

Intro to Plotting in R

brouwern@gmail.com

July 2017

Contents

1 Prerequisites	21
2 Create book infrastructure	23
2.1 Change index.rmd	23
2.2 Update README.md	23
2.3 Update _output.yml	23
2.4 Use subdirectories	23
2.5 Keep the merge file	23
2.6 Workflow	24
2.7 Set up options for entire	24
2.8 Skip files	24
2.9 Suppress warnings etc	24
2.10 Do not stop on error	24
3 Computational biology and ethics	25
3.1 Reading list	25
4 Working outline	27
4.1 TODO	27
4.2 General book conventions	27
4.3 Book names to use / riff off of for BOOK names	27
4.4 Outline	28
4.5 Ideal features of book	30
4.6 Test code	30
I Introduction	33
5 Welcome to computational biology for all!	37
5.1 R and RStudio - your new best friends!	37
5.2 How to use this book - be an active learner!	38
5.3 Biological scope of this book	38
6 What is Computational Biology?	39

6.1	Notes	39
6.2	Luscombe et al 2001	40
6.3	Taprich et al 20121	40
6.4	Smith 2015 Frontiers in Genetics	40
6.5	Good readings / references	41
6.6	Potential references	41
6.7	Original info, from Eco Data Science project	41
6.8	What is data science?	42
6.9	Refereces	42
6.10	Bibliography	43
7	How will this book teach computational biology?	45
7.1	What you'll learn	45
7.2	Teaching approach	46
7.3	Requirements	46
7.4	What's not in this book / course	46
8	What is R and why use it?	47
8.1	How do we typically use software in science?	47
8.2	What does R do?	48
8.3	Why use R	48
8.4	Who uses it?	48
8.5	R and computational reproducibility	48
8.6	Alternatives to R	49
9	The layout of RStudio	51
9.1	RStudio (Desktop Or Cloud)	51
9.2	R emulators	51
9.3	Other ways	52
9.4	RStudio at a glance	52
II	Getting started with R	55
10	Hello R! A simple R plotting exercise	59
10.1	Key Ideas	59
10.2	Data in R	59
10.3	Loading data that comes with R	60
10.4	Plotting simple datasets	60
10.5	Commands in R	61
10.6	Arguements	62
10.7	Arguements and more arguments	62
10.8	Multiple arguments at the same time	64
10.9	Comments	64
10.10	Now you try it	65
10.11	A note on plotting	65

CONTENTS	5
11 Downloading R packages (and their data)	67
11.1 Loading data from R packages	67
11.2 OPTIONAL: What functions come with base R?	68
11.3 OPTIONAL: What packages come with base R?	68
11.4 Load data from an external R package	69
11.5 Optional: Seeing all of your installed packages	70
11.6 Downloading packages using RStudio	70
12 Data in dataframes	71
13 Plotting data in dataframes	75
13.1 Base-R graphics	76
13.2 You try it	82
14 Build your own dataframe part I: Vectors	85
14.1 Dataframes are a type of data structure	85
14.2 Dataframes are made from vectors	86
14.3 Make your own vectors	86
14.4 Checking the length of vectors	87
14.5 Another example	88
14.6 Try it yourself	89
14.7 A complicate dataframe to make	89
14.8 Build the dataframe	89
14.9 References	90
15 Build your own dataframe	91
15.1 Your turn	92
15.2 References	92
16 Build your own dataframe	93
16.1 Your turn	94
16.2 References	94
17 Introduction to ggplot and ggpubr	95
17.1 Regression line	97
17.2 Smoother	99
17.3 correlation	100
17.4 Long time series	101
17.5 Regression lines	109
17.6 Correlations	111
17.7 Original data	113
18 Loading packages & data from GitHub	117
18.1 Introduction	117
18.2 [] Accessing GitHub using devtools	118
18.3 [] Downloading the wildlifeR package with install_github()	118
18.4 [] The wildlifeR packge webiste	119

19 Installing from github	121
20 Packages & their dependencies	123
20.1 Optional: Make a plot with ggpubr	123
20.2 Challenge	124
21 Load data from internet	127
21.1 Base R plot	127
21.2 ggpubr - have to stack it	127
22 Assignment: Building dataframes by hand	129
22.1 Introduction	129
22.2 To do	129
22.3 Preliminaries	130
22.4 The data	130
22.5 Creating vectors in R	131
22.6 Checking the length of vectors	132
22.7 Exponents and scientific notation	132
22.8 Some more vectors	135
22.9 Make a dataframe	138
22.10 Make Figure 1 with base R	138
22.11 Make Figure 1 with ggpubr	139
22.12 Add statistics	143
23 Dataframe by hand - problem set - KEY	145
23.1 Preliminaries	145
23.2 Step one: Make each column	145
23.3 Basic Data exploration	147
23.4 Part 2: Build the dataframe	147
23.5 Part 3: Make figure	148
23.6 Graph of length.bp	149
23.7 Create new variable: T.m/length.bp	150
23.8 New plot: T.m801 versus P.GC	151
23.9 Grading function	152
24 Accession number; leave this code as is	153
25 Length of 16sRNA gene in BP	155
26 Percent GC	157
27 Melting temperature	159
28 References - leave this as is	161
29 Dataframe by hand - problem set	165
29.1 Preliminaries	165

29.2 Step one: Make each column	165
29.3 Basic Data exploration	166
29.4 Part 2: Build the dataframe	166
29.5 Part 3: Make figure	166
29.6 Graph of length.bp vs T.m	167
29.7 Create new variable: T.m/length.bp	167
29.8 New plot: T.m801 versus P.GC	167
30 Making a simple dataframe	169
30.1 Working with data in 2 seperate vectors	169
31 Load data from internet	175
31.1 Introduction	175
31.2 [] Downloading a .csv file using getURL()	176
31.3 OPTIONAL: Plotting West Virginia Eagle Data	177
32 Worked example: Data visualization in R	179
32.1 Preliminaries	179
32.2 Data exploration: Getting to know data	180
32.3 Numeric summaries data	181
32.4 Plotting data	182
32.5 Adding color to a plot	183
32.6 Changing symbols	184
32.7 Adding more stuff to a plot	185
32.8 Boxplots	186
32.9 Multivariate groups	186
32.10 Notes for Further study	190
33 ggpubr	193
33.1 Vocab	193
33.2 Learning objectives	193
33.3 Introduction	193
33.4 Preliminaries	194
33.5 Make a basic ggpubr plot	194
33.6 Scatter plots in ggpubr	196
33.7 Coloring by a categorical variable	197
33.8 Coloring by a continuous numeric variable	197
33.9 Adding a line of best fit	198
33.10 Adding a data ellipse	199
33.11 Add a correlation coefficient	199
33.12 TASK	200
34 gpibr - allometric data	201
34.1 Vocab	201
34.2 Learning objectives	201
34.3 Introduction	201

34.4 Preliminaries	202
34.5 Make a basic ggpubr scatterplot	202
34.6 Scatter plots in ggpubr	202
34.7 Adding a line of best fit	203
34.8 Adding a data ellipse	203
34.9 Add a correlation coefficient	204
34.10 Make an allometric plot	205
34.11 Change color by a categorical variable	206
34.12 Change color by a continuous variable	207
34.13 Add a smoother	207
34.14 Task	208
35 Improving plots	211
35.1 Vocab	212
35.2 Learning objectives	213
35.3 Introduction	213
35.4 Preliminaries	213
35.5 Make a basic ggpubr plot	214
35.6 Scatter plots in ggpubr	215
35.7 Coloring by a categorical variable	216
35.8 Coloring by a continuous numeric variable	217
35.9 Adding a line of best fit	217
35.10 Adding a data ellipse	218
35.11 Add a correlation coefficient	218
35.12 TASK	219
36 Scatter plots	221
36.1 Checkpoint: Plot the data	221
37 3D scatterplots	223
37.1 Key concepts / vocab	223
37.2 Preliminaries	223
37.3 3D scatterplots	224
37.4 Plane of best fit	227
37.5 From 3D plots to PCA	228
37.6 Task	229
38 Scatterplots and logs	231
38.1 Regression / correlation; logs	231
39 Making fancy plots	235
40 Using ggplot's stat_summary	239
40.1 Standard plot of means w/ error bars	239
40.2 w/ stat_summary(...geom = "pointrange")	239
40.3 Plot just means	240
40.4 w/ geom = "point"	240

CONTENTS	9
41 Making a simple data frame for boxplots	241
42 Working with data in 2 seperate vectors	243
42.1 Summary Statistics	243
42.2 Making a dataframe	244
42.3 Making a boxplot	244
42.4 Making a histogram	245
42.5 Making a density plot	245
42.6 Advanced	246
42.7 Historical reference	246
43 Barplots	249
44 Boxplots - how they are made	251
45 Boxplots	253
45.1 Preliminaries	253
45.2 Boxplots	253
45.3 Aside: Notched boxplots	254
45.4 Fill	254
45.5 Adding raw data	256
45.6 Jitter the raw data	256
46 Central tendency	263
46.1 Preliminaries	263
46.2 Getting to know the frogs	263
46.3 Summary statistics	264
47 Error plots	267
47.1 Plot means with error bars	267
47.2 arms	272
48 How histograms are made	275
49 Histograms	277
49.1 Preliminaries	277
49.2	280
49.3 Examples of histogram	280
49.4 Skewed data	280
49.5 Heavily skewed data	281
49.6	281
49.7 Fill	282
49.8 Central tendency	283
49.9 Median	284
49.10Density plot	285
49.11Align vert	288
49.12Density plot overlay	288

49.13Density plot	288
50 Boxplots	291
50.1 Preliminaries	291
50.2 Boxplots	291
50.3 Fill	292
50.4 Central tendency	293
50.5 Median	293
50.6 Density plot	294
50.7 Align vert	297
50.8 Density plot overlay	298
50.9 Density plot	298
51 Frog arms	299
51.1 Preliminaries	299
51.2 A 1st encounter with dplyr	299
51.3 group_by	301
51.4 Alternatives	303
51.5 Your turn	303
52 Doing math for stats by hand: standard deviation	305
52.1 Calcualting variance & standard deviation by hand, step by step	305
52.2 “Deviations” between the mean and each observation	305
52.3 Start making a dataframe (df)	306
53 Calculate the “Squared deviations” between the mean and each observation	307
54 Sum of squares between the mean and each deviation	309
55 And now...the variance	311
56 And now...the standard deviation	313
57 The standard error (SE)	315
58 R quirks: putting things in vectors	317
58.1 How R works with strings of numbers	317
58.2 How R works with strings of numbers	317
58.3 Fixing the problem	318
59 Summary stats for main PA eagle dataframe	319
60 Variability	321
60.1 Preliminaries	321
61 Understanding data helpfiles	323

CONTENTS	11
62 P-values	325
63 Introduction	327
64 P-value definition to memorize	329
65 P-values are tricky	331
66 P-value example: coin flipping	333
67 P-value example 1	335
68 P-value example 2: heights of elementary school students	337
69 P-values and biological sequences	339
70 Inference-by-eye using RA Fisher's Cat Data	341
70.1 Load the data	341
70.2 Look at the data	341
70.3 Summarize the body weight (Bwt) data old-school using summaryBy()	341
70.4 Plot the data	342
70.5 T-test	344
71 Bootstrapping Tutorial: Fst from Evans et al 2016	345
72 Fst data	347
72.1 Load Fst into vector	347
72.2 Mean Fst	347
72.3 Distribution of Fst values	347
73 Bootstrapping Fst	351
73.1 Define function for the mean	351
73.2 Running boot	352
74 Compare Fst to Qst	355
74.1 Plot with ggplot	355
75 From the original text:	359
75.1 Doing math for stats by hand	359
76 Doing Math in R	361
77 Introduction	363
77.1 The basic math stuff in R	363
78 T-test - deprecated? newer version?	365
78.1 Preliminaries	365

78.2 T-test	366
78.3 Arm girth	369
79 3D plotting	371
80 Coding multivariate data - color and shape	373
81 Facets	375
82 Multivariate Data Analysis: Making Sense of High-Dimensional Data w/PCA etc	377
82.1 Preliminaries	377
82.2 High-dimensional Data in Biology	378
82.3 Exploratory versus Hypothesis testing	382
83 Overview: The vegan package for PCA	385
83.1 Principal components analysis - base R	387
83.2 Principal components analysis - vegan	388
83.3 Task	389
84 PCA etc worked example	391
84.1 Assignment	391
84.2 Introduction	391
84.3 Data	392
84.4 Scatter plot	392
84.5 Cluster analysis	393
84.6 PCA	394
85 Another PCA tutorial with Fisher's Iris Data	397
85.1 Important functions	397
85.2 Useful packages	397
85.3 Data: Fisher's Irises	397
85.4 Data visualization	398
85.5 3D visualziation	399
85.6 Beyond 3D dimensions with ordination	399
85.7 Run PCA	399
85.8 Visualize with biplot	400
85.9 Plotting by hand	401
86 Beyond PC1 and PC2	405
86.1 Summary command	406
86.2 PCA by hand	408
86.3 Set up data	408
86.4 covariance vs. correlation	408
87 eigen values	411
87.1 Calculation of the the Principal Components	412

CONTENTS	13
87.2 results of matrix mult for PCA	418
88 vegan PCA: Principal Components Analysis with vegans rda function	427
89 Fisher's Irises	429
90 Visualize variables in 2D	431
91 Do PCA with base R function	433
91.1 Run PCA	433
91.2 Plot Biplot in base graphics	433
92 Do PCA with vegan::rda	435
92.1 run a vegan PCA with rda()	435
92.2 Plot the RDA	435
93 vegan's ordination plotting functions	437
93.1 Add "hulls" around each species	437
93.2 Make plot pretty	438
94 Hierarchical clustering with Euclidean distances	443
94.1 Preliminaries	443
94.2 Vocab	443
94.3 Note: This is NOT a real analysis	444
94.4 Load data	444
94.5 Plot the data	444
94.6 Distances and distance matrices	445
94.7 Calculate Euclidean distance	445
94.8 Distance knowlesi to yoelli	451
94.9 Distances with the dist() function	451
94.10Forming the first cluster	451
95 Cluster analysis with allometric data	453
95.1 Cluster analysis	453
95.2 Preliminaries	453
95.3 PCA data exploration	454
95.4 Task	457
96 Higgs and Atwood student version with updates	459
96.1 Introduction	459
96.2 Preliminaries	459
96.3 Raw data exploration	462
96.4 Principal component analysis	468
96.5 Cluster analysis	470
97 Higgs and Atwood student version with updates - copy or alt	473

97.1 Introduction	473
97.2 Preliminaries	473
97.3 Raw data exploration	476
97.4 Principal component analysis	482
97.5 Cluster analysis	484
98 Writing functions - practice problems	487
98.1 Introduction	487
98.2 Part 1: Pure practice	487
98.3 Practice Function 3: Make your own natural log function	488
98.4 Practice Function 4: Make function that converts DNA directly to RNA	488
99 Computational Biology Portfolio: Practice Working with Strings	489
99.1 Introduction	489
99.2 Purpose	489
99.3 Gonder et al 2007 data	489
99.4 Gonder et al 2007 new sequences	490
99.5 Searching Genbank	492
99.6 Gonder et al 2007 previously published sequences	492
99.7 Searching Genbank	494
100 Writing user-friendly functions	495
100.1 Version 1	495
100.2 Version 2	496
100.3 Version 3	496
100.4 Version 4: Adding conditions and warnings	497
100.5 Version 5	498
101 Intro to R objects	501
101.1 Commands used	501
101.2 R Objects	501
101.3 Differences between objects	501
101.4 The Data	502
101.5 The assignment operator “ <code><-</code> ” makes object	502
101.6 You can save statistical output as an object	504
101.7 How does R save statistical output	505
101.8 Lists in R	506
102 An aside about t-tests	509
103 Working with square brackets	511
103.1 Square brackets for real	512
103.2 Difference between means using square brackets	513
103.3 Upshot: plotting output of an R statistical test	513
103.4 The “not bad way”: typing directly into R	514

CONTENTS	15
103.5The best way	518
103.6Debrief	524
104for loops in R - copy? alt?	525
104.1Introduction	525
104.2Reminders	525
104.3For loop 1: The hard-coded for loop (aka, the don't actually do this for loop)	526
104.4For loop 3: the classic for loop	529
104.5For loop 3: The Power-user for loop	530
104.6For loop 4: A for loop for all purposes	533
104.7Advanced aside: if(), else and next statements	533
104.8On avoiding for loops	535
104.9Challenge	537
105Function practice	539
105.1Part 1: Pure practice	539
105.2Practice Function 3: Make your own natural log function	540
106for() loops in R	541
106.1Introduction	541
106.2Reminders	541
106.3For loop 1: The hard-coded for loop (aka, the don't actually do this for loop)	542
106.4For loop 3: the classic for loop	545
106.5For loop 3: The Power-user for loop	546
106.6For loop 4: A for loop for all purposes	549
106.7Advanced aside: if(), else and next statements	550
106.8On avoiding for loops	551
106.9Challenge	553
107Random number generation	555
107.1Introduction	555
107.2Using set.seed() to make simulations reproducible	555
107.3Types of random number generators	557
107.4Reporting simulations	557
107.5Notes:	558
108Sample test question-Randomization	559
108.1Sample test question:	559
108.2Your answer:	559
108.3Hint 0	560
108.4Hint 1:	560
108.5Hint 2 (follow up to Hint 0)	560
108.6Hint 3:	560
108.7Answer	560

109Simulating random amino acid sequences	563
109.1Selecting random letters from a vector	563
109.2Select random letters from the whole alphabet	564
109.3Select random amino acids to build polypeptide	564
110Selecting different species and landovers from the wildlifeR package	565
110.1Introduction	565
110.2Plotting Data	569
110.3Removing an outlier3	570
110.4Plotting polished graphs	571
110.5Data modeling	572
110.6Transformations	577
111Data Cleaning: Filtering focal rows with dplyr	579
111.1Introduction	579
111.2Learning objectives	579
111.3Packages	579
111.4R code	580
111.5BBS bird count data	580
111.6AOU Species code dataframe	581
111.7dplyr and filter()	582
111.8Isolate rows from a large dataframe using dplyr's filter function .	584
111.9References	586
111.10References	586
112Data cleaning: Merging 2 large data sets with dplyr	587
112.1Introduction	587
112.2References	587
112.3Learning objectives	588
112.4Packages	588
112.5R code	588
112.6Isolate focal columns	590
113Bookdown Notes	603
113.1Inline R code	604
113.2Redact expressions or lines of code	605
113.3Do not stop on error	605
113.4Style code blocks and text output	605
113.5Embed a web page	605
113.6Emojii	605
113.7Comment out text	605
113.8Review biology through charts	606
113.9TODO	606
114Packages that could be worth learning	607

CONTENTS	17
114.Formatting output stats: xplain	607
114.Processing images: magick	607
115Allometry data	611
116Errorplot datasets	613
117error plot	615
117.1errorplot	615
117.2Health-related	618
118Datasets	619
118.1Non-linear curve	620
118.2Noisy correlation	621
118.3Regression; quantile regression ; climate change	621
118.4Logistic regression	622
118.5Regression; Fitness	622
119 x 2 table	623
119.1Heritablity	624
119.2Mendelian genetics	624
119.3Genomics	624
119.4Time series data library	624
119.5Data sets	624
120Multivariate	627
121Datasets - regression	629
121.1Repated measures	629
121.2Multiple regression	629
121.3GLM	641
122Regression	643
123Time series / climate changes	645
124Improving bas R plots: moving axes labels with mtext()	649
125Standard plot 1 panel plot	651
125.1Issues	651
125.2Change margins around plot	651
125.3Move axis labels closer to axis	652
126No tick marks & tick labels	657
127FAQ: base R	659
127.1Preliminaries	659
127.2R Plotting FAQ	660

127.3Tables	660
127.4How do I make a strip chart	670
127.5GGPLOT	671
127.6How do I get rid of the box around...	675
127.7How do I get rid of the grid lines within the plot in ggplot?	675
127.8How do I set the font of the axes titles?	675
127.9How do I adjust the positions of the axes titles?	675
127.10How do I drop, remove or suppress the legend in ggplot?	675
127.11Remove facet strip completely	676
127.12How do I increase the size text of the within facets?	676
127.13How do I increase the font size of facet labels?	676
127.14How do I reorder a factor variable for plotting in ggplot?	676
127.15Colorblind pallettes	676
127.16 Plotting FAQ	687
127.17How do I make a strip chart with base R	695
127.18	696
128Graphical parameters in R with par(): With great power comes great responsibility	699
128.1Save plotting defaults	699
128.2Data	699
128.3A single-panel plot - default	700
128.4A single-panel plot - explicitly with par()	700
128.5A vertical 2-panel plot - with par()	700
128.6A horizontal 2-panel plot - with par()	701
128.7A 4-panel plot - with par()	701
128.8A horizontal 2-panel plot - with par() and adjusted margins	701
128.9Reset defaults	702
128.10Reference	702
128.11Make some more changes to shorten the group names	703
128.12GGPLOT	705
128.13ave ggplot	721
129Basic statistics equations	723
129.1Variance	723
129.2Standard deviation	723
129.3Standard error	723
129.4Pooled variance for two-sample t-test	723
129.5Standard error of the difference	724
129.6LS regression slope	724
130Regression intercept	725
130.1Word equation	725
130.2Math equation	725
130.3Do some algebra to solve for the intercept a	725

CONTENTS	19
131F statistics (F ratio)	727
131.1Chi ² test	727
131.2Power	727
132Two equivalent ways to set up a paired t-test in R	729
133Paired T-test on 2 column data	731
134Paired t-test on one column of data	733
135Running & reporting paired t-tests in R	735
135.1Create dataframe	735
135.2Paired t-test	736
135.3Reporting the results of a paired t-test	737
136Reporting statistical result from a 2-sample t-test	739
136.1Introduction	739
136.2Outline of tasks	739
136.3Dataset up	740
136.4Plot raw data	740
136.5The hypotheses	741
136.6Do t-test	741
136.7Report the results	741
137Standard error of the difference between means	743
138Standard error for the difference between 2 means	745
139Exmple data: rat bladders	747
1401) Make data table & calculate summary stats	749
140.1The Data	749
140.2Create table of data and summary stats	749
140.3The Data table	750
1412) Function to calculate the SE of the difference	753
141.1Pooled variance	753
142Standard error of the difference	755
142.1Equation for SE of difference	755
142.2Function to calcualte SE of difference	755
142.3Apply function SE.diff	756
143Final dataframe	757
144Anscomb's quartet	759
144.1The Data	759

145Plots	761
145.1Plot y1 vs x1	761
145.2Adjust y and x limits	761
145.3Calcualte correlation of y1 and x1	762
145.4Linear regression y1 ~ x1	762
145.5Add R ² values	763
146Make two side by side plots	765
147Super scripts in Rmarkdown text, R plots, and Rmarkdown tables	769
147.1Superscript in Rmarkdown	769
147.2Plotting super scripts in R plots	769
147.3Annotate plot with text()	770
147.4Annotate plot with mtext()	770
147.5Super script in Rmarkdown table using pander()	771
148Types of data	773
149Summary tables in base R	779
150Undergraduate editors	785
150.1Winter 2021	785
150.2Original students emailed	785
150.3Original email	786
151TODO	787

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading #.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): <https://yihui.org/tinytex/>.

Chapter 2

Create book infrastructure

<https://bookdown.org/yihui/bookdown/get-started.html>

2.1 Change index.rmd

- Title
- Author
- Description

2.2 Update README.md

2.3 Update _output.yml

Changes basic features of this file as needed

2.4 Use subdirectories

To organize .rmd files in subdirectories change _bookdown.yml to include
`rmd_subdir: true`

2.5 Keep the merge file

For reference this can be useful

in _bookdown.yml chane delete_merged_file: true

2.6 Workflow

2.6.1 Build book

Use “Build book” button on Build tab of Env / Hist / Connections / Build / Tutorial

2.6.2 Clean up if needed

```
rmarkdown::clean_site(preview = FALSE)
```

2.7 Set up options for entire

2.8 Skip files

put underscore in front of file name, eg.

_skipme.Rmd

2.9 Suppress warnings etc

In chunk: message=FALSE, warning=FALSE

```
suppressMessages(library(foo))
```

2.10 Do not stop on error

<https://bookdown.org/yihui/rmarkdown-cookbook/opts-error.html>

```
1 + "a"
```

```
## Error in 1 + "a": non-numeric argument to binary operator
```

Chapter 3

Computational biology and ethics

3.1 Reading list

Guglielmim, G. 2019. Facing up to genome injustic. Nature 569.

Chapter 4

Working outline

4.1 TODO

add to compbio4all package

- frogarms
- sparrow telomeres
- gene size distribution

4.2 General book conventions

- “BOOK” = major unit; each BOOK in own folder
 - Book names are generally based on important / interesting books related to R, data science, quantitative biology, etc.
- “Part” = major subunit; each Part in own subfolder
- Chapter = individual lesson/exercise; separate .Rmd files

Many chapters have sections labeled **NEW text** followed by one flagged **OLD text**. The old text is from my previous book project and likely needs to be re-written and aligned with the new format.

4.3 Book names to use / riff off of for BOOK names

- Tufte: “The Visual Display of Quantitative Information”, “Beautiful evidence”, “Visual Explanations”, “Envisioning information”

4.4 Outline

Not necessarily up to date with current structure of books

4.4.1 BOOK 1: Introduction: Programming with data (Chambers 1998)

General info on R, bioinformatics, comp. bio., the book etc

4.4.2 BOOK 2: “Biology by the numbers” (Burton)

Actual title of Burton: “Biology by numbers”

- Plotting, data summary

4.4.2.1 Part 1: (Time series plots, and lots of bits and pieces)

Implemented

- Time series plots - base R

Not implemented

- Cumulative number of amino acids discovered
- Clines?
- Eagle data -

4.4.2.2 Part 2: Scatter plots

- -log
- Log-log

4.4.2.3 Part 3: Drawing lines

- time series
- (Lambda?)
- scatter plots
- summary
- prediction

4.4.2.4 Part 4: smoothers?

4.4.2.5 Part 5: Facetings

4.4.2.6 Part 6: Correlations

- Summarizing a scatterplot with a single number
- Anscombes quarter
- r
- R^2 ?

4.4.2.7 Part 7: Putting it all together

scatter plot with line, R, etc

4.4.2.8 Part 8: Introduction to high dimensional data

- Big data vs high dimensional data
- Scatter plot matrices
- Correlation matrices

4.4.2.9 Part 9: Distributions

- histograms
- density plot
- comparing 2 histograms
- Amino acids
- Gene length
- Bones in mammal bodies

4.4.2.10 Part 10: Summarizing distributions numerically

- mean
- median
- SD
- boxplot for 1 group?

4.4.2.11 Part 11: Boxplots

- Sparrow telomeres stuff?

4.4.2.12 Part 12: Inference by eye

- Standard error and confidence interval
- Sparrow telomeres

4.4.2.13 Part 13: P values

- betatropin case studies

4.4.3 BOOK 3: (Dimension reduction, or boiling down the numbers)

- PCA?
- Clustering
- Euclidean distance (cover this first before genetic dist?)

4.4.4 BOOK 4: Sequences

- Pairwise alignment
- pairwise alignmen -> PID etc converts sequences to numbers

4.4.5 BOOK 5: “Tree thinking”

4.5 Ideal features of book

Some of this is available for RMarkdown but not yet part of bookdown

Available - bookdown * special notices + <https://bookdown.org/yihui/bookdown/custom-blocks.html>

Available - rmarkdown * code folding * embed webpages -> google sheets, google forms, BLAST, etc * horizontal lines (“horizontal rule”)

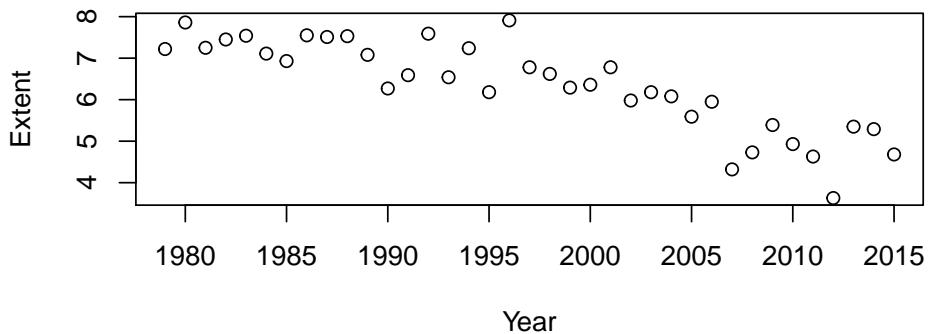
Not available / unknown * boxed text

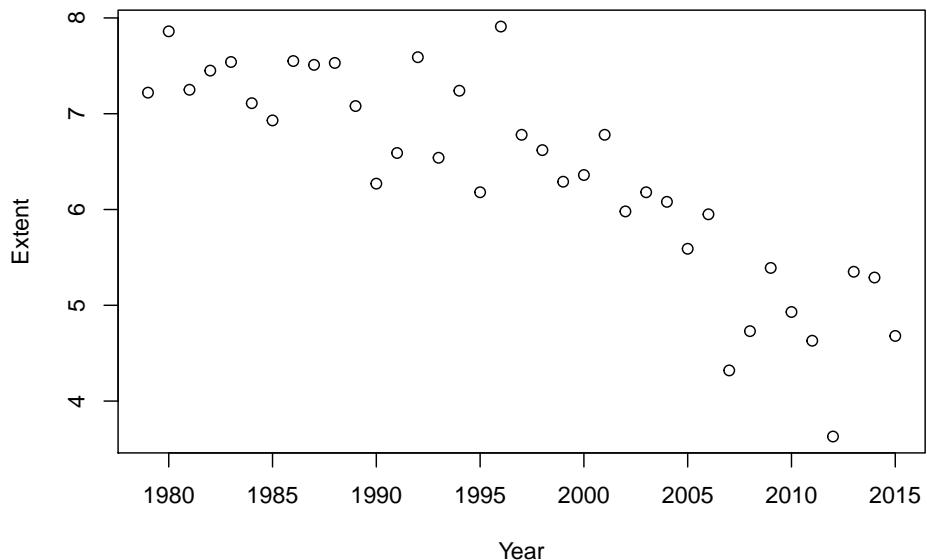
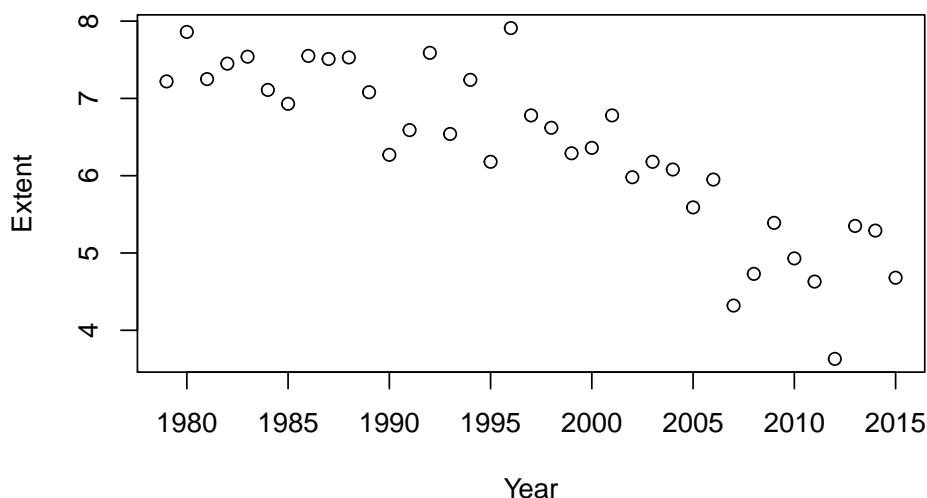
4.6 Test code

```
library(Stat2Data)
data(SeaIce)
```

4.6.1 This is a default-sized plot

Default



4.6.2 fig.height = 5, fig.width = 7**4.6.3 fig.height = 4, fig.width = 6**

Part I

Introduction

In Part I of this book I'll introduce why the role of computers in biology is important and lay out the scope of this book. I'll also walk through the basic steps of getting R and RStudio up and running on your computer, walk through a basic R session, and discuss the various ways we'll interact with R using **code** and **script files**. I'll also introduce **rmarkdown**, a way of integrating the R code of scripts with basic capabilities of word processing and web development. Scripts organized in rmarkdown are a handy way to organize and **annotate** **code**, store and disseminate your work, and structure your analyses and data presentations so that the computations behind them are fully **reproducible**.

Chapter 5

Welcome to computational biology for all!

Welcome to *Computational Biology for All!*.

This book will introduce you to key concepts of computational biology using the software R. It covers such topics as statistics, data science, bioinformatics, building phylogenetic trees, and building computer models of biological processes.

I will make only two assumptions in this book:

1. You are interested in biology and need a computer to answer a question
2. You've had some college-level biology or are willing to read some basic background information in the book, the appendices or the internet.

That's it. If you've forgotten how many amino acids are specified by the genetic code (or never learned; its 20), have never run computer code, or aren't sure what "computational biology" is no worries. We'll work through everything step by step, review often, and link to additional resources.

5.1 R and RStudio - your new best friends!

R is one of the major computer languages used by scientists. "R" refers both to the computer language itself, and to the base software which runs the computer code for us. Most people write and run their code in a special program that acts like a word processor for coding; these goes by the fancy name **Integrated Development Environments** or **IDEs**. In this book we'll use the popular IDE software called **RStudio**.

You should get access to the software combination of the software **R** and **RStudio()** either on the cloud via an account with (**RStudio Cloud**)[\[https://rstudio.cloud/\]](https://rstudio.cloud/) or on your own hard drive. (Some institutions may have

their own special implementation of R and RStudio; ask your tech support about this.)

If you are brand-new to using R the easiest way to get started is using RStudio Cloud. Getting R and RStudio set up on your own computer isn't (usually) difficult, but RStudio Cloud even easier. For information on R, RStudio, and RStudio Cloud see the Appendices. There are also many videos on the internet walking you through these topics.

If I've already lost you a bit with any of this information don't worry - we'll cover more details in the following chapters, and the Appendices cover how to get start with R, RStudio, and RStudio step-by-step.

5.2 How to use this book - be an active learner!

While you can read this as a regular book, it is meant to be an **active learning text**. That means you'll get much more out of it if you are working through all the code step by step. There are two ways to do this:

1. Read the book like a book (printed, PDF, website) and type the code in RStudio.
2. Download the associated **Active Learning Notebooks** and work through them in RStudio.

The **Active Learning Notebooks** contain **ALL** of the text and code, and is the recommended way to experience the book.

5.3 Biological scope of this book

"Computational Biology" means different things to different people, and I'll discuss how it can be defined in a later chapter. In general, though, I'll apply a very broad definition and touch upon all aspects of biology, from biochemistry to ecology. My starting point will generally by the topics classically associated with computational biology: bioinformatics, genomics, and building phylogenetic trees. Moreover, when I cover other topics like population dynamics or community ecology I'll often use molecular-biology related examples, such as population growth of transposons in our genomes and community diversity of bacteria in our guts (the so-called gut microbiome, which is studied using molecular sequencing technologies). If you're primary interest is in ecology everything in this book will be applicable at the very least in terms of the techniques, and hopefully it will help everyone expand their idea of how ecological concepts can be applied.

Chapter 6

What is Computational Biology?

This is mostly notes.

6.1 Notes

6.1.1 NIH definition

See file NIH_def_bioinfo_2000.rmd for full text of the NIH document

NOTES:

- bioinformatics relevant to all life sciences (not just biology)
- Bioinf/comp bio both “rooted” in life science, comp sci, info sci and technologies
- Bioinformatics: application focused, access / understanding from data
- Comp bio: theory/discovery approaches

Bioinformatics: Research, development, or application of computational tools and approaches for expanding the use of biological, medical, behavioral or health data, including those to acquire, store, organize, archive, analyze, or visualize such data.

Computational Biology: The development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, behavioral, and social systems

6.2 Luscombe et al 2001

N.M. Luscombe, D. Greenbaum, M. Gerstein. Bioinformatics definition committee. 2001. What is Bioinformatics? A Proposed Definition and Overview of the Field <https://www.thieme-connect.com/products/ejournals/abstract/10.1055/s-0038-1634431>

“Our definition is as follows: Bioinformatics is conceptualizing biology in terms of macromolecules (in the sense of physical-chemistry) and then applying “informatics” techniques (derived from disciplines such as applied maths, computer science, and statistics) to understand and organize the information associated with these molecules, on a large-scale.” Luscombe et al 2000

“Analyses in bioinformatics predominantly focus on three types of large datasets available in molecular biology: macromolecular structures, genome sequences, and the results of functional genomics experiments (eg expression data). Additional information includes the text of scientific papers and “relationship data” from metabolic pathways, taxonomy trees, and protein-protein interaction networks.” Luscombe et al 2000

6.3 Taprich et al 2021

Taprich et al. 2021. An instructional definition and assessment rubric for bioinformatics instruction. <https://iubmb.onlinelibrary.wiley.com/doi/full/10.1002/bmb.21361>

“An interdisciplinary field that is concerned with the development and application of algorithms that analyze biological data to investigate the structure and function of biological polymers and their relationships to living systems.”

6.4 Smith 2015 Frontiers in Genetics

Smith. 2015. Broadening the definition of a bioinformatician. *Frontiers in Genetics* <https://www.frontiersin.org/articles/10.3389/fgene.2015.00258/full>

“On a given day, I spend much of my research time staring at nucleotide sequences on a computer screen and theorizing about the evolution of genomes; thus, I feel comfortable calling myself a bioinformatician, or at the very least a scientist who primarily uses bioinformatics for his research. If asked, most of my colleagues, mentors, and students would also define me as a bioinformatician. But there is one small catch: I don’t know how to program computer software or curate databases, and I am even quite pathetic at writing UNIX

commands, which according to some precludes me from having the title of bioinformatician.” Smith (2015)

“I imagine that many of the scientists reading this essay will consider me an imposter, an amateur who points, clicks, and stumbles his way through the complicated landscape of bioinformatics. ... I believe that as a research community we need to broaden our definition of what it means to be a bioinformatician, not restricting it to only those who develop software or design and maintain data resources. ... [T]he term bioinformatician should encompass the countless and ever growing number of scientists who use computers and bioinformatics programs to address fundamental questions in biology.” Smith (2015)

6.5 Good readings / references

Koonin, EV. 20xx. Primer: Computational genomics. Current Biology - Magazine. R155-158...

6.6 Potential references

https://asistdl.onlinelibrary.wiley.com/doi/full/10.1002/asi.20133?casa_token=iDFXKee8e1gAAAAA%3AD7vtxphVMi2MHDEIvWi7Y3twQcKUvdI5Co3pF11-OgoTdk1dEHe8H5XKv67mnz1uyGyFRoMeqrbeY_Q

https://www.geneticsmr.com/sites/default/files/articles/year2017/vol16-1/pdf/gmr-16-01-gmr.16019645_0.pdf

https://ieeexplore.ieee.org/abstract/document/1678036?casa_token=H9KSA nKN-XIAAAA:R_uzPueqLDOCWccNZgdRkaRxpOo3J2RSMpVG7ZLAebTVp0-tqGl0TKQderfudEOS2efgl9oWVw https://dl.acm.org/doi/abs/10.1145/1031120.1031122?casa_token=YBJzo0bknd8AAAAA:tzp2T3uGqb1MXRPtSc cpKXAyKqtmtORa6B_yWGA9dQ9Pj4UElgeH8LK0hVJ_2PaAF51I3oqY8N-EkA

6.7 Original info, from Eco Data Science project

[NOTE: This section is currently under development. The paper by Touchon & McCoy (2016) and its references lay out many of the reasons for the statistical focus of this book and relates to all biology, not just ecology.]

“Ecological questions and data are becoming increasingly complex and as a result we are seeing the development and proliferation of sophisticated statistical approaches in the ecological literature. ... It is

no longer sufficient to only ask ‘whether’ or ‘which’ experimental manipulations significantly deviate from null expectations. Instead, we are moving toward parameter estimation and asking ‘**how much**’ and in ‘**what direction**’ ecological processes are affected by different mechanisms” (Touchon & McCoy 2016, *Ecosphere*, emphasis mine)

“Spreadsheets are often used as the basis of data collection and education; but this is potentially problematic since spreadsheets typically do not promote good data management practices.... The features of spreadsheets that make them desirable for the average researcher, such as extensibility, use of formatting for organization, embedding charts, make them undesirable for preparing data for long-term archiving and reuse.”(Strasser & Hampton 2012 *Ecosphere*)

6.8 What is data science?

Data analysis include “procedures for analyzing data, techniques for interpreting the results..., ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data.” John Tukey, “The future of data analysis”, *Annals of Mathematical Statistics*, 1962.

- People argue about what data science is
- What Tukey calls “data analysis” is now termed “data science” by many.
- Some define data science as closely allied with computer science and want its use most closely associated with things like “big data”, data mining, machine learning, and artificial intelligence.
- Others, such as RStudio’s Hadley Wickham (creator of ggplot2, dplyr, and most of the infrastructure of the tidyverse of R package) define it more broadly to involve all aspects of the life cycle of data.
- (Wickham also defines a data scientist as “A data scientist is a statistician who is wearing a bow tie” <https://twitter.com/hadleywickham/status/906146116412039169?lang=en>)

6.9 References

Strasser & Hampton 2012. The fractured lab notebook: undergraduates and ecological data management training in the United States. *EcoSphere*. <https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1890/ES12-00139.1>

Touchon & McCoy 2017. The mismatch between current statistical practice and doctoral training in ecology. *EcoSphere*. <https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1890/ES17-00013.1>

y.com/doi/abs/10.1002/ecs2.1394

Tukey. 1962. The future of data analysis. Annals of Mathematical Statistics. <https://www.jstor.org/stable/2237638>

6.10 Bibliography

Relevant papers cited by Touchon & McOy 2017.

Barraquand, F., T. H. G. Ezard, P. S. Jørgensen, N. Zimmerman, S. Chamberlain, R. Salguero-Gómez, T. J. Curran, and T. Poisot. 2014. Lack of quantitative training among early-career ecologists: a survey of the problem and potential solutions. PeerJ 2:e285.

Butcher, J. A., J. E. Groce, C. M. Lituma, M. C. Cocimano, Y. Sánchez-Johnson, A. J. Campomizzi, T. L. Pope, K. S. Reyna, and A. C. S. Knipps. 2007. Persistent controversy in statistical approaches in wildlife sciences: a perspective of students. Journal of Wildlife Management 71:2142–2144

Ellison, A. M., and B. Dennis. 2009. Paths to statistical fluency for ecologists. Frontiers in Ecology and the Environment 8:362–370.

Germano, J. D. 2000. Ecology, statistics, and the art of misdiagnosis: the need for a paradigm shift. Environmental Reviews 7:167–190.

Quinn, J. F., and A. E. Dunham. 1983. On hypothesis testing in ecology and evolution. American Naturalist 122:602–617.

Chapter 7

How will this book teach computational biology?

“The rise of computer programming, computational power, and modern statistical approaches may...” allow “...scientists to ask new questions and to extract more information from data than ever before.”
(Touchon & McCoy 2016, Ecosphere)

7.1 What you’ll learn

7.1.1 General skills that you’ll learn

- **Statistical computing** using R, RStudio, and rmarkdown
- **Data analysis**, from t-tests to mixed models in R
- **Data visualization**, with an emphasis on ggplot2
- **Data science**, from data management best practices to data cleaning with dplyr
- **Computational reproducibility**, from formatting scripts to using rmarkdown to write reproducible reports

7.1.2 Computational biology skills you’ll learn

- Working with sequence data
- alignments
- Phylogenetics
- ...

7.2 Teaching approach

- Always explore and visualize data
- Step-by-step instructions
- Frequently refreshing and review
- Comprehensive and self-contained
- Worked example
- “Modify this code” tasks
- Code completion tasks (key steps missing)
- Broken code tasks (fix non-functional code)
- Links to Jupyter notebooks (not yet implemented)
- “Active Learning Notebooks” - all content and code in .Rmd format

7.3 Requirements

- R
- RStudio
- External packages loaded via RStudio

7.4 What's not in this book / course

Chapter 8

What is R and why use it?

[these notes are from a lecture and have not been re-written much yet]

R is a powerful piece of software used for data science and data analysis. In this chapter I will briefly introduce the advantages of using R, why you might want to learn it, and also indicate some alternatives and adjuncts you could consider.

8.1 How do we typically use software in science?

Most scientists rely on both general and specialized pieces of software for various parts of their work. For data entry they likely use spreadsheet software Excel, though increasingly Google Sheets. For data analysis they might use one of many options, such as GraphPad Prism, Minitab, SAS, SPSS, or STATA. For making plots, many people will export their results back to Excel, while others use specialized software like SigmaPlot. Many scientists also use specialized programs; in ecology many researchers do GIS in ArcGIS or QGIS, mark-recapture analysis in Program MARK or Distance, use RAMAS or Vortex for population viability analysis, or build custom mathematical programs in MatLab or Python. If they do multivariate statistics like [ordination]([https://en.wikipedia.org/wiki/Ordination_\(statistics\)](https://en.wikipedia.org/wiki/Ordination_(statistics))) they may use a specialized stats program like PC-ORD.

Since software can be expensive, some scientists will rely on Excel for all of their work. Excel can do many things, but it can't do everything all the specialized types of software can do. Moreover, its very limited in the range of statistics it can do and graphs it can make.

8.2 What does R do?

R is amazing because it has been explicitly developed to do several things very well, particularly statistics, math, making great-looking figures, and writing computer programs to automate these tasks. Additionally, R has been extended by developers to be able to be a powerful tool for data cleaning and organization, to be used as a GIS, and as an integrated word processor and website make for publishing work.

8.3 Why use R

In addition to its many capabilities, R has the advantage that it is

- free anyone, always
- used by statisticians to develop new statistical techniques, so new techniques often come out 1st in R
- used by almost all ecological statisticians to develop new techniques (mark recapture, distance sampling)

8.4 Who uses it?

R continues to increase in popularity. Among data scientists it is second only to Python. Among academics it has eclipsed SAS in many fields. It is also used by analyses in many large companies, such Facebook, and by journalists looking for stories in or reporting on large volumes of data

see <http://blog.revolutionanalytics.com/2014/05/companies-using-r-in-2014.html> for further discussion.

8.5 R and computational reproducibility

One factor potentially contributing to R's popularity, or at least a major bonus for using it, is ease of use for making analyses reproducible. All commands in R are typed out and the best way to do this is in a static **script file** from which you send commands to R to execute. This creates a record of your analyses. This feature is shared by other programs such as SAS and Stats, and other programming languages such as Matlab and Python. The advantage of R is that the script files are simply plain text files which anyone can open and - if they've downloaded R, which is free - they can run. Developers have also created numerous tools for creating **reproducible analysis workflows** and which allow R to be used in all data-related aspects of a project, from **data cleaning to formatting journal submissions**. What this means is that without becoming an expert programmer you can set up your work so that you can re-run all of your data cleaning, analyses, and graph building with a

single command in R. This makes what you've done auditable, transparent, and easy to re-use for future work.

8.6 Alternatives to R

R has many advantages, but it has one critical issue: the learning curve. R is a command-line driven analysis tool, which means you type out specific commands for almost everything single thing R does. Excel is pretty user friendly, and several stats programs similarly use point-and-click interfaces, such as SPSS, JMP, and Stata SAS also requires a lot of command writing, but is generally consider more user friendly than R.

Recently, two free point-and-click statistical analysis programs have been release that are built on R but require no programming. JASP (“Just another statistics program”) has an emphasis on Bayesian statistics, particularly Bayesian hypothesis testing using Bayes factors (an approach increasing in popularity, especially in psychology, but which some Bayesians, like Andrew Gelman, disavow). While JASP is based on R, it does not currently allow access to the underlying R code.

Jamovi has a similar spirit as JASP (indeed, it was founded by developers who had worked on JASP) but is more transparent about the underlying R code being used to run the analysis.

Chapter 9

The layout of RStudio

To move forward with this book you need to get access to R. There are several ways to do this, and if you are participating in a class your instructor may have a favored way to do it.

9.1 RStudio (Desktop Or Cloud)

There are two primary ways to use R relevant to this book:

1. **Create an account at RStudio Cloud.** This is the easiest way to get started; you get 15 hours per month free which is enough for you to get your bearings. Complete instructions for doing this are available in the appendices. Note that RStudio Cloud and RStudio are essentially identical and so all instructions related to RStudio are relevant to RStudio cloud.
2. **Download R AND RStudio desktop.** Key here is that you need TWO pieces of software, R and RStudio. Complete instructions are in the appendices.

These two methods are essentially equivalent. To jump in and get started I recommend RStudio cloud. See the appendices for step by step instructions.

9.2 R emulators

There are also two other ways of using R which we'll occasionally use. I mention them here briefly. They essentially emulate the R experience but eliminate the hairy details.

TODO: move to appendices?

1. **Shiny Apps:** These are independent mini-programs that allow you to explore or carryout some aspect of R functionality without doing any coding. They typically live on the internet, but can also be brought up from RStudio desktop. An example of a simple Shiny App on the internet is (here)[<https://shiny.rstudio.com/gallery/faithful.html>]: (<https://shiny.rstudio.com/gallery/faithful.html>) [<https://shiny.rstudio.com/gallery/faithful.html>]
2. **leanr tutorials:** These are interactive tutorials where you do basic coding in your web browser. Here's an example: <https://learnr-examples.shinyapps.io/ex-data-filter/>

9.3 Other ways

Two other ways you may encounter R

1. **Use a Jupyter notebook:** Jupyter notebooks are an advanced R emulator. They are really cool, allowing you to essentially combine text (like a tutorial) with code that can be run within a web browser.
2. **Use regular old R:** When you download R an icon will appear on your desktop. You can access old-school R that way, but almost no one ever does. For a bit more information about this see the appendix.

9.4 RStudio at a glance

Now we'll get started with RStudio. We'll get to know what it looks like and configure it a bit for our needs.

If you are using RStudio desktop the RStudio logo looks like this.

TODO: INSERT IMAGE

Whether on the Cloud or Desktop, when you open up you'll be greeted by a fairly busy array of menus and things. Don't panic! A typical fresh starting point in RStudio/RStudio Cloud is shown in Figure 2.

INSERT IMAGE

When referring to RStudio (and equivalently RStudio Cloud - this is the last time I'll mention this so hopefully get the point), there are two terms that need to be understood. As shown in Figure 3, there is 1) the **console** section of RStudio and 2) the **script editor or source viewer**.

INSERT IMASGE

(A "cheat sheet" called the "RStudio IDE Cheat Sheet" details all of RStudio's many features and is available at <https://www.rstudio.com/resources/cheatsheets/>

ets/. It very thorough, though a bit dense. I don't recommend it for beginners but you should remember that it exists.)

9.4.1 The console versus the script editor

You can type and enter text into both the **console** and the **script editor**. The console, however, responds actively like a calculator, while the script editor works more like a text editor. Information can be passed unidirectionally from the script editor to the console, but not the other way.

9.4.1.1 The R console

The **console** in RStudio gives you **interactive programming** environment that is very similar to a scientific calculator. If you click your mouse inside the console and type `1 + 1` then press enter you will see the following type of output

```
1 + 1
```

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

Note that right in front of where you typed `1+1` there is a `>` symbol. This is always in the R console and never needs to be typed. You may occasionally see it printed in books or on websites, but it doesn't ever need to be typed.

One thing to note about R is that it's not particular about spacing. All of the following things will yield the same results

```
1+1  
1 + 1  
1      +      1  
1 +                 1
```

Got it? Awesome! Now we're ready for some real data analysis.

Part II

Getting started with R

NEW text

Part 2 of the book we'll do get started using R. Like any language R is a versatile tool for communication, but has conventions, quirks, idioms and dialects that new users have to become comfortable with. In general my goal in this section will be to help you get a general sense of R, let you see a bit of the wild – and overwhelming – world of possibilities for how R is written, and outline the more specific aspects I'll use to help facilitate learning.

We'll have three specific goals:

1. Run some initial, simple commands in R to see how it works.
2. Take a broad tour of the wide world of R so see the many faces of R code you may encounter in the wild.
3. Highlight the particular way coding conventions and idioms of R I'll use.

If you find yourself getting confused or overwhelmed don't worry. A goal of this section is to prevent *future* confusion by giving you a sense for the many different ways R can be written and that you will eventually encounter on the internet or other books, and to contrast them with what you'll use in the book. In this book I've worked to keep things consistent and to either use the simplest methods to accomplish the goal or to carefully break down hard tasks or concepts so you can master them. Once you start typing into your search engine "R code ..." you will get MANY different types of code which you may not yet be prepared to work with.

OLD text

In this we'll discuss two of the most fundamental steps of data analysis: getting your hard-earned data into the @\\$*%# program you want to do you analysis in. This is often one of the most frustrating steps for beginners because R, like most data analysis tools that aren't spreadsheets, it seemingly very picky about how it wants to receive data. Luckily there's a method behind the madness, and also some handy tools in RStudio to help with this.

Most data starts off as a spreadsheet before it enters R. Loading spreadsheet data into R will be our end goal, but as a run up will step through several easier tasks that will highlight the core principles of getting data into R, in particular loading additional software into R (called packages) and loading the data in those packages into R. We'll also load data directly from the internet into R.

The chapters in this section are cover the following:

1. Loading R packages to get new software into R
2. Loading and looking at data from R packages
3. Loading packages from the software repository GitHub
4. Loading data from the internet
5. Loading spreadsheets as .csv files

6. Loading Excel spreadsheets

Chapter 10

Hello R! A simple R plotting exercise

10.1 Key Ideas

- Commands: R uses simple typed commands to do everything
- Data: loading data that comes with R using the `data()` command.
- Plotting: Making simple plots with the `plot()`

10.2 Data in R

We'll start our exploration of R with an classic dataset from ecology showing one of the most striking patterns in wildlife biology: population cycles. Populations are always changing, whether it declines in the number bacteria in our gut after we take an antibiotic or increases in raptor species after highly toxic pesticides were banded in the mid-twentieth century.

A few populations show dramatic **oscillations**, with rapid increases soon followed by dramatic drops, as if the population were on a roller coaster. One such population are Canada lynx (*Lynx canadensis*).

The table below shows a snapshot of these data

```
Quitting from lines 756-761 (compbio4all-book.Rmd) Error in check_caption(caption)
: The caption should be string (character class) or NULL. Calls: local ... pandoc.table -> cat -> pandoc.table.return -> check_caption
```

You've probably plotted data like this by hand using graph paper, point by point by locating the x-y coordinates. In a spreadsheet, you could highlight these columns and click on the "Make graph" icons to make the initial plot, then adjusting things by clicking on parts of the plot you want to change.

```
# "If you're not familiar with this process..."  
# Add example of excel plot  
# TODO: add image? video?
```

Spreadsheets are said to operate under the principle of “**What You See Is What You Get**”, or **WYSIWIGY**. They use a fully mouse-drive **Graphical User Interface (GUI)** where everything is done by pointing and clicking. Every time you make a plot you do these steps,

R is *very* different - you only see things when you want to see them and you do everything via typed commands. This is a large paradigm shift for most people, so we’ll start very very slow.

10.3 Loading data that comes with R

The Canada lynx is not just famous with ecologists but also familiar to statisticians who have frequently used it to test statistical methods for studying **time series** - basically long-term datasets of the same thing. Because of this, some lynx data is embedded within R and easy to access: all you have to do is type `data(lynx)` into the console and press the “Enter” key.

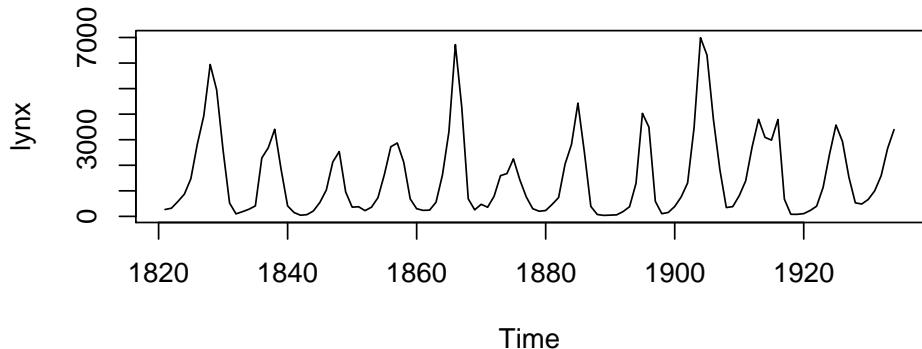
```
data(lynx)
```

You might be wondering “Ok, now what?” because nothing apparently happened. What you’ve done is loaded the `lynx` data into R’s active memory, which it will wait for your next command.

10.4 Plotting simple datasets

Now in the console type `plot(lynx)` and press enter. You should see readout like what you see below ...

```
plot(lynx)
```



... and this plot.

This is a simplified example, but that's the basics of working in R:

1. Load data
2. Use commands like `print` tell R to do something with.

For most of this book we'll use data that's been mostly pre-packaged for you to work with and loaded using the `data()` command. Real data analyses require more steps, and later in the book we'll briefly cover them so you are familiar with them when you see them elsewhere; further details can also be found in the appendix.

10.5 Commands in R

The words “data” and “plot” in R represent **commands**; R associates specific code therefore actions with it. To indicate commands in the text I'll always write it like this: `data()`. The parentheses are important in R. After the word representing the command there is always a **parenthesis** (. Other things go after the parenthesis, and the command is completed with a matching parenthesis.). To emphasize that things using go within the parentheses I will often write commands like this `data(...)` where the ... in this case is the name of a dataset.

In some cases you can issue a command, like `data()`, and R does something only behind the scenes. Often, though, we'll elicit a reaction from other, either data will appear in the Console or a plot will be created.

Our use of the `plot()` command was pretty standard; there were two pieces to it:

1. The command, `plot()`
2. The data, `lynx`

Data in R – and especially in computational biology - can take on many forms which we'll cover as throughout the book. All data is presented in R by an **object** stored behind the scenes in R's memory. The fact that data in R is usually resting out of view until we do something explicitly with it can take some getting used to, since usually we work with data printed out on a page or displayed in a spreadsheet.

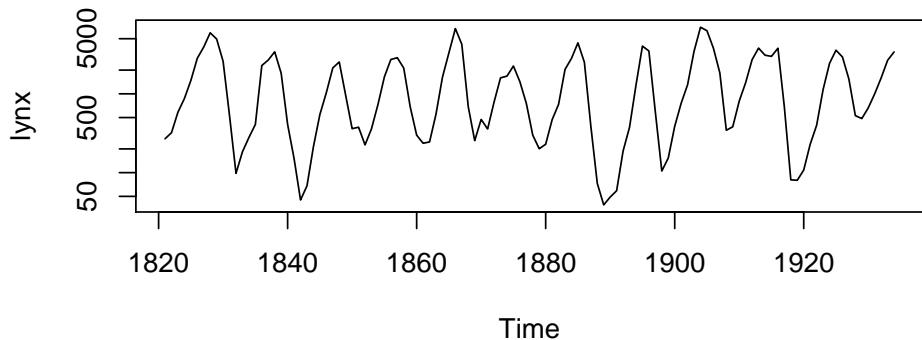
```
### TODO: show picture,
## left - spreadsheet of data
## right - blank screen
## "data ready to be plotted in in a spreadsheet versus R
```

Commands in R almost always include an object within them. Next we'll consider something else that commands : **arguments**.

10.6 Arguements

A common mathematical operation when doing data analysis is taking the log of something. (For now we won't worry about what the log is or why we use it' we'll come back to this little bit of math frequently though). We can tell R to plot the log of our lynx data by adding the argument `log = "y"` to the `'plot(...)` command. This alters the graph a bit which, for particular. data analysis purposes will come in handy (more on that later).

```
plot(lynx, log = "y")
```



Arguments always have an equals sign with them, so I'll emphasize this by typically writing them as `argument =`. One tricky thing about arguments is that they can take on letters, words, or numbers, and sometimes there need to be quotation marks like `log = "y"`, but not always.

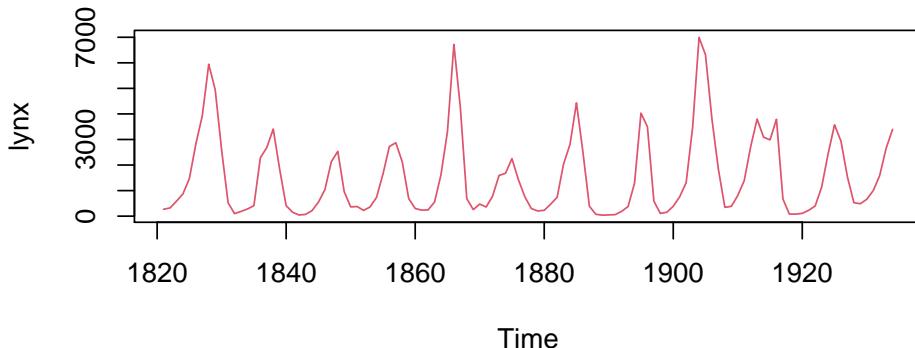
10.7 Arguements and more arguments

We now have three things going on

1. A command
2. A data object (`lynx`)
3. An argument, `log = "y"`

Most functions in R have multiple arguments that can be invoked Try the following code `plot(lynx, col = 3)`. That is the `plot()` function with the argument `col = 3` added. What do you think `col = 2` means? Try different values like 4, 5, and 6.

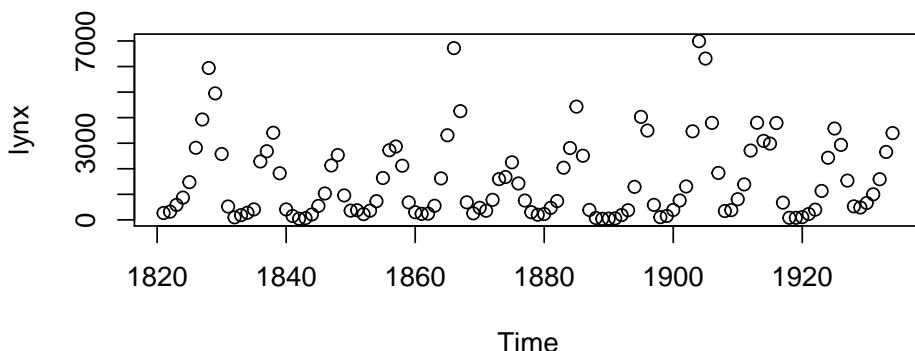
```
plot(lynx, col = 2)
```



Now try this: `plot(lynx, type = "p")`

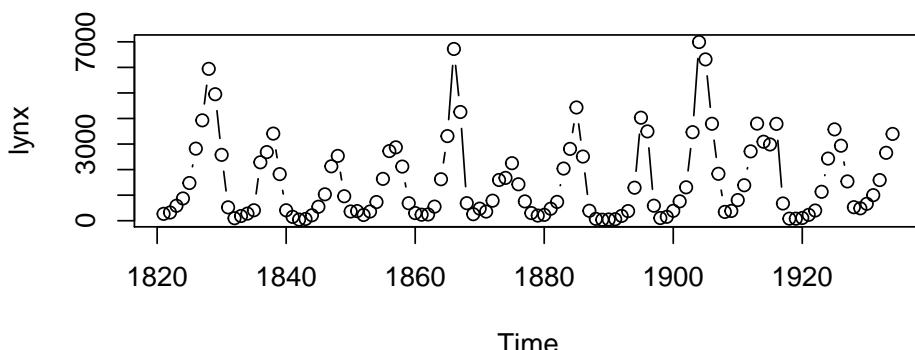
Note that there are quotation marks around the p.

```
plot(lynx, type = "p")
```



Now instead of "p" use "b", which stands for "both". What do you think the "both" is referring to?

```
plot(lynx, type = "b")
```



```
# add assignment
```

```
## TODO: add follow exercises
### R, biology

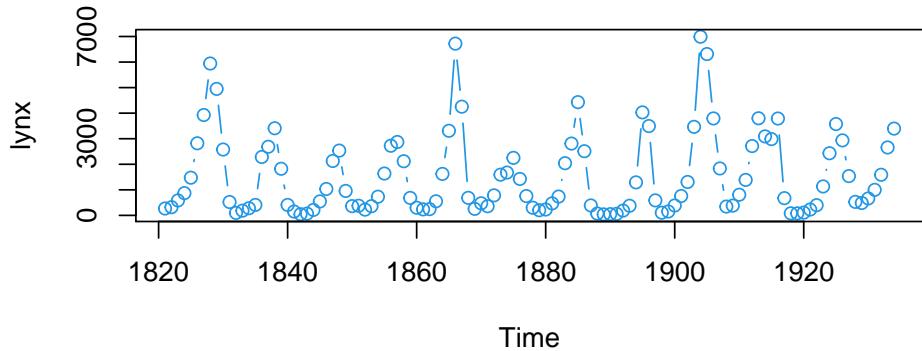
## TODO Add as optional? cover elsewhere

plot(log(lynx))
```

10.8 Multiple arguments at the same time

Functions can not only have multiple arguments, but they can take on multiple arguments at the same time.

```
plot(lynx, col = 4, type = "b")
```



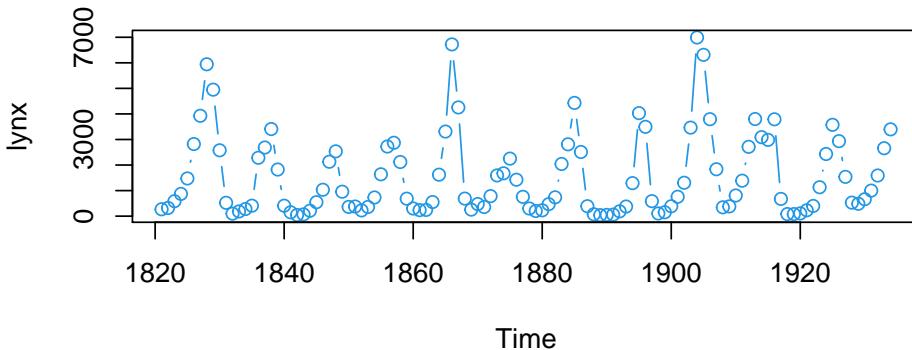
One thing about R is that it doesn't care about **line breaks** within a command, so I do this if I want

```
plot(lynx,
     col = 4,
     type = "b")
```

10.9 Comments

One thing that putting things on multiple lines allows you do to is add **comments** to your code if you place a hashtag (aka pound symbol) in front of it.

```
plot(lynx,          # data object
     col = 4,        # color argument
     type = "b")    # type of graph argument
```

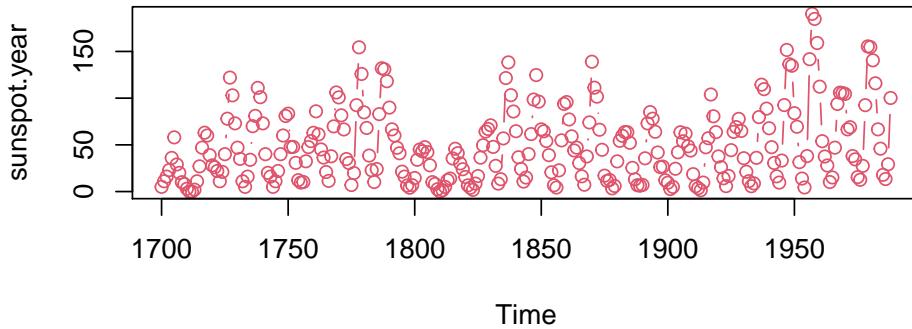


10.10 Now you try it

Here's a challenge: there is another dataset that comes with R called `sunspot.year` (There was a hypothesized link between the Canada lynx and sunspots that we'll explore later). See if you can do the following things on your own in the R console

1. Using the `data()` command, load `sunspot.year` data into R's active memory
2. Use the `col =` argument to make the color of the line different than black.
3. Use the `type =` argument to make the plot show points connected by a line.

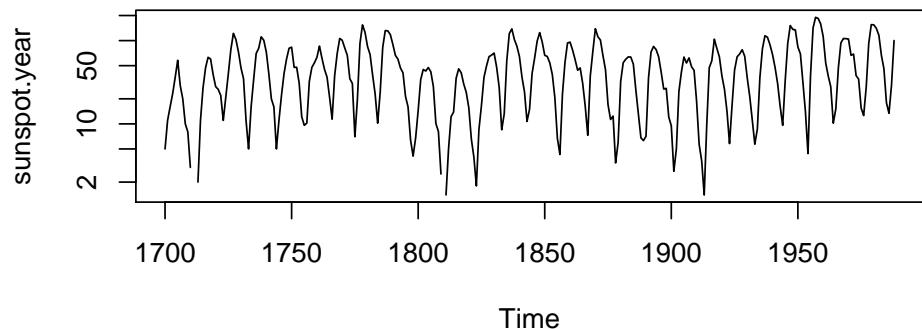
Your plot should look like this



10.11 A note on plotting

One of the great things about R is that it can make really nice plots. You'll soon see that there are many ways to do the same basic thing in R, and this includes making plot. **Data visualization** is a key aspect of modern science, so its important to build up your skills, including knowing about the different ways plots are made in R. Don't worry though - we'll build all of this up step by step.

```
# With a warning  
plot(sunspot.year, log = "y")
```



Chapter 11

Downloading R packages (and their data)

11.1 Loading data from R packages

NEW text: Base R however is surrounded by a universe of extensions built by statistician, programmers, academics and businesses that use R for analyses. Some of these are fairly standard and are downloaded along with base R and just need to be explicitly installed. Other have to be downloaded from the internet and installed. Most packages contain data in order to demonstrate what they do.

Old text: When you install R you get *Base R**, which is the core set of functions, functionality, and some data sets. Base R however is surrounded by a universe of extensions built by statistician, programmers, academics and businesses that use R for analyses. A lot of R's functionality is found in these packages, including data sets, special plotting functions, and statistical tools for the analysis of complex data. Some of these are fairly standard and are downloaded along with base R and just need to be explicitly installed. Other have to be downloaded from the internet and installed. Most packages contain data in order to demonstrate what they do; working with data from packages will be covered in a later lesson.

This book relies heavily on an R package I've written called "combio4all" (<https://brouwern.github.io/combio4all/>) that contains the datasets used throughout the book, as well as some helpful R functions I've written.

Most R packages you'll use are stored on the CRAN website where you download R (<https://cran.r-project.org/>). R and RStudio have functions and tools for downloading and managing packages that we'll briefly introduce in this exercise.

Another place a package can be stored online is a code repository like GitHub. The `wildlifeR` package lives on GitHub and can be downloaded directly from there. Many packages on CRAN also occur on GitHub, especially if programmers are actively developing, updating, and managing the package. We'll cover downloading packages from GitHub in the next exercise.

11.1.1 Functions & Argueuments

- `install.packages`
 - `dependencies = TRUE`
 - `library`
-

11.2 OPTIONAL: What functions come with base R?

The following section is optional

If for some reason you want to see *all* the functions that come with base R, type this into the console and press enter. (`ls` stands for “list” and is a function we’ll use more later).

```
ls("package:base")
```

As R has been developed there has also built up a cannon of tried and true packages that are downloaded automatically when you download R, but they aren’t brought into R’s working memory unless you tell R.

11.3 OPTIONAL: What packages come with base R?

If you want to see all of the packages that come with base R, do this.
`library()` is a function you will use a lot.

```
.libPaths(" ")
library()
```

One package that is part of this cannon is MASS, which stands for Modern Applied Statistics in S. “S” is the precursor to R, and MASS is the package that accompanies the book of the same name, which is one of the original books on S/R. (<https://www.springer.com/us/book/9780387954578>)

End optional section

11.4 Load data from an external R package

Many packages have to be explicitly downloaded and installed in order to use their functions and datasets. Note that this is a **two step process**: 1. Download package from internet 1. Explicitly tell R to load it

11.4.1 Step 1: Downloading packages

There are a number of ways to install packages. One of the easiest is to use `install.packages()`. Note that it might be better to call this “`download.packages`” since after you install it, you also have to load it!

Well download a package used for plotting called `Stat2Data`.

```
install.packages("Stat2Data")
```

Often when you download a package you'll see a fair bit of red text. Usually there's nothing of interest here, but sometimes you need to read over it for hints about why something didn't work.

```
For example, when I downloaded this package I got this cryptic message in bright red text: trying URL 'https://cran.rstudio.com/bin/macosx/contrib/4.0/Stat2Data_2.0.0.tgz'  
Content type 'application/x-gzip' length 1177728 bytes (1.1 MB)  
===== downloaded 1.1  
MB
```

Followed by this in less insistend black: The downloaded binary packages are in /var/folders/q8/gwjfr69n05vf4h15l6hdl4d8x1zk5v/T//RtmpeRHx2y/downloaded_packages

This is all perfectly normal.

11.4.2 Aside: don't re-download packages all the time

TODO: explain

You typically only need to do this once. Or occassionally.

11.4.3 Step 2: Explicitly loading a package

The `install.packages()` function just downloads the package software to R; now you need to tell R explicitly “I want to work with the package”. This is done using the `library()` function. (Its called library because another name for packages is “libraries”).

```
library(Stat2Data)
```

As frequently is the case, R doesn't look like its doing much, but you've actually just installed a bunch of cool datasets.

11.5 Optional: Seeing all of your installed packages

The following section is optional

If for some reason you want to see everything you've downloaded, do this.

```
installed.packages()
```

End optional section

11.6 Downloading packages using RStudio

RStudio has a point-and-click interface to download packages. In the pane that says “Files, Plots, Packages, Help, Viewer” click on “Packages”. When the panel shift below “Packages” it will say “Install, Update, Packrat.” Click on “Install.” (There might be a lag during this process as RStudio get info about your packages). In the pop up widow there will be a middle field “Packages” where you can type the name of your package. There’s an auto-complete feature to help you in case you forget the name. Then click “install.” Note that in the bottom right corner of the pop up is a checked box next to “Install dependencies.” Leave that checked; more on that later.

I don’t do this but include any download information in script

Chapter 12

Data in dataframes

```
library(Stat2Data)
```

An interesting dataset in Stat2Data is SeaIce. Load it with the `data()` command.

```
data(SeaIce)
```

SeaIce shows 37 years of the area of frozen ice in the arctic, from 1979 to 1993. The lynx data we worked with previously was in a special format that you'll probably rarely encounter ever again. It was nice to use, however, because it required very little code to plot.

SeaIce, however, is a typical R data object in the form of a **dataframe**. Dataframes are fundamental units of analysis in R. Most of the data you will load into R and work within R will be in a dataframe. They have the same basic structure as a spreadsheet, but R keeps them hidden in memory and you have to use commands to explore them.

12.0.1 Looking at dataframes with `View()`

To get a spreadsheet-like view of a dataframe you can use the `View` command

```
View(SeaIce)
```

This will bring up the data in a spreadsheet like viewer as a new tab in the script editor, similar to this.

Year	Extent	Area	t
1979	7.22	4.54	1
1980	7.86	4.83	2
1981	7.25	4.38	3

Year	Extent	Area	t
1982	7.45	4.38	4
1983	7.54	4.64	5
1984	7.11	4.04	6
1985	6.93	4.18	7
1986	7.55	4.67	8
1987	7.51	5.61	9
1988	7.53	5.32	10

Note: Unlike a spreadsheet you cannot edit the data when it is called up using `View()`

Like a spreadsheet the data are organized in columns and rows. Each **column** represents a type of information:

- **Year:** when data were collected
- **Extent:** the amount of area within the ice-bound region
- **Area:** total area of ice, minus any non-ice area (land, melted water)
- **t:** time point, from 1 (1979) to 15 (1993)

You can think of **Extent** as similar to the size of a country, and **Area** as the actual amount of land in a country minus any lakes.

Each row represents a different year of data; row 1 is the **Extent** and **Area** for 1979, row 2 is the **Extent** and **Area** for 1980 and so on.

12.0.2 Looking at dataframes in the console

Another common way to examine data is simply type the name of the data in the console and press enter. This prints it out; however, if its a large data frame this may take up a LOT of room. (I'll just show an exert here).

`SealIce`

```
##   Year Extent Area t
## 1 1979    7.22 4.54 1
## 2 1980    7.86 4.83 2
## 3 1981    7.25 4.38 3
## 4 1982    7.45 4.38 4
## 5 1983    7.54 4.64 5
## 6 1984    7.11 4.04 6
## 7 1985    6.93 4.18 7
## 8 1986    7.55 4.67 8
## 9 1987    7.51 5.61 9
## 10 1988   7.53 5.32 10
## 11 1989   7.08 4.83 11
## 12 1990   6.27 4.51 12
```

```
## 13 1991   6.59 4.47 13
## 14 1992   7.59 5.38 14
## 15 1993   6.54 4.53 15
```


Chapter 13

Plotting data in dataframes

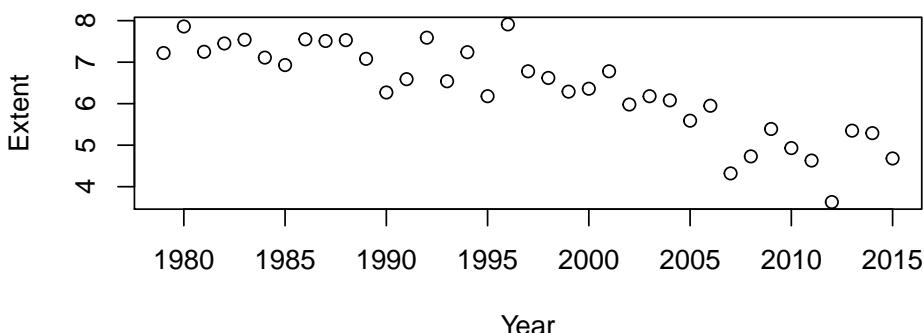
```
library(Stat2Data)
data(SeaIce)
```

Most data in R are organized into dataframes. Similar to when we plot data in a spreadsheet, to plot data from a dataframe we need to tell R exactly what we want on the **x-axis (horizontal)** and **y-axis (vertical)**.

Note: For reasons we don't have to get in to, the `lynx` data were a special case where we didn't have to define `x` and `y`

We can plot the Extent of artic sea ice again using the `plot()` command, and using a cool convention in R called **formula notation**. Formula notation uses the **(tilde)**[<https://en.wikipedia.org/wiki/Tilde>] symbol `~`. In math, `~` can have several meanings. In R, it means “relates to”, “versus”, “depends on.” So we can plot the relation between `Year` and seac Extent as a **y versus x** relation as `Extent ~ Year`. We also have to include the argument `data = SeaIce` so R knows where to get `Extent` and `Year`.

```
plot(Extent ~ Year, data = SeaIce) # Note, both words capitalized.
```



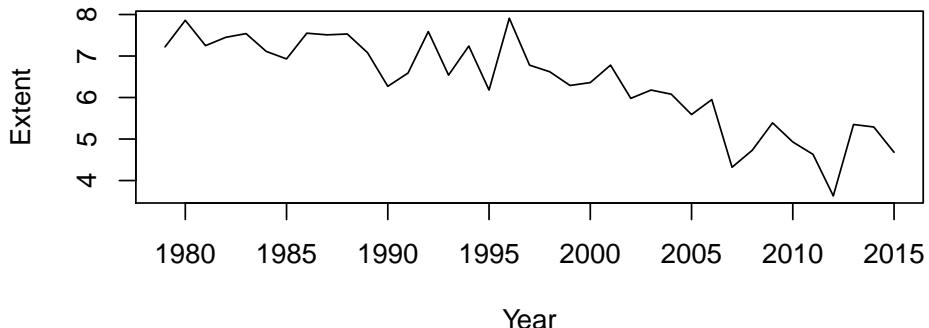
13.1 Base-R graphics

When we use the `plot` command were using **Base R** graphics. As noted before there are several ways to make plots in R and you should be able to spot which one is which when looking at code. We'll cover some of the key features of Base R graphics now. While different plotting methods have different commands and arguments, they all share a common feature: everything in a plot can be customized, and each element is customized with a command or argument.

13.1.1 Type of points

`plot()` can draw dots or lines. We make it use lines using the `type = "l"` argument (note that the l is in quotes)

```
plot(Extent ~ Year, type = "l", data = SeaIce) # Note: l in quotes
```



As noted before, R doesn't mind if you split things on lines. To keep track of the things I'm doing to the plot I'll format things like this

```
plot(Extent ~ Year, # relationship
     type = "l",      # type of plot; Note: l in quotes
     data = SeaIce)   # data
```

As we did with the `lynx` data we can combine points and lines with `type = "b"`. (Do you recall what "b" stands for?)

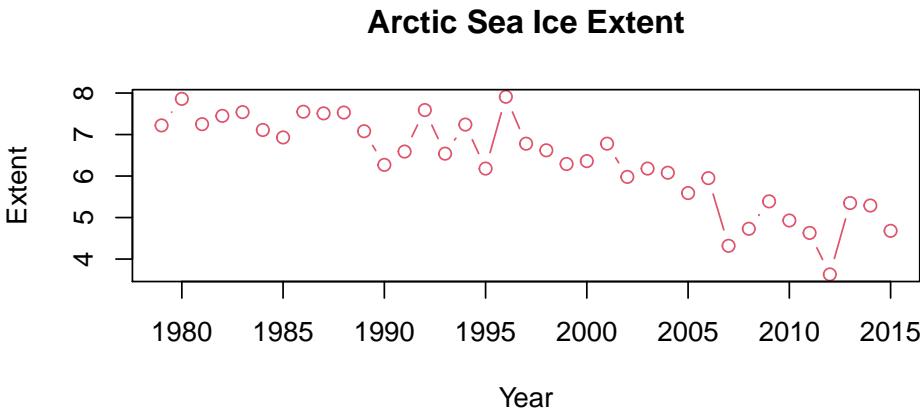
```
plot(Extent ~ Year, # relationship
     type = "b",      # type of plot; Note: b in quotes
     data = SeaIce)   # data
```

We can adjust color with `col =` Recall the this is just a number, not in quotes.

```
plot(Extent ~ Year, # relationship
     type = "b",      # type of plot; Note: b in quotes
     col = 2,          # color; no quotes
     data = SeaIce)   # data
```

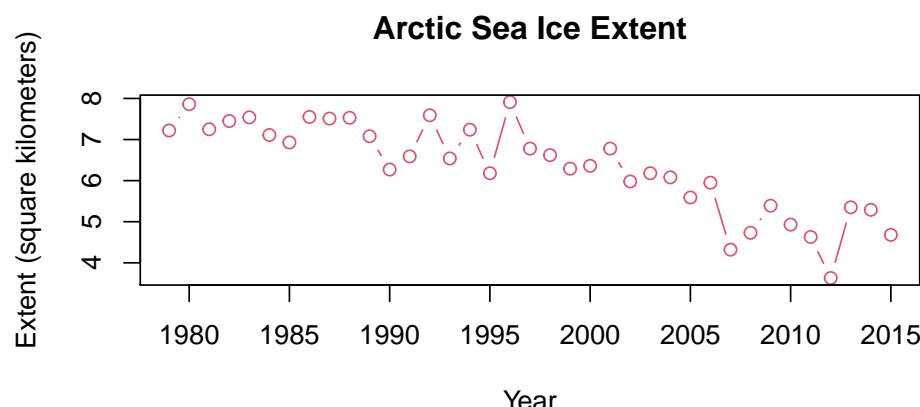
We can add a main title to with `main = ...`

```
plot(Extent ~ Year, # relationship
     type = "b",      # type of plot; Note: b in quotes
     col = 2,          # color; no quotes
     main = "Arctic Sea Ice Extent", # main title, in quotes
     data = SeaIce) # data
```



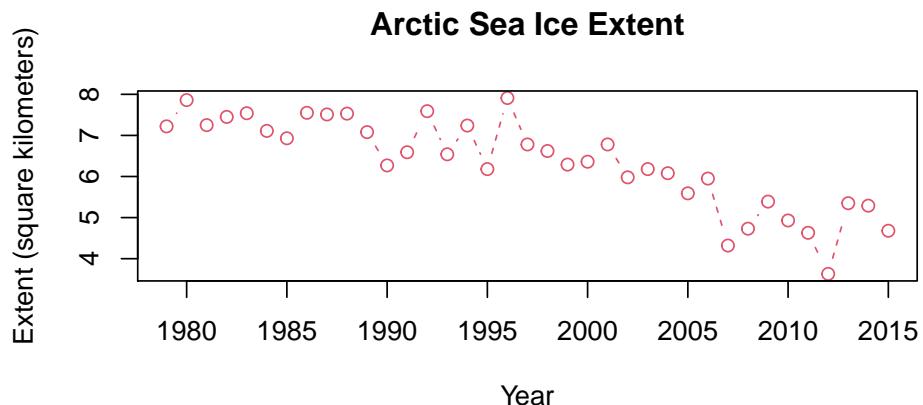
Its always good to include your units. `Extent` and `Area` are in square kilometers. We can say specifically what we want for the y-axis label using `ylab = ...`

```
plot(Extent ~ Year, # relationship
     type = "b",      # type of plot; Note: b in quotes
     col = 2,          # color; no quotes
     main = "Arctic Sea Ice Extent", # main title, in quotes
     ylab = "Extent (square kilometers)", # y-axis label
     data = SeaIce) # data
```



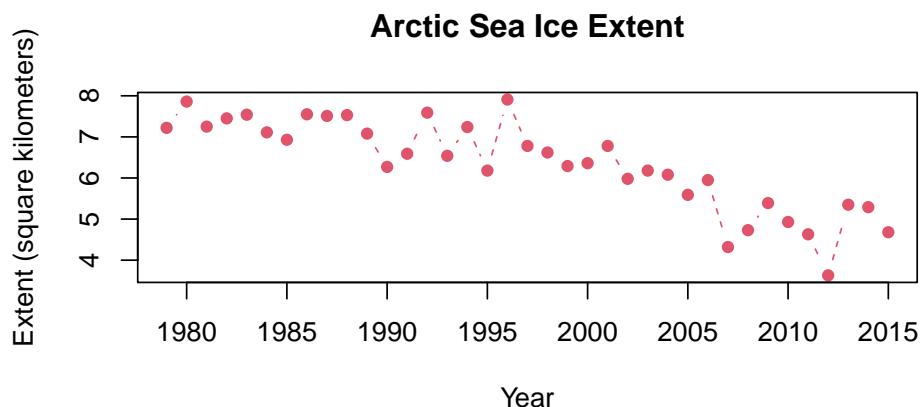
We can change the appearance of the line using `lty = ...`, which stands for “line type”:

```
plot(Extent ~ Year,    # relationship
     type = "b",        # type of plot; Note: b in quotes
     col = 2,           # color; no quotes
     main = "Arctic Sea Ice Extent", # main title, in quotes
     ylab = "Extent (square kilometers)",
     lty = 2,           # line type; not quoted
     data = SeaIce)    # data
```



I'm not a fan of the open circles for plotting points; we can change those too using the argument `pch =`.

```
plot(Extent ~ Year,    # relationship
     type = "b",        # type of plot; Note: b in quotes
     col = 2,           # color; no quotes
     main = "Arctic Sea Ice Extent", # main title, in quotes
     ylab = "Extent (square kilometers)",
     lty = 2,           # line type; not quoted
     pch = 16,          # point type; not quoted
     data = SeaIce)    # data
```



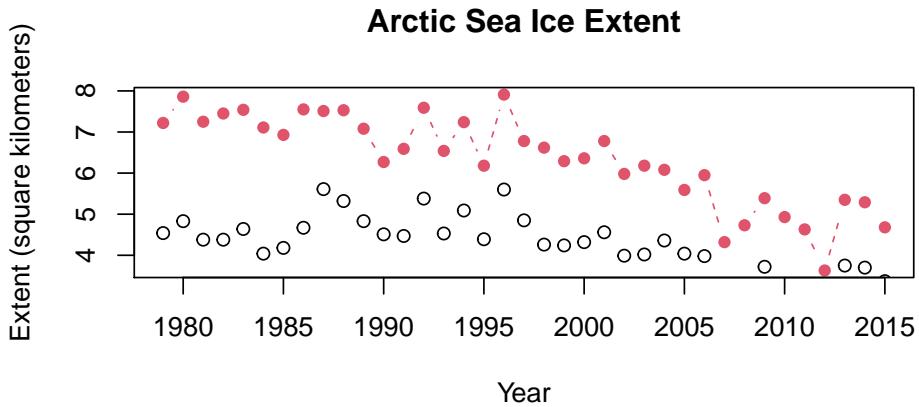
13.1.2 Plotting two columns of data

Often we want to represent two distinct things on our graph. In spreadsheets these are called separate **series** of data. When making a **time series plot** like this one we can add a new column of data using a species command called **points()** which works very similar to **plot()**.

Note: The **points()** command only works if its it precede by a statement from the **plot()** command

```
# Main plot: Extent
plot(Extent ~ Year, # relationship
     type = "b",      # type of plot; Note: b in quotes
     col = 2,         # color; no quotes
     main = "Arctic Sea Ice Extent", # main title, in quotes
     ylab = "Extent (square kilometers)", 
     lty = 2,         # line type; not quoted
     pch = 16,        # point type; not quoted
     data = SeaIce) # data

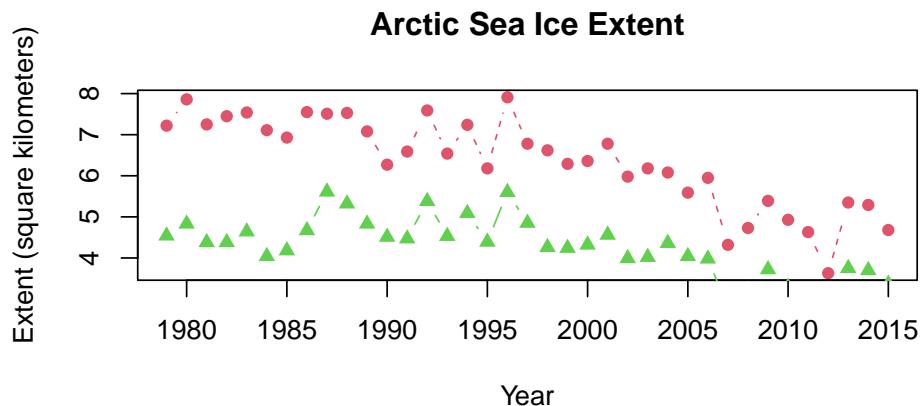
# Second column of data: Area
points(Area ~ Year, data = SeaIce)
```



I can now customize the sea ice **Area** part of the plot by add **arguements** to the **points()** statement.

```
# Main plot: Extent
plot(Extent ~ Year, # relationship
     type = "b",      # type of plot; Note: b in quotes
     col = 2,         # color; no quotes
     main = "Arctic Sea Ice Extent", # main title, in quotes
     ylab = "Extent (square kilometers)", 
     lty = 2,         # line type; not quoted
     pch = 16,        # point type; not quoted
     data = SeaIce) # data
```

```
# Second column of data: Area
points(Area ~ Year,
       type = "b",
       col = 3,
       pch = 17,
       data = SeaIce)
```



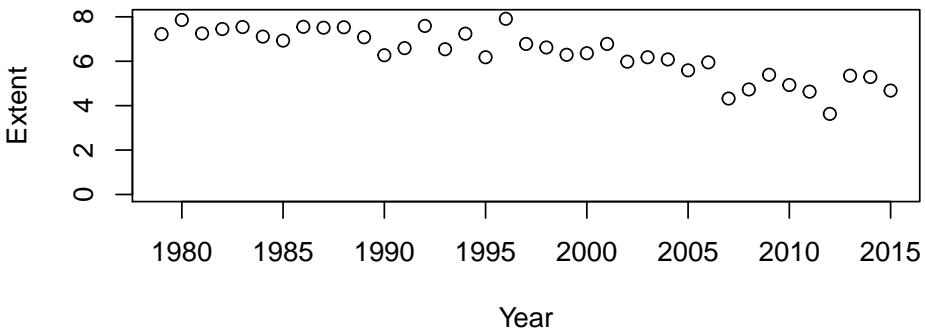
There's a problem with my graph though - can you spot it? It only really becomes apparent when you connect the dots with a line.

13.1.3 Changing the range of a plot axis

Let's go back to our original plot and forget about all the fancy arguments and adding `points()` for a little bit. The problem with the last graph we made is that some points are not showing - if you look in the lower right-hand part points around 2006-2008 and 2011-2013 are not showing up because the y-axis stops around 3.5. We can fix this by adding a new argument which sets the limits of the y-axis: `ylim =` To do this correctly, we have to introduce a new function: `c()`. This is actually one of the most common functions in R. In the `c()` we need to tell R the lower and upper limits we want for the y-axis. Let's do 0 and 8, which will be coded as `c(0,8)`.

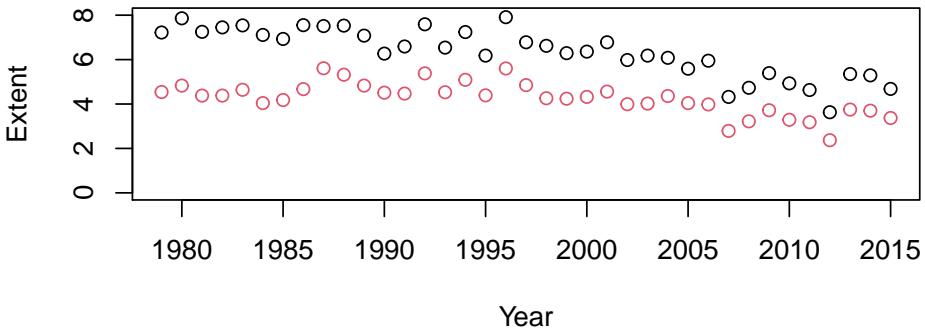
So, to set the y-axis limits we do this:

```
plot(Extent ~ Year,
      ylim = c(0,8), # the c(...) function to set limits
      data = SeaIce)
```



Now we have a bunch more space at the bottom. We can add our `points()` back to see if this work:

```
plot(Extent ~ Year,
      ylim = c(0,8), # the c(...) function to set limits
      data = SeaIce)
points(Area ~ Year,
       data = SeaIce,
       col = 2)
```



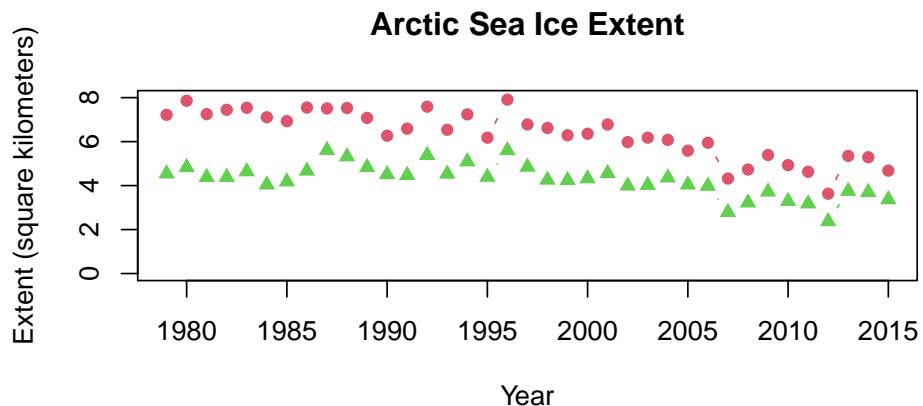
Note: `ylim = ...` only goes in `plot()`, not `points()`. View()

Let's re-make our fancy plots now with `ylim = ...` set.

```
# Main plot: Extent
plot(Extent ~ Year,
      type = "b",
      col = 2,
      main = "Arctic Sea Ice Extent",
      ylab = "Extent (square kilometers)",
      lty = 2,
      pch = 16,
      ylim = c(0,8), # <#<== y-axis limits
      data = SeaIce)

# Second column of data: Area
```

```
points(Area ~ Year,
      type = "b",
      col = 3,
      pch = 17,
      data = SeaIce)
```



13.2 You try it

13.2.1 Fixer-uppers

Fix the code below so it works

```
plot(Extent , Year,
     data = SeaIce)
```

Fix the code below so it works

```
plot(Extent ~ Year, # relationship
     type = b,       # type of plot; Note: b in quotes
     col = "2",       # color; no quotes
     data = SeaIce) # data
```

13.2.2 Intermediate

Make a plot with `Area` on the x-axis and `Extent` on the y-axis.

```
## Write the code below
```

13.2.3 Advanced

Based on the code above, make a plot of the `SeaIce` data where `Area` appears within the `plot()` statement, and `Extent` is in `points()`.

Write the code below:

Chapter 14

Build your own dataframe part I: Vectors

14.1 Dataframes are a type of data structure

For small datasets its often useful to build your own dataframes. Dataframes are **objects** in R; in particular they are a type of **data structure**. “Data structure” is just a fancy way of saying “holding or organizing data.”

We'll work with the `SeaIce` data again.

```
library(Stat2Data)
data("SeaIce")
```

If we're curious about what kind of data is in an object we can use the `class()` command to see how is is classified.. Generally the first thing R spits out is the most important.

```
class(SeaIce)

## [1] "data.frame"

We see that SeaIce is a data.frame.
```

A related command is `is()`

```
is(SeaIce)

## [1] "data.frame"      "list"        "oldClass"     "vector"
```

This command is more **verbose** and tells of several things, the most relevant of which is that `SeaIce` is a dataframe.

The SeaIce dataframe has several columns. We can pull up their names using the `names()` command.

```
names(SeaIce)

## [1] "Year"   "Extent"  "Area"    "t"
```

There are times when we may want to examine just a single column. We can isolate a single column using a species notation which uses a dollar sign. To get the `Extent` column we do this

```
SeaIce$Extent

## [1] 7.22 7.86 7.25 7.45 7.54 7.11 6.93 7.55 7.51 7.53 7.08 6.27 6.59 7.59 6.54
## [16] 7.24 6.18 7.91 6.78 6.62 6.29 6.36 6.78 5.98 6.18 6.08 5.59 5.95 4.32 4.73
## [31] 5.39 4.93 4.63 3.63 5.35 5.29 4.68
```

14.2 Dataframes are made from vectors

Running this we get just a stream of numbers. In R, such a stream of numbers is called a **vector**. Like dataframes, vectors are a type of data structure. We can check this with `is()`

```
is(SeaIce$Extent)

## [1] "numeric" "vector"
```

The first result is `numeric`, which indicates that there are numbers in this vector; the second result is `vector`.

All columns in a dataframe are vectors:

```
is(SeaIce$Year)

## [1] "integer"          "double"           "numeric"
## [4] "vector"           "data.frameRowLabels"

is(SeaIce$Area)

## [1] "numeric" "vector"
```

14.3 Make your own vectors

We can make our own vectors using the `c()` command. (`c()` does MANY things in R, so we'll see it a lot).

Here is some more recent data on the Canda Lynx from 1984 to 2002 (Poole 2003, Table 3):

```
c(13445, 8625, 6853, 6953, 6574,
 8265, 9977, 7579, 11542, 7180,
```

```

4713, 4907, 2819, 5171, 6873,
6148, 8573, 9361, 11226)

## [1] 13445 8625 6853 6953 6574 8265 9977 7579 11542 7180 4713 4907
## [13] 2819 5171 6873 6148 8573 9361 11226

```

We can save this to an R object using a very species function in R called the **assigment operator**

```

lynx.ca <- c(13445, 8625, 6853, 6953, 6574,
            8265, 9977, 7579, 11542, 7180,
            4713, 4907, 2819, 5171, 6873,
            6148, 8573, 9361, 11226)

```

We've now made a brand new object in R called `lynx.ca`. We can see what it is by just typing its name in the console...

```

lynx.ca

## [1] 13445 8625 6853 6953 6574 8265 9977 7579 11542 7180 4713 4907
## [13] 2819 5171 6873 6148 8573 9361 11226

...and confirming what it is with is() and class()

is(lynx.ca)

## [1] "numeric" "vector"

class(lynx.ca)

## [1] "numeric"

```

The data are from 1984 to 2002. We can make a `year` vector like this

```

year <- c(1984, 1985, 1986, 1987,
        1988, 1989, 1990, 1991, 1992,
        1993, 1994, 1995, 1996, 1997,
        1998, 1999, 2000, 2001, 2002)

```

Its rather tedious to do that, so R has a trick. If we want a sequence of numbers we can use the `seq()` command, which has the arguments `from = ...` and `to = ...`

```

year <- seq(from = 1984, to = 2002)

```

14.4 Checking the length of vectors

Typing in data directly into R is errorprone and whenever we do it we should check our work carefully. A first check should be that we entered in all the

numbers; we can do this by getting R to tell us the length of the vector. Can you guess what the command is called?

```
length(lynx.ca)
```

```
## [1] 19
```

Let's check that our year vector is the same length

```
length(year)
```

```
## [1] 19
```

14.5 Another example

Another famous example

Howlett and Majerus. 1987. The understanding of industrial melanism in the peppered moth (*Biston betularia*) .(Lepidoptera: Geometridae). 30: 31-44.

Table 2

```
year.ne <- c(1954, 1961, 1970, 1975, 1981, 1982, 1983, 1984, 1985)
```

```
moths.ne <- c(92.95, 94.78, 75, 64.7, 45.9, 50, 42.9, 42.1, 39.6)
```

(Raw data is slightly more complicated, but this works)

```
length(year.ne)
```

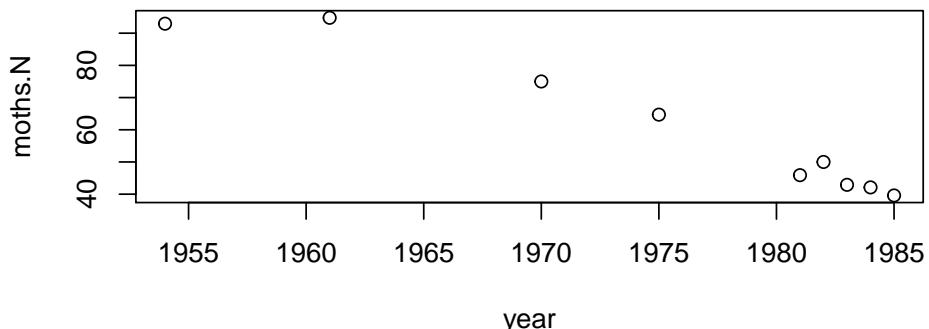
```
## [1] 9
```

```
length(moths.ne)
```

```
## [1] 9
```

```
moths <- data.frame(year = year.ne,
                      moths.N = moths.ne)
```

```
plot(moths.N ~ year, data = moths)
```



Not how R doesn't need extra space for the years where there are no data

14.6 Try it yourself

The original lynx data that comes with R spans the year 1821–1934. Make a sequence of numbers called year.lynx using the `seq()` command.

1957:1964

```
## [1] 1957 1958 1959 1960 1961 1962 1963 1964
```

There was a hypothesis that sunspot number and/or size impact lynx populations. R has a dataset called sunspots that runs from 1749 to 1983. Make a vector called year.sunspot using the `seq()` command.

1749:1983

```
##  [1] 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763
## [16] 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778
## [31] 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793
## [46] 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808
## [61] 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823
## [76] 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838
## [91] 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853
## [106] 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868
## [121] 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883
## [136] 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898
## [151] 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913
## [166] 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928
## [181] 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943
## [196] 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958
## [211] 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973
## [226] 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983
```

14.7 A complicate dataframe to make

14.8 Build the dataframe

This code makes all of the columns and makes a dataframe

```
aa      <-c('A','C','D','E','F','G','H','I','K','L','M','N','P','Q','R','S','T','V','W','Y')
MW.da    <-c(89,121,133,146,165,75,155,131,146,131,149,132,115,147,174,105,119,117,204,181)
volume   <-c(67,86,91,109,135,48,118,124,135,124,124,96,90,
           114,148,73,93,105,163,141)
bulkiness <-c(11.5,13.46,11.68,13.57,19.8,3.4,13.69,21.4,
            15.71,21.4,16.25,12.28,17.43,
            14.45,14.28,9.47,15.77,21.57,21.67,18.03)
```

```

polarity <-c(0,1.48,49.7,49.9,0.35,0,51.6,0.13,49.5,0.13,
           1.43,3.38,1.58,3.53,52,1.67,1.66,0.13,2.1,1.61)
isoelectric.pt <-c(6,5.07,2.77,3.22,5.48,5.97,7.59,6.02,9.74,5.98,
                     5.74,5.41,6.3,5.65,10.76,5.68,6.16,5.96,5.89,5.66)
hydrophobe.34 <-c(1.8,2.5,-3.5,-3.5,2.8,-0.4,-3.2,4.5,-3.9,3.8,1.9,
                     -3.5,-1.6,-3.5,-4.5,-0.8,-0.7,4.2,-0.9,-1.3)
hydrophobe.35 <-c(1.6,2,-9.2,-8.2,3.7,1,-3,3.1,-8.8,2.8,3.4,-4.8,
                     -0.2,-4.1,-12.3,0.6,1.2,2.6,1.9,-0.7)
saaH20 <-c(113,140,151,183,218,85,194,182,211,180,204,158,
            143,189,241,122,146,160,259,229)
faal.fold <-c(0.74,0.91,0.62,0.62,0.88,0.72,0.78,0.88,0.52,
                0.85,0.85,0.63,0.64,0.62,0.64,0.66,0.7,0.86,0.85,0.76)
polar.req <-c(7,4.8,13,12.5,5,7.9,8.4,4.9,10.1,4.9,5.3,10,
                 6.6,8.6,9.1,7.5,6.6,5.6,5.2,5.4)
freq <-c(7.8,1.1,5.19,6.72,4.39,6.77,2.03,6.95,6.32,
          10.15,2.28,4.37,4.26,3.45,5.23,6.46,5.12,7.01,1.09,3.3)
charge<-c('un','un','neg','neg','un','un','pos','un','pos',
          'un','un','un','un','pos','un','un','un','un')
hydropathy<-c('hydrophobic','hydrophobic','hydrophilic',
               'hydrophilic','hydrophobic','neutral','hydrophobic','hydrophilic',
               'hydrophilic','hydrophilic','neutral','neutral',
               'hydrophobic','hydrophobic','neutral')
volume.cat<-c('verysmall','small','small','medium',
               'verylarge','verysmall','medium','large','large',
               'large','large','small','small','medium','large','verysmall','small','medium')
polarity.cat<-c('nonpolar','nonpolar','polar','polar',
                  'nonpolar','nonpolar','polar','nonpolar',
                  'polar','nonpolar','nonpolar','polar','nonpolar','polar',
                  'polar','polar','polar','nonpolar','nonpolar','polar')
chemical<-c('aliphatic','sulfur','acidic','acidic','aromatic',
              'aliphatic','basic','aliphatic','basic','aliphatic','sulfur',
              'amide','aliphatic','amide','basic','hydroxyl','hydroxyl',
              'aliphatic','aromatic','aromatic')

aa_dat <- data.frame(aa,MW.da,volume,bulkiness,
                      polarity,isoelectric.pt,hydrophobe.34,hydrophobe.35,
                      saaH20,faal.fold, polar.req,freq,charge,hydropathy,volume.cat,polarity.cat,chemical)

```

14.9 References

Poole, Kim G. 2003. A review of the Canada Lynx, *Lynx canadensis*, in Canada. Canadian [Field-Naturalist 117: 360-376.] (<https://www.canadianfieldnaturalist.ca/index.php/cfn/article/view/738>) DOI: <https://doi.org/10.22621/cfn.v11i3.738>

Chapter 15

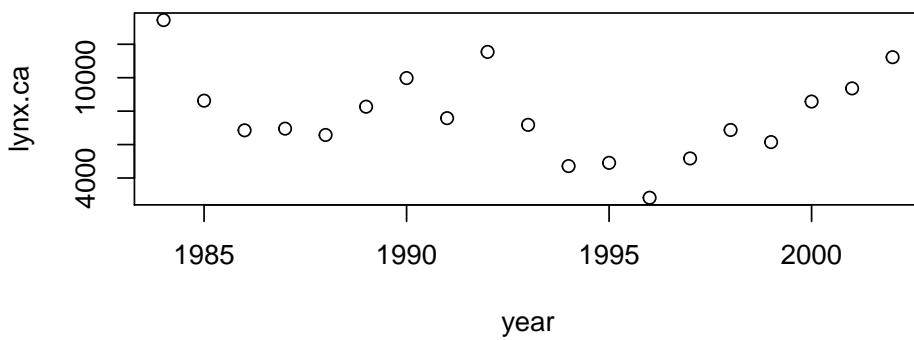
Build your own dataframe

```
lynx.ca <- c(13445, 8625, 6853, 6953, 6574,
 8265, 9977, 7579, 11542, 7180,
 4713, 4907, 2819, 5171, 6873,
 6148, 8573, 9361, 11226)

year <- c(1984, 1985, 1986, 1987,
 1988, 1989, 1990, 1991, 1992,
 1993, 1994, 1995, 1996, 1997,
 1998, 1999, 2000, 2001, 2002)

lynx.new <- data.frame(lynx.ca = lynx.ca,
                       year = year)

plot(lynx.ca ~ year, data = lynx.new)
```



As in the lynx dataset that comes with R, we see wild swings in abundance over short periods of time.

15.1 Your turn

Make this plot pretty

15.2 References

Poole, Kim G. 2003. A review of the Canada Lynx, *Lynx canadensis*, in Canada. Canadian Field-Naturalist 117: 360-376. [(<https://www.canadianfieldnaturalist.ca/index.php/cfn/article/view/738>)] DOI: <https://doi.org/10.22621/cfn.v11i3.738>

Chapter 16

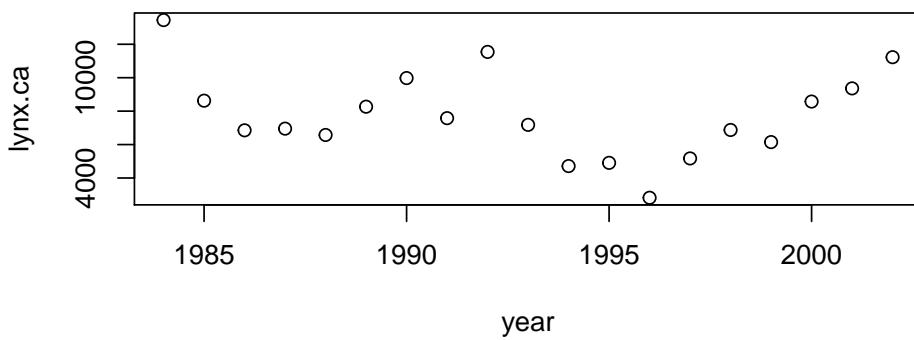
Build your own dataframe

```
lynx.ca <- c(13445, 8625, 6853, 6953, 6574,
 8265, 9977, 7579, 11542, 7180,
 4713, 4907, 2819, 5171, 6873,
 6148, 8573, 9361, 11226)

year <- c(1984, 1985, 1986, 1987,
 1988, 1989, 1990, 1991, 1992,
 1993, 1994, 1995, 1996, 1997,
 1998, 1999, 2000, 2001, 2002)

lynx.new <- data.frame(lynx.ca = lynx.ca,
                       year = year)

plot(lynx.ca ~ year, data = lynx.new)
```



As in the lynx dataset that comes with R, we see wild swings in abundance over short periods of time.

16.1 Your turn

Make this plot pretty

16.2 References

Poole, Kim G. 2003. A review of the Canada Lynx, *Lynx canadensis*, in Canada. Canadian Field-Naturalist 117: 360-376. [(<https://www.canadianfieldnaturalist.ca/index.php/cfn/article/view/738>)] DOI: <https://doi.org/10.22621/cfn.v11i3.738>

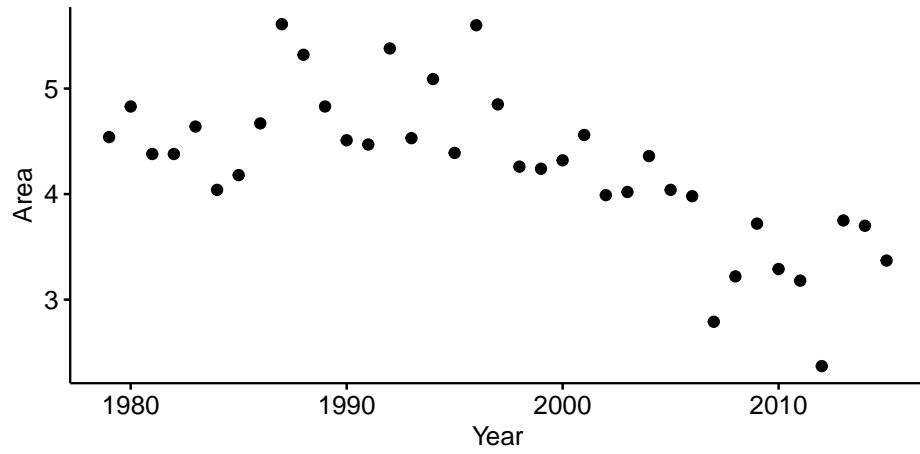
Chapter 17

Introduction to ggplot and ggpubr

```
##SeaIce
library(Stat2Data)
library(ggplot2)
library(ggpubr)

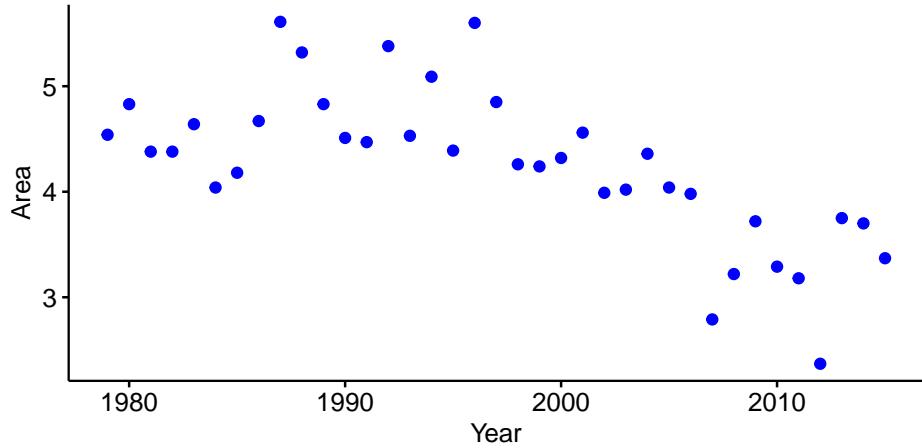
data(SeaIce)

ggscatter(y = "Area",
          x = "Year",
          data = SeaIce)
```

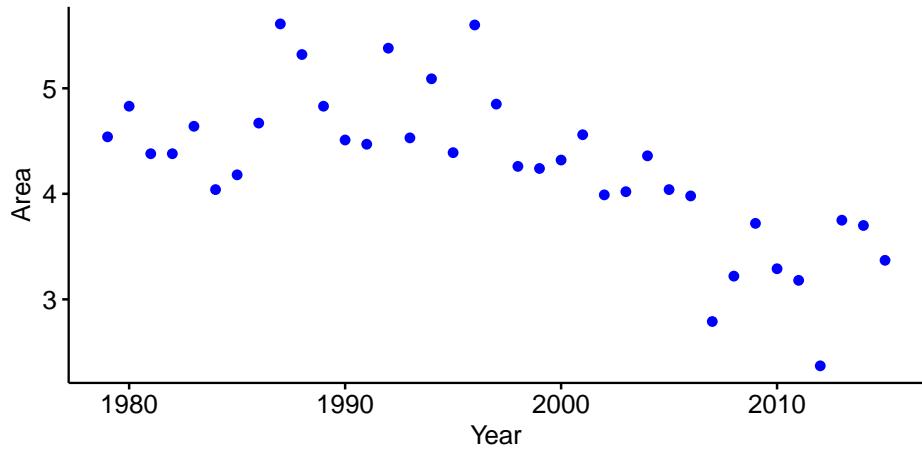


```
ggscatter(y = "Area",
          x = "Year",
```

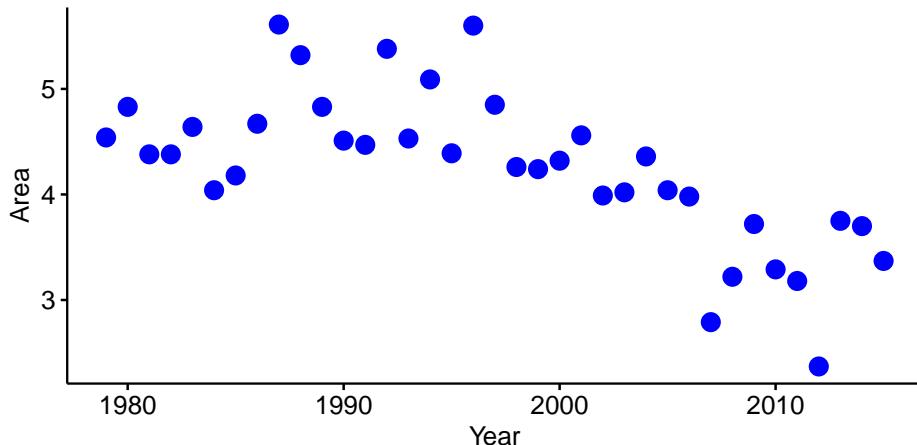
```
color = "blue", #in quotes  
data = SeaIce)
```



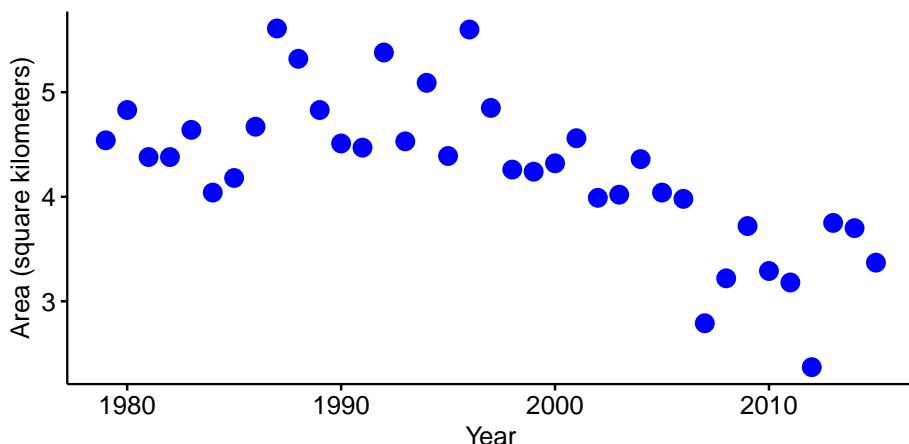
```
ggscatter(y = "Area",  
         x = "Year",  
         color = "blue",  
         shape = 16,      # not quotes  
         data = SeaIce)
```



```
ggscatter(y = "Area",  
         x = "Year",  
         color = "blue",  
         shape = 16,      #  
         size = 4,        # not quotes  
         data = SeaIce)
```



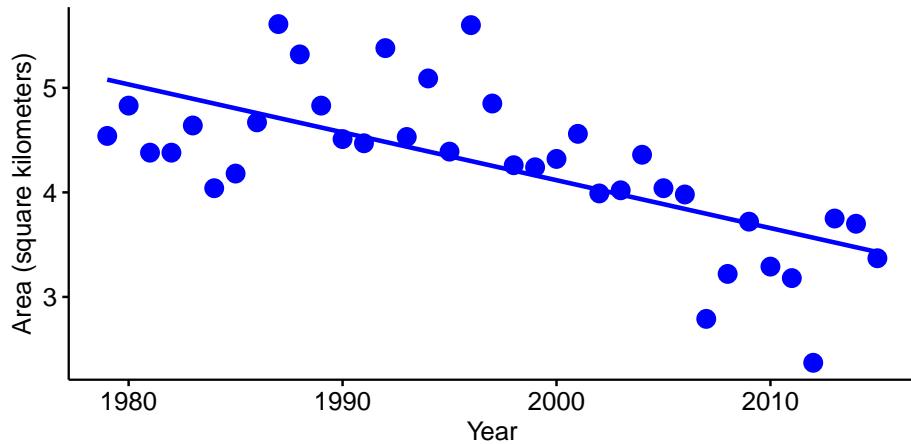
```
ggscatter(y = "Area",
          x = "Year",
          color = "blue",
          shape = 16,      #
          size = 4,       #
          ylab = "Area (square kilometers)", # in quotes
          data = SeaIce)
```



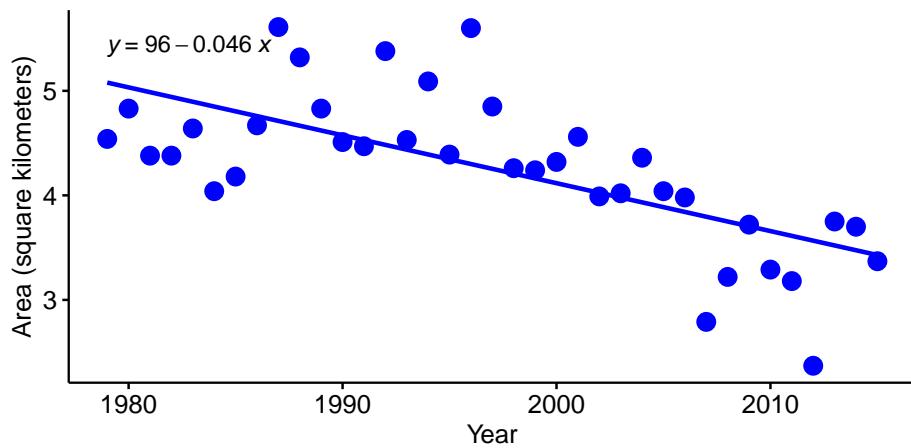
17.1 Regression line

```
ggscatter(y = "Area",
          x = "Year",
          color = "blue",
          shape = 16,      #
          size = 4,       #
```

```
ylab = "Area (square kilometers)",
add = "reg.line",
data = SeaIce)
```

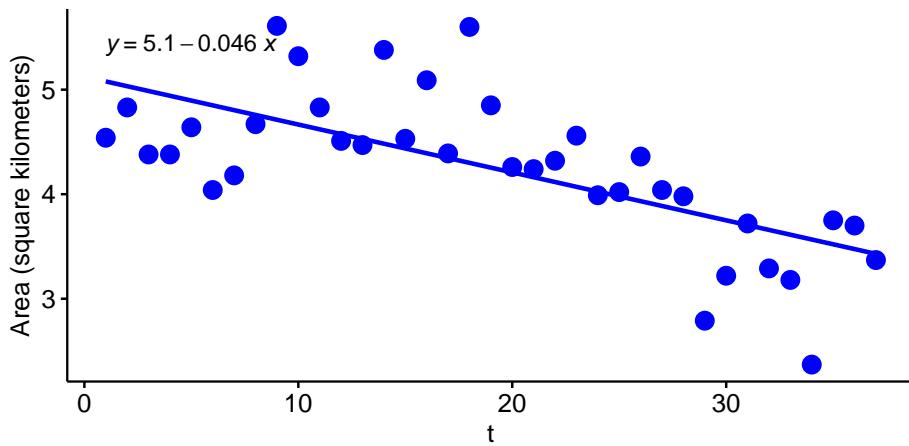


```
ggsscatter(y = "Area",
x = "Year",
color = "blue",
shape = 16,      #
size = 4,        #
ylab = "Area (square kilometers)",
add = "reg.line",
data = SeaIce) +
stat_regrline_equation()
```

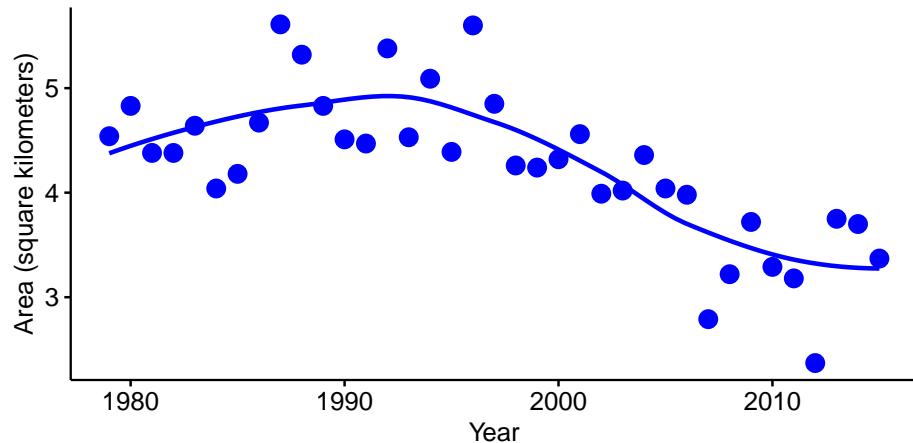


Why is equation different

```
ggscatter(y = "Area",
          x = "t",
          color = "blue",
          shape = 16,      #
          size = 4,       #
          ylab = "Area (square kilometers)",
          add = "reg.line",
          data = SeaIce) +
  stat_regress_line()
```

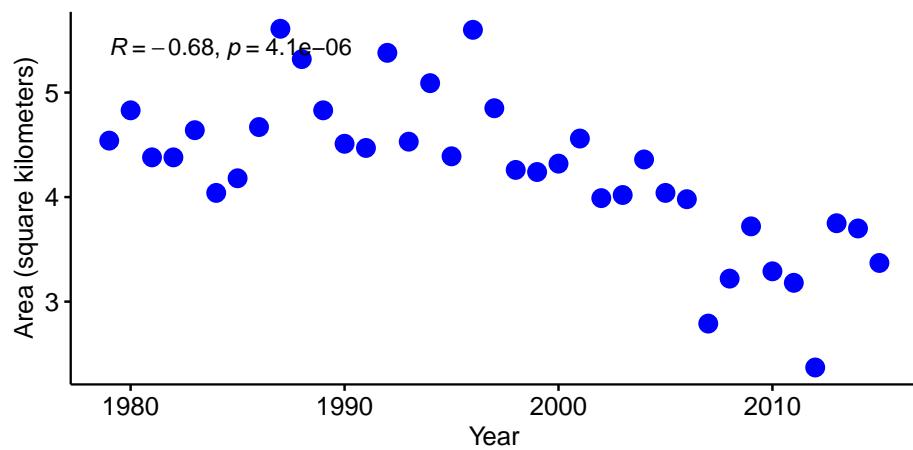


```
ggscatter(y = "Area",
          x = "Year",
          color = "blue",
          shape = 16,      #
          size = 4,       #
          ylab = "Area (square kilometers)",
          add = "loess",
          data = SeaIce)
```



17.2 Correlation

```
ggscatter(y = "Area",
          x = "Year",
          color = "blue",
          shape = 16,      #
          size = 4,        #
          ylab = "Area (square kilometers)",
          cor.coef = T,
          data = SeaIce)
```



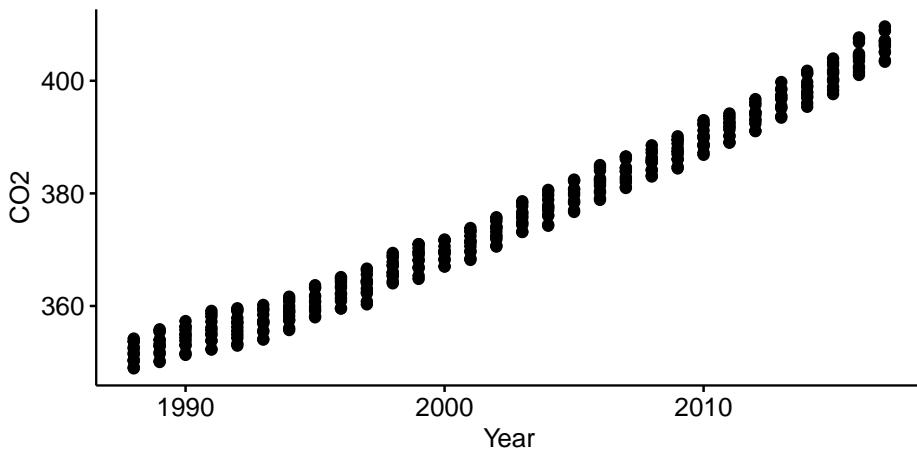
17.4 Long time series

```
data(CO2Hawaii)
```

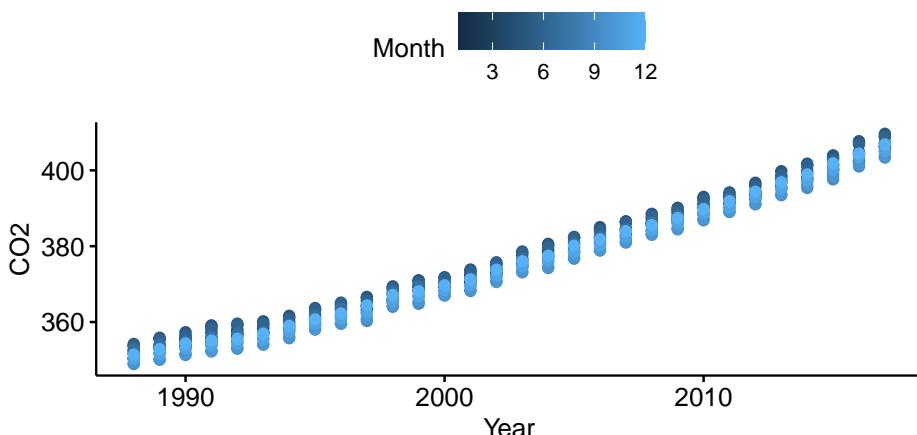
Year

Why stacked?

```
ggscatter(y = "CO2",
          x = "Year",
          data = CO2Hawaii)
```

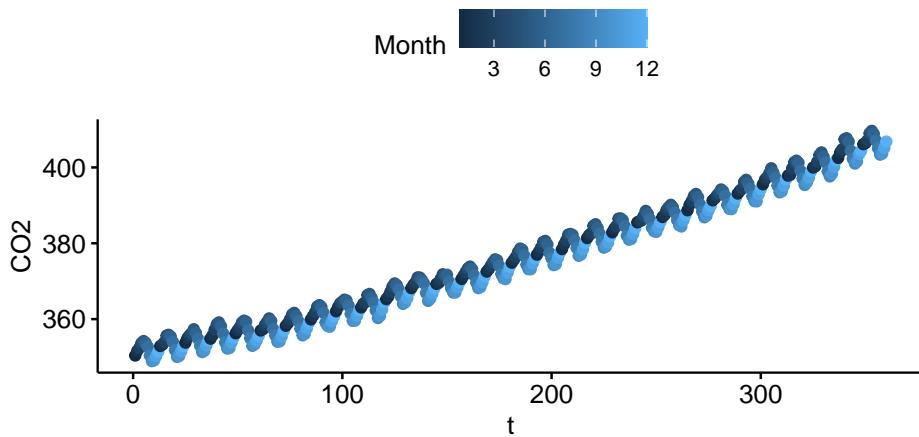


```
ggscatter(y = "CO2",
          x = "Year",
          color = "Month",
          data = CO2Hawaii)
```

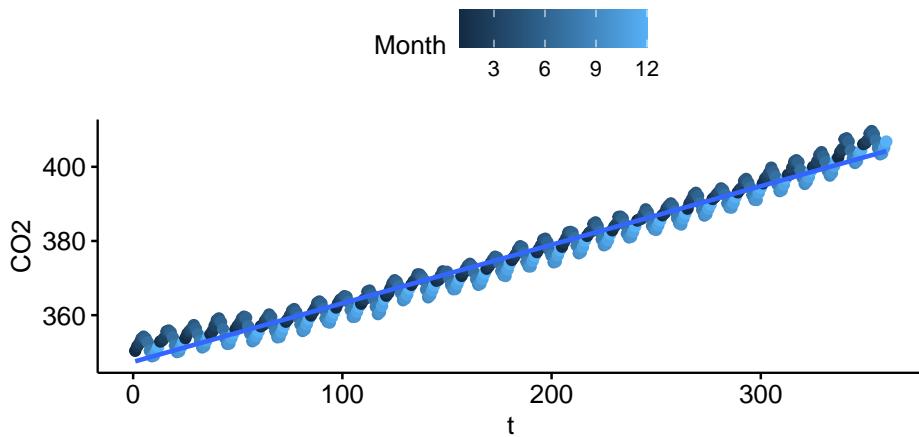


What's different?

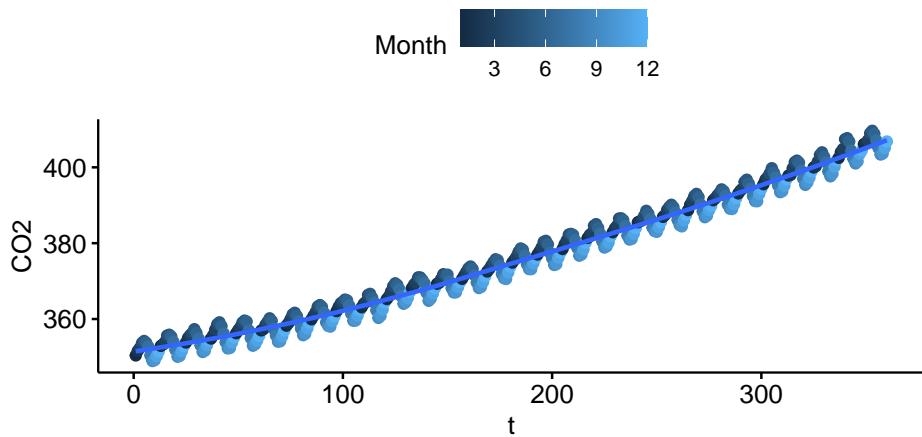
```
ggscatter(y = "CO2",
          x = "t",
          color = "Month",
          data = CO2Hawaii)
```



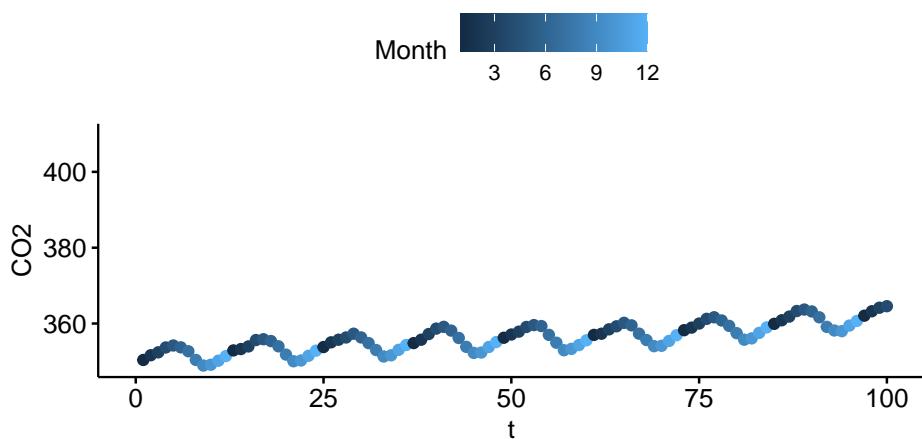
```
ggscatter(y = "CO2",
          x = "t",
          color = "Month",
          add = "reg.line",
          data = CO2Hawaii)
```



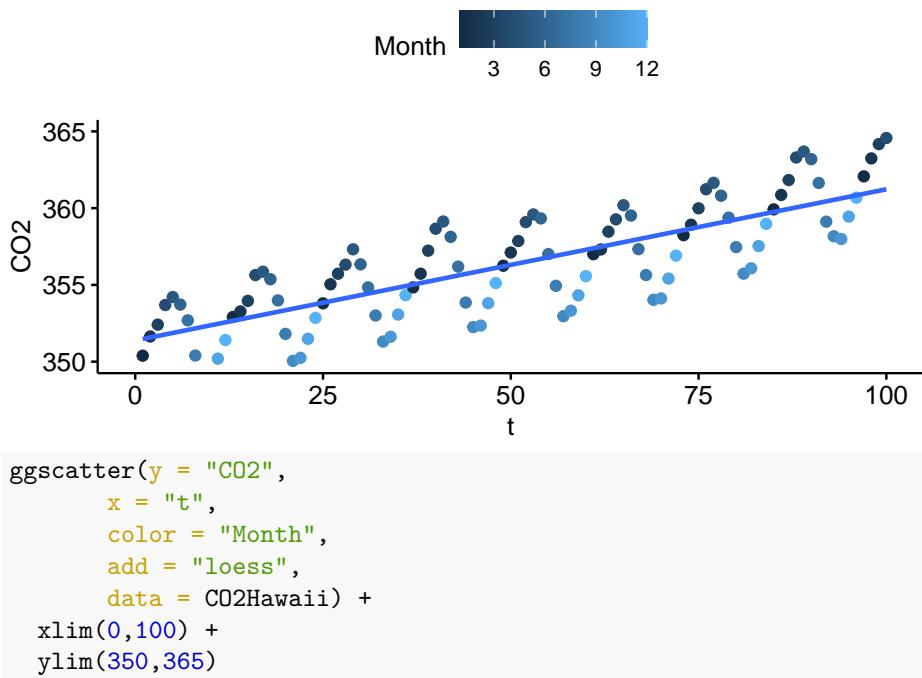
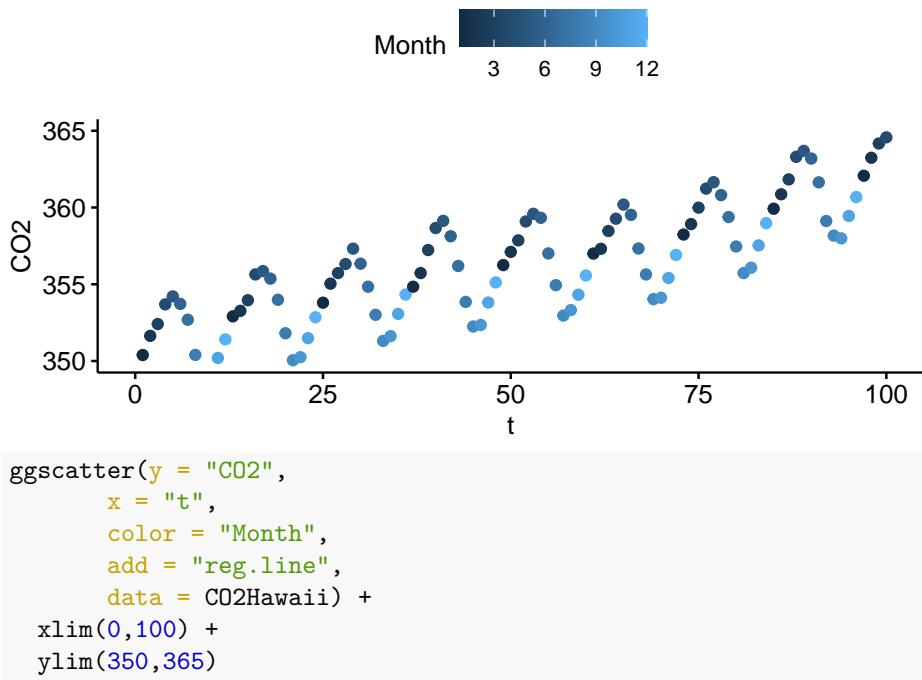
```
ggscatter(y = "CO2",
          x = "t",
          color = "Month",
          add = "loess",
          data = CO2Hawaii)
```

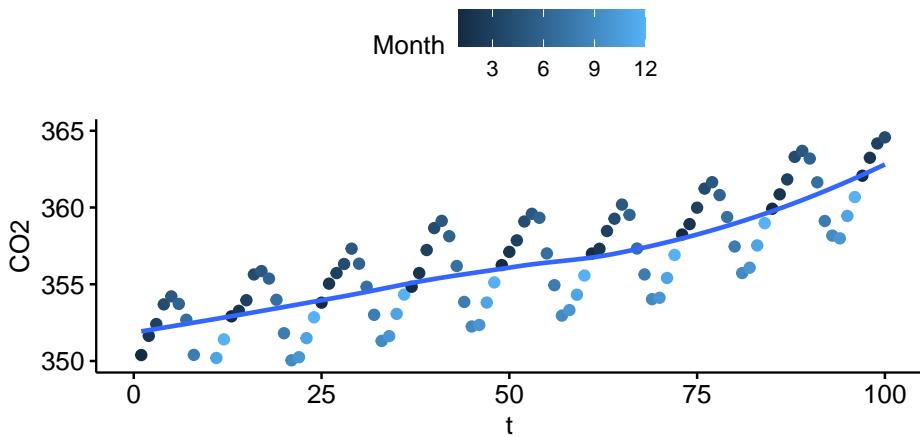


```
```r
ggscatter(y = "CO2",
 x = "t",
 color = "Month",
 data = CO2Hawaii) +
 xlim(0,100)
```



```
ggscatter(y = "CO2",
 x = "t",
 color = "Month",
 data = CO2Hawaii) +
 xlim(0,100) +
 ylim(350,365)
```





```
##Moth data - 2 cities
```

Cook and Grant 2000. Frequency of insularia during the decline in melanics in the peppered. Heredity 85: 580-585.

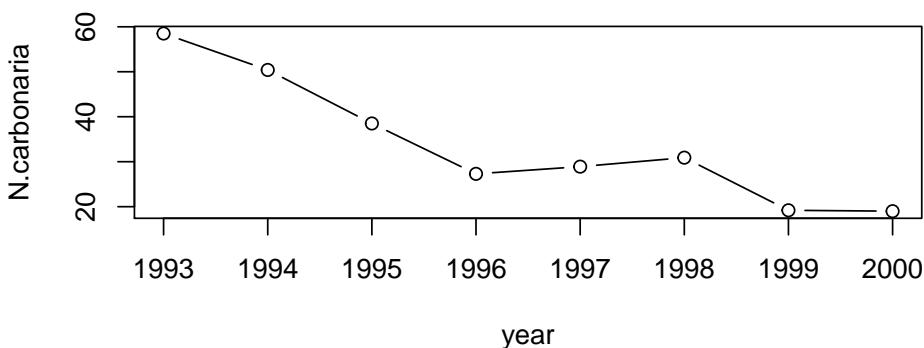
Make dataframe

```
##Nottingham data
```

```
year.nottingham <- seq(from = 1993,
 to = 2000)
carbonaria.nottingham <- c(58.5, 50.4, 38.5, 27.3, 28.9,
 30.9, 19.2, 19)

df.nottingham <- data.frame(year = year.nottingham,
 N.carbonaria = carbonaria.nottingham,
 city = "Nottingham")

plot(N.carbonaria ~ year, data = df.nottingham, type = "b")
```



How many times did R print “Nottingham” How did it make that choice?

```

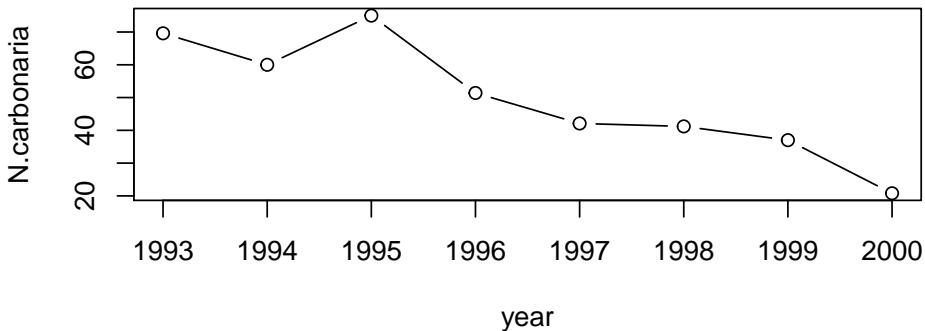
year.york <- seq(from = 1993,
 to = 2000)

carbonaria.york <- c(69.6, 60, 75.0, 51.4, 42.1,
 41.2, 37.0, 20.8)

df.york <- data.frame(year = year.york,
 N.carbonaria = carbonaria.york,
 city = "York")

plot(N.carbonaria ~ year, data = df.york, type = "b")

```



```

df.moth <- rbind(df.nottingham,
 df.york)

is(df.moth)

```

```

[1] "data.frame" "list" "oldClass" "vector"
dim(df.moth)

```

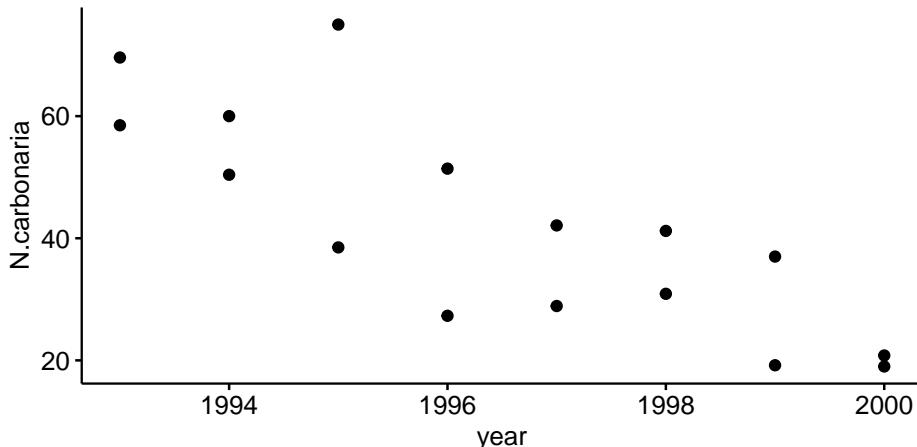
```

[1] 16 3
install.packages("ggplot2")

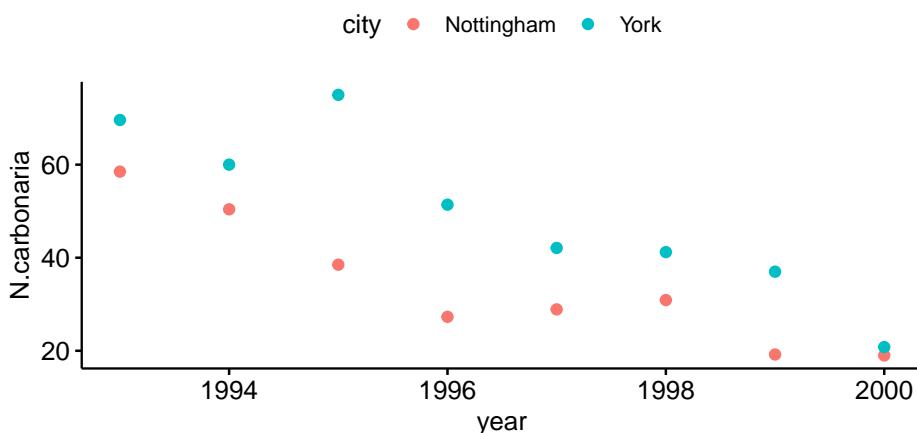
library(ggplot2)
library(ggpubr)

ggscatter(y = "N.carbonaria", # in quotes
 x = "year", # in quotes
 data = df.moth) # in quotes

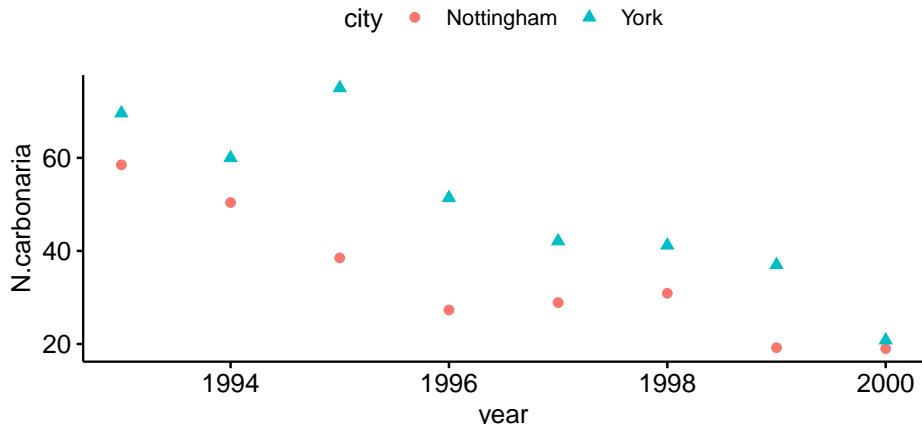
```



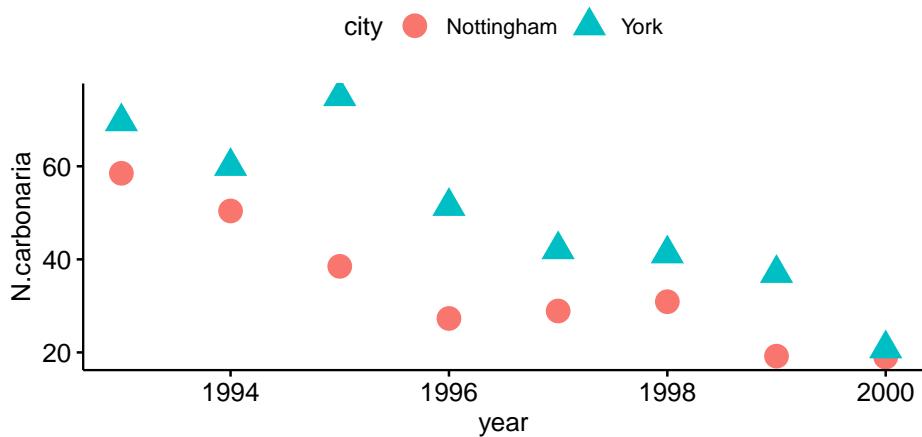
```
ggscatter(y = "N.carbonaria",
 x = "year",
 color = "city", # in quotes
 data = df.moth)
```



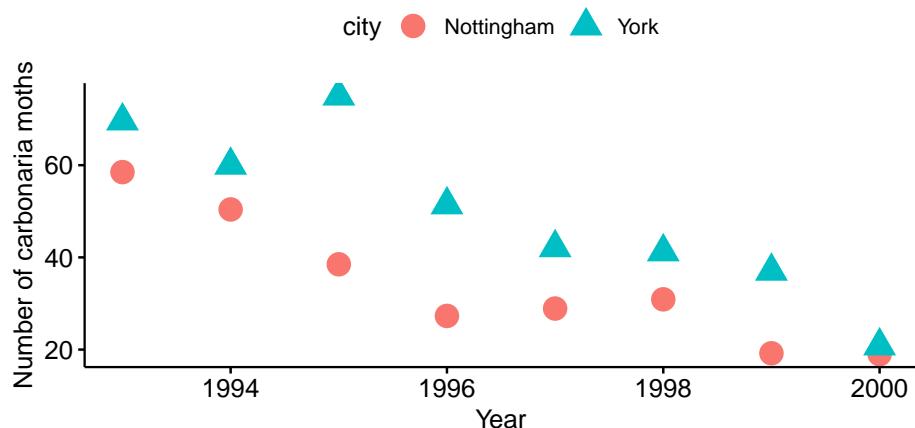
```
ggscatter(y = "N.carbonaria",
 x = "year",
 color = "city",
 shape = "city", # in quotes
 data = df.moth)
```



```
ggscatter(y = "N.carbonaria",
 x = "year",
 color = "city",
 shape = "city",
 size = 5, # not in quotes!
 data = df.moth)
```

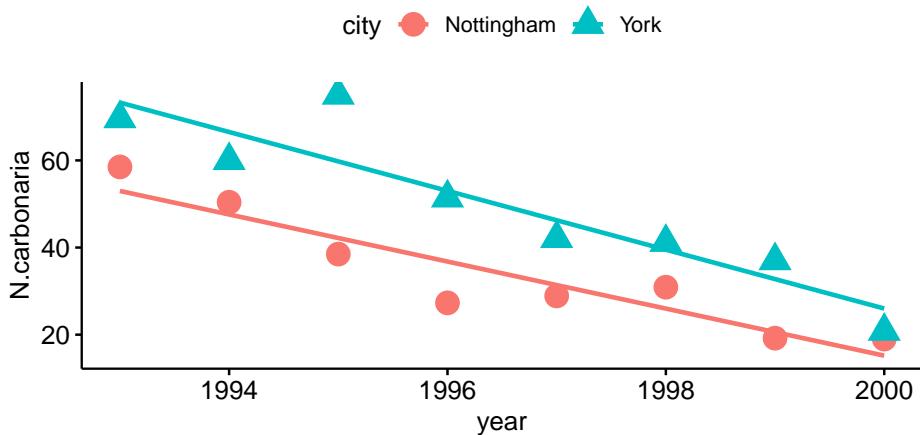


```
ggscatter(y = "N.carbonaria",
 x = "year",
 color = "city",
 shape = "city",
 size = 5,
 data = df.moth,
 ylab = "Number of carbonaria moths", # labels in quotes
 xlab = "Year") # in quotes
```

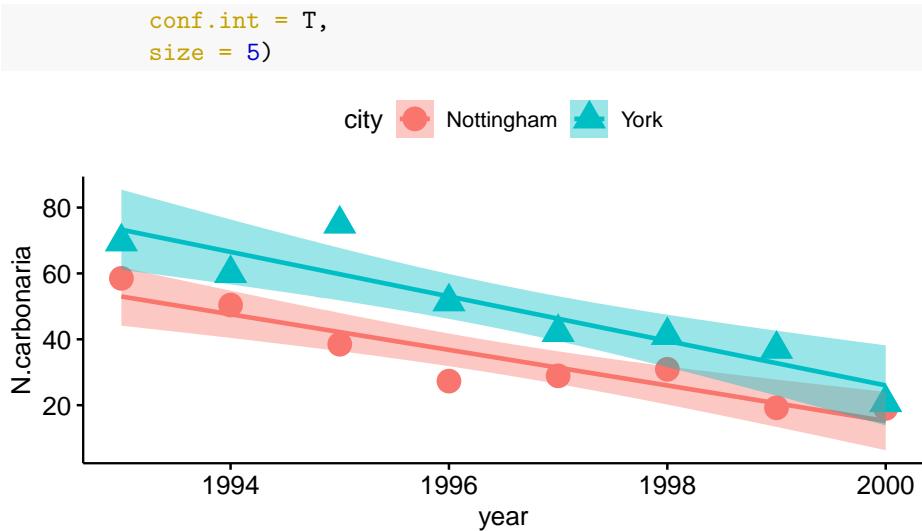


## 17.5 Regression lines

```
ggscatter(y = "N.carbonaria",
 x = "year",
 data = df.moth,
 color = "city",
 shape = "city",
 add = "reg.line",
 size = 5)
```

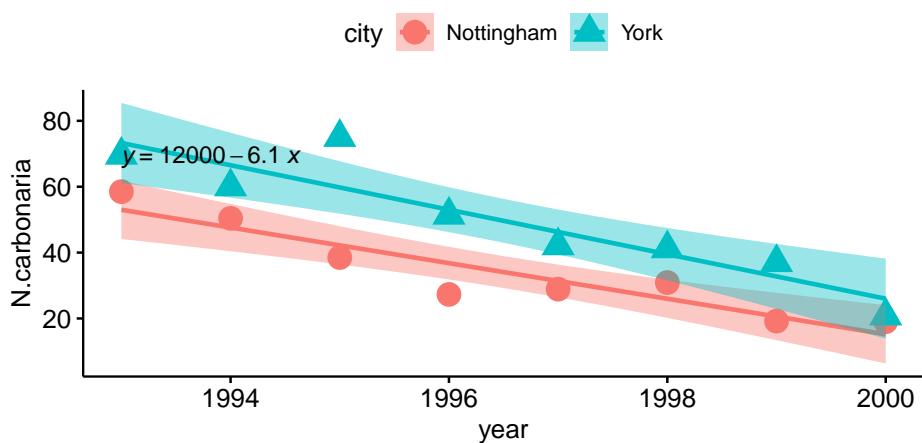


```
ggscatter(y = "N.carbonaria",
 x = "year",
 data = df.moth,
 color = "city",
 shape = "city",
 add = "reg.line",
```



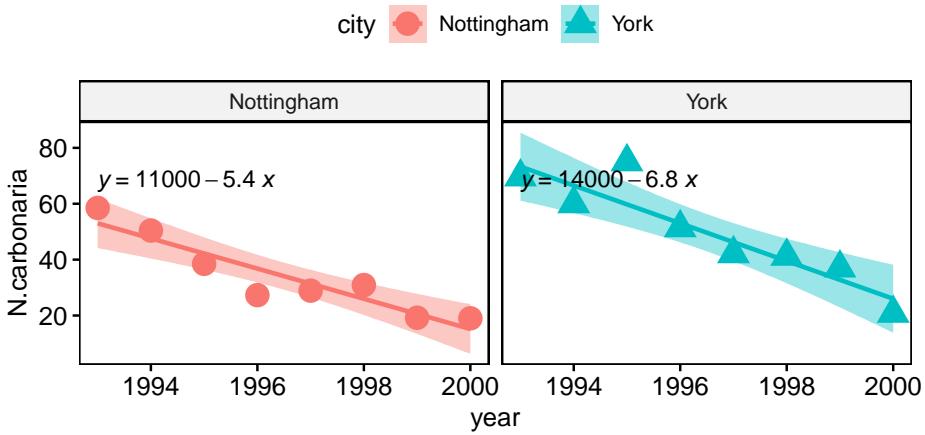
Only 1 line

```
ggscatter(y = "N.carbonaria",
 x = "year",
 data = df.moth,
 color = "city",
 shape = "city",
 add = "reg.line",
 conf.int = T,
 size = 5) +
stat_regrline_equation()
```



Facetings - regressions for both lines

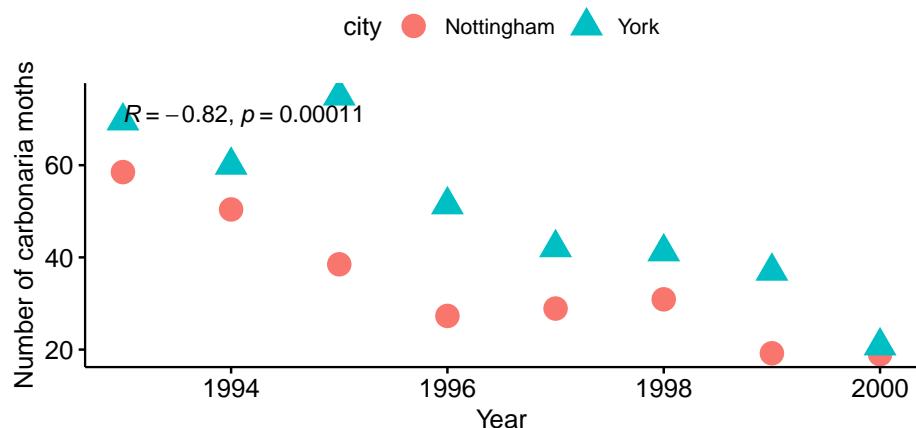
```
ggscatter(y = "N.carbonaria",
 x = "year",
 data = df.moth,
 color = "city",
 shape = "city",
 add = "reg.line",
 conf.int = T,
 facet.by = "city",
 size = 5) +
stat_regrline_equation()
```



## 17.6 Correlations

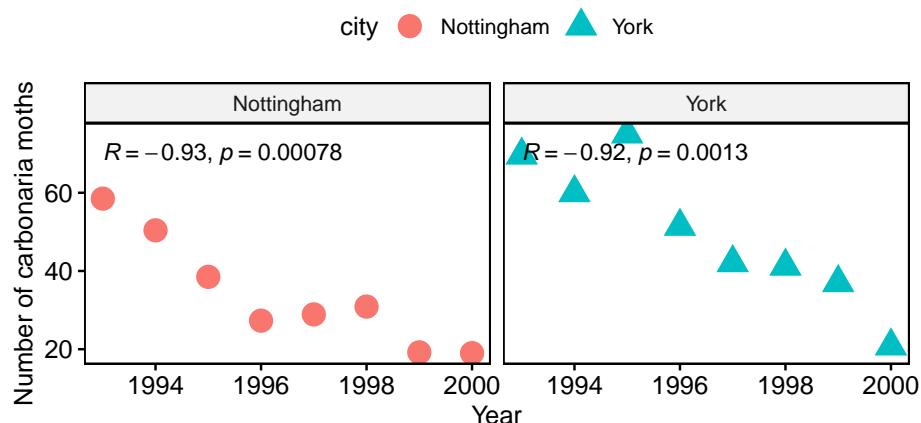
Only 1 correlations coefficient

```
ggscatter(y = "N.carbonaria",
 x = "year",
 color = "city",
 shape = "city",
 size = 5,
 data = df.moth,
 cor.coef = T,
 ylab = "Number of carbonaria moths", # labels in quotes
 xlab = "Year") # in quotes
```

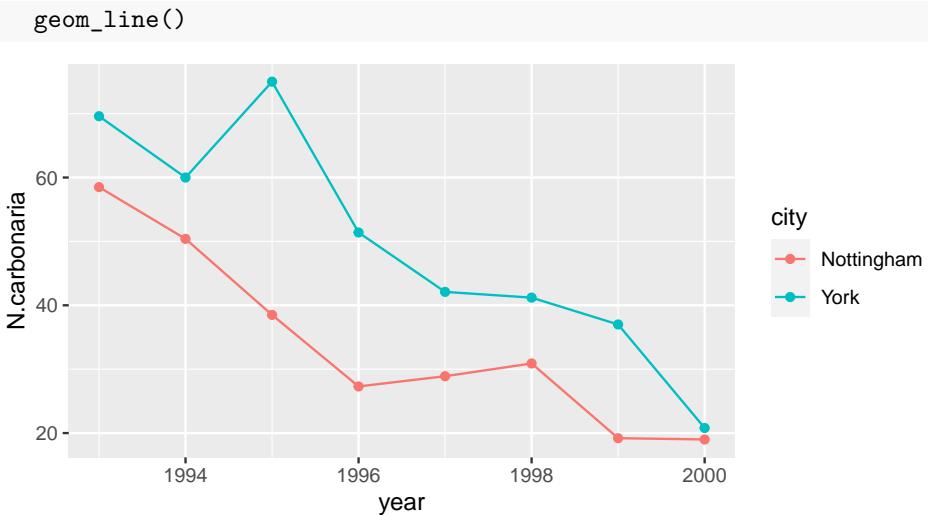


Faceting gives us 2

```
ggscatter(y = "N.carbonaria",
 x = "year",
 color = "city",
 shape = "city",
 size = 5,
 data = df.moth,
 cor.coef = T,
 facet.by = "city",
 ylab = "Number of carbonaria moths", # labels in quotes
 xlab = "Year") # in quotes
```



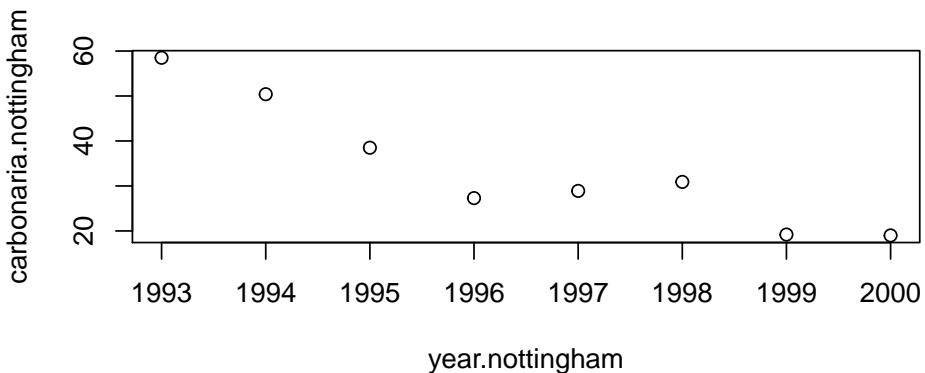
```
ggplot(data = df.moth,
 aes(y = N.carbonaria,
 x = year,
 color = city)) +
 geom_point() +
```



## 17.7 Original data

```
Cook and Grant 2000. Frequency of insularia during the decline
in melanics in the peppered. Heredity 85: 580-585.
year.nottingham <- seq(from = 1993,
 to = 2000)
carbonaria.nottingham <- c(58.5, 50.4, 38.5, 27.3, 28.9,
 30.9, 19.2, 19)

plot(carbonaria.nottingham ~ year.nottingham)
```



```
year.york <- seq(from = 1990,
 to = 2000)

carbonaria.york <- c(57.9, 61.8, 74.1, 69.6, 60, 75.0,
```

51.4, 42.1, 41.2, 37.0, 20.8)

```
plot(carbonaria.york ~ year.york)
```

A scatter plot showing the relationship between the year and the value of carbonaria.york. The x-axis is labeled "year.york" and ranges from 1990 to 2000. The y-axis is labeled "carbonaria.york" and ranges from 20 to 60. There are 11 data points plotted as open circles. The values show a general downward trend over time.

year.york	carbonaria.york
1990	58
1991	62
1992	68
1993	65
1994	58
1995	68
1996	50
1997	42
1998	40
1999	38
2000	20

```
year.manchester <- c("1952-1964", "1968-1974", "1990-1998")
carbonaria.manchester <- c(98.5, 95.9, 20.8)
```

```
year.kent <- c(1971, 1975, 1979, 1983, 1987,
1990, 1991, 1992, 1993, 1994)
```

```
carbonaria.kent <- c(78.2, 76.6, 71.7, 64.7, 54.3,
42.3, 30.9, 33.5, 23.0, 23.8)
```

```
plot(carbonaria.kent ~ year.kent)
```

A scatter plot showing the relationship between the year and the value of carbonaria.kent. The x-axis is labeled "year.kent" and ranges from 1975 to 1990. The y-axis is labeled "carbonaria.kent" and ranges from 30 to 70. There are 10 data points plotted as open circles. The values show a general downward trend over time.

year.kent	carbonaria.kent
1971	78.2
1975	76.6
1979	71.7
1983	64.7
1987	54.3
1990	42.3
1991	30.9
1992	33.5
1993	23.0
1994	23.8

```
year.hampshire <- c("1957-1964", "1965-1970", "1973-1977")
```

```
carbonaria.hampshire <- c(7.1, 5.8, 6.5)
```

Long time series  
insularia, not carbonaria!

Clarke, Mani and Wynne. 1985. Evolution in reverse: clean air and the peppered moth. Biological Journal of the Linnean Society 26: 189-199.



# Chapter 18

## Loading packages & data from GitHub

TODO= change wildlifeR to compbio4all?

### 18.1 Introduction

**GitHub** is an online platform for hosting and sharing code. More formally it is called a **software repository**. It is very popular with software developers, especially those creating open-source applications, and has also been adopted whole-hearted by many data scientists and data analysts.

GitHub has many features and uses. One of the most basic ones is to use GitHub like Dropbox to R backup copies of code on GitHub. GitHub also can act like a kind of web server to host websites, online books like this one, and provide access to open source software. Many people working on R packages use GitHub to host their package while its being developed or expanded. When a package is finished, it often is then submitted to CRAN, and the version on GitHub is used as the **development version** where new features are being developed and tested.

You can access packages on GitHub to get the newest version before something has been submitted to CRAN, or packages that haven't or maybe will never end up on CRAN. This book relies on a package I've written called **wildlifeR** for datasets and some functions. In this short exercise we'll download a package from CRAN we need to interact with GitHub, and then download **wildlifeR**. We'll also go to the **wildlifeR** website to learn more about the package.

### 18.1.1 Learning objectives

### 18.1.2 Learning goals

### 18.1.3 Functions & Arguements

- library
- devtools::install\_github
- scatter.smooth
- \$

### 18.1.4 Packages

- devtools
- wildlifeR

### 18.1.5 Potential hangups

- We'll use the “\$” operator to tell scatter.smooth() what to plot, which is different than how ggpibr and ggplot2 work; sigh...

## 18.2 [ ] Accessing GitHub using devtools

The devtools package is used by many people who write R packages and includes a function for downloading from GitHub

```
install.packages("devtools", dependencies = TRUE) # []
```

devtools has a lot of dependencies so this might take a while.

Once everything is downloaded, load the package explicitly with library()

```
library(devtools)
```

## 18.3 [ ] Downloading the wildlifeR package with install\_github()

My github site is at <https://github.com/brouwern> and the code for wildlifeR is <https://github.com/brouwern/wildlifeR>. You can access the files directly if you want, but that isn't necessary. We can download the package just like it was on CRAN using install\_github(). You'll probably see some red text and a LOT of black text as install\_github() talks with GitHub.

```
install_github("brouwern/wildlifeR")
```

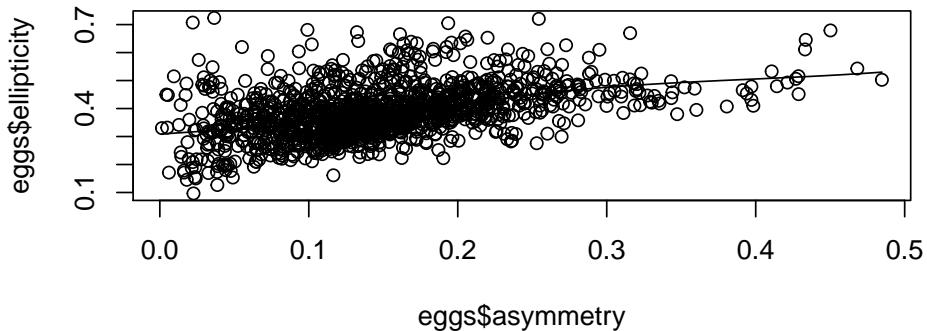
Now we can put it all explicitly into memory

```
library(wildlifeR) # []
```

---

**OPTIONAL:** Accessing data from `wildlifeR` One of the datasets in `wildlifeR` is called “eggs.” It has data from a paper by Stoddard et al. (2017) in Science called [Avian egg shape: Form, function, and evolution.] (<http://science.sciencemag.org/content/356/6344/1249>). We can plot the relationship between egg asymmetry and ellipticity using the base R function `scatter.smooth()`, which draws a type of regression line through the data for us (Note that the syntax for `scatter.smooth()` is, sadly, different than `plot()` and other plotting functions...).

```
scatter.smooth(eggs$asymmetry, eggs$ellipticity)
```



---

## 18.4 [ ] The `wildlifeR` package website

Some packages have websites that summarize the package contents. If you visit <https://brouwern.github.io/wildlifeR/> you can find out information on each dataset and function under the “Reference” tab, and see how the datasets and functions are used under the “Articles” tab.



## Chapter 19

# Installing from github

```
install.packages("devtools")
library(devtools)
```

Dependencies



# Chapter 20

## Packages & their dependencies

TODO: change from crabs to something else

R packages frequently use other R packages (which frequently use other R packages...). When an R package requires another package, its called a **dependency**. Depending on who and how the package was written up, dependencies might not be an issue or could be a problem.

As noted above when you download packages using RStudio's point and click interface there's a box that should be checked called "Install dependencies."

If you are using `install.packages()` there is an extra argument "dependencies = TRUE" that elicits the same behavior. I'll use this to get an add-on for ggplot2 called `ggbubr`.

```
install.packages("ggbubr", dependencies = TRUE)
```

We can then install this

```
library(ggbubr)
library(MASS)
```

---

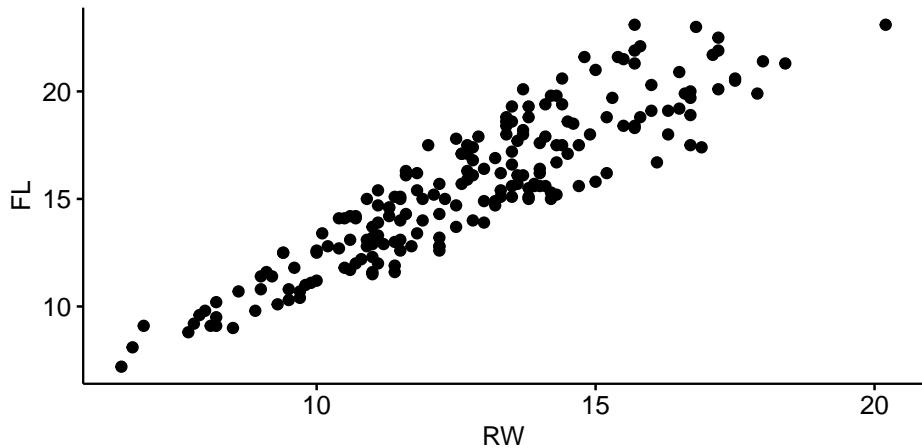
### 20.1 Optional: Make a plot with ggbubr

This section is optional

`ggbubr` is an add on to `ggplot`. (This means that `ggbubr` has `ggplot` as a dependency). Note that the syntax for `ggbubr` function we use, `ggscatter()`, has

a different syntax (again) than ggplot's qplot() function and base R's plot() function.

```
ggscatter(data = crabs, y = "FL", x = "RW") # use quotes!
```



End optional section

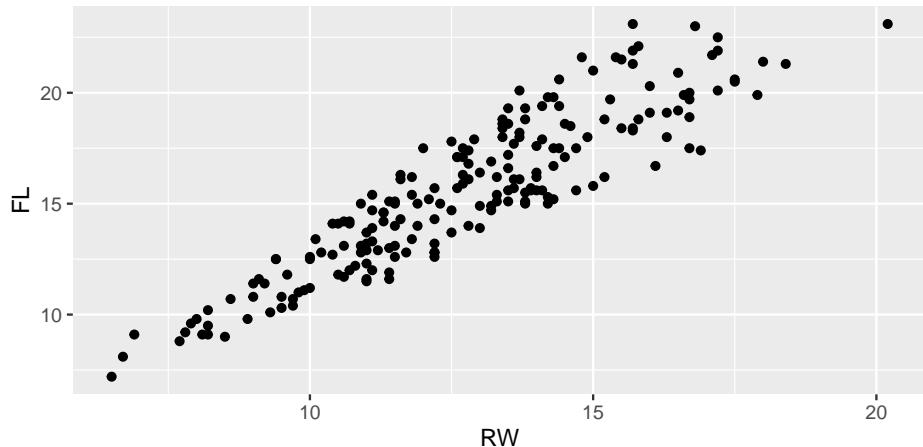
---

## 20.2 Challenge

An another add-on to ggplot2 is cowplot, which stands for “Claus O. Wilke Plot”. Download cowplot from CRAN using either the point-and-click method or `install.packages`, and then load it using `library`. Then run the following R code, which should make the plot below.

Note that “FL” and “RW” are NOT in quotation marks as they are for ggscatter()!

```
qplot(data = crabs, y = FL, x = RW) #no quotes!
```





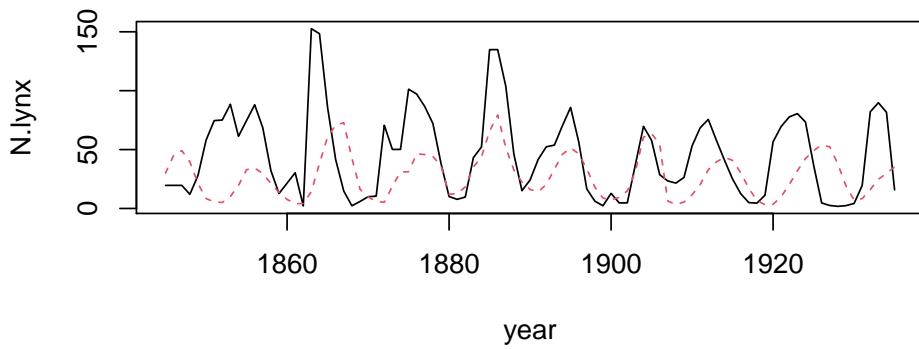
# Chapter 21

## Load data from internet

```
my.url <- url("http://people.whitman.edu/~hundledr/courses/M250F03/LynxHare.txt")
lynx_vs_hares <- read.table(my.url,
 header = FALSE,
 col.names = c("year", "N.lynx", "N.hare"))
```

### 21.1 Base R plot

```
plot(N.lynx ~ year, data = lynx_vs_hares, type = "l")
lines(N.hare ~ year, data = lynx_vs_hares, type = "l", col = 2, lty = 2)
```



### 21.2 ggpublisher - have to stack it

write function - stack data

```

stack_data <- function(df,
 x.col.name,
 y1.col.name,
 y2.col.name,
 ystack.col.name = "stacked",
 y1.label = "group1",
 y2.label = "group2"){
 n.row <- dim(df)[1]
 x <- df[,x.col.name]
 y1 <- df[,y1.col.name]
 y2 <- df[,y2.col.name]
 y1.lab <- rep(y1.label, n.row)
 y2.lab <- rep(y2.label, n.row)

 df.new <- data.frame(x = rep(x,2),
 y = c(y1,y2),
 group.num = c(rep(1,n.row),
 rep(2,n.row)),
 group.name = c(y1.lab, y2.lab))
 names(df.new)[1:2] <- c(x.col.name,
 ystack.col.name)

 return(df.new)
}

names(lynx_vs_hares)

[1] "year" "N.lynx" "N.hare"
out <- stack_data(df = lynx_vs_hares,
 x.col.name = "year",
 y1.col.name = "N.lynx",
 y2.col.name = "N.hare",
 ystack.col.name = "N",
 y1.label = "lynx",
 y2.label = "hare")

```

# Chapter 22

## Assignment: Building dataframes by hand

```
library(compbio4all)
```

### 22.1 Introduction

This exercise is meant to challenge you to build a dataframe by hand in R. It is based off of the Table 1 in Drake (1991) “A constant rate of spontaneous mutation in DNA-base microbes.”

There are two versions of this tutorial. This versions will guide you through the process of building the dataframe and exploring it. A second version is designed to challenge your understanding of the steps we went through.

### 22.2 To do

a student noted on an earlier vs. of the assignment; feel free to take this approach when possible. I may modify the assignment in the future to reflect this approach

“I was finding the exponent notation” used in the assignment ”tedious, so I tried what I’ve always done in other languages and it works. Other programmers might be more comfortable with this so it might be worth mentioning if we do more exercises with exponents:

Using “E” (or “e”) immediately after a number followed by the power work, and you only need to include sign if the notation is for a decimal (e.g. 1,000,000 can be represented by “1E6” and 0.01 by “1E-2”).

In the data frames exercise, the vector with numbers from the paper could've been done in one step like this: `vector <- c(6.41E3, 4.85E4, 1.6E5...)`. I'm not sure if this would make more sense or be more efficient for others a well, but I personally prefer this style of notation."

## 22.3 Preliminaries

### 22.3.1 Non-R materials

For this you'll need a copy of Drake (1991), which can be found at <https://www.pnas.org/content/88/16/7160>

We will focus on rebuiling Table 1 and Figure 1 entirely in R without importing the data from a spreadsheet.

### 22.3.2 Packages

The only package you will need is `ggpubr`. Load `ggpubr` using `library()`:

### 22.3.3 Ignore this

deprecated

## 22.4 The data

Table 1 of Drake (1991) has the following columns. Note that the symbol mu ( $\mu$ ) is frequently used to represent mutation rates.

1. A list of organisms (viruses, bacteria and invertebrate eukaryotes)
2. genome sizes (G) for each organism, in scientific notation
3. A target gene, which was assessed for the occurrence of mutations
4. An estimate of the mutation rate per bases (bp) of the genome. This gets log transformed and used as the y variable in figure 1. I'll use the symbol  $u.\text{bp}$  for this variable
5. An estimate of the mutation rate per genome ( $u.g$ ). This variable isn't in scientific notation so we'll work with it first because its a bit easier to read.

The organism column has gaps in it because there are multiple target genes for these species

1. 3 E. coli genes
2. 3 S. cerevisiae genes
3. 2 N. crassa genes

**Note that the very last numbers that appear in the mutation columns are means, not data!**

## 22.5 Creating vectors in R

We can build up a dataframe in R by making a **vector** for each column. This is a bit tedious but will build up our skills, so be patient.

We can make a vector of organism names like this. We need to make sure each word representing an organisms is in quotes, and that there is a comma after each word.

```
org <- c("M13", "lamb", "T2", "T4", "EC", "EC", "EC", "SC", "SC", "SC", "NC", "NC")
```

We can use line breaks while defining an R object, which can make it easier to keep track of what were typing. The following code is equivalent to what we just ran. Because there are multiple E. coli (EC) , SC and NC, I'm putting them on their own lines.

```
org <- c("M13", "lamb", "T2", "T4",
 "EC", "EC", "EC",
 "SC", "SC", "SC",
 "NC", "NC")
```

If you want to be fancy, use rep() for the repeated (I will rarely do this in class though).

```
org <- c("M13", "lamb", "T2", "T4",
 rep("EC", 3),
 rep("SC", 3),
 rep("NC", 2))
```

As always we need to check to make sure that the object we're making is what we think it is and represents what we want. Use the following commands to check on the object:

- is
- is.vector
- is.matrix
- class
- length
- nchar

Next let's make the next vector. Below is the code you'll need, but I've left off the last 3 numbers. Paste the code into the code chunk and add the necessary numbers from the original table. Don't forget the commas. Also note that numbers **don't** get quotes around them.

```
G.p <- c(6.41, 4.85, 1.60, 1.66, 4.70, 4.70, 4.70, 1.38, 1.38)
```

```
Run the code to make the G.p vector here
be sure to add the necessary numbers
```

Again, we explore this. Run these functions and note the output

- is
- class
- length
- nchar

```
Type code to explore the G.p vector here
```

## 22.6 Checking the length of vectors

Vectors in R have length, not dimension, so to see how big they are we use the length() command; dim() won't work. Neither will ncol() or nrow().

All of our columns have the same number of elements in them. We can make sure we don't have any errors in our data entry by using a logical operation, ==, to confirm that the length of each vector is the same. This is very useful when vectors are big and/or when we're writing code to automate a process

First, check the length of one of the vectors

```
Confirm the length of of one of the vectors
```

Next, run this code to check that the two lengths are the same: length(G.p) == length(org)

```
Check that the lengths are the same
```

## 22.7 Exponents and scientific notation

To take the exponent of something in R use the up caret ^

So, if I want to type 1 million I can do this

```
1000000
```

```
[1] 1e+06
```

or I can do this

```
1*10^6
```

```
[1] 1e+06
```

Note that the output is in “e” notation. I could type this if I wanted

```
1e+06
```

```
[1] 1e+06
```

In the chunk below, write a logical expression using == to confirm that 1000000 is equal to 1\*10^6

To represent the genome size column I could do this (just showing first 3 numbers)

```
G.p <- c(6.41*10^5, 4.85*10^4, 1.60*10^5)
```

However, when typing up this data I found it easier to break up the columns containing exponents into multiple columns, otherwise I was getting cross eyed.

One way to do this would be to split each number into what I'm calling a "prefix" and a "suffix". Above we made a vector called G.p, which is the prefix of these numbers - the part before the multiplication symbol.

We can make another vector that contains the suffix - the  $10^x$  part.

```
G.s <- c(10^3, 10^4, 10^5, 10^5, 10^6, 10^6, 10^7, 10^7, 10^7, 10^7)
```

We can then multiple these together and get the value we want

```
G.p*G.s
```

```
[1] 6.41e+08 4.85e+08 1.60e+10 6.41e+10 4.85e+10 1.60e+11 6.41e+11 4.85e+11
[9] 1.60e+12 6.41e+12 4.85e+11 1.60e+12
```

Note that this is doing is multiplying each element of G.p by the corresponding element of G.s. We could break this up like this

```
G.p[1]*G.s[1]
```

```
[1] 6.41e+08
```

```
G.p[2]*G.s[2]
```

```
[1] 4.85e+08
```

This works pretty well, but I found typing all those exponents to still be a little annoying, especially since many get repeated. I think its easier to just type the exponent. We'll call this object "G.exp" for exponent

```
G.exp <- c(3, 4, 5, 5, 6, 6, 7, 7, 7, 7, 7)
```

Again, if you want to be fancy you can play around with the rep() function, though this is optional

```
G.exp <- c(3, 4,
 rep(5, 2),
 rep(6, 3),
 rep(7, 5))
```

Now, to get the numbers we want, we can do this: G.p\*10^G.exp Can you figure out what's going on? If not, check the next chunk

Here I've added some parentheses to clarify the order of operations

```
G.p*(10^G.exp)
```

```
[1] 6.41e+08 4.85e+08 1.60e+10 6.41e+10 4.85e+10 1.60e+11 6.41e+11 4.85e+11
[9] 1.60e+12 6.41e+12 4.85e+11 1.60e+12
```

If its still not clear, then maybe this will help:

```
G.s == 10^G.exp
```

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

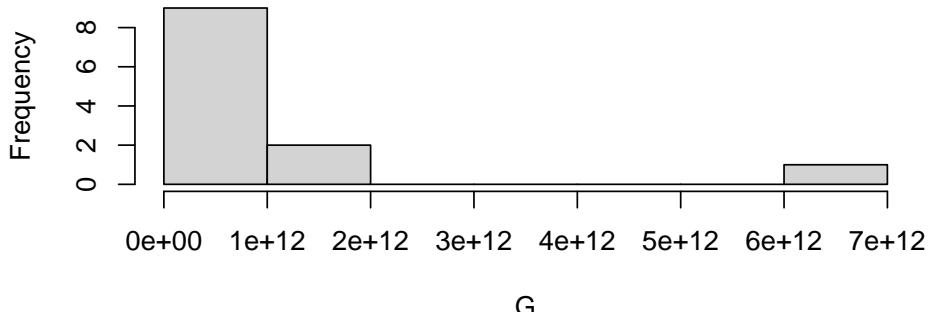
Let's make a final G vector with genome sizes. This is the "prefix" I made (G.p) multiplied by 10 raised to the exponent I defined in the vector G.exp.

```
G <- G.p*(10^G.exp)
```

We can make a basic plot showing the distribution of genome sizes using the histogram function in R, hist().

```
hist(G)
```

**Histogram of G**

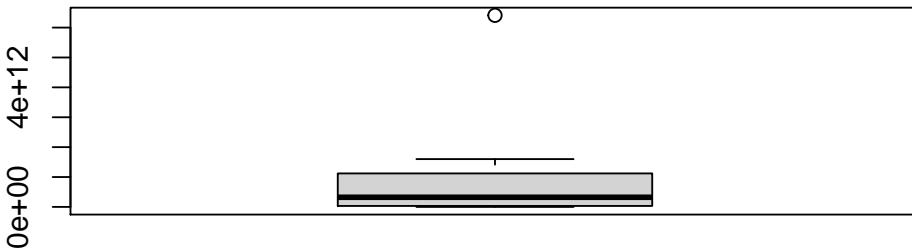


The far left value is 0e+00, which just means 0.0. The first tick mark is 1e+07. Write this out using an exponent symbol ^ below:

How do you interpret this graph? Ask your neighbor if they know. If you aren't comfortable interpreting histograms see [datavizcatalogue.com/methods/histogram.html](http://datavizcatalogue.com/methods/histogram.html) and <https://en.wikipedia.org/wiki/Histogram>

Another way to look at this data would be with a boxplot

```
boxplot(G)
```



This is actually a really ugly plot, but whatever.

`hist()` and `boxplot()` are great for quick and dirty plots. Normally though I'll use `ggpubr` to make plots using `gghistogram` and `ggboxplot`

## 22.8 Some more vectors

Next we have the per genome mutation rate. The “u” is a stand in for “mu” which is commonly used for mutation rates. So “u.bp” reads “mutation rate in bp.” First let’s do the “prefix” and assign it to a vector `u.bp.p` (mutation rate in bp prefix).

```
u.bp.p <- c(7.2, 7.7, 2.7, 2, 4.1, 6.9, 5.1, 2.8, 7.9, 1.7, 4.5, 4.6)
```

Confirm that this vector is the exact same size as our previous ones (Hint: requires “=”)

Confirm that what you just made was a vector. (Hint: what “is” it?)

Now let’s do the exponent and save it to a vector `u.bp.exp`.

```
u.bp.exp <- c(-7,
 -8, -8, -8,
 -10, -10, -10, -10,
 -9,
 -10,
 -11,
 -10)
```

For those who want to be fancy, can you do this using the `rep()` command? (Totally optional).

As before, make sure this new vector `u.bp.exp` is the right length

```
length(u.bp.exp)
```

```
[1] 12
```

## 136 CHAPTER 22. ASSIGNMENT: BUILDING DATAFRAMES BY HAND

Now, make the final column by carrying out the math. To do this put the following components together correctly

- The “prefix” u.bp.p
- The caret  $\hat{}$
- The exponent u.bp.exp

Assign the result to an object u.bp. Try to type this out yourself.

As always, check that the size of u.bp is correct

The last column is the per genome mutation rate, u.g.

```
u.g <- c(0.0046, 0.0038, 0.0043,
 0.0033, 0.0019, 0.0033,
 0.0024, 0.0038, 0.11,
 0.0024, 0.0019, 0.019)
```

We can confirm that the vector is that same size as the previous ones

```
length(G.s) == length(u.g)
```

```
[1] TRUE
```

I'm not 110% sure about the methods for this paper. I'm wondering if the per genome mutation rate u.g can be calculated from the per base pair mutation rate (u.bp) and the genome size (G). How do you think you could do this calculation in terms of the biology, the math, and in R? Save the output as an object u.g.2 (2nd version of u.g).

Hint: Don't over think it; there's just one mathematical operation, either multiplication or division (The answer is a bit further down).

Some more hints: G is the size of each genome. So the first genome in our vector has the size in base pairs (bp)

```
G[1]
```

```
[1] 6.41e+08
```

u.bp is the rate of mutations per base pairs. The mutation rate per base pairs for the first genome in the vector is

```
u.bp[1]
```

```
[1] 7.2e-07
```

So, if we want to determine the number of mutations per genome, we multiply the size of the genome in bp by the per bp rate of mutation

```
G[1]*u.bp[1]
```

```
[1] 461.52
```

We can compare this to what is in the table as the per genome rate of mutation (u.g)

```
u.g[1]
```

```
[1] 0.0046
```

We can do all the multiplication of the whole vector in one shot like this: . Run this code in the chunk below

So we have the value of u.g from the column in Table 1 that we typed up above, and now we have our attempt to re-calculate it. One way to check them against each other is to make a simple dataframe and compare by eye

```
data.frame(u.g, u.g.2)
```

```
u.g u.g.2
1 0.0046 461.520
2 0.0038 37.345
3 0.0043 432.000
4 0.0033 1282.000
5 0.0019 19.885
6 0.0033 110.400
7 0.0024 326.910
8 0.0038 135.800
9 0.1100 12640.000
10 0.0024 1089.700
11 0.0019 21.825
12 0.0190 736.000
```

This looks pretty close. Let's round things off so its easier to read. Rounding is done with the round() command.

To round something in R, we need to give it a number (or numbers in a vector) and an indication of how many digits to round off to. In the original table the ubmres are rounded off to 4 digits

```
round(0.0037, 4)
```

```
[1] 0.0037
```

Instead of u.g.2, run the previous data.frame code with round(u.g.2, 4). That is, instead of u.g.2, put in round(u.g.2, 4)

THings don't round off perfectly, perhaps because of other rounding they did during their workflow. But there's something weird with the first number. Either I made a type or somethign else is up. Let me know what you think!

Another useful comparison here is ==. See if you can predict what will happen when you run: u.g == round(u.g.2, 4)

When doing comparisons like this, why might it be really important to consider rounding?

## 22.9 Make a dataframe

We can put all our pieces together into a dataframe like this with `data.frame()`

```
table1 <- data.frame(organism = org,
 G = G,
 target = NA,
 u.bp = u.bp,
 u.g = u.g)
```

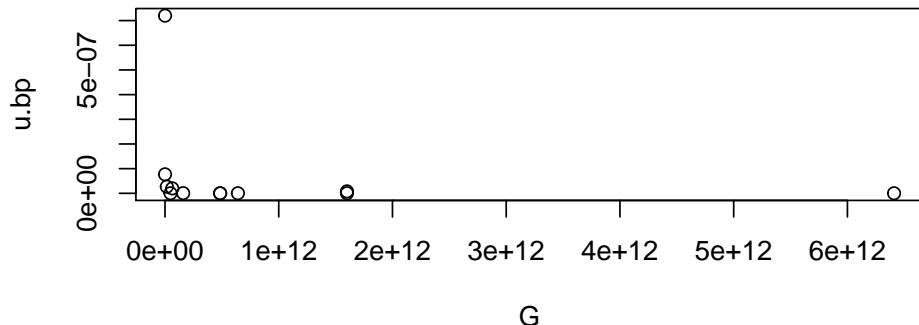
For each column I am specifying a name and telling it the vector to turn into a column. For target I'm telling it just to fill it in with NA, which means there is no data

As always, we want to check what we just made. Run at least 2 commands exploring the size, shape or content of the datarame

## 22.10 Make Figure 1 with base R

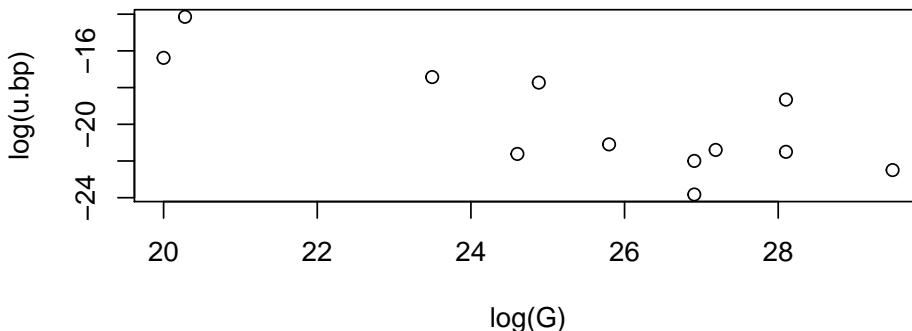
We can make figure 1 with the basic R `plot()` command. FIrst let's plot the raw data.

```
plot(u.bp ~ G, data = table1)
```



If you look at the figure its actually on the log scale. We can nest the log function within the plot function like this

```
plot(log(u.bp) ~ log(G), data = table1)
```



Check out the y axis and the x axis of the plot and compare it to the figure (get the figure at <https://www.pnas.org/content/88/16/7160>). What's wrong?

R's default log() function uses the natural log ln. To do the base 10 log change the code to use log10

## 22.11 Make Figure 1 with ggpublisher

ggpubr makes nicer plots than base R, but you can't nest functions in it. So what we need to do is make a column of the logged variables. For Genome size it requires this:

```
table1$log10.G <- log10(table1$G)
```

Now make a new column for logged u.bp, using log10. Call it log10.u.bp. Try to type it out yourself.

While we're at it let's make a log10 mutation rate per genome column called log10.u.g

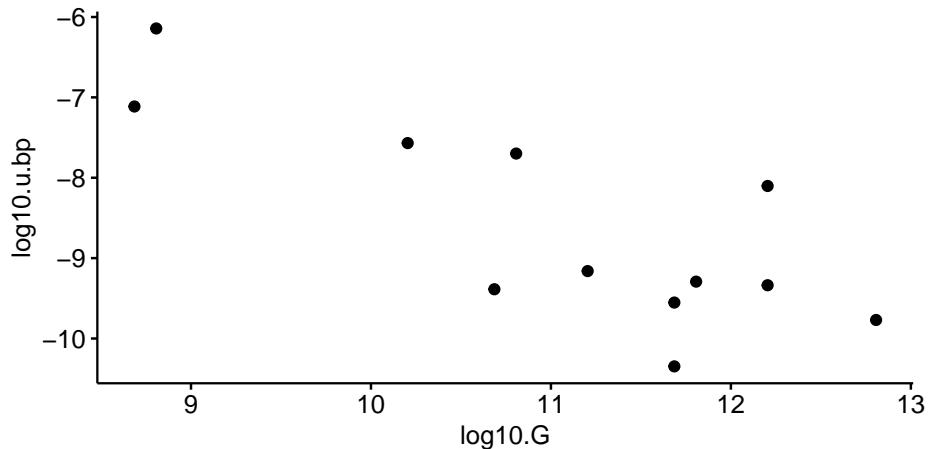
Check our columns

```
summary(table1$log10.u.bp)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
-10.347 -9.429 -9.227 -8.623 -7.666 -6.143
```

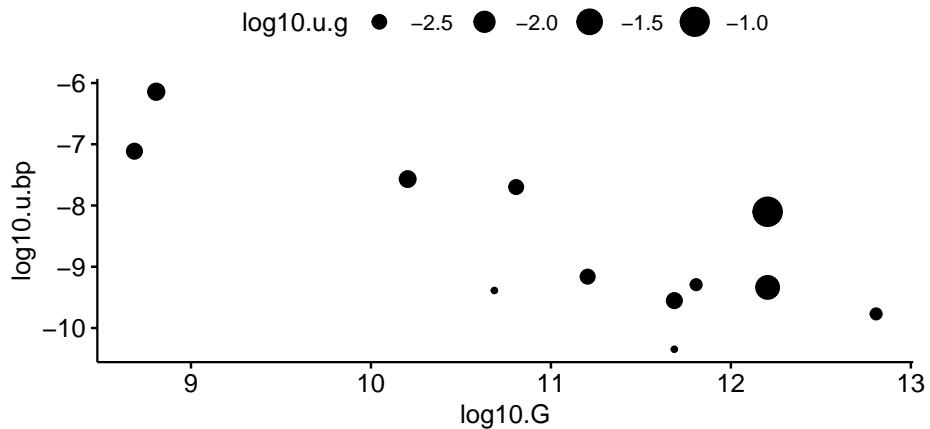
We can make scatter plot using ggscatter

```
library(ggpubr)
ggscatter(data = table1,
 y = "log10.u.bp",
 x = "log10.G")
```



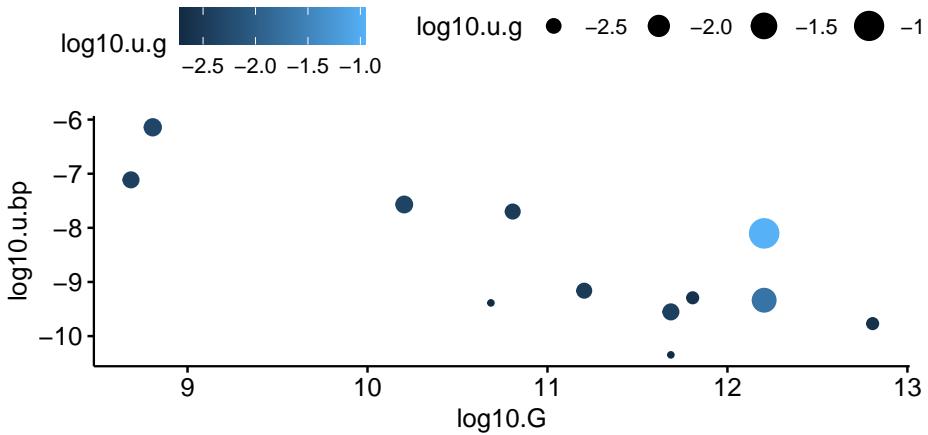
We can make lots of nice tweaks with `ggpubr`. Let's vary the size of our data points based on the log of the mutation rate per genome, `log10.u.g`. This lets assess another pattern on top of the main correlation between G and u\_bp

```
ggscatter(data = table1,
 y = "log10.u_bp",
 x = "log10.G",
 size = "log10.u.g") #size
```



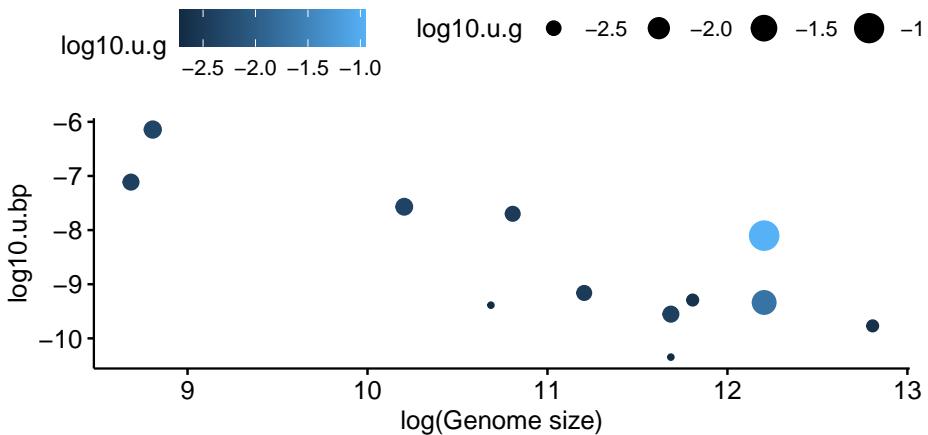
Now let's add color based on `log10.u.g` also

```
ggscatter(data = table1,
 y = "log10.u_bp",
 x = "log10.G",
 size = "log10.u.g", #size
 color = "log10.u.g") #color
```



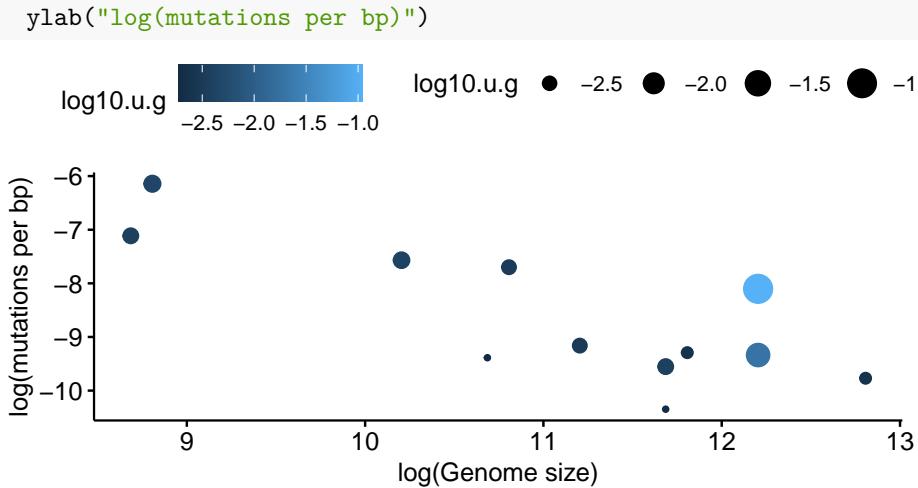
The x axis isn't very clear using the normal column name. Can you spot what is different about the code below?

```
ggscatter(data = table1,
 y = "log10.u.bp",
 x = "log10.G",
 size = "log10.u.g",
 color = "log10.u.g") +
 xlab("log(Genome size)")
```

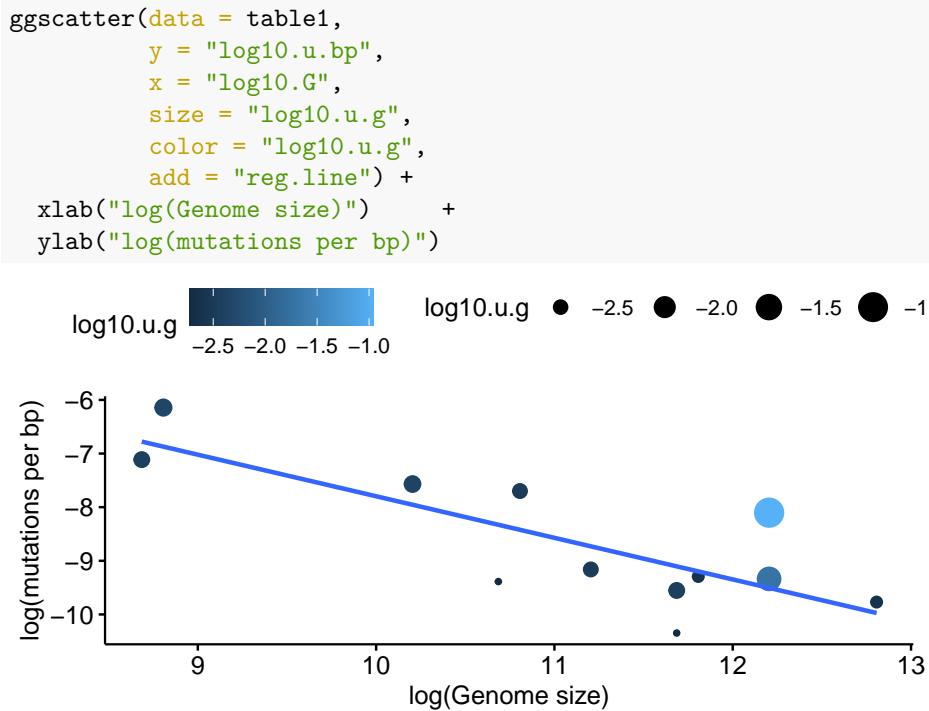


Now let's re-label the y axis. Note the plus signs after the lines with xlab() and ylab(). What do you think they are doing?

```
ggscatter(data = table1,
 y = "log10.u.bp",
 x = "log10.G",
 size = "log10.u.g",
 color = "log10.u.g") +
 xlab("log(Genome size)") +
 ylab("log10.u.bp")
```

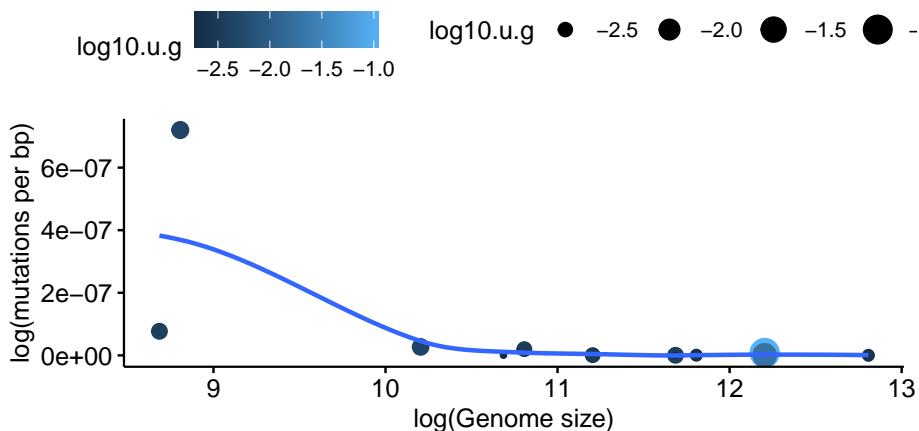


There is a strong correlation between genome size and mutation rate (on the log scale). We can further emphasize this pattern with a regression line. Note that within the call to `ggscatter()` I've added `add = "reg.line"`



We can also fit curved lines easily if we want. See if you can figure out what's different about this plot. (The line I've added is called a **smoother** made using a specific statistical method called **loess**.)

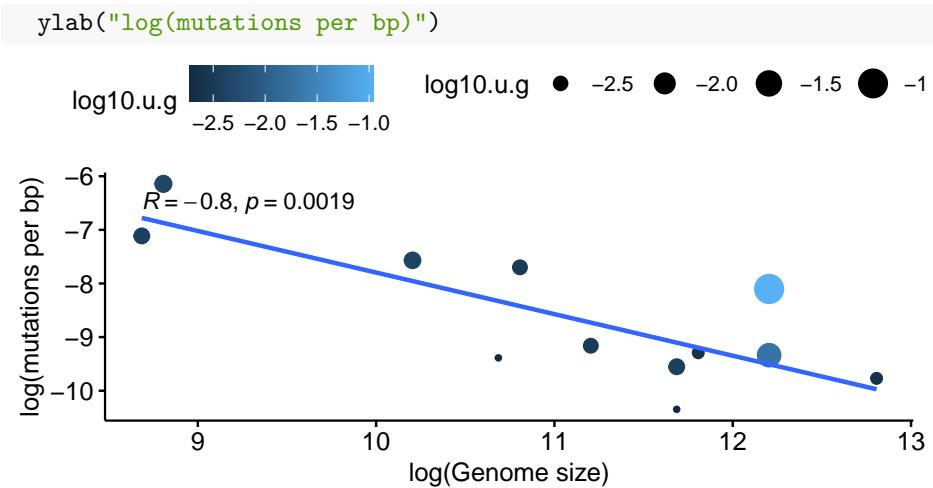
```
ggscatter(data = table1,
 y = "u_bp",
 x = "log10.G",
 size = "log10.u.g",
 color = "log10.u.g",
 add = "loess") +
 xlab("log(Genome size)") +
 ylab("log(mutations per bp)")
```



## 22.12 Add statistics

We can do some quick statistics on the fly here by adding `cor.coef =TRUE`. What this does is calculates the strength of the relationship between the x and the y variable (on the log scale), quantifies it with a statistic R, and calculates statistically a p-value to give us a sense of whether R is not 0. This is a very strong relationship, so R is almost -1 (R varies between -1 and 1) and the p-value is very small. Note that the technical definition of a p-value has a very precise meaning; generally consider  $p < 0.05$  to be “significant”. This means that the pattern we’re seeing in the data is unlikely to have occurred due to chance; however, it does not mean that if we repeated the study with a new data set that we wouldn’t get a different result. That is, it doesn’t mean that our conclusions are true!

```
ggscatter(data = table1,
 y = "log10.u_bp",
 x = "log10.G",
 size = "log10.u.g",
 color = "log10.u.g",
 add = "reg.line",
 cor.coef =TRUE) +
 xlab("log(Genome size)") +
```



# Chapter 23

## Dataframe by hand - problem set - KEY

```
library(compbio4all)
```

HAS SCORING FUNCTION AT END

SAVE THIS FILE TO A SAFE PLACE WHERE YOU CAN RELOCATE IT

In this exercise we will rebuild a data table and figure in a paper by kimura\_KEY et al. “Growth temperatures of archaeal communities can be estimated from the guanine-plus-cytosine contents of 16 S rRNA gene fragments”. The paper can be downloaded from <https://doi.org/10.1111/1758-2229.12035>

### 23.1 Preliminaries

This data is found in the dayhoff package. The only other package you need it ggpibr for plotting.

```
library(ggpibr)
```

### 23.2 Step one: Make each column

Read the data off of the original Table 1 and assign them to the objects put in place below. Delete the NA.

```
strain initials (eg, "Methanocalculus pumilus" = "MP")
strain <- c("MP", "NB", "MO", "MH", "SA", "SM", "PO")

Growth temperatures
```

```

T.min <- c(25, 25, 40, 50, 57, 70, 80)
T.opt <- c(35, 45, 60, 70, 80, 92, 105)
T.max <- c(45, 55, 75, 80, 89, 98, 110)

Accession number; leave this code as is
accession <- rep(NA, length(strain))

Length of 16sRNA gene in BP
length.bp <- c(790,796,791,808,808,810,808)

Percent GC
P.GC <- c(55.4,57.8,59.8,62.3,64.1,66.5,68.9)

Melting temperature
T.m <- c(86.2,87.2,88.2,88.9,89.7,90.3,90.9)

References - leave this as is
reference <-rep(NA, length(strain))

```

### 23.2.1 Quality control

In the chunk below, type a command to check the size of one of the objects you just made.

```
length(T.m)
```

```
[1] 7
```

In the chunk below, type a command to check the type of **data structure** of the object you just made.

```
is(T.m)
```

```

[1] "numeric" "vector" "index" "repValue"
[5] "numLike" "number" "atomicVector" "numericVector"
[9] "repValueSp" "Mnumeric"
is.vector(T.m)

```

```
[1] TRUE
```

```
class(T.m)
```

```

[1] "numeric"
is.numeric(T.m)

```

```
[1] TRUE
```

```
typeof(T.m) #acceptable, but this doesn't tell you data structure
```

```
[1] "double"
```

### 23.3 Basic Data exploration

What is the mean T.opt? Replace the NA and type the appropriate code below

```
mean.T.opt <- mean(T.opt)
mean.T.opt
```

```
[1] 69.57143
```

What is the maximum P.GC value?

```
max.P.GC <- max(P.GC)
max.P.GC
```

```
[1] 68.9
```

### 23.4 Part 2: Build the dataframe

Now, re-make Table 1 of kimura\_KEY et al (2013).

Type in the code necessary to build that dataframe; assign it to an object called “kimuar”

```
Delete the "NA" and add the appropriate code
kimura_KEY <- data.frame(strain,
 T.min, T.opt, T.max,
 accession,
 length.bp, P.GC, T.m,
 reference)
```

```
kimura_KEY
```

	strain	T.min	T.opt	T.max	accession	length.bp	P.GC	T.m	reference
## 1	MP	25	35	45	NA	790	55.4	86.2	NA
## 2	NB	25	45	55	NA	796	57.8	87.2	NA
## 3	MO	40	60	75	NA	791	59.8	88.2	NA
## 4	MH	50	70	80	NA	808	62.3	88.9	NA
## 5	SA	57	80	89	NA	808	64.1	89.7	NA
## 6	SM	70	92	98	NA	810	66.5	90.3	NA
## 7	PO	80	105	110	NA	808	68.9	90.9	NA

Using the dataframe as the source of the data, type a command which tells you the mean of the T.min values

```
mean(kimura_KEY$T.min)
```

```
[1] 49.57143
```

## 23.5 Part 3: Make figure

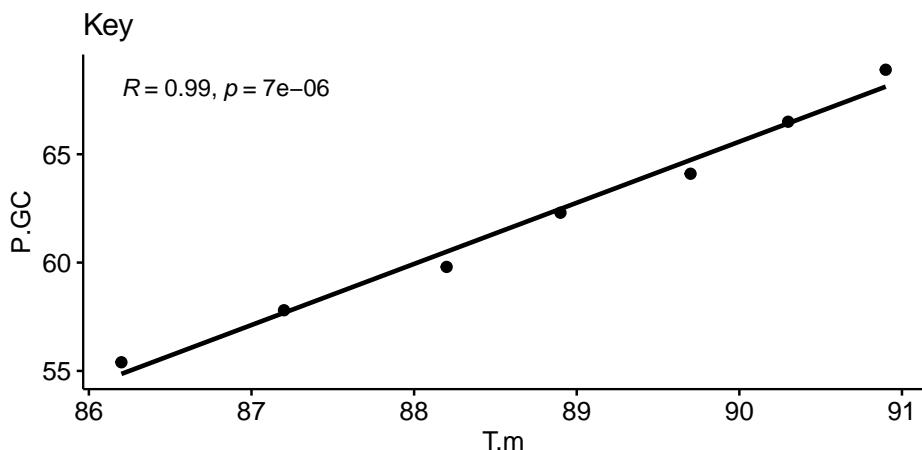
Re-make Figure 2 of kimura\_KEY et al (2013). **Include the following elements**

- Data points from Table 1
- Regression line (line of best fit)
- Correlation coefficient (will show up as “R”; Plot 2 in the paper actually has  $R^2$ )
- P-value (created by same argument as the correlation coefficient)

The original Figure 2 has the sample size listed and some small error bars around the data points. Don’t worry about this.

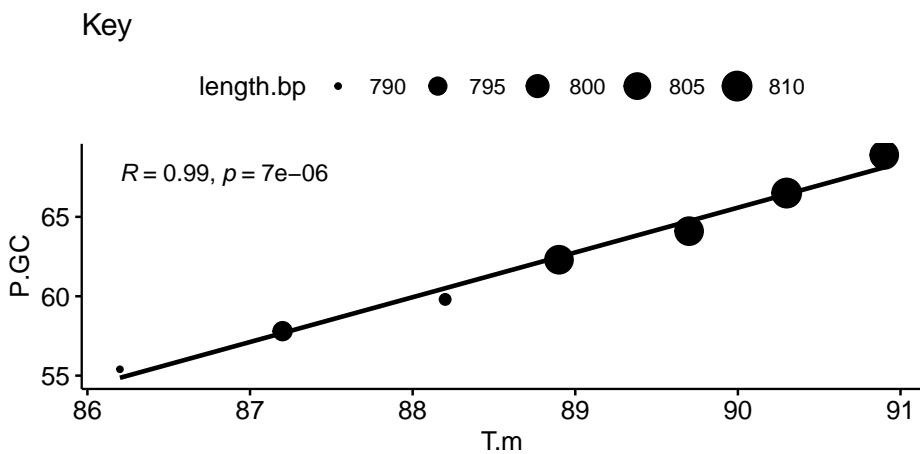
In addition to previous exercises you may wish to consult this site: <http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/78-perfect-scatter-plots-with-correlation-and-marginal-histograms/>

```
#Note: making plots doesn't involve the assignment operator
ggscatter(data = kimura_KEY,
 y = "P.GC",
 x = "T.m",
 add = "reg.line",
 cor.coef = T) +
 ggttitle("Key")
```



Now add the argument `size = "length.bp"` so that the size of the points is scaled to that column.

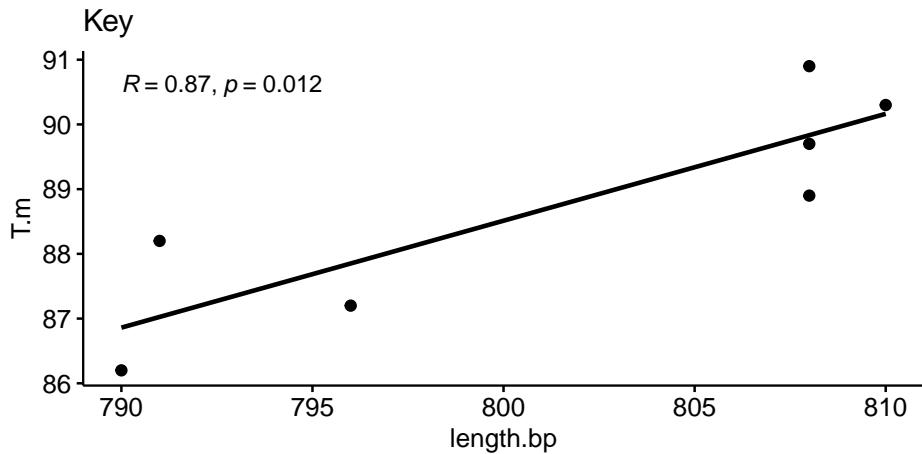
```
#Note: making plots doesn't involve the assignment operator
ggscatter(data = kimura_KEY,
 y = "P.GC",
 x = "T.m",
 add = "reg.line",
 size = "length.bp",
 cor.coef = T) +
 ggtitle("Key")
```



## 23.6 Graph of length.bp

Create a graph that shows the relationship between T. and length.bp

```
ggscatter(data = kimura_KEY,
 y = "T.m",
 x = "length.bp",
 add = "reg.line",
 cor.coef = T) +
 ggtitle("Key")
```



## 23.7 Create new variable: T.m/length.bp

Divide T.m by by length.bp; assign this to a column in the table called T.m.per.bp

```
kimura_KEY$T.m.per.bp <- kimura_KEY$T.m/kimura_KEY$length.bp
```

```
Other versions also work and are equivalent
T.m/length.bp == c(T.m/length.bp)
```

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

```
kimura_KEY$T.m/kimura_KEY$length.bp == c(T.m/length.bp)
```

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

### 23.7.1 Short answer question: What am I having you do this? (2-4 sentences)

Assign the mean length.bp to an object called mean.length.bp

```
mean.length.bp <- mean(kimura_KEY$length.bp)
```

Multiply the column T.m.per.bp by the value mean.length.bp. Assign it to a column in the dataframe called T.m.per.mean.bp

```
kimura_KEY$T.m.per.mean.bp <- kimura_KEY$T.m.per.bp*mean.length.bp
```

The new column created

```
mean(kimura_KEY$T.m.per.mean.bp)
```

```
[1] 88.76684
```

**23.7.2 Short answer question:** What am I having you do this? (2-4 sentences)

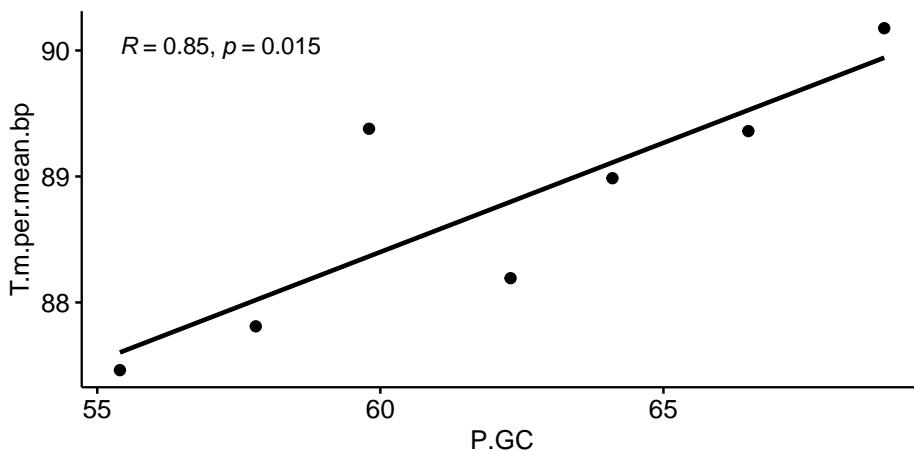
## 23.8 New plot: T.m801 versus P.GC

Create a graph like Figure 2 from the original paper but using T.m.per.mean.bp.

### 23.8.1 Correct axes

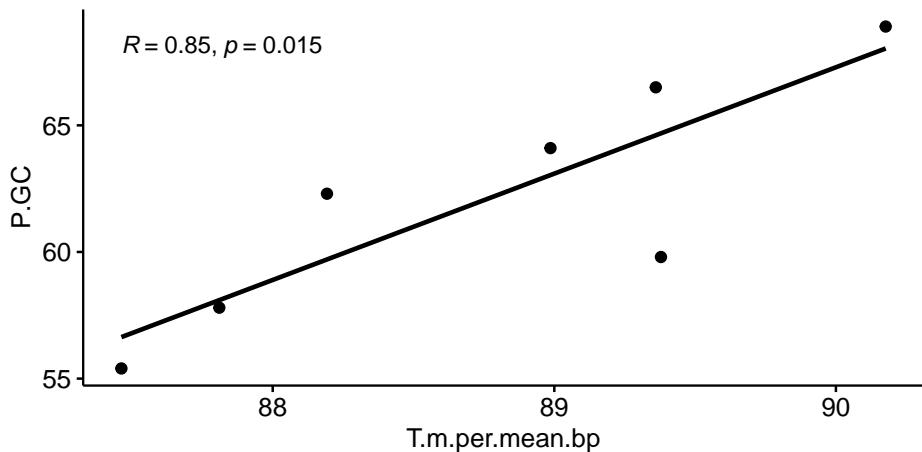
T.m (now T.m.per.mean.bp) on x-axis y axis is still P.GC

```
ggscatter(data = kimura_KEY,
 y = "T.m.per.mean.bp",
 x = "P.GC",
 add = "reg.line",
 cor.coef = T)
```



### 23.8.2 Flipped axes

```
ggscatter(data = kimura_KEY,
 x = "T.m.per.mean.bp",
 y = "P.GC",
 add = "reg.line",
 cor.coef = T)
```



**23.8.3 Short answer question:** What has changed with this new plot? Why? (3-5 sentences)

SAVE THIS FILE TO A SAFE PLACE WHERE YOU CAN RELOCATE IT

## 23.9 Grading function

```
T.min <- T.opt <- T.max <-
```

## **Chapter 24**

**Accession number; leave  
this code as is**

accession <-

154      *CHAPTER 24. ACCESSION NUMBER; LEAVE THIS CODE AS IS*

## Chapter 25

# Length of 16sRNA gene in BP

```
length_bp <-
```



# **Chapter 26**

# **Percent GC**

P.GC <-



## **Chapter 27**

# **Melting temperature**

T.m



# Chapter 28

## References - leave this as is

```
reference <-
rename final data frame
kimura <- kimura_KEY

#key objects and columns created
key.list <- list(#strain = c("MP", "NB", "MO", "MH", "SA", "SM", "PO"),
T.min = c(25, 25, 40, 50, 57, 70, 80),
T.opt = c(35, 45, 60, 70, 80, 92, 105),
T.max = c(45, 55, 75, 80, 89, 98, 110),

accession = rep(NA, length(strain)),
length.bp = c(790,796,791,808,808,810,808),
P.GC = c(55.4,57.8,59.8,62.3,64.1,66.5,68.9),
T.m = c(86.2,87.2,88.2,88.9,89.7,90.3,90.9),
reference = rep(NA, length(strain)),
mean.T.opt = mean(c(35, 45, 60, 70, 80, 92, 105)),
max.P.GC = max(c(55.4,57.8,59.8,62.3,64.1,66.5,68.9)),
mean.length.bp = mean(c(790,796,791,808,808,810,808)),
T.m.per.bp = c(86.2,87.2,88.2,88.9,89.7,90.3,90.9)/c(790,796,791,808,808,810,808),
T.m.per.mean.bp = c(86.2,87.2,88.2,88.9,89.7,90.3,90.9)/c(790,796,791,808,808,810,808)*mean(c(790,796,791,808,808,810,808))

score_assignment <- function(){
 score.dataframe <- data.frame(
 i = 1:length(key.list),
 object =names(key.list),
 exists = NA,
 correct = NA,
 stringsAsFactors = F)
 for(i in 1:length(key.list)) {
```

```

object.i.name <- score.dataframe$object[i]
score.dataframe$exists[i] <- exists(object.i.name, envir = .GlobalEnv)

see if object exists in memory
if(exists(object.i.name) == TRUE){
 object.i <- try(get(object.i.name,envir = .GlobalEnv))
 object.i.check <- key.list[[object.i.name]] == object.i
 object.i.score <- ifelse(any(object.i.check != TRUE) == FALSE, 1, 0)
 score.dataframe$correct[i] <- object.i.score
}

see if object stored just in dataframe
if(exists(object.i.name) == FALSE){
 object.i <- try(kimura[,object.i.name])
 object.i.check <- key.list[[object.i.name]] == object.i
 object.i.score <- ifelse(any(object.i.check != TRUE) == FALSE, 1, 0)
 score.dataframe$correct[i] <- object.i.score

 score.dataframe$exists[i] <- ifelse(score.dataframe$correct[i] == 1,TRUE,FALSE)
}

}
return(score.dataframe)
}

setwd("C:/Users/lisanjie/Downloads/problemset1")

files.rmd <- list.files(pattern = ".Rmd")
length(files.rmd)
scores.df <- data.frame(email = rep(NA,length(files.rmd)),
 score.exists = NA,
 score.correct = NA)

for(i in 1:nrow(scores.df)){

 # Hack: scrub from memory anything related to answers
 ## otherwise previous students work gets used to score current
 ## student
 all.files <- ls()
 i.keep <- which(all.files %in% c("scores.df", "files.rmd","key.list","score_assignment"))
 rm(list = all.files[-i.keep])

 rmd.file.i <- files.rmd[i]
}

```

```

rmd.file.i <- "Problem set 1 (for Test 1)_agf39_attempt_2019-10-01-13-43-55_data_frame_by_han

Read in raw text
tx <- readLines(rmd.file.i)

clean bad code
remove rm(list = ls())
tx <- gsub(pattern = "rm\\\\(list", replace = "#rm list", x = tx)

remove install.packages()
tx <- gsub(pattern = "install.packages", replace = "# install.packages", x = tx)

remove update.packages()
tx <- gsub(pattern = "update.packages", replace = "# update.packages", x = tx)

remove update.packages()
tx <- gsub(pattern = "system\\\\(", replace = "# system", x = tx)
tx <- gsub(pattern = "setwd\\\\(", replace = "# setwd", x = tx)
tx <- gsub(pattern = "source\\\\(", replace = "# source", x = tx)

writeLines(tx, con=rmd.file.i)

run the .rmd file and load into global environment
knitr::knit(rmd.file.i, quiet = F)

email.i <- gsub("(.*)([_])([a-z][a-z][a-z][0123456789][0123456789][.]*)([_])(.*)","\\3",rmd.file.i)

score.table.i <- try(score_assignment())
exists.score.i <- try(length(which(score.table.i$exists == TRUE)))
correct.score.i <- try(length(which(score.table.i$correct == TRUE)))

scores.df$email[i] <- email.i
scores.df$score.exists[i] <- exists.score.i
scores.df$score.correct[i] <- correct.score.i
#View(scores.df)
}

write.csv(scores.df, file = "scores_df2.csv")

```



# Chapter 29

## Dataframe by hand - problem set

```
library(compbio4all)
```

SAVE THIS FILE TO A SAFE PLACE WHERE YOU CAN RELOCATE IT

In this exercise we will rebuild a data table and figure in a paper by Kimura et al. “Growth temperatures of archaeal communities can be estimated from the guanine-plus-cytosine contents of 16 S rRNA gene fragments”. The paper can be downloaded from <https://doi.org/10.1111/1758-2229.12035>. Note that the article is paywalled so that if you are not on campus when you access it you will have to first log in to my.pitt.edu and then access the paper via the library.

### 29.1 Preliminaries

This data is found in the dayhoff package. The only other package you need is ggpubr for plotting.

```
library(ggpubr)
```

### 29.2 Step one: Make each column

Read the data off of the original Table 1 and assign them to the objects put in place below. Delete the NAs; these are placeholders for you to put your code. However, do leave the NAs for “accession” and “reference”; this will create a column of NAs that will serve as a placeholder (in the future I’ll probably type these up myself but haven’t yet).

### 29.2.1 Quality control

In the chunk below, type a command to check the size of one of the objects you just made.

```
TYPE CODE BELOW
```

In the chunk below, type a command to check the type of data structure of the object you just made.

```
TYPE CODE BELOW
```

## 29.3 Basic Data exploration

What is the mean T.opt? Replace the NA and type the appropriate code below

.

```
mean.T.opt <- NA
```

What is the maximum P.GC value?

```
max.P.GC <- NA
```

## 29.4 Part 2: Build the dataframe

Now, re-make Table 1 of Kimura et al (2013).

Type in the code necessary to build that dataframe; assign it to an object called “kimura”

```
Delete the "NA" and add the appropriate code
kimura <- NA
```

Using the dataframe as the source of the data, type a command which tells you the mean of the T.min values

```
TYPE CODE BELOW
```

## 29.5 Part 3: Make figure

Re-make Figure 2 of Kimura et al (2013). Include the following elements

- Data points from Table 1
- Regression line (line of best fit)
- Correlation coefficient (will show up as “R”; Plot 2 in the paper actually has  $R^2$ ; don’t worry about the difference in values)
- P-value (created by same argument as the correlation coefficient)

The original Figure 2 has the sample size listed and some small error bars around the data points. Don’t worry about this.

NOTE: the plot you can make with ggpublisher will have slightly different p-values and correlation coefficient values. No worries. The goal is to get the general theme of the plot.

In addition to previous exercises you may wish to consult this site: <http://www.sthda.com/english/articles/24-ggpubr-publication-ready-plots/78-perfect-scatter-plots-with-correlation-and-marginal-histograms/>

Now add the argument size = "length.bp" so that the size of the points is scaled to that column.

## 29.6 Graph of length.bp vs T.m

Create a graph that shows the relationship between T.m and length.bp

## 29.7 Create new variable: T.m/length.bp

Divide T.m by length.bp; assign this to a column in the table called T.m.per.bp

### 29.7.1 Short answer question: Why am I having you do this? (2-4 sentences)

What does dividing T.m by length.bp accomplish? (consider what the plot of length.bp vs T.m tells you about the inherent relationship between length and T.m and how that might confuse the relationship between T.m and P.GC )

Assign the mean length.bp to an object called mean.length.bp

```
mean.length.bp <- NA
```

Multiply the column T.m.per.bp by the value mean.length.bp. Assign it to a column in the dataframe called T.m.per.mean.bp

### 29.7.2 Short answer question: Why am I having you do this? (2-4 sentences)

What does multiplying T.m.per.bp by the mean length accomplish?

## 29.8 New plot: T.m801 versus P.GC

Create a graph like Figure 2 from the original paper but using T.m.per.mean.bp.

**29.8.1 Short answer question: What has changed with this new plot? Why? (3-5 sentences)**

Explain what is different about this plot of T.m801 versus P.GC from the original one of T.m. versus P.GC

SAVE THIS FILE TO A SAFE PLACE WHERE YOU CAN RELOCATE IT

# Chapter 30

## Making a simple dataframe

In this walkthrough we'll take a simple dataset that is presented as lists of separate numbers (eg 1,2,3,4) and put them into vectors. We'll then take these vectors and make simply dataframe. We'll then make a boxplot using ggplot2 and ggpubr.

### 30.1 Working with data in 2 seperate vectors

- Data often come in 2 separate sets of numbers, such as “the control group had lengths 4, 5, 6, 3, and 3, while the treatment group had length 11, 12, 6, 4 and 7”.
- One way to work with data like this in R is to load each set of numbers into a “vector” using the `c()` function

```
control <- c(4, 5, 6, 3, 3)
treatment <- c(11, 12, 6, 4, 7)
```

#### 30.1.1 Summary Statistics

We can calculate the mean and other summary statistics of each vector on its own

```
mean(control)
[1] 4.2
sd(control)
[1] 1.30384
summary(control)
Min. 1st Qu. Median Mean 3rd Qu. Max.
4 5 6 4.2 11.0 12
```

```
3.0 3.0 4.0 4.2 5.0 6.0
```

### 30.1.2 Making a dataframe

To plot data its is almost always best to set it up in a dataframe.

Starting with the raw data we could make a dataframe like this

```
dat <- data.frame(length = c(4, 5, 6, 3, 3,
 11, 12, 6, 4, 7),
 group = c("C", "C", "C", "C", "C",
 "T", "T", "T", "T", "T"))
```

We can make a dataframe from the two vectors we've already typed up like this by surrounding them with a c() like this ”c(control, treatment)”

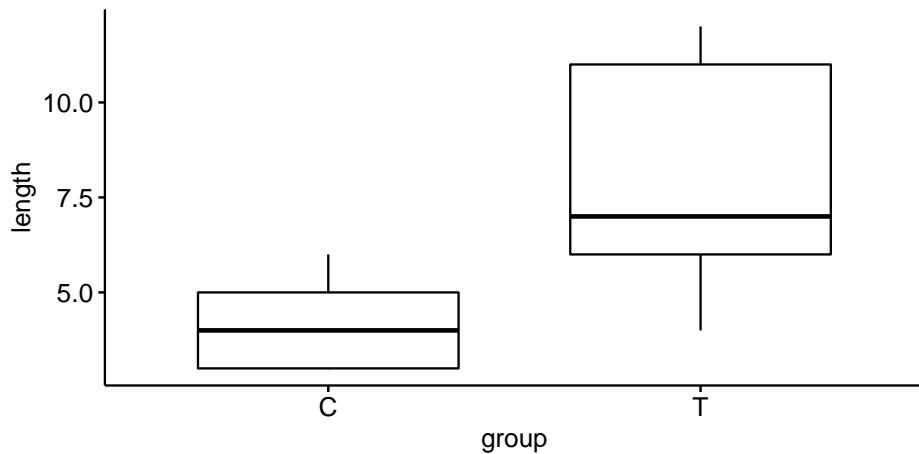
```
dat <- data.frame(length = c(control,
 treatment),
 group = c("C", "C", "C", "C", "C",
 "T", "T", "T", "T", "T"))
```

### 30.1.3 Making a boxplot

We can make a boxplot from the data like this using ggpibr

```
library(ggplot2)
library(ggpibr)

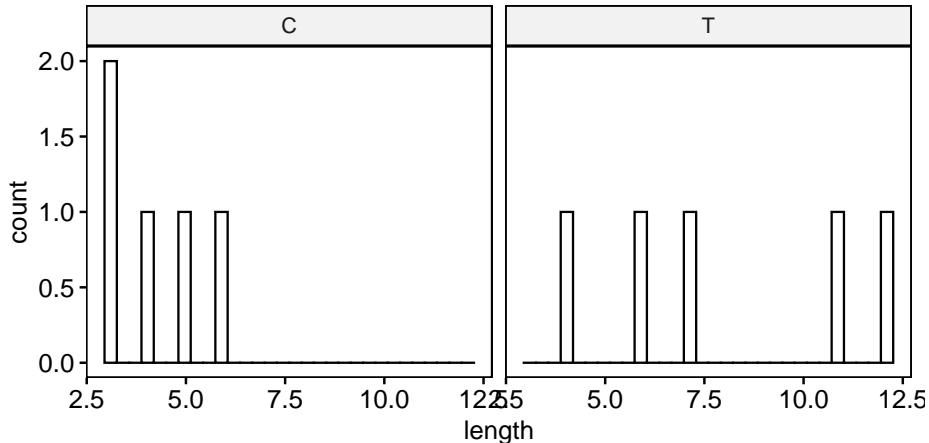
ggpibr::ggboxplot(data = dat,
 y = "length",
 x = "group")
```



### 30.1.4 Making a histogram

- An alternative to using a boxplot is a histogram.
- We can split up the data by treatment using facet.by = "group"
- Because my sample dataset has so few datapoints this doesn't look very good.

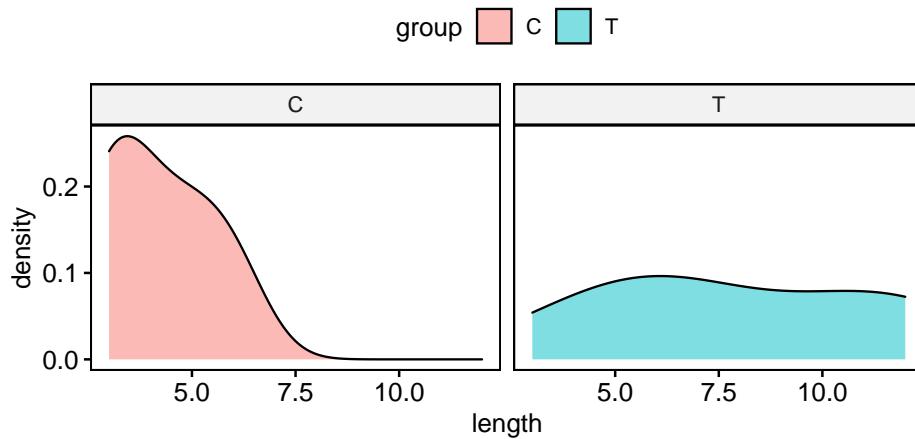
```
ggpubr::gghistogram(data = dat,
 x = "length",
 facet.by = "group")
```



### 30.1.5 Making a density plot

A plot that is similar to a histogram is a density plot.

```
ggpubr::ggdensity(data = dat,
 x = "length",
 facet.by = "group",
 fill = "group")
```



### 30.1.6 Advanced

You can save yourself some typing by using the `rep()` command when making your dataframe since the group codes “C” and “T” get repeated. IF you do this you need to make sure you keep track of all your parentheses.

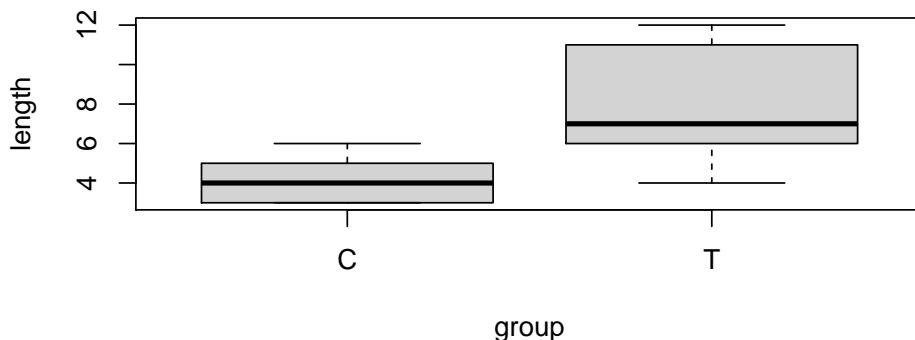
```
dat <- data.frame(length = c(control,
 treatment),
 group = c(rep("C", 5),
 rep("T", 5)))
```

### 30.1.7 Historical reference

#### 30.1.7.1 Base R

In the old days (eg 2008...) we used to make our boxplot using the `boxplot` function.

```
boxplot(length ~ group, data = dat)
```



In the less old days (eg 2015) we used to make our boxplots using regular `qplot` or `ggplot`

### 30.1.7.2 ggplot's qplot

```
ggplot2::qplot(data = dat,
 y = length,
 x = group,
 geom = "boxplot")
```

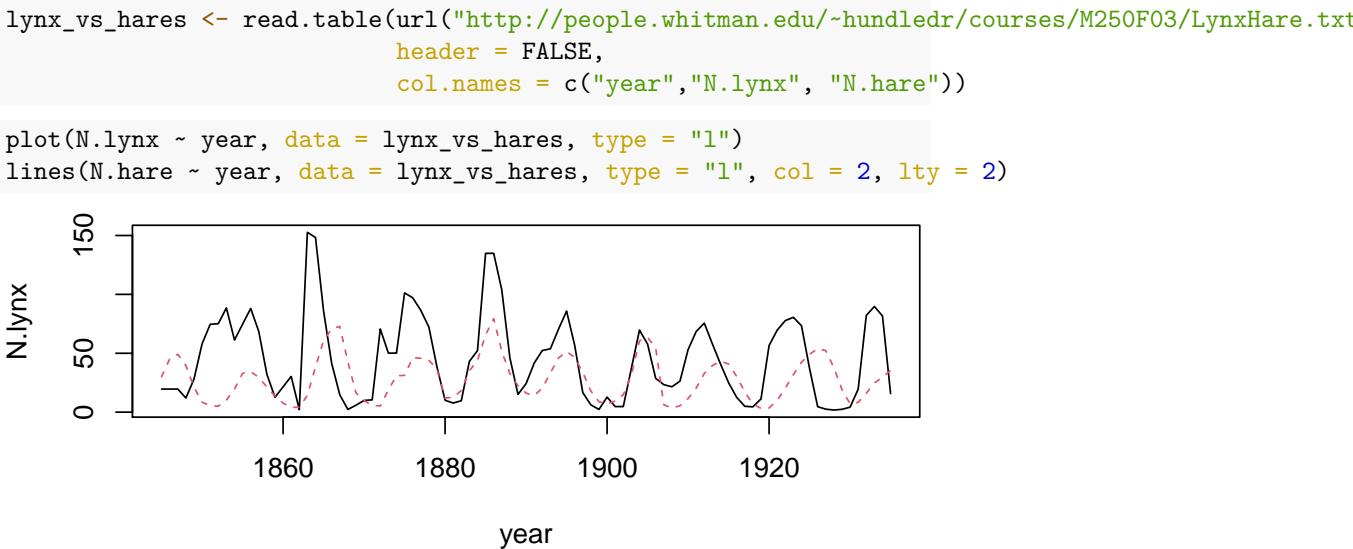
### 30.1.7.3 Standard ggplot

```
ggplot2::ggplot(data = dat,
 aes(y = length,
 x = group)) +
 geom_boxplot()
```



# Chapter 31

## Load data from internet



### 31.1 Introduction

Its possible to download data directly from the internet, including

- Spreadsheets directly posted online as .csv or .txt
- Spreadsheets contained within a GitHub repository, including a package
- Google Sheets
- and many other formats

In this short exercise we'll download some data that is stored as a raw .csv file within the inner workings of the `wildlifeR` package.

### 31.1.1 Learning Goals & Outcomes

By the end of this lesson students should be able to download basic data sources from the internet using `getURL()` from the `RCurl` package.

### 31.1.2 Functions & Arguements

- `RCurl::getURL()`
- `scatter.smooth`

### 31.1.3 Packages

- `RCurl install.packages()`

### 31.1.4 Potential hangups

- Bad internet connection
- Firewall problems

## 31.2 [ ] Downloading a .csv file using `getURL()`

The package `RCurl` provides functions for accessing online material. First we need the package

```
install.packages("RCurl", dependencies = TRUE) # []
```

As always, once we install a package we need to **really** install it with `library()`. (You might see some red text as `RCurl` loads up some of its **dependencies**)

```
library(RCurl) # []
```

We then use the `getURL()` to prep the info we need for downloading that .csv we want.

The file we want is “`eaglesWV.csv`”. It is located at this rather long URL:  
`https://raw.githubusercontent.com/brouwern/wildlifeR/master/inst/extdata/eaglesWV.csv`

First, we’ll use the “`<-`” assignment operator to store the shortened URL in an R object. Be sure to put the URL in quotes.

```
eaglesWV.url <- "https://raw.githubusercontent.com/brouwern/wildlifeR/master/inst/extda
```

Next we’ll use the `getURL()` function to set things up, storing the info in a new object “`eaglesWV.url_2`” (note the “`_2`” on the end).

First, get the URL from the URL-containing object we just made.

```
eaglesWV.url_2 <- getURL(eaglesWV.url)
```

Now use `read.csv()` to actually get it.

```
eaglesWV_2 <- read.csv(text = eaglesWV$url_2)
```

We can preview the downloaded dataset using `summary()` or any other command we want

```
summary(eaglesWV_2)
```

---

### 31.3 OPTIONAL: Plotting West Virginia Eagle Data

**This section is optional**

Thankfully, eagles having been increasing exponentially in West Virginia since the 1980s.

```
scatter.smooth(y = eaglesWV_2$WV, x = eaglesWV_2$year)
```

**End optional section**

---



# Chapter 32

## Worked example: Data visualization in R

```
library(compbio4all)
```

In this chapter we'll test out some of R's **data visualization** capabilities by exploring the chemical properties of amino acids. The chemical properties of amino acids are very important for understanding sequence evolution because mutations are more likely to replace an amino acid with a chemically similar one than with one that is very different. For example, a mutation is more likely to change one hydrophilic amino acid for another than to change a hydrophilic amino acid to a hydrophobic one. This is because the precise secondary and tertiary structure - and thus the function – of a protein depends on the chemical properties of all of its amino acids.

### 32.1 Preliminaries

#### 32.1.1 Packages

First, load general packages for plotting. There are many ways to plot data in R, but the most popular and powerful these days is ggplot2. We'll be using an extension of ggplot2 called ggpubr.

```
library(ggplot2)
library(ggpubr)
```

#### 32.1.2 Data

Load data with the `data()` command

```
data(aa_chars_subset)
```

This name is long, so let's use the **assignment operator** `<-` to give it a shorter name, just "aa". Run the code to do this.

```
aa <- aa_chars_subset
```

## 32.2 Data exploration: Getting to know data

Our data in this particular case is a **data.frame**, which is kind of like a **spreadsheet** in R-land. We can confirm this using the `is()` command. In the code chunk below type `is(aa)` and run the chunk. Four or so things will show up in the console. When you do this usually the first one is the most relevant one.

Spreadsheets are grids of data, but we can't see yet what we're looking at. This is a major obstacle in moving from working with a visual tool like Excel to R. We can use R commands to start to get an idea of what we are working with.

First question: How big is this grid? The `dim()` command tells us how large things like dataframes and matrices are in R. Type `dim(aa)` in the code chunk below. The first number will be the number of rows, the second will be the number of columns.

Ok, so we are getting a sense for this dataframe thingy, but its still a bit abstract. If we want to see the whole thing like it was a spreadsheet we can use `View()`. Type `View(aa)` in the chunk below. Note that its "V" uppercase, not lower case. This is kind of annoying, but R is case sensitive.

What happens when you run `View()` on `aa`?

Often we'd like to see the dataframe, or at least part of it, in the R console or in our R **notebook**. We can see part of the dataframe with `head(aa)`. Type this in the chunk below and run it. R might print things out in a weird way depending on how wide your screen is. You can adjust the width of the **RStudio pane** you working in or zoom out and re-run the command to see if it helps.

We can also use `tail()` to see the bottom of the dataframe. Type `tail(aa)` in the chunk below.

If we want to see the whole dataframe we can just type `aa` in a chunk and run it.

```
aa
```

Sometimes we may want to just see the names of all the columns. This is done with `names(aa)`. Type this code and run it below

### 32.3 Numeric summaries data

We can get numeric summaries of data using the `summary()` command. Type `summary(aa)` in the chunk below and run it.

Data can take many forms. If data are **numeric** then you can calculate means, medians, minimums, and maximums (we'll ignore "1st Qu" and "3rd Qu" for now). If data are **categorical** you can count up how many rows of data are in each category.

It can be useful to get a summary of just one column. You can do this by selecting the column with a dollar sign like this: `summary(aa$volume)`. Type this in the chunk below and run it (don't type the period)

There are times when we might just want one type of numeric summary, like the mean, for a column. In that case we can do this: `mean(aa$volume)`. Type that below.

In the assignment associated with this exercise you will be asked to provide the mean for a column. Run the appropriate code in the chunk below and enter the answer on TopHat. (Note that the exercise might not be activated until after class).

In the assignment associated with this exercise you will be asked to provide the max for a column. This is done with the `max()` command. Run the appropriate code in the chunk below and enter the answer on TopHat.

For **categorical variables** data are described by discrete words and fall into groups. For example, the "hydropathy" column describes each amino acid as hydrophobic, phydrophilic, or neutral. We can see how many amino acids are in each group using `summary(aa$hydropathy)`. Run this code in the chunk below.

It doesn't make sense to take the mean, max, etc of categorical data. If you try, R will **throw and error**, though often what it says is pretty cryptic.

```
mean(aa$letter)
```

```
[1] NA
```

When we have two categorical variables, like charge and hydropathy, we may want to know how the data fall out into both categories at the same time. For example, we have a hydropathy column and a polarity.cat column. How many amino acids are hydrophilic AND polar?

We can do this by typing `table(aahydropathy, aapolarity.cat)`. Remember, though, that we can let RStudio look up the variables names. Practice this by typing `table(aa$` and then waiting a second. You can then select from the list of variable names. Add the comma, then type `aa$` again and select from the list again. Finish with the right parentheses.

## 32.4 Plotting data

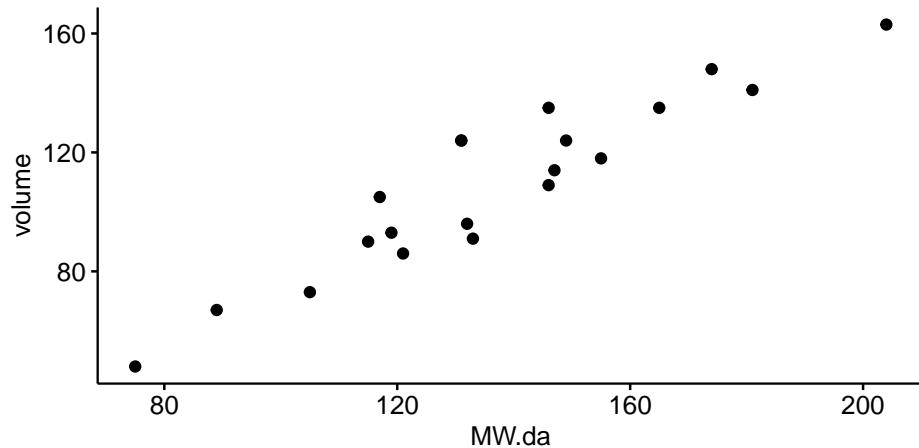
Numeric summaries from `summary()`, `mean()`, and `table()` are useful, but pictures are usually most helpful. We're going to use a plotting package called `ggpubr`, which extends the famous `ggplot2` package by Hadley Wickham. Make sure that you have run the code library(`ggpubr`) to get `ggpubr` R into memory..

These data have a column called `MW.da`, which is the molecular weight in daltons, and also a column called `volume`. What is the relationship between the two variables? Make a guess.

We can visualize the relationship with a **scatterplot**. A scatterplot has a **numeric variable** on the x-axis and a different numeric variable on the y-axis. The code below makes a simple scatter plot. Note the following things

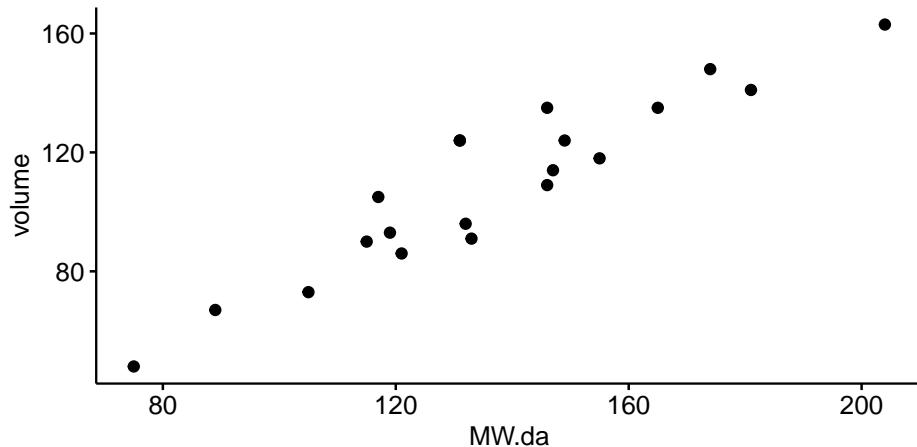
1. there are 3 arguments, `data`, `x` and `y`
2. an equals sign (`=`) assigns a value to an argument
3. `aa` is NOT in quotes, while `MW.da` and `volume` are. This is a bit annoying.

```
ggscatter(data = aa, x = "MW.da", y = "volume")
```



One thing about R code is that it can span multiple lines. In the code below click after the first command and press enter, then click after the second comma and press enter. Then run the chunk. What happens?

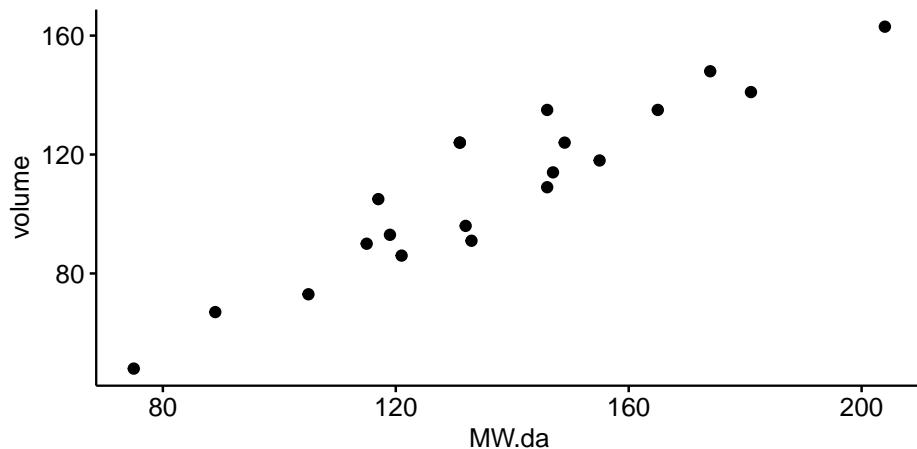
```
ggscatter(data = aa, x = "MW.da", y = "volume")
```



Writing code on multiple lines can make it easier to read. It also allows you to add comments to each part of a piece of code. To make a comment, we need the **comment character**, which in R is the hashtag #.

In the code below put a # after each line, and add a note about what the line does, then run the code.

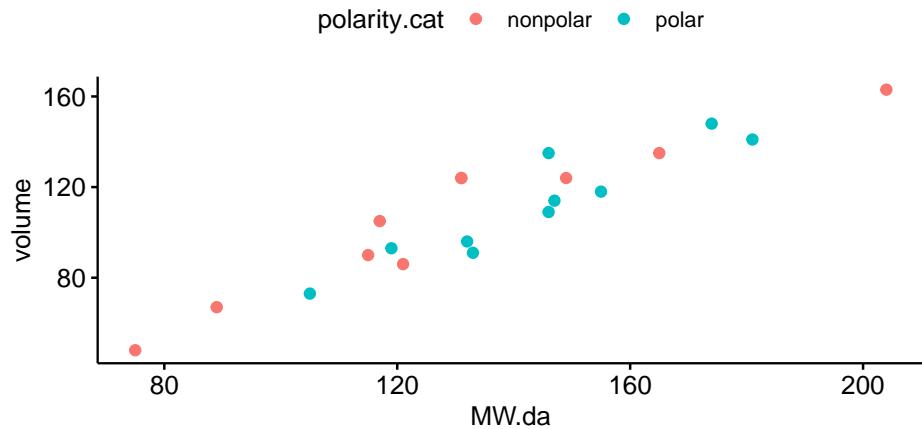
```
ggscatter(data = aa,
 x = "MW.da",
 y = "volume")
```



## 32.5 Adding color to a plot

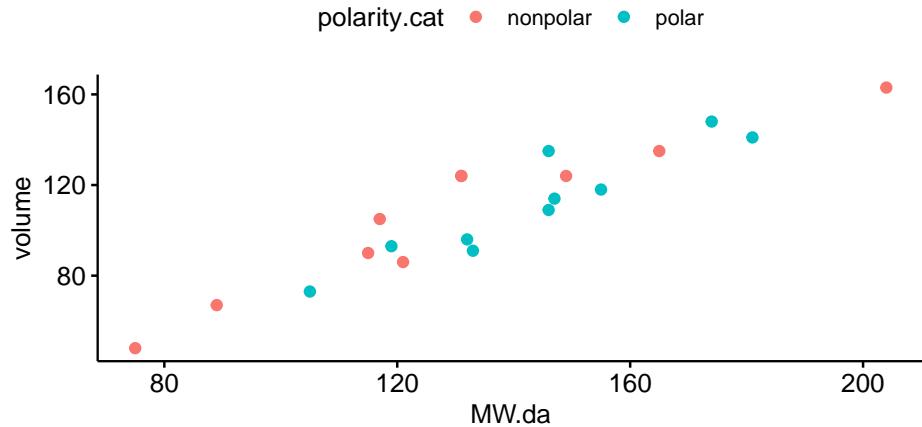
We can color code things easily in R using categorical variables. Let's assign a different color to the different polarities. We do this by add in "color =" to the code. Note that we have to put the categorical variable name in quotes "

```
ggscatter(data = aa,
 x = "MW.da",
 y = "volume",
 color = "polarity.cat")
```



We could also color code by a different variable, such as charge, hydropathy, or volumne.ca. Selection one of these and replace “polarity.cat” with it.

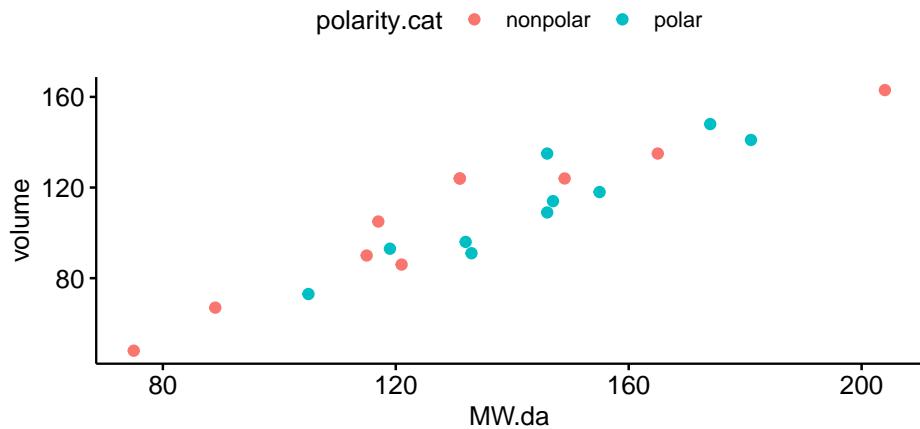
```
ggscatter(data = aa,
 x = "MW.da",
 y = "volume",
 color = "polarity.cat") # replace this; leave the quotes
```



## 32.6 Changing symbols

We can also make a plot more interesting by changing the plotting symbols. In the code below change the word color to shape. (Both of these are called arguments)

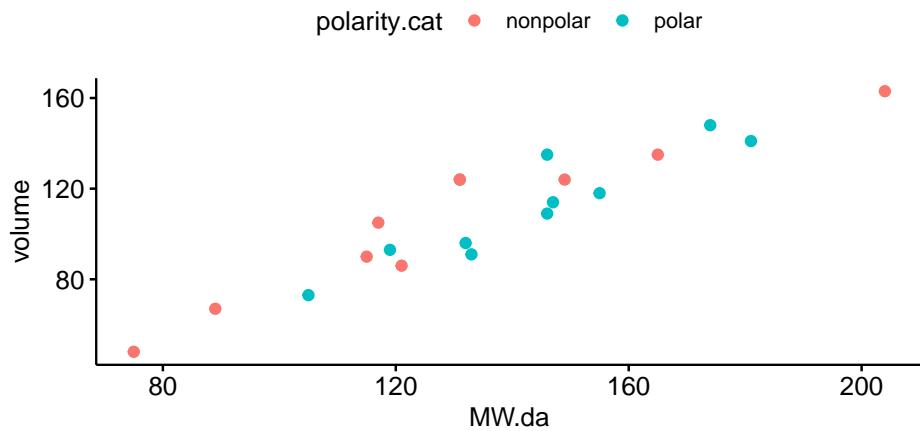
```
ggscatter(data = aa,
 x = "MW.da",
 y = "volume",
 color = "polarity.cat") # replace this; leave the quotes
```



## 32.7 Adding more stuff to a plot

The relationship between the two variables is pretty obvious, but we can emphasize it by adding a line of best fit, aka a regression line. In the code below remove the word “color” and change the word for a categorical variable to reg.line.

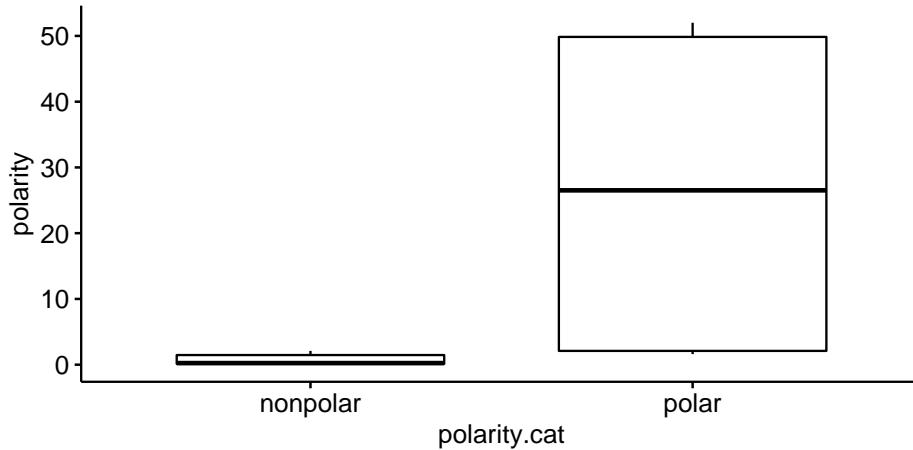
```
ggscatter(data = aa,
 x = "MW.da",
 y = "volume",
 color = "polarity.cat")
```



## 32.8 Boxplots

Another way to look at data is to plot categorical variables along the x-axis and a numeric variable on the y. We can do this with boxplots and the function `ggboxplot()`. Polarity can be measured quantitatively, so we can compare the categories in `polarity.cat` to the numeric values in `polarity`.

```
ggboxplot(data = aa,
 x = "polarity.cat",
 y = "polarity")
```



In a boxplot the thick line shows you the median. Most of the data points fall within the boxes, and the lines that extend up and down (“whiskers”) usually extend to the max and minimum. What is the approximate median (to the nearest 5 units) of polar amino acids? How much variation is there? (Answer on TopHat)

Hydrophobicity is a major characteristic of amino acids. This data set has two variables, `hydrophobe.34` and `hydrophobe.34` (the numbers just refer to different references where the values were derived). Make a boxplot using `ggboxplot` with `polarity.cat` on the x axis (as above) `hydrophobe.34` on the y axis. Remember to put things in quotes as needed.

## 32.9 Multivariate groups

There are many important characteristics of amino acids. When data have multiple variables like this it is often called **multivariate data**. (There is an important difference between multivariate data and multivariable analyses, but I'll let your stats profs teach about that). Multivariate data can be hard to make sense of. A classic way to think about amino acids is to make a **Venn Diagram** showing similarities between amino acids. You can see this Venn Diagram here: <http://www.russelllab.org/aas/>. Amino acids that are close together are

generally similar, especially if they are within the same circle. So I and L are similar to each other, and Y and W are similar, but I and Y are not so similar.

These Venn Diagrams are based on deep knowledge of the chemistry of the amino acids. Let's see if we can use a computer algorithm called **cluster analysis** to arrive at similar conclusions.

First, for cluster analysis we are only going to use the **numeric variables**. Let's get rid of those variables. The first column has our amino acid letter, and columns 2 through 11 have numeric variables. (column 12 is a numeric variable, relative frequency, but we're going to drop it).

To make our subset we're going to use R's **square bracket notation**.

First, run the code below and think about what happened

```
aa[,]
```

Now run this code and think about it:

```
aa[, 1]
```

Now run this code and think about it.

```
aa[1,]
```

```
letter MW.da volume bulkiness polarity isoelectric.pt hydrophobe.34
1 A 89 67 11.5 0 6 1.8
hydrophobe.35 saaH2O faal.fold polar.req freq charge hydropathy volume.cat
1 1.6 113 0.74 7 7.8 un hydrophobic verysmall
polarity.cat
1 nonpolar
```

Now instead of 1 type 1:2. Try both scenarios.

Ok, now let's make our subset. We want the second through 11th column, and all of the rows

```
aa_subset <- aa[, 2:11]
```

Check that you got what you want using head(aa\_subset)

Now we're going to do something a bit goofy with the names of the amino acids. What we need to do is assign them to the row names of our aa\_subset dataframe. Just run the code below

```
row.names(aa_subset) <- aa$letter
```

Now look at what we have. What's on the far left hand side of what gets displayed?

```
head(aa_subset)
```

```
MW.da volume bulkiness polarity isoelectric.pt hydrophobe.34 hydrophobe.35
```

```

##> #> A 89 67 11.50 0.00 6.00 1.8 1.6
##> #> C 121 86 13.46 1.48 5.07 2.5 2.0
##> #> D 133 91 11.68 49.70 2.77 -3.5 -9.2
##> #> E 146 109 13.57 49.90 3.22 -3.5 -8.2
##> #> F 165 135 19.80 0.35 5.48 2.8 3.7
##> #> G 75 48 3.40 0.00 5.97 -0.4 1.0
##> #> saaH20 faal.fold polar.req
##> #> A 113 0.74 7.0
##> #> C 140 0.91 4.8
##> #> D 151 0.62 13.0
##> #> E 183 0.62 12.5
##> #> F 218 0.88 5.0
##> #> G 85 0.72 7.9

```

Compare that to

```
head(aa)
```

```

##> letter MW.da volume bulkiness polarity isoelectric.pt hydrophobe.34
##> 1 A 89 67 11.50 0.00 6.00 1.8
##> 2 C 121 86 13.46 1.48 5.07 2.5
##> 3 D 133 91 11.68 49.70 2.77 -3.5
##> 4 E 146 109 13.57 49.90 3.22 -3.5
##> 5 F 165 135 19.80 0.35 5.48 2.8
##> 6 G 75 48 3.40 0.00 5.97 -0.4
##> hydrophobe.35 saaH20 faal.fold polar.req freq charge hydrophathy volume.cat
##> 1 1.6 113 0.74 7.0 7.80 un hydrophobic verysmall
##> 2 2.0 140 0.91 4.8 1.10 un hydrophobic small
##> 3 -9.2 151 0.62 13.0 5.19 neg hydrophilic small
##> 4 -8.2 183 0.62 12.5 6.72 neg hydrophilic medium
##> 5 3.7 218 0.88 5.0 4.39 un hydrophobic verylarge
##> 6 1.0 85 0.72 7.9 6.77 un neutral verysmall
##> polarity.cat
##> 1 nonpolar
##> 2 nonpolar
##> 3 polar
##> 4 polar
##> 5 nonpolar
##> 6 nonpolar

```

Ok, now let's do cluster analysis. Cluster analysis is an **exploratory data analysis** tool that let's you look for natural groupings in data. Its related to **machine learning**. Its common in genetics and community ecology, among other things. We can do a cluster analysis in R like this.

First, we need to assess how similar are amino acids are to each other in a way to take into account all 10 columns of our data. This can be done with a thing called a **multivariate distance matrix**. We won't worry about the details

right now. (If you want, you can try to think about points spread out in 3D space and measuring the distance between each one; now think about them spread out in 10-dimensional space and think about the distance between each one. Don't hurt yourself doing this).

```
aa_dist <- dist(aa_subset, method = "euclidean") # distance matrix
```

The `dist()` command just made a **distance matrix** for us. We can use it to see what R tells us about it

```
is(aa_dist)
```

```
[1] "dist"
```

This is a big matrix with lots of decimal places, so we can look at it most easily if we type `round(aa_dist)`. Type that code below

This might not fit well on your screen. If you have trouble try this (sorry, not going to explain it)

```
round(as.matrix(aa_dist)[1:5,1:5])
```

```
A C D E F
A 0 46 81 112 147
C 46 0 53 75 102
D 81 53 0 39 101
E 112 75 39 0 71
F 147 102 101 71 0
```

Small numbers mean that, taking all 10 variables into account, two amino acids are similar to each other. For example, if you look at just the upper left hand part of the big grid A and C have a value of 46. Big numbers mean that, taking all 10 variables into account, two amino acids are very different. A and E have a value of 112. So A and C are closer together or more similar than A and E.

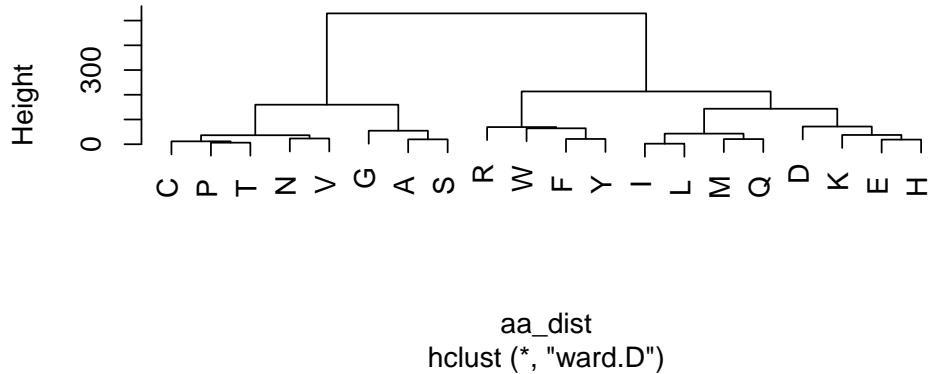
This matrix thingy is hard to interpret, so we can turn it into a graph. We'll use the `hclust()` function, which is a type of cluster analysis.

```
aa_cluster <- hclust(aa_dist, method="ward.D")
```

Now let's plot the results using R's basic `plot()` command

```
plot(aa_cluster)
```

### Cluster Dendrogram



As I noted above, according to expert biochemists, I and L are similar to each other, and Y and W are similar, but I and Y are not so similar. Could R's cluster algorithm figure this out?

To use phylogeny terms, I and L appear as **Sisters** on the tree. Cool, that worked. However, Y and F are sisters, and W is one step removed. Refer back to the Venn Diagram. Are F and Y considered similar by the experts?

On the original Venn Diagram there are 3 large circles, and several smaller ones. What are the labels of the circles? (Answer in TopHat)

On the Venn Diagram the following amino acids are considered hydrophobic (note that this isn't exactly the same as our data)

C T V G A I L M F Y W H K

There's some overlap with the left most group, but its far from perfect. So our branch diagram has 3 major groupings, and the experts have 3 major groups, but they don't match up perfectly. There could be many reasons for this, but it could be that we're considering different data than what the experts considered important, or that we used a computer algorithm that actually knows nothing about biochemistry!

## 32.10 Notes for Further study

The following notes may be added to this chapter in the future; ignore them unless you are super curious about clustering algorithms

### 32.10.1 UPGMA

Wikipedia has a step by step introduction to UPGMA: <https://en.wikipedia.org/wiki/UPGMA>

```

http://www.nmsr.org/upgma.htm

aa_cluster<- hclust(aa_dist, method="ward.D")

b <- sim2dist(BLOSUM45[1:20,1:20], maxSim = max(BLOSUM45[1:20,1:20]))
b_dist <- dist(b, method = "euclidean") # distance matrix
par(mfrow = c(1,2))

plot(hclust(b_dist, method="ward.D"))
plot(hclust(aa_dist, method="ward.D"))

BLOSUM45

```

### 32.10.2 Neighbor joining

[https://en.wikipedia.org/wiki/Neighbor\\_joining](https://en.wikipedia.org/wiki/Neighbor_joining)

original publication has examples

other examples online

### 32.10.3 Ward's method

Based on changes in variance; least squares?

[https://en.wikipedia.org/wiki/Ward%27s\\_method](https://en.wikipedia.org/wiki/Ward%27s_method)

### 32.10.4 hclust algorithms

hclust can use the follwing methods to build the tree

- “ward.D”, “ward.D2”
- “single”: “closely related to the minimal spanning tree”
- “complete”
- “average” (= UPGMA), which is a method previous used for building phylogenetic trees
- “mcquitty” (= WPGMA), “median” (= WPGMC)
- “centroid” (= UPGMC).

From ?hclust “This function performs a hierarchical cluster analysis using a set of dissimilarities for the n objects being clustered. Initially, each object is assigned to its own cluster and then the algorithm proceeds iteratively, at each stage joining the two most similar clusters, continuing until there is just a single cluster. At each stage distances between clusters are recomputed by the Lance–Williams dissimilarity update formula according to the particular clustering method being used.”



# Chapter 33

## ggpubr

```
library(compbio4all)
```

### 33.1 Vocab

- wrapper
- ggplot2
- ggpibr
- line of best fit
- $y = B_0 + B_1 \cdot x$
- residual
- data ellipse
- correlation coefficient

### 33.2 Learning objectives

- Know what a wrapper is
- Know the relationship between ggplot2 and ggpibr
- Be able to run code that makes graphs with ggpibr
- Know how color and shape can make a plot easier to interpret

### 33.3 Introduction

ggplot2 is one of the most well-known and widely used R packages. It is arguably the most powerful and flexible package for creating plots in R.

ggplot2 has a very steep learning curve. ggpibr is a package that creates “wrappers” for much of ggplot2’s core functionality. A **wrapper** is a function that

runs another function for you, often making its use easier while also making certain decisions for you by setting certain defaults.

ggpubr is really cool, but unfortunately its syntax is different from both base R plotting and ggplot2. I will provide you ggpublisher code whenever we need it; feel free to experiment, but you'll need to tinker with it or read the help file.

This Software Check point will have you do the following things to get you set up to use these packages

- Download ggplot2 and ggpublisher
- Install a data set of allometric data similar to the mammals data of MASS
- Make some basic plots

## 33.4 Preliminaries

### 33.4.1 Download packaged

Only do this once, then comment out of the script.

```
install.packages("ggplot2")
iinstall.packages("ggpubr")
```

### 33.4.2 Load the libraries

```
library(ggplot2)
library(ggpublisher)
```

### 33.4.3 Load the msleep package

The mammals dataset is a classic dataset in the MASS package. msleep is an updated version of the data that includes more numeric data (e.g. hours of sleep) and categorical data (e.g. if a species is endangered)

```
data(msleep)
```

## 33.5 Make a basic ggpublisher plot

Let's make a boxplot in ggpublisher. ggpublisher has hand functions like ggboxplot, gghistogram, and ggscatter.

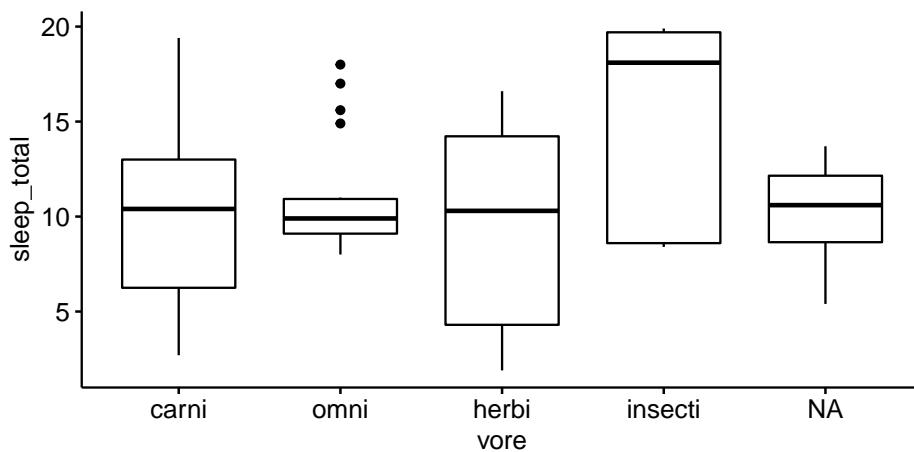
### 33.5.1 ggpublisher syntax

ggpubr does NOT use formula notation like base R function. You have to explicitly define a y and an x variable. Additionally, the variables MUST be in quotes.

Let's make a boxplot of the amount of sleep an organism gets (`sleep_total`) and what it eats (`vore`).

Note: Ignore any errors; these are due to NAs in the data.

```
ggboxplot(y = "sleep_total",
 x = "vore",
 data = msleep)
```

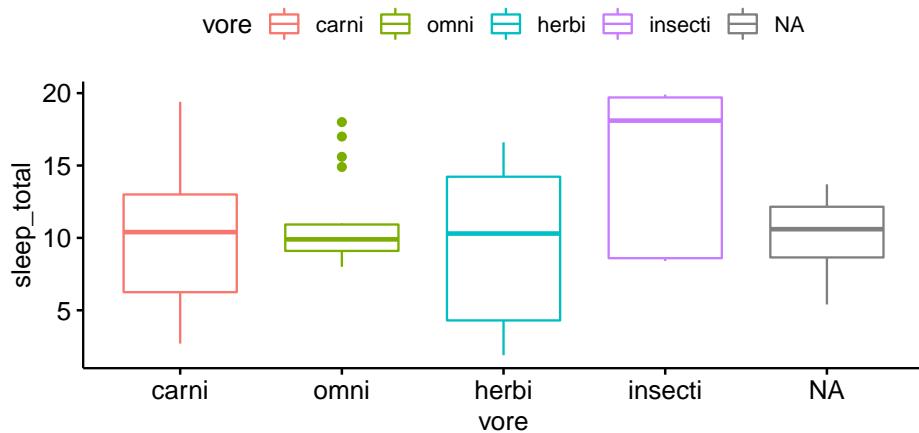


### 33.5.2 Color-coding data

The x-axis is “vore” and is labeled. A general principle of data visualization is that its always good to vary color, size and shape whenever possible, even if its redundant. This helps reinforce the groupings or patterns in the data.

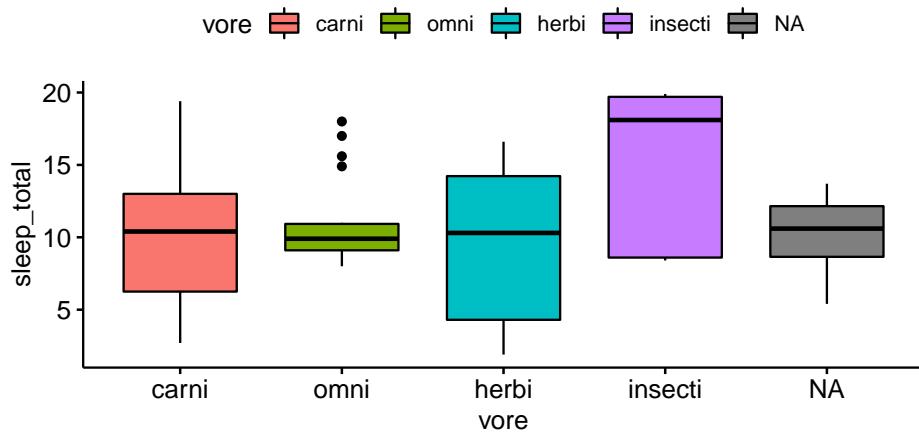
Change the color of the lines of the boxes:

```
ggboxplot(y = "sleep_total",
 x = "vore",
 color = "vore",
 data = msleep)
```



Change the fill inside the boxes:

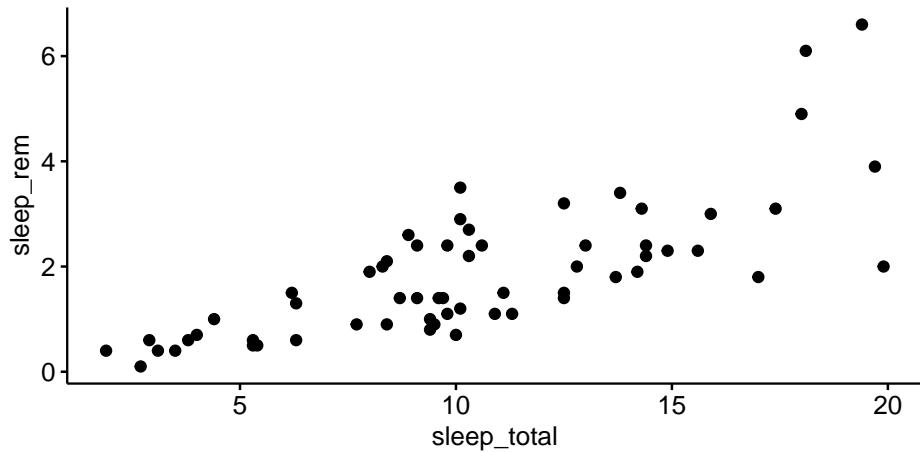
```
ggboxplot(y = "sleep_total",
 x = "vore",
 fill = "vore",
 data = msleep)
```



### 33.6 Scatter plots in ggpubr

Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 data = msleep)
```



### 33.7 Coloring by a categorical variable

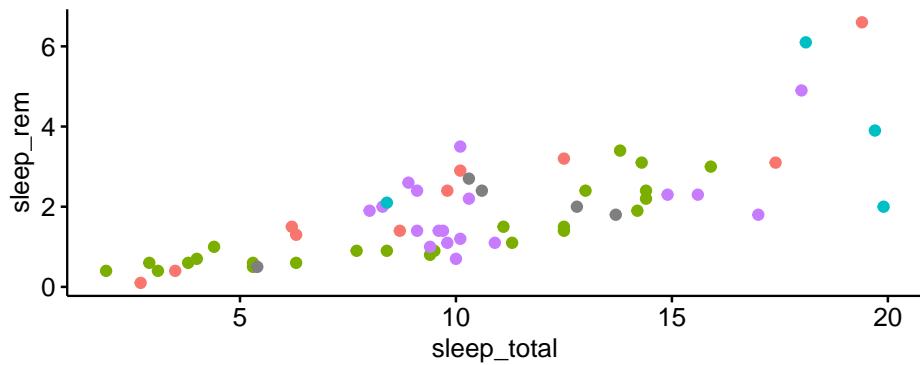
Note that almost everything goes in quotes

y = "sleep\_rem", x = "sleep\_total", color = "vore",

Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 color = "vore",
 data = msleep)
```

vore ● carni ● herbi ● insecti ● omni ● NA



### 33.8 Coloring by a continuous numeric variable

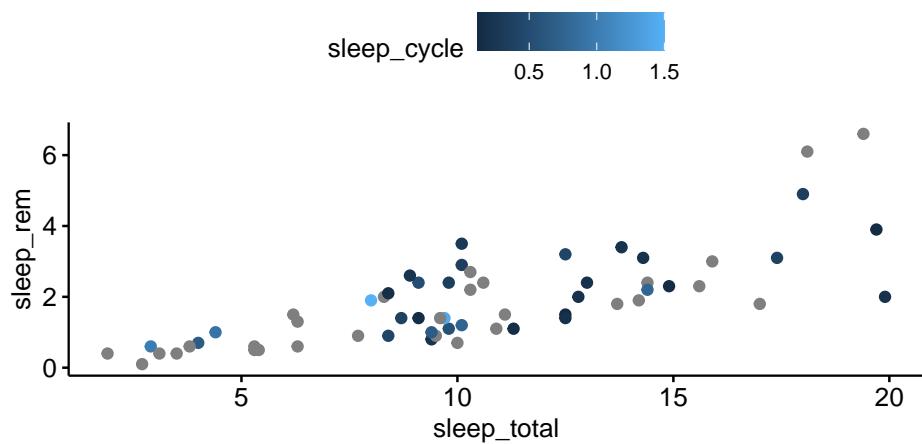
A third dimension can be added by color-coding the scatterplot.

Note that almost everything goes in quotes

```
y = "sleep_rem", x = "sleep_total", color = "sleep_cycle",
```

Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 color = "sleep_cycle",
 data = msleep)
```



### 33.9 Adding a line of best fit

The line has the general form

$$y = m \cdot x + b$$

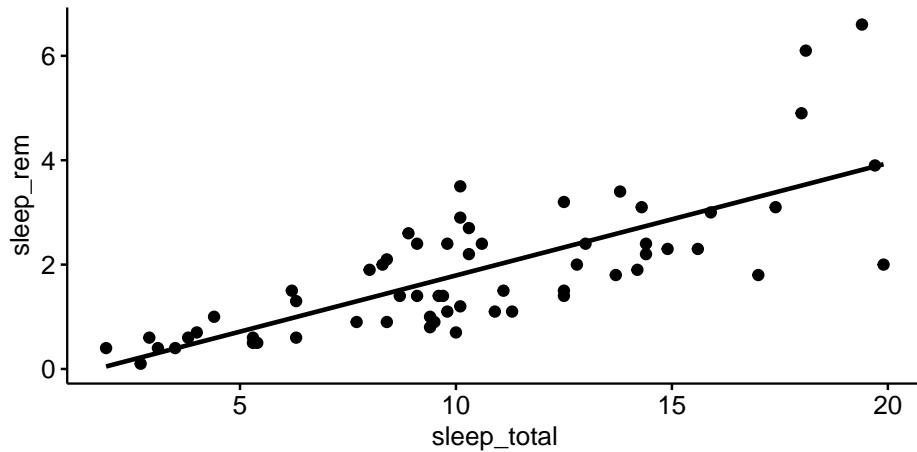
Which stats folks write as

$$y = B_0 + B_1 \cdot x$$

The distance from the line to each data point is called the **residual**.

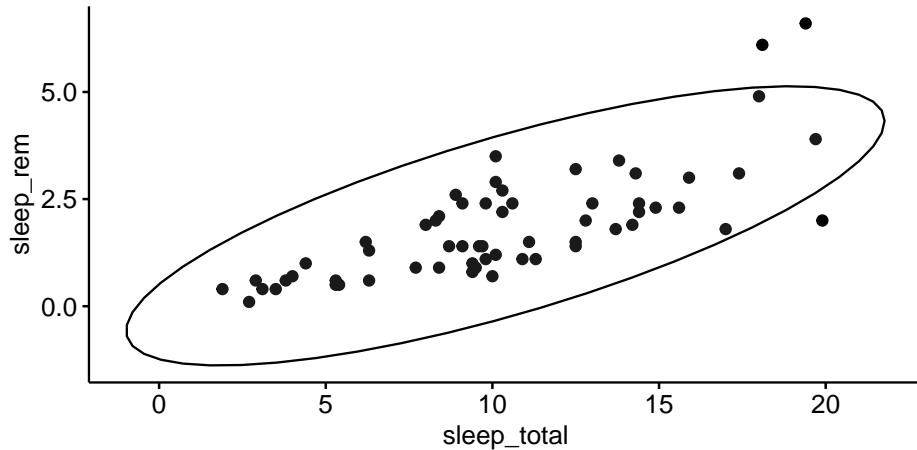
Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 add = "reg.line", # line of best fit
 data = msleep)
```



### 33.10 Adding a data ellipse

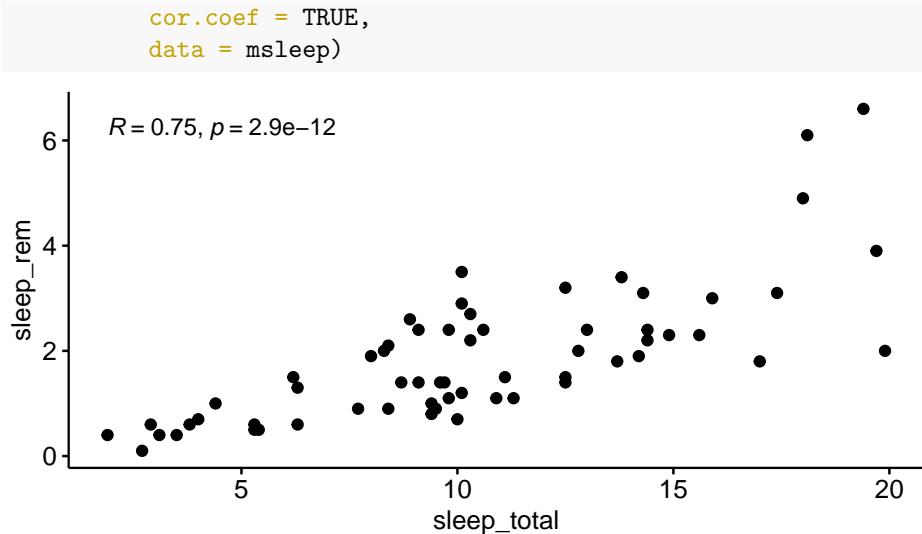
```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 ellipse = TRUE, # data ellipse
 data = msleep)
```



### 33.11 Add a correlation coefficient

By adding "cor.coef = TRUE" correlation coefficient, as well as a p-value for the significance of the correlation coefficient (testing the hypothesis that it is 0).

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
```



### 33.12 TASK

Make a scatter plot with the elements indicate below an upload the image (not the code) to the assignment found here: <https://canvas.pitt.edu/courses/45284/assignments/460717>

The figure should have these elements:

- sleep\_cycle on the y axis
- sleep\_total on the x axis
- a line of best fit
- a correlation coefficient
- a data ellipse

If this doesn't work, check that everything is in quotes and that there is a comma at the end of each line. If it doesn't work, email your code to your UTA and CC me, and/or come to office hours.

```
put your code below
```

# Chapter 34

## gpubr - allometric data

Allometric data - classic case of regression, using logs, using non-linear model too

```
library(compbio4all)
```

### 34.1 Vocab

- wrapper
- ggplot2
- ggpibr
- \$ operator
- smoother
- continuous data
- categorical data

### 34.2 Learning objectives

- Know what a wrapper is
- Know the relationship between ggplot2 and ggpibr
- Be able to run code that makes graphs with ggpibr
- Know how color and shape can make a plot easier to interpret
- Know how to use the \$ operator to make new columns

### 34.3 Introduction

This Software Check point will have you practice using ggpibr.

## 34.4 Preliminaries

### 34.4.1 Download packaged

Only do this once, then comment out of the script. You probably already did this in the previous Code Checkpoint.

```
install.packages("ggplot2")
iinstall.packages("ggpubr")
```

### 34.4.2 Load the libraries

```
library(ggplot2)
library(ggpubr)
```

### 34.4.3 Load the msleep package

The mammals dataset is a classic dataset in the MASS package. msleep is an updated version of the data that includes more numeric data (e.g. hours of sleep) and categorical data (e.g. if a species is endangered)

```
data(msleep)
```

## 34.5 Make a basic ggpubr scatterplot

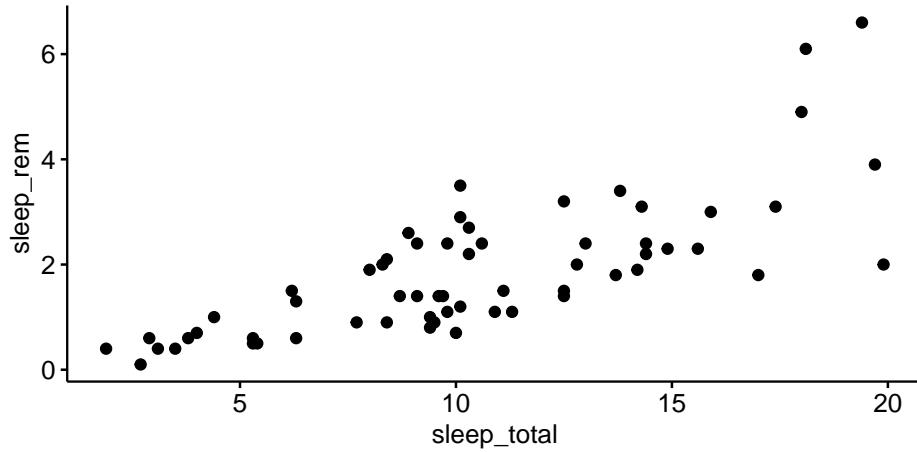
### 34.5.1 ggpubr syntax

ggpubr does NOT use formula notation like base R function. You have to explicitly define a y and an x variable. Additionally, the variables MUST be in quotes.

## 34.6 Scatter plots in ggpubr

Ignore any errors.

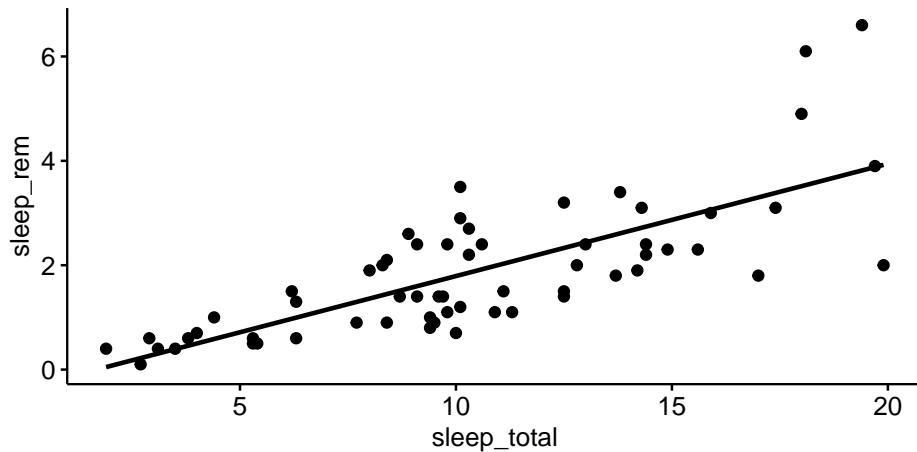
```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 data = msleep)
```



### 34.7 Adding a line of best fit

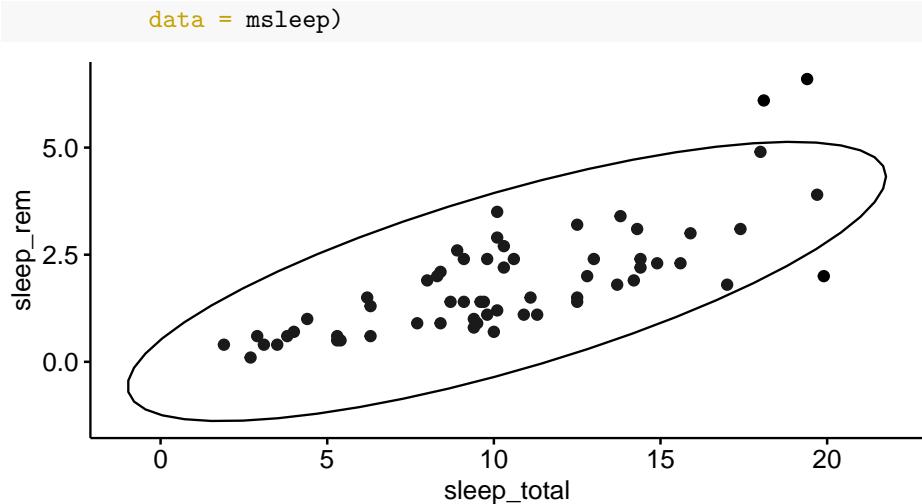
Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 add = "reg.line", # line of best fit
 data = msleep)
```



### 34.8 Adding a data ellipse

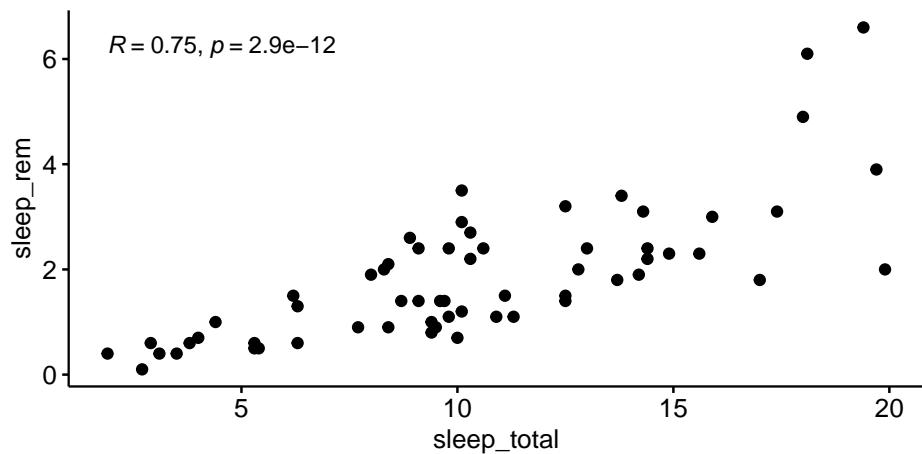
```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 ellipse = TRUE, # data ellipse
```



### 34.9 Add a correlation coefficient

By adding "cor.coef = TRUE" correlation coefficient, as well as a p-value for the significance of the correlation coefficient (testing the hypothesis that it is 0).

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 cor.coef = TRUE,
 data = msleep)
```



#### 34.9.1 Allometry

NOTE: A question about the basic biological issues related to allometry could appear on the next test.

The mammal and msleep data are often used to display the concept of **allometric relationships**. “Allometry, in its broadest sense, describes how the characteristics of living creatures change with size” (Shingleton 2010). Ecologists and evolutionary biologists are often interested how different morphological, physiological, ecological, and life history factors vary with size. For example, larger organism typically have smaller brains, few offspring, and live longer, and these relationships are often linear when plotted on a log-log scale.

Allometry is not an inherently computational discipline, though it can involve a lot of math. One area of interest to ecologists is how things like metabolic rate and energy consumption vary as organisms increase in size.

Allometry research is often used to inform computational research, especially simulation models. For example, allometric models can be used to predict the size and growth patterns of trees in models that simulate the growth of forests.

### 34.9.2 Allometry and log scales

Taking the natural log of data is often used to re-scale it or make relationships linear. Allometric data is often plotted on a log-log scale: both the x and the y variables are logged.

To do this, we’ll make new columns. Let’s take the log of brain weight (brainwt) and body weight (bodywt)

First, brain weight. We can make a new column using the \$ operator. This operator can be used to select a single column, like this

```
mean(msleep$brainwt, na.rm = T)
```

```
[1] 0.2815814
```

It can also be used to *create* a new variable in a dataframe; here, I make a new column “brainwt\_log” that does not yet exist in the msleep dataframe.

```
msleep$brainwt_log <- log(msleep$brainwt)
```

The same thing for bodywt

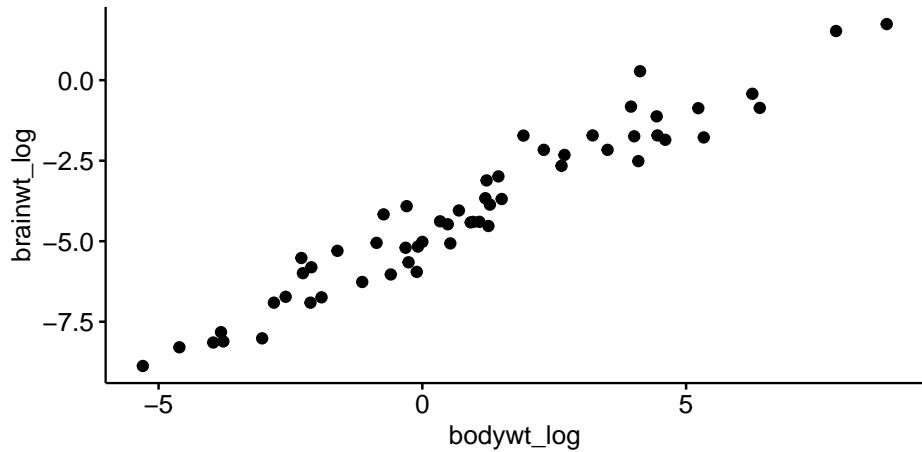
```
msleep$bodywt_log <- log(msleep$bodywt)
```

## 34.10 Make an allometric plot

ggpubr does NOT use formula notation like base R function. You have to explicitly define a y and an x variable. Additionally, the variables MUST be in quotes.

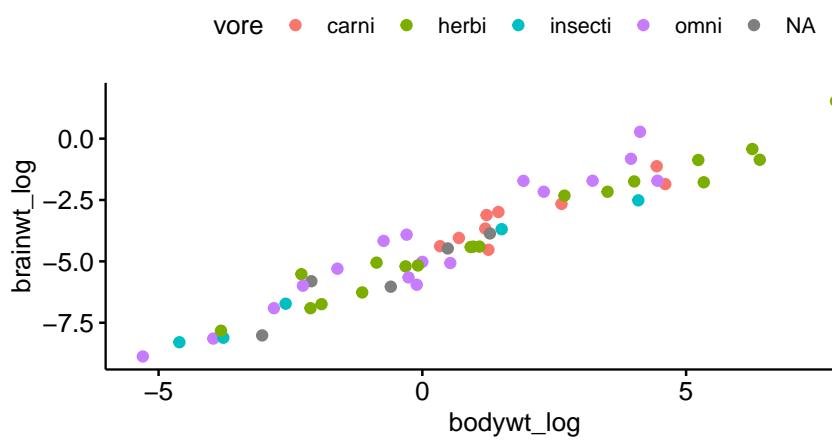
Note: Ignore any errors; these are due to NAs in the data.

```
ggscatter(y = "brainwt_log",
 x = "bodywt_log",
 data = msleep)
```

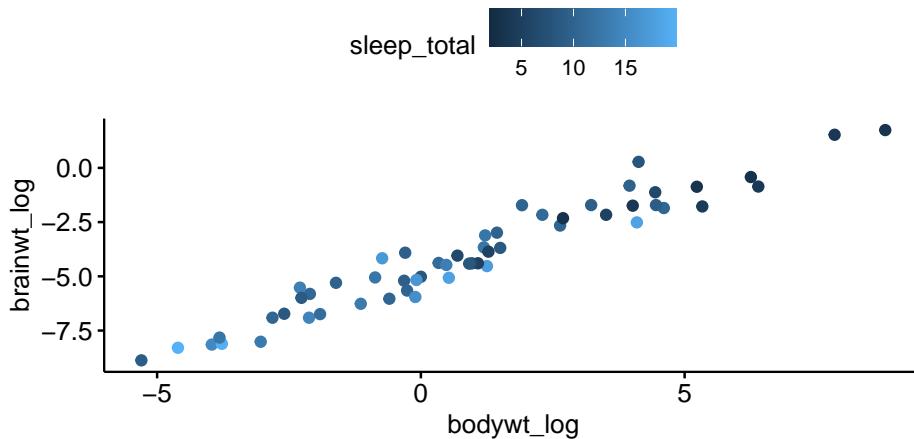


### 34.11 Change color by a categorical variable

```
ggscatter(y = "brainwt_log",
 x = "bodywt_log",
 color = "vore", # color =
 data = msleep)
```



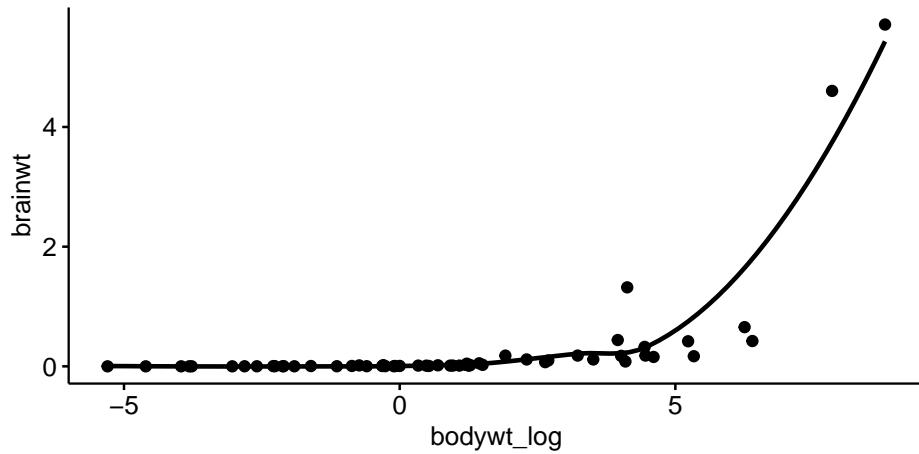
```
ggscatter(y = "brainwt_log",
 x = "bodywt_log",
 color = "sleep_total",
 data = msleep)
```



### 34.13 Add a smoother

A **smoother** is a data exploration tool which helps you visualize trends in the data. There are many ways to calculate them, but conceptually they work by taking doing something akin to taking a weight average of sets of adjacent points. A simple type is a **loess** smoother, which can easily be added in ggpubr.

```
ggscatter(y = "brainwt",
 x = "bodywt_log",
 add = "loess",
 data = msleep)
```

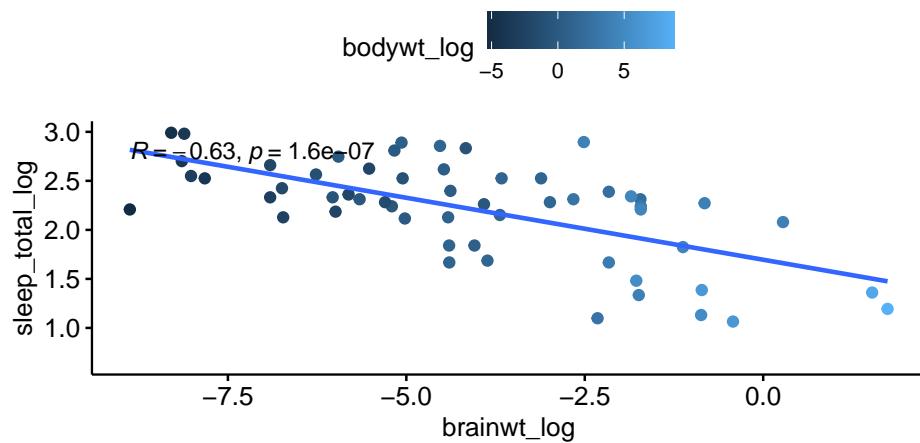


### 34.14 Task

- Create a new column `sleep_total_log`
- plot `sleep_total_log` on the y axis
- plot `brainwt_log` on the x axis
- add a line of best fit
- add a correlation coefficient
- color-code the data by `bodywt_log`

If you have a problem, make sure that things are in quotes as appropriate, and that there is a comma at the end of each line as needed. Note that `cor.coef = TRUE` doesn't use quotes.

```
msleep$sleep_total_log <- log(msleep$sleep_total)
ggscatter(y = "sleep_total_log",
 x = "brainwt_log",
 add = "reg.line",
 cor.coef = TRUE,
 color = "bodywt_log",
 data = msleep)
```





# Chapter 35

## Improving plots

```
library(ggplot2)
library(ggpubr)
library(cowplot)
data(msleep)
msleep$sleep_total_log <- log10(msleep$sleep_total)
msleep$bodywt_log <- log10(msleep$bodywt)

lm.out <- coef(lm(msleep$sleep_total_log ~ msleep$bodywt_log))

p1 <- ggscatter(y = "sleep_total_log",
 x = "bodywt_log",
 data = msleep,
 #size = 2,
 xlab = "Animal body weight\n(log scale)\n",
 ylab = "Animal brain size\n(log scale)",
 #rug = TRUE,
 #add = "reg.line",
 #cor.coef = T,
 #cor.coef.coord = c(-1.9,0.3),
 #cor.coef.size = 7
)

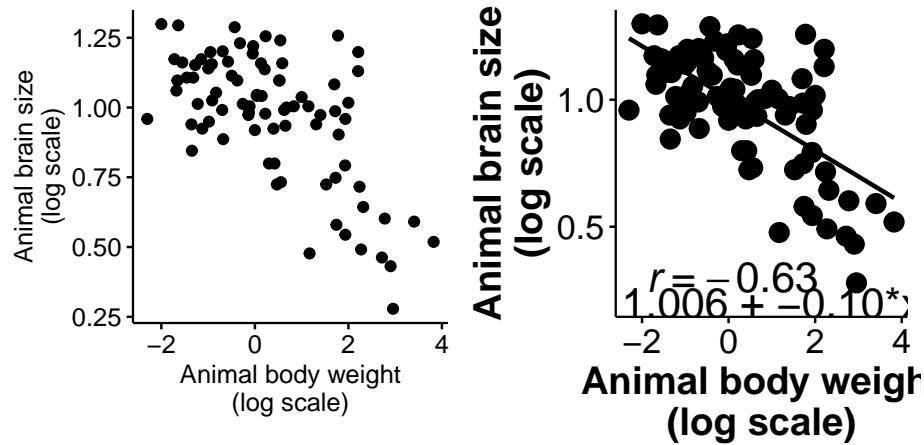
p2 <- ggscatter(y = "sleep_total_log",
 x = "bodywt_log",
 data = msleep,
 cor.coeff.args = list(aes(label = ..r.label..),
 cor.coef.name = "r"),
 size = 4,
```

```

xlab = "Animal body weight\n(log scale)",
ylab = "Animal brain size\n(log scale)",
#rug = TRUE,
add = "reg.line",
cor.coef = T,
cor.coef.coord = c(-1.9,0.3),
cor.coef.size = 7
) +
theme(axis.text=element_text(size=16),
axis.title=element_text(size=18,face="bold")) +
annotate("text",
x = 0.2,
y = 0.2,
size = 7,
label = "y = 1.006 + -0.10*x")

```

```
plot_grid(p1,p2)
```



```
library(compbio4all)
```

### 35.1 Vocab

- wrapper
- ggplot2
- ggpubr
- line of best fit
- $y = B_0 + B_1 \cdot x$

- residual
- data ellipse
- correlation coefficient

## 35.2 Learning objectives

- Know what a wrapper is
- Know the relationship between ggplot2 and ggpibr
- Be able to run code that makes graphs with ggpibr
- Know how color and shape can make a plot easier to interpret

## 35.3 Introduction

ggplot2 is one of the most well-known and widely used R packages. It is arguably the most powerful and flexible package for creating plots in R.

ggplot2 has a very steep learning curve. ggpibr is a package that creates “wrappers” for much of ggplot2’s core functionality. A **wrapper** is a function that runs another function for you, often making its use easier while also making certain decisions for you by setting certain defaults.

ggpibr is really cool, but unfortunately its syntax is different from both base R plotting and ggplot2. I will provide you ggpibr code whenever we need it; feel free to experiment, but you’ll need to tinker with it or read the help file.

This Software Check point will have you do the following things to get you set up to use these packages

- Download ggplot2 and ggpibr
- Install a data set of allometric data similar to the mammals data of MASS
- Make some basic plots

## 35.4 Preliminaries

### 35.4.1 Download packaged

Only do this once, then comment out of the script.

```
install.packages("ggplot2")
iinstall.packages("ggpibr")
```

### 35.4.2 Load the libraries

```
library(ggplot2)
library(ggpibr)
```

### 35.4.3 Load the msleep package

The mammals dataset is a classic dataset in the MASS package. msleep is an updated version of the data that includes more numeric data (e.g. hours of sleep) and categorical data (e.g. if a species is endangered)

```
data(msleep)
```

## 35.5 Make a basic ggpubr plot

Let's make a boxplot in ggpubr. ggpubr has hand functions like ggboxplot, gghistogram, and ggscatter.

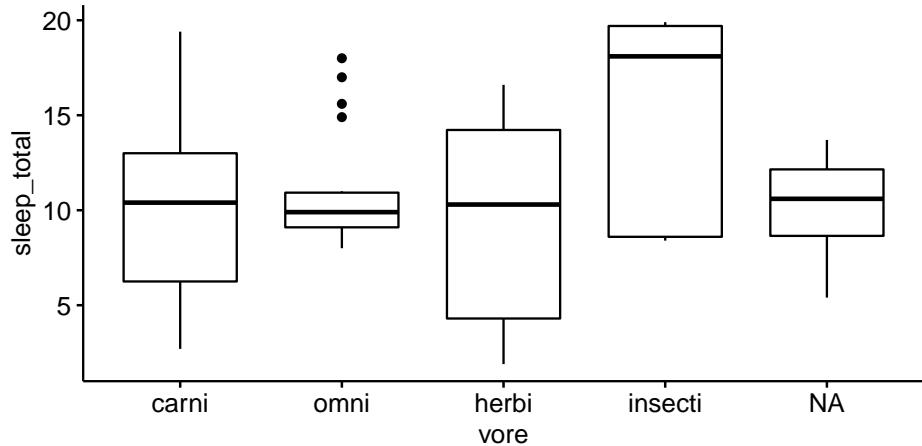
### 35.5.1 ggpubr syntax

ggpubr does NOT use formula notation like base R function. You have to explicitly define a y and an x variable. Additionally, the variables MUST be in quotes.

Let's make a boxplot of the amount of sleep an organism gets (sleep\_total) and what it eats (vore).

Note: Ignore any errors; these are due to NAs in the data.

```
ggboxplot(y = "sleep_total",
 x = "vore",
 data = msleep)
```

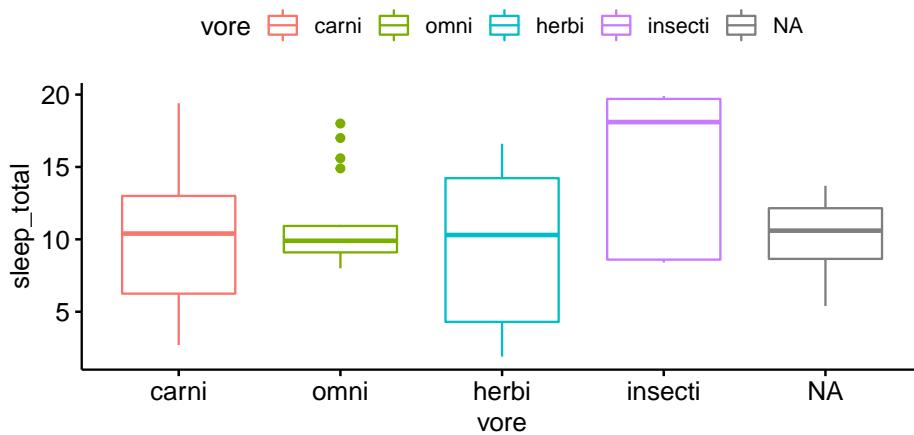


### 35.5.2 Color-coding data

The x-axis is “vore” and is labeled. A general principle of data visualization is that its always good to vary color, size and shape whenever possible, even if its redundant. This helps reinforce the groupings or patterns in the data.

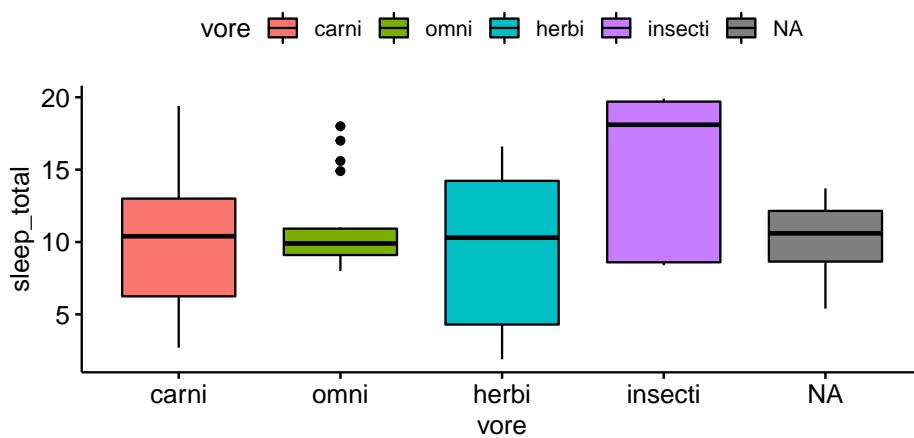
Change the color of the lines of the boxes:

```
ggboxplot(y = "sleep_total",
 x = "vore",
 color = "vore",
 data = msleep)
```



Change the fill inside the boxes:

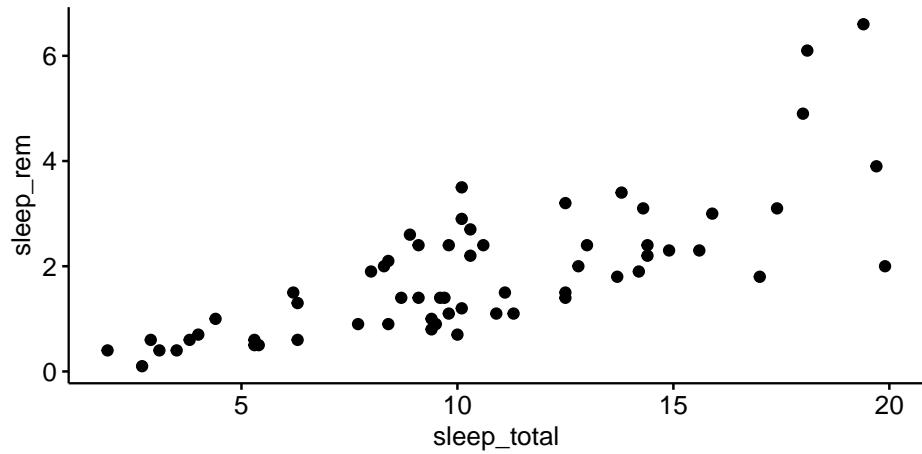
```
ggboxplot(y = "sleep_total",
 x = "vore",
 fill = "vore",
 data = msleep)
```



## 35.6 Scatter plots in ggpubr

Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 data = msleep)
```



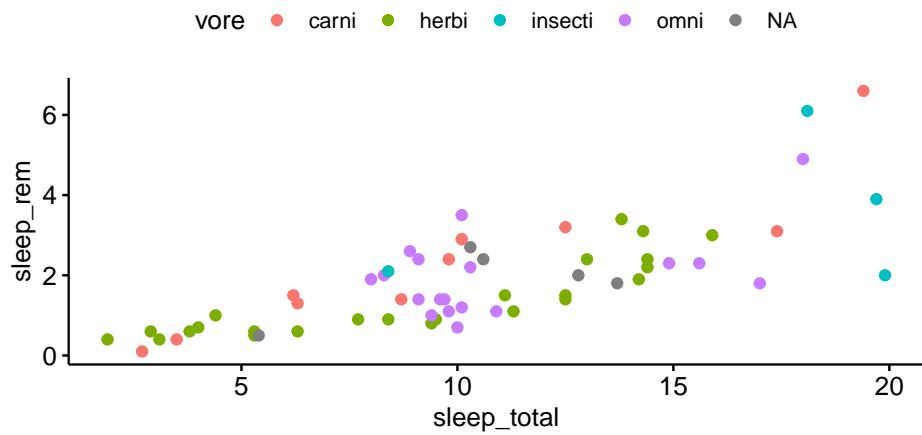
### 35.7 Coloring by a categorical variable

Note that almost everything goes in quotes

y = "sleep\_rem", x = "sleep\_total", color = "vore",

Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 color = "vore",
 data = msleep)
```



## 35.8 Coloring by a continuous numeric variable

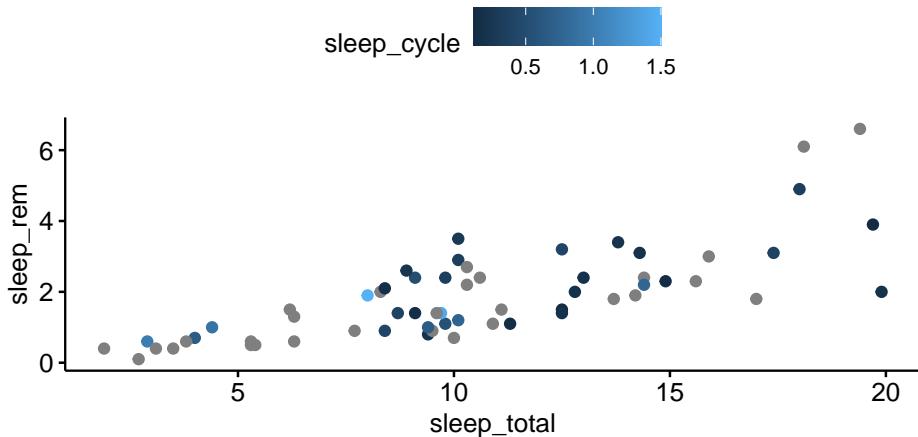
A third dimension can be added by color-coding the scatterplot.

Note that almost everything goes in quotes

```
y = "sleep_rem", x = "sleep_total", color = "sleep_cycle",
```

Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 color = "sleep_cycle",
 data = msleep)
```



## 35.9 Adding a line of best fit

The line has the general form

$$y = m \cdot x + b$$

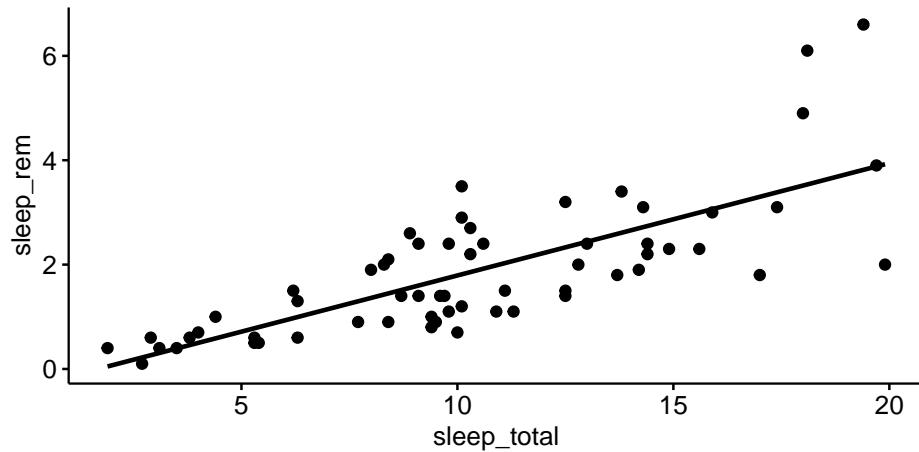
Which stats folks write as

$$y = B_0 + B_1 \cdot x$$

The distance from the line to each data point is called the **residual**.

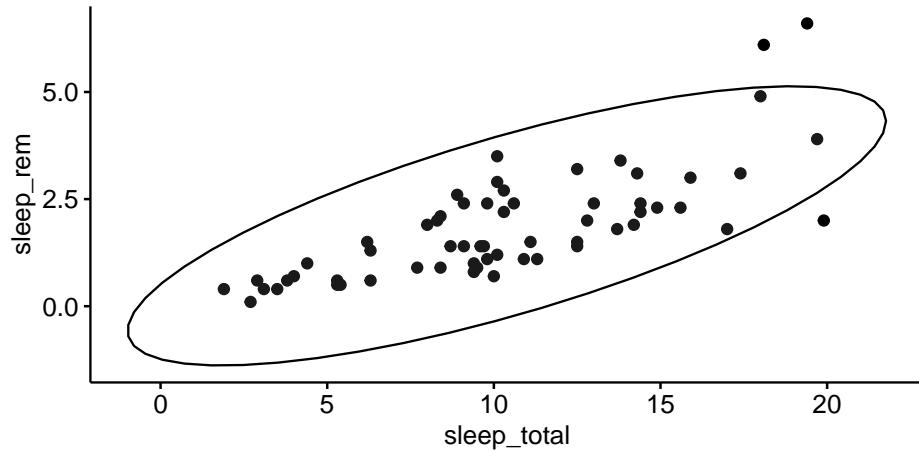
Ignore any errors.

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 add = "reg.line", # line of best fit
 data = msleep)
```



### 35.10 Adding a data ellipse

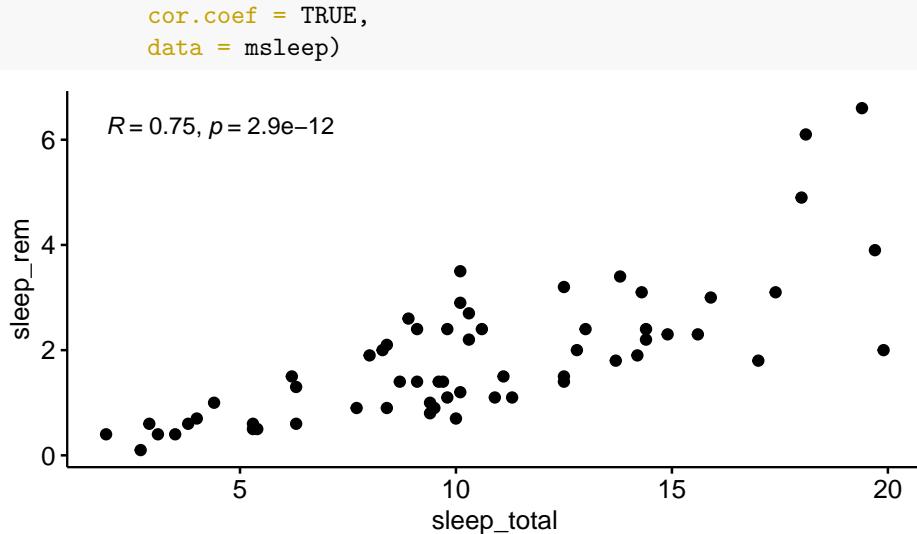
```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
 ellipse = TRUE, # data ellipse
 data = msleep)
```



### 35.11 Add a correlation coefficient

By adding "cor.coef = TRUE" correlation coefficient, as well as a p-value for the significance of the correlation coefficient (testing the hypothesis that it is 0).

```
ggscatter(y = "sleep_rem",
 x = "sleep_total",
```



## 35.12 TASK

Make a scatter plot with the elements indicate below an upload the image (not the code) to the assignment found here: <https://canvas.pitt.edu/courses/45284/assignments/460717>

The figure should have these elements:

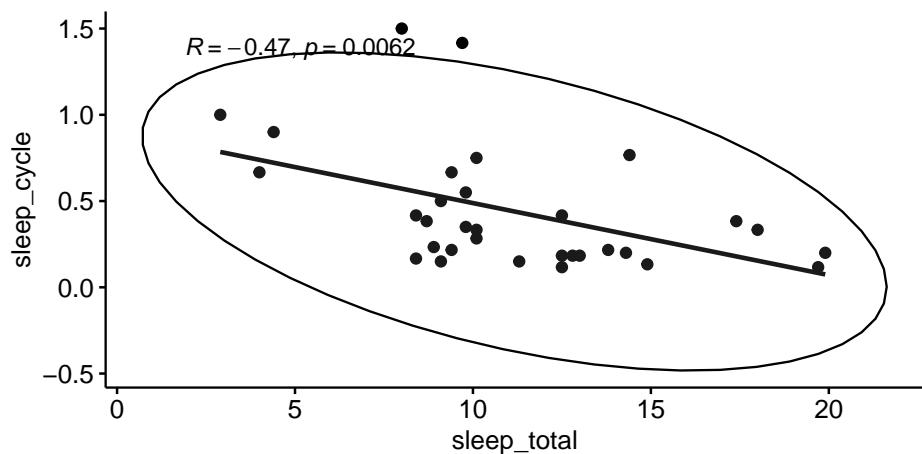
- sleep\_cycle on the y axis
- sleep\_total on the x axis
- a line of best fit
- a correlation coefficient
- a data ellipse

If this doesn't work, check that everything is in quotes and that there is a comma at the end of each line. If it doesn't work, email your code to your UTA and CC me, and/or come to office hours.

```
put your code below
```

### 35.12.1 Key

```
ggscatter(y = "sleep_cycle",
 x = "sleep_total",
 cor.coef = TRUE,
 ellipse = TRUE,
 add = "reg.line",
 data = msleep)
```



# Chapter 36

## Scatter plots

```
library(compbio4all)
```

### 36.1 Checkpoint: Plot the data

Load the data and make a plot that matches Figure 2.8 on page 26 of the Higgs and Atwood chapter 2 reading

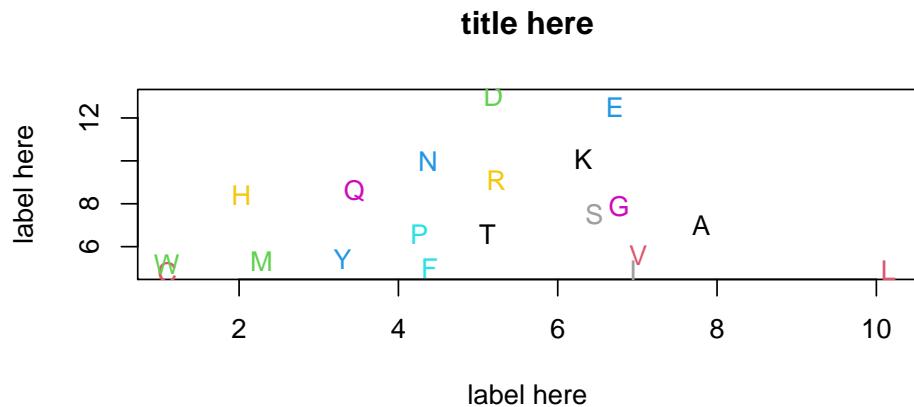
In order to make this plot, you will have to

- Read Chapter
- Examine Figure 2.8 carefully
- Examine Table 1 of Higgs 2009, especially the notes at the bottom of the table under the data themselves
- Use the basic R code below and change the variables being plotted
- Label the axes as in Chapter 2.
- Render your plot to RPubs, screen grab it, and submit it.
- Contact me AND your UTA if/when you run into trouble.
- The UTAs and I all have office hours on Monday

The plot() function is set up to make a blank plot. text() then uses the same tilda notation as plot() to plot the labels.

```
par(mfrow = c(1,1))
plot(polar.req ~ freq, data = aa_dat,
 xlab = "label here",
 ylab = "label here",
 main = "title here",
 col = 0)
text(polar.req ~ freq,
 labels = aa,
```

```
data = aa_dat,
col = 1:20)
```



# Chapter 37

## 3D scatterplots

```
library(compbio4all)
```

### 37.1 Key concepts / vocab

- scatterplot matrix
- correlation
- scatterplot3d
- plane of best fit
- color coding to help with visualization
- create line of best fit / plane of best fit with lm()

### 37.2 Preliminaries

#### 37.2.1 Download the package

Only do this once, then comment out the code. You may have already downloaded ggplot2 if you have done the other code checkpoint.

```
#scatterplot3d
#install.packages("scatterplot3d")

#ggplot2 (if needed)
#install.packages("ggplot2")
```

#### 37.2.2 Load packages

```
library(scatterplot3d)
library(ggplot2)
```

### 37.2.3 Load data

```
data("msleep")
```

## 37.3 3D scatterplots

The syntax for scatterplot3d is kind of clunky. I usually have to use some trial and error. I will provide you code when you need to use it.

### 37.3.1 Allometry and log scales

Taking the natural log of data is often used to re-scale it or make relationships linear. Allometric data is often plotted on a log-log scale: both the x and the y variables are logged.

To do this, we'll make new columns. Let's take the log of brain weight (brainwt) and body weight (bodywt)

First, brain weight. We can make a new column using the \$ operator. This operator can be used to select a single column, like this

```
mean(msleep$brainwt, na.rm = T)
```

```
[1] 0.2815814
```

It can also be used to *create* a new variable in a dataframe; here, I make a new column "brainwt\_log" that does not yet exist in the msleep dataframe.

```
msleep$brainwt_log <- log(msleep$brainwt)
```

The same thing for sleep\_rem\_log

```
msleep$sleep_rem_log <- log(msleep$sleep_rem)
```

### 37.3.2 Basic 3D scatterplot

Did I mention that the synatx is clunky?

We'll look at 3 variables.

First, a function we'll use

```
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)
{
 usr <- par("usr"); on.exit(par(usr))
 par(usr = c(0, 1, 0, 1))
 r <- abs(cor(x, y))
```

```

txt <- format(c(r, 0.123456789), digits = digits)[1]
txt <- paste0(prefix, txt)
if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
text(0.5, 0.5, txt, cex = cex.cor * r)
}

```

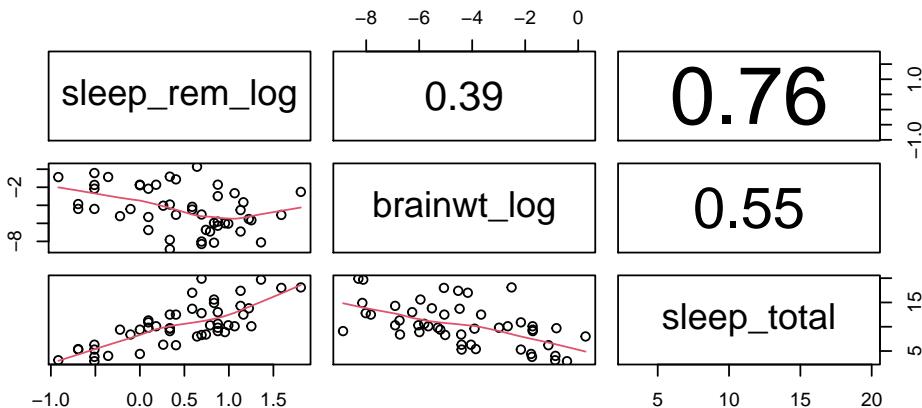
First a scatterplot matrix. I will give you this code whenever needed; you should be able to interpret it (Note: This code is more complex than usual because of NA values in the data; again, I'll give you the code as needed, you just need to interpret the output)

Also note that the absolute value of the correlation coefficients are being shown; negative correlations are not being indicated (both of the 0.59 values should be -0.59 to indicate the direction of the correlation)

```

plot(~sleep_rem_log+brainwt_log+sleep_total,
 data = na.omit(msleep[, c("sleep_rem_log","brainwt_log","sleep_total")]),
 panel = panel.smooth,
 upper.panel = panel.cor)

```



For an obscure reason, I need to turn the msleep object into a dataframe.

```
is(msleep)
```

```

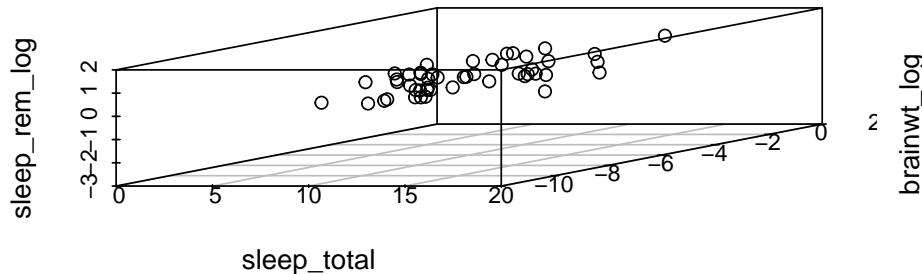
[1] "tbl_df" "tbl" "data.frame" "list" "oldClass"
[6] "vector"
msleep_df <- data.frame(msleep)

```

Now the actual 3D plot

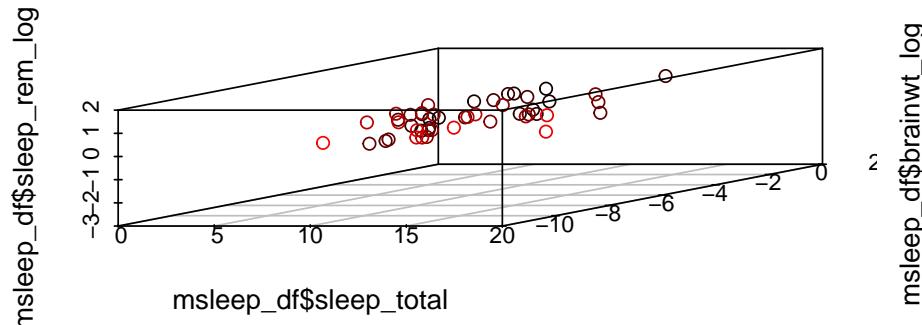
The variables are in the order x, y, z.

```
scatterplot3d(msleep_df[, c("sleep_total","brainwt_log","sleep_rem_log")])
```



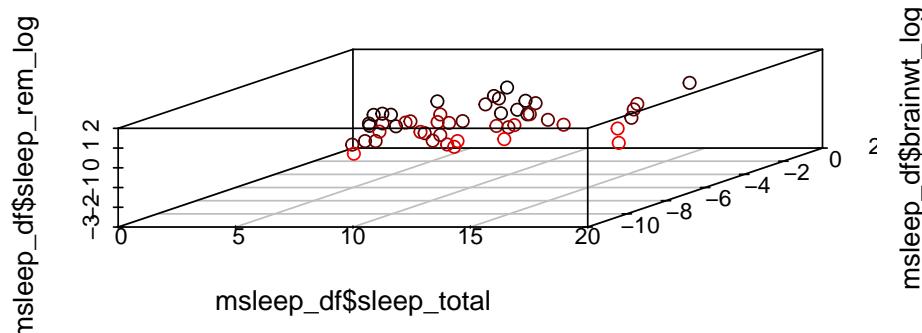
Color coding can be added to highlight the values of the z axis.

```
scatterplot3d(x = msleep_df$sleep_total,
 y = msleep_df$brainwt_log,
 z = msleep_df$sleep_rem_log,
 highlight.3d = TRUE)
```



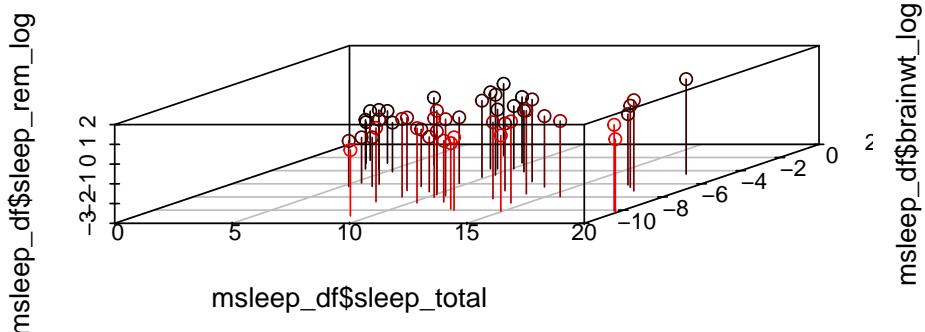
The angle can be varied to change the view. It can be hard to figure out what works best.

```
scatterplot3d(x = msleep_df$sleep_total,
 y = msleep_df$brainwt_log,
 z = msleep_df$sleep_rem_log,
 angle = 60,
 highlight.3d = TRUE)
```



A useful feature is to drop a line from each point done to the x-y plane to help visualize the z axis. (This is NOT a residual).

```
scatterplot3d(x = msleep_df$sleep_total,
 y = msleep_df$brainwt_log,
 z = msleep_df$sleep_rem_log,
 highlight.3d = TRUE,
 angle = 60,
 type="h")
```



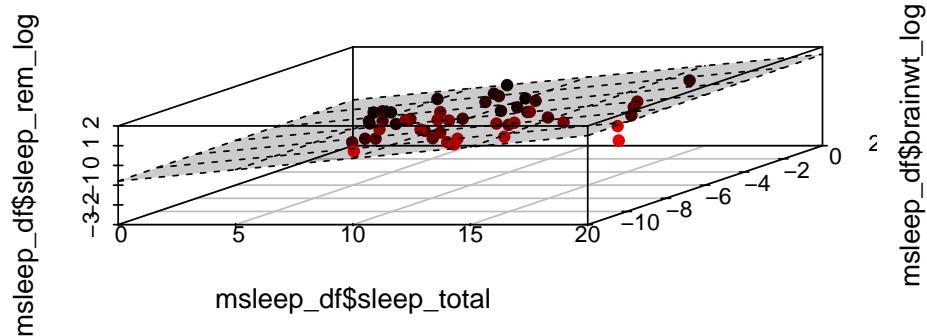
## 37.4 Plane of best fit

The function lm() is used to fit lines to data (or planes). You need to know that it does this, but I will write any code that is needed.

```
Create line of best fit using lm()
plane<-lm(sleep_rem_log ~ sleep_total + brainwt_log, data = msleep_df)

Do some calculations to calculate a "plane of best fit"
using the predict() function
predict(plane)

draw 3D scatter plot
s3d <- scatterplot3d(x = msleep_df$sleep_total,
 y = msleep_df$brainwt_log,
 z = msleep_df$sleep_rem_log,
 pch = 16,
 angle = 60,
 highlight.3d = TRUE,
)
s3d$plane3d(plane, draw_polygon = TRUE)
```



For more information of scatterplot3D

<https://rpubs.com/mase0501/320482>

## 37.5 From 3D plots to PCA

Conceptually, you can kind of think that PCA is making a 3D plot (if you gave it just 3 variables), figuring out an angle where the data is spread out the most, then flattening the data back down to 2D. The following code kind of evokes this. Don't worry about what the code is doing.

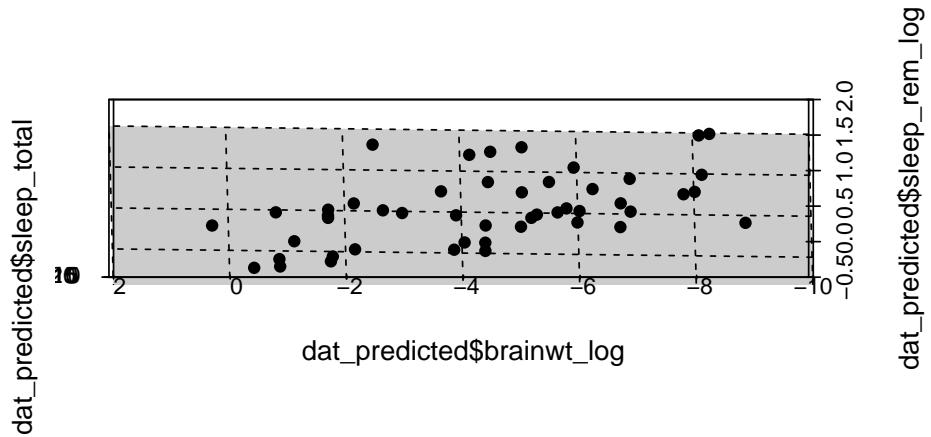
Some preparation

```
dat_predicted <- data.frame(sleep_total = plane$model$sleep_total,
 brainwt_log = plane$model$brainwt_log,
 sleep_rem_log = predict(plane))
```

Plotted basically as 2d

```
s3d_alt <- scatterplot3d(x = dat_predicted$sleep_total,
 y = dat_predicted$brainwt_log,
 z = dat_predicted$sleep_rem_log,
 pch = 16,
 angle = -1,
 scale.y = .01,
)

add plane
s3d_alt$plane3d(plane, draw_polygon = TRUE)
```



## 37.6 Task

Take the last code chunk and copy it below. Add the argument `highlight.3d = TRUE` and make the graph. Upload the graph to this assignment

<https://canvas.pitt.edu/courses/45284/assignments/460736>

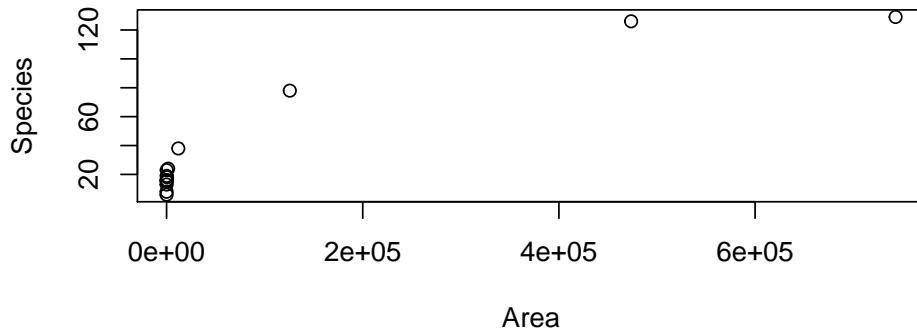


# Chapter 38

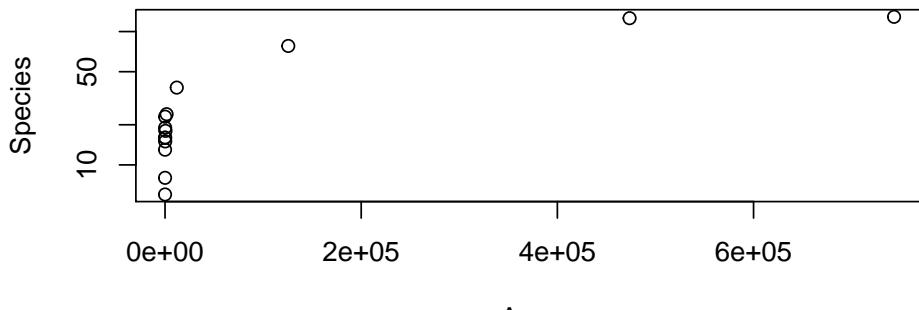
## Scatterplots and logs

```
library(Stat2Data)
data(SpeciesArea)

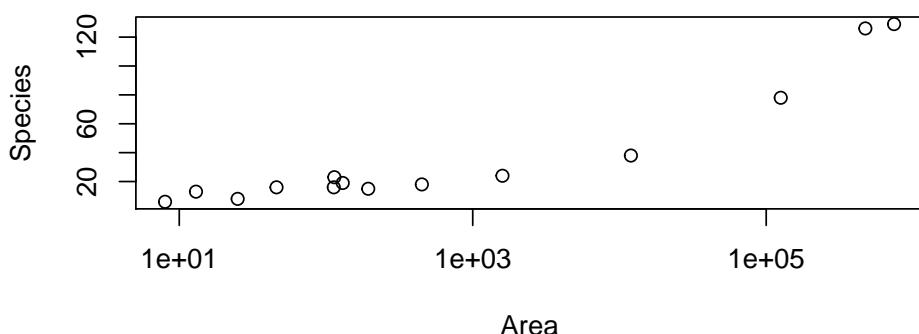
plot(Species ~ Area, data = SpeciesArea)
```



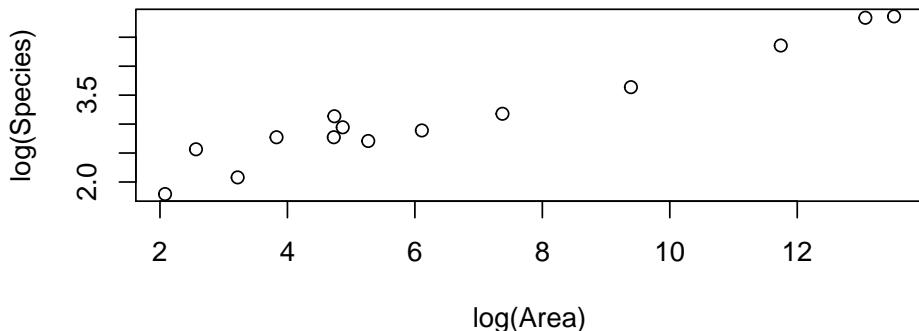
```
plot(Species ~ Area, data = SpeciesArea, log = "y")
```



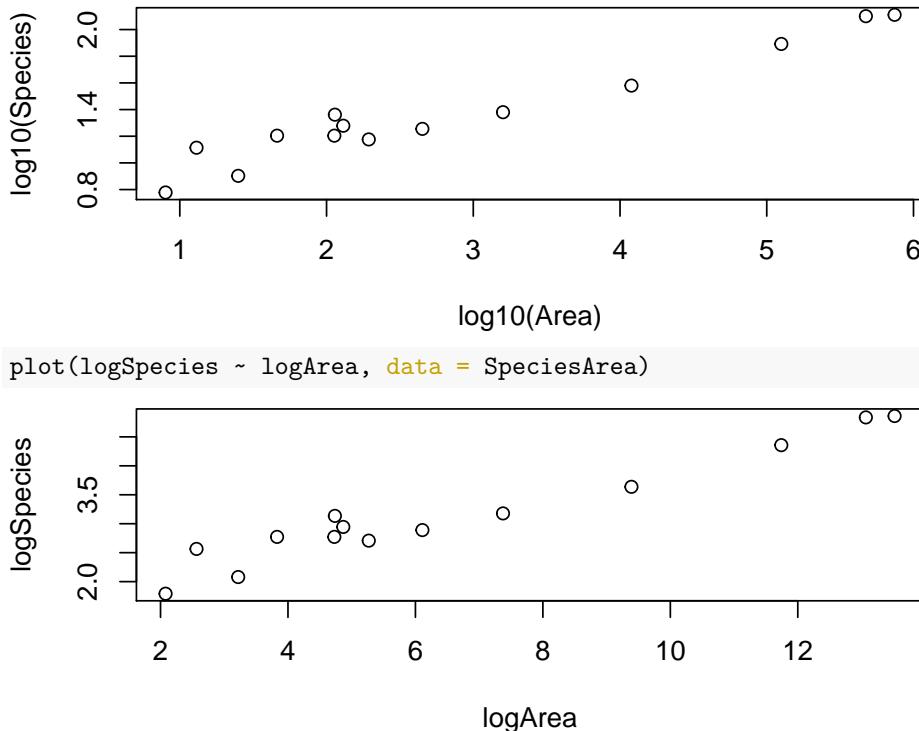
```
plot(Species ~ Area, data = SpeciesArea, log = "x")
```



```
plot(log(Species) ~ log(Area), data = SpeciesArea)
```



```
plot(log10(Species) ~ log10(Area), data = SpeciesArea)
```



Heaney, Lawrence R. (1984) "Mammalian species richness on islands on the Sunda Shelf, Southeast Asia," *Oecologia*, 61:11 17.



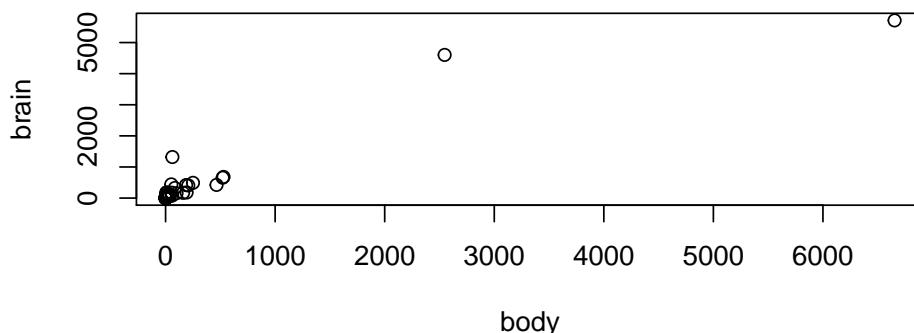
# Chapter 39

## Making fancy plots

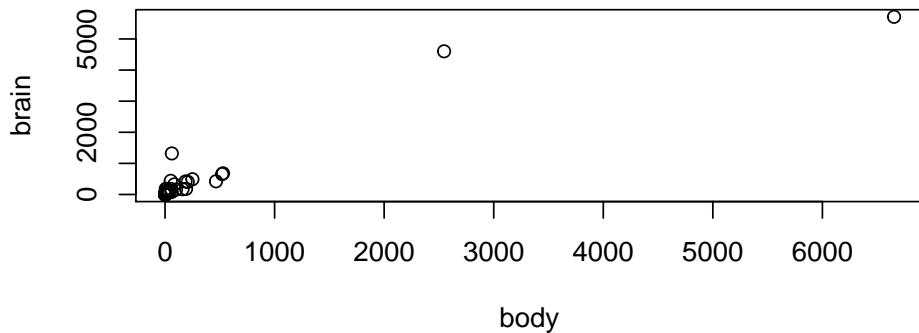
```
MASS Animals Brain and Body Weights for 28 Species # allometry
```

MASS mammals Brain and Body Weights for 62 Species of Land Mammals  
Selected from: Allison, T. and Cicchetti, D. V. (1976) Sleep in mammals: ecological and constitutional correlates. Science 194, 732–734.

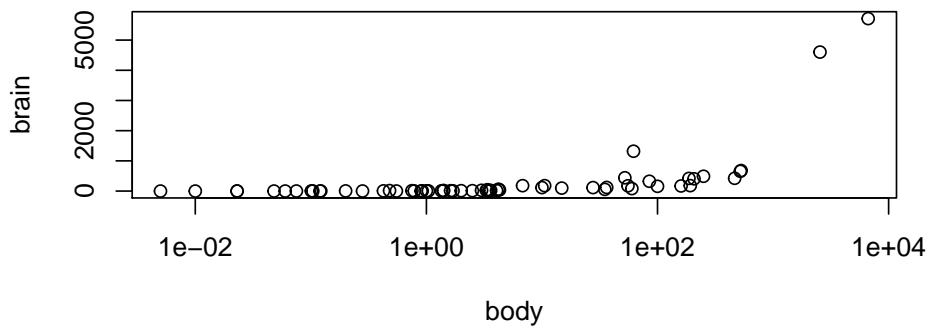
```
library(MASS)
data(mammals)
plot(mammals)
```



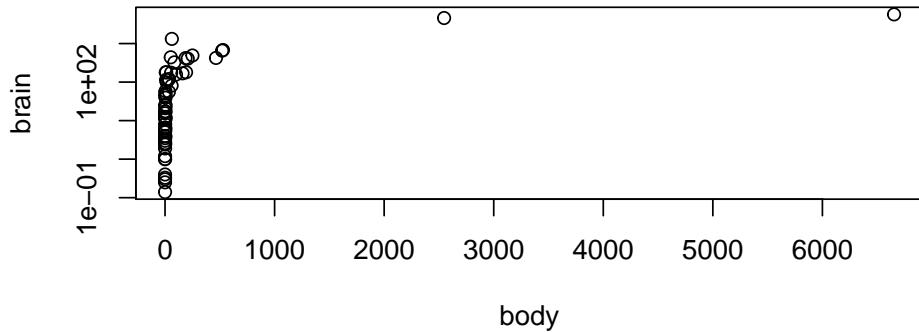
```
plot(brain ~ body, data = mammals)
```



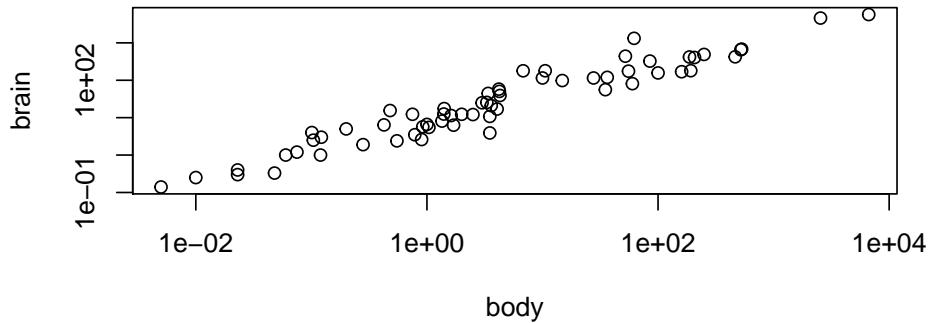
```
plot(brain ~ body, data = mammals, log = "x")
```



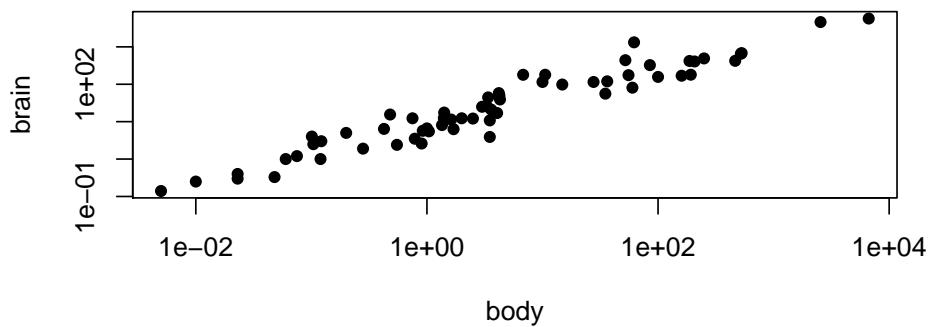
```
plot(brain ~ body, data = mammals, log = "y")
```



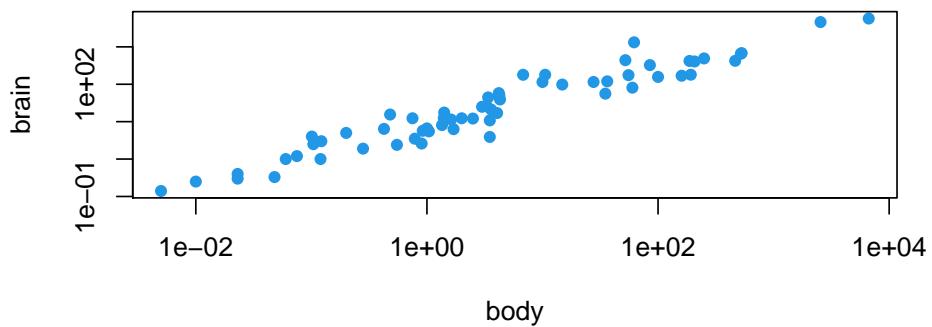
```
plot(brain ~ body, data = mammals, log = "xy")
```



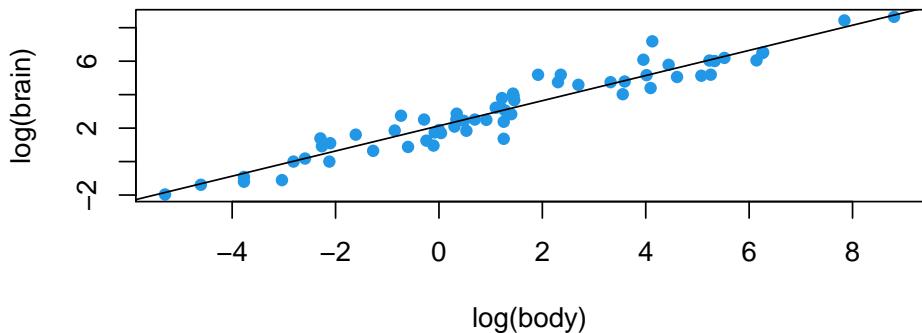
```
plot(brain ~ body, data = mammals, log = "xy", pch = 16)
```



```
plot(brain ~ body, data = mammals, log = "xy", pch = 16, col = 4)
```



```
brain_model <- lm(log(brain) ~ log(body), data = mammals)
plot(log(brain) ~ log(body), data = mammals, pch = 16, col = 4)
abline(brain_model)
```



# Chapter 40

## Using ggplot's stat\_summary

Some grouped data

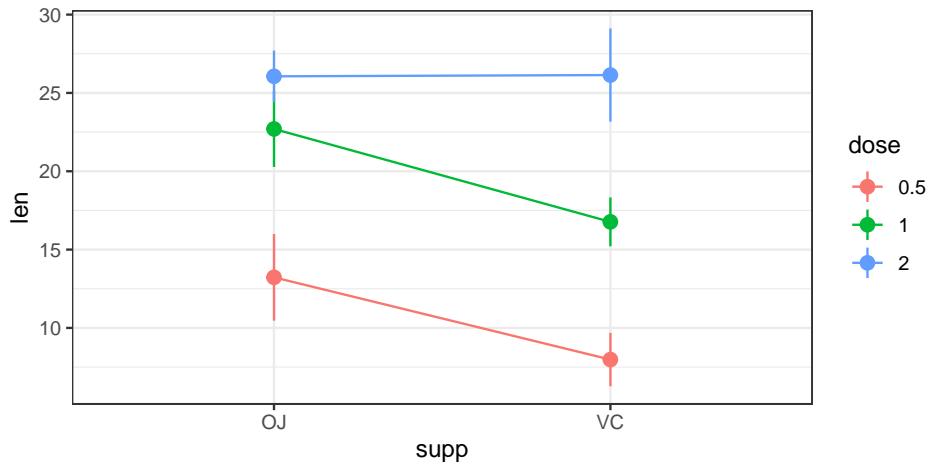
```
data(ToothGrowth)
ToothGrowth$dose <- factor(ToothGrowth$dose)
```

### 40.1 Standard plot of means w/ error bars

### 40.2 w/ stat\_summary(...geom = "pointrange")

```
se <- function(x){
 sd(x)/sqrt(length(x))
}

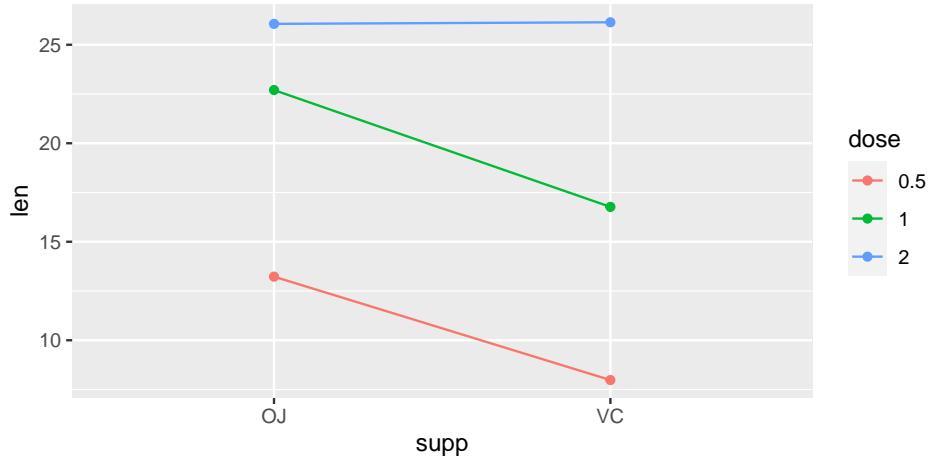
library(ggplot2)
ggplot(ToothGrowth,
 aes(y = len, x = supp, colour = dose, group = dose)) +
 stat_summary(fun.y = mean,
 fun.ymin = function(x) mean(x) - 1.96*se(x),
 fun.ymax = function(x) mean(x) + 1.96*se(x),
 geom = "pointrange") +
 stat_summary(fun.y = mean,
 geom = "line") +
 theme_bw()
```



### 40.3 Plot just means

#### 40.4 w/ geom = "point"

```
ggplot(ToothGrowth, aes(y = len, x = supp, colour = dose, group = dose)) +
 stat_summary(fun.y = mean,
 geom = "point") +
 stat_summary(fun.y = mean,
 geom = "line")
```



## Chapter 41

# Making a simple data frame for boxplots

```
library(compbio4all)
```

In this walkthrough we'll take a simple dataset that is presented as lists of separate numbers (eg 1,2,3,4) and put them into vectors. We'll then take these vectors and make simply dataframe. We'll then make a boxplot using ggplot2 and ggpubr.



## Chapter 42

# Working with data in 2 separate vectors

- Data often come in 2 separate sets of numbers, such as “the control group had lengths 4, 5, 6, 3, and 3, while the treatment group had length 11, 12, 6, 4 and 7”.
- One way to work with data like this in R is to load each set of numbers into a “vector” using the `c()` function

```
control <- c(4, 5, 6, 3, 3)
treatment <- c(11, 12, 6, 4, 7)
```

### 42.1 Summary Statistics

We can calculate the mean and other summary statistics of each vector on its own

```
mean(control)

[1] 4.2

sd(control)

[1] 1.30384

summary(control)

Min. 1st Qu. Median Mean 3rd Qu. Max.
3.0 3.0 4.0 4.2 5.0 6.0
```

## 42.2 Making a dataframe

To plot data its is almost always best to set it up in a dataframe.

Starting with the raw data we could make a dataframe like this

```
dat <- data.frame(length = c(4, 5, 6, 3, 3,
 11, 12, 6, 4, 7),
 group = c("C", "C", "C", "C", "C",
 "T", "T", "T", "T", "T"))
```

We can make a dataframe from the two vectors we've already typed up like this by surrounding them with a c() like this " c(control, treatment) "

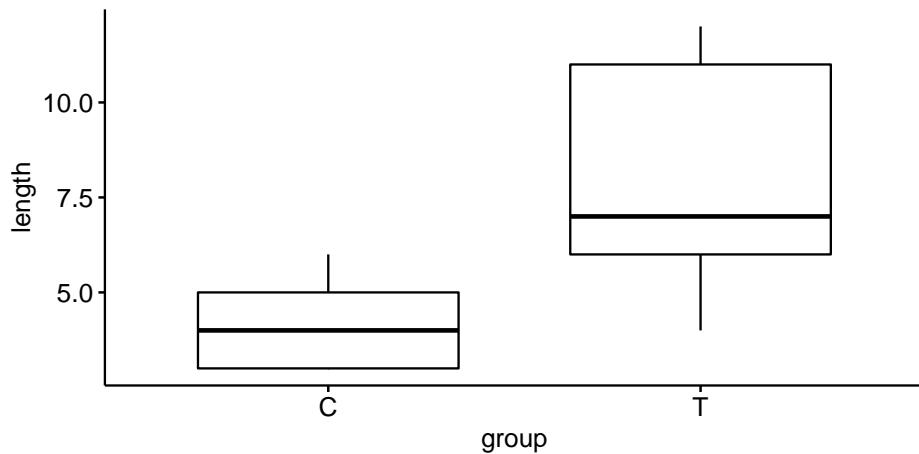
```
dat <- data.frame(length = c(control,
 treatment),
 group = c("C", "C", "C", "C", "C",
 "T", "T", "T", "T", "T"))
```

## 42.3 Making a boxplot

We can make a boxplot from the data like this using ggpibr

```
library(ggplot2)
library(ggpibr)

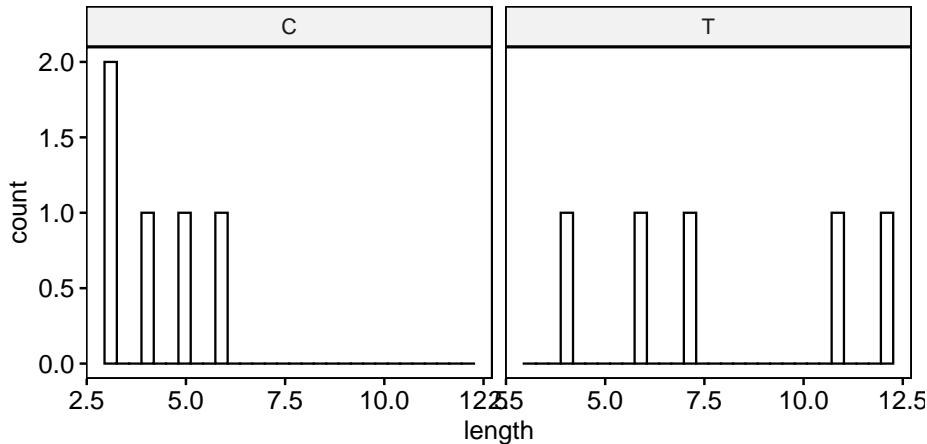
ggboxplot(data = dat,
 y = "length",
 x = "group")
```



## 42.4 Making a histogram

- An alternative to using a boxplot is a histogram.
- We can split up the data by treatment using `facet.by = "group"`
- Because my sample dataset has so few datapoints this doesn't look very good.

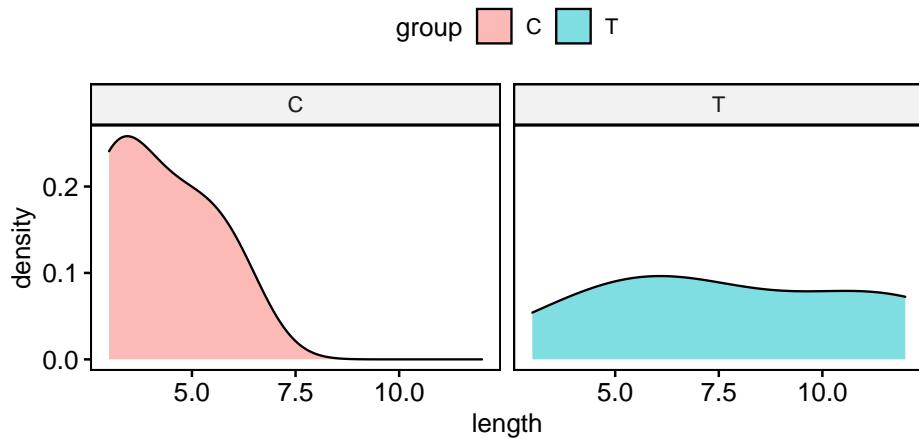
```
gghistogram(data = dat,
 x = "length",
 facet.by = "group")
```



## 42.5 Making a density plot

A plot that is similar to a histogram is a density plot.

```
ggdensity(data = dat,
 x = "length",
 facet.by = "group",
 fill = "group")
```



## 42.6 Advanced

You can save yourself some typing by using the `rep()` command when making your dataframe since the group codes “C” and “T” get repeated. IF you do this you need to make sure you keep track of all your parentheses.

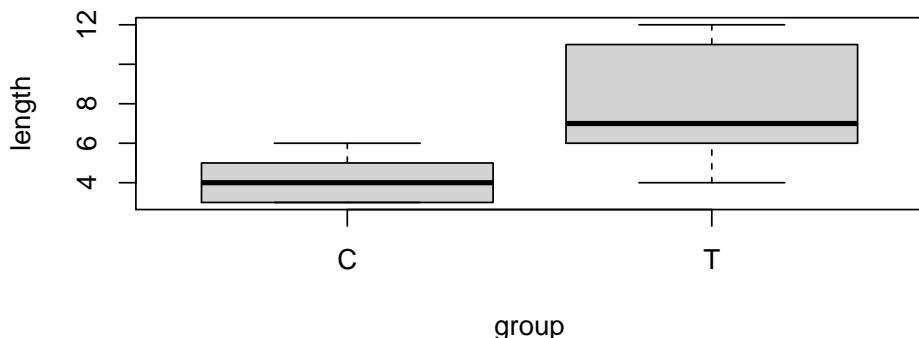
```
dat <- data.frame(length = c(control,
 treatment),
 group = c(rep("C", 5),
 rep("T", 5)))
```

## 42.7 Historical reference

### 42.7.1 Base R

In the old days (eg 2008...) we used to make our boxplot using the `boxplot` function.

```
boxplot(length ~ group, data = dat)
```



In the less old days (eg 2015) we used to make our boxplots using regular qplot or ggplot

### 42.7.2 ggplot's qplot

```
qplot(data = dat,
 y = length,
 x = group,
 geom = "boxplot")
```

### 42.7.3 Standard ggplot

```
ggplot(data = dat,
 aes(y = length,
 x = group)) +
 geom_boxplot()
```



## Chapter 43

# Barplots

Data from [https://useast.ensembl.org/Homo\\_sapiens/Info/Annotation](https://useast.ensembl.org/Homo_sapiens/Info/Annotation) Wed Jun 30 23:53:30 2021”

Assembly GRCh38.p13 (Genome Reference Consortium Human Build 38), INSDC Assembly GCA\_000001405.28, Dec 2013

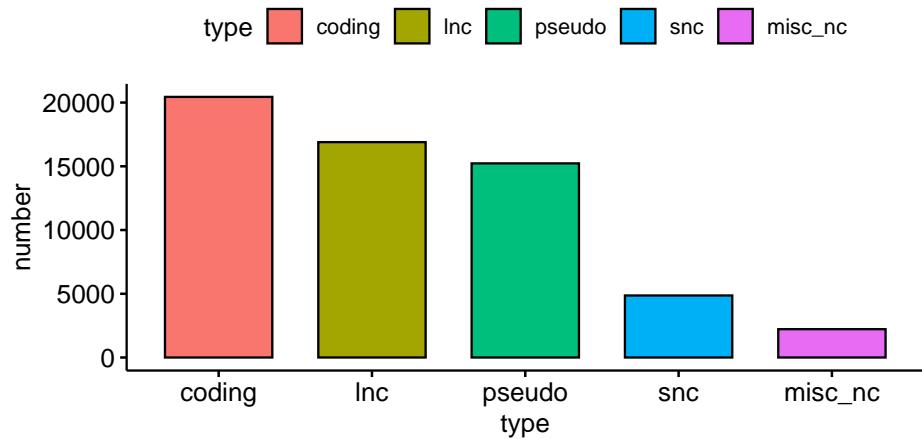
“This site provides a data set based on the December 2013 Homo sapiens high coverage assembly GRCh38 from the Genome Reference Consortium. This assembly is used by UCSC to create their hg38 database. The data set consists of gene models built from the genewise alignments of the human proteome as well as from alignments of human cDNAs using the cDNA2genome model of exonerate.”

Small non coding genes 4,865 Long non coding genes 16,896 (incl 307 readthrough) Misc non coding genes 2,221 Pseudogenes 15,228 (incl 6 readthrough)

```
number <- c(20442, 4865, 16896, 2221, 15228)
type <- c("coding", "snc", "lnc", "misc_nc", "pseudo")
genome_stats <- data.frame(type, number)
```

miniassignment - values that are out of order put into order

```
ggbarplot(data = genome_stats,
 y = "number",
 x = "type",
 fill = "type",
 order = c("coding", "lnc", "pseudo", "snc", "misc_nc"))
```



## **Chapter 44**

### **Boxplots - how they are made**



# Chapter 45

## Boxplots

### 45.1 Preliminaries

#### 45.1.1 Load packages

```
library(compbio4all)
library(ggplot2)
library(cowplot)
library(ggpubr)
```

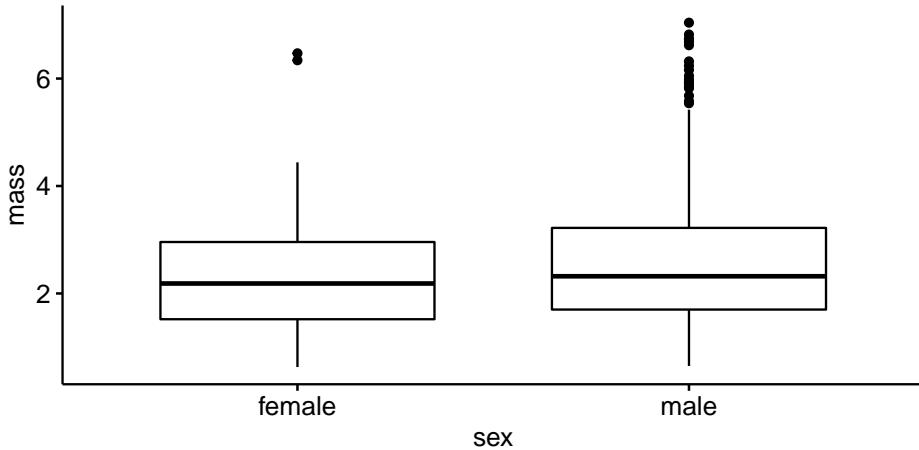
#### 45.1.2 Load data

```
data(frogarms)
```

### 45.2 Boxplots

Basic boxplot

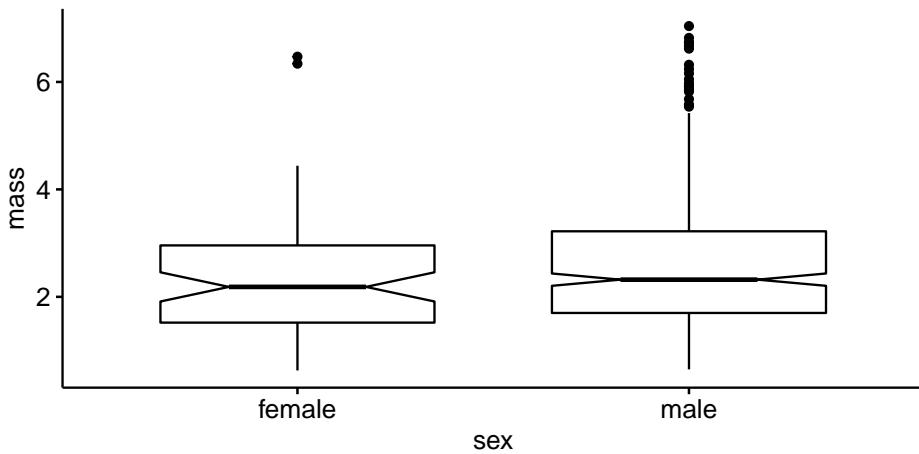
```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex")
```



### 45.3 Aside: Notched boxplots

We'll use the original frogarms data frame first for this. These aren't commonly used; the notches work kind of like confidence intervals to determine if medians are different.

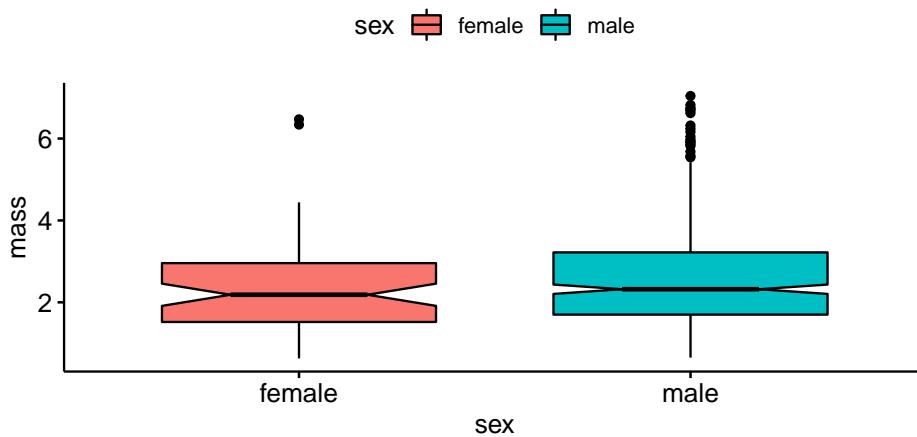
```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 notch = TRUE)
```



### 45.4 Fill

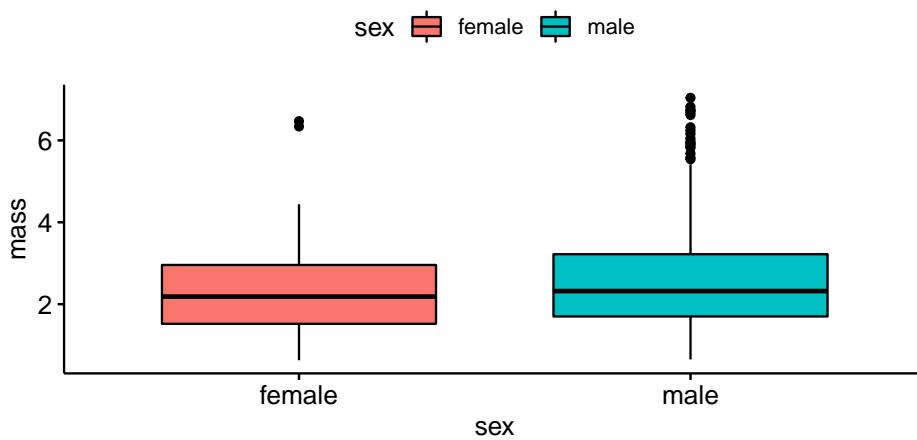
Add colored fill; note that it is “fill” not “color”. Color changes the color of the lines

```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 notch = TRUE,
 fill = "sex")
```



We can turn off the nothing by adding a “#” character before it. This is called **“commenting out”**

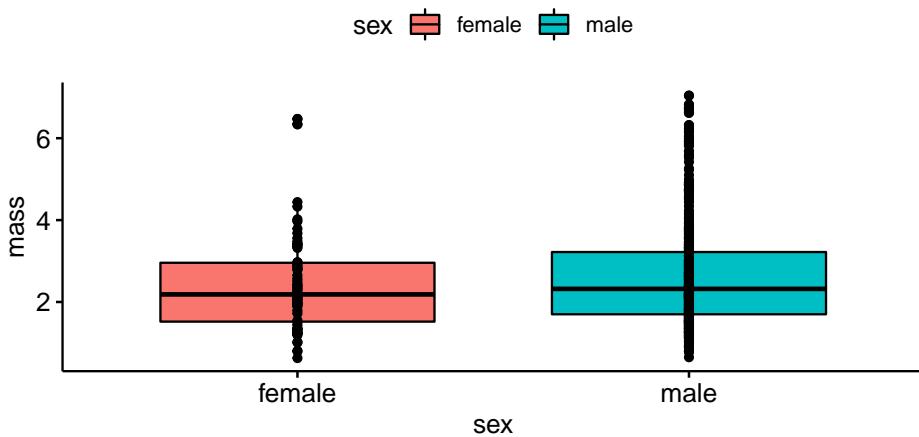
```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 #notch = TRUE,
 fill = "sex")
```



## 45.5 Adding raw data

Add raw data to the plot. This works best with small datasets

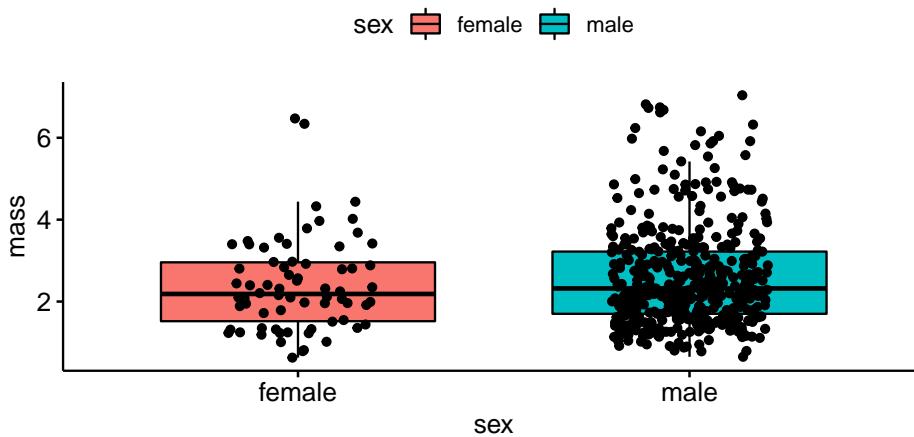
```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 #notch = TRUE,
 fill = "sex",
 add = "point")
```



## 45.6 Jitter the raw data

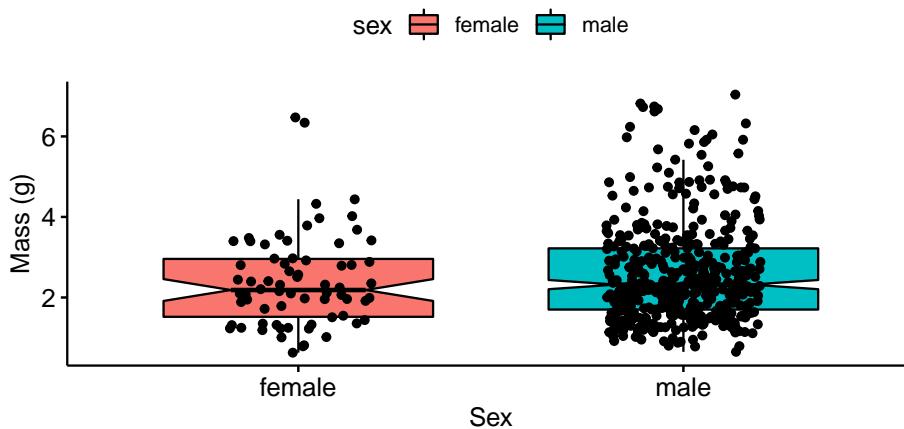
This can be helpful, though `ggpubr::ggboxplot` doesn't allow much control over the "jittering". Jittering helpful when you have large datasets and want to avoid overlap in the points.

```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 #notch = TRUE,
 fill = "sex",
 add = "jitter")
```



Label axes

```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 notch = TRUE,
 fill = "sex",
 add = "jitter",
 xlab = "Sex", #x axis (horizontal)
 ylab = "Mass (g)") #y axis (vertical)
```



Add title not usually done for publication but useful for keeping track of things and for presentations

```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 notch = TRUE,
 fill = "sex".
```

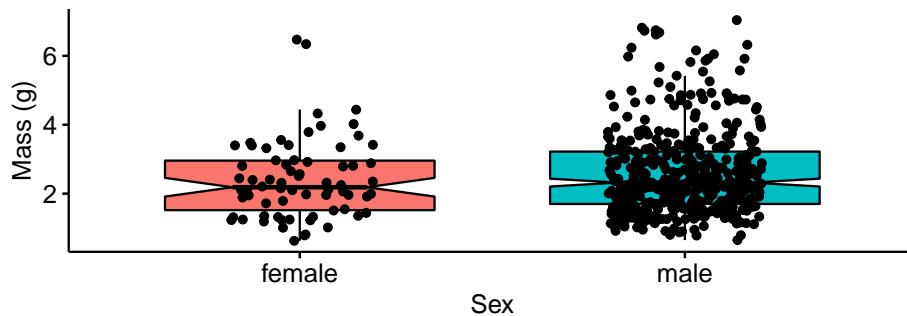
```

add = "jitter",
xlab = "Sex",
ylab = "Mass (g)",
main = "Mass of Australian frogs by sex") #Main title

```

Mass of Australian frogs by sex

sex female male



Move legend to bottom

```

ggboxplot(data = frogarms,
y = "mass",
x = "sex",
notch = TRUE,
fill = "sex",
add = "jitter",
xlab = "Sex",
ylab = "Mass (g)",
main = "Mass of frogs by sex",
legend = "bottom")

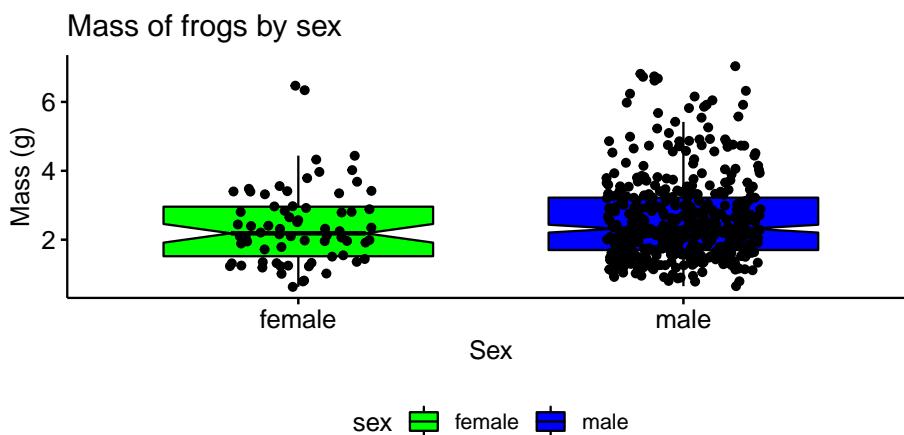
```

Mass of frogs by sex

sex female male

Change color palette

```
ggboxplot(data = frogarms,
 y = "mass",
 x = "sex",
 notch = TRUE,
 fill = "sex",
 add = "jitter",
 xlab = "Sex",
 ylab = "Mass (g)",
 main = "Mass of frogs by sex",
 legend = "bottom",
 palette = c("green","blue"))
```



#### 45.6.1 Plotting multiple plots with cowplot::plot\_grid

```
my.frogs <- make_my_data2L(dat = frogarms,
 my.code = "nlb24", # #<= change this!
 cat.var = "sex",
 n.sample = 20,
 with.rep = FALSE)

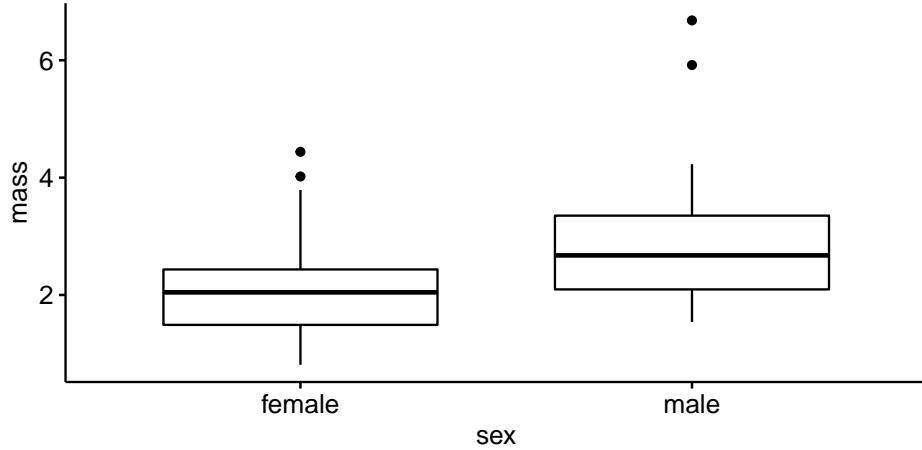
Codes should only contain letters and numbers
Your special code is 4782969
(You don't really need to know this, though). 4782969
NOTE: This function only works properly for data with TWO levels to the categorical var.
eg male vs. female; it doesn't work for >2 levels (eg red vs blue vs. green)
dim(my.frogs)

[1] 40 8
```

We can save a plot to an R object

```
gg.my.frogs<- ggboxplot(data = my.frogs,
 y = "mass",
 x = "sex")
```

Call just the object (eg, just type it into the console. or highlight just the word)  
`gg.my.frogs`

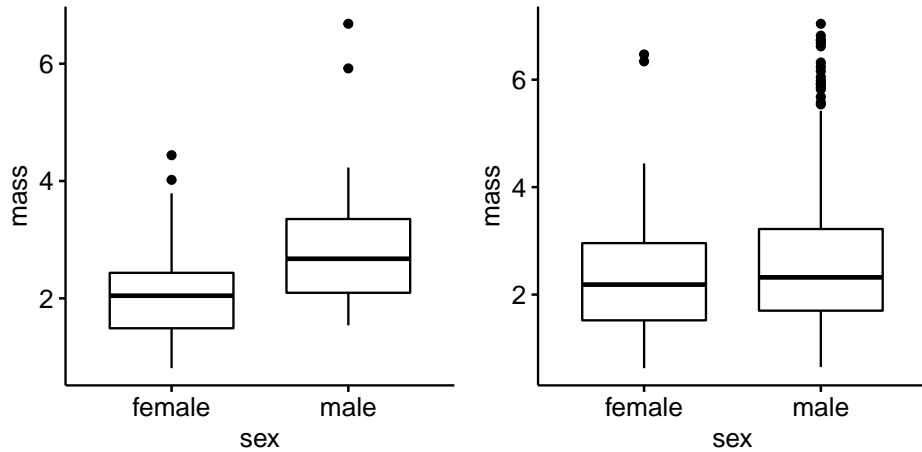


Make an object using the frogarms data

```
gg.frogarms <- ggboxplot(data = frogarms, #use original data
 y = "mass",
 x = "sex")
```

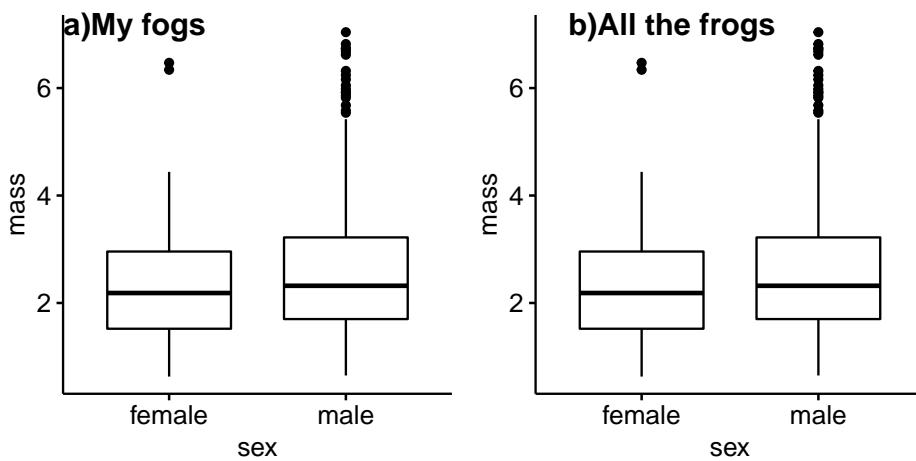
Now plot both

```
plot_grid(gg.my.frogs,
 gg.frogarms)
```



Add labels. Note that alignment is off sometimes.

```
plot_grid(gg.frogarms,
 gg.frogarms,
 labels = c("a)My frogs","b)All the frogs"))
```





# Chapter 46

## Central tendency

```
library(compbio4all)
```

### 46.1 Preliminaries

#### 46.1.1 Load packagtes

```
library(compbio4all)
```

#### 46.1.2 Load data

```
data(frogarms)
```

### 46.2 Getting to know the frogs

```
dim(frogarms)
```

```
[1] 509 7
```

```
nrow(frogarms)
```

```
[1] 509
```

```
ncol(frogarms)
```

```
[1] 7
```

```
head(frogarms)
```

```

i.row i.frog sex mass sv.length forearm arm
1 1 1 male 2.50 32.56 3.45 4.00
2 2 2 male 2.28 29.32 3.35 3.67
3 3 3 male 2.38 30.30 3.56 4.86
4 4 4 male 2.53 26.67 3.58 4.08
5 5 5 male 1.52 25.92 2.87 3.89
6 6 6 male 2.06 27.16 3.42 3.56
tail(frogarms)

i.row i.frog sex mass sv.length forearm arm
504 504 511 female 3.40 37.23 2.60 2.40
505 505 512 female 1.25 27.52 1.59 1.51
506 506 513 female 3.42 38.44 2.64 2.44
507 507 514 female 3.97 37.10 2.32 2.42
508 508 515 female 3.48 38.02 2.48 2.54
509 509 516 female 2.57 31.96 2.08 1.99
names(frogarms)

[1] "i.row" "i.frog" "sex" "mass" "sv.length" "forearm"
[7] "arm"
?frogarms

```

## 46.3 Summary statistics

R is a giant calculator

### 46.3.1 Overall summary

Whole dataframe

```

summary(frogarms)

i.row i.frog sex mass sv.length
Min. : 1 Min. : 1.0 female: 70 Min. :0.630 Min. :20.97
1st Qu.:128 1st Qu.:129.0 male :439 1st Qu.:1.690 1st Qu.:27.95
Median :255 Median :256.0 Median :2.290 Median :31.07
Mean :255 Mean :257.8 Mean :2.577 Mean :31.40
3rd Qu.:382 3rd Qu.:387.0 3rd Qu.:3.210 3rd Qu.:34.49
Max. :509 Max. :516.0 Max. :7.040 Max. :43.86
##
forearm arm
Min. :1.160 Min. :1.220
1st Qu.:2.655 1st Qu.:3.165
Median :3.115 Median :4.020
Mean :3.194 Mean :4.083

```

```
3rd Qu.:3.750 3rd Qu.:4.855
Max. :5.570 Max. :7.910
NA's :7 NA's :7

Just a single column
summary(frogarms$mass)

Min. 1st Qu. Median Mean 3rd Qu. Max.
0.630 1.690 2.290 2.577 3.210 7.040
```

#### 46.3.2 Individual summary stats

```
mean(frogarms$mass)

[1] 2.576994
var(frogarms$mass)

[1] 1.4948
• median()
```



# Chapter 47

## Error plots

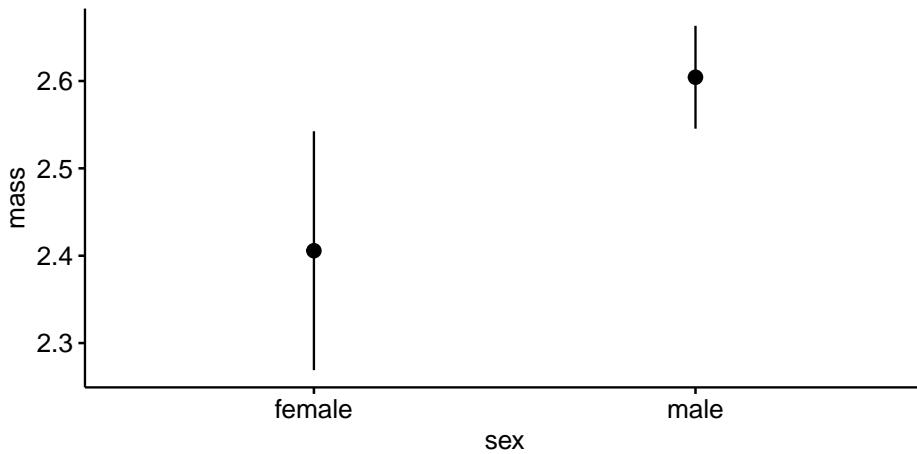
```
library(compbio4all)
data(frogarms)

library(ggplot2)
library(ggpubr)
library(cowplot)
```

### 47.1 Plot means with error bars

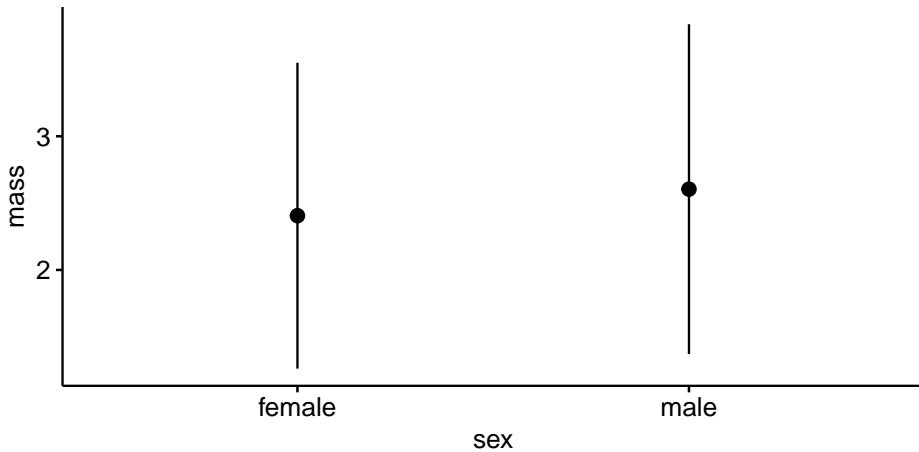
Super hand function ggerrorplot() Default is mean +/- 1 standard error

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex")
```



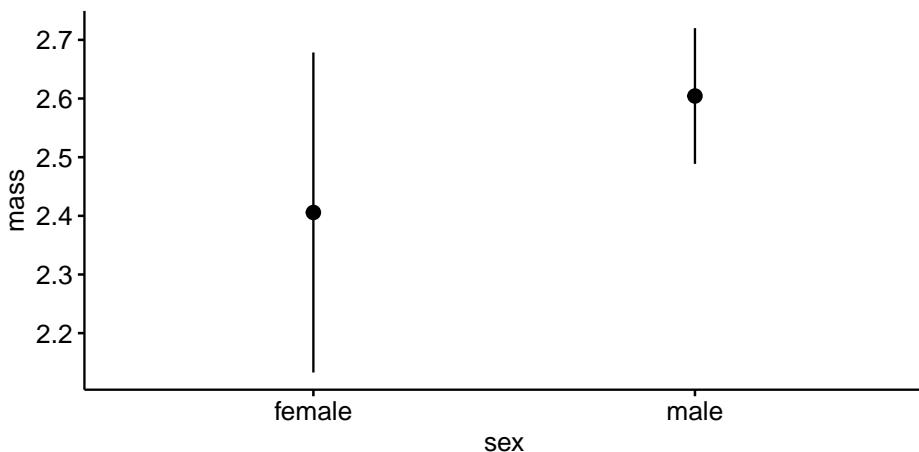
Mean and se; note sd is not often used in publications

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_sd")
```



Mean and 95% confidence interval

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci")
```



Plot your data and original data

```
#your data
gg.frogarms <- ggerrorplot(data = frogarms,
 y = "mass",
```

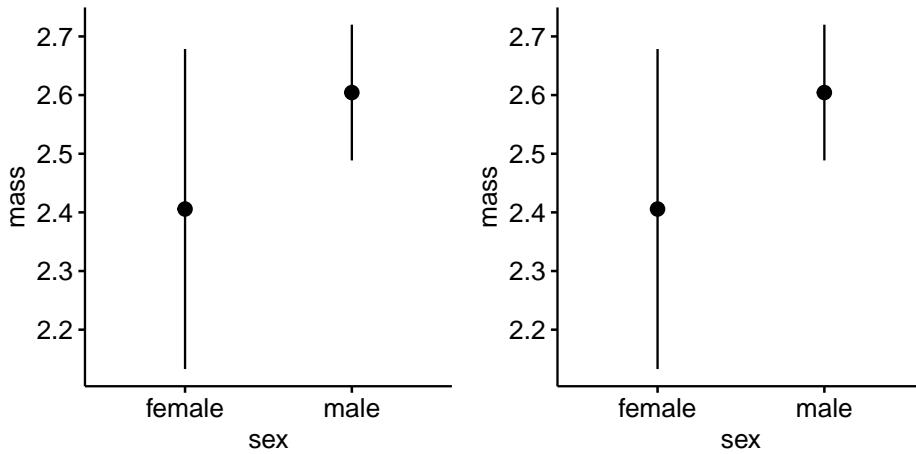
```

x = "sex",
desc_stat = "mean_ci")

#all of the data
gg.all.frogs <- ggerrorplot(data = frogarms, #change data
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci")

cowplot:::plot_grid(gg.frogarms, gg.all.frogs)

```

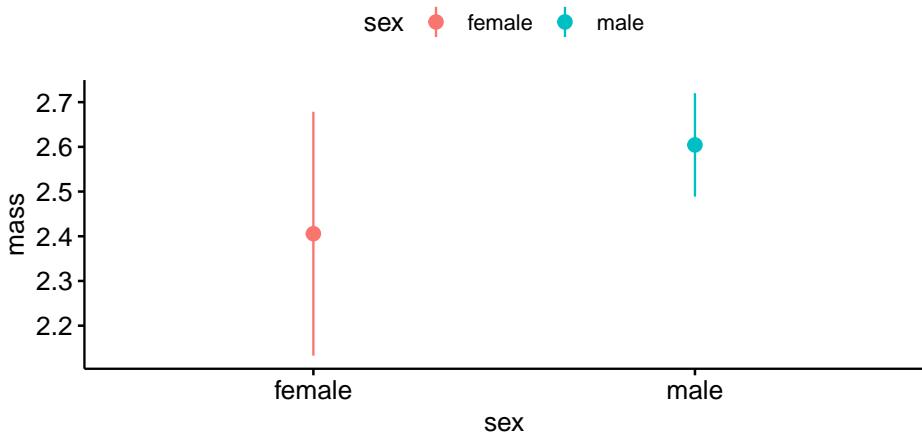


Set colors

```

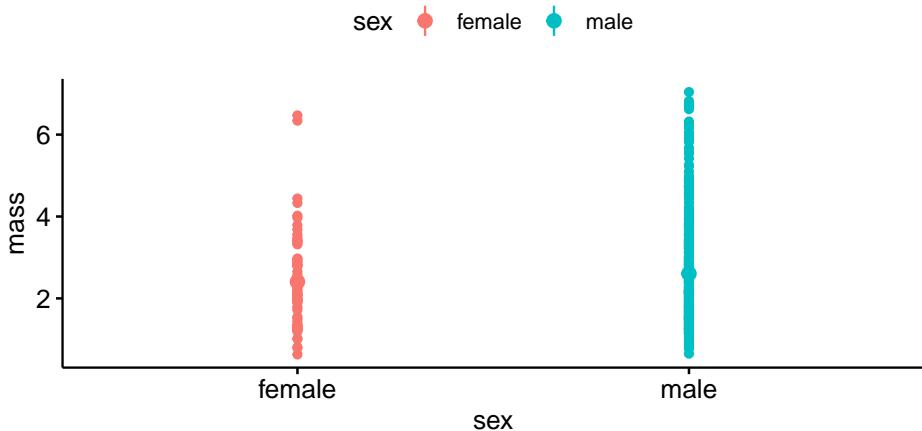
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci",
 color = "sex")

```



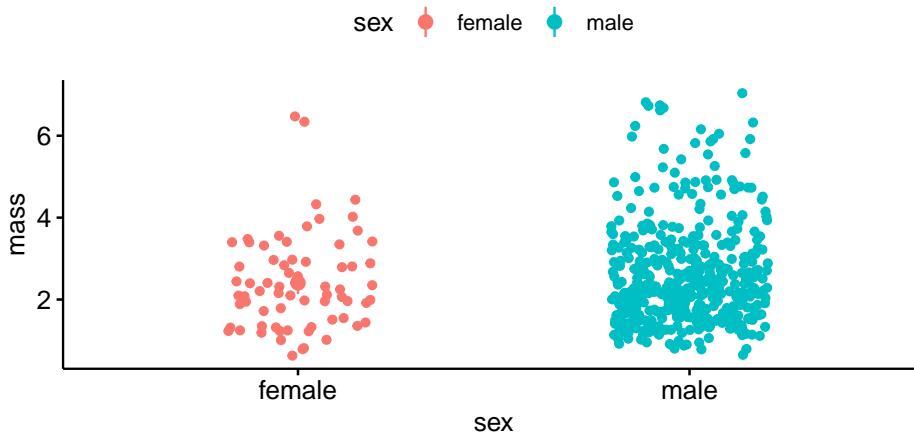
Add raw data. kinda crazy

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci",
 color = "sex",
 shape = "sex",
 add = "point")
```



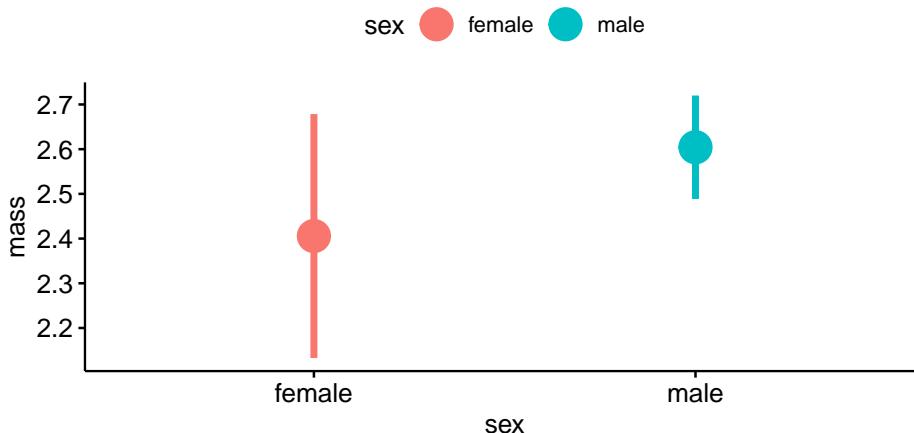
Jitter raw data. even crazier

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci",
 color = "sex",
 add = "jitter")
```



Back to just the means Increase size

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci",
 color = "sex",
 size = 1.5) #
```

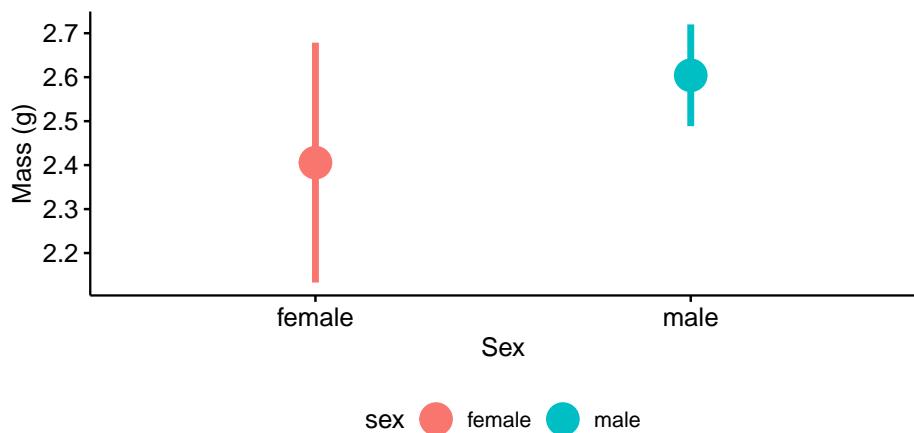


Move legend to the bottom

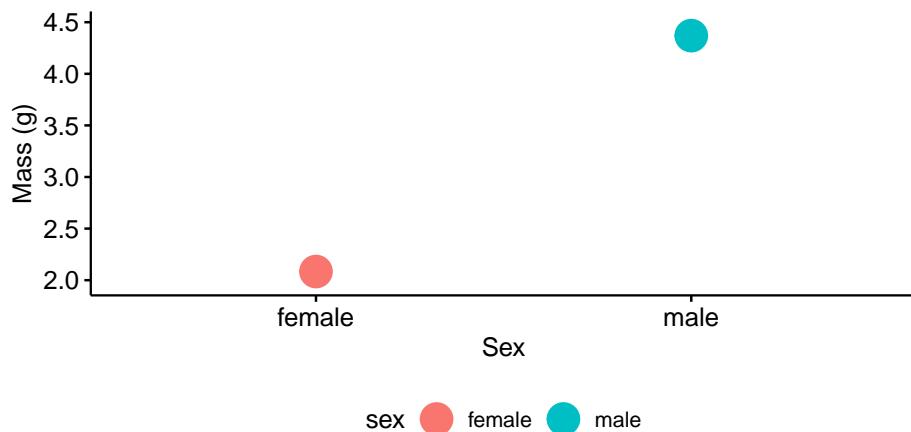
?set all of this stuff as eval = F and have students figure out how to add it? maybe not - goal is just to do "1st encounter"

```
ggerrorplot(data = frogarms,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci",
 color = "sex",
```

```
size = 1.5,
xlab = "Sex",
ylab = "Mass (g)",
legend = "bottom") #
```



```
ggerrorplot(data = frogarms,
 y = "arm",
 x = "sex",
 desc_stat = "mean_ci",
 color = "sex",
 size = 1.5,
 xlab = "Sex",
 ylab = "Mass (g)",
 legend = "bottom") #
```





## **Chapter 48**

### **How histograms are made**



# Chapter 49

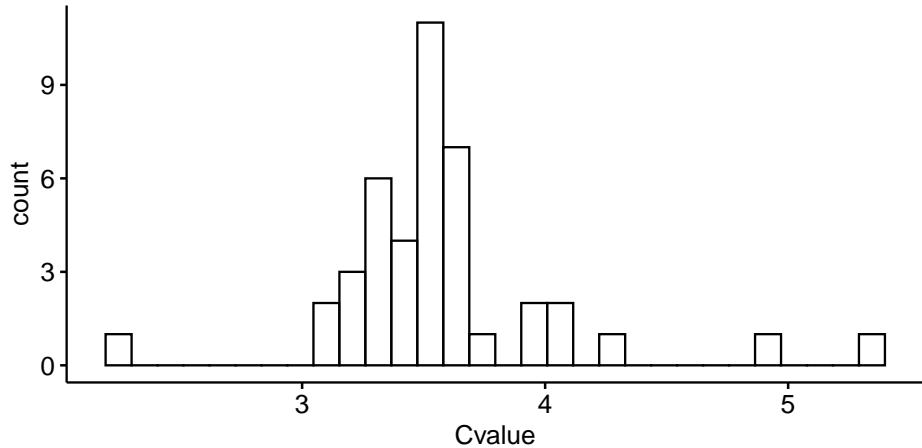
## Histograms

### 49.1 Preliminaries

```
library(compbio4all)
library(ggplot2)
library(cowplot)
library(ggpubr)

Cvalue <- c(3.63, 3.25, 2.26, 3.43, 3.48,
 3.98, 4.90, 4.02, 3.61, 3.52,
 3.36, 3.36, 3.4, 4.08, 3.93,
 3.57, 3.23, 3.5, 3.08, 3.37,
 3.54, 3.28, 3.25, 3.76, 3.41,
 3.56, 3.29, 3.56, 3.33, 3.5,
 3.5, 3.12, 3.67, 4.32, 3.58,
 3.61, 3.63, 3.53, 3.68, 3.66,
 3.3, 5.36)
i <- 1:length(Cvalue)
df <- data.frame(i,Cvalue)

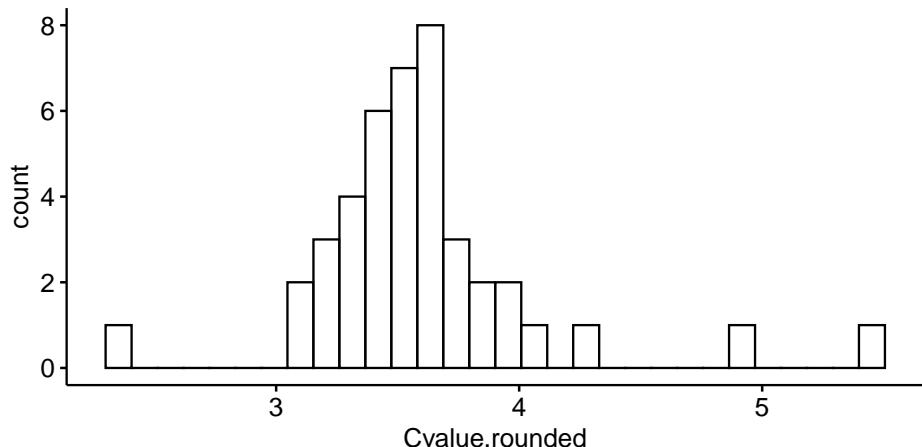
gghistogram(data = df, x = "Cvalue")
```



```
Cvalue <- sort(Cvalue)
Cvalue.rounded <- round(Cvalue,digits = 1)
length(seq(2.3,5.4, by = 0.1))
```

```
[1] 32
df2 <- data.frame(i,Cvalue.rounded)

gghistogram(data = df2, x = "Cvalue.rounded")
```



```
x <- Cvalue.rounded

x.unique <- as.numeric(names(table(x)))

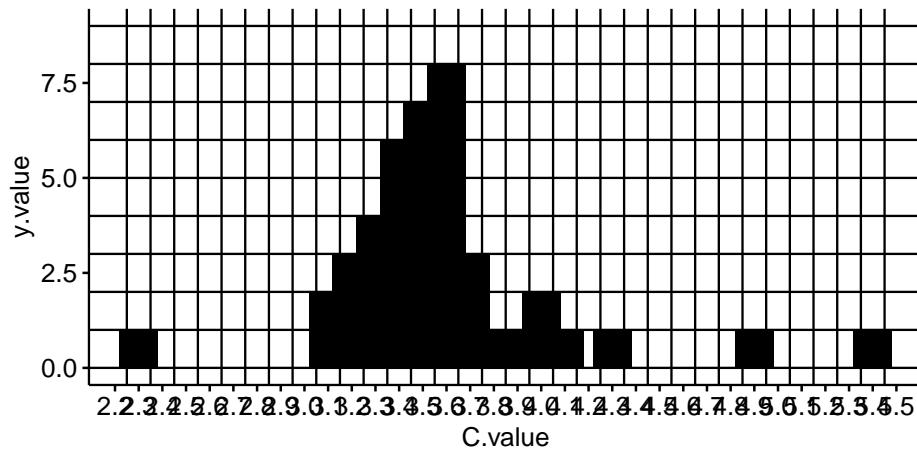
y <- rep(0, length(x))
for(i in 1:length(x.unique)){
 x.i <- which(Cvalue.rounded == x.unique[i])
```

```

y[x.i] <- 1:length(x.i)
}

df3 <- data.frame(name = 1:length(x),
 C.value = x,
 y.value = c(y-0.5))
ggscatter(data = df3,
 x = "C.value",
 y = "y.value",
 # shape = 0,
 shape = 15,
 size = 8#,
 #color = "white",
 #label.rectangle = T,
 #label = "name"
) +
geom_vline(xintercept = c(seq(2.25,5.45,0.1)))+
 geom_hline(yintercept = c(seq(0,9,1))) +
 scale_x_continuous(breaks=seq(2.2,5.5,0.1))

```



### 49.1.2 Load data

“hpc” stands for “human protein-coding” genes.

```

data("genes_hpc")
data("frogarms")

```

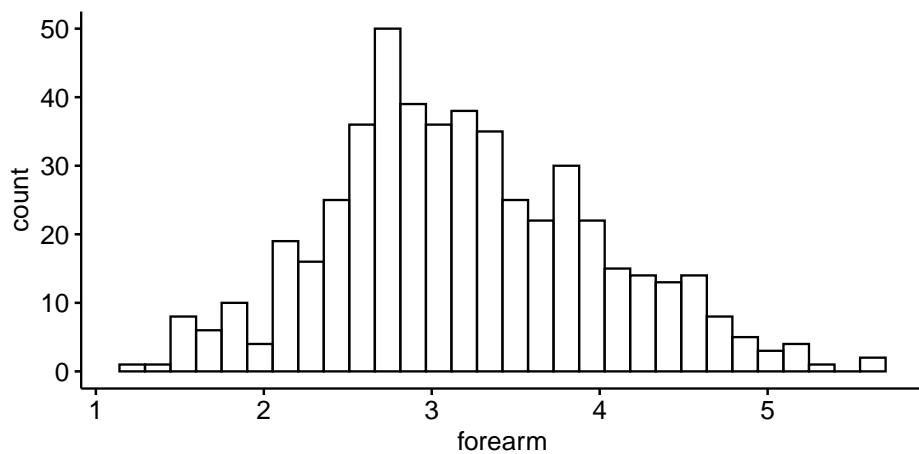
## 49.2

### 49.3 Examples of histogram

#### 49.3.1 Normal-ish data

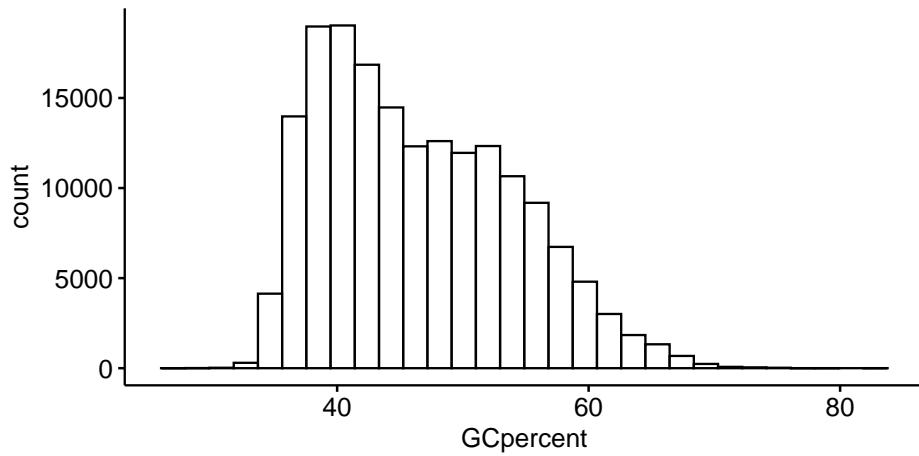
A normal curve looking one

```
gghistogram(data = frogarms,
 x = "forearm")
```



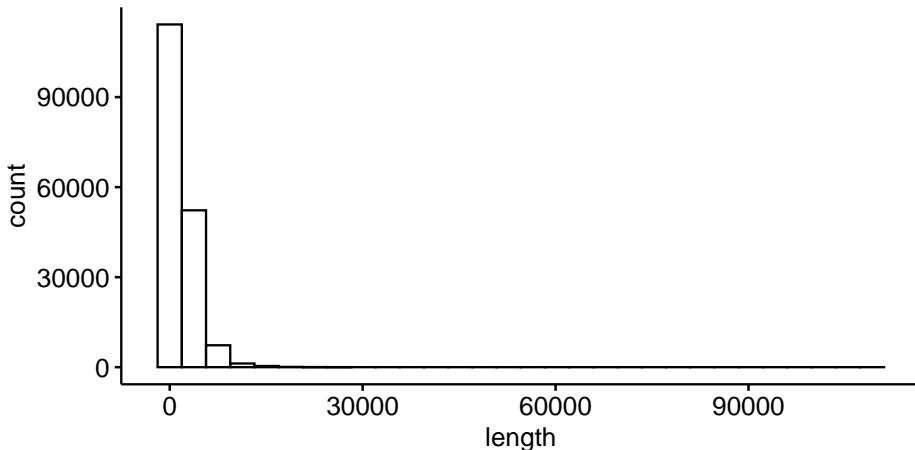
### 49.4 Skewed data

```
gghistogram(data = genes_hpc,
 x = "GCpercent")
```



## 49.5 Heavily skewed data

```
gghistogram(data = genes_hpc,
 x = "length")
```



Usually a warning message: Using `bins = 30` by default. Pick better value with the argument `bins`

Distribution has a few outliers

```
max(genes_hpc$length)
```

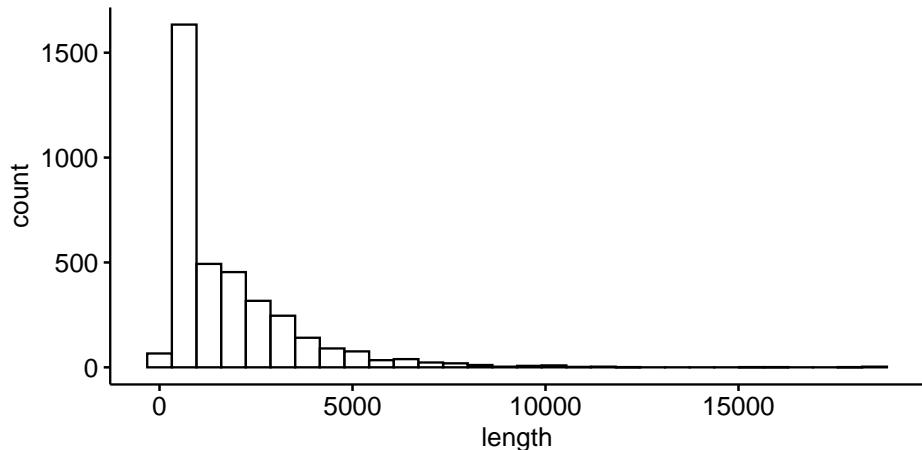
```
[1] 109224
ges <- which(genes_hpc$length > 15000)

i.MT <- which(genes_hpc$chromosome == "MT")
i.X <- which(genes_hpc$chromosome == "X")
i.Y <- which(genes_hpc$chromosome == "Y")
i.1 <- which(genes_hpc$chromosome == "1")
i.15 <- which(genes_hpc$chromosome == "15")
i.22 <- which(genes_hpc$chromosome == "22")
```

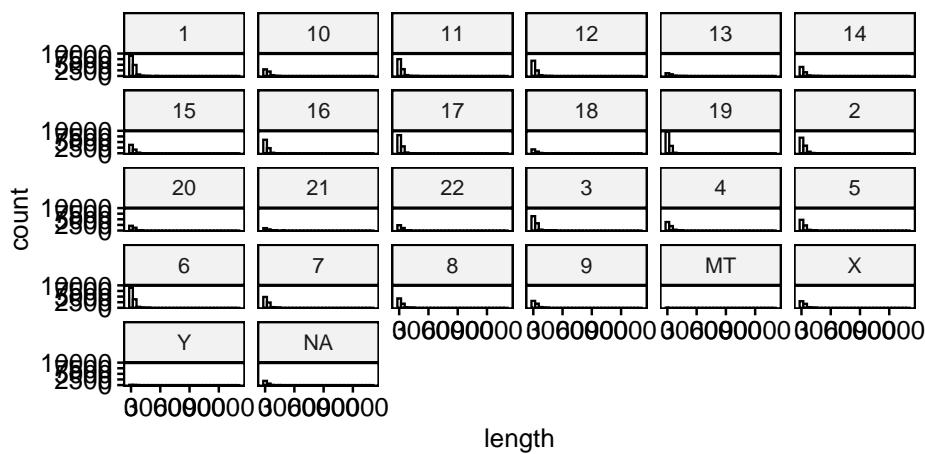
## 49.6

X chromosome looks nice Y chromosome goofy 1 looks ok, large peak 15 ok, large peak 22 better than 1

```
gghistogram(data = genes_hpc[i.22,],
 x = "length",
 bins = 30)
```



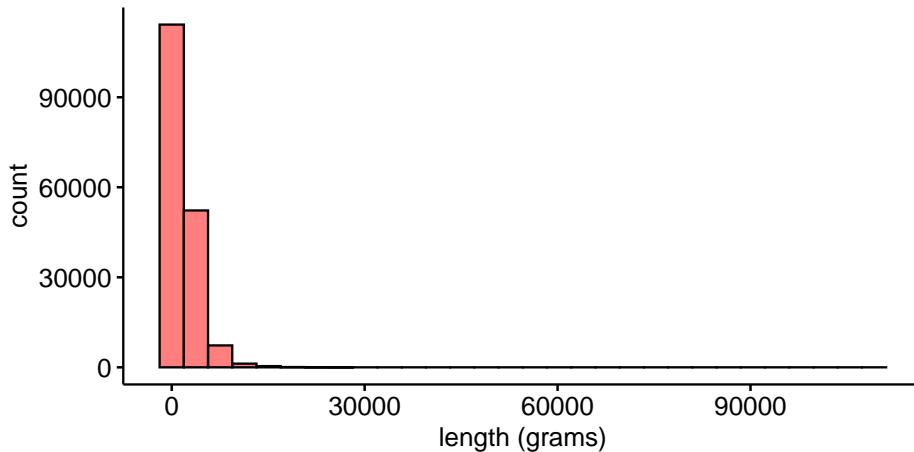
```
gghistogram(data = genes_hpc,
 x = "length",
 facet.by = "chromosome",
 bins = 30)
```



## 49.7 Fill

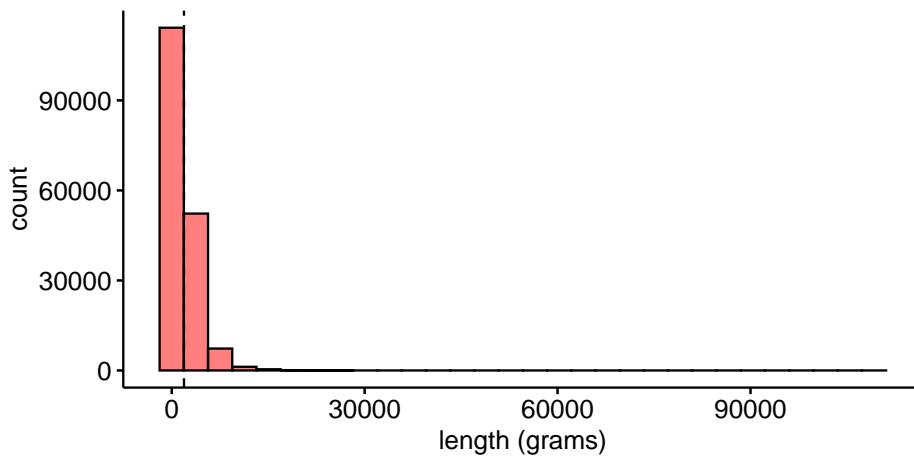
Add colored fill; note that it is “fill” not “color”. Color changes the color of the lines

```
gghistogram(data = genes_hpc,
 x = "length",
 fill = "red",
 xlab = "length (grams)")
```



## 49.8 Central tendency

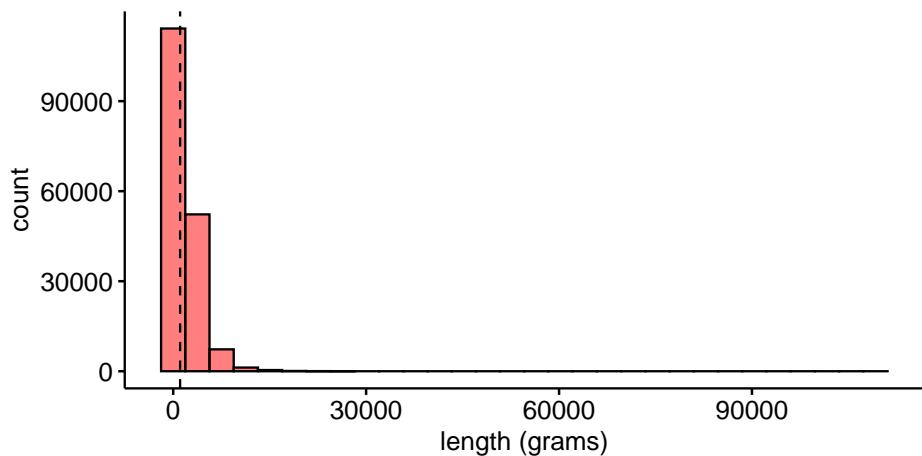
```
gghistogram(data = genes_hpc,
 x = "length",
 fill = "red",
 xlab = "length (grams)",
 add = "mean")
```



Warning messages: 1: Using `bins = 30` by default. Pick better value with the argument `bins`. 2: `geom_vline()`: Ignoring `mapping` because `xintercept` was provided. 3: `geom_vline()`: Ignoring `data` because `xintercept` was provided.

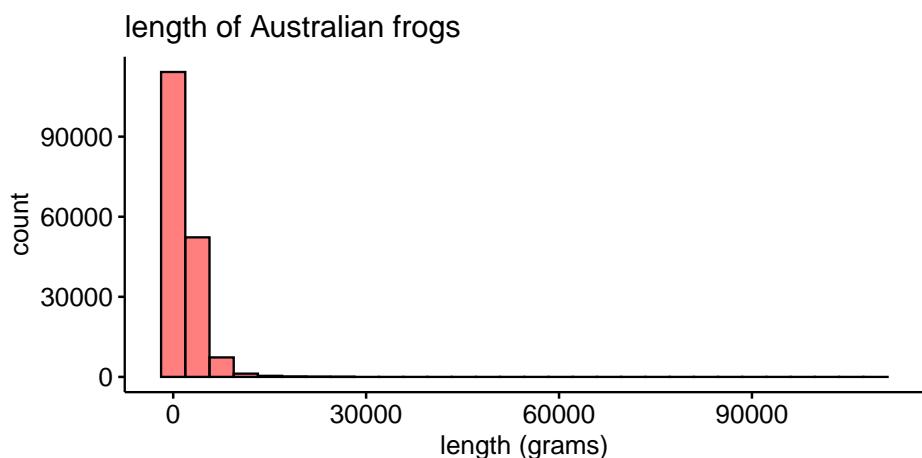
## 49.9 Median

```
gghistogram(data = genes_hpc,
 x = "length",
 fill = "red",
 xlab = "length (grams)",
 add = "median")
```



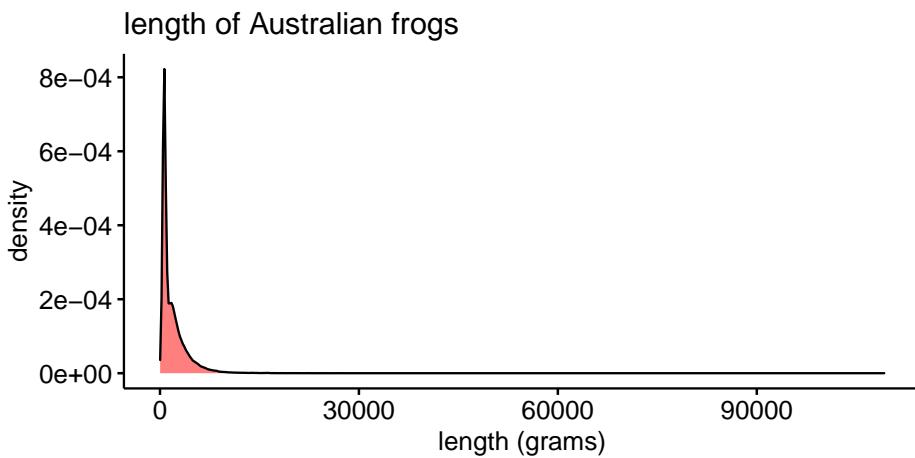
Add title not usually done for publication but useful for keeping track of things and for presentations

```
gghistogram(data = genes_hpc,
 x = "length",
 fill = "red",
 xlab = "length (grams)",
 main = "length of Australian frogs ") #Main title
```



## 49.10 Density plot

```
ggdensity(data = genes_hpc,
 x = "length",
 fill = "red",
 xlab = "length (grams)",
 main = "length of Australian frogs") #Main title
```



### 49.10.1 Plotting multiple plots with cowplot::plot\_grid

make into boxplot version exercise since shown first here as hist

```
my.frogs <- make_my_data2L(dat = frogarms,
 my.code = "nlb24", # #<= change this!
 cat.var = "sex",
 n.sample = 20,
 with.rep = FALSE)

Codes should only contain letters and numbers
Your special code is 4782969
(You don't really need to know this, though). 4782969
NOTE: This function only works properly for data with TWO levels to the categorical var.
eg male vs. female; it doesn't work for >2 levels (eg red vs blue vs. green)
dim(my.frogs)

[1] 40 8
```

We can save a plot to an R object

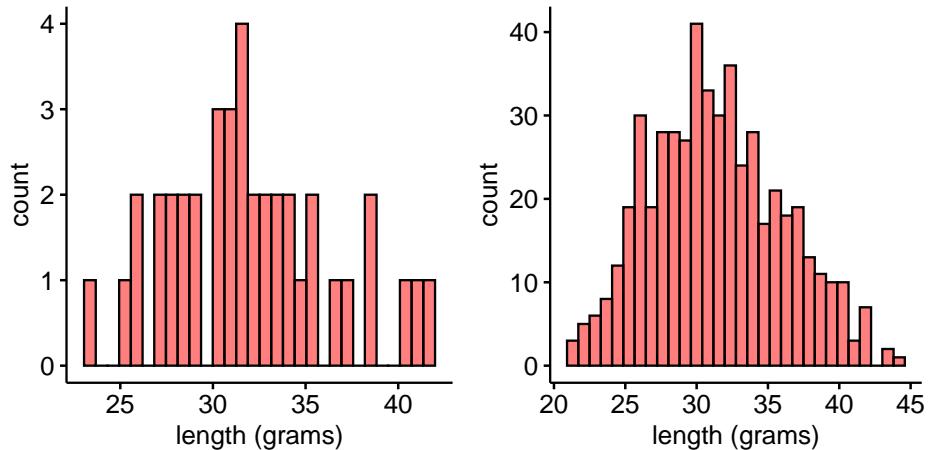
```
gg.hist.my.frogs <- gghistogram(data = my.frogs,
 x = "sv.length",
 fill = "red",
```

```
xlab = "length (grams)" #Main title

gg.hist.frog <- gghistogram(data = frogarms,
 x = "sv.length",
 fill = "red",
 xlab = "length (grams)" #Main title
```

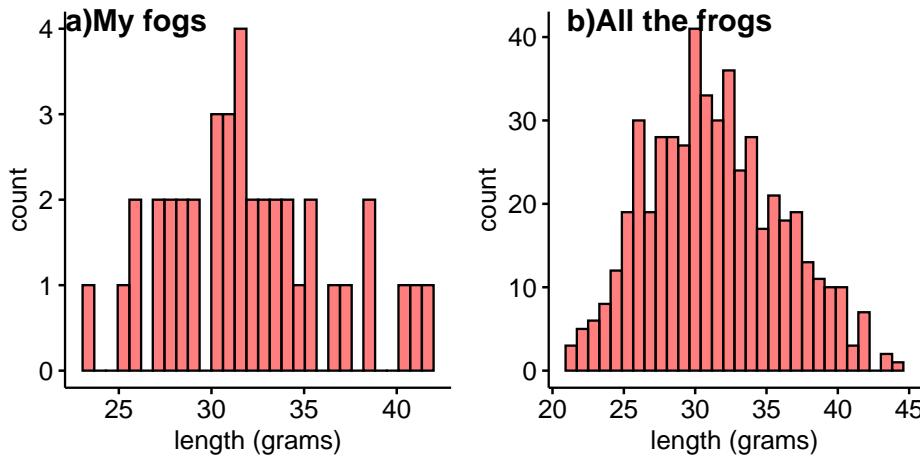
Now plot both

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog)
```



Add labels. Note that alignment is off sometimes.

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog,
 labels = c("a)My frogs","b)All the frogs"))
```



Add lines for means

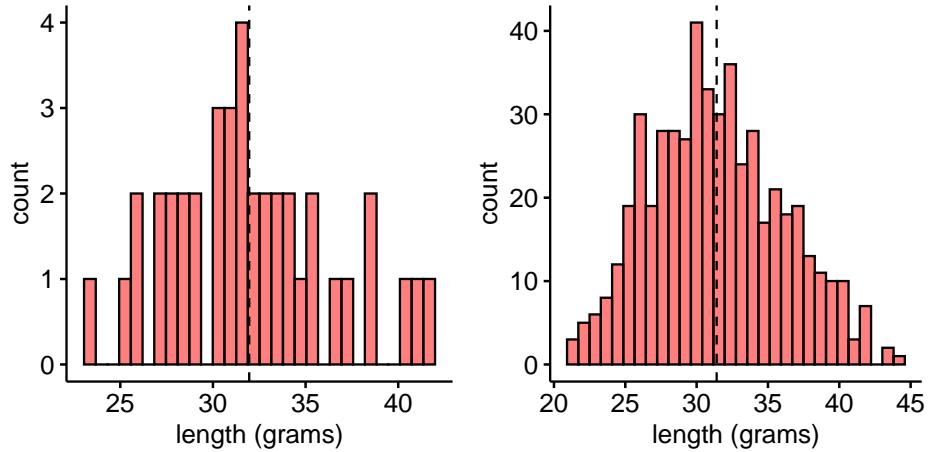
We can save a plot to an R object

```
gg.hist.my.frogs <- gghistogram(data = my.frogs,
 x = "sv.length",
 fill = "red",
 add = "mean",
 xlab = "length (grams)") #Main title

gg.hist.frog <- gghistogram(data = frogarms,
 x = "sv.length",
 fill = "red",
 add = "mean",
 xlab = "length (grams)") #Main title
```

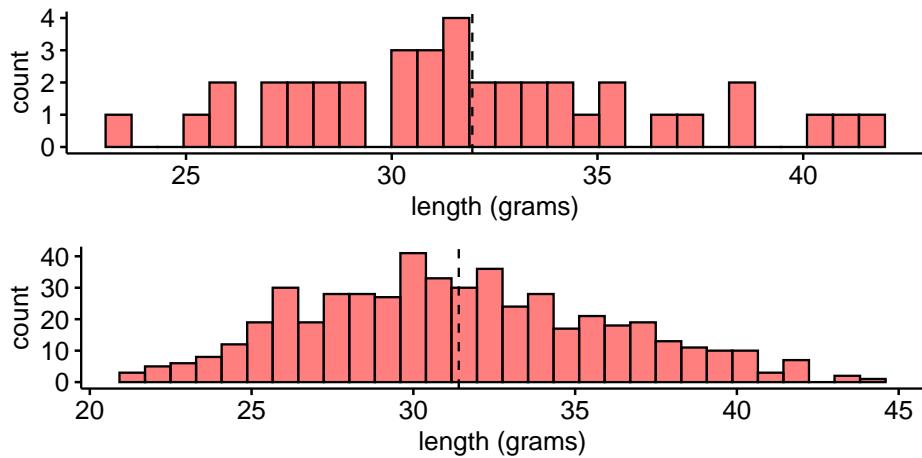
Now plot both

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog)
```



### 49.11 Align vert

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog,
 nrow = 2)
```

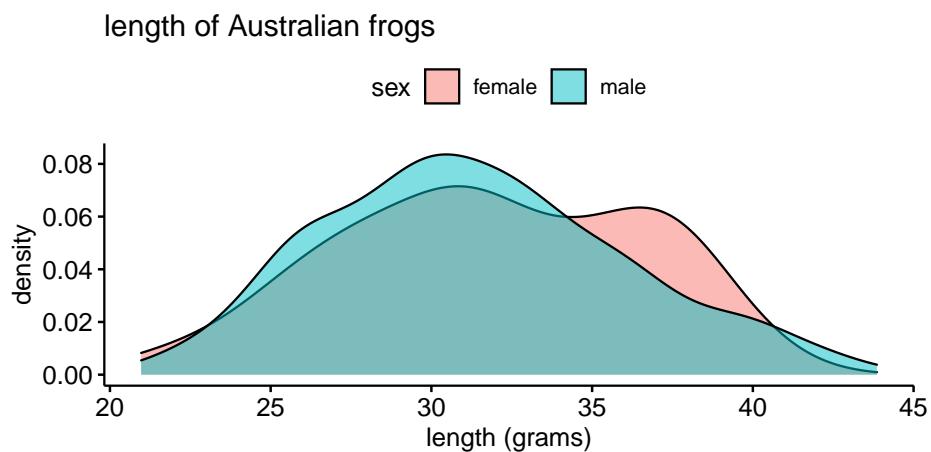


### 49.12 Density plot overlay

### 49.13 Density plot

```
ggdensity(data = frogarms,
 x = "sv.length",
 fill = "sex",
```

```
xlab = "length (grams)",
main = "length of Australian frogs ") #Main title
```





# Chapter 50

## Boxplots

### 50.1 Preliminaries

```
50.1.1 Load packages
library(compbio4all)
library(ggplot2)
library(cowplot)
library(ggpubr)
```

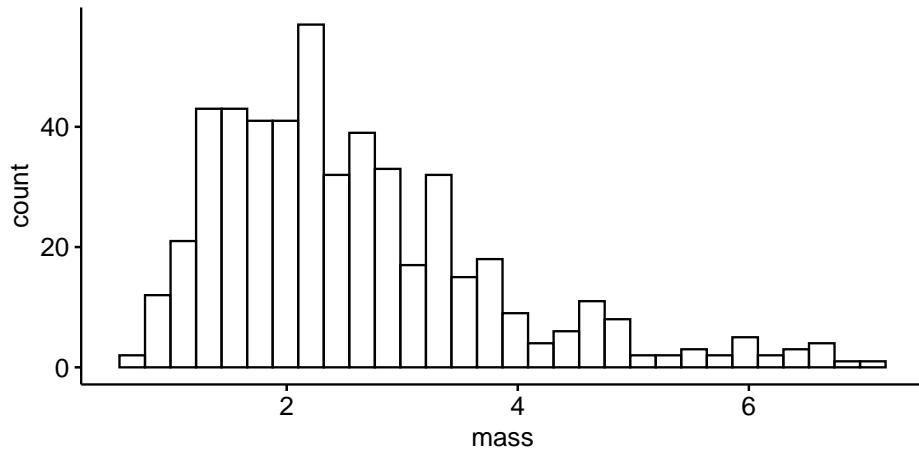
```
50.1.2 Load data
```

```
data(frogarms)
```

### 50.2 Boxplots

Basic histogram

```
gghistogram(data = frogarms,
 x = "mass")
```

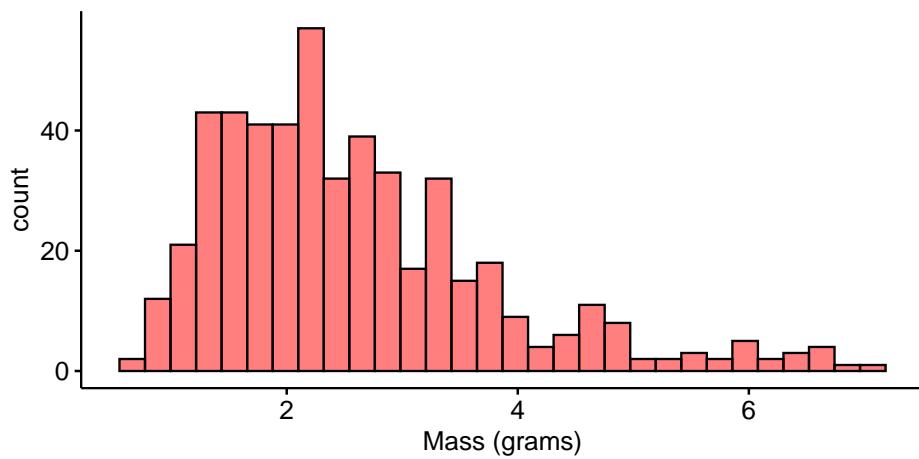


Usually a warning Warning message: Using `bins = 30` by default. Pick better value with the argument `bins`

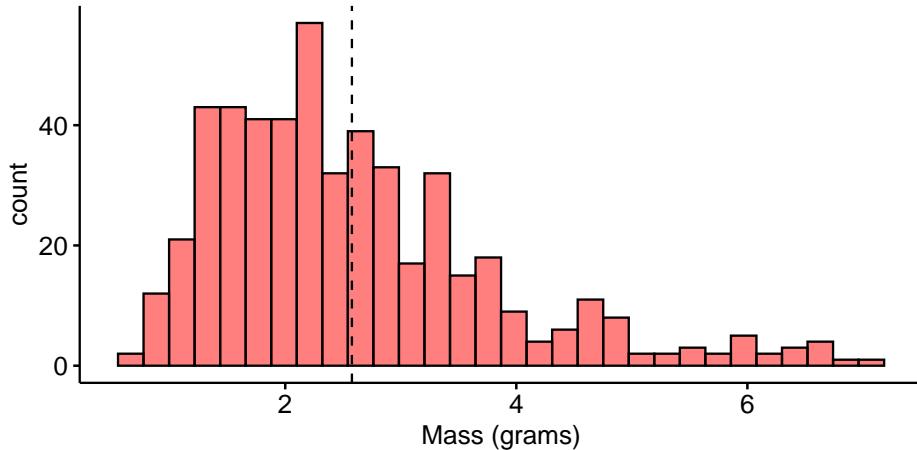
### 50.3 Fill

Add colored fill; note that it is “fill” not “color”. Color changes the color of the lines

```
gghistogram(data = frogarms,
 x = "mass",
 fill = "red",
 xlab = "Mass (grams)")
```

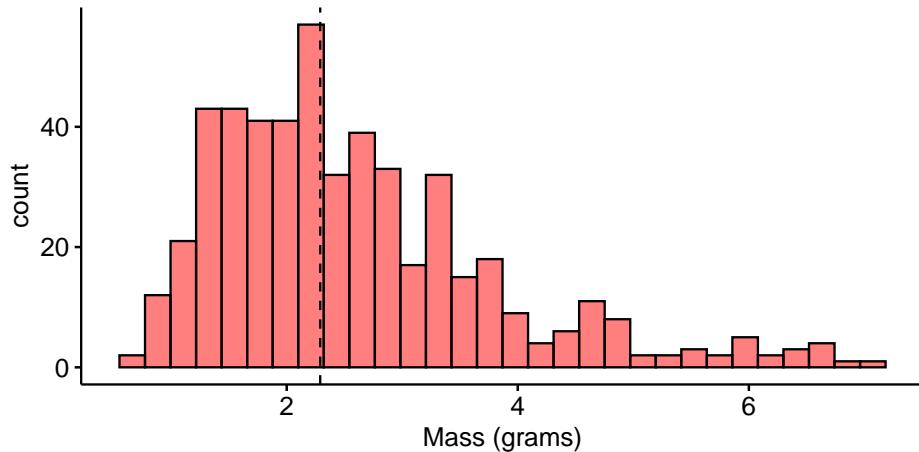


```
gghistogram(data = frogarms,
 x = "mass",
 fill = "red",
 xlab = "Mass (grams)",
 add = "mean")
```



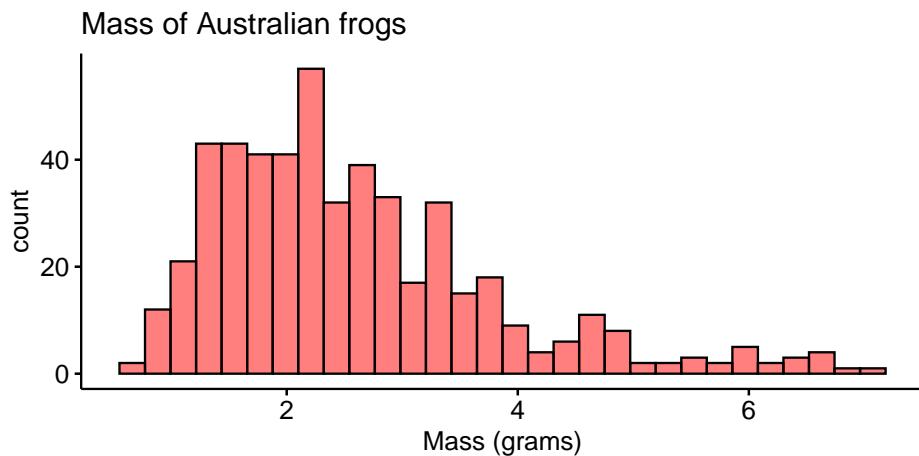
Warning messages: 1: Using `bins = 30` by default. Pick better value with the argument `bins`. 2: `geom_vline()`: Ignoring `mapping` because `xintercept` was provided. 3: `geom_vline()`: Ignoring `data` because `xintercept` was provided.

```
gghistogram(data = frogarms,
 x = "mass",
 fill = "red",
 xlab = "Mass (grams)",
 add = "median")
```



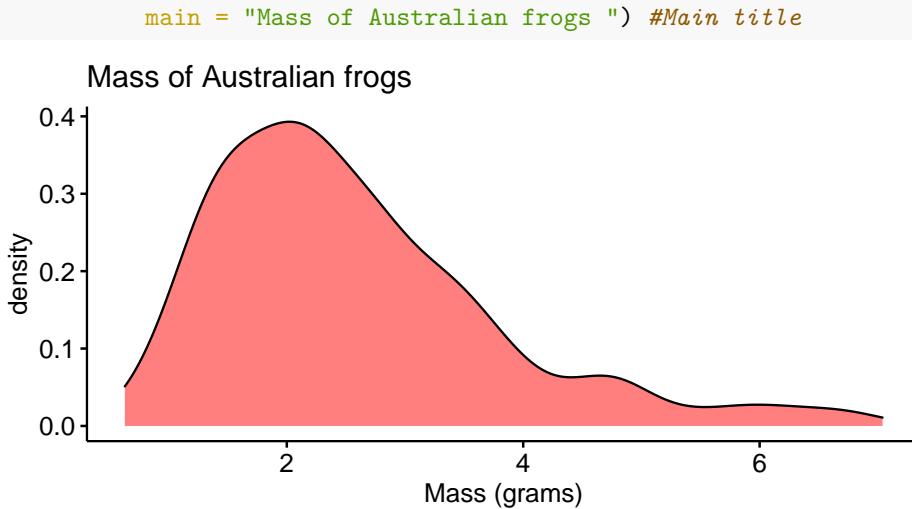
Add title not usually done for publication but useful for keeping track of things and for presentations

```
gghistogram(data = frogarms,
 x = "mass",
 fill = "red",
 xlab = "Mass (grams)",
 main = "Mass of Australian frogs ") #Main title
```



## 50.6 Density plot

```
ggdensity(data = frogarms,
 x = "mass",
 fill = "red",
 xlab = "Mass (grams)",
```



### 50.6.1 Plotting multiple plots with cowplot::plot\_grid

make into boxplot version exercise since shown first here as hist

```
my.frogs <- make_my_data2L(dat = frogarms,
 my.code = "nlb24", # #<= change this!
 cat.var = "sex",
 n.sample = 20,
 with.rep = FALSE)

Codes should only contain letters and numbers
Your special code is 4782969
(You don't really need to know this, though). 4782969
NOTE: This function only works properly for data with TWO levels to the categorical var.
eg male vs. female; it doesn't work for >2 levels (eg red vs blue vs. green)
dim(my.frogs)

[1] 40 8
```

We can save a plot to an R object

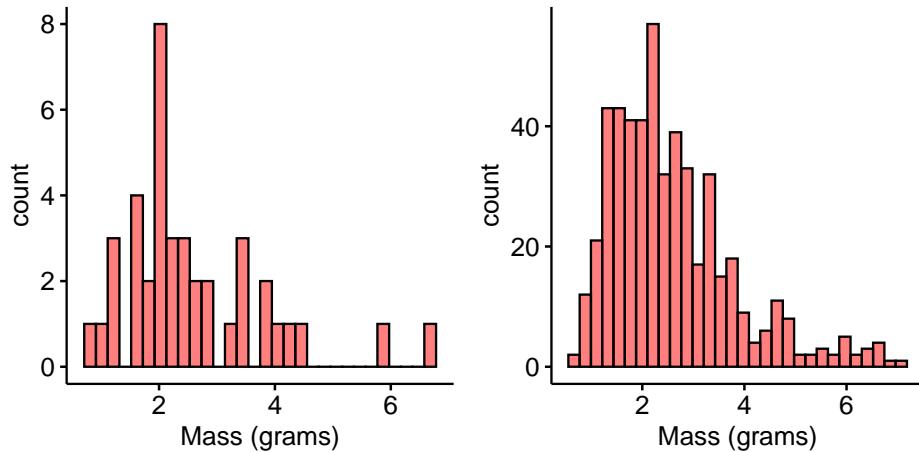
```
gg.hist.my.frogs <- gghistogram(data = my.frogs,
 x = "mass",
 fill = "red",
 xlab = "Mass (grams)") #Main title

gg.hist.frog <- gghistogram(data = frogarms,
 x = "mass",
```

```
fill = "red",
xlab = "Mass (grams)" #Main title
```

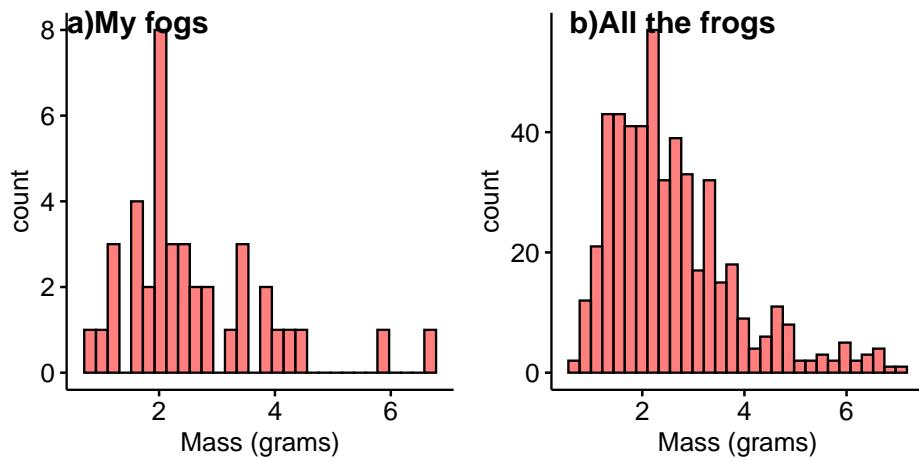
Now plot both

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog)
```



Add labels. Note that alignment is off sometimes.

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog,
 labels = c("a)My frogs","b)All the frogs"))
```



Add lines for means

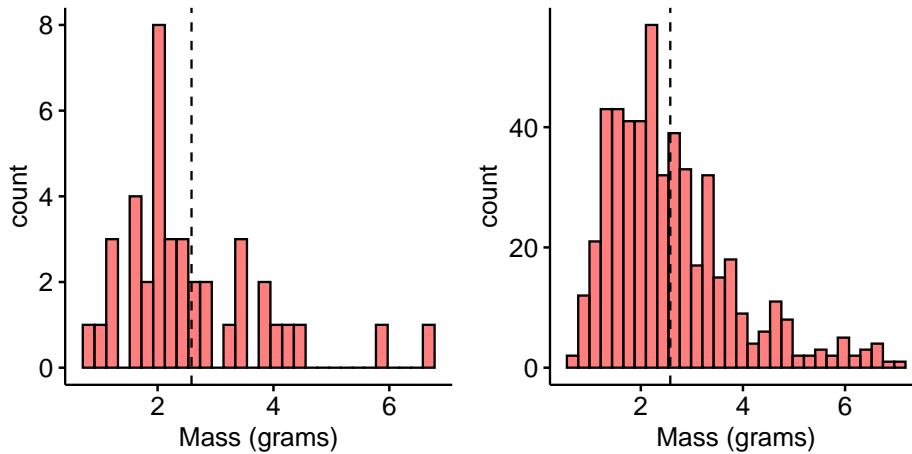
We can save a plot to an R object

```
gg.hist.my.frogs <- gghistogram(data = my.frogs,
 x = "mass",
 fill = "red",
 add = "mean",
 xlab = "Mass (grams)") #Main title

gg.hist.frog <- gghistogram(data = frogarms,
 x = "mass",
 fill = "red",
 add = "mean",
 xlab = "Mass (grams)") #Main title
```

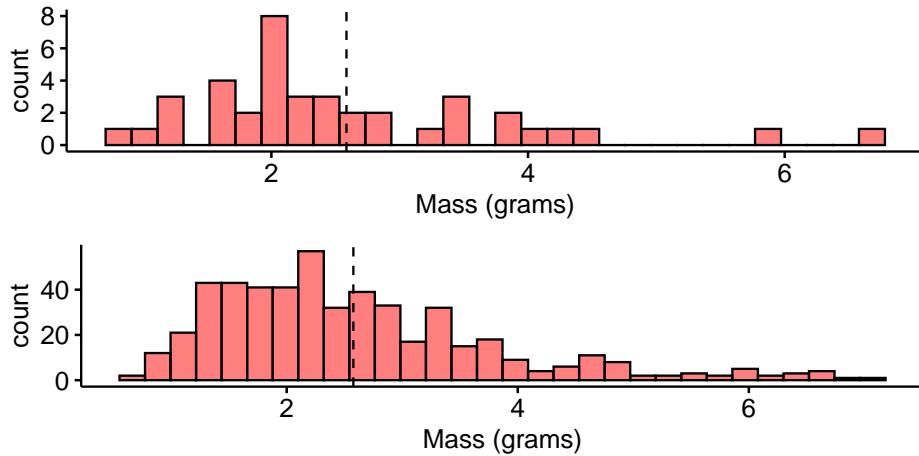
Now plot both

```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog)
```



## 50.7 Align vert

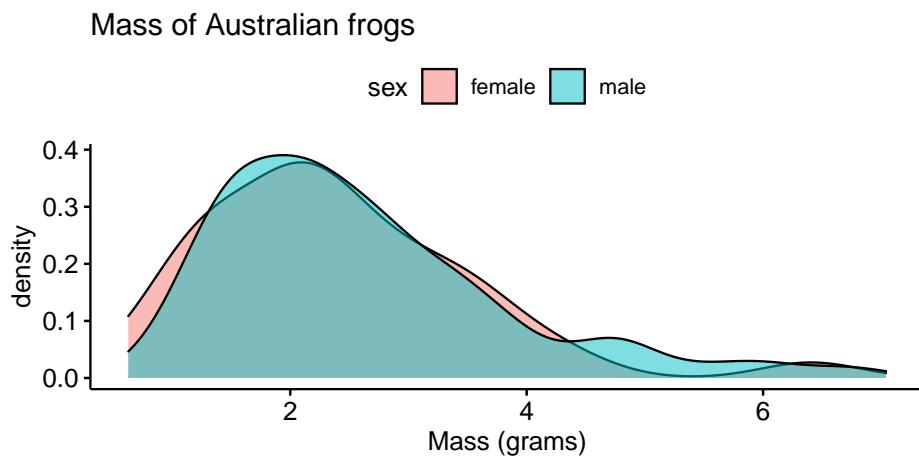
```
plot_grid(gg.hist.my.frogs,
 gg.hist.frog,
 nrow = 2)
```



## 50.8 Density plot overlay

## 50.9 Density plot

```
ggdensity(data = frogarms,
 x = "mass",
 fill = "sex",
 xlab = "Mass (grams)",
 main = "Mass of Australian frogs ") #Main title
```



# Chapter 51

## Frog arms

```
library(compbio4all)
```

### 51.1 Preliminaries

#### 51.1.1 Load packages

```
library(compbio4all)
library(ggplot2)
library(cowplot)
library(ggpubr)
library(dplyr)
```

#### 51.1.2 Load data

```
data(frogarms)
```

### 51.2 A 1st encounter with dplyr

dplyr is a package that provides numerous functions for manipulating data. We will use two handy functions

- summarize() / summarise()
- group\_by()

dplyr can use a handy syntax that invokes “pipes”. You can string together R commands using the function %>%

When using pipes, you start with a dataframe and follow it with an action you want done to it. So, for example, previously when we wanted the mean of the mass column we did this

```
mean(frogarms$mass)
```

Which is kind of read like a normal mathematical equation or function, where you start from inside the parentheses and work out. R lets you nest as many functions as you want. If I want to round my mean I wrap “mean(frogarms\$mass)” in round(...)

```
round(mean(frogarms$mass))
```

```
[1] 3
```

Using pipes to get the mean I write things more like a sentence:

```
frogarms$mass %>% mean() #note parentheses.
```

```
[1] 2.576994
```

Which reads kind of like “Take the mass column and the datagrame and apply the mean() function to it.” Note that the parentheses have to be included even though there is nothing in them.

To round the mean we would do this

```
frogarms$mass %>% mean() %>% round()
```

```
[1] 3
```

Which read left to right like a sentence is “Take the mass column, calculate the mean and then round it.”

Note that the round() command has an argument for how many digits you want to round to. You include that in the parentheses

```
frogarms$mass %>% mean() %>% round(digits = 2)
```

```
[1] 2.58
```

### 51.2.0.1 dplyr's summarize() command

Instead of mean(data\$column) we can use summarise()/summarize() and pipes

Grand mean of mass

```
frogarms %>% summarise(mean(mass))
```

```
mean(mass)
1 2.576994
```

this is maybe more complicated than “mean(frogarms\$mass)” but overall the pipe framework and summarise pays off when combined with group\_by()

### 51.3 group\_by

For some more info on group\_by see

<https://www.r-bloggers.com/using-r-quickly-calculating-summary-statistics-with-dplyr/> [https://www3.nd.edu/~steve/computing\\_with\\_data/24\\_dplyr/dplyr.html](https://www3.nd.edu/~steve/computing_with_data/24_dplyr/dplyr.html) <http://www.datacarpentry.org/R-genomics/04-dplyr.html>

We can use group\_by() to slit things up by a categorical variable. Here, we can say “take frogarms, split up the data by the sex column, and apply the mean function to each subset.”

```
frogarms %>%
 group_by(sex) %>%
 summarise(mean(mass))

A tibble: 2 x 2
sex `mean(mass)`
<fct> <dbl>
1 female 2.41
2 male 2.60
```

note that the column heading in is `mean(mass)`, which is what is in `summarise()`.

A handy thing about summarize is you can pass it labels Mean mass by sex w/ label

```
frogarms %>%
 group_by(sex) %>%
 summarise(mass.mean = mean(mass))

A tibble: 2 x 2
sex mass.mean
<fct> <dbl>
1 female 2.41
2 male 2.60
```

You can lable thigns anything, eg “puppies”.

```
frogarms %>%
 group_by(sex) %>%
 summarise(puppies = mean(mass))

A tibble: 2 x 2
sex puppies
<fct> <dbl>
1 female 2.41
2 male 2.60
```

You can pass any summarise function to summarise. We can give it sd to get the sd of mass by sex.

```
frogarms %>%
 group_by(sex) %>%
 summarise(mass.sd = sd(mass))

A tibble: 2 x 2
sex mass.sd
<fct> <dbl>
1 female 1.14
2 male 1.23
```

What makes dplyr::group\_by and summarize() really powerful is that you can pass it multiple summary functions at the same time

```
frogarms %>%
 group_by(sex) %>%
 summarise(mass.mean = mean(mass),
 mass.sd = sd(mass))

A tibble: 2 x 3
sex mass.mean mass.sd
<fct> <dbl> <dbl>
1 female 2.41 1.14
2 male 2.60 1.23
```

dplyr has a handy function n() for getting your sample size.

```
frogarms %>%
 group_by(sex) %>%
 summarise(mass.mean = mean(mass),
 mass.sd = sd(mass),
 n = n())

A tibble: 2 x 4
sex mass.mean mass.sd n
<fct> <dbl> <dbl> <int>
1 female 2.41 1.14 70
2 male 2.60 1.23 439
```

Pass it a novel function

make my\_sd1

```
frogarms %>%
 group_by(sex) %>%
 summarise(mass.mean = my_sd1(mass))
```

## 51.4 Alternatives

### 51.4.1 doBy::summaryBy

The doBy package has a nice syntax. I don't really see many people use it

```
library(doBy)
summaryBy(mass ~ sex, data = frogarms, FUN = mean)

summaryBy(mass ~ sex, data = frogarms, FUN = c(mean, sd))
```

### 51.4.2 tapply()

tapply is pretty old school

```
tapply(X = frogarms$mass, INDEX = frogarms$sex, FUN = mean)

female male
2.405714 2.604305
```

### 51.4.3 reshape2::dcast

What I've used most of my career thus far. Am slowly switch to dplyr.

```
library(reshape2)
dcast(data = frogarms,
 formula = sex ~ .,
 value.var = "mass",
 fun.aggregate = mean)

sex .
1 female 2.405714
2 male 2.604305
```

---

## 51.5 Your turn

The function make\_my\_data2L() will extact out a random subset of the data. Change “my.code” to your school email address, minus the “?” or whatever your affiliation is.

```
my.frogs <- make_my_data2L(dat = frogarms,
 my.code = "nlb24", # #<= change this!
 cat.var = "sex",
 n.sample = 20,
 with.rep = FALSE)

Codes should only contain letters and numbers
```

```
Your special code is 4782969
(You don't really need to know this, though). 4782969
NOTE: This function only works properly for data with TWO levels to the categorical
eg male vs. female; it doesn't work for >2 levels (eg red vs blue vs. green)
```

n.sample is set to 20. This is set up to extract 20 unique individuals of each sex.  
Check that your data frame is  $2 \times 20 = 40$  rows using the dim() command.

Assignment...

Can compare your subset to the original data

```
summary(frogarms$mass)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.630 1.690 2.290 2.577 3.210 7.040
```

Handy trick: stack up the data with rbind()

```
rbind(summary(my.frogs$mass),
 summary(frogarms$mass))
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
[1,] 0.81 1.9125 2.17 2.585250 3.325 6.68
[2,] 0.63 1.6900 2.29 2.576994 3.210 7.04
```

## Chapter 52

# Doing math for stats by hand: standard deviation

”\_hpc” stands for “Homo sapiens, protein-coding”

```
library(compbio4all)
data(genes_hpc)
```

Put gene lengths in vector

```
gene_lengths <- genes_hpc$length
```

### 52.1 Calcualting variance & standard deviation by hand, step by step

Variance and standard deviation are a fundamental quantity in statistics.

We'll step through each part of the calculation

### 52.2 “Deviations” between the mean and each observation

Calculate the difference between each observation and the mean of all observations that you calculated above.

Note that here, R is doing math on a set of 10 numbers in the object “genes\_hpc” and a single number, the mean, in “my.mean”

```
Yi.deviations <- gene_lengths-mean(gene_lengths)
```

### 52.3 Start making a dataframe (df)

We can keep track of the math by making a spreadsheet-like object in R called a matrix or dataframe using the cbind() command. cbind() means “column bind”

```
my.df <- cbind(gene_lengths, #original list of gene lengths
 Yi.deviations) #
```

Look at the matrix. Note that some of the deviations are positive and some are negative.

```
my.df
```

## Chapter 53

# Calculate the “Squared deviations” between the mean and each observation

Take your set of deviations and square them using “ $\wedge 2$ ”

Here, we've done math on a whole list of numbers at the same time.

```
Yi.deviations.square <- Yi.deviations^2
```

Squaring is key b/c it makes deviations greater than the mean and less than the mean equivalent. That is, we go from “deviations” that are positive and negative to “squared deviations” that are all negative.

Add the square deviations to the dataframe

```
my.df <- cbind(my.df, Yi.deviations.square)
```

Look at your expanding matrix

```
my.df
```

308 CHAPTER 53. CALCULATE THE “SQUARED DEVIATIONS” BETWEEN THE MEAN AND EA

## Chapter 54

# Sum of squares between the mean and each deviation

we can now calcualte the “sum of square deviations” or the “sum of squares”. This is the numerator (the thing on top) in the variance and standard deviation equations.

```
my.sum.of.squares<- sum(Yi.deviations.square)
```

This is a rather big number and I'm glad we don't have to calcualte it by hand : )

```
my.sum.of.squares
```



# Chapter 55

## And now...the variance

We can now use the sum of squares(SS) to calculate the variance. This is the SS divided by the sample size minus one. Recall that we calculated the sample size above and put it in an object called “my.N”. We subtract n to get what is known as the “Degrees of freedom”. More on this later

```
#The sample size
my.N <- length(gene_lengths)

#degrees of freedom
dfs <- my.N - 1

#The var
my.var <- my.sum.of.squares/dfs

#Could also do it more directly
my.var <- my.sum.of.squares/(my.N - 1)
```



# Chapter 56

## And now...the standard deviation

The standard deviation is just the square root of the variance.

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

```
my.sd <- sqrt(my.var)
```

We can check if our results are the same as R

```
#Variance
var(gene_lengths)

[1] 4337405
my.var

[1] 4337405
#stddev
sd(gene_lengths)

[1] 2082.644
my.sd

[1] 2082.644
```

We can check this also like this usig “==”

```
#Variance
var(gene_lengths) == my.var

[1] TRUE

#stdev
sd(gene_lengths) == my.sd

[1] TRUE
```

## Chapter 57

# The standard error (SE)

The standard error is a fundamental quantity in stats. It is used as a measure of variation in sample data and is useful for making error bars for means. It is the standard deviation divided by the square root of the sample size.

```
my.se <- my.sd/sqrt(my.N)
```



# Chapter 58

## R quirks: putting things in vectors

### 58.1 How R works with strings of numbers

### 58.2 How R works with strings of numbers

R has a somewhat strange behavior. Look at this example where we are trying to find the mean of 3 numbers, 0.86, 1.02, and 1.02.

It might be tempting to calculate the mean of these three numbers like this.

```
mean(0.86,1.02,1.02)
```

```
[1] 0.86
```

Notice, however, that the value that gets returned, 0.86, is exactly the same as the 1st value in the string of numbers. Hmmmm.... This is definitely not the mean.

Now try this: put the numbers into a “vector” using the functionc c()

```
y <- c(0.86,1.02,1.02)
```

We can check that the numbers are there

```
y
```

```
[1] 0.86 1.02 1.02
```

And we can get the mean using the mean command

```
mean(y)
```

```
[1] 0.9666667
```

This seems like a more reasonable answer, but since R seems to be acting weird we can check our answer another way

```
sum(y)/length(y)
```

```
[1] 0.9666667
```

Ok, seems ok. So whats wrong with the origin way, of just running “mean(0.86,1.02,1.02)” ?

In general, R functions carry out operations on **vectors**, or sets/strings of numbers that have been packaged together using the **c()** command or contained in the column of a dataframe or matrix. The function **mean()** is expecting to get a vector, not a list of raw data. Technically, what its doing is interpreting the value 0.86 as a vector, taking its mean (which is 0.86) and ingoring the rest of the numbers. Unfortuantely, it isn’t giving you a warning message.

### 58.3 Fixing the problem

We can get our oringal bit of code to work by just putting the data 0.86,1.02,1.02 into a vector using **c()**, like this

```
mean(c(0.86,1.02,1.02))
```

```
[1] 0.9666667
```

This is just a more direct way of doing the following, just using a single line of code instead of two.

```
y <- c(0.86,1.02,1.02)
mean(y)
```

```
[1] 0.9666667
```

# Chapter 59

## Summary stats for main PA eagle dataframe

```
library(compbio4all)
data("eaglesPA")
```

There are many commands for summary data in R, such as mean, median. However, you have to be careful about NAs!

```
mean(eaglesPA$eagles)
```

```
[1] NA
```

So, you asked for the mean of the eagles data, and you got NA. That's really annoying.

### 59.0.1 “NA” is a big deal in R

Try this

```
mean(eaglesPA$eagles, na.rm = T)
```

```
[1] 61.52632
```

“na.rm = T”, which means “na.rm = TRUE”, which means, “should I remove the NAs = yes, do it”

### 59.0.2 R also is VERY picky about upper vs. lower case

```
Mean(eaglesPA$eagles, na.rm = T)
```

```
Error in Mean(eaglesPA$eagles, na.rm = T): could not find function "Mean"
```

Note that the R error message is not very helpful : (

### 59.0.3 Most basic R commands are lower case

```
mean(eaglesPA$eagles, na.rm = T)
median(eaglesPA$eagles, na.rm = T)
min(eaglesPA$eagles, na.rm = T)
max(eaglesPA$eagles, na.rm = T)
summary(eaglesPA$eagles)
sd(eaglesPA$eagles, na.rm = T)
```

### 59.0.4 Standard error in R

The standard error (se) is a very common summary statistics but for some reason there is not a function for it in base R

#### 59.0.4.1 Calcualte the standard error by hand

Use the sd() command and the square root command sqrt

```
sd(eaglesPA$eagles, na.rm = T)/sqrt(15)
```

```
[1] 19.91098
```

# Chapter 60

## Variability

- `min()`
- `max()`
- `var()`
- `sd()`
- `range()`
- `nrow()` or `length()` (for sample size)

### 60.1 Preliminaries

#### 60.1.1 Load packages

```
library(compbio4all)
```

#### 60.1.2 Load data

```
data(frogarms)
```

`range()` returns two values in a vector

```
range(frogarms$mass)
```

```
[1] 0.63 7.04
```

Note that R doesn't return a very common statistic, the standard error (SE). This can be calculated by hand.

```
sd(frogarms$mass)/sqrt(length(frogarms$mass))
```

```
[1] 0.05419169
```

Write a function

```
my_sd1 <- function(dat_column){
 sd(dat_column)/sqrt(length(dat_column))
}

my_sd2 <- function(dat, column){
 sd(dat[,column])/sqrt(length(dat[,column]))
}

my_sd3 <- function(dat, column, digits.round = 3){
 se <- sd(dat[,column])/sqrt(length(dat[,column]))
 round(se, digits = digits.round)
}

my_sd2(dat = frogarms, column = "mass")
[1] 0.05419169
```

## **Chapter 61**

# **Understanding data helpfiles**



## Chapter 62

### P-values

```
library(compbio4all)
```



## **Chapter 63**

# **Introduction**

P-values are tricky. You need to memorize the precise definition of them. The notes below provide a definition and also some additional explanations. There is also code to simulate a scenario when the null hypothesis is true so you can see how often low p-values occur.



## Chapter 64

# P-value definition to memorize

You need to memorize the definition of a p-value. Here's a good version:

"the P-value is the probability of getting data as extreme as our data (just by chance) if the null hypothesis is, in fact, true. In other words, it is the [probability] that chance alone would produce data that differ from the null hypothesis." (Sadava et al, Statistics Primer)

A similar version:

"In many statistical analyses, we ask whether the null hypothesis of random variation among individuals can be rejected. The P-value is a guide to making that decisions. A statistical P-value measure the probability that observed or more extreme difference would be found *if the null hypothesis were true.* (Gotelli and Ellison pg 94, 1st ed)

Here's another version, this time emphasizing the situation that occurs in a t-test or similar statistical procedure:

The p-value is the Probability of observing your data comparing two groups (OR data where the difference between groups is even greater) IF there was actually no underlying difference between groups and the difference between the groups you observed is just due to chance.



## **Chapter 65**

### **P-values are tricky**

“Many statistical analyses are reported with P values... P values are often misunderstood because they answer a question you probably never thought to ask.”  
Motulsky 2017, pg 129.



## Chapter 66

### P-value example: coin flipping

“You flip a coin 20 times and observed 16 heads and 4 tails. Because the probability of heads is 50%, you’d expect 10 heads in 20 flips. How unlikely is it to find 16 heads in 20 flips? Should you suspect that the coin is unfair?” (page 129) ... “The P value answers this question: If the coin tosses were random and the answers were recorded correctly, what is the chance that when you flip a coin 20 times, you’ll observed results as extreme (or more extreme) than those you observed... - that is, that you would observed 16 or more, or 4 or fewer, heads? The answer, the P value, is 0.0118. You’d only see results this far from an even split of heads to tails in 1.18% of 20-coin runs” (“20 coin runs” = sets of 20 coin flips. If I have everyone in the class flip a coin 20 time and take data, each persons data would be a “20-coin run”. If I had a class of 200 students, a p value of 0.0118 means that out of 200 students, 1 would probably get at least 16 heads) (page 131)”

We we flip a coin we know the null hypothesis is almost definitely true, or we can easily check a coin to see if it might be damaged or biased. Usually in science we are assuming (or hoping) that the null hypothesis is false, and the p-value calculation requires the odd mental trick of viewing things from the perspective of the null.



## **Chapter 67**

### **P-value example 1**

“Consider a situation where investigators compared a mean in groups given two alternative treatments and reported a p-value of 0.03.” A correct interpretation of this p-value is “If the population means are identical (the null hypothesis is true), there is a 3% chance of observing a difference as large as you observed (or larger)” (Motulsky pg 139).



## Chapter 68

### P-value example 2: heights of elementary school students

Compare height of 10 randomly chosen 1st graders from Ms. Bs class to 10 randomly chosen 1st graders in Mr. W's class. Mr. W's class is on average average 10 cm taller. The p value 0.10. This means that, if there was actually no difference in the height of 1st graders (Which is likely to be true), the probability of getting a 10 cm difference in heights is 0.10. This means that if you repeated this experiment in 10 schools where there was no difference in height between classes, you'd expect to still see a 10 cm difference 1 in 10 times.



## Chapter 69

# P-values and biological sequences

“The p value is the probability of a chance alignment occurring with the score in question or better.” (Pevner Chapter 4, pg 143). “A p value below 0.05 is traditionally used to define statistical significance (i.e., to reject the null hypothesis that your query sequence is not related to any database sequence). If the null hypothesis is true, then 5% of all random alignments will result in an apparently significant score. An [P] value of 0.05 or less may therefore be considered significant.” (Page 144). **I don't 110% like this statement but its close enough.**



# Chapter 70

## Inference-by-eye using R.A Fisher's Cat Data

A an example of doing “inference by eye” using R.A. Fischer’s cat data. A good summary of this idea is Cummings et al. 2007. Error bars in experimental biology. <http://jcb.rupress.org/content/177/1/7.short>

### 70.1 Load the data

```
library(MASS)
data(cats)
```

### 70.2 Look at the data

```
dim(cats)
summary(cats)
```

### 70.3 Summarize the body weight (Bwt) data old-school using `summaryBy()`

A more modern way would be to use `dplyr()`

```
library(doBy)

#get the mean and SD
cat.df1 <- summaryBy(Bwt ~ Sex, data = cats, FUN = c(mean, sd))
```

```
#get the sample size using length()
cat.df2 <- summaryBy(Bwt ~ Sex, data = cats, FUN = c(length))

#make a combined dataframe
cat.df3 <- merge(cat.df1,cat.df2)

#calculate the standard error SE by hand

cat.df3$SE <- cat.df3$Bwt.sd/sqrt(cat.df3$Bwt.length)
```

Look at the results

```
cat.df3
```

## 70.4 Plot the data

### 70.4.1 Visualize the raw data

```
par(mfrow = c(1,1),mar = c(3,3.5,1,1))

boxplot(Bwt ~ Sex, data = cats)
```

### 70.4.2 Plot the means with error bars

This uses the errbar() function. A modern contemporary way would use ggplot2 and possibly its extension using ggpibr.

#### 70.4.2.1 The real data

This is the actual data. The 95% confidence intervals do not overlap, which indicates that the p-value for the t-test will be less than 0.05.

```
library(Hmisc)
par(mfrow = c(1,2),mar = c(3,3.5,1,1))

y.lim <- c(2.295,3)
errbar(1:2,
 y = cat.df3$Bwt.mean,
 yplus = cat.df3$Bwt.mean + cat.df3$SE,
 yminus = cat.df3$Bwt.mean - cat.df3$SE,
 xlab = "",
 ylab = "",
 xlim=c(0.5,2.5),
 ylim = y.lim,
 xaxt="n",cex =1)
axis(side=1,at=1:2,labels=cat.df3$Sex)
```

```

mtext("Sex", side = 1, line = 2, cex = 2)
mtext("Mass (g)", side = 2, line = 2.1, cex = 1.3)

errbar(1:2,
 y = cat.df3$Bwt.mean,
 yplus = cat.df3$Bwt.mean + 1.96*cat.df3$SE,
 yminus = cat.df3$Bwt.mean-1.96*cat.df3$SE,
 xlab = "",
 ylab = "",
 xlim=c(0.5,2.5),
 ylim = y.lim,
 xaxt="n",cex =1)
axis(side=1,at=1:2,labels=cat.df3$Sex)
mtext("Sex", side = 1, line = 2, cex = 2)
mtext("Mass (g)", side = 2, line = 2.1, cex = 1.3)

```

#### 70.4.2.2 Modified data with a non-significant different

Make an alternative version of the data where there isn't a difference between the male and female cats

```

cat.df3.mod <- cat.df3
cat.df3.mod$Bwt.mean[1] <- cat.df3$Bwt.mean[2]-cat.df3$Bwt.mean[2]*0.0425

```

The overlap of the error bars here is greater than 1/2 the length of the bar; therefore the p-value for a t-test will be  $> 0.05$ .

```

y.lim <- c(2.6975,3)
par(mar = c(3,3.5,1,1))
errbar(1:2,
 y = cat.df3.mod$Bwt.mean,
 yplus = cat.df3.mod$Bwt.mean + cat.df3.mod$SE,
 yminus = cat.df3.mod$Bwt.mean-cat.df3.mod$SE,
 xlab = "",
 ylab = "",
 xlim=c(0.5,2.5),
 ylim = y.lim,
 xaxt="n",cex =1)
axis(side=1,at=1:2,labels=cat.df3.mod$Sex)
mtext("Sex", side = 1, line = 2, cex = 2)
mtext("Mass (g)", side = 2, line = 2.1, cex = 1.3)

errbar(1:2,
 y = cat.df3.mod$Bwt.mean,
 yplus = cat.df3.mod$Bwt.mean + 1.96*cat.df3.mod$SE,
 yminus = cat.df3.mod$Bwt.mean-1.96*cat.df3.mod$SE,

```

```
 xlab = "",
 ylab = "",
 xlim=c(0.5,2.5),
 ylim = y.lim,
 xaxt="n",cex =1)
axis(side=1,at=1:2,labels=cat.df3.mod$Sex)
mtext("Sex", side = 1, line = 2, cex = 2)
mtext("Mass (g)", side = 2, line = 2.1, cex = 1.3)
```

## 70.5 T-test

A t-test for the differenec between female and male cats.

```
t.test(Bwt ~ Sex, data = cats)
summary(lm(Bwt ~ -1+Sex, data = cats))
```

## Chapter 71

# Bootstrapping Tutorial: Fst from Evans et al 2016



# Chapter 72

## Fst data

### 72.1 Load Fst into vector

```
Fstperloc <- c(0.8589462,0.4133128,0.6851604,0.7366967,0.7925415,0.7161766,0.4683899,0.8355689)
```

### 72.2 Mean Fst

```
Fst.mean <- mean(Fstperloc)
Fst.mean
[1] 0.6883491
```

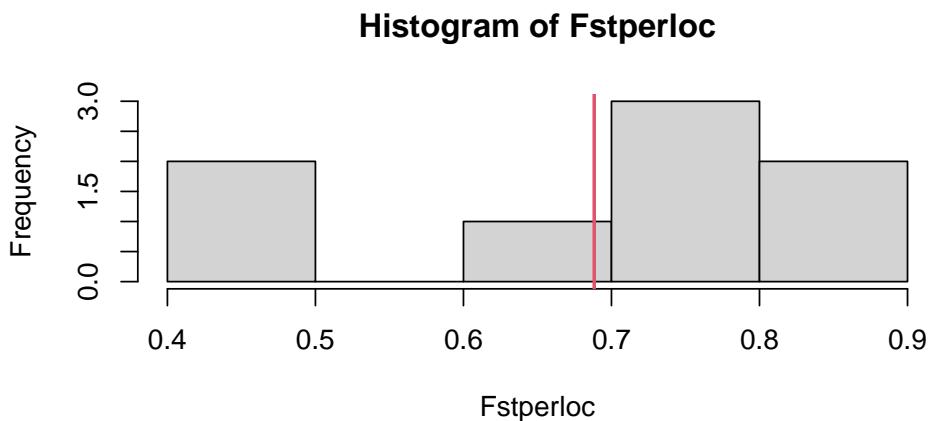
### 72.3 Distribution of Fst values

- 2 Fst values lower than the rest
- Non-normality is a good reason to use bootstrapping

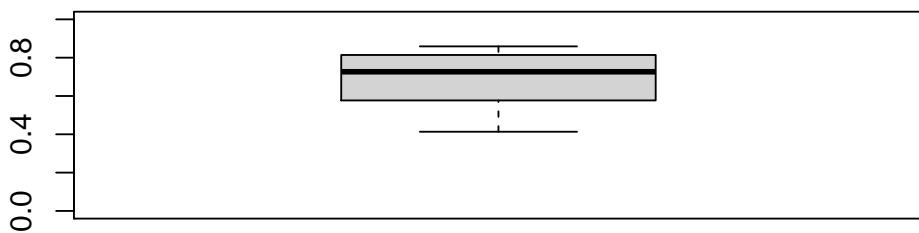
#### 72.3.1 Histogram base R

```
#histogram of Fst
hist(Fstperloc)

#vertical line for mean
abline(v =Fst.mean, col = 2, lwd = 2)
```



```
#boxplot
boxplot(Fstperloc,
 ylim = c(0,1))
```

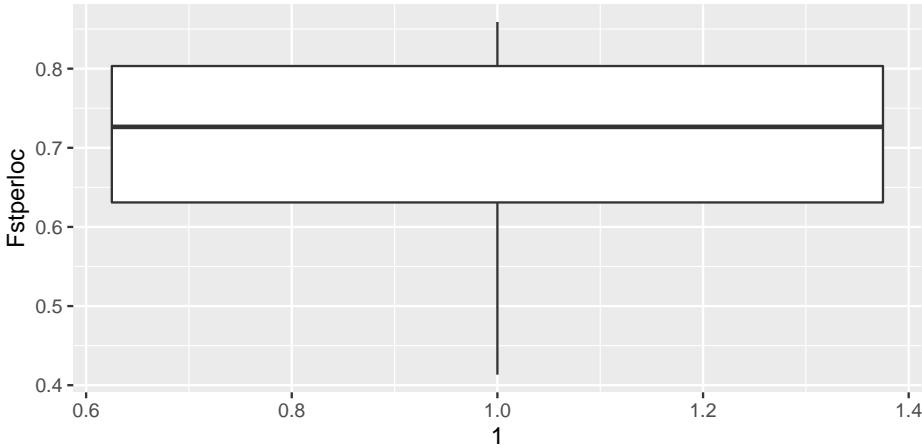


### 72.3.3 Boxplot ggplot

Just the boxplot

```
library(ggplot2)

qplot(y = Fstperloc,
 x = 1,
 geom = "boxplot")
```

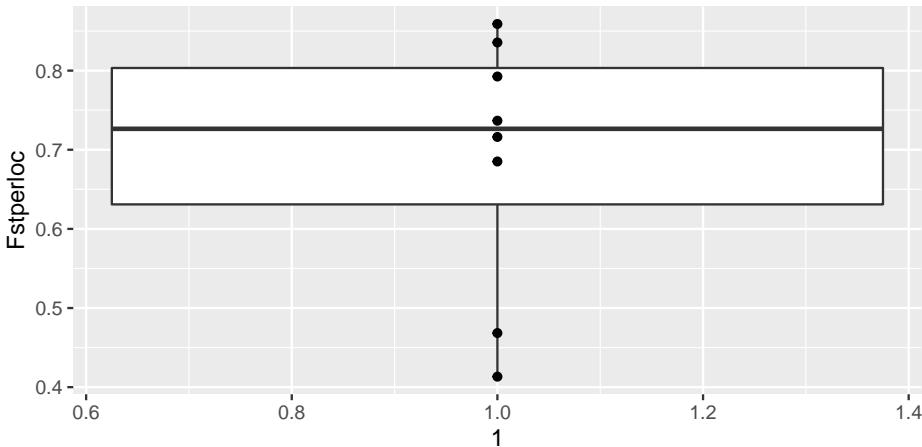


Boxplot w/ raw data overlayed

- Set 2 “geoms” via “geom = c(“boxplot”, “point”)

```
library(ggplot2)

qplot(y = Fstperloc,
 x = 1,
 geom = c("boxplot",
 "point"))
```





# Chapter 73

## Bootstrapping Fst

Boostrapping with boot requires

- Defining a function for the desired summary statistic that is used by the `boot()` function
  - Want the mean and its confidence intervals
  - A tricky part about `boot()` is that you have to write a full R function
  - This requires a fair bit code for somethings as simple as the mean
- Extracting the desired confidence interval with `boot.ci()`

Load the package

```
#load package
library(boot)
```

### 73.1 Define function for the mean

- Function synatax takes a particualr form in R
- Here is the R code define a bran new function `mean.fun()` function that will take 2 arguements
  - `dat`: our raw data
  - `idx`: an index of resampled data
- The original data has 8 Fst values
- What `boot()` does is generate 8 random index (`idx`) values from the set `c(1,2,3,4,5,6,7,8)`
  - Sampling is with replacement
- “`dat[idx]`” calls up those index values
  - Note, there can be duplicate index values b/c of does sampling w/replacement
- The `mean.fun()` takes the mean of the data matching those index values
- This is repeatd many many times to build a sampling distribution

### 73.1.1 The function mean.fun()

```
mean.fun <- function(dat, idx) {mean(dat[idx], na.rm=TRUE)}
```

We can test the function

```
normal R mean function
mean(Fstperloc)
```

```
[1] 0.6883491
#mean.fun()
mean.fun(Fstperloc, idx = c(1:8))
```

```
[1] 0.6883491
#An arbitay set of idx values
mean.fun(Fstperloc, idx = c(1,1,2,4,5,2,8,7))
```

```
[1] 0.6722144
```

R has function for generating random numbers. We can generate numbers from a uniform distribution from 1 to 8 like this using runif() for “random uniform”

```
runif(n = 8,min = 1, max = 8)
```

```
[1] 7.159470 4.422562 1.819271 1.121999 6.335273 6.608221 3.382114 7.324451
```

We can make these whole numbers by rounding them

```
round(runif(n = 8,min = 1, max = 8),0)
```

```
[1] 1 4 7 8 3 7 4 5
```

## 73.2 Running boot

boot() takes 3 arguments

- The vector of raw data, Fstperloc
- mean.fun, the function for the summary statistic
- The number of replicates R

### 73.2.1 Run the function

Run boot()

```
bootfst = boot(Fstperloc, mean.fun, R=1000)
```

The raw output of boot() isn’t very interesting; it just gives you the original mean

```
bootfst

##
ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
Call:
boot(data = Fstperloc, statistic = mean.fun, R = 1000)
##
##
Bootstrap Statistics :
original bias std. error
t1* 0.6883491 -0.00182523 0.05372295

boot.ci() extracts the confidence interval
boot.ci(bootfst, type=c("norm"))
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates
##
CALL :
boot.ci(boot.out = bootfst, type = c("norm"))
##
Intervals :
Level Normal
95% (0.5849, 0.7955)
Calculations and Intervals on Original Scale
```

There are numerous types of bootstrap confidence intervals: “norm”, “basic”, “stud”, “perc”, “bca”. You can get multiples like this

```
boot.ci(bootfst, type=c("norm",
 "basic",
 "perc",
 "bca"))

BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates
##
CALL :
boot.ci(boot.out = bootfst, type = c("norm", "basic", "perc",
"bca"))
##
Intervals :
Level Normal Basic
95% (0.5849, 0.7955) (0.5896, 0.7942)
```

```
Level Percentile BCa
95% (0.5825, 0.7871) (0.5720, 0.7773)
Calculations and Intervals on Original Scale
```

# Chapter 74

## Compare Fst to Qst

Make a dataframe with Fst info and Qst

- All values reported in text

```
Fst.vs.Qst <- data.frame(Statistic = c("Fst", "Qst"),
 Mean = c(0.6883491, 1.0),
 ci.lo = c(0.556, 0.175),
 ci.hi = c(0.772, 1.0))
```

### 74.1 Plot with ggplot

I do a little trick here to plot the raw data next to Fst

#### 74.1.1 Generate Jittered locations

- I use rnorm to generate random x axis values centered on 1.12.
- Mean Fst and its CI will get plotted at the 1.0 location on the x axis, though it will be labeled w/“Fst”, not 1.0
- This generates jittered points

```
Fstperloc.2 <- data.frame(Fstperloc = Fstperloc,
 y = rnorm(length(Fstperloc), 1.12, 0.015))
```

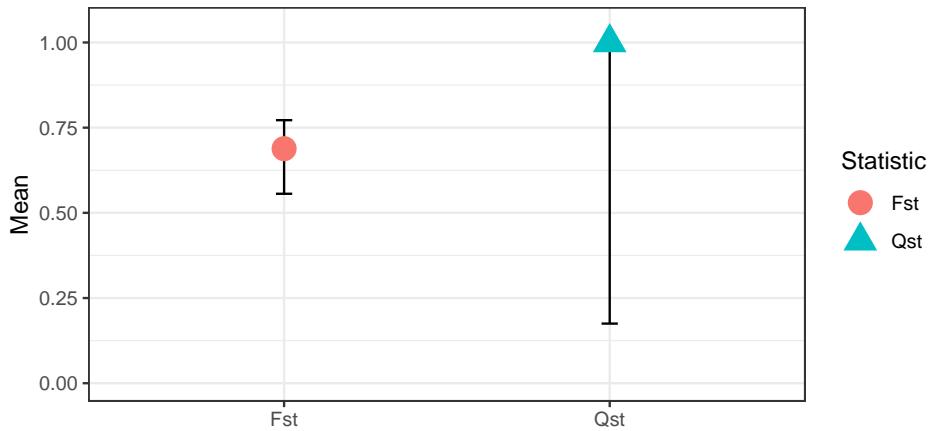
##### 74.1.1.1 Plot data w/ jittered point

```
library(ggplot2)
qplot(y = Mean,
 x = Statistic,
 data = Fst.vs.Qst) +
 geom_errorbar(aes(ymin = ci.lo, ymax = ci.hi),
```

```

 width = 0.05) +
geom_point(aes(color = Statistic,
 shape = Statistic),
 size = 5) +
theme_bw() +
ylim(0,1.05) +
xlab("")

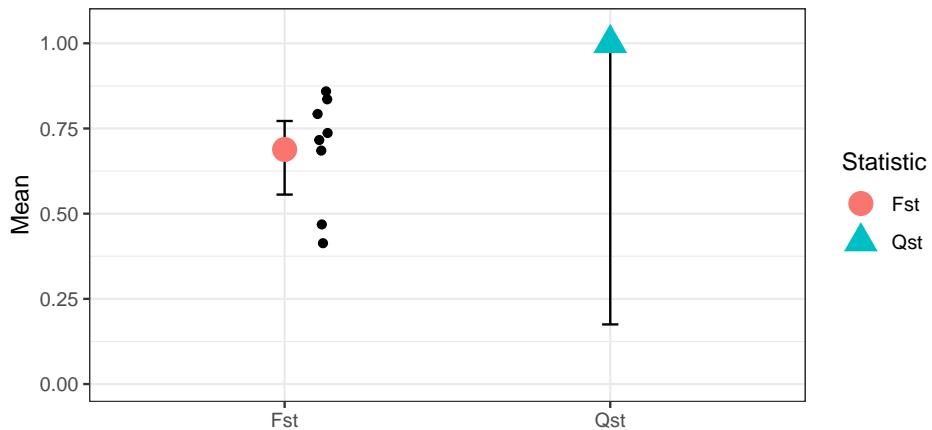
```



```

library(ggplot2)
qplot(y = Mean,
 x = Statistic,
 data = Fst.vs.Qst) +
 geom_errorbar(aes(ymin = ci.lo, ymax = ci.hi),
 width = 0.05) +
 geom_point(aes(color = Statistic,
 shape = Statistic),
 size = 5) +
 theme_bw() +
 geom_point(data = Fstperloc.2,
 aes(y = Fstperloc, x = y)) +
 ylim(0,1.05) +
 xlab("")

```



```
library(ggplot2)
```



# Chapter 75

## From the original text:

Under the heading: “GENETIC BASIS OF SINIGRIN VARIATION” The main conclusions “Genetic variation in sinigrin was partitioned primarily among, rather than within, populations, with c. 9% of total phenotypic variation attributable to population (Table S4) and 0% attributable to grandmaternal or selfed family. The resulting extreme estimate of QST (1,0.175-1, 95% CI) is not unexpected in a largely selfing species which likely establishes new populations from a small number of colonizing genotypes [citation?]. Populations were also highly diverged in neutral markers, with an overall FST of 0.688 (0.556-0.772, 95% CI).”

Continuing “Nonetheless, each population contained multiple multilocus genotypes (MLGT) (ranging from 0.63 to 2.83 multilocus genotype in offspring per field maternal family across populations), so the low within-population quantitative genetic variation is not solely due to sampling a single selfed lineage. This finding is at odds with our previous work suggesting a rapid evolution of sinigrin concentrations, which should require high levels of intrapopulation genetic variation on which to act (Lankau et al. 2009). However, the levels of intrapopulation genetic variation are likely to vary across the introduced range.””

### 75.1 Doing math for stats by hand

```
library(compbio4all)
data(genes_hpc)
```

Put gene lengths in vector

```
gene_lengths <- genes_hpc$length
```

Make subset so that things can all be seeing on-screen Then do for whole vector Assignment: do for GC Follow up: data manipulation with dplyr: 1)2 groups:

mt vs rest of genome 2)3 groups: mt vs. regular vs. sex 3)may groups: by chromosome number

### 75.1.1 Calculating the mean gene length by hand

#### 75.1.1.1 The numerator

```
length.sum <- sum(gene_lengths)
```

#### 75.1.2 The denomintor

```
N <- length(gene_lengths)
```

#### 75.1.3 The mean

```
length_mean <- length.sum/N
```

Same thing to get the mean, just in 1 step

```
sum(gene_lengths)/length(gene_lengths)
```

```
[1] 1907.157
```

We can check our answer like this using the “==” function which asks R “are these two objects EXACTLY the same?”

```
length_mean == mean(gene_lengths)
```

```
[1] TRUE
```

We can do the same thing to check that the standard deviation is indeed the square root of the mean

```
sd(gene_lengths) == sqrt(var(gene_lengths))
```

```
[1] TRUE
```

If the were not the same, R would say “FALSE”

- Because R is very very very precise any rounding errors will result in R saying that two things are NOT the same. Sometimes some R functions do do some rounding so you have to check “FALSE” sometimes.

## Chapter 76

# Doing Math in R

- R can do everything a scientific calculator or spreadsheet can do. This includes basic functions like +, - , /, sqrt, etc, and basic functions like mean(), median(), sd() for the standard deviation and var() for the variance.
- In general in this course we will focus on having do R most of the calculations for our stats. However, its important to understand some of the underlying math.
- Today, to practice doing math in R and to learn about basic statsitical functions, we'll do calcualtions of the mean, variance, "sum of squares", and standard devition "by hand" in R. We'll then compare them to the output R produces.

```
library(compbio4all)
data(genes_hpc)
```

Put gene lengths in vector

```
gene_lengths <- genes_hpc$length
```



# Chapter 77

## Introduction

- The data we'll use for this is discussed in Chapter 4 of Whitlock & Schluter. The author's accessed information on the human genome about the length of every gene that has been identified. This results in a list of 175579 genes and their length in DNA bases pairs (A, T, C and Gs).
- These genes vary in length from  $\infty$  bp to  $-\infty$
- mean = NA bp
- median = bp.

### 77.1 The basic math stuff in R

The summed length of all the genes (not meaninful on its own)

```
sum(gene_lengths)
```

```
[1] 334856778
```

The mean gene length

```
mean(gene_lengths)
```

```
[1] 1907.157
```

```
min(gene_lengths)
```

```
[1] 19
```

```
max(gene_lengths)
```

```
[1] 109224
```

The variance (var)

```
var(gene_lengths)
[1] 4337405
The standard deviation (sd)
sd(gene_lengths)
```

```
[1] 2082.644
```

Note that the standard deviation is just the square root of the variance

```
#SD using R's function
sd(gene_lengths)
```

```
[1] 2082.644
```

#SD as the sqrt of the variance

```
sqrt(var(gene_lengths))
```

```
[1] 2082.644
```

The min, max, etc.

```
min(gene_lengths)
```

```
[1] 19
```

```
max(gene_lengths)
```

```
[1] 109224
```

```
sum(gene_lengths)
```

```
[1] 334856778
```

# Chapter 78

## T-test - deprecated? newer version?

### 78.1 Preliminaries

#### 78.1.1 Load packages

```
library(compbio4all)
library(ggplot2)
library(cowplot)
library(ggpubr)
library(dplyr)
```

#### 78.1.2 Load data

```
data(frogarms)
```

#### 78.1.3 Subset your data

The function `make_my_data2L()` will extract out a random subset of the data. Change “my.code” to your school email address, minus the “?” or whatever your affiliation is.

```
my.frogs <- make_my_data2L(dat = frogarms,
 my.code = "nlb24", #<= change this!
 cat.var = "sex",
 n.sample = 20,
 with.rep = FALSE)
```

```

Codes should only contain letters and numbers
Your special code is 4782969
(You don't really need to know this, though). 4782969
NOTE: This function only works properly for data with TWO levels to the categorical
eg male vs. female; it doesn't work for >2 levels (eg red vs blue vs. green)

t-test used to tell if two groups are different

```

## 78.2 T-test

```

##
Welch Two Sample t-test
##
data: mass by sex
t = -2.018, df = 34.868, p-value = 0.05134
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-1.535699558 0.004699558
sample estimates:
mean in group female mean in group male
2.2025 2.9680

spits out R's standard t.test table

Save to object
mass.t <- t.test(mass ~ sex, data = my.frogs)

```

look at w/broom::glance. Relables thiugns a bit odd and would be nice to round.  
will stick with original R output

```

library(broom)
glance(mass.t)

A tibble: 1 x 10
estimate estimate1 estimate2 statistic p.value parameter conf.low conf.high
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 -0.765 2.20 2.97 -2.02 0.0513 34.9 -1.54 0.00470
... with 2 more variables: method <chr>, alternative <chr>

```

```

mass.t

##
Welch Two Sample t-test
##
data: mass by sex

```

```
t = -2.018, df = 34.868, p-value = 0.05134
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-1.535699558 0.004699558
sample estimates:
mean in group female mean in group male
2.2025 2.9680
```

Check the means using dplyr

```
my.frogs %>% group_by(sex) %>% summarize(mean.mass = mean(mass))

A tibble: 2 x 2
sex mean.mass
<fct> <dbl>
1 female 2.20
2 male 2.97
```

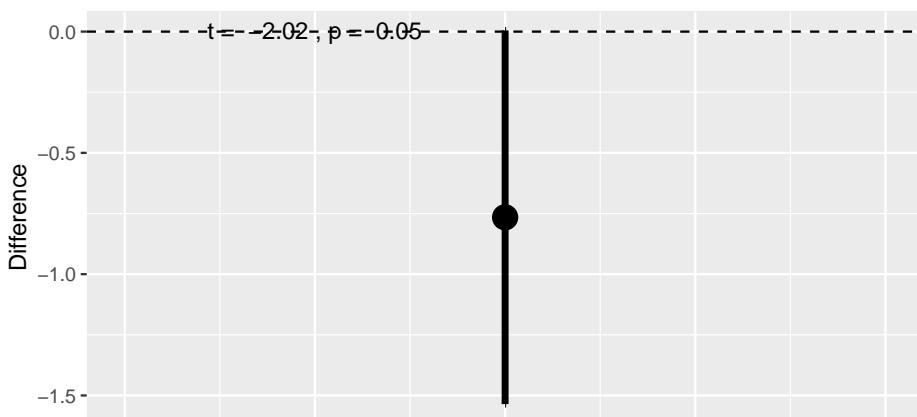
What does all of this mean?

Quiz

$p = p$  interpretation =  $df =$  why  $df$  fractional? (this one is hard!)  $t =$  What would happen to  $p$  if  $t$  was bigger? What is a “95% CI” What is this a 95% CI for?

What does the CI mean?

```
plot_t_test_ES(mass.t)
```



Make a plot of the means with error bars. Save to an object called gg.means

```
gg.means <- ggerrorplot(data = my.frogs,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci") +
```

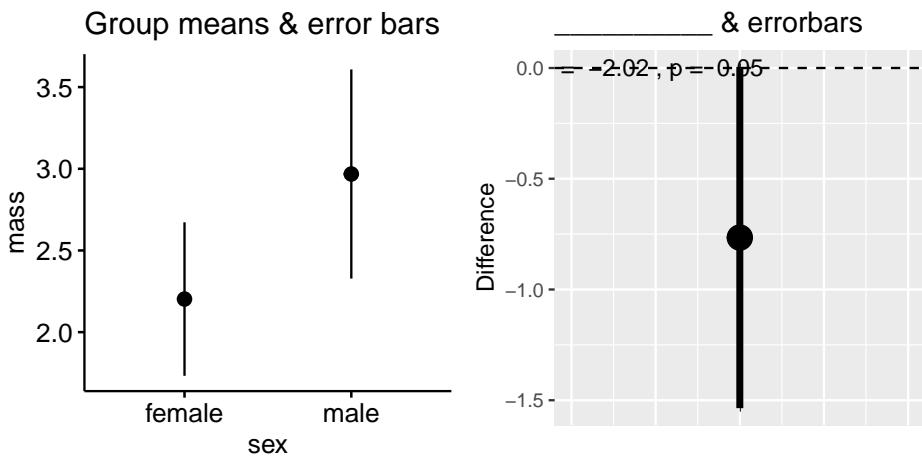
```
ggttitle("Group means & error bars")
```

Save effect size

```
gg.ES <-plot_t_test_ES(mass.t) +
 ggttitle("----- & errorbars")
```

Plot both

```
plot_grid(gg.means, gg.ES)
```



A hint

```
gg.means <-ggerrorplot(data = my.frogs,
 y = "mass",
 x = "sex",
 desc_stat = "mean_ci") +
 ylab("Mass (g)") +
 xlab("Sex") +
 ggttitle("Group means & error bars")

gg.ES <-plot_t_test_ES(mass.t) +
 ylab("Difference between groups (g)") +
 ggttitle("----- & errorbars")

plot_grid(gg.means, gg.ES)
```



Did anyone get a significant result?

### 78.3 Arm girth

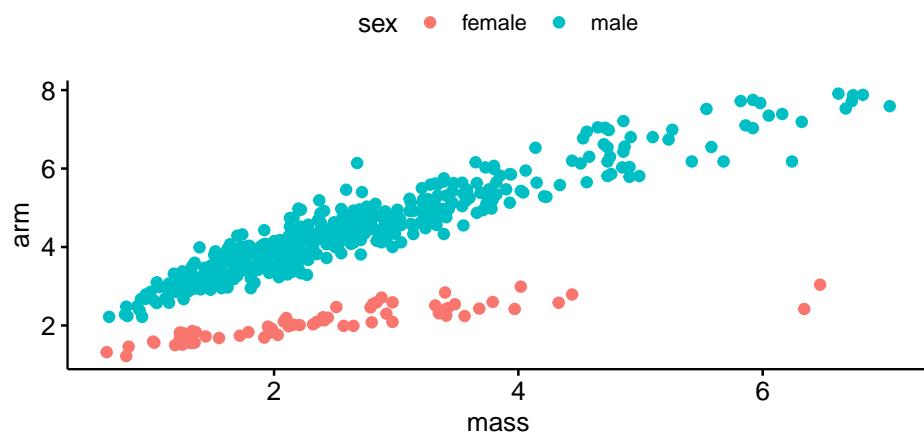
(do same thing for arm girth. see the super significant values?)

Now we are going to unpack this

There is a **major** flaw in this analysis. consider the following graph where the mass is plotted on the x-axis and the arm girth is plotted on the y-axis. Does arm girth vary just because of sex, or because of sex and mass?

This is ANCOVA. Sometimes taught as an extension of ANOVA, or as a type of regression.

```
ggscatter(data = frogarms,
 y = "arm",
 x = "mass",
 color = "sex")
```



## **Chapter 79**

### **3D plotting**



## **Chapter 80**

# **Coding multivariate data - color and shape**



## **Chapter 81**

### **Facets**



# Chapter 82

## Multivariate Data Analysis: Making Sense of High-Dimensional Data w/PCA etc

```
library(compbio4all)
```

### 82.1 Preliminaries

#### 82.1.1 Packages

```
library(vegan)
#library(ggpubr)
```

#### 82.1.2 Data sets

```
data(BCI)
data(BCI.env)
```

#### 82.1.2.1 Vocab / Functions

grep(), gsub(), multivariate data, high-dimensional data, PCA, principle components

## 82.2 High-dimensional Data in Biology

### 82.2.1 Morphology: How do species vary

Morphology: numeric data

#### 82.2.1.1 3 Iris species

```
summary(iris$Species)
```

```
setosa versicolor virginica
50 50 50
```

4 morphological measurements; each is a dimension of data

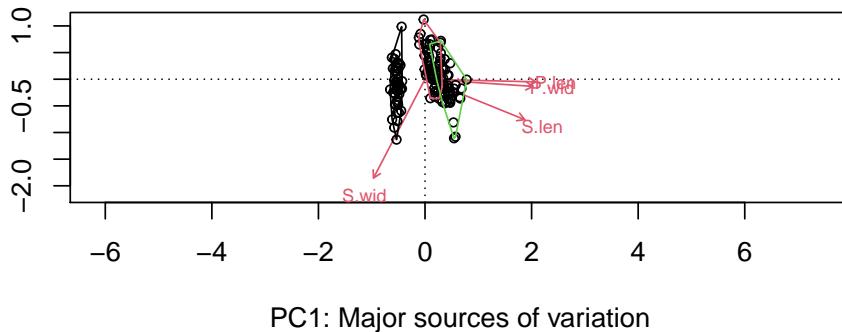
```
summary(iris[,-5])
```

```
Sepal.Length Sepal.Width Petal.Length Petal.Width
Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300
Median :5.800 Median :3.000 Median :4.350 Median :1.300
Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500
names(iris) <- gsub("Sepal", "S", names(iris))
names(iris) <- gsub("Petal", "P", names(iris))
names(iris) <- gsub("Length", "len", names(iris))
names(iris) <- gsub("Width", "wid", names(iris))
```

#### 82.2.1.2 PCA

```
par(mfrow = c(1,1))
PC doesn't like categorical variables
iris.pca <- vegan::rda(iris[,-5], scale = TRUE)
biplot(iris.pca,
 display = c("sites",
 "species"),
 type = c("text",
 "points"),
 xlab = "PC1: Major sources of variation",
 ylab = "PC2: Secondary sources of variation")
ordihull(iris.pca,
 group = iris$Species,
 col = c(1,2,3))
```

PC2: Secondary sources of variation



```

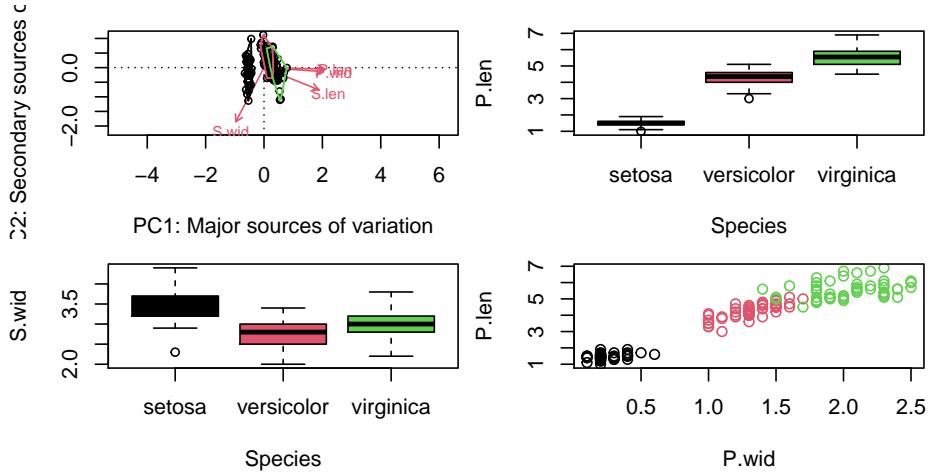
par(mfrow = c(2,2), mar = c(4,4,1,0.5))
biplot(iris.pca,
 display = c("sites",
 "species"),
 type = c("text",
 "points"),
 xlab = "PC1: Major sources of variation",
 ylab = "PC2: Secondary sources of variation")
ordihull(iris.pca,
 group = iris$Species,
 col = c(1,2,3))

ordihull(iris.pca,
 group = iris$Species,
 col = c(1,2,3))

boxplot(P.len ~ Species,
 data = iris,
 col = c(1,2,3))

boxplot(S.wid ~ Species,
 data = iris,
 col = c(1,2,3))
plot(P.len ~ P.wid, data = iris, col = as.numeric(iris$Species))

```



### 82.2.2 Ecology: Do trees prefer certain habitats

- 50 plots in a forest, each tree identified and counted.
- Each tree species is a dimension of data (column)

```
you should be familiar with all of this code
```

```
50 plots
nrow(BCI)
```

```
[1] 50
```

```
225 different species
ncol(BCI)
```

```
[1] 225
```

```
head(BCI)[5:10]
```

```
Adelia.triloba Aegiphila.panamensis Alchornea.costaricensis
1 0 0 2
2 0 0 1
3 0 0 2
4 3 0 18
5 1 1 3
6 0 0 2
Alchornea.latifolia Alibertia.edulis Allophylus.psilospermus
1 0 0 0
2 0 0 0
3 0 0 0
4 0 0 0
5 0 0 1
6 1 0 0
```

```
3 major habitat types
summary(BCI.env$Habitat)

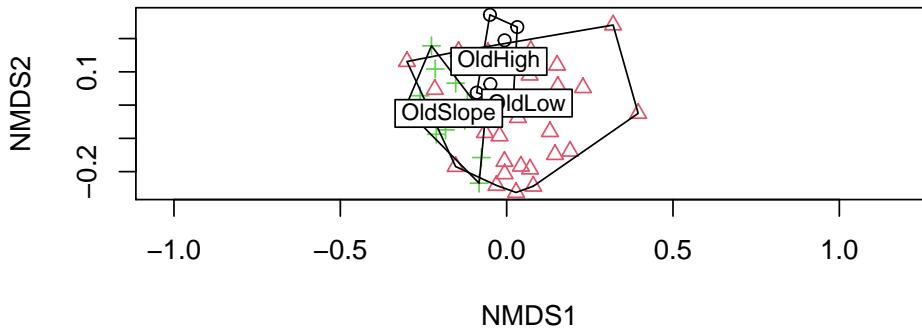
OldHigh OldLow OldSlope Swamp Young
8 26 12 2 2

Set minor habitats to NA
you should be able to understand what this code is doing

BCI.env$Habitat[grep("Swamp",BCI.env$Habitat)] <- NA
BCI.env$Habitat[grep("Young",BCI.env$Habitat)] <- NA
```

### 82.2.2.1 Do certain habitats have the same species?

```
Square root transformation
Wisconsin double standardization
Run 0 stress 0.2539738
Run 1 stress 0.2556125
Run 2 stress 0.3043432
Run 3 stress 0.2633574
Run 4 stress 0.2536839
... New best solution
... Procrustes: rmse 0.0130077 max resid 0.06050684
Run 5 stress 0.2590272
Run 6 stress 0.2549637
Run 7 stress 0.2542488
Run 8 stress 0.2536839
... Procrustes: rmse 1.949259e-05 max resid 0.0001015199
... Similar to previous best
Run 9 stress 0.2568008
Run 10 stress 0.2547549
Run 11 stress 0.2561394
Run 12 stress 0.2600732
Run 13 stress 0.2547549
Run 14 stress 0.273073
Run 15 stress 0.2556126
Run 16 stress 0.2551672
Run 17 stress 0.2568763
Run 18 stress 0.2791241
Run 19 stress 0.256146
Run 20 stress 0.2586517
*** Solution reached
```



### 82.2.3 Phylogenetics

Each loci in an alignment is a dimension

```
DNA.dat[,1:10]
```

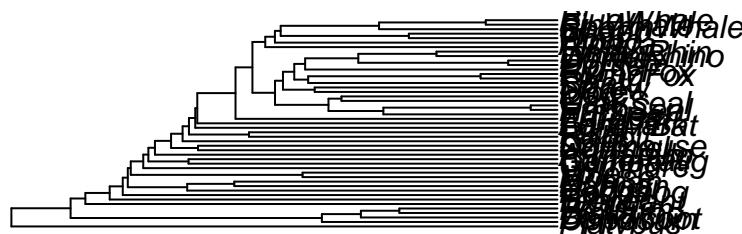
```

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "G" "A" "A" "A" "C" "C" "G" "G" "G" "C"
[2,] "T" "A" "T" "A" "C" "C" "G" "T" "G" "C"
[3,] "T" "A" "T" "A" "C" "C" "G" "G" "G" "A"
[4,] "A" "A" "A" "A" "C" "C" "G" "G" "G" "C"

```

Clustering methods can summarize sequences as phylogenetic tree

```
d_hamming <- dist.hamming(Laurasiatherian)
tree <- upgma(d_hamming)
plot(tree)
```



```
par(mfrow = c(1,1))
```

#### 82.2.4 Transcriptomics

- Calculate relative level of gene expression in different genes.
  - Each gene is a dimension

### 82.3 Exploratory versus Hypothesis testing

- Hypothesis testing - generates statistical values (p-values, etc)

- Exploratory - uses algorithms to generates graphs

```
myTree <- ape::read.tree(text='(Exploratory, Hypothesis-testing);')
plot(myTree, direction = "rightwards", cex = 3,
 main = "Types of Data Analysis: ")
```

### Types of Data Analysis:





# Chapter 83

## Overview: The vegan package for PCA

```
library(compbio4all)
```

### 83.0.1 Download packages

Only do this once, then comment out of the script. You probably already did this in the previous Code Checkpoint.

```
install.packages("ggplot2")
install.package("vegan")
```

### 83.0.2 Load the libraries

```
library(ggplot2)
library(vegan)
```

### 83.0.3 Load the msleep package

The mammals dataset is a classic dataset in the MASS package. msleep is an updated version of the data that includes more numeric data (e.g. hours of sleep) and categorical data (e.g. if a species is endangered)

```
data(msleep)
```

### 83.0.4 Subset numeric variables

`msleep` has a mixture of numeric and categorical variables. We want only the numeric data and a few key labels, which I'll also have to set up a special way. Don't worry about the details

First, all the columns we may want

```
msleep_num <- msleep[,c("sleep_total","sleep_rem",
 "sleep_cycle","awake","brainwt" ,
 "bodywt","vore","order")]
```

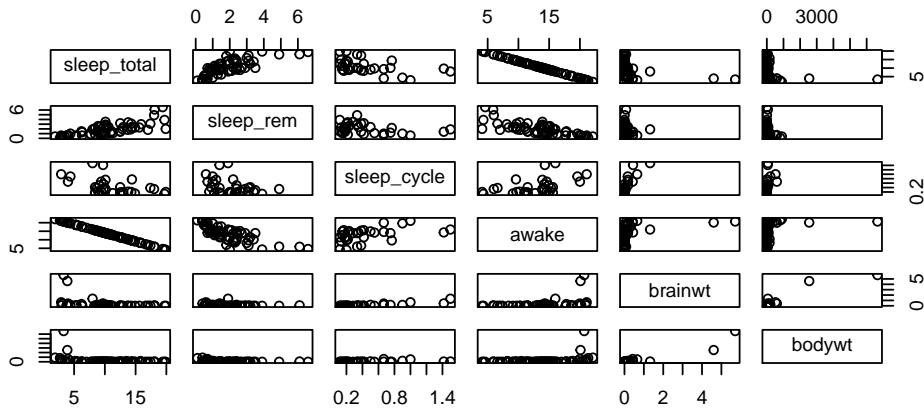
All data should be continuous numeric that can take on decimal values, except the last 2

```
summary(msleep_num[,-c(7:8)])
```

```
sleep_total sleep_rem sleep_cycle awake
Min. : 1.90 Min. :0.100 Min. :0.1167 Min. : 4.10
1st Qu.: 7.85 1st Qu.:0.900 1st Qu.:0.1833 1st Qu.:10.25
Median :10.10 Median :1.500 Median :0.3333 Median :13.90
Mean :10.43 Mean :1.875 Mean :0.4396 Mean :13.57
3rd Qu.:13.75 3rd Qu.:2.400 3rd Qu.:0.5792 3rd Qu.:16.15
Max. :19.90 Max. :6.600 Max. :1.5000 Max. :22.10
NA's :22 NA's :51
brainwt bodywt
Min. :0.00014 Min. : 0.005
1st Qu.:0.00290 1st Qu.: 0.174
Median :0.01240 Median : 1.670
Mean :0.28158 Mean :166.136
3rd Qu.:0.12550 3rd Qu.: 41.750
Max. :5.71200 Max. :6654.000
NA's :27
```

We can look at these data with a scatterplot matrix

```
plot(msleep_num[,-c(7:8)])
```



Convert to data frame

```
msleep_num <- data.frame(msleep_num)
```

Remove missing values

```
msleep_num <- na.omit(msleep_num)
```

## 83.1 Principal components analysis - base R

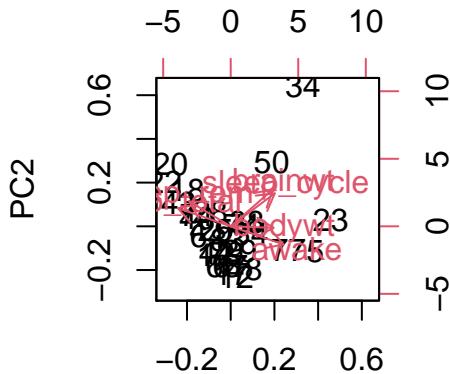
Principal component analysis is typically done using base R functions.

Run the PCA

```
pca.out <- prcomp(msleep_num[,-c(7,8)], scale = TRUE)
```

Plot the output

```
biplot(pca.out)
```



## 83.2 Principal components analysis - vegan

The base R PCA output isn't very flexible. The R package `vegan` has a function `rda()` which does PCA and has many more nice features.

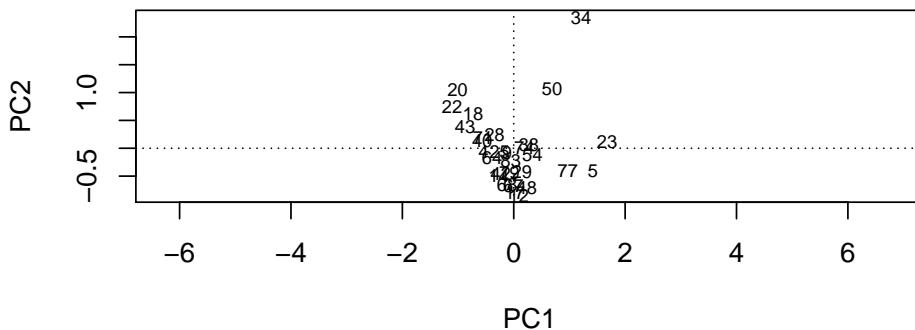
For another example see <https://rpubs.com/brouwern/veganpca>

Run the PCA using `rda()`

```
rda.out <- vegan::rda(msleep_num[,-c(7,8)], scale = TRUE)
```

This displays the 2D PCA plot without the arrows. For more info on what this code does, see the RPubs document linked above

```
biplot(rda.out, display = "sites")
```



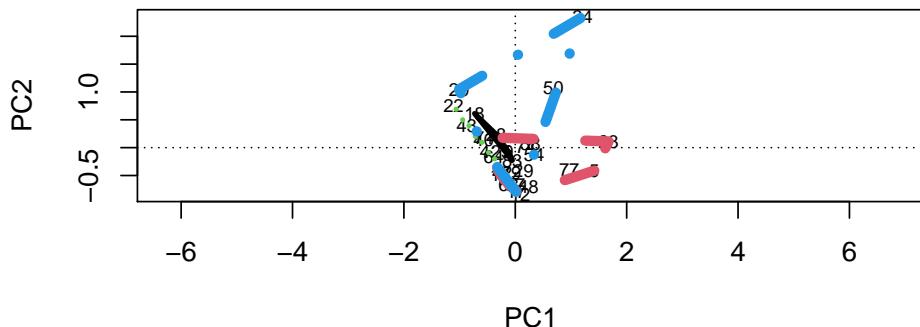
`vegan` has some nice tools for groups things.

In this dataset I don't expect there to be any interest groups, but I'll check anyway. I will supply this code if needed.

PCA is an exploratory method. First I'll see if there are any groupings based on diet ("vore"). Not really

```
biplot(rda.out, display = "sites")

vegan::ordihull(rda.out,
 group = msleep_num$vore,
 col = 1:7,
 lty = 1:7,
 lwd = c(3,6))
```



### 83.3 Task

Copy the previous code chunk and change the group so that the taxonomic order is plotted. Upload the image to this assignment. Consider if there are any meaningful groups.



# Chapter 84

## PCA etc worked example

```
library(compbio4all)
library(ggpubr)
```

### 84.1 Assignment

1. Write a 1 paragraph description of the dataset and the goal of the analysis.  
Indicate the exact table and page number that the data can from.
2. Provide a Citation for the dataset
3. Create the dataset from vectors and write a 2-3 sentence description about the process, explaining it to someone who uses Excel, not R.
4. Make a scatterplot using ggpubr::ggscater(), plotting variables against each other and using a third for color and a fourth for size.
5. Run PCA in vegan using the rda() function
6. Write a paragraph comparing your PCA results to the original one from the reading (page x, y)
7. Run a cluster analysis that replicates how the original cluster analysis was done. Use the correct distance (cosine, Manhattan, etc) and clustering algorithm (WPGMA vs. UPGMA)
8. Write a paragraph comparing your clustering results to the original one from the reading.
9. Write a 1-paragraph conclusion summarizing to what degree you were able to replicate the original results.

### 84.2 Introduction

Write a 1 paragraph description of the dataset and the goal of the analysis.

Provide a properly formatted citation for the dataset

[https://www.sciencedirect.com/science/article/pii/S0003347212004824?casa\\_token=F0SJxpVbnZgAAAAA:mjn-N8qJdirz1yGd84quoF97FRJwWGBLUXFsGOt1vJytKmfb6esEu43baKwxjibiOizL8mZfsQ#bib29](https://www.sciencedirect.com/science/article/pii/S0003347212004824?casa_token=F0SJxpVbnZgAAAAA:mjn-N8qJdirz1yGd84quoF97FRJwWGBLUXFsGOt1vJytKmfb6esEu43baKwxjibiOizL8mZfsQ#bib29)

Ecology of culture: do environmental factors influence foraging tool use in wild chimpanzees, *Pan troglodytes verus*?

### 84.3 Data

Set up dataframe from vectors.

```
#Tai Northa Tai Southa Yealéa,b Bossouw,c Seringbarab,d Gashakae Lopéf
IC IC IC Guinea Guinea Nigeria Ga
1 2 3 4 5 6 7
country <- c("IC","IC","IC","Guinea","Guinea","Nigeria","Gabon")
site <- c("tai_T","tai_S","yealea","bossou","seringbara","gashaka","l
coula_edulis <- c(17.7, 38.8, 2.4, 0, 0, 0, 6.7)
detarium_spp <- c(0.2, 0.1, 0.3, 0, 0, 0.1, 0.05)
elaeis_guineensis <- c(0.01, 0.01, 4.2, 13.5, 1, 0.3, 0.9)
parinari_spp <- c(1.3, 1.5, 1.5, 0.01, 3.4, 0, 0)
panda_oleosa <- c(0.5, 1.1, 0.2, 0, 0, 0, 0.1)
sacoglottis_gabonensis <- c(6.3, 1.2, 0, 0, 0, 0.8)

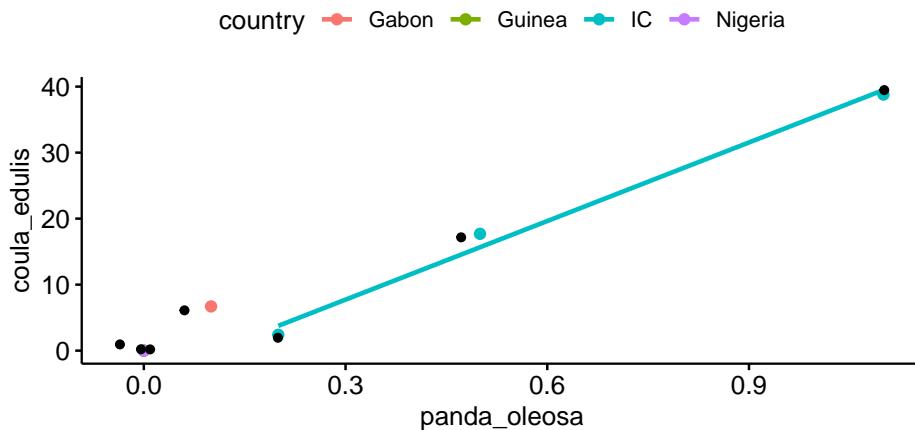
dat <- data.frame(country,
 site,
 coula_edulis,
 detarium_spp,
 elaeis_guineensis,parinari_spp,
 panda_oleosa,sacoglottis_gabonensis)

row.names(dat) <- dat[,2]
```

### 84.4 Scatter plot

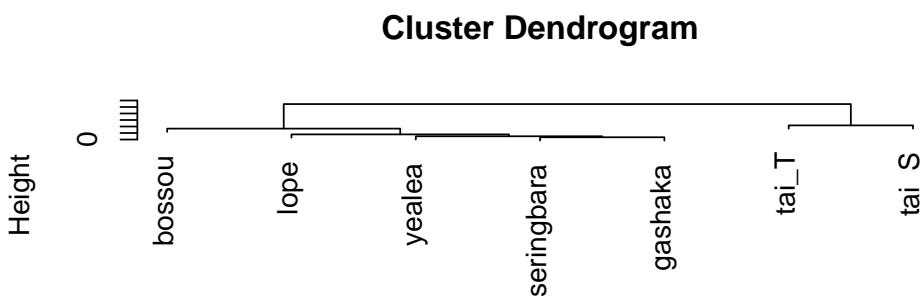
Add color, size and shape differences.

```
ggscatter(y = "coula_edulis",
 x = "panda_oleosa",
 color = "country",
 add = "reg.line",
 # ellipse = T,
 data = dat) + geom_jitter()
```



## 84.5 Cluster analysis

```
dist_euclid <- dist(dat[,-c(1:2)])
clust_out <- hclust(dist_euclid,
 method = "ward.D")
plot(clust_out,
 labels = row.names(dat))
```



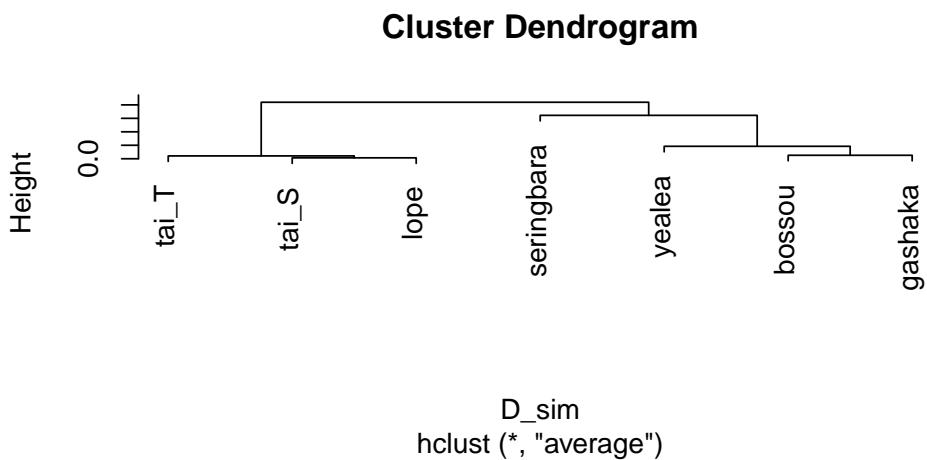
dist\_euclid  
hclust (\*, "ward.D")

Cosine distances

```
Matrix <- as.matrix(dat[,-c(1,2)])
sim <- Matrix / sqrt(rowSums(Matrix * Matrix))
sim <- sim %*% t(sim)
#Convert to cosine dissimilarity matrix (distance matrix).

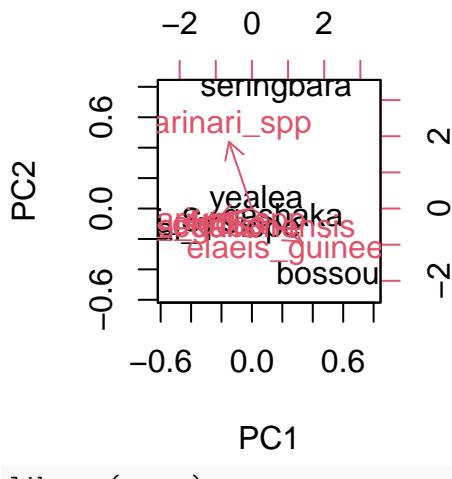
D_sim <- as.dist(1 - sim)
```

```
clust_out <- hclust(D_sim,
 method = "average")
plot(clust_out,
 labels = row.names(dat))
```



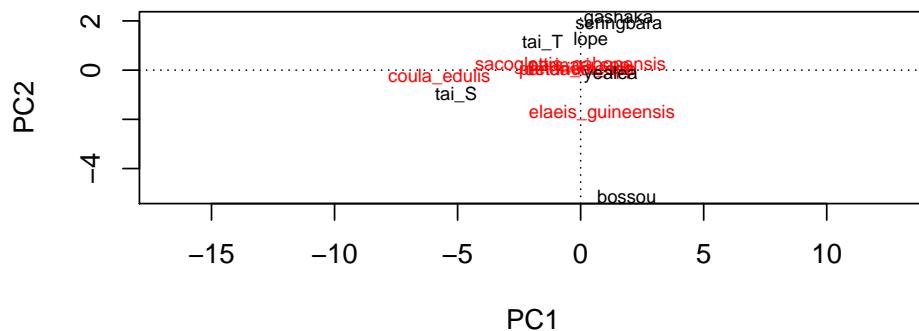
## 84.6 PCA

```
biplot(prcomp(dat[,-c(1:2)],center = T,scale = T))
```



```
library(vegan)

vegan_pca <- rda(dat[,-c(1:2)])
plot(vegan_pca)
```





# Chapter 85

## Another PCA tutorial with Fisher's Iris Data

```
library(compbio4all)
```

TODO: swap out irises

### 85.1 Important functions

- 2 functions in R for doing PCA
  - princomp()
  - prcomp()
  - difference = ?
  - princomp seems more flexible

### 85.2 Useful packages

- FactoMineR: has PCA plotting tools
- ade4: has PCA plotting tools

### 85.3 Data: Fisher's Irises

From R's help file: >“This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.”

- 3 species

- 150 observations
- 4 characteristics
  - “Sepal.Length”
  - “Sepal.Width”
  - “Petal.Length”
  - “Petal.Width”
- Characteristics highly correlated
  - plants/species with long sepals also have long petals
  - cf (allometric scaling)[<https://en.wikipedia.org/wiki/Allometry>]

```
data(iris)
dim(iris)

[1] 150 5

names(iris)

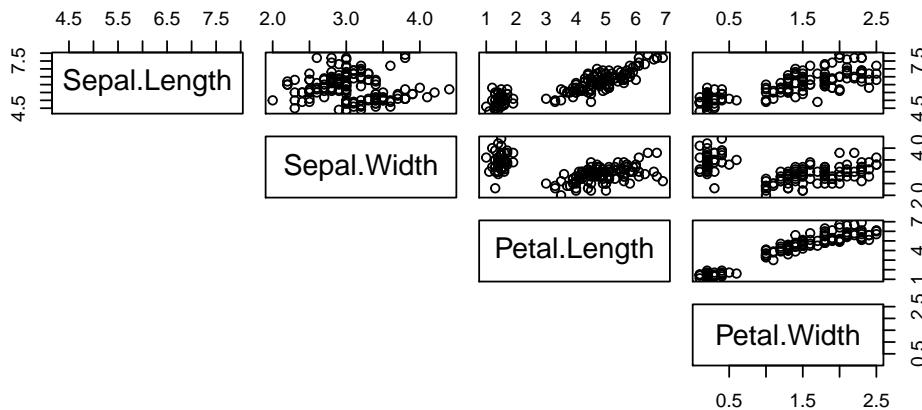
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

## 85.4 Data visualization

### 85.4.1 2D visualization

- plot each of the 4 traits against each other
- “-5” drops the last columns, which is species names

```
pairs(iris[,-5], lower.panel = NULL)
```



- Most traits highly correlated
  - ie, Tell me the width of a petal, and I can guess the length pretty well
- Some traits not as well correlated

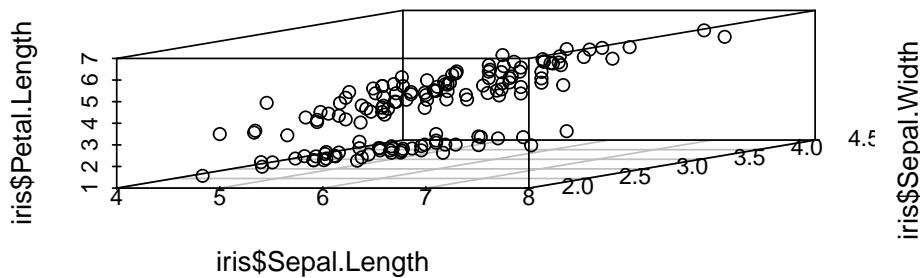
- tell me sepal width, and I won't be able to guess sepal lenght very well

## 85.5 3D visualization

- We can visualization up to 3 dimensions at one time

```
library(scatterplot3d)

scatterplot3d::scatterplot3d(iris$Sepal.Length, # x axis
 iris$Sepal.Width, # y axis
 iris$Petal.Length, # z axis
)
```



## 85.6 Beyond 3D dimensions with ordination

- ordination is often called a “dimensions reduction” technique
- Take “high dimensional data” and plot in 2D or 3D while preserving key features of the original data
- This is done by representing multidimensional distances as fewer dimensions

Two steps

- First, run PCA
- 2nd, visualize w/“biplot”

## 85.7 Run PCA

### 85.7.1 PCA with prcomp

- note 1-sided formula
  - “ $\sim$  Sepal.Length + ...”

```
prcomp.iris <- prcomp(~ Sepal.Length +
 Sepal.Width +
 Petal.Length +
```

```
Petal.Width,
 data = iris)
```

### 85.7.2 PCA with princomp

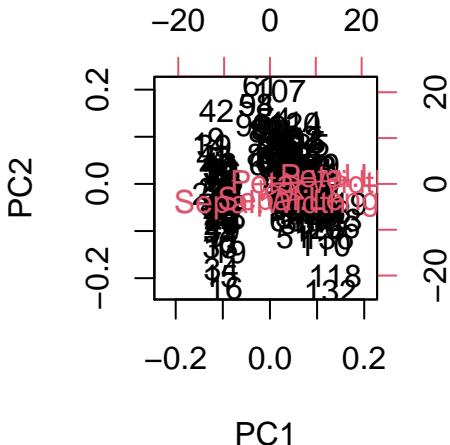
- cor = TRUE: use correlation matrix instead of covariance matrix; relates to underlying math
- scores = TRUE: keep underlying output

```
princomp.iris <- princomp(~ Sepal.Length +
 Sepal.Width +
 Petal.Length +
 Petal.Width,
 data = iris,
 cor = TRUE,
 scores = TRUE)
```

## 85.8 Visualize with biplot

- A biplot is a visualization of the PCA output
- Data points plotted as numbers (row number)
- X axis = “principal component 1” (PC1)
- Y axis = “principal component 2” (PC2)
- These can be thought of as “latent” variables that are implied by the data (?)
- Red arrows:
  - Relationship between original traits and the new PC
  - Arrows that overlap are highly correlated

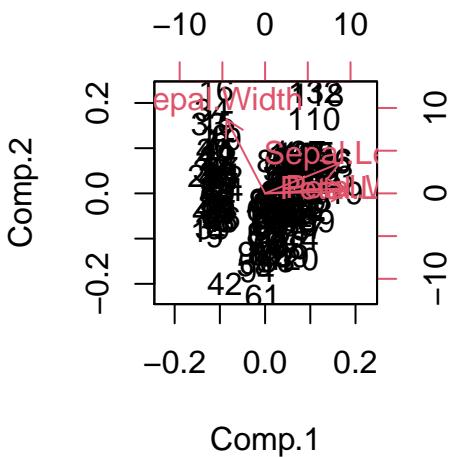
```
biplot(prcomp.iris)
```



### 85.8.2 princomp output

Difference is minor; appears to mostly be due to scaling, not a different answer.

```
biplot(princomp.iris)
```



## 85.9 Plotting by hand

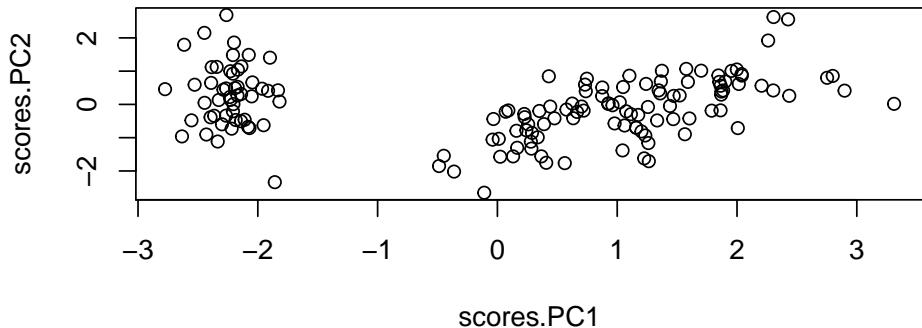
### 85.9.1 Extract underlying data (“scores”)

```
scores.PC1 <- princomp.iris$scores[, "Comp. 1"]
scores.PC2 <- princomp.iris$scores[, "Comp. 2"]
```

Plotting by hand; note that axes aren't standardized/scaled the same way as biplot(). To fix this just need to appropriate transformation

402 CHAPTER 85. ANOTHER PCA TUTORIAL WITH FISHER'S IRIS DATA

```
plot(scores.PC1,
 scores.PC2)
```

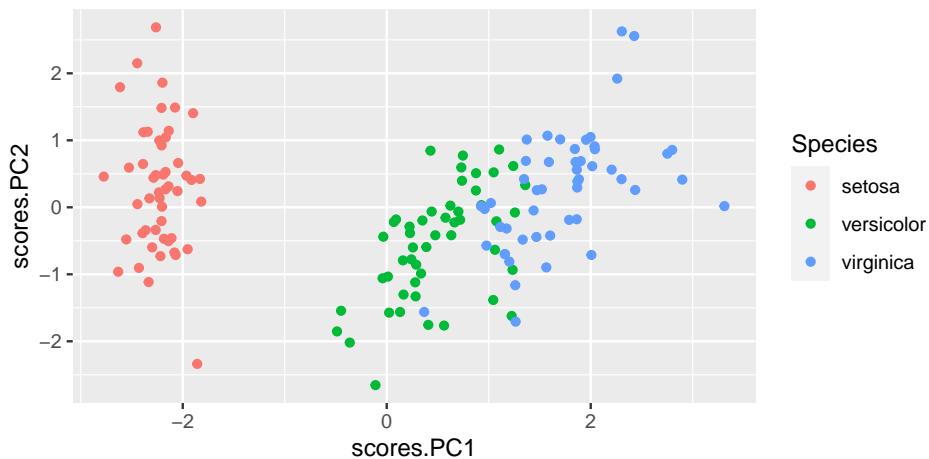


Add scores to raw data

```
iris2 <- iris
iris2$scores.PC1 <- scores.PC1
iris2$scores.PC2 <- scores.PC2
```

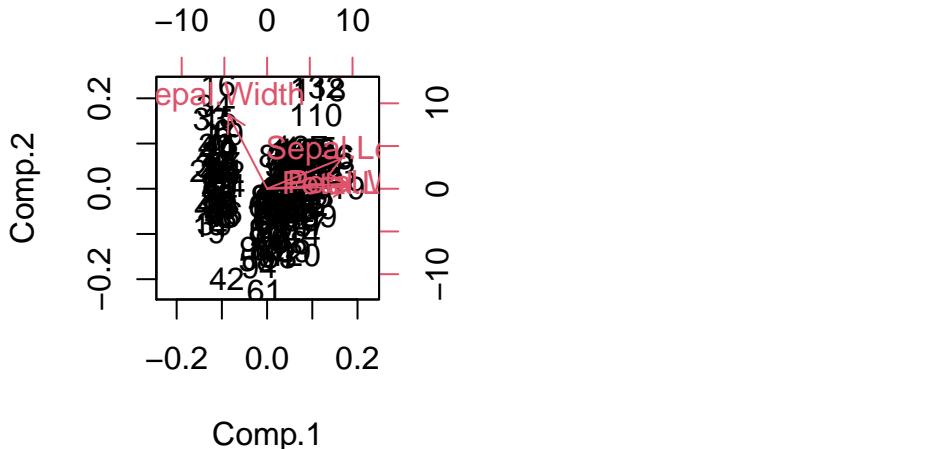
Plot in ggplot with species color coded

```
library(ggplot2)
library(cowplot)
qplot(x = scores.PC1,
 y = scores.PC2,
 color = Species,
 data = iris2)
```



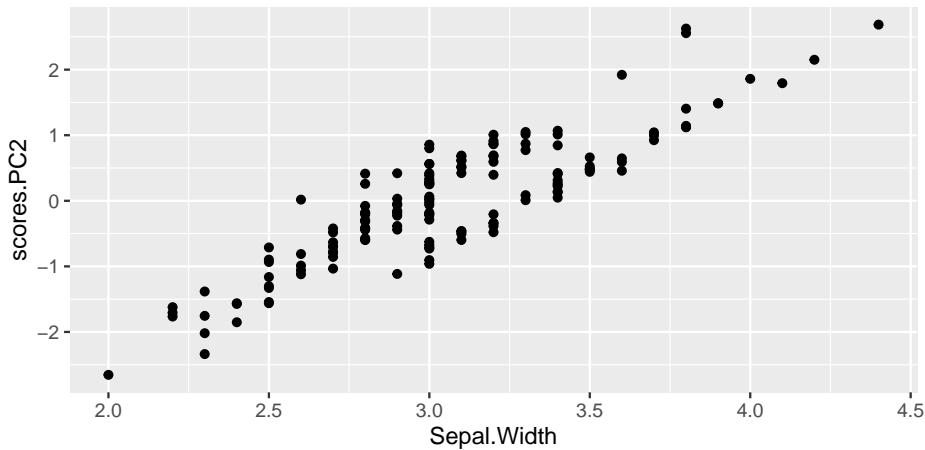
The biplot indicates that Sepal.Width falls mostly along the y axis

```
biplot(princomp.iris)
```



This implies that it is being represent by PC2 and highly correlated with it. We can plot the raw data for sepal with against PC2

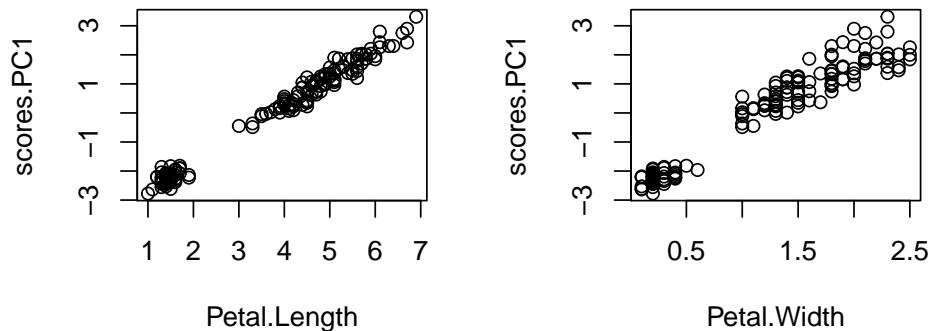
```
qplot(y = scores.PC2,
 x = Sepal.Width,
 data = iris2)
```



The labels might be hard to read, but petal width and petal length are pointing to the right along the x axis. They are therefore correlated with PC1

```
par(mfrow = c(1,2))
plot(scores.PC1~Petal.Length, data = iris2)
plot(scores.PC1~Petal.Width, data = iris2)
```

404 CHAPTER 85. ANOTHER PCA TUTORIAL WITH FISHER'S IRIS DATA

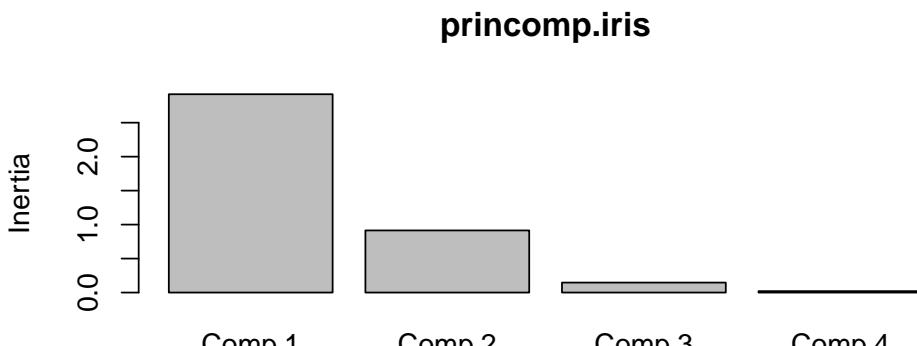


# Chapter 86

## Beyond PC1 and PC2

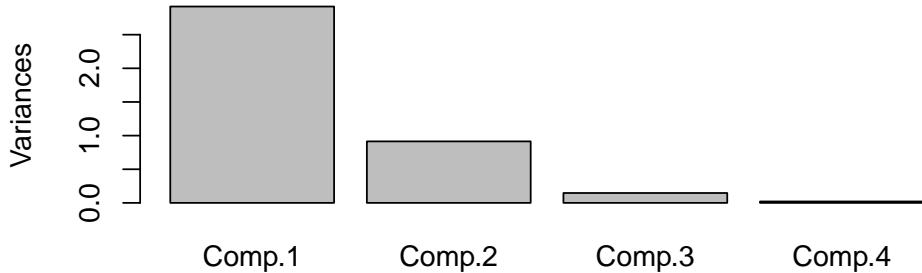
PCA creates as many new/synthetic/latent PCs as there are variables. Typically the first 2 capture most of the interesting features of the data, but sometimes the additional ones are also useful. A screeplot can be used to determine when PCs are becoming less informative. When there is a steep decline between PCs, there is still information to be gained by looking at them. The following plot implies the PC3 might be useful to look at, but not PC4. There's a step drop from 1 to 2, and 2 to 3, but not 3 to 4.

```
screeplot(princomp.iris)
```



```
#equivalent:
plot(princomp.iris)
```

### princomp.iris



## 86.1 Summary command

```
#summary on prcomp
summary(prcomp.iris)

Importance of components:
PC1 PC2 PC3 PC4
Standard deviation 2.0563 0.49262 0.2797 0.15439
Proportion of Variance 0.9246 0.05307 0.0171 0.00521
Cumulative Proportion 0.9246 0.97769 0.9948 1.00000
summary(prcomp.iris, loadings = TRUE) #doesn't do anything

Importance of components:
PC1 PC2 PC3 PC4
Standard deviation 2.0563 0.49262 0.2797 0.15439
Proportion of Variance 0.9246 0.05307 0.0171 0.00521
Cumulative Proportion 0.9246 0.97769 0.9948 1.00000
#summary on princomp, the more versatile command
summary(princomp.iris)

Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4
Standard deviation 1.7083611 0.9560494 0.38308860 0.143926497
Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
Cumulative Proportion 0.7296245 0.9581321 0.99482129 1.000000000
summary(princomp.iris, #see ?summary.princomp for details
 loadings = TRUE)

Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4
Standard deviation 1.7083611 0.9560494 0.38308860 0.143926497
Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
```

```

Cumulative Proportion 0.7296245 0.9581321 0.99482129 1.0000000000
##
Loadings:
Comp.1 Comp.2 Comp.3 Comp.4
Sepal.Length 0.521 0.377 0.720 0.261
Sepal.Width -0.269 0.923 -0.244 -0.124
Petal.Length 0.580 -0.142 -0.801
Petal.Width 0.565 -0.634 0.524

#summary with additional commands
summary(princomp.iris, #see summary.princomp() for details
 cutoff = 0.1,
 digits = 1)

Importance of components:
Comp.1 Comp.2 Comp.3 Comp.4
Standard deviation 1.7083611 0.9560494 0.38308860 0.143926497
Proportion of Variance 0.7296245 0.2285076 0.03668922 0.005178709
Cumulative Proportion 0.7296245 0.9581321 0.99482129 1.0000000000

structure of PCA output
#structure of PCA output
str(prcomp.iris)

List of 6
$ sdev : num [1:4] 2.056 0.493 0.28 0.154
$ rotation: num [1:4, 1:4] 0.3614 -0.0845 0.8567 0.3583 -0.6566 ...
..- attr(*, "dimnames")=List of 2
...$: chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
...$: chr [1:4] "PC1" "PC2" "PC3" "PC4"
$ center : Named num [1:4] 5.84 3.06 3.76 1.2
..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
$ scale : logi FALSE
$ x : num [1:150, 1:4] -2.68 -2.71 -2.89 -2.75 -2.73 ...
..- attr(*, "dimnames")=List of 2
...$: chr [1:150] "1" "2" "3" "4" ...
...$: chr [1:4] "PC1" "PC2" "PC3" "PC4"
$ call : language prcomp(formula = ~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width)
- attr(*, "class")= chr "prcomp"

str(princomp.iris)

List of 7
$ sdev : Named num [1:4] 1.708 0.956 0.383 0.144
..- attr(*, "names")= chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
$ loadings: 'loadings' num [1:4, 1:4] 0.521 -0.269 0.58 0.565 0.377 ...
..- attr(*, "dimnames")=List of 2

```

```

...$: chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
...$: chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
$ center : Named num [1:4] 5.84 3.06 3.76 1.2
..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
$ scale : Named num [1:4] 0.825 0.434 1.759 0.76
..- attr(*, "names")= chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
$ n.obs : int 150
$ scores : num [1:150, 1:4] -2.26 -2.08 -2.36 -2.3 -2.39 ...
..- attr(*, "dimnames")=List of 2
...$: chr [1:150] "1" "2" "3" "4" ...
...$: chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
$ call : language princomp(formula = ~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width)
- attr(*, "class")= chr "princomp"

```

## 86.2 PCA by hand

in another script?

- from pdf “Principal Component Analysis using R” by anon
- dated November 25, 2009

## 86.3 Set up data

Create dataframe with just the numeric data (drop the `class`)

```
iris.num.data <- iris[,1:4]
```

## 86.4 covariance vs. correlation

### 86.4.1 covariance matrix

covariances can take on any numeric value

```
cov(iris.num.data)
```

```

Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length 0.6856935 -0.0424340 1.2743154 0.5162707
Sepal.Width -0.0424340 0.1899794 -0.3296564 -0.1216394
Petal.Length 1.2743154 -0.3296564 3.1162779 1.2956094
Petal.Width 0.5162707 -0.1216394 1.2956094 0.5810063

```

### 86.4.2 correlation matrix

correlations are all -1 to 1

```
cor(iris.num.data)

Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length 1.0000000 -0.1175698 0.8717538 0.8179411
Sepal.Width -0.1175698 1.0000000 -0.4284401 -0.3661259
Petal.Length 0.8717538 -0.4284401 1.0000000 0.9628654
Petal.Width 0.8179411 -0.3661259 0.9628654 1.0000000
```

Correlation = covariance of re-scaled data, where “scaling” is mean-centered and standardized by SD

covariance matrix on scaled data == cor matrix called on raw data

confirm this; rounding needed to get correct answer, probably due to printing errors

```
round(cov(scale(iris.num.data)),10) ==
 round(cor(iris.num.data),10)

Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length TRUE TRUE TRUE TRUE
Sepal.Width TRUE TRUE TRUE TRUE
Petal.Length TRUE TRUE TRUE TRUE
Petal.Width TRUE TRUE TRUE TRUE
```



# Chapter 87

## eigen values

- PCA is a type of “eigen decomposition”
- eigen() command generates eigenvalues and eigenvectors
- use \$ operator on eigen results to get what you want

```
all eigen... results
eigen(cov(iris.num.data))

eigen() decomposition
$values
[1] 4.22824171 0.24267075 0.07820950 0.02383509
##
$vectors
[,1] [,2] [,3] [,4]
[1,] 0.36138659 -0.65658877 -0.58202985 0.3154872
[2,] -0.08452251 -0.73016143 0.59791083 -0.3197231
[3,] 0.85667061 0.17337266 0.07623608 -0.4798390
[4,] 0.35828920 0.07548102 0.54583143 0.7536574

#eigen valeus only
e.val <- eigen(cov(iris.num.data))$values

#eigen vectors only
e.vec <- eigen(cov(iris.num.data))$vectors
```

we could use covariances instead off correlations and get different results

```
#eigen(cov(iris.num.data))$values

[1] 2.91849782 0.91403047 0.14675688 0.02071484

calling scale() within cov() should give us the same result
```

```
eigen(cov(scale(iris.num.data)))$values

[1] 2.91849782 0.91403047 0.14675688 0.02071484

Chekc this

round(eigen(cov(scale(iris.num.data)))$values,10) ==
 round(eigen(cor(iris.num.data))$values,10)

[1] TRUE TRUE TRUE TRUE
```

## 87.1 Calculation of the the Principal Components

- this requires doing matrix multiplication
  - `%*%`
- of the data times the eigen vectors
- NB: data must be in matrix form, not dataframe

```
#doesn't work
iris.num.data %*% e.vec

PC <- as.matrix(iris.num.data) %*% e.vec

this spits out a value for every element in your
data matrix
dim(iris.num.data) == dim(PC)
```

```
[1] TRUE TRUE

#what this is doing is multiplying the data by each
#eigen vector successively
```

```
x <- cbind(as.matrix(iris.num.data) %*% e.vec[,1]
,as.matrix(iris.num.data) %*% e.vec[,2]
,as.matrix(iris.num.data) %*% e.vec[,3]
,as.matrix(iris.num.data) %*% e.vec[,4])
```

```
x == PC

[,1] [,2] [,3] [,4]
[1,] TRUE TRUE TRUE TRUE
[2,] TRUE TRUE TRUE TRUE
[3,] TRUE TRUE TRUE TRUE
[4,] TRUE TRUE TRUE TRUE
[5,] TRUE TRUE TRUE TRUE
[6,] TRUE TRUE TRUE TRUE
[7,] TRUE TRUE TRUE TRUE
```

```
[8,] TRUE TRUE TRUE TRUE
[9,] TRUE TRUE TRUE TRUE
[10,] TRUE TRUE TRUE TRUE
[11,] TRUE TRUE TRUE TRUE
[12,] TRUE TRUE TRUE TRUE
[13,] TRUE TRUE TRUE TRUE
[14,] TRUE TRUE TRUE TRUE
[15,] TRUE TRUE TRUE TRUE
[16,] TRUE TRUE TRUE TRUE
[17,] TRUE TRUE TRUE TRUE
[18,] TRUE TRUE TRUE TRUE
[19,] TRUE TRUE TRUE TRUE
[20,] TRUE TRUE TRUE TRUE
[21,] TRUE TRUE TRUE TRUE
[22,] TRUE TRUE TRUE TRUE
[23,] TRUE TRUE TRUE TRUE
[24,] TRUE TRUE TRUE TRUE
[25,] TRUE TRUE TRUE TRUE
[26,] TRUE TRUE TRUE TRUE
[27,] TRUE TRUE TRUE TRUE
[28,] TRUE TRUE TRUE TRUE
[29,] TRUE TRUE TRUE TRUE
[30,] TRUE TRUE TRUE TRUE
[31,] TRUE TRUE TRUE TRUE
[32,] TRUE TRUE TRUE TRUE
[33,] TRUE TRUE TRUE TRUE
[34,] TRUE TRUE TRUE TRUE
[35,] TRUE TRUE TRUE TRUE
[36,] TRUE TRUE TRUE TRUE
[37,] TRUE TRUE TRUE TRUE
[38,] TRUE TRUE TRUE TRUE
[39,] TRUE TRUE TRUE TRUE
[40,] TRUE TRUE TRUE TRUE
[41,] TRUE TRUE TRUE TRUE
[42,] TRUE TRUE TRUE TRUE
[43,] TRUE TRUE TRUE TRUE
[44,] TRUE TRUE TRUE TRUE
[45,] TRUE TRUE TRUE TRUE
[46,] TRUE TRUE TRUE TRUE
[47,] TRUE TRUE TRUE TRUE
[48,] TRUE TRUE TRUE TRUE
[49,] TRUE TRUE TRUE TRUE
[50,] TRUE TRUE TRUE TRUE
[51,] TRUE TRUE TRUE TRUE
[52,] TRUE TRUE TRUE TRUE
[53,] TRUE TRUE TRUE TRUE
```

```
[54,] TRUE TRUE TRUE TRUE
[55,] TRUE TRUE TRUE TRUE
[56,] TRUE TRUE TRUE TRUE
[57,] TRUE TRUE TRUE TRUE
[58,] TRUE TRUE TRUE TRUE
[59,] TRUE TRUE TRUE TRUE
[60,] TRUE TRUE TRUE TRUE
[61,] TRUE TRUE TRUE TRUE
[62,] TRUE TRUE TRUE TRUE
[63,] TRUE TRUE TRUE TRUE
[64,] TRUE TRUE TRUE TRUE
[65,] TRUE TRUE TRUE TRUE
[66,] TRUE TRUE TRUE TRUE
[67,] TRUE TRUE TRUE TRUE
[68,] TRUE TRUE TRUE TRUE
[69,] TRUE TRUE TRUE TRUE
[70,] TRUE TRUE TRUE TRUE
[71,] TRUE TRUE TRUE TRUE
[72,] TRUE TRUE TRUE TRUE
[73,] TRUE TRUE TRUE TRUE
[74,] TRUE TRUE TRUE TRUE
[75,] TRUE TRUE TRUE TRUE
[76,] TRUE TRUE TRUE TRUE
[77,] TRUE TRUE TRUE TRUE
[78,] TRUE TRUE TRUE TRUE
[79,] TRUE TRUE TRUE TRUE
[80,] TRUE TRUE TRUE TRUE
[81,] TRUE TRUE TRUE TRUE
[82,] TRUE TRUE TRUE TRUE
[83,] TRUE TRUE TRUE TRUE
[84,] TRUE TRUE TRUE TRUE
[85,] TRUE TRUE TRUE TRUE
[86,] TRUE TRUE TRUE TRUE
[87,] TRUE TRUE TRUE TRUE
[88,] TRUE TRUE TRUE TRUE
[89,] TRUE TRUE TRUE TRUE
[90,] TRUE TRUE TRUE TRUE
[91,] TRUE TRUE TRUE TRUE
[92,] TRUE TRUE TRUE TRUE
[93,] TRUE TRUE TRUE TRUE
[94,] TRUE TRUE TRUE TRUE
[95,] TRUE TRUE TRUE TRUE
[96,] TRUE TRUE TRUE TRUE
[97,] TRUE TRUE TRUE TRUE
[98,] TRUE TRUE TRUE TRUE
[99,] TRUE TRUE TRUE TRUE
```

```
[100,] TRUE TRUE TRUE TRUE
[101,] TRUE TRUE TRUE TRUE
[102,] TRUE TRUE TRUE TRUE
[103,] TRUE TRUE TRUE TRUE
[104,] TRUE TRUE TRUE TRUE
[105,] TRUE TRUE TRUE TRUE
[106,] TRUE TRUE TRUE TRUE
[107,] TRUE TRUE TRUE TRUE
[108,] TRUE TRUE TRUE TRUE
[109,] TRUE TRUE TRUE TRUE
[110,] TRUE TRUE TRUE TRUE
[111,] TRUE TRUE TRUE TRUE
[112,] TRUE TRUE TRUE TRUE
[113,] TRUE TRUE TRUE TRUE
[114,] TRUE TRUE TRUE TRUE
[115,] TRUE TRUE TRUE TRUE
[116,] TRUE TRUE TRUE TRUE
[117,] TRUE TRUE TRUE TRUE
[118,] TRUE TRUE TRUE TRUE
[119,] TRUE TRUE TRUE TRUE
[120,] TRUE TRUE TRUE TRUE
[121,] TRUE TRUE TRUE TRUE
[122,] TRUE TRUE TRUE TRUE
[123,] TRUE TRUE TRUE TRUE
[124,] TRUE TRUE TRUE TRUE
[125,] TRUE TRUE TRUE TRUE
[126,] TRUE TRUE TRUE TRUE
[127,] TRUE TRUE TRUE TRUE
[128,] TRUE TRUE TRUE TRUE
[129,] TRUE TRUE TRUE TRUE
[130,] TRUE TRUE TRUE TRUE
[131,] TRUE TRUE TRUE TRUE
[132,] TRUE TRUE TRUE TRUE
[133,] TRUE TRUE TRUE TRUE
[134,] TRUE TRUE TRUE TRUE
[135,] TRUE TRUE TRUE TRUE
[136,] TRUE TRUE TRUE TRUE
[137,] TRUE TRUE TRUE TRUE
[138,] TRUE TRUE TRUE TRUE
[139,] TRUE TRUE TRUE TRUE
[140,] TRUE TRUE TRUE TRUE
[141,] TRUE TRUE TRUE TRUE
[142,] TRUE TRUE TRUE TRUE
[143,] TRUE TRUE TRUE TRUE
[144,] TRUE TRUE TRUE TRUE
[145,] TRUE TRUE TRUE TRUE
```

```

[146,] TRUE TRUE TRUE TRUE
[147,] TRUE TRUE TRUE TRUE
[148,] TRUE TRUE TRUE TRUE
[149,] TRUE TRUE TRUE TRUE
[150,] TRUE TRUE TRUE TRUE

#scaled data %*% eigen-things = "PC" object
#the eigen-vectors are the "loadings" that
#transform the raw data into principal component scores

for matrix multiplication to work the number
of columns of m1 must be equal to the number
of rows of m2 in
m1 %*% m2

ncol(iris.num.data) == nrow(e.vec)

[1] TRUE

#The number of ROWS of m1 determines the number of rows
in the resulting product of the matrix mult
as.matrix(iris.num.data[1,]) %*% e.vec

[,1] [,2] [,3] [,4]
1 2.81824 -5.64635 -0.6597675 -0.03108928
as.matrix(iris.num.data[1:2,]) %*% e.vec

[,1] [,2] [,3] [,4]
1 2.818240 -5.646350 -0.6597675 -0.03108928
2 2.788223 -5.149951 -0.8423170 0.06567484

the number of rows of m1 we use doesn't affect the calculation
row.1st <- as.matrix(iris.num.data[1,]) %*% e.vec
row.all <- as.matrix(iris.num.data[,]) %*% e.vec
row.1st == row.all[1,]

[,1] [,2] [,3] [,4]
1 TRUE TRUE TRUE TRUE

#This is because matrix mult involves carrying
#out multiplication and addition functions
#on a row by row basis in m1
#(and a columns by column basis in m3)

#1st row of m1 by entire 1st column of m2
as.matrix(iris.num.data[1,]) %*% e.vec

[,1] [,2] [,3] [,4]
1 2.81824 -5.64635 -0.6597675 -0.03108928

```

```

#if we multiple the first row of our m1
#by just the 1st column of the our m2
#we get the FIRST element of our entire
#"PC" object
ele.1 <- as.matrix(iris.num.data[,]) %*% e.vec[,1]

ele.1 == PC[1,1]

[,1]
1 TRUE

we could write a little program to do the addition and
multiplication that the magic %*% operator does if
we wanted

m1 <- as.matrix(iris.num.data) #turn data into a matrix
n.rows <- nrow(m1)
n.cols <- ncol(m1)
m2 <- e.vec
ncol(m1) == nrow(m2)

make a blank matrix to hold results
m.out <- m1
m.out[,] <- NA
colnames(m.out) <- NULL
i <- 1
j <- 1
#loop over rows i and columns j
for (i in 1:n.rows)
{
 for (j in 1:n.cols)
 {
 m.out[i, j] <- sum(m1[i,] * m2[, j])
 }
}

dim(m.out)
dim(PC)

all.equal doesn't seem to be happy for some reason...
all.equal(m.out, PC)

look for any mismatches by hand
comp.ms <- round(m.out, 5) == round(PC, 5)
any(comp.ms == "FALSE")

```

## 87.2 results of matrix mult for PCA

```

a property of this new data matrix is
that the VARIANCES of PC equal the eigen values
(not the eigen vectors)
and the covarainces should be (approximately) zero

#recall that in a var-covar matrix that
#the variances are on the diagonal
the the covars are everything else

the eigen values
e.val

[1] 4.22824171 0.24267075 0.07820950 0.02383509

#covariance matrix of object PC from our matrix multiplicatio
cov(PC)

[,1] [,2] [,3] [,4]
[1,] 4.228242e+00 7.676599e-16 -2.363754e-16 -9.339419e-19
[2,] 7.676599e-16 2.426707e-01 8.393740e-17 2.254831e-16
[3,] -2.363754e-16 8.393740e-17 7.820950e-02 2.418740e-17
[4,] -9.339419e-19 2.254831e-16 2.418740e-17 2.383509e-02

#to view just the variances we can use diag()
diag(cov(PC))

[1] 4.22824171 0.24267075 0.07820950 0.02383509

#check this
round(diag(cov(PC)),10) == round(e.val,10)

[1] TRUE TRUE TRUE TRUE

#its a pain to write out round()
#the function all.equal() checks for
#near equalness of all of the elements

all.equal(diag(cov(PC)), e.val)

[1] TRUE

what happens if we call cor() on PC?
variances (on the diagonal) are all 1
cor(PC)

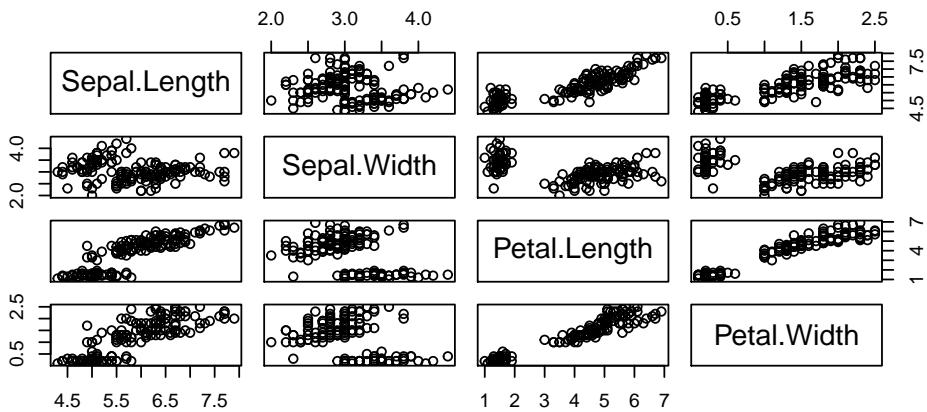
[,1] [,2] [,3] [,4]
[1,] 1.000000e+00 7.578447e-16 -4.110480e-16 -2.941925e-18
[2,] 7.578447e-16 1.000000e+00 6.092801e-16 2.964809e-15

```

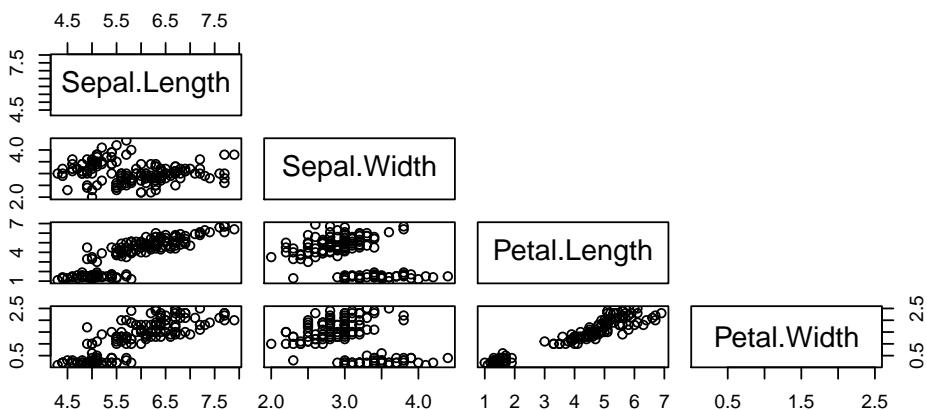
```
[3,] -4.110480e-16 6.092801e-16 1.000000e+00 5.602101e-16
[4,] -2.941925e-18 2.964809e-15 5.602101e-16 1.000000e+00
```

**### what do these data look like?**

# the original pairwise plots  
`pairs(iris.num.data)`



**#suppress upper panele**  
`pairs(iris.num.data, upper.panel = NULL)`

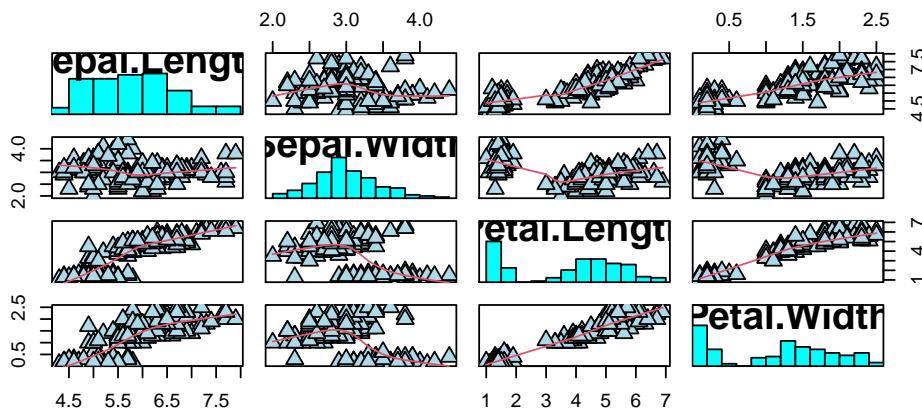


**## FANCY: put histograms on the diagonal**  
`panel.hist <- function(x, ...)`  
{  
 usr <- par("usr"); on.exit(par(usr))  
 par(usr = c(usr[1:2], 0, 1.5) )  
 h <- hist(x, plot = FALSE)  
 breaks <- h\$breaks; nB <- length(breaks)  
 y <- h\$counts; y <- y/max(y)  
 rect(breaks[-nB], 0, breaks[-1], y, col = "cyan", ...)

```

}
pairs(iris.num.data,
 panel = panel.smooth,
 cex = 1.5, pch = 24, bg = "light blue",
 diag.panel = panel.hist,
 cex.labels = 2, font.labels = 2)

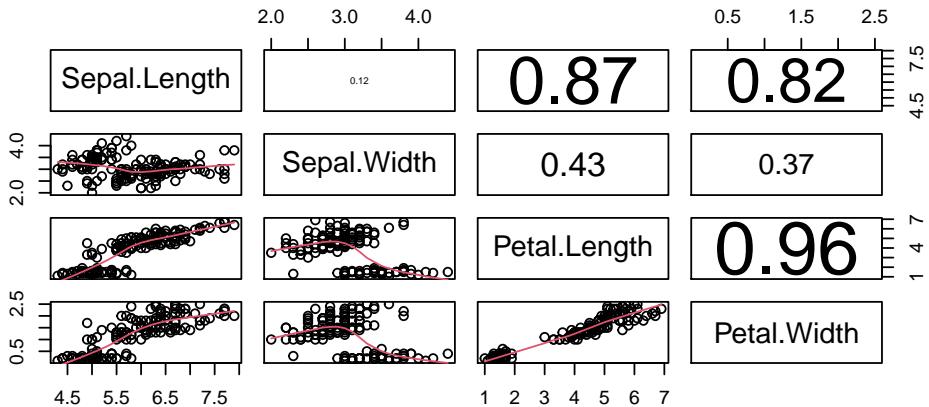
```



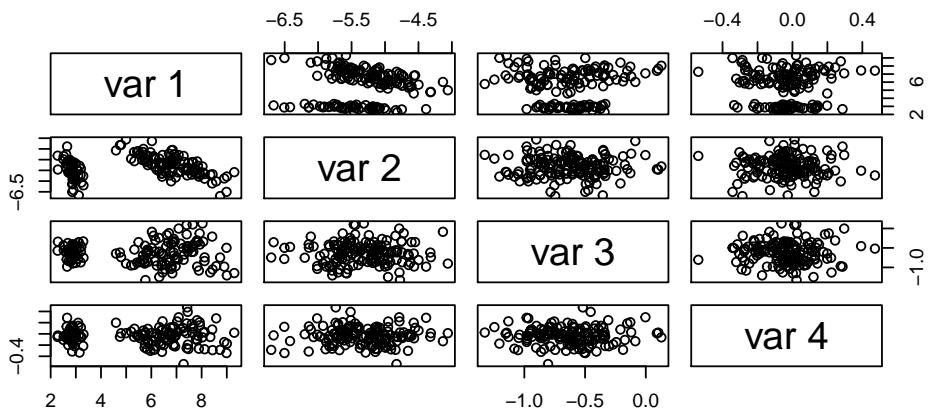
```

put (absolute) correlations on the upper panels,
with size proportional to the correlations.
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)
{
 usr <- par("usr"); on.exit(par(usr))
 par(usr = c(0, 1, 0, 1))
 r <- abs(cor(x, y))
 txt <- format(c(r, 0.123456789), digits = digits)[1]
 txt <- paste0(prefix, txt)
 if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
 text(0.5, 0.5, txt, cex = cex.cor * r)
}
pairs(iris.num.data, lower.panel = panel.smooth, upper.panel = panel.cor)

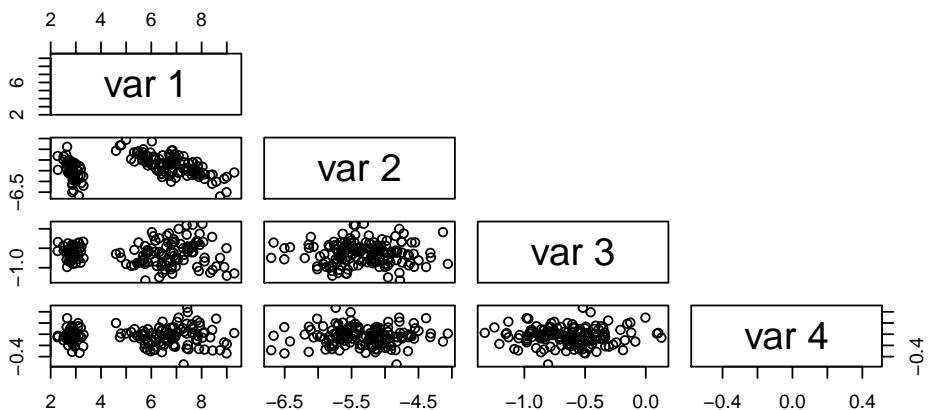
```



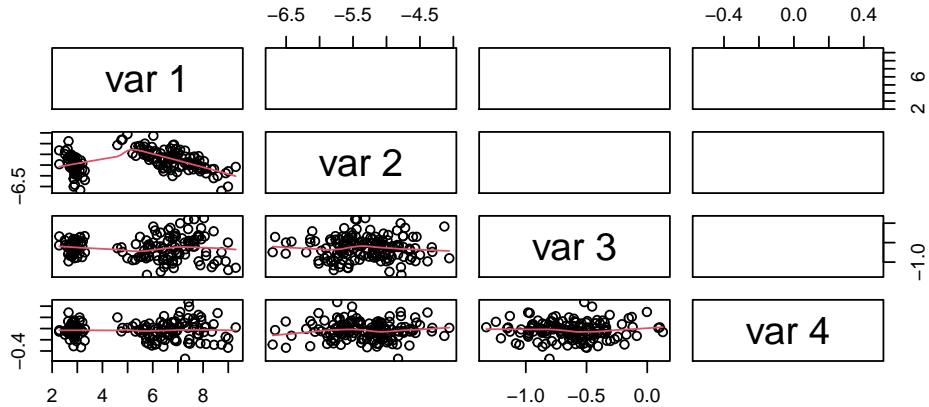
```
#pairwise comparisons of the PC object
pairs(PC)
```



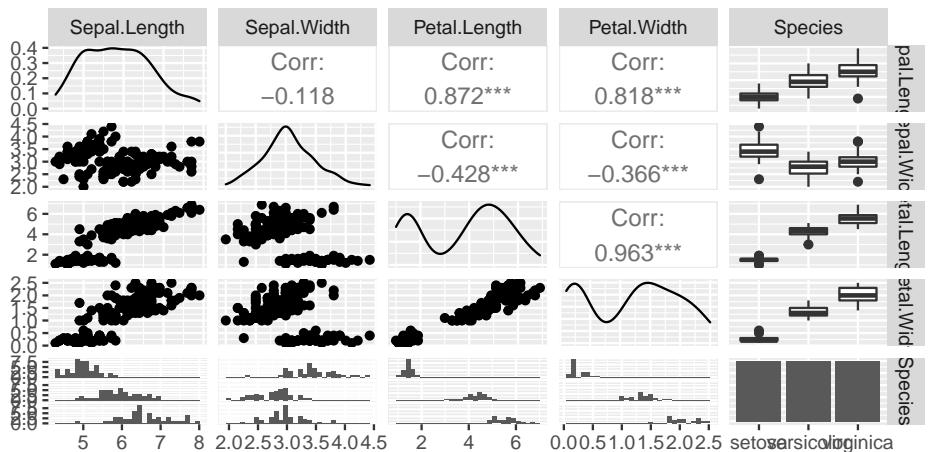
```
pairs(PC, upper.panel = NULL)
```



```
pairs(PC, lower.panel = panel.smooth, upper.panel = panel.cor)
```



```
for more summary and distributional information than you could
ever ask for:
library(ggplot2)
library(GGally)
ggpairs(iris)
```



```
###
```

```
dat.scale <- scale(iris.num.data)
```

```
t(iris.num.data) %*% PC
```

```
[,1] [,2] [,3] [,4]
Sepal.Length 5050.499 -4692.8148 -560.6014 -28.112346
Sepal.Width 2470.135 -2469.3416 -282.8001 -16.430569
Petal.Length 3641.393 -2996.5344 -355.2870 -20.504475
Petal.Width 1215.601 -955.5895 -107.3096 -3.323412
```

```
t(dat.scale) %*% PC

[,1] [,2] [,3] [,4]
Sepal.Length 274.9496 -28.670293 -8.1907943 1.3530687
Sepal.Width -122.1702 -60.571637 15.9855992 -2.6050968
Petal.Length 305.7327 3.551127 0.5032551 -0.9653406
Petal.Width 296.1348 3.580561 8.3447612 3.5114517

t(PC) %*% dat.scale

Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,] 274.949598 -122.170177 305.7326729 296.134755
[2,] -28.670293 -60.571637 3.5511272 3.580561
[3,] -8.190794 15.985599 0.5032551 8.344761
[4,] 1.353069 -2.605097 -0.9653406 3.511452

e.vec

[,1] [,2] [,3] [,4]
[1,] 0.36138659 -0.65658877 -0.58202985 0.3154872
[2,] -0.08452251 -0.73016143 0.59791083 -0.3197231
[3,] 0.85667061 0.17337266 0.07623608 -0.4798390
[4,] 0.35828920 0.07548102 0.54583143 0.7536574

Proportion of Variance explained
by the different components

#proportion explained as a percent
e.val/sum(e.val)*100

[1] 92.4618723 5.3066483 1.7102610 0.5212184

#should add to 100%
sum(e.val/sum(e.val)*100)

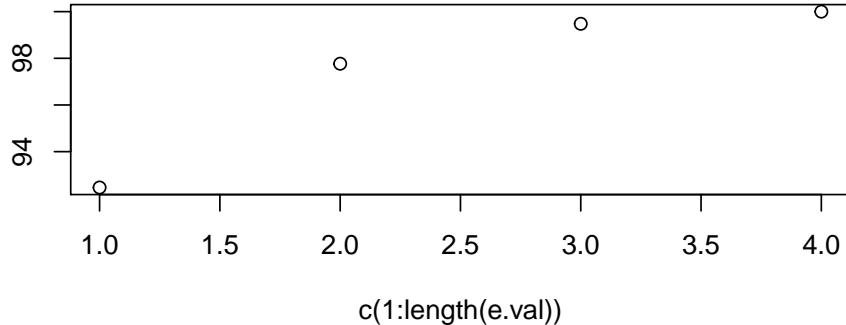
[1] 100

#can see how quickly variance is eaten up by each
#successive component by calculating cumulative sum

cumsum(e.val/sum(e.val)*100)

[1] 92.46187 97.76852 99.47878 100.00000

plot cumulative variance explained against
number of components
plot(cumsum(e.val/sum(e.val)*100) ~
 c(1:length(e.val)),
 ylab = "cumulative variance explained against")
```

cumulative variance explained against  
c(1:length(e.val))

```
So how does all this eigen-craziness compare to what
prcomp acutally does
```

```
prcomp(iris.num.data)
```

```
Standard deviations (1, ..., p=4):
[1] 2.0562689 0.4926162 0.2796596 0.1543862
##
Rotation (n x k) = (4 x 4):
PC1 PC2 PC3 PC4
Sepal.Length 0.36138659 -0.65658877 0.58202985 0.3154872
Sepal.Width -0.08452251 -0.73016143 -0.59791083 -0.3197231
Petal.Length 0.85667061 0.17337266 -0.07623608 -0.4798390
Petal.Width 0.35828920 0.07548102 -0.54583143 0.7536574
```

*#the standard deviations can be converted to the variances  
#we computed above  
#The "rotations" are the same as the eigenvectors  
#we computed above*

*#the standard deviations that prcomp() gives  
#can be converted to variances  
#which are equal to the EIGENVALUES*

```
#extract sdeviations
prcomp(iris.num.data)$sdev
```

```
[1] 2.0562689 0.4926162 0.2796596 0.1543862
#convert them to variances
prc.vars <- prcomp(iris.num.data)$sdev^2
```

```
#compar to the eigenvalues we calculated with
#eigen() above
all.equal(e.val, prc.vars)

[1] TRUE

#so, the sd^2 from prcomp() can also give
#use the proportion of explained variance

prc.vars/sum(prc.vars)

[1] 0.924618723 0.053066483 0.017102610 0.005212184

cumsum(prc.vars/sum(prc.vars))

[1] 0.9246187 0.9776852 0.9947878 1.0000000
```



## Chapter 88

# vegan PCA: Principal Components Analysis with vegans rda function

TODO: swap out irises

```
library(compbio4all)
library(ggplot2)
```

PCA (Principal Components Analysis) is easy in R, but the standard biplot() function is a little clunky. The vegan package can do PCA using the rda() function (normally for redundancy analysis) and has some nice plotting functions. (Note that ggplot is also developing biplot tools).



# Chapter 89

## Fisher's Irises

- 3 species
- 4 response variables

```
data("iris")
summary(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width
Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100
1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300
Median :5.800 Median :3.000 Median :4.350 Median :1.300
Mean :5.843 Mean :3.057 Mean :4.358 Mean :1.500
3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800
Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500
Species
setosa :50
versicolor:50
virginica :50

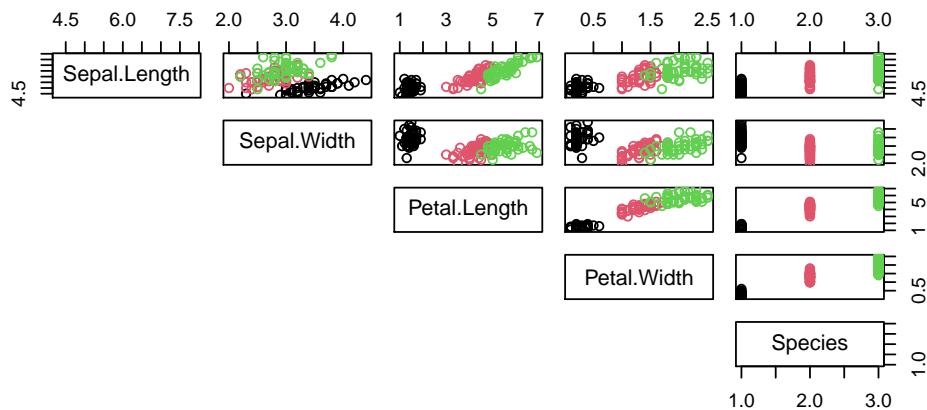
```



# Chapter 90

## Visualize variables in 2D

```
pairs(iris,
 lower.panel = NULL,
 col = as.numeric(iris$Species))
```



Note: can also be done in ggplot using GGalley::ggpairs; however, not updated for most recent version of R.



# Chapter 91

## Do PCA with base R function

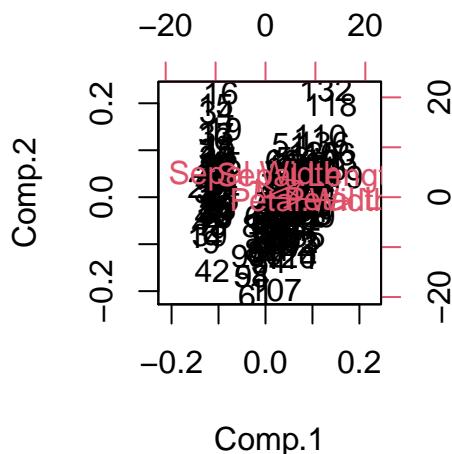
### 91.1 Run PCA

- NOTE: there is also a similar function to `princomp()`, `prcomp()`

```
#"[,-5]" drops that last columns
##the species code
iris.pca <- princomp(iris[,-5])
```

### 91.2 Plot Biplot in base graphics

```
biplot(iris.pca)
```



There are also ggplot biplot code somewhere out there on the internet.

# Chapter 92

## Do PCA with vegan::rda

- rda = redundancy analysis
- normally RDA is used for “constrained ordination” (ordination w/covariates or predictor)
- without predictors, RDA is the same as PCA
- in vegan, giving the function rda() dataframe without predictors runs a PCA very similar to princomp

### 92.1 run a vegan PCA with rda()

```
library(vegan)
my.rda <- rda(iris[,-5])

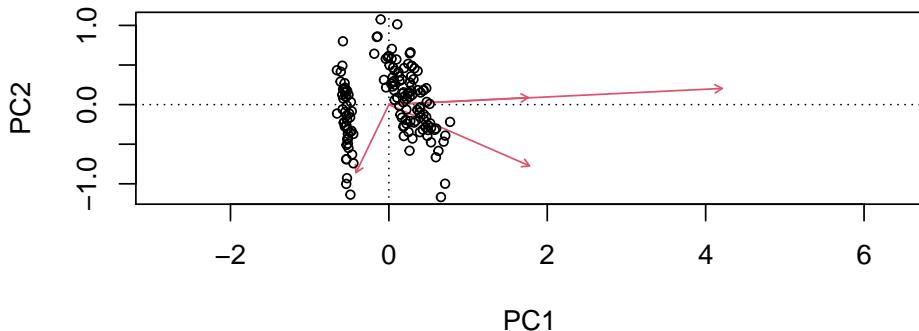
#[, -5] drops that last columns
##the species code
```

### 92.2 Plot the RDA

Note to get help when doing an RDA biplot, call ?biplot.rda; however, use biplot() for the actual plotting.

#### 92.2.1 Default biplot of RDA output

```
biplot(my.rda)
```

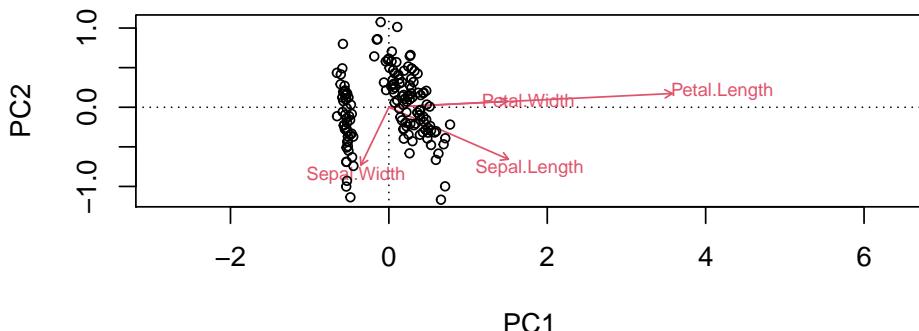


Note that vegan's scaling is different than a normal biplot.

### 92.2.2 biplot.rda options

- vegan is package for ecological analysis and so its functions are oriented towards particular kinds of analyses
- the “display” argument is not in the normal biplot command used with princomp
  - “display = ‘sites’” corresponds to the rows of data; one datapoint per row
  - “display = ‘species’” corresponds to the columns of data; one line/arrow per column
- “type” is an argument also not in biplots originating from princomp
  - “type = ‘text’” plots row numbers and column names as labels
  - “type = points” plots points

```
biplot(my.rda,
 display = c("sites",
 "species"),
 type = c("text",
 "points"))
```



# Chapter 93

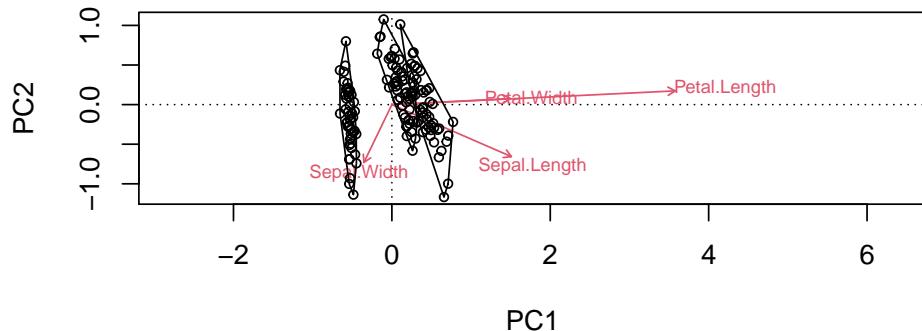
## vegan's ordination plotting functions

- vegan has lots of helpful functions for plotting ordinations.
  - ordihull: convex polygons around groups of points
  - ordiellipse: ellipses around centroids (SD or SE)

```
#2.1 Add "hulls" around each species

#make basic plot
biplot(my.rda,
 display = c("sites",
 "species"),
 type = c("text",
 "points"))

#Add "hulls"
ordihull(my.rda,
 group = iris$Species)
```



```

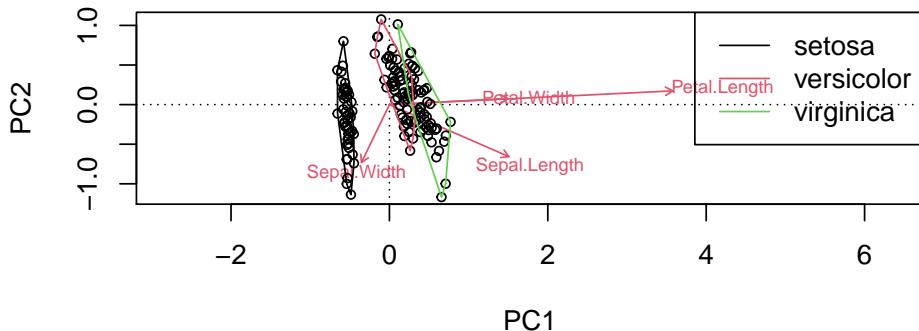
#make basic plot
biplot(my.rda,
 display = c("sites",
 "species"),
 type = c("text",
 "points"))

#get names of species
levels() extracts factor levels
from the column
spp.names <- levels(iris$Species)

#Add hulls
ordihull(my.rda,
 group = iris$Species,
 col = c(1,2,3))

#Add legend
legend("topright",
 col = c(1,2,3),
 lty = 1,
 legend = spp.names)

```



### 93.2.1 with ggplot2

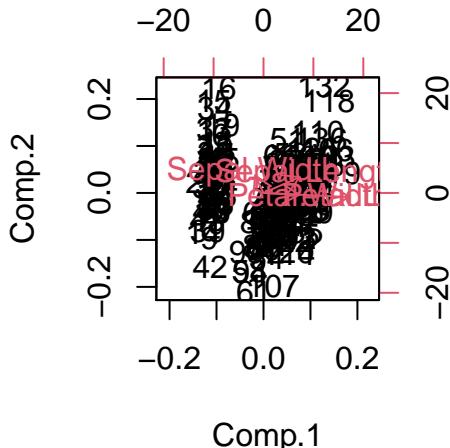
```
library(vegan)

data("iris")

head(iris)

Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1 3.5 1.4 0.2 setosa
2 4.9 3.0 1.4 0.2 setosa
3 4.7 3.2 1.3 0.2 setosa
4 4.6 3.1 1.5 0.2 setosa
5 5.0 3.6 1.4 0.2 setosa
6 5.4 3.9 1.7 0.4 setosa
```

```
biplot(princomp(iris[,-5]))
```

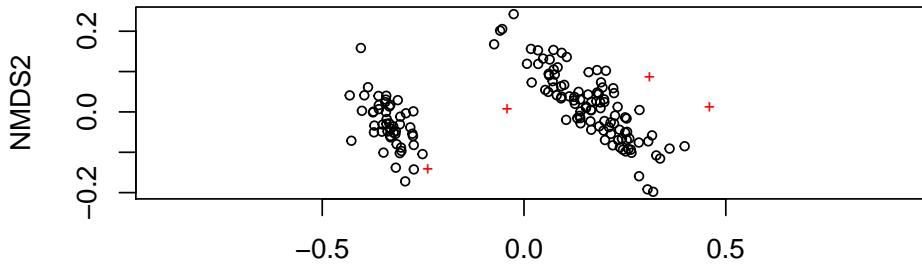


```
iris.nmds <- metaMDS(iris[,-5], k = 3,
 distance = "bray")
```

```

Run 0 stress 0.02621503
Run 1 stress 0.02707938
Run 2 stress 0.02639755
... Procrustes: rmse 0.002913562 max resid 0.02079714
Run 3 stress 0.0277647
Run 4 stress 0.02804685
Run 5 stress 0.02713272
Run 6 stress 0.02758246
Run 7 stress 0.0262832
... Procrustes: rmse 0.002167095 max resid 0.02050869
Run 8 stress 0.02729978
Run 9 stress 0.02711306
Run 10 stress 0.02756013
Run 11 stress 0.02626883
... Procrustes: rmse 0.001797375 max resid 0.02028614
Run 12 stress 0.02803615
Run 13 stress 0.02758615
Run 14 stress 0.02642399
... Procrustes: rmse 0.003370143 max resid 0.02042885
Run 15 stress 0.02874337
Run 16 stress 0.02706142
Run 17 stress 0.02628306
... Procrustes: rmse 0.00444814 max resid 0.03509845
Run 18 stress 0.02685179
Run 19 stress 0.02817189
Run 20 stress 0.02639021
... Procrustes: rmse 0.002934925 max resid 0.03440355
*** No convergence -- monoMDS stopping criteria:
17: no. of iterations >= maxit
3: scale factor of the gradient < sfgrmin
ordiplot(iris.nmds)

```



```

data.scores <- as.data.frame(scores(iris.nmds)) #Using the scores function from vegan
data.scores$site <- rownames(data.scores) # create a column of site names, from the r

```

```

data.scores$grp <- iris$Species # add the grp variable created earlier
head(data.scores) #look at the data

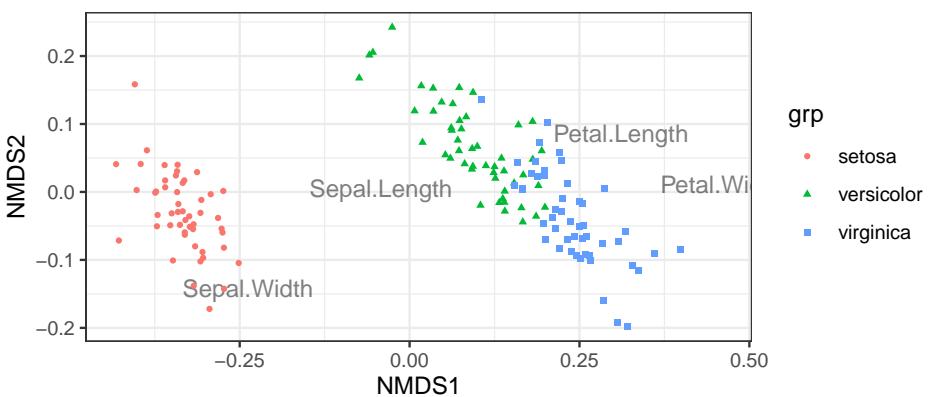
NMDS1 NMDS2 NMDS3 site grp
1 -0.3379574 -0.048440775 0.005385290 1 setosa
2 -0.3422254 0.030486914 -0.027724372 2 setosa
3 -0.3742305 -0.001107934 -0.050632825 3 setosa
4 -0.3597269 0.017007292 -0.062590798 4 setosa
5 -0.3522222 -0.048971752 0.009293821 5 setosa
6 -0.2732540 -0.081992936 0.080747624 6 setosa

species.scores <- as.data.frame(scores(iris.nmds, "species")) #Using the scores function from v
species.scores$species <- rownames(species.scores) # create a column of species, from the rowna
head(species.scores) #look at the data

NMDS1 NMDS2 NMDS3 species
Sepal.Length -0.04207674 0.005187992 0.02941634 Sepal.Length
Sepal.Width -0.23785738 -0.142109787 0.02088258 Sepal.Width
Petal.Length 0.31123683 0.086623797 -0.02858720 Petal.Length
Petal.Width 0.45928423 0.010939488 -0.10889289 Petal.Width

ggplot() +
 geom_text(data=species.scores,
 aes(x=NMDS1,y=NMDS2,
 label=species),alpha=0.5) + # add the species labels
 geom_point(data=data.scores,
 aes(x=NMDS1,y=NMDS2,shape=grp,colour=grp),size=1) + # add the point markers
 #geom_text(data=data.scores,aes(x=NMDS1,y=NMDS2,label=site),size=6,vjust=0) + # add the site l
 #scale_colour_manual(values=c("A" = "red", "B" = "blue")) +
 coord_equal() +
 theme_bw()

```





# Chapter 94

## Hierarchical clustering with Euclidean distances

```
library(compbio4all)
```

### 94.1 Preliminaries

#### 94.1.1 Download packages

Only do this if needed

```
install.packages("ggplot2")
install.packages("ggpubr")
```

#### 94.1.2 Install packages

```
library(ggplot2)
library(ggpubr)
```

### 94.2 Vocab

using color to display a third dimension, distance, Euclidean distance, numeric data, binary data, categorical data, dist()

### 94.3 Note: This is NOT a real analysis

This is a toy analysis to get accross a specific point about distances. The data are real and there are interesting biological questions about these data, but this is not a way explore them.

### 94.4 Load data

From <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2651158/table/T1/>

- GC = the G+C% of the genome
- mb = the size of the genome in megabase
- spp = specifif epithet of the organism
- chromosomes
- genes = predicted number of genes in the genome

```
data
GC <- c(42.3, 37.5, 22.6, 19.4)
mb <- c(26.8, 23.5, 23.1, 23.3)
genes <- c(5433, 5188, 5875, 5403)
chromosomes <- c(14, 14, 14, 14)
spp <- c("vivax", "knowlesi", "yoelii", "falciparum")

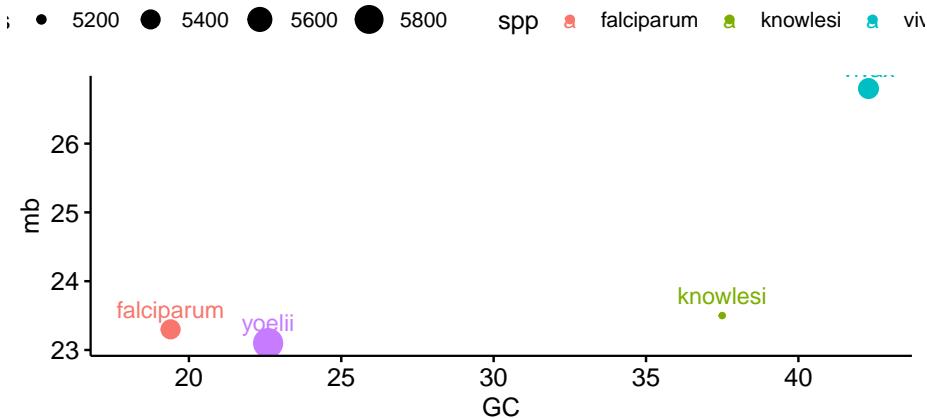
make dataframe
plasmo <- data.frame(spp, mb, GC, genes)

label rows
row.names(plasmo) <- spp
```

### 94.5 Plot the data

A third dimenions of the data is added by using size = genes.

```
ggscatter(y = "mb",
 x = "GC",
 color = "spp",
 size = "genes",
 data = plasmo,
 label = "spp",
 xlim = c(18, 42.5))
```



## 94.6 Distances and distance matrices

The first step in cluster analysis and many related exploratory methods is to select a measure of **distance**. There are many ways to calculate distances, and the proper choice depends on the type of data and the analysis.

- When data are **numeric** (measured with a scale, calipers, rulers and/or can take on decimal values, and/or can be described by Bell-curve/Normal distribution) **Euclidean distances** may be appropriate.
- When data are **binary** (0 or 1, present or absent), **categorical** (A, T, C, or G) other methods will almost definitely be better.
- When data are **counts** it will depend. When counts are relatively low and can contain zero (number of offspring) you may have to be careful. When counts are large (number of RNA transcripts) it may be just fine.
- Each field (ecology, phylogenetics, genomics) has best practices about selecting the proper distance.
- It can be tempting to try out different distance methods to see which one gives you the results you like. This is a form of p-hacking: manipulating your data, analysis, etc to get the statistical result or graph you want.

<https://bitesizebio.com/31497/guilty-p-hacking/#:~:text=The%20term%20p%2Dhacking%2C%20coined,decisions%20of%20the%20analyst>

## 94.7 Calculate Euclidean distance

Euclidean distance is what we learned in high-school geometry

$$a^2 + b^2 = c^2$$

We'll use this notation below:  $h^2 + v^2 = d^2$

- $h$  = the horizontal distance between 2 points
- $v$  = vertical distance between 2 point
- $d$  = the diagonal Euclidean distance between 2 points

Euclidean distance extend intuitively to three dimensions. It can also be extended to any number of dimensions with no modification to the the math, though its much harder to visualize.

#### 94.7.1 Distance from vivax to knowlesi

First, calculate the horizontal distance (x-axis; GC)

The data are

```
plasmo[c(1,2),]
```

```
spp mb GC genes
vivax vivax 26.8 42.3 5433
knowlesi knowlesi 23.5 37.5 5188
```

GC content is on the x-axis. Vivax has a GC content of 42.3. knowlesi has a GC content of 37.5 The horizontal distance is

**42.3–37.5**

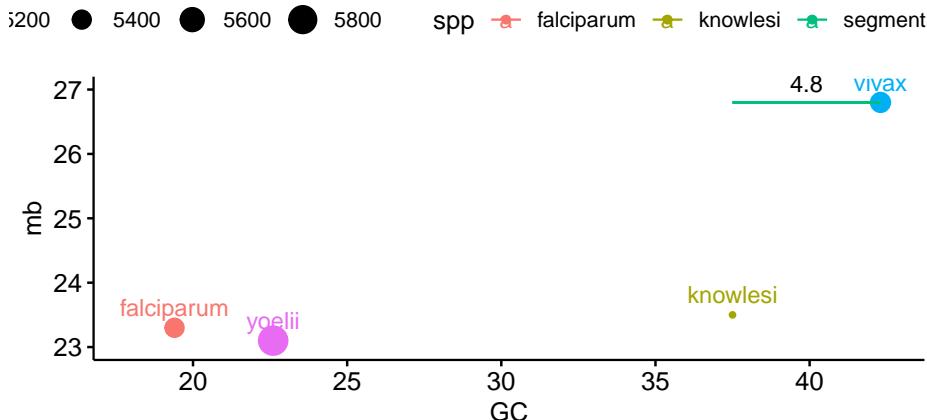
```
[1] 4.8
```

Let's call this distance  $h$

**h <- 42.3–37.5**

The horizontal distance between the points can be visualized like this

```
ggscatter(y = "mb",
 x = "GC",
 color = "spp",
 size = "genes",
 data = plasmo,
 label = "spp",
 xlim = c(18, 42.5),
 ylim = c(23, 27)) +
 # vivax vs knowlesi
 # horiz line
 geom_segment(aes(y = 26.8, x = 37.5,
 yend = 26.8, xend = 42.3,
 colour = "segment")) +
 annotate("text",
 x = mean(c(42.3,37.5)),
 y = 27.1,
 label = h)
```



Genome size in megabases is on the y-axis. Vivax has a genome size of mb = 26.8 and knowlesi is mb = 23.5. The vertical distance is therefore

**26.8-23.5**

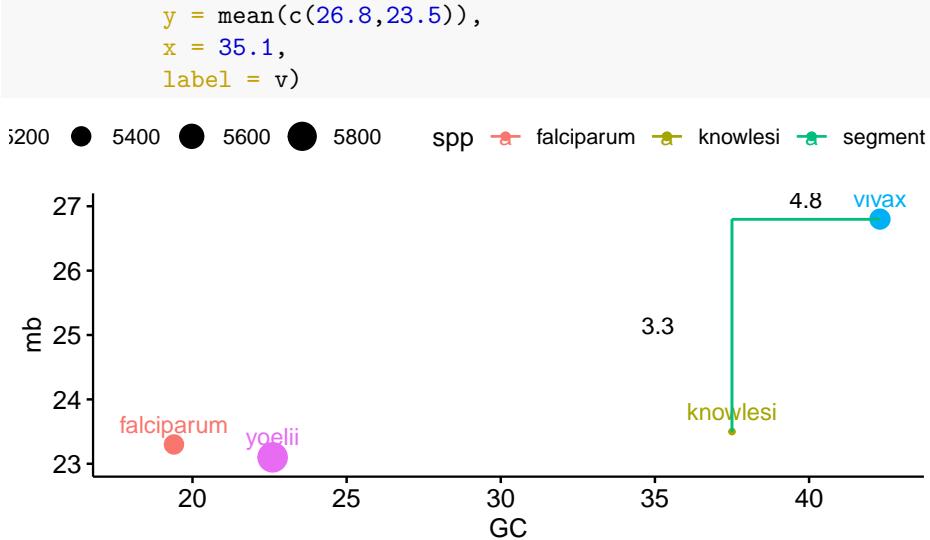
**## [1] 3.3**

Let's call this distance v

v <- 26.8-23.5

We can visualize the vertical and horizontal distances like this

```
ggscatter(y = "mb",
 x = "GC",
 color = "spp",
 size = "genes",
 data = plasmo,
 label = "spp",
 xlim = c(18, 42.5),
 ylim = c(23, 27)) +
 # vivax vs knowlesi
 # horiz line
 geom_segment(aes(y = 26.8, x = 37.5,
 yend = 26.8, xend = 42.3,
 colour = "segment")) +
 annotate("text",
 x = mean(c(42.3,37.5)),
 y = 27.1,
 label = h) +
 # vert line
 geom_segment(aes(y = 23.5, x = 37.5,
 yend = 26.8, xend = 37.5,
 colour = "segment")) +
 annotate("text",
```



The Euclidean distance is

```

d2 <- h^2 + v^2
d <- sqrt(d2)
d
```

```
[1] 5.824946
```

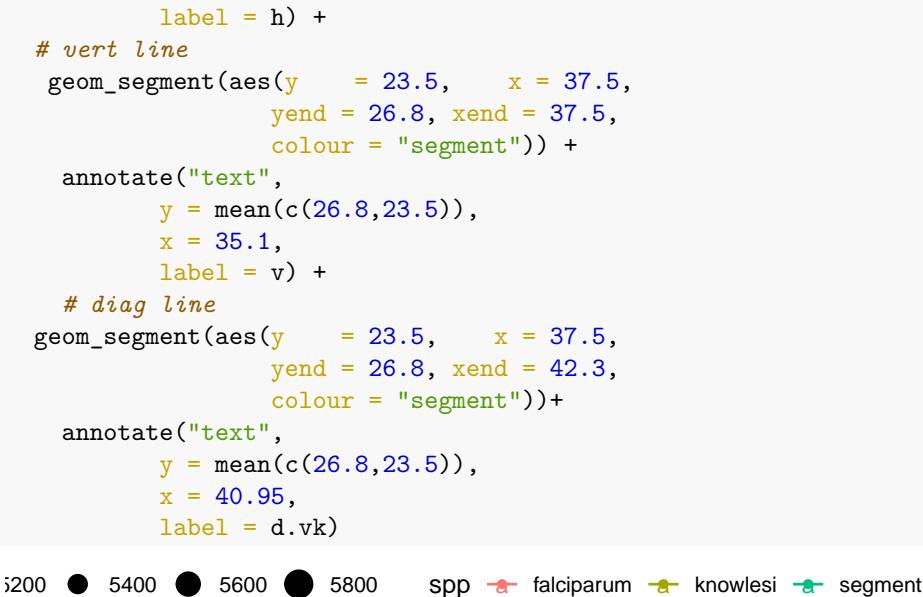
Round it off and call it “d.vk” for “vivax to knowlesi”.

```
d.vk <- round(d, 3)
```

We can visualize this

```

ggscatter(y = "mb",
 x = "GC",
 color = "spp",
 size = "genes",
 data = plasmo,
 label = "spp",
 xlim = c(18, 42.5),
 ylim = c(23, 27)) +
 # vivax vs knowlesi
 # horiz line
 geom_segment(aes(y = 26.8, x = 37.5,
 yend = 26.8, xend = 42.3,
 colour = "segment")) +
 annotate("text",
 x = mean(c(42.3,37.5)),
 y = 27.1,
```



If you want to be fancy, you can calculate the Euclidean distance in one step like this

```
sqrt((42.3-37.5)^2 + (26.8-23.5)^2)
```

```
[1] 5.824946
```

#### 94.7.2 Distance from falciparum to yoelli

These distance calculations are carried out pairwise for each pair of species. For example, the data from falciparum and yeoli are

```
plasmo[c(3,4),]
```

```
spp mb GC genes
```

## 450CHAPTER 94. HIEARCHICAL CLUSTERING WITH EUCLIDEAN DISTANCES

```
yoelii yoelii 23.1 22.6 5875
falciparum falciparum 23.3 19.4 5403
```

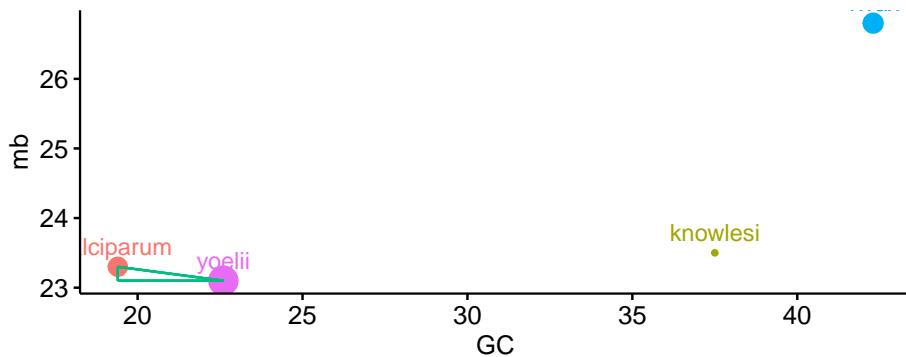
You can work the math out step by step yourself In one line its this, saved to d.fy for distance to falciparum to yoeli

```
d.fy <- sqrt((23.1-23.3)^2 + (22.6 - 19.4)^2)
```

We can visualized the horizontal, vertical, and diagonal (Euclidean) distances like this

```
ggscatter(y = "mb",
 x = "GC",
 color = "spp",
 size = "genes",
 data = plasmo,
 label = "spp") +
falciparum vs yoelii
horiz line
geom_segment(aes(y = 23.1, x = 22.6,
 yend = 23.1, xend = 19.4,
 colour = "segment")) +
vert line
geom_segment(aes(y = 23.1, x = 19.4,
 yend = 23.3, xend = 19.4,
 colour = "segment")) +
diag line
geom_segment(aes(y = 23.1, x = 22.6,
 yend = 23.3, xend = 19.4,
 colour = "segment"))
```

5200 ● 5400 ● 5600 ● 5800 spp — falciparum — knowlesi — segment



## 94.8 Distance knowlesi to yoelii

This would be repeated for all 6 possible comparisons. For knowlesi to yeolii it is

```
knowlesi vs yoelii
plasmo[c(2,3),]

spp mb GC genes
knowlesi knowlesi 23.5 37.5 5188
yoelii yoelii 23.1 22.6 5875
d.ky <- sqrt((23.5-23.1)^2 + (37.5 - 22.6)^2)
```

## 94.9 Distances with the dist() function

Compare my math to the dist() function. See the help file for other distance metrics.

```
d.vk

[1] 5.825

d.fy

[1] 3.206244

d.ky

[1] 14.90537

dist(plasmo[,c("GC","mb")],method = "euclidean")

vivax knowlesi yoelii
knowlesi 5.824946
yoelii 20.044451 14.905368
falciparum 23.165923 18.101105 3.206244
```

Save the distance matrix

```
d_mat <- dist(plasmo[,c("GC","mb")],method = "euclidean")
```

## 94.10 Forming the first cluster

The first cluster is formed by combining the two closest items. In this case we are considering taxa, though the data we are working with would never be used to build an acutal phylogenetic tree.

The minimum distance is yeolii to falciparum

```
d.fy
[1] 3.206244
```

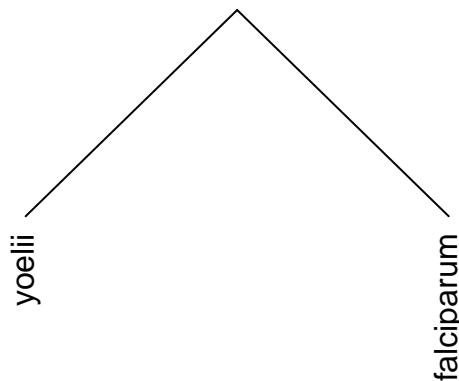
The minimum function works on a matrix like this

```
min(d_mat)
```

```
[1] 3.206244
```

The distance between the cluster is just this distance. Cluster analysis typically assumes a perfectly bifurcating (branching) tree. This looks like this

```
d_mat_temp <- dist(plasmo[c(3,4),c("GC", "mb")], method = "euclidean")
clust_temp <- as.dendrogram(hclust(d_mat_temp))
plot(clust_temp, type = "triangle", axes = F)
```



# Chapter 95

## Cluster analysis with allometric data

TODO: swap out iris data

```
library(compbio4all)
```

### 95.1 Cluster analysis

The PCA we did on the mammal sleep data didn't indicate any interesting groups. Instead lets work with a famous dataset with known strong groups.

Read about these data here [https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)

You should definitely read this as further background on PCA and cluster analysis.

For more on PCA see <https://rpubs.com/brouwern/veganpca>

Notes on drawing dendograms: <http://www.sthda.com/english/wiki/beautiful-dendrogram-visualizations-in-r-5-must-known-methods-unsupervised-machine-learning#plot.dendrogram-function>

### 95.2 Preliminaries

#### 95.2.1 Download packages

Only do this once, then comment out of the script. You probably already did this in the previous Code Checkpoint.

```
install.packages("ggplot2")
install.package("vegan")
```

### 95.2.2 Load the libraries

```
library(ggplot2)
library(vegan)
```

## 95.3 PCA data exploration

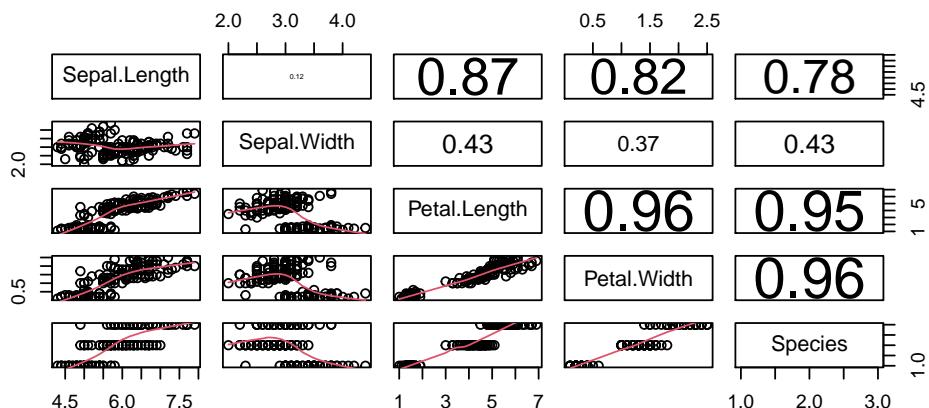
```
data(iris)
```

Scatterplot matrix

```
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)
{
 usr <- par("usr"); on.exit(par(usr))
 par(usr = c(0, 1, 0, 1))
 r <- abs(cor(x, y))
 txt <- format(c(r, 0.123456789), digits = digits)[1]
 txt <- paste0(prefix, txt)
 if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
 text(0.5, 0.5, txt, cex = cex.cor * r)
}
```

When looking at a plot like this you should understand how to read it, what the correlations mean, what the red lines mean, etc.

```
plot(iris, upper.panel = panel.cor,
 panel = panel.smooth)
```

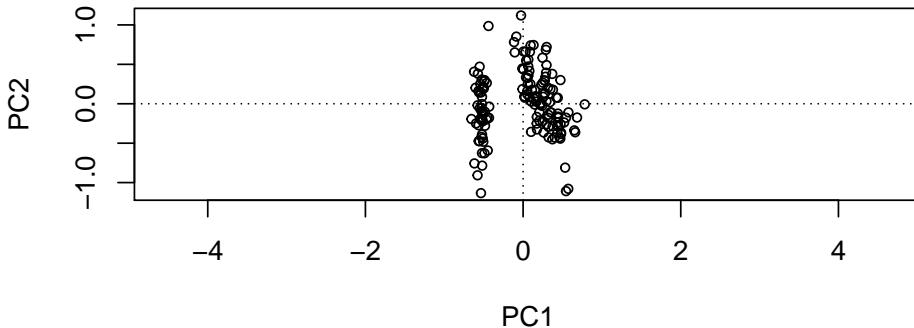


Run the PCA using rda()

```
rda.out <- vegan::rda(iris[,-5], scale = TRUE)
```

This displays the 2D PCA plot without the arrows. For more info on what this code does, see the RPubs document linked above

```
biplot(rda.out, display = "sites")
```



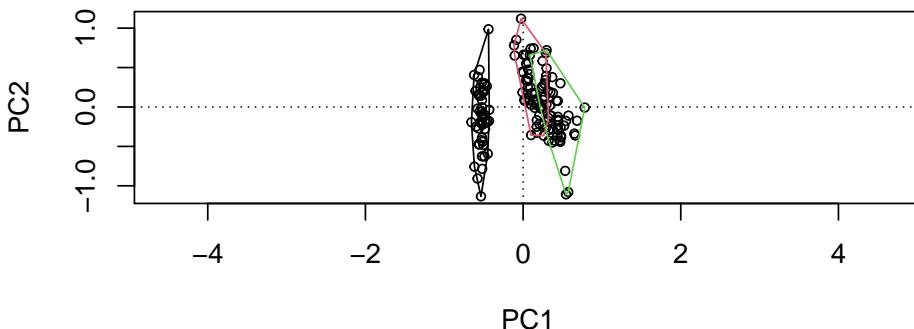
vegan has some nice tools for groups things.

In this dataset I don't expect there to be any interest groups, but I'll check anyway. I will supply this code if needed.

PCA is an exploratory method. First I'll see if there are any groupings based on diet ("vore"). Not really

```
biplot(rda.out, display = "sites")
```

```
vegan::ordihull(rda.out,
 group = iris$Species,
 col = 1:3)
```



### 95.3.1 Hierarchical cluster analysis

Cluster analysis in biology often involves building tree diagrams (dendrograms), which uses hierarchical cluster algorithms.

Hierarchical clustering involves first calculating a distance matrix. There are MANY ways to calculate a distance matrix depending on the data and application. The easiest one to think about is Euclidean distance.

Add row names. Don't worry about what this is doing

```
row.names(iris) <- paste(iris$Species, 1:nrow(iris), sep = ".")
```

There's a TON of data here so let's randomly cut it in half. You should understand this code

```
i <- 1:nrow(iris)
i <- sample(i, length(i)/2, replace = F)
iris_sub <- iris[i,]

dist_euc <- dist(iris_sub[,-5],
 method = "euclidean")
```

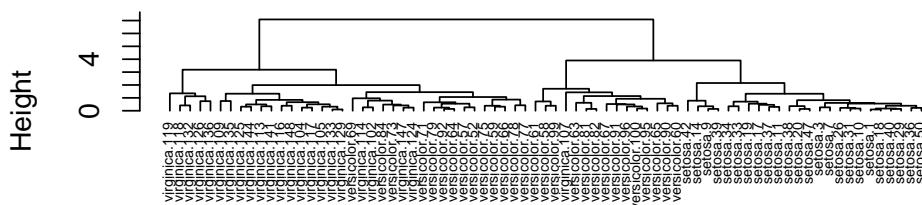
We can then carry out a cluster analysis using the `hclust()` function. The default function used for calculating branch lengths is called "complete." Other options allow you to implement UPGMA, WPGMA and other common forms.

```
clust_euc <- hclust(dist_euc)
```

Plotting the dendrogram results in a very messy graph

```
plot(clust_euc, hang = -1, cex = 0.5)
```

**Cluster Dendrogram**



dist\_euc  
hclust (\*, "complete")

It can help in R to plot things horizontally. This requires converting to a different object type.

```
par(mar = c(1, 1, 1, 1))
is(clust_euc)

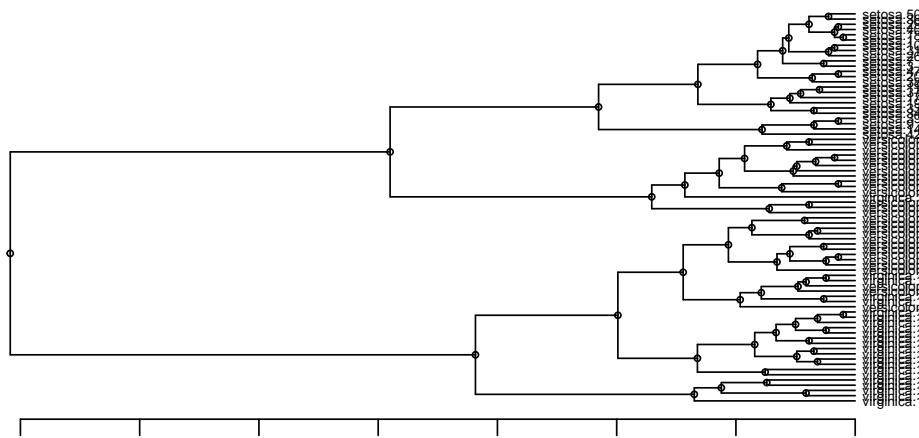
[1] "hclust"
```

```
dendro_euc <- as.dendrogram(clust_euc)

is(dendro_euc)

[1] "dendrogram"

plot(dendro_euc,horiz = T,nodePar = list(pch = c(1,NA),
 cex = 0.5,
 lab.cex = 0.5))
```



## 95.4 Task

Generate the cluster diagram and upload it to this assignment

<https://canvas.pitt.edu/courses/45284/assignments/460774>



# **Chapter 96**

## **Higgs and Atwood student version with updates**

Add your name to “author” above

NOTE 1: When you submit this please remove all “TODO” prompts, NOTES, instructions. etc by me. THe final product should look like a blog post, NOT a homework assignment

NOTE 2: Each code chunk should have a line that indicates what is being done, like an instructional blog post.

### **96.1 Introduction**

WRITING TODO: Write a 4-5 sentence introduce describing what is done by this script and why we care about it biologically.

### **96.2 Preliminaries**

#### **96.2.1 Packages**

1. CODE-TODO: load ALL packages
2. COMMENT-TODO: add a comment to indicate what each packages does
3. WRITING-TODO: describe in 1-2 sentences the relationship between ggpubr R and ggplot2.

#### **96.2.2 Build the dataframe**

1. CODE-TODO: build the amino acid dataframe I provided.
2. WRITING-TODO: write 2-3 sentences about what the data represent.

## 460CHAPTER 96. HIGGS AND ATWOOD STUDENT VERSION WITH UPDATES

3. TODO: read the comments I wrote in the code; you will want to refer to these later

These data are derived mostly from Higgs (2009) Table 1. They are similar to Higgs and Attwood 2005 but have extra columns. Nathan Brouwer added additional categorical variables based on online information.

Make the vectors.

```
1 letter code
aa <-c('A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N',
 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y')

molecular weight in dalton
MW.da <-c(89, 121, 133, 146, 165, 75, 155, 131, 146, 131, 149, 132, 115, 147, 174, 105, 119, 117, 204)

vol from van der Waals radii
vol <-c(67, 86, 91, 109, 135, 48, 118, 124, 135, 124, 124, 96, 90,
 114, 148, 73, 93, 105, 163, 141)

bulk - a measure of the shape of the side chain
bulk <-c(11.5, 13.46, 11.68, 13.57, 19.8, 3.4, 13.69, 21.4,
 15.71, 21.4, 16.25, 12.28, 17.43,
 14.45, 14.28, 9.47, 15.77, 21.57, 21.67, 18.03)

pol - a measure of the electric field strength around the molecule
pol <-c(0, 1.48, 49.7, 49.9, 0.35, 0, 51.6, 0.13, 49.5, 0.13,
 1.43, 3.38, 1.58, 3.53, 52, 1.67, 1.66, 0.13, 2.1, 1.61)

isoelec point
isoelec <-c(6, 5.07, 2.77, 3.22, 5.48, 5.97, 7.59, 6.02, 9.74, 5.98,
 5.74, 5.41, 6.3, 5.65, 10.76, 5.68, 6.16, 5.96, 5.89, 5.66)

1st Hydrophobicity scale
H20pho.34 <-c(1.8, 2.5, -3.5, -3.5, 2.8, -0.4, -3.2, 4.5, -3.9, 3.8, 1.9,
 -3.5, -1.6, -3.5, -4.5, -0.8, -0.7, 4.2, -0.9, -1.3)

2nd Hydrophobicity scale
H20pho.35 <-c(1.6, 2, -9.2, -8.2, 3.7, 1, -3, 3.1, -8.8, 2.8, 3.4, -4.8,
 -0.2, -4.1, -12.3, 0.6, 1.2, 2.6, 1.9, -0.7)

Surface area accessible to water in an unfolded peptide
saaH20 <-c(113, 140, 151, 183, 218, 85, 194, 182, 211, 180, 204, 158,
 143, 189, 241, 122, 146, 160, 259, 229)
```

```

Fraction of accessible area lost when a protein folds
faal.fold <-c(0.74,0.91,0.62,0.62,0.88,0.72,0.78,0.88,0.52,
 0.85,0.85,0.63,0.64,0.62,0.64,0.66,0.7,0.86,0.85,0.76)

Polar requirement
polar.req <-c(7,4.8,13,12.5,5,7.9,8.4,4.9,10.1,4.9,5.3,10,
 6.6,8.6,9.1,7.5,6.6,5.6,5.2,5.4)

relative frequency of occurrence
"The frequencies column shows the mean
percentage of each amino acid in the protein sequences
of modern organisms"
freq <-c(7.8,1.1,5.19,6.72,4.39,6.77,2.03,6.95,6.32,
 10.15,2.28,4.37,4.26,3.45,5.23,6.46,5.12,7.01,1.09,3.3)

##
charges
un = Un-charged
neg = negative
pos = positive
charge<-c('un','un','neg','neg','un','un','pos','un','pos',
 'un','un','un','un','un','pos','un','un','un','un')

hydropathy
hydropathy<-c('hydrophobic','hydrophobic','hydrophilic',
 'hydrophilic','hydrophobic','neutral','neutral',
 'hydrophobic','hydrophilic','hydrophobic','hydrophobic',
 'hydrophilic','neutral',
 'hydrophilic','hydrophilic','neutral','neutral',
 'hydrophobic','hydrophobic','neutral')

vol
vol.cat<-c('verysmall','small','small','medium',
 'verylarge','verysmall','medium','large','large',
 'large','large','small','small','medium','large',
 'verysmall',
 'small','medium','verylarge','verylarge')

pol
pol.cat<-c('nonpolar','nonpolar','polar','polar',
 'nonpolar','nonpolar','polar','nonpolar',
 'polar','nonpolar','nonpolar','polar','nonpolar','polar',
 'polar','polar','polar','nonpolar','nonpolar','polar')

```

```
chemical
chemical<-c('aliphatic','sulfur','acidic','acidic','aromatic',
 'aliphatic','basic','aliphatic','basic','aliphatic','sulfur',
 'amide','aliphatic','amide','basic','hydroxyl','hydroxyl',
 'aliphatic','aromatic','aromatic')
```

Build the dataframes.

```
Build dataframe
aa_dat <- data.frame(aa,
 MW.da, vol,
 bulk, pol,
 isoelec, H2Opho.34, H2Opho.35,
 saaH2O, faal.fold, polar.req,
 freq, charge, hydropathy,
 vol.cat, pol.cat, chemical)
```

## 96.3 Raw data exploration

### 96.3.1 XXXXXXXX

TODO: Examine the code below. What is it called? Replace the XXXXXXXX above with the name of this thing

```
cor_ <- round(cor(aa_dat[,-c(1,13:17)]),2)
diag(cor_) <- NA
cor_[upper.tri(cor_)] <- NA
cor_
```

	MW.da	vol	bulk	pol	isoelec	H2Opho.34	H2Opho.35	saaH2O	faal.fold
## MW.da	NA	NA	NA	NA	NA	NA	NA	NA	NA
## vol	0.93	NA	NA	NA	NA	NA	NA	NA	NA
## bulk	0.55	0.73	NA	NA	NA	NA	NA	NA	NA
## pol	0.29	0.24	-0.20	NA	NA	NA	NA	NA	NA
## isoelec	0.20	0.36	0.08	0.27	NA	NA	NA	NA	NA
## H2Opho.34	-0.27	-0.08	0.44	-0.67	-0.20	NA	NA	NA	NA
## H2Opho.35	-0.25	-0.16	0.32	-0.85	-0.26	0.85	NA	NA	NA
## saaH2O	0.97	0.99	0.64	0.29	0.35	-0.18	-0.23	NA	NA
## faal.fold	0.11	0.18	0.49	-0.53	-0.18	0.84	0.79	0.12	NA
## polar.req	-0.05	-0.19	-0.53	0.76	-0.11	-0.79	-0.87	-0.11	-0.81
## freq		-0.52	-0.30	-0.04	-0.01	0.02	0.26	-0.02	-0.38
## polar.req freq									-0.18
## MW.da		NA	NA						
## vol		NA	NA						
## bulk		NA	NA						
## pol		NA	NA						

```
isoelec NA NA
H2Opho.34 NA NA
H2Opho.35 NA NA
saaH2O NA NA
faal.fold NA NA
polar.req NA NA
freq 0.14 NA
```

WRITING-TODO: Describe what the code below is doing and summarzie the next 2 TODOs in 2-3 sentences. TODO: Which variables have the most positive correlation? TODO: What have most negative correlation?

```
which(cor_ == max(cor_, na.rm = T), arr.ind = T)
```

```
row col
saaH2O 8 2
max(cor_, na.rm = T)

[1] 0.99
min(cor_, na.rm = T)
```

```
[1] -0.87
which(cor_ == min(cor_, na.rm = T), arr.ind = T)
```

```
row col
polar.req 10 7
```

### 96.3.2 Plot 0: Scatterplot matrix

**Code TODO:** Make a scatterplot matrix of all of the data. Included the following variables:

1. MW.da
2. vol
3. pol
4. H2Opho.34
5. polar.req

PLUS 2 others (6 total)

NOTE: Depending on how you do this you may have to make sure the first column doesn't go into the plotting command

**TODO:** Write 1-2 sentences describing what this below code does in your scatter plot matrix below

```
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor,...)
{
 usr <- par("usr"); on.exit(par(usr))
```

```

par(usr = c(0, 1, 0, 1))
r <- abs(cor(x, y))
txt <- format(c(r, 0.123456789), digits = digits)[1]
txt <- paste0(prefix, txt)
if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
text(0.5, 0.5, txt, cex = cex.cor * r)
}

```

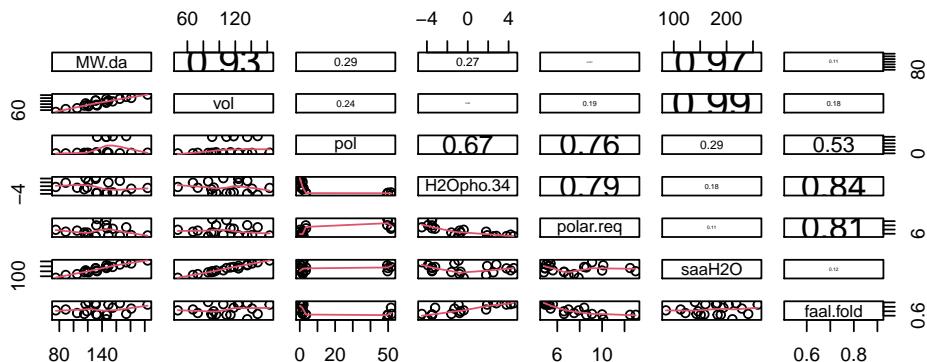
TOD: WRITE 2-3 sentences describing the relationships between the 6 variables you plotted; which have strong correlations which ones weak, which ones have non-linear relationships.

```

plot(aa_dat[,c("MW.da", "vol", "pol",
 "H2Opho.34", "polar.req",
 "saaH2O", "faal.fold")],
 panel = panel.smooth,
 upper.panel = panel.cor ,
 main = "Example")

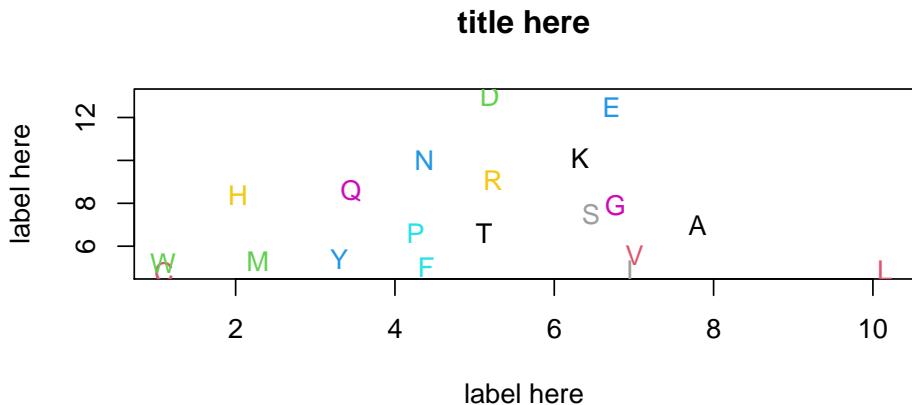
```

### Example



### 96.3.3 Plot 1: Replication Higgs and Attwood Figure 2.8

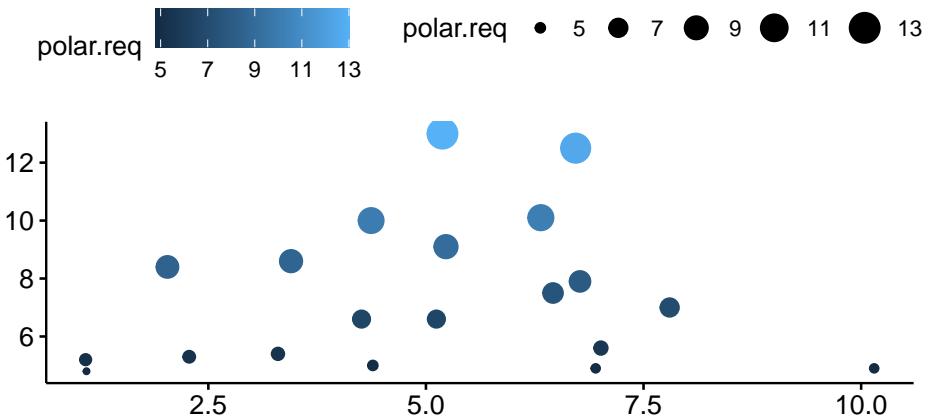
1. CODE-TODO: make a plot that matches Figure 2.8 on page 26 of the Higgs and Atwood chapter 2. Make sure you have plotted the correct axes
2. CODE-TODO: label the x and y axes.
3. COMMENT-TODO: indicate what is doing the main plotting and what is adding the text labels.
4. WRITING-TODO: Write 1-2 sentences describing what the plot represents



96.3.4 Figure 2: HA Figure 2.8 with ggpubr

1. TODO: Make a ggpubr version of HA Figure 2.8. Additionally, use both COLOR and SHAPE to represent 2 additional NUMERIC (NOT categorical) variables of your choice.
2. TODO: label the axes with the full name of the variables

```
ggscatter(y = "polar.req",
 x = "freq",
 size = "polar.req",
 color = "polar.req",
 data = aa_dat,
 xlab = "",
 ylab = "")
```



96.3.5 Figure 3: Highly correlated data

CODE:

1. TODO: Identify 2 variables that are highly positively correlated and plot them in GGPUBR
2. TODO: add a regression line
3. TODO: add a data ellipse
4. TODO: add a correlation coefficient
5. TODO: label the axes with the full name of the variables

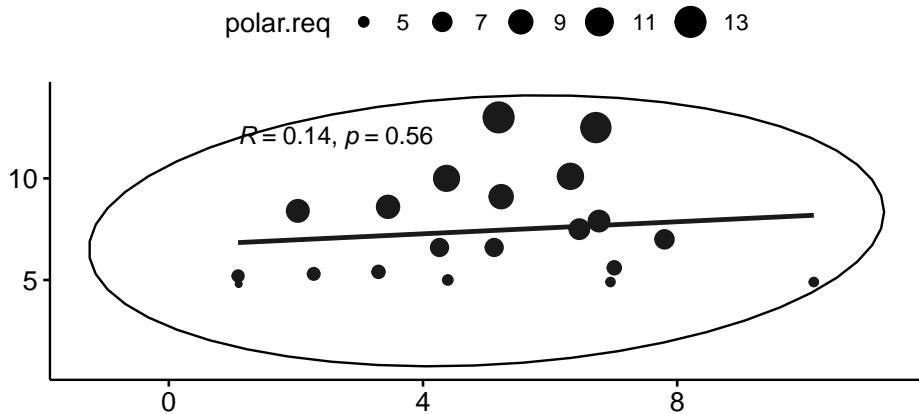
## COMMENTS:

1. TODO: add a comment indicating which line of the ggpibr is adding the line
2. TODO: add a comment indicating which line of the ggpibr is adding the ellipse TODO: add a comment indicating which line of the ggpibr is adding the correlation coefficient

## WRITING

- 1.TODO: describe the general mathematical form of the regression line in 1-2 sentences 1.TODO: describe the general meaning of the data ellipse in 1-2 sentences

```
ggscatter(y = "polar.req",
 x = "freq",
 size = "polar.req",
 add = "reg.line", # line of best fit
 ellipse = TRUE, # data ellipse
 cor.coef = TRUE, # correlation coef
 # color = "freq",
 data = aa_dat,
 xlab = "",
 ylab = "")
```



96.3.6 Figure 4: Non-linear relationship

CODE:

1. TODO: Identify 2 variables that have a non-linear relationship between them (scatter plot has a curved shape).
2. TODO: Use ggpublisher to plot a scatterplot with a **loess smoother**. Use google or the ggubr help files to figure out how to do this
3. TODO: also vary the size and color of the data by 2 other NUMERIC variables
4. TODO: label the axes with the full name of the variables

### 96.3.7 Figure 5: Non-linear relationship on LOG scale

CODE:

1. TODO: Log-transform both of your the variables you used in Figure 4 make the plot again
2. TODO: a regression line instead of a loess smoother
3. TODO: also vary the size and color of the data by 2 other NUMERIC variables

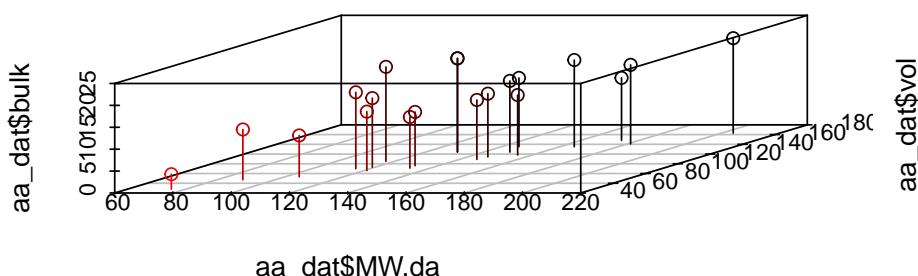
WRITING:

1. TODO: describe what the log transformation does in 1 to 2 sentences.

### 96.3.8 Figure 6: 3D Scatterplot

1. CODE TODO: Make a 3D scatterplot of 3 variables that have relatively strong correlations.
2. CODE TODO: use highlight.3d = TRUE to colored code
3. CODE TODO: Use type="h"
4. WRITING TODO: Summarize the relationships and correlations between the variables you chose in 2-4 sentences.

```
scatterplot3d(x = aa_dat$MW.da,
 y = aa_dat$vol,
 z = aa_dat$bulk,
 highlight.3d = TRUE,
 type="h")
```



## 96.4 Principal component analysis

TODO: Write 3-4 sentences describing the basic use and interpretation of PCA as we have used it in this class. Refer to Higgs and Attwood, and provide a useful quote that explains PCA. Cite the quote as (Higgs and Attwood 2005, pg XX).

TODO: Write 3-4 sentences describing the 2 basic ways we made PCA plots in this class.

### 96.4.1 Data prep

TODO: To make life easier, make a new version of your dataset which drops any categorical variables. Call it aa\_dat2 TODO: Additionally: compare Table 2.2 and the data provided. I give you ~11 numeric variables, while Table 2.2 has only 8. Read the information carefully and remove the extra variables

TODO: Write 1-2 sentences describing the approach you used to make this new dataframe.

```
aa_dat2 <- aa_dat[,-c(1,
 2,11,12,
 13,14,15,16,17)]
```

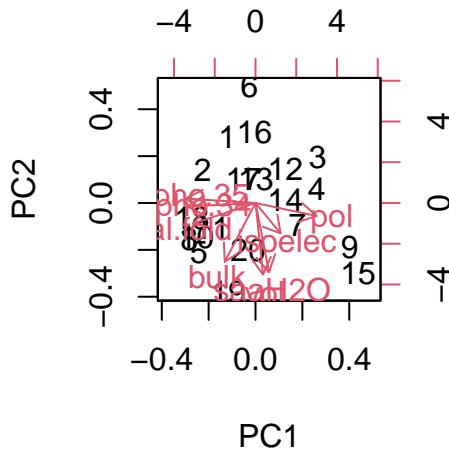
### 96.4.2 Figure 7: Principal components analysis - base R

Run a PCA and create a PCA plot using base R command.

```
pca.out <- prcomp(aa_dat2, scale = TRUE)
```

Plot the output

```
biplot(pca.out)
```



We can carry out the numeric analysis using the `rda()` function from the `vegan` package.

```
rda.out <- vegan::rda(aa_dat2,
 scale = TRUE)
```

We then extract what are known as the PCA score - don't worry about what this means right now, we'll dig into it in a moment. Basically, we're going from 15 columns of data to just 2, which we pull out of `rda.out` with the `scores()` function.

```
rda_scores <- scores(rda.out)
```

### 96.4.3 Principal Components Analysis (PCA)

We can reduce the **dimensionality** of complicated data sets in order to make them easier to visualize. In the case of our amino acid dataset we can go from 15 different variables to just to.

Principal Components Analysis (PCA) is common form of **dimension reduction**. We'll first produce the standard output of a PCA analysis – a **biplot** – before digging into the theory behind what is happening.

We can carry out the numeric analysis using the `rda()` function from the `vegan` package.

**TASK:** Call the `rda()` function on the `aa_dat2` object and save it to the `rda.out` object provided below.

# COMPLETE THIS CODE:

```
rda.out <- ## YOUR CODE GOES HERE ##
```

We then extract what are known as the PCA score - don't worry about what this means right now, we'll dig into it in a moment. Basically, we're going from 15 columns of data to just 2, which we pull out of `rda.out` with the `scores()` function.

**TASK:** Call the `scores()` function on the `rda.out` object produced above; save the output to the `rda_scores` object provided

# COMPLETE THIS CODE:

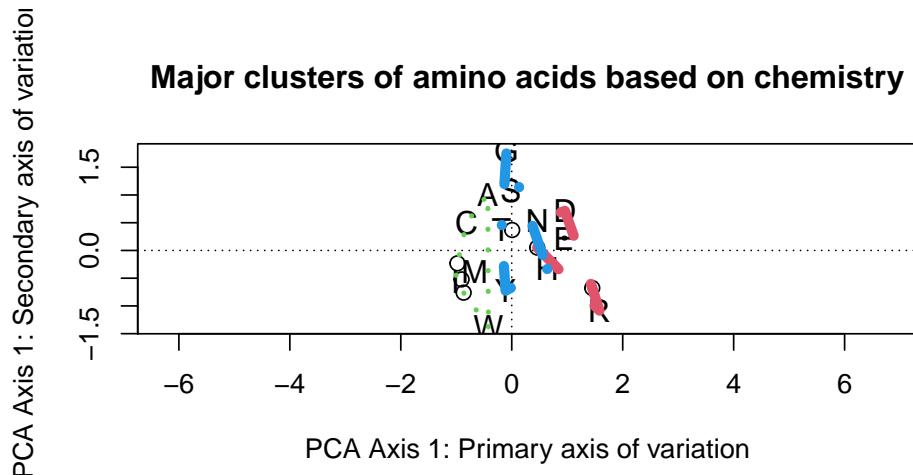
```
rda_scores <- # COMPLETE THIS CODE:
```

Now we'll make a **biplot**. This is similar in some ways to a scatter plot. We'll also add some additional annotation to the plot to help us see meaningful grouping of our amino acids. First, we'll label each amino acids with its 1-letter code. Then we'll delineate all amino acids within pre-specified groups to see if these groups map to the entire set of columns.

```
Build plot
biplot(rda.out, display = "sites", col = 0,
 main = "Major clusters of amino acids based on chemistry",
 xlab = "PCA Axis 1: Primary axis of variation",
 ylab = "PCA Axis 1: Secondary axis of variation")

add point
orditorp(rda.out, display = "sites", labels = aa_dat$aa, cex = 1.2)

add groupings
ordihull(rda.out,
 group = aa_dat$hydropathy,
 col = 1:7,
 lty = 1:7,
 lwd = c(3,6))
```



## 96.5 Cluster analysis

TODO: Write 2-3 sentences describing the purpose of cluster analysis. TODO: Write 3-5 sentences describing how UPGMA works.

### 96.5.1 XXXXXXXXXX cluster analysis

TODO: What type of cluster analysis is UPGMA? Change the XXXXXXXX above to this. NOTE: use the aa\_dat2 data we used above for all of this.

Add row names. Don't worry about what this code is doing

```
row.names(aa_dat2) <- paste(aa_dat$aa,
 #aa_dat$hydropathy,
```

```
sep = ".")
```

CODE-TODO: Create a distance matrix using the `dist()` function. Set the distance to be euclidean using `method = "euclidean"`. Call the output `dist_euc`  
WRITING-TODO: Write 2-3 sentences describing what a distance matrix is.  
WRITING-TODO: write 2-3 sentences describing what Euclidean distance is.

```
dist_euc <- dist(aa_dat2,
 method = "euclidean")
```

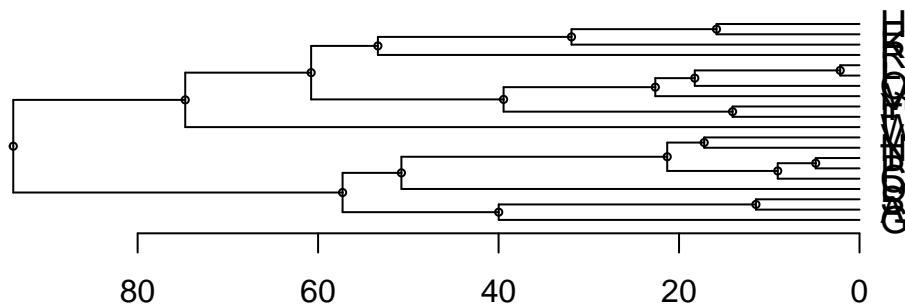
CODE-TODO: Use the `hclust()` function to do cluster analysis. Use the UPGMA algorithm (see the help file for the proper setting)

COMMENT-TODO: Add a comment that indicates what part of the code is setting it to use UPGMA.

```
clust_euc <- hclust(dist_euc,
 method = "average")
```

CODE-TODO: Plot the dendrogram (Optional - convert to a dendrogram as I did) WRITING-TODO: Write 2-5 sentences describing how your cluster diagram compares to the cluster diagram produced by Higgs and Atwood (2005) in Plate 2.2. (This is a cluster diagram that also includes a heat map)

```
dendro_euc <- as.dendrogram(clust_euc)
plot(dendro_euc, horiz = T,
 nodePar = list(pch = c(1,NA),
 cex = 0.5,
 lab.cex = 1.2))
```





# Chapter 97

## Higgs and Atwood student version with updates - copy or alt

Add your name to “author” above

NOTE 1: When you submit this please remove all “TODO” prompts, NOTES, instructions. etc by me. THe final product should look like a blog post, NOT a homework assignment

NOTE 2: Each code chunk should have a line that indicates what is being done, like an instructional blog post.

### 97.1 Introduction

WRITING TODO: Write a 4-5 sentence introduce describing what is done by this script and why we care about it biologically.

### 97.2 Preliminaries

#### 97.2.1 Packages

1. CODE-TODO: load ALL packages
2. COMMENT-TODO: add a comment to indicate what each packages does
3. WRITING-TODO: describe in 1-2 sentences the relationship between ggpubr R and ggplot2.

### 97.2.2 Build the dataframe

1. CODE-TODO: build the amino acid dataframe I provided.
2. WRITING-TODO: write 2-3 sentences about what the data represent.
3. TODO: read the comments I wrote in the code; you will want to refer to these later

These data are derived mostly from Higgs (2009) Table 1. They are similar to Higgs and Attwood 2005 but have extra columns. Nathan Brouwer added additional categorical variables based on online information.

Make the vectors.

```
1 letter code
aa <-c('A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N',
 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'Y')

molecular weight in dalton
MW.da <-c(89, 121, 133, 146, 165, 75, 155, 131, 146, 131, 149, 132, 115, 147, 174, 105, 119, 117, 204)

vol from van der Waals radii
vol <-c(67, 86, 91, 109, 135, 48, 118, 124, 135, 124, 124, 96, 90,
 114, 148, 73, 93, 105, 163, 141)

bulk - a measure of the shape of the side chain
bulk <-c(11.5, 13.46, 11.68, 13.57, 19.8, 3.4, 13.69, 21.4,
 15.71, 21.4, 16.25, 12.28, 17.43,
 14.45, 14.28, 9.47, 15.77, 21.57, 21.67, 18.03)

pol - a measure of the electric field strength around the molecule
pol <-c(0, 1.48, 49.7, 49.9, 0.35, 0, 51.6, 0.13, 49.5, 0.13,
 1.43, 3.38, 1.58, 3.53, 52, 1.67, 1.66, 0.13, 2.1, 1.61)

isoelec point
isoelec <-c(6, 5.07, 2.77, 3.22, 5.48, 5.97, 7.59, 6.02, 9.74, 5.98,
 5.74, 5.41, 6.3, 5.65, 10.76, 5.68, 6.16, 5.96, 5.89, 5.66)

1st Hydrophobicity scale
H20pho.34 <-c(1.8, 2.5, -3.5, -3.5, 2.8, -0.4, -3.2, 4.5, -3.9, 3.8, 1.9,
 -3.5, -1.6, -3.5, -4.5, -0.8, -0.7, 4.2, -0.9, -1.3)

2nd Hydrophobicity scale
H20pho.35 <-c(1.6, 2, -9.2, -8.2, 3.7, 1, -3, 3.1, -8.8, 2.8, 3.4, -4.8,
 -0.2, -4.1, -12.3, 0.6, 1.2, 2.6, 1.9, -0.7)

Surface area accessible to water in an unfolded peptide
```

```

saaH20 <-c(113,140,151,183,218,85,194,182,211,180,204,158,
 143,189,241,122,146,160,259,229)

Fraction of accessible area lost when a protein folds
faal.fold <-c(0.74,0.91,0.62,0.62,0.88,0.72,0.78,0.88,0.52,
 0.85,0.85,0.63,0.64,0.62,0.64,0.66,0.7,0.86,0.85,0.76)

Polar requirement
polar.req <-c(7,4.8,13,12.5,5,7.9,8.4,4.9,10.1,4.9,5.3,10,
 6.6,8.6,9.1,7.5,6.6,5.6,5.2,5.4)

relative frequency of occurrence
"The frequencies column shows the mean
percentage of each amino acid in the protein sequences
of modern organisms"
freq <-c(7.8,1.1,5.19,6.72,4.39,6.77,2.03,6.95,6.32,
 10.15,2.28,4.37,4.26,3.45,5.23,6.46,5.12,7.01,1.09,3.3)

##
charges
un = Un-charged
neg = negative
pos = positive
charge<-c('un','un','neg','neg','un','un','pos','un','pos',
 'un','un','un','un','un','pos','un','un','un','un')

hydropathy
hydropathy<-c('hydrophobic','hydrophobic','hydrophilic',
 'hydrophilic','hydrophobic','neutral','neutral',
 'hydrophobic','hydrophilic','hydrophobic','hydrophobic',
 'hydrophilic','neutral',
 'hydrophilic','hydrophilic','neutral','neutral',
 'hydrophobic','hydrophobic','neutral')

vol
vol.cat<-c('verysmall','small','small','medium',
 'verylarge','verysmall','medium','large','large',
 'large','large','small','small','medium','large',
 'verysmall',
 'small','medium','verylarge','verylarge')

pol

```

```

pol.cat<-c('nonpolar','nonpolar','polar','polar',
 'nonpolar','nonpolar','polar','nonpolar',
 'polar','nonpolar','nonpolar','polar','nonpolar','polar',
 'polar','polar','polar','nonpolar','nonpolar','polar')

chemical
chemical<-c('aliphatic','sulfur','acidic','acidic','aromatic',
 'aliphatic','basic','aliphatic','basic','aliphatic','sulfur',
 'amide','aliphatic','amide','basic','hydroxyl','hydroxyl',
 'aliphatic','aromatic','aromatic')

```

Build the dataframes.

```

Build dataframe
aa_dat <- data.frame(aa,
 MW.da, vol,
 bulk, pol,
 isoelec,H2Opho.34, H2Opho.35,
 saaH20, faal.fold, polar.req,
 freq, charge, hydropathy,
 vol.cat, pol.cat, chemical)

```

## 97.3 Raw data exploration

### 97.3.1 XXXXXXXX

TODO: Examine the code below. What is it called? Replace the XXXXXXXX above with the name of this thing

```

cor_ <- round(cor(aa_dat[,-c(1,13:17)]),2)
diag(cor_) <- NA
cor_[upper.tri(cor_)] <- NA
cor_

```

	MW.da	vol	bulk	pol	isoelec	H2Opho.34	H2Opho.35	saaH20	faal.fold
## MW.da	NA	NA	NA	NA	NA	NA	NA	NA	NA
## vol	0.93	NA	NA	NA	NA	NA	NA	NA	NA
## bulk	0.55	0.73	NA	NA	NA	NA	NA	NA	NA
## pol	0.29	0.24	-0.20	NA	NA	NA	NA	NA	NA
## isoelec	0.20	0.36	0.08	0.27	NA	NA	NA	NA	NA
## H2Opho.34	-0.27	-0.08	0.44	-0.67	-0.20	NA	NA	NA	NA
## H2Opho.35	-0.25	-0.16	0.32	-0.85	-0.26	0.85	NA	NA	NA
## saaH20	0.97	0.99	0.64	0.29	0.35	-0.18	-0.23	NA	NA
## faal.fold	0.11	0.18	0.49	-0.53	-0.18	0.84	0.79	0.12	NA
## polar.req	-0.05	-0.19	-0.53	0.76	-0.11	-0.79	-0.87	-0.11	-0.81
## freq	-0.52	-0.30	-0.04	-0.01	0.02	0.26	-0.02	-0.38	-0.18

```

polar.req freq
MW.da NA NA
vol NA NA
bulk NA NA
pol NA NA
isoelec NA NA
H2Opho.34 NA NA
H2Opho.35 NA NA
saaH2O NA NA
faal.fold NA NA
polar.req NA NA
freq 0.14 NA

```

WRITING-TODO: Describe what the code below is doing and summarzie the next 2 TODOs in 2-3 sentences. TODO: Which variables have the most postive correlation? TODO: What have most negative correlation?

```
which(cor_ == max(cor_, na.rm = T), arr.ind = T)
```

```

row col
saaH2O 8 2
max(cor_, na.rm = T)

```

```
[1] 0.99
```

```
min(cor_, na.rm = T)
```

```
[1] -0.87
```

```
which(cor_ == min(cor_, na.rm = T), arr.ind = T)
```

```

row col
polar.req 10 7

```

### 97.3.2 Plot 0: Scatterplot matrix

**Code TODO:** Make a scatterplot matrix of all of the data. Included the following variables:

1. MW.da
2. vol
3. pol
4. H2Opho.34
5. polar.req

PLUS 2 others (6 total)

NOTE: Depending on how you do this you may have to make sure the first column doesn't go into the plotting command

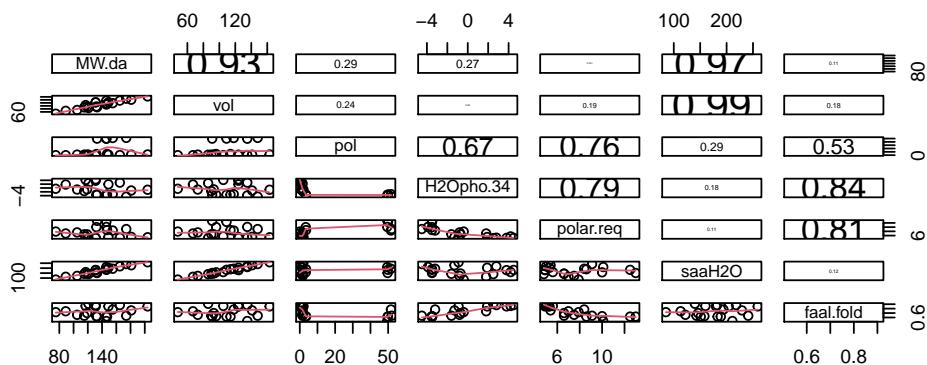
**TODO:** Write 1-2 sentences describing what this below code does in your scatter plot matrix below

```
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor,...)
{
 usr <- par("usr"); on.exit(par(usr))
 par(usr = c(0, 1, 0, 1))
 r <- abs(cor(x, y))
 txt <- format(c(r, 0.123456789), digits = digits)[1]
 txt <- paste0(prefix, txt)
 if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
 text(0.5, 0.5, txt, cex = cex.cor * r)
}
```

**TOD:** WRITE 2-3 sentences describing the relationships between the 6 variables you plotted; which have strong correlations which ones weak, which ones have non-linear relationships.

```
plot(aa_dat[,c("MW.da", "vol", "pol",
 "H2Opho.34", "polar.req",
 "saaH2O", "faal.fold")],
 panel = panel.smooth,
 upper.panel = panel.cor ,
 main = "Example")
```

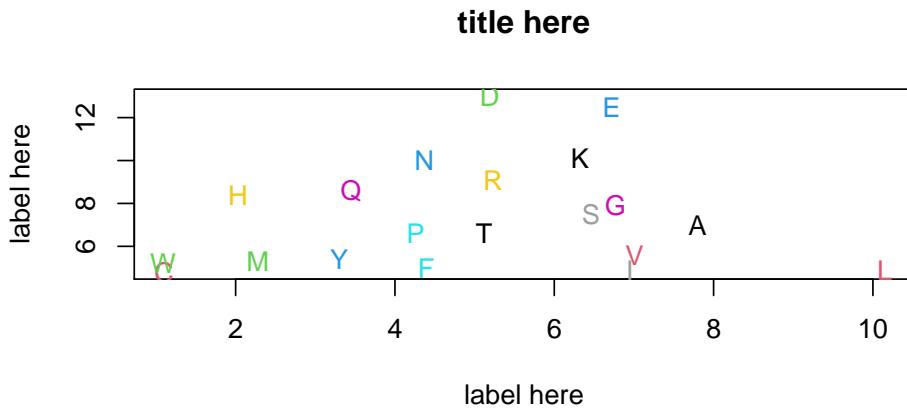
### Example



### 97.3.3 Plot 1: Replication Higgs and Attwood Figure 2.8

1. CODE-TODO: make a plot that matches Figure 2.8 on page 26 of the Higgs and Atwood chapter 2. Make sure you have plotted the correct axes
2. CODE-TODO: label the x and y axes.
3. COMMENT-TODO: indicate what is doing the main plotting and what is adding the text labels.
4. WRITING-TODO: Write 1-2 sentences describing what the plot represents

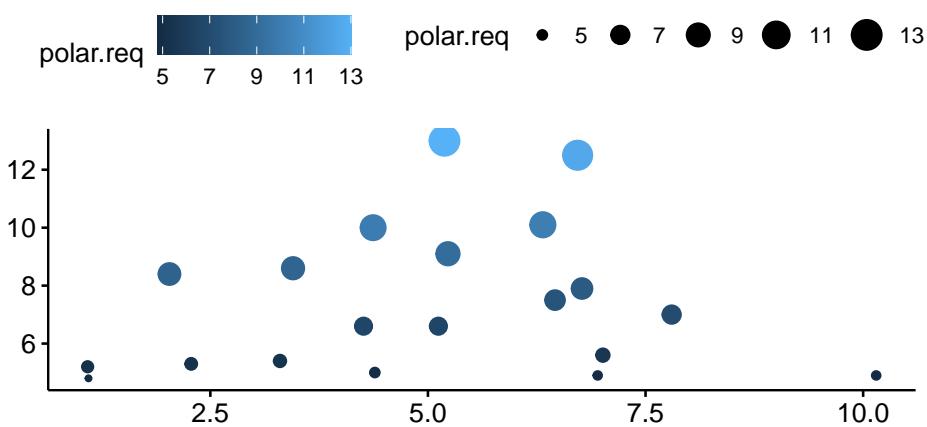
sents



#### 97.3.4 Figure 2: HA Figure 2.8 with ggpublisher

1. TODO: Make a ggpublisher version of HA Figure 2.8. Additionally, use both COLOR and SHAPE to represent 2 additional NUMERIC (NOT categorical) variables of your choice.
2. TODO: label the axes with the full name of the variables

```
ggscatter(y = "polar.req",
 x = "freq",
 size = "polar.req",
 color = "polar.req",
 data = aa_dat,
 xlab = "",
 ylab = "")
```



### 97.3.5 Figure 3: Highly correlated data

CODE:

1. TODO: Identify 2 variables that are highly positively correlated and plot them in GGPUBR
2. TODO: add a regression line
3. TODO: add a data ellipse
4. TODO: add a correlation coefficient
5. TODO: label the axes with the full name of the variables

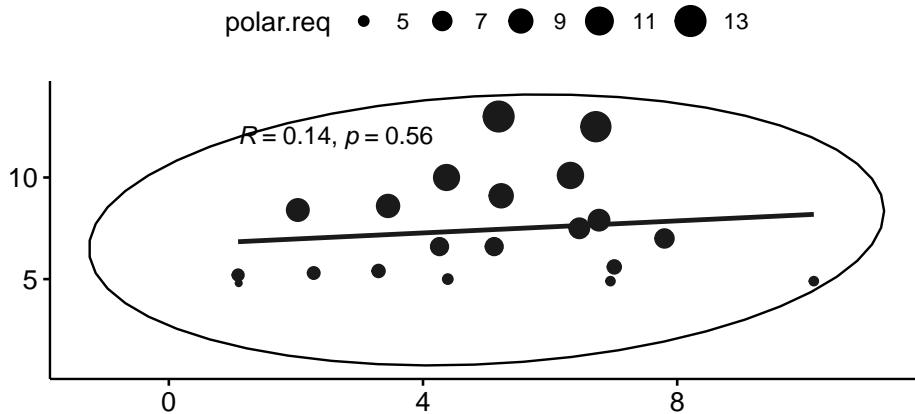
COMMENTS:

1. TODO: add a comment indicating which line of the ggpibr is adding the line
2. TODO: add a comment indicating which line of the ggpibr is adding the ellipse TODO: add a comment indicating which line of the ggpibr is adding the correlation coefficient

WRITING

- 1.TODO: describe the general mathematical form of the regression line in 1-2 sentences 1.TODO: describe the general meaning of the data ellipse in 1-2 sentences

```
ggscatter(y = "polar.req",
 x = "freq",
 size = "polar.req",
 add = "reg.line", # line of best fit
 ellipse = TRUE, # data ellipse
 cor.coef = TRUE, # correlation coef
 # color = "freq",
 data = aa_dat,
 xlab = "",
 ylab = "")
```



### 97.3.6 Figure 4: Non-linear relationship

CODE:

1. TODO: Identify 2 variables that have a non-linear relationship between them (scatter plot has a curved shape).
2. TODO: Use ggpublisher to plot a scatterplot with a **loess smoother**. Use google or the ggpubr help files to figure out how to do this
3. TODO: also vary the size and color of the data by 2 other NUMERIC variables
4. TODO: label the axes with the full name of the variables

### 97.3.7 Figure 5: Non-linear relationship on LOG scale

CODE:

1. TODO: Log-transform both of your the variables you used in Figure 4 make the plot again
2. TODO: a regression line instead of a loess smoother
3. TODO: also vary the size and color of the data by 2 other NUMERIC variables

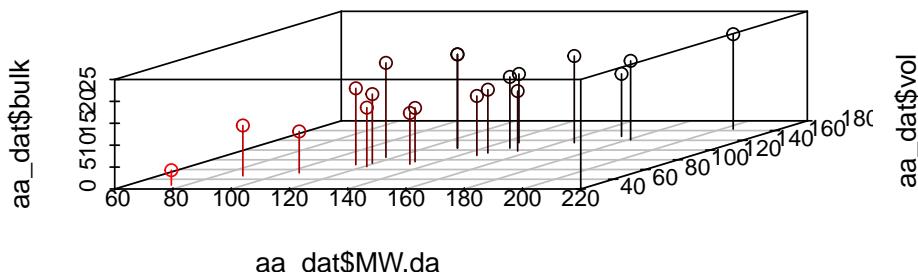
WRITING:

1. TODO: describe what the log transformation does in 1 to 2 sentences.

### 97.3.8 Figure 6: 3D Scatterplot

1. CODE TODO: Make a 3D scatterplot of 3 variables that have relatively strong correlations.
2. CODE TODO: use highlight.3d = TRUE to colored code
3. CODE TODO: Use type="h"
4. WRITING TODO: Summarize the relationships and correlations between the variables you chose in 2-4 sentences.

```
scatterplot3d(x = aa_dat$MW.da,
 y = aa_dat$vol,
 z = aa_dat$bulk,
 highlight.3d = TRUE,
 type="h")
```



## 97.4 Principal component analysis

TODO: Write 3-4 sentences describing the basic use and interpretation of PCA as we have used it in this class. Refer to Higgs and Attwood, and provide a useful quote that explains PCA. Cite the quote as (Higgs and Attwood 2005, pg XX).

TODO: Write 3-4 sentences describing the 2 basic ways we made PCA plots in this class.

### 97.4.1 Data prep

TODO: To make life easier, make a new version of your dataset which drops any categorical variables. Call it aa\_dat2 TODO: Additionally: compare Table 2.2 and the data provided. I give you ~11 numeric variables, while Table 2.2 has only 8. Read the information carefully and remove the extra variables

TODO: Write 1-2 sentences describing the approach you used to make this new dataframe.

```
aa_dat2 <- aa_dat[,-c(1,
 2,11,12,
 13,14,15,16,17)]
```

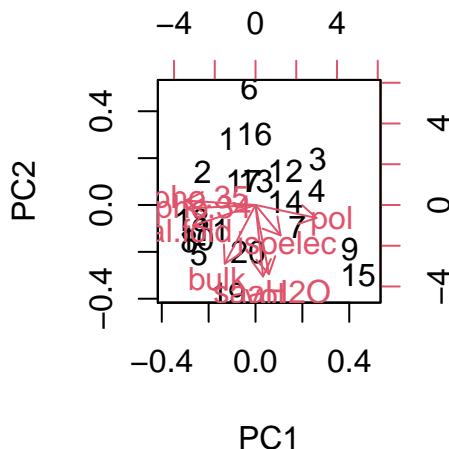
### 97.4.2 Figure 7: Principal components analysis - base R

Run a PCA and create a PCA plot using base R command.

```
pca.out <- prcomp(aa_dat2, scale = TRUE)
```

Plot the output

```
biplot(pca.out)
```



We can carry out the numeric analysis using the `rda()` function from the `vegan` package.

```
rda.out <- vegan::rda(aa_dat2,
 scale = TRUE)
```

We then extract what are known as the PCA score - don't worry about what this means right now, we'll dig into it in a moment. Basically, we're going from 15 columns of data to just 2, which we pull out of `rda.out` with the `scores()` function.

```
rda_scores <- scores(rda.out)
```

### 97.4.3 Principal Components Analysis (PCA)

We can reduce the **dimensionality** of complicated data sets in order to make them easier to visualize. In the case of our amino acid dataset we can go from 15 different variables to just to.

Principal Components Analysis (PCA) is common form of **dimension reduction**. We'll first produce the standard output of a PCA analysis – a **biplot** – before digging into the theory behind what is happening.

We can carry out the numeric analysis using the `rda()` function from the `vegan` package.

```
rda.out <- vegan::rda(aa_dat2,
 scale = TRUE)
```

We then extract what are known as the PCA score - don't worry about what this means right now, we'll dig into it in a moment. Basically, we're going from 15 columns of data to just 2, which we pull out of `rda.out` with the `scores()` function.

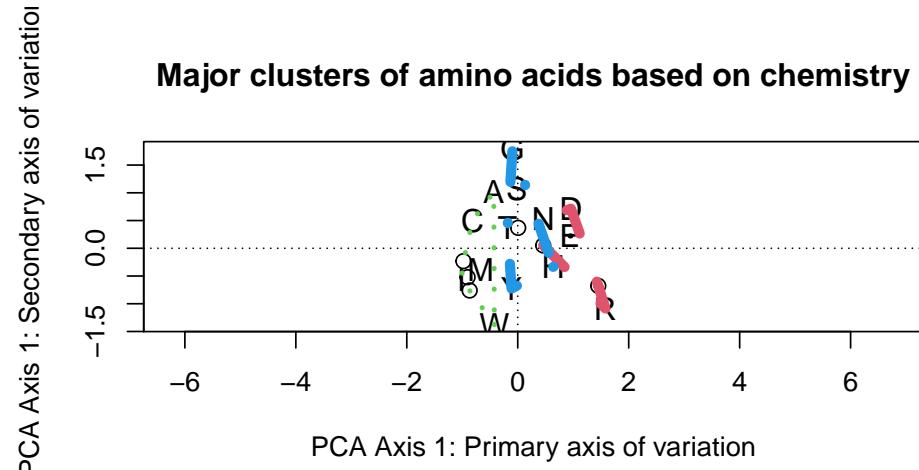
```
rda_scores <- scores(rda.out)
```

Now we'll make a **biplot**. This is similar in some ways to a scatter plot. We'll also add some additional annotation to the plot to help us see meaningful grouping of our amino acids. First, we'll label each amino acids with its 1-letter code. Then we'll delineate all amino acids within pre-specified groups to see if these groups map to the entire set of columns.

```
Build plot
biplot(rda.out, display = "sites", col = 0,
 main = "Major clusters of amino acids based on chemistry",
 xlab = "PCA Axis 1: Primary axis of variation",
 ylab = "PCA Axis 1: Secondary axis of variation")

add point
orditorp(rda.out, display = "sites", labels = aa_dat$aa, cex = 1.2)
```

```
add groupings
ordihull(rda.out,
 group = aa_dat$hydropathy,
 col = 1:7,
 lty = 1:7,
 lwd = c(3,6))
```



## 97.5 Cluster analysis

TODO: Write 2-3 sentences describing the purpose of cluster analysis. TODO: Write 3-5 sentences describing how UPGMA works.

### 97.5.1 XXXXXXXXXXXXXXXXX cluster analysis

TODO: What type of cluster analysis is UPGMA? Change the XXXXXXXX above to this. NOTE: use the aa\_dat2 data we used above for all of this.

Add row names. Don't worry about what this code is doing

```
row.names(aa_dat2) <- paste(aa_dat$aa,
 #aa_dat$hydropathy,
 sep = ".")
```

CODE-TODO: Create a distance matrix using the dist() function. Set the distance to be euclidean using method = "euclidean". Call the output dist\_euc  
WRITING-TODO: Write 2-3 sentences describing what a distance matrix is.  
WRITING-TODO: write 2-3 sentences describing what Euclidean distance is.

```
dist_euc <- dist(aa_dat2,
 method = "euclidean")
```

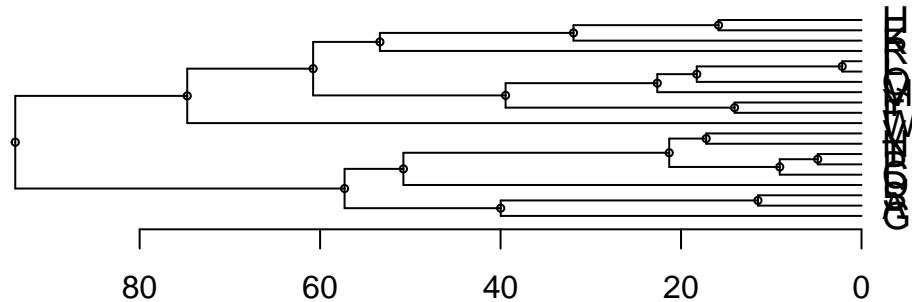
CODE-TODO: Use the `hclust()` function to do cluster analysis. Use the UPGMA algorithm (see the help file for the proper setting)

COMMENT-TODO: Add a comment that indicates what part of the code is setting it to use UPGMA.

```
clust_euc <- hclust(dist_euc,
 method = "average")
```

CODE-TODO: Plot the dendrogram (Optional - convert to a dendrogram as I did) WRITING-TODO: Write 2-5 sentences describing how your cluster diagram compares to the cluster diagram produced by Higgs and Atwood (2005) in Plate 2.2. (This is a cluster diagram that also includes a heat map)

```
dendro_euc <- as.dendrogram(clust_euc)
plot(dendro_euc, horiz = T,
 nodePar = list(pch = c(1,NA),
 cex = 0.5,
 lab.cex = 1.2))
```





# Chapter 98

## Writing functions - practice problems

```
library(compbio4all)
```

### 98.1 Introduction

Writing functions can be a hard skill to master. Here are some practice problems. A key is available.

### 98.2 Part 1: Pure practice

#### 98.2.1 Practice Function 1: Hello world!

Write a function that takes no argument, and every time you run it says “Hello world!” Call the function hello\_world()

#### 98.2.2 Practice Function 2: Hello \_\_\_\_\_ !

Write a function that takes a word as an argument, and every time you run it says “Hello \_\_\_\_\_”, where the blank is filled by the argument. Try making the function with and without a default. Call the function hello().

**Advanced (for fun):** Include in the function a test to see if what was passed to the argument is a string and return the message “no valid string was entered” if something other than a string was entered.

### 98.3 Practice Function 3: Make you own natural log function

R uses `log()` for natural and `log10()` for the base 10 log. It would be easier for non-R people to read code if there was a function with “e” in its name to indicate when the natural log is being used. Write a function that takes a number as an argument and returns the natural log. Call the function `log_e`

**Advanced (for fun):** make a function called `my_log()` that has an argument “type” to specify the type of log you want.

### 98.4 Practice Function 4: Make function that converts DNA directly to RNA

The `gsub()` function can be used to replace characters in a string. For example

```
str <- "ATCG"
gsub("T", "U", str)
```

```
[1] "AUCG"
str <- "ATTCG"
gsub("T", "U", str)
```

```
[1] "AUUCG"
str <- "ATTCGTTT"
gsub("T", "U", str)
```

```
[1] "AUUCGUUU"
str <- "AAAAA"
gsub("T", "U", str)
```

```
[1] "AAAAA"
```

Create a function called `dna_to_rna` that takes a character string as an argument and turns T to U.

# Chapter 99

## Computational Biology Portfolio: Practice Working with Strings

```
library(compbio4all)
```

### 99.1 Introduction

Gonder et al. (2007) published phylogenetic analyses of chimp population using the **XX Name of gene / sequence XX** sequence from the chimp **XX Type of DNA - nuclear? mitochondrial? both XX**. Gonder et al (2007) used both previously published and newly sequenced data to build their phylogenetic trees. Chimp sequences were compared to **XX what was the outgroup?**.

### 99.2 Purpose

**XX Write a brief summary of what this script does XX.**

### 99.3 Gonder et al 2007 data

Gonder et al report their **XX What are these serial number thingy's called? XX** in two paragraphs.

First, the previously published data they used was described as:

“We deposited all previously unpublished sequences in GenBank under the accession numbers DQ140188-DQ140269.” (Gonder et al

2006 pg 1110)

Second, the new sequences they worked up were described as:

“We obtained chimpanzee HVRI sequences from other studies from GenBank under the accession numbers L35381–L35443, U77186–U77293, AFO59042–AFO59052, and AF137481–AF137412. We obtained human HVRI sequences from the Human Mitochondrial Genome Database (mtDB) at <http://www.genpat.uu.se/mtDB/sequences.html>. We aligned sequences from GenBank and mtDB with reference to sequences produced for the study. Sequences we produced and used correspond to bases 16,048–16,391 of the human mtDNA Cambridge Reference Sequence (Andrews et al., 1999).” (Gonder et al 2006 pg 1111)

## 99.4 Gonder et al 2007 new sequences

The XX What are these serial number thingy's called? XX for their new sequences were reported as having the range:

DQ140188-DQ140269

This includes XX How many sequences XX? sequences. XX Describe what this code does in 1 to 2 sentences XX

### 99.4.1 Creating the codes

First, let me prepare some things

```
PREP

The first part of the code: letters
prefix <- "DQ"

The second part of the code:
Numeric
start
suffix.start <- 140188

end
suffix.end <- 140269
```

Now I'll re-create the actual codes using seq() and paste()

```
RECREATE CODES

seq()
Use the seq() command to generate the whole range
```

```

of numeric values
suffix <- seq(from = suffix.start, to = suffix.end, by = 1)

How many codes?
length(suffix)

[1] 82

paste()
Use paste with sep = ""
###
DQ.codes <- paste(prefix, suffix, sep = "")

check length
length(DQ.codes)

[1] 82

```

#### 99.4.2 Selecting a code for demonstration

The grep() function is a **XX What is the general name for the type of function grep() is? XX?** I can use it to select particular elements from a vector containing text or numbers. grep() is a difficult command to master and its VERY picky about its input. I'm just going to demonstrate what it can do here - you'll need a lot of practice to get good with it.

```

Day of the month of was born
as CHARACTER data
day.of.month.born <- "21"

What grep() does
grep(pattern = day.of.month.born, # pattern to search for
 x = DQ.codes) # where to search

[1] 23 24 25 26 27 28 29 30 31 32 34

storing grep() output
my.DQ.index <- grep(pattern = day.of.month.born,
 x = DQ.codes)

```

Locating a code to use

```
DQ.codes[my.DQ.index]
```

```

[1] "DQ140210" "DQ140211" "DQ140212" "DQ140213" "DQ140214" "DQ140215"
[7] "DQ140216" "DQ140217" "DQ140218" "DQ140219" "DQ140221"

```

I am going to use this code “DQ140221”

When I enter it into the NCBI website I will drop the quotation marks  
DQ140221

## 99.5 Searching Genbank

I searched the NCBI databases for this code (DQ140221) from this website

<https://www.ncbi.nlm.nih.gov/>

NCBI returned this summary

<https://www.ncbi.nlm.nih.gov/search/all/?term=DQ140221>

From the NCBI search results I search Genbank, which returned this entry.

<https://www.ncbi.nlm.nih.gov/nuccore/DQ140221.1/>

- The sequence is from **Pan troglodytes** with no subspecies specified.
- The sequence is **344 bases long**
- The authors are **Gonder et al. (unpublished)**
- NCBI indicates that there are no publications linked to this entry.
- NCBI indicates there is **1 “popset”** linked to this entry. **The PopSet is available at <https://www.ncbi.nlm.nih.gov/popset/?term=DQ140221>**

## 99.6 Gonder et al 2007 previously published sequences

The previously published sequences had the following code:

- L35381–L35443
- U77186–U77293
- AF059042–AF059052
- AF137481–AF137412

I selected the **XX what series? XX** series because **XX What did you select it?XX**. This series has **XX after you work things up use length() to determine this number XX** codes in it.

I then generate all the codes in the series and selected one that contained the day I was born.

### 99.6.1 Creating the codes

First, let me prepare some things

```
PREP
```

```
The first part of the code: letters
```

```


prefix <- "DQ" ##1) Change to your pick, L, U, or AF

The second part of the code:
Numeric
start

suffix.start <- 140188 ##2a) Change to your pick
MAKE SURE IT MATCHES THE PREFIX!

end

##2b) Change to your pick
MAKE SURE IT MATCHES THE PREFIX
suffix.end <- 140269

```

Now I'll re-create the actual codes using seq() and paste()

```

RECREATE CODES

seq()
Use the seq() command to generate the whole range
of numeric values
suffix <- seq(from = suffix.start,
 to = suffix.end,
 by = 1)

How many codes?
##3)determine this and add it to the notes above

paste()
Use paste with sep = ""
###
DQ.codes <- paste(prefix,
 suffix,
 sep = "")

check length

```

## 99.6.2 Selecting a code for demonstration

```

Day of the month of was born
as CHARACTER data
##4) Change to a number 1 to 31

```

```

day.of.month.born <- "21"

What grep() does
grep(pattern = day.of.month.born, # pattern to search for
 x = DQ.codes) # where to search

[1] 23 24 25 26 27 28 29 30 31 32 34
storing grep() output
my.DQ.index <- grep(pattern = day.of.month.born,
 x = DQ.codes)

```

Locating a code to use

```
DQ.codes[my.DQ.index]
```

```
[1] "DQ140210" "DQ140211" "DQ140212" "DQ140213" "DQ140214" "DQ140215"
[7] "DQ140216" "DQ140217" "DQ140218" "DQ140219" "DQ140221"
```

I am going to use this code ""

When I enter it into the NCBI website I will drop the quotation marks

## 99.7 Searching Genbank

I searched the NCBI databases for this code (**## YOUR CODE HERE##**) from this website

<https://www.ncbi.nlm.nih.gov/>

NCBI returned this summary

```
YOUR summary HERE##
```

From the NCBI search results I search Genbank, which returned this entry.

```
YOUR genebank entry HERE##
```

- The sequence is from **##YOUR SPECIES w/SUBSPECIES if listed##** with no subspecies specified.
- The sequence is **##SEQUENCE LENGTH##**
- The authors are **##AUTHORS HERE##**
- NCBI indicates **##NUMBER OF PUBLICATIONS IF ANY LINKED##**.
- NCBI indicates there is **##NUMBER OF POPSETS IF ANY ##** linked to this entry. **##LINK TO POPSET##**

# Chapter 100

## Writing user-friendly functions

A function to simulate a random sequence

### 100.1 Version 1

```
r_molec_seq_vs1 <- function(units,
 prob,
 length){
 # create sequence
 ## "units" = molecular subunits to sample from
 ### can be DNA (ATCG), mRNA, amino acids, etc
 ## "length" = length of sequence to generate
 ## "prob" = probability of sampling an element of "units"
 seq.n.i <- sample(x = units,
 size = length,
 replace = TRUE,
 prob = prob)

 # convert to character string
 seq.n.i <- paste(seq.n.i, sep = "", collapse = "")

 # return result
 return(seq.n.i)
}
```

Test the function. There's no defaults so it

```
r_molec_seq_vs1()

r_molec_seq_vs1(units =c("A", "T", "C", "G"),
 prob = c(0.25, 0.25, 0.25, 0.25),
 length = 10)
```

```
[1] "GTGGTTATA"
```

Test with real data, the Robinson and Robinson amino acid frequencies

```
data(robinson_aafreq)
r_molec_seq_vs1(units = robinson_aafreq$aa1,
 prob = robinson_aafreq$aa.freq,
 length = 10)
```

```
[1] "FSGDSGNLIA"
```

100.2 Version 2

Add defaults for units and prob

```
r_molec_seq_vs2 <- function(units = c("A", "T", "C", "G"),
 prob = c(0.25, 0.25, 0.25, 0.25),
 length){

 seq.n.i <- sample(x = units,
 size = length,
 replace = TRUE,
 prob = prob)

 seq.n.i <- paste(seq.n.i, sep = "", collapse = "")

 return(seq.n.i)
}
```

Now it works even if all we give it is a length

```
r molec seq vs2(length = 100)
```

## [1] "TTTAAGTCACGAGTGGATAAGAACAGCTTGGAGCACTCACTTAGATAAGGCGAGAGTGTATCGATGCCTACGTTA

## 100.3 Version 3

Now we'll give it a deafult for length

```

length = 100){

seq.n.i <- sample(x = units,
 size = length,
 replace = TRUE,
 prob = prob)

seq.n.i <- paste(seq.n.i, sep = "", collapse = "")

return(seq.n.i)

}

```

Now it works even if the parentheses are empty

```
r_molec_seq_vs3()
```

```
[1] "TGATCAGAAATCCGATTCATGATTGTTCTCATGAGGGTGTAAAGCAATTGGGAGTGGCTAAGCGAGCTTGAACCAATTGTCTATATGT
```

## 100.4 Version 4: Adding conditions and warnigns

This version tests whether the function is being run with the defaults, and throws a warning if that's true. The defaults are meant just for testing the function, and if someone runs the function with the defaults it might be that they forgot to change them.

```

r_molec_seq_vs4 <- function(units = c("A", "T", "C", "G"),
 prob = c(0.25, 0.25, 0.25, 0.25),
 length = 100){

 if(all(units == c("A", "T", "C", "G")) == TRUE &
 all(prob == c(0.25, 0.25, 0.25, 0.25)) == TRUE &
 length == 100){
 warning("Note: all parameters set to defaults. Did you want to change something?")
 }

 seq.n.i <- sample(x = units,
 size = length,
 replace = TRUE,
 prob = prob)

 seq.n.i <- paste(seq.n.i, sep = "", collapse = "")

 return(seq.n.i)

}

```

This throws a warning

```
r_molec_seq_vs4()
```

```
[1] "CCAAGAGCTAGGTACACGGGATGTGAATCGTCCCCACGTAGAACTTGCACCGAGGCCTGTTAAAGGCTTGACC
```

This doesn't

```
r_molec_seq_vs4(prob = c(0.3,0.3,0.2,0.2))
```

```
[1] "TTAAAACTAAGCACAGTGGCGCTTGTATCGGTGTCTCAAAACTACATCAGAAGCGTTGGAGGCCCTCCGCTCA
```

## 100.5 Version 5

Here I've added an additional condition to ask whether the user wants to return a string or a vector.

```
r_molec_seq_vs5 <- function(units = c("A","T","C","G"),
 prob = c(0.25,0.25,0.25,0.25),
 length = 100,
 as.string = TRUE){

 if(all(units == c("A","T","C","G")) == TRUE &
 all(prob == c(0.25,0.25,0.25,0.25)) == TRUE &
 length == 100){
 warning("Note: all parameters set to defaults. Did you want to change something?")
 }

 seq.n.i <- sample(x = units,
 size = length,
 replace = TRUE,
 prob = prob)

 if(as.string == TRUE){
 seq.n.i <- paste(seq.n.i,sep = "",collapse = "")

 }

 return(seq.n.i)
}
```

Return as a vector

```
r_molec_seq_vs5(prob = c(0.3,0.3,0.2,0.2),
 as.string = F,
 length = 10)
```

```
[1] "T" "G" "T" "A" "G" "A" "A" "G" "G" "A"
```

Return as a string

```
r_molec_seq_vs5(prob = c(0.3,0.3,0.2,0.2),
 as.string = T,
 length = 10)
```

```
[1] "CAATATATTG"
```

The default is to return a string, so as.string = T is option

```
r_molec_seq_vs5(prob = c(0.3,0.3,0.2,0.2),
 length = 10)
```

```
[1] "CGTGATGTTC"
```



# Chapter 101

## Intro to R objects

### 101.1 Commands used

- <-
- c()
- length()
- dim()
- is()
- str()
- dim()
- list()

### 101.2 R Objects

- Everything in R is an object, works with an object, tells you about an object, etc
- We'll do a simple data analysis with a t.test and then look at properties of R objects
- There are several types of objects: **vectors**, **matrices**, **lists**, **dataframes**
- R objects can hold numbers, text, or both
- A typical dataframe has columns of **numeric data** and columns of text that represent **factor variables** (aka “**categorical variables**”)

### 101.3 Differences between objects

Different objects are used and show up in different contexts.

- Most practical stats work in R is done with **dataframes**.

- A dataframe is kind of like a spreadsheet, loaded into R.
- For the sake of simplicity, we often load data in as a **vector**. This just makes things smoother when we are starting out.
- **vectors** pop up in many places, usually in a support role until you start doing more programming.
- **matrices** are occassionaly used for applied stats stuff but show up more for programming. A matrix is like a stripped-down dataframe.
- **lists** show up everywhere, but you often don't know it; many R functions make lists
- Understanding **lists** will help you efficiently work with stats output and make plots.

## 101.4 The Data

We'll use the following data to explore R objects.

Motulsky 2nd Ed, Chapter 30, page 220, Table 30.1. Maximal relaxaction of muscle strips of old and young rat bladders stimualted w/ high concentrations of nonrepinephrine (Frazier et al 2006). Response variable is %E.max

## 101.5 The assignment operator “`<-`” makes object

The code above has made two objects. We can use several commands to learn about these objects.

- `is()`: what an object is, ie, vector, matrix, list, dataframe
- `length()`:how long an object is; only works with vectors and lists, not dataframes!
- `dim()`: how long AND how wide and object is; doesn't work with vectors, only dataframes and matrices :(

### 101.5.1 `is()`

What is our “old.E.max” object?

```
is(old.E.max)

[1] "numeric" "vector" "index" "replValue"
[5] "numLike" "number" "atomicVector" "numericVector"
[9] "EnumerationValue" "replValueSp" "Mnumeric"

is(young.E.max)

[1] "numeric" "vector" "index" "replValue"
[5] "numLike" "number" "atomicVector" "numericVector"
[9] "EnumerationValue" "replValueSp" "Mnumeric"
```

Its a vector, containing numeric data.

What if we made a vector like this?

```
cat.variables <- c("old", "old", "old", "old",
 "old", "old", "old", "old", "old")
```

And used `is()`

```
is(cat.variables)

[1] "character" "vector" "data.frameRowLabels"
[4] "SuperClassMethod" "index" "atomicVector"
[7] "EnumerationValue"
```

It tells us we have a vector, containing character data. Not sure why it feels the need to tell us all the other stuff...

### 101.5.2 `length()`

Our vector has 9 elements, or is 9 elements long.

```
length(old.E.max)
```

```
[1] 9
```

Note that `dim()`, for dimension, doesn't work with vector

```
dim(old.E.max)
```

```
NULL
```

It would be nice if it said something like “1 x 9” for 1 row tall and 9 elements long. So it goes.

### 101.5.3 `str()`

`str()` stands for “structure”.

*It summarizes info about an object;* I find it most useful for looking at lists.

\* If our vector here was really really long, `str()` would only show the first part of the vector

```
str(old.E.max)
```

```
num [1:9] 20.8 2.8 50 33.3 29.4 38.9 29.4 52.6 14.3
```

### 101.5.4 `c()`

- We typically use `c()` to gather together things like numbers, as we did to make our objects above.
- note: this is lower case “c”!
- Uppercase is something else

- For me, R's font makes it hard sometimes to tell the difference between “c” and “C”
- If code isn't working, one problem might be a “C” instead of a “c”

Use `c()` to combine two objects

```
old.plus.new <- c(old.E.max, young.E.max)
```

Look at the length

```
length(old.plus.new)
```

```
[1] 17
```

Note that `str()` just shows us the first few digits, not all 17

```
str(old.plus.new)
```

```
num [1:17] 20.8 2.8 50 33.3 29.4 38.9 29.4 52.6 14.3 45.5 ...
```

## 101.6 You can save statistical output as an object

We can run a `t.test` like this, and the output goes to R's “console”

```
t.test(old.E.max, young.E.max)
```

```
##
Welch Two Sample t-test
##
data: old.E.max and young.E.max
t = -3.6242, df = 13.778, p-value = 0.002828
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-37.501081 -9.590586
sample estimates:
mean of x mean of y
30.16667 53.71250
```

We can save the output to an object using the assignment operator

```
rat.t.test <- t.test(old.E.max, young.E.max)
```

Then call up the output like this

```
rat.t.test
```

```
##
Welch Two Sample t-test
##
data: old.E.max and young.E.max
```

```
t = -3.6242, df = 13.778, p-value = 0.002828
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-37.501081 -9.590586
sample estimates:
mean of x mean of y
30.16667 53.71250
```

## 101.7 How does R save statistical output

Let's use `is()` and `str()` to see what this `rat.t.test` object really is

```
is(rat.t.test)
```

```
[1] "htest"
```

- Its an “htest” class of object. Probably means “hypothesis test”.
- Many R functions make their own special classes of objects
- For example, the `lm()` function for linear regression (aka “linear model”) makes an “lm” class of object
- Many classes of objects behave like lists, as we’ll explain below

Let's use `str()`

```
str(rat.t.test)
```

```
List of 10
$ statistic : Named num -3.62
..- attr(*, "names")= chr "t"
$ parameter : Named num 13.8
..- attr(*, "names")= chr "df"
$ p.value : num 0.00283
$ conf.int : num [1:2] -37.5 -9.59
..- attr(*, "conf.level")= num 0.95
$ estimate : Named num [1:2] 30.2 53.7
..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
$ null.value : Named num 0
..- attr(*, "names")= chr "difference in means"
$ stderr : num 6.5
$ alternative: chr "two.sided"
$ method : chr "Welch Two Sample t-test"
$ data.name : chr "old.E.max and young.E.max"
- attr(*, "class")= chr "htest"
```

- Interesting; lots of stuff.
- THe first thing it says is “List of 9”. So its a list

- Note that each one of these things corresponds to what is printed out as the output of t.test
- (A peek a deepR: The difference between what str() is showing us and the organized table we get from running t.test has to do with what is known as the “print method” and how it interacts with objects of class htest.)

## 101.8 Lists in R

- I used R for years without ever making a list
- But lists are everywhere in R
- Learning how to access lists can greatly aid in working with the output of statistical models and test
- This is especially true when plotting
- Components in lists can be accessed two different ways: using dollar signs \$ and using square brackets [ ]
- (the same is true for dataframes)
- We'll focus on using dollar signs \$

If I run just the name of the object, I get all of the output of the list

```
rat.t.test
```

```

Welch Two Sample t-test

data: old.E.max and young.E.max
t = -3.6242, df = 13.778, p-value = 0.002828
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-37.501081 -9.590586
sample estimates:
mean of x mean of y
30.16667 53.71250
```

Say I want to just see the p-value. I can use the dollar sign like this

```
rat.t.test$p.value
```

```
[1] 0.002828427
```

What about the “test statistic?”

```
rat.t.test$statistic
```

```
t
-3.624246
```

The means of each group (old vs young)

```
rat.t.test$estimate
mean of x mean of y
30.16667 53.71250
```

Try to get the confidence intervals (“confint”; answer below) ...

*#Type your attempt here:*

Confidence interval from t.test object:

```
rat.t.test$conf.int
```

```
[1] -37.501081 -9.590586
attr(,"conf.level")
[1] 0.95
```

Do you know what this confidence interval means in this context?



## Chapter 102

### An aside about t-tests

- We often say we use a t-test to “test whether the means of 2 groups are different”
- This is true, but the actual math is a bit different
- If two groups have the same mean, the **difference** between those means should be approximatley zero
- Any difference between those two groups is do to random noise / sampling variation / bad luck etc.
- Mathematically, what a t-test does is test whether the **difference between two means** is statistically difference from 0.0
- The **confidence interval** provided by `t.test()` is the the 95% CI around the difference between the 2 means.
- If this CI is contains zero, we would say that the two means are not statistically diffrence from zero
- Therefore, the two groups are not different



# Chapter 103

## Working with square brackets

- t.test doesn't directly give you the difference between the two means
- we can calculate it easily like this

```
30.16667 - 53.71250
```

```
[1] -23.54583
```

Or like this

```
mean(old.E.max) - mean(young.E.max)
```

```
[1] -23.54583
```

Challenge: can you save the output from the mean() commands to objects, then do math on those objects? Answer below...

```
#Type your attempt here:
```

Make objects holding the means

```
#old
old.mean <- mean(old.E.max)
```

```
#young
young.mean <- mean(young.E.max)
```

Subtract them

```
old.mean - young.mean
```

```
[1] -23.54583
```

Instead of calcuating the mean and their difference by hands let's get the info from the t.test output.

First, get the means. Recall that for some reason they are called the “estimate” Answer below...

*#Type your attempt here:*

```
rat.t.test$estimate
```

```
mean of x mean of y
30.16667 53.71250
```

Save this to an object

```
rat.means <- rat.t.test$estimate
```

Take a look with length(), is() and str()

*#what is this object?*  
is(rat.means)

```
[1] "numeric" "vector" "index" "replValue"
[5] "numLike" "number" "atomicVector" "numericVector"
[9] "EnumerationValue" "replValueSp" "Mnumeric"
#how long is the vector?
length(rat.means)
```

*## [1] 2*

*#take a peek at it*  
str(rat.means)

```
Named num [1:2] 30.2 53.7
- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
```

*Aside: This vector also happens to have labels attached to it, which we can access with the names() command*

```
names(rat.means)
```

```
[1] "mean of x" "mean of y"
```

## 103.1 Square brackets for real

Now let's use this object to calcualte the difference between the means.

- length() told us that the vector object has length of 2.
- Each number in the vector is an “element”
- we can select elements using square brackets

The first element

```
rat.means[1]
```

```
mean of x
30.16667
```

The 2nd element

```
rat.means[2]
```

```
mean of y
53.7125
```

Note that it gets angry if we try to access the 5th element

```
rat.means[5]
```

```
<NA>
NA
```

## 103.2 Difference between means using square brackets

Can you figure out how to calculate the difference using our rat.means objects and square brackets?

*#Type your answer here:*

```
rat.means[1] - rat.means[2]
```

```
mean of x
-23.54583
```

Let's save that to an object

```
diff.means <- rat.means[1] - rat.means[2]
```

## 103.3 Upshot: plotting output of an R statistical test

- The best way to make a plot of the results of a statistical test or model is to plot the “Effect size” and its confidence interval
- For a t-test, the effect size is the difference between the means
- This is important b/c the effect size is the most direct representation of what the test is doing
- In contrast, people often plot group means with the confidence intervals.
- THis is fine, but isn’t representing what the test is doing

- (We'll talk a lot in this class about effect sizes...)

How would we take the results of this t.test and plot it?

- **Old school:** Pen and paper (if it were the 1980s...)
- **What many people do:** Type numbers into Excel and make boxplots :(
- **Not bad:** Type numbers into new data objects and make plots in R
- **Best:** Access data from the lists and plot in R :)

## 103.4 The “not bad way”: typing directly into R

### 103.4.1 “NNot bad way” with a bar plot

We already have our difference between means

```
diff.means
```

```
mean of x
-23.54583
```

Let's make an object with the confidence interval

```
ci <- c(-37.501081, -9.590586)
```

We can load the barplot2() function from gplots() and plot the data.

\* Use library(gplots) to load the package \* you can get so called “help” for barplot2 using ?barplot2 \* we want to use the height =..., plot.ci = TRUE, ci.l=, and ci.u= “arguements” of the function

A very very ugly plot of just the mean

```
library(gplots)
barplot2(height = diff.means)
```



mean of x

Try to add the rest of the info. Hint: to get the lower ci access the 1st elemtn of our confidence itnerval object, ci, using square brackets

```
#Your attempt:
```

```
library(gplots)
pinkunicorn <- rat.t.test$conf.int

barplot2(height = diff.means,
 plot.ci = TRUE,
 ci.l=pinkunicorn[1], #ci.l = lower CI
 ci.u = pinkunicorn[2]) #ci.u = upper CI
```

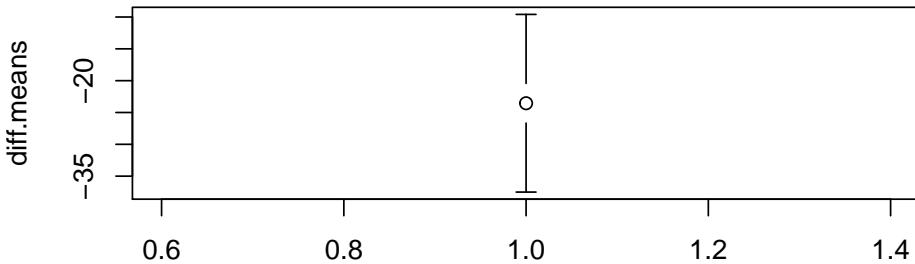


Ok, this is pretty ugly but its technically correct.

### 103.4.2 “NNot bad way” with using a single dot

- The gplots package has a function plotCI() which plots means as dots, instead of bars, and puts a CI around them
- Note that the instead of ci.l= it uses li = for the lower limit
- ... and instead of ci.u = it uses ui

```
plotCI(x = diff.means,
 li= ci[1], #use "li", not "ci.l"!
 ui = ci[2])
```

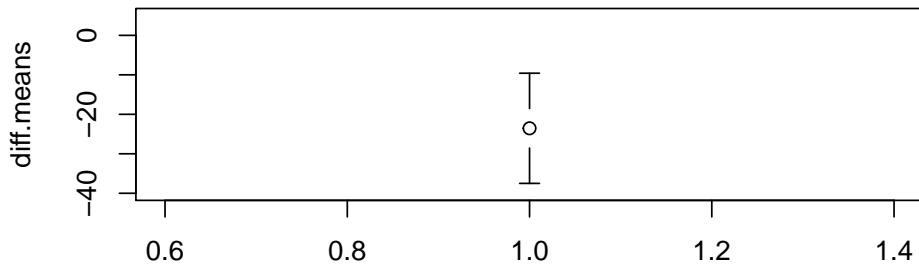


#### 103.4.2.1 Making the plot better part 1

- Since our hypothesis test is with respect to zero (no difference between means) we should include it as reference point.
- Most plotting functions can take on the argument xlim = and ylim =

- You can play with its values to it suits your tastes

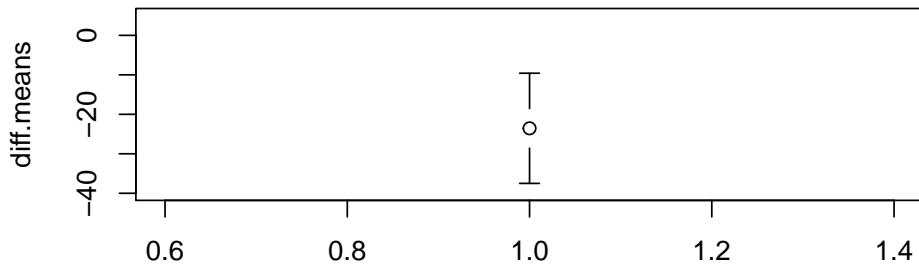
```
plotCI(x = diff.means,
 li=ci[1], #use "li", not "ci.l"!!!!
 ui = ci[2],
 ylim = c(-40,5)) #y limits
```



Note that we can use a vector here with the `ylim =` argument instead of numbers.

```
#make a vector containing the plotting limits
ylimits <- c(-40,5)

#plot
plotCI(x = diff.means,
 li=ci[1],
 ui = ci[2],
 ylim = ylimits) #use vector of values for limits
```



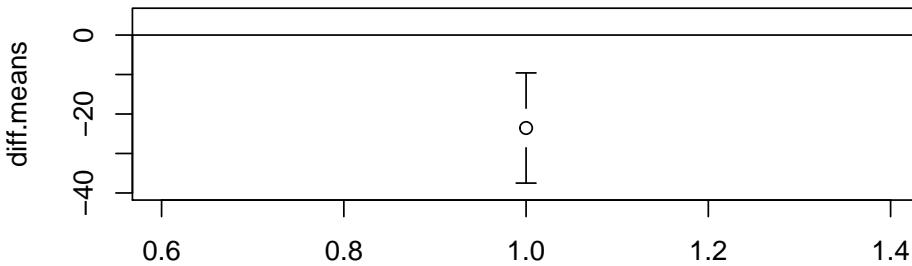
### 103.4.2.2 Making the plot better part 2

- To be really specific we could plot a line at 0.0 that shows the hypothesis being tested.
- The `abline()` function can plot lines at different angles.
- the “ab” part relates to the common geometric formula for a line,  $y = m*x + b$
- or rather  $y = a*x+b$
- for just a horizontal line we can use `abline(h = 0)`

Make a plot with line

```
#make the plot
plotCI(x = diff.means,
 li=ci[1],
 ui = ci[2],
 ylim = ylims)

#add the line
abline(h = 0)
```



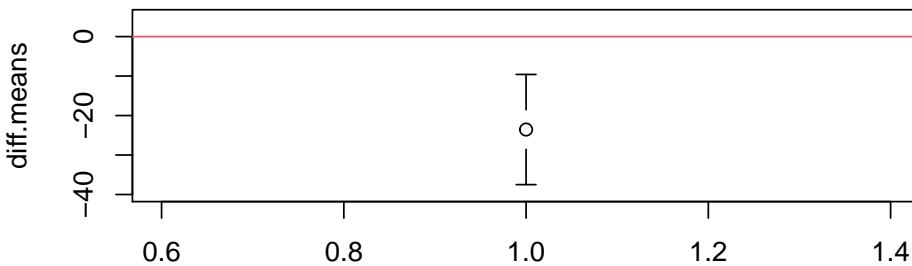
To change the colors of the lines we can use the col = argument within abline. col = 2 makes a red line Try it

```
Your attempt
```

Dot plot with red line at zero

```
#make the plot
plotCI(x = diff.means,
 li=ci[1],
 ui = ci[2],
 ylim = ylims)

#add the line
abline(h = 0, col = 2)
```



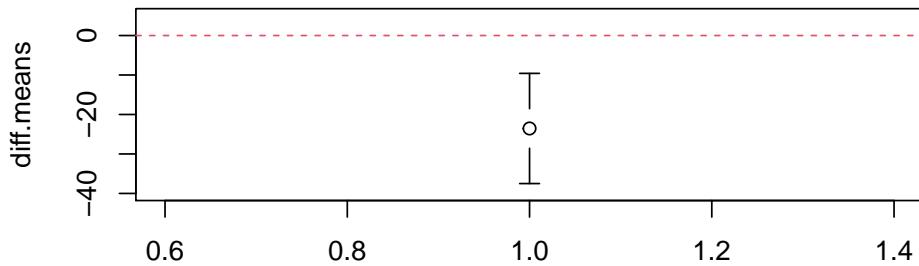
- We can change the type of line from solid to dashed using the argument “lty” for “line type”
- lty = 1 is a solid line,
- lty = 2 is a dashed line

#Try it

Answer:

```
#make the plot
plotCI(x = diff.means,
 li=ci[1],
 ui = ci[2],
 ylim = ylims)

#add the line
abline(h = 0, col = 2, lty = 2)
```



## 103.5 The best way

Recall all of this stuff is in a list from the t.test

`rat.t.test`

```
##
Welch Two Sample t-test
##
data: old.E.max and young.E.max
t = -3.6242, df = 13.778, p-value = 0.002828
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-37.501081 -9.590586
sample estimates:
mean of x mean of y
30.16667 53.71250
```

We can access components of lists using dollar signs \$

`rat.t.test$conf.int`

```
[1] -37.501081 -9.590586
attr(,"conf.level")
[1] 0.95
```

What if we want just the first confidence limit? We can combine using a dollar sign and brackets

```
rat.t.test$conf.int[1]
```

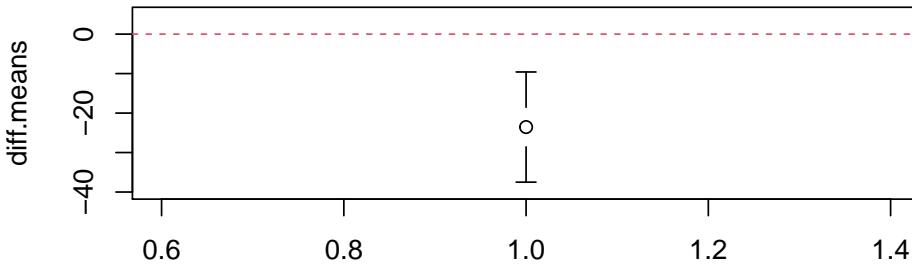
```
[1] -37.50108
```

<br.

We can therefore access everything directly from our t.test output when we plot. First, let's use our diff.means for the mean difference and access the confidence intervals directly

```
#make the plot
plotCI(x = diff.means,
 li= rat.t.test$conf.int[1],
 ui = rat.t.test$conf.int[2],
 ylim = ylims)

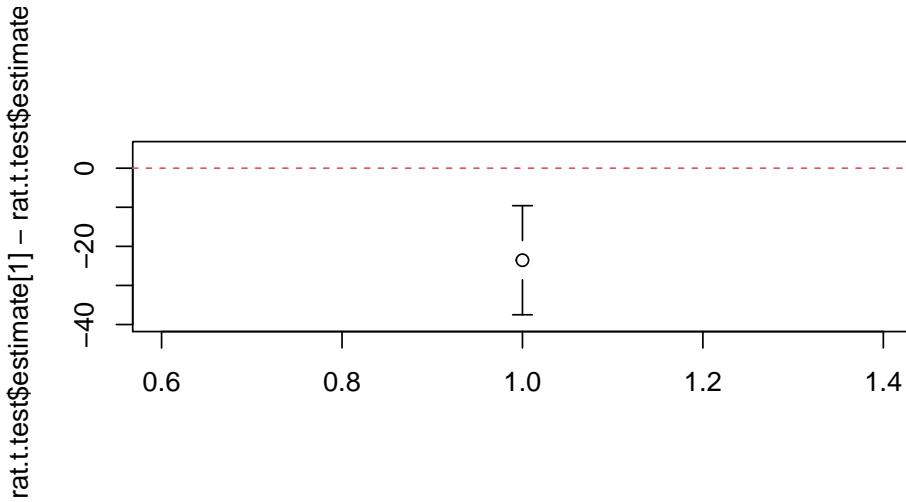
#add the line
abline(h = 0, col = 2, lty = 2)
```



This will get a big ugly, but let's even get our mean difference from the t.test object. We'll do this by using `rat.t.test$estimate[1] - rat.t.test$estimate[2]`

```
#make the plot
plotCI(x = rat.t.test$estimate[1] -
 rat.t.test$estimate[2],
 li= rat.t.test$conf.int[1],
 ui = rat.t.test$conf.int[2],
 ylim = ylims)

#add the line
abline(h = 0, col = 2, lty = 2)
```



### 103.5.1 A final fancy touch

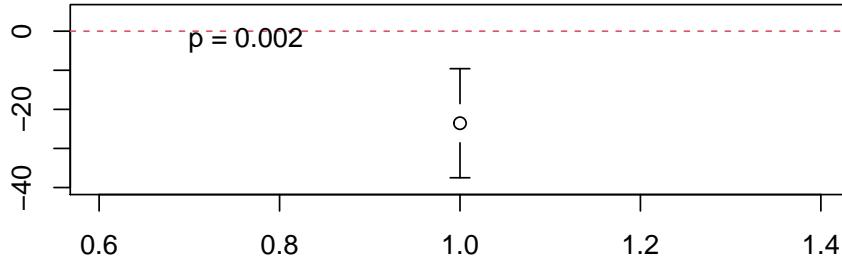
- Say we want to add the p-value for the test to this effect size plot.
- We can do this with the `text()` command like this

```
#make the plot
plotCI(x = rat.t.test$estimate[1] -
 rat.t.test$estimate[2],
 li= rat.t.test$conf.int[1],
 ui = rat.t.test$conf.int[2],
 ylim = ylimits)

#add the line
abline(h = 0, col = 2, lty = 2)

ADD THE TEXT
text(x = 0.7627, #x coordiante
 y = -2.2, #y coord
 "p = 0.002") #text: the p values
```

```
rat.t.test$estimate[1] - rat.t.test$estimate
```

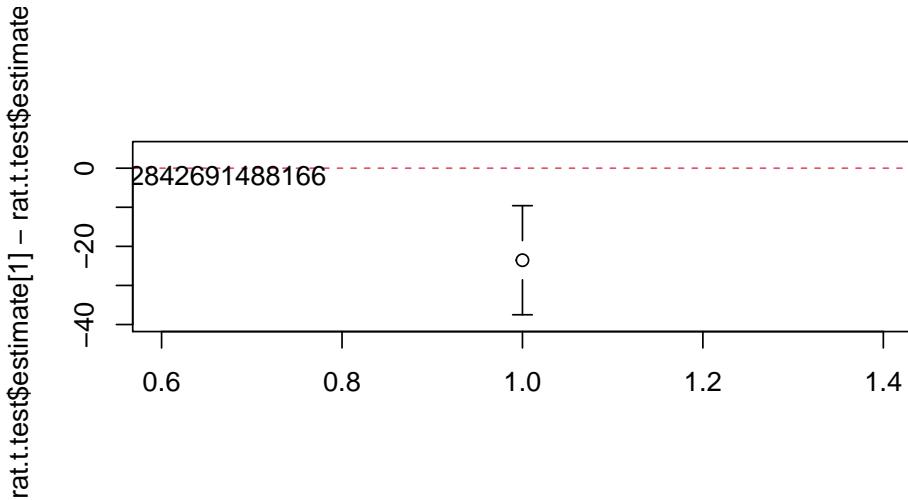


Since we know how to access stuff from R objects we can do this also be

```
#make the plot
plotCI(x = rat.t.test$estimate[1] -
 rat.t.test$estimate[2],
 li= rat.t.test$conf.int[1],
 ui = rat.t.test$conf.int[2],
 ylim = ylims)

#add the line
abline(h = 0, col = 2, lty = 2)

ADD THE TEXT
text(x = 0.627, #x coordiane
 y = -2, #y coord
 rat.t.test$p.value) #text: the p values
```



Note that its plotted off center.

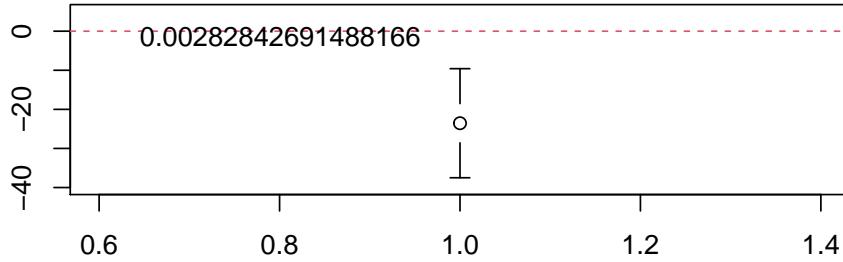
\* This is annoying deafult of `text()`. \* this is fixed using `pos = 4` to plot to the right of the coordinates you define

```
#make the plot
plotCI(x = rat.t.test$estimate[1] -
 rat.t.test$estimate[2],
 li= rat.t.test$conf.int[1],
 ui = rat.t.test$conf.int[2],
 ylim = ylims)

#add the line
abline(h = 0, col = 2, lty = 2)

ADD THE TEXT
text(x = 0.627, #x coordiante
 y = -2, #y coord
 pos = 4,
 rat.t.test$p.value) #text: the p values
```

```
rat.t.test$estimate[1] - rat.t.test$estimate
```



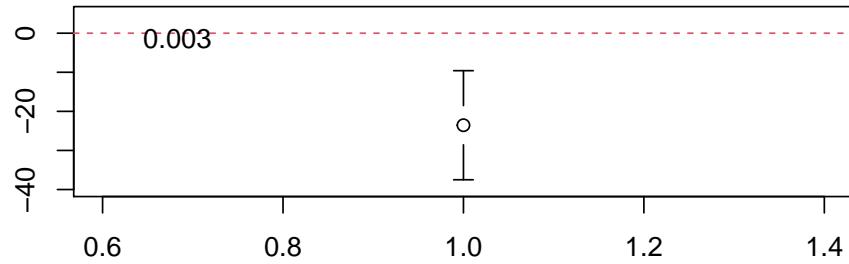
Now maybe we should round this using the `round()` command \* `round(..., 3)` will round to 3 decimal places

```
#make the plot
plotCI(x = rat.t.test$estimate[1] -
 rat.t.test$estimate[2],
 li= rat.t.test$conf.int[1],
 ui = rat.t.test$conf.int[2],
 ylim = ylims)

#add the line
abline(h = 0, col = 2, lty = 2)

ADD THE TEXT
text(x = 0.627, #x coordiante
 y = -2, #y coord
 pos = 4,
 round(rat.t.test$p.value,3)) #text: the p values
```

rat.t.test\$estimate[1] - rat.t.test\$estimate



## 103.6 Debrief

We can...

- learn about objects using `length()`, `is()`, `str()`
- access parts of list using `$` (and also brackets)
- access parts of vectors using square brackets `[ ]`
- save the output of a model / test to an object
- access part of lists for plotting instead of copying stuff

# Chapter 104

## or loops in R - copy? alt?

### 104.1 Introduction

There are several way to write for() loops in R. For this assignment, I asked you to write a loop which printed out each letter of the alphabet.

There are several ways to write a for loop in R. I will show several different ways. Out of habitat I will generally use the style of (for i in 1:length(x)), though this is not an optimal way to do things. I will discuss this way as well as some more sounds approaches.

### 104.2 Reminders

R has a built in object with has the data, called LETTERS.

```
LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

The object LETTERS is a vector containing character data.

```
is(LETTERS)
```

```
[1] "character" "vector" "data.frameRowLabels"
[4] "SuperClassMethod" "index" "atomicVector"
[7] "EnumerationValue"
```

We we access each element of a vector with its index number. To get “A”

```
LETTERS[1]
```

```
[1] "A"
```

To get "Z"

```
LETTERS [26]
```

```
[1] "Z"
```

If I want, I can access a given letter by assigning its index number to an object. Say I assign 26 to the letter i

```
i <- 26
```

I can get Z like this then

```
LETTERS[i]
```

```
[1] "Z"
```

### 104.3 For loop 1: The hard-coded for loop (aka, the don't actually do this for loop)

This is a bad way to do a for loop but makes the process totally transparent. There are 26 letters in the alphabet, so there are 26 elements in the vector LETTERS. Therefore, what we want R to do - print out a letter from the alphabet - need to be repeated 26 times. So our for loop needs to repeat the same action 26 times.

We can make an **index** for R like this

```
1:26
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

Similarly, we could do this

```
c(1:26)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

The colon in R is used to expand a sequence of numbers. Its a shortcut for the seq() command.

```
seq(from = 1, to = 26, by = 1)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

Remember that index values can be used to pull out elements from a vectors. So as noted above if I want to print the letter A, which is stored in the first element of the LETTERS vector, I can do this

#### 104.3. FOR LOOP 1: THE HARD-CODED FOR LOOP (AKA, THE DON'T ACTUALLY DO THIS FOR LOOP)52

```
LETTERS[1]
```

```
[1] "A"
```

If I want, I could store this output in a an object

```
letter.A <- LETTERS[1]
letter.A
```

```
[1] "A"
```

What a for loop can do is cycle through each index value, 1 to 26, and put it in between the bracket. It would look like this loop below;note that I added an extra step where I print out the index value.

```
for(i in 1:26){
 print(i) #print the index
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

```
[1] 1
[1] "A"
[1] 2
[1] "B"
[1] 3
[1] "C"
[1] 4
[1] "D"
[1] 5
[1] "E"
[1] 6
[1] "F"
[1] 7
[1] "G"
[1] 8
[1] "H"
[1] 9
[1] "I"
[1] 10
[1] "J"
[1] 11
[1] "K"
[1] 12
[1] "L"
[1] 13
[1] "M"
[1] 14
```

```
[1] "N"
[1] 15
[1] "O"
[1] 16
[1] "P"
[1] 17
[1] "Q"
[1] 18
[1] "R"
[1] 19
[1] "S"
[1] 20
[1] "T"
[1] 21
[1] "U"
[1] 22
[1] "V"
[1] 23
[1] "W"
[1] 24
[1] "X"
[1] 25
[1] "Y"
[1] 26
[1] "Z"
```

In this loop I used 1:26 for my index. This is **hard coding** the index values, which is never a good idea. But for the sake of illustration i makes it totally obvious what we're doing.

As I said above 1:26 is the same as c(1:26), so this loop does the same thing

```
for(i in 1:26){
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

The colon : is a shortcut for the seq() command; if I wanted to be 100% transparent to someone who didn't know R I could write this:

```
for(i in seq(from = 1, to = 26, by = 1)){
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

The step of assigning the current letter indexed by i to an object letter.i isn't actually necessary. I could simplify the code like this by **nesting** the LETTERS[i] statement within the print() statement. The letter indexed by i therefore gets

pulled up and directly **passed** to print().

```
for(i in 1:26){
 print(LETTERS[i]) #print letter i
}
```

Lines and spacing are arbitrary in R. When a for loop is short you may see people do this to save space.

```
for(i in 1:26){ print(LETTERS[i]) }
```

Or even this, dropping the curly braces.

```
for(i in 1:26) print(LETTERS[i])
```

Some people value compact code; there's no penalty in any way for using multiple lines so its generally considered best practice to split code over more lines rather than fewer.

## 104.4 For loop 3: the classic for loop

A common way to code for loops is to use a function like length() for vectors or dim() for matrices to determine how many times a loop must repeat its action. Instead of hard-coding 1:26 to get the values 1, 2, 3,... 26 I can do this:

```
1:length(LETTERS)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

This is useful in more complex situations where you could end up changing how many elements are in LETTERS. For this approach, the for loop looks like this

```
for(i in 1:length(LETTERS)){
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

In this formulation, R is making an index 1 to 26 based on the length of LETTERS. Its then assigning a value to i each time it goes through the loop. Then it pulls out the appropriate letter based on i. This is basically the same as the first example, just the values of the index, 1 to 26, is determine by R and not set by hand.

I've written hundreds of loops this way and it this way works just fine. However, in more complicated code a problem can arise if someone a 0 gets its way into a loop. See <https://jef.works/R-style-guide/#loops> for a reference to this.

The code below is optional, but if your curious it displays a behavior which may not be desirable.

```
bad.index <- NULL
length(bad.index)

[1] 0

for(i in 1:length(bad.index)){
 print(bad.index[i])
}

NULL
NULL
```

This problem is most likely to arise when a for loop is embedded in a larger program and vectors like LETTERS get generated by one part of the program and get fed later into the loop.

For this course I will probably continue to use the general format of for(i in 1:length(x)) out of habit. Hopefully by next year I'll be re-trained to use one of the techniques shown below.

## 104.5 For loop 3: The Power-user for loop

Probably the cleanest way to do a for loop in R is like this next example. A scientific survey on Twitter indicates that this how the cool kids write their loops.

In this version, R steps through each element of the vector LETTERS and on the fly assigns that element to the object i. Print then tells R to print the letter to the console.

```
for(i in LETTERS){
 print(i)
}
```

Note that if we leave out print() then we get blank screen. R is basically just reading the numbers to itself.

```
for(i in LETTERS){
 i
}
```

The use if "i" is arbitrary. We could just j instead

```
for(j in LETTERS){
 print(j)
}
```

To remember this format it might be useful to think about it like this: instead of using i or j, use something representative of what the loop is accessing from LETTERS. In the next version, I'll use letter instead of i.

```
for(letter in LETTERS){
 print(letter)
}
```

```
[1] "A"
[1] "B"
[1] "C"
[1] "D"
[1] "E"
[1] "F"
[1] "G"
[1] "H"
[1] "I"
[1] "J"
[1] "K"
[1] "L"
[1] "M"
[1] "N"
[1] "O"
[1] "P"
[1] "Q"
[1] "R"
[1] "S"
[1] "T"
[1] "U"
[1] "V"
[1] "W"
[1] "X"
[1] "Y"
[1] "Z"
```

This kind of reads like "for each letter in the vector LETTERS, print the current letter.

This general approach avoids the problem with the first two versions of the loops. Recall that I made an object called bad.index that was just NULL

```
bad.index <- NULL
```

This truly is nothing; it contains no data and has a length of zero.

```
is(bad.index)

[1] "NULL" "OptionalFunction" "optionalMethod"

class(bad.index)

[1] "NULL"
```

```
length(bad.index)
```

```
[1] 0
```

If I use this in a for loop in the current style I get no output

```
for(i in bad.index){
 print(i)
}
```

This is because bad.index has nothing to in it.

A limitation of this method is that i (or j, or letters) isn't an index, but takes on an assigned value from LETTERS. So in my old school loop I can do this, which allows me to print 2 consecutive letters. This is done by done math on the index i.

```
for(i in 1:length(LETTERS)){
 print(LETTERS[c(i,i+1)])
}
```

```
[1] "A" "B"
[1] "B" "C"
[1] "C" "D"
[1] "D" "E"
[1] "E" "F"
[1] "F" "G"
[1] "G" "H"
[1] "H" "I"
[1] "I" "J"
[1] "J" "K"
[1] "K" "L"
[1] "L" "M"
[1] "M" "N"
[1] "N" "O"
[1] "O" "P"
[1] "P" "Q"
[1] "Q" "R"
[1] "R" "S"
[1] "S" "T"
[1] "T" "U"
[1] "U" "V"
[1] "V" "W"
[1] "W" "X"
[1] "X" "Y"
[1] "Y" "Z"
[1] "Z" NA
```

With the other approach, I can't do math on i

```
for(i in LETTERS){
 print(i)
 print(i+1)
}
```

There's probably a way around this, but its not apparent to me right now.

## 104.6 For loop 4: A for loop for all purposes

For the discerning R programming who doesn't like for loop version 3 there is a function called seq\_along()

seq\_along() determines what is a valid index for a given object

```
seq_along(LETTERS)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

If you pass it a null value it can tell

```
seq_along(bad.index)
```

```
integer(0)
```

seq\_along therefore avoids the bad features of 1:length(x)

```
for(i in seq_along(bad.index)){
 print(bad.index[i])
}
```

A for loop with seq\_along(LETTERS) looks like this

```
for(i in seq_along(LETTERS)){
 print(LETTERS[i])
}
```

You can also do math on the index generated by seq\_along)<sup>9</sup>

```
for(i in seq_along(LETTERS)){
 print(LETTERS[c(i, i+1)])
}
```

## 104.7 Advanced aside: if(), else and next statements

The following is a more advanced bit of R programming information and can be skipped

Its common to put if() statements within loop to screen for certain conditions. In the code I've been using which prints out 2 consecutive letters, when I get to i = 26, the code LETTERS[c(i, i+1)] produces LETTERS[c(26, 26+1)] = LETTERS[c(26, 27)]. There is no letter 27 so an NA is thrown. I could avoid this using an if statement

```
for(i in seq_along(LETTERS)){
 if(i+1 > 26){
 print(LETTERS[c(26,26)])
 } else
 print(LETTERS[c(i, i+1)])
}

[1] "A" "B"
[1] "B" "C"
[1] "C" "D"
[1] "D" "E"
[1] "E" "F"
[1] "F" "G"
[1] "G" "H"
[1] "H" "I"
[1] "I" "J"
[1] "J" "K"
[1] "K" "L"
[1] "L" "M"
[1] "M" "N"
[1] "N" "O"
[1] "O" "P"
[1] "P" "Q"
[1] "Q" "R"
[1] "R" "S"
[1] "S" "T"
[1] "T" "U"
[1] "U" "V"
[1] "V" "W"
[1] "W" "X"
[1] "X" "Y"
[1] "Y" "Z"
[1] "Z" "Z"
```

Instead of else I could also use next.

```
for(i in seq_along(LETTERS)){
 if(i+1 > 26){
 print(LETTERS[c(26,26)])
 next
 }
}
```

```

print(LETTERS[c(i, i+1)])
}

```

```

[1] "A" "B"
[1] "B" "C"
[1] "C" "D"
[1] "D" "E"
[1] "E" "F"
[1] "F" "G"
[1] "G" "H"
[1] "H" "I"
[1] "I" "J"
[1] "J" "K"
[1] "K" "L"
[1] "L" "M"
[1] "M" "N"
[1] "N" "O"
[1] "O" "P"
[1] "P" "Q"
[1] "Q" "R"
[1] "R" "S"
[1] "S" "T"
[1] "T" "U"
[1] "U" "V"
[1] "V" "W"
[1] "W" "X"
[1] "X" "Y"
[1] "Y" "Z"
[1] "Z" "Z"

```

## 104.8 On avoiding for loops

The following section is for background. You will not be expected to write any of the code shown.

In many cases for() loops can be avoided in R. This is because R is designed to make actions on entire vectors, dataframes, and matrices easy. R also has a set of functions called apply(), and a package called purr, which allow you to avoid writing for loops directly. This relates (in part) to the concept of **vectorization** in R. For a deep dive in to some of these issues see <https://www.noamross.net/archives/2014-04-16-vectorization-in-r-why/>.

Here's an example. Let's say I had a DNA sequence stored in a vector, which each base in a separate slot in the vector.

```
a.sequence <- c("A", "T", "C", "A", "A", "A", "G", "G", "G")
```

What if for some reason I wanted these letters to be lower case. R has a handy function called tolower() which makes upper case tolower case

```
tolower("A")
```

```
[1] "a"
```

In some programming languages, to turn all of these letters to lower case you might have to do this (I've written this out with extra code to make it obvious what I'm doing)

```
for(i in 1:length(a.sequence)){
 lowercase.letter.i <- tolower(a.sequence[i]) #make lower case
 a.sequence[i] <- lowercase.letter.i #overwrite old entry
}
```

In R (and perhaps some other languages) I can do this

```
a.sequence <- c("A", "T", "C", "A", "A", "A", "G", "G", "G")
tolower(a.sequence)
```

```
[1] "a" "t" "c" "a" "a" "a" "g" "g" "g"
```

This is a somewhat trivial example. Let's say I have a matrix of DNA sequences, with each row a different sequence.

First, I'll make up some data. Don't worry about what this is doing

```
dna <- c("A", "T", "C", "G")
make_seq <- function() sample(x = dna, size = 5, replace = T)
my.matrix <- rbind(make_seq(), make_seq(), make_seq(), make_seq(), make_seq())
```

The matrix looks like this

```
my.matrix
```

```
[,1] [,2] [,3] [,4] [,5]
[1,] "A" "T" "C" "G" "T"
[2,] "A" "G" "G" "A" "T"
[3,] "A" "G" "G" "G" "C"
[4,] "C" "G" "G" "T" "C"
[5,] "C" "T" "T" "C" "A"
```

In some programming languages I'd have to write a loop to change this to lower case. In R I can use a fancy function called apply() to do it in one step

```
apply(my.matrix, 1, tolower)
```

```
[,1] [,2] [,3] [,4] [,5]
[1,] "a" "a" "a" "c" "c"
```

```
[2,] "t" "g" "g" "g" "t"
[3,] "c" "g" "g" "g" "t"
[4,] "g" "a" "g" "t" "c"
[5,] "t" "t" "c" "c" "a"
```

## 104.9 Challenge

Create a vector that contains the four codes for the DNA bases, A, T, C, G.  
Write a for() loop which prints these out.



# Chapter 105

## Function practice

### 105.1 Part 1: Pure practice

#### 105.1.1 Practice Function 1: Hello world!

Write a function that takes no argument, and every time you run it says “Hello world!” Call the function hello\_world()

```
#best
hello_world <- function(){
 return("Hello world")
}
hello_world()
```

```
[1] "Hello world"
these work too
hello_world <- function(){
 print("Hello world")
}
hello_world()
```

```
[1] "Hello world"
hello_world <- function(){
 cat("Hello world")
}
hello_world()
```

```
Hello world
```

### 105.1.2 Practice Function 2: Hello \_\_\_\_\_ !

Write a function that takes a word as an argument, and every time you run it says “Hello \_\_\_\_\_”, where the blank is filled by the argument. Try making the function with and without a default. Call the function hello().

```
hello <- function(x){
 hello.x <- paste("Hello ",x)
 return(hello.x)}
hello("mom")

[1] "Hello mom"
hello <- function(x){
 print("Hello")
 print(x)
}
hello("mom")

[1] "Hello"
[1] "mom"
```

### 105.2 Practice Function 3: Make your own natural log function

R uses log() for natural and log10() for the base 10 log. It would be easier for non-R people to read code if there was a function with “e” in its name to indicate when the natural log is being used. Write a function that takes a number as an argument and returns the natural log. Call the function log\_e

```
log_e <- function(x){
 log(x)
}

log_e(10)

[1] 2.302585
log(10)

[1] 2.302585
```

# Chapter 106

## for() loops in R

```
library(compbio4all)
```

### 106.1 Introduction

There are several way to write for() loops in R. For this assignment, I asked you to write a loop which printed out each letter of the alphabet.

There are several ways to write a for loop in R. I will show several different ways. Out of habitat I will generally use the style of (for i in 1:length(x)), though this is not an optimal way to do things. I will discuss this way as well as some more sounds approaches.

### 106.2 Reminders

R has a built in object with has the data, called LETTERS.

```
LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

The object LETTERS is a vector containing character data.

```
is(LETTERS)
```

```
[1] "character" "vector" "data.frameRowLabels"
[4] "SuperClassMethod" "index" "atomicVector"
[7] "EnumerationValue"
```

We we access each element of a vector with its index number. To get “A”

```
LETTERS[1]
```

```
[1] "A"
```

To get "Z"

```
LETTERS[26]
```

```
[1] "Z"
```

If I want, I can access a given letter by assigning its index number to an object. Say I assign 26 to the letter i

```
i <- 26
```

I can get Z like this then

```
LETTERS[i]
```

```
[1] "Z"
```

### 106.3 For loop 1: The hard-coded for loop (aka, the don't actually do this for loop)

This is a bad way to do a for loop but makes the process totally transparent. There are 26 letters in the alphabet, so there are 26 elements in the vector LETTERS. Therefore, what we want R to do - print out a letter from the alphabet - need to be repeated 26 times. So our for loop needs to repeat the same action 26 times.

We can make an **index** for R like this

```
1:26
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

Similarly, we could do this

```
c(1:26)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

The colon in R is used to expand a sequence of numbers. Its a shortcut for the seq() command.

```
seq(from = 1, to = 26, by = 1)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

### 106.3. FOR LOOP 1: THE HARD-CODED FOR LOOP (AKA, THE DON'T ACTUALLY DO THIS FOR LOOP)54

Remember that index values can be used to pull out elements from a vectors. So as noted above if I want to print the letter A, which is stored in the first element of the LETTERS vector, I can do this

```
LETTERS[1]
```

```
[1] "A"
```

If I want, I could store this output in a an object

```
letter.A <- LETTERS[1]
letter.A
```

```
[1] "A"
```

What a for loop can do is cycle through each index value, 1 to 26, and put it in between the bracket. It would look like this loop below;note that I added an extra step where I print out the index value.

```
for(i in 1:26){
 print(i) #print the index
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

```
[1] 1
[1] "A"
[1] 2
[1] "B"
[1] 3
[1] "C"
[1] 4
[1] "D"
[1] 5
[1] "E"
[1] 6
[1] "F"
[1] 7
[1] "G"
[1] 8
[1] "H"
[1] 9
[1] "I"
[1] 10
[1] "J"
[1] 11
[1] "K"
[1] 12
[1] "L"
```

```
[1] 13
[1] "M"
[1] 14
[1] "N"
[1] 15
[1] "O"
[1] 16
[1] "P"
[1] 17
[1] "Q"
[1] 18
[1] "R"
[1] 19
[1] "S"
[1] 20
[1] "T"
[1] 21
[1] "U"
[1] 22
[1] "V"
[1] 23
[1] "W"
[1] 24
[1] "X"
[1] 25
[1] "Y"
[1] 26
[1] "Z"
```

In this loop I used 1:26 for my index. This is **hard coding** the index values, which is never a good idea. But for the sake of illustration i makes it totally obvious what we're doing.

As I said above 1:26 is the same as c(1:26), so this loop does the same thing

```
for(i in 1:26){
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

The colon : is a shortcut for the seq() command; if I wanted to be 100% transparent to someone who didn't know R I could write this:

```
for(i in seq(from = 1, to = 26, by = 1)){
 letter.i <- LETTERS[i] #get leter i
 print(letter.i) #print letter i
}
```

The step of assigning the current letter indexed by `i` to an object `letter.i` isn't actually necessary. I could simplify the code like this by **nesting** the `LETTERS[i]` statement within the `print()` statement. The letter indexed by `i` therefore gets pulled up and directly **passed** to `print()`.

```
for(i in 1:26){
 print(LETTERS[i]) #print letter i
}
```

Lines and spacing are arbitrary in R. When a for loop is short you may see people do this to save space.

```
for(i in 1:26){ print(LETTERS[i]) }
```

Or even this, dropping the curly braces.

```
for(i in 1:26) print(LETTERS[i])
```

Some people value compact code; there's no penalty in any way for using multiple lines so its generally considered best practice to split code over more lines rather than fewer.

## 106.4 For loop 3: the classic for loop

A common way to code for loops is to use a function like `length()` for vectors or `dim()` for matrices to determine how many times a loop must repeat its action. Instead of hard-coding `1:26` to get the values `1, 2, 3, ..., 26` I can do this:

```
1:length(LETTERS)

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

This is useful in more complex situations where you could end up changing how many elements are in `LETTERS`. For this approach, the for loop looks like this

```
for(i in 1:length(LETTERS)){
 letter.i <- LETTERS[i] #get letter i
 print(letter.i) #print letter i
}
```

In this formulation, R is making an index 1 to 26 based on the length of `LETTERS`. Its then assigning a value to `i` each time it goes through the loop. Then it pulls out the appropriate letter based on `i`. This is basically the same as the first example, just the values of the index, 1 to 26, is determine by R and not set by hand.

I've written hundreds of loops this way and it this way works just fine. However, in more complicated code a problem can arise if someone a 0 gets its way into a loop. See <https://jef.works/R-style-guide/#loops> for a reference to this.

The code below is optional, but if your curious it displays a behavior which may not be desirable.

```
bad.index <- NULL
length(bad.index)

[1] 0
for(i in 1:length(bad.index)){
 print(bad.index[i])
}

NULL
NULL
```

This problem is most likely to arise when a for loop is embedded in a larger program and vectors like LETTERS get generated by one part of the program and get fed later into the loop.

For this course I will probably continue to use the general format of for(i in 1:length(x)) out of habit. Hopefully by next year I'll be re-trained to use one of the techniques shown below.

## 106.5 For loop 3: The Power-user for loop

Probably the cleanest way to do a for loop in R is like this next example. A scientific survey on Twitter indicates that this how the cool kids write their loops.

In this version, R steps through each element of the vector LETTERS and on the fly assigns that element to the object i. Print then tells R to print the letter to the console.

```
for(i in LETTERS){
 print(i)
}
```

Note that if we leave out print() then we get blank screen. R is basically just reading the numbers to itself.

```
for(i in LETTERS){
 i
}
```

The use if "i" is arbitrary. We could just j instead

```
for(j in LETTERS){
 print(j)
}
```

To remember this format it might be useful to think about it like this: instead of using i or j, use something representative of what the loop is accessing from LETTERS. In the next version, I'll use letter instead of i.

```
for(letter in LETTERS){
 print(letter)
}
```

```
[1] "A"
[1] "B"
[1] "C"
[1] "D"
[1] "E"
[1] "F"
[1] "G"
[1] "H"
[1] "I"
[1] "J"
[1] "K"
[1] "L"
[1] "M"
[1] "N"
[1] "O"
[1] "P"
[1] "Q"
[1] "R"
[1] "S"
[1] "T"
[1] "U"
[1] "V"
[1] "W"
[1] "X"
[1] "Y"
[1] "Z"
```

This kind of reads like "for each letter in the vector LETTERS, print the current letter.

This general approach avoids the problem with the first two versions of the loops. Recall that I made an object called bad.index that was just NULL

```
bad.index <- NULL
```

This truly is nothing; it contains no data and has a length of zero.

```
is(bad.index)
```

```
[1] "NULL" "OptionalFunction" "optionalMethod"
```

```
class(bad.index)

[1] "NULL"
length(bad.index)

[1] 0
```

If I use this in a for loop in the current style I get no output

```
for(i in bad.index){
 print(i)
}
```

This is because bad.index has nothing to in it.

A limitation of this method is that i (or j, or letters) isn't an index, but takes on an assigned value from LETTERS. So in my old school loop I can do this, which allows me to print 2 consecutive letters. This is done by done math on the index i.

```
for(i in 1:length(LETTERS)){
 print(LETTERS[c(i,i+1)])
}
```

```
[1] "A" "B"
[1] "B" "C"
[1] "C" "D"
[1] "D" "E"
[1] "E" "F"
[1] "F" "G"
[1] "G" "H"
[1] "H" "I"
[1] "I" "J"
[1] "J" "K"
[1] "K" "L"
[1] "L" "M"
[1] "M" "N"
[1] "N" "O"
[1] "O" "P"
[1] "P" "Q"
[1] "Q" "R"
[1] "R" "S"
[1] "S" "T"
[1] "T" "U"
[1] "U" "V"
[1] "V" "W"
[1] "W" "X"
[1] "X" "Y"
```

```
[1] "Y" "Z"
[1] "Z" NA
```

With the other approach, I can't do math on i

```
for(i in LETTERS){
 print(i)
 print(i+1)
}
```

There's probably a way around this, but its not apparent to me right now.

## 106.6 For loop 4: A for loop for all purposes

For the discerning R programming who doesn't like for loop version 3 there is a function called seq\_along()

seq\_along() determines what is a valid index for a given object

```
seq_along(LETTERS)
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26
```

If you pass it a null value it can tell

```
seq_along(bad.index)
```

```
integer(0)
```

seq\_along therefore avoids the bad features of 1:length(x)

```
for(i in seq_along(bad.index)){
 print(bad.index[i])
}
```

A for loop with seq\_along(LETTERS) looks like this

```
for(i in seq_along(LETTERS)){
 print(LETTERS[i])
}
```

You can also do math on the index generated by seq\_along()

```
for(i in seq_along(LETTERS)){
 print(LETTERS[c(i, i+1)])
}
```

## 106.7 Advanced aside: if(), else and next statements

The following is a more advanced bit of R programming information and can be skipped

Its common to put if() statements within loop to screen for certain conditions. In the code I've been using which prints out 2 consecutive letters, when I get to  $i = 26$ , the code `LETTERS[c(i, i+1)]` produces `LETTERS[c(26, 26+1)] = LETTERS[c(26, 27)]`. There is no letter 27 so an NA is thrown. I could avoid this using an if statement

```
for(i in seq_along(LETTERS)){
 if(i+1 > 26){
 print(LETTERS[c(26, 26)])
 } else
 print(LETTERS[c(i, i+1)])
}

[1] "A" "B"
[1] "B" "C"
[1] "C" "D"
[1] "D" "E"
[1] "E" "F"
[1] "F" "G"
[1] "G" "H"
[1] "H" "I"
[1] "I" "J"
[1] "J" "K"
[1] "K" "L"
[1] "L" "M"
[1] "M" "N"
[1] "N" "O"
[1] "O" "P"
[1] "P" "Q"
[1] "Q" "R"
[1] "R" "S"
[1] "S" "T"
[1] "T" "U"
[1] "U" "V"
[1] "V" "W"
[1] "W" "X"
[1] "X" "Y"
[1] "Y" "Z"
[1] "Z" "Z"
```

Instead of else I could also use next.

```

for(i in seq_along(LETTERS)){
 if(i+1 > 26){
 print(LETTERS[c(26,26)])
 next
 }
 print(LETTERS[c(i, i+1)])
}

[1] "A" "B"
[1] "B" "C"
[1] "C" "D"
[1] "D" "E"
[1] "E" "F"
[1] "F" "G"
[1] "G" "H"
[1] "H" "I"
[1] "I" "J"
[1] "J" "K"
[1] "K" "L"
[1] "L" "M"
[1] "M" "N"
[1] "N" "O"
[1] "O" "P"
[1] "P" "Q"
[1] "Q" "R"
[1] "R" "S"
[1] "S" "T"
[1] "T" "U"
[1] "U" "V"
[1] "V" "W"
[1] "W" "X"
[1] "X" "Y"
[1] "Y" "Z"
[1] "Z" "Z"

```

## 106.8 On avoiding for loops

The following section is for background. You will not be expected to write any of the code shown.

In many cases for() loops can be avoided in R. This is because R is designed to make actions on entire vectors, dataframes, and matrices easy. R also has a set of functions called apply(), and a package called purr, which allow you to avoid writing for loops directly. This relates (in part) to the concept of **vectorization** in R. For a deep dive in to some of these issues see <https://www.noamross.net>

/archives/2014-04-16-vectorization-in-r-why/.

Here's an example. Let's say I had a DNA sequence stored in a vector, which each base in a separate slot in the vector.

```
a.sequence <- c("A", "T", "C", "A", "A", "A", "G", "G", "G")
```

What if for some reason I wanted these letters to be lower case. R has a handy function called tolower() which makes upper case tolower case

```
tolower("A")
```

```
[1] "a"
```

In some programming languages, to turn all of these letters to lower case you might have to do this (I've written this out with extra code to make it obvious what I'm doing)

```
for(i in 1:length(a.sequence)){
 lowercase.letter.i <- tolower(a.sequence[i]) #make lower case
 a.sequence[i] <- lowercase.letter.i #overwrite old entry
}
```

In R (and perhaps some other languages) I can do this

```
a.sequence <- c("A", "T", "C", "A", "A", "A", "G", "G", "G")
tolower(a.sequence)
```

```
[1] "a" "t" "c" "a" "a" "a" "g" "g" "g"
```

This is a somewhat trivial example. Let's say I have a matrix of DNA sequences, with each row a different sequence.

First, I'll make up some data. Don't worry about what this is doing

```
dna <- c("A", "T", "C", "G")
make_seq <- function() sample(x = dna, size = 5, replace = T)
my.matrix <- rbind(make_seq(), make_seq(), make_seq(), make_seq(), make_seq())
```

The matrix looks like this

```
my.matrix
```

```
[,1] [,2] [,3] [,4] [,5]
[1,] "T" "C" "C" "C" "A"
[2,] "A" "T" "C" "G" "C"
[3,] "G" "G" "C" "G" "G"
[4,] "T" "C" "A" "A" "C"
[5,] "G" "A" "C" "G" "G"
```

In some programming languages I'd have to write a loop to change this to lower case. In R I can use a fancy function called apply() to do it in one step

```
apply(my.matrix,1,lower)
[,1] [,2] [,3] [,4] [,5]
[1,] "t" "a" "g" "t" "g"
[2,] "c" "t" "g" "c" "a"
[3,] "c" "c" "c" "a" "c"
[4,] "c" "g" "g" "a" "g"
[5,] "a" "c" "g" "c" "g"
```

## 106.9 Challenge

Create a vector that contains the four codes for the DNA bases, A, T, C, G.  
Write a for() loop which prints these out.



# Chapter 107

## Random number generation

### 107.1 Introduction

Generating random numbers is key to many tasks in statistics and computational biology. Usually we can focus on our analysis or simulation experiments and not worry about the details, but it's good to be familiar with some basic concerns when it comes to random number generations

### 107.2 Using `set.seed()` to make simulations reproducible

The way that random numbers are generated in practice uses algorithms that produce random-looking numbers and random-behaving sets up numbers. However, these algorithms are actually deterministic based on a given numeric input. Normally when we generate a random number in R, R grabs a constantly value to use as the input.

The `sys.time()` function can be used to tell us the time

```
Sys.time()
```

```
[1] "2021-07-07 00:14:30 EDT"
```

R can keep track of very small scales of time; here, I set to measure seconds to 3 decimal places (1000s of sectons)

```
options(digits.secs = 3)
Sys.time()
```

```
[1] "2021-07-07 00:14:30.714 EDT"
```

```
Sys.time()

[1] "2021-07-07 00:14:30.715 EDT"
```

Its a little more complicated than this, but basically R uses a constantly changing baseline as a starting point for random number generation.

Normally, when we generate random numbers this means that every time we call a random number generation function we get a different results.

For example, runif() generates a random value from 0 to 1. Run this 4 times and you get four values.

```
runif(n = 1)

[1] 0.84798
runif(n = 1)

[1] 0.5099076
runif(n = 1)

[1] 0.3490523
runif(n = 1)

[1] 0.2272622
```

Prior to generating a random number we can fix the starting point of the random number generator, called the **seed**. R will use the same input (seed) each time it runs its random number generator, and so the same output will be given. Every time you run this, you should get 0.5074782.

```
set.seed(10)
runif(n = 1)

[1] 0.5074782
set.seed(10)
runif(n = 1)

[1] 0.5074782
set.seed(10)
runif(n = 1)

[1] 0.5074782
set.seed(10)
runif(n = 1)

[1] 0.5074782
```

Note that the `runif()` generates random uniform data, and `rnorm()` generates random normal data, so even with the same seed they give different results. For random normal data, the random value should be 0.01874617.

```
set.seed(10)
rnorm(n = 1)

[1] 0.01874617
```

A different seed will give you a different value

```
set.seed(11)
rnorm(n = 1)

[1] -0.5910311
```

And again

```
set.seed(12)
rnorm(n = 1)

[1] -1.480568
```

The value of the seed has no inherent importance.

## 107.3 Types of random number generators

R has many random number generators, with fancy names like

- “Marsaglia-Multicarry”
- “Super-Duper”
- “Mersenne-Twister”
- “L’Ecuyer-CMRG”

You can see the default random number generator with `RNGkind()`

```
RNGkind()

[1] "Mersenne-Twister" "Inversion" "Rejection"
```

## 107.4 Reporting simulations

As long as you know the version of R someone used and the seed they used, you should be able to reproduce their results using their code. Ideally the code definitive version of a simulation reported in a paper should report the seed so that the **exact** results of the paper can be reproduced by running the code. So, in the case of my BLAST-related project, when I’m finally ready to write up a report, I should put `set.seed()` in the proper places in the code, give it a number, run the whole workflow, and report the results from that run.

## 107.5 Notes:

From the set.seed help file: “Initially, there is no seed; a new one is created from the current time and the process ID when one is required. Hence different sessions will give different simulation results, by default.”

```
Sys.time()
```

```
[1] "2021-07-07 00:14:30.800 EDT"
```

```
set.seed(Sys.time())
```

R's sample() function actually had some problems <https://arxiv.org/abs/1809.06520>

## Chapter 108

# Sample test question-Randomization

```
library(compbio4all)
```

### 108.1 Sample test question:

I have six groups in my class Neural Defects. For homework they are reading a paper with six figures. I want to randomly assign a figure to each group and make sure each figure is only used once, and that every figure is used. How can I do this in R?

1. Store what you are randomizing in a vector called “figures”
2. Write the code to do the randomization
3. Store the output in a vector called “rand.figures”
4. Use an appropriate command to check that the results the proper size
5. Use an appropriate command to check that the result is the proper data structure

### 108.2 Your answer:

Write the code needed to do this below.

```
your answer: replace the NAs with your code
```

```
object to hold figure names
figures <- NA
```

```
object to to randomization
rand.figures <- NA

check size of object

check data structure
```

### 108.3 Hint 0

- This will require something that we normally don't do when we generate random sequences of letters.

### 108.4 Hint 1:

- To do this you'll want to use the sample() command.  
For information about sample() call up the help file using ?sample

```
?sample
```

### 108.5 Hint 2 (follow up to Hint 0)

- sample() can “**sample with replacement**”, which is what we normally do for generating random sequences of letters
- OR it can **sample “without replacement”** which is actually what I want to do to make sure every figure is used once and only once

### 108.6 Hint 3:

A key argument in sample() is replace = ...

### 108.7 Answer

```
your answer: replace the NAs with your code

All of these will work the same
figures <- c(1,2,3,4,5,6)
figures <- c("1","2","3","4","5","6")
figures <- c("one","two","three","four","five","six")
```

```
rand.figures <- sample(figures, size = 6, replace = FALSE)

check size
length(rand.figures)

[1] 6

check data structure
is(rand.figures)

[1] "character" "vector" "data.frameRowLabels"
[4] "SuperClassMethod" "index" "atomicVector"
[7] "EnumerationValue"
```

Note: when you are sampling **WITH** replacement you could do this with the `runif()` command. However, this won't work when sampling without replacement. (We'll, if you really wanted to you could make it work, but it would require a loop and would be inefficient. If you want a challenge, you could try to figure this out using a `while()` loop instead of a `for()` loop)



# Chapter 109

## Simulating random amino acid sequences

```
library(compbio4all)
```

### 109.1 Selecting random letters from a vector

Make a vector with some letters

```
my.letters <- c("A", "B", "C", "D")
```

Use sample() to select one random letter

```
sample(x = my.letters, size = 1, replace = T)
```

```
[1] "B"
```

Select 10 letters, with replacement (replace = T), so that the same letter can occur more than once

```
sample(x = my.letters, size = 10, replace = T)
```

```
[1] "B" "A" "C" "B" "D" "A" "A" "C" "D" "D"
```

Select 100 letters

```
sample(x = my.letters, size = 10, replace = T)
```

```
[1] "A" "D" "A" "D" "A" "A" "C" "C" "B" "C"
```

## 109.2 Select random letters from the whole alphabet

The object LETTERS has the whole alphabet

```
LETTERS
```

```
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
```

Select 10 letters from whole alphabet

```
sample(x = my.letters, size = 10, replace = T)
```

```
[1] "A" "C" "B" "D" "D" "C" "B" "A" "D" "A"
```

## 109.3 Select random amino acids to build polypeptide

Make a vector of all letters that represent amino acid

```
all.aas <- c("A", "C", "D", "E", "F", "G", "H", "I", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "V", "W", "X", "Y", "Z")
```

Randomly select an amino acid

```
sample(x = all.aas, size = 1, replace = T)
```

```
[1] "E"
```

Make a vector of 200 amino acids

```
sample(x = all.aas, size = 200, replace = T)
```

I may want to represent the size of the fake polypeptide with an object

```
my.polypep.size <- 200
sample(x = all.aas, size = my.polypep.size, replace = T)
```

I can save this an object

```
my.polypep.size <- 200
my.pp <- sample(x = all.aas, size = my.polypep.size, replace = T)
```

I can make a single character string with not spaces using paste

```
paste(x = my.pp, sep = "", collapse = "")
```

```
[1] "WTTKLMSNGGYHFQRRCIVHWYLMYESNMIDEPHTQKVDAIGGNLDFWYGQWGEYIVALHKKEWMPEFISMTHWSWN
```

# Chapter 110

## Selecting different species and landovers from the `wildlifeR` package

```
library(compbio4all)
```

### 110.1 Introduction

Most examples associated with the BBS data from the `wildlifeR` package relate to forest birds and how their abundance changes with variation in forest cover (eg the Scarlet Tanager and the Pileated Woodpecker). In this example we'll look at urban and forest habitats.

#### 110.1.1 Installing `wildlifeR`

The current version of `wildlifeR` can be downloaded like this:

```
#load devtools
library(devtools)

#download wildlifeR from github
install_github("brouwern/wildlifeR")

#load wildlifeR into your current R sessions
library(wildlifeR)
```

### 110.1.2 Selecting a focal bird species

We'll look at the Brown Headed Cowbirds. Its alphabetic code is "BHCO".

### 110.1.3 Determining the numeric species code

In their database the USGS uses AOU numeric codes, **not** alphabetic codes. We can look up numeric codes using the dataset AOU\_species\_codes included in wildlifeR.

First, load the AOU code data

```
data(AOU_species_codes)
```

Then make sure the dplyr package is loaded

```
library(dplyr)
```

We can see the entry for BHCO and its numeric code in the "spp.num" code using the dplyr function filter()

```
AOU_species_codes %>% filter(alpha.code == "BHCO")
```

```
spp.num alpha.code name spp order
1 4950 BHCO Brown-headed Cowbird Molothrus ater 1089
```

A more detailed walk through of this process can be found in the vignette "Data Cleaning: Filtering focal rows with dplyr". This vignette can most easily be accessed at the wildlifeR website <https://brouwern.github.io/wildlifeR/>.

The filter() code above tells us that the the numeric code is 4950 We can use this number to extract just the BHCO data. We'll save this data to an object called "BBA\_PA\_BHCO". (Note that the species code is **not** the same row number of the dataframe).

```
BBS_PA_BHCO <- BBS_PA %>% filter(Aou == "4950")
```

### 110.1.4 Selecting a focal analysis question

We'll be examining this data:

- How does the abundance of a focal bird species in Pennsylvania vary with the type of habitat in the landscape.

### 110.1.5 Formatting data for habitat analyses

#### 110.1.5.1 Subsetting bird data

The habitat data used wildlifeR was collected in 2006. We will therefore use the filter() command again to isolate just that year of data. (Note that "Year" has a capital "Y" while the rest is lowercase).

```
BBS_PA_BHCO_2 <- BBS_PA_BHCO %>% filter(Year == 2006)
```

### 110.1.5.2 Preparing habitat data

The habitat data are found in the dataframe BBS\_PA\_landcover\_1km. For each BBS route the number of pixels of different habitat was determined using a GIS. These data are from the National Land Cover Dataset (NLCD) and each column is a different type of cover.

```
data(BBS_PA_landcover_1km)
```

Urban cover classes are numbers 21 through 24. These represent various “developed” landcovers with different intensities of development and impervious surfaces. See the help file using ?BBS\_PA\_landcover\_1km for more information. Forest land covers are numbers 41, 42, 43.

We'll isolate both the urban and forestlandcovers and the Route number in a new dataframe called BBS\_PA\_landcover\_1km\_2. We'll also take the “SUM” column, which is the total number of pixels in the buffer around the BBS route (this varies somewhat between routes and so landcover needs to be expressed as a proportions; see below)

```
BBS_PA_landcover_1km_2 <- BBS_PA_landcover_1km %>%
 dplyr::select(Route,
 NLCD.21, # Developed Open Space
 NLCD.22, # ... Low Intensity
 NLCD.23, # ... Medium Intensity
 NLCD.24, # ... High Intensity
 NLCD.41, # Deciduous forest
 NLCD.42, # Evergreen
 NLCD.43, # Mixed
 SUM)
```

### 110.1.5.3 Combining landcovers

**110.1.5.3.1 Combining urban landcover classes** For the sake of simplicity let's combine all four of these urban landcovers into a single “total.urban” land cover class. We'll add up the four NLCD columns put the result into a new column “total.urban.”

We'll use dplyr's mutate() command to achieve this. We “pipe” the BBS\_PA\_landcover\_1km\_2 dataframe to mutate(); within mutate we say we want a new column “total.urban”, and that this new column will be built by adding up for columns from within BBS\_PA\_landcover\_1km\_2.

```
BBS_PA_landcover_1km_2 <- BBS_PA_landcover_1km_2 %>%
 mutate(total.urban = NLCD.21+NLCD.22+NLCD.23+NLCD.24)
```

The results looks like this:

```
head(BBS_PA_landcover_1km_2)

Route NLCD.21 NLCD.22 NLCD.23 NLCD.24 NLCD.41 NLCD.42 NLCD.43 SUM
1 1 9025 4861 1085 230 44953 563 797 98591
2 2 11340 3425 142 6 25119 1130 48 82345
3 3 5638 2078 344 34 34154 521 78 86328
4 4 4757 1753 250 32 39813 1695 60 88526
5 5 5273 1238 269 15 32244 5892 3518 97708
6 6 3444 384 71 7 50529 4674 13320 87503
total.urban
1 15201
2 14913
3 8094
4 6792
5 6795
6 3906
```

**110.1.5.3.2 Combining forest land cover classes** We'll do the same thing for the forest landcovers

```
BBS_PA_landcover_1km_2 <- BBS_PA_landcover_1km_2 %>%
 dplyr::mutate(total.forest = NLCD.41+NLCD.42+NLCD.43)
```

#### 110.1.5.4 Calculating proportionages of habitat

The raw habitat data are actual counts of the number of pixels classified as a given habitat by a GIS. The number of pixels varies a little bit between routes, so to make everything comparable we need to calculate the proportionage of each forest cover. This can be done by dividing each habitat class by the SUM column

```
BBS_PA_landcover_1km_2$urban.proportion <- BBS_PA_landcover_1km_2$total.urban/BBS_PA_1
BBS_PA_landcover_1km_2$forest.proportion <- BBS_PA_landcover_1km_2$total.forest/BBS_PA_1
```

#### 110.1.5.5 Merging bird counts and habitat cover

Now we need to line up the data on the number of BHCO observed on each route in our BBA\_PA\_BHCO\_2 dataframe with the habitat data in BBS\_PA\_landcover\_1km\_2. This can be done using the full\_join() function in dplyr. This joins (combines) the two dataframes together.

```
BBS_PA_BHCO_3 <- dplyr::full_join(BBS_PA_BHCO_2 ,
 BBS_PA_landcover_1km_2,
 by = "Route")
```

The habitat data in BBS\_PA\_landcover\_1km\_2 contains data from each route

in PA, but BHCO wasn't seen on every route. To line things up dplyr has inserted NA value where data were missing from BBA\_PA\_BHCO\_2. We can get rid of these NAs like this

First, make all the "years" equal to 2006, all the Aou codes equal to 4950, and assign an alphabetic species code for each reference

```
BBS_PA_BHCO_3$Year <- 2006
BBS_PA_BHCO_3$Aou <- 4950
BBS_PA_BHCO_3$name <- "BHCO"
```

There are NA values in the "StopTotal" column, which has our counts of the number of birds. We want these to be zero. We can fix this, using the function NA\_to\_zero() from the wildlifeR package.

```
BBS_PA_BHCO_4 <- NA_to_zero(dat = BBS_PA_BHCO_3,
 column = "SpeciesTotal")
```

The data are now read for analysis. To save the data for future use use the write.csv() command.

```
write.csv(BBS_PA_BHCO_4, file = "BHCO_vs_forest_cover.csv")
```

## 110.2 Plotting Data

### 110.2.1 Exploratory analysis

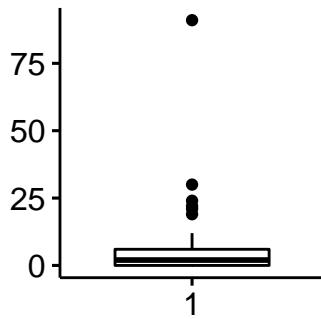
This dataset contains an extreme outlier where >75 cowbirds were seen. This is suspect to me. We can visualize this using a boxplot and a histogram

#### 110.2.1.1 Boxplot

The function ggboxplot() in the ggpibr package make boxplots.

```
library(ggpibr)

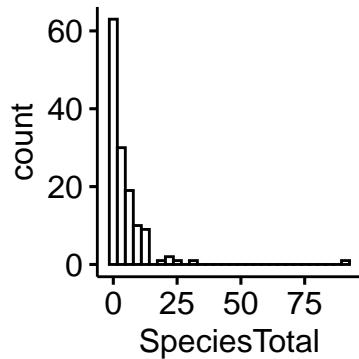
ggboxplot(data = BBS_PA_BHCO_4,
 y = "SpeciesTotal",
 ylab = "",
 xlab = "")
```



### 110.2.1.2 Histograms

`gghistogram()` makes histograms. Note that we use the argument `x = "SpeciesTotal"`.

```
gghistogram(data = BBS_PA_BHCO_4,
 x = "SpeciesTotal")
```



## 110.3 Removing an outlier3

This observation is very very odd so I am going to remove it. Normally I would do a very thorough investigation of this datapoint and try to go back to the raw data if possible to determine what is going on. For the sake of this exercise, I will make the very strong assumption that its a complete error (eg typo during data entry) and remove it. This should only be done to your own data with extreme care, thorough documentation, and total transparency.

I can use a logical argument using the “`>`” operator to identify the row that the outlier is in.

```
which(BBS_PA_BHCO_4$SpeciesTotal > 75)
```

```
[1] 76
```

I will save this row number to an object called “outlier.index” then set the value of SpeciesTotal in that row to NA

```
outlier.index <- which(BBS_PA_BHCO_4$SpeciesTotal > 75)

BBS_PA_BHCO_4$SpeciesTotal[outlier.index] <- NA
```

Again, I would normally only do this after very thorough investigation as to the origin of this extreme value.

## 110.4 Plotting polished graphs

We can visualize the relationship between the Brown-headed cowbird and forest cover using a scatterplot using the ggpublisher extension to ggplot2. Since we’re examining two variables we’ll make a two-panel plot with total urban habitat (as a proportion) on the left and total forest habitat on the right.

To set up this graph we’ll use the plotgrid() function from the cowplot library.

First, we need to make two plots and assign them to R objects. We’ll call them urban.plot and forest.plot. Note that to make these plots look nice I do the following things

- Add a regression line using add = “reg.line”
- Add a confidence interval to the line using conf.int = TRUE,
- Label the x and y axes using xlab and ylab
- Change the shape of the points in the 2nd graph using shape = 23
- Change the fill color using fill = “geen”

Note that you can only change the fill color for certain shape numbers (21 through 25).

```
urban.plot <- ggscatter(data = BBS_PA_BHCO_4,
 y = "SpeciesTotal",
 x = "urban.proportion",
 add = "reg.line",
 conf.int = TRUE,
 xlab = "Proportion urban habitat",
 ylab = "Number of cowbirds")

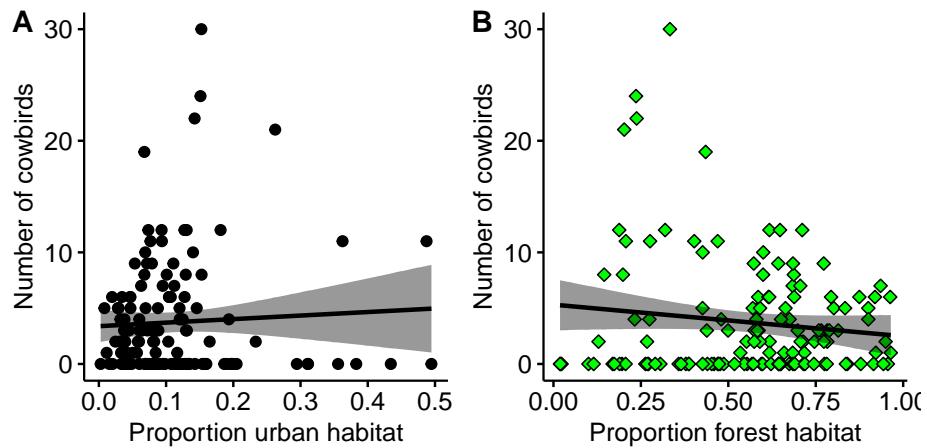
forest.plot <- ggscatter(data = BBS_PA_BHCO_4,
 y = "SpeciesTotal",
 x = "forest.proportion",
 add = "reg.line",
 conf.int = TRUE,
 xlab = "Proportion forest habitat",
 ylab = "Number of cowbirds",
 shape = 23,
```

```
fill = "green")
```

Now I'll use plot\_grid() to layout the two panels

```
library(cowplot)

plot_grid(urban.plot,
 forest.plot,
 labels = c("A", "B"))
```



## 110.5 Data modeling

### 110.5.1 Modeling habitat

Three models

```
m.null <- lm(SpeciesTotal ~ 1, BBS_PA_BHCO_4)

m.urban <- lm(SpeciesTotal ~ urban.proportion, BBS_PA_BHCO_4)

m.forest <- lm(SpeciesTotal ~ forest.proportion, BBS_PA_BHCO_4)
```

### 110.5.2 Present results

#### 110.5.2.1 Tidy results

Look at “tidy” output using the tidy() function from the broom package. This is brief but doesn't contain everything. For more info on broom see <https://cran.r-project.org/web/packages/broom/vignettes/broom.html>

We'll also set R to round things off for us a bit using the option options(digits=4)

Note that the output of `tidy()` will often be in scientific notation, where “e-x” is used to indicate what is often written as “ $10^{-x}$ ”. Eg, “5e-2” equals  $5 \times 10^{-2}$ , which equals 0.05. 5e-3 equals 0.005.

```
library(broom)

options(digits=3)

tidy(m.null)

A tibble: 1 x 5
term estimate std.error statistic p.value
<chr> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) 3.74 0.451 8.29 1.01e-13

tidy(m.urban)

A tibble: 2 x 5
term estimate std.error statistic p.value
<chr> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) 3.39 0.728 4.65 0.00000787
2 urban.proportion 3.17 5.06 0.626 0.532

tidy(m.forest)

A tibble: 2 x 5
term estimate std.error statistic p.value
<chr> <dbl> <dbl> <dbl> <dbl>
1 (Intercept) 5.32 1.16 4.57 0.0000110
2 forest.proportion -2.83 1.93 -1.47 0.145
```

### 110.5.3 Full results

Look at full output using `summary()`

#### 110.5.3.1 Full results null model

The null model has only an intercept. Its `summary()` output is not particularly interesting. The intercept here is highly significant, but this really isn’t very useful to us.

```
summary(m.null)

##
Call:
lm(formula = SpeciesTotal ~ 1, data = BBS_PA_BHCO_4)
##
Residuals:
Min 1Q Median 3Q Max
-1.00 -0.50 -0.10 0.30 1.00
```

574 CHAPTER 110. SELECTING DIFFERENT SPECIES AND LANDOVERS FROM THE WILDLIFE

```
-3.74 -3.74 -1.74 2.26 26.26
##
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.743 0.451 8.29 1e-13 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
Residual standard error: 5.26 on 135 degrees of freedom
(1 observation deleted due to missingness)
```

### 110.5.3.2 Full results urban landcover model

The urban landcover model has two parameters: intercept and the slope. Note that R doesn't provide names that are necessarily easy to understand. It also provides things printed out to lots of decimal places, which can be distracting, hence why we set options(digits=4).

```
summary(m.urban)

##
Call:
lm(formula = SpeciesTotal ~ urban.proportion, data = BBS_PA_BHCO_4)
##
Residuals:
Min 1Q Median 3Q Max
-4.95 -3.66 -1.61 2.23 26.13
##
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.385 0.728 4.65 7.9e-06 ***
urban.proportion 3.170 5.061 0.63 0.53

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
Residual standard error: 5.28 on 134 degrees of freedom
(1 observation deleted due to missingness)
Multiple R-squared: 0.00292, Adjusted R-squared: -0.00452
F-statistic: 0.392 on 1 and 134 DF, p-value: 0.532
```

### 110.5.3.3 Full results forest model

The mixed forest model has the same basic format, though the values are different.

```
summary(m.forest)

##
Call:
lm(formula = SpeciesTotal ~ forest.proportion, data = BBS_PA_BHCO_4)
##
Residuals:
Min 1Q Median 3Q Max
-5.26 -3.65 -1.57 2.36 25.63
##
Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 5.32 1.16 4.57 1.1e-05 ***
forest.proportion -2.83 1.93 -1.47 0.14
```

```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
Residual standard error: 5.24 on 134 degrees of freedom
(1 observation deleted due to missingness)
Multiple R-squared: 0.0158, Adjusted R-squared: 0.00844
F-statistic: 2.15 on 1 and 134 DF, p-value: 0.145

```

#### 110.5.4 Hypothesis testing

P-values from the summary() command are not to be used for final analyses. To compare to models and get a p-value use the anova() command.

##### 110.5.4.1 Deciduous versus null

```

anova(m.null,
 m.urban)

Analysis of Variance Table
##
Model 1: SpeciesTotal ~ 1
Model 2: SpeciesTotal ~ urban.proportion
Res.Df RSS Df Sum of Sq F Pr(>F)
1 135 3742
2 134 3731 1 10.9 0.39 0.53

```

##### 110.5.4.2 Mixed versus null

```

anova(m.null,
 m.forest)

Analysis of Variance Table
##
Model 1: SpeciesTotal ~ 1
Model 2: SpeciesTotal ~ forest.proportion
Res.Df RSS Df Sum of Sq F Pr(>F)
1 135 3742
2 134 3683 1 59.1 2.15 0.14

```

##### 110.5.4.3 Model comparison

Compare models w/AIC

```

library(bbmle)
AICtab(m.null,
 m.urban,
 m.forest)

```

```
dAIC df
m.forest 0.0 3
m.null 0.2 2
m.urban 1.8 3
```

## 110.6 Transformations

### 110.6.1 Carry out log transformation

```
BBS_PA_BHCO_4 <- BBS_PA_BHCO_4 %>% mutate(SpeciesTotal.log =
log(SpeciesTotal+1))
```

### 110.6.2 Build new models

```
m.null.log <- lm(SpeciesTotal.log ~ 1, BBS_PA_BHCO_4)

m.urban.log <- lm(SpeciesTotal.log ~ urban.proportion, BBS_PA_BHCO_4)

m.forest.log <- lm(SpeciesTotal.log ~ forest.proportion, BBS_PA_BHCO_4)
```

### 110.6.3 Compare with anova()

```
anova(m.null.log,
 m.urban.log)

Analysis of Variance Table
##
Model 1: SpeciesTotal.log ~ 1
Model 2: SpeciesTotal.log ~ urban.proportion
Res.Df RSS Df Sum of Sq F Pr(>F)
1 135 138
2 134 137 1 0.907 0.89 0.35

anova(m.null.log,
 m.forest.log)

Analysis of Variance Table
##
Model 1: SpeciesTotal.log ~ 1
Model 2: SpeciesTotal.log ~ forest.proportion
Res.Df RSS Df Sum of Sq F Pr(>F)
1 135 138
2 134 137 1 0.385 0.38 0.54
```

#### 110.6.4 Compare with AIC

```

AICtab(m.null.log,
 m.urban.log,
 m.forest.log)

dAIC df
m.null.log 0.0 2
m.urban.log 1.1 3
m.forest.log 1.6 3

```

### 110.6.5 Model curved relationships

# Chapter 111

## Data Cleaning: Filtering focal rows with dplyr

```
library(compbio4all)
```

### 111.1 Introduction

This vignette walks through the process of isolating a subset of a dataframe using the dplyr command filter(). In particular we'll use filter() to isolate rows of data that correspond to particular species of birds from a very large dataset. To do this, we'll have to learn a bit about how the row indices that identify which row data occur in within a dataframe.

### 111.2 Learning objectives

- Determine row indices using which()
- Practice using the package dplyr
  - use dplyr filter

### 111.3 Packages

- dplyr
- wildlifeR

## 111.4 R code

### 111.4.1 Load Libraries

The codes used by the American Ornithological Union (AOU) and BBS data for Pennsylvania are in the `wildlifeR` package. We'll use the `dplyr` package to "reshape" into the proper format for analysis.

```
library(wildlifeR)
library(dplyr)
library(ggplot2)
library(ggpubr)
```

### 111.4.2 Data

We'll use 2 datasets:

1. AOU codes for each species of bird in North American
2. Bird counts from the BBS data for PA, and

You can learn more about these data sets using the help command `?BBS_PA` and `?AOU_species_codes`

Load both sets of data

```
AOU species codes
data(AOU_species_codes)

BBS data for PA
data(BBS_PA)
```

## 111.5 BBS bird count data

Look at the `BBS_PA` data.

```
head(BBS_PA)
```

	countrynum	statenum	Route	RPID	Year	Aou	StopTotal	SpeciesTotal
## 1	840	72	1	101	1970	1940	2	2
## 2	840	72	1	101	1970	2730	1	1
## 3	840	72	1	101	1970	3091	2	2
## 4	840	72	1	101	1970	3131	1	1
## 5	840	72	1	101	1970	3160	21	30
## 6	840	72	1	101	1970	3250	1	2

This contains information on every bird species on every BBS route in Pennsylvania since the 1960s.

Using the `dim()` command we can see that this is a VERY big file

```
dim(BBS_PA)
[1] 244185 8
```

Almost 1/4 million rows! This would be a very very big spreadsheet. When data files get this big we often do things to save space both in terms of file size and also on the screen when viewing it. Note that even though this datafile is about birds, there are no bird names (nor names of countries or states). Everything is coded using numbers.

To identify birds in the BBS dataset the USGS uses codes designated by the Aou (American Ornithological Union). These are typically 4 number codes unique to each species.

## 111.6 AOU Species code dataframe

The dataset we loaded above with the command `data(AOU_species_codes)` loaded dataset with all of the Aou numeric codes and the corresponding common and latin names of the speices.

Using `dim()` we can see that there are >1000 different species identified in this dataset

```
dim(AOU_species_codes)
[1] 1175 5
```

Look at the data themselves

```
head(AOU_species_codes)
```

	spp.num	alpha.code	name	spp	order
## 1	10	WEGR	Western Grebe	Aechmophorus occidentalis	96
## 2	11	CLGR	Clark's Grebe	Aechmophorus clarkii	98
## 3	12	WCGR	Western/Clark's Grebe	hybrid	97
## 4	20	RNGR	Red-necked Grebe	Podiceps grisegena	94
## 5	30	HOGR	Horned Grebe	Podiceps auritus	93
## 6	40	EAGR	Eared Grebe	Podiceps nigricollis	95

This dataframe tells us the following thigns

- spp.num: a numeric code for each species
- alapha.code: a 4-letter alphabetic code for each species
- name: the common name for each species
- spp: scientific name
- order: taxonomic order

We want to use the alphabetic code and/or the common name to indentify the numeric species code. We can then use this numeric code to isoalted data from the BBS dataset.

### 111.6.1 Determine the code for the European starling

If we know the 4-letter alphabetic code we can use the which() command to locate the numeric code.

For example, the code for the European starling is “EUST”. The following code determine which ROW NUMBER the European startling occur in within the AOU\_species\_codes dataframe (and saves it to an object “EUST.row.number”)

```
EUST.row.number <- which(AOU_species_codes$alpha.code == "EUST")
```

If you google “AOU species codes” you can usually locate a PDF or website with the codes.

What we have retrieved with which() is a number that is the ROW that has “EUST” in it is

```
EUST.row.number
```

```
[1] 643
```

The number happens to be ‘r EUST.row.number’

We can look at what is in row ‘r EUST.row.number’ like this, but giving R the dataframe and using square brackets [ ]. Note the comma after “EUST.row.number”.

```
AOU_species_codes[EUST.row.number,]
```

```
spp.num alpha.code name spp order
717 4930 EUST European Starling Sturnus vulgaris 864
```

Or like this. Again , note the comma after “643”.

```
AOU_species_codes[643,]
```

```
spp.num alpha.code name spp order
717 4930 EUST European Starling Sturnus vulgaris 864
```

This tells us that the in row 643 there is data on the European starling, *Sturnus vulgaris*. In particular, it has a species number of 4930. This number is what we need to extract European starling data from the BBS\_PA dataframe.

```
EUST.numeric.code <- 4940
```

### 111.7 dplyr and filter()

which() is the classic way of doing this in R. You can do the exact same thing with the dplyr function filter(). Note the use of the “pipe” “%>%” which connects the dataset AOU\_species\_codes with the filter command.

```
AOU_species_codes %>% filter(alpha.code == "EUST")
spp.num alpha.code name spp order
1 4930 EUST European Starling Sturnus vulgaris 864
```

More on dplyr and filter() below.

### 111.7.1 OPTIONAL/ADVANCED: Advanced searching using grep

The above approach requires that you know the alphabetic species code, which we fed to the which(). R has a flexible search tool called grep() that allows you to search for text, parts of words, and parts of sentences. grep() allows us to search using words like this within the “name” column.

We can get the same result as above like this

```
grep("European starling",
 AOU_species_codes$name,
 ignore.case = TRUE)
```

```
[1] 643
```

If we use just the word starling, though, we can more than one number. Each of these is a row number where the word “starling” occurs.

```
grep("starling",
 AOU_species_codes$name,
 ignore.case = TRUE)
```

```
[1] 643 1129 1130 1138
```

We can save these numbers to an R object then see what they pull up in the AOU names dataframe

Store the row numbers

```
i.starling <- grep("starling",
 AOU_species_codes$name,
 ignore.case = TRUE)
```

See what they correspond to

```
AOU_species_codes[i.starling,]
```

	spp.num	alpha.code	name	spp	order
## 717	4930	EUST	European Starling	Sturnus vulgaris	864
## 1235	8640	SAST	Samoan Starling	Aplonis atrifusca	863
## 1236	8650	POST	Polynesian Starling	Aplonis tabuensis	862
## 1244	8720	MIST	Micronesian Starling	Aplonis opaca	861

There are 4 species of starling in the AOU\_species\_codes dataframe.

We could also use “European” with grep. There are 4 birds with this word in their common name

```
#store the row numbers
i.European <- grep("European",
 AOU_species_codes$name,
 ignore.case = TRUE)

#see what they match
AOU_species_codes[i.European,]

spp.num alpha.code name spp order
195 1380 EGWT European Green-winged Teal Anas crecca crecca 57
373 2710 EUGP European Golden-Plover Pluvialis apricaria 289
717 4930 EUST European Starling Sturnus vulgaris 864
781 5261 EUGO European Goldfinch Carduelis carduelis 1154
```

grep() is very flexible and can take parts of words. Let's try just “Euo”

```
#store the row numbers
i.Euro <- grep("Euro",
 AOU_species_codes$name,
 ignore.case = TRUE)

#see what they match
AOU_species_codes[i.Euro,]

spp.num alpha.code name spp order
195 1380 EGWT European Green-winged Teal Anas crecca crecca 57
373 2710 EUGP European Golden-Plover Pluvialis apricaria 289
717 4930 EUST European Starling Sturnus vulgaris 864
781 5261 EUGO European Goldfinch Carduelis carduelis 1154
```

It would work also with “pean”, “star”, or any other set of letters.

## 111.8 Isolate rows from a large dataframe using dplyr’s filter function

Above we used the which() and dplyr::filter() functions to figure out the AOU code for our focal bird. Now we'll use that information to create a more manageable subset of BBS data.

First, load dplyr if you haven't. Then use the big BBS\_PA dataset in conjunction with a pipe and filter() to subset just the European starling data. Note that use the “<-” operation to send all the output to a new dataframe

```
#load dplyr
#library(dplyr)

#filter just the Euro starling
BBS_PA_EUST <- BBS_PA %>% filter(Aou == 4940)
```

Here I gave filter the Aou code 4940. I could also have given it the object I made above that stored the number

```
#object with the code
EUST.numeric.code
```

```
[1] 4940
```

Use the object instead of the number 4940

```
BBS_PA_EUST <- BBS_PA %>% filter(Aou == EUST.numeric.code)
```

The BBS\_PA\_EUST contains just rows from BBS\_PA that contain the numeric code 4940 in the Aou column.

Let's look at what we've done. Call dim() on the original BBS\_PA data and the subset BBS\_PA\_EUST and see how the See how the dataframe has change.

```
dim(BBS_PA)
```

```
[1] 244185 8
```

```
dim(BBS_PA_EUST)
```

```
[1] 1615 8
```

We've gone from a 1/4 million row dataframe BBS\_PA to a 1600 row dataframe BBS\_PA\_EUST.

```
summary(BBS_PA_EUST)
```

	countrynum	statenum	Route	RPID	Year
## Min.	:840	Min. :72	Min. : 1	Min. :101	Min. :1966
## 1st Qu.	:840	1st Qu.:72	1st Qu.: 16	1st Qu.:101	1st Qu.:1984
## Median	:840	Median :72	Median : 35	Median :101	Median :1997
## Mean	:840	Mean :72	Mean : 83	Mean :101	Mean :1995
## 3rd Qu.	:840	3rd Qu.:72	3rd Qu.: 62	3rd Qu.:101	3rd Qu.:2006
## Max.	:840	Max. :72	Max. :907	Max. :101	Max. :2016
## Aou		StopTotal	SpeciesTotal		
## Min.	:4940	Min. : 1.00	Min. : 1.0		
## 1st Qu.	:4940	1st Qu.: 1.00	1st Qu.: 2.0		
## Median	:4940	Median : 3.00	Median : 6.0		
## Mean	:4940	Mean : 4.21	Mean :10.7		
## 3rd Qu.	:4940	3rd Qu.: 6.00	3rd Qu.:14.0		
## Max.	:4940	Max. :20.00	Max. :78.0		

### 111.8.1 What next?

This vignette shows how to take a really big dataframe and isolate just the subset of data that is required. In the `wildlifeR` two analyses with these data are outlined: analysis of change over time, and analysis of abundance versus landcover.

1. If you want to examine change in a species over time, see the vignette ... which uses the `wildlifeR` function `sample_BBS_routes()` to create a simple dataframe for looking at change over time.
2. If you want to examine change in abundance versus a landcover variable see “Data Cleaning: Merging 2 large data sets with dplyr”, which is an extended vignette which reviews what we’ve done here and extends it to access and merge-in landcover data.

## 111.9 References

### 111.9.1 Books

A good introduction to using `dplyr` is:

Beckerman et al’s book “Getting start with R: An introduction for biologists” 2nd ed.

### 111.9.2 Websites

“Selecting columns and renaming are so easy with `dplyr`” <https://blog.exploratory.io/selecting-columns-809bdd1ef615>

## 111.10 References

### 111.10.1 Books

A good introduction to using `dplyr` is:

Beckerman et al’s book “Getting start with R: An introduction for biologists” 2nd ed.

### 111.10.2 Websites

“Selecting columns and renaming are so easy with `dplyr`” <https://blog.exploratory.io/selecting-columns-809bdd1ef615>

# Chapter 112

## Data cleaning: Merging 2 large data sets with dplyr

```
library(compbio4all)
```

### 112.1 Introduction

This vignette walks through the process of taking 2 complimentary sets of data and “merging” them using the powerful function `full_join()` in the package `dplyr`. We’ll take data on 1) the number of birds observed along USGS Breeding Bird Survey (BBS) routes and merge that ecological data with 2) geographic information on the types of landcover along those routes. In addition to this primary tasks of merging datasets, we’ll also do several other things to clean up the data to make it easy to use.

### 112.2 References

#### 112.2.1 Books

A good introduction to using `dplyr` is:

Beckerman et al’s book “Getting start with R: An introduction for biologists”  
2nd ed.

#### 112.2.2 Websites

“Selecting columns and renaming are so easy with `dplyr`” <https://blog.exploratory.io/selecting-columns-809bdd1ef615>

### 112.3 Learning objectives

- Practice using the package dplyr
  - use dplyr pipes: %>%
- General dataframe cleaning with dplyr
  - Select focal columns using select()
  - Select focal rows using filter()
  - Change columns names with rename()
- Merge 2 dataframes using full\_join()
- Plotting data using the ggpublisher extension of ggplot2

### 112.4 Packages

- dplyr
- ggplot2
- ggpublisher
- wildlifeR

### 112.5 R code

#### 112.5.1 Load Libraries

BBS and landcover data for Pennsylvania are in the wildlifeR package, and we'll use the dplyr package to "reshape" into the proper format for analysis.

```
library(dplyr)
library(wildlifeR)
```

#### 112.5.2 Data

We'll use 2 datasets:

1. Bird counts from the BBS data for PA, and
2. USGS landcover classifications (e.g. urban, deciduous forest, etc) for a "buffer" 1 km around each BBS route in PA.

You can learn more about these data sets using the help command ?BBS\_PA and "BBS\_PA\_landcover\_1km".

Load both sets of data

```
BBS data for PA
data(BBS_PA)

Landcover data for BBS routes in PA
data(BBS_PA_landcover_1km)
```

### 112.5.3 Look at the BBS\_PA dataframes

#### 112.5.3.1 Definitions of

```
head(BBS_PA)
```

```
countrynum statenum Route RPID Year Aou StopTotal SpeciesTotal
1 840 72 1 101 1970 1940 2 2
2 840 72 1 101 1970 2730 1 1
3 840 72 1 101 1970 3091 2 2
4 840 72 1 101 1970 3131 1 1
5 840 72 1 101 1970 3160 21 30
6 840 72 1 101 1970 3250 1 2
```

For our work, the important columns are

- Route: a categorical variables that defines the unique ID of each route in the State of PA
- Year: a numeric variables that defines the year the data were collected. The minimum is 1966. This is a potential focal predictor variable for examining change over time.
- Aou: a code that designates each species, as defined by the American Ornithological Union (AOU). (Note that this A is uppercase while the rest is lowercase)
- SpeciesTotal: the total number of individuals of a given species that were observed on a given route. This will be our focal response variable.

(NOTE: We will **not** be using the “StopTotal” column. A BBS routes is composed of 50 point counts, called “stops.” StopTotal is the total number of stops out of 50 on which a given species in a given year was observed. Its max value is 50, or 100% of the 50 stops on the route)

see ?BBS\_PA for more information on these variables

#### 112.5.3.2 Size of BBS\_PA dataframe

The BBS data is really really big dataset!:

```
dim(BBS_PA)
```

```
[1] 244185 8
```

```
summary(BBS_PA)
```

```
countrynum statenum Route RPID Year
Min. :840 Min. :72 Min. : 1 Min. :101 Min. :1966
1st Qu.:840 1st Qu.:72 1st Qu.: 31 1st Qu.:101 1st Qu.:1984
Median :840 Median :72 Median : 58 Median :101 Median :1997
Mean :840 Mean :72 Mean :108 Mean :101 Mean :1995
3rd Qu.:840 3rd Qu.:72 3rd Qu.: 90 3rd Qu.:101 3rd Qu.:2007
```

```
Max. :840 Max. :72 Max. :911 Max. :203 Max. :2016
Aou StopTotal SpeciesTotal
Min. : 60 Min. : 1.0 Min. : 1
1st Qu.: 4610 1st Qu.: 2.0 1st Qu.: 2
Median : 5630 Median : 4.0 Median : 5
Mean : 5493 Mean : 7.2 Mean : 14
3rd Qu.: 6670 3rd Qu.:10.0 3rd Qu.: 15
Max. :22860 Max. :50.0 Max. :992
```

There are so many rows b/c

- There are 50 years of data.
- There are 202 species of birds observed during the BBS in Pennsylvania
- There are 137 routes in Pennsylvania

Note, however, that if you multiply `yearsspeciesroutes` you get an even **bigger** number.

**50\*202\*137**

```
[1] 1383700
```

This is because

- The number of routes has changed over time
- Data are only given for a species if it was observed on a route

Any important consequence of this is that there are only data on a species when it is actually observed. So if a bird is seen on route 99 in 2015 but not 2016, there is a row of data for it in 2015 but not 2016. Therefore, there are **no zeros** in the `StopTotal` column, which gives the number of each species seen on a route in a given year.

We can see using `summary()` that the minimum value in the `StopTotal` column is indeed 1; not a single zero.

`summary(BBS_PA$StopTotal)`

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.0 2.0 4.0 7.2 10.0 50.0
```

## 112.6 Isolate focal columns

- For our analyses we don't need every column in the dataset
- Let's look at the Year, Aou species code, Route number, and SpeciesTotal columns
  - All the data are from the state of Pennsylvania in the USA, so we don't need these columns.
- We can use the `select()` function from the `dplyr` package to isolate the columns we want.

- We tell dplyr the dataframe we want to work on, BBS\_PA, add a “pipe” %>%, and then use the select() command to tell it the columns within the dataframe we want.
- We’ll save them into a new object called BBS\_PA2
- Note that the “A” in “Aou” is uppercase while the rest are lowercase)
- See page 60-61 in Beckerman et al’s book “Getting start with R: An introduction for biologists” for more info on select()

```
#load dplyr if you haven't already
#library(dplyr)

#look at the names of the full dataframe
names(BBS_PA)

[1] "countrynum" "statenum" "Route" "RPID" "Year"
[6] "Aou" "StopTotal" "SpeciesTotal"

#use select() to isolate focal columns
and put into a new dataframe
##(Note that this A in Aou is uppercase
while the rest of the letters are lowercase)

#BBS_PA2 <- select(.data = BBS_PA,
Year, Aou, Route, SpeciesTotal)

#dplyr was not working for me so I did this another way of doing
#this

BBS_PA2 <- BBS_PA[,c("Year", "Aou", "Route", "SpeciesTotal")]

#look at columns in new dataframe
names(BBS_PA2)

[1] "Year" "Aou" "Route" "SpeciesTotal"
```

### 112.6.1 Isolate focal species using filter()

- Each row is data on a different combination of: a species, a year, and a route
- We want to isolate rows that just relate to our focal species; we’ll use the Scarlet Tanager
- The AOU (American ornithological Union) code for the Scarlet Tanager (SCTA) is 6080.
- We can tell R to just give us data on SCTA using the filter() command in the dplyr package.
- See page 62-65 in Beckerman et al’s book “Getting start with R: An in-

introduction for biologists” for more info on filter()

### 112.6.1.1 An example BBS dataframe

- The following dataframe is a mock up of the general structure of the BBS data.
- In the “Aou” column are listed several numbers: 5980, 6080, 5950, 6110, and several others
  - Each one of these numbers represents a different species
  - We want to isolate just the codes 6080, which represent the scarlet tanager
  - We want to discard the rest

```
Year Aou Route SpeciesTotal
2910 2011 5980 2 10
2974 2012 5980 2 9
3047 2013 5980 2 6
2911 2011 6080 2 3 we want this row
2909 2011 5950 2 4
2973 2012 5950 2 1
3046 2013 5950 2 5
2975 2012 6080 2 3 we want this row
2912 2011 6110 2 1
2976 2012 6110 2 3
3049 2013 6130 2 23
3048 2013 6080 2 3 we want this row
2913 2011 6120 2 4
2977 2012 6120 2 2
3050 2013 6140 2 7
```

That is, we want this

```
Year Aou Route SpeciesTotal
1 2011 6080 2 3 we want this row
2 2012 6080 2 3 we want this row
3 2013 6080 2 3 we want this row
```

### 112.6.2 Select just focal rows with dplyr::filter()

#### 112.6.2.1 Focal bird rows: 6080, the scarlet tanager

- Again, the “%>%” is called a pipe that connects the dataframe BBS\_PA2 with the filter() command
- “==” in the filter() commands means “exactly equals”

```
library(dplyr)

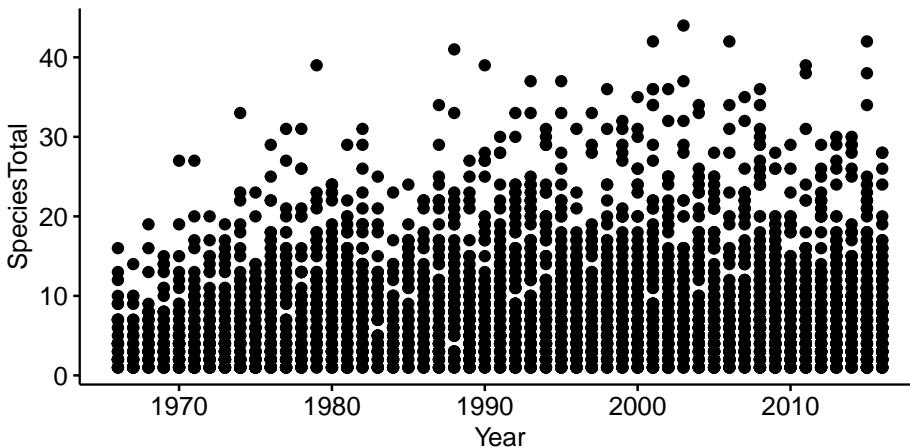
BBS_PA_SCTA <- BBS_PA2 %>% filter(Aou == 6080)
```

### 112.6.2.2 Plot filtered BBS data

We can see how many birds were observed on all the routes each year using ggscatter() from the ggpubr() package.

```
library(ggplot2)
library(ggpublisher)

ggscatter(data = BBS_PA_SCTA,
 y = "SpeciesTotal",
 x = "Year")
```



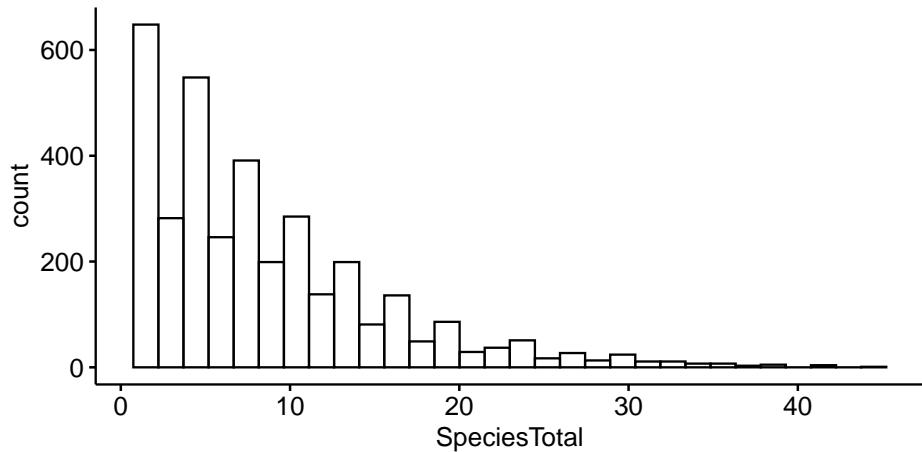
Note however that as discussed above this data frame only contains data from routes where SCTA was observed; when SCTA wasn't observed, nothing is recorded at all, hence no zeros. We can see this with summary()

```
summary(BBS_PA_SCTA)
```

##	Year	Aou	Route	SpeciesTotal
## Min.	:1966	Min. :6080	Min. : 1	Min. : 1.0
## 1st Qu.	:1983	1st Qu.:6080	1st Qu.: 29	1st Qu.: 3.0
## Median	:1996	Median :6080	Median : 56	Median : 7.0
## Mean	:1995	Mean :6080	Mean :113	Mean : 8.6
## 3rd Qu.	:2007	3rd Qu.:6080	3rd Qu.: 88	3rd Qu.:12.0
## Max.	:2016	Max. :6080	Max. :911	Max. :44.0

Or by looking at a histogram

```
gghistogram(BBS_PA_SCTA,
 x = "SpeciesTotal")
```



### 112.6.2.3 Focal year rows

For this exercise we are only going to consider data from 2006, since that is the year that the landcover data we'll be merging with comes from. (notes that "Year" has a capital "Y" while the rest is lowercase).

```
BBS_PA_SCTA_2 <- BBS_PA_SCTA %>% filter(Year == 2006)
```

```
BBS_PA_SCTA_2 <- BBS_PA_SCTA %>% dplyr::filter(Year == 2006)
```

We can see that we've greatly reduced the size of the original dataframe

```
dim(BBS_PA)
```

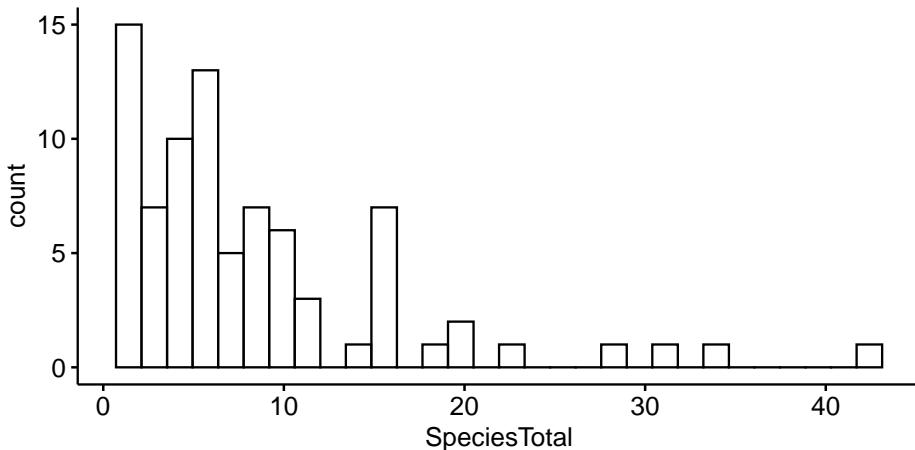
```
[1] 244185 8
dim(BBS_PA_SCTA)
```

```
[1] 3535 4
dim(BBS_PA_SCTA_2)
```

```
[1] 82 4
```

### 112.6.2.4 Plot filtered SCTA data

```
gghistogram(data = BBS_PA_SCTA_2,
 x = "SpeciesTotal")
```



### 112.6.3 Clean Landcover data

- We'll step back from the bird data now and switch to the data on the type of landcover surrounding the BBS routes.
- There are a large number of columns of data in the landcover data frame
- Note that a couple of the columns names aren't very good: "RT..NAME" and "RT..LENG."
- Columns that start with "NLCD" have data on the number of pixels of a certain land cover class surrounding a route. There are about 15 or so of these, for everything from open water to tundra.
- See ?BBS\_PA\_landcover\_1km for information on each of these

```
names(BBS_PA_landcover_1km)
```

```
[1] "Route" "BCR" "State" "RT..NAME" "RT..LENG." "NLCD.0"
[7] "NLCD.11" "NLCD.12" "NLCD.21" "NLCD.22" "NLCD.23" "NLCD.24"
[13] "NLCD.31" "NLCD.41" "NLCD.42" "NLCD.43" "NLCD.52" "NLCD.71"
[19] "NLCD.81" "NLCD.82" "NLCD.90" "NLCD.95" "SUM"
```

#### 112.6.3.1 Isoalte focal columns:forest cover

Let's focus on the columns that pertain to forest cover, which are contained in columns NLCD41, NLCD42, NLCD43. These represent three different types of forest, e.g. deciduous, coniferous, mixed. You find out information on them from the help file for the dataset using ?BBS\_PA\_landcover\_1km.

We can subset these columns as we did for the BBS bird count data using select(). Note that there is a period between "NLCD" and the number, and that "SUM" is in all caps.

```
BBS_PA_landcover_1km_2 <- BBS_PA_landcover_1km %>%
 select(Route, NLCD.41, NLCD.42, NLCD.43, SUM)
```

### 112.6.3.2 Add up total forest cover

Let's add up the NLCD.41, NLCD.42, and NLCD.43 columns

```
forest.total <- rowSums(BBS_PA_landcover_1km_2[c("NLCD.41",
 "NLCD.42",
 "NLCD.43")])
```

We can add this new column to the dataframe like this

```
BBS_PA_landcover_1km_2$forest.total <- forest.total
```

And see that it's there using head()

```
head(BBS_PA_landcover_1km_2)
```

	Route	NLCD.41	NLCD.42	NLCD.43	SUM	forest.total
## 1	1	44953	563	797	98591	46313
## 2	2	25119	1130	48	82345	26297
## 3	3	34154	521	78	86328	34753
## 4	4	39813	1695	60	88526	41568
## 5	5	32244	5892	3518	97708	41654
## 6	6	50529	4674	13320	87503	68523

The “SUM” column tells us the total number of GIS pixels were actually used to determine all of the landcover in the BBS\_PA\_landcover\_1km\_2 dataframe. This varies a bit so we will actually want to use the percentage of forest cover, not the raw total. We can calculate the percentage like this

```
forest.percent <- BBS_PA_landcover_1km_2$forest.total / BBS_PA_landcover_1km_2$SUM
```

And add it to the dataframe like this

```
BBS_PA_landcover_1km_2$forest.percent <- forest.percent
```

Note that we can actually do the math and add the new data to the dataframe in one step like this:

```
BBS_PA_landcover_1km_2$forest.percent <- BBS_PA_landcover_1km_2$forest.total / BBS_PA_
```

We don't need the individual columns for each of the three cover classes anymore (NLCD.41, 42, 43) so we can use select() to focus just on our new forest.percent and forest.total columns

We'll call this new dataframe BBS\_PA\_landcover\_1km\_3

```
BBS_PA_landcover_1km_3 <- BBS_PA_landcover_1km_2 %>%
 select(Route, forest.total, SUM, forest.percent,
 NLCD.41, NLCD.42, NLCD.43)
```

### 112.6.4 Merge BBS data and landcover data

We have reduced the size and made changes two dataframe: BBS\_PA, with counts of birds, and BBS\_PA\_landcover\_1km, which has information on the habitats around BBS routes in PA.

Now we'll make a new dataframe that *combines* these two separate data sets. This is one of the most powerful features of R - taking big sets of data and with a few lines of codes merging them into a new data set.

We'll do this using the full\_join() command from the dplyr package. All we need to do is 1) tell full\_join the names of the two data sets and 2) tell the function what column is shared between the data sets.

The fact that the "Route" column occurs in both data sets allows them to be matched up.

```
BBS_PA_SCTA_3 <- full_join(BBS_PA_SCTA_2 ,
 BBS_PA_landcover_1km_3,
 by = "Route")
```

Look at the dataframe

```
head(BBS_PA_SCTA_3)
```

	Year	Aou	Route	SpeciesTotal	forest.total	SUM	forest.percent	NLCD.41
## 1	2006	6080	2	6	26297	82345	0.319	25119
## 2	2006	6080	3	6	34753	86328	0.403	34154
## 3	2006	6080	4	3	41568	88526	0.470	39813
## 4	2006	6080	5	1	41654	97708	0.426	32244
## 5	2006	6080	6	8	68523	87503	0.783	50529
## 6	2006	6080	8	5	56098	87248	0.643	49029
##			NLCD.42	NLCD.43				
## 1		1130		48				
## 2		521		78				
## 3		1695		60				
## 4		5892		3518				
## 5		4674		13320				
## 6		4304		2765				

Compare the size of this dataframe and the original landcover dataframe.

```
#the BBS data that were merged
dim(BBS_PA_SCTA_2)

[1] 82 4

#the landcover data that were merged
dim(BBS_PA_landcover_1km_3)

[1] 137 7
```

```
#the final merged dataframe
dim(BBS_PA_SCTA_3)
```

```
[1] 137 10
```

### 112.6.5 Dealing with NAs

If we look at our new dataframe we see that some of our rows now contains NAs

```
summary(BBS_PA_SCTA_3)
```

```
Year Aou Route SpeciesTotal forest.total
Min. :2006 Min. :6080 Min. : 1 Min. : 1.0 Min. : 1563
1st Qu.:2006 1st Qu.:6080 1st Qu.: 35 1st Qu.: 3.0 1st Qu.:35308
Median :2006 Median :6080 Median : 69 Median : 6.0 Median :51222
Mean :2006 Mean :6080 Mean :137 Mean : 8.3 Mean :47706
3rd Qu.:2006 3rd Qu.:6080 3rd Qu.:132 3rd Qu.:10.0 3rd Qu.:62998
Max. :2006 Max. :6080 Max. :911 Max. :42.0 Max. :84033
NA's :55 NA's :55 NA's :55 NA's :55
SUM forest.percent NLCD.41 NLCD.42
Min. : 63828 Min. :0.019 Min. : 1377 Min. : 59
1st Qu.: 84435 1st Qu.:0.403 1st Qu.:27990 1st Qu.: 572
Median : 87008 Median :0.590 Median :40555 Median : 1707
Mean : 86994 Mean :0.555 Mean :39383 Mean : 2891
3rd Qu.: 89201 3rd Qu.:0.711 3rd Qu.:50529 3rd Qu.: 4871
Max. :110663 Max. :0.965 Max. :73426 Max. :11389
##
NLCD.43
Min. : 0
1st Qu.: 531
Median : 1579
Mean : 5432
3rd Qu.: 8346
Max. :39534
##
```

This occurred b/c the original BBS data only contains data when a species is observed - if it isn't seen, nothing is entered. So each "NA" in the new dataframe we made represents a route for which, in 2006, the SCTA wasn't observed.

Its easy to fix the Year and Aou columns because the all have the same values. All of the years = 2006, and all of the Aou columns = 6080, for Scarlet tanager. The following code will fill in any of the missing values

```
BBS_PA_SCTA_3$Year <- 2006
BBS_PA_SCTA_3$Aou <- 6080
```

Actually, since the code "6080" isn't very meaningful, let's add the letters

“SCTA” to a column to make it easy to remember what we are looking at. Let’s make a new column called “name” and put “SCTA” in it.

```
BBS_PA_SCTA_3$name <- "SCTA"
```

We’ll make this a factor variable

```
BBS_PA_SCTA_3$name <- factor(BBS_PA_SCTA_3$name)
```

Summary will show us what we’ve done: now there are no NAs in Year or Aou and there’s a new column called name

```
summary(BBS_PA_SCTA_3)
```

```
Year Aou Route SpeciesTotal forest.total
Min. :2006 Min. :6080 Min. : 1 Min. : 1.0 Min. : 1563
1st Qu.:2006 1st Qu.:6080 1st Qu.: 35 1st Qu.: 3.0 1st Qu.:35308
Median :2006 Median :6080 Median : 69 Median : 6.0 Median :51222
Mean :2006 Mean :6080 Mean :137 Mean : 8.3 Mean :47706
3rd Qu.:2006 3rd Qu.:6080 3rd Qu.:132 3rd Qu.:10.0 3rd Qu.:62998
Max. :2006 Max. :6080 Max. :911 Max. :42.0 Max. :84033
NA's :55
SUM forest.percent NLCD.41 NLCD.42
Min. : 63828 Min. :0.019 Min. : 1377 Min. : 59
1st Qu.: 84435 1st Qu.:0.403 1st Qu.:27990 1st Qu.: 572
Median : 87008 Median :0.590 Median :40555 Median : 1707
Mean : 86994 Mean :0.555 Mean :39383 Mean : 2891
3rd Qu.: 89201 3rd Qu.:0.711 3rd Qu.:50529 3rd Qu.: 4871
Max. :110663 Max. :0.965 Max. :73426 Max. :11389
##
NLCD.43 name
Min. : 0 SCTA:137
1st Qu.: 531
Median : 1579
Mean : 5432
3rd Qu.: 8346
Max. :39534
##
```

To fill in the NAs for the SpeciesTotal column (the counts of the number of birds) requires a new function: `is.na()`. `is.na()` determines if a row in a column has NA or it doesn’t. `is.na()` returns “TRUE” whenever there is an NA

```
is.na(BBS_PA_SCTA_3$SpeciesTotal)
```

```
[1] FALSE FALSE
[13] FALSE FALSE
[25] FALSE FALSE
[37] FALSE FALSE
```

```
[49] FALSE FALSE
[61] FALSE FALSE
[73] FALSE TRUE TRUE
[85] TRUE TRUE
[97] TRUE TRUE
[109] TRUE TRUE
[121] TRUE TRUE
[133] TRUE TRUE TRUE TRUE TRUE TRUE
```

We can use `is.na` with a function from `dplyr` called `mutate()` to change these NAs to 0s.

**This code is pretty complex, actually, so don't worry if you don't get it the 1st try** Actually, since its pretty trick for beginners, I've made a new function that does it more simply (see next chunk of code)

```
BBS_PA_SCTA_4 <- NA_to_zero(dat = BBS_PA_SCTA_3, column = "SpeciesTotal")
```

If you compare the dataset `BBS_PA_SCTA_3` to `BBS_PA_SCTA_4` that was made with `NA_to_zero()` you can see that the NAs were removed

```
#with NAs
summary(BBS_PA_SCTA_3$SpeciesTotal)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
```

```
1.0 3.0 6.0 8.3 10.0 42.0 55
```

```
#with NAs removed by NA_to_zero()
summary(BBS_PA_SCTA_4$SpeciesTotal)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
0 0 2 5 7 42
```

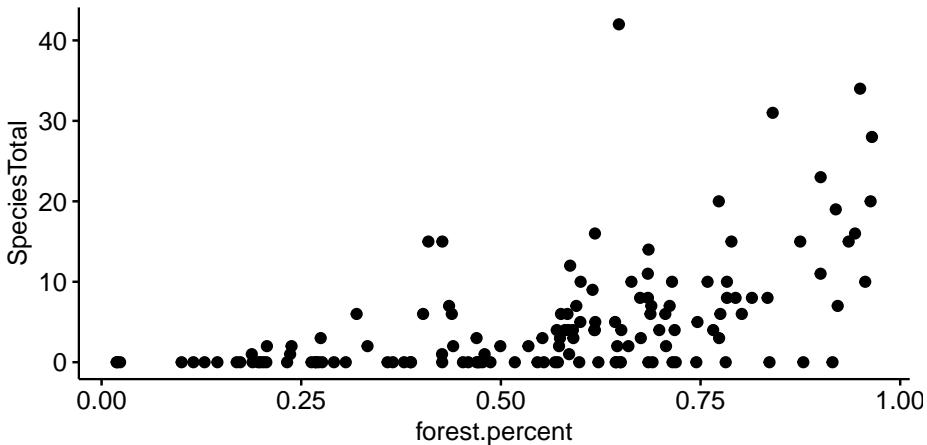
Since the above code is rather long, the following function from the `wildlifeR` package will do the same thing

```
BBS_PA_SCTA_3 <- NA_to_zero(dat = BBS_PA_SCTA_3,
 column = "SpeciesTotal")
```

### 112.6.6 PLOT relationship between SCTA and forest cover

We can now finally check out the relationship between the Scarlet tanager and forest cover

```
ggscatter(data = BBS_PA_SCTA_3,
 y = "SpeciesTotal",
 x = "forest.percent")
```



### 112.6.7 Re-organizing rows

- Dataframes typically put the information columns to the left and the data columns to the right
- We can reorder our columns in BBS\_PA\_SCTA\_3 using select()
- We can also rename columns using the selection function using the format “new.name = old.name”
- Its generally good to use a consistent format for all of your column names
- I also think its good to avoid uppercase letters
- Using using standard abbreviations is also good b/c shorter names are easier to type; here I'll use “spp” instead of “species” , “for” instead of forest, and “tot” instead of “total”

```
BBS_PA_SCTA_4 <- BBS_PA_SCTA_3 %>% select(year = Year,
 aou = Aou,
 route = Route,
 name = name,
 spp.tot = SpeciesTotal,
 for.tot = forest.total,
 NLCD.sum = SUM,
 for.percent = forest.percent)
```

```
BBS_PA_SCTA_4 <- BBS_PA_SCTA_3 %>% dplyr::select(year = Year,
 aou = Aou,
 route = Route,
 name = name,
 spp.tot = SpeciesTotal,
 for.tot = forest.total,
 NLCD.sum = SUM,
 for.percent = forest.percent)
```

### 112.6.8 Saving to .csv

When we have a dataset prepared its a good idea to save it to a spreadsheet as a .csv file for safe keeping. This is done with the write.csv() function

```
write.csv(BBS_PA_SCTA_4, file = "SCTA_vs_forest_cover.csv")
```

# Chapter 113

## Bookdown Notes

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 113. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter ??.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

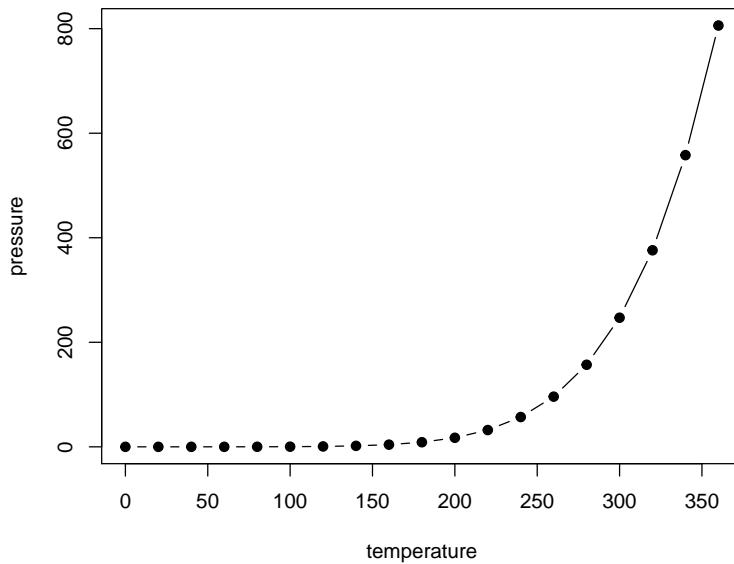


Figure 113.1: Here is a nice figure!

Table 113.1: Here is a nice table!

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
setosa.1	5.1	3.5	1.4	0.2	setosa
setosa.2	4.9	3.0	1.4	0.2	setosa
setosa.3	4.7	3.2	1.3	0.2	setosa
setosa.4	4.6	3.1	1.5	0.2	setosa
setosa.5	5.0	3.6	1.4	0.2	setosa
setosa.6	5.4	3.9	1.7	0.4	setosa
setosa.7	4.6	3.4	1.4	0.3	setosa
setosa.8	5.0	3.4	1.5	0.2	setosa
setosa.9	4.4	2.9	1.4	0.2	setosa
setosa.10	4.9	3.1	1.5	0.1	setosa
setosa.11	5.4	3.7	1.5	0.2	setosa
setosa.12	4.8	3.4	1.6	0.2	setosa
setosa.13	4.8	3.0	1.4	0.1	setosa
setosa.14	4.3	3.0	1.1	0.1	setosa
setosa.15	5.8	4.0	1.2	0.2	setosa
setosa.16	5.7	4.4	1.5	0.4	setosa
setosa.17	5.4	3.9	1.3	0.4	setosa
setosa.18	5.1	3.5	1.4	0.3	setosa
setosa.19	5.7	3.8	1.7	0.3	setosa
setosa.20	5.1	3.8	1.5	0.3	setosa

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 113.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 113.1.

```
knitr::kable(
 head(iris, 20), caption = 'Here is a nice table!',
 booktabs = TRUE
)
```

You can write citations, too. For example, we are using the `bookdown` package (Xie, 2020) in this sample book, which was built on top of R Markdown and `knitr` (Xie, 2015).

## 113.1 Inline R code

### 113.1.1 Code not run, just for example

Because of this, some lynx data is embedded within R and easy to access: all you have to do is type `data(lynx)`

## 113.2 Redact expressions or lines of code

<https://bookdown.org/yihui/rmarkdown-cookbook/hook-hide.html>

Expressions  
Lines of code

## 113.3 Do not stop on error

<https://bookdown.org/yihui/rmarkdown-cookbook/opts-error.html>

```
1 + "a"
Error in 1 + "a": non-numeric argument to binary operator
```

## 113.4 Style code blocks and text output

<https://bookdown.org/yihui/rmarkdown-cookbook/chunk-styling.html#chunk-styling>

```
mtcars[1:5, "mpg"]
[1] 21.0 21.0 22.8 21.4 18.7
```

## 113.5 Embed a web page

<https://bookdown.org/yihui/rmarkdown-cookbook/include-url.html>

## 113.6 Emojii

```
emo::ji("warning")
##
```

<https://unpkg.com/emojilib@3.0.2/dist/emoji-en-US.json>

Warning **Note:** Unlike a spreadsheet you cannot edit the data when it is called up using `View()`

Question There's a problem with my graph though - can you spot it? It only really becomes apparent when you connect the dots with a line.

## 113.7 Comment out text

highlight text then Cmd + Shift + C

### 113.8 Review biology through charts

chargaff1951

Higgs 2009 amino acids

### 113.9 TODO

go through all my repos to look for addiationl code

blast-stats project <https://github.com/brouwern/blaststats>

<https://github.com/brouwern/carnivorephylo>

<https://github.com/brouwern/CaseStudiesEcoStats>

## Chapter 114

# Packages that could be worth learning

### 114.1 Formatting output stats: xplain

<http://www.zuckarelli.de/xplain/index.html> [http://www.zuckarelli.de/xplain/xplain\\_cheatsheet.pdf](http://www.zuckarelli.de/xplain/xplain_cheatsheet.pdf)

### 114.2 Processing images: magick

<https://cran.r-project.org/web/packages/magick/vignettes/intro.html>

This has a lot of dependencies and can take a long time to set up.

```
install.packages("magick",dependencies = T)
```

Here's an example of how to resize some png files used in this book

```
#get names of files
files <- list.files(path = "./images/Medley1998csv",pattern = "Medley1998",full.names = T)

#make sure i just have png files, not the ppt
files <- files[grep("png",files)]

#look at file
print(image_read(files[1]))

newfiles <- gsub("Medley1998csv-", "Medley1998csv-800x",files)

for(i in 1:length(files)){
 image_write(image_resize(image_read(files[i]), "800x"))}
```

```
, path = newfiles[i]
, format = "png")
}
```

<https://discoveringthegenome.org/glossary/Bioinformatics> <http://www.informatics.jax.org/glossary> <http://rosalind.info/glossary/> <https://rpubs.com/k8huber/335128>

ENTREZ: an integrated collection of databases hosted by NCBI

NCBI: National Center for Biotechnology Information.

setwd getwd list.files read.csv ls dim(), names() and summary()

install.packages() RCurl::getURL() scatter.smooth

software library syntax quoted text Git GitHub repository repo software repository software developers developers open source CRAN source code source (verb) development version (of software) version control .csv .xls .xlsx spreadsheet GUI point-and-click interface working directory wd

sequence database: a compilation of macromolecular sequences which can be searched. Most databases are online and can be searched multiple ways. Searching by an accession numbers provides direct access to a particular sequences. Databases can often be searched by keywords to look for sequences that have been flagged with particular features. Databases can also be searched by algorithm to look for sequences that are similar to a sequence of interest.

accession number: a unique identifier for a gene, protein or mRNA transcript.

R: a statistical programming language used for statistics, data science, bioinformatics, machine learning, and some areas of computational biology. R is an interpreted language typically run from scripts. The term R is also used to refer to the software used to run R code. Like Python, R is an open source language. Unlike Python, R is not a fully functional programming language and has limited ability to make stand-alone applications. It does have a powerful extension for making web apps called Shiny.

RStudio: is the name of the most popular IDE for the R language. RStudio is the name of both the software and the company which makes it.

Python: an open source language which is popular for data science, machine learning/deep learning, bioinformatics, and web programming. In contrast with R, Python is a fully featured language that can be used to build stand alone pieces of software; YouTube, Reddit, Spotify and many other websites rely heavily on Python.

RStudio Cloud: A web-based implementation of R and RStudio that allows you to use all of R and RStudio's functionality in a web browser. This allows R to be run on almost any device, including tablets and watches. It also allows multiple individuals to be provided with identical copies of R working environments

which prevents issues due to software incompatibilities between different users' computers.

interpreted language: a language where code is executed on the fly, often in small pieces or scripts. R, Python and Java Script are scripting languages. (This is a rough definition)

compiled language: a language where code is compiled into an executable file or program. Compiled languages are typically used to build stand-alone pieces of software. C, C++ and Java are compiled languages. (This is a rough definition)

IDE: Integrated Development Environment. A workspace or front end for coding which provides useful tools for programming, such as file management, code highlighting, tab completion, etc.

.R file: A basic R file containing only R code in the form of text (letters, numbers, basic symbols) with no embedded images, pictures or graphs. Basic installations of R only work with .R files.

.Rmd file: An R Markdown file.

.txt file: A basic file format that contains only letters, numbers, punctuation and basic symbols. No figures, images, can occur in .txt files.

HTML: Hyper Text Markup Language. Code used to build websites.

comment character: Character such as “#” (in R) which indicates that whatever follows it is not code but just words.

assignment: Indicating that concrete information should be stored in an abstract variable. For example, “x<-1” tells R to print return the number 1 any time the letter “x” is used. So, in this case, “x+x” yield 2.

assignment operator: Most R users currently use “<-”, though some prefer “=”. Python uses “=”.

bioinformatics computational biology literate programming FASTA file: Ubiquitous file format for macromolecular sequences. BLAST: Basic Local Alignment Search Tool. An algorithm and search tool for aligning macromolecular sequences. This allows researchers to search a database for similar sequences. To “BLAST a sequence” means to have the search engine compare a sequence against a database for similar sequences.

GUI: Graphical User Interface. A graphical front-end for a program, in particular where all operations are carried out via a point-and-click interface. Most programs for aligning sequences and building phylogenetic trees have a GUI.

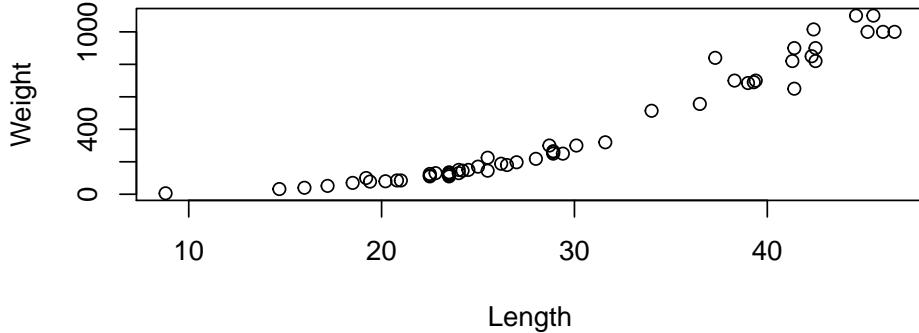


# Chapter 115

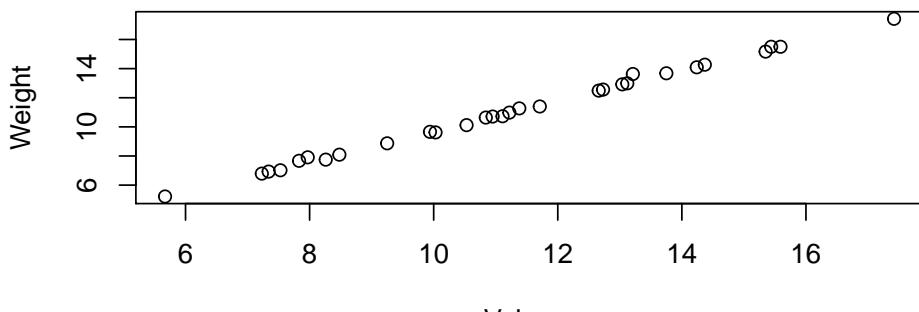
## Allometry data

Misc. notes on data related to allometry

```
library(Stat2Data)
data(Perch)
plot(Weight ~ Length, data = Perch)
```



```
library(Stat2Data)
data(Oysters) # linear; prediction
plot(Weight ~ Volume, data = Oysters)
```



BlueJays # allometry

## Chapter 116

# Errorplot datasets

Misc. notes on data useful for demonstrating errorplots

```
library(Stat2Data)
data(Tadpoles)
```

Venesky MD, Hanlon SM, Lynch K, Parris MJ, Rohr JR. (2013) “Optimal digestion theory does not predict the effect of pathogens on intestinal plasticity,” Biol Lett 9: 20130038. <http://dx.doi.org/10.1098/rsbl.2013.0038>



# Chapter 117

## error plot

MASS eagles Foraging Ecology of Bald Eagles Knight, R. L. and Skagen, S. K. (1988) Agonistic asymmetries and the foraging ecology of Bald Eagles. *Ecology* 69, 1188–1194.

```
install.packages("DAAG")
install.packages("carData")
install.packages("Stat2Data")
library(DAAG)
library(carData)
data(moths) # errorplot

library(DAAG)
data(moths) # errorplot

data(Depredations) # error plot
plot(number ~ longitude, data = Depredations)
```

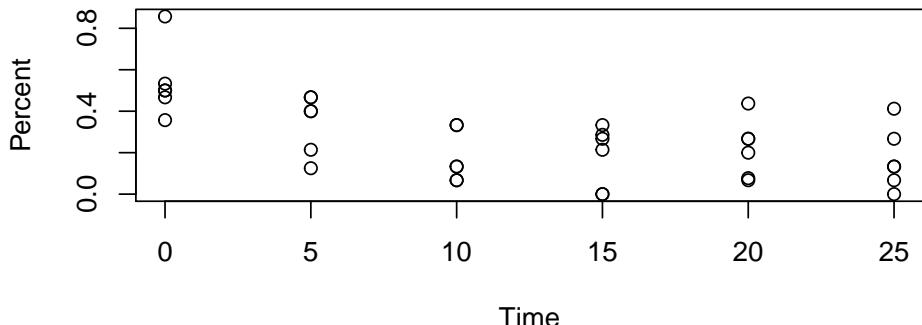
Harper, Elizabeth K. and Paul, William J. and Mech, L. David and Weisberg, Sanford (2008), Effectiveness of Lethal, Directed Wolf-Dpredation Control in Minnesota, *Journal of Wildlife Management*, 72, 3, 778-784. <http://dx.doi.org/10.2193/2007-273>

### 117.1 errorplot

```
library(Stat2Data)

#errorplot
data(Tadpoles)
data(Sparrows)
```

```
percentage over time
data(SeaSlugs)
plot(SeaSlugs)
```



```
data(Pines) # includes deer damage
```

```
data(MouseBrain) #counts
```

*#survival*

```
data(Leafhoppers)
```

```
data(BirdCalcium)
```

```
BirdCalcium #
```

#>	Bird	Sex	Hormone	Group	Ca
#> 1	1	male	no	M No	14.5
#> 2	2	male	no	M No	11.0
#> 3	3	male	no	M No	10.8
#> 4	4	male	no	M No	14.3
#> 5	5	male	no	M No	10.0
#> 6	6	male	yes	M Yes	32.0
#> 7	7	male	yes	M Yes	23.8
#> 8	8	male	yes	M Yes	28.8
#> 9	9	male	yes	M Yes	25.0
#> 10	10	male	yes	M Yes	29.3
#> 11	11	female	no	F No	16.5
#> 12	12	female	no	F No	18.4
#> 13	13	female	no	F No	12.7
#> 14	14	female	no	F No	14.0

```
#> 15 15 female no F No 12.8
#> 16 16 female yes F Yes 31.9
#> 17 17 female yes F Yes 26.2
#> 18 18 female yes F Yes 21.3
#> 19 19 female yes F Yes 35.8
#> 20 20 female yes F Yes 40.2

library(stevedata)
data(Amyloid)
Amyloid # errorplot
#> Group Abeta
#> 1 NCI 114
#> 2 NCI 41
#> 3 NCI 276
#> 4 NCI 0
#> 5 NCI 16
#> 6 NCI 228
#> 7 NCI 927
#> 8 NCI 0
#> 9 NCI 211
#> 10 NCI 829
#> 11 NCI 1561
#> 12 NCI 0
#> 13 NCI 276
#> 14 NCI 959
#> 15 NCI 16
#> 16 NCI 24
#> 17 NCI 325
#> 18 NCI 49
#> 19 NCI 537
#> 20 MCI 73
#> 21 MCI 33
#> 22 MCI 16
#> 23 MCI 8
#> 24 MCI 276
#> 25 MCI 537
#> 26 MCI 0
#> 27 MCI 569
#> 28 MCI 772
#> 29 MCI 0
#> 30 MCI 260
#> 31 MCI 423
#> 32 MCI 780
#> 33 MCI 1610
```

```
#> 34 MCI 0
#> 35 MCI 309
#> 36 MCI 512
#> 37 MCI 797
#> 38 MCI 24
#> 39 MCI 57
#> 40 MCI 106
#> 41 mAD 407
#> 42 mAD 390
#> 43 mAD 1154
#> 44 mAD 138
#> 45 mAD 634
#> 46 mAD 919
#> 47 mAD 1415
#> 48 mAD 390
#> 49 mAD 1024
#> 50 mAD 1154
#> 51 mAD 195
#> 52 mAD 715
#> 53 mAD 1496
#> 54 mAD 407
#> 55 mAD 1171
#> 56 mAD 439
#> 57 mAD 894
```

HawkTail HawkTail2 Goldenrod FruitFlies2 FruitFlies FlightResponse  
library(Stat2Data) Cuckoo # errorplot CrabShip # errorplot FishEggs # errorplot  
CO2 {datasets} R Documentation Carbon Dioxide Uptake in Grass Plants  
MASS genotype Rat Genotype Data # errorplot?

## 117.2 Health-related

boot cd4 # paired t-test  
library(Stat2Data) DiabeticDogs # repeated measures Forbath, N., A. B. Ken-shole, and G. Hetenyi, Jr. (1967), "Turnover lactic acid in normal and diabetic dogs calculated by two tracer methods," Am. J. Physiol. v. 212, pp.1179 - 1183.

# Chapter 118

## Datasets

Misc. notes on data that may be interesting

```
library(robustbase)
data(vaso) #Vaso Constriction Skin Data Set
```

multivariate; gene expression, clustering <https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression>

multivar, regression

```
library(robustbase)
data(toxicity)
```

Maguna, F.P., Núñez, M.B., Okulik, N.B. and Castro, E.A. (2003) Improved QSAR analysis of the toxicity of aliphatic carboxylic acids; Russian Journal of General Chemistry 73, 1792–1798.

```
library(robustbase) data(biomassTill)
```

```
library(robustbase) data(possumDiv)
```

library(robustbase) phosphor This dataset investigates the effect from inorganic and organic Phosphorus in the soil upon the phosphorus content of the corn grown in this soil, from Prescott (1975).

library(robustbase) NOxEmissions A typical medium sized environmental data set with hourly measurements of NOx pollution content in the ambient air.

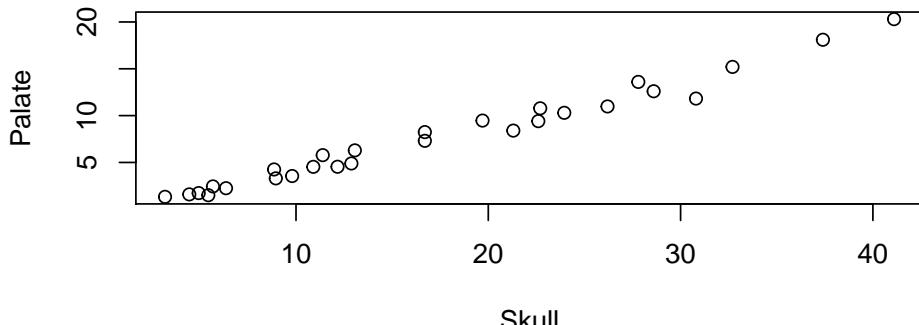
```
library(robustbase)
data(salinity)
```

This is a data set consisting of measurements of water salinity (i.e., its salt concentration) and river discharge taken in North Carolina's Pamlico Sound, recording some bi-weekly averages in March, April, and May from 1972 to 1977.

This dataset was listed by Ruppert and Carroll (1980). In Carrol and Ruppert (1985) the physical background of the data is described.

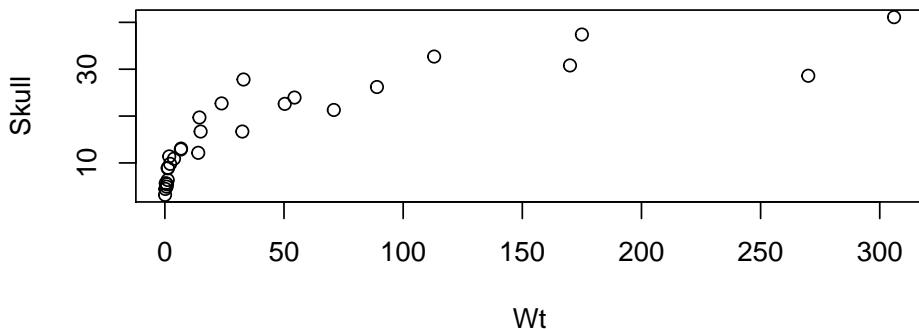
Fitch

```
library(Stat2Data)
data(Fitch)
plot(Palate ~ Skull, data = Fitch) # linear
```



Skull

```
plot(Skull ~ Wt, data = Fitch) # non-linear
```

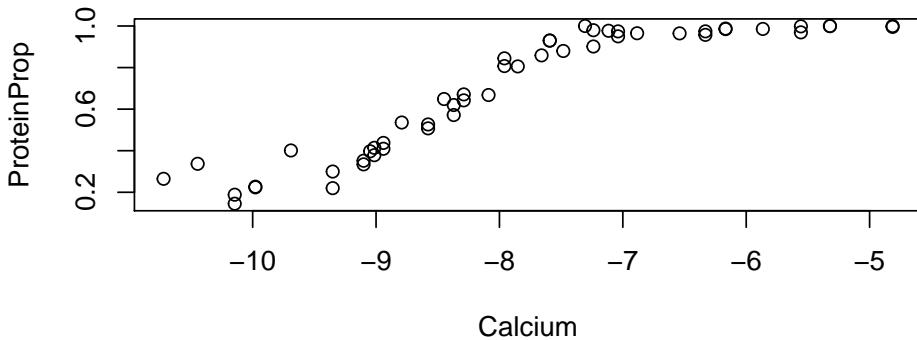


Wt

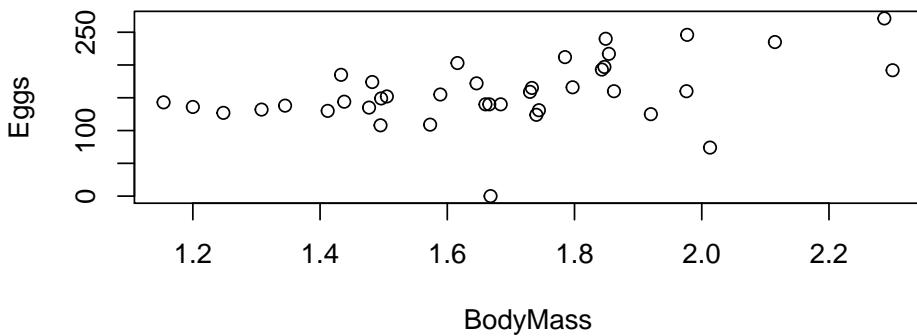
Fitch, W. Tecumseh (2000), "Skull dimensions in relation to body size in nonhuman mammals: The causal bases for acoustic allometry," *Zoology*, 103, 40-58.

## 118.1 Non-linear curve

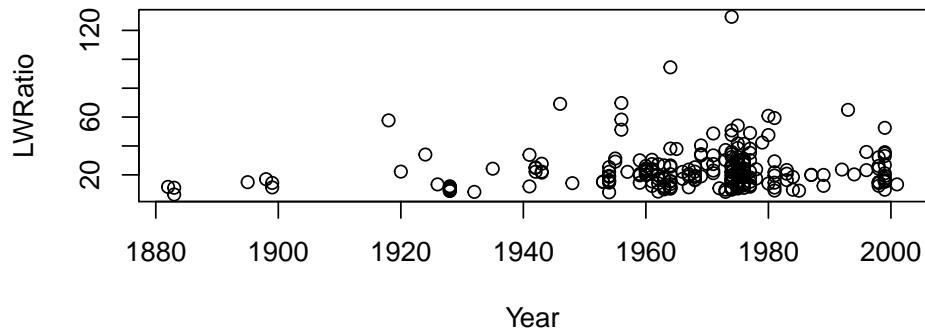
```
library(Stat2Data)
data(Fluorescence)
plot(ProteinProp ~ Calcium, data = Fluorescence)
```

**118.2 Noisy correlation**

```
library(Stat2Data)
data(MothEggs)
plot(Eggs ~ BodyMass, data = MothEggs)
```

**118.3 Regression; quantile regression ; climate change**

```
library(Stat2Data)
data(LeafWidth)
plot(LWRatio ~ Year, data = LeafWidth)
```



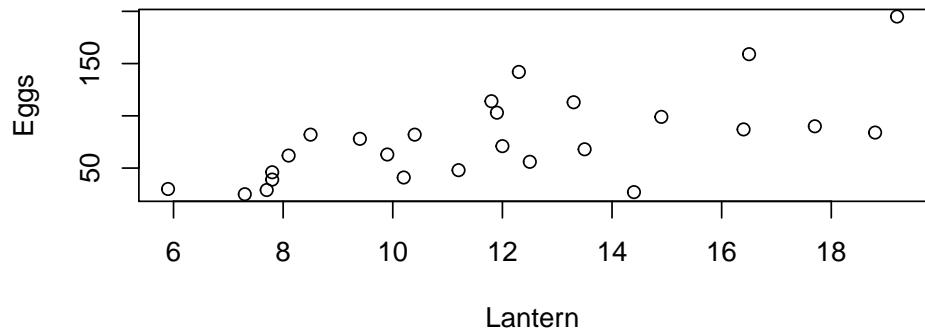
Guerin, G., Wen, H., Lowe, A. (2012), "Leaf morphology shift linked to climate change," Biol. Lett., 8, doi: 10.1098/rsbl.2012.0458

## 118.4 Logistic regression

```
library(Stat2Data)
data(Gunnels)
```

## 118.5 Regression; Fitness

```
library(Stat2Data)
data(GlowWorms)
plot(GlowWorms)
```



Hopkins J, Baudry G, Candolin U, Kaitala A. (2015), "I'm sexy and I glow it: female ornamentation in a nocturnal capital breeder," Biol. Lett. 11: 20150599. <http://dx.doi.org/10.1098/rsbl.2015.0599>

BrainpH #

# Chapter 119

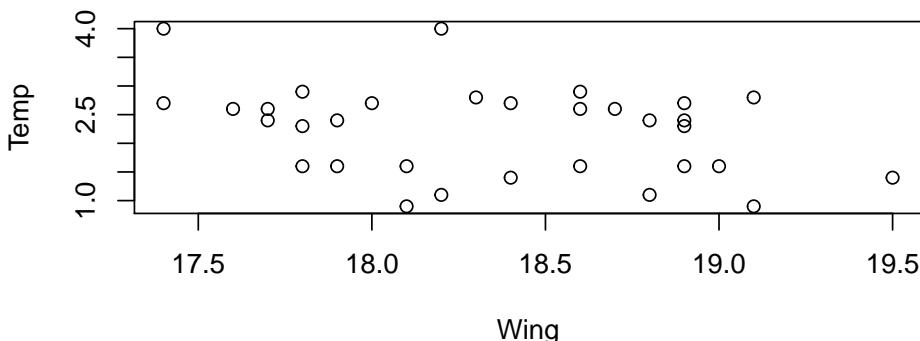
## 2 x 2 table

TMS # TMS Effects of transcranial magnetic stimulation (TMS) on migraine headaches Based on results in R. B. Lipton, et. al. (2010) “Single-pulse Transcranial Magnetic Stimulation for Acute Treatment of Migraine with Aura: A Randomised, Double-blind, Parallel-group, Sham-controlled Trial,” 9(4):373-380.

```
library(Stat2Data) CancerSurvival
```

ButterfliesBc # regression, climate chnage Digitized data from plots in Bowden, J. et al., “High-Arctic butterflies become smaller with rising temperatures”, published in Biology Letters 11: 20150574

```
library(Stat2Data)
data(ButterfliesBc)
plot(Temp ~ Wing, data = ButterfliesBc)
```



```
install.packages("stevedata")
library(stevedata)
data(Datasaurus)
Datasaurus
```

```

install.packages("survival")
library(survival)
install.packages("MASS")

MASS caith Colours of Eyes and Hair of People in Caithness
MASS GAGurine Level of GAG in Urine of Children
MASS snails Snail Mortality Dat
HistData Arbuthnot Arbuthnot's data on male and female birth ratios in London from 1629-1710.
HistData ChestSizes Chest measurements of 5738 Scottish Militiamen HistData Cholera William Farr's Data on Cholera in London, 1849

```

## 119.1 Heritability

HistData Galton Galton's data on the heights of parents and their children HistData GaltonFamilies Galton's data on the heights of parents and their children, by child HistData PearsonLee Pearson and Lee's data on the heights of parents and children classified by gender

## 119.2 Mendelian genetics

hwde IndianIrish Observed genotype frequencies at MN and S loci, for 2 populations hwde mendelABC Mendel's F2 trifactorial data for seed shape (A: round or wrinkled), cotyledon color (B: albumen yellow or green), and seed coat color (C: grey-brown or white)

## 119.3 Genomics

ade4 bacteria AA, codon, nucleotide counts for 43 spp of bacteria

## 119.4 Time series data library

<https://pkg.yangzhuoranyang.com/tsdl/reference/tsdl.html>

## 119.5 Data sets

ChickWeight Weight versus age of chicks on different diets chickwts Chicken Weights by Feed Type CO2 Carbon Dioxide Uptake in Grass Plants co2 Mauna Loa Atmospheric CO2 Concentration InsectSprays Effectiveness of Insect Sprays

iris Edgar Anderson's Iris Data  
iris3 Edgar Anderson's Iris Data  
islands Areas of the World's Major Landmasses  
LakeHuron Level of Lake Huron 1875-1972  
Loblolly Growth of Loblolly pine trees  
lynx Annual Canadian Lynx trappings  
1821-1934 npk Classical N, P, K Factorial Experiment  
Orange Growth of Orange Trees  
OrchardSprays Potency of Orchard Sprays  
PlantGrowth Results from an Experiment on Plant Growth  
precip Annual Precipitation in US Cities  
rivers Lengths of Major North American Rivers  
rock Measurements on Petroleum Rock Samples  
trees Girth, Height and Volume for Black Cherry Trees  
women Average Heights and Weights for American Women

<http://www.zoology.ubc.ca/~whitlock/ABD/teaching/datasets.html>



# Chapter 120

## Multivariate

Misc. notes on data related to multivariate analyses / visualization

MASS waders Counts of Waders at 15 Sites in South Africa R.W. Summers, L.G. Underhill, D.J. Pearson and D.A. Scott (1987) Wader migration systems in south and eastern Africa and western Asia. Wader Study Group Bulletin 49 Supplement, 15–34.

MASS crabs Morphological Measurements on Leptograpsus Crabs Campbell, N.A. and Mahon, R.J. (1974) A multivariate study of variation in two species of rock crab of genus Leptograpsus. Australian Journal of Zoology 22, 417–425.

```
install.packages("HistData")
library(HistData)

install.packages("robustbase")
library(robustbase)
data(possumDiv) # multiple regression, multivar

plot(Diversity ~ Habitat, data = possumDiv)
```

Lindenmayer, D. B., Cunningham, R. B., Tanton, M. T., Nix, H. A. and Smith, A. P. (1991) The conservation of arboreal marsupials in the montane ash forests of the central highlands of victoria, south-east australia: III. The habitat requirements of leadbeater's possum *gymnobelideus leadbeateri* and models of the diversity and abundance of arboreal marsupials. Biological Conservation 56, 295–315.

Lindenmayer, D. B., Cunningham, R. B., Tanton, M. T., Smith, A. P. and Nix, H. A. (1990) The conservation of arboreal marsupials in the montane ash forests of the victoria, south-east australia, I. Factors influencing the occupancy of trees with hollows, Biological Conservation 54, 111–131.



# Chapter 121

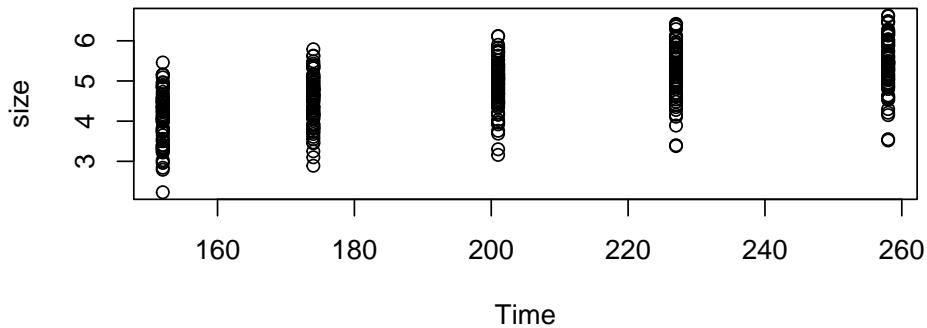
## Datasets - regression

Misc. notes on data related to regression.

### 121.1 Repeated measures

MASS Sitka Growth Curves for Sitka Spruce Trees in 1988

```
library(MASS)
data(Sitka)
plot(size ~ Time, data = Sitka)
```



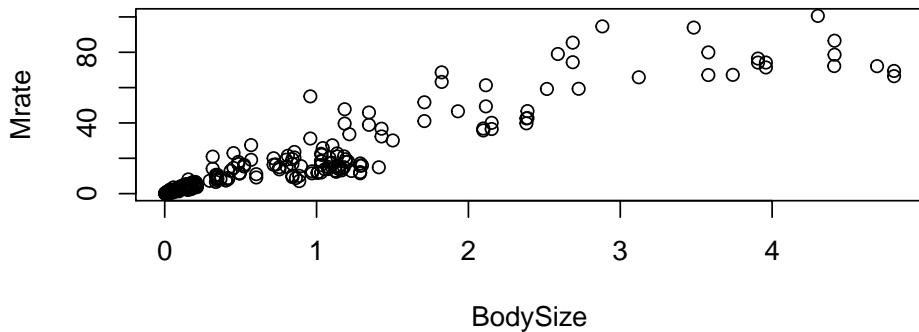
MASS Sitka89 Growth Curves for Sitka Spruce Trees in 1989

MASS cats Anatomical Data from Domestic Cats # regression

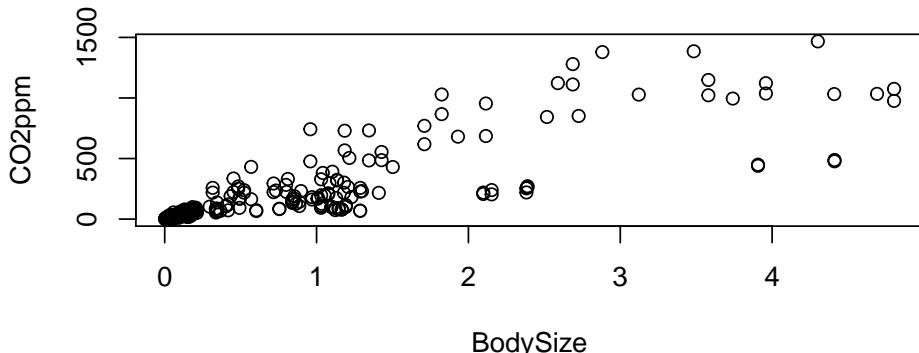
### 121.2 Multiple regression

```
library(Stat2Data)
data(MetabolicRate)
```

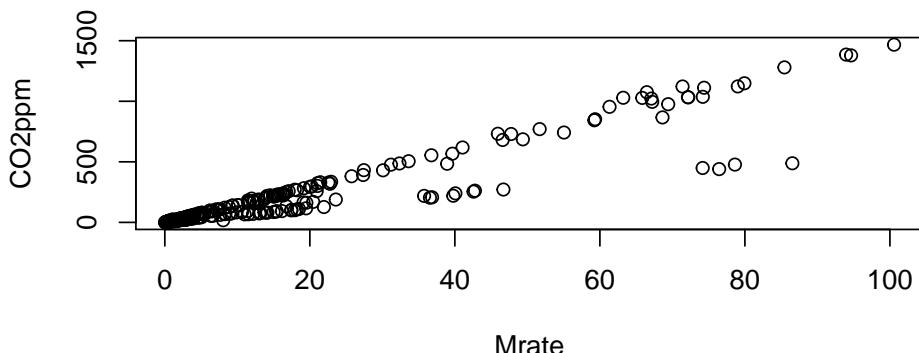
```
plot(Mrate ~ BodySize, data = MetabolicRate)
```



```
plot(CO2ppm ~ BodySize, data = MetabolicRate)
```



```
plot(CO2ppm ~ Mrate, data = MetabolicRate)
```



```
library(Stat2Data)
Fertility ## regression, multiple regression
```

```
library(Stat2Data)
data(ElephantsMF)
data(ElephantsFB)
```

```
ElephantsMF # regression, multiple regression
#> Age Height Sex
#> 1 1.40 120.0 M
#> 2 17.50 227.0 M
#> 3 12.75 235.0 M
#> 4 11.17 210.0 M
#> 5 12.67 220.0 M
#> 6 12.67 189.0 M
#> 7 12.25 225.0 M
#> 8 12.17 204.0 M
#> 9 28.17 265.9 M
#> 10 11.67 233.0 M
#> 11 16.33 241.0 M
#> 12 12.58 212.0 M
#> 13 25.42 251.5 M
#> 14 12.42 201.0 M
#> 15 26.50 257.7 M
#> 16 12.58 249.0 M
#> 17 11.58 206.0 M
#> 18 12.25 212.0 M
#> 19 11.58 238.0 M
#> 20 11.58 217.0 M
#> 21 26.67 268.6 M
#> 22 25.67 237.2 M
#> 23 15.25 257.0 M
#> 24 28.75 302.4 M
#> 25 11.33 201.0 M
#> 26 12.08 238.0 M
#> 27 12.50 200.0 M
#> 28 24.17 253.1 M
#> 29 11.75 202.0 M
#> 30 9.75 224.0 M
#> 31 9.25 206.0 M
#> 32 9.00 180.0 M
#> 33 9.50 210.0 M
#> 34 9.50 208.0 M
#> 35 23.92 281.6 M
#> 36 9.08 195.0 M
#> 37 26.08 304.1 M
#> 38 8.83 212.0 M
#> 39 8.17 193.0 M
#> 40 18.83 251.4 M
#> 41 25.42 294.1 M
#> 42 18.75 219.3 M
#> 43 8.25 181.0 M
```

```
#> 44 8.33 188.0 M
#> 45 24.17 287.1 M
#> 46 8.17 193.0 M
#> 47 21.25 272.9 M
#> 48 18.67 271.3 M
#> 49 8.33 199.0 M
#> 50 21.33 228.7 M
#> 51 8.50 194.0 M
#> 52 18.75 258.1 M
#> 53 8.42 162.0 M
#> 54 6.83 161.0 M
#> 55 17.25 296.8 M
#> 56 7.17 199.0 M
#> 57 18.75 221.4 M
#> 58 7.42 181.0 M
#> 59 7.00 183.0 M
#> 60 0.75 84.0 M
#> 61 4.67 170.0 M
#> 62 4.67 167.0 M
#> 63 0.67 99.0 M
#> 64 4.67 155.0 M
#> 65 0.67 115.0 M
#> 66 11.08 206.6 M
#> 67 0.01 91.0 M
#> 68 10.25 200.0 M
#> 69 0.17 86.0 M
#> 70 0.25 85.0 M
#> 71 17.25 266.8 M
#> 72 10.75 227.6 M
#> 73 0.17 96.0 M
#> 74 3.67 163.0 M
#> 75 10.58 233.1 M
#> 76 10.00 209.9 M
#> 77 9.50 213.0 M
#> 78 12.17 217.6 M
#> 79 11.50 234.9 M
#> 80 9.83 175.0 M
#> 81 7.33 151.3 M
#> 82 5.75 171.6 M
#> 83 11.33 225.9 M
#> 84 7.67 205.0 M
#> 85 10.08 232.8 M
#> 86 4.83 168.0 M
#> 87 7.25 188.2 M
#> 88 4.75 176.9 M
```

```
#> 89 6.75 205.5 M
#> 90 9.42 218.4 M
#> 91 4.58 136.4 M
#> 92 9.33 199.7 M
#> 93 7.42 199.9 M
#> 94 9.50 178.0 M
#> 95 9.50 227.5 M
#> 96 8.75 204.3 M
#> 97 7.42 205.5 M
#> 98 8.50 210.3 M
#> 99 6.33 165.8 M
#> 100 8.58 223.2 M
#> 101 7.33 171.1 M
#> 102 1.58 119.2 M
#> 103 7.08 162.6 M
#> 104 6.58 165.9 M
#> 105 3.83 152.3 M
#> 106 5.92 171.6 M
#> 107 6.00 164.6 M
#> 108 5.67 197.5 M
#> 109 0.08 108.1 M
#> 110 3.00 136.4 M
#> 111 0.25 85.2 M
#> 112 11.08 217.9 F
#> 113 3.08 176.8 M
#> 114 5.25 164.0 M
#> 115 2.33 132.9 F
#> 116 2.42 144.8 F
#> 117 26.50 206.1 F
#> 118 11.75 207.0 F
#> 119 0.33 85.6 F
#> 120 2.42 127.0 M
#> 121 26.75 227.3 F
#> 122 1.25 114.0 M
#> 123 13.42 203.0 F
#> 124 28.42 228.3 F
#> 125 2.08 131.3 F
#> 126 7.50 182.5 F
#> 127 5.50 150.6 F
#> 128 27.25 226.2 F
#> 129 6.58 178.6 F
#> 130 0.67 104.0 M
#> 131 0.01 83.0 F
#> 132 0.33 84.1 M
#> 133 3.33 156.1 F
```

```
#> 134 11.17 217.1 F
#> 135 0.17 96.2 M
#> 136 14.58 214.3 F
#> 137 0.58 99.0 M
#> 138 12.42 214.0 F
#> 139 8.42 178.0 F
#> 140 16.50 217.0 F
#> 141 25.42 240.4 F
#> 142 1.50 132.8 F
#> 143 23.33 238.1 F
#> 144 1.83 115.2 M
#> 145 26.75 235.4 F
#> 146 0.08 84.2 F
#> 147 6.75 166.2 F
#> 148 11.42 213.3 F
#> 149 16.50 222.2 F
#> 150 17.67 215.0 F
#> 151 1.30 125.0 F
#> 152 8.33 200.0 F
#> 153 27.80 265.0 F
#> 154 1.33 137.0 F
#> 155 28.25 215.9 F
#> 156 26.08 216.0 F
#> 157 0.50 97.3 F
#> 158 8.75 207.7 F
#> 159 26.67 220.5 F
#> 160 20.17 232.0 F
#> 161 21.50 227.1 F
#> 162 19.58 223.8 F
#> 163 27.33 225.6 F
#> 164 8.67 193.4 F
#> 165 2.42 160.8 F
#> 166 0.10 82.0 M
#> 167 8.25 175.0 F
#> 168 17.50 200.1 F
#> 169 0.33 122.4 M
#> 170 25.25 218.0 F
#> 171 15.50 217.6 F
#> 172 21.25 227.2 F
#> 173 5.17 181.5 F
#> 174 16.50 228.5 F
#> 175 0.67 105.3 F
#> 176 11.25 205.0 F
#> 177 12.25 225.0 F
#> 178 2.33 144.1 M
```

```
#> 179 7.33 173.7 F
#> 180 15.25 220.3 F
#> 181 2.25 106.3 F
#> 182 7.33 162.4 F
#> 183 30.25 277.8 F
#> 184 27.42 207.0 F
#> 185 3.25 155.6 F
#> 186 3.83 157.9 F
#> 187 29.25 226.1 F
#> 188 3.33 160.6 M
#> 189 7.67 205.2 F
#> 190 0.25 102.0 F
#> 191 9.67 197.0 F
#> 192 8.25 187.0 F
#> 193 26.42 210.3 F
#> 194 6.75 167.8 F
#> 195 12.33 217.0 F
#> 196 16.42 221.0 F
#> 197 11.33 196.0 F
#> 198 0.67 117.7 F
#> 199 16.75 232.0 F
#> 200 2.08 143.8 F
#> 201 8.42 168.0 F
#> 202 15.75 232.0 F
#> 203 21.42 249.4 F
#> 204 1.58 146.8 F
#> 205 24.67 192.5 F
#> 206 4.58 131.3 F
#> 207 18.75 216.3 F
#> 208 10.92 194.8 F
#> 209 8.67 175.0 F
#> 210 1.08 140.2 F
#> 211 0.75 99.0 F
#> 212 8.25 204.0 F
#> 213 15.42 214.0 F
#> 214 17.58 190.3 F
#> 215 21.92 226.2 F
#> 216 22.17 210.3 F
#> 217 9.75 200.2 F
#> 218 10.08 215.3 F
#> 219 0.50 75.5 M
#> 220 21.67 232.2 F
#> 221 0.25 90.0 F
#> 222 9.42 205.0 F
#> 223 11.42 203.0 F
```

```
#> 224 1.42 119.1 F
#> 225 1.50 117.8 F
#> 226 0.50 116.9 M
#> 227 9.33 229.5 F
#> 228 17.50 234.7 F
#> 229 25.25 241.9 F
#> 230 28.58 234.8 F
#> 231 0.33 84.3 M
#> 232 16.67 220.4 F
#> 233 1.50 114.9 F
#> 234 11.50 190.0 F
#> 235 25.83 243.7 F
#> 236 9.33 193.9 F
#> 237 1.50 104.8 F
#> 238 23.92 253.4 F
#> 239 0.25 110.4 M
#> 240 0.20 94.0 M
#> 241 0.10 91.0 F
#> 242 21.83 258.9 F
#> 243 18.58 215.2 F
#> 244 1.83 150.2 M
#> 245 12.83 199.0 F
#> 246 18.75 216.9 F
#> 247 7.42 187.6 F
#> 248 1.58 145.3 M
#> 249 12.50 178.0 F
#> 250 10.08 208.4 F
#> 251 3.33 169.9 M
#> 252 8.75 207.5 F
#> 253 8.42 176.0 F
#> 254 12.25 214.0 F
#> 255 0.50 94.0 F
#> 256 16.75 198.4 F
#> 257 10.67 198.4 F
#> 258 3.92 163.0 F
#> 259 30.50 227.0 F
#> 260 2.08 132.4 M
#> 261 0.01 79.0 F
#> 262 0.33 105.0 M
#> 263 11.33 215.0 F
#> 264 8.17 186.0 F
#> 265 2.20 140.0 M
#> 266 15.33 213.0 F
#> 267 11.42 208.0 F
#> 268 8.08 169.0 F
```

```

#> 269 9.58 192.0 F
#> 270 8.25 175.0 F
#> 271 26.58 250.8 F
#> 272 0.42 117.0 M
#> 273 0.04 91.5 M
#> 274 15.58 206.1 F
#> 275 32.17 261.0 F
#> 276 28.00 232.3 F
#> 277 0.08 75.6 F
#> 278 5.42 173.3 M
#> 279 25.58 221.6 F
#> 280 5.92 162.1 F
#> 281 7.58 193.3 F
#> 282 10.00 204.8 F
#> 283 25.08 259.2 F
#> 284 4.75 122.8 F
#> 285 10.50 209.2 F
#> 286 19.92 190.2 F
#> 287 21.25 225.5 F
#> 288 13.17 221.0 F
ElephantsFB # regression, multiple regression
#> Age Height Firstborn
#> 1 1.40 120.0 0
#> 2 17.50 227.0 0
#> 3 12.75 235.0 0
#> 4 11.17 210.0 0
#> 5 12.67 220.0 1
#> 6 12.67 189.0 1
#> 7 12.25 225.0 0
#> 8 12.17 204.0 0
#> 9 28.17 265.9 0
#> 10 11.67 233.0 1
#> 11 16.33 241.0 0
#> 12 12.58 212.0 0
#> 13 25.42 251.5 1
#> 14 12.42 201.0 1
#> 15 26.50 257.7 0
#> 16 12.58 249.0 0
#> 17 11.58 206.0 0
#> 18 12.25 212.0 0
#> 19 11.58 238.0 0
#> 20 11.58 217.0 0
#> 21 26.67 268.6 0
#> 22 25.67 237.2 1
#> 23 15.25 257.0 0

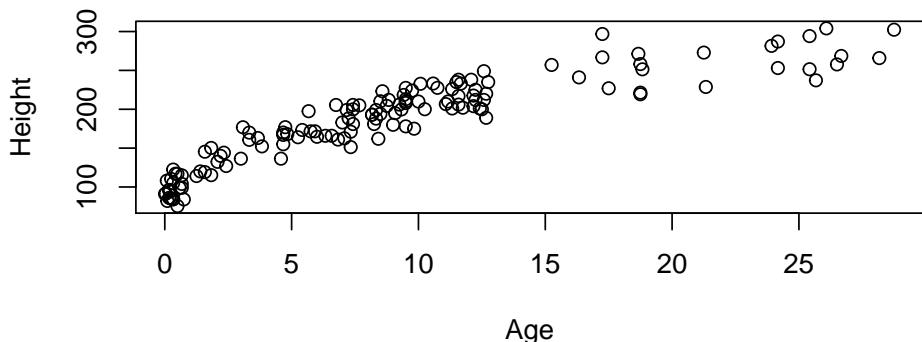
```

```
#> 24 28.75 302.4 0
#> 25 11.33 201.0 0
#> 26 12.08 238.0 0
#> 27 12.50 200.0 0
#> 28 24.17 253.1 0
#> 29 11.75 202.0 1
#> 30 9.75 224.0 1
#> 31 9.25 206.0 0
#> 32 9.00 180.0 0
#> 33 9.50 210.0 0
#> 34 9.50 208.0 0
#> 35 23.92 281.6 1
#> 36 9.08 195.0 1
#> 37 26.08 304.1 1
#> 38 8.83 212.0 0
#> 39 8.17 193.0 0
#> 40 18.83 251.4 0
#> 41 25.42 294.1 0
#> 42 18.75 219.3 1
#> 43 8.25 181.0 0
#> 44 8.33 188.0 0
#> 45 24.17 287.1 0
#> 46 8.17 193.0 0
#> 47 21.25 272.9 0
#> 48 18.67 271.3 0
#> 49 8.33 199.0 0
#> 50 21.33 228.7 0
#> 51 8.50 194.0 0
#> 52 18.75 258.1 0
#> 53 8.42 162.0 0
#> 54 6.83 161.0 1
#> 55 17.25 296.8 0
#> 56 7.17 199.0 0
#> 57 18.75 221.4 1
#> 58 7.42 181.0 1
#> 59 7.00 183.0 0
#> 60 0.75 84.0 0
#> 61 4.67 170.0 0
#> 62 4.67 167.0 0
#> 63 0.67 99.0 0
#> 64 4.67 155.0 0
#> 65 0.67 115.0 0
#> 66 11.08 206.6 0
#> 67 0.01 91.0 0
#> 68 10.25 200.0 0
```

```
#> 69 0.17 86.0 0
#> 70 0.25 85.0 0
#> 71 17.25 266.8 0
#> 72 10.75 227.6 0
#> 73 0.17 96.0 0
#> 74 3.67 163.0 0
#> 75 10.58 233.1 0
#> 76 10.00 209.9 0
#> 77 9.50 213.0 0
#> 78 12.17 217.6 0
#> 79 11.50 234.9 0
#> 80 9.83 175.0 0
#> 81 7.33 151.3 0
#> 82 5.75 171.6 0
#> 83 11.33 225.9 0
#> 84 7.67 205.0 0
#> 85 10.08 232.8 0
#> 86 4.83 168.0 1
#> 87 7.25 188.2 0
#> 88 4.75 176.9 0
#> 89 6.75 205.5 0
#> 90 9.42 218.4 0
#> 91 4.58 136.4 0
#> 92 9.33 199.7 1
#> 93 7.42 199.9 0
#> 94 9.50 178.0 0
#> 95 9.50 227.5 0
#> 96 8.75 204.3 0
#> 97 7.42 205.5 0
#> 98 8.50 210.3 0
#> 99 6.33 165.8 0
#> 100 8.58 223.2 0
#> 101 7.33 171.1 0
#> 102 1.58 119.2 1
#> 103 7.08 162.6 0
#> 104 6.58 165.9 1
#> 105 3.83 152.3 0
#> 106 5.92 171.6 0
#> 107 6.00 164.6 0
#> 108 5.67 197.5 0
#> 109 0.08 108.1 0
#> 110 3.00 136.4 0
#> 111 0.25 85.2 1
#> 112 3.08 176.8 0
#> 113 5.25 164.0 0
```

```
#> 114 2.42 127.0 1
#> 115 1.25 114.0 0
#> 116 0.67 104.0 0
#> 117 0.33 84.1 0
#> 118 0.17 96.2 1
#> 119 0.58 99.0 0
#> 120 1.83 115.2 0
#> 121 0.10 82.0 1
#> 122 0.33 122.4 0
#> 123 2.33 144.1 0
#> 124 3.33 160.6 0
#> 125 0.50 75.5 0
#> 126 0.50 116.9 0
#> 127 0.33 84.3 0
#> 128 0.25 110.4 0
#> 129 0.20 94.0 1
#> 130 1.83 150.2 0
#> 131 1.58 145.3 0
#> 132 3.33 169.9 0
#> 133 2.08 132.4 0
#> 134 0.33 105.0 0
#> 135 2.20 140.0 0
#> 136 0.42 117.0 0
#> 137 0.04 91.5 0
#> 138 5.42 173.3 0
```

```
plot(Height ~ Age, data = ElephantsFB)
```



Caterpillars # regression, multiple regression; nonlinear? BirdNest # regression, multiple regression

## 121.3 GLM

```
install.packages("robustbase")
library(robustbase)
data(possumDiv) # multiple regression, multivar

plot(Diversity ~ Habitat, data = possumDiv)
```

Lindenmayer, D. B., Cunningham, R. B., Tanton, M. T., Nix, H. A. and Smith, A. P. (1991) The conservation of arboreal marsupials in the montane ash forests of the central highlands of victoria, south-east australia: III. The habitat requirements of leadbeater's possum *gymnobelideus leadbeateri* and models of the diversity and abundance of arboreal marsupials. *Biological Conservation* 56, 295–315.

Lindenmayer, D. B., Cunningham, R. B., Tanton, M. T., Smith, A. P. and Nix, H. A. (1990) The conservation of arboreal marsupials in the montane ash forests of the victoria, south-east australia, I. Factors influencing the occupancy of trees with hollows, *Biological Conservation* 54, 111–131.



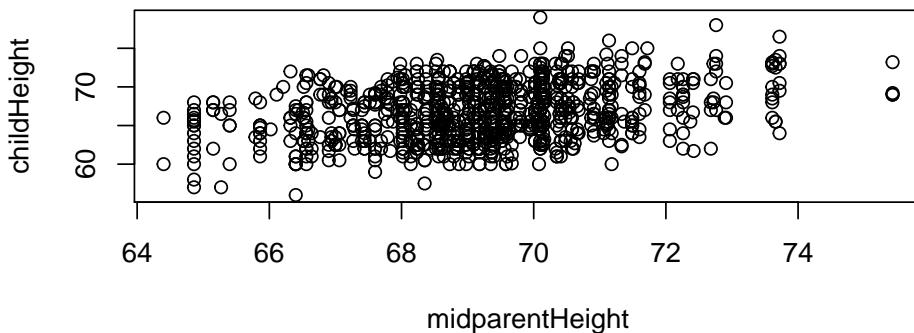
## Chapter 122

# Regression

Misc. notes on data related to regression.

HistData GaltonFamilies Galton's data on the heights of parents and their children, by child

```
install.packages("HistData")
library(HistData)
data(GaltonFamilies)
plot(childHeight ~ midparentHeight, data = GaltonFamilies)
```

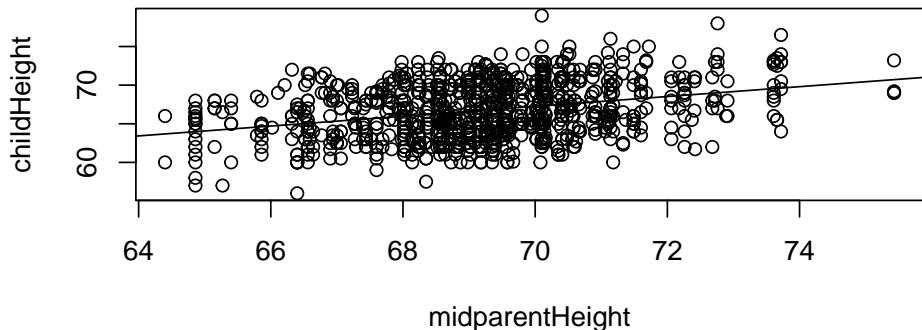


```
height_model <- lm(childHeight ~ midparentHeight, data = GaltonFamilies)
summary(height_model)
#>
#> Call:
#> lm(formula = childHeight ~ midparentHeight, data = GaltonFamilies)
#>
#> Residuals:
```

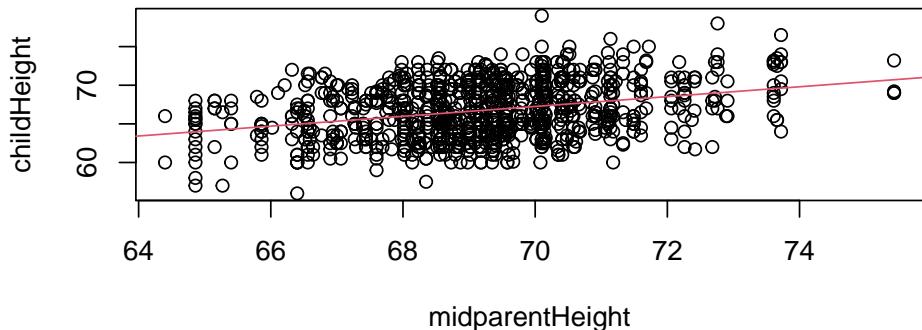
```
#> Min 1Q Median 3Q Max
#> -8.957 -2.699 -0.215 2.796 11.685
#>
#> Coefficients:
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 22.6362 4.2651 5.31 1.4e-07 ***
#> midparentHeight 0.6374 0.0616 10.35 < 2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 3.39 on 932 degrees of freedom
#> Multiple R-squared: 0.103, Adjusted R-squared: 0.102
#> F-statistic: 107 on 1 and 932 DF, p-value: <2e-16
```

```
coef(height_model)
#> (Intercept) midparentHeight
#> 22.636 0.637
```

```
plot(childHeight ~ midparentHeight, data = GaltonFamilies)
abline(height_model)
```



```
plot(childHeight ~ midparentHeight, data = GaltonFamilies)
abline(height_model, col = 2)
```



# Chapter 123

## Time series / climate changes

Misc. notes on data related to time series and climates change

### 123.0.1 Stat2Data

```
library(Stat2Data)
```

```
SeaIce CO2SouthPole # time series CO2Hawaii # time series CO2Germany #
time series, 1 year, daily CO2
```

Could stack CO2SouthPole, CO2Hawaii, CO2Germany

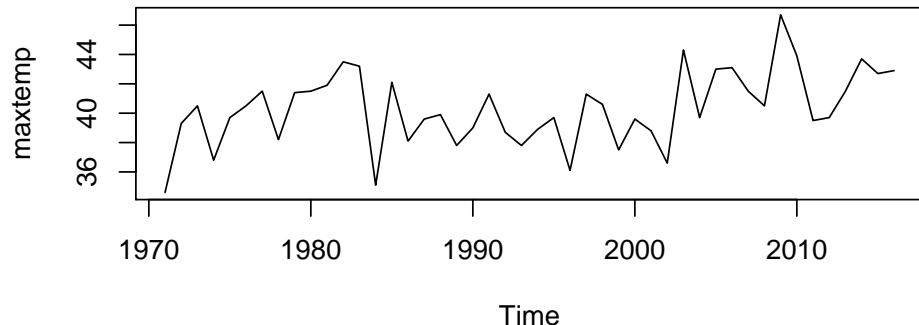
Reshape SeaIce

### 123.0.2 fpp2

“Do you want to install from sources the package which needs compilation?  
(Yes/no/cancel)”

```
install.packages("fpp2") # has lots of dependencies

library(fpp2)
data(maxtemp) # climate change
plot(maxtemp)
```



### 123.0.3 Package astsa: Applied Statistical Time Series Analysis

```
Spectral envelope analysis
#install.packages("astsa")
library(astsa)
data(EBV) #Entire Epstein-Barr Virus (EBV) Nucleotide Sequence
```

blood Daily Blood Work bnrfl1ebv Nucleotide sequence - BNRF1 Epstein-Barr  
bnrfl1hvs Nucleotide sequence - BNRF1 of Herpesvirus saimiri

- cardox ts objec; Monthly Carbon Dioxide Levels at Mauna Loa

cmort Cardiovascular Mortality from the LA Pollution study dna2vector  
Convert DNA Sequence to Indicator Vectors EBV Entire Epstein-Barr Virus  
(EBV) Nucleotide Sequence flu ts object; Monthly pneumonia and influenza  
deaths in the U.S., 1968 to... fmri fMRI - complete data set fmri1 fMRI Data  
Used in Chapter 1 globtemp Global mean land-ocean temperature deviations to  
2015 globtempl Global mean land (only) temperature deviations to 2015 gtemp  
Global mean land-ocean temperature deviations gtemp2 Global Mean Surface  
Air Temperature Deviations gtemp\_land Global mean land temperature  
deviations - updated to 2017 gtemp\_ocean Global mean ocean temperature  
deviations - updated to 2017 Hare Snowshoe Hare HCT Hematocrit Levels hor  
Hawaiian occupancy rates lap LA Pollution-Mortality Study

Lynx Canadian Lynx <http://people.whitman.edu/~hundledr/courses/M250F03/M250.html> <http://people.whitman.edu/~hundledr/courses/M250F03/LynxHare.txt> NOTE about the data: I have not been able to verify this data, but this is the data (or rather the graph) that is always cited. This particular set of data came from scanning in the graph from Odum's "Fundamentals of Ecology", p. 191 which is often cited. Odum says that his graph is taken from MacLulich's "Fluctuations in the numbers of varying hare", 1937, which is not widely available. Some authors caution that this data is actually a composition of several time series, and should probably not be analyzed as a whole, and that some of the lynx data was actually missing. It is said that the data was collected from Hudson's Bay historical records, and does not reflect animal

populations, but rather the number of pelts turned in for trading (a large number of which came from Native Americans- mentioned because there were some medical outbreaks during these years which could account for skewed data). The data is presented here with these cautions.

part Particulate levels from the LA pollution study PLT Platelet Levels polio Poliomyelitis cases in US rec Recruitment (number of new fish index) salt Salt Profiles saltemp Temperature Profiles so2 SO2 levels from the LA pollution study soi Southern Oscillation Index soiltemp Spatial Grid of Surface Soil Temperatures sunspotz Biannual Sunspot Numbers tempr Temperatures from the LA pollution study WBC White Blood Cell Levels



## Chapter 124

# Improving bas R plots: moving axes labels with `mtext()`

Some notes on adjusting margins and using the `mtext()` function to place axis labels closer the axes.



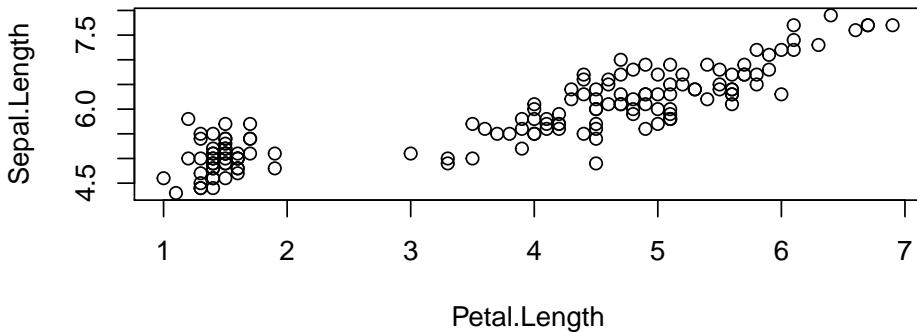
# Chapter 125

## Standard plot 1 panel plot

### 125.1 Issues

- Large margin above the plot
- Large distance between labels on the tick marks and the axis label

```
data(iris)
with(iris, plot(Sepal.Length ~ Petal.Length))
```

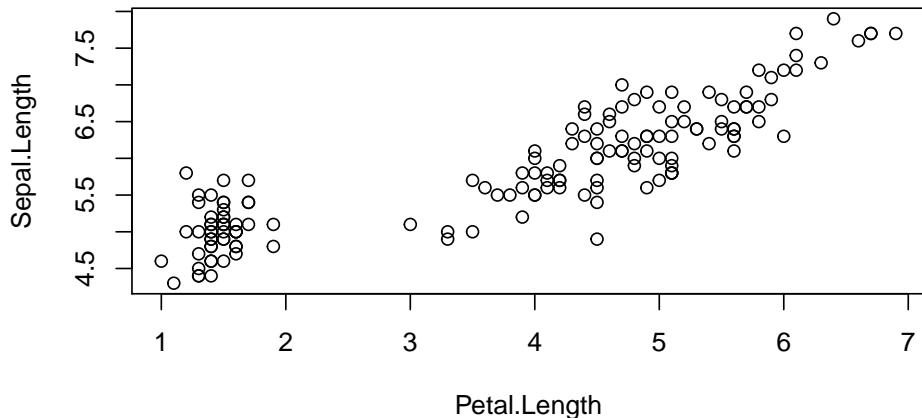


### 125.2 Change margins around plot

- use “`par(mar = c(lower, left,top,right))`”
- default is `mar = c(5, 4, 4, 2) + 0.1`.

Here, I decrease the top and left margins to 1.

```
par(mar = c(5,4,1,1))
with(iris, plot(Sepal.Length ~ Petal.Length))
```



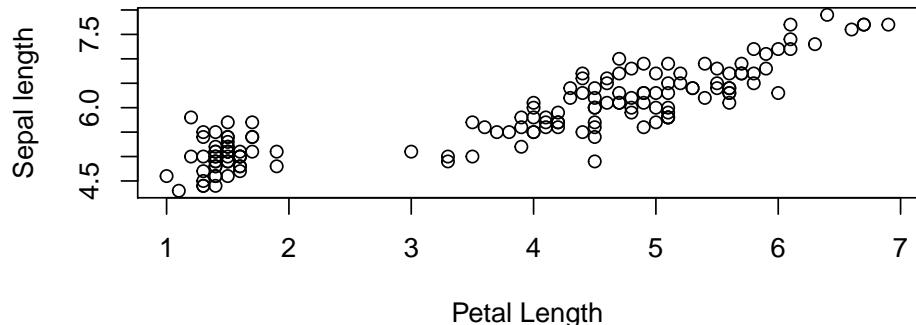
### 125.3 Move axis labels closer to axis

- set axis labels to `xlab = ""` and `ylab = ""`
- Add new labels using `mtext()`
- `mtext` stands for “margin text”
- `mtext` takes several arguments:

Here, I decrease the top and left margins to 1.

#### 125.3.1 Change axis labels

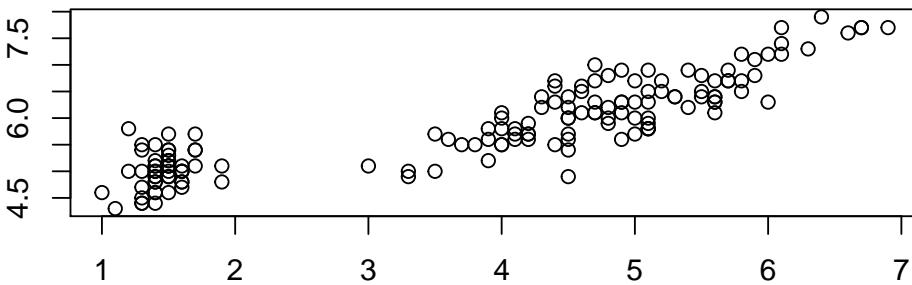
```
with(iris, plot(Sepal.Length ~ Petal.Length,
 xlab = "Petal Length",
 ylab = "Sepal length"))
```



#### 125.3.2 Plot with no axis labels

- `ylab = ""` sets it to print nothing

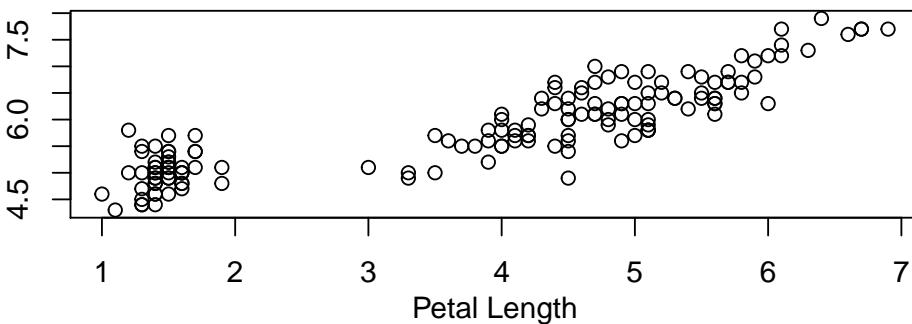
```
with(iris, plot(Sepal.Length ~ Petal.Length,
 xlab = "",
 ylab = ""))
```



### 125.3.3 Plot with x axis added with mtext()

- side = 1 sets it to the lower margin
- side = 2 would do the left
- side= 3 top, side = 4 right
- line = 2 sets it 2 lines below the axis
- it usually takes some trial and error to get it where you want it

```
with(iris, plot(Sepal.Length ~ Petal.Length,
 xlab = "",
 ylab = ""))
#add text
mtext(text = "Petal Length",
 side = 1,#side 1 = bottom
 line = 2)
```



### 125.3.4 Plot with y axis added with mtext()

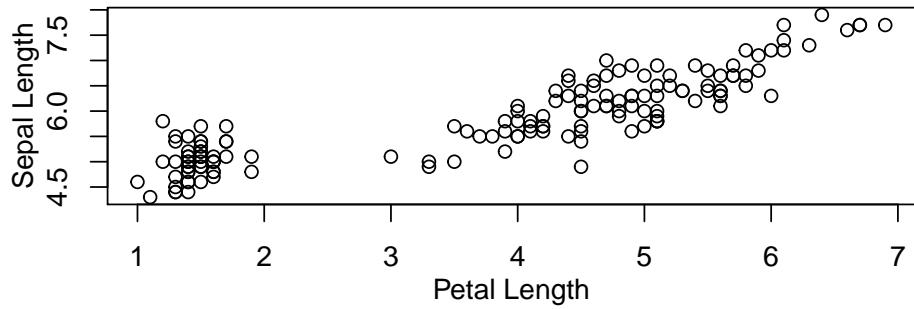
- side = 2 would do the left

```
with(iris, plot(Sepal.Length ~ Petal.Length,
 xlab = "",
 ylab = ""))

x axis
mtext(text = "Petal Length",
 side = 1, line = 2)

y axis

mtext(text = "Sepal Length",
 side = 2, #side 2 = left
 line = 2)
```



### 125.3.5 Reset margins

- Get rid of blank space by changing margins
- set mar = c(2,2,1,1)

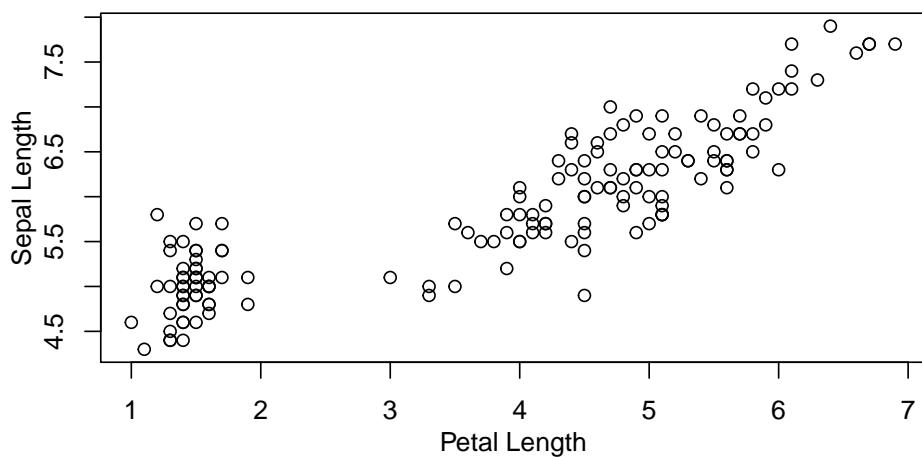
```
par(mar = c(3,3,1,1))

with(iris, plot(Sepal.Length ~ Petal.Length,
 xlab = "",
 ylab = ""))

x axis
mtext(text = "Petal Length",
 side = 1, line = 2)

y axis

mtext(text = "Sepal Length",
 side = 2, #side 2 = left
 line = 2)
```





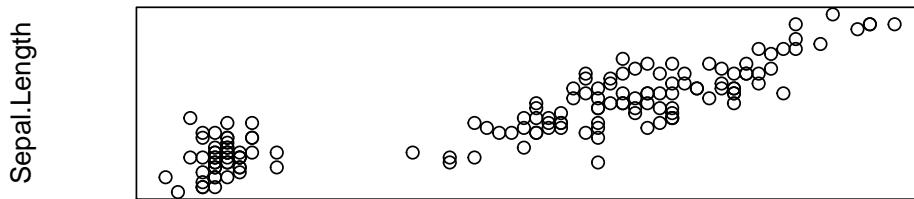
## Chapter 126

### No tick marks & tick labels

If for some reason you want no tick marks and no numbers along the axies, you can set `par(yaxt = "m")`

```
#change par
par(yaxt="n",#remove from y
 xaxt='n')#remove from x

with(iris, plot(Sepal.Length ~ Petal.Length))
```

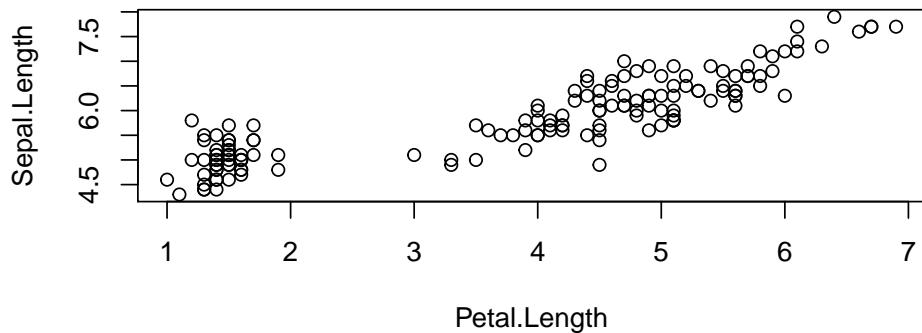


Petal.Length

Set it back tiwht `yaxt = "t"`

```
#change par
par(yaxt="t",#remove from y
 xaxt='t')#remove from x

with(iris, plot(Sepal.Length ~ Petal.Length))
```



# Chapter 127

## FAQ: base R

*Things to cover*

- boxplot - base, qplot, ggplot
- boxplot ordered
- histogram
- scatterplot
- scatterplot w/vert/hort/1:1 line
- scatterplot with regression lines
- scatterplot with spline smoother
- scatterplot matrix

### 127.1 Preliminaries

#### 127.1.1 Load Martin 1992 data

```
library(compbio4all)
data("martin1995")
martin <- martin1995
```

#### 127.1.2 Clean data

Change the “non-excavating” group to “secondary cavity”

```
martin$group <- gsub("non-excavating", "2ndary cav.", martin$group)
```

Change the “group-nesting” group to “ground”

```
martin$group <- gsub("group-nesting", "ground", martin$group)
```

Make some more changes to shorten the group names

```

martin$group <- gsub("Subcanopy or canopy","subcan-/canopy",martin$group)
martin$group <- gsub("shrub or low foliage","shrub",martin$group)
martin$group <- gsub("excavating","excav",martin$group)

```

Classify general latitudinal region as tropical, boreal, or temperate

```

martin$trop.or.temp <- "temperate"
i.tropics <- which(martin$lat < 23)
i.boreal <- which(martin$lat > 54)

martin$trop.or.temp[i.tropics] <- "tropics"
martin$trop.or.temp[i.boreal] <- "boreal"

```

## 127.2 R Plotting FAQ

### 127.2.1 Question

- T1) How do I make simple tables in R?
- T2) How do I make 2-way tables in R?
- T3) How do I make 3-way tables in R?

## 127.3 Tables

Our object martin has a column `group` giving the general location where each species in the dataframe nests and another columns for where the birds forage.

### 127.3.0.1 T1) How do I make simple 1-way tables in R?

We can make a simple table for where birds nest, where they forage and their general latitudinal region like this:

```

#Table for nest location
table(martin$group)
#>
#> 2ndary cav. excav ground shrub subcan-/canopy
#> 18 11 26 49 18

#Table for foraging location
table(martin$foraging.site)
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> 1 15 1 13 24 38 30

#Table for general latitudinal location

```

```
table(martin$trop.or.temp)
#>
#> boreal temperate tropics
#> 3 18 1
```

A nice command to learn to use in R is with(). Here's what it can do when making a table; If you were to read this out loud like you were telling R what to do, it would be something like this "With the dataframe called data, make a table from the group column." This isn't that helpful in this situation, but for 2-way tables and other tasks it is nice.

```
#Making a table with with()
with(martin, table(group))
#> group
#> 2ndary cav. excav ground shrub subcan-/canopy
#> 18 11 26 49 18
```

An aside on tables: The summary() command can also give you simple 1-way tables. However, the table() command will make a table out of whatever you give it, while summary() is more generic. For example, the way that these particular data have been loaded in, R is treating information that is in text form, like nesting location, as text. So when I use summary, I get this

```
summary(martin$group)
#> Length Class Mode
#> 122 character character
```

R is very picky about different kinds of data. In order for R to make a table from this column using the summary() command, you'd need to tell it that the data belong to different categories or different levels of "factor variable". This is done with the factor() command. here, I'll tell R that the group column is factor data and then make a table with summary()

```
#Change character data to a factor
martin$group <- factor(martin$group)

#Now I get a table from summary()
summary(martin$group)
#> 2ndary cav. excav ground shrub subcan-/canopy
#> 18 11 26 49 18
```

### 127.3.0.2 T2) How do I make simple 2-way tables in R?

We can cross classify the nesting groups and foraging groups very easily, aka, we can make a 2-way table.

Note that you have to be explicit about where each piece of data from by putting "martin\$" in front of both columns

```
#With nesting location in rows
table(martin$group, martin$foraging.site)
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 7 1 2 6 1 1
#> excav 0 0 0 10 1 0 0
#> ground 1 0 0 1 3 18 3
#> shrub 0 5 0 0 1 19 24
#> subcan-/canopy 0 3 0 0 13 0 2

#With nesting location in columns
table(martin$foraging.site, martin$group)
#>
#> 2ndary cav. excav ground shrub subcan-/canopy
#> 0 0 1 0 0
#> Aerial 7 0 0 5 3
#> Aquatic 1 0 0 0 0
#> Bark 2 10 1 0 0
#> Canopy 6 1 3 1 13
#> Ground 1 0 18 19 0
#> Shrub 1 0 3 24 2
```

You can save yourself some typing by using the with() command. If you read this outloud it would be “with the table dataframe, make a 2-way table from the group and foraging site columns”

```
with(martin, table(group, foraging.site))
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 7 1 2 6 1 1
#> excav 0 0 0 10 1 0 0
#> ground 1 0 0 1 3 18 3
#> shrub 0 5 0 0 1 19 24
#> subcan-/canopy 0 3 0 0 13 0 2
```

### 127.3.0.3 T3)How do I make 3-way tables in R?

You can cross-classify categorical data as many ways as you'd like. Let's make a 3-way classification based on nest type, foraging and lattidue.

```
table(martin$group, martin$foraging.site, martin$trop.or.temp)
#> , , = boreal
#>
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 0 0 0 0 0 0
```

```

#> excav 0 0 0 0 0 0 0
#> ground 0 0 0 0 0 3 0
#> shrub 0 0 0 0 0 0 0
#> subcan-/canopy 0 0 0 0 0 0 0
#>
#> , , = temperate
#>
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 7 1 2 6 1 1
#> excav 0 0 0 10 1 0 0
#> ground 0 0 0 1 3 15 3
#> shrub 0 5 0 0 1 19 24
#> subcan-/canopy 0 3 0 0 13 0 2
#>
#> , , = tropics
#>
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 0 0 0 0 0 0
#> excav 0 0 0 0 0 0 0
#> ground 1 0 0 0 0 0 0
#> shrub 0 0 0 0 0 0 0
#> subcan-/canopy 0 0 0 0 0 0 0

```

This can be simplified a bit using with()

```

with(martin, table(group, foraging.site, trop.or.temp))
#> , , trop.or.temp = boreal
#>
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 0 0 0 0 0 0
#> excav 0 0 0 0 0 0 0
#> ground 0 0 0 0 0 3 0
#> shrub 0 0 0 0 0 0 0
#> subcan-/canopy 0 0 0 0 0 0 0
#>
#> , , trop.or.temp = temperate
#>
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 7 1 2 6 1 1
#> excav 0 0 0 10 1 0 0
#> ground 0 0 0 1 3 15 3
#> shrub 0 5 0 0 1 19 24

```

```
#> subcan-/canopy 0 3 0 0 13 0 2
#>
#> , , trop.or.temp = tropics
#>
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> 2ndary cav. 0 0 0 0 0 0 0
#> excav 0 0 0 0 0 0 0
#> ground 1 0 0 0 0 0 0
#> shrub 0 0 0 0 0 0 0
#> subcan-/canopy 0 0 0 0 0 0 0
```

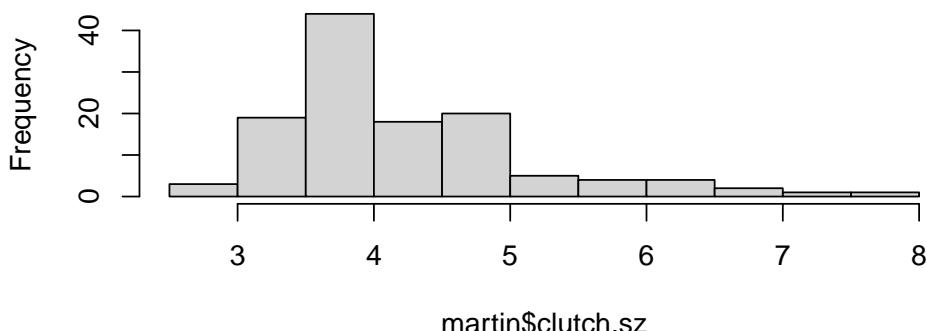
### 127.3.1 BP1) How do I make histograms in R?

The basics: R makes histograms in a snap in R, something that I do not know of an efficient way to do in Excel. Let's make a histogram of the different clutch sizes (number of eggs laid per nest) for the different species. Since I've forgotten exactly what I called this column I'll first figure that out using the names() function, then make a histogram with hist(). R automatically "bins" the data and plots the frequency of observations in each bin along the y-axis.

```
#Look at the names in the data dataframe
names(martin)
#> [1] "i.bird" "group" "spp" "clutch.sz"
#> [5] "inc.dur" "nest.dur" "nest.suc" "pred"
#> [9] "broods" "surv.adult" "lat" "foraging.site"
#> [13] "refs" "trop.or.temp"

#Make a histogram of clutch size
hist(martin$clutch.sz)
```

**Histogram of martin\$clutch.sz**

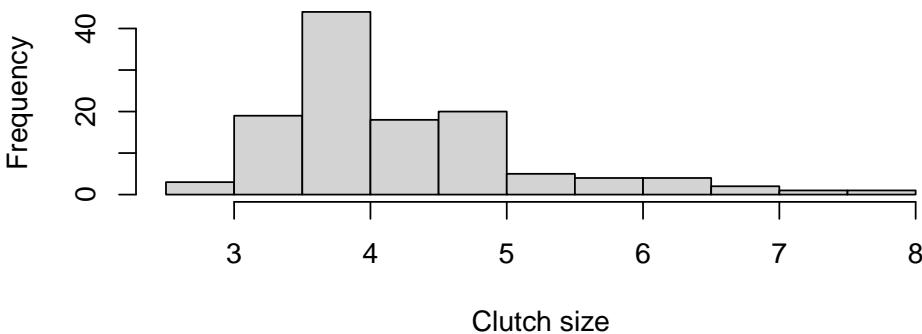


Finesse: Changing titles R automatically assigns names to everything. Here I

change the x-axis (called the x label) and title, which is called the “main title”.

```
hist(martin$clutch.sz,
 xlab = "Clutch size",
 main = "data's clutch-size data")
```

**data's clutch-size data**

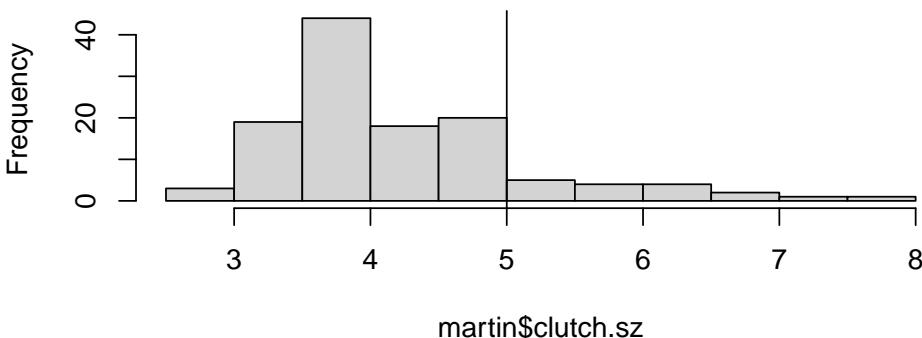


Finesse: Adding a vertical line The bird I study typically lays 5 eggs. To make it easy to compare my bird to other birds, I can add a vertical line at 5. This requires two separate commands. First I'll make the plot, then add the line. The line is made with the abline() command, which stands for “A-B line”, where A stands for the intercept of a line B stands for its slope. abline() can also make horizontal lines and vertical lines, but the programmers must have thought that abhvline() was too long of a name.

```
#Make histogram
hist(martin$clutch.sz)

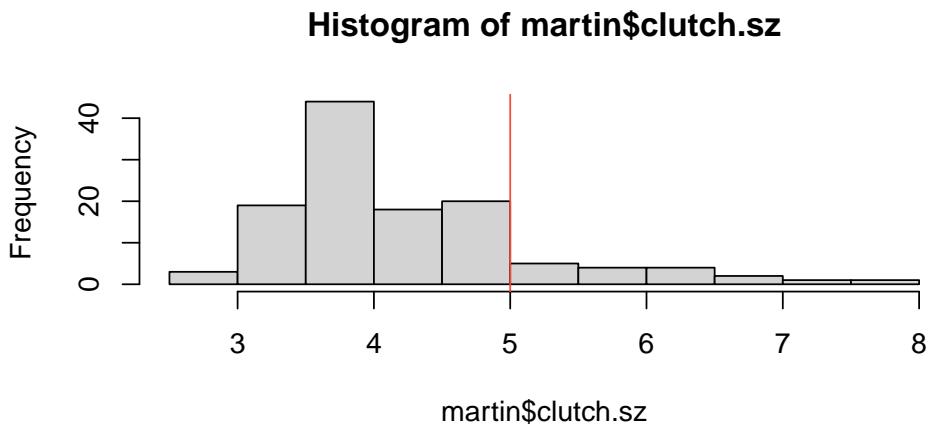
#Add vertical line at 5
abline(v = 5)
```

**Histogram of martin\$clutch.sz**



The color is black so it blends in. I can change the color using the col= command within abline line

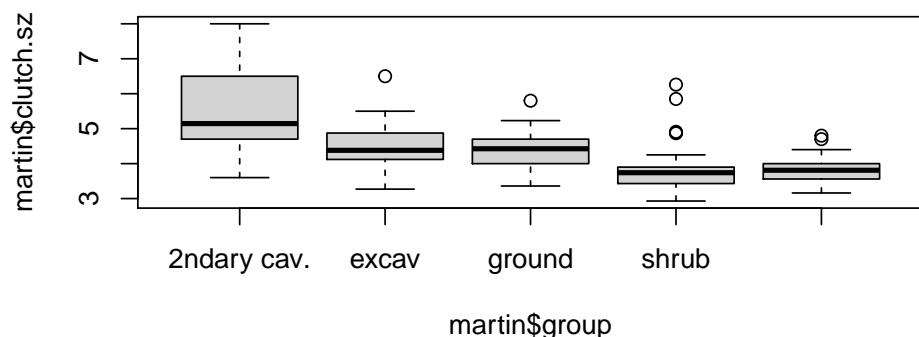
```
#Make histogram
hist(martin$clutch.sz)
abline(v = 5, col = 2)
```



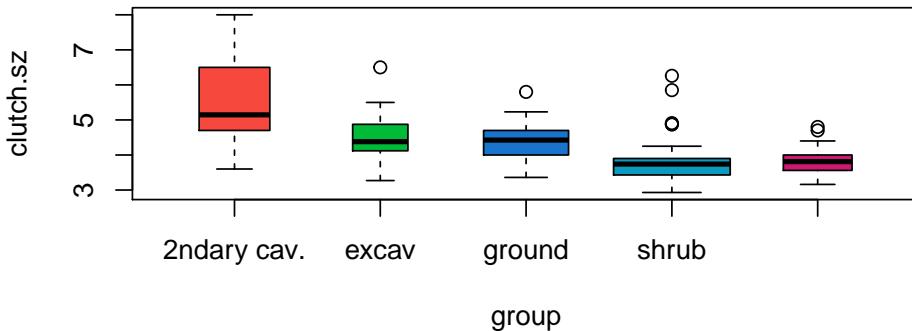
### 127.3.2 BP2) How do I make boxplot in R?

Oneliner:

```
boxplot(martin$clutch.sz ~ martin$group)
```

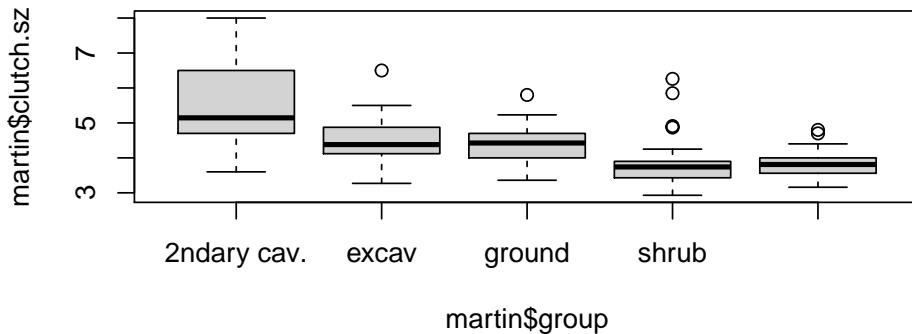


```
par(mfrow = c(1,1))
boxplot(clutch.sz ~ group, data = martin,
 varwidth = TRUE, #modify the width proportional to sample size
 notch = FALSE, #notches can be used to judge if medians are
 #different. Work best with relatively large sample
 #sizes.
 col = 2:7) #colors
```



**Details:** Boxplots are recommended by many statisticians as an excellent way to summarize the distribution of data. Boxplots also a breeze in R, while they are more or less impossible in Excel (I've seen instructions for them but it looks like a pain). Let's make a box plot of clutch size compared to birds from different nesting locations. The boxplot() function uses a tilda, ~ in it. Like many R functions, such as those for regression, the continuous variable goes to the left of ~ and the categorical variables go to the right.

```
boxplot(martin$clutch.sz ~ martin$group)
```



### 127.3.3 BP3) How do I make scatterplots in R?

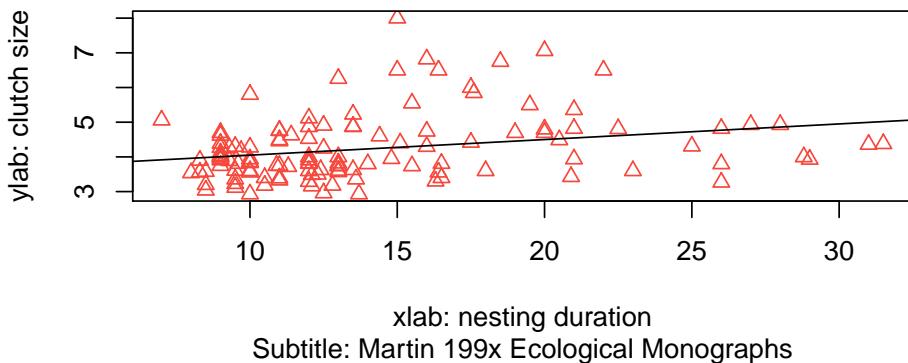
*Oneliner:* `plot(clutch.sz ~ nest.dur, data = martin)`

*Full code:*

```
#Plot
plot(clutch.sz ~ nest.dur, data = martin,
 main = "Main Title: Clutch size vs. nesting duration",
 sub = "Subtitle: Martin 199x Ecological Monographs",
 xlab = "xlab: nesting duration",
 ylab = "ylab: clutch size",
 col = 2, #change the color
 pch = 2) #change th shape of the symbol
```

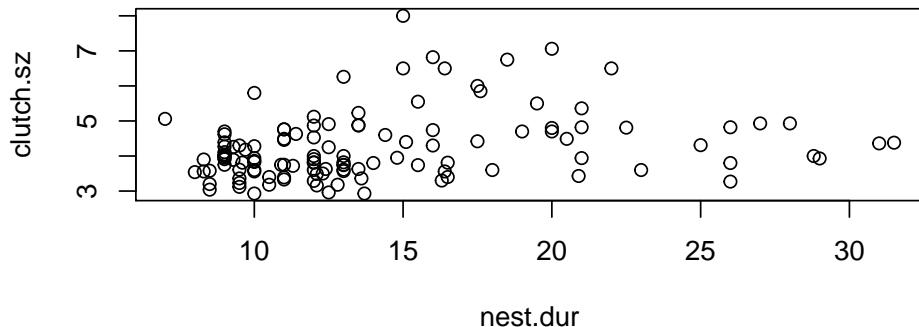
```
#Regression line
abline(a = 3.6, #intercept
 b = 0.045) #slope
```

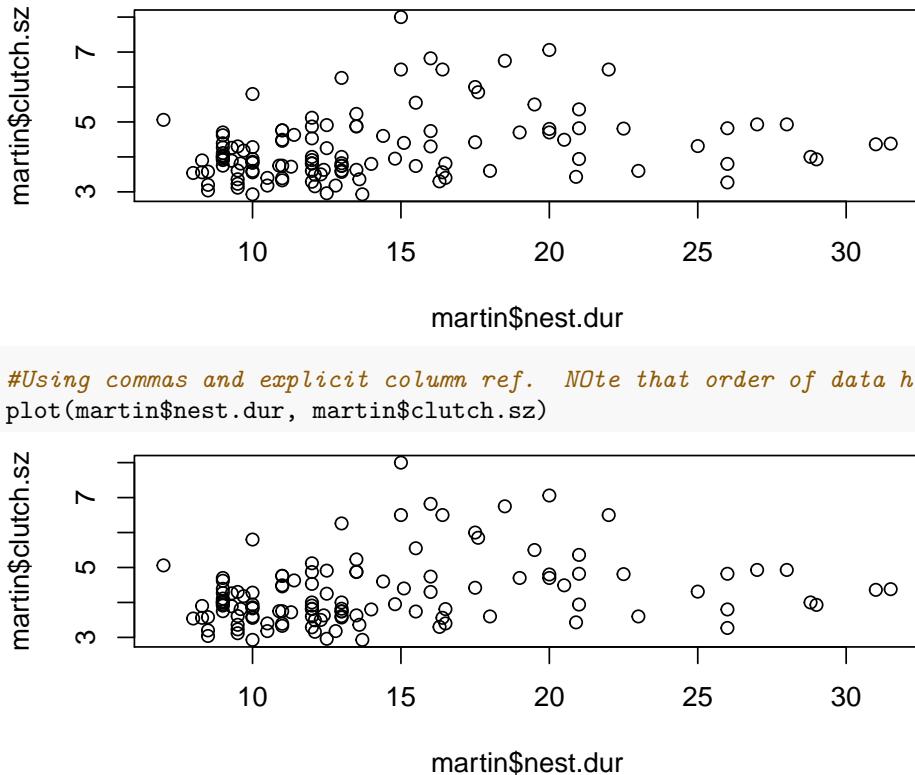
### Main Title: Clutch size vs. nesting duration



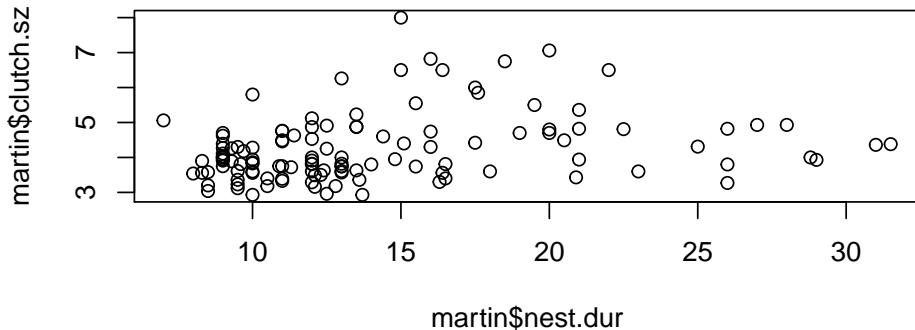
*Details:* Scatter plots are more or less the default plot type of R. There are several minor variation that all do the same thing. The 1st is probably the easiest to remember because it is similar to the standard format for many other R commands, such as linear regression.

```
#Using a tilda ~ and "data ="
plot(clutch.sz ~ nest.dur, data = martin)
```





```
#Using commas and explicit column ref. Note that order of data has been flipped.
plot(martin$nest.dur, martin$clutch.sz)
```



#### 127.3.4 BP4) How do I put a regression line through a scatterplot in R?

Oneliner: `plot(clutch.sz ~ nest.dur, data = martin) abline(a = 3.6, b = 0.045)`

Details

#### 127.3.5 BP5) How do I put 2 plots next to each other in R?

Oneliner: `par(mfrow = c(1,2)) plot(clutch.sz ~ nest.dur, data = martin)
plot(clutch.sz ~ surv.adult, data = martin)`

Details R can relativley easily format plots next to each other, but the function is rather cryptic, involving the `par()` command with “`mfrow =`” in. To plot two figures next to each other use ‘`par(mfrow = c(1,2))`’, which basically says “put two plots in a 1 by 2 grid.” The default setting for R is `par(mfrow = c(1,1))`, which corresponds to a  $1 \times 1$  grid. You can probably guess that to put two plots on top of each other you would use `par(mfrow = c(2,1))`, and so forth.

```
#Two plots next to each other
par(mfrow = c(1,2))
```

```

plot(clutch.sz ~ nest.dur, data = martin)
plot(clutch.sz ~ surv.adult, data = martin)

#Two plots on top of each other
par(mfrow = c(2,1))
plot(clutch.sz ~ nest.dur, data = martin)
plot(clutch.sz ~ surv.adult, data = martin)

```

*Finnese:* Lots of tweaks can be done to plot in general, and to multiple plots in a grid, such as the space between each plots. The code, however, is rather cryptic. If you look at the help menu for the par() command you'll see dozen of options. For quickly making plots look nice, many people just export into Power Point and orient and annotat them there. Learning how to feed commands to par() to make plots and groups of plots look nice can save you time in the long run, but it takes time to figure out what you want and what you like. I find that the R package ggplot2 with its function qplot() makes nicer default plots. GGPLOT has its own complicated syntax, but is preffered by many R users.

### 127.3.6 How do I put two plots next to each other using ggplot?

Oneliner library(ggplot2) qplot(y = pred, x = nest.dur, data = martin)

### 127.3.7 How do I add a regression line to a scatterplot with ggplot?

Oneliner library(ggplot2) qplot(y = pred, x = nest.dur, data = martin, geom = c("smooth", "point"), method = "lm")

Full code qplot(y = pred, x = nest.dur, data = martin, geom = c("smooth", "point"), se = FALSE, method = "lm", xlab = "xlab = nesting duration", ylab = "ylab = predation rate", main = "main title: predation vs. nesting duration")

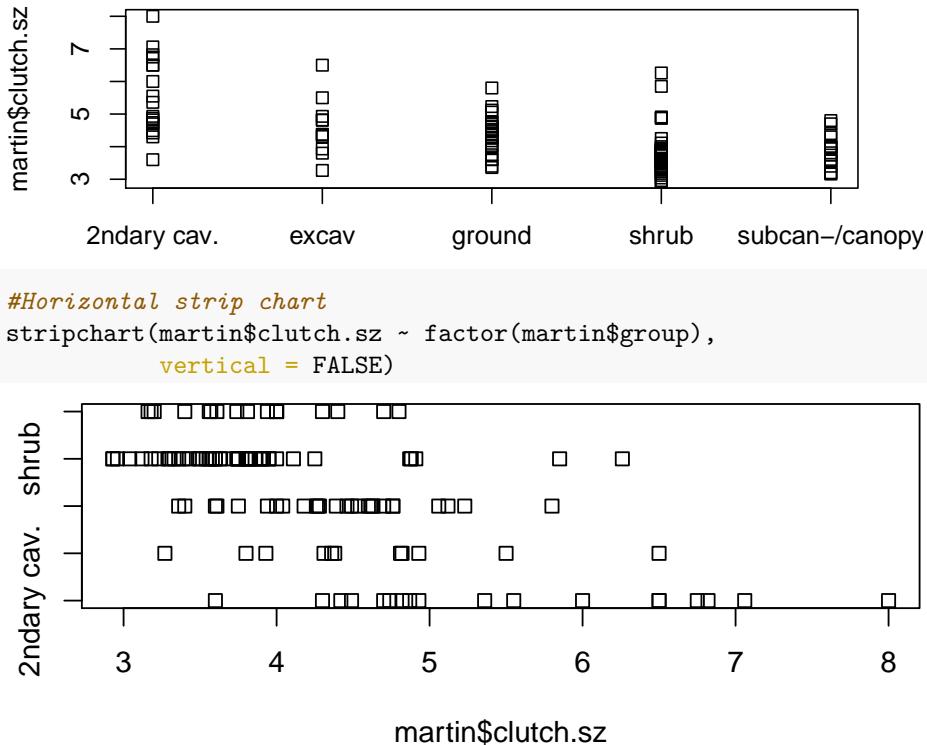
## 127.4 How do I make a strip chart

Box plot are great but don't work well for small data set.

```

#Vertical strip chart
stripchart(martin$clutch.sz ~ factor(martin$group),
 vertical = TRUE)

```



## 127.5 GGPLOT

```
library(ggplot2)
```

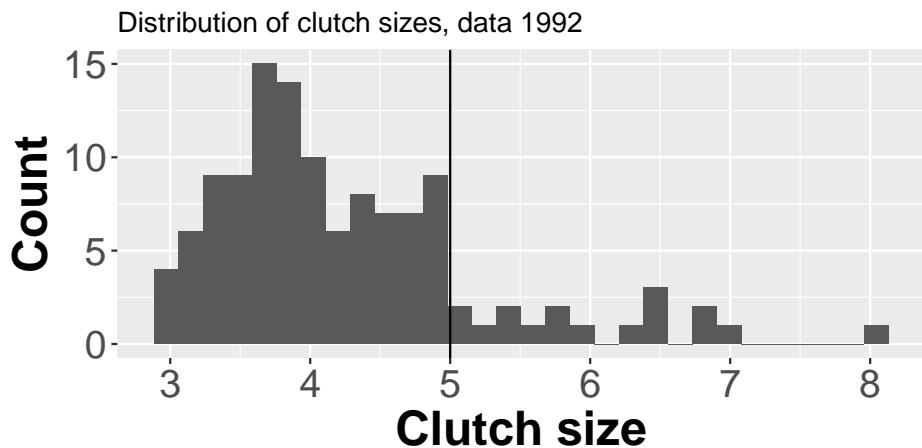
- 127.5.1 Increase the font size for the x and y axes in ggplot?
- 127.5.2 How do I increase the SIZE OF THE TICK LABELS on the x and y axes?
- 127.5.3 How do I increase the size of TITLES or labels on the x and y axes?
- 127.5.4 How do I make these things bold?

Use “theme(axis.text = element\_text(size = ...))” for the axis labels. Use “theme(axis.title = element”

```
library(ggplot2)

qplot(clutch.sz,
 data = martin) +
```

```
geom_vline(xintercept = 5) +
xlab("Clutch size") +
ylab("Count") +
ggtitle("Distribution of clutch sizes, data 1992") +
theme(axis.text=element_text(size=18), #increase font size to 18
 axis.title=element_text(size=22,
 face="bold")) #use bold text
```



### 127.5.5 How do I add horizontal and vertical lines to a ggplot?

```
geom_hline(xintercept = 0) geom_vline(yintercept = 0)
```

### 127.5.6 How do I flip or rotate the coordinates to that the y-axis becomes the x-axis in ggplot?

```
coord_flip()
#> <ggproto object: Class CoordFlip, CoordCartesian, Coord, gg>
#> aspect: function
#> backtransform_range: function
#> clip: on
#> default: FALSE
#> distance: function
#> expand: TRUE
#> is_free: function
#> is_linear: function
#> labels: function
#> limits: list
#> modify_scales: function
#> range: function
```

```
#> render_axis_h: function
#> render_axis_v: function
#> render_bg: function
#> render_fg: function
#> setup_data: function
#> setup_layout: function
#> setup_panel_guides: function
#> setup_panel_params: function
#> setup_params: function
#> train_panel_guides: function
#> transform: function
#> super: <ggproto object: Class CoordFlip, CoordCartesian, Coord, gg>
```

This can be used to make coefficient plots and forests plots.

### 127.5.7 How do I flip the y axis so that positive values are on the opposite side in ggplot?

```
scale_y_reverse()
```

### 127.5.8 How do I manually set the colors for the legend or scale in ggplot?

### 127.5.9 How do I change the colors of the legend or scale in ggplot?

```
scale_colour_manual(name="Experimental\nSettings",
 values=colors_BeS.3,
 labels=c("Greenhouse", "Greenhouse\n& Field"))
```

### 127.5.10 How do I make a confidence band around a line in ggplot?

(regression, linear model, CI, confidence interval)

```
geom_ribbon(aes(ymin=SE.real.minus,
 ymax=SE.real.plus),
 alpha=0.15,
 linetype = 0)
```

**127.5.11** How do i set the limits on the x or y axis in ggplot?

**127.5.12** How do I re-name the x or y axis in ggplot?

The commands `scale_x_continuous()` and `scale_y_continuous()` can do several things. See also `xlab()` and `ylab()`

```
scale_x_continuous(name="Stem Length",limits=c(15,100)) + #Stem Length (cm)
scale_y_continuous(name="Reversion Probability")
```

**127.5.13** How do I make changes to the legend in ggplot?

**127.5.14** How do I change the position of the legend in ggplot?

**127.5.15** How do I change the title over the legend?

**127.5.16** How do I change the font of the *title* of the legend in ggplot?

**127.5.17** How do I change the font of the *labels* of the legend in ggplot?

```
#keywords: ggplot, legend, font, bold,
theme(
 legend.position="right",
 #legend.position=c(0.15, .85),
 legend.title = element_text(colour="black",
 size=24,
 face="bold"),
 legend.text = element_text(colour="black",
 size = 20,
 face = "bold")) +
```

**127.5.18** How do I change the thickness of the axes in ggplot?

```
theme(axis.line = element_line(size=2, color = "black")) #makes axes lines thicker
```

**127.5.19** How do I get rid of/remove/suppress the box around the plotting field

```
keywords: box, border, edging, around blot
theme(panel.border = element_rect(color = "white"))
```

## 127.6 How do I get rid of the box around...

```
keywords: box, border, edging, around blot
theme(legend.key = element_rect(colour = 'white')) + # gets rid of box around length key items
```

## 127.7 How do I get rid of the grid lines within the plot in ggplot?

```
theme(panel.grid.minor=element_blank(),
 panel.grid.major=element_blank())
```

## 127.8 How do I set the font of the axes titles?

## 127.9 How do I adjust the positions of the axes titles?

```
theme(axis.title.x = element_text(face="bold", size=32),
 axis.text.x = element_text(vjust=0.95,
 size=32,
 face="bold"),
 axis.title.y = element_text(face="bold",
 size=32,
 vjust=0.35),
 axis.text.y = element_text(vjust=0.5,
 size=32,face="bold"))
```

## 127.10 How do I drop, remove or suppress the legend in ggplot?

```
theme(legend.position = "none")
```

## 127.11 Remove facet strip completely

<http://stackoverflow.com/questions/10547487/r-removing-facet-wrap-labels-completely-in-ggplot2>

```
theme(strip.background = element_blank(),
 strip.text.x = element_blank())
```

## 127.12 How do I increase the size text of the within facets?

## 127.13 How do I increase the font size of facet labels?

<http://stackoverflow.com/questions/2751065/how-can-i-manipulate-the-strip-text-of-facet-plots-in-ggplot2/2751201#2751201>

```
theme(strip.text.x = element_text(size = 8, colour = "orange", angle = 90))
```

## 127.14 How do I reorder a factor variable for plotting in ggplot?

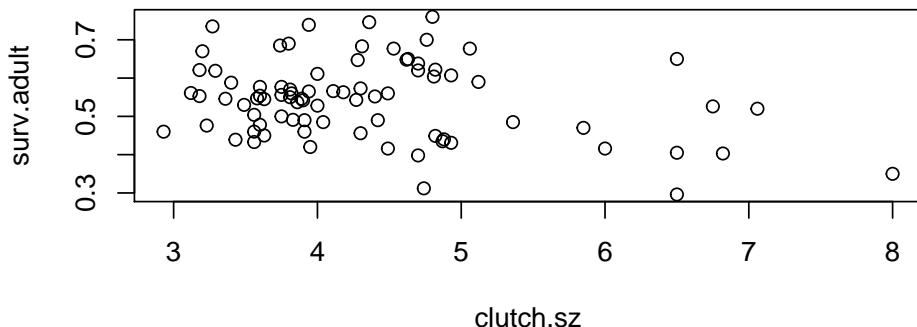
```
levs <- mult.comp.df$names.ordered[order.x]
mult.comp.df$names.ordered <- factor(mult.comp.df$names.ordered,
 levels = levs)
```

## 127.15 Colorblind palletes

color blind pallettes <http://dr-k-lo.blogspot.com/2013/07/a-color-blind-friendly-palette-for-r.html> [http://www.cookbook-r.com/Graphs/Colors\\_%28ggplot2%29/](http://www.cookbook-r.com/Graphs/Colors_%28ggplot2%29/)

Plot clutch size versus duration of incubation using basic R plotting function

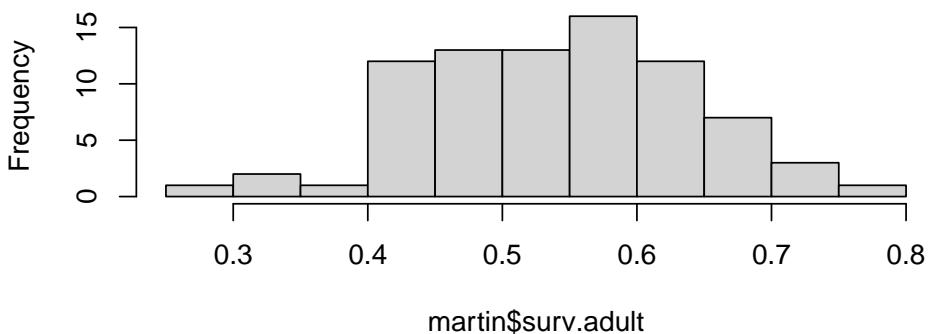
```
plot(surv.adult ~ clutch.sz, data = martin)
```



Plot distribution of survival rates

```
hist(martin$surv.adult)
```

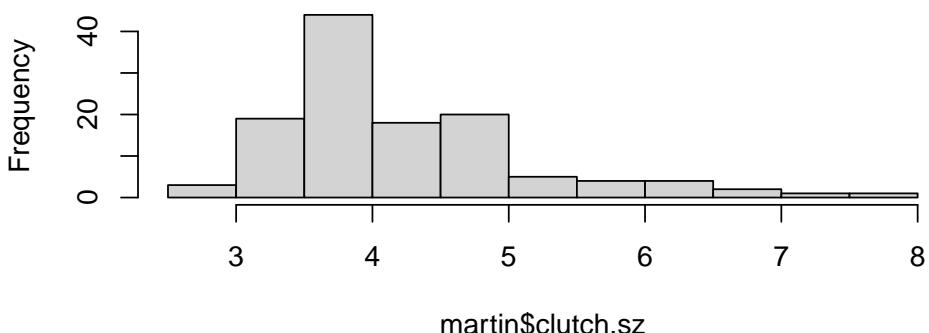
**Histogram of martin\$surv.adult**



Plot distribution of clutch sizes

```
hist(martin$clutch.sz)
```

**Histogram of martin\$clutch.sz**

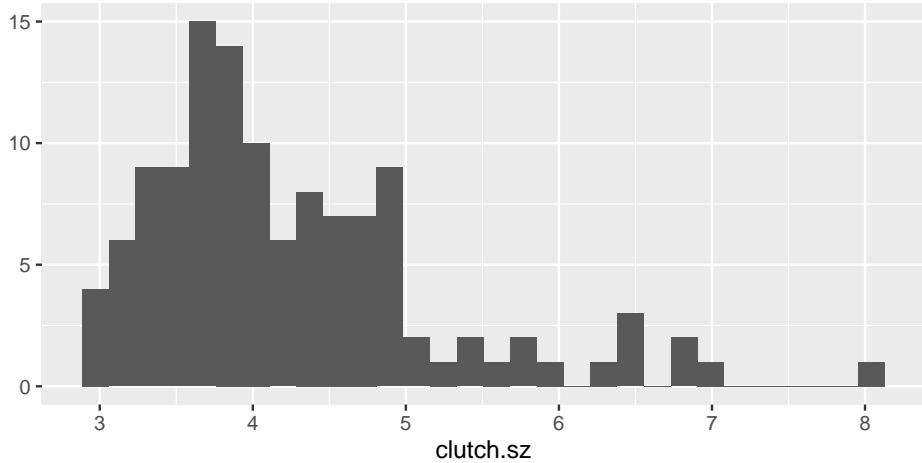


Load ggplot package

```
library(ggplot2)
```

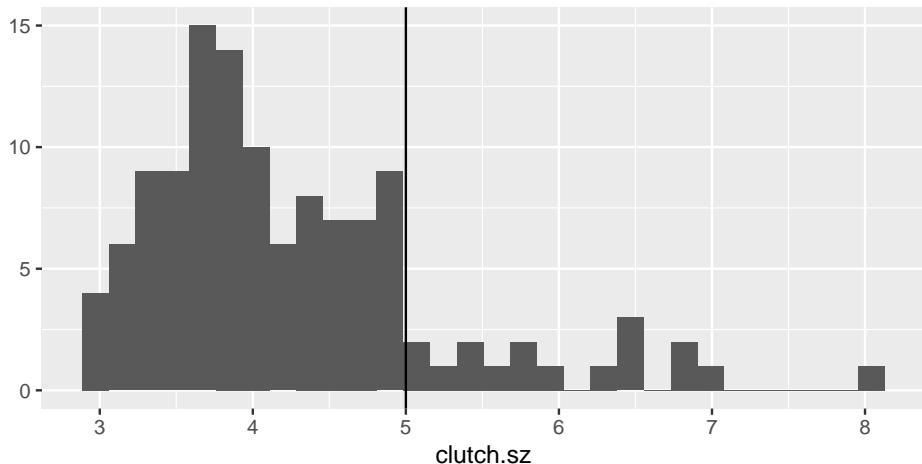
Plot distribution of clutch sizes in ggplot

```
qplot(clutch.sz,
 data = martin)
```



Add line at 6 for LOWA

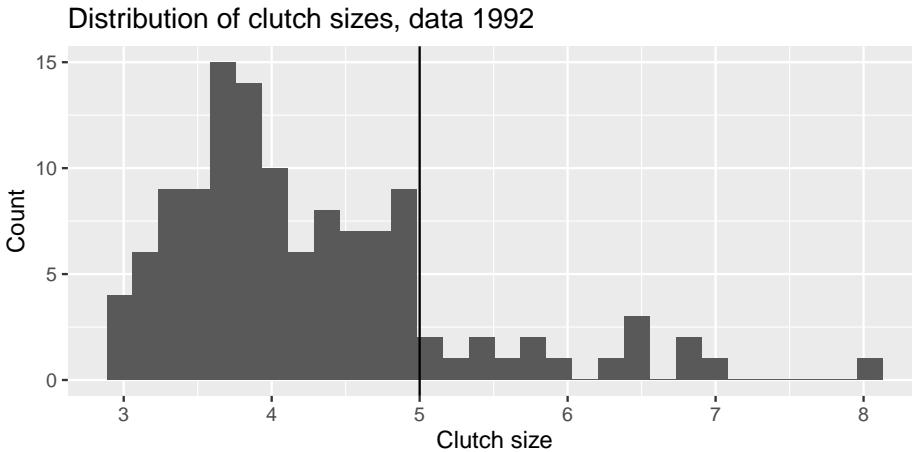
```
qplot(clutch.sz,
 data = martin) + geom_vline(xintercept = 5)
```



Add labels

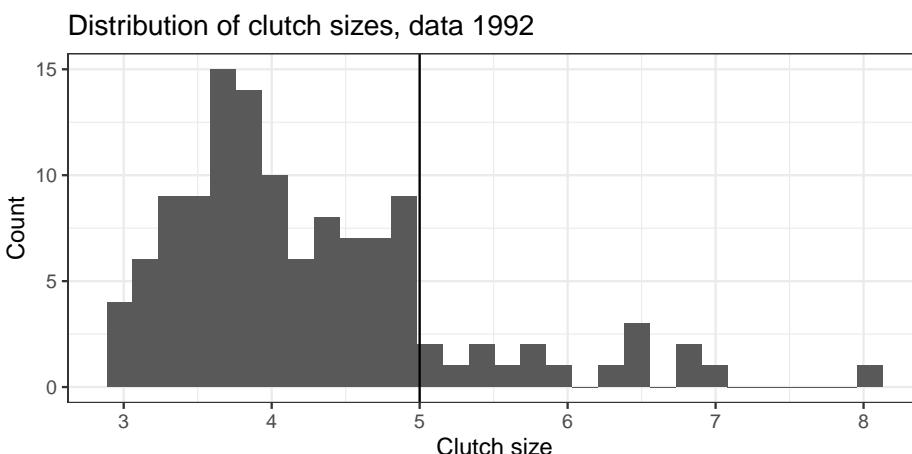
```
qplot(clutch.sz,
 data = martin) +
 geom_vline(xintercept = 5) +
```

```
xlab("Clutch size") +
ylab("Count") +
ggtitle("Distribution of clutch sizes, data 1992")
```



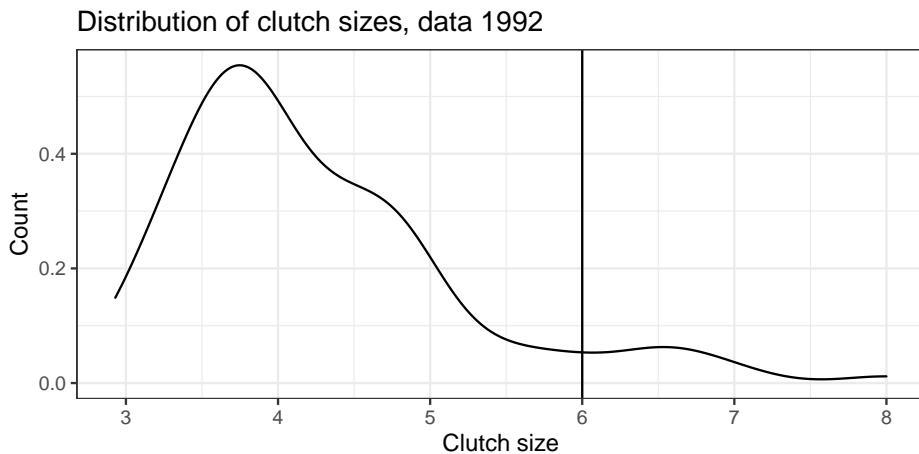
Make background white

```
qplot(clutch.sz,
 data = martin) +
 geom_vline(xintercept = 5) +
 xlab("Clutch size") +
 ylab("Count") +
 ggtitle("Distribution of clutch sizes, data 1992") +
 theme(axis.text=element_text(size=18),
 axis.title=element_text(size=22,face="bold")) +
 theme_bw()
```



Plot as density curve

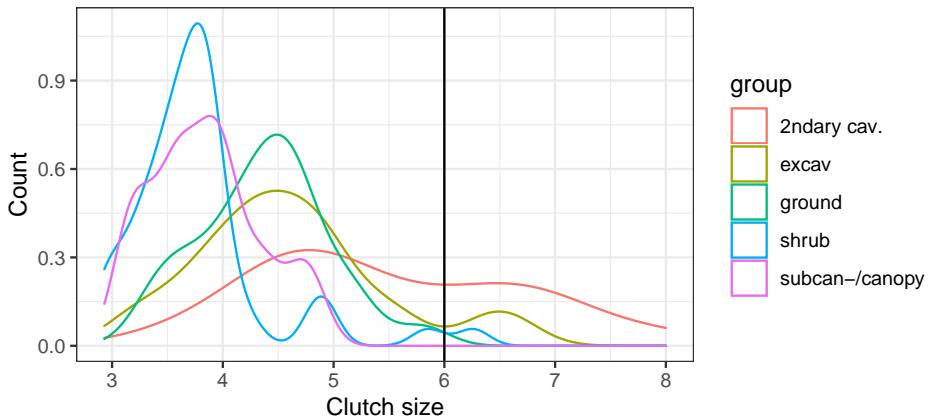
```
qplot(clutch.sz,
 data = martin,
 geom = "density") +
 geom_vline(xintercept = 6) +
 xlab("Clutch size") +
 ylab("Count") +
 ggtitle("Distribution of clutch sizes, data 1992") +
 theme(axis.text=element_text(size=18),
 axis.title=element_text(size=22,face="bold")) +
 theme_bw()
```



Plot as separate density curve for each nest type group

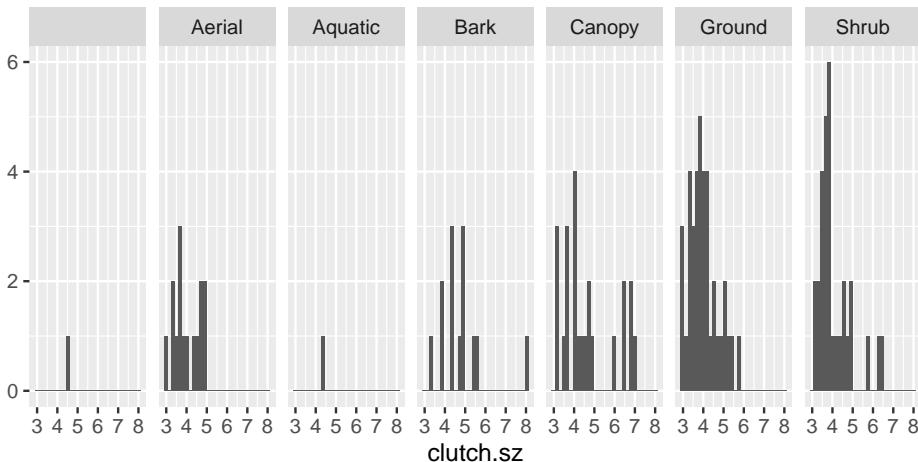
```
qplot(clutch.sz,
 data = martin,
 geom = "density",
 color = group,
 group = group) +
 geom_vline(xintercept = 6) +
 xlab("Clutch size") +
 ylab("Count") +
 ggtitle("Distribution of clutch sizes, data 1992") +
 theme(axis.text=element_text(size=18),
 axis.title=element_text(size=22,face="bold")) +
 theme_bw()
```

Distribution of clutch sizes, data 1992



Plot distribution of clutch sizes by FORAGING site

```
qplot(clutch.sz,
 data = martin,
 facets = . ~ foraging.site)
```



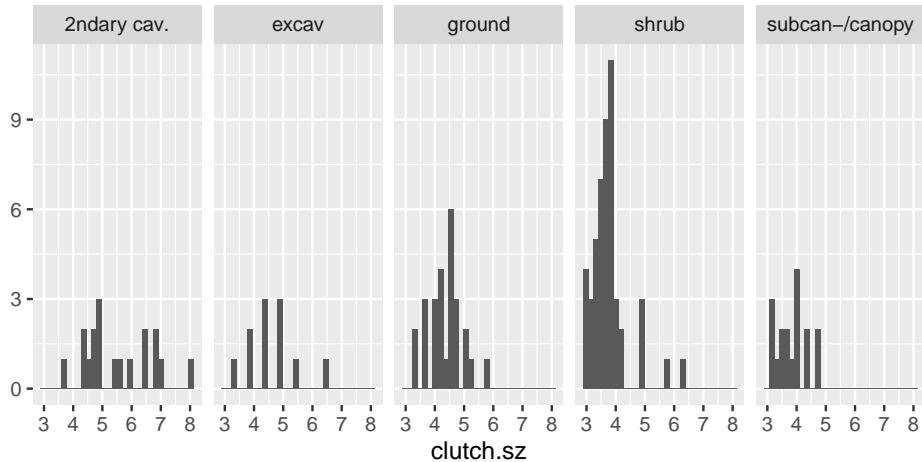
Use facet wrap

```
qplot(clutch.sz,
 data = martin) +
 facet_wrap(~foraging.site)
```



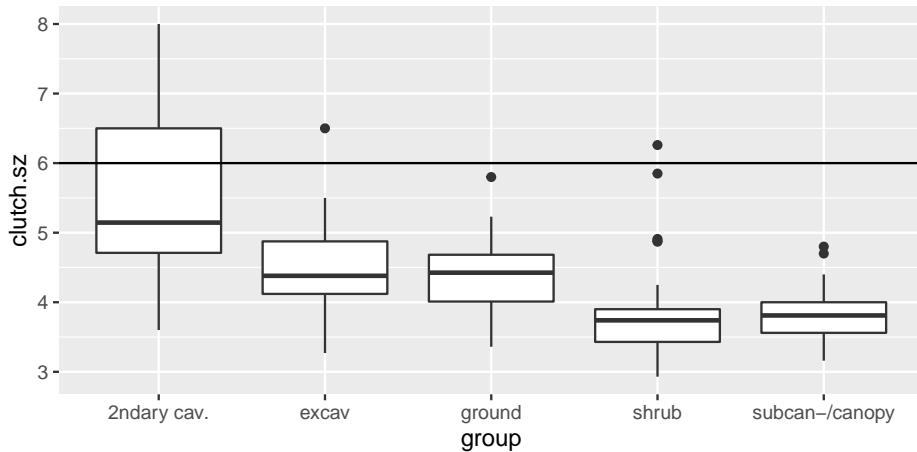
Plot distribution of clutch sizes by habitat group

```
qplot(clutch.sz,
 data = martin,
 facets = . ~ group)
```



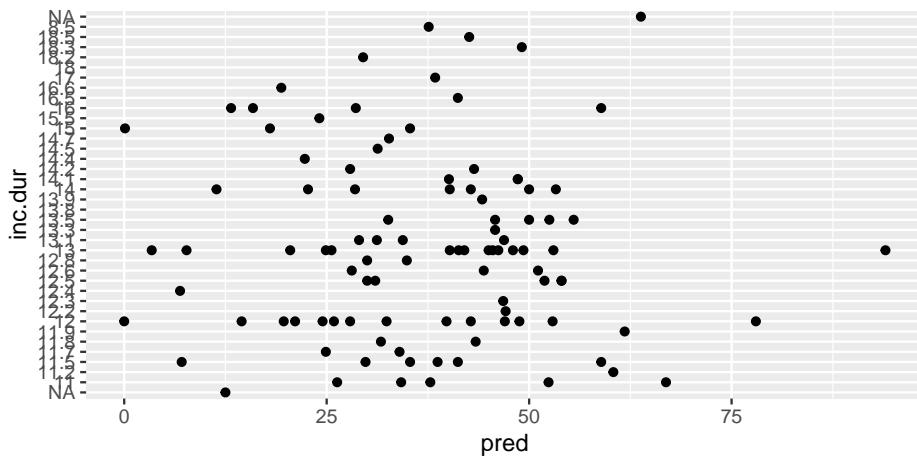
Boxplot

```
qplot(y = clutch.sz,
 x = group,
 geom = "boxplot",
 data = martin) +
 geom_hline(yintercept = 6)
```



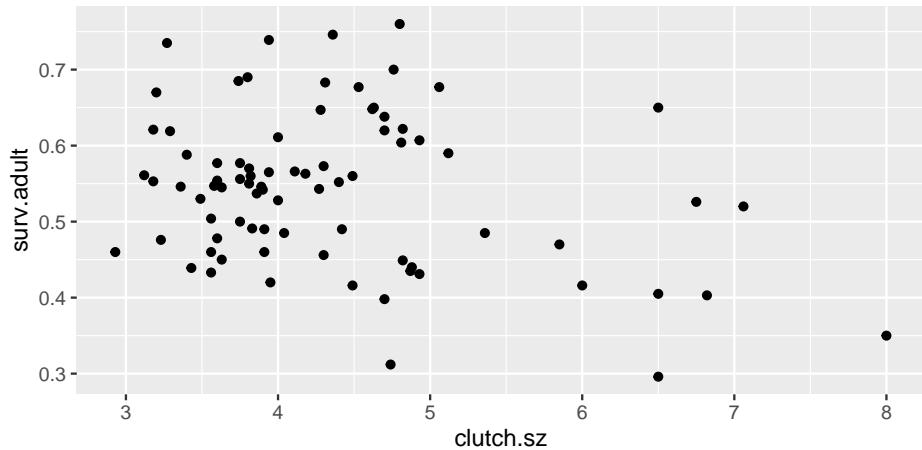
Incubation period vs. predation

```
qplot(y = inc.dur,
 x = pred,
 data = martin)
```



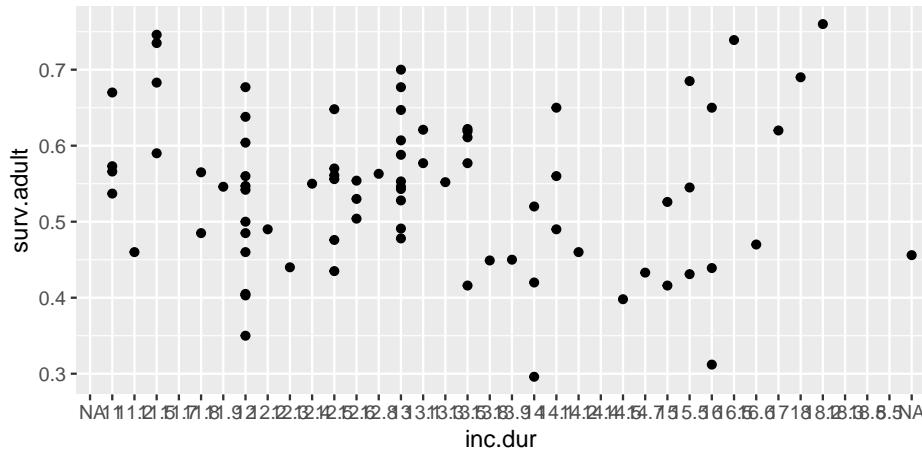
Plot adult survival versus duration of incubation in ggplot

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin)
```



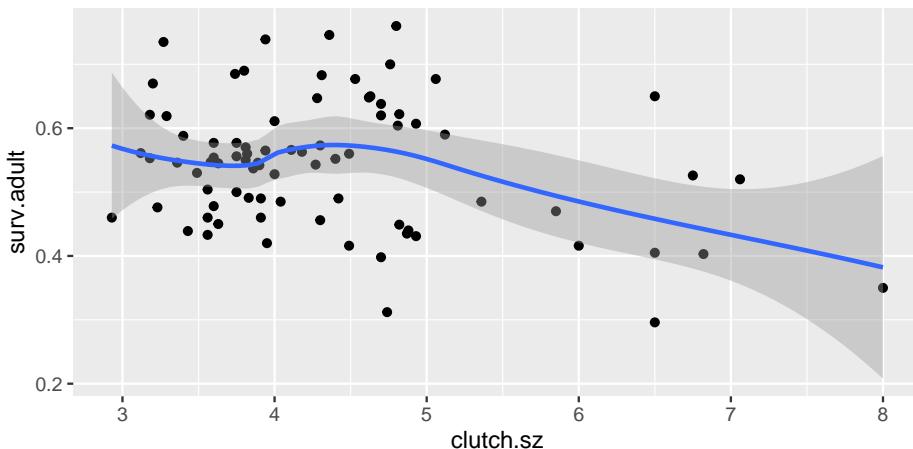
Plot adult survival versus duration of incubation in ggplot

```
qplot(y = surv.adult,
 x = inc.dur,
 data = martin)
```



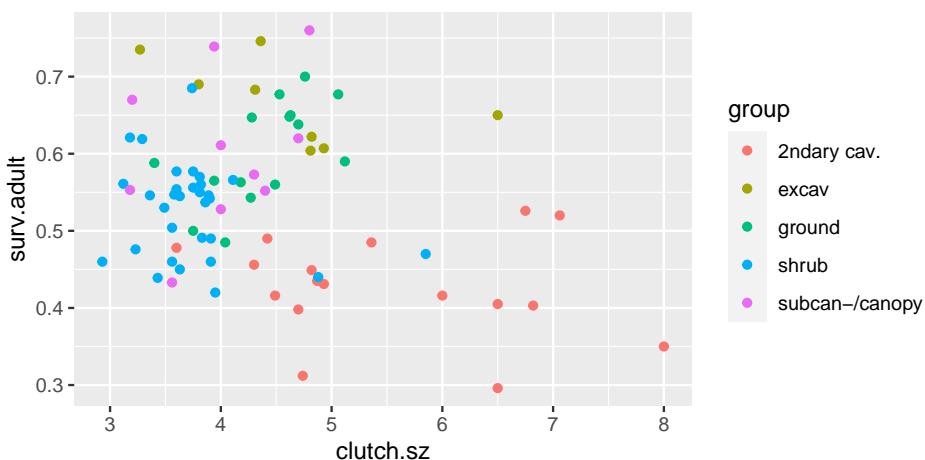
Add a trend line

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 geom = c("point", "smooth"))
```



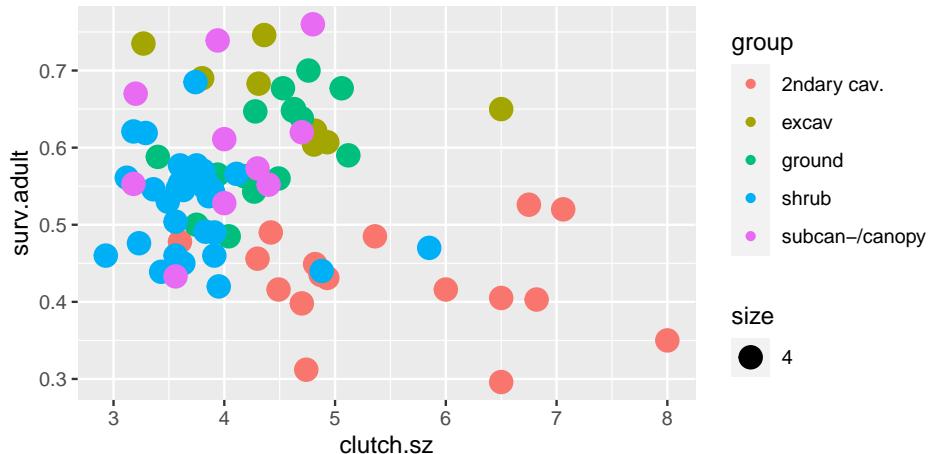
Color code each group

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 color = group)
```



Make size of points bigger

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 color = group,
 size = 4)
```



Add trend line to each group

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 color = group,
 geom = c("point", "smooth"),
 method = "lm",
 se = FALSE,
 size = 4)
```

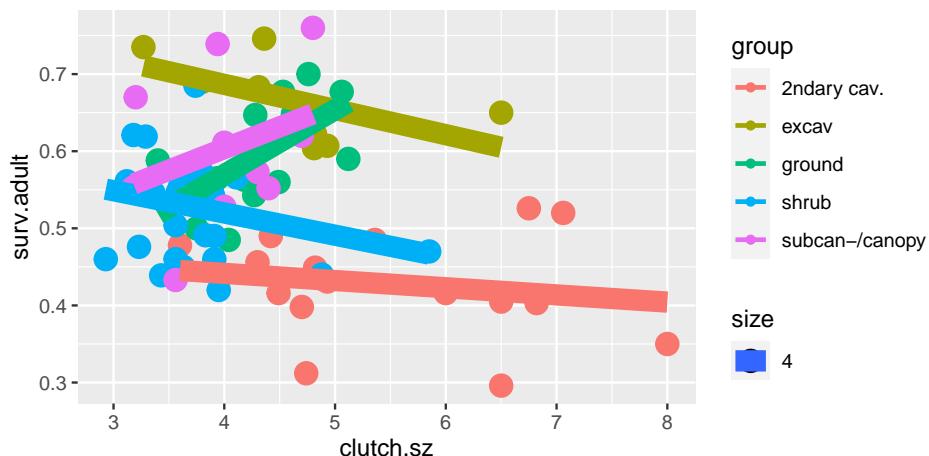
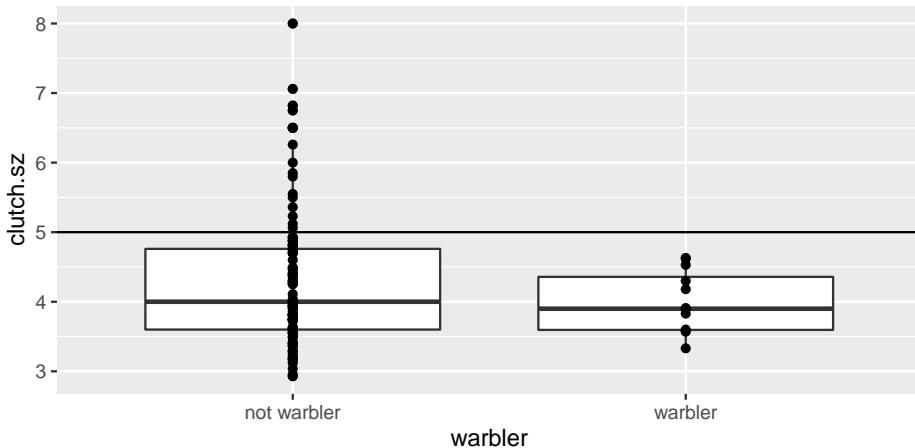


Figure out which species have “warbler” in their name

```
martin$warbler <- "not warbler"
martin$warbler[grep("Warb", martin$spp)] <- "warbler"

qplot(y = clutch.sz,
 x = warbler,
```

```
geom = c("boxplot", "point"),
data = martin) + geom_hline(yintercept = 5)
```



### 127.15.1 OLD / Alt

*Things to cover*

boxplot - base, qplot, ggplot boxplot ordered histogram scatterplot scatterplot w/vert/hort/1:1 scatterplot with regression lines scatterplot with spline smoother

scatterplot matrix

### 127.15.2 Load Martin 1992 data

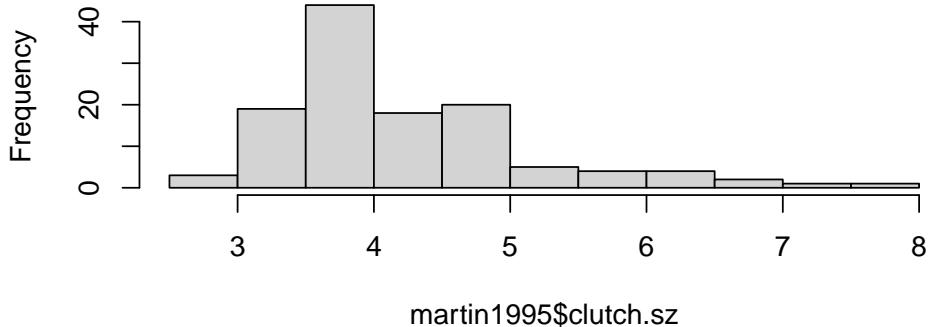
## 127.16 R Plotting FAQ

### 127.16.1 BP1) How do I make histograms in R?

The basics: R makes histograms in a snap in R, something that I do not know of an efficient way to do in Excel. Let's make a histogram of the different clutch sizes (number of eggs laid per nest) for the different species. Since I've forgotten exactly what I called this column I'll first figure that out using the names() function, then make a histogram with hist(). R automatically "bins" the data and plots the frequency of observations in each bin along the y-axis.

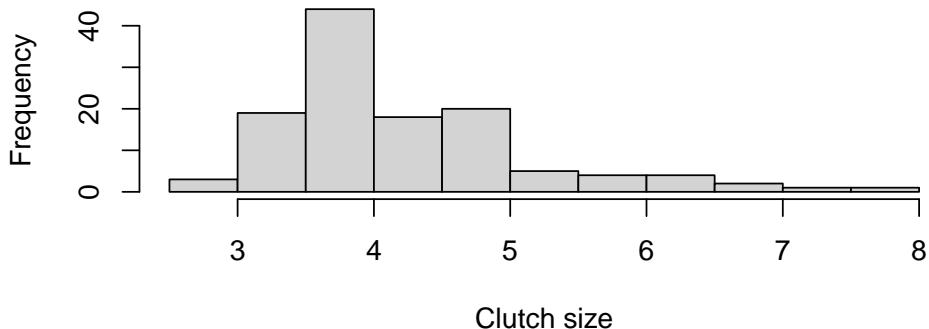
```
#Look at the names in the martin1995 dataframe
names(martin1995)
#> [1] "i.bird" "group" "spp" "clutch.sz"
#> [5] "inc.dur" "nest.dur" "nest.suc" "pred"
#> [9] "broods" "surv.adult" "lat" "foraging.site"
#> [13] "refs"
```

```
#Make a histogram of clutch size
hist(martin1995$clutch.sz)
```

**Histogram of martin1995\$clutch.sz**

Finesse: Changing titles R automatically assigns names to everything. Here I change the x-axis (called the x label) and title, which is called the “main title”.

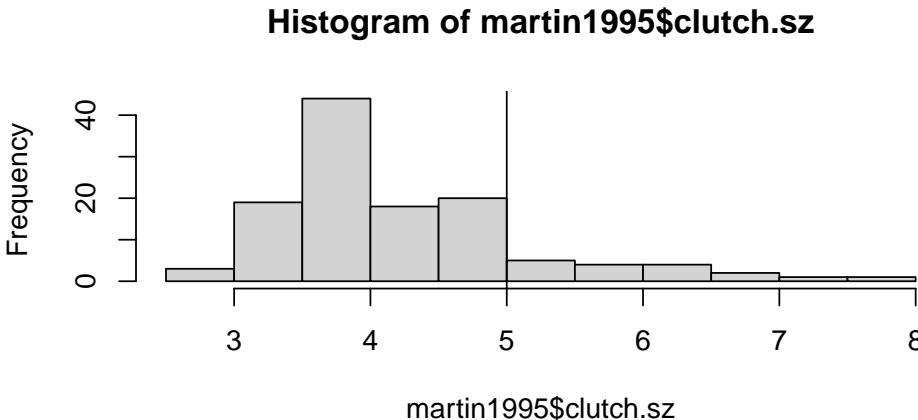
```
hist(martin1995$clutch.sz, xlab = "Clutch size", main = "data's clutch-size data")
```

**data's clutch-size data**

Finesse: Adding a vertical line The bird I study typically lays 5 eggs. To make it easy to compare my bird to other birds, I can add a vertical line at 5. This requires two separate commands. First I'll make the plot, then add the line. The line is made with the abline() command, which stands for “A-B line”, where A stands for the intercept of a line B stands for its slope. abline() can also make horizontal lines and vertical lines, but the programmers must have thought that abvline() was too long of a name.

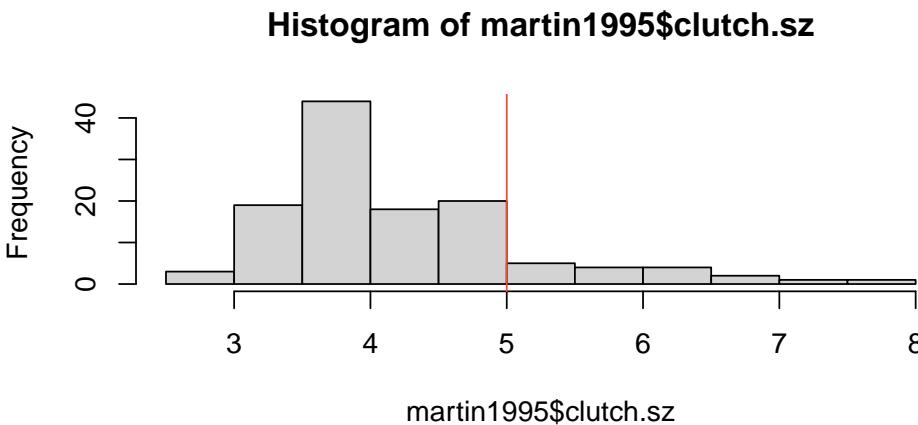
```
#Make histogram
hist(martin1995$clutch.sz)
```

```
#Add vertical line at 5
abline(v = 5)
```



The color is black so it blends in. I can change the color using the col= command within abline line

```
#Make histogram
hist(martin1995$clutch.sz)
abline(v = 5, col = 2)
```

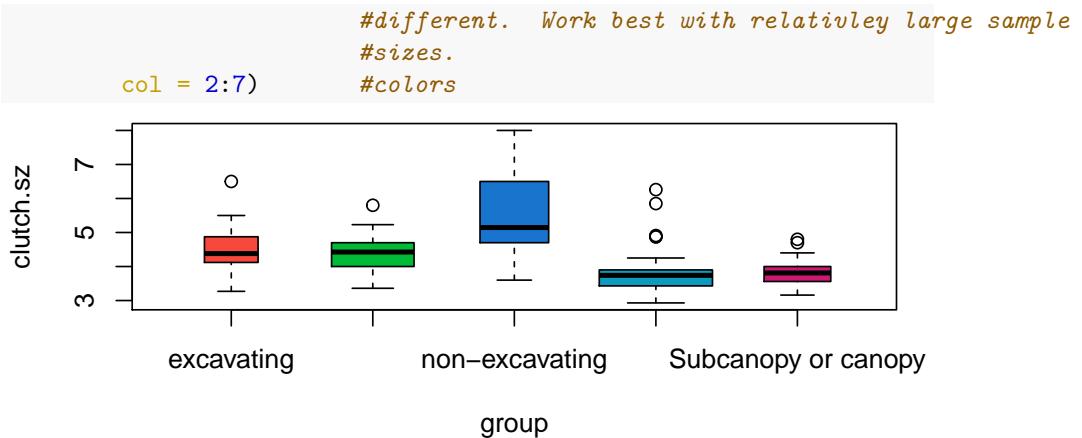


## 127.16.2 BP2) How do I make boxplot in R?

Oneliner: boxplot(martin1995\$clutch.sz ~ martin1995\$group)

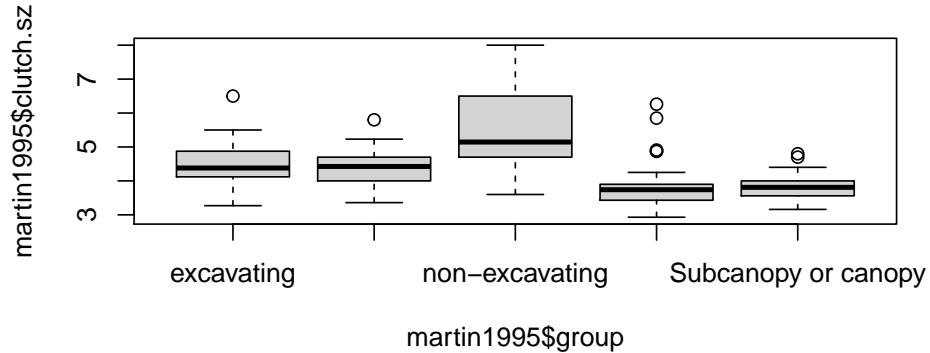
Full code

```
par(mfrow = c(1,1))
boxplot(clutch.sz ~ group, data = martin1995,
 varwidth = TRUE, #modify the width proportional to sample size
 notch = FALSE, #notches can be used to judge if medians are
```



Details: Boxplots are recommended by many statisticians as an excellent way to summarize the distribution of data. Boxplots also a breeze in R, while they are more or less impossible in Excel (I've seen instructions for them but it looks like a pain). Let's make a box plot of clutch size compared to birds from different nesting locations. The boxplot() function uses a tilda, ~ in it. Like many R functions, such as those for regression, the continuous variable goes to the left of ~ and the categorical variables go to the right.

```
boxplot(martin1995$clutch.sz ~ martin1995$group)
```



### 127.16.3 BP3) How do I make scatterplot in R?

*Oneliner:* `plot(clutch.sz ~ nest.dur, data = martin1995)`

*Full code:*

```
#Plot
plot(clutch.sz ~ nest.dur, data = martin1995,
 main = "Main Title: Clutch size vs. nesting duration",
 sub = "Subtitle: Martin 199x Ecological Monographs",
 xlab = "xlab: nesting duration",
```

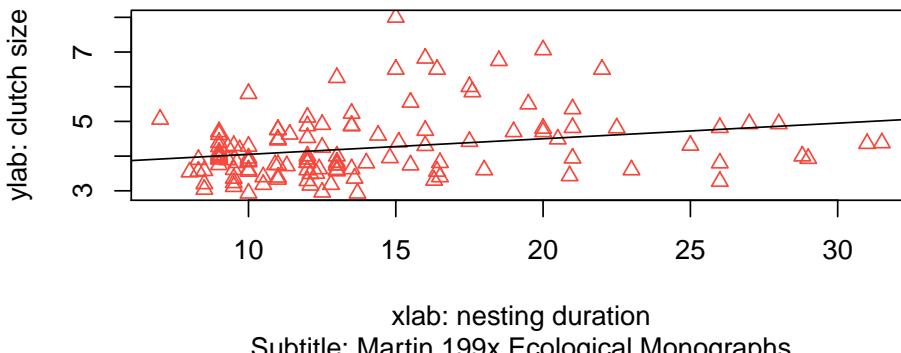
```

ylab = "ylab: clutch size",
col = 2, #change the color
pch = 2) #change th shape of the symbol

#Regression line
abline(a = 3.6, #intercept
 b = 0.045) #slope

```

### Main Title: Clutch size vs. nesting duration



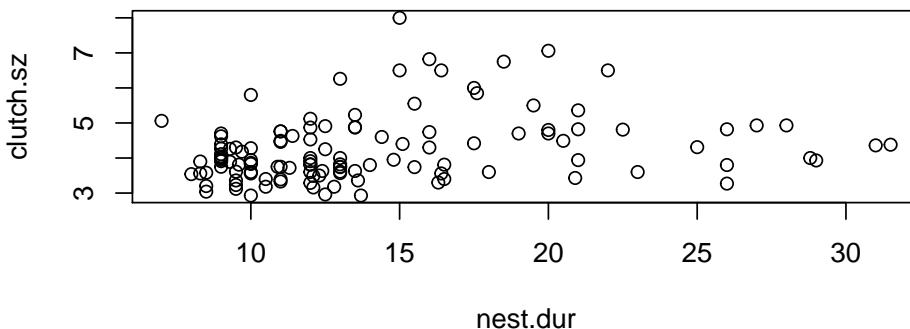
xlab: nesting duration  
Subtitle: Martin 199x Ecological Monographs

*Details:* Scatter plots are more or less the default plot type of R. There are several minor variation that all do the same thing. The 1st is probably the easiest to remember because it is similar to the standard format for many other R commands, such as linear regression.

```

#Using a tilda ~ and "data ="
plot(clutch.sz ~ nest.dur, data = martin1995)

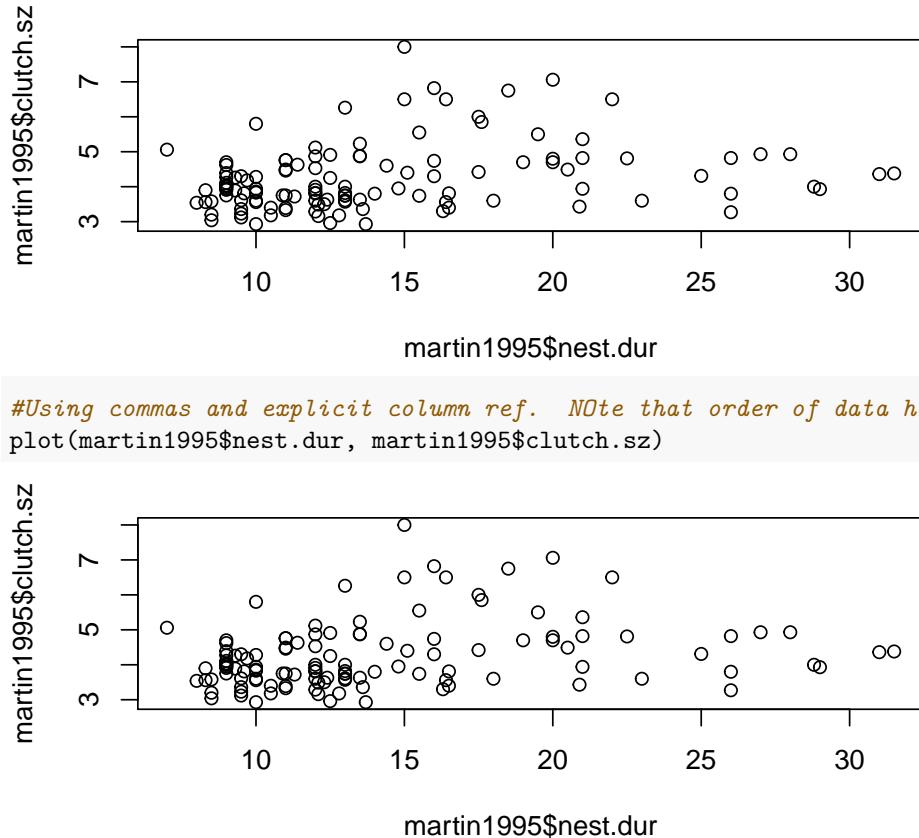
```



```

#Using tilda and explicit column references
plot(martin1995$clutch.sz ~ martin1995$nest.duration)

```



#### 127.16.4 BP4) How do I put a regression line through a scatterplot in R?

Oneliner: `plot(clutch.sz ~ nest.dur, data = martin1995) abline(a = 3.6, b = 0.045)`

Details

#### 127.16.5 BP5) How do I put 2 plots next to each other in R?

Oneliner: `par(mfrow = c(1,2)) plot(clutch.sz ~ nest.dur, data = martin1995)
plot(clutch.sz ~ surv.adult, data = martin1995)`

Details R can relativley easily format plots next to each other, but the function is rather cryptic, involving the `par()` command with “`mfrow =`” in. To plot two figures next to each other use ‘`par(mfrow = c(1,2))`’, which basically says “put two plots in a 1 by 2 grid.” The default setting for R is `par(mfrow = c(1,1))`, which corresponds to a  $1 \times 1$  grid. You can probably guess that to put two plots

on top of each other you would use `par(mfrow = c(2,1))`, and so forth.

```
#Two plots next to each other
par(mfrow = c(1,2))
plot(clutch.sz ~ nest.dur, data = martin1995)
plot(clutch.sz ~ surv.adult, data = martin1995)

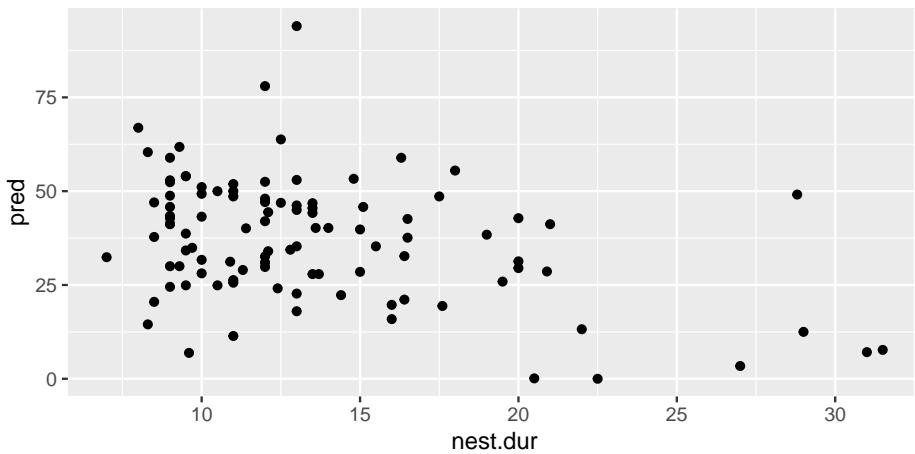
#Two plots on top of each other
par(mfrow = c(2,1))
plot(clutch.sz ~ nest.dur, data = martin1995)
plot(clutch.sz ~ surv.adult, data = martin1995)
```

*Finnese:* Lots of tweaks can be done to plot in general, and to multiple plots in a grid, such as the space between each plots. The code, however, is rather cryptic. If you look at the help menu for the `par()` command you'll see dozen of options. For quickly making plots look nice, many people just export into Power Point and orient and annotat them there. Learning how to feed commands to `par()` to make plots and groups of plots look nice can save you time in the long run, but it takes time to figure out what you want and what you like. I find that the R package `ggplot2` with its function `qplot()` makes nicer default plots. `GGPLOT` has its own complicated syntax, but is prefferred by many R users.

### 127.16.6 How do I put two plots next to each other using ggplot?

Oneliner

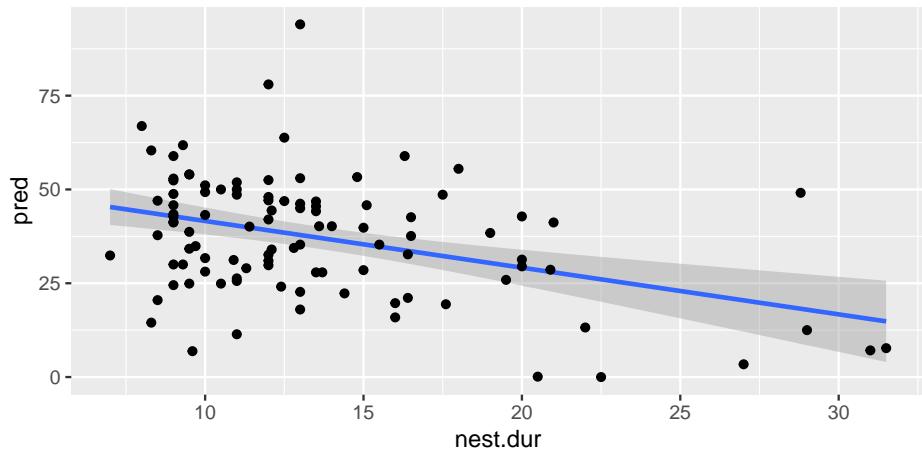
```
library(ggplot2)
qplot(y = pred, x = nest.dur, data = martin1995)
```



### 127.16.7 How do I add a regression line to a scatterplot with ggplot?

Oneliner

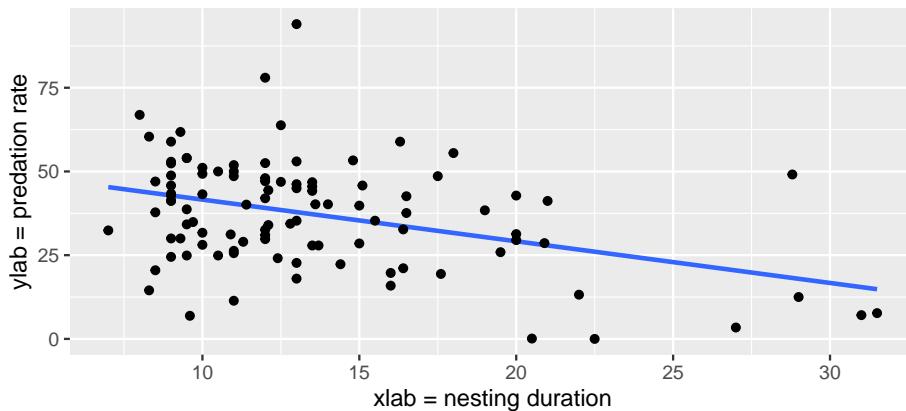
```
library(ggplot2)
qplot(y = pred, x = nest.dur, data = martin1995,
 geom = c("smooth", "point"), method = "lm")
```



Full code

```
qplot(y = pred, x = nest.dur, data = martin1995,
 geom = c("smooth", "point"),
 se = FALSE,
 method = "lm",
 xlab = "xlab = nesting duration",
 ylab = "ylab = predation rate",
 main = "main title: predation vs. nesting duration")
```

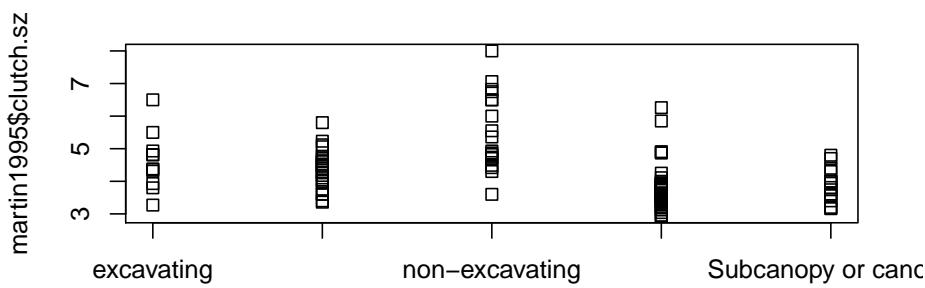
main title: predation vs. nesting duration



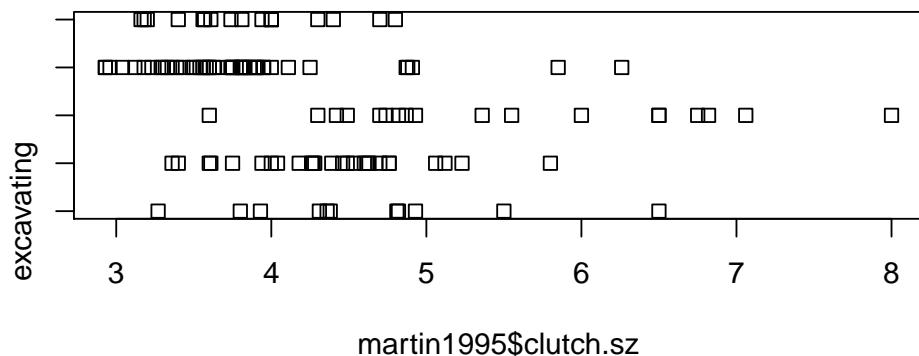
## 127.17 How do I make a strip chart with base R

Box plot are great but don't work well for small data set.

```
#Vertical strip chart
stripchart(martin1995$clutch.sz ~ factor(martin1995$group),
 vertical = TRUE)
```

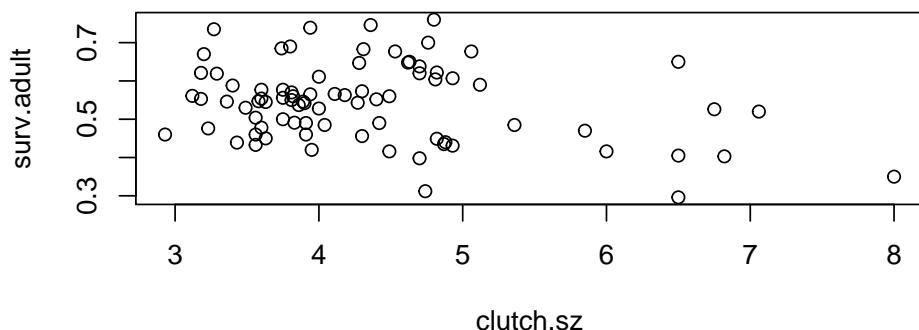


```
#Horizontal strip chart
stripchart(martin1995$clutch.sz ~ factor(martin1995$group),
 vertical = FALSE)
```

**127.18**

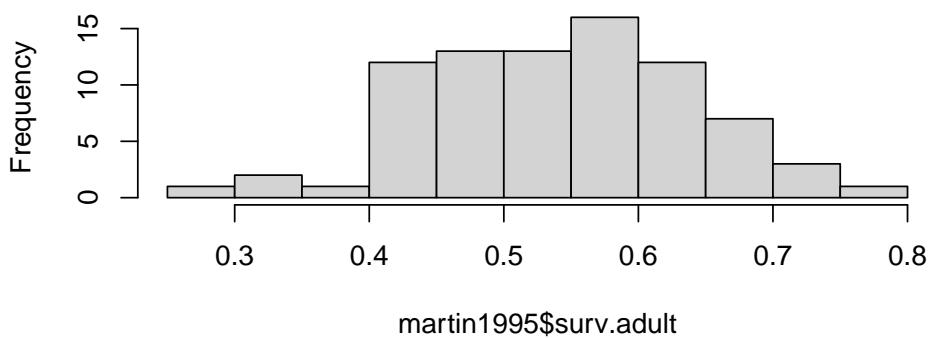
Plot clutch size versus duration of incubation using basic R plotting function

```
plot(surv.adult ~ clutch.sz, data = martin1995)
```



Plot distribution of survival rates

```
hist(martin1995$surv.adult)
```

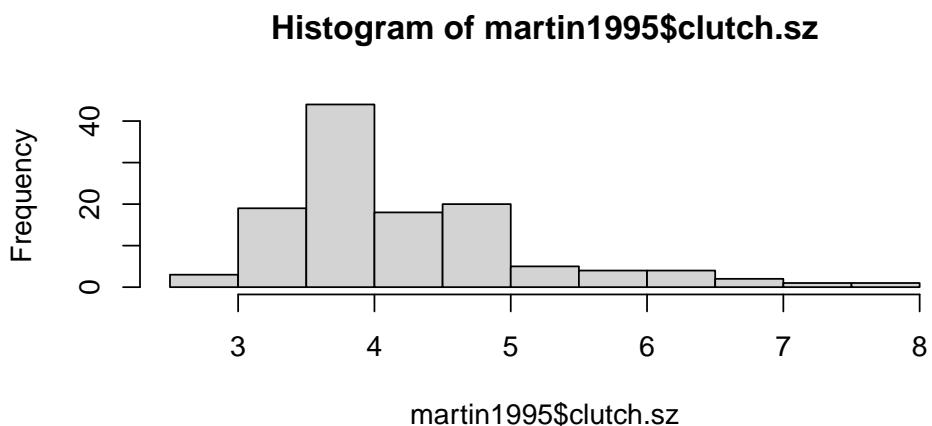
**Histogram of martin1995\$surv.adult**

127.18.

697

Plot distribution of clutch sizes

```
hist(martin1995$clutch.sz)
```





## Chapter 128

# Graphical parameters in R with par(): With great power comes great responsibility

```
library(compbio4all)
```

### 128.1 Save plotting defaults

```
par_defaults <- par()
```

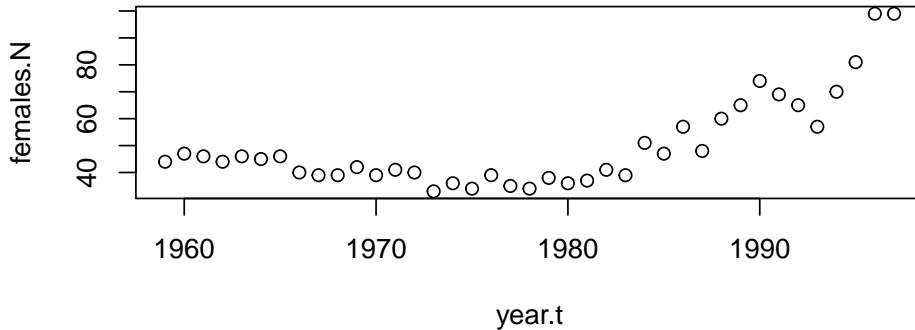
### 128.2 Data

```
census <- 1:39
year.t <- 1959:1997
females.N <- c(44,47,46,44,46,
 45,46,40,39,39,
 42,39,41,40,33,
 36,34,39,35,34,
 38,36,37,41,39,
 51,47,57,48,60,
 65,74,69,65,57,
 70,81,99,99)
lambda.i <- females.N[-1]/females.N[-length(females.N)]
```

```
lambda.i <- c(lambda.i,NA)
lambda_log <- log(lambda.i)
bear_N <- data.frame(census,
 year.t,
 females.N,
 lambda.i,
 lambda_log)
```

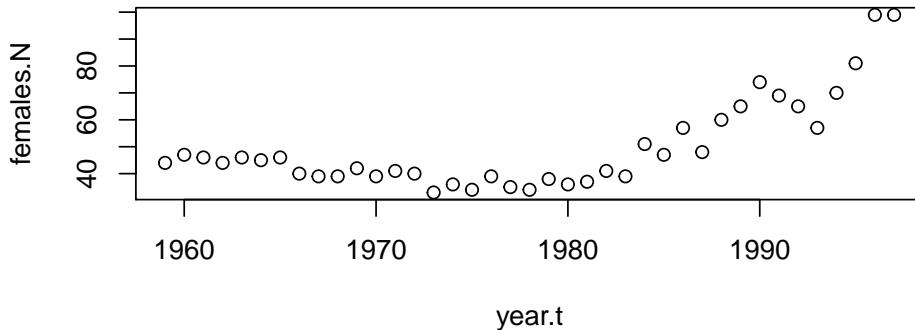
### 128.3 A single-panel plot - default

```
plot(females.N ~ year.t, data = bear_N)
```



### 128.4 A single-panel plot - explicitly with par()

```
par(mfrow = c(1,1))
plot(females.N ~ year.t, data = bear_N)
```



### 128.5 A vertical 2-panel plot - with par()

TODO doesn't like to knit

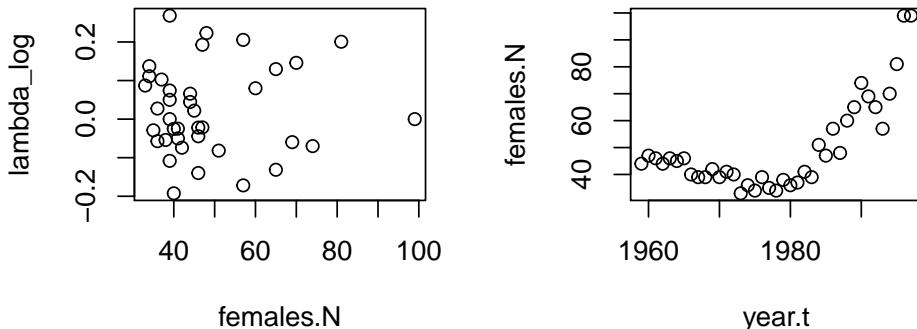
```
par(mfrow = c(2,1))

plot(lambda_log ~ females.N, data = bear_N)
plot(females.N ~ year.t, data = bear_N)
```

## 128.6 A horizontal 2-panel plot - with par()

```
par(mfrow = c(1,2))

plot(lambda_log ~ females.N, data = bear_N)
plot(females.N ~ year.t, data = bear_N)
```



## 128.7 A 4-panel plot - with par()

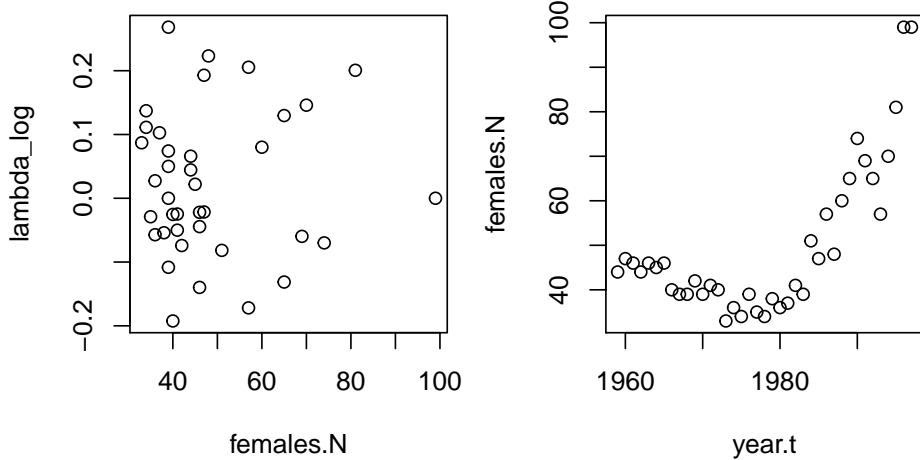
```
par(mfrow = c(2,2))

plot(lambda_log ~ females.N, data = bear_N)
plot(females.N ~ year.t, data = bear_N)
```

## 128.8 A horizontal 2-panel plot - with par() and adjusted margins

```
par(mfrow = c(1,2), mar = c(4,4,1,1))

plot(lambda_log ~ females.N, data = bear_N)
plot(females.N ~ year.t, data = bear_N)
```



## 128.9 Reset defaults

This fails - why

```
par(par_defaults)
```

*Things to cover* boxplot - base, qplot, ggplot boxplot ordered histogram scatterplot scatterplot w/vert/hort/1:1 scatterplot with regression lines scatterplot with spline smoother

scatterplot matrix

## 128.10 Reference

Martin, TE. 1995. Avian Life History Evolution in Relation to Nest Sites, Nest Predation, and Food. Ecological Monographs. <http://onlinelibrary.wiley.com/doi/10.2307/2937160/full>

**Abstract:** “Food limitation is generally thought to underlie much of the variation in life history traits of birds. I examined variation and covariation of life history traits of 123 North American Passeriformes and Piciformes in relation to nest sites, nest predation, and foraging sites to examine the possible roles of these ecological factors in life history evolution of birds. Annual fecundity was strongly inversely related to adult survival, even when phylogenetic effects were controlled. Only a little of the variation in fecundity and survival was related to foraging sites, whereas these traits varied strongly among nest sites. Interspecific differences in nest predation were correlated with much of the variation in life history traits among nest sites, although energy trade-offs with covarying traits also may account for some variation. For example, increased nest predation is associated with a shortened nestling period and both are associated with more broods per year, but number of broods is inversely correlated with clutch

## 128.11. MAKE SOME MORE CHANGES TO SHORTEN THE GROUP NAMES<sup>703</sup>

size, possibly due to an energy trade-off. Number of broods was much more strongly correlated with annual fecundity and adult survival among species than was clutch size, suggesting that clutch size may not be the primary fecundity trait on which selection is acting. Ultimately, food limitation may cause trade-offs between annual fecundity and adult survival, but differences among species in fecundity and adult survival may not be explained by differences in food abundance and instead represent differing tactics for partitioning similar levels of food limitation. Variation in fecundity and adult survival is more clearly organized by nest sites and more closely correlated with nest predation; species that use nest sites with greater nest predation have shorter nestling periods and more broods, yielding higher fecundity, which in turn is associated with reduced adult survival. Fecundity also varied with migratory tendencies; short-distance migrants had more broods and greater fecundity than did neotropical migrants and residents using similar nest sites. However, migratory tendencies and habitat use were confounded, making separation of these two effects difficult. Nonetheless, the conventional view that neotropical migrants have fewer broods than residents was not supported when nest site effects were controlled.”

```
##Load Martin 1995 data
library(compbio4all)
data(martin1995)
martin <- martin1995

library(ggplot2)
```

Change the “non-excavating” group to “secondary cavity”

```
martin$group <- gsub("non-excavating","2ndary cav.",martin$group)
```

Change the “group-nesting” group to “ground”

```
martin$group <- gsub("group-nesting","ground",martin$group)
```

## 128.11 Make some more changes to shorten the group names

```
martin$group <- gsub("Subcanopy or canopy","subcan-/canopy",martin$group)

martin$group <- gsub("shrub or low foliage","shrub",martin$group)

martin$group <- gsub("excavating","excav",martin$group)
```

Classify general latitudinal region as tropical, boreal, or temperate

```

martin$trop.or.temp <- "temperate"
i.tropics <- which(martin$lat < 23)
i.boreal <- which(martin$lat > 54)

martin$trop.or.temp[i.tropics] <- "tropics"
martin$trop.or.temp[i.boreal] <- "boreal"

```

### 128.11.1 How do I put two plots next to each other using ggplot?

Oneliner

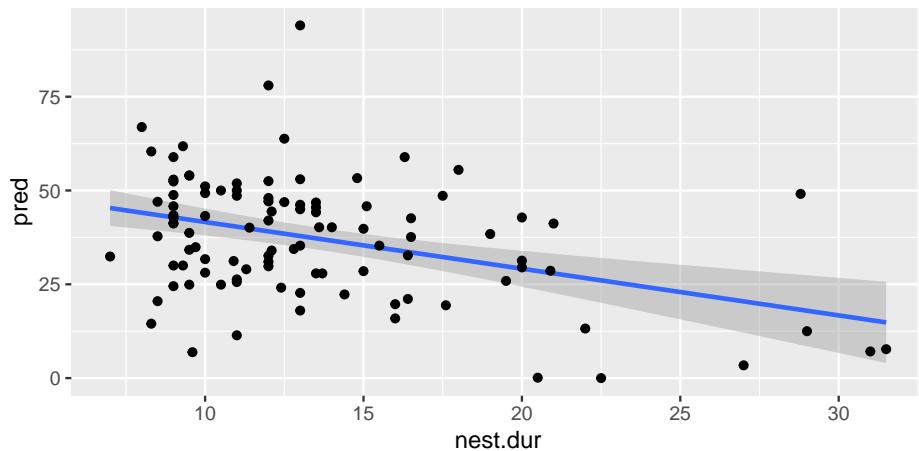
### 128.11.2 How do I add a regression line to a scatterplot with ggplot?

Oneliner

```

qplot(y = pred,
 x = nest.dur,
 data = martin,
 geom = c("smooth", "point"),
 method = "lm")

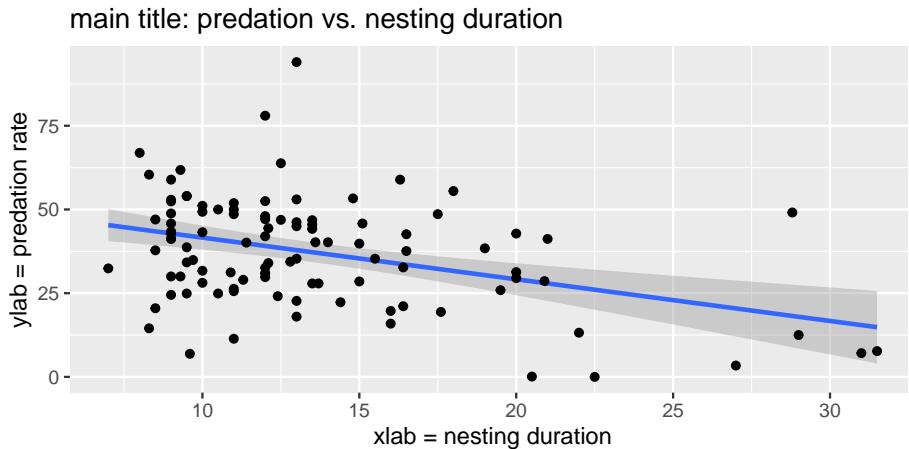
```



```

qplot(y = pred, x = nest.dur, data = martin,
 geom = c("smooth", "point"),
 #se = FALSE,
 method = "lm",
 xlab = "xlab = nesting duration",
 ylab = "ylab = predation rate",
 main = "main title: predation vs. nesting duration")

```



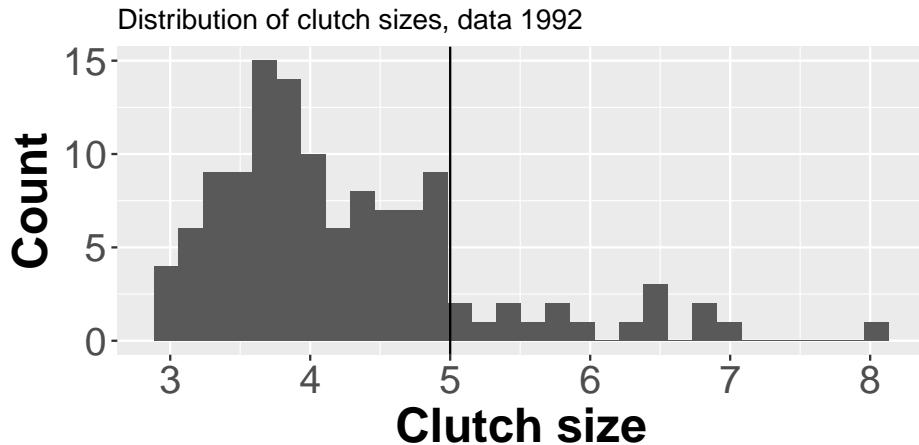
128.12 GGPlot

```
library(ggplot2)
```

- 128.12.1 Increase the font size for the x and y axes in ggplot?
  - 128.12.2 How do I increase the SIZE OF THE TICK LABELS on the x and y axes?
  - 128.12.3 How do I increase the size of TITLES or labels on the x and y axes?

#### 128.12.4 How do I make these things bold?

Use `“theme(axis.text = element_text(size = ...))”` for the axis labels. Use `“theme(axis.title = element”`



#### 128.12.5 How do I add horizontal and vertical lines to a ggplot?

```
geom_hline(xintercept = 0)
geom_vline(yintercept = 0)
```

#### 128.12.6 How do I flip or rotate the coordinates to that the y-axis becomes the x-axis in ggplot?

```
coord_flip()
```

This can be used to make coefficient plots and forests plots.

#### 128.12.7 How do I flip the y axis so that positive values are on the opposite side in ggplot?

```
scale_y_reverse()
```

#### 128.12.8 How do I manually set the colors for the legend or scale in ggplot?

#### 128.12.9 How do I change the colors of the legend or scale in ggplot?

```
scale_colour_manual(name="Experimental\nSettings",
 values=colors_BeS.3,
 labels=c("Greenhouse", "Greenhouse\n& Field"))
```

### 128.12.10 How do I make a confidence band around a line in ggplot?

keywords: (regression, linear model, CI, confidence interval)

```
geom_ribbon(aes(ymin=SE.real.minus,
 ymax=SE.real.plus),
 alpha=0.15,
 linetype = 0)geom_ribbon(aes(ymin=SE.real.minus,
 ymax=SE.real.plus),
 alpha=0.15,
 linetype = 0)
```

### 128.12.11 How do i set the limits on the x or y axis in ggplot?

The commands `scale_x_continuous()` and `scale_y_continuous()` can do several things. See also `xlab()` and `ylab()`

```
scale_x_continuous(name="Stem Length",limits=c(15,100)) + #Stem Length (cm)
scale_y_continuous(name="Reversion Probability")
```

### 128.12.13 How do I make changes to the legend in ggplot?

#### 128.12.14 How do I change the position of the legend in ggplot?

#### 128.12.15 How do I change the title over the legend?

#### 128.12.16 How do I change the font of the *title* of the legend in ggplot?

#### 128.12.17 How do I change the font of the *labels* of the legend in ggplot?

```
#keywords: ggplot, legend, font, bold,
theme(
 legend.position="right",
 #legend.position=c(0.15, .85),
 legend.title = element_text(colour="black",
 size=24,
 face="bold"),
 legend.text = element_text(colour="black",
 size = 20,
```

```
face = "bold")) +
```

### 128.12.18 How do I change the thickness of the axes in ggplot?

```
theme(axis.line = element_line(size=2, color = "black")) #makes axes lines thicker
```

### 128.12.19 How do I get rid of/remove/suppress the box around the plotting field

```
keywords: box, border, edging, around blot
theme(panel.border = element_rect(color = "white"))
```

#### 128.12.19.1 How do I get rid of the box around...

```
keywords: box, border, edging, around blot
theme(legend.key = element_rect(colour = 'white')) + # gets rid of box around length
```

#### 128.12.19.2 How do I get rid of the grid lines within the plot in ggplot?

```
theme(panel.grid.minor=element_blank(),
 panel.grid.major=element_blank())
#> List of 2
#> $ panel.grid.major: list()
#> ... - attr(*, "class")= chr [1:2] "element_blank" "element"
#> $ panel.grid.minor: list()
#> ... - attr(*, "class")= chr [1:2] "element_blank" "element"
#> - attr(*, "class")= chr [1:2] "theme" "gg"
#> - attr(*, "complete")= logi FALSE
#> - attr(*, "validate")= logi TRUE
```

#### 128.12.19.3 How do I set the font of the axes titles?

#### 128.12.19.4 How do I adjust the positions of the axes titles?

```
theme(axis.title.x = element_text(face="bold", size=32),
 axis.text.x = element_text(vjust=0.95,
 size=32,
 face="bold"),
 axis.title.y = element_text(face="bold",
 size=32,
```

```
 vjust=0.35),
 axis.text.y = element_text(vjust=0.5,
 size=32,face="bold"))

#> List of 4
#> $ axis.title.x:List of 11
#> ..$ family : NULL
#> ..$ face : chr "bold"
#> ..$ colour : NULL
#> ..$ size : num 32
#> ..$ hjust : NULL
#> ..$ vjust : NULL
#> ..$ angle : NULL
#> ..$ lineheight : NULL
#> ..$ margin : NULL
#> ..$ debug : NULL
#> ..$ inherit.blank: logi FALSE
#> ..- attr(*, "class")= chr [1:2] "element_text" "element"
#> $ axis.title.y:List of 11
#> ..$ family : NULL
#> ..$ face : chr "bold"
#> ..$ colour : NULL
#> ..$ size : num 32
#> ..$ hjust : NULL
#> ..$ vjust : num 0.35
#> ..$ angle : NULL
#> ..$ lineheight : NULL
#> ..$ margin : NULL
#> ..$ debug : NULL
#> ..$ inherit.blank: logi FALSE
#> ..- attr(*, "class")= chr [1:2] "element_text" "element"
#> $ axis.text.x :List of 11
#> ..$ family : NULL
#> ..$ face : chr "bold"
#> ..$ colour : NULL
#> ..$ size : num 32
#> ..$ hjust : NULL
#> ..$ vjust : num 0.95
#> ..$ angle : NULL
#> ..$ lineheight : NULL
#> ..$ margin : NULL
#> ..$ debug : NULL
#> ..$ inherit.blank: logi FALSE
#> ..- attr(*, "class")= chr [1:2] "element_text" "element"
#> $ axis.text.y :List of 11
#> ..$ family : NULL
```

```
#> ..$ face : chr "bold"
#> ..$ colour : NULL
#> ..$ size : num 32
#> ..$ hjust : NULL
#> ..$ vjust : num 0.5
#> ..$ angle : NULL
#> ..$ lineheight: NULL
#> ..$ margin : NULL
#> ..$ debug : NULL
#> ..$ inherit.blank: logi FALSE
#> ... attr(*, "class")= chr [1:2] "element_text" "element"
#> - attr(*, "class")= chr [1:2] "theme" "gg"
#> - attr(*, "complete")= logi FALSE
#> - attr(*, "validate")= logi TRUE
```

#### 128.12.19.5 How do I drop, remove or suppress the legend in ggplot?

```
theme(legend.position = "none")
```

#### 128.12.19.6 Remove facet strip completely

<http://stackoverflow.com/questions/10547487/r-removing-facet-wrap-labels-completely-in-ggplot2>

```
theme(strip.background = element_blank(),
 strip.text.x = element_blank())
```

#### 128.12.19.7 How do I increase the size text of the within facets?

#### 128.12.19.8 How do I increase the font size of facet labels?

<http://stackoverflow.com/questions/2751065/how-can-i-manipulate-the-strip-text-of-facet-plots-in-ggplot2/2751201#2751201>

```
theme(strip.text.x = element_text(size = 8, colour = "orange", angle = 90))
```

#### 128.12.19.9 How do I reorder a factor variable for plotting in ggplot?

```
levs <- mult.comp.df$names.ordered[order.x]
mult.comp.df$names.ordered <- factor(mult.comp.df$names.ordered,
 levels = levs)
```

#### 128.12.19.10 colorblind pallettes

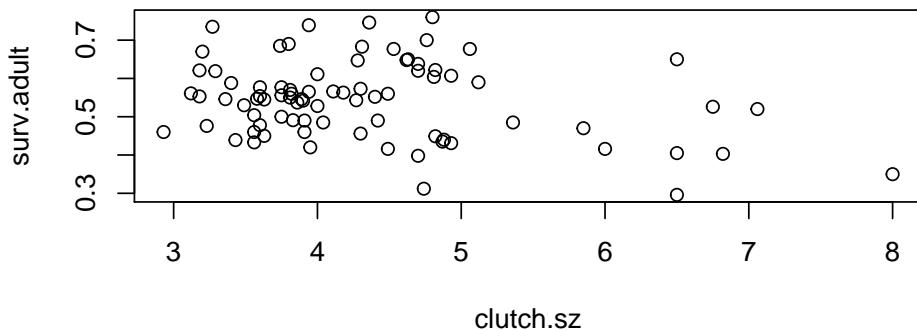
color blind pallettes <http://dr-k-lo.blogspot.com/2013/07/a-color-blind-friendly-palette-for-r.html> [http://www.cookbook-r.com/Graphs/Colors\\_%28ggplot](http://www.cookbook-r.com/Graphs/Colors_%28ggplot)

```
2%29/
```

```
#mult.comp.df$names <- with(mult.comp.df, reorder(names, coefficients, function(x) -order(x)))
```

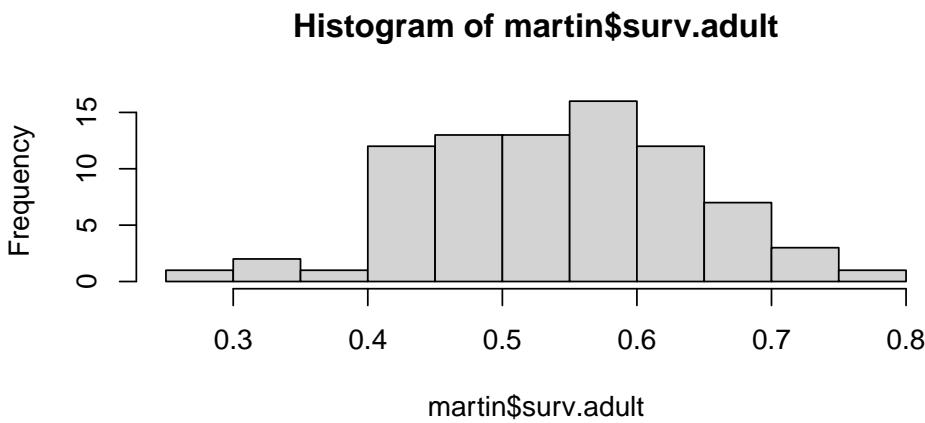
Plot clutch size versus duration of incubation using basic R plotting function

```
plot(surv.adult ~ clutch.sz, data = martin)
```



Plot distribution of survival rates

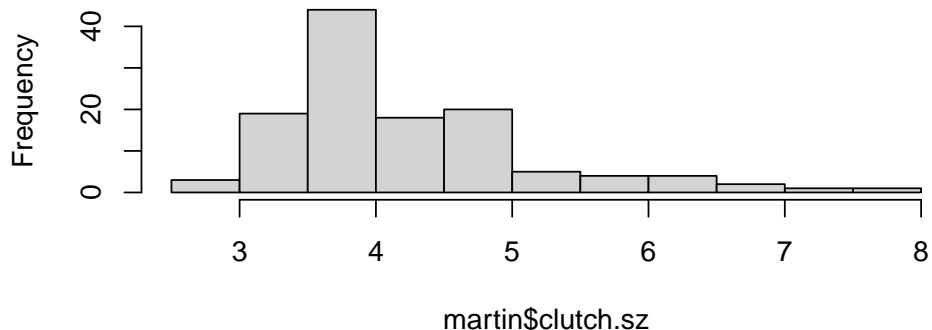
```
hist(martin$surv.adult)
```



Plot distribution of clutch sizes

```
hist(martin$clutch.sz)
```

### Histogram of martin\$clutch.sz

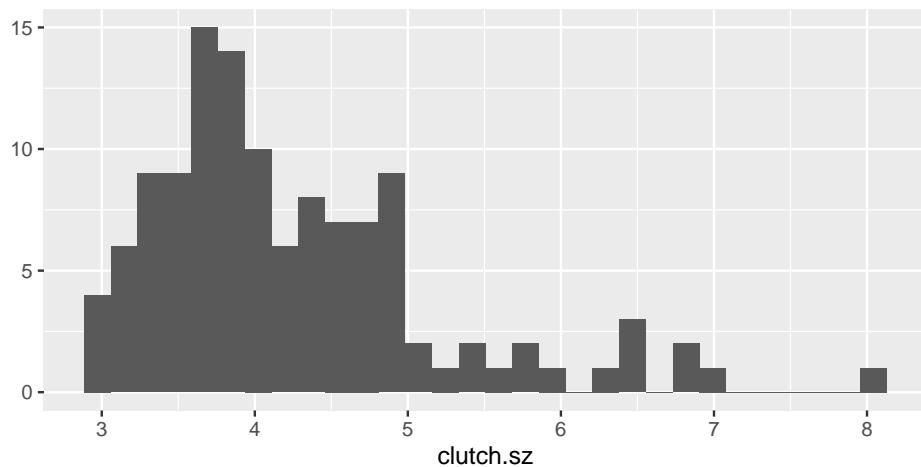


Load ggplot package

```
library(ggplot2)
```

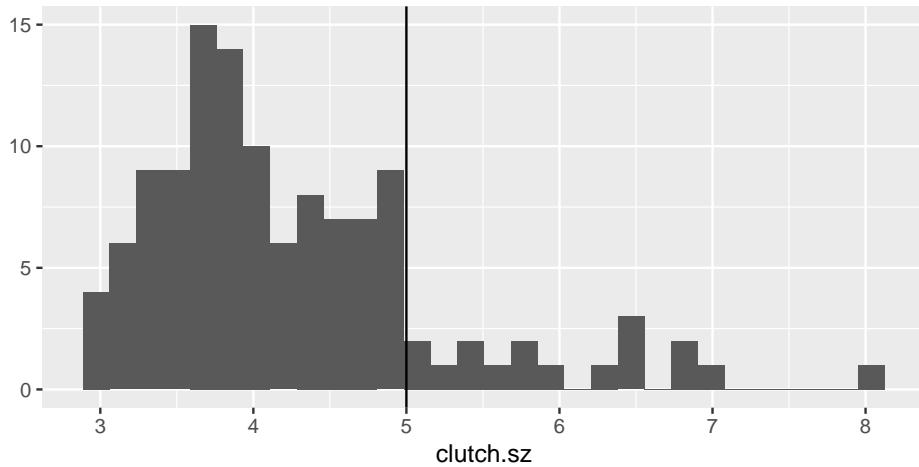
Plot distribution of clutch sizes in ggplot

```
qplot(clutch.sz,
 data = martin)
```



Add line at 6 for LOWA

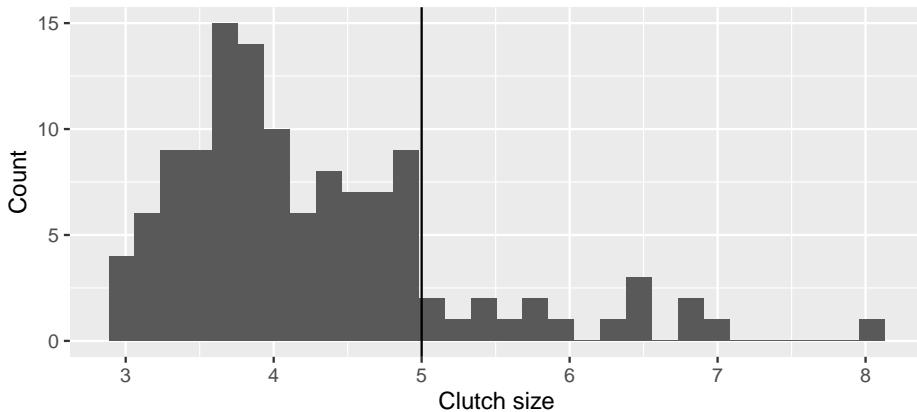
```
qplot(clutch.sz,
 data = martin) + geom_vline(xintercept = 5)
```



Add labels

```
qplot(clutch.sz,
 data = martin) +
 geom_vline(xintercept = 5) +
 xlab("Clutch size") +
 ylab("Count") +
 ggtitle("Distribution of clutch sizes, data 1992")
```

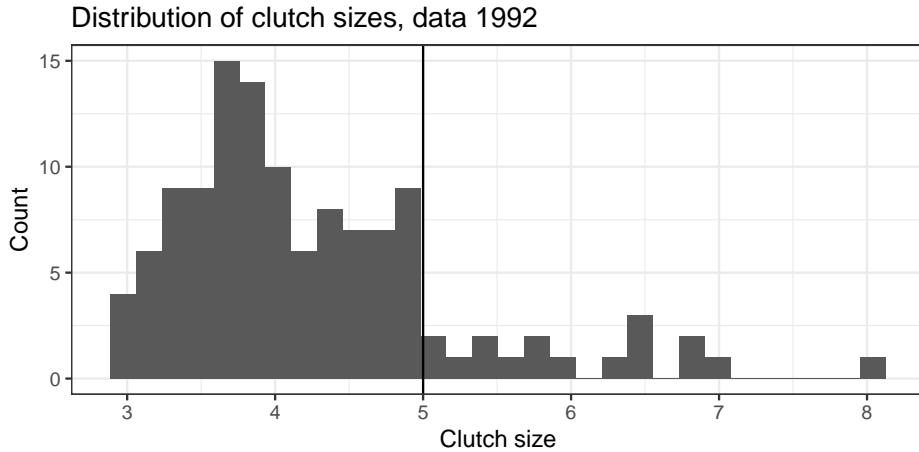
Distribution of clutch sizes, data 1992



Make background white

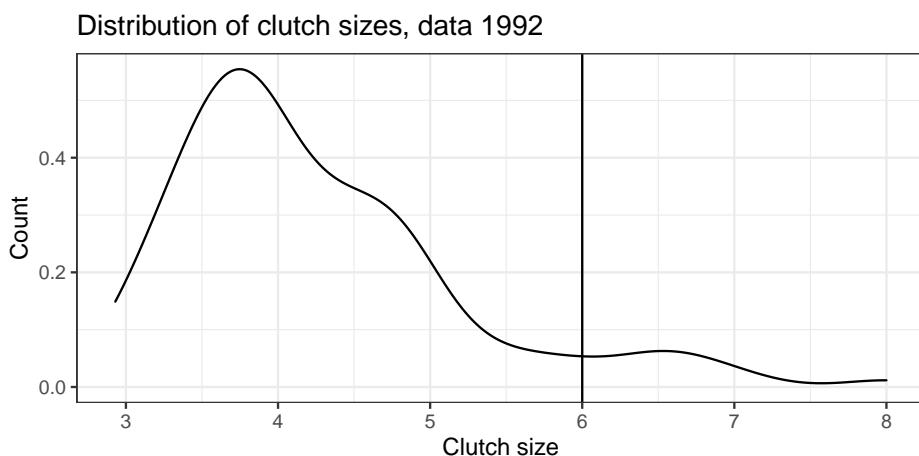
```
qplot(clutch.sz,
 data = martin) +
 geom_vline(xintercept = 5) +
 xlab("Clutch size") +
 ylab("Count") +
 ggtitle("Distribution of clutch sizes, data 1992") +
```

```
theme(axis.text=element_text(size=18),
 axis.title=element_text(size=22,face="bold")) +
theme_bw()
```



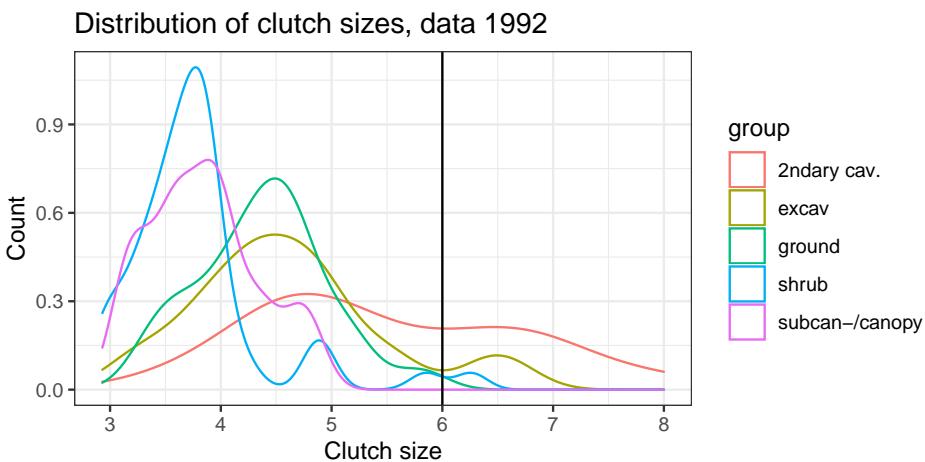
Plot as density curve

```
qplot(clutch.sz,
 data = martin,
 geom = "density") +
geom_vline(xintercept = 6) +
xlab("Clutch size") +
ylab("Count") +
ggtitle("Distribution of clutch sizes, data 1992") +
theme(axis.text=element_text(size=18),
 axis.title=element_text(size=22,face="bold")) +
theme_bw()
```



Plot as separate density curve for each nest type group

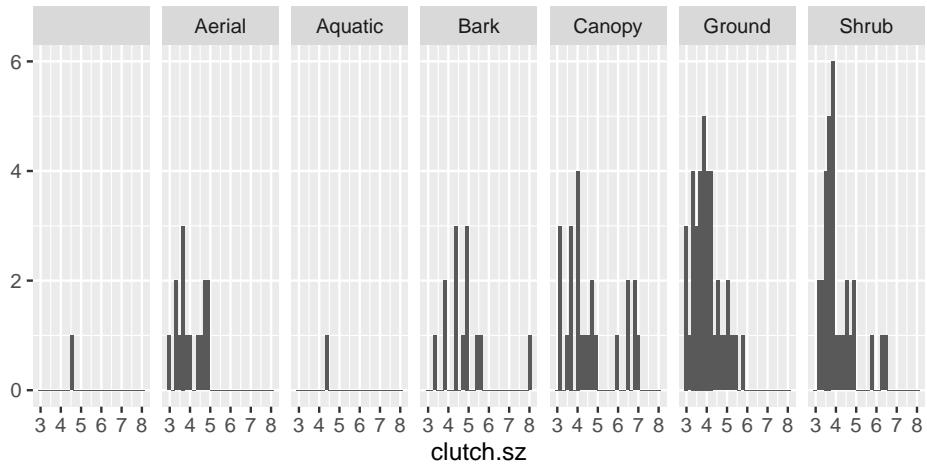
```
qplot(clutch.sz,
 data = martin,
 geom = "density",
 color = group,
 group = group) +
 geom_vline(xintercept = 6) +
 xlab("Clutch size") +
 ylab("Count") +
 ggtitle("Distribution of clutch sizes, data 1992") +
 theme(axis.text=element_text(size=18),
 axis.title=element_text(size=22,face="bold")) +
 theme_bw()
```



Plot distribution of clutch sizes by FORAGING site

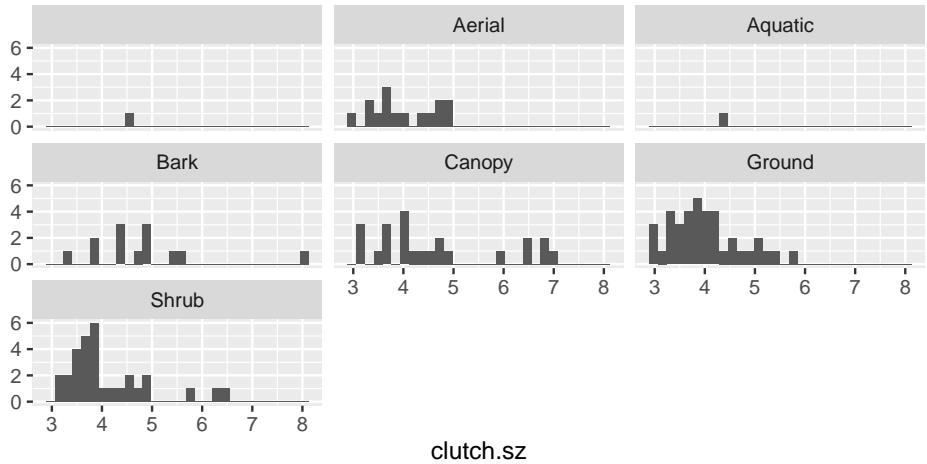
```
qplot(clutch.sz,
 data = martin,
 facets = . ~ foraging.site)
```

716 CHAPTER 128. GRAPHICAL PARAMETERS IN R WITH PAR(): WITH GREAT POWER COMES GREAT RESPONSIBILITY



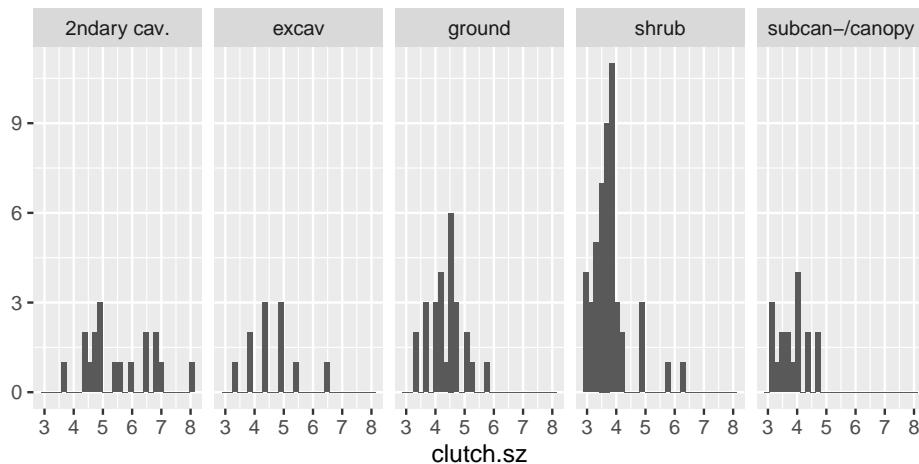
Use facet wrap

```
qplot(clutch.sz,
 data = martin) +
 facet_wrap(~foraging.site)
```



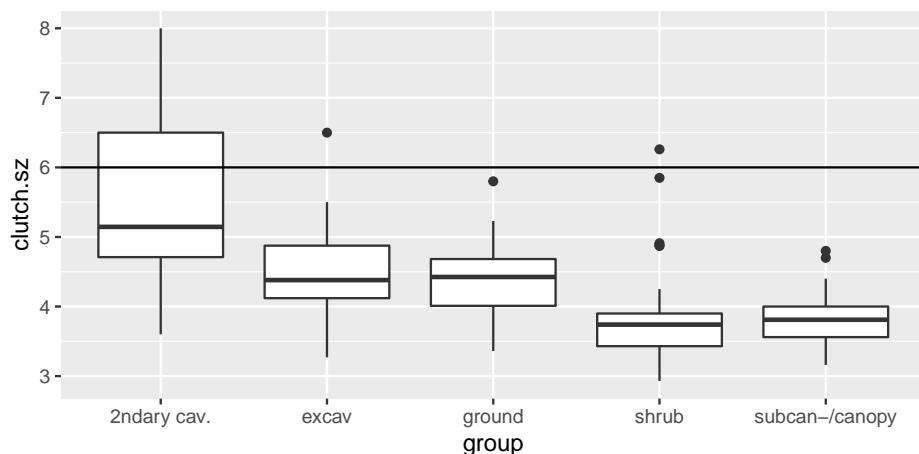
Plot distribution of clutch sizes by habitat group

```
qplot(clutch.sz,
 data = martin,
 facets = . ~ group)
```



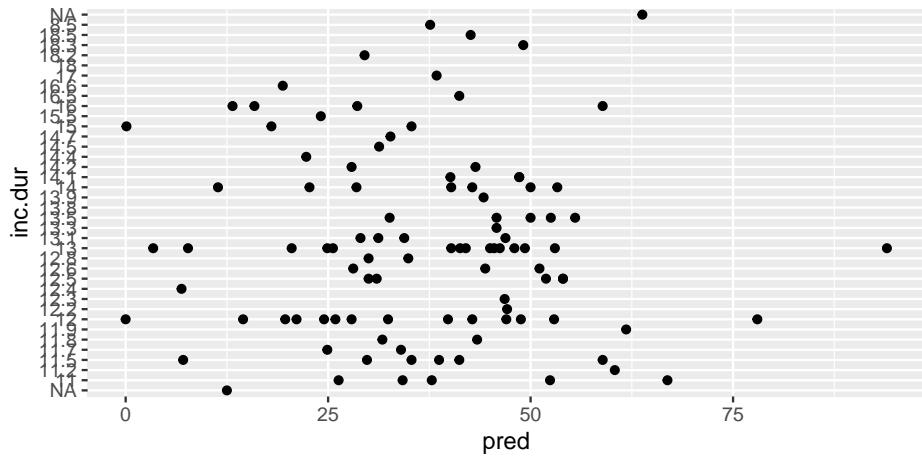
Boxplot

```
qplot(y = clutch.sz,
 x = group,
 geom = "boxplot",
 data = martin) +
 geom_hline(yintercept = 6)
```



Incubation period vs. predation

```
qplot(y = inc.dur,
 x = pred,
 data = martin)
```



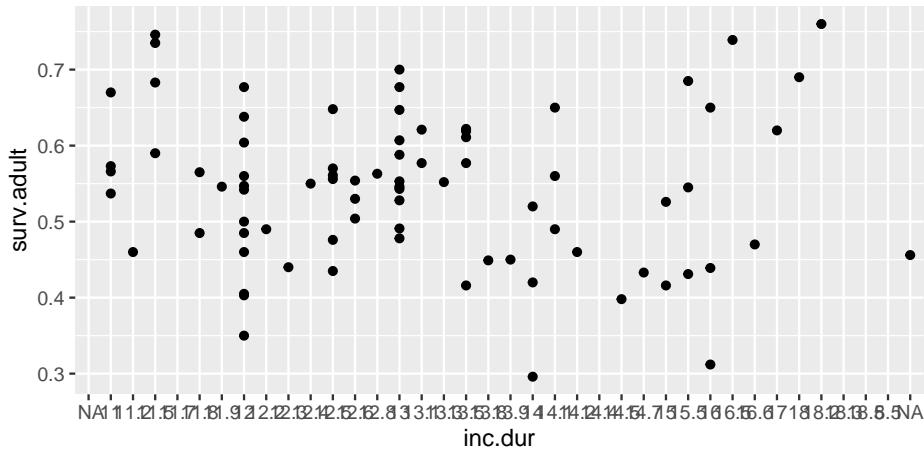
Plot adult survival versus duration of incubation in ggplot

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin)
```



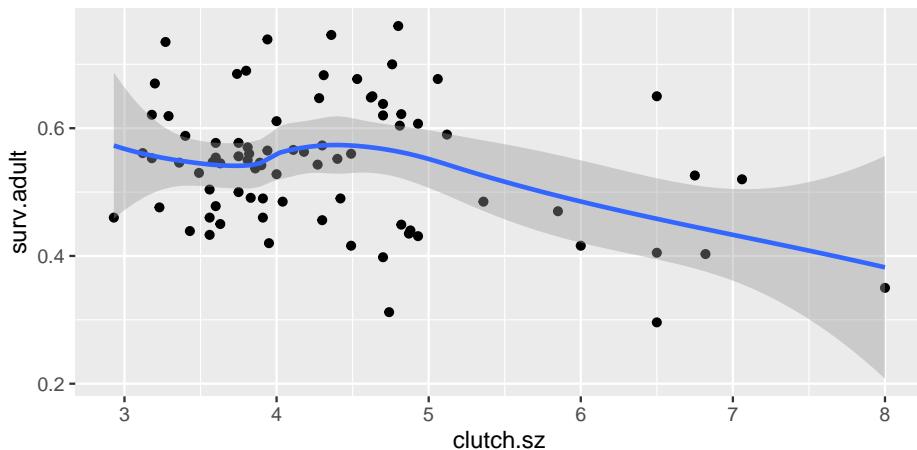
Plot adult survival versus duration of incubation in ggplot

```
qplot(y = surv.adult,
 x = inc.dur,
 data = martin)
```



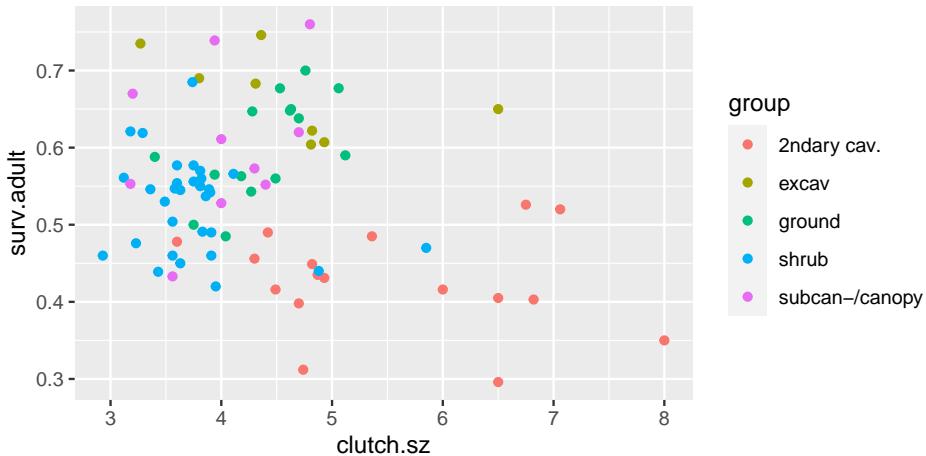
Add a trend line

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 geom = c("point", "smooth"))
```



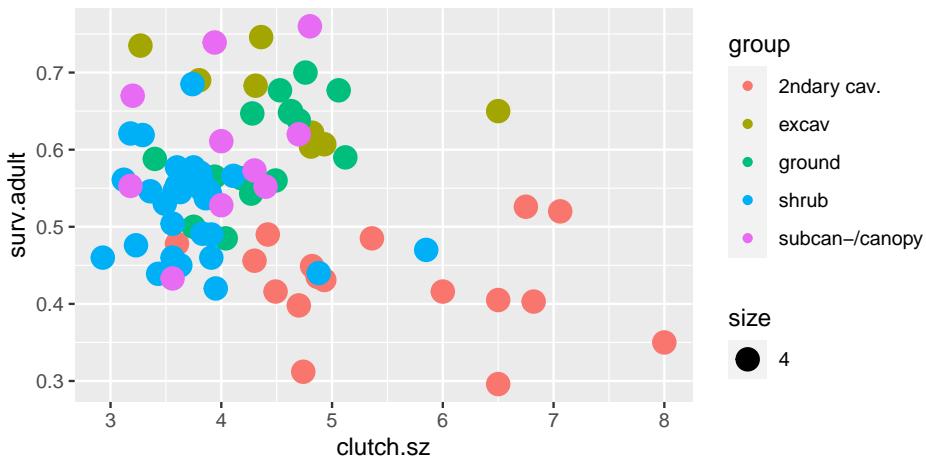
Color code each group

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 color = group)
```



Make size of points bigger

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 color = group,
 size = 4)
```



Add trend line to each group

```
qplot(y = surv.adult,
 x = clutch.sz,
 data = martin,
 color = group,
 geom = c("point", "smooth"),
 method = "lm",
 se = FALSE,
```

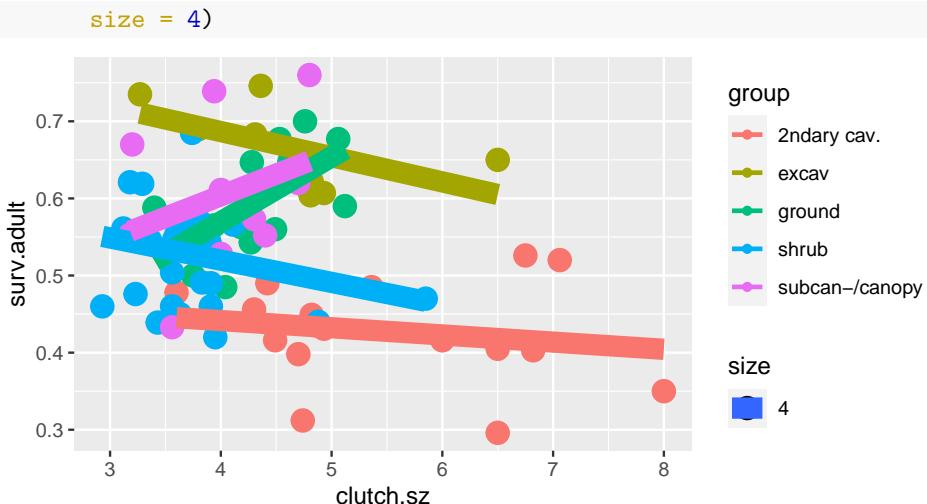
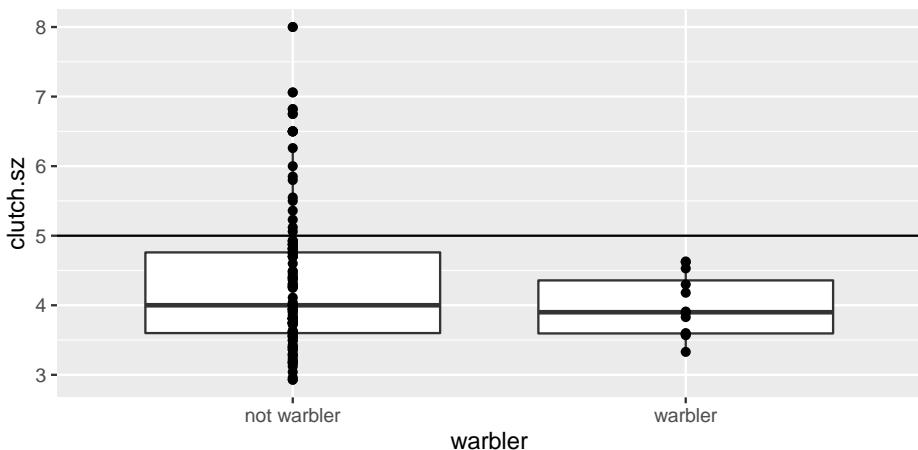


Figure out which species have “warbler” in their name

```
martin$warbler <- "not warbler"
martin$warbler[grep("Warb", martin$spp)] <- "warbler"
```

```
qplot(y = clutch.sz,
 x = warbler,
 geom = c("boxplot", "point"),
 data = martin) + geom_hline(yintercept = 5)
```



### 128.13 save ggplot

```
ggsave(...)
```



## Chapter 129

# Basic statistics equations

### 129.1 Variance

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

### 129.2 Standard deviation

$$\sigma = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

### 129.3 Standard error

#### 129.3.1 Equation

$$SE = \frac{SD}{\sqrt{N}}$$

#### 129.3.2 R-ish notation

$$SE = \frac{SD(x)}{\sqrt{\text{length}(x)}}$$

### 129.4 Pooled variance for two-sample t-test

In words in terms of variance:

$$var_p = \frac{df_1 * var_1 + df_2 * var_2}{df_1 + df_2}$$

In words in terms of standard deviation (SD)

$$var_p = \frac{df_1 * SD_1^2 + df_2 * SD_2^2}{df_1 + df_2}$$

In symbols, in terms of the variance ( $\sigma^2$ )

$$\sigma_p^2 = \frac{df_1 * \sigma_1^2 + df_2 * \sigma_2^2}{df_1 + df_2}$$

## 129.5 Standard error of the difference

$$SE_{diff} = \sqrt{s_p^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}$$

## 129.6 LS regression slope

- NOTE: squaring and multiplication occur before summing
  - therefore for denominator square each observation 1st, then sum them all up

### 129.6.1 Words

$$slope = \frac{\sum(X \text{ deviations}) * (Y \text{ deviations})}{\sum(X \text{ deviations}^2)}$$

### 129.6.2 Math

$$b = \frac{\sum(X_i - \bar{X}) * (Y_i - \bar{Y})}{\sum(X_i - \bar{X}^2)}$$

# Chapter 130

## Regression intercept

- Your mean Y value is related to your mean x value like this

### 130.1 Word equation

$$\bar{Y} = Intercept + Slope * \bar{X}$$

### 130.2 Math equation

$$\bar{Y} = a + b * \bar{X}$$

### 130.3 Do some algebra to solve for the intercept a

$$a = b * \bar{X} - \bar{Y}$$



# Chapter 131

## F statistics (F ratio)

### 131.1 Chi<sup>2</sup> test

(I just copied and pasted this and have not checked it)

$$\chi^2 = \sum_{i=1}^n \frac{(O_i - E_i)^2}{E_i} = N \sum_{i=1}^n \frac{(O_i/N - p_i)^2}{p_i}$$

### 131.2 Power

$$Power \propto \frac{ES * \alpha * \sqrt{n}}{\sigma^2}$$



## Chapter 132

# Two equivalent ways to set up a paired t-test in R

```
library(compbio4all)
```

First, I'll set up some random data. Let's say this is the heart of 5 students before they take my test and after the test. This first chunk of R code just makes the data and can be ignored

```
#Make some fake data
students <- c("Zetta", "Jude", "April", "Hendrick", "Cindy", "Modou", "Percy")

n <- length(students)
before.test <- round(runif(n, 45, 65), 1)

after.test <- round(before.test + 10 +
 rnorm(n, 0, 3), 1)

data.2.num.columns <- data.frame(students, before.test, after.test)

data.1.num.column <- data.frame(students = rep(students, 2),
 heart.rate = c(before.test, after.test),
 when = c(rep("before", n), rep("after", n)))
```

Data for a paired t-test can be made two different ways. First, it can be in a format where each of the two measurements on a subject/object/etc is in a separate column. There are therefore two columns of numeric data. One column is for the 1st time point (before), the other column is for the 2nd time point (after)

## 730CHAPTER 132. TWO EQUIVALENT WAYS TO SET UP A PAIRED T-TEST IN R

```
data.2.num.columns
#> students before.test after.test
#> 1 Zetta 53.2 62.9
#> 2 Jude 62.7 72.3
#> 3 April 63.8 74.3
#> 4 Hendrick 45.9 59.7
#> 5 Cindy 55.6 60.4
#> 6 Modou 62.8 77.9
#> 7 Percy 56.0 67.5
```

Alternative, the data can be set up with all of the numeric data in a single column. That is, the “before” data are stacked in a single column on top of the “After” data. Then, which time an observation corresponds to is indicated in another column. Here, the 1st 7 rows are from the “before” time period and the 2nd 7 rows are from the “after” time period.

```
data.1.num.column
#> students heart.rate when
#> 1 Zetta 53.2 before
#> 2 Jude 62.7 before
#> 3 April 63.8 before
#> 4 Hendrick 45.9 before
#> 5 Cindy 55.6 before
#> 6 Modou 62.8 before
#> 7 Percy 56.0 before
#> 8 Zetta 62.9 after
#> 9 Jude 72.3 after
#> 10 April 74.3 after
#> 11 Hendrick 59.7 after
#> 12 Cindy 60.4 after
#> 13 Modou 77.9 after
#> 14 Percy 67.5 after
```

Note that the names of the subjects occur in the **exact same order** within both groups. The first person listed is Zetta in row 1, which is a “before” measurement. The first “after” measurement (row 8) must therefore also be Zetta. The 2nd “before” row is for Jude, and therefore the 2nd “After” measurement must be for Jude (row 9). If the order is perfect and you do a paired t-test on the data, the answer will be wrong.

## Chapter 133

# Paired T-test on 2 column data

For the first data set, we can calculate the difference between the measurements and then do a one-sample t-test on the differences between the before and after

First, do the math

```
data.2.num.columns$before.minus.after <- data.2.num.columns$before.test - data.2.num.columns$after
```

Then do the one-sample t-test

```
t.test(data.2.num.columns$before.minus.after,
 mu = 0)
#>
#> One Sample t-test
#>
#> data: data.2.num.columns$before.minus.after
#> t = -9, df = 6, p-value = 1e-04
#> alternative hypothesis: true mean is not equal to 0
#> 95 percent confidence interval:
#> -13.80 -7.63
#> sample estimates:
#> mean of x
#> -10.7
```

Note that you get the exact same answer if you do after minus before, just the t-value is the opposite sign.

```
data.2.num.columns$after.minus.before <- data.2.num.columns$after.test - data.2.num.columns$before
t.test(data.2.num.columns$after.minus.before,
```

```
 mu = 0)
#>
#> One Sample t-test
#>
#> data: data.2.num.columns$after.minus.before
#> t = 9, df = 6, p-value = 1e-04
#> alternative hypothesis: true mean is not equal to 0
#> 95 percent confidence interval:
#> 7.63 13.80
#> sample estimates:
#> mean of x
#> 10.7
```

## Chapter 134

# Paired t-test on one column of data

We can have R do the math for us if the data is organized with all the numeric data in a single column. However, the data MUST be oragnized properly.

```
t.test(heart.rate ~ when,
 data = data.1$num.column,
 paired = TRUE)
#>
#> Paired t-test
#>
#> data: heart.rate by when
#> t = 9, df = 6, p-value = 1e-04
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> 7.63 13.80
#> sample estimates:
#> mean of the differences
#> 10.7
```

Note that I MUST include \*\* paired = TRUE \*\*. If we foret this we will run a standard two-sample test and get the wrong answer. Try it to see what happens. The t value will be wrong as well as the degrees of freedom.



# Chapter 135

## Running & reporting paired t-tests in R

```
library(compbio4all)
```

This vignette will walk through how to carry out a paired t-test in R and report the results.

### 135.1 Create dataframe

We'll use some data from a paper by Faaborg et al on bird populations in Puerto Rico. We'll look at 9 species of warblers, and compare the number of birds captured in mist nests in 1991 and in 2005 to determine if on average birds are declining at this study site

```
#make a vector of species names.
species <- c("OVEN", "WEWA", "NOWA",
 "BWWA", "HOWA", "AMRE",
 "CMWA", "NOPA", "PRWA")

#Number of birds of each species captured in 1991
N.1991 <- c(29, 6, 4, 60, 8, 19, 9, 7, 4)
N.2005 <- c(24, 5, 0, 16, 3, 9, 2, 5, 8)

#make dataframe
dat <- data.frame(species,
 N.1991,
 N.2005)
```

Take a look at the dataframe; we have 3 columns, one with the names of the 9

species, one with the number caught in 1991, and one with the number caught in 2005

```
head(dat)
#> species N.1991 N.2005
#> 1 OVEN 29 24
#> 2 WEWA 6 5
#> 3 NOWA 4 0
#> 4 BWWA 60 16
#> 5 HOWA 8 3
#> 6 AMRE 19 9
```

## 135.2 Paired t-test

Its a bit confusing, but there are multiple ways to do a paired t-test in R. (I can think about about 6, will focus on the 2 easiest ones). Paired t-tests are actually just a 1-sample t-test where the “1 sample” is a set of differences between pairs of data points. Each one of our species has a pair of data points: abundance in 1991 and abundance in 2005. We can give R the raw data and t.test will calculate the difference on the fly, or we can calculate the difference ourselves. If we let R calculate the difference, we **must** tell it that we are looking for a paired t-test by telling it “paired = TRUE”. If we calculate the difference ourselves we **must** tell it we want a 1-sample t-test, which is done by giving it a mean value against which to test the null hypothesis (“mu = 0”).

### 135.2.1 Paired t-test, Version 1

Paired t-test carried out by giving the t.test() function 2 columns from a dataframe.

- Note there is no “~”, just the name of each column, followed by a comma
- must include paired = TRUE

```
t.test(dat$N.1991, #column 1, then a comma
 dat$N.2005, #column 2;
 paired = TRUE)
#>
#> Paired t-test
#>
#> data: dat$N.1991 and dat$N.2005
#> t = 2, df = 8, p-value = 0.1
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -2.52 18.97
#> sample estimates:
#> mean of the differences
```

```
#> 8.22
```

### 135.2.2 Paired t-test, Version 2

Paired t-test as a 1-sample t-test on the *difference* between two columns.

- First calculate the difference between the columns
- T-test is given one column
- Note there is no “~”, just the name of the column that has the differences
- must set mu = 0
- there is NO “paired = TRUE”

```
#make new column with the difference between 1991 an 2005
dat$difference <- dat$N.1991 - dat$N.2005

#t.test() on difference
t.test(dat$difference,
 mu = 0)

#>
#> One Sample t-test
#>
#> data: dat$difference
#> t = 2, df = 8, p-value = 0.1
#> alternative hypothesis: true mean is not equal to 0
#> 95 percent confidence interval:
#> -2.52 18.97
#> sample estimates:
#> mean of x
#> 8.22
```

## 135.3 Reporting the results of a paired t-test

When we report a paired t-test we should give the p-value, the t statistic, and the degrees of freedom (df). Note that for a paired t-test the df are equal to n-1, where n is the number of pairs in the data set (eg, the number of differences calculated), *not* the total number of separate datapoints.

We should also report the effect size, which for a paired t-test is mean difference between the pairs; we should also report the 95% confidence interval for the effect size. Here, the mean difference is 8.2, which means on average there were 8 fewer individuals of each species captured in 2005 versus 1991. The 95 CI around this difference is large, from -2.5 to 19. Since it contains 0.0, the p value is greater than 0.05.

I would report the results of the t-test like this:

“There was a marginally significant difference in the number of birds of the 9

species captured in 1991 versus 2005 (paired t-test:  $t = 1.76$ ,  $df = 8$ ,  $p = 0.12$ ). The mean difference in the number captured between years was 8.2 birds (95%CI: -2.5 to 19)."

# Chapter 136

## Reporting statistical result from a 2-sample t-test

```
library(compbio4all)
```

### 136.1 Introduction

This vignette shows an example of reporting the results from a 2-sample t-test using data on the impact of invasive trout on salmon survival. The data are originally from Levin et al (2002) are used in an example in chapter 12 of Whitlock & Schutte 2nd. See `?brook_trout_ABD` for more details.

#### 136.1.1 References

Levin et al. 2002. Non-indigenous brook trout and the demise of Pacific salmon: a forgotten threat? PRSB 269. DOI: 10.1098/rspb.2002.2063

### 136.2 Outline of tasks

- Load the data into R
- Create a boxplot of the raw data
- State the relevant statistical null ( $H_0$ ) and alternative ( $H_a$ ) hypotheses
- Carry out an appropriate t-test
- Report the appropriate results in a full sentence as it would appear in a report or scientific paper

### 136.3 Dataset up

The data are available in the `wildlifeR` package and can be loaded using `data(wildlifeR)`. Note that if you use the `dataframe` in `wildlifeR` you have to calculate the survival rate by hand. I will remake the data by hand as an example of making a simple `dataframe`.

The following code contains the essential parts of the `dataframe`: a column for the survival rate and for whether brook trout are present or absent.

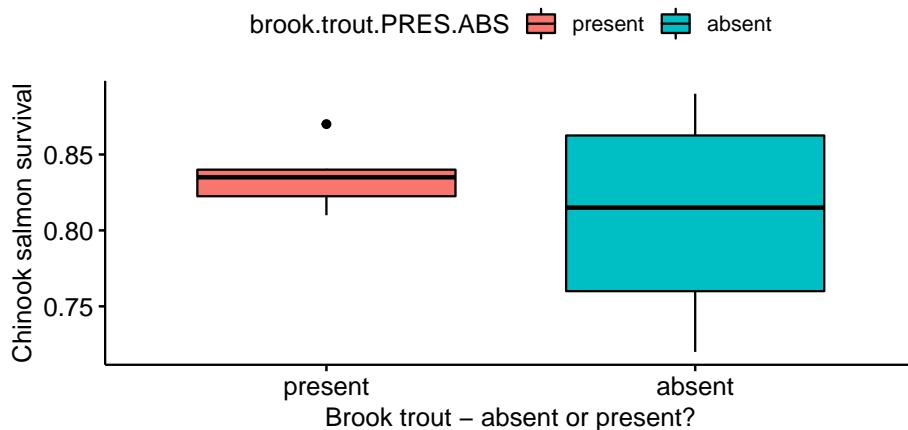
```
salmon <- data.frame(survival = c(0.83, 0.87, 0.82,
 0.84, 0.81, 0.84,
 0.72, 0.84, 0.75,
 0.79, 0.89, 0.87),
 brook.trout.PRES.ABS =
 c("present", "present", "present",
 "present", "present", "present",
 "absent", "absent", "absent",
 "absent", "absent", "absent"))
```

### 136.4 Plot raw data

I'll make a boxplot of the raw data using the `ggboxplot()` function from the package `ggpubr`, which contains wrappers that extend `ggplot2`. Be sure to download and install these packages if needed.

```
#library(ggplot2)
library(ggpubr)

ggboxplot(data = salmon,
 y = "survival",
 x = "brook.trout.PRES.ABS",
 fill = "brook.trout.PRES.ABS",
 xlab = "Brook trout - absent or present?",
 ylab = "Chinook salmon survival")
```



## 136.5 The hypotheses

The null ( $H_0$ ) and alternative hypotheses ( $H_a$ ) are as follows:

- $H_0$ : The survival rates of Chinook salmon are the same whether brook trout are present or absent
- $H_a$ : The presence of brook trout changes survival rates of salmon.

## 136.6 Do t-test

```
t.test(survival ~ brook.trout.PRES.ABS,
 data = salmon)
#>
#> Welch Two Sample t-test
#>
#> data: survival by brook.trout.PRES.ABS
#> t = -0.9, df = 6, p-value = 0.4
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -0.0961 0.0461
#> sample estimates:
#> mean in group absent mean in group present
#> 0.810 0.835
```

## 136.7 Report the results

For the real data, the results could be reported like this: “There was no evidence that the mean survival of salmon when brook trout are present (mean = 0.81) is different than when brook trout are absent (mean = 0.84; 2-sample t-test:  $p$

$= 0.44$ ,  $t = 0.82$ ,  $n = 12$  streams,  $df = 6$ .)”

Normally I would also report the standard errors (SE) around the means, but for this exercise we will ignore it.

### 136.7.1 Alternative results

What if the results really looked like this?

```
#>
#> Welch Two Sample t-test
#>
#> data: fake.surv by salmon$brook.trout.PRES.ABS
#> t = 4, df = 6, p-value = 0.01
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> 0.0317 0.1723
#> sample estimates:
#> mean in group absent mean in group present
#> 0.814 0.712
```

The results could be reported like this: “Survival of chinook salmon in streams where brook trout were present (mean = 0.71) was significantly lower than when brook trout were absent (mean = 0.81) with a mean difference of 0.10 (95% CI: 0.03-0.17; 2-sample t-test  $p = 0.012$ ,  $t = 3.6$ ,  $n = 12$  stream,  $df = 5.73$ )”

## Chapter 137

# Standard error of the difference between means

```
library(compbio4all)
```

- TO DO:

Double check degrees of freedom vs. sample size in equations



## **Chapter 138**

# **Standard error for the difference between 2 means**

- The difference between 2 means is used to calculate the t-statistic
- Another component is the standard error (SE) for the difference between 2 means
- The difference between 2 means is the effect size for a t-test (and any comparison of two groups; technically, an unstandardized effect size)
- The SE of the difference tells us about the uncertainty around the estimate of the effect size



## Chapter 139

### Exmple data: rat bladders

- 1st, I'll build up the components need to calculate the difference between 2 means using data from Motulsky's Intuitive Biostats, Chapter 30.
- 2nd, I'll make some functions that will help me do the calculations
- 3rd, I'll calcuale the SE of the difference



# Chapter 140

## 1) Make data table & calculate summary stats

### 140.1 The Data

Motulsky 2nd Ed, Chapter 30, page 220, Table 30.1. Maximal relaxaction of muscle strips of old and young rat bladders stimualted w/ high concentrations of nonrepinephrine (Frazier et al 2006). Response variable is %E.max

### 140.2 Create table of data and summary stats

This is not a standard R format but just used for illustartion and displaying data

```
The data
df.rats <- data.frame(old = c(20.8,2.8,50.0,33.3,29.4,38.9, 29.4,52.6,14.3),
 young = c(45.5,55.0, 60.7, 61.5, 61.1, 65.5,42.9,37.5, NA))

#Means
Calcualte means
mean.old <- with(df.rats, mean(old))
mean.young <- with(df.rats, mean(young, na.rm = T))

##put means into a vector
means.rats <- c(mean.old,mean.young)

add means to dataframe
df.rats2 <- rbind(df.rats, means.rats)
```

## 750CHAPTER 140. 1) MAKE DATA TABLE & CALCULATE SUMMARY STATS

```

##name the row that the means are in
row.names(df.rats2)[dim(df.rats2)[1]] <- "means"

#Standard deviation
##calc sd and add it to the dataframe
sd.old <- with(df.rats, sd(old))
sd.young <- with(df.rats, sd(young, na.rm = T))
sd.rats <- c(sd.old, sd.young)
df.rats2 <- rbind(df.rats2, sd.rats)
row.names(df.rats2)[dim(df.rats2)[1]] <- "SD"

#Sample size
n.old <- 9
n.young <- 8
n.rats <- c(n.old, n.young)
df.rats2 <- rbind(df.rats2, n.rats)
row.names(df.rats2)[dim(df.rats2)[1]] <- "N"

#Difference between means
Calculate difference btween the means
diff.means <- df.rats2["means", 2] -
 df.rats2["means", 1]

Add difference two dataframe
df.rats2 <- rbind(df.rats2, c(diff.means, NA))

##Label row
row.names(df.rats2)[dim(df.rats2)[1]] <- "Diff. means"

```

### 140.3 The Data table

- Here's the finished datatable
- Again, this is NOT a standard R format for working with data; this was just done to display the data like one might do in a spreadsheet

```

#pander package makes nice tables
library(pander)
pander(df.rats2)

```

	old	young
1	20.8	45.5
2	2.8	55
3	50	60.7

	old	young
<b>4</b>	33.3	61.5
<b>5</b>	29.4	61.1
<b>6</b>	38.9	65.5
<b>7</b>	29.4	42.9
<b>8</b>	52.6	37.5
<b>9</b>	14.3	NA
<b>means</b>	30.17	53.71
<b>SD</b>	16.09	10.36
<b>N</b>	9	8
<b>Diff. means</b>	23.55	NA

752CHAPTER 140. 1) MAKE DATA TABLE & CALCULATE SUMMARY STATS

# Chapter 141

## 2) Function to calculate the SE of the difference

### Steps

- Pool the variances of each sample
- Calc. SE of the difference using pooled variance

I'll make some functions that do this

### 141.1 Pooled variance

#### 141.1.1 Equation for pooled variance

(need to double check this; am not very good at code for equations yet)

In words in terms of variance:

$$var_p = \frac{df_1 * var_1 + df_2 * var_2}{df_1 + df_2}$$

In words in terms of standard deviation (SD)

$$var_p = \frac{df_1 * SD_1^2 + df_2 * SD_2^2}{df_1 + df_2}$$

In symbols, in terms of the variance ( $\sigma^2$ )

$$\sigma_p^2 = \frac{df_1 * \sigma_1^2 + df_2 * \sigma_2^2}{df_1 + df_2}$$

### 141.1.2 An R function to calcualte the pooled variance

Give this funtion the two sets of degrees of freedom and the standard deviations, and it returns the pooled variance.

```
#Formula for POOLED standard deviation
```

```
Note the formulas squares SD to get variance
var.pooled <- function(df1,df2,SD1,SD2){
 (df1*SD1^2 + df2*SD2^2)/(df1+df2)
}
```

### 141.1.3 Calcualte pooled variance using function

Using data from our data frame df.rats2

- 1st, get the degrees of freedom, df, into a vector w/ 2 element
- 2nd, get the standard deviations (SD) into a vector

```
Calculate pooled variance
```

```
extract N and SD for easy access
dfs <- df.rats2["N",]
SDs <- df.rats2["SD",]

dfs
#> old young
#> N 9 8

SDs
#> old young
#> SD 16.1 10.4

Apply function for pooled sd

Note: df = sample size - 1
var.pool <- var.pooled(df1 = dfs[1]-1,
 df2 = dfs[2]-1,
 SD1 = SDs[1],
 SD2 = SDs[2])

Add to pooled variance dataframe
df.rats2["var.pool",] <- c(var.pool,NA)
```

## Chapter 142

# Standard error of the difference

### 142.1 Equation for SE of difference

$$SE_d = \sqrt{var_p^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}$$

$$SE_d = \sqrt{\sigma_p^2 \left( \frac{1}{n_1} + \frac{1}{n_2} \right)}$$

- SE of the difference aka SE of the effect size
- Based on pooled variance, var.p
- Weighted by each sample size

### 142.2 Function to calculate SE of difference

This function takes the pooled variance and the two sample sizes.

```
Standard error of difference
Note that this uses sample size, NOT degrees of freedom (df)
SE.diff <- function(var.pool, n1,n2){
 sqrt(var.pool*(1/n1 + 1/n2))
}
```

### 142.3 Apply function SE.diff

```
#Apply function
Note: uses sample size, NOT df
se.dif <- SE.diff(var.pool,
 n1 = dfs[1],
 n2 = dfs[2])

#Update df
df.rats2["SE.diff",] <- c(se.dif, NA)
```

# Chapter 143

## Final dataframe

**Table 30.2: Results of unpaired t-test.** Motulsky Chapter 30, page 221.

```
pander(round(df.rats2,3))
```

	old	young
<b>1</b>	20.8	45.5
<b>2</b>	2.8	55
<b>3</b>	50	60.7
<b>4</b>	33.3	61.5
<b>5</b>	29.4	61.1
<b>6</b>	38.9	65.5
<b>7</b>	29.4	42.9
<b>8</b>	52.6	37.5
<b>9</b>	14.3	NA
<b>means</b>	30.17	53.71
<b>SD</b>	16.09	10.36
<b>N</b>	9	8
<b>Diff. means</b>	23.55	NA
<b>var.pooled</b>	188.3	NA
<b>SE.diff</b>	6.667	NA



# Chapter 144

## Anscomb's quartet

```
library(compbio4all)
```

### 144.1 The Data

#### 144.1.1 x variables

```
#x1 through x3 are the same
x1 <- c(10,8,13,9,11,14,6,4,12,7,5)
x3 <- x2 <- x1

#x4
x4 <- c(8,8,8,8,8,8,19,8,8,8)
```

#### 144.1.2 y variables are all different

```
y1 <- c(8.04,6.95,7.58,8.81,8.33,9.96,7.24,4.26,10.84,4.82,5.68)
y2 <- c(9.14,8.14,8.74,8.77,9.26,8.1,6.13,3.1,9.13,7.26,4.74)
y3 <- c(7.46,6.77,12.74,7.11,7.81,8.84,6.08,5.39,8.15,6.42,5.73)
```

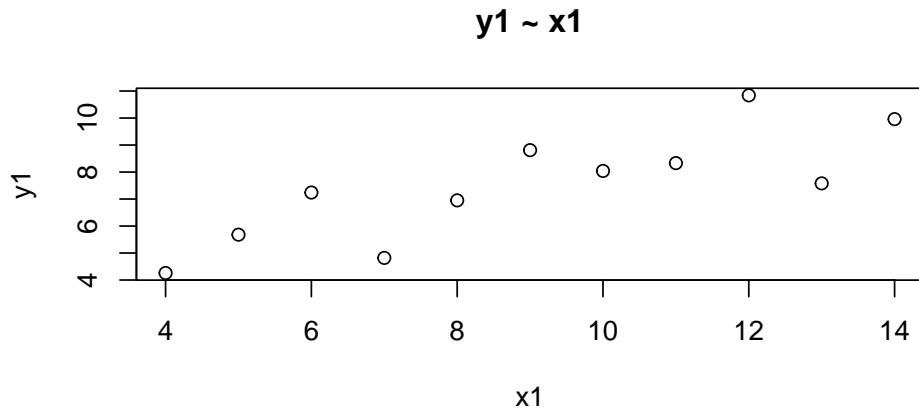


# Chapter 145

## Plots

### 145.1 Plot y1 vs x1

```
plot(y1 ~ x1,
 main = "y1 ~ x1")
```

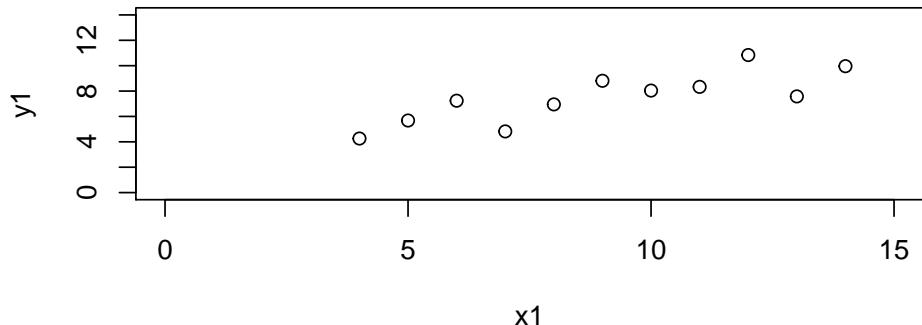


### 145.2 Adjust y and x limits

- arguments “xlim =”” and “ylim =”

```
plot(y1 ~ x1,
 main = "y1 ~ x1",
 xlim = c(0,15),
 ylim = c(0,14))
```

**y1 ~ x1**



### 145.3 Calcualte correlation of y1 and x1

```
cor(x1,y1)
#> [1] 0.816
```

### 145.4 Linear regression y1 ~ x1

- Save output to object “y1x1”
- Not “data =” is required b/c the data are living in seperate vectors stored directly into R’s memory

#### 145.4.1 Do linear regression

```
y1x1 <- lm(y1 ~ x1)
```

#### 145.4.2 Look at output

```
summary(y1x1)
#>
#> Call:
#> lm(formula = y1 ~ x1)
#>
#> Residuals:
#> Min 1Q Median 3Q Max
#> -1.9213 -0.4558 -0.0414 0.7094 1.8388
#>
#> Coefficients:
#> Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 3.000 1.125 2.67 0.0257 *
#> x1 0.816 0.283 2.89 0.0145 *
#> ---
```

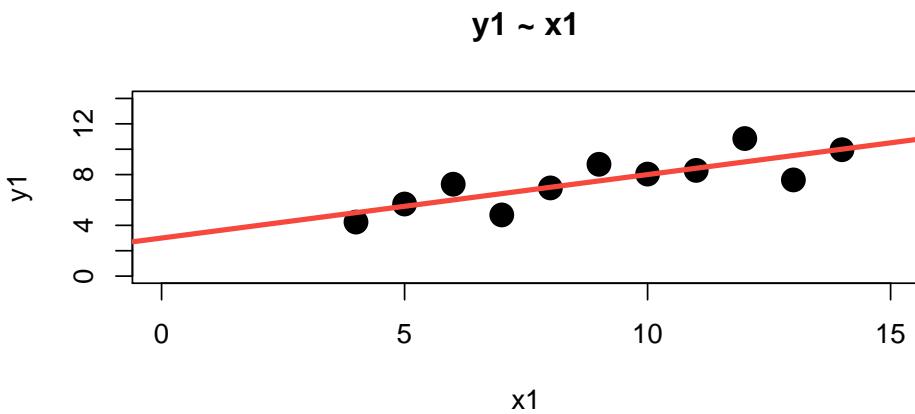
```
#> x1 0.500 0.118 4.24 0.0022 **
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.24 on 9 degrees of freedom
#> Multiple R-squared: 0.667, Adjusted R-squared: 0.629
#> F-statistic: 18 on 1 and 9 DF, p-value: 0.00217
```

#### 145.4.3 Plot data with regression line

- abline() add regression line
- col = changes color
- lwd = changes line width
- pch = 16 changes plotting symbol
- cex = 2 increase point size

```
#scatter plot
plot(y1 ~ x1,
 main = "y1 ~ x1",
 xlim = c(0,15), #axis limits
 ylim = c(0,14),
 pch = 16, #pnt shape
 cex = 2) #pnt size

#regression line
abline(y1~x1,
 col = 2, #line color
 lwd = 3) #line width
```



#### 145.5 Add $R^2$ values

- get  $R^2$  by saving output of summary command

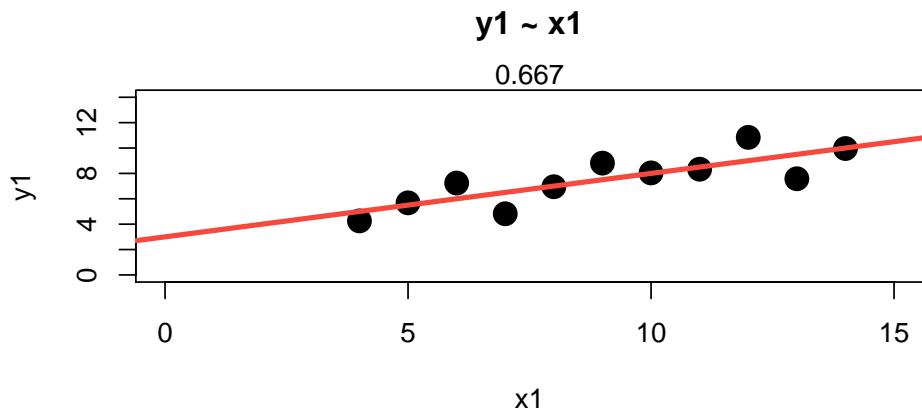
- `y1x1r2 <- summary(y1x1)$r.squared`
- use `mtext()` to add annotation of  $R^2$

```
#scatter plot
plot(y1 ~ x1,
 main = "y1 ~ x1",
 xlim = c(0,15), #axis limits
 ylim = c(0,14),
 pch = 16, #pnt shape
 cex = 2) #pnt size

#get r.squared
y1x1r2 <- summary(y1x1)$r.squared
y1x1r2 <- round(y1x1r2,3)

#annotation
mtext(text = y1x1r2)

#regression line
abline(y1x1,
 col = 2, #line color
 lwd = 3) #line width
```



# Chapter 146

## Make two side by side plots

- 1<sup>st</sup>, add another set of data, ie y2 and x2, etc
- Do regression on new data
  - Look at coefficients and R<sup>2</sup>
- Plot side by side with 1st data
  - use par(mfrow = c(1,2)) to set up a 1 x 2 panel
- NOTE: both plots must be in same Rmarkdown “chunk”

```
#SET up 1 x 2 panel
par(mfrow = c(1,2))

#PLOT 1
##scatter plot 1
plot(y1 ~ x1,
 main = "y1 ~ x1",
 xlim = c(0,15), #axis limits
 ylim = c(0,14),
 pch = 16, #pnt shape
 cex = 2) #pnt size

#get r.squared
y1x1r2 <- summary(y1x1)$r.squared
y1x1r2 <- round(y1x1r2,3)

#annotation
mtext(text = y1x1r2)

#regression line 1
abline(y1x1,
 col = 2, #line color
 lwd = 3) #line width
```

```
#PLOT2

regression y2 ~ x2
y2x2 <- lm(y2 ~ x2)

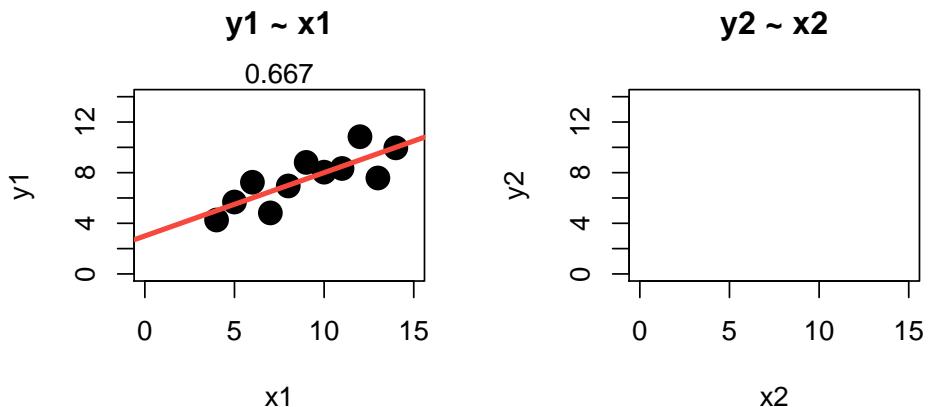
##scatter plot 2
note: I set col = 0 to hide output
deletec this or change it to 1
plot(y2 ~ x2,
 main = "y2 ~ x2",
 xlim = c(0,15), #axis limits
 ylim = c(0,14),
 col = 0,
 pch = 16, #pnt shape
 cex = 2) #pnt size

#get r.squared for 2nd model
y2x2r2 <- summary(y2x2)$r.squared
y2x2r2 <- round(y2x2r2,3)

2nd R^26 annotation
note: I set col = 0 to hide output
deletec this or change it to 1
mtext(text = y2x2r2,
 col = 0)

2nd regression line
note: I set col = 0 to hide output
deletec this or change it to 1
abline(y2x2,
 col = 0, #line color
 lwd = 3) #line width
```

767





# Chapter 147

## Super scripts in Rmarkdown text, R plots, and Rmarkdown tables

Superscripting is easy in MS Word. Its harder in Rmarkdown and R, and unfortunately requires different code depending on the context. Below I show how to do it within the text of rendered Rmarkdown document, in a plot (works for any R context, whether in Rmarkdown or not), and in a rendered Rmarkdown table using the package `pander`.

### 147.1 Superscript in Rmarkdown

- “ $R^2$ ” prints  $R^2$
- “ $R^2$ ” prints  $R^2$ 
  - note 2<sup>nd</sup> carrot after the “2”

### 147.2 Plotting super scripts in R plots

- This requires using the `expression()` and `paste()` commands
- `expression()` converts  $R^2$  into R w/ a superscript
- `paste()` is used to combine  $R^2$  w/ the value
- This works in any R context and is not specific to Rmarkdown

Model cat data

```
library(MASS)
data("cats")
```

```
mod <- lm(Hwt ~ Bwt, data = cats)
```

Do summary of model

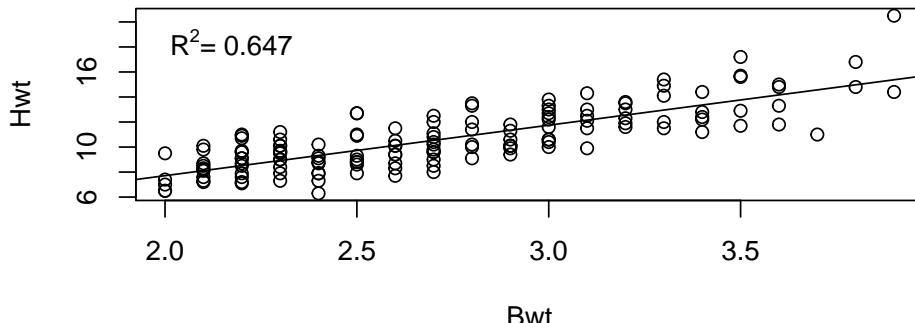
```
mod.sum <- summary(mod)
```

Get  $R^2$  value

```
library(grDevices)
R2 <- mod.sum$r.squared
R2.exp <- expression(paste(" ", R^2, "= 0.647"))
```

### 147.3 Annotate plot with text()

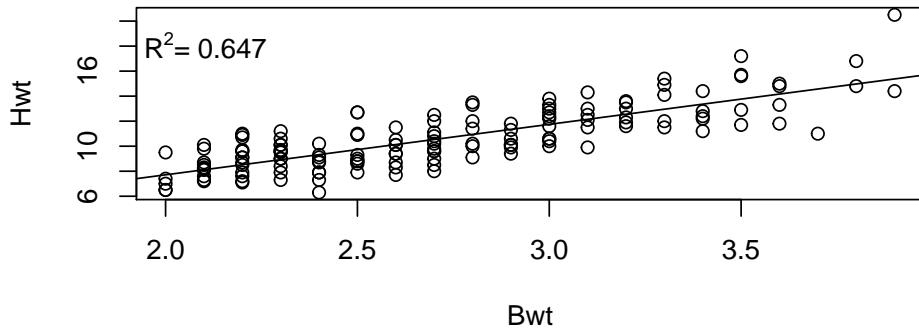
```
plot(Hwt ~ Bwt, data = cats)
abline(mod)
text(x = 1.95, y = 18,
 label = R2.exp,
 pos = 4)
```



# NOTE: pos = 4 right justifies the symbol

### 147.4 Annotate plot with mtext()

```
plot(Hwt ~ Bwt, data = cats)
abline(mod)
mtext(R2.exp,
 side = 3, #side = 3 sets to the top margin
 line = -1.75, #-1.75 sets below the box
 adj = 0) #adj=0 left justifies
```



## 147.5 Super script in Rmarkdown table using pande()

```
temp.df <- data.frame(c(mod.sum$fstatistic[1],mod.sum$r.squared))

names(temp.df) <- NULL

row.names(temp.df) <- c("F","R^2")

library(pander)
pander(temp.df)
```

<b>F</b>	259.8
<i>R</i> <sup>2</sup>	0.6466



# Chapter 148

## Types of data

```
library(compbio4all)
library(ggpubr)
```

Data is the lifeblood of science. Science is science because it is based on empirical observations about the natural world. There's lots of other essential components - hypotheses, theories, models, math - but empirical observations are either the input for these other activities or the final arbitrator of their value.

We therefore use the word "data" a lot in science, and use it as adjective or adverb for many scientific activities. We do **data collection** in our labs, record data in lab notebooks or **data sheets**, make graphs to do **data visualization**, use stats to do **data analysis**, and we read published paper to evaluate other scientists' data.

Data, however, can take on many different forms. Just as "data" can be used as an adjective or adverb to describe many scientific activities, there are many adjectives which specify exactly the nature of the data we are working with.

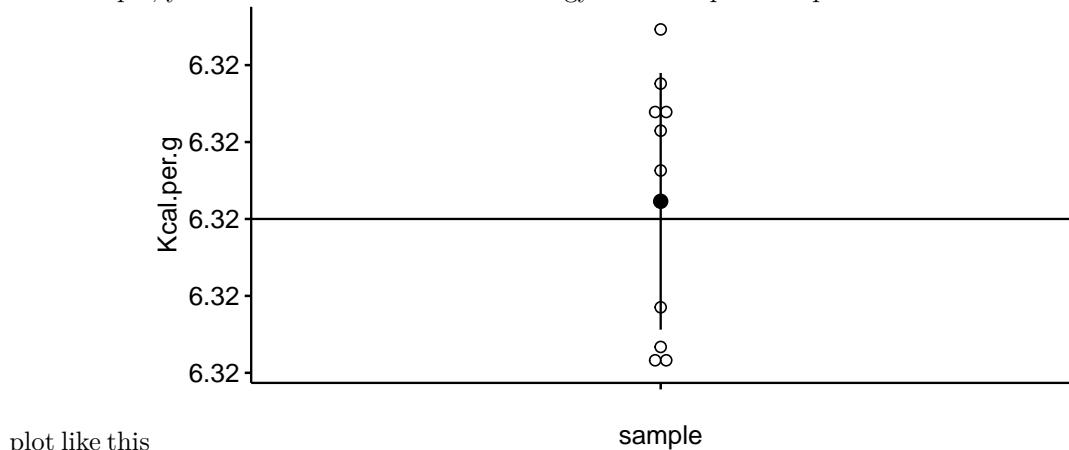
For example, you might want to know if a particular allele (genetic variant) occurs in a strain of **E. coli**. You therefore do PCR with primers specific to that allele and run out the reaction on a gel. The presence or absence of a DNA band on the gel is a type of **qualitative data**: evidence for whether the strain possess a certain quality.

[transition]

Frequently we carry out repeated observations or measurements, such as recording how quickly an object falls to calculate acceleration due to gravity, how frequently UV radiation prevents the growth of bacteria, or rating how much damage an herbivore caused to a leaf on a scale from 1 to 10. These **numeric data** are usually summarized mathematically using means and percentages. If

you're being thorough - and you should – you should characterize variation between repeated observations using standard deviations (SD), standard errors (SE), or confidence intervals (CI).

For example, you could measure the caloric energy in 10 samples and produce a



plot like this

sample

#### 148.0.1 Weapons of math destruction

Something that is not always appreciated is that once you start applying math like means and standard deviations to data you need to be mindful of exactly what type of data you are working. Indeed, applying the wrong kind of mathematical operation to specific kinds of data can result in nonsensical output or even erroneous conclusion. This is most usually a problem when percentages are involved or rating scales. In the next section I'll work through an extended example to show you the kind of problem that can come up. After that we'll talk about different kinds of data and the right kind of math to deal with them.

#### 148.0.2 Example: survival rates of bacteria

Here's an example of how math can run amok. Let's say I want to know how frequently a strain of bacteria is killed by UV radiation. I have 3 petri dishes and I dilute out a broth culture of bacteria so that 10 cells end up on each plate. I then put the plates into an incubator that contains a UV light. When I check back on the plates I get the following results:

Plate 1: 2 of the 10 cells grew into colonies  
 Plate 2: 0 of the 10 cells grew into colonies  
 Plate 3: 0 of the 10 cells grew into a colony

So I have 3 data points: 20% survival, 0% survival, and 0% survival. I can calculate the mean survival rate as

$$(20\% + 0\% + 0\%) / 3 = 20\% / 3 = 6.67\%$$

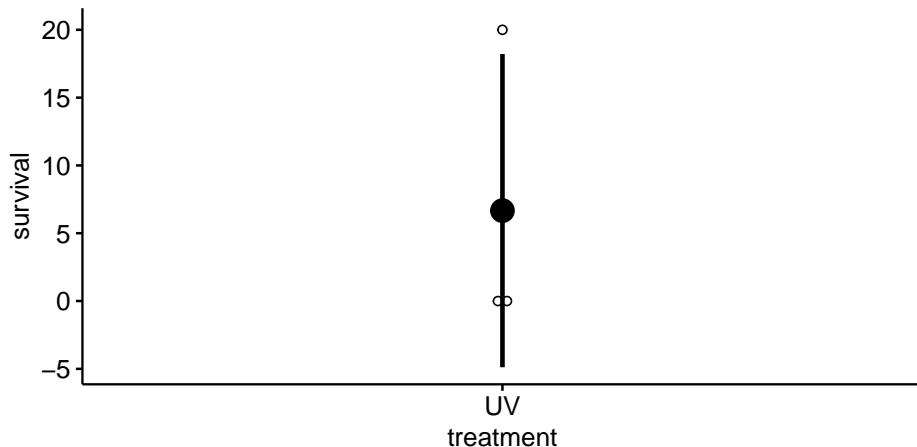
Plugging these numbers into a spreadsheet we can quickly calculate a standard deviation, for example using Excel's STDEV.S() function like this:

```
#> [1] 6.67
#> [1] 11.5
```

	A
1	**bacteria.survival**
2	20
3	0
4	0
5	=STDEV.S(A2:A4)

This tells us that the standard deviation of the data is 11.5. You may have learned that a general rule of thumb is that we should expect about 60% of our data points to fall within plus or minus 1 SD of our mean.

It's always good to plot data; ideally you want to plot the raw data and your mean, and also the standard deviation as error bars around the mean. For our study we get something like this



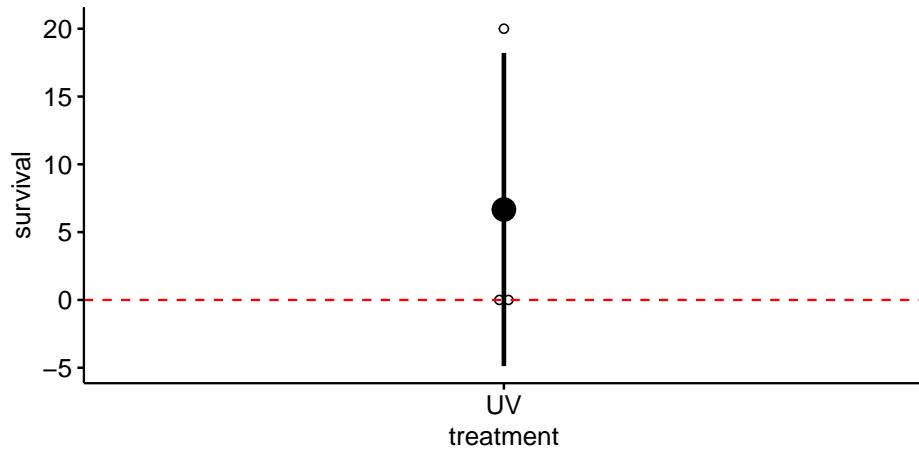
This plot has 3 elements related to the data:

1. The 3 small dots are the 3 data points
2. The big point is the mean
3. The error bars extend plus 1 SD from the SD and minus 1 SD from the mean

Look carefully at this graph; there's something peculiar. Can you spot it?

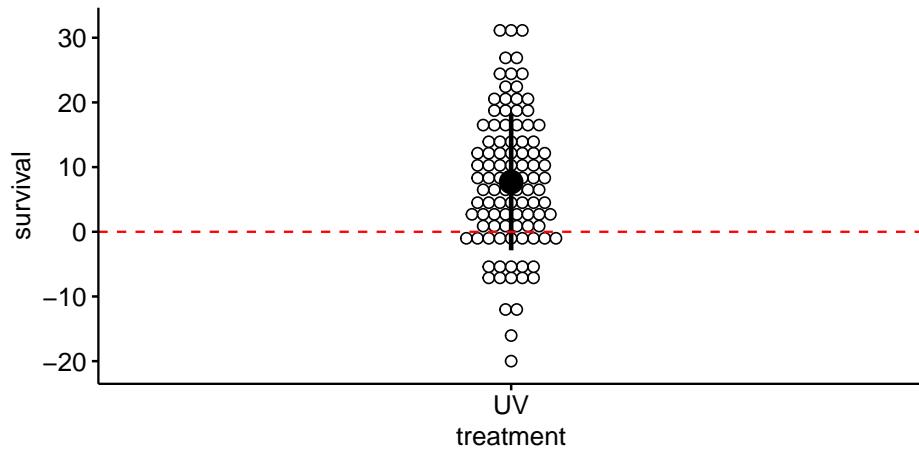
Let's think about what we plotted. The data points are what they are: 20%, 0% and 0%. Nothing wrong with that. The mean looks ok too; it's supposed to represent the "Central tendency" of the data or your best guess about what should happen if you repeated the experiment. For example, if you plated 100 bacteria you expect 6.67% to survive, so about 6 or 7.

What about the error bars? There's actually something wrong there. Recall that about 60% of our data should between  $+1SD$  and  $-1SD$ . Another way of thinking about this is that if we ran the experiment many time, we'd expect 60% of data points to be within those error bars, and only 40% higher or lower. Given what error bars represent, there's something amiss with ours. Let's add something to the graph to see if you can spot it:



I've added a line at 0. Two of our data points are 0%; 0 is a perfectly valid data point; it could even be the mean if all of our data were 0%. Notice, however, that the error bar extends *beyond* 0. It actually goes down to about -5%. Recall what I just said about standard deviations: if you repeated your experiment many times (say 100) you'd expect 60% to be plotted within those error bars, and 40% to be higher or lower. What our error bars are implying is that data points could fall between 0 and -5%, and some may even be less than that!

In a picture, this is what this would look like, where each dot is one of 100 imaginary replications of our study.



Wow! What's implied by our initial graph is that many experiments could produce negative values, sometimes with "survival" of -20%!

Our simple math of the standard deviation really went afoul when we plugged percentages into it. The solution to this problem is to use a different formulation of the standard deviation that is appropriate for this type of frequency data. We won't worry about the math, but if we do things correctly we should get error bars which almost go to zero (2%) and still extend up to about 20%.

```
surv <- c(.2,0,0)
trials <- c(10,10,10)
summary(glm(surv ~ 1, weights = trials, family = "binomial"))

arm:::invlogit(-2.6391)
arm:::invlogit(-2.6391+0.7319*sqrt(3))*100
arm:::invlogit(-2.6391-0.7319*sqrt(3))*100
```

### 148.0.3 Types of data

Statisticians have specific classifications for different types of data. You will see different ways of presenting this these classifications; what I have done here is focused on the most practical features that impact common data analyses.

1)quantitative data (but trying to think of a better name) 2)ordinal 3)frequencies (better name?) 4)counts 5)continuous percentages

skip: nominal (categorical)

for the mathematically inclined :these categories relate to the type of distributions which can be used to describe data

- **Quantitative data** measurements such as length, mass, pH, area, current, produced by instruments such as rulers, calipers, meters.

“continuous quantitative data”

- **Ordinal data** or “ordered categories” produced by ranking, scoring, or classifying something. The most familiar ordinal scale is letter grades in school. Common now are also online rankings, such as the number of stars from 1 to 4 used for Amazon products. In medicine we use pain scale from 1 to 10, and in orthopedics and physical therapy joint mobility scales are used. Ecologists studying herbivory and disease frequently assign a score to plants from 1 (no damage) to 10 (completely destroyed). In principle most ordinal data could be measured with some instrument to yield quantitative data, but that might be too expensive or difficult.

ranks vs ordinal

“ Ordinal measurements have imprecise differences between consecutive values,

but have a meaningful order to those values, and permit any order-preserving transformation.”

- **Frequency data**, which comes from determining the resolution of either-or occurrences, such as the survival of bacterial cells or inheritance of a Y chromosome during conception. A key feature is that each “occurrence” happens on its own, like flipping a coin. Frequency data can be summarized using a percentage, where the denominator is the number of occurrences and the numerator is the number of times a particular outcome happens. Don’t confuse frequency data with count data OR continuous percentage data (see below)! A key diagnostic of frequency data is that you can always identify a denominator. For example, we can count up the number of games the Pittsburgh Pirates have won this season; we can also calculate the percentage of wins. Since there is a denominator available (total number of games, this is frequency data). Similarly, there’s always an upper bound to frequencies: you can’t win more than 100% of the games or can’t have more than 100% of your fruit flies be XY male. Another diagnostic of count data is that the numerator and denominator are both always whole numbers.

counted fractions Tukey

- **Count data** of the number of items or objects: number of flowers, number of offspring, number of tumors. This count data should not be confused with frequency data. A key diagnostic: frequency data can be summarized using a percentage, while this never makes sense for count data. A key diagnostic of frequency data is that you can always identify a denominator. For example, we can count up the number of games the Pittsburgh Pirates have won this season; we can also calculate the percentage of wins. Since there is a denominator available (total number of games, this is frequency data). Similarly, there’s no upper bound. So while you can’t win more than 100% of your baseball games (frequencies) there’s no limit on the number of runs you can score in a single game (count data)

discrete quantitative

- **Continuous percentages**, such how much of a petri dish is covered by bacteria, how much of a patient’s body was burned, or the exact percent of a leaf eaten by an herbivore. For this type of data, usually the numerator and the denominator are some form of quantitative data.

# Chapter 149

## Summary tables in base R

### 149.0.1 Load Martin 1992 data

```
library(compbio4all)
data(martin1995)
```

### 149.0.2 Questions

T1)How do I make simple tables in R? T2)How do I make 2-way tables in R?  
T3)How do I make 3-way tables in R?

#### 149.0.2.1 Tables

Our object data has a column group giving the general location where each species in the dataframe nests and another columns for where the birds forage.

**149.0.2.1.0.1 T1)How do I make simple 1-way tables in R?** We can make a simple table for where birds nest, where they forage and their general latitudinal region like this:

```
#Table for nest location
table(martin1995$group)
#>
#> excavating ground-nesting non-excavating
#> 11 26 18
#> shrub or low foliage Subcanopy or canopy
#> 49 18

#Table for forage location
table(martin1995$foraging.site)
#>
```

```
#> Aerial Aquatic Bark Canopy Ground Shrub
#> 1 15 1 13 24 38 30
#Table for general latitudinal location
table(martin1995$trop.or.temp)
#> <table of extent 0>
```

A nice command to learn to use in R is with(). Here's what it can do when making a table; If you were to read this out loud like you were telling R what to do, it would be something like this "With the dataframe called data, make a table from the group column." This isn't that helpful in this situation, but for 2-way tables and other tasks it is nice.

```
#Making a table with with()
with(martin1995, table(group))
#> group
#> excavating ground-nesting non-excavating
#> 11 26 18
#> shrub or low foliage Subcanopy or canopy
#> 49 18
```

An aside on tables: The summary() command can also give you simple 1-way tables. However, the table() command will make a table out of whatever you give it, while summary() is more generic. For example, the way that these particular data have been loaded in, R is treating information that is in text form, like nesting location, as text. So when I use summary, I get this

```
summary(martin1995$group)
#> Length Class Mode
#> 122 character character
```

R is very picky about different kinds of data. In order for R to make a table from this column using the summary() command, you'd need to tell it that the data belong to different categories or different levels of "factor variable". This is done with the factor() command. here, I'll tell R that the group column is factor data and then make a table with summary()

```
#Change character data to a factor
martin1995$group <- factor(martin1995$group)

#Now I get a table from summary()
summary(martin1995$group)
#> excavating ground-nesting non-excavating
#> 11 26 18
#> shrub or low foliage Subcanopy or canopy
#> 49 18
```

**149.0.2.1.0.2 T2)How do I make simple 2-way tables in R?** We can cross classify the nesting groups and foraging groups very easily, aka, we can make a 2-way table.

Note that you have to be explicit about where each piece of data from by putting “data\$” in front of both columns

```
#With nesting location in rows
table(martin1995$group, martin1995$foraging.site)
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 10 1 0 0
#> ground-nesting 1 0 0 1 3 18 3
#> non-excavating 0 7 1 2 6 1 1
#> shrub or low foliage 0 5 0 0 1 19 24
#> Subcanopy or canopy 0 3 0 0 13 0 2

#With nesting location in columns
table(martin1995$foraging.site, martin1995$group)
#>
#> excavating ground-nesting non-excavating shrub or low foliage
#> 0 1 0 0
#> Aerial 0 0 7 5
#> Aquatic 0 0 1 0
#> Bark 10 1 2 0
#> Canopy 1 3 6 1
#> Ground 0 18 1 19
#> Shrub 0 3 1 24
#>
#> Subcanopy or canopy
#> 0
#> Aerial 3
#> Aquatic 0
#> Bark 0
#> Canopy 13
#> Ground 0
#> Shrub 2
```

You can save yourself some typing by using the with() command. If you read this outloud it would be “with the table dataframe, make a 2-way table from the group and foraging site columns”

```
with(martin1995, table(group, foraging.site))
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 10 1 0 0
#> ground-nesting 1 0 0 1 3 18 3
```

```
#> non-excavating 0 7 1 2 6 1 1
#> shrub or low foliage 0 5 0 0 1 19 24
#> Subcanopy or canopy 0 3 0 0 13 0 2
```

**149.0.2.1.0.3 T3)How do I make 3-way tables in R?** You can cross-classify categorical data as many ways as you'd like. Let's make a 3-way classification based on nest type, foraging and latitude.

Classify general latitudinal region as tropical, boreal, or temperate

```
martin1995$trop.or.temp <- "temperate"
i.tropics <- which(martin1995$lat < 23)
i.boreal <- which(martin1995$lat > 54)

martin1995$trop.or.temp[i.tropics] <- "tropics"
martin1995$trop.or.temp[i.boreal] <- "boreal"

table(martin1995$group, martin1995$foraging.site, martin1995$trop.or.temp)
#> , , = boreal
#>
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 0 0 0 0
#> ground-nesting 0 0 0 0 0 3 0
#> non-excavating 0 0 0 0 0 0 0
#> shrub or low foliage 0 0 0 0 0 0 0
#> Subcanopy or canopy 0 0 0 0 0 0 0
#>
#> , , = temperate
#>
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 10 1 0 0
#> ground-nesting 0 0 0 1 3 15 3
#> non-excavating 0 7 1 2 6 1 1
#> shrub or low foliage 0 5 0 0 1 19 24
#> Subcanopy or canopy 0 3 0 0 13 0 2
#>
#> , , = tropics
#>
#>
#> Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 0 0 0 0
#> ground-nesting 1 0 0 0 0 0 0
#> non-excavating 0 0 0 0 0 0 0
#> shrub or low foliage 0 0 0 0 0 0 0
```

```
#> Subcanopy or canopy 0 0 0 0 0 0 0 0
```

This can be simplified a bit using with()

```
with(martin1995, table(group, foraging.site, trop.or.temp))
#> , , trop.or.temp = boreal
#>
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 0 0 0 0
#> ground-nesting 0 0 0 0 0 3 0
#> non-excavating 0 0 0 0 0 0 0
#> shrub or low foliage 0 0 0 0 0 0 0
#> Subcanopy or canopy 0 0 0 0 0 0 0
#>
#> , , trop.or.temp = temperate
#>
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 10 1 0 0
#> ground-nesting 0 0 0 1 3 15 3
#> non-excavating 0 7 1 2 6 1 1
#> shrub or low foliage 0 5 0 0 1 19 24
#> Subcanopy or canopy 0 3 0 0 13 0 2
#>
#> , , trop.or.temp = tropics
#>
#> foraging.site
#> group Aerial Aquatic Bark Canopy Ground Shrub
#> excavating 0 0 0 0 0 0 0
#> ground-nesting 1 0 0 0 0 0 0
#> non-excavating 0 0 0 0 0 0 0
#> shrub or low foliage 0 0 0 0 0 0 0
#> Subcanopy or canopy 0 0 0 0 0 0 0
```



# **Chapter 150**

## **Undergraduate editors**

### **150.1 Winter 2021**

Aman Virmani AMV88@pitt.edu Noah Burget NGB23@pitt.edu Emily Jenison EMJ45@pitt.edu Havannah Tung LET61@pitt.edu Nitish?

### **150.2 Original students emailed**

"Burget, Noah G" <ngb23@pitt.edu>,

"Whitfield, Tyler S" Tsw32@pitt.edu, "Kalepalli, Nishita" nik78@pitt.edu, ARR130@pitt.edu, "Sheng, Tao" tao\_sheng@pitt.edu, "Shetty, Shriya S" SSS92@pitt.edu, "Cherupally, Vijay Kiran" VIC38@pitt.edu, "Cheng, Claire A" cac332@pitt.edu, "Murray, Anna M" Amm498@pitt.edu, "Virmani, Aman" amv88@pitt.edu, brd98@pitt.edu, "Varghese, Christopher G" cgv7@pitt.edu, "Morley, Lucas N" Lnm48@pitt.edu, "Prem, Michelle" mrp105@pitt.edu, "Fiore, Jack A" Jaf192@pitt.edu, "Shetty, Saumya S" sss93@pitt.edu, "Chamania, Nitish" nic64@pitt.edu, "Walbridge, Jordan H" jhw43@pitt.edu, "Hutchins, Theresa R" Trh53@pitt.edu, thk44@pitt.edu, "Reich, Isaac J" ijr11@pitt.edu, "Walsh, Jake" jaw312@pitt.edu, "Spellman, David A" das320@pitt.edu, "Ramaradj, Priyanka" prr44@pitt.edu, "Tung, Havannah" Let61@pitt.edu, "Katz, Tyler J" TJK68@pitt.edu, "Shi, Tingjia" tis46@pitt.edu, "Sharoubeem, Paula" Pas195@pitt.edu, "Khamar, Ronak K" rkk19@pitt.edu, "Wang, Li-Ting" liw89@pitt.edu, "Jenison, Emily Grace" emj45@pitt.edu, "Dowling, Rileigh" rid32@pitt.edu

### 150.3 Original email

A spreadsheet where you can sign up for tasks is here: <https://docs.google.com/spreadsheets/d/1MjUMNidAHfTThrF0p7MjOUysDC-LGAz3RYgbhJJWGA/edit?usp=sharing>

A book which I want to integrate into “Comp Bio For All!” but haven’t done any work on yet is <https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/>

I’d like to integrate the sections below into CB4ALL. The main task here is turning these webpages into .rmd files with the code in code chunks, section titles formatted, etc.

<https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#plotting-multivariate-data> <https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#calculating-summary-statistics-for-multivariate-data>  
<https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#calculating-correlations-for-multivariate-data> <https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#standardising-variables> <https://little-book-of-r-for-multivariate-analysis.readthedocs.io/en/latest/src/multivariateanalysis.html#principal-component-analysis>

I’ve worked on most of this book <https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/>

but there are several chapters I have not yet done anything with. These chapters need to be pasted into .rmd, code put into code chunks, etc. <https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/chapter10.html> [https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/reviewexercises\\_answers.html](https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/reviewexercises_answers.html) <https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/chapter11.html>

All graphs in the book are currently done with base R graphics, For any chapter, including the ones I’ve worked on you can re-work the graphs using ggpubr or ggplot2.

# Chapter 151

## TODO

<https://rdrr.io/cran/NameNeedle/man/needles.html>

<https://open.oregonstate.education/appliedbioinformatics/chapter/chapter-3/>

<https://farhanma.github.io/pyseq/>

<https://iubmb.onlinelibrary.wiley.com/doi/full/10.1002/bmb.21027>

<http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/pdfsides/07-needlemanwunsch-smithwaterman.pdf>

<http://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Needleman-Wunsch>

[https://wilkelab.org/classes/SDS348/2019\\_spring/labs/lab13-solution.html](https://wilkelab.org/classes/SDS348/2019_spring/labs/lab13-solution.html)

<https://www.biostars.org/p/146980/>

<https://gist.github.com/juliuskittler/ed53696ac1e590b413aac2dddf0457f6>

machine-learning using sequenceds  
<https://academic.oup.com/bioinformatics/article/34/14/2499/4924718>  
<https://academic.oup.com/bib/article/21/3/1047/5475015>  
<https://cran.r-project.org/web/packages/ftrCOOL/ftrCOOL.pdf>  
<https://ilearn.erc.monash.edu/>  
<https://www.pnas.org/content/92/19/8700>  
<https://academic.oup.com/bioinformatics/article/33/1/122/2525676>

<https://www.nature.com/articles/nrg3920>  
<https://academic.oup.com/bioinformatics/article/31/2/279/2365812?login=true>

1. Copy text and code from original website to .rmd files
2. Move any functions AC wrote to .R files
3. Move code into R code chunks
4. Compile list of packages used in the chapter; make sure these are loaded at beginning of chapter
5. Compile list of AC functions used in the chapter

6. Compile list of dad used in the chapter
7. Add AC functions to biodata package
8. Add dat to the biodata package
9. Make sure biodata gets called at beginning of chapter to load any ASC

# Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.21.