Introduction to R
(with Tidyverse)

Simon Andrews
v2020-07
R can just be a calculator

> 3+2
[1] 5

> 2/7
[1] 0.2857143

> 5^10
[1] 9765625
Storing numerical data in variables

10 -> x
y <- 20

x
[1] 10

x+y
[1] 30

x+y -> z
Variable names

• The rules
  • Can't start with a number
  • Made up of letters, numbers dots and underscores

• The guidelines
  • Make the name mean something (x = bad, weight = good)
  • Keep variables all lower case
  • Separate words with dots or underscores
    gene_name or gene.name are the preferred options
Storing text in variables

my.name <- "simon"

my.other.name <- 'andrews'
Running a simple function

\[ \sqrt{10} \]

[1] 3.162278
Looking up help

?sqrt

MathFun {base}

Miscellaneous Mathematical Functions

Description

abs(x) computes the absolute value of x, sqrt(x) computes the (principal) square root of x, \( \sqrt{x} \).

The naming follows the standard for computer languages such as C or Fortran.

Usage

abs(x)
sqrt(x)

Arguments

x a numeric or complex vector or array.

Details

These are \texttt{internal generic primitive} functions: methods can be defined for them individually or via the \texttt{Math} group generic. For complex arguments (and the default method), \( z, \texttt{abs}(z) \rightarrow \texttt{Mod}(z) \) and \texttt{sqrt}(z) \rightarrow z^0.5.

abs(x) returns an integer vector when x is integer or logical.
Searching Help

Search Results

Help pages:

- Biostrings::class:MultipleAlignment
- Biostrings::class:XString
- Biostrings::class:XStringSet
- Biostrings::class:XStringSetList
- Biostrings::class:XStringViews
- Biostrings::letterFrequency
- Biostrings::longestConsecutive
- Biostrings::lc_prefix
- Biostrings::extractAt
- crayon::col_substr
- crayon::col_substring
- Hmisc::makeNet
- Hmisc::sed
- S4Vectors::Rcutils
- strings::stri_sub
- strings::stri_sub
- base::regmatches
- base::substr

MultipleAlignment objects
XString objects
XStringSet objects
XStringSetList objects
The XStringViews class
Calculate the frequency of letters in a biological sequence, or the consensus matrix of a set of sequences
Obtain the length of the longest substring containing only 'letter'
Longest Common Prefix/Suffix/Substring searching functions
Extract/replace arbitrary substrings from/in a string or set of strings.
Substring(s) of an ANSI colored string
Substring(s) of an ANSI colored string
creates a string that is a repeat of a substring
Character String Editing and Miscellaneous Character Handling Functions
Common operations on Rle objects
Extract a Substring From or Replace a Substring In a Character Vector
Extract and replace substrings from a character vector.
Extract or Replace Matched Substrings
Substrings of a Character Vector
Substrings of a Character Vector

Description

Extract or replace substrings in a character vector.

Usage

\[
\text{substr}(x, \text{start}, \text{stop})
\]
\[
\text{substring}(\text{text}, \text{first}, \text{last} = 10000001)
\]
\[
\text{substr}(x, \text{start}, \text{stop}) \leftarrow \text{value}
\]
\[
\text{substring}(\text{text}, \text{first}, \text{last} = 10000001) \leftarrow \text{value}
\]

Arguments

- \text{x, text} a character vector.
- \text{start, first} integer The first element to be replaced.
- \text{stop, last} integer The last element to be replaced.
- \text{value} a character vector, recycled if necessary.
Passing arguments to functions

```r
substr(my.name, 2, 4)
[1] "imo"

substr(x=my.name, start=2, stop=4)
[1] "imo"

substr(
   start=2,
   stop=4,
   x=my.name
)
[1] "imo"
```
Exercise 1
Everything is a vector

- Vectors are the most basic unit of storage in R
- Vectors are ordered sets of values of the same type
  - Numeric
  - Character (text)
  - Factor (repeated text values)
  - Logical (TRUE or FALSE)
  - Date etc...

10 -> x

x is a vector of length 1 with 10 as its first value
Creating vectors manually

• Use the `c` (combine) function

  `c(1, 2, 4, 6, 3) -> simple.vector`

  `c("simon", "laura", "anne", "jo", "steven") -> some.names`

• Data must be of the same type

  `c(1, 2, 3, "fred")`

  `[1] "1" "2" "3" "fred"`
Functions for creating vectors

• `rep` - repeat values

  ```r
  rep(2, times=10)
  [1] 2 2 2 2 2 2 2 2 2 2
  
  rep("hello", times=5)
  [1] "hello" "hello" "hello" "hello" "hello" "hello"
  
  rep(c("dog", "cat"), times=3)
  [1] "dog" "cat" "dog" "cat" "dog" "cat"
  
  rep(c("dog", "cat"), each=3)
  [1] "dog" "dog" "dog" "cat" "cat" "cat"
  ```
Functions for creating vectors

- seq - create numerical sequences
  - No required arguments!
    - from
    - to
    - by
    - length.out

- Specify enough that the series is unique
Functions for creating vectors

- **seq** - create numerical sequences

seq(from=2, by=3, to=14)
[1] 2 5 8 11 14

seq(from=3, by=10, to=40)
[1] 3 13 23 33

seq(from=5, by=3.6, length.out=5)
[1] 5.0 8.6 12.2 15.8 19.4
Functions for creating vectors

• Sampling from statistical distributions
  • `rnorm`
  • `runif`
  • `rpois`
  • `rbeta`
  • `rbinom`

```r
rnorm(10000)
```

• Statistically testing vectors
  • `t.test`
  • `lm`
  • `cor.test`
  • `aov`

```r
t.test(
  c(1,5,3),
  c(10,15,30)
)
```
Language shortcuts for vector creation

• Single elements
  ```
c("simon")
"simon"
```

• Integer series
  ```
seq(from=4,to=20,by=1)
4:20
```

[1] 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Viewing large variables

• In the console
  head(data)
  tail(data,n=10)

• Graphically
  View(data)  [Note capital V!]
  Click in Environment tab
Vectorised Operations

\[ 2 + 3 \]
\[ [1] \ 5 \]

\[ \text{c}(2,4) + \text{c}(3,5) \]
\[ [1] \ 5 \ 9 \]

simple.vector

\begin{array}{cccccc}
1 & 2 & 4 & 6 & 3 \\
\end{array}

simple.vector \times 100

\begin{array}{cccccc}
100 & 200 & 400 & 600 & 300 \\
\end{array}
Rules for vectorised operations

• Equivalent positions are matched

Vector 1

| 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Vector 2

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |

14 16 18 20 22 24 26 28
Rules for vectorised operations

- Shorter vectors are recycled

Vector 1

3 4 5 6 7 8 9 10

Vector 2

11 12 13 14

14 16 18 20 18 20 22 24
Rules for vectorised operations

• Incomplete vectors generate a warning

Warning message:
In 3:10 + 11:13 :
  longer object length is not a multiple of shorter object length
Vectorised Operations

c(2,4) + c(3,5)
[1] 5 9

simple.vector
1 2 4 6 3

simple.vector * 100
100 200 400 600 300
Exercise 2
R Data Structures
Vector

• 1D Data Structure of fixed type

<table>
<thead>
<tr>
<th>scores</th>
<th>mean(scores)</th>
<th>sd(scores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.7</td>
<td></td>
</tr>
</tbody>
</table>
List

• Collection of vectors

```
results$count
mean(results$count)
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.8</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2.7</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td></td>
</tr>
</tbody>
</table>
Data Frame

- Collection of vectors with same lengths
- Gain the concept of 'rows'

```r
all.results$mon
mean(all.results$mon)
```
Tibble

- Collection of vectors with same lengths
- Gain the concept of 'rows'

```r
all.results$mon
mean(all.results$mon)
```
Tibbles are nicer dataframes

```r
> head(as.data.frame(data))

<table>
<thead>
<tr>
<th>Probe</th>
<th>Chromosome</th>
<th>Start</th>
<th>End</th>
<th>Probe</th>
<th>Strand</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL645608.2</td>
<td>1</td>
<td>911435</td>
<td>914948</td>
<td>+</td>
<td>AL645608.2</td>
<td></td>
</tr>
<tr>
<td>LINC02593</td>
<td>1</td>
<td>916865</td>
<td>921016</td>
<td>-</td>
<td>LINC02593</td>
<td></td>
</tr>
<tr>
<td>SAMD11</td>
<td>1</td>
<td>923928</td>
<td>944581</td>
<td>+</td>
<td>SAMD11</td>
<td></td>
</tr>
<tr>
<td>TMEM51-AS1</td>
<td>1</td>
<td>15111815</td>
<td>15153618</td>
<td>-</td>
<td>TMEM51-AS1</td>
<td></td>
</tr>
<tr>
<td>TMEM51</td>
<td>1</td>
<td>15152532</td>
<td>15220478</td>
<td>+</td>
<td>TMEM51</td>
<td></td>
</tr>
<tr>
<td>FHAD1</td>
<td>1</td>
<td>15247272</td>
<td>15400283</td>
<td>+</td>
<td>FHAD1</td>
<td></td>
</tr>
</tbody>
</table>

Description

1 novel transcript
2 long intergenic non-protein coding RNA 2593 [Source:HGNC Symbol;Acc:HGNC:53933]
3 sterile alpha motif domain containing 11 [Source:HGNC Symbol;Acc:HGNC:28706]
4 TMEM51 antisense RNA 1 [Source:HGNC Symbol;Acc:HGNC:26301]
5 transmembrane protein 51 [Source:HGNC Symbol;Acc:HGNC:25488]
6 forkhead associated phosphopeptide binding domain 1 [Source:HGNC Symbol;Acc:HGN]
Tibbles are nicer dataframes

```r
> head(as_tibble(data))
# A tibble: 6 x 12
     Probe Chromosome Start   End `Probe Strand` Feature ID    Description
   <chr>    <dbl>    <dbl>    <dbl>        <chr>    <chr> <chr>
 1   AL64~    1.0 9.11e5 9.15e5       + AL6456~ ENSG~ novel tran~
 2   LINC~    1.0 9.17e5 9.21e5       - LINC02~ ENSG~ long inter~
 3   SAMD~    1.0 9.24e5 9.45e5       + SAMD11 ENSG~ sterile al~
 4   TMEM~    1.0 1.51e7 1.52e7       - TMEM51~ ENSG~ TMEM51 ant~
 5   TMEM~    1.0 1.52e7 1.52e7       + TMEM51~ ENSG~ transmembr~
 6   FHAD1    1.0 1.52e7 1.54e7       + FHAD1   ENSG~ forkhead a~
# ... with 4 more variables: `Feature Strand` <chr>, Type <chr>, `Feature
#   Orientation` <chr>, Distance <dbl>
```
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
Installation and calling

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

```
-- Attaching packages  ------- tidyverse 1.3.0 --
  v ggplot2 3.3.2      v purrr 0.3.4
  v tibble 3.0.2      v dplyr 1.0.0
  v tidyr 1.1.0       v stringr 1.4.0
  v readr 1.3.1       v forcats 0.5.0
```

```
-- Conflicts  ---------- tidyverse_conflicts() --
  x dplyr::filter() masks stats::filter()
  x dplyr::lag() masks stats::lag()
```
Reading and Writing Files with readr

• Provides functions to read from text files into tibbles or write from tibbles to text files

  • `read_csv("file.csv")` -> data
  • `read_tsv("file.tsv")` -> data

  • `write_csv(data,"file.csv")`
  • `write_tsv(data,"file.csv")`
Specifying file paths

• You can use full file paths, but it's a pain
  
  ```r
  read_csv("0:/Training/R_tidyverse_intro_data/neutrophils.csv")
  ```

• Just set the 'working directory' and then just provide a file name
  • `setwd(path)`
    • Session > Set Working Directory > Choose Directory
    • File > New Project > Existing Directory

• Use [Tab] to fill in file paths in the editor
  • `read_tsv(""`) – put the cursor in the quotes and press tab
Reading files with readr

```r
> read_tsv("trumpton.txt") -> trumpton
Parsed with column specification:
cols(
  LastName = col_character(),
  FirstName = col_character(),
  Age = col_double(),
  Weight = col_double(),
  Height = col_double()
)

> 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
```

Reading files with `readr`
Exercise 3
'Tidy' Data Format

• Tibbles give you a 2D data structure where each column must be of a fixed data type
• Often data can be put into this sort of structure in more than one way
• Is there a right / wrong way to structure your data?

• Tidyverse has an opinion!
Long vs Wide Data Modelling

• Consider a simple experiment:

• Two genes tested (ABC1 and DEF1)
• Two conditions (WT and KO)
• Three replicates for each condition
## Wide Format

<table>
<thead>
<tr>
<th>Gene</th>
<th>WT_1</th>
<th>WT_2</th>
<th>WT_3</th>
<th>KO_1</th>
<th>KO_2</th>
<th>KO_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC1</td>
<td>8.86</td>
<td>4.18</td>
<td>8.90</td>
<td>4.00</td>
<td>14.52</td>
<td>13.39</td>
</tr>
<tr>
<td>DEF1</td>
<td>29.60</td>
<td>41.22</td>
<td>36.15</td>
<td>11.18</td>
<td>16.68</td>
<td>1.64</td>
</tr>
</tbody>
</table>

- Compact
- Easy to read
- Shows linkage for genes
- No explicit genotype or replicate
- Values spread out over multiple rows and columns
- Not extensible to more metadata
Long Format

<table>
<thead>
<tr>
<th>Gene</th>
<th>Genotype</th>
<th>Replicate</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC1</td>
<td>WT</td>
<td>1</td>
<td>8.86</td>
</tr>
<tr>
<td>ABC1</td>
<td>WT</td>
<td>2</td>
<td>4.18</td>
</tr>
<tr>
<td>ABC1</td>
<td>WT</td>
<td>3</td>
<td>8.90</td>
</tr>
<tr>
<td>ABC1</td>
<td>KO</td>
<td>1</td>
<td>4.00</td>
</tr>
<tr>
<td>ABC1</td>
<td>KO</td>
<td>2</td>
<td>14.52</td>
</tr>
<tr>
<td>ABC1</td>
<td>KO</td>
<td>3</td>
<td>13.39</td>
</tr>
<tr>
<td>DEF1</td>
<td>WT</td>
<td>1</td>
<td>29.60</td>
</tr>
<tr>
<td>DEF1</td>
<td>WT</td>
<td>2</td>
<td>41.22</td>
</tr>
<tr>
<td>DEF1</td>
<td>WT</td>
<td>3</td>
<td>36.15</td>
</tr>
<tr>
<td>DEF1</td>
<td>KO</td>
<td>1</td>
<td>11.18</td>
</tr>
<tr>
<td>DEF1</td>
<td>KO</td>
<td>2</td>
<td>16.68</td>
</tr>
<tr>
<td>DEF1</td>
<td>KO</td>
<td>3</td>
<td>1.64</td>
</tr>
</tbody>
</table>

- More verbose (repeated values)
- Explicit genotype and replicate
- All values in a single column
- Extensible to more metadata
Filtering and subsetting

- Tidyverse (specifically dplyr) comes with functions to manipulate your data.

- All functions take a tibble as their first argument
- All functions return a modified tibble
  - Selecting columns
  - Logical subsetting
The data we're starting with

```r
> trumpton
# A tibble: 7 x 5
   LastName FirstName Age Weight Height
 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 select to pick columns

```r
> select(trumpton, FirstName, LastName, Weight)

# A tibble: 7 x 3

  FirstName LastName Weight
  <chr>     <chr>     <dbl>
1  Chris     Hugh      90
2  Adam      Pew       102
3  Daniel    Barney    88 
4  Chris     McGrew    97 
5  Carl      Cuthbert  91 
6  Liam      Dibble    94 
7  Doug      Grub      89 
```
You can use positions instead of names

```r
> select(trumpton, 2,4)
# A tibble: 7 x 2
  FirstName Weight
  <chr>    <dbl>
1 Chris    90
2 Adam    102
3 Daniel  88
4 Chris   97
5 Carl    91
6 Liam    94
7 Doug    89
```
You can use negative selections

```r
> select(trumpton, -LastName)
# A tibble: 7 x 4
    FirstName Age Weight Height
  <chr>    <dbl>  <dbl>  <dbl>
1  Chris    26     90    175
2  Adam     32    102    183
3 Daniel   18     88    168
4  Chris    48     97    155
5  Carl     28     91    188
6 Liam      35     94    145
7 Doug     31     89    164
```
Functional selections using filter

```r
> filter(trumpton, Height>=170)
# A tibble: 3 x 5
  LastName FirstName Age Weight Height
  <chr>    <chr> <int> <dbl>  <dbl>
1 Hugh     Chris  26     90     175
2 Pew      Adam  32    102     183
3 Cuthbert Carl  28     91     188
```
Types of filter you can use

• Greater than
  - weight > 20
  - weight >= 30

• Less than
  - height < 170
  - height <= 180

• Equal to (or not)
  - value == 5
  - name == "simon"
  - name != "simon"

```r
> filter(trumpton, FirstName == "Chris")
# A tibble: 2 x 5
  LastName FirstName Age Weight Height
  <chr>    <chr>  <db1> <db1> <db1>
1  Hugh     Chris   26    90    175
2 McGrew   Chris   48    97    155
```
You can transform data in a filter

Select rows where the difference (in either direction) is more than 5

\[
\text{filter(\text{transform.data, difference} > 5)}
\]

\[
\begin{array}{ccc}
\text{WT} & \text{KO} & \text{difference} \\
1 & -8.69 & -2.38 & 6.31 \\
\end{array}
\]

\[
\text{filter(\text{transform.data, difference} < -5)}
\]

\[
\begin{array}{ccc}
\text{WT} & \text{KO} & \text{difference} \\
1 & -1.93 & -10.0 & -8.10 \\
\end{array}
\]

\[
\text{filter(\text{transform.data, abs(difference)} > 5)}
\]

\[
\begin{array}{ccc}
\text{WT} & \text{KO} & \text{difference} \\
1 & -1.93 & -10.0 & -8.10 \\
2 & -8.69 & -2.38 & 6.31 \\
\end{array}
\]
Exercise 4
Combining Multiple Operations

• Find people who are:
  1. Taller than 170cm
  2. Called Chris

• Then report only their age and weight
Combing multiple operations

• The long winded way...
• Three separate operations with two intermediate variables
• Works, but is ugly!

```r
> filter(trumpton, Height >= 170) -> answer1
> filter(answer1, FirstName == "Chris") -> answer2
> select(answer2, Age, Weight)

# A tibble: 1 x 2
  Age  Weight
  <dbl>  <dbl>
1  26    90
```
Pipes to the rescue

• All tidyverse functions take a tibble as their first argument

• All tidyverse functions return a tibble

• You can therefore chain operations together, passing the output of one function as the first input to another

Data → Filter 1 → Filter 2 → Selection
The pipe operator: `%>%`

- Takes the data on its left and makes it the first argument to a function on its right.

```r
> select(trumpton, -LastName)
# A tibble: 7 x 4
##   FirstName Age Weight Height
##    <chr> <int> <int>    <int>
## 1    Chris   26     90     175
## 2   Adam     32    102     183
## 3  Daniel    18     88     168
## 4    Chris    48     97     155
## 5   Carl     28     91     188
## 6   Liam     35     94     145
## 7   Doug     31     89     164
```

```r
> trumpton %>% select(-LastName)
# A tibble: 7 x 4
##   FirstName Age Weight Height
##    <chr> <int> <int>    <int>
## 1    Chris   26     90     175
## 2   Adam     32    102     183
## 3  Daniel    18     88     168
## 4    Chris    48     97     155
## 5   Carl     28     91     188
## 6   Liam     35     94     145
## 7   Doug     31     89     164
```
Combining Multiple Operations with Pipes

• Give the age and weight for people who are taller than 170cm and called Chris

```r
trumpton %>% filter(Height>=170) %>% filter(FirstName=="Chris") %>% select(Age,Weight)
```

```
# A tibble: 1 x 2
   Age Weight
  <dbl>   <dbl>
1   26    90
```
Exercise 5
Plotting figures and graphs with ggplot

• ggplot is the plotting library for tidyverse
  • Powerful
  • Flexible

• Follows the same conventions as the rest of tidyverse
  • Data stored in tibbles
  • Data is arranged in 'tidy' format
  • Tibble is the first argument to each function
Code structure of a ggplot graph

• Start with a call to ggplot()
  • Pass the tibble of data
  • Say which columns you want to use

• Say which graphical representation you want to use
  • Points, lines, barplots etc

• Customise labels, colours annotations etc.
Geometries and Aesthetics

• Geometries are types of plot
  
  - `geom_point()`  
    Point geometry, (x/y plots, stripcharts etc)  
  - `geom_line()`  
    Line graphs  
  - `geom_boxplot()`  
    Box plots  
  - `geom_bar()`  
    Barplots  
  - `geom_histogram()`  
    Histogram plots

• Aesthetics are graphical parameters which can be adjusted in a given geometry
Aesthetics for `geom_point()`

Aesthetics

`geom_point()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `alpha`
- `colour`
- `fill`
- `group`
- `shape`
- `size`
- `stroke`
Mappings can be quantitative or categorical
How do you define aesthetics

• Fixed values
  • Colour all points red
  • Make the points size 4

• Encoded from your data – called an **aesthetic mapping**
  • Colour according to genotype
  • Size based on the number of observations

• Aesthetic mappings are set using the `aes()` function, normally as an argument to the `ggplot` function

```r
data %>% ggplot(aes(x=weight, y=height, colour=genotype))
```
Putting things together

• Identify the tibble with the data you want to plot
• Decide on the geometry (plot type) you want to use
• Decide which columns will modify which aesthetic

• Call `ggplot(aes(...))`
• Add a `geom_XXX` function call
Our first plot...

```r
> expression
# A tibble: 12 x 4
   Gene  WT    KO pValue
   <chr> <dbl> <dbl> <dbl>
1 Mia1  5.83  3.24  0.1
2 Snrpa 8.59  5.02  0.001
3 Itpkc 8.49  6.16  0.04
4 Adck4 7.69  6.41  0.2
5 Numbl 8.37  6.81  0.1
6 Ltbp4 6.96 10.4  0.001
7 Shkbp1 7.57  5.83  0.1
8 Spnb4 10.7  9.38  0.2
9 Blvrb  7.32  5.29  0.05
10 Pgam1 0.0   0.285 0.5
11 Sertad3 8.13  3.02  0.0001
12 Sertad1 7.69  4.34  0.01
```

```r
ggplot(expression, aes(x=WT, y=KO)) + geom_point()
```

- Identify the tibble with the data you want to plot
- Decide on the geometry (plot type) you want to use
- Decide which columns will modify which aesthetic
- Call `ggplot(aes(...))`
- Add a `geom_` function call
Our second plot...

```r
> ggplot(expression, aes(x=WT, y=KO)) + geom_line()
```

# A tibble: 12 x 4

<table>
<thead>
<tr>
<th>Gene</th>
<th>WT</th>
<th>KO</th>
<th>pValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;chr&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
<td>&lt;dbl&gt;</td>
</tr>
<tr>
<td>Mia1</td>
<td>5.83</td>
<td>3.24</td>
<td>0.1</td>
</tr>
<tr>
<td>Snrpa</td>
<td>8.59</td>
<td>5.02</td>
<td>0.001</td>
</tr>
<tr>
<td>Itpkc</td>
<td>8.49</td>
<td>6.16</td>
<td>0.04</td>
</tr>
<tr>
<td>Adck4</td>
<td>7.69</td>
<td>6.41</td>
<td>0.2</td>
</tr>
<tr>
<td>Numbl</td>
<td>8.37</td>
<td>6.81</td>
<td>0.1</td>
</tr>
<tr>
<td>Ltbp4</td>
<td>6.96</td>
<td>10.4</td>
<td>0.001</td>
</tr>
<tr>
<td>Shkbpl</td>
<td>7.57</td>
<td>5.83</td>
<td>0.1</td>
</tr>
<tr>
<td>Spnb4</td>
<td>10.7</td>
<td>9.38</td>
<td>0.2</td>
</tr>
<tr>
<td>Blvrb</td>
<td>7.32</td>
<td>5.29</td>
<td>0.05</td>
</tr>
<tr>
<td>Pgaml</td>
<td>0</td>
<td>0.285</td>
<td>0.5</td>
</tr>
<tr>
<td>Sertad3</td>
<td>8.13</td>
<td>3.02</td>
<td>0.0001</td>
</tr>
<tr>
<td>Sertad1</td>
<td>7.69</td>
<td>4.34</td>
<td>0.01</td>
</tr>
</tbody>
</table>
Our third plot...

```r
expression %>%
  ggplot (aes(x=WT, y=KO)) +
  geom_point(color="red2", size=5)
```
Exercise 6
Other plot types

• Barplots
  • geom_bar
  • geom_col

• Histograms
  • geom_histogram

• Density plots
  • geom_density
Drawing a barplot (geom_col())

Aesthetics

geom_bar() understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill
- group
- linetype
- size

- Plot the expression values for the WT samples for all genes
- What is your X?
- What is your Y?

> expression
  # A tibble: 12 x 4
  Gene      WT     KO pValue
  <chr>    <dbl>  <dbl> <dbl>
1 Mia1     5.83  3.24  0.1
2 Snrpa    8.59  5.02  0.001
Our bar plot...

```r
expression %>%
  ggplot(aes(x=Gene, y=WT)) +
  geom_col()
```
Our bar plot...

```r
expression %>%
  ggplot(aes(x=Gene, y=WT)) +
  geom_col(fill="red2")
```
Counting bar plot...

dogs %>%
ggplot(aes(x=size)) +
geom_bar()

> dogs
# A tibble: 56 x 2
  size               breed
  <chr>            <chr>
1 Extra Large (XL) Airedale Terrier
2 Extra-Extra Large (XXL or 2XL) Akita
3 Extra Large (XL) American Foxhound
4 Extra Large (XL) Australian Shepherd
5 Extra Large (XL) Bassett Hound
6 Medium (M)       Beagle
7 Extra-Extra Large (XXL or 2XL) Bernese Mountain Dog
8 Medium (M)       Bichon Frise
9 Small (S)        Boston Terrier
10 Medium (M)      Boston Terrier
# ... with 46 more rows
Plotting distributions - histograms

```
many.values %>%
ggplot(aes(values)) +
  geom_histogram(binwidth = 0.1, fill="yellow", colour="black")
```

```
> many.values
# A tibble: 100,000 x 2
  values genotype
  <dbl>  <chr>
1  1.90   KO
2  2.39   WT
3  4.32   KO
4  2.94   KO
5  0.728  WT
6 -0.280  WT
7  0.337  WT
8 -1.31   WT
9  1.55   WT
10 1.86   KO
```
Plotting distributions - density

many.values %>%
ggplot(aes(values)) +
geom_density(fill="yellow", colour="black")
Plotting distributions - density

many.values %>%
ggplot(aes(x=values, fill=genotype)) +
ggeom_density(colour="black")
Plotting distributions - density

many.values %>%
ggplot(aes(x=values, fill=genotype)) +
geom_density(colour="black", alpha=0.5)
Other annotation geometries

```r
expression %>%
ggplot(aes(x=WT, y=KO, label=Gene)) +
  geom_point() +
  ggtitle("Expression level comparison") +
  xlab("WT Expression level (log2 RPM)") +
  ylab("KO Expression level (log2 RPM)") +
  geom_text(vjust=1.2)
```
Exercise 7