Using the R Tidyverse packages

Simon Andrews

v2019-04
What's wrong with R?

- Stupid legacy design choices
- Inconsistent API
- Difficult to follow code logical flow
- Multiple ways to model data
Stupid Design Choices

data.frame(
    gene=c("ABC10","DEF10","GHI10"),  
    "gene name"=c("ABC10 gene","DEF10 gene","GHI10 Gene"),  
    value=c(10,12,14),  
    row.names=1
  ) -> some.data

<table>
<thead>
<tr>
<th>gene</th>
<th>&quot;gene name&quot;</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC10</td>
<td>ABC10 gene</td>
<td>10</td>
</tr>
<tr>
<td>DEF10</td>
<td>DEF10 gene</td>
<td>12</td>
</tr>
<tr>
<td>GHI10</td>
<td>GHI10 Gene</td>
<td>14</td>
</tr>
</tbody>
</table>
Stupid Design Choices

class(some.data$name)
[1] "factor"

some.data["ABC1",]
       name value
ABC10  ABC10 gene   10

<table>
<thead>
<tr>
<th>gene.name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC1</td>
<td>ABC10 gene</td>
</tr>
<tr>
<td>DEF1</td>
<td>DEF10 gene</td>
</tr>
<tr>
<td>GHI1</td>
<td>GHI10 Gene</td>
</tr>
</tbody>
</table>
Stupid Design Choices

```r
# Sample data
interesting.samples <- c("sample1","sample2")
small.data[,interesting.samples]

    sample1  sample2
ABC10  -1.9571303  0.2026509
DEF10  1.1210838 -0.2530054
GHI10 -0.5148094 -1.3140919

interesting.samples <- c("sample3")
small.data[,interesting.samples]

[1] -0.4676404  1.3811674  0.1435959
```
Inconsistent API

barplot(1:3, horiz=TRUE)

stripchart(1:3, vertical = FALSE)

min(DATA)
log(DATA, [other options])
substr(DATA, [other options])
grep(pattern, DATA, [other options])
Difficult to follow logical flow

\[
sqrt{\min(\log(\text{abs(some.data)}+1, \text{base}=2)))}
\]

Which function does this belong to?
Tidyverse
https://www.tidyverse.org/

• Collection of R packages
  – Aims to fix many of core R's structural problems
  – Common design and data philosophy
  – Designed to work together, but integrate seamlessly with other parts of R
Tidyverse Packages

- Tibble - data storage
- ReadR - reading data from files
- TidyR - Model data correctly
- DplyR - Manipulate and filter data
- Ggplot2 - Draw figures and graphs
Tidyverse Philosophies

- All data stored in a tibble (like a data frame)

- Make all functions perform a single, simple operation
  - Function names should be descriptive - normally a verb
    - All functions take a tibble as their first argument, allowing 'pipes' to be constructed
    - All (well most) functions return a tibble

- Use functional programming (not OO)
  - Don't modify existing data - create a modified copy
Using Tidyverse
Things to cover today

• Installing Tidyverse

• Reading files and using tibbles

• Restructuring data into ‘tidy’ format

• Transforming tidy data
  – Subsetting and filtering
  – Grouping and Summarising
  – Extending and Joining

• Final exercise
Installation and calling

- `install.packages("tidyverse")`
- `library("tidyverse")`

-- Attaching packages --------- tidyverse 1.2.1 --
`ggplot2` 3.1.0  `purrr` 0.2.5
`tibble` 2.0.1  `dplyr` 0.7.8
`tidyr` 0.8.2  `stringr` 1.3.1
`readr` 1.3.1  `forcats` 0.3.0
Reading Files with readr

• Provides functions which mirror R's read.table
  – read_csv("file.csv")
  – read_tsv("file.tsv")
  – read_delim("file.txt")
  – read_fwf("file.txt", col_positions=c(1,3,6))
Reading files with readr

• Output is always a tibble

• No name translation

• Guesses appropriate formats
  – Says what formats were guessed (can get it wrong though!)
  – No string to factor conversion
## Reading files with readr

```r
> read_tsv("trumpton.txt") -> trumpton

Parsed with column specification:

```r
cols(
    LastName = col_character(),
    FirstName = col_character(),
    Age = col_double(),
    Weight = col_double(),
    Height = col_double()
)
```

```r
> trumpton

# A tibble: 7 x 5

## A tibble: 7 x 5
LastName  FirstName Age  Weight Height
<chr>     <chr>    <dbl> <dbl>  <dbl>
1 Hugh     Chris    26    90    175
2 Pew      Adam     32   102    183
3 Barney   Daniel   18    88    168
4 McGrew   Chris    48    97    155
5 Cuthbert Carl    28    91    188
6 Dibble   Liam     35    94    145
7 Grub     Doug     31    89    164
```
Fixing guessed columns

> read_csv("Child_Variants.csv") -> child_tbl

Parsed with column specification:

```r
cols(
  CHR = col_double(),
  POS = col_double(),
  dbSNP = col_character(),
  REF = col_character(),
  ALT = col_character(),
  QUAL = col_double(),
  GENE = col_character(),
  ENST = col_character(),
  MutantReads = col_double(),
  COVERAGE = col_double(),
  MutantReadPercent = col_double()
)
```

Warning: 477 parsing failures.

<table>
<thead>
<tr>
<th>row</th>
<th>col</th>
<th>expected</th>
<th>actual</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>25346</td>
<td>CHR</td>
<td>a double</td>
<td>MT</td>
<td>'Child_Variants.csv'</td>
</tr>
<tr>
<td>25347</td>
<td>CHR</td>
<td>a double</td>
<td>MT</td>
<td>'Child_Variants.csv'</td>
</tr>
<tr>
<td>25348</td>
<td>CHR</td>
<td>a double</td>
<td>MT</td>
<td>'Child_Variants.csv'</td>
</tr>
<tr>
<td>25349</td>
<td>CHR</td>
<td>a double</td>
<td>MT</td>
<td>'Child_Variants.csv'</td>
</tr>
<tr>
<td>25350</td>
<td>CHR</td>
<td>a double</td>
<td>MT</td>
<td>'Child_Variants.csv'</td>
</tr>
<tr>
<td>.....</td>
<td>.......</td>
<td>........</td>
<td>......</td>
<td>........................</td>
</tr>
</tbody>
</table>

See problems(...) for more details.

• Types are guessed on first 1000 lines

• Warnings for later mismatches

• Values converted to NA
Fixing guessed columns

```r
> read_csv(
  "Child_Variants.csv",
  col_types=cols(CHR=col_character())
) -> child_tbl
```

# A tibble: 25,822 x 11

<table>
<thead>
<tr>
<th></th>
<th>CHR</th>
<th>POS</th>
<th>dbSNP</th>
<th>REF</th>
<th>ALT</th>
<th>QUAL</th>
<th>GENE</th>
<th>ENST</th>
<th>MutantReads</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>69270</td>
<td>.</td>
<td>A</td>
<td>G</td>
<td>16</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>69511</td>
<td>rs75~</td>
<td>A</td>
<td>G</td>
<td>200</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>69761</td>
<td>.</td>
<td>A</td>
<td>T</td>
<td>200</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>69897</td>
<td>rs75~</td>
<td>T</td>
<td>C</td>
<td>59</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>877831</td>
<td>rs66~</td>
<td>T</td>
<td>C</td>
<td>200</td>
<td>SAMD~</td>
<td>ENST~</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>881627</td>
<td>rs22~</td>
<td>G</td>
<td>A</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>52</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>887801</td>
<td>rs38~</td>
<td>A</td>
<td>G</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>47</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>888639</td>
<td>rs37~</td>
<td>T</td>
<td>C</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>888659</td>
<td>rs37~</td>
<td>T</td>
<td>C</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>17</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>889158</td>
<td>rs13~</td>
<td>G</td>
<td>C</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>25</td>
<td>28</td>
</tr>
</tbody>
</table>
```

# ... with 25,812 more rows, and 1 more variable: MutantReadPercent <dbl>
Tibbles vs Data Frames

• Most operations are compatible
  – my.tibble$column
  – my.tibble[[3]]
  – my.tibble[1:10,2:3]
  – Any function taking data frame as argument
Tibbles vs Data Frames

• Tibbles print more nicely
  – Tell you the column types
  – Tell you the overall dimensions
  – Omit columns which don't fit
  – Truncate values which don’t fit
  – Show fewer rows by default (10 vs 1000)

• Tibbles don't support row names
  – Causes issues with some packages, especially BioConductor
# A tibble: 25,822 x 11

## Printing tibbles

- Tells you the column types
- Tells you the overall dimensions
- Omits columns which don't fit
- Truncates values which don’t fit
- Shows fewer (10 vs 1000) rows by default

<table>
<thead>
<tr>
<th>CHR</th>
<th>POS</th>
<th>dbsNP</th>
<th>REF</th>
<th>ALT</th>
<th>QUAL</th>
<th>GENE</th>
<th>ENST</th>
<th>MutantReads</th>
<th>COVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>69270</td>
<td>.</td>
<td>A</td>
<td>16</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>69511</td>
<td>rs75~</td>
<td>A</td>
<td>G</td>
<td>200</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>69761</td>
<td>.</td>
<td>A</td>
<td>T</td>
<td>200</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>69897</td>
<td>rs75~</td>
<td>T</td>
<td>C</td>
<td>59</td>
<td>OR4F5</td>
<td>ENST~</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>877831</td>
<td>rs66~</td>
<td>T</td>
<td>C</td>
<td>200</td>
<td>SAMD~</td>
<td>ENST~</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>881627</td>
<td>rs22~</td>
<td>G</td>
<td>A</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>52</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>887801</td>
<td>rs38~</td>
<td>A</td>
<td>G</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>47</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>888639</td>
<td>rs37~</td>
<td>T</td>
<td>C</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>888659</td>
<td>rs37~</td>
<td>T</td>
<td>C</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>889158</td>
<td>rs13~</td>
<td>G</td>
<td>C</td>
<td>200</td>
<td>NOC2L</td>
<td>ENST~</td>
<td>25</td>
</tr>
</tbody>
</table>

# ... with 25,812 more rows, and 1 more variable: MutantReadPercent <dbl>
Creating tibbles

- Any read method from `readr`
- Using `as_tibble()` on a data frame
- Using `tibble()` from vectors
Exercise 1
Reading Data into Tibbles
"Tidy Data"

- Many ways to store the same data
- Different structures need different code to process them
- Good to have a common design structure

```r
# A tibble: 3 x 8
  Gene  Chr Start  End Sample1_WT Sample2_WT Sample3_KO Sample4_KO
  <chr> <dbl>   <dbl> <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
1 Gnai3   2    163898 167465      9.39       10.9       33.5       81.9
2 Pbsn    5   4888573 4891351     91.7      59.6       45.3       82.3
3 Cdc45   7  1250084 1262669     69.2      36.1       54.4       38.1
```
Wide vs Long

• Wide format
  – Rows are measures
  – Samples are columns
  – Implied (often incorrectly) linkage between measures on the same row for different samples
  – Works OK when measures are paired and equal
  – Tricky to alter as you need to select columns first
Wide vs Long

• Long format is the standard "Tidy" format
  – Every row is a single observation
  – Every column should be a different type of annotation or measure
  – Should never have the same measure for difference samples in multiple columns

  – Slightly more complex for paired studies
  – More flexible and consistent generally.
Converting to "Tidy" format

• Put all measures into a single column
• Add a 'sample' and 'genotype' column
• Duplicate the gene information as required
  – Or separate it into a different table

<table>
<thead>
<tr>
<th>Gene</th>
<th>Chr</th>
<th>Start</th>
<th>End</th>
<th>Sample1_WT</th>
<th>Sample2_WT</th>
<th>Sample3_KO</th>
<th>Sample4_KO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnai3</td>
<td>2</td>
<td>163898</td>
<td>167465</td>
<td>9.39</td>
<td>10.9</td>
<td>33.5</td>
<td>81.9</td>
</tr>
<tr>
<td>Pbsn</td>
<td>5</td>
<td>4888573</td>
<td>4891351</td>
<td>91.7</td>
<td>59.6</td>
<td>45.3</td>
<td>82.3</td>
</tr>
<tr>
<td>Cdc45</td>
<td>7</td>
<td>1250084</td>
<td>1262669</td>
<td>69.2</td>
<td>36.1</td>
<td>54.4</td>
<td>38.1</td>
</tr>
</tbody>
</table>
Converting to "Tidy" format

```r
# A tibble: 3 x 8
   Gene Chr Start    End Sample1_WT Sample2_WT Sample3_KO Sample4_KO
  <chr> <dbl> <dbl>   <dbl>       <dbl>       <dbl>       <dbl>       <dbl>
1  Gnai3   2  163898  167465       9.39       10.9       33.5       81.9
2  Pbsn   5 4888573 4891351      91.7        59.6       45.3       82.3
3  Cdc45   7 1250084 1262669     69.2        36.1       54.4       38.1
```

non.normalised %>%
  gather(sample, value, -Gene,-Chr,-Start,-End) %>%
  separate(sample,into=c("sample","genotype"),sep="_")

Converting to "Tidy" format

```r
# A tibble: 12 x 7

  Gene  Chr Start  End     sample genotype value
  <chr> <dbl> <dbl> <dbl> <chr>   <chr>    <dbl>
1 Gnai3   2  163898 167465 Sample1  WT      9.39
2 Pbsn    5  4888573 4891351 Sample1  WT     91.7
3 Cdc45   7  1250084 1262669 Sample1  WT     69.2
4 Gnai3   2  163898 167465 Sample2  WT     10.9
5 Pbsn    5  4888573 4891351 Sample2  WT     59.6
6 Cdc45   7  1250084 1262669 Sample2  WT     36.1
7 Gnai3   2  163898 167465 Sample3  KO     33.5
8 Pbsn    5  4888573 4891351 Sample3  KO     45.3
9 Cdc45   7  1250084 1262669 Sample3  KO     54.4
10 Gnai3  2  163898 167465 Sample4  KO     81.9
11 Pbsn   5  4888573 4891351 Sample4  KO     82.3
12 Cdc45  7  1250084 1262669 Sample4  KO     38.1
```
Converting to "Tidy" format

- Can clean up duplicated information

```r
# A tibble: 12 x 4

Gene sample genotype value
<chr> <chr> <chr> <dbl>
1 Gnai3 Sample1 WT 9.39
2 Pbsn Sample1 WT 91.7
3 Cdc45 Sample1 WT 69.2
4 Gnai3 Sample2 WT 10.9
5 Pbsn Sample2 WT 59.6
6 Cdc45 Sample2 WT 36.1
7 Gnai3 Sample3 KO 33.5
8 Pbsn Sample3 KO 45.3
9 Cdc45 Sample3 KO 54.4
10 Gnai3 Sample4 KO 81.9
11 Pbsn Sample4 KO 82.3
12 Cdc45 Sample4 KO 38.1

# A tibble: 3 x 4

Gene Chr Start     End
<chr> <dbl> <dbl> <dbl>
1 Gnai3 2 163898 167465
2 Pbsn 5 4888573 4891351
3 Cdc45 7 1250084 1262669
```

- These can be recombined later on as needed.

Babraham Bioinformatics
Tidying operations

• gather
  – Takes multiple columns of the same type and puts them into a pair of key-value columns

• spread
  – Takes a key-value column pair and spreads them out to multiple columns of the same type

• separate
  – Splits a delimited column into multiple columns

• unite
  – Combines multiple columns into one
Using `gather` function:

```r
gather(data, key_name, value_name, columns)
```

```r
gather(gather.data, sample, value, -genes)
```

---

**Example code:**

```r
# A tibble: 12 x 3
 genes sample  value
 <chr> <chr>   <dbl>
1 Nanog sample1 0.615
2 Sfi1 sample1 0.831
3 GAPDH sample1 0.676
4 Nanog sample2 0.446
5 Sfi1 sample2 0.809
6 GAPDH sample2 0.888
7 Nanog sample3 0.274
8 Sfi1 sample3 0.401
9 GAPDH sample3 0.435
10 Nanog sample4 0.409
11 Sfi1 sample4 0.446
12 GAPDH sample4 0.348
```

---

**Warning:**

Column selections can be positive (sample1, sample2, sample3) or negative (-genes).
Using `spread`:

```
spread(data,key_col,value_col)
```

```
spread(spread.data,dimension,measures)
```
Using `separate`:

```r
separate(data, col, into_vector, sep)
```

```r
separate(sep.data, size, into=c("h", "w"), sep="x", convert = TRUE)
```

---

```r
> sep.data

# A tibble: 4 x 2
#  standard size
#   <chr>  <chr>
# 1  SVGA  800x600
# 2  XGA  1024x768
# 3 WXGA 1280x720
# 4  FHD 1920x1080

# A tibble: 4 x 3
#  standard    h    w
#   <chr>    <int> <int>
# 1  SVGA      800   600
# 2  XGA     1024    768
# 3 WXGA    1280    720
# 4  FHD    1920   1080
```

---

`convert=TRUE` means that the type for the new columns will be re-guessed since it might well change.
Using `unite`

`unite(data, cols, colname, sep)`

`unite(unite.data, chr, start, strand, col="locus", sep=":":)`

```r
> unite.data

# A tibble: 3 x 4
  chr   start strand value
  <chr> <dbl> <chr>  <dbl>
1 1     12876 +   15.0
2 2     1298712 +  5.60
3 X     1291872 -  72.1

# A tibble: 3 x 2
locus   value
<chr>   <dbl>
1 1:12876:+  15.0
2 2:1298712:+  5.60
3 x:1291872:-  72.1
```
Multiple operations

# A tibble: 3 x 8

<table>
<thead>
<tr>
<th>Gene</th>
<th>Chr</th>
<th>Start</th>
<th>End</th>
<th>Sample1_WT</th>
<th>Sample2_WT</th>
<th>Sample3_KO</th>
<th>Sample4_KO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gnai3</td>
<td>2</td>
<td>163898</td>
<td>167465</td>
<td>9.39</td>
<td>10.9</td>
<td>33.5</td>
<td>81.9</td>
</tr>
<tr>
<td>Pbsn</td>
<td>5</td>
<td>4888573</td>
<td>4891351</td>
<td>91.7</td>
<td>59.6</td>
<td>45.3</td>
<td>82.3</td>
</tr>
<tr>
<td>Cdc45</td>
<td>7</td>
<td>1250084</td>
<td>1262669</td>
<td>69.2</td>
<td>36.1</td>
<td>54.4</td>
<td>38.1</td>
</tr>
</tbody>
</table>

- gather the sample columns into sample and value
- Split the sample column into sample and genotype with separate
Multiple operations

gather(non.norm,sample,value,-Gene,-Chr,-Start,-End) -> n1
separate(n1, sample, into=c("sample","geno"),sep="_") -> n2

separate(
gather(
  non.normalised,
sample,
value,
-Gene,-Chr,-Start,-End
),
sample,
into=c("sample","genotype"),
sep="_
) -> norm2
The %>% operator (pipe)

• All* tidyverse functions take a tibble as the first argument and return a modified tibble

• Often need to 'chain' several functions together.

• The %>% operator takes the argument on its left and makes it the first argument to the function on its right
  – Correct logical flow to the code
  – Easier to read and interpret
  – Cleaner (fewer intermediate variables, less typing)

*Well nearly all, anyway.
The %>% operator (pipe)

data %>% head()

head(data)

tibble(a=1:10, b=21:30) %>% nrow()
Multiple operations

# A tibble: 3 x 8

## non.norm %>%

  non.norm %>%
  gather(sample, value, -Gene, -Chr, -Start, -End) %>%
  separate(sample, into = c("sample", "geno"), sep="_")
Exercise 2
Restructuring data into ‘tidy’ format
Transforming data with dplyr
Functions in dplyr

• Replacement for core selecting, sorting and filtering

• Operations can be split into groups
  – Subsetting or filtering a tibble
  – Grouping and summarising variables
  – Extending tibbles by adding data
Subsetting and Filtering

• **slice** select rows by position

• **select** select columns by name/position

• **arrange** sort rows

• **distinct** deduplication

• **filter** functional selection of rows
Trumpton

# A tibble: 7 x 5

  LastName FirstName Age Weight Height
  <chr>    <chr>  <dbl>  <dbl>  <dbl>
1 Hugh     Chris   26     90    175
2 Pew      Adam    32    102    183
3 Barney   Daniel  18     88    168
4 McGrew   Chris   48     97    155
5 Cuthbert Carl   28     91    188
6 Dibble   Liam    35     94    145
7 Grub     Doug    31     89    164
Using `slice` function:

```
slice(data, rows)
```

Example:
```
trumpton %>% slice(1, 4, 7)
```

# A tibble: 3 x 5
## LastName  FirstName  Age  Weight  Height
## <chr>      <chr>   <dbl>   <dbl>    <dbl>
1 Hugh       Chris    26       90       175
2 McGrew     Chris    48       97       155
3 Grub       Doug     31       89       164

Note, rows can be separate (as above) or combined into one vector.
Using `select`

`select(data, cols)`

`trumpton %>% select(LastName, Age, Height)`

# A tibble: 7 x 3

<table>
<thead>
<tr>
<th>LastName</th>
<th>Age</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hugh</td>
<td>26</td>
<td>175</td>
</tr>
<tr>
<td>Pew</td>
<td>32</td>
<td>183</td>
</tr>
<tr>
<td>Barney</td>
<td>18</td>
<td>168</td>
</tr>
<tr>
<td>McGrew</td>
<td>48</td>
<td>155</td>
</tr>
<tr>
<td>Cuthbert</td>
<td>28</td>
<td>188</td>
</tr>
<tr>
<td>Dibble</td>
<td>35</td>
<td>145</td>
</tr>
<tr>
<td>Grub</td>
<td>31</td>
<td>164</td>
</tr>
</tbody>
</table>
Using `select`

```
select(data, -unwanted_cols)
```

```
trumpton %>% select(-LastName)
```

```r
# A tibble: 7 x 4

<table>
<thead>
<tr>
<th></th>
<th>FirstName</th>
<th>Age</th>
<th>Weight</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chris</td>
<td>26</td>
<td>90</td>
<td>175</td>
</tr>
<tr>
<td>2</td>
<td>Adam</td>
<td>32</td>
<td>102</td>
<td>183</td>
</tr>
<tr>
<td>3</td>
<td>Daniel</td>
<td>18</td>
<td>88</td>
<td>168</td>
</tr>
<tr>
<td>4</td>
<td>Chris</td>
<td>48</td>
<td>97</td>
<td>155</td>
</tr>
<tr>
<td>5</td>
<td>Carl</td>
<td>28</td>
<td>91</td>
<td>188</td>
</tr>
<tr>
<td>6</td>
<td>Liam</td>
<td>35</td>
<td>94</td>
<td>145</td>
</tr>
<tr>
<td>7</td>
<td>Doug</td>
<td>31</td>
<td>89</td>
<td>164</td>
</tr>
</tbody>
</table>
```
Defining Selected Columns

• Common rules used throughout tidyverse.

• Single definitions (name, position or function)
  Positive  weight, height, length, 1, 2, 3, last_col()
  Negative  -chromosome, -start, -end, -1, -2, -3

• Range selections
  3:5  -(3:5)  height:length  -(height:length)

• Functional selections (positive or negative)
  starts_with()  -starts_with()
  ends_with()  -ends_with()
  contains()  -contains()
  matches()  -matches
Using `select` helpers

```r
> child.variants
CHR POS dbSNP REF ALT QUAL GENE ENST MutantReads COVERAGE MutantReadPercent

> child.variants %>% select(REF, COVERAGE)
  REF  COVERAGE

> child.variants %>% select(-CHR, -ENST)
  POS dbSNP  REF  ALT  QUAL  GENE  MutantReads COVERAGE MutantReadPercent

> child.variants %>% select(5:last_col())
  ALT  QUAL  GENE  ENST  MutantReads COVERAGE MutantReadPercent

> child.variants %>% select(POS:GENE)
  POS dbSNP  REF  ALT  QUAL  GENE

> child.variants %>% select(-(POS:GENE))
  CHR  ENST  MutantReads COVERAGE MutantReadPercent

> child.variants %>% select(starts_with("Mut"))
  MutantReads MutantReadPercent

> child.variants %>% select(-ends_with("t", ignore.case = TRUE))
  CHR  POS dbSNP  REF  QUAL  GENE  MutantReads COVERAGE

> child.variants %>% select(contains("Read"))
  MutantReads MutantReadPercent
```
Using `arrange`:

`arrange(data, cols)`

`trumpton %>% arrange(Age)`

# A tibble: 7 x 5

<table>
<thead>
<tr>
<th>LastName</th>
<th>FirstName</th>
<th>Age</th>
<th>Weight</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barney</td>
<td>Daniel</td>
<td>18</td>
<td>88</td>
<td>168</td>
</tr>
<tr>
<td>Hugh</td>
<td>Chris</td>
<td>26</td>
<td>90</td>
<td>175</td>
</tr>
<tr>
<td>Cuthbert</td>
<td>Carl</td>
<td>28</td>
<td>91</td>
<td>188</td>
</tr>
<tr>
<td>Grub</td>
<td>Doug</td>
<td>31</td>
<td>89</td>
<td>164</td>
</tr>
<tr>
<td>Pew</td>
<td>Adam</td>
<td>32</td>
<td>102</td>
<td>183</td>
</tr>
<tr>
<td>Dibble</td>
<td>Liam</td>
<td>35</td>
<td>94</td>
<td>145</td>
</tr>
<tr>
<td>McGrew</td>
<td>Chris</td>
<td>48</td>
<td>97</td>
<td>155</td>
</tr>
</tbody>
</table>
Using arrange

arrange(data, cols)

trumpton %>% arrange(desc(Age))

# A tibble: 7 x 5
##   LastName FirstName Age  Weight Height
##    <chr>     <chr> <dbl> <dbl>   <dbl>
## 1  McGrew    Chris  48    97     155
## 2   Dibble   Liam  35    94     145
## 3    Pew     Adam  32   102     183
## 4    Grub    Doug  31    89     164
## 5  Cuthbert  Carl  28    91     188
## 6    Hugh    Chris 26    90     175
## 7  Barney   Daniel 18    88     168
Using `distinct` 

```
distinct(data, cols, .keep_all=TRUE)
```

trumpton %>% distinct(FirstName, .keep_all=TRUE)

# A tibble: 6 x 5
## LastName   FirstName  Age  Weight Height
## <chr>       <chr>     <dbl> <dbl> <dbl>
## 1 Hugh       Chris     26    90   175
## 2 Pew        Adam      32   102   183
## 3 Barney     Daniel    18    88   168
## 4 Cuthbert   Carl      28    91   188
## 5 Dibble     Liam      35    94   145
## 6 Grub       Doug      31    89   164

Note, “keep_all” has a dot before it
Using `filter` function:

```r
filter(data, conditions)
```

```r
trumpton %>% filter(Height > 170 & Age < 30)
```

# A tibble: 6 x 5

<table>
<thead>
<tr>
<th>LastName</th>
<th>FirstName</th>
<th>Age</th>
<th>Weight</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;chr&gt;</td>
<td>&lt;chr&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
</tr>
<tr>
<td>1 Hugh</td>
<td>Chris</td>
<td>26</td>
<td>90</td>
<td>175</td>
</tr>
<tr>
<td>2 Cuthbert</td>
<td>Carl</td>
<td>28</td>
<td>91</td>
<td>188</td>
</tr>
</tbody>
</table>
Clashes with other packages

- Many other packages define a function called `select` (e.g., MASS)
- In R, the last package loaded, wins. Tidyverse usually loses since it’s loaded at the start!
- In these cases you need to specify the package as well as the function:
  ```r
dplyr::select()
```
- Can affect other functions too
Clashes with other packages

> library(MASS)
Attaching package: ‘MASS’
The following object is masked from ‘package:dplyr’:

  select

> dplyr::select
function (.data, ...)
{
  UseMethod("select")
}
<bytecode: 0x0000000017481b90>
<environment: namespace:dplyr>

> select
function (obj)
UseMethod("select")
<bytecode: 0x0000000025458408>
<environment: namespace:MASS>
Exercise 3
Filtering and selecting with dplyr
Grouping and Summarising

- `group_by` sets groups for summarisation
- `ungroup` removes grouping information
- `summarise` collapse grouped variables
- `count` count grouped variables
Grouping and Summarising Workflow

1. Load a tibble with repeated values in one or more columns

2. Use `group_by` to select all of the categorical columns you want to combine to define your groups

3. Run `summarise` saying how you want to combine the quantitative values

4. Run `ungroup` to remove any remaining group information
1. Load a tibble with repeated values in one or more columns

2. Use `group_by` to select all of the categorical columns you want to combine to define your groups

3. Run `summarise` saying how you want to combine the quantitative values

4. Run `ungroup` to remove any remaining group information
Grouping and Summarising

> group.data

# A tibble: 8 x 5
Sample Genotype Sex   Height Length
<dbl> <chr>    <chr>  <dbl>  <dbl>
1     1 WT       M         15    200
2     2 WT       F         13    185
3     3 WT       F         14    221
4     4 WT       M         18    265
5     5 KO       M         26    120
6     6 KO       F         22    165
7     7 KO       F         19    143
8     8 KO       M         27    110

• Want to get average Height and Length for each sex, but still split by genotype
Grouping and Summarising Workflow

1. Load a tibble with repeated values in one or more columns

2. Use `group_by` to select all of the categorical columns you want to combine to define your groups

3. Run `summarise` saying how you want to combine the quantitative values

4. Run `ungroup` to remove any remaining group information
Grouping and Summarising

- Discard: Sample
- Group: Genotype, Sex
- Group: Height, Length
- Mean: Categorical
- Median: Quantitative

• Want to get average Height and Length for each combination of sex and genotype
Grouping and Summarising

```r
group.data %>% group_by(Genotype, Sex)
```

# A tibble: 8 x 5
# Groups:   Genotype, Sex [4]

<table>
<thead>
<tr>
<th>Sample</th>
<th>Genotype</th>
<th>Sex</th>
<th>Height</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>WT</td>
<td>M</td>
<td>15</td>
<td>200</td>
</tr>
<tr>
<td>2</td>
<td>WT</td>
<td>F</td>
<td>13</td>
<td>185</td>
</tr>
<tr>
<td>3</td>
<td>WT</td>
<td>F</td>
<td>14</td>
<td>221</td>
</tr>
<tr>
<td>4</td>
<td>WT</td>
<td>M</td>
<td>18</td>
<td>265</td>
</tr>
<tr>
<td>5</td>
<td>KO</td>
<td>M</td>
<td>26</td>
<td>120</td>
</tr>
<tr>
<td>6</td>
<td>KO</td>
<td>F</td>
<td>22</td>
<td>165</td>
</tr>
<tr>
<td>7</td>
<td>KO</td>
<td>F</td>
<td>19</td>
<td>143</td>
</tr>
<tr>
<td>8</td>
<td>KO</td>
<td>M</td>
<td>27</td>
<td>110</td>
</tr>
</tbody>
</table>
Grouping and Summarising
Workflow

1. Load a tibble with repeated values in one or more columns

2. Use `group_by` to select all of the categorical columns you want to combine to define your groups

3. Run `summarise` saying how you want to combine the quantitative values

4. Run `ungroup` to remove any remaining group information
Grouping and Summarising

group.data %>%
group_by(Genotype, Sex) %>%
summarise(Height2 = mean(Height), Length = median(Length))

<table>
<thead>
<tr>
<th>Sample</th>
<th>Genotype</th>
<th>Sex</th>
<th>Height2</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discard</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group</td>
<td>Group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>Median</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

# A tibble: 4 x 4
# Groups:   Genotype [2]

<table>
<thead>
<tr>
<th>Genotype</th>
<th>Sex</th>
<th>Height2</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>KO</td>
<td>F</td>
<td>20.5</td>
<td>154</td>
</tr>
<tr>
<td>KO</td>
<td>M</td>
<td>26.5</td>
<td>115</td>
</tr>
<tr>
<td>WT</td>
<td>F</td>
<td>13.5</td>
<td>203</td>
</tr>
<tr>
<td>WT</td>
<td>M</td>
<td>16.5</td>
<td>232.</td>
</tr>
</tbody>
</table>
Grouping and Summarising
Workflow

1. Load a tibble with repeated values in one or more columns

2. Use `group_by` to select all of the categorical columns you want to combine to define your groups

3. Run `summarise` saying how you want to combine the quantitative values

4. Run `ungroup` to remove any remaining group information
Ungrouping

• A summarise operation removes the last level of grouping (“sex” in our worked example)

• Other levels of grouping (“Genotype”) remain annotated on the data, so you could do an additional summarisation if needed

• If you’re not going to use them it’s a good idea to use `ungroup` to remove remaining groups so they don’t interfere with other operations
Exercise 4
Grouping and Summarising
Extending tibbles

• **mutate**  compute a new column
• **add_row**  adds an additional row
• **bind_rows**  join two tibbles by row
• **add_column**  adds an additional column
• **bind_cols**  join two tibbles by column
• **rename**  renames an existing column
Using `mutate`

```r
mutate(data, newcol=oldcol1+oldcol2)
```

```r
trumpton %>% mutate(bmi=Weight/(Height/100)^2)
```

# A tibble: 7 x 6

<table>
<thead>
<tr>
<th>LastName</th>
<th>FirstName</th>
<th>Age</th>
<th>Weight</th>
<th>Height</th>
<th>bmi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hugh</td>
<td>Chris</td>
<td>26</td>
<td>90</td>
<td>175</td>
<td>29.4</td>
</tr>
<tr>
<td>Pew</td>
<td>Adam</td>
<td>32</td>
<td>102</td>
<td>183</td>
<td>30.5</td>
</tr>
<tr>
<td>Barney</td>
<td>Daniel</td>
<td>18</td>
<td>88</td>
<td>168</td>
<td>31.2</td>
</tr>
<tr>
<td>McGrew</td>
<td>Chris</td>
<td>48</td>
<td>97</td>
<td>155</td>
<td>40.4</td>
</tr>
<tr>
<td>Cuthbert</td>
<td>Carl</td>
<td>28</td>
<td>91</td>
<td>188</td>
<td>25.7</td>
</tr>
<tr>
<td>Dibble</td>
<td>Liam</td>
<td>35</td>
<td>94</td>
<td>145</td>
<td>44.7</td>
</tr>
<tr>
<td>Grub</td>
<td>Doug</td>
<td>31</td>
<td>89</td>
<td>164</td>
<td>33.1</td>
</tr>
</tbody>
</table>
Using `add_row`

```
add_row(data, key1=value1, key2=value2)
```

```
trumpton %>%
  add_row(
    LastName = "Andrews", FirstName = "Simon",
    Age = 39, Weight = 80, Height = 190
  )
```

```
# A tibble: 8 x 5
  LastName FirstName Age Weight Height
  <chr>     <chr>  <dbl>  <dbl>  <dbl>
1 Hugh     Chris   26     90    175
2 Pew      Adam    32    102    183
3 Barney   Daniel  18     88    168
4 McGrew   Chris   48     97    155
5 Cuthbert Carl   28     91    188
6 Dibble   Liam    35     94    145
7 Grub     Doug    31     89    164
8 Andrews  Simon   39     80    190
```
Using `bind_rows` 

`bind_rows(data1, data2)`

`bind_rows(some.data, some.more.data)`

```r
> some.data
# A tibble: 3 x 2
  gene  value
  <chr> <dbl>
1 ABC10 10
2 DEF10 12
3 GHI10 14

> some.more.data
# A tibble: 6 x 2
  gene  value
  <chr> <dbl>
1 ABC10 10
2 DEF10 12
3 GHI10 14
4 JKL10 16
5 MNO10 18
6 PQR10 20
```
Using `add_column`

```r
add_column(data, newcol=new_vector)
```

```r
add_column(some.data, value2=c(16,18,20))
```

# A tibble: 3 x 3
  gene value value2
  <chr> <dbl>   <dbl>
1 ABC10 10     16
2 DEF10 12     18
3 GHI10 14     20
Using `bind_cols`

`bind_cols(tibble1, tibble2)`

`bind_cols(some.data, some.more.data)`

```r
> some.data
# A tibble: 3 x 2
  gene  value
  <chr> <dbl>
1 ABC10   10
2 DEF10   12
3 GHI10   14
```

```r
> some.more.data
# A tibble: 3 x 2
  gene1 value1
  <chr>   <dbl>
1 JKL10    16
2 MNO10    18
3 PQR10    20
```
Using `rename`

```r
rename(data, new_name = name)
```

```r
rename(some.data, gene_name = gene)
```

---

```r
> some.data
# A tibble: 3 x 2
gene value
<chr> <dbl>
1 ABC10 10
2 DEF10 12
3 GHI10 14
```

```r
# A tibble: 3 x 2
gene_name value
<chr> <dbl>
1 ABC10 10
2 DEF10 12
3 GHI10 14
```
Joining tibbles x and y

- **left_join**: join matching values from y into x
- **right_join**: join matching values of x into y
- **inner_join**: join x and y keeping only rows in both
- **full_join**: join x and y keeping all values in both
## Join types

### join1

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon</td>
<td>3</td>
</tr>
<tr>
<td>Steven</td>
<td>6</td>
</tr>
<tr>
<td>Felix</td>
<td>2</td>
</tr>
</tbody>
</table>

### join2

<table>
<thead>
<tr>
<th>name</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Felix</td>
<td>10</td>
</tr>
<tr>
<td>Anne</td>
<td>25</td>
</tr>
<tr>
<td>Simon</td>
<td>36</td>
</tr>
</tbody>
</table>

### left_join

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>Steven</td>
<td>6</td>
<td>NA</td>
</tr>
<tr>
<td>Felix</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

### right_join

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Felix</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Anne</td>
<td>NA</td>
<td>25</td>
</tr>
<tr>
<td>Simon</td>
<td>3</td>
<td>36</td>
</tr>
</tbody>
</table>

### inner_join

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>Felix</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

### full_join

<table>
<thead>
<tr>
<th>name</th>
<th>count</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simon</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td>Steven</td>
<td>6</td>
<td>NA</td>
</tr>
<tr>
<td>Felix</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Anne</td>
<td>NA</td>
<td>25</td>
</tr>
</tbody>
</table>
Joining options

- **by** specify the columns to join on
  - Simple name if it's the same between both
    by="gene"
  - Paired names if they differ between x and y
    by=c("gene" = "gene_name")

- **suffix** the text suffix for duplicated column names
Exercise 5
Extending and Joining