ABRF 2017 Satellite workshop - Hands-on 1 : Introduction to R and experimental design

Meena Choi and Ting Huang

3/25/2017

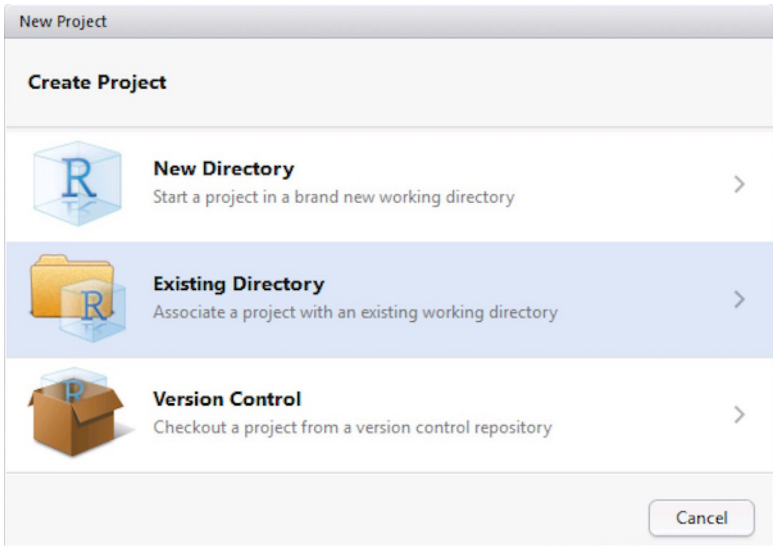
Summary

- 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.

---

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.

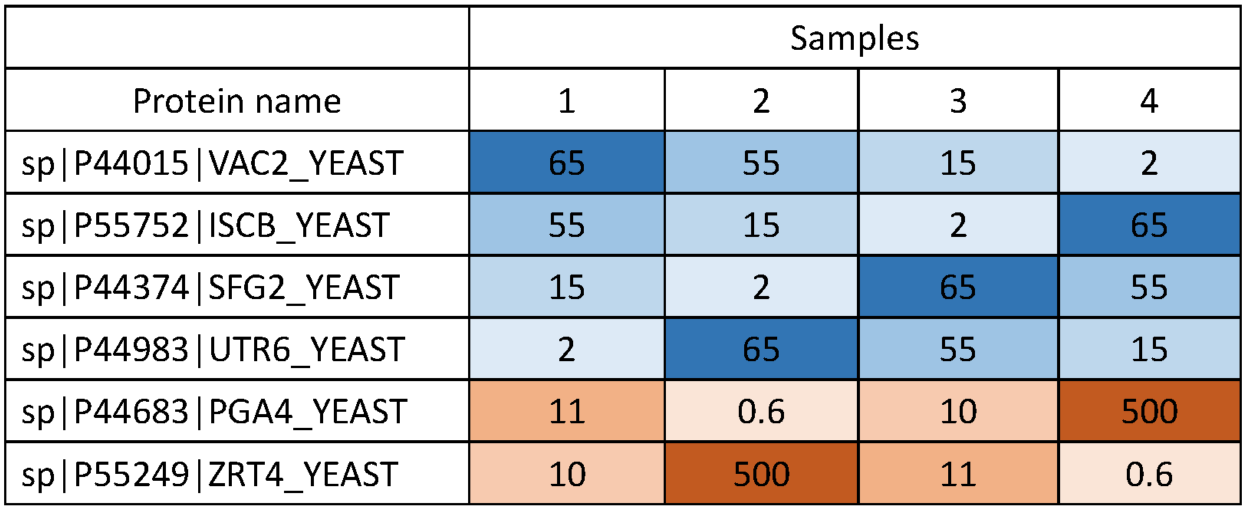
```r
getwd()
```

```
## [1] "/Users/meenachoi/Dropbox/visits/2017/03/ABRF/ABRF2017_wTing/Handson1"
```

---

2. Reading in data

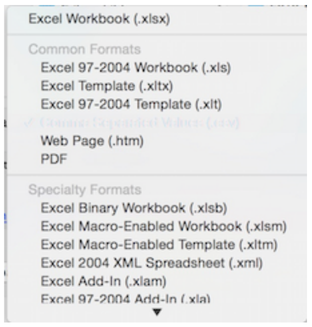
The file we'll be reading in is a dataset that has been 1) processed in Skyline and 2) summarized by each run and protein with `MSstats`. We will practice with it.



	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: Spike-in protein information

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.

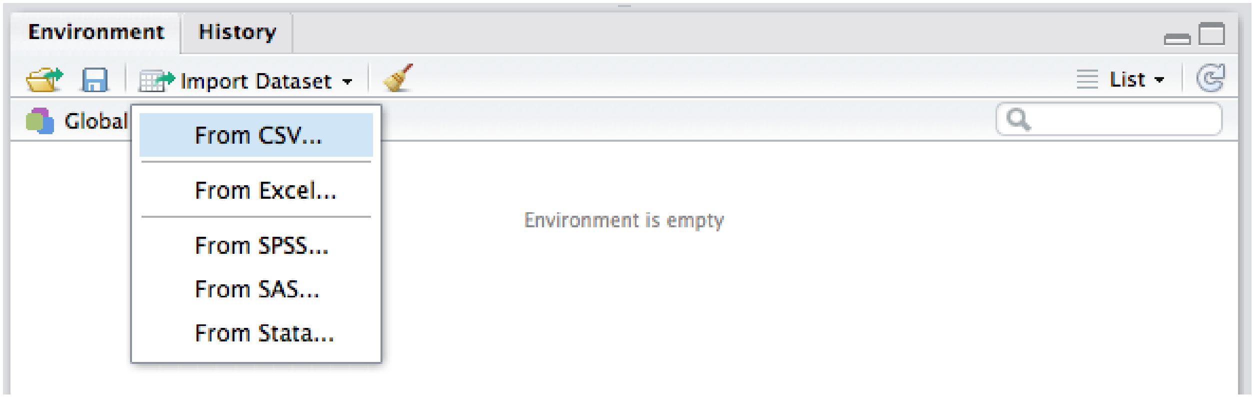


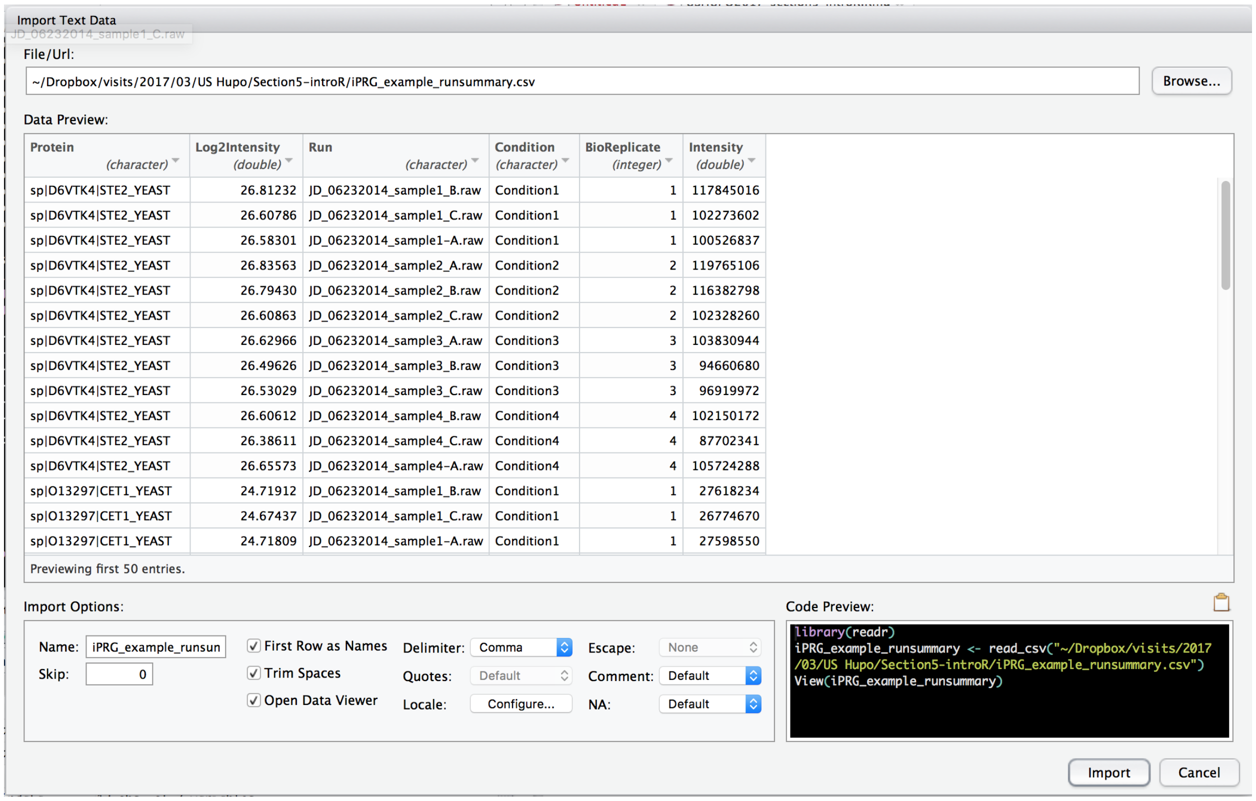
Excel Workbook (.xlsx)

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 `RatPlasmaData.csv` file and inspect that Rstudio correctly parsed the text file into an R `data frame`.




Import Text Data

File/Url:

~/Dropbox/visits/2017/03/US Hupo/Section5-introR/iPRG_example_runsummary.csv

Browse...

Data Preview:

Protein (character)	Log2Intensity (double)	Run (character)	Condition (character)	BioReplicate (integer)	Intensity (double)
sp|D6VTK4|STE2_YEAST	26.81232	JD_06232014_sample1_B.raw	Condition1	1	117845016
sp|D6VTK4|STE2_YEAST	26.60786	JD_06232014_sample1_C.raw	Condition1	1	102273602
sp|D6VTK4|STE2_YEAST	26.58301	JD_06232014_sample1-A.raw	Condition1	1	100526837
sp|D6VTK4|STE2_YEAST	26.83563	JD_06232014_sample2_A.raw	Condition2	2	119765106
sp|D6VTK4|STE2_YEAST	26.79430	JD_06232014_sample2_B.raw	Condition2	2	116382798
sp|D6VTK4|STE2_YEAST	26.60863	JD_06232014_sample2_C.raw	Condition2	2	102328260
sp|D6VTK4|STE2_YEAST	26.62966	JD_06232014_sample3_A.raw	Condition3	3	103830944
sp|D6VTK4|STE2_YEAST	26.49626	JD_06232014_sample3_B.raw	Condition3	3	94660680
sp|D6VTK4|STE2_YEAST	26.53029	JD_06232014_sample3_C.raw	Condition3	3	96919972
sp|D6VTK4|STE2_YEAST	26.60612	JD_06232014_sample4_B.raw	Condition4	4	102150172
sp|D6VTK4|STE2_YEAST	26.38611	JD_06232014_sample4_C.raw	Condition4	4	87702341
sp|D6VTK4|STE2_YEAST	26.65573	JD_06232014_sample4-A.raw	Condition4	4	105724288
sp|O13297|CET1_YEAST	24.71912	JD_06232014_sample1_B.raw	Condition1	1	27618234
sp|O13297|CET1_YEAST	24.67437	JD_06232014_sample1_C.raw	Condition1	1	26774670
sp|O13297|CET1_YEAST	24.71809	JD_06232014_sample1-A.raw	Condition1	1	27598550

Previewing first 50 entries.

Import Options:

Name: iPRG_example_runsun
Skip: 0
✓ First Row as Names
✓ Trim Spaces
✓ Open Data Viewer
Delimiter: Comma
Quotes: Default
Locale: Configure...
Escape: None
Comment: Default
NA: Default

Code Preview:

```
library(readr)
iPRG_example_runsummary <- read_csv("~/Dropbox/visits/2017
/03/US Hupo/Section5-introR/iPRG_example_runsummary.csv")
View(iPRG_example_runsummary)
```

Import Cancel

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.

---

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.

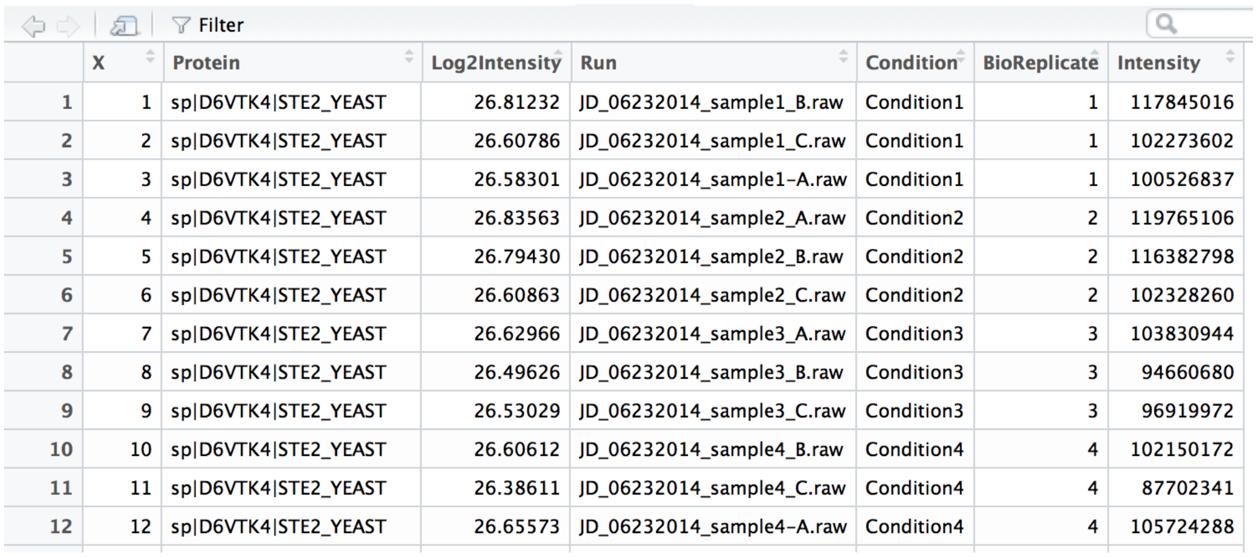
Ok, let's do some more data exploration by examining how R read in the iPRG dataset.

`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



Filter

	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 2: Example data for this section

```
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.

```
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.

```
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.

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

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

```
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.

```
# subset of data for condition1
iprg.condition1 <- iprg[iprg$Condition == 'Condition1', ]
iprg.condition1.bio1 <- iprg[iprg$Condition == 'Condition1'
```

```r
                               & iprg$BioReplicate == '1', ]
nrow(iprg.condition1.bio1)
```

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

```r
# subset of data for condition1 or condition2
iprg.condition1.2 <- iprg[iprg$Condition == 'Condition1'
                          | iprg$Condition == 'Condition2', ]
nrow(iprg.condition1.2)
```

```
## [1] 18160
```

```r
unique(iprg.condition1.2$Condition)
```

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

```r
# subset of data for condition1 or condition2
iprg.condition1.2 <- iprg[which(iprg$Condition %in% c('Condition1', 'Condition2')), ]
nrow(iprg.condition1.2)
```

```
## [1] 18160
```

```r
unique(iprg.condition1.2$Condition)
```

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

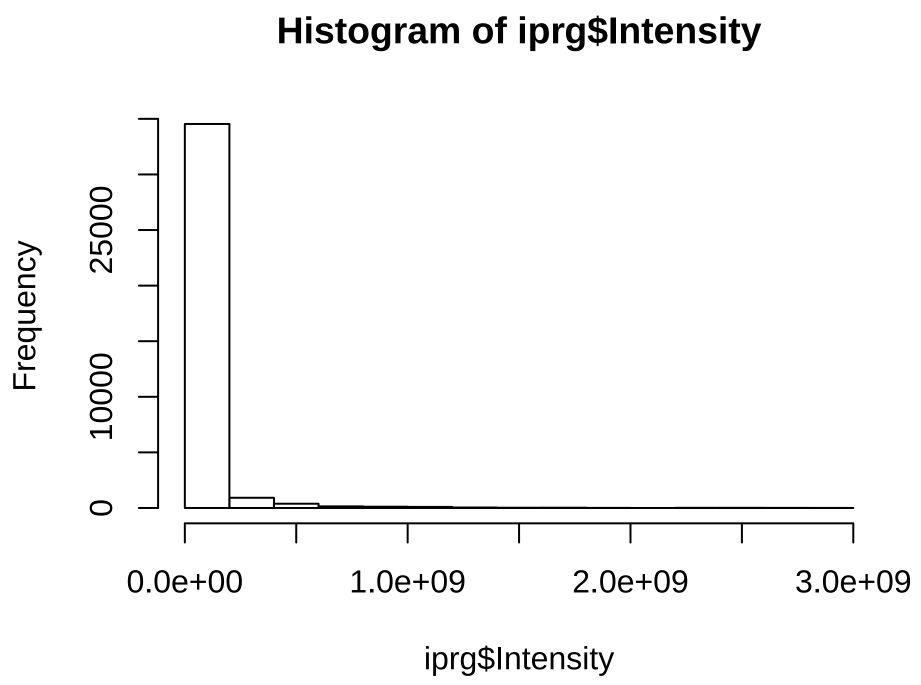
---

4. Summarizing and Visualizing data

4.1 Histogram

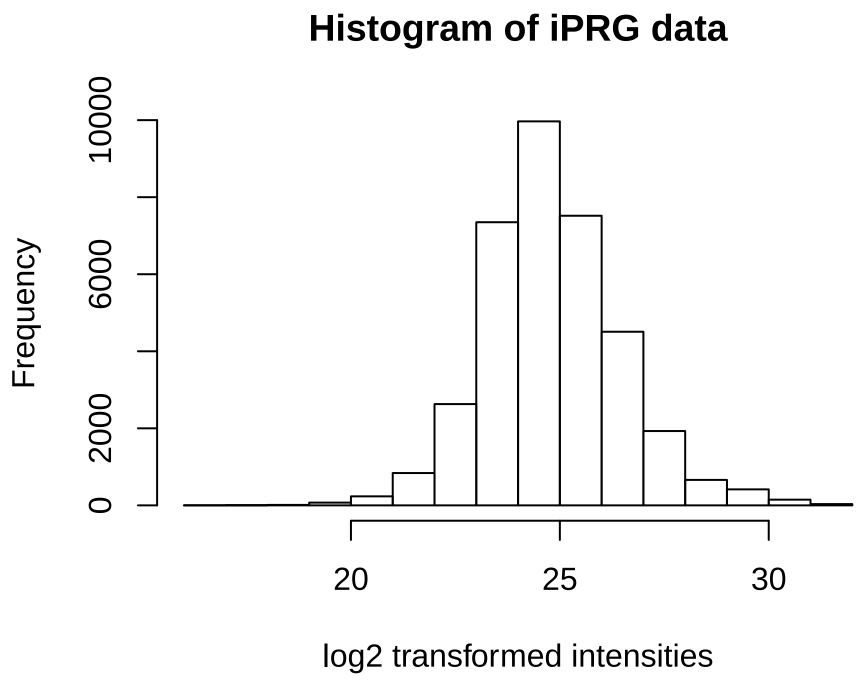
Make a histogram of all the MS1 intensities, quantified by Skyline, for `iPRG_example`.

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

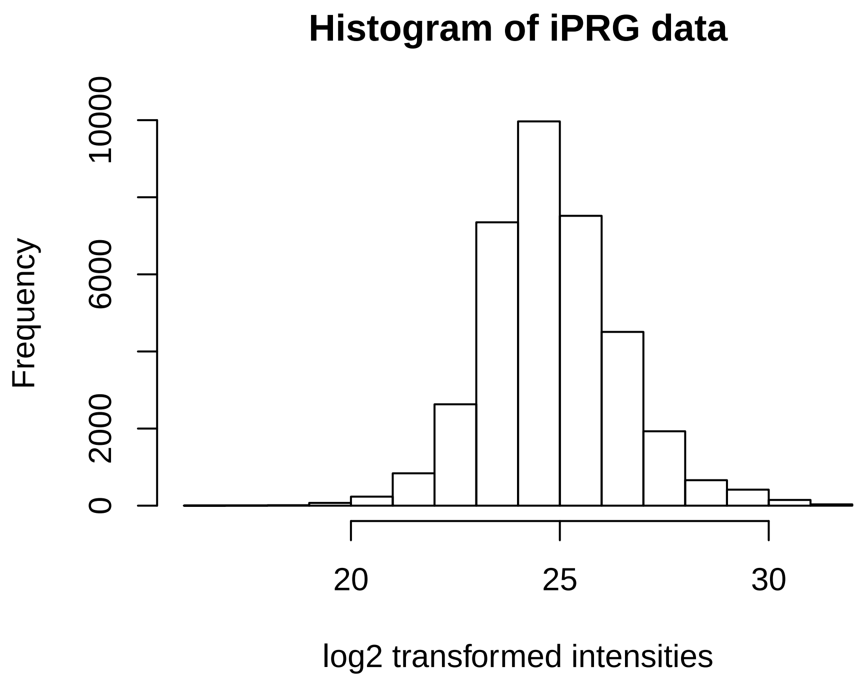


Our histogram looks quite skewed. How does this look on log-scale? Do you recognize this distribution? The distribution for log2-transformed intensities looks very similar to the normal distribution. The advantage of working with normally distributed data is that we can apply a variety of statistical 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.

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



```
# 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)
```

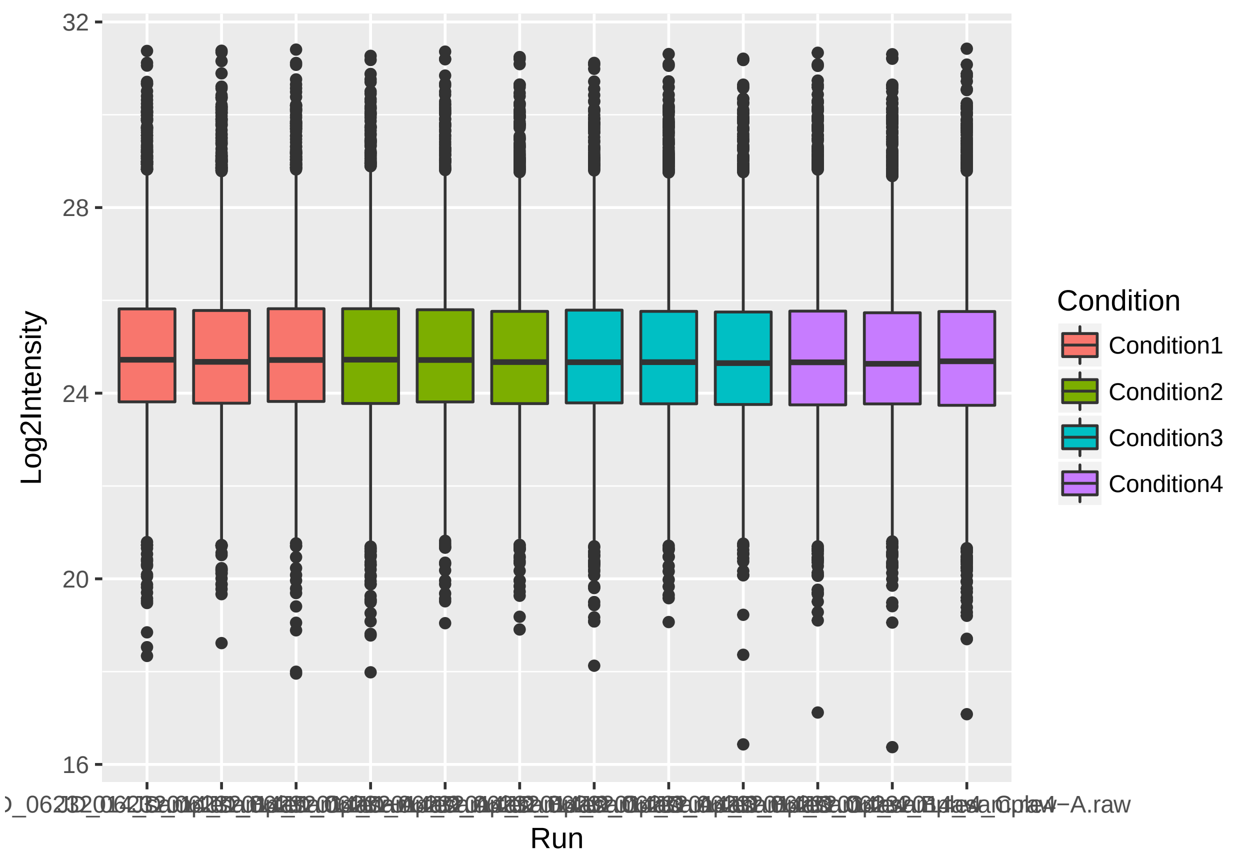
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   16.37   23.78   24.68   24.82   25.78   31.42
```

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.

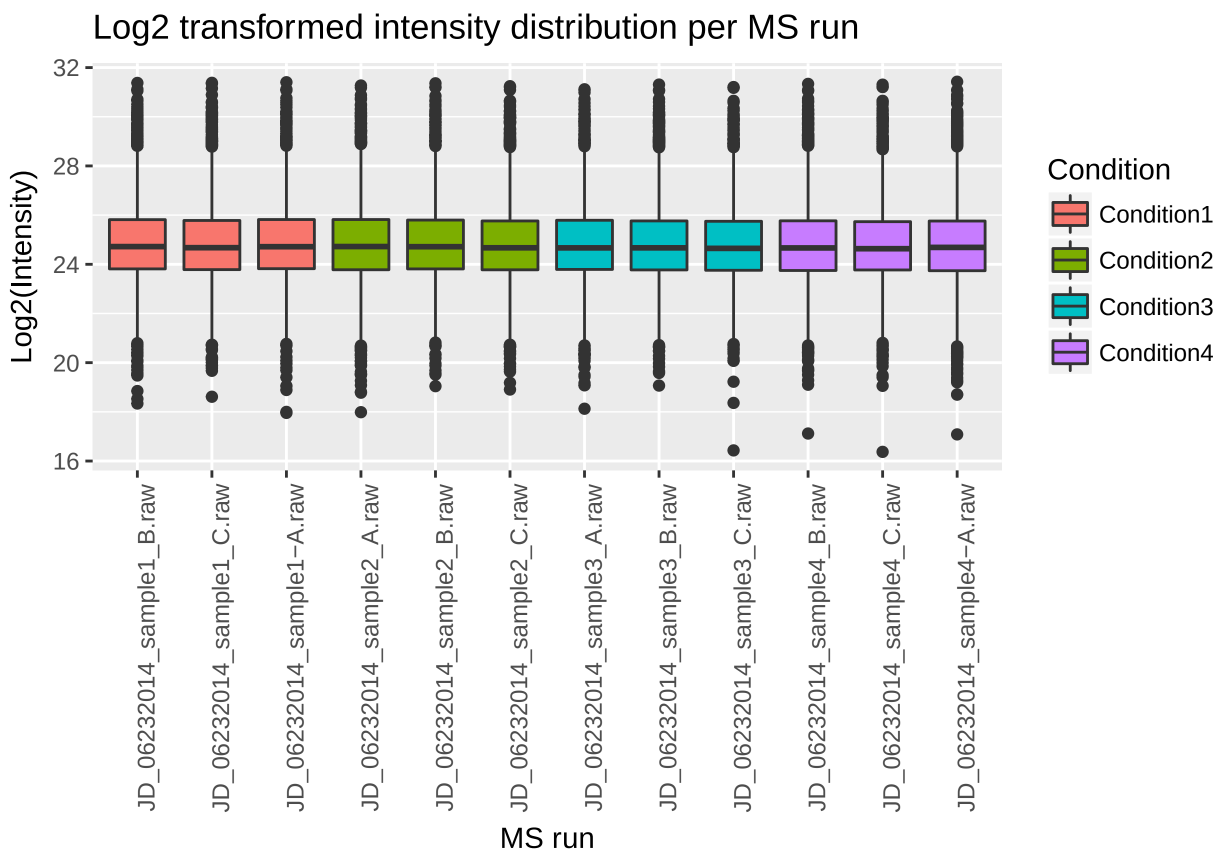
Let's make the boxplot with `ggplot2`, one of the most popular and powerful R packages for making graphics. The syntax of ggplot2 might seem a bit intimidating at first, but besides the advantage of having full control over all graphical elements of your plot, two other advantages are that 1) it's very 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'))
```



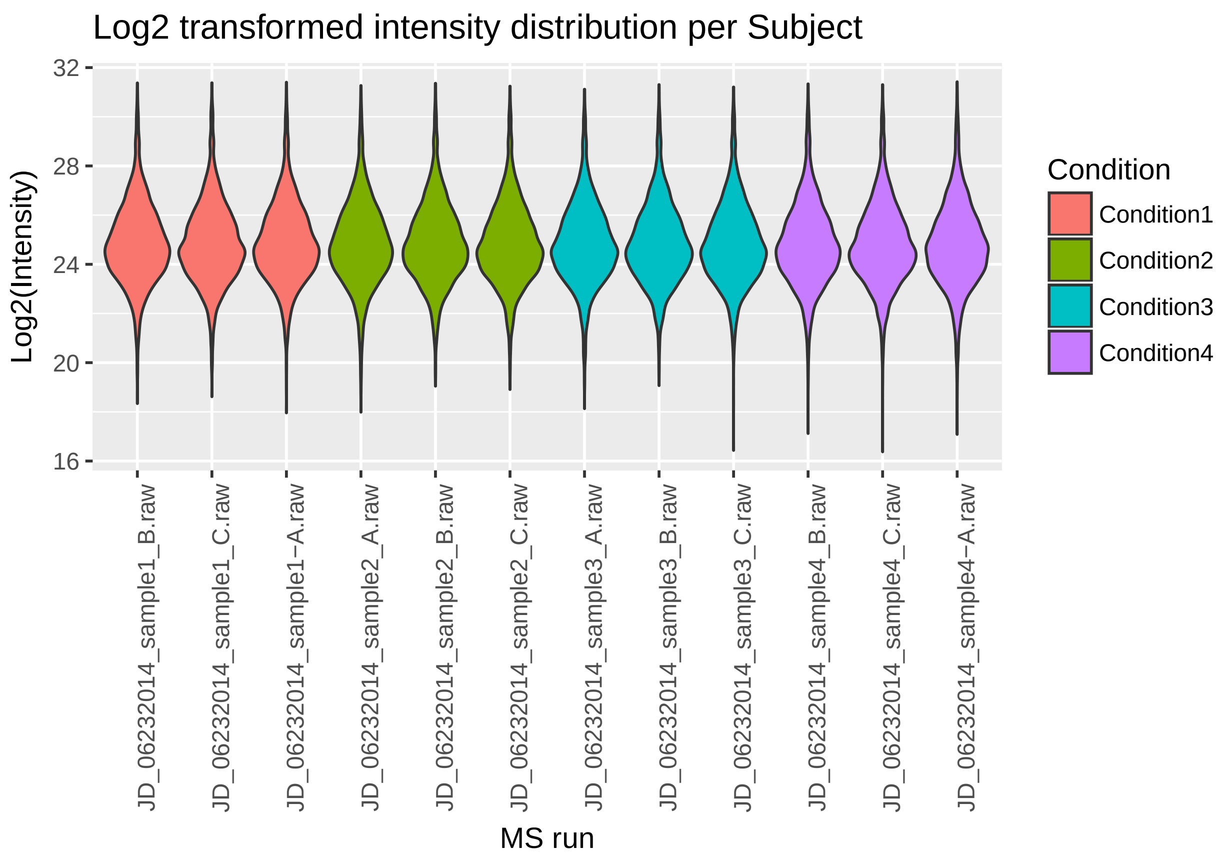
Now let's rename all axis labels and title, and rotate the x-axis labels 90 degrees. We can add those specifications using the `labs` and `theme` functions of the `ggplot2` package.

```
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]  9 10  2
```

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

```
## [1] 5 6 8
```

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
```

```r
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
```

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

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

6. 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!

```
save.image(file='Section1.RData')
```

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

```
rm(list=ls())
load(file = 'Section1.RData')
```