US HUPO 2019 Short course - Section 5 : Introduction to R and basic statistics

Meena Choi

March 2, 2019

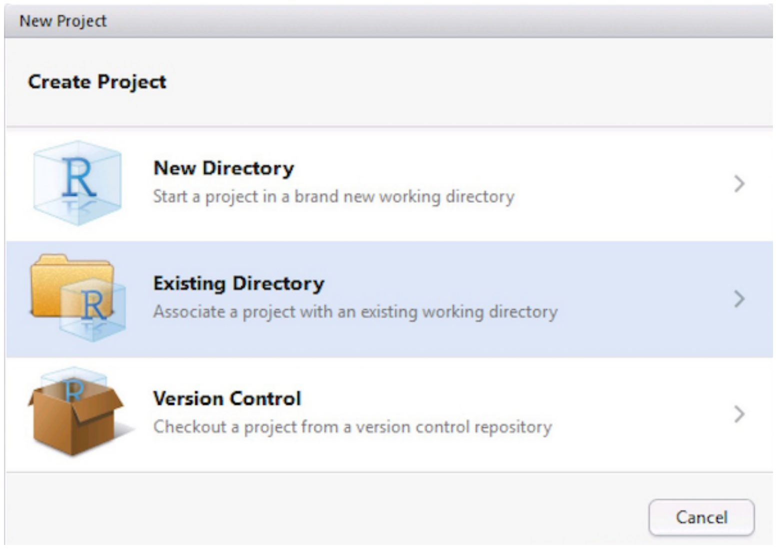
Summary

- Creating a new RStudio project
- Reading in data in R
- Data exploration, subsetting and replacement
- Visualizing data
- Select random sample and randomize MS run orders.
- Calculate simple statistics and visualize them using ggplot2.
- Statistical hypothesis testing by t-test.
- Saving your work

---

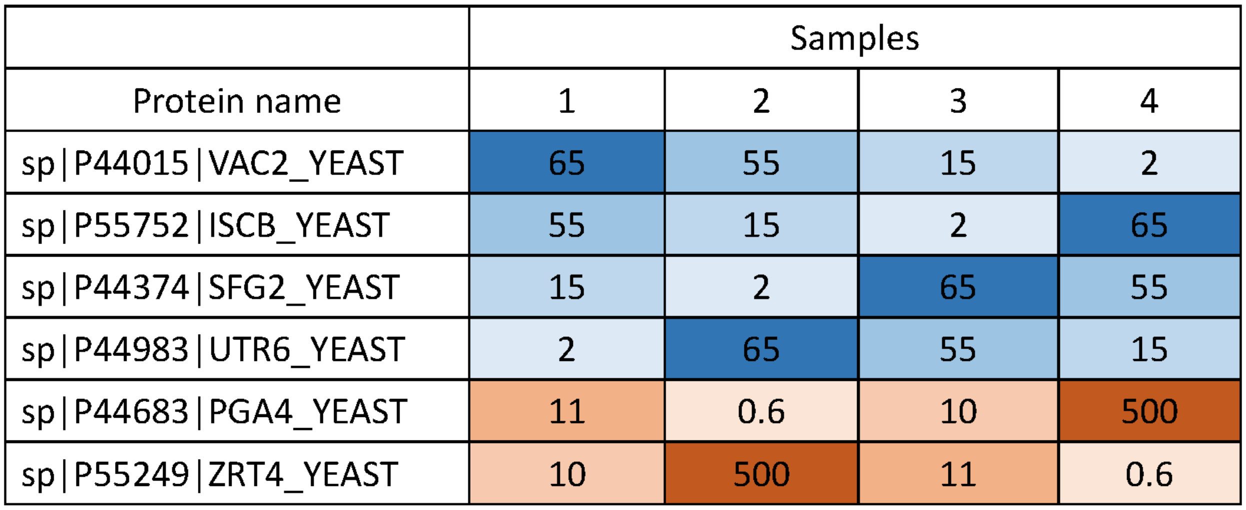
1. Create a new Rstudio project

From the menu, select **File > New Project...**, then select **Existing Directory** and choose the directory where you downloaded this script and the example datasets for this tutorial. All the output files we'll be creating in this tutorial will be saved in the 'working directory' that now has been set by Rstudio.


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Let's verify the working directory path with the get working directory command.

```
getwd()
```



	Samples			
Protein name	1	2	3	4
sp|P44015|VAC2_YEAST	65	55	15	2
sp|P55752|ISCB_YEAST	55	15	2	65
sp|P44374|SFG2_YEAST	15	2	65	55
sp|P44983|UTR6_YEAST	2	65	55	15
sp|P44683|PGA4_YEAST	11	0.6	10	500
sp|P55249|ZRT4_YEAST	10	500	11	0.6

Figure 1:

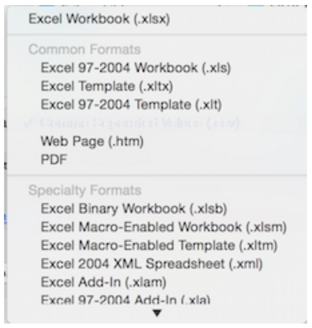
```
## [1] "/Users/meenachoi/Dropbox/visits/2019/03_USHUPO/shortcourse/prep-Day1"
```

---

2. Reading in data

The file we'll be reading in is a dataset that has been processed in Skyline and exported as a comma separated values (.csv) file, formatted for MSstats. We'll talk more about MSstats tomorrow.

Tip Often you'll get data delivered as a Microsoft Excel file. You can export any spreadsheet to a `.csv` (comma separated values) file in Excel through the ***Save As.. > Format: Comma Separated Values (.csv)*** menu item.


Excel Workbook (.xlsx)
Common Formats
Excel 97-2004 Workbook (.xls)
Excel Template (.xltx)
Excel 97-2004 Template (.xlt)
Web Page (.htm)
PDF
Specialty Formats
Excel Binary Workbook (.xlsb)
Excel Macro-Enabled Workbook (.xlsm)
Excel Macro-Enabled Template (.xltm)
Excel 2004 XML Spreadsheet (.xml)
Excel Add-In (.xlam)
Excel 97-2004 Add-In (.xla)

In Rstudio, go to the ***environnment*** pane, click on the ***Import Dataset*** dropdown and choose ***From Text File...*** from the dropdown menu. Import the `iPRG_example_runsummary.csv` file and inspect that Rstudio correctly parsed the text file into an R `data frame`.

Now inspect the Rstudio ***Console*** and ***Environment*** pane again. Notice that a new variable for the `iPRG_example` data frame was created in the environment by executing the `read.csv` function. Let's have a look at the documentation for this function by pulling up the help pages with the `?`.

```
iprg <- read.csv("iPRG_example_runsummary.csv")
```

Tip: try using RStudio's auto-complete functionality by pressing TAB on any partially typed function or variable in RStudio.

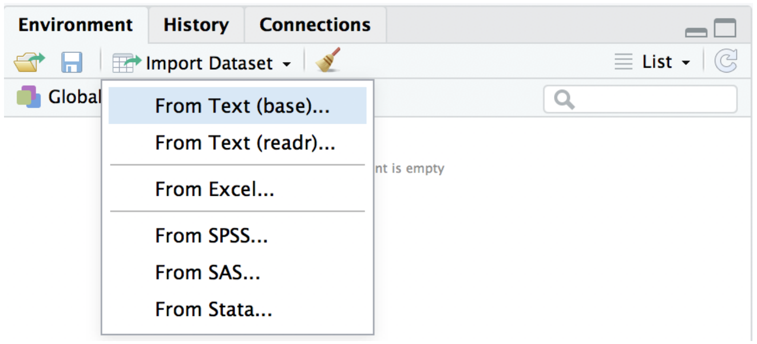
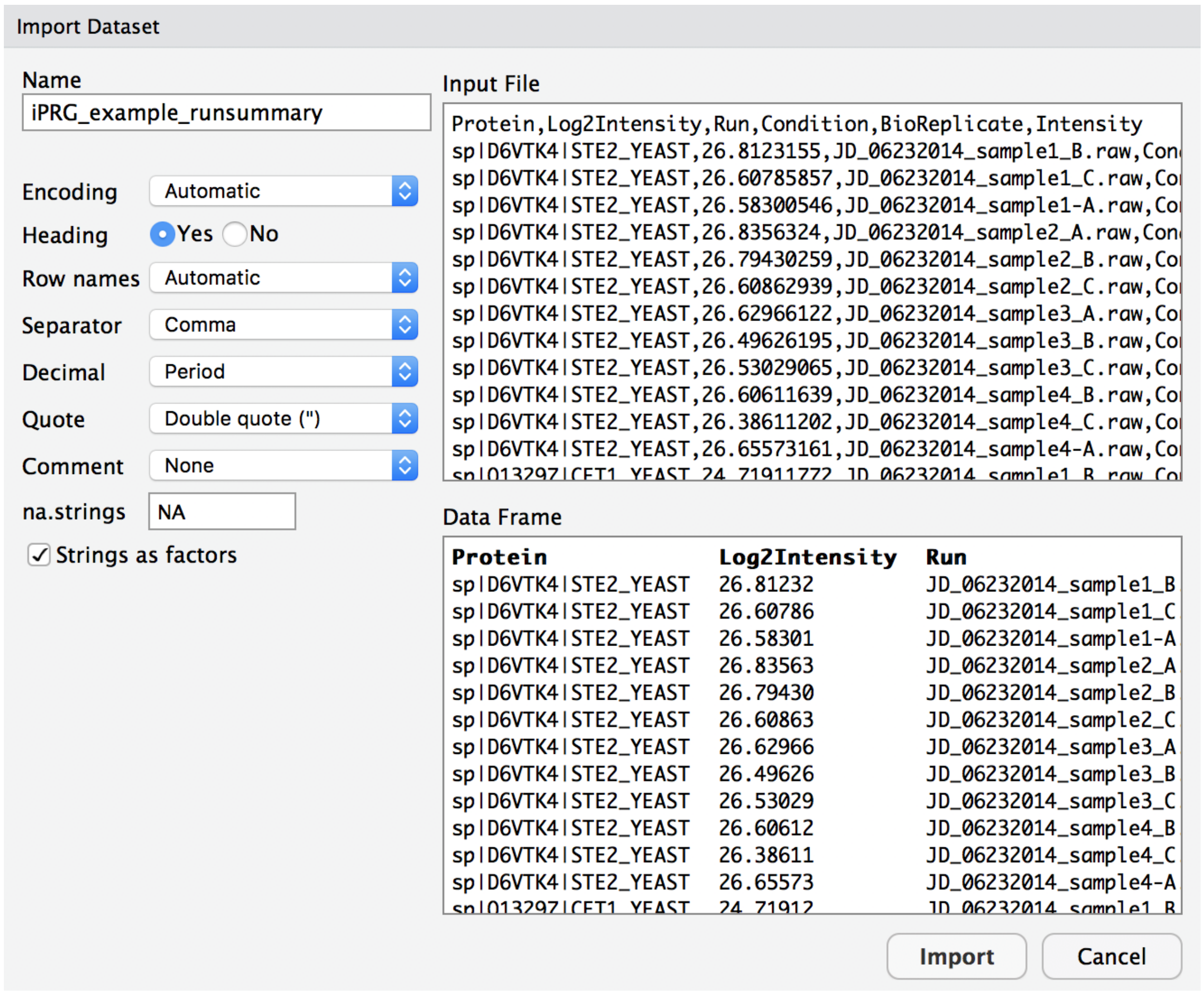


Figure 2:


Import Dataset

Name	iPRG_example_runsummary
Encoding	Automatic
Heading	[✓ Yes] No
Row names	Automatic
Separator	Comma
Decimal	Period
Quote	Double quote (")
Comment	None
na.strings	NA
✓ Strings as factors	

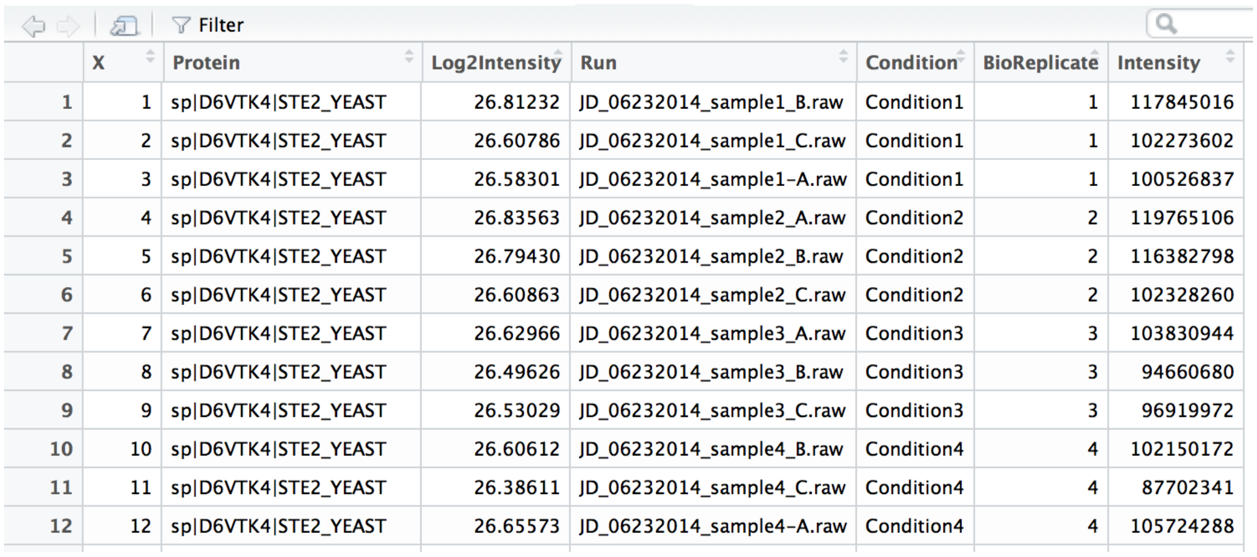
Input File

```
Protein,Log2Intensity,Run,Condition,BioReplicate,Intensity
sp|D6VTK4|STE2_YEAST,26.8123155,JD_06232014_sample1_B.raw,Con
sp|D6VTK4|STE2_YEAST,26.60785857,JD_06232014_sample1_C.raw,Co
sp|D6VTK4|STE2_YEAST,26.58300546,JD_06232014_sample1-A.raw,Co
sp|D6VTK4|STE2_YEAST,26.8356324,JD_06232014_sample2_A.raw,Con
sp|D6VTK4|STE2_YEAST,26.79430259,JD_06232014_sample2_B.raw,Co
sp|D6VTK4|STE2_YEAST,26.60862939,JD_06232014_sample2_C.raw,Co
sp|D6VTK4|STE2_YEAST,26.62966122,JD_06232014_sample3_A.raw,Co
sp|D6VTK4|STE2_YEAST,26.49626195,JD_06232014_sample3_B.raw,Co
sp|D6VTK4|STE2_YEAST,26.53029065,JD_06232014_sample3_C.raw,Co
sp|D6VTK4|STE2_YEAST,26.60611639,JD_06232014_sample4_B.raw,Co
sp|D6VTK4|STE2_YEAST,26.38611202,JD_06232014_sample4_C.raw,Co
sp|D6VTK4|STE2_YEAST,26.65573161,JD_06232014_sample4-A.raw,Co
sp|Q13297|CET1_YEAST,24.71911772,JD_06232014_sample1_B.raw,Co
```

Data Frame

Protein	Log2Intensity	Run
sp|D6VTK4|STE2_YEAST	26.81232	JD_06232014_sample1_B
sp|D6VTK4|STE2_YEAST	26.60786	JD_06232014_sample1_C
sp|D6VTK4|STE2_YEAST	26.58301	JD_06232014_sample1-A
sp|D6VTK4|STE2_YEAST	26.83563	JD_06232014_sample2_A
sp|D6VTK4|STE2_YEAST	26.79430	JD_06232014_sample2_B
sp|D6VTK4|STE2_YEAST	26.60863	JD_06232014_sample2_C
sp|D6VTK4|STE2_YEAST	26.62966	JD_06232014_sample3_A
sp|D6VTK4|STE2_YEAST	26.49626	JD_06232014_sample3_B
sp|D6VTK4|STE2_YEAST	26.53029	JD_06232014_sample3_C
sp|D6VTK4|STE2_YEAST	26.60612	JD_06232014_sample4_B
sp|D6VTK4|STE2_YEAST	26.38611	JD_06232014_sample4_C
sp|D6VTK4|STE2_YEAST	26.65573	JD_06232014_sample4-A
sp|Q13297|CET1_YEAST	24.71912	JD_06232014_sample1_B

Figure 3:



	X	Protein	Log2Intensity	Run	Condition	BioReplicate	Intensity
1	1	sp|D6VTK4|STE2_YEAST	26.81232	JD_06232014_sample1_B.raw	Condition1	1	117845016
2	2	sp|D6VTK4|STE2_YEAST	26.60786	JD_06232014_sample1_C.raw	Condition1	1	102273602
3	3	sp|D6VTK4|STE2_YEAST	26.58301	JD_06232014_sample1-A.raw	Condition1	1	100526837
4	4	sp|D6VTK4|STE2_YEAST	26.83563	JD_06232014_sample2_A.raw	Condition2	2	119765106
5	5	sp|D6VTK4|STE2_YEAST	26.79430	JD_06232014_sample2_B.raw	Condition2	2	116382798
6	6	sp|D6VTK4|STE2_YEAST	26.60863	JD_06232014_sample2_C.raw	Condition2	2	102328260
7	7	sp|D6VTK4|STE2_YEAST	26.62966	JD_06232014_sample3_A.raw	Condition3	3	103830944
8	8	sp|D6VTK4|STE2_YEAST	26.49626	JD_06232014_sample3_B.raw	Condition3	3	94660680
9	9	sp|D6VTK4|STE2_YEAST	26.53029	JD_06232014_sample3_C.raw	Condition3	3	96919972
10	10	sp|D6VTK4|STE2_YEAST	26.60612	JD_06232014_sample4_B.raw	Condition4	4	102150172
11	11	sp|D6VTK4|STE2_YEAST	26.38611	JD_06232014_sample4_C.raw	Condition4	4	87702341
12	12	sp|D6VTK4|STE2_YEAST	26.65573	JD_06232014_sample4-A.raw	Condition4	4	105724288

Figure 4:

---

3. Data exploration

Let's explore some basic properties of our dataset. Go to the RStudio Environment pane and double click the `iPRG_example` entry. This data is in 'long' format, which is an easier data format for data manipulation operations such as selecting, grouping, summarizing, etc.

Data exported out of spectral processing or quantification tools is often also formatted in 'wide' format, which is easier to read when we would like to compare values (i.e intensity values) for specific subjects (i.e peptides) across different values for a variable of interest such as (i.e conditions). We'll format a summary of this dataset as a 'wide' data frame later in this tutorial.

Ok, let's do some more data exploration by examining how R read in the iPRG dataset.

`class` shows the type of a variable, in this case a 'data.frame'.

```
class(iprg)
```

```
## [1] "data.frame"
```

`dim` shows the dimension of a data.frame, which are the number of rows and the number of columns

```
dim(iprg)
```

```
## [1] 36321     6
```

`colnames` is short for column names.

```
colnames(iprg)
```

```
## [1] "Protein"       "Log2Intensity" "Run"           "Condition"
## [5] "BioReplicate"  "Intensity"
```

`head` shows the first 6 rows of data. Try `tail` to show the last 6 rows of data.

```r
head(iprg)
```

```
##                Protein Log2Intensity                      Run  Condition
## 1 sp|D6VTK4|STE2_YEAST      26.81232 JD_06232014_sample1_B.raw Condition1
## 2 sp|D6VTK4|STE2_YEAST      26.60786 JD_06232014_sample1_C.raw Condition1
## 3 sp|D6VTK4|STE2_YEAST      26.58301 JD_06232014_sample1-A.raw Condition1
## 4 sp|D6VTK4|STE2_YEAST      26.83563 JD_06232014_sample2_A.raw Condition2
## 5 sp|D6VTK4|STE2_YEAST      26.79430 JD_06232014_sample2_B.raw Condition2
## 6 sp|D6VTK4|STE2_YEAST      26.60863 JD_06232014_sample2_C.raw Condition2
##   BioReplicate Intensity
## 1            1 117845016
## 2            1 102273602
## 3            1 100526837
## 4            2 119765106
## 5            2 116382798
## 6            2 102328260
```

Let's explore the type of every column/variable and a summary for the value range for every column.

```r
summary(iprg)
```

```
##                     Protein      Log2Intensity
##  sp|D6VTK4|STE2_YEAST :   12   Min.   :16.37
##  sp|O13297|CET1_YEAST :   12   1st Qu.:23.78
##  sp|O13329|FOB1_YEAST :   12   Median :24.68
##  sp|O13539|THP2_YEAST :   12   Mean   :24.82
##  sp|O13547|CCW14_YEAST:   12   3rd Qu.:25.78
##  sp|O13563|RPN13_YEAST:   12   Max.   :31.42
##  (Other)              :36249
##                          Run             Condition     BioReplicate
##  JD_06232014_sample1_C.raw: 3027   Condition1:9079   Min.   :1.0
##  JD_06232014_sample2_A.raw: 3027   Condition2:9081   1st Qu.:2.0
##  JD_06232014_sample2_B.raw: 3027   Condition3:9081   Median :3.0
##  JD_06232014_sample2_C.raw: 3027   Condition4:9080   Mean   :2.5
##  JD_06232014_sample3_A.raw: 3027                     3rd Qu.:3.0
##  JD_06232014_sample3_B.raw: 3027                     Max.   :4.0
##  (Other)                  :18159
##    Intensity
##  Min.   :8.485e+04
##  1st Qu.:1.440e+07
##  Median :2.690e+07
##  Mean   :6.387e+07
##  3rd Qu.:5.771e+07
##  Max.   :2.881e+09
##
```

Inspect the possible values for the `Conditions` and the `BioReplicate` (8th) column using the named and numbered column selection syntax for data frames.

```r
unique(iprg[, 'Condition'])
```

```
## [1] Condition1 Condition2 Condition3 Condition4
## Levels: Condition1 Condition2 Condition3 Condition4
```

```r
unique(iprg[, 4])
```

```
## [1] Condition1 Condition2 Condition3 Condition4
## Levels: Condition1 Condition2 Condition3 Condition4
```

```
unique(iprg[, c('Condition', 'BioReplicate', 'Run')])
```

```
##     Condition BioReplicate                       Run
## 1  Condition1            1 JD_06232014_sample1_B.raw
## 2  Condition1            1 JD_06232014_sample1_C.raw
## 3  Condition1            1 JD_06232014_sample1-A.raw
## 4  Condition2            2 JD_06232014_sample2_A.raw
## 5  Condition2            2 JD_06232014_sample2_B.raw
## 6  Condition2            2 JD_06232014_sample2_C.raw
## 7  Condition3            3 JD_06232014_sample3_A.raw
## 8  Condition3            3 JD_06232014_sample3_B.raw
## 9  Condition3            3 JD_06232014_sample3_C.raw
## 10 Condition4            4 JD_06232014_sample4_B.raw
## 11 Condition4            4 JD_06232014_sample4_C.raw
## 12 Condition4            4 JD_06232014_sample4-A.raw
```

Select subsets of rows from iPRG dataset: i.e we might be interested in working from here on only with Condition1 or all measurements on one particular MS run.

```
iprg.condition1 <- iprg[iprg$Condition == 'Condition1', ]
iprg.condition1.bio1 <- iprg[iprg$Condition == 'Condition1'
                              & iprg$BioReplicate == '1', ]
nrow(iprg.condition1.bio1)
```

```
## [1] 9079
```

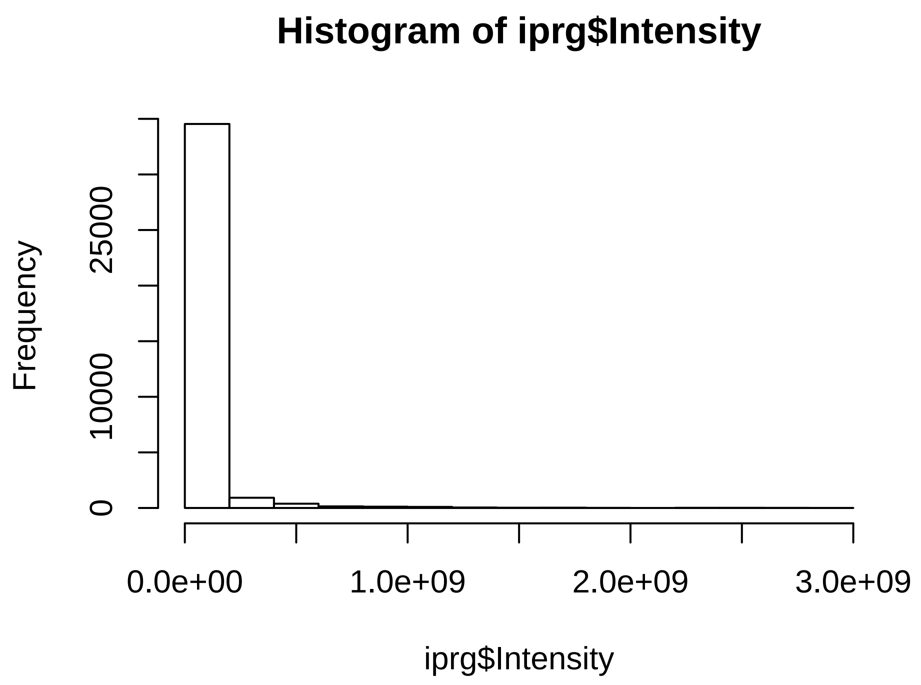
---

4. Summarizing and Visualizing data

4.1 Histogram

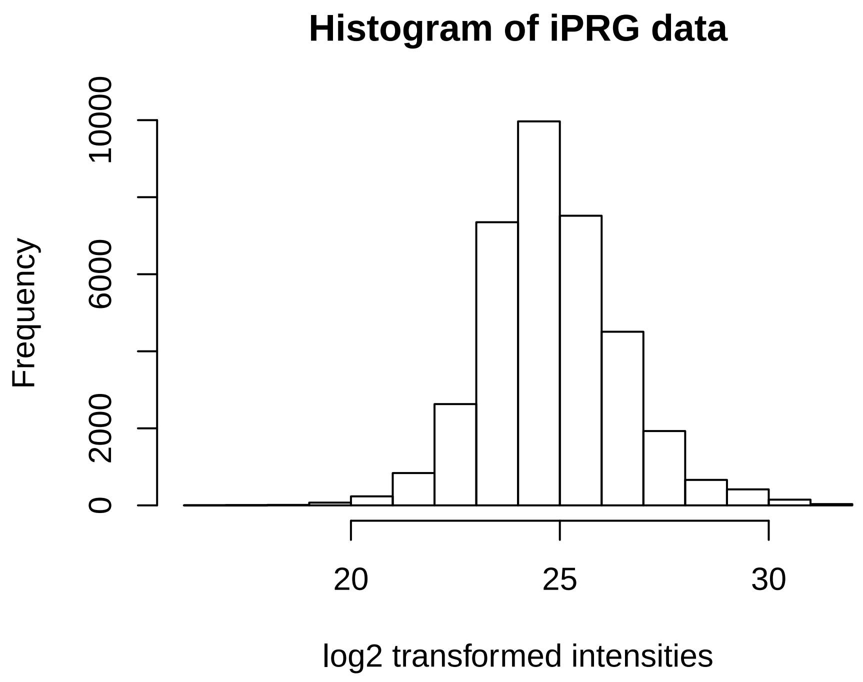
Make a histogram of all the MS1 intensities, quantified by Skyline, for `iPRG_example`.

```
hist(iprg$Intensity)
```

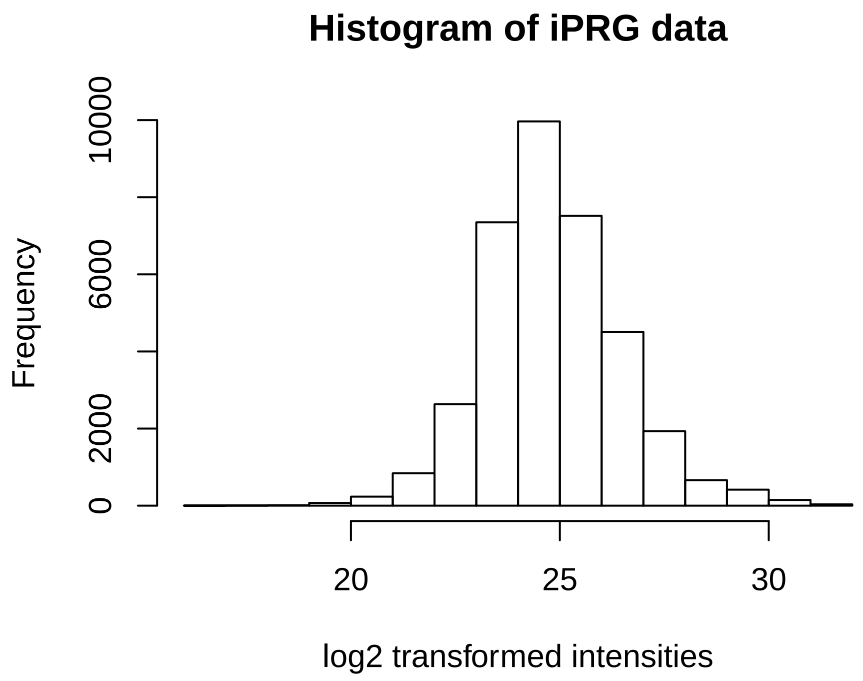


Our histogram looks quite skewed. How does this look on log-scale? Do you recognize this distribution? The distribution for log2-transformed intensities looks very similar to the normal distribution. The advantage of working with normally distributed data is that we can apply a variety of statistical tests to analyzeand interpret our data. Let's add a log2-scaled intensity column to our data so we don't have to transform the original intensities every time we need them.

```r
hist(iprg$Log2Intensity,
     xlab="log2 transformed intensities", main="Histogram of iPRG data")
```



```r
# or directly transform the intensities.
hist(log2(iprg$Intensity),
     xlab="log2 transformed intensities", main="Histogram of iPRG data")
```



We look at the summary for the log2-transformed values including the value for the mean. Let's fix that first.

```
summary(iprg$Log2Intensity)
```

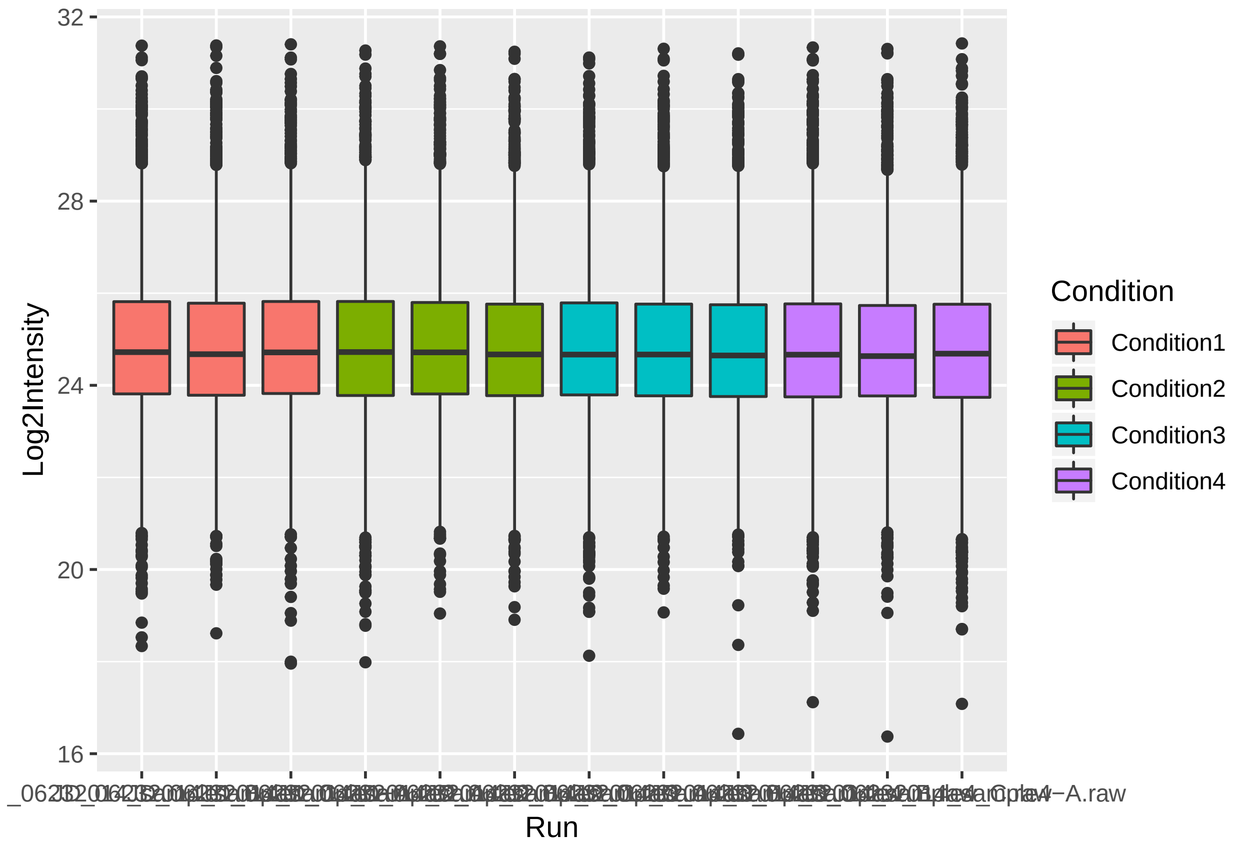
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   16.37   23.78   24.68   24.82   25.78   31.42
```

4.2 Boxplot or box-and-whisker plot

Boxplots are extremely useful becasue they allow us to quickly visualize the data distribution, without making assumptions of the distribution type (non-parametric). We can read up on what statistics the different elements of a box-and-whisker represent in the R help files.

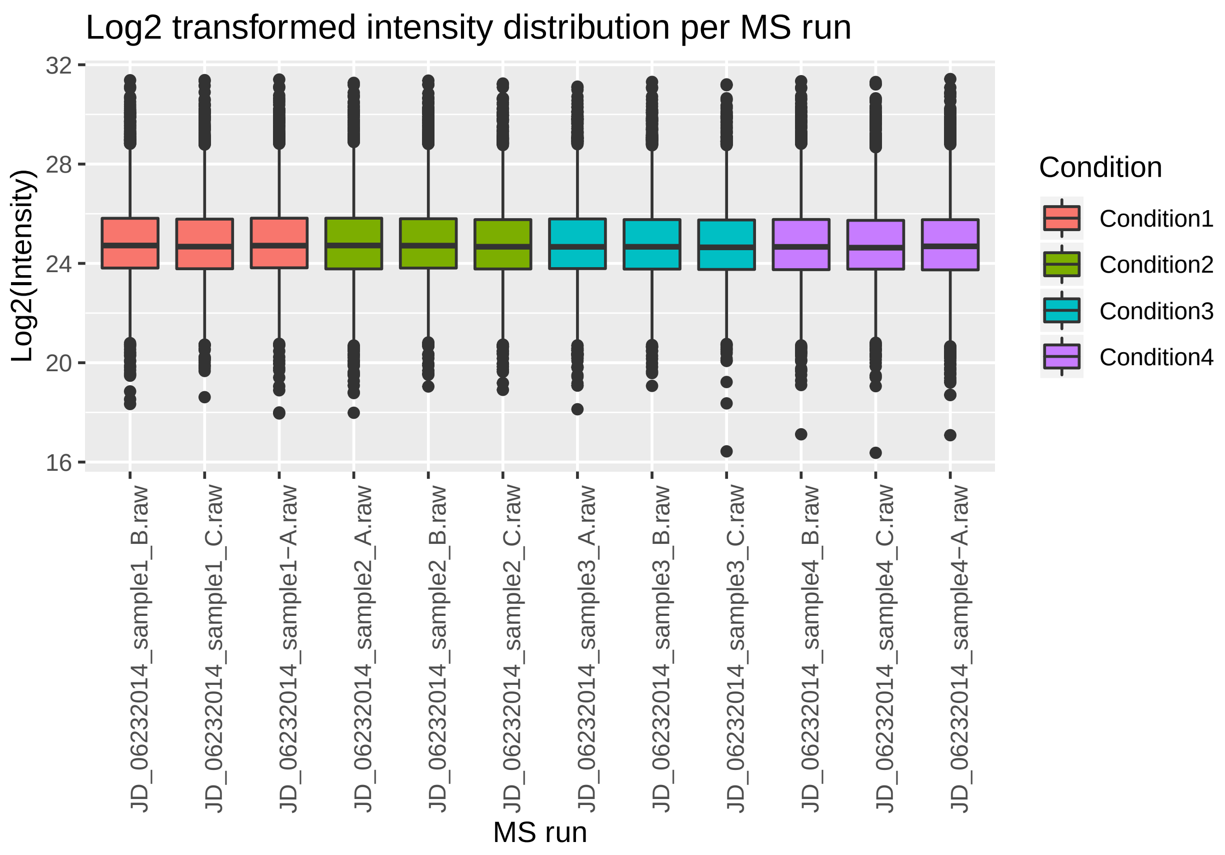
Let's make the boxplot with `ggplot2`, one of the most popular and powerful R packages for making graphics. The syntax of ggplot2 might seem a bit intimidating at first, but besides the advantage of having full control over all graphical elements of your plot, two other advantages are that 1) it's very straightforward to automatically assign distinguishing graphical elements to subsets of your data and 2) switching between plot types requires little changes to your code. Let's start with a bare bones boxplot.

```
library(ggplot2)
ggplot(aes_string(x='Run', y='Log2Intensity'), data=iprg)+
                geom_boxplot(aes_string(fill='Condition'))
```



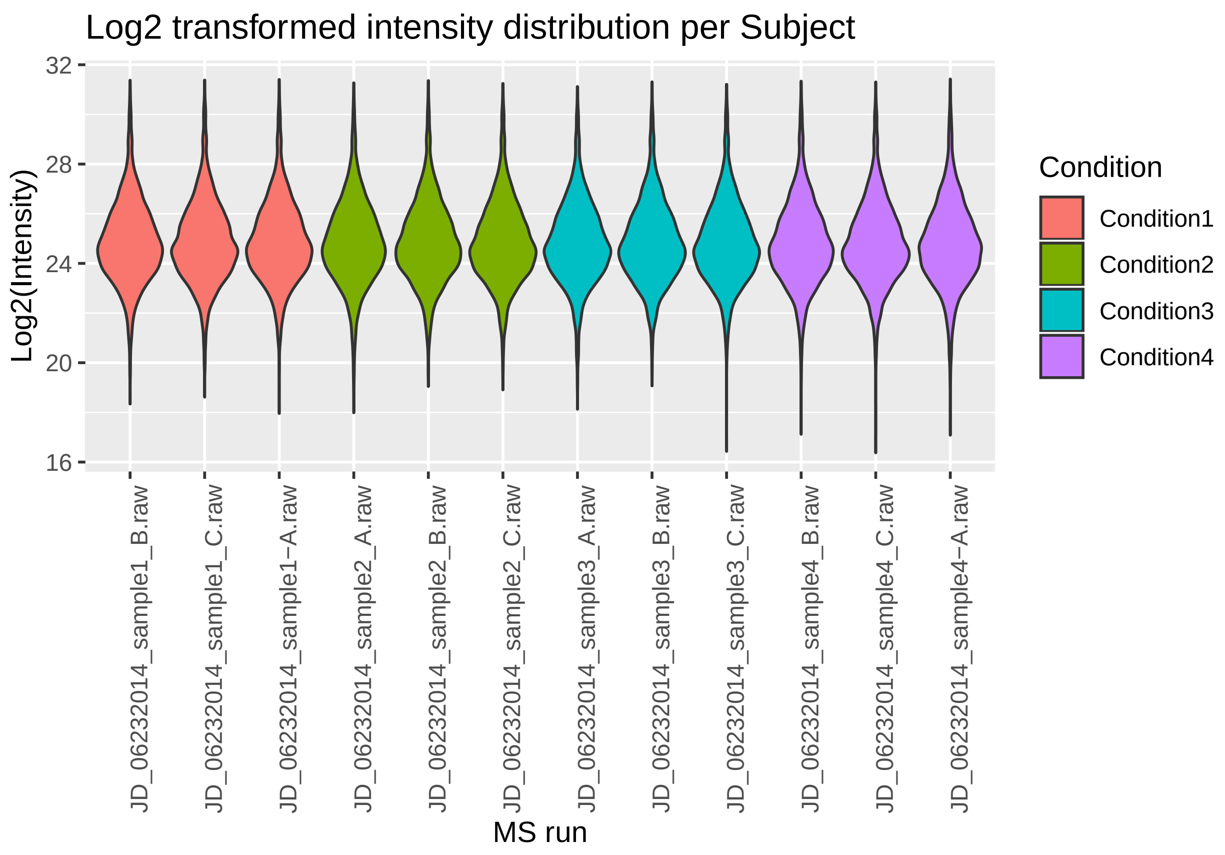
Now let's rename all axis labels and title, and rotate the x-axis labels 90 degrees. We can add those specifications using the `labs` and `theme` functions of the `ggplot2` package.

```
ggplot(aes_string(x='Run', y='Log2Intensity'), data=iprg)+
            geom_boxplot(aes_string(fill='Condition'))+
          labs(title='Log2 transformed intensity distribution per MS run',
               y='Log2(Intensity)',
               x='MS run')+
          theme(axis.text.x=element_text(angle=90))
```



And easily switch from a boxplot to a violin plot representation by changing the `geom` type.

```
ggplot(aes_string(x='Run', y='Log2Intensity'), data=iprg)+
          geom_violin(aes_string(fill='Condition'))+
        labs(title='Log2 transformed intensity distribution per Subject',
             y='Log2(Intensity)',
             x='MS run')+
        theme(axis.text.x=element_text(angle=90))
```



---

5. Randomization

5.1 Random selection of samples from a larger set

This particular dataset contains a total of 10 subjects across conditions. Suppose we label them from 1 to 14 and randomly would like to select 3 subjects we can do this using the `sample` function. When we run `sample` another time, different subjects will be selected. Try this a couple times.

```
sample(10, 3)
```

```
## [1]  1  6 10
```

```
sample(10, 3)
```

```
## [1] 1 3 9
```

Now suppose we would like to select the same randomly selected samples every time, then we can use a random seed number.

```
set.seed(3728)
sample(10, 3)
```

```
## [1] 5 8 7
```

```
set.seed(3728)
sample(10, 3)
```

```
## [1] 5 8 7
```

5.2 Completely randomized order of MS runs

We can also create a random order using all elements of iPRG dataset. Again, we can achieve this using `sample`, asking for exactly the amount of samples in the subset. This time, each repetition gives us a different order of the complete set.

```r
msrun <- unique(iprg$Run)
msrun
```

```
##  [1] JD_06232014_sample1_B.raw JD_06232014_sample1_C.raw
##  [3] JD_06232014_sample1-A.raw JD_06232014_sample2_A.raw
##  [5] JD_06232014_sample2_B.raw JD_06232014_sample2_C.raw
##  [7] JD_06232014_sample3_A.raw JD_06232014_sample3_B.raw
##  [9] JD_06232014_sample3_C.raw JD_06232014_sample4_B.raw
## [11] JD_06232014_sample4_C.raw JD_06232014_sample4-A.raw
## 12 Levels: JD_06232014_sample1_B.raw ... JD_06232014_sample4-A.raw
```

```r
# randomize order among all 12 MS runs
sample(msrun, length(msrun))
```

```
##  [1] JD_06232014_sample3_A.raw JD_06232014_sample3_C.raw
##  [3] JD_06232014_sample1_B.raw JD_06232014_sample4-A.raw
##  [5] JD_06232014_sample3_B.raw JD_06232014_sample4_B.raw
##  [7] JD_06232014_sample2_C.raw JD_06232014_sample4_C.raw
##  [9] JD_06232014_sample2_B.raw JD_06232014_sample1-A.raw
## [11] JD_06232014_sample1_C.raw JD_06232014_sample2_A.raw
## 12 Levels: JD_06232014_sample1_B.raw ... JD_06232014_sample4-A.raw
```

```r
# different order will be shown.
sample(msrun, length(msrun))
```

```
##  [1] JD_06232014_sample1_B.raw JD_06232014_sample3_B.raw
##  [3] JD_06232014_sample2_C.raw JD_06232014_sample1-A.raw
##  [5] JD_06232014_sample4_B.raw JD_06232014_sample2_A.raw
##  [7] JD_06232014_sample2_B.raw JD_06232014_sample3_A.raw
##  [9] JD_06232014_sample4_C.raw JD_06232014_sample1_C.raw
## [11] JD_06232014_sample3_C.raw JD_06232014_sample4-A.raw
## 12 Levels: JD_06232014_sample1_B.raw ... JD_06232014_sample4-A.raw
```

5.3 Randomized block design

- Allow to remove known sources of variability that you are not interested in.
- Group conditions into blocks such that the conditions in a block are as similar as possible.
- Randomly assign samples with a block.

This particular dataset contains a total of 12 MS runs across 4 conditions, 3 technical replicates per condition. Using the `block.random` function in the `psych` package, we can achieve randomized block designs!

```r
# use 'psych' package
library(psych)
```

```
##
## Attaching package: 'psych'
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
msrun <- unique(iprg[, c('Condition','Run')])
msrun
```

```
##     Condition                       Run
## 1  Condition1 JD_06232014_sample1_B.raw
## 2  Condition1 JD_06232014_sample1_C.raw
## 3  Condition1 JD_06232014_sample1-A.raw
## 4  Condition2 JD_06232014_sample2_A.raw
## 5  Condition2 JD_06232014_sample2_B.raw
## 6  Condition2 JD_06232014_sample2_C.raw
## 7  Condition3 JD_06232014_sample3_A.raw
## 8  Condition3 JD_06232014_sample3_B.raw
## 9  Condition3 JD_06232014_sample3_C.raw
## 10 Condition4 JD_06232014_sample4_B.raw
## 11 Condition4 JD_06232014_sample4_C.raw
## 12 Condition4 JD_06232014_sample4-A.raw
```

```
# 4 Conditions of 12 MS runs randomly ordered
block.random(n=12, c(Condition=4))
```

```
##     blocks Condition
## S1       1         2
## S2       1         1
## S3       1         3
## S4       1         4
## S5       2         1
## S6       2         4
## S7       2         2
## S8       2         3
## S9       3         4
## S10      3         2
## S11      3         3
## S12      3         1
```

---

6. Basic statistical summaries in R

6.1 Calculate simple statistics

Let's start data with one protein as an example and calculate the mean, standard deviation, standard error of the mean across all replicates per condition. We then store all the computed statistics into a single summary data frame for easy access.

We can use the **aggregate** function to compute summary statistics

```
# check what proteins are in dataset, show all protein names
unique(iprg$Protein)
```

```
# Let's start with one protein, named "sp|P44015|VAC2_YEAST"
oneproteindata <- iprg[iprg$Protein == "sp|P44015|VAC2_YEAST", ]
```

```r
# there are 12 rows in oneproteindata
oneproteindata
```

```
##                     Protein Log2Intensity                       Run
## 21096 sp|P44015|VAC2_YEAST      26.30163 JD_06232014_sample1_B.raw
## 21097 sp|P44015|VAC2_YEAST      26.11643 JD_06232014_sample1_C.raw
## 21098 sp|P44015|VAC2_YEAST      26.29089 JD_06232014_sample1-A.raw
## 21099 sp|P44015|VAC2_YEAST      25.81957 JD_06232014_sample2_A.raw
## 21100 sp|P44015|VAC2_YEAST      26.11527 JD_06232014_sample2_B.raw
## 21101 sp|P44015|VAC2_YEAST      26.08498 JD_06232014_sample2_C.raw
## 21102 sp|P44015|VAC2_YEAST      23.14806 JD_06232014_sample3_A.raw
## 21103 sp|P44015|VAC2_YEAST      23.32465 JD_06232014_sample3_B.raw
## 21104 sp|P44015|VAC2_YEAST      23.29555 JD_06232014_sample3_C.raw
## 21105 sp|P44015|VAC2_YEAST      20.94536 JD_06232014_sample4_B.raw
## 21106 sp|P44015|VAC2_YEAST      21.71424 JD_06232014_sample4_C.raw
## 21107 sp|P44015|VAC2_YEAST      20.25209 JD_06232014_sample4-A.raw
##        Condition BioReplicate Intensity
## 21096 Condition1            1  82714388
## 21097 Condition1            1  72749239
## 21098 Condition1            1  82100518
## 21099 Condition2            2  59219741
## 21100 Condition2            2  72690802
## 21101 Condition2            2  71180513
## 21102 Condition3            3   9295260
## 21103 Condition3            3  10505591
## 21104 Condition3            3  10295788
## 21105 Condition4            4   2019205
## 21106 Condition4            4   3440629
## 21107 Condition4            4   1248781
```

6.1.1 Calculate mean per groups

```r
# splits 'oneproteindata' into subsets by 'Condition',
# then, compute 'FUN=mean' of 'log2Int'
sub.mean <- aggregate(Log2Intensity ~ Condition, data=oneproteindata, FUN=mean)
sub.mean
```

```
##    Condition Log2Intensity
## 1 Condition1      26.23632
## 2 Condition2      26.00661
## 3 Condition3      23.25609
## 4 Condition4      20.97056
```

6.1.2 Calculate SD(standard deviation) per groups

```r
# The same as mean calculation above. 'FUN' is changed to 'sd'.
sub.sd <- aggregate(Log2Intensity ~ Condition, data=oneproteindata, FUN=sd)
sub.sd
```

```
##    Condition Log2Intensity
## 1 Condition1    0.10396539
## 2 Condition2    0.16268179
## 3 Condition3    0.09467798
## 4 Condition4    0.73140174
```

6.1.3 Count the number of observation per groups

```
# The same as mean calculation. 'FUN' is changed 'length'.
sub.len <- aggregate(Log2Intensity ~ Condition, data=oneproteindata, FUN=length)
sub.len
```

```
##    Condition Log2Intensity
## 1 Condition1             3
## 2 Condition2             3
## 3 Condition3             3
## 4 Condition4             3
```

6.1.4 Calculate SE(standard error of mean) per groups

$SE = \sqrt{\frac{s^2}{n}}$

```
sub.se <- sqrt(sub.sd$Log2Intensity^2/sub.len$Log2Intensity)
sub.se
```

```
## [1] 0.06002444 0.09392438 0.05466236 0.42227499
```

```
# make the summary table including the results above (mean, sd, se and length).
summaryresult <- data.frame(Group=c("Condition1", "Condition2", "Condition3", "Condition4"),
                            mean=sub.mean$Log2Intensity,
                            sd=sub.sd$Log2Intensity,
                            se=sub.se,
                            length=sub.len$Log2Intensity)
summaryresult
```

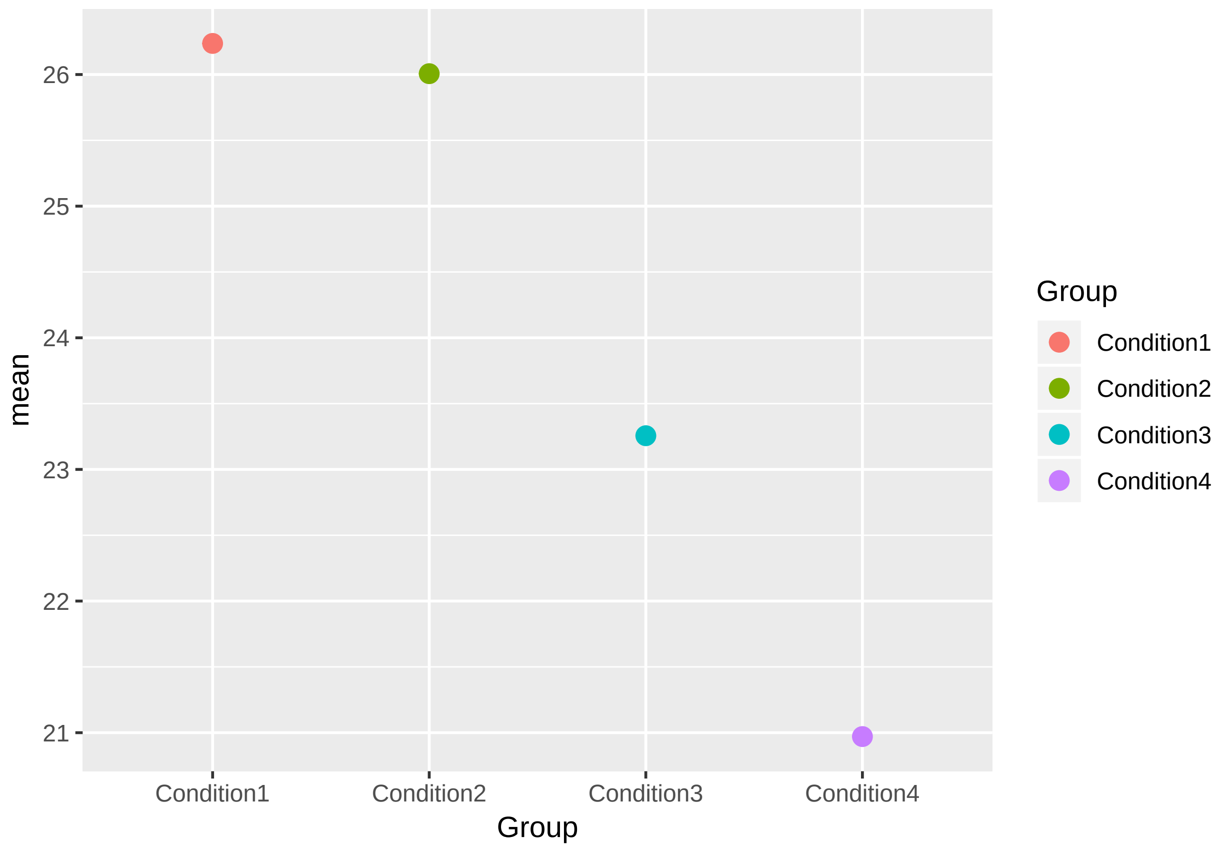
```
##        Group     mean         sd         se length
## 1 Condition1 26.23632 0.10396539 0.06002444      3
## 2 Condition2 26.00661 0.16268179 0.09392438      3
## 3 Condition3 23.25609 0.09467798 0.05466236      3
## 4 Condition4 20.97056 0.73140174 0.42227499      3
```

6.2 Visualization with error bars for descriptive purpose

'error bars' can have a variety of meanings or conclusions if what they represent is not precisely specified. Below we provide some examples of which types of error bars are common. We're using the summary of protein `sp|P44015|VAC2_YEAST` from the previous section and the `ggplot2` package as it provides a convenient way to make easily adaptable plots.

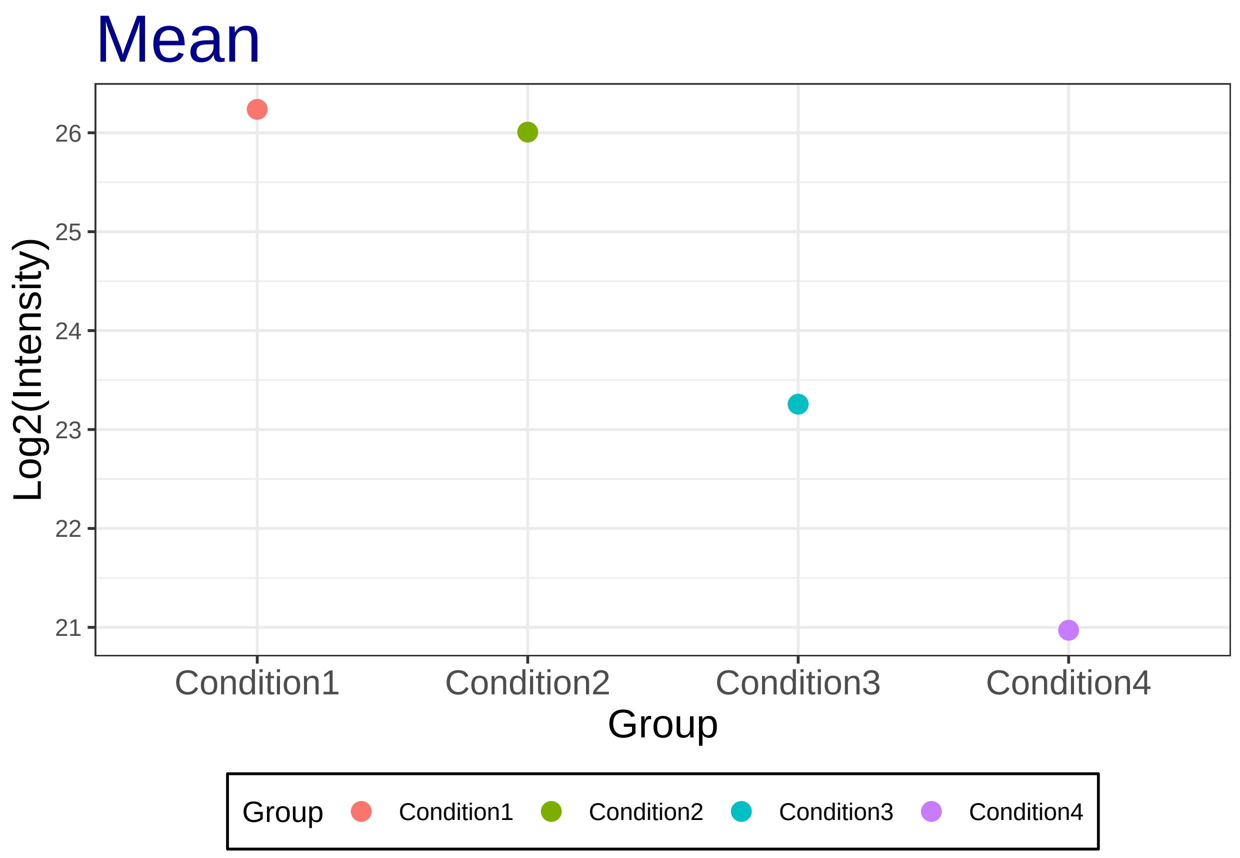
```
# Let's draw plots with mean and error bars
library(ggplot2)

# means without any errorbar
ggplot(aes(x=Group, y=mean, colour=Group), data=summaryresult)+
      geom_point(size=3)
```

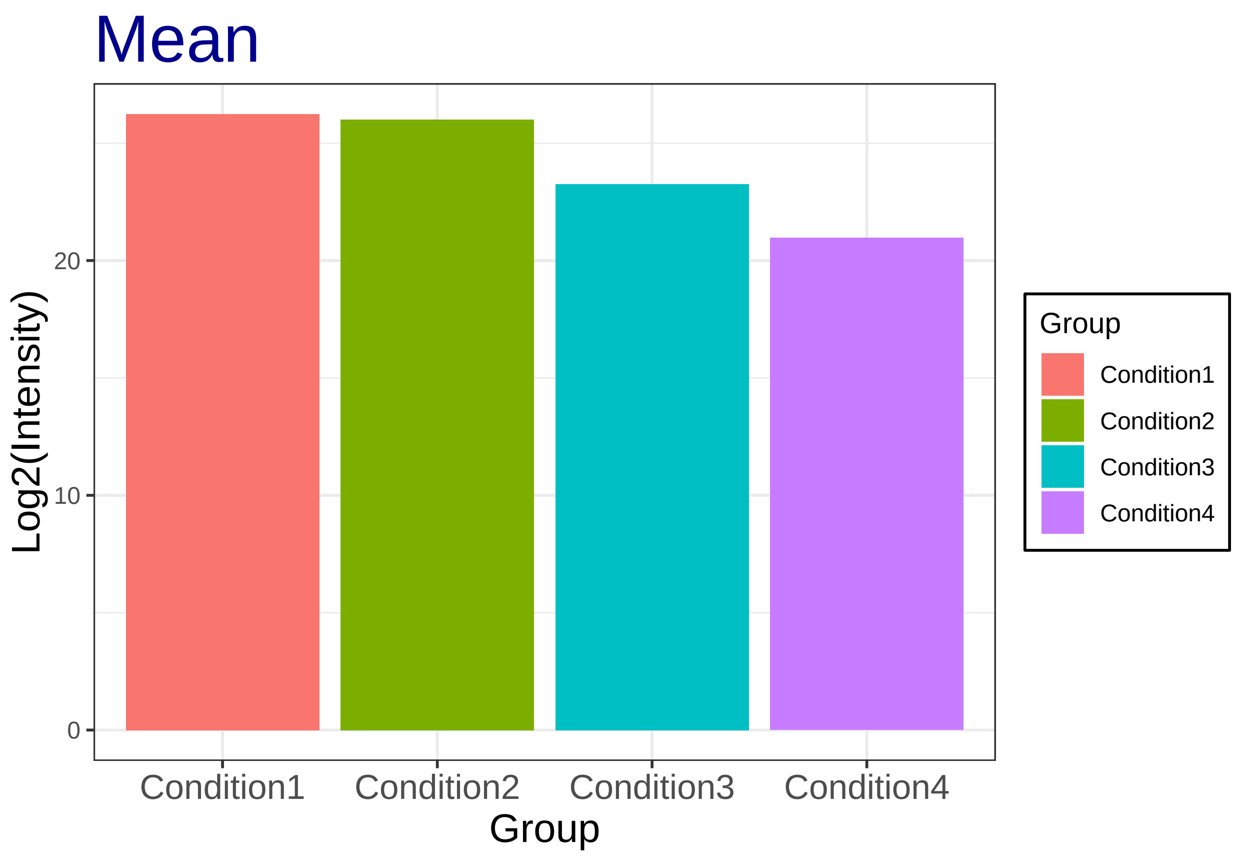


```
# Let's change a number of visual properties to make the plot more atttractive
# Let's change the labels of x-axis and y-axis and title:
# add labs(title="Mean", x="Condition", y='Log2(Intensity)')
# Let's change background color for white : add theme_bw()
# Let's change size or color of labels of axes and title, text of x-axis : in theme
# Let's change the position of legend :'none' remove the legend
# Let's make the box for legend
# Let's remove the box for legend key.

ggplot(aes(x=Group, y=mean, colour=Group), data=summaryresult)+
      geom_point(size=3)+
      labs(title="Mean", x="Group", y='Log2(Intensity)')+
      theme_bw()+
      theme(plot.title = element_text(size=25, colour="darkblue"),
            axis.title.x = element_text(size=15),
            axis.title.y = element_text(size=15),
            axis.text.x = element_text(size=13),
            legend.position = 'bottom',
            legend.background = element_rect(colour='black'),
            legend.key = element_rect(colour='white'))
```

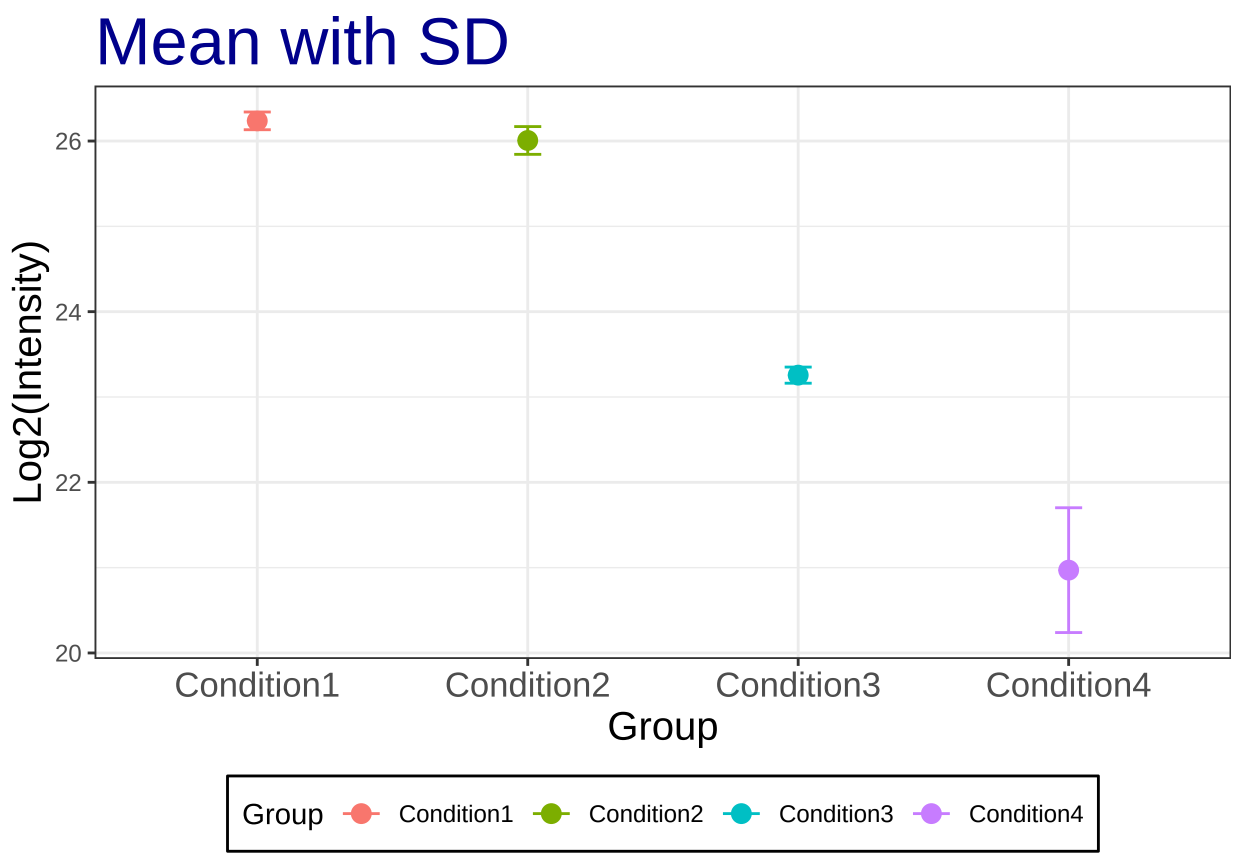


```
# Very similar but now as a bar plot.
ggplot(aes(x=Group, y=mean, fill=Group), data=summaryresult)+
      geom_bar(position=position_dodge(), stat='identity')+
      scale_x_discrete('Group')+
      labs(title="Mean", x="Group", y='Log2(Intensity)')+
      theme_bw()+
      theme(plot.title = element_text(size=25, colour="darkblue"),
            axis.title.x = element_text(size=15),
            axis.title.y = element_text(size=15),
            axis.text.x = element_text(size=13),
            legend.background = element_rect(colour='black'),
            legend.key = element_rect(colour='white'))
```

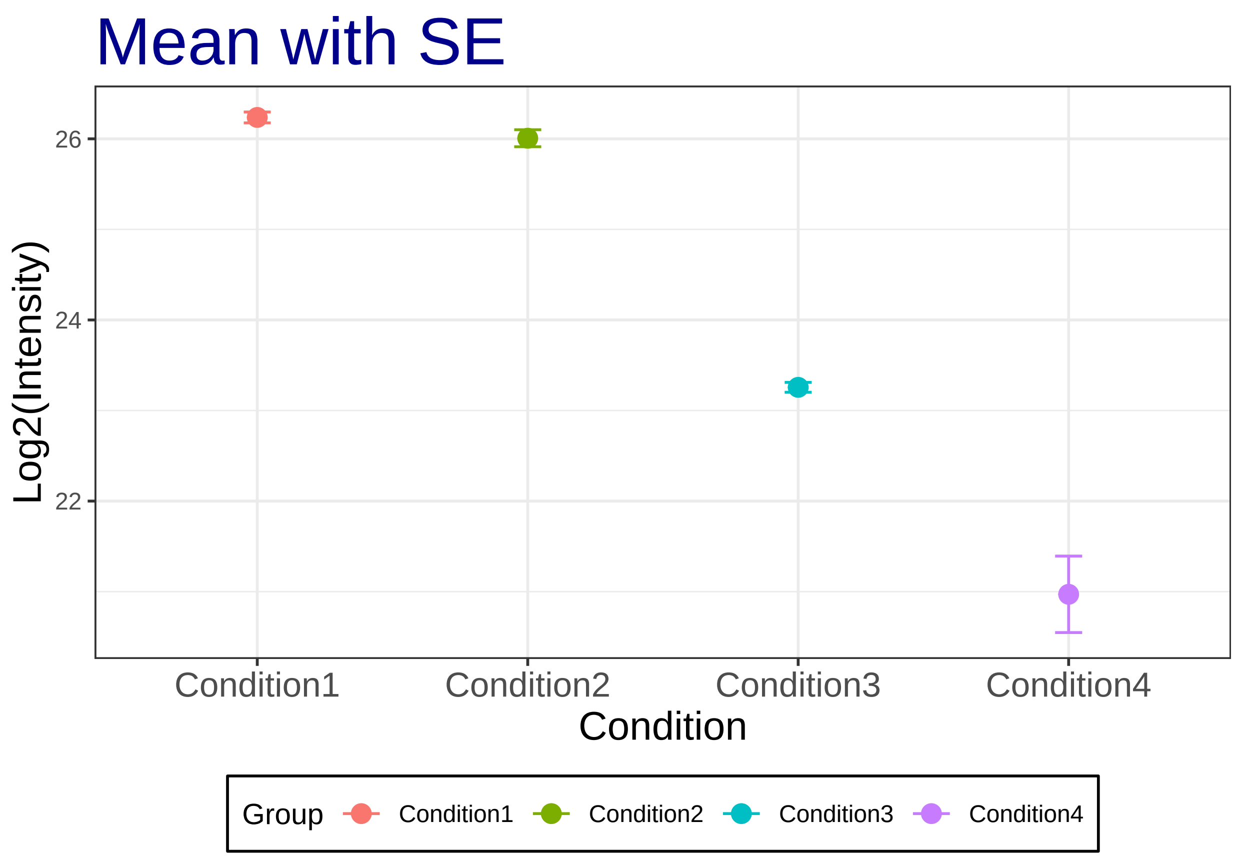


For the sake of this tutorial we'll continue adding error bars for different statistics with the point plots. We'll leave it as an exercise to add error bars to the barplots. Let's first add the standard deviation, then the standard error of the mean. Which one is smaller?

```
# mean with SD
ggplot(aes(x=Group, y=mean, colour=Group), data=summaryresult)+
      geom_point(size=3)+
      geom_errorbar(aes(ymax = mean + sd, ymin=mean - sd), width=0.1)+
      scale_x_discrete('Group')+
      labs(title="Mean with SD", x="Group", y='Log2(Intensity)')+
      theme_bw()+
      theme(plot.title = element_text(size=25, colour="darkblue"),
            axis.title.x = element_text(size=15),
            axis.title.y = element_text(size=15),
            axis.text.x = element_text(size=13),
            legend.position = 'bottom',
            legend.background = element_rect(colour='black'),
            legend.key = element_rect(colour='white'))
```



```r
# mean with SE
ggplot(aes(x=Group, y=mean, colour=Group), data=summaryresult)+
      geom_point(size=3)+
      geom_errorbar(aes(ymax = mean + se, ymin=mean - se), width=0.1)+
      labs(title="Mean with SE", x="Condition", y='Log2(Intensity)')+
      theme_bw()+
      theme(plot.title = element_text(size=25, colour="darkblue"),
            axis.title.x = element_text(size=15),
            axis.title.y = element_text(size=15),
            axis.text.x = element_text(size=13),
            legend.position = 'bottom',
            legend.background = element_rect(colour='black'),
            legend.key = element_rect(colour='white'))
```



Note : The SE is narrow than the SD!

6.3 Calculate the confidence interval

Now that we've covered the standard error of the mean and the standard deviation, let's investigate how we can add custom confidence intervals (CI) for our measurement of the mean. We'll add these CI's to the summary results we previously stored for protein `sp|P44015|VAC2_YEAST`

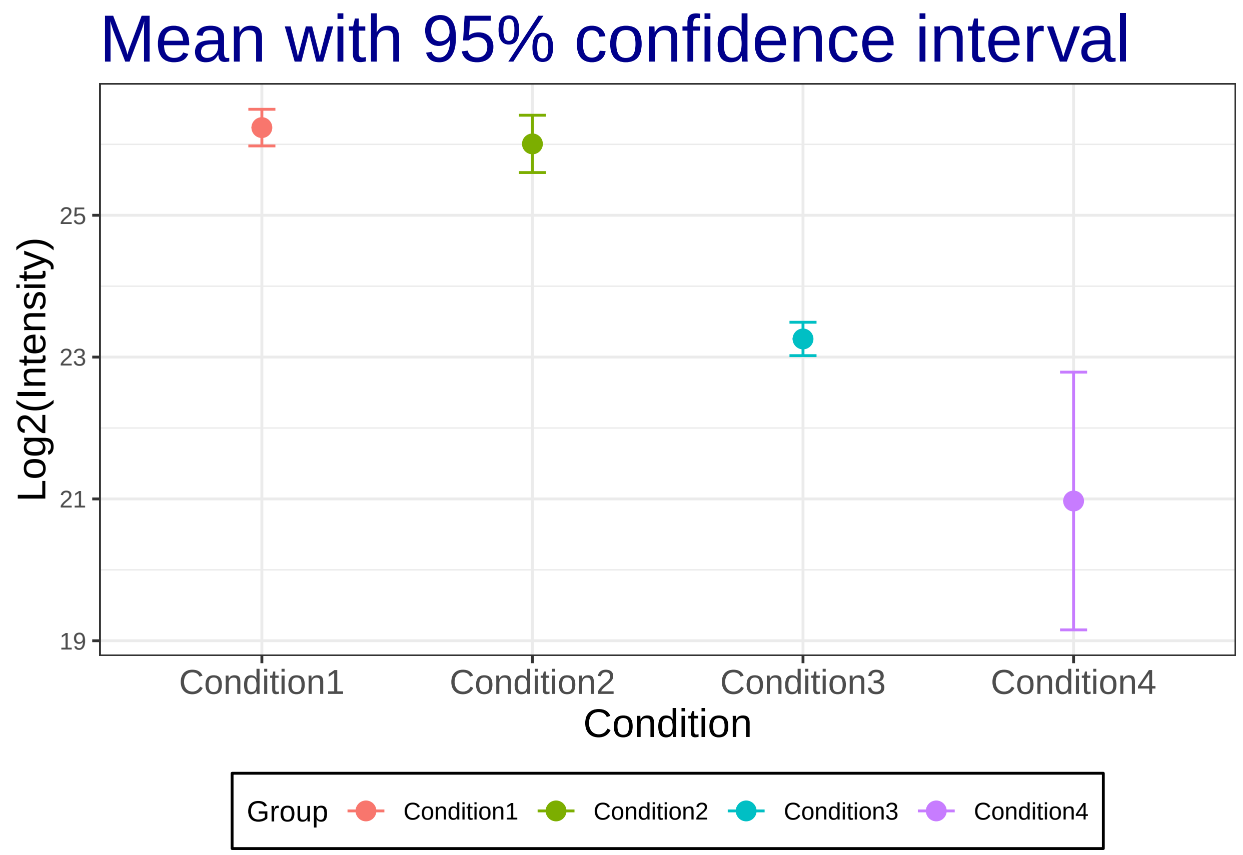
Confidence interval : mean $\pm$ $(SE \times \alpha/2$ quantile of t distribution)

```
# 95% confident interval
# Be careful for setting quantile for two-sided. need to divide by two for error.
# For example, 95% confidence interval, right tail is 2.5% and left tail is 2.5%.

summaryresult$ciw.lower.95 <- summaryresult$mean - qt(0.975,summaryresult$len-1)*summaryresult$se
summaryresult$ciw.upper.95 <- summaryresult$mean + qt(0.975,summaryresult$len-1)*summaryresult$se
summaryresult
```

```
##        Group     mean         sd         se length ciw.lower.95
## 1 Condition1 26.23632 0.10396539 0.06002444      3     25.97805
## 2 Condition2 26.00661 0.16268179 0.09392438      3     25.60248
## 3 Condition3 23.25609 0.09467798 0.05466236      3     23.02090
## 4 Condition4 20.97056 0.73140174 0.42227499      3     19.15366
##   ciw.upper.95
## 1     26.49458
## 2     26.41073
## 3     23.49128
## 4     22.78746
```

```
# mean with 95% two-sided confidence interval
ggplot(aes(x=Group, y=mean, colour=Group), data=summaryresult)+
      geom_point(size=3)+
      geom_errorbar(aes(ymax = ciw.upper.95, ymin=ciw.lower.95), width=0.1)+
      labs(title="Mean with 95% confidence interval", x="Condition", y='Log2(Intensity)')+
      theme_bw()+
      theme(plot.title = element_text(size=25, colour="darkblue"),
            axis.title.x = element_text(size=15),
            axis.title.y = element_text(size=15),
            axis.text.x = element_text(size=13),
            legend.position = 'bottom',
            legend.background = element_rect(colour='black'),
            legend.key = element_rect(colour='white'))
```

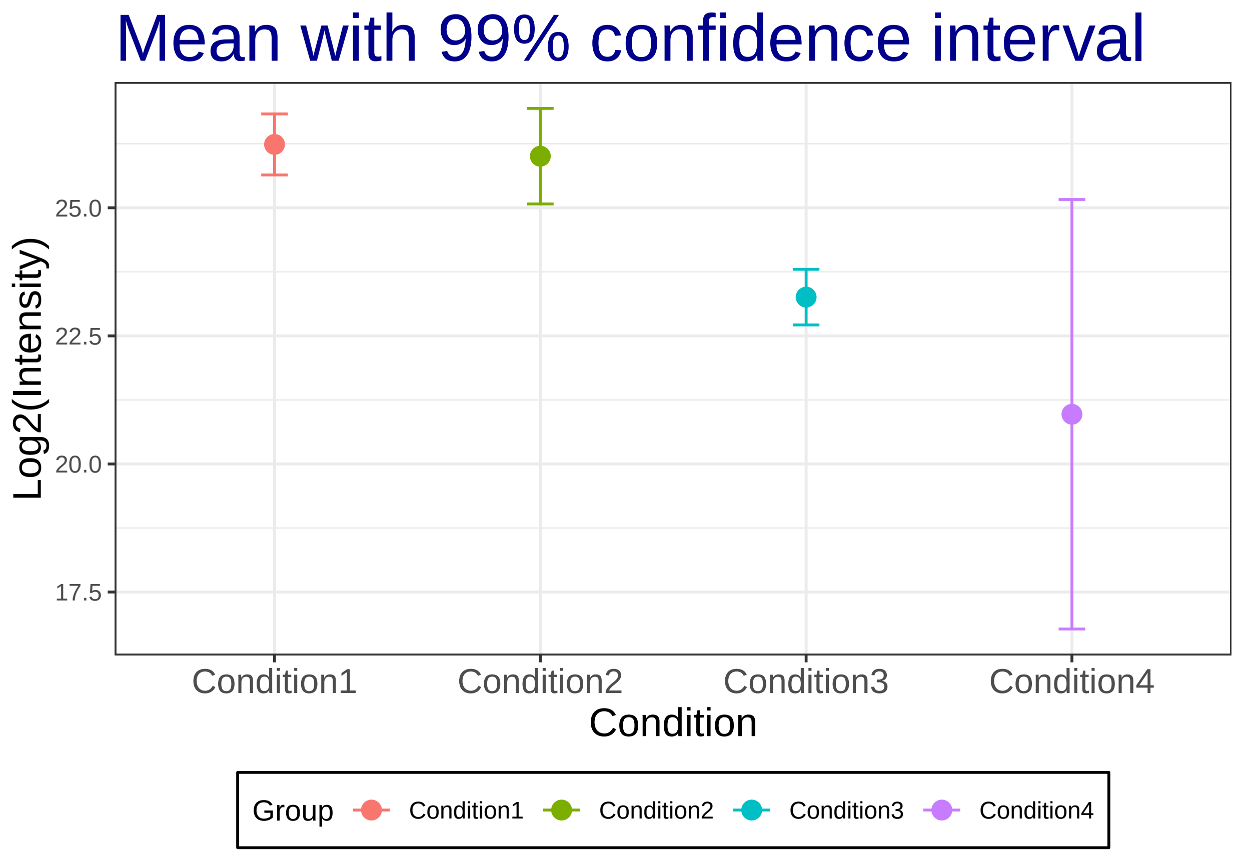


Let's repeat that one more time for the 99% two-sided confidence interval.

```
# mean with 99% two-sided confidence interval
summaryresult$ciw.lower.99 <- summaryresult$mean - qt(0.995,summaryresult$len-1)*summaryresult$se
summaryresult$ciw.upper.99 <- summaryresult$mean + qt(0.995,summaryresult$len-1)*summaryresult$se
summaryresult
```

```
##        Group     mean         sd         se length ciw.lower.95
## 1 Condition1 26.23632 0.10396539 0.06002444      3     25.97805
## 2 Condition2 26.00661 0.16268179 0.09392438      3     25.60248
## 3 Condition3 23.25609 0.09467798 0.05466236      3     23.02090
## 4 Condition4 20.97056 0.73140174 0.42227499      3     19.15366
##   ciw.upper.95 ciw.lower.99 ciw.upper.99
## 1     26.49458     25.64058     26.83205
## 2     26.41073     25.07442     26.93879
## 3     23.49128     22.71357     23.79860
## 4     22.78746     16.77955     25.16157
```

```
ggplot(aes(x=Group, y=mean, colour=Group), data=summaryresult)+
      geom_point(size=3)+
      geom_errorbar(aes(ymax = ciw.upper.99, ymin=ciw.lower.99), width=0.1)+
      labs(title="Mean with 99% confidence interval", x="Condition", y='Log2(Intensity)')+
      theme_bw()+
      theme(plot.title = element_text(size=25, colour="darkblue"),
            axis.title.x = element_text(size=15),
            axis.title.y = element_text(size=15),
            axis.text.x = element_text(size=13),
            legend.position = 'bottom',
            legend.background = element_rect(colour='black'),
            legend.key = element_rect(colour='white'))
```



comment : Let's compare all three versions of the means with different error bars. Every gap between error bars is different. Furthermore, error bars with SD and CI are overlapping between groups! Error bars for SD show the spread of the population while error bars based on SE reflect the uncertainty in the mean and depend on the sample size. Confidence intervals of `n` on the other hand mean that the interval captures the population mean `n %` of the time. When the sample size increases, CI and SE are getting closer to each other.

---

7. Statistical hypothesis test in R

Two sample t-test for one protein with one feature

Next, we'll perform a t-test whether protein `sp|P44015|VAC2_YEAST` has a change in abundance between Condition 1 and Condition 2.

Hypothesis :

H_0 : 'Status quo', no change in abundance, Condition1 - Condition2 = 0

H_a : change in abundance, Condition1 - Condition 2 $\neq 0$

observed $t = \frac{\textbf{difference of group means}}{\textbf{estimate of variation}} = \frac{(mean_{Condition1} - mean_{Condition2})}{SE} \sim t_{\alpha/2,df}$

Standard error, $SE = \sqrt{\frac{s^2_{Condition1}}{n_{Condition1}} + \frac{s^2_{Condition2}}{n_{Condition2}}}$

$n_{Condition1}$: Number of replicates

$s^2_{Condition1} = \frac{1}{n_{Condition1}-1}\sum(Y_{1i} - \bar{Y_{1\cdot}})^2$: Sample variance

```
# First, get two conditions only, because t.test only works for two groups (conditions).
oneproteindata.condition12 <- oneproteindata[which(oneproteindata$Condition %in%
                                              c('Condition1', 'Condition2')), ]
unique(oneproteindata.condition12$Condition)
```

```
## [1] Condition1 Condition2
## Levels: Condition1 Condition2 Condition3 Condition4
```

```
unique(oneproteindata$Condition)
```

```
## [1] Condition1 Condition2 Condition3 Condition4
## Levels: Condition1 Condition2 Condition3 Condition4
```

```
# t test for different abundance (log2Int) between Groups (Condition)
result <- t.test(oneproteindata.condition12$Log2Intensity ~ oneproteindata.condition12$Condition,
                 var.equal=FALSE)
# show the summary of t-test including confidence interval level with 0.95
result
```

```
##
##  Welch Two Sample t-test
##
## data:  oneproteindata.condition12$Log2Intensity by oneproteindata.condition12$Condition
## t = 2.0608, df = 3.4001, p-value = 0.1206
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1025408  0.5619598
## sample estimates:
## mean in group Condition1 mean in group Condition2
##                 26.23632                 26.00661
```

We can redo the t-test and change the confidence interval level for the log2 fold change.

```
result.ci90 <- t.test(oneproteindata.condition12$Log2Intensity ~ oneproteindata.condition12$Condition,
                      var.equal=FALSE,
                      conf.level=0.9)
result.ci90
```

```
##
##  Welch Two Sample t-test
##
## data:  oneproteindata.condition12$Log2Intensity by oneproteindata.condition12$Condition
## t = 2.0608, df = 3.4001, p-value = 0.1206
## alternative hypothesis: true difference in means is not equal to 0
## 90 percent confidence interval:
##  -0.02049268  0.47991165
## sample estimates:
## mean in group Condition1 mean in group Condition2
##                 26.23632                 26.00661
```

Let's have a more detailed look at what information we can learn from the results our t-test.

```
# name of output
names(result)
```

```
## [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name"
```

```
# mean for each group
result$estimate
```

```
## mean in group Condition1 mean in group Condition2
##                 26.23632                 26.00661
```

```
# log2 transformed fold change between groups : Disease-Healthy
result$estimate[1]-result$estimate[2]
```

```
## mean in group Condition1
##                0.2297095
```

```
# test statistic value, T value
result$statistic
```

```
##        t
## 2.060799
```

```
# standard error
(result$estimate[1]-result$estimate[2])/result$statistic
```

```
## mean in group Condition1
##                0.1114662
```

```
# degree of freedom
result$parameter
```

```
##       df
## 3.400112
```

```
# p value for two-sides testing
result$p.value
```

```
## [1] 0.1206139
```

```
# 95% confidence interval for log2 fold change
result$conf.int
```

```
## [1] -0.1025408  0.5619598
## attr(,"conf.level")
## [1] 0.95
```

```r
# p value calculation for one side
1-pt(result$statistic, result$parameter)
```

```
##          t
## 0.06030697
```

```r
# p value for two sides, which is the same as pvalue from t test (result$p.value)
2*(1-pt(result$statistic, result$parameter))
```

```
##         t
## 0.1206139
```

We can also manually compute our t-test statistic using the formulas we descibed above and compare it with the `summaryresult`.

```r
summaryresult12 <- summaryresult[1:2, ]

# test statistic, It is the same as 'result$statistic' above.
diff(summaryresult12$mean) # same as result$estimate[1]-result$estimate[2]
```

```
## [1] -0.2297095
```

```r
sqrt(sum(summaryresult12$sd^2/summaryresult12$length)) # same as stand error
```

```
## [1] 0.1114662
```

```r
diff(summaryresult12$mean)/sqrt(sum(summaryresult12$sd^2/summaryresult12$length))
```

```
## [1] -2.060799
```

---

8. Saving your work

You can save plots to a number of different file formats. PDF is by far the most common format because it's lightweight, cross-platform and scales up well but jpegs, pngs and a number of other file formats are also supported. Let's redo the last barplot but save it to the file system this time.

Let's save the boxplot as pdf file.

```r
pdf('boxplot_log2intensity_distribution_byMSrun.pdf', width=10, height=8)
ggplot(aes_string(x='Run', y='Log2Intensity'), data=iprg)+
            geom_boxplot(aes_string(fill='Condition'))+
          labs(title='Log2 transformed intensity distribution per MS run',
               y='Log2(Intensity)',
               x='MS run')+
          theme(axis.text.x=element_text(angle=90))
dev.off()
```

```
## pdf
##   2
```

Finally, we can save this whole session you worked so hard on!

```r
save.image(file='Day1.RData')
```

Ok let's give it a rest for today. Saving an .RData is the easiest way to pick up your work right where you left it!

```r
rm(list=ls())
load(file = 'Day1.RData')
```