Introduction to the GRIN2 Package

Abdelrahman Elsayed, PhD and Stanley Pounds, PhD

2025-06-23

Introduction

The **GRIN2** package is an improved version of GRIN software that streamlines its use in practice to analyze genomic lesion data, accelerate its computing, and expand its analysis capabilities to answer additional scientific questions including a rigorous evaluation of the association of genomic lesions with RNA expression.

T-ALL Example dataset

- Genomic Landscape of T-ALL (https://pubmed.ncbi.nlm.nih.gov/28671688/)
- RNA-seq and WES data for 265 patients identified 6,861 genomic lesions
- Clinical outcome data

1) Obtain clinical, lesion and gene expression data

```
data(clin_data)
data(lesion_data)
data(expr_data)

# Please make sure that coordinates in the lesion data file (loc.start and loc.end) are ba
sed on GRCh38 (hg38) genome assembly. Multiple tools that include the UCSC LiftOver tool
(<https://genome.ucsc.edu/cgi-bin/hgLiftOver>) can be used for such conversions.

head(lesion_data)
#>       ID chrom loc.start   loc.end lsn.type
#> 1 PARFIH    16  67616879  67616879 mutation
#> 2 PARFIH     3  49722238  49722238 mutation
#> 3 PARFIH     1   6253852   6253852 mutation
#> 4 PARFIH     1 235766122 235766122 mutation
#> 5 PARFIH     4 141132499 141132499 mutation
#> 6 PARFIH     9  27206739  27206739 mutation

# Specify a folder on your local machine to store the analysis results:
# resultsPath=tempdir()
# knitr::opts_knit$set(root.dir = normalizePath(path = resultsPath))
```

2) Retrieve Genomic Annotations for Genes and Regulatory Features

```
# A subset (417 genes) retrieved from ensembl BioMart will be used as an example gene anno
tation data file:
data(hg38_gene_annotation)

# To retrieve a complete annotation data for genes and regulatory features from ensembl Bi
oMart, users can use get.ensembl.annotation function:
# hg38.ann <- get.ensembl.annotation("Human_GRCh38")

head(hg38_gene_annotation)
#>                 gene chrom loc.start   loc.end
#> 20  ENSG00000131697     1   5862811   5992473
#> 123 ENSG00000162607     1  62436297  62451804
#> 280 ENSG00000143653     1 246724409 246768137
#> 376 ENSG00000136643     1 213051233 213274774
#> 387 ENSG00000035687     1 244408494 244451909
#> 431 ENSG00000142676     1  23691742  23696835
#>                                                                  description
#> 20                          nephrocystin 4 [Source:HGNC Symbol;Acc:HGNC:19104]
#> 123         ubiquitin specific peptidase 1 [Source:HGNC Symbol;Acc:HGNC:12607]
#> 280 saccharopine dehydrogenase (putative) [Source:HGNC Symbol;Acc:HGNC:24275]
#> 376         ribosomal protein S6 kinase C1 [Source:HGNC Symbol;Acc:HGNC:10439]
#> 387              adenylosuccinate synthase 2 [Source:HGNC Symbol;Acc:HGNC:292]
#> 431                  ribosomal protein L11 [Source:HGNC Symbol;Acc:HGNC:10301]
#>     gene.name        biotype chrom.strand chrom.band
#> 20      NPHP4 protein_coding           -1     p36.31
#> 123      USP1 protein_coding            1      p31.3
#> 280    SCCPDH protein_coding            1        q44
#> 376   RPS6KC1 protein_coding            1      q32.3
#> 387     ADSS2 protein_coding           -1        q44
#> 431     RPL11 protein_coding            1     p36.11
```

3) Retrieve Chromosome Size Data

```
# chromosome size data:
head(hg38_chrom_size)
#>   chrom      size
#> 1     1 248956422
#> 2     2 242193529
#> 3     3 198295559
#> 4     4 190214555
#> 5     5 181538259
#> 6     6 170805979

# chromosome size data can be also retrieved for GRCh38 (hg38) genome build from chr.info
txt file available on UCSC genome browser:
```

4) Run Genomic Random Interval (GRIN) Analysis

```
# grin.stats is the function that can be used to excute the GRIN statistical framework:
grin.results=grin.stats(lesion_data,
                        hg38_gene_annotation,
                        hg38_chrom_size)

# In addition to annotated genes, users can run GRIN analysis on regulatory sequences from
ensembl regulatory build and FANTOM5 project.
```

5) Now, let’s Take a Look on the GRIN Output Results:

```
# Extract GRIN results table:
grin.table=grin.results$gene.hits
sorted.results <- grin.table[order(as.numeric(as.character(grin.table$p2.nsubj))),]
```

First section of GRIN results table will include gene annotation in addition to the number of subjects affected by each type of lesions:

```
head(sorted.results[,c(7,11:14)])
#>       gene.name nsubj.fusion nsubj.gain nsubj.loss nsubj.mutation
#> 2760       PTEN            0          2         23             37
#> 16361       MYB            4         26          2             13
#> 4114       ETV6            2          0         15              7
#> 4755     CDKN1B            0          0         20              4
#> 2818        WT1            0          0          9             24
#> 19924     DDX3X            2          1          2              4
```

Results will also include the probability (p) and FDR adjusted q-value for each gene to be affected by each type of lesion:

```
head(sorted.results[,c(7,19:22)])
#>       gene.name q.nsubj.fusion q.nsubj.gain q.nsubj.loss q.nsubj.mutation
#> 2760       PTEN   1.000000e+00  1.00000e+00 6.962771e-35     1.544248e-76
#> 16361       MYB   3.147439e-09  3.00149e-50 9.120621e-01     7.453079e-27
#> 4114       ETV6   5.973467e-03  1.00000e+00 1.044321e-12     1.485813e-06
#> 4755     CDKN1B   1.000000e+00  1.00000e+00 1.089932e-21     4.375947e-06
#> 2818        WT1   1.000000e+00  1.00000e+00 1.644414e-06     6.329540e-50
#> 19924     DDX3X   5.527172e-05  1.00000e+00 9.120621e-01     1.554019e-06
```

Another important part of the output is the constellation results testing if the gene is affected by one type of lesions (p1.nusubj) or a constellation of two types of lesions (p2.nsubj), three types of lesions (p3.nsubj), etc.. with FDR adjusted q-values added to the table as well:

```
head(sorted.results[,c(7,27:30)])
#>       gene.name     q1.nsubj     q2.nsubj     q3.nsubj q4.nsubj
#> 2760       PTEN 1.366953e-76 2.096383e-69 1.000000e+00        1
#> 16361       MYB 4.981666e-51 5.754654e-53 4.590287e-29        1
#> 4114       ETV6 1.530741e-12 9.529379e-12 1.255076e-09        1
#> 4755     CDKN1B 8.925351e-22 7.501134e-11 1.000000e+00        1
#> 2818        WT1 6.788063e-50 8.984019e-11 1.000000e+00        1
#> 19924     DDX3X 7.495956e-07 2.637376e-10 1.000000e+00        1
```

The second part of the results table report the same set of results but for the number of hits affecting each gene for each lesion type instead of the number of unique affected subjects. For example, if NOTCH1 gene is affected by 4 mutations in the same subject, this event will be counted as 4 hits in the n.hits stats but 1 subject in the n.subj stats:

```
head(sorted.results[,c(7,31:34)])
#>       gene.name nhit.fusion nhit.gain nhit.loss nhit.mutation
#> 2760       PTEN           0         2        39            45
#> 16361       MYB           4        27         2            14
#> 4114       ETV6           2         0        16             7
#> 4755     CDKN1B           0         0        20             4
#> 2818        WT1           0         0         9            27
#> 19924     DDX3X           2         1         2             5
```

6) Write GRIN Results

```
# write.grin.xlsx function return an excel file with multiple sheets that include GRIN res
ults table, interpretation of each column in the results, and methods paragraph
# write.grin.xlsx(grin.results, "T-ALL_GRIN_result_annotated_genes.xlsx")

# To return the results table without other information (will be helpful in case of large
lesion data files where the gene.lsn.data sheet will be > 1 million rows that halt the wri
te.grin.xlsx function).
grin.res.table=grin.results$gene.hits
```

7) Genome-wide Lesion Plot

```
genomewide.plot=genomewide.lsn.plot(grin.results,
                                    max.log10q=50)
```

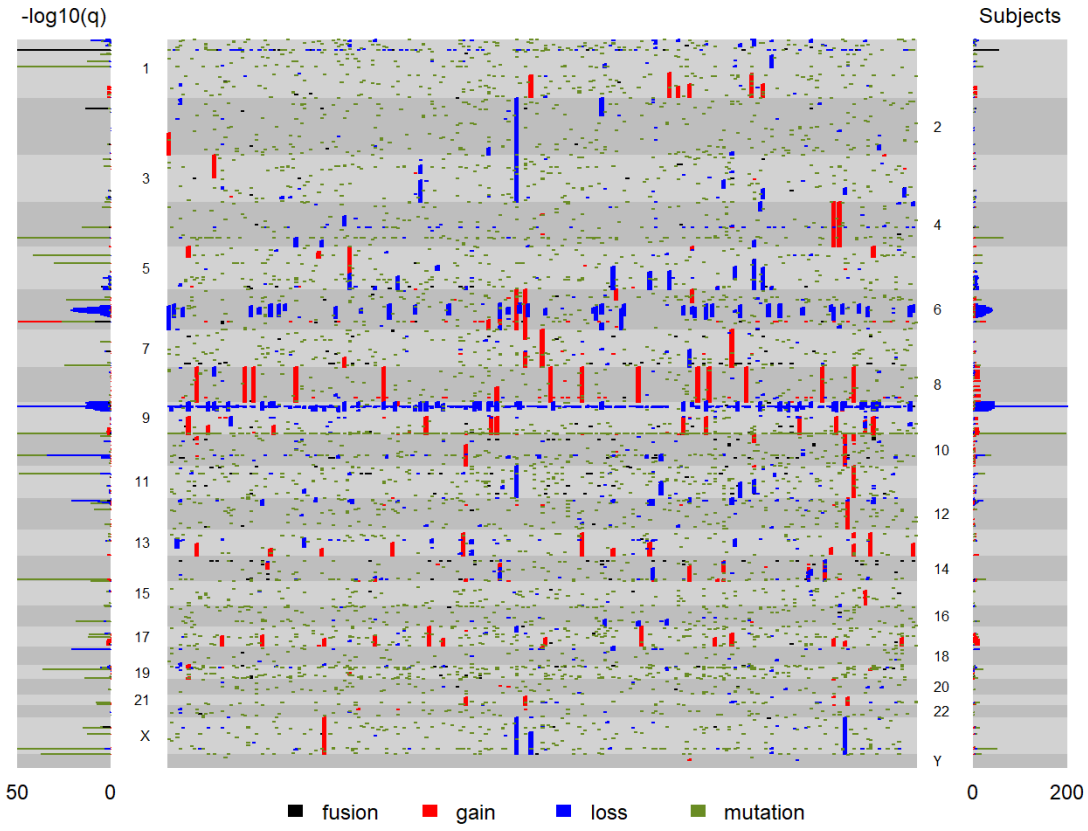

Figure 1. Genome-wide lesion plot

```
# This function use the list of grin.results
```

8) Stacked Barplot for a List of Genes of Interest

```
# This barplot shows the number of patients affected by different types of lesions in a li
st of genes of interest:
count.genes=as.vector(c("CDKN2A", "NOTCH1", "CDKN2B", "TAL1", "FBXW7", "PTEN", "IRF8",
                        "NRAS", "BCL11B", "MYB", "LEF1","RB1", "MLLT3", "EZH2", "ETV6",
                        "CTCF", "JAK1", "KRAS", "RUNX1", "IKZF1", "KMT2A", "RPL11", "TCF
7",
                        "WT1", "JAK2", "JAK3", "FLT3"))

# return the stacked barplot
grin.barplt(grin.results,
            count.genes)
```

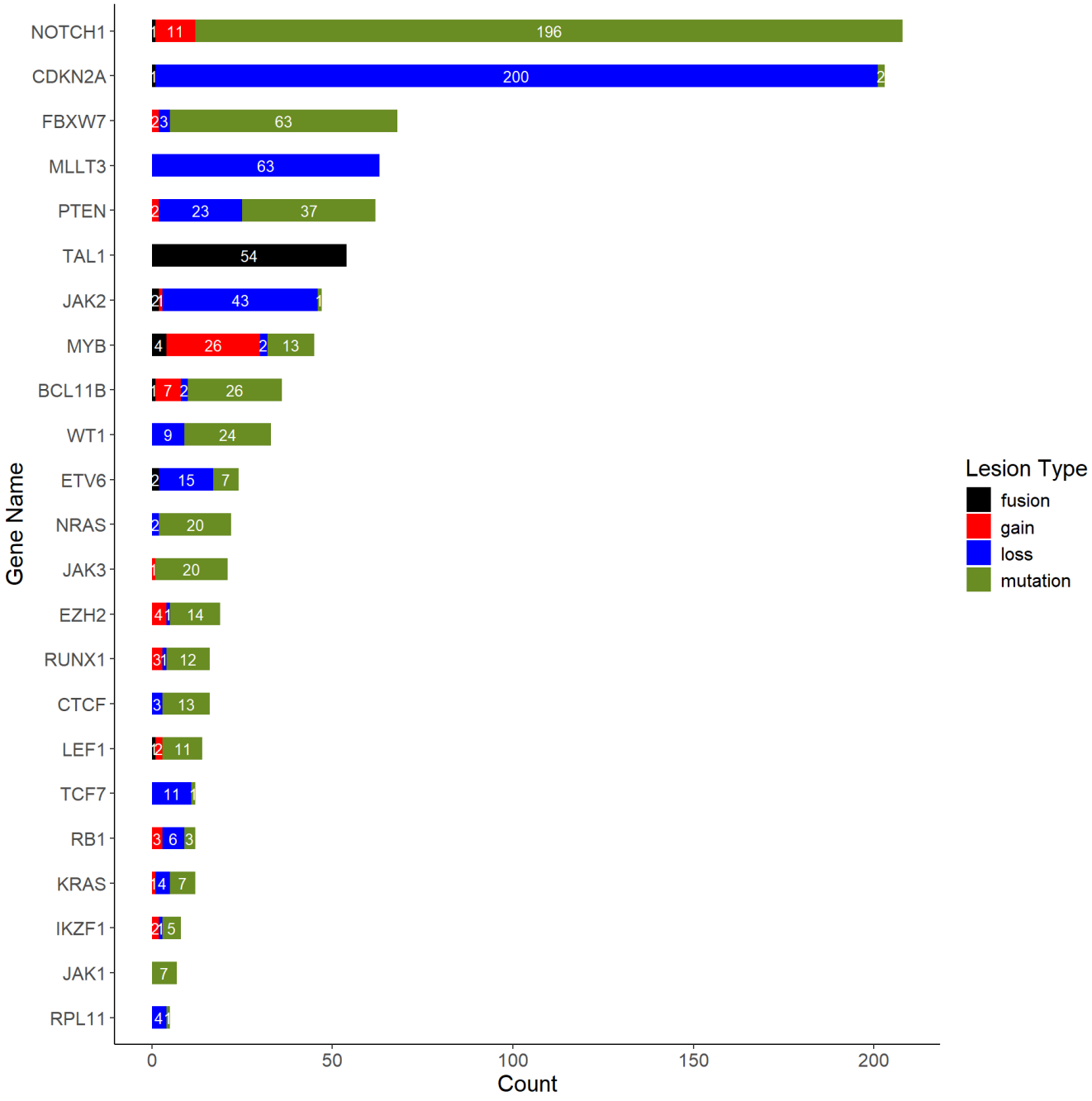

Figure 2. stacked barplot with number of patients affected by different types of lesions in a list of genes of interest

9) Prepare GRIN Lesion Matrix for an OncoPrint Type of Display

```
# First identify the list of genes to be included in the oncoprint:
oncoprint.genes=as.vector(c("ENSG00000101307", "ENSG00000171862", "ENSG00000138795",
                            "ENSG00000139083", "ENSG00000162434", "ENSG00000134371",
                            "ENSG00000118058", "ENSG00000171843", "ENSG00000139687",
                            "ENSG00000184674", "ENSG00000118513", "ENSG00000197888",
                            "ENSG00000111276", "ENSG00000258223", "ENSG00000187266",
                            "ENSG00000174473", "ENSG00000133433", "ENSG00000159216",
                            "ENSG00000107104", "ENSG00000099984", "ENSG00000078403",
                            "ENSG00000183150", "ENSG00000081059", "ENSG00000175354",
                            "ENSG00000164438"))

# Prepare a lesion matrix for the selected list of genes with each row as a gene and each
column is a patient (this matrix is compatible with oncoPrint function in ComplexHeatmap p
ackage):
oncoprint.mtx=grin.oncoprint.mtx(grin.results,
                                 oncoprint.genes)

head(oncoprint.mtx[,1:6])
#>        PASGFH    PASKSY    PASTDU PASYWF PATDRC PATFWF
#> TCF7    loss;     loss; mutation;  loss;  loss;  loss;
#> KANK1             gain;
#> CDKN1B  loss;     loss;                   loss;  loss;
#> MYB           mutation;
#> LEF1
#> ETV6    loss;     loss;                   loss;  loss;

# Use onco.print.props function to specify a height proportion for each lesion category:
onco.props<-onco.print.props(lesion.data,
                             hgt = c("gain"=5, "loss"=4, "mutation"=2, "fusion"=1))
#> Error: object 'lesion.data' not found
column_title = "" # optional


# use oncoprint function from ComplexHeatmap library to plot the oncoprint:
```

10) Lesion Plots for a Certain Gene, Locus or the Whole Chromosome

```
# First call the data for "hg38_cytoband":
data(hg38_cytoband)

# Plots Showing a Selected Type of Lesion (Ex: deletions) Affecting a region of Interest:
cdkn2a.locus=lsn.transcripts.plot(grin.results, transTrack = FALSE,
                                  hg38.cytoband=hg38_cytoband, chrom=9,
plot.start=19900000, plot.end=25600000,
lesion.grp = "loss", spec.lsn.clr = "blue")
#> Error in lsn.transcripts.plot(grin.results, transTrack = FALSE, hg38.cytoband = hg38_cy
toband, : The following Bioconductor package(s) are required but not installed: Gviz, grid
Graphics, GenomeInfoDb, ensembldb.
#> Please install them using:
#> BiocManager::install(c("Gviz", "gridGraphics", "GenomeInfoDb", "ensembldb"))

# This type of lesion plots can be also prepared for a certain gene or locus of interest w
ith transcripts track added using the same lsn.transcripts.plot function.
```

```
# Plots Showing Different Types of Lesions Affecting the whole chromosome:
chrom.plot=lsn.transcripts.plot(grin.results, transTrack = FALSE,
                                  hg38.cytoband=hg38_cytoband, chrom=9,
plot.start=1, plot.end=141000000)
#> Error in lsn.transcripts.plot(grin.results, transTrack = FALSE, hg38.cytoband = hg38_cy
toband, : The following Bioconductor package(s) are required but not installed: Gviz, grid
Graphics, GenomeInfoDb, ensembldb.
#> Please install them using:
#> BiocManager::install(c("Gviz", "gridGraphics", "GenomeInfoDb", "ensembldb"))
```

11) Gene-Lesion Matrix for later computations

```
# Prepare gene and lesion data for later computations
# This lesion matrix has all lesion types that affect a single gene in one row. It can be
used to run association analysis with expression data (part of alex.prep.lsn.expr functio
n)

# First step is to prepare gene and lesion data for later computations
gene.lsn=prep.gene.lsn.data(lesion_data,
                            hg38_gene_annotation)
# Then determine lesions that overlap each gene (locus)
gene.lsn.overlap= find.gene.lsn.overlaps(gene.lsn)
# Finally, build the lesion matrix using prep.lsn.type.matrix function:
gene.lsn.type.mtx=prep.lsn.type.matrix(gene.lsn.overlap,
                                       min.ngrp=5)
# prep.lsn.type.matrix function return each gene in a row, if the gene is affected by mult
iple types of lesions (for example gain AND mutations), entry will be denoted as "multipl
e" for this specific patient.
# min.ngrp can be used to specify the minimum number of patients with a lesion to be inclu
ded in the final lesion matrix.

head(gene.lsn.type.mtx[,1:5])
#>                  PARASZ PARAYM PARCVM PAREGZ PARFDL
#> ENSG00000005700 "loss" "none" "loss" "none" "none"
#> ENSG00000010810 "loss" "none" "loss" "none" "none"
#> ENSG00000014123 "loss" "none" "loss" "none" "none"
#> ENSG00000056972 "loss" "none" "loss" "none" "none"
#> ENSG00000057663 "loss" "none" "loss" "none" "none"
#> ENSG00000065615 "loss" "none" "loss" "none" "none"
```

Associate Lesions with EXpression (ALEX)

12) Prepare Expression and Lesion Data for ALEX-KW Test and ALEX-plots

```
# alex.prep.lsn.expr function prepare expression, lesion data and return the set of genes
with both types of data available ordered by gene IDs in rows and patient IDs in columns:

alex.data=alex.prep.lsn.expr(expr_data,
                             lesion_data,
                             hg38_gene_annotation,
                             min.expr=1,
                             min.pts.lsn=5)

# ALEX ordered lesion data:
alex.lsn=alex.data$alex.lsn
head(alex.lsn[,1:5])
#>                 PARASZ PARAYM PARCVM PAREGZ PARFDL
#> ENSG00000005339   none   none   none   none   none
#> ENSG00000005700   loss   none   loss   none   none
#> ENSG00000006283   none   none   none   none   none
#> ENSG00000010438   none   loss   none   none   none
#> ENSG00000010810   loss   none   loss   none   none
#> ENSG00000014123   loss   none   loss   none   none

# ALEX ordered expression data:
alex.expr=alex.data$alex.expr
head(alex.expr[,1:5])
#>                 PARASZ PARAYM PARCVM PAREGZ PARFDL
#> ENSG00000005339  4.012  3.718  3.253  2.294  3.568
#> ENSG00000005700  2.503  3.738  3.011  3.524  3.437
#> ENSG00000006283  0.035  0.000  0.016  0.005  0.043
#> ENSG00000010438  0.155  0.000  0.454  0.000  0.153
#> ENSG00000010810  3.992  4.402  3.985  3.934  4.443
#> ENSG00000014123  3.009  4.245  3.699  3.032  3.932
```

13) Run Kruskal-Wallis Test for Association between Lesion and Expression Data

```
# KW.hit.express function runs Kruskal-Wallis test for association between lesion groups a
nd expression level of the same corresponding gene:

alex.kw.results=KW.hit.express(alex.data,
                               hg38_gene_annotation,
                               min.grp.size=5)
```

14) Now, let's Take a Look on the ALEX Kruskal-wallis Results Table:

```
# order the genes by the ones with most significant KW q-value:
sorted.kw <- alex.kw.results[order(as.numeric(as.character(alex.kw.results$q.KW))),]
```

First section of the results table will include gene annotation in addition to the kruskal-wallis test p and q values evaluating if there's a statistically significant differences in the gene expression level between different lesion groups:

```
head(sorted.kw[,c(6,7,11,12)])
#>                 gene.name        biotype         p.KW         q.KW
#> ENSG00000198642     KLHL9 protein_coding 7.141766e-21 1.963986e-18
#> ENSG00000120159     CAAP1 protein_coding 2.417600e-20 3.324200e-18
#> ENSG00000162367      TAL1 protein_coding 1.152926e-18 1.056849e-16
#> ENSG00000137073     UBAP2 protein_coding 4.612214e-18 3.170897e-16
#> ENSG00000107185      RGP1 protein_coding 1.465553e-16 8.060539e-15
#> ENSG00000147889    CDKN2A protein_coding 5.650917e-16 2.590004e-14
```

For each gene, results table will include the number of patients affected by each type of lesion in addition to number of patients affected by multiple types of lesions in the same gene and patients without any lesion:

```
head(sorted.kw[,c(13:18)])
#>                 fusion_n.subjects gain_n.subjects loss_n.subjects
#> ENSG00000198642                 0               0              99
#> ENSG00000120159                 0               1              44
#> ENSG00000162367                54               0               0
#> ENSG00000137073                 0               0              36
#> ENSG00000107185                 0               0              36
#> ENSG00000147889                 1               0             199
#>                 multiple_n.subjects mutation_n.subjects none_n.subjects
#> ENSG00000198642                   0                   0            164
#> ENSG00000120159                   0                   0            218
#> ENSG00000162367                   0                   0            209
#> ENSG00000137073                   0                   0            227
#> ENSG00000107185                   0                   0            227
#> ENSG00000147889                   1                   1             61
```

Results table will also include the mean expression of the gene by different lesion groups in addition to the median expression and standard deviation.

```
head(sorted.kw[,c(19:24)])
#>                 fusion_mean gain_mean loss_mean multiple_mean mutation_mean
#> ENSG00000198642          NA        NA 1.8904242            NA            NA
#> ENSG00000120159          NA     3.779 2.6011136            NA            NA
#> ENSG00000162367    3.494315        NA        NA            NA            NA
#> ENSG00000137073          NA        NA 2.6841944            NA            NA
#> ENSG00000107185          NA        NA 1.7349722            NA            NA
#> ENSG00000147889    3.215000        NA 0.3813266         2.425         4.571
#>                 none_mean
#> ENSG00000198642  3.063372
#> ENSG00000120159  3.309950
#> ENSG00000162367  1.584507
#> ENSG00000137073  3.441229
#> ENSG00000107185  2.534004
#> ENSG00000147889  1.311738
```

15) Waterfall Plots for Side-by-side Representation of Lesion and Expression Data

```
# waterfall plots allow a side-by-side representation of expression and lesion data of the
gene of interest.
# First prepare expression and lesion data for waterfall plots:
WT1.waterfall.prep=alex.waterfall.prep(alex.data,
                                       alex.kw.results,
                                       "WT1",
                                       lesion_data)

# alex.waterfall.plot can be used to return the plot
WT1.waterfall.plot=alex.waterfall.plot(WT1.waterfall.prep,
                                       lesion_data)
```

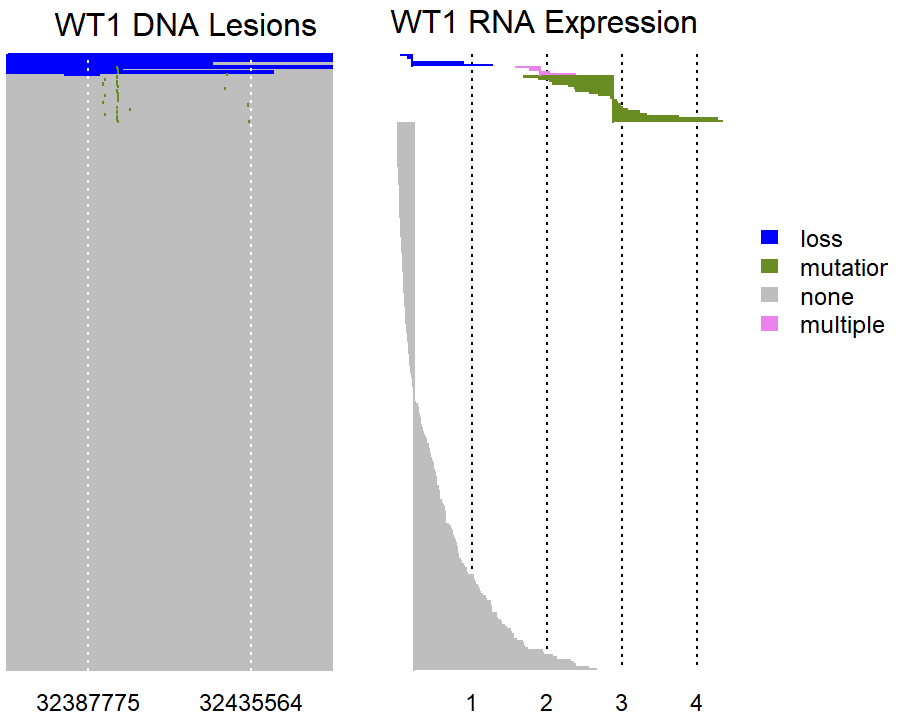

Figure 5. JAK2 Water-fall plot which offers a side-by-side graphical representation of lesion and expression data for each patient

16) Visualize Lesion and Expression Data by Pathway (JAK/STAT Pathway)

```
data("pathways")

# alex.pathway will return two panels figure of lesion and expression data of ordered subj
ects based on the computed lesions distance in all genes assigned to the pathway of intere
st:
alex.path=alex.pathway(alex.data,
                       lsn.data = lesion_data,
                       pathways = pathways,
                       selected.pathway = "Jak_Pathway")
```

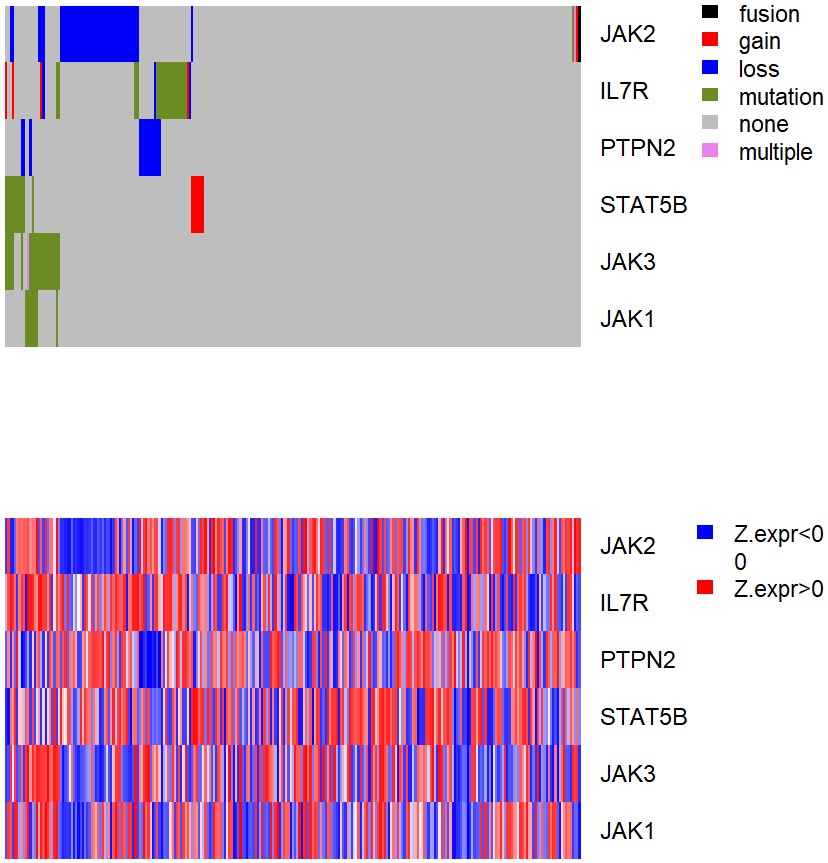

Figure 6. Ordered Lesion and Expression Data based on the Clustering Analysis on the pathway level (JAK/STAT pathway)

```
# To return ordered lesion and expression data of the genes assigned to the pathway of int
erest (same patients order in the plot):
alex.path[1:10,1:5]
#>               PASGFH   PATMRE   PASWXZ   PATZYC   PATBRV
#> JAK2 _lsn       none     none     loss     loss     none
#> JAK3 _lsn   mutation mutation mutation mutation     none
#> JAK1 _lsn       none     none     none     none     none
#> IL7R _lsn       gain     none     none     gain     none
#> STAT5B _lsn mutation mutation mutation mutation mutation
#> PTPN2 _lsn      none     none     none     none     none
#> JAK2 _expr     1.833    3.147    1.542     1.25    2.144
#> JAK3 _expr     3.065    3.264    4.533    3.798     4.45
#> JAK1 _expr     4.314     5.29    5.126    4.192     5.78
#> IL7R _expr     4.025    5.053    6.579    5.496    4.684
```

17) Lesion Binary Matrix for Association Analysis with Clinical Outcomes

```
# This type of lesion matrices with each gene affected by a certain type of lesion in a se
parate row is very helpful to run multiple levels of association analysis that include ass
ociation between lesions and treatment outcomes.

# Users should first Prepare gene and lesion data and determine lesions that overlap each
gene (locus):
gene.lsn=prep.gene.lsn.data(lesion_data,
                            hg38_gene_annotation)
gene.lsn.overlap= find.gene.lsn.overlaps(gene.lsn)

# use prep.binary.lsn.mtx function to prepare the lesion binary matrix:
lsn.binary.mtx.atleast5=prep.binary.lsn.mtx(gene.lsn.overlap,
                                            min.ngrp=5)
# Each row is a lesion type that affect a certain gene for example NOTCH1_mutation (entry
will be labelled as 1 if the patient is affected by by this type of lesion and 0 otherwis
e).
# min.ngrp can be used to specify the minimum number of patients with a lesion to be inclu
ded in the final lesion matrix.

head(lsn.binary.mtx.atleast5[,1:5])
#>                            PATXKW PASHNK PARMUC PATKWU PARXMV
#> ENSG00000005339_mutation      0      1      1      1      1
#> ENSG00000005700_loss          0      0      0      0      0
#> ENSG00000006283_gain          0      0      0      0      0
#> ENSG00000010438_loss          0      0      0      0      0
#> ENSG00000010810_loss          0      0      0      0      0
#> ENSG00000014123_loss          0      0      0      0      0
```

18) Run Association Analysis for Lesions with Clinical Outcomes

```
# Prepare Event-free Survival (EFS) and Overall Survival (OS) as survival objects:
clin_data$EFS <- Surv(clin_data$efs.time, clin_data$efs.censor)
clin_data$OS <- Surv(clin_data$os.time, clin_data$os.censor)

# List all clinical variables of interest to be included in the association analysis:
clinvars=c("MRD.binary", "EFS", "OS")

# Run association analysis between lesions and clinical variables:
assc.outcomes=grin.assoc.lsn.outcome(lsn.binary.mtx.atleast5,
                                     clin_data,
                                     hg38_gene_annotation,
                                     clinvars)

# Optional: Adjust for covariates using the 'covariate' argument
```

19) Now, let's Take a Look on the Results Table of the Association between Lesions and Treatment Outcomes:

```
# order the genes by the ones with most significant KW q-value:
sorted.outcomes <- assc.outcomes[order(as.numeric(as.character(assc.outcomes$`MRD.binary.p
-value`))),]
```

First section of the results table will include gene annotation in addition to the odds ratio, lower95, upper95 confidence intervals in addition to p and FDR adjusted q-values for the logistic regression models testing for the association between lesions and binary outcome variables such as Minimal Residual Disease (MRD). COX proportional hazard models will be used in case of survival objects such as Event-free survival (EFS) and Overall Survival (OS) with hazard ratios reported instead of odds ratio:

```
head(sorted.outcomes[1:7,c(6,11,14,15)])
#>                          gene.name MRD.binary.odds.ratio MRD.binary.p-value
#> ENSG00000147889_loss        CDKN2A             0.2132367       5.889274e-07
#> ENSG00000099810_loss          MTAP             0.2750191       1.458384e-05
#> ENSG00000184937_mutation       WT1             6.8888889       2.698869e-05
#> ENSG00000198642_loss         KLHL9             0.3205882       6.114398e-04
#> ENSG00000171843_loss         MLLT3             0.2253290       1.057842e-03
#> ENSG00000188352_loss         FOCAD             0.3019324       1.326181e-03
#>                          MRD.binary.q-value
#> ENSG00000147889_loss           9.850801e-05
#> ENSG00000099810_loss           1.219696e-03
#> ENSG00000184937_mutation       1.504771e-03
#> ENSG00000198642_loss           2.556839e-02
#> ENSG00000171843_loss           3.538838e-02
#> ENSG00000188352_loss           3.697101e-02
```

Results table will also include the number of patients with/without lesion who experienced or did not experience the event:

```
head(sorted.outcomes[1:7,c(6, 16:19)])
#>                          gene.name MRD.binary.event.with.lsn
#> ENSG00000147889_loss        CDKN2A                         37
#> ENSG00000099810_loss          MTAP                         36
#> ENSG00000184937_mutation       WT1                         16
#> ENSG00000198642_loss         KLHL9                         14
#> ENSG00000171843_loss         MLLT3                          6
#> ENSG00000188352_loss         FOCAD                         10
#>                          MRD.binary.event.without.lsn
#> ENSG00000147889_loss                               33
#> ENSG00000099810_loss                               34
#> ENSG00000184937_mutation                           54
#> ENSG00000198642_loss                               56
#> ENSG00000171843_loss                               64
#> ENSG00000188352_loss                               60
#>                          MRD.binary.no.event.with.lsn
#> ENSG00000147889_loss                              163
#> ENSG00000099810_loss                              154
#> ENSG00000184937_mutation                            8
#> ENSG00000198642_loss                               85
#> ENSG00000171843_loss                               57
#> ENSG00000188352_loss                               69
#>                          MRD.binary.no.event.without.lsn
#> ENSG00000147889_loss                                  31
#> ENSG00000099810_loss                                  40
#> ENSG00000184937_mutation                             186
#> ENSG00000198642_loss                                 109
#> ENSG00000171843_loss                                 137
#> ENSG00000188352_loss                                 125
```

```
library(GRIN2)
```