vector4
```

```
## [1] 1 1 2 2 3 3
```

Data Structures: Vectors

```
vector5 <- rep(c(1, 0, 7), times = 5) #repeats c(1,0,7) vector 5 times
vector5
```

```
##  [1] 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7
```

```
vector6 <- rep(c(0, 4), times = c(5,3)) # you could tell R how many times to repeat each value.
vector6
```

```
## [1] 0 0 0 0 0 4 4 4
```

```
vector7 <- rep(1:4,length.out=6) # you could also repeat the vector to the specifield length even if th
vector7
```

```
## [1] 1 2 3 4 1 2
```

```
vector8 <- seq(from = 7.5, to = 2.5, by = -0.5) # vector with a numbers 7.5 to 2.5 in steps of 0.5.
vector8
```

```
##  [1] 7.5 7.0 6.5 6.0 5.5 5.0 4.5 4.0 3.5 3.0 2.5
```

Data Structures: Vectors

We can transpose a vector:

```
t(vector1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2  5.3    6   -2    4
```

We can access elements in a vector using the element's vector position:

```
vector1[c(2,4)]
```

```
## [1] 2 6
```

We can add, subtract, multiply, divide, square root, and exponentiate vectors and scalars of numeric class. These can be done using operations:

```
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector7
```

```
## [1] 1 2 3 4 1 2
```

```
vector1 + vector7
```

```
## [1]  2.0  4.0  8.3 10.0 -1.0  6.0
```

Data Structures: Vectors

We can apply functions on vectors:

```
mean(vector1)
```

```
## [1] 2.716667
```

```
median(vector1)
```

```
## [1] 3
```

Data Structures: Matrices

We can create matrix using matrix(c(), byrow=, ncol=). Matrices can mix element class types.

```
matrix1 <- matrix(c(1,3,2,2,8,9), ncol=2)
matrix1
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    8
## [3,]    2    9
```

```
matrix2 <- matrix(c(1,3,2,2,8,9), ncol=2, byrow = T)
matrix2
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    2
## [3,]    8    9
```

Data Structures: Matrices

We can access elements in matrix using element position:

```
c(matrix1[1,2], matrix1[2,2],matrix1[3,1])
```

```
## [1] 2 8 2
```

Matrix operations (matrix multiplication, etc.) can also be performed.

Transpose of matrices (It is basically to interchange of rows and columns:

```
t(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    2    8    9
```

We can also create vectors from a matrix:

```
matrix1[2, ]
```

```
## [1] 3 8
```

```
matrix1[ ,2]
```

```
## [1] 2 8 9
```

Data Structures: Data Frames

Data frames are used for storing data tables. It is usually a list of equal vectors of the same length.

```
c <- c(2, 3, 5)
d <- c("aa", "bb", "cc")
e <- c(TRUE, FALSE, TRUE)
df <- data.frame(c, d, e)
colnames(df) <- c("numeric", "character", "logical") # Add column names
df
```

```
##   numeric character logical
## 1       2        aa    TRUE
## 2       3        bb   FALSE
## 3       5        cc    TRUE
```

Reading Data into R

- Use functions such as:
  - read.table()
  - read.csv()

R Package Repos

- CRAN

```
> install.packages("fortunes")
```

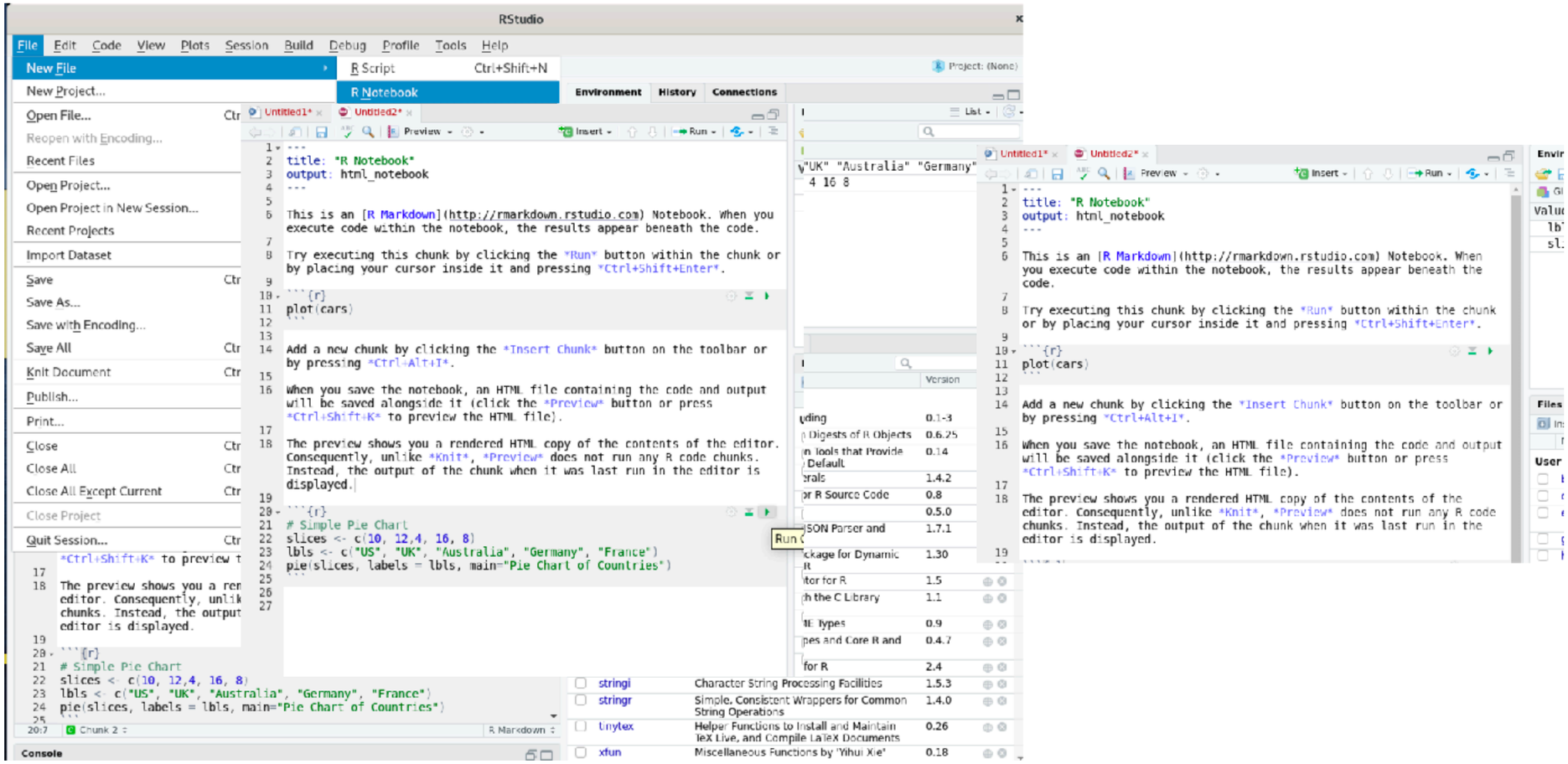
- Bioconductor

```
source("https://bioconductor.org/biocLite.R")
biocLite("S4Vectors")
```

- GitHub

```
library(githubinstall)
githubinstall("AnomalyDetection")
```

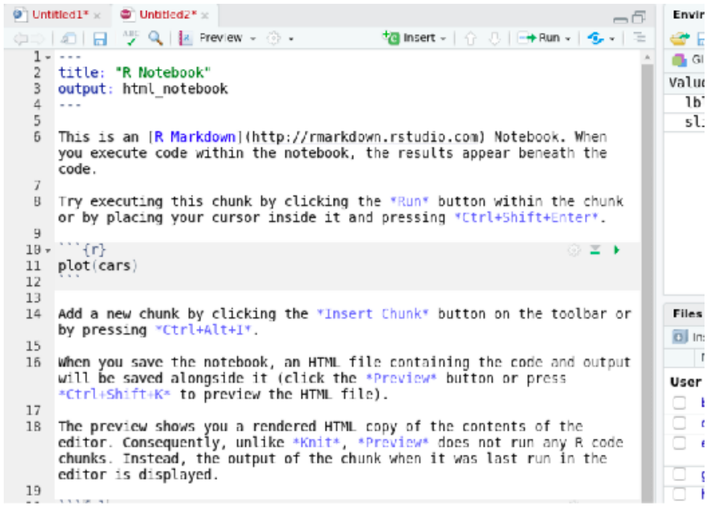
R Markdown

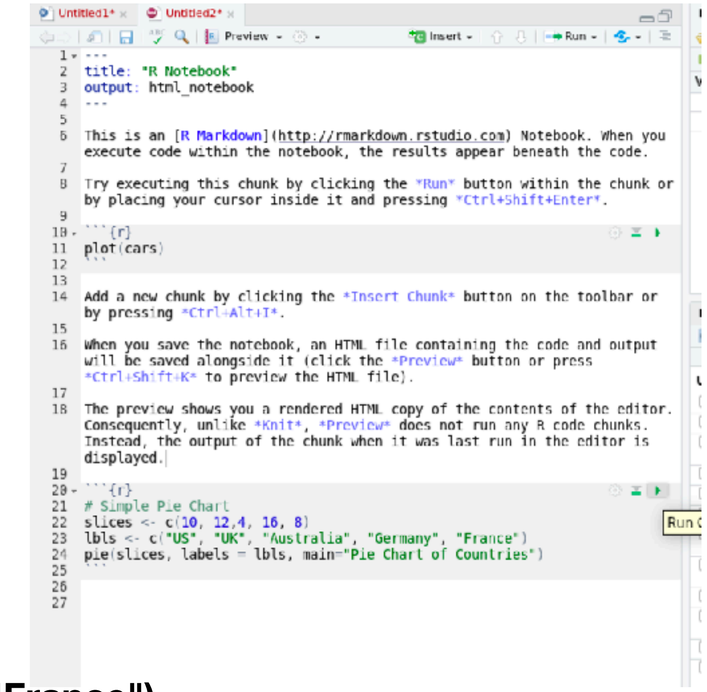

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
New File
New Project...
Open File...
Reopen with Encoding...
Recent Files
Open Project...
Open Project in New Session...
Recent Projects
Import Dataset
Save
Save As...
Save with Encoding...
Save All
Knit Document
Publish...
Print...
Close
Close All
Close All Except Current
Close Project
Quit Session...
R Script Ctrl+Shift+N
R Notebook
Environment History Connections
Untitled1* Untitled2*
Preview Insert Run
---
title: "R Notebook"
output: html_notebook

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.
Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
```{r}
plot(cars)
```
Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
```{r}
# Simple Pie Chart
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```
Chunk 2
R Markdown
Console
stringi Character String Processing Facilities 1.5.3
stringr Simple, Consistent Wrappers for Common String Operations 1.4.0
tinytex Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents 0.26
xfun Miscellaneous Functions by 'Yihui Xie' 0.18

R Markdown

R notebook

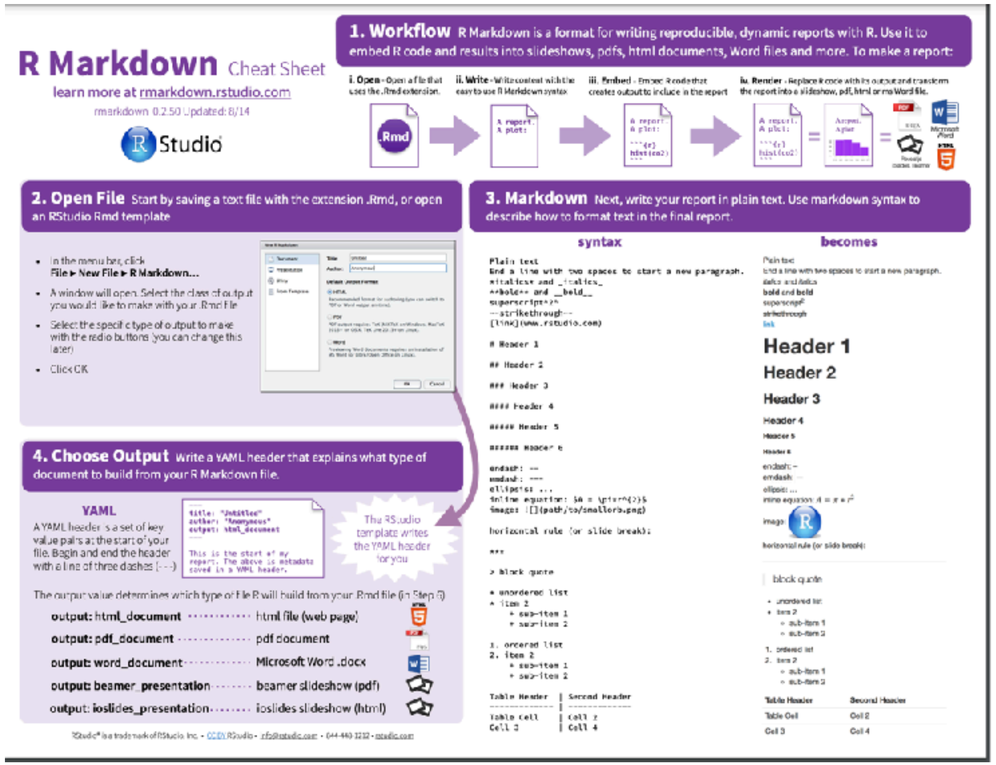





```
# Simple Pie Chart
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```

R Markdown Cheat Sheet

https://rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf

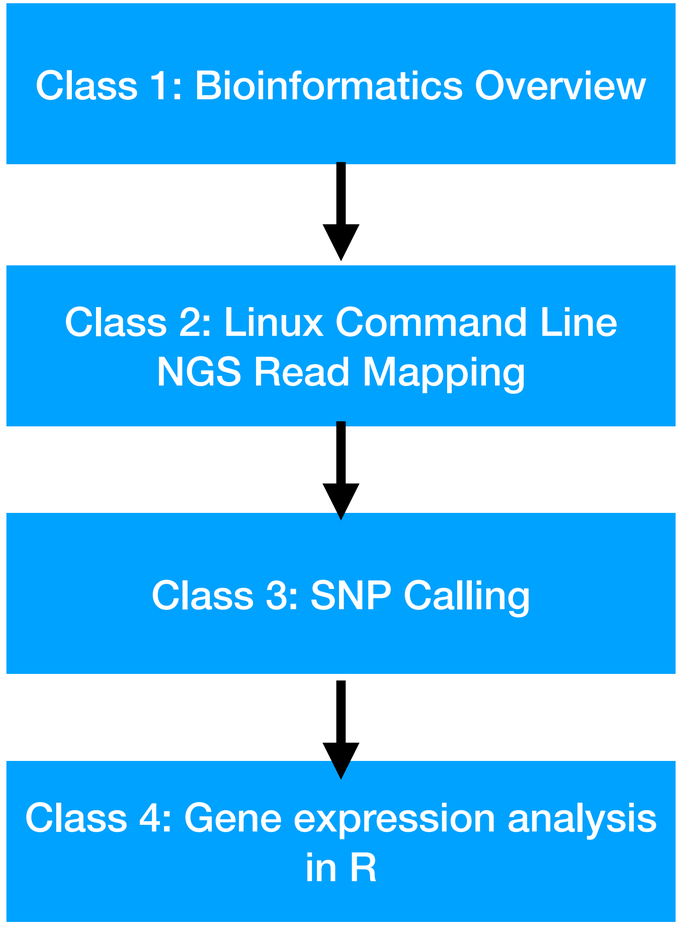


R Resources

- R site: https://cran.r-project.org/manuals.html
- Cheat sheets: https://r-dir.com/reference/crib-sheets.html
- Code school : http://tryr.codeschool.com/
- Books: Bioinformatics Data Skills

Course Overview

Make sure you have gene_counts_matrix.csv for next time!!! :)



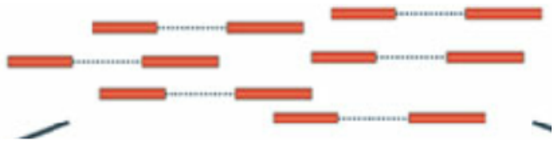
RNA-Seq Analysis
Tools for DE

Sequence

Assemble

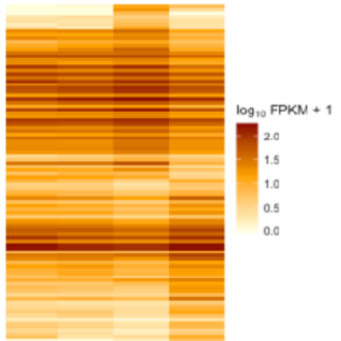
Analysis

Illumina
(today)



Hisat2
(today)

Stringtie
(today)
edgeR
(last class)



Data for following exercises

Two RNA-seq datasets used in the tomato genome project were downloaded from the SRA in .sra format and extracted using the SRA toolkit (NCBI).

http://www.ncbi.nlm.nih.gov/sra

They were already cleaned. All data is from S. pimpinellifolium.

Datasets :

- breaker fruit (two files)
- immature fruit (two files)

In this exercise, we will map the reads to tomato chromosome 4 using reference-guided assembly

Connect to remote server ‘thompson’

- Open terminal or Putty and type your username and password. Example (this is NOT your username):

ssh bioinfo0@thompson.sgn.cornell.edu

- #The password is bioinfo00

edgeR: a Bioconductor package for differential expression analysis of digital gene expression data

Robinson MD, McCarthy DJ and Smyth GK (2010). “edgeR: a Bioconductor package for differential expression analysis of digital gene expression data.” *Bioinformatics*, 26, pp. -1.

http://www.bioconductor.org/packages/release/bioc/vignettes/edgeR/inst/doc/
edgeRUsersGuide.pdf

General pipeline for differential expression analysis

(edgeR uses negative binomial distribution)

- **Filtering**
- **Normalization**
- **Dispersion estimation**
- **Hypothesis testing**

Datasets - immature fruit (2 biological replicates)
SRR404331
SRR404333

- breaker fruit (2 biological replicates)
SRR404334
SRR404336

- Loading edgeR : library(“edgeR”)

- Setting working directory:
  setwd("/home/bioinfo/Desktop/Slch04_demo")

- Importing data into R:

```
tomct <- read.csv("gene_count_matrix.csv", row.names=1,
    header=TRUE)
```

- Grouping:

```
my_group <- (c(“im”, “im”, “br”, “br”))
```

edgeR stores RNA-seq data as a DGEList object
(data structure = list)

- Putting data into a DGEList object:

```
tom_dgl <- DGEList(counts=tomct, group=my_group)
```

- Checking the DGEList

```
tom_dgl
```

$counts: gene names, gene counts for each library

$samples: library names, groups, lib.size, norm.factors

> tom_dgl

An object of class "DGEList"

$counts

	SRR404331_ch4.sort	SRR404333_ch4.sort	SRR404334_ch4.sort	SRR404336_ch4.sort
gene:Solyc10g054820.2	0	0	0	0
gene:Solyc12g098195.1	0	0	0	0
gene:Solyc10g046810.1	0	0	0	0
gene:Solyc02g062000.3	0	0	0	0
gene:Solyc07g019650.3	0	0	0	0

34874 more rows ...

$samples

	group	lib.size	norm.factors
SRR404331_ch4.sort	im	468307	1
SRR404333_ch4.sort	im	398150	1
SRR404334_ch4.sort	br	304826	1
SRR404336_ch4.sort	br	494570	1

- **head(tom_dgl$counts)**

	SRR404331_ch4.sort	SRR404333_ch4.sort	SRR404334_ch4.sort	SRR404336_ch4.sort
gene:Solyc10g054820.2	0	0	0	0
gene:Solyc12g098195.1	0	0	0	0
gene:Solyc10g046810.1	0	0	0	0
gene:Solyc02g062000.3	0	0	0	0
gene:Solyc07g019650.3	0	0	0	0
gene:Solyc01g013760.1	0	0	0	0

- **dim(tom_dgl$counts)**

  34879 4

- **tom_dgl$samples**

	group	lib.size	norm.factors
SRR404331_ch4.sort	im	468307	1
SRR404333_ch4.sort	im	398150	1
SRR404334_ch4.sort	br	304826	1
SRR404336_ch4.sort	br	494570	1

BTI

Data processing

- **Saving a copy of raw data before data processing**

  tom_dgl.rawdata <- tom_dgl

- **Filtering to remove very low counts:**

  If you want to keep genes with more than 1 CPM, in at least 2 samples

  keep <- rowSums(cpm(tom_dgl)>1) >= 2

  (TRUE=1, FALSE=0)

  (Question: Is 1 count per million suitable for our data?)

- **Checking filtering step**

  table(keep)

  FALSE TRUE

  33286 1593

BTI

- **Modifying DGEList**

  `tom_dgl <- tom_dgl[keep, , keep.lib.sizes=FALSE]`

- **Checking filtered data set**

  `dim(tom_dgl)`

  1593 4

  `head(tom_dgl$counts)`

	SRR404331_ch4.sort	SRR404333_ch4.sort	SRR404334_ch4.sort	SRR404336_ch4.sort
gene:Solyc04g050480.3	59	57	38	69
gene:Solyc04g080270.3	1050	1388	869	1138
gene:Solyc04g008310.2	480	256	111	179
gene:Solyc04g074240.3	657	526	373	570
gene:Solyc04g071590.3	0	27	0	7
gene:Solyc04g054840.1	3	0	1	1

- **tom_dgl$samples**

	group	lib.size	norm.factors	
SRR404331_ch4.sort	im	468168	1	(468307)
SRR404333_ch4.sort	im	398023	1	(398150)
SRR404334_ch4.sort	br	304799	1	(304826)
SRR404336_ch4.sort	br	494176	1	(494570)

Normalization

- edgeR

  - uses TMM (Trimmed mean of M-value) method to eliminate RNA composition effect

  - automatically adjusts for difference in library size caused by sequencing depth

  - doesn't adjust for gene length

Normalization

- tom_dgl <- calcNormFactors(tom_dgl)
- tom_dgl$samples

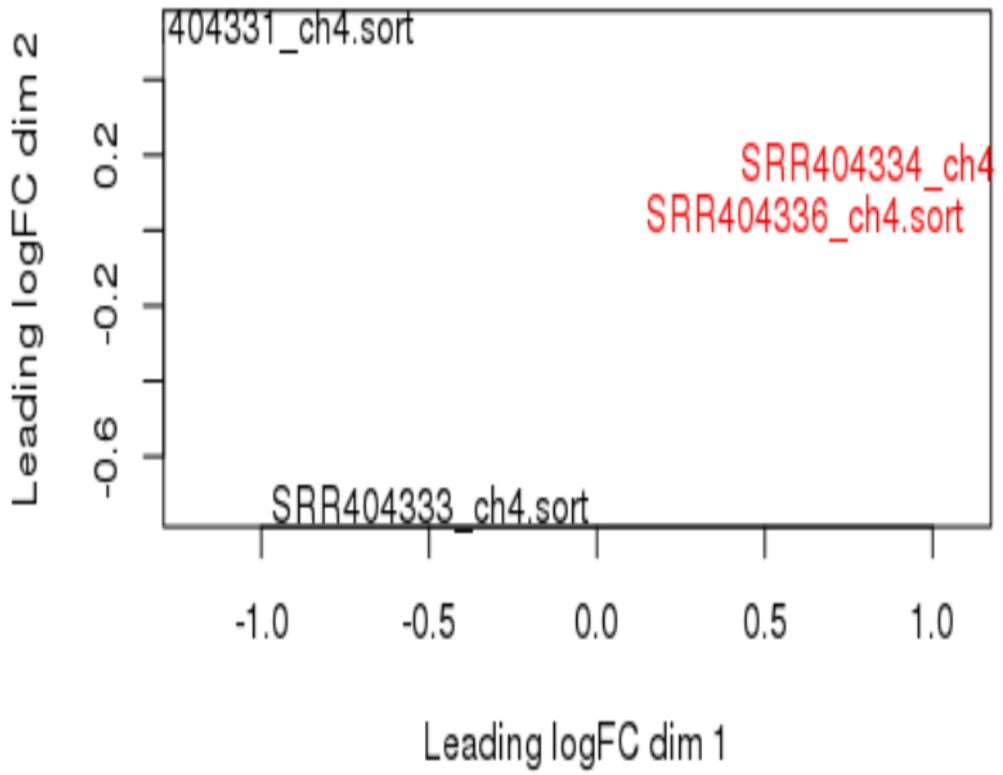
	group	lib.size	norm.factors
SRR404331_ch4.sort	im	468168	0.8929376
SRR404333_ch4.sort	im	398023	1.0082953
SRR404334_ch4.sort	br	304799	1.0647144
SRR404336_ch4.sort	br	494176	1.0431769

BTI

Data Exploration

Multi-dimensional (MDS) plot: plotMDS (y)

plotMDS(tom_dgl, col=as.numeric(tom_dgl$samples$group))



General pipeline

- Filtering
- Normalization
- Dispersion estimation
- Hypothesis testing

edgeR package

- Classic edgeR: testing single factor

  -Exact test

- Generalized linear models (glms): testing multiple factors

  -Likelihood ratio test
  -Quasi-likelihood method

Pairwise comparison (classic edgeR)

- Dispersion estimation

```
tom_dgl <- estimateDisp(tom_dgl)
```

Differential expression analysis

- **Hypothesis testing**
  tom_de <- exactTest(tom_dgl)
- **To display the most significant tags**
  topTags(tom_de, n=10)

Comparison of groups: im-br

	logFC	logCPM	PValue	FDR
gene:Solyc04g074840.3	10.674057	11.661710	5.008535e-101	7.978595e-98
gene:Solyc04g079960.1	4.780808	9.749903	1.003905e-52	7.996107e-50
gene:Solyc04g078460.3	3.010089	9.361145	4.374771e-27	2.323003e-24
gene:Solyc04g076780.3	-3.572084	9.164866	7.345144e-27	2.925203e-24
gene:Solyc04g009960.3	3.129447	10.605411	1.079804e-25	3.440257e-23
gene:Solyc04g071615.1	-2.843780	11.217141	9.526994e-25	2.529417e-22
gene:Solyc04g071650.3	-3.425306	10.313879	2.530218e-24	5.758053e-22
gene:Solyc04g081300.3	-3.222494	10.382392	4.571451e-24	9.102901e-22
gene:Solyc04g079900.3	-2.638129	9.831421	1.733762-23	3.068758e-21
gene:Solyc04g079560.3	-3.836033	8.269721	7.778344e-23	1.239090e-20

- Selecting differentially expressed genes at a FDR of 5%

```
tom_de_05 <- decideTestsDGE(tom_de)

tom_de_05
   TestResults matrix
   0  0 -1  0  0
   1588 more rows ...


summary(tom_de_05)
   -1   142      (down-regulated)
    0  1331
    1   120      (up-regulated)
```

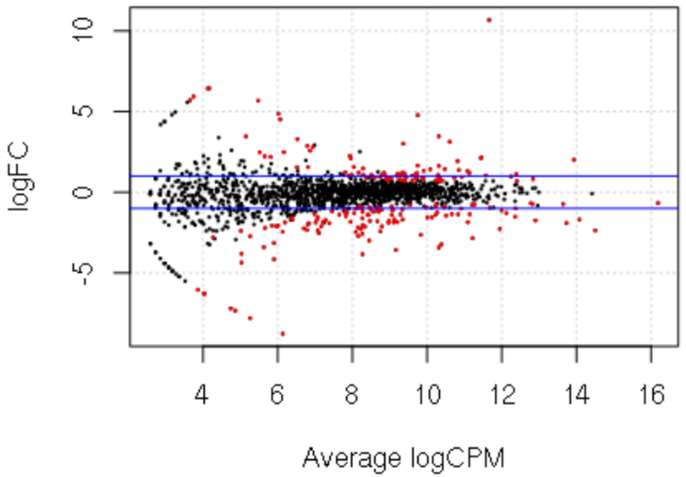
BTI

- Generating a dataframe containing DE genes at a FDR at 5%

```
isDE <- as.logical(tom_de_05)

tom_de_05name <- rownames(tom_dgl)[isDE]

tom_de_05.table <- tom_de[tom_de_05name,]
```

Smear Plot

```
plotSmear(tom_de, de.tags = tom_de_05name)
abline(h = c(-1, 1), col = "blue")
```



The blue lines indicate 2 fold-changes

- **Exporting data**

```
write.csv(tom_de_05.table$table, file="tom_de_05")

write.csv(tom_de$table, file="tom_de")
```

Data Visualization and Manipulation in R

- Building a plot from scratch
- Adding columns to data
- Attaching and detaching data
- Statistical analysis
- The aggregate and split functions

Data Visualization and Manipulation in R

- Download data001_genome_size.csv and data002_gene_expression.tab from the course website (access from inside your VM!)

- Use setwd() to set your working directory to the location of the files data001_genome_size.csv and data002_gene_expression.tab that you downloaded

BCBC

BTI

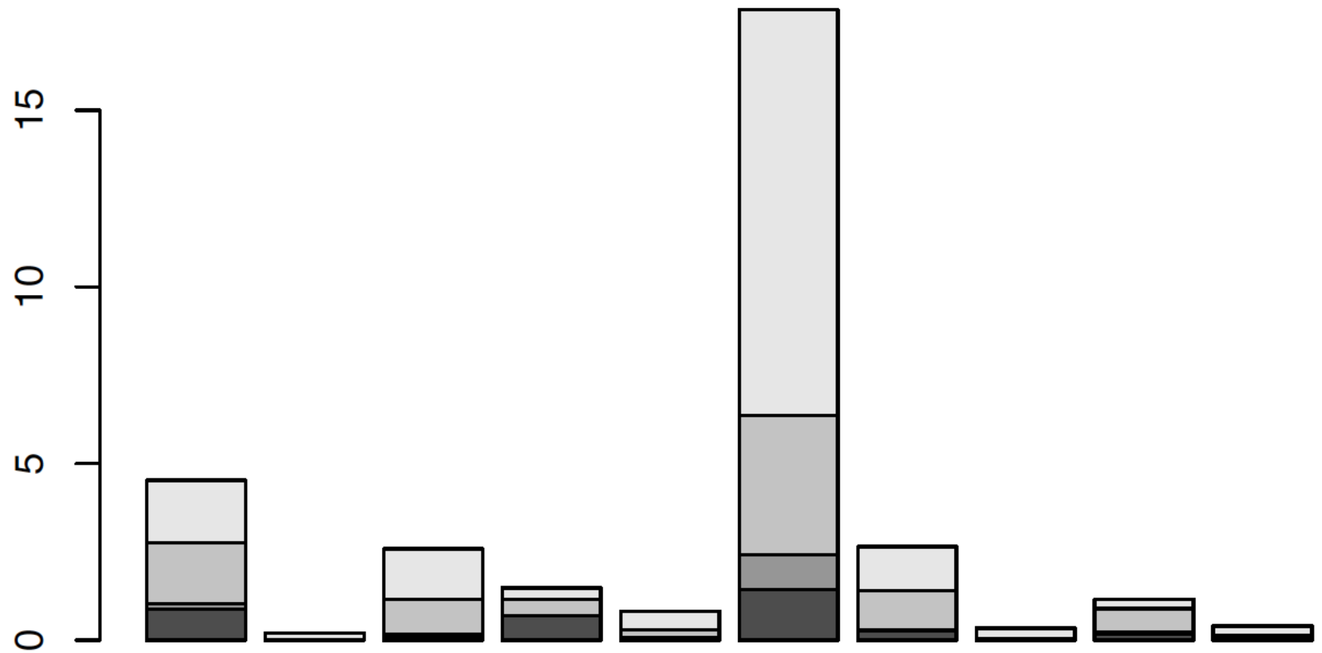
Let's build a plot from scratch!

```
data002_gene_expression <- read.delim("data002_gene_expression.tab")
head(data002_gene_expression, n=10)
```

```
##      gene_short_name EV10_FPKM   EV21_FPKM TS10_FPKM TS21_FPKM
## 1  NbC25469554g0001 0.8703800 0.164763000  1.718770  1.774910
## 2  NbC25730842g0001 0.0000000 0.000639717  0.000000  0.213441
## 3  NbC26298068g0003 0.1169390 0.052417000  0.992478  1.425080
## 4  NbS00000194g0006 0.6909000 0.010203600  0.466495  0.315410
## 5  NbS00000341g0006 0.0563956 0.023237300  0.221736  0.523042
## 6  NbS00000565g0004 1.4402400 0.985119000  3.939010 11.475700
## 7  NbS00001525g0108 0.2755520 0.002241480  1.132370  1.239610
## 8  NbS00001801g0009 0.0298086 0.025351900  0.000000  0.286725
## 9  NbS00002670g0014 0.1909280 0.027332700  0.674384  0.275545
## 10 NbS00002765g0028 0.0131139 0.024502700  0.104305  0.265162
```

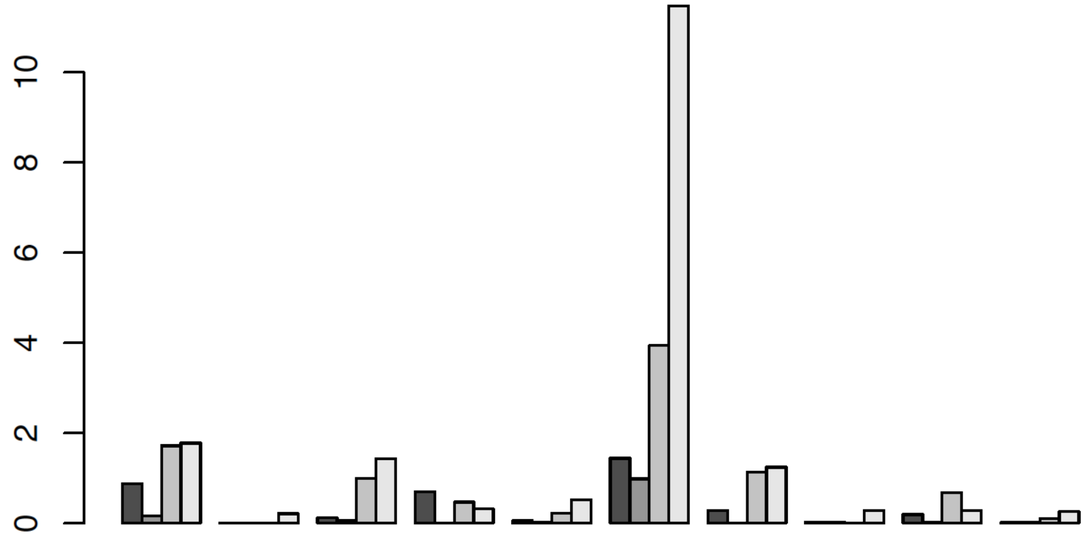
We start with a basic bar plot of gene expression with treatment as categories

```
gene_exp <- as.matrix(data002_gene_expression[,2:5])
barplot(t(gene_exp))
```



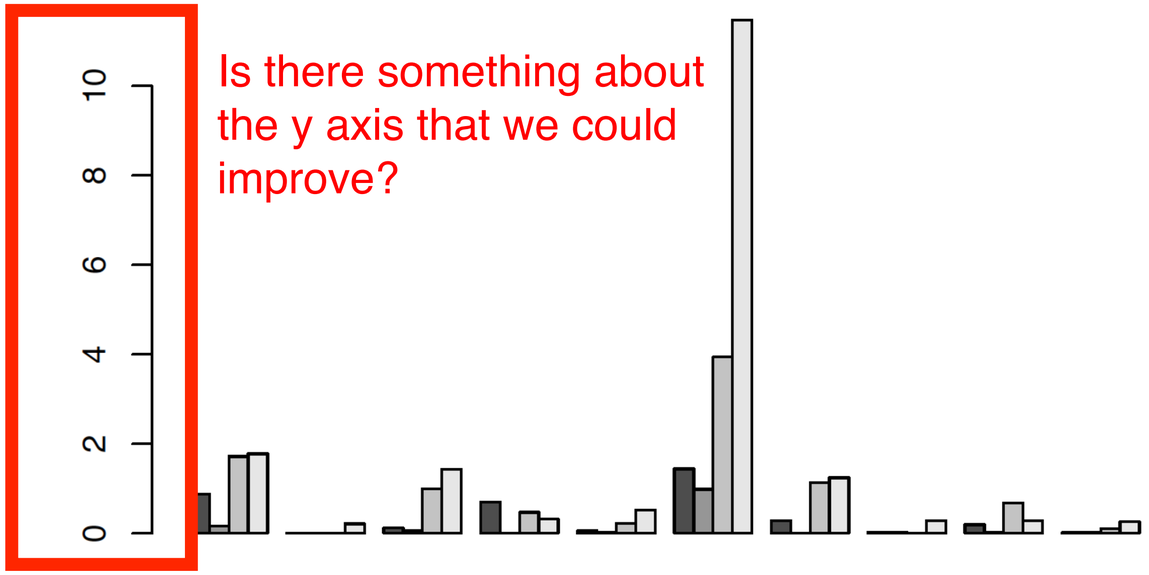
Let's change it so that the gene expression values are not stacked

```
barplot(t(gene_exp), beside=T)
```



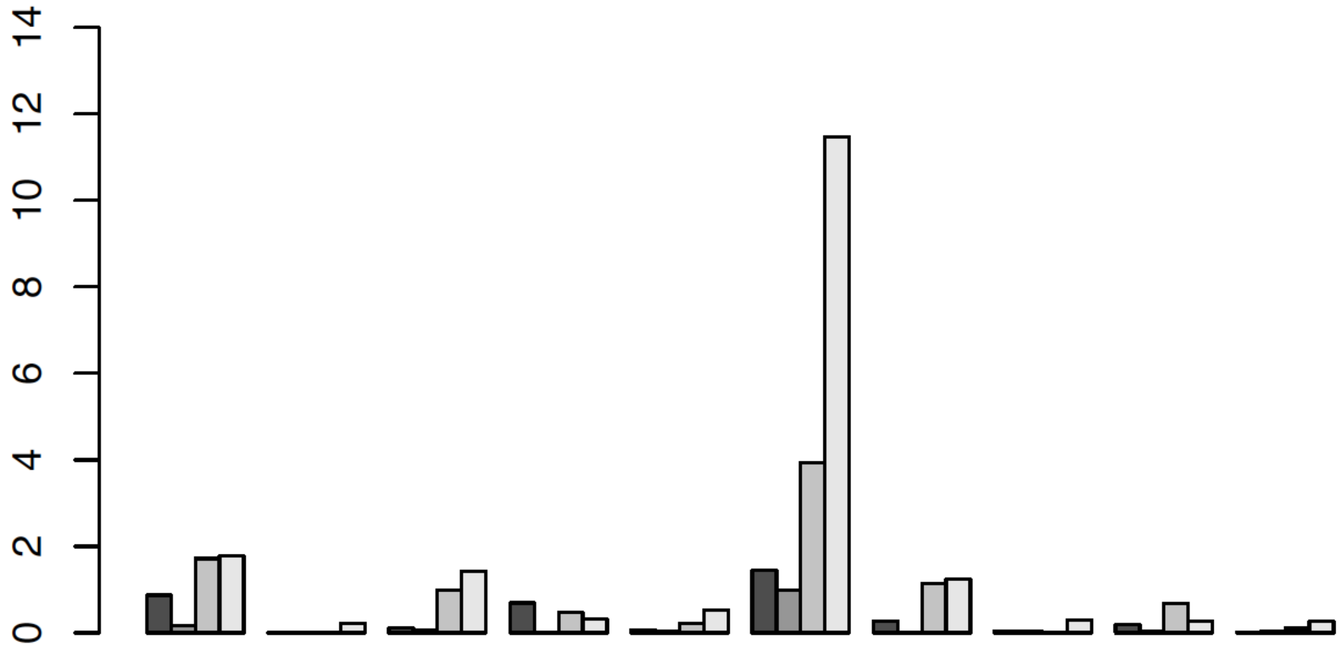
Let's change it so that the gene expression values are not stacked

```
barplot(t(gene_exp), beside=T)
```



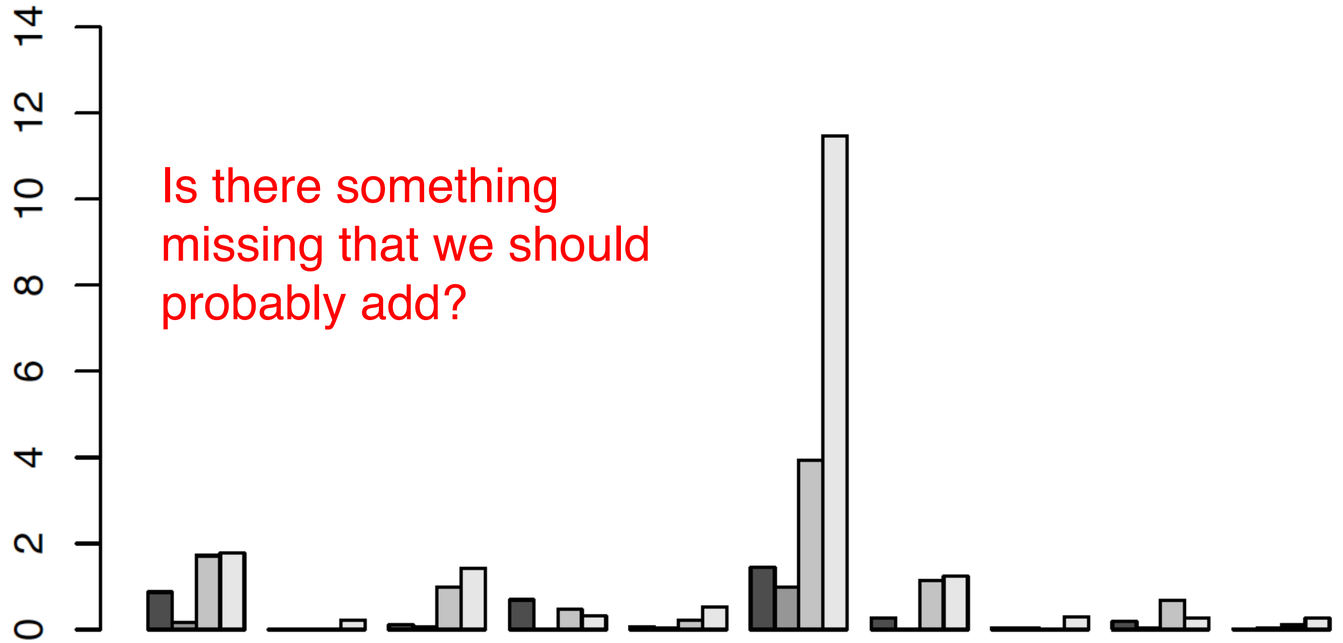
Fixing the y axis limit

```
barplot(t(gene_exp), beside=T, ylim=c(0,15))
```



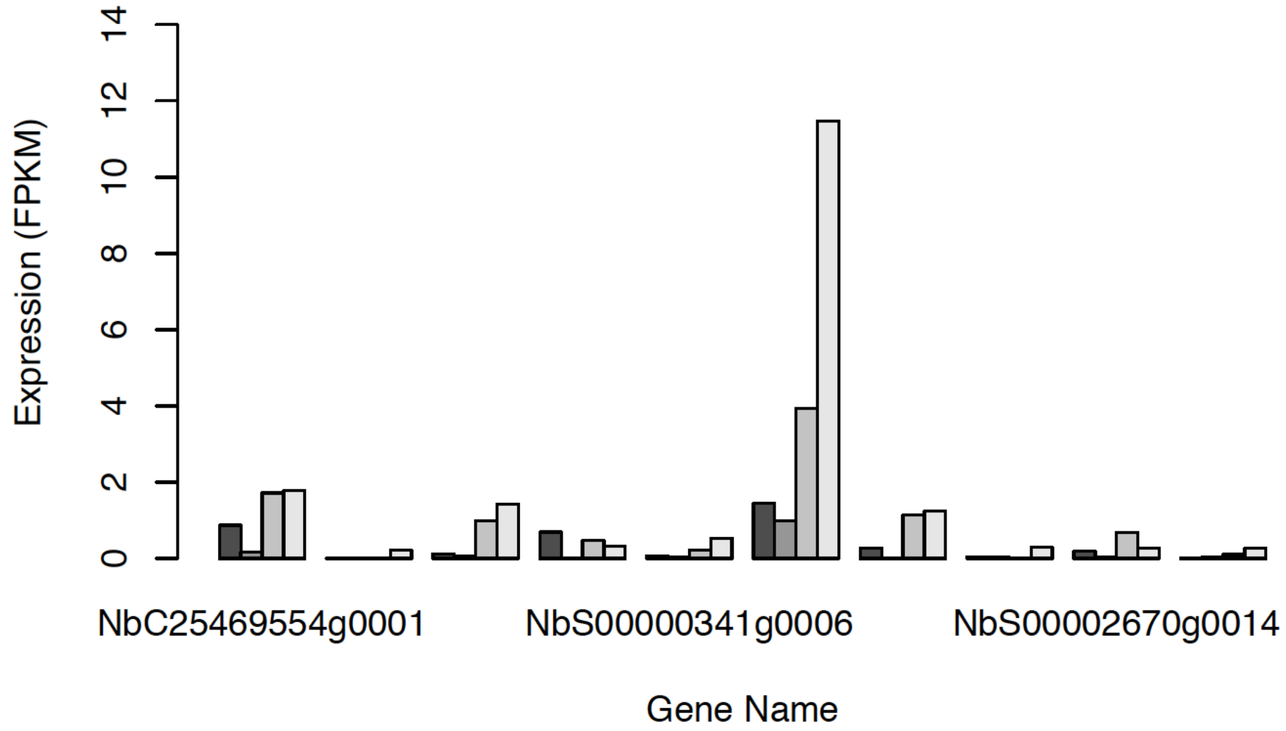
Fixing the y axis limit

```
barplot(t(gene_exp), beside=T, ylim=c(0,15))
```



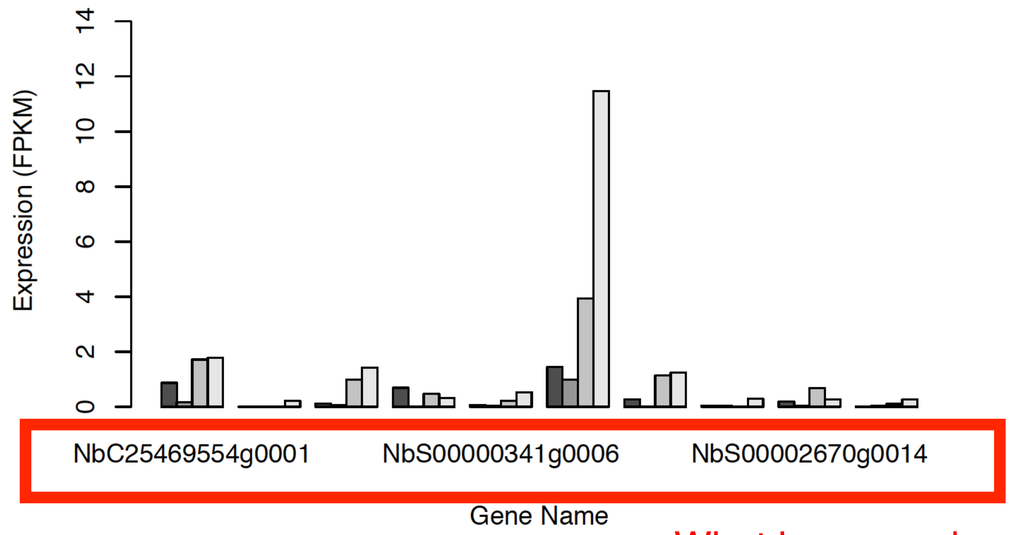
Adding axis labels

```
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, xlab="Gene Name")
```



Adding axis labels

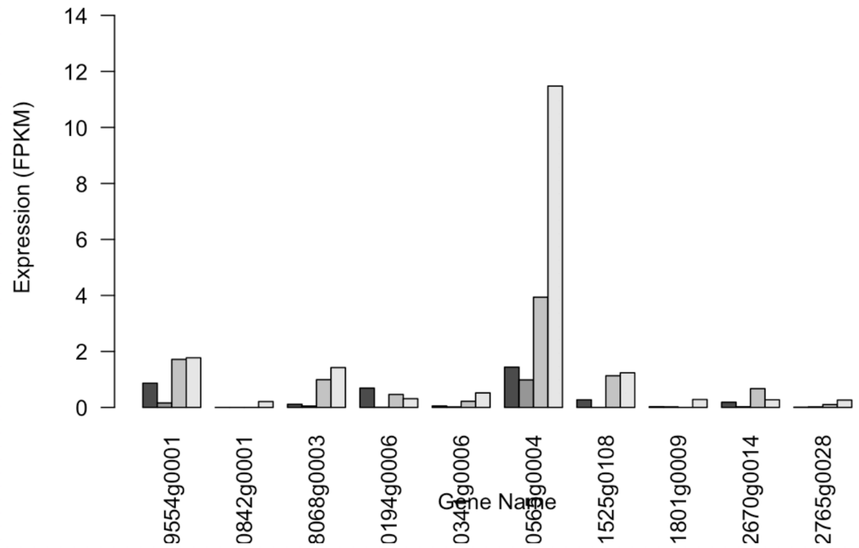
```
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, xlab="Gene Name")
```



What is wrong down here?

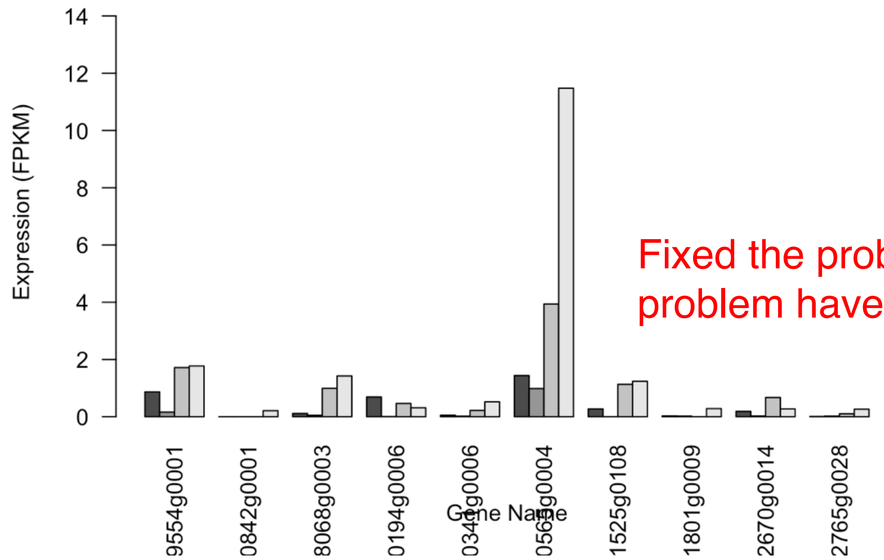
Change x axis labels to appear vertically

```
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, xlab="Gene Name", las=2)
```



Change x axis labels to appear vertically

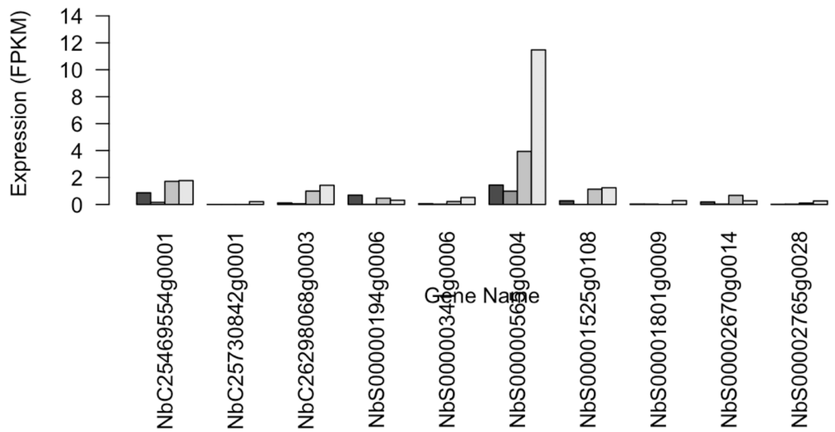
```
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, xlab="Gene Name", las=2)
```



Fixed the problem, but what new problem have we revealed?

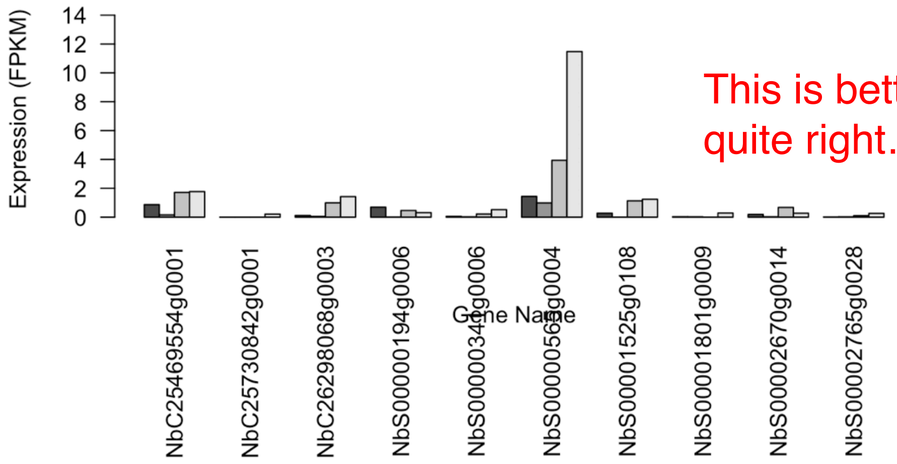
Adjust margins such that gene names are not cut off

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, xlab="Gene Name", las=2)
```



Adjust margins such that gene names are not cut off

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, xlab="Gene Name", las=2)
```

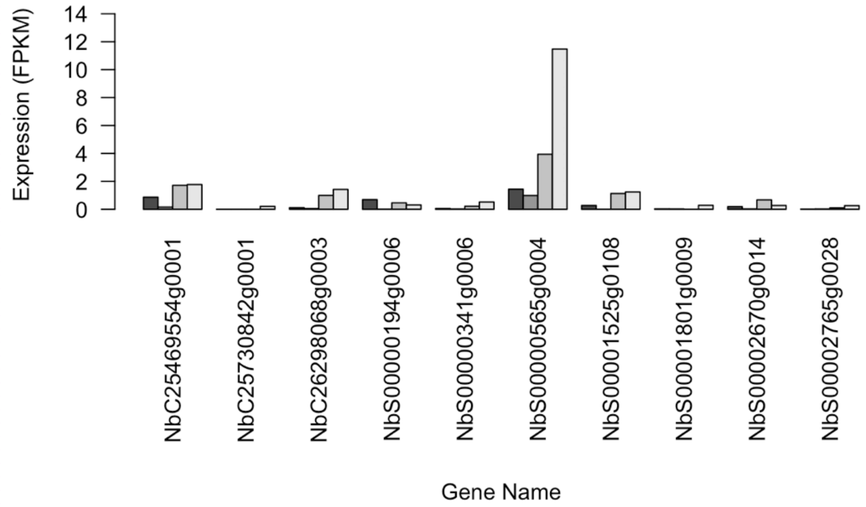


This is better, but still not quite right…

Fix ‘Gene Name’ label location

Use mtext instead of xlab

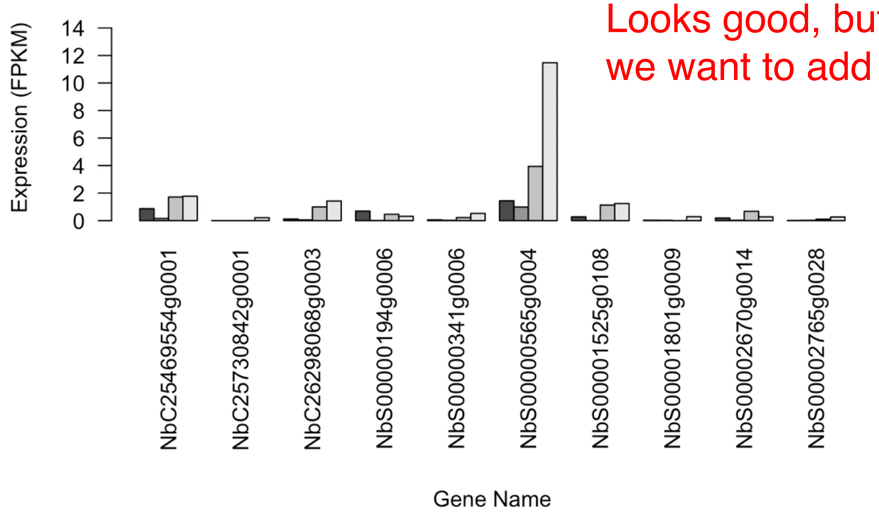
```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, las=2)
mtext("Gene Name", 1, 10)
```



Fix ‘Gene Name’ label location

Use mtext instead of xlab

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, las=2)
mtext("Gene Name", 1, 10)
```

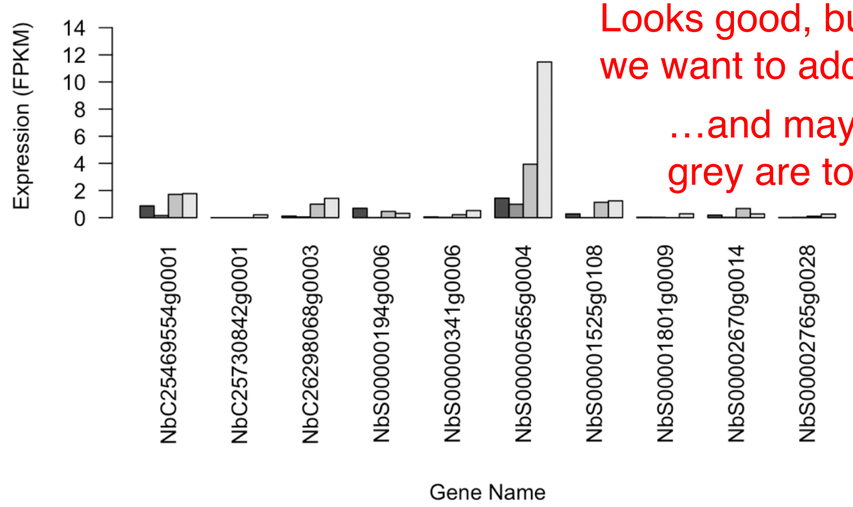


Looks good, but maybe we want to add a title…

Fix ‘Gene Name’ label location

Use mtext instead of xlab

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression$gene_sh
ort_name, las=2)
mtext("Gene Name", 1, 10)
```

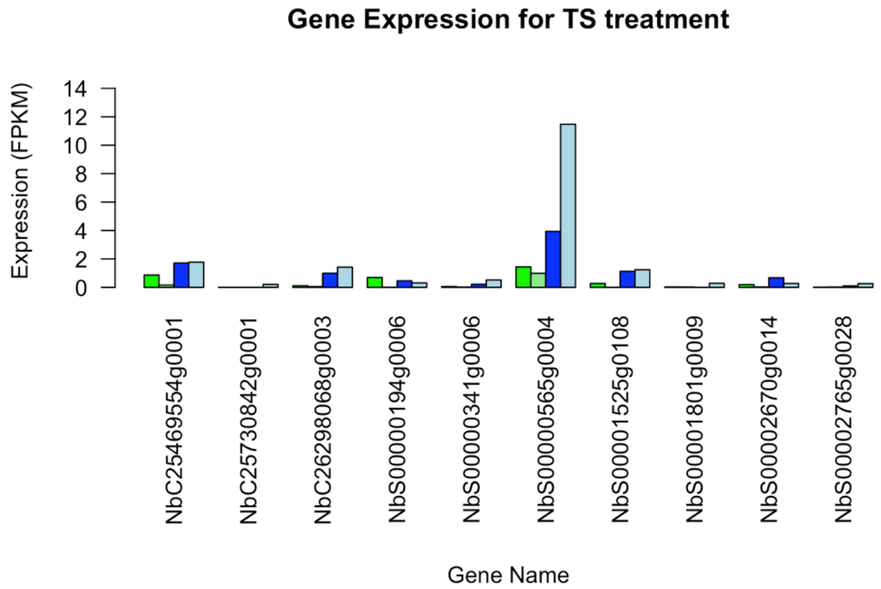


Looks good, but maybe we want to add a title…

…and maybe shades of grey are too boring….

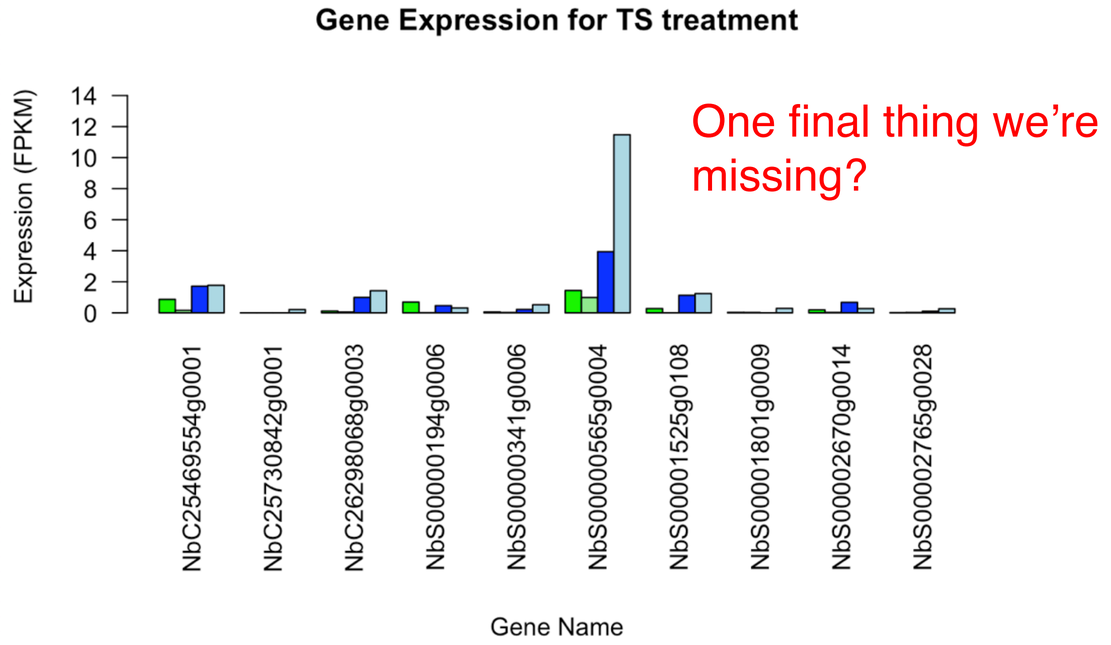
Adding a title and colors

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression
$gene_short_name, las=2, col=c("green", "lightgreen", "blue", "lightblue"), main="Gene Expression for TS treatme
nt")
mtext("Gene Name", 1, 10)
```



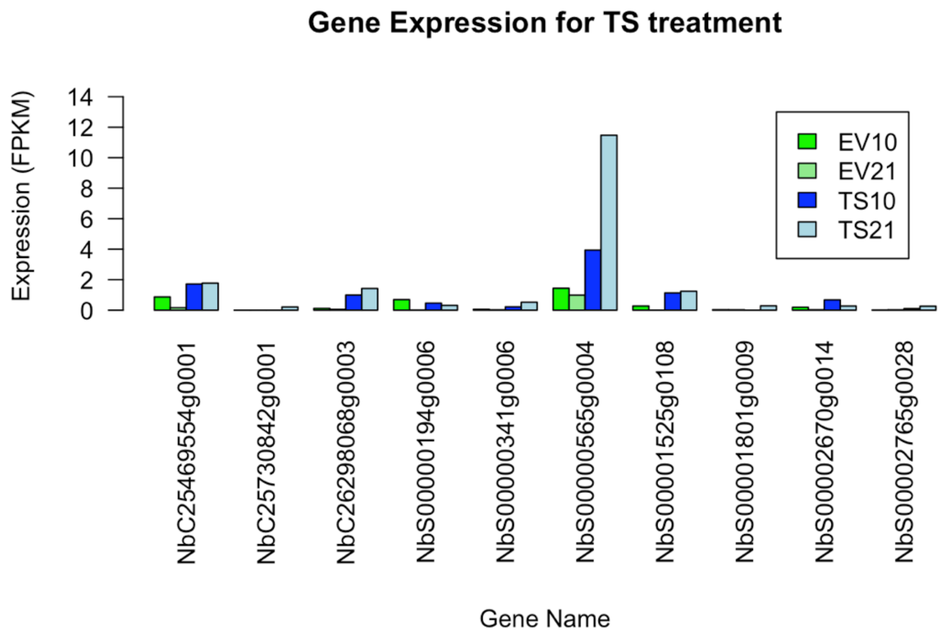
Adding a title and colors

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression
$gene_short_name, las=2, col=c("green", "lightgreen", "blue", "lightblue"), main="Gene Expression for TS treatme
nt")
mtext("Gene Name", 1, 10)
```



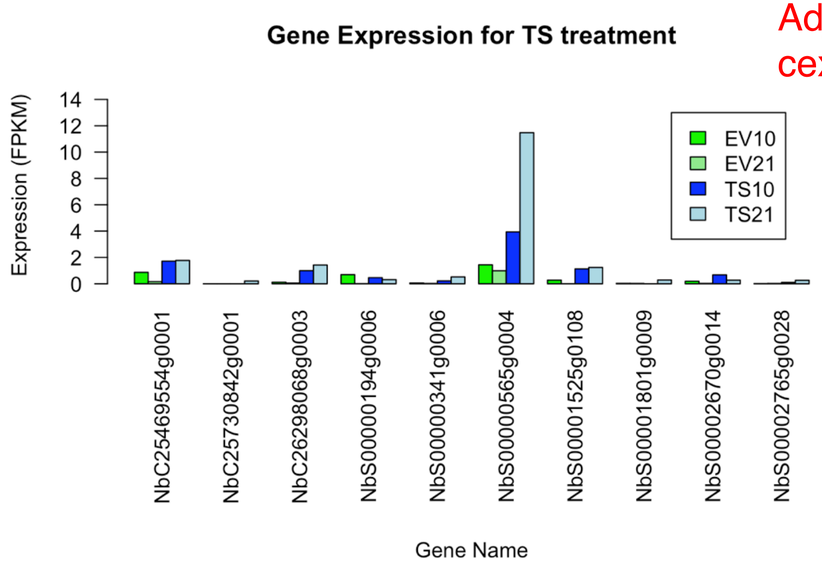
Adding a legend

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression
$gene_short_name, las=2, col=c("green", "lightgreen", "blue", "lightblue"), main="Gene Expression for TS treatme
nt")
mtext("Gene Name", 1, 10)
legend(40,13,legend=c("EV10", "EV21", "TS10", "TS21"), fill=c("green", "lightgreen", "blue", "lightblue"))
```



Adding a legend

```
par(oma=c(8,0,0,0))
barplot(t(gene_exp), beside=T, ylim=c(0,15), ylab="Expression (FPKM)", names.arg=data002_gene_expression
$gene_short_name, las=2, col=c("green", "lightgreen", "blue", "lightblue"), main="Gene Expression for TS treatme
nt")
mtext("Gene Name", 1, 10)
legend(40,13,legend=c("EV10", "EV21", "TS10", "TS21"), fill=c("green", "lightgreen", "blue", "lightblue"))
```



Adjust x-y position and cex if necessary