Mouse Protein Expression Analysis

YNKM

10/13/2022

Contents

Installing and Loading required packages

```
options(repos = c(CRAN = "https://cran.rstudio.com"))
packages <- c("readxl", "tidyverse", "reshape2", "psych", "corrplot", "randomForest",
              "e1071", "caret", "neuralnet", "nnet", "NeuralSens","ipred",
              "DataExplorer", "ggplot2", "FactoMineR", "factoextra", "devtools",
              "ggbiplot", "pROC", "broom", "ape", "dendextend")

# readxl: for reading excel data file
# tidyverse: for general data utility functions
# psych: for pairs panel plot
# corrplot: for correlation matrix
# e1071: for Naive Bayes
# caret: for data preparation and evaluation of model
# neuralnet: for NN modeling
```

```r
# nnet: for multinomial logistic regression
# ipred: for bagging
# randomForest: for stacked learner and variable importance
# DataExplorer and ggplot2: for plots
# FactoMineR, factoextra, devtools, ggbiplot: for PCA plots
# pROC: for ROC and AUC plot

# Install packages not yet installed
installed_packages <- packages %in% rownames(installed.packages())
if (any(installed_packages == FALSE)) {
  install.packages(packages[!installed_packages])
}

# Packages loading
invisible(lapply(packages, library, character.only = TRUE))
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.1.8     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 1.0.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
##
## Attaching package: 'reshape2'
##
##
## The following object is masked from 'package:tidyr':
##
##     smiths
##
##
##
## Attaching package: 'psych'
##
##
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
##
##
## corrplot 0.92 loaded
##
## randomForest 4.7-1.1
##
## Type rfNews() to see new features/changes/bug fixes.
##
##
## Attaching package: 'randomForest'
##
##
## The following object is masked from 'package:psych':
```

```
##
##     outlier
##
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
##
## The following object is masked from 'package:ggplot2':
##
##     margin
##
##
## Loading required package: lattice
##
##
## Attaching package: 'caret'
##
##
## The following object is masked from 'package:purrr':
##
##     lift
##
##
##
## Attaching package: 'neuralnet'
##
##
## The following object is masked from 'package:dplyr':
##
##     compute
##
##
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
##
## Loading required package: usethis
##
## Loading required package: plyr
##
## ------------------------------------------------------------------------------
##
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
##
## ------------------------------------------------------------------------------
##
##
## Attaching package: 'plyr'
##
##
## The following objects are masked from 'package:dplyr':
##
```

```
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
##
##
## The following object is masked from 'package:purrr':
##
##     compact
##
##
## Loading required package: scales
##
##
## Attaching package: 'scales'
##
##
## The following objects are masked from 'package:psych':
##
##     alpha, rescale
##
##
## The following object is masked from 'package:purrr':
##
##     discard
##
##
## The following object is masked from 'package:readr':
##
##     col_factor
##
##
## Loading required package: grid
##
## Type 'citation("pROC")' for a citation.
##
##
## Attaching package: 'pROC'
##
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
##
##
##
## Attaching package: 'ape'
##
##
## The following object is masked from 'package:dplyr':
##
##     where
##
##
##
## ----------------------
```

```
## Welcome to dendextend version 1.16.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ----------------------
##
##
##
## Attaching package: 'dendextend'
##
##
## The following objects are masked from 'package:ape':
##
##     ladderize, rotate
##
##
## The following object is masked from 'package:stats':
##
##     cutree
```

Understanding the data

[1] **Mouse ID**

[2:78] **Values of expression levels of 77 proteins**; the names of proteins are followed by N indicating that they were measured in the nuclear fraction. *For example: DYRK1A_n*

[79] **Genotype**: control (c) or trisomy (t)

[80] **Treatment type**: memantine (m) or saline (s)

[81] **Behavior**: context-shock (CS) or shock-context (SC)

[82] **Class**: c-CS-s, c-CS-m, c-SC-s, c-SC-m, t-CS-s, t-CS-m, t-SC-s, t-SC-m

Download, Load and Investigate the expression dataset

```
# define the URL -- you could dynamically build this
URL <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00342/Data_Cortex_Nuclear.xls"
# download file
download.file(URL, destfile="Data_Cortex_Nuclear.xls")
# load the file
ncortex.raw <- as.data.frame(read_excel("Data_Cortex_Nuclear.xls"))
head(ncortex.raw)
```

```
dim(ncortex.raw)
```

```
## [1] 1080   82
```

```
str(ncortex.raw)
```

```
## 'data.frame':    1080 obs. of  82 variables:
##  $ MouseID       : chr  "309_1" "309_2" "309_3" "309_4" ...
##  $ DYRK1A_N      : num  0.504 0.515 0.509 0.442 0.435 ...
##  $ ITSN1_N       : num  0.747 0.689 0.73 0.617 0.617 ...
##  $ BDNF_N        : num  0.43 0.412 0.418 0.359 0.359 ...
##  $ NR1_N         : num  2.82 2.79 2.69 2.47 2.37 ...
##  $ NR2A_N        : num  5.99 5.69 5.62 4.98 4.72 ...
##  $ pAKT_N        : num  0.219 0.212 0.209 0.223 0.213 ...
##  $ pBRAF_N       : num  0.178 0.173 0.176 0.176 0.174 ...
##  $ pCAMKII_N     : num  2.37 2.29 2.28 2.15 2.13 ...
##  $ pCREB_N       : num  0.232 0.227 0.23 0.207 0.192 ...
##  $ pELK_N        : num  1.75 1.6 1.56 1.6 1.5 ...
##  $ pERK_N        : num  0.688 0.695 0.677 0.583 0.551 ...
##  $ pJNK_N        : num  0.306 0.299 0.291 0.297 0.287 ...
##  $ PKCA_N        : num  0.403 0.386 0.381 0.377 0.364 ...
##  $ pMEK_N        : num  0.297 0.281 0.282 0.314 0.278 ...
##  $ pNR1_N        : num  1.022 0.957 1.004 0.875 0.865 ...
##  $ pNR2A_N       : num  0.606 0.588 0.602 0.52 0.508 ...
##  $ pNR2B_N       : num  1.88 1.73 1.73 1.57 1.48 ...
##  $ pPKCAB_N      : num  2.31 2.04 2.02 2.13 2.01 ...
##  $ pRSK_N        : num  0.442 0.445 0.468 0.478 0.483 ...
##  $ AKT_N         : num  0.859 0.835 0.814 0.728 0.688 ...
##  $ BRAF_N        : num  0.416 0.4 0.4 0.386 0.368 ...
##  $ CAMKII_N      : num  0.37 0.356 0.368 0.363 0.355 ...
##  $ CREB_N        : num  0.179 0.174 0.174 0.179 0.175 ...
##  $ ELK_N         : num  1.87 1.76 1.77 1.29 1.32 ...
##  $ ERK_N         : num  3.69 3.49 3.57 2.97 2.9 ...
##  $ GSK3B_N       : num  1.54 1.51 1.5 1.42 1.36 ...
##  $ JNK_N         : num  0.265 0.256 0.26 0.26 0.251 ...
##  $ MEK_N         : num  0.32 0.304 0.312 0.279 0.274 ...
##  $ TRKA_N        : num  0.814 0.781 0.785 0.734 0.703 ...
##  $ RSK_N         : num  0.166 0.157 0.161 0.162 0.155 ...
##  $ APP_N         : num  0.454 0.431 0.423 0.411 0.399 ...
##  $ Bcatenin_N    : num  3.04 2.92 2.94 2.5 2.46 ...
##  $ SOD1_N        : num  0.37 0.342 0.344 0.345 0.329 ...
##  $ MTOR_N        : num  0.459 0.424 0.425 0.429 0.409 ...
##  $ P38_N         : num  0.335 0.325 0.325 0.33 0.313 ...
##  $ pMTOR_N       : num  0.825 0.762 0.757 0.747 0.692 ...
##  $ DSCR1_N       : num  0.577 0.545 0.544 0.547 0.537 ...
##  $ AMPKA_N       : num  0.448 0.421 0.405 0.387 0.361 ...
##  $ NR2B_N        : num  0.586 0.545 0.553 0.548 0.513 ...
##  $ pNUMB_N       : num  0.395 0.368 0.364 0.367 0.352 ...
##  $ RAPTOR_N      : num  0.34 0.322 0.313 0.328 0.312 ...
##  $ TIAM1_N       : num  0.483 0.455 0.447 0.443 0.419 ...
##  $ pP70S6_N      : num  0.294 0.276 0.257 0.399 0.393 ...
##  $ NUMB_N        : num  0.182 0.182 0.184 0.162 0.16 ...
##  $ P70S6_N       : num  0.843 0.848 0.856 0.76 0.768 ...
```

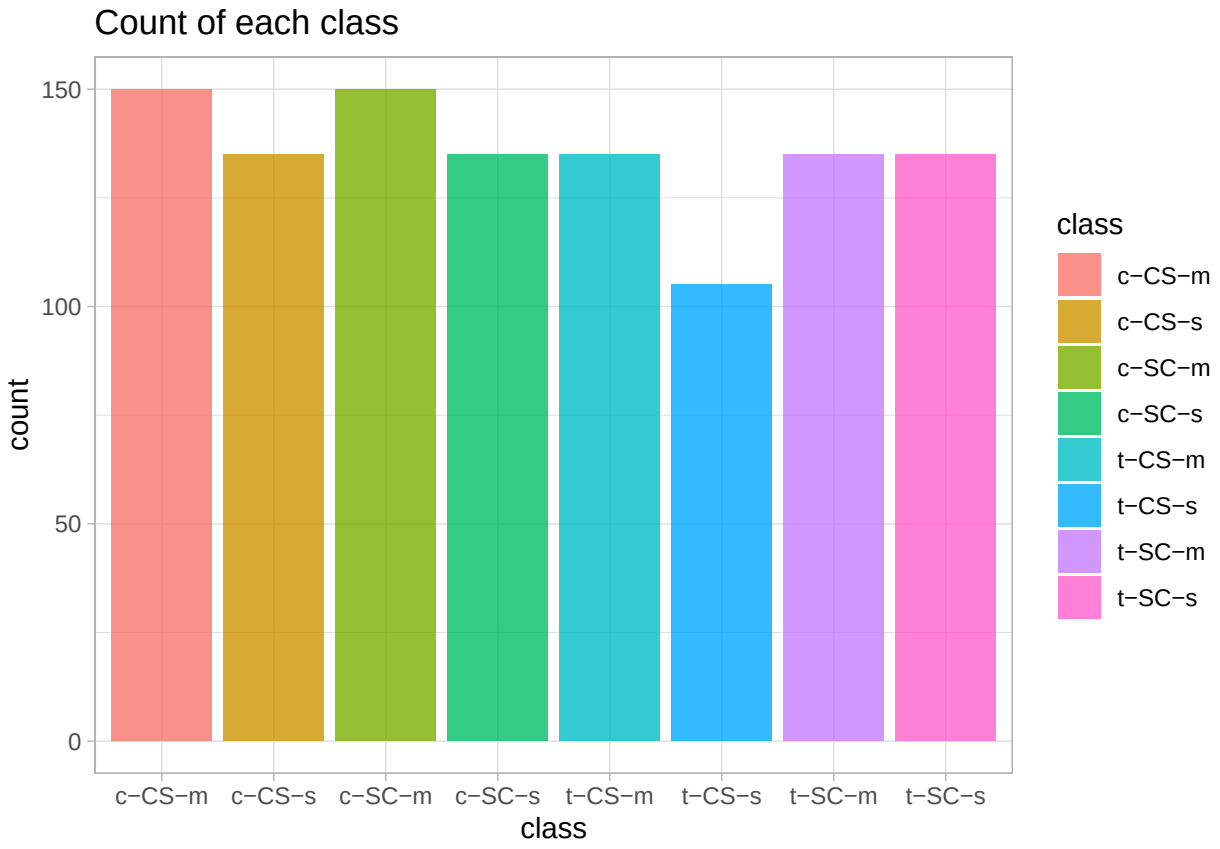
```
##  $ pGSK3B_N       : num  0.193 0.195 0.201 0.184 0.186 ...
##  $ pPKCG_N        : num  1.44 1.44 1.52 1.61 1.65 ...
##  $ CDK5_N         : num  0.295 0.294 0.302 0.296 0.297 ...
##  $ S6_N           : num  0.355 0.355 0.386 0.291 0.309 ...
##  $ ADARB1_N       : num  1.34 1.31 1.28 1.2 1.21 ...
##  $ AcetylH3K9_N   : num  0.17 0.171 0.185 0.16 0.165 ...
##  $ RRP1_N         : num  0.159 0.158 0.149 0.166 0.161 ...
##  $ BAX_N          : num  0.189 0.185 0.191 0.185 0.188 ...
##  $ ARC_N          : num  0.106 0.107 0.108 0.103 0.105 ...
##  $ ERBB4_N        : num  0.145 0.15 0.145 0.141 0.142 ...
##  $ nNOS_N         : num  0.177 0.178 0.176 0.164 0.168 ...
##  $ Tau_N          : num  0.125 0.134 0.133 0.123 0.137 ...
##  $ GFAP_N         : num  0.115 0.118 0.118 0.117 0.116 ...
##  $ GluR3_N        : num  0.228 0.238 0.245 0.235 0.256 ...
##  $ GluR4_N        : num  0.143 0.142 0.142 0.145 0.141 ...
##  $ IL1B_N         : num  0.431 0.457 0.51 0.431 0.481 ...
##  $ P3525_N        : num  0.248 0.258 0.255 0.251 0.252 ...
##  $ pCASP9_N       : num  1.6 1.67 1.66 1.48 1.53 ...
##  $ PSD95_N        : num  2.01 2 2.02 1.96 2.01 ...
##  $ SNCA_N         : num  0.108 0.11 0.108 0.12 0.12 ...
##  $ Ubiquitin_N    : num  1.045 1.01 0.997 0.99 0.998 ...
##  $ pGSK3B_Tyr216_N: num  0.832 0.849 0.847 0.833 0.879 ...
##  $ SHH_N          : num  0.189 0.2 0.194 0.192 0.206 ...
##  $ BAD_N          : num  0.123 0.117 0.119 0.133 0.13 ...
##  $ BCL2_N         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ pS6_N          : num  0.106 0.107 0.108 0.103 0.105 ...
##  $ pCFOS_N        : num  0.108 0.104 0.106 0.111 0.111 ...
##  $ SYP_N          : num  0.427 0.442 0.436 0.392 0.434 ...
##  $ H3AcK18_N      : num  0.115 0.112 0.112 0.13 0.118 ...
##  $ EGR1_N         : num  0.132 0.135 0.133 0.147 0.14 ...
##  $ H3MeK4_N       : num  0.128 0.131 0.127 0.147 0.148 ...
##  $ CaNA_N         : num  1.68 1.74 1.93 1.7 1.84 ...
##  $ Genotype       : chr  "Control" "Control" "Control" "Control" ...
##  $ Treatment      : chr  "Memantine" "Memantine" "Memantine" "Memantine" ...
##  $ Behavior       : chr  "C/S" "C/S" "C/S" "C/S" ...
##  $ class          : chr  "c-CS-m" "c-CS-m" "c-CS-m" "c-CS-m" ...
```

Data Manipulation and Cleaning

Outcome classes

1. **c-CS-s**: control mice, stimulated to learn, injected with saline (9 mice)
2. **c-CS-m**: control mice, stimulated to learn, injected with memantine (10 mice)
3. **c-SC-s**: control mice, not stimulated to learn, injected with saline (9 mice)
4. **c-SC-m**: control mice, not stimulated to learn, injected with memantine (10 mice)
5. **t-CS-s**: trisomy mice, stimulated to learn, injected with saline (7 mice)
6. **t-CS-m**: trisomy mice, stimulated to learn, injected with memantine (9 mice)
7. **t-SC-s**: trisomy mice, not stimulated to learn, injected with saline (9 mice)
8. **t-SC-m**: trisomy mice, not stimulated to learn, injected with memantine (9 mice)

```
ggplot(ncortex.raw, aes(class)) +
  geom_bar(aes(fill = class), alpha = 0.8) +
  theme_light() +
  labs(title = "Count of each class")
```



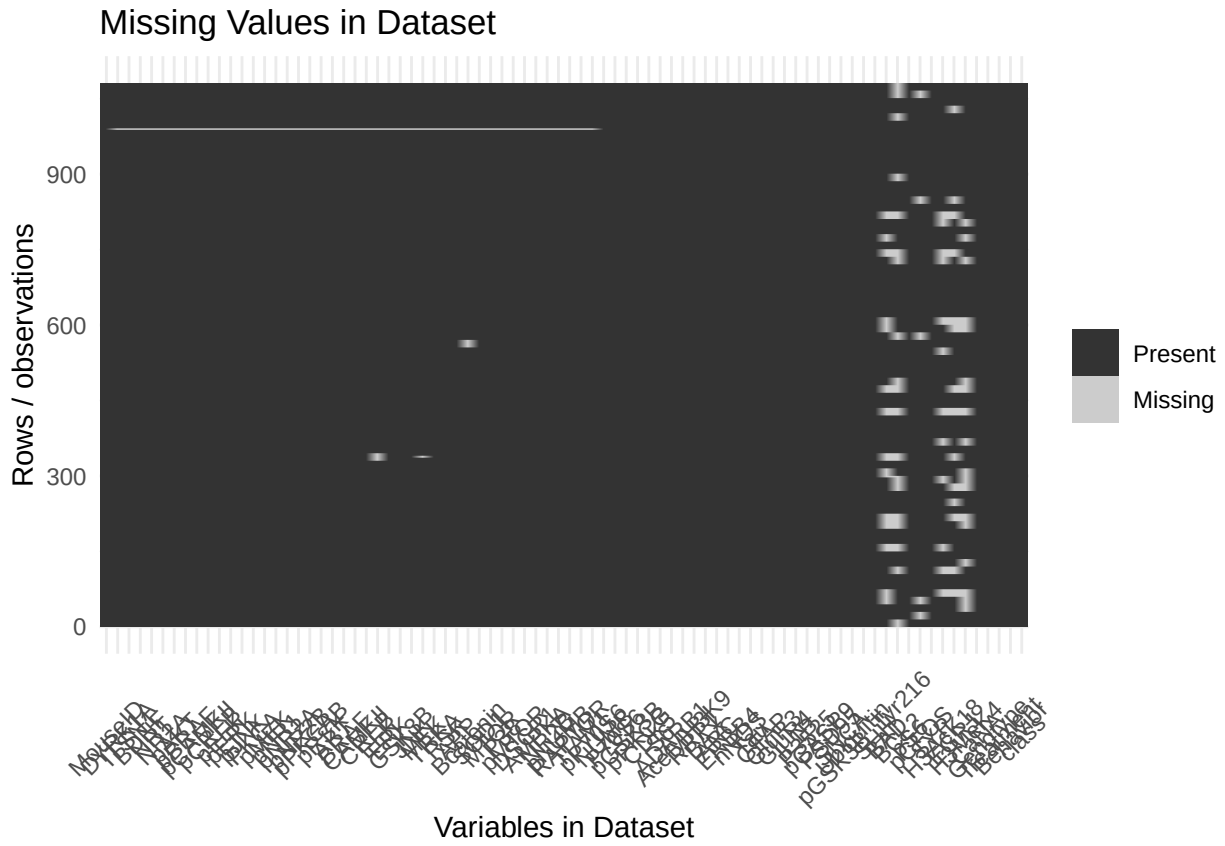
Data Manipulation & Cleaning

```
# since we are focusing our analysis on the `class` variable, we are going to drop
# the rest of the categorical variables from the dataNC along with the `MouseID`column
ncortex <- subset(ncortex.raw, select = -c(MouseID, Behavior, Genotype, Treatment))
# We have the numeric features (77) and a multi-class target variable (1)
ncortex$class <- as.factor(ncortex.raw$class)

proteins <- names(ncortex[1:77]) #creating list of protein names
classes <- ncortex$class
```

Imputation of missing values

```
names(ncortex.raw) <- gsub("_N", "", names(ncortex.raw)) #removing _N in protein names

#check for missing values
ncortex.raw %>%
  is.na %>%
```

```
  melt %>%
  ggplot(data = .,
         aes(x = Var2,
             y = Var1)) +
  geom_raster(aes(fill = value)) +
  scale_fill_grey(name = "",
                  labels = c("Present","Missing")) +
  theme_minimal() +
  theme(axis.text.x  = element_text(angle=45, vjust=0.5)) +
  labels(x = "Variables in Dataset",
       y = "Rows / observations",
       title = "Missing Values in Dataset")
```




```
# check the columns with NAs
featNA <- names(which(sapply(ncortex, anyNA)))
dataNA <- ncortex[, featNA]

# total number of NAs
sum(is.na(dataNA))
```

```
## [1] 1396
```

```
# check the distribution of NAs in each column
colSums(is.na(dataNA))
```

```
##   DYRK1A_N    ITSN1_N     BDNF_N      NR1_N     NR2A_N     pAKT_N    pBRAF_N
##          3          3          3          3          3          3          3
##  pCAMKII_N    pCREB_N     pELK_N     pERK_N     pJNK_N     PKCA_N     pMEK_N
##          3          3          3          3          3          3          3
##     pNR1_N    pNR2A_N    pNR2B_N   pPKCAB_N     pRSK_N      AKT_N     BRAF_N
##          3          3          3          3          3          3          3
##   CAMKII_N     CREB_N      ELK_N      ERK_N    GSK3B_N      JNK_N      MEK_N
##          3          3         18          3          3          3          7
##     TRKA_N      RSK_N      APP_N Bcatenin_N     SOD1_N     MTOR_N      P38_N
##          3          3          3         18          3          3          3
##    pMTOR_N    DSCR1_N    AMPKA_N     NR2B_N    pNUMB_N   RAPTOR_N    TIAM1_N
##          3          3          3          3          3          3          3
##   pP70S6_N      BAD_N     BCL2_N    pCFOS_N  H3AcK18_N     EGR1_N   H3MeK4_N
##          3        213        285         75        180        210        270
```

```
# all columns have NAs below 50% of the sample size. Hence, they can be imputed.
# computing mean of all columns using colMeans()
means <- colMeans(dataNA, na.rm = TRUE)

# replacing NA with mean value of each column
for(i in colnames(dataNA)){
  dataNA[,i][is.na(dataNA[,i])] <- means[i]
}

# check missing values in our feature dataNC frame
sum(is.na(dataNA))
```

```
## [1] 0
```

```
# replace the imputed features with the original dataNC
ncortex[,featNA] <- dataNA

# get numeric feature
nc.feat <- ncortex[,-ncol(ncortex)]
# get target variables
nc.Class <- ncortex$class

# check basic statistics
summary(ncortex)
```

```
##     DYRK1A_N         ITSN1_N           BDNF_N           NR1_N
##  Min.   :0.1453   Min.   :0.2454   Min.   :0.1152   Min.   :1.331
##  1st Qu.:0.2882   1st Qu.:0.4737   1st Qu.:0.2877   1st Qu.:2.059
##  Median :0.3665   Median :0.5664   Median :0.3167   Median :2.297
##  Mean   :0.4258   Mean   :0.6171   Mean   :0.3191   Mean   :2.297
##  3rd Qu.:0.4876   3rd Qu.:0.6975   3rd Qu.:0.3480   3rd Qu.:2.528
##  Max.   :2.5164   Max.   :2.6027   Max.   :0.4972   Max.   :3.758
##
##      NR2A_N           pAKT_N           pBRAF_N          pCAMKII_N
##  Min.   :1.738   Min.   :0.06324   Min.   :0.06404   Min.   :1.344
##  1st Qu.:3.160   1st Qu.:0.20582   1st Qu.:0.16462   1st Qu.:2.480
##  Median :3.763   Median :0.23125   Median :0.18227   Median :3.330
##  Mean   :3.844   Mean   :0.23317   Mean   :0.18185   Mean   :3.537
```

```
##  3rd Qu.:4.425   3rd Qu.:0.25722   3rd Qu.:0.19723   3rd Qu.:4.481
##  Max.   :8.483   Max.   :0.53905   Max.   :0.31707   Max.   :7.464
##
##     pCREB_N           pELK_N          pERK_N           pJNK_N
##  Min.   :0.1128   Min.   :0.429   Min.   :0.1492   Min.   :0.05211
##  1st Qu.:0.1908   1st Qu.:1.206   1st Qu.:0.3375   1st Qu.:0.28153
##  Median :0.2107   Median :1.356   Median :0.4442   Median :0.32127
##  Mean   :0.2126   Mean   :1.429   Mean   :0.5459   Mean   :0.31351
##  3rd Qu.:0.2346   3rd Qu.:1.561   3rd Qu.:0.6632   3rd Qu.:0.34869
##  Max.   :0.3062   Max.   :6.113   Max.   :3.5667   Max.   :0.49343
##
##      PKCA_N           pMEK_N            pNR1_N          pNR2A_N
##  Min.   :0.1914   Min.   :0.05682   Min.   :0.5002   Min.   :0.2813
##  1st Qu.:0.2818   1st Qu.:0.24429   1st Qu.:0.7436   1st Qu.:0.5913
##  Median :0.3130   Median :0.27718   Median :0.8215   Median :0.7206
##  Mean   :0.3179   Mean   :0.27503   Mean   :0.8258   Mean   :0.7269
##  3rd Qu.:0.3523   3rd Qu.:0.30335   3rd Qu.:0.8983   3rd Qu.:0.8473
##  Max.   :0.4740   Max.   :0.45800   Max.   :1.4082   Max.   :1.4128
##
##     pNR2B_N          pPKCAB_N          pRSK_N            AKT_N
##  Min.   :0.3016   Min.   :0.5678   Min.   :0.09594   Min.   :0.06442
##  1st Qu.:1.3813   1st Qu.:1.1686   1st Qu.:0.40415   1st Qu.:0.59732
##  Median :1.5632   Median :1.3688   Median :0.44064   Median :0.68224
##  Mean   :1.5620   Mean   :1.5253   Mean   :0.44285   Mean   :0.68224
##  3rd Qu.:1.7485   3rd Qu.:1.8812   3rd Qu.:0.48181   3rd Qu.:0.75891
##  Max.   :2.7240   Max.   :3.0614   Max.   :0.65096   Max.   :1.18217
##
##      BRAF_N          CAMKII_N          CREB_N           ELK_N
##  Min.   :0.1439   Min.   :0.2130   Min.   :0.1136   Min.   :0.4977
##  1st Qu.:0.2644   1st Qu.:0.3309   1st Qu.:0.1619   1st Qu.:0.9479
##  Median :0.3267   Median :0.3605   Median :0.1796   Median :1.1028
##  Mean   :0.3785   Mean   :0.3634   Mean   :0.1805   Mean   :1.1734
##  3rd Qu.:0.4127   3rd Qu.:0.3938   3rd Qu.:0.1957   3rd Qu.:1.3182
##  Max.   :2.1334   Max.   :0.5862   Max.   :0.3196   Max.   :2.8029
##
##      ERK_N          GSK3B_N           JNK_N            MEK_N
##  Min.   :1.132   Min.   :0.1511   Min.   :0.0463   Min.   :0.1472
##  1st Qu.:1.994   1st Qu.:1.0233   1st Qu.:0.2204   1st Qu.:0.2473
##  Median :2.403   Median :1.1604   Median :0.2448   Median :0.2731
##  Mean   :2.474   Mean   :1.1726   Mean   :0.2416   Mean   :0.2728
##  3rd Qu.:2.871   3rd Qu.:1.3094   3rd Qu.:0.2632   3rd Qu.:0.3005
##  Max.   :5.198   Max.   :2.4758   Max.   :0.3872   Max.   :0.4154
##
##      TRKA_N           RSK_N            APP_N          Bcatenin_N
##  Min.   :0.1987   Min.   :0.1074   Min.   :0.2356   Min.   :1.135
##  1st Qu.:0.6173   1st Qu.:0.1496   1st Qu.:0.3665   1st Qu.:1.830
##  Median :0.7049   Median :0.1667   Median :0.4022   Median :2.127
##  Mean   :0.6932   Mean   :0.1684   Mean   :0.4048   Mean   :2.147
##  3rd Qu.:0.7740   3rd Qu.:0.1845   3rd Qu.:0.4419   3rd Qu.:2.419
##  Max.   :1.0016   Max.   :0.3051   Max.   :0.6327   Max.   :3.681
##
##      SOD1_N           MTOR_N           P38_N           pMTOR_N
##  Min.   :0.2171   Min.   :0.2011   Min.   :0.2279   Min.   :0.1666
##  1st Qu.:0.3197   1st Qu.:0.4110   1st Qu.:0.3520   1st Qu.:0.6836
```

```
##  Median :0.4460   Median :0.4525   Median :0.4080   Median :0.7597
##  Mean   :0.5426   Mean   :0.4525   Mean   :0.4153   Mean   :0.7590
##  3rd Qu.:0.6953   3rd Qu.:0.4880   3rd Qu.:0.4662   3rd Qu.:0.8412
##  Max.   :1.8729   Max.   :0.6767   Max.   :0.9333   Max.   :1.1249
##
##     DSCR1_N          AMPKA_N           NR2B_N          pNUMB_N
##  Min.   :0.1553   Min.   :0.2264   Min.   :0.1848   Min.   :0.1856
##  1st Qu.:0.5312   1st Qu.:0.3267   1st Qu.:0.5151   1st Qu.:0.3128
##  Median :0.5770   Median :0.3588   Median :0.5637   Median :0.3476
##  Mean   :0.5852   Mean   :0.3684   Mean   :0.5653   Mean   :0.3571
##  3rd Qu.:0.6343   3rd Qu.:0.4007   3rd Qu.:0.6143   3rd Qu.:0.3926
##  Max.   :0.9164   Max.   :0.7008   Max.   :0.9720   Max.   :0.6311
##
##     RAPTOR_N         TIAM1_N          pP70S6_N         NUMB_N
##  Min.   :0.1948   Min.   :0.2378   Min.   :0.1311   Min.   :0.1180
##  1st Qu.:0.2762   1st Qu.:0.3721   1st Qu.:0.2821   1st Qu.:0.1593
##  Median :0.3050   Median :0.4074   Median :0.3787   Median :0.1782
##  Mean   :0.3158   Mean   :0.4186   Mean   :0.3945   Mean   :0.1811
##  3rd Qu.:0.3473   3rd Qu.:0.4559   3rd Qu.:0.4807   3rd Qu.:0.1972
##  Max.   :0.5267   Max.   :0.7221   Max.   :1.1292   Max.   :0.3166
##
##     P70S6_N          pGSK3B_N          pPKCG_N          CDK5_N
##  Min.   :0.3441   Min.   :0.09998   Min.   :0.5988   Min.   :0.1812
##  1st Qu.:0.8267   1st Qu.:0.14925   1st Qu.:1.2968   1st Qu.:0.2726
##  Median :0.9313   Median :0.16021   Median :1.6646   Median :0.2938
##  Mean   :0.9431   Mean   :0.16121   Mean   :1.7066   Mean   :0.2924
##  3rd Qu.:1.0451   3rd Qu.:0.17174   3rd Qu.:2.1130   3rd Qu.:0.3125
##  Max.   :1.6800   Max.   :0.25321   Max.   :3.3820   Max.   :0.8174
##
##       S6_N           ADARB1_N       AcetylH3K9_N         RRP1_N
##  Min.   :0.1302   Min.   :0.5291   Min.   :0.05253   Min.   :-0.06201
##  1st Qu.:0.3167   1st Qu.:0.9305   1st Qu.:0.10357   1st Qu.: 0.14902
##  Median :0.4010   Median :1.1283   Median :0.15042   Median : 0.16210
##  Mean   :0.4292   Mean   :1.1974   Mean   :0.21648   Mean   : 0.16663
##  3rd Qu.:0.5349   3rd Qu.:1.3802   3rd Qu.:0.26965   3rd Qu.: 0.17741
##  Max.   :0.8226   Max.   :2.5399   Max.   :1.45939   Max.   : 0.61238
##
##      BAX_N             ARC_N            ERBB4_N          nNOS_N
##  Min.   :0.07233   Min.   :0.06725   Min.   :0.1002   Min.   :0.09973
##  1st Qu.:0.16817   1st Qu.:0.11084   1st Qu.:0.1470   1st Qu.:0.16645
##  Median :0.18074   Median :0.12163   Median :0.1564   Median :0.18267
##  Mean   :0.17931   Mean   :0.12152   Mean   :0.1565   Mean   :0.18130
##  3rd Qu.:0.19158   3rd Qu.:0.13196   3rd Qu.:0.1654   3rd Qu.:0.19857
##  Max.   :0.24114   Max.   :0.15875   Max.   :0.2087   Max.   :0.26074
##
##      Tau_N             GFAP_N           GluR3_N          GluR4_N
##  Min.   :0.09623   Min.   :0.08611   Min.   :0.1114   Min.   :0.07258
##  1st Qu.:0.16799   1st Qu.:0.11277   1st Qu.:0.1957   1st Qu.:0.10889
##  Median :0.18863   Median :0.12046   Median :0.2169   Median :0.12355
##  Mean   :0.21049   Mean   :0.12089   Mean   :0.2219   Mean   :0.12656
##  3rd Qu.:0.23394   3rd Qu.:0.12772   3rd Qu.:0.2460   3rd Qu.:0.14195
##  Max.   :0.60277   Max.   :0.21362   Max.   :0.3310   Max.   :0.53700
##
##      IL1B_N           P3525_N          pCASP9_N         PSD95_N
```

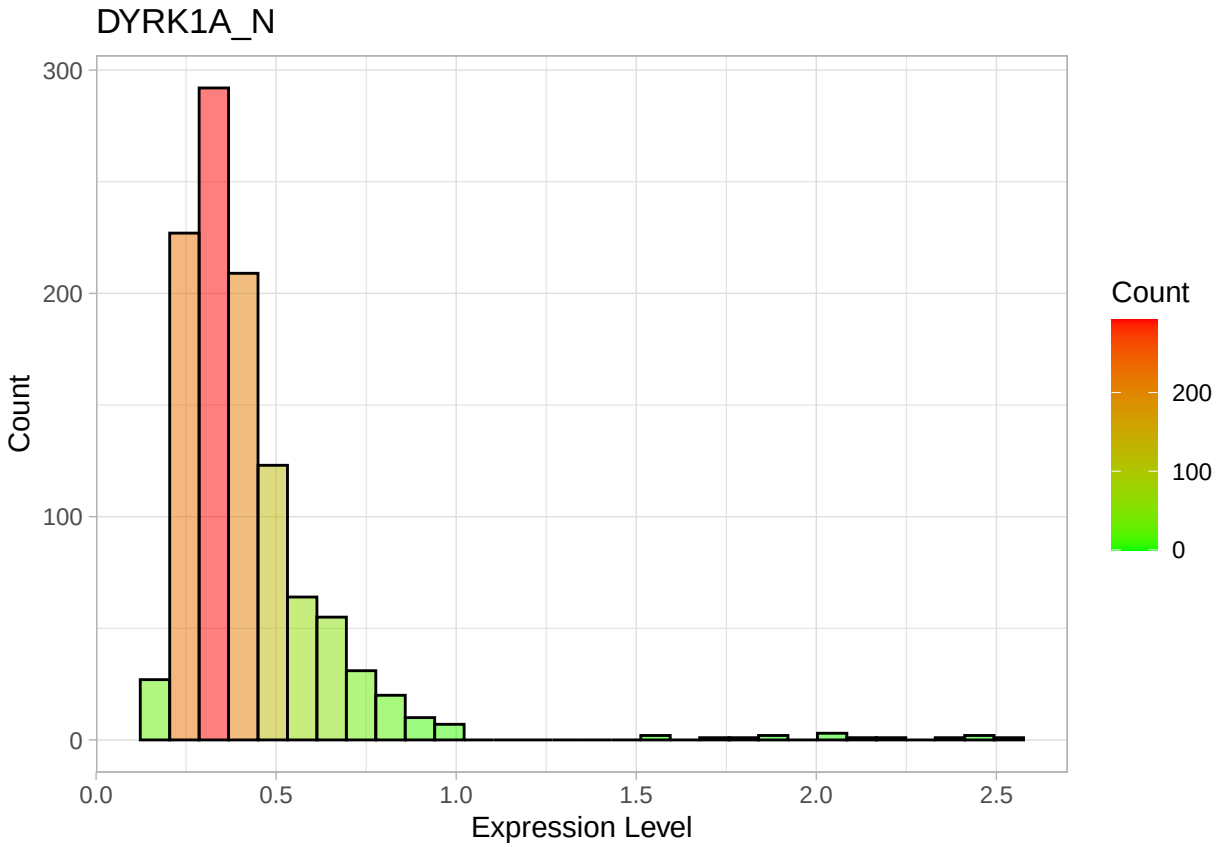
```
##  Min.   :0.2840   Min.   :0.2074   Min.   :0.8532   Min.   :1.206
##  1st Qu.:0.4756   1st Qu.:0.2701   1st Qu.:1.3756   1st Qu.:2.079
##  Median :0.5267   Median :0.2906   Median :1.5227   Median :2.242
##  Mean   :0.5273   Mean   :0.2913   Mean   :1.5483   Mean   :2.235
##  3rd Qu.:0.5770   3rd Qu.:0.3116   3rd Qu.:1.7131   3rd Qu.:2.420
##  Max.   :0.8897   Max.   :0.4437   Max.   :2.5862   Max.   :2.878
##
##      SNCA_N        Ubiquitin_N     pGSK3B_Tyr216_N      SHH_N
##  Min.   :0.1012   Min.   :0.7507   Min.   :0.5774   Min.   :0.1559
##  1st Qu.:0.1428   1st Qu.:1.1163   1st Qu.:0.7937   1st Qu.:0.2064
##  Median :0.1575   Median :1.2366   Median :0.8499   Median :0.2240
##  Mean   :0.1598   Mean   :1.2393   Mean   :0.8488   Mean   :0.2267
##  3rd Qu.:0.1733   3rd Qu.:1.3631   3rd Qu.:0.9162   3rd Qu.:0.2417
##  Max.   :0.2576   Max.   :1.8972   Max.   :1.2046   Max.   :0.3583
##
##      BAD_N            BCL2_N            pS6_N            pCFOS_N
##  Min.   :0.0883   Min.   :0.08066   Min.   :0.06725   Min.   :0.08542
##  1st Qu.:0.1410   1st Qu.:0.11999   1st Qu.:0.11084   1st Qu.:0.11437
##  Median :0.1579   Median :0.13476   Median :0.12163   Median :0.12847
##  Mean   :0.1579   Mean   :0.13476   Mean   :0.12152   Mean   :0.13105
##  3rd Qu.:0.1676   3rd Qu.:0.13933   3rd Qu.:0.13196   3rd Qu.:0.14243
##  Max.   :0.2820   Max.   :0.26151   Max.   :0.15875   Max.   :0.25653
##
##      SYP_N          H3AcK18_N          EGR1_N          H3MeK4_N
##  Min.   :0.2586   Min.   :0.07969   Min.   :0.1055   Min.   :0.1018
##  1st Qu.:0.3981   1st Qu.:0.13397   1st Qu.:0.1592   1st Qu.:0.1743
##  Median :0.4485   Median :0.16961   Median :0.1831   Median :0.2054
##  Mean   :0.4461   Mean   :0.16961   Mean   :0.1831   Mean   :0.2054
##  3rd Qu.:0.4908   3rd Qu.:0.18717   3rd Qu.:0.1961   3rd Qu.:0.2192
##  Max.   :0.7596   Max.   :0.47976   Max.   :0.3607   Max.   :0.4139
##
##      CaNA_N          class
##  Min.   :0.5865   c-CS-m :150
##  1st Qu.:1.0814   c-SC-m :150
##  Median :1.3174   c-CS-s :135
##  Mean   :1.3378   c-SC-s :135
##  3rd Qu.:1.5858   t-CS-m :135
##  Max.   :2.1298   t-SC-m :135
##                   (Other):240
```

Exploratory Data Analysis

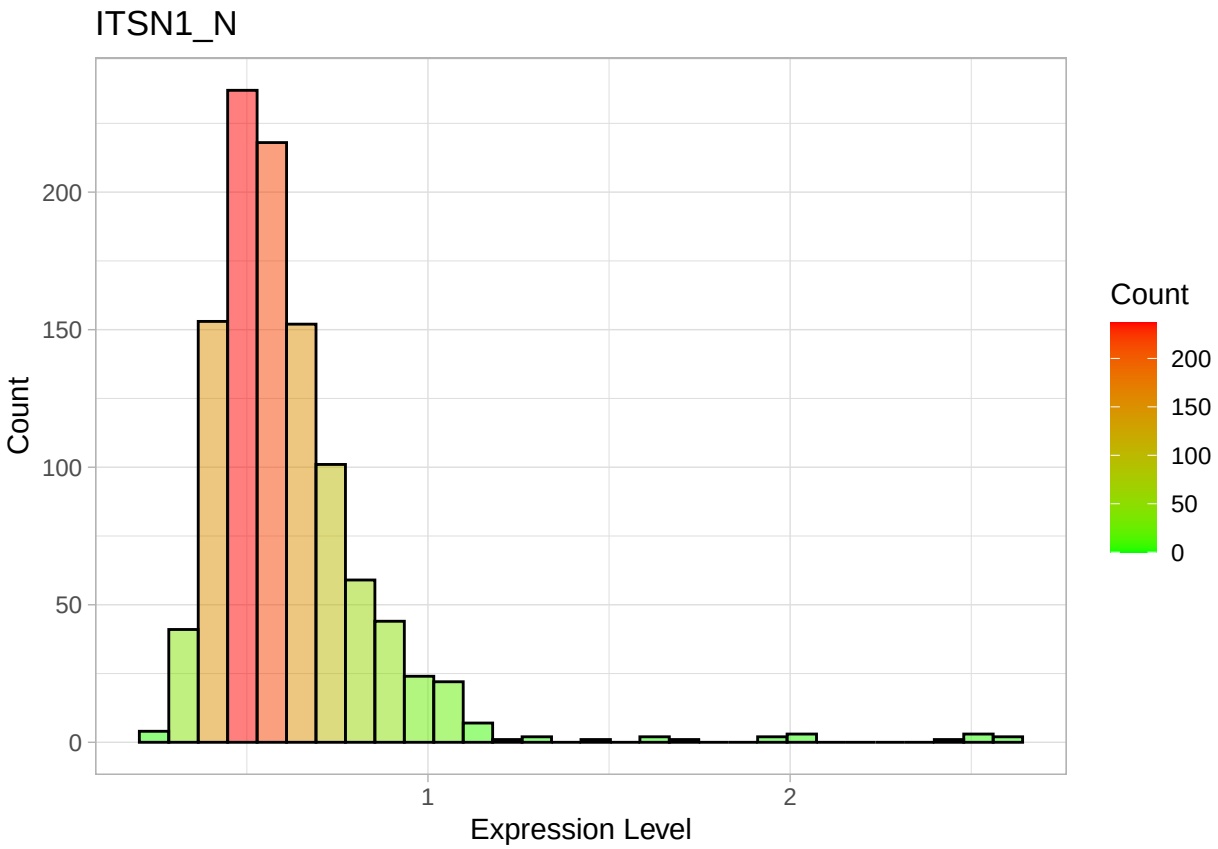
Histogram of Protein Expression Level

```
for(i in 1:length(proteins)){
  plot <- ggplot(ncortex, aes(eval(parse(text = proteins[i])))) +
    geom_histogram(aes(fill=..count..), color = "black", alpha = 0.5) +
    scale_fill_gradient("Count", low="green", high="red") +
    labs(title = proteins[i],
         x = "Expression Level",
         y = "Count") +
    theme_light()
```

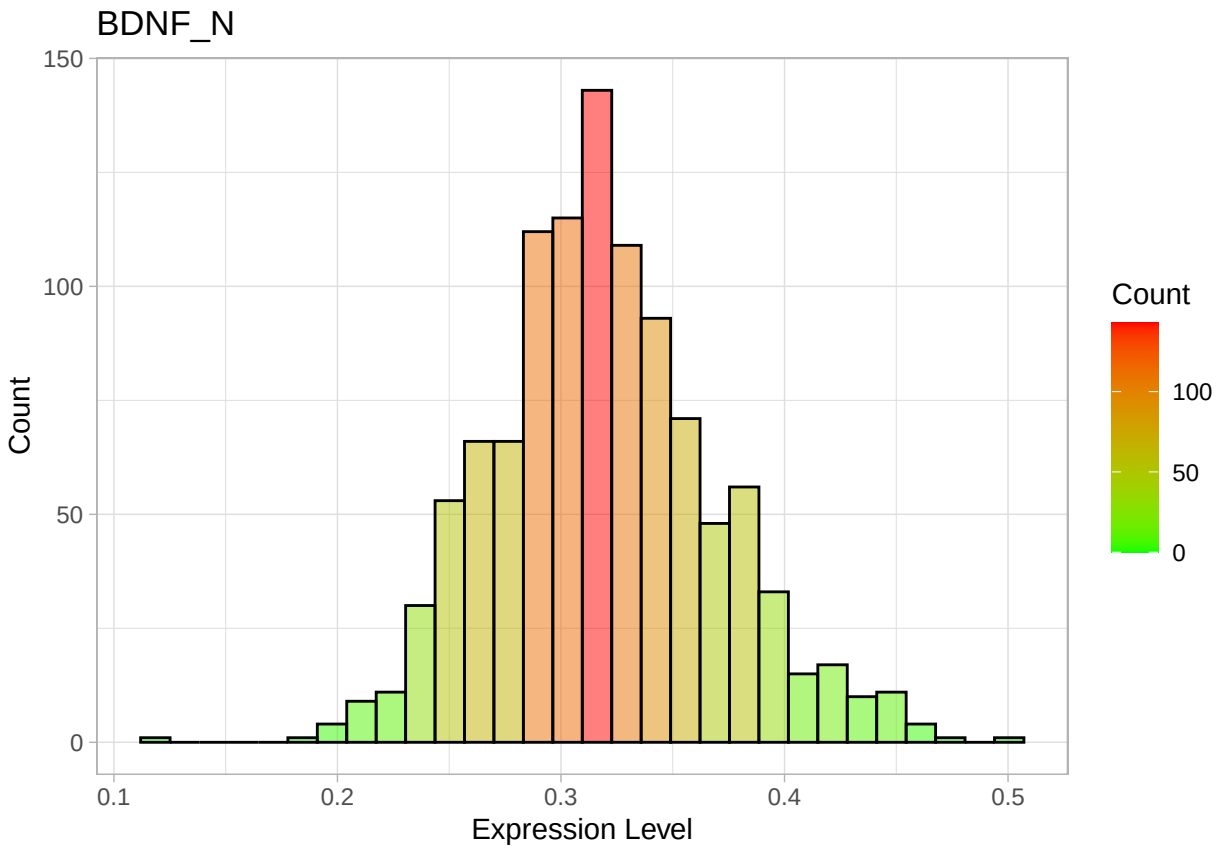
```
  cat("#### ", proteins[i], "\n")
  print(plot)
  cat('\n\n')
}
```



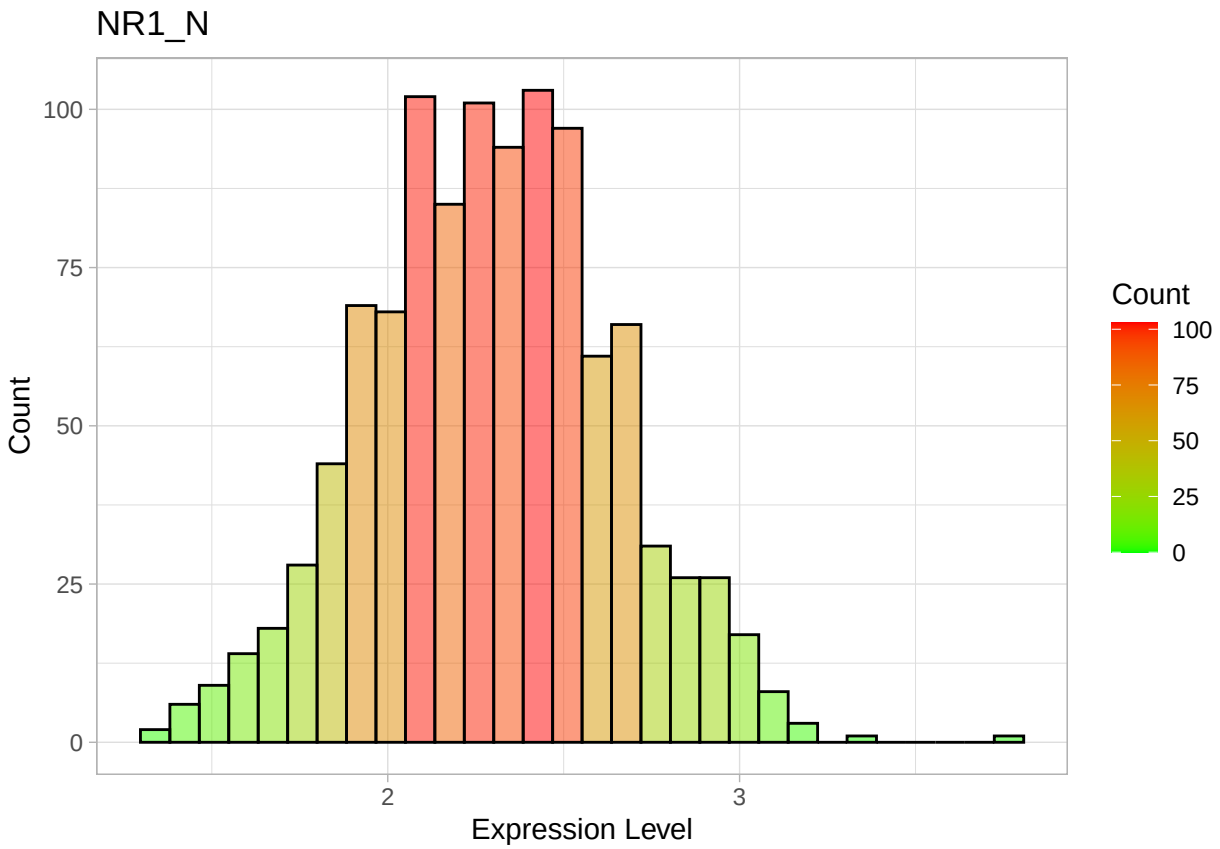
DYRK1A_N


ITSN1_N
Count
200
150
100
50
0
1
2
Expression Level
Count
200
150
100
50
0

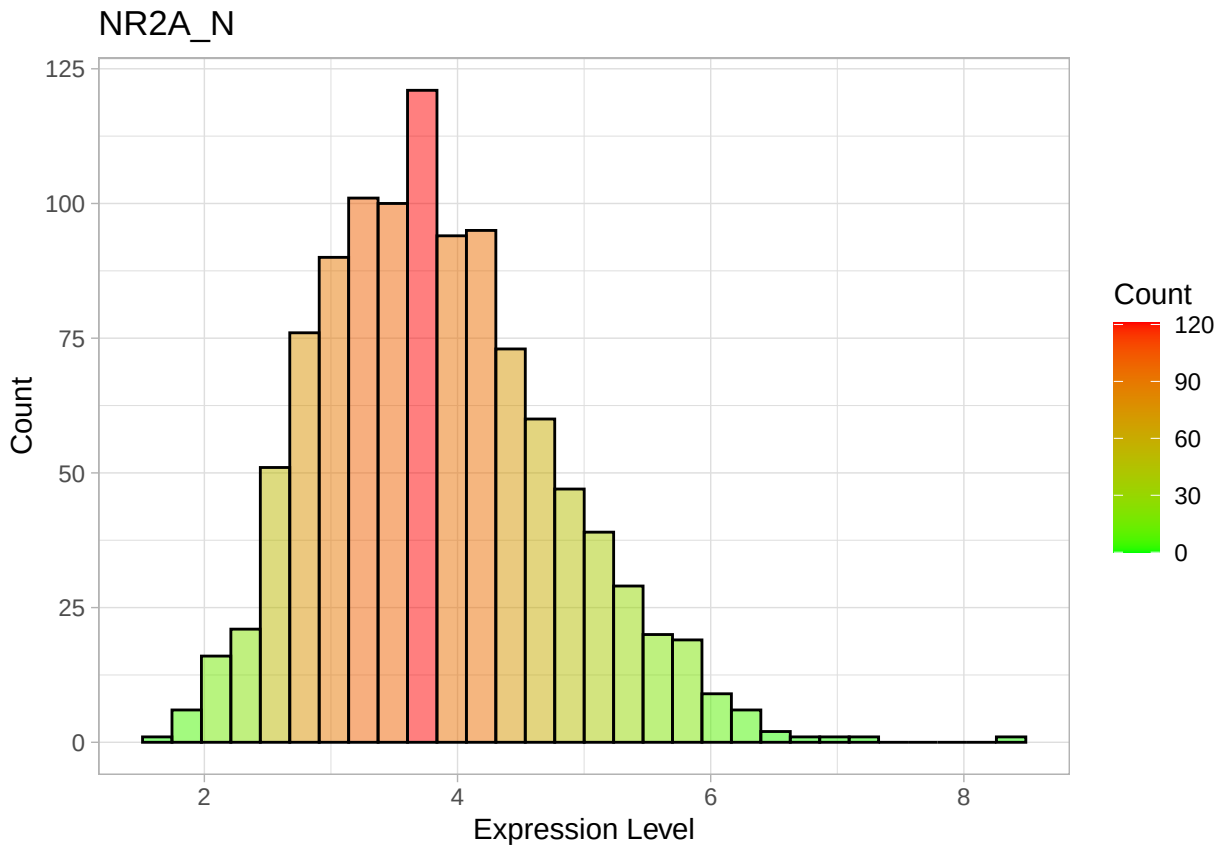
ITSN1__N


BDNF_N
150
100
50
0
Count
0.1
0.2
0.3
0.4
0.5
Expression Level
Count
100
50
0

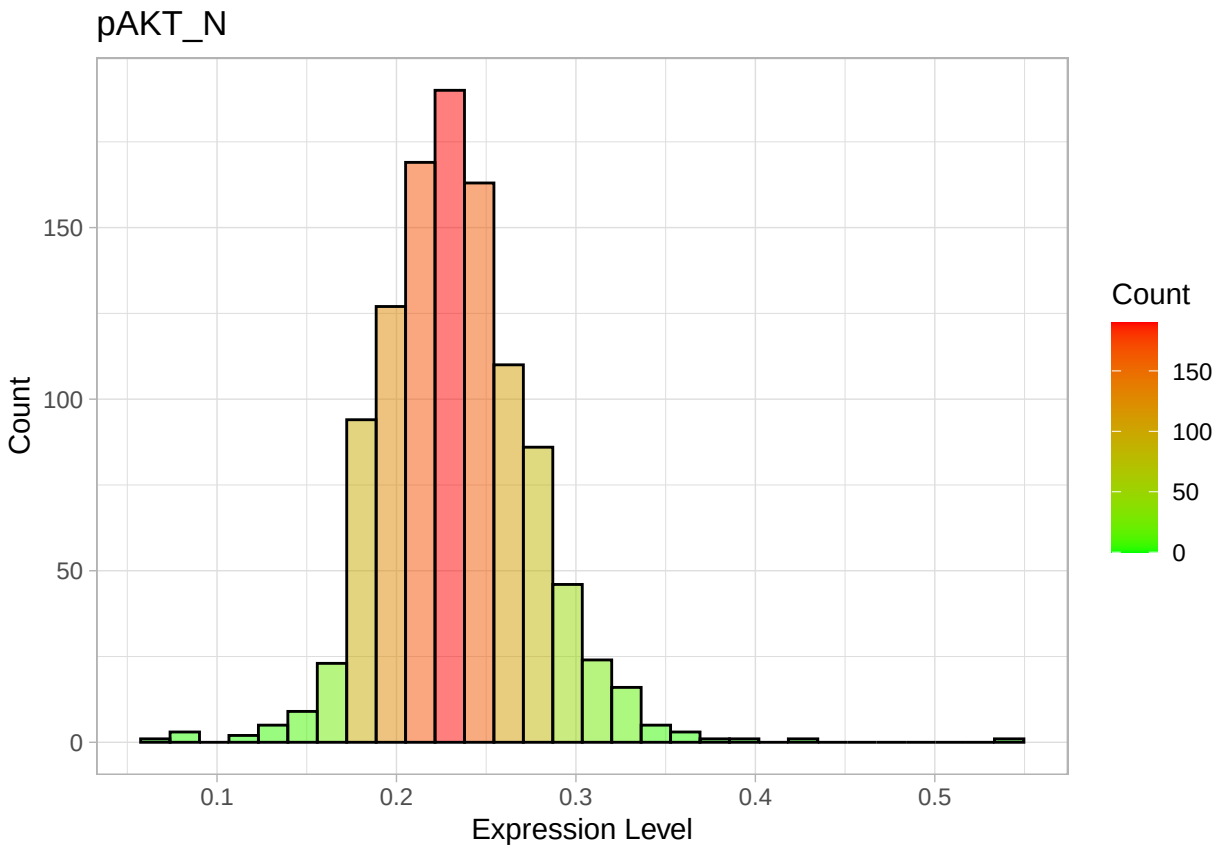
BDNF_N


NR1_N
Count
100
75
50
25
0
2
3
Expression Level
Count
100
75
50
25
0

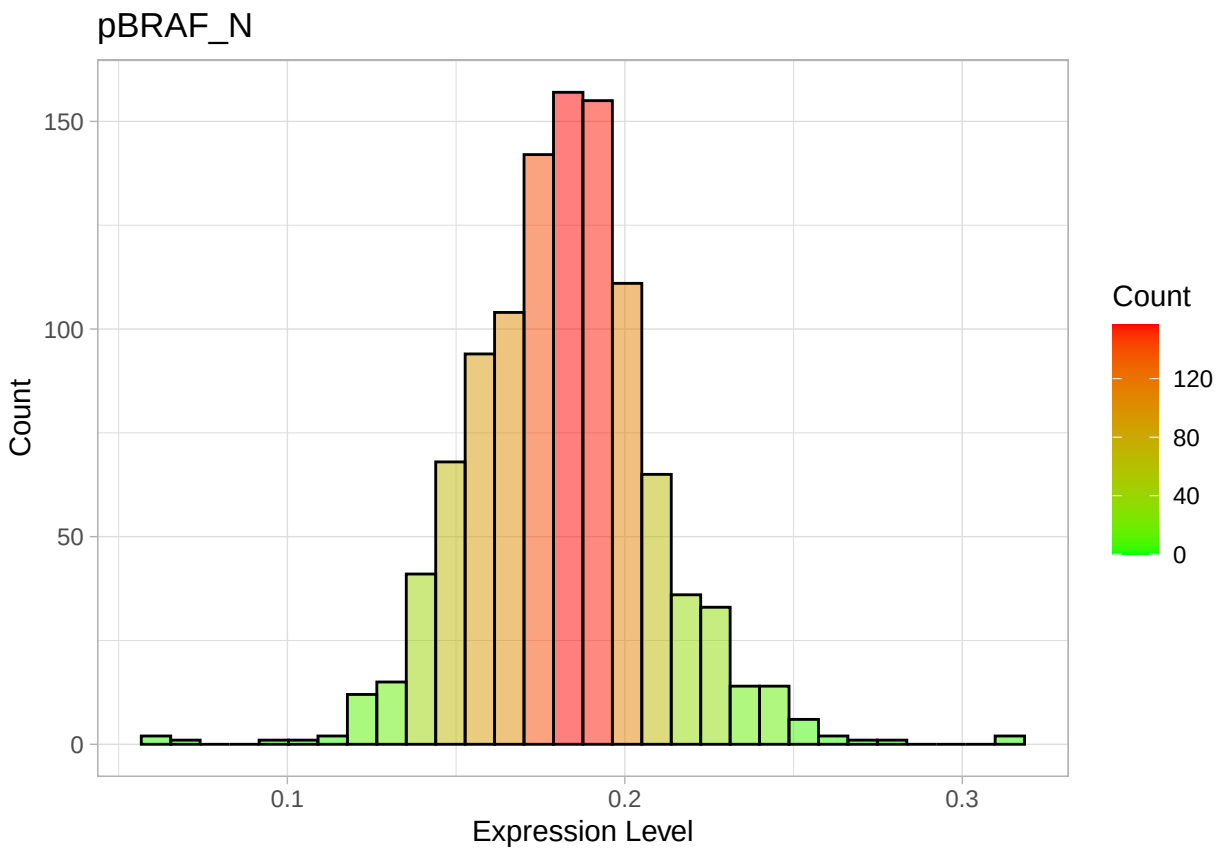
NR1_N


NR2A_N
125
100
75
50
25
0
Count
2
4
6
8
Expression Level
Count
120
90
60
30
0

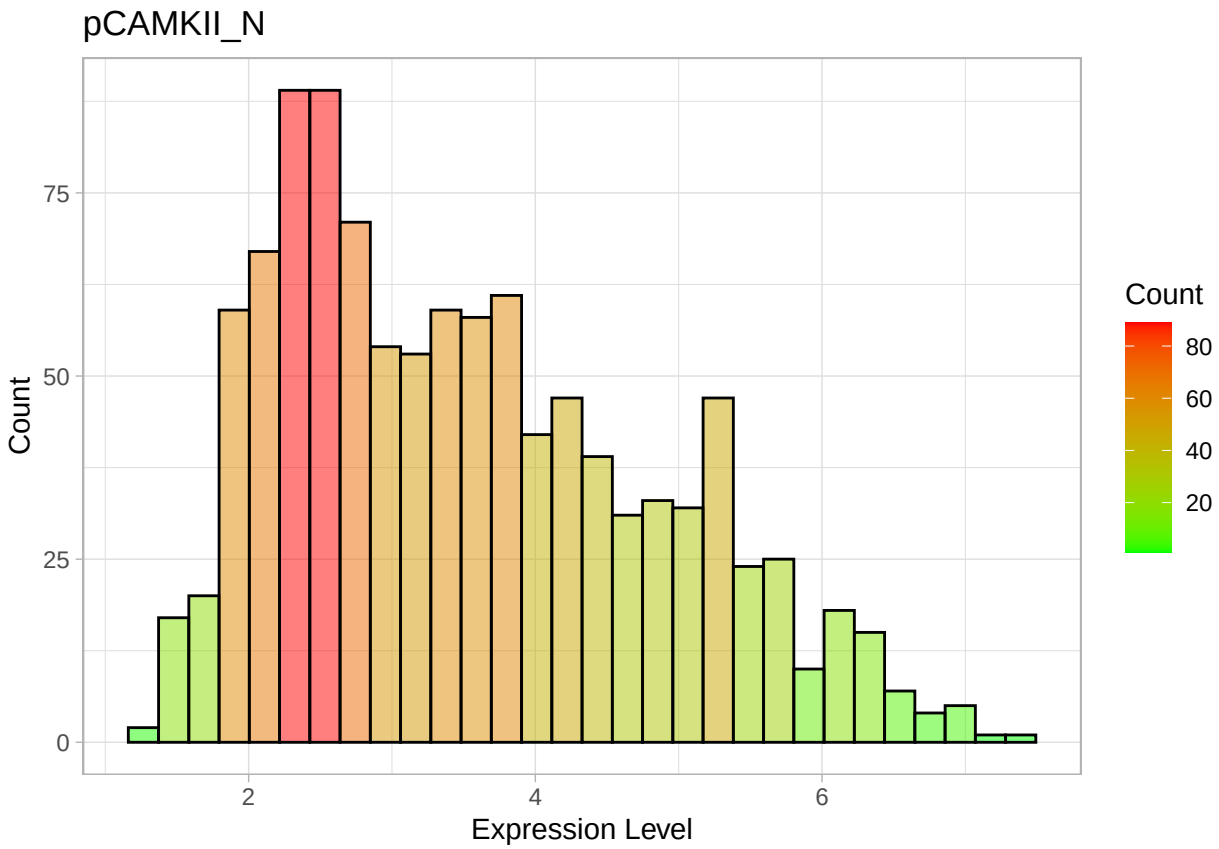
NR2A_N


pAKT_N
Count
150
100
50
0
0.1
0.2
0.3
0.4
0.5
Expression Level
Count
150
100
50
0

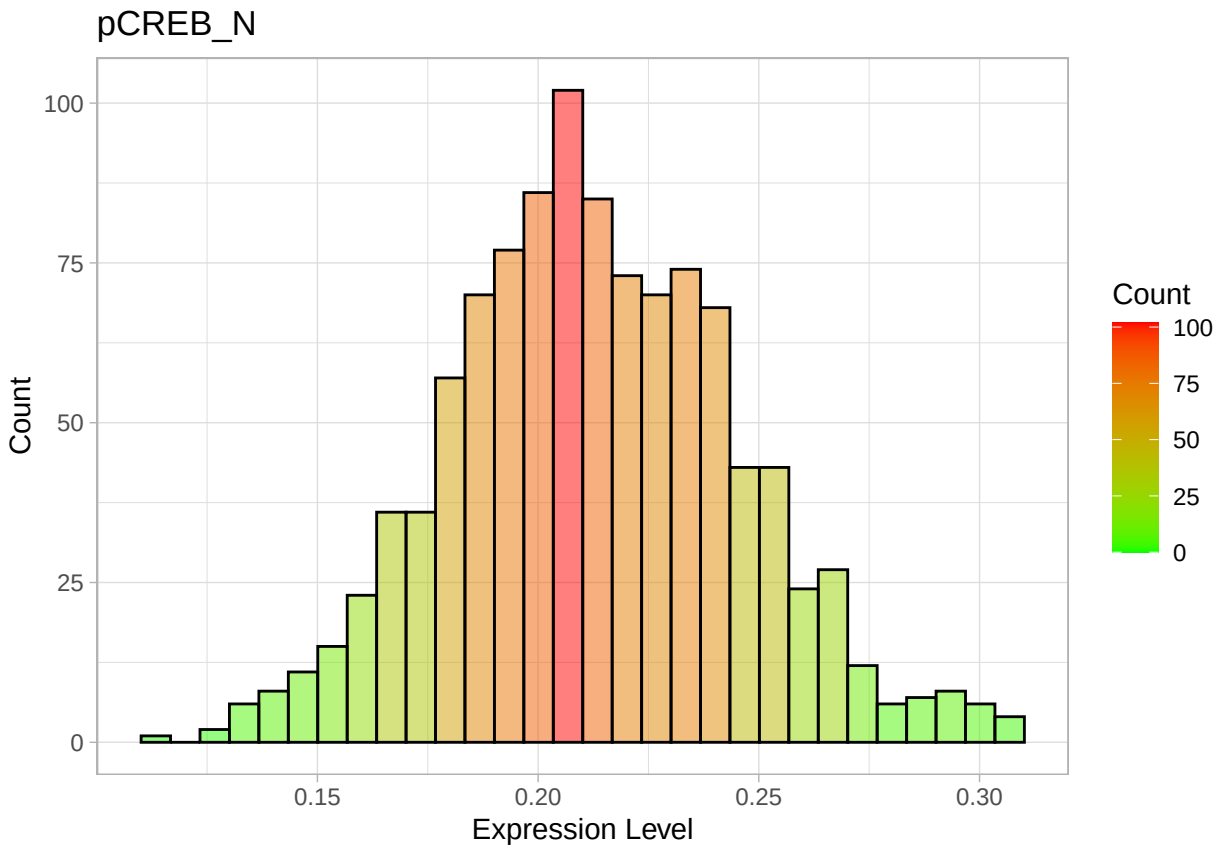
pAKT_N


pBRAF_N
150
100
50
0
Count
0.1
0.2
0.3
Expression Level
Count
120
80
40
0

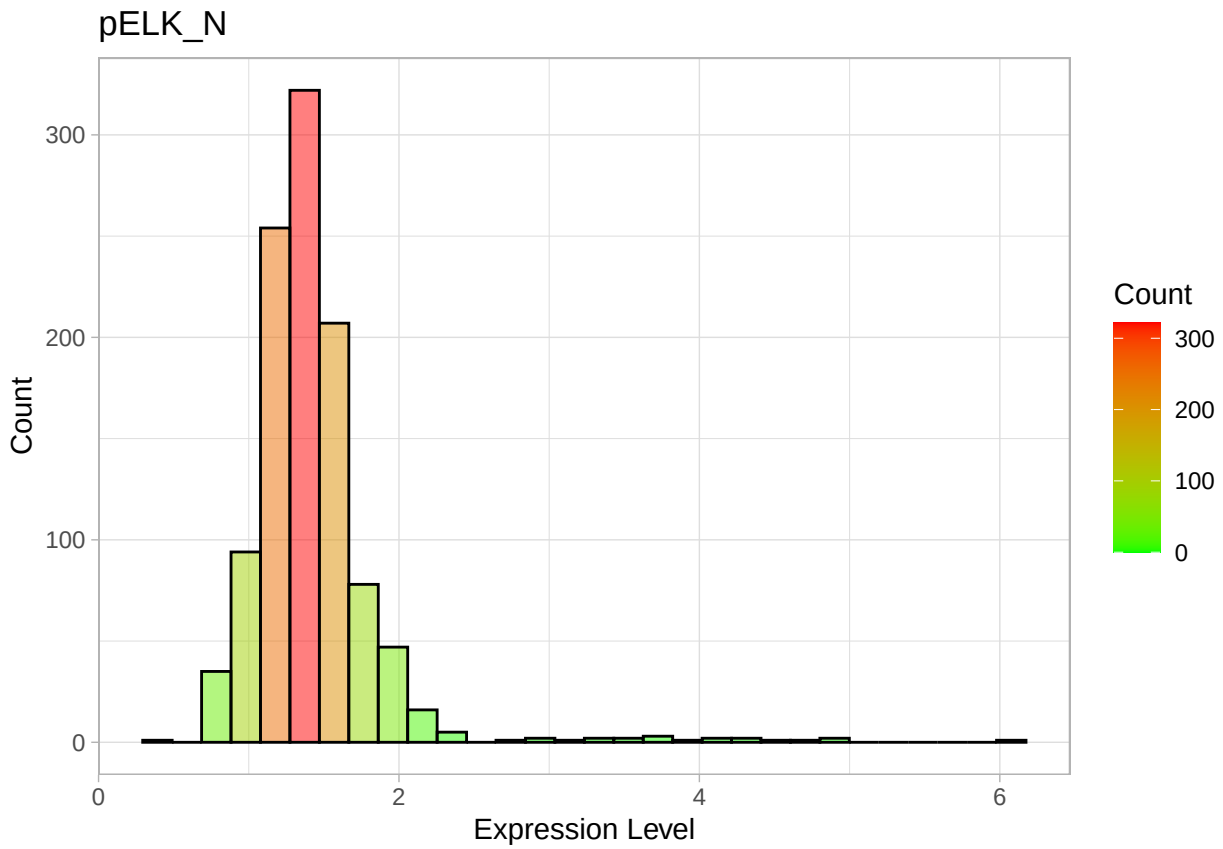
pBRAF_N


pCAMKII_N
Count
75
50
25
0
2
4
6
Expression Level
Count
80
60
40
20

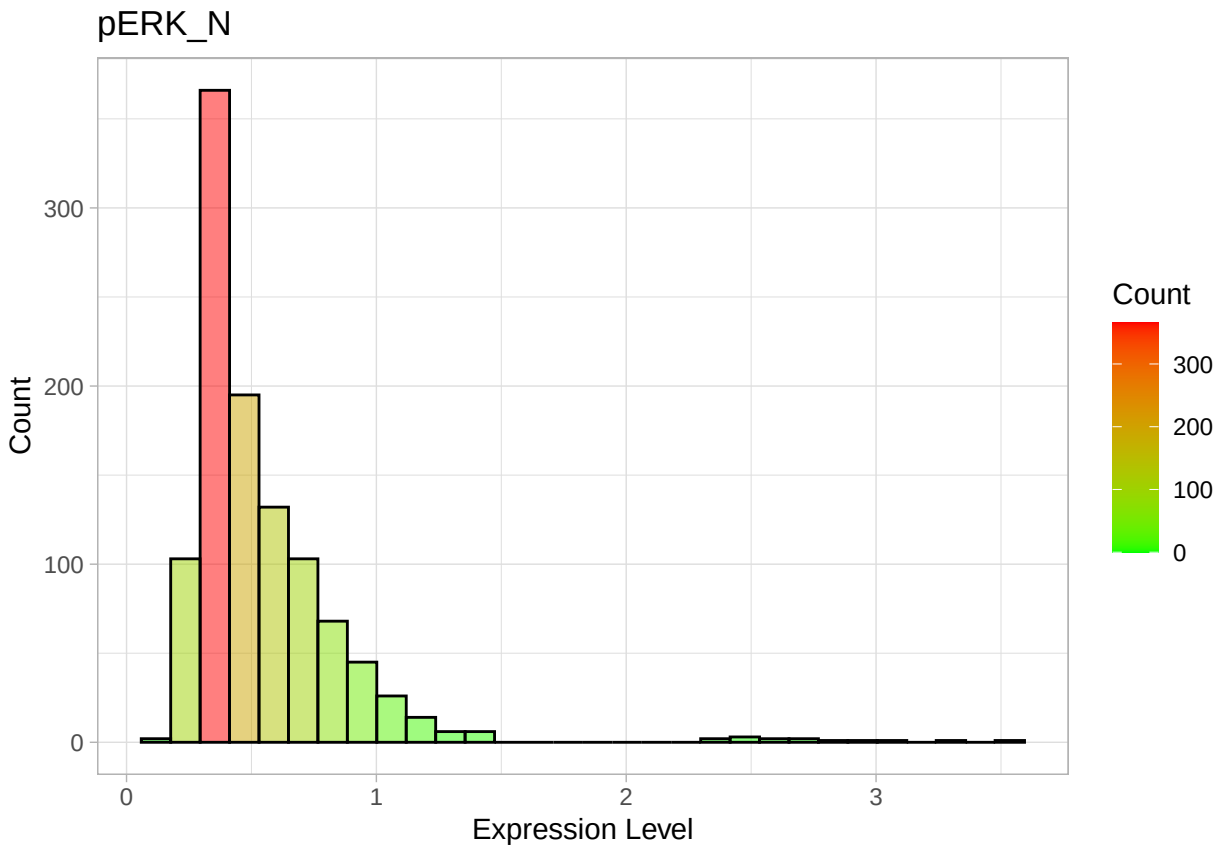
pCAMKII_N


pCREB_N
Count
100
75
50
25
0
0.15
0.20
0.25
0.30
Expression Level
Count
100
75
50
25
0

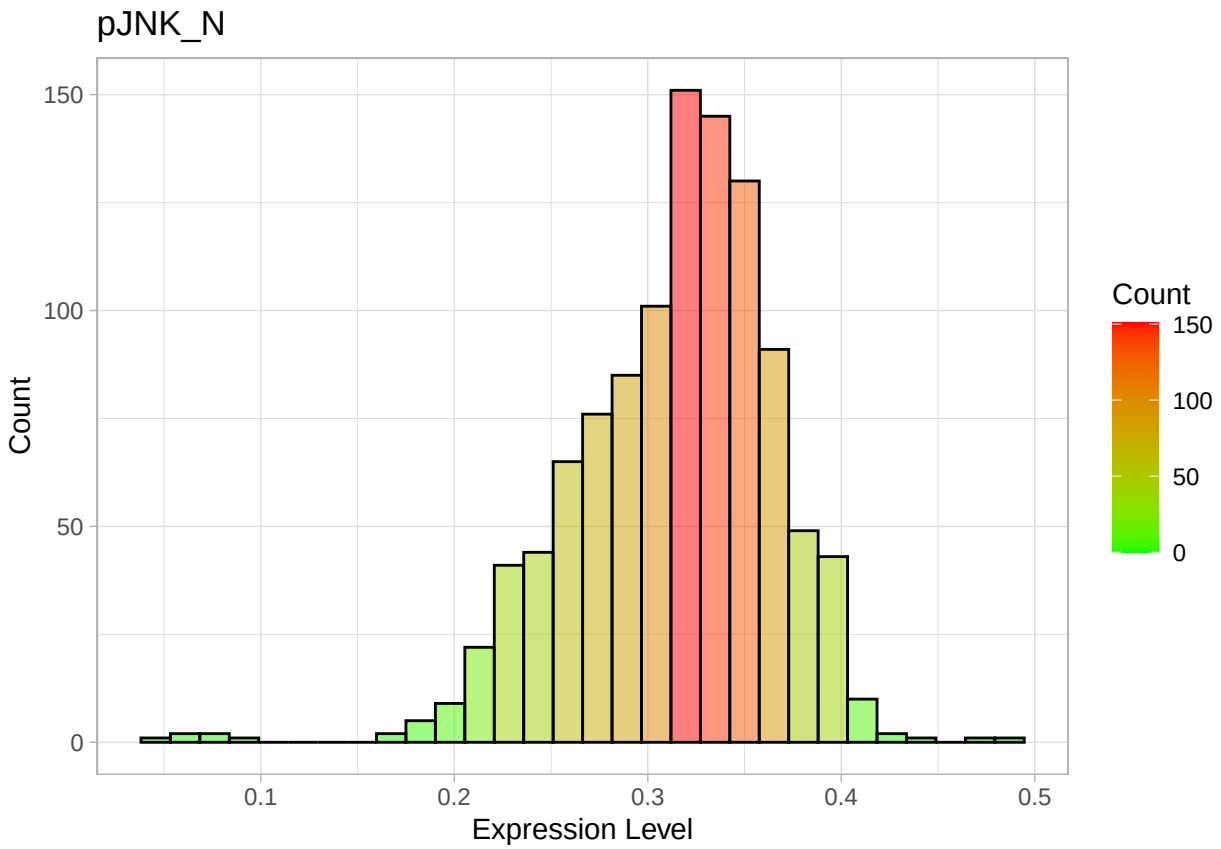
pCREB_N


pELK_N
Count
300
200
100
0
0
2
4
6
Expression Level
Count
300
200
100
0

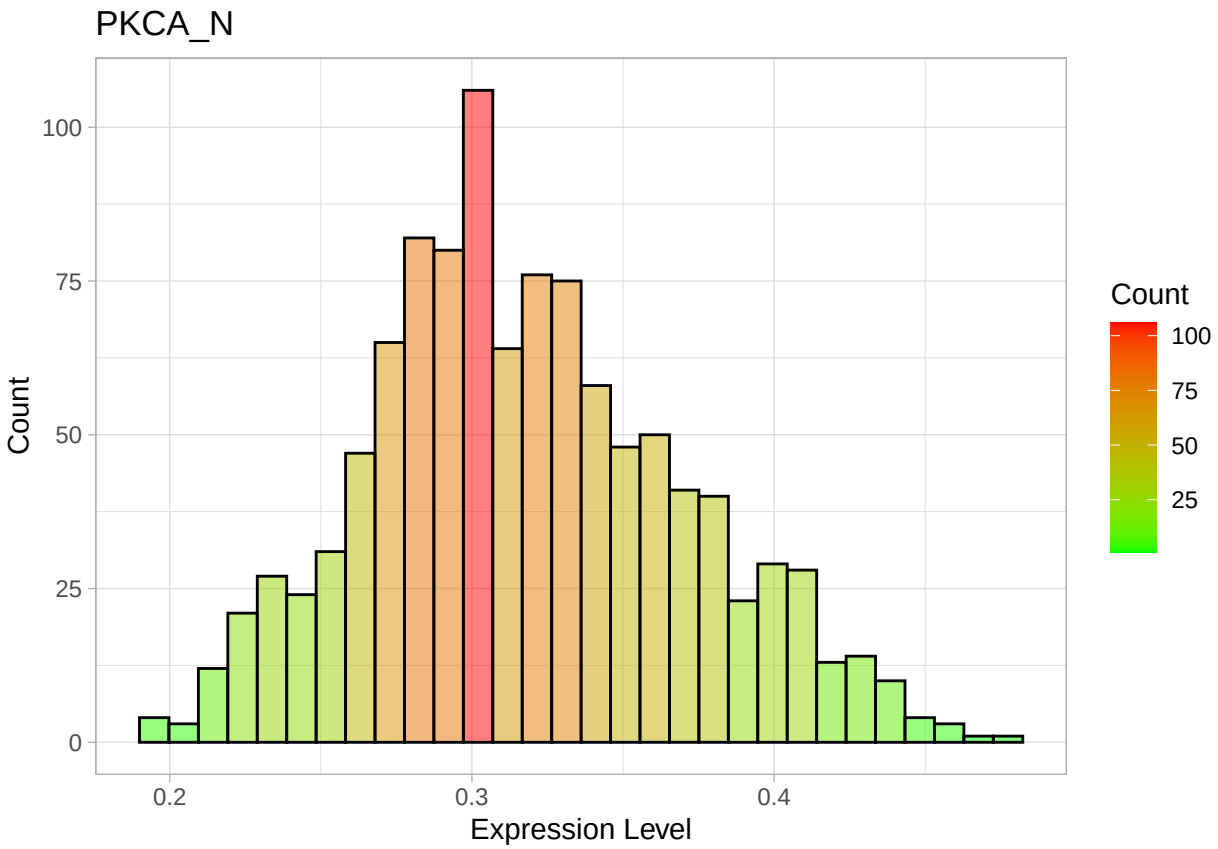
pELK_N


pERK_N
Count
300
200
100
0
0
1
2
3
Expression Level
Count
300
200
100
0

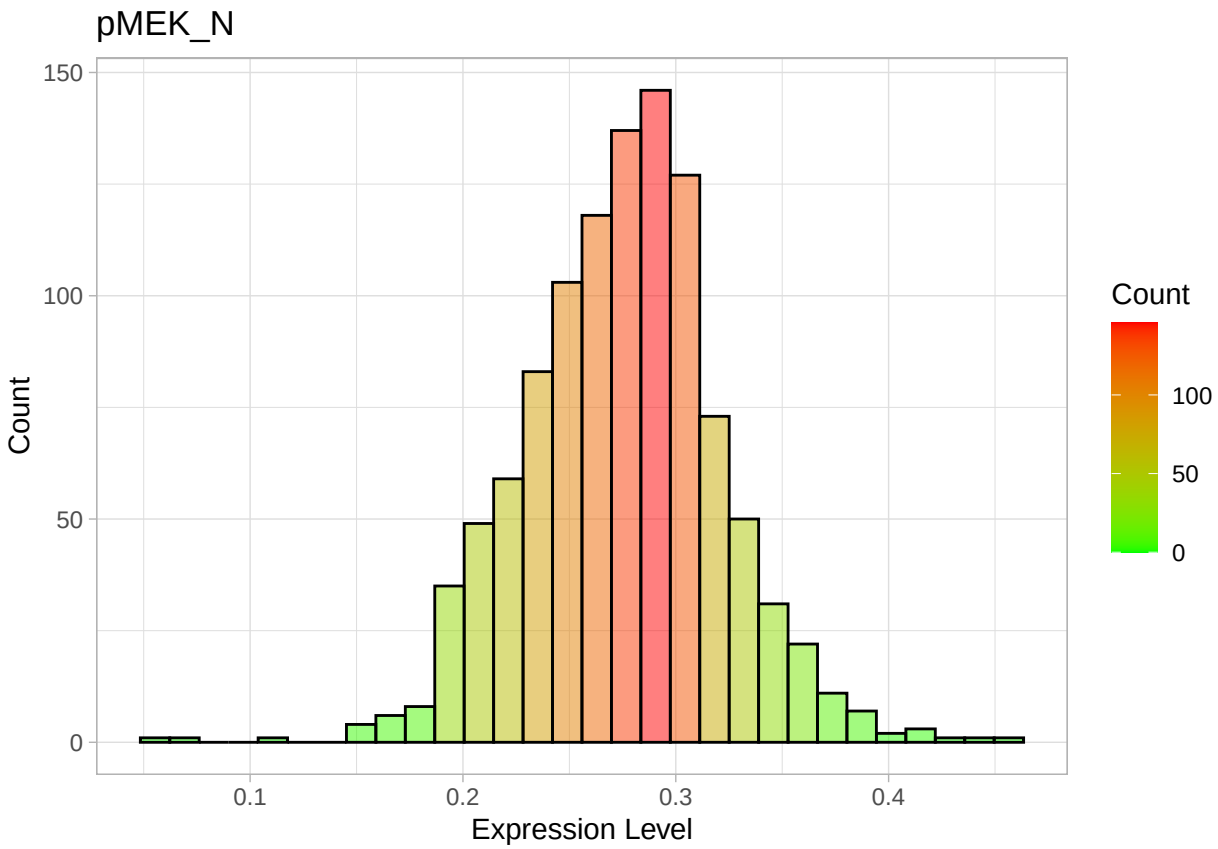
pERK_N


pJNK_N
150
100
50
0
Count
0.1
0.2
0.3
0.4
0.5
Expression Level
Count
150
100
50
0

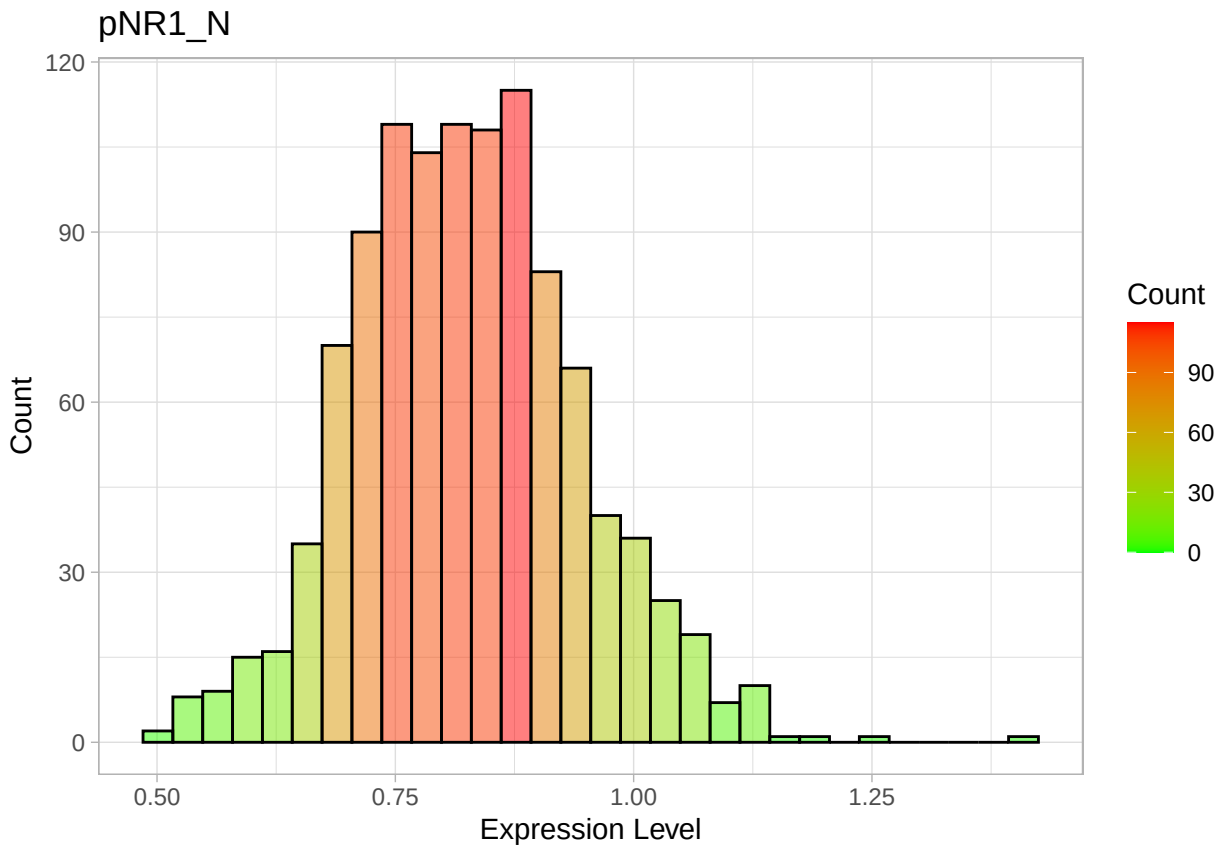
pJNK_N


PKCA_N
Count
100
75
50
25
0
0.2
0.3
0.4
Expression Level
Count
100
75
50
25

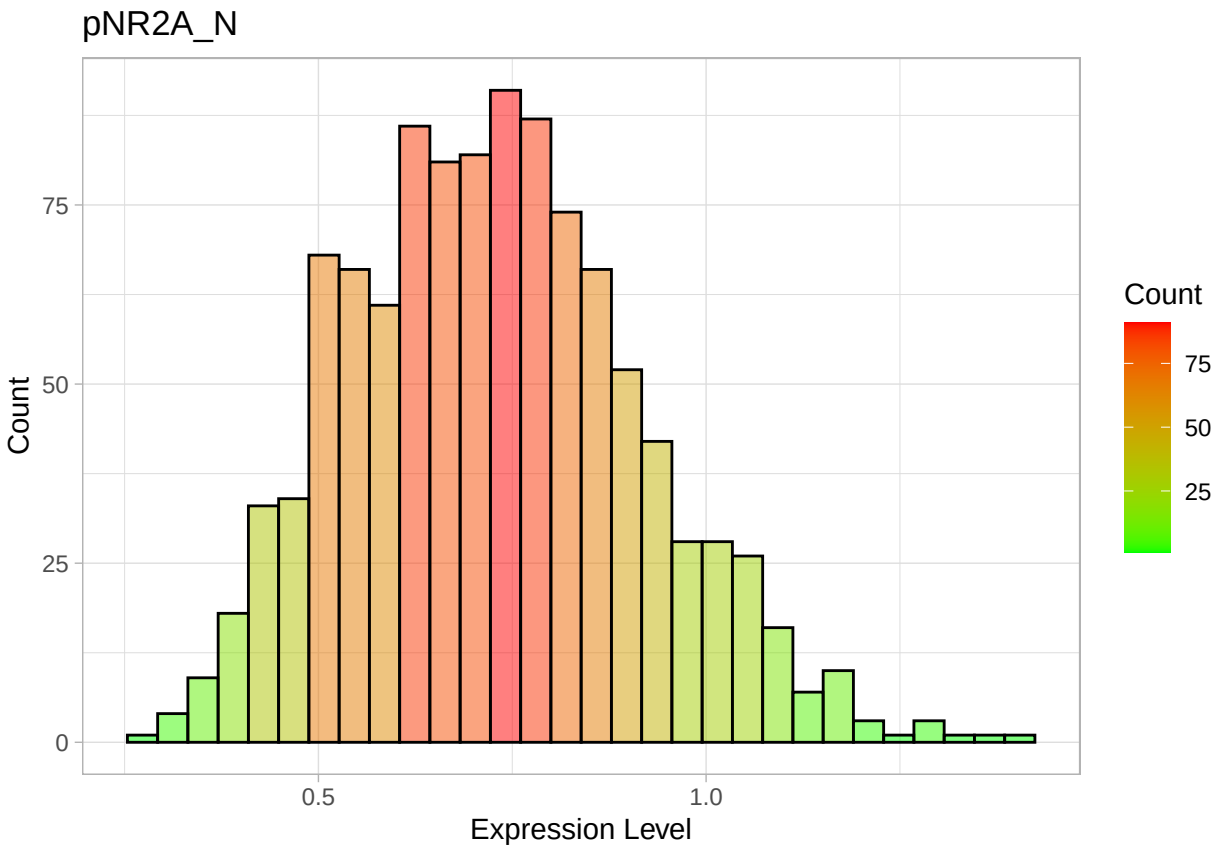
PKCA_N


pMEK_N
150
100
50
0
Count
0.1
0.2
0.3
0.4
Expression Level
Count
100
50
0

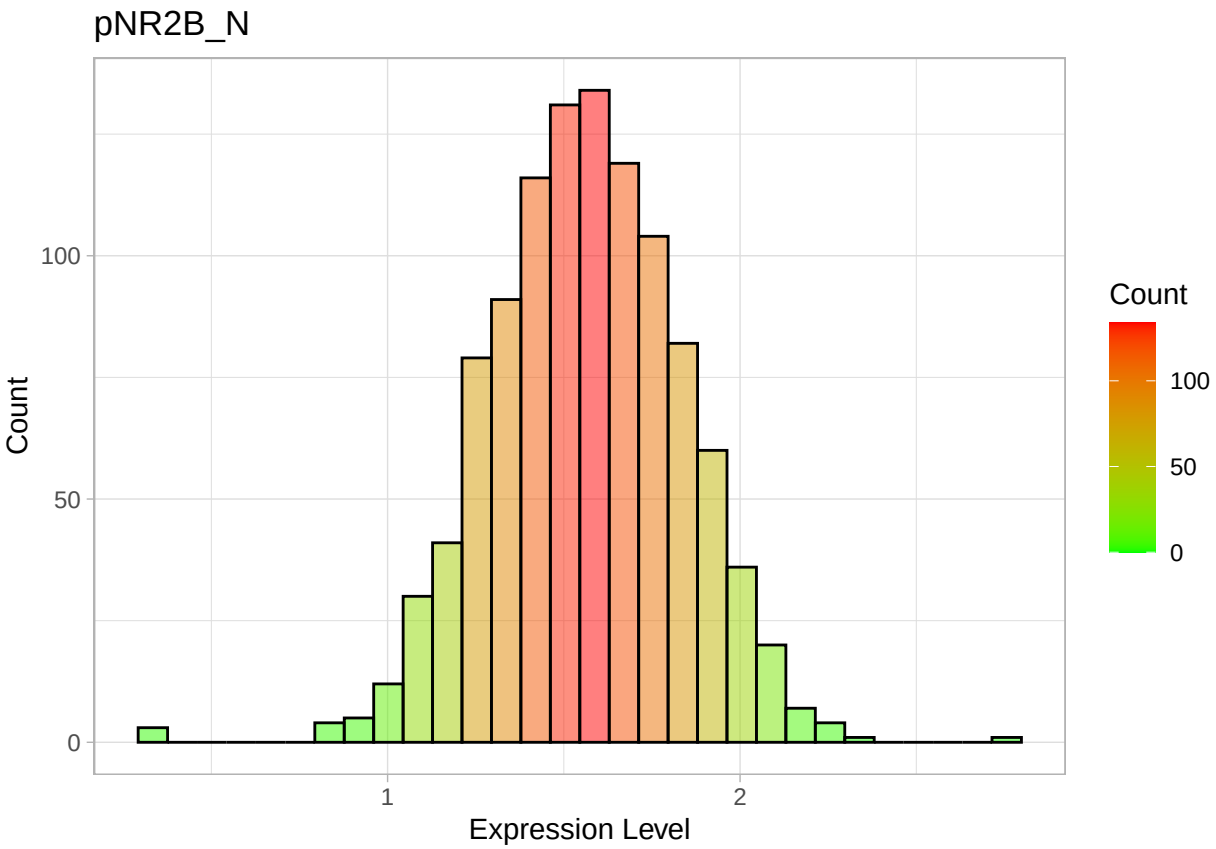
pMEK_N


pNR1_N
120
90
60
30
0
Count
0.50
0.75
1.00
1.25
Expression Level
Count
90
60
30
0

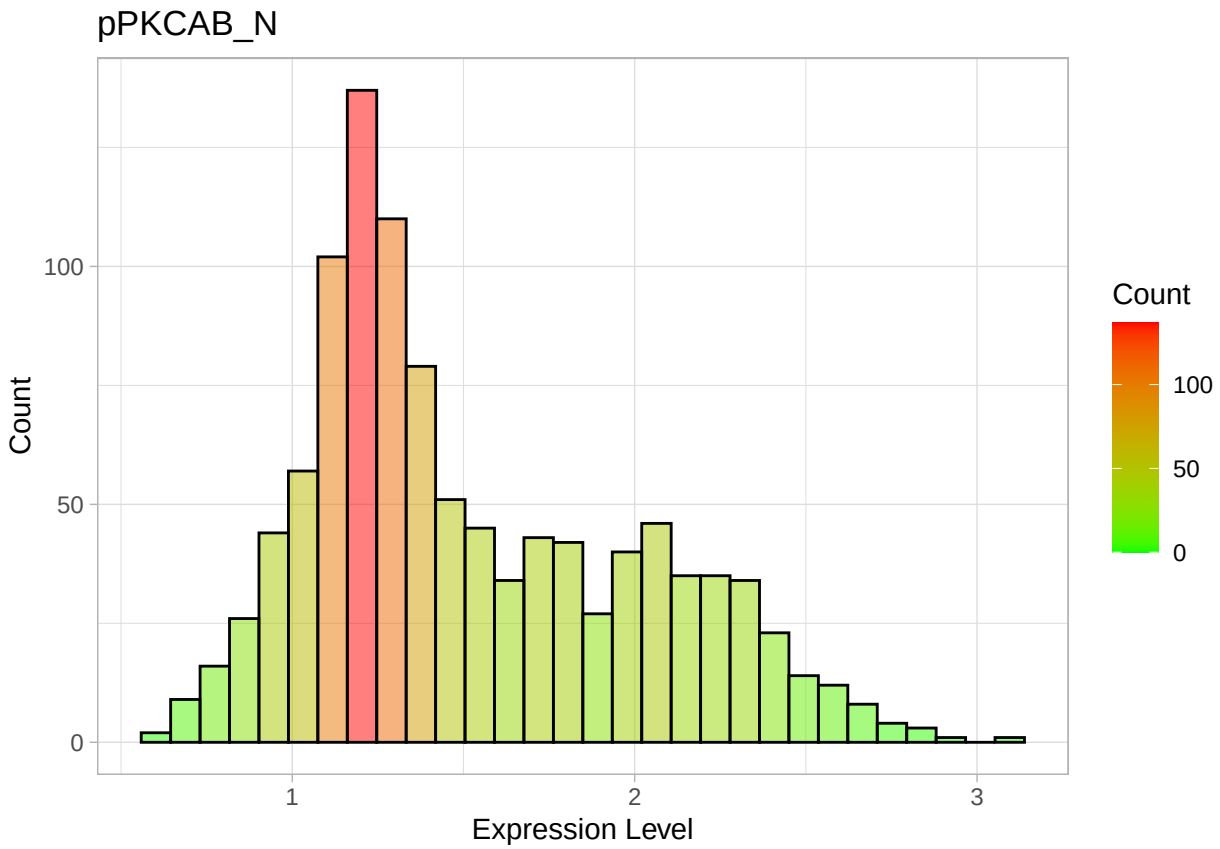
pNR1_N



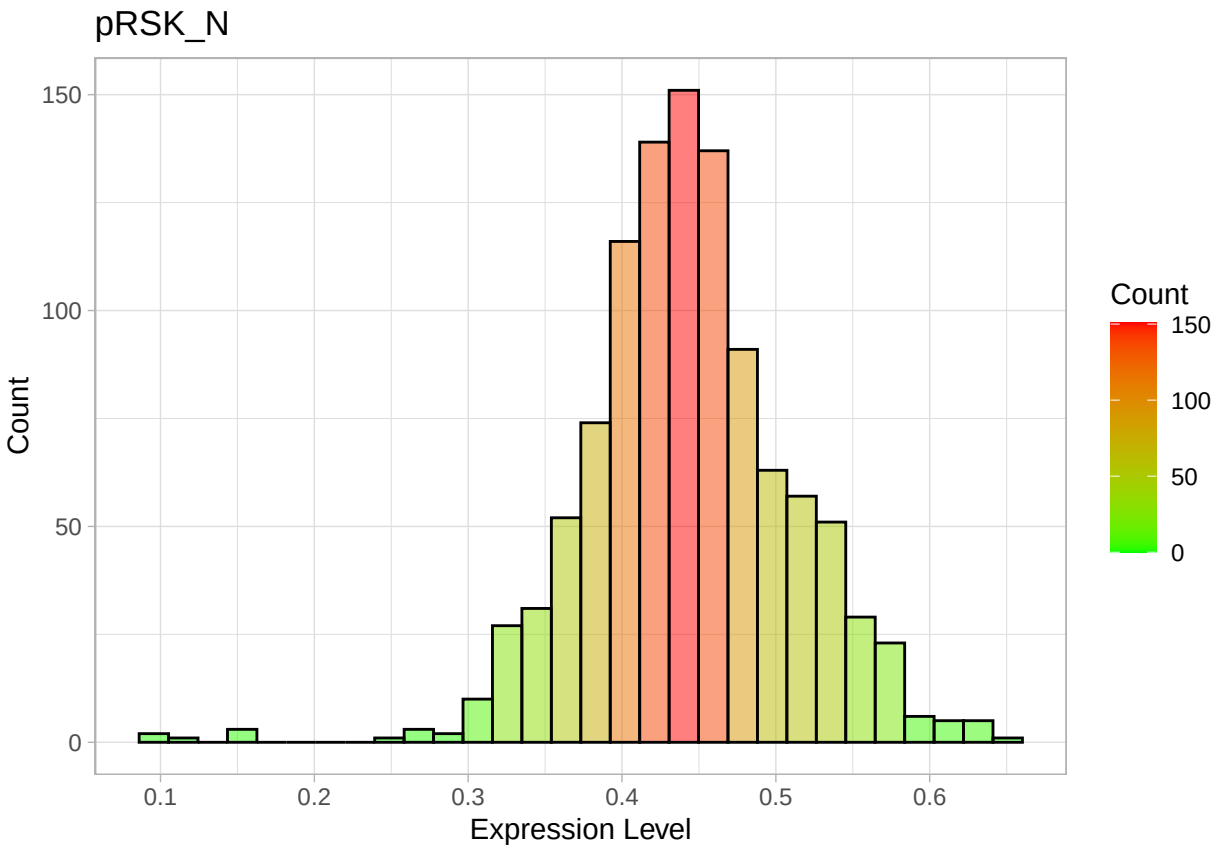
pNR2A_N



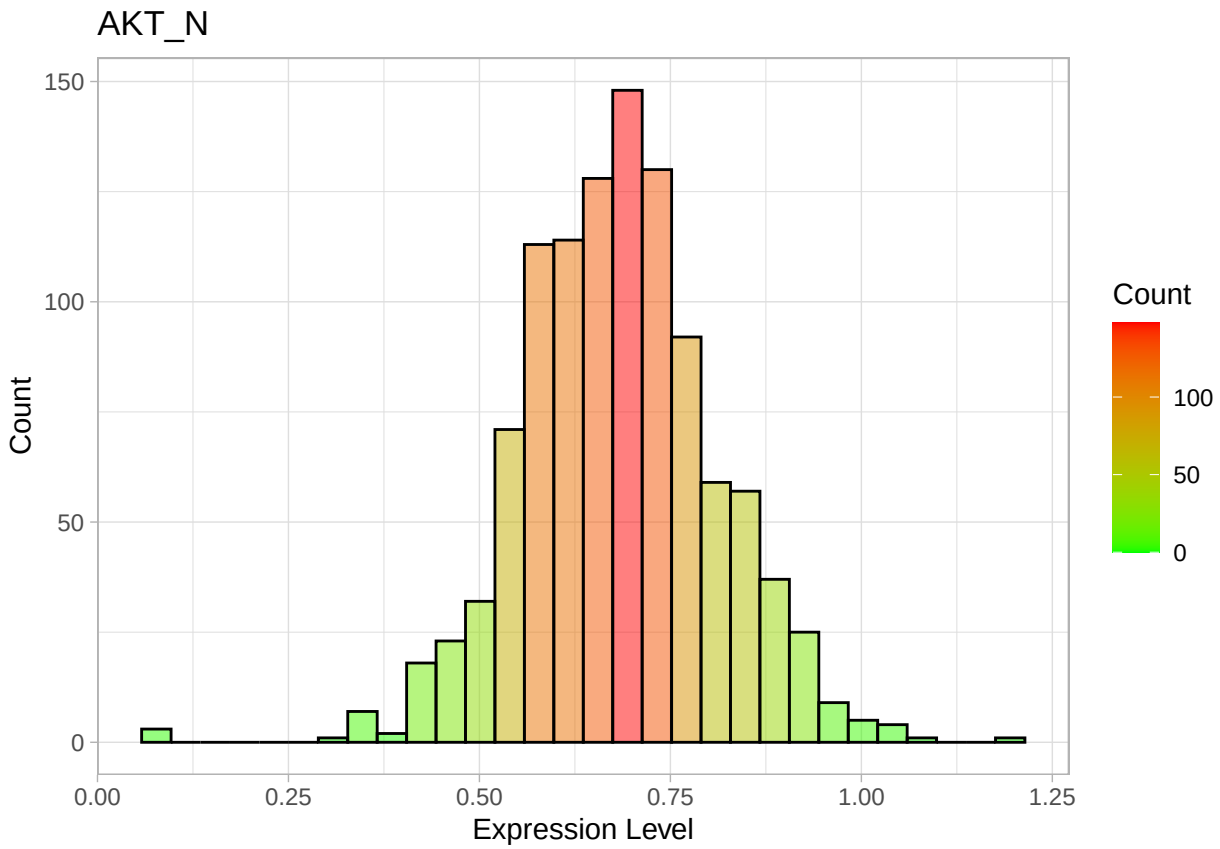
pNR2B_N



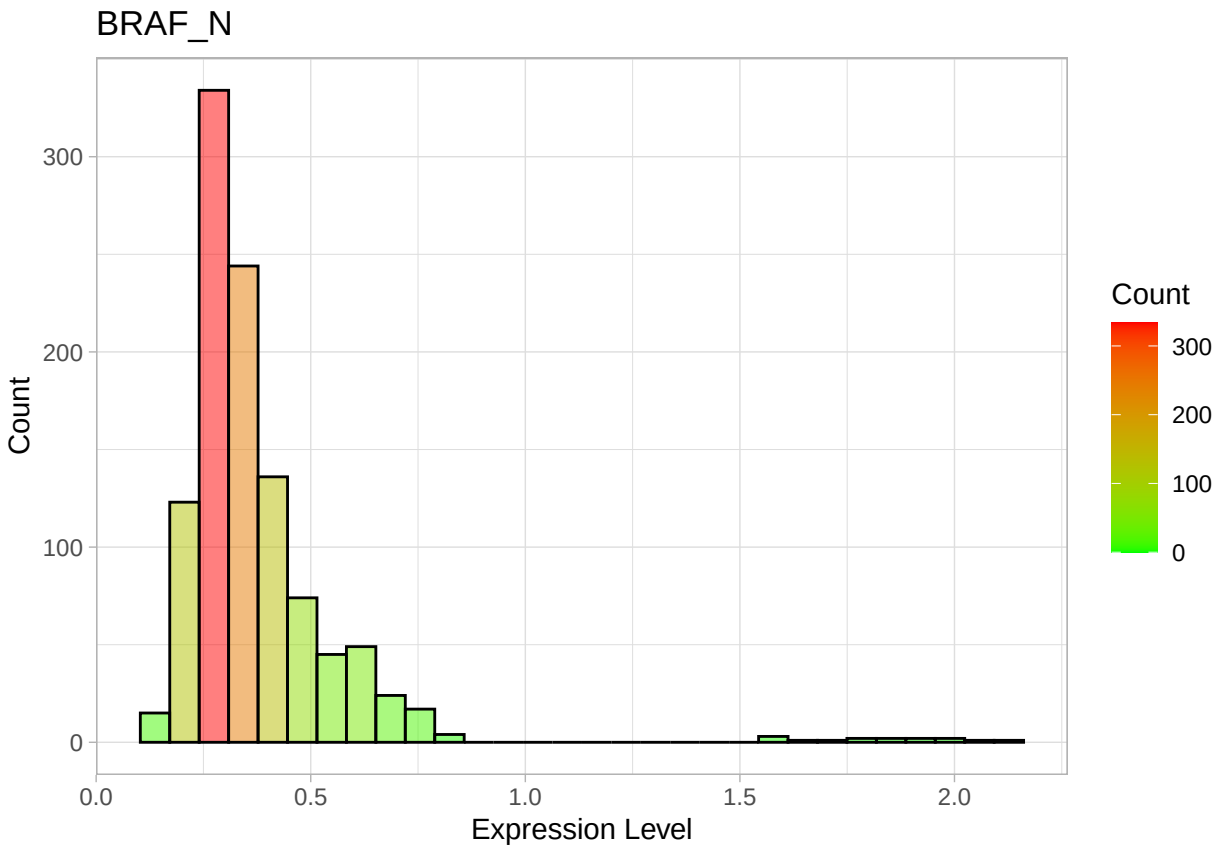
pPKCAB_N


pRSK_N
150
100
50
0
Count
0.1
0.2
0.3
0.4
0.5
0.6
Expression Level
Count
150
100
50
0

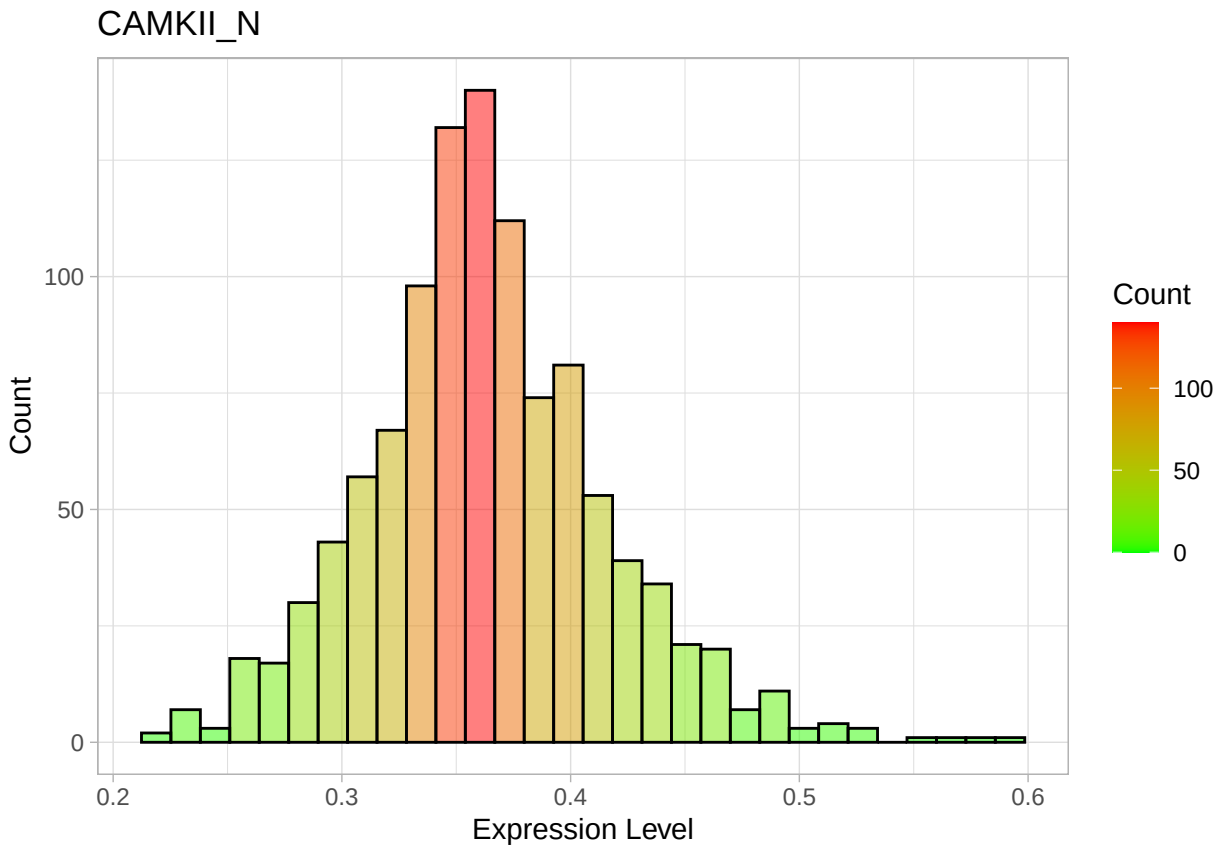
pRSK_N


AKT_N
150
100
50
0
Count
0.00
0.25
0.50
0.75
1.00
1.25
Expression Level
Count
100
50
0

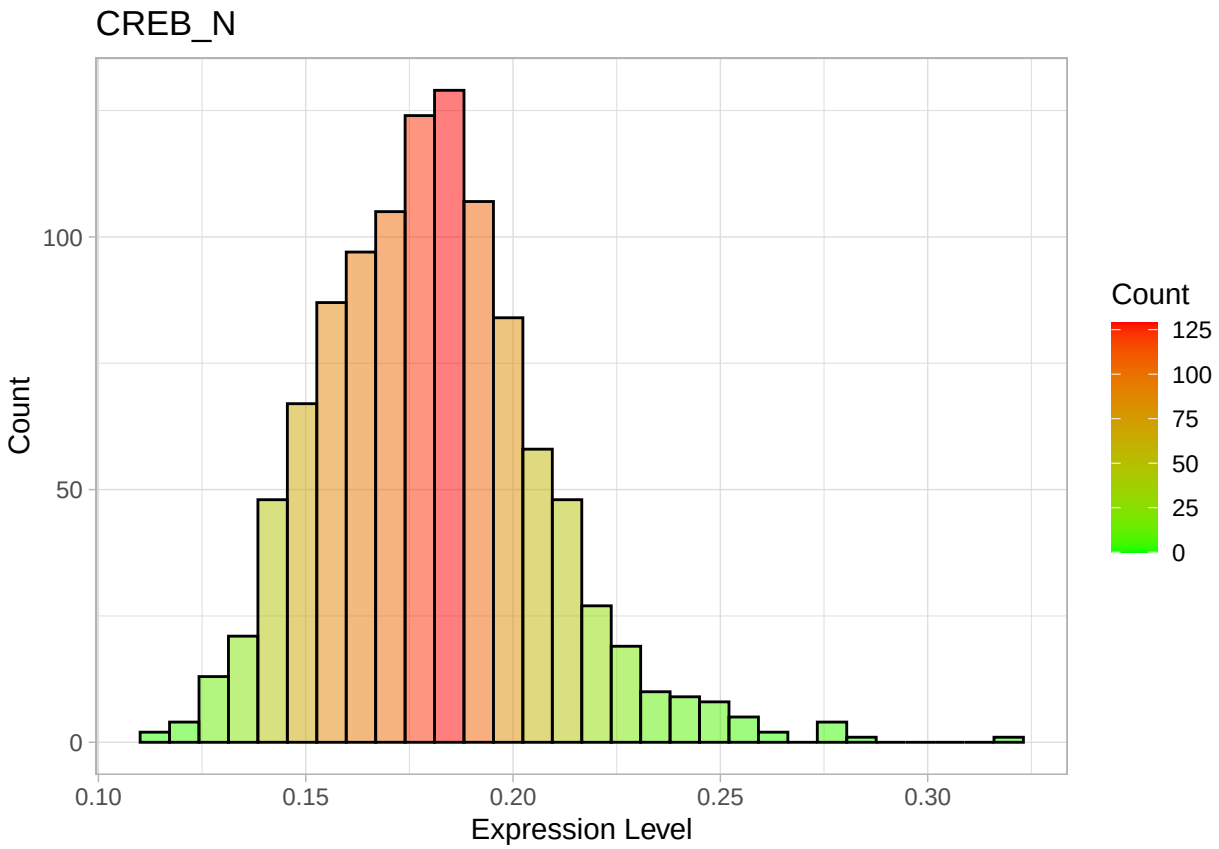
AKT_N


BRAF_N
Count
300
200
100
0
0.0
0.5
1.0
1.5
2.0
Expression Level
Count
300
200
100
0

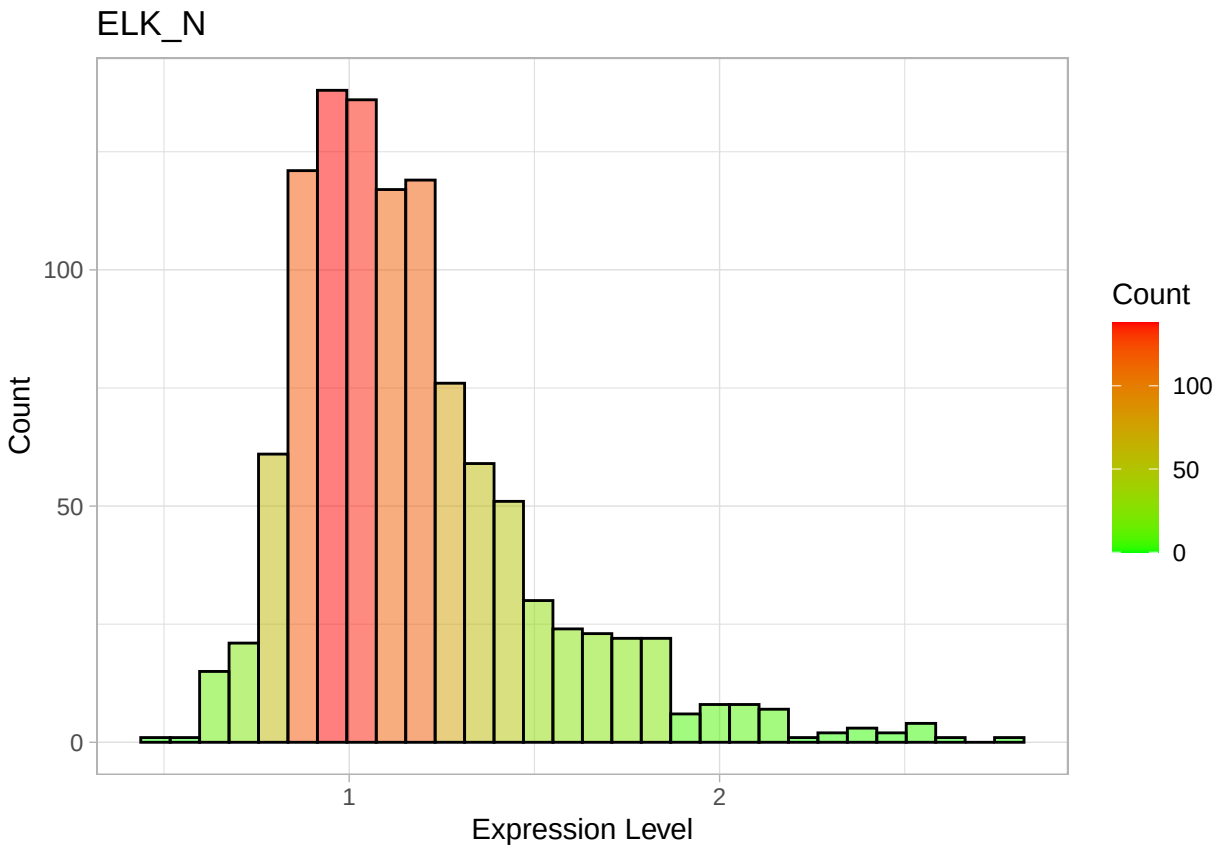
BRAF_N


CAMKII_N
Count
100
50
0
0.2
0.3
0.4
0.5
0.6
Expression Level
Count
100
50
0

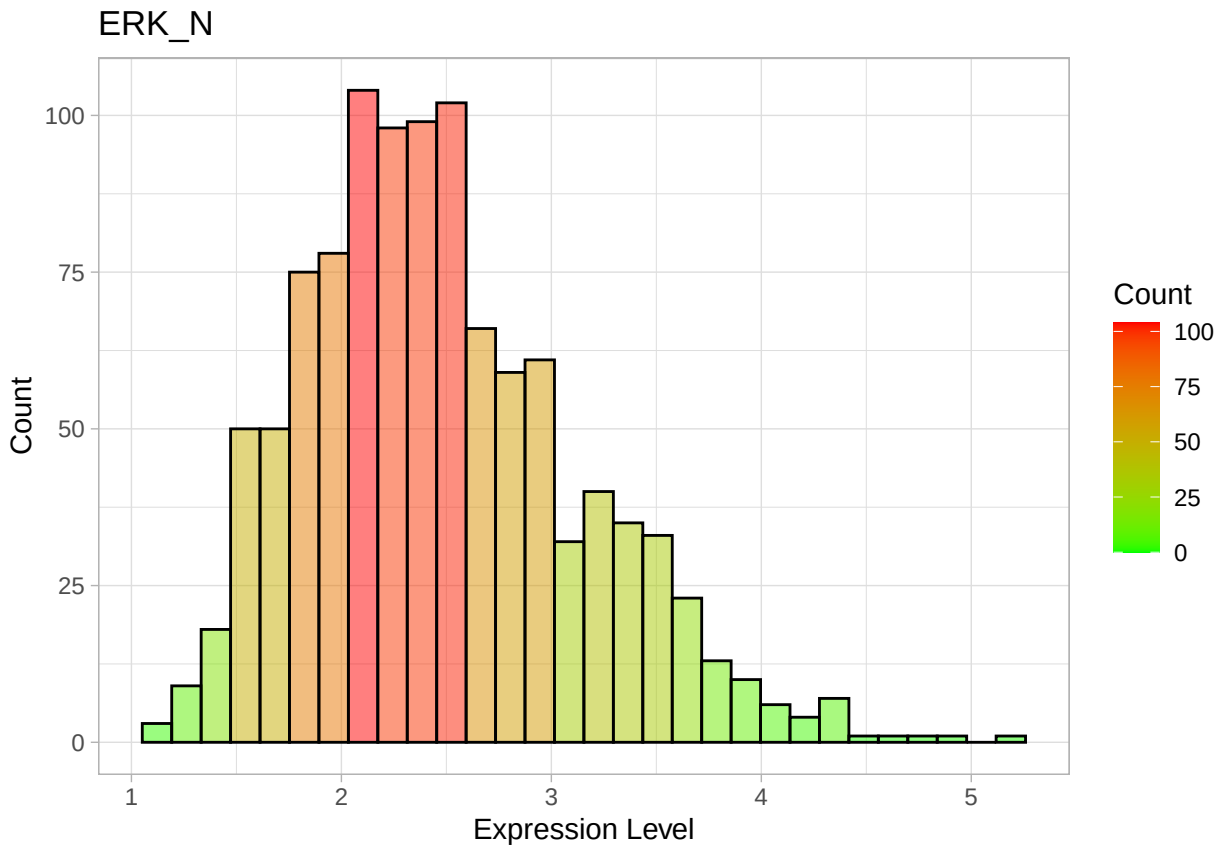
CAMKII_N


CREB_N
Count
100
50
0
0.10
0.15
0.20
0.25
0.30
Expression Level
Count
125
100
75
50
25
0

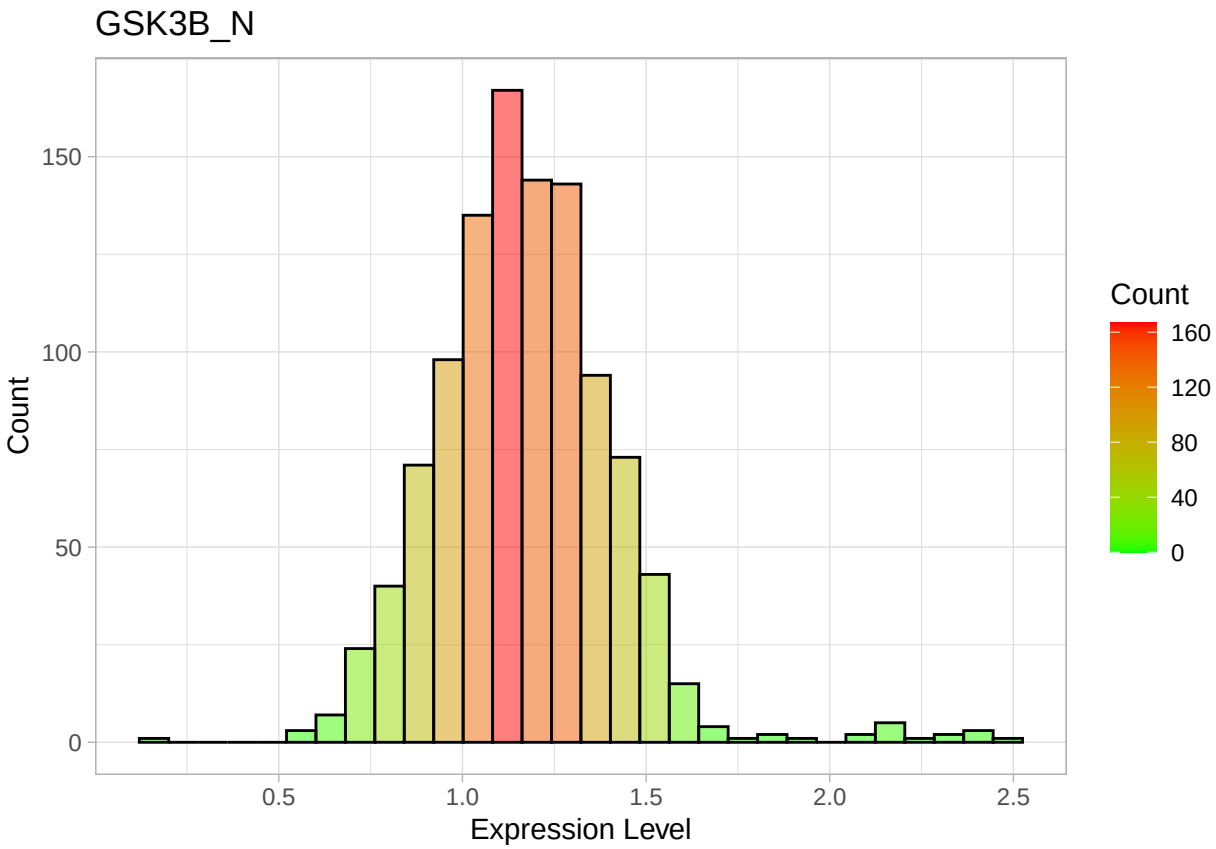
CREB_N



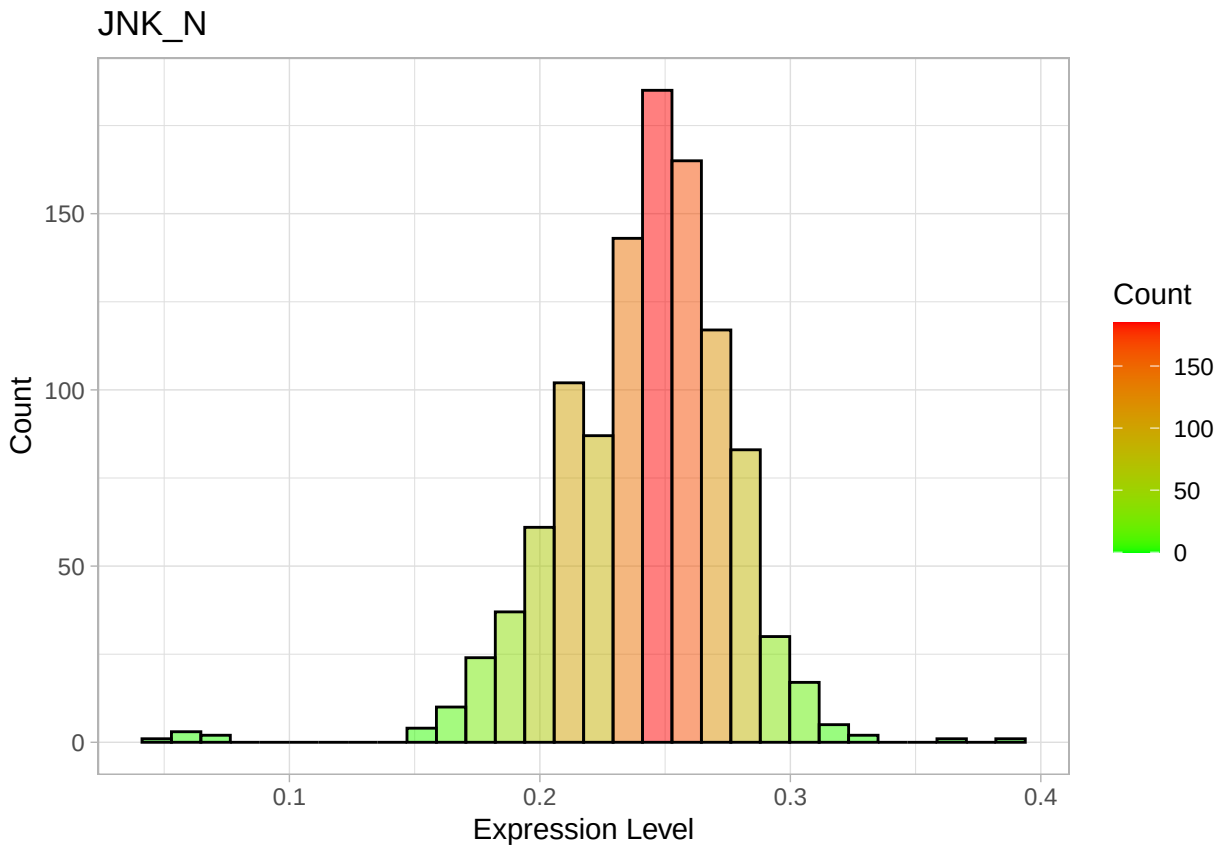
ELK_N


ERK_N
Count
100
75
50
25
0
1
2
3
4
5
Expression Level
Count
100
75
50
25
0

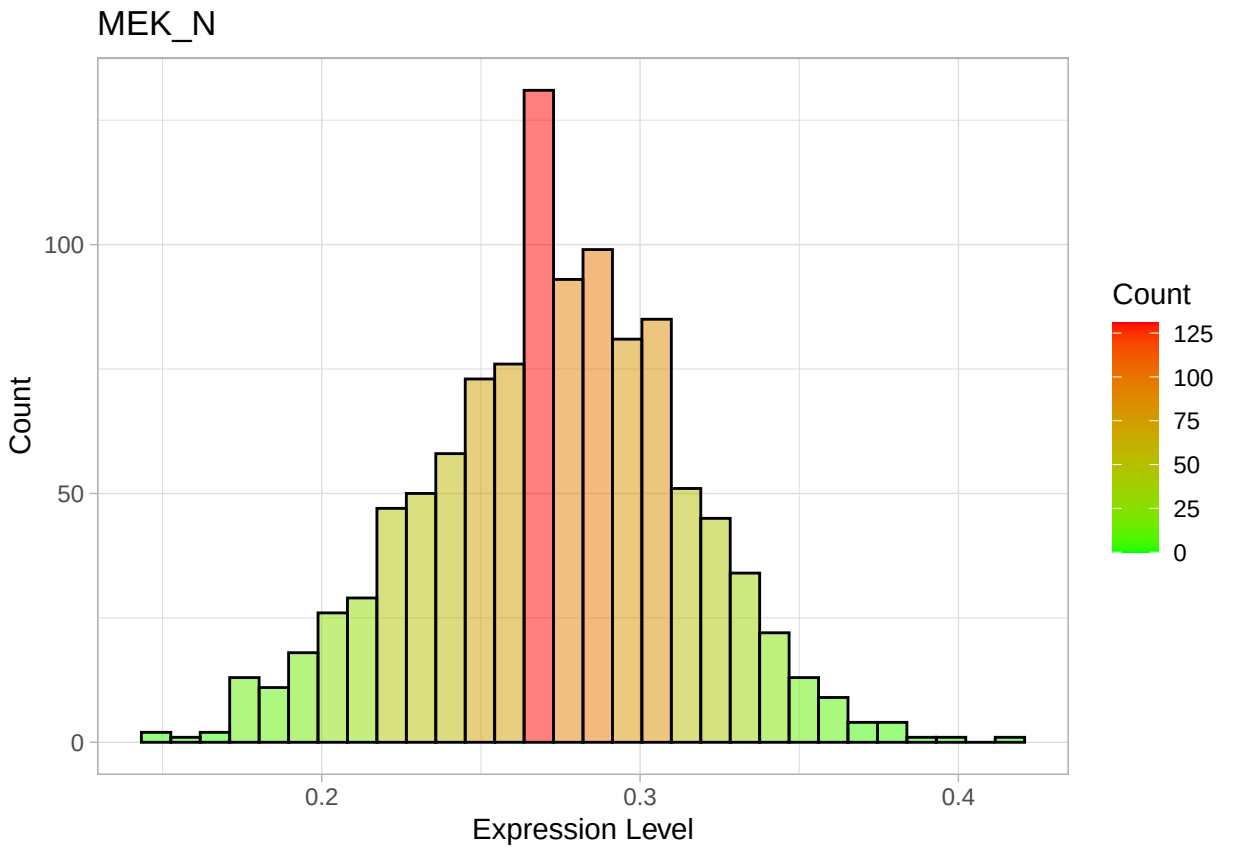
ERK_N


GSK3B_N
150
100
50
0
Count
0.5
1.0
1.5
2.0
2.5
Expression Level
Count
160
120
80
40
0

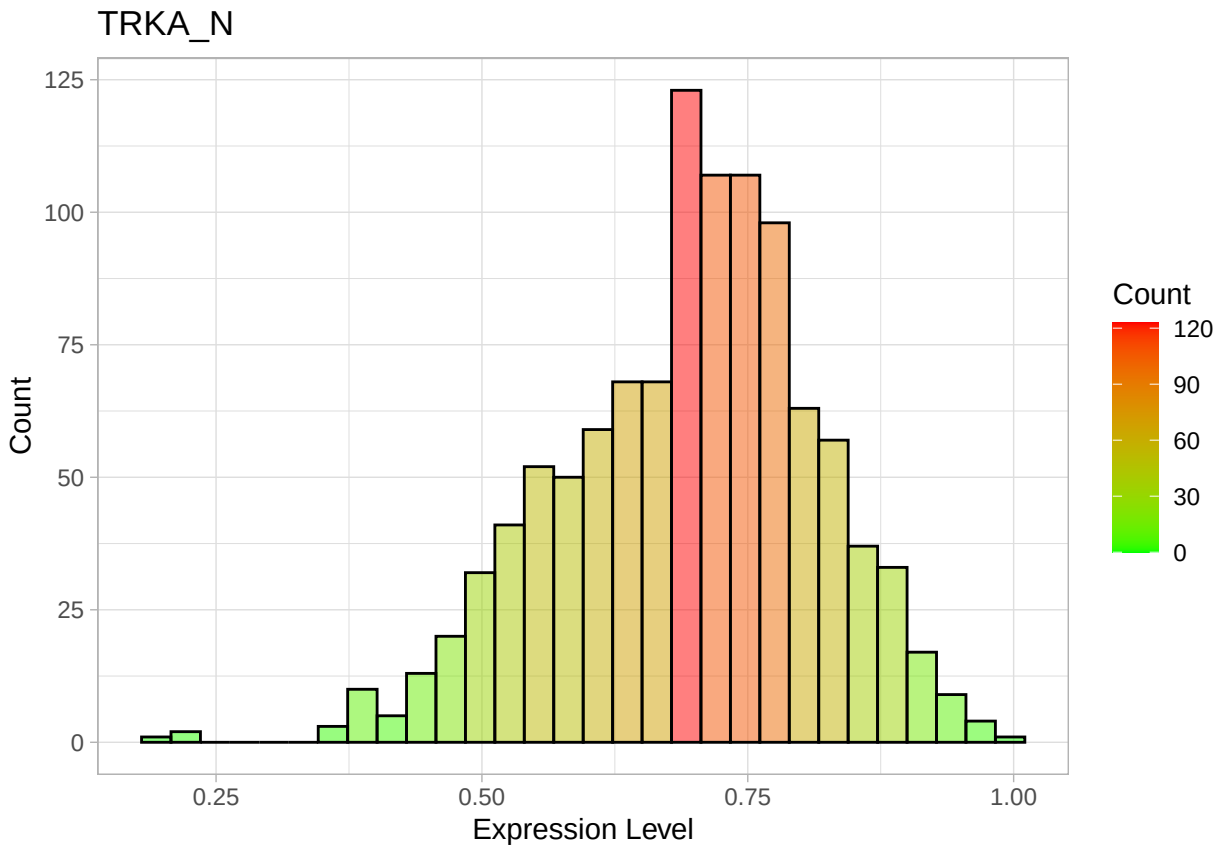
GSK3B_N


JNK_N
Count
150
100
50
0
0.1
0.2
0.3
0.4
Expression Level
Count
150
100
50
0

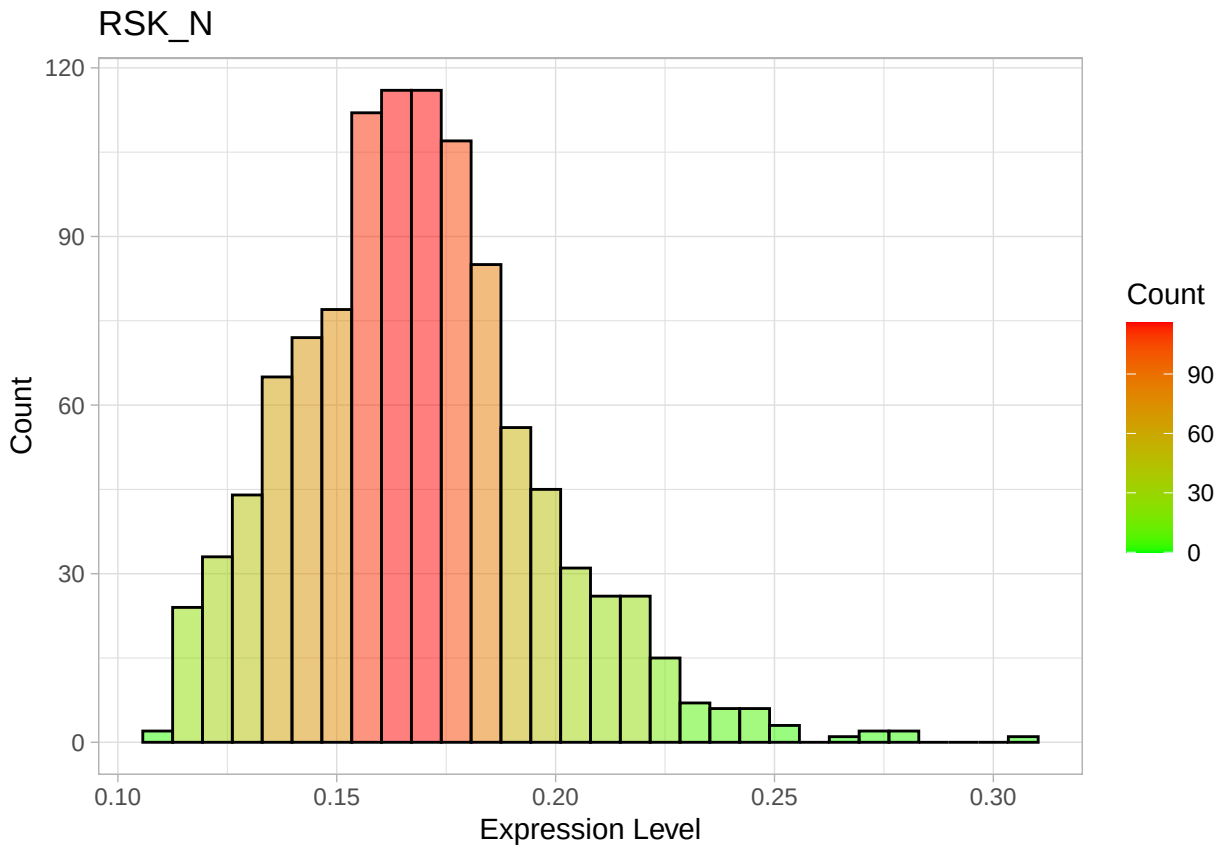
JNK_N


MEK_N
Count
100
50
0
0.2
0.3
0.4
Expression Level
Count
125
100
75
50
25
0

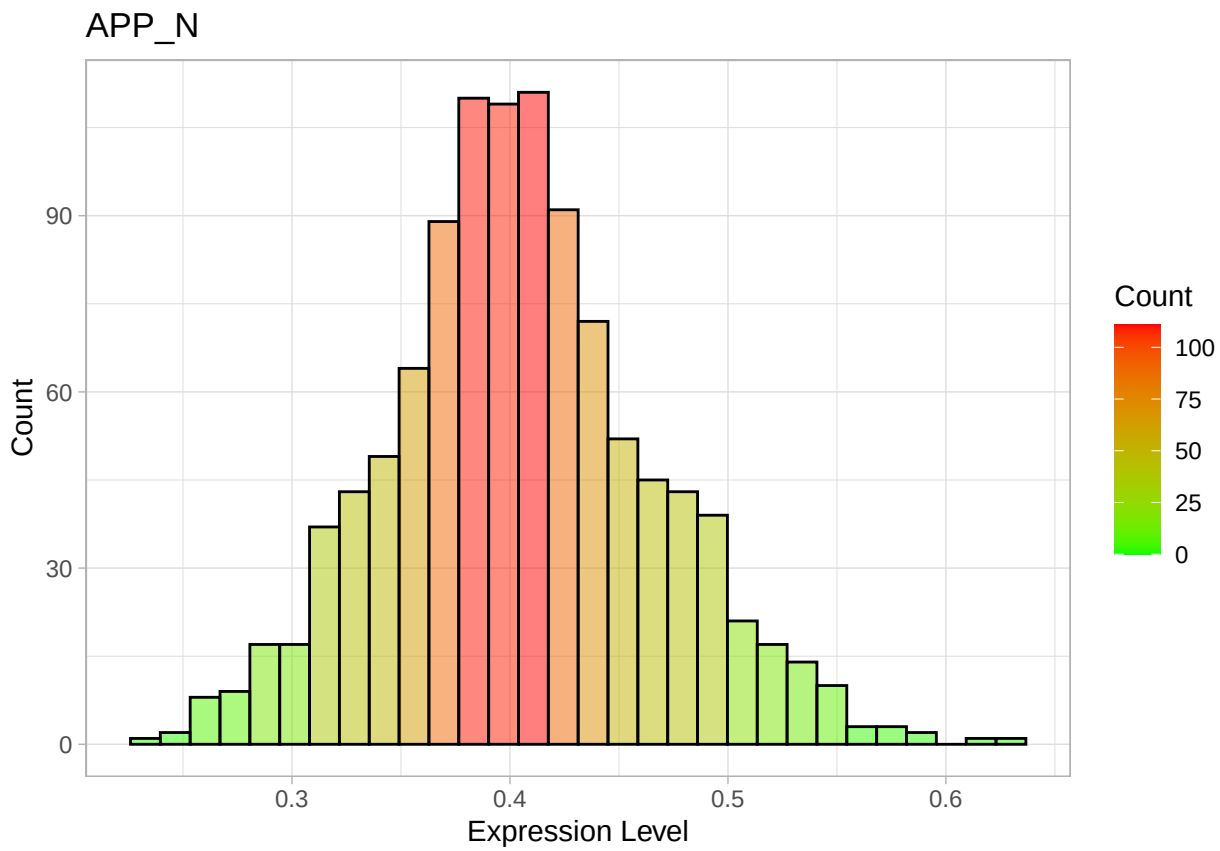
MEK_N


TRKA_N
125
100
75
50
25
0
Count
0.25
0.50
0.75
1.00
Expression Level
Count
120
90
60
30
0

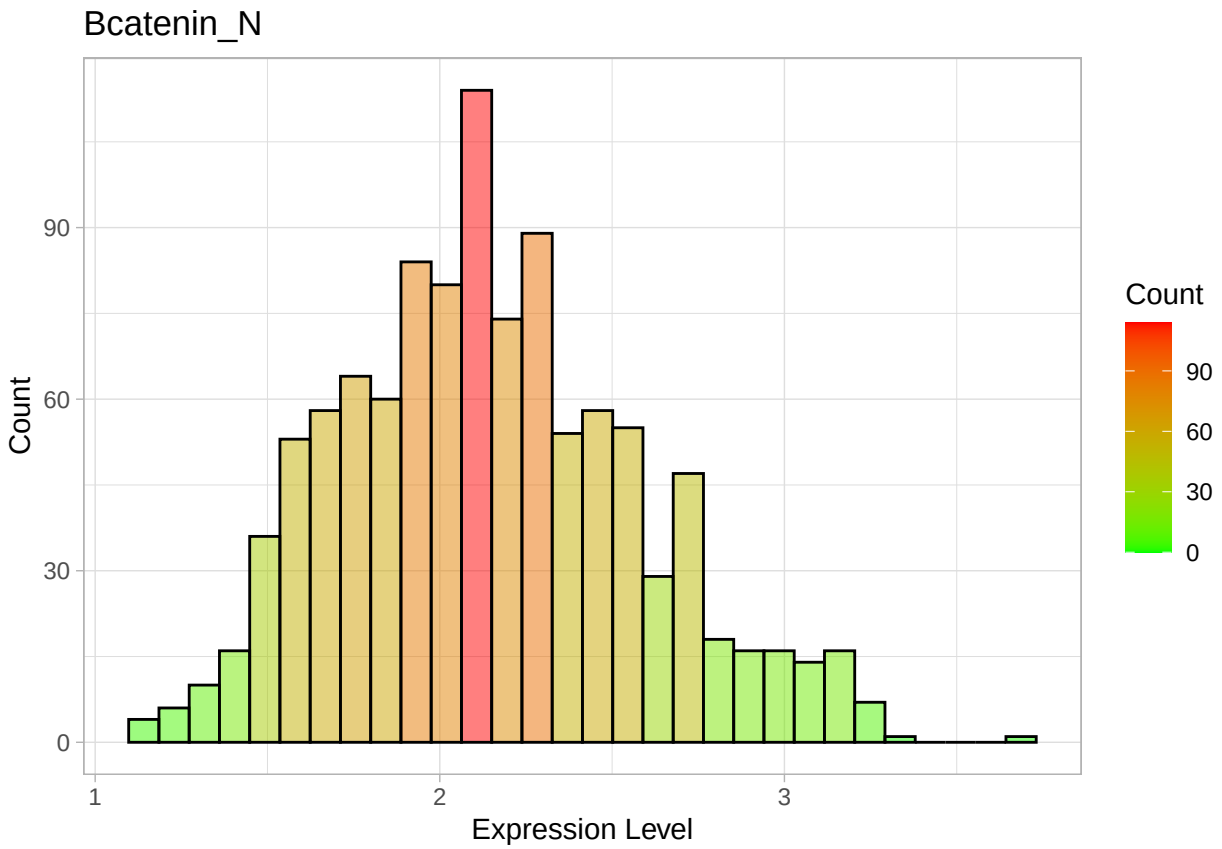
TRKA_N


RSK_N
120
90
60
30
0
Count
0.10
0.15
0.20
0.25
0.30
Expression Level
Count
90
60
30
0

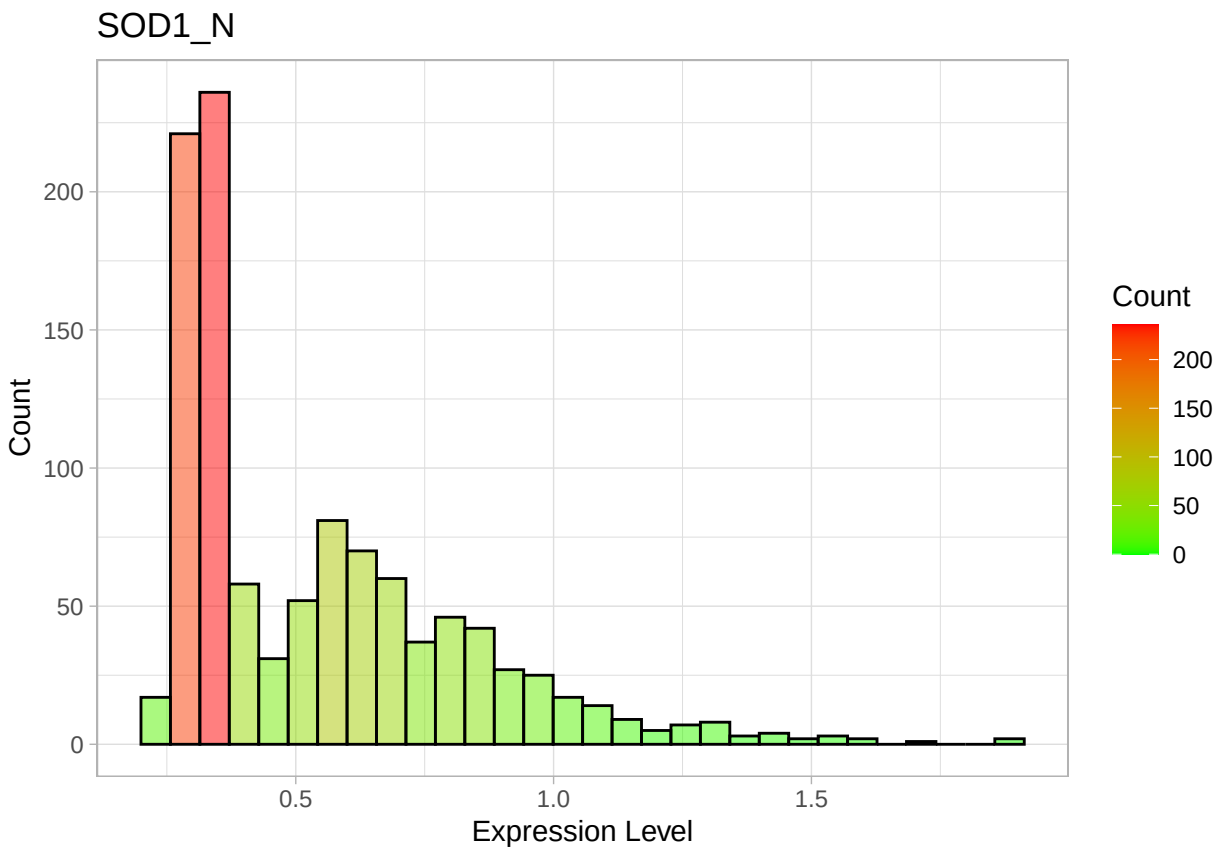
RSK_N


APP_N
Count
90
60
30
0
0.3
0.4
0.5
0.6
Expression Level
Count
100
75
50
25
0

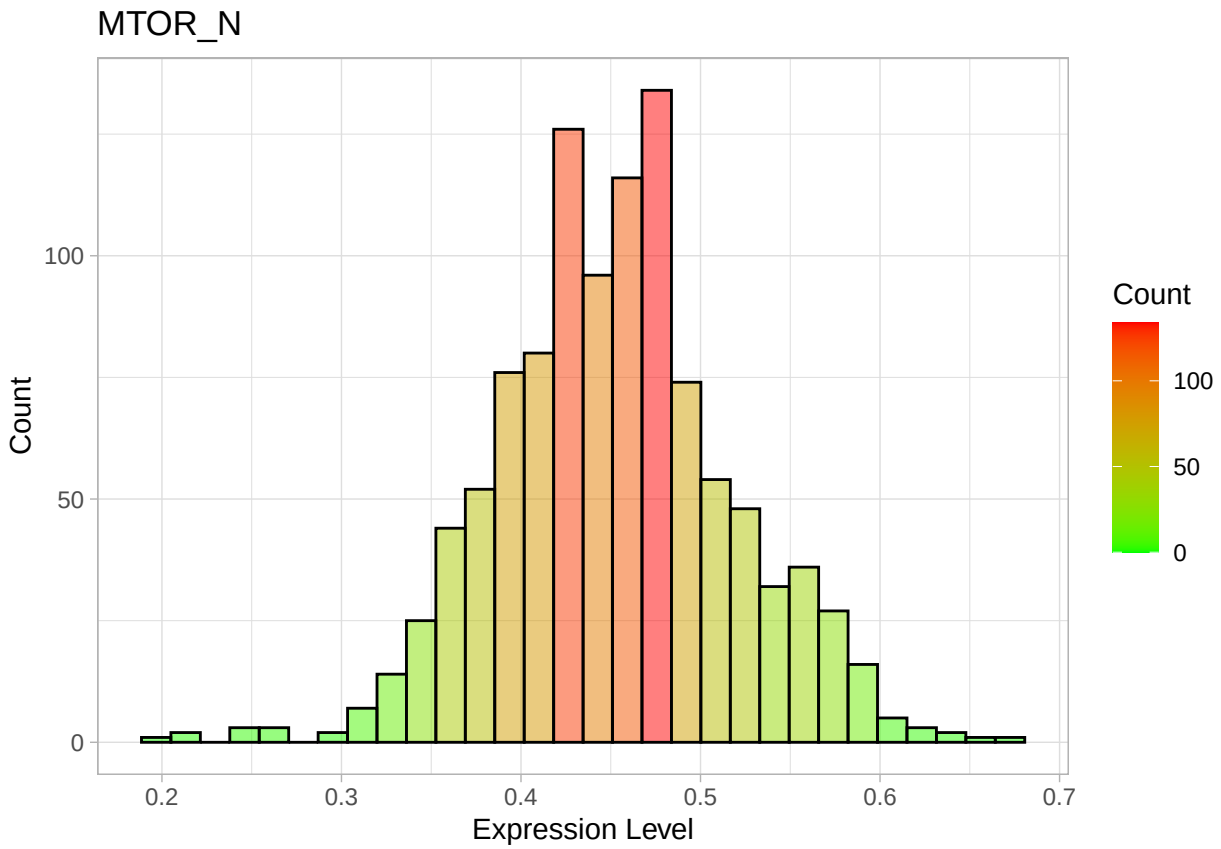
APP_N


Bcatenin_N
Count
90
60
30
0
1
2
3
Expression Level
Count
90
60
30
0

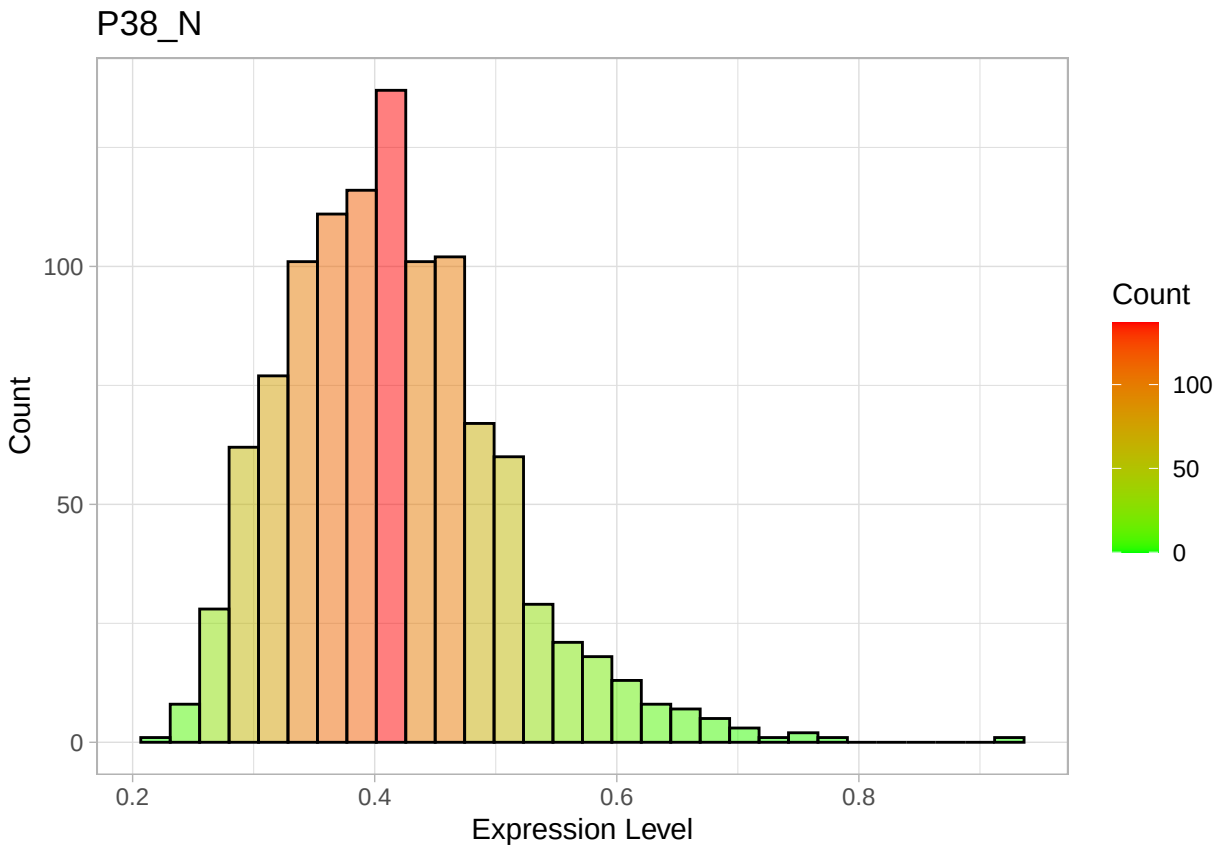
Bcatenin_N


SOD1_N
Count
200
150
100
50
0
0.5
1.0
1.5
Expression Level
Count
200
150
100
50
0

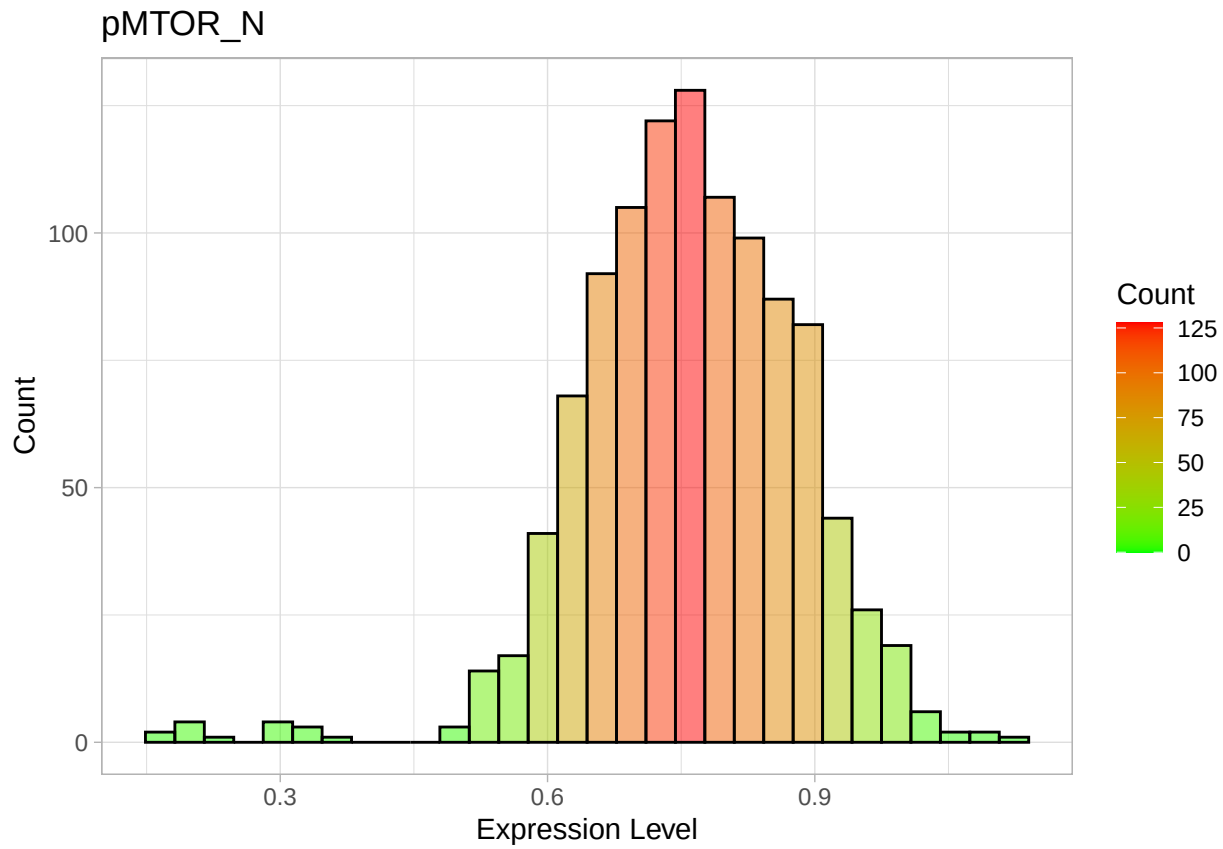
SOD1_N


MTOR_N
Count
100
50
0
0.2
0.3
0.4
0.5
0.6
0.7
Expression Level
Count
100
50
0

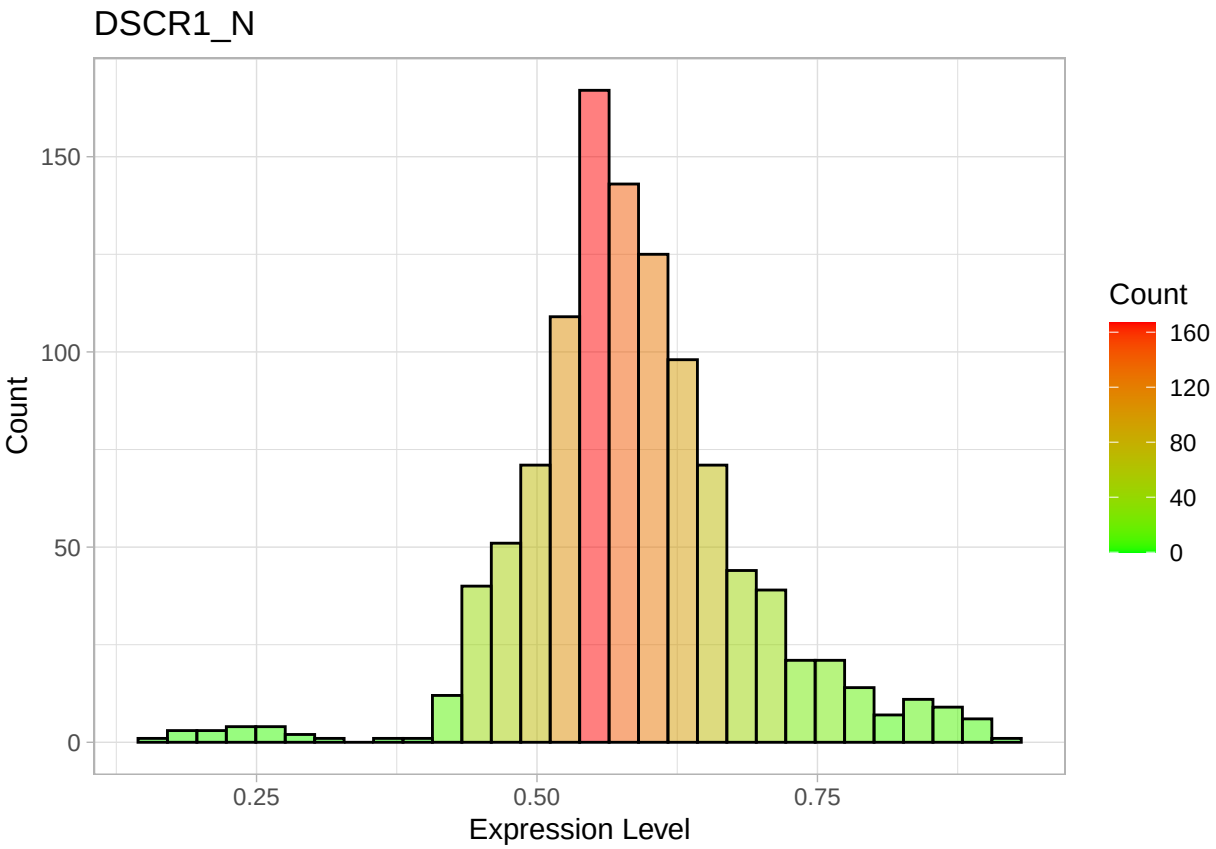
MTOR_N



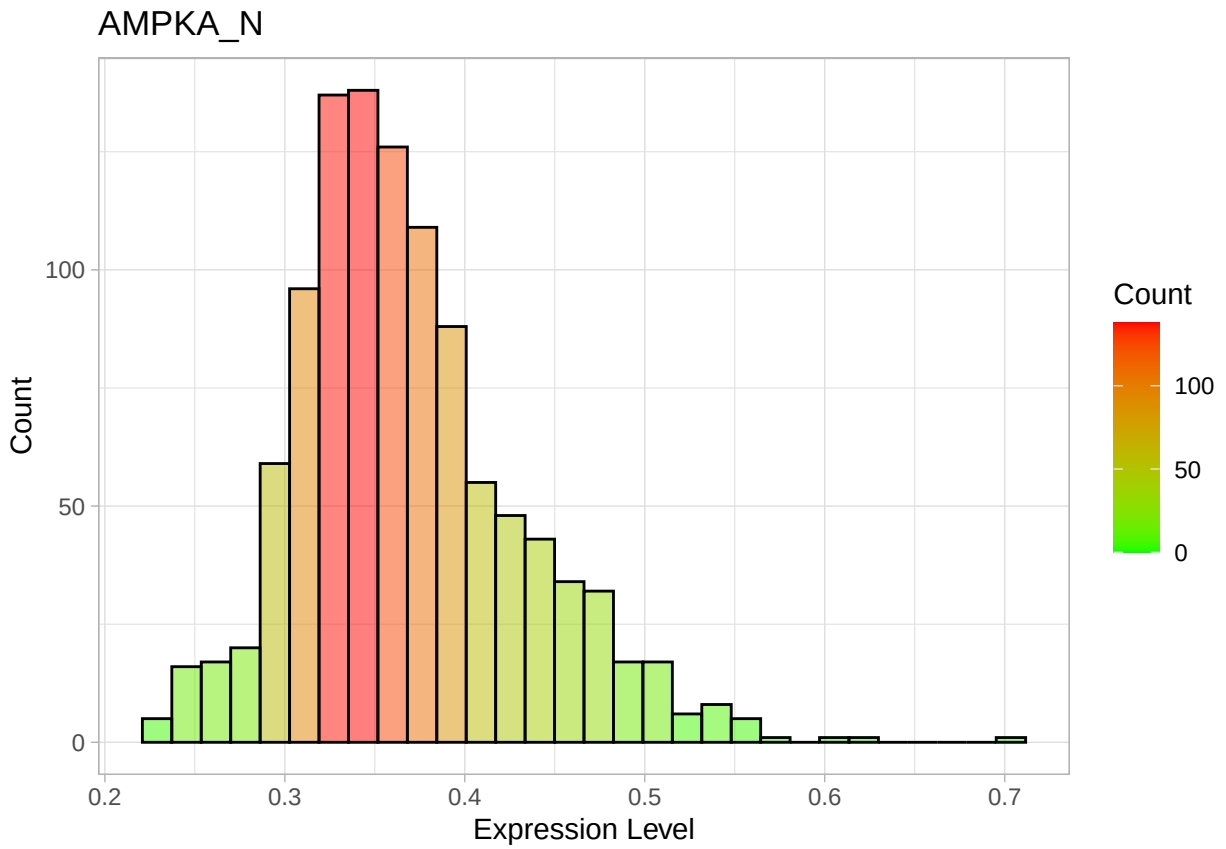
P38_N


pMTOR_N
Count
100
50
0
0.3
0.6
0.9
Expression Level
Count
125
100
75
50
25
0

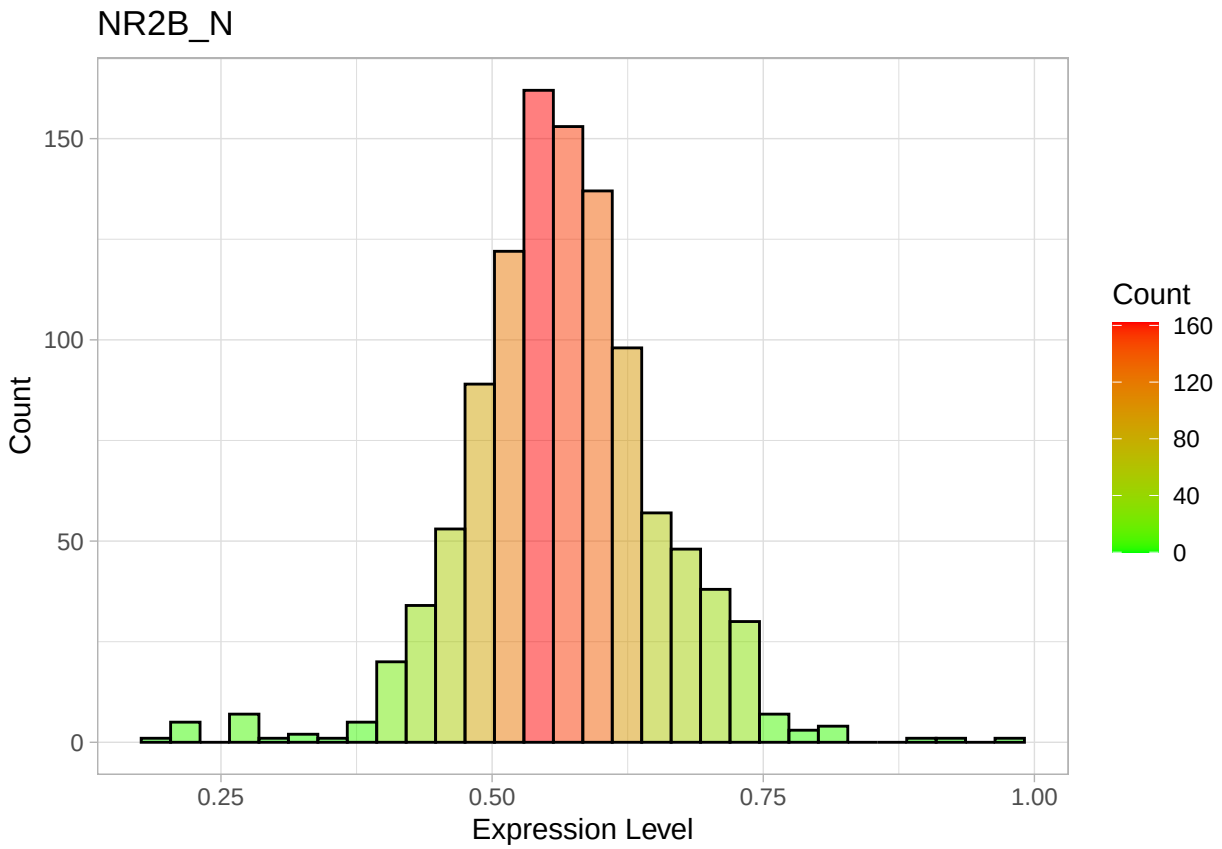
pMTOR_N


DSCR1_N
150
100
50
0
Count
0.25
0.50
0.75
Expression Level
Count
160
120
80
40
0

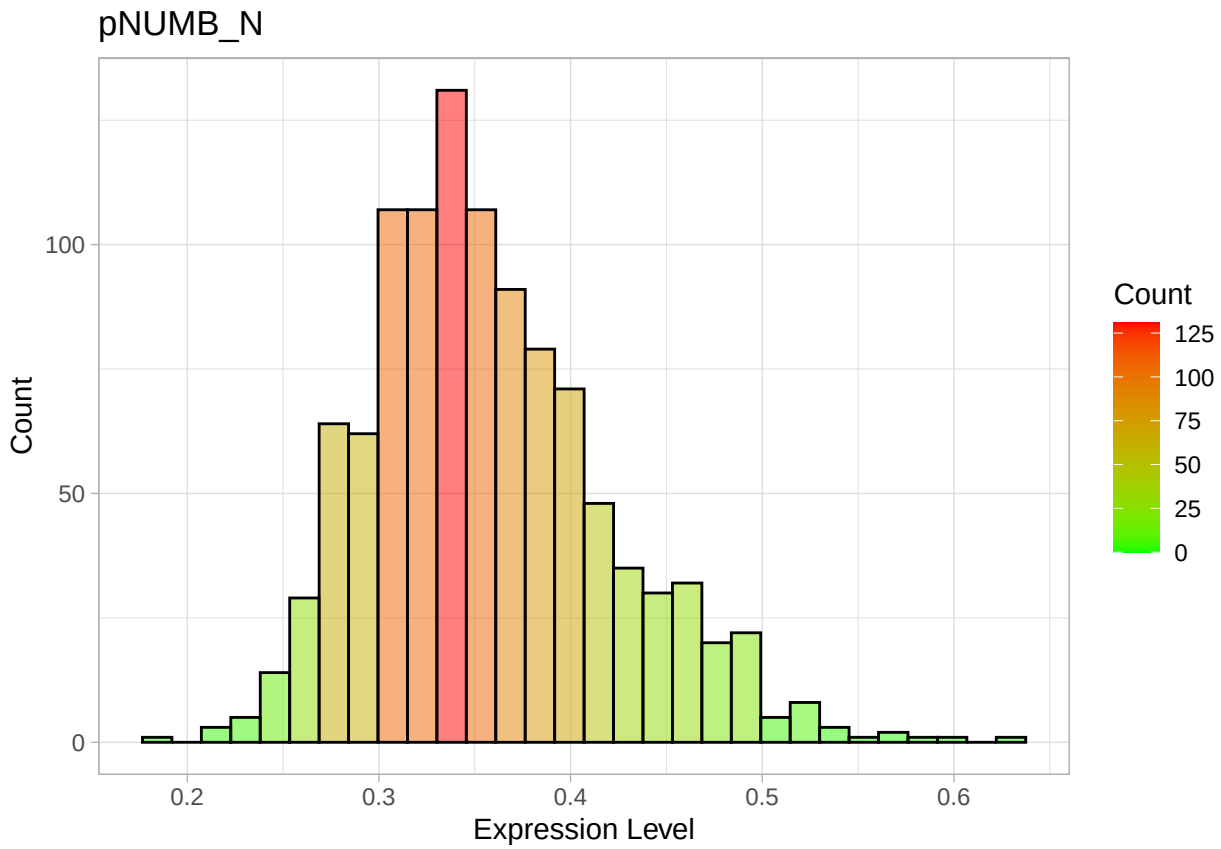
DSCR1_N


AMPKA_N
Count
100
50
0
0.2
0.3
0.4
0.5
0.6
0.7
Expression Level
Count
100
50
0

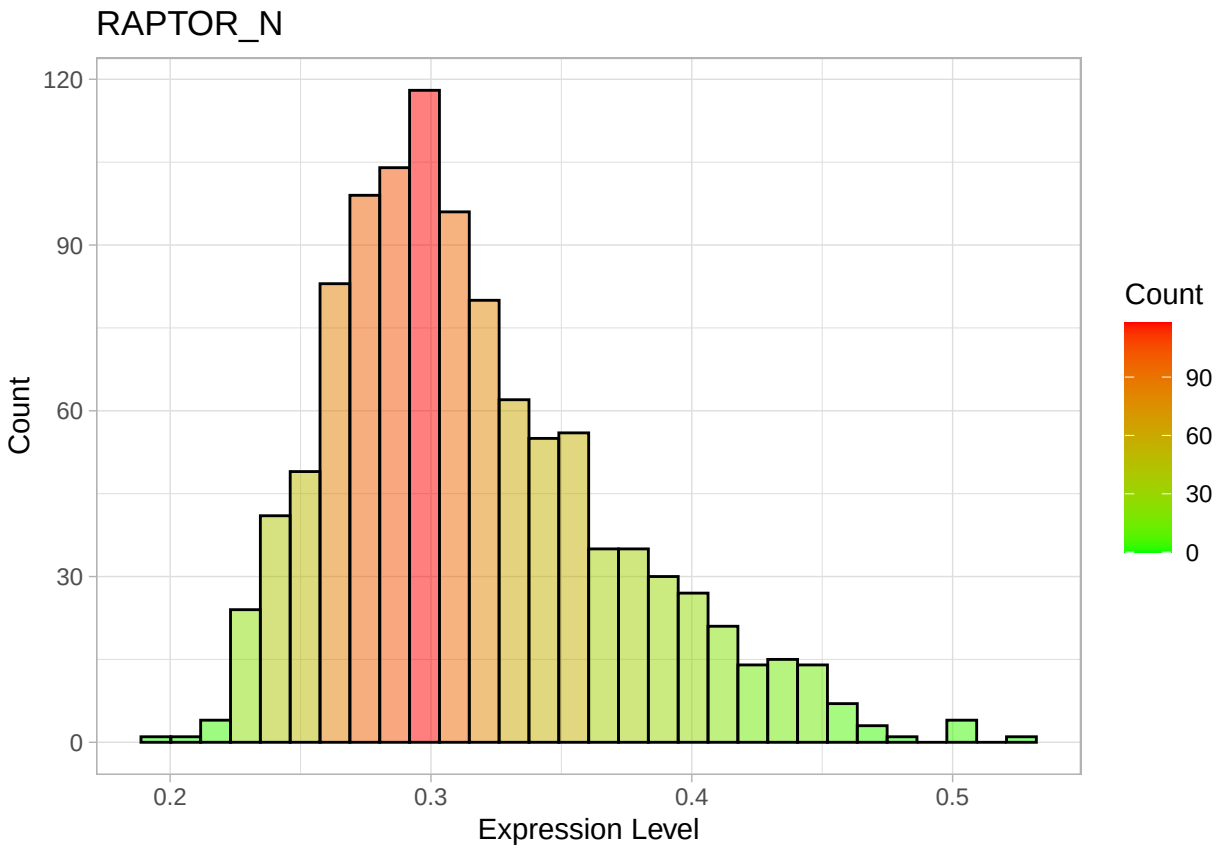
AMPKA_N


NR2B_N
150
100
50
0
Count
0.25
0.50
0.75
1.00
Expression Level
Count
160
120
80
40
0

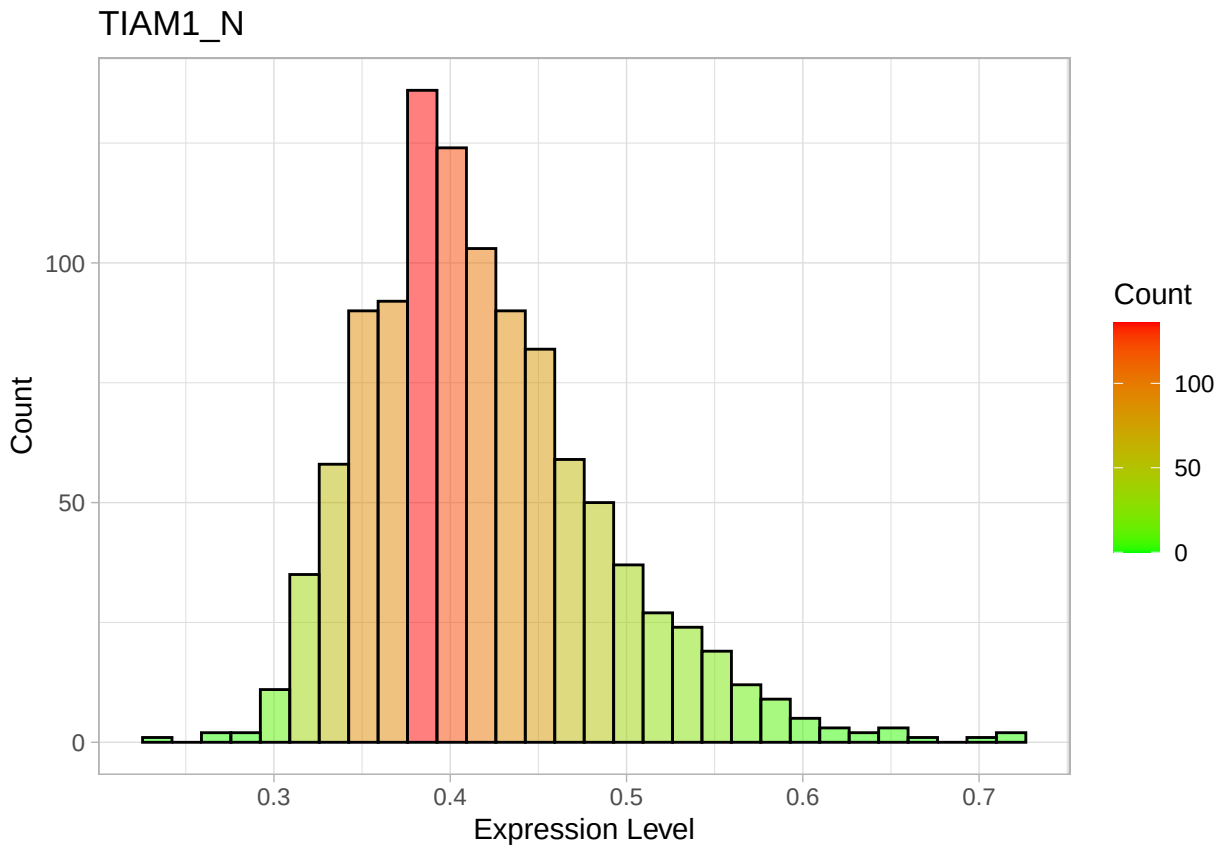
NR2B_N


pNUMB_N
Count
100
50
0
0.2
0.3
0.4
0.5
0.6
Expression Level
Count
125
100
75
50
25
0

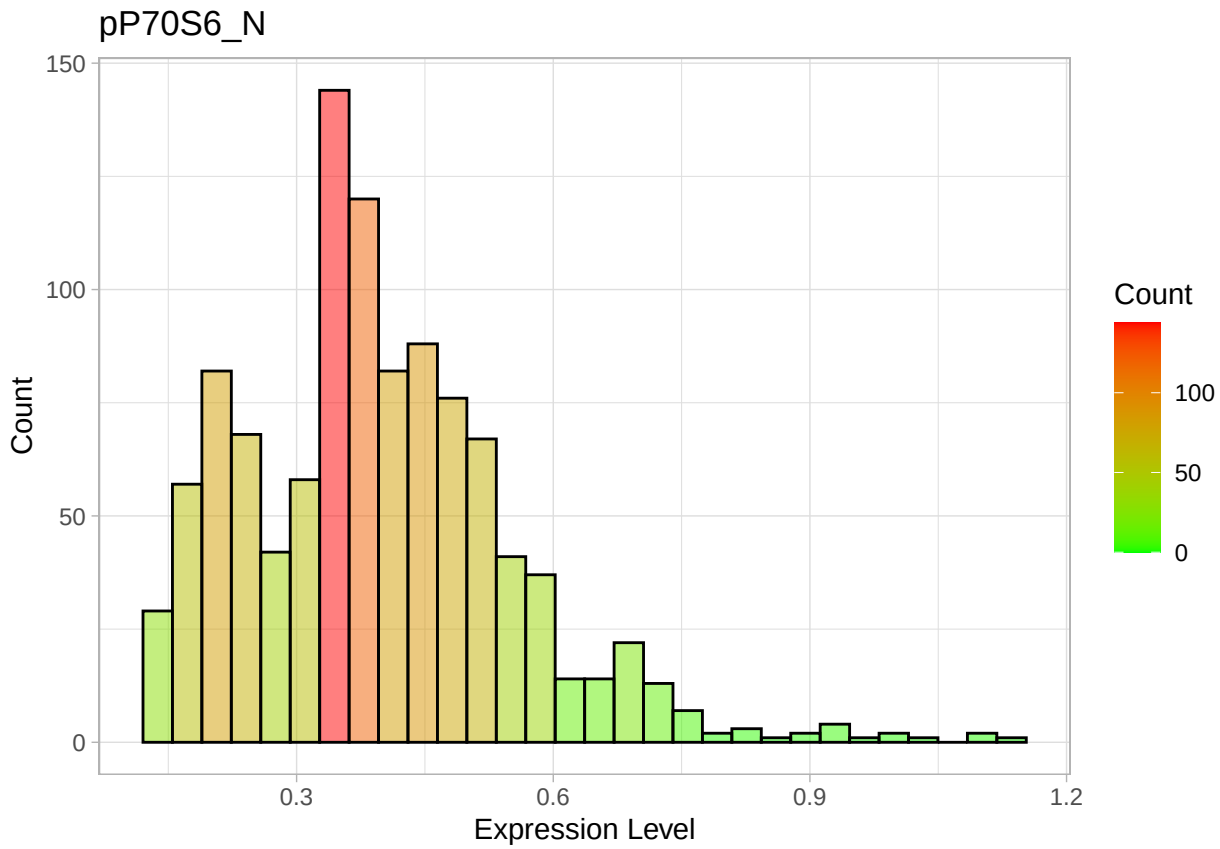
pNUMB_N


RAPTOR_N
120
90
60
30
0
Count
0.2
0.3
0.4
0.5
Expression Level
Count
90
60
30
0

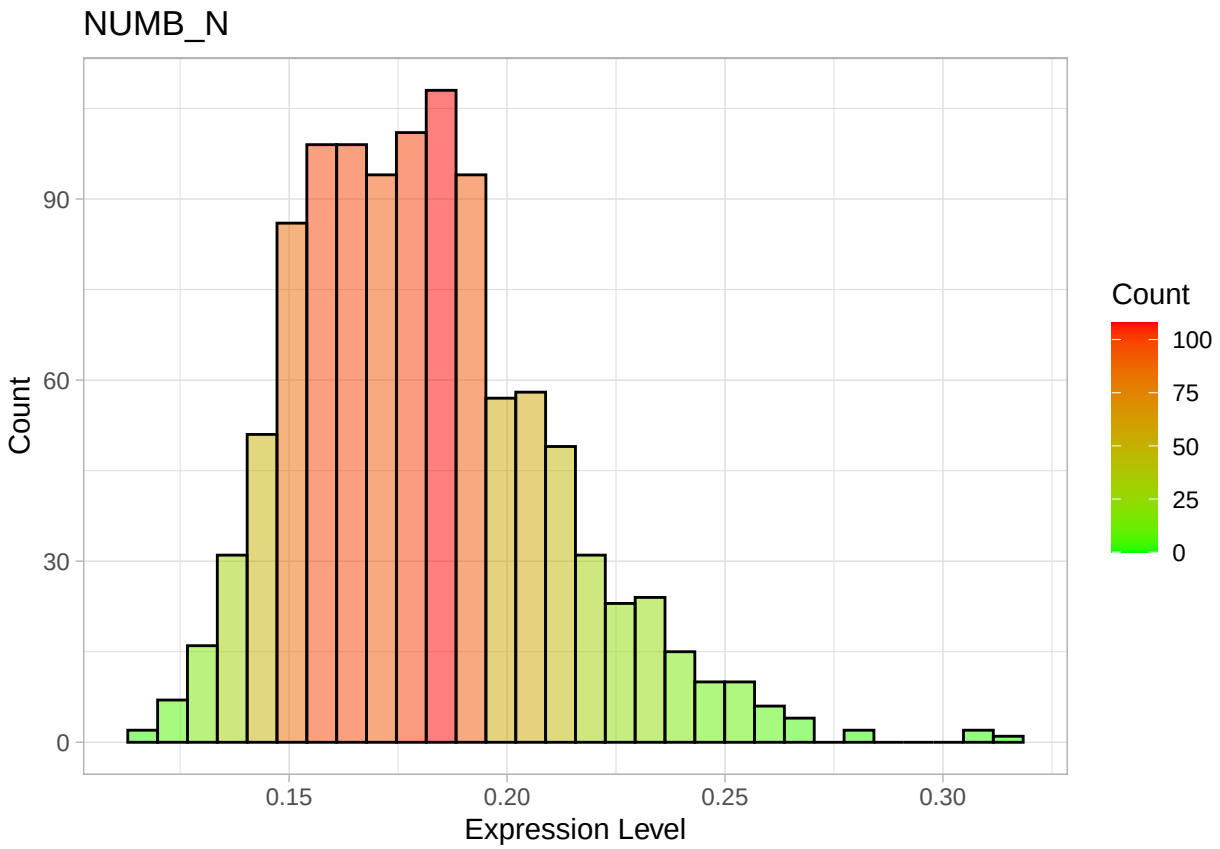
RAPTOR_N


TIAM1_N
Count
100
50
0
0.3
0.4
0.5
0.6
0.7
Expression Level
Count
100
50
0

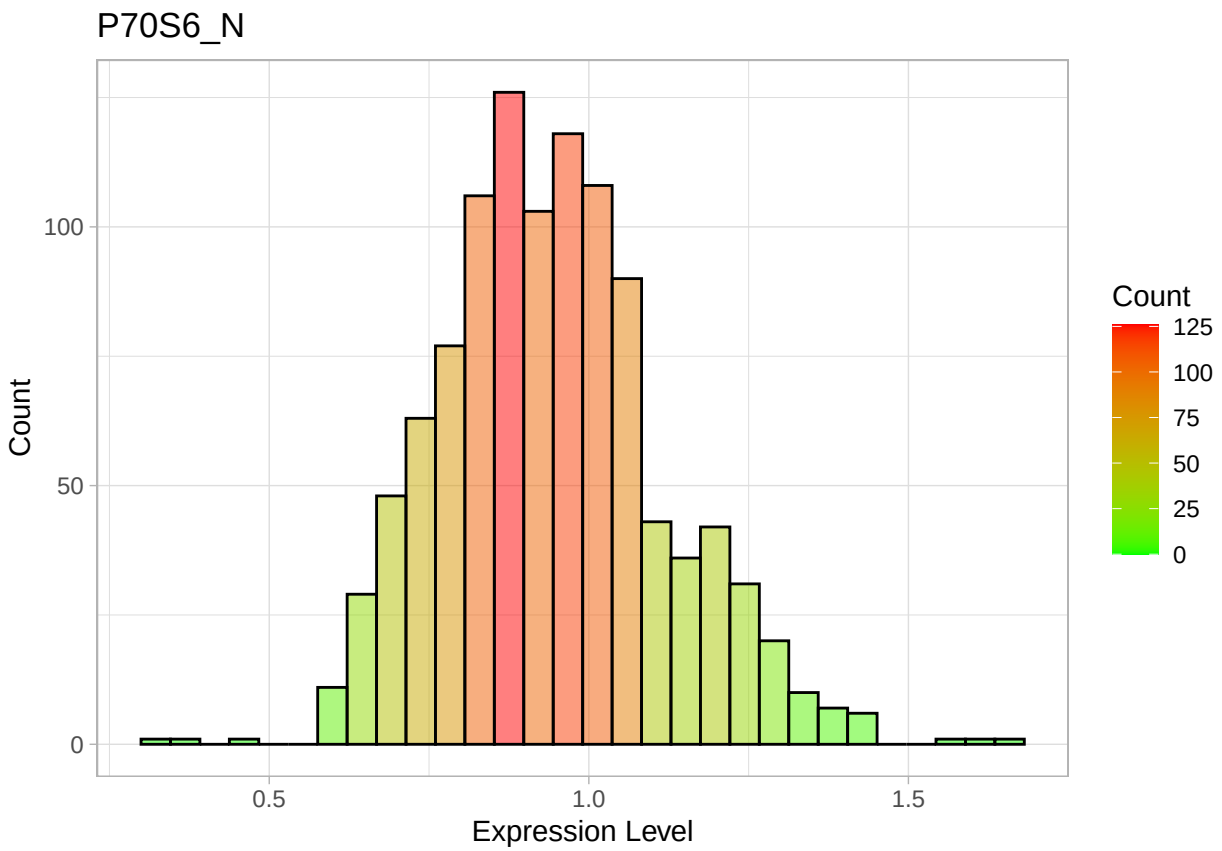
TIAM1_N


pP70S6_N
150
100
50
0
Count
0.3
0.6
0.9
1.2
Expression Level
Count
100
50
0

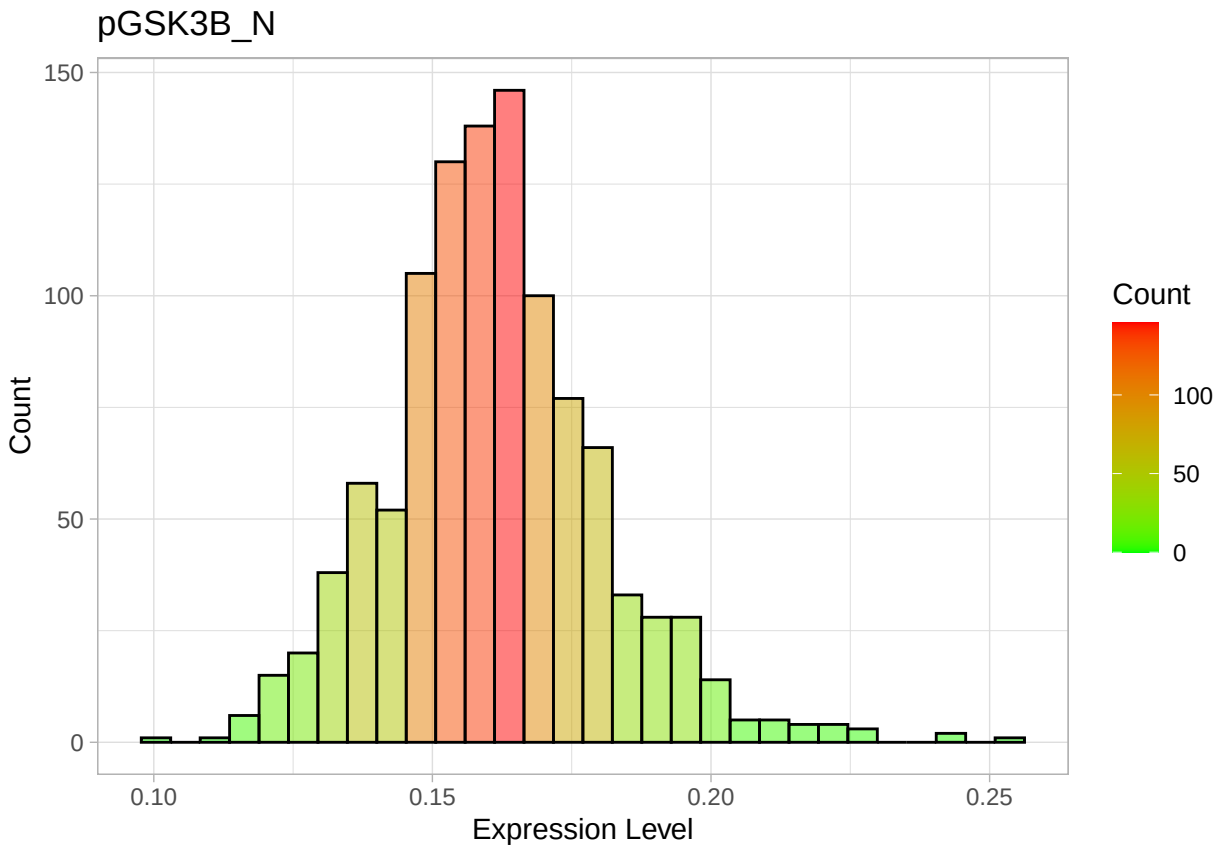
pP70S6_N


NUMB_N
Count
90
60
30
0
0.15
0.20
0.25
0.30
Expression Level
Count
100
75
50
25
0

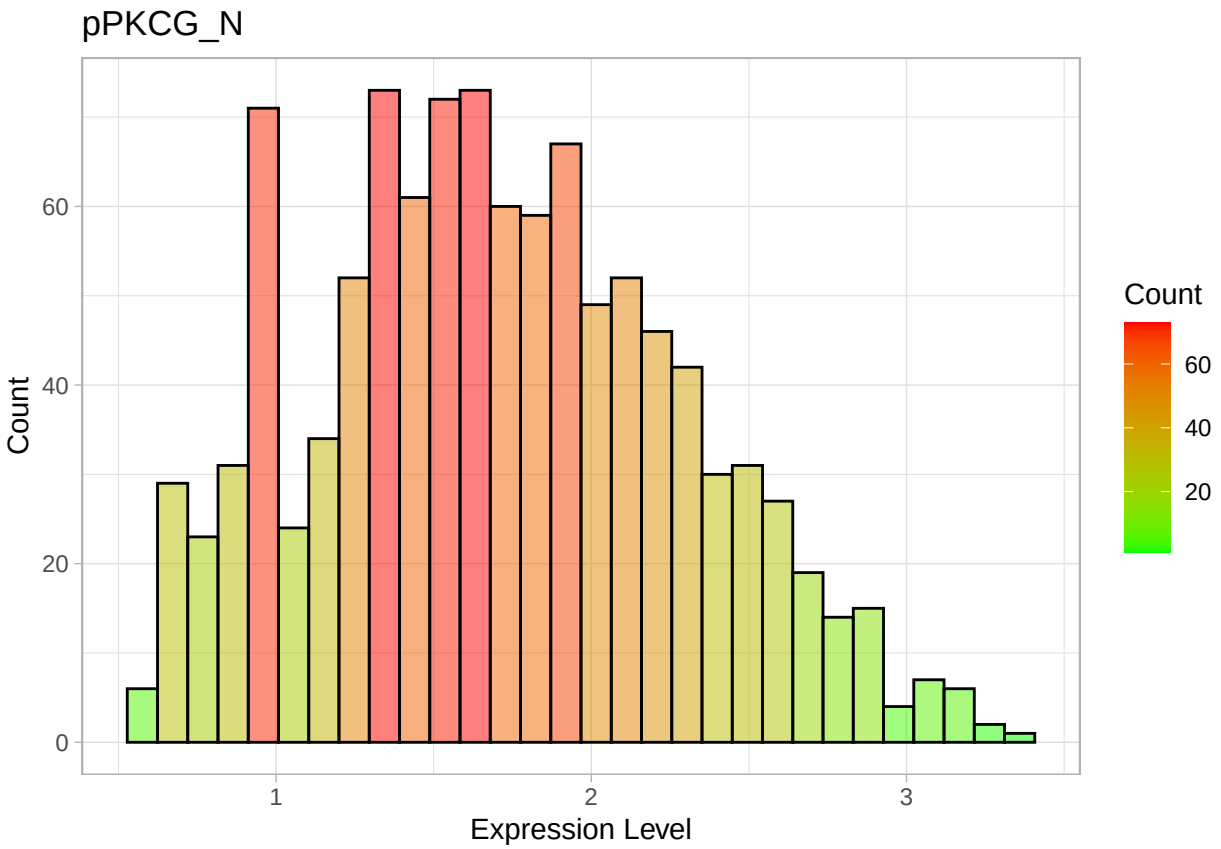
NUMB_N


P70S6_N
Count
100
50
0
0.5
1.0
1.5
Expression Level
Count
125
100
75
50
25
0

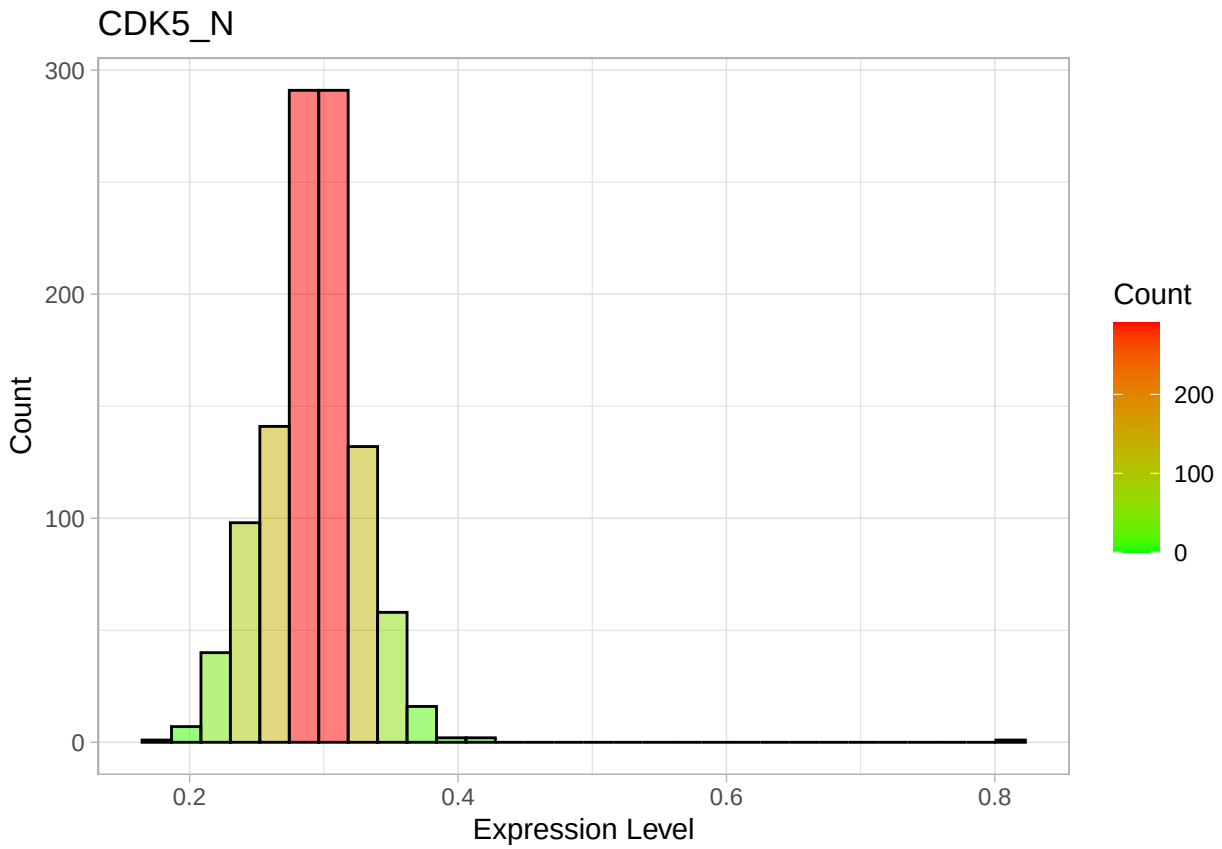
P70S6_N


pGSK3B_N
150
100
50
0
Count
0.10
0.15
0.20
0.25
Expression Level
Count
100
50
0

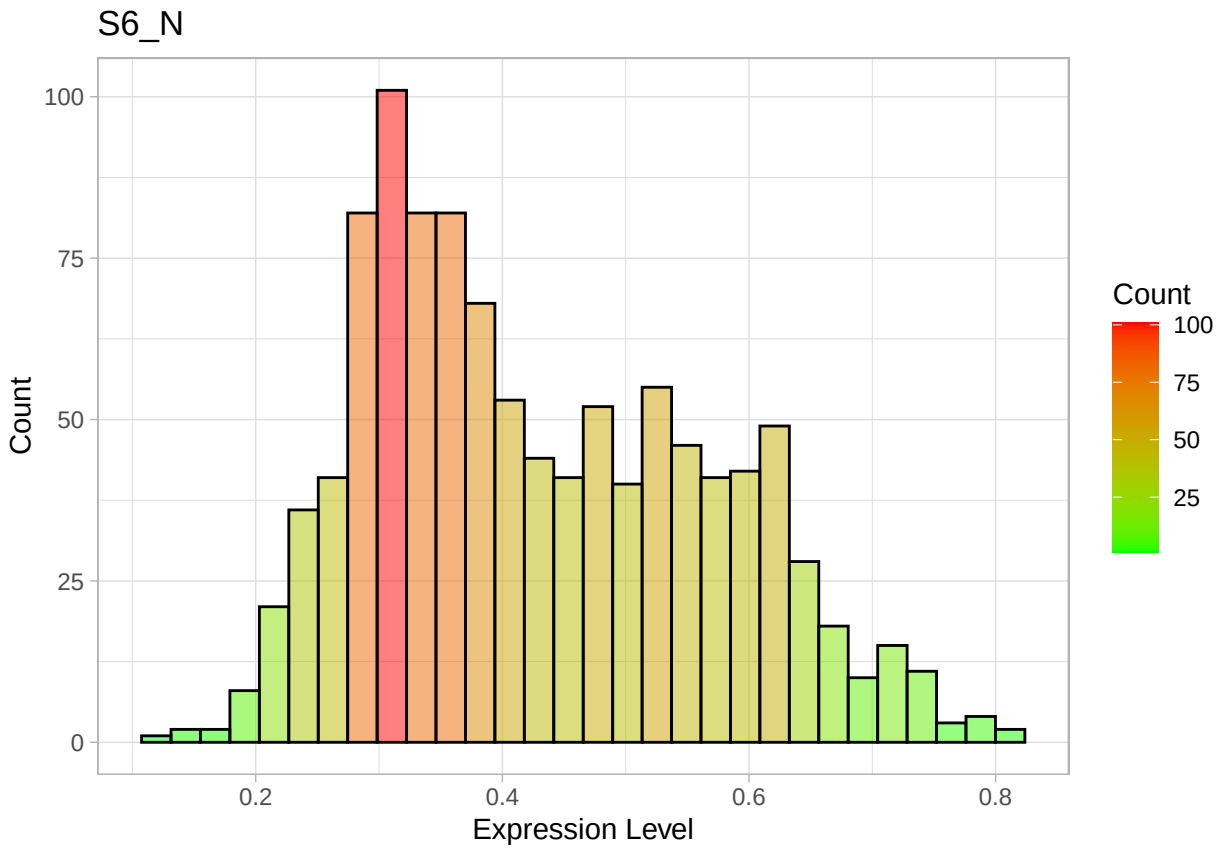
pGSK3B_N



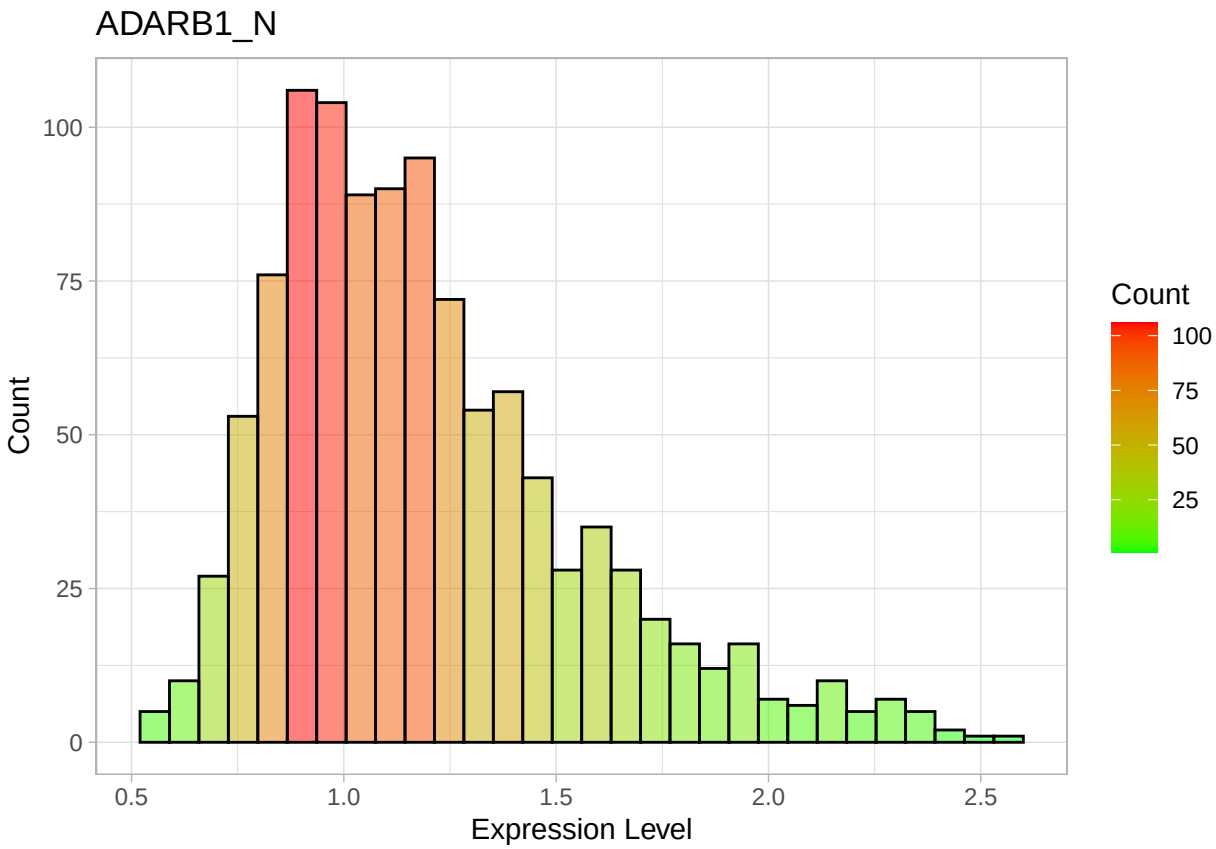
pPKCG_N


CDK5_N
300
200
100
0
Count
0.2
0.4
0.6
0.8
Expression Level
Count
200
100
0

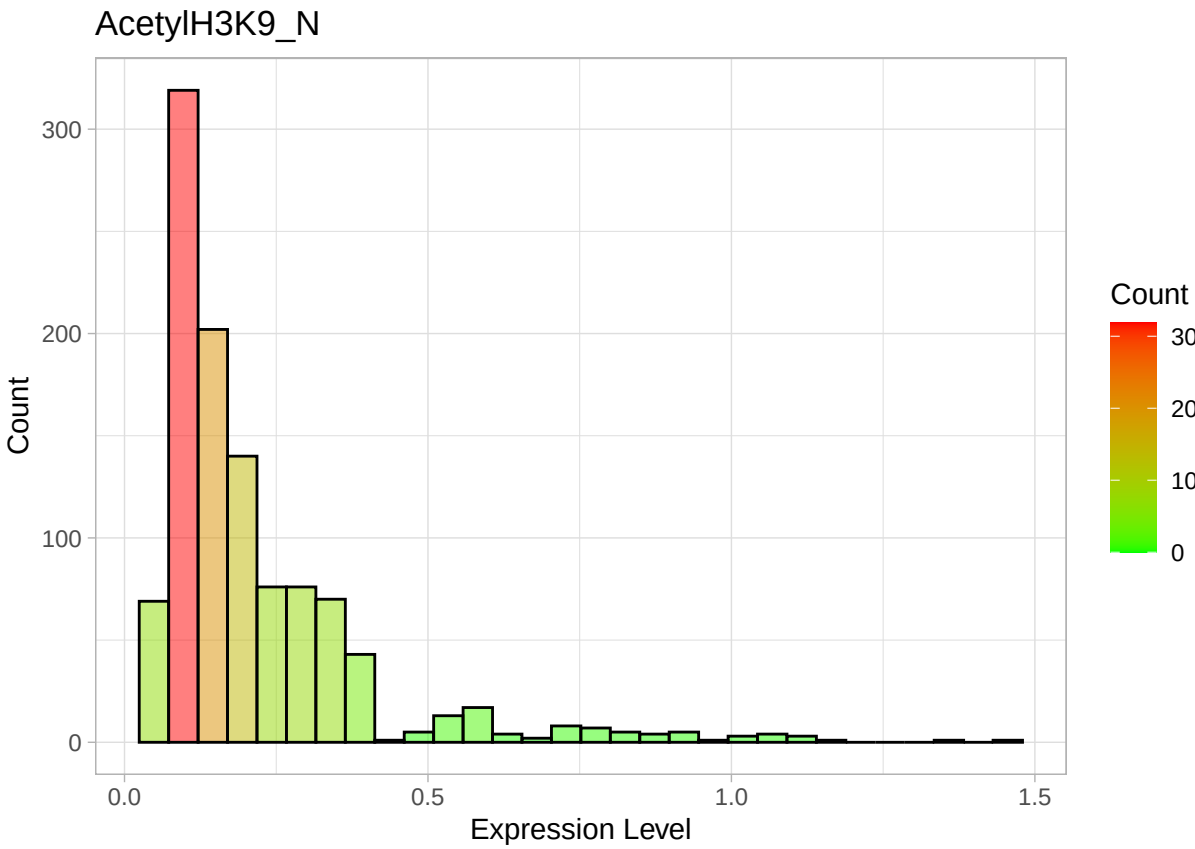
CDK5_N


S6_N
100
75
50
25
0
Count
0.2
0.4
0.6
0.8
Expression Level
Count
100
75
50
25

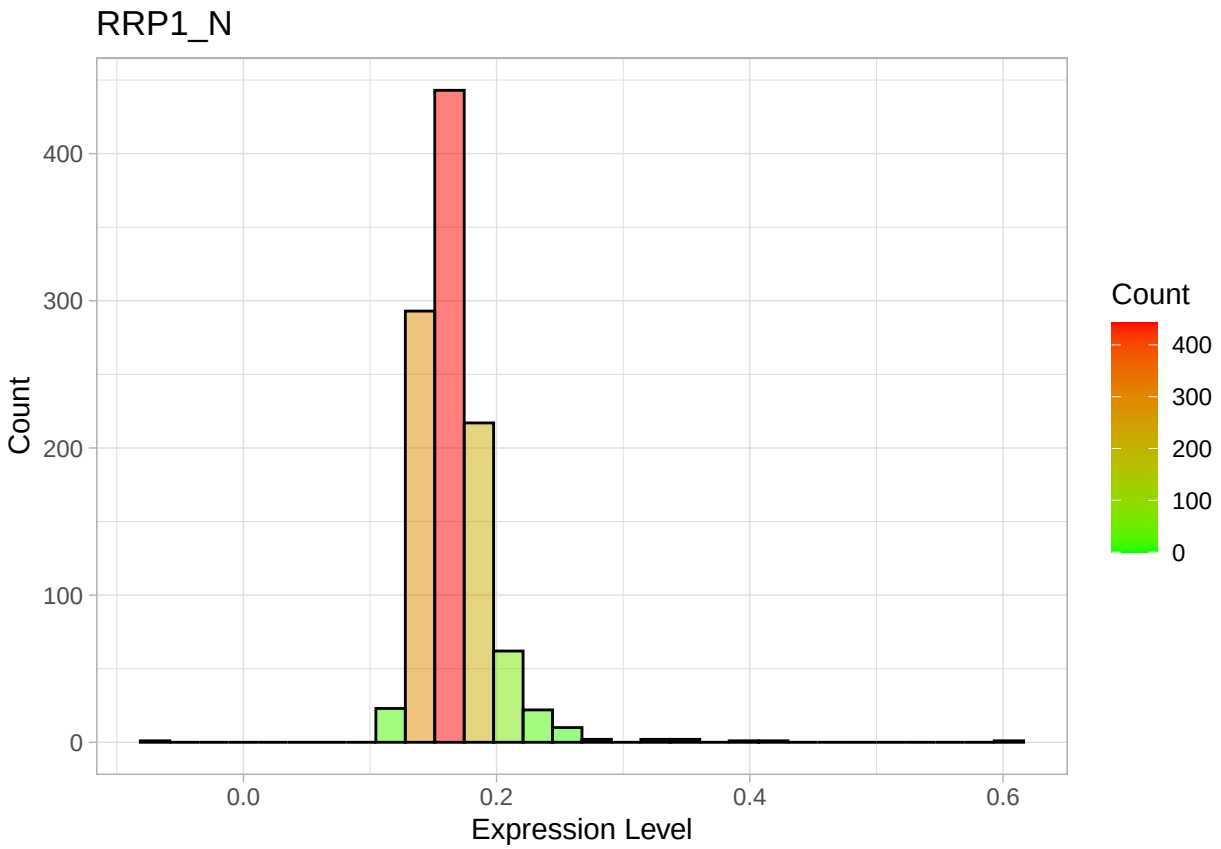
S6_N


ADARB1_N
Count
100
75
50
25
0
0.5
1.0
1.5
2.0
2.5
Expression Level
Count
100
75
50
25

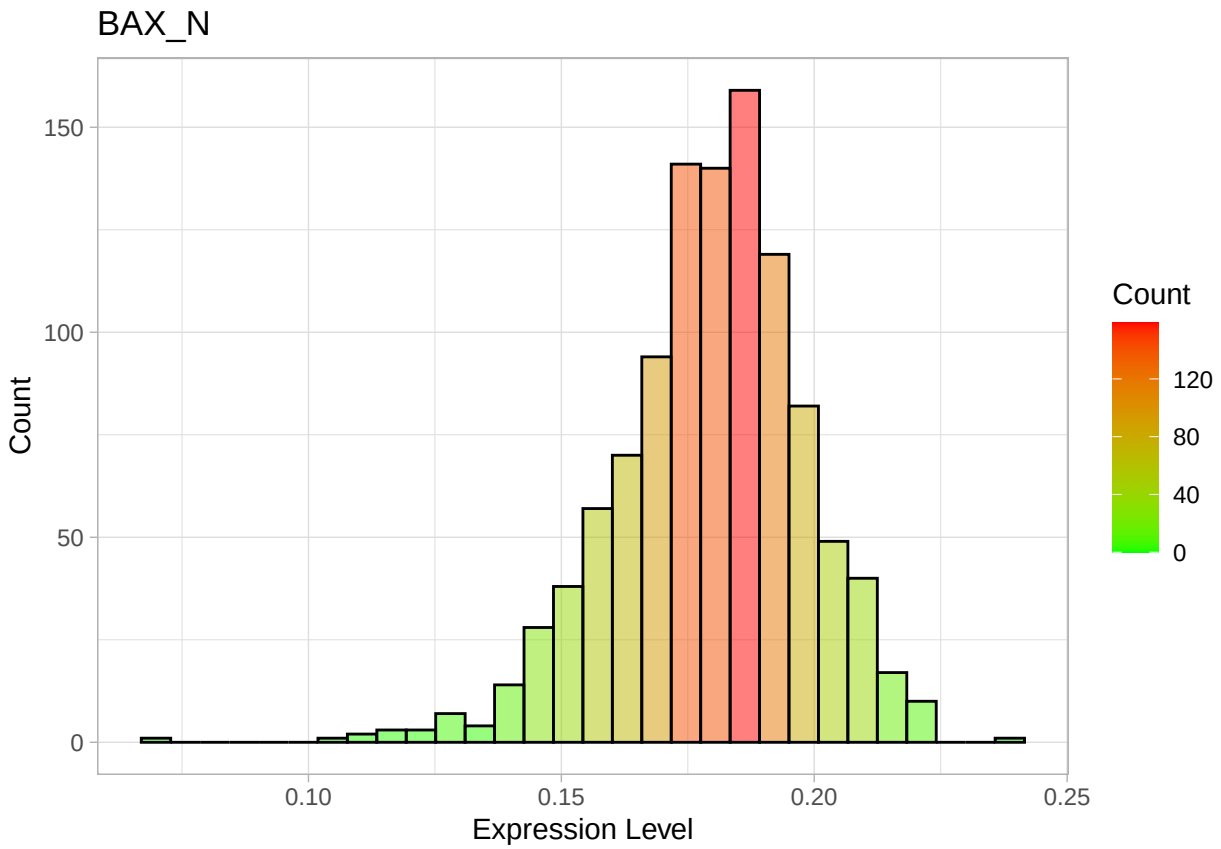
ADARB1_N



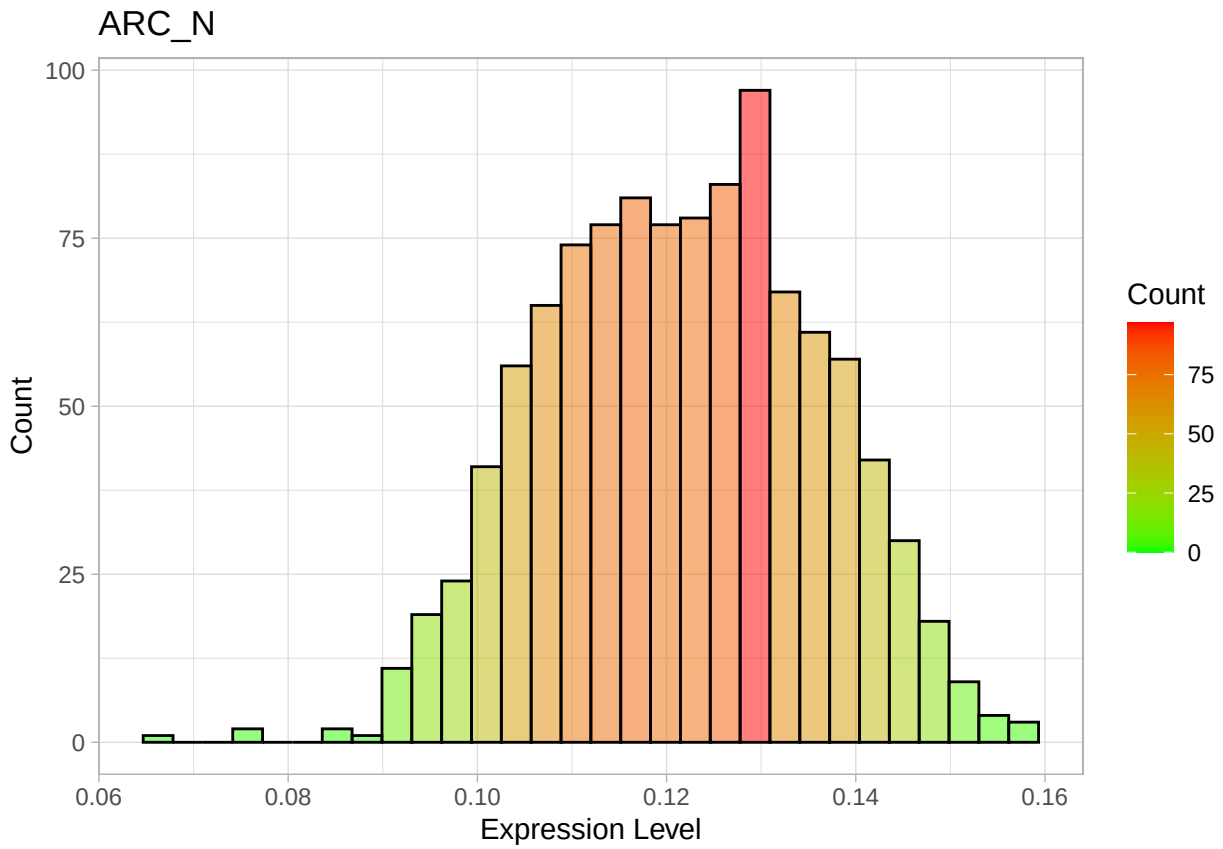
AcetylH3K9_N


RRP1_N
Count
400
300
200
100
0
0.0
0.2
0.4
0.6
Expression Level
Count
400
300
200
100
0

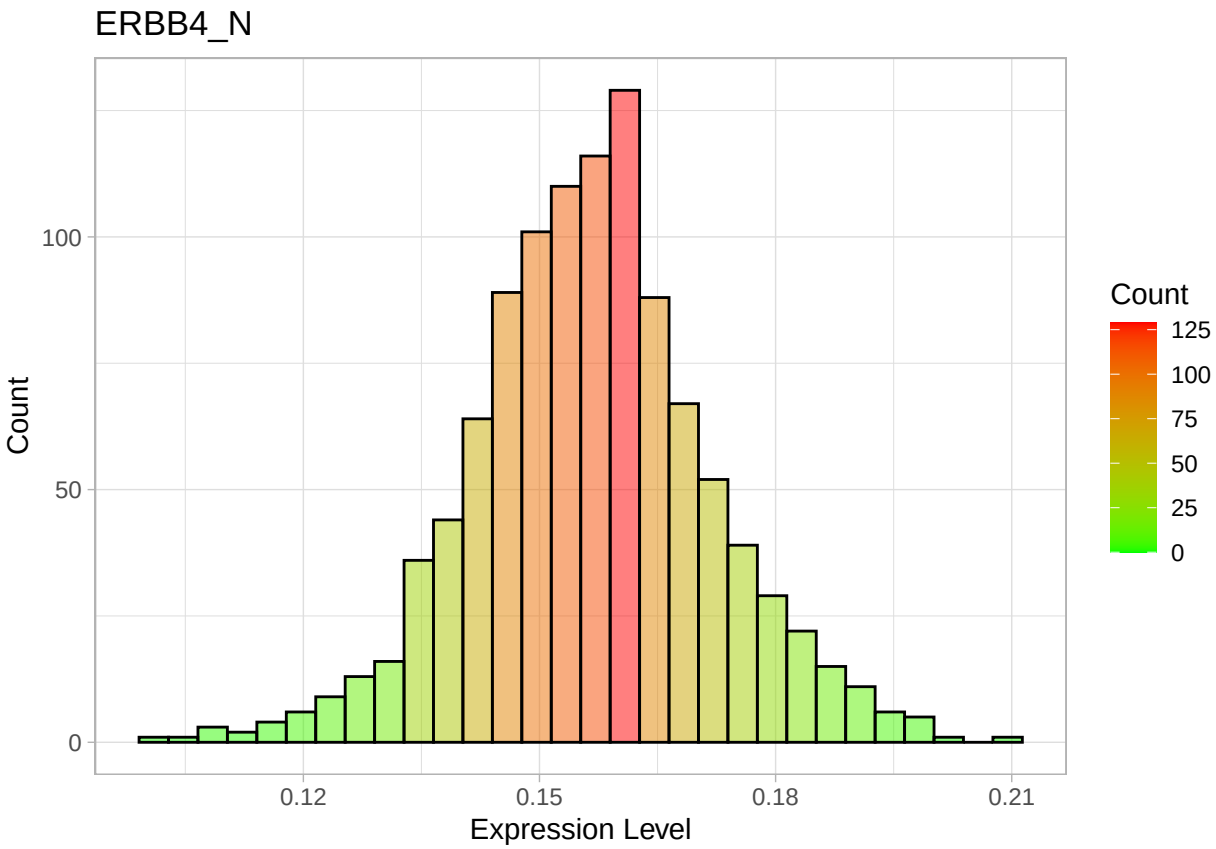
RRP1_N


BAX_N
150
100
50
0
Count
0.10
0.15
0.20
0.25
Expression Level
Count
120
80
40
0

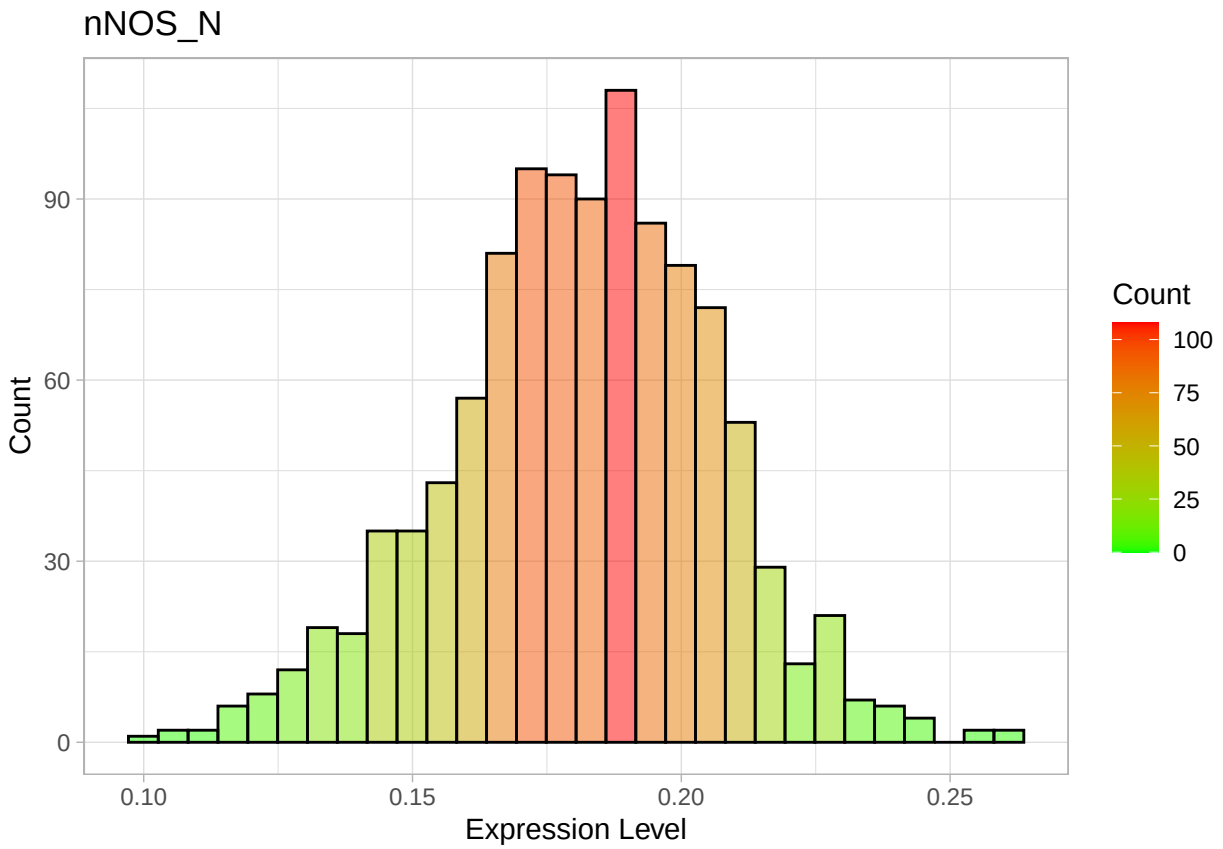
BAX_N


ARC_N
Count
100
75
50
25
0
0.06
0.08
0.10
0.12
0.14
0.16
Expression Level
Count
75
50
25
0

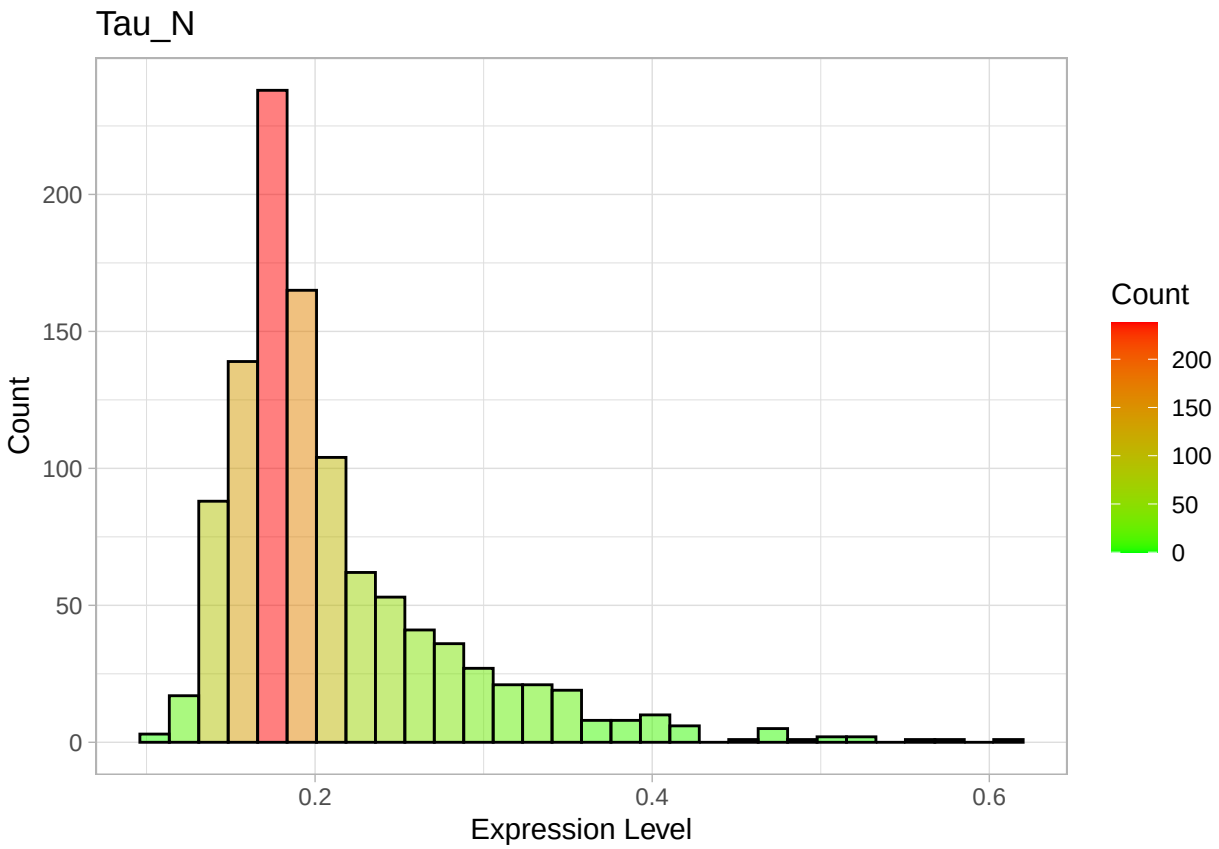
ARC_N


ERBB4_N
Count
100
50
0
0.12
0.15
0.18
0.21
Expression Level
Count
125
100
75
50
25
0

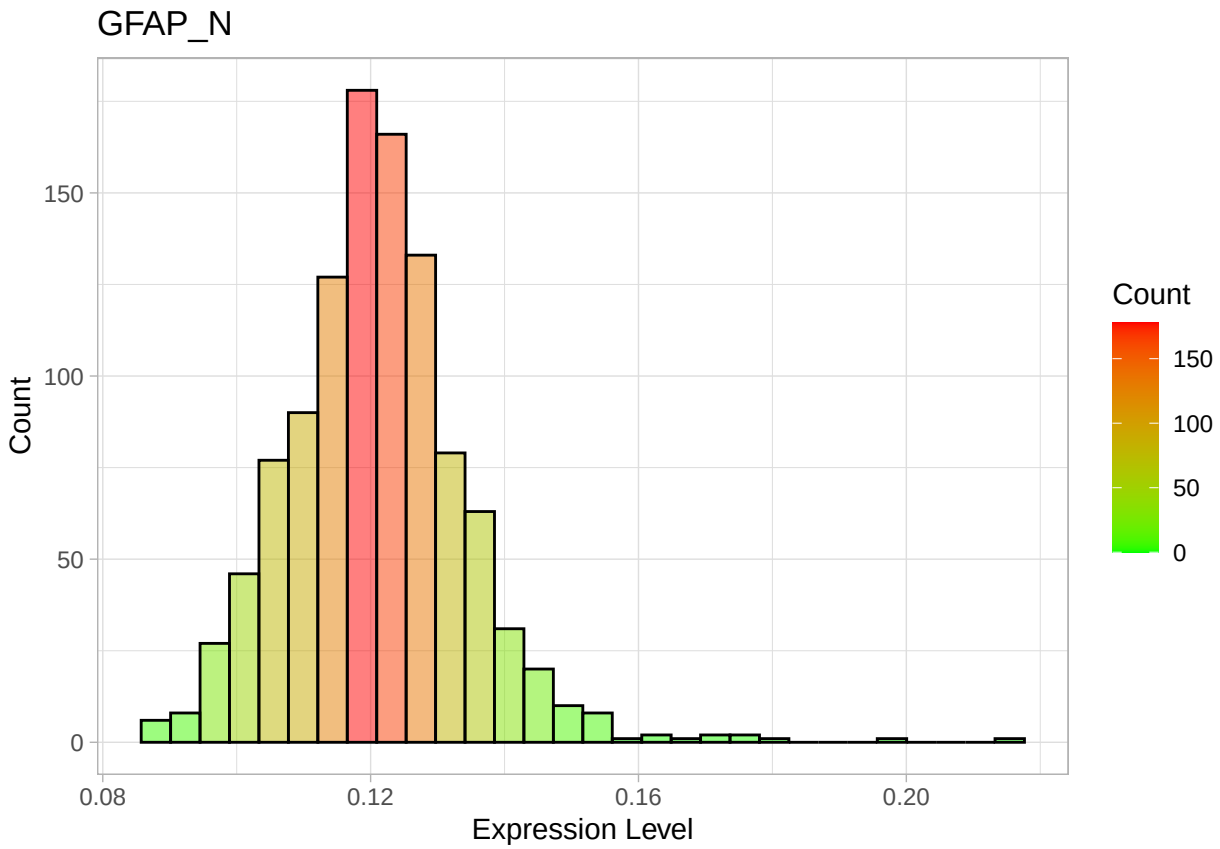
ERBB4_N


nNOS_N
Count
90
60
30
0
0.10
0.15
0.20
0.25
Expression Level
Count
100
75
50
25
0

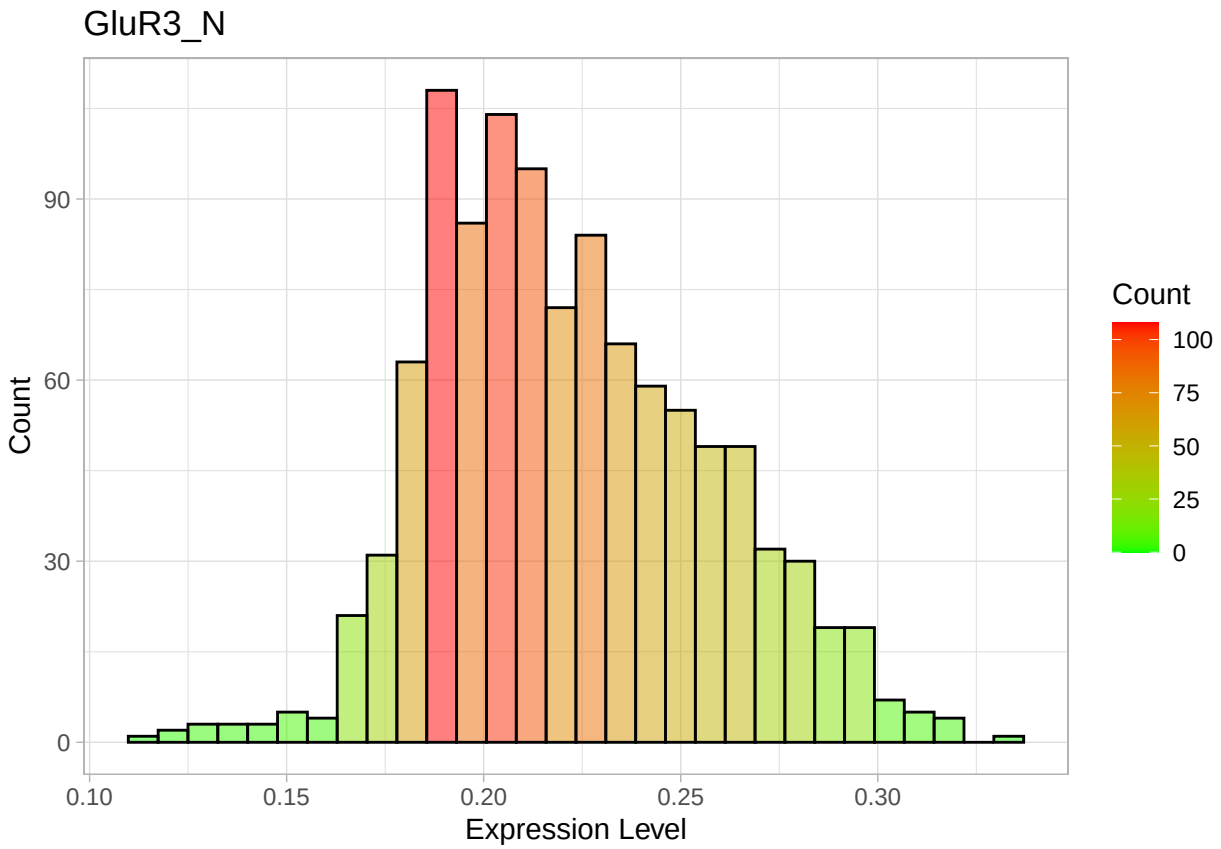
nNOS_N


Tau_N
Count
200
150
100
50
0
0.2
0.4
0.6
Expression Level
Count
200
150
100
50
0

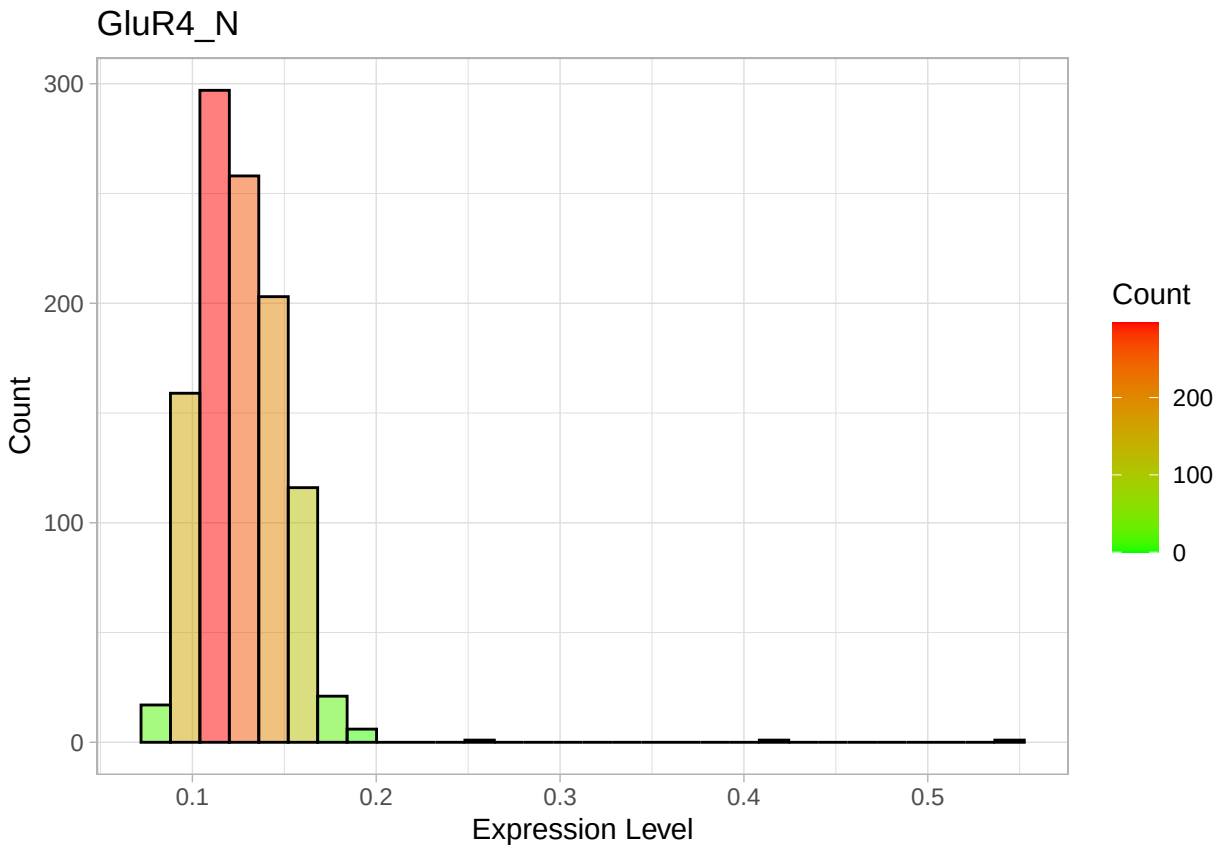
Tau_N


GFAP_N
Count
150
100
50
0
0.08
0.12
0.16
0.20
Expression Level
Count
150
100
50
0

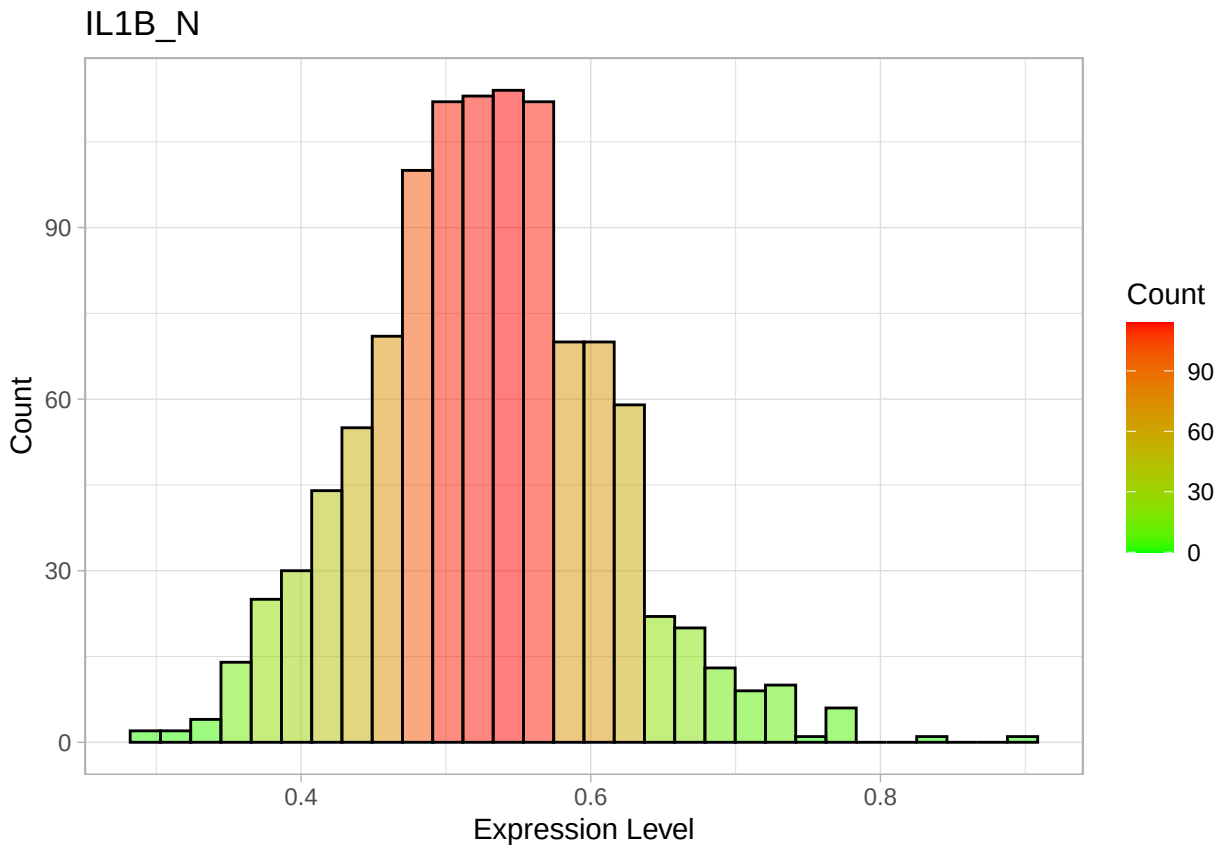
GFAP_N


GluR3_N
Count
90
60
30
0
0.10
0.15
0.20
0.25
0.30
Expression Level
Count
100
75
50
25
0

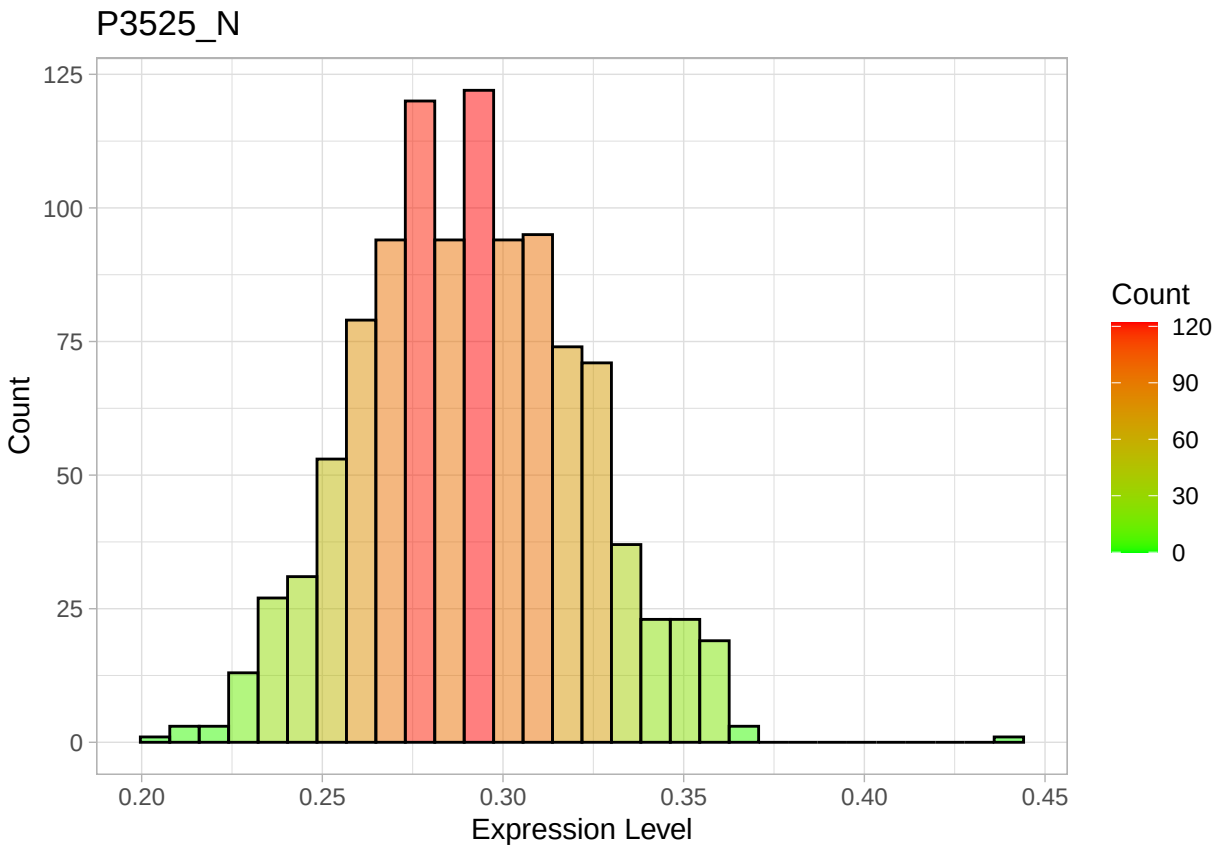
GluR3_N


GluR4_N
300
200
100
0
Count
0.1
0.2
0.3
0.4
0.5
Expression Level
Count
200
100
0

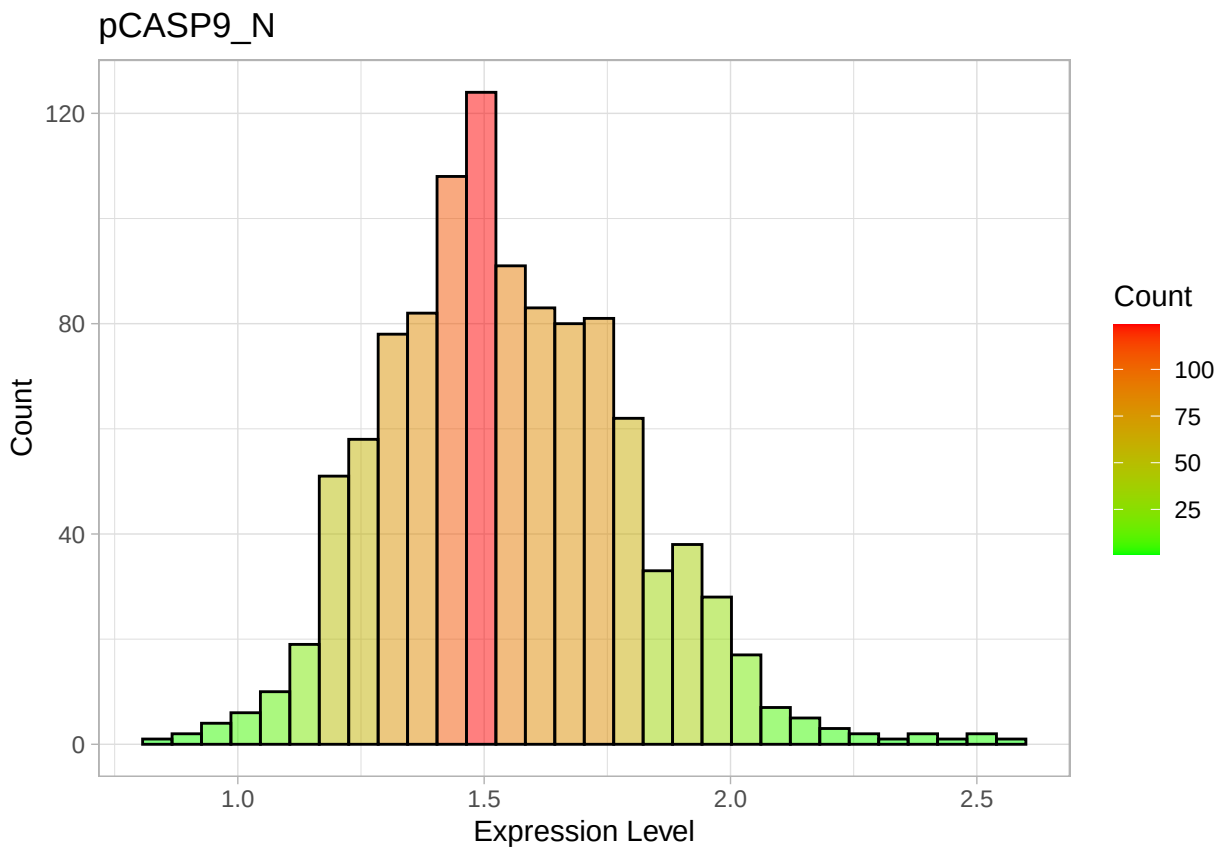
GluR4_N


IL1B_N
Count
90
60
30
0
0.4
0.6
0.8
Expression Level
Count
90
60
30
0

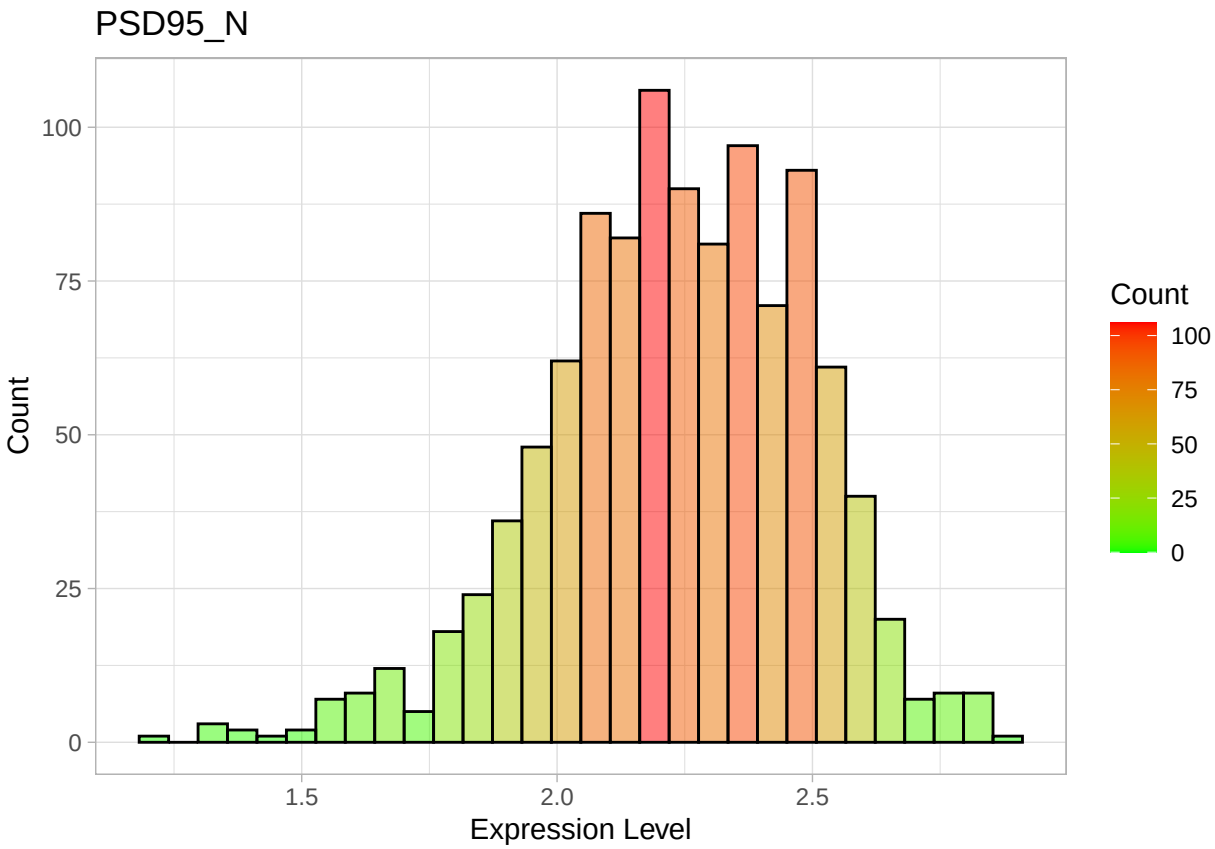
IL1B_N


P3525_N
125
100
75
50
25
0
Count
0.20
0.25
0.30
0.35
0.40
0.45
Expression Level
Count
120
90
60
30
0

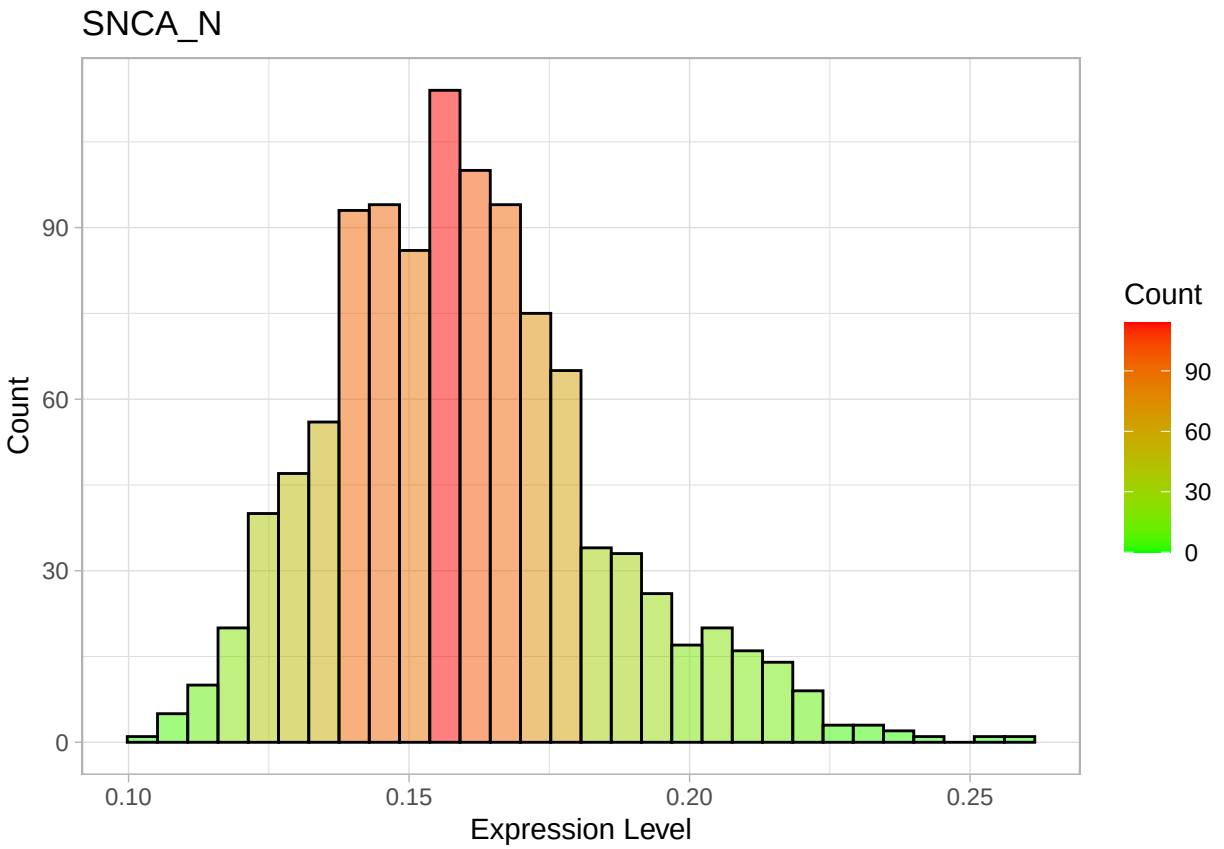
P3525_N


pCASP9_N
120
80
40
0
Count
1.0
1.5
2.0
2.5
Expression Level
Count
100
75
50
25

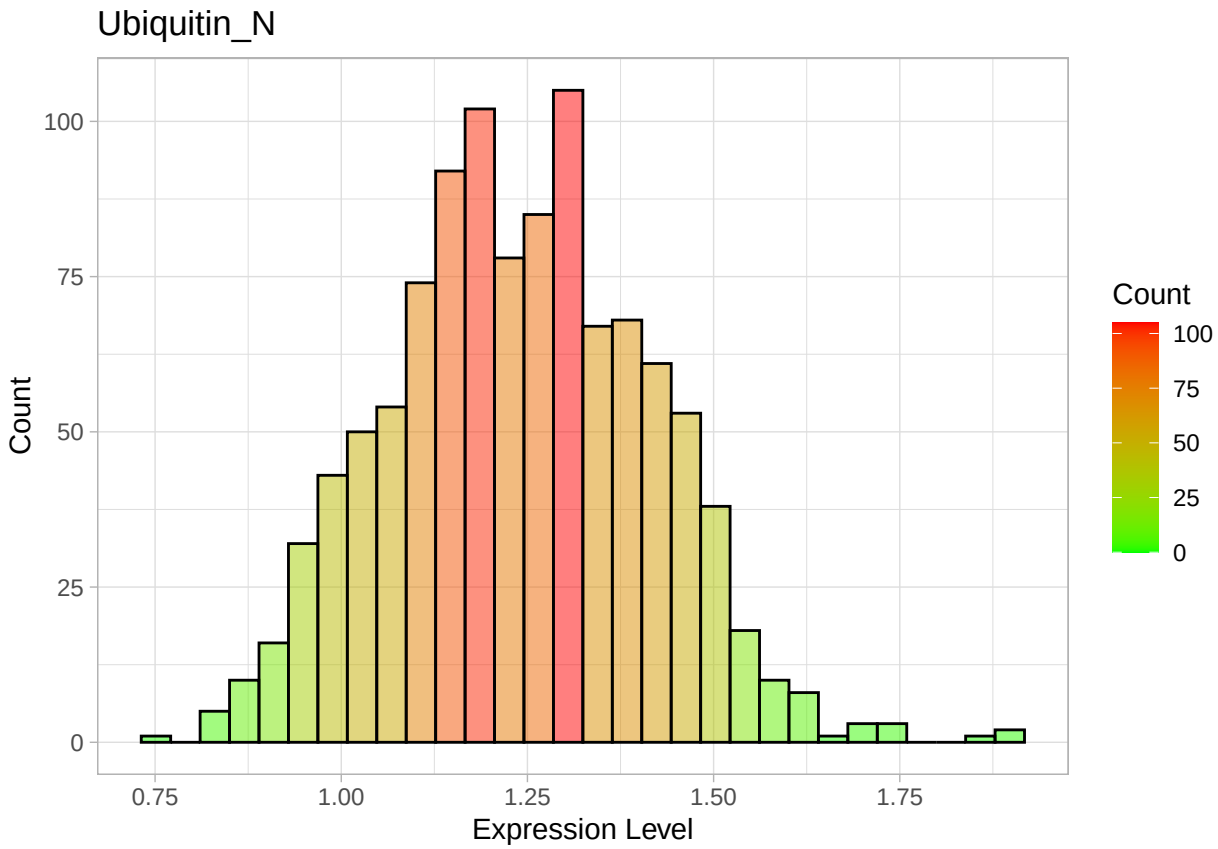
pCASP9_N


PSD95_N
Count
100
75
50
25
0
1.5
2.0
2.5
Expression Level
Count
100
75
50
25
0

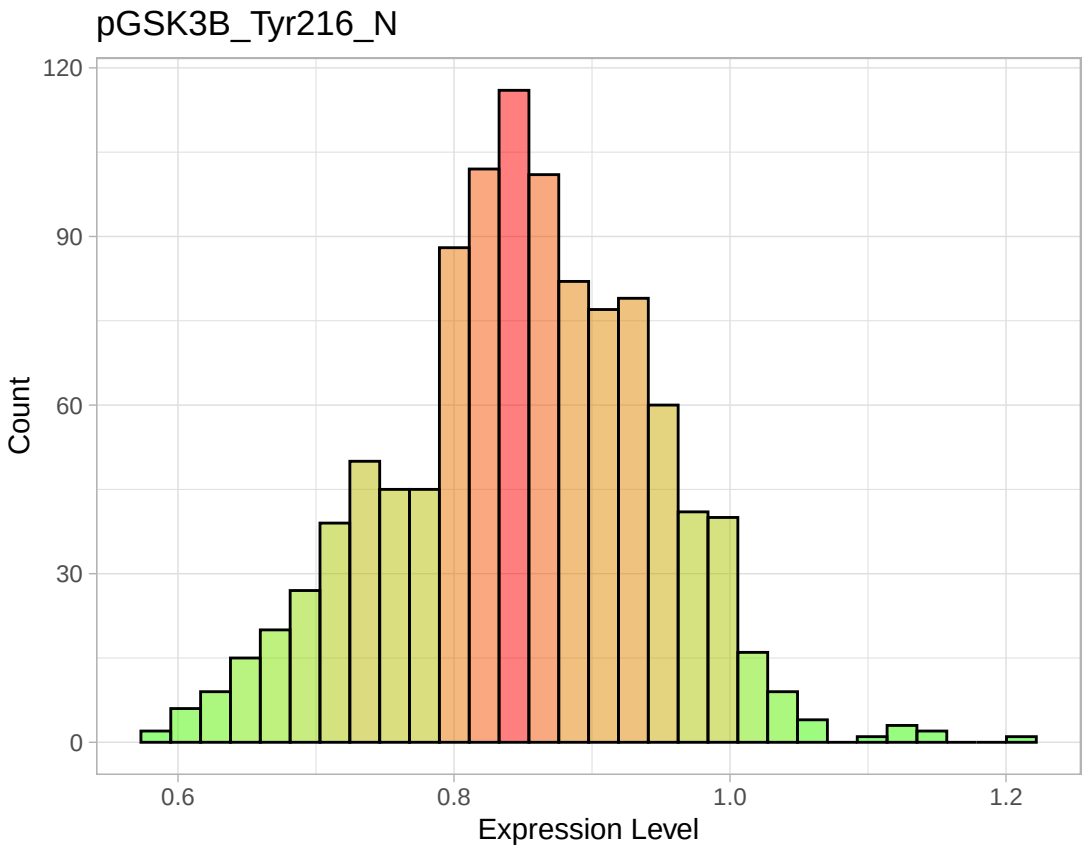
PSD95_N


SNCA_N
Count
90
60
30
0
0.10
0.15
0.20
0.25
Expression Level
Count
90
60
30
0

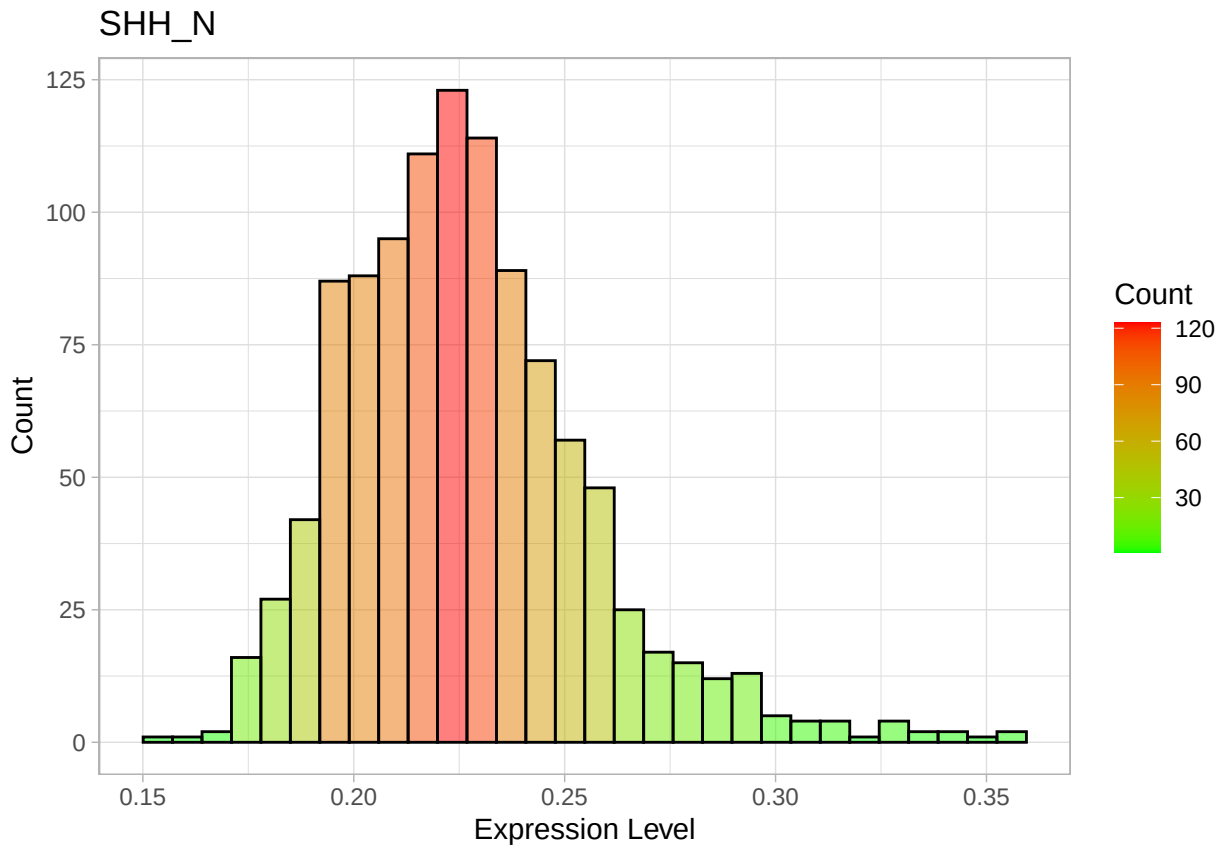
SNCA_N


Ubiquitin_N
Count
100
75
50
25
0
0.75
1.00
1.25
1.50
1.75
Expression Level
Count
100
75
50
25
0

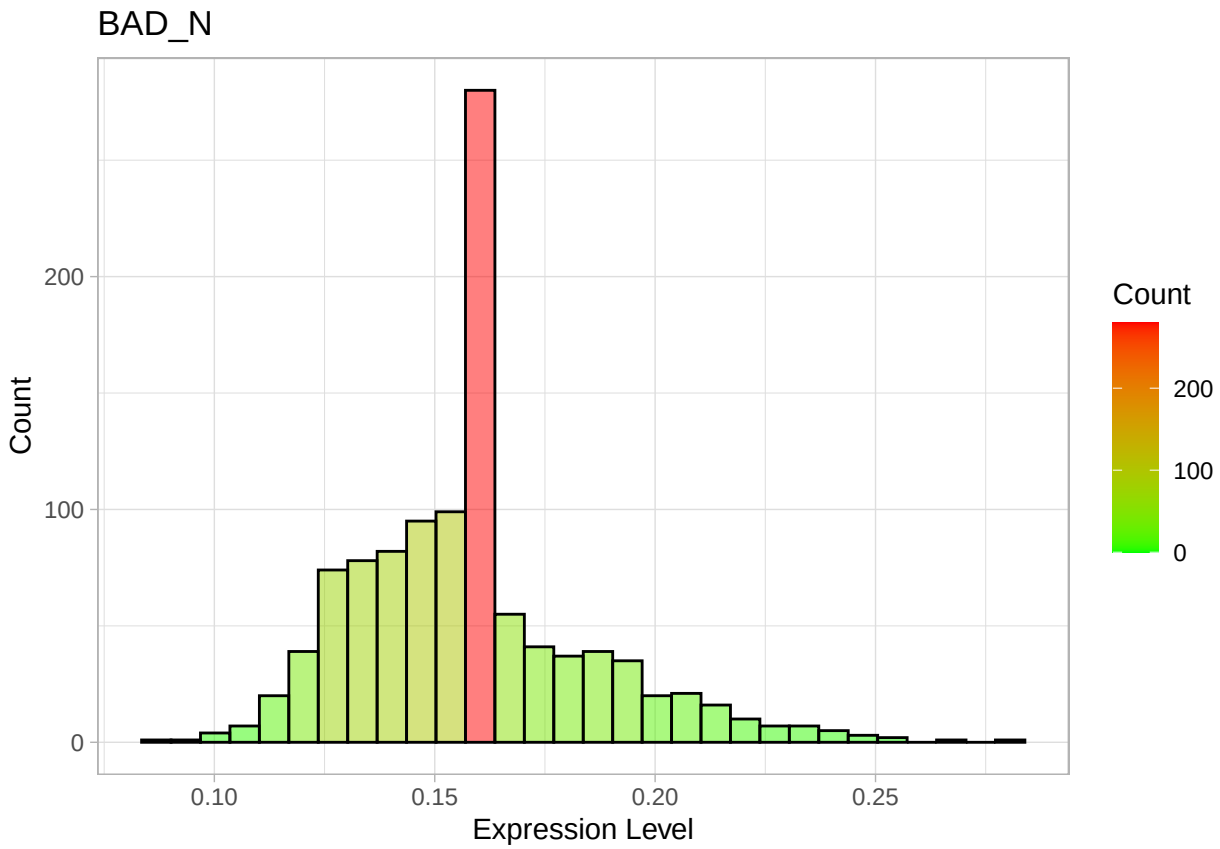
Ubiquitin_N



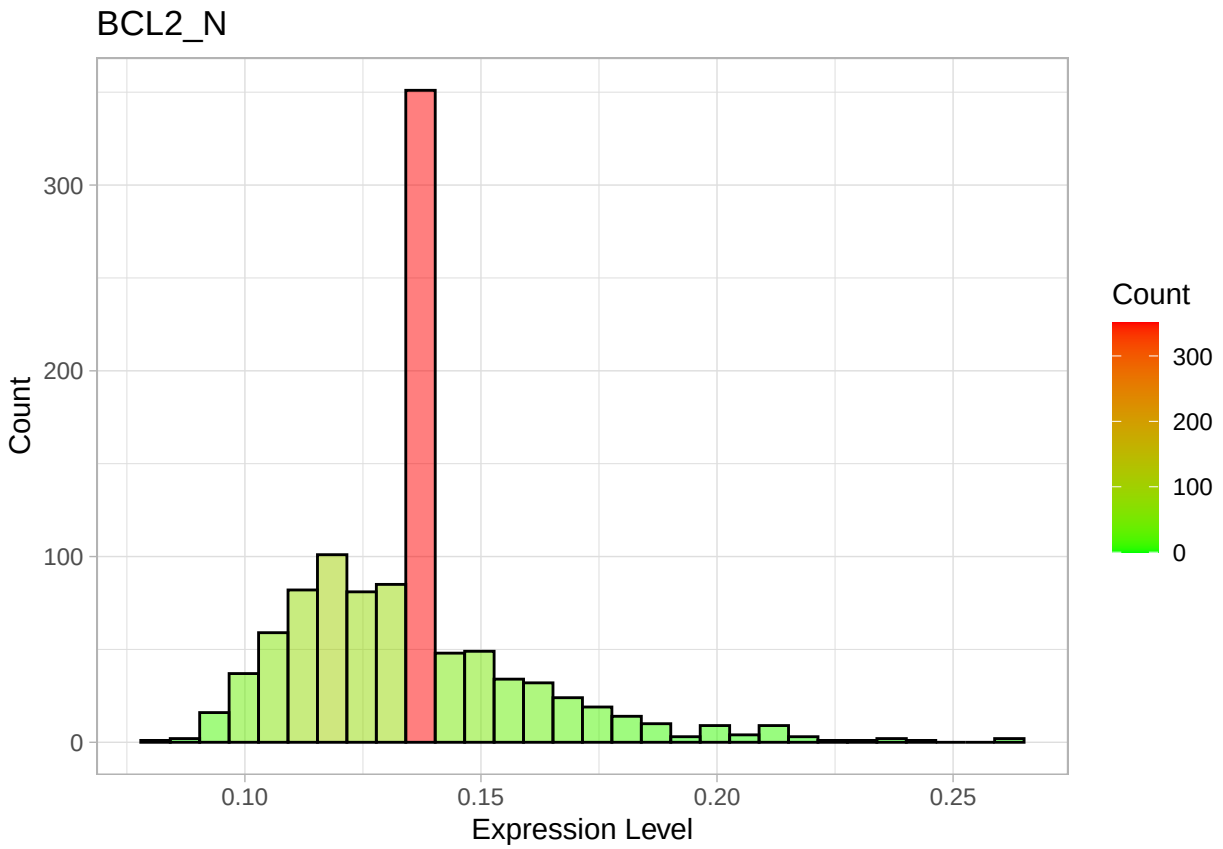
pGSK3B__Tyr216__N


SHH_N
125
100
75
50
25
0
Count
0.15
0.20
0.25
0.30
0.35
Expression Level
Count
120
90
60
30

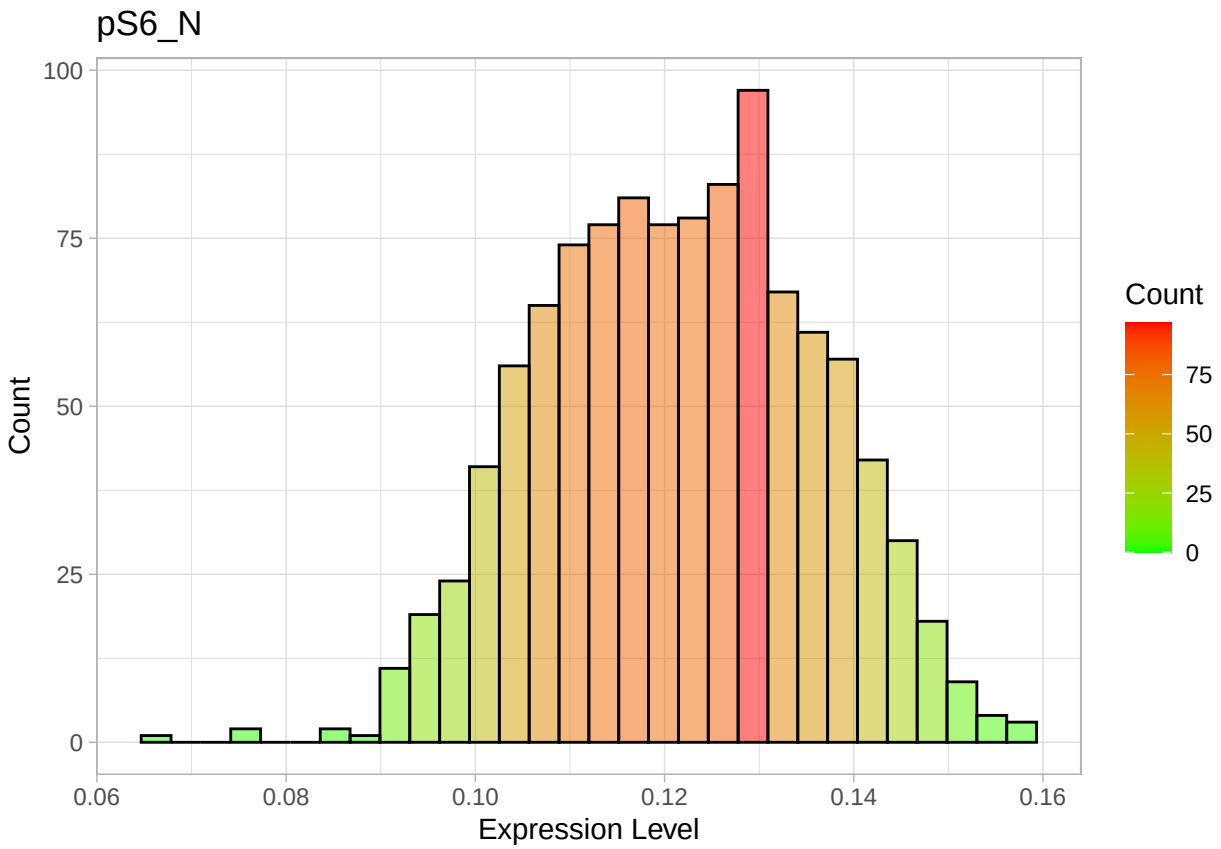
SHH_N


BAD_N
Count
200
100
0
0.10
0.15
0.20
0.25
Expression Level
Count
200
100
0

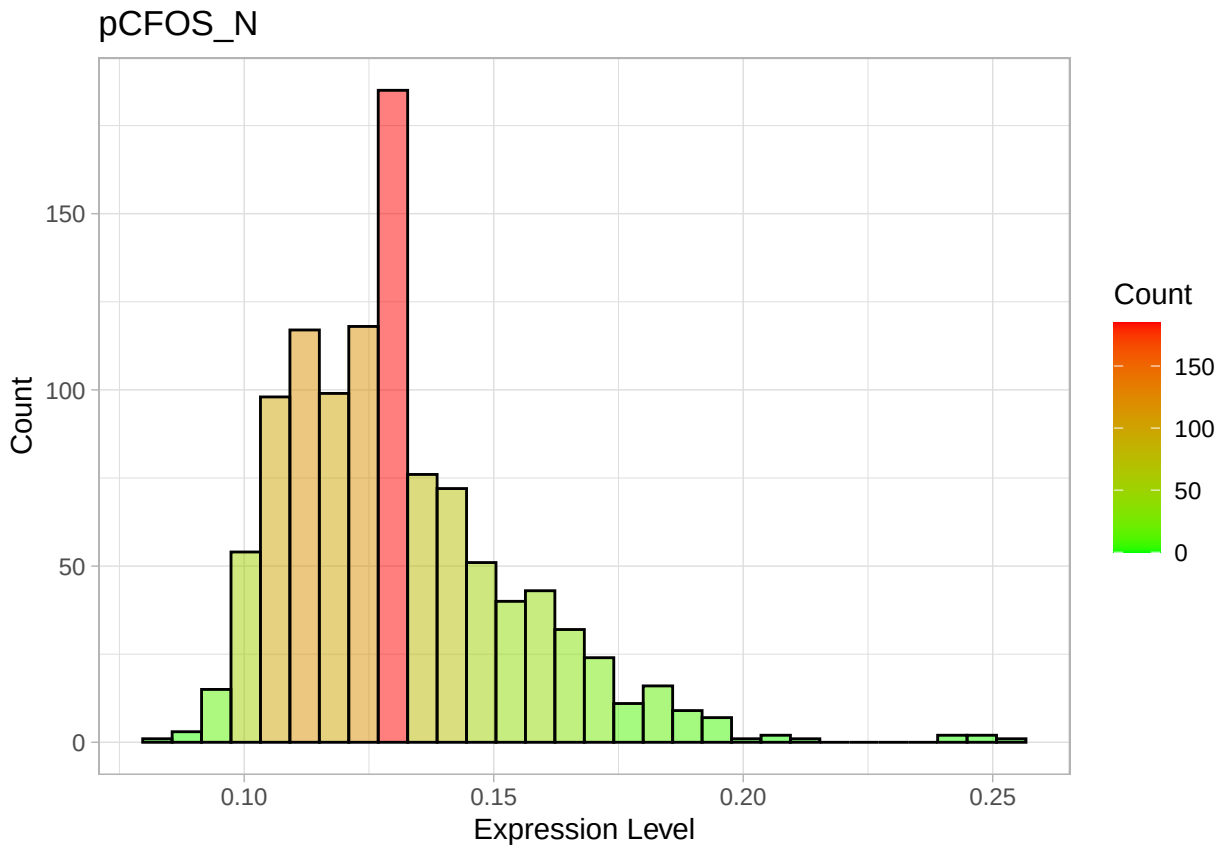
BAD_N


BCL2_N
Count
300
200
100
0
0.10
0.15
0.20
0.25
Expression Level
Count
300
200
100
0

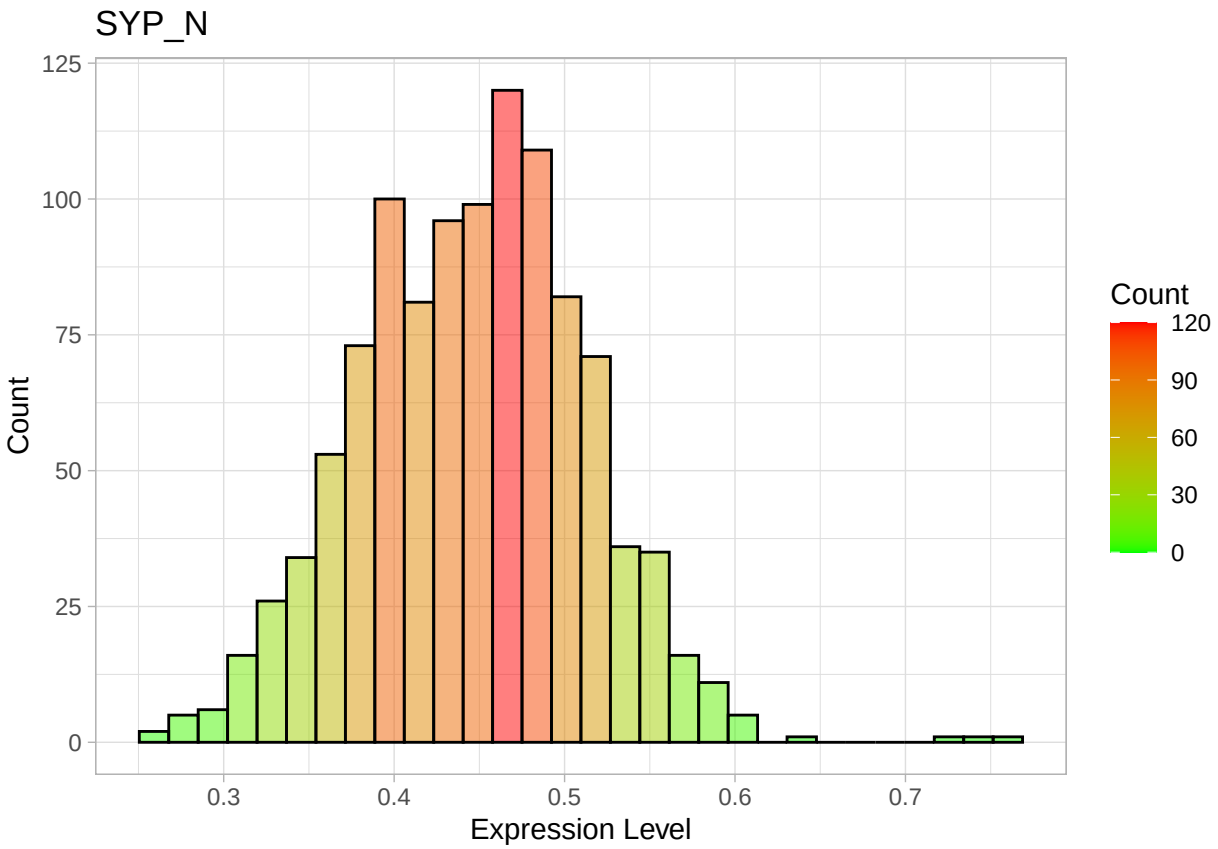
BCL2_N


pS6_N
100
75
50
25
0
Count
0.06
0.08
0.10
0.12
0.14
0.16
Expression Level
Count
75
50
25
0

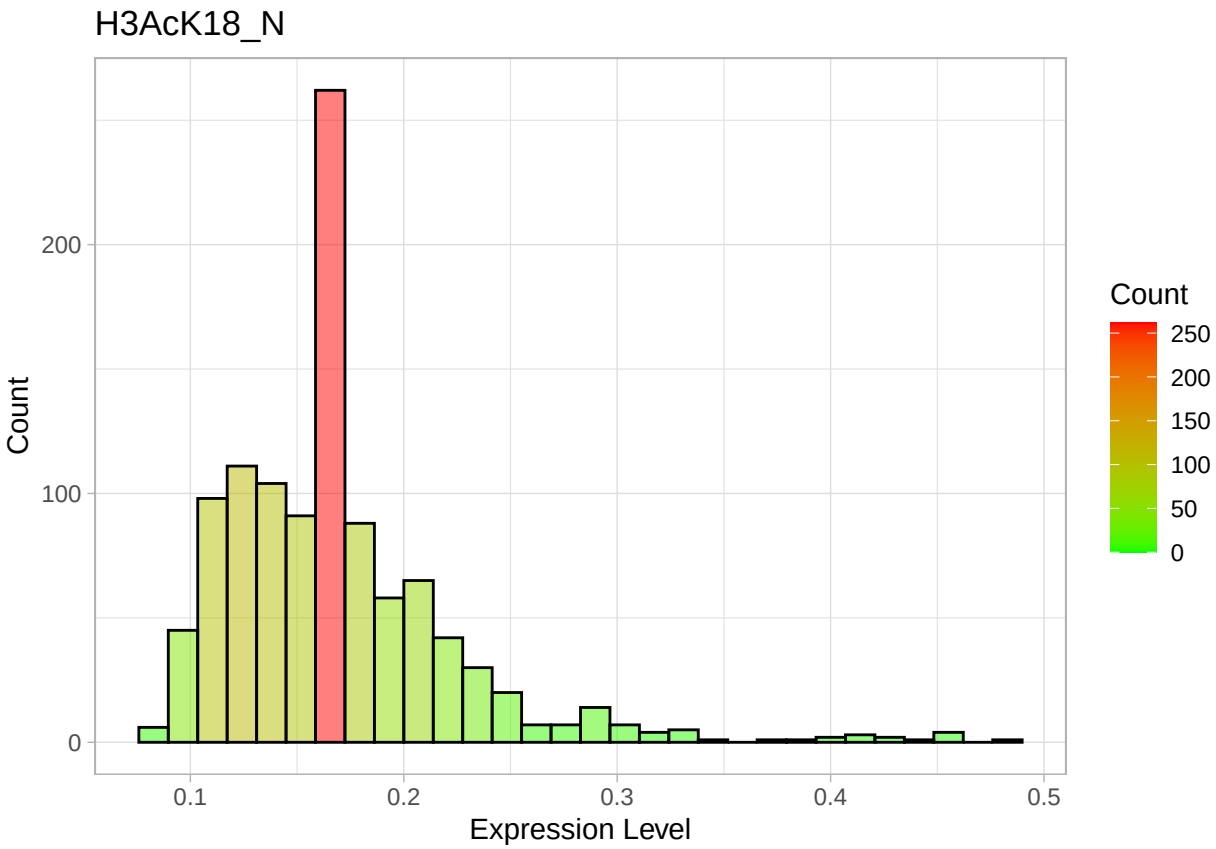
pS6__N


pCFOS_N
Count
150
100
50
0
0.10
0.15
0.20
0.25
Expression Level
Count
150
100
50
0

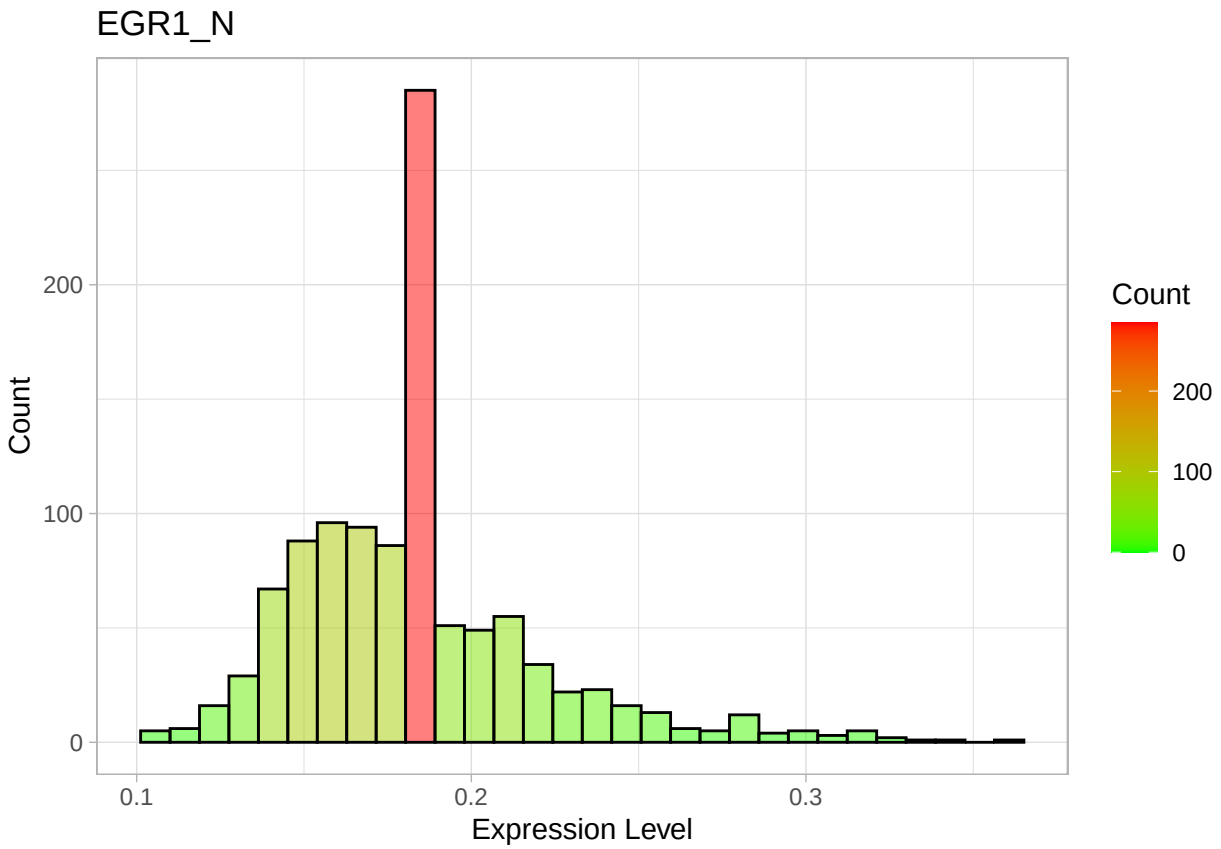
pCFOS_N


SYP_N
125
100
75
50
25
0
Count
0.3
0.4
0.5
0.6
0.7
Expression Level
Count
120
90
60
30
0

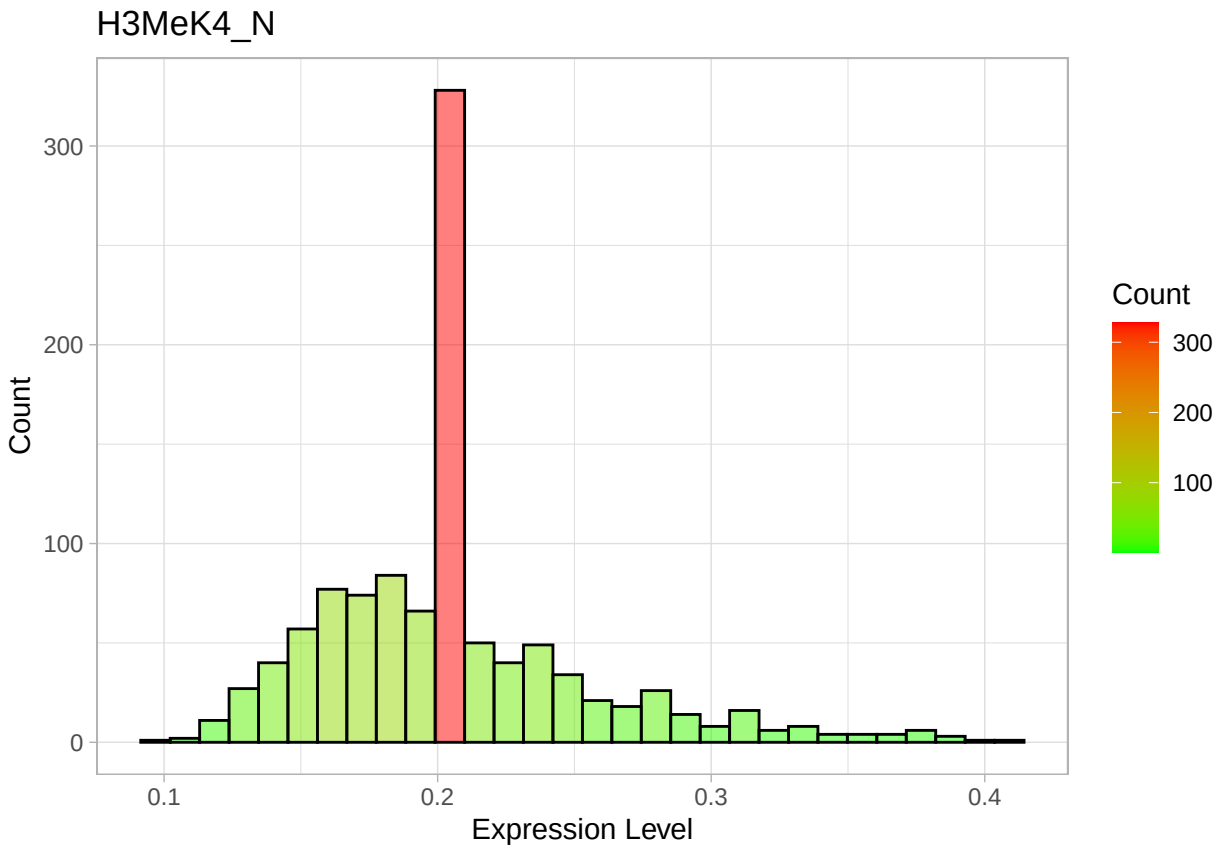
SYP_N


H3AcK18_N
Count
200
100
0
0.1
0.2
0.3
0.4
0.5
Expression Level
Count
250
200
150
100
50
0

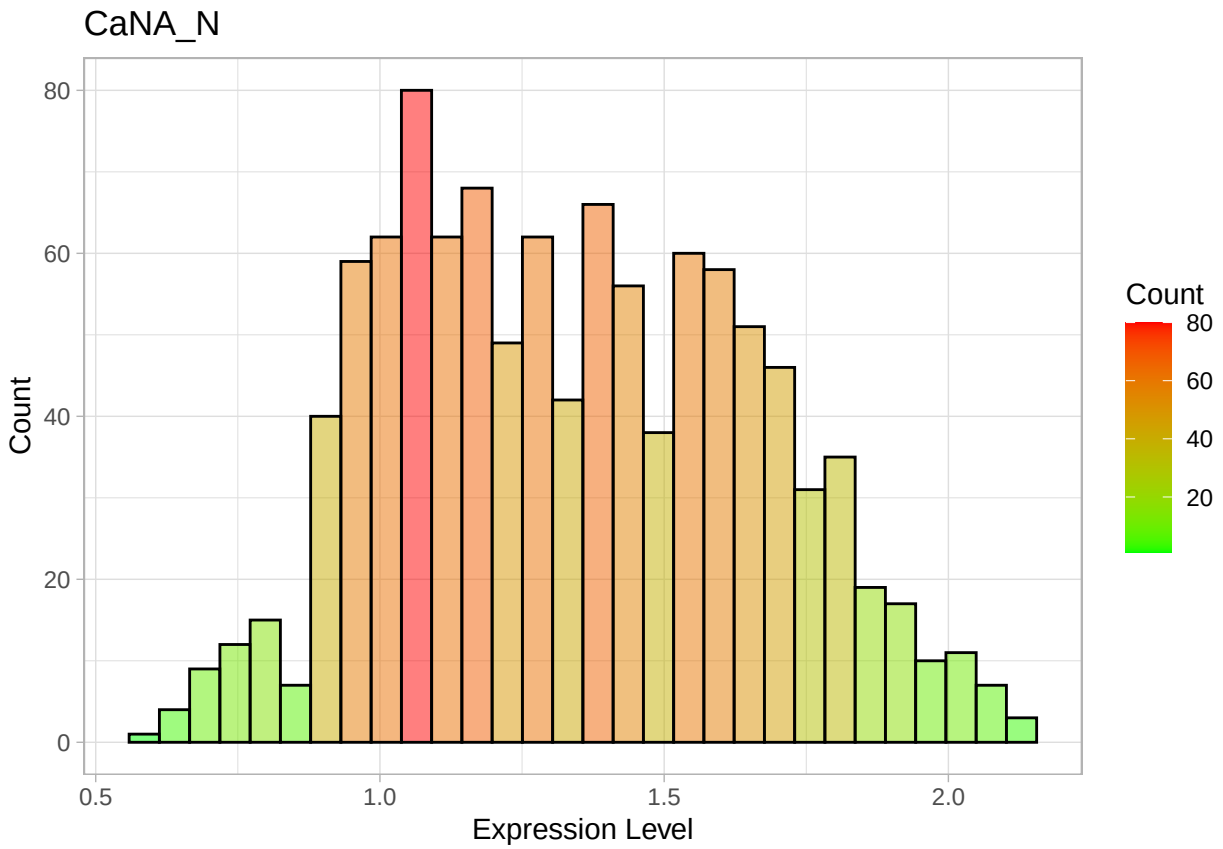
H3AcK18_N


EGR1_N
Count
200
100
0
0.1
0.2
0.3
Expression Level
Count
200
100
0

EGR1_N

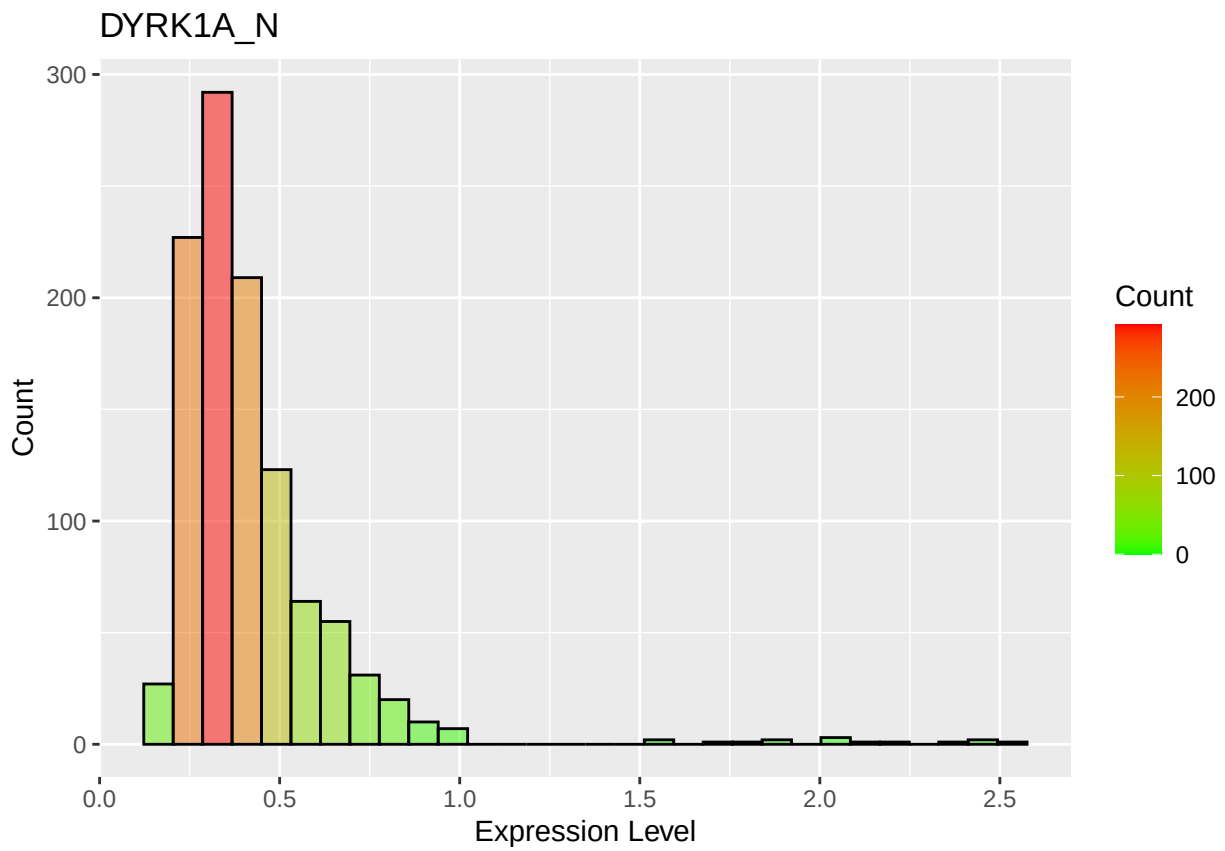

H3MeK4_N
Count
300
200
100
0
0.1
0.2
0.3
0.4
Expression Level
Count
300
200
100

H3MeK4_N



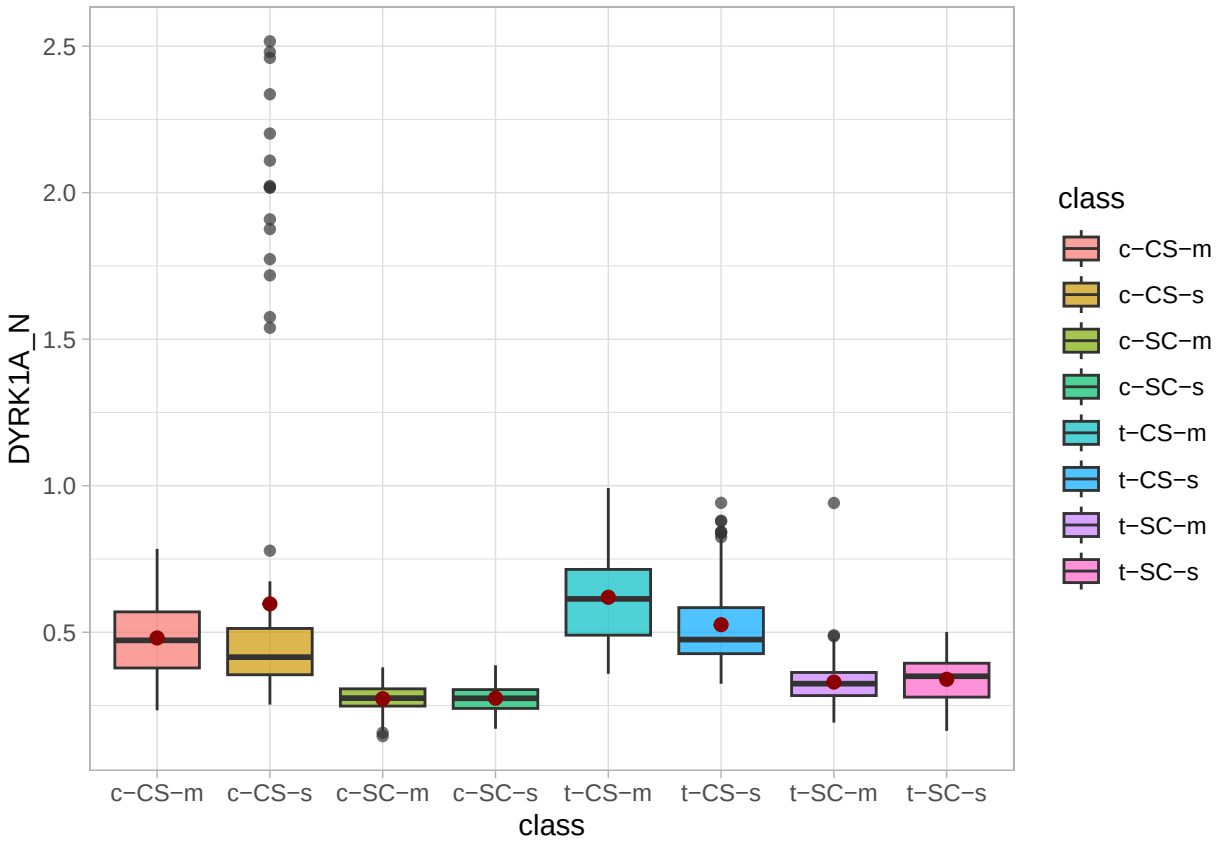
CaNA_N

```
ggplot(ncortex, aes(x = eval(parse(text = proteins[1])))) +
    geom_histogram(aes(fill=..count..), color = "black", alpha = 0.5) +
    scale_fill_gradient("Count", low="green", high="red") +
    labs(title = proteins[1],
         x = "Expression Level",
         y = "Count")
```

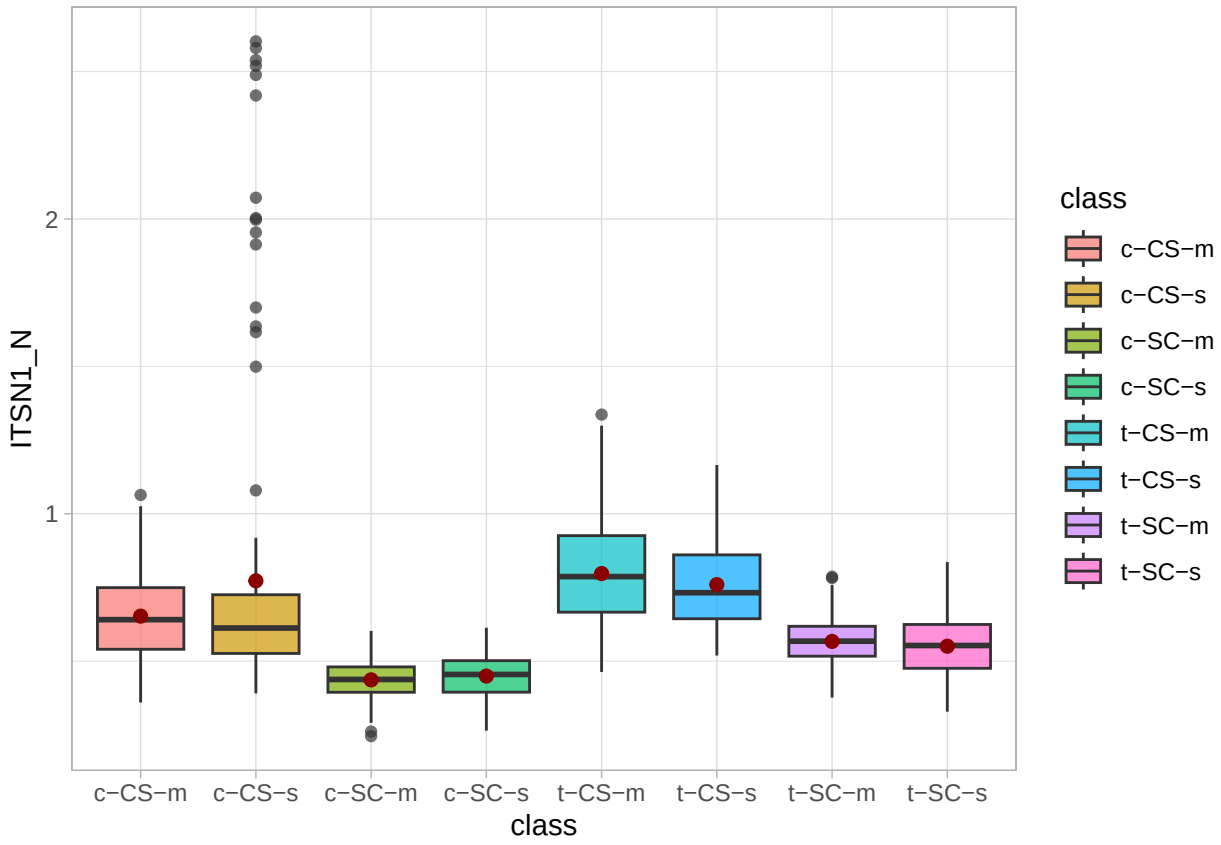


Distribution of Protein Expression Level by Class

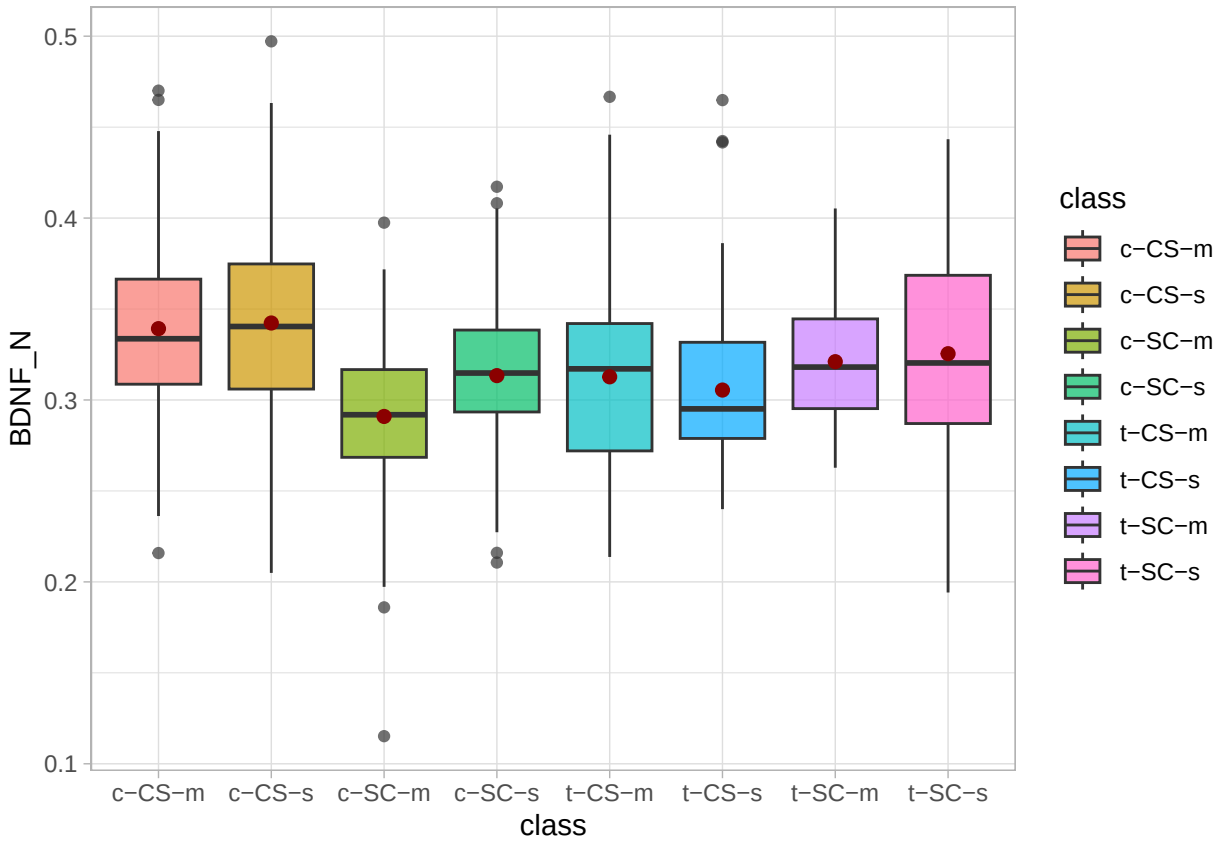
```
for(i in 1:length(proteins)){
  plot <- ggplot(ncortex, aes(x = class, y = eval(parse(text = proteins[i])))) +
    geom_boxplot(aes(fill = class), alpha = 0.7) +
    stat_summary(fun.y=mean, colour="darkred", geom="point", size=2) +
    labs(y = proteins[i]) +
    theme_light()
  cat("#### ", proteins[i], "\n")
  print(plot)
  cat('\n\n')
}
```


DYRK1A_N
2.5
2.0
1.5
1.0
0.5
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

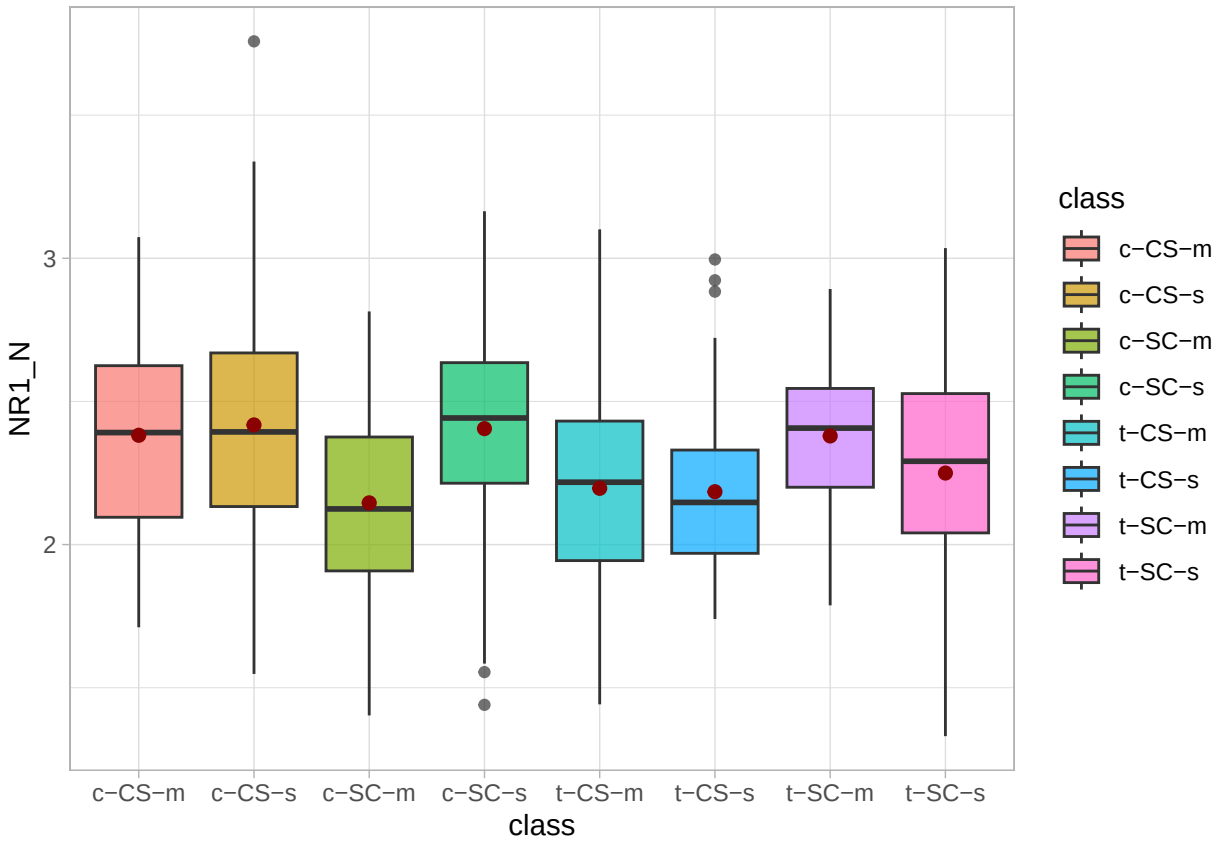
DYRK1A_N


ITSN1_N
2
1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

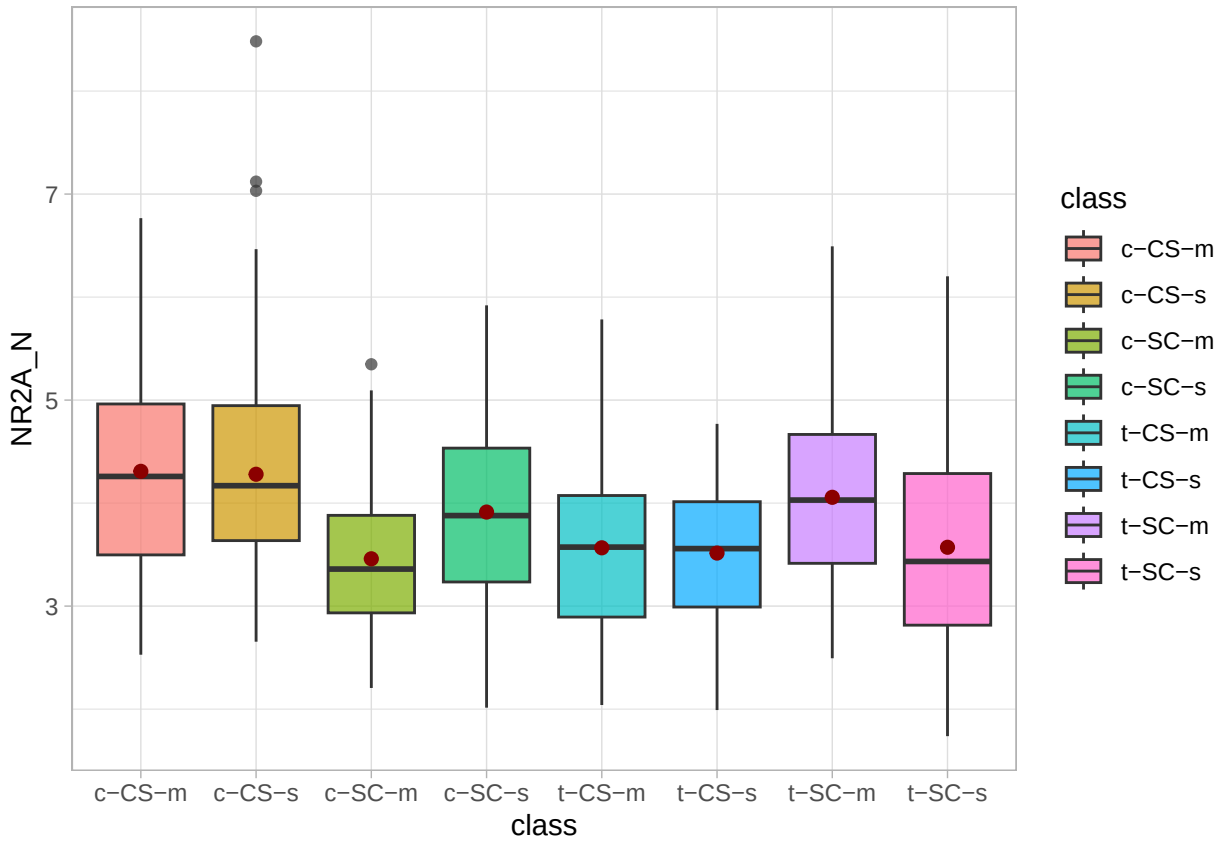
ITSN1_N


0.5
0.4
0.3
0.2
0.1
BDNF_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

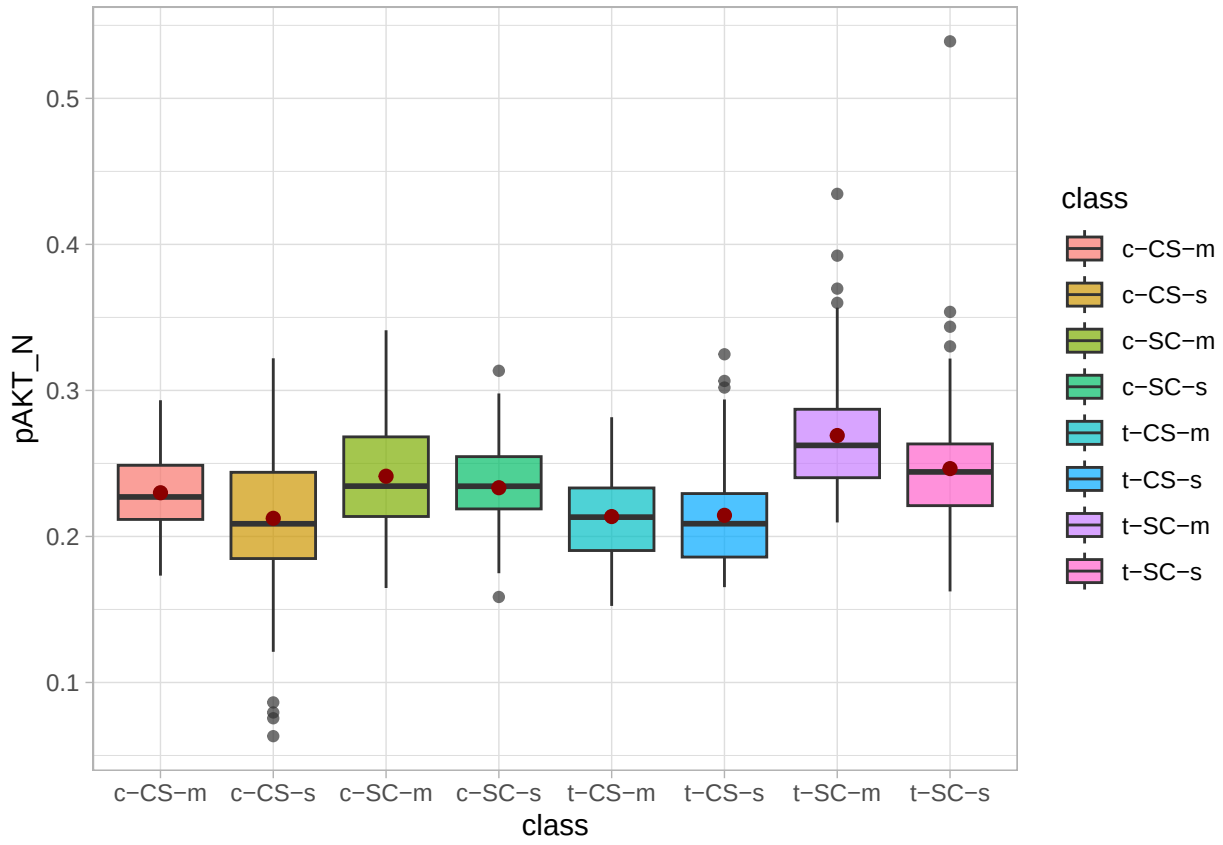
BDNF_N


NR1_N
3
2
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

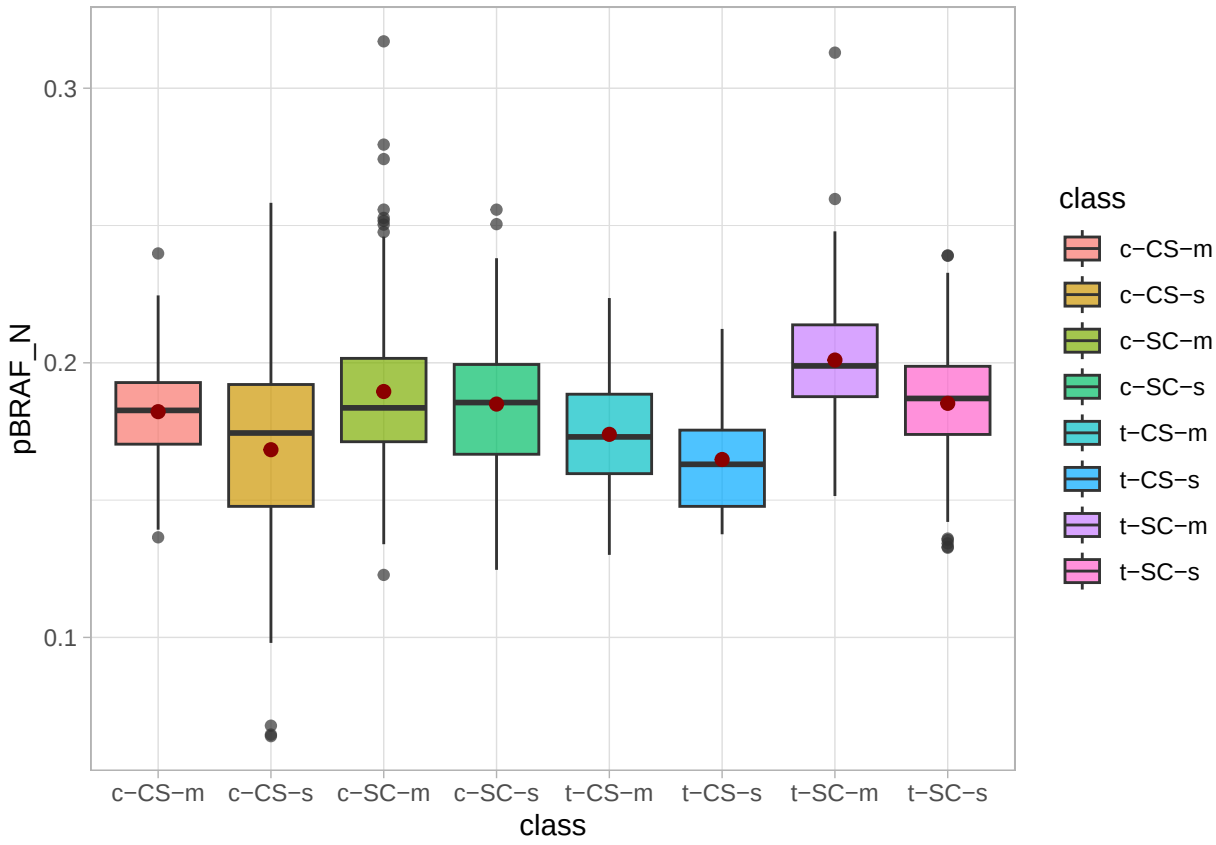
NR1_N


NR2A_N
7
5
3
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

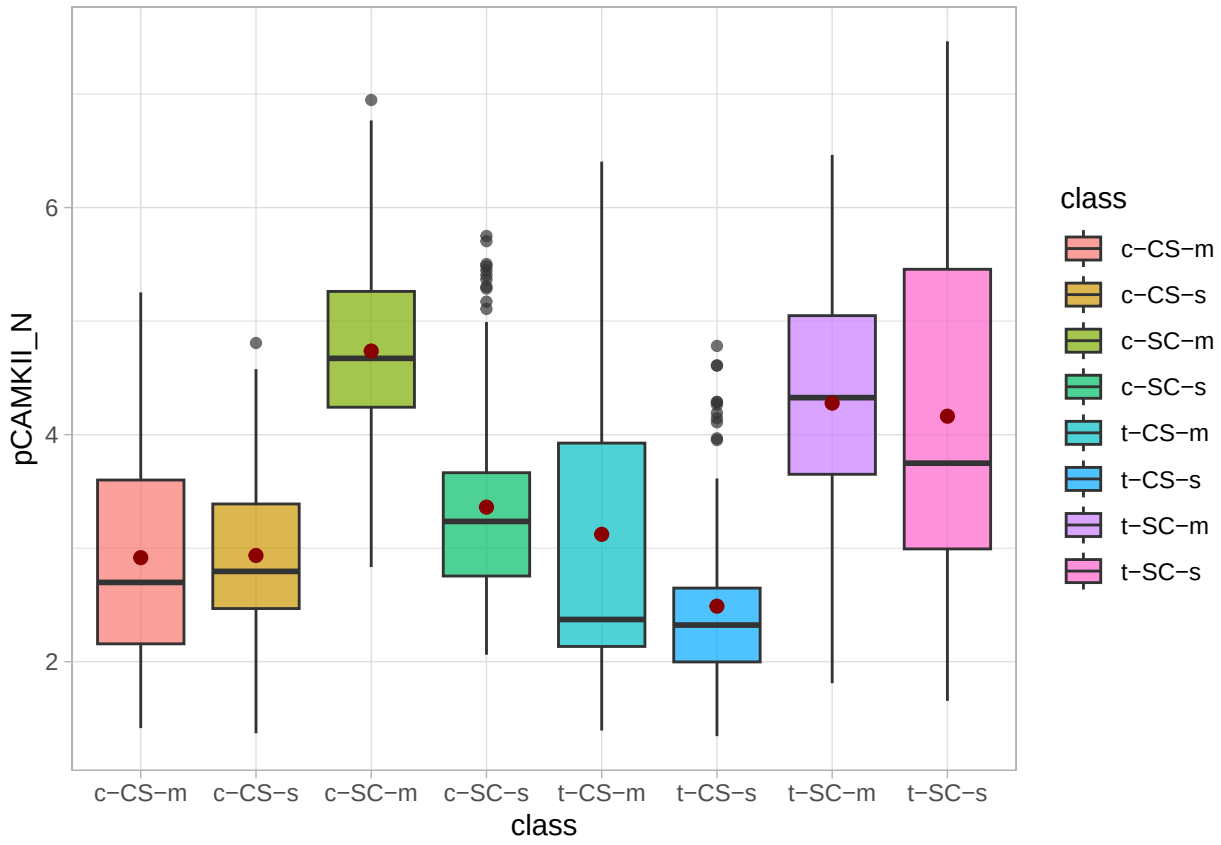
NR2A_N


0.5
0.4
0.3
0.2
0.1
pAKT_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

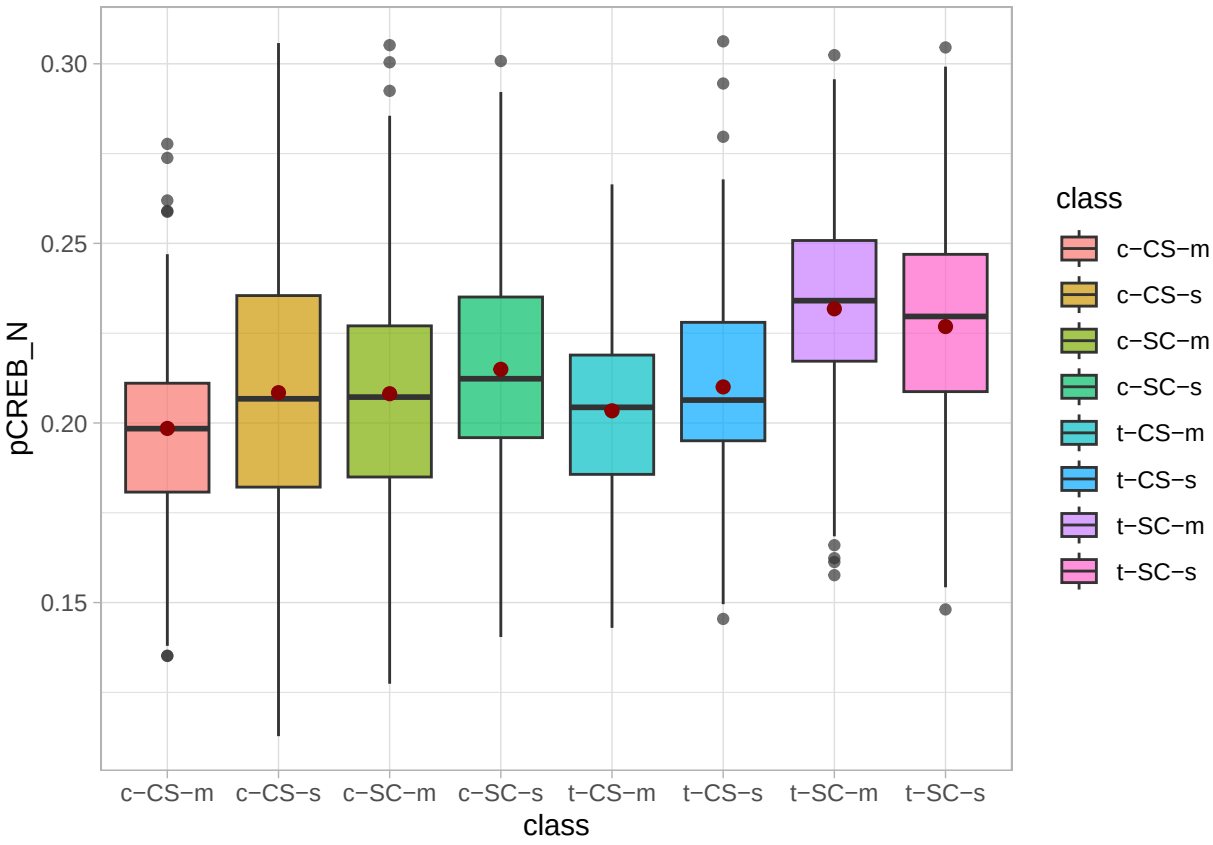
pAKT_N


0.3
0.2
0.1
pBRAF_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

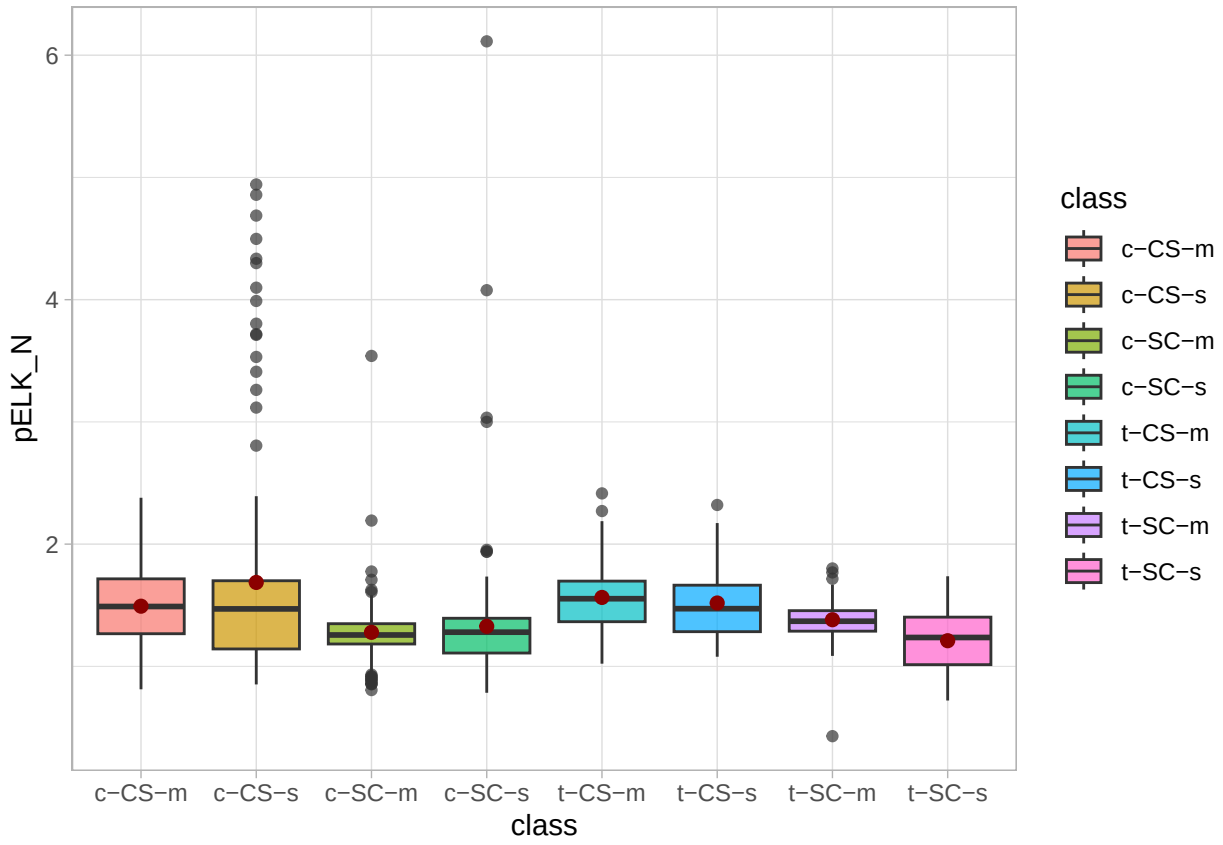
pBRAF_N


pCAMKII_N
6
4
2
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

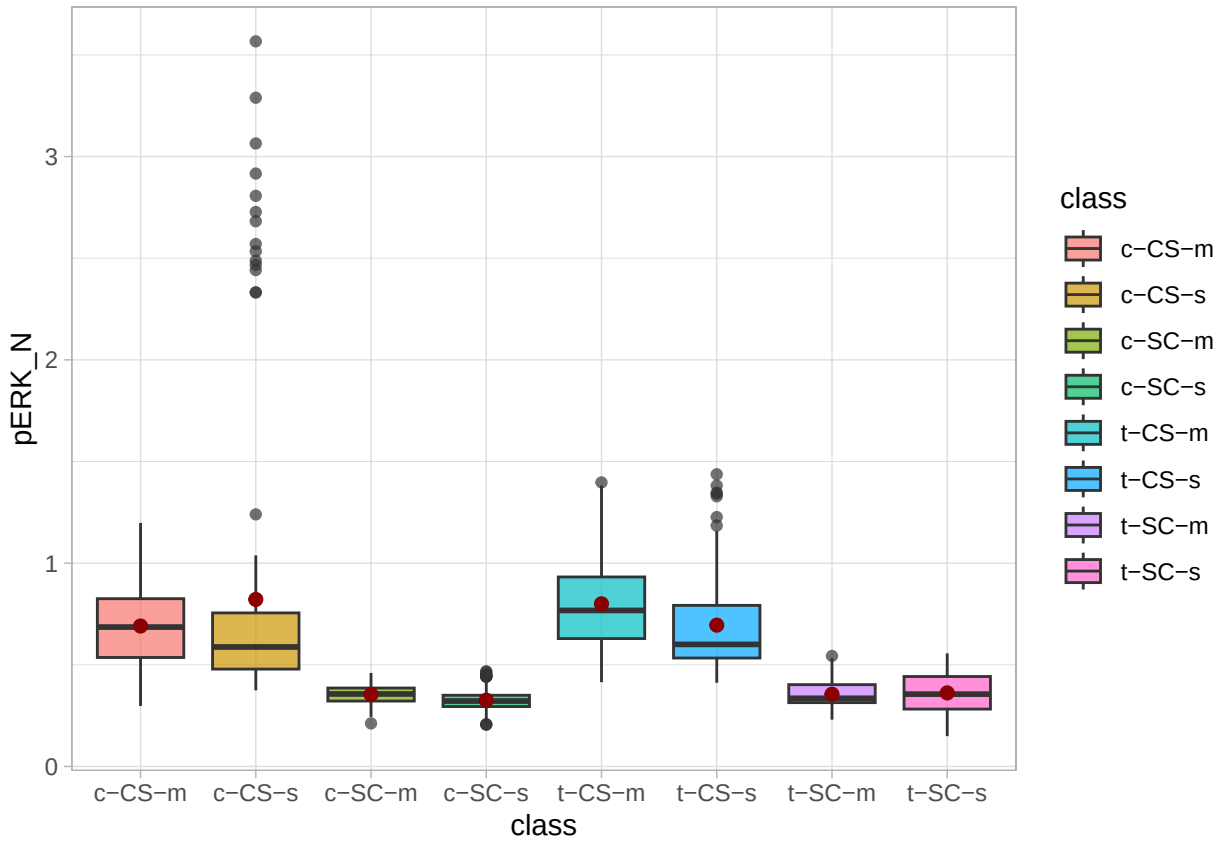
pCAMKII_N


0.30
0.25
0.20
0.15
pCREB_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

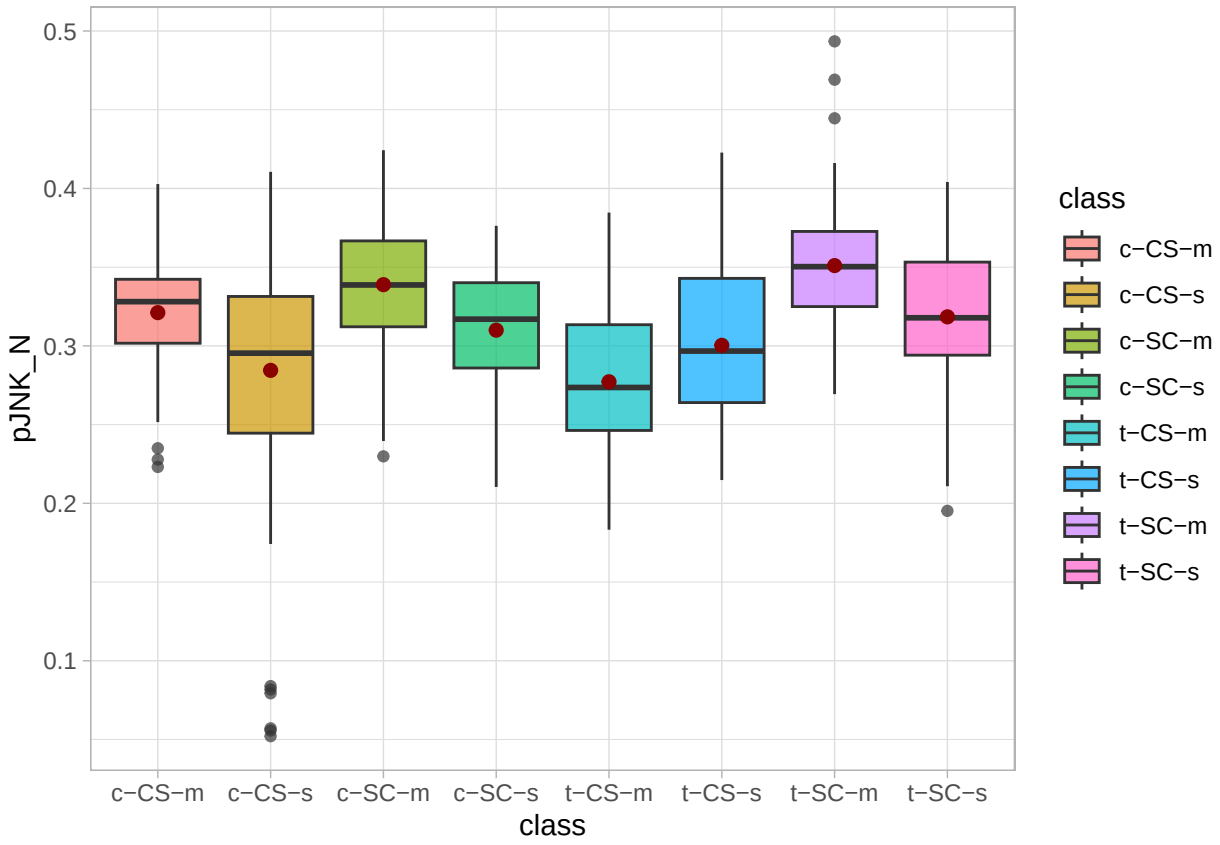
pCREB_N


6
4
2
pELK_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

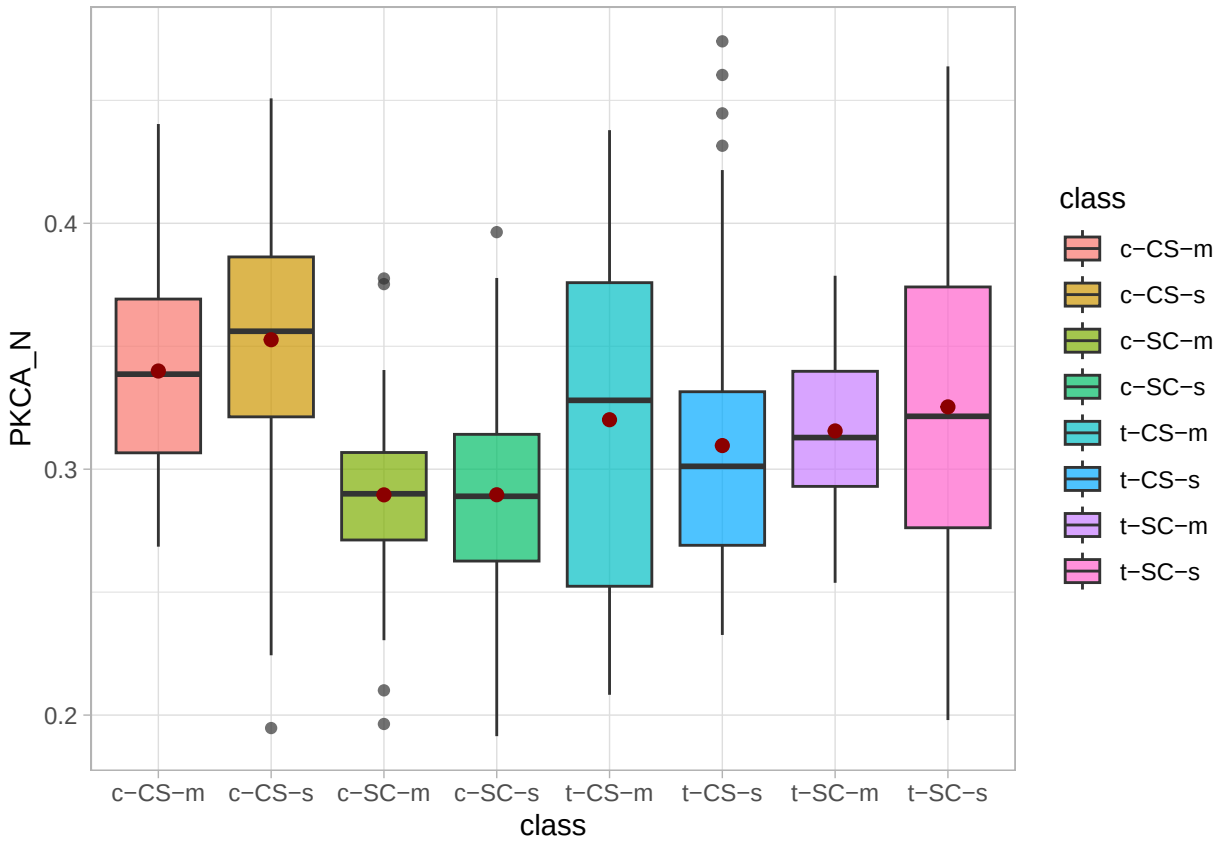
pELK_N


pERK_N
0
1
2
3
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

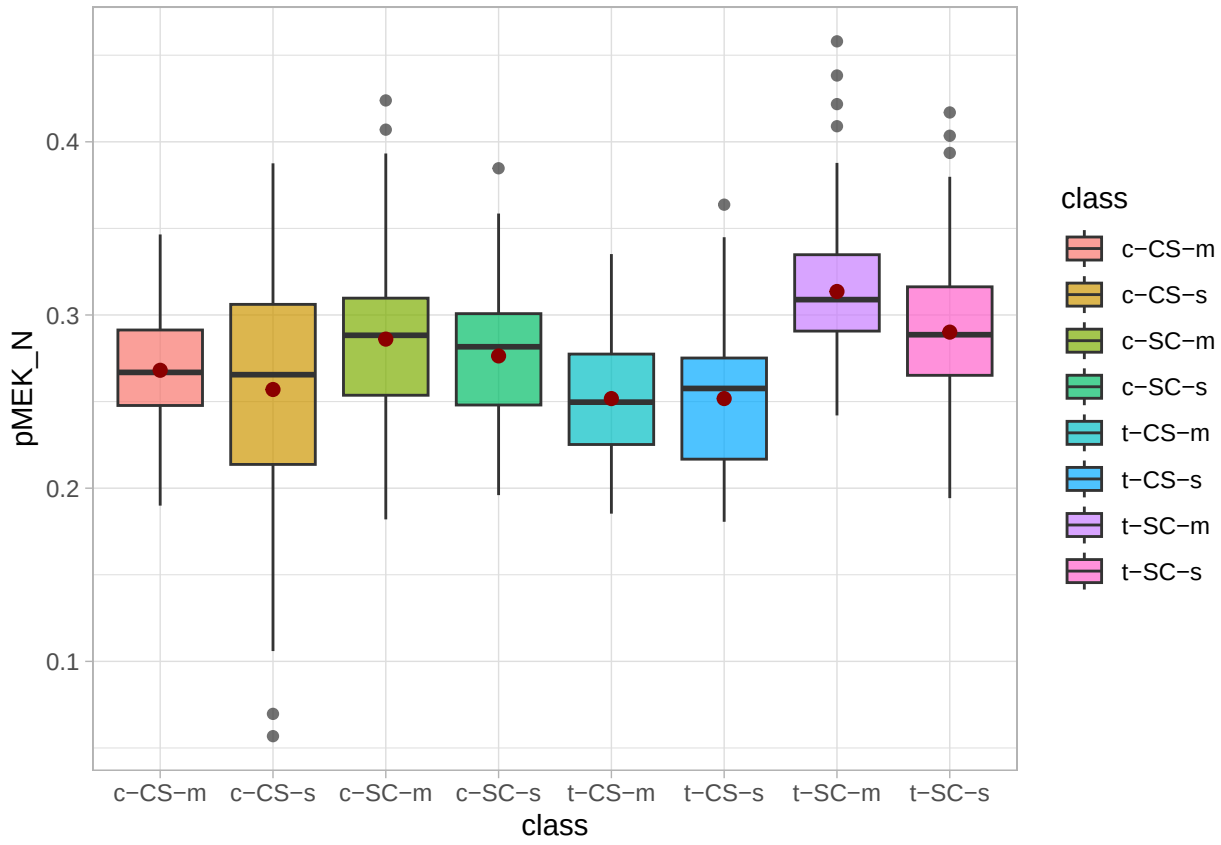
pERK_N


0.5
0.4
0.3
0.2
0.1
pJNK_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

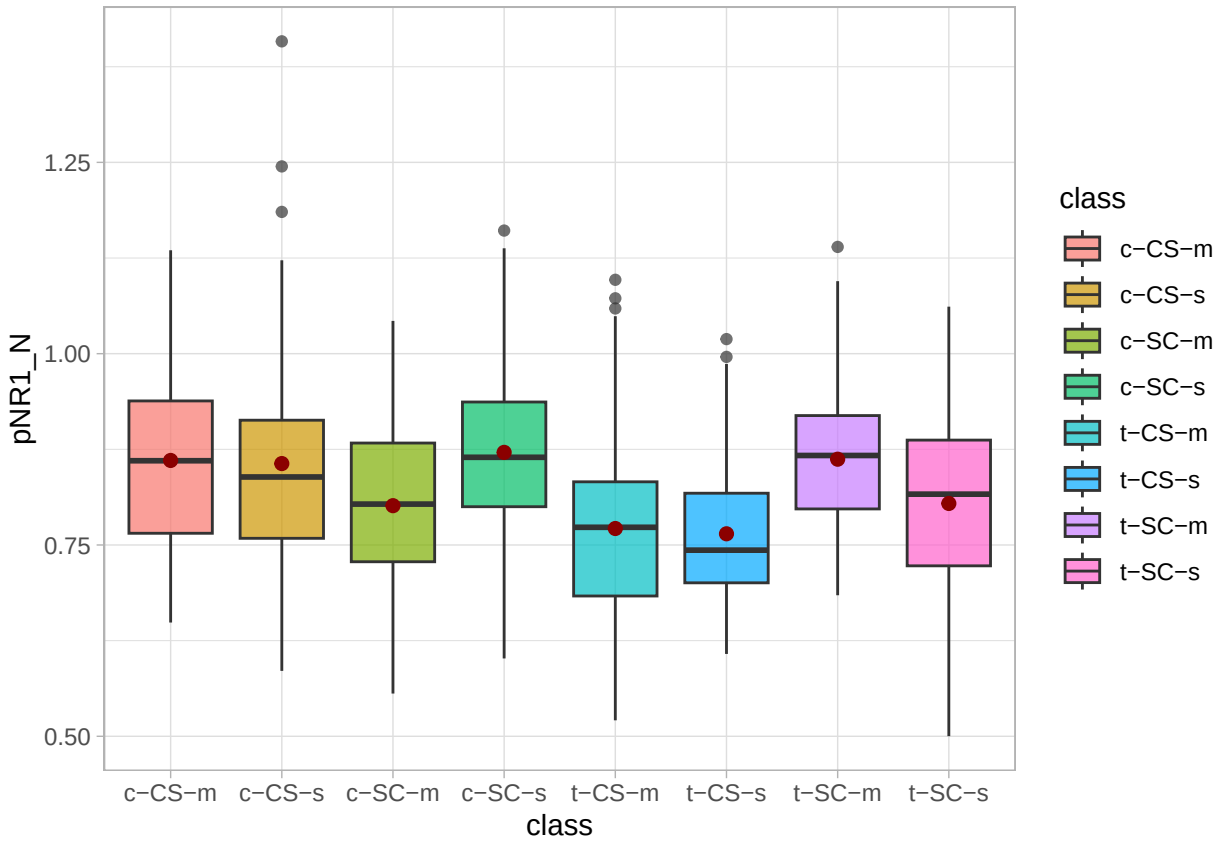
pJNK_N


PKCA_N
0.4
0.3
0.2
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

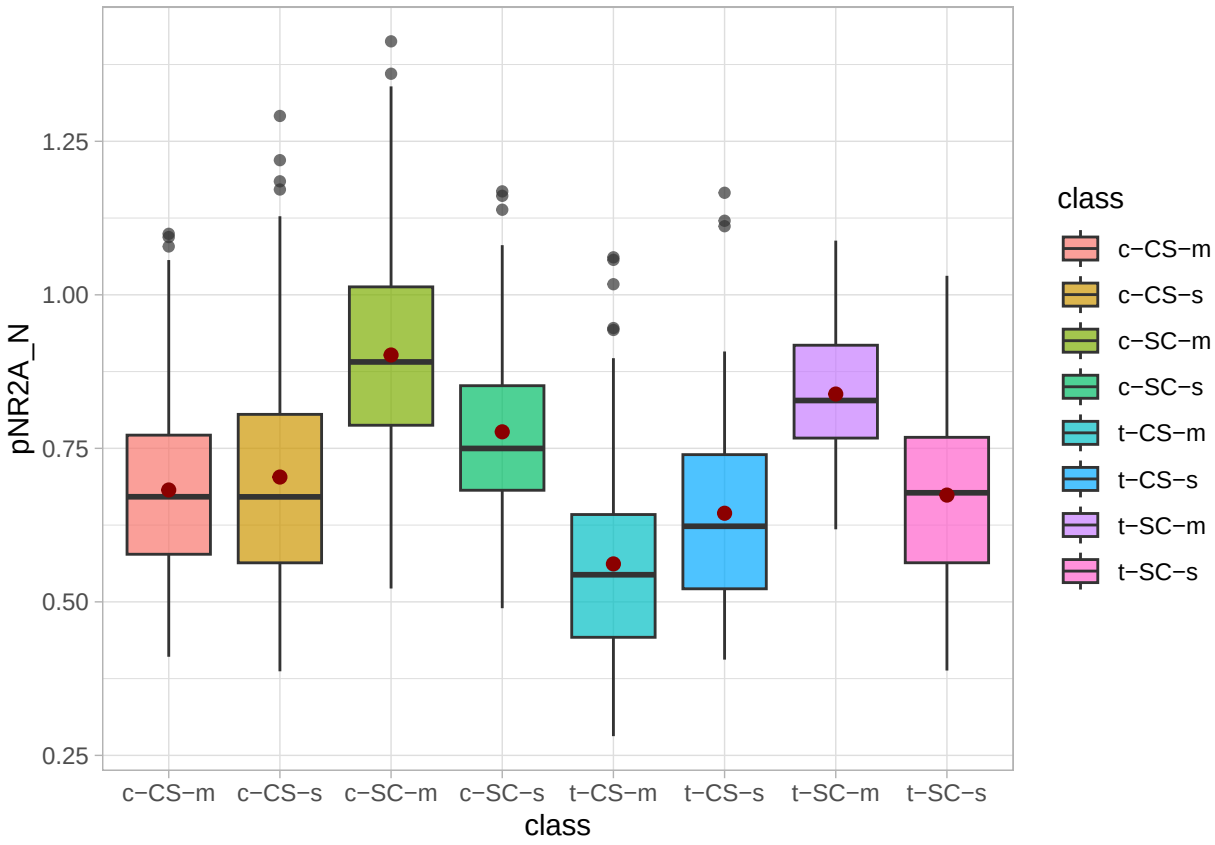
PKCA_N


pMEK_N
0.4
0.3
0.2
0.1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

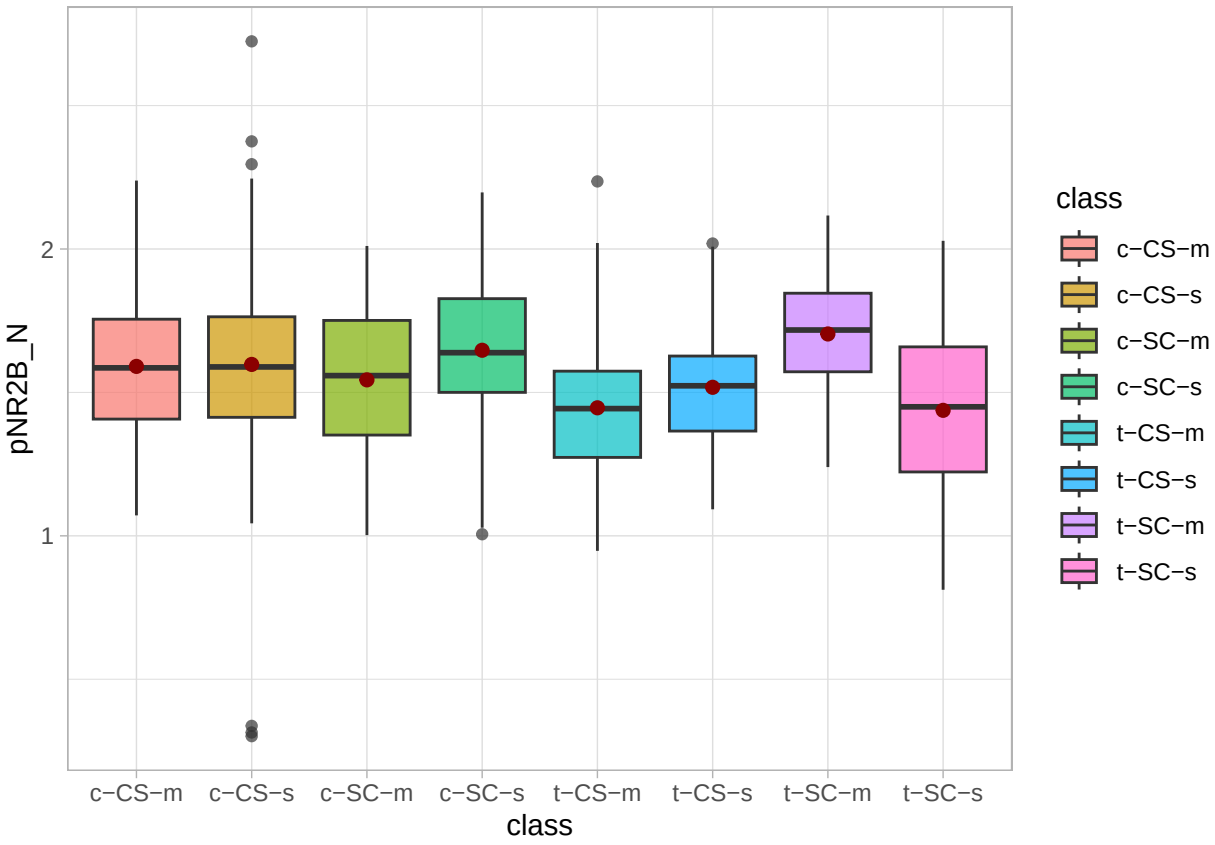
pMEK_N


1.25
1.00
0.75
0.50
pNR1_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

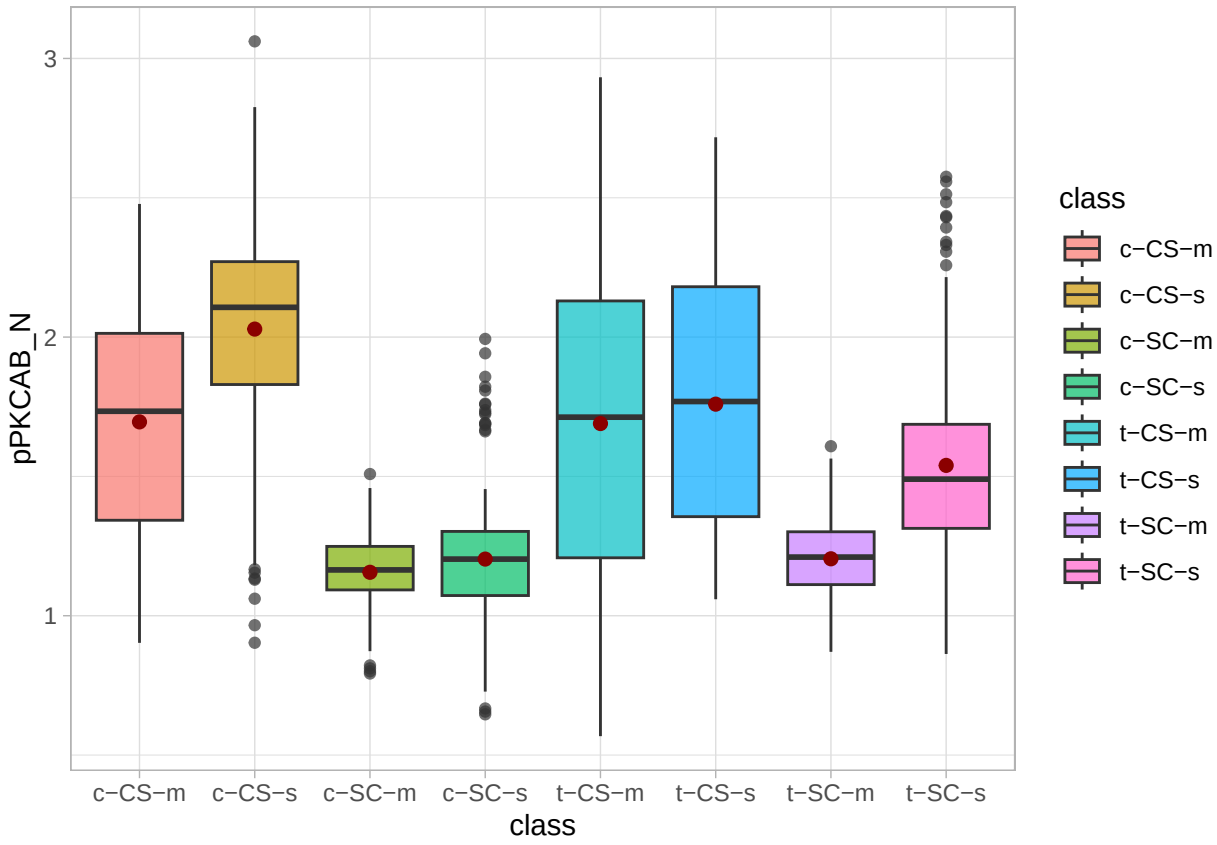
pNR1_N


pNR2A_N
1.25
1.00
0.75
0.50
0.25
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

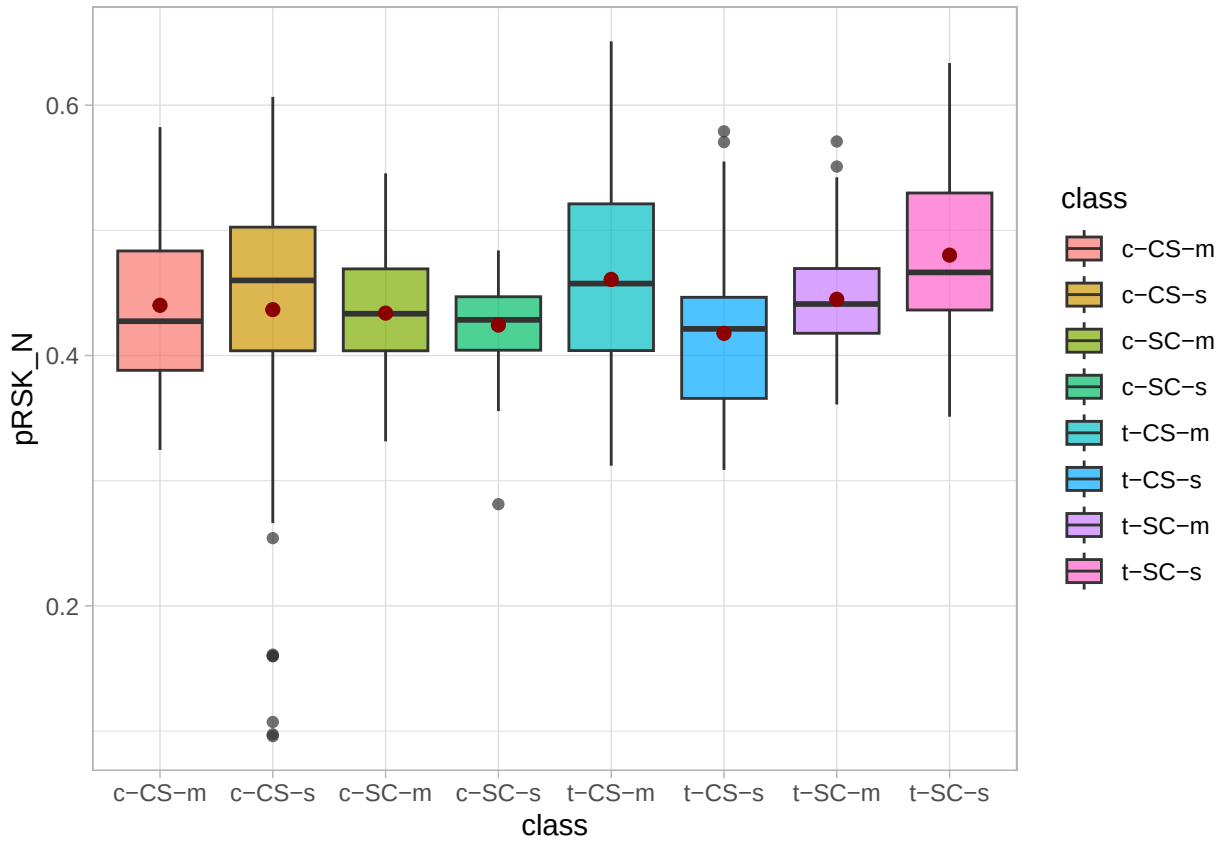
pNR2A_N


pNR2B_N
2
1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

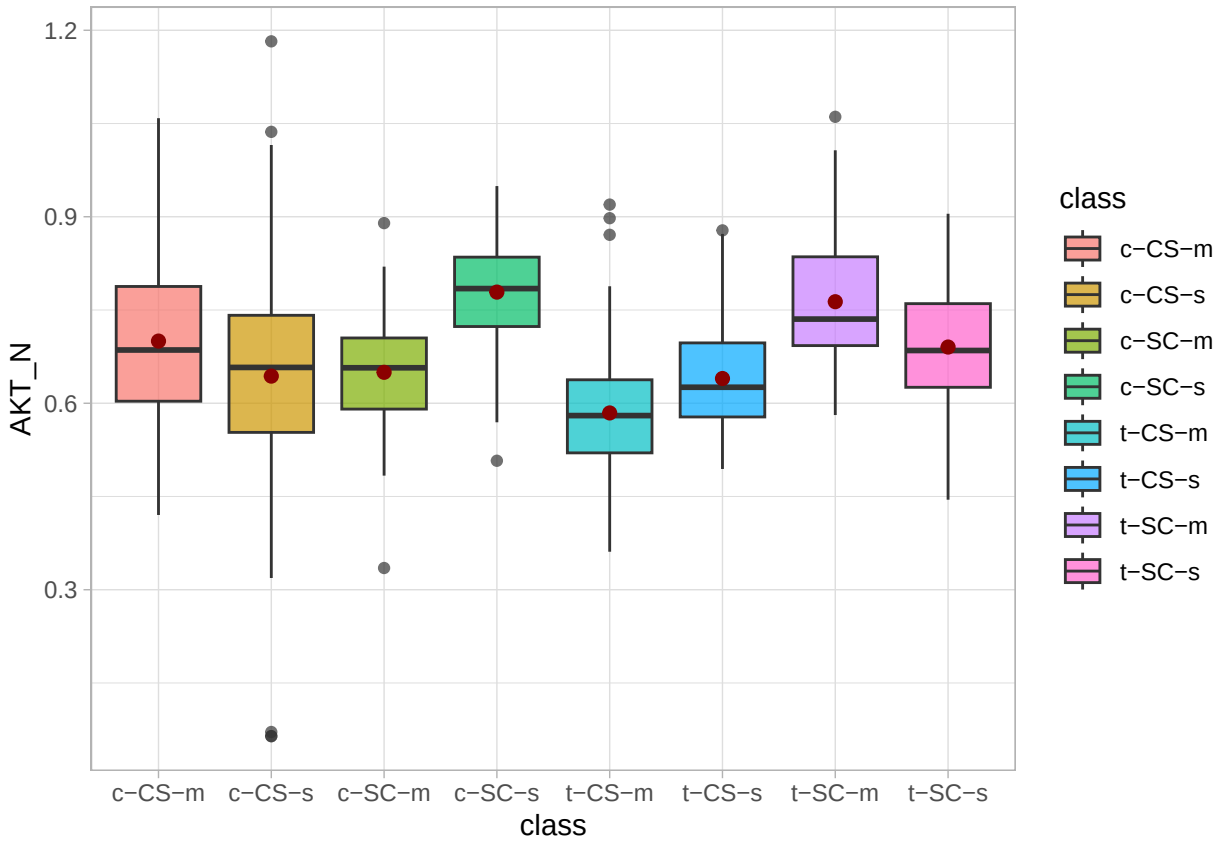
pNR2B_N


pPKCAB_N
3
2
1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

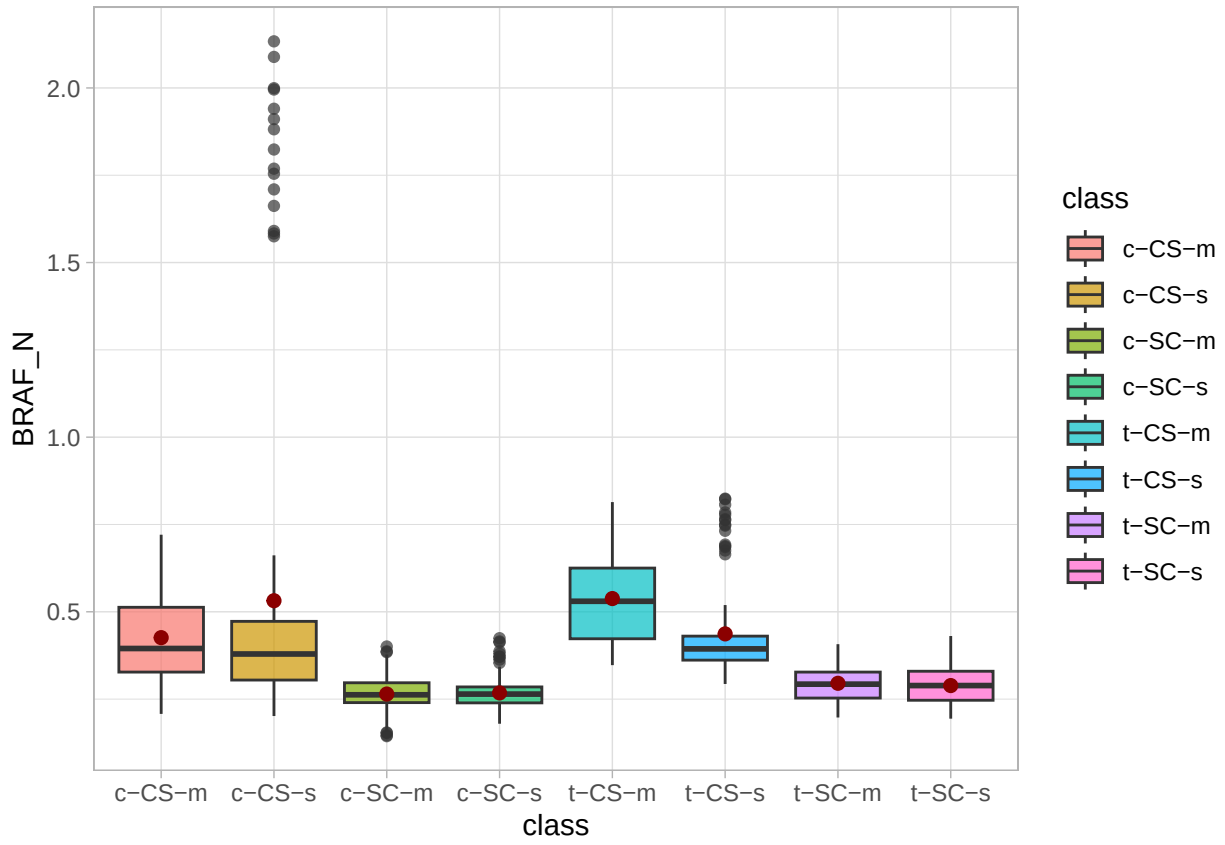
pPKCAB_N


0.6
0.4
0.2
pRSK_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

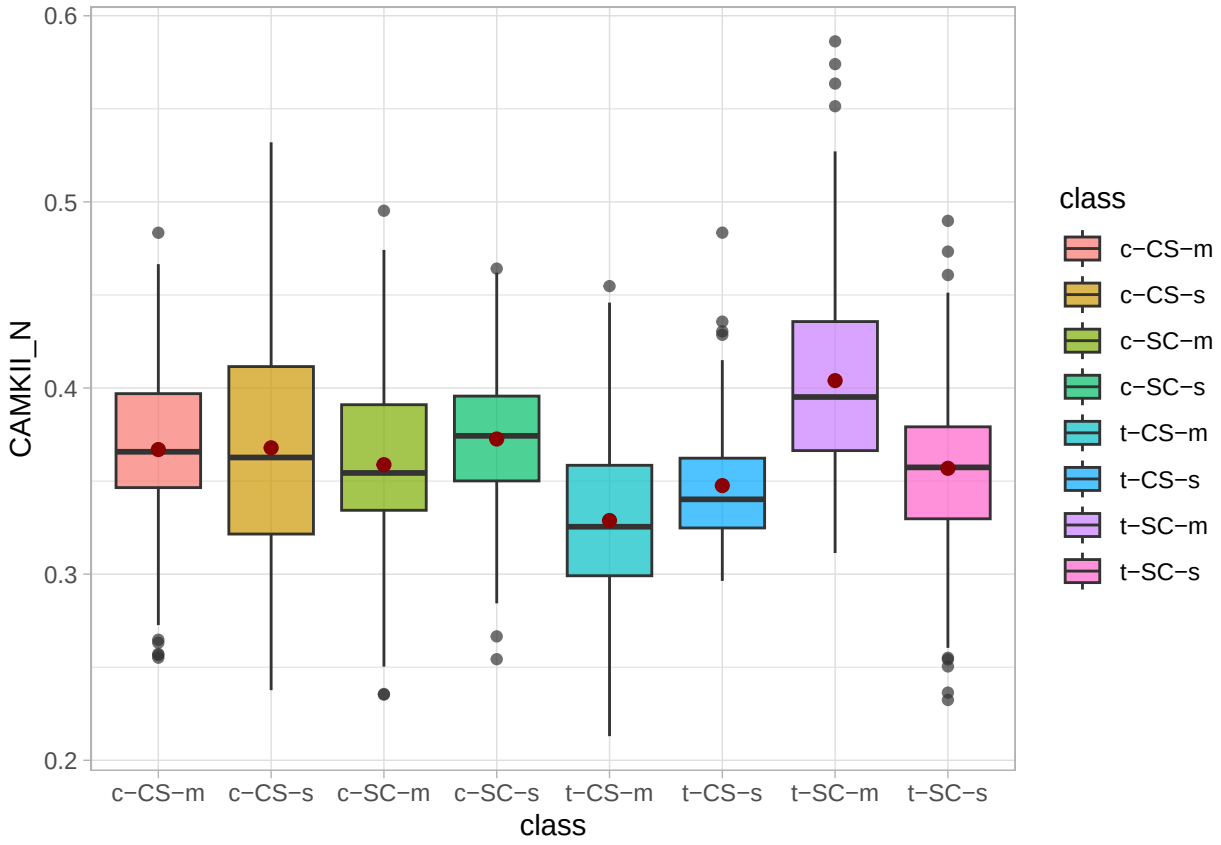
pRSK_N


1.2
0.9
0.6
0.3
AKT_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

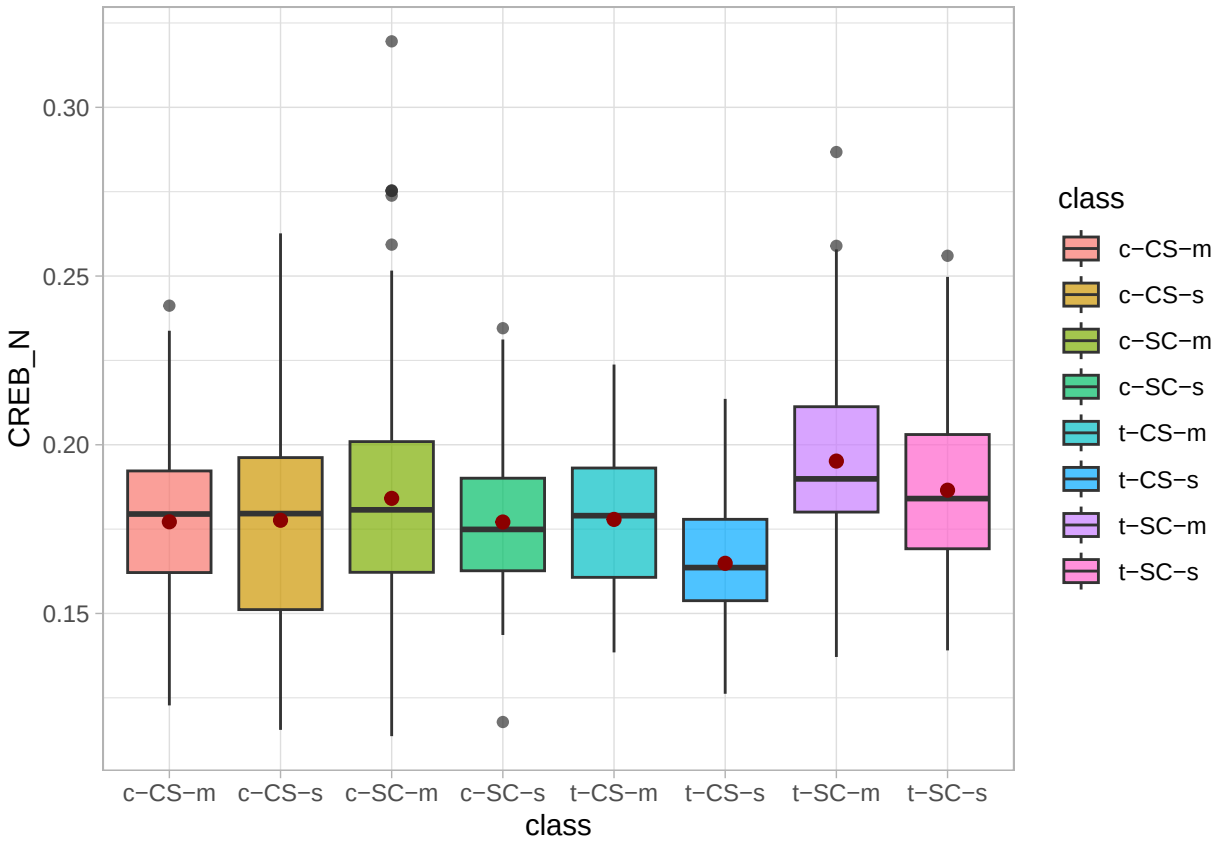
AKT_N


BRAF_N
2.0
1.5
1.0
0.5
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

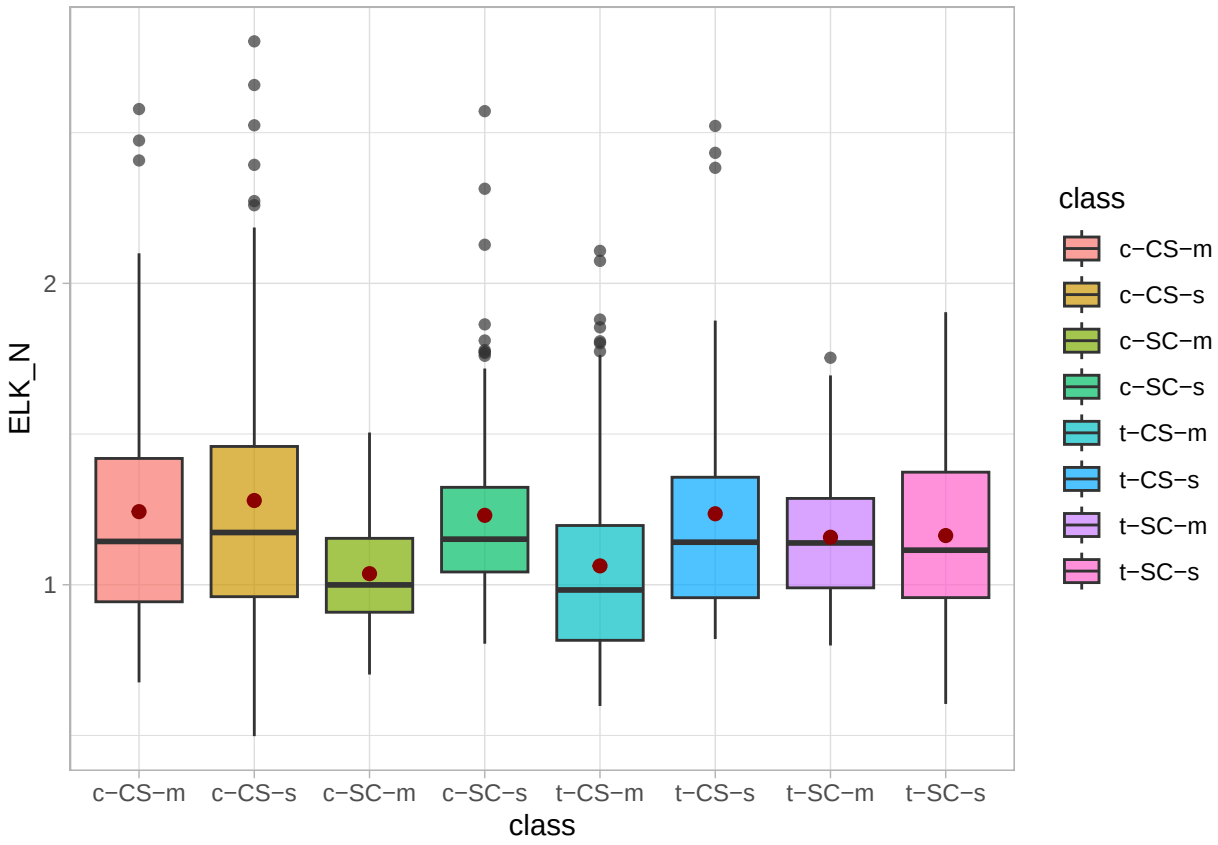
BRAF_N


0.6
0.5
0.4
0.3
0.2
CAMKII_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

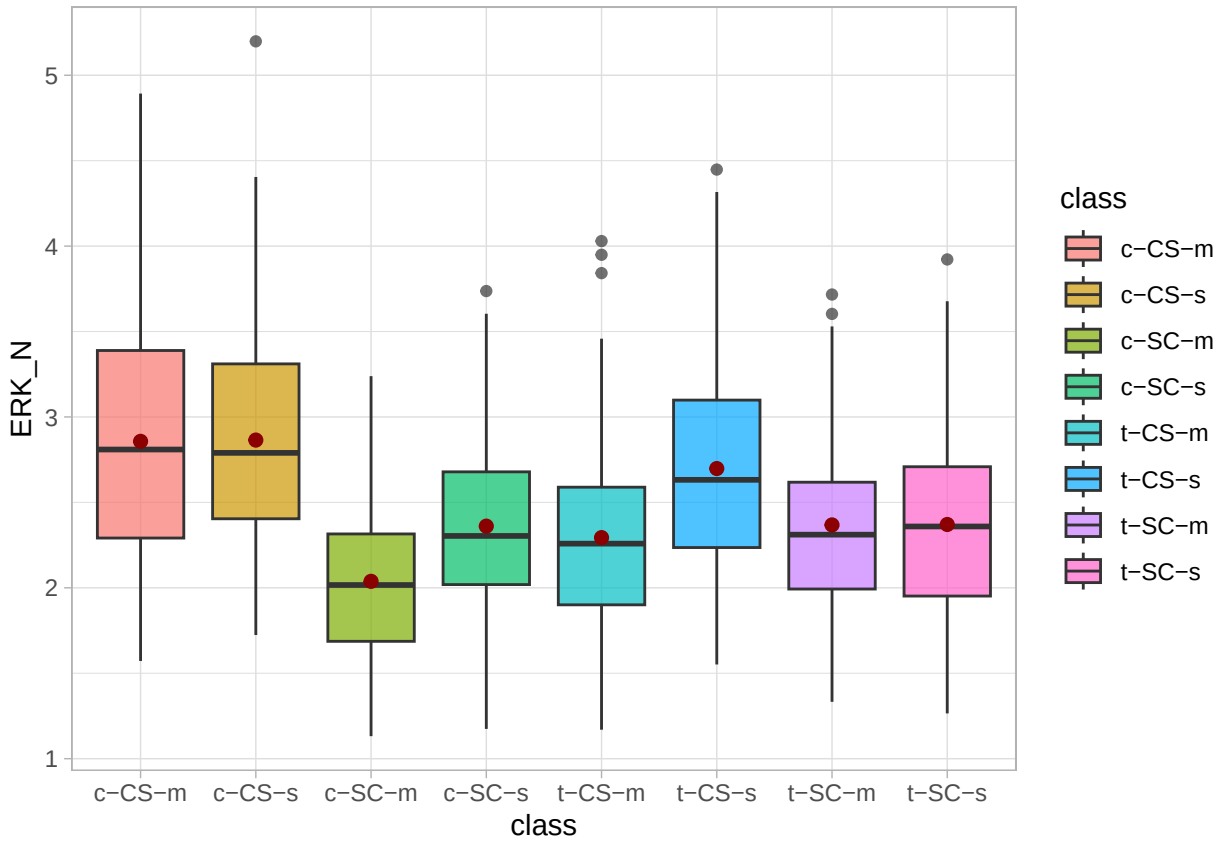
CAMKII_N


0.30
0.25
0.20
0.15
CREB_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

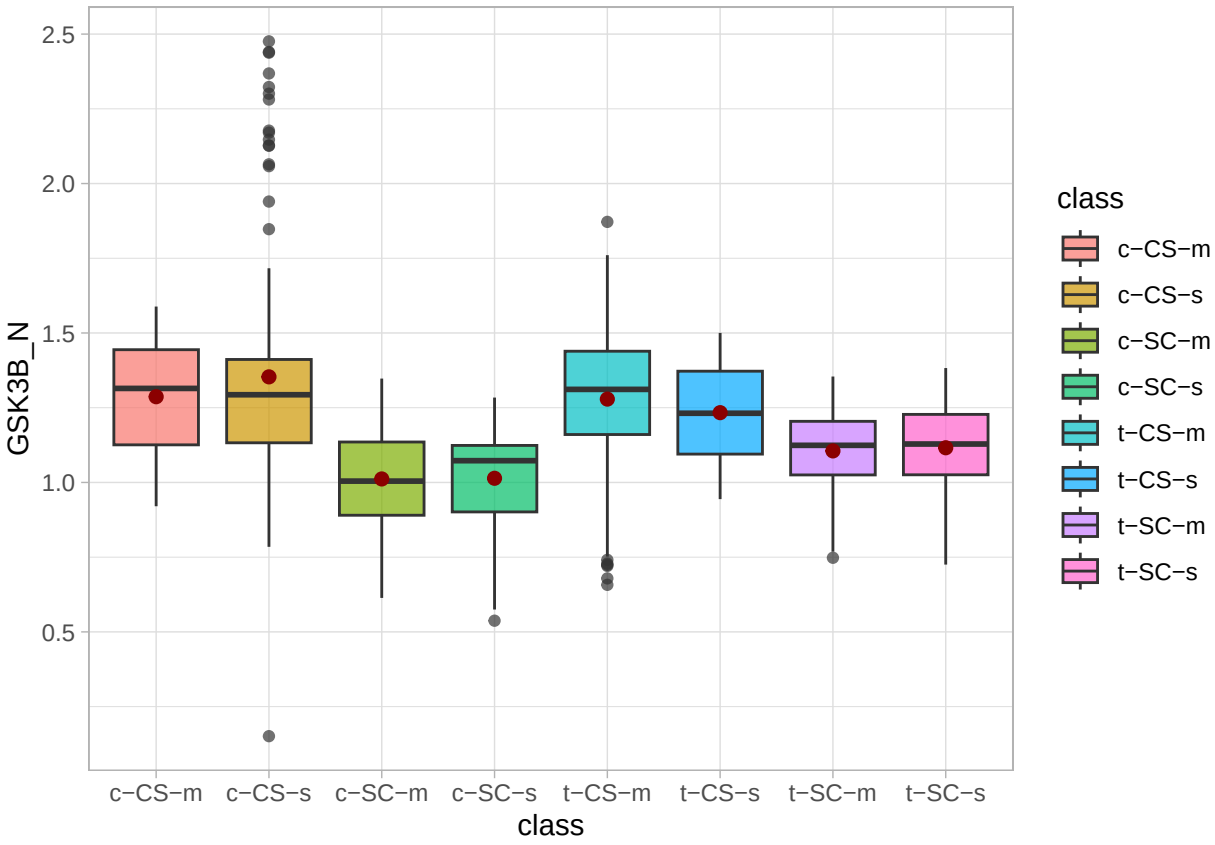
CREB_N


ELK_N
2
1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

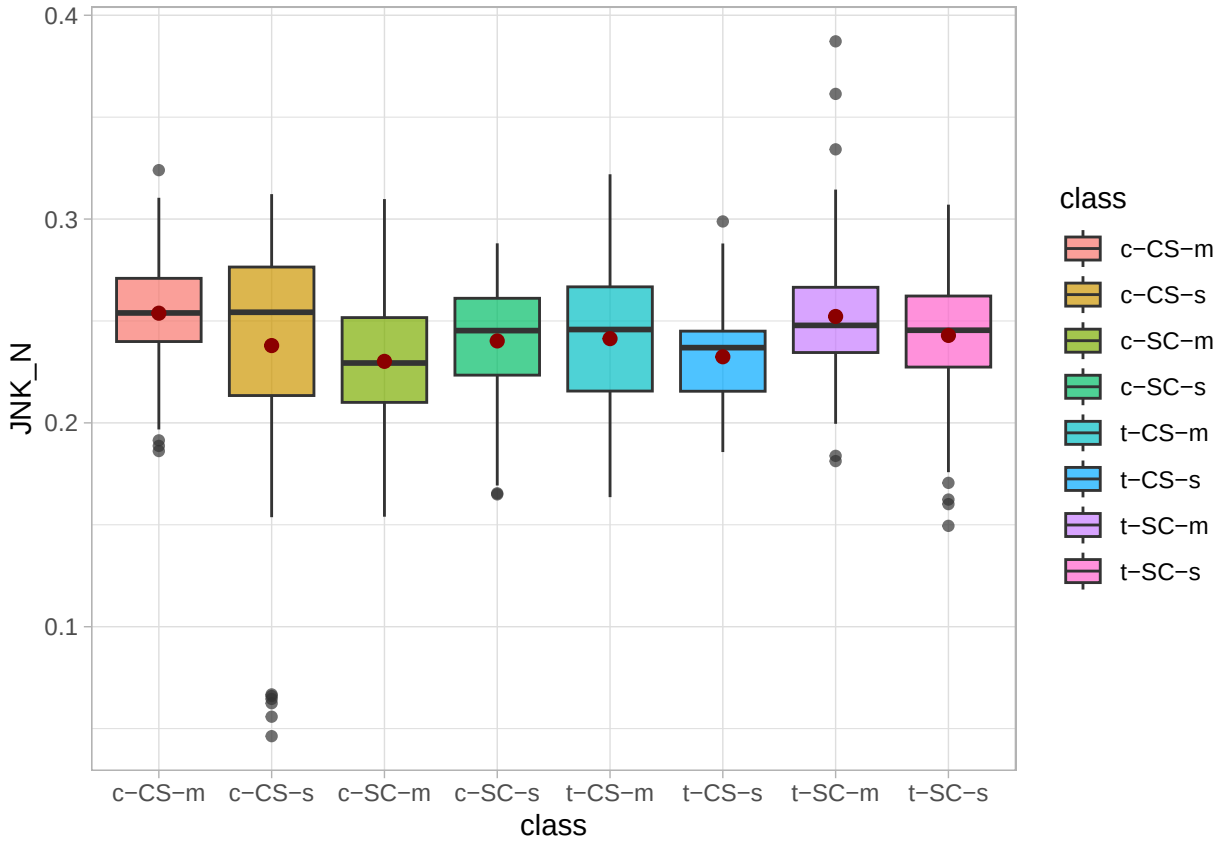
ELK_N


ERK_N
5
4
3
2
1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

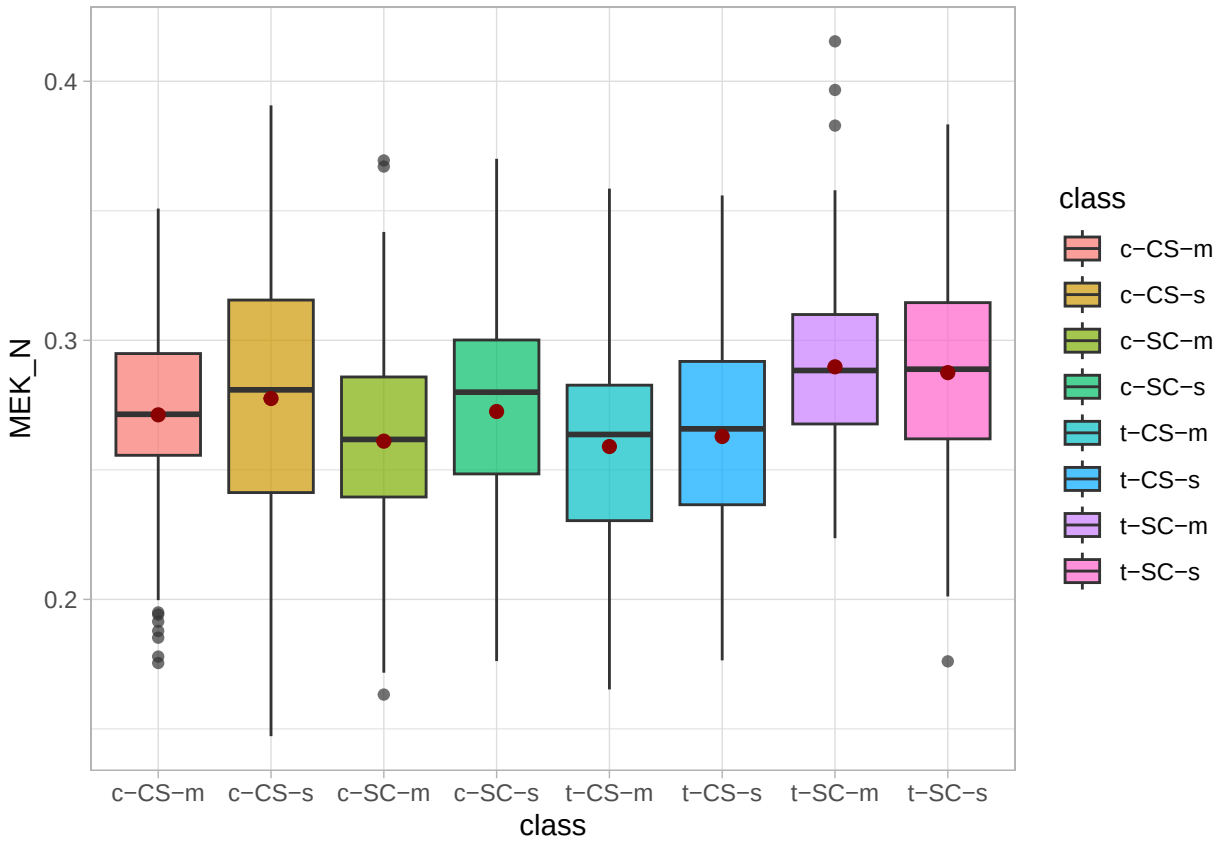
ERK_N


GSK3B_N
2.5
2.0
1.5
1.0
0.5
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

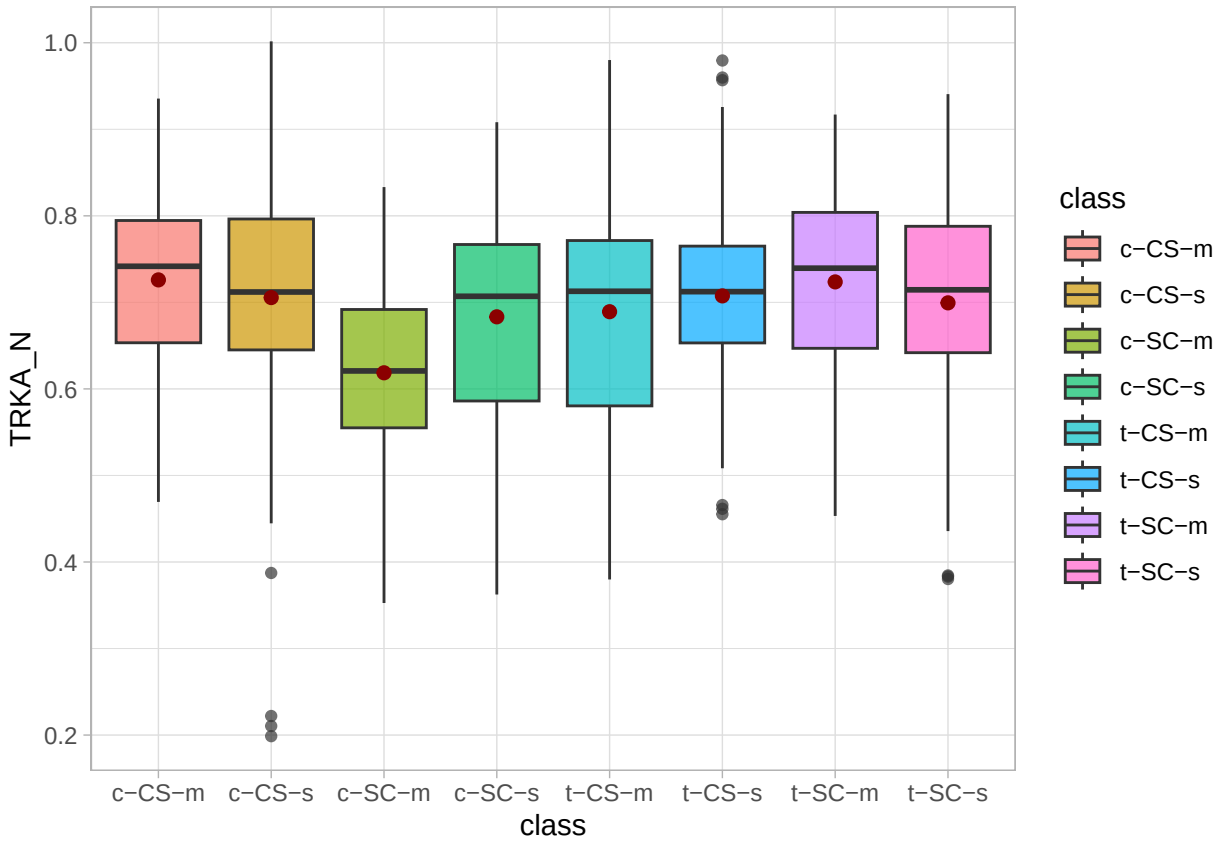
GSK3B_N


0.4
0.3
0.2
0.1
JNK_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

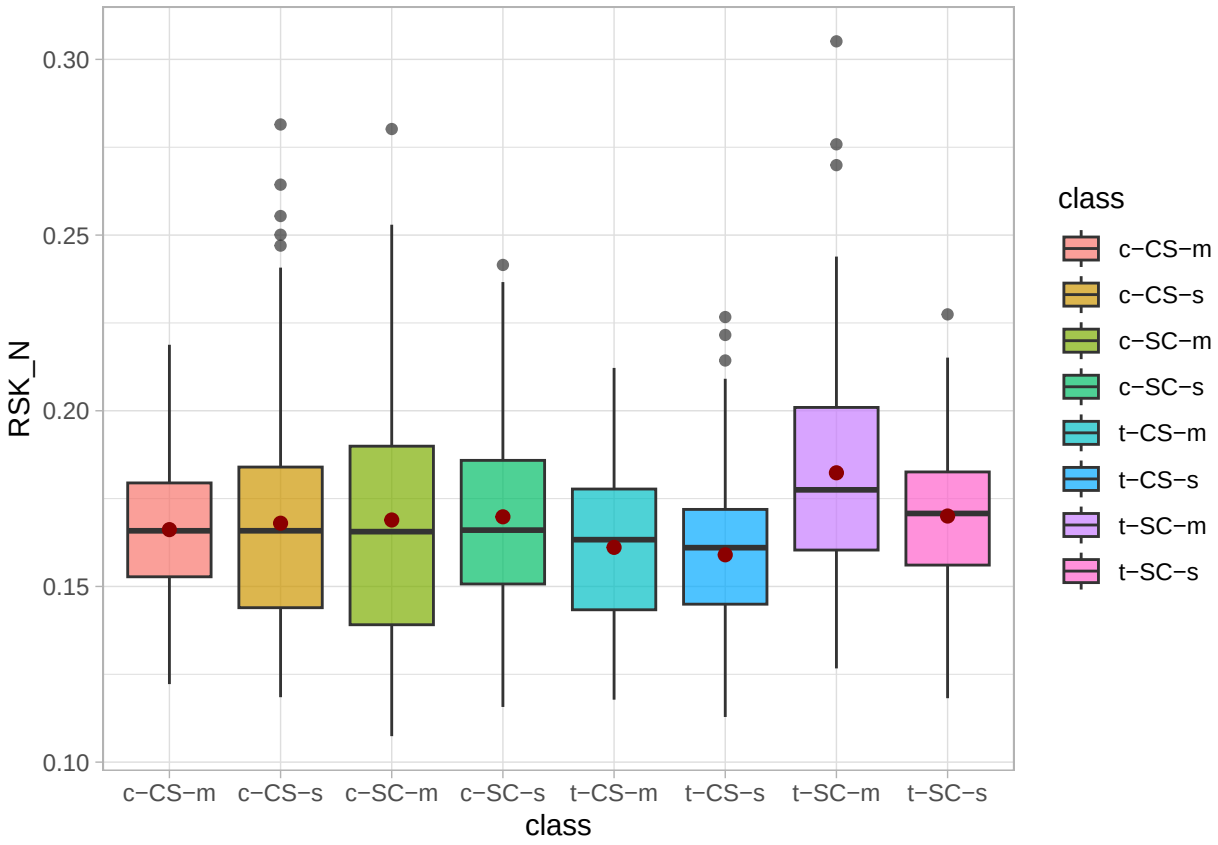
JNK_N


0.4
0.3
0.2
MEK_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

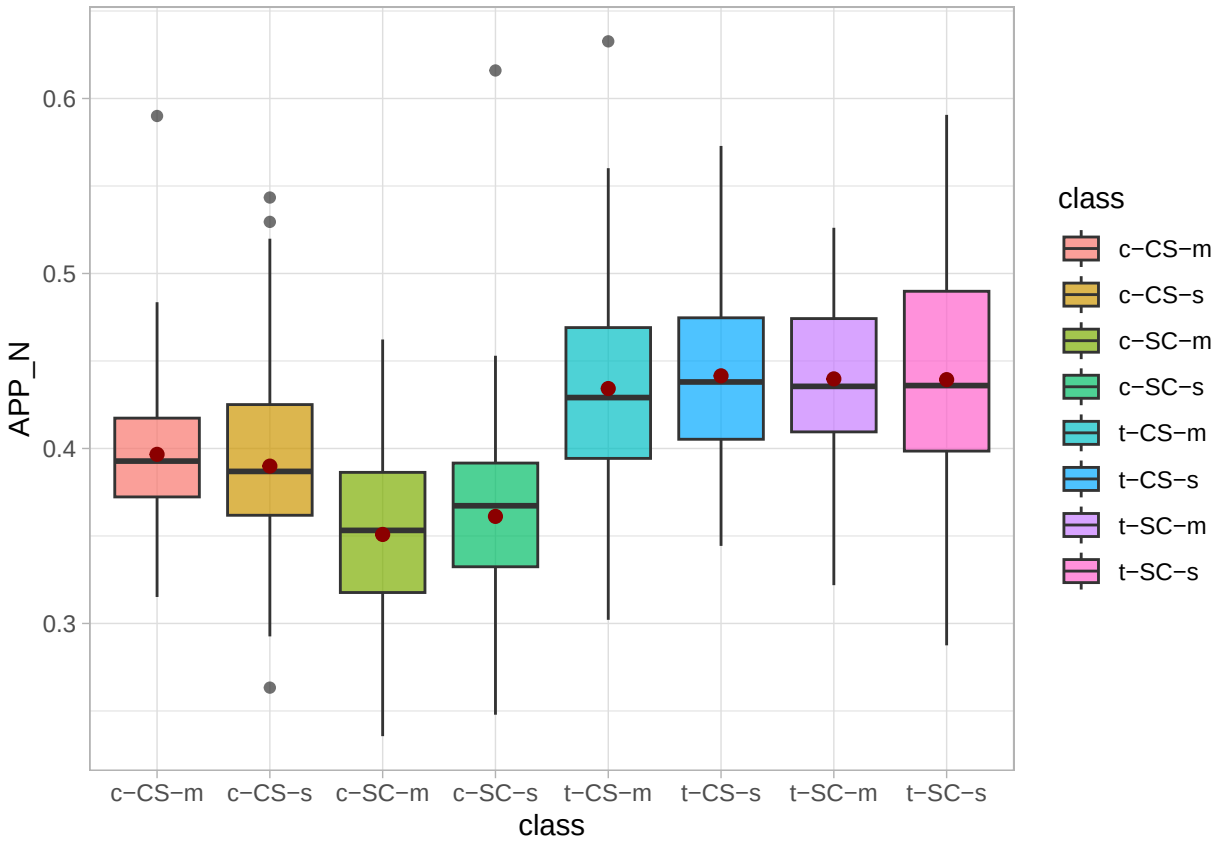
MEK_N


1.0
0.8
0.6
0.4
0.2
TRKA_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

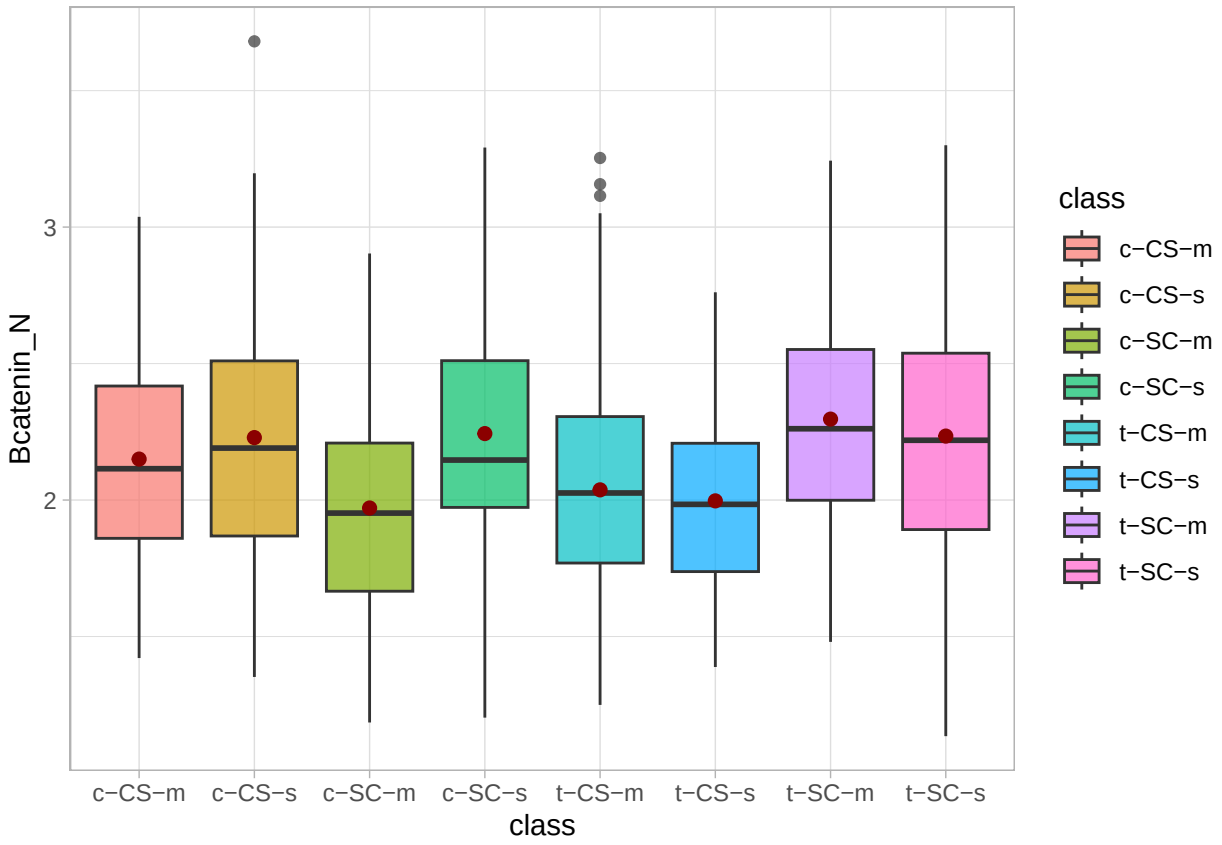
TRKA_N


0.30
0.25
0.20
0.15
0.10
RSK_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

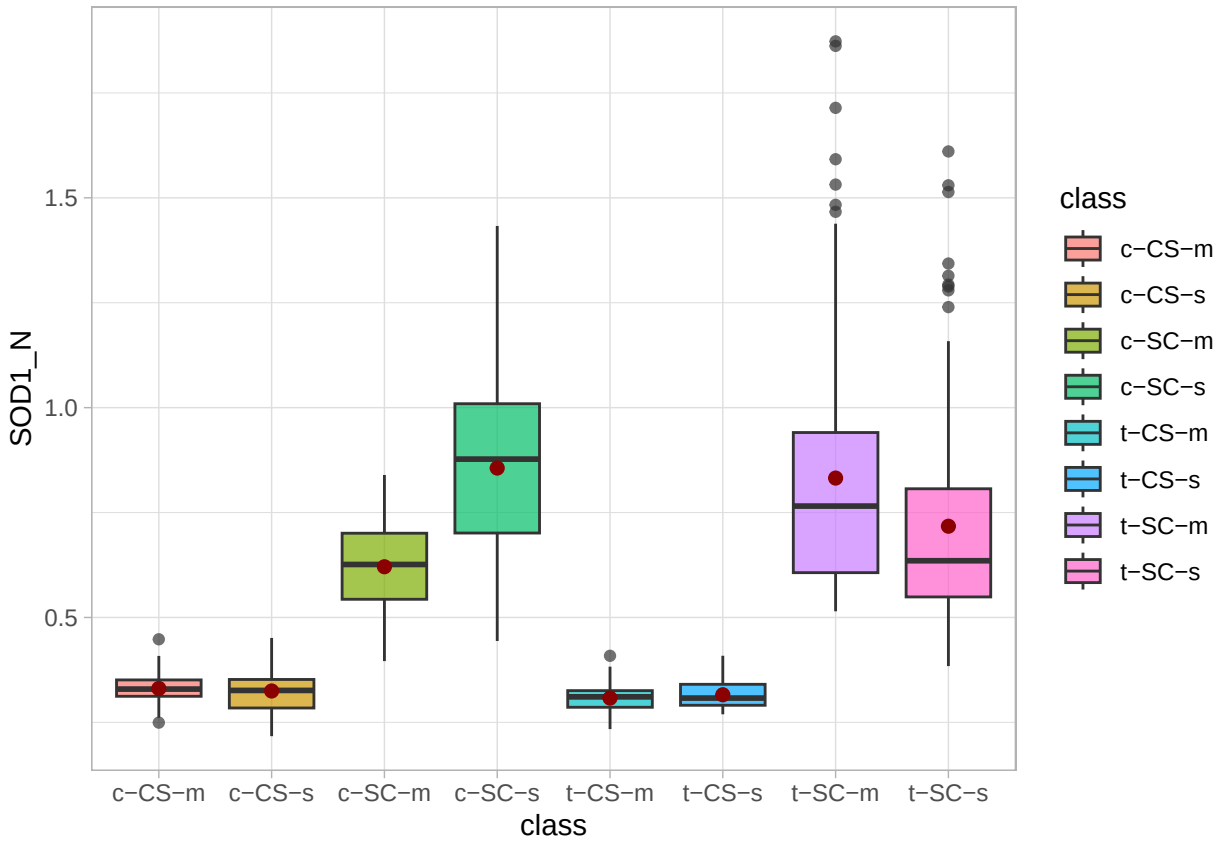
RSK__N


APP_N
0.6
0.5
0.4
0.3
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

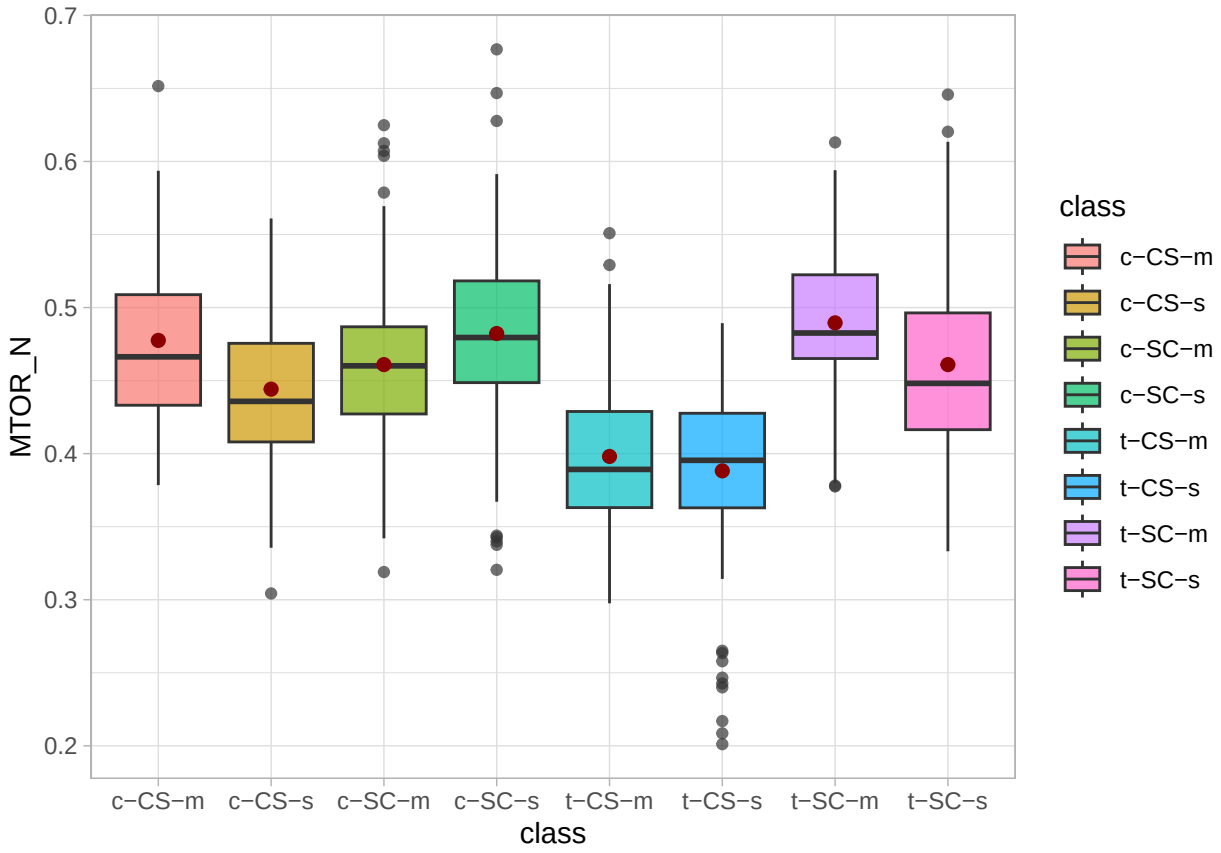
APP_N


Bcatenin_N
3
2
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

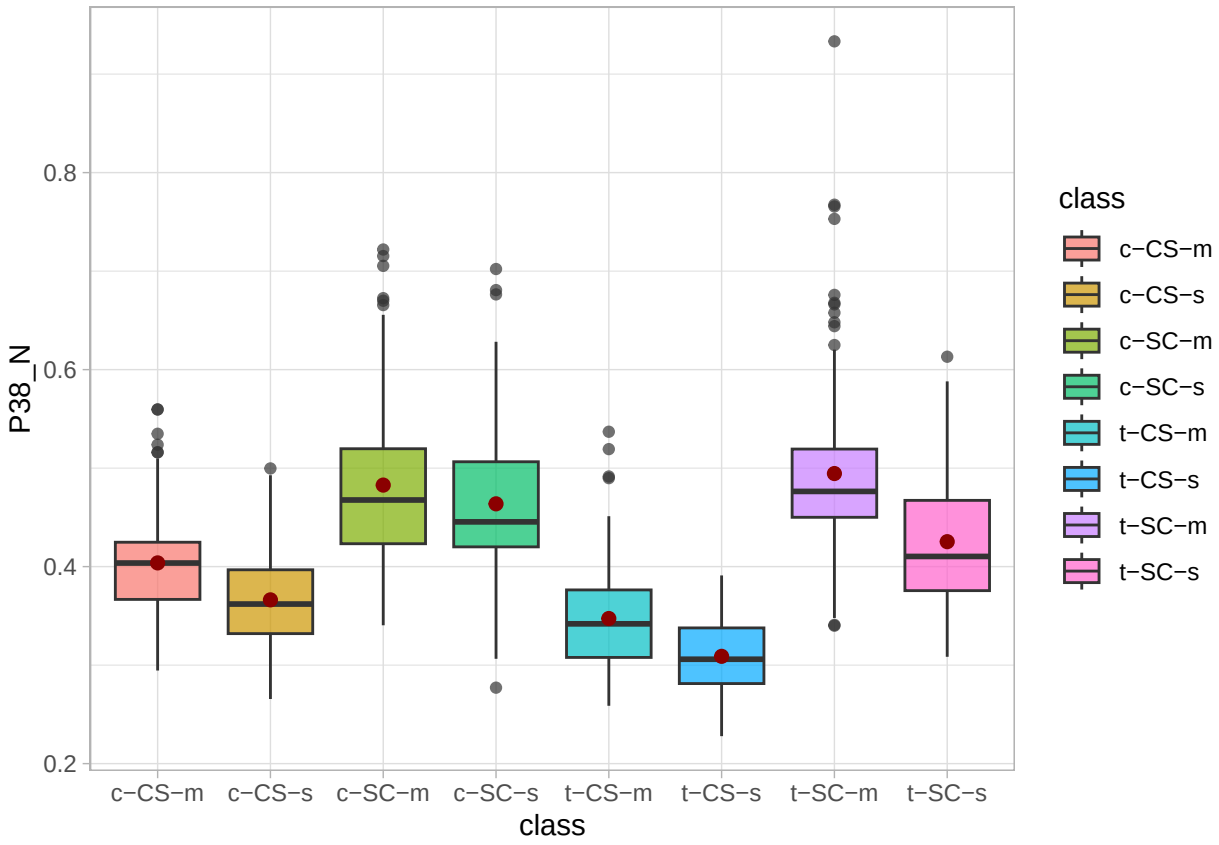
Bcatenin_N


SOD1_N
1.5
1.0
0.5
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

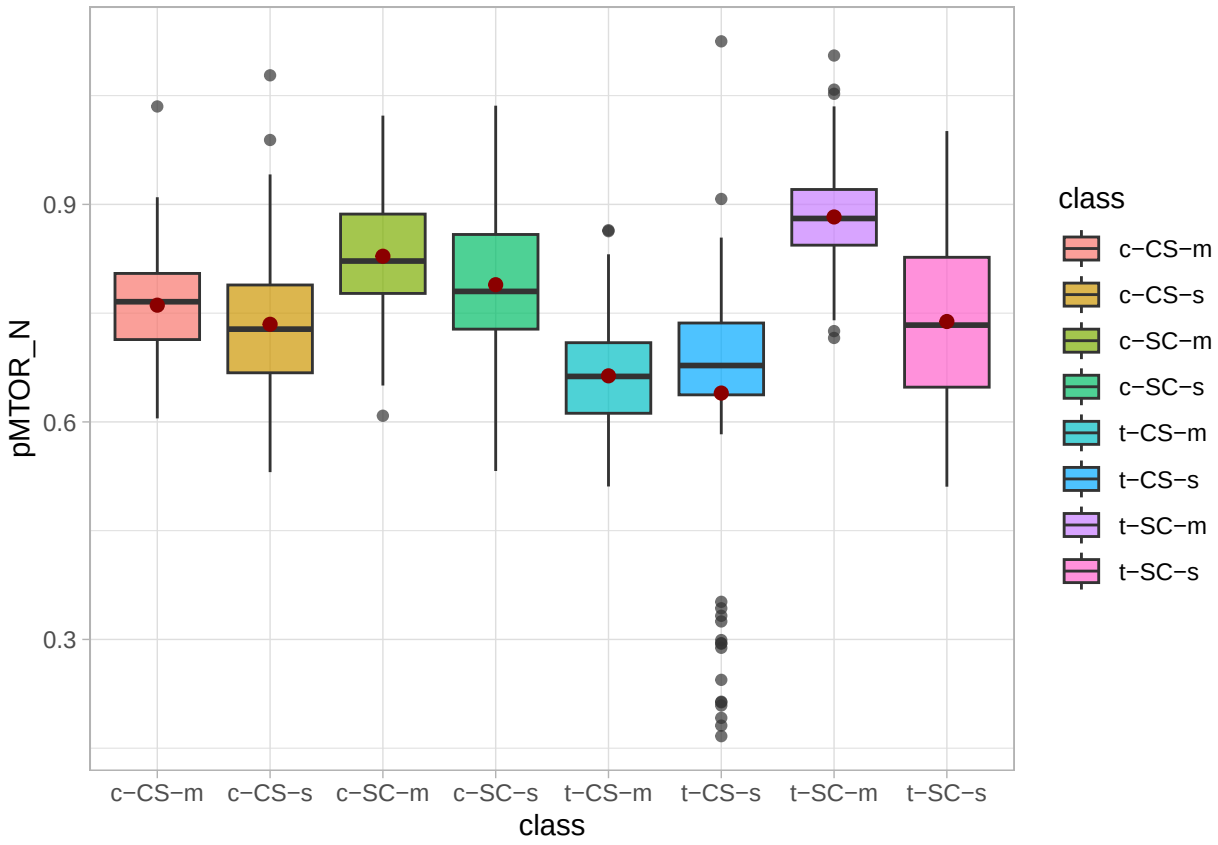
SOD1_N


0.7
0.6
0.5
0.4
0.3
0.2
MTOR_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

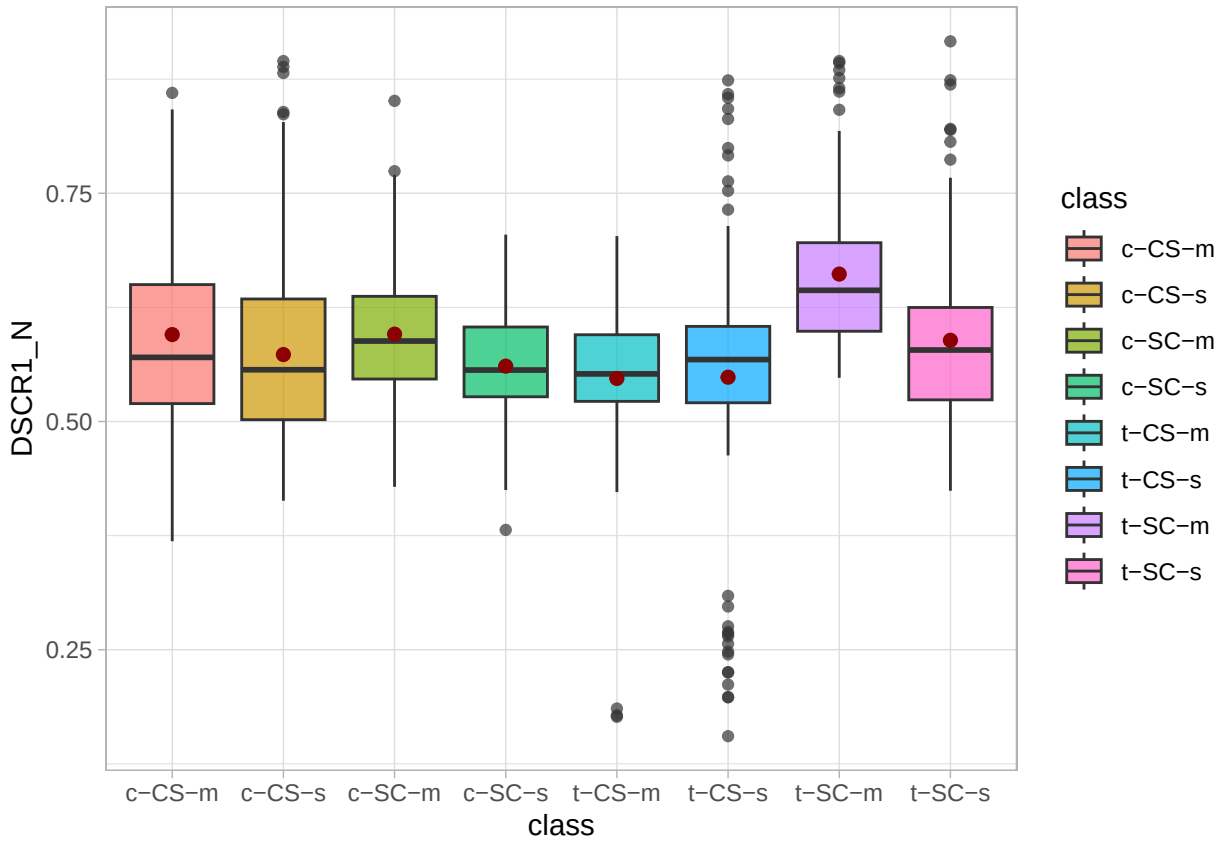
MTOR_N


P38_N
0.8
0.6
0.4
0.2
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

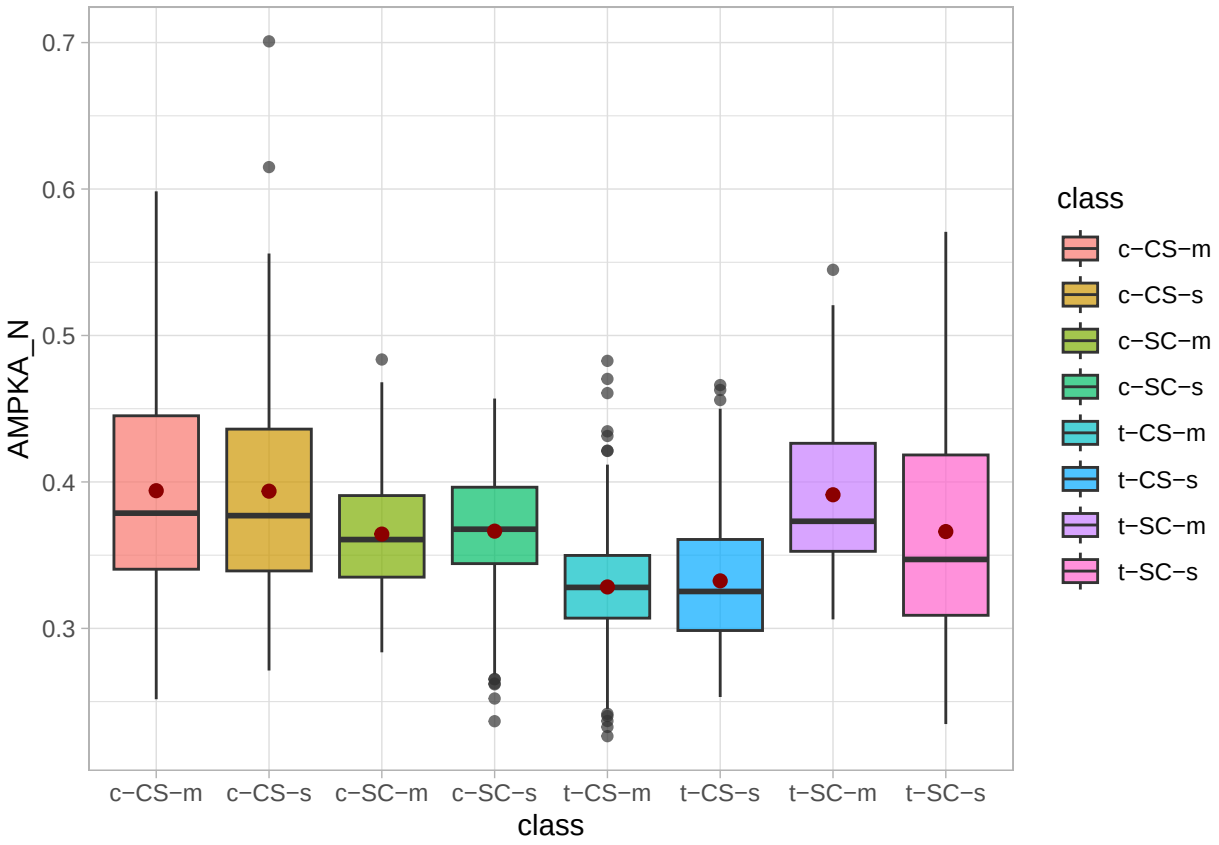
P38_N


pMTOR_N
0.9
0.6
0.3
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

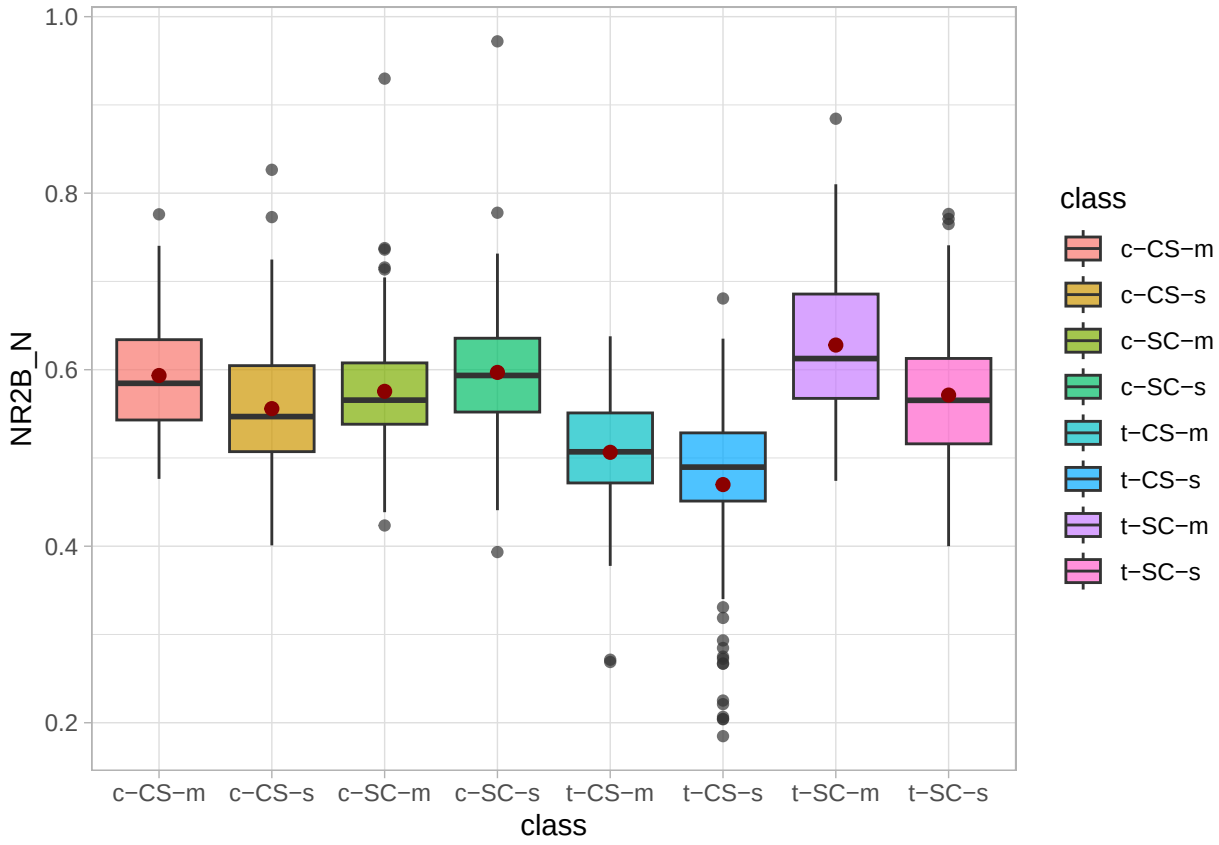
pMTOR_N


DSCR1_N
0.75
0.50
0.25
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

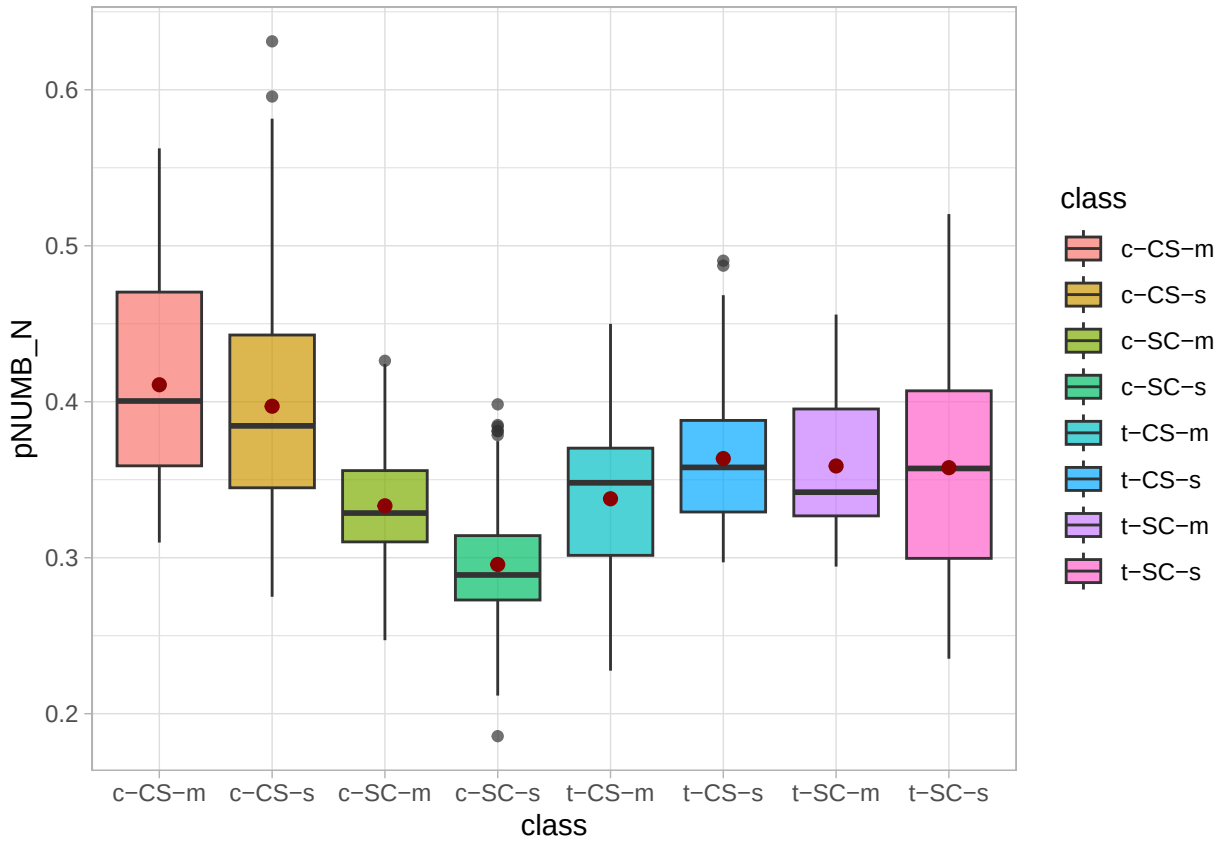
DSCR1_N


0.7
0.6
0.5
0.4
0.3
AMPKA_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

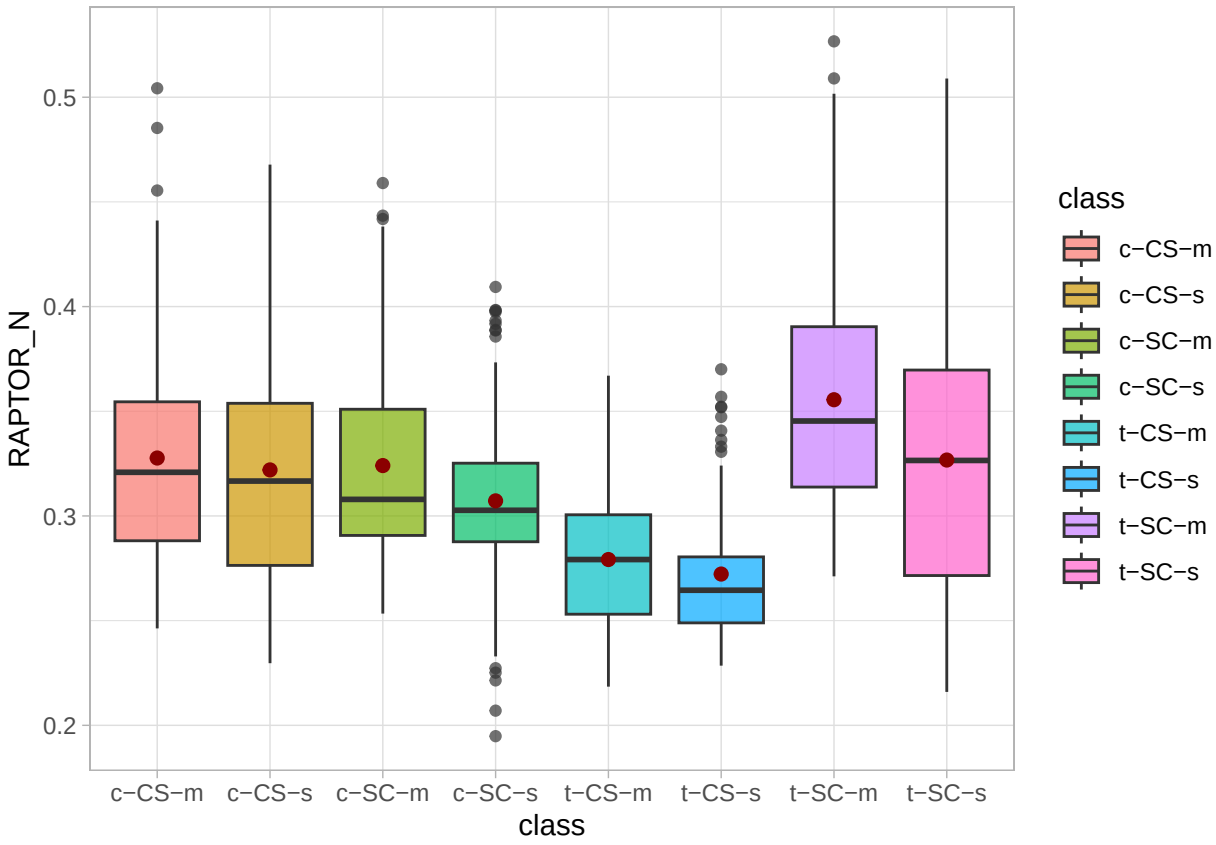
AMPKA_N


1.0
0.8
0.6
0.4
0.2
NR2B_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

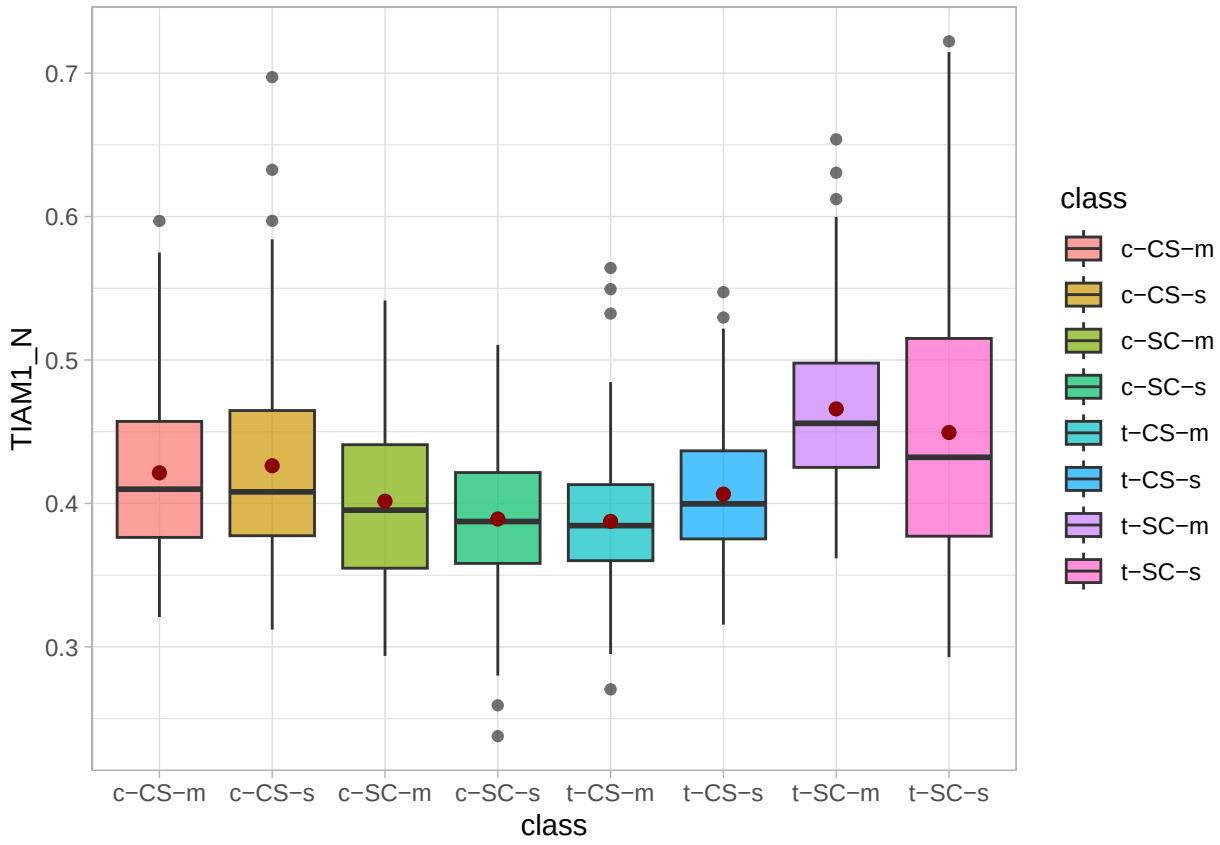
NR2B_N


0.6
0.5
0.4
0.3
0.2
pNUMB_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

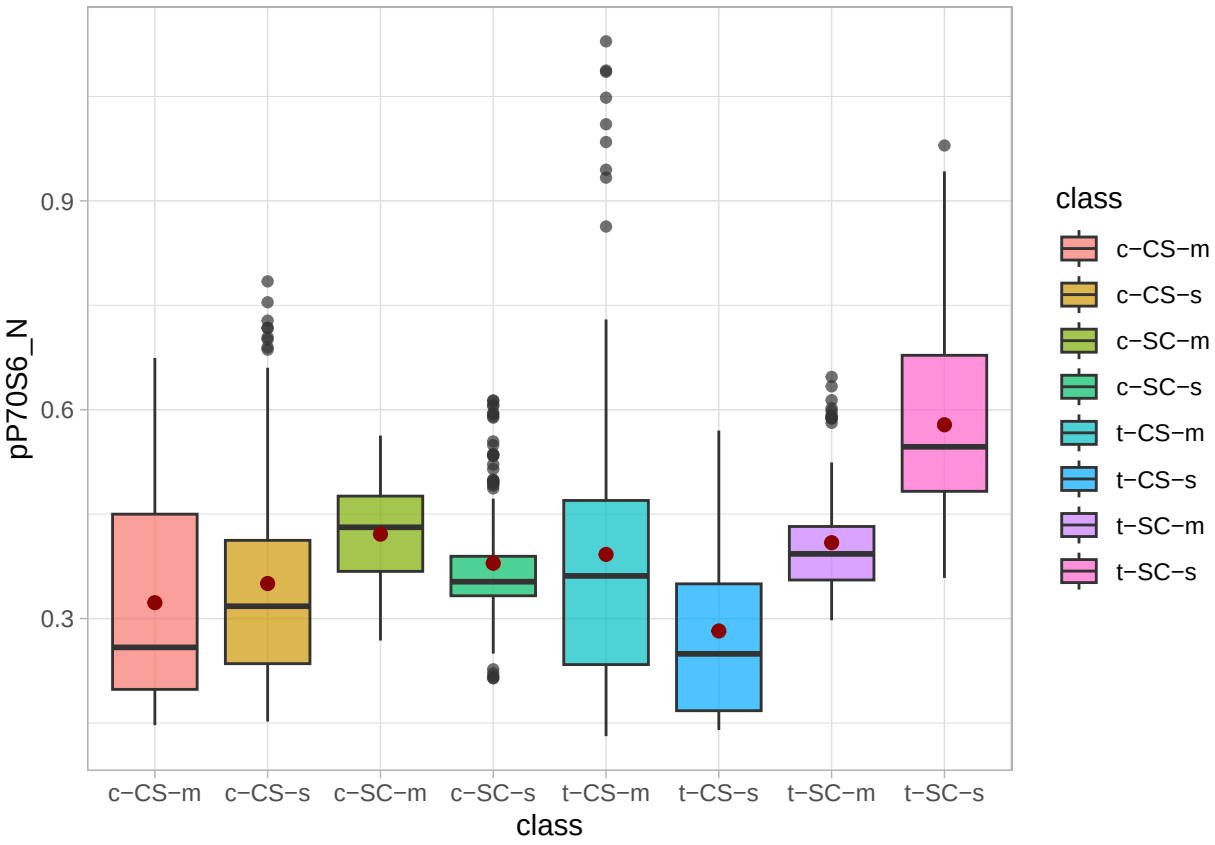
pNUMB_N


RAPTOR_N
0.5
0.4
0.3
0.2
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

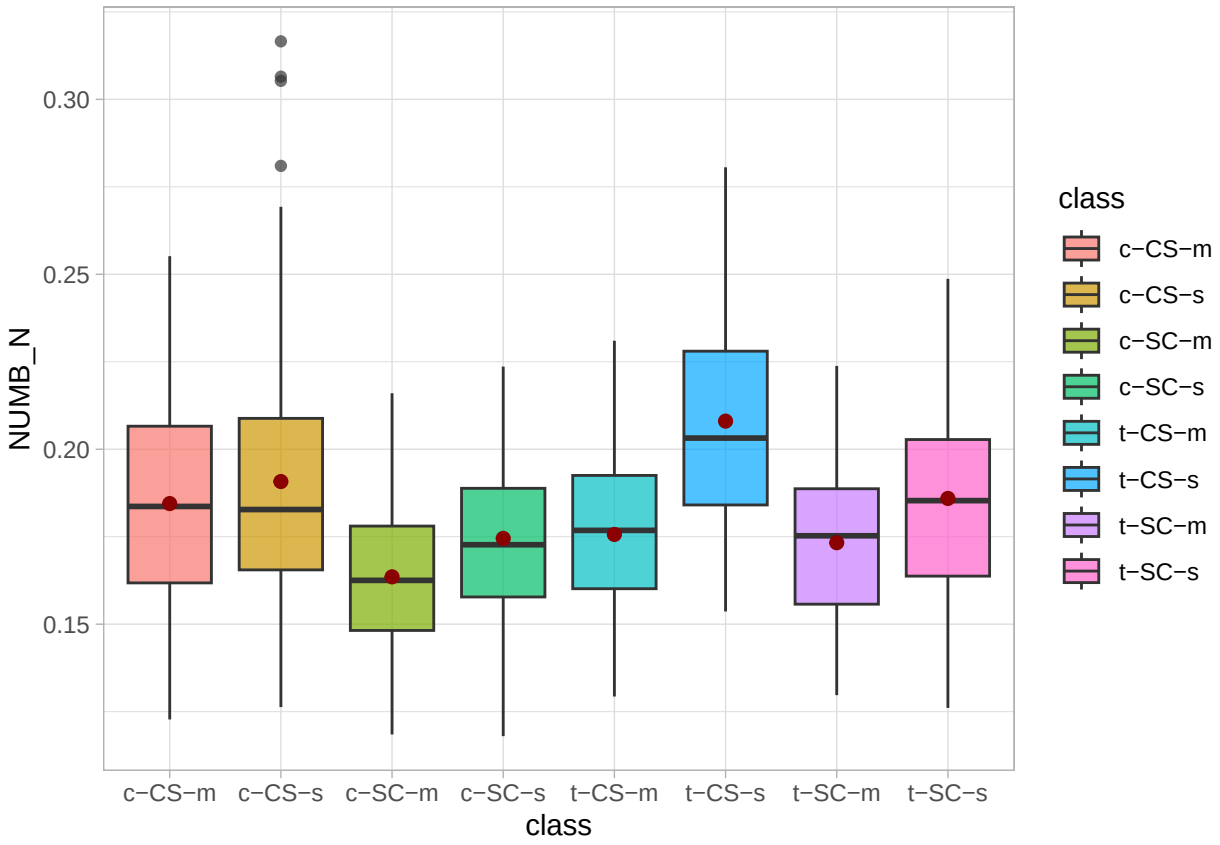
RAPTOR_N


0.7
0.6
0.5
0.4
0.3
TIAM1_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

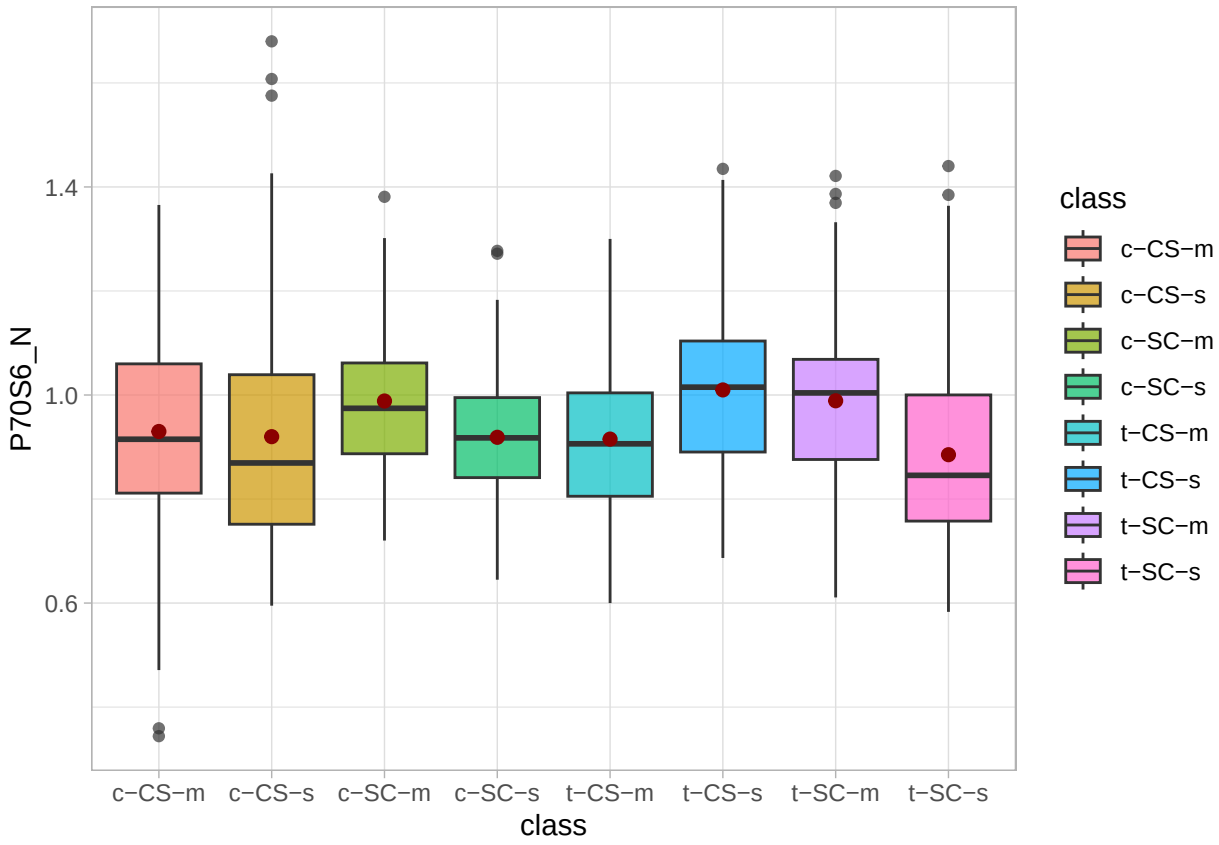
TIAM1_N


pP70S6_N
0.9
0.6
0.3
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

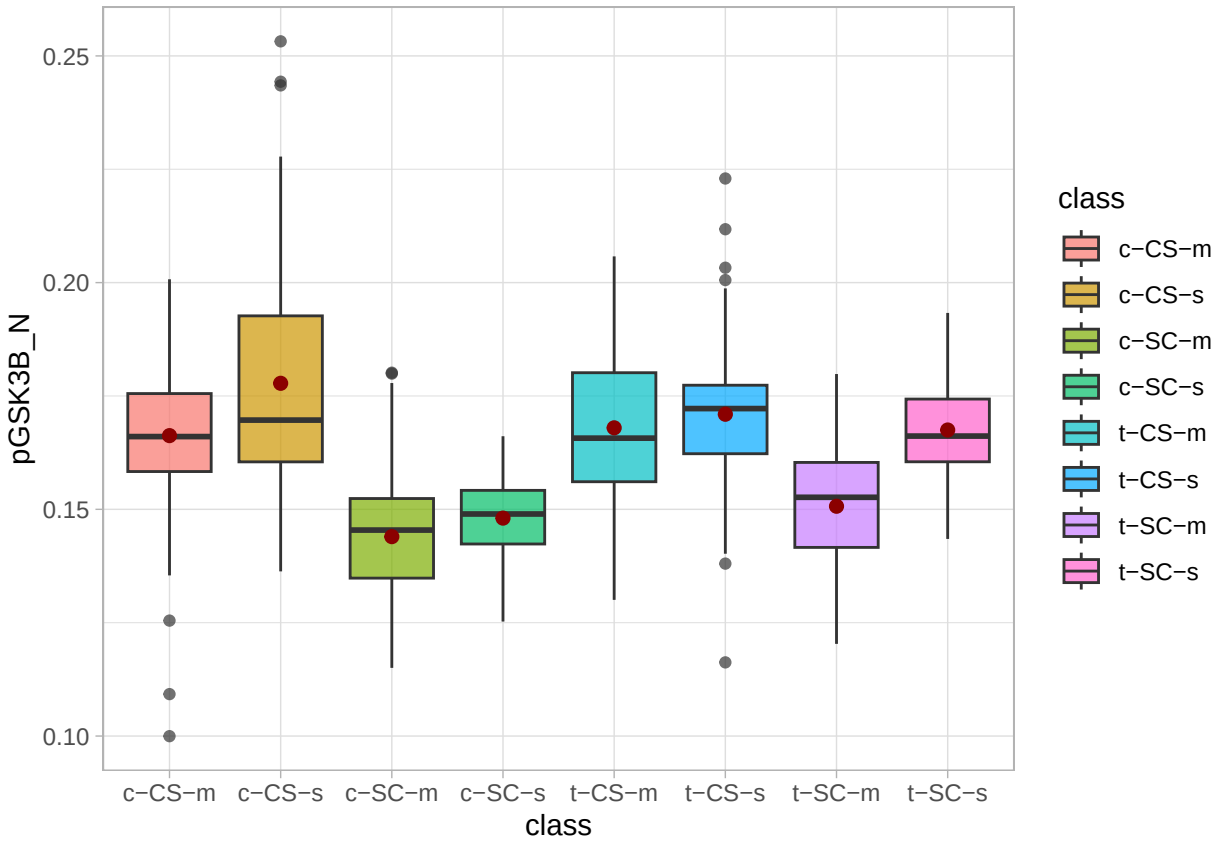
pP70S6_N


0.30
0.25
0.20
0.15
NUMB_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

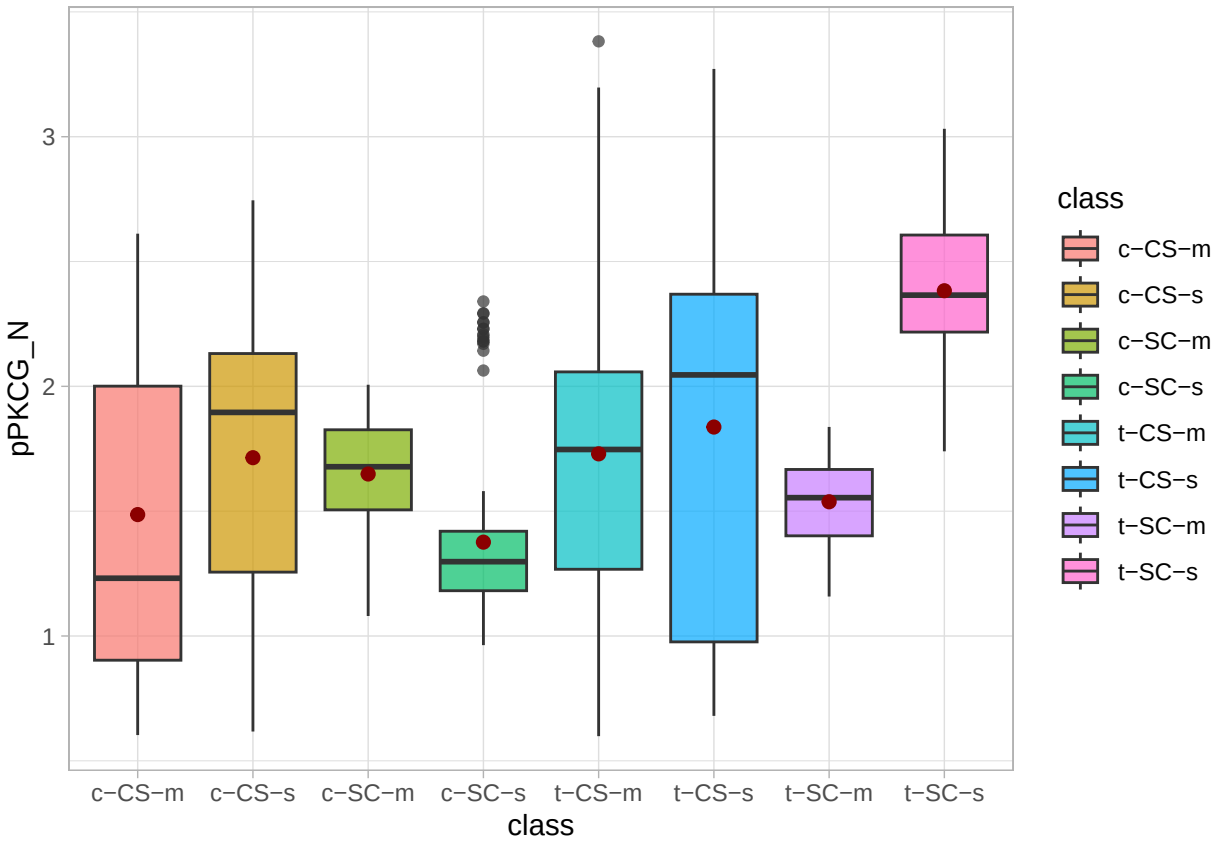
NUMB_N


P70S6_N
1.4
1.0
0.6
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

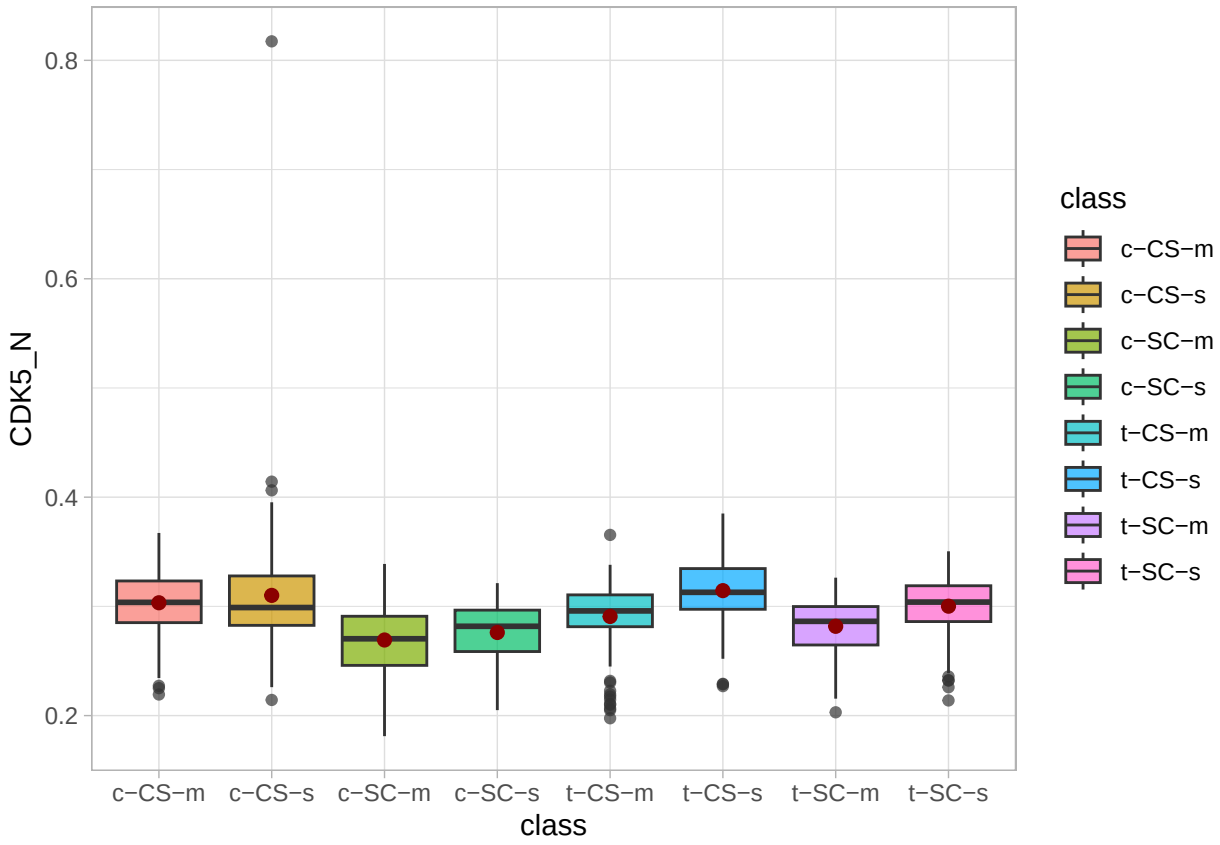
P70S6_N


0.25
0.20
0.15
0.10
pGSK3B_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

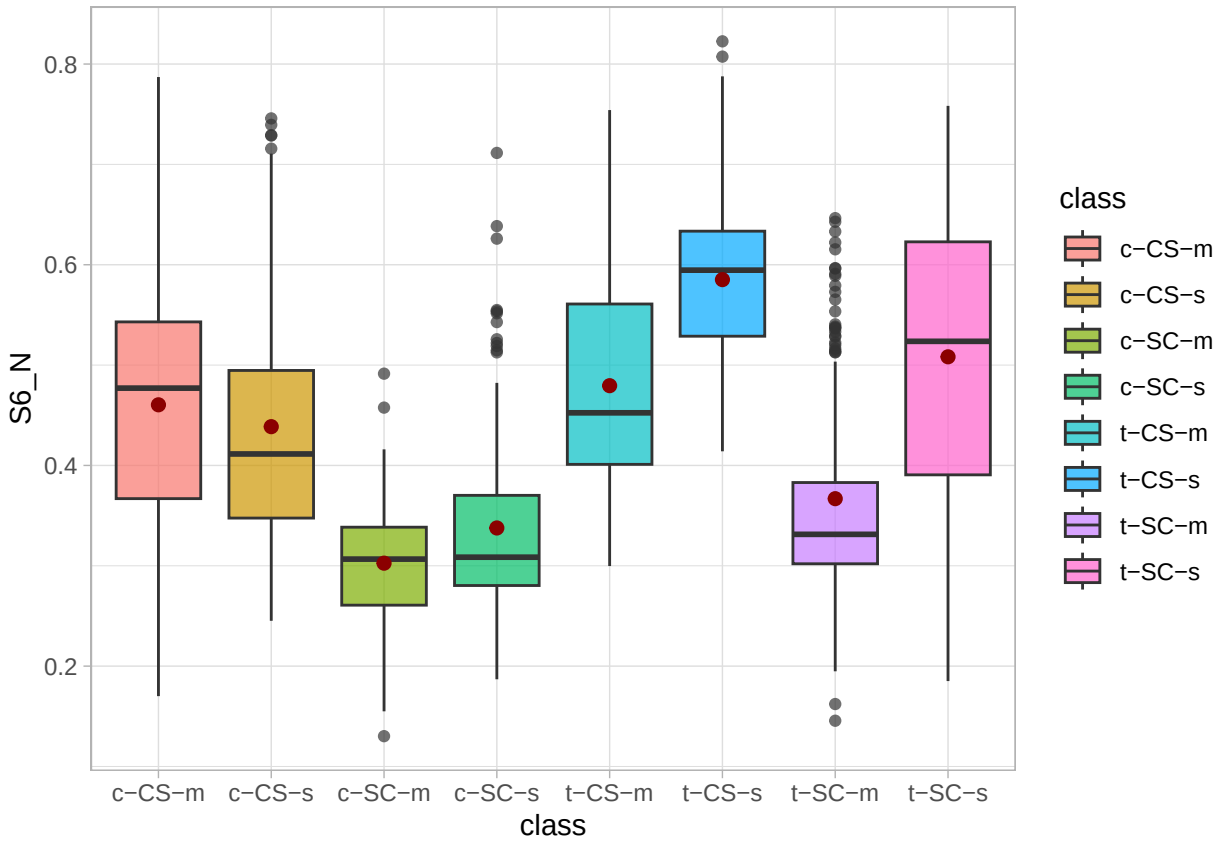
pGSK3B_N


pPKCG_N
3
2
1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

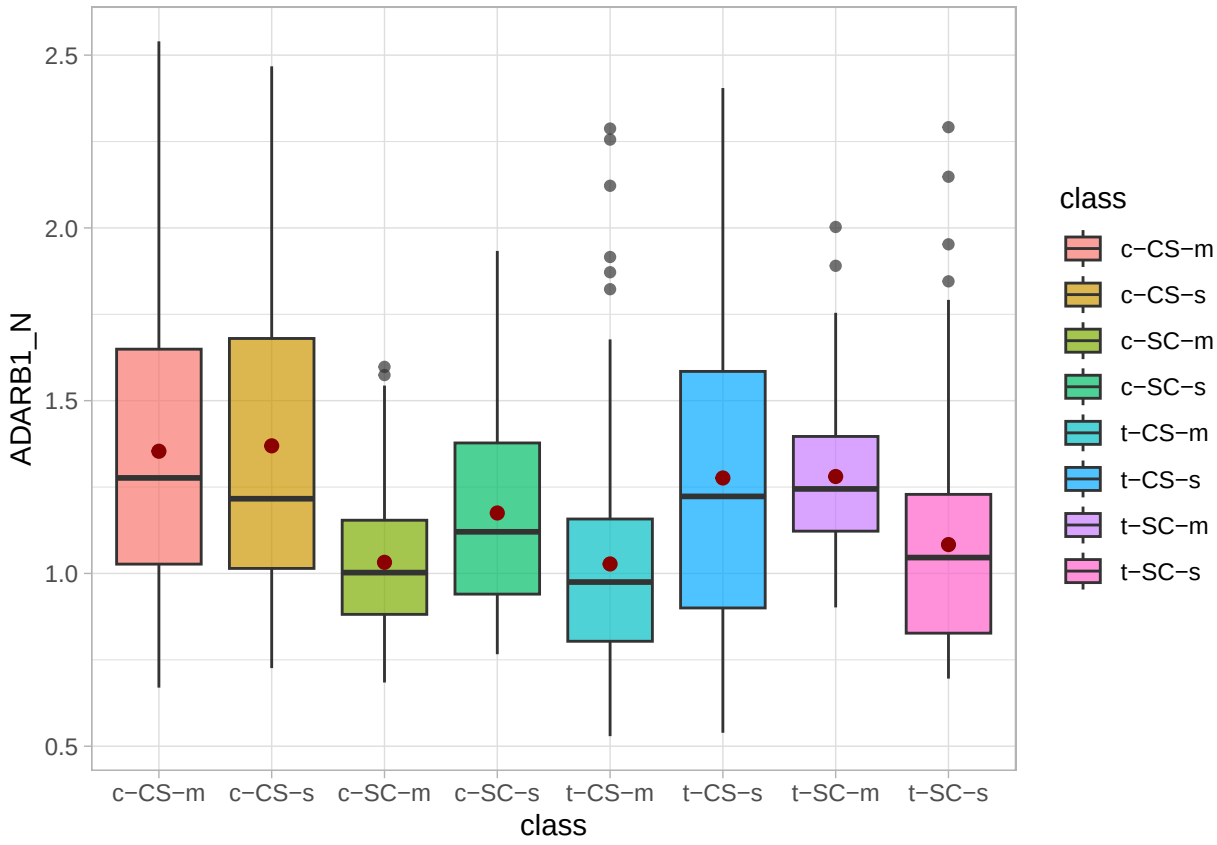
pPKCG_N


0.8
0.6
0.4
0.2
CDK5_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

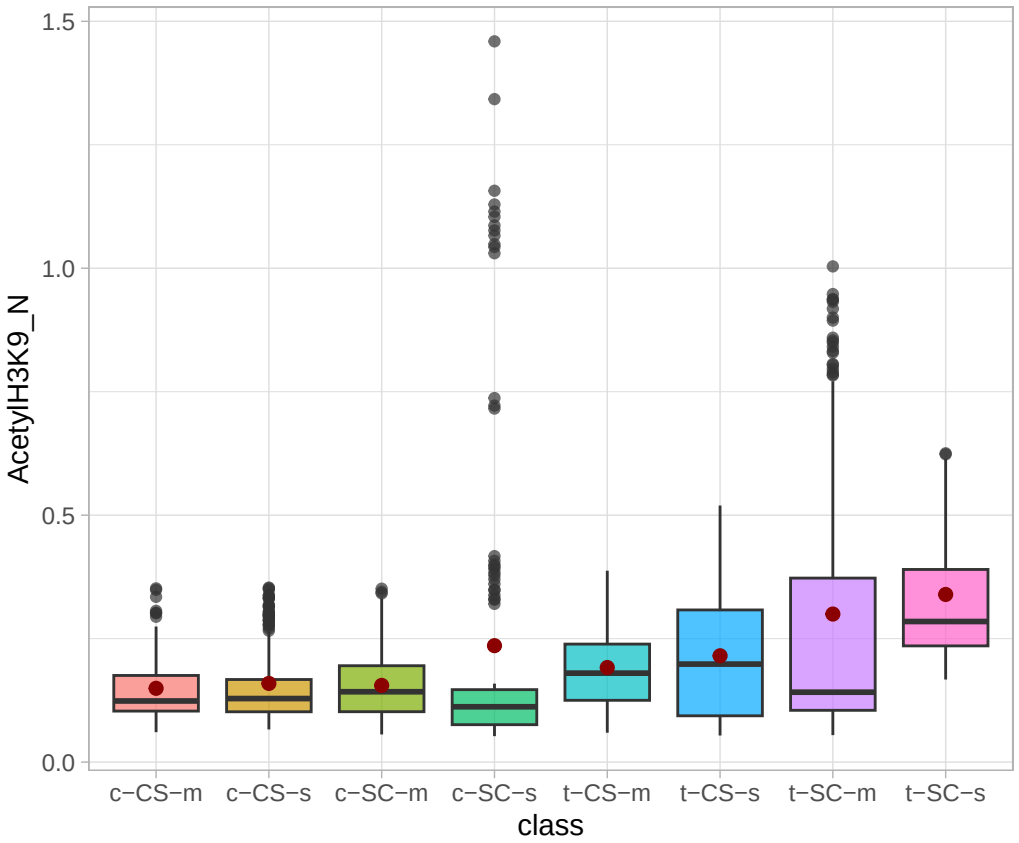
CDK5_N


0.8
0.6
0.4
0.2
S6_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

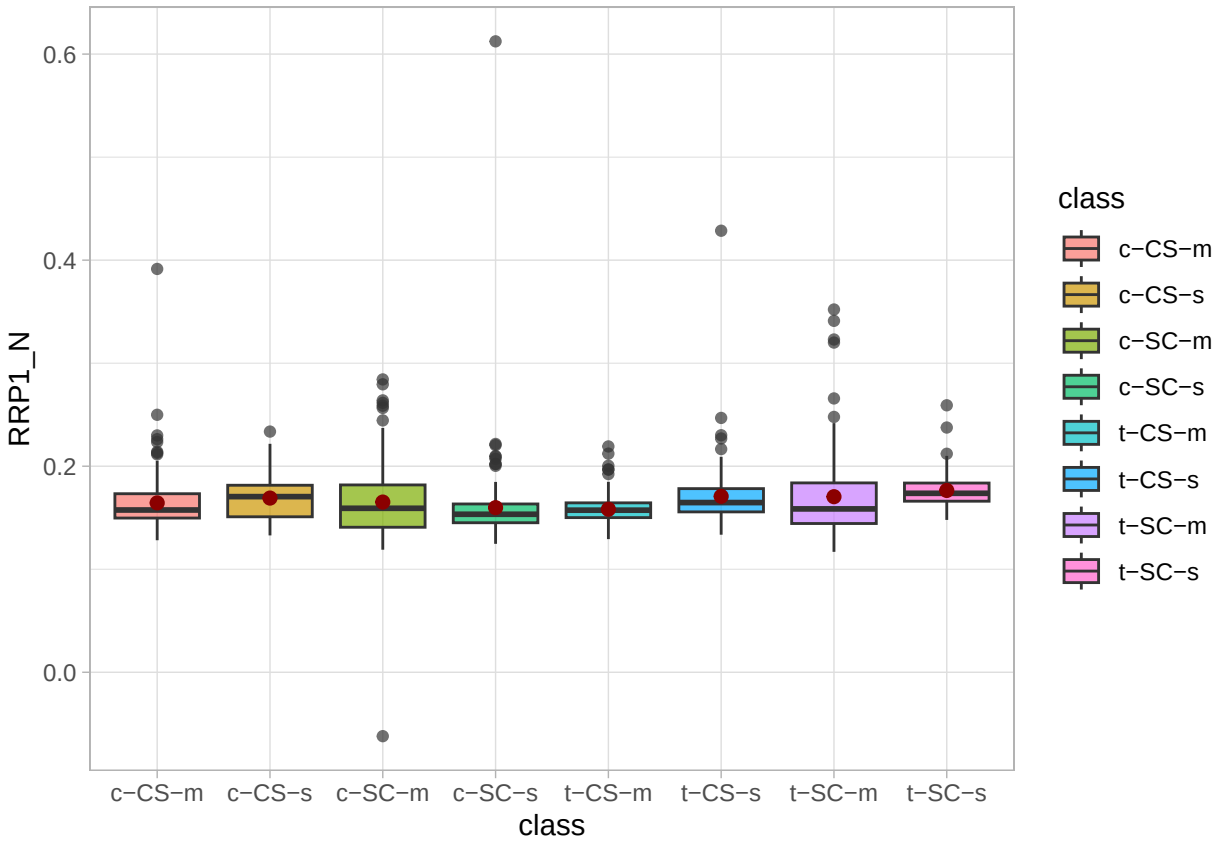
S6_N


2.5
2.0
1.5
1.0
0.5
ADARB1_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

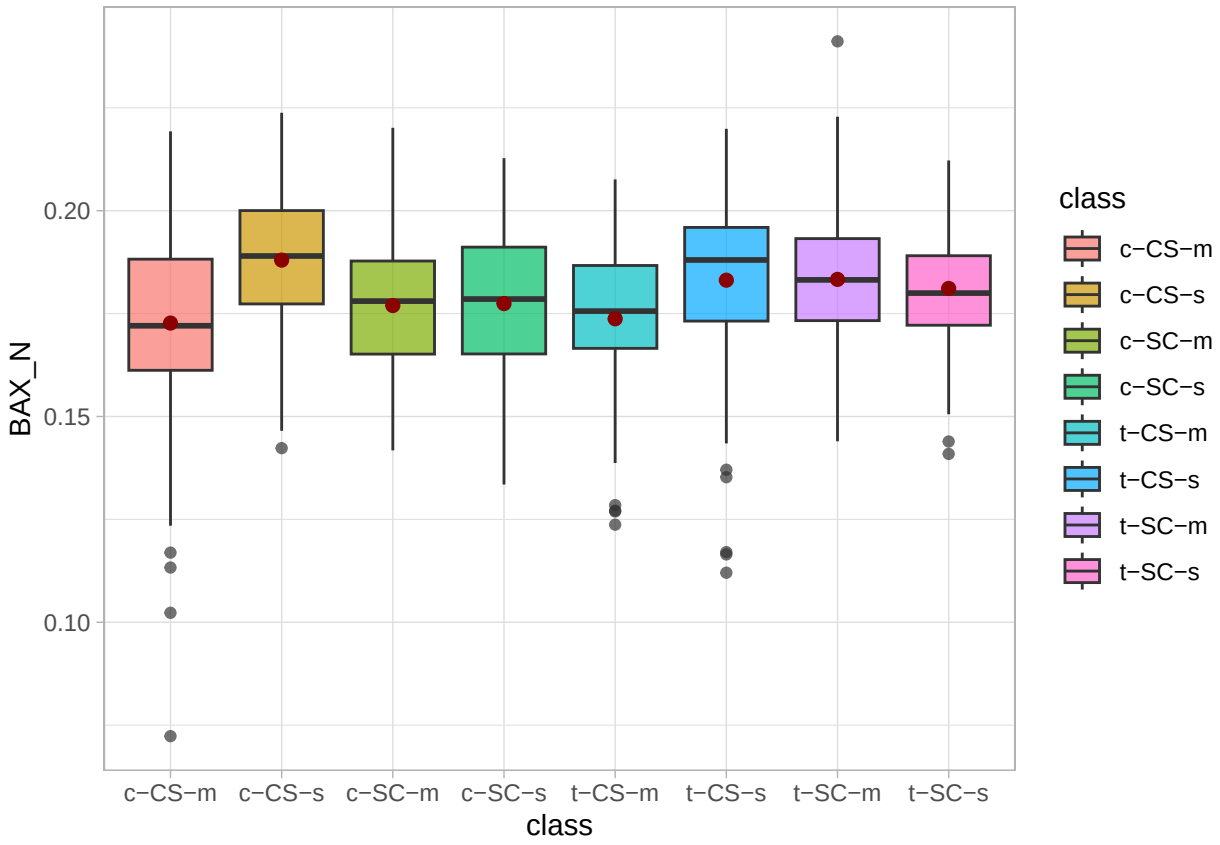
ADARB1_N



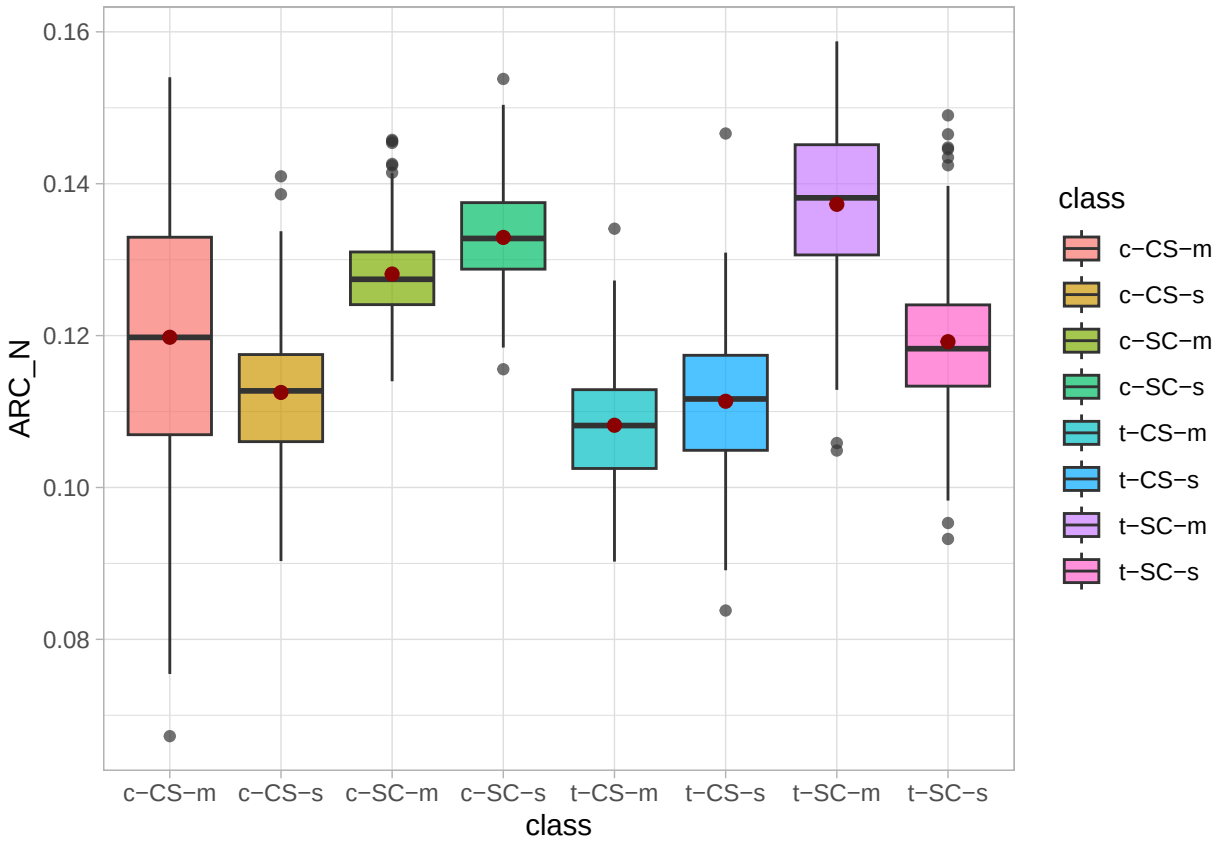
AcetylH3K9_N


0.6
0.4
0.2
0.0
RRP1_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

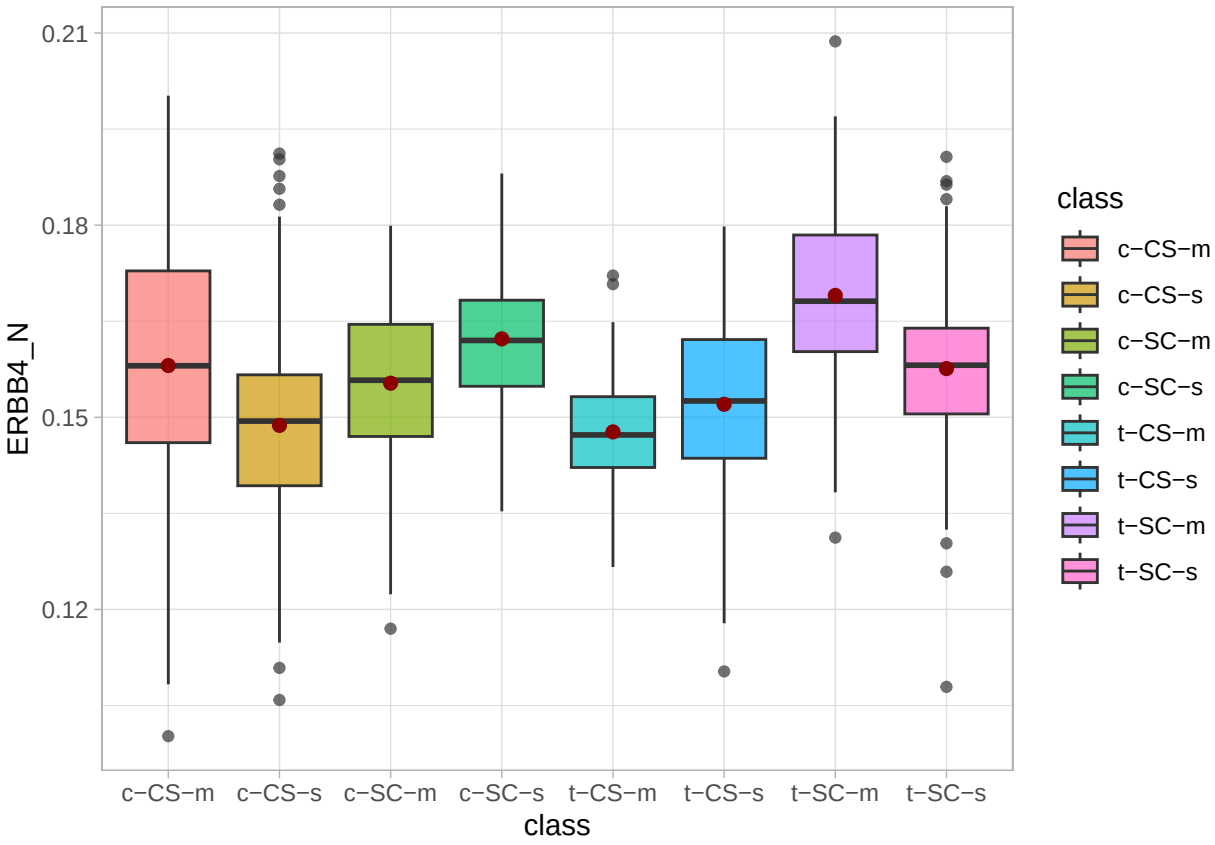
RRP1_N


BAX_N
0.20
0.15
0.10
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

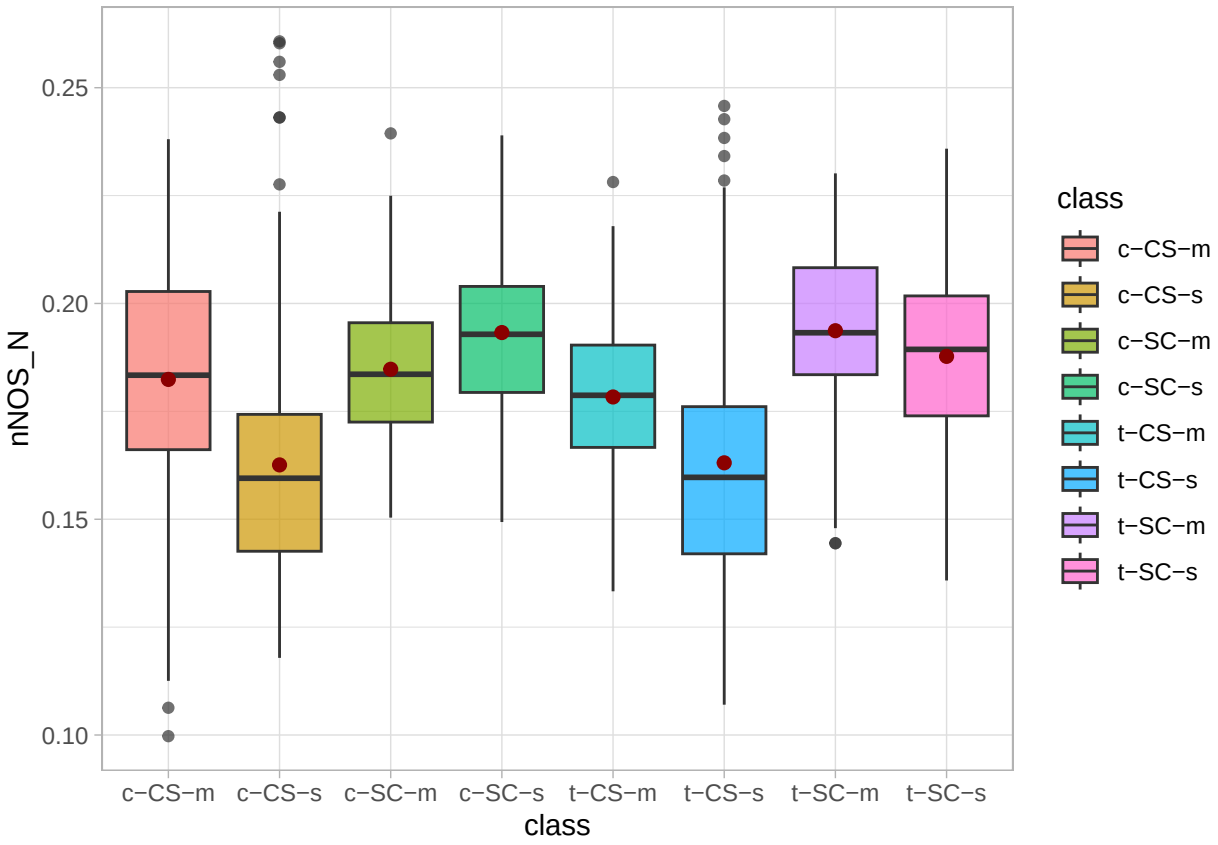
BAX_N


0.16
0.14
0.12
0.10
0.08
ARC_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

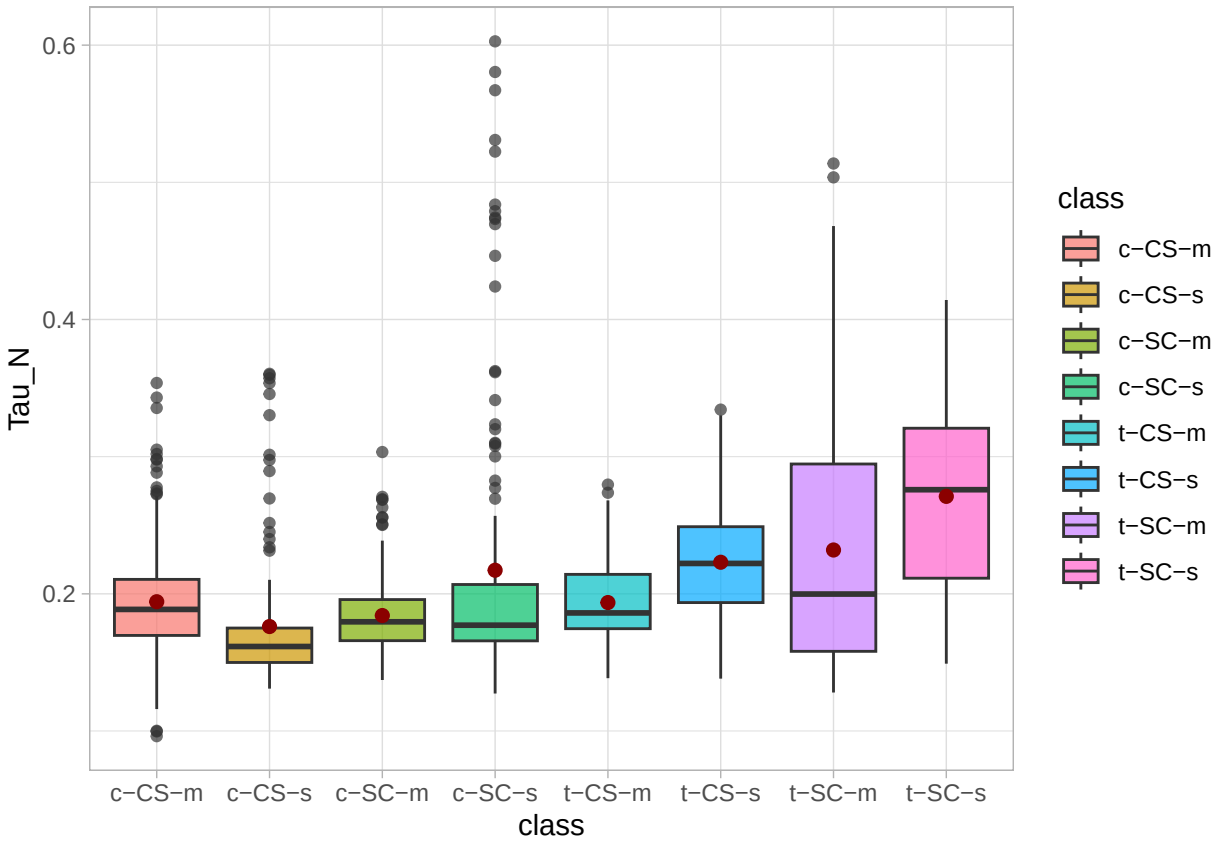
ARC_N


0.21
0.18
0.15
0.12
ERBB4_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

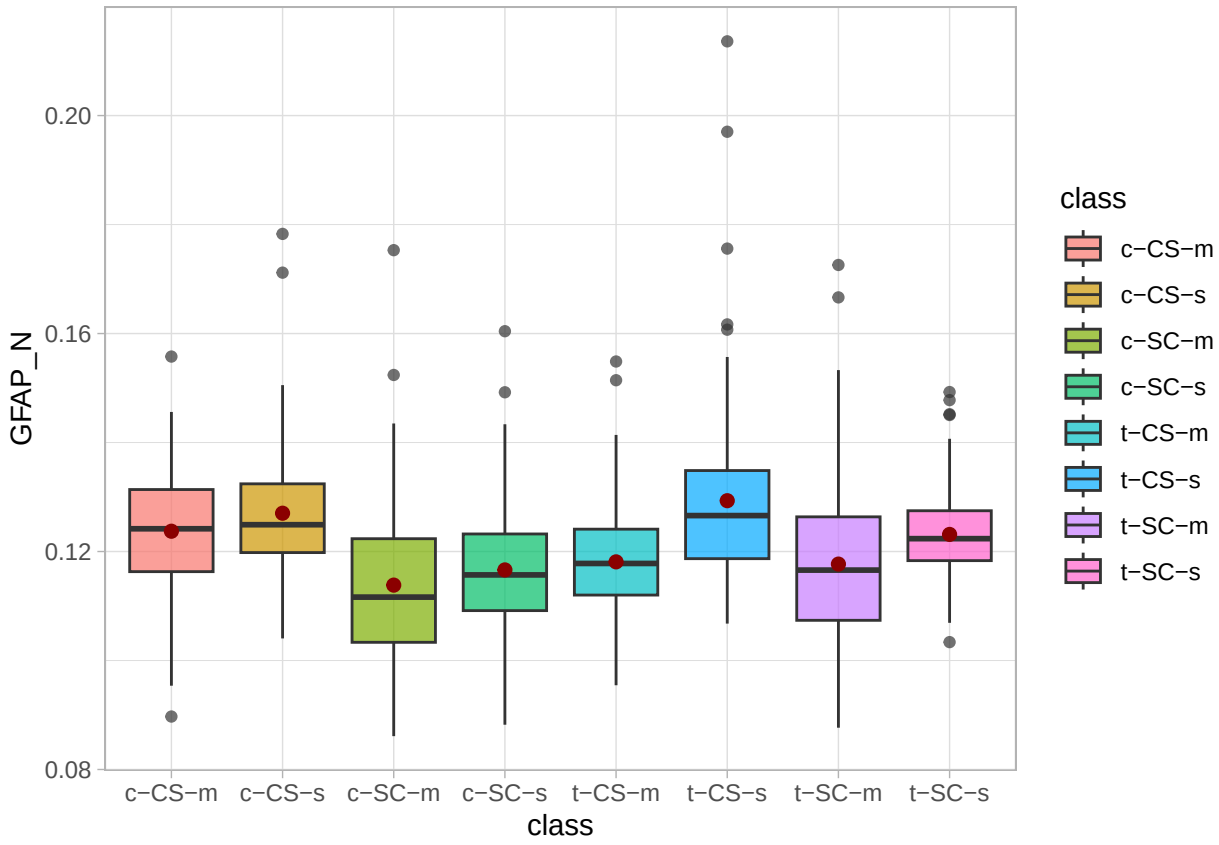
ERBB4_N


0.25
0.20
0.15
0.10
nNOS_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

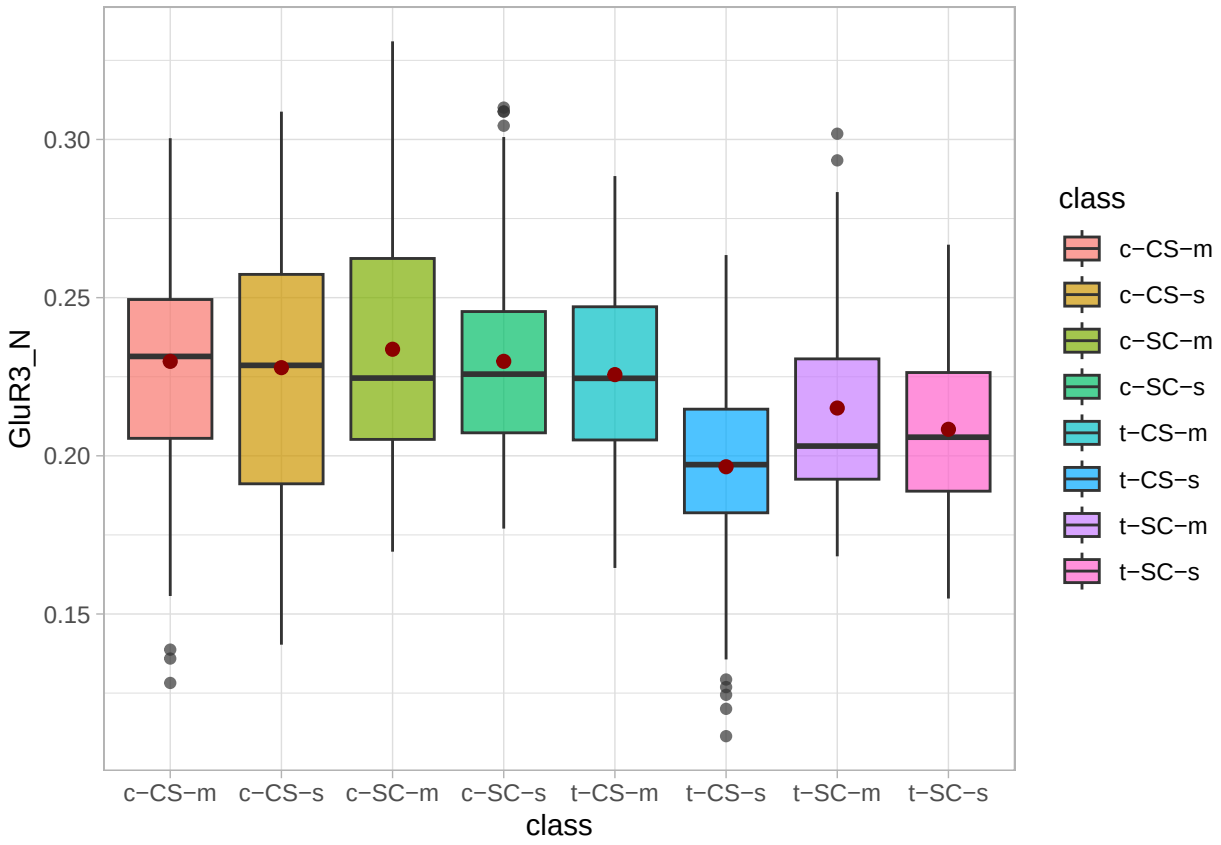
nNOS_N


0.6
0.4
0.2
Tau_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

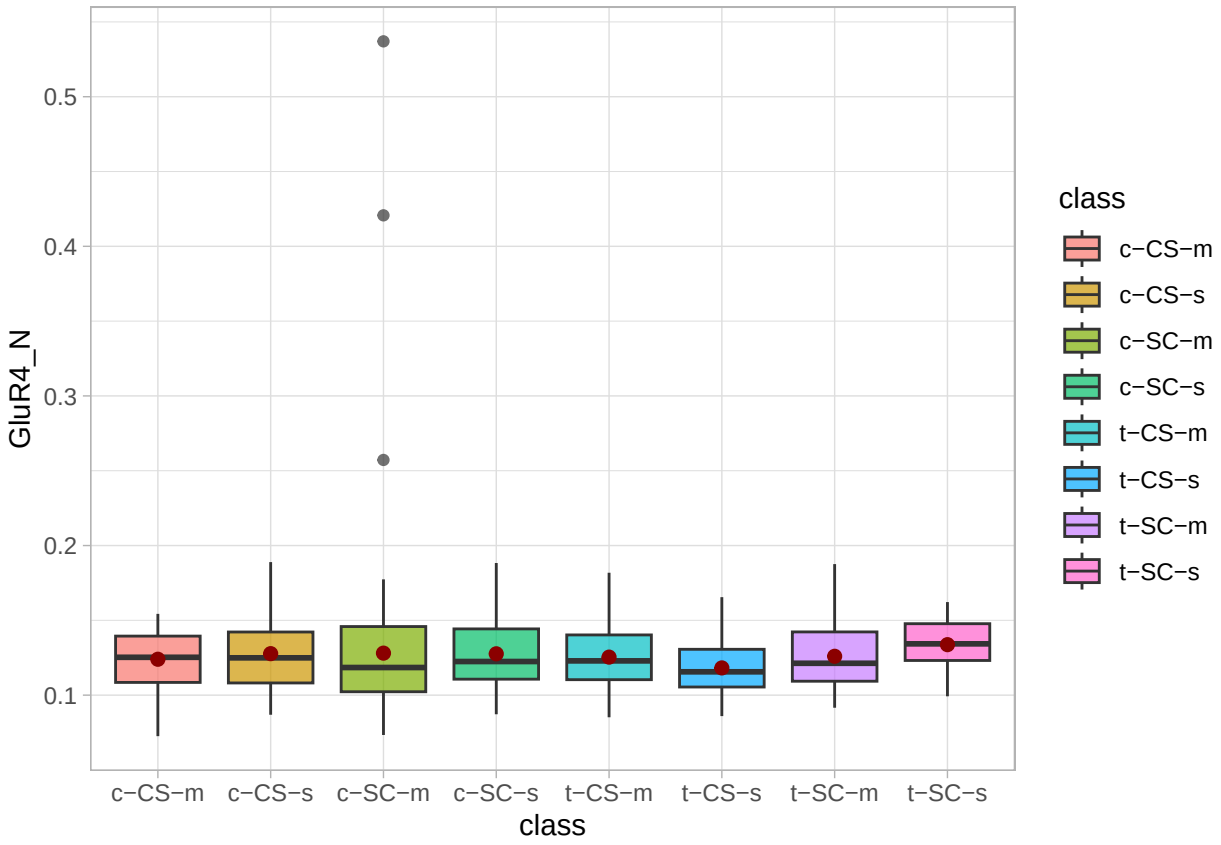
Tau_N


GFAP_N
0.20
0.16
0.12
0.08
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

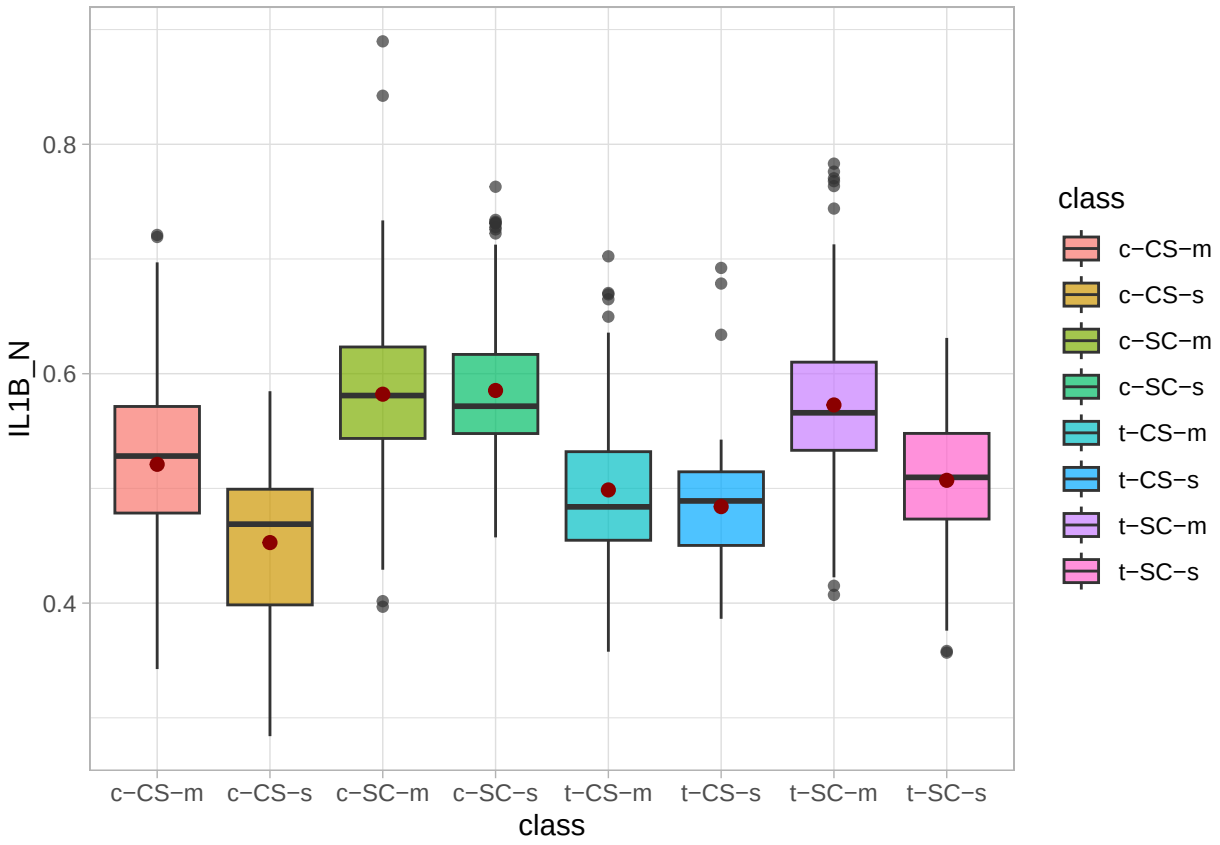
GFAP_N


0.30
0.25
0.20
0.15
GluR3_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

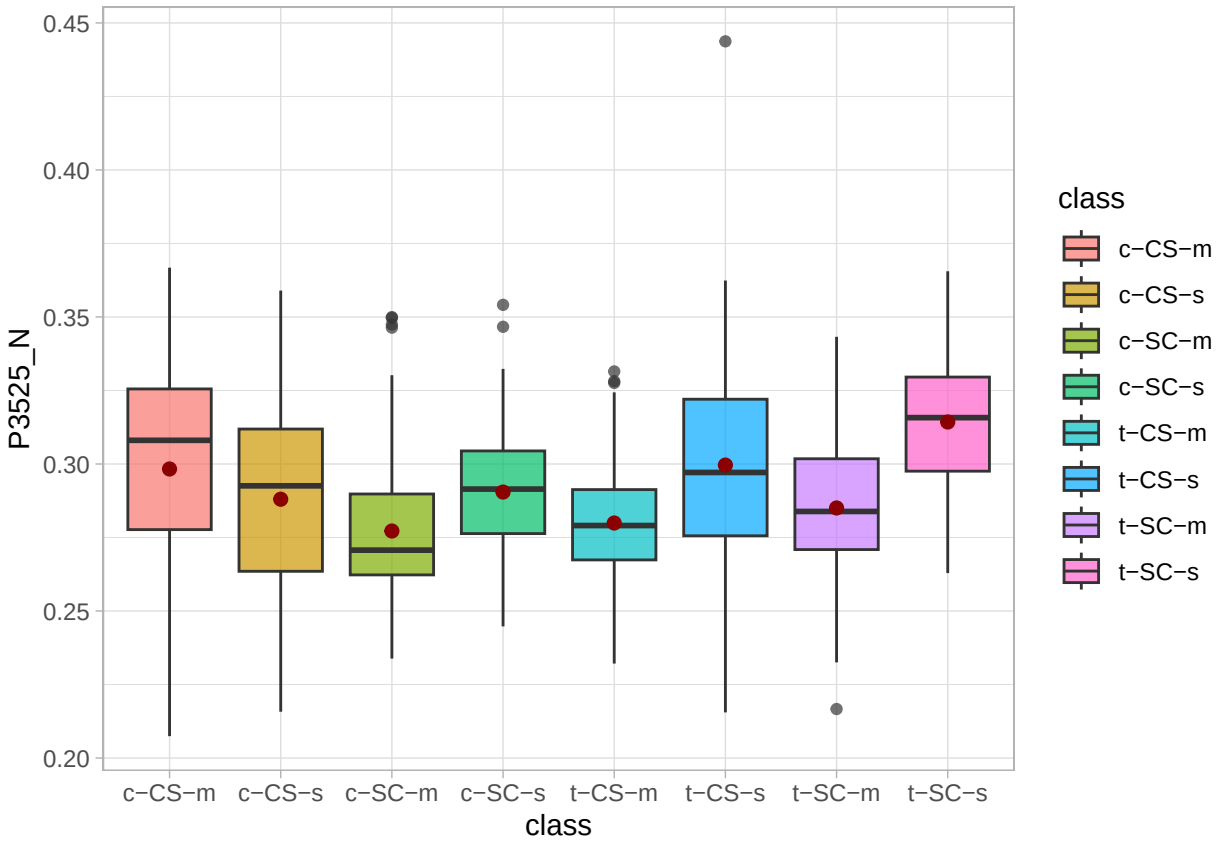
GluR3_N


GluR4_N
0.5
0.4
0.3
0.2
0.1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

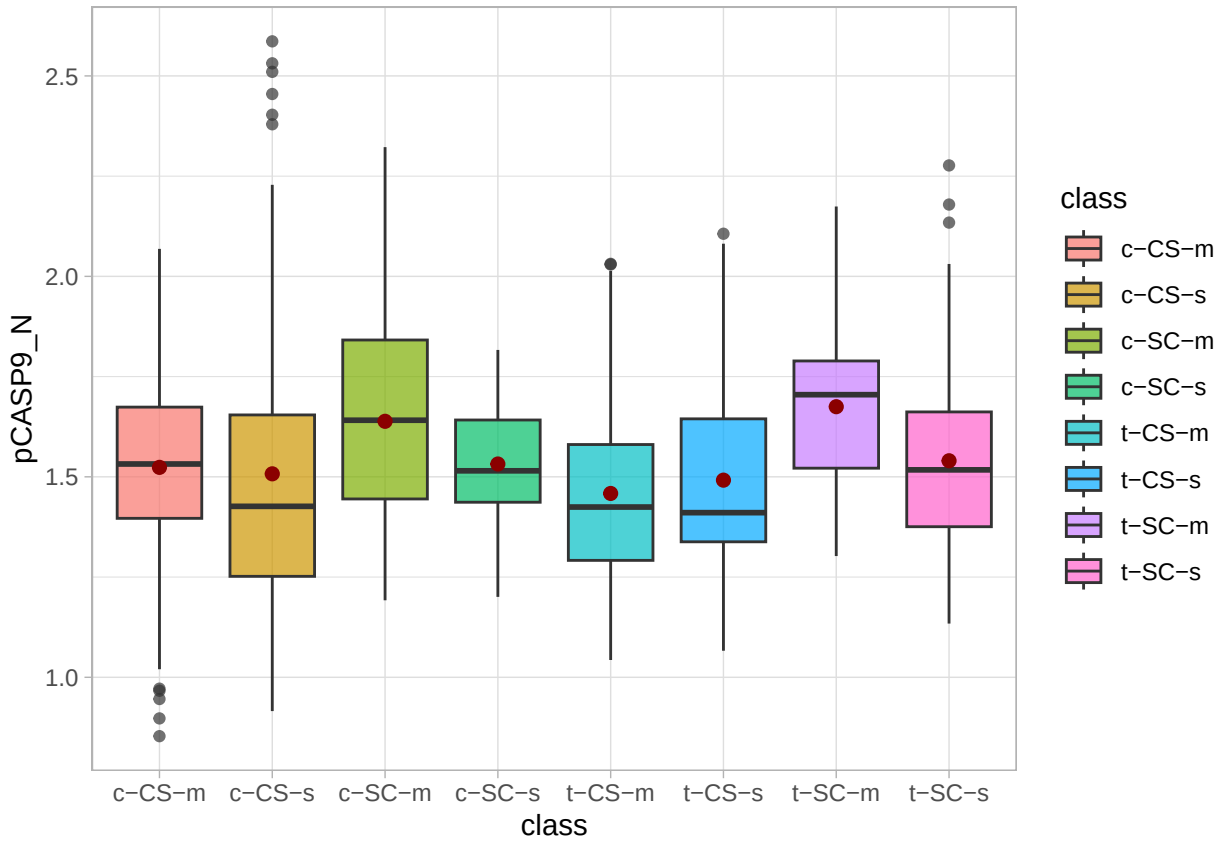
GluR4_N


IL1B_N
0.8
0.6
0.4
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

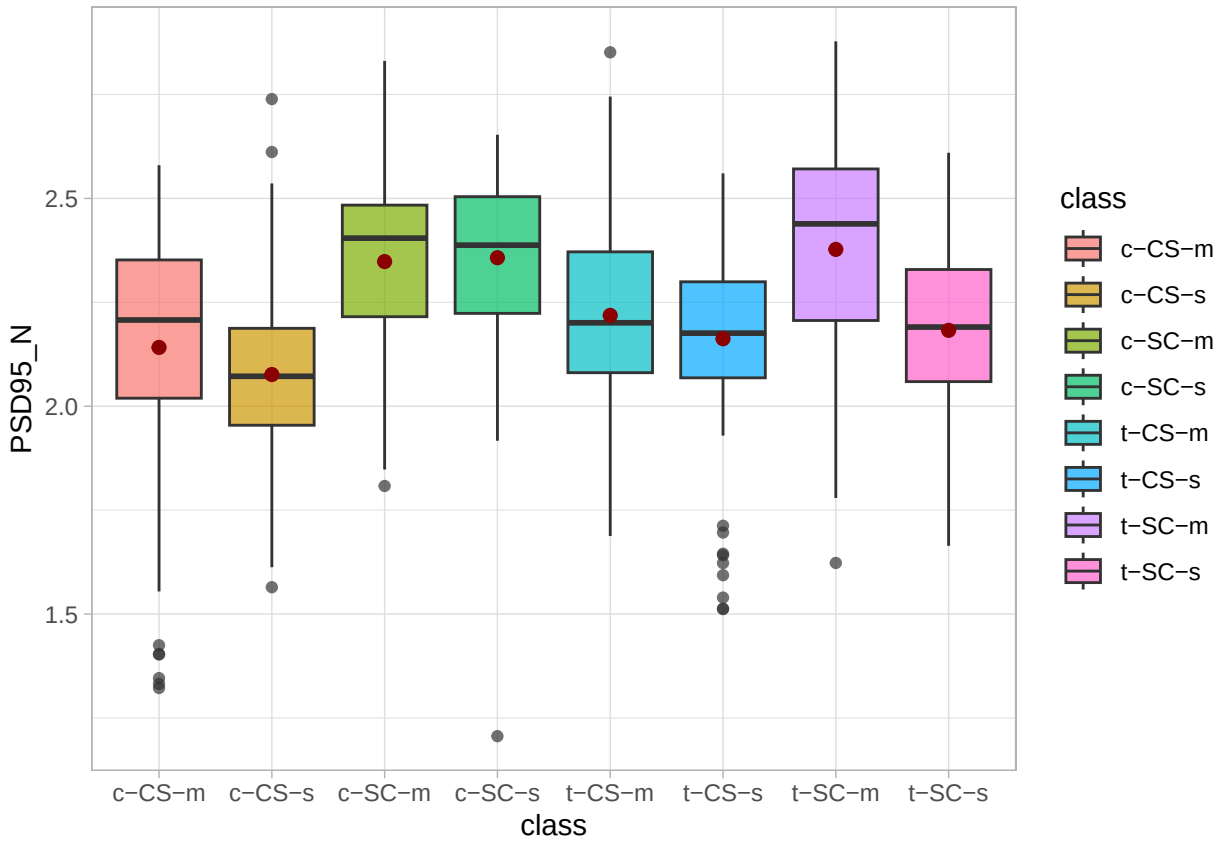
IL1B_N


0.45
0.40
0.35
0.30
0.25
0.20
P3525_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

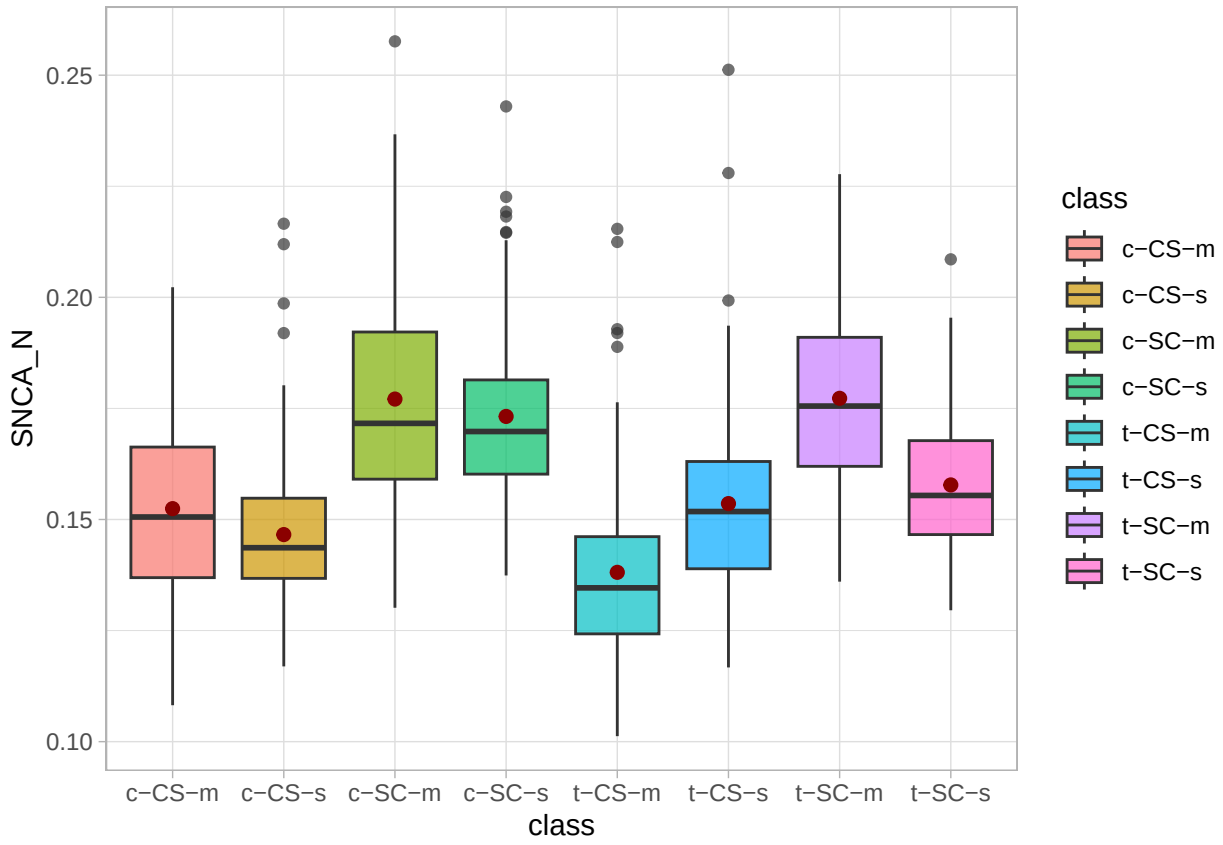
P3525_N


pCASP9_N
2.5
2.0
1.5
1.0
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

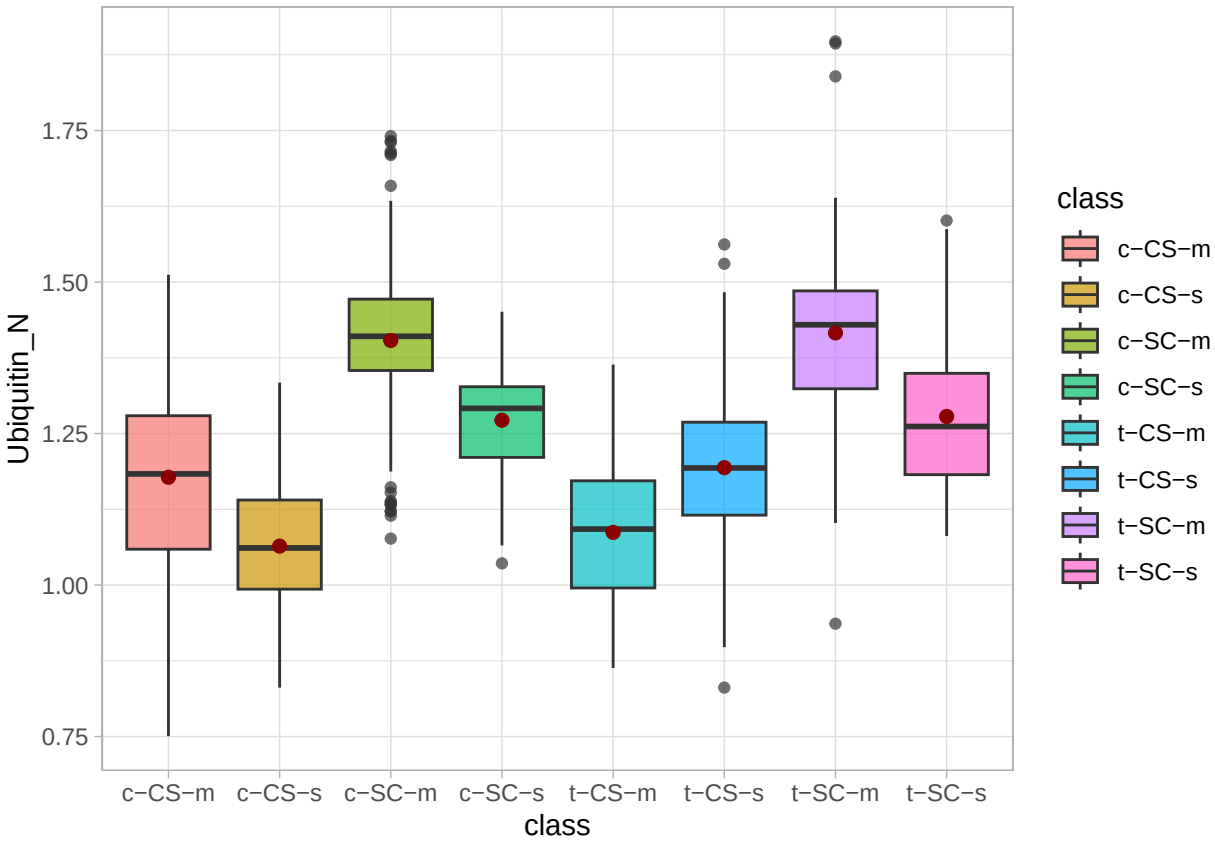
pCASP9_N


PSD95_N
2.5
2.0
1.5
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

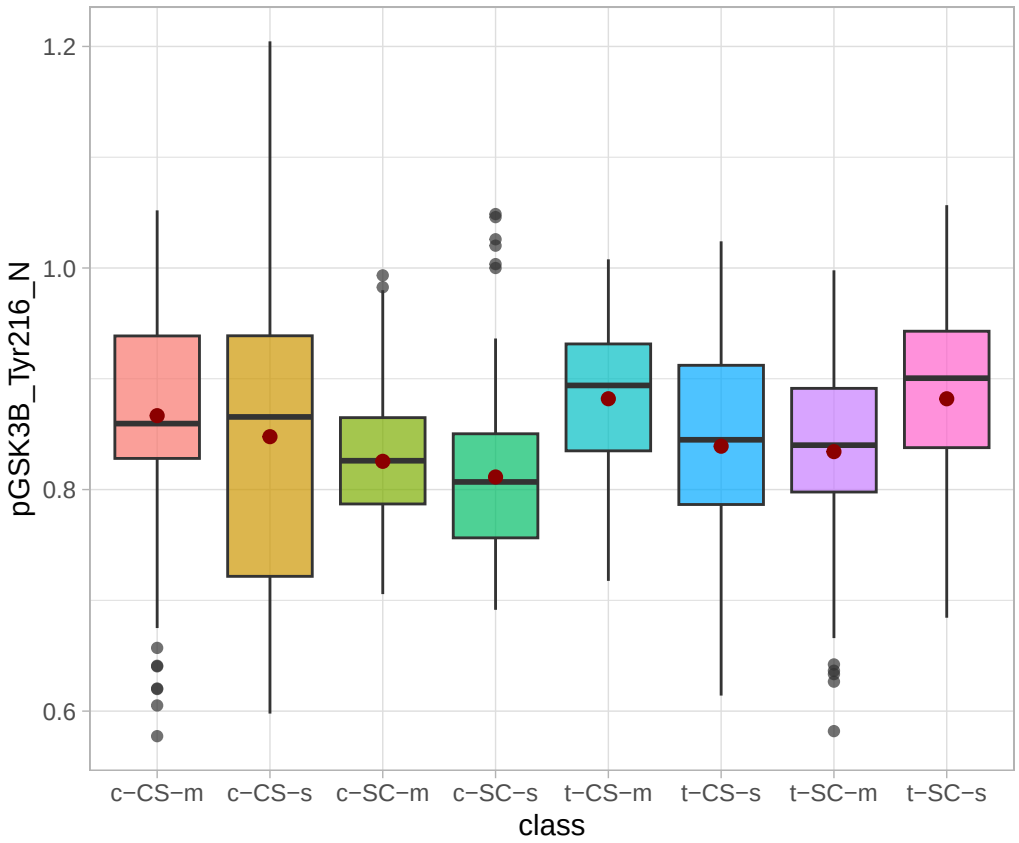
PSD95_N


0.25
0.20
0.15
0.10
SNCA_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

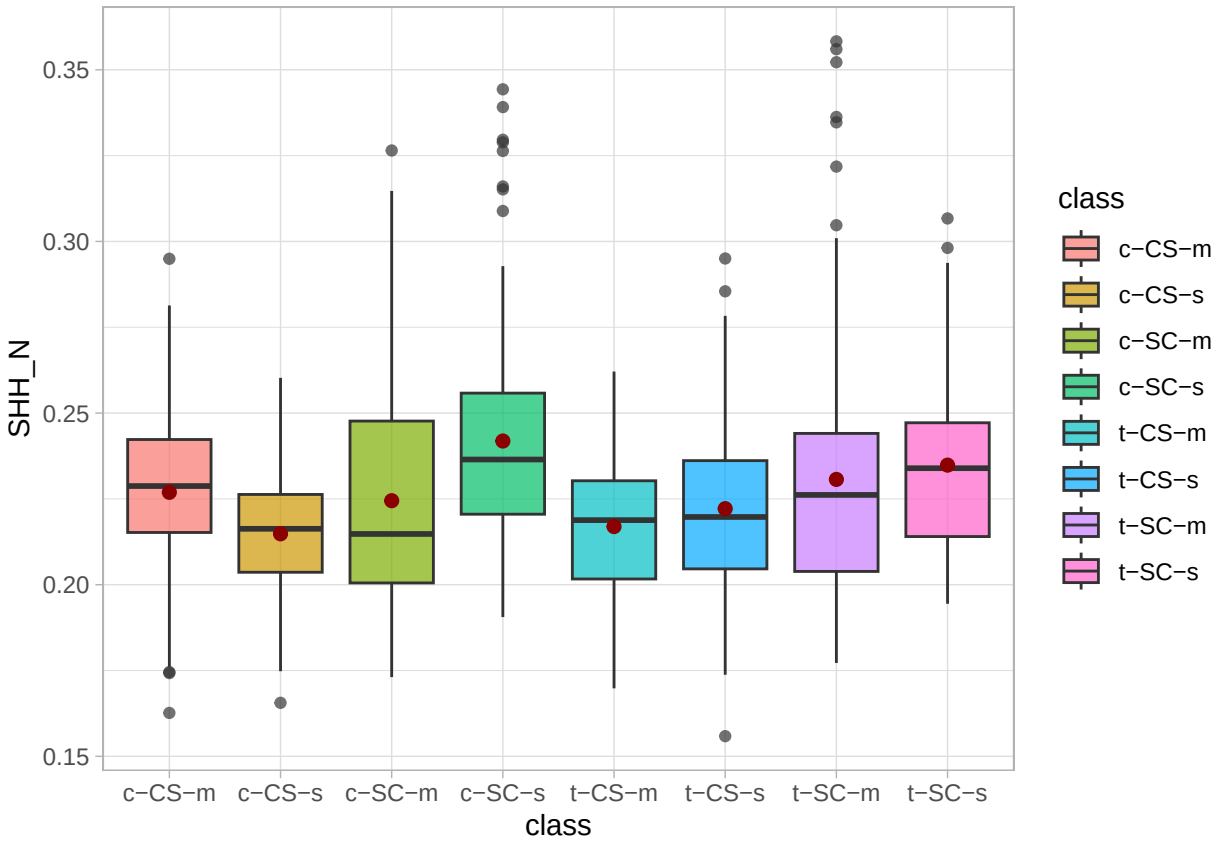
SNCA_N


1.75
1.50
1.25
1.00
0.75
Ubiquitin_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

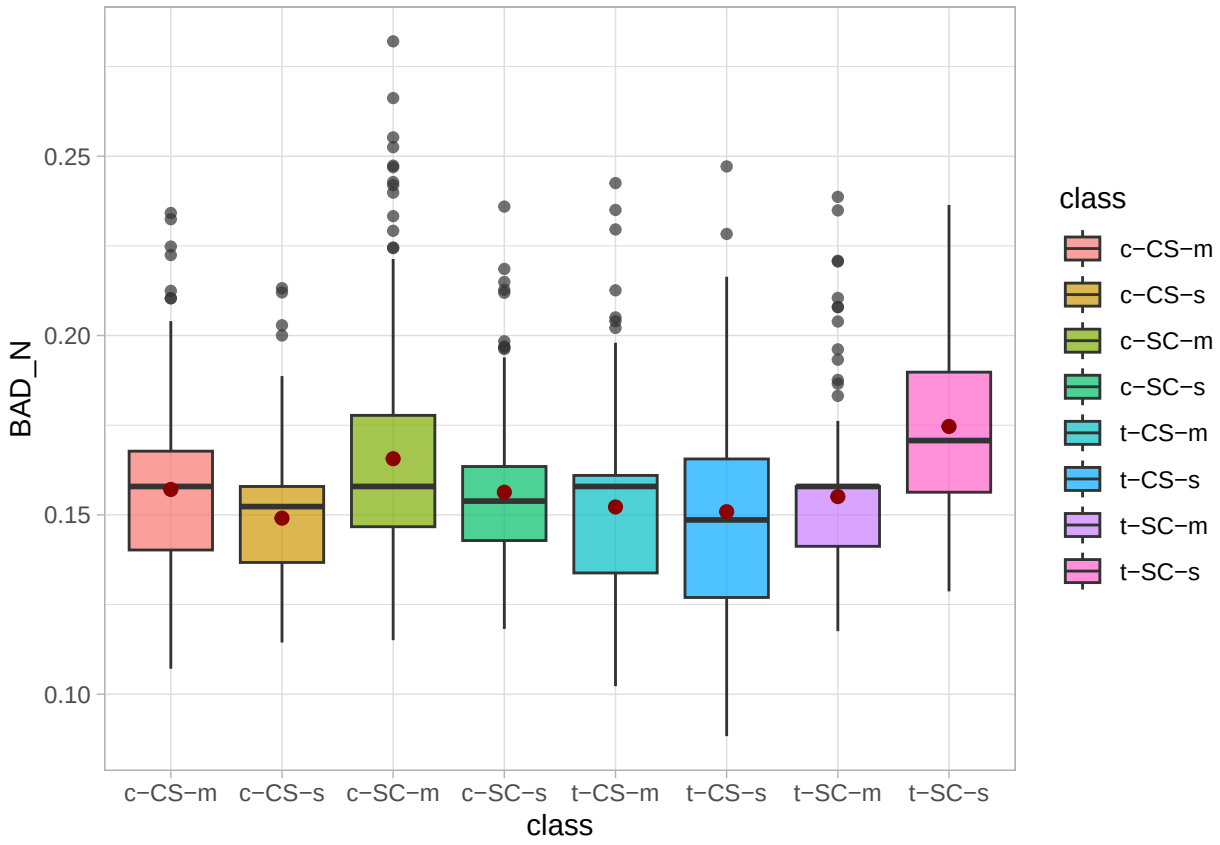
Ubiquitin_N



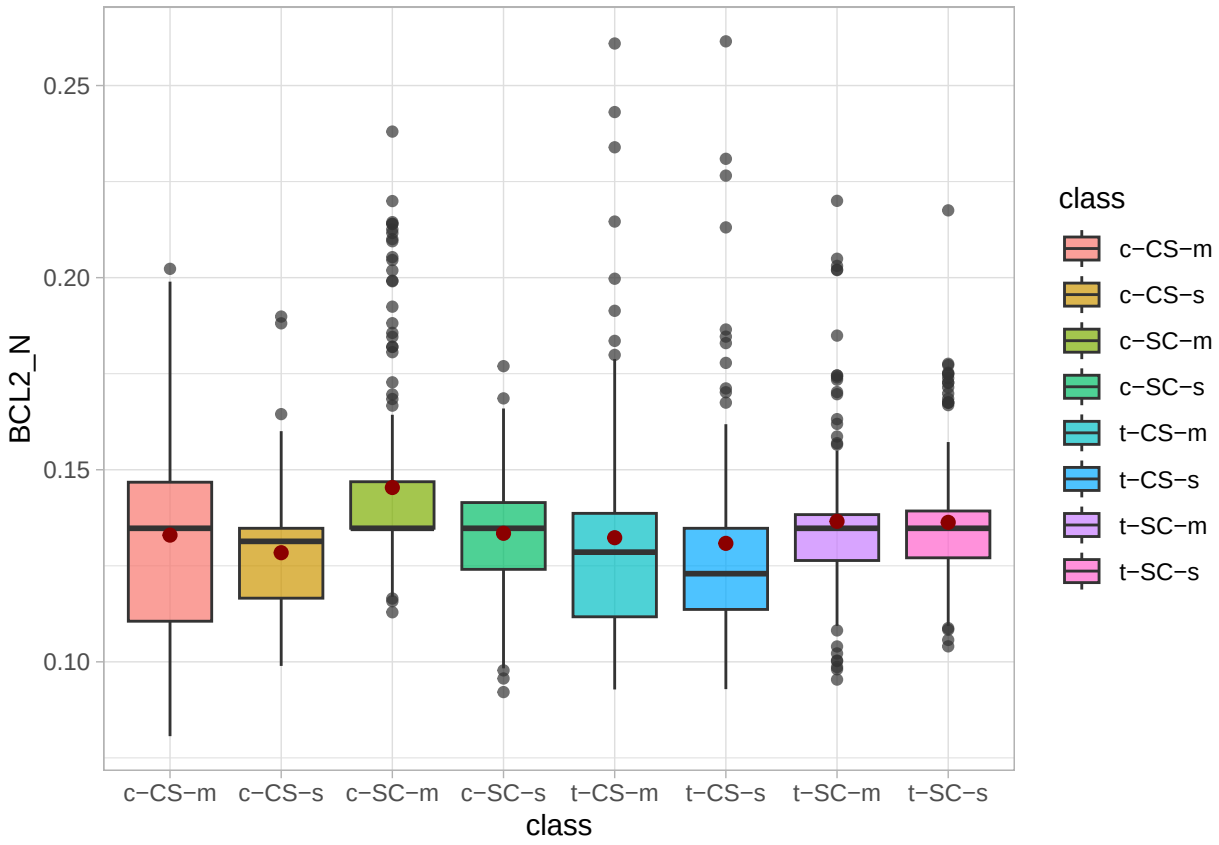
pGSK3B__Tyr216__N


SHH_N
0.35
0.30
0.25
0.20
0.15
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

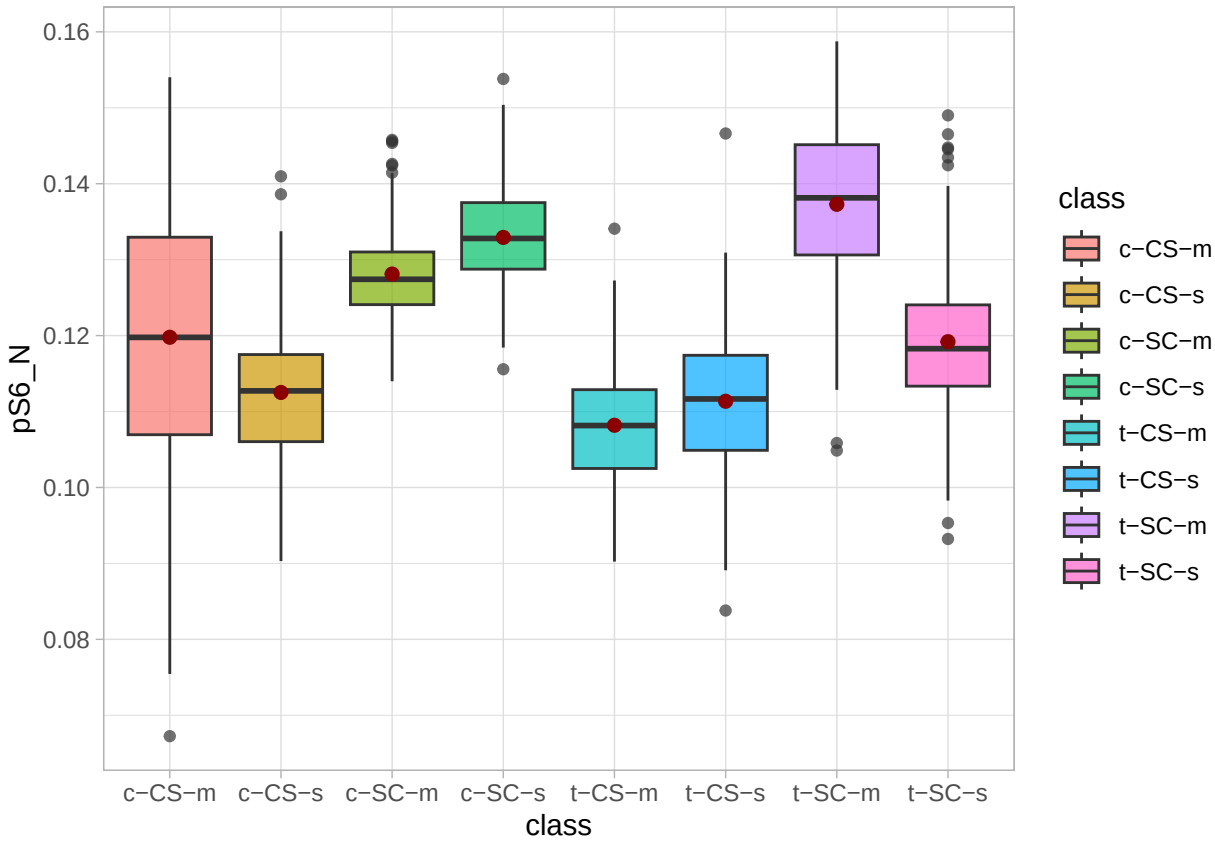
SHH_N


BAD_N
0.25
0.20
0.15
0.10
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

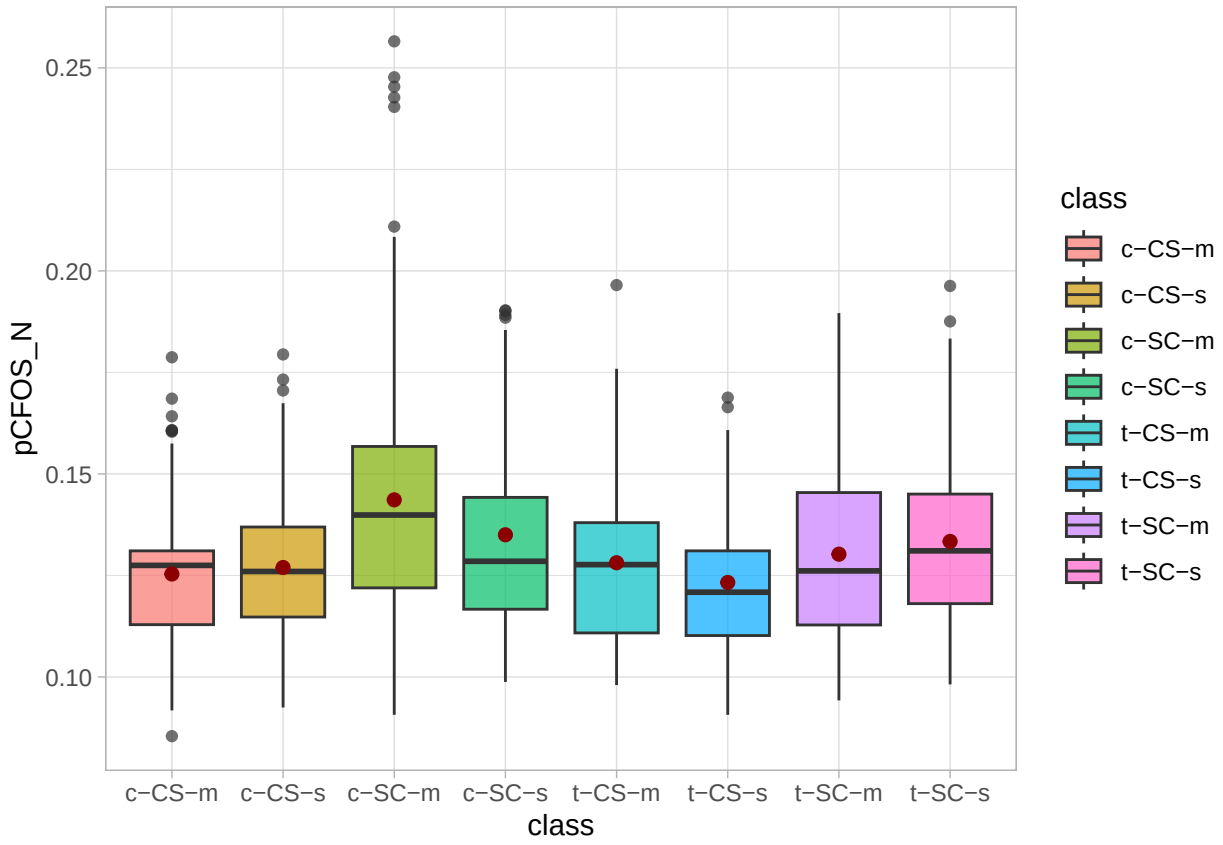
BAD_N


0.25
0.20
0.15
0.10
BCL2_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

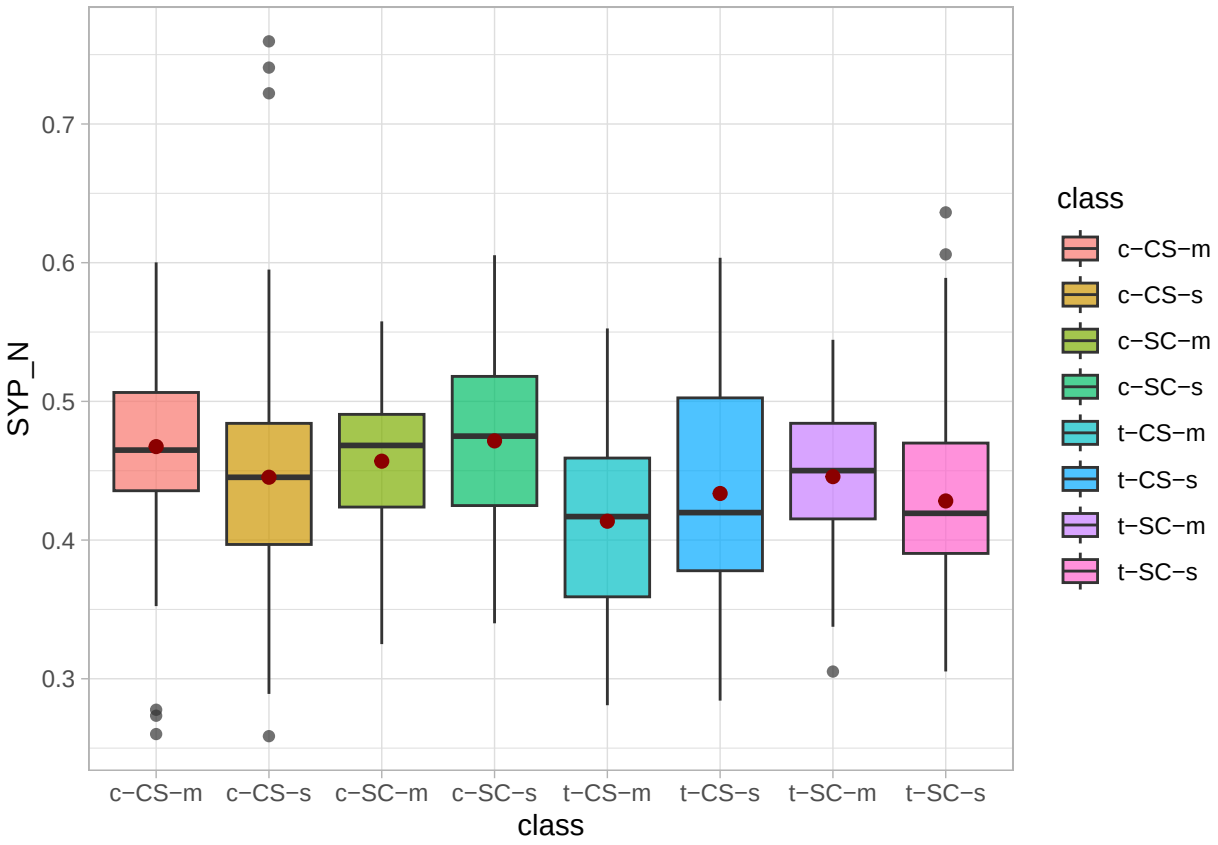
BCL2_N


0.16
0.14
0.12
0.10
0.08
pS6_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

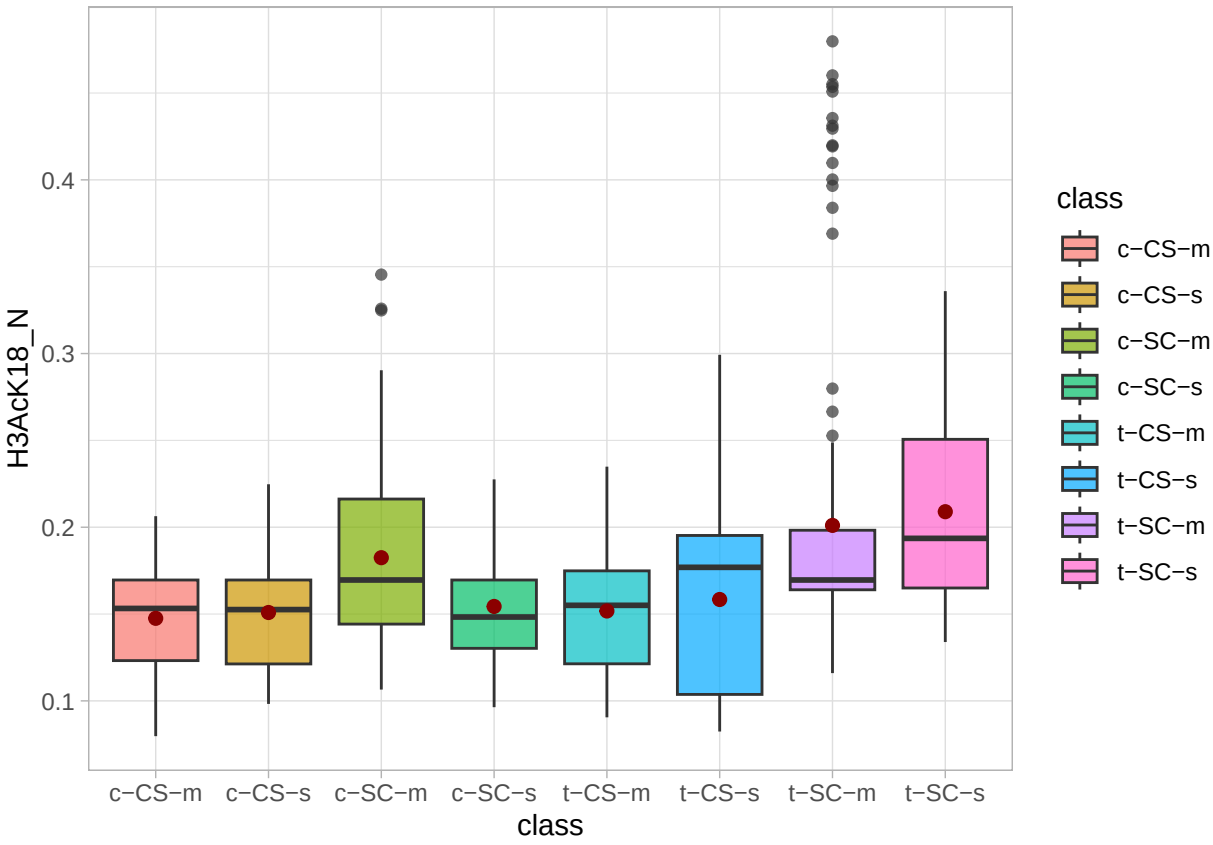
pS6_N


0.25
0.20
0.15
0.10
pCFOS_N
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

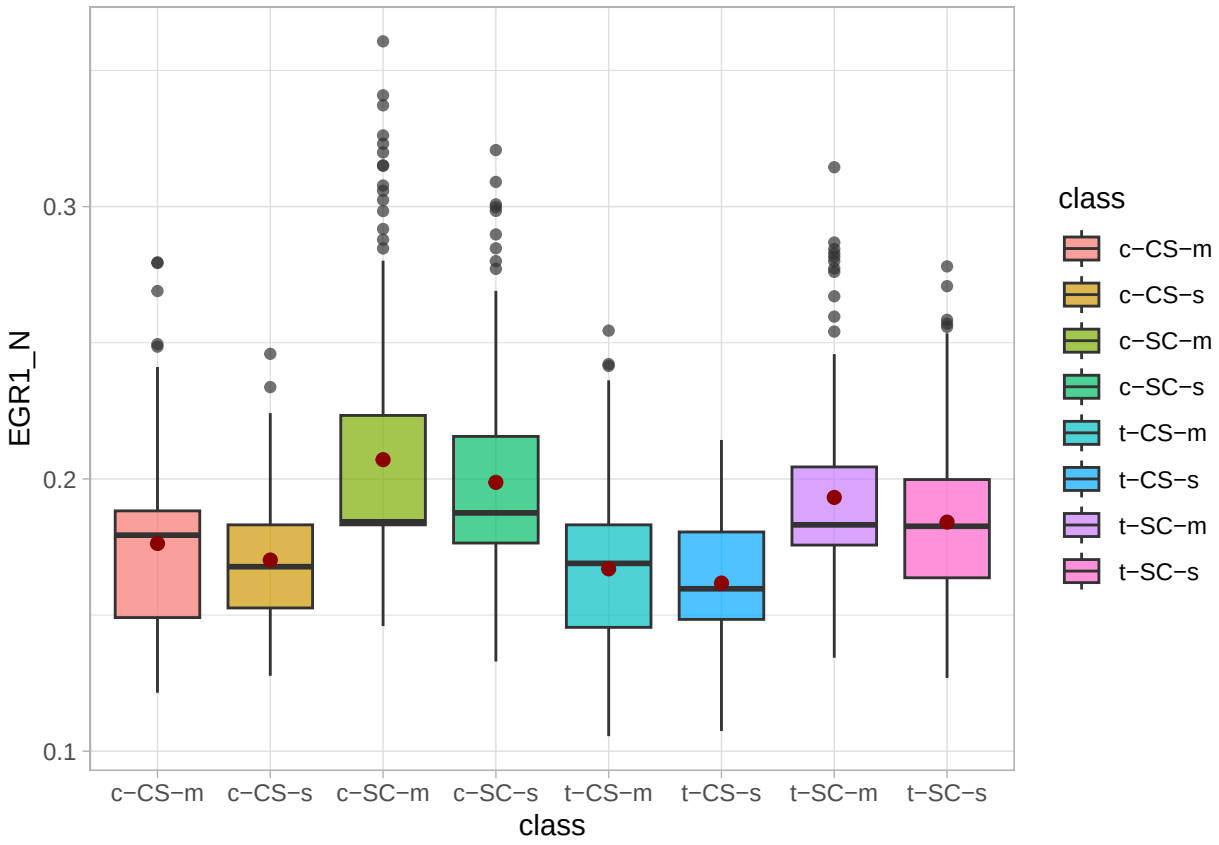
pCFOS_N


SYP_N
0.7
0.6
0.5
0.4
0.3
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

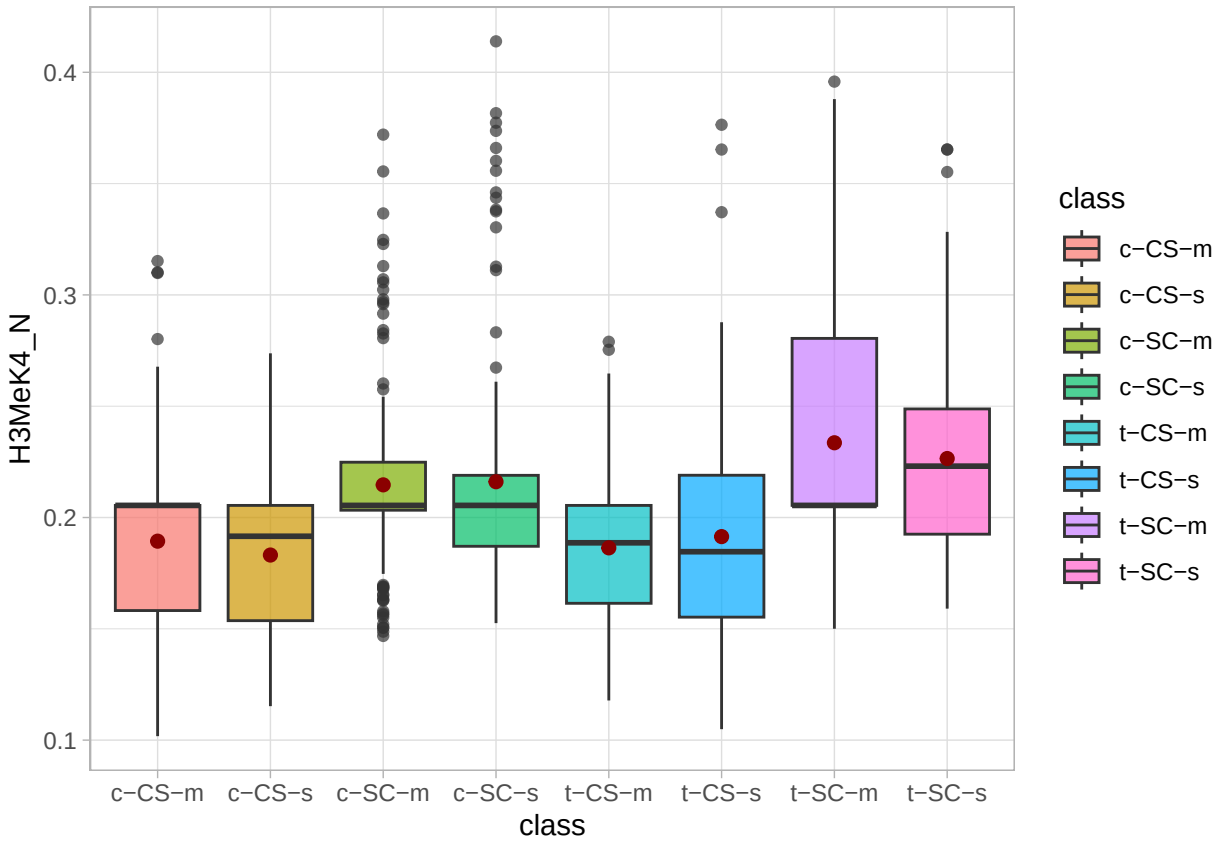
SYP_N


H3AcK18_N
0.4
0.3
0.2
0.1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

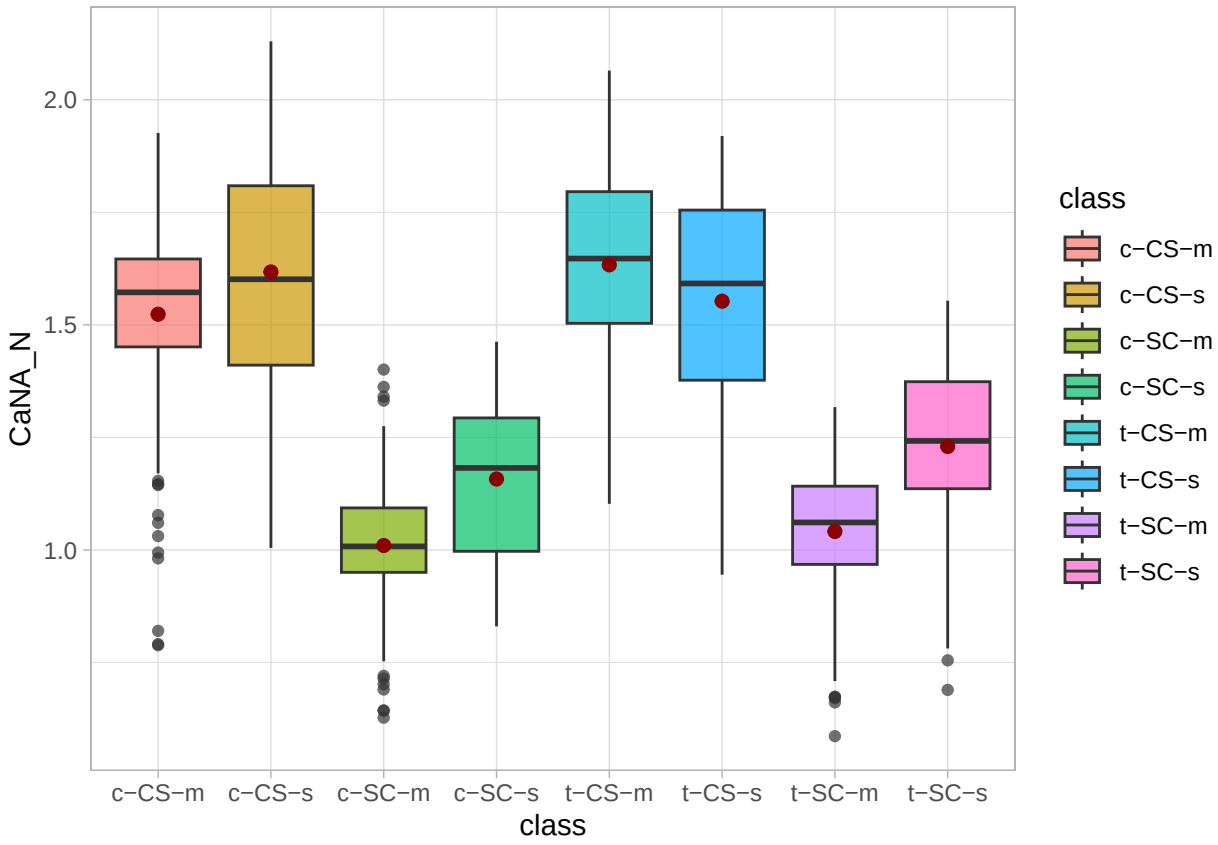
H3AcK18_N


EGR1_N
0.3
0.2
0.1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

EGR1_N


H3MeK4_N
0.4
0.3
0.2
0.1
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s
class
class
c−CS−m
c−CS−s
c−SC−m
c−SC−s
t−CS−m
t−CS−s
t−SC−m
t−SC−s

H3MeK4_N



CaNA_N

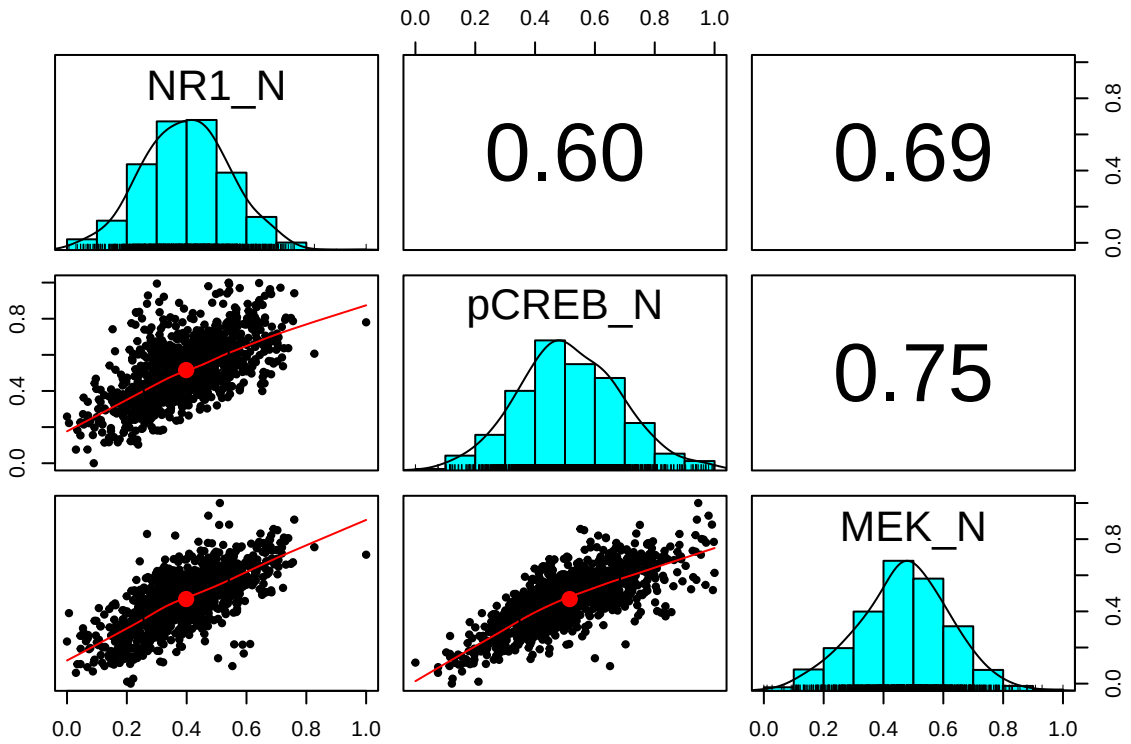
Normalization

For normalization, we use min-max scaling from preProcess() and validate it using Shapiro-Wilk test.

```
# Min-max scaling (range 0-1 for each column)
nc.feat <- ncortex[1:77]
min_max_scaling <- function(data) {
  pp <- preProcess(data.frame(data), method=c("range"))
  scaled.nc <- predict(pp, as.data.frame(data))
  return(scaled.nc)
}
scaled_ncortex <- min_max_scaling(data=nc.feat)

# check normality distribution using Shapiro-wilk test
testNormality <- function(data) {
  norm.res <- data.frame(do.call(
    cbind, lapply(data, function(x) shapiro.test(x)["p.value"])))
  unnorm.col <- list(which(norm.res > 0.05))
  print(paste("The unnormalized column(s) present in the data are", unnorm.col))
  return(unlist(unnorm.col))
}

# check our scaled data for any unnormalized column(s)
colX <- testNormality(data = scaled_ncortex)
```

[1] “The unnormalized column(s) present in the data are c(4, 9, 28)”

```
# check the distribution curves for the above columns
pairs.panels(scaled_ncortex[,colX])
```



After plotting the distribution curves of the above columns, we observe that they are fairly normal and do not require further transformation.

Feature Study

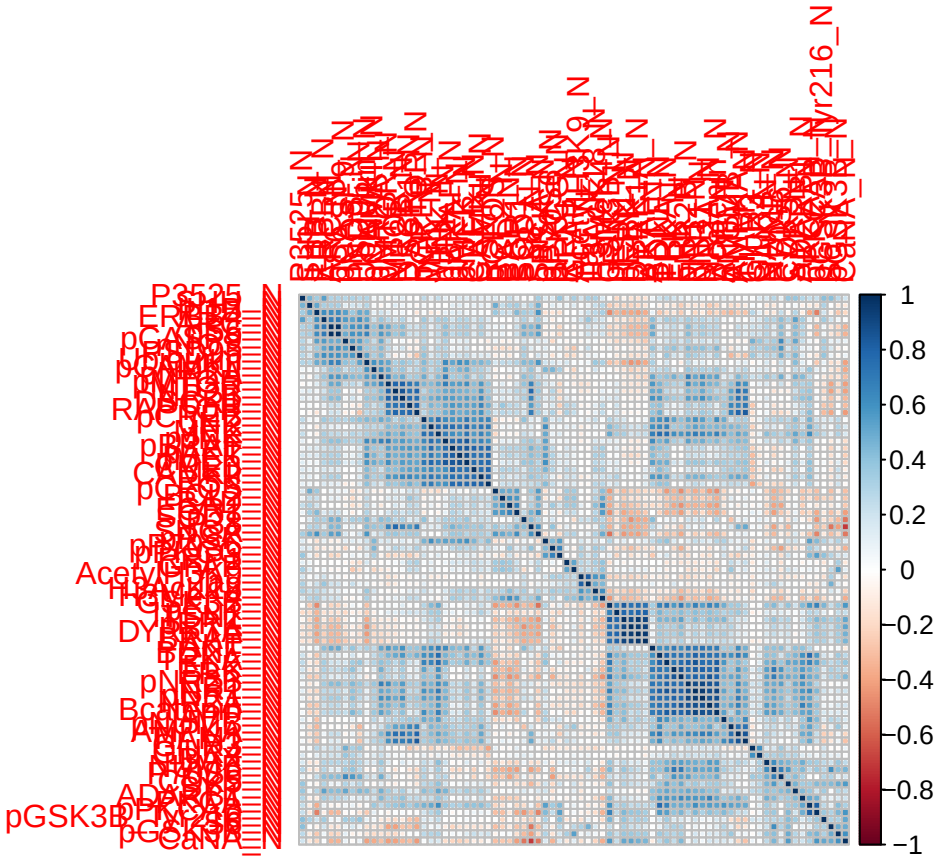
```
##########################################
########## Feature Engineering ###########
##########################################

# All columns are normally distributed
norm_ncortex <- scaled_ncortex
norm_ncortex$Class <- nc.Class
head(norm_ncortex)
```

```
dim(norm_ncortex)
```

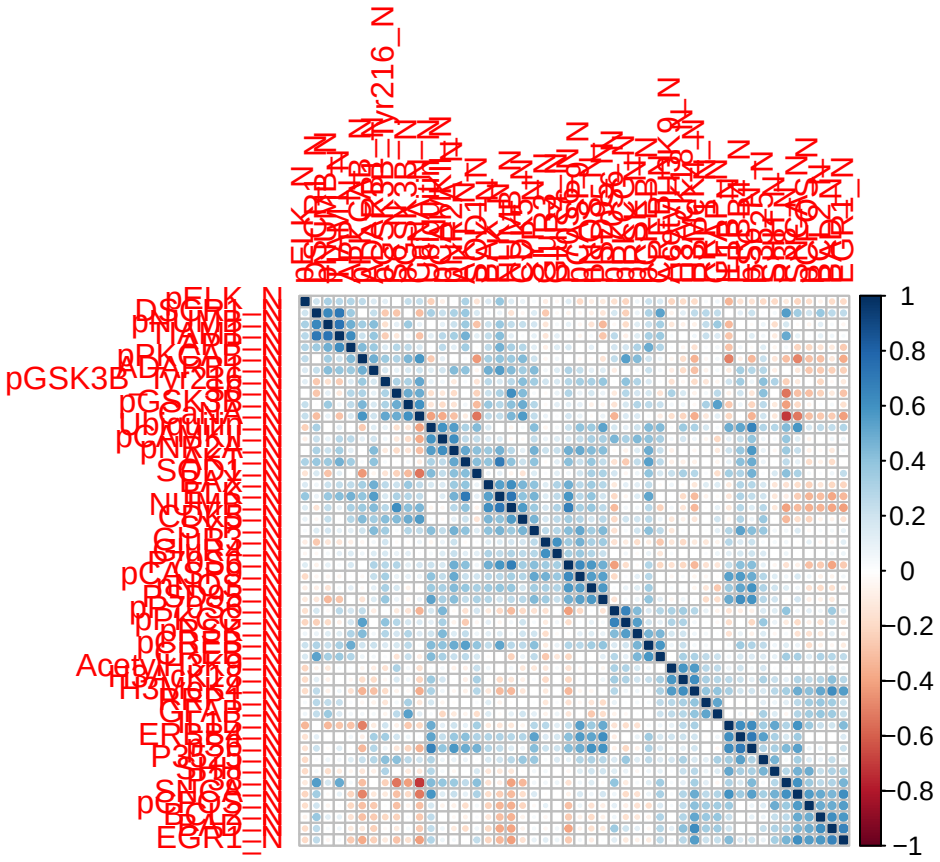
```
## [1] 1080   78
```

```
## Method-1: Correlation Cut-off
rawData <- scaled_ncortex
```

```
# compute the correlation matrix
corMatNC1 <- cor(rawData)
# visualize the matrix, clustering features by correlation index
corrplot(corMatNC1, order = "hclust")
```

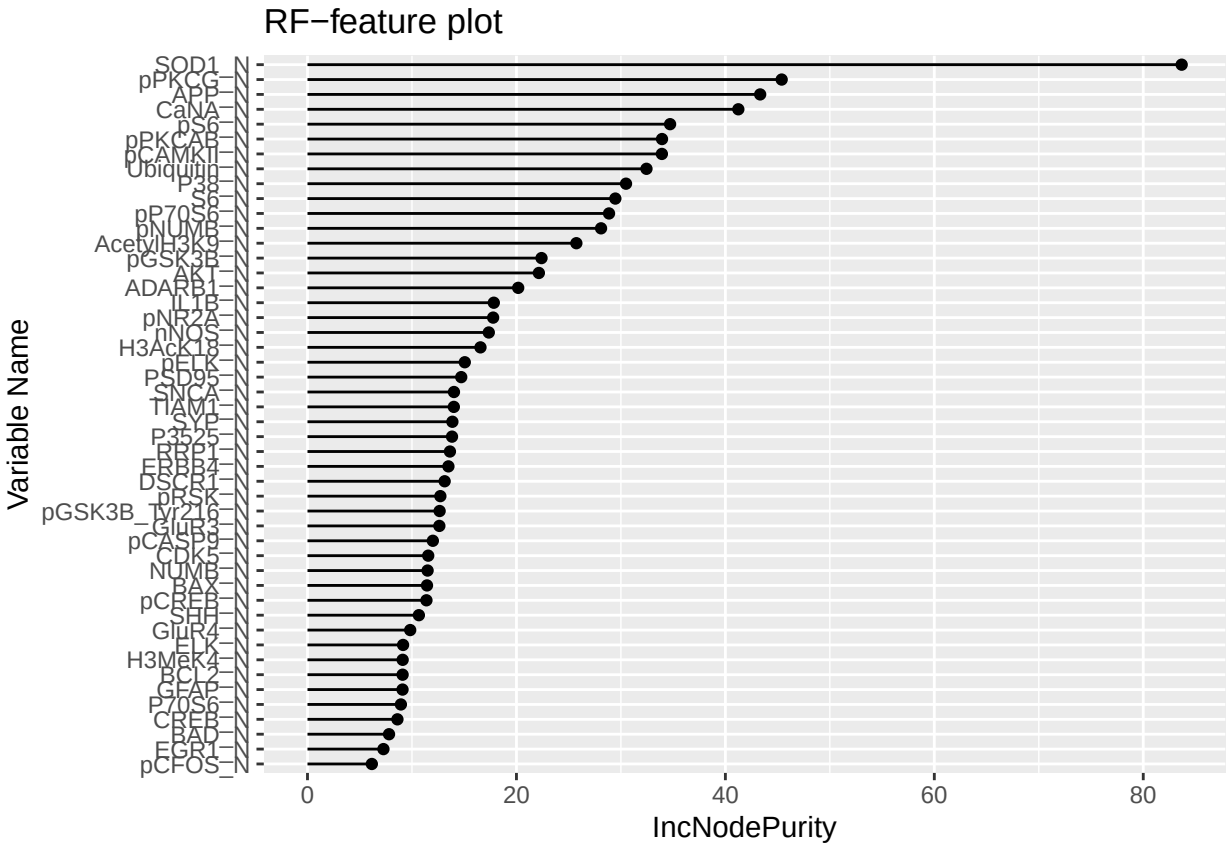



```
# After inspecting the matrix, we set the correlation threshold at 0.75
# Apply correlation filter at 0.75
highlyCor <- findCorrelation(corMatNC1, 0.75)
#then we remove all the variable correlated with more 0.75.
reduced_data <- rawData[,-highlyCor]
# check the effect of correlation cut-off filter
corMatNC2 <- cor(reduced_data)
corrplot(corMatNC2, order = "hclust")
```



For correlation method, we use an ideal factor of 0.75 coefficient value as the threshold. So variables having inter-correlation with more than 0.75 value are dropped.

```
## Method -2: Variable Importance
feat <- reduced_data
feat$Class <- nc.Class
# training `randomForest` to calculate feature importance
rf  <- randomForest(Class ~., data = feat)
var_imp <- varImp(rf, scale = FALSE)
# sort the score in decreasing order
var_imp_df <- data.frame(cbind(variable = rownames(var_imp), score = var_imp[,1]))
var_imp_df$score <- as.double(var_imp_df$score)
rf_scores <- var_imp_df[order(var_imp_df$score,decreasing = T),]
# setting up filter threshold --from the figure plotted down below
rf.filter <- sum(round(rf_scores$score) > 15)
# extracting data from filter
rf.feat <- ncortex[,rf_scores$variable[1:rf.filter]]
rf.feat$Class <- ncortex$class

# plotting the rf_scores to determine threshold
ggplot(rf_scores, aes(x=reorder(variable, score), y=score)) +
  geom_point() +
  geom_segment(aes(x=variable,xend=variable,y=0,yend=score)) +
  ggtitle("RF-feature plot") +
  ylab("IncNodePurity") +
  xlab("Variable Name") +
  coord_flip()
```



From the Rf-feature plot, we observe a constant downward trend from variable 15. Hence, we choose the threshold of "15" to filter our final set of features.

PCA

We perform PCA on our filtered data before we create feature subsets based on learning outcomes signified by our target classes.

```
##########################################
################ PCA  ####################
##########################################
# initialize data for PCA
dataPCA <- rf.feat
# dim(dataPCA)
# colnames(dataPCA)

## feature subset
# normal and failed learning (subset-1)
NL.data1 <- dataPCA %>% filter(Class == 'c-CS-s')
FL.data <- dataPCA %>% filter(Class == 't-CS-s')
# normal and rescued learning (subset-2)
NL.data2 <- dataPCA %>% filter(Class == 'c-CS-m')
RL.data <- dataPCA %>% filter(Class == 't-CS-m')

# feature subset
feat_data1 <- rbind(NL.data1, FL.data)
```

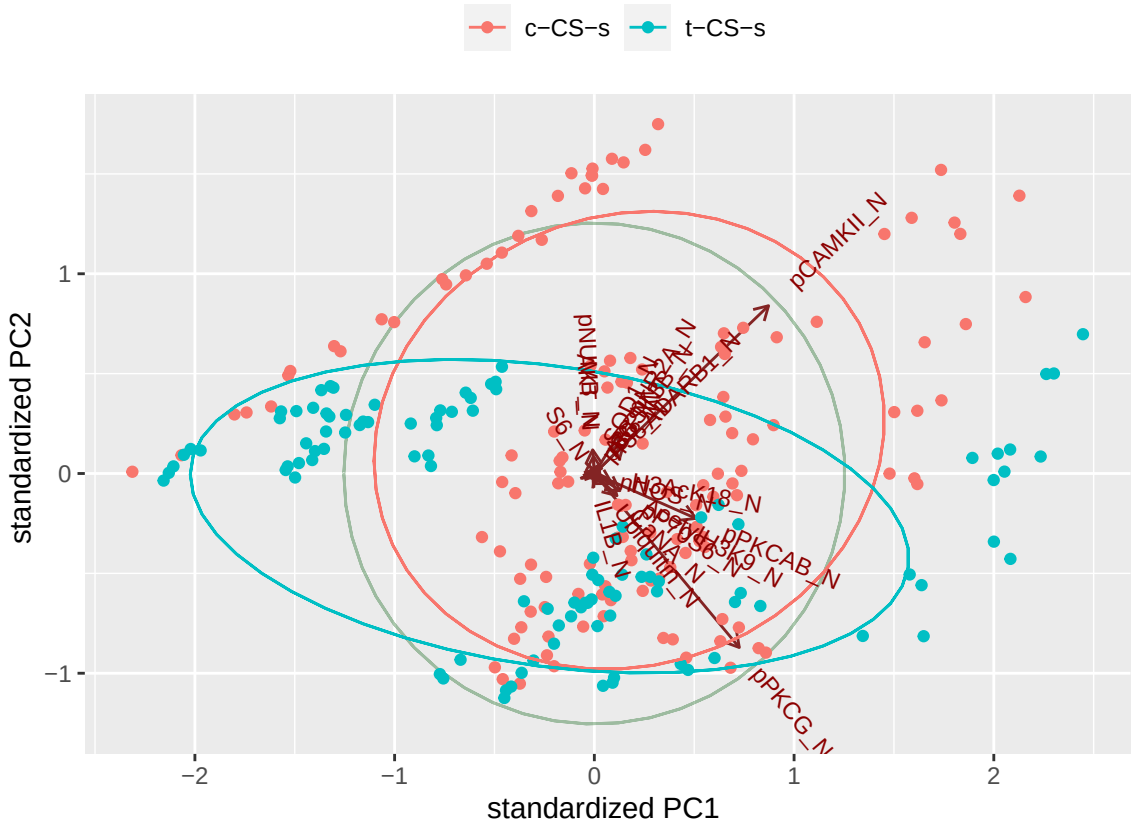
```
dim(feat_data1)
```

```
## [1] 240  21
```

```
feat_data2 <- rbind(NL.data2, RL.data)
dim(feat_data2)
```

```
## [1] 285  21
```

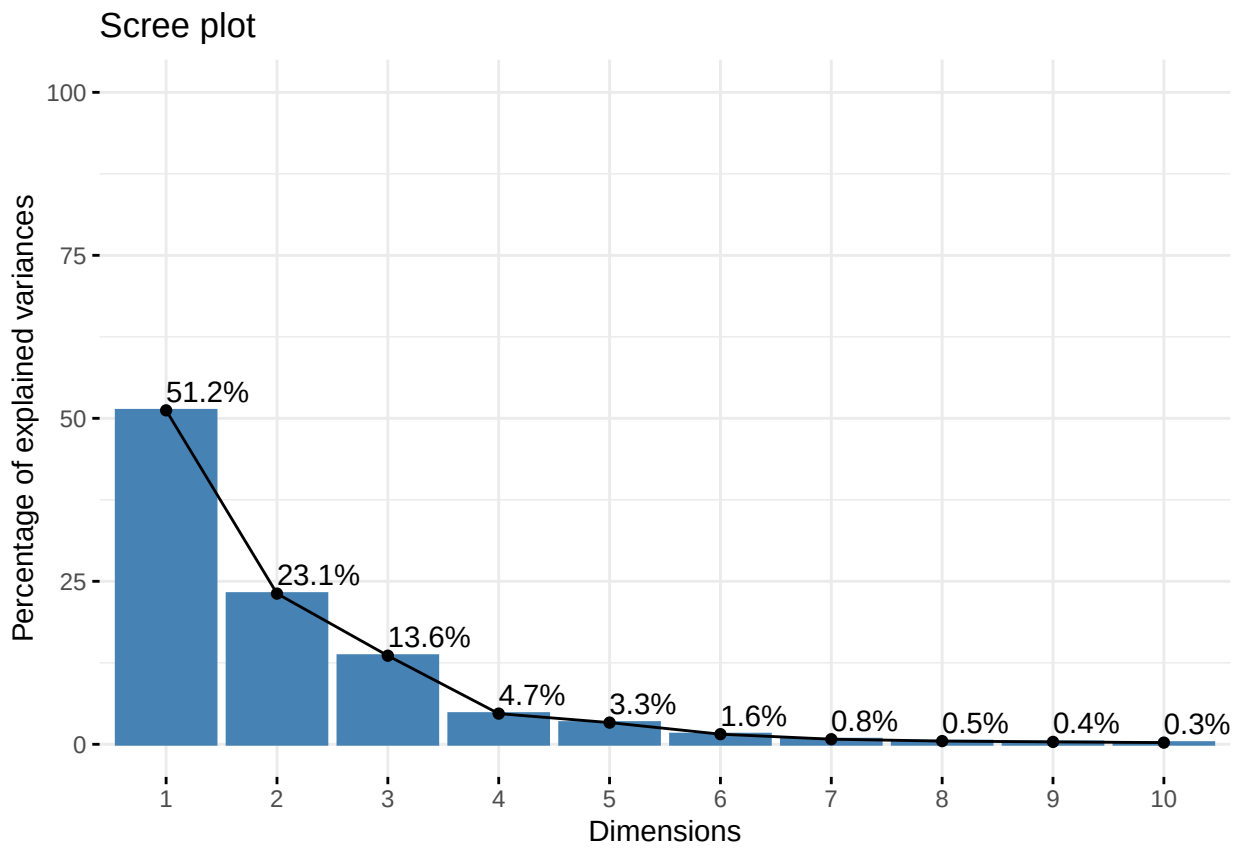
```
# PCA with function PCA
pca1 <- PCA(feat_data1[,-ncol(feat_data1)], scale.unit=F, graph=F)
pca2 <- PCA(feat_data2[,-ncol(feat_data2)], scale.unit=F, graph=F)

# custom function to analyze PCA results
pcaEval <- function(pca){
  scr.pca <- fviz_eig(pca, addlabels = T, ylim = c(0, 100))
  p.pca <- plot(pca, choix = "var", shadow = TRUE, select = "cos2")
  return(list(scr.pca, p.pca))
}

# biplot --to check distribution of observation among the two classes
PCA.plot <- function(pca, fdata) {
  ggbiplot(pca,
           groups = fdata$Class,
           ellipse = TRUE,
           circle = TRUE,
           ellipse.prob = 0.7) +
    scale_color_discrete(name = '')+
    theme(legend.direction = 'horizontal', legend.position = 'top')
}

# check pca results for feature subset-1
PCA.plot(pca1, feat_data1)  # biplot
```



From the above plot, we observe from pca1 analysis on feature subset-1 that there are few features that provide strong variance for the given learning outcomes such as `pCAMKII_N, pPKCG_N, pPKCAB_N`.

```
feval1 <- pcaEval(pca= pca1)
feval1[1] # scree plot
```

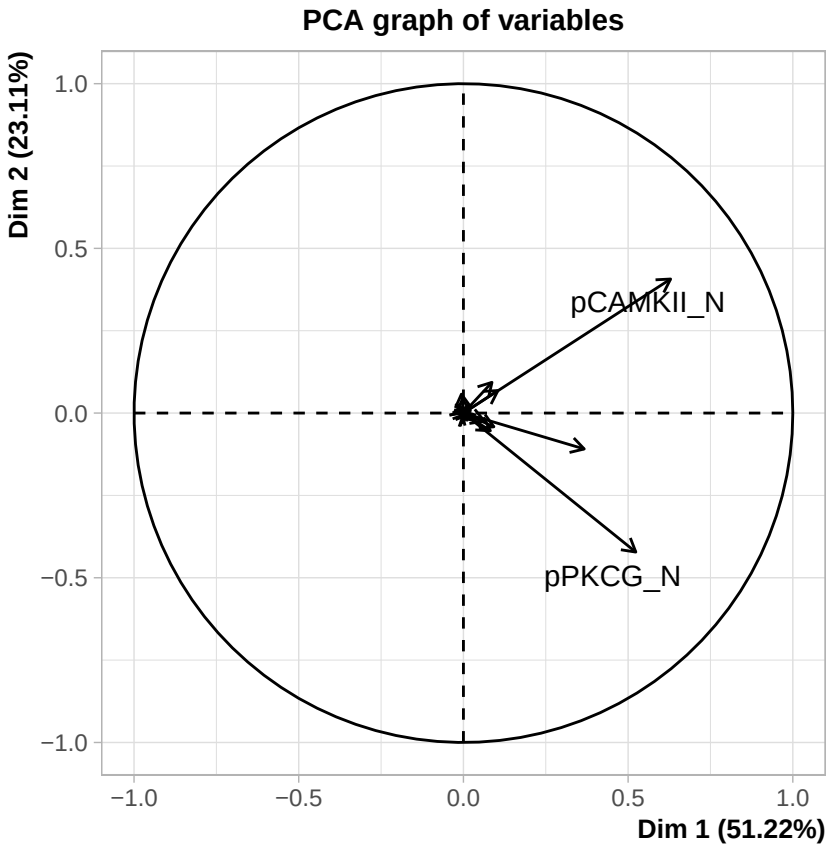
```
## [[1]]
```



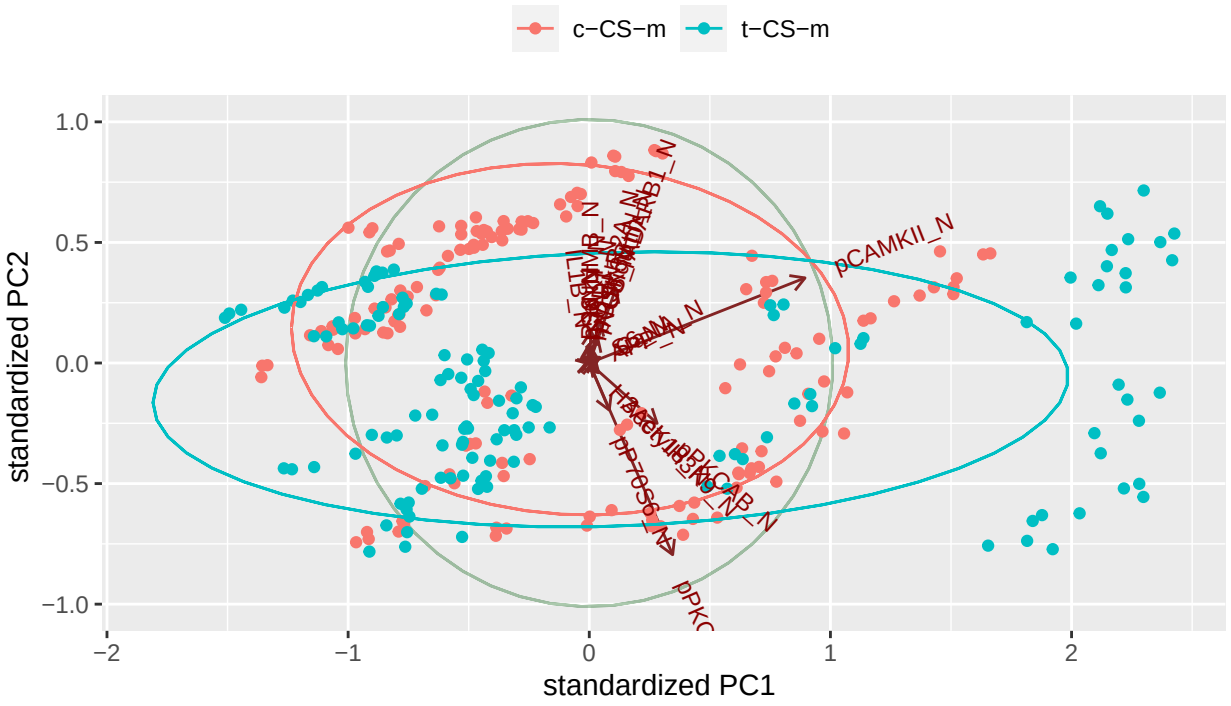
Here, from the scree plot, we notice that the first two PCs approximately describe 75% of variance in the subset-1.

```
feval1[2] # plot of important variables
```

```
## [[1]]

## Warning: ggrepel: 18 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



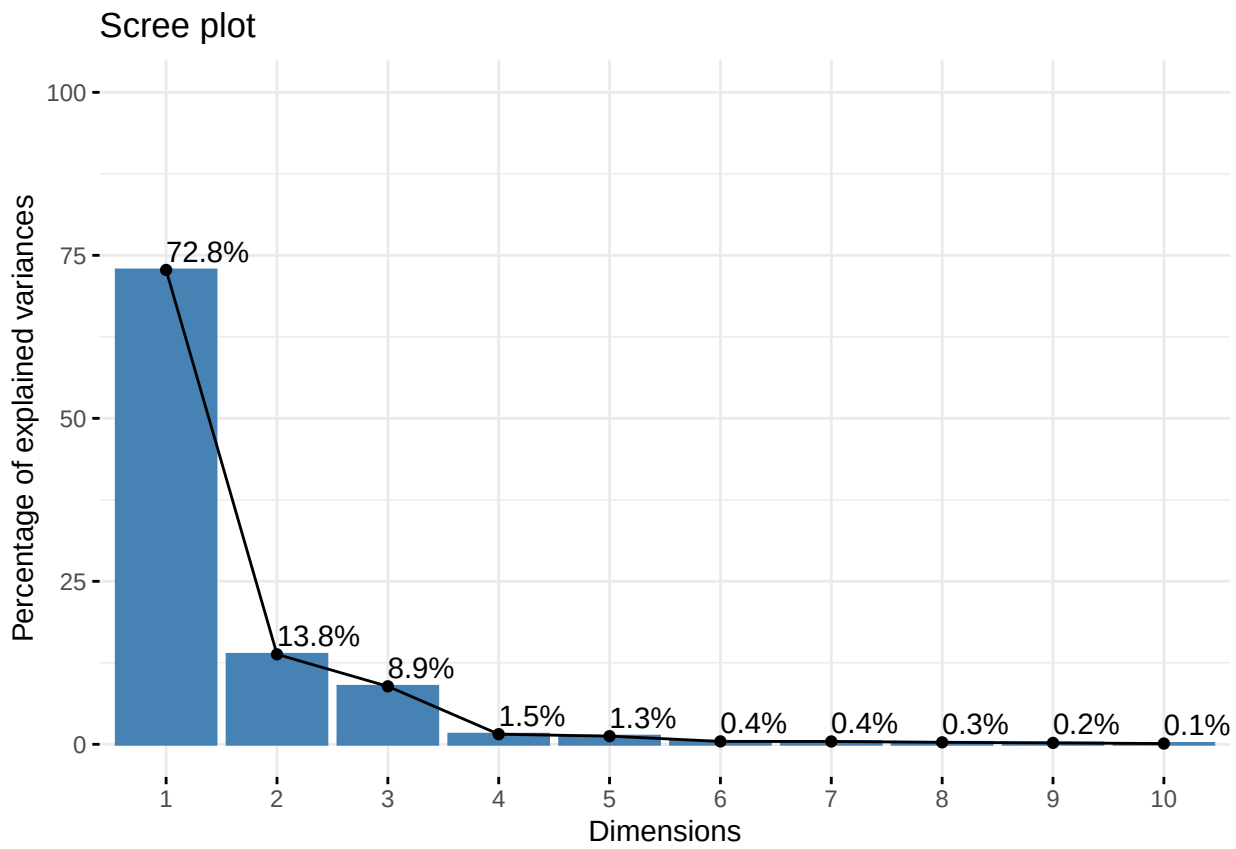

```
# check pca results for feature subset-2
PCA.plot(pca2, feat_data2) # biplot
```



From the biplot of subset-2, we get to understand that are similar features showing strong variance for different set of learning outcomes too.

```
feval2 <- pcaEval(pca= pca2)
feval2[1] # scree plot
```

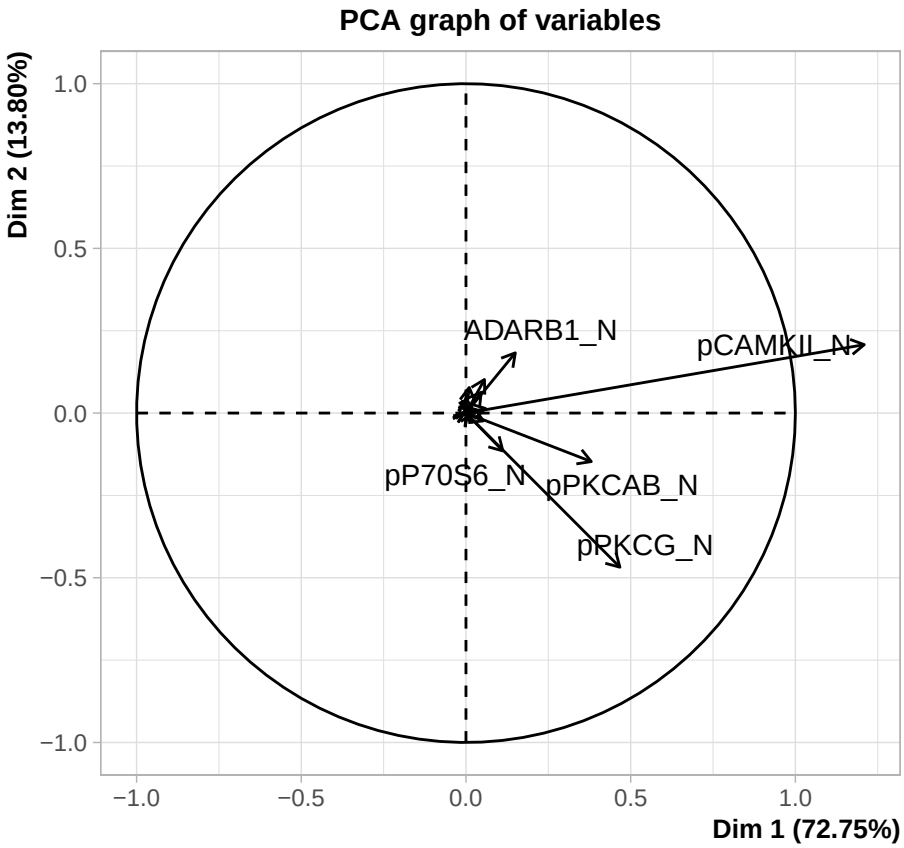
```
## [[1]]
```



From the second scree plot, we observe an explosive amount of variance from the first two PCs ~85%.

```
feval2[2] # plot of important variables
```

```
## [[1]]

## Warning: ggrepel: 15 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



The graph of variables are now very distinct for subset-2 with some handful features highlighted that may involve strongly for producing the respective learning outcomes.

Multi-class Classification Study

Splitting data set

```
########################################
####### Train-Valid-Test splitting #######
########################################
dataNC <- rf.feat
head(rf.feat)
```

```
# Simple into 3 sets
## --train: 60%
## --valid: 20%
## --test: 20%
set.seed(100)
splits = c(train = 0.6, test = 0.2, valid = 0.2)
grp = sample(cut(seq(nrow(dataNC)), nrow(dataNC)*cumsum(c(0,splits)),
                 labels = names(splits)))
# get the data splitted in a list
res = split(dataNC, grp)

# train set
```

```
trainX <- res$train
dim(trainX)
```

```
## [1] 648  21
```

```
# valid set
validX <- res$valid
dim(validX)
```

```
## [1] 216  21
```

```
# valid set
testX <- res$test
dim(testX)
```

```
## [1] 216  21
```

Custom formula

```
# formula for training
form <- paste(names(trainX)[1:ncol(trainX)-1], collapse = " + ")
formula.NC <- formula(paste(names(trainX)[ncol(trainX)], form, sep = " ~ "))
formula.NC  # check
```

```
## Class ~ SOD1_N + pPKCG_N + APP_N + CaNA_N + pS6_N + pPKCAB_N +
##     pCAMKII_N + Ubiquitin_N + P38_N + S6_N + pP70S6_N + pNUMB_N +
##     AcetylH3K9_N + pGSK3B_N + AKT_N + ADARB1_N + IL1B_N + pNR2A_N +
##     nNOS_N + H3AcK18_N
```

Training models

Naive Bayes

```
##########################################
############ Base Modeling  ##############
##########################################
##Naive Bayes
# fit the model
nb.model <- e1071::naiveBayes(formula.NC, trainX)
# make predictions using valid set
nb.pred <- predict(nb.model, validX, type="class")
# check reference and predicted classification
nb.cm.valid <- confusionMatrix(nb.pred, validX$Class)
nb.cm.valid$table
```

```
##            Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     19      5      0      0      0      3      0      0
##     c-CS-s      5     20      0      0      6      0      0      0
##     c-SC-m      0      0     31      0      0      0      3      0
```

```
##     c-SC-s      0      0      0     13      0      0      1      0
##     t-CS-m     10      0      0      0     26      3      0      0
##     t-CS-s      0      1      0      0      2     17      0      0
##     t-SC-m      0      0      0      0      0      0     21      0
##     t-SC-s      0      0      0      5      0      0      0     25
```

```
nb.cm.valid$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.7962963 0.7657094
```

Here, from our base 'NB' model, we observe there are several high chunks of misclassification. Hence, we perform a typical k-fold cross-validation (k=5) to improve the classification of the model.

```
# k-fold cross validation (k=5)
set.seed(213)
NB.tune <- data.frame(fL=c(0,0.5,1.0), usekernel = TRUE, adjust=c(0,0.5,1.0))
# model training with tuned hyper-parameters
NB.model <- caret::train(formula.NC,trainX,'nb',
             trControl=trainControl(method= 'cv',number= 5),
             tuneGrid = NB.tune)
```

```
## Warning: model fit failed for Fold1: fL=0.0, usekernel=TRUE, adjust=0.0 Error in density.default(xx,

## Warning: model fit failed for Fold2: fL=0.0, usekernel=TRUE, adjust=0.0 Error in density.default(xx,

## Warning: model fit failed for Fold3: fL=0.0, usekernel=TRUE, adjust=0.0 Error in density.default(xx,

## Warning: model fit failed for Fold4: fL=0.0, usekernel=TRUE, adjust=0.0 Error in density.default(xx,

## Warning: model fit failed for Fold5: fL=0.0, usekernel=TRUE, adjust=0.0 Error in density.default(xx,

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.

## Warning in train.default(x, y, weights = w, ...): missing values found in
## aggregated results
```

```
# make predictions using valid set
NB.pred <- predict(NB.model, validX)
NB.cm.valid <- confusionMatrix(NB.pred, validX$Class)
NB.cm.valid$table
```

```
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     20      2      0      0      2      3      0      0
##     c-CS-s      6     22      0      0      5      0      0      0
##     c-SC-m      0      0     28      0      0      0      1      0
##     c-SC-s      0      0      3     13      0      0      1      0
##     t-CS-m      8      2      0      0     25      2      0      0
##     t-CS-s      0      0      0      0      2     18      0      0
##     t-SC-m      0      0      0      0      0      0     23      0
##     t-SC-s      0      0      0      5      0      0      0     25
```

```
NB.cm.valid$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.8055556 0.7767277
```

Artificial Neural Netwok

Here, I have set the argument "linear.output" to `FALSE`in order to tell the model that I want to apply the activation function and that I am not doing a regression task. Then I set the activation function to `logistic` (which by the way is the default option) in order to apply the logistic function.As far as the number of hidden neurons and layers, I tried some combinations and the one used seemed to perform slightly better than the others (around 1% of accuracy difference in cross-validation score.

```
set.seed(334)
# fit the model
NN.model = neuralnet::neuralnet(formula.NC, trainX,
                     hidden=c(14, 10, 8), linear.output = FALSE,
                     act.fct ="logistic")
# plot the neural network
plot(NN.model)

tablePred.NN <- function(model, data){
  # make predictions using valid set
  raw.pred <- data.frame(neuralnet::compute(model,
                              data.frame(data[,-ncol(data)]))$net.result)
  # initialize target labels
  labels <- c("c-CS-m", "c-CS-s", "c-SC-m", "c-SC-s",
              "t-CS-m", "t-CS-s", "t-SC-m", "t-SC-s")
  pred.label <- data.frame(max.col(raw.pred)) %>%
  mutate(prediction=labels[max.col.raw.pred.])
  # make predictions using valid set
  preds <- as.factor(pred.label[,2])
  confMat <- confusionMatrix(validX$Class, preds, mode="prec_recall")
  return(list(preds,confMat))
}
NN.cm.valid <- tablePred.NN(NN.model, validX)
NN.cm.valid[2]
```

```
## [[1]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     32      1      0      0      0      1      0      0
##     c-CS-s      1     22      0      0      2      1      0      0
##     c-SC-m      0      0     31      0      0      0      0      0
##     c-SC-s      1      2      0     13      0      0      0      2
##     t-CS-m      1      4      0      0     29      0      0      0
##     t-CS-s      0      0      0      0      3     20      0      0
##     t-SC-m      0      0      0      0      0      0     25      0
##     t-SC-s      0      0      1      0      0      0      0     24
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9074
##                  95% CI : (0.8606, 0.9425)
##     No Information Rate : 0.162
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8935
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: c-CS-m Class: c-CS-s Class: c-SC-m Class: c-SC-s
## Precision                   0.9412        0.8462        1.0000       0.72222
## Recall                      0.9143        0.7586        0.9688       1.00000
## F1                          0.9275        0.8000        0.9841       0.83871
## Prevalence                  0.1620        0.1343        0.1481       0.06019
## Detection Rate              0.1481        0.1019        0.1435       0.06019
## Detection Prevalence        0.1574        0.1204        0.1435       0.08333
## Balanced Accuracy           0.9516        0.8686        0.9844       0.98768
##                      Class: t-CS-m Class: t-CS-s Class: t-SC-m Class: t-SC-s
## Precision                   0.8529       0.86957        1.0000        0.9600
## Recall                      0.8529       0.90909        1.0000        0.9231
## F1                          0.8529       0.88889        1.0000        0.9412
## Prevalence                  0.1574       0.10185        0.1157        0.1204
## Detection Rate              0.1343       0.09259        0.1157        0.1111
## Detection Prevalence        0.1574       0.10648        0.1157        0.1157
## Balanced Accuracy           0.9127       0.94681        1.0000        0.9589
```

Multinomial Logistic Regression

For our third model, we use Multinomial logistic regression from `nnet()` package since it is highly preferred if you want to perform multi-class classification using logistic regression.

```
set.seed(232)
# fit the model
MGR.model <- nnet::multinom(formula.NC, data = trainX, trace = F)
# make predictions using valid set
MGR.pred <- predict(MGR.model, newdata=validX)
# check reference and predicted classification
MGR.cm.valid <- confusionMatrix(validX$Class, MGR.pred)
MGR.cm.valid$table
```

```
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     31      2      0      0      0      0      1      0
##     c-CS-s      1     23      0      0      2      0      0      0
##     c-SC-m      0      0     31      0      0      0      0      0
##     c-SC-s      0      0      0     18      0      0      0      0
##     t-CS-m      2      1      0      0     30      1      0      0
##     t-CS-s      0      0      0      0      1     22      0      0
##     t-SC-m      0      0      0      1      0      0     24      0
##     t-SC-s      0      0      0      0      0      0      0     25
```

```
MGR.cm.valid$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.9444444 0.9361891
```

Test performance of Base learners

```
# evaluate base learners test performances
# model1
NB.pred.test <- predict(NB.model, testX)
NB.cm.test <- confusionMatrix(NB.pred.test, testX$Class, mode="prec_recall")
NB.cm.test$table
```

```
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     27      0      0      0      3      3      0      0
##     c-CS-s      6     18      0      0      6      0      0      0
##     c-SC-m      0      0     27      0      0      0      0      0
##     c-SC-s      0      0      3     25      0      0      0      0
##     t-CS-m      3      2      0      0     21      2      0      0
##     t-CS-s      0      3      0      0      1     15      0      0
##     t-SC-m      0      0      0      3      0      0     22      0
##     t-SC-s      1      0      0      0      0      0      0     25
```

```
# model2
NN.tablePred <- tablePred.NN(NN.model, testX)
NN.cm.test <- NN.tablePred[2]
NN.cm.test
```

```
## [[1]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     33      0      0      0      1      0      0      0
##     c-CS-s      1     22      2      1      0      0      0      0
##     c-SC-m      3      0     27      0      0      0      1      0
##     c-SC-s      0      0      0     18      0      0      0      0
##     t-CS-m      0      0      0      9     25      0      0      0
##     t-CS-s      0      0      0      0      0     20      3      0
##     t-SC-m      0      1      1      0      5      0     18      0
##     t-SC-s      0      0      1      0      0      0      0     24
##
## Overall Statistics
##
##                Accuracy : 0.8657
##                  95% CI : (0.8129, 0.9082)
##     No Information Rate : 0.1713
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.846
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: c-CS-m Class: c-CS-s Class: c-SC-m Class: c-SC-s
## Precision                  0.9706        0.8462        0.8710       1.00000
## Recall                     0.8919        0.9565        0.8710       0.64286
## F1                         0.9296        0.8980        0.8710       0.78261
## Prevalence                 0.1713        0.1065        0.1435       0.12963
## Detection Rate             0.1528        0.1019        0.1250       0.08333
## Detection Prevalence       0.1574        0.1204        0.1435       0.08333
## Balanced Accuracy          0.9432        0.9679        0.9247       0.82143
##                      Class: t-CS-m Class: t-CS-s Class: t-SC-m Class: t-SC-s
## Precision                  0.7353       0.86957       0.72000        0.9600
## Recall                     0.8065       1.00000       0.81818        1.0000
## F1                         0.7692       0.93023       0.76596        0.9796
## Prevalence                 0.1435       0.09259       0.10185        0.1111
## Detection Rate             0.1157       0.09259       0.08333        0.1111
## Detection Prevalence       0.1574       0.10648       0.11574        0.1157
## Balanced Accuracy          0.8789       0.99235       0.89105        0.9974
```

```
# model3
MGR.pred.test <- predict(MGR.model, testX)
MGR.cm.test <- confusionMatrix(MGR.pred.test, testX$Class)
MGR.cm.test$table
```

```
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     30      0      0      0      1      0      0      0
##     c-CS-s      1     21      0      0      1      1      0      0
##     c-SC-m      0      0     28      1      0      0      2      0
##     c-SC-s      0      2      0     25      0      0      0      0
##     t-CS-m      3      0      0      0     29      0      0      0
##     t-CS-s      1      0      0      0      0     19      0      1
##     t-SC-m      2      0      2      2      0      0     20      0
##     t-SC-s      0      0      0      0      0      0      0     24
```

Bagging

```
## In-built Ensemble Model --bagging
# fit the model
bag.model <- bagging(formula.NC, trainX)
# estimate predictions on test data
bag.pred <- predict(bag.model, testX)
# get the model performance
bag.cm.test <- confusionMatrix(as.factor(bag.pred), testX$Class)
bag.cm.test$table
```

```
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     34      0      0      0      0      1      0      0
```

```
##      c-CS-s      0     22      0      0      1      2      0      0
##      c-SC-m      1      0     28      0      0      0      1      1
##      c-SC-s      0      0      2     27      0      0      0      0
##      t-CS-m      2      1      0      0     29      0      0      0
##      t-CS-s      0      0      0      0      1     17      0      0
##      t-SC-m      0      0      0      1      0      0     21      0
##      t-SC-s      0      0      0      0      0      0      0     24
```

Stacked Ensemble Learner

Here, we create a stacked ensemble learner to check whether we can improve the classification performance for analysis even more. Here, we first generate raw/prob prediction values of our three base models on valid data and use it as level-one dataset for training ensemble learner using "rf" (as decision tree based models always perform quite well for classification). After fitting our model on level-one dataset, we generate raw test predictions from base learner as our test dataset for predicting values from our super learner.

```
# Custom Stacked Ensemble learner
stackLearner <- function(model1, model2, model3, testData, validData){
  ## --NB.model: model1 (Naive Bayes)
  ## --NN.model: model2 (Artificial Neural Network)
  ## --MGR.model: model3 (Multinomial Logistic regression)

  # raw predictions from the base models
  validpred1 <- predict(model1, validData, type= "raw")
  validpred2 <- predict(model2, validData, type= "raw")
  colnames(validpred2) <- colnames(validpred1)
  validpred3 <- predict(model3, validData, type= "probs")
  # Generate level-one dataset for training the ensemble super-learner
  basePred <- data.frame(validpred1, validpred2, validpred3, Class= validData$Class,
                         stringsAsFactors = F)

  # Train the ensemble
  set.seed(270)
  modelStack <- train(Class~., data = basePred, method = "rf")

  # Generate predictions on the test set
  testPred1 <- predict(model1, newdata = testData,type= "raw")
  testPred2 <- predict(model2, newdata = testData, type= "raw")
  testPred3 <- predict(model3, newdata = testData,type= "probs")

  # Using the base learner test set predictions,
  # create the level-one dataset to feed to the ensemble
  testPredLevelOne <- data.frame(testPred1, testPred2, testPred3,
                                 Class= testX$Class, stringsAsFactors = F)
  # estimate predictions over tested base models
  colnames(testPredLevelOne) <- colnames(basePred)
  combPred <- predict(modelStack, testPredLevelOne)

  # Evaluate ensemble test performance
  confMatrix <- confusionMatrix(combPred, testData$Class, mode="prec_recall")
  return(list(confMatrix, combPred))
}
```

```r
# func call
stackResults  <- stackLearner(model1 = NB.model,
                              model2 = NN.model,
                              model3 = MGR.model,
                              testData = testX, validData = validX)
# Ensemble prediction matrix
ENS.confMat <- stackResults[1]
ENS.confMat
```

```
## [[1]]
## Confusion Matrix and Statistics
##
##           Reference
## Prediction c-CS-m c-CS-s c-SC-m c-SC-s t-CS-m t-CS-s t-SC-m t-SC-s
##     c-CS-m     36      1      0      0      1      1      0      0
##     c-CS-s      0     20      0      0      1      1      0      0
##     c-SC-m      0      0     29      1      0      0      2      0
##     c-SC-s      0      2      1     27      0      0      0      0
##     t-CS-m      1      0      0      0     29      0      0      0
##     t-CS-s      0      0      0      0      0     18      0      1
##     t-SC-m      0      0      0      0      0      0     20      0
##     t-SC-s      0      0      0      0      0      0      0     24
##
## Overall Statistics
##
##                Accuracy : 0.9398
##                  95% CI : (0.8993, 0.9676)
##     No Information Rate : 0.1713
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9308
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: c-CS-m Class: c-CS-s Class: c-SC-m Class: c-SC-s
## Precision                  0.9231       0.90909        0.9062        0.9000
## Recall                     0.9730       0.86957        0.9667        0.9643
## F1                         0.9474       0.88889        0.9355        0.9310
## Prevalence                 0.1713       0.10648        0.1389        0.1296
## Detection Rate             0.1667       0.09259        0.1343        0.1250
## Detection Prevalence       0.1806       0.10185        0.1481        0.1389
## Balanced Accuracy          0.9781       0.92960        0.9753        0.9742
##                      Class: t-CS-m Class: t-CS-s Class: t-SC-m Class: t-SC-s
## Precision                  0.9667       0.94737       1.00000        1.0000
## Recall                     0.9355       0.90000       0.90909        0.9600
## F1                         0.9508       0.92308       0.95238        0.9796
## Prevalence                 0.1435       0.09259       0.10185        0.1157
## Detection Rate             0.1343       0.08333       0.09259        0.1111
## Detection Prevalence       0.1389       0.08796       0.09259        0.1111
## Balanced Accuracy          0.9650       0.94745       0.95455        0.9800
```

From the matrix summary, it is quite evident that are stacked ensemble showcased exceptional results in

terms of accuracy and kappa metrics (~98% and ~97%).

Evaluation Metrics for all models

```
# custom functio to print important metrics of our analyses
evalMetrics <- function(confMat){
  # metrics-1
  print("The accuracy and kappa of the Naive Bayes model are")
  print(round(confMat$overall[1:2] * 100,2))

  # metrics-2
  print("The precision, recall and F1-score for target classes are")
  print(confMat$byClass[c(1,2,5,6),c(5,6,7)])
}

# model-1: Naive Bayes
evalMetrics(confMat = NB.cm.test)
```

```
## [1] "The accuracy and kappa of the Naive Bayes model are"
## Accuracy    Kappa
##    83.33    80.90
## [1] "The precision, recall and F1-score for target classes are"
##                Precision    Recall        F1
## Class: c-CS-m 0.8181818 0.7297297 0.7714286
## Class: c-CS-s 0.6000000 0.7826087 0.6792453
## Class: t-CS-m 0.7500000 0.6774194 0.7118644
## Class: t-CS-s 0.7894737 0.7500000 0.7692308
```

```
# model-3: Multi-nomial Logistic Regression
evalMetrics(confMat = MGR.cm.test)
```

```
## [1] "The accuracy and kappa of the Naive Bayes model are"
## Accuracy    Kappa
##    90.74    89.38
## [1] "The precision, recall and F1-score for target classes are"
##                Precision    Recall        F1
## Class: c-CS-m 0.9677419 0.8108108 0.8823529
## Class: c-CS-s 0.8750000 0.9130435 0.8936170
## Class: t-CS-m 0.9062500 0.9354839 0.9206349
## Class: t-CS-s 0.9047619 0.9500000 0.9268293
```

```
# model-5: Bagging
evalMetrics(confMat = bag.cm.test)
```

```
## [1] "The accuracy and kappa of the Naive Bayes model are"
## Accuracy    Kappa
##    93.52    92.55
## [1] "The precision, recall and F1-score for target classes are"
##                Precision    Recall        F1
## Class: c-CS-m 0.9714286 0.9189189 0.9444444
```

```
## Class: c-CS-s 0.8800000 0.9565217 0.9166667
## Class: t-CS-m 0.9062500 0.9354839 0.9206349
## Class: t-CS-s 0.9444444 0.8500000 0.8947368
```

Comparing our base models over the test set, it appears that multinomial logistic regression has surpassed all the other models with respect to accuracy and kappa of ~91% and ~90%.

For other metrics, more importantly, if we check the class `t-CS-m` & `t-CS-s` indicating Rescued learning and Failed learning, we observe again that precision, recall, and F1-score of MGR model is beating other model values.

Although, when we compare MGR to Bagging, we see altogether different results. It seems logical as bagging being an ensemble tree model which fits much better to classification data.

Furthermore, we can also firmly say that our ***Stacked Ensemble Super learner*** has performed sub-par from all the other models in all metrics and even beating the built-in ensemble model `-bagging`.

ROC curve

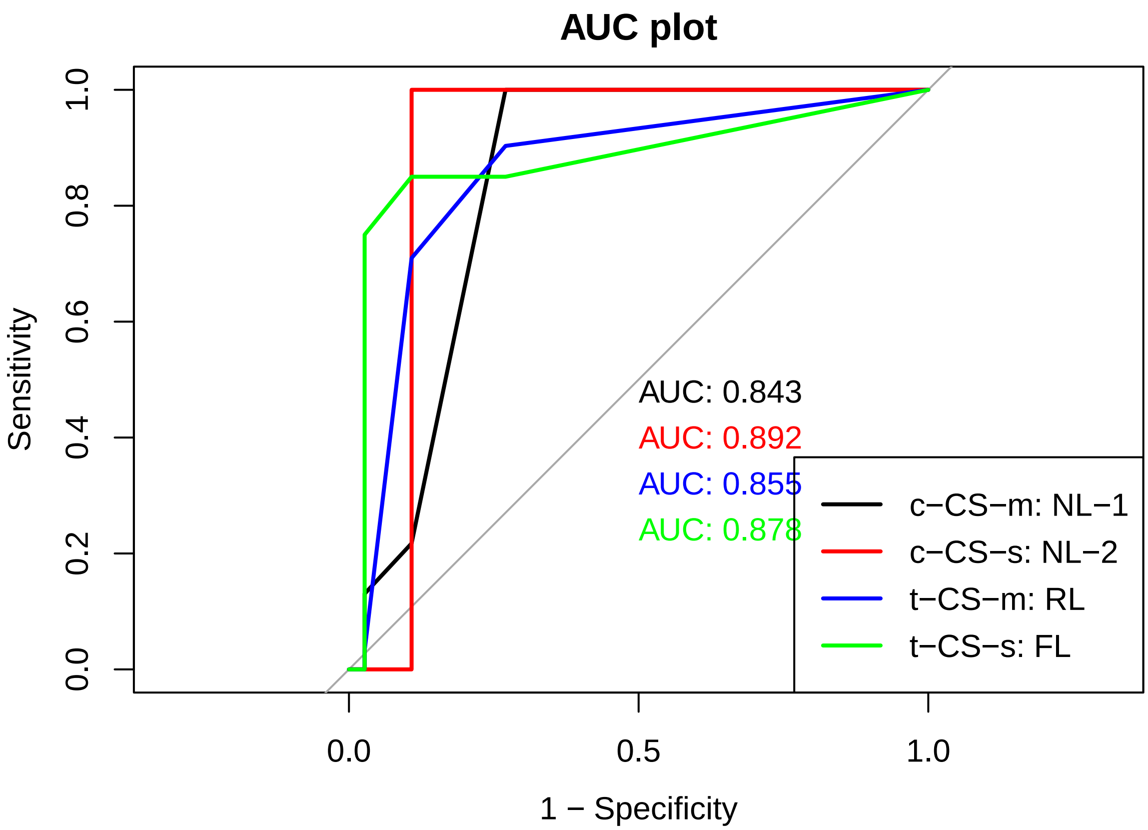
As our first step for evaluation, we plot ROC curve along the respective AUC values for the target sub-classes that define three major learning outcomes i.e. normal learning, rescued learning, failed learning

```
# plotting the classes that are significant to our learning outcomes
# normal learning: "c-CS-m"
rocModels <- function(pred, test){
  roc.pred <- as.ordered(pred)
  # get roc results for every class label
  res<- pROC::multiclass.roc(test$Class, roc.pred, quiet = T,
                             levels = c("c-CS-m", "c-CS-s", "c-SC-m", "c-SC-s",
                                        "t-CS-m", "t-CS-s", "t-SC-m", "t-SC-s"))

  plot.roc(res$rocs[[1]],
           print.auc=T,
           legacy.axes = T, main="AUC plot")
  # normal learning: "c-CS-s"
  plot.roc(res$rocs[[2]],
           add=T, col = 'red',
           print.auc = T,
           legacy.axes = T,
           print.auc.adj = c(0,3))
  # rescued learning: "t-CS-m"
  plot.roc(res$rocs[[4]],add=T, col = 'blue',
           print.auc=T,
           legacy.axes = T,
           print.auc.adj = c(0,5))
  # failed learning: "t-CS-s"
  plot.roc(res$rocs[[5]],
           add=T, col = 'green',
           print.auc = T,
           legacy.axes = T,
           print.auc.adj = c(0,7))
  legend('bottomright',
         legend = c('c-CS-m: NL-1','c-CS-s: NL-2',
                    't-CS-m: RL', 't-CS-s: FL'),
```

```r
         col=c('black','red','blue', 'green'),lwd=2)

}
# Plot AUC plots for all models
# model-1: Naive Bayes
rocModels(pred=NB.pred.test, test=testX)
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
```

```
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

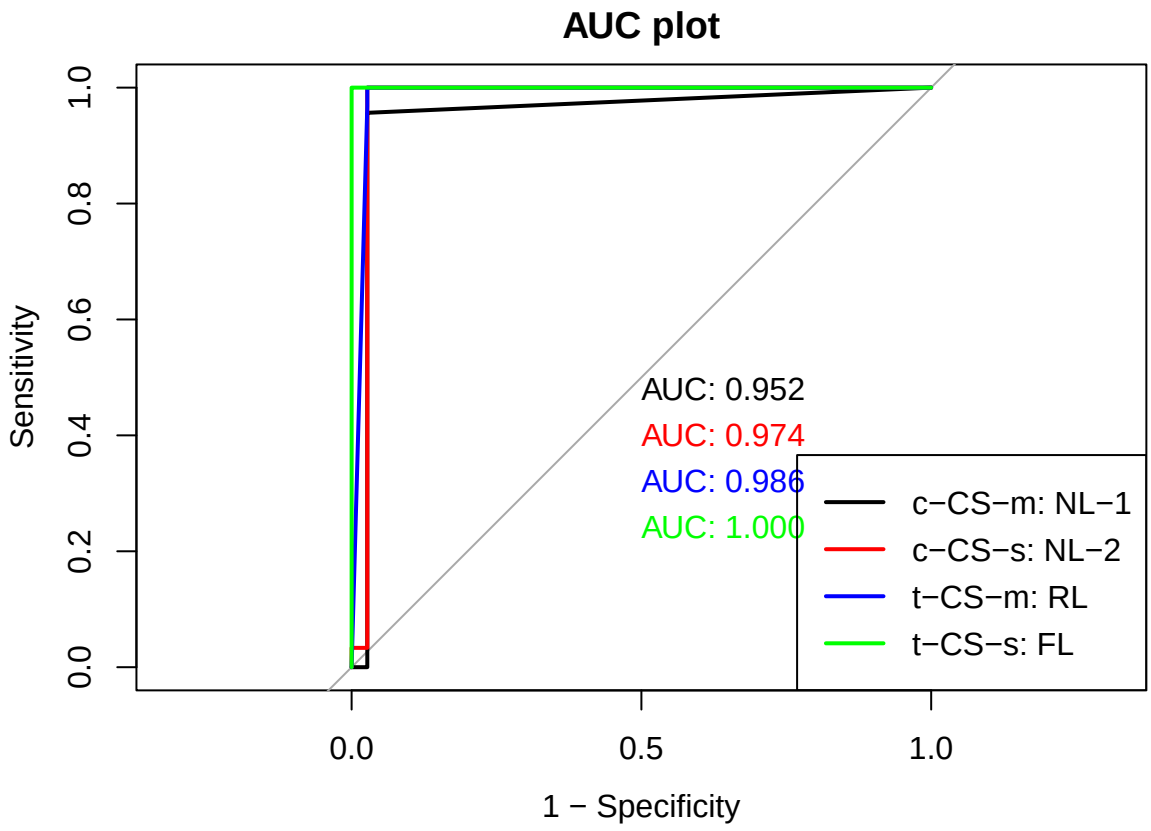



```
# model-2: Artificial Neural Network
rocModels(pred=unlist(NN.tablePred[1]), test = testX)
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

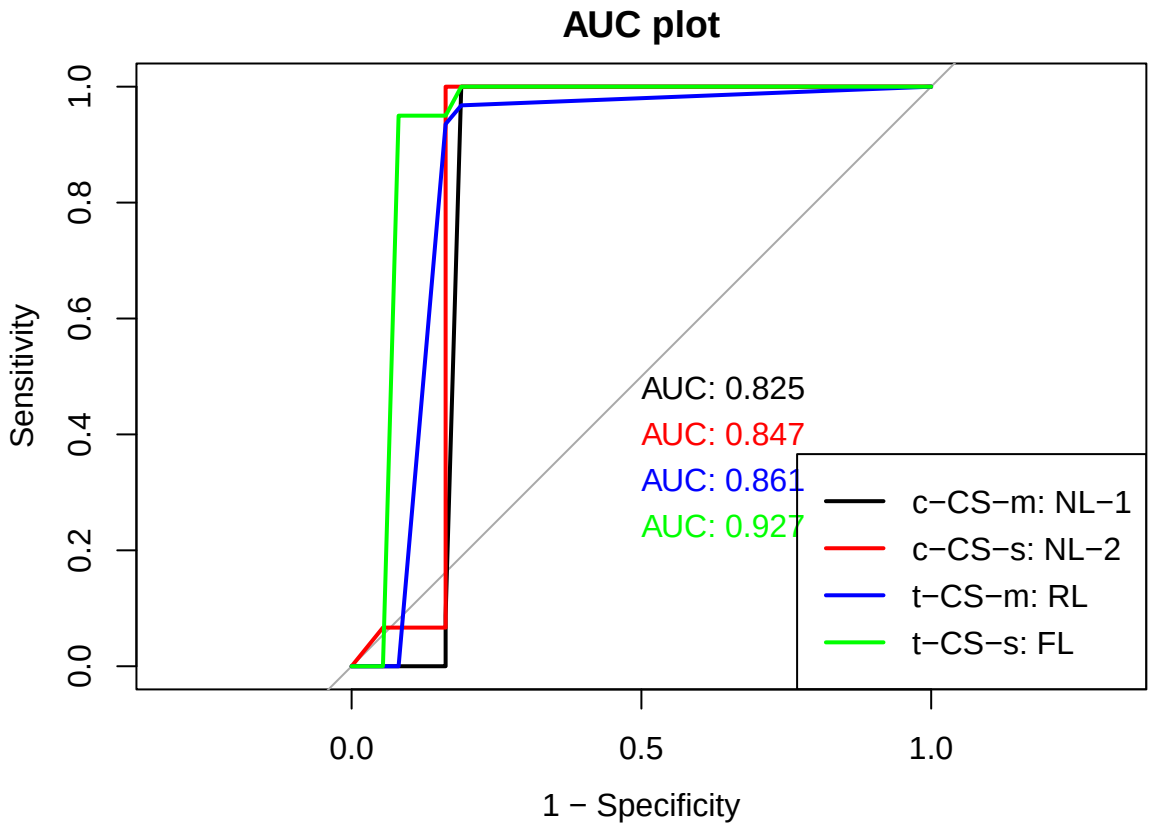



```
# model-3: Multi-nomial Logistic Regression
rocModels(pred=MGR.pred.test, test = testX)
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

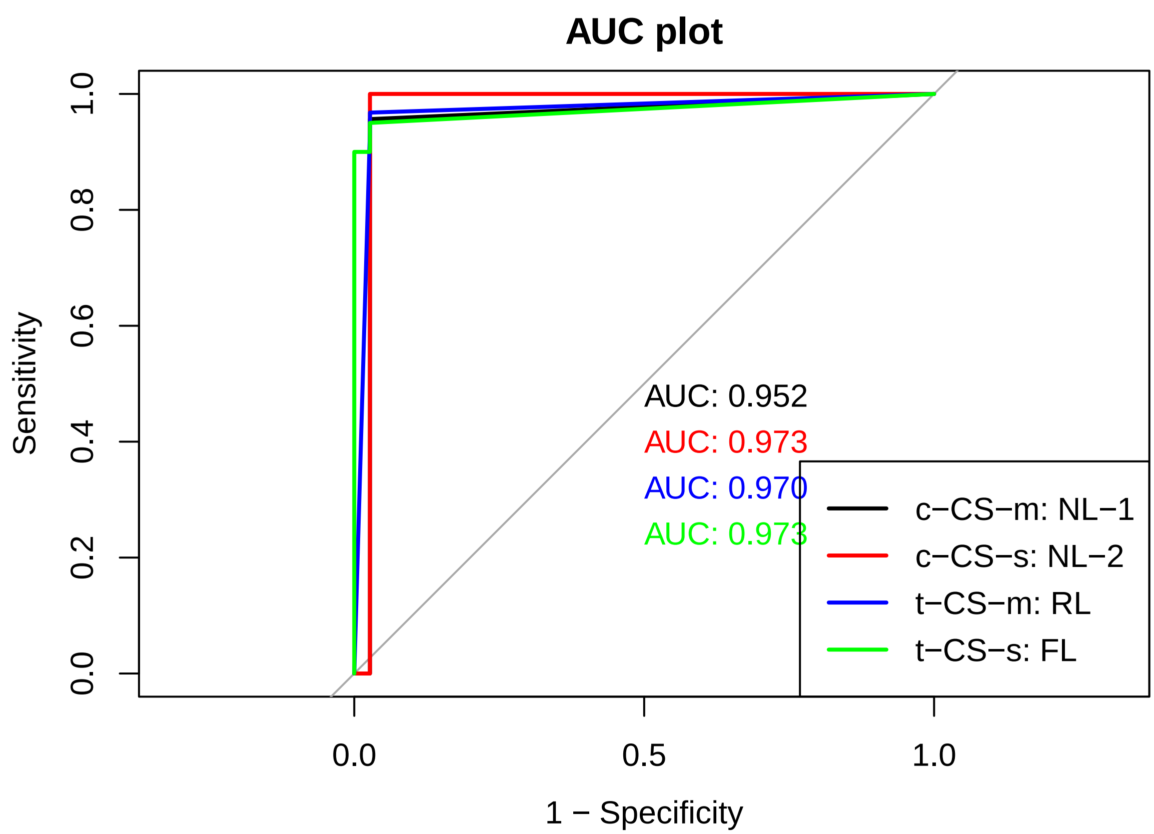



```
# model-4: Stacked Ensemble Learner
rocModels(pred=unlist(stackResults[2]), test = testX)
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
```

```
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
```

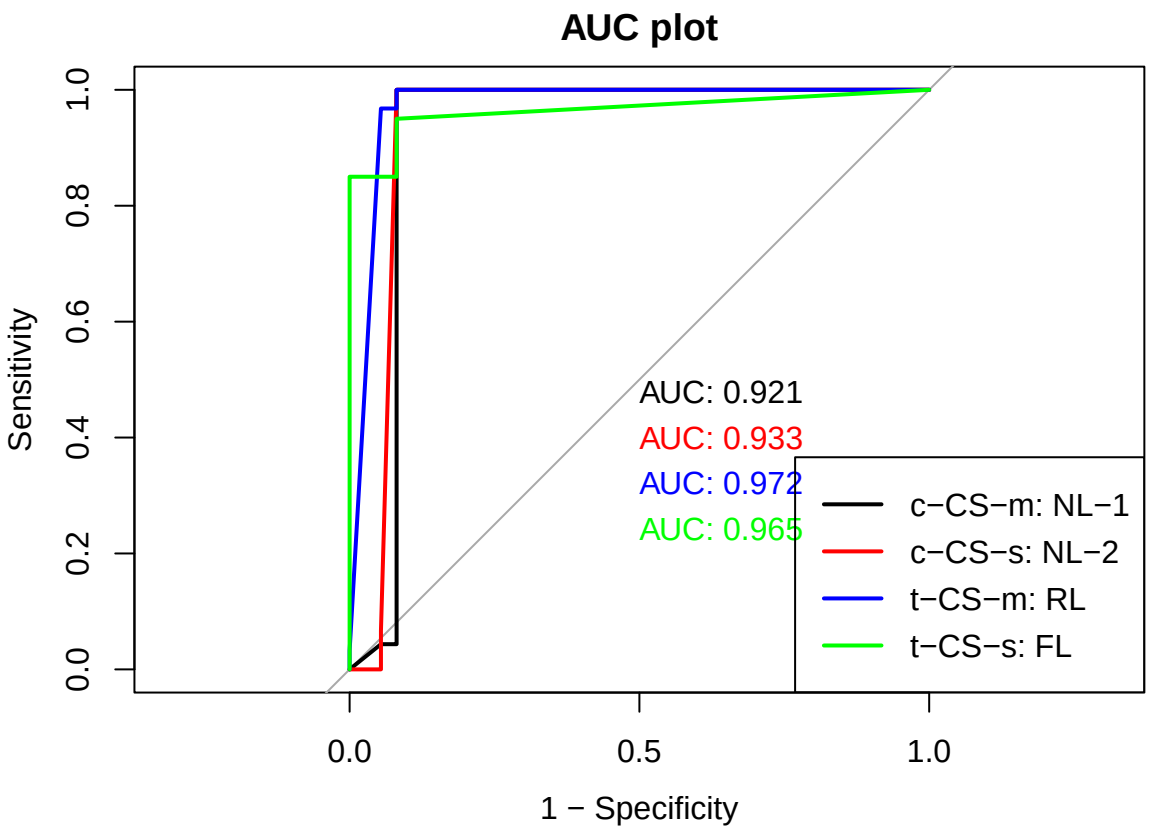
```
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```




```
# model-5: Bagging
rocModels(pred=bag.pred, test = testX)
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```

```
## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.

## Warning in value[[3L]](cond): Ordered predictor converted to numeric vector.
## Threshold values will not correspond to values in predictor.
```


AUC plot
Sensitivity
1 − Specificity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
AUC: 0.921
AUC: 0.933
AUC: 0.972
AUC: 0.965
c−CS−m: NL−1
c−CS−s: NL−2
t−CS−m: RL
t−CS−s: FL