Welcome to **Part Two** of the proteomics data analysis. The ultimate goal of this exercise is to identify proteins whose abundance is different between the drug-resistant cells and the control. In other words, we are looking for a list of differentially regulated proteins that may shed light on how cells escape the cancer-killing action of a drug.

Below, I have demonstrated the steps to acquiring a proteomics data set and performing data pre-processing. We will pick up from the cleaned data set and confront the missing value problem in proteomics.

The analysis of DNA and RNA, the blueprint of life and its carbon copy, has become a staple in the burgeoning field of molecular biology. An emerging and exciting area of study that adds another dimension to our understanding of cellular biology is that of proteomics, or the study of proteins inside the cell. The use of mass spectrometry has enabled the identification and quantification of thousands of proteins in a single experiment.

In this tutorial series, I will break down the steps to process a high-throughput proteomics data set derived from mass spectrometry analysis as follows:

* **Data acquisition and cleaning**
* Data filtering and missing value imputation
* Statistical testing and data interpretation

**Source of Proteomics Data**

To obtain a sample data set, I combed through a proteomics data repository called [PRIDE](https://www.ebi.ac.uk/pride/archive/simpleSearch) and found an interesting study on drug resistance in breast cancer cell lines. I downloaded the [raw files](https://www.ebi.ac.uk/pride/archive/projects/PXD002057/files), which are the output of mass spectrometry analysis, and processed them using a software called [MaxQuant](http://www.biochem.mpg.de/5111795/maxquant) to map the spectral data to protein sequences. A total of six raw files, corresponding to two conditions (one resistant line and one control) with three replicates each, were used. There are numerous other tools for processing mass spectrometry data (e.g. Mascot, SEQUEST, ProteinProspector), and the final data table of protein abundance measurements will vary base on the approach. The starting point for this tutorial is the MaxQuant *ProteinGroups* output file, which can be downloaded [here](https://ucsf.box.com/s/1wt4hjdt3yq9g8fcs8oam1btg66dzt54).

**Data Acquisition**

The first step is to read the tab-separated data file into R.

# Read raw file

raw = read.delim("proteinGroups.txt", stringsAsFactors = FALSE, colClasses = "character")

Our raw data is an enormous 1787-by-79 data frame. Proteins are arranged in rows and the descriptors in columns. The primary columns of interest are those containing intensity measurements, which reflect protein abundances.

# Extract names of intensity columns

grep("^LFQ.intensity", names(raw), value = TRUE)

## [1] "LFQ.intensity.Parental\_bR1" "LFQ.intensity.Parental\_bR2"

## [3] "LFQ.intensity.Parental\_bR3" "LFQ.intensity.Resistant\_bR1"

## [5] "LFQ.intensity.Resistant\_bR2" "LFQ.intensity.Resistant\_bR3"

Again, we have a total of six samples. The Parental represents intensity data from the breast cancer cell line SKBR3 while the Resistant is an drug-resistant cell line derived from culturing the parentals in the presence of an inhibitor. This small molecule targets epidermal growth factor receptor (EGFR), a cell-surface protein that is frequently over-expressed in breast tumors leading to increased cell proliferation. For more information regarding the study, please see the original [publication](https://europepmc.org/abstract/MED/26883193).

**Data Cleaning**

**Remove False Hits**

The next step after data acquisition is to clean and organize our data. The first order of business is to remove false hits, including contaminants, reverse proteins, and proteins identified by site. These are annotated with a “+” under the columns Potential.contaminant, Reverse, and Only.identified.by.site. We filter the data frame by keeping rows without a “+” annotation in any of the three columns.

library(dplyr) # for data manipulation

# Filter false hits

df = raw %>%

filter(Potential.contaminant != "+") %>%

filter(Reverse != "+") %>%

filter(Only.identified.by.site != "+")

Often there is a column that indicates the confidence in protein identification. In our case, Q.value represents the probability that the protein is a false hit. A typical cutoff is set at 0.01. Fortunately, MaxQuant takes care of this operation and ensures that all Q values are below the threshold.

# Summary of Q values

summary(as.numeric(df$Q.value))

## Min. 1st Qu. Median Mean 3rd Qu. Max.

## 0.0000000 0.0000000 0.0000000 0.0007193 0.0012759 0.0091429

**Extract Protein and Gene IDs**

A quick look at Protein.IDs and Fasta.headers columns tells us that the protein IDs, protein names, and gene IDs are all lumped together.

# View first 6 entries in Protein.IDs

head(df$Protein.IDs)

## [1] "sp|A0AV96|RBM47\_HUMAN" "sp|A0FGR8|ESYT2\_HUMAN" "sp|A1L0T0|ILVBL\_HUMAN"

## [4] "sp|A4D1S0|KLRG2\_HUMAN" "sp|A5YKK6|CNOT1\_HUMAN" "sp|A6NDG6|PGP\_HUMAN"

# View first 6 entries in Fasta.headers

head(df$Fasta.headers)

## [1] ">sp|A0AV96|RBM47\_HUMAN RNA-binding protein 47 OS=Homo sapiens GN=RBM47 PE=1 SV=2"

## [2] ">sp|A0FGR8|ESYT2\_HUMAN Extended synaptotagmin-2 OS=Homo sapiens GN=ESYT2 PE=1 SV=1"

## [3] ">sp|A1L0T0|ILVBL\_HUMAN Acetolactate synthase-like protein OS=Homo sapiens GN=ILVBL PE=1 SV=2"

## [4] ">sp|A4D1S0|KLRG2\_HUMAN Killer cell lectin-like receptor subfamily G member 2 OS=Homo sapiens GN=KLRG2 PE=1 SV=3"

## [5] ">sp|A5YKK6|CNOT1\_HUMAN CCR4-NOT transcription complex subunit 1 OS=Homo sapiens GN=CNOT1 PE=1 SV=2"

## [6] ">sp|A6NDG6|PGP\_HUMAN Glycerol-3-phosphate phosphatase OS=Homo sapiens GN=PGP PE=1 SV=1"

We will use regular expressions to extract the protein names into a column named Protein.name, the UniProt protein IDs into Protein, and the gene IDs into Gene. Note that some rows are associated with multiple identifiers separated by semicolons. In those instances, we will isolate the first entry.

# Isolate the first entry

df$Protein.IDs = sub(";.\*", "", df$Protein.IDs)

df$Fasta.headers = sub(";.\*", "", df$Fasta.headers)

# Extract Protein name

regex = regexpr("(?<=\_HUMAN.).\*(?=.OS)", df$Fasta.headers, perl = TRUE)

df$Protein.name = regmatches(df$Fasta.headers, regex)

# Extract UniProtID

regex = regexpr("(?<=\\|).\*(?=\\|)", df$Protein.IDs, perl = TRUE)

df$Protein = regmatches(df$Protein.IDs, regex)

# Extract Gene ID

regex = regexpr("((?<=\\|[[:alnum:]]{6}\\|).\*(?=\_HUMAN)|(?<=\\|[[:alnum:]]{10}\\|).\*(?=\_HUMAN))",

df$Protein.IDs, perl = TRUE)

df$Gene = regmatches(df$Protein.IDs, regex)

**Transform Intensity Columns**

Due to our function call for reading the data table, all columns are cast as the character data type. We will convert the intensity columns to the numeric data type for downstream analysis.

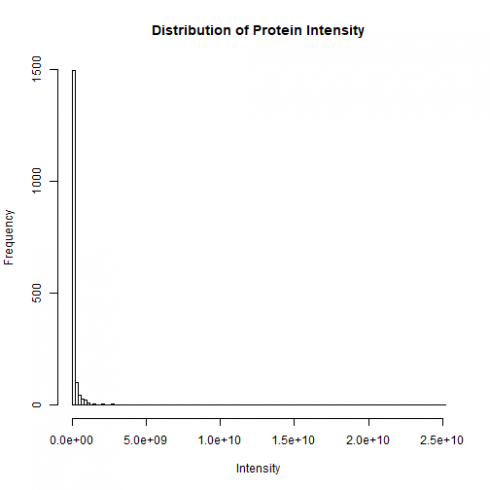
# Extract names of intensity columns

intensity.names = grep("^LFQ.intensity", names(df), value = TRUE)

# Cast as numeric

df[intensity.names] = sapply(df[intensity.names], as.numeric)

Now let’s examine the distribution of protein intensities in a sample. Below is a histogram of the protein intensities in the Parental\_bR1 sample.

[](https://datascienceplus.com/wp-content/uploads/2018/08/unnamed-chunk-9-1.png)

The distribution is clearly skewed to the right with a few highly abundant proteins. To normalize the distribution, it is common practice to log2-transform the intensity data.

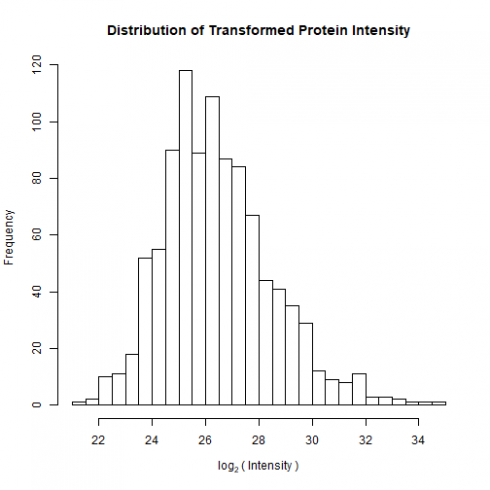
# Assign column names for log2-transformed data

LOG.names = sub("^LFQ.intensity", "LOG2", intensity.names) # rename intensity columns

# Transform data

df[LOG.names] = log2(df[intensity.names])

Here’s the transformed distribution on Parental\_bR1 (much better!):

[](https://datascienceplus.com/wp-content/uploads/2018/08/unnamed-chunk-11-1.png)

Again, the outline for this tutorial series is as follows:

* Data acquisition and cleaning
* **Data filtering and missing value imputation**
* Statistical testing and data interpretation

**Missing Value Problem**

Although mass spectrometry-based proteomics has the advantage of detecting thousands of proteins from a single experiment, it faces certain challenges. One problem is the presence of missing values in proteomics data. To illustrate this, let's examine the first few rows of the log~2~-transformed and raw protein abundance values.

library(dplyr)

head(select(df, Gene, starts\_with("LOG2")))

*## Gene LOG2.Parental\_bR1 LOG2.Parental\_bR2 LOG2.Parental\_bR3*

*## 1 RBM47 -Inf -Inf 21.87748*

*## 2 ESYT2 25.6019 25.56180 25.68763*

*## 3 ILVBL -Inf 20.76474 -Inf*

*## 4 KLRG2 -Inf -Inf 22.31786*

*## 5 CNOT1 -Inf -Inf -Inf*

*## 6 PGP -Inf -Inf -Inf*

*## LOG2.Resistant\_bR1 LOG2.Resistant\_bR2 LOG2.Resistant\_bR3*

*## 1 -Inf -Inf -Inf*

*## 2 -Inf -Inf -Inf*

*## 3 -Inf -Inf -Inf*

*## 4 -Inf -Inf -Inf*

*## 5 -Inf -Inf 29.14207*

*## 6 -Inf -Inf 22.46269*

head(select(df, Gene, starts\_with("LFQ")))

*## Gene LFQ.intensity.Parental\_bR1 LFQ.intensity.Parental\_bR2*

*## 1 RBM47 0 0*

*## 2 ESYT2 50926000 49530000*

*## 3 ILVBL 0 1781600*

*## 4 KLRG2 0 0*

*## 5 CNOT1 0 0*

*## 6 PGP 0 0*

*## LFQ.intensity.Parental\_bR3 LFQ.intensity.Resistant\_bR1*

*## 1 3852800 0*

*## 2 54044000 0*

*## 3 0 0*

*## 4 5228100 0*

*## 5 0 0*

*## 6 0 0*

*## LFQ.intensity.Resistant\_bR2 LFQ.intensity.Resistant\_bR3*

*## 1 0 0*

*## 2 0 0*

*## 3 0 0*

*## 4 0 0*

*## 5 0 592430000*

*## 6 0 5780200*

It is hard to miss the -Inf, which represent protein intensity measurements of 0 in the raw data set. We consider these data points as missing values, or a lack of quantification in the indicated samples. This is a common issue in proteomics experiments, and it arises due to sample complexity and randomness (or stochasticity) in sampling.

For example, imagine pouring out a bowl of Lucky Charms cereal containing a thousand different marshmallows. Let's say there is only one coveted rainbow marshmallow for every one thousand pieces. The likelihood of your bowl containing the rare shape is disappointingly low. In our situation, there are approximately 20,000 proteins expressed in a given cell, and many in low quantities. Hence, the probability of consistently capturing proteins with low expression across all experiments is small.

**Data Filtering**

To overcome the missing value problem, we need to remove proteins that are sparsely quantified. The hypothesis is that a protein quantified in only one out of six samples offers insufficient grounds for comparison. In addition, the protein could have been mis-assigned.

One of many filtering schemes is to keep proteins that are quantified in at least two out of three replicates in one condition. To jog your memory, we have two conditions, one drug-resistant cell line and a control, and three replicates each. The significance of replicates will be discussed in **Part 3** of the tutorial. For now, we will briefly clean the data frame and apply filtering.

## Data cleaning: Extract columns of interest

df = select(df, Protein, Gene, Protein.name, starts\_with("LFQ"), starts\_with("LOG2"))

glimpse(df)

*## Observations: 1,747*

*## Variables: 15*

*## $ Protein "A0AV96", "A0FGR8", "A1L0T0", "A4D...*

*## $ Gene "RBM47", "ESYT2", "ILVBL", "KLRG2"...*

*## $ Protein.name "RNA-binding protein 47", "Extende...*

*## $ LFQ.intensity.Parental\_bR1 0, 50926000, 0, 0, 0, 0, 0, 0, 0, ...*

*## $ LFQ.intensity.Parental\_bR2 0, 49530000, 1781600, 0, 0, 0, 0, ...*

*## $ LFQ.intensity.Parental\_bR3 3852800, 54044000, 0, 5228100, 0, ...*

*## $ LFQ.intensity.Resistant\_bR1 0, 0, 0, 0, 0, 0, 0, 0, 0, 8213400...*

*## $ LFQ.intensity.Resistant\_bR2 0, 0, 0, 0, 0, 0, 0, 0, 0, 6903700...*

*## $ LFQ.intensity.Resistant\_bR3 0, 0, 0, 0, 592430000, 5780200, 0,...*

*## $ LOG2.Parental\_bR1 -Inf, 25.60190, -Inf, -Inf, -Inf, ...*

*## $ LOG2.Parental\_bR2 -Inf, 25.56180, 20.76474, -Inf, -I...*

*## $ LOG2.Parental\_bR3 21.87748, 25.68763, -Inf, 22.31786...*

*## $ LOG2.Resistant\_bR1 -Inf, -Inf, -Inf, -Inf, -Inf, -Inf...*

*## $ LOG2.Resistant\_bR2 -Inf, -Inf, -Inf, -Inf, -Inf, -Inf...*

*## $ LOG2.Resistant\_bR3 -Inf, -Inf, -Inf, -Inf, 29.14207, ...*

## Data filtering function

filter\_valids = function(df, conditions, min\_count, at\_least\_one = FALSE) {

# df = data frame containing LOG2 data for filtering and organized by data type

# conditions = a character vector dictating the grouping

# min\_count = a numeric vector of the same length as "conditions" indicating the minimum

# number of valid values for each condition for retention

# at\_least\_one = TRUE means to keep the row if min\_count is met for at least one condition

# FALSE means min\_count must be met across all conditions for retention

log2.names = grep("^LOG2", names(df), value = TRUE) # Extract LOG2 column names

cond.names = lapply(conditions, # Group column names by conditions

function(x) grep(x, log2.names, value = TRUE, perl = TRUE))

cond.filter = sapply(1:length(cond.names), function(i) {

df2 = df[cond.names[[i]]] # Extract columns of interest

df2 = as.matrix(df2) # Cast as matrix for the following command

sums = rowSums(is.finite(df2)) # count the number of valid values for each condition

sums >= min\_count[i] # Calculates whether min\_count requirement is met

})

if (at\_least\_one) {

df$KEEP = apply(cond.filter, 1, any)

} else {

df$KEEP = apply(cond.filter, 1, all)

}

return(df) # No rows are omitted, filter rules are listed in the KEEP column

}

## Apply filtering

df.F = filter\_valids(df,

conditions = c("Parental", "Resistant"),

min\_count = c(2, 2),

at\_least\_one = TRUE)

The output data frame df.F is a copy of df with an additional *KEEP* column indicating the rows to retain. We will complete the filtering using the following operation and then check out the first couple of rows.

df.F = filter(df.F, KEEP)

head(select(df.F, Gene, starts\_with("LOG2")))

*## Gene LOG2.Parental\_bR1 LOG2.Parental\_bR2 LOG2.Parental\_bR3*

*## 1 ESYT2 25.60190 25.56180 25.68763*

*## 2 EIF3C 26.93022 27.11644 26.83231*

*## 3 NACAM 27.71299 27.66756 27.53527*

*## 4 DX39A 25.90933 25.69806 25.93283*

*## 5 BACH 25.07153 25.39110 25.06027*

*## 6 MYO1C 27.16471 27.48416 27.43841*

*## LOG2.Resistant\_bR1 LOG2.Resistant\_bR2 LOG2.Resistant\_bR3*

*## 1 -Inf -Inf -Inf*

*## 2 26.29148 26.04087 26.46083*

*## 3 29.03570 28.68295 28.89753*

*## 4 26.39331 26.54022 -Inf*

*## 5 -Inf -Inf -Inf*

*## 6 27.61400 27.02263 27.44530*

Notice that the protein in the first row is quantified in the *Parental* line but not the *Resistant* one. Proteins like this are of great interest to us as they are likely implicated in the mechanism of drug resistance. In addition, note that the final number of proteins after filtering (1031) is roughly 60% the original number (1747). Filtering reduces our list of proteins to ones quantified in a reasonably consistent manner.

**Data Normalization**

Before we proceed to imputation, we need to account for technical variability in the amount of sample analyzed by the mass spectrometer from one run to another. This is an issue parallel to the variation in sequencing depth in RNAseq experiments. To normalize out these technical differences, we performed a global median normalization. For each sample, the median of the log~2~-transformed distribution is subtracted from all the values.

## Data normalization function

median\_centering = function(df) {

# df = data frame containing LOG2 columns for normalization

LOG2.names = grep("^LOG2", names(df), value = TRUE)

df[, LOG2.names] = lapply(LOG2.names,

function(x) {

LOG2 = df[[x]]

LOG2[!is.finite(LOG2)] = NA # Exclude missing values from median calculation

gMedian = median(LOG2, na.rm = TRUE)

LOG2 - gMedian

}

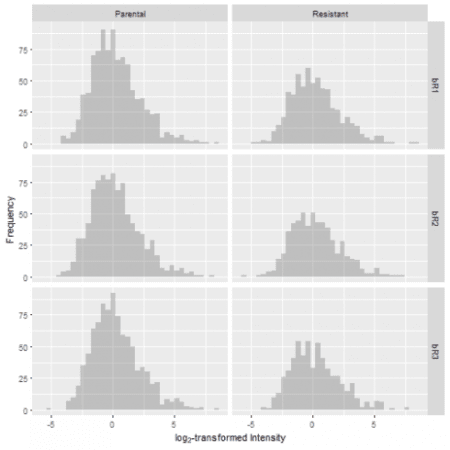
)

return(df)

}

## Normalize data

df.FN = median\_centering(df.F)

The result is that each sample is centered at a log~2~(intensity) of 0.  
[](https://i0.wp.com/datascienceplus.com/wp-content/uploads/2018/08/unnamed-chunk-50-1.png?ssl=1)

**Data Imputation**

After filtering and normalization, some missing values remain. How do we deal with them from here? The statistical approach designed to answer such a question is called imputation. For a thorough discussion of imputation on proteomics data sets.

Since missing values are associated with proteins with low levels of expression, we can substitute the missing values with numbers that are considered “small” in each sample. We can define this statistically by drawing from a normal distribution with a mean that is down-shifted from the sample mean and a standard deviation that is a fraction of the standard deviation of the sample distribution. Here's a function that implements this approach:

## Data imputation function

impute\_data = function(df, width = 0.3, downshift = 1.8) {

# df = data frame containing filtered

# Assumes missing data (in df) follows a narrowed and downshifted normal distribution

# use\_keep = filter rows using KEEP column prior to imputation

LOG2.names = grep("^LOG2", names(df), value = TRUE)

impute.names = sub("^LOG2", "impute", LOG2.names)

# Create new columns indicating whether the values are imputed

df[impute.names] = lapply(LOG2.names, function(x) !is.finite(df[, x]))

# Imputation

set.seed(1)

df[LOG2.names] = lapply(LOG2.names,

function(x) {

temp = df[[x]]

temp[!is.finite(temp)] = NA

temp.sd = width \* sd(temp[df$KEEP], na.rm = TRUE) # shrink sd width

temp.mean = mean(temp[df$KEEP], na.rm = TRUE) -

downshift \* sd(temp[df$KEEP], na.rm = TRUE) # shift mean of imputed values

n.missing = sum(is.na(temp))

temp[is.na(temp)] = rnorm(n.missing, mean = temp.mean, sd = temp.sd)

return(temp)

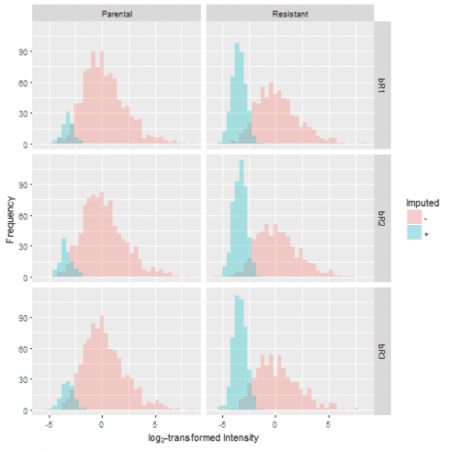
})

return(df)

}

## Apply imputation

df.FNI = impute\_data(df.FN)

Let's graphically evaluate the results by overlaying the distribution of the imputed values over the original distribution. In doing so, we observe that the number of missing values is greater in the resistant condition compared to the control. Furthermore, the missing values take on a narrow spread at the lower end of the sample distribution, which reflects our notion that low levels of protein expression produce missing data.  
[](https://i2.wp.com/datascienceplus.com/wp-content/uploads/2018/08/unnamed-chunk-52-1.png?ssl=1)

**Summary**

This is the second of three tutorials on proteomics data analysis. I have described the approach to handling the missing value problem in proteomics.

In the final tutorial, we are ready to compare protein expression between the drug-resistant and the control lines. This involves performing a two-sample Welch's t-test on our data to extract proteins that are differentially expressed. Moreover, we will discuss ways to interpret the final output of a high-throughput proteomics experiment. Stay tuned for the revelation of proteins that may play a role in driving the resistance of tumor cells.