gRog: An introduction

Zachary Skidmore

2015-07-16

Introduction

Intuitively visualizing and interpreting data from high-throughput genomic technologies continues to be challenging. "Genomic Visualizations in R" (GenVisR) attempts to alleviate this burden by providing highly customizable publication-quality graphics focused primarily on a cohort level (i.e., multiple samples/patients). GenVisR attempts to maintain a high degree of flexibility while leveraging the abilities of ggplot2 and bioconductor to achieve this goal.

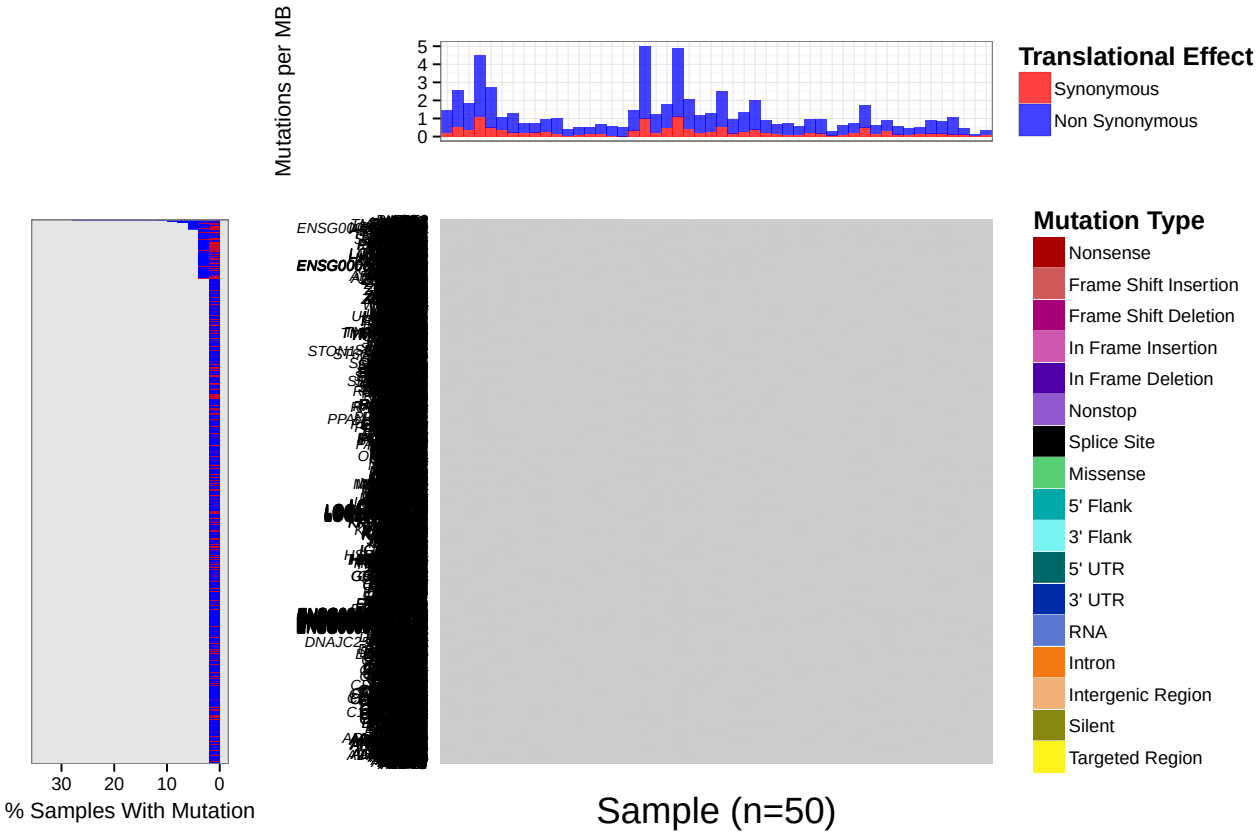
Functions

mutSpec

`mutSpec` provides a method of visualizing the mutational landscape of a cohort. The input to `mutSpec` consists of a data frame derived from either a .maf (version 2.4) file or a file in MGI annotation format (obtained from The Genome Modeling System). `mutspec` will display the mutation occurrence and type in the main panel while showing the mutation rate and the percentage of samples with a mutation in the side panels. Conflicts arising from multiple mutations in the same gene/sample cell are resolved by a hierarchical removal of mutations keeping the most deleterious as defined by the order of the "mutation type" legend.

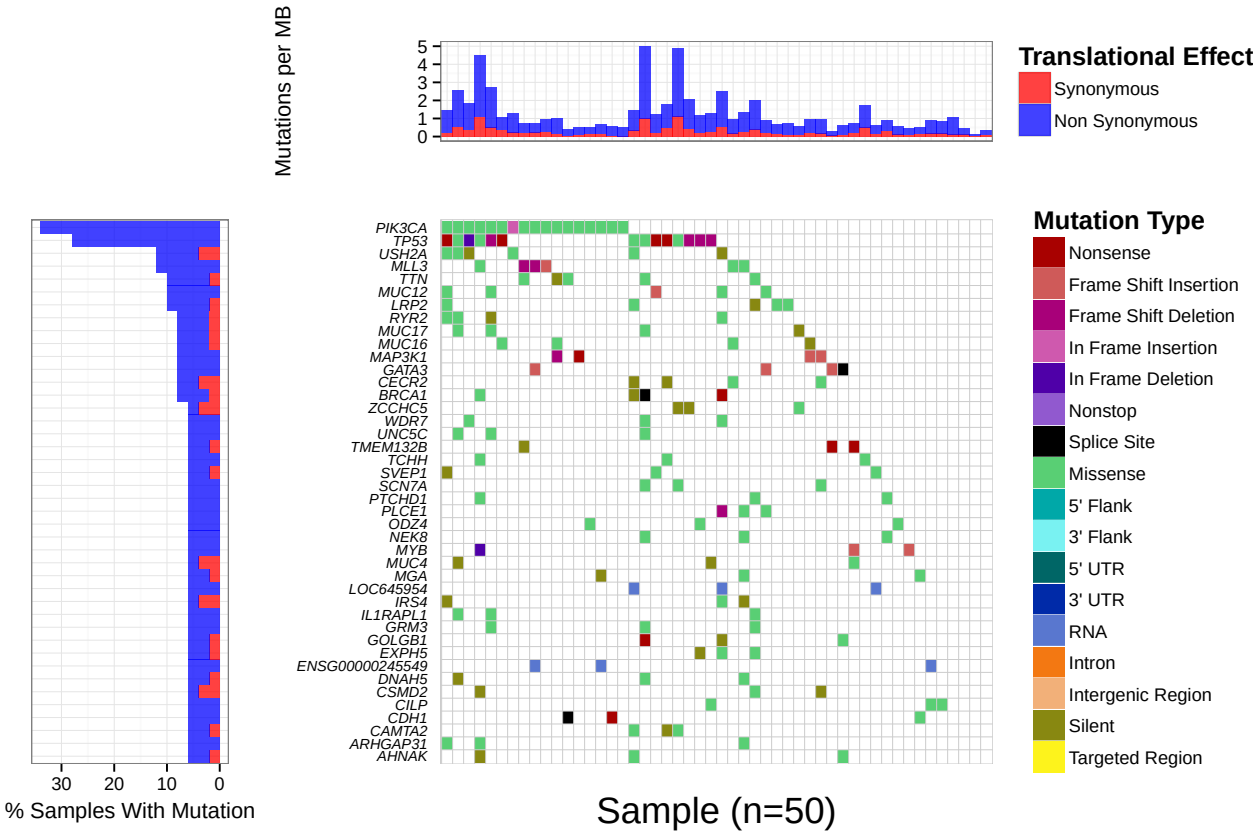
`BRCA_MAF` is a truncated MAF file consisting of 50 samples from the TCGA project corresponding to Breast invasive carcinoma (complete data). Using this dataset we can view the default behavior of `mutSpec`:

```
# Load GenVisR and set seed
library(GenVisR)
set.seed(426)

# Plot the mutation landscape
mutSpec(brcaMAF)
```



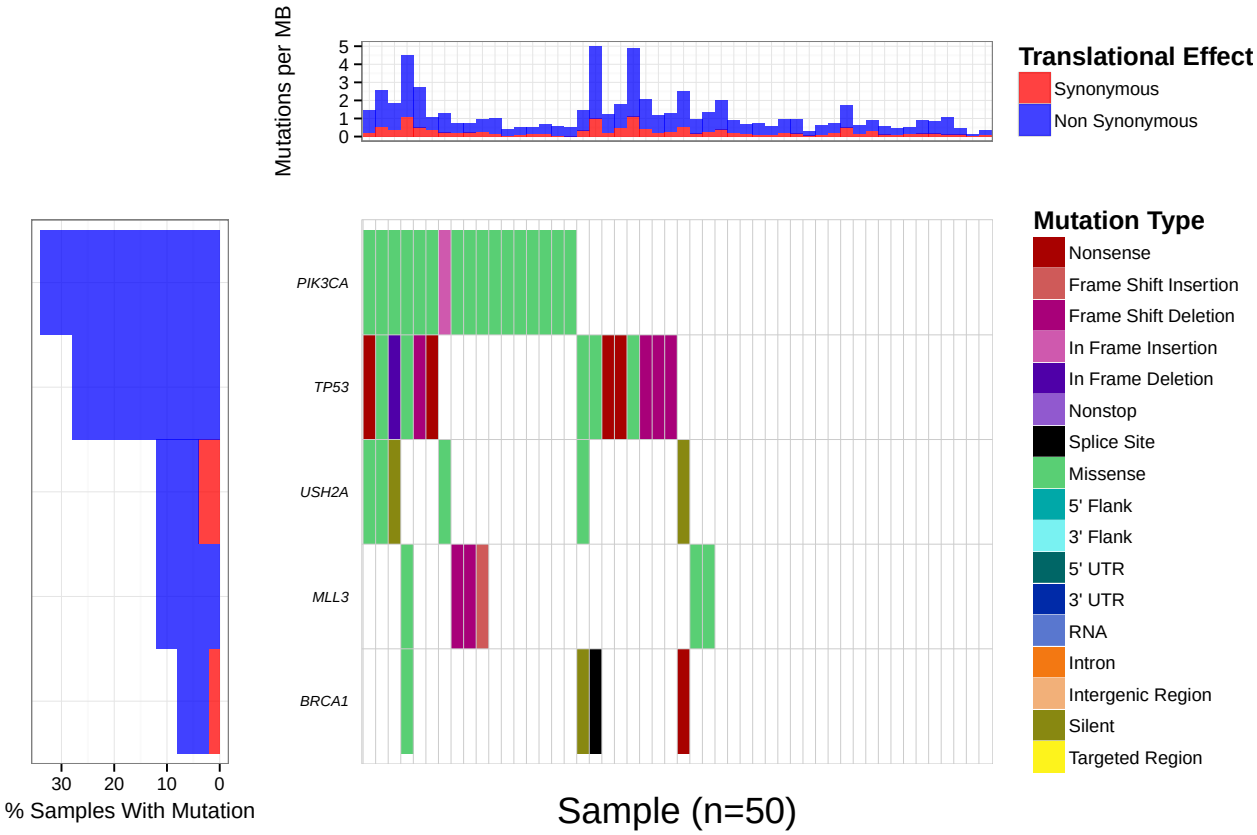
This type of view is of limited use without expanding the graphic device given the large number of genes. Often it is beneficial to reduce the number of cells in the plot by limiting the number of genes plotted. There are two ways to accomplish this, the `main.recurrence_cutoff` parameter accepts an integer and will remove genes from the data which do not have at least x number of samples mutated. For example the `brcaMAF` data set contains 50 samples, to plot genes with mutations present in at least 3 (6%) of samples one would specify the following.

```
# Plot only genes with mutations in 6% or more of samples
mutSpec(brcaMAF, main.recurrence_cutoff=3)
```



Alternatively one can select genes of interest using the `main.genes` parameter. This parameter accepts a case insensitive character vector of genes present in the data. For example, if it was desirable to plot only the following genes "PIK3CA", "TP53", "USH2A", "MLL3", AND "BRCA1":

```
# Plot only the specified genes
mutSpec(brcaMAF, main.genes=c("PIK3CA", "TP53", "USH2A", "MLL3", "BRCA1"))
```

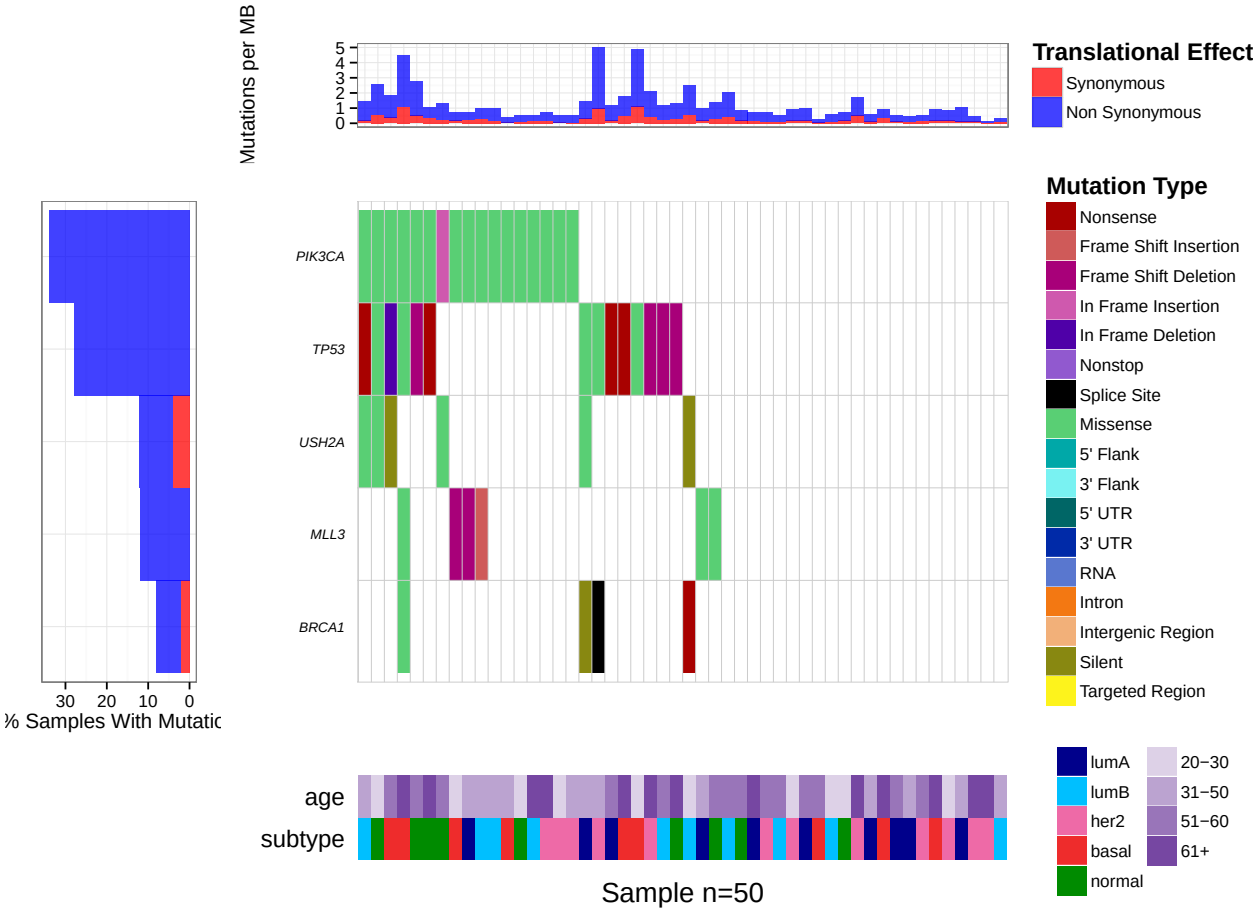


It is important to note that the mutation burden sub plot does not change during these subsets, this is calculated directly from the input via the formula: *mutations in sample/coverage space* $*$ 1000000. The coverage space defaults to the size in base pairs of the "SeqCap EZ Human Exome Library v2.0". This default can be changed via the parameter `coverageSpace`. This calculation is only meant to be a rough estimate as actual coverage space can vary from sample to sample, for a more accurate calculation the user has the option to supply an optional argument `mutBurden` giving the users own calculation for each sample, this should be a data frame with column names 'sample', 'mut_burden' taking the following form:

sample	mut_burden
TCGA-A1-A0SO-01A-22D-A099-09	2.51230753508422
TCGA-A2-A0EU-01A-22W-A071-09	3.04552735491393
TCGA-A2-A0ER-01A-21W-A050-09	2.12594754181242
TCGA-A2-A0EN-01A-13D-A099-09	1.84659978901238
TCGA-A1-A0SI-01A-11D-A142-09	2.4449286518028
TCGA-A2-A0D0-01A-11W-A019-09	2.53573701380388
TCGA-A2-A0D0-01A-11W-A019-09	2.11398439712244
TCGA-A1-A0SI-01A-11D-A142-09	1.61824706091035
TCGA-A2-A0CT-01A-31W-A071-09	2.45637135459403
TCGA-A2-A04U-01A-11D-A10Y-09	2.22746510988545

In addition to specifying the mutation burden the user also has the ability to plot additional clinical data. The clinical data supplied should be a data frame in "long" format with column names "sample", "variable", "value". It is recommended to use the `melt` function in the package reshape2 to coerce data into this format. Here we add clinical data to be plotted and specify a custom order and colours for these variables putting these values in two columns within the clinical plot legend:

```
# create fake clinical data
subtype <- c('lumA', 'lumB', 'her2', 'basal', 'normal')
subtype <- sample(subtype, 50, replace=TRUE)
age <- c('20-30', '31-50', '51-60', '61+')
age <- sample(age, 50, replace=TRUE)
sample <- as.character(unique(brcaMAF$Tumor_Sample_Barcode))
clinical <- as.data.frame(cbind(sample, subtype, age))

# melt the data
library(reshape2)
clinical <- melt(clinical, id.vars=c('sample'))

# Run mutSpec
mutSpec(brcaMAF, clinDat=clinical, clin.var.colour=c('lumA'='blue4', 'lumB'='deepskyblue',
'her2'='hotpink2', 'basal'='firebrick2', 'normal'='green4', '20-30'='#ddd1e7',
'31-50'='#bba3d0', '51-60'='#9975b9', '61+'='#7647a2'),
main.genes=c("PIK3CA", "TP53", "USH2A", "MLL3", "BRCA1"), clin.legend.col=2, clin.var.order=
c('lumA', 'lumB', 'her2', 'basal', 'normal', '20-30', '31-50', '51-60', '61+'))
```



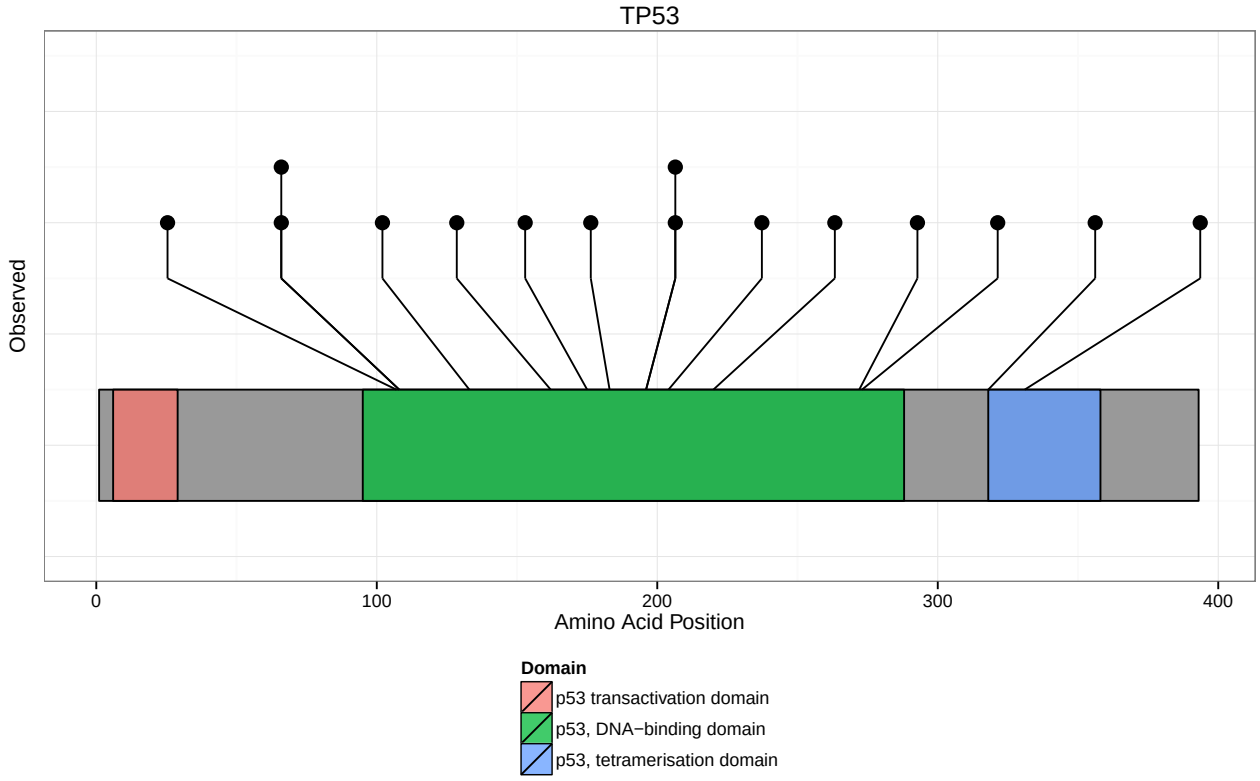
Occasionally there may be samples not represented within the .maf file (due to a lack of mutations). It may still be desirable to plot these samples. To accomplish this simply add the relevant samples into the appropriate column before loading the data and leave the rest of the columns as NA. Additionally it may be desireable to plot data not in a standard format. If this is the case it is recommended to set the `file_type` parameter to 'MGI' and name columns as 'sample', 'gene_name', and 'trv_type'. Valid levels for 'trv_type' are: "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", "5_prime_flanking_region", "3_prime_flanking_region", "3_prime_untranslated_region", "5_prime_untranslated_region", "rna", "intronic", and "silent".

lolliplot

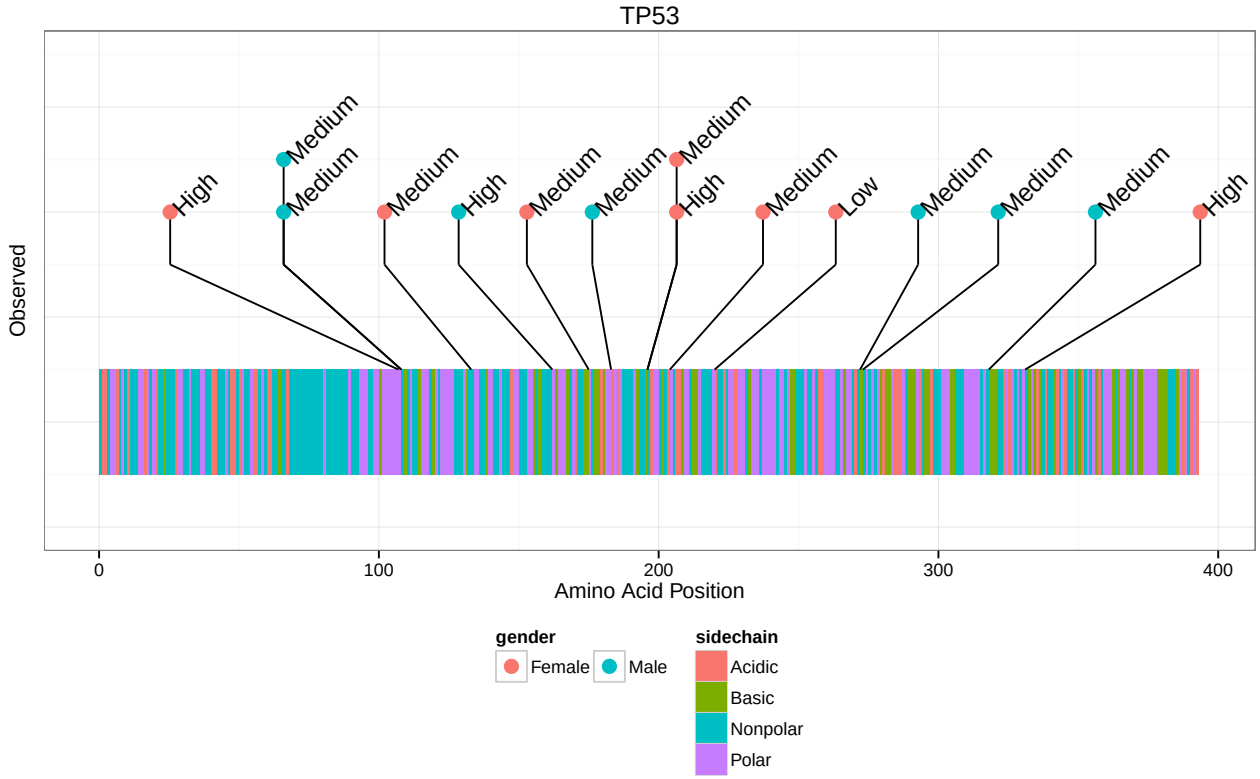
`lolliplot` provides a method for visualizing mutation hotspots overlayed on a protein framework. Basic input consist of a data frame with required columns 'transcript_name', 'gene' and 'amino_acid_change' giving the ensembl transcript id, gene name, and the amino acid change in p. notation respectivley. The data frame must contain only one unique transcript name. `lolliplot` queries various online databases to extract meta data for the transcript framework and as such needs an active internet connection. `lolliplot` also assumes the species to be *H.sapiens*, this assumption can be changed via the parameter `taxId` which takes a UniProt Taxonomic identifier.

```
# Create input data
data <- brcaMAF[brcaMAF$Hugo_Symbol == 'TP53',c('Hugo_Symbol', 'amino_acid_change_WU')]
data <- as.data.frame(cbind(data, 'ENST00000269305'))
colnames(data) <- c('gene', 'amino_acid_change', 'transcript_name')

# Call lolliplot
lolliplot(data)
```



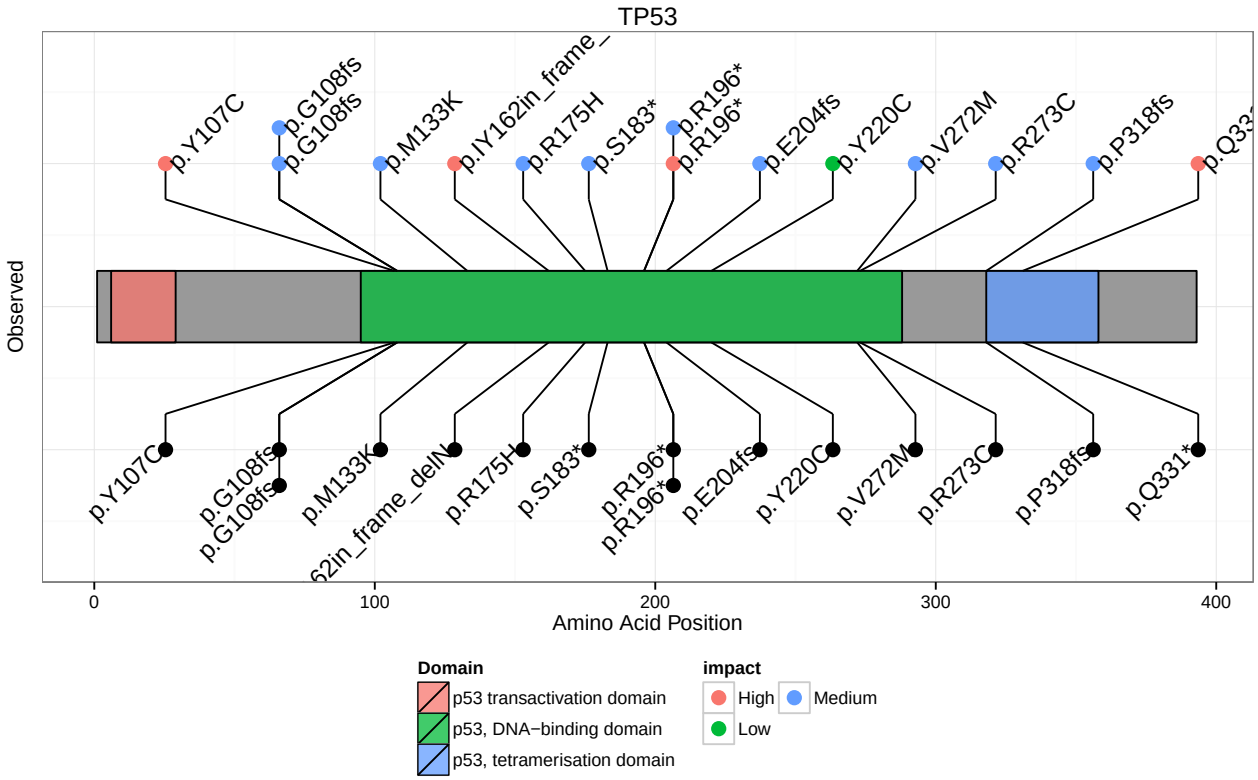
In an effort to maintain a high degree of flexibility the user has the option of selecting columns on which to fill and label. The parameters `fillCol` and `labelCol` allow this behavior by taking column names on which to fill and label respectively. Additionally one can plot the amino acid sidechain information in lieu of protein domains.

```
# Add additional columns to the data
data$gender <- sample(c("Male", "Female"), 15, replace=T)
data$impact <- sample(c("Low", "Medium", "High"), 15, replace=T)

# Call lolliplot
lolliplot(data, fillCol='gender', labelCol='impact', plot_sidechain=TRUE)
```



The user has the option of plotting an additional track in the area underneath the gene track via the argument `y`. Input for this additional layer consists of a data frame with column names 'transcript_name' and 'amino_acid_change' in p. notation. `lolliplot` will capture the input corresponding to the input given in `x` and plot the subsequent data. Additional fill and label columns are allowed however they must match those variables given in the `fillCol` and `labelCol` parameters.

```
# Create additional data
data2 <- data[,2:4]

# Call lolliplot
lolliplot(data, y=data2, fillCol='impact', labelCol='amino_acid_change')
```



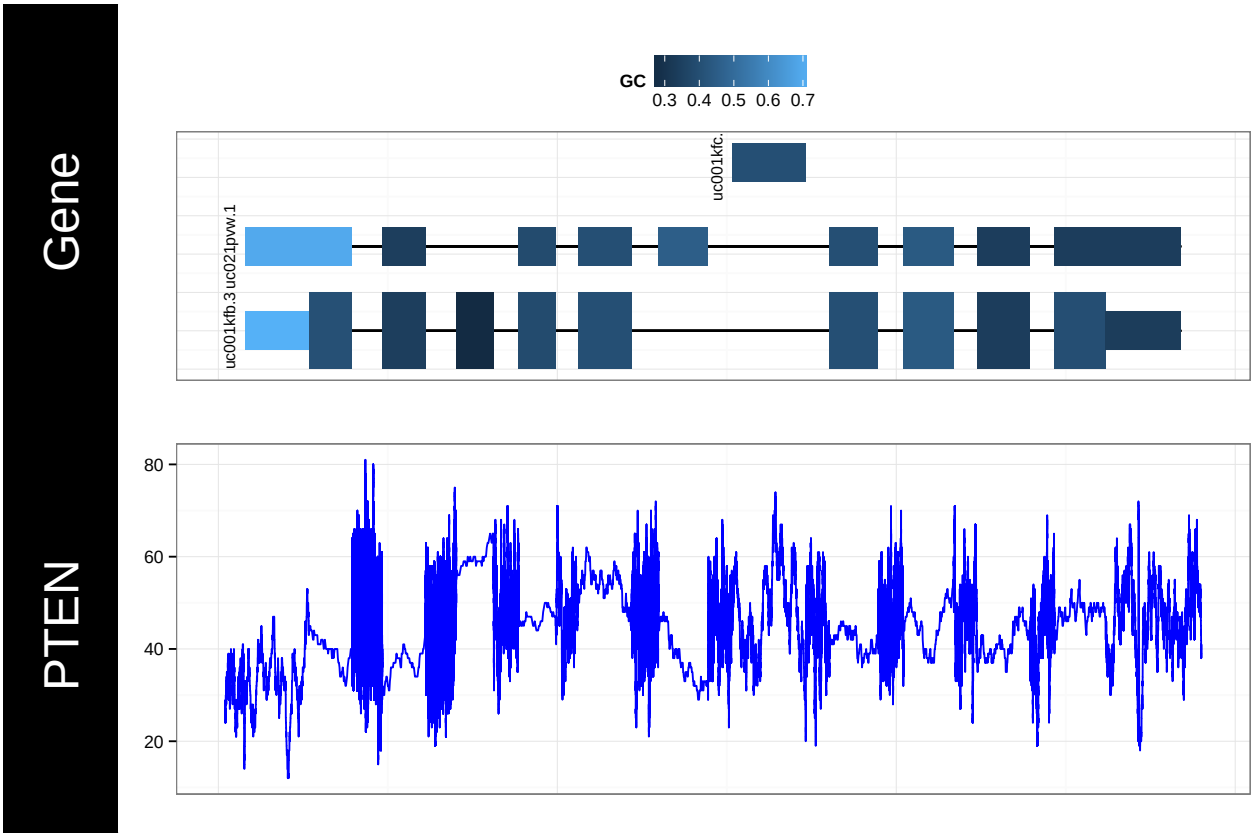
`lolliplot` uses a force field model from the package FField to repulse and attract data in an attempt to achieve a reasonable degree of seperation between points. Suitable defaults have been set for the majority of use cases. On occasion the user may need to manually adjust the force field parameters. This can be done for both upper and lower tracks individually via `rep.fact`, `rep.dist.lmt`, `attr.fact`, `adj.max`, `adj.lmt`, `iter.max` please see documentation for FField::FFieldPtRep for a complete description of these parameters.

genCov

`genCov` provides a methodology for viewing coverage information in relation to a gene track. It takes a named list of data frames with each data frame containing column names "end" and "cov" and rows corresponding to coordinates within the region of interest. Additional required arguments are a Granges object specifying the region of interest, a BSgenome for gc content calculation, and a TxDb object containing transcription metadata (see the package Granges for more information). genCov will plot a genomic features track and align coverage data in the list to the plot. It is recommended to use bedtools multicov to obtain coverage information for a region of interest. We demonstrate `genCov` functionality using the attached data set ptenCOV which contains coverage information for the gene PTEN.

```
# Load transcript meta data
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

# Load BSgenome
library(BSgenome.Hsapiens.UCSC.hg19)
genome <- BSgenome.Hsapiens.UCSC.hg19

# Define a region of interest
gr <- GRanges(seqnames=c("chr10"), ranges=IRanges(start=c(89622195), end=c(89729532)), strand=strand(c(
```

```
# Define preloaded coverage data as named list
data <- list("PTEN" = ptenCOV)

# Call genCov
genCov(data, txdb, gr, genome, gene.transcript_name_size=3)
```



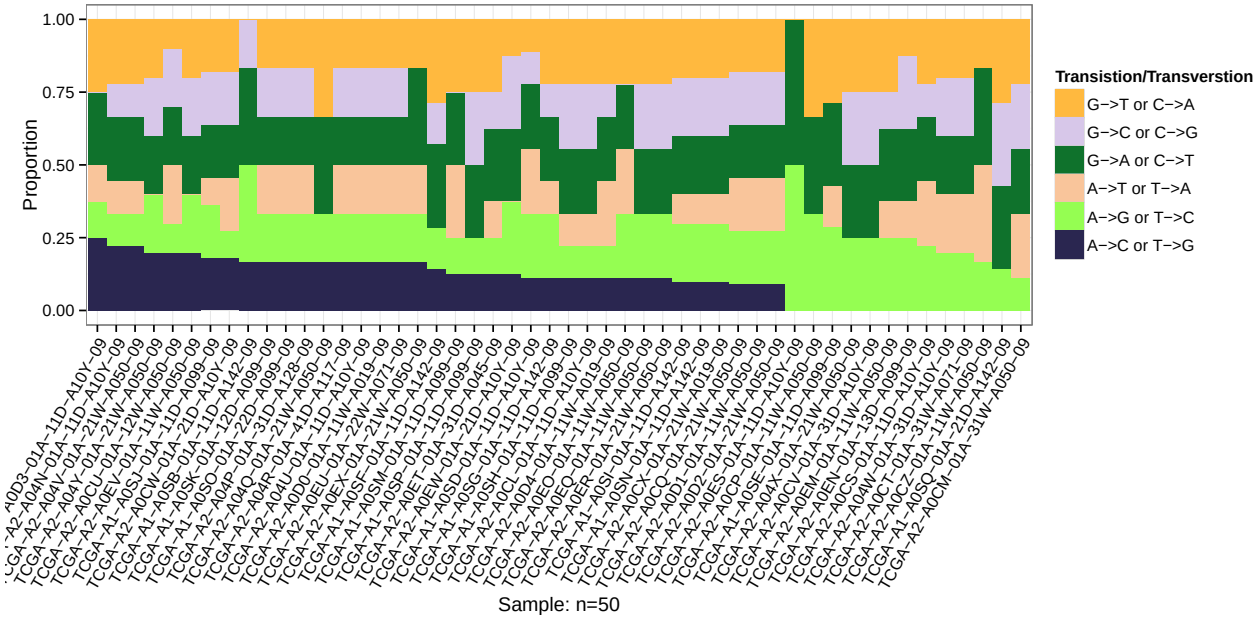
By default genCov will perform a log compression of genomic space for each feature type,'Intron','CDS','UTR'. The degree of compression can be set via the parameter `base` which will perform the appropriate log compression for the features specified in `transform`. The user can turn off this behavior by setting transform to `NULL`. Defaults to log-10 compression for intronic space, and log-2 compression for CDS and UTR regions. Here we turn off the compression and reduce all gene transcripts to one representative pseudo-transctipt.

```
# Turn off feature compression and reduce gene transcripts
# Need to fix, reduce functionality is broken
#genCov(data, txdb, gr, genome, transform=NULL, reduce=TRUE)
```

TvTi

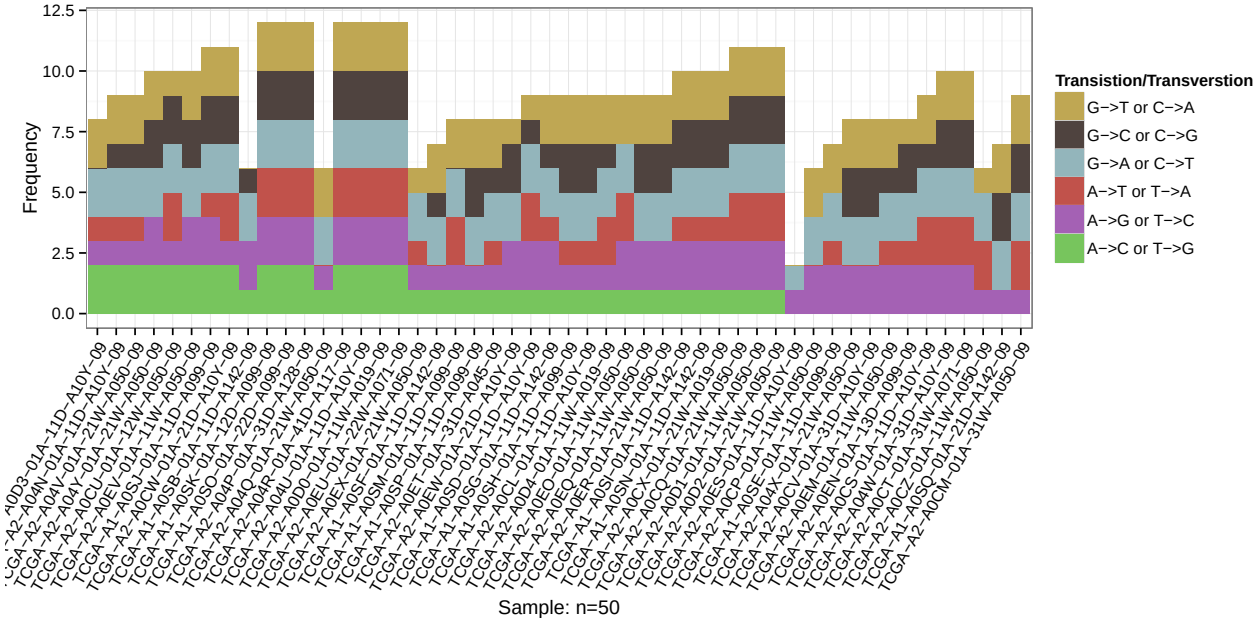
`TvTi` provides a framework for visualizing transversions and transitions for a given cohort. Input consists of a .maf (version 2.4) file containing sample and allele information (see .maf spec). Alternativley the `file_type` parameter can be set to "MGI" with the input supplied consisting of a data frame with column names "sample", "reference", and "variant". TvTi will remove indels and multinucleotide calls from the input and plot the proportion of Transition/Transversion types for each sample specified in the input file.

```
# Call TvTi
TvTi(brcaMAF, x_axis_text_angle=60)
```



`TvTi` will also plot the observed frequency of each Transition/Transversion type in lieu of proportion if the `type` parameter is set to "Frequency". Here we plot the observed frequency from `brcaMAF` and change the default colors of the plot. When modifying the color palette via the `palette` parameter specify a character vector of length 6 contaning a new color for each Transition/Transversion type.

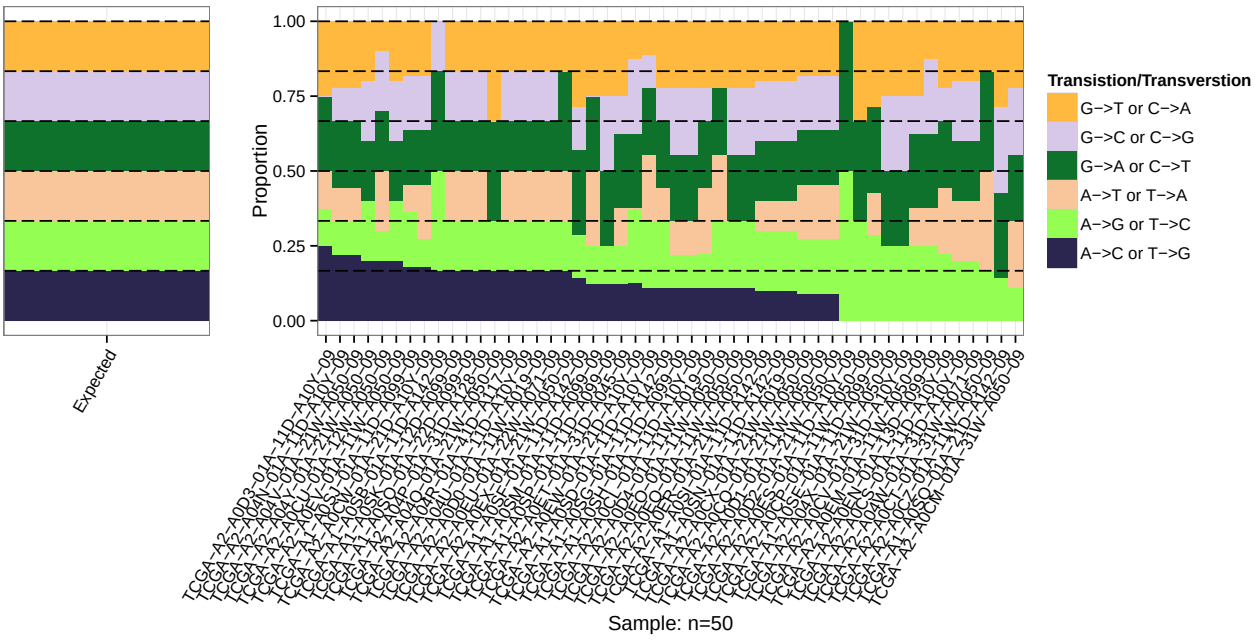
```
# Plot the frequency with a different color pallete
TvTi(brcaMAF, type='Frequency', palette=c("#77C55D", "#A461B4", "#C1524B", "#93B5BB", "#4F433F", "#BFA7
```



If there are prior expectations about the transition/transversion rate the user can specify that information via the argument `y` which takes a named vector with names corresponding to each transition/transversion

type. The vector must be of length 6 with names "A->C or T->G", "A->G or T->C", "A->T or T->A", "G->A or C->T", "G->C or C->G", and "G->T or C->A". The Resultant plot will contain an additional subplot corresponding to the apriori expectations.

```
# Create a named vector of apriori expectations
expec <- c("A->C or T->G"=1/6, "A->G or T->C"=1/6, "A->T or T->A"=1/6, "G->A or C->T"=1/6, "G->C or C->

# Call TvTi with the additional data
TvTi(brcaMAF, expec, x_axis_text_angle=60)
```

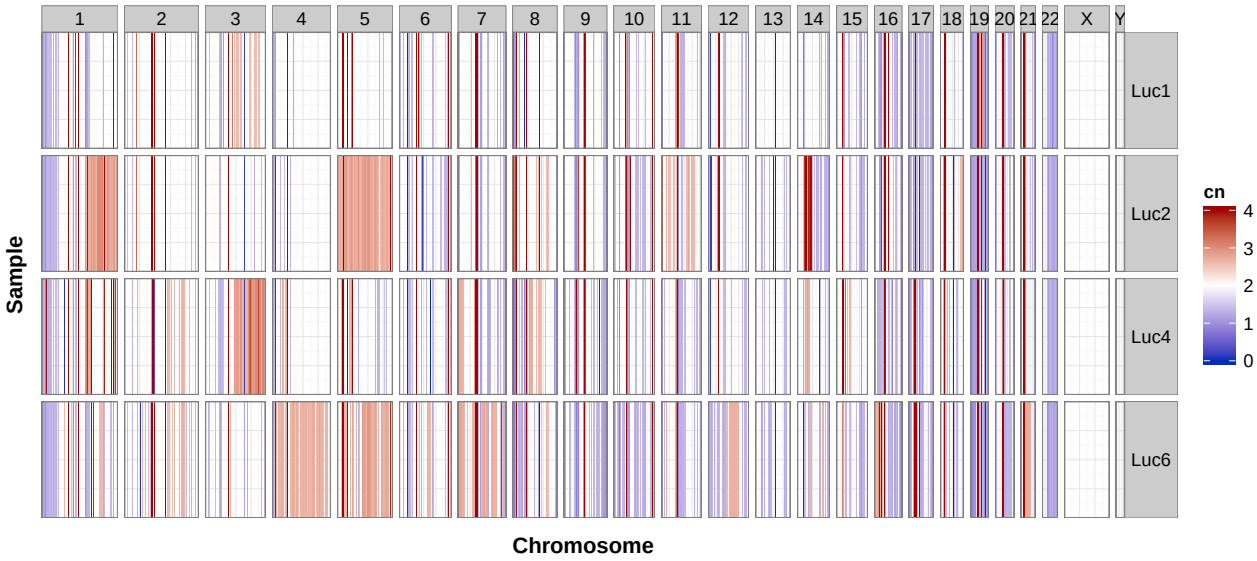


cnSpec

cnSpec produces a plot displaying copy number segments at a cohort level. Basic input consists of a data frame with column names 'chromosome', 'start', 'end' 'segmean' and 'sample' with rows denoting segments with copy number alterations. A UCSC genome is also required, defaults to 'hg19', to determine chromosomal boundaries. cnSpec will produce a grid faceted on chromosome and sample displaying all CN segment calls in the input. Here we use the attached data set LucCNseg containing copy number segment calls for 4 samples from whole genome sequencing data.

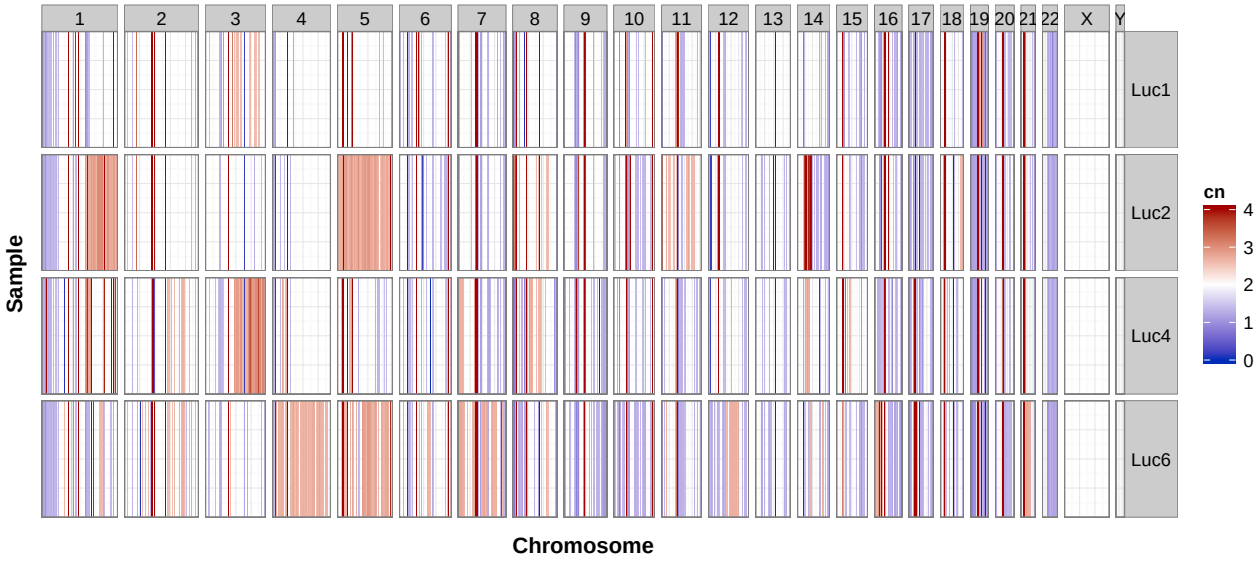
```
# Call cnSpec with minimum required inputs
cnSpec(LucCNseg, genome="hg19")
```

```
## genome specified is preloaded, retrieving data...
```



By default a few select genomes are attached as part of GenVisR, these are "hg38", "hg19", "mm10", "mm9", "rn5". If input into `genome` is not one of the previously mentioned genomes cnSpec will attempt to query the UCSC sql database to obtain chromosomal boundary information. This has been built in as a convenience, if internet connectivity is an issue, or if copy number segment calls are derived from an assemlby not supported by UCSC the user can specify chromosomal boundaries via the argument `y`. This should take the form of a data frame with column names "chromosome", "start", "end". An example of this is provided in the attached data set hg19chr.

```
# Call cnSpec with the y parameter
cnSpec(LucCNseg, y=hg19chr)
```

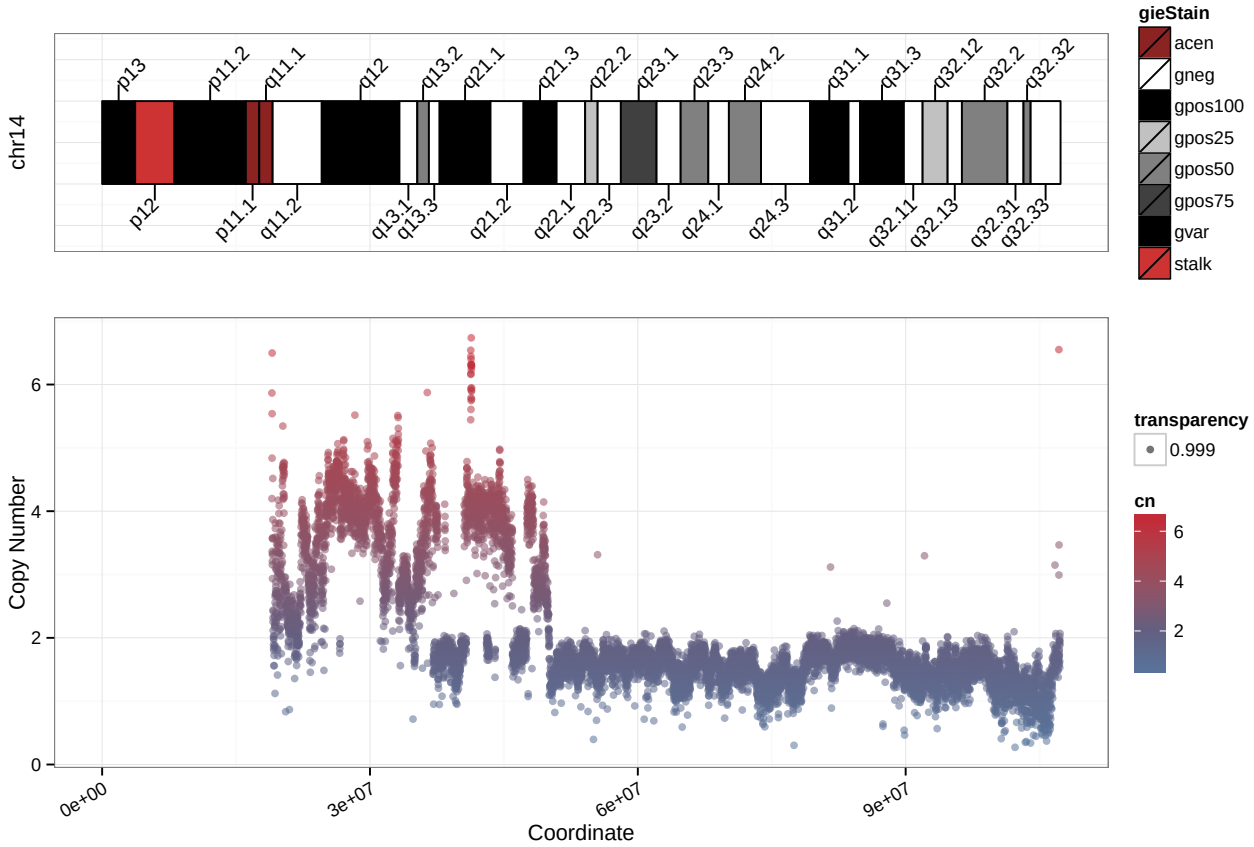


cnView

cnView provides a method for visualizing raw copy number calls focused on either a single chromosome or all chromosomes. Unlike the majority of plots within GenVisR cnView is intended to be used for a single sample. Input consists of a data frame with column names "chromosome", "coordinate", "cn", and "p_value"(optional) as well as a specification of which chromosome to plot `chr` and which genome assembly should be used for

genome. The algorithm will produce an ideogram on the top track and plot copy number calls beneath. If a "p_value" column is present in the input data cnView will create a transparency for all calls/observations based on that column. Eliminating the "p_value" column will terminate this behavior. Here we demonstrate cnView with raw copy number calls for chromosome 14 from the attached data set Luc2CNraw.

```
# Call cnView with basic input
cnView(Luc2CNraw, chr='chr14', genome='hg19', ideo.chr_txt_size=4)
```

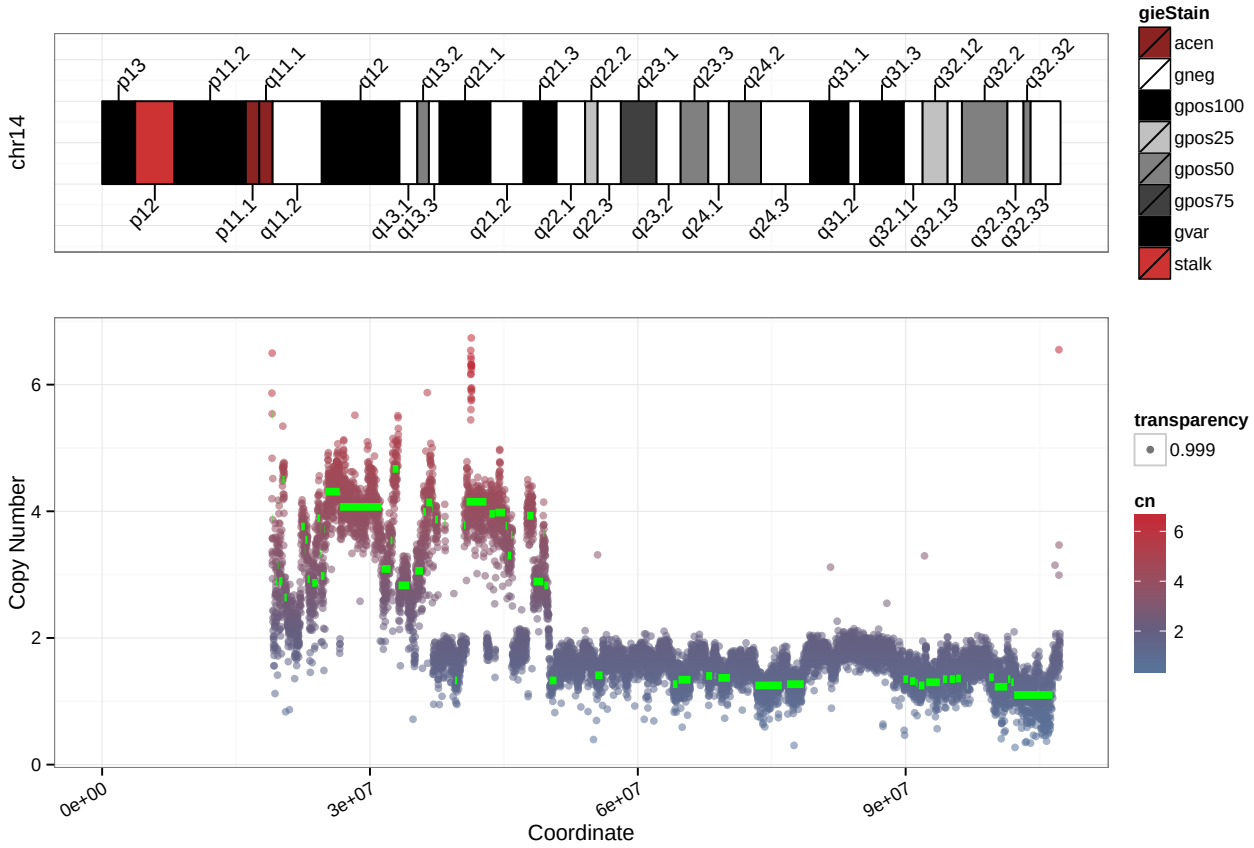


NULL

cnView obtains ideogram information and chromosomal boundaries either via a preloaded genome or the UCSC sql database if it is available. In the interest of flexibility the user also has the option of specifying cytogenetic information to the argument y. This input should take the form of a data frame with column names "chrom", "chromStart", "chromEnd", "name", "gieStain". This format mirrors what is retrievable via the afore mentioned database.

If it is desired, cnView has the ability to overlay segment calls on the plot. This is achieved by providing a data frame with column names: "chromosome", "start", "end", and "segmean" to the argument z.

```
# Obtain CN segments for Luc2 corresponding to chromosome 14
CNseg <- LucCNseg[LucCNseg$sample == 'Luc2' & LucCNseg$chromosome == 14,]

# call cnView with segment data
cnView(Luc2CNraw, z=CNseg, chr='chr14', genome='hg19', ideo.chr_txt_size=4)
```

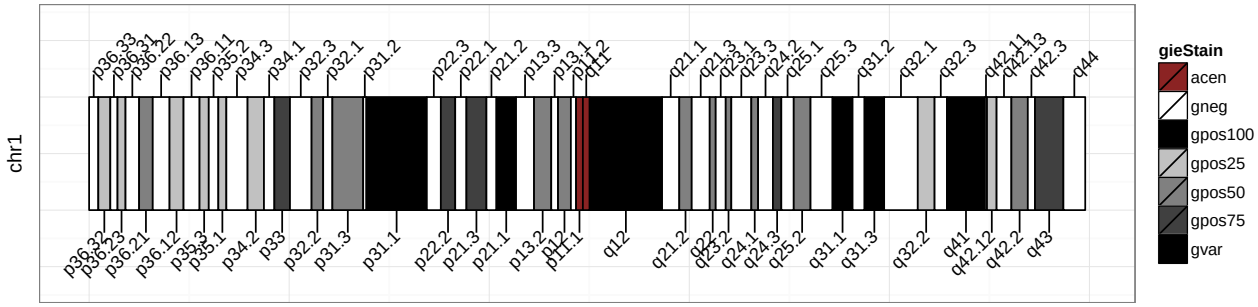


NULL

ideoView

The user has the ability to plot an ideogram representative of the chromosome of interest for a given assembly via the function `ideoView`. Basic input consists of a data frame with column names: "chrom", "chromStart", "chromEnd", "name", "gieStain" mirroring the format retrievable from the UCSC sql database, and a chromosome for which to display `chromsome`. Here we use the preloaded genome hg38 in the attached data set cytoGeno.

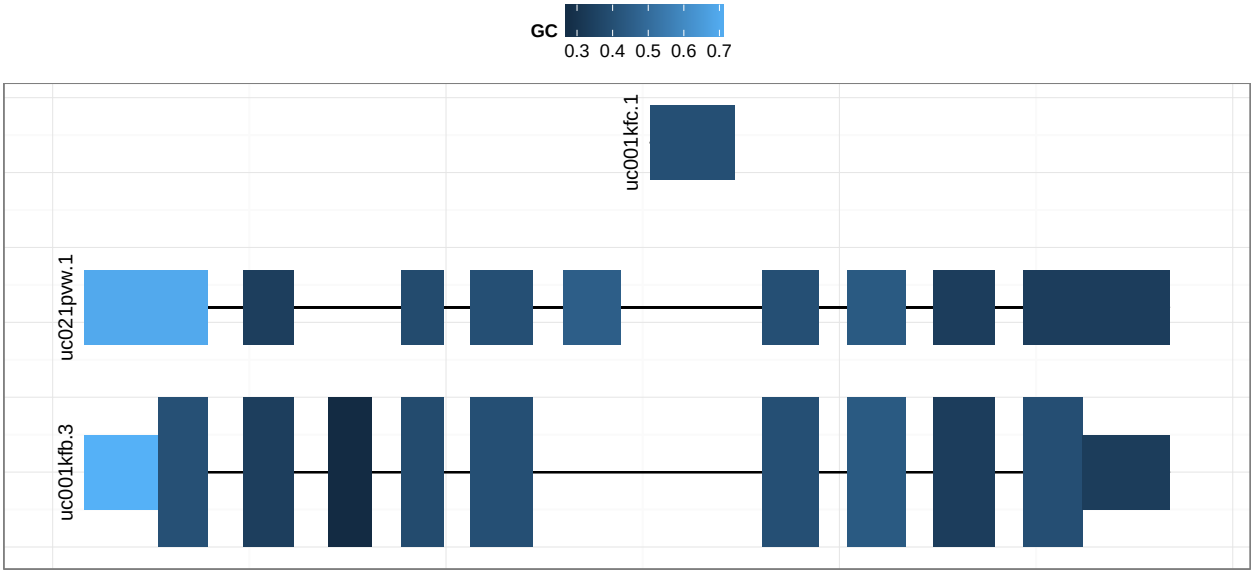
```
# Obtain cytogenetic information for the genome of interest
data <- cytoGeno[cytoGeno$genome == 'hg38',]

# Call ideoView for chromosome 1
ideoView(data, chromosome='chr1', chr_txt_size=4)
```



geneViz

It is also possible to plot just a gene of interest identified by specifying a Txdb object, GRanges object, and a BSgenome via a call to `geneViz`. The algorithm will plot genomic features for a single gene bounded by the Granges object overlaying gc content calculations over those features obtained from the provided BSgenome.

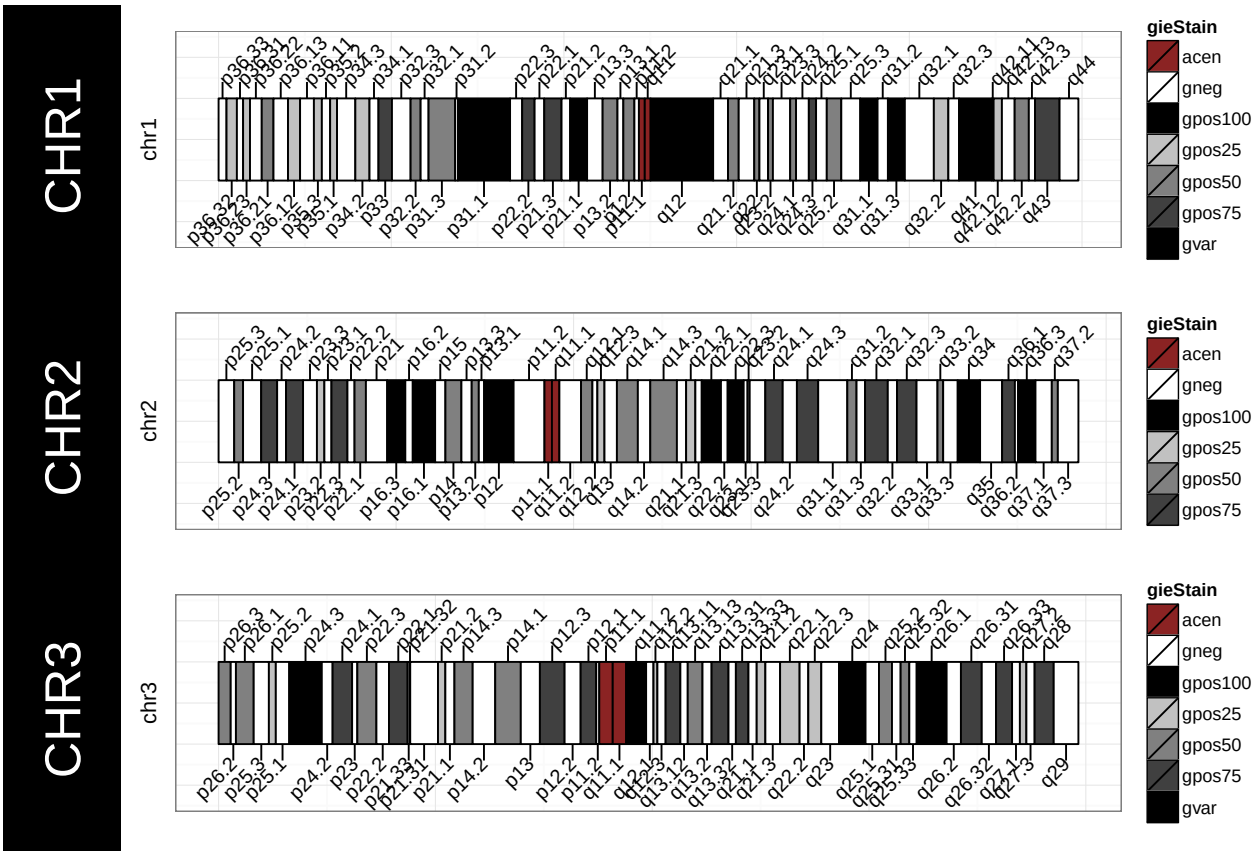
```
# need transcript data for reference
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

# need a biostrings object for reference
genome <- BSgenome.Hsapiens.UCSC.hg19

# need Granges object
gr <- GRanges(seqnames=c("chr10"), ranges=IRanges(start=c(89622195), end=c(89729532)), strand=strand(c(

# Plot the graphic
geneViz(txdb, gr, genome)
```



trackViz

Often it is usefull to plot multiple graphics in the same window. `trackViz` provides a mechanism for accomplishing this using a named list. The names in the list will be displayed as labels. It should be noted that elements within the list should be ggplot2 objects.

```
# Obtain cytogenetic information for the genome of interest
data <- cytoGeno[cytoGeno$genome == 'hg38',]

chr1 <- ideoView(data, chromosome='chr1', chr_txt_size=4)
chr2 <- ideoView(data, chromosome='chr2', chr_txt_size=4)
chr3 <- ideoView(data, chromosome='chr3', chr_txt_size=4)

data <- list("CHR1" = chr1, "CHR2" = chr2, "CHR3" = chr3)

trackViz(data, list=TRUE)
```



NULL

Hints

Due to the complex nature and variablity of the graphics produced by GenVisR it is recommended that the user adjust the graphics device size for all outputs manually. If not given enough space within the graphics device grob objects will start to collide This can be done via the following:

```
pdf(file="plot.pdf", height=8, width=14)
# Call a GenVisR function
mutSpec(brcaMAF)
dev.off()
```

For the majority of plots there is a layer parameter, this allows the user to specify additional ggplot2 layers.