Data pre-processing methods for high-throughput glycomics

Lucija Klarić & Frano Vučković

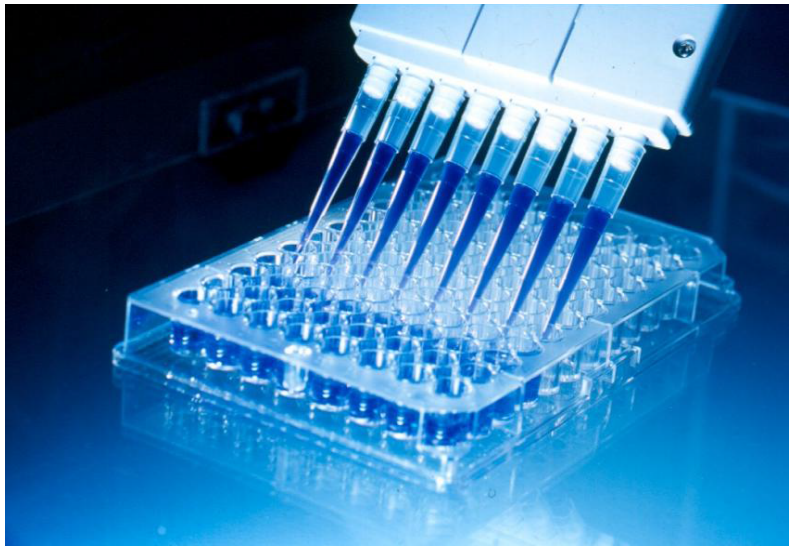
Introduction

- High-throughput analysis of glycosylation
  - UPLC
  - LCMS
  - MALDI
  - CGE-LIF

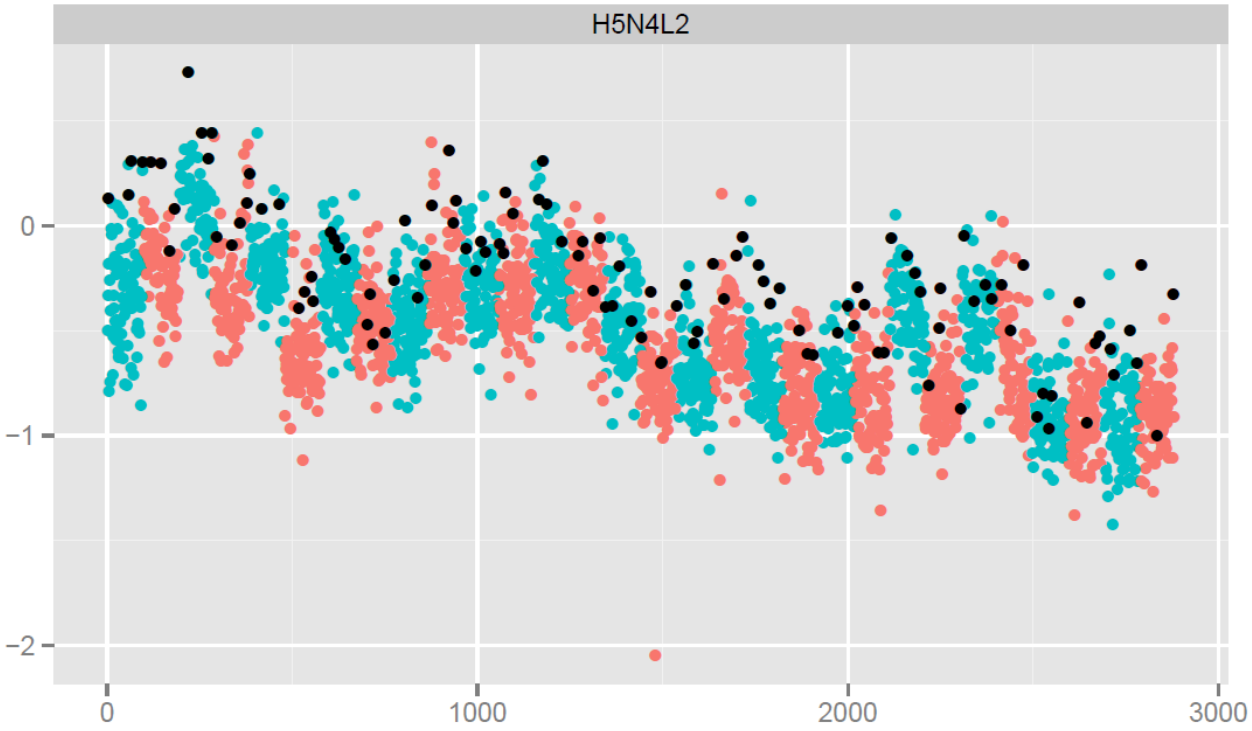
Introduction

- Accuracy of high-throughput glycomic methods is highly affected by complicated experimental procedure

- Complicated sample collection procedure
- Highly trained personnel
- Complex machines
- Huge set of chemicals

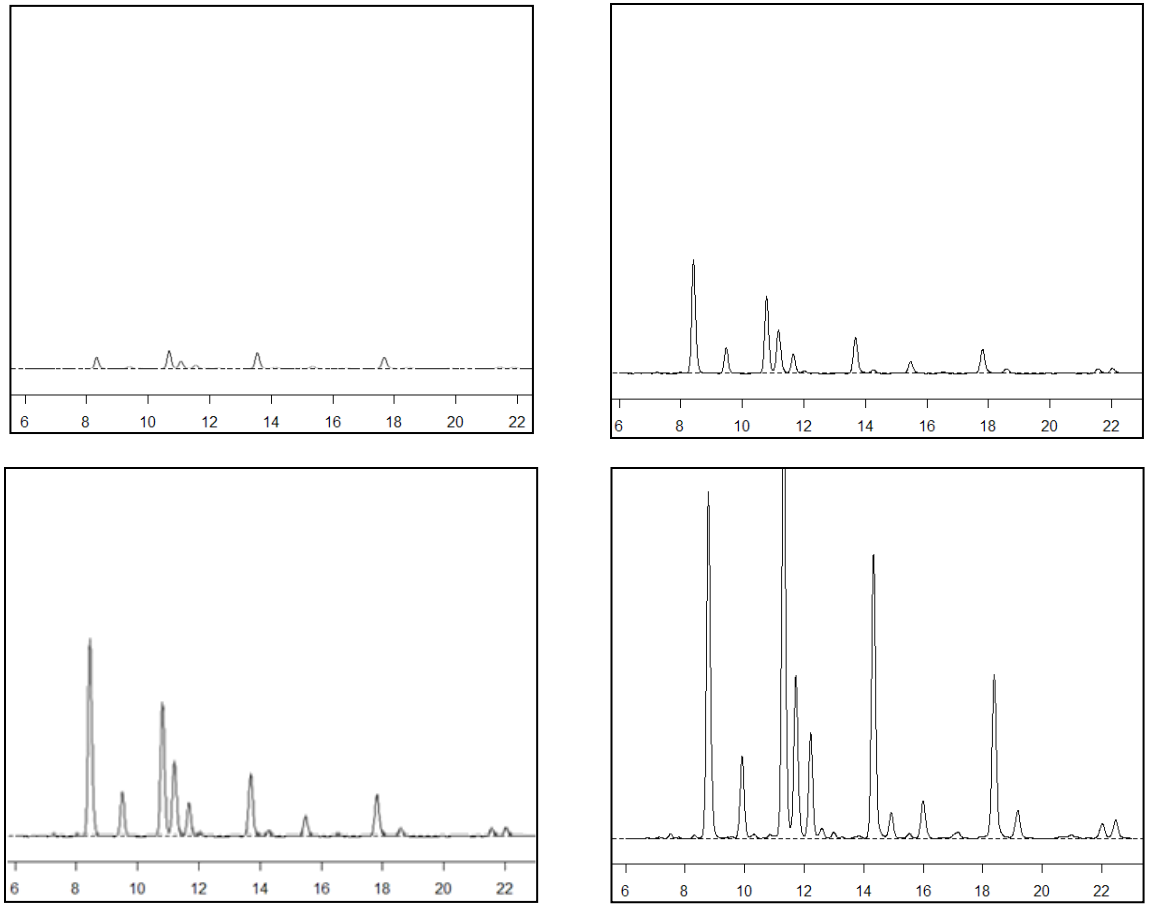


Introduction



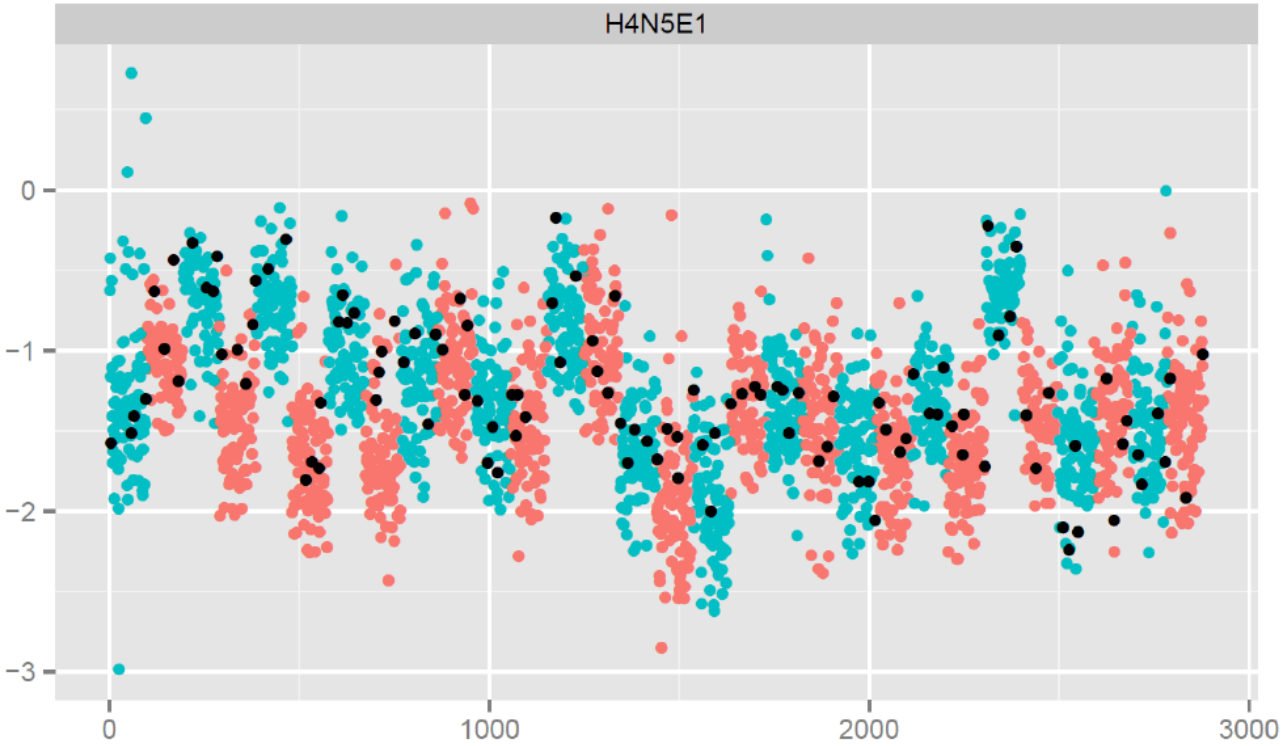
- Experiment can take several months and during that time many experimental conditions can vary
- Need for normalization and batch correction arises naturally

Introduction



- Intensities of signals can vary up to 1000 fold
- Normalization makes measurements comparable

Introduction



- Normalization does not take care of batch effects
- Batch effects can decrease statistical power and lead to wrong conclusions

Objectives

- To acquire knowledge on the basic steps of glycomic data pre-processing
- To become aware of potential error sources
- To learn principles of experimental design
- To get introduced to principles of data quality assessment

Dataset description

- UPLC plasma data
- 1389 samples (16 plates)
- 62 standards (~ 4 per plate)
- 109 duplicated samples
- completely randomized design (without blocking)

Normalization methods

- *Largest peak* normalization

$$X'_{ij} = \frac{X_{ij}}{X_{imax}}$$

- *Total area* normalization

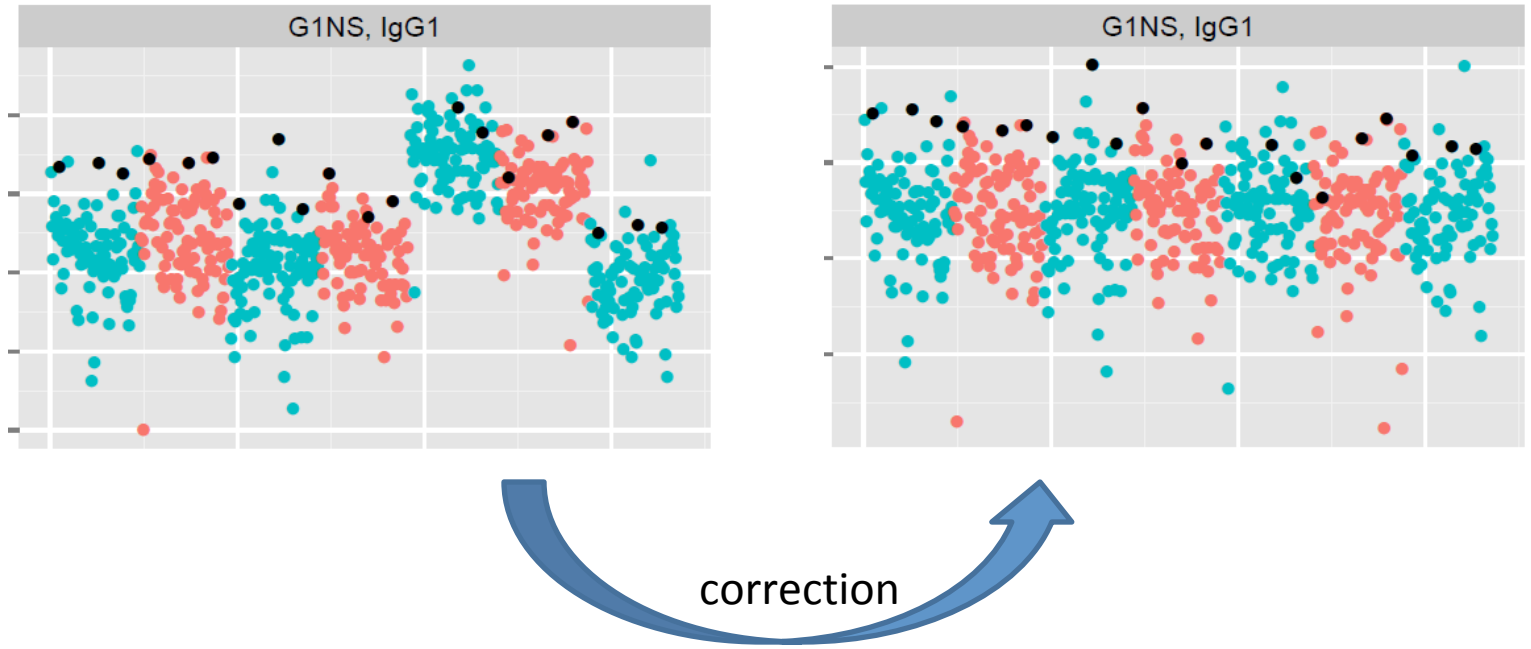
$$X'_{ij} = \frac{X_{ij}}{\sum_{j=1}^{J} X_{ij}}$$

- *Scale* normalization

$$X'_{ij} = \frac{X_{ij} - mean\ (X_i)}{sd\,(X_i)}$$

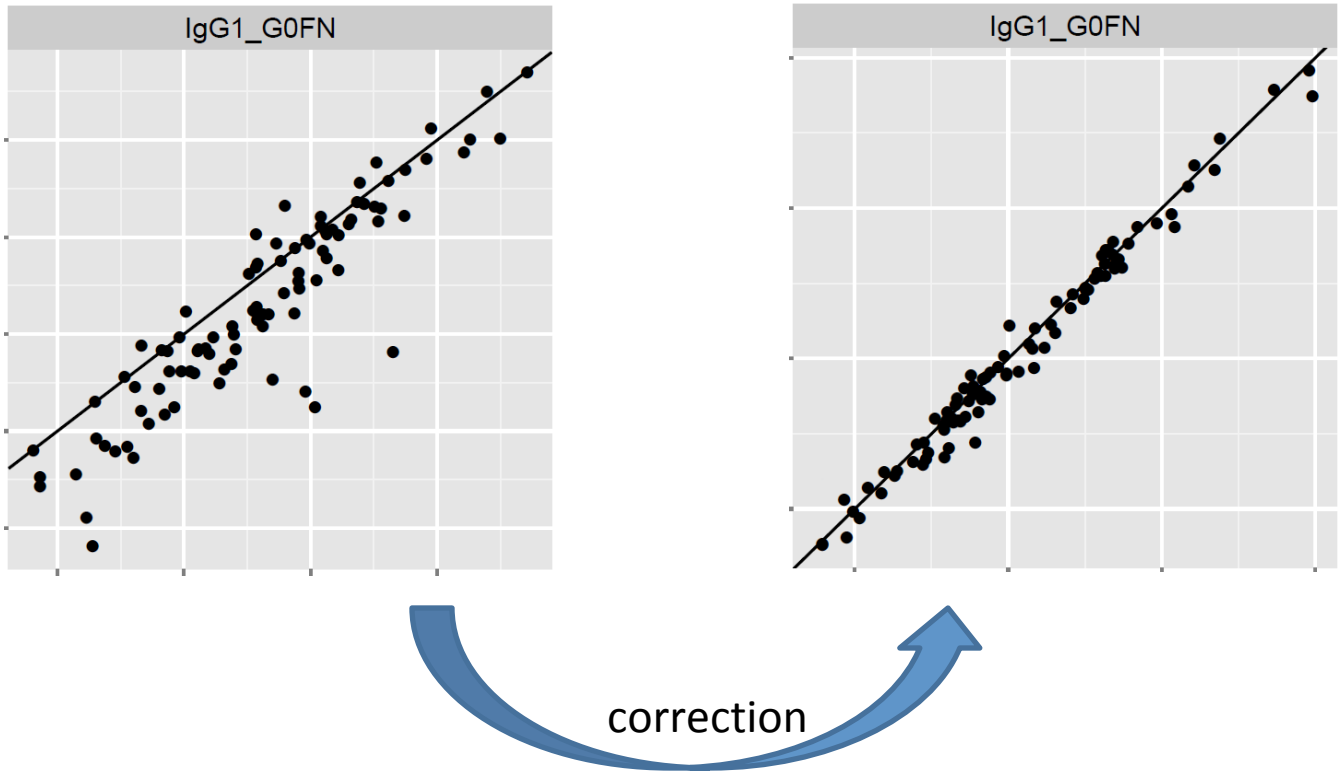
- *Global, Mixed model, Quantile* normalizations

Methods



- Variation of standards should be reduced after preprocessing procedure

Methods



- Correlation of replicates should increase after preprocessing procedure

Batch correction methods

- *Linear regression*
- *Linear mixed model*
- *ComBat* correction
- *RUV*
- *SVA*

Glycan data preprocessing

Our data is separated in three different tables, one containing unprocessed glycan data (UPLC_Plasma_Raw), one information on experimental design (UPLC_Plasma_ExpDes) and one containing clinical data (UPLC_Plasma_Clinic).

For this analysis we will need several packages.

```
library(sva)
library(lme4)
library(ggplot2)
library(plyr)
library(dplyr)
library(tidyr)
```

We have also prepared two functions (optional).

```
total_area_normalization = function(f_data) {
            glycans = grep("^GP[0-9\\.ab]+$",colnames(f_data),value=T)
            f_data[glycans] =
f_data[glycans]*100/rowSums(f_data[glycans])
            return(f_data)
}

empirical_bayes_BC = function(f_data) {
            glycans = grep("^GP[0-9\\.ab]+$",colnames(f_data),value=T)
            f_data_p = f_data
            f_data_e = t(as.matrix(f_data[glycans]))
            f_data_b = f_data$Plate
            f_data_m = model.matrix(~ 1,data = f_data_p)
            combat_data = ComBat(dat=f_data_e, batch=f_data_b,
mod=f_data_m, par.prior=TRUE, prior.plots=FALSE)
            f_data[glycans] = t(combat_data)
            return(f_data)
}
```

First we need to merge glycan and clinical data

```
data_path = "C:/Users/lucija/Dropbox/imforfuture/meetings/m12/data/"

gly_d = read.delim(paste0(data_path, "UPLC_Plasma_Raw.txt"),
stringsAsFactors = FALSE)
exp_d = read.delim(paste0(data_path, "UPLC_Plasma_ExpDes.txt"),
stringsAsFactors = FALSE)

### merging using tidyverse
# using the inner join because we are not interested in samples for
which there's no experimental info
```

```
glycans = inner_join(exp_d, gly_d, by = "Sample")

# classical way of merging:
# glycans = merge(gly_d, exp_d, by = "Sample")
```

To judge normalisation and batch correction we need to look at the variance of standards and correlation between duplicates. These are denoted in the dataset in the column Sample, with prefix or suffix "stand" and "_D".

To flag replicates in the table we first remove "_D" suffix from the Sample and using duplicated() function. This function will denote the first repetition of the same value as duplicated. Combined with fromLast=T in the same function, we can flag both replicates. The function works in a way that if there are two of the same in a vector, it will denote the second one as a duplicate. The same function with fromLast=T flag will do the oposite, will flag the first occurence as a duplicate. With the two combined we can later easily filter out only replicated samples (filter(replicates == 1)).

```
glycans = glycans %>%
  mutate(stands = ifelse(grepl("stand", Sample), "stand_yes",
"stand_no")) %>%
  mutate(tempSample = gsub("_D", "", Sample)) %>%
  # denoting the second sample in the duplicated pair
  mutate(replicates_1 = ifelse(duplicated(tempSample), 1, 0)) %>%
  # denoting the first sample in the duplicated pair
  mutate(replicates_2 = ifelse(duplicated(tempSample, fromLast = T), 1,
0)) %>%
  mutate(replicates = replicates_1 + replicates_2) %>%
  # removing temporary columns
  select(-replicates_1, -replicates_2)

 gly_repl = glycans %>% filter(replicates == 1)

 dim(gly_repl)

## [1] 222  43
```

To check the correlation between replicated samples in the raw data we need the data in the format where each row is a sample and two columns represent its glycan levels - one original measurement and one the duplicated measurement.

```
gly_repl = gly_repl %>%
  # marking which sample is actually a duplicate
  mutate(duplicated = (grepl("_D", Sample))) %>%
  # just renaming elements in the column so that it's easier to handle
them later on
  mutate(duplicated = ifelse(duplicated == T, "repl_yes", "repl_no"))

# reformatting to a long format for quicker handling of all glycans at
once
```

```
gly_repl_l = gly_repl %>% gather(glycan, level, contains("GP"))

## reformating the data so that each sample has just one row and two
columns, one for each measurement
gly_repl_l = gly_repl_l %>%
  # removing all columns that are not needed. they would also mess up
spreading to the wide format if left
  select(-Sample, -Plate, -Column, -Row, -replicates, -stands) %>%
  spread(duplicated, level)

head(gly_repl_l)
```

```
##   tempSample  glycan repl_no repl_yes
## 1    ID_0055     GP1 2473785  2473785
## 2    ID_0055    GP10 1206338  1206338
## 3    ID_0055    GP11  386289   386289
## 4    ID_0055    GP12  571375   571375
## 5    ID_0055    GP13  254678   254678
## 6    ID_0055 GP14.15 8794893  8794893
```

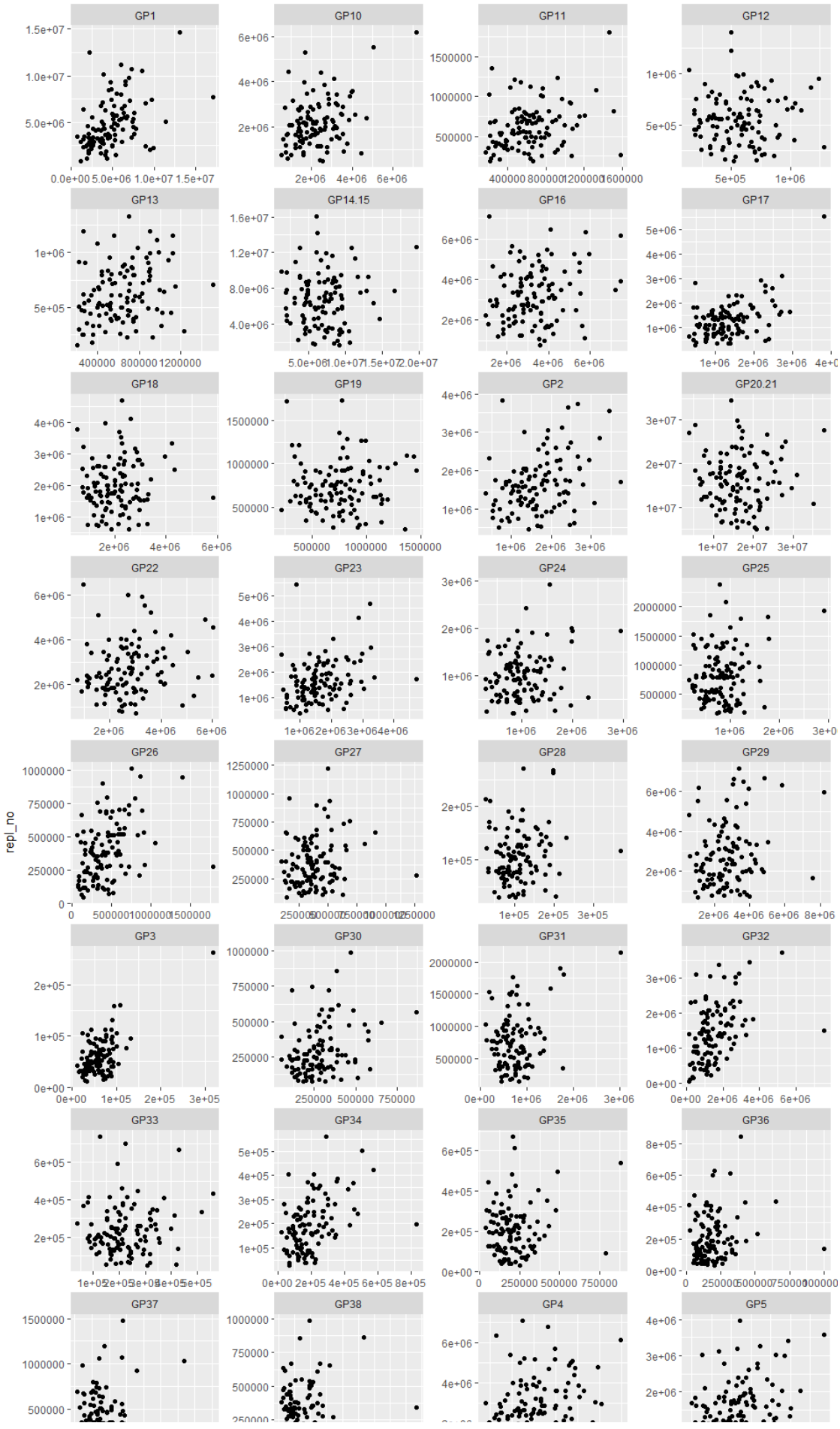
Now it's very simple to calculate and visualise correlation between replicates for every glycan.

```
# calculating correlation
gly_repl_cor_raw = gly_repl_l %>%
  group_by(glycan) %>%
  summarise(repl_cor = cor(repl_yes,repl_no)) %>%
  ungroup()

head(gly_repl_cor_raw)
```

```
## # A tibble: 6 × 2
##    glycan   repl_cor
##     <chr>      <dbl>
## 1     GP1 0.44301366
## 2    GP10 0.37485267
## 3    GP11 0.27477033
## 4    GP12 0.02982881
## 5    GP13 0.25821812
## 6 GP14.15 0.05658971
```

Visualising correlation of replicates.

```
ggplot(gly_repl_l, aes(x = repl_yes, y = repl_no)) +
  geom_point() +
  facet_wrap(~glycan, scales = "free", ncol = 4)
```


GP1
GP10
GP11
GP12
GP13
GP14.15
GP16
GP17
GP18
GP19
GP2
GP20.21
GP22
GP23
GP24
GP25
GP26
GP27
GP28
GP29
GP3
GP30
GP31
GP32
GP33
GP34
GP35
GP36
GP37
GP38
GP4
GP5
repl_no

Now switching to standards to calculate what percentage of the variance in every glycan can be explained by variance in standards.

```
raw_stands_var = glycans %>%
  gather(glycan, level, contains("GP")) %>%
  group_by(glycan, stands) %>%
  summarise(variance = var(log(level)))

raw_stands_var = raw_stands_var %>%
  spread(stands, variance) %>%
  mutate(expVar = 100*stand_yes/stand_no)

head(raw_stands_var)
```

```
## Source: local data frame [6 x 4]
## Groups: glycan [6]
##
##    glycan  stand_no stand_yes    expVar
##     <chr>     <dbl>     <dbl>     <dbl>
## 1     GP1 0.2957300 0.1545654  52.26572
## 2    GP10 0.2285738 0.1869484  81.78910
## 3    GP11 0.2204822 0.1948788  88.38752
## 4    GP12 0.2193157 0.2606160 118.83141
## 5    GP13 0.2038661 0.1601523  78.55759
## 6 GP14.15 0.2444623 0.2917176 119.33031
```

Now we are ready to perform normalisation. In this tutorial we will use total area normalisation, but be aware that different normalisations skew the data in different ways and that a different normalisation might be more appropriate for your downstream analyses.

```
glycans_norm = total_area_normalization(glycans)
```

Comparing correlation between replicates and variation of standards before and after normalisation. First computing the quantites on the normalised data, then merging with the same performed on the raw data.

```
# taking the same code as above, just renaming variables; it would be
wiser to write a function for this

# correlation of replicates
gly_repl_norm = glycans_norm %>% filter(replicates == 1)
gly_repl_norm = gly_repl_norm %>%
  mutate(duplicated = (grepl("_D", Sample))) %>%
  mutate(duplicated = ifelse(duplicated == T, "repl_yes", "repl_no"))

gly_repl_norm_l = gly_repl_norm %>%
  gather(glycan, level, contains("GP")) %>%
  select(-Sample, -Plate, -Column, -Row, -replicates, -stands) %>%
  spread(duplicated, level)
```

```r
gly_repl_cor_norm = gly_repl_norm_l %>%
  group_by(glycan) %>%
  summarise(repl_cor = cor(repl_yes,repl_no)) %>%
  ungroup()

repl_cor_rawVsNorm = merge(gly_repl_cor_raw, gly_repl_cor_norm, by =
"glycan", suffixes = c(".raw", ".norm"))

data.frame(repl_cor_rawVsNorm)
```

```
##     glycan repl_cor.raw repl_cor.norm
## 1      GP1  0.443013658    0.78389410
## 2     GP10  0.374852666    0.74053354
## 3     GP11  0.274770325    0.34912322
## 4     GP12  0.029828812    0.49253602
## 5     GP13  0.258218117    0.59065464
## 6   GP14.15  0.056589714    0.35997352
## 7     GP16  0.194796858    0.72172322
## 8     GP17  0.512934814    0.89073087
## 9     GP18  0.065209521    0.68641666
## 10    GP19  0.007048208    0.25385044
## 11     GP2  0.337446471    0.82271817
## 12 GP20.21  0.009591320    0.70089482
## 13    GP22  0.166658541    0.83786316
## 14    GP23  0.318825745    0.83671644
## 15    GP24  0.177366333    0.65993067
## 16    GP25  0.187622188    0.56321194
## 17    GP26  0.398682404    0.77792249
## 18    GP27  0.155285438    0.78167561
## 19    GP28  0.125009222    0.30609158
## 20    GP29  0.224559095    0.74454122
## 21     GP3  0.634354523    0.84527620
## 22    GP30  0.324817055    0.83339840
## 23    GP31  0.288999655    0.63326218
## 24    GP32  0.434755888    0.86212958
## 25    GP33  0.105014736    0.15428280
## 26    GP34  0.437377321    0.88982108
## 27    GP35  0.142257472    0.22960845
## 28    GP36  0.164219576    0.45894219
## 29    GP37  0.134209867    0.29807577
## 30    GP38  0.255796447    0.50772776
## 31     GP4  0.314891896    0.67443573
## 32     GP5  0.319489847    0.60482762
## 33     GP6  0.295056739    0.67811043
## 34     GP7  0.050544217    0.65244184
## 35     GP8  0.156852770    0.42652308
## 36     GP9  0.070522488    0.07490509
```

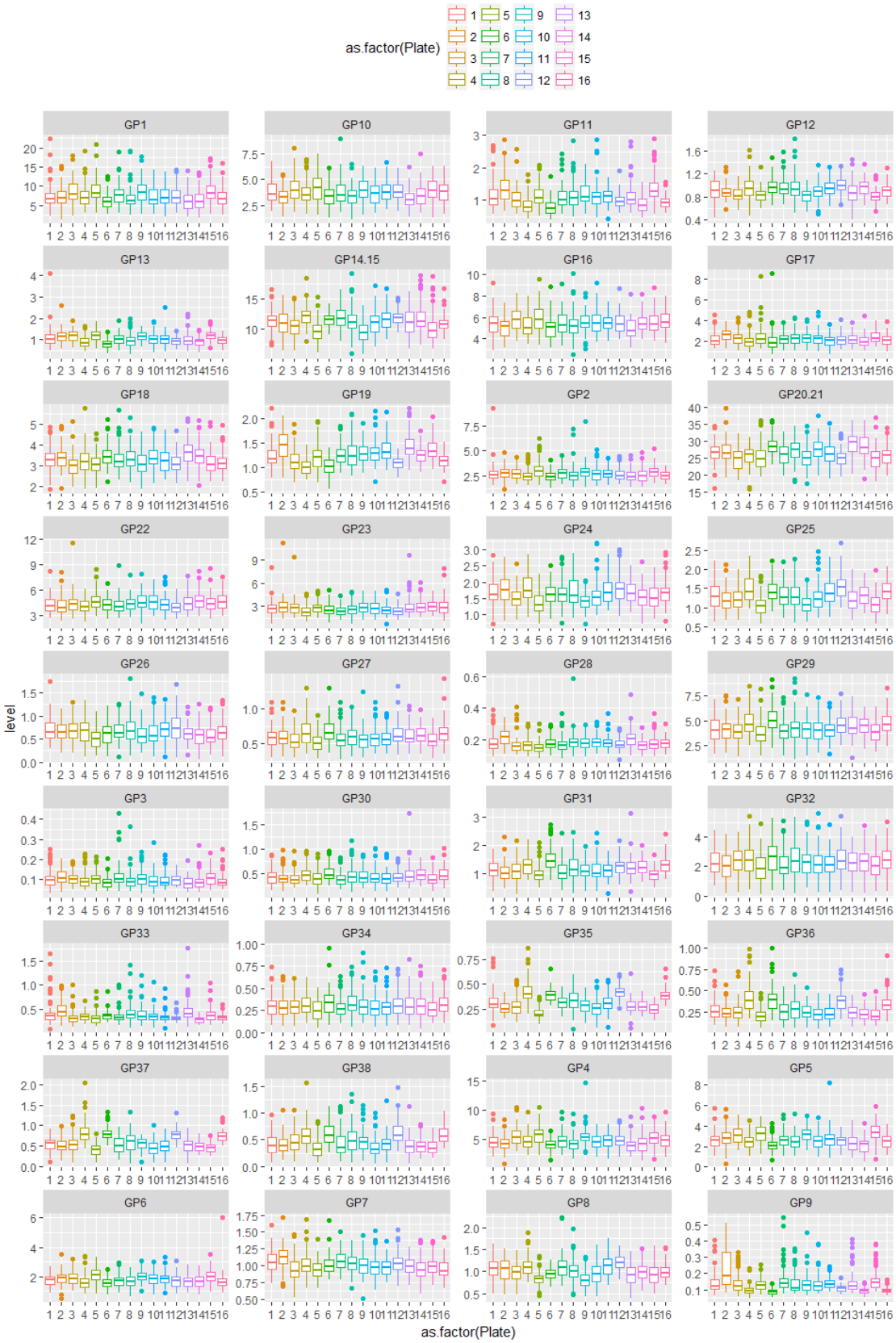
```r
# experimental variation
norm_stands_var = glycans_norm %>%
  gather(glycan, level, contains("GP")) %>%
  group_by(glycan, stands) %>%
  summarise(variance = var(log(level)))

norm_stands_var = norm_stands_var %>%
  spread(stands, variance) %>%
  mutate(expVar = 100*stand_yes/stand_no)

stands_var_rawVsNorm = merge(raw_stands_var, norm_stands_var, by =
"glycan", suffixes = c(".raw", ".norm"))

data.frame(select(stands_var_rawVsNorm, glycan, expVar.raw,
expVar.norm))
```

```
##     glycan expVar.raw expVar.norm
## 1      GP1   52.26572    6.812304
## 2     GP10   81.78910   11.249561
## 3     GP11   88.38752   50.971146
## 4     GP12  118.83141   41.480516
## 5     GP13   78.55759   14.841195
## 6   GP14.15  119.33031   62.698293
## 7     GP16   98.66729   17.128172
## 8     GP17   77.03749   10.118027
## 9     GP18  106.15666   11.688611
## 10    GP19  112.35290   81.593162
## 11     GP2   84.80448    4.565832
## 12 GP20.21  110.34916   12.439509
## 13    GP22   91.20701    8.583853
## 14    GP23   70.25781    5.754030
## 15    GP24  101.76874   24.226019
## 16    GP25  108.53049   54.296121
## 17    GP26   79.62426   17.364921
## 18    GP27   92.95039   12.447082
## 19    GP28   97.73799   28.563492
## 20    GP29   81.59311   13.494180
## 21     GP3   68.40062    7.472540
## 22    GP30   66.89891   13.115264
## 23    GP31   86.57479   25.600200
## 24    GP32   53.78245    6.238206
## 25    GP33  102.07653   97.428556
## 26    GP34   66.37169    7.785532
## 27    GP35  101.61103   49.194818
## 28    GP36   94.27512   48.944232
## 29    GP37   94.62845   46.507281
## 30    GP38   83.26019   36.273458
## 31     GP4   72.59347   10.546742
## 32     GP5   65.97245   12.166535
## 33     GP6   78.39492   11.832175
```

```
## 34     GP7  122.50185   28.780332
## 35     GP8  106.66118   54.290535
## 36     GP9   91.28749   66.025555
```

Moving to batch correction. To visually assess if batch effect is present we plot the distribution of each glycan on different plates.

```
# it's easiest to plot the data if they are in long format
glycans_norm_l = glycans_norm %>% gather(glycan, level, contains("GP"))

# visualising batch effect
ggplot(glycans_norm_l, aes(x = as.factor(Plate), y = level, colour =
as.factor(Plate))) +
  geom_boxplot() +
  facet_wrap(~glycan, scales = "free", ncol = 4) +
  theme(legend.position = "top", legend.direction = "horizontal")
```

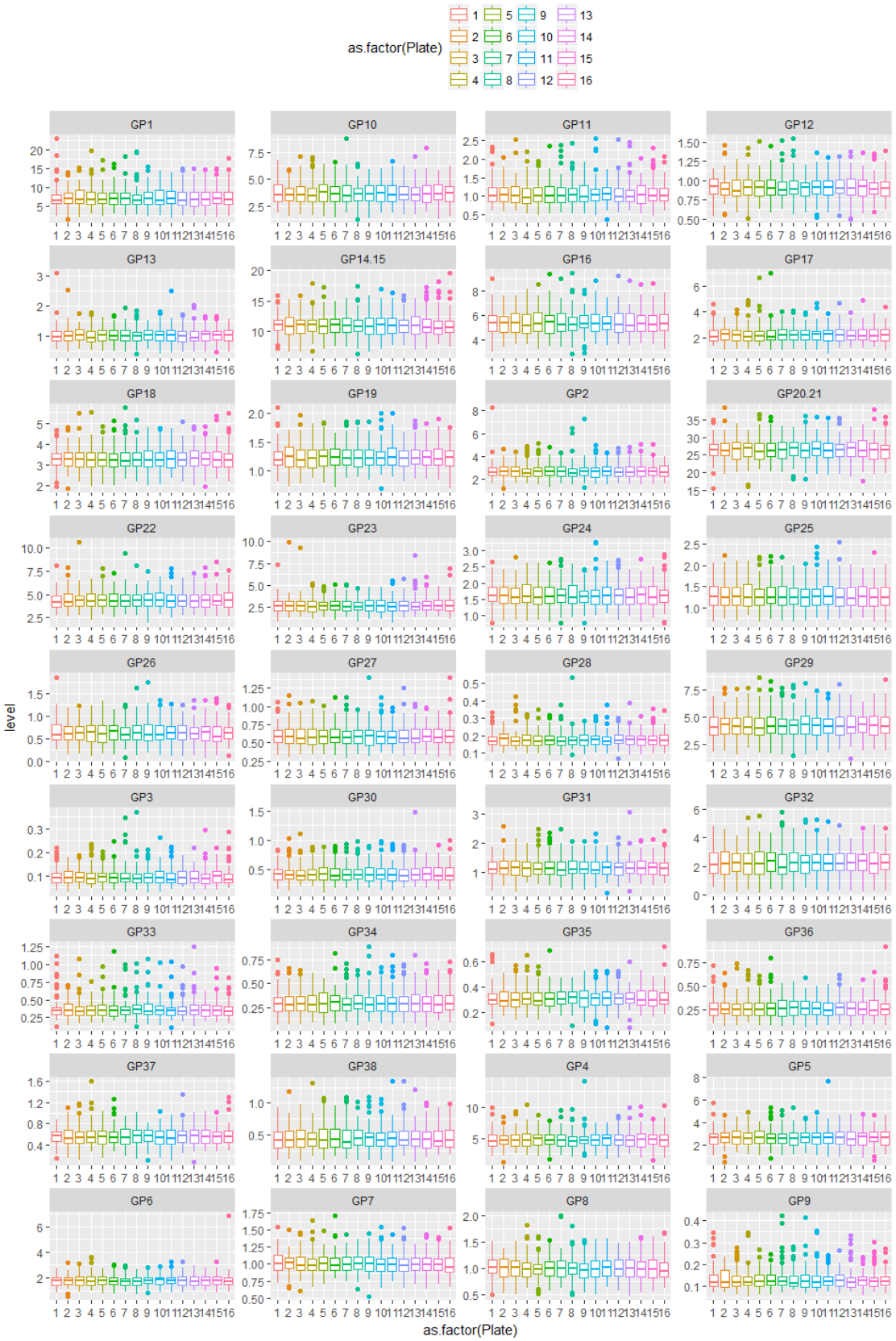

as.factor(Plate)
1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16
GP1
GP10
GP11
GP12
GP13
GP14.15
GP16
GP17
GP18
GP19
GP2
GP20.21
GP22
GP23
GP24
GP25
GP26
GP27
GP28
GP29
GP3
GP30
GP31
GP32
GP33
GP34
GP35
GP36
GP37
GP38
GP4
GP5
GP6
GP7
GP8
GP9
level
as.factor(Plate)

Since normalised glycans are still right skewed (check distribution), prior to batch correction we log transform the data. There are many different ways the batch correction can be done, but here we are using empirical Bayes approach from Johnson *et al*[1].

```
# log transforming
glycans_norm_l = glycans_norm_l %>%
  mutate(level = log(level))

# to use the provided function for batch correction we need to reformat
data back to the wide format
glycans_norm_bc = glycans_norm_l %>% spread(glycan, level)
glycans_norm_bc = empirical_bayes_BC(glycans_norm_bc)
```

```
## Found 16 batches
## Adjusting for 0 covariate(s) or covariate level(s)
## Standardizing Data across genes
## Fitting L/S model and finding priors
## Finding parametric adjustments
## Adjusting the Data
```

```
# transforming data back to original scale and plotting again to check
if we removed batch effect
glycans_norm_bc_l = glycans_norm_bc %>%
  gather(glycan, level, contains("GP")) %>%
  mutate(level = exp(level))

ggplot(glycans_norm_bc_l, aes(x = as.factor(Plate), y = level, colour =
as.factor(Plate))) +
  geom_boxplot() +
  facet_wrap(~glycan, scales = "free", ncol = 4) +
  theme(legend.position = "top", legend.direction = "horizontal")
```


as.factor(Plate)
1 5 9 13
2 6 10 14
3 7 11 15
4 8 12 16
GP1
GP10
GP11
GP12
GP13
GP14.15
GP16
GP17
GP18
GP19
GP2
GP20.21
GP22
GP23
GP24
GP25
GP26
GP27
GP28
GP29
GP3
GP30
GP31
GP32
GP33
GP34
GP35
GP36
GP37
GP38
GP4
GP5
GP6
GP7
GP8
GP9
level
as.factor(Plate)

```
# reformatting back to wide format
glycans_norm_bc = glycans_norm_bc_l %>% spread(glycan, level)
```

For the last step we want to again check if batch correction further reduced experimental variation in our data. Recalculating correlation between replicates and variance of standards.

```
# taking the same code as above, just renaming variables; it would be
wiser to write a function for this
# correlation of replicates
gly_repl_norm_bc = glycans_norm_bc %>% filter(replicates == 1)
gly_repl_norm_bc = gly_repl_norm_bc %>%
  mutate(duplicated = (grepl("_D", Sample))) %>%
  mutate(duplicated = ifelse(duplicated == T, "repl_yes", "repl_no"))

gly_repl_norm_bc_l = gly_repl_norm_bc %>%
  gather(glycan, level, contains("GP")) %>%
  select(-Sample, -Plate, -Column, -Row, -replicates, -stands) %>%
  spread(duplicated, level)

gly_repl_cor_norm_bc = gly_repl_norm_bc_l %>%
  group_by(glycan) %>%
  summarise(repl_cor.norm.bc = cor(repl_yes,repl_no)) %>%
  ungroup()

repl_cor_rawNormBC = merge(repl_cor_rawVsNorm, gly_repl_cor_norm_bc, by
= "glycan")

data.frame(repl_cor_rawNormBC)
```

```
##     glycan repl_cor.raw repl_cor.norm repl_cor.norm.bc
## 1      GP1  0.443013658    0.78389410        0.8264207
## 2     GP10  0.374852666    0.74053354        0.8293276
## 3     GP11  0.274770325    0.34912322        0.5493190
## 4     GP12  0.029828812    0.49253602        0.5722496
## 5     GP13  0.258218117    0.59065464        0.7321666
## 6  GP14.15  0.056589714    0.35997352        0.4802999
## 7     GP16  0.194796858    0.72172322        0.7750713
## 8     GP17  0.512934814    0.89073087        0.9033585
## 9     GP18  0.065209521    0.68641666        0.7864400
## 10    GP19  0.007048208    0.25385044        0.4097470
## 11     GP2  0.337446471    0.82271817        0.8767046
## 12 GP20.21  0.009591320    0.70089482        0.8015319
## 13    GP22  0.166658541    0.83786316        0.8326428
## 14    GP23  0.318825745    0.83671644        0.8772560
## 15    GP24  0.177366333    0.65993067        0.7499168
## 16    GP25  0.187622188    0.56321194        0.7085905
## 17    GP26  0.398682404    0.77792249        0.8027670
## 18    GP27  0.155285438    0.78167561        0.8777749
## 19    GP28  0.125009222    0.30609158        0.4153772
```

```
## 20    GP29  0.224559095    0.74454122        0.8097732
## 21     GP3  0.634354523    0.84527620        0.8820450
## 22    GP30  0.324817055    0.83339840        0.8445967
## 23    GP31  0.288999655    0.63326218        0.7979835
## 24    GP32  0.434755888    0.86212958        0.8903560
## 25    GP33  0.105014736    0.15428280        0.3001086
## 26    GP34  0.437377321    0.88982108        0.9005346
## 27    GP35  0.142257472    0.22960845        0.5255115
## 28    GP36  0.164219576    0.45894219        0.6578326
## 29    GP37  0.134209867    0.29807577        0.4586853
## 30    GP38  0.255796447    0.50772776        0.6513245
## 31     GP4  0.314891896    0.67443573        0.8083786
## 32     GP5  0.319489847    0.60482762        0.7750068
## 33     GP6  0.295056739    0.67811043        0.8408983
## 34     GP7  0.050544217    0.65244184        0.6871894
## 35     GP8  0.156852770    0.42652308        0.4764526
## 36     GP9  0.070522488    0.07490509        0.2519967
```

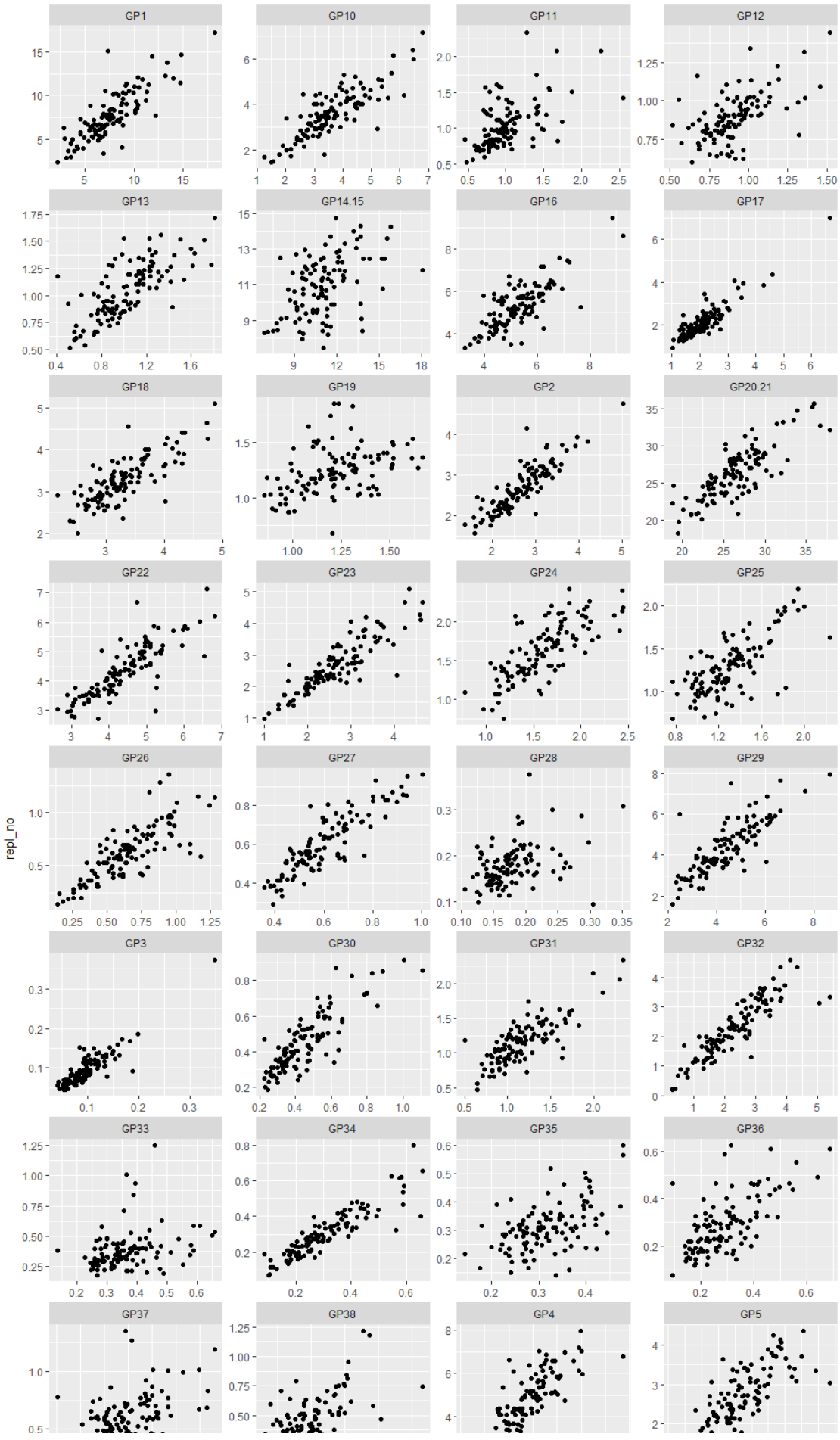
```
# experimental variation
norm_bc_stands_var = glycans_norm_bc %>%
  gather(glycan, level, contains("GP")) %>%
  group_by(glycan, stands) %>%
  summarise(variance = var(log(level)))

norm_bc_stands_var = norm_bc_stands_var %>%
  spread(stands, variance) %>%
  mutate(expVar.norm.bc = 100*stand_yes/stand_no)

stands_var_rawNormBC = merge(stands_var_rawVsNorm, norm_bc_stands_var,
by = "glycan")


data.frame(select(stands_var_rawNormBC, glycan, expVar.raw,
expVar.norm, expVar.norm.bc))
```

```
##     glycan expVar.raw expVar.norm expVar.norm.bc
## 1      GP1   52.26572    6.812304       4.881767
## 2     GP10   81.78910   11.249561       9.626310
## 3     GP11   88.38752   50.971146      41.981448
## 4     GP12  118.83141   41.480516      33.002395
## 5     GP13   78.55759   14.841195       9.674768
## 6  GP14.15  119.33031   62.698293      47.299094
## 7     GP16   98.66729   17.128172      16.897413
## 8     GP17   77.03749   10.118027       9.832281
## 9     GP18  106.15666   11.688611      10.660390
## 10    GP19  112.35290   81.593162      70.007577
## 11     GP2   84.80448    4.565832       3.607144
## 12 GP20.21  110.34916   12.439509       9.338465
## 13    GP22   91.20701    8.583853       9.284685
## 14    GP23   70.25781    5.754030       6.700992
```

```
## 15    GP24  101.76874   24.226019      18.270364
## 16    GP25  108.53049   54.296121      36.068473
## 17    GP26   79.62426   17.364921      13.306782
## 18    GP27   92.95039   12.447082       8.976243
## 19    GP28   97.73799   28.563492      29.130863
## 20    GP29   81.59311   13.494180      13.875738
## 21     GP3   68.40062    7.472540       6.618995
## 22    GP30   66.89891   13.115264      15.141372
## 23    GP31   86.57479   25.600200      22.402050
## 24    GP32   53.78245    6.238206       5.727181
## 25    GP33  102.07653   97.428556      91.269274
## 26    GP34   66.37169    7.785532       6.255156
## 27    GP35  101.61103   49.194818      36.858968
## 28    GP36   94.27512   48.944232      41.793506
## 29    GP37   94.62845   46.507281      47.855908
## 30    GP38   83.26019   36.273458      32.582843
## 31     GP4   72.59347   10.546742       8.759204
## 32     GP5   65.97245   12.166535       6.706478
## 33     GP6   78.39492   11.832175       7.247404
## 34     GP7  122.50185   28.780332      25.629125
## 35     GP8  106.66118   54.290535      47.729384
## 36     GP9   91.28749   66.025555      72.813943
```

Plotting correlation of replicates again.

```
ggplot(gly_repl_norm_bc_l, aes(x = repl_yes, y = repl_no)) +
  geom_point() +
  facet_wrap(~glycan, scales = "free", ncol = 4)
```


GP1
GP10
GP11
GP12
GP13
GP14.15
GP16
GP17
GP18
GP19
GP2
GP20.21
GP22
GP23
GP24
GP25
GP26
GP27
GP28
GP29
GP3
GP30
GP31
GP32
GP33
GP34
GP35
GP36
GP37
GP38
GP4
GP5
repl_no

References

1. Johnson, W. E., Li, C. & Rabinovic, A. Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics (Oxford, England)* **8,** 118–27 (2007).