Visualizing Small Variants

Zachary L. Skidmore [1]

[1]McDonnell Genome Institute - Washington University

2018-01-03

Abstract

Instructions for visualizing small variants using the GenVisR package

Package

GenVisR 1.10.0

Contents

1 GenVisR a brief introduction . . . 2
1.1 getting started . . . 2
2 Visualization of small variants . . . 2
2.1 Supported files . . . 2
2.2 Plot Objects . . . 3
3 Basic Pipeline . . . 3
3.1 Reading in data . . . 3
3.2 Constructing a Plot Object and viewing plotted data . . . 4
3.3 Drawing and saving a plot object . . . 5
4 Functions for visualizing small variants . . . 5
4.1 Waterfall . . . 5
4.2 MutSpectra . . . 9
4.3 Rainfall . . . 10
4.4 Lolliplot . . . 12
4.5 SmallVariantSummary . . . 12
5 Session Info . . . 12

1 GenVisR a brief introduction

Intuitively visualizing and interpreting data from high-throughput genomic technologies continues to be challenging. Often creating a publication ready graphic not only requires extensive manipulation of data but also an in-depth knowledge of graphics libraries. As such creating such visualizations has traditionally taken a significant amount of time in regards to both data pre-processing and aesthetic manipulations. GenVisR (Genomic Visulizations in R) attempts to alleviate this burden by providing highly customizable publication-quality graphics in an easy to use structure. Many of the plotting functions in this library have a focus in the realm of human cancer genomics however we support a large number of species and many of the plotting methods incorprated within are of use for visualizing any type of genomic abnormality.

1.1 getting started

For the majority of users we recommend installing GenVisR from the release branch of Bioconductor, Installation instructions using this method can be found on the GenVisR landing page on Bioconductor.

Please note that GenVisR imports a few packages that have "system requirements", in most cases these requirements will already be installed. If they are not please follow the instructions to install these packages given in the R terminal. Briefly these packages are: "libcurl4-openssl-dev" and "libxml2-dev"

Once GenVisR is successfully installed it will need to be loaded. For the purposes of this vignette we do this here and set a seed to ensure reproducibility.

```
# set a seed
set.seed(426)

# load GenVisR into R
library(GenVisR)
```

2 Visualization of small variants

Genomic changes come in many forms, one such form are single nucleotide variants (SNVS) and small insertions and deletions (INDELS). These variations while small can have a devestating impact on the health of cells and are the basis for many genomic diseases. We provide 5 functions for visualizing the impact these changes can have and to aid in recognizing patterns that could be associated with genomic diseases.

2.1 Supported files

In order to be as flexible as possible GenVisR is designed to work with 3 file formats which encode information regarding small variants. The first of these are vep formated files from the Variant Effect Predictor (VEP), a tool developed by ensembl to annotate the effect a variant has on a genome. Mutation Annotation Format (MAF) files are also widely used in

the bioinformatics community particularly within The Cancer Genome Atlas (TCGA) project and are supported by GenVisR. Annotation files from the Genome Modeling System (GMS) are also supported. In the interest of flexibility a fourth option exists as well, users can supply a `data.table` object to any function designed to work with small variants. Users should note however that specific columns are required for functions when using this method (see specific functions for details).

2.2 Plot Objects

The output from `Waterfall()`, `Lolliplot()`, `Rainfall()`, `SmallVariantSummary()`, and `MutSpectra()` are objects which store the data which was plotted as well as the individual subplots, and the final arranged plot. The reason for storing the data and plots in such a manner is two-fold. First allowing access to the data which was plotted provides transparency as to how the plot was produced. Secondly as all this items can be accessed by the user it allows for greater flexibility for customizing plots. These data and plots can be accessed with the `getData()` and `getGrob()` functions respectively. The final plot can be printed with the `drawPlot()` function which takes one of the afore mentioned objects.

3 Basic Pipeline

3.1 Reading in data

When constructing a plot with GenVisR there are three basic steps. First one must load the data into R, as mentioned previously GenVisR supports three filetypes which can be read in this manner. An example for each is supplied below using test data installed with the package.

3.1.1 Basic Syntax

1. Reading in a MAF file

```
# get the disk location for maf test file
testFileDir <- system.file("extdata", package = "GenVisR")
testFile <- Sys.glob(paste0(testFileDir, "/brca.maf"))

# define the objects for testing
mafObject <- MutationAnnotationFormat(testFile)
```

2. Reading in a VEP file

```
# get the disk location for test files
testFileDir <- system.file("extdata", package = "GenVisR")
testFile <- Sys.glob(paste0(testFileDir, "/*.vep"))

# define the object for testing
vepObject <- VEP(testFile)
```

3. Reading in a GMS file

```
# get the disk location for test files
testFileDir <- system.file("extdata", package = "GenVisR")
testFile <- Sys.glob(paste0(testFileDir, "/FL.gms"))

# define the objects for testing
gmsObject <- GMS(testFile)
```

3.1.2 Viewing Data

In some cases you may want to view data after it has been read in, there are functions available to make viewing these data easier. Briefly these are `getSample()`, `getPosition()`, `getMutation()`, and `getMeta()` which are globally available for each object type (GMS, VEP, etc.). Let's test one out by viewing the samples from the VEP file that was read in with `VEP()`.

```
# view the samples from the VEP file
getSample(vepObject)
```

3.1.3 Additional Notes

All of these functions expect a path to a file of the appropriate type. Optionally a wildcard can be supplied with * as was done with `VEP()` causing the function to read in multiple files at once. With the exception of MAF files these functions **expect the files being read in to have a column called "sample"** (MAF files already have a sample designation within the file). If a sample column is not found one will be created based on the filenames. Each of these functions will attempt to infer a file specification version from the file header. If a version is not found one will need to be specified via the `version` parameter.

3.2 Constructing a Plot Object and viewing plotted data

Once the data is read in and stored in one of the previously mentioned objects the data can be plotted with one of the plotting functions. We will go over each plotting function in more detail in section 4 however to continue with our example pipeline let's create a simple waterfall plot for the VEP annotated data.

```
waterfallPlot <- Waterfall(vepObject, recurrence = 0.4)
```

With the plot object created we can view the actual data making up the plot with the `getData()` function. This is an accessor function to pull out specific data making up the plot (Refer to the R documentation for `Waterfall()` to see available slots in the object which hold data). Let's use it here to extract the data making up the main plot panel, we can specify the slot either by name or it's index.

```
# extract data by the slot name
getData(waterfallPlot, name="primaryData")

# extract data by the slot index (same as above)
getData(waterfallPlot, index=1)
```

3.3 Drawing and saving a plot object

With the plot object defined we can now visualize the plot, this is achieved by calling the `drawPlot()` function on the plot object. The plot can be saved simply by opening a graphics device using the base R functions such as `pdf()`, `svg()` etc. It should be noted that space within the graphics device will need to be adequate in order to draw a plot. If plots appear to be overlapping try giving the plot more space to draw via the `height` and `width` parameters.

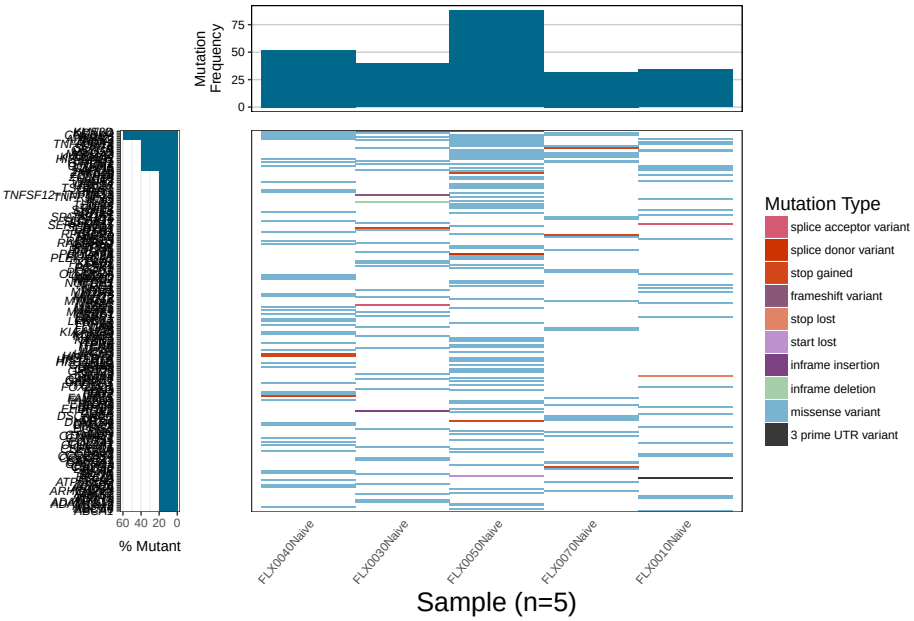
```
# draw the plot
drawPlot(waterfallPlot)

# draw the plot and save it to a pdf
pdf(file="waterfall.pdf", height=10, width=15)
drawPlot(waterfallPlot)
dev.off()
```

4 Functions for visualizing small variants

4.1 Waterfall

The Waterfall plot is designed to make recognizing patterns related to co-occurring or mutually exclusive events easier to visualize across a cohort. It plots mutations for a gene/sample in a hierarchical order beginning with the most recurrently mutated gene for a cohort and working it's way down. By default two sub-plots are also displayed which show the mutation burden for a sample and the number of genes effected by a mutation in a cohort. Let's start by making a simple waterfall plot with default parameters from the MAF file we read in earlier.

```
# draw a waterfall plot for the maf object
drawPlot(Waterfall(vepObject, recurrence = 0.2))
```

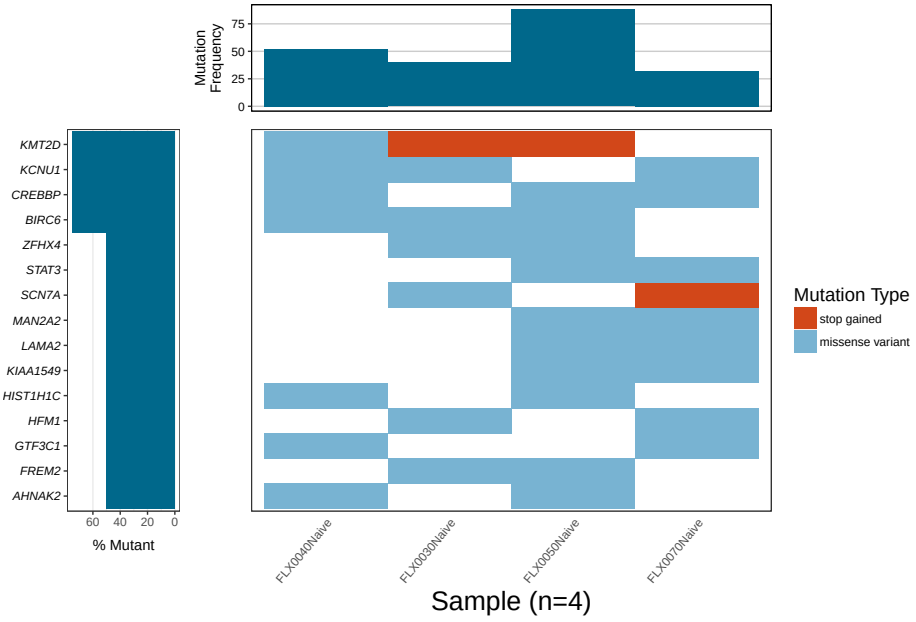


4.1.1 Using filtering parameters with Waterfall

Relevant Parameters: `samples`, `recurrence`, `genes`, `geneMax`, `mutation`

Often it is the case that the input data supplied to the `Waterfall()` function will contain thousands of genes and hundreds of samples. While `Waterfall()` can handle such scenarios the graphics device `Waterfall()` would neeed to output to would have to be enlarged to such a degree that the visualization may become unwieldy. To alleviate such issues and show the most relevant data `Waterfall()` provides a suite of filtering parameters. Let's filter our plot using a few of these, suppose we only wanted to visualize those genes which occur in 50% of the cohort and we further didn't care about sample "FLX0010Naive". We could tell `Waterfall()` this by giving the samples we only wanted plotted via the `samples` parameter and setting the `recurrence` parameter to .50.

```
# show those genes which recur in 50% of the cohort for these 4 samples
drawPlot(Waterfall(vepObject, recurrence = 0.5, samples = c("FLX0040Naive", "FLX0070Naive",
    "FLX0050Naive", "FLX0030Naive")))
```



4.1.2 Mutation sub-plot

Relevant Parameters: `coverage`, `plotA`, `plotATally`, `plotALayers`

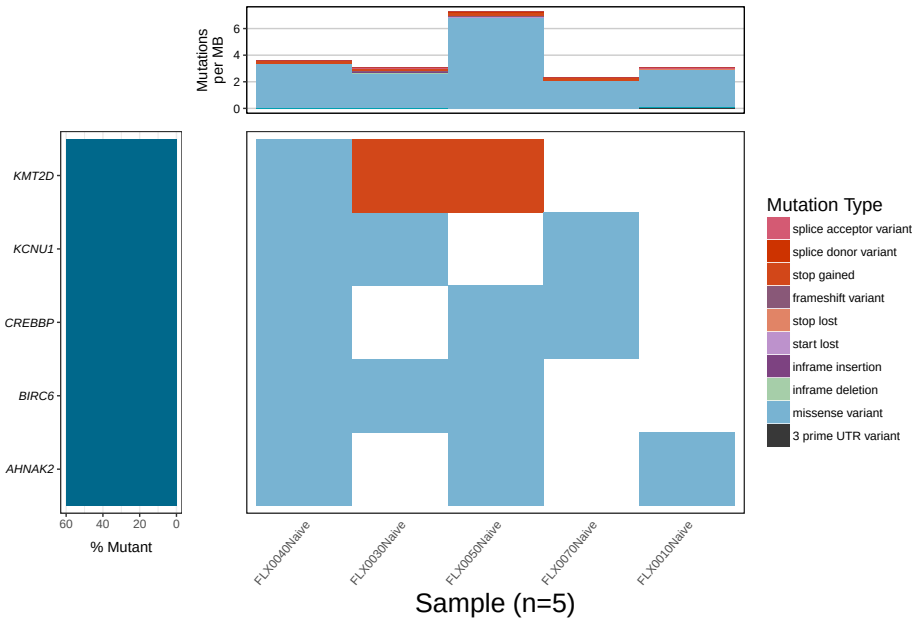
You might have noticed that when filtering the waterfall plot the top sub-plot didn't change i.e. for sample "FLX0040Naive" the Frequency of mutations remained around 50. You may also have noticed a warning when constructing a waterfall plot for the VEP data saying that duplicate genomic locations were detected. This is important to note and understand, regardless of any filtering that occurs the top sub-plot will always be based on the original input with one caveat. It will check for duplicate variants *(i.e. same sample, allele, genomic position)* and remove variants which are deemed to be duplicated. This occurs in the case of VEP as different transcripts can be annotated with different mutation types but still be the same variant which is why we see the warning. By default `Waterfall()` will simply output the frequencies observed in the cohort. However a mutation burden can be displayed as well

by changing the value for parameter `plotA` from "frequency" to "burden". In doing so you will also **need to supply a value to `coverage`** giving the aproximate space in base pairs for which a mutation could have been called. Doing so will plot a mutation burden instead of a mutation frequency calculated via the following formula:

```
Mutation Burden = (# of observed mutations per sample)/(# of bases for which a
mutation could be called) * 1000000
```

For a more accurate mutation burden calculation we can also supply coverage values individual per sample via a named vector, let's do that and tell `Waterfall()` to calculate a mutation burden by mutation type + sample rather than sample alone via the parameter `plotATally`. Lastly we will set the `drop` parameter to FALSE to avoid dropping mutations from the legend that are not in the main plot but are in the top sub-plot.

```
# define a coverage for each sample
sampCov <- c(FLX0040Naive = 14500000, FLX0070Naive = 13900000, FLX0050Naive = 12100000,
    FLX0030Naive = 1.3e+07, FLX0010Naive = 1.1e+07)

drawPlot(Waterfall(vepObject, recurrence = 0.5, coverage = sampCov, plotA = "burden",
    plotATally = "complex", drop = FALSE))
```



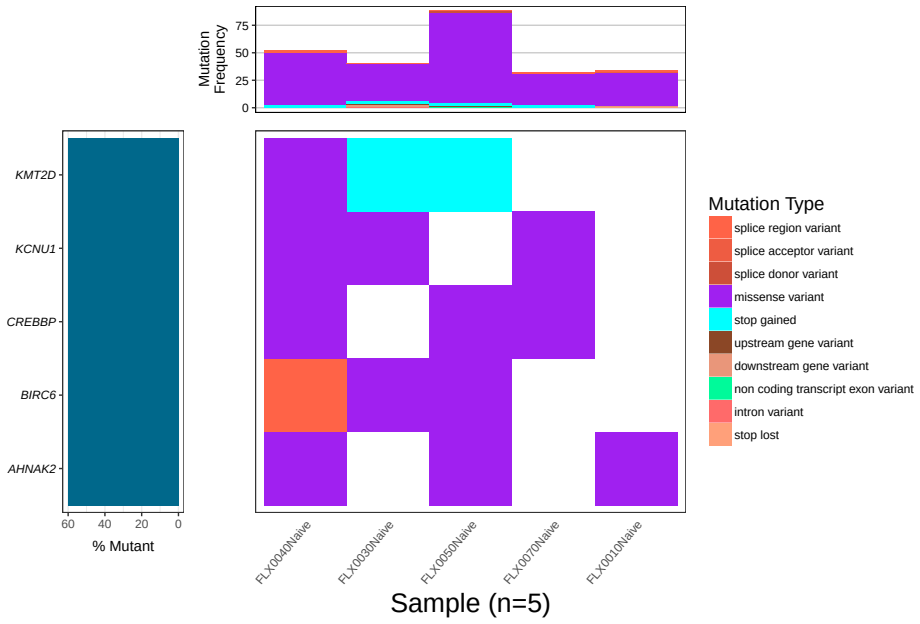
4.1.3 Altering the mutation colors and hierarchy

Relevant Parameters: `mutationHierarchy`

You may ask yourself what happens when there are two different mutations for the same gene/sample, which one is plotted? For each file type a pre-defined hierarchy of mutations has been established which follows the order of the "Mutation Type" legend from top to bottom. By default mutations are based on a hierarchy of being most deleterious however both the priority and color of mutations can be changed by supplying a data.table to the `mutationHierarchy` parameter. Let's assume that we are particularly interested in splice variants, by default these might not show up because a "missense_variant" is deemed to be

more deleterious but let's change that by supplying a custom hierarchy along with colors to go along with it. To do that we will need to give the parameter a data.table object with column names "mutation" and "color".

```
# find which mutations are in the input data
mutations <- getMutation(vepObject)$Conse

# define a new color and hierarchy for mutations
library(data.table)
newHierarchy <- data.table(mutation = c("splice_region_variant", "splice_acceptor_variant",
    "splice_donor_variant", "missense_variant", "stop_gained"), color = c("tomato1",
    "tomato2", "tomato3", "purple", "cyan"))

# draw the plot
drawPlot(Waterfall(vepObject, recurrence = 0.5, mutationHierarchy = newHierarchy,
    plotATally = "complex", drop = FALSE))
```



Here we can see that for sample "FLX0040Naive" on gene *BIRC6* there is actually a splice_region_variant as well as a missense_variant.

Note: *If mutations are found in the input but are not listed in the mutationHierarchy parameter they will be addded and assigned random colors*

Note: *In a VEP formated file mutations can be comma delimited, if they are not specifically specified in the mutation hierarchy as comma delimited the mutations are split up and collapsed*

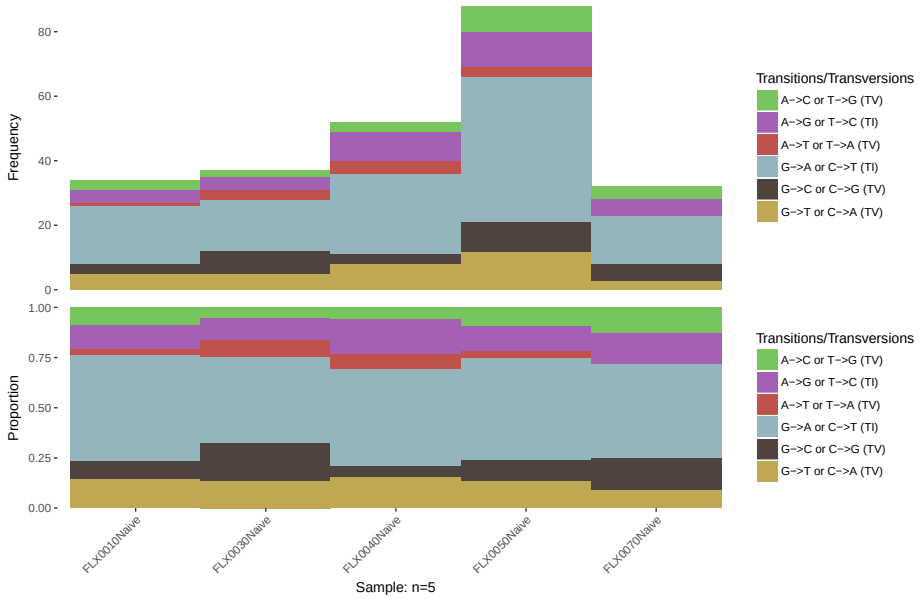
4.2 MutSpectra

The MutSpectra plot is designed to make recognizing bias in the relative rates of transitions and transversions easier. It annotates transitions and transversions based on the variant call and then plots the relative proportion observed in each sample. A frequency plot for these transitions and transversions is also calculated and displayed in order to determine if an observed ratio is attributable to a low number of mutations for a given sample.

4.2.1 Input

Input to the `MutSpectra()` function can be just one of the afore mentioned and supported file format object discussed in section 3.1.1. *In cases where the file format does not provide a reference base, such as with vep, a `BSgenome` object will also need to be provided* in order to annotate reference bases for a given position. These `BSgenome` objects are a core object class in bioconductor and can be installed through the instructions on the bioconductor site. Let's go ahead and create a MutSpectra plot for the vep data. We first determine which assembly vep used for these annotations ussing the `getHeader()` function and can see that it is GRCh37. We next load in the BSgenome for GRCh37 (GRCh37 and hg19 are analogous assemblies with very minor differences) and supply the BSgenome object to the parameter `BSgenome`.

```
# determine the appropriate BSgenome object to use from the vep header
assembly <- getHeader(vepObject)
assembly <- assembly[grepl("assembly", assembly$Info), ]

# load in the correct BSgenome object
library(BSgenome.Hsapiens.UCSC.hg19)

# create a MutSpectra plot
drawPlot(MutSpectra(vepObject, BSgenome = BSgenome.Hsapiens.UCSC.hg19))
```



Note: *In the example above you'll see a warning related to chromosome mismatches, this is part of the difference between GRCh37 and hg19, chromosomes are designated differently between the two (i.e. chr1 and 1).* `MutSpectra()` *is smart enough to recognize this and append chr to all chromosomes.*

Note: *If you are unsure what assembly to use try running* `MutSpectra()` *without specifying one. It will attempt to match an assembly from bioconductor with the information in the header.*

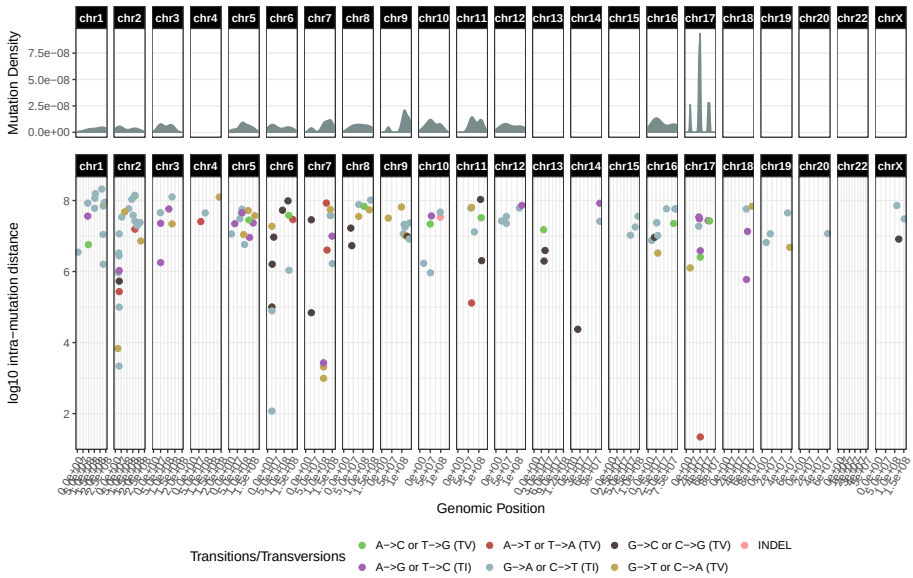
4.3 Rainfall

The Rainfall plot is designed to make visualizing kataegis across the entire genome easier. It calculates the genomic distances between consecutive mutations for each sample and chromosome on a log10 scale. This information is displayed on the y-axis over the context of genome coordinates on the x-axis. A density plot for these mutations is also displayed above the main plot to display the number of mutations within a mutation hotspot.

4.3.1 Input

As with the MutSpectra plots basic input consists of an object corresponding to one of the supported file types. Further a BSgenome object is required if the input format does not contain a reference column (see the MutSpectra section for more details). Let's start by creating a Rainfall plot for our VEP object. You might notice in the plot that density estimates are missing for a few chromosomes, this is normal and is done intentionally to avoid artificially skewing the y-axis when there are too few points to construct a confident density plot.

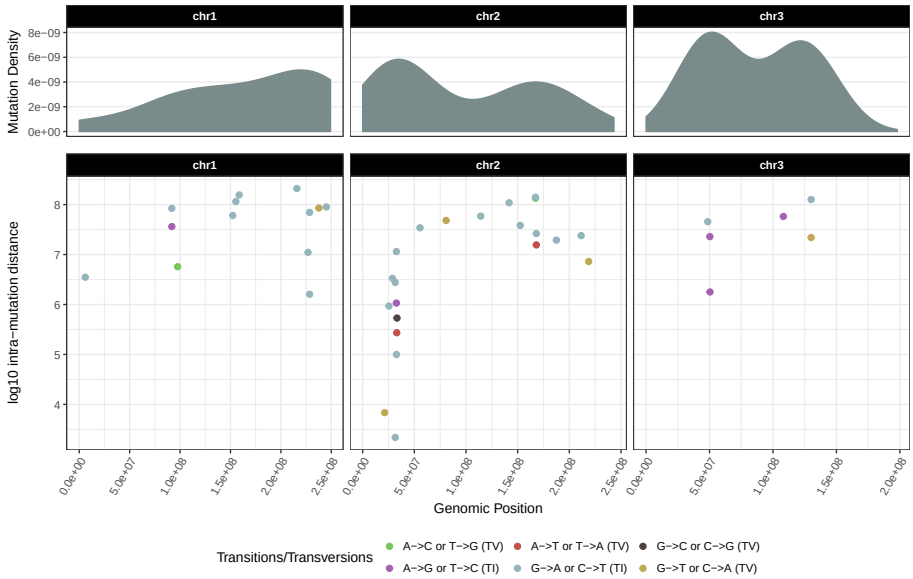
```
# determine the appropriate BSgenome object to use from the vep header
assembly <- getHeader(vepObject)
assembly <- assembly[grepl("assembly", assembly$Info), ]

# load in the correct BSgenome object
library(BSgenome.Hsapiens.UCSC.hg19)

# create a Rainfall plot
drawPlot(Rainfall(vepObject, BSgenome = BSgenome.Hsapiens.UCSC.hg19))
```



4.3.2 Using filtering parameters with Rainfall

By default the `Rainfall()` function plots everything within the input, this means that it is not only plotting all chromosomes but also all samples. In some cases this may not be desireable but fortunately `Rainfall()` has two parameters to limit the data which is plotted, these are `sample` and `chromosomes` which will limit the samples and chromosomes ploted respectively. Let's go ahead and try them out, we'll limit our plot to only chromosomes 1-3 and only sample "FLX0010Naive".

```
# create a Rainfall plot limiting the chromosomes and samples plotted
drawPlot(Rainfall(vepObject, BSgenome = BSgenome.Hsapiens.UCSC.hg19, sample = c("FLX0010Naive"),
    chromosomes = c("chr1", "chr2", "chr3")))
```



4.4 Lolliplot

TODO: rework the existing lolliplot to be quicker and look better

4.5 SmallVariantSummary

TODO: create a class to output various summary plots regarding the number of mutations etc.

5 Session Info

```
sessionInfo()
## R version 3.4.2 (2017-09-28)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.1
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] BSgenome.Hsapiens.UCSC.hg19_1.4.0 BSgenome_1.46.0
##  [3] rtracklayer_1.38.0                Biostrings_2.46.0
##  [5] XVector_0.18.0                    GenomicRanges_1.30.0
##  [7] GenomeInfoDb_1.14.0               IRanges_2.12.0
##  [9] S4Vectors_0.16.0                  BiocGenerics_0.24.0
## [11] data.table_1.10.4-3               GenVisR_1.10.0
## [13] BiocStyle_2.6.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.14               lattice_0.20-35
##  [3] prettyunits_1.0.2          Rsamtools_1.30.0
##  [5] gtools_3.5.0               assertthat_0.2.0
##  [7] rprojroot_1.2              digest_0.6.12
##  [9] R6_2.2.2                   plyr_1.8.4
## [11] backports_1.1.1            RSQLite_2.0
## [13] evaluate_0.10.1            ggplot2_2.2.1
## [15] zlibbioc_1.24.0            rlang_0.1.4.9000
## [17] GenomicFeatures_1.30.0     progress_1.1.2
## [19] lazyeval_0.2.1             blob_1.1.0
```

```
## [21] Matrix_1.2-11              rmarkdown_1.6
## [23] labeling_0.3               RMySQL_0.10.13
## [25] BiocParallel_1.12.0        stringr_1.2.0
## [27] RCurl_1.95-4.8             bit_1.1-12
## [29] biomaRt_2.34.0             munsell_0.4.3
## [31] DelayedArray_0.4.0         compiler_3.4.2
## [33] htmltools_0.3.6            SummarizedExperiment_1.8.0
## [35] tibble_1.3.4               gridExtra_2.3
## [37] GenomeInfoDbData_0.99.1    bookdown_0.5
## [39] matrixStats_0.52.2         XML_3.98-1.9
## [41] viridisLite_0.2.0          GenomicAlignments_1.14.0
## [43] bitops_1.0-6               grid_3.4.2
## [45] gtable_0.2.0               DBI_0.7
## [47] magrittr_1.5               formatR_1.5
## [49] scales_0.5.0               stringi_1.1.6
## [51] reshape2_1.4.2             viridis_0.4.0
## [53] tools_3.4.2                FField_0.1.0
## [55] bit64_0.9-7                Biobase_2.38.0
## [57] yaml_2.1.14                AnnotationDbi_1.40.0
## [59] colorspace_1.3-2           memoise_1.1.0
## [61] knitr_1.17
```