Introduction to Pre-Processing Serological Data

2025-05-22

Introduction

The purpose of this document is to provide an introduction to pre-processing your serological data. In general, pre-processing can be thought of as a series of steps that transform raw data (data directly from the scanner or imager in the lab) into data that can be used to do analysis and make inferences. After working through this lab exercise you should be able to:

- Understand how to read in raw data from a scanner or imager
- Filter the data using bead count
- Remove background signal from your fluorescence readings
- Visualize standard curves and where your cohort samples fall relative to these
- Fit a standard curve and estimate sample concentrations using this standard curve
- Estimate and remove plate-to-plate effects using a linear model

Review: General Housekeeping

Before we start, let's navigate to the appropriate working directory. You can accomplish this by navigating to the "Session" tab of RStudio, and choosing "Set Working Directory" –> "Choose Directory" and using your file browser to navigate to the `seroanalytics_workshop` folder. Alternatively, you can modify the code below as appropriate for your files to get to the `seroanalytics_workshop` folder.

```
# set my_path to be the working directory location of where
# the *seroanalytics_workshop* folder is stored on your
# computer
my_path <- "/OneDrive-JohnsHopkins/seroanalytics_workshop"

# source the functions file from the directory using the
# my_path location the functions file is saved in the
# Source/ folder within the working directory
source(paste(my_path, "Source/utils.R", sep = "/"))
```

Read In Files

We will begin by defining the plate setup as we did in Lab 2b. This includes antigen names, the names of control and blank or background wells, the standard curve sample names, dilution values and locations on plates, as well as replicate values if there happens to be more than one standard curve on each plate.

```
agx_names <- c("SNAP", "WNV", "YF", "JE3", "ZIKA", "DENV", "CHIKV",
    "GLURPR2", "CSP", "PfAMA1", "PfMSP119", "WRUV", "WMEV")

ctrls <- c("POS1", "POS2", "NEG1", "NEG2")

bg_samples <- c("BLANK1", "BLANK2")
# note that here we only have 1 standard curve per plate so
# the replicates all have value 1.
standard_curve_df = data.frame(Sample = paste0("P", 1:8), Dilution = c(1/100,
    1/200, 1/400, 1/800, 1/1600, 1/3200, 1/6400, 1/12800), Location = c("65(1,A9)",
    "66(1,B9)", "67(1,C9)", "68(1,D9)", "69(1,E9)", "70(1,F9)",
    "71(1,G9)", "72(1,H9)"), Replicate = rep(1, 8))
```

Now, we can use the `read_and_tidy()` function from Lab 2b to read in our 4 example plates in the `Data` folder.

```
plate_1_tidy <- read_and_tidy(file_name = paste(my_path, "Data/Raw Plate 1 dataset.csv",
    sep = "/"), plate_number = 1, num_wells = 96, antigen_names = agx_names,
    control_samples = ctrls, background_samples = bg_samples,
    standard_curve_values = standard_curve_df, bead_threshold = 30)

plate_2_tidy <- read_and_tidy(file_name = paste(my_path, "Data/Raw Plate 2 dataset.csv",
    sep = "/"), plate_number = 2, num_wells = 96, antigen_names = agx_names,
    control_samples = ctrls, background_samples = bg_samples,
    standard_curve_values = standard_curve_df, bead_threshold = 30)

plate_3_tidy <- read_and_tidy(file_name = paste(my_path, "Data/Raw Plate 3 dataset.csv",
    sep = "/"), plate_number = 3, num_wells = 96, antigen_names = agx_names,
    control_samples = ctrls, background_samples = bg_samples,
    standard_curve_values = standard_curve_df, bead_threshold = 30)

plate_4_tidy <- read_and_tidy(file_name = paste(my_path, "Data/Raw Plate 4 dataset.csv",
    sep = "/"), plate_number = 4, num_wells = 96, antigen_names = agx_names,
    control_samples = ctrls, background_samples = bg_samples,
    standard_curve_values = standard_curve_df, bead_threshold = 30)
```

Now we can perform some basic checks on each plate using the `head()` functions and some summaries.

```
# head function to ensure the structure is correct
head(plate_1_tidy)
```

```
##   Location   Sample Antigen  MFI BeadCount Plate Sample_Type Low_Beads
## 1  1(1,A1) Unknown1    SNAP   80       173     1  TestSample         0
## 2  2(1,B1) Unknown2    SNAP   74       122     1  TestSample         0
## 3  3(1,C1) Unknown3    SNAP  214       161     1  TestSample         0
## 4  4(1,D1) Unknown4    SNAP 1495       119     1  TestSample         0
## 5  5(1,E1) Unknown5    SNAP  226       132     1  TestSample         0
## 6  6(1,F1) Unknown6    SNAP  189       120     1  TestSample         0
```

```
head(plate_2_tidy)
```

```
##   Location   Sample Antigen  MFI BeadCount Plate Sample_Type Low_Beads
```

```
## 1  1(1,A1) Unknown1    SNAP  320       171     2  TestSample         0
## 2  2(1,B1) Unknown2    SNAP  258       124     2  TestSample         0
## 3  3(1,C1) Unknown3    SNAP  253       157     2  TestSample         0
## 4  4(1,D1) Unknown4    SNAP  116       116     2  TestSample         0
## 5  5(1,E1) Unknown5    SNAP 7285       130     2  TestSample         0
## 6  6(1,F1) Unknown6    SNAP  245       116     2  TestSample         0
```

```
head(plate_3_tidy)
```

```
##   Location   Sample Antigen  MFI BeadCount Plate Sample_Type Low_Beads
## 1  1(1,A1) Unknown1    SNAP 5115       177     3  TestSample         0
## 2  2(1,B1) Unknown2    SNAP 3933       120     3  TestSample         0
## 3  3(1,C1) Unknown3    SNAP  451       164     3  TestSample         0
## 4  4(1,D1) Unknown4    SNAP  240       121     3  TestSample         0
## 5  5(1,E1) Unknown5    SNAP 1395       136     3  TestSample         0
## 6  6(1,F1) Unknown6    SNAP  167       116     3  TestSample         0
```

```
head(plate_4_tidy)
```

```
##   Location   Sample Antigen   MFI BeadCount Plate Sample_Type Low_Beads
## 1  1(1,A1) Unknown1    SNAP  2975       175     4  TestSample         0
## 2  2(1,B1) Unknown2    SNAP   258       124     4  TestSample         0
## 3  3(1,C1) Unknown3    SNAP  1935       158     4  TestSample         0
## 4  4(1,D1) Unknown4    SNAP   295       123     4  TestSample         0
## 5  5(1,E1) Unknown5    SNAP 17565       135     4  TestSample         0
## 6  6(1,F1) Unknown6    SNAP   441       115     4  TestSample         0
```

```
# take note of the number of columns, the column names and
# the information that is contained in each column.

# now we can see how many antigens and samples are
# represented on each plate
plate_1_agxSummary <- summary(as.factor(plate_1_tidy$Antigen))
plate_1_agxSummary
```

```
##    CHIKV      CSP     DENV  GLURPR2      JE3   PfAMA1 PfMSP119     SNAP
##       96       96       96       96       96       96       96       96
##     WMEV      WNV     WRUV       YF     ZIKA
##       96       96       96       96       96
```

```
# try repeating this on your own for each plate, how many
# antigens are there in this assay? Does each sample have
# an output for each antigen?

# now we can get a summary of different sample types
plate_1_sampleSummary <- summary(as.factor(plate_1_tidy$Sample_Type))
plate_1_sampleSummary
```

```
##         BG       Ctrl   StdCurve TestSample
##         26         52        104       1066
```

```
# what do you notice? Do all sample types sum to 96? Why
# not? (hint think how many measurements are available for
# each sample, is it just 1 or more than 1)? What happens
# if you divide the number within each sample category by
# the number of different antigens?  repeat this for all 4
# plates
```

Exercise 1:

Use the code above to accomplish the same task with your data. First ensure that your data (`.csv` files) are in the appropriate folder (we suggest saving them to the `Data` folder in the `seroanalytics_workshop` directory). Be sure to also input the appropriate values into the `read_and_tidy()` function, e.g., you likely will not have the same antigen names or names for the blank, control, and standard curve samples as the example data. Be sure to use all values that are specific to your data set.

Filtering

Now that we have read in our raw data, we can filter out data that falls below our minimum bead count threshold using the `filter_low_beads()` function as follows:

```
# note we are assigning the filtered output to a new object
# with a new name, we want to be able to see each stage of
# our pipeline separately, so we want to avoid overwriting
# objects as we go.

plate_1_filt <- filter_low_beads(plate_1_tidy)
plate_2_filt <- filter_low_beads(plate_2_tidy)
plate_3_filt <- filter_low_beads(plate_3_tidy)
plate_4_filt <- filter_low_beads(plate_4_tidy)


# now, compare the filtered output to the tidy output, do
# you see any differences?
dim(plate_1_tidy)
```

```
## [1] 1248    8
```

```
dim(plate_1_filt)
```

```
## [1] 1248    8
```

```
# it appears that for plate 1 there are no differences, in
# other words there were no samples that fell below the
# minimum bead count threshold (this may not always be the
# case).  we can confirm that by checking the bead count
# variable in our original 'tidy' data as follows:
summary(as.factor(plate_1_tidy$Low_Beads))
```

```
##    0
## 1248
```

```
# now, repeat this with the remaining 3 example plates
# (plates 2-4)
```

Exercise 2:

Use the code above to accomplish the same task with your data. Make sure to perform adequate checks, as shown above, on all your plates to be sure the `filter_low_beads()` function removed only what you expected.

Background Removal

Now that we have a filtered data set, we can remove background using the blank wells on each plate. Note, there are different ways one can consider removing background, today we will focus only on removing background using subtraction. In other words, we will subtract our estimate of background signal from each MFI value as follows:

```
# note we signal that we want to use the subtraction method
# using the 'method' argument of the function
plate_1_bg <- rm_background(plate_1_filt, method = "subtraction")
plate_2_bg <- rm_background(plate_2_filt, method = "subtraction")
plate_3_bg <- rm_background(plate_3_filt, method = "subtraction")
plate_4_bg <- rm_background(plate_4_filt, method = "subtraction")


# now we will check what our new dataset looks like take
# note of any new columns that are present that were not
# present in the filtered data
head(plate_1_bg)
```

```
##   Location   Sample Antigen  MFI BeadCount Plate Sample_Type Low_Beads
## 1  1(1,A1) Unknown1    SNAP   80       173     1  TestSample         0
## 2  2(1,B1) Unknown2    SNAP   74       122     1  TestSample         0
## 3  3(1,C1) Unknown3    SNAP  214       161     1  TestSample         0
## 4  4(1,D1) Unknown4    SNAP 1495       119     1  TestSample         0
## 5  5(1,E1) Unknown5    SNAP  226       132     1  TestSample         0
## 6  6(1,F1) Unknown6    SNAP  189       120     1  TestSample         0
##   Median_BG MFI_BG
## 1     107.5  -27.5
## 2     107.5  -33.5
## 3     107.5  106.5
## 4     107.5 1387.5
## 5     107.5  118.5
## 6     107.5   81.5
```

```
# what do you notice in the first few rows of the new
# column called 'MFI_BG' which is the background subtracted
# MFI? Hint are all values >0? If not why not?


# perform the same task on the other example plates (2-4).
```

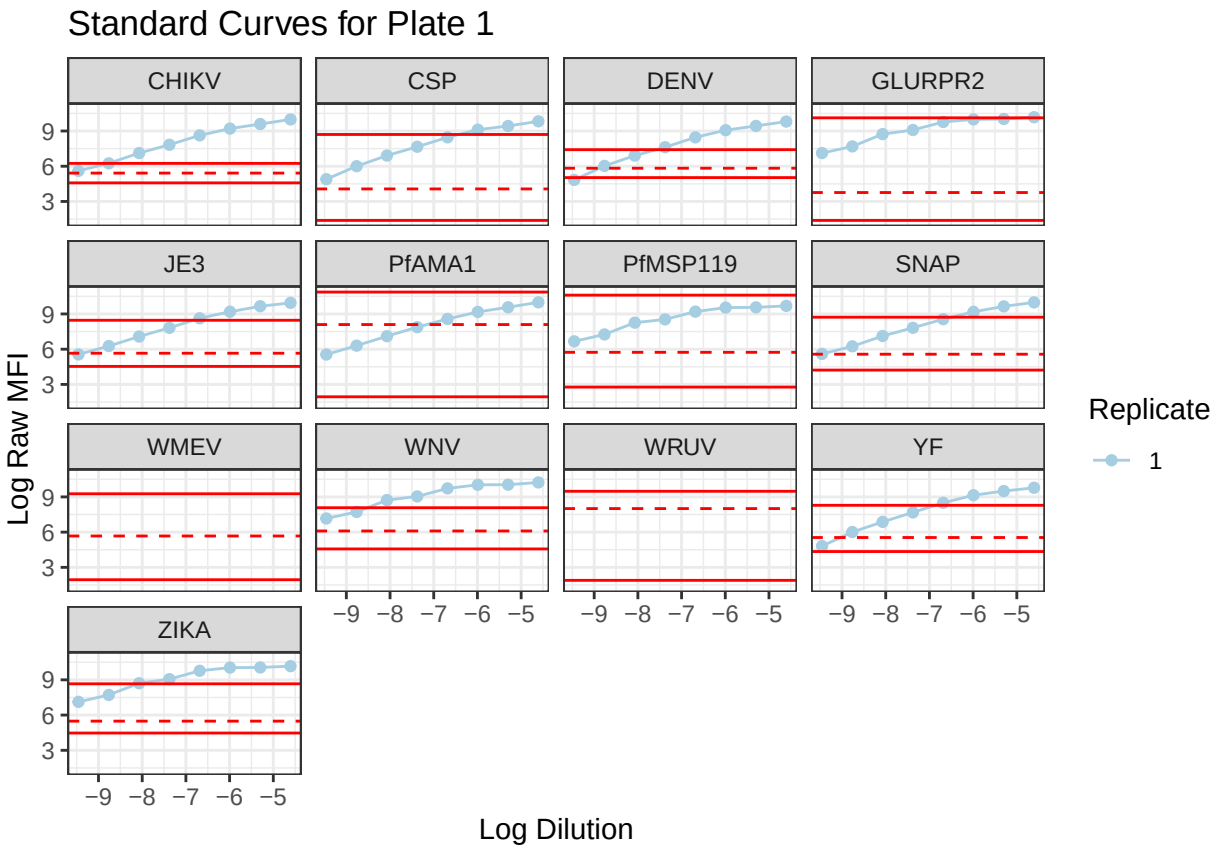
Exercise 3:

Use the code above to accomplish the same task with your data. Make sure to perform adequate checks, as shown above, on all your plates to be sure the `rm_background()` function produced the desired results.

Standard Curves (Standardization)

First, we will visualize the standard curves in our example data set with raw MFI values as inputs.

```
# this function makes use of the standard curve information
# that first appeared in the plateSetup section above.
plate_1_stdCurve <- plot_std_curves(plate_norm_df = plate_1_bg,
    std_curve_values = standard_curve_df, input = "MFI")
# note this function outputs a list of different objects we
# are only interested in the first object which is why we
# use the '[[1]]' notation, this signals that from the
# object 'plate_1_stdCurve' we only want to see the first
# object, which is the plot.
plate_1_stdCurve[[1]]
```




```
# note also that 2 antigens (WMEV, measles, and WRUV
# rubella) do not have standard curve information shown
# here, this is expected.  Take note of the scale on the x,
# y axes, what do you notice?  The red dashed line is the
```

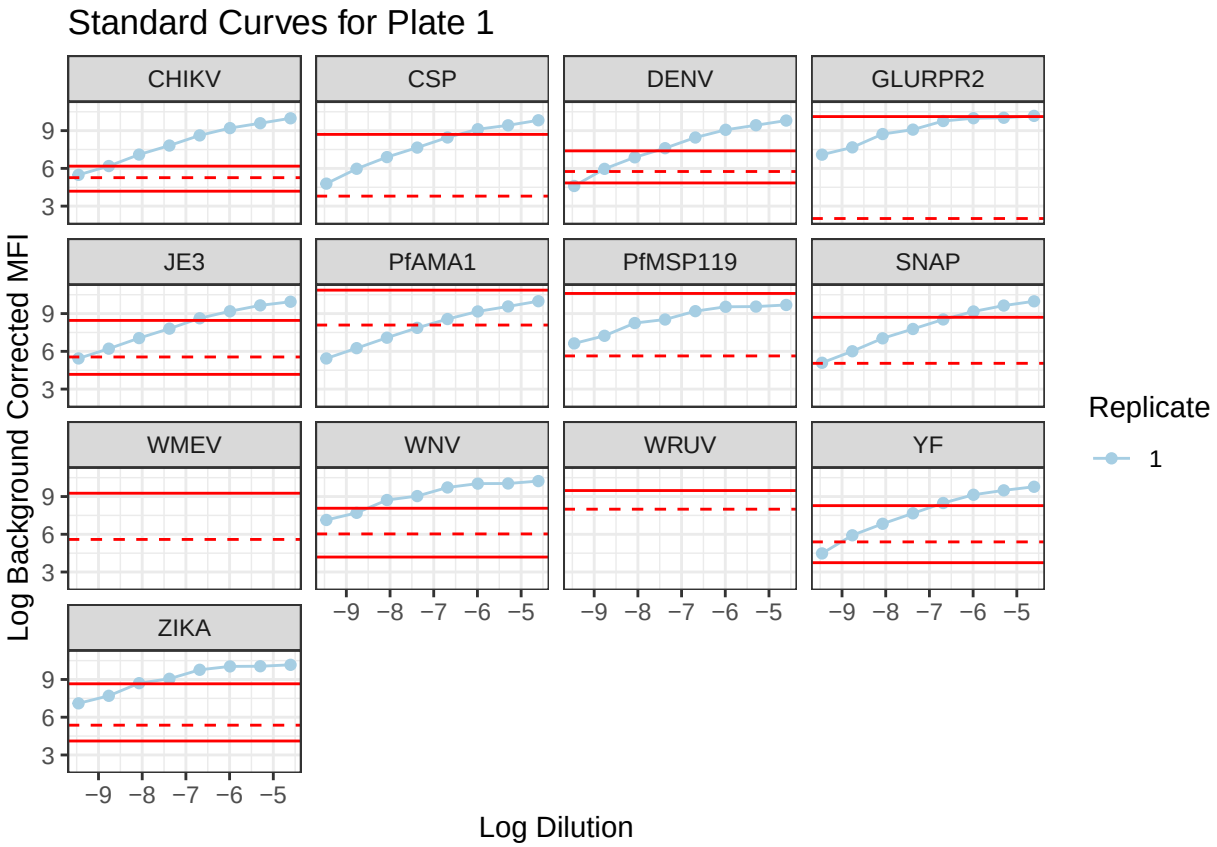
```
# median MFI of samples on that plate, the solid red lines
# represent where 95% of the samples lie on the log MFI
# scale.  what do you notice about the location of the
# samples relative to the standard curve?

# repeat this for all example plates (2-4)
```

Now, we will visualize the standard curves in our example data set with background subtracted MFI values as inputs.

```
plate_1_stdCurve_BG <- plot_std_curves(plate_1_bg, std_curve_values = standard_curve_df,
    input = "bgMFI")

plate_1_stdCurve_BG[[1]]
```




```
# Some lower 95% bounds are missing now, why might this be
# (hint recall what happens when you try to take the
# natural log of values less than or equal to 0).  do you
# notice any differences in the standard curves? Sample
# values relative to the standard curves, do they overlap
# in the same way?

# repeat this for all example plates (2-4)
```

Exercise 4:

Use the code above to accomplish the same task with your data. Make sure to use the correct inputs for standard curves relative to what is on your plates.

Now, we will choose some antigens to fit our standard curves to each antigen (all but measles and rubella). By fitting our standard curves, we use a model to describe the relationship between log dilution and MFI (for now, we will only consider MFI on the raw scale, not the background subtracted scale). Using this relationship, we will estimate the dilution (or concentration) of samples in our data using the MFIs of each sample.

```
# first remove measles and rubella samples
plate_1_bg_noMEVnoRUV <- plate_1_bg[-which(plate_1_bg$Antigen %in%
    c("WMEV", "WRUV")), ]

plate_2_bg_noMEVnoRUV <- plate_2_bg[-which(plate_2_bg$Antigen %in%
    c("WMEV", "WRUV")), ]

plate_3_bg_noMEVnoRUV <- plate_3_bg[-which(plate_3_bg$Antigen %in%
    c("WMEV", "WRUV")), ]

plate_4_bg_noMEVnoRUV <- plate_4_bg[-which(plate_4_bg$Antigen %in%
    c("WMEV", "WRUV")), ]


# use this function to both estimate the standard curve
# function and back out the estimated sample concentrations
# on the log scale
plate_1_standard <- get_concentration_FlexFit(plate_1_bg_noMEVnoRUV,
    std_curve_values = standard_curve_df, input = "MFI")

plate_2_standard <- get_concentration_FlexFit(plate_2_bg_noMEVnoRUV,
    std_curve_values = standard_curve_df, input = "MFI")


plate_3_standard <- get_concentration_FlexFit(plate_3_bg_noMEVnoRUV,
    std_curve_values = standard_curve_df, input = "MFI")


plate_4_standard <- get_concentration_FlexFit(plate_4_bg_noMEVnoRUV,
    std_curve_values = standard_curve_df, input = "MFI")


# now check what plate_1_standard looks like first use the
# head() function:
head(plate_1_standard)
```

```
##   Location   Sample Antigen  MFI BeadCount Plate Sample_Type Low_Beads
## 1  1(1,A1) Unknown1    SNAP   80       173     1  TestSample         0
## 2  2(1,B1) Unknown2    SNAP   74       122     1  TestSample         0
## 3  3(1,C1) Unknown3    SNAP  214       161     1  TestSample         0
## 4  4(1,D1) Unknown4    SNAP 1495       119     1  TestSample         0
## 5  5(1,E1) Unknown5    SNAP  226       132     1  TestSample         0
## 6  6(1,F1) Unknown6    SNAP  189       120     1  TestSample         0
##   Median_BG MFI_BG  log_MFI Log_Conc_bg
## 1     107.5  -27.5 4.382027  -11.245271
```

```
## 2     107.5  -33.5 4.304065  -11.434799
## 3     107.5  106.5 5.365976   -9.703695
## 4     107.5 1387.5 7.309881   -7.856442
## 5     107.5  118.5 5.420535   -9.640806
## 6     107.5   81.5 5.241747   -9.852062
```

```
# which new columns have been added?

# look for NA values in estimated concentration, these can
# occur when the samples are beyond the upper or lower
# limits of the standard curve which we visualized using
# the plots above.
summary(plate_1_standard$Log_Conc_bg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -13.696  -9.773  -9.084  -8.695  -8.190   5.085     413
```

```
# in particular, let's check if certain types of samples
# (e.g., controls) have NA output from this step.
summary(plate_1_standard$Log_Conc_bg[which(plate_1_standard$Sample_Type ==
    "Ctrl")])
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
## -12.20786  -9.73827  -9.14359  -8.42746  -7.88417   0.02499        20
```

```
# Now repeat this for all example plates (2-4)
```

Exercise 5:

Use the code above to accomplish the same task with your data. Make sure to use the correct inputs for standard curves relative to what is on your plates, and to filter out certain antigens if there is no standard curve for those antigens in your data.

Normalization (Plate Effects)

Now, we will perform normalization by estimating batch effects between plates and then subtracting those batch effects from all MFIs.

```
# first bind all 4 standardized plates together into 1 data
# frame

complete_std_plates_df <- rbind(plate_1_standard, plate_2_standard,
    plate_3_standard, plate_4_standard)

# now using this complete data frame we will run our
# normalization function to estimate and remove batch
# effects between plates
normalized_df <- get_norm_df(complete_std_plates_df, method = "MFI")

# note there are new columns including: 1. 'norm method',
# this tells you what normalization method was employed, in
```

```r
# our example this is either no normalization which is
# represented as 'Log_Conc_bg' or LM norm MFI which is the
# method that estimates batch effects and subtracts them
# from each plate.  2. 'Norm_MFI' this is the actual
# normalized value 3. Input value is Log conc bg which is
# how we represent that the standardized values from above
# were used as inputs to this function.
head(data.frame(normalized_df))
```

```
##   Plate   Sample Location Sample_Type Antigen Norm_Method Norm_MFI Input_Value
## 1     1 Unknown1  1(1,A1)  TestSample    SNAP     log_MFI 4.382027     log_MFI
## 2     1 Unknown1  1(1,A1)  TestSample    SNAP LM_norm_MFI 4.382027     log_MFI
## 3     1 Unknown2  2(1,B1)  TestSample    SNAP     log_MFI 4.304065     log_MFI
## 4     1 Unknown2  2(1,B1)  TestSample    SNAP LM_norm_MFI 4.304065     log_MFI
## 5     1 Unknown3  3(1,C1)  TestSample    SNAP     log_MFI 5.365976     log_MFI
## 6     1 Unknown3  3(1,C1)  TestSample    SNAP LM_norm_MFI 5.365976     log_MFI
```

```r
# consider whether NAs or NaNs were introduced in this
# procedure:
summary(normalized_df$Norm_MFI)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## -31.686  -8.787   2.909  -0.671   5.976  11.449    4456
```

Exercise 6:

Use the code above to accomplish the same task with your data. Make sure to bind all plates together into one data frame.