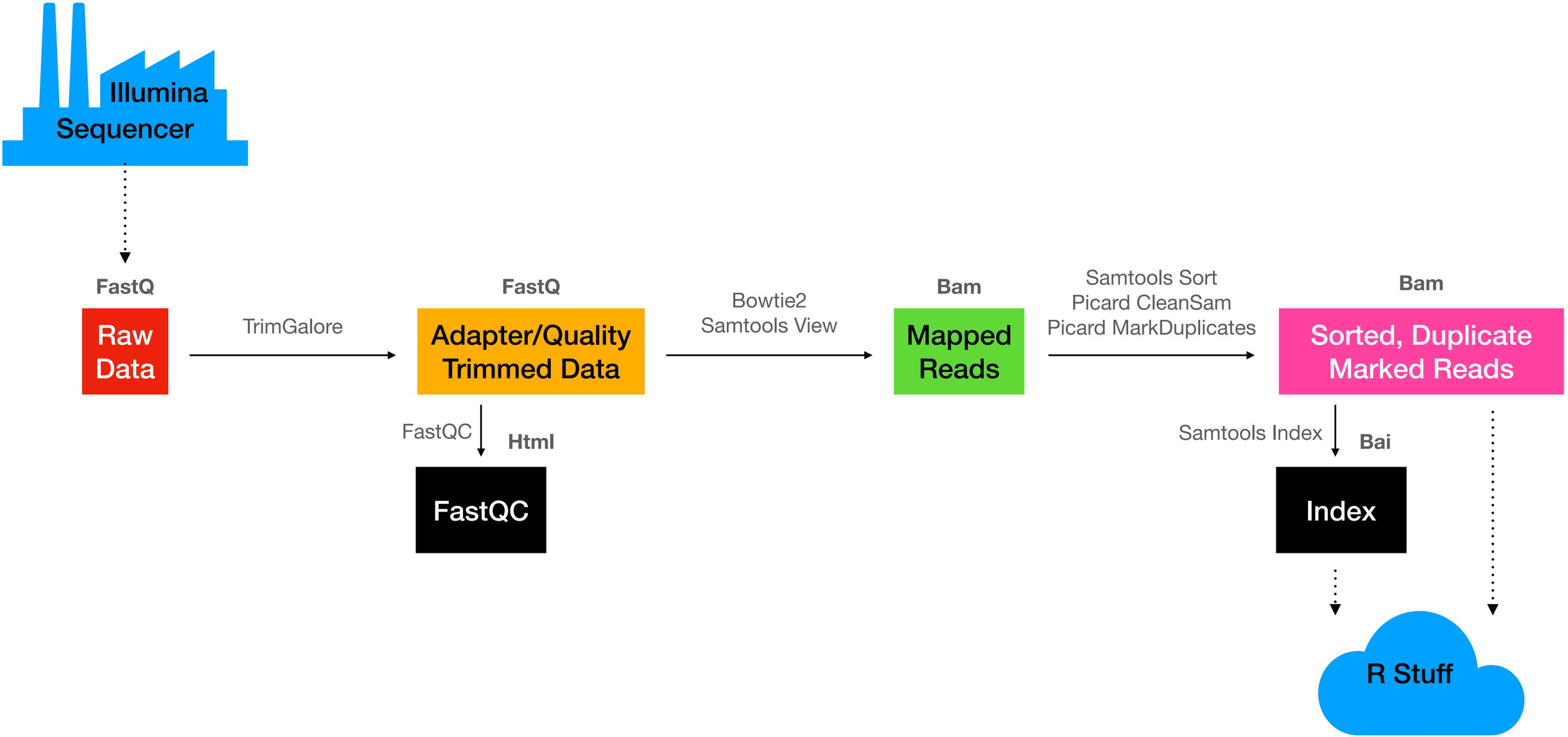
Intro to R for Biologists

IBiS Special Topics, Fall 2021

Class 10: Oct. 18, 2021 R: Genomic Ranges

Erik Andersen and Shelby Blythe

General Workflow



Genomic Ranges Present Unique Challenges

- Chromosome assemblies are like number lines, but there are usually >1 of them.
- The values we are interested in (often ‘mapped reads’) represent small vectors on those number lines.
  - Literally “vector”: there is a directionality (+ strand, - strand). Depending on **strand**, the ‘beginning’ is either at the end or the beginning of the vector, relative to the chromosome number line.
  - Certain ranges are conceptually forbidden (position -1, or beyond the end of the assembly).

The .bed format

- Standard format for minimally describing genomic alignments.
- 6 Columns:
  1. Chromosome
  2. Start
  3. Stop
  4. Name
  5. Score
  6. Strand

chrX	16841873	16841898	CG13000-RA	0	-
chrX	16841898	16841923	CG13000-RA	0	-
chrX	16841923	16841948	CG13000-RA	0	-
chrX	16841948	16841973	CG13000-RA	0	-
chrX	16841973	16841998	CG13000-RA	0	-
chrX	16841998	16842023	CG13000-RA	0	-
chrX	16842023	16842048	CG13000-RA	0	-
chrX	16842048	16842073	CG13000-RA	0	-
chrX	16842073	16842098	CG13000-RA	0	-
chrX	16842098	16842123	CG13000-RA	0	-
chrX	16842123	16842148	CG13000-RA	0	-
chrX	16842148	16842173	CG13000-RA	0	-
chrX	16842173	16842198	CG13000-RA	0	-
chrX	16842198	16842223	CG13000-RA	0	-
chrX	16842223	16842248	CG13000-RA	0	-
chrX	16842248	16842273	CG13000-RA	0	-
chrX	16842273	16842298	CG13000-RA	0	-
chrX	16842298	16842323	CG13000-RA	0	-
chrX	16842323	16842348	CG13000-RA	0	-
chrX	16842348	16842373	CG13000-RA	0	-
chrX	16842373	16842398	CG13000-RA	0	-
chrX	16842398	16842423	CG13000-RA	0	-

Good for representation/recording, poor for performing queries:
- Do any regions overlap?
- Combine overlapping regions
- What is the middle basepair position?
- Widen all ranges to 50 bp relative to the most 5’ position

The `GenomicRanges` Package Can Help

- `GenomicRanges`:
  - Stores .bed-like ranged data
  - Supports additional ‘metadata’ in data.frame format
  - Keeps information about the reference genome on hand to guide operations
  - Has built-in operations that are strand-, feature-, and genome-aware
  - Reduces the overall memory footprint relative to keeping this information as independent data.frames.

```
library(GenomicRanges)
```

Your First GRanges Object:

The GRanges constructor:

- You can manually enter a set of chromosomes (seqnames), start and stop coordinates (IRanges), as well as strand information.
- Here, we've specified the 10th base on "chr2L" on both the minus and plus strand.

```{r}
my.ranges = GRanges(
  seqnames =c('chr2L','chr2L'),
  IRanges(start = c(10, 10), end = c(10, 10)),
  strand = c("+","-")
  )

my.ranges
```

```
GRanges object with 2 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]    chr2L        10      +
  [2]    chr2L        10      -
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Your First GRanges Object:

The `GRanges` constructor:

- The utility of Granges is that it has a number of built-in functions that take into account the particularities of working with genomic data.
- Here we use `findOverlaps` to tell us who overlaps with who. Note: it is strand-aware.

```{r}
findOverlaps(my.ranges)
```

```
SelfHits object with 2 hits and 0 metadata columns:
      queryHits subjectHits
      <integer>   <integer>
  [1]         1           1
  [2]         2           2
  -------
  queryLength: 2 / subjectLength: 2
```

Your First GRanges Object:

The `GRanges` constructor:

- We can use `width` to return the width of each element.
- We can also use `resize` to change the width.
- `resize` can do the job relative to the center, end, or start of each element… Note that "start" and "end" are both strand-aware!

```{r}
width(my.ranges)
```

```
[1] 1 1
```

```{r}
resize(my.ranges, fix = 'start', width = 10)
```

```
GRanges object with 2 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]    chr2L     10-19      +
  [2]    chr2L      1-10      -
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```{r}
resize(my.ranges, fix = 'center', width = 10)
```

```
GRanges object with 2 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]    chr2L      5-14      +
  [2]    chr2L      5-14      -
  -------
  seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Your First GRanges Object:

The GRanges constructor:

- Good old “c” can be used to append more ranges as they become available.

```
more.ranges = c(
  my.ranges,
  resize(GRanges(seqnames = rep('chr2R', 10),
          IRanges(seq(1,50, by = 5)),
          strand = rep('+', 10)
  ), width = 20, fix = 'start')
)

more.ranges

```

```
GRanges object with 12 ranges and 0 metadata columns:
       seqnames    ranges strand
          <Rle> <IRanges>  <Rle>
   [1]    chr2L        10      +
   [2]    chr2L        10      -
   [3]    chr2R      1-20      +
   [4]    chr2R      6-25      +
   [5]    chr2R     11-30      +
   ...      ...       ...    ...
   [8]    chr2R     26-45      +
   [9]    chr2R     31-50      +
  [10]    chr2R     36-55      +
  [11]    chr2R     41-60      +
  [12]    chr2R     46-65      +
  -------
```

Your First GRanges Object:

The GRanges constructor:

- There are also a set of operations that you can use to ‘flatten’ the GRanges object, like reduce, which will merge all overlapping ranges.

- Note, however, that these operations are also strand-aware.

```
GenomicRanges::reduce(more.ranges)
```

```
GRanges object with 3 ranges and 0 metadata columns:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]    chr2L        10      +
  [2]    chr2L        10      -
  [3]    chr2R      1-65      +
  -------
```

Genomic Ranges will also hold “metadata”

- We often want to score and annotate our ranges. The GRanges object will hold metadata in a pseudo-data.frame format.
- It is created and accessed through the `mcols` function.
- **Note: some operations (e.g. `reduce`) will not propagate metadata.**

```
mcols(more.ranges) = data.frame(
  score = runif(12, min = 1, max = 100)
  )

more.ranges
```

```
GRanges object with 12 ranges and 1 metadata column:
        seqnames    ranges strand |     score
           <Rle> <IRanges>  <Rle> | <numeric>
    [1]    chr2L        10      + |   55.9378
    [2]    chr2L        10      - |   18.1306
    [3]    chr2R      1-20      + |   52.4724
    [4]    chr2R      6-25      + |   31.4954
    [5]    chr2R     11-30      + |   57.3156
    ...      ...       ...    ... .       ...
    [8]    chr2R     26-45      + |   82.7391
    [9]    chr2R     31-50      + |   92.2378
   [10]    chr2R     36-55      + |   62.1848
   [11]    chr2R     41-60      + |   57.9362
   [12]    chr2R     46-65      + |   11.6748
   -------
```

Additional features:

- GRanges objects have holders for genome information:
  - seqinfo: names of chromosomes, lengths, “is circular”…
  - seqlevels: the names of the chromosomes, in order
  - seqlengths: the lengths of the chromosomes
  - genome: just a name for your reference (e.g., “dm6”)

Genome Resources

Reference assemblies and transcriptomes

- You can get a genome assembly and transcriptome for your favorite (standard) model system in GRanges format.
- The genome assembly data is good for getting/setting seqinfo metadata in your Granges objects, as well as mining sequences.
- The TxDb object is (as good as your critter's annotation).

```
library(BSgenome.Dmelanogaster.UCSC.dm6)

## BSgenome.Celegans.UCSC.ce11
## BSgenome.Mmusculus.UCSC.mm10
## BSgenome.Hsapiens.UCSC.hg38
## are all available from Bioconductor: install e.g.:
## BiocManager::install("BSgenome.Celegans.UCSC.ce11")

library(TxDb.Dmelanogaster.UCSC.dm6.ensGene)
```

Getting all promoter ranges:

GenomicFeatures

- The GenomicFeatures library has functions for pulling out standard regions of interest.
  - Promoters
  - Genes
  - Exons
  - CDS
  - UTRs

```
library(GenomicFeatures)

prom = GenomicFeatures::promoters(
  TxDb.Dmelanogaster.UCSC.dm6.ensGene
)

prom
```

```
GRanges object with 34920 ranges and 2 metadata columns:
                      seqnames      ranges strand |     tx_id     tx_name
                         <Rle>   <IRanges>  <Rle> | <integer> <character>
  FBtr0475186            chr2L   5529-7728      + |         1 FBtr0475186
  FBtr0309810            chr2L 19952-22151      + |         2 FBtr0309810
  FBtr0347585            chr2L 52817-55016      + |         3 FBtr0347585
  FBtr0345732            chr2L 63999-66198      + |         4 FBtr0345732
  FBtr0345733            chr2L 64318-66517      + |         5 FBtr0345733
           ...              ...         ...    ... .       ...         ...
  FBtr0346886 chrUn_CP007120v1   5113-7312      - |     34916 FBtr0346886
  FBtr0347010 chrUn_DS483646v1  9328-11527      - |     34917 FBtr0347010
  FBtr0347035 chrUn_DS483910v1   -1301-898      + |     34918 FBtr0347035
  FBtr0302352 chrUn_DS484581v1    155-2354      - |     34919 FBtr0302352
  FBtr0347034 chrUn_DS484898v1    541-2740      - |     34920 FBtr0347034
  -------
  seqinfo: 1870 sequences (1 circular) from dm6 genome
```

Getting all Promoter Sequences:

Biostrings

- With a set of ranges and a reference genome, the Biostrings package will allow you to get the actual sequence for each range.

```{r}
library(Biostrings)

# eliminate duplicates... and trim...
prom = trim(prom[!duplicated(prom)])

getSeq(Dmelanogaster, prom)

```

```
DNAStringSet object of length 23223:
         width seq                                                      names
     [1]  2200 TATCTTATATTACCGCAAACACAAA...GCTCAGAGCGGATCTCAATATTTA FBtr0475186
     [2]  2200 TATGCTCTTGTGGTGGTTTGGTTTT...TTTCCCGAAGCGTTGCGCGGGTAA FBtr0309810
     [3]  2200 GTTCCGATGTTTATATTTACTGCGT...CCGTGCCGGCCAACATTTTGTCAC FBtr0347585
     [4]  2200 TAACGTTTTTTTTTTTTTTGCTGTA...CTAAAAATCCAAATTCTGCACACT FBtr0345732
     [5]  2200 AAAATGCACTACGTGCTAAGGTGGC...TAGTTCAAGGAGAGCGCTTCATGC FBtr0345733
     ...   ... ...
 [23219]  2200 GGAGTCGTGCCGTAGGGTACACAGC...TTAGACCTCGGTTTGGTGTCGTCA FBtr0346886
 [23220]  2200 GGGTCTTCCTTCTCAAGATTAGGTA...TATGTGATAAATGGTGCCCATTTA FBtr0347010
 [23221]   898 AGTGATAGCAGACAACTGTATGTGT...TAACAATCCAATTTTTGTAAAGCA FBtr0347035
 [23222]  1716 GCCAGTTTGTTCTAAAATTTGTGAG...TGAGGAGTACGGCATGATTCACGC FBtr0302352
 [23223]   831 TCATAACTCCTCGTTGTATGTTCCG...GAATGAGCTATGCAAACAATCCGA FBtr0347034
```

Export your sequences as .fasta

- The ‘rtracklayer’ library is an import:export utility that will convert GRanges/Biostrings data to common formats:
  - .bed
  - .wig
  - .bigWig
  - .fa
- It will also import (some of) these data as GRanges objects as well…

```{r}
library(rtracklayer)

export(getSeq(Dmelanogaster, prom),
       con = "~/Desktop/fly_promoters_2k_up_0.2k_down.txt",
       format = 'fasta')
```

fly_promoters_2k_up_0.2k_down.txt

```
>FBtr0475186
TATCTTATATTACCGCAAACACAAAATGACAATGCACGACATAGAGAGAAAGAGAGATATTCAGATTGCCTCTCATTGTC
TCACCCATATTATGGGAACCAAATATGAGCACGTATGCGAGAGGAGTGCCAACATATTGTGCTCTACGATTTTTTTGCAA
CCCAAAATGGCGGCGTACGAACGAGATGATAATATATTCAAATTGCCGCTAATCAGAAGCAAGTTTATTGCAATGTTCAG
TGCAGCGCAAAATGGCCGCTCAAGAAAAGGCTCGAATATATATTGCCTGCCTCTCATTCACTCTCTTTTATTACCGCAAG
ACCAAAATGACAATGTACAACAGAGAGAGCAAGAGAGATATTTAGATTGCCTCTCCTTGTCTCTCCCATATTATAGAGAC
CGAAAATGATTGCGTATGCGAGAAGAGTGCCATTGTATTGAGCTCCTCGACCCAAAATAGCGTCGGACGAACGAGATTAT
ATATTTAAAATGCCGATCATTTTCTCATCCATATAAATACTACCGAAAATGACTGTCTAAAGGTACTCATCGACTATATT
TAAATCTGTGTATTTCTGTGAATAGATTGACCTTTGCAATTTTTAACGGCATTGTCTATTAAATTAATATAATTTTCTTT
TTTGATGAATATTTAACCGAACATTTACTTGAAATTAAATTATAAAATTGGTTAAATAATGTTGAAATCTTACTTTCAGC
TAAATGGGGCTATTTTGCAAGGGTTCCATCATGACATTGGTAAATAATTTTTAAAGAATTAATTGTAAGTTCCAATAGAC
TGGAAATTATTTTGCAATATCATTCTTATCCCTATTTCCAAAAGCGAATTATTAGTTGCGTGAAAATCAGAAGGAAAATT
ATTTAACGTGTTATGCCACGCCAAATAGCCGCGCAATAGGAAGCTAGACTATATAATGACTGCAACGAAAATTGTAAATT
CCAATTAAAAGGATATTATTGTGCGATTTCACTTTAATTCTTATTTCAAAAAAGTTAATTATTAGTTGACGGAAATCAGA
ACGAATTTCACCGCAACGTCTTATGCAGCACAAAATGGCGGCGCAAAAGGATGGTTGCATATACAATAACTTCATCTCAT
TCAATCTCTCCTATATTACCGCAAACTCGAAAGCCAAAACACGAATGATGAAGAGGGATAGATTTTATTGGGACAAAAAT
GATAGGTCACGCGAGAGGAGTGGTCTAAATTTTACTCTCACAAAAATGTTGGCAATACAAAATGGCGGCGGAATGAAGAG
GTGAAAATATATTAAAATTGCCGCTCATTTTCTTCGCGGTAGAATTAGGACTGAACGTTGCCGGGTATAGGATCTCTATT
GATGGCCTTTACTTATAAAGTGTATTTCTACAGATCAAATTACTTTTTACTCTTTATCAATATTTAAATATTATAAATTG
ATTTAGTTAAAATACAATTCGAACAATCTTTTCTCCAAATAATAATAATGTTTAATACCTATTTGCGCATATGCGTTTAT
TTTTGGGATTTAATTTTAACATTTTTCAACAAAACCGTTACAAATGTAATTTTAAATCAGGAAACGACTTTGGTATGAAA
ATATGTTTTTTTGTGCGCTTTTAAACATGTAACTGCTCTTTTGTGCTGTTTTATTGAATGCTATCACAGCGTAAAATTTT
AGTTTTAATACCAATACATTGGGAATAATTTGCGATTTCATTCTATTCTTATGCCCAAATAAGGAAATAGTTTCCGGCAA
AAAATCAGAATTTAGCTTTTACAAAAACTAGAGAGGAGAGGACAATATTATAATTGTAGACCGTTTTAAACACTTTAAAA
TGTTTAACCATTTATCAATTATTCTACTAAATGTAGGTGATTTTATTTATTAGAATACGAATTCTTTATCTGAATCGAAC
TAAGTAAGCCTAAGCGCTTAGGAAAAATACATACTTGACGAGTAGAGTGAAATAATTACAAATATTAGACATATCCATTG
CTACTCGCATGTAGAGATTTCCACTTATGTTTTCTCTACTTTCAGCAACCGAGAAGAGAACCCACGTTTGAACAAGTATC
GGCGTGTGGACAACAGCTATCCCCGCTTCATAACGAATGAGGCTGCCGAGGACCTGATTTACAAGAAGTCCATGGGCGAG
CGGGATCAGCCACAGAGCTCAGAGCGGATCTCAATATTTA
>FBtr0309810
TATGCTCTTGTGGTGGTTTGGTTTTGATGAATAAATAAGTATATTAATTGTGGCCGAATTTATTCTAAACTGAAAATAAT
AATAAAAATTAATCAAATTTTCAATAAGTAAAAAATTAAAAAGGAACTTGTATATTTTTTCACTCTTATGAATAAAACCG
AGAATTTAAATTTAGGTAATACTATTAATACCTTTTATAGATACGGACGATACGTTTAATATTATTCTGGTATACTTTAA
TCAAAATTTGTTTAACAAAAAGGGTTTTACATTCTTGAAGTCATAGGAAAATCAGTGGACAGATAGATCTTGGCTCCCAA
GATTTTCCAAACTAGTTGTGCTTATTTTCTTAAAACTTCAAAGATTTGCATGTAGGCAGTTTAATAAACATGTTTGTACA
TATGTATATGTACATTAGGAAAAATGAAATGATGAGCGATCTGTACTCTTTGCAATGTGTATGTATGTACATATGTACCT
CCCTTTAATGAACTAGCAATTCCGTCACGCACACGAAATATTTTTTTATTTGTAAATCTTTTTTAAAGTTTGTTTAAACA
```

What about .bam files?

- The GenomicAlignments package (with help from Rsamtools) will allow you to either:
  - Import your .bam file as a GRanges object. OR…
  - Count features in a .bam file that overlap with a GRanges object that contains regions of interest.
- Slightly different if you have Single- or Paired-end reads.

What about .bam files?

- First, tell R where to look.
- Specify the path to the directory with the .bams
- Use `list.files()` with the `pattern` option to get the relevant filenames.

```{r}
# paired-end
library(GenomicAlignments)
bam.dir = "~/Dropbox/R_for_Biologists_2021/data/testbams/PE_Bam"
bam.files = list.files(bam.dir, pattern = 'mapped.md.bam$')

bam.files
```

```
[1] "PE_Test.mapped.md.bam"
```

What about .bam files?

- Now, we dive into `readGAlignmentPairs`
- This function will import the .bam as not-quite a GRanges object…
- But this is the raw read data…

```{r}
raw = readGAlignmentPairs(paste0(bam.dir,"/", bam.files[1]))

raw
```

```
GAlignmentPairs object with 1939580 pairs, strandMode=1, and 0 metadata columns:
                            seqnames strand   :      ranges  --      ranges
                               <Rle>  <Rle>   :   <IRanges>  --   <IRanges>
        [1]                    chr2L      +   :     104-139  --   1600-1635
        [2]                    chr2L      +   :     304-339  --   2173-2208
        [3]                    chr2L      +   :     375-408  --   1757-1791
        [4]                    chr2L      +   :     399-431  --   1217-1252
        [5]                    chr2L      +   :     476-511  --    993-1028
        ...                      ...    ... ...         ... ...         ...
  [1939576] chrY_CP007108v1_random      +   : 56667-56701  -- 56782-56805
  [1939577] chrY_CP007108v1_random      -   : 56863-56898  -- 56834-56869
  [1939578] chrY_CP007108v1_random      +   : 61520-61553  -- 61641-61674
  [1939579] chrY_CP007108v1_random      +   : 63725-63760  -- 63896-63931
  [1939580] chrY_CP007108v1_random      +   : 65783-65817  -- 65896-65931
  -------
  seqinfo: 1870 sequences from an unspecified genome
```

What about .bam files?

- We can (must) filter the data to remove unwanted stuff. At a minimum:
  - Unmapped reads
  - Improperly paired reads
- To do this, we use an `Rsamtools` helper function, `ScanBamParam`

```{r}
raw = readGAlignmentPairs(
  file = paste0(bam.dir, '/', bam.files[1]),
  param = ScanBamParam(
    flag = scanBamFlag(
      isPaired = TRUE,
      isProperPair = TRUE,
      isUnmappedQuery = FALSE,
      isSecondaryAlignment = FALSE
      ),
    mapqFilter = 10
    )
  )

raw
```

```
GAlignmentPairs object with 1639574 pairs, strandMode=1, and 0 metadata columns:
                          seqnames strand   :      ranges  --      ranges
                             <Rle>  <Rle>   :   <IRanges>  --   <IRanges>
         [1]                 chr2L      +   :   5427-5462  --   5443-5478
         [2]                 chr2L      +   :   5497-5531  --   5708-5742
         [3]                 chr2L      -   :   5578-5613  --   5567-5601
         [4]                 chr2L      +   :   5606-5641  --   5612-5647
         [5]                 chr2L      +   :   5629-5664  --   5632-5667
         ...                   ...    ... ...         ... ...         ...
   [1639570] chrY_CP007114v1_random      -   : 29979-30014  -- 29872-29907
   [1639571] chrY_CP007107v1_random      +   : 29421-29454  -- 29434-29469
   [1639572] chrY_CP007107v1_random      -   : 66868-66903  -- 66794-66827
   [1639573] chrY_CP007108v1_random      -   : 24182-24216  -- 24007-24042
   [1639574] chrY_CP007108v1_random      +   : 26288-26323  -- 26457-26492
   -------
   seqinfo: 1870 sequences from an unspecified genome
```

What about .bam files?

- And now we can make a GRanges object.
  - A little different: the (lowercase) `granges` function will convert the `GAlignmentPairs` object.

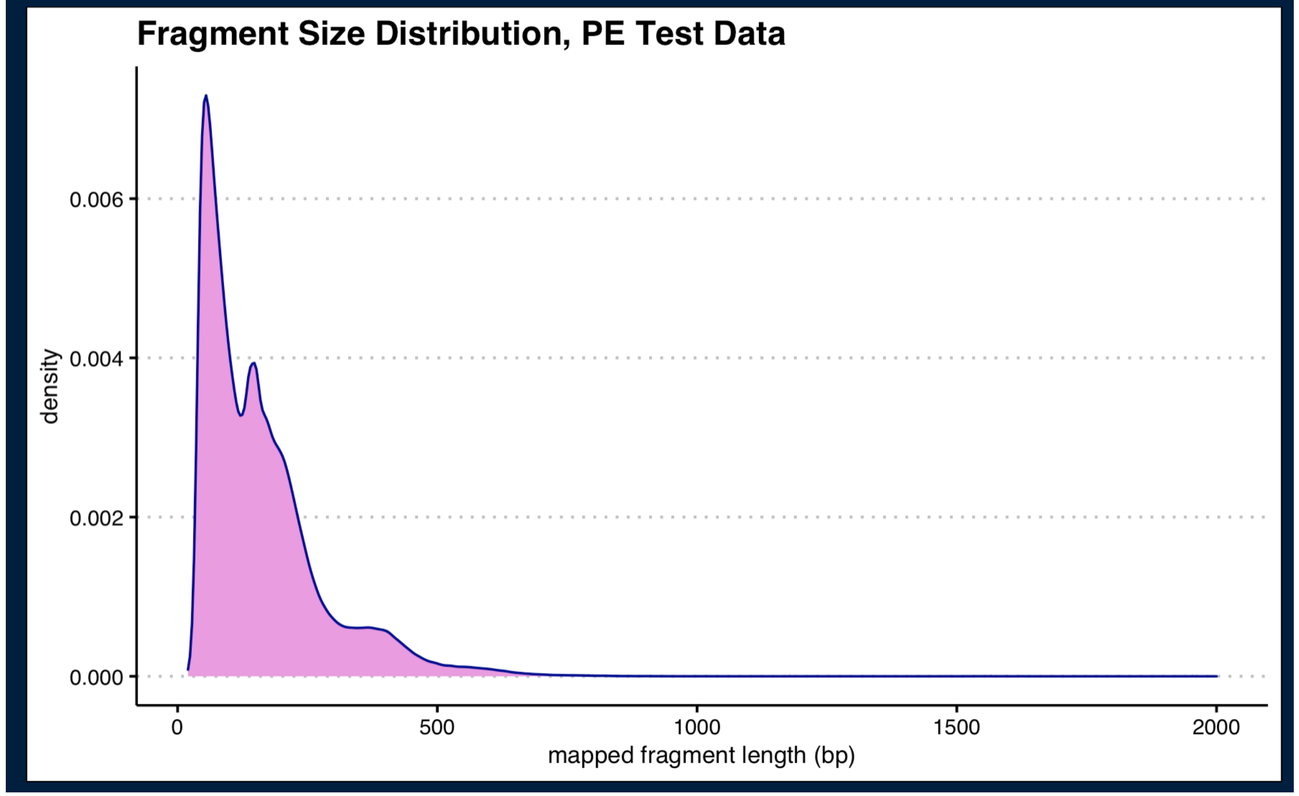
```{r}
gr = granges(raw)

gr
```

```
GRanges object with 1639574 ranges and 0 metadata columns:
                            seqnames      ranges strand
                               <Rle>   <IRanges>  <Rle>
        [1]                    chr2L   5427-5478      +
        [2]                    chr2L   5497-5742      +
        [3]                    chr2L   5567-5613      -
        [4]                    chr2L   5606-5647      +
        [5]                    chr2L   5629-5667      +
        ...                      ...         ...    ...
   [1639570] chrY_CP007114v1_random 29872-30014      -
   [1639571] chrY_CP007107v1_random 29421-29469      +
   [1639572] chrY_CP007107v1_random 66794-66903      -
   [1639573] chrY_CP007108v1_random 24007-24216      -
   [1639574] chrY_CP007108v1_random 26288-26492      +
   -------
   seqinfo: 1870 sequences from an unspecified genome
```

Now, you have analytic superpowers

- Let's use the GRanges `width` function to plot the distribution of fragment sizes from our trimmed, mapped data.

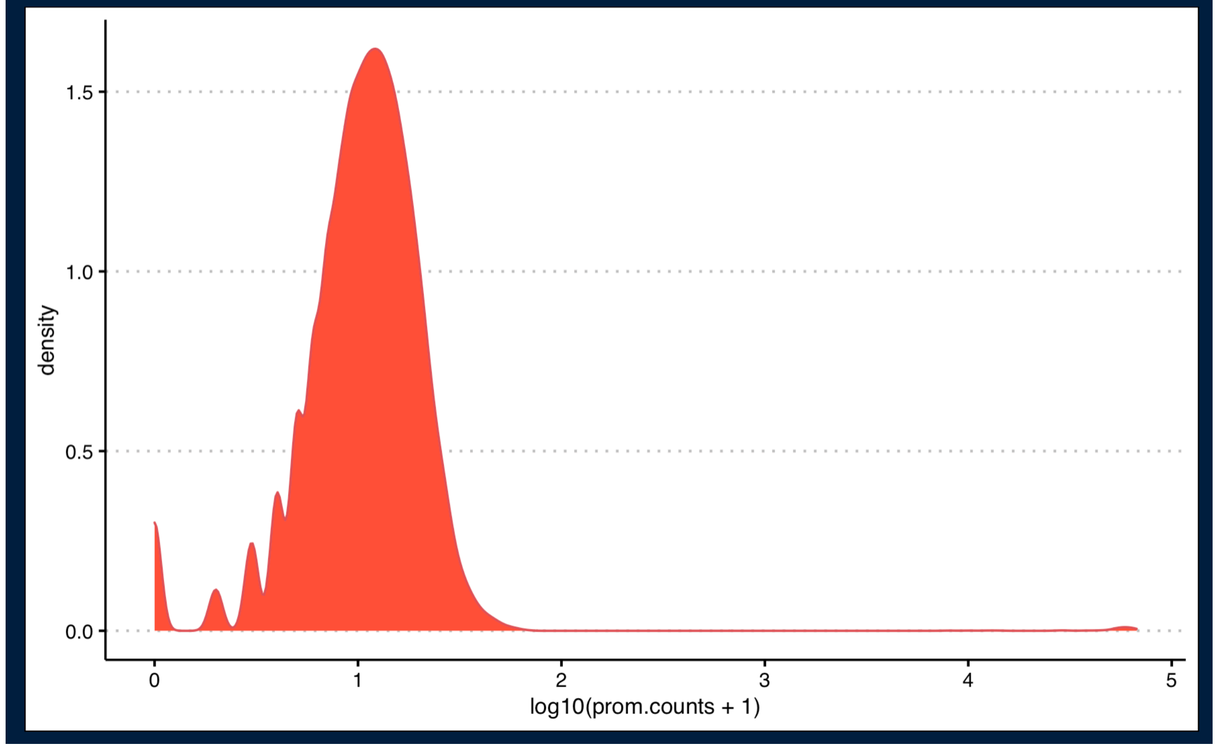
```{r}
ggplot(data.frame(width = width(gr)), aes(x = width)) +
  geom_density(color = 'darkblue', fill = 'plum') +
  theme_clean() +
  labs(title = 'Fragment Size Distribution, PE Test Data',
       x = 'mapped fragment length (bp)')
```



Now, you have analytic superpowers

- Let's use `GenomicRanges::countOverlaps` and `table` to report how many of these fragments overlap with the promoters we previously defined.

```{r}
data.frame(prom.counts = countOverlaps(prom, gr)) %>%
  ggplot(aes(x = log10(prom.counts + 1))) +
  geom_density(col = 'indianred', fill = 'tomato') +
  theme_clean()
```



Next class:

- Generating .wiggle files of average coverage over N-bp bins.
  - Exporting these data to IGV or UCSC genome browser

- Making heat maps, meta gene plots over regions of interest.

Activity:

- A further exploration of Genomic Ranges working up to “import some of your own data”.

- A good resource to develop your expertise in GRanges stuff is the GRanges vignette on Bioconductor, especially the “How-To” document.
- https://bioconductor.org/packages/release/bioc/vignettes/GenomicRanges/inst/doc/GenomicRangesIntroduction.html
- http://bioconductor.org/packages/devel/bioc/vignettes/GenomicRanges/inst/doc/GenomicRangesHOWTOs.pdf