Introduction to R
(with Tidyverse)

Simon Andrews
v2022-06
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
  • Made up of letters, numbers dots and underscores
  • Can't start with a number or underscore

• 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

"simon" -> my.name

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

\[ \text{sqrt}(10) \]

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

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

Help pages:

- Biostrings::class MultipleAlignment
- Biostrings::class XString
- Biostrings::class XStringSet
- Biostrings::class XStringSetList
- Biostrings::class XStringViews
- Biostrings::letterFrequency
- Biostrings::longestConsecutive
- Biostrings::lcprefix
- Biostrings::extractAt
- crayon::col_substr
- crayon::col_substring
- Hmisc::makeNet
- Hmisc::sedint
- S4Vectors::Rleutils
- strings::stri_sub
- strings::stri_sub
- base::regmatches
- base::substr

MultipleAlignment objects
BString object
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
Searching Help

substr {base}

Substrings of a Character Vector

Description

Extract or replace substrings in a character vector.

Usage

substr(x, start, stop)
substring(text, first, last = 1000000L)
substr(x, start, stop) <- value
substring(text, first, last = 1000000L) <- value

Arguments

x, text       a character vector.
start, first  integer. The first element to be replaced.
stop, last    integer. The last element to be replaced.
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
  
  \[
  c(1,2,4,6,3) \rightarrow \text{simple.vector}
  \]
  
  \[
  c(\text{"simon"},\text{"laura"},\text{"anne"},\text{"jo"},\text{"louise"}) \rightarrow \text{some.names}
  \]

• Data must be of the same type
  
  \[
  c(1,2,3,\text{"fred"})
  \]
  
  \[
  [1] \ "1" \ "2" \ "3" \ "fred"
  \]
Functions for creating vectors

• rep - repeat values

  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

  rnorm(10000)

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

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

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

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

  ```r
[1] 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
  ```
Vectorised Operations

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

\[ c(2,4,7) + c(3,5,1) \]
\[ [1] \ 5 \ 9 \ 8 \]

simple.vector
\[
1 \ 2 \ 4 \ 6 \ 3
\]

simple.vector * 100
\[
100 \ 200 \ 400 \ 600 \ 300
\]
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 |

\[
\begin{array}{cccccccccc}
\text{Vector 1} & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{Vector 2} & 11 & 12 & 13 & 14 \\
\end{array}
\]
Rules for vectorised operations

• Incomplete vectors generate a warning

Vector 1

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

Vector 2

| 11 | 12 | 13 |

14 16 18 17 19 21 20 22

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

c(2,4,7) + c(3,5,1)
[1] 5 9 8

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$counts
mean(results$counts)
```

```
results
```

```
<table>
<thead>
<tr>
<th>ratios</th>
<th>counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>2</td>
<td>1.2</td>
</tr>
<tr>
<td>3</td>
<td>3.3</td>
</tr>
<tr>
<td>4</td>
<td>1.8</td>
</tr>
<tr>
<td>5</td>
<td>2.7</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
</tr>
<tr>
<td>3</td>
<td>200</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))
   Probe   Chromosome Start      End Probe Strand Feature                                   Description
   1    AL645608.2     1   911435   914948 + AL645608.2 novel transcript
   2     LINC02593     1   916865   921016 - LINC02593
   3      SAMD11      1   923928   944581 + SAMD11
   4  TMEM51-AS1      1  15111815 15153618 - TMEM51-AS1
   5      TMEM51      1  15152532 15220478 + TMEM51
   6      FHAD1      1  15247272 15400283 + FHAD1
```

- **Probe**
- **Chromosome**: 1
- **Start**: 911435
- **End**: 914948
- **Probe**: AL645608.2
- **Strand**: +
- **Feature**: AL645608.2

- **Description**: novel transcript

- **Probe**
- **Chromosome**: 1
- **Start**: 916865
- **End**: 921016
- **Probe**: LINC02593
- **Strand**: -
- **Feature**: LINC02593

- **Probe**
- **Chromosome**: 1
- **Start**: 923928
- **End**: 944581
- **Probe**: SAMD11
- **Strand**: +
- **Feature**: SAMD11

- **Probe**
- **Chromosome**: 1
- **Start**: 15111815
- **End**: 15153618
- **Probe**: TMEM51-AS1
- **Strand**: -
- **Feature**: TMEM51-AS1

- **Probe**
- **Chromosome**: 1
- **Start**: 15152532
- **End**: 15220478
- **Probe**: TMEM51
- **Strand**: +
- **Feature**: TMEM51

- **Probe**
- **Chromosome**: 1
- **Start**: 15247272
- **End**: 15400283
- **Probe**: FHAD1
- **Strand**: +
- **Feature**: FHAD1

- **Description**: novel transcript

- **Probe**
- **Chromosome**: 1
- **Start**: 911435
- **End**: 914948
- **Probe**: AL645608.2
- **Strand**: +
- **Feature**: AL645608.2

- **Description**: novel transcript
Tibbles are nicer dataframes

> head(as_tibble(data))
# A tibble: 6 x 12
Probes Chromosome Start End `Probe Strand` Feature ID Description
<chr> <dbl> <dbl> <dbl> <chr> <chr> <chr> <chr>
1 AL64~ 1 9.11e5 9.15e5 + AL6456~ ENSG~ novel tran
2 LINC~ 1 9.17e5 9.21e5 LINC02~ ENSG~ long inter
3 SAMD~ 1 9.24e5 9.45e5 + SAMD11 ENSG~ sterile al
4 TMEM~ 1 1.51e7 1.52e7 - TMEM51~ ENSG~ TMEM51 ant
5 TMEM~ 1 1.52e7 1.52e7 + TMEM51 ENSG~ transmembr
6 FHAD1 1 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

• Once per machine (don’t include in script)
  • `install.packages("tidyverse")`

• Once per R session (DO include in script)
  • `library(tidyverse)`

```
-- Attaching packages ------- tidyverse 1.3.1 --
v ggplot2 3.3.3  v purrr 0.3.4
v tibble 3.1.2  v dplyr 1.0.6
v tidyr 1.1.3  v stringr 1.4.0
v readr 2.0.0  v forcats 0.5.1

-- 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_delim("file.txt")` -> data
    • `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
  `read_delim("O:/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

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

> read_delim("trumpton.txt") -> trumpton
Rows: 7 Columns: 5

-- Column specification -----------------------------
Delimiter: "\t"
chr (2): LastName, FirstName
dbl (3): Age, Weight, Height

> 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
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

- Compact
- Easy to read
- Shows linkage for genes

<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>

- No explicit genotype or replicate
- Values spread out over multiple rows and columns
- Not extensible to more metadata
### Long Format

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

<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>
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
  <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 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>       <int> <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

> filter(trumpton, Height>=170)
# A tibble: 3 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 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"

> filter(trumpton, FirstName == "Chris")

# A tibble: 2 x 5

  LastName FirstName Age Weight Height
     <chr>    <chr> <db>  <db>  <db>
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

```r
> transform.data
# A tibble: 10 x 3
   WT     KO difference
  <dbl> <dbl>      <dbl>
1  -5.11  -3.29       1.81
2   1.12  -1.85       -2.97
3  -3.99  -3.77       0.222
4  -4.18  -2.46       1.72
5  -1.93  -10.0       -8.10
6  -8.69  -2.38       6.31
7  -0.670  2.73        3.40
8  -1.15  -2.59       -1.43
9  -1.98  1.83        3.80
10 -1.06  0.372       1.43
```

```r
> filter(transform.data, difference > 5)
# A tibble: 1 x 3
   WT    KO difference
  <dbl> <dbl>      <dbl>
1  -8.69  -2.38      6.31
```

```r
> filter(transform.data, difference < -5)
# A tibble: 1 x 3
   WT    KO difference
  <dbl> <dbl>      <dbl>
1  -1.93  -10.0       -8.10
```

```r
> filter(transform.data, abs(difference) > 5)
# A tibble: 2 x 3
   WT     KO difference
  <dbl> <dbl>      <dbl>
1  -1.93  -10.0       -8.10
2  -8.69  -2.38        6.31
```
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!

> 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 $\rightarrow$ Filter 1 $\rightarrow$ Filter 2 $\rightarrow$ 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>   <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

> trumpton %>% select(-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
```
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
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))
```
Mappings can be quantitative or categorical
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_` function call
Our first plot...

```r
# 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    0.00  0.285 0.5
11 Sertad3  8.13  3.02  0.0001
12 Sertad1  7.69  4.34  0.01
```
Our second 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  0.285  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_line()
```
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

• Distribution Plots
  • geom_histogram
  • 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?

```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
```
Our first bar plot...

```r
expression %>%
  ggplot(aes(x=Gene, y=WT)) +
  geom_col()
```
Our second 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(x=values)) +
geom_histogram(binwidth = 0.1, fill="yellow", colour="black")
Plotting distributions - density

```r
> 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 -1.31  WT
8  0.337 WT
9  1.55  WT
10 1.86  KO

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

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

```r
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

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

eexpression %>%
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