Visualization with R, RStudio and RMarkdown

Genomic Data Visualization & Integration

HUGEN 2073

(Slides adapted from Ryan Minster with permission)

Learning Objectives

By the end of the session, students will be able to:

- Create and manipulate objects and vectors in R
- Manipulate data frames in R
- Create very basic plots in R

ACHTUNG!

- I have tried to catch Microsoft autocorrects but may have missed some.
- Beware the difference between ", “, and ”.
- Beware the difference between ', ‘, ’, and `.
- Beware the difference between -, –, and — (a hyphen, an en dash and an em dash, respectively). Also - vs --. Also = vs ==.
- Double check capitalization.
- Beware the difference between O, o and 0, and between 1, I and l.

Software

- **R** is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.
- **RStudio** is an **integrated development environment (IDE)** for R.
- **RMarkdown** is a package for processing markup documents in Rstudio that weave together narrative text and (usually) R code into formatted output in HTML, PDFs, MS Word etc.

R

- R is a programming language and free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing.
- It is widely used by statisticans and data miners, and especially among academics (see *free* above).

R

- R performs **mathematical calculations**

```
> 1 + 2   # comments can be added after hashes
[1] 3
```

- Often these calculations are in the form of **functions**.

```
> sqrt(4)
[1] 2
> mean(c(1,2,3,4))
[1] 2.5
> rnorm(n = 2, mean = 0, sd = 1)
[1] -0.4171808 -0.2499255
> rnorm(2, 0, 1)
[1] -0.06301373 -0.97962780
```

R

- Functions can be **nested** inside of other functions.

```
> rnorm(n = sqrt(mean(c(2,4,6))), mean = 1 - 1, sd = 1)
[1] -0.4171808 -0.2499255
```

R

- To get **documentation** about a function use ?.

```
> ?rnorm
```

- To search **documentation** use ??.

```
> ??normal
```

R

- It can save information into **objects** (which are like variables)

```
> a <- 1
> a
[1] 1
> a <- 1 + 2
> a
[1] 3
> beta <- 2 + 2
> beta
[1] 4
> a + beta
[1] 7
```

R

- These **objects** can be **vectors** of values

```
> a <- c(1, 2, 3, 4)
> a
[1] 1 2 3 4
> a <- c(TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE)
> a
[1]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
```

R

- You can choose **subsets** of values in a vector

```
> a <- c("a", "b", "c", "d")
> a
[1] "a" "b" "c" "d"
> a[2]
[1] "b"
> a[2:3]
[1] "b" "c"
> a[2] <- "z"
> a
[1] "a" "z" "c" "d"
```

R

- You can operate on **subsets** of values in a vector

```
> a <- c(1, 2, 3, 4)
> a
[1] 1 2 3 4
> a[c(2,4)] <- a[c(2,4)] + c(0.3, 0.5)
> a
[1] 1 2.3 3 4.5
```

R

- Using **conditions** and TRUE and FALSE can be handy

```
> a <- c(1, 2, 3, 4)
> a == 2
[1] FALSE  TRUE FALSE FALSE
> a[a == 2]
[1] 2
> a[c(FALSE, TRUE, FALSE, FALSE)]
[1] 2
```

R

- R can work with tables of data called data frames.

```
> a <- read.csv("data3.csv")  # read in CSV data from file
> a
   name   x    y
1 Anna 1.1 0.50
2 Xiao 1.9 0.75
3 José 3.0 0.80
```

R

- Since data frames are two dimensional, to subset you need **two coordinates**:

```
> a
   name   x    y
1 Anna 1.1 0.50
2 Xiao 1.9 0.75
3 José 3.0 0.80
> a[2, 3]
[1] 0.75
> a[3, 1]
[1] "José"
```

R

- If you leave **one of the two coordinates empty**, you'll get a vector:

```
> a
   name   x    y
1 Anna 1.1 0.50
2 Xiao 1.9 0.75
3 José 3.0 0.80
> a[2, ]
   name   x    y
2 Xiao 1.9 0.75
> a[, 1]
[1] "Anna" "Xiao" "José"
```

R

- You can use conditions and functions within the [s and]s too.

```
> a
   name   x    y
1 Anna 1.1 0.50
2 Xiao 1.9 0.75
3 José 3.0 0.80
> a[a[, 1] == "Xiao", c("x")]
[1] 1.9
```

R

- We often work with a **whole column at a time**, and there is a shortcut for that using $.

```
> a
   name   x    y
1 Anna 1.1 0.50
2 Xiao 1.9 0.75
3 José 3.0 0.80
> a$name
[1] "Anna" "Xiao" "José”
>
```

R

- Note these three are exactly equal:

```
> identical(a$name, a[, 1])
[1] TRUE
> identical(a$name, a[, c("name")])
[1] TRUE
> identical(a[, 1], a[, c("name")])
[1] TRUE
```

R

- R can also manipulate tables of data by adding columns that are transformations of other columns:

```
> a$z <- a$x + a$y
> a
   name   x    y    z
1 Anna 1.1 0.50 1.60
2 Xiao 1.9 0.75 2.65
3 José 3.0 0.80 3.80
```

R

- R can also manipulate tables of data by adding columns that are transformations of other columns:

```
> a$b <- sqrt(a$z)
> a
  name   x    y    z        b
1 Anna 1.1 0.50 1.60 1.264911
2 Xiao 1.9 0.75 2.65 1.627882
3 José 3.0 0.80 3.80 1.949359
```

R

- You can also add new data to a data frame.

```
> a$r <- runif(length(a$z))
> a
   name   x    y    z          r
1 Anna 1.1 0.50 1.60 0.38347170
2 Xiao 1.9 0.75 2.65 0.55333872
3 José 3.0 0.80 3.80 0.04019723
>
> write.csv(a, "newdata.csv") # write object a to CSV file
```

R

- **Reading** and **writing** CSV data into and out of R

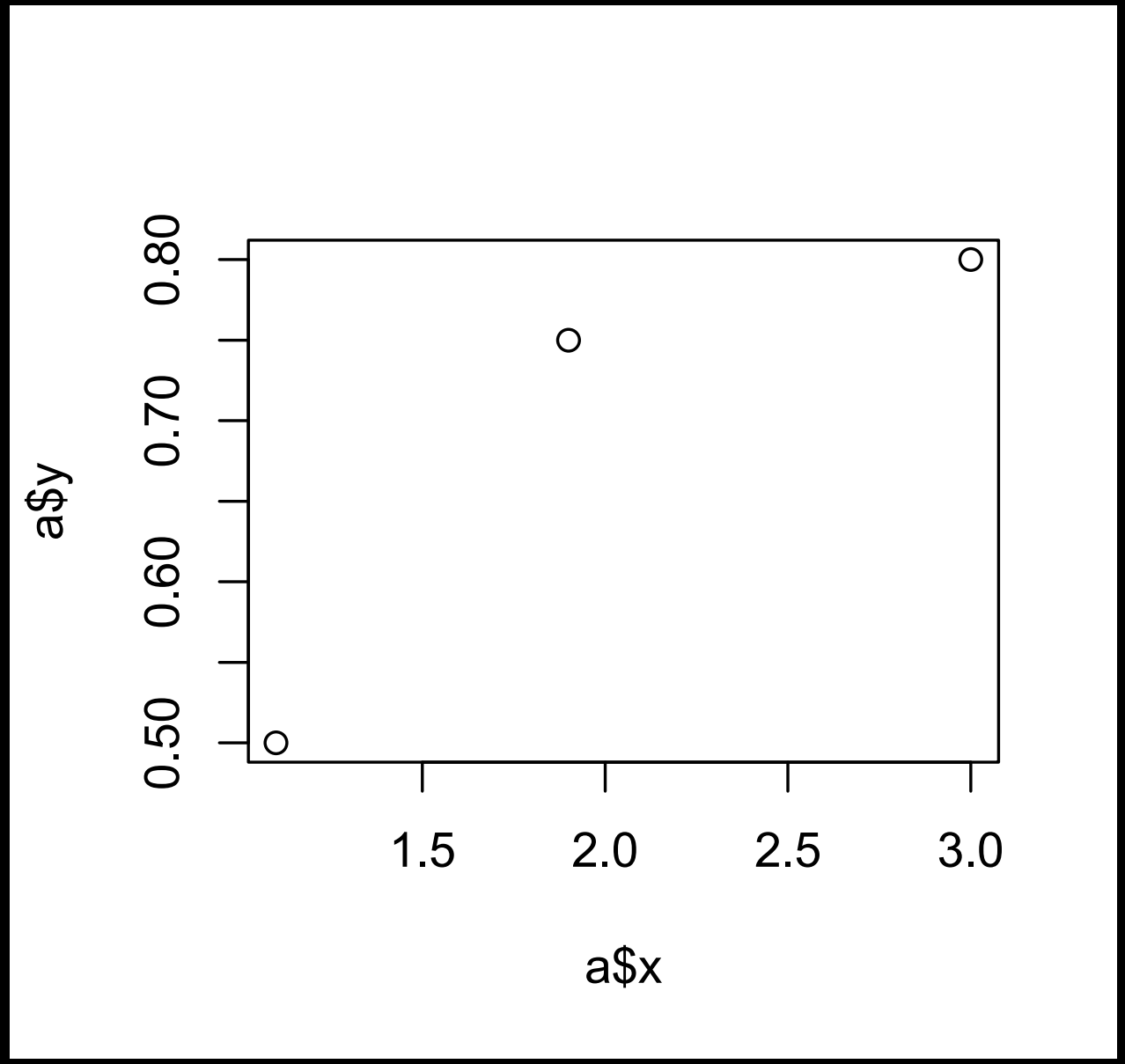
```
> a <- read.csv("data3.csv")
> a
  name   x    y    z          r
1 Anna 1.1 0.50 1.60 0.38347170
2 Xiao 1.9 0.75 2.65 0.55333872
3 José 3.0 0.80 3.80 0.04019723
>
> write.csv(a, "newdata.csv")
```

R

- R can also create graphics.

```
> plot(a$x, a$y)
```

```
> a[, c("x", "y")]
    x    y
1 1.1 0.50
2 1.9 0.75
3 3.0 0.80
```

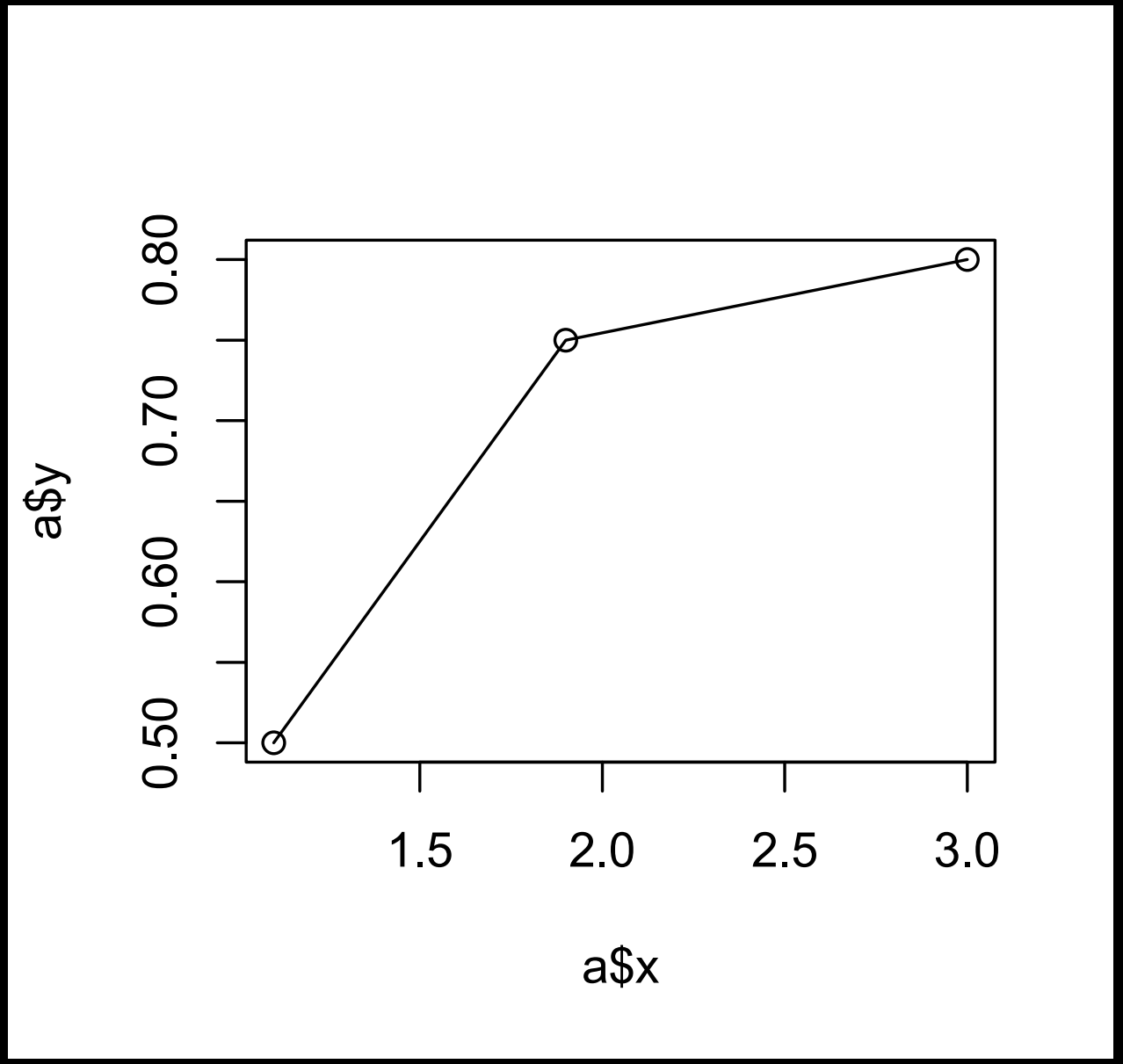


R

- R can also create graphics.

```
> plot(a$x, a$y,
    type = "o")
```

```
> a[, c("x", "y")]
    x    y
1 1.1 0.50
2 1.9 0.75
3 3.0 0.80
```

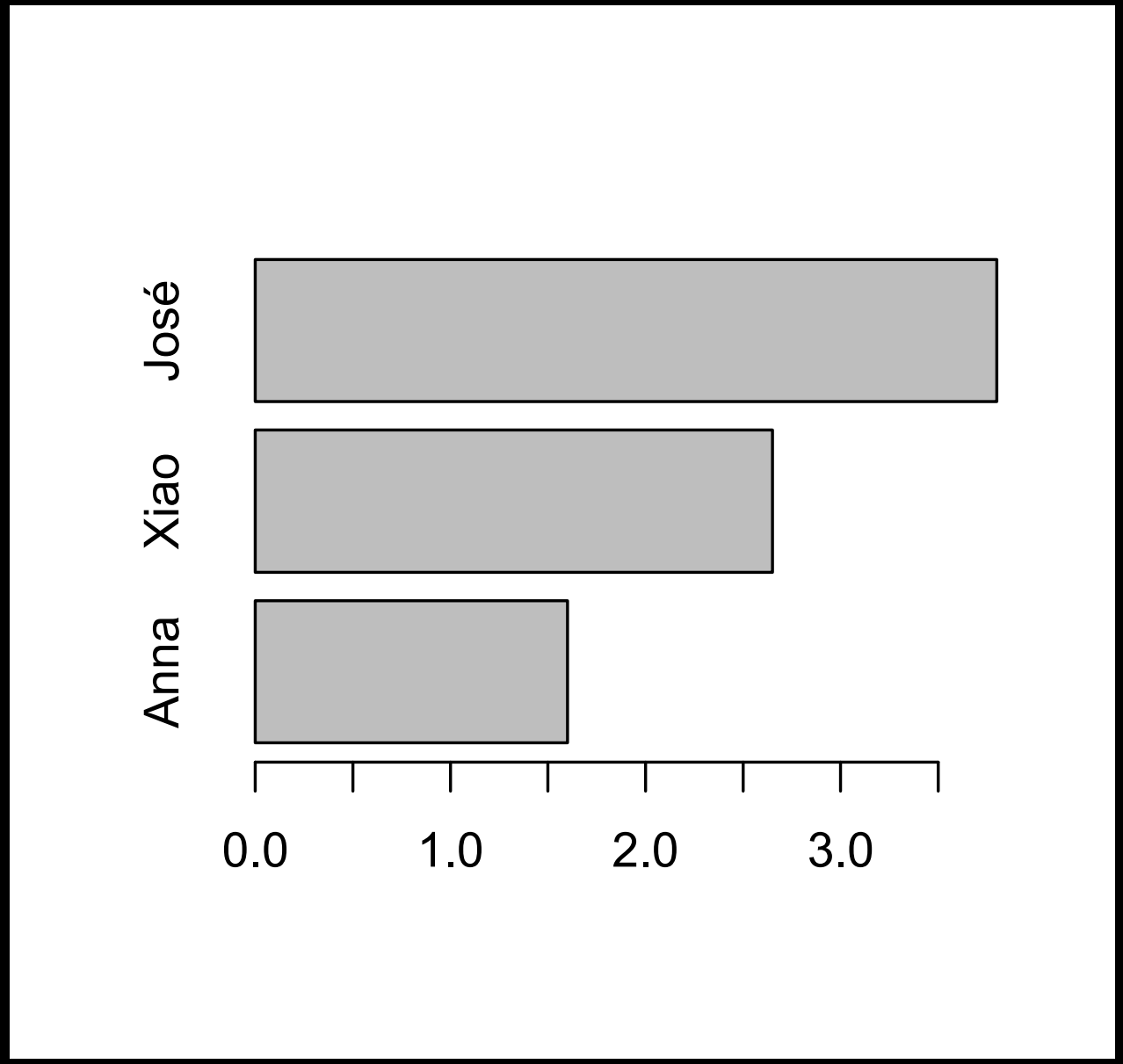


R

- R can also create graphics.

```
> barplot(a$z,
    horiz = TRUE,
    names.arg=a$name)

> a[, c("name", "z")]
   name    z
1 Anna 1.60
2 Xiao 2.65
3 José 3.80
```

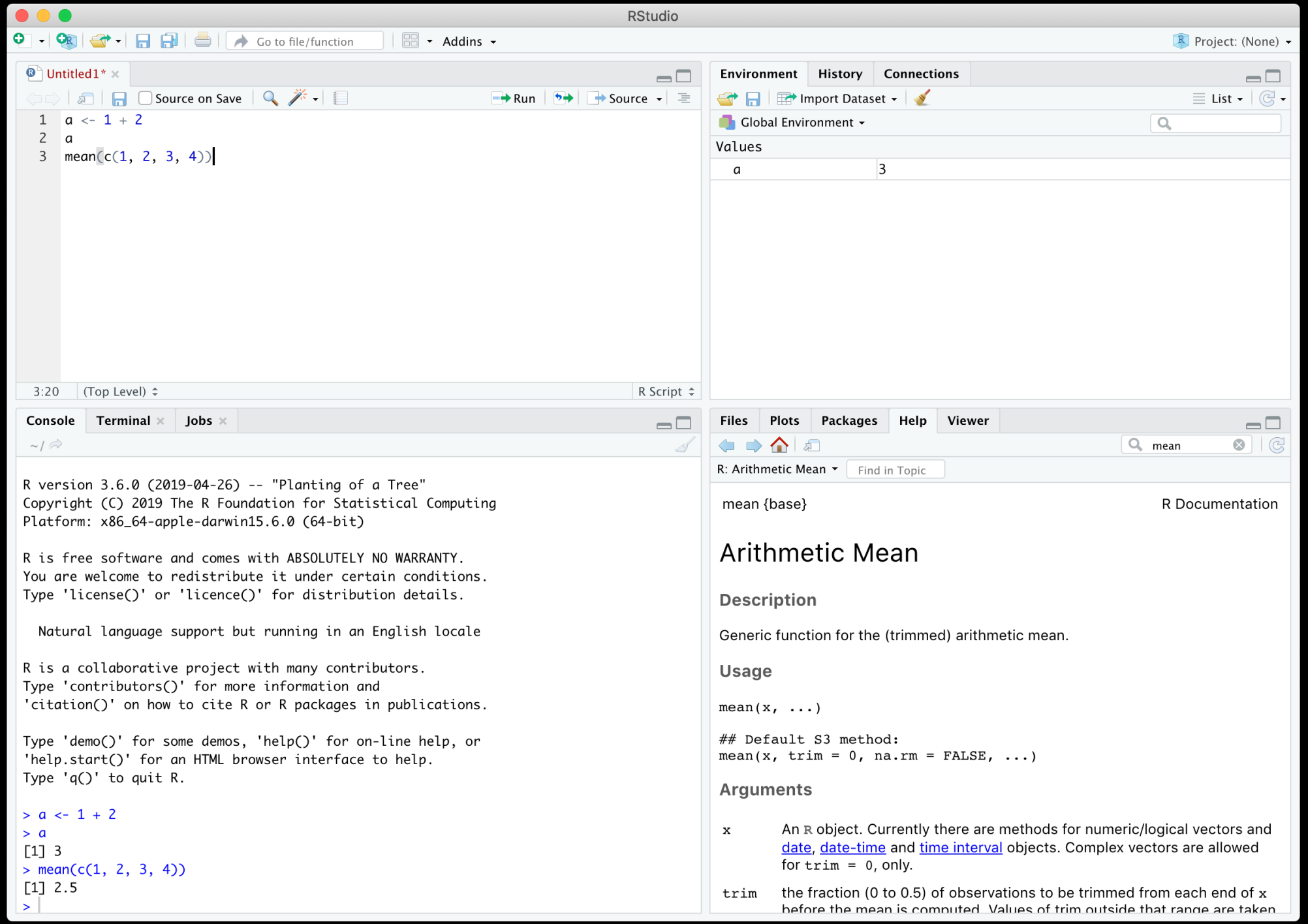


R

- R's basic capabilities can be extended by writing **additional functions** that can be make available publicly through **packages**.
  - A major extension of R's basic graphics uses a package called `ggplot2`

RStudio

- RStudio is an integrated development environment (IDE) for R.


RStudio
Go to file/function
Addins
Project: (None)
Untitled1*
Source on Save
Run
Source
1 a <- 1 + 2
2 a
3 mean(c(1, 2, 3, 4))
3:20
(Top Level)
R Script
Environment
History
Connections
Import Dataset
List
Global Environment
Values
a
3
Console
Terminal
Jobs
R version 3.6.0 (2019-04-26) -- "Planting of a Tree"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (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.
> a <- 1 + 2
> a
[1] 3
> mean(c(1, 2, 3, 4))
[1] 2.5
>
Files
Plots
Packages
Help
Viewer
mean
R: Arithmetic Mean
Find in Topic
mean {base}
R Documentation
Arithmetic Mean
Description
Generic function for the (trimmed) arithmetic mean.
Usage
mean(x, ...)
Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
Arguments
x
An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
trim
the fraction (0 to 0.5) of observations to be trimmed from each end of x

RMarkdown

- RMarkdown is a package for processing markup documents in RStudio that weave together narrative text and (usually) R code into formatted output in HTML, PDFs, MS Word etc.
- Markup documents are data files that are primarily geared toward telling a browser (or other type of program) how to display a document:
  - Make *this* a header, make *that* italics etc.
  - RMarkdown has the bonus of being able to incorporate statistical programming code right into the narrative document itself.

Basic RMarkdown File

````
---
title: "Plots for 2073"
  output:html_document
---
## Squares

Here is a plot of numbers and their squares.

```{r}
x <- c(1,2,3,4,5,6,7,8,9,10)
y <- x^2
plot(x, y)
```
````

Basic RMarkdown Formatting

- Make a header:
  ```
  ## Header
  ```
- Make text bold:
  ```
  **This text will be bold.**
  ```
- Make text italics:
  ```
  *This text will be italic.*
  ```

Basic RMarkdown Code Chunks

- To add some R code (this is called a **chunk**):

````
```{r}
1 + 2
x <- c(1,2,3,4)
y <- mean(x)
```
````

- To add some R embedded in the narrative text itself (**inline**):

```
The mean of the `r length(c(1,2,3,4))` values is `r x`.
```

Cool RMarkdown Stuff

- Add a floating table of contents:

```
output:
  html_document:
    toc: true
    toc_float: true
```

- Add code folding:

```
output:
  html_document:
    toc: true
    toc_float: true
    code_folding: hide
```

Cool RMarkdown Stuff

- Make a pretty table:

```
library(knitr)
kable(a)
```

- RMarkdown Cookbook:
  https://bookdown.org/yihui/rmarkdown-cookbook/

R Graphics

- R’s graphing capabilities are prodigious and (can be) astonishing, vibrant and useful.

- Google: `fancy r plots`

R Graphics

- Two of the graphics systems in R are **base graphics** and **`ggplot2` graphics**, which have different vocabularies for rendering plots.
- **Base graphics** are your factory default R functions
  - They can produce beautiful graphics
  - Interaction is initially more intuitive than `ggplot2`
  - Sophisticated changes from the default settings can be a pain to program
- **`ggplot2` graphics** are an upgrade
  - They can produce beautiful graphics
  - Interaction is initially less intuitive than base graphics
  - Sophisticated changes from the default settings are easier to program

Base Graphics

- `plot()` · scatter plots
- `boxplot()` · box plots
- `hist()` · histograms
- `barplot()` · bar plots

Base Graphics

- Common arguments of plotting commands
  - `x = <VECTOR1>, y = <VECTOR2>` · the data to be plotted
  - `main =` · plot title
  - `xlab =` · *x* axis label
  - `ylab =` · *y* axis label
  - `xlim =` · range of *x* axis (vector of length 2)
  - `ylim =` · range of *y* axis (vector of length 2)

Base Graphics

- So plotting numbers and their squares:

```
> x <- c(1,2,3,4,5,6,7,8,9,10)
> y <- x^2
> plot(x, y)
> plot(x, y, main = "Squares",
  xlab = "Integers", ylab = "Squares",
  xlim = c(-10, 110), ylim = c(-10, 100))
```

Base Graphics

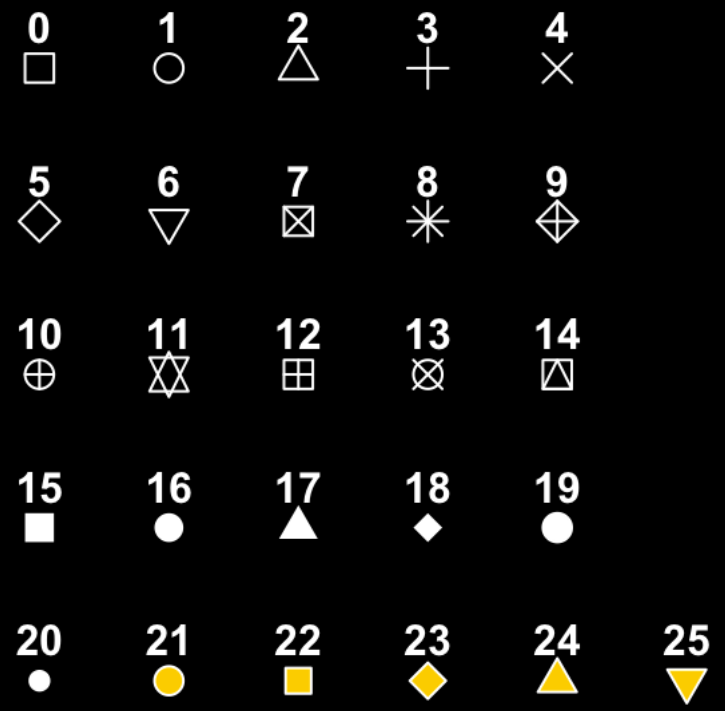
- Common arguments of plotting commands
- These can vectorized (different attributes for different points)
  - `cex =` · size of data points
  - `col =` · color
  - `pch =` · plotting symbols
  - `lty =` · line type
  - `lwd =` · line width (also applies to outline of data points)

`lty` Values

- You can use numbers or character strings
  - `0` or `"blank"` · no line
  - `1` or `"solid"` · default
  - `2` or `"dashed"` or `"44"` · dashed line
  - `3` or `"dotted"` or `"13"` · dotted line
  - `4` or `"dotdash"` or `"1343"` · dot dash dot dash line
  - `5` or `"longdash"` or `"73"` · dashes are longer
  - `6` or `"twodash"` or `"2262"` · short long short long dashes

pch Values

- You can use numbers or character strings
  - Numbers:



The inner color can be set for 21–25

Here the outline color is white and the inner color is yellow

  - Characters · Display the character itself as the point

`col` Values

- R has eight basic colors
  - 1 · black
  - 2 · red
  - 3 · green
  - 4 · blue
  - 5 · cyan
  - 6 · magenta
  - 7 · yellow
  - 8 · gray

`col` Values

- R has much more available in terms of color using predefined palettes and encoding using RGB, CMYK, and HSB systems (for example).

Base Graphics

- So, plotting numbers and their squares:

```
> x <- c(1,2,3,4,5,6,7,8,9,10)
> y <- x^2
> plot(x, y)
> plot(x, y, main = "Squares",
  xlab = "Integers", ylab = "Squares",
  xlim = c(-10, 110), ylim = c(-10, 100),
  pch = c(15,16,17), col = c(2,3,4), cex = 3,
  lwd = 2)
```

Base Graphics

- Common commands for adding to an existing plot:
  - `abline()` · adds a straight line
  - `legend()` · adds a legend
  - `points()` · adds additional points to a plot (like a second series)
  - `lines()` · adds a line series to a plot

Base Graphics

- Let's add a line between the points:

```
> lines(x, y, col = 1, lwd = 4, lty = "dotted")
```

- Let's also add a vertical line at 9 and a horizontal line at 81:

```
> abline(v = 9, lwd = 4, col = 2)
> abline(h = 81, lwd = 4, col = 4)
```

- Let's finally add a second set of points:

```
> y2 <- x * 9
> points(x, y2, col = 1, lwd = 4, cex = 2)
```

Visualization with R: Example with R Base Graphics

Genomic Data Visualization & Integration

HUGEN 2073

(Slides adapted from Ryan Minster with permission)

Learning Objectives

By the end of the session, students will be able to:

- Create a scatterplot with two categories of data points using R base graphics

Create a Scatterplot in R Base Graphics

Steps

1. Read the data into R
2. Execute the optical method of analysis to review the data
3. Plot a scatterplot of height vs age
4. Plot a scatterplot of height vs age stratified by sex

Read in the Data

```
data <-
  read.csv("20220120-rlm-2073_Data.csv")
```

Optical Method

LOOK! AT! THE! DATA!

- How many rows and columns are there?

```
dim(data)
```

- What are the values in the data?

```
head(data)
tail(data)
summary(data)
```

Optical Method cont'd

- "table()" categorical data

```
table(data$sex)
```

- Look at basic histograms of continuous data using hist()

```
hist(data$age)
hist(data$height)
```

- Play around with breaks to see different shapes of the histograms

```
hist(data$height, breaks = 10)
hist(data$height, breaks = 100)
hist(data$height, breaks = 1000)
```

Create a Scatterplot using `plot()`

- If we are plotting height vs age
  - That is, by convention, **dependent** vs **independent** variable, *or*
  - ***y*** vs ***x***
  - So, ***y*** **= height** and ***x*** **= age**

```
plot(x = data$age, y = data$height) or
plot(data$age, data$height
```

Create a Scatterplot using `plot()`

- Change points shape and color

```
plot(data$age, data$height,
     pch = 16, col = alpha("black", 0.25))
```

- `alpha()` is a function that lets you add transparency to a color, such that `0.25` means 25% opaque and 75% transparent.

Stratify Scatterplot by Category

- If we are plotting height vs age by sex, we can
  - Plot female and male participants **separately**
  - Plot female and male participants **together using color, shape, or both to distinguish them**

Stratify Scatterplot by Category Separately

- Plotting separately

```
plot(data$age[data$sex == "F"],
     data$height[data$sex == "F"],
     pch = 17, col = alpha("darkgreen", 0.25))
plot(data$age[data$sex == "M"],
     data$height[data$sex == "M"],
     pch = 16, col = alpha("purple", 0.25))
```

Stratify Scatterplot by Category Separately

- But the *y* axis scales are different, so difficult to compare!
  Here, the *x* axis scales happen to match, but they could be different too.

```
summary(data$height)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  143.0   164.0   171.0   171.6   179.0   204.0 
plot(data$age[data$sex == "F"],
     data$height[data$sex == "F"],
     pch = 17, col = alpha("darkgreen", 0.25),
     ylim = c(143, 204))
plot(data$age[data$sex == "M"],
     data$height[data$sex == "M"],
     pch = 16, col = alpha("purple", 0.25),
     ylim = c(143, 204))
```

Stratify Scatterplot by Category Together

- Plotting together, I have two approaches to choose from:
  - Plot female participants first, then **overlay** male participants
  - Plot **simultaneously**, supplying `pch` and `col` with vectors that correspond to how each data point will be displayed

Stratify Scatterplot by Category Overlay

- Plotting one then the other

```
plot(data$age[data$sex == "F"],
     data$height[data$sex == "F"],
     pch = 17, col = alpha("darkgreen", 0.25))
points(data$age[data$sex == "M"],
       data$height[data$sex == "M"],
       pch = 16, col = alpha("purple", 0.25))
```

Stratify Scatterplot by Category Overlay

- But the *y* axis scales was based on the female participants, so data points for tallest male participants are off the plot!
  Again, the *x* axis scales happen to work, but the same issue could occur.

```
summary(data$height)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  143.0   164.0   171.0   171.6   179.0   204.0
plot(data$age[data$sex == "F"],
     data$height[data$sex == "F"],
     pch = 17, col = alpha("darkgreen", 0.25),
     ylim = c(143, 204))
points(data$age[data$sex == "M"],
       data$height[data$sex == "M"],
       pch = 16, col = alpha("purple", 0.25))
```

Stratify Scatterplot by Category Simultaneous

- Plotting simultaneously
  - First create vectors holding what will be given to `pch` and `col`.

```
data$pch[data$sex == "F"] <- 17
data$pch[data$sex == "M"] <- 16
data$col[data$sex == "F"] <- "darkgreen"
data$col[data$sex == "M"] <- "purple"
```

  - Check out the data to see if it looks right LOOK! AT! THE! DATA!

```
head(data)
table(data$sex, data$pch)
table(data$sex, data$col)
```

Stratify Scatterplot by Category Simultaneous

- Plotting simultaneously
  - Second provide those vectors to `pch` and `col` in `plot()`.

```
plot(data$age, data$height,
     pch = data$pch, col = alpha(data$col, 0.25))
```

  - Note that you had to do more to set up the data, but less in the actual plotting code itself. No checking axes limits or additional functions to overlay additional points.

Final Notes

- Of course, final plots would include better axis labels, a title, and a legend, at the least.

- Additionally, beware using shapes and colors that code for male and female in ways that align with and perpetuate sexist stereotypes of masculinity and femininity.
  - Such encodings can be culture-specific.
  - Often in American culture, pink and round ( ) is female and blue and angular ( ) is male. Avoid such mappings.