Introduction to DrugScreenExplorer

Junyan Lu

28 July 2018

Contents

1 Getting started

DrugScreenExplorer is an R package designed for streamlined processing, quality control and visualizing of high-throughput drug sensitivity screen data.

```
library(DrugScreenExplorer)
library(tidyverse)
library(gridExtra)
library(tools)
```

2 Data import

2.1 Introduction of input file types

For creating the complete drug screen datasets, three types of data are needed:

1) the raw screening data in text format, which stores the values out from a plate reader;
2) the well annotation or plate layout, which stores the metadata for all the wells on a screening plate. The most important annotations are drug names and concentrations;
3) the plate annotation, which stores the metadata for each plate. For example, the sample information, if each plate represents one sample.

The examples for each type of the input files can be found in the test data affiliated with this package:

Raw screening data

```
system.file("testData/rawData", package = "DrugScreenExplorer")
## [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/DrugScreenExplorer/testData/rawData"
```

If the text files containing the raw screening data are organized by subfolders, the subfolders will be considered as batches.

Plate annotation file

```
system.file("testData/plateAnno/plateAnno.csv", package = "DrugScreenExplorer")
## [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/DrugScreenExplorer/testData/plateAnno/
```

The *fileName* column is mandatory. If the plates do not all have the same layout, for example, several plates are used for the same sample but with different drug collections, a *plateID* column can be included to indicate the plate index.

Well annotation file

```
system.file("testData/wellAnno/wellAnno.csv", package = "DrugScreenExplorer")
## [1] "/Library/Frameworks/R.framework/Versions/3.5/Resources/library/DrugScreenExplorer/testData/wellAnno/
```

The *wellID* column is mandatory. If plates have different layouts, a *plateID* column can be used as in the plate annotation file.

2.2 Functions for helping create annotation files

Besides create the well and plate annotations files manually, there are two functions can be used to help create the annotation input files.

2.2.1 createPlateInput()

The function creates the framework for the plate annotation input based on the raw data files and folder structure in the raw data folder.

```
rawFolder <- system.file("testData/rawData", package = "DrugScreenExplorer")
```

```
createPlateInput(rawDir = rawFolder,
                 file = "plateAnno.csv",
                 entries = c("sampleID", "patientID"))
```

An csv file names “plateAnno.csv” will be created and it has three columns *fileName*, *sampleID* and *patientID*. The *fileName* column is automatically filled based on the files in the raw data folder, while *sampleID* and *patientID* columns need to be filled manually.

If the text files containing the raw data are grouped in sub-folders, and you want the sub-folders to be considered as batches. The createPlateInput function can also generate the batch information automatically if the *batchAsFolder* is TRUE.

```
rawFolder <- system.file("testData/rawData", package = "DrugScreenExplorer")
createPlateInput(rawDir = rawFolder,
                 file = "plateAnno.csv",
                 entries = c("sampleID", "patientID"),
                 batchAsFolder = TRUE)
```

In the output *plateAnno.csv* file, there’s an additional column *batch*, which contains the batch information. Notice that the folder names are not used as batch names. Instead, the series of integer numbers will be assigned to batches based on the ordering of the folder names.

2.2.2 createWellInput()

The function creates the framework for the well annotation input.

```
createWellInput(file = "wellAnno.csv", colNum = 24, rowNum = 16,
                entries = c("name", "concentration"))
```

A *wellAnno.csv* file will be created for a 384-well (16*24) plate. The *wellID* is automatically filled by the *name* and *concentration* columns need to be filled manual.

If more than one plate layouts for a sample are used in the screen, for example, when one plate is not enough for testing all the drugs, the *platePerSample* parameter can be used to specify how many different plate layouts are used for each sample.

```
createWellInput(file = "wellAnno.csv", colNum = 24, rowNum = 16,
                entries = c("name", "concentration"),
                platePerSample = 2)
```

In the *wellAnno.csv* file, an additional column *plateID* will be created, indicating the index of the plate layout type. It is important that a *plateID* column must also be created manually in the plate annotation file, to specify which plate layout a certain plate is using.

2.3 Read in the whole experiment

After preparing the raw data folder, the plate annotation file and the well annotation file, the whole experiment can be read in using the `readScreen()` function.

```
plateFile <- system.file("testData/plateAnno/plateAnno.csv",
                         package = "DrugScreenExplorer")
wellFile <- system.file("testData/wellAnno/wellAnno.csv",
                         package = "DrugScreenExplorer")
screenData <- readScreen(rawDir = rawFolder, plateAnnotationFile = plateFile,
                         wellAnnotationFile = wellFile,
                         rowRange = c(3,18), colRange = 2,
                         sep = "[;\t]",
                         negWell <- c("DMSO","PBS"), posWell = c())
head(screenData)
## # A tibble: 6 x 11
##   wellID rowID colID  value fileName batch sampleID patientID name
##   <fct>  <fct> <fct>  <dbl> <fct>    <fct> <fct>    <fct>     <fct>
## 1 A001   A0    01    1.08e6 CTGLumi~ 1     11PB0010 P0010     DMSO
## 2 A002   A0    02    8.56e5 CTGLumi~ 1     11PB0010 P0010     DMSO
## 3 A003   A0    03    1.01e6 CTGLumi~ 1     11PB0010 P0010     A-12~
## 4 A004   A0    04    2.10e4 CTGLumi~ 1     11PB0010 P0010     Afat~
## 5 A005   A0    05    1.15e6 CTGLumi~ 1     11PB0010 P0010     A-12~
## 6 A006   A0    06    5.58e5 CTGLumi~ 1     11PB0010 P0010     Afat~
## # ... with 2 more variables: concentration <dbl>, wellType <fct>
```

A tidy table containing the drug screening result and metadata will be created.
Because each platform can generate different raw data format, the *rowRange* and *colRange* function can tell the function the staring and ending place of the numeric data that represent the signal. Different delimiters can be specified by *sep* and regular expression is supported. The postie control and negative control wells can be specified by *negWell* and *posWell*. The input for *negWell* or *posWell* can be either a character vector of *wellIDs* or drug names if the *name* column is given in the well annotation file. Wells that are not labeled as positive controls ("pos") or negative controls ("neg") will be labelled as "drug" in the "wellType" column in the final output table. This column is important for plate-wise normalization.

The inhibition rate for each well can also be calculated at this stage by specifying "normalization = TRUE". Currently, two methods are supported: "negatives", which calculates the inhibition rate by simple dividing each measurement on the plate by the median of the values of negative control wells on the same plate; "both", which uses the "normalized percent inhibition (NPI)" is used. For an inhibition assay, this method divides the difference between each measurement on a plate and the median measurement of the positive controls on that plate by the difference between the median measurement of negative controls and positive controls on that well.

```
screenData <- readScreen(rawDir = rawFolder, plateAnnotationFile = plateFile,
                         wellAnnotationFile = wellFile,
                         rowRange = c(3,18), colRange = 2,
                         sep = "[;\t]",
                         negWell <- c("DMSO","PBS"), posWell = c(),
                         normalization = TRUE, method = "negatives", discardLayer = 2)
head(screenData %>% select(wellID, value, sampleID, name, concentration, wellType, normVal))
## # A tibble: 6 x 7
##   wellID   value sampleID name      concentration wellType normVal
##   <fct>    <dbl> <fct>    <fct>             <dbl> <fct>      <dbl>
## 1 A001   1083480 11PB0010 DMSO                0.3 neg       1.13
## 2 A002    856480 11PB0010 DMSO                0.3 neg       0.892
## 3 A003   1014920 11PB0010 A-1210477          10   sample    1.06
## 4 A004     21000 11PB0010 Afatinib           10   sample    0.0219
## 5 A005   1151520 11PB0010 A-1210477           2   sample    1.20
## 6 A006    557760 11PB0010 Afatinib            2   sample    0.581
```

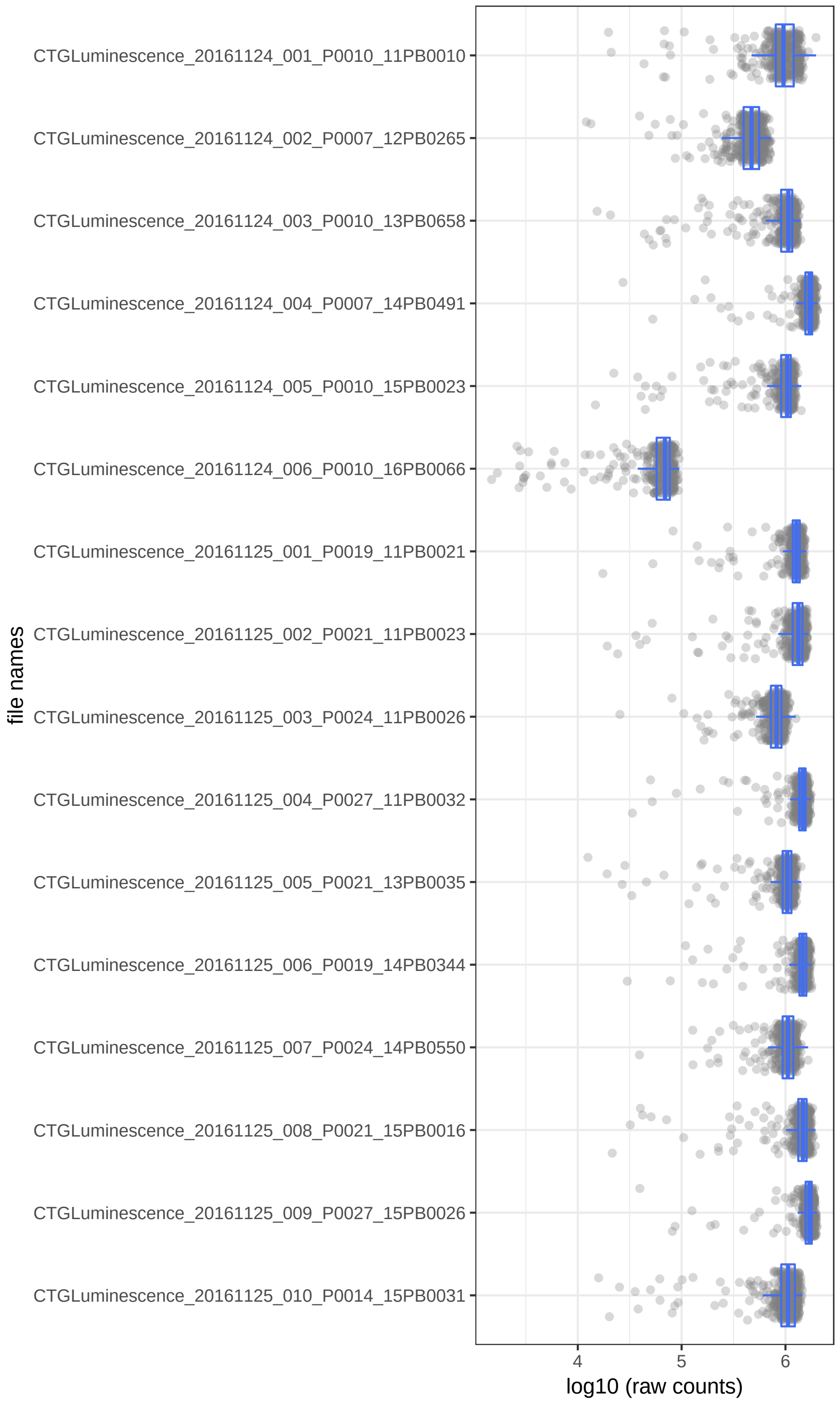
A new column, "normVal", is created, which contains the normalized values. Inhibition rate can also be calculated later by using *normalizePlate()* function.

3 Quality control

3.1 Raw count distribution

The DrugScreenExplorer package provides several functions for checking the screen plot. First, we can use the *plotRawCount* function to plot the distribution of raw count (signal) on each plate, to check whether a certain plate has a strange behavior.

```
g <- plotRawCount(screenData, ifLog10=TRUE)
plot(g)
```

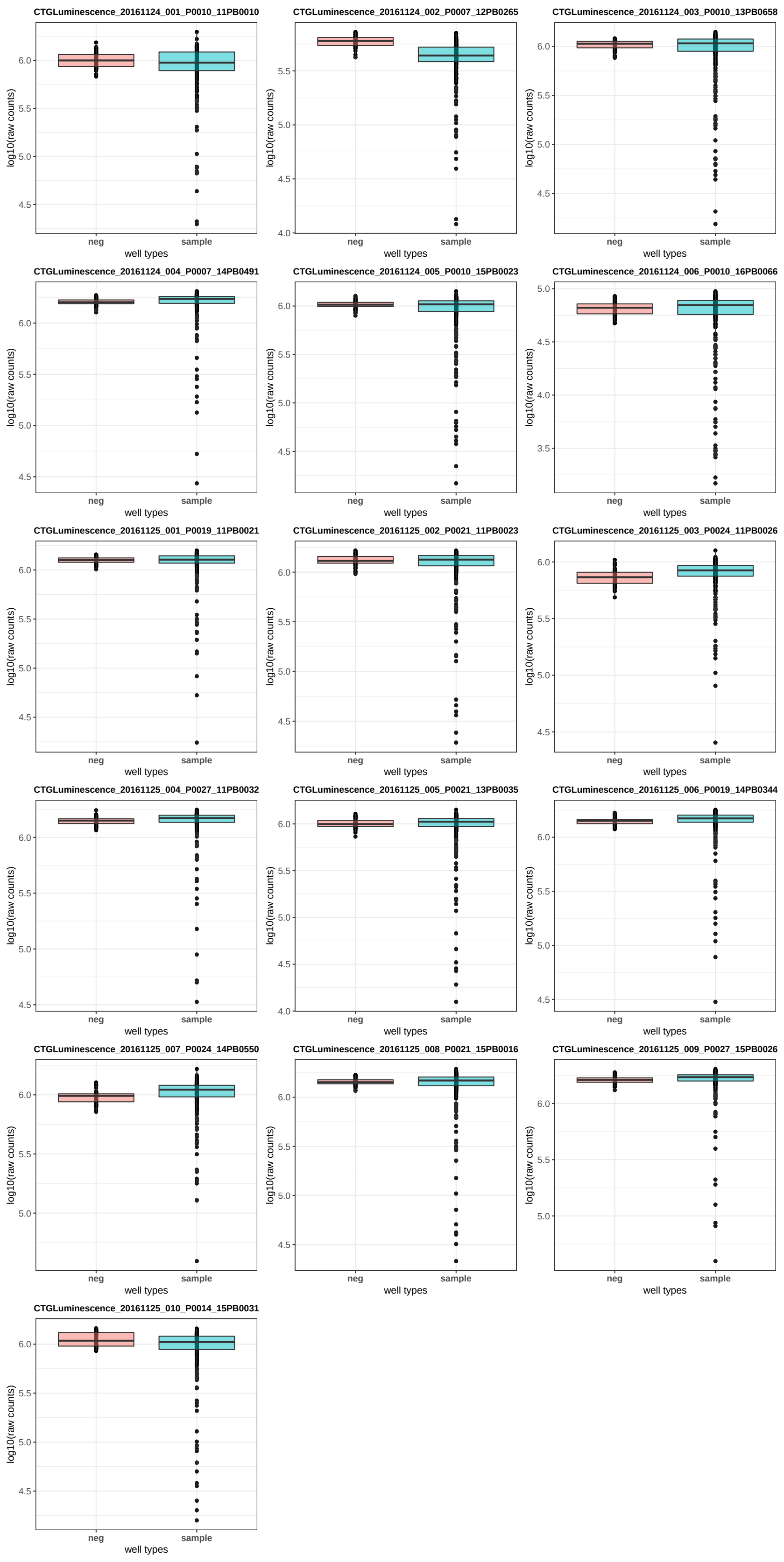


We can notice that the plate "CTGLuminescence_20161124_006_P0010_16PB0066" has a rather overall low counts that other plates.

3.2 Raw count distribution for each well type

If the plate layout information has been provided in the "wellType" column, the raw count distribution for each well type on each plate can also be plotted to check whether the raw count distribution for each well type is reasonable, for example, the raw count for negative control wells should be generally higher than sample wells.

```
g <- plotTypeDist(screenData, ifLog10 = TRUE)
grid.arrange(grobs = g, ncol=3)
```

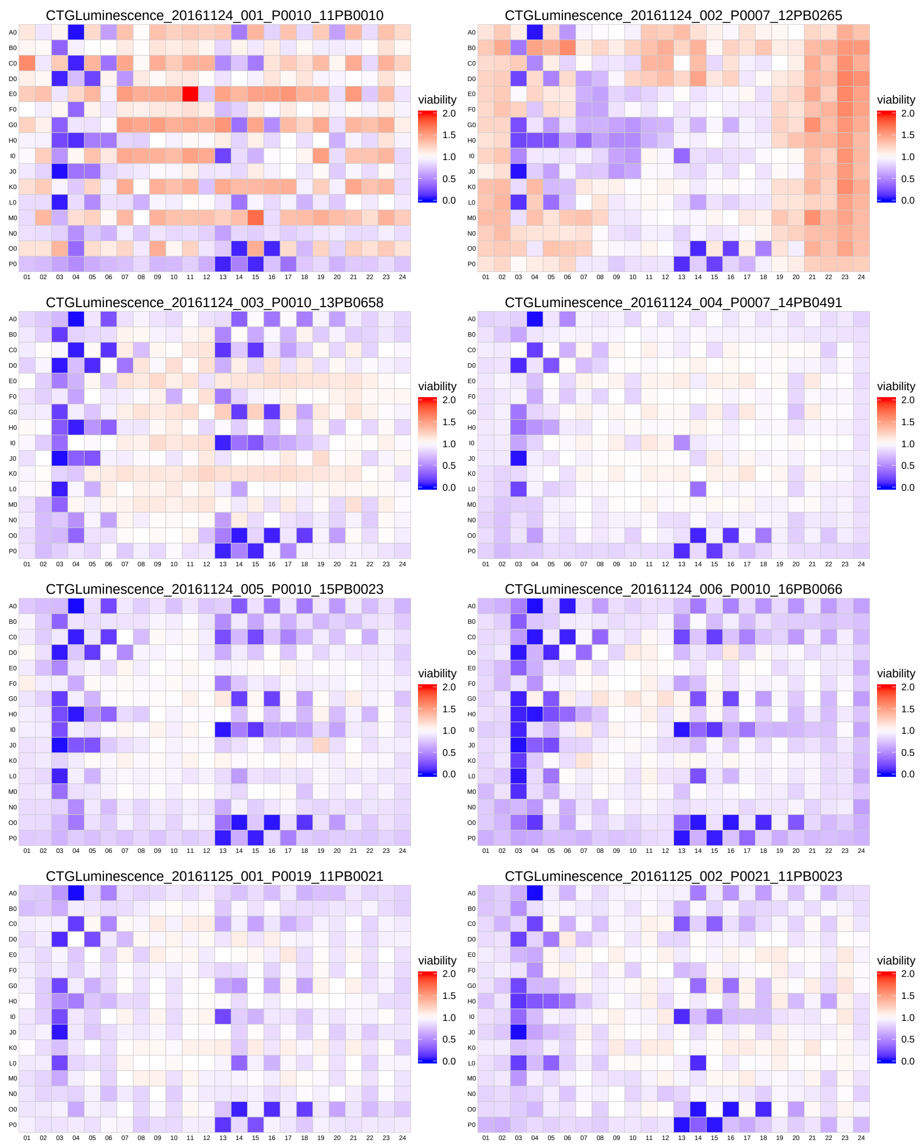

CTGLuminescence_20161124_001_P0010_11PB0010
CTGLuminescence_20161124_002_P0007_12PB0265
CTGLuminescence_20161124_003_P0010_13PB0658
CTGLuminescence_20161124_004_P0007_14PB0491
CTGLuminescence_20161124_005_P0010_15PB0023
CTGLuminescence_20161124_006_P0010_16PB0066
CTGLuminescence_20161125_001_P0019_11PB0021
CTGLuminescence_20161125_002_P0021_11PB0023
CTGLuminescence_20161125_003_P0024_11PB0026
CTGLuminescence_20161125_004_P0027_11PB0032
CTGLuminescence_20161125_005_P0021_13PB0035
CTGLuminescence_20161125_006_P0019_14PB0344
CTGLuminescence_20161125_007_P0024_14PB0550
CTGLuminescence_20161125_008_P0021_15PB0016
CTGLuminescence_20161125_009_P0027_15PB0026
CTGLuminescence_20161125_010_P0014_15PB0031
log10(raw counts)
well types
neg
sample

The *plotTypeDist* function can also output a pdf file by specifying the *pdfName* parameter.

3.3 Plate heatmap plot

A direct way to examine the screen quality is to plot the signal intensity heatmap for each screen. The platePlot function can be used to plot the heatmaps. From plate heatmap plots, screen artifacts, such as incubation effect, can be spotted.

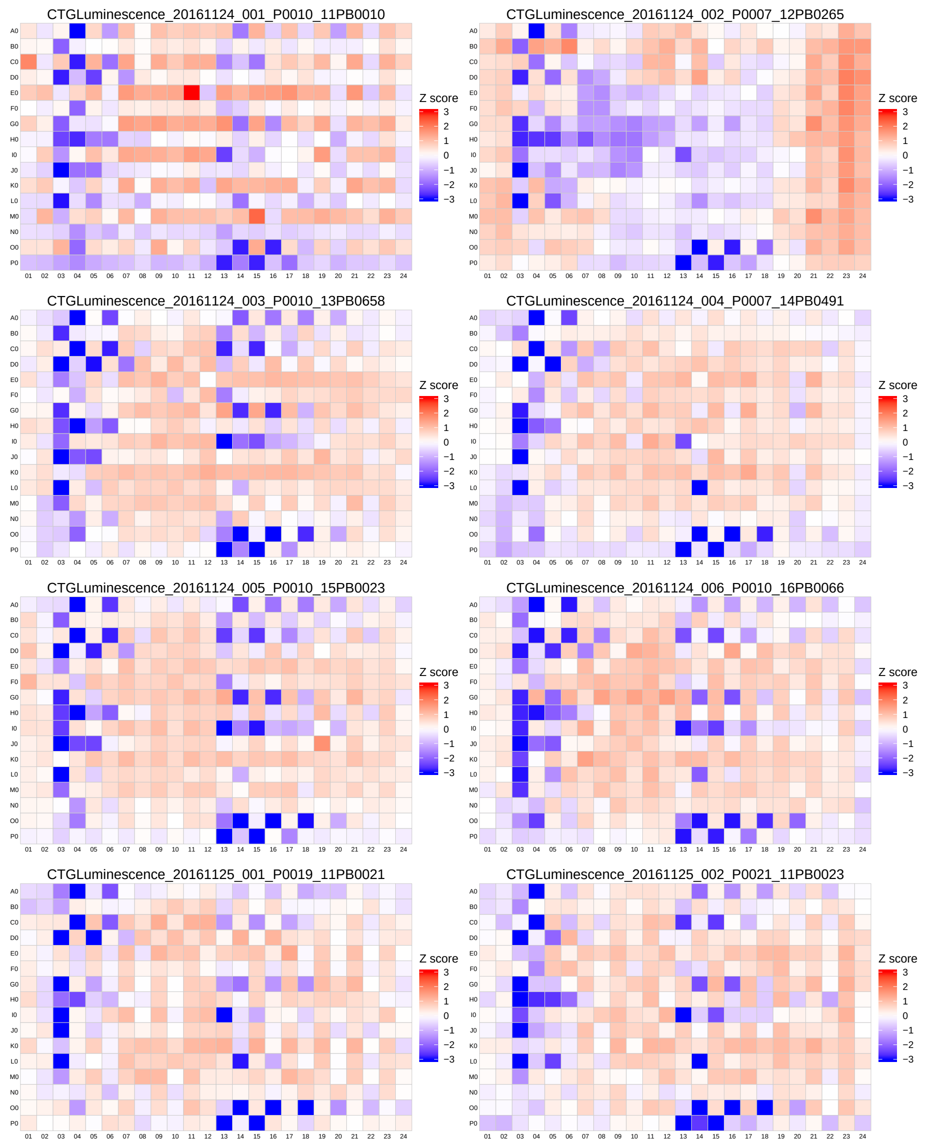
```r
g <- platePlot(screenData = screenData, plotPlate = "all", plotType = "viability")
grid.arrange(grobs = g[1:8], ncol = 2)
```



If *plotType* = *"viability"*, the percent inhibition rate relative to negative controls will be used for the heatmap and the screen data must be normalized before hand, i.e the *normVal* column must be in the *screenData* object.

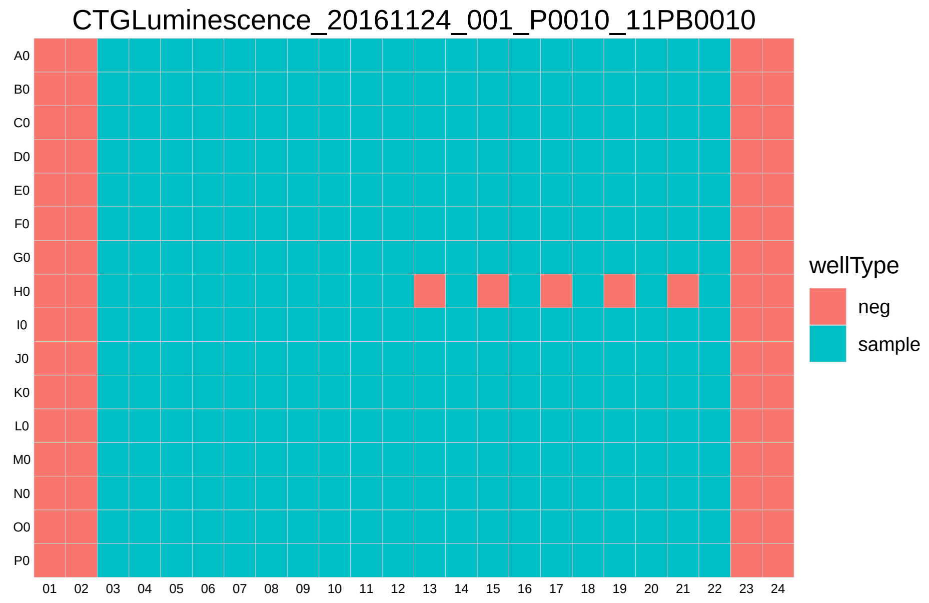
For unnormalized values, per-plate z-score can be used for heatmap plot.

```r
g <- platePlot(screenData = screenData, plotPlate = "all", plotType = "zscore")
grid.arrange(grobs = g[1:8], ncol = 2)
```



The *platePlot* function can also be used to plot the plate layout, by specifying the *type* parameter as “layout”

```r
g <- platePlot(screenData = screenData, plotPlate = "all", plotType = "layout")
g[[1]]
```



3.4 Automatic screen quality report

An html Rmarkdown report on the screen quality can be automatically generated by using the *makeReport* function. Depend on the annotations included in the screen data tidy table, different quality check items will be included in the report.

```
makeReport(screenData = screenData, showCode = FALSE,
           title = "Report for my drug screening project",
           author = "Junyan Lu", ifPlatePlot = TRUE)
```

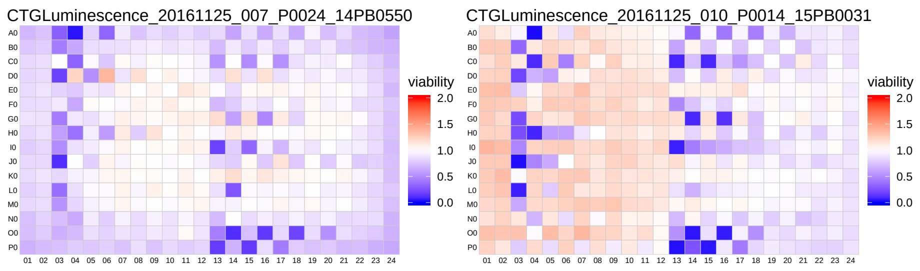
An html file will be created in *report* folder under the working directory.
The information on more customizable options can be found at the manual page for *makeReport*

4 Adjusting incubation effect

From the plate heatmap plots above, we can notice some obvious artifacts on the screening plates. For example, some wells in a certain region of a plate show significantly higher or lower signal intensity. This is know as incubation effect, because usually those artifacts are caused by technical problems, like solvent evaporation, pipetting issues, and etc.,. If those problematic wells are only located on the edge of the plate, it is also called edge effect.

Plates show typical incubation effect and edge effect.

```
plotPlates <- c("CTGLuminescence_20161125_007_P0024_14PB0550",
                "CTGLuminescence_20161125_010_P0014_15PB0031")
subScreen <- filter(screenData, fileName %in% plotPlates)
g <- platePlot(subScreen, plotPlate = "all", plotType = "viability")
grid.arrange(grobs=g, ncol=2)
```



The plate on the left shows typical edge effect, because the wells on the edge of the plate generally show lower values than the well in the middle. On the other hand, the plate on the right show incubation effect, in which left half of the plate shows higher value than the left half.

DrugScreenExplorer package provides two functions *fitEdgeEffect* and *correctEdgeEffect* for estimating and correcting incubation effect or edge effect on the plate.

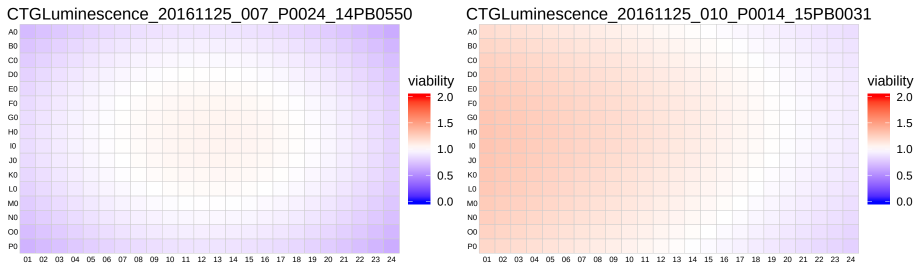
Firstly, the incubation effect can be estimated by using a local regression fit method.

```
subScreen <- fitEdgeEffect(subScreen, method = "loess", useNeg = TRUE,
                           useLowConcentrations = 1, span = 1)
```

The estimated incubation effect will be stored in the *edgeFactor* column.
useNeg = *TRUE* means only negative control wells are used for incubation effect estimation.
useLowConcentrations = *1* means the lowest 1 concentration of the drugs is also considered as the *de facto* negative controls and used for incubation effect estimation.
Please refer to the manual page for more explanations of the parameters.

The incubation (edge) effect can be visualized using the *platePlot* function by specifying *plotType* = *edgeEffect*

```
g <- platePlot(subScreen, plotType = "edgeEffect")
grid.arrange(grobs = g, ncol =2)
```



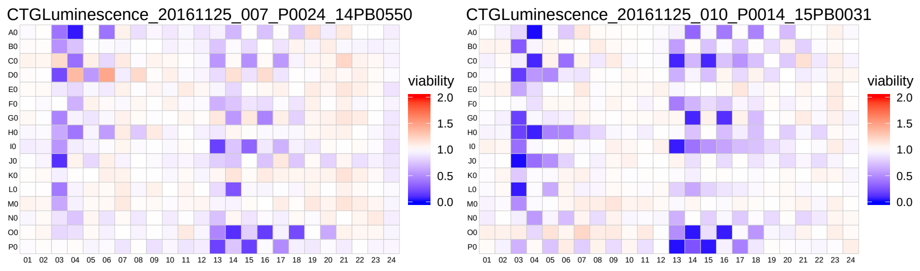
Next, the incubation effect can be removed from the screen plate by *correctEdgeEffect* function.

```
subScreen <- correctEdgeEffect(subScreen, correctMethod = "bliss")
```

In this example, the edge effect will be removed by Bliss drug combination model, in which the real effect equal observed effect divided by edge factor. The corrected value is stored in a new column *normVal.cor*
Currently, the estimation and correction of incubation effect can only be performed on normalized values

The corrected values can also be visualized by *platePlot* function, by specifying *plotType* = *"viability"* and *ifCorrected* = *TRUE*

```
g <- platePlot(subScreen, plotType = "viability", ifCorrected = TRUE)
grid.arrange(grobs = g, ncol =2)
```



Now the incubation effect has been largely removed.

5 Interactive data exploration using Shiny app

Prepare data set for shiny app

```
screenData.corr <- correctEdgeEffect(screenData, correctMethod = "bliss")
makeShiny(screenData.corr, sampleAnnotations = c("batch","sampleID","patientID"))
```

The functions avaialbe for the shiny app will be dependent on the sample annotations provided by the user.

Run shiny app

The shiny app can be excuted through running “app.R” under “shiny” folder.