Workshop 2:
The tidyverse and beyond

- we built this software on base R code

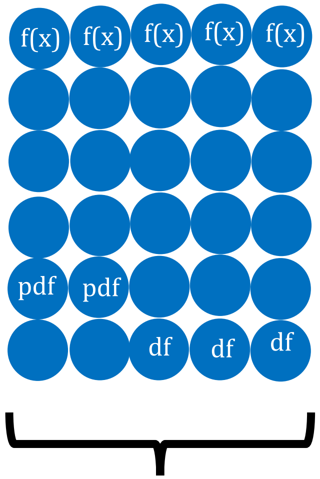
Brendan Palmer,
Statistics & Data Analysis Unit,
Clinical Research Facility - Cork

CRF-C
HRB Clinical Research Facility Cork

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

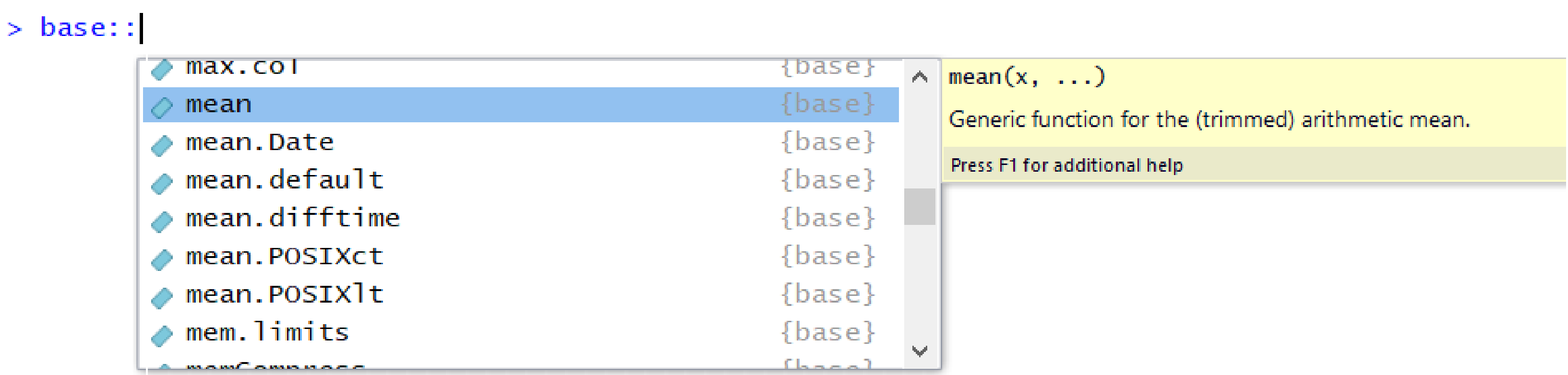
HRB
Health
Research
Board

Recall



Base R: Comes pre-loaded

Functions



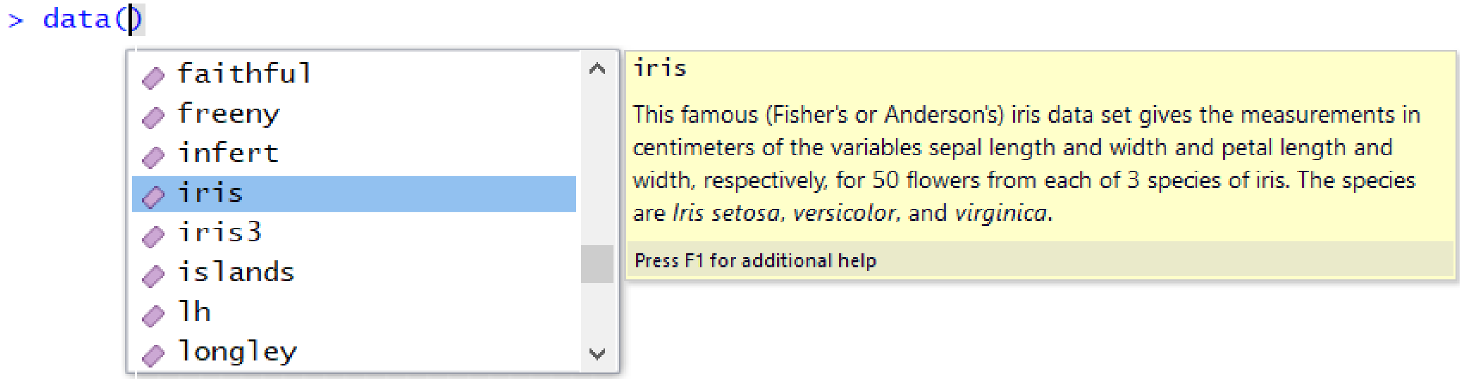
```
> base::
  max.col            {base}
  mean               {base}
  mean.Date          {base}
  mean.default       {base}
  mean.difftime      {base}
  mean.POSIXct       {base}
  mean.POSIXlt       {base}
  mem.limits         {base}
  memCompress        {base}
```

mean(x, ...)

Generic function for the (trimmed) arithmetic mean.

Press F1 for additional help

Data sets



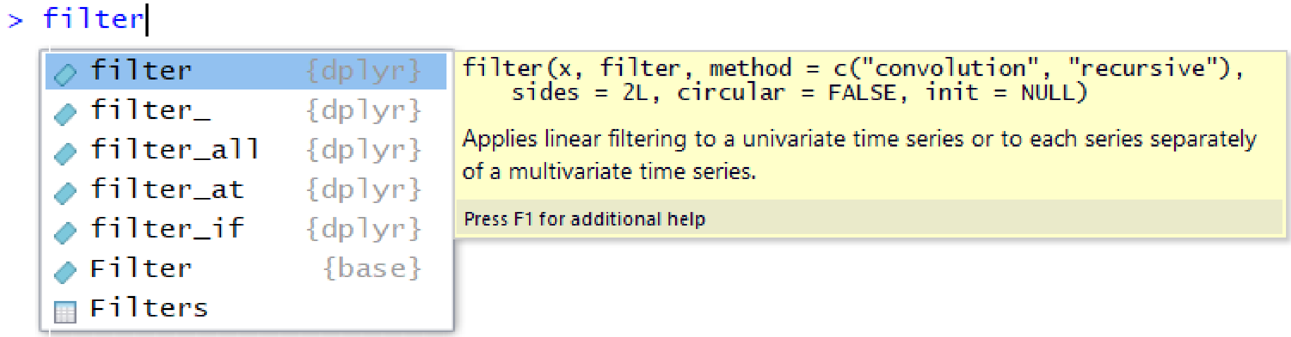
```
> data()
  faithful
  freeny
  infert
  iris
  iris3
  islands
  lh
  longley
```

iris

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.

Press F1 for additional help

Conflicts



```
> filter
  filter        {dplyr}
  filter_       {dplyr}
  filter_all    {dplyr}
  filter_at     {dplyr}
  filter_if     {dplyr}
  Filter         {base}
  Filters
```

```
filter(x, filter, method = c("convolution", "recursive"),
    sides = 2L, circular = FALSE, init = NULL)
```

Applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

Press F1 for additional help

Base R
Cheat Sheet

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.

`help.search(‘weighted mean’)`
Search the help files for a word or phrase.

`help(package = ‘dplyr’)`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object’s structure.

`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages(‘dplyr’)`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

Working Directory

`getwd()`
Find the current working directory (where inputs are found and outputs are sent).

`setwd(‘C://file/path’)`
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

`sort(x)`
Return x sorted.

`rev(x)`
Return x reversed.

`table(x)`
See counts of values.

`unique(x)`
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x[‘apple’]`	Element with name ‘apple’.

Programming

For Loop

```
for (variable in sequence){

   Do something

}
```

Example

```
for (i in 1:4){

   j <- i + 10

   print(j)

}
```

While Loop

```
while (condition){

   Do something

}
```

Example

```
while (i < 5){

   print(i)

   i <- i + 1

}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print(‘Yes’)
} else {
   print(‘No’)
}
```

Functions

```
function_name <- function(var){

   Do something

   return(new_variable)
}
```

Example

```
square <- function(x){

   squared <- x*x

   return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table(‘file.txt’)`	`write.table(df, ‘file.txt’)`	Read and write a delimited text file.
`df <- read.csv(‘file.csv’)`	`write.csv(df, ‘file.csv’)`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load(‘file.RData’)`	`save(df, file = ‘file.Rdata’)`	Read and write an R data file, a file type special for R.

Conditions								
	`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
	`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.

`as.factor`	`'1', '0', '1',`
	`levels: '1', '0'`
		Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Data Frames

Also see the **dplyr** package.

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x`   `df[[2]]`

Understanding a data frame

`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.

Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Dates

See the **lubridate** package.

RStudio® is a trademark of RStudio, Inc. • CC BY Mhairi McNeill • mhairihmcneill@gmail.com • 844-448-1212 • rstudio.com

Learn more at **web page or vignette** • package version • Updated: 3/15

Creating objects

For most of us, R is simply the creation of and manipulation of objects

```
new_object <- c(1, 2, 3)
```

- the objects are then fed into functions to create amazing new objects

```
amazing_new_object <- function(new_object)
```

Broadly speaking the following is true in R:

```
-  information

-  data frame  <- function(information)

-  plot        <- function(data frame)

-  model       <- function(data frame)
```

Data structures

Data structures encompass everything from our new object created in the last slide and beyond

Data can be stored in different forms

```
# double (for double precision floating point numbers)
typeof(1)

# character
typeof("string")

# logical
typeof(FALSE)

# missing values are represented by NA
example <- c(1, 2, NA, 4)
```

Other examples include integers and complex numbers

Types of data structures I

```
# Vectors:

These come in two forms
- A: Atomic vectors contain exactly one type of data

all_numbers         <- c(1, 2, 0.5, -0.5, 3.4)

all_characters      <- c("One", "too", "3")

all_logical         <- c(TRUE, FALSE) # NOTE: Always type it out

- B: Lists allow combinations of different types of data

this_is_a_list      <- list(1, TRUE, "Three", "4")

 typeof(this_is_a_list)
[1] "list"

this_is_also_a_list <- list(all_numbers, all_characters, all_logical)
```

Types of data structures I

```
# Matrices/Arrays:

- You can have a matrix of two or more dimensions
  a_matrix <- matrix(1:9, 3, 3)

- Vectors and matrices can only contain one type of data

- If you try to create a vector with more than one data type, then it will
  undergo coercion to the least common denominator

- The coercion rule goes:
                   logical -> integer -> numeric -> complex -> character

- You can perform coercions yourself on vectors
  nums_as_characters <- as.character(all_numbers)

  back_to_numbers    <- as.numeric(nums_as_characters)
```

Worksheet

ws2_script1_data_structures_I.R

Types of data structures II

```
# Data frames:
-  These are a special type of list
-  Observations are in rows
-  Variables are in columns
-  Labels or other metadata may also be present

- a_data_frame <- data.frame(number = 1:10,
                             char    = sample(letters, 10),
                             this_is_a_col_name  = rep(c(TRUE, FALSE), 5))

-  In the tidyverse data frames are called "tibbles"
      -  Tibbles are one of the unifying features of the tidyverse
      -  The two main differences are with;
         (a) printing: tibbles have a defined print method
         (b) subsetting: use of the [ function

-  Not all functions work with tibbles
-  To convert back it's just a matter of typing:
  old_data_frame <- as.data.frame(your_tbl_df)
```

Worksheet ws2_script2_data_structures_II

Indexing

- Indexing can occur in one or two dimensions

- One dimension:

```
new_object <- c(1, 2, 3)
new_object[1]
[1] 1
```

- Two dimensions

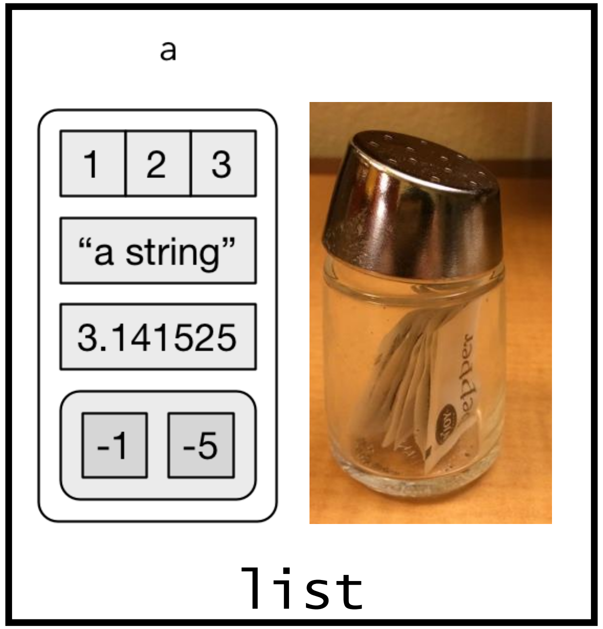
```
a_data_frame[1, 1]

a_data_frame$number[1]
```

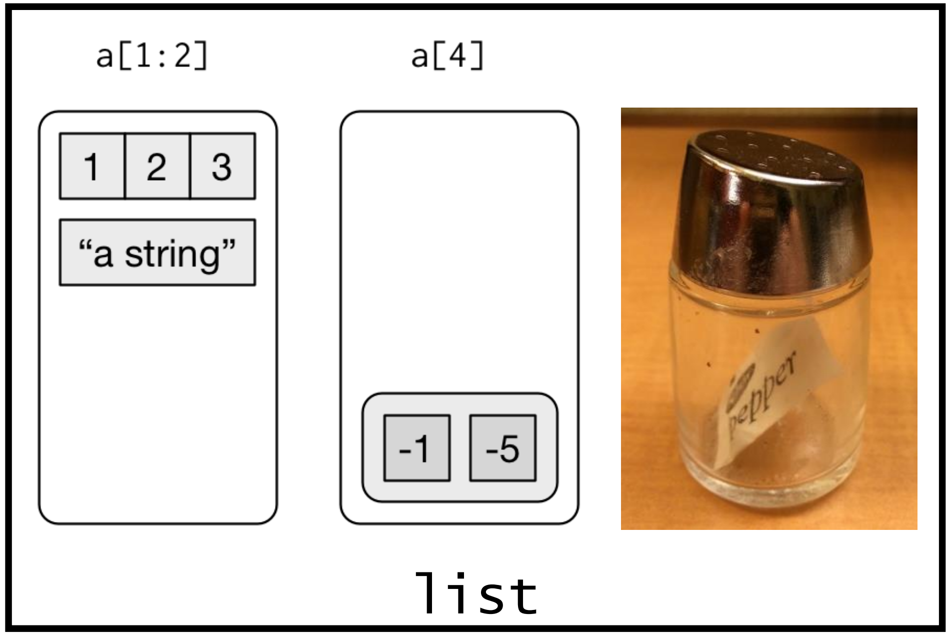
- In the tidyverse we don't use [much as dplyr::filter() and dplyr::select() allow you to solve the same problems

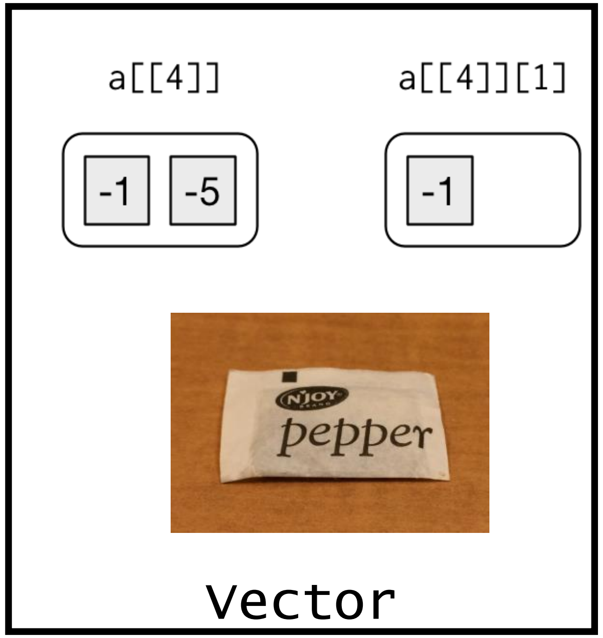
- However, given so much of the R has been written using these, it's worth recognising and understanding them

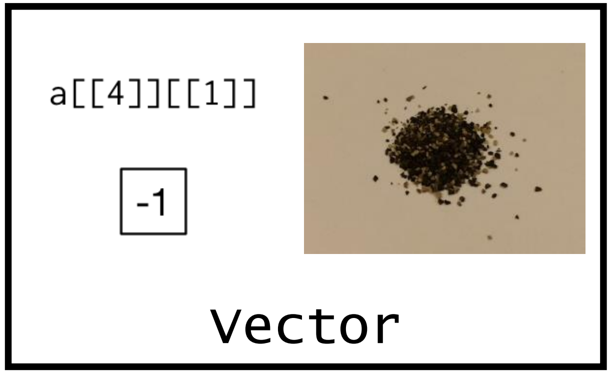
Indexing

```
# Recall
- this_is_also_a_list <- list(all_numbers, all_characters, all_logical)
```









```
# Important
-   [ extracts a sublist, results will be a list
-   [[ extracts a single component
-   $ is similar to [[ for named elements of a list
```

R for Data Science, Wickham and Grolemund

Worksheet

ws2_script3_indexing.R

Types of data structures III

# Factors

- In R, factors are used to work with categorical variables

- Historically they were easier to work with than characters, hence many baseR functions automatically convert characters to factors

- This does not happen in the tidyverse

- The forcats packages is designed to handle factors in the tidyverse

- One of the most important uses of factors is in statistical modeling; since categorical variables enter into statistical models differently than continuous variables, storing data as factors insures that the modeling functions will treat such data correctly

Worksheet ws2_script4_factors.R

SPOOKY STORY TIME

Once upon a time……

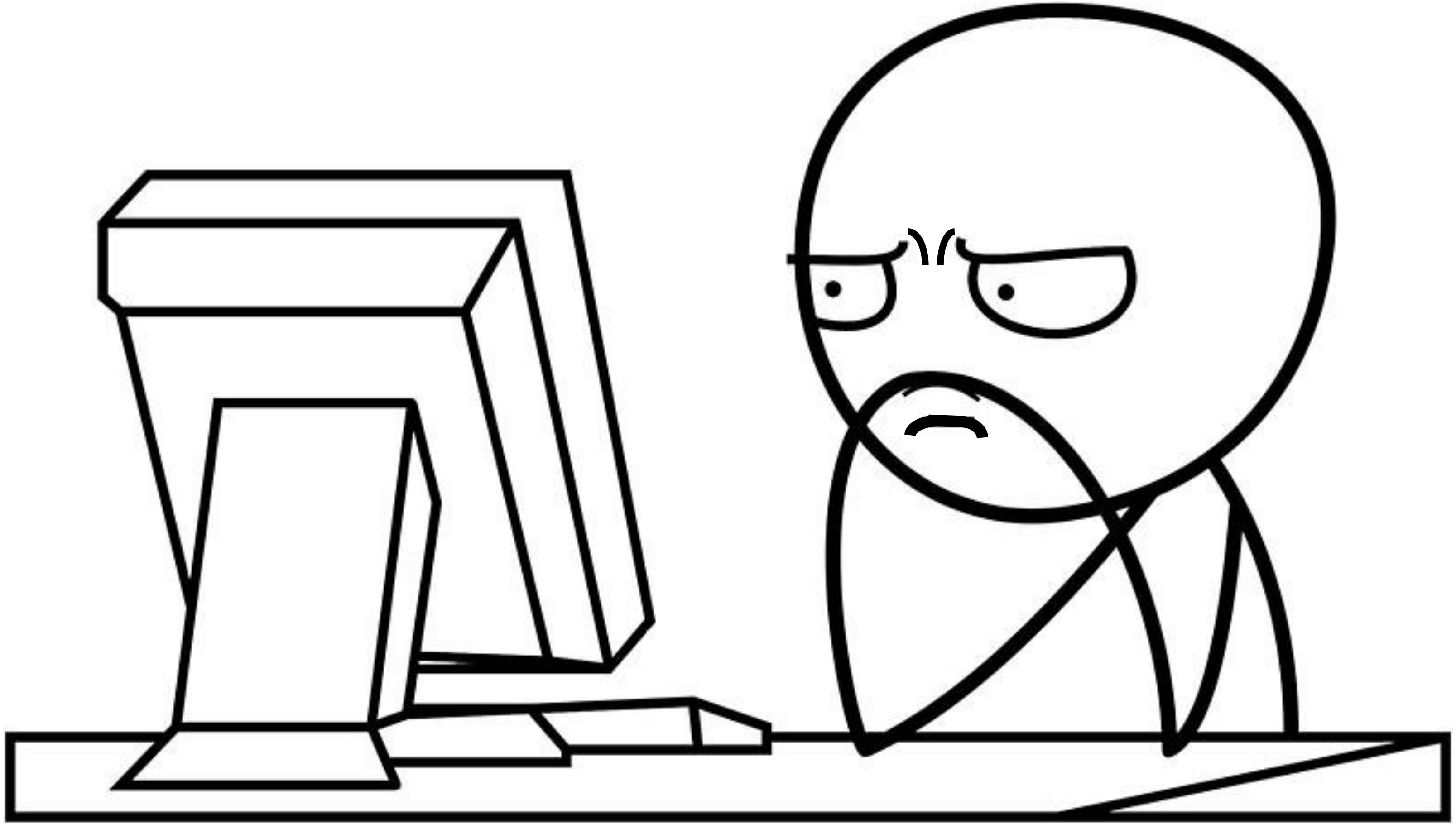
A Researcher collected some data

The Researcher was very good

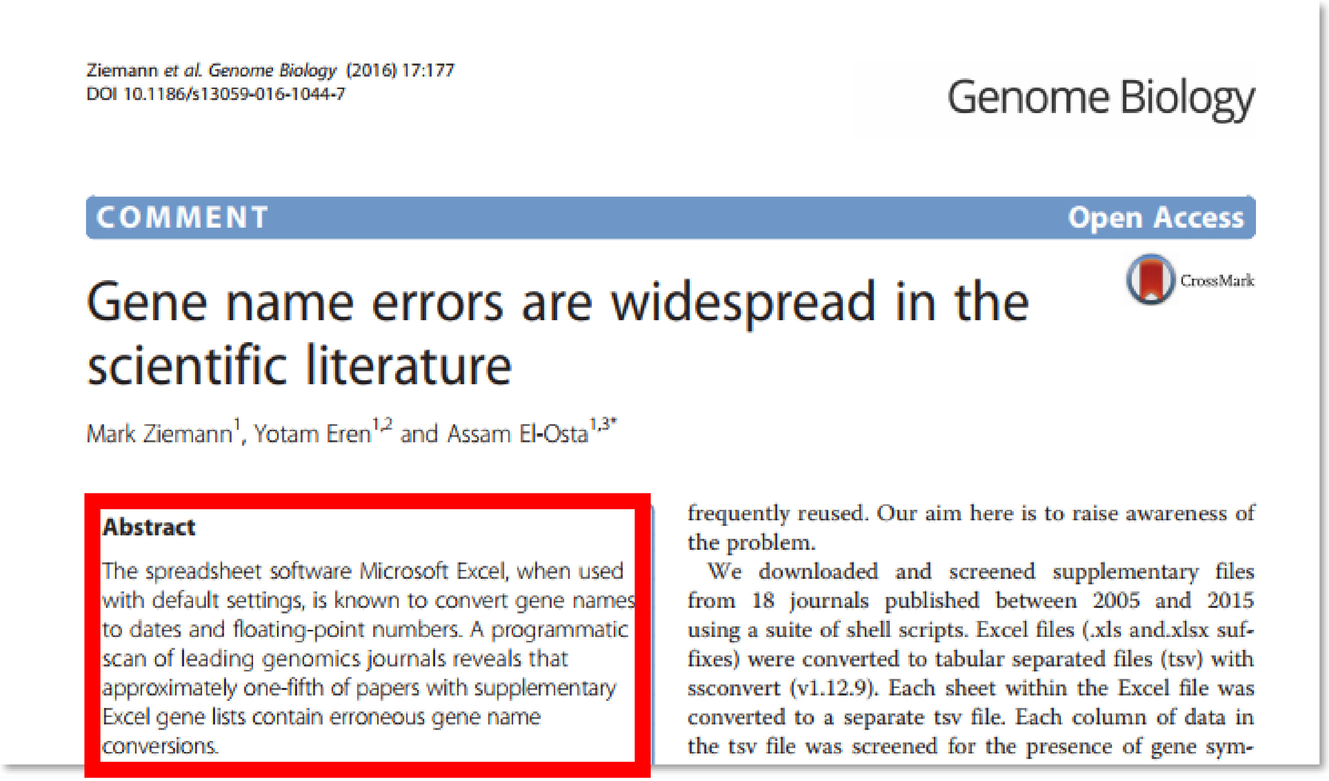
The data was very good

But then……………………

…….. The Researcher inputted that data into Excel using default settings



Then Excel converted some gene names to dates



Ziemann *et al. Genome Biology* (2016) 17:177
DOI 10.1186/s13059-016-1044-7

Genome Biology

COMMENT Open Access

CrossMark

Gene name errors are widespread in the scientific literature

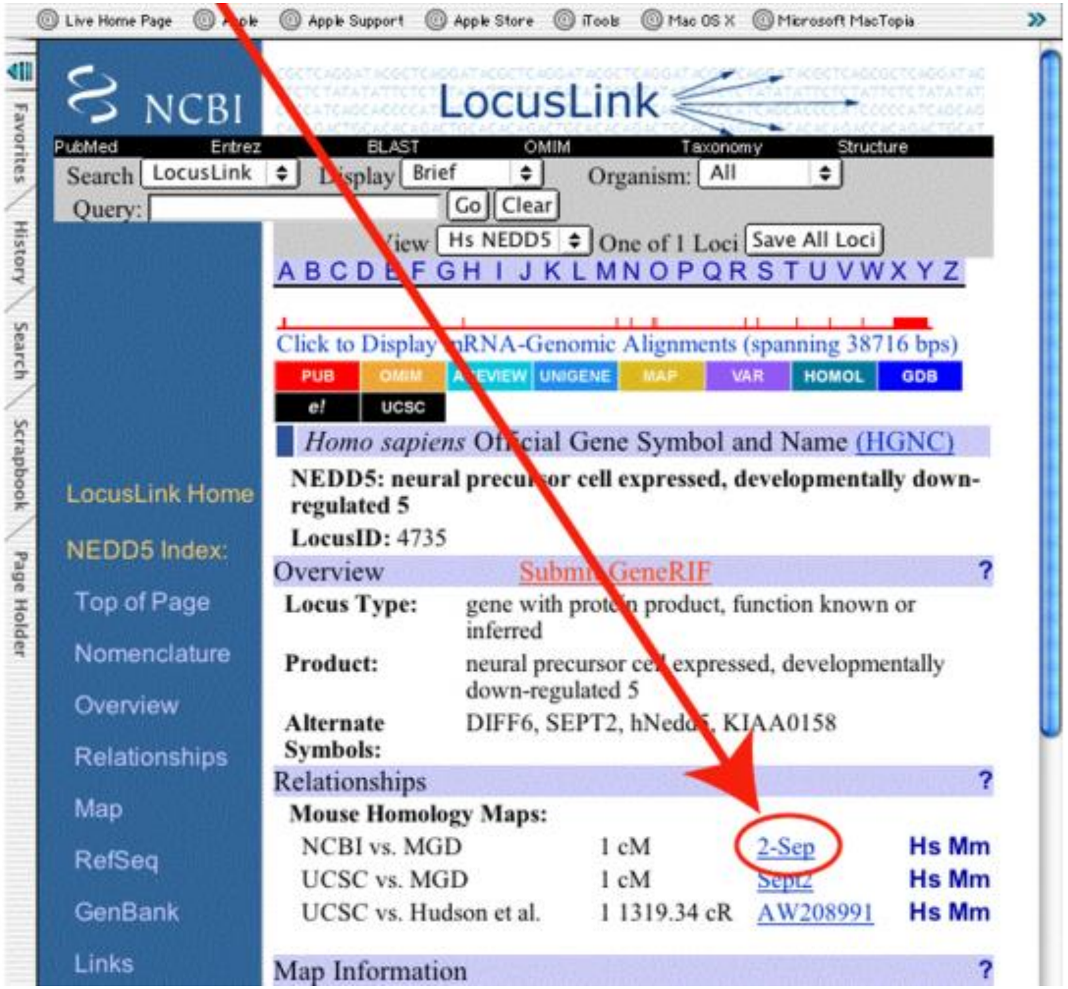
Mark Ziemann[1], Yotam Eren[1,2] and Assam El-Osta[1,3*]

Abstract

The spreadsheet software Microsoft Excel, when used with default settings, is known to convert gene names to dates and floating-point numbers. A programmatic scan of leading genomics journals reveals that approximately one-fifth of papers with supplementary Excel gene lists contain erroneous gene name conversions.

frequently reused. Our aim here is to raise awareness of the problem.

We downloaded and screened supplementary files from 18 journals published between 2005 and 2015 using a suite of shell scripts. Excel files (.xls and.xlsx suffixes) were converted to tabular separated files (tsv) with ssconvert (v1.12.9). Each sheet within the Excel file was converted to a separate tsv file. Each column of data in the tsv file was screened for the presence of gene sym-



Search LocusLink Display Brief Organism: All

Query:

View Hs NEDD5 One of 1 Loci

Click to Display mRNA-Genomic Alignments (spanning 38716 bps)

Homo sapiens Official Gene Symbol and Name (HGNC)

NEDD5: neural precursor cell expressed, developmentally down-regulated 5

LocusID: 4735

Overview Submit GeneRIF ?

Locus Type:	gene with protein product, function known or inferred
Product:	neural precursor cell expressed, developmentally down-regulated 5
Alternate Symbols:	DIFF6, SEPT2, hNedd5, KIAA0158

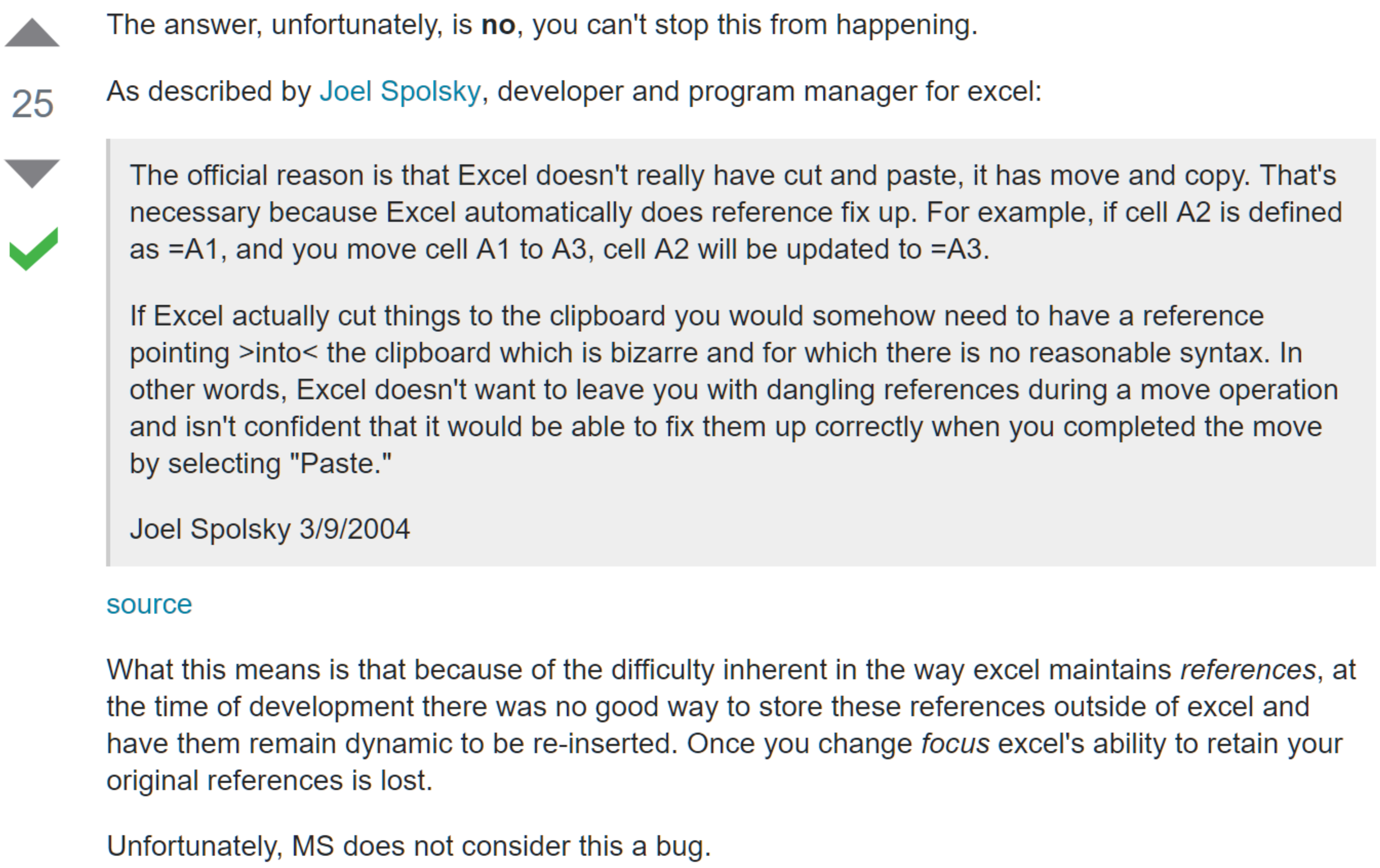
Relationships ?

Mouse Homology Maps:

NCBI vs. MGD	1 cM	2-Sep	Hs Mm
UCSC vs. MGD	1 cM	Sept2	Hs Mm
UCSC vs. Hudson et al.	1 1319.34 cR	AW208991	Hs Mm

Map Information ?

Excel also frequently got clipboard amnesia



The answer, unfortunately, is **no**, you can't stop this from happening.

As described by Joel Spolsky, developer and program manager for excel:

> The official reason is that Excel doesn't really have cut and paste, it has move and copy. That's necessary because Excel automatically does reference fix up. For example, if cell A2 is defined as =A1, and you move cell A1 to A3, cell A2 will be updated to =A3.
>
> If Excel actually cut things to the clipboard you would somehow need to have a reference pointing >into< the clipboard which is bizarre and for which there is no reasonable syntax. In other words, Excel doesn't want to leave you with dangling references during a move operation and isn't confident that it would be able to fix them up correctly when you completed the move by selecting "Paste."
>
> Joel Spolsky 3/9/2004

source

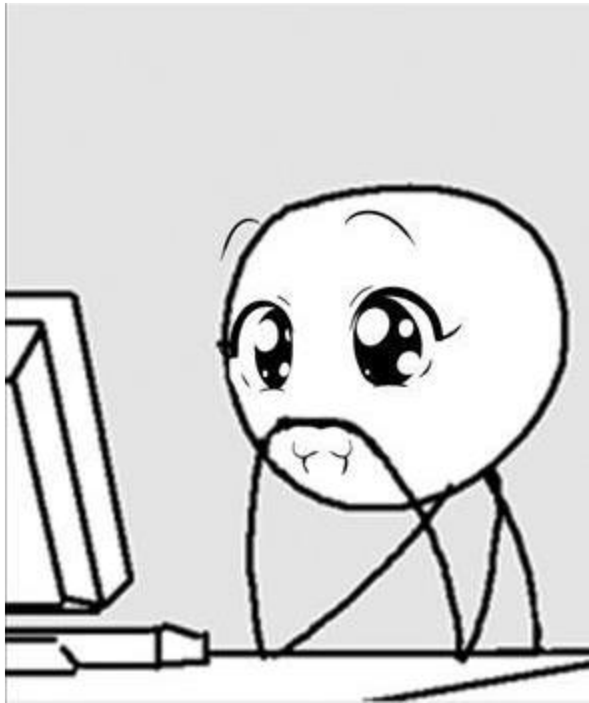
What this means is that because of the difficulty inherent in the way excel maintains *references*, at the time of development there was no good way to store these references outside of excel and have them remain dynamic to be re-inserted. Once you change *focus* excel's ability to retain your original references is lost.

Unfortunately, MS does not consider this a bug.

Then one day…..

The Researcher accidently used sort without selecting all the columns

and inadvertently randomised the data……



Help is at hand: R for Excel Users http://rex-analytics.com/things-live-r-r-excel-users/

Importing data with readr

- Used to read plain text rectangular files into R (e.g. csv)

- read_csv is the equivalent of read.csv in base R

- readr has a number of advantages over base R import function
    - ~10X faster
    - produces tibbles
    - doesn’t convert character vectors to factors
    - more reproducible (readr code on your computer is likely to work on another computer)

Data Import

with readr, tibble, and tidyr

Cheat Sheet

R's **tidyverse** is built around **tidy data** stored in **tibbles**, an enhanced version of a data frame.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

Other types of data

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Write functions

Save **x**, an R object, to **path**, a file path, with:

write_csv(x, path, na = "NA", append = FALSE, col_names = !append)
Tibble/df to comma delimited file.

write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append)
Tibble/df to file with any delimiter.

write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append)
Tibble/df to a CSV for excel

write_file(x, path, append = FALSE)
String to file.

write_lines(x, path, na = "NA", append = FALSE)
String vector to file, one element per line.

write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)
Object to RDS file.

write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)
Tibble/df to tab delimited files.

Read functions

Read tabular data to tibbles

These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive())

```
a,b,c
1,2,3
4,5,NA
```

→

A	B	C
1	2	3
4	5	NA

read_csv()
Reads comma delimited files.
read_csv("file.csv")

```
a;b;c
1;2;3
4;5;NA
```

→

A	B	C
1	2	3
4	5	NA

read_csv2()
Reads Semi-colon delimited files.
read_csv2("file2.csv")

```
a|b|c
1|2|3
4|5|NA
```

→

A	B	C
1	2	3
4	5	NA

read_delim(delim, quote = "\"", escape_backslash = FALSE, escape_double = TRUE) Reads files with any delimiter.
read_delim("file.txt", delim = "|")

```
a b c
1 2 3
4 5 NA
```

→

A	B	C
1	2	3
4	5	NA

read_fwf(col_positions)
Reads fixed width files.
read_fwf("file.fwf", col_positions = c(1, 3, 5))

read_tsv()
Reads tab delimited files. Also **read_table()**.
read_tsv("file.tsv")

Useful arguments

```
a,b,c
1,2,3
4,5,NA
```

Example file
write_csv (path = "file.csv", x = read_csv("a,b,c\n1,2,3\n4,5,NA"))

A	B	C
1	2	3
4	5	NA

No header
*read_csv("file.csv", **col_names = FALSE**)*

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
*read_csv("file.csv", **col_names = c("x", "y", "z")**)*

1	2	3
4	5	NA

Skip lines
*read_csv("file.csv", **skip = 1**)*

A	B	C
1	2	3

Read in a subset
*read_csv("file.csv", **n_max = 1**)*

A	B	C
1	2	3
NA	NA	NA

Missing Values
*read_csv("file.csv", **na = c("4", "5", ".")**)*

Read non-tabular data

read_file(file, locale = default_locale())
Read a file into a single string.

read_file_raw(file)
Read a file into a raw vector.

read_lines(file, skip = 0, n_max = -1L, locale = default_locale(), na = character(), progress = interactive())
Read each line into its own string.

read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive())
Read each line into a raw vector.

read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())
Apache style log files.

Parsing data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer

sex is a character

earn is a double (numeric)

1. Use **problems()** to diagnose problems
   x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing
   - **col_guess()** - the default
   - **col_character()**
   - **col_double()**
   - **col_euro_double()**
   - **col_datetime**(format = "") Also **col_date**(format = "") and **col_time**(format = "")
   - **col_factor**(levels, ordered = FALSE)
   - **col_integer()**
   - **col_logical()**
   - **col_number()**
   - **col_numeric()**
   - **col_skip()**

   x <- read_csv("file.csv", col_types = cols(
   A = col_double(),
   B = col_logical(),
   C = col_factor()
   ))

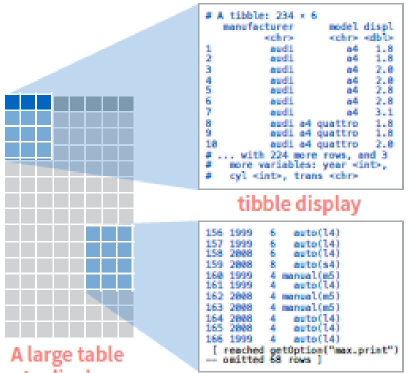
3. Else, read in as character vectors then parse with a parse_ function.
   - **parse_guess**(x, na = c("", "NA"), locale = default_locale())
   - **parse_character**(x, na = c("", "NA"), locale = default_locale())
   - **parse_datetime**(x, format = "", na = c("", "NA"), locale = default_locale()) Also **parse_date()** and **parse_time()**
   - **parse_double**(x, na = c("", "NA"), locale = default_locale())
   - **parse_factor**(x, levels, ordered = FALSE, na = c("", "NA"), locale = default_locale())
   - **parse_integer**(x, na = c("", "NA"), locale = default_locale())
   - **parse_logical**(x, na = c("", "NA"), locale = default_locale())
   - **parse_number**(x, na = c("", "NA"), locale = default_locale())

   x$A <- parse_number(x$A)

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve two behaviors:

- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen.
- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting



```
# A tibble: 234 × 6
   manufacturer       model displ
          <chr>       <chr> <dbl>
1          audi          a4   1.8
2          audi          a4   1.8
3          audi          a4   2.0
4          audi          a4   2.0
5          audi          a4   2.8
6          audi          a4   2.8
7          audi          a4   3.1
8          audi a4 quattro   1.8
9          audi a4 quattro   1.8
10         audi a4 quattro   2.0
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

tibble display

```
156 1999   6   auto(l4)
157 1999   6   auto(l4)
158 2008   6   auto(l4)
159 2008   8   auto(s4)
160 1999   4 manual(m5)
161 1999   4   auto(l4)
162 2008   4 manual(m5)
163 2008   4 manual(m5)
164 2008   4   auto(l4)
165 2008   4   auto(l4)
166 1999   4   auto(l4)
 [ reached getOption("max.print")
 -- omitted 68 rows ]
```

A large table to display

data frame display

- Control the default appearance with options:

  options(tibble.print_max = n,
  tibble.print_min = m, tibble.width = Inf)

- View entire data set with **View(**x, title) or **glimpse(**x, width = NULL, …)

- Revert to data frame with **as.data.frame()** (required for some older packages)

Construct a tibble in two ways

tibble(…)
Construct by columns.
tibble(x = 1:3,
y = c("a", "b", "c"))

Both make this tibble

```
A tibble: 3 x 2
      x     y
  <int> <dbl>
1     1     a
2     2     b
3     3     c
```

tribble(…)
Construct by rows.
tribble(
~x, ~y,
1, "a",
2, "b",
3, "c")

as_tibble(x, …) Convert data frame to tibble.

enframe(x, name = "name", value = "value")
Converts named vector to a tibble with a names column and a values column.

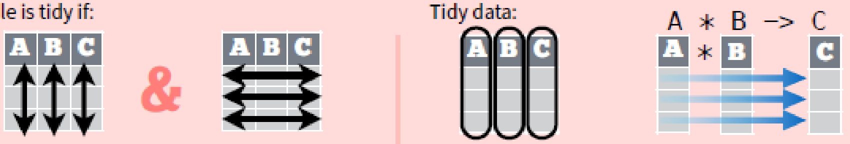
is_tibble(x) Test whether x is a tibble.

Tidy Data with tidyr

Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

Tidy data:



Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Makes variables easy to access as vectors

Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout. Each uses the idea of a key column: value column pair.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE)

Gather moves column names into a key column, gathering the column values into a single value column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL)

Spread moves the unique values of a key column into the column names, spreading the values of a value column across the new columns that result.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...)
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up"))
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...)
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x,list(x2 = 2), x2)

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list())
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...)
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split and Combine Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate,
into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE)

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

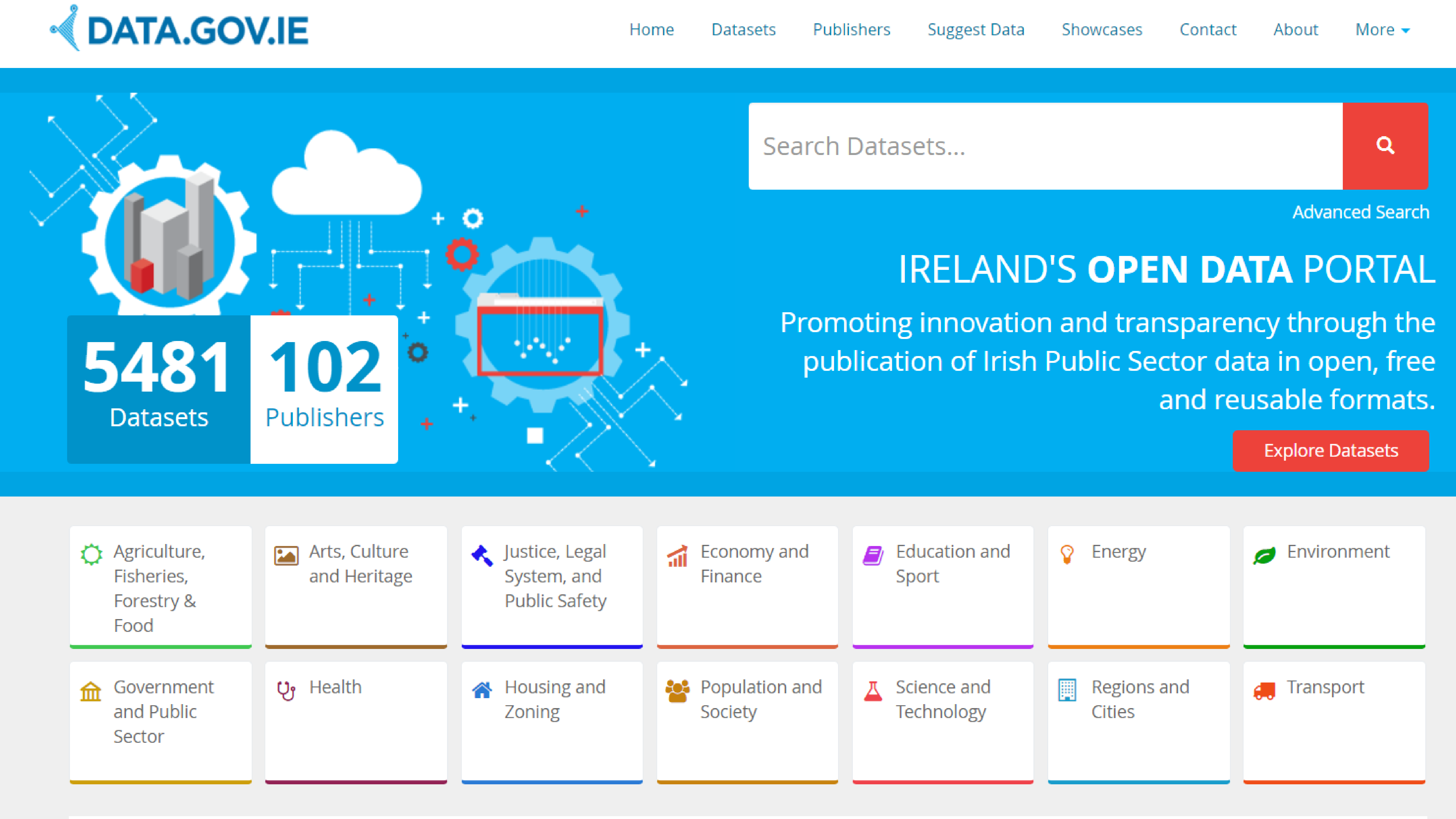
unite(table5, century, year,
col = "year", sep = "")

Importing data from stata and other stats packages

- Stata, SPSS and SAS file formats can be imported **and** exported using the tidyverse package "haven"

Importing data with other packages

- Very often the data you handle may not be your own
    - collaborators
    - government
    - online databases
    - publications
    - webpages



IRELAND'S **OPEN DATA** PORTAL

Promoting innovation and transparency through the publication of Irish Public Sector data in open, free and reusable formats.

Worksheet

ws2_script5_data_import.R

Introductory R Workshops

~~Week 1 (13th February):~~
~~Take a parachute and jump (into the tidyverse)~~

~~- tidying and visualisation of NGS data using sample R scripts~~

~~Week 2 (20th February):~~
~~We built this software on base R code~~

~~- overview and structure of R syntax~~

Week 3 (27th February):
Sending an SOS to the world

- how to identify with errors in your code and get help