‘Cellular abundance shapes function in piRNA-guided genome defense’

"Pavol Genzor, Parthena Konstantinidou*, Daniel Stoyko* *equal contributors*
Amirhossein Manzourolajdad, Celine Marlin Andrews, Alexandra R. Elchert, Constantinos Stathopoulos, Astrid D. Haase

This vignette describes the computational materials & methods associated with this manuscript. Please visit **HaaseLab/piRNA__Diversity github repository** to download functions used in the various scripts and analyses. Please refer to the GEO data set **GSE156058** associated with this study for adapter sequences and raw data. The analysis in this vignette was not performed with full data sets, but only subset of the data to demonstrate the materials and methods.

About small RNA Libraries.
To be able to account for the PCR duplication when quantifying individueal piRNA sequences, cloning procedure utilized adapter sequences containing multiple random nucleotides. Each ligated read contains 10 N’s or **U**nique **M**olecular **I**dentifiers **(UMIs)**. There are 8N at the 5-prime and 2N at the 3-prime of the small RNA that allows for 4^10 or 1,048,576 possible combinations.

Library structure: 5-FivePrimeAdapter–*NNNNNNNN*–**smallRNA**–*NN*–ThreePrimeAdapter-3

Pre-requisites
* Acquire the raw sequencing data from your facility or GEO at NCBI * Ensure that you have the appropriate 5-prime and 3-prime adapter sequences * Ensure you have access to computing cluster and a computer with R environment * Ensure that you have all the necessary software installed and running * NOTE: some images may not have rendered perfectly

Vignette Content

DATA PREPARATION

1. Pre-process fastq files.

First, the raw sequencing data was cleaned up with ***cutadapt*** by removing the adapter sequences and keeping the UMIs. Refer to ***cutadapt*** manual for option description. The following commands were executed in the *bash* environment in the terminal. Remember to placeholder variables for your own sequences, directories, and files on you machine.

```
bash$ # initiate appropriate vertion of cutadapt
bash$ module load cutadapt/2.3
bash$
bash$ # split the sequencing lane without trimming
bash$ cutadapt --no-trim -m 19 \
               -g FivePrimeAdapter \
               -a file\:InputFiles/ThreePrimeAdapterSequences.fa \
               -o OutputDirectory/{name}.SNT.fastq.gz \
               --untrimmed-o OutputDirectory/untrimmed.SNT.fastq.gz \
               InputDirectory/SequencingLane_R1_001.fastq.gz
bash$
bash$ # remove the constant adapter sequences
bash$ cutadapt -j 6 --trimmed-only -m 19 \
               -g FivePrimeAdapter \
               -a ThreePrimeAdapter \
               -o OutputDirectory/SampleName.10N.fastq.gz \
               InputDirectory/SampleName.SNT.fastq.gz
bash$
bash$

# SNT = split not trimmed, 10N = 10 UMIs
```

2. Generate unique sequence fasta files

Next, the small RNAs with UMIs can be cleared of PCR duplication. Full script of all functions can be found at ***piRNA__Diversity github***. In the next few chunks we will:

- initiate R session, and load custom function
- remove PCR duplicates using UMIs
- remove UMIs
- export the cleaned reads into **.fasta**

```
# NOTE: If you are working on small files or have a lot of memory,
#       this can be run on laptop

bash$ # initiate an R session
bash$ module load R/4.0.5
```

```
bash$
bash$ ## enter R environment
bash$ R
```

```
# NOTE: Below script sets up the directories and loads the functions

# IMPORTANT: directory on your machine with all functions in this vignette
#            downloaded from github
FUN.DIR <- "/Users/YourUsername/YourDiversityScriptsFolder/"

# File directories
FASTQ.FILE = "SampleName_conditionA_replicate1.SNT.fastq.gz"
OUTPUT.DIR = "/YourComputer/WorkingDirectory/fastaFiles/"

## Load function
source(paste0(FUN.DIR,"prepareFastq.R"))

## Run the analysis on single file
prepareFastq(FASTQ.FILE = FASTQ.FILE,
             OUTPUT.DIR = OUTPUT.DIR,
             REMOVE.PCR.DUPLICATES = TRUE,
             REMOVE.UMI.N = TRUE,
             FIVE.PRIME.N.NUMBER = 8,
             THREE.PRIME.N.NUMBER = 2,
             FILTER.BY.SIZE = TRUE,
             SIZE.RANGE = c(18,32))
```

- The prepareFastq() function will generate three folders:
  - **libraryStats**
    - contains statistics with information about processing (*.stats.txt)
  - **totalReads**
    - contains all the piRNA reads including duplicates (*.ALLREADS.fa)
  - **uniqueSequences**
    - contains only unique piRNA sequences only (*.UNIQSEQS.fa)

3. Remove structural contaminants and align to the genome

Next, we proceed with previously generated ****.UNIQSEQS.fa*** and use ***STAR*** aligner to align data:

- First, align sequences to structural genome (tRNA, rRNA, snoRNA, . . .)
  - Structural RNAs can be downloaded from UCSC
  - Follow STAR manual for instruction on how to prepare genome index
- Second, align un-mapped sequences from previous step to species-specific genome
  - There are various sources where species genome files can be downloaded
  - Follow STAR manual for instruction on how to prepare genome index

```
# NOTE: Aligning to structural genome

bash$ # load appropriate version of aligner
bash$ module load STAR/2.5.2b
bash$
bash$ # align to structural genome
bash$ STAR  --runThreadN 6 \
            --runMode alignReads \
            --genomeDir /LocationOfYourStructuralGenome/ \
            --outSAMtype BAM SortedByCoordinate \
            --limitBAMsortRAM 10000000000 \
            --outFilterMultimapNmax 100 \
            --outFilterMismatchNmax 1 \
            --outReadsUnmapped Fastx \
            --outSAMattributes NH HI NM MD AS nM \
            --readFilesIn /InputDirectory/SampleName.UNIQSEQS.fa \
            --outFileNamePrefix /OutputDirectory/mappedToStructural/SampleName.
```

```
# NOTE: Aligning to species-specific genome
# NOTE:   - use *.Unmapped.out.mate1 file generated after
          previous code has been executed

bash$ # align to species genome
bash$ STAR  --runThreadN 6 \
            --runMode alignReads \
            --genomeDir /LocationOfYourSpeciesGenome/ \
            --sjdbGTFfile /LocationOfYourSpeciesFiles/gene.gtf \
            --outSAMtype BAM SortedByCoordinate \
            --limitBAMsortRAM 10000000000 \
            --outFilterMultimapNmax 100 \
            --winAnchorMultimapNmax 100 \
            --sjdbOverhang 100 \
            --outSAMattributes NH HI NM MD AS nM \
            --readFilesIn /InputDirectory/mappedToStructural/SampleName.Unmapped.out.mate1 \
            --outFileNamePrefix /OutputDirectory/mappedToGenome/SampleName.
```

4. Load and pre-process files in the R.

Once the replicate ***bam** files are generated in the previous chunks, we load the data in R environment and prepare it for analysis:

- Load data into R
- Remove potential miRNA contaminants
- Combine replicate data sets
- Calculate sequence overlap between samples
- Combine overlapped data to get Total, Common, and Rare groups
- Full script of all functions can be found at ***piRNA_Diversity github***.

```
# NOTE: LOAD DATA INTO R
```

```r
# load function
source(paste0(FUN.DIR,"filterBam.R"))

## named vector of paths
BAM.PATH.L <- c("rep1"="/Users/genzorp/Documents/GITHUB/piDiversity/data/bamFiles/FHIP/STAR/FH-Piwi-IP_
                "rep2"="/Users/genzorp/Documents/GITHUB/piDiversity/data/bamFiles/FHIP/STAR/FH-Piwi-IP_
                "rep3"="/Users/genzorp/Documents/GITHUB/piDiversity/data/bamFiles/FHIP/STAR/FH-Piwi-IP_

## load multiple files using lapply loop
BAM.L <- lapply(names(BAM.PATH.L), function(s){
  message(paste0("Processing: ",s))
  filterBam(BAMFILE = BAM.PATH.L[[s]],
            BSSPECIES = "Dmelanogaster",
            EXTENTION = ".Aligned.sortedByCoord.out.bam",
            SIMPLECIGAR = TRUE,
            INCLUDE.SECONDARY.ALIGNEMNT=FALSE,
            STANDARD.CONTIGS.ONLY = TRUE,
            GET.ORIGINAL.SEQUENCE = FALSE,
            PERFECT.MATCH.ONLY = TRUE,
            FILTER.BY.FLAG = TRUE,
            SELECTFLAG = c(0,16),
            USE.SIZE.FILTER=TRUE,
            READ.SIZE.RANGE = c(18,32),
            TAGS = c("NH","NM","MD"),
            WHAT = c("flag"),
            SPLIT.NAME.BY = "-") })
names(BAM.L) <- names(BAM.PATH.L)
```

```r
# NOTE: REMOVE miRNA CONTAMINANTS

# libraries
library("GenomicRanges")

# load the function
source(paste0(FUN.DIR,"miRbase2BED.R"))

# path to miRbase annotation
miRBASE.PATH = "/Users/genzorp/Documents/DATA/Annotation/Dmelanogaster/
                dm6_miRBase_dme.gff3"

# make Genomic Range
mi.GR <- makeGRangesFromDataFrame(df = miRbase2BED(miRBASEFILE = miRBASE.PATH),
                                  keep.extra.columns = TRUE)

# Filter data
BAM.L <- lapply(BAM.L, function(s){subsetByOverlaps(x = s, ranges = mi.GR,
                                                    type = "any", invert = TRUE)})

## save ALL image to work with later
#save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/
#three_filtered_bams_all.RData")
```

```
# NOTE: for simplicity only fraction of data was used here
#       - first 500 thousand sequences

SUBSET_500K=500000
BAM.L <- lapply(BAM.L,function(s){s[1:SUBSET_500K]})

## save loaded 500K to object for faster loading later
save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/
           three_filtered_bams_500K.RData")
```

```


# NOTE: COMBINE REPLICATES

# load data
#load(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/three_filtered_bams_all.RD
load(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/three_filtered_bams_500K.RDa

# Load the function
source(paste0(FUN.DIR,"combineThreeGRS.R"))

# combine three technical replicates
BIO.REP <- suppressMessages(
  combineThreeGRS(GRL = BAM.L,
                  REPLICATE.NAMES = c("rep1","rep2","rep3"),
                  MC.CORES = 3))

# show results
BIO.REP
```

```
## GRanges object with 926340 ranges and 3 metadata columns:
##              seqnames            ranges strand |         N        NH      MULT
##                 <Rle>         <IRanges>  <Rle> | <integer> <integer> <integer>
##          [1]    chr2L         4646-4663      - |         3        54         4
##          [2]    chr2L         4714-4739      - |         2        55         3
##          [3]    chr2L         4718-4744      - |         1        55         1
##          [4]    chr2L         4721-4747      - |         2        55         2
##          [5]    chr2L         4722-4742      - |         2        55         2
##          ...      ...               ...    ... .       ...       ...       ...
##     [926336]    chr2L 19536850-19536869      + |         1         1         1
##     [926337]    chr2L 19536850-19536873      + |         1         1         1
##     [926338]    chr2L 19536850-19536875      + |         1         1         7
##     [926339]    chr2L 19536850-19536876      + |         1         1         1
##     [926340]    chr2L 19536851-19536875      + |         1         1         1
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
# all unique sequences
summary(duplicated(BIO.REP))
```

```
##    Mode   FALSE
## logical  926340
```

```
# save ALL
#save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/three_filtered_bams_
```

- While combining replicate data sets we:
  - Counted in how many replicates is each sequence present (N)
  - Recorded the highest sequence multi-mapping score (NH-tag)
  - Calculated the sequence cumulative abundance for identical genome positions (MULT)

```
# NOTE: CALCULATE SEQUENCE OVERLAPS

## Load the function
source(paste0(FUN.DIR,"threeSampleSequenceOverlaps.R"))

## Generate overlaps by sequence
BIOREP.OV.L <- suppressMessages(
  threeSampleSequenceOverlaps(GRL = BAM.L,
                              BSSPECIES = "Dmelanogaster",
                              MC.CORES = 3))
```

```
##
##  FALSE
## 500000
##
##  FALSE
## 500000
##
##  FALSE
## 500000
##    tempName realName Input_N Common_N Exclusive_N Pairwise Pairwise_N
## 1:       FA     rep1  500000   151910      166999       AB     243917
## 2:       FB     rep2  500000   151910      167334       BC     240659
## 3:       FC     rep3  500000   151910      170257       CA     240994
```

```
## show list content
names(BIOREP.OV.L)
```

```
## [1] "Common"     "Exclusive"  "Pairwise"   "SampleInfo"
```

```
## save ALL
#save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/three_filtered_bams_
```

- The objects generated during sequence overlap process are lists:
  - **Common** GR - found in each biological replicate
  - **Rare (Exclusive)** GR - found only in individual biological replicates
  - **Pairwise** GR - shared between pairs of biological replicates
  - **SampleInfo** table summarizing overlaps that can be use to create a Venn diagram

```r
# NOTE: COMBINTE TO GET TOTAL, COMMON, AND RARE
# BEWARE: combining large data sets can be computationally very expnesive

# TOTAL
TOTAL.GR <- suppressMessages(
  combineThreeGRS(GRL = BAM.L,
                  REPLICATE.NAMES = names(BAM.L),
                  MC.CORES = 3))

# ensure there are no duplicates
#summary(duplicated(TOTAL.GR))

## EXCLUSIVE
RARE.GR <- c(BIOREP.OV.L[["Exclusive"]][[1]],
             BIOREP.OV.L[["Exclusive"]][[2]],
             BIOREP.OV.L[["Exclusive"]][[3]])

# ensure there are no duplicates
#summary(duplicated(RARE.GR))

## COMMON
COMMON.GR <- suppressMessages(
  combineThreeGRS(GRL = BIOREP.OV.L[["Common"]],
                  REPLICATE.NAMES = names(BIOREP.OV.L[["Common"]]),
                  MC.CORES = 3))

# ensure each is found in all samples
#unique(mcols(COMMON.GR)[["N"]])

# save ALL
#save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/three_filtered_bams_
```

- After performing all the above operation, it is convenient to identify object that are intended for analysis, and save them into a *RData object for faster loading. Otherwise, depending on the size of the samples, the data preparation will take long time.

FIGURE-RELATED SCRIPTS

- **Figure 1**
- ***"The sequence diversity of piRNAs exceeds the capacity of an individual cell and generates cell-to-cell variability"***

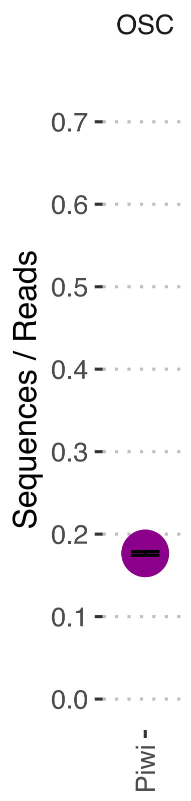
```r
# NOTE: Figure 1A

# load saved data
load("/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/three_filtered_bams_all_brep_ov_co
# load libraries
suppressPackageStartupMessages({library("data.table"); library("dplyr");
  library("ggplot2"); library("ggpubr"); library("GenomicRanges")})

# calculate Sequence versus Read ratio (SoR)
SoR <- lapply(names(BAM.L), function(i){
  GR <- BAM.L[[i]]
  DT <- suppressWarnings(as.data.table(GR, keep.rownames = TRUE))
  DT <- DT[,c("rn","MULT")]
  NDT <- data.table(Sample = i,
                    Sequences = nrow(DT),
                    Reads = sum(DT[["MULT"]]),
                    SeqPerRead = nrow(DT)/sum(DT[["MULT"]]))
  return(NDT) })

# Combine and view results
SoR <- setDT(bind_rows(SoR))
SoR
```

```
##    Sample Sequences    Reads SeqPerRead
## 1:   rep1   4499298 25621559  0.1756059
## 2:   rep2   4478531 25566855  0.1751694
## 3:   rep3   4423222 24681183  0.1792143
```

```r
# add information and summarize
SoR[["Sample"]] <- c(paste0("OSC",1:3))
SoR[["Species"]] <- c(rep("OSC",3))
SoR[["Protein"]] <- c(rep("Piwi",3))
SoR[["Mean"]] <- mean(SoR[["SeqPerRead"]])
SoR[["SD"]] <- sd(SoR[["SeqPerRead"]])
SoR
```

```
##    Sample Sequences    Reads SeqPerRead Species Protein      Mean          SD
## 1:   OSC1   4499298 25621559  0.1756059     OSC    Piwi 0.1766632 0.002220082
## 2:   OSC2   4478531 25566855  0.1751694     OSC    Piwi 0.1766632 0.002220082
## 3:   OSC3   4423222 24681183  0.1792143     OSC    Piwi 0.1766632 0.002220082
```

```r
# plot
ggplot() + theme_pubclean() +
  geom_point(data = SoR, aes(x=factor(Protein), y = Mean, colour = Protein),
             shape = 16, size=8) +
  geom_errorbar(data = SoR, aes(x = Protein, ymin = Mean-SD, ymax = Mean+SD),
                colour = "black", width = 0.4, size = 0.5) +
  scale_colour_manual(values =c("darkmagenta")) +
  facet_wrap(~Species, ncol = 3, scales = "free_x") +
  scale_y_continuous(limits = c(0,0.75), breaks = seq(0,1,0.1)) +
  labs(x="", y="Sequences / Reads", fill="") +
  theme(aspect.ratio = 8, legend.position = "none",
        strip.background = element_blank(),
        axis.text.x = element_text(angle = 90, hjust = 0, vjust = 0.5),
        axis.text = element_text(size = 10))
```



```
# NOTE: Figure 1B

# Accumulation Curve#
# If you want to use the rational function approximation that asymptotically approaches
# a linear curve, you have to download the old version 1.2.1 at
# https://cran.r-project.org/src/contrib/Archive/preseqR/ and use the old function
# preseqR.rfa.species.accum.curve (set the option asym.linear=1 inside the function).
# before proceeding:
# install.packages('preseqR') after locally downloading the above version.
# install.packages("local destination",repos=NULL,type="source")

# libraries
library('stringr');library('gplots');library('ggplot2')
library('data.table');library('plyr');library('dplyr')
library('RColorBrewer');library('dendextend');library('cluster')
library('reshape2');library('rdist');library('rdist')
library('factoextra');library('treeClust');library('clv')
library('preseqR');library('scales')

# ELBOW POINT:
# Function to calculate the elbow point. Point with maximum curvature is defined as
# elbow point of saturation.This function elbow takes input arrays (x and y).
# It will find the point with maximum curvature at index'ind' and returns values x(ind) and y(ind).

elbow <-function(inputx,inputy,scalex,scaley){
  ninputx=inputx/(scalex)
  ninputy=inputy/(scaley)
  ninputx=smooth(ninputx);ninputy=smooth(ninputy)
  curvature=rep(0,length(inputy))
  ypvector=rep(0,length(inputy))
  for (i in 2:(length(inputy)-1)){
    yp=(ninputy[i+1]-ninputy[i-1])/(ninputx[i+1]-ninputx[i-1])
    ypp=(ninputy[i+1]-2*ninputy[i]+ninputy[i-1])/(((ninputx[i+1]-ninputx[i-1])/2)^2)
    curvature[i]= abs(ypp)/((1+(yp^2))^(3/2))
    ypvector[i]=yp }
  curvature[1]=curvature[2]
  ypvector[1]=ypvector[2]
  curvature[i+1]=curvature[i]
  ypvector[i+1]=ypvector[i]
  curvature=smooth(curvature)
  inputx=smooth(inputx);inputy=smooth(inputy)
  dx=(inputx[which.max(curvature)+1]-inputx[which.max(curvature)-1])/2
  dy=(inputy[which.max(curvature)+1]-inputy[which.max(curvature)-1])/2
  ret=c(inputx[which.max(curvature)],inputy[which.max(curvature)],dx,dy)
  return(ret) }

#======= load data ================
Exp1path="data/03162020/MILI_PS_1_Multiplicity.csv"
Exp2path="data/03162020/MILI_PS_2_Multiplicity.csv"
Exp3path="data/03162020/MIWI_PS_1_Multiplicity.csv"
Exp4path="data/03162020/MIWI_PS_2_Multiplicity.csv"
Exp5path="data/03122020/FH-Piwi-IP_1_Multiplicity.csv"
Exp6path="data/03122020/FH-Piwi-IP_2_Multiplicity.csv"
Exp7path="data/03122020/FH-Piwi-IP_3_Multiplicity.csv"

Exp1=read.csv(Exp1path,sep=",")
Exp2=read.csv(Exp2path,sep=",")
Exp3=read.csv(Exp3path,sep=",")
Exp4=read.csv(Exp4path,sep=",")
Exp5=read.csv(Exp5path,sep=",")
Exp6=read.csv(Exp6path,sep=",")
Exp7=read.csv(Exp7path,sep=",")

exp1=data.matrix(Exp1,rownames.force = NA)
exp2=data.matrix(Exp2,rownames.force = NA)
exp3=data.matrix(Exp3,rownames.force = NA)
exp4=data.matrix(Exp4,rownames.force = NA)
exp5=data.matrix(Exp5,rownames.force = NA)
exp6=data.matrix(Exp6,rownames.force = NA)
exp7=data.matrix(Exp7,rownames.force = NA)

#==================================
#Real Data:
total=c(sum(exp1[,1]*exp1[,2]),sum(exp2[,1]*exp2[,2]),
        sum(exp3[,1]*exp3[,2]),sum(exp4[,1]*exp4[,2]),
        sum(exp5[,1]*exp5[,2]),sum(exp6[,1]*exp6[,2]),
        sum(exp7[,1]*exp7[,2]))

unique=c(sum(exp1[,2]),sum(exp2[,2]),
         sum(exp3[,2]),sum(exp4[,2]),
         sum(exp5[,2]),sum(exp6[,2]),
         sum(exp7[,2]))

maxtotal = max(total)

#Estimation:
estimatorExp1 <- preseqR.rSAC.bootstrap(exp1, r=1)
estimatorExp2 <- preseqR.rSAC.bootstrap(exp2, r=1)
estimatorExp3 <- preseqR.rSAC.bootstrap(exp3, r=1)
estimatorExp4 <- preseqR.rSAC.bootstrap(exp4, r=1)
estimatorExp5 <- preseqR.rSAC.bootstrap(exp5, r=1)
estimatorExp6 <- preseqR.rSAC.bootstrap(exp6, r=1)
estimatorExp7 <- preseqR.rSAC.bootstrap(exp7, r=1)

#==================================
predict=20
predict1=predict/(total[1]/maxtotal)
predict2=predict/(total[2]/maxtotal)
predict3=predict/(total[3]/maxtotal)
predict4=predict/(total[4]/maxtotal)
predict5=predict/(total[5]/maxtotal)
predict6=predict/(total[6]/maxtotal)
predict7=predict/(total[7]/maxtotal)
t1=c(seq(0,0.99,by=0.01),seq(1,predict1,by=0.01))
t2=c(seq(0,0.99,by=0.01),seq(1,predict2,by=0.01))
t3=c(seq(0,0.99,by=0.01),seq(1,predict3,by=0.01))
t4=c(seq(0,0.99,by=0.01),seq(1,predict4,by=0.01))
t5=c(seq(0,0.99,by=0.01),seq(1,predict5,by=0.01))
t6=c(seq(0,0.99,by=0.01),seq(1,predict6,by=0.01))
t7=c(seq(0,0.99,by=0.01),seq(1,predict7,by=0.01))
uniqueExp1=estimatorExp1$f(t1)
uniqueExp2=estimatorExp2$f(t2)
uniqueExp3=estimatorExp3$f(t3)
uniqueExp4=estimatorExp4$f(t4)
uniqueExp5=estimatorExp5$f(t5)
uniqueExp6=estimatorExp6$f(t6)
uniqueExp7=estimatorExp7$f(t7)

xExp1=total[1]*t1
xExp2=total[2]*t2
xExp3=total[3]*t3
xExp4=total[4]*t4
xExp5=total[5]*t5
xExp6=total[6]*t6
xExp7=total[7]*t7

setnames <-c(Exp1path,Exp2path,Exp3path,Exp4path,Exp5path,Exp6path,Exp7path)
csvfilenames <- substring(setnames, regexpr("\\/[^\\/]*$", setnames) +1 )
multnames <- as.character(c(str_remove(csvfilenames, ".csv"),"real"))

x.lim = min(max(xExp1),max(xExp2),max(xExp3),max(xExp4),max(xExp5),max(xExp6),max(xExp7))
y.lim = max(uniqueExp1,uniqueExp2,uniqueExp3,uniqueExp4,uniqueExp5,uniqueExp6,uniqueExp7)
xcut=x.lim*1.05
xExp1max=length(xExp1[which(xExp1 <= xcut)])
```

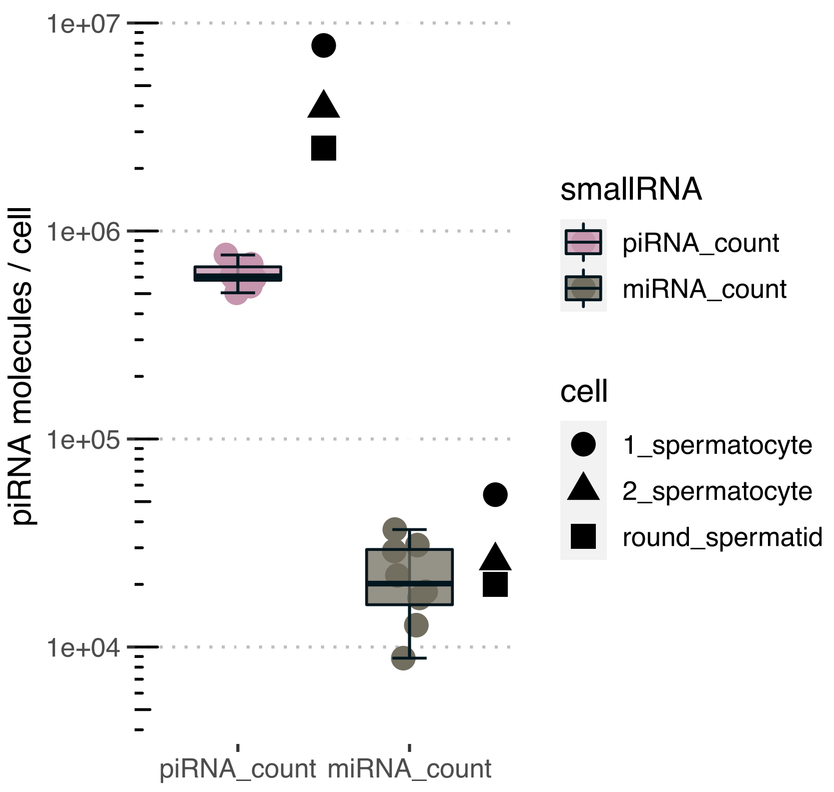
```r
xExp2max=length(xExp2[which(xExp2 <= xcut)])
xExp3max=length(xExp3[which(xExp3 <= xcut)])
xExp4max=length(xExp4[which(xExp4 <= xcut)])
xExp5max=length(xExp5[which(xExp5 <= xcut)])
xExp6max=length(xExp6[which(xExp6 <= xcut)])
xExp7max=length(xExp7[which(xExp7 <= xcut)])

df1 <- data.frame("abundance" = xExp1[1:xExp1max], "unique" = uniqueExp1[1:xExp1max],
                  set = multnames[1], stringsAsFactors = FALSE)
df12 <- data.frame("abundance" = xExp2[1:xExp2max], "unique" = uniqueExp2[1:xExp2max],
                   set = multnames[2], stringsAsFactors = FALSE)
df123 <- data.frame("abundance" = xExp3[1:xExp3max], "unique" = uniqueExp3[1:xExp3max],
                    set = multnames[3], stringsAsFactors = FALSE)
df1234 <- data.frame("abundance" = xExp4[1:xExp4max], "unique" = uniqueExp4[1:xExp4max],
                     set = multnames[4], stringsAsFactors = FALSE)
df12345 <- data.frame("abundance" = xExp5[1:xExp5max], "unique" = uniqueExp5[1:xExp5max],
                      set = multnames[5], stringsAsFactors = FALSE)
df123456 <- data.frame("abundance" = xExp6[1:xExp6max], "unique" = uniqueExp6[1:xExp6max],
                       set = multnames[6], stringsAsFactors = FALSE)
df1234567 <- data.frame("abundance" = xExp7[1:xExp7max], "unique" = uniqueExp7[1:xExp7max],
                        set = multnames[7], stringsAsFactors = FALSE)

# real data
dfreal <- data.frame("abundance" = total, "unique" = unique, set = "real")

# combine tables
df <- setDT(rbind(df1,df12,df123,df1234,df12345,df123456,df1234567,dfreal))
df[["set"]] <- as.character(df[["set"]])
legendTitle = "set"
df[["set"]] <- factor(df[["set"]], levels = multnames)
plot_colors <- c("dark green","light green","dark blue","light blue",
                 "purple","red","pink","black")

# Plot
ggplot(df, aes(x = abundance, y = unique, colour = set)) +
  scale_color_manual(values = plot_colors, name = paste(legendTitle,sep = ""),
                     labels = paste(multnames, sep = "")) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[1]),colour=plot_colors[1],
             linetype="dashed",size = 0.3) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[2]),colour=plot_colors[2],
             linetype="dashed",size = 0.3) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[3]),colour=plot_colors[3],
             linetype="dashed",size = 0.3) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[4]),colour=plot_colors[4],
             linetype="dashed",size = 0.3) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[5]),colour=plot_colors[5],
             linetype="dashed",size = 0.3) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[6]),colour=plot_colors[6],
             linetype="dashed",size = 0.3) +
  geom_vline(aes(xintercept=df[set=="real",]$abundance[7]),colour=plot_colors[7],
             linetype="dashed",size = 0.3) +
  geom_point(data = subset(df, set == "real"), colour = plot_colors[8]) +
  geom_line(data = subset(df, set != "real")) +
  scale_x_continuous(limits = c(0,4e8), breaks = seq(0,4e8,5e7)) +
  scale_y_continuous(limits = c(0,13e6), breaks = seq(0,13e6,1e6)) +
  ggtitle(outputpath) +
  theme_bw() +
  theme(plot.title = element_text(size = 8),
        panel.grid = element_blank(),
        aspect.ratio = 1,
        legend.position = "bottom")
```

```
# NOTE: Figure 1C

# load quantification data
QUANT.CSV="/Users/genzorp/Documents/GITHUB/piDiversity/data/QUANT2/Thenia/piRNA_quantification_data.csv
QUANT.DTM <- melt.data.table(data = fread(QUANT.CSV),
                             id.vars = c("Sample"),
                             measure.vars = c("piRNA_count","miRNA_count"),
                             variable.name = "smallRNA",
                             value.name = "count" )
# take a peak
head(QUANT.DTM, n=3)
```

```
##    Sample    smallRNA    count
## 1:      A piRNA_count 592158.9
## 2:      B piRNA_count 767060.5
## 3:      C piRNA_count 692749.4
```

```
# published data of piRNA and miRNA quantificaiton: PMID:30193099
ZAM.PIMI.DT <- data.table("cell"=c("1_spermatocyte","2_spermatocyte","round_spermatid"),
                          "piRNA_count"=c(7800000,3900000,2500000),
                          "miRNA_count"=c(54000,26000,20000))

ZAM.PIMI.DTM <- melt.data.table(data = ZAM.PIMI.DT, id.vars = "cell",
                                variable.name = "smallRNA",
                                value.name = "count")

# plot settings
two_cols <- c("#C695AE","#726E60")

# plot the piRNA abundances
ggplot() + theme_pubclean() +
  geom_point(data = QUANT.DTM, aes(x = smallRNA, y = count, colour = smallRNA),
             position = position_jitterdodge(),
             size = 4, shape = 16) +
  stat_boxplot(data = QUANT.DTM, aes(x = smallRNA, y = count, fill = smallRNA),
               geom = "errorbar", width = 0.2, colour = "#01161E") +
  geom_boxplot(data = QUANT.DTM, aes(x = smallRNA, y = count, fill = smallRNA),
               alpha = 0.75, width = 0.5, colour = "#01161E") +
  geom_point(data = ZAM.PIMI.DTM, aes(x = smallRNA, y = count, shape = cell),
             size = 4, position = position_nudge(x = 0.5)) +
  scale_fill_manual(values = two_cols) +
  scale_colour_manual(values = two_cols) +
  scale_y_continuous(trans = "log10", limits = c(5e3,1e7)) +
  annotation_logticks(sides = "l") +
  ylab("piRNA molecules / cell") + xlab("") +
  theme(aspect.ratio = 2, legend.position = "right")
```

```
## Warning: Ignoring unknown aesthetics: fill
```

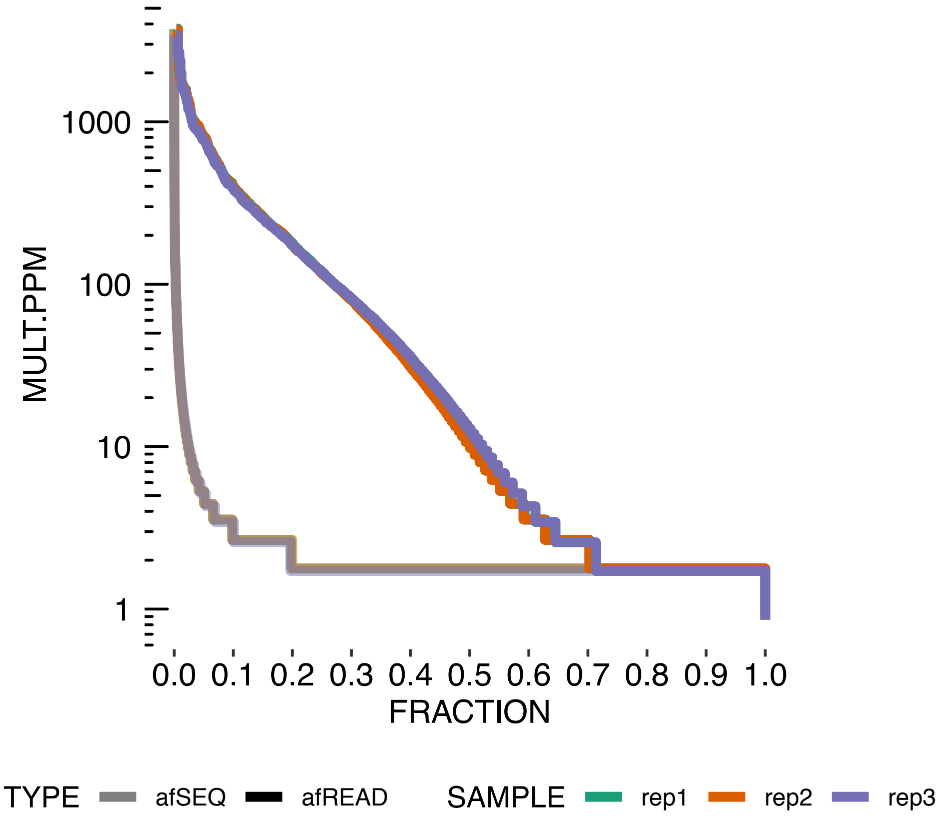


- **Figure 2**
- ***"A skewed distribution of sequence abundance results in a few common and many rare piRNAs."***

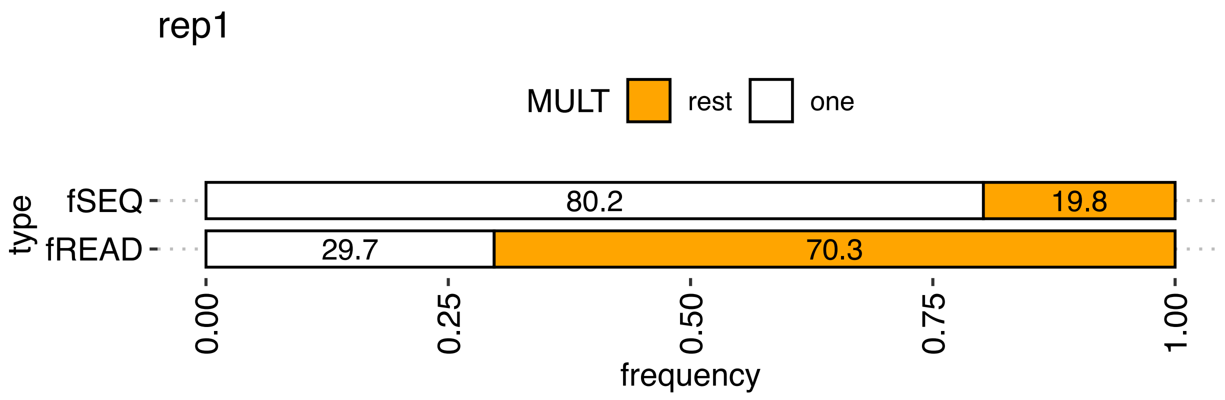
```
# NOTE: Figure 2A

# Load the functions
source(paste0(FUN.DIR,"simpleGRFilter.R"))
source(paste0(FUN.DIR,"simpleStepsPlot.R"))
source(paste0(FUN.DIR,"simpleMultBarPlot.R"))
source(paste0(FUN.DIR,"simpleSRViolin.R"))

# select only the sequences within desired size range
OSC.S249.BIOREP.L <- lapply(BAM.L, function(s){
  suppressMessages(
    simpleGRFilter(GR = s,
                   SIZE.RANGE = c(24,29))) })

## Fig 2A-C
```

```
# Plot Steps
suppressMessages(
  simpleStepsPlot(GRL = OSC.S249.BIOREP.L,
                  Y.PPM = TRUE))
```

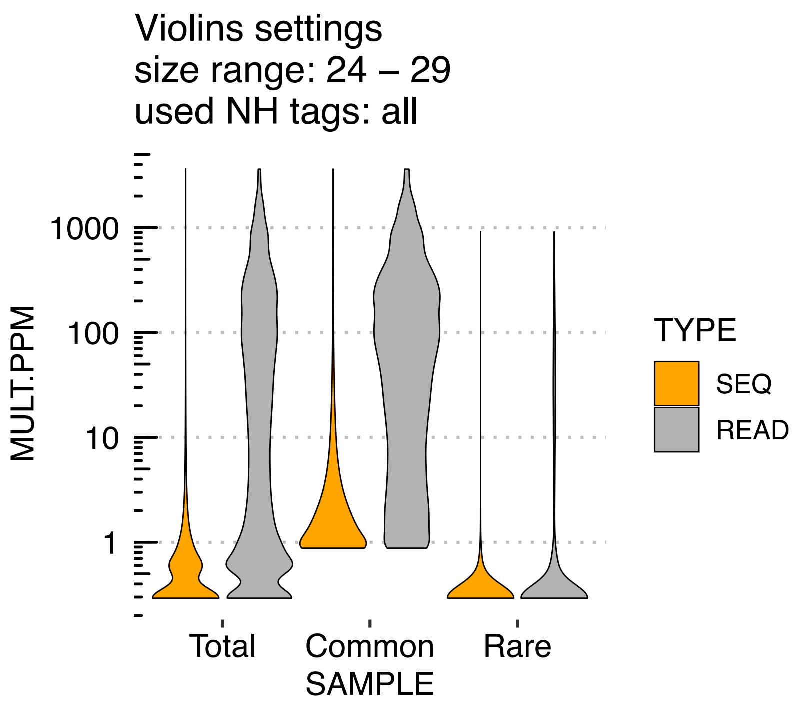
```
## Warning: Using alpha for a discrete variable is not advised.
```



```
# Plot fraction
suppressMessages(
  simpleMultBarPlot(GR = OSC.S249.BIOREP.L[["rep1"]],
                    RANGE.NAME = "rep1"))
```



```r
## Fig 2D-E

# Combine ranges into list
GRL <- list("Common"=COMMON.GR, "Rare"=RARE.GR, "Total"=TOTAL.GR)

# Plot violins
suppressMessages(
  simpleSRViolin(GRL = GRL,
                 SOURCE.DIR = FUN.DIR,
                 NH.TAG = NULL,
                 SIZE.RANGE = c(24,29),
                 SAMPLE.ORDER = c("Total","Common","Rare"),
                 Y.PPM = TRUE))
```

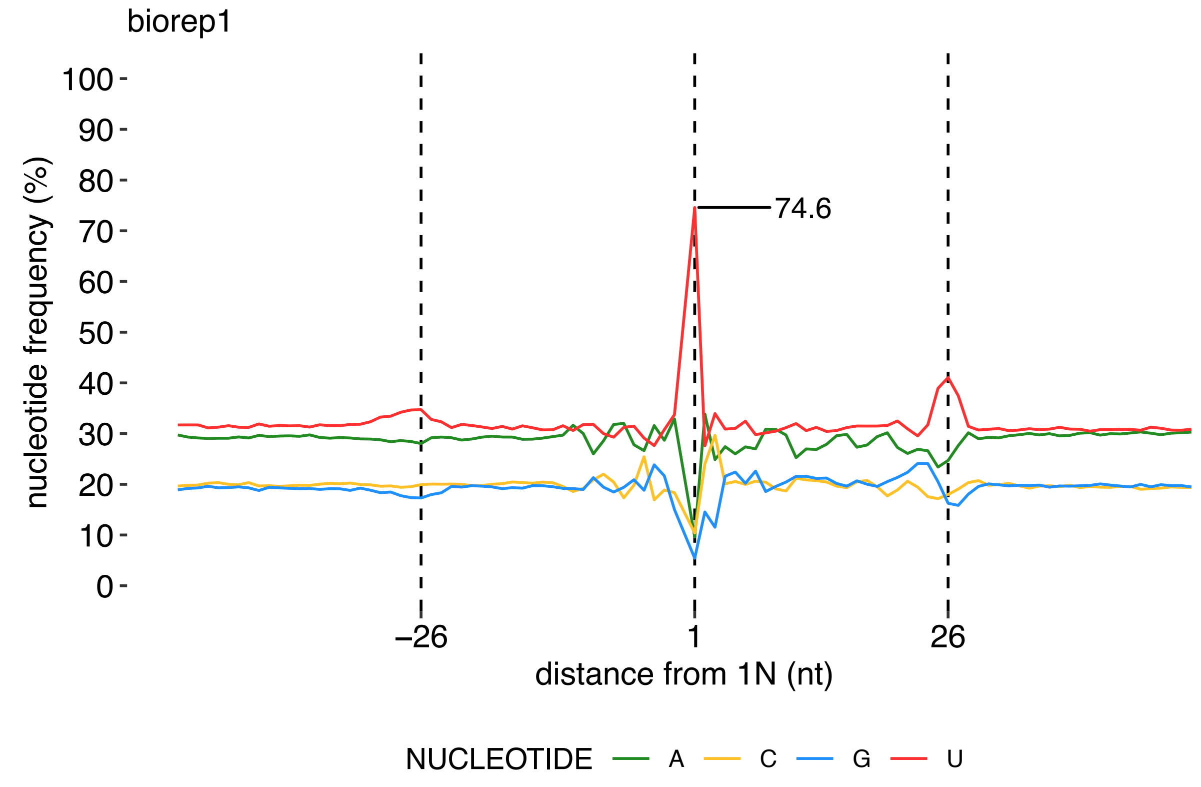


TYPE	Total	Common	Rare
SEQ	1	0.14	0.57
READ	1	0.68	0.16

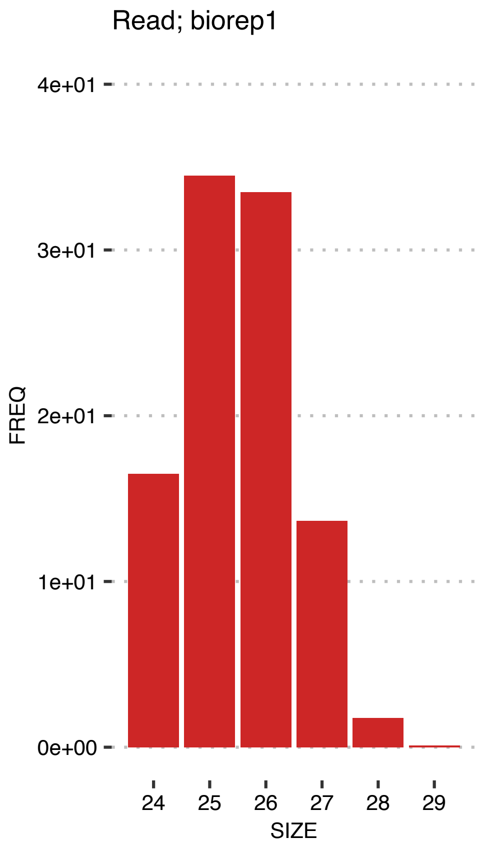
```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name              grob
## 1 1 (1-1,1-1) arrange     gtable[layout]
## 2 2 (2-2,1-1) arrange gtable[colhead-fg]
```

- **Figure S2 and Figure 3B**
- ***Metagene analysis***

```r
# NOTE: Metagene analysis for Figure S2 and Figure 3B

# load functions
source(paste0(FUN.DIR,"simpleGRFilter.R"))
source(paste0(FUN.DIR,"simpleMetageneTab.R"))
source(paste0(FUN.DIR,"simpleMetageneRegularPlotGG.R"))
source(paste0(FUN.DIR,"simpleSDfromGR.R"))

# Filter the data
BIO.REP.U.S249 <- suppressMessages(
  simpleGRFilter(GR = BIO.REP,
                 RANGE.NAME = "biorep1",
                 NH.TAG = 1,
                 SIZE.RANGE = c(24,29)))

# Generate metagene data tables
#   - For 5-prime, for 3-prime change the ALIGN.END
MG.TAB <- suppressMessages(
  simpleMetageneTab(GR = BIO.REP.U.S249,
                    RANGE.NAME = "biorep1",
                    BSSPECIES = "Dmelanogaster",
                    USE.READS = TRUE,
                    ALIGN.END = 5,
                    EXPAND.BY = 50) )

# plot 5-prime metagene
suppressMessages(
  simpleMetageneRegularPlotGG(METAGENE.DT = MG.TAB[["frequency"]],
                              SAMPLE.NAME = "biorep1",
                              Y.LIMITS = c(0,100),
                              PIRNA.SIZE = 26,
                              ASPECT.RATIO = 0.5) )
```



```
# plot size distribution of the sequences or reads
suppressMessages(
  simpleSDfromGR(GR = BIO.REP.U.S249,
                 SAMPLE.NAME = "biorep1",
                 USE.READS = TRUE,
                 PLOT.FREQ = TRUE,
                 YLIMS = c(0,40),
                 BAR.FILL = "firebrick3",
                 BAR.LINE = NA,
                 ASPECT.RATIO = 2) )
```



- **Figure 3 & 4**
- ***"Precursor and processing preferences determine piRNA sequence abundance."***
  - Before we can plot the results we have to re-process the data by:
    * Generate piRNA precursor index for **Rsubread** aligner
    * Export the piRNAs into new fasta file
    * Align the piRNA sequences to piRNA precursors
    * Load the new bam files back into R
    * Filter the data
    * Count and calculate the summaries
    * Select data for plotting
    * Plot the results

```
# NOTE: Generate piRNA precursor index

## load library
library("Rsubread");library("GenomicRanges");
library("data.table"); library("ShortRead")

## Load precursors into GR
OSC.PRECURSORS.GR <- makeGRangesFromDataFrame(df = fread("/Users/genzorp/Documents/GITHUB/piDiversity/da
                             col.names = c("chr","start","end","name","rank","strand")),
                             keep.extra.columns = TRUE)
```

```r
## Extract genome sequence
OSC.PRECURSORS.SEQ <- getSeq(Dmelanogaster, OSC.PRECURSORS.GR)
names(OSC.PRECURSORS.SEQ) <- mcols(OSC.PRECURSORS.GR)[["name"]]

## Export fasta
GENOME_FA_DIR <- "/Users/genzorp/Documents/GITHUB/piDiversity/data/temp/genome_fa/"
writeFasta(object = OSC.PRECURSORS.SEQ, file = paste0(GENOME_FA_DIR,"osc_precursors.fa"))


## INDEX GENOME
GENOME_IDX_DIR <- "/Users/genzorp/Documents/GITHUB/piDiversity/data/temp/genome_rsub/"
OSC.PRECURSORS.IDX <- suppressMessages(
  buildindex(basename = paste0(GENOME_IDX_DIR,"osc_precursors"),
             reference = paste0(GENOME_FA_DIR,"osc_precursors.fa")) )
```

```r
# NOTE: Export the piRNAs into new fasta file

# load function
source(paste0(FUN.DIR,"makeFastaFromGR.R"))

# make fasta from the Total, Common, and Rare Genomic Ranges file
FA_DIR = "/Users/genzorp/Documents/GITHUB/piDiversity/data/temp/granges_fa/"
suppressMessages(
  GR.SEQ.L <- makeFastaFromGR(GRL = GRL,
                  BSSPECIES = "Dmelanogaster",
                  SAVE.TO.FILE = TRUE,
                  FA.DIR = FA_DIR,
                  MC.CORES = 3 ) )
```

```r
# NOTE: Align the piRNA sequences to piRNA precursors

# Provide path to newly made fasta files
GR.FASTA.L <- paste0(FA_DIR,grep("*all.fa$",list.files(FA_DIR), value = TRUE))
names(GR.FASTA.L) <- c(nth(tstrsplit(list.files(FA_DIR),split="_"),1))

# Set oath for alignment output
SUBREAD_OUTPUT <- "/Users/genzorp/Documents/GITHUB/piDiversity/data/temp/output_rsub/"

# Align files to genome with Rsubread
OSC.ALIGN.PRECURSORS <- lapply(names(GR.FASTA.L),function(s){
  suppressMessages(
    aRES <- suppressMessages(
      align(index = paste0(GENOME_IDX_DIR,"osc_precursors"),
            readfile1 = GR.FASTA.L[[s]],
            output_file = paste0(SUBREAD_OUTPUT,s,".bam"),
            sortReadsByCoordinates = TRUE,
            maxMismatches = 1,
            nthreads = 3,
            unique = FALSE,
            nBestLocations = 100) ) )
  return(aRES)})
```

```
# NOTE: Load the new bam files back into R

# load function
source(paste0(FUN.DIR,"filterBam.R"))

# create named vector of paths
MAPPED.GR.PATHS <- paste0(SUBREAD_OUTPUT, grep("bam$",list.files(SUBREAD_OUTPUT), value = TRUE))
names(MAPPED.GR.PATHS) <- nth(tstrsplit(grep("bam$",list.files(SUBREAD_OUTPUT), value = TRUE), split="\

# load bam files into new object
MAPPED.BAM.L <- lapply(names(MAPPED.GR.PATHS), function(s){
  message(paste0("Processing: ",s))
  suppressMessages(
    filterBam(BAMFILE = MAPPED.GR.PATHS[[s]],
              BSSPECIES = "Dmelanogaster",
              EXTENTION = ".bam",
              STANDARD.CONTIGS.ONLY = FALSE,
              TAGS = c("NH","NM"),
              SPLIT.NAME.BY = "_")) })
```

```
## Processing: Common

##    INPUT SIZE_FILTER FLAG_FILTER MISMATCH_FILTER FINAL
## 1: 49472         100         100         95.8522 47420

## Processing: Rare

##     INPUT SIZE_FILTER FLAG_FILTER MISMATCH_FILTER  FINAL
## 1: 157887         100         100        99.48317 157071

## Processing: Total

##     INPUT SIZE_FILTER FLAG_FILTER MISMATCH_FILTER  FINAL
## 1: 296458         100         100        98.79376 292882
```

```
names(MAPPED.BAM.L) <- names(MAPPED.GR.PATHS)
```

```
# NOTE: Filter the data

# load function
source(paste0(FUN.DIR,"simpleGRFilter.R"))

# filter conditions
#  - only single mappers (NH=1)
#  - appropriate size range (24-29-nt long)
#  - originate from piRNA precursor (STRAND=YES)

MAPPED.FILTERED.BAM.L <- lapply(names(MAPPED.BAM.L),function(r){
  filtered.GR <- suppressMessages(
    simpleGRFilter(GR = MAPPED.BAM.L[[r]],
                   RANGE.NAME = r,
```

```r
                   NH.TAG = 1,
                   SIZE.RANGE = c(24,29),
                   STRAND = "YES",
                   SEQNAMES.SPLIT = ":") )
  seqlevels(filtered.GR) <- seqlevelsInUse(filtered.GR)
  return(filtered.GR) })
names(MAPPED.FILTERED.BAM.L) <- names(MAPPED.BAM.L)
```

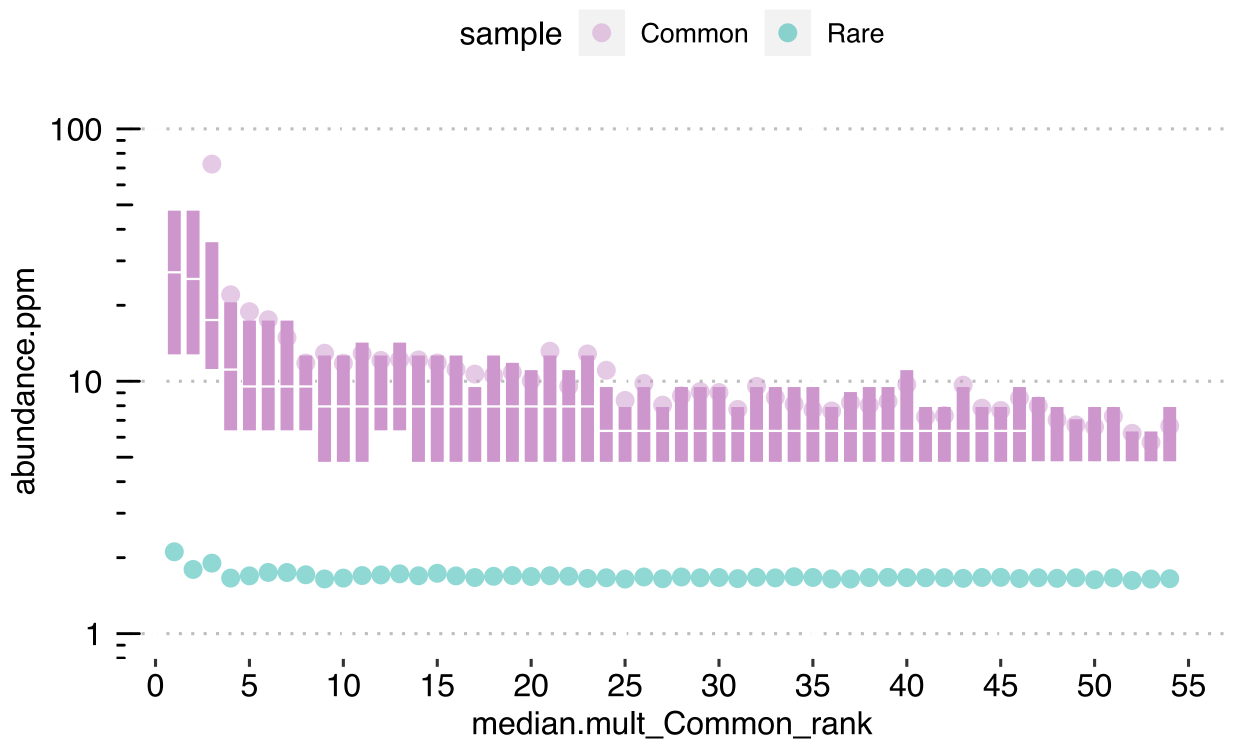
```r
# NOTE: Count and calculate summaries

# load function
source(paste0(FUN.DIR,"simpleClusterSRCountsPerGR.R"))
source(paste0(FUN.DIR,"simpleClusterStatsPerGR.R"))

# Count sequences and reads
SR.COUNTS.DTL <- lapply(names(MAPPED.FILTERED.BAM.L), function(r){
  suppressMessages(
    simpleClusterSRCountsPerGR(GR = MAPPED.FILTERED.BAM.L[[r]],
                               RANGE.NAME = r,
                               CLUSTER.NAME.COLUMN = "seqnames")) })
names(SR.COUNTS.DTL) <- names(MAPPED.FILTERED.BAM.L)

# Calculate mean and median
R.STAT.DTL <- lapply(names(MAPPED.FILTERED.BAM.L), function(r){
  suppressMessages(
    simpleClusterStatsPerGR(GR = MAPPED.FILTERED.BAM.L[[r]],
                            RANGE.NAME = r,
                            CLUSTER.NAME.COLUMN = "seqnames")) })
names(R.STAT.DTL) <- names(MAPPED.FILTERED.BAM.L)

# Create piRNA precursor table
PRECURSOR.DT <- setDT(join_all(dfs = list(join_all(dfs = SR.COUNTS.DTL,
                                                   by = "CLUSTER",
                                                   type = "full"),
                                          join_all(dfs = R.STAT.DTL,
                                                   by = "CLUSTER",
                                                   type = "full")),
                               by = "CLUSTER", type = "full"))

# Filter: only regions with at least 100 seqeunces
SEQ.CUTOFF = 100
SEQ.CUTOFF.COLUMN <- grep("_Common",grep("seq.count",colnames(PRECURSOR.DT),
                                         value = TRUE), value = TRUE)
PRECURSOR.FILTERERD.DT <- PRECURSOR.DT[PRECURSOR.DT[[SEQ.CUTOFF.COLUMN]] >= SEQ.CUTOFF]

# Rank: based on the median common piRNA abundance
RANK.COLUMN <- grep("_Common",grep("median",colnames(PRECURSOR.DT),
                                   value = TRUE),value = TRUE)
setorderv(x = PRECURSOR.FILTERERD.DT, cols = RANK.COLUMN, order = -1)
PRECURSOR.FILTERERD.DT[,paste0(RANK.COLUMN,"_rank") := seq(1,nrow(PRECURSOR.FILTERERD.DT))]
PRECURSOR.FILTERERD.DT[["CLUSTER"]] <- droplevels(PRECURSOR.FILTERERD.DT[["CLUSTER"]])
```

```r
# NOTE: Select data for plotting

# Prepare data table needed for box plot
BOX.TABLE.L <- lapply(names(MAPPED.FILTERED.BAM.L), function(r){
  # a range & table
  a.GR <- MAPPED.FILTERED.BAM.L[[r]]
  a.DT <- as.data.table(a.GR)

  # subset
  a.DT <- a.DT[,c("seqnames", "MULT")]
  colnames(a.DT) <- c("CLUSTER","abundance")
  a.DT[["sample"]] <- r

  # return
  return(a.DT)})

# Calculations
PPM = 1000000
BP.DT <- rbindlist(l = BOX.TABLE.L)
BP.DT[,"total.reads" := sum(mcols(MAPPED.FILTERED.BAM.L[["Total"]])[["MULT"]])]
BP.DT[,"abundance.ppm" := (abundance / total.reads) * PPM]
BP.DT[,"median.abundance.ppm" := lapply(.SD, median),
      by = c("sample","CLUSTER"), .SDcols = c("abundance.ppm")]
BP.DT[,"mean.abundance.ppm" := lapply(.SD, mean),
      by = c("sample","CLUSTER"), .SDcols = c("abundance.ppm")]

# Extract ranked information only for piRNA precursors with sufficient data
BP <- BP.DT[CLUSTER %in% PRECURSOR.FILTERERD.DT[["CLUSTER"]]]
BPM <- setDT(join_all(
  dfs = list(BP,PRECURSOR.FILTERERD.DT[,.SD,
                                      .SDcols= c("CLUSTER",paste0(RANK.COLUMN,"_rank"))]),
                      by = "CLUSTER", type = "full"))
BPM[["CLUSTER"]] <- factor(BPM[["CLUSTER"]], levels = PRECURSOR.FILTERERD.DT[["CLUSTER"]])


# Subset for graphing
BPM.common <- BPM[sample %in% "Common"]
BPM.rare <- BPM[sample %in% "Rare"]
BP.mean <- BPM[, lapply(.SD, mean),
               by = c("CLUSTER",paste0(RANK.COLUMN,"_rank"),"sample"),
               .SDcols = "abundance.ppm"]
BP.mean <- BP.mean[sample %in% c("Common","Rare")]
```

```r
# NOTE: Plot the results - Figure 3A & Figure 4A

# plot settings
FAM = "Helvetica"; XYT = 12; TCOL = "black"

# plot
ggplot() + theme_pubclean() +
  ## common
  geom_boxplot(data = BPM.common, aes_string(x = paste0(RANK.COLUMN,"_rank"),
                                             y = "abundance.ppm",
                                     group = "CLUSTER", colour = "sample",
                                     fill = "sample"),
               outlier.shape = NA, coef = 0, lwd = 0.2,
               colour = "white", fill = "plum3") +
  ## rare
  geom_boxplot(data = BPM.rare, aes_string(x = paste0(RANK.COLUMN,"_rank"),
                                           y = "abundance.ppm",
                                   group = "CLUSTER", colour = "sample",
                                   fill = "sample"),
               outlier.shape = NA, coef = 0, lwd = 0.2,
               colour = "white", fill = "lightseagreen") +

  ## points
  geom_point(data = BP.mean,
             aes_string(x = paste0(RANK.COLUMN,"_rank"), y = "abundance.ppm",
                        colour = "sample"),
             shape = 16, size = 3, alpha = 0.5) +
  ## scales
  scale_y_continuous(trans = "log10", limits = c(1,100)) +
  scale_x_continuous(breaks = seq(0,500,5)) +
  scale_colour_manual(values = c("plum3", "lightseagreen")) +
  scale_fill_manual(values = c("plum3", "lightseagreen")) +

  ## theme
  annotation_logticks(sides = "l") +
  theme(aspect.ratio = 0.5,
        axis.ticks.y = element_blank(),
        axis.text = element_text(family = FAM, size = XYT, color = TCOL),
        axis.title = element_text(family = FAM, size = XYT, colour = TCOL))
```

```
## Warning: Removed 517 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```

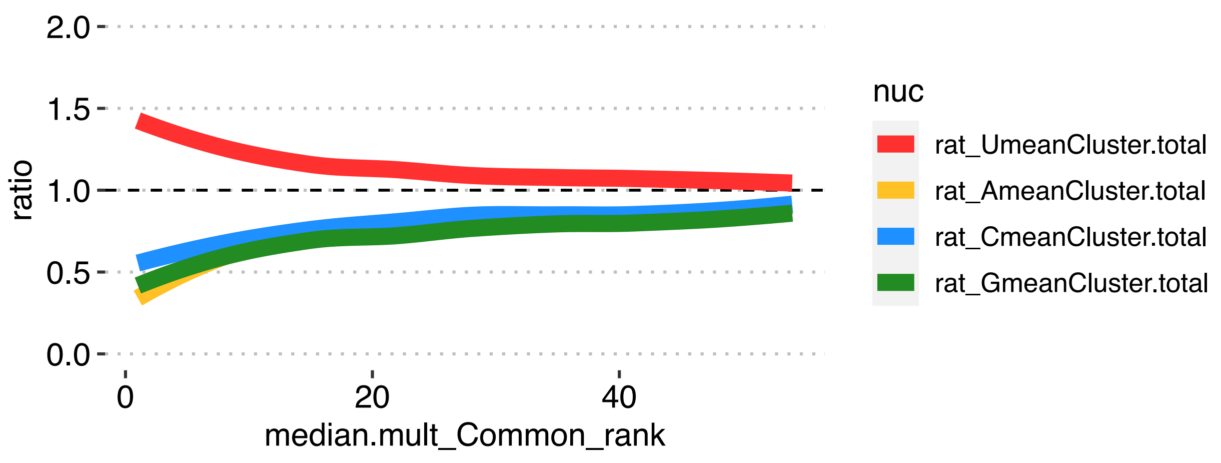


```
# NOTE: Calculate first-nucleotide metrics

## load function
source(paste0(FUN.DIR,"simpleClusterFirstNucStatsPerGR.R"))

## extract first nucleotide count and calculate statistics
N1PC.DTL <- lapply(names(MAPPED.FILTERED.BAM.L),function(s){
  suppressMessages(
    simpleClusterFirstNucStatsPerGR(GR = MAPPED.FILTERED.BAM.L[[s]],
                                    RANGE.NAME = s,
                                    BSSPECIES = "Dmelanogaster",
                                    SOURCE.DIR = FUN.DIR)) })
names(N1PC.DTL) <- names(MAPPED.FILTERED.BAM.L)

## Combine all
N1DT <- setDT(join_all(dfs = N1PC.DTL, by = "CLUSTER", type = "full"))

## Add normalization to parts per million
PPM = 1000000
N1DT[,"mean.T.total.ppm" := (mean.T_Total/total.reads_Total) * PPM]
N1DT[,"mean.A.total.ppm" := (mean.A_Total/total.reads_Total) * PPM]
N1DT[,"mean.G.total.ppm" := (mean.G_Total/total.reads_Total) * PPM]
N1DT[,"mean.C.total.ppm" := (mean.C_Total/total.reads_Total) * PPM]
N1DT[,"mean.AGC.total.ppm" := (mean.AGC_Total/total.reads_Total) * PPM]
N1DT[,"median.T.total.ppm" := (median.T_Total/total.reads_Total) * PPM]
N1DT[,"median.A.total.ppm" := (median.A_Total/total.reads_Total) * PPM]
N1DT[,"median.G.total.ppm" := (median.G_Total/total.reads_Total) * PPM]
N1DT[,"median.C.total.ppm" := (median.C_Total/total.reads_Total) * PPM]
N1DT[,"median.AGC.total.ppm" := (median.AGC_Total/total.reads_Total) * PPM]
```

```r
## calculate ratios
N1DT[,"rat.UnonU.mean.total.raw" := mean.T_Total / mean.AGC_Total]
N1DT[,"rat_UmeanCluster.total" := mean.T_Total / cluster.mean_Total]
N1DT[,"rat_AmeanCluster.total" := mean.A_Total / cluster.mean_Total]
N1DT[,"rat_CmeanCluster.total" := mean.C_Total / cluster.mean_Total]
N1DT[,"rat_GmeanCluster.total" := mean.G_Total / cluster.mean_Total]
N1DT[,"rat.UnonU.median.total.raw" := median.T_Total / median.AGC_Total]
N1DT[,"rat_UmedianCluster.total" := median.T_Total / cluster.median_Total]
N1DT[,"rat_AmedianCluster.total" := median.A_Total / cluster.median_Total]
N1DT[,"rat_CmedianCluster.total" := median.C_Total / cluster.median_Total]
N1DT[,"rat_GmedianCluster.total" := median.G_Total / cluster.median_Total]

# Combine with ranked order from previous figure and setup for plotting
N1.TAB <- setDT(join_all(dfs = list(N1DT,
                                    BP.mean[sample %in% "Common",
                                            .SD,
                                            .SDcols=c("CLUSTER",paste0(RANK.COLUMN,"_rank") )]),
                         by = "CLUSTER", type = "full"))
N1.TAB <- na.omit(N1.TAB)
setorderv(N1.TAB, cols = c(paste0(RANK.COLUMN,"_rank")), order = 1)
DT.F3D <- N1.TAB[,.SD,
                 .SDcols=c("CLUSTER",
                           paste0(RANK.COLUMN,"_rank"),
                           grep("meanCluster",colnames(N1.TAB),value=T))]

DT.F3Dm <- melt.data.table(data = DT.F3D,
                           id.vars = c("CLUSTER",
                                       paste0(RANK.COLUMN,"_rank")),
                           variable.name = "nuc",
                           value.name = "ratio")
```

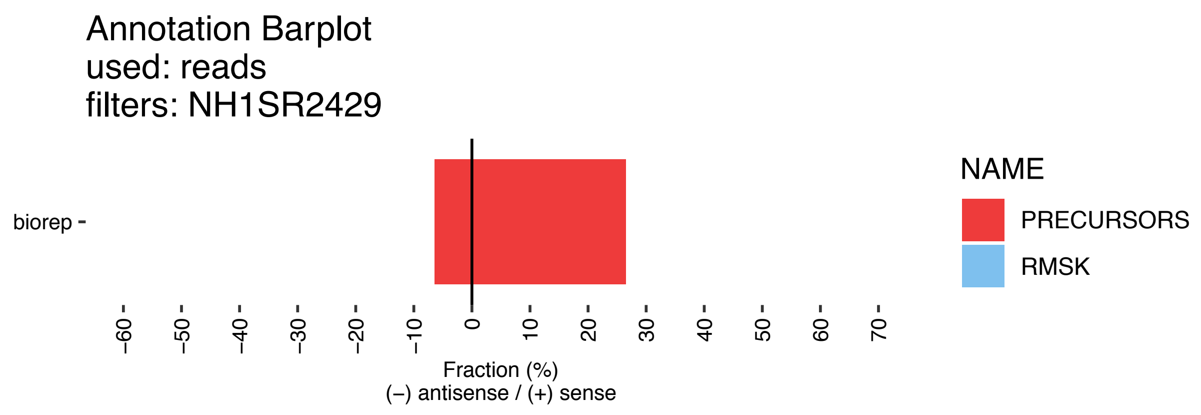
```r
# NOTE: Figure 3D & 4C plots

# Plot
ggplot() + theme_pubclean() +
  geom_hline(yintercept = 1, linetype = "dashed") +

  geom_smooth(data = DT.F3Dm, aes_string(x = paste0(RANK.COLUMN,"_rank"),
                                         y = "ratio",
                                         colour = "nuc"),
              se = FALSE, size = 3, alpha = 0.5) +
  scale_y_continuous(limits = c(0,2)) +
  scale_colour_manual(values = c("firebrick1","goldenrod1","dodgerblue1","forestgreen")) +
  theme(aspect.ratio = 0.5, legend.position = "right",
        axis.text = element_text(family = FAM, size = XYT, color = TCOL),
        axis.title = element_text(family = FAM, size = XYT, colour = TCOL))
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```r
# NOTE: Figure 3Ain & E, Annotating piRNAs by origin and target

# load function
source(paste0(FUN.DIR,"rmskGTF2BED.R"))
source(paste0(FUN.DIR,"annotateGRanked.R"))
source(paste0(FUN.DIR,"annotateRankedBP.R"))

# load rmsk annotations
RMSK.PATH= "/Users/genzorp/Documents/DATA/Annotation/Dmelanogaster/dm6_rmsk_TE.gtf"
RMSK.GR <- suppressMessages(
  rmskGTF2BED(RMSK.GTF = RMSK.PATH, RETURN.GR = TRUE))

# compare RMSK with piRNA precursors
#OSC.PRECURSORS.GR

# annotation in order CAT1 > CAT2 ... > CAT5
ANOT.DT.L <- suppressMessages(
  annotateGRanked(GR = BIO.REP.U.S249,
                  SAMPLE.NAME = "biorep",
                  NH.TAG = 1,
                  SIZE.RANGE = c(24,29),
                  CATEGORY.NAMES = c("PRECURSORS","RMSK"),
                  CATEGORY.1.GR = OSC.PRECURSORS.GR,
                  CATEGORY.2.GR = RMSK.GR,
                  SOURCE.DIR = FUN.DIR) )

# plot the distribution of annotation categories
annotateRankedBP(ANN.TAB.L = ANOT.DT.L,
                 COORD.FLIP = TRUE,
                 ASPECT.RATIO = 0.2,
                 Y.LIMS = c(-60,70))
```

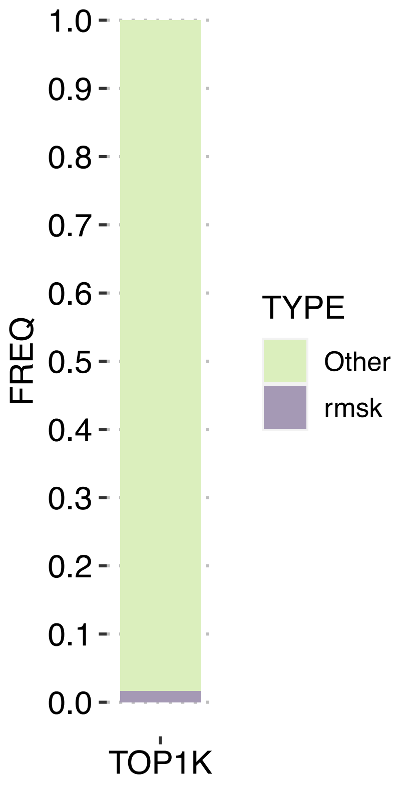
```
## Warning: Removed 1 rows containing missing values (position_stack).
```

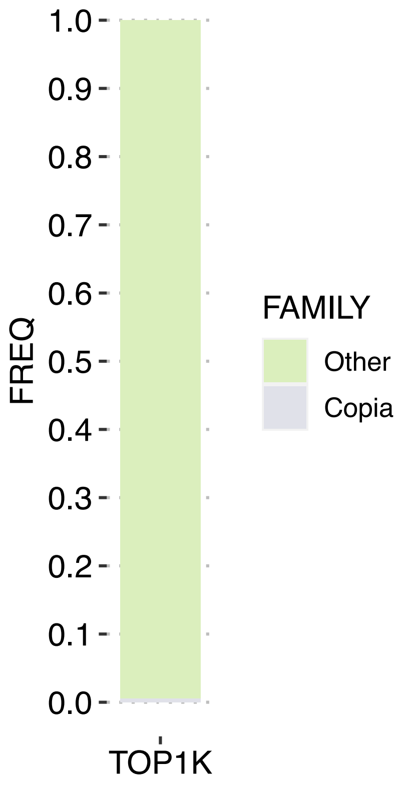


```
# WARNING: this image may not render properly in .pdf
```

```
# NOTE: Figure 3A, Annotating transportable elements

suppressPackageStartupMessages({
  library("ggplot2"); library("ggbio"); library("ggpubr")})

# load function
source(paste0(FUN.DIR,"simpleGRfilter.R"))
source(paste0(FUN.DIR,"simpleAnnotateTE.R"))

# Use previously generated RMSK GR
# RMSK.GR

# Filter the data
BIO.REP.U.S249 <- suppressMessages(
  simpleGRFilter(GR = BIO.REP,
               RANGE.NAME = "biorep1",
               NH.TAG = 1,
               SIZE.RANGE = c(24,29)))

## Order by abundance, calculate ppm and add order
BSSPECIES = "Dmelanogaster"
BIO.REP.U.S249.RANKED <- BIO.REP.U.S249[order(mcols(BIO.REP.U.S249)[["MULT"]],
                                             decreasing = TRUE)]
mcols(BIO.REP.U.S249.RANKED)[["PPM"]] <- (mcols(BIO.REP.U.S249.RANKED)[["MULT"]] /
                                            sum(mcols(BIO.REP.U.S249.RANKED)[["MULT"]]) )*1000000
mcols(BIO.REP.U.S249.RANKED)[["ORDER"]] <- seq(1,length(BIO.REP.U.S249.RANKED))


## TOP 1000 IN RMSK
TOP.IN.RMSK <- suppressMessages(
  simpleAnnotateTE(GR = BIO.REP.U.S249.RANKED[1:1000],
                 RMSK.GR = RMSK.GR,
                 OVERLAP.TYPE = "within") )

TOP.IN.RMSK.FAM <- TOP.IN.RMSK[["FAMILY"]]
TOP.IN.RMSK.FAM[["GROUP"]] = "TOP1K"
TOP.IN.RMSK.FAM[["TYPE"]] = ifelse(TOP.IN.RMSK.FAM[["FAMILY"]] %in% "OTHER","Other","rmsk")
```

```r
# settings
TCOL="black"; FAM="Helvetica"; XYT=12

# plot
plot.TE <- ggplot() + theme_pubclean() +
  geom_bar(data = TOP.IN.RMSK.FAM,
           aes(x = GROUP, y = FREQ, fill = TYPE), stat = "identity")+
  scale_y_continuous(breaks = seq(0,1,0.1)) + xlab("") +
  scale_fill_manual(values = c("#DBEFBD","#A599B5")) +
  theme(aspect.ratio = 7, legend.position = "right",
        axis.text.x = element_text(family = FAM, size = XYT, color = TCOL),
        axis.text.y = element_text(family = FAM, size = XYT, color = TCOL))


## TOP 1000 in TE families
TOP.IN.RMSK.FAM.AS <- TOP.IN.RMSK.FAM[FREQ > 0 & STRAND %in% "AS"]
TOP.Other <- data.table(FAMILY="Other",STRAND="NA", COUNT=1000-sum(TOP.IN.RMSK.FAM.AS[["COUNT"]]),
                        FREQ=1-sum(TOP.IN.RMSK.FAM.AS[["FREQ"]]),GROUP="TOP1K",TYPE="rmsk")
TOP.IN.RMSK.FAM.AS <- rbindlist(l = list(TOP.IN.RMSK.FAM.AS,TOP.Other))
TOP.IN.RMSK.FAM.AS[["FAMILY"]] <- factor(TOP.IN.RMSK.FAM.AS[["FAMILY"]],
                                         levels = rev(TOP.IN.RMSK.FAM.AS[["FAMILY"]]))

## plot
plotTEFAM <- ggplot() + theme_pubclean() +
  geom_bar(data = TOP.IN.RMSK.FAM.AS,
           aes(x = GROUP, y = FREQ, fill = FAMILY), stat = "identity")+
  scale_y_continuous(breaks = seq(0,1,0.1)) +xlab("") +
  scale_fill_manual(values = c("#DBEFBD","#E0E1E9","#E7CEE3","#E4B4C2","#FF99BD")) +
  theme(aspect.ratio = 7, legend.position = "right",
        axis.text.x = element_text(family = FAM, size = XYT, color = TCOL),
        axis.text.y = element_text(family = FAM, size = XYT, color = TCOL))

## ARRANGE AND PRINT
arranged3Fplots <- ggarrange(plotlist = list(plot.TE,plotTEFAM),
                             ncol = 2)
arranged3Fplots
```





```
## NOTE: Figure 3F, flamenco specific mapping

# libraries
library("BSgenome.Dmelanogaster.UCSC.dm6"); library("GenomicAlignments")
library("GenomeInfoDb"); library("parallel"); library("dplyr");
library("BSgenome"); library("data.table")

# load functions
source(paste0(FUN.DIR,"simpleAnnotateTE.R"))
source(paste0(FUN.DIR,"rmskGTF2BED.R"))

# Start with raw bam files
# named vector of paths
BAM.PATH.L <- c("rep1"="/Users/genzorp/Documents/GITHUB/piDiversity/data/bamFiles/FHIP/STAR/FH-Piwi-IP_
                "rep2"="/Users/genzorp/Documents/GITHUB/piDiversity/data/bamFiles/FHIP/STAR/FH-Piwi-IP_
                "rep3"="/Users/genzorp/Documents/GITHUB/piDiversity/data/bamFiles/FHIP/STAR/FH-Piwi-IP_

# quick load settings
TAGS = c("NH","NM","MD"); WHAT = c("flag"); SIMPLE.CIGAR = TRUE; BSSPECIES = "Dmelanogaster"
PARAM = Rsamtools::ScanBamParam(flag = Rsamtools::scanBamFlag(isUnmappedQuery = FALSE),
                                tag = TAGS, simpleCigar = SIMPLE.CIGAR, what = WHAT)
ORIGINAL.FLAM <- GRanges(seqnames = "chrX", ranges=IRanges(start= 21631891, end = 21790731), strand = "

# Load and filter the data
# - only perfectly mapping to flam, 24-29-nt long
OFLAM.ALL.GRL <- mclapply(names(BAM.PATH.L),mc.cores = 3, function(s){
  message("Loading GA for ...")
  message(paste0("\t",s))
  GA <- GenomicAlignments::readGAlignments(file = BAM.PATH.L[[s]],
                                              use.names = TRUE, param = PARAM)
  message("\tfiltering")
  GA.FLAM <- subsetByOverlaps(x = GA, ranges = ORIGINAL.FLAM, type = "any")
  seqlevels(x = GA.FLAM, pruning.mode="coarse") <- c("chrX")
  GA.FLAM <- GA.FLAM[mcols(GA.FLAM)[["NM"]] %in% 0]
  GA.FLAM <- GA.FLAM[width(GA.FLAM) %in% seq(24,29)]

  message("\tformatting")
  FLAM.GA.GR <- GenomicRanges::granges(GA.FLAM, use.names = TRUE, use.mcols = TRUE)
  mcols(FLAM.GA.GR)[["MULT"]] <- dplyr::nth(tstrsplit(
    dplyr::nth(tstrsplit(names(FLAM.GA.GR), split = "-"), -1), split = "M"),-1)
  mcols(FLAM.GA.GR)[["SEQ"]] <- BSgenome::getSeq(eval(parse(text = BSSPECIES)), FLAM.GA.GR)

  message("\tcleaning")
  mcols(FLAM.GA.GR)[["flag"]] <- NULL
  mcols(FLAM.GA.GR)[["MD"]] <- NULL
  mcols(FLAM.GA.GR)[["NM"]] <- NULL
  return(FLAM.GA.GR) })

## combine into a single GR
OFLAM.ALL.GR <- do.call("c",OFLAM.ALL.GRL)
rm(OFLAM.ALL.GRL)

# save image
save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/flamOnly_biorep_filte
```

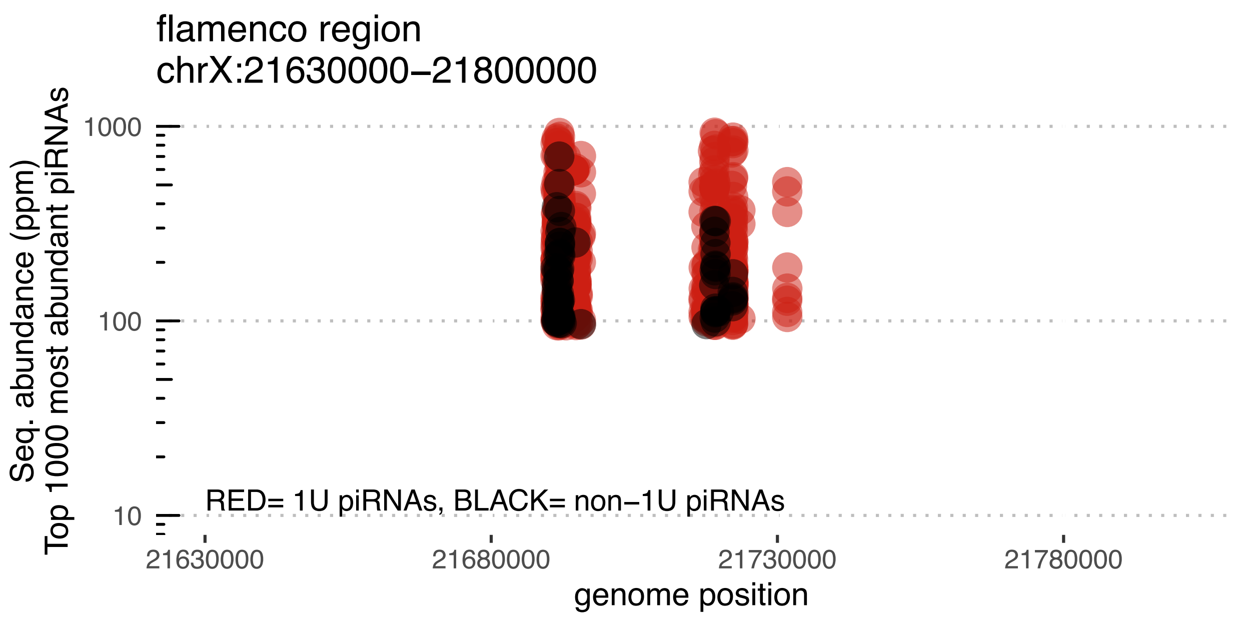
```
## NOTE Figure 3F plot A

# libraries
library("ggplot2"); library("ggbio"); library("ggpubr");
library("BSgenome"); library("BSgenome.Dmelanogaster.UCSC.dm6")
```

```
## Loading required package: rtracklayer
```

```
## load the data
load("/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/flamOnly_biorep_filtered.RData")

# Sort and prepare biological replicate
BSSPECIES="Dmelanogaster"
GR.B <- BIO.REP[order(mcols(BIO.REP)[["MULT"]], decreasing = TRUE)]
mcols(GR.B)[["PPM"]] <- (mcols(GR.B)[["MULT"]] / sum(mcols(GR.B)[["MULT"]]) )*1000000
mcols(GR.B)[["ORDER"]] <- seq(1,length(GR.B))
mcols(GR.B)[["SEQ"]] <- BSgenome::getSeq(eval(parse(text = BSSPECIES)), GR.B)

# Convert GR to table and add info
OFLAM.ALL.DT <- as.data.table(OFLAM.ALL.GR)
mcols(OFLAM.ALL.GR)[["MAPPING"]] <- ifelse(mcols(OFLAM.ALL.GR)[["NH"]] %in% 1, "1","2")
mcols(OFLAM.ALL.GR)[["FIRST"]] <- ifelse(substr(mcols(OFLAM.ALL.GR)[["SEQ"]],
                                                start = 1, stop = 1) %in% "T", "1","2")
OFLAM.ALL.DT[["readNames"]] <- names(OFLAM.ALL.GR)

## add ppm to the sequences
B1000.INFLAM.GR <- OFLAM.ALL.GR[mcols(OFLAM.ALL.GR)[["SEQ"]] %in% mcols(GR.B[1:1000])[["SEQ"]]]
```

```r
B1000.INFLAM.DT <- as.data.table(B1000.INFLAM.GR)
B1000.DT <- as.data.table(GR.B[1:1000])
B1000.INFLAM.DT <-setDT(join_all(dfs = list(B1000.INFLAM.DT, B1000.DT[,c("SEQ","ORDER","PPM")]),
                                 by = "SEQ", type = "left"))
B1000.INFLAM.GR <- makeGRangesFromDataFrame(df = B1000.INFLAM.DT, keep.extra.columns = TRUE)

## extract only unique sequences
B1000.INFLAM.U.GR <- unique(B1000.INFLAM.GR)
B1000.INFLAM.U.DT <- as.data.table(B1000.INFLAM.U.GR)
B1000.INFLAM.U.DT[["FIRST"]] <- factor(B1000.INFLAM.U.DT[["FIRST"]], levels = c(2,1))

# flamenco start and end
FST = 21630000; FEN = 21800000

## plot
ggplot() + theme_pubclean() +
  geom_point(data = B1000.INFLAM.U.DT[FIRST %in% 1], aes(x = start, y = PPM),
             shape = 16, size = 5, alpha = 0.5, color = "#CD1F13") +
  geom_point(data = B1000.INFLAM.U.DT[FIRST %in% 2], aes(x = start, y = PPM),
             shape = 16, size = 5, alpha = 0.5, color = "black") +
  scale_y_continuous(trans = "log10", limits = c(10,1000)) +
  scale_x_continuous(limits = c(FST, FEN), breaks = seq(FST,FEN,50000)) +
  ggtitle(paste0("flamenco region\n","chrX:",FST,"-",FEN)) +
  annotation_logticks(sides = "l") +
  xlab("genome position") +
  ylab("Seq. abundance (ppm)\nTop 1000 most abundant piRNAs") +
  annotate(geom = "text", x = FST, y = 12, hjust = 0,
           label="RED= 1U piRNAs, BLACK= non-1U piRNAs") +
  theme(aspect.ratio = 0.4,
        axis.ticks.y = element_blank())
```

```
## Warning: Removed 13 rows containing missing values (geom_point).
```



```
# NOTE: Mappability

# select dataset
AllMap <- GR.B
UnqMap <- AllMap[AllMap$NH == 1]

# select cluster, (original flam)
TheCluster <- GRanges(seqnames = "chrX", ranges=IRanges(start= 21631891, end = 21790731), strand = "+")
mcols(TheCluster)[["WIDTH"]] <- end(TheCluster)-start(TheCluster)

#make shortcut objects
ReadsInClust <- IRanges::subsetByOverlaps(UnqMap, TheCluster, type = "within")
mcols(TheCluster)[["DENSITY"]] <- sum(as.data.table(ReadsInClust)$MULT)/mcols(TheCluster)[["WIDTH"]]
ReadsInClust <- IRanges::subsetByOverlaps(UnqMap, TheCluster, type = "within")
RangeVect <- c(as.data.table(TheCluster)$start:as.data.table(TheCluster)$end)
TheChromo <- as.vector(as.data.table(TheCluster)$seqnames)
TheStrand <- as.vector(as.data.table(TheCluster)$strand)
RefGen <- BSgenome::getBSgenome("BSgenome.Dmelanogaster.UCSC.dm6")

#size distribution
ReadDistrib <- as.data.table(GR.B)[,sum(MULT), by=width]
ReadDistrib[,V1 := V1/sum(ReadDistrib$V1)]

# Create dm6 genome index using Rsubread
# - refer to manual for instructions

# create all possible piRNAs
NtoT <- lapply(c(18:32), function(a){
  vect <- split(RangeVect, ceiling(seq_along(RangeVect)/a))
  vect <- Filter(function(z){length(z) == a},vect)
  vectDT <- data.table(start = unlist(lapply(vect, min)), end = unlist(lapply(vect, max)))
  manyvect <- lapply(c(0:(a-1)), function(b){
    return(data.table(start = (vectDT$start+b), end = (vectDT$end+b)))
  })
  vectF <- bind_rows(manyvect)
  vectF <- vectF[end <= max(RangeVect)]
  GR1 <- GRanges(seqnames = TheChromo, ranges=IRanges(start=vectF$start, end = vectF$end),
                 strand = TheStrand)
  return(as.data.table(GR1)) })

NtoT <- bind_rows(NtoT)
NtoT <- GRanges(seqnames = NtoT$seqnames, ranges=IRanges(start=NtoT$start, end = NtoT$end),
                strand = NtoT$strand)
names(NtoT) <- paste0("Num_",c(1:length(NtoT)))

# extract sequences for each
NtoT_seq <- BSgenome::getSeq(eval(parse(text = BSSPECIES)), NtoT)

# write to file
# writeFasta(NtoT_seq, file="/YourDirectories/fastaFiles/files/oflam_all.fa")
#

# align to genome using Rsubread
```

```r
# align(index="/YourDirectories/Dm6",
#       readfile1 =  "/YourDirectories/oflam_all.fa",
#       output_file = "/YourDirectories/Rsubread/aligned/oflam_all.bam",
#       maxMismatches = 0, nthreads = 6, sortReadsByCoordinates = TRUE)

# load and identify mult-mapping reads
GA <- readGAlignments(file = "/YourDirectories/Rsubread/aligned/oflam_all.bam",
                      use.names = TRUE, param = ScanBamParam(tag = c("NH"), what = c("flag")))
GA.DT <- as.data.table(GA)
GA.DT$Name <- names(GA)
mcols(NtoT)[["MULTI"]] <- 1
mcols(NtoT[names(NtoT) %in% GA.DT[flag == 0][seqnames == "chrX"]
           [start %in% RangeVect][end %in% RangeVect][,Name]])[["MULTI"]] <- 0

# scale 'un-mappability' per length
Hmmm <- lapply(c(18:32),function(a){

  if(a %in% width(NtoT[NtoT$MULTI == 1])){
    GR <- NtoT[width(NtoT) == a & NtoT$MULTI == 1]
    GR.DT <- as.data.table(GR)
    GR.DT$Names <- names(GR)
    min(GR.DT$start)
    max(GR.DT$end)
    covDT <- as.data.table(coverage(GR))[!1:(min(RangeVect)-1),3]
    covDT[,value := (value/a)*ReadDistrib[width == a][,V1]]
    covDT <- rbind(covDT, data.table(
      value = rep(0, times=((max(RangeVect)-min(RangeVect)+1)-nrow(covDT)))))
    return(covDT)
  }
  else{return(data.table(value = rep(0,(max(RangeVect)-min(RangeVect)+1))))} })

# sum up scaled un-mappabilities by length
Hmmm <- bind_cols(Hmmm)
colnames(Hmmm) <- paste0(c(18:32),"nt")
Hmmm[,Coverage := matrixStats::rowSums2(as.matrix(Hmmm))]
Hmmm[,Position := c(1:nrow(Hmmm))]

# get coverage and make final table
RIC.DT <- as.data.table(ReadsInClust)
RIC.DT <- RIC.DT[rep(seq.int(1,nrow(RIC.DT)), RIC.DT$MULT)]
RIC <- GRanges(seqnames = RIC.DT$seqnames,
               ranges = IRanges(start=RIC.DT$start, end = RIC.DT$end),
               strand = RIC.DT$strand)

as.data.table(coverage(RIC)[[]])  [!1:(min(RangeVect)-1)]

FinalCov <- as.data.table(coverage(RIC)[[6]])[!1:(min(RangeVect)-1)]
FinalCov$Position <- c(start(TheCluster):max(end(RIC)) )-(start(TheCluster)-1)
FinalCov <- merge(FinalCov,Hmmm[,c("Coverage","Position")],  all.y = T, by="Position")
FinalCov$Position <- c(start(TheCluster):end(TheCluster))
colnames(FinalCov) <- c("POSITION", "COVERAGE", "MAP")

# take moving average (we used 2001)
```

```r
library(accelerometry)
FinalCov$MAP.TR <- movingaves(c(rep(0,1000), FinalCov$MAP, rep(0,1000)), 2001)
FinalCov$Cov.TR <- movingaves(c(rep(0,1000), FinalCov$COVERAGE, rep(0,1000)), 2001)
FinalCov$MultiCovTR <- movingaves(c(rep(0,1000), FinalCov$MultiCov, rep(0,1000)), 2001)

TRANS = "log10"
YLIMS.1 = c(10,1000)

ggplot() +
  theme_bw() +
  geom_segment(data = FinalCov, aes(x=POSITION, xend=POSITION,
                                    y= YLIMS.1[1] ,yend = YLIMS.1[2],
                                    color = MAP.TR)) +
  ## SCALES
  scale_color_gradient(low = "white",high = "grey80") +
  scale_y_continuous(trans = TRANS, limits= YLIMS.1) +
  scale_x_continuous(breaks = seq(21631000,21800000,10000)) +
  annotation_logticks(sides = "l") +

  ggtitle(paste0("ORIGINAL FLAM  ", "chrX:",
                 min(FinalCov[["POSITION"]]), " - ",
                 max(FinalCov[["POSITION"]]))) +
  xlab("Genomic position") + ylab("Seq. abundance (ppm)") +

  ## THEME
  theme(aspect.ratio = 0.3,
        legend.position = "bottom",
        panel.border = element_blank(),
        panel.grid = element_blank())
```

Additional Figures * ***Analysis of the piRNA sensors***

```r
# NOTE: piRNA Sensors - prepare data

# Sensor information
# Flam_1st: chrX:21,635,292-21,635,521:+ (230bp)
# Flam_2nd: chrX:21,635,062-21,635,291:+ (230bp)
# Flam_100bp: chrX:21,635,292-21,635,391:+ (100bp)
# Flam_460bp: chrX:21,635,062-21,635,521:+ (460bp)
# CG17514: chr3L:27,970,220-27,970,449:- (230bp)
# Tj: chr2L:19,466,747-19,466,976:+ (230bp)
# Path:  chr3L:9,495,281-9,495,510:- (230bp)
# CycB3:  chr3R:24,868,353-24,868,582:-  (230bp)
# CycB3_5:  chr3R:24,870,599-24,870,828:- (230bp)

# Genomic Ranges components
CHR = c("chrX","chrX","chrX","chrX","chr3L","chr2L","chr3L","chr3R","chr3R")
START = c(21635292,21635062,21635292,21635062,27970220,19466747,9495281,24868353,24870599)
END = c(21635521,21635291,21635391,21635521,27970449,19466976,9495510,24868582,24870828)
STRAND = c("+","+","+","+","-","+","-","-","-")
NAME = c("Flam230a","Flam230b","Flam100","Flam460","CG17514","Tj","Path","CycB3","CycB35")
SI <- keepStandardChromosomes(seqinfo(eval(parse(text = "Dmelanogaster"))))
```

```r
# Genomic ranges
SEN.GR <- GRanges(seqnames = CHR,
                  ranges = IRanges(start = START, end = END),
                  strand = STRAND,
                  seqinfo = SI)
mcols(SEN.GR)[["name"]] <- paste0(CHR,":",START,"-",END,":",STRAND,":",NAME)
SEN.GR

# Get sequence
SEN.SEQ <- getSeq(eval(parse(text = "Dmelanogaster")), SEN.GR)
names(SEN.SEQ) <- mcols(SEN.GR)[["name"]]

# Write individual fasta files
lapply(names(SEN.SEQ), function(n){
  A.SEQ <- SEN.SEQ[n]
  N <- gsub(":","_",n)
  writeFasta(object = A.SEQ, file = paste0(FA.DIR,N,".fa")) })

## In the bash on the server

##
##  1. Make genomes from each fasta for your aligner
##  2. Align piRNAs to each sensor and
##  3. Proced with bam files
##

# find the new bam files
BAM.PATH = "/Users/genzorp/Documents/GITHUB/piDiversity/Sensors/stranded_STAR/"
EXTENTION = ".Aligned.sortedByCoord.out.bam$"
FILE.NAMES = grep(EXTENTION, list.files(BAM.PATH), value = TRUE)
SAMPLE.NAMES = gsub(EXTENTION, "", FILE.NAMES)
SENSOR.NAMES <- nth(tstrsplit(SAMPLE.NAMES, split = "_"),4)
BAM.FILE.PATHS <- paste0(BAM.PATH,FILE.NAMES)
names(BAM.FILE.PATHS) <- SENSOR.NAMES
BAM.FILE.PATHS

# Load the bam files
source(paste0(FUN.DIR,"filterBam.R"))
SENSOR.BAM.L <- lapply(names(BAM.FILE.PATHS), function(s){

  message(paste0("Loading: ",s))
  SGR <- suppressMessages(filterBam(BAMFILE = BAM.FILE.PATHS[[s]],
                  BSSPECIES = "Dmelanogaster",
                  PERFECT.MATCH.ONLY = TRUE,
                  READ.SIZE.RANGE = c(24,29),
                  STANDARD.CONTIGS.ONLY = FALSE,
                  GET.ORIGINAL.SEQUENCE = TRUE,
                  SPLIT.NAME.BY = "_"))

  ## Filter by actual strand of piRNA
  mcols(SGR)[["rs"]] <- nth(tstrsplit(names(SGR),split = "_"),4)
  mcols(SGR)[["gs"]] <- nth(tstrsplit(seqnames(SGR),split = ":"),3)
  message(length(SGR))
  SGR <- SGR[mcols(SGR)[["rs"]] == mcols(SGR)[["gs"]],]
  message(length(SGR))

  ## Return
  return(SGR)})
names(SENSOR.BAM.L) <- names(BAM.FILE.PATHS)

# save image
save.image(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/sensor_mapped.RData")
```

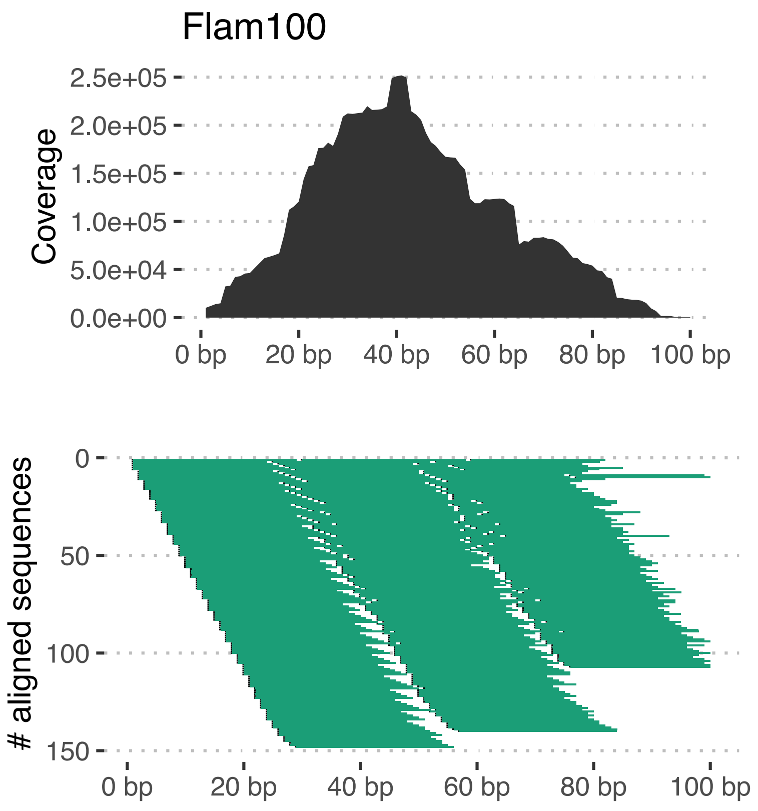
```
# NOTE: piRNA SENSORS - analyze data

# libraries
suppressPackageStartupMessages({
  library(ggfortify); library(scales);
  library(ggbio)})

# load prepared data
load(file = "/Users/genzorp/Documents/GITHUB/piDiversity/sessions/revisions/sensor_mapped.RData")

# Expand mapped data into reads
READS.BAM.L <- suppressWarnings(suppressMessages(
  lapply(names(SENSOR.BAM.L), function(s){
    message(paste0("Expanding to reads: ",s))
    A.GR <- SENSOR.BAM.L[[s]]
    DT.GR <- as.data.table(A.GR, keep.rownames = "aseq")
    setkey(DT.GR, cols = "aseq")
    DT.LONG <- DT.GR[rep(aseq, MULT)]
    R.GR <- makeGRangesFromDataFrame(DT.LONG, keep.extra.columns = TRUE)
    return(R.GR)}) ))
names(READS.BAM.L) <- names(SENSOR.BAM.L)

# plot piRNA read coverage
Flam100.COV <- ggbio::autoplot(READS.BAM.L[["Flam100"]], stat = "coverage", main="Flam100")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```

```
Flam100.COV <- Flam100.COV + theme_pubclean() +
  scale_y_continuous(labels = scientific_format()) +
  theme(aspect.ratio = 0.5)

# plot mapped piRNA sequences
Flam100.SEQ <-  ggbio::autoplot(object = SENSOR.BAM.L[["Flam100"]], aes(fill = strand))
Flam100.SEQ <- Flam100.SEQ + scale_y_reverse() + theme_pubclean() +
  ylab("# aligned sequences") +
  theme(aspect.ratio = 0.5, legend.position = "none")
```

```
## Scale for 'y' is already present. Adding another scale for 'y', which will
## replace the existing scale.
```

```
# Arrange
SENSOR.MAPPING.PLOT <- ggarrange(plotlist = list(Flam100.COV@ggplot,
                                                 Flam100.SEQ@ggplot), nrow = 2)

SENSOR.MAPPING.PLOT
```



bash Scripts

These scripts were used to count the total number of calibrators in publicly available small RNA libraries in order to estimate correction factor (kcorr) **PMID:30193099**, and for quantification of miRNA and piRNA populations in data generated in this paper.

```
# NOTE: Bash script used to count calibrator sequences in public small RNA libraries
#       for calculation of correction factor (kcorr)
#       - mouse male germ cell data from PMID: 30193099

#!/bin/bash

# ABOUT
# Find the number of calibrator sequences by looking for their full sequence
# without mismatches

# inputs
input=$1
```

```
name=$(basename ${input%.fastq.gz})

echo "Counting the Calibrators"
printf "Calibrator\nA_CalibratorSequence" > "${name}"_Calibrator_seq.txt

echo "Counts" > "${name}"_Calibrator_Count.txt
zgrep 'CalibratorSequence' "${name}".fastq.gz | wc -l >> "${name}"_Calibrator_Count.txt

paste "${name}"_Calibrator_seq.txt \
      "${name}"_Calibrator_Count.txt > "${name}"_Calibrator_Counts.txt

# remove intermediate files
rm "${name}"_Calibrator_seq.txt
rm "${name}"_Calibrator_Count.txt

echo "Done"
```

```
# NOTE: Bash script used to count calibrator sequences in small RNA libraries
#       generated for this paper in order to quantify piRNA and miRNA populations.

#!/bin/bash
module load cutadapt
module load samtools
module load bowtie/1.2.3
module load fastqc

# Provide fastq file as first argument to this bash script
input=$1
name=$(basename ${input%.fastq})
mkdir "${name}"_Analysis

# Remove 3-prime adapter sequence, keep > 29-nt
echo "removing adaptor"
cutadapt  -a TGACTGTGGAATTCTCGGGTGCCAAGG \
          --overlap 10 \
          -m 29 \
          --untrimmed-output "${name}"_Untrimmed.fastq \
          -o "${name}"_trimmed.fastq "${name}".fastq \
          > "${name}"_Remove3Adaptor_Report.txt

mv "${name}"_Untrimmed.fastq "${name}"_Analysis
mv "${name}"_trimmed.fastq "${name}"_Analysis
mv "${name}"_Remove3Adaptor_Report.txt "${name}"_Analysis
cd "${name}"_Analysis

# Collapse by sequence / remove PCR duplicates
echo "removing PCR duplicates"
cat "${name}"_trimmed.fastq | awk 'NR%4==2' | sort | uniq -c | \
    awk '{OFS= "\n"; print ">"NR"-"$1,$2}' > "${name}"_trimmed_collapsed.fasta

# Remove UMIs (8N at 5'end & 2N at 3'end)
echo "removing UMIs"
cutadapt  -u 8 -u -2 \
          -o "${name}"_trimmed_collapsed_UMIremoved.fasta \
          "${name}"_trimmed_collapsed.fasta > \
          "${name}"_RemoveUMI_Report.txt

# Remove structural RNAs
echo "Align reads to structural RNAs"
bowtie  /YourStructuralIndexFolder/dm6_strRNAs_Bowtie1index \
        -v 1 -m 100 -p 12 --best --strata \
        --un NonStructural_"${name}"_trimmed_collapsed_UMIremoved.fasta \
        -f "${name}"_trimmed_collapsed_UMIremoved.fasta \
        -S 2> "${name}"_AlignToStractural_Report.txt | \
        samtools view -Sh > Structural_"${name}"_trimmed_collapsed_UMIremoved.sam

# Find the numbers of calibrators looking for their 25 first nt
echo "Counting the Calibrators"
printf  "Calibrator\n1_CALIBRATORSEQUENCEONE\n2_CALIBRATORSEQUENCETWO" > \
        "${name}"_Calibrator_seq.txt
echo Counts > "${name}"_Calibrator_Count.txt

grep \
  'CALIBRATORSEQUENCEONE' NonStructural_"${name}"_trimmed_collapsed_UMIremoved.fasta | \
  wc >> "${name}"_Calibrator_Count.txt
grep \
  'CALIBRATORSEQUENCETWO' NonStructural_"${name}"_trimmed_collapsed_UMIremoved.fasta | \
  wc >> "${name}"_Calibrator_Count.txt

paste "${name}"_Calibrator_seq.txt "${name}"_Calibrator_Count.txt > \
      "${name}"_Calibrator_Counts.txt

rm "${name}"_Calibrator_seq.txt
rm "${name}"_Calibrator_Count.txt

# Remove calibrators with cutadapt - use a first 25-nt of each calibrator
echo "Removing the Calibrators"
cutadapt  \
  -g file:/FolderWithCalibratorFasta/Calibrators_first25nt.fasta \
  --overlap 25 -m 20 -e 0 \
  -o NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators.fasta \
  NonStructural_"${name}"_trimmed_collapsed_UMIremoved.fasta > \
  "${name}"_RemoveCalibrators_Report.txt

# Identify miRNAs
echo "Mapping against miRNAs"
bowtie \
  /YourMiRNAIndexFolder/dm_mature_miRNAs_Bowtie1index \
  -v 1 -m 100 -p 12 --best --strata \
  --al NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAs.fasta \
  --un NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout.fasta \
  -f NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators.fasta \
  -S 2> "${name}"_AlignTomiRNA_Report.txt | \
  samtools view -Sh > \
  Mapped_NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miR_Aligned.sam
```

```
# Filter to keep 24-29-nt long reads only
echo "Selecting for 24-29nt long reads"
cutadapt  \
  -m 24 -M 29 \
  -o NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout_24to29.fasta \
  NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout.fasta > \
  "${name}"_keep24to29nt_Report.txt

# Map to Dm6 genome
echo "Aligning to Dm6 genome"
bowtie  \
  /YourGenomeIndexFolder/dm6_Bowtie1index \
  -v 1 -m 100 -p 12 --best --strata \
  --al Mapped_NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout_24to29.fasta \
  --un Unmapped_NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout_24to29.fasta \
  -f NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout_24to29.fasta \
  -S 2> "${name}"_AlignToDm6_Report.txt | \
  samtools view -Sh > \
  NonStructural_"${name}"_trimmed_collapsed_UMI_noCalibrators_miRNAout_24to29_Dm6Aligned.sam
```

Functions

These custom functions are used throughout the manuscript to prepare data, generate count tables or various objects, and create plots that are found throughout the manuscrits. If you interested in using these functions, or have any issues please go to **HaaseLab/piRNA__Diversity github repository** for up-to-date versions or to contact us.

Function: prepareFastq()

```
prepareFastq <- function(
                        ## INPUT/OUTPUT
                        FASTQ.FILE=NULL,
                        OUTPUT.DIR=NULL,

                        ## OPTION: PCR DUPLICATE REMOVAL
                        REMOVE.PCR.DUPLICATES=FALSE,

                        ## OPTION: UMI REMOVAL
                        REMOVE.UMI.N=FALSE,
                        FIVE.PRIME.N.NUMBER=8,
                        THREE.PRIME.N.NUMBER=2,

                        ## OPTION: SIZE FILTERING
                        FILTER.BY.SIZE=FALSE,
                        SIZE.RANGE=NULL,

                        ## OUTPUT
                        RETURN.TABLE=FALSE){

  ## AUTHOR: Pavol Genzor
  ## Use: Pre-process .fastq file with or without UMIs
```

```r
## PG 06.24.2021; Version3; Refined

## load libraries
suppressPackageStartupMessages({library(data.table);library(plyr);
  library(dbplyr);library(ShortRead)})

## check inputs
if(is.null(FASTQ.FILE)) stop("Please provide fastq file !")
if(isTRUE(FILTER.BY.SIZE)){if(is.null(SIZE.RANGE))
  stop("Please provide read size range !")}

message("Processing ...")

## extract file names and set output.dir
FILE.NAME <- dplyr::nth(data.table::tstrsplit(basename(FASTQ.FILE),split = "\\."),1)
if(!is.null(OUTPUT.DIR)){OUT.DIR <- OUTPUT.DIR} else { OUT.DIR <- dirname(FASTQ.FILE) }

message("\treading in the .fastq file")
message(paste0("  ",FILE.NAME))
FASTQ.F <- ShortRead::readFastq(dirPath = FASTQ.FILE)

message("\textracting sequence information")
FQSEQ <- ShortRead::sread(FASTQ.F)
LENGTH.FQSEQ <- length(FQSEQ)

if(isTRUE(REMOVE.PCR.DUPLICATES)){
  message("\tremoving PCR duplicates")
  NOPCRDUPS <- BiocGenerics::unique(FQSEQ)

  # for stat report
  LENGTH.NOPCRDUPS = length(NOPCRDUPS)
  PCNTDUPS = 100 - (length(NOPCRDUPS)/length(FQSEQ))*100
  message(paste0("\t\tduplication percentage: ",round(PCNTDUPS, digits = 3)))
} else {
  message("\tNOT removing PCR duplicates")
  NOPCRDUPS <- FQSEQ; LENGTH.NOPCRDUPS = NA; PCNTDUPS = NA}

if(isTRUE(REMOVE.UMI.N)){
  message("\tremowing UMI sequences")
  message(paste0("  5‘N: ",FIVE.PRIME.N.NUMBER," 3‘N: ",THREE.PRIME.N.NUMBER))
  NOUMI <- XVector::subseq(x = NOPCRDUPS,
                           start = FIVE.PRIME.N.NUMBER+1,
                           end = -(THREE.PRIME.N.NUMBER+1))

  # for stat report
  LENGTH.NOUMI = length(NOUMI)
} else {
  message("\tNOT removing UMI sequences")
  NOUMI <- NOPCRDUP; LENGTH.NOUMI = NA}

if(isTRUE(FILTER.BY.SIZE)){
  message("\tremoving reads outside of SIZE.RANGE")
  message(paste0(" ",paste(seq(SIZE.RANGE[1],SIZE.RANGE[2]),collapse = ", ")))
```

```r
    NOUMI.ALL <- NOUMI[width(NOUMI) %in% seq(SIZE.RANGE[1],SIZE.RANGE[2])]
    NOUMI.ALL <- ShortRead::srsort(NOUMI.ALL)

    # for stat report
    LENGTH.NOUMI.ALL <- length(NOUMI.ALL)
  } else {
    message("\tNOT removing reads outside of SIZE.RANGE")
    NOUMI.ALL <- NOUMI; NOUMI.ALL <- ShortRead::srsort(NOUMI.ALL)
    LENGTH.NOUMI.ALL = NA}

  message("\tcreating a data.table of all reads")
  DTT <- as.data.table(NOUMI.ALL)
  DTT[["rname"]] <- paste0(FILE.NAME,"-","R",seq(1,nrow(DTT)))
  colnames(DTT) <- gsub("x","seq",colnames(DTT))

  # for stat report
  LENGTH.DTT <- nrow(DTT)

  message("\tcollapsing into unique sequences")
  DT <- as.data.table(NOUMI.ALL)[,.N, by=c("x")]
  DT[["sname"]] <- paste0("S",seq(1,nrow(DT)),"M",DT[["N"]])
  DT[["lsname"]] <- paste0(FILE.NAME,"-","S",seq(1,nrow(DT)),"M",DT[["N"]])
  colnames(DT) <- gsub("x","seq",colnames(DT))

  # for stat report
  LENGTH.DT <- nrow(DT)

  message("\tjoining ALL and UNIQUE")
  LDT <- setDT(plyr::join_all(dfs = list(DTT,DT), by = "seq", type = "full"))
  LDT[["rsname"]] <- paste0(LDT[["rname"]],LDT[["sname"]])

  message("\tcreating new DNAStringSet objects")
  UNIQ.SEQ <- DNAStringSet(DT[["seq"]])
  names(UNIQ.SEQ) <- DT[["lsname"]]
  ALL.SEQ <- DNAStringSet(LDT[["seq"]])
  names(ALL.SEQ) <- LDT[["rsname"]]

  # the stat report
  message("\tcreating stat report")
  LIBSTAT <- data.table(metric = c("initialFastqReads","withoutPCRDuplicates",
                                   "withoutUMI","selectedSizeRange", "allReads",
                                   "uniqueSequences"),
                        values = c(LENGTH.FQSEQ, LENGTH.NOPCRDUPS, LENGTH.NOUMI,
                                   LENGTH.NOUMI.ALL, LENGTH.DTT, LENGTH.DT) )

  message("\twritting .fasta files and saving reports")
  OUT.DIR.NAMES <- c("totalReads/","uniqueSequences/","libraryStats/")
  OUT.DIR.PATHS <- paste(OUT.DIR,OUT.DIR.NAMES, sep = "/")
  for(i in OUT.DIR.PATHS){ifelse(dir.exists(path = i),TRUE,dir.create(path = i))}

  message(paste0(" output directory: ",OUT.DIR))
  writeFasta(object = ALL.SEQ,
             file = paste0(OUT.DIR,"/totalReads/",FILE.NAME,".ALLREADS.fa"))
  writeFasta(object = UNIQ.SEQ,
             file = paste0(OUT.DIR,"/uniqueSequences/",FILE.NAME,".UNIQSEQS.fa"))
  write.csv(x = LIBSTAT,
            file = paste0(OUT.DIR,"/libraryStats/",FILE.NAME,".stats.txt"))

  if(isTRUE(RETURN.TABLE)){
    message(" returning table");message("Done."); message("")
    return(LDT)
  } else {message("Done.");message("") }

}
```

Function: filterBam()

```r


filterBam <- function(
                      ## INPUTS
                      BAMFILE=NULL,
                      BSSPECIES=NULL,
                      EXTENTION=".Aligned.sortedByCoord.out.bam",

                      ## OPTIONS
                      SIMPLECIGAR=TRUE,
                      INCLUDE.SECONDARY.ALIGNEMNT=FALSE,

                      GET.ORIGINAL.SEQUENCE=FALSE,
                      STANDARD.CONTIGS.ONLY=TRUE,
                      PERFECT.MATCH.ONLY=TRUE,

                      FILTER.BY.FLAG=TRUE,
                      SELECTFLAG=c(0,16),

                      USE.SIZE.FILTER=TRUE,
                      READ.SIZE.RANGE=c(18,50),

                      TAGS=c("NH","NM","MD"),
                      WHAT=c("flag"),

                      ## name specific
                      SPLIT.NAME.BY="-"
                      ){

  ## AUTHOR: Pavol Genzor
  ## Use: Load filtered bam into R
  ## 06.25.21; Version 7; refined

  ## NOTES
  ## Flag: 256 = not primary alignment; 272 = reverse strand not primary alignment;
  ## Flag: 0 = forward unpaired unique alignment 16 = reverse unpaired unique alignment
  ## Tags: NH:i:1 = unique alignment
  ## Tags: NM = edit distance to the reference
  ##
```

```r
## NOTE: This program requires prepareFastq.R to be able to extract multiplicity information

## libraries
suppressPackageStartupMessages({library("data.table");library("dplyr");library("Rsamtools");
  library("GenomicAlignments");library("BSgenome.Hsapiens.UCSC.hg38");
  library("BSgenome.Dmelanogaster.UCSC.dm6"); library("BSgenome.Mmusculus.UCSC.mm10")})

## check input
if(is.null(BAMFILE)) stop("Please provide full path to a .bam file !!!")
if(is.null(BSSPECIES)) stop("Please provide BSSPECIES name !!!")
if(isTRUE(GET.ORIGINAL.SEQUENCE)){WHAT=c("flag","seq")}

## for report
PROGRESS.L <- list()

FILE.NAME <- gsub(EXTENTION,"",basename(BAMFILE))
message("Processing ...")

## PARAMETERS FOR LOADING BAM FILE
PARAM = Rsamtools::ScanBamParam(flag = Rsamtools::scanBamFlag(isUnmappedQuery = FALSE,
                                                              isSecondaryAlignment = INCLUDE.SECONDAR
                                tag = TAGS, simpleCigar = SIMPLECIGAR, what = WHAT)
message(" prepared loading parameters")
message(paste0("\tTAGS:\t",paste(TAGS, collapse = ", ")))
message(paste0("\tCIGAR:\t",ifelse(isTRUE(SIMPLECIGAR),"simple cigar","all cigar")))
message(paste0("\tWHAT:\t",paste0(WHAT,collapse = ", ")))
message(" loading .bam file into GAlignemnts")
GA <- GenomicAlignments::readGAlignments(file = BAMFILE, use.names = TRUE, param = PARAM)

## ***
GA.IN <- length(GA)
PROGRESS.L[["INPUT"]] <- GA.IN
message(paste0("\tIMPORTED: ", GA.IN))

if(isTRUE(USE.SIZE.FILTER)){
  message(" filtering by read size")
  message(paste0("\tRANGE:\t",paste0(READ.SIZE.RANGE, collapse = "-")))
  GA <- GA[width(GA) %in% seq(READ.SIZE.RANGE[1],READ.SIZE.RANGE[2],by = 1)]

  ## ***
  REMAINDER = (length(GA)/GA.IN)*100
  PROGRESS.L[["SIZE_FILTER"]] <- REMAINDER
  message(paste0("\tREMAINDER: ", round(REMAINDER, digits = 2) )) }

if(isTRUE(STANDARD.CONTIGS.ONLY)){
  message(" removing non-standard contigs")
  REG.CHR <- standardChromosomes(eval(parse(text = BSSPECIES)))
  GAR <- GA[seqnames(GA) %in% REG.CHR]
  GenomeInfoDb::seqlevels(GAR) <- REG.CHR

  ## ***
  REMAINDER = (length(GAR)/GA.IN)*100
  PROGRESS.L[["CONTIG_FILTER"]] <- REMAINDER
```

```
    message(paste0("\tREMAINDER: ", round(REMAINDER, digits = 2) )) }
  else{ GAR <- GA }

  if(isTRUE(FILTER.BY.FLAG)){
    message(" selecting ONLY primary alignemnts") ## Adjust to use different flags
    GARP <- GAR[mcols(GAR)[["flag"]] %in% SELECTFLAG]
    message(" removing flag column")
    mcols(GARP)[["flag"]] <- NULL

    ## ***
    REMAINDER = (length(GARP)/GA.IN)*100
    PROGRESS.L[["FLAG_FILTER"]] <- REMAINDER
    message(paste0("\tREMAINDER: ", round(REMAINDER, digits = 2) )) }
  else { GARP <- GAR }

  if(isTRUE(PERFECT.MATCH.ONLY)){
    message(" removing reads with mismatches")
    GARP <- GARP[mcols(GARP)[["NM"]] %in% c(0)]
    mcols(GARP)[["NM"]] <- NULL
    mcols(GARP)[["MD"]] <- NULL

    ## ***
    REMAINDER = (length(GARP)/GA.IN)*100
    PROGRESS.L[["MISMATCH_FILTER"]] <- REMAINDER
    message(paste0("\tREMAINDER: ", round(REMAINDER, digits = 2) )) }

  if(isTRUE(GET.ORIGINAL.SEQUENCE)){
    message(" retrieving original read sequences")
    BAMSEQ <- mcols(GARP)[["seq"]]
    ISONMINUS <- as.logical(GenomicAlignments::strand(GARP) == "-")
    BAMSEQ[ISONMINUS] <- Biostrings::reverseComplement(BAMSEQ[ISONMINUS])
    mcols(GARP)[["seq"]] <- BAMSEQ }

  message(" converting to GRanges")
  GARP.GR <- GenomicRanges::granges(GARP, use.names = TRUE, use.mcols = TRUE)
  PROGRESS.L[["FINAL"]] <- length(GARP.GR)

  message(" adding multiplicity column [MULT]")
  mcols(GARP.GR)[["MULT"]] <- as.integer(dplyr::nth(
    data.table::tstrsplit(
      dplyr::nth(
        data.table::tstrsplit(names(GARP.GR),split = SPLIT.NAME.BY),-1), split = "M"),-1))

  ## RETURN
  message("Done!")
  message("")
  print(as.data.table(PROGRESS.L))
  return(GARP.GR)
}
```

Function: miRbase2BED()

```r
miRbase2BED <- function(
                        ## INPUT
                        miRBASEFILE = NULL,

                        ## OPTIONS
                        miRNAOnly = TRUE){

  ## AUTHOR: Pavol Genzor
  ## Use: Load miRbase files into R
  ## 06.28.21; Version 3; refined, removed saving option

  ## LIBRARIES
  suppressPackageStartupMessages({library("data.table")})

  ## check for input
  if(is.null(miRBASEFILE)){stop("Please provide .gtf2 or .gff3 file from miRbase")}

  ## which file type
  FILE.TYPE <- tstrsplit(tail(unlist(tstrsplit(miRBASEFILE, split = "/")), n=1), split = "\\.")[[2]]

  ## GFF3

  if(FILE.TYPE %in% "gff3"){
    message("Processing .gff3 file")
    GFF <- fread(miRBASEFILE, header = FALSE)
    GFF.COLNAMES <- c("chr","score0","feature","start","end","score1","strand","score2","description")
    colnames(GFF) <- GFF.COLNAMES

    ## split descriptions column
    ID = tstrsplit(tstrsplit(GFF$description, split = ";")[[1]], split = "=")[[2]]
    ALIAS = tstrsplit(tstrsplit(GFF$description, split = ";")[[2]], split = "=")[[2]]
    NAME = tstrsplit(tstrsplit(GFF$description, split = ";")[[3]], split = "=")[[2]]
    ORIGIN = tstrsplit(tstrsplit(GFF$description, split = ";")[[4]], split = "=")[[2]]

    ## make bed
    miRBED <- data.table(chr = GFF$chr, start = GFF$start, end = GFF$end,
                         name = NAME, feature = GFF$feature, strand = GFF$strand,
                         gene_id = ID, alias = ALIAS, origin =  ORIGIN)

    ## only mature miRNAs
    if(isTRUE(miRNAOnly)){miRBED <- miRBED[miRBED$feature %in% "miRNA",]} }

  ## GTF2

  if(FILE.TYPE %in% "gtf2"){
    message("Processing .gtf2 file")
    GTF2 <- fread(miRBASEFILE, header = FALSE)
    GTF2.COLNAMES <- c("chr","database","feature","start","end","score1","strand","score2","description
    colnames(GTF2) <- GTF2.COLNAMES

    ## split descriptions column
    GID <- tstrsplit(tstrsplit(tstrsplit(GTF2$description, split = ";")[[1]], split =" ")[[2]], split =
    TID <- tstrsplit(tstrsplit(tstrsplit(GTF2$description, split = ";")[[2]], split =" ")[[3]], split =

    ## MAKE BED TABLE
    miRBED <- data.table(chr = GTF2$chr, start = GTF2$start,end = GTF2$end,
                         gene_id = GID, transript_id = TID, strand = GTF2$strand,
                         database = GTF2$database)

    ## only mature miRNAs
    if(isTRUE(miRNAOnly)){miRBED <- miRBED[miRBED$feature %in% "miRNA",]}}

  message("Done.")
  return(miRBED)
}
```

Function: combineThreeGRS()

```
combineThreeGRS <- function(
                            ## INPUT
                            GRL=NULL,
                            REPLICATE.NAMES=NULL,

                            ## SETTINGS
                            MC.CORES=3){

  ## AUTHOR: Pavol Genzor
  ## Use: Combine three genomic ranges by sequence
  ## 06.25.21; Version 2

  ## NOTE: This function takes time and resources

  ## LIBRARIES
  suppressPackageStartupMessages({library("data.table");library("plyr");
    library("dplyr");library("GenomicRanges");library("stringi");library("parallel")})

  ## INPUT CHECKING
  if(is.null(GRL)) stop("Provide named list of Genomic Ranges")
  if(isFALSE(class(GRL) %in% c("list"))) stop("Input must be a list of GRanges")
  if(isTRUE(is.null(names(GRL)))) stop("This list needs to be named")
  if(is.null(REPLICATE.NAMES)) stop("Provide names for three replicates")
  if(!length(REPLICATE.NAMES)==3) stop("Provide exactly THREE replicates")

  message("Proceeding ...")

  message(" combining replicates")
  T.REPS.GR <- c(GRL[[REPLICATE.NAMES[1]]],
                 GRL[[REPLICATE.NAMES[2]]],
                 GRL[[REPLICATE.NAMES[3]]])

  message(" making a data.table")
  T.REP.DT <- as.data.table(T.REPS.GR)
  message(" removing width")
  T.REP.DT[[c("width")]] <- NULL

  message(" creating a unique ID")
  T.REP.DT[["id"]] <- stringi::stri_c(T.REP.DT[["seqnames"]],
                                      T.REP.DT[["start"]],
                                      T.REP.DT[["end"]],
                                      T.REP.DT[["strand"]], sep = "__")

  message(" summarizing mcols")
  USQ.COUNT <- T.REP.DT[,.N, by = "id"]
  USQ.NH <- T.REP.DT[,lapply(.SD, max), by = "id", .SDcols = c("NH")]
  USQ.MULT <- T.REP.DT[,lapply(.SD, sum), by = "id", .SDcols = c("MULT")]

  message(" combining summary results") ## SLOWER
  NF <- setDT(plyr::join_all(dfs = list(USQ.COUNT,USQ.NH,USQ.MULT),
                             by = "id", type = "full"))

  message(" expanding the ID")
  EXPANDED.ID.L <- mclapply(seq(1,4), mc.cores = MC.CORES, function(i){
    dplyr::nth(data.table::tstrsplit(NF[["id"]],split="__",fixed=TRUE),i) })
  names(EXPANDED.ID.L) <- c("chr","start","end","strand")

  message(" adding expanded ID items to the table")
  NF[["chr"]] <- EXPANDED.ID.L[["chr"]]
  NF[["start"]] <- EXPANDED.ID.L[["start"]]
  NF[["end"]] <- EXPANDED.ID.L[["end"]]
  NF[["strand"]] <- EXPANDED.ID.L[["strand"]]

  message(" converting to GRanges")
  NF.GR <- GenomicRanges::makeGRangesFromDataFrame(df = NF, keep.extra.columns = TRUE)

  message(" cleaning up")
  mcols(NF.GR)[["id"]] <- NULL

  message("done!")
  return(NF.GR)
}
```

Function: combineTwoGRS()

```
combineTwoGRS <- function(
                            ## INPUT
                            GRL=NULL,
                            DUPLICATE.NAMES=NULL,

                            ## SETTINGS
                            MC.CORES=3
                            ){

  ## AUTHOR: Pavol Genzor
```

```r
## Use: Combine duplicatre genomic ranges by sequence
## 06.28.21; Version 3; refined

## NOTE: This function takes time

## LIBRARIES
suppressPackageStartupMessages({library("data.table");library("plyr");
  library("dplyr");library("GenomicRanges");library("stringi");library("parallel")})

## INPUT CHECKING
if(is.null(GRL)) stop("Provide named list of Genomic Ranges")
if(isFALSE(class(GRL) %in% c("list"))) stop("Input must be a list of GRanges")
if(isTRUE(is.null(names(GRL)))) stop("This list needs to be named")
if(is.null(DUPLICATE.NAMES)) stop("Provide names for two samples")
if(!length(DUPLICATE.NAMES)==2) stop("Provide exactly TWO ranges")

message("Proceeding ...")

message(" combining duplicates")
D.REPS.GR <- c(GRL[[DUPLICATE.NAMES[1]]],
               GRL[[DUPLICATE.NAMES[2]]])

message(" making a data.table")
D.REP.DT <- as.data.table(D.REPS.GR)

message(" removing width")
D.REP.DT[[c("width")]] <- NULL

message(" creating a unique ID")
D.REP.DT[["id"]] <- stringi::stri_c(D.REP.DT[["seqnames"]],
                                    D.REP.DT[["start"]],
                                    D.REP.DT[["end"]],
                                    D.REP.DT[["strand"]], sep = "__")

message(" summarizing mcols")
## Sumarizing happens over UNIQUE ID here
USQ.COUNT <- D.REP.DT[,.N, by = "id"]
USQ.NH <- D.REP.DT[,lapply(.SD, max), by = "id", .SDcols = c("NH")]
USQ.MULT <- D.REP.DT[,lapply(.SD, sum), by = "id", .SDcols = c("MULT")]

message(" combining summary results") ## SLOW
NF <- setDT(plyr::join_all(dfs = list(USQ.COUNT,USQ.NH,USQ.MULT),
                           by = "id", type = "full"))

message(" expanding the ID")
EXPANDED.ID.L <- mclapply(seq(1,4), mc.cores = MC.CORES, function(i){
  dplyr::nth(data.table::tstrsplit(NF[["id"]],split="__",fixed=TRUE),i) })
names(EXPANDED.ID.L) <- c("chr","start","end","strand")

message(" adding expanded ID items to the table")
NF[["chr"]] <- EXPANDED.ID.L[["chr"]]
NF[["start"]] <- EXPANDED.ID.L[["start"]]
NF[["end"]] <- EXPANDED.ID.L[["end"]]
```

```
  NF[["strand"]] <- EXPANDED.ID.L[["strand"]]

  message(" converting to GRanges")
  NF.GR <- GenomicRanges::makeGRangesFromDataFrame(df = NF, keep.extra.columns = TRUE)

  message(" cleaning up")
  mcols(NF.GR)[["id"]] <- NULL

  message("done!")
  return(NF.GR)
}
```

Function: threeSampleSequenceOverlaps()

```
threeSampleSequenceOverlaps <- function(
                                        ## INPUT
                                        GRL=NULL,
                                        BSSPECIES=NULL,

                                        ## SETTINGS
                                        MC.CORES=4,

                                        ## OUTPUT
                                        REMOVE.SEQ=TRUE){

  ## AUTHOR: Pavol Genzor
  ## Use: Calculate three sample overlaps
  ## 06.25.21; Version 3

  ## load libraries
  suppressPackageStartupMessages({library("data.table");library("parallel");
    library("GenomicRanges");library("GenomicAlignments");library("BSgenome.Hsapiens.UCSC.hg38");
    library("BSgenome.Dmelanogaster.UCSC.dm6");library("BSgenome.Mmusculus.UCSC.mm10")})

  #source("/Users/genzorp/Documents/GITHUB/piDiversity/r/extractFastqMultiplicity.R")

  ## INPUT CHECKING
  if(is.null(GRL)) stop("Please provide named GRanges list !")
  if(is.null(BSSPECIES)) stop("Please provide BSSPECIES name !")
  if(!isTRUE(class(GRL) %in% "list")) stop("GRL must be a named LIST !")
  if(is.null(names(GRL))) stop("GRP must be a NAMED list !!!")
  if(!isTRUE(unique(unlist(lapply(GRL,class)) %in% "GRanges"))) stop("All the components must be GRange

  message("Processing ...")

  ## get sequences
  message("\textracting sequences")
  GRLS <- mclapply(names(GRL), mc.cores = MC.CORES, function(i){
    BSgenome::getSeq(eval(parse(text = BSSPECIES)),GRL[[i]]) })
  names(GRLS) <- names(GRL)
```

```
message("\tchecking sequence duplication")
for(i in names(GRLS)) {print(table(duplicated(GRLS[[i]]))  )}

message("\n\tadding sequence to GRanges")
for(i in names(GRLS)) {mcols(GRL[[i]])[["seq"]] <- GRLS[[i]]}

message("\tmaking name map and assigning\n")
TEMP.NAMES <- c("FA","FB","FC")
names(GRLS) <- TEMP.NAMES
NAME.MAP <- data.table(tempName = TEMP.NAMES,
                       realName = names(GRL),
                       Input_N = lapply(GRLS,length))
##
## ALL COMMON
##

message(" OVERLAPPING: all common")
SEQ.THREE.COMMON <- Reduce(BiocGenerics::intersect,GRLS)

message("\tfiltering orginal GRL")
THREE.COMMON.GRL <- lapply(names(GRL),function(i){
  GRL[[i]][mcols(GRL[[i]])[["seq"]] %in% SEQ.THREE.COMMON] })
names(THREE.COMMON.GRL) <- paste0(names(GRL),"_COMMON")

# remove seq column
if(isTRUE(REMOVE.SEQ)){
  THREE.COMMON.GRL <- lapply(THREE.COMMON.GRL,function(i){
    GR <- i; mcols(GR)[["seq"]] <- NULL;return(GR)}) }

##
## ALL EXCLUSIVE
##

message(" OVERLAPPING: all exclusive")
SEQ.THREE.EXCLUSIVE <- lapply(TEMP.NAMES,function(i){
  TEMP.NAME.ORDER <- c(i,TEMP.NAMES[!TEMP.NAMES %in% i])
  AN.EXCLUSIVE.SEQ <- Reduce(BiocGenerics::setdiff, GRLS[TEMP.NAME.ORDER])
  return(AN.EXCLUSIVE.SEQ)})

names(SEQ.THREE.EXCLUSIVE)  <- paste0(names(GRL))

message("\tfiltering orginal GRL")
THREE.EXCLUSIVE.GRL <- lapply(names(GRL), function(i){
  GRL[[i]][mcols(GRL[[i]])[["seq"]] %in% SEQ.THREE.EXCLUSIVE[[i]]] })
names(THREE.EXCLUSIVE.GRL)  <- paste0(names(GRL),"_EXCLUSIVE")

# remove seq column
if(isTRUE(REMOVE.SEQv)){
  THREE.EXCLUSIVE.GRL <- lapply(THREE.EXCLUSIVE.GRL,function(i){
    GR <- i; mcols(GR)[["seq"]] <- NULL;return(GR)}) }

##
## ALL PAIRWISE
```

```
  ##

  message(" OVERLAPPING: all pairwise")
  PW.ORDER <- list(c("FA","FB"),c("FB","FC"),c("FC","FA"))
  names(PW.ORDER) <- c("AB","BC","CA")
  SEQ.THREE.PAIRWISE <- lapply(PW.ORDER,function(i){
    A.PAIRWISE.SEQ <- Reduce(BiocGenerics::intersect, list(GRLS[[i[1]]],GRLS[[i[[2]]]]))
    return(A.PAIRWISE.SEQ)})
  names(SEQ.THREE.PAIRWISE)  <- names(PW.ORDER)

  message("\tfiltering orginal GRL")
  PAIRWISE.GRL <- lapply(names(PW.ORDER), function(i){
    PAIR.REAL.NAMES <- unlist(lapply(PW.ORDER[[i]],function(j){NAME.MAP[tempName %in% j][["realName"]]}
    PAIR.GRL <- lapply(PAIR.REAL.NAMES, function(k){
      GRL[[k]][mcols(GRL[[k]])[["seq"]] %in% SEQ.THREE.PAIRWISE[[i]]] })
    names(PAIR.GRL) <- paste0(PAIR.REAL.NAMES,"_",i,"-PAIRWISE")

    # remove seq column
    if(isTRUE(REMOVE.SEQ)){
      PAIR.GRL <- lapply(PAIR.GRL,function(i){
        GR <- i; mcols(GR)[["seq"]] <- NULL;return(GR)}) }

    return(PAIR.GRL) })
  names(PAIRWISE.GRL) <- names(PW.ORDER)

  ## COMPILE SUMMARIES

  message("\tpreparing summaries")
  NAME.MAP[["Common_N"]] <- length(SEQ.THREE.COMMON)
  NAME.MAP[["Exclusive_N"]] <- lapply(SEQ.THREE.EXCLUSIVE,length)
  NAME.MAP[["Pairwise"]] <- names(PW.ORDER)
  NAME.MAP[["Pairwise_N"]] <- lapply(SEQ.THREE.PAIRWISE,length)

  ## COMPILE RESULTS & RETURN

  RES.L <- list("Common"=THREE.COMMON.GRL,
                "Exclusive"=THREE.EXCLUSIVE.GRL,
                "Pairwise"=PAIRWISE.GRL,
                "SampleInfo"=NAME.MAP)

  message("\treturning list of GRanges"); message("Completed.")
  message(paste0("\n","Samples Info"))
  print(NAME.MAP)
  return(RES.L)
}
```

Function: twoSampleSequenceOverlaps()

```
twoSampleSequenceOverlaps <- function(
                                     ## INPUT
                                     GRL=NULL,
                                     BSSPECIES=NULL,

                                     ## SETTINGS
                                     MC.CORES=4,

                                     ## OUTPUT
                                     REMOVE.SEQ=TRUE
                                     ){

 ## AUTHOR: Pavol Genzor
 ## Use: Calculate two sample overlaps
 ## 06.28.21; Version 3; refined

 ## load libraries
 suppressPackageStartupMessages({library("data.table");library("parallel");
   library("GenomicRanges");library("GenomicAlignments");library("BSgenome.Hsapiens.UCSC.hg38");
   library("BSgenome.Dmelanogaster.UCSC.dm6");library("BSgenome.Mmusculus.UCSC.mm10")})

 ## INPUT CHECKING
 if(is.null(GRL)) stop("Please provide named GRanges list !")
 if(is.null(BSSPECIES)) stop("Please provide BSSPECIES name !")
 if(!isTRUE(class(GRL) %in% "list")) stop("GRL must be a named LIST !")
 if(is.null(names(GRL))) stop("GRP must be a NAMED list !!!")
 if(!isTRUE(unique(unlist(lapply(GRL,class)) %in% "GRanges"))) stop("All the components must be GRange

 message("Processing ...")

 ## GET SEQUENCES
 message("\textracting sequence")
 GRLS <- mclapply(names(GRL), mc.cores = MC.CORES, function(i){
   BSgenome::getSeq(eval(parse(text = BSSPECIES)),GRL[[i]]) })
 names(GRLS) <- names(GRL)

 message("\tchecking sequence duplication")
 for(i in names(GRLS)) {print(table(duplicated(GRLS[[i]]))  )}

 message("\t\nadding sequence to GRanges")
 for(i in names(GRLS)) {mcols(GRL[[i]])[["seq"]] <- GRLS[[i]]}

 message("\tmaking name map and assigning")
 TEMP.NAMES <- c("FA","FB")
 names(GRLS) <- TEMP.NAMES
 NAME.MAP <- data.table(tempName = TEMP.NAMES,
                        realName = names(GRL),
                        Input_N = lapply(GRLS,length))
 ##
 ## ALL COMMON
 ##

 message(" OVERLAPPING: all common")
 SEQ.TWO.COMMON <- Reduce(BiocGenerics::intersect,GRLS)

 message("\tfiltering orginal GRL")
```

```
  TWO.COMMON.GRL <- lapply(names(GRL),function(i){
    A.GR <- GRL[[i]]
    N.GR <- A.GR[mcols(A.GR)[["seq"]] %in% SEQ.TWO.COMMON]

    ## remove sequence
    if(isTRUE(REMOVE.SEQ)){mcols(N.GR)[["seq"]] <- NULL}

    ## return
    return(N.GR) })
  names(TWO.COMMON.GRL) <- paste0(names(GRL),"_COMMON")

  ##
  ## ALL EXCLUSIVE
  ##

  message(" OVERLAPPING: all exclusive")
  SEQ.TWO.EXCLUSIVE <- lapply(TEMP.NAMES,function(i){
    TEMP.NAME.ORDER <- c(i,TEMP.NAMES[!TEMP.NAMES %in% i])
    AN.EXCLUSIVE.SEQ <- Reduce(BiocGenerics::setdiff, GRLS[TEMP.NAME.ORDER])
    return(AN.EXCLUSIVE.SEQ)})
  names(SEQ.TWO.EXCLUSIVE)  <- paste0(names(GRL))

  message("\tfiltering orginal GRL")
  TWO.EXCLUSIVE.GRL <- lapply(names(GRL), function(i){
    A.GR <- GRL[[i]]
    N.GR <- A.GR[mcols(A.GR)[["seq"]] %in% SEQ.TWO.EXCLUSIVE[[i]]]

    ## remove sequence
    if(isTRUE(REMOVE.SEQ)){mcols(N.GR)[["seq"]] <- NULL}

    ## return
    return(N.GR)})
  names(TWO.EXCLUSIVE.GRL)  <- paste0(names(GRL),"_EXCLUSIVE")

  ##
  ## COMPILE SUMMARIES
  ##

  message(" preparing summaries")
  NAME.MAP[["Common_N"]] <- length(SEQ.TWO.COMMON)
  NAME.MAP[["Exclusive_N"]] <- lapply(SEQ.TWO.EXCLUSIVE,length)

  ## CREATE A RESULTS LIST
  RES.L <- list(TWO.COMMON.GRL, TWO.EXCLUSIVE.GRL, NAME.MAP)
  names(RES.L) <- c("Common","Exclusive","SampleInfo"); message("Completed.")
  message(paste0("\n","Samples Info"))
  print(NAME.MAP)

  ## RETURN
  return(RES.L)
}
```

Function: simpleGRFilter()

```r
simpleGRFilter <- function(
                          ## INPUT
                          GR = NULL,
                          RANGE.NAME = NULL,

                          ## FILTERS
                          NH.TAG = NULL,
                          SIZE.RANGE = NULL,
                          STRAND = NULL,

                          # OPTIONS
                          SEQNAMES.SPLIT = ":",
                          L.STRAND.POSITION = 3){

  ## AUTHOR: Pavol Genzor
  ## Use: Function to filter a single GR
  ## 06.28.21; Version 2, refined

  ## Libraries
  suppressPackageStartupMessages({library("data.table");library("plyr");
    library("Biostrings");library("GenomicRanges")})

  ## Check input
  if(is.null(GR)) stop("Please provide a GR !")
  if(ncol(mcols(GR)) == 0) stop("GR needs to have mcols() !")
  if(!"NH" %in% colnames(mcols(GR))) stop("mcols() need to have NH column !")
  if(!"MULT" %in% colnames(mcols(GR))) stop("mcols() need to have MULT column !")
  FILTER.LIST <- c("NH.TAG"=NH.TAG,"SIZE.RANGE" = SIZE.RANGE,"STRAND" = STRAND)
  if(is.null(FILTER.LIST)) stop("Please provide at least one filter criteria !")

  ## PROCEED
  message(paste0(" FUN: filtering..."))

  ## MAPPING
  if(!is.null(NH.TAG)){
    message(paste0("\tselect mappers: ",NH.TAG))
    GR <- GR[mcols(GR)[["NH"]] %in% NH.TAG]
  } else { message("\tselect mappers: all") }

  ## SIZE RANGE
  if(!is.null(SIZE.RANGE)){
    message(paste0("\tsize range used: ", min(SIZE.RANGE),"-", max(SIZE.RANGE)))
    SIZE.RANGE <- seq(min(SIZE.RANGE),max(SIZE.RANGE))
    GR <- GR[width(GR) %in% SIZE.RANGE]
  } else { message("\tall sizes") }

  ## STRAND
  if(!is.null(STRAND)){
    message("\tmatched strand: YES")
    mcols(GR)[["STRAND"]] <- nth(tstrsplit(seqnames(GR), split=SEQNAMES.SPLIT),L.STRAND.POSITION)
    GR <- GR[strand(GR) == mcols(GR)[["STRAND"]]]
  } else { message("\tmatched strand: NO") }
  ## RETURN
  return(GR)
}
```

Function: simpleStepsPlot()

```

simpleStepsPlot <- function(
                            ## INPUT
                            GRL = NULL,

                            ## OPTIONS
                            Y.PPM = FALSE,

                            ## SETTINGS
                            RETURN.ALL = FALSE,
                            COLORS = scale_color_brewer(palette = "Dark2"),
                            LWD.STEP = 1.75,
                            Y.TRANS = "log10",
                            X.LIMS = NULL,
                            Y.LIMS = NULL,
                            FAM = "Helvetica",
                            TCOL = "black",
                            XYT = 12,
                            ASPECT.RATIO = 1,
                            LEGEND.POSITION = "bottom",

                            ## RETURN
                            RETUN.ALL = FALSE){

  ## AUTHORS: Pavol Genzor and Daniel Stoyko
  ## Use: Sript to make Steps plots
  ## 06.28.21; Version 3; refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("dplyr");
    library("ggplot2");})

  ## INPUT CHECKING
  if(is.null(GRL)) stop("Please provide a GRL !")
  if(!is.list(GRL)) stop("GRL needs to be a list !")
  if(is.null(names(GRL))) stop("GRL has to be named !")

  ## LOOP THROUGH SAMPLES
  SAMPLE.L <- lapply(names(GRL), function(s){

    ## Subset
    GR <- GRL[[s]]
    if(!"MULT" %in% colnames(mcols(GR))) stop("MULT column is missing !")
    ## Process
    DT <- as.data.table(mcols(GR)[["MULT"]])[,.N, by = "V1"]
    colnames(DT) <- c("MULT","SEQ")
    DT[,READ := MULT*SEQ]
    DT[,fSEQ := (SEQ/sum(SEQ))]
    DT[,fREAD := (READ/sum(READ))]
    setorderv(x = DT, cols = "MULT", order = -1)
    DT[,afSEQ := Reduce('+', fSEQ, accumulate = T)]
    DT[,afREAD := Reduce('+', fREAD, accumulate = T)]
    DT[["ORG.MULT"]] <- DT[["MULT"]]
    DT[["SAMPLE"]] <- s

    ## CONVERT ABUNDANCE TO PPM
    if(isTRUE(Y.PPM)){
      message("... MULT to PPM conversion (adjust Y.LIMS) ")
      DT[["MULT"]] <- (DT[["MULT"]] / sum(DT[["READ"]])) * 1000000}

    ## Return
    return(DT)})

  ## Y LABELS: MULT vs PPM
  if(isTRUE(Y.PPM)){ Y.LAB <- ylab("MULT.PPM") } else { Y.LAB<- ylab("MULT") }

  ## Combine samples
  LDT <- rbindlist(SAMPLE.L)

  ## Melt & Prepare
  DTM <- melt(data = LDT,
              id.vars = c("MULT","SAMPLE"),
              measure.vars = c("afSEQ","afREAD"),
              value.name = "FRACTION", variable.name = "TYPE")
  DTM[["SAMPLE"]] <- factor(DTM[["SAMPLE"]], levels = unique(DTM[["SAMPLE"]]))

  ## PLOT
  SGG <- ggplot() + theme_bw() +
    ## DATA
    geom_step(data = DTM, aes(x = FRACTION, y = MULT, alpha = TYPE, colour = SAMPLE),
              lwd = LWD.STEP) +

    ##COLORS
    COLORS + Y.LAB +

    ## SCALES
    scale_alpha_discrete(range=c(0.5, 1)) +
    scale_y_continuous(limits = Y.LIMS, trans = Y.TRANS) +
    scale_x_continuous(limits = X.LIMS, breaks = seq(0,1,0.1)) +
    annotation_logticks(sides = "l", ) +

    ## THEME
    theme(aspect.ratio = 1, legend.position = "bottom",
          panel.grid = element_blank(), panel.border = element_blank(),
          axis.title = element_text(family = FAM, colour = TCOL, size = XYT),
          axis.text = element_text(family = FAM, color = TCOL, size = XYT),
```

```r
          axis.ticks.y = element_blank())

  ## RESULTS
  RES.L <- list("sampleList" = SAMPLE.L, "plotData" = DTM, "ggplot" = SGG)

  ## RETURN
  if(isTRUE(RETURN.ALL)){ return(RES.L) } else { return(RES.L[["ggplot"]]) }
}
```

Function: simpleMultBarPlot()

```r


simpleMultBarPlot <- function(GR = NULL,
                              RANGE.NAME = NULL,

                              ## SETTINGS
                              COLORS = c("orange","white"),
                              FAM = "Helvetica",
                              TCOL = "black",
                              XYT = 12,
                              ASPECT.RATIO = 0.1,
                              BAR.WIDTH = 0.75,

                              ## RETURNS
                              RETURN.ALL = FALSE){


  ## AUTHOR: Pavol Genzor
  ## Use: Sript to make simple barplot from multiplicities (to supplement STEPS plot)
  ## 06.28.2021; Version 2; refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("dplyr");
    library("ggplot2");library("ggpubr")})

  ## INPUT CHECKING
  if(is.null(GR)) stop("Please provide a GRL !")
  if(is.null(RANGE.NAME)) stop("Please provide a RANGE.NAME !")

  ## Process data
  DT <- as.data.table(mcols(GR)[["MULT"]])[,.N, by = "V1"]
  colnames(DT) <- c("MULT","SEQ")
  DT[,READ := MULT*SEQ]
  DT[,fSEQ := (SEQ/sum(SEQ))]
  DT[,fREAD := (READ/sum(READ))]
  setorderv(x = DT, cols = "MULT", order = -1)
  DT[,afSEQ := Reduce('+', fSEQ, accumulate = T)]
  DT[,afREAD := Reduce('+', fREAD, accumulate = T)]

  ## Reduce to relevant and melt
  DTS <- rbindlist(l = list(DT[MULT %in% 1],DT[!MULT %in% 1][,lapply(.SD,sum)]))
  DTS[["MULT"]] <- factor(x = c("one","rest"), levels = c("rest","one"))
  DTM <- melt(DTS,id.vars = "MULT",
              measure.vars = c("fSEQ","fREAD"), variable.name = "type",
              value.name = "frequency")
  DTM[["type"]] <- factor(x = DTM[["type"]] , levels = c("fREAD","fSEQ"))

  ## Plot
  GGBAR <- ggplot() + theme_pubclean() +
    ## Data
    geom_bar(data = DTM, aes(x = type, y = frequency, fill = MULT),
             colour = "black",stat = "identity", width = BAR.WIDTH) +
    ## Label
    geom_text(data = DTM, aes(x = type, y = frequency, label = 100*round(frequency,digits = 3)),
              position = position_stack(vjust = 0.5),
              colour=TCOL, family=FAM) +
    ## Colors
    scale_fill_manual(values = COLORS) +
    ## Coordinates & title
    coord_flip() + ggtitle(paste0(RANGE.NAME)) +
    ## Theme
    theme(aspect.ratio = ASPECT.RATIO, panel.grid = element_blank(),
          axis.text.x = element_text(family = FAM, colour = TCOL, size = XYT,
                                     angle = 90, vjust = 0.5, hjust = 1),
          axis.text.y = element_text(family = FAM, colour = TCOL, size = XYT))

  ## Results
  RES.L <- list("plotData" = DTM, "ggplot" = GGBAR)
  if(isTRUE(RETURN.ALL)) { return(RES.L) }
  else { return(RES.L[["ggplot"]]) }
}
```

Function: simpleSRViolin()

```
simpleSRViolin <- function(
                              ## INPUT
                              GRL = NULL,

                              ## OPTIONS
                              Y.PPM = FALSE,

                              ## SETTINGS
                              SAMPLE.ORDER = names(GRL),
                              SIZE.RANGE = c(18,50),
                              NH.TAG = NULL,
                              TOTAL.SAMPLE="Total",

                              ## OUTPUT
                              ADD.TABLE = TRUE,
                              SIG.FIGURES = 2,
                              SOURCE.DIR = NULL,
                              RETURN.ALL = FALSE,
                              ## PLOT SETTINGS
                              Y.LIMS = NULL,
                              VIOLIN.ADJUST = 1,
                              VIOLIN.SCALE = "width",
                              ASPECT.RATIO = 1,
                              LEGEND.POSITION = "right",
                              TCOL = "black",FAM = "Helvetica",XYT = 12,
                              TRANS = "log10",PLOT.COLORS = c("orange1","grey70")){

## AUTHORS: Pavol Genzor & Daniel Stoyko
## Use: Calculate and plot sequence/read violings
## 06.28.21; Version 6; fixed and refined

## LIBRARIES
suppressPackageStartupMessages({
  library("data.table"); library("dplyr"); library("GenomicRanges")
  library("ggplot2"); library("tidyverse"); library("reshape");
  library("gridExtra");library("ggpubr")})

## INPUT CHECKING
if(is.null(GRL)) stop("Please provide a named GRL object !")
if(isFALSE(is.list(GRL))) stop("Input is not a GRL - Please proide a list of GRanges !")
if(is.null(names(GRL))) stop("Please make sure that GRL is named !")
if(is.null(SOURCE.DIR)) stop("Please provide a SOURCE.DIR with functions !")
if(!"MULT" %in% colnames(mcols(GRL[[1]]))) stop("GRL needs to have MULT column !")
if(!"NH" %in% colnames(mcols(GRL[[1]]))) stop("GRL needs to have NH column !")

## FUNCTION
source(paste0(SOURCE.DIR,"simpleGRFilter.R"))

## PROCEED
message("Processing...")

##
## DATA LOOP
##

GRL.DT.L <- lapply(names(GRL), function(i){

  ## SUBSET
  GR <- GRL[[i]]

  ## FILTER
  GR <- simpleGRFilter(GR = GR,
                       RANGE.NAME = i,
                       NH.TAG = NH.TAG,
                       SIZE.RANGE = SIZE.RANGE)

  ## PREP THE TABLE
  GR.DT <- as.data.table(GR)
  NDT <- GR.DT[,.N, by = "MULT"]
  setorderv(x = NDT, cols = "MULT")
  colnames(NDT) <- c("MULT","SEQ")
  ## CALCULATE
  NDT[, READ := MULT * SEQ]
  NDT[, SAMPLE := i]

  ## REARRANGE
  mNDT <- setDT(melt.data.table(data = NDT,
                                id.vars = c("MULT","SAMPLE"),
                                variable.name = "TYPE",
                                value.name = "COUNT"))
  ## RETURN
  return(mNDT) })

## COMNBINE AND ORGANIZE FOR PLOTTING
message(" calculating & plotting")
GR.DT <- rbindlist(GRL.DT.L)
GR.DT[["SAMPLE"]] <- factor(GR.DT[["SAMPLE"]], levels = SAMPLE.ORDER)
GR.DT[["TYPE"]] <- factor(GR.DT[["TYPE"]], levels = c("SEQ","READ"))
GR.DT[["GROUP"]] <- paste(GR.DT[["SAMPLE"]],GR.DT[["TYPE"]], sep = "_")
for(i in unique(GR.DT[["GROUP"]])) set(x = setDT(GR.DT), i = which(GR.DT[["GROUP"]] %in% i),
                                       j = "GROUP_SUM", value = sum(GR.DT[GROUP %in% i][["COUNT"]]))
setDT(GR.DT)

  ##
## TABLE FOR PLOT
##

TAB.DT <- GR.DT[, sum(.SD), by = c("SAMPLE","TYPE"), .SDcols = "COUNT"]
for(t in unique(TAB.DT[["TYPE"]])) set(x = setDT(TAB.DT),
                                       i = which(TAB.DT[["TYPE"]] %in% t),
                                       j = "FRACTION",
                                       value = round(TAB.DT[TYPE %in% t][["V1"]]/
                                                       TAB.DT[TYPE %in% t & SAMPLE %in% TOTAL.SAMPLE]
                                                     digits = SIG.FIGURES))
TAB.W <- dcast.data.table(data = TAB.DT, formula = TYPE ~ SAMPLE, value.var = "FRACTION")

## CONVERT TO PPM
if(isTRUE(Y.PPM)){
  message("... MULT to PPM conversion (adjust Y.LIMS) ")
  NOMINATOR <- GR.DT[["MULT"]]
  DENOMINATOR <- unique(GR.DT[GROUP %in% paste(TOTAL.SAMPLE,"READ",sep="_")][["GROUP_SUM"]])
  PPMs <- ((NOMINATOR/DENOMINATOR) * 1000000)
  GR.DT[["MULT"]] <- PPMs
  Y.LAB <- ylab("MULT.PPM")
} else { Y.LAB <- ylab("MULT") }

##
## VIOLIN PLOT
##

GG.VIOL <-  ggplot() + theme_pubclean() +
  ## DATA
  geom_violin(data = GR.DT, aes(x = SAMPLE, y = MULT, weight = COUNT/GROUP_SUM, fill = TYPE),
              color="black", lwd = 0.25, adjust= VIOLIN.ADJUST, scale = VIOLIN.SCALE,
                position = position_dodge(width = 1)) +

    ## TITLE
    ggtitle(paste0("Violins settings\n",
                   paste0("size range: ", min(SIZE.RANGE)," - ", max(SIZE.RANGE)),"\n",
                   paste0("used NH tags: ", ifelse(is.null(NH.TAG),"all",NH.TAG) ))) + Y.LAB +

    ## SCALES
    scale_fill_manual(values = PLOT.COLORS) +
    scale_y_continuous(trans = TRANS, limits = Y.LIMS) +
    annotation_logticks(sides = "l") +
    theme(aspect.ratio = ASPECT.RATIO,
          axis.ticks.y = element_blank(),
          legend.position = LEGEND.POSITION,
          panel.grid = element_blank(),
          axis.text = element_text(family = FAM, color =TCOL, size = XYT))

  ## CONVERT INTO GROB
  if(isTRUE(ADD.TABLE)){
    TBL <- tableGrob(d = TAB.W, theme = ttheme_minimal(), rows = NULL)
    GG.VIOL <- grid.arrange(GG.VIOL,TBL, nrow = 2, heights = c(5,1))}

  ## RETURN
  message("Done.")
  if(isTRUE(RETURN.ALL)){
    RES.L <- list("plotData" = GR.DT,"ggplot" = GG.VIOL)
    return(RES.L)
  } else { return(GG.VIOL) }
}
```

Function: simpleMetageneTab()

```
simpleMetageneTab <- function(
                              ## INPUT
                              GR = NULL,
                              RANGE.NAME = NULL,
                              BSSPECIES = NULL,

                              ## OPTIONS
                              USE.READS = TRUE,
                              ALIGN.END = NULL,
                              EXPAND.BY = NULL,

                              ## OUTPUT OPTIONS
                              RETURN.TWO.TABLES = TRUE,

                              ## SETTINGS
                              ALPHABET = c("T","C","G","A")){

  ## AUTHOR: Pavol Genzor
  ## Use: Attain metagene table from a GenomicRange aligned to 5- or 3-prime end
```

```r
## 06.30.21; Version 4; Refined

## LIBRARIES
suppressPackageStartupMessages({library("data.table");library("ShortRead"); library("BSgenome");
  library("GenomicRanges"); library("hiReadsProcessor"); library("BSgenome.Dmelanogaster.UCSC.dm6");
  library("BSgenome.Mmusculus.UCSC.mm10"); library("BSgenome.Hsapiens.UCSC.hg38")})

## INPUT CHECKING
if(is.null(GR)){stop("Provide filtered read GR !")}
if(is.null(RANGE.NAME)){stop("Provide file name to RANGE.NAME !")}
if(is.null(BSSPECIES)){stop("Provide BSpecies name !!!")}
if(is.null(EXPAND.BY)){stop("Provide distance value to EXPANDBY !")}
if(is.null(ALIGN.END)){stop("Provide the end of small RNA to align. '5' or '3' !")}

message("Processing ...")
message(paste0(" sample: ",RANGE.NAME))

## EXTEND BOUNDARIES
expandLimits <- c(-EXPAND.BY, EXPAND.BY)

if(ALIGN.END == 5){
  ## #1 is first nucleotide
  message(" aligning to 5‘ end")
  GR <- GenomicRanges::resize(GR, width = EXPAND.BY, fix = "start")
  GR <- GenomicRanges::resize(GR, width = EXPAND.BY*2, fix = "end") }

if(ALIGN.END == 3){
  ## #1 is +1 nucleotide
  message(" aligning to 3‘ end")
  GR <- GenomicRanges::resize(GR, width = EXPAND.BY, fix = "end")
  GR <- GenomicRanges::resize(GR, width = EXPAND.BY*2, fix = "start") }

message(" removing unused contigs")
GNM <- BSgenome::getBSgenome(eval(parse(text = BSSPECIES)))
GNM.GR <- GRanges(seqnames = names(GNM),
                  ranges = IRanges(start = 1, end = GenomeInfoDb::seqlengths(GNM)))
GR <- IRanges::subsetByOverlaps(GR, GNM.GR, type = "within")

message(" getting sequences (slow)")
GRS <- BSgenome::getSeq(eval(parse(text = BSSPECIES)), GR)

if(isTRUE(USE.READS)){
  message(" expanding sequences to reads")
  GRS <- replicateReads(GRS, mcols(GR)[["MULT"]]) }

message(" acquiring counts")
GR.CNT <- as.data.table(ShortRead::alphabetByCycle(GRS, alphabet = ALPHABET))
GR.CNT[["NAME"]] <- paste("count",RANGE.NAME,gsub("T","U",ALPHABET), sep = "__")
data.table::setcolorder(GR.CNT, c("NAME",colnames(GR.CNT)[!colnames(GR.CNT) %in% "NAME"]))

message(" calculating frequencies")
GR.FREQ <- data.table(NAME = GR.CNT[["NAME"]],
                      GR.CNT[,(.SD / lapply(.SD,sum))*100,
                                .SDcols = colnames(GR.CNT)[!colnames(GR.CNT) %in% "NAME"] ])
  GR.FREQ[["NAME"]] <- gsub("count","frequency",GR.FREQ[["NAME"]])

  message(" combining and labeling")
  GR.META <- data.table(data.table::rbindlist(list(GR.CNT,GR.FREQ)))
  colnames(GR.META) <- c("NAME",as.character(c(seq(expandLimits[1],-1),seq(1,expandLimits[2]))))


  ## RETURN
  message("Done.")

  if(isTRUE(RETURN.TWO.TABLES)){
    TWO.TABLES.L <- list(GR.META[grep("count",GR.META[["NAME"]]),],GR.META[grep("frequency",GR.META[["N
    names(TWO.TABLES.L) <- c("count","frequency")
    return(TWO.TABLES.L) }
  else{ return(GR.META) }
}
```

Function: simpleMetageneRegularPlotGG()

```


simpleMetageneRegularPlotGG <- function(
                                    ## INPUT
                                    METAGENE.DT=NULL,
                                    SAMPLE.NAME=NULL,
                                    PLOT.BY.VALUE="frequency",

                                    ## SETTINGS
                                    ID.VAR="NAME",

                                    ## PLOT SETTINGS
                                    PIRNA.SIZE=26,
                                    Y.LIMITS=NULL,
                                    Y.BREAKS=seq(0,100,10),
                                    NUC.COLORS = rev(c("firebrick1","dodgerblue1","goldenrod1","forestg
                                    ASPECT.RATIO = 1,

                                    ## PLOT SETTINGS
                                    FAM="Helvetica",
                                    XYT=12,
                                    TCOL="black",
                                    LEGEND.POSITION="bottom"
                                    ){

  ## AUTHOR: Pavol Genzor
  ## Use: Plot simple metagene plot
  ## 06.30.21; Version 6; refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("dplyr"); library("ggrepel")
    library("ggplot2")})
```

```
## INPUT CHECKING
if(is.null(METAGENE.DT)) stop("Provide a metagene frequency table !!!")
if(sum(METAGENE.DT[[2]]) != 100) stop("This is not a frequency table !!!")
if(is.null(SAMPLE.NAME)) stop("Provide metagene SAMPLE.NAME")

## PROCESS DATA
message("Processing ...")
MGm <- data.table::melt(METAGENE.DT,
                        id.vars = ID.VAR,
                        value.name = PLOT.BY.VALUE,
                        variable.name = "POSITION")
MGm[["POSITION"]] <- as.numeric(as.character(MGm[["POSITION"]]))
MGm[["SAMPLE"]] <- dplyr::nth(data.table::tstrsplit(MGm[[ID.VAR]],split="__"),2)
MGm[["NUCLEOTIDE"]] <- dplyr::nth(data.table::tstrsplit(MGm[[ID.VAR]],split="__"),-1)

## METRICS & SETTINGS
MAX.FREQ.U <- MGm[,lapply(.SD, max), by = NUCLEOTIDE, .SDcols = "frequency"][NUCLEOTIDE %in% "U"][["f
FIRST.NUC <- MGm[NUCLEOTIDE %in% "U" & POSITION %in% 1]
X.BREAKS <- c(-PIRNA.SIZE,1,PIRNA.SIZE)

## PLOT
message(" plotting")
SMG <- ggplot() + theme_bw() +

  ## PIRNA LINES
  geom_vline(xintercept = c(-PIRNA.SIZE,1,PIRNA.SIZE), colour = "black", linetype = "dashed") +

  ## DATA
  geom_line(data = MGm, aes_string(x = "POSITION", y = PLOT.BY.VALUE, colour = "NUCLEOTIDE", group =

  ## U FREQ MAX
  geom_text_repel(data = FIRST.NUC, aes(x = POSITION, y = eval(parse(text = PLOT.BY.VALUE)),
                                        label = round(eval(parse(text = PLOT.BY.VALUE)), digits = 1))
                  box.padding = 2, nudge_x = 2) +

  ## TITLES
  ggtitle(SAMPLE.NAME) + xlab("distance from 1N (nt)") + ylab("nucleotide frequency (%)") +

  ## SCALES
  scale_colour_manual(values = NUC.COLORS) +
  scale_x_continuous(breaks = X.BREAKS) +
  scale_y_continuous(limits = Y.LIMITS, breaks = Y.BREAKS) +

  ## THEME
  theme(panel.grid = element_blank(),
        panel.border = element_blank(),
        legend.position = LEGEND.POSITION,
        aspect.ratio = ASPECT.RATIO,
        plot.title = element_text(family = FAM, size = XYT, colour = TCOL),
        axis.text = element_text(family = FAM, size = XYT, colour = TCOL),
        axis.title = element_text(family = FAM, size = XYT, colour = TCOL))

message("Done.")
```

```
  return(SMG)
}
```

Function: simpleSDfromGR()

```
simpleSDfromGR <- function(
                            ## INPUT
                            GR=NULL,
                            SAMPLE.NAME=NULL,

                            ## OPTIONS
                            USE.READS=TRUE,
                            PLOT.FREQ=TRUE,
                            YLIMS=NULL,

                            ## SETTINGS
                            ASPECT.RATIO=1,
                            X.BREAKS.BY=1,
                            BAR.FILL="grey80",
                            BAR.LINE="black",
                            XYT=8,
                            FAM="Helvetica",
                            TCOL="black",
                            RETURN.ALL=FALSE){

  ## AUTHOR: Pavol Genzor
  ## Use: Generate simple size distribution plot from GR
  ## 06.30.21; Version 4; refined

  ## LIBRARIES
  suppressPackageStartupMessages({library(data.table);library(ggplot2);library(GenomicRanges)})

  ## INPUT CHECKING
  if(is.null(GR)) stop("Please provide a GR  made from .bam file !")
  if(is.null(SAMPLE.NAME)) stop("Please provide SAMPLE.NAME !")
  if(!"MULT" %in% colnames(mcols(GR))) stop("The GRanges must have MULT column!")

  ## OPTIONS
  PLOT.ON.Y=ifelse(isTRUE(PLOT.FREQ),"FREQ","COUNT")

  ## MAKE TABLE
  GR.DT  <- as.data.table(GR)

  ## COUNT SEQ OR READ
  if(isTRUE(USE.READS)){
    message("using reads")
    METRIC.USED = "Read"
    SD.DT <- GR.DT[, lapply(.SD,sum), by = "width", .SDcols = "MULT"] }
  else {
    message("using sequences")
    METRIC.USED = "Sequence"
    SD.DT <- GR.DT[,.N, by = "width"] }

  colnames(SD.DT) <- c("SIZE","COUNT")
  SD.DT[, FREQ := lapply(.SD,function(i){(i/sum(.SD))*100 }),.SDcols = c("COUNT")]
  SD.DT[["SIZE"]] <- as.numeric(SD.DT[["SIZE"]])

  ## PLOT
  SD.GG <- ggplot() + theme_pubclean() +
    geom_bar(data = SD.DT, aes_string(x = "SIZE", y = PLOT.ON.Y),
             stat = "identity", colour = BAR.LINE, fill = BAR.FILL) +
    scale_x_continuous(breaks = seq(15,50,X.BREAKS.BY)) +
    scale_y_continuous(labels = function(x) format(x, scientific = TRUE),
                       limits = YLIMS,
                       breaks = seq(0,100,10)) +
    ggtitle(paste(METRIC.USED,SAMPLE.NAME, sep = "; ")) +
    theme(panel.grid = element_blank(),
          title = element_text(family = FAM, size = XYT, colour = TCOL),
          axis.text = element_text(family = FAM, size = XYT, colour = TCOL),
          axis.title = element_text(family = FAM, size = XYT, colour = TCOL),
          panel.background = element_rect(fill = "transparent",colour = NA),
          plot.background = element_rect(fill = "transparent",colour = NA),
          aspect.ratio = ASPECT.RATIO)

  ## RETURN
  if(isTRUE(RETURN.ALL)){
    RES.L <- list(SD.DT,SD.GG)
    names(RES.L) <- c(paste0(METRIC.USED,"_data_table"),"ggplot")
    return(RES.L)
  } else { return(SD.GG) }
}
```

Function: simpleClusterSRCountsPerGR()

```
simpleClusterSRCountsPerGR <- function(
                                      ## INPUT
                                      GR = NULL,
                                      RANGE.NAME = NULL,
                                      CLUSTER.NAME.COLUMN = "seqnames",
                                      SAMPLE.SEP = "_"){

  ## AUTHOR: Pavol Genzor
  ## Use: Function to get sequence and read counts per cluster
  ## 07.07.2021; Version 2; refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("plyr"); library("dplyr");.
    library("GenomicRanges"); library("reshape")})

  ## INPUT CHECKING
  if(is.null(GR)) stop("Please provide a GR!")
  if(is.null(RANGE.NAME)) stop("Please provide a RANGE.NAME!")
  if(isFALSE("MULT" %in% colnames(mcols(GR)))) stop("GR must have MULT column!")

  message(" FUN: cluster counts")
  a.DT <- as.data.table(GR)

  ## sequences
  seq.count.DT <- a.DT[,.N, by = CLUSTER.NAME.COLUMN]
  colnames(seq.count.DT) <- c("CLUSTER", paste("seq.count", RANGE.NAME, sep = SAMPLE.SEP))
  seq.count.DT[,paste("seq.total",RANGE.NAME, sep = SAMPLE.SEP) := length(GR)]

  ## reads
  read.count.DT <- a.DT[, lapply(.SD, function(i){sum(i)}), by = CLUSTER.NAME.COLUMN, .SDcols = "MULT"]
  colnames(read.count.DT) <- c("CLUSTER",paste("read.count", RANGE.NAME, sep = SAMPLE.SEP))
  read.count.DT[,paste("read.total",RANGE.NAME, sep = SAMPLE.SEP) := sum(mcols(GR)[["MULT"]])]

  ## join
  sr.count.DT <- setDT(join_all(dfs = list(seq.count.DT,read.count.DT), by = "CLUSTER", type = "full"))
  for (j in names(sr.count.DT)) set(sr.count.DT,which(is.na(sr.count.DT[[j]])),j,0)

  ## return
  return(sr.count.DT)
}
```

Function: simpleClusterStatsPerGR()

```
simpleClusterStatsPerGR <- function(
                                    ## INPUT
                                    GR = NULL,
                                    RANGE.NAME = NULL,
                                    CLUSTER.NAME.COLUMN = "seqnames",
                                    SAMPLE.SEP = "_"){


  ## AUTHOR: Pavol Genzor
  ## Use: Function to get mean and median stats per cluster
  ## 07.07.2021; Version 2; refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("plyr"); library("dplyr");.
    library("GenomicRanges"); library("reshape")})

  ## INPUT CHECKING
  if(is.null(GR)) stop("Please provide a GR!")
  if(is.null(RANGE.NAME)) stop("Please provide a RANGE.NAME!")
  if(isFALSE("MULT" %in% colnames(mcols(GR)))) stop("GR must have MULT column!")

  message(" FUN: cluster stats")
  a.DT <- as.data.table(GR)
```

```r
  ## calculate mean
  cluster.mean.MULT <- a.DT[,lapply(.SD, function(i){mean(i)}), by = CLUSTER.NAME.COLUMN, .SDcols = "MUI
  colnames(cluster.mean.MULT) <- c("CLUSTER",paste("mean.mult",RANGE.NAME, sep = SAMPLE.SEP))

  ## calculate median
  cluster.median.MULT <- a.DT[,lapply(.SD, function(i){median(as.double(i))}), by = CLUSTER.NAME.COLUMN
  colnames(cluster.median.MULT) <- c("CLUSTER",paste("median.mult",RANGE.NAME, sep = SAMPLE.SEP))

  ## compile
  cluster.stat.MULT <- setDT(join_all(dfs = list(cluster.mean.MULT, cluster.median.MULT), by = "CLUSTER

  ## return
  return(cluster.stat.MULT)
}
```

Function: simpleNewGRFromReadNames()

```r
simpleNewGRFromReadNames <- function(## INPUT
                                     GR = NULL,

                                     ## OPTIONS
                                     INCLUDE.N = FALSE,

                                     ## OPTIONS
                                     READ.NAMES.SPLIT = "_"){

  ## AUTHOR: Pavol Genzor
  ## Use: To make new GR from read names
  ## 07.07.21; Version 3; refined

  ## LIBRARIES
  suppressPackageStartupMessages({library(data.table);library(dplyr);library(plyr);library(GenomicRange

  ## INPUT CHECKING
  if(is.null(GR)) stop("Please provide a GR with read names !")
  if(!"NH" %in% colnames(mcols(GR)) || !"MULT" %in% colnames(mcols(GR)))
    stop("Make sure mcols(GR) have NH and MULT columns !")

  message(" FUN: decoding names")
  NAMES.L <- tstrsplit(names(GR),split = READ.NAMES.SPLIT)
  AGR <- GRanges(seqnames = NAMES.L[[1]],
                 ranges = IRanges(start = as.numeric(NAMES.L[[2]]), end = as.numeric(NAMES.L[[3]])),
                 strand = NAMES.L[[4]])
  names(AGR) <- names(GR)
  mcols(AGR)[["CLUSTER"]] <- as.character(seqnames(GR))

  ## Option: Include sample count
  if(isTRUE(INCLUDE.N)){
    mcols(AGR)[["N"]] <- nth(tstrsplit(NAMES.L[[5]], split = "N"),2) }

  mcols(AGR)[["NH"]] <- mcols(GR)[["NH"]]
  mcols(AGR)[["MULT"]] <- mcols(GR)[["MULT"]]

  ## RETURN
  return(AGR)
}
```

Function: simpleClusterFirstNucStatsPerGR()

```
simpleClusterFirstNucStatsPerGR <- function(
                                            ## INPUT
                                            GR = NULL,
                                            RANGE.NAME = NULL,
                                            CLUSTER.NAME.COLUMN = "seqnames",

                                            BSSPECIES = NULL,
                                            BSSPECIES.VERSION = "mm10",

                                            SAMPLE.SEP = "_",
                                            READ.NAMES.SPLIT = "_",
                                            ALPHABET = c("A","C","G","T"),
                                            SOURCE.DIR = NULL){

  ## AUTHOR: Pavol Genzor
  ## Use: Function to get statistics surrounding the first nucleotie of piRNAs per cluster
  ## 07.07.2021; Version 2; Refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("plyr"); library("dplyr"); library("reshape"); library("BSgenome");
    library("GenomicRanges"); library("GenomeInfoDb"); library("XVector"); library("Biostrings")  })

  ## CONDITIONAL LIBRARIES
  if(BSSPECIES.VERSION %in% "mm9"){
    if(any(grepl("package:BSgenome.Mmusculus.UCSC.mm10", search()))) detach("package:BSgenome.Mmusculus
    suppressWarnings(suppressMessages(library("BSgenome.Mmusculus.UCSC.mm9")))
  } else {
    if(any(grepl("package:BSgenome.Mmusculus.UCSC.mm9", search()))) detach("package:BSgenome.Mmusculus.U
    suppressWarnings(suppressMessages(library("BSgenome.Mmusculus.UCSC.mm10"))) }

  ## HOMEMADE FUNCTIONS
  source(paste0(SOURCE.DIR,"simpleNewGRFromReadNames.R"))

  ## INPUT CHECKING
  if(is.null(GR)) stop("Please provide a GR!")
  if(is.null(BSSPECIES)) stop("Please provide BSSPECIES!")
  if(is.null(RANGE.NAME)) stop("Please provide a RANGE.NAME!")
  if(isFALSE("MULT" %in% colnames(mcols(GR)))) stop("GR must have MULT column!")
  if(is.null(SOURCE.DIR)) stop("Please provide SOURCE.DIR with functions!")

  ## transpose with names
  N.GR <- simpleNewGRFromReadNames(GR = GR,
                                  READ.NAMES.SPLIT = READ.NAMES.SPLIT)

  message(" FUN: cluster counts")

  ## fix the levels
  seqlevels(N.GR) <- sortSeqlevels(seqlevels(N.GR))
  seqinfo(N.GR) <- keepStandardChromosomes(seqinfo(x = eval(parse(text = BSSPECIES))))

  ## get sequences
  message("\tgetting sequences ... slow")
  N.SEQ <- getSeq(eval(parse(text = BSSPECIES)), N.GR)

  ## add first name & nucleotide
  mcols(N.GR)[["seq"]] <- subseq(N.SEQ, start = 1, end = 1)

  ## make data.table
  N.DT <- as.data.table(N.GR)
  N.DT[["sample"]] <- RANGE.NAME

  ## message
  message(" FUN: calculate stats")

  ## total
  TOTAL.SAMPLE.READS <- sum(N.DT[["MULT"]])

  ## counts
  message("\t calculating counts")
  COUNTS <- N.DT[,sum(.SD), by = c("CLUSTER","seq"), .SDcols = "MULT"]
  COUNT <- dcast.data.table(data = COUNTS, CLUSTER ~ seq, value.var = "V1")
  colnames(COUNT) <- c("CLUSTER",paste(paste("count",ALPHABET, sep = "."),RANGE.NAME,sep = SAMPLE.SEP))
  COUNT[, paste("count.SUM",RANGE.NAME,sep = SAMPLE.SEP) := sum(.SD), by = "CLUSTER"]
  COUNT <- COUNT[!is.na(COUNT[[paste("count.SUM",RANGE.NAME,sep = SAMPLE.SEP)]])]
  COUNT[, paste("total.reads",RANGE.NAME,sep = SAMPLE.SEP) := TOTAL.SAMPLE.READS]

  # per cluster mean
  message("\t calculating means")
  P.CL.MEANS <- N.DT[,lapply(.SD, mean), by = c("CLUSTER"), .SDcols = "MULT"]
  colnames(P.CL.MEANS) <- c("CLUSTER",paste("cluster.mean",RANGE.NAME,sep = "_"))

  # per nucleotide mean
  P.NT.MEANS <- N.DT[,lapply(.SD, mean), by = c("CLUSTER","seq"), .SDcols = "MULT"]
  MEAN <- dcast.data.table(data = P.NT.MEANS, CLUSTER ~ seq, value.var = "MULT")
  colnames(MEAN) <- c("CLUSTER",paste(paste("mean",ALPHABET, sep = "."),RANGE.NAME,sep = SAMPLE.SEP))
  MEAN[, paste("mean.SUM",RANGE.NAME,sep = SAMPLE.SEP) := sum(.SD), by = "CLUSTER"]
  nMEAN <- setDT(join_all(dfs = list(MEAN,P.CL.MEANS),  by = "CLUSTER", type = "full"))
  nMEAN <- nMEAN[!is.na(nMEAN[[paste("mean.SUM",RANGE.NAME,sep = SAMPLE.SEP)]])]
  nMEAN[, paste("mean.AGC",RANGE.NAME,sep = SAMPLE.SEP) := rowMeans(.SD), by = "CLUSTER",.SDcols =
          paste(paste("mean",c("A","C","G"), sep = "."),RANGE.NAME,sep = SAMPLE.SEP)]

  # per cluster median
  message("\t calculating medians")
  P.CL.MEDIANS <- N.DT[,lapply(.SD, median), by = c("CLUSTER"), .SDcols = "MULT"]
  colnames(P.CL.MEDIANS) <- c("CLUSTER",paste("cluster.median",RANGE.NAME,sep = "_"))
```

```
  # per nucleotide median
  P.NT.MEDIANS <- N.DT[,lapply(.SD, median), by = c("CLUSTER","seq"), .SDcols = "MULT"]
  MEDIAN <- dcast.data.table(data = P.NT.MEDIANS, CLUSTER ~ seq, value.var = "MULT")
  colnames(MEDIAN) <- c("CLUSTER",paste(paste("median",ALPHABET, sep = "."),RANGE.NAME,sep = SAMPLE.SEP
  MEDIAN[, paste("median.SUM",RANGE.NAME,sep = SAMPLE.SEP) := sum(.SD), by = "CLUSTER"]
  nMEDIAN <- setDT(join_all(dfs = list(MEDIAN,P.CL.MEDIANS),  by = "CLUSTER", type = "full"))
  nMEDIAN <- nMEDIAN[!is.na(nMEDIAN[[paste("median.SUM",RANGE.NAME,sep = SAMPLE.SEP)]])]
  nMEDIAN[, paste("median.AGC",RANGE.NAME,sep = SAMPLE.SEP) := lapply(.SD,median), by = "CLUSTER",.SDco
          paste(paste("median",c("A","C","G"), sep = "."),RANGE.NAME,sep = SAMPLE.SEP)]

  # join stat tables
  N1J <- setDT(join_all(dfs = list(COUNT,nMEAN,nMEDIAN), by = "CLUSTER", type = "full"))

  ## return
  return(N1J)
}
```

Function: rmskGTF2BED()

```
rmskGTF2BED <- function(## INPUT
                        RMSK.GTF = NULL,

                          ## OUTPUT OPTIONS
                        RETURN.GR = TRUE,
                        KEEP.ALL.CONTIGS = FALSE,

                        ## EXPORT OPTIONS
                        EXPORT.BED = FALSE,

                        ## OTHER
                        MC.CORES = 4,
                        GTF.COLUMN.NAMES = c("chr","database","feature","start","end","score1","strand"
                        ){

  ## AUTHOR: Pavol Genzor
  ## Use: Function to load rmsk gtf annotation
  ## 07.08.21; Verions 6, Refined

  ## LIBRARIES
  suppressPackageStartupMessages({library("data.table");library("stringr");
    library("GenomicRanges") })

  ## INPUT CHECKING
  if(is.null(RMSK.GTF)){stop("Please provide rmsk (TE toolkit) GTF file !")}

  message("Processing ...")
  message(" loading")
  GTF <- data.table::fread(input = RMSK.GTF, nThread = MC.CORES)
  colnames(GTF) <- GTF.COLUMN.NAMES
  ## EXTRACT FROM DESCRIPTION COLLUMN
  DESCRIPTION <- stringr::str_split(GTF[["description"]], pattern = "\"", simplify = TRUE)

  message(" creating bed")
  TE.BED <- data.table(chr = GTF[["chr"]],
                       start = GTF[["start"]],
                       end = GTF[["end"]],
                       name = paste(GTF[["database"]], seq(1,nrow(GTF)), sep = "_"),
                       strand = GTF[["strand"]],
                       gene_id = DESCRIPTION[,2],
                       transcript_id = DESCRIPTION[,4],
                       family_id = DESCRIPTION[,6],
                       class_id = DESCRIPTION[,8])

  if(isFALSE(KEEP.ALL.CONTIGS)){
    # note: not the best way
    message(" removing unused contigs")
    TE.BED <- TE.BED[grep("_",TE.BED[["chr"]],invert = TRUE),] }

  if(isTRUE(EXPORT.BED)){
    FILE.NAME <- data.table::tstrsplit(utils::tail(unlist(tstrsplit(GTF, split = "/")), n=1), split = "
    utils::write.table(x = TE.BED, file = paste(FILE.NAME,".bed", sep = ""),
                       quote = FALSE, sep = "\t", row.names = FALSE, col.names = FALSE)
    message(paste("Saved",paste(FILE.NAME,".bed", sep = ""),"file in the working directory", sep = " ")

  message("Done.")
  if(isTRUE(RETURN.GR)){
    TE.GR <- makeGRangesFromDataFrame(df = TE.BED, keep.extra.columns = TRUE)
    return(TE.GR) }
  else{return(TE.BED) }
}
```

Function: annotateGRanked()

```
annotateGRanked <- function(
                              ## INPUTS
                              GR=NULL,

                              ## ANNOTATIONS
                              CATEGORY.1.GR=NULL,
                              CATEGORY.2.GR=NULL,
                              CATEGORY.3.GR=NULL,
                              CATEGORY.4.GR=NULL,
                              CATEGORY.5.GR=NULL,
                              CATEGORY.NAMES=NULL,

                              ## OPTIONS
                              SAMPLE.NAME=NULL,
                              USE.READS=TRUE,

                              ## FILTER
```

```r
                            NH.TAG = NULL, SIZE.RANGE = c(18,50),

                            ## SETTINGS
                            REPORT.METRIC="FRACTION",
                            OVERLAP.TYPE="within",
                            SOURCE.DIR = NULL,
                            RETURN.ALL=FALSE){

## AUTHOR: Pavol Genzor
## Use: Annotate GR in ranked manner
## 07.08.21; Version 5; Refined

## LIBRARIES
suppressPackageStartupMessages({
  library("data.table"); library("dplyr"); library("GenomicRanges")})

## INPUT CHECKING
if(is.null(GR)) stop("Please provide the GR to annotate !")
CATS <- c("CAT1" = CATEGORY.1.GR,"CAT2" = CATEGORY.2.GR,
          "CAT3" = CATEGORY.3.GR,"CAT4" = CATEGORY.4.GR,"CAT5" = CATEGORY.5.GR)
if(is.null(unlist(CATS))) stop("Please provide at least one annotation category in order !")
if(is.null(CATEGORY.NAMES)) stop("Please provide vector of ordered (!) CATEGORY.NAMES !")
if(is.null(SOURCE.DIR)) stop("Please provide SOURCE.DIR with filter function !")

## FUNCTIONS
source(paste0(SOURCE.DIR,"simpleGRFilter.R"))

## PROGRAM
message("Annotating ...")
message(" order:\n\tCAT1 > CAT2 > CAT3 > CAT4 > CAT5 > OTHER")

## FILTER

FINFO <- paste0("NH",NH.TAG,"SR",min(SIZE.RANGE),max(SIZE.RANGE))
GR <- simpleGRFilter(GR = GR, RANGE.NAME = SAMPLE.NAME,
                     NH.TAG = NH.TAG,
                     SIZE.RANGE = SIZE.RANGE)
message("")

## OVERLAPPING

## INITIATE LIST
FINAL.GRL <- GRangesList()

message(" calculating overlaps")

if(!is.null(CATEGORY.1.GR)){
  message("\twith Category 1")
  IN.CAT1.S <- subsetByOverlaps(x = GR, ranges = CATEGORY.1.GR, type = OVERLAP.TYPE)
  GR <- subsetByOverlaps(x = GR, ranges = IN.CAT1.S, type = OVERLAP.TYPE, invert = TRUE)
  IN.CAT1.AS <- subsetByOverlaps(x = GR, ranges = invertStrand(CATEGORY.1.GR), type = OVERLAP.TYPE)
  GR <- subsetByOverlaps(x = GR, ranges = IN.CAT1.AS, type = OVERLAP.TYPE, invert = TRUE)
```

```
    FINAL.GRL[["CAT1_S"]] <- IN.CAT1.S
    FINAL.GRL[["CAT1_AS"]] <- IN.CAT1.AS
  } else {
    IN.CAT1.S <- GRanges()
    IN.CAT1.AS <- GRanges() }

  if(!is.null(CATEGORY.2.GR)){
    message("\twith Category 2")
    IN.CAT2.S <- subsetByOverlaps(x = GR, ranges = CATEGORY.2.GR, type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT2.S, type = OVERLAP.TYPE, invert = TRUE)
    IN.CAT2.AS <- subsetByOverlaps(x = GR, ranges = invertStrand(CATEGORY.2.GR), type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT2.AS, type = OVERLAP.TYPE, invert = TRUE)

    FINAL.GRL[["CAT2_S"]] <- IN.CAT2.S
    FINAL.GRL[["CAT2_AS"]] <- IN.CAT2.AS
  } else {
    IN.CAT2.S <- GRanges()
    IN.CAT2.AS <- GRanges() }

  if(!is.null(CATEGORY.3.GR)){
    message("\twith Category 3")
    IN.CAT3.S <- subsetByOverlaps(x = GR, ranges = CATEGORY.3.GR, type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT3.S, type = OVERLAP.TYPE, invert = TRUE)
    IN.CAT3.AS <- subsetByOverlaps(x = GR, ranges = invertStrand(CATEGORY.3.GR), type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT3.AS, type = OVERLAP.TYPE, invert = TRUE)

    FINAL.GRL[["CAT3_S"]] <- IN.CAT3.S
    FINAL.GRL[["CAT3_AS"]] <- IN.CAT3.AS
  } else {
    IN.CAT3.S <- GRanges()
    IN.CAT3.AS <- GRanges() }

  if(!is.null(CATEGORY.4.GR)){
    message("\twith Category 4")
    IN.CAT4.S <- subsetByOverlaps(x = GR, ranges = CATEGORY.4.GR, type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT4.S, type = OVERLAP.TYPE, invert = TRUE)
    IN.CAT4.AS <- subsetByOverlaps(x = GR, ranges = invertStrand(CATEGORY.4.GR), type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT4.AS, type = OVERLAP.TYPE, invert = TRUE)

    FINAL.GRL[["CAT4_S"]] <- IN.CAT4.S
    FINAL.GRL[["CAT4_AS"]] <- IN.CAT4.AS
  } else {
    IN.CAT4.S <- GRanges()
    IN.CAT4.AS <- GRanges() }

  if(!is.null(CATEGORY.5.GR)){
    message("\twith Category 5")
    IN.CAT5.S <- subsetByOverlaps(x = GR, ranges = CATEGORY.5.GR, type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT5.S, type = OVERLAP.TYPE, invert = TRUE)
    IN.CAT5.AS <- subsetByOverlaps(x = GR, ranges = invertStrand(CATEGORY.5.GR), type = OVERLAP.TYPE)
    GR <- subsetByOverlaps(x = GR, ranges = IN.CAT5.AS, type = OVERLAP.TYPE, invert = TRUE)

    FINAL.GRL[["CAT5_S"]] <- IN.CAT5.S
```

```r
  FINAL.GRL[["CAT5_AS"]] <- IN.CAT5.AS
} else {
  IN.CAT5.S <- GRanges()
  IN.CAT5.AS <- GRanges() }

message("\tremainder is in OTHER")
GR <- subsetByOverlaps(x = GR, ranges = c(unlist(FINAL.GRL)), type = OVERLAP.TYPE, invert = TRUE)
FINAL.GRL[["OTHER"]] <- GR

## SEQUENCE OR READ TABLE

if(isTRUE(USE.READS)){
  message(" extracting READS info")
  DT <- as.data.table(unlist(lapply(FINAL.GRL, function(i){
    sum(mcols(i)[["MULT"]])})), keep.rownames = TRUE)
  DT[["dataType"]] = "reads"}
else{
  message(" extracting SEQUENCE info")
  DT <- as.data.table(unlist(lapply(FINAL.GRL, function(i){
    length(i)})), keep.rownames = TRUE)
  DT[["dataType"]] = "sequences"}

## PROCESSING THE RESULS

colnames(DT) <- c("ANNOTATION", paste0(SAMPLE.NAME,"__COUNT"),"dataType")
DT[["STRAND"]] <- nth(tstrsplit(DT[["ANNOTATION"]],split="_"),2)
COUNT.COL.NAME <- grep("COUNT",colnames(DT),value = TRUE)
DT[["FRACTION"]] <- lapply(DT[[COUNT.COL.NAME]],function(i){( i/sum(DT[[COUNT.COL.NAME]]))*100 })
colnames(DT) <- gsub("FRACTION",paste0(SAMPLE.NAME,"__FRACTION"), colnames(DT))

##  MELT AND PREPARE DATA

DTM <- suppressWarnings(melt.data.table(DT, id.vars = c("ANNOTATION","STRAND","dataType"), value.name
DTM[["CATS"]] <-  nth(tstrsplit(DTM[["ANNOTATION"]],split="_"),1)
DTM[["MEASURE"]] <- nth(tstrsplit(DTM[["variable"]],split="__"),2)
DTM[["UPDOWN"]] <- ifelse(DTM[["STRAND"]] %in% "S","+","-")
DTM[["UD.VALUE"]] <- as.double(paste0(DTM[["UPDOWN"]],DTM[["VALUE"]]))
DTM[["SAMPLE"]] <- nth(tstrsplit(DTM[["variable"]],split="__"),1)

## ADD COLORS
COL.DT <- data.table(CATS = c("CAT1","CAT2","CAT3","CAT4","CAT5","OTHER"),
                     COLS = c("brown2","skyblue2","goldenrod2","gold","chartreuse4","grey30"))
for(c in DTM[["CATS"]]) set(x = setDT(DTM),
                            i = which(DTM[["CATS"]] %in% c),
                            j = "COLS",
                            value = COL.DT[CATS %in% c][["COLS"]])

## ADD NAMES
CATEGORIES = unique(DTM[["CATS"]])[!unique(DTM[["CATS"]]) %in% "OTHER"]
for(n in 1:length(CATEGORIES)) set(x = DTM,
                                   i = which(DTM[["CATS"]] %in% CATEGORIES[n]),
                                   j = "NAME",
                                   value = CATEGORY.NAMES[n])
  for(i in 1:nrow(DTM)) set(x = DTM, i = which(is.na(DTM[["NAME"]])), j = "NAME","OTHER")

  ## SUBSET BY REPORT.METRIC
  DTM <- DTM[MEASURE %in% REPORT.METRIC]
  DTM[["FILTERS"]] <- FINFO

  ## SPLIT INTO TWO TABLES
  DTM.UP <- DTM[UPDOWN %in% "+"]
  DTM.UP[["CATS"]] <- factor(DTM.UP[["CATS"]], levels = rev(DTM.UP[["CATS"]]))
  DTM.DOWN <- DTM[UPDOWN %in% "-"]
  DTM.DOWN[["CATS"]] <- factor(DTM.DOWN[["CATS"]], levels = rev(DTM.DOWN[["CATS"]]))

  ## RETURN
  RES.L <- list("DTM.UP" = DTM.UP, "DTM.DOWN" = DTM.DOWN, "DTM" = DTM, "COLORS" = COL.DT)
  if(isTRUE(RETURN.ALL)){
    return(RES.L)
  } else { return(RES.L[c("DTM.UP","DTM.DOWN")])}

}
```

Function: annotateRankedBP()

```r
annotateRankedBP <- function(
                              ## INPUT
                              ANN.TAB.L = NULL,

                              ## SETTINGS
                              ASPECT.RATIO = 5,
                              Y.LIMS = c(-100,100),
                              Y.LAB = "Fraction (%)",
                              FAM = "Helvetica", TCOL = "black", XYT = 12,
                              RETURN.ALL = FALSE){

  ## AUTHOR: Pavol Genzor
  ## Use: To plot sense and anti-sense annotations
  ## 07.08.21; Version 2; refined

  ## LIBRARIES
  suppressPackageStartupMessages({
    library("data.table"); library("dplyr"); library("ggplot2")})

  ## INPUT
  if(is.null(ANN.TAB.L)) stop("Please provide ANN.TAB.L !")
  if(!is.list(ANN.TAB.L)) stop("Please provide list of tables !")

  ## Combine samples into directional tables
  DT.UP <- setDT(ANN.TAB.L[["DTM.UP"]])
  DT.UP[["SAMPLE"]] <- factor(DT.UP[["SAMPLE"]], levels = unique(DT.UP[["SAMPLE"]]))

  DT.DOWN <- setDT(ANN.TAB.L[["DTM.DOWN"]])
  DT.DOWN[["SAMPLE"]] <- factor(DT.DOWN[["SAMPLE"]], levels = unique(DT.DOWN[["SAMPLE"]]))
  ## All data table
  DT <- rbindlist(list(DT.UP,DT.DOWN))

  ## Plot
  GGBP <- ggplot() + theme_bw() +
    ## DATA
    geom_bar(data = DT.UP, aes(x = SAMPLE, y = UD.VALUE),
             stat = "identity",
             fill = DT.UP[["COLS"]], colour = NA) +
    geom_bar(data = DT.DOWN, aes(x = SAMPLE, y = UD.VALUE),
             stat = "identity",
             fill = DT.DOWN[["COLS"]], colour = NA) +

    ## LABELS
    geom_text(data = DT.UP, aes(x = SAMPLE, y = UD.VALUE, label = NAME),
              position = position_stack(vjust = 0.5),
              colour=TCOL, family=FAM, size = 2) +
    geom_text(data = DT.DOWN[NAME %in% "OTHER"], aes(x = SAMPLE, y = UD.VALUE, label = NAME),
              position = position_stack(vjust = 0.5),
              colour="white", family=FAM, size = 2) +

    ## LABS
    xlab("") + ylab(paste0(Y.LAB,"\n(-) antisense / (+) sense")) +
    ggtitle(paste0("Annotation Barplot\n",
                   "used: ",unique(DT[["dataType"]]),"\n",
                   "filters: ",unique(DT[["FILTERS"]]))) +

    ## LINE
    geom_hline(yintercept = 0, size = 0.5, colour = "black" ) +

    ## SCALES
    scale_y_continuous(limits = Y.LIMS, breaks = seq(-100,100,10)) +

    ## THEME
    theme(aspect.ratio = ASPECT.RATIO,
          panel.border = element_blank(), panel.grid = element_blank(),
          axis.text.x = element_text(family = FAM, color = TCOL, size = XYT,
                                     angle = 90, vjust = 0.5, hjust = 1),
          axis.text.y = element_text(family = FAM, color = TCOL, size = XYT),
          axis.title = element_text(family = FAM, color = TCOL, size = XYT))


  ## RESULTS
  RES.L <- list("plotData" = DT, "plot" = GGBP)

  ## RETURN
  if(isTRUE(RETURN.ALL)) {
    return(RES.L) } else { return(RES.L[["plot"]]) }
}
```

Finished!