NEU Proteomics Capstone Module:
Hack-a-thon 2: Build Your Own Data Pipeline

You guys!

Again, everything you need:

bit.ly/PCCSEData

Goal for the hack-a-thon

- Build a pipeline to get from raw MS-based ratios to “processed” data matrices that maximize replicate recall and separation from non-replicates

- Steps to consider:
  - Filtration of data, outlier removal, normalization

- Ways to get there:
  - Use our pipeline but play with parameters
  - Roll your own pipeline
  - Start from raw MS signal (area under curve) and roll your own pipeline

What you will need

- Source data
  - P100_Example_PRM_Data_Unprocessed.gct (ratios)
  - P100_Example_PRM_RawArea_Data_Unprocessed.gct (area)
- A method of reading source data and manipulating source data
  - R, Excel
- A method for writing a “csv” file in a standard format

Evaluation Metrics

- Evaluation of separation of replicates from non-replicates for each team
  - The evaluation script is provided
- Evaluation of same on a held-out data set
- Comparison with what current P100 pipeline would yield

Code Review: The Master Processing Function

```
6  ####################################
7  #                                  #
8  #    MASTER PROCESSING FUNCTIONS   #
9  #                                  #
10 ####################################
11
12 P100processGCTMaster <- function (gctFileName=NULL,repAnnot=NULL, probeAnnot=NULL, dataTable=NULL,
13                                    fileOutput=TRUE,outputFileName=NULL, processMode='full',
14                                    optim=TRUE,log2=TRUE,samplePctCutoff=0.8, probePctCutoff=0.9, probeSDCutoff=3,
15                                    distSDcutoff=5,overrideEmbeddedParameters=FALSE)
16 {
17   ##########################################################################
18   #  This function is the major entry point for automated data processing of P100.     #
19   #  It supports two modes of data input and output.                                   #
20   #  If the parameter 'gctFileName' is not NULL, it will look for a local file of that #
21   #  name.                                                                             #
22   #  If the paramters 'repAnnot', 'probeAnnot', and 'dataTable' are not NULL, it will  #
23   #  assume that these are data objects from Panorama report views and proceed         #
24   #  accordingly.                                                                      #
25   #  Local users andor PANAORAMA should pass these parameters as named arguments!!!    #
26   #  Also, now supporting processMode ('full','quick') switch for less redundancy      #
27   ##########################################################################
28
```

```
>result<-P100processGCTMaster(gctFileName =
   'P100_Example_PRM_Data_Unprocessed.gct', log2=FALSE)
```

- This will run our pipeline with all defaults, generate an output file with the suffix '.processed.gct' in your current working directory

Deeper look at the main pipeline

```
P100processGCT <- function (g,optim=TRUE,log2=TRUE,samplePctCutoff=0.8, probePctCutoff=0.9,
  probeSDCutoff=3, distSDcutoff=3, probeGroupNormalization=FALSE)
{

  static_headers<-g$static_headers;
  surviving_headers<-g$surviving_headers;
  surviving_rowAnnots<-g$surviving_rowAnnots;
  colnames(surviving_rowAnnots)<-static_headers[1,];
  dt<-g$dt;
  #gctFileName=g$gctFileName;

  #log2 transform
  if (log2) {
    dt[dt==0]=NA;
    dt<-log(dt)/log(2);
    surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,"L2X");
  }

  s<-P100filterSamplesForPoorCoverage(dt, pctFilterCutoff=samplePctCutoff)
  surviving_headers<-surviving_headers[,s$colsPassing];
  surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,paste0("SF",round(samplePctCutoff*10,0)));

  #check for explicit probe rejection
  goodProbes<-logical(length=dim(surviving_rowAnnots)[1]);
  goodProbes[]<-TRUE;
  if ('pr_probe_suitability_manual' %in% colnames(surviving_rowAnnots)) {
    goodProbes<-as.logical(surviving_rowAnnots$pr_probe_suitability_manual);
  }
  f<-P100filterProbes(s$filteredData, pctFilterCutoff=probePctCutoff, sdFilterCutoff=probeSDCutoff, explicitRejects=goodProbes);
  surviving_rowAnnots<-surviving_rowAnnots[f$rowsPassing,];
  surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,paste0("PF",round(probePctCutoff*10,0)));
  surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,paste0("PSDF",round(probeSDCutoff,0)));
```

Deeper look at the main pipeline continued

```
  o<-P100optimizeSampleBalance(f$filteredData);

  b<-P100filterOutlierSamplesAndApplyCorrections(dt=f$filteredData,optData=o,sdFilterCutoff=distSDcutoff,optim=optim);
  surviving_headers<-surviving_headers[,b$colsPassing];
  if (optim) {
    surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,"LLB");
  }
  surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,paste0("OF",round(distSDcutoff,0)));

  n<-P100rowMedianNormalize(b$filteredData);
  if (length(unique(surviving_rowAnnots$pr_probe_normalization_group)) > 1 ||
    length(unique(t(surviving_headers['det_normalization_group_vector',]))) > 1) {
    probeGroupNormalization<-TRUE;
  }

  if (probeGroupNormalization) {
    n<-GCPprobeGroupSpecificRowMedianNormalize(data=b$filteredData,ra=surviving_rowAnnots,sth=static_headers,sh=surviving_headers)
    surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,"GMN");
  } else {
    surviving_headers<-.updateProvenanceCode(static_headers,surviving_headers,"RMN");
  }

  q<-list(inputData=dt,initialSampleFiltering=s,probeFiltering=f,optimizationParams=o,secondarySampleCorrectionAndFiltering=b,
          normalizedData=n,outputData=n$normalizedData,static_headers=static_headers,surviving_headers=surviving_headers,
          surviving_rowAnnots=surviving_rowAnnots,colsAnnot=g$colsAnnot,rowsAnnot=g$rowsAnnot);
  return(q);
}
```

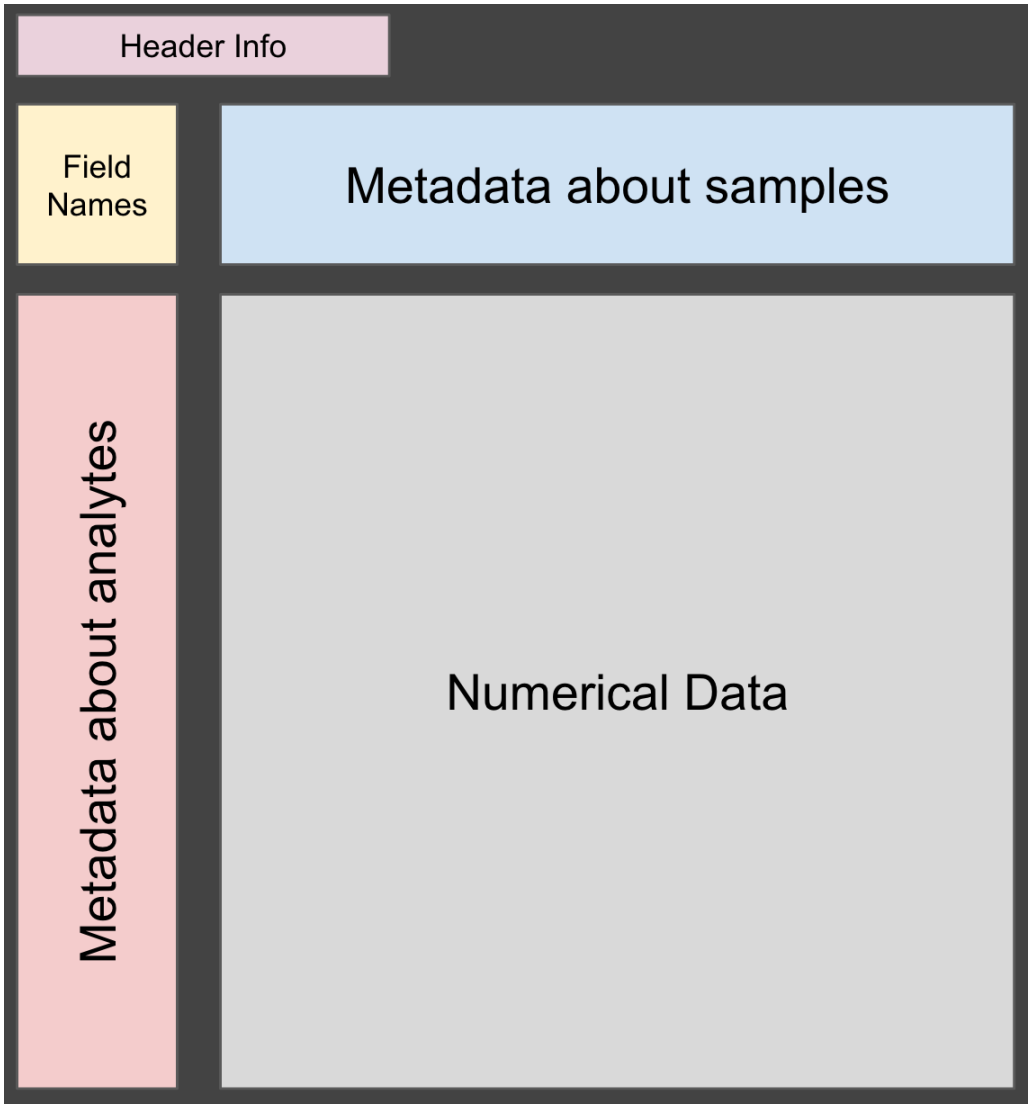
Pipeline step function convention

- Most functions return a list object with the form:

```
{object}$originalData
{object}$filteredData
{object}$colsPassing –or– {object}$rowsPassing
```

- There are some minor variants

Reminder about data file format: GCT



- This is a tab-delimited text file; you can drag/drop into Excel

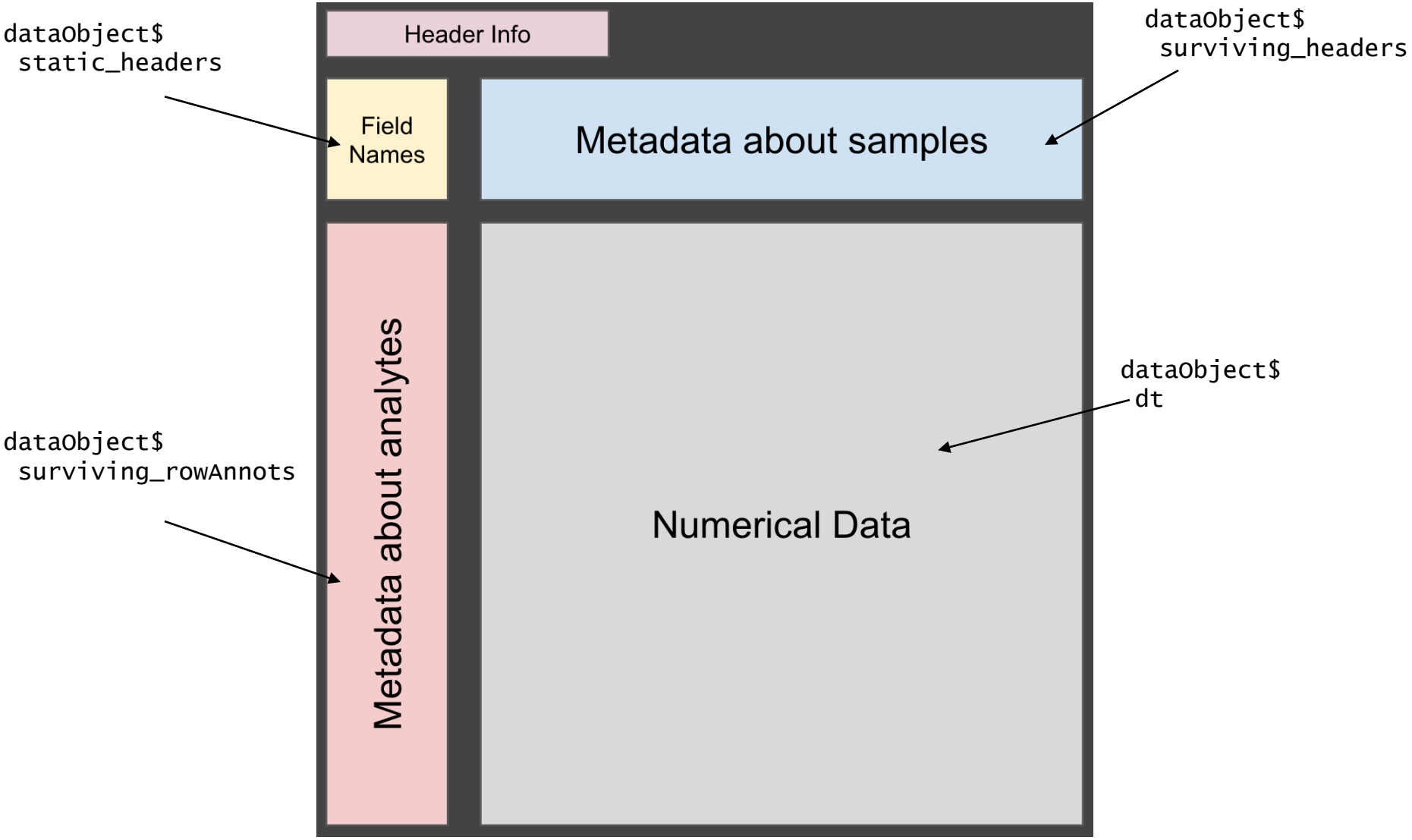
Reading the data using the P100 code-base in R

```
install.packages('jsonlite')

source('p100_processing.R')

dataObject<-
P100provideGCTlistObjectFromFile('p100_sourcedata_….gct')
```

Reading the data using the P100 code-base in R



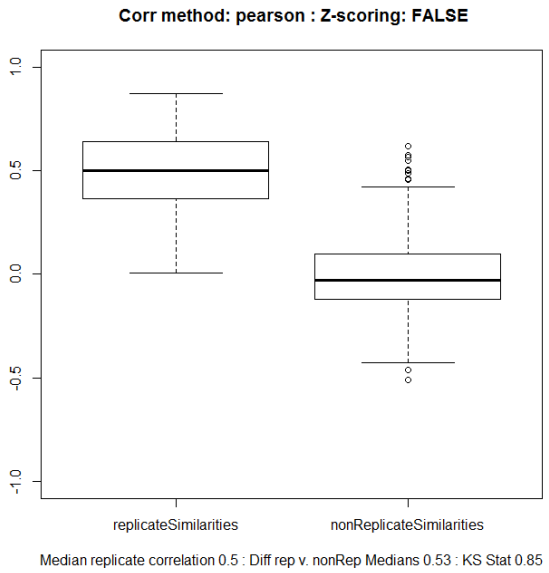
Output format (.csv file)

	PP3-3D167-001A01	PP3-3D167-002A02	PP3-3D167-003A03	PP3-3D167-004A04
10011_DYRK_Y321_IYQY[+80]IQSR	-0.166	-0.008	0.397	-0.210
1024_ISPK1_S369_TPKDS[+80]PGIPPSANAHQLFR	0.204	0.109	-0.106	0.330
1078_ARM2_S87_RNS[+80]SEASSGDFLDLK	0.128	0.085	0.285	-0.477
1130_HSPC216_S321_LPLVPES[+80]PRR	0.042	0.102	-0.003	-0.059
1142_CTG26_S956_ANAS[+80]PQKPLDLK	0.086	0.114	-0.032	0.027
12_KIAA0701_S935_SMS[+80]VDLSHIPLKDPLLFK	0.322	-0.102	-0.408	-0.202
1458_OCIA_S108_LENS[+80]PLGEALR	0.609	0.338	0.288	0.563
1503_PDK1_S241_ANS[+80]FVGTAQYVSPELLTEK	0.013	0.101	0.192	0.035
151_ABI1_S184_TNPPTQKPPS[+80]PPMSGR	-0.171	-0.077	-0.104	-0.378
1797_WDR20_S465_SNS[+80]LPHSAVSNAGSK	-0.096	0.080	0.156	-0.259
1811_MAP4_S2218_VGS[+80]LDNVGHLPAGGAVK	0.195	0.005	-0.222	0.000
2328_IQGAP3_S1424_S[+80]LTAHSLLPLAEK	0.833	0.843	0.742	0.288
2577_C9orf88_S692_AAPEAS[+80]SPPASPLQHLLPGK	-0.006	-0.435	-0.561	0.023
290_TMSL3_S2_S[+122]DKPDM[+16]AEIEKFDK	0.475	0.090	-0.560	0.273
290_TMSL3_S2_S[+122]DKPDMAEIEKFDK	0.027	0.108	-0.038	0.072

Evaluating your progress:

```
> source('../../code/p100_processing.R')
Loading required package: jsonlite
> source('../../code/NEU_Capstone_Evaluation_Functions.R')
> output<-P100replicateSimilarityEvaluatorFromGCT('../P100_Example_PRM_Data_Processed.gct')
[1] "Reading first file...\n"
> summary(output)
                         Length Class  Mode
replicateSimilarities     31    -none- numeric
nonReplicateSimilarities 496    -none- numeric
medianDiff                 1    -none- numeric
ksstat                     1    -none- numeric
replicateMedian            1    -none- numeric
> output$medianDiff
[1] 0.5290405
> output$replicateMedian
[1] 0.4986485
> output$ksstat
        D
0.8467742
> output_from_csv_version<-P100replicateSimilarityEvaluatorFromCSV('../P100_Example_PRM_Data_Processed.csv')
[1] "Reading first file...\n"
```

CRITICAL N.B. for csv'ers: keep "P100_Example_PRM_Data_Unprocessed.gct" in your working directory or specify a path to it with the parameter 'basegct=…' in the function call.
It allows sample ID to metadata mapping.



Options (first one is default):
similarity_method ('pearson','spearman')
zscore (FALSE,TRUE)

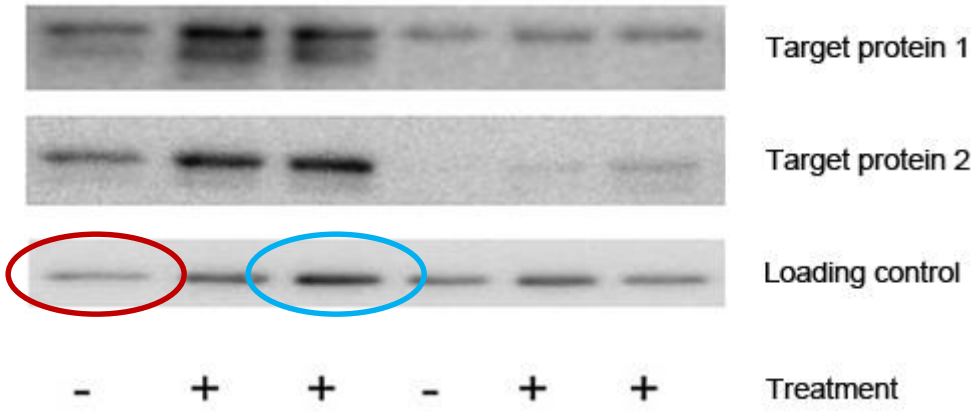
Example call:

```
output<-P100replicateSimilarityEvaluatorFromGCT
 ('../P100_Example_PRM_Data_Processed.gct',
  zscore=TRUE, similarity_method='spearman')
```

EXTRA SLIDES

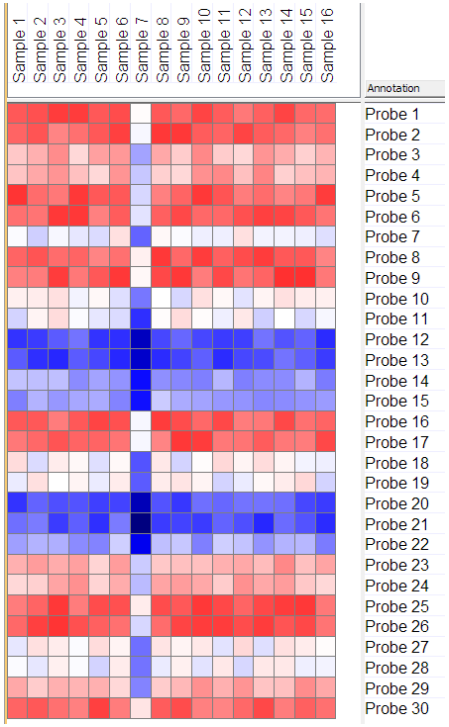
A word on “Load Balancing” in P100

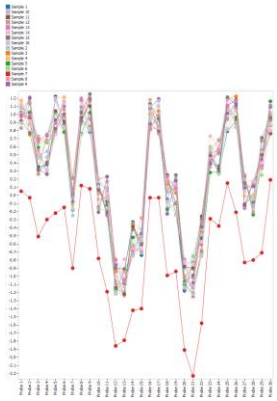
Typical western blot:



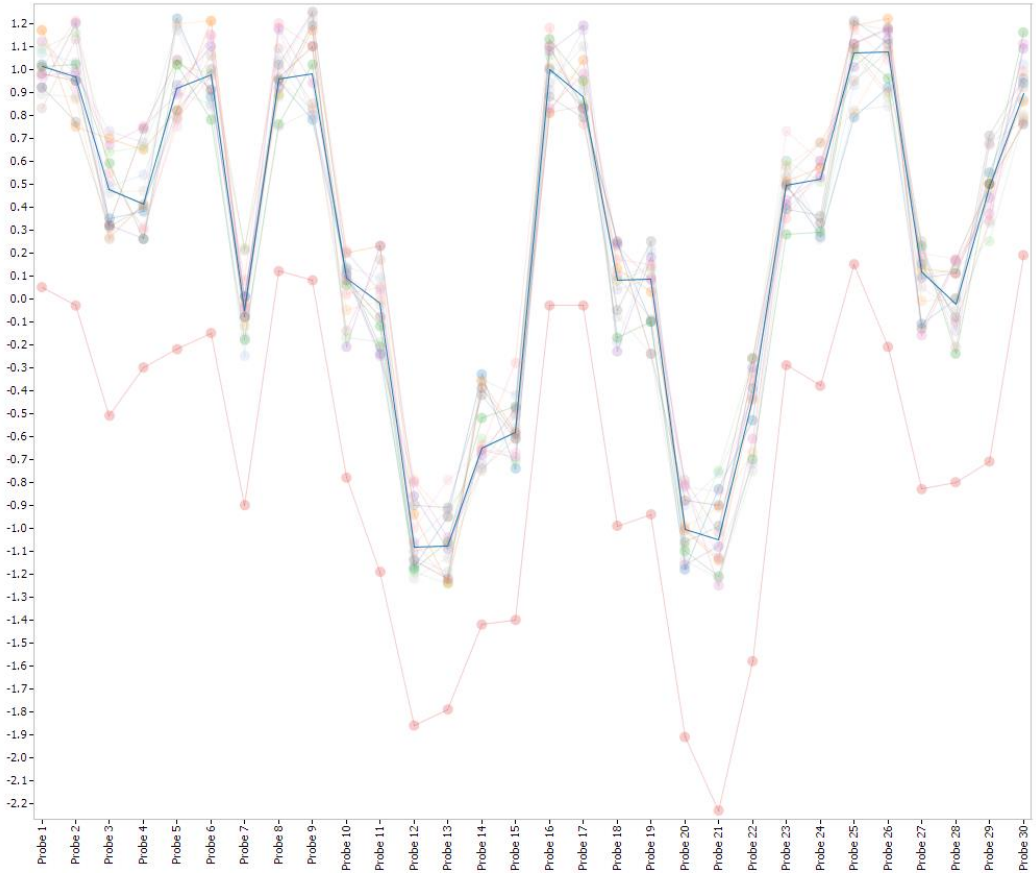
credit: BioRad website

An analogous situation in proteomics data…

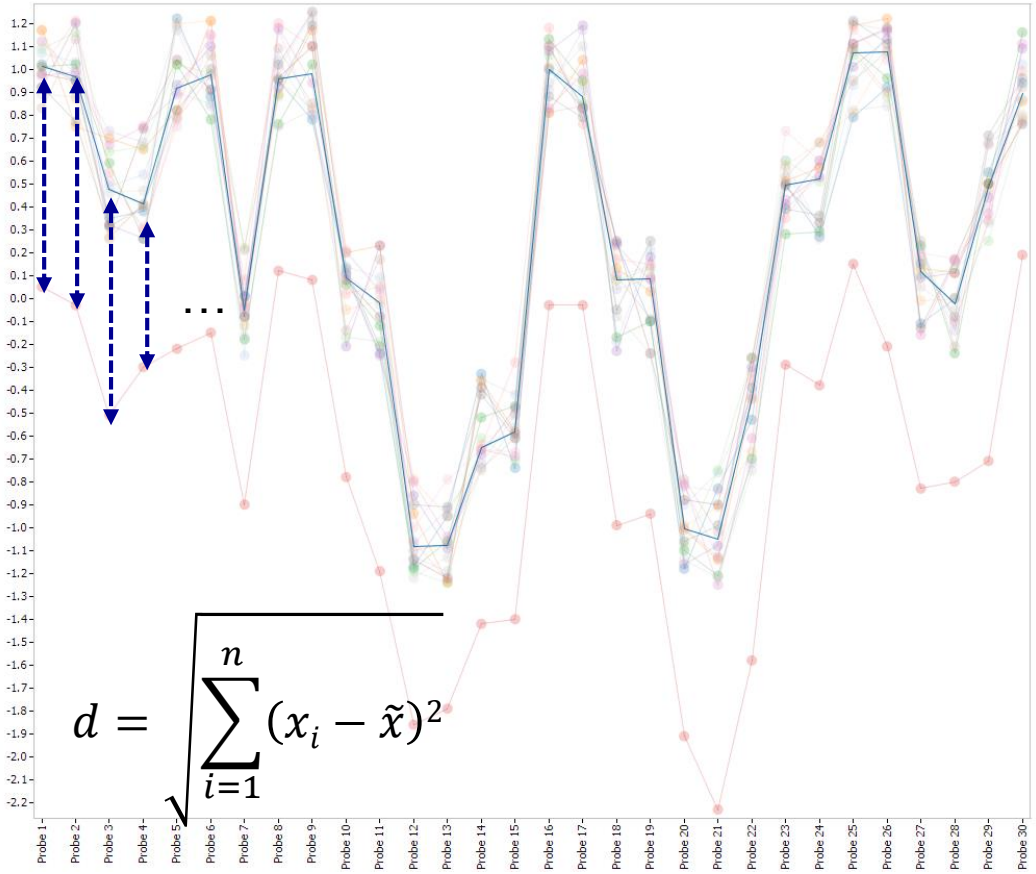




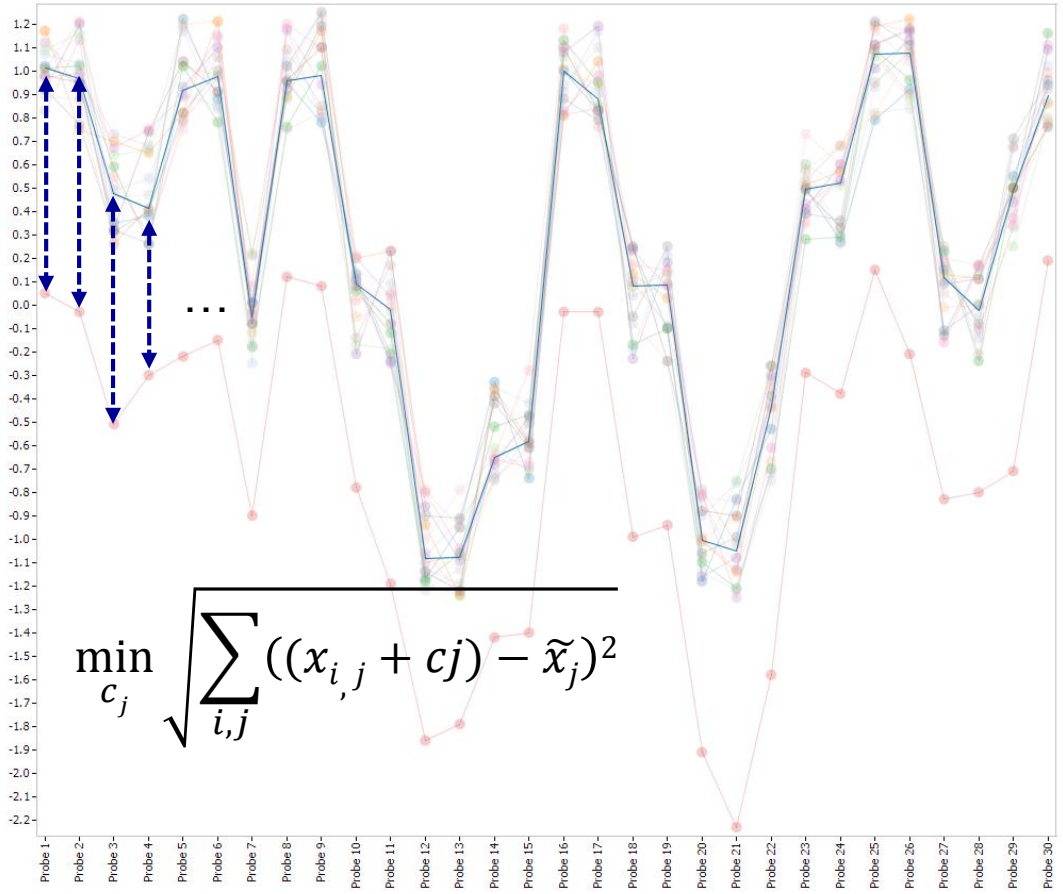
Find Median Profile



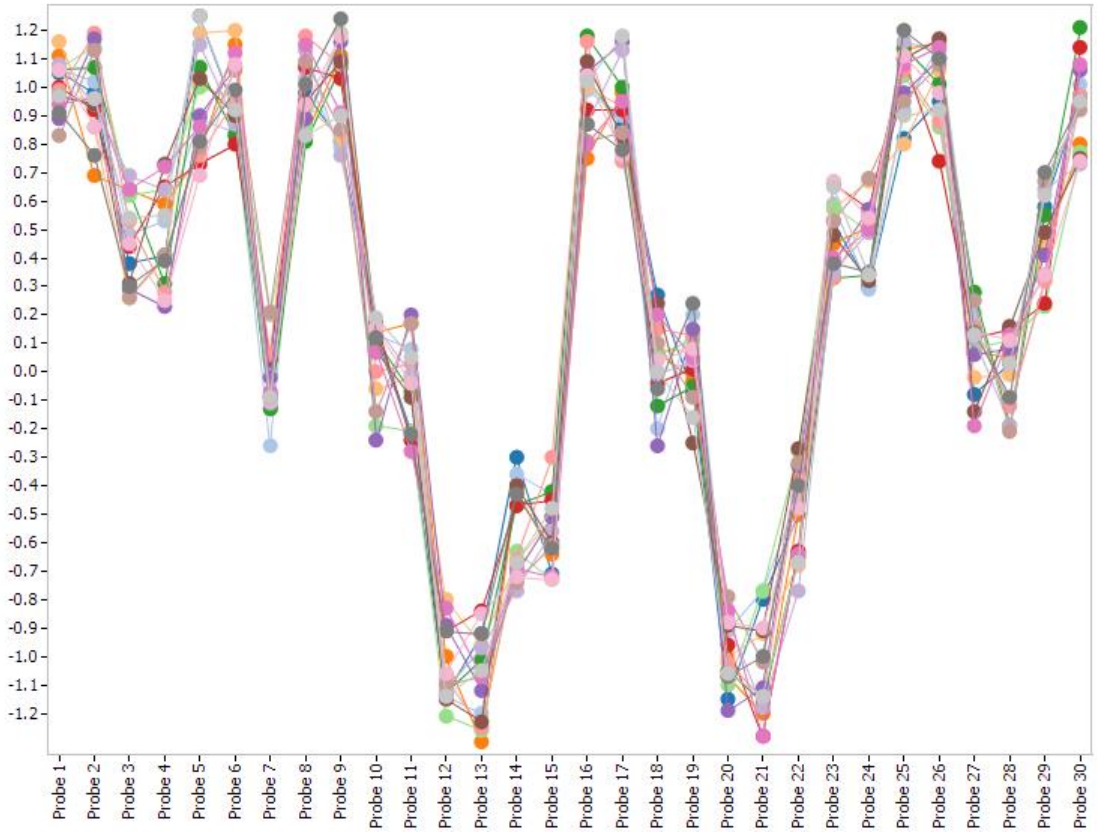
Compute distance from each profile to median

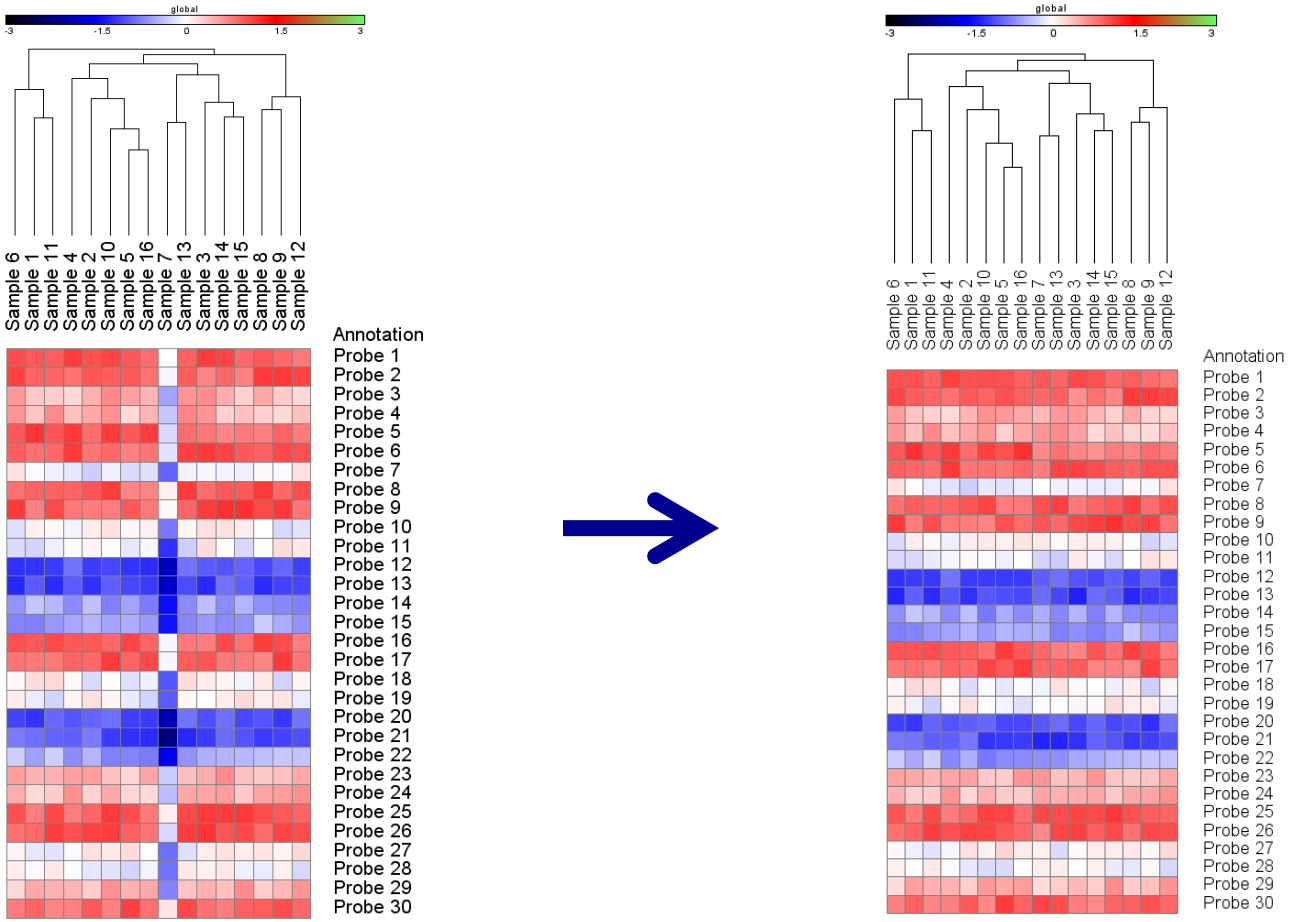


Find a constant to minimize distance for each sample

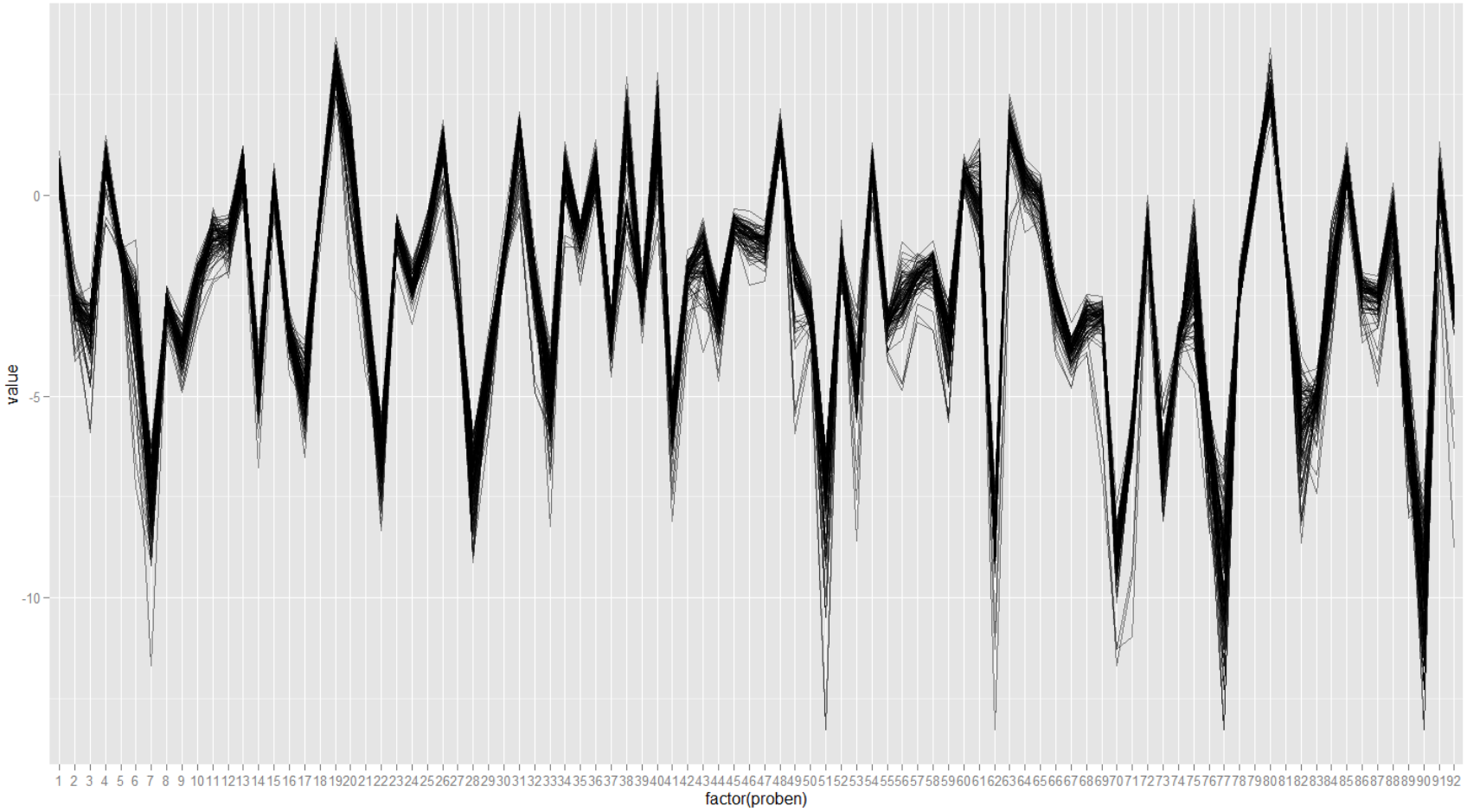


Apply corrections

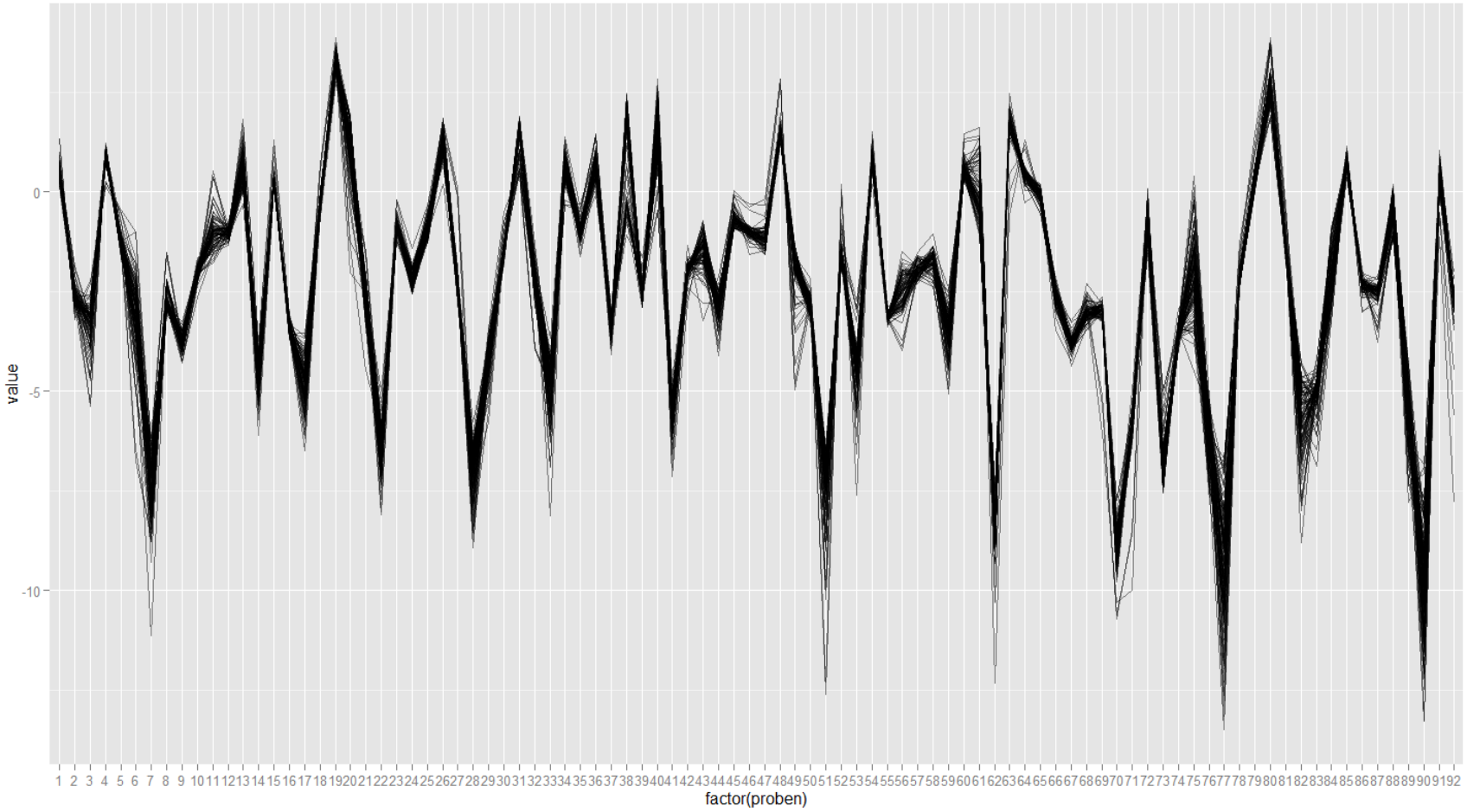



global
-3
-1.5
0
1.5
3
Sample 6
Sample 1
Sample 11
Sample 4
Sample 2
Sample 10
Sample 5
Sample 16
Sample 7
Sample 13
Sample 3
Sample 14
Sample 15
Sample 8
Sample 9
Sample 12
Annotation
Probe 1
Probe 2
Probe 3
Probe 4
Probe 5
Probe 6
Probe 7
Probe 8
Probe 9
Probe 10
Probe 11
Probe 12
Probe 13
Probe 14
Probe 15
Probe 16
Probe 17
Probe 18
Probe 19
Probe 20
Probe 21
Probe 22
Probe 23
Probe 24
Probe 25
Probe 26
Probe 27
Probe 28
Probe 29
Probe 30

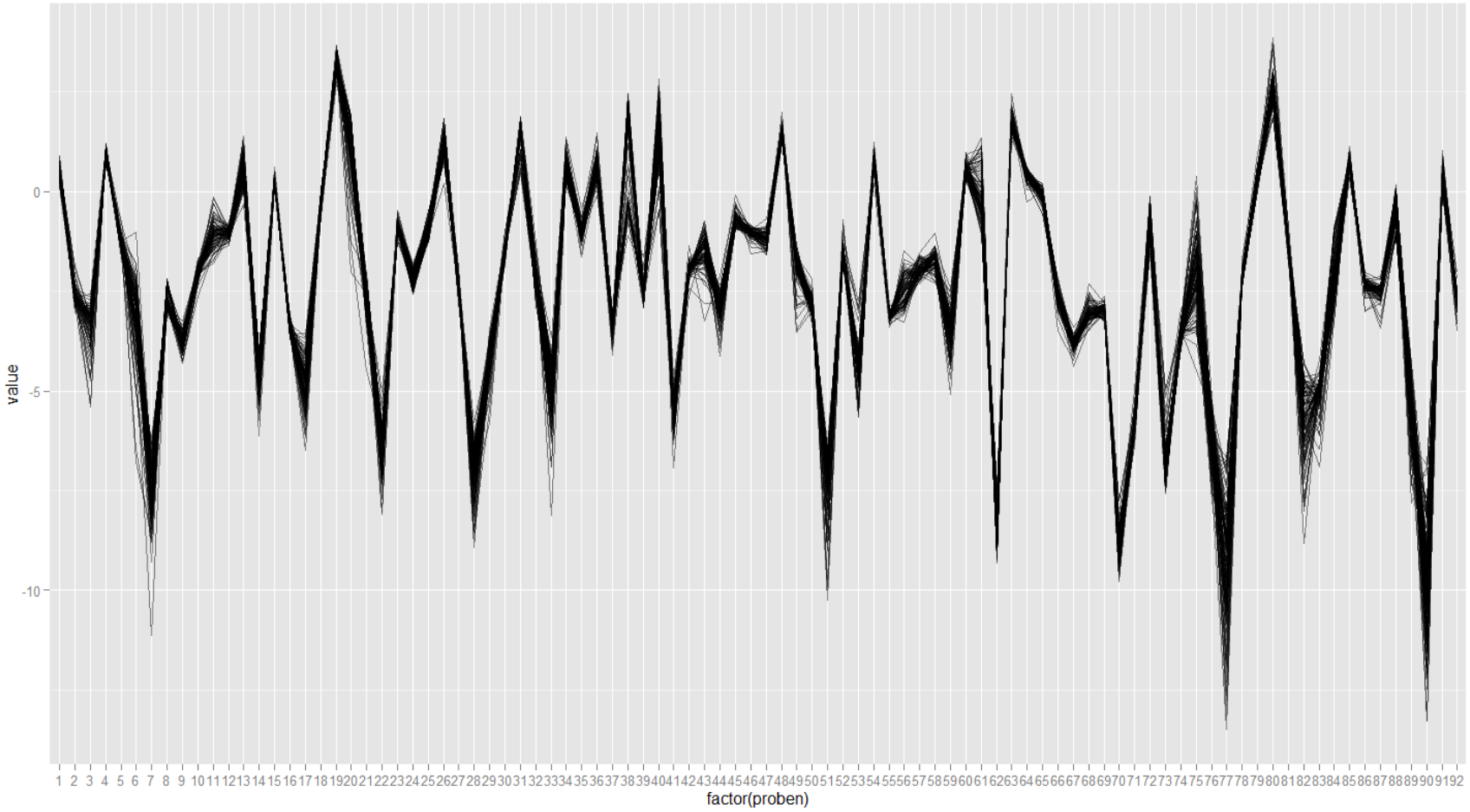
Original data (A375 cells, 95 samples x 92 probes)



Load balanced data (A375 cells, 95 samples x 92 probes)



Load balanced + outliers removed data (A375 cells, 92 samples x 92 probes)



Row normalized (A375 cells, 92 samples x 92 probes)