Boutros.Lab statistical tests and plotting in R

UIJEONG LEE

08. 14. 2020

Introduction

Gene expression is the measurement of the amount of mRNA in cells, and microarray analysis is the most famous and widely used for discovering differentially expressed genes. In microarray data, a row represents the expression amount of each gene, and a column shows the expression amount in one sample. In this study, we will understand the microarray data and handle it in R programming. It will also be tried to find out the correlation between subtype A and B of tumor in each gene using several ways including parametric tests, non-parametric tests, permutation-based approaches, etc.

Method and Result

Before performing plotting, let's get the basic R skills such as reading a file and calling base functions for fundamental statistical tests.

Q1: Basic R skills

Q1.1. Read the file AHR-test-file.txt

To read files, many formats are used such as read.csv(), read.table(), readxl::read_excel(), read.delim(), etc.

> read.table(file, haeder, sep, na.strings, stringsAsFactors)

- file : the name of the file which the data are to be read from. Each row of the table appears as one line of the file.
- header=TRUE the file contains the names of the variables as its first line
- sep='\t' the field separator character
- na.strings ="NA" a character vector of strings which are to be interpreted as NA values.
- stringsAsFactors= default.stringsAsFactors() by default, 'stringsAsFactors' is set to TRUE, and character vectors be converted to factors.

```
> AHRtest <- read.table(file = "AHR-test-file.txt",
                        header = TRUE,
                        sep = "\t",
                        na.strings = "NA",
                        stringsAsFactors = default.stringsAsFactors())
```

```
> AHRtest
       X Control Treated
1 Mouse1    3.77    4.15
2 Mouse2    3.07    4.03
3 Mouse3    4.00    3.03
4 Mouse4    3.85    4.21
5 Mouse5    3.11    3.99
6 Mouse6    3.21    3.96
7 Mouse7    3.43    3.95
8 Mouse8    3.03    3.10
9 Mouse9    3.13      NA
```

Q1.2. Perform a t-test between control and treated

In statistical analysis, it is important to distinguish between parametric tests, and non-parametric tests, or permutation tests. Parametric tests have a higher power if the underlying model is normally distributed. On the other hand, non-parametric tests have less stringent assumptions on the data distribution. The p-value is to test whether there is a statistically significant difference in the average of two independent populations; if the normality assumption is satisfied through var.test(), the independent sample T-test is used. The test statistics for the T test are computed based on the "Student T" distribution. In this practice, A student's t-test is performed with two independent samples (Control and Treated).

> t.test(group1, group2, alternative, conf, var.equal, paired)

- alternative="two.sided" is the alternative hypothesis and refers to the two sided t test. Allowed value is one of "two.sided" (default), "greater" or "less".
- conf=0.95 is the confidence interval.
- var.equal = FALSE since the variances are not equal. Although note that it is hard to gage variances with so few points and without knowledge if it's a normal distribution.
- paired= FALSE because the data points are not related to each other.

> var.test(AHR.test$Control, AHR.test$Treated, na.rm=TRUE)$p.value

```
[1] 0.5710858
```

> t.test(AHR.test$Control, AHR.test$Treated, alternative="two.sided", conf=0.95, var.equal=FALSE, paired=FALSE)

```
        Welch Two Sample t-test

data:  AHR.test$Control and AHR.test$Treated
t = -1.9456, df = 13.543, p-value = 0.07275
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.84760609  0.04260609
sample estimates:
mean of x mean of y
   3.4000    3.8025
```

The p-value of the test is 0.07275, which is higher than the significance level alpha = 0.05. We can conclude that the null hypothesis is accepted at the 95% interval and the Control is

not different from the Treated with a p-value = 0.07275. It is similar at the 95% confidence interval level.

Q1.2. Perform a wilcoxon test between control and treated

If the normality assumptions of the two groups are not satisfied, the nonparametric test, such as the Wilcoxon test, is used. The parameters 'alternative', 'paired', etc are used as the t.test above. 'exact' means whether an exact p-value should be computed.

```
> wilcox.test(group1, group2, alternative, var.equal, paired, exact)
```

```
> wilcox.test(AHR.test$Control, AHR.test$Treated, alternative="two.sided", paired=FALSE, conf.level=0.95, exact=FALSE)
```

```
        Wilcoxon rank sum test with continuity correction

data:  AHR.test$Control and AHR.test$Treated
W = 18.5, p-value = 0.1017
alternative hypothesis: true location shift is not equal to 0
```

The p-value of the test is 0.1017, which is higher than the significance level alpha = 0.05. However, the p-value of the Wilcoxon rank sum test is different from the student's t test since the student's t-test examines the difference of the mean between pairs and the Wilcoxon test looks at the median difference.

Q1.3. Calculate a fold-change between control and treated

Fold change is calculated simply as the ratio of the difference between group2 and group1 over the original value. A simple fold change is calculated by the mean of the treated compared to the mean of the control. NAs are removed by na.omit() for having the accurate result, which is crucial when performing statistical tests.

```
> fold.change <- log2(mean(na.omit(AHR.test$Treated))/mean(na.omit(AHR.test$Control)))
```

```
> fold.change
[1] 0.1614135
```

Q2: Intro to file merging, for-loops, and plotting in R

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.

Q2.1. Read the two input files

Any parameters are set as arguments to the function. The code wraps all of the specified arguments into a function.

```
> function(input variables){
#some code
#return output variables
return(output variables);
 }
```

```
> read.txt <- function(x, sep){
  input <- read.table(x,
                      header=TRUE,
                      sep=sep,
                      na.strings = "NA",
                      stringsAsFactors = default.stringsAsFactors());
}
> input1 <- read.txt("input1.txt","\t")
> input2 <- read.txt("input2.txt","\t")
```

```
> head(input1)
    GeneID Patient1 Patient2 Patient3
1    10_at 5.826513 5.812383 5.962752
2   100_at 4.849585 4.410785 4.964242
3  1000_at 5.488923 5.765557 5.394642
4 10000_at 5.935700 6.181357 6.160881
5 10001_at 5.868707 5.600402 5.121879
6 10002_at 6.382565 6.521967 6.611568
> head(input2)
    GeneID Patient4 Patient5 Patient6 Patient7 Patient8 Patient9 Patient10 Patient11 Patient12
1 10467_at 2.906744 2.902403 2.876276 2.863292 2.992057 3.019094  3.029536  2.885635  3.169263
2 10376_at 3.024524 3.247500 2.867480 3.040735 3.024020 3.040650  3.353770  3.084541  3.092607
3 10427_at 3.286163 3.192643 3.600886 3.091638 3.286230 3.363795  3.274333  3.193151  2.944669
4 10054_at 3.300220 3.017577 3.000097 3.303757 2.807682 2.920627  3.511917  3.077447  3.178701
5 10520_at 3.608240 3.280419 3.578523 3.355977 3.735443 3.852667  3.696070  3.784540  3.571498
6 10199_at 3.878531 3.449912 3.457973 3.366949 3.588549 3.543370  3.451776  3.681291  3.447575
```

Q2.2. Combine and Sort the two files into one file

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:

Q2.2.1. Sort each file individually, and then use the cbind function

```
#splitting the character and arranging based on GeneID
> data.sort <- function(x, split, sortID=GeneID){
        tmp.data <- t(matrix(
```

```
        unlist(strsplit(as.character(x$GeneID),
          split=split))));
      geneid <- order(as.numeric(tmp.data));
      x.sort <- x[geneid,];
      return(x.sort);
}
> input1.sort <- data.sort(x=input1, split="_at")
> input2.sort <- data.sort(x=input2, split="_at")

> input.total <- cbind(input1.sort[,1:4], input2.sort[2:10])
```

```
> head(input.total,25)
     GeneID Patient1 Patient2 Patient3 Patient4 Patient5 Patient6 Patient7 Patient8 Patient9 Patient10 Patient11
1     10_at 5.826513 5.812383 5.962752 5.425051 5.715586 5.550814 5.543699 5.688899 5.539306  5.999191  5.880362
2    100_at 4.849585 4.410785 4.964242 4.700009 4.793152 4.623870 4.705424 4.728206 4.854953  4.850722  4.940739
130  101_at 4.360164 4.312885 4.269303 4.214558 4.444963 4.270195 4.149806 4.168928 4.304143  4.334573  4.325074
212  102_at 4.421065 4.297603 4.344519 4.379882 4.257506 4.255225 4.311998 4.458278 4.381143  4.465407  4.514826
308  103_at 5.838492 6.216083 5.899785 6.004787 5.709706 6.150187 5.543895 6.052662 5.677693  5.676091  6.077499
378  104_at 5.387028 5.328917 5.170283 5.212654 5.277789 5.376813 5.453342 5.279830 5.249571  5.182742  5.338926
467  105_at 5.285845 5.497362 5.191306 5.285948 5.390995 4.998524 5.140843 5.043365 5.263119  4.974324  5.499289
3   1000_at 5.488923 5.765557 5.394642 5.800958 5.710882 6.024407 5.834485 5.633735 5.739122  5.805569  5.804742
```

Q2.2.2. Use only the merge function

```
> input.all.merge <- merge(input1.sort, input2.sort, key="GeneID")
#sorting the merged file based on GeneID
> input.all.sort <- data.sort(x=input.all.merge, split="_at")
```

```
> head(input.all.sort,25)
     GeneID Patient1 Patient2 Patient3 Patient4 Patient5 Patient6 Patient7 Patient8 Patient9 Patient10 Patient11
1     10_at 5.826513 5.812383 5.962752 5.425051 5.715586 5.550814 5.543699 5.688899 5.539306  5.999191  5.880362
2    100_at 4.849585 4.410785 4.964242 4.700009 4.793152 4.623870 4.705424 4.728206 4.854953  4.850722  4.940739
130  101_at 4.360164 4.312885 4.269303 4.214558 4.444963 4.270195 4.149806 4.168928 4.304143  4.334573  4.325074
212  102_at 4.421065 4.297603 4.344519 4.379882 4.257506 4.255225 4.311998 4.458278 4.381143  4.465407  4.514826
308  103_at 5.838492 6.216083 5.899785 6.004787 5.709706 6.150187 5.543895 6.052662 5.677693  5.676091  6.077499
378  104_at 5.387028 5.328917 5.170283 5.212654 5.277789 5.376813 5.453342 5.279830 5.249571  5.182742  5.338926
467  105_at 5.285845 5.497362 5.191306 5.285948 5.390995 4.998524 5.140843 5.043365 5.263119  4.974324  5.499289
3   1000_at 5.488923 5.765557 5.394642 5.800958 5.710882 6.024407 5.834485 5.633735 5.739122  5.805569  5.804742
```

#checking the NA in the content

Using the is.na() function can find missing values. This function returns the value of true and false for each value in a data set.

```
> sum(is.na(input.all.sort))
```

```
> sum(is.na(input.all.sort))
[1] 0
```

Q2.3. Perform a t-test comparing the first three tumours to the last nine tumours for *each* gene using a for-loop

A for-loop allows for the function placed within the loop to be repeated a specific number of times, as defined by the programmer.

```
> for (index in start.value : end.value) {
statements;
```

```
}

> gene.p.value <- NULL

> for ( i in 1:nrow(input.all.sort)) {
    x <- t.test(
      as.vector(input.all.sort[i,c(2:4)]),
      as.vector(input.all.sort[i,c(5:13)]),
      alternative="two.sided",
      paired = FALSE,
      conf.int = TRUE,
      conf.level = 0.95,
      exact = TRUE
      )$p.value;
    genename <- as.vector(input.all.sort[i,1]);
    gene.p.value <- rbind(gene.p.value, c(genename, x)
  );
}

#changing to the dataframe format
> gene.p.value <- data.frame(
  Gene_ID = gene.p.value[,1],
  P.value = as.numeric(gene.p.value[,2]))
```

```
> head(gene.p.value)
  GeneID    P.value
1  10_at 0.03013882
2 100_at 0.89591074
3 101_at 0.33285626
4 102_at 0.95639483
5 103_at 0.46861693
6 104_at 0.85592425
```

Q2.4. Plot a histogram of the p-values

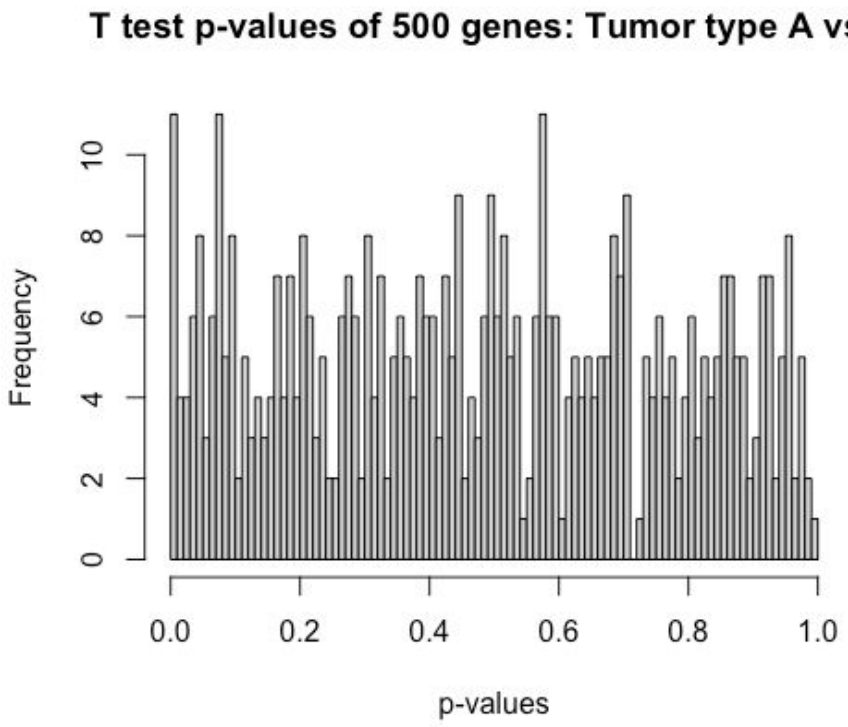
Histogram can be created using the hist() function in R. This function takes in a vector of values.

```
> hist(x, main,xlab,xlim,ylim,breaks)
```

- main indicates the title of the chart.
- xlab indicates the title of the x axis.
- xlim is used to specify the range of values on the x-axis.
- ylim is used to specify the range of values on the y-axis.
- breaks :a vector giving the breakpoints between histogram cells,
- freq : if TRUE, the histogram graphic is a representation of frequencies, the counts component of the result; if FALSE, probability densities, component density, are plotted.

```
#plotting histograms using hist()
> hist(gene.p.value$P.value,
```

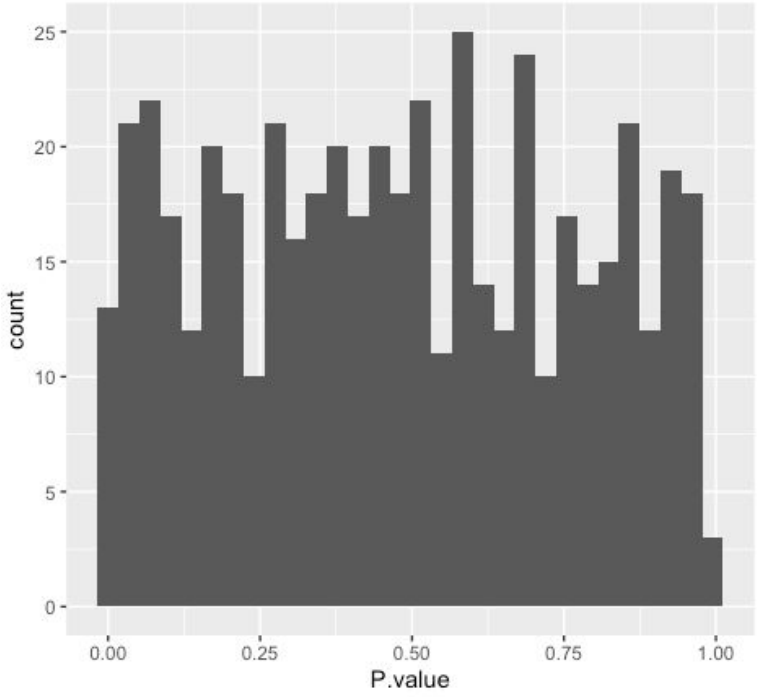
```
main = "T test p-values of 500 genes: Tumor type A vs B",
freq = TRUE,
xlab = "p-values",
breaks = seq(0,1, 0.01))
```



#plotting histograms using ggplot()

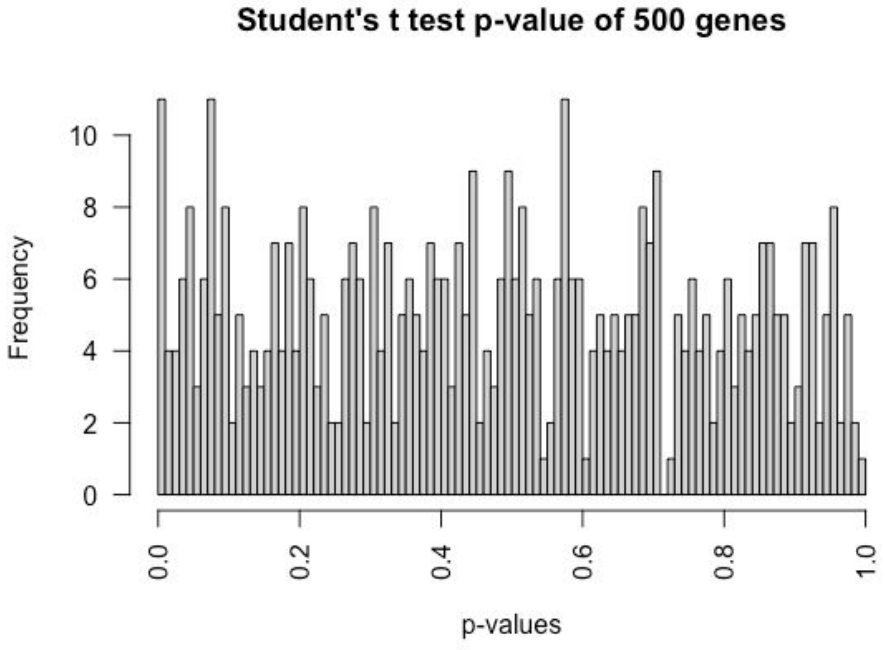
ggplot2 package creates a number of graphs we need. All ggplot2 plots begin with ggplot(), supplying data and aesthetic mappings, aes(), and then add layers, scales, coords and facets with + geom_ function.

```
> install.packages("ggplot2")
> library(ggplot2)
> ggplot(data=gene.p.value,
      aes(x=P.value)) + geom_histogram()
```

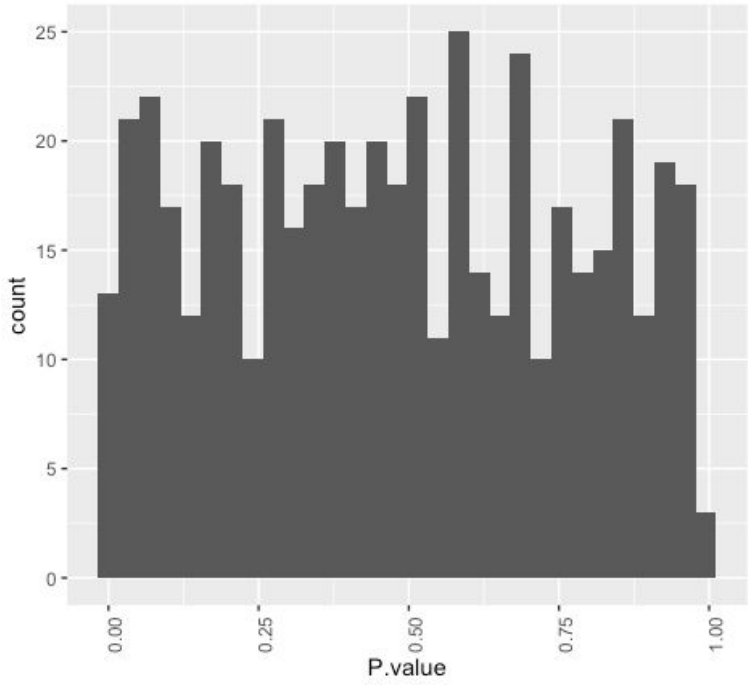


Q2.5. Are your axis labels rotated 90 degrees? If so, fix this.

```
#plotting histograms using hist()
> hist(gene.p.value$P.value,
    main = "Student's t test p-value of 500 genes",
    freq = TRUE,
    xlab = "p-values",
    breaks = seq(0,1, 0.01),
    las = 2
    )
```

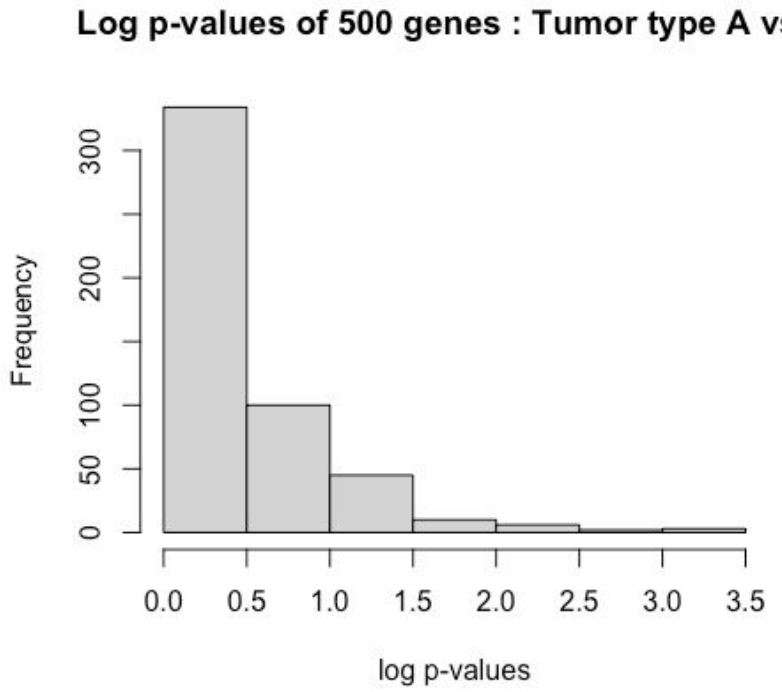


```
#plotting histograms using ggplot()
> ggplot(data = gene.p.value, aes(x = P.value)) + geom_histogram() +
  theme(axis.text.x=element_text(angle = 90))
```



Q2.6. Your histogram might look a bit weird in normal space, consider plotting it in log-space

```
> log <- -log(as.numeric(gene.p.value[,2]),base=10)
> log.gene <- cbind('Gene_ID'=gene.p.value[,1],'log.p.value'=log)
> log.p.value <- data.frame(Gene_ID = log.gene[,1], Log.p.value = as.numeric(log.gene[,2]))
> hist(log.p.value$Log.p.value)
```



The p-value from the t test is the method of obtaining the probability whether the null hypothesis that the experiment is not different with the comparison is rejected. In this project, it is shown the distribution of the variation of p-values comparing 500 genes between tumor subtypes of A and B using the student's t test. In the histogram, the distribution of frequency of p.value is fairly evenly distributed from 0 to 1. It means that this is not normally distributed. This rejects the null hypothesis(a variable is normally distributed in some population) and follows the alternative hypothesis. It could be concluded that it is needed to conduct non-parametric tests such as a Wilcoxon test.

Q3: Developing plottings and the apply( ) function

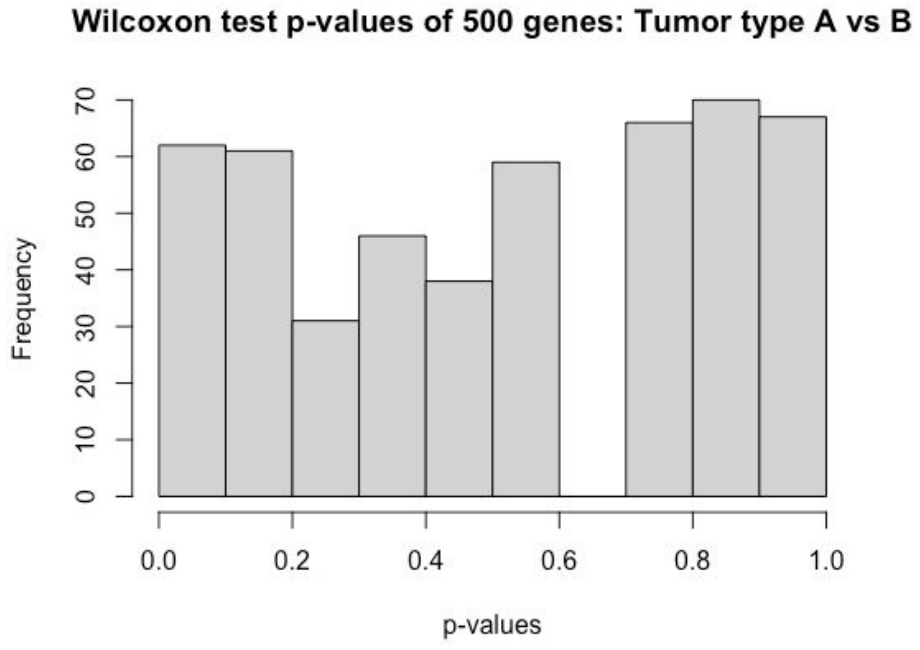
Q3.1. Your next question might be if a t-test was inappropriate for this analysis. Repeat the above comparison using a Wilcoxon test and fold-changes.

```
> wilcox.fold <- NULL

> for ( i in 1:nrow(input.all.sort)) {
#getting p-value using the Wilcoxon test
  x <- wilcox.test(
    as.numeric(input.all.sort[i,c(2:4)]),
    as.numeric(input.all.sort[i,c(5:13)]),
    alternative="two.sided",
    paired = FALSE,
    conf.int = FALSE,
    conf.level = 0.95,
```

```
    exact = FALSE
    )$p.value;
  genename <- as.vector(input.all.sort[i,1]);
#getting the value of log2 fold change
  fold.changes <- as.numeric(mean(unlist(input.all.sort[i,c(2:4)]))) /
as.numeric(mean(unlist(input.all.sort[i,c(5:13)])))
  fold.changes <- log2(unlist(fold.changes))
#combining the p-value and the fold change
  wilcox.fold <- rbind(wilcox.fold, c(genename, x, fold.changes)
  );
}

#changing to the dataframe format
> wilcox.fold <- data.frame(Gene_ID = wilcox.fold[,1], wilcox.p.value =
as.numeric(wilcox.fold[,2]), fold.change = as.numeric(wilcox.fold[,3]))
```

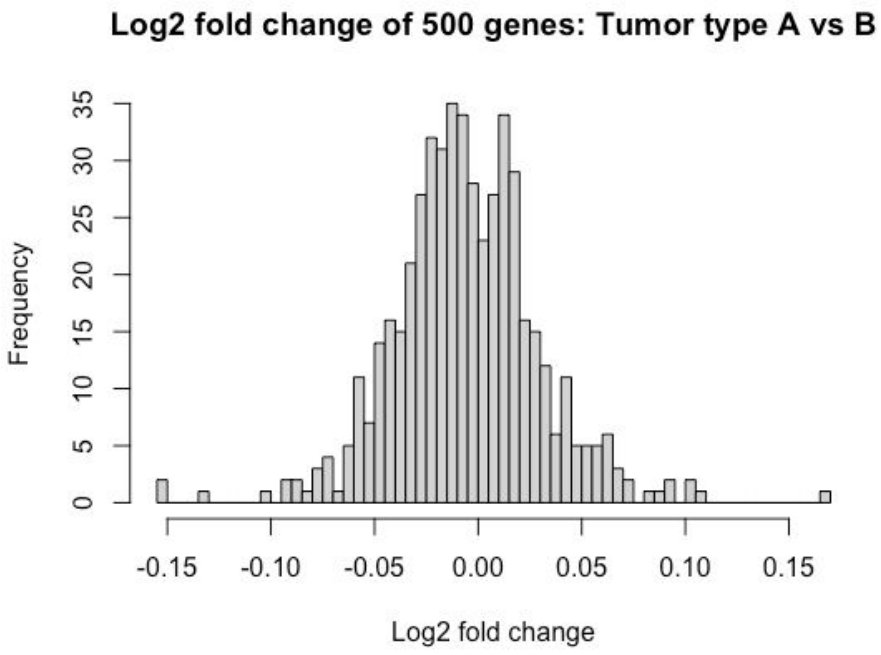
```
> head(wilcox.fold)
  Gene_ID wilcox.p.value   fold.change
1   10_at       0.1390867  0.0504886254
2  100_at       0.8533074 -0.0076552979
3  101_at       0.4595426  0.0140144777
4  102_at       1.0000000 -0.0009685211
5  103_at       0.5790997  0.0270797933
6  104_at       0.8533074 -0.0038603777
```

```
#plotting the histogram of the wilcox.p.value
> hist(wilcox.fold$wilcox.p.value,
    main = "Wilcoxon test p-values of 500 genes: Tumor type A vs B",
    freq = TRUE,
    xlab = "p-values",
    breaks = seq(0,1, 0.1))
```



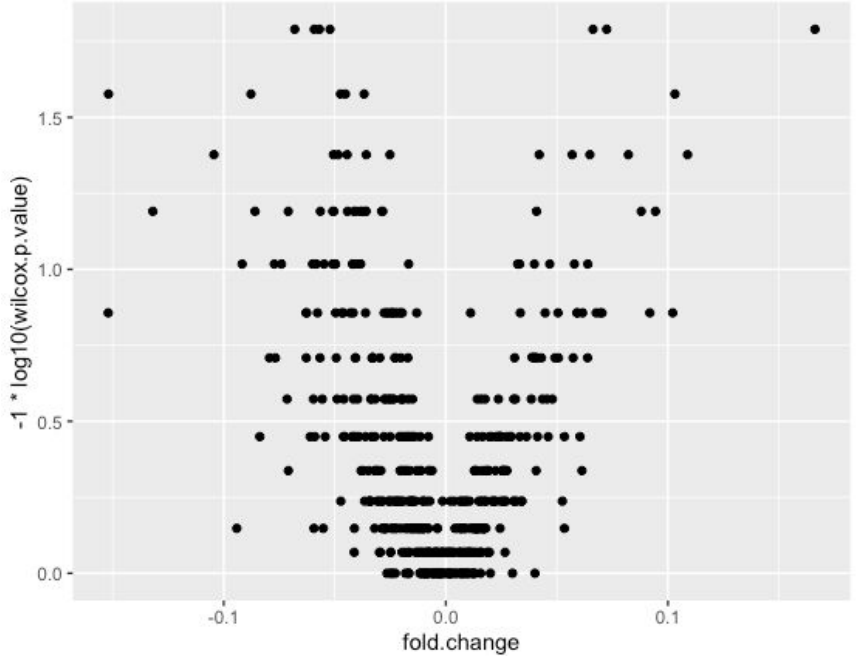
#plotting the histogram of the fold change

```
> hist(wilcox.fold$fold.change,
    main = "Log2 fold change of 500 genes: Tumor type A vs B",
    freq = TRUE,
    xlab = "Log2 fold change",
    breaks = 100)
```



#plotting the volcano plot of the fold change and p.value

```
> library(ggplot2)
volcano = ggplot(data = wilcox.fold, aes(x = fold.change, y = -1*log10(wilcox.p.value)))
volcano + geom_point()
```



There are the results between group1[first three column] and group2[last nine column] obtained by t-test, Wilcoxon test and fold change for each gene. The student's t-test tends to have more power with a larger population. However, the population in 12 samples is not enough to make this data meaningful for t test, and thus it makes sense to try a non parametric test like the Wilcoxon. The Wilcoxon test is less sensitive to the population since it looks at the median difference. However, With the graph of the Wilcoxon test, there are many p-values over 0.05 and irregularity spreaded. The results say that we cannot calculate exact p-values and another method is needed such as fold change which is more elementary with no assumptions. In addition, it is known that the fold change or log2 fold change shows gene expression and is used with DEG analysis. It is visually seen as a scatter plot or a volcano plot.

Q3.2. 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.)

The apply() family in the R base package is a much faster method of achieving the same thing as the for-loop. Apply functions all work in the same manner but depending on which base is used. In this section, 'apply' implementing function to specified margins is used.

> apply(X, MARGIN, FUN, ...)

- X is an array or a matrix
- Margin is a parameter dictating what the function is being applied to. 1 is for rows, 2 is for columns, every value in matrix is dictated by 1:2)

```
> ttest.p.value <- apply(input.all.sort[,2:13], 1, function(x){
  tmp <- as.vector(x);
#getting the p-value of the t test
  p.value <-t.test(
    as.vector(tmp[c(1:3)]),
    as.vector(tmp[c(4:12)]),
    alternative="two.sided",
    paired = FALSE,
    conf.int = TRUE,
    conf.level = 0.95,
    exact = TRUE
    )$p.value;
#getting the p-value of the Wilcoxon test
  wilcox <- wilcox.test(
    as.numeric(tmp[c(1:3)]),
    as.numeric(tmp[c(4:12)]),
    alternative="two.sided",
    paired = FALSE,
    conf.int = FALSE,
    conf.level = 0.95,
```

```
    exact = FALSE
    )$p.value;
#getting the value of fold change
  fold.changes <- mean(as.numeric(tmp[c(4:12)]), na.rm=TRUE) /
mean(as.numeric(tmp[c(1:3)]), na.rm=TRUE);
#return the vectors containing only p-values
  return(c(p.value, wilcox));
  }
)

#transposing a matrix or data.frame
> ttest.p.value <- t(ttest.p.value)
```

```
> head(ttest.p.value)
          [,1]      [,2]
1   0.03013882 0.1390867
2   0.89591074 0.8533074
130 0.33285626 0.4595426
212 0.95639483 1.0000000
308 0.46861693 0.5790997
378 0.85592425 0.8533074
```

Q4: Learning about the "multiple testing problem"

In a hypothesis test, there is a bogus significant result(about 5%). If thousands of tests are run, the number of false alarms increases dramatically. For instance, If we use the standard alpha level of 5% in 10,000 separate hypothesis tests, this statistic may be mistakenly selected that 500 genes, 5% of 10,000 genes, are meaningful between the two groups. This means that we would get around 500 significant results, most of which will be false alarms. When we run multiple hypothesis tests, this large number of false alarms is produced, and it is called the multiple testing problem. (Or multiple comparisons problem). They focus on specificity by controlling type1(false positive) error rates such as the family-wise-error rate or the false discovery rate. As a result ,it is necessary to make corrections for the possibility of false positives using several adjusting methods when doing a high number of hypothesis tests. One of the ways is the FDR(False-Discovery Rate) approach that is a method of adjusting the proportion of the hypotheses that are not actually significant. Another is the Bonferroni method which is the most intuitive and simple way to control for the familywise error rate. The significance level is divided into the number of times to test the level, and thus it sets as the significance level of the individual test. For example, if you do 5000 T-tests, each T-test significance level is 0.05 / 5000 = 0.0001.

Q4.1. Find R commands to adjust for multiple-testing using the two best-known adjustments (FDR and Bonferroni).
Given a set of p-values, p-values could be adjusted using several methods including p.adjust.

```
> p.adjust(p, method = p.adjust.methods, n = length(p))
```

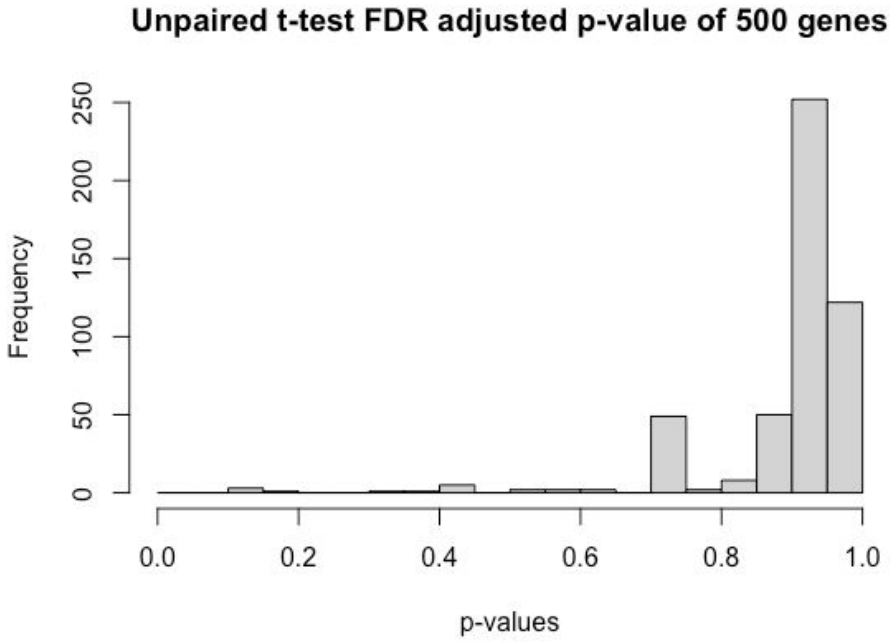
- p is the numeric vector of p-values from the t test or the Wilcoxon test
- method is correction methods such as fdr, bonferroni, BH, none, etc
- n=500 the number of comparisons is 500

```
#adjusting by fdr
> fdr.ttest <- p.adjust(p.values[,1], method = 'fdr', n=500)
> fdr.wilcox <- p.adjust(p.values[,2], method = 'fdr', n=500)

#adjusting by bonferroni
> bonferroni.ttest <- p.adjust(p.values[,1], method = 'bonferroni', n=500)
> bonferroni.wilcox <- p.adjust(p.values[,2], method = 'bonferroni', n=500)

#combining each result into a dataframe
> multiple.testing <- data.frame(ttest.p.value= as.numeric(p.values[,1]),
                      wilcox.p.vlaue= as.numeric(p.values[,2]),
                      fdr.ttest=as.numeric(fdr.ttest),
                      fdr.wilcox=as.numeric(fdr.wilcox),
                      bonferroni.ttest=as.numeric(bonferroni.ttest),
                      bonferroni.wilcox=as.numeric(bonferroni.wilcox))
```

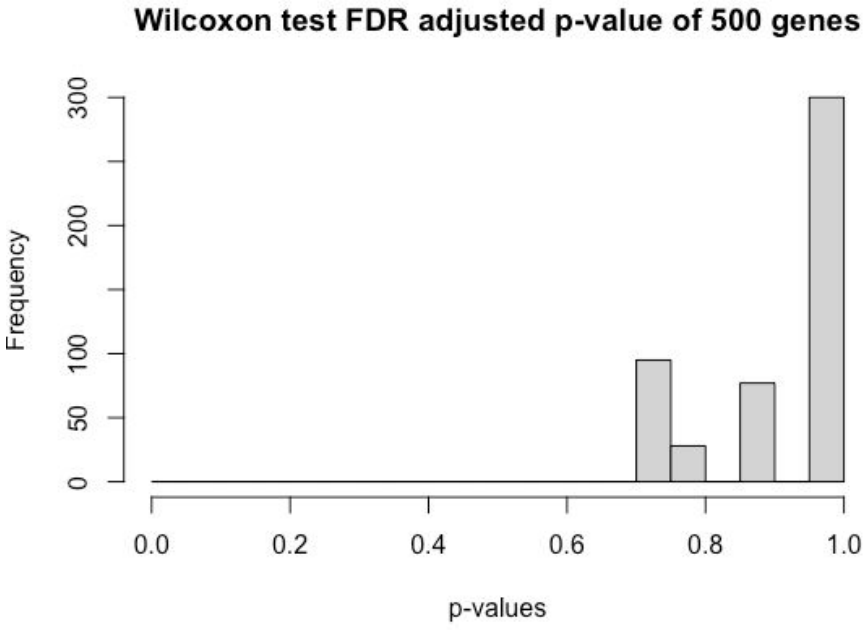
```
> head(multiple.testing)
  ttest.p.value wilcox.p.vlaue fdr.ttest fdr.wilcox bonferroni.ttest bonferroni.wilcox
1    0.03013882      0.1390867 0.7138753  0.7320354                1                1
2    0.89591074      0.8533074 0.9772713  0.9853434                1                1
3    0.33285626      0.4595426 0.9305152  0.9654256                1                1
4    0.95639483      1.0000000 0.9776355  1.0000000                1                1
5    0.46861693      0.5790997 0.9305152  0.9749154                1                1
6    0.85592425      0.8533074 0.9716167  0.9853434                1                1
```

Q4.2. Recreate the histograms above. What does this tell you?

```
#creating the histogram of the t test FDR adjusted p-value
> hist(multiple.testing$fdr.ttest,
    main = "Unpaired t-test FDR adjusted p-value of 500 genes",
    xlab = "p-values",
    breaks = seq(0,1,0.05)
    )
```

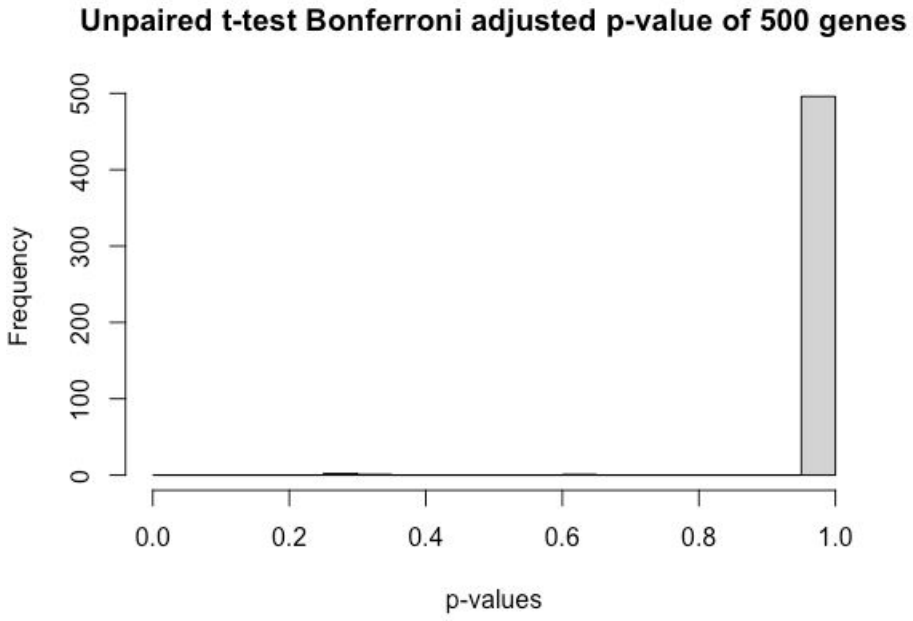



```
#creating the histogram of the Wilcoxon test FDR adjusted p-value
> hist(multiple.testing$fdr.wilcox,
    main = "Wilcoxon test FDR adjusted p-value of 500 genes",
    xlab = "p-values",
    breaks = seq(0,1,0.05)
    )
```

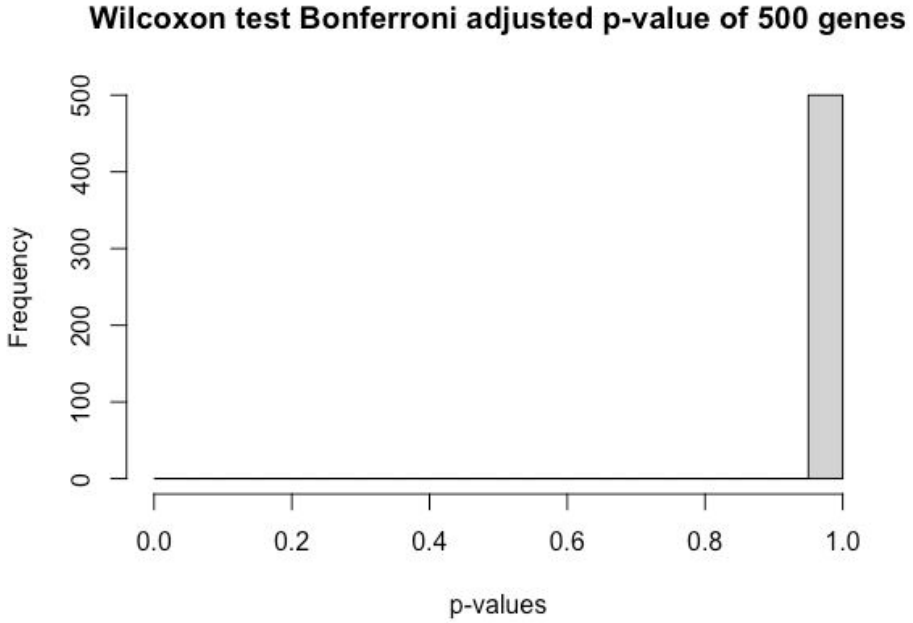



```
#creating the histogram of the t test Bonferroni adjusted p-value
> hist(multiple.testing$bonferroni.ttest,
```

```
    main = "Unpaired t-test Bonferroni adjusted p-value of 500 genes",
    xlab = "p-values",
    breaks = seq(0,1,0.05)
    )
```




```
#creating the histogram of the Wilcoxon test Bonferroni adjusted p-value
> hist(multiple.testing$bonferroni.wilcox,
    main = "Wilcoxon test Bonferroni adjusted p-value of 500 genes",
    xlab = "p-values",
    breaks = seq(0,1,0.05))
```



With the FDR and Bonferroni corrections, It is corrected for false positives that arise from multiple testing. Above the results, I can see that the FDR correction is more tolerant than the Bonferroni about a t-test and a Wilcoxon test. A number of genes with p-values(t-test)

between 0.1 - 0.15 are located when performing the FDR correction, and it could be statistically significant. On the other hand, the Bonferroni correction is more stringent and corrects most p-values for the t-test and Wilcoxon test to 1. Thus, there would be no genes which have any statistical significance by using the Bonferroni correction.

Q5: Bootstrap permutation approach

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.

Q5.1. Calculate the median of the first three columns for each gene

```
> median.first.3 <- apply(input_merge, 1, function(x) median(as.numeric(x[2:4])))
```

```
> head(median.first.3)
[1] 5.826513 4.849585 5.488923 6.160881 5.600402 6.521967
```

Q5.2. Use a permutation test to estimate the expected value for each gene:

```
> bootstrap.larger <- NULL
> bootstrap.smaller <- NULL
> bootstrap <- NULL
> num = 1
#d. Repeat a.-c. 1000 times
> while (num < 1001) {
  median.first.3 <- apply(input_merge, 1, function(x) median(as.numeric(x[2:4])));
#a. Randomly select three columns from amongst all 12
 random.3 <- apply(input_merge, 1, function(x) sample(as.numeric(x[2:13]),3, replace =
TRUE));
#b. Calculate their median
  median.random <- apply(random.3, 2, function(x) median(x));
  median.first3.random <- cbind(median.first.3, median.random)
#c. Determine if this value is larger or smaller than that of the first 3 columns
  bootstrap.larger <- apply(median.first3.random, 1, function(x) x[2] > x[1])
  bootstrap.smaller <- apply(median.first3.random, 1, function(x) x[2] < x[1])
  bootstrap <- cbind(bootstrap, bootstrap.larger);
  num <- num + 1
}
```

```
> head(median.first3.random)
     median.first.3 median.random
[1,]       5.826513      5.812383
[2,]       4.849585      4.850722
[3,]       5.488923      5.488923
[4,]       6.160881      5.935700
[5,]       5.600402      5.732801
[6,]       6.521967      6.344943
```

```
> frequency <- NULL
> frequency <- apply(bootstrap, 1, function(x) sum(x))
```

```
> frequency
  [1] 165 241 932  56 848  23 845 868 350 639 234 376  19 927 743 622 638 267 617 504  61
 [22] 856 645 757 621 637 356  77 929 840  74 845 273 853 745 726 639 253  69 148 719 492
 [43] 379 158 264 458 842 483 151 932 286 744 407 500 731 735  68 626 919 732 395 923 284
 [64] 514 474  14 838 386 247 748 613 157 351 160 370 479 478 928 647 489 935 373 156 836
 [85] 144 510 822 714 268 159 470 400 610 480 736 857 373 855 730 283  74 622  16 352 717
[106] 165 760 832  79 489 722  72 744  63 271 930 841 746 745 922 362 247 159 919 254 246
[127]  19 748 772 236 624 861  73 618 387 851 816 625  21 929 622 394 736 838 162 657 163
[148] 845 833 504 519 936 750 163 911  76 267 828 361 934 147 368 623 726 499 721 603 724
```

Q5.3. Use the frequencies in 2. to estimate a p-value for each gene

```
> estimate.frequency <- frequency/1000
```

```
> estimate.frequency
  [1] 0.165 0.241 0.932 0.056 0.848 0.023 0.845 0.868 0.350 0.639 0.234 0.376 0.019 0.927
 [15] 0.743 0.622 0.638 0.267 0.617 0.504 0.061 0.856 0.645 0.757 0.621 0.637 0.356 0.077
 [29] 0.929 0.840 0.074 0.845 0.273 0.853 0.745 0.726 0.639 0.253 0.069 0.148 0.719 0.492
 [43] 0.379 0.158 0.264 0.458 0.842 0.483 0.151 0.932 0.286 0.744 0.407 0.500 0.731 0.735
 [57] 0.068 0.626 0.919 0.732 0.395 0.923 0.284 0.514 0.474 0.014 0.838 0.386 0.247 0.748
 [71] 0.613 0.157 0.351 0.160 0.370 0.479 0.478 0.928 0.647 0.489 0.935 0.373 0.156 0.836
 [85] 0.144 0.510 0.822 0.714 0.268 0.159 0.470 0.400 0.610 0.480 0.736 0.857 0.373 0.855
 [99] 0.730 0.283 0.074 0.622 0.016 0.352 0.717 0.165 0.760 0.832 0.079 0.489 0.722 0.072
```

Q5.4. Perform a false-discovery adjustment on the p-values (?p.adjust)

```
> estimate.frequency.fdr <- p.adjust(estimate.frequency, method = "fdr", n=1000)
```

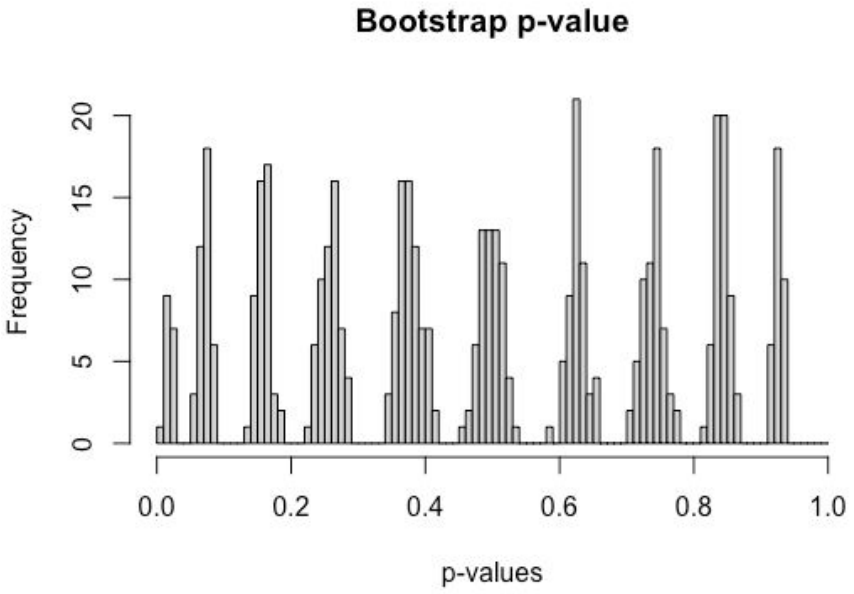
Q5.5. Write your results (gene ID, observed median, expected median, p-value, adjusted p-value) to file in a tab-delimited format

```
> total.results <- data.frame(
        GeneID=input_merge$GeneID,
        observed.median= median.random,
        expected.median=median.first.3,
        p.value=estimate.frequency,
        adjusted.p.value=estimate.frequency.fdr)
```

```
> head(total.results)
    GeneID observed.median expected.median p.value adjusted.p.value
1    10_at        5.880362        5.826513   0.167                1
2   100_at        4.964242        4.849585   0.251                1
3  1000_at        5.765557        5.488923   0.933                1
4 10000_at        5.782830        6.160881   0.084                1
5 10001_at        5.641978        5.600402   0.858                1
6 10002_at        6.294820        6.521967   0.014                1
```

Q5.6. Plot a histogram of the (unadjusted) p-values.

```
> hist(total.results$p.value,
    main = "Bootstrap p-value",
    freq = TRUE,
    xlab = "p-values",
    breaks = seq(0,1, 0.01))
```



The Bootstrap randomly selects samples with replacement from a small population, and it forms a normal distribution and obtains a confidence interval(95%). Above the results, there is the probability that the median of the randomly three selected values (from the 12 types) is greater than the median of the three samples (tumor type A), which is obtained by repeating it 1000 times for each gene. Using histograms, we investigate the bootstrap distribution geometry using histograms. The bootstrap distribution is for the statistics selected from each resample, and thus the more times the repetition is, the more accurate the estimate for the standard error or confidence interval. We can see the histogram shows us that these probabilities are not normally distributed as we got many results.

Q6: Using Boutros.Lab packages to rework previous examples

```
> library(BoutrosLab.statistics.general)
> library(BoutrosLab.plotting.general)
```

Q6.1. Using BoutrosLab.statistics.general for all p-value extraction functions

```
> set.seed(12345)

# t.test p.values with BoutrosLab.statistics.general
> t.test.p.values <- apply(input.all.sort[,2:13], 1, function(x)
  BoutrosLab.statistics.general::get.ttest.p(
    as.numeric(x),
    group1 = c(1:3),
    group2 = c(4:12)
  )
)
```

```
> head(t.test.p.values)
         1          2        130        212        308        378
0.03013882 0.89591074 0.33285626 0.95639483 0.46861693 0.85592425
```

```
# Wilcox test p.values with BoutrosLab.statistics.general
> wilcox.test.p.values <- apply(input.all.sort[,2:13], 1, function(x)
  BoutrosLab.statistics.general::get.utest.p(
    as.numeric(x),
    group1 = c(1:3),
    group2 = c(4:12)
  )
)
```

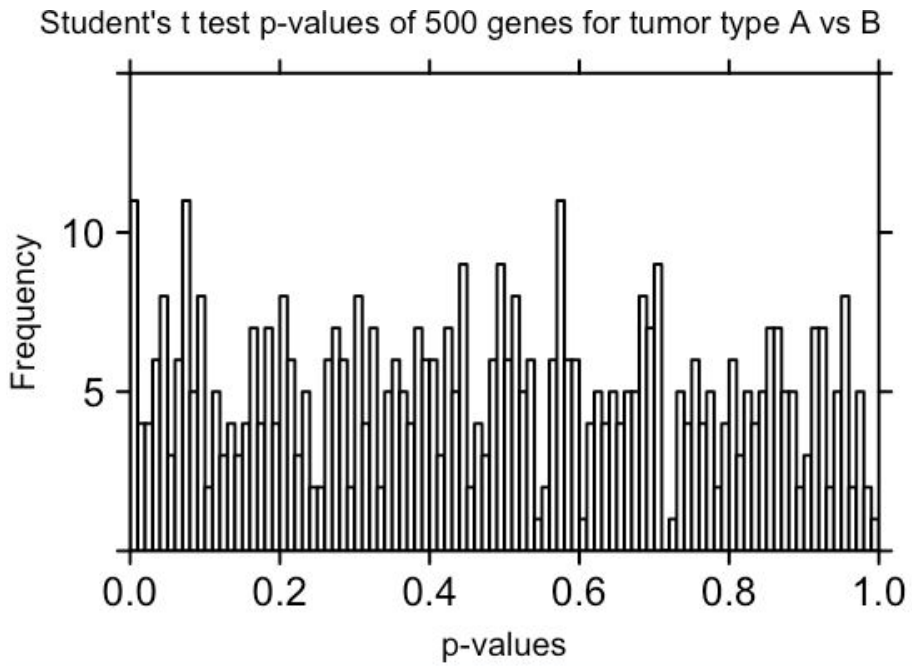
```
> head(wilcox.test.p.values)
        1         2       130       212       308       378
0.1454545 0.8636364 0.4818182 1.0000000 0.6000000 0.8636364
```

```
# fold change with BoutrosLab.statistics.general
> foldchange.values <- apply(input.all.sort[,2:13], 1, function(x)
  BoutrosLab.statistics.general::get.foldchange(
    as.numeric(x),
    group1 = c(1:3),
    group2 = c(4:12),
    logged = FALSE
  )
)
```

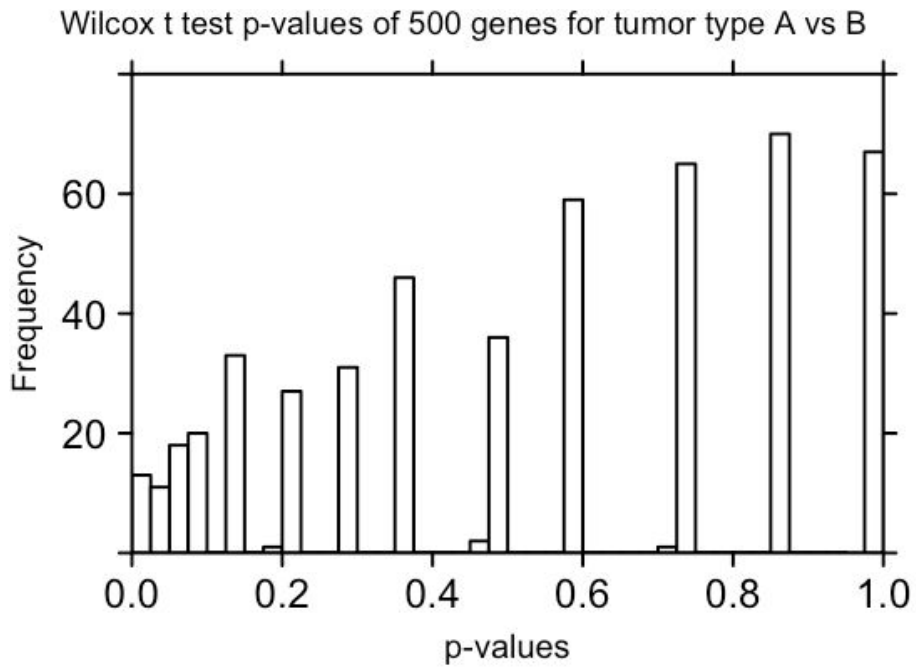
```
> head(foldchange.values)
        1         2       130       212       308       378
0.9656092 1.0053204 0.9903329 1.0006716 0.9814048 1.0026794
```

Q6.2. Using BoutrosLab.plotting.general for all plots

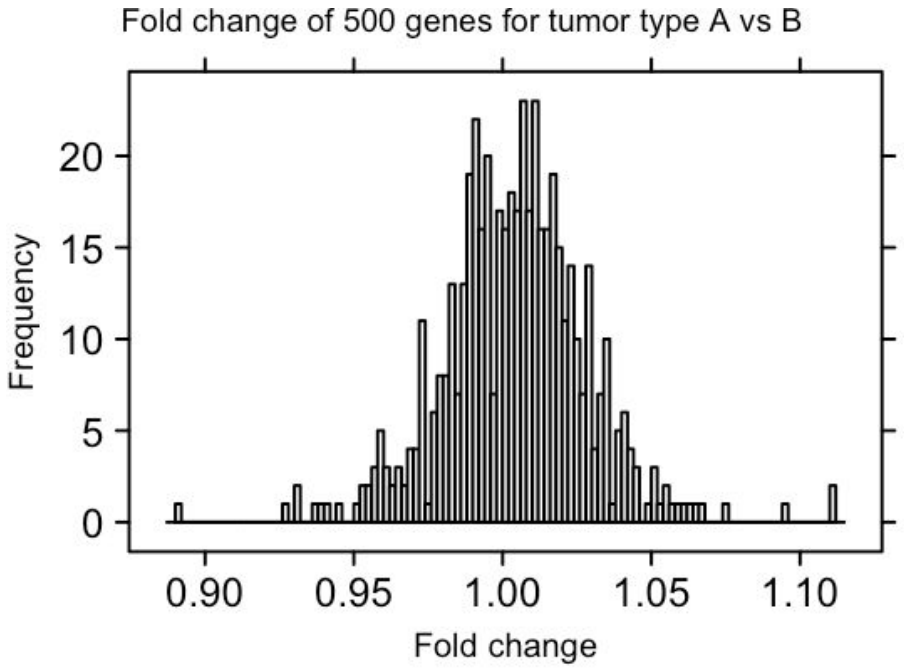
```
# Histogram of t.test p.values with BoutrosLab.plotting.general
> BoutrosLab.plotting.general::create.histogram(
  x = as.numeric(t.test.p.values),
  main = "Student's t test p-values of 500 genes for tumor type A vs B",
  main.cex = 1.2,
  xlab.label = "p-values",
  xlab.cex = 1.25,
  xlimits = c(0,1),
  ylimits = c(0, 15),
  ylab.cex = 1.25,
  ylab.label = "Frequency",
  type = "count",
  xat =  c(seq(0, 1, 0.2)),
  breaks = seq(0, 1, 0.01),
)
```



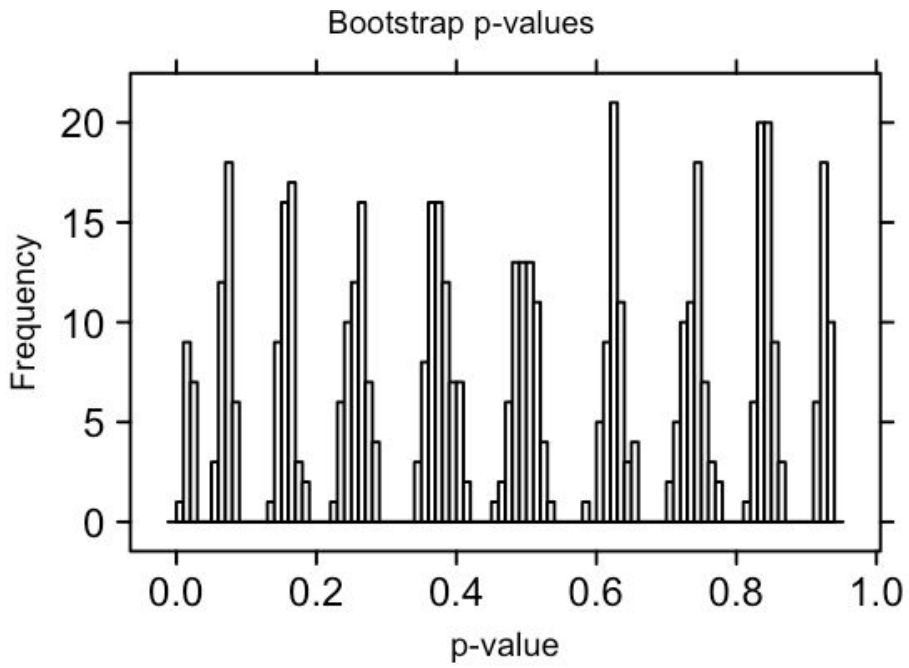
```
# Histogram of wilcox p.values with BoutrosLab.plotting.general
> BoutrosLab.plotting.general::create.histogram(
  x = as.numeric(wilcox.test.p.values),
  main = "Wilcox t test p-values of 500 genes for tumor type A vs B",
  main.cex = 1.2,
  xlab.label = "p-values",
  xlab.cex = 1.25,
  xlimits = c(0,1),
  ylimits = c(0, 80),
  ylab.label = "Frequency",
  ylab.cex = 1.25,
  type = "count",
  xat =  c(seq(0, 1, 0.2)),
  breaks = seq(0, 1, 0.025),
)
```



```
# Histogram of t.test p.values with BoutrosLab.plotting.general
> BoutrosLab.plotting.general::create.histogram(
  x = as.numeric(foldchange.values),
  main = "Fold change of 500 genes for tumor type A vs B",
  main.cex = 1.2,
  xlab.label = "Fold change",
  xlab.cex = 1.25,
  ylab.label = "Frequency",
  ylab.cex = 1.25,
  type = "count",
  breaks = 100,
)
```



```
# Histogram of bootstrap p.values with BoutrosLab.plotting.general
> BoutrosLab.plotting.general::create.histogram(
  x = as.numeric(total.results$p.value),
  main = "Bootstrap p-values",
  main.cex = 1.2,
  xlab.label = "p-value",
  xlab.cex = 1.25,
  ylab.label = "Frequency",
  ylab.cex = 1.25,
  type = "count",
  xat =  c(seq(0, 1, 0.2)),
  breaks = 100,
)
```



Reference

D.Scholtens and A.von Heydebreck, Analysis of differential gene expression studies, Springer, 2005, Online ISBN 978-0-387-29362-2

Jeong-Seok Choi, Biostatistics for Multiple Testing, Korean J Otorhinolaryngol-Head Neck Surg, 2020;63(3):97-100 / eISSN 2092-6529

Kim et al, A network-based approach to detect disease-related genes using differentially expressed gene analysis, Korea database society, 2012;23;(3) / ISSN 1598-9798

James (JD) Long and Paul Teetor, R Cookbook, 2nd Edition, 2019, O'Reilly Media, Inc.

Boutros Lab, BI-manual, 2020

https://www.rdocumentation.org/

https://www.tutorialspoint.com/r/

http://www.sthda.com/english/wiki/unpaired-two-samples-t-test-in-r

https://datacookbook.kr/76

https://www.bioconductor.org/

https://www.learnbyexample.org/r-bar-plot-ggplot2/

https://hbctraining.github.io/Training-modules/Visualization_in_R/lessons/03_advanced_visualizations.html

https://www.datanovia.com/en/lessons/wilcoxon-test-in-r/

https://hbctraining.github.io/Training-modules/Visualization_in_R/lessons/03_advanced_visualizations.html

https://rfriend.tistory.com/118

https://medium.com/@hslee09/r-%ED%86%B5%EA%B3%84%EB%B6%84%EC%84%9D-%EC%A0%81%ED%95%A9%EB%8F%84-%EA%B2%80%EC%A0%95-caf48dca9de1

https://rpubs.com/jmhome/R_data_processing

http://bpg.oicr.on.ca/API/BoutrosLab.plotting.general/5.3.4/index.html