

Systems biology

bbeaR: an R package and framework for epitope-specific antibody profiling

Maria Suprun ¹, Randall J. Ellis², Hugh A. Sampson¹ and Mayte Suárez-Fariñas^{3,4,*}

¹Department of Pediatrics, Allergy and Immunology, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA, ²Departments of Neuroscience and Psychiatry, Addiction Institute of Mount Sinai, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA, ³Department of Population Health Science and Policy, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA and ⁴Department of Genetics and Genomic Sciences, Icahn Institute for Data Science and Genomic Technology, Icahn School of Medicine at Mount Sinai, New York, NY 10029, USA

*To whom correspondence should be addressed.

Associate Editor: Alfonso Valencia

Received on June 18, 2020; revised on October 13, 2020; editorial decision on December 6, 2020; accepted on January 16, 2021

Abstract

Summary: Analysis of epitope-specific antibody repertoires has provided novel insights into the pathogenesis of inflammatory disorders, especially allergies. A novel multiplex immunoassay, termed Bead-Based Epitope Assay (BBEA), was developed to quantify levels of epitope-specific immunoglobulins, including IgE, IgG, IgA and IgD isotypes. *bbeaR* is an open-source R package, developed for the BBEA, provides a framework to import, process and normalize .csv data files exported from the Luminex reader, evaluate various quality control metrics, analyze differential epitope-binding antibodies with linear modeling, visualize results and map epitopes' amino acid sequences to their respective primary protein structures. *bbeaR* enables streamlined and reproducible analysis of epitope-specific antibody profiles.

Availability and implementation: *bbeaR* is open-source and freely available from GitHub as an R package: <https://github.com/msuprun/bbeaR>; vignettes included.

Contact: mayte.suarezfarinas@mssm.edu

Supplementary information: [Supplementary data](#) are available at *Bioinformatics* online.

1 Introduction

Antibodies are part of the adaptive humoral immunity and offer essential protection against a variety of pathogens. When antibodies recognize a specific site on an antigenic protein, i.e. an epitope, an immune complex is formed, allowing the immune system to fight the pathogen (Lu et al., 2018; Sela-Culang et al., 2013). The variable domain of an antibody recognizes a wide range of molecules, while the constant domain defines its isotype (i.e. IgM, IgD, IgG, IgA and IgE) and effector functions (Lu et al., 2018). In the case of viral or bacterial infection, IgG and IgM antibodies could neutralize or help clear the pathogen (Forthal, 2014). In allergy, where an innocuous protein is recognized as an antigen, cross-linked IgE activates mast cells and basophils, which readily release inflammatory molecules such as histamine, heparin and tryptase, leading to allergic reactions (Burton and Oettgen, 2011). Quantifying epitope-specific antibodies has many applications, including (i) diagnostics, where the amount of IgE specific to different epitopes can indicate clinical reactivity and severity of allergy (Sackesen et al., 2019; Santos et al., 2020), (ii) prognostics, where isotypes induced during immunotherapy can identify responders (Suarez-Farinas et al., 2019; Sugimoto et al., 2016), (iii) insights into the natural disease evolution and pathogenesis (Suprun et al., 2020).

Bead-Based Epitope Assay (BBEA) is a Luminex-based high-throughput assay developed to measure the levels of many epitope-specific antibodies simultaneously and outperforms peptide microarrays (Suprun et al., 2019). Here, we present an open-source R package, *bbeaR*, which facilitates the management, preprocessing and quality control of BBEA data (Fig. 1). It creates a data structure that eases the differential antibodies analysis using *limma* or linear modeling and contains custom visualizations. *bbeaR* comes with a vignette with a detailed walkthrough of two examples, showcasing the package utility. While the framework is described using an example of food allergies, it can be applied to any disease or experimental condition where antibody responses are of interest. To the authors' knowledge, *bbeaR* is the only R package currently available providing comprehensive pipeline for import, quality control and analytics.

2 Materials and methods

BBEA is run on 96-well microplates, allowing the measurement of many plasma/serum epitope-specific antibodies on many patients simultaneously (Breen, 2017). We recommend that experiments are randomized using *PlateDesigner* (Suprun and Suárez-Fariñas, 2019), to avoid

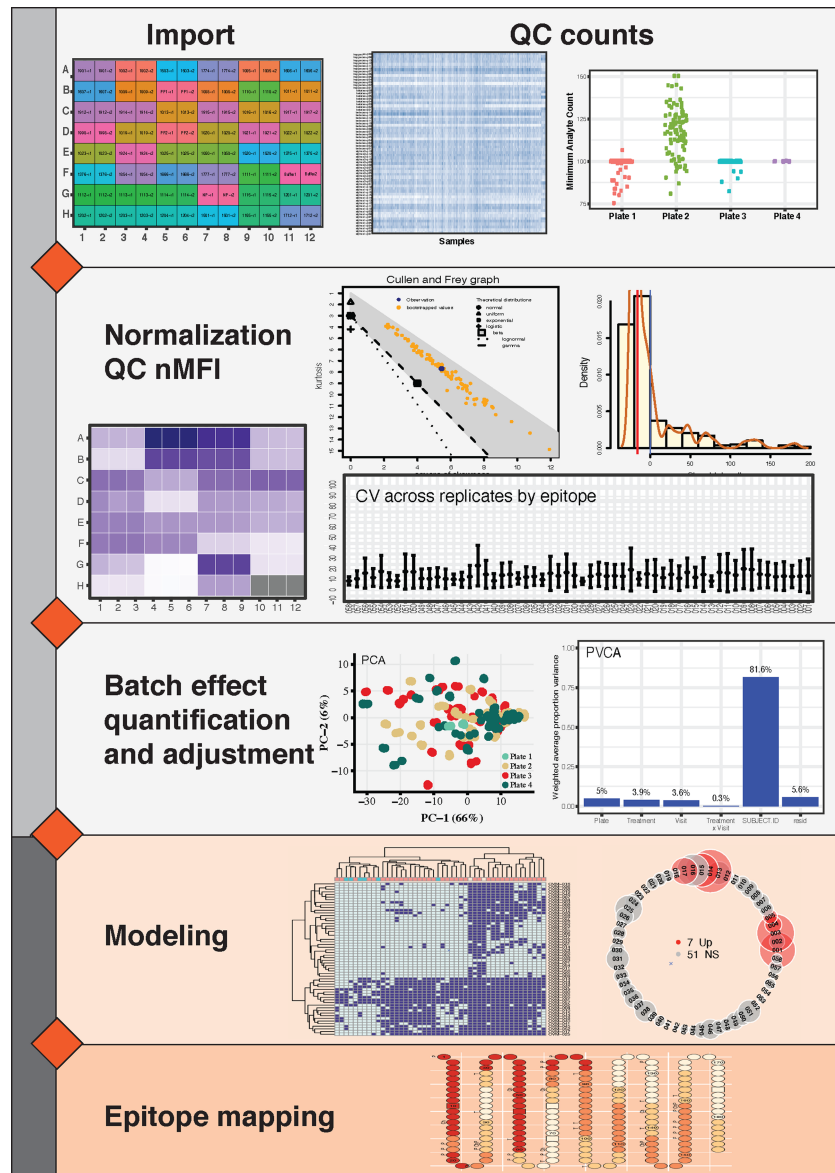


Fig. 1. Functionality schematic of *bbeAR* for data import, normalization, quality control (QC), analysis and visualizations

confounding by technical factors, such as plate or well position. *PlateDesigner*'s experimental metadata can be uploaded directly to the Luminex reader, avoiding data entry errors. The assay's readout, exported from Luminex as a comma-separated values (.csv) file for each plate, is a starting point of the *bbeAR* pipeline.

Csv files from each plate are imported simultaneously, extracting counts and Median Fluorescence Intensity (MFI) tables, along with metadata about the plate run. The MFI is provided for each Luminex bead, i.e. an analyte, which uniquely corresponds to an individual epitope, and annotations are mapped. The counts undergo a first round of quality controls, where samples or antibodies can be removed if they do not reach sufficient reads, e.g. at least 25 counts per analyte (Bjornstal et al., 2011). Then, for each epitope-specific antibody j and sample i , the MFI signal is normalized as

$$Y_{ij} = \log_2(MFI_{ij} + \text{offset})$$

$$nMFI_{ij} = \begin{cases} 0, & Y_{ij} < \frac{\sum_{k \in BG} Y_{kj}}{N_{BG}}, \\ a \left(Y_{ij} - \frac{\sum_{k \in BG} Y_{kj}}{N_{BG}} \right), & \text{otherwise} \end{cases}$$

where BG are background wells. The distribution of nMFI is evaluated using Cullen-Frey plots and histograms, and agreement between technical replicates assessed by plate-layout plots, coefficients of variation (CV) and intra-class correlation coefficients (ICC). If an experiment consists of several microplate runs, plate effect can be identified using principal component analysis (PCA) and the nMFI can be adjusted for plate and well effects using a *limma* model (Silver et al., 2009; Smyth, 2004; Suprun et al., 2019) or other approaches like SVA/ComBat (Leek et al., 2012).

The differential epitope-specific antibodies can be identified by comparing patient groups, using either classical linear models or *limma* (Silver et al., 2009; Smyth, 2004). We have implemented custom *net circle* visualization plots that present fold-changes and P -values across antibodies ordered by the epitopes' position on the protein. Additionally, amino acid sequences of the epitopes are mapped to the protein using the topology plot and can be quite useful when a 3D crystal structure is not available. This plot extracts protein characteristics from the UniProt database, and displays additional information, i.e. sites of disulfide bridges, glycosylation and enzymatic cleavage.

3 Usage example

The features of *bbeeR* are illustrated in two detailed examples, provided as vignettes: (i) comparison of egg epitope-specific IgE antibodies in egg-allergic children and controls (Suprun *et al.*, 2020) and (ii) immunomodulation of milk epitope-specific IgEs in patients undergoing oral immunotherapy (Suarez-Farinas *et al.*, 2019) (Supplementary Files S1 and S2). While the egg experiment was performed on one microplate, the milk experiment was performed on four plates and required a plate effect adjustment. A brief walk-through of the milk allergy project is outlined below.

Levels of IgE specific to 66 allergenic milk epitopes were measured before and after treatment in 47 milk-allergic patients undergoing two years of immunotherapy with either omalizumab or placebo adjuvant (Suarez-Farinas *et al.*, 2019). First, to make sure the MFI values are valid, it is important to have sufficient numbers of analyte (that represent epitopes) counts. In all of the plates, minimum counts were above 75, which is considered reliable (Bjornstal *et al.*, 2011). nMFI values are stored in *ExpressionSet* object along with epitope annotation and patients' phenotypic or clinical information, allowing for a seamless use of commonly used -omic methods. PCA visualization indicates the presence of a plate effect, which can be quantified by principal variance component analysis (Bushel, 2019; Li *et al.*, 2009) to account for ~5% of the total variability. Since the samples were randomly allocated to plates, the plate effect was adjusted using linear models. The distribution of the nMFIs is investigated through the Cullen–Frey plot (Delignette-Muller and Dutang, 2015), which is shown to be close to log-normal. As such, parametric models using the nMFI can be used in downstream analyses. Reliability across technical replicates are measured by CV and ICC. After the reliability is ascertained, the replicates are averaged. Changes from baseline to the end of treatment in placebo and omalizumab groups are modeled with *limma* and show decreased levels of more epitope-specific IgEs in the placebo arm, visualized by the *net circle* plots.

4 Conclusion

bbeeR provides a flexible functionality for analyzing epitope-specific antibodies quantified with the BBEA or other Luminex-based platforms. After data loading and pre-processing, the workflow is similar to that of microarray and RNA-seq pipelines, allowing users to implement other tools developed for these high-throughput technologies. *bbeeR* allows seamless and reproducible profiling of epitope-specific antibodies in R.

Acknowledgements

The authors thank Robert Getts, Helena L. Chang, Rohit Raghunathan and Ronaldo Silva for the technical input on the development of the package.

Funding

M.S. was funded by the Integrated Pharmacological Sciences Training grant from the National Institute of General Medical Sciences (NIGMS) [T32GM062754]. H.A.S. was funded in part by a grant from National Institute of Allergy and Infectious Diseases (NIAID/NIH) [UM1 AI109565]. M.S.-F. was partially supported by an institutional grant from AllerGenis LLC.

Conflict of Interest: none declared.

Data availability

The data underlying this article are available as part of the *bbeeR* package.

References

- Bjornstal, O. *et al.* (2011) The impact of decreased bead count to determine concentrations of amyloid beta1-42, total-tau, and phosphorylated-tau181 in human cerebrospinal fluid using xMAP technology. *J. Pharm. Sci.*, **100**, 4655–4663.
- Breen, E.J. (2017) Protein multiplexed immunoassay analysis with R. *Methods Mol. Biol.*, **1619**, 495–537.
- Burton, O.T. and Oettgen, H.C. (2011) Beyond immediate hypersensitivity: evolving roles for IgE antibodies in immune homeostasis and allergic diseases. *Immunol. Rev.*, **242**, 128–143.
- Bushel, P. (2019) *pvca: Principal Variance Component Analysis (PVCa)*. R package version 1.26.0. <https://bioconductor.org/packages/release/bioc/html/pvca.html> (10 May 2020, date last accessed).
- Delignette-Muller, M.L. and Dutang, C. (2015) *fitdistrplus: an R package for fitting distributions*. **64**, 34.
- Forthal, D.N. (2014) Functions of antibodies. *Microbiol. Spectr.*, **2**, 1–17.
- Leek, J.T. *et al.* (2012) The *sva* package for removing batch effects and other unwanted variation in high-throughput experiments. *Bioinformatics*, **28**, 882–883.
- Li, J. *et al.* (2009) Principal variance components analysis: estimating batch effects in microarray gene expression data. In: Scherer, A. (ed.) *Batch Effects and Noise in Microarray Experiments: Sources and Solutions*. John Wiley & Sons, Ltd. pp. 141–154.
- Lu, L.L. *et al.* (2018) Beyond binding: antibody effector functions in infectious diseases. *Nat. Rev. Immunol.*, **18**, 46–61.
- Sackesen, C. *et al.* (2019) A new Luminex-based peptide assay to identify reactivity to baked, fermented, and whole milk. *Allergy*, **74**, 327–336.
- Santos, A.F. *et al.* (2020) IgE to epitopes of Ara h 2 enhance the diagnostic accuracy of Ara h 2-specific IgE. *Allergy*, **75**, 2309–2318.
- Sela-Culang, I. *et al.* (2013) The structural basis of antibody–antigen recognition. *Front. Immunol.*, **4**, 302.
- Silver, J.D. *et al.* (2009) Microarray background correction: maximum likelihood estimation for the normal-exponential convolution. *Biostatistics*, **10**, 352–363.
- Smyth, G.K. (2004) Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Stat. Appl. Genet. Mol. Biol.*, **3**, Article3.
- Suarez-Farinas, M. *et al.* (2019) Predicting development of sustained unresponsiveness to milk oral immunotherapy using epitope-specific antibody binding profiles. *J. Allergy Clin. Immunol.*, **143**, 1038–1046.
- Sugimoto, M. *et al.* (2016) Differential response in allergen-specific IgE, IgGs, and IgA levels for predicting outcome of oral immunotherapy. *Pediatr. Allergy Immunol.*, **27**, 276–282.
- Suprun, M. *et al.* (2020) Ovomucoid epitope-specific repertoire of IgE, IgG4, IgG1, IgA1 and IgD antibodies in egg allergic children. *Allergy*, **75**, 2633–2643.
- Suprun, M. *et al.* (2019) Novel Bead-Based Epitope Assay is a sensitive and reliable tool for profiling epitope-specific antibody repertoire in food allergy. *Sci. Rep.*, **9**, 18425.
- Suprun, M. and Suárez-Fariñas, M. (2019) PlateDesigner: a web-based application for the design of microplate experiments. *Bioinformatics*, **35**, 1605–1607.

bbeaR - Egg Allergy Example

Maria Suprun, Randall J. Ellis, Hugh A. Sampson, Mayte Suárez-Fariñas

10/08/2020

Contents

About	1
Installation	2
Raw Data Import	2
Quality Control of the Raw Data	6
Data Normalization	8
QC of Normalized Data	9
Distributions	9
Technical Replicates	11
Averaging Technical Replicates	14
Differential Analysis - Limma Modeling	16
Protein Topology Plot	18

About

The **Bead-Based Epitope Assay** (*BBEA*) can be used to quantify the amount of epitope- or peptide-specific antibodies (e.g., IgE) in plasma or serum samples. A detailed assay description is outlined in this publication.

Previous studies used peptide microarray technology and found that immunoglobulin (Ig)E specific to sequential epitopes of ovomucoid protein, a major allergen in egg-white, could be a marker of egg allergy diagnosis and persistence (Martinez-Botas et al (2013) and Jarvinen et al (2007)). In this study and tutorial, we evaluated if BBEA can be used to identify IgE-binding epitopes. We analyzed levels of IgE to 58 peptides (15-mer, 12 amino acid overlap), covering the entire sequence of hen's egg-white ovomucoid protein, measured using *BBEA* in 38 allergic children and 6 controls.

We will start by installing *bbeaR* and reading in the *BBEA*'s raw data (alternatively, one can load an R dataset that comes with this package). We will look at several quality control (QC) measures and normalize the

data. Then we will create a topology plot, to highlight immunodominant regions on the ovomucoid protein. At the end, we will demonstrate an approach to identify sequential epitope-specific (ses)IgE antibodies that are different between allergic children and controls, using limma modeling framework.

Installation

bbeaR is an R package and is installed using a standard GitHub code:

```
library(devtools)
install_github('msuprun/bbeaR')
```

Loading additional packages that will be used in the analyses.

```
library(bbeaR)
library(plyr)
library(stringr)
library(ggplot2)
library(gridExtra)
library(pheatmap)
library(RColorBrewer)
library(limma)
```

Raw Data Import

44 patient samples were assayed in duplicates, ran within the same batch on one 96-well plate. The run additionally included two positive control pools (aka “PP”, a mix of plasma from several allergic patients), a negative pool (aka “NP”, a mix of plasma from several healthy patients), and 2 wells without any sample (aka “Buffer”, for a background quantification).

The original .csv files from the Luminex-200 assay, generated with the xPONENT® software, can be downloaded here.

The .csv sheet consists of several sections, of which “Median”, “Count”, and “NetMFI” hold the assay’s readout values for each epitope-specific antibody (j) in columns and each sample (i) in rows:

Data Type:	Median										
Location	Sample	Analyte 7	Analyte 8	Analyte 10	Analyte 12	Analyte 15	Analyte 19	Analyte 20	Analyte 21	Analyte 22	Analyte 27
1(1,A1)	1901-r1	1	1	0.5	1	1	1	1	1	2	1
2(1,A2)	1901-r2	0	1	0	1	0	1	1	1	2	1
3(1,A3)	1002-r1	0	1	1	0	1	1	0	1	1	1
4(1,A4)	1002-r2	1	1	1	1	1	1	1	1	1	0
5(1,A5)	1503-r1	0	0	1	1	1.5	1	2	1	2	0
6(1,A6)	1503-r2	0	0.5	0	1	1	1	1	1	1	1
7(1,A7)	1774-r1	0	0	1	1	1	1	0	1	2	1
8(1,A8)	1774-r2	0	0	0	1	1	1	0	1	1	1
9(1,A9)	1005-r1	1	1	1	1	1	2	1	1	2	1
10(1,A10)	1005-r2	1	1	1	1	1	1	1	2	2	1
11(1,A11)	1606-r1	9	14.5	1	1	3.5	1	19	3	3	1
12(1,A12)	1606-r2	10	18	2	1	4	1	25	3	2	1
13(1,B1)	1607-r1	1	1	1	1	1	1	1	1	2	1
14(1,B2)	1607-r2	1	0	1	1	1	1	1	2	1	1
15(1,B3)	1009-r1	373	764	5	1	10	1	2	15	8	1
16(1,B4)	1009-r2	392.5	831.5	5	1	9	2	2	15	8	1
17(1,B5)	PP1-r1	29	67	26	55	10	3	5	8	7	47
18(1,B6)	PP1-r2	28.5	64.5	29	57	10	3	4	7	8	61

First, we need to create a plate layout using the `create.plate.db()` function. This layout will be used in the import and some of the plotting functions. The only input to this function is the direction of the plate read (horizontal or vertical) that was used by the Luminex instrument.

```
l <- create.plate.db(direction = "horizontal")
plate.design.db <- l$plate.design.db
plate.design <- l$plate.design
plate.design
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## [1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9" "A10" "A11" "A12"
## [2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9" "B10" "B11" "B12"
## [3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9" "C10" "C11" "C12"
## [4,] "D1" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9" "D10" "D11" "D12"
## [5,] "E1" "E2" "E3" "E4" "E5" "E6" "E7" "E8" "E9" "E10" "E11" "E12"
## [6,] "F1" "F2" "F3" "F4" "F5" "F6" "F7" "F8" "F9" "F10" "F11" "F12"
## [7,] "G1" "G2" "G3" "G4" "G5" "G6" "G7" "G8" "G9" "G10" "G11" "G12"
## [8,] "H1" "H2" "H3" "H4" "H5" "H6" "H7" "H8" "H9" "H10" "H11" "H12"
```

Then read a `.csv` file of the one plate.

Note: a separate tutorial (Milk Allergy Example) has an example of importing and processing multiple plates.

```
bbeaEgg <- bbea.read.csv(fname = "BBEA_OVM_58p_IgE_patients.csv")
```

```
## [1] "Reading file: BBEA_OVM_58p_IgE_patients.csv"
```

The `bbea` list object contains several elements, extracted from the assay's output.

```
names(bbeaEgg)
```

```
## [1] "Median"      "NetMFI"      "Count"      "AssayInfo" "pData"
```

The **Median**, **NetMFI**, and **Count** are matrices with rows as Analytes (epitopes) and columns as Samples. Note: this is changed from the original data layout, where samples were in rows and epitopes in columns, to make the dataset usable with common analytical tools in R.

The **Median** are the Median Fluorescence Intensities (MFIs) and the **NetMFI** are the Medians normalized to background. In our example, **Median** and **NetMFI** have exactly same values, since normalization was not selected during the assay run. This post has more details about the Luminex outputs.

```
bbeaEgg$Median[1:5, 1:2]
```

```
##      OVM_58p_IgE_patients_1(1,A1) OVM_58p_IgE_patients_2(1,A2)
## Analyte 7                        1.0                        0
## Analyte 8                        1.0                        1
## Analyte 10                       0.5                        0
## Analyte 12                       1.0                        1
## Analyte 15                       1.0                        0
```

The **Count** are the numbers of beads counted per analyte, and is important quality control measure.

```
bbeaEgg$Count[1:5, 1:2]
```

```
##           OVM_58p_IgE_patients_1(1,A1) OVM_58p_IgE_patients_2(1,A2)
## Analyte 7                               80                               91
## Analyte 8                               75                               84
## Analyte 10                              132                              114
## Analyte 12                              129                               87
## Analyte 15                              102                               81
```

The **AssayInfo** saves parameters of the assay.

```
bbeaEgg$AssayInfo[1:15, 1:2]
```

```
##           V1           V2
## 1           Program      xPONENT
## 2           Build        3.1.971.0
## 3           Date         6/25/2019
## 4
## 5           SN           LX10014268401
## 6           Batch OVM_58p_IgE_patients
## 7           Version      1
## 8           Operator
## 9           ComputerName  XPONENT31-PC
## 10          Country Code  409
## 11          ProtocolName OVM_58p_c70_6.25.2019
## 12          ProtocolVersion 1
## 13          ProtocolDescription
## 14 ProtocolDevelopingCompany
## 15          SampleVolume  100 uL
```

Finally, *bbea.read.csv()* creates a **phenotype (p)Data** that has some basic information about the samples and the assay run.

```
colnames(bbeaEgg$pData)
```

```
## [1] "Location"      "Sample"        "filename"
## [4] "File"          "Plate"         "SampleNumber"
## [7] "Well.Number"   "Well.Letter"   "Well_coord"
## [10] "print.plate.order" "letters_numeric" "Plate.Date"
## [13] "Plate.Time"    "Plate.TimeHR"   "CountSum"
## [16] "CountMean"     "CountMin"       "CountMax"
```

```
bbeaEgg$pData[1:2, 1:2]
```

```
##           Location Sample
## OVM_58p_IgE_patients_1(1,A1) 1(1,A1) 1901-r1
## OVM_58p_IgE_patients_2(1,A2) 2(1,A2) 1901-r2
```

We can now add external information about the samples and analytes.

We will load already imported data that includes **phenotype (clinical) data** *PDegg* and an annotation file *AnnotEgg*. Note: this will also load a *bbea* list object (generated in the previous steps).

```
data(Egg)
```

The annotation **AnnotEgg** dataset contains the mapping of Luminex beads (Analytes) to the peptides/epitopes.

```
dim(AnnotEgg)
```

```
## [1] 58 5
```

```
head(AnnotEgg)
```

```
##      Analyte Peptide Protein PeptideName lableName
## OVM-001 Analyte 38 OVM-001    OVM         001      001
## OVM-002 Analyte 40 OVM-002    OVM         002      002
## OVM-003 Analyte 41 OVM-003    OVM         003      003
## OVM-004 Analyte 42 OVM-004    OVM         004      004
## OVM-005 Analyte 44 OVM-005    OVM         005      005
## OVM-006 Analyte 71 OVM-006    OVM         006      006
```

Changing the *bbeaEgg* object to have peptide names instead of analyte numbers.

```
bbeaEgg <- bbea.changeAnnotation(bbea.obj = bbeaEgg,
                                annotation = AnnotEgg,
                                newNameCol = "Peptide",
                                AnalyteCol = "Analyte")
bbeaEgg$Median[1:5, 1:2]
```

```
##      OVM_58p_IgE_patients_1(1,A1) OVM_58p_IgE_patients_2(1,A2)
## OVM-009                        1.0                        0
## OVM-031                        1.0                        1
## OVM-011                        0.5                        0
## OVM-014                        1.0                        1
## OVM-020                        1.0                        0
```

The **PDegg** dataset has clinical information about our samples.

```
dim(PDegg)
```

```
## [1] 44 4
```

```
head(PDegg)
```

```
##   PTID Age Egg.sIgE      Group
## 1 1901  NA      NA Atopic Control
## 2 1002  16      NA Atopic Control
## 3 1503   2    2.29    Egg Allergy
## 4 1774   7   16.80    Egg Allergy
## 5 1005   7   76.70    Egg Allergy
## 6 1606   4   35.20    Egg Allergy
```

We will clean up the pData and then merge it with PDegg. This step is not necessary, but will be useful when we do statistical modelling.

```
bbeaEgg$pData <- mutate(bbeaEgg$pData,
  PTID = sapply(strsplit(Sample, "\\-"), head, 1),
  SampleType = ifelse(grepl("Buff", Sample), "Buffer",
    ifelse(grepl("PP|NP", Sample),
      sapply(strsplit(Sample, "\\-"), head, 1), "Patient")))
PDm <- merge(bbeaEgg$pData, PDegg, by = "PTID", all.x = T)
rownames(PDm) <- PDm$File
bbeaEgg$pData <- PDm[colnames(bbeaEgg$Median),] # making sure samples are arranged correctly
```

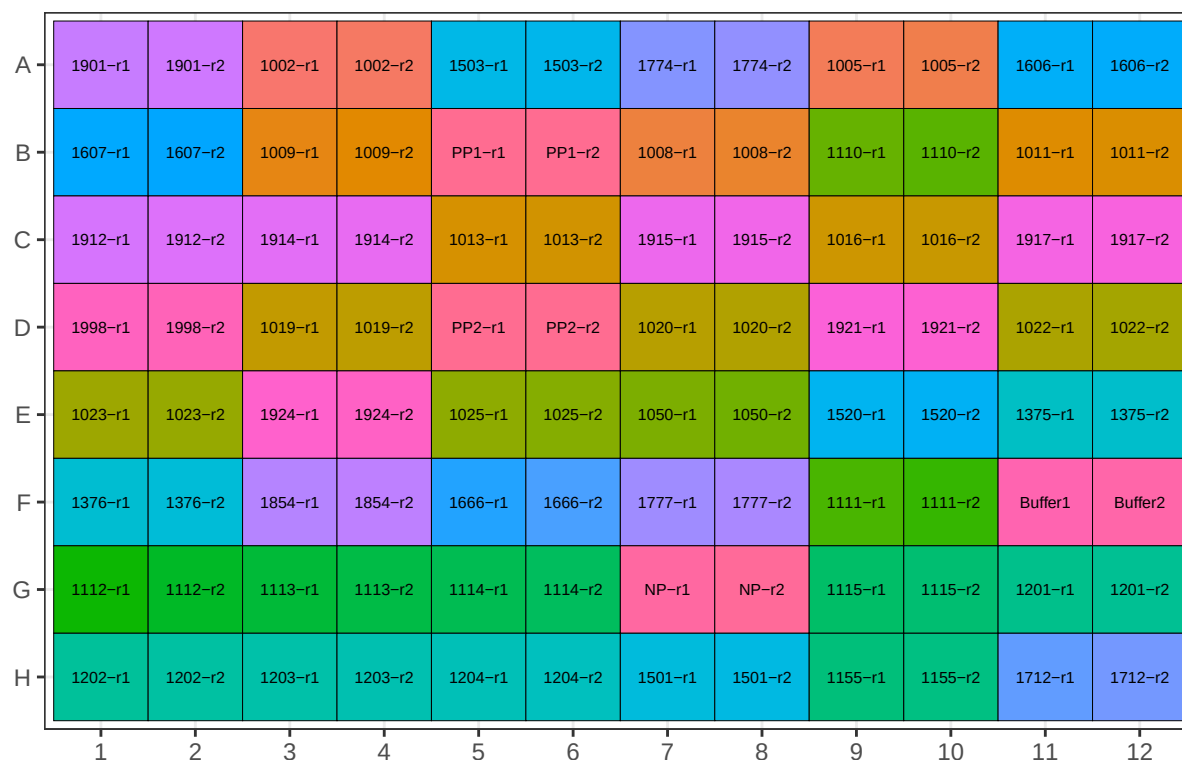
Quality Control of the Raw Data

Layout of the experimental plate.

Tip: to make sure no position bias is present in the data, samples have to be randomized among wells and plates, as outlined in this publication.

```
bbea.QC.Image(bbeaEgg$pData,
  filename = "QC.Image.",
  plate.design.db = plate.design.db)
```

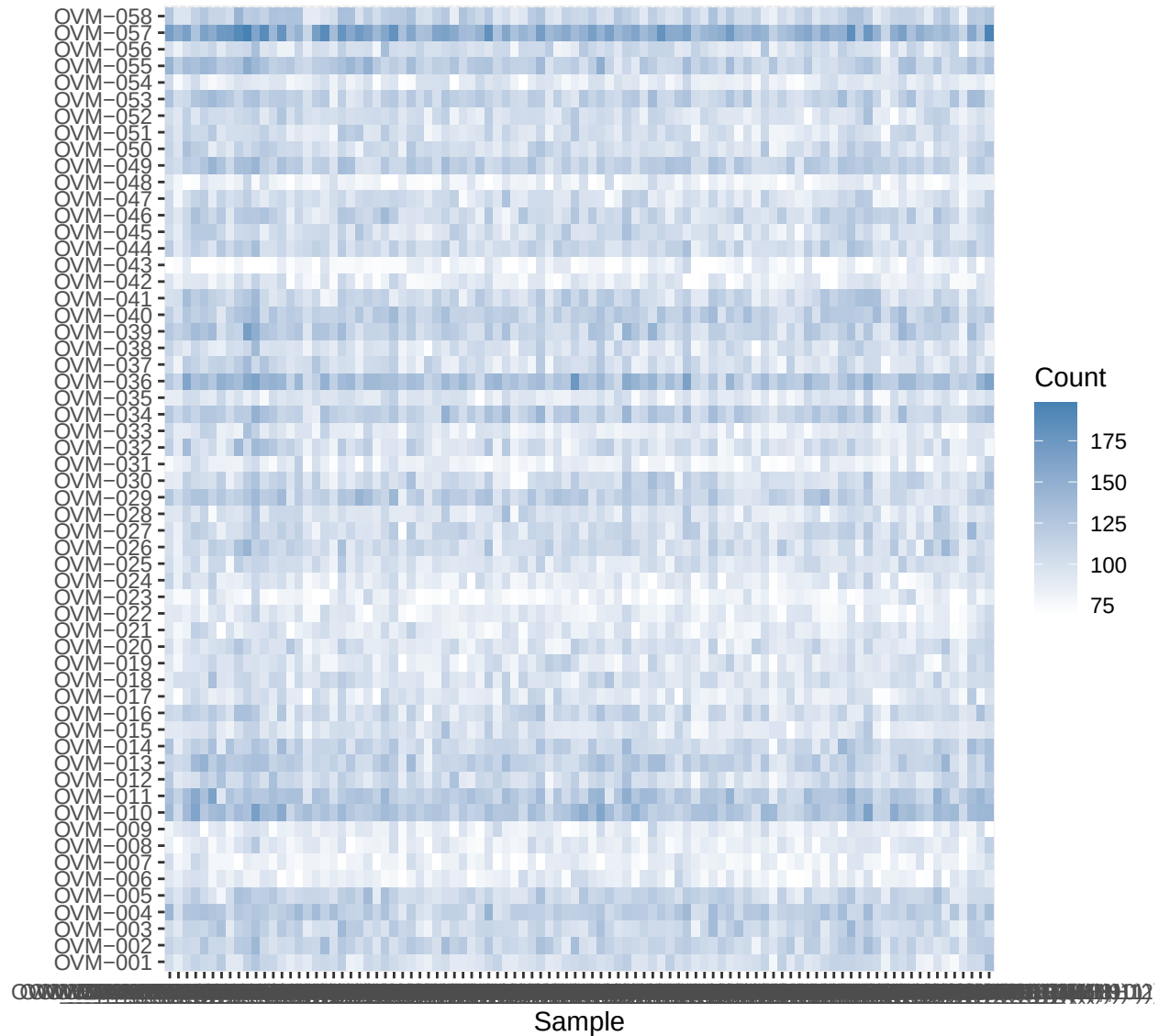
Experimental Date: 20190625



We want to make sure that there are no missing samples or analytes. This would be reflected by the very low counts (<25).

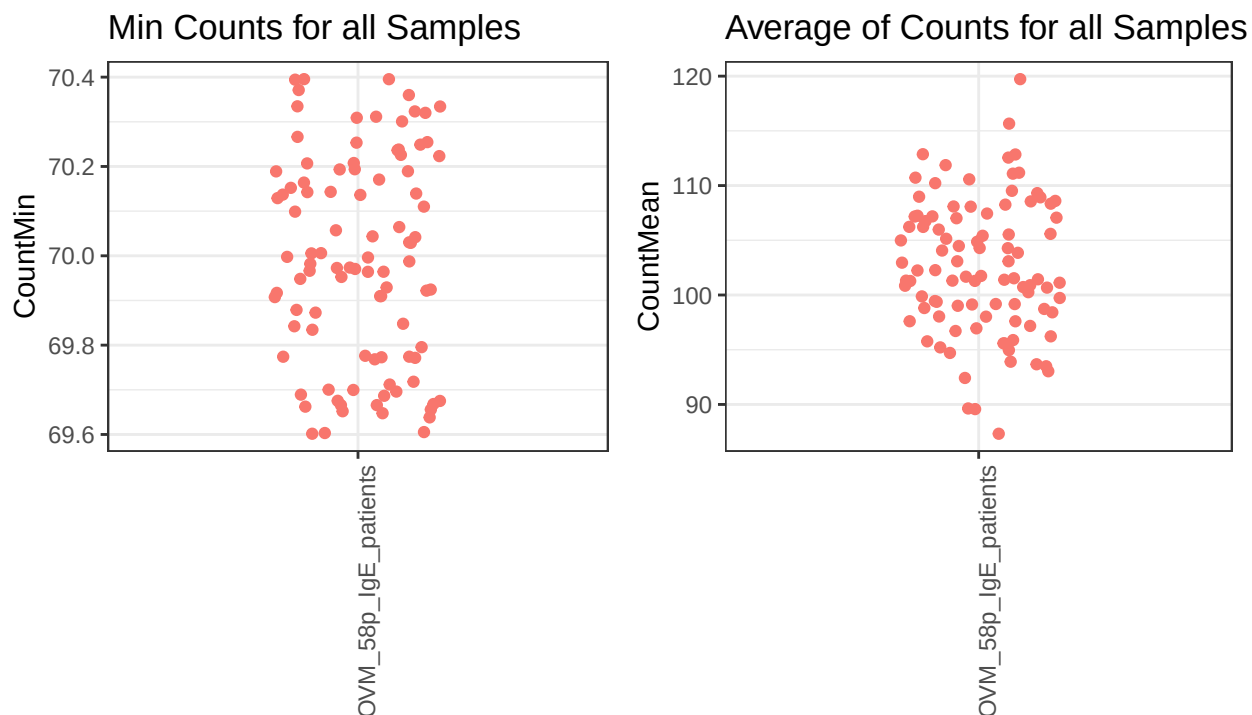
The heatmap shows that all samples and analytes were included in the experimental run.

```
bbea.QC.heatmap.counts(bbeaEgg,
  getlog2 = FALSE,
  filename = "QC.CountsHeatmap.pdf",
  plateVar = "Plate", ann = NULL)
```



Now we look at the overall distribution of counts: the minimum count is ~69, which is pretty good.

```
p<-bbea.QC.Samples(bbeaEgg,
  filename = "QC.",
  plateVar = 'Plate',
  gt = 25)
grid.arrange(p$pmin, p$pavg, nrow=1)
```



Our counts look good, so we don't need to exclude any samples. However, if this were not the case, samples with low counts can be removed using the `bbea.subset()` function, only keeping samples with average counts > 25 .

```
bbea.sub <- bbea.subset(bbeaEgg, statement = (bbeaEgg$pData$CountMean > 25))
```

Data Normalization

We convert the **Median** to normalized **nMFI** by taking the $\log_2$ of values and subtracting the average of the background wells. Note: the *Sample* column of the **pData** should include a “Buffer” string in the wells dedicated for the background.

```
bbeaEgg$pData$Sample # samples 71 & 72 are the background wells
```

```
## [1] "1901-r1" "1901-r2" "1002-r1" "1002-r2" "1503-r1" "1503-r2" "1774-r1"
## [8] "1774-r2" "1005-r1" "1005-r2" "1606-r1" "1606-r2" "1607-r1" "1607-r2"
## [15] "1009-r1" "1009-r2" "PP1-r1" "PP1-r2" "1008-r1" "1008-r2" "1110-r1"
## [22] "1110-r2" "1011-r1" "1011-r2" "1912-r1" "1912-r2" "1914-r1" "1914-r2"
## [29] "1013-r1" "1013-r2" "1915-r1" "1915-r2" "1016-r1" "1016-r2" "1917-r1"
## [36] "1917-r2" "1998-r1" "1998-r2" "1019-r1" "1019-r2" "PP2-r1" "PP2-r2"
## [43] "1020-r1" "1020-r2" "1921-r1" "1921-r2" "1022-r1" "1022-r2" "1023-r1"
## [50] "1023-r2" "1924-r1" "1924-r2" "1025-r1" "1025-r2" "1050-r1" "1050-r2"
## [57] "1520-r1" "1520-r2" "1375-r1" "1375-r2" "1376-r1" "1376-r2" "1854-r1"
## [64] "1854-r2" "1666-r1" "1666-r2" "1777-r1" "1777-r2" "1111-r1" "1111-r2"
## [71] "Buffer1" "Buffer2" "1112-r1" "1112-r2" "1113-r1" "1113-r2" "1114-r1"
## [78] "1114-r2" "NP-r1" "NP-r2" "1115-r1" "1115-r2" "1201-r1" "1201-r2"
## [85] "1202-r1" "1202-r2" "1203-r1" "1203-r2" "1204-r1" "1204-r2" "1501-r1"
## [92] "1501-r2" "1155-r1" "1155-r2" "1712-r1" "1712-r2"
```

```
bbeaN <- MFI2nMFI(bbeaEgg,
  offset = 0.5, # a constant to add to avoid taking a log of 0
  rmNeg = TRUE) # if a value of a sample is below background, assign "0"
```

```
## [1] "OVM_58p_IgE_patients"
```

```
names(bbeaN)
```

```
## [1] "nMFI"      "NetMFI"     "Count"      "AssayInfo" "pData"
```

```
bbeaN$nMFI[1:5, 1:2]
```

```
##          OVM_58p_IgE_patients_1(1,A1) OVM_58p_IgE_patients_2(1,A2)
## OVM-009                      0.7924813                0.000000
## OVM-031                      1.5849625                1.584963
## OVM-011                      0.0000000                0.000000
## OVM-014                      1.5849625                1.584963
## OVM-020                      0.0000000                0.000000
```

R object of class ExpressionSet *eset* is convenient for a high-throughput data analysis.
bbea object can be converted to *eset*:

```
eset <- nMFI2Eset(nMFI.object = bbeaN)
eset
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 58 features, 94 samples
## element names: exprs
## protocolData: none
## phenoData
## sampleNames: OVM_58p_IgE_patients_1(1,A1)
## OVM_58p_IgE_patients_2(1,A2) ... OVM_58p_IgE_patients_96(1,H12) (94
## total)
## varLabels: PTID Location ... Group (23 total)
## varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
```

QC of Normalized Data

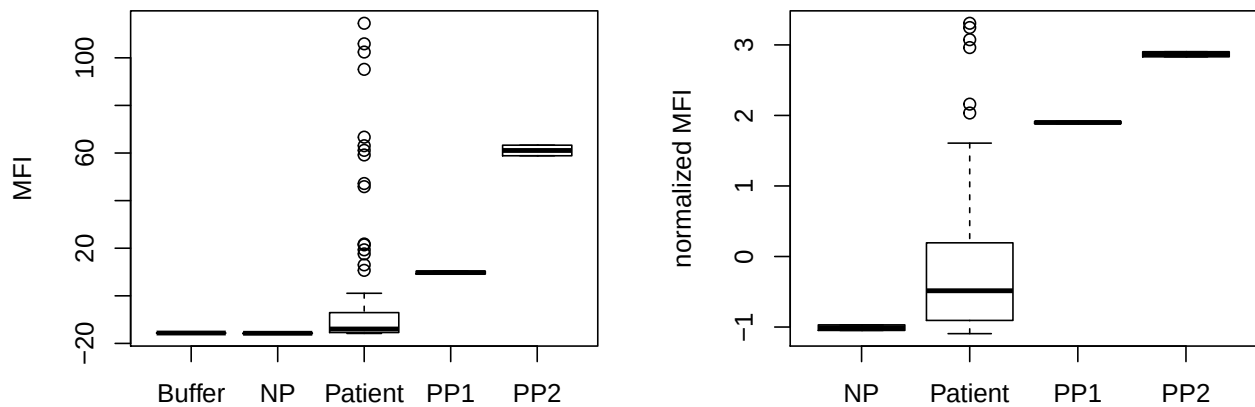
Distributions

Overall distribution:

Since there are different levels of the antibody to each peptide, the data will be scaled before plotting. Y-axis of the boxplot represents a mean MFI or nMFI of 50 scaled peptides. The plots show that IgE to sequential peptides, as expected, is higher in egg allergic patients and positive pools (PP), but not background wells (buffer) or negative control pool (NP).

```
adjMFI <- t(apply(as.matrix(bbeaEgg$Median), 1, function(x){x - mean(x, na.rm=T)}))
adjnMFI<-t(apply(as.matrix(exprs(eset)), 1, function(x){x - mean(x, na.rm=T)}))

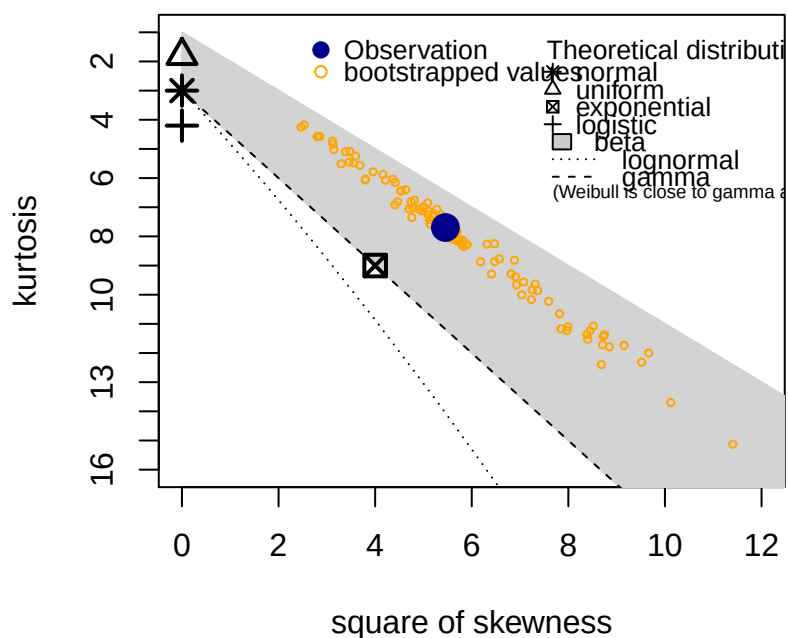
par(mfrow = c(1, 2))
boxplot(colMeans(adjMFI) ~ bbeaEgg$pData$SampleType,
        xlab = " ", ylab = "MFI")
boxplot(colMeans(adjnMFI) ~ eset$SampleType,
        xlab = " ", ylab = "normalized MFI")
```



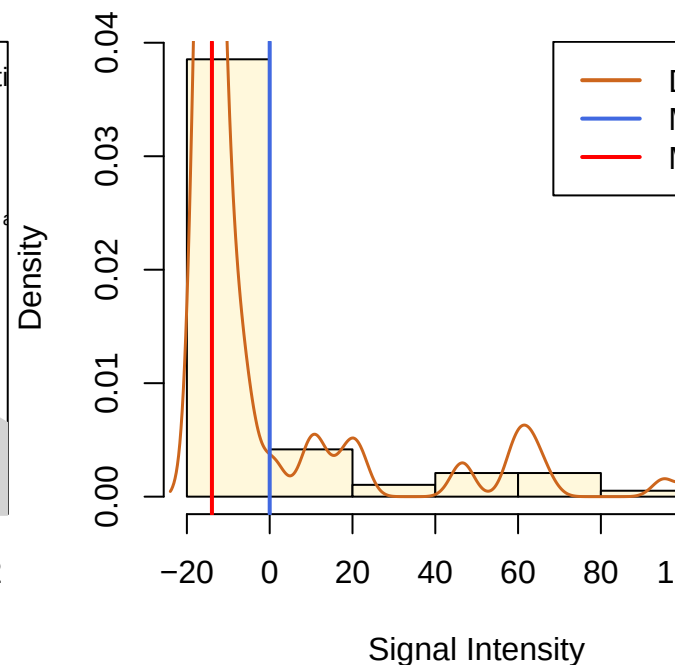
Cullen-Frey plots can be used to evaluate the distribution of the data. It shows how the skewness and kurtosis of our data compare to the theoretical distributions. *CullenFreyPlot()* function is a wrapper of the `fitdistrplus::descdist()`.

```
CullenFreyPlot(adjMFI, filename = "QC.CullenFrey.MFI")
```

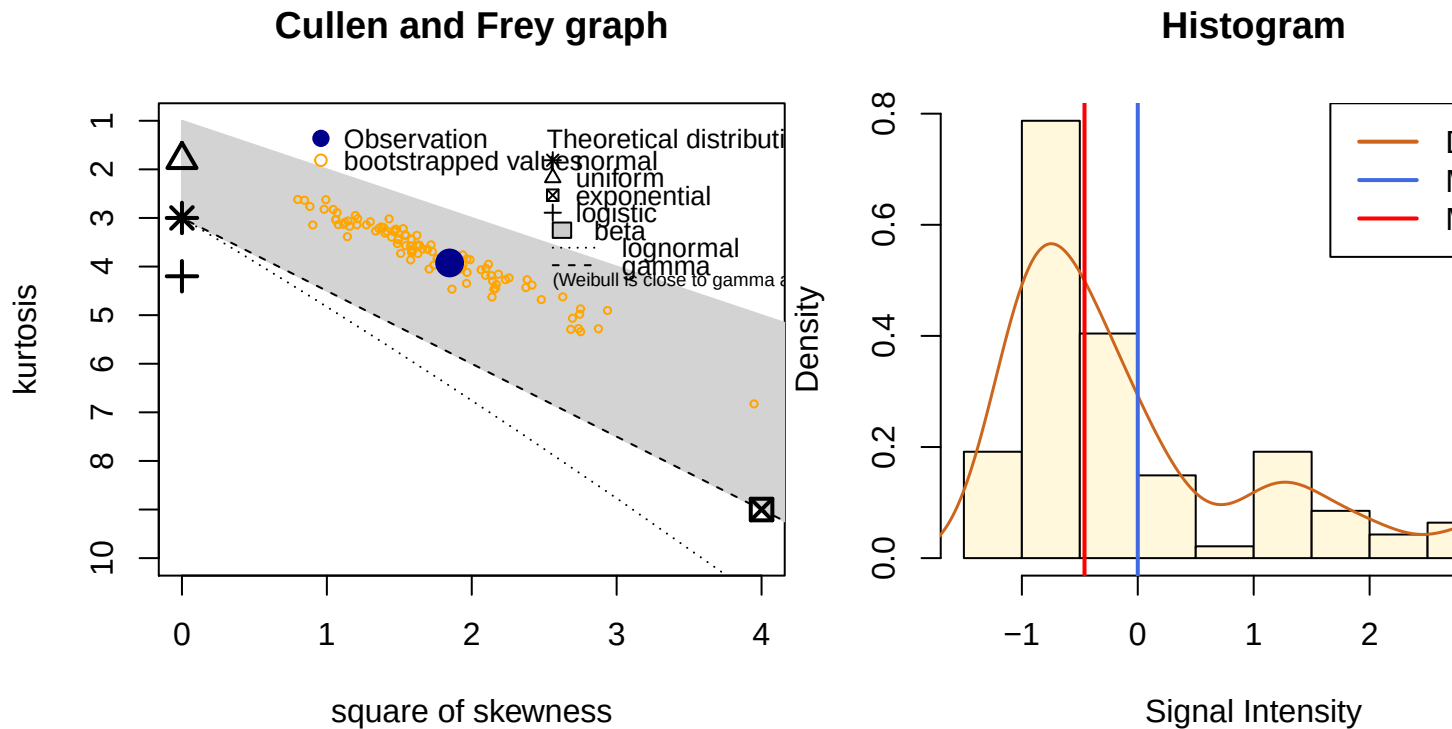
Cullen and Frey graph



Histogram



```
CullenFreyPlot(adjnMFI, filename = "QC.CullenFrey.nMFI")
```



We can see that while both MFI and nMFI data are skewed, nMFI data is closer to log-normal rather than exponential distributions.

Technical Replicates

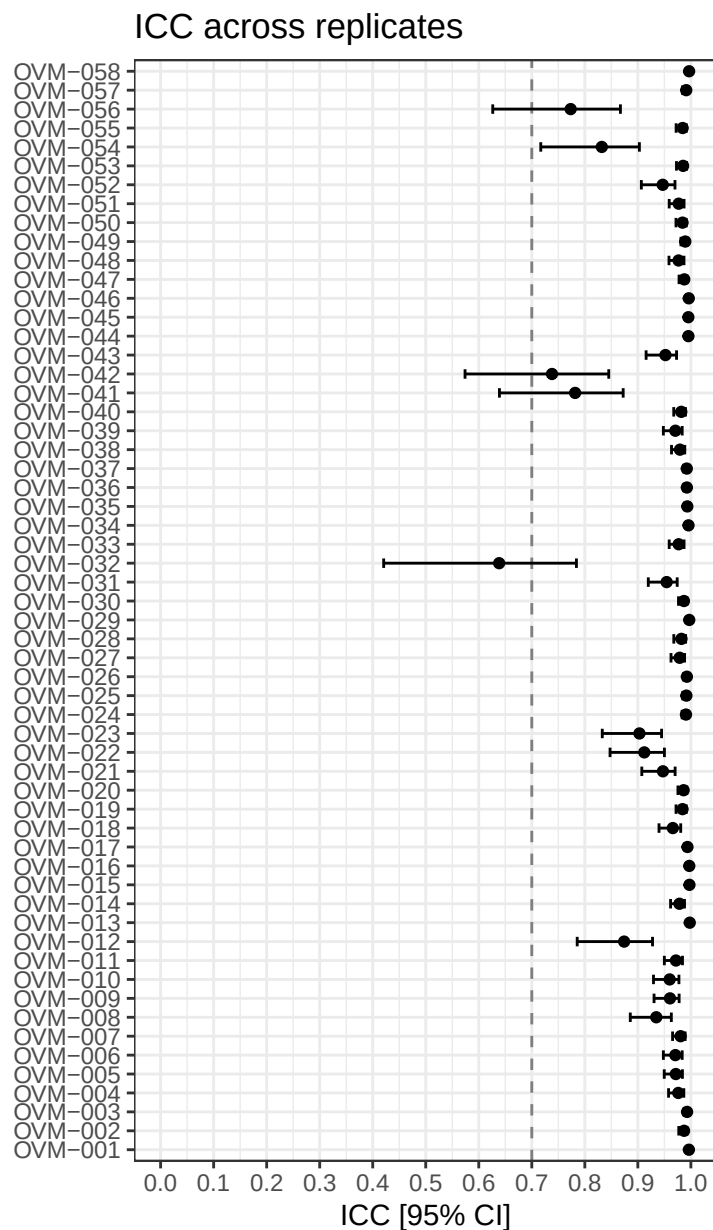
Intraclass Correlation Coefficient (ICC) is used to evaluate agreement among technical replicates for each peptide, where 0 means no agreement, and 1 is a perfect agreement. The function returns the ICC and a 95% confidence interval.

```
icc.db <- getICCbyPeptide(eset,
                          UR=c("PTID", "Plate")) # what is the unit of replication? in this case, it is j
head(icc.db)
```

```
##      Peptide      ICC      LCI      UCI
## OVM-009 OVM-009 0.9606181 0.9305577 0.9778404
## OVM-031 OVM-031 0.9542997 0.9197155 0.9742303
## OVM-011 OVM-011 0.9718032 0.9499818 0.9841880
## OVM-014 OVM-014 0.9784793 0.9617347 0.9879478
## OVM-020 OVM-020 0.9862967 0.9756282 0.9923270
## OVM-023 OVM-023 0.9030485 0.8328782 0.9447205
```

```
ggplot(icc.db, aes(x = Peptide, y = ICC)) +
  geom_hline(yintercept = 0.7, color = "grey50", linetype = 2) +
  geom_point() +
  geom_errorbar(aes(ymax = UCI, ymin = LCI), width = 0.5) +
  scale_y_continuous(limits = c(0, 1), breaks = seq(0, 1, 0.1)) +
```

```
labs(x = "", y = ("ICC [95% CI]"), title = "ICC across replicates") +  
theme_bw() + coord_flip()
```



Coefficient of Variation (CV) is used to estimate variability of the replicates. Generally, for biological assays the desired CV is below 20%.

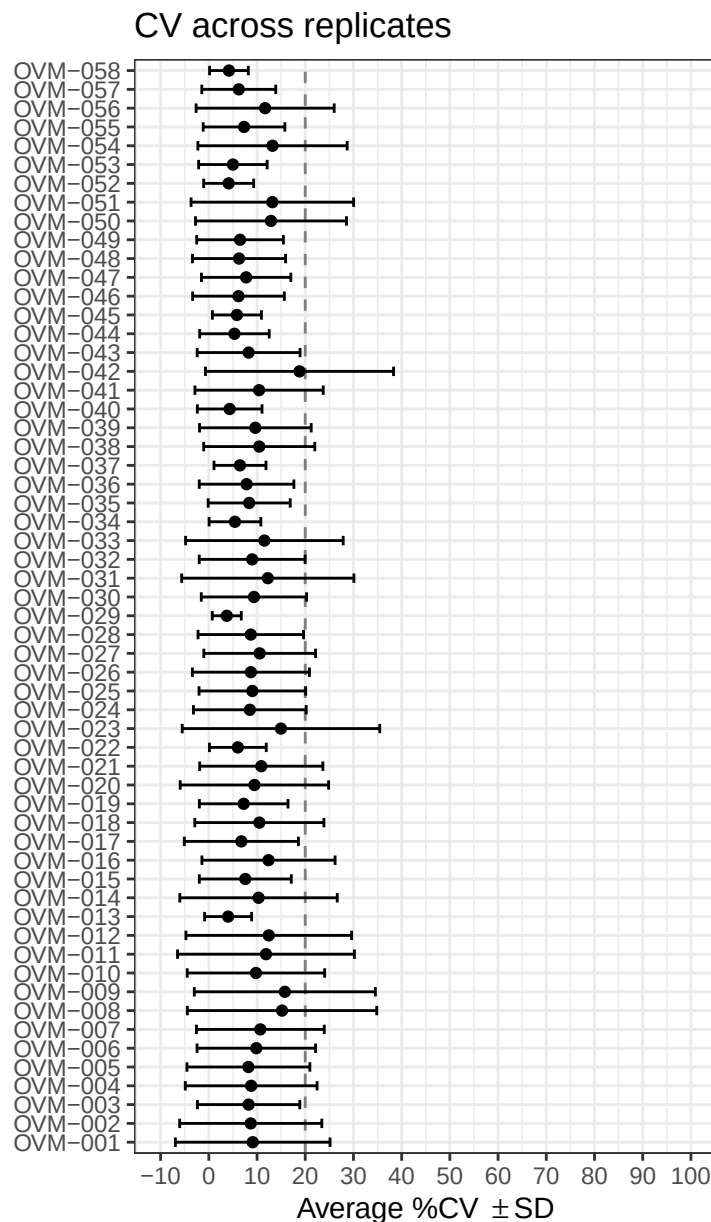
```
cv.db <- getCVbyPeptide.MFI(bbeaEgg,  
                             UR = c("PTID", "Plate")) # what is the unit of replication? in this case, it
```

```
head(cv.db)
```

```
## Peptide mean.cv sd.cv se.cv median.cv  
## 1 OVM-001 9.102478 16.04143 2.291632 1.170530
```

```
## 2 OVM-002 8.692819 14.74045 2.105779 0.000000
## 3 OVM-003 8.253370 10.61832 1.516903 4.876598
## 4 OVM-004 8.788103 13.67184 1.953120 0.000000
## 5 OVM-005 8.209540 12.74839 1.821199 0.000000
## 6 OVM-006 9.849911 12.29611 1.756588 3.142697
```

```
ggplot(cv.db, aes(x = Peptide,y = mean.cv)) +
  geom_hline(yintercept = 20, color = "grey50", linetype = 2) +
  geom_point() +
  geom_errorbar(aes(ymax = mean.cv + sd.cv, ymin = mean.cv - sd.cv),width = 0.5) +
  scale_y_continuous(limits = c(-10,100),breaks = seq(-10,100,10)) +
  labs(x = "", y = expression("Average %CV "+ "%-%"SD"), title = "CV across replicates") +
  theme_bw() + coord_flip()
```



Averaging Technical Replicates

If there are no samples to exclude based on ICC, CV and QC of counts, technical replicates can be averaged for the downstream analyses.

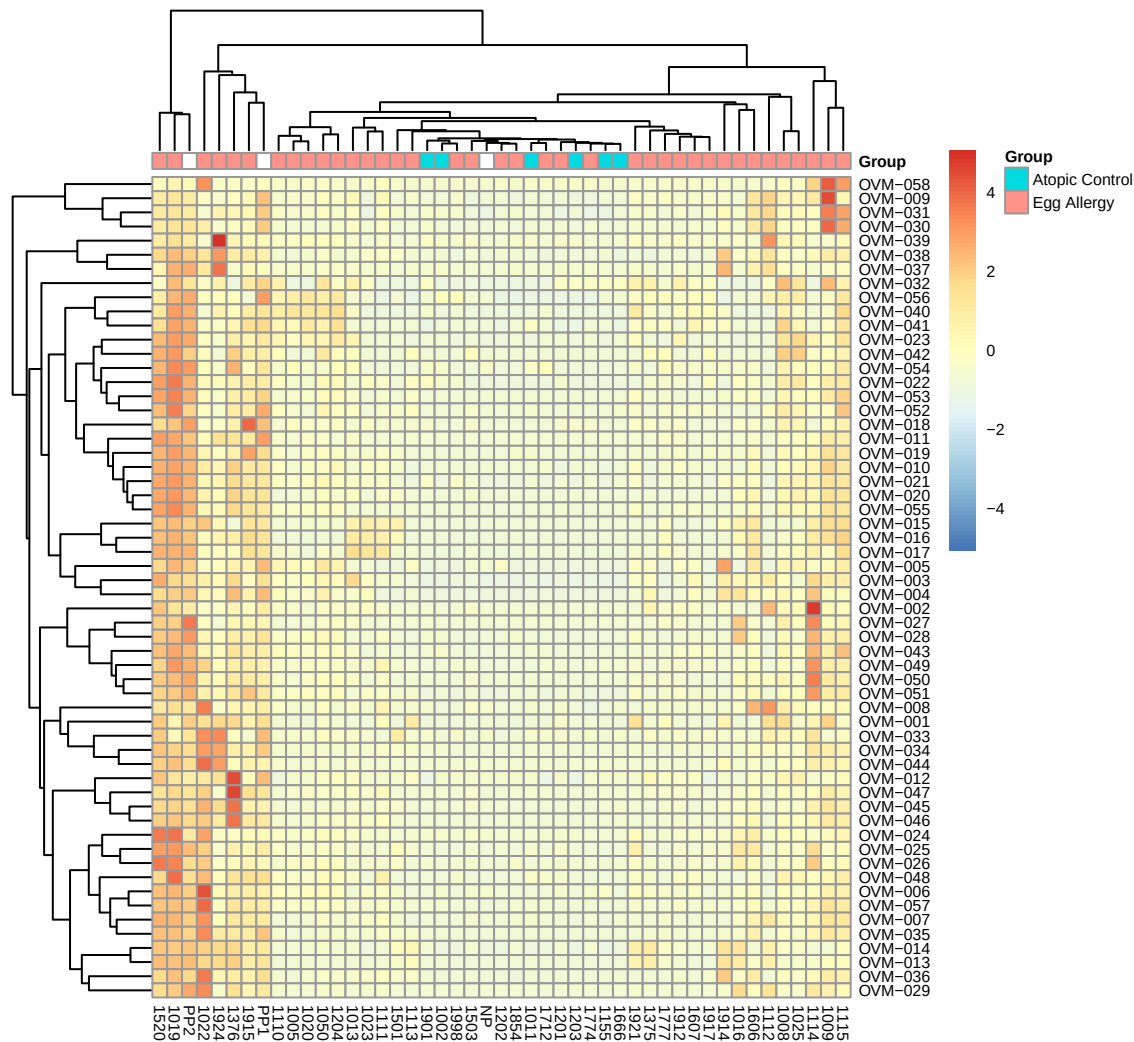
```
eset.avg <- getAveragesByReps(eset, UR = 'PTID')
eset.avg
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 58 features, 47 samples
##   element names: exprs
## protocolData: none
## phenoData
##   rowNames: 1002 1005 ... PP2 (47 total)
##   varLabels: PTID Location ... uf (24 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```

```
exprs(eset.avg)[1:5,1:3]
```

```
##           1002      1005      1008
## OVM-009 0.3962406 0.7924813 2.1961587
## OVM-031 1.5849625 1.5849625 2.8073549
## OVM-011 0.0000000 0.0000000 0.7369656
## OVM-014 0.7924813 1.5849625 2.8073549
## OVM-020 0.0000000 0.0000000 1.4036775
```

```
pheatmap(exprs(eset.avg), scale = "row", fontsize = 6,
          annotation_col = subset(pData(eset.avg), select = Group))
```

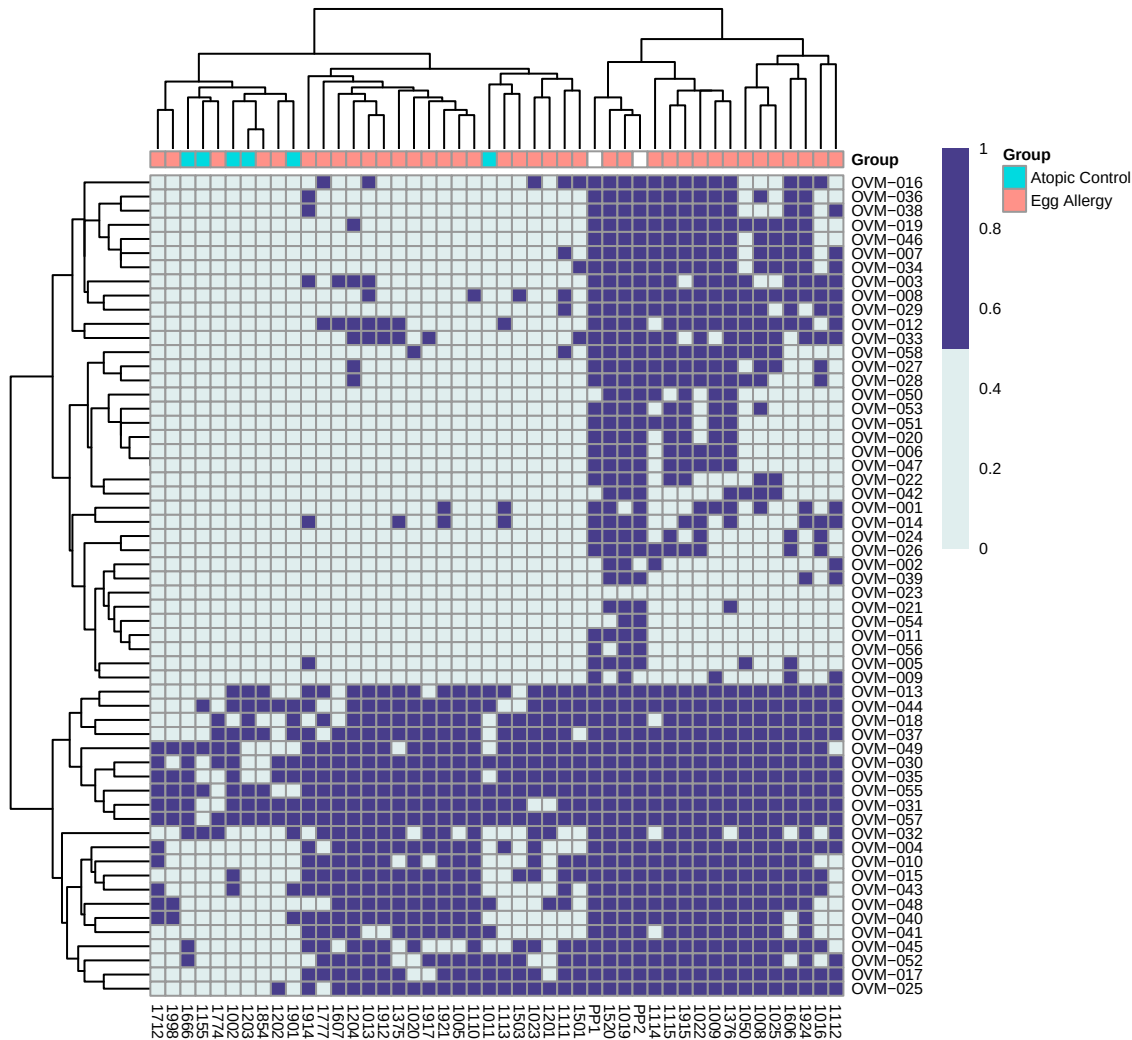


The heatmap shows that few of the patients have higher IgE levels to several ovomucoid peptides. This is consistent with a previous study (Martinez-Botas et al 2013) that used peptide microarrays to show heterogeneity of ses-IgE responses among egg allergic patients. Higher levels or diversity (heatmap below) of ses-IgE could be associated with allergy persistence (Jarvinen et al 2007) , however, a lack of detailed clinical data prevents further conclusions.

Another useful statistic of IgE levels is a binary value of whether the IgE an epitope is “present”, which can also help evaluate ses-IgE diversity. ses-IgE can be defined as “present” if the MFI 2-3 standard deviations above the background (aka “Buffer” wells). Sometimes it might be more important to compare the MFI to negative control samples rather than the buffer. This can be easily achieved by specifying a desired string (e.g., “NegativeControl” or “NP”) in the *buffer.name* argument of *binarizeMFIbySD()* function.

```
binary.db <- binarizeMFIbySD(bbea.object = bbeaEgg,
                             buffer.name = "NP", # a string in the "Sample" column identifying reference
                             plateVar = "Plate", # column indicating batch/plate
                             UR = 'PTID', # unique sample ID
                             nSD = 3) # number of standard deviations above the buffer
sampleNames(binary.db) <- sapply(strsplit(sampleNames(binary.db), "_"),tail, 1)
# can remove background (Buffer) wells
binary.db <- binary.db[, !grepl("Buffer", sampleNames(binary.db))]
```

```
pheatmap(exprs(binary.db), fontsize = 6,
         annotation_col = subset(pData(eset.avg)[sampleNames(binary.db),], select = Group),
         color = c("azure2", "darkslateblue"))
```



The blue color represents epitope-specific IgE antibodies detected in our samples. Several patients (right heatmap cluster) have higher ses-IgE diversity, which could potentially be a marker of egg allergy persistence.

Differential Analysis - Limma Modeling

We are interested in identifying epitope-specific IgE antibodies that are different between egg allergic patients and atopic controls. Atopic controls are patients that are not allergic to egg but have other types of allergies.

For this, we will use a limma framework where a linear model is fit to every peptide.

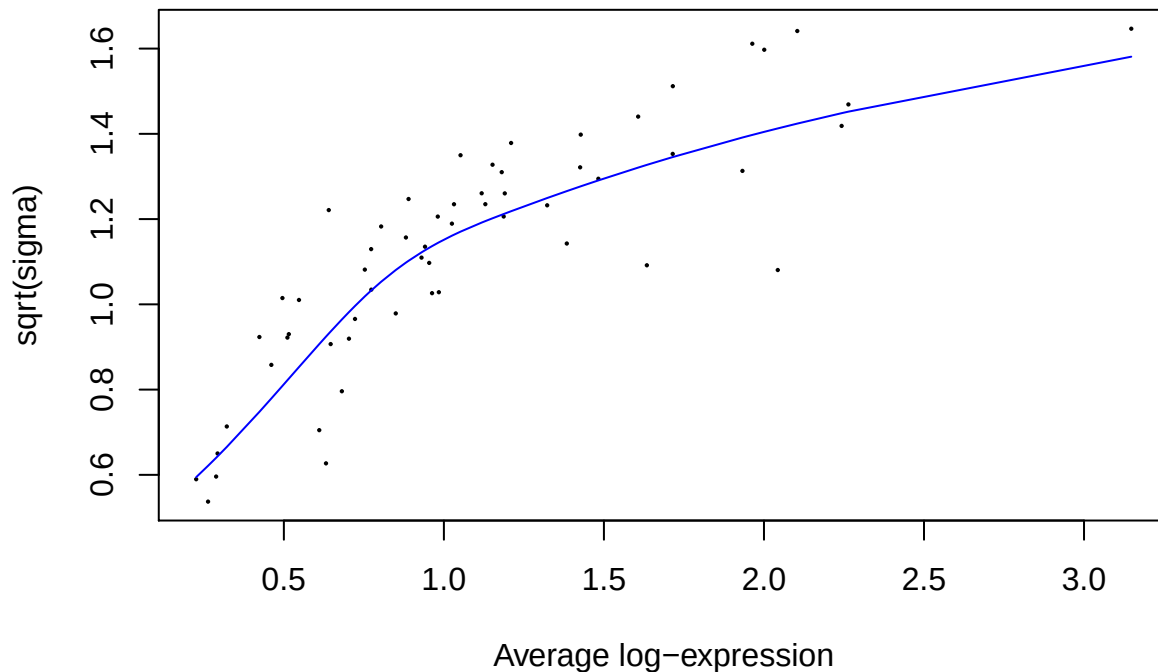
```
# removing control samples
eseti <- eset.avg[, which(!grepl("NP|PP", eset.avg$PTID))]

design <- model.matrix(~ Group, data = pData(eseti))
colnames(design) <- make.names(colnames(design))
```

```

fit <- lmFit(eseti, design)
# means
contr_m <- makeContrasts(Controls = X.Intercept.,
                        EggAllergic = X.Intercept. + GroupEgg.Allergy, levels = design)
# differences
contr_d <- mutate(as.data.frame(contr_m), Allergic_vs_Controls = EggAllergic - Controls)
contr_d <- subset(contr_d, select=Allergic_vs_Controls)
fitm <- eBayes(contrasts.fit(fit, contr_m)) # marginal group means
ebfit <- eBayes(contrasts.fit(fit, contr_d), trend=TRUE) # lgFCH allergic vs controls
par(mfrow = c(1, 1))
plotSA(ebfit) # this plot shows that we have a variance trend, so we should keep trend=TRUE in the eBay

```



```

D <- decideTests(ebfit, method = "separate", adjust.method = "none",
                p.value = 0.05, lfc = log2(1))
t(summary(D)) # number of significant peptides

```

```

##                Down NotSig Up
## Allergic_vs_Controls    0    51  7

```

```

# Table of top epitope-specific IgE antibodies
topTable(ebfit, number=10)

```

```

##          logFC  AveExpr      t    P.Value adj.P.Val      B
## OVM-003 2.5566883 2.2426355 2.866741 0.005948507 0.1757321 -2.416145
## OVM-014 2.9364076 3.1476426 2.503801 0.015419553 0.1757321 -3.052960
## OVM-004 1.1397038 0.9842896 2.328726 0.023742196 0.1757321 -3.337743
## OVM-005 0.9838123 0.8496561 2.230536 0.029993222 0.1757321 -3.490561
## OVM-001 2.4363516 2.1041219 2.155915 0.035675735 0.1757321 -3.603246
## OVM-015 2.2767442 2.0008656 2.119685 0.038760804 0.1757321 -3.656859
## OVM-017 1.8205973 1.6069205 2.059586 0.044393440 0.1757321 -3.744178

```

```
## OVM-016 2.1669335 1.9635633 1.989332 0.051866367 0.1757321 -3.843657
## OVM-051 1.5296773 1.9327391 1.968229 0.054312795 0.1757321 -3.872984
## OVM-040 0.7422013 0.7033638 1.940260 0.057707254 0.1757321 -3.911452
```

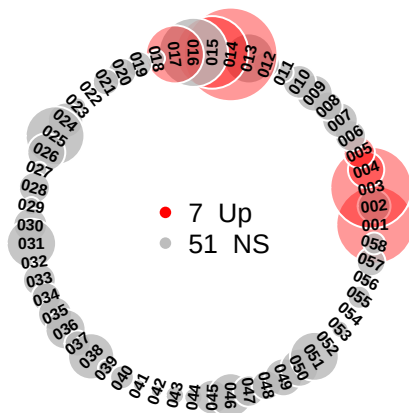
We have identified 7 epitope-specific IgE antibodies that are higher in the allergic group. In this example, we have a small control group (n=6), so the p-values were not corrected for multiple comparison.

Significant epitope-specific IgE can be visualized using the “net circle” plot.

Area of the circle is proportional to the absolute value of the difference (Allergic vs Controls). Color of the circle indicates the direction (Higher/Up=red, Lower/Down=blue, Non-Significant (NS)=grey) and significance. User needs to make sure that the Annotation file has a column named “labelName”, as it will be used to print names of the epitopes.

```
Run_doNetCirclePlot(ebfit, D, AnnotEgg, fname = "lmFit.EAvsAC.")
```

IgFCH @ Allergic_vs_Controls



Protein Topology Plot

We can now use our experimental data to overlay it on the amino acid sequence of the protein. Ovomucoid doesn't have a complete 3D crystal structure, so a “topology” plot can be helpful in identifying specific areas on a protein that are recognized by patients' IgE.

First, we need to create a protein layout. We will use *drawProteins* package to download information about the protein sequence from the UniProt database. The meta data includes amino acid (aa) positions of SIGNAL and CHAIN sequences, as well as information about reactive sites, glycosylation, disulfate bridges.

```
json <- drawProteins::get_features("P01005")
```

```
## [1] "Download has worked"
```

```
ovm_p01005_meta <- drawProteins::feature_to_dataframe(json)
# Ovomucoid has a signal peptide sequence, that was not part of the peptides used in the assay.
ovm_signalEnd <- ovm_p01005_meta$end[which(ovm_p01005_meta$type == "SIGNAL")]
ovm_p01005_meta <- mutate(ovm_p01005_meta,
                           beginMinusSignal = begin - ovm_signalEnd,
```

```

                                endMinusSignal = end - ovm_signalEnd)
# complete aa sequence
ovm_p01005_aaseq <- json[[1]]$sequence
# removing signal peptide
ovm_p01005_aaseqCHAIN <- substr(ovm_p01005_aaseq, ovm_signalEnd + 1, (nchar(ovm_p01005_aaseq)))

```

getTopologyPlotDB() function will construct a dataframe that can be used for plotting. The *getEnzymeCuts()* function is based on the R package cleaver and can add sites of enzymatic cuts (by trypsin, pepsin, and chymotrypsin).

```

ovm_plotDB <- getTopologyPlotDB(ovm_p01005_aaseq, # a character string with amino acid sequence
                                ymax = 20, # number of amino acids in one column
                                offset = 3, # the first few amino acids to be below the rest of the plot
                                meta = TRUE, # do we have the metadata
                                metadb = ovm_p01005_meta) # metadata dataframe

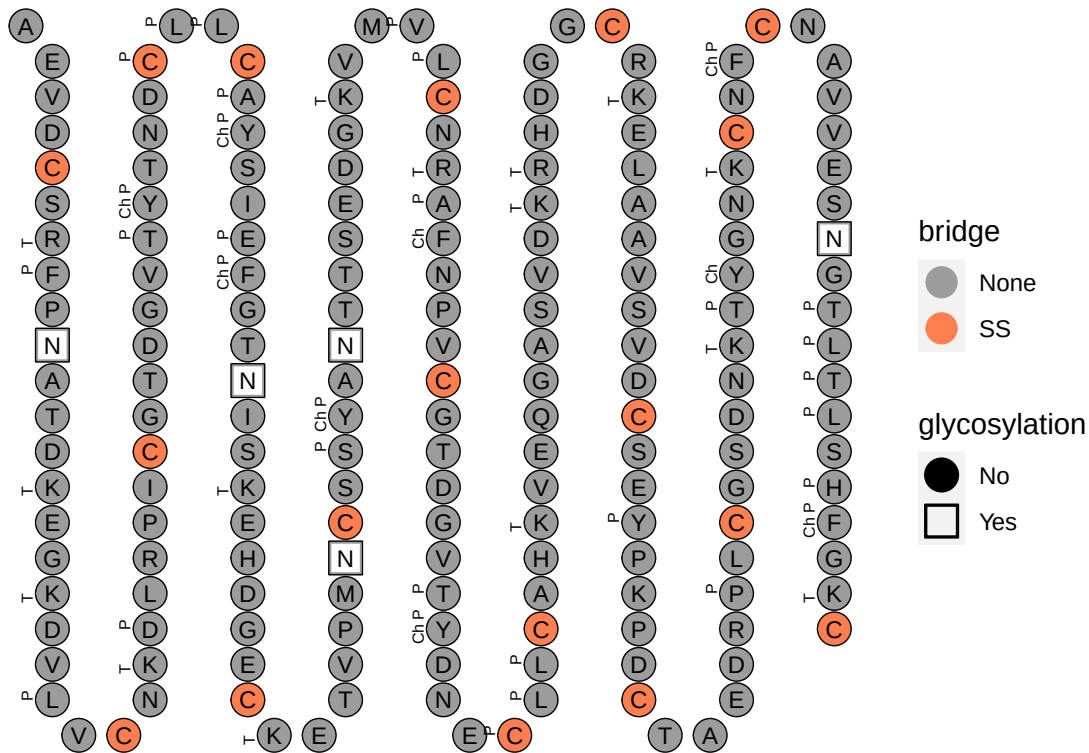
ovm_plotDB <- getEnzymeCuts(ovm_p01005_aaseq, # a character string with amino acid sequence
                             topo = TRUE, # do we have a dataframe generated with getTopologyPlotDB()
                             topodb = ovm_plotDB, # name of the getTopologyPlotDB() dataframe
                             rmsignal = TRUE, # remove the signal peptide chain?
                             signalend = 24) # at what residue does the signal chain end

ovm_plotDB <- mutate(ovm_plotDB,
                     EnzymeRmSignal = ifelse(is.na(EnzymeRmSignal), "", EnzymeRmSignal))

makeTopologyPlotBase(subset(ovm_plotDB, type != "Signal")) +
  geom_point(size = 5.5, color='black', aes(shape = glycosylation)) +
  geom_point(size = 5, aes(color = disulf, shape = glycosylation)) +
  scale_shape_manual(values = c(19, 0, 11), labels = c("No", "Yes")) +
  scale_color_manual("bridge", values = c("grey61", "coral"), labels = c("None", "SS")) +
  geom_text(aes(label = AA), size = 3) +
  labs(title = paste("Ovomucoid (P01005)")) +
  geom_text(data = subset(ovm_plotDB, EnzymeRmSignal != ""), aes(label = EnzymeRmSignal),
            size = 2, vjust = -1.7, angle = 90)

```

Ovomucoid (P01005)



The plot shows that several Asparagine (N) residues are glycosylated; the Cysteine residues colored in orange form disulfide bridges; and there are many potential enzymatic cleavage sites (P=Pepsin; T=Trypsin, Ch=Chymotrypsin).

Now we will overlay this topology plot with our experimental data to see if any residues stand out.

```
# Getting sequences of the overlapping peptides (overlap by 12 aa, offset 3)
peptDB_OVM <- do.call("rbind", sapply(seq(1, nchar(ovm_p01005_aaseqCHAIN), 3),
  function(i){
    substr(ovm_p01005_aaseqCHAIN, i, 14 + i)
  }, simplify = FALSE))
peptDB_OVM <- subset(mutate(as.data.frame(peptDB_OVM),
  Peptide = paste0("OVM-", str_pad(1:nrow(peptDB_OVM), 3, pad = "0")),
  Peptide.Number = 1 : nrow(peptDB_OVM)),
  nchar(as.character(V1)) == 15)
colnames(peptDB_OVM)[1] <- "Sequence"

# generate topology dataframe by using the means of the egg allergic patients
dbt <- getTopologynMFI(eset = eset.avg[, which(eset.avg$Group == "Egg Allergy")],
  peptideDB = peptDB_OVM,
  topoDB = ovm_plotDB,
  endSignal = ovm_signalEnd)
# assigning percentiles to the average level for each peptide
dbt$avgp <- cut(dbt$avg,
  breaks = quantile(dbt$avg, probs=seq(0, 1, by = 0.25), na.rm = TRUE),
  include.lowest = TRUE, labels = c("<25%", "25-50%", "50-75%", ">75%"))

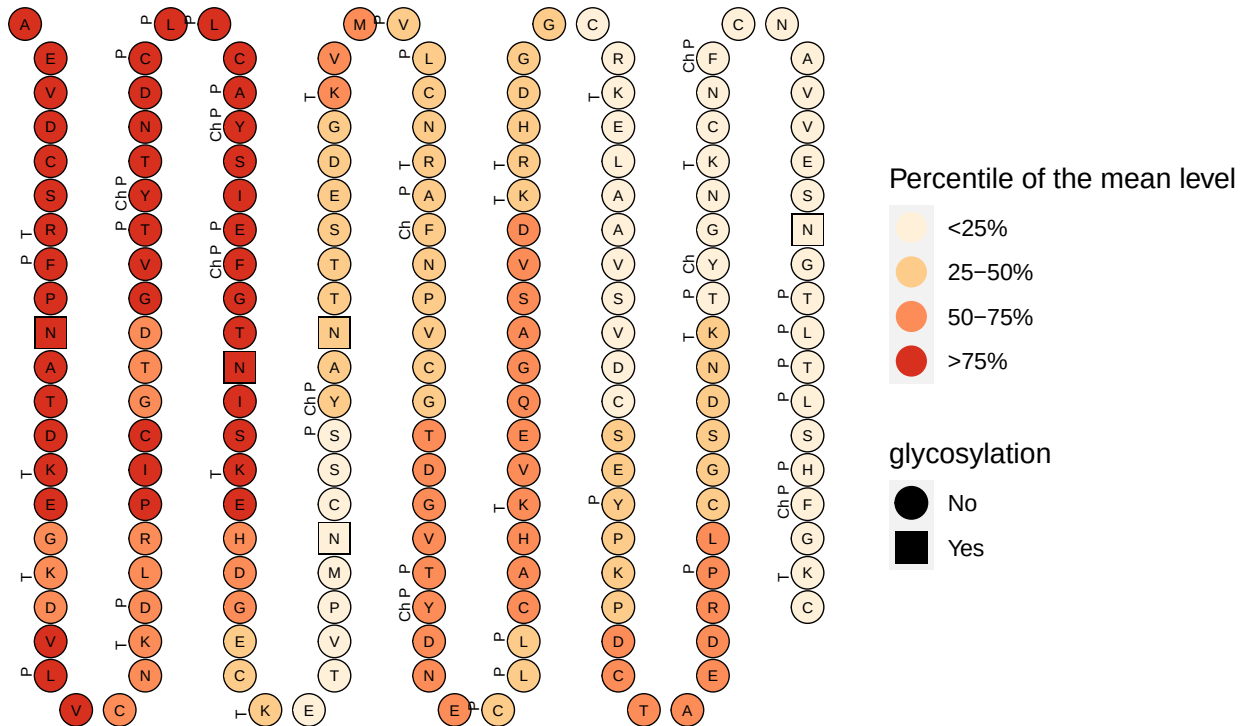
makeTopologyPlotBase(subset(dbt, type != "Signal")) +
```

```

geom_point(size = 5.5, color = 'black', aes(shape = glycosylation)) +
geom_point(size = 5, aes(color = avgp, shape = glycosylation)) +
scale_color_brewer("Percentile of the mean level", palette = "OrRd") +
scale_shape_manual(values = c(19, 15), labels = c("No", "Yes")) +
geom_text(aes(label = AA), size = 2) +
geom_text(data = subset(dbt, EnzymeRmSignal != ""), aes(label = EnzymeRmSignal),
          size = 2, vjust = -1.7, angle = 90) +
labs(title = "Mean nMFI's percentile for each amino acid")

```

Mean nMFI's percentile for each amino acid



The topology plot shows that most IgE antibodies of our egg allergic patients were directed to the several groups of amino acids in the N-terminal of the ovomucoid protein. This way we have identified IgE directed at sequential peptides associated with egg allergy, constituting immunogenic IgE epitopes previously identified by several other groups that used different epitope mapping technologies (Cooke et al (1997), Shreffler et al (2005), Jarvinen et al (2007)).

bbeaR - Milk Allergy Example

Maria Suprun, Randall J. Ellis, Hugh A. Sampson, Mayte Suárez-Fariñas

10/08/2020

Contents

About	1
Installation	2
Raw Data Import	2
Quality Control of the Raw Data	6
Data Normalization	8
QC of Normalized Data	9
Batch Effect	10
Distributions	14
Technical Replicates	16
Averaging Technical Replicates	18
Differential Analysis - Limma Modeling	20

About

The **Bead-Based Epitope Assay** (*BBEA*) can be used to quantify the amount of epitope- or peptide-specific antibodies (e.g., IgE) in plasma or serum samples. A detailed assay description is outlined in this publication

In this tutorial, we will analyze a dataset of the sequential epitope-specific (*ses-*) *IgE* profiles in patients allergic to milk that were treated with milk oral immunotherapy (OIT) (Suárez-Fariñas et al (2018)). Subjects (7-35 years of age) with IgE-mediated cow's milk allergy were randomized 1:1 to receive omalizumab or placebo in a double-blind, placebo-controlled trial to evaluate whether the addition of omalizumab to milk OIT would reduce treatment-related adverse reactions and increase the frequency of patients achieving “sustained unresponsiveness” to milk. Detailed description and primary outcomes of this trial were published in Wood et al (2015).

The levels of 66 *ses*-IgE antibodies were measured using *BBEA* in plasma of 47 enrolled subjects at baseline and at 32-months of treatment with milk OIT plus placebo or omalizumab.

We will start by installing *bbeaR* and reading in the *BBEA*'s raw data (alternatively, one can load an R dataset that comes with this package). We will look at several quality control (QC) measures and normalize the data. Then we will demonstrate an approach to identify ses-IgE antibodies that is different between children treated with Placebo or Omalizumab as adjuvants for milk Oral Immunotherapy (mOIT), using limma modeling framework.

Installation

bbeaR is an R package and is installed using a standard github command:

```
library(devtools)
install_github('msuprun/bbeaR')
```

Loading additional packages that will be used in the analyses.

```
library(bbeaR)
library(plyr)
library(stringr)
library(ggplot2)
library(gridExtra)
library(pheatmap)
library(RColorBrewer)
library(limma)
```

Raw Data Import

47 patients were assayed before and after treatment, for a total of 94 samples, each ran in triplicate on four 96-well plates. The runs additionally included three background wells, without any sample (aka “Buffer”, for background signal quantification).

The original .csv files from the Luminex-200 assay, generated with the xPONENT® software, can be downloaded here. The .csv sheet consists of several sections, of which “Median”, “Count”, and “NetMFI” hold the assay’s readout values for each epitope-specific antibody (*j*) in columns and each sample (*i*) in rows:

Data Type:	Median											
Location	Sample	Analyte 5	Analyte 7	Analyte 8	Analyte 10	Analyte 12	Analyte 14	Analyte 15	Analyte 16	Analyte 18	Analyte 19	Analyte 21
1(1,A1)	Unknown1	6	72.5	10	4	18	35	22	2	1	2	1
2(1,A2)	Unknown1	7	75	10	4	19	33	21	2	1	2	1
3(1,A3)	Unknown1	6	72	10	4	19	27	21	2	1	2	1
4(1,A4)	Unknown9	49.5	246.5	428	277	12	227	439	295.5	6	240	48
5(1,A5)	Unknown9	55	238	411.5	265	12	211	417	292	6	226.5	45
6(1,A6)	Unknown9	50	241	440	263	11	213.5	412	294.5	6	236	44
7(1,A7)	Unknown17	32	79	127.5	67.5	26	29	212	58	64	233	3
8(1,A8)	Unknown17	34	77	116	62	24	29	191	55	60	213	2
9(1,A9)	Unknown17	36	79	113.5	62	24	31	203	56	65	205	3
10(1,A10)	Unknown25	3	8	5	8	2	3	10	4	1	28	18
11(1,A11)	Unknown25	3	8	6	8	2	3	8	3	1	28	17.5
12(1,A12)	Unknown25	3	7	5	6	2	3	8	3	1	25	17
13(1,B1)	Unknown2	2	23	3	1	2	8.5	6	1	1	1	1
14(1,B2)	Unknown2	3	27	4	2	2	10	7	1	0	1	0
15(1,B3)	Unknown2	2	25.5	3	1	2	9	7	1	1	1	1
16(1,B4)	Unknown10	16	57.5	156	97	8	60	187	106	2	74	15
17(1,B5)	Unknown10	15	56	167	105	9	57	188	101	2	73	16
18(1,B6)	Unknown10	16	58	161	98	8	59	186	108	2	75	15.5
19(1,B7)	Unknown18	14	44	42	27	19	19	104.5	18	28	83	3
20(1,B8)	Unknown18	12	45	42	30	19	19	101	17	27	85	3
21(1,B9)	Unknown18	14	45	40	27	20	17	119	18	29	82.5	3

First, we need to create a plate layout using the `create.plate.db()` function. This layout will be used in the import and some of the plotting functions. The only input to this function is the direction of the plate read (horizontal or vertical).

```
l <- create.plate.db(direction = "horizontal")
plate.design.db <- l$plate.design.db
plate.design <- l$plate.design
plate.design
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## [1,] "A1" "A2" "A3" "A4" "A5" "A6" "A7" "A8" "A9" "A10" "A11" "A12"
## [2,] "B1" "B2" "B3" "B4" "B5" "B6" "B7" "B8" "B9" "B10" "B11" "B12"
## [3,] "C1" "C2" "C3" "C4" "C5" "C6" "C7" "C8" "C9" "C10" "C11" "C12"
## [4,] "D1" "D2" "D3" "D4" "D5" "D6" "D7" "D8" "D9" "D10" "D11" "D12"
## [5,] "E1" "E2" "E3" "E4" "E5" "E6" "E7" "E8" "E9" "E10" "E11" "E12"
## [6,] "F1" "F2" "F3" "F4" "F5" "F6" "F7" "F8" "F9" "F10" "F11" "F12"
## [7,] "G1" "G2" "G3" "G4" "G5" "G6" "G7" "G8" "G9" "G10" "G11" "G12"
## [8,] "H1" "H2" "H3" "H4" "H5" "H6" "H7" "H8" "H9" "H10" "H11" "H12"
```

Then read all four `.csv` files.

Note: a separate tutorial (Egg Allergy Example) has an example of importing and processing a single plate.

```
file_names <- dir(path = "./", pattern=".csv",
                  all.files = FALSE, full.names = TRUE)
bbea <- bbea.read.csv.all(file_names[grepl("MOIT_IgE",file_names)])
```

```
## [1] "Reading file: MOIT_IgEplate_02.csv"
## [1] "Reading file: MOIT_IgEplate_03.csv"
## [1] "Reading file: MOIT_IgEplate_04.csv"
## [1] "Reading file: MOIT_IgEplate01.csv"
```

The `bbea` list object contains several elements, extracted from the assay's output.

```
names(bbea)
```

```
## [1] "Median"      "NetMFI"      "Count"      "pData"      "AssayInfo"
```

The **Median**, **NetMFI**, and **Count** are matrices with rows as Analytes (epitopes) and columns as Samples. Note: this is changed from the original data layout, where samples were in rows and epitopes in columns, to make the dataset usable with common analytical tools in R.

The **Median** are the Median Fluorescence Intensities (MFIs) and the **NetMFI** are the Medians normalized to background. In our example, **Median** and **NetMFI** have exactly same values, since normalization was not selected during the assay run. This post has more details about the Luminex outputs.

```
bbea$Median[1:5, 1:2]
```

```
##      Plate2.1.1.A1. Plate2.9.1.A2.
## Analyte 10          5              4
## Analyte 12          1              1
## Analyte 14          2              2
## Analyte 15          5              4
## Analyte 16          1              0
```

The **Count** are the numbers of beads counted per analyte, and is important quality control measure.

```
bbea$Count[1:5, 1:2]
```

```
##           Plate2.1.1.A1. Plate2.9.1.A2.
## Analyte 10           161           149
## Analyte 12           147           140
## Analyte 14           191           165
## Analyte 15           175           143
## Analyte 16           194           144
```

The **AssayInfo** saves parameters of the assay.

```
bbea$AssayInfo[1:15, 1:2]
```

```
##           V1
## 1           Program
## 2           Build
## 3           Date
## 4
## 5           SN
## 6           Batch
## 7           Version
## 8           Operator
## 9           ComputerName
## 10          Country Code
## 11          ProtocolName
## 12          ProtocolVersion
## 13          ProtocolDescription
## 14 ProtocolDevelopingCompany
## 15          SampleVolume
##
##           V2
## 1           xPONENT
## 2           3.1.971.0
## 3           7/22/15
## 4
## 5           LX10010124401
## 6           Plate2
## 7           1
## 8
## 9           XPONENT31-PC
## 10          409
## 11          Milk OIT IgE Plate2 07.22.15
## 12          1
## 13 Run by Gustavo Gimenez, Peiatric Allergy and Immunology Department
## 14
## 15          90 uL
```

Finally, *bbea.read.csv()* creates a **phenotype (p)Data** that has some basic information about the samples and the assay run.

```
colnames(bbea$pData)
```

```
## [1] "Location"      "Sample"        "filename"
## [4] "File"          "Plate"         "SampleNumber"
## [7] "Well.Number"   "Well.Letter"   "Well_coord"
## [10] "print.plate.order" "letters_numeric" "Plate.Date"
## [13] "Plate.Time"     "Plate.TimeHR"   "CountSum"
## [16] "CountMean"      "CountMin"       "CountMax"
```

```
bbea$pData[1:2, 1:2]
```

```
##           Location Sample
## Plate2.1.1.A1.  1(1,A1) Unknown1
## Plate2.9.1.A2.  9(1,A2) Unknown1
```

We can now add external information about the samples and analytes.

Load already imported data that includes **phenotype** (clinical) data *PD* and an annotation file *Annot*. Note: this will also load a *bbea* list object (generated in the previous steps).

```
data(Milk)
```

The annotation **Annot** dataset contains the mapping of Luminex beads (Analytes) to the peptides/epitopes.

```
dim(Annot)
```

```
## [1] 66  6
```

```
head(Annot)
```

```
##           Analyte      Peptide Protein labelName Bead PeptideName
## alphas1-p01 Analyte 5 alphas1-p01 alphas1    a1.p01    5 alphas1-p01
## alphas1-p02 Analyte 58 alphas1-p02 alphas1    a1.p02   58 alphas1-p02
## alphas1-p03 Analyte 7 alphas1-p03 alphas1    a1.p03    7 alphas1-p03
## alphas1-p04 Analyte 8 alphas1-p04 alphas1    a1.p04    8 alphas1-p04
## alphas1-p05 Analyte 60 alphas1-p05 alphas1    a1.p05   60 alphas1-p05
## alphas1-p06 Analyte 10 alphas1-p06 alphas1    a1.p06   10 alphas1-p06
```

Changing the *bbea* object to have milk epitope names instead of Luminex analyte numbers.

```
bbea <- bbea.changeAnnotation(bbea.obj = bbea,
                             annotation = Annot,
                             newNameCol = "Peptide",
                             AnalyteCol = "Analyte")
bbea$Median[1:5, 1:2]
```

```
##           Plate1.1.1.A1. Plate1.2.1.A2.
## alphas1-p06             4             4
## alphas1-p08            18            19
## alphas1-p09            35            33
## alphas1-p13            22            21
## alphas1-p17             2             2
```

The **PD** dataset has clinical information about our samples.

```
dim(PD)
```

```
## [1] 285 5
```

```
head(PD)
```

```
##  SUBJECT.ID    Visit Treatment Plate Location
## 1    MILK01 Baseline   Placebo Plate1 34(1,C10)
## 2    MILK01 Baseline   Placebo Plate1 35(1,C11)
## 3    MILK01 Baseline   Placebo Plate1 36(1,C12)
## 4    MILK01 Month32    Placebo Plate1 46(1,D10)
## 5    MILK01 Month32    Placebo Plate1 47(1,D11)
## 6    MILK01 Month32    Placebo Plate1 48(1,D12)
```

We will clean up the pData and then merge it with PD. This step is not necessary, but will be useful when we do statistical modelling.

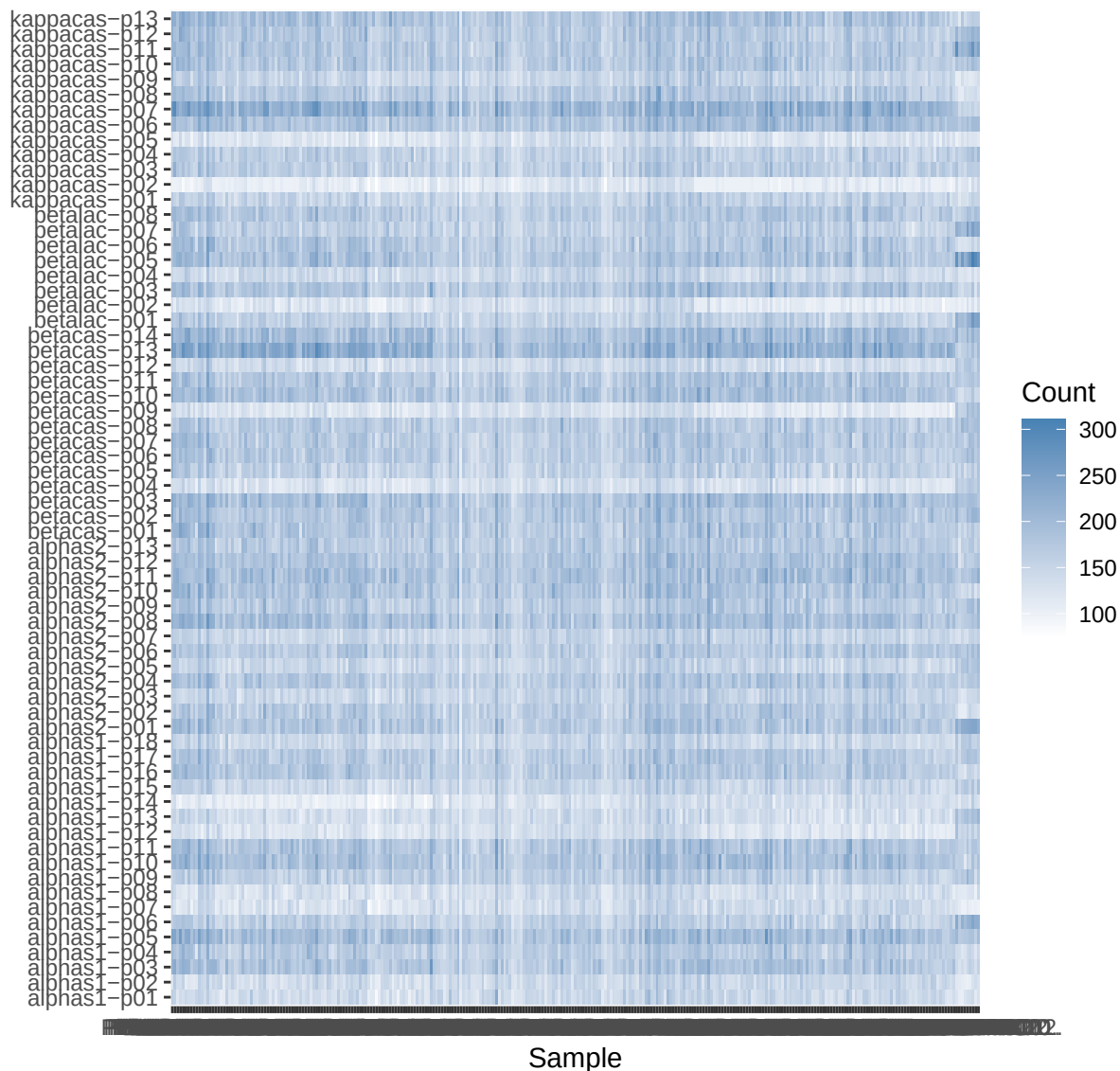
```
bbea$pData$Rows <- rownames(bbea$pData)
bbea$pData <- mutate(merge(bbea$pData,PD,by=c("Location","Plate"),all.x=T),
                      Sample=mapvalues(SUBJECT.ID,NA,"Buffer"))
rownames(bbea$pData) <- bbea$pData$Rows
```

Quality Control of the Raw Data

We want to make sure that there are no missing samples or analytes. This would be reflected by the very low counts (<25).

The heatmap shows that all samples and analytes were included.

```
bbea.QC.heatmap.counts(bbea,
                        getlog2=FALSE,
                        filename="QC.CountsHeatmap.pdf",
                        plateVar="Plate", ann=NULL)
```

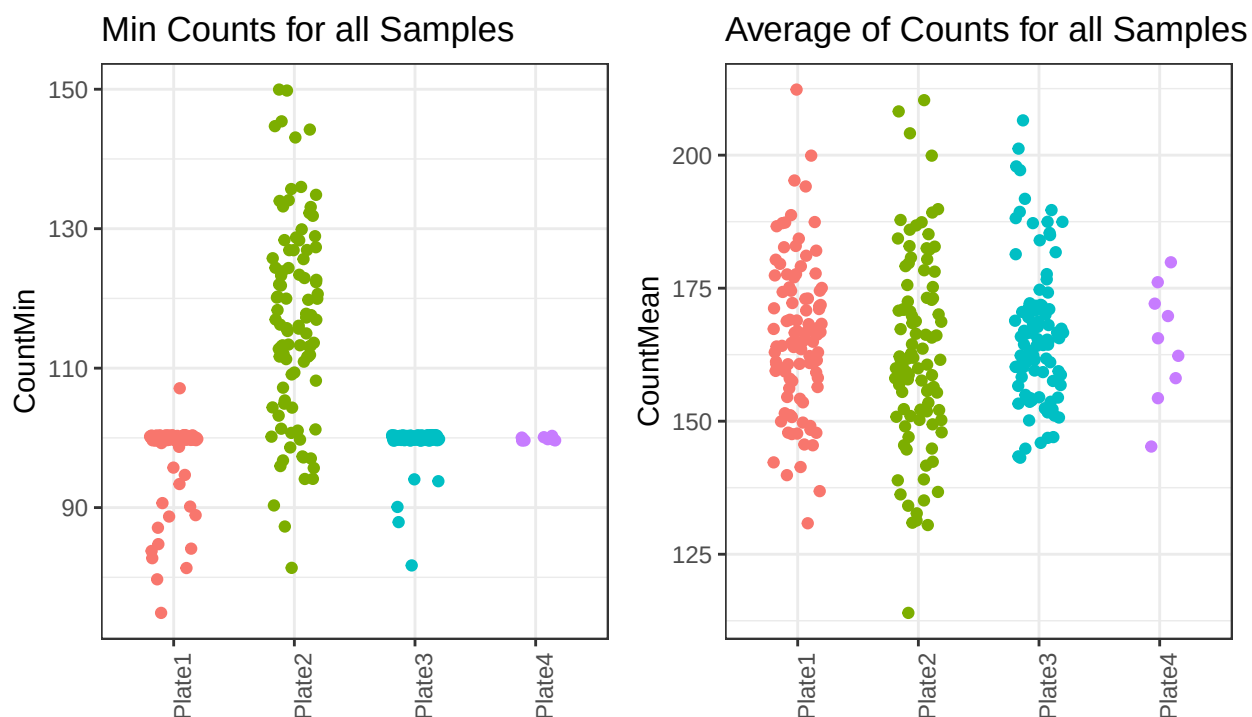


All counts seem to be high and we don't see any specific sample or epitope to be missing. However, when having multiple plates, it might be more useful to have a counts heatmap for each plate separately. This can be achieved by running `bbea.QC.heatmap.counts.byPlate()` function.

```
bbea.QC.heatmap.counts.byPlate(bbea, getlog2=FALSE,
                               filename="QC.CountsHeatmap",
                               plateVar="Plate", ann=NULL, he=7, wi=9)
```

Now we look at the overall distribution of counts: the minimum count is ~75, which is pretty good.

```
p<-bbea.QC.Samples(bbea,
                   filename = "QC.",
                   plateVar = 'Plate',
                   gt = 25)
grid.arrange(p$pmin, p$pavg, nrow=1)
```



Our counts look good, so we don't need to exclude any samples. However, if this were not the case, samples with low counts can be removed using the `bbea.subset()` function, only keeping samples with average counts > 25 .

```
bbea.sub <- bbea.subset(bbea, statement = (bbea$pData$CountMean > 25))
```

Data Normalization

We convert the **Median** to normalized **nMFI** by taking the $\log_2$ of values and subtracting the average of the background wells. Note: the *Sample* column of the **pData** should include a "Buffer" string in the wells dedicated for the background.

```
bbeaN <- MFI2nMFI(bbea,
  offset = 0.5, # a constant to add to avoid taking a log of 0
  rmNeg = TRUE) # if a value of a sample is below background, assign "0"
```

```
## [1] "Plate1"
## [1] "Plate2"
## [1] "Plate3"
## [1] "Plate4"
```

```
names(bbeaN)
```

```
## [1] "nMFI"      "NetMFI"    "Count"     "pData"     "AssayInfo"
```

```
bbeaN$nMFI[1:5, 1:2]
```

```
##           Plate1.1.1.A1. Plate1.2.1.A2.
## alphas1-p06      2.641604      2.641604
## alphas1-p08      4.681133      4.757081
## alphas1-p09      5.621426      5.537768
## alphas1-p13      4.435211      4.369623
## alphas1-p17      1.793607      1.793607
```

R object of class ExpressionSet *eset* is convenient for a high-throughput data analysis.
bbea object can be converted to *eset*:

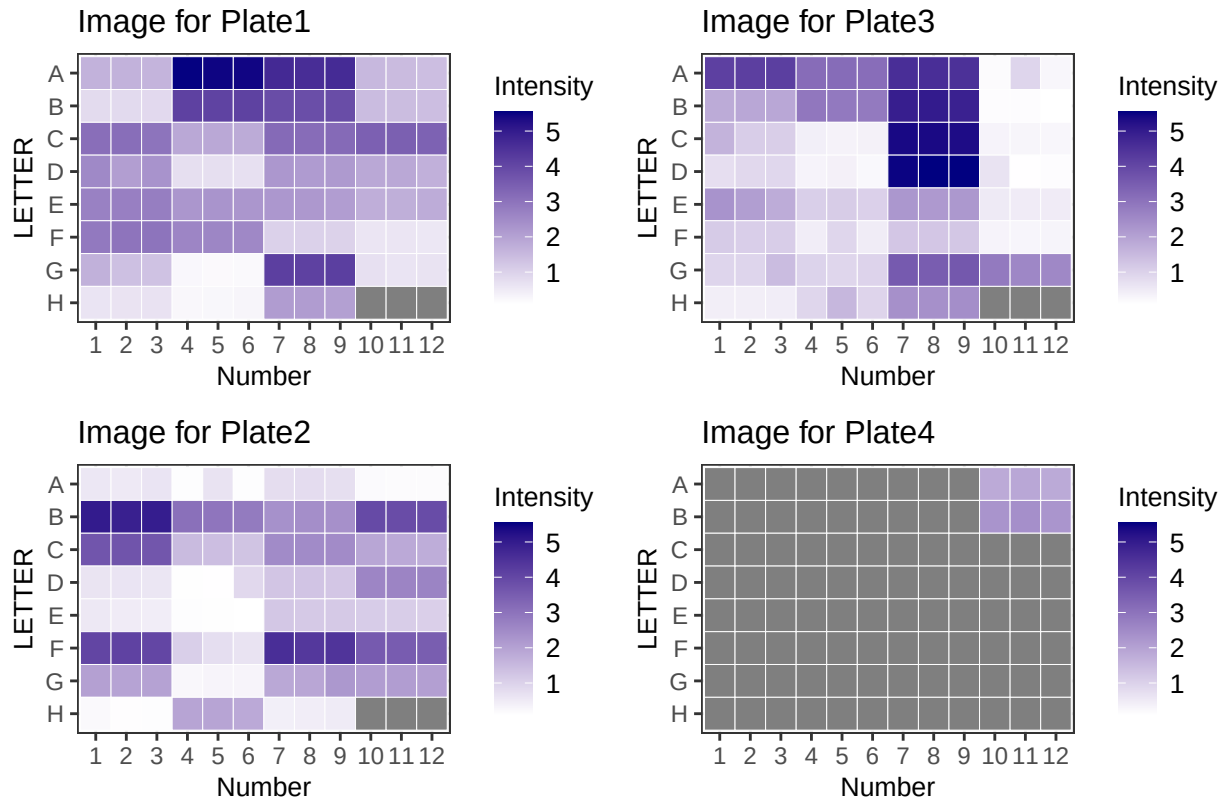
```
eset <- nMFI2Eset(nMFI.object = bbeaN)
eset
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 66 features, 285 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: Plate1.1.1.A1. Plate1.2.1.A2. ... Plate4.24.1.B12. (285
##     total)
##   varLabels: Location Plate ... Treatment (22 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```

QC of Normalized Data

We can print the layouts of all experimental plates, to quickly do a visual inspection of the samples.

```
Image.Plate(bbeaN)
```



In this figure, we can clearly see triplicates and several wells (grey) that were used for the background calculation. We can also notice that last plate (plate #4) had only two samples.

Batch Effect

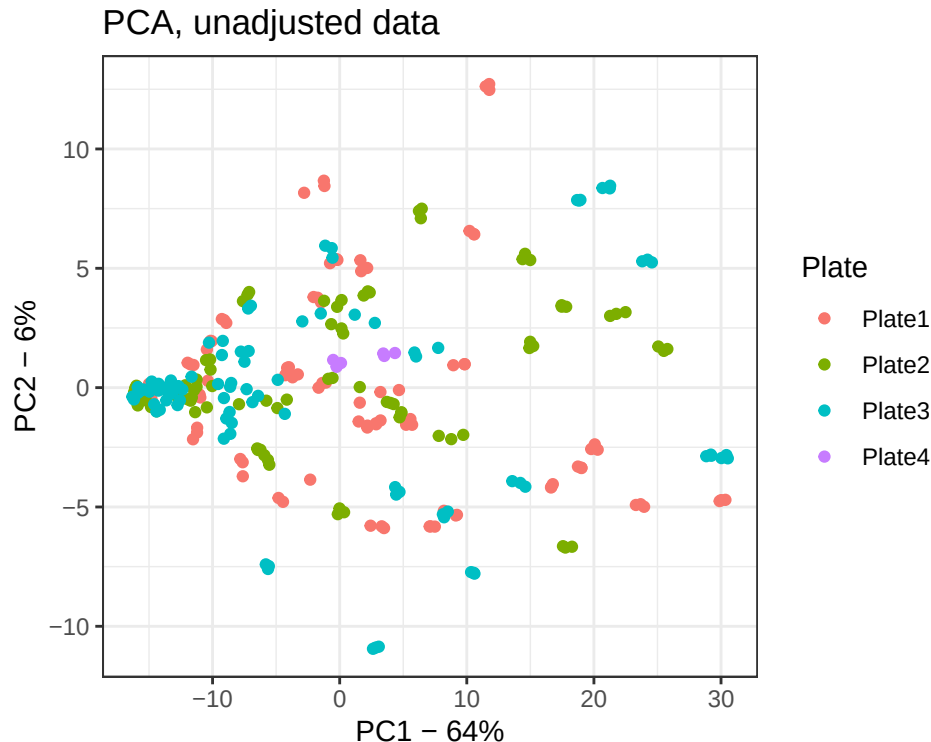
Batch effects are a well-known phenomenon in the high-throughput experiments, i.e. Microarrays, RNAseq, Luminex. For the **BBEA**, batch effects are individual microplate runs. Those are the effects that capture experimental rather than biological variability. Batch effects are easy to detect and eliminate, if experimental conditions are randomized across plate runs.

Principal Component Analysis (PCA) is a broadly used dimensionality reduction technique that allows visualization of high-dimensional data (such as epitopes, genes, proteins, etc.). Data is projected onto a few principal components (PCs), designed to maximize the explained variance of the data, and as such, the 1st PC explains most of the variance, followed by the 2nd PC, 3rd PC, and etc. A researcher can then use a 2D or 3D plot of a small number of PCs to evaluate 'overall' differences in the samples. If the samples are visually clustered together by plate, this indicates that variability in the data can be explained by the plate, which is oftentimes undesirable, since it means that technical experimental factors (instrument calibration, room light and temperature, etc.) are affecting the data.

```
pca.db<-getPCAs(eset)
pca.db$varex[1:10]
```

```
## PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10
## 64 6 5 3 2 2 2 2 1 1
```

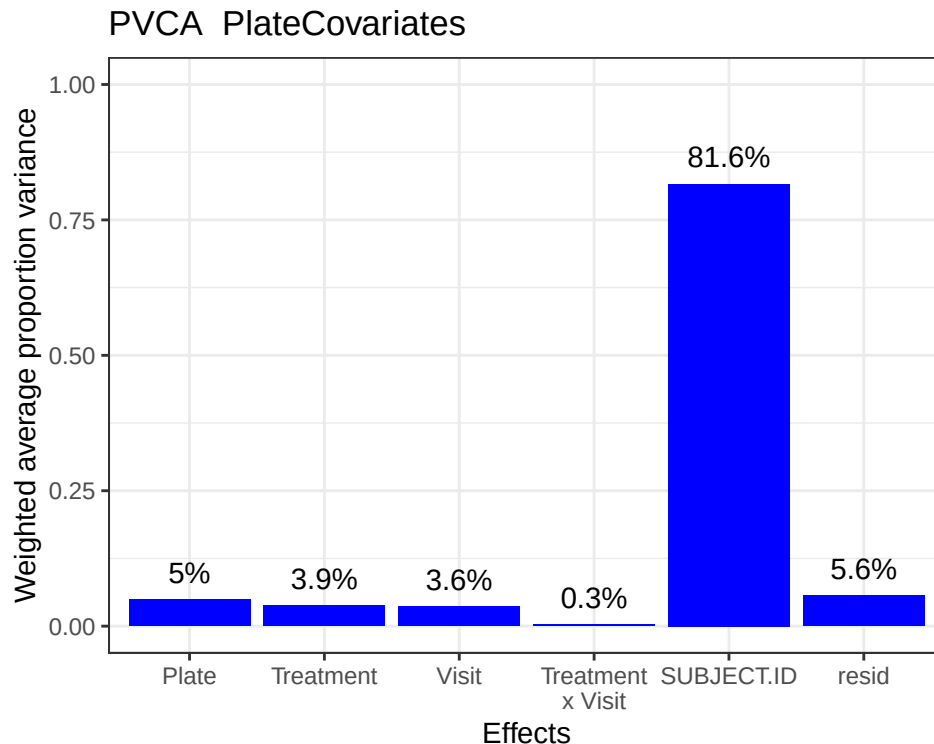
```
ggplot(pca.db$db, aes(x=PC.1, y=PC.2, color=Plate)) + geom_point() + theme_bw() +
  labs(x=paste0("PC1 - ", pca.db$varex[1], "%"),
       y=paste0("PC2 - ", pca.db$varex[2], "%"),
       title='PCA, unadjusted data')
```



While PCA is a quick way to visualize the data, it does not offer a succinct way to quantify how much of the observed variation is explained by the experimental factors, especially when there are many of such factors and/or there are suspected interaction effects. Principal Variance Component Analysis (PVCA) can be used to quantify the amount of variability attributed to different experimental conditions or sample phenotypes. It models the multivariate distribution of the PCs computed for the PCA as a function of experimental factors (such as plate, visit, diagnosis, etc.) and estimates the total variance explained by each factor via mixed-effect models. As a result, the researcher receives as output a proportion of variance that is attributed to each factor. This can be used to quantify what percentage of the total variance is explained by each factor and thus evaluate the most relevant experimental factors, including plate.

pvcaBatchAssess.bbea() function is based on the *pvca* R package available through Bioconductor.

```
pvca.obj <- pvcaBatchAssess.bbea(eset,
                                threshold=0.8,
                                batch.factors=c('Plate', 'Treatment', 'Visit', "SUBJECT.ID"),
                                include.inter='Treatment:Visit')
pvca.plot(pvca.obj, fname='PVCA.Plate.Covariates', ht=4, wd=5.5,
           order=c('Plate', 'Treatment', 'Visit', 'Treatment x Visit', "SUBJECT.ID", 'resid'))
```



The PVCA analysis shows that 5% of the variability is attributed to the “plate” effect. There are many ways to remove this variability. We will show the example of fitting a limma model for each epitope with Plate as a covariate and then subtracting the plate coefficient. A more detailed description of this method is provided in this publication.

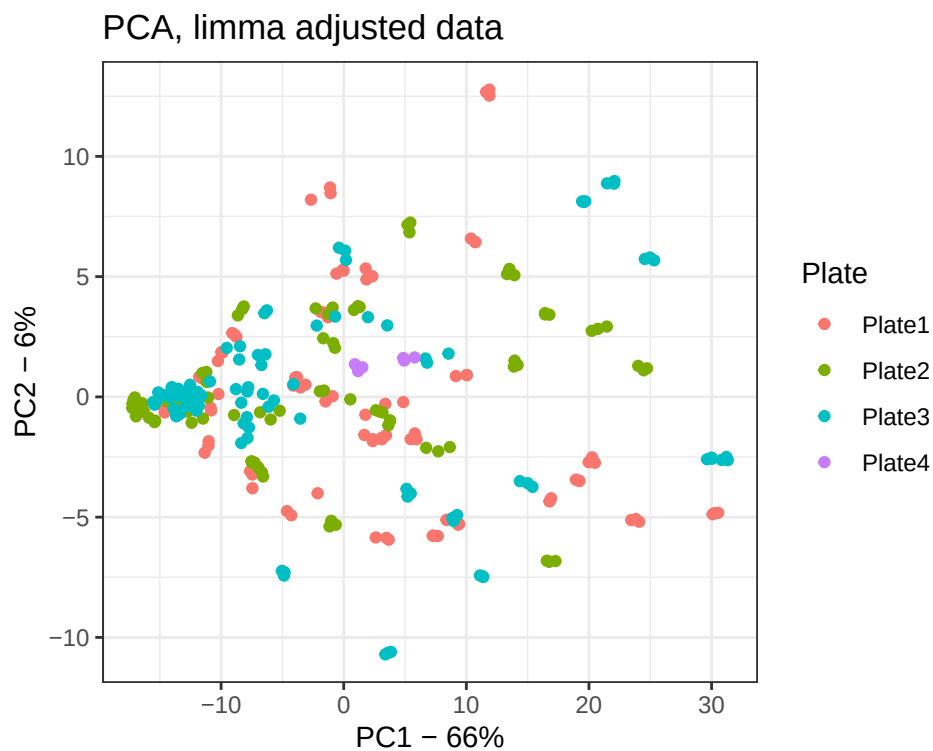
```
design <- model.matrix(~ Plate + Visit*Treatment, data=pData(eset))
cor <- duplicateCorrelation(eset, design, block=eset$SUBJECT.ID) # this estimates the correlation between
fit<-lmFit(exprs(eset), design, block=eset$SUBJECT.ID, correlation=cor$consensus)
coefs2adjust <- colnames(fit$coefficients)[grep('Plate', colnames(fit$coefficients))]
adj.exprs <- exprs(eset) - fit$coefficients[,coefs2adjust] %*% t(design[,coefs2adjust])
eset.lm <- eset
exprs(eset.lm) <- adj.exprs
```

PCA after the adjustment looks almost identical to the unadjusted one, as oftentimes the batch effect is not easily detectable “by eye”.

```
pca.db<-getPCAs(eset.lm)
pca.db$varex[1:10]
```

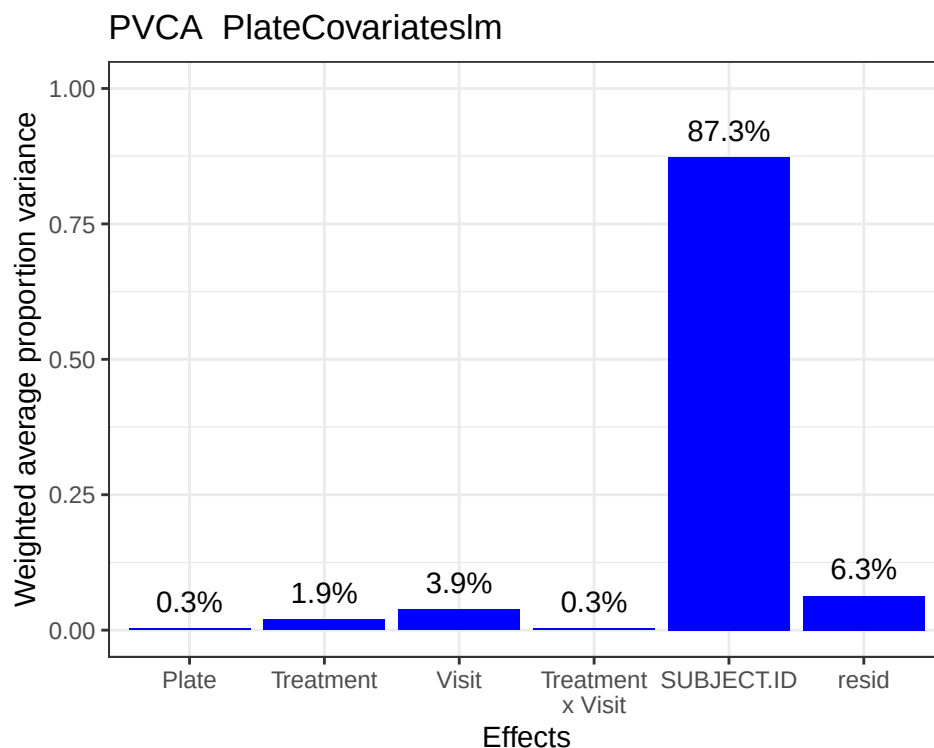
```
## PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10
## 66 6 4 3 2 2 2 2 1 1
```

```
ggplot(pca.db$db, aes(x=PC.1, y=PC.2, color=Plate)) + geom_point() + theme_bw() +
  labs(x=paste0("PC1 - ", pca.db$varex[1], "%"),
       y=paste0("PC2 - ", pca.db$varex[2], "%"),
       title='PCA, limma adjusted data')
```



However, the PVCA analysis shows that after the adjustment, “plate” accounts for 0.3% of the overall variability.

```
pvca.obj<-pvcaBatchAssess.bbea(eset.lm,
                              threshold=0.8,
                              batch.factors=c('Plate','Treatment','Visit',"SUBJECT.ID"),
                              include.inter='Treatment:Visit')
pvca.plot(pvca.obj, fname='PVCA.Plate.Covariates.lm', ht=4, wd=5.5,
          order=c('Plate','Treatment','Visit','Treatment x Visit',"SUBJECT.ID",'resid'))
```



Distributions

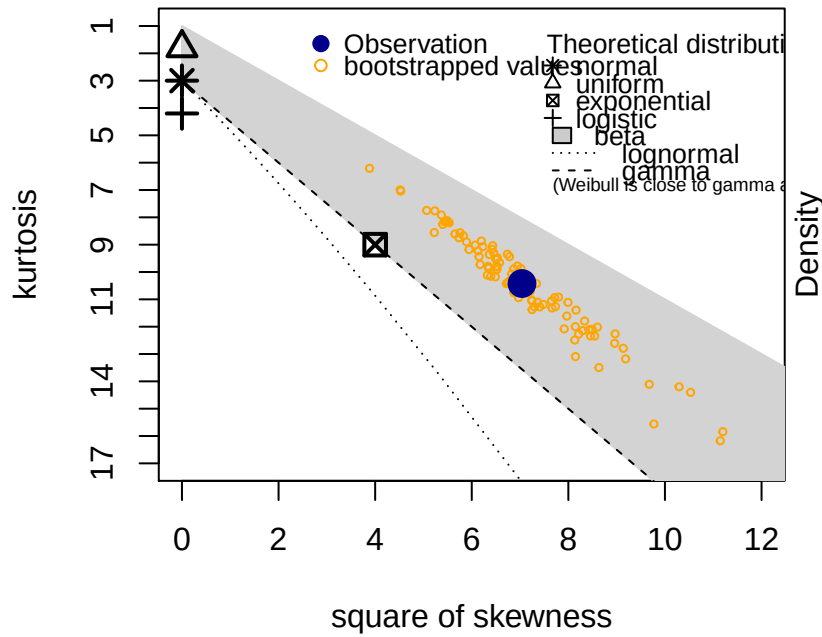
Cullen-Frey plots can be used to evaluate the distribution of the data. It shows how the skewness and kurtosis of our data compare to the theoretical distributions.

CullenFreyPlot() function is a wrapper of the `fitdistrplus::descdist()`. Since there are different levels of the antibody to each peptide, the data will be scaled before plotting. Y-axis of the boxplot represents a mean MFI or nMFI of 50 scaled peptides.

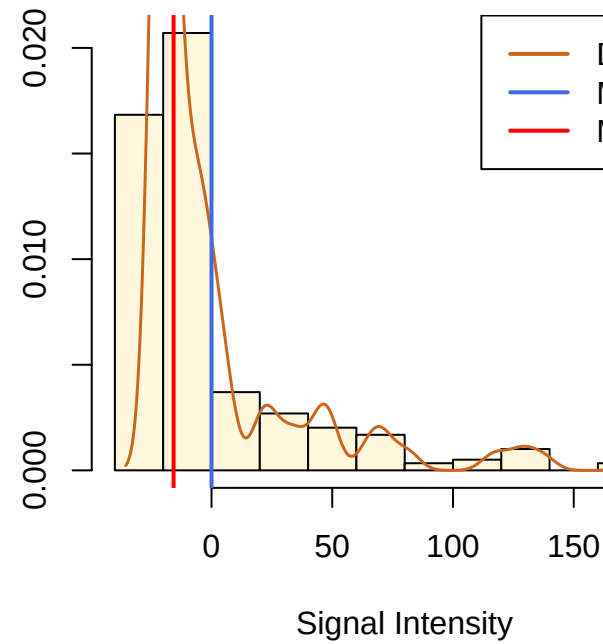
```
adjMFI <- t(apply(as.matrix(bbea$Median), 1, function(x){x - mean(x, na.rm=T)}))
adjnMFI<-t(apply(as.matrix(exprs(eset)), 1, function(x){x - mean(x, na.rm=T)}))

CullenFreyPlot(adjMFI, filename = "QC.CullenFrey.MFI")
```

Cullen and Frey graph

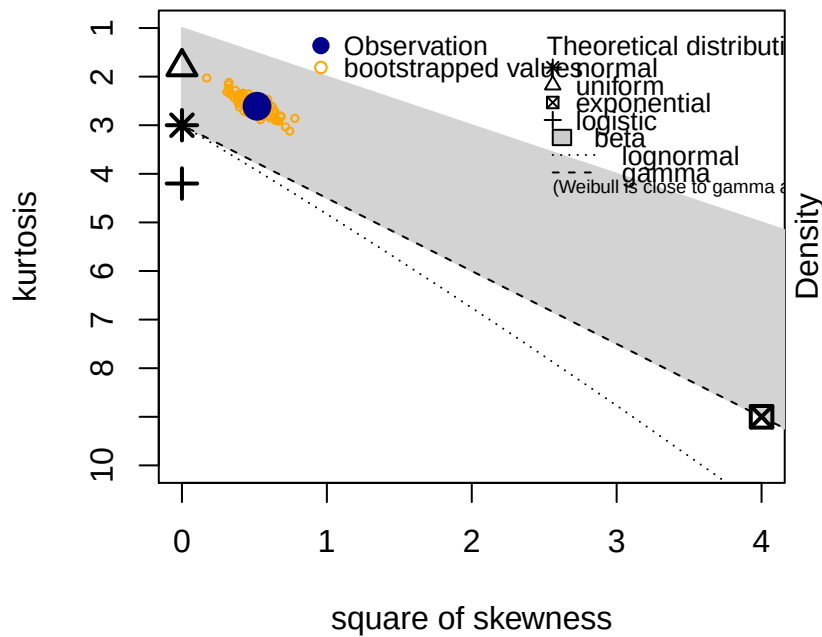


Histogram

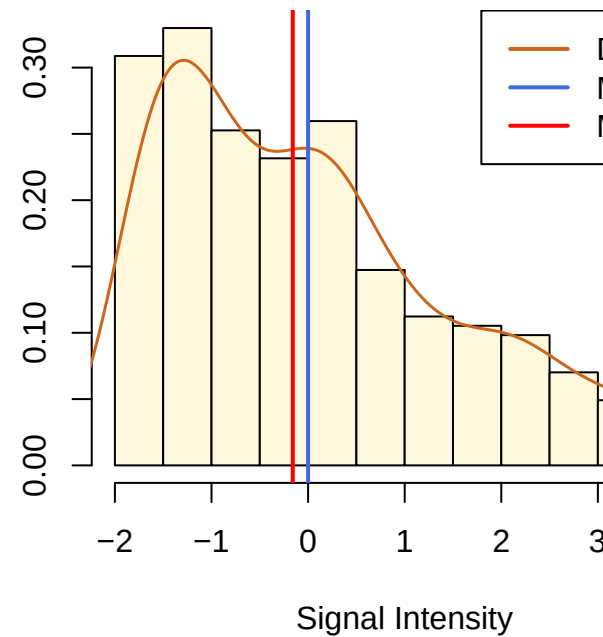


```
CullenFreyPlot(adjnMFI, filename = "QC.CullenFrey.nMFI")
```

Cullen and Frey graph



Histogram



We can see that while both MFI and nMFI data are skewed, nMFI data is closer to log-normal rather than exponential distributions.

Technical Replicates

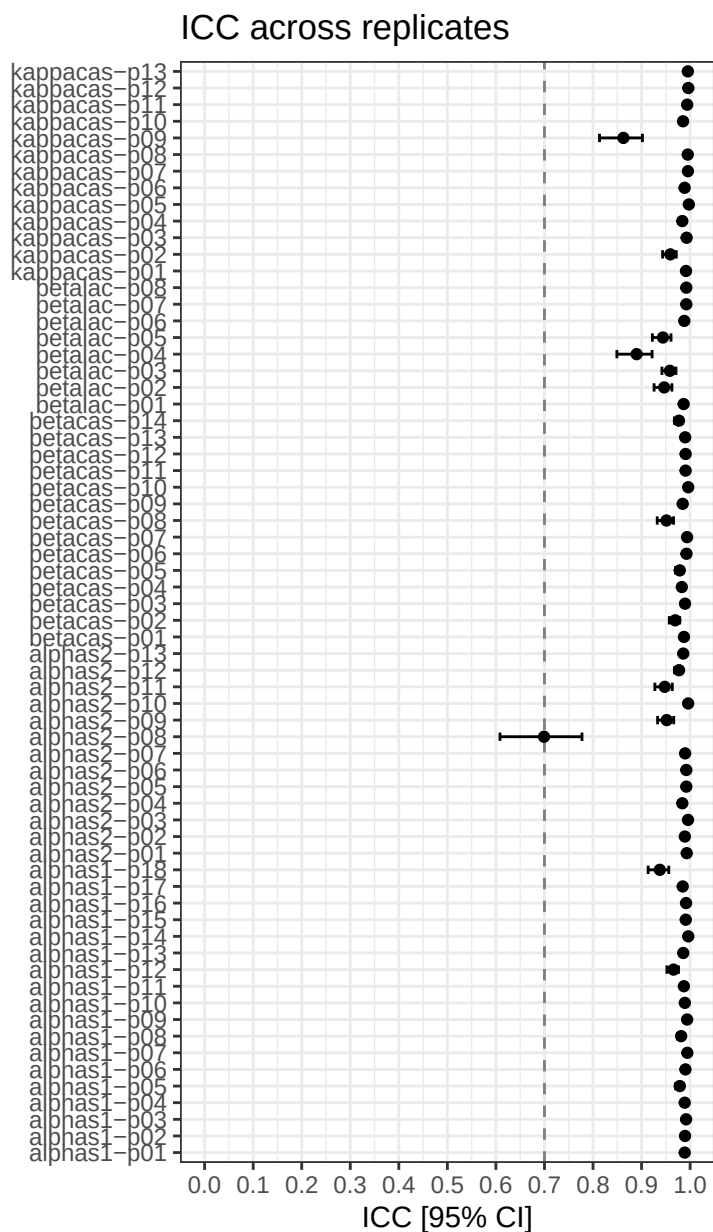
Intraclass Correlation Coefficient (ICC) is used to evaluate agreement among technical replicates for each peptide, where 0 means no agreement, and 1 is a perfect agreement.

The function returns the ICC and a 95% confidence interval.

```
icc.db <- getICCbyPeptide(eset, UR=c("SUBJECT.ID", "Visit")) # what is the unit of replication? in this  
head(icc.db)
```

##	Peptide	ICC	LCI	UCI
##	alphas1-p06	0.9903578	0.9864010	0.9933039
##	alphas1-p08	0.9815516	0.9741442	0.9871304
##	alphas1-p09	0.9938350	0.9913312	0.9957099
##	alphas1-p13	0.9858467	0.9801419	0.9901349
##	alphas1-p17	0.9847886	0.9786641	0.9893949
##	alphas1-p18	0.9375927	0.9135439	0.9560780

```
ggplot(icc.db, aes(x = Peptide, y = ICC)) +  
  geom_hline(yintercept = 0.7, color = "grey50", linetype = 2) +  
  geom_point() +  
  geom_errorbar(aes(ymax = UCI, ymin = LCI), width = 0.5) +  
  scale_y_continuous(limits = c(0, 1), breaks = seq(0, 1, 0.1)) +  
  labs(x = "", y = ("ICC [95% CI]"), title = "ICC across replicates") +  
  theme_bw() + coord_flip()
```



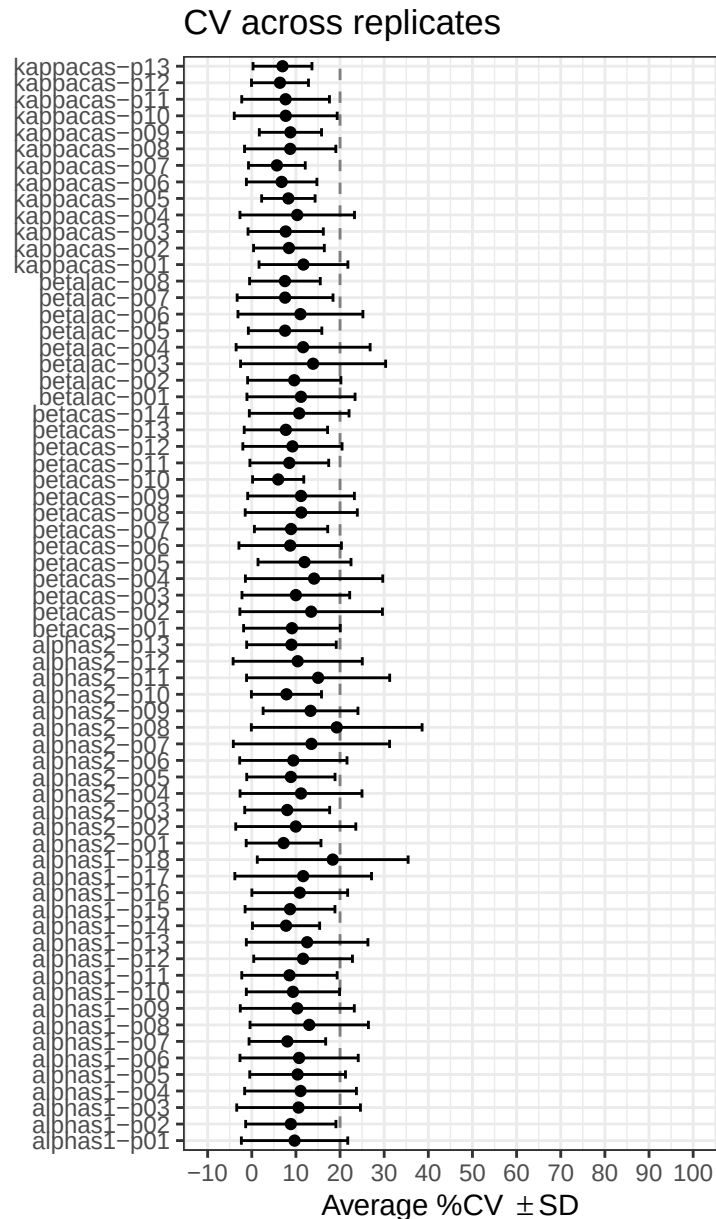
Coefficient of Variation (CV) is used to estimate variability of the replicates. Generally, for biological assays the desired CV is below 20%.

```
cv.db <- getCVbyPeptide.MFI(bbea, UR = c("SUBJECT.ID", "Visit")) # what is the unit of replication? in t
```

```
head(cv.db)
```

##	Peptide	mean.cv	sd.cv	se.cv	median.cv
## 1	alphas1-p01	9.694102	12.05228	1.236538	5.555556
## 2	alphas1-p02	8.842593	10.22483	1.049046	5.808490
## 3	alphas1-p03	10.605762	14.01331	1.437736	5.412659
## 4	alphas1-p04	11.050888	12.66247	1.299142	5.972589
## 5	alphas1-p05	10.385327	10.84504	1.112677	7.530656
## 6	alphas1-p06	10.724139	13.39413	1.374209	5.555556

```
ggplot(cv.db, aes(x = Peptide,y = mean.cv)) +
  geom_hline(yintercept = 20, color = "grey50", linetype = 2) +
  geom_point() +
  geom_errorbar(aes(ymax = mean.cv + sd.cv, ymin = mean.cv - sd.cv),width = 0.5) +
  scale_y_continuous(limits = c(-10,100),breaks = seq(-10,100,10)) +
  labs(x = "", y = expression("Average %CV "+ "%+-%"SD"), title = "CV across replicates") +
  theme_bw() + coord_flip()
```



Averaging Technical Replicates

If there are no samples to exclude based on ICC, CV and QC of counts, technical triplicates can be averaged for the downstream analyses.

```
eset.avg <- getAveragesByReps(eset, UR = c('SUBJECT.ID', 'Visit'))
eset.avg
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 66 features, 94 samples
##   element names: exprs
## protocolData: none
## phenoData
##   rowNames: MILK01_Baseline MILK01_Month32 ... MILK77_Month32 (94
##     total)
##   varLabels: SUBJECT.ID Visit ... uf (23 total)
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```

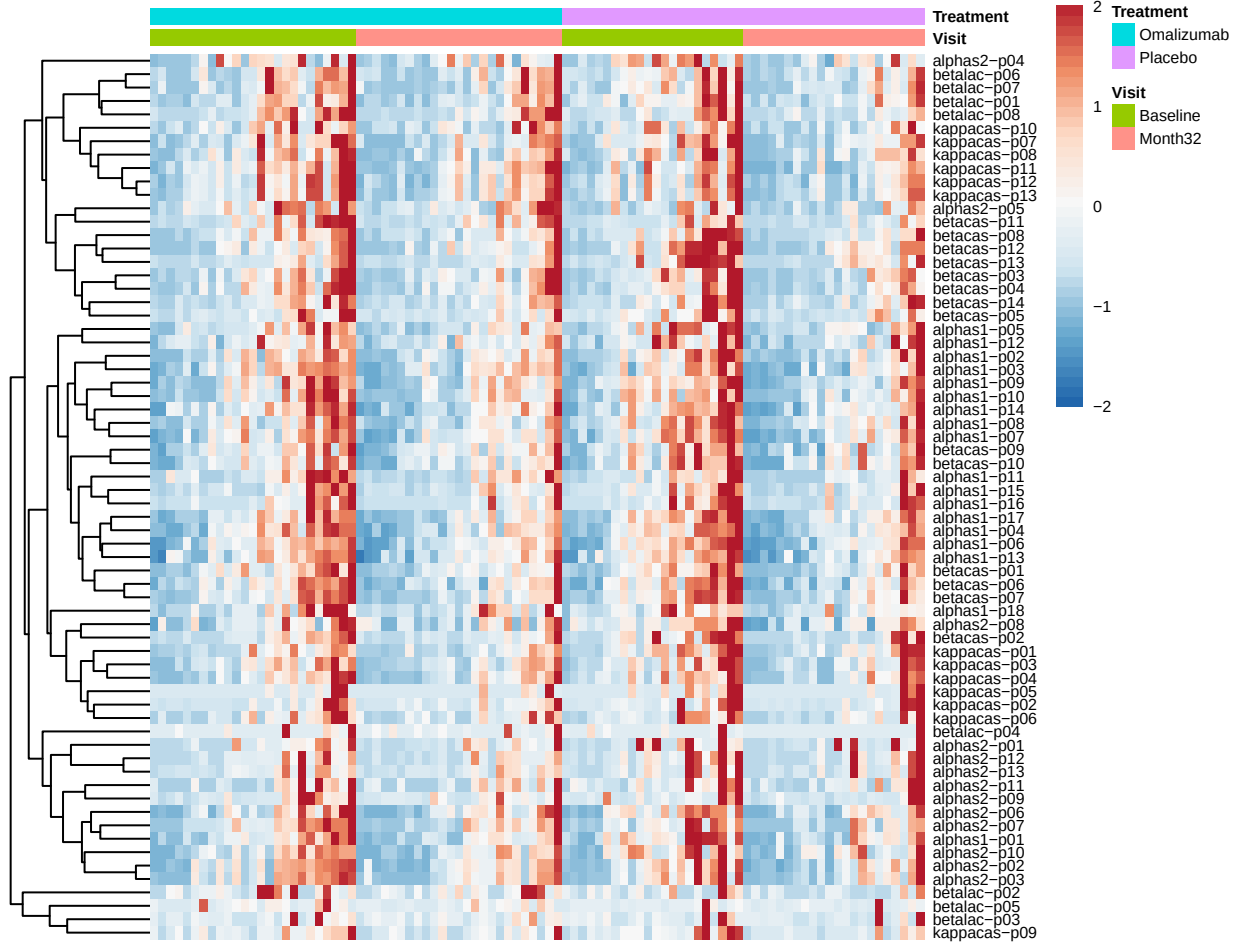
```
exprs(eset.avg)[1:5,1:3]
```

```
##           MILK01_Baseline MILK01_Month32 MILK02_Baseline
## alphas1-p06           6.747561           4.447867           6.317169
## alphas1-p08           5.685712           2.983762           4.935668
## alphas1-p09           3.559142           1.955416           8.414214
## alphas1-p13           3.379424           1.427095           8.261503
## alphas1-p17           2.839442           1.302297           7.474694
```

```
# will calculate an average of 66 epitopes for each sample
```

```
eset.avg$esIgE.avg <- colMeans(exprs(eset.avg))
```

```
ord <- with(pData(eset.avg), order(Treatment, Visit, esIgE.avg))
pheatmap(exprs(eset.avg)[,ord], scale = "row", fontsize = 6,
  annotation_col = subset(pData(eset.avg)[ord,], select = c(Visit, Treatment)),
  cluster_cols = FALSE, breaks = seq(-2, 2, 0.1),
  color = rev(colorRampPalette(brewer.pal(7, "RdBu"))(length(seq(-2, 2, 0.1)))),
  show_colnames = FALSE)
```



The heatmap shows that maybe ses-IgE levels somewhat decreased post treatment. To test this statistically, we will do some modelling.

Differential Analysis - Limma Modeling

Generally, milk OIT has been shown to induce decreases in antigen-specific IgE (Mantyla et al (2018)); however, ses-IgEs have not yet been evaluated. Additionally, we are interested whether ses-IgE antibodies decreased after patients were treated with an adjuvant Omalizumab or Placebo in addition to milk OIT.

For this, we will use a limma framework where a linear model is fit to every peptide to compare omalizumab and placebo arms.

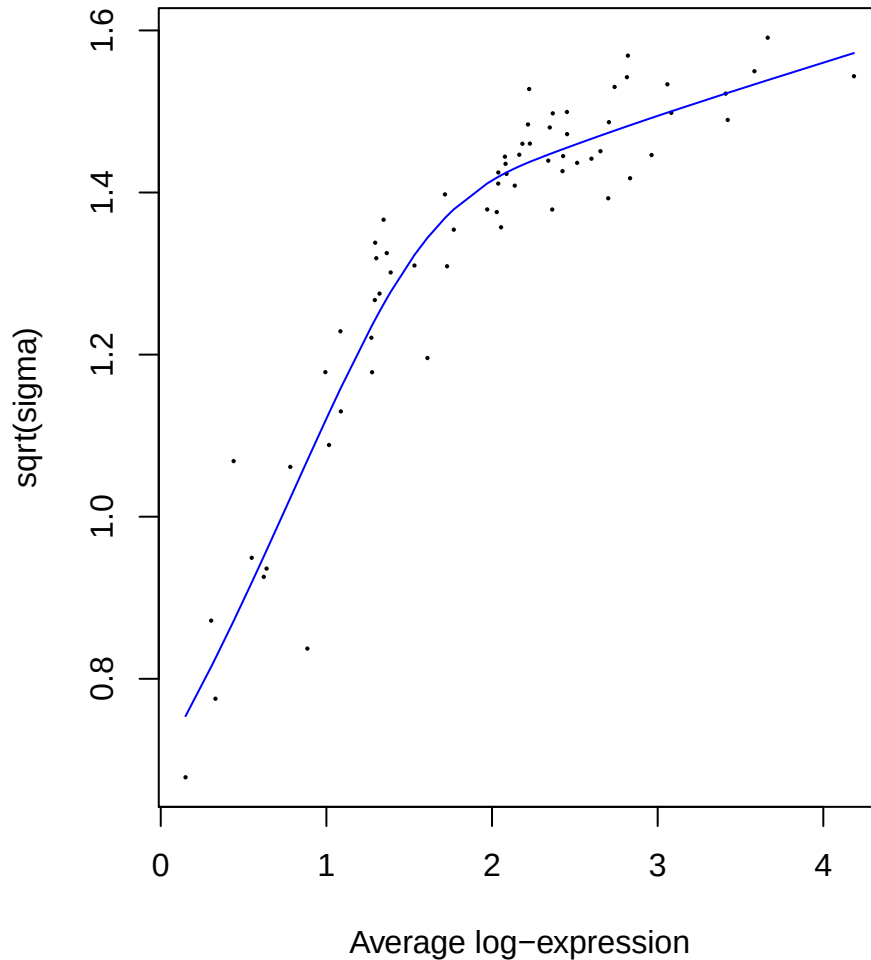
```
design <- model.matrix(~ Visit*Treatment, data=pData(eset.avg))
colnames(design) <- make.names(colnames(design))
cor <- duplicateCorrelation(exprs(eset.avg), design, block=eset.avg$SUBJECT.ID)
fit <- lmFit(exprs(eset.avg), design, block=eset.avg$SUBJECT.ID, correlation=cor$consensus)
contrm <- makeContrasts(Placebo.Baseline = X.Intercept. + TreatmentPlacebo,
                        Placebo.Post = X.Intercept. + TreatmentPlacebo + VisitMonth32 + VisitMonth32.Treatment,
                        Omalizumab.Baseline = X.Intercept.,
                        Omalizumab.Post = X.Intercept.+VisitMonth32,
                        levels=design)
contrd <- mutate(as.data.frame(contrm),
                 PostvsBaseline.Placebo = Placebo.Post - Placebo.Baseline,
```

```

PostvsBaseline.Omalizumab = Omalizumab.Post - Omalizumab.Baseline,
TreatmentEffect = PostvsBaseline.Omalizumab - PostvsBaseline.Placebo)
contrd <- contrd[,!colnames(contrd)%in%colnames(contrm)]

fitm <- eBayes(contrasts.fit(fit,contrm)) # marginal group means
ebfit <- eBayes(contrasts.fit(fit,contrd),trend=T) # lgFCH of changes
plotSA(ebfit)

```



```

D <- decideTests(ebfit,method="separate",adjust.method="BH",p.value=0.05,lfc=log2(1))
t(summary(D)) # number of significant epitopes

```

```

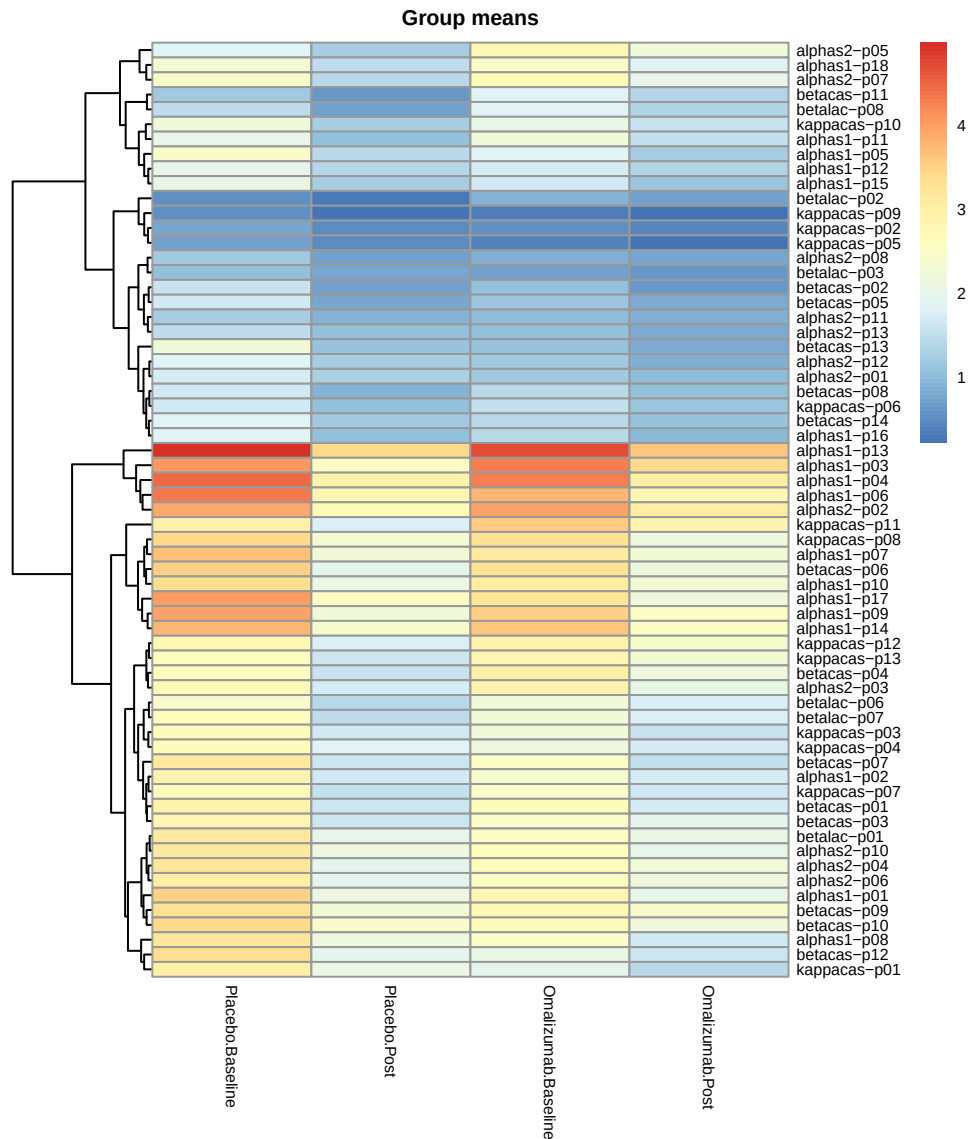
##               Down NotSig Up
## PostvsBaseline.Placebo      63      3  0
## PostvsBaseline.Omalizumab   45     21  0
## TreatmentEffect              0     66  0

```

```

# significant epitopes (FDR<0.05)
sepits <- rownames(D)[which(D[, "PostvsBaseline.Placebo"] != 0)]
pheatmap(fitm$coefficients[sepits,], cluster_cols = F,
         main = "Group means", fontsize = 6)

```

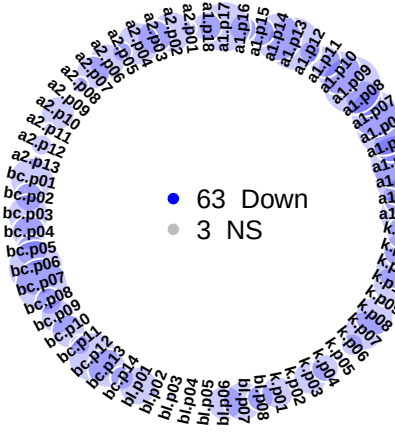


We can see that ses-IgE decreased in both Omalizumab and Placebo groups, with greater changes in more (63/66) ses-IgEs in the Placebo arm.

As described in the primary publication of the trial outcomes by (Wood et al (2015)), the majority of safety measures were in fact improved while the withdrawal rate and time needed to achieve maintenance dosing were decreased in patients taking omalizumab as compared to placebo. However, the overall clinical efficacy was similar among both groups. Further mechanistic study of this cohort (Frischmeyer-Guerrero et al (2017)) showed that milk OIT plus omalizumab had distinct changes in basophil reactivity (but not T-cells) compared to the placebo arm. Wood et al (2015) and Suarez-Farinas et al (2018) found that decreases in IgE binding to milk proteins and epitopes appeared less pronounced in subjects after milk OIT plus omalizumab compared with those receiving milk OIT plus placebo. This observation (at least in terms of IgE specific to the whole protein extract) can be explained by omalizumab 1) forming complexes with soluble IgE, which results in the increase of total IgE (since these complexes take longer to clear) and 2) reducing free IgE that will in turn diminish the amount of IgE that could bind to FcεRI receptors on the surface of mast cells and basophils, thus reducing the allergic symptoms (Lowe et al (2009)).

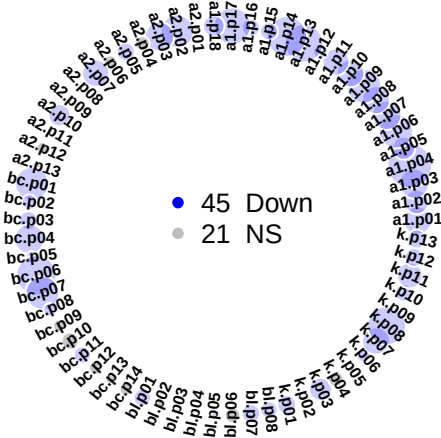
```
Run_doNetCirclePlot(ebfit[,1], D[,1], Annot, fname = "lmFit.Placebo.")
```

IgFCH @ PostvsBaseline.Placebo



```
Run_doNetCirclePlot(ebfit[,2], D[,2], Annot, fname = "lmFit.Omalizumab.")
```

IgFCH @ PostvsBaseline.Omalizumab



For the circle plots, user needs to make sure that the Annotation file has a column named “lableName”, as it will be used to print names of the epitopes.