[1.21 R Training: Learn how to use R for statistical analyses 1](#_Toc63030812)

[Q1.txt 1](#_Toc63030813)

[Using R 2](#_Toc63030814)

[Theory 4](#_Toc63030815)

[Coding 5](#_Toc63030816)

[Q2.txt 5](#_Toc63030817)

[Q3.txt 12](#_Toc63030818)

[Q4.txt 15](#_Toc63030819)

[Q5.txt 18](#_Toc63030820)

[Q6.txt 19](#_Toc63030821)

[Q7.txt 21](#_Toc63030822)

[1.22 R Training: Learn how to use the BoutrosLab plotting general package 21](#_Toc63030823)

1.21 R Training: Learn how to use R for statistical analyses

**In bioinformatics it is often hard to identify an appropriate analytical statistical test. This may be true because assumptions are violated, or because no standard test exists. In either case, a common solution is to use permutation-based approaches, like a bootstrap.**

**The expectation for this training is that your learning be largely self directed (Google is your friend!) but be sure to ask for help when needed. There are no bad questions.**

**Answer the seven R statistics questions located on box**

**https://uclahs.box.com/s/ftb042ijrajsnt8obr24jb5b4uq9mr4p**

**Remember to create a script with the code you used to answer each question (i.e. Q1.R). Please use base R functions unless explicitly stated in the question.**

Q1.txt

**# install the necessary internal Boutros Lab packages from source.**

**install.packages("/Users/yupan/Downloads/new-members-documentation-training-and-guides/r-training/R-packages/BoutrosLab.utilities\_1.9.10.tar.gz", type = "source", dependencies = TRUE, repos = NULL);**

**install.packages("/Users/yupan/Downloads/new-members-documentation-training-and-guides/r-training/R-packages/BoutrosLab.statistics.general\_2.1.3.tar.gz", type = "source", dependencies = TRUE, repos = NULL);**

**# Install BoutrosLab.plotting.general from CRAN.**

**install.packages("BoutrosLab.plotting.general")**

**# Make sure that packages were installed properly**

**library("BoutrosLab.plotting.general")**

Using R

**=======**

**1. Read the file AHR-test-file.txt**

> df <- read.table("/Users/yupan/Downloads/new-members-documentation-training-and-guides/r-training/1. statistical-analysis/input-files/AHR-test-file.txt", header=TRUE)

> #get the first 10 rows

> df[1:10,]

       Control Treated

Mouse1    3.77    4.15

Mouse2    3.07    4.03

Mouse3    4.00    3.03

Mouse4    3.85    4.21

Mouse5    3.11    3.99

Mouse6    3.21    3.96

Mouse7    3.43    3.95

Mouse8    3.03    3.10

Mouse9    3.13      NA

**2. Perform a t-test between control and treated**

I'm not sure what "control" means in the data.

Since each mouse has "control" and "treatment", I assume it means each mouse was measured before and after the treatment. If this is the case, then I should perform a paired t-test.

Mouse 9 has "treated" as NA, so filter out this record.

> df.wo.na <- df[complete.cases(df),]

> df.wo.na

               Control Treated

Mouse1    3.77    4.15

Mouse2    3.07    4.03

Mouse3    4.00    3.03

Mouse4    3.85    4.21

Mouse5    3.11    3.99

Mouse6    3.21    3.96

Mouse7    3.43    3.95

Mouse8    3.03    3.10

t.test(x,

       y = NULL, #an optional (non-empty) numeric vector of data values.

       alternative = c("two.sided", "less", "greater"), #two.sided" default

       mu = 0,

       paired = FALSE,

       var.equal = FALSE,

       conf.level = 0.95,

       …)

# the other parameters, alternative, mu, var.equal, conf.level just take default values.

> res.ttest <- t.test(df.wo.na$Control,

                              df.wo.na$Treated,

                              paired=TRUE)

> res.ttest

Paired t-test

data:  df.wo.na$Control and df.wo.na$Treated

t = -1.6917, df = 7, p-value = 0.1345

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

 -0.8841939  0.1466939

sample estimates:

mean of the differences

               -0.36875

Because res$p.value equals to 0.13455 and conf.level takes the default 0.95, the null test cannot be rejected, namely, the difference is more likely due to randomness.

**3. Perform a wilcoxon test between control and treated**

> res.wilcox <- wilcox.test(df.wo.na$Control,

                                            df.wo.na$Treated,

                                            paired = TRUE)

> res.wilcox

Wilcoxon signed rank exact test

data:  df.wo.na$Control and df.wo.na$Treated

V = 8, p-value = 0.1953

alternative hypothesis: true location shift is not equal to 0

Again, p-value = 0.1953 means we cannot reject the null hypothesis.

**4. Calculate a fold-change between control and treated**

A fold change is basically a ratio. It indicates the number of times something has changed in comparison to an original amount.

> fold.change <- mean(df.wo.na$Treated)/mean(df.wo.na$Control)

[1] 1.10739

Another popular variant is log2 fold change.

> log2(mean(df.wo.na$Treated)/mean(df.wo.na$Control))

[1] 0.1471632

Theory

**======**

**1. Different type of t-tests**

There are three types of t-tests:

* One sample t-test.
* Paired sample t-test.
* Independent two-sample t-test
  + with equal variance
  + with unequal variance(Welch's t-test).

**2. Different two-sample tests**

* parametric
  + t-test: 1 categorical predictor, 1 quantitative outcome variable
  + ANOVA: 1 or more categorical predictor, 1 quantitative outcome variable. Usually used when there are +3 samples, but 2 samples can be used too.
* non-parametric
  + Wilcoxon Rank-Sum test(Mann Whitney U Test) for unpaired samples.
  + Wilcoxon Signed-rank test for paired samples.
  + Kolmogorov-Smirnov test (KS test) for unpaired samples.

**3. Which test to use when?**

If the data is parametric(normal),

    if two samples: use t-test. If two independent samples' variances are not equal, then use Welch's t-test, which adjusts DF. T-tests check if the difference of the means of two samples is significant.

    if more than two samples: use ANOVA, but need to make sure the variances of all groups are equal.

If the population is non-parametric, then use non-parametric tests, say Wilcoxon test, which sorts the data, then check the difference of the medians of two samples is significant. the Wilcoxon test is less sensitive to outliers because it looks at the median difference.

The advantage of using a parametric test instead of a nonparametric equivalent is that the former will have more statistical power than the latter. In other words, a parametric test is more able to lead to a rejection of H0. Most of the time, the p-value associated to a parametric test will be lower than the p-value associated to a nonparametric equivalent that is run on the same data.

Nonparametric tests are valid in a broader range of situations.

Coding

**======**

**1. Good file-opening semantics**

The R base function read.table() is a general function that can be used to read a file in table format. The data will be imported as a data frame.

Depending on the file formats, several variants of read.table() are available, including read.csv(), read.csv2(), read.delim() and read.delim2().

**2. R help**

We can use help(cmd), ?cmd, example(cmd) commands to find out the info of cmd.

More details can be found in <https://www.r-project.org/help.html>

**3. Parameterization in R functions**

We should use functions to modularize our codes, avoid hard-coded settings and consider reusability.

A good explanation is <https://www.r-bloggers.com/2016/07/good-parameterisation-in-r/>

Q2.txt

**In bioinformatics it is often hard to identify an appropriate analytical statistical test. This may be true because assumptions are violated, or because no standard test exists. In either case, a common solution is to use permutation-based approaches, like a bootstrap.**

**In this problem you will be comparing a number of different statistical approaches for comparing two groups.  The biological question here involves two tumour subtypes, A and B.  You have three samples of subtype A and nine samples of subtype B.  You have used a small microarray platform to measure the mRNA levels of 500 genes on each of these 12 tumours.  You wonder which genes differ between the two tumour subtypes.**

**Your first input data file (input1.txt) contains one column for the gene identifier and three columns of numeric data. Your second input data file (input2.txt) contains one column for the gene identifier and nine columns of numeric data. For both files, each column represents a biological sample (human tumour) and each row represents a gene.  The three numeric columns in input1.txt represent tumour-type A and the nine numeric columns in input2.txt represent tumour-type B.  In statistical terms, you need to determine if the three columns in input1.txt represent a random sample from the overall data for each gene.**

**Your first steps to answer this question are:**

**1. Read the two input files**

df1 <- read.table("/Users/yupan/Downloads/new-members-documentation-training-and-guides/r-training/1. statistical-analysis/input-files/input1.txt", header=TRUE)

df2 <- read.table("/Users/yupan/Downloads/new-members-documentation-training-and-guides/r-training/1. statistical-analysis/input-files/input2.txt", header=TRUE)

**2. Combine the two files into one file that contains the data for all 12 tumours. Make sure that the three columns in input1.txt precede the nine columns in input2.txt. Do this in the following two ways, and verify that they produce the same result:**

**a) Sort each file individually, and then use the cbind function**

**b) Use only the merge function**

Install dplyr, which includes a bunch of functions that make data frame operation easier.

> install.packages("dplyr")

> library(dplyr)

# The d is for dataframes, the plyr is to evoke pliers

# Sort both df1 and df2 by "GeneID" in asc order.

> df1.sorted <- df1[order(df1$GeneID),]

> df2.sorted <- df2[order(df2$GeneID),]

> nrow(df1.sorted)

[1] 500

> ncol(df1.sorted)

[1] 4

> nrow(df2.sorted)

[1] 500

> ncol(df2.sorted)

[1] 10

# Combine df1.sorted and df2.sorted, remove the duplicated "GeneID" in df2.sorted.

> df.combined.by.cbind <- cbind(df1.sorted[,1:4],df2.sorted[,2:10])

> summary(df.combined.by.cbind)

    GeneID             Patient1         Patient2         Patient3

 Length:500         Min.   : 2.814   Min.   : 3.044   Min.   : 3.016

 Class :character   1st Qu.: 4.802   1st Qu.: 4.823   1st Qu.: 4.768

 Mode  :character   Median : 5.563   Median : 5.615   Median : 5.566

                    Mean   : 5.544   Mean   : 5.546   Mean   : 5.484

                    3rd Qu.: 6.287   3rd Qu.: 6.251   3rd Qu.: 6.190

                    Max.   :10.983   Max.   :10.182   Max.   :10.203

    Patient4         Patient5         Patient6         Patient7

 Min.   : 2.907   Min.   : 2.902   Min.   : 2.867   Min.   : 2.863

 1st Qu.: 4.800   1st Qu.: 4.796   1st Qu.: 4.820   1st Qu.: 4.791

 Median : 5.635   Median : 5.626   Median : 5.583   Median : 5.657

 Mean   : 5.559   Mean   : 5.556   Mean   : 5.554   Mean   : 5.561

 3rd Qu.: 6.335   3rd Qu.: 6.267   3rd Qu.: 6.317   3rd Qu.: 6.262

 Max.   :10.333   Max.   :10.041   Max.   :10.654   Max.   :10.526

    Patient8         Patient9        Patient10        Patient11

 Min.   : 2.808   Min.   : 2.921   Min.   : 3.030   Min.   :2.886

 1st Qu.: 4.784   1st Qu.: 4.801   1st Qu.: 4.806   1st Qu.:4.784

 Median : 5.557   Median : 5.547   Median : 5.599   Median :5.592

 Mean   : 5.535   Mean   : 5.529   Mean   : 5.538   Mean   :5.541

 3rd Qu.: 6.272   3rd Qu.: 6.281   3rd Qu.: 6.272   3rd Qu.:6.268

 Max.   :10.721   Max.   :10.453   Max.   :10.209   Max.   :9.895

   Patient12

 Min.   : 2.945

 1st Qu.: 4.801

 Median : 5.625

 Mean   : 5.541

 3rd Qu.: 6.247

 Max.   :10.448

# The merge() function saves the troubles of sorting, joining.

> df.combined.by.merge <- merge(df1,df2)

# all\_equal() function proves that df.combined.by.merge, df.combined.by.cbind are same.

> all\_equal(df.combined.by.merge, df.combined.by.cbind)

[1] TRUE

**3. Perform a t-test comparing the first three tumours to the last nine tumours for \*each\* gene using a for-loop**

# Create a vector to store all the p\_values.

res.p.values <- c()

# Get the of the rows in the merged dataframe

count.df.combined.by.merge <- nrow(df.combined.by.merge)

# Loop through each row of df.combined.by.merge

for (i in 1:count.df.combined.by.merge) {

    # split each row into "input1" and "input2"

    input1 <- df.combined.by.merge[i, 2:4]

    input2 <- df.combined.by.merge[i, 5:13]

    # apply t.test to input1 and input2

    res.ttest <- t.test(input1, input2)

    # append the t.test's p.value into res.p.values

    res.p.values <-c(res.p.values, res.ttest$p.value)

}

# Create a vector to store all the p\_values.

res.p.values <- c()

# Loop through each row of df.combined.by.merge

for (i in 1:nrow(df.combined.by.merge)) {

    # split each row into "input1" and "input2"

    input1 <- df.combined.by.merge[i, 2:4]

    input2 <- df.combined.by.merge[i, 5:13]

    # apply wilcox.test to input1 and input2

    res.wilcox.test <- wilcox.test(unlist(input1),

    unlist(input2),

    mu=0,

    alt="two.sided",

    paired=F,

    conf.int=F,

    conf.level=0.95,

    exact=F

    )

    # append the t.test's p.value into res.p.values

    res.p.values <-c(res.p.values, res.wilcox.test$p.value)

}

**4. Plot a histogram of the p-values**

https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/hist

hist(res.p.values,

     main = "t-test p\_values - input1 vs input2",

     xlab = "p-values",

     breaks = seq(0,1, 0.01), # set bucket width as 0.01, since we are interested to check the distribution of p\_values (0.01, 0.05, 0.1). total 100 buckets.

     xaxp = c(0,1,20), # set tick width as 0.5, total 20 ticks

     ylim = c(0, 15)) # set the y limit to accommodate all the spikes.

Chart, bar chart, histogram

Description automatically generated

**5. Are your axis labels rotated 90 degrees?  If so, fix this.**

<https://www.tenderisthebyte.com/blog/2019/04/25/rotating-axis-labels-in-r/>

# remove y-axis

hist(res.p.values,

     main = "t-test p\_values - input1 vs input2",

     xlab = "p-values",

     breaks = seq(0,1, 0.01), # set bucket width as 0.01, since we are interested to check the distribution of p\_values (0.01, 0.05, 0.1)

     xaxp = c(0,1,20), # set ticker width as 0.5

     ylim = c(0, 15), # make sure the y limit have proper head room

     yaxt = "n")  # this removes y-axis

# redraw the y-axis

> axis(side = 2, las = 2, mgp = c(3, 0.75, 0))

The "side" parameter specifies the side of the chart on which to draw the axis, which takes an integer in {1,2,3,4}:

1: for below,

2: for left,

3: for above,

4: for right.

The "las" parameter defines the orientation of the tick mark labels, which takes an integer in {0,1,2,3}:

0: always parallel to the axis [default],

1: always horizontal,

2: always perpendicular to the axis,

3: always vertical.

Chart, histogram

Description automatically generated

**6. Your histogram might look a bit weird in normal space, consider plotting it in log-space**

**7. What does this distribution tell you?**

The p\_values are evenly distributed in the range (0,1).

It'd be interesting to look into those genes that gave p\_values < 0.01, also check the effect sizes, in order to find out the genes that differentiate tumour-type A and tumour-type B.

Q3.txt

**Okay, your next question might be if a t-test was inappropriate for this analysis.  Repeat the above comparison using a Wilcoxon test and fold-changes.  Create suitable plots to compare the results and write a paragraph describing the similarities/differences.**

res.p.values <- c()

# Loop through each row of df.combined.by.merge

for (i in 1:nrow(df.combined.by.merge)) {

    # split each row into "input1" and "input2"

    input1 <- df.combined.by.merge[i, 2:4]

    input2 <- df.combined.by.merge[i, 5:13]

    # apply wilcox.test to input1 and input2

    #wilcox.test takes vector as inputs

    res.wilcox.test <- wilcox.test(unlist(input1),

    unlist(input2),

    paired=FALSE,

    exact=FALSE

    )

    # append the t.test's p.value into res.p.values

    res.p.values <-c(res.p.values, res.wilcox.test$p.value)

}

hist(res.p.values,

     main = "Wilcoxon test p-values",

     xlab = "p-values",

     xaxp = c(0,1,20),

     breaks = seq(0,1,0.025),

     ylog = TRUE

)

Chart, histogram

Description automatically generated

fold.changes <- c()

# Loop through each row of df.combined.by.merge

for (i in 1:nrow(df.combined.by.merge)) {

    input1.mean <- mean(unlist(df.combined.by.merge[i, 2:4]))

    input2.mean <- mean(unlist(df.combined.by.merge[i, 5:13]))

    fold.change <- log2(input1.mean / input2.mean)

    fold.changes <- c(fold.changes, fold.change)

}

hist(fold.changes,

    main = "Histogram of Log2 fold changes",

    xlab = "Log2 fold changes",

    breaks = 100,

    ylog = TRUE

)

Chart, histogram

Description automatically generated

To this point you've been doing these analyses using a loop (for or while statements).  You'll next want to learn to use the R function "apply" to generate a vector of p-values (t- and u-tests) and of fold-changes.  Remember, the vectors should only contain p-values, not any other information.  You will need to figure out how to do this using apply, and this will require you to create what's called a "wrapper function".

t.test.p.values <- apply(df.combined.by.merge,

                                      1,

                                      function(x) {

                                          ttest.res <- t.test(as.numeric(x[2:4]),

                                                                     as.numeric(x[5:13]))

                                          ttest.res$p.value

                                      })

wilcox.test.p.values <- apply(df.combined.by.merge,

                                               1,

                                                function(x) {

                                                    utest.res = wilcox.test(as.numeric(x[2:4]),

                                                                                        as.numeric(x[5:13]),

                                                                                        exact=F)

                                                    utest.res$p.value

                                                })

# Use "apply" to calculate fold changes

fold.changes <- apply(df.combined.by.merge,

                                    1,

                                    function(x) mean(as.numeric(x[5:13]))/(mean(as.numeric(x[2:4]))))

Q4.txt

**There are a number of statistical challenges involved in executing so many statistical tests.  The primary one is called the "multiple testing" problem.  What is it?  Find R commands to adjust for multiple-testing using the two best-known adjustments (FDR and Bonferroni).  Recreate the histograms above.  What does this tell you?**

Suppose we randomly picked one guy from the street and let him predict the results of tossing a fair coin 20 times.

If he predicted correctly all 20 times, then he probably has some super power in prediction.

However, if we tried this for each person living on earth and found some people who predicted correctly all 20 times, then we cannot say these people have super power in prediction, because when we tried so many trials, some people would inevitably get lucky (70 billion \* (0.5^20) = 6676, namely, 6676 people could get lucky).

This happens in research tests too. If we try many tests, we will get multiple factors with statistical significances, however, some factors would get lucky as in the above coin-tossing case.

To avoid this, researchers use some methods to adjust the p-values of each test. The adjustment will make the p-values larger so that it is harder to reject H0.

Bonferroni correction (calculate FWER) is too stringent, which increases false negative rate. BHY correction (calculate FDR) is more commonly used.

t.test.p.values.fdr.adjusted <- p.adjust(t.test.p.values, method = 'fdr', n = length(t.test.p.values))

wilcox.test.p.values.fdr.adjusted <- p.adjust(wilcox.test.p.values, method = 'fdr', n = length(wilcox.test.p.values))

**Hopefully you have been saving your code in scripts as suggested in the bl-manulal. The code you create is an important resource to you and your colleagues and it is for this reason that we want to be able to track changes to this code and share it with others. One way to do this is via GitHub. Please read and follow the steps in the git/GitHub guide on Box https://uclahs.box.com/s/qv3olx8qnkuz4znl1rt6dkah4erbqce8**

**You will want to add your code and output from questions 1-4 to your local r-statistics folder within your <username>-training repository before pushing to Github.**

Q5.txt

**As noted above it is often hard to identify an appropriate analytical statistical test. This may be true because assumptions are violated, or because no standard test exists. In either case, a common solution is to use permutation-based approaches, like a bootstrap.  Your final task is to develop a bootstrap-like test in R.  This may be quite challenging at first.**

**1. Calculate the median of the first three columns for each gene**

**2. Use a permutation test to estimate the expected value for each gene:**

**a. Randomly select three columns from amongst all 12**

**b. Calculate their median**

**c. Determine if this value is larger or smaller than that of the first 3 columns**

**d. Repeat a.-c. 1000 times**

**3. Use the frequencies in 2. to estimate a p-value for each gene**

**4. Perform a false-discovery adjustment on the p-values (?p.adjust)**

**5. Write your results (gene ID, observed median, expected median, p-value, adjusted p-value) to file in a tab-delimited format**

**6. Plot a histogram of the (unadjusted) p-values. What does this tell you?**

Q6.txt

**To this point all the analyses have been performed using standard R functions.  Fortunately the BoutrosLab repository has a variety of functions that will aid in making the plots and statistics.  Modify your code to use:**

**a) BoutrosLab.statistics.general for all p-value extraction functions**

/Users/yupan/Downloads/new-members-documentation-training-and-guides/r-training/R-packages/BoutrosLab.statistics.general/man/get.ttest.p.Rd

says

\arguments{

  \item{x}{Vector of numbers to analyze}

  \item{group1}{Vector of TRUE/FALSE indicating which are the x samples for the t-test}

  \item{group2}{Vector of TRUE/FALSE indicating which are the y samples for the t-test}

  \item{paired}{Do a paired t-test?}

  \item{var.equal}{Assume Welch's correction?}

  \item{alternative}{What's the null-hypothesis?}

}

\value{No return value}

\author{Paul C. Boutros}

\examples{

get.ttest.p(

x = rnorm(100),

group1 = c(rep(TRUE, 50), rep(FALSE, 50)),

group2 = c(rep(FALSE, 50), rep(TRUE, 50)),

paired = FALSE,

var.equal = FALSE,

alternative = 'two.sided'

);

}

# Use BoutrosLab.statistics.general::get.ttest.p

t.test.p.values.BL <- apply(df.combined.by.merge[,2:13],

                                            1,

                                            function(x) BoutrosLab.statistics.general::get.ttest.p(

                                                 x,

                                                 group1 = c(1:3),

                                                 group2 = c(4:12)))

# Use BoutrosLab.statistics.general::get.utest.p

wilcox.test.p.values.BL <- apply(df.combined.by.merge[,2:13],

                                                   1,

                                                   function(x) BoutrosLab.statistics.general::get.utest.p(

                                                       x,

                                                       group1 = c(1:3),

                                                       group2 = c(4:12)))

# Use BoutrosLab.statistics.general::get.foldchange

fold.changes.BL <- apply(df.combined.by.merge[,2:13],

                                          1,

                                          function(x) BoutrosLab.statistics.general::get.foldchange(

                                               as.numeric(x),

                                               group1 = c(1:3),

                                               group2 = c(4:12)))

**b) BoutrosLab.plotting.general for all plots**

BoutrosLab.plotting.general::create.histogram(

    x = t.test.Boutros.p.values,

    main = "ttest p-values histogram by BoutrosLab.plotting.general::create.histogram",

    main.cex = 2,

    xlab.label = "p-values",

    type = "count",

    xlimits = c(0,1),

    ylimits = c(0, 15),

    breaks = seq(0, 1, 0.01),

    xat =  c(seq(0, 1, 0.1))

)

Chart, histogram

Description automatically generated

Make these modifications for all questions from Q2 onwards.

Q7.txt

**Lastly it's important (critical in fact!) to describe your statistical analysis accurately and precisely.  Write a publication-quality methods section describing what you've done.  Make sure you are appropriately referencing your methods with citations to the literature (specific reference format does not matter for this).**

**Make sure that you have added all your code and outputs from questions 1-7 to your GitHub training repository!**

1.22 R Training: Learn how to use the BoutrosLab plotting general package

**Why does data visualization matter? After all, data visualization takes work. Datasets are often complex, having millions of data points and many dimensions. Decisions have to be made regarding the best way of highlighting the main message. Aesthetic considerations require knowledge of colour theory, typography, composition, and more. Why do people go through all the trouble?**

**The reason is that it is often faster and easier for a person to process data visually. Noticing relationships between data points in a spreadsheet requires much more careful attention compared to viewing the same data points in a chart. Graphs can be used to first find trends in the data, and later to illustrate and highlight messages in data for communication to others.**

**• Refer to Chapter 7, Colour Guide, for best practices when creating plots.**

**• Answer the four R plotting questions on Box**

**https://uclahs.box.com/s/1be67l6kdcdwpbcdf0u9s28c2wxv79hw.**

**• As you did with the R statistics questions, push code to your training repository on GitHub**

**and add PCB as a reviewer.**

**INTRO: BoutrosLab.plotting.general**

**=======**

**This series of questions is designed to help teach the basics of using BoutrosLab.plotting.general.**

**1. Familiarize yourself with the functions available in BoutrosLab.plotting.general.**

**-These are easily viewable at** [**http://bpg.oicr.on.ca/API/BoutrosLab.plotting.general/5.3.4/index.html**](http://bpg.oicr.on.ca/API/BoutrosLab.plotting.general/5.3.4/index.html)

**-Note the scope and customizability of the different functions**

**-Think about what data-types would be most appropriate for the different plot-types**

Text

Description automatically generated

**SIMPLE PLOTS: BoutrosLab.plotting.general**

**=======**

**2a. Create a simple scatterplot using BoutrosLab.plotting.general.**

**-Use the 'cars' dataset provided the R Datasets package**

**-Set a title and x- and y-axis labels that make sense**

**-Ensure that the axes ranges make sense**

**-Fiddle with other parameters as needed to make the figure aesthetically pleasing (e.g. font size, line width, etc)**

scatterplot.mpg.displacement <- BoutrosLab.plotting.general:::create.scatterplot(data = mtcars,

   formula = disp ~ mpg,

   main = 'BoutrosLab.plotting.general:::create.scatterplot mtcars',

   main.cex = 2,

   xlab.label = 'MPG',

   xlimits = c(8.5, 37.5),

   xat =  seq(10, 35, 5),

   xlab.cex = 2,

   ylab.label = 'Displacement',

   ylimits = c(50, 500),

   ylab.cex = 2

  );

scatterplot.mpg.displacement

Chart, scatter chart

Description automatically generated

**2b. Try another function now. Create a heatmap displaying data found in the 'Loblolly' dataset found in the R Datasets package.**

**-Note: you may need to reorganize the data**

**-Do all formatting necessary to improve presentation (e.g. colour scheme, etc)**

<https://stat.ethz.ch/R-manual/R-patched/library/datasets/html/Loblolly.html>

The 'Loblolly' dataset contains

* height: a numeric vector of tree heights (ft).
* age: a numeric vector of tree ages (yr).
* Seed: an ordered factor indicating the seed source for the tree. The ordering is according to increasing maximum height.

str(Loblolly)

Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and 'data.frame': 84 obs. of  3 variables:

 $ height: num  4.51 10.89 28.72 41.74 52.7 ...

 $ age   : num  3 5 10 15 20 25 3 5 10 15 ...

 $ Seed  : Ord.factor w/ 14 levels "329"<"327"<"325"<..: 10 10 10 10 10 10 13 13 13 13 ...

Reorganize the data so that we form a matrix with age as x axis, height of each seed as y axis. Namely, we align the heights of seeds by ages.

Loblolly.reshaped <- reshape(Loblolly, idvar = 'age', timevar = 'Seed', direction = 'wide')

str(Loblolly.reshaped)

Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and 'data.frame': 6 obs. of  15 variables:

 $ age       : num  3 5 10 15 20 25

 $ height.301: num  4.51 10.89 28.72 41.74 52.7 ...

 $ height.303: num  4.55 10.92 29.07 42.83 53.88 ...

 $ height.305: num  4.79 11.37 30.21 44.4 55.82 ...

 $ height.307: num  3.91 9.48 25.66 39.07 50.78 ...

 $ height.309: num  4.81 11.2 28.66 41.66 53.31 ...

 $ height.311: num  3.88 9.4 25.99 39.55 51.46 ...

 $ height.315: num  4.32 10.43 27.16 40.85 51.33 ...

 $ height.319: num  4.57 10.57 27.9 41.13 52.43 ...

 $ height.321: num  3.77 9.03 25.45 38.98 49.76 ...

 $ height.323: num  4.33 10.79 28.97 42.44 53.17 ...

 $ height.325: num  4.38 10.48 27.93 40.2 50.06 ...

 $ height.327: num  4.12 9.92 26.54 37.82 48.43 ...

 $ height.329: num  3.93 9.34 26.08 37.79 48.31 ...

 $ height.331: num  3.46 9.05 25.85 39.15 49.12 ...

# Use BoutrosLab.plotting.general:::create.heatmap to create heatmap

BoutrosLab.plotting.general:::create.heatmap(

    # Remove the age column

    x = Loblolly.reshaped[,-1],

    main = 'Loblolly growth heatmap by seed and age',

    main.cex = 2,

    xlab.label = 'Age',

    xlab.cex = 2,

    xaxis.cex = 0.75,

    ylab.label = 'Seed',

    ylab.cex = 2,

    yaxis.cex = 0.75,

    # Define the y axis labels

    yaxis.lab = (c( substring(colnames(Loblolly.wide[,-1]), 8, 10))),

    # Define the x axis labels

    xaxis.lab = c(row.names(Loblolly.wide)),

    # Don't rotote the x axis labels

    xaxis.rot = 0,

    # Show the grids

    grid.row = TRUE,

    grid.col = TRUE,

    # Don't show the clustering lines

    clustering.method = 'none',

    # Don't show the key heatmap

    print.colour.key = FALSE,

)

A picture containing text, building, screenshot

Description automatically generated

**2c. Take a look at the 'ChickWeight' dataset in the R Datasets package**

**-Think about which plot-type should be used to effectively highlight the relationship between the growth of chicks and their diet.**

**-Create the plot using BoutrosLab.plotting.general functions**

**-Note that you have many options here, including plotting all the data points, aggregating the data, controlling for certain factors, etc.**

<https://rpubs.com/YaRrr/WPA0Answers>

> str(ChickWeight)

Classes ‘nfnGroupedData’, ‘nfGroupedData’, ‘groupedData’ and 'data.frame': 578 obs. of  4 variables:

 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...

 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...

 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 15 15 15 15 ...

 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...

 - attr(\*, "formula")=Class 'formula'  language weight ~ Time | Chick

  .. ..- attr(\*, ".Environment")=<environment: R\_EmptyEnv>

 - attr(\*, "outer")=Class 'formula'  language ~Diet

  .. ..- attr(\*, ".Environment")=<environment: R\_EmptyEnv>

 - attr(\*, "labels")=List of 2

  ..$ x: chr "Time"

  ..$ y: chr "Body weight"

 - attr(\*, "units")=List of 2

  ..$ x: chr "(days)"

  ..$ y: chr "(gm)"

> head(ChickWeight)

  weight Time Chick Diet

1     42    0     1    1

2     51    2     1    1

3     59    4     1    1

4     64    6     1    1

5     76    8     1    1

6     93   10     1    1

We can create a histogram of weights for each diet

par(mfrow = c(2, 2))

for (diet.i in 1:4) {

    hist(x = ChickWeight$weight[ChickWeight$Diet == diet.i],

        xlab = "weights",

        xlim = c(0, 400),

        main = paste("Chick Weights\nDiet ", diet.i, sep = ""))

}

Diagram

Description automatically generated with low confidence

pirateplot (violin plot) reveals the relation more clearly.

> install.packages("yarrr")

> library(yarrr)

> yarrr.guide()

>  pirateplot(formula = weight ~ Diet,

           data = ChickWeight,

           main = "Chicken weights by Diet ")

Chart

Description automatically generated

Diet3 yields most weight spread.

Diet4 yields most chicken of similar weight (around 140).

COMPLEX PLOTS: BoutrosLab.plotting.general

=======

3.

- Take a look at "Q3\_SampleOutput.tiff"

- This is the figure you will be re-creating using BoutrosLab.plotting.general functions.

- The data used to generate the figure is found in the "Q3\_SeqControl\_data.tsv" file

======= BACKGROUND INFO =======

The first thing you need to do is understand the figure. This is a figure used to describe the performance of SeqControl.

SeqControl is a framework used to make predictions in sequencing data quality. For example, a researcher may want to conduct an experiment which requires sequencing data. The researcher may begin by generating a subset of the data, and measuring metrics of the data. These metrics can be combined into a statistical model to predict features of the experiment, in order to improve the choices made when sequencing the rest of the data required.

An overview figure "SeqControl\_Overview.tff" can be found in the outputs folder

The statistical model referred to in this figure is called a random forest. Essentially, this model makes a prediction regarding the quality of sequencing data that will be generated. The fraction of 'yes' vote indicates the confidence that six lanes of sequencing will achieve 50x coverage.

The actual observed coverage of the data is shown by the colour of the bar, with black indicating high coverage, and grey indicating lower than 50x coverage.

There are also five covariate bars beneath the barplot.

The first covariate bar indicates the tumour sample from which the sequencing data came from. The second covariate bar indicates how the tumour sample was prepared - either FFPE or frozen. The remaining three covariates indicate some metric related to the data:

-% Bases > 0

-Unique start points

-Average reads/start

======= UNDERSTANDING THE DATA =======

The data for this plot has been organized into a tab-delimited file. The header of the file indicates what kind of data is stored in each column.

======= PLOTTING =======

Overview: The main plot in this figure is made using create.barplot. The covariates are made using create.heatmap. The legend is created using legend.grob. All of these elements are combined together using create.multipanelplot.

Step 1. You will need to read in the data and do some reformatting. If you look at the final plot, the "yes votes" are ordered in decreasing order. This is not how they are ordered in the data file. You will need to reorder the data after you read it in (don't make changes to the original data file).

Step 2. Create a heatmap displaying the cpcgene sample names. The colour scheme to use for this is given in the "Q3\_PlottingInfo.R" file. (Note that create.multipanelplot can be used to adjust plot proportions at the end).

Step 3. Create heatmaps for each of the three metrics. If you read the header names, "Average.reads.start", "Unique.start.points", and "X..Bases...0.quality" are the columns you are looking for. The colour schemes for these range from white to either "deeppink", "darkblue", or "darkorange" (these are R colour names). Note that you will have to show the metric values on a continuous scale.

Step 4. For the FFPE covariate bar, you'll need to know that the samples which are FFPE are samples "CPCG0102P" and "CPCG0103P". Also, the colour scheme consists of "white" and "darkslategrey".

Step 5. Plot the barplot. The colour scheme consists of "black" and "grey". The "outcome" column in the data file tells you what the observed coverage was for a sample. An outcome of 0 indicates coverage <50x, and 1 indicates coverage >=50x.

Step 6. Create a legend for each of the covariates. Ensure that the legend displays either continuous or discrete schemes as appropriate. The individual legends can be combined using the legend.grob function. The labels for "unique start points" can be formatted using the scientific.notation function.

Step 7. Create a legend for the barplot. This should be created separately for ease of placement in the final figure.

Step 8. Combine all of the plots and legends together using create.multipanelplot.

\*\*\* Bonus marks if your code complies with BoutrosLab coding standard. :)

WORKING WITH LARGE DATASETS: BoutrosLab.plotting.general

=======

4.

- Take a look at "Q4\_SampleOutput.tiff"

- Re-create this figure using BoutrosLab.plotting.general functions.

- The data file you will use is named "Q4\_HetStudy\_data.txt"

Understanding the figure: (more information on the project can be found by reading the paper https://www.nature.com/articles/ng.3315)

- The fraction plot at the top of the figure indicates the proportion of mutations per bin. (Note: your results will not look identical to the plot provided).

- The literature covariate bars labelled "Baca", "Berger", and "Weischenfeldt" are studies. The colours indicate whether or not a mutation in that window of the genome was seen in that study.

- The large plot of small coloured bars indicates locations in chromosomes (x-axis) in different samples (y-axis) of mutations. CTX = chromosomal translocation, ITX = intrachrosomosomal translocation, INV = inversion.

- The right-side covariates identify clinical qualities of the samples. The first covariate indicates which patient the samples come from. The second covariate indicates whether the sample was collected by surgery or biopsy. The third covariate indicates the Gleason score of the sample. The fourth covariate indicates how the tissue samples were preserved.

Understanding the data:

- The patient IDs are identified in the column headers.

- The cohorts are such that samples "CPCG0001", "CPCG0003", "CPCG0006", "CPCG0020", and "CPCG0042" are Bx (biopsy), and all others are Sx (surgery)

- For tissue type, all "F0" samples are Frozen, while all other samples are FPPE

- The Gleason score can be matched to the sample as shown in the sample output plot

- The literature covariate data (Baca, Berger, and Weischenfeldt) are listed in the last three columns of the data file.

- The fraction plot at the top is found by the proportion of mutations per patient

- The notation for None, CTX, ITX and INV is determined by the numeric coding of 0, 1, 2, 3.

Colour schemes used:

-main plot: c('white', 'cornflowerblue','darkolivegreen4','darkred')

-Cohort: c('royalblue', 'pink')

-Gleason score: c('3+4' = 'yellow', '4+3' = 'orange','4+4' = 'red')

-Tissue type: c('Frozen' = colours()[532],'FFPE' = colours()[557])

-Patient ID: c(

'CPCG0001' = 'blue',

'CPCG0003' = 'purple',

'CPCG0006' = 'green',

'CPCG0020' = 'orange',

'CPCG0042' = 'yellow',

'CPCG0099' = 'black',

'CPCG0102' = 'wheat4',

'CPCG0103' = 'green4',

'CPCG0183' = 'grey',

'CPCG0184' = 'red4')

-Literature: you may choose your own appropriate colour scheme

Sample Order for plotting:

sample.order <- c(

        "CPCG0001F0",

        "CPCG0003F0",

        "CPCG0006F0",

        "CPCG0020F0",

        "CPCG0042F0",

        "CPCG0099F0",

        "CPCG0099P1",

        "CPCG0102F0",

        "CPCG0102P1",

        "CPCG0102P2",

        "CPCG0103P7",

        "CPCG0103F0",

        "CPCG0103P2",

        "CPCG0103P1",

        "CPCG0103P4",

        "CPCG0103P3",

        "CPCG0103P8",

        "CPCG0103P5",

        "CPCG0103P6",

        "CPCG0183F0",

        "CPCG0183P2",

        "CPCG0183P1",

        "CPCG0183P3",

        "CPCG0184P3",

        "CPCG0184P1",

        "CPCG0184F0",

        "CPCG0184P2",

        "CPCG0184P4"

        );