

# Tidy Analysis of Genomic Data

Michael Love  
Dept of Genetics &  
Dept of Biostatistics  
UNC-Chapel Hill

December 2022

# Data organization depends on purpose

Table 1

	Genotype A			Genotype B		
	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3
Drug 1	0.084	0.853	0.096	0.067	0.367	0.392
Drug 2	0.696	0.998	0.182	0.085	0.698	0.791
Drug 3	0.409	0.093	0.495	0.003	0.768	0.689
	Key:	Potential outlier				

## “Tidy data” is organized for programming

One row per observation, one column per variable

```
head(dat)
```

```
## # A tibble: 6 x 5
```

```
##   drug  genotype    rep outlier value
##   <fct> <chr>      <dbl> <lgl>   <dbl>
## 1 1      a          1 FALSE   0.793
## 2 1      a          2 FALSE   0.211
## 3 1      a          3 FALSE   0.205
## 4 2      a          1 FALSE   0.515
## 5 2      a          2 FALSE   0.902
## 6 2      a          3 FALSE   0.844
```

# The pipe

```
command | command | command > output.txt
```

*“Pipes rank alongside the hierarchical file system and regular expressions as one of the most powerful yet elegant features of Unix-like operating systems.”*

<http://www.linfo.org/pipe.html>

In R we use '%>%' instead of '|' to chain operations.

# Verb-based operations

In the R package *dplyr*:

- ▶ `mutate()` adds new variables that are functions of existing variables.
- ▶ `select()` picks variables based on their names.
- ▶ `filter()` picks cases based on their values.
- ▶ `slice()` picks cases based on their position.
- ▶ `summarize()` reduces multiple values down to a single summary.
- ▶ `arrange()` changes the ordering of the rows.
- ▶ `group_by()` perform any operation by group.

<https://dplyr.tidyverse.org/>

## Summarize after grouping

A useful paradigm is to *group* data and then *summarize*:

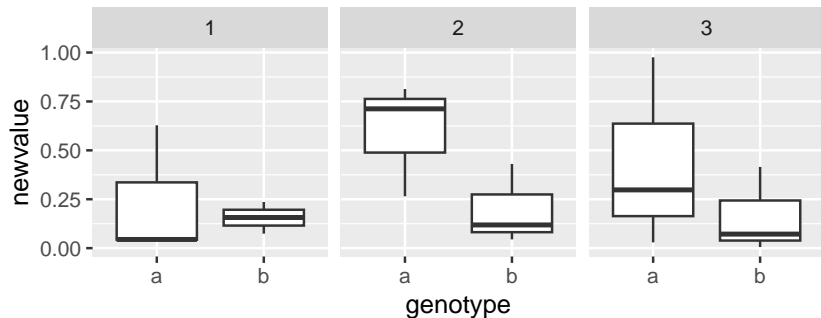
```
dat %>%  
  filter(!outlier) %>%  
  group_by(drug, genotype) %>%  
  summarize(mu_hat = mean(value))
```

## Summarized output

```
## # A tibble: 6 x 3
## # Groups:   drug [3]
##   drug  genotype mu_est
##   <fct> <chr>      <dbl>
## 1 1      a          0.403
## 2 1      b          0.334
## 3 2      a          0.754
## 4 2      b          0.404
## 5 3      a          0.359
## 6 3      b          0.329
```

## Piping directly into plots!

```
dat %>%  
  mutate(newvalue = value^2) %>%  
  ggplot(aes(genotype, newvalue)) +  
  geom_boxplot() +  
  facet_wrap(~drug)
```





# Genomic range data is often already tidy

chr1	100122271	100122495	Peak_75319	65	.	4.24709	6.53
chr1	100148962	100149149	Peak_47035	78	.	5.42118	7.87
chr1	10035625	10035783	Peak_83599	60	.	4.24908	6.01
chr1	10113652	10114012	Peak_22696	102	.	5.88792	10.2
chr1	10165234	10165473	Peak_61426	70	.	4.89948	7.04
chr1	10166426	10166654	Peak_52303	75	.	4.05875	7.56
chr1	10166709	10167142	Peak_101485	56	.	4.29447	5.62
chr1	10228978	10229286	Peak_56552	73	.	4.40606	7.37
chr1	10233774	10233984	Peak_54437	74	.	4.78393	7.43
chr1	10257595	10257832	Peak_144324	43	.	3.23111	4.35
chr1	10300983	10301435	Peak_55477	74	.	4.26907	7.41
chr1	10485619	10485897	Peak_128866	48	.	3.79116	4.85
chr1	10486926	10487197	Peak_64148	68	.	4.92835	6.83
chr1	105184501	105185026	Peak_98454	56	.	4.04794	5.69
chr1	105199317	105199602	Peak_117608	49	.	3.59369	4.96
chr1	105310436	105310779	Peak_23716	100	.	5.55389	10.0
chr1	105312808	105313002	Peak_104599	54	.	3.38229	5.46
chr1	105367824	105367998	Peak_12375	123	.	7.39252	12.3

## Tidy advantages

- ▶ Many already familiar with dplyr and ggplot2
- ▶ Avoid intermediate variables, e.g.:

```
dat3 <- dat2[dat2$signal > x]
```

- ▶ Aim is for *readable* code

## Why “tidy analysis” for genomics?

- ▶ Encourages exploration
- ▶ Encourages efficiency: fewer calls out of R
- ▶ Generalizes from simple to complex cases
- ▶ Developer side: modularity is easier to maintain

## Bringing range data into R

ENCODE mouse embryonic fibroblast, H3K4me1:

```
library(plyranges)
pks <- read_narrowpeaks("ENCFF231UNV.bed.gz")
```

Alternatively:

```
pks <- read.csv("file.csv") %>%
  rename(seqnames = chr) %>%
  as_granges()
```

## Another common paradigm, separating single column

```
pks <- read.delim("file.tsv") %>%  
  separate(location, ":-", # e.g. chr1:123-456  
            into=c("seqnames","start","end")) %>%  
  as_granges()
```

## Bringing range data into R

```
pks %>% slice(1:3) %>% select(signalValue)
```

```
## GRanges object with 3 ranges and 1 metadata column:
```

```
##      seqnames      ranges strand | signalValue
##      <Rle>         <IRanges>  <Rle> |    <numeric>
## [1]      chr1 100122272-100122495      * |      4.24709
## [2]      chr1 100148963-100149149      * |      5.42118
## [3]      chr1  10035626-10035783      * |      4.24908
## -----
##      seqinfo: 22 sequences (1 circular) from mm10 genome
```

## Example use of *plyranges*

- ▶ For a set of query ranges, tiles (here three 1 Mb ranges)
- ▶ Find overlaps between pks and tiles
- ▶ Optional specification of maxgap and/or minoverlap

```
tiles
```

```
## GRanges object with 3 ranges and 1 metadata column:
##           seqnames           ranges strand |   tile_id
##           <Rle>             <IRanges>  <Rle> | <integer>
## [1]      chr1 51000001-52000000      * |         1
## [2]      chr1 52000001-53000000      * |         2
## [3]      chr1 53000001-54000000      * |         3
## -----
## seqinfo: 22 sequences (1 circular) from mm10 genome
```

## Overlaps with join

- ▶ Join-by-overlaps is a flexible paradigm for performing overlaps
- ▶ “Left” vs. “inner” joins: 0-matches are / aren’t included

```
pks %>%  
  select(score) %>% # just `score` column  
  join_overlap_inner(tiles) %>% # overlap -> add cols from tiles  
  group_by(tile_id) %>% # group matches by which tile  
  slice(which.max(score)) # take the top scoring peak
```

```
## GRanges object with 3 ranges and 2 metadata columns:  
## Groups: tile_id [3]  
##           seqnames                ranges strand |           score   tile_id  
##           <Rle>                 <IRanges>  <Rle> | <numeric> <integer>  
## [1]      chr1 51507255-51507557      * |         283         1  
## [2]      chr1 52253831-52254329      * |         177         2  
## [3]      chr1 53757564-53757891      * |         265         3  
## -----  
## seqinfo: 22 sequences (1 circular) from mm10 genome
```



# Counting overlaps

- ▶ Use “.” to specify self within a command
- ▶ Add number of overlaps to each entry in tiles:

```
tiles %>% mutate(n_overlaps = count_overlaps(., pks))
```

```
## GRanges object with 3 ranges and 2 metadata columns:
```

##	seqnames	ranges	strand	tile_id	n_overlaps
##	<Rle>	<IRanges>	<Rle>	<integer>	<integer>
##	[1] chr1	51000001-52000000	*	1	73
##	[2] chr1	52000001-53000000	*	2	36
##	[3] chr1	53000001-54000000	*	3	22

```
## -----
```

```
## seqinfo: 22 sequences (1 circular) from mm10 genome
```

## More complex cases

- ▶ The most common cases are computing summaries, overlaps
- ▶ More complex computations are possible, e.g.:
  - ▶ For peaks near genes, compute correlation of cell-type-specific accessibility and expression (Wancen Mu)
  - ▶ For regulatory variants falling in ATAC peaks, visualize their distribution stratified by SNP and peak categories (Jon Rosen)
  - ▶ For looped and unlooped enhancer-promoter pairs, compare average ATAC and RNA time series, while controlling for genomic distance and contact frequency (Eric Davis)
  - ▶ For DHS in a region of interest with particular genomic characteristics, compare overlap with functional annotation within and in comparison to matched regions from elsewhere in genome (Euphy Wu, Lexi Bounds, Pat Sullivan)

## Going further: extracting info from fitted models

- ▶ Nest → map → unnest
- ▶ Allows model fitting within data groups, see also glance and augment

```
library(broom)
pks %>%
  join_overlap_inner(tiles) %>%
  as_tibble() %>%
  select(tile_id, score, qValue) %>%
  nest(data = -tile_id) %>%
  mutate(fit = map(data, ~lm(score ~ qValue, data=)),
         fitted = map(fit, ~.x$fitted)) %>%
  unnest(c(data, fitted))
```

## Going further: extracting info from fitted models

```
## # A tibble: 131 x 5
##   tile_id score qValue fit      fitted
##   <int> <dbl>  <dbl> <list> <dbl>
## 1       1     92   6.25 <lm>    91.9
## 2       1    135   9.85 <lm>   134.
## 3       1     68   4.22 <lm>    67.9
## 4       1     75   4.84 <lm>    75.2
## 5       1     43   2.23 <lm>    44.4
## 6       1     68   4.22 <lm>    67.9
## 7       1     98   6.77 <lm>   98.0
## 8       1    100   6.90 <lm>   99.5
## 9       1     36   1.70 <lm>    38.1
## 10      1     68   4.22 <lm>    67.9
## # ... with 121 more rows
```

## Some pointers

- ▶ TSS: `anchor_5p() %>% mutate(width=1)`
- ▶ Overlaps can specify `*_directed` or `*_within`
- ▶ Flatten/break up ranges: `reduce_ranges`, `disjoin_ranges`
- ▶ Concatenating ranges: `bind_ranges` with `.id` argument
- ▶ Overlaps are handled often with “joins”: `join_overlap_*`, `join_nearest`, `join_nearest_downstream`, etc.
- ▶ Also `add_neareast_distance`
- ▶ Load *plyranges* last to avoid name masking with *AnnotationDbi* and *dplyr*

## More *plyranges*-based tutorials online

- ▶ *plyranges* vignettes (on Bioc and GitHub)
- ▶ Enrichment of peaks and genes: “Fluent Genomics” workflow
- ▶ *nullranges* vignettes (on Bioc and GitHub)
- ▶ Other examples, incl. bootstrap: “Tidy Ranges Tutorial”
- ▶ BioC2022: Wancen Mu & Eric Davis *nullranges* workshop
- ▶ #tidiness\_in\_bioc and #nullranges Slack channels

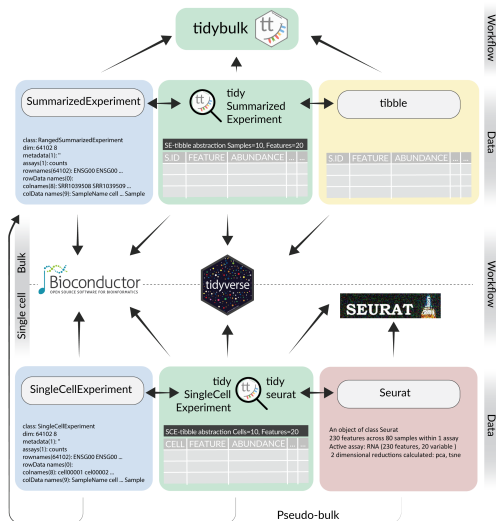
# Summary: tidy analysis for genomic range data



*nullranges* development sponsored by CZI EOSS



# See also: tidy analysis of transcriptomic data



## Tidy Transcriptomics



# Reading

- ▶ Lee, S, Cook, D, Lawrence, M. plyranges: a grammar of genomic data transformation. *Genome Biology* (2019) [10.1186/s13059-018-1597-8](https://doi.org/10.1186/s13059-018-1597-8)
- ▶ Lee S, Lawrence M, Love MI. Fluent genomics with plyranges and tximeta. *F1000Research* (2020) [10.12688/f1000research.22259.1](https://doi.org/10.12688/f1000research.22259.1)
- ▶ plyranges vignettes [sa-lee.github.io/plyranges](https://sa-lee.github.io/plyranges)
- ▶ Tidy Ranges Tutorial [nullranges.github.io/tidy-ranges-tutorial](https://nullranges.github.io/tidy-ranges-tutorial)
- ▶ nullranges: bootRanges, matchRanges [nullranges.github.io/nullranges](https://nullranges.github.io/nullranges)
- ▶ excluderanges [dozmorovlab.github.io/excluderanges](https://dozmorovlab.github.io/excluderanges)

Tidy analysis for matrix data:

- ▶ Mangiola, S, Molania, R, Dong, R et al. tidybulk: an R tidy framework for modular transcriptomic data analysis. *Genome Biology* (2021) [10.1186/s13059-020-02233-7](https://doi.org/10.1186/s13059-020-02233-7)
- ▶ tidySE, tidySCE, tidyseurat [stemangiola.github.io/tidytranscriptomics](https://stemangiola.github.io/tidytranscriptomics)