Kernel: R (system-wide)

Lab 1b: GenVisR

GenVisR is a Bioconductor package that is used for visualizing genomic data (small variants, copy number alterations and data quality) and is especially usefully when looking at multiple species of interest.

Install the necessary packages

- Required: `BiocManager` , `GenVisR` , `gridExtra` , `grid` , `reshape2`
- Change the cell below to a code cell and run it to install the packages.
    - **IMPORTANT**: You only need to run the cell below once.

```
# This prelimiary step is only necessary for the CoCalc system
dir.create(path = Sys.getenv("R_LIBS_USER"), showWarnings = FALSE, recursive = TRUE)
lib_path <- Sys.getenv("R_LIBS_USER")

# Now, we actually install the packages
# source: https://bioconductor.org/packages/release/bioc/html/GenVisR.html
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager", lib = lib_path)
BiocManager::install("GenVisR", lib = lib_path)
libraries <- c("gridExtra", "reshape2")
for (i in libraries){
    install.packages(i, lib = lib_path)
}
```

```
# In Ex 1a, we installed the packages in the folder path lib_path; so we
# explicitly state which folder to locate GenVisR using lib.loc when loading
# the library.
lib_path <- Sys.getenv("R_LIBS_USER")
library(gridExtra)
library(grid)
library(reshape2)
library(dplyr)
library(GenVisR, lib.loc = lib_path)
library(data.table)
```

Load GenVisR

1. Load the “GenVisR” and “data.table” libraries, if they are not installed (see Support Protocol 1). Note that data.table is a GenVisR dependency and will be installed automatically when installing GenVisR.

In [0]:
```
library(gridExtra)
library(grid)
library(reshape2)
library(dplyr)
library(GenVisR)
library(data.table)
```

Waterfall plot

The `waterfall()` function helps us:

- Visualize the types of mutations within a cohort
- Visualize the mutation burden in a data set to determine if the mutation burden conforms to expectations
- Visualize the proportion of samples with a mutated gene
- Find patterns within clinical data
- Determine mutually exclusive or co-occurring genomic events

Interpretation:

- Main panel: the mutation occurrence and type

- Top and side panels: the mutation burden and the percentage of samples with a mutation

Input file types:

- custom: `data.frame` or `data.table` containing the column names `sample`, `gene`, `variant_class`
- VEP: Ensembl Variant Effect Predictor
- MAF: (McLaren et al., 2016)

1. Import the mutation data

We will be using a sample dataset containing variant calls from a Phase1 clinical trial. Note that the imported tsv file is a data.frame object.

In [0]:
```
mutation_df <- read.delim("data/BKM120_Mutation_Data.tsv", sep="\t")
mutation_df[1:3,]
```

`Waterfall()` looks for specific column names within the data.table: e.g. `sample`, `gene`, `mutation`, and `amino acid change` (not required, but we will use it later)

Let's take a look at the column names first.

In [0]:
```
colnames(mutation_df)
```

Then we rename the columns.

In [0]:
```
mutation_df <- mutation_df %>% rename( sample = patient,
                                       gene = gene.name,
                                       mutation = trv.type,
                                       amino.acid.change = amino.acid.change)
colnames(mutation_df)
```

Note: The backticks above are necessary because of the spaces in our column names.

2. Define the mutation hierarchy

When there are **multiple mutations** for the **same gene or sample**, we need to prioritize the most deleterious mutation type using the mutation hierarchy. The **mutation hierarchy** is a `data.table`, where the most significant mutation type comes first. Conflicts arising from multiple mutations in the same gene/sample cell are resolved by a hierarchical removal of mutations defined by the mutation hierarchy.

In [0]:
```
# Here are the standard mutation hierarchies for each input data type
# (check your input data to determine the most applicable option)
MGI <- c("nonsense", "frame_shift_del",
         "frame_shift_ins", "splice_site_del",
         "splice_site_ins", "splice_site",
         "nonstop", "in_frame_del", "in_frame_ins",
         "missense", "splice_region_del",
         "splice_region_ins", "splice_region",
         "5_prime_flanking_region",
         "3_prime_flanking_region",
         "3_prime_untranslated_region",
         "5_prime_untranslated_region", "rna",
         "intronic", "silent")
MAF <- c("Nonsense_Mutation", "Frame_Shift_Ins",
         "Frame_Shift_Del", "Translation_Start_Site",
         "Splice_Site", "Nonstop_Mutation",
         "In_Frame_Ins", "In_Frame_Del",
         "Missense_Mutation", "5\'Flank",
         "3\'Flank", "5\'UTR", "3\'UTR", "RNA", "Intron",
         "IGR", "Silent", "Targeted_Region", "", "")
```

In [0]:
```
mut_Hierarchy <- data.table(
    "mutation"=c("nonsense", "frame_shift_del", "frame_shift_ins", "in_frame_del", "splice_site_del",
"splice_site", "missense", "splice_region", "rna"),
    # define the colours for each mutation type
    color=c("#FF0000", "#00A08A", "#F2AD00", "#F98400", "#5BBCD6", "#046C9A", "#D69C4E", "#000000", "#446455")
)
```

Store `Waterfall()` in as `plotData`. The output includes graphical objects and the underlying data from which they were constructed as well.

In [0]:
```
# (optional) You may modify the settings here to adjust the figure dimensions
options(repr.plot.width = 10, repr.plot.height = 6)
```

In [0]:
```
plotData <- Waterfall(mutation_df , mutationHierarchy = mut_Hierarchy)
drawPlot(plotData)
```

Note: The Translational Effect legend for the top panel only has the category "NA" because our sample dataset doesn't have silent mutations.

---

3. Specify the cutoff threshold (conditionally applicable)

When there is a large number of genes, you can reduce the number of cells in the plot by limiting the number of genes plotted.

`recurrence` removes genes from the data which do not have at least x proportion of samples mutated (where x is a numeric value between 0 and 1).

Here is an example plotting only genes with mutations in **6% or more** of samples.

In [0]:
```
# (optional) You may modify the settings here to adjust the figure dimensions
options(repr.plot.width = 10, repr.plot.height = 6)
```

In [0]:
```
plotData_filter <- Waterfall(mutation_df, mutationHierarchy = mut_Hierarchy, recurrence = 0.06)
drawPlot(plotData_filter)
```

4. Export the plot

Next, we save this plot to a PDF device for viewing and/or publication.

In [0]:
```
# open a PDF device
pdf(file="Figure_1.pdf", height=8, width=12)

# output the plot
drawPlot(plotData_filter)

# close the graphics device
dev.off()
```

Saving to a PDF is optional. In addition to `pdf()`, you can alse use `png()`, `jpeg()`, `tiff()`, and `bmp()` functions to export the plot as other file types.

Exercise 1a: Plot genes with mutations in 10% or more samples (1 pt)

Instructions: In the code cell below, use the `waterfall()` function to plot the genes with mutations in 10% or more samples.

Input parameters:

- Mutation data: `mutation_df`
- Mutation hierarchy: `mut_Hierarchy`

Ouput:

- Store the `waterfall()` object in the variable `ex_1a`
- Use `drawPlot()` to display the plot.

In [0]:
```
# Enter your code here
```

Exercise 1b: Interpretation of the plot from Exercise 1a (1 pt)

In the markdown cell below, answer the following question in 1-2 sentences:

- What are the difference(s) between the waterfall plots from **Exercise 1a** and `plotData` from earlier that included all of the genes?

Note: Your answer should go beyond discussing the difference in function parameters.

WRITE YOUR ANSWER HERE

WRITE YOUR ANSWER HERE

Clinical Data in Waterfall plot

It is often beneficial to view clinical annotations for each patient in the context of mutations to identify important associations between patient or treatment categoriese. For example, mutations in a specific gene may be associated with treatment response, sex, tumor stage, or subtype.

For this section of the lab, we will use the same mutation data (Phase 1 clinical trial) from previously and the corresponding clinical data for this set of patient sample.

1. Import the clinical data

In [0]:
```
clinical_data <- read.delim("data/BKM120_Clinical.tsv", sep = "\t")
clinical_data[1:3,]
```

In [0]:
```
colnames(clinical_data)
```

Next we create a subset of the clinical data: sample information and one or more clinical variables. This is the clinical information we need to annotate the waterfall plot.

In [0]:
```
clinical_data <- clinical_data %>% select(sample.Number, Best.response, PTEN.Immunohist.)
```

2. Make sure the sample column matches the mutation data

We have to clean up this clinical data `data.frame` to match the mutation data from earlier.

- The `sample.Number` column name should match the sample column `sample` that was used in `mutation_df` previously.
- The sample ID in this `sample.Number` column should match the sample ID format in the `sample` column from `mutation_df` .

Let's take a look at `mutation_df` again.

In [0]:
```
mutation_df[1:3,]
```

Next apply the changes.

In [0]:
```
clinical_data  <- clinical_data %>%
                    rename(sample = sample.Number) %>%  # change column name to `sample`
                    # mutate() applies changes to each cell in a specified column
                    # gsub() reads the string in `sample` and replaces the pattern matching "WU0+" with an empty
string ""
                    mutate(sample = gsub("WU0+","",sample))
clinical_data[1:3,]
```

3. Rearrange patient sample order by clinical category

We use the `arrange()` function from `dplyr` to sort the rows of a `data.frame` or `data.table` by specific columns.

In [0]:
```
clinical_data <- clinical_data %>%
   arrange(Best.response, PTEN.Immunohist.)
```

Now that the values in the `sample` column is sorted by `Best.response` and `PTEN.Immunohist.` , we store this as a vector `sample_levels` .

In [0]:
```
sample_levels <- clinical_data$sample
sample_levels
```

Then we use `factor()` to transform the column `sample` to a categorical variable, so we can specify the order of levels for this variable using the `levels` parameter.

In [0]:
```
clinical_data$sample <- factor(clinical_data$sample, levels=sample_levels)
mutation_df$sample <- factor(mutation_df$sample, levels=sample_levels)
```

4. Create the Waterfall plot with the clinical data

4. Create the Waterfall plot with the clinical data

The `Clinical()` function uses clinical data as the input and stores it as an object that we can use in the `Waterfall()` function to create a waterfall plot with clinical annotation. `Clinical()` looks for a `data.frame` or `data.table` with a `sample` column. All other columns will be treated as clinical variables.

Here we create a named vector for clinical variables and colors to set a color palette. In our example, we have six clinical variables that we must supply names and colors for. We use hex codes here to designate color, but any character string that R can recognize as a color will work.

In [0]:
```
# set up a color palette and specify that we would like our legend split up into two columns.
color_palette <- c("Progressive Disease"="#798E87",
                   "Stable Disease"="#C27D38",
                   "Partial Response"="#CCC591",
                   "negative"="#29211F",
                   "positive"="#9C964A",
                   "unknown"="#85D4E3")
```

In [0]:
```
clinicalData <- Clinical(inputData = clinical_data, palette = color_palette, legendColumns = 2)
```

In [0]:
```
# (optional) You may modify the settings here to adjust the figure dimensions
options(repr.plot.width = 10, repr.plot.height = 8)
```

In [0]:
```
plotData <- Waterfall(mutation_df,
                      mutationHierarchy = mut_Hierarchy,
                      clinical = clinicalData,
                      sampleOrder = sample_levels,
                      plotATally= "simple",
                     plotA = "frequency")
drawPlot(plotData)
```

Exercise 2a: Create a waterfall plot with clinical annotations (3 pt)

Instructions: In the code cell below, use the `Clinical()` and `Waterfall()` functions to plot the genes with mutations in **10% or more samples**.

Use the following information to create the waterfall plot:

- Mutation data: `mutation_df` (from earlier)
- Mutation hierarchy: `mut_Hierarchy` (from earlier)
- Clinical data: `data/BKM120_Clinical.tsv`
- Clinical categories: `Best.response` , `PTEN.Immunohist.` , `PgR.Status`
- Sort patient `sample` by `PgR.Status` , `PTEN.Immunohist.` , and `Best.response` (**in this order**).

Output:

- Store the `waterfall()` object in the variable `ex_2a`
- Use `drawPlot()` to display the plot.

In [0]:
```
# Enter your code here
```

Exercise 2b: Interpretation of the plot from Exercise 2a (1 pt)

In the markdown cell below, answer the following question in 1-2 sentences:

- Describe any mutation patterns that may be associated with the included clinical categories.

YOUR ANSWER HERE

Mutation Burden in Waterfall plot

Tumour mutational burden (TMB) is a predictive biomarker for benefit from immune-checkpoint inhibition across cancer types and within some cancer types. The predictive value of TMB is only proven for certain histologies and even for these, sensitivity and specificity for the

prediction of benefit are limited.

TMB is broadly defined as the number of somatic mutations per megabase (Mb) of interrogated genomic sequence. The specific TMB calculation depends on the gene panel assay used to sequence targeted regions (Sha et al. 2020). The use of large sequencing panels and tissue samples of sufficient tumour purity are key to ensuring accurate TMB quantification and precise patient stratification.

- If a cancer has a **high** tumor mutation burden level of **10 mut/Mb or greater**, it's more likely to respond to immunotherapy medication. This means that a specific drug type—immune checkpoint inhibitors—may activate the immune system to help the body recognize cancer cells. With more mutations, the higher the chance that one of the mutations will be able to identify and target cancer cells.
- Cancers with **low** tumor mutation burden levels of **10 mut/Mb or lower** may be less successful in activating the immune system to identify and target cancer cells.

The `Waterfall()` function uses the following formula to estimate mutation burden:

`(# of mutations)/(coverage space) * 1,000,000`

- `coverageSpace` : An integer representing the **coverage space** (the number of base pairs adequately covered in the experiment)

To add mutation burden to the waterfall plot, we to update the following parameters in the `Waterfall()` function:

- `coverage` : custom `TSV` file containing a sample column and the number of covered to a certain minimum threshold.
- `plotA` : set as `"burden"`
- (if you are providing your own values) `mutBurden` : A `data.frame` with two columns `sample` and `mut_burden`
  - `sample` needs to match the samples in mutation data
  - `mut_burden` contains the user-defined mutational burden values

By default, the `Waterfall()` displays observed mutation frequencies from the input data in its top panel. However, this can be misleading due to variances in the genomic space sequenced within the cohort. So we calculate the tumor mutation burden (TMB), which normalizes for covered space.

1. Import coverage data

First we import the coverage data from a custom TSV file. The file contains a sample column and the number of bases covered to at least 20×.

In [0]:
```
mutationBurden <-
read.delim("data/bkm120_coverage.txt", sep="\t")
mutationBurden[1:3,]
```

The sample column must match the sample identifiers used in the mutation data, so we'll use the same `gsub()` from earlier to rename the samples.

In [0]:
```
mutationBurden  <- mutationBurden %>%
                    # mutate() applies changes to each cell in a specified column
                    # gsub() reads the string in `sample` and replaces the pattern matching "WU0+" with an empty
string ""
                    mutate(sample = gsub("WU0+","",sample))
mutationBurden[1:3,]
```

2. Coverage space

Next we create a vector that contains the coverage space values for each sample.

In [0]:
```
coverage_space<- as.numeric(mutationBurden$coverage)
names(coverage_space) <- mutationBurden$sample
```

3. Create the Waterfall plot with mutation burden

1. Set `plotA` parameter as `"burden"` .
2. Assign the vector `coverage_space` to the parameter `coverage` for the function to calculate the mutation burden.
3. Add cell labels using `labelColumn` to indicate amino acid change.

In [0]:
```
# (optional) You may modify the settings here to adjust the figure dimensions
options(repr.plot.width = 18, repr.plot.height = 9)
```

In [0]:
```
plotData <- Waterfall(mutation_df,
                      mutationHierarchy = mut_Hierarchy,
                      coverage=coverage_space,
                      labelColumn="amino.acid.change", # optional
                      labelSize = 3.5,
                      plotA = "burden")
drawPlot(plotData)
```

Important: If you subset the mutation data only include some genes, it would reduce the accuracy of the mutation burden calculated by `Waterfall()`. You can use some `Waterfall()` parameters for limiting the genes and mutations plotted without affecting the mutation burden calculation. These parameters are `recurrence`, `plotGenes`, `maxGenes` and `rmvSilent`.

Exercise 3a: Add mutation burden to your figure from Ex 2a (1 pt)

Use the coverage data from `"data/bkm120_coverage.txt"` to add mutation burden to **your figure from Exercise 2a**. Include additional cell labels using `labelColumn` to indicate amino acid change.

Ouput:

- Store the `Waterfall()` object in the variable `ex_3a`
- Use `drawPlot()` to display the plot.
- (Optional) You may export the plot as a pdf to get a clearer look of the cell labels.

In [0]:
```
# Enter your code here
```

In [0]:
```
testthat::test_that("ex_3a exists", {
  expect_true(exists("ex_3a"))
})

### BEGIN HIDDEN TESTS

# students will NOT see this extra test
test_that("Object fields are correct", {
  expect_equal(ex_3a$labelColumn, "amino.acid.change", info = "One of your parameters is incorrect")
  expect_equal(ex_3a$clinical, ex_2a$clinical, info = "One of your parameters is incorrect")
  expect_equal(ex_3a$recurrence, 0.1, info = "One of your parameters is incorrect")
})
### END HIDDEN TESTS
```

Exercise 3b: Interpretation of the figure (3 pt)

Answer the following questions in the markdown cell below:

1. What is the most commonly mutated gene in this cohort?
2. Are there any recurrently mutated amino acid positions in this gene?
3. If yes, are they known mutation hotspots? How would you determine this?

YOUR ANSWER HERE

Transition/transversion plot

The `TvTi()` function allows us to visualize transition and transversion proportions/frequencies for a given cohort sample. We can compare these proportions with expectations to understand mutation profile patterns of a patient cohort.

For example, the mutation data could be from a patient cohort sample exploring the mutational effect of smoking status of patients with lung carcinomas. A sample interpretation of the plot would be "smoking tends to increase G -> T/C -> A transversions due to oxidative damage."

1. Plotting the transition and transversion proportions

Input:

- `fileType` : `<str>` MAF, MGI
  - MAF: contains sample and allele information
  - MGI: a data frame with column names "sample", "reference", and "variant" (usually source this data from Genome Modelling System)

We will use the same `mutation_df` dataset as previously to create the Transition/transversion plot.

In [0]:
```
# create the plot
TvTi_prop <- TvTi(mutation_df , fileType="MGI")
```

The proportion of transitions and transversions which can be misleading if the frequency of mutation events is not high enough to get an accurate calculation of the relative proportion.

2. Plotting the transition and transversion frequencies

1. Set `type` parameter to `"Frequency"` to plot the observed frequency of each Transition/Transversion type.
2. Assign a vector of colours to the `palette` parameter plot the the frequency by Transition/Transversion type. The length of the vector should match the number of Transition/Transversion categories.

In [0]:
```
TvTi_freq <- TvTi(mutation_df, type = "Frequency", fileType = "MGI")
```

3. Adding clinical data to the Tv/Ti plot

`TvTi()` requires a specific clinical data table format. To add clinical data to the Tv/Ti plot, we need to:

1. Convert `clinical_data` to long format using `melt()` .
2. Rename the column names to `sample` , `variable` , and `value` .

In [0]:
```
# convert to long format
clinical_data_long <- melt(data=clinical_data, id.vars="sample")

# rename columns
colnames(clinical_data_long) <- c("sample", "variable", "value")
```

We'll use the same color palette as the waterfall plot with clinical data from earlier.

In [0]:
```
# set up a color palette for the clinical categories
color_palette <- c("Progressive Disease"="#798E87",
                   "Stable Disease"="#C27D38",
                   "Partial Response"="#CCC591",
                   "negative"="#29211F",
                   "positive"="#9C964A",
                   "unknown"="#85D4E3")
```

We add these additional parameters to add the clinical data to our plot:

- `clinData` : a `data frame` with rows representing clinical data. The data frame should be in **long format** and columns must be named as `sample` , `variable` , and `value`
- `clinLegCol` : An integer specifying the number of columns in the legend for the clinical data
- `clinVarCol` : A vector specifying the color palette for the clinical categories (ex. `"variable"="colour"`)

In [0]:
```
TvTi_prop <- TvTi(mutation_df,
                  fileType = "MGI",
                  clinData= clinical_data_long,
                  clinLegCol = 2,
                  clinVarCol = color_palette,
                  sample_order_input = sample_levels
                 )
```

4. Combine the two plots

1. Set the `out` (output) parameter as `"grob"` .

2. Use the `arrangeGrob()` function from `gridExtra` package to combine the two plots into one.

3. Use `gridDraw()` to display the output.

In [0]:
```
# (optional) You may modify the settings here to adjust the figure dimensions
options(repr.plot.width = 10, repr.plot.height = 12)
```

In [0]:
```
TvTi_freq <- TvTi(mutation_df,
                  type = "Frequency",
                  fileType = "MGI",
                  clinData= clinical_data_long,
                  clinLegCol = 2,
                  clinVarCol = color_palette,
                  sample_order_input = sample_levels,
                  out = "grob"
                 )

TvTi_prop <- TvTi(mutation_df,
                  fileType = "MGI",
                  clinData= clinical_data_long,
                  clinLegCol = 2,
                  clinVarCol = color_palette,
                  sample_order_input = sample_levels,
                  out = "grob"
                 )
finalGrob <- arrangeGrob(TvTi_freq, TvTi_prop, ncol=1, heights = c(1,1))
grid.draw(finalGrob)
```

We can see from the resulting plot that quite a few samples have low mutation frequencies. This means we should interpret these samples with caution.

Exercise 4: Short Answer (1 pt)

Note: The `TvTi()` is currently deprecated but still usable. `MutSpectra()` is the new equivalent, but still active under development. So we will just practice the interpretation of transition/transversion plots.

Instructions:

- In the markdown cell below, describe your interpretation of the combined figure above in 1-2 sentences. (1 pt)

YOUR ANSWER HERE

Supplemental Information

Sources

- Procedure:
    - https://currentprotocols.onlinelibrary.wiley.com/doi/10.1002/cpz1.252
    - https://genviz.org/course/
- Dataset: Ma, C. X. et al. A Phase I Trial of BKM120 (Buparlisib) in Combination with Fulvestrant in Postmenopausal Women with Estrogen Receptor–Positive Metastatic Breast Cancer. Clinical Cancer Research 22, 1583-1591 (2016). https://doi.org/10.1158/1078-0432.Ccr-15-1745

Other Useful Packages

- CoMut - A Python library for creating comutation plots to visualize genomic and phenotypic information
- maftools - An R package to summarize, analyze and visualize MAF files
- somaticfreq - Rapid genotyping of known somatic hotspot variants from the tumor BAM files. Generates a browsable/sharable HTML report