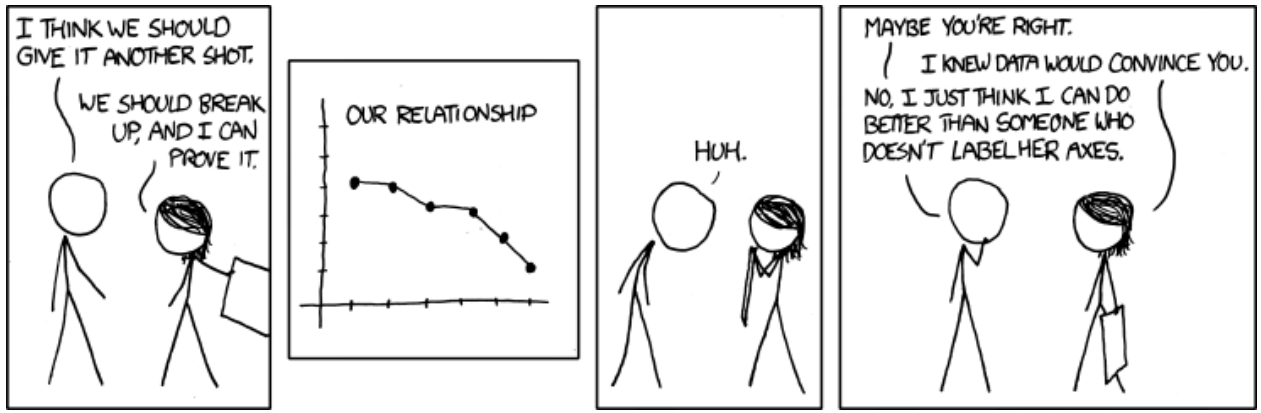
Q-Step Smart Skills: R Refresher

Lorien Jasny[1]

[1]Q-Step Centre, Exeter University
l.jasny@exeter.ac.uk



Understanding Code

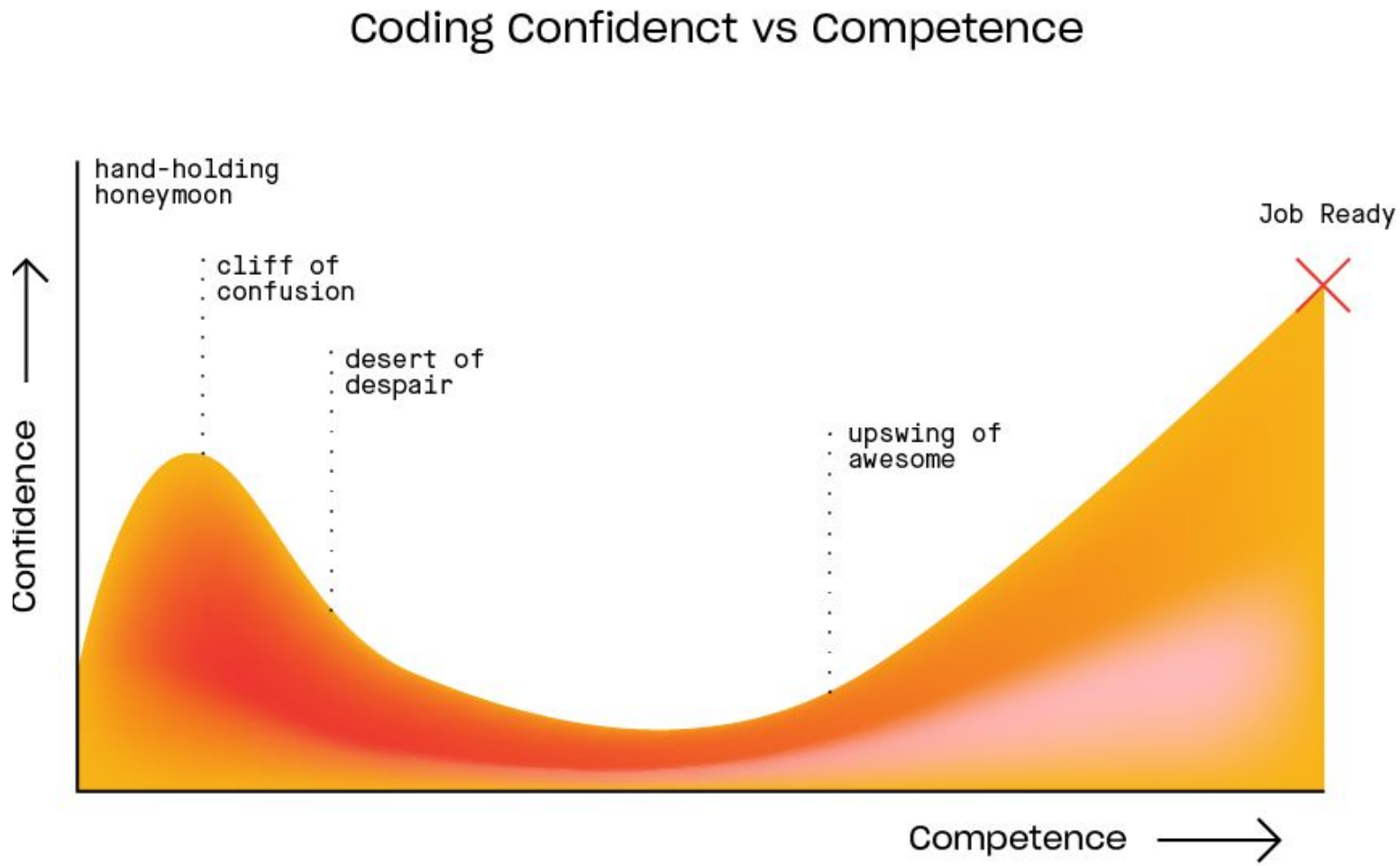
- Writing code is the practice of taking something we want our computer to do and translating it into a language the computer can understand

Understanding Code

- Writing code is the practice of taking something we want our computer to do and translating it into a language the computer can understand
- Therefore learning to code is learning a language

Understanding Code

- Writing code is the practice of taking something we want our computer to do and translating it into a language the computer can understand
- Therefore learning to code is learning a language
- Going back to basics to refresh our knowledge and un-learn some bad habits


Coding Confidenct vs Competence
hand-holding honeymoon
cliff of confusion
desert of despair
upswing of awesome
Job Ready
Confidence
Competence

The R Environment

The R Environment

- **R** is both a language and a software platform

The R Environment

- **R** is both a language and a software platform
- **R** software is open-source, cross-platform, and free

The R Environment

- **R** is both a language and a software platform
- **R** software is open-source, cross-platform, and free
- Its home on the web: http://www.r-project.org

Basics

Basics

- **R** uses a comand-like environment, like stata or sas

Basics

- **R** uses a comand-like environment, like stata or sas
- **R** is highly extendible

Basics

- **R** uses a comand-like environment, like stata or sas
- **R** is highly extendible
  - You can write your own custom functions

Basics

- **R** uses a comand-like environment, like stata or sas
- **R** is highly extendible
  - You can write your own custom functions
  - There are 8000 free add-on packages

Basics

- **R** uses a comand-like environment, like stata or sas
- **R** is highly extendible
  - You can write your own custom functions
  - There are 8000 free add-on packages
- Generally good at reading in/writing out other file formats

Basics

- **R** uses a comand-like environment, like stata or sas
- **R** is highly extendible
  - You can write your own custom functions
  - There are 8000 free add-on packages
- Generally good at reading in/writing out other file formats
- Everything in **R** is an object – data, functions, everything

Fundamentals

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*
  - **R** tries to interpret what you've asked it to do (evaluation)

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*
  - **R** tries to interpret what you've asked it to do (evaluation)
  - If it understands what you've written, it does it (execution)

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*
  - **R** tries to interpret what you've asked it to do (evaluation)
  - If it understands what you've written, it does it (execution)
  - If it doesn't, it will likely give you an error or a warning

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*
  - **R** tries to interpret what you've asked it to do (evaluation)
  - If it understands what you've written, it does it (execution)
  - If it doesn't, it will likely give you an error or a warning
- Some commands trigger **R** to print to the screen, others don't

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*
  - **R** tries to interpret what you've asked it to do (evaluation)
  - If it understands what you've written, it does it (execution)
  - If it doesn't, it will likely give you an error or a warning
- Some commands trigger **R** to print to the screen, others don't
- If you type an incomplete command, **R** will usually respond by changing the command prompt to the '+' character

Fundamentals

- When you type commands at the prompt '>' and hit *ENTER*
  - **R** tries to interpret what you've asked it to do (evaluation)
  - If it understands what you've written, it does it (execution)
  - If it doesn't, it will likely give you an error or a warning
- Some commands trigger **R** to print to the screen, others don't
- If you type an incomplete command, **R** will usually respond by changing the command prompt to the '+' character
  - Hit *ESC* on a Mac to cancel
  - Type in *Ctrl* + *C* on Windows and Linux to cancel

Fundamentals

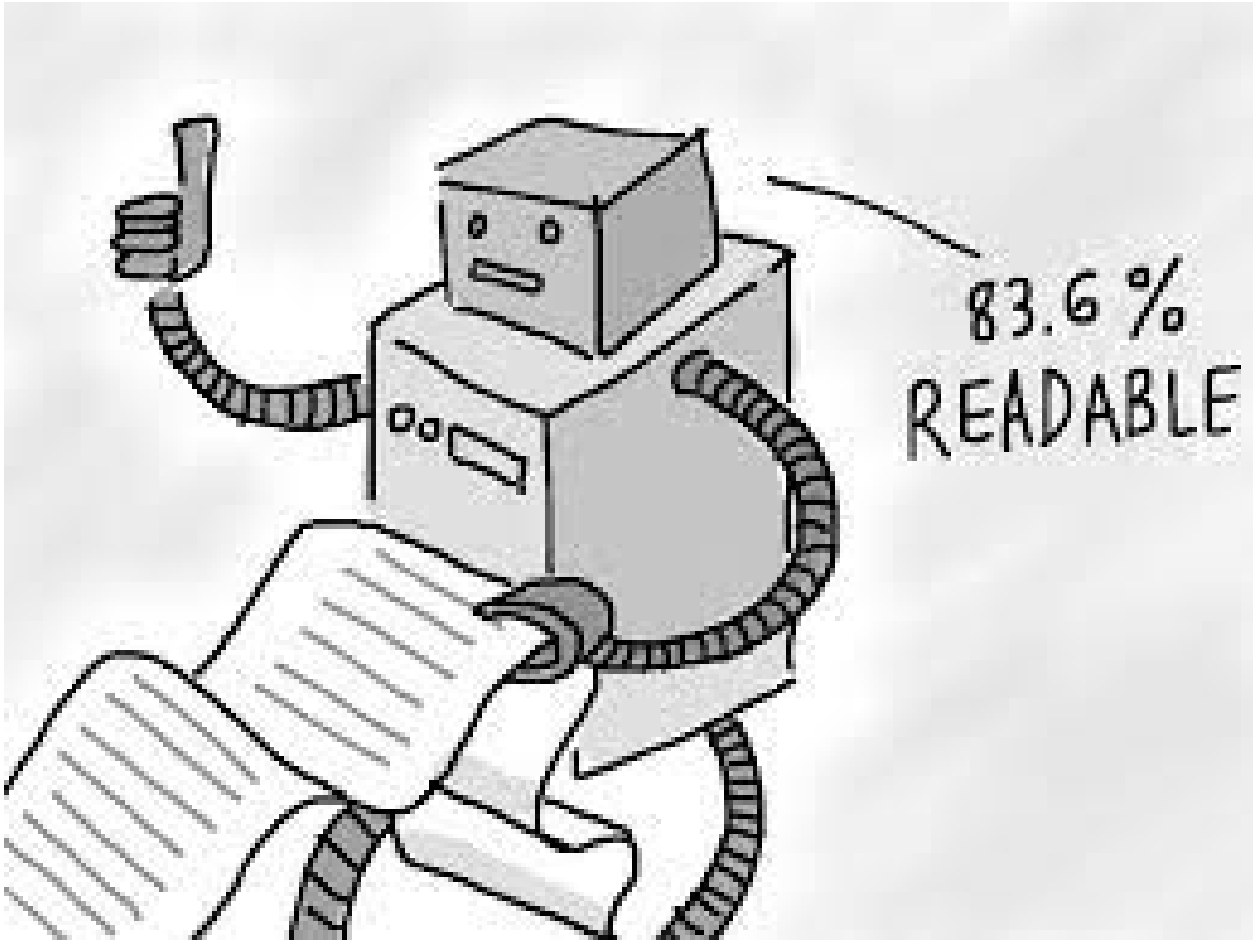
- There are a series of symbols that mean specific things in the R language (called operators)

Fundamentals

- There are a series of symbols that mean specific things in the R language (called operators)
- You should eventually be able to read them like words

$<-$	The 'assign' operator
#	Make a comment
?	Ask for help
==	Left is exactly equal to right
!=	Left is not equal to right
&	AND - returns a TRUE/FALSE if both sides of the argument are true
|	OR - returns a TRUE/FALSE if either side of the argument is true

Making your R code readable



Making your code readable

- The Art of Readable Code [@boswell2011], available as a PDF here, gives general principles

Making your code readable

- The Art of Readable Code [@boswell2011], available as a PDF here, gives general principles
- "Code should be written to minimize the time it would take for someone else to understand it" (2011:3) – even when (or especially if) that someone else is future you

Making your code readable

- The Art of Readable Code [@boswell2011], available as a PDF here, gives general principles
- "Code should be written to minimize the time it would take for someone else to understand it" (2011:3) – even when (or especially if) that someone else is future you
- Use the code that is easiest to understand, utilising comments and sectioning, and giving objects meaningful names

Making your code readable

- Comments

Making your code readable

- Comments
  - Write all your comments first and then fill in the code

Making your code readable

- Comments
  - Write all your comments first and then fill in the code
- Spacing

Making your code readable

- Comments
  - Write all your comments first and then fill in the code
- Spacing
  - Some languages restrict it, R doesn't

Making your code readable

- Comments
  - Write all your comments first and then fill in the code
- Spacing
  - Some languages restrict it, R doesn't
  - Break things up into smallest pieces (I'll return to this idea)

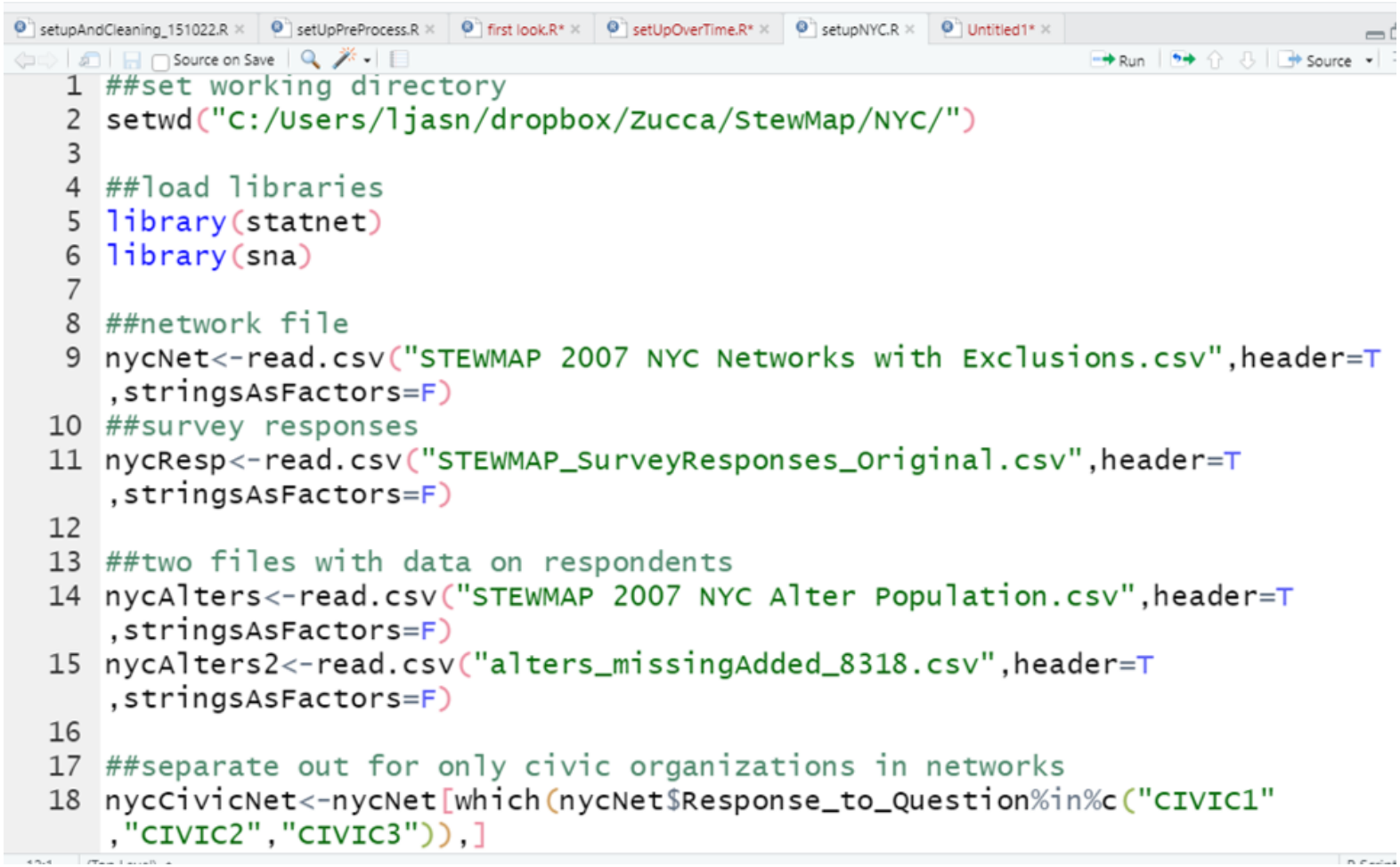
Making your code readable

- Comments
  - Write all your comments first and then fill in the code
- Spacing
  - Some languages restrict it, R doesn't
  - Break things up into smallest pieces (I'll return to this idea)
- Naming convention

Making your code readable

- Comments
  - Write all your comments first and then fill in the code
- Spacing
  - Some languages restrict it, R doesn't
  - Break things up into smallest pieces (I'll return to this idea)
- Naming convention
  - If you look at my dissertation code, I have items called `crap1, crap2, crap3` ...I've gotten better!

Making your code readable



Making your code readable

What can I do better?

Making your code readable

What can I do better?

- better file names

Making your code readable

What can I do better?

- better file names
- more comments - what's in the files, what are the two additional files??

Making your code readable

What can I do better?

- better file names
- more comments - what's in the files, what are the two additional files??
- Whitespace isn't bad

Comments

There are different types of comments that are useful to include in your scripts. Try to think from the reader's perspective: what would they want to and need to know? You might want to use 'director' style commenting, where you include:

Comments

There are different types of comments that are useful to include in your scripts. Try to think from the reader's perspective: what would they want to and need to know? You might want to use 'director' style commenting, where you include:

- The aim of what you're doing (or trying to do)

Comments

There are different types of comments that are useful to include in your scripts. Try to think from the reader's perspective: what would they want to and need to know? You might want to use 'director' style commenting, where you include:

- The aim of what you're doing (or trying to do)
- How you're doing it (sometimes including why you didn't do it another way)

Comments

There are different types of comments that are useful to include in your scripts. Try to think from the reader's perspective: what would they want to and need to know? You might want to use 'director' style commenting, where you include:

- The aim of what you're doing (or trying to do)
- How you're doing it (sometimes including why you didn't do it another way)
- Notes on anything you don't understand, or didn't work how you thought it would

Comments

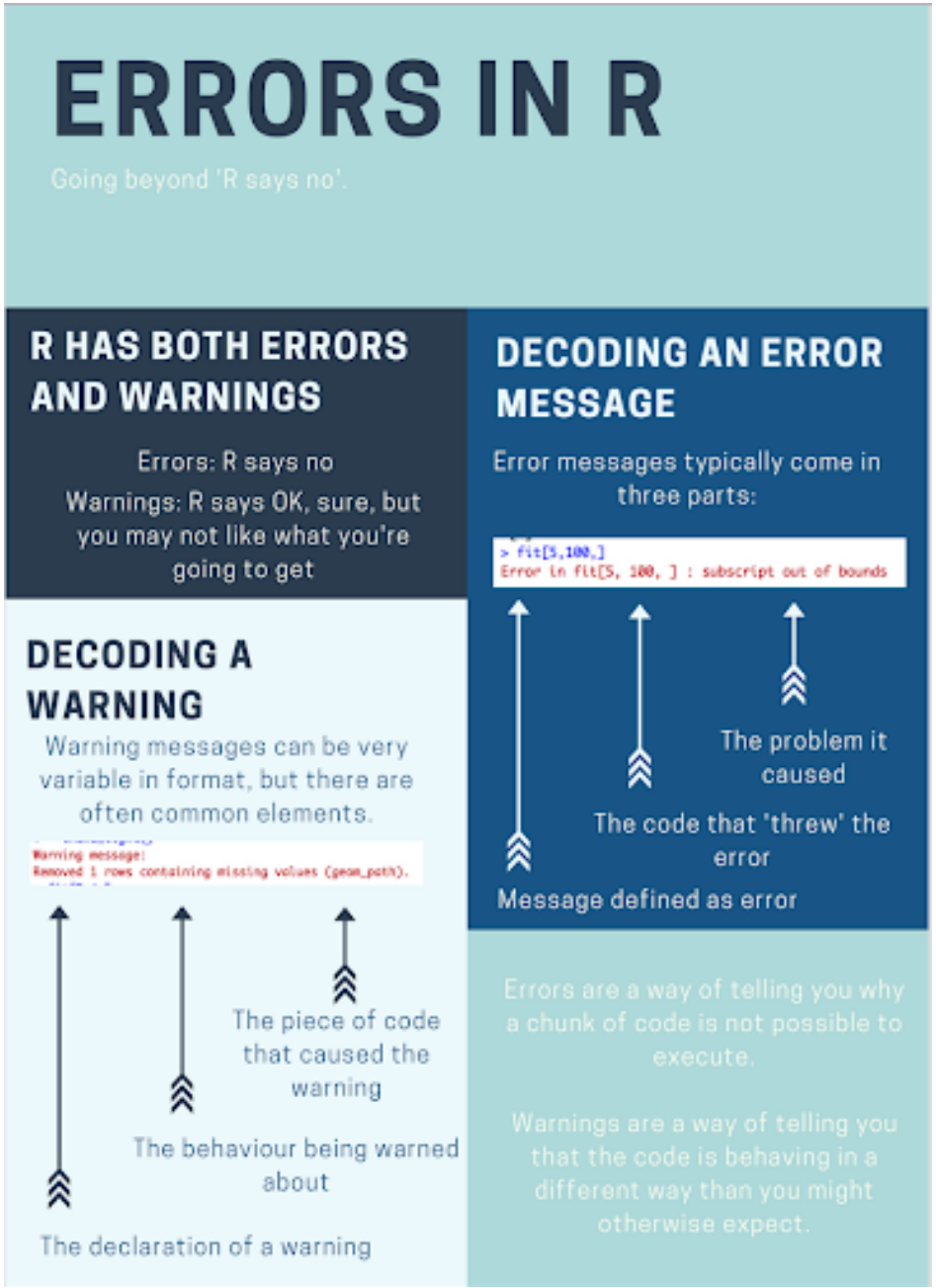
There are different types of comments that are useful to include in your scripts. Try to think from the reader's perspective: what would they want to and need to know? You might want to use 'director' style commenting, where you include:

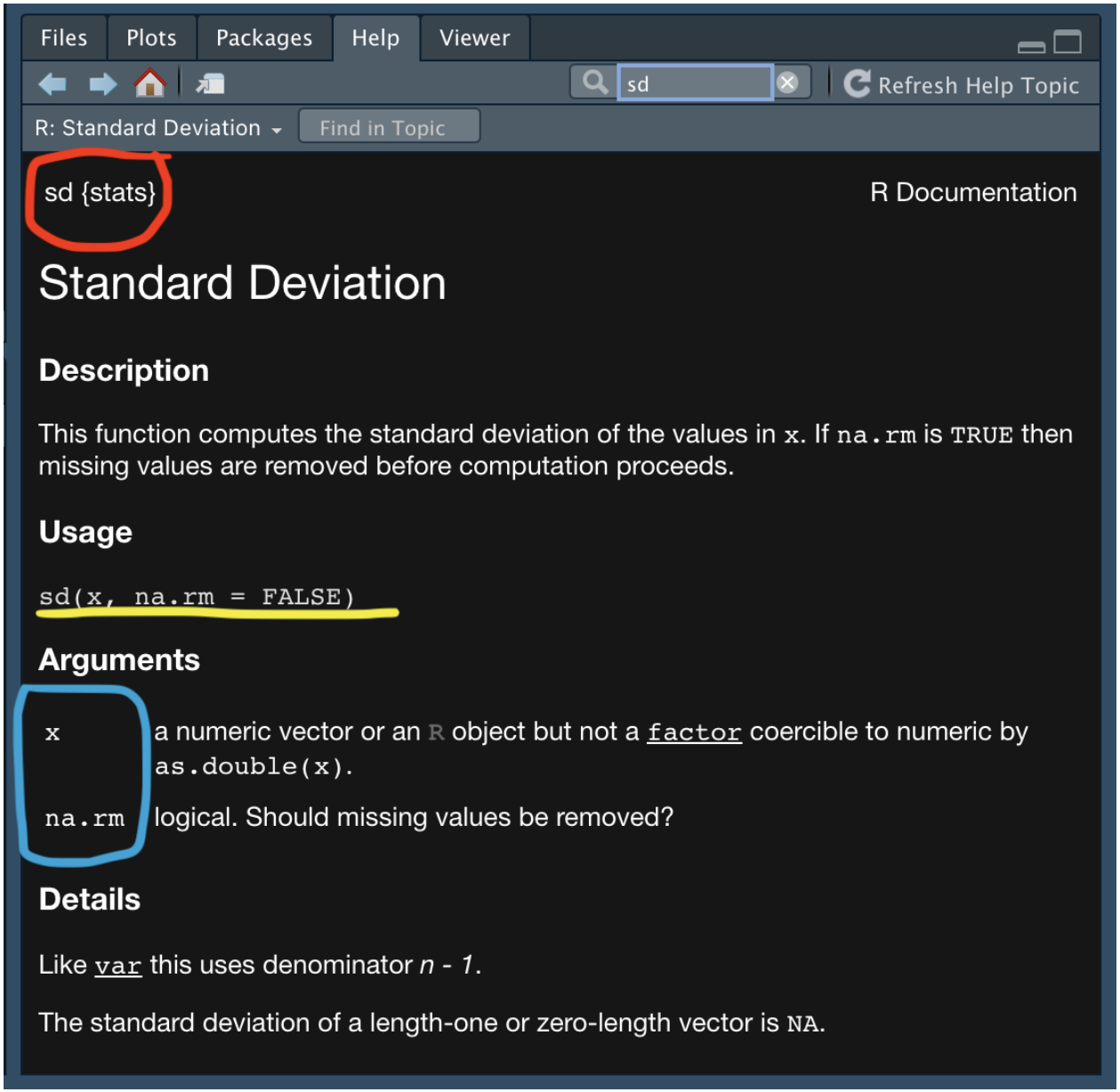
- The aim of what you're doing (or trying to do)
- How you're doing it (sometimes including why you didn't do it another way)
- Notes on anything you don't understand, or didn't work how you thought it would
- The results of the code (i.e. paraphrasing the output)

Naming Conventions

- Use sensible names for your objects
- This includes making sure the style of the naming is easy to read
- Only use letters, numbers, underscores and full stops

```
some.people <- "code like this"

other_people <- "prefer this"

WhoAreThesePeople <- "don't be them"

thesepeoplearetheworst <- "really don't be them"
```


ERRORS IN R
Going beyond 'R says no'.
R HAS BOTH ERRORS AND WARNINGS
Errors: R says no
Warnings: R says OK, sure, but you may not like what you're going to get
DECODING AN ERROR MESSAGE
Error messages typically come in three parts:
> fit[5,100,]
Error in fit[5, 100,] : subscript out of bounds
The problem it caused
The code that 'threw' the error
Message defined as error
DECODING A WARNING
Warning messages can be very variable in format, but there are often common elements.
Warning message:
Removed 1 rows containing missing values (geom_path).
The piece of code that caused the warning
The behaviour being warned about
The declaration of a warning
Errors are a way of telling you why a chunk of code is not possible to execute.
Warnings are a way of telling you that the code is behaving in a different way than you might otherwise expect.


Files
Plots
Packages
Help
Viewer
sd
Refresh Help Topic
R: Standard Deviation
Find in Topic
sd {stats}
R Documentation
Standard Deviation
Description
This function computes the standard deviation of the values in x. If na.rm is TRUE then missing values are removed before computation proceeds.
Usage
sd(x, na.rm = FALSE)
Arguments
x
a numeric vector or an R object but not a factor coercible to numeric by as.double(x).
na.rm
logical. Should missing values be removed?
Details
Like var this uses denominator n - 1.
The standard deviation of a length-one or zero-length vector is NA.

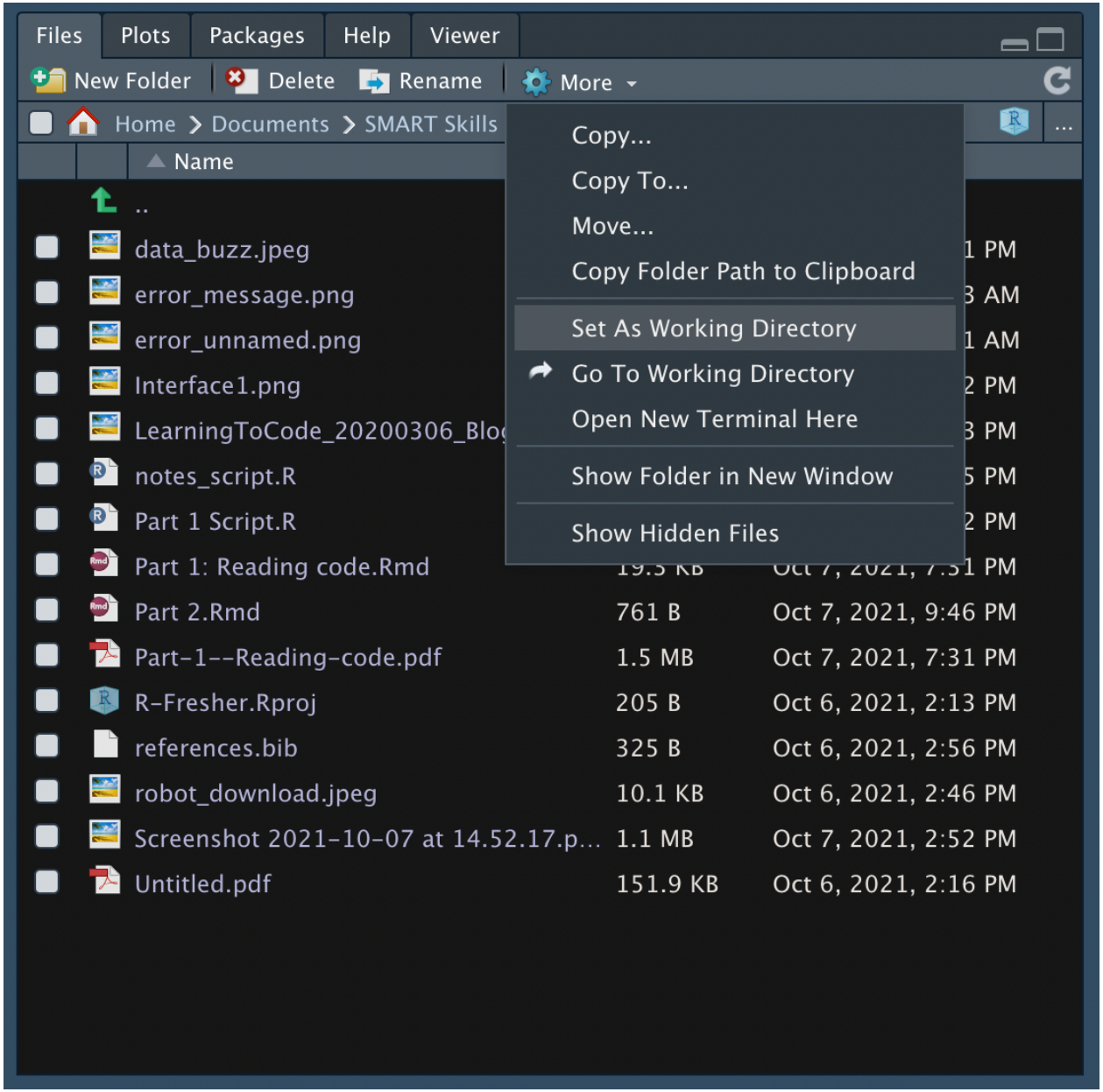
How does **R** read in and write out data?

Working Directories

- The working directory tells R where we want it to read data from and where it should write data
- Usually on your computer, but it can be a cloud location such as Dropbox
- We tell **R** where we want the working directory is through giving it a file path - a roadmap of files that leads to the location

```
# ask R for the current working directory
setwd()

# change the working directory to a different location
setwd(C:/Users/ljasn/classes/R-Fresher/My slides")
```

Working Directories



Reading different data formats

.csv	.csv stands for Comma Separated Values and it is the most common data format because it's easy readable by most software. For this, we use the simple command `read.csv()`
.xls	.xls and .xlsx are the latest suffixes for an Excel file. Most of the time, it is easier to open the data in Excel then save it as a .csv before using the above command in R. However, to read .xls files directly into R, you'll first need to install and load the `readxl` package and then use `read_excel()`
.sav	.sav is the suffix for data saved from SPSS. Sometimes it will be the only available format when downloading large survey data. For this, you will need to install and load the `haven` package and use `read_sav()`
.dta	Similarly, .dta is a STATA saved data format. This also relies on the `haven` package and uses `read_dta()`

Writing files out

write.csv	save the table as a .csv file in either your working directory or another folder if you supply a path - see help file
write.table	save a table as a .txt file or other extension
save	save a list of objects to use later - usually specify as an .Rdata file
save.image	save your whole environment to use later - usually specify as an .Rdata file

Data Structures in R

Data Structures in **R**

- **R** has several built-in data types and structures

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”
    - “3.14”

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”
    - “3.14”

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”
    - “3.14”
- Common data structures

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”
    - “3.14”
- Common data structures
  - Vectors

Data Structures in R

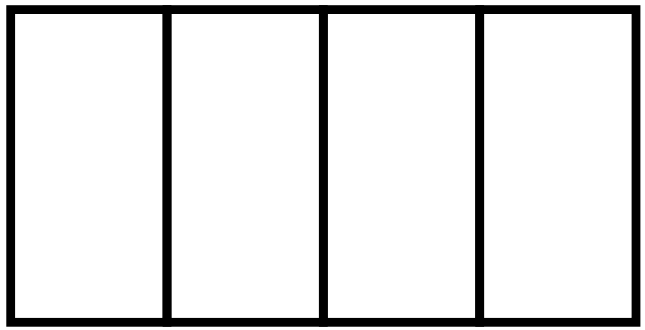
- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”
    - “3.14”
- Common data structures
  - Vectors
  - Matrices

Data Structures in **R**

- **R** has several built-in data types and structures
- Common data types:
  - Numeric (integers, numbers, etc.)
    - 12
    - 3.14
  - Strings (alphanumeric characters in quotation marks)
    - “hello”
    - “3.14”
- Common data structures
  - Vectors
  - Matrices
  - Data Frames

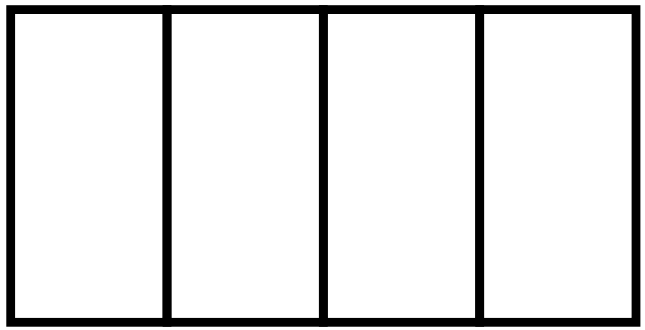
Vectors

A vector is a one-dimensional data structure



Vectors

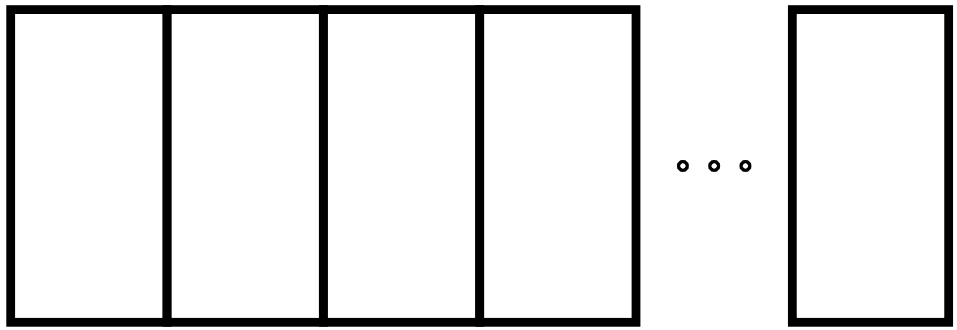
A vector is a one-dimensional data structure



- Vectors are indexed starting at 1

Vectors

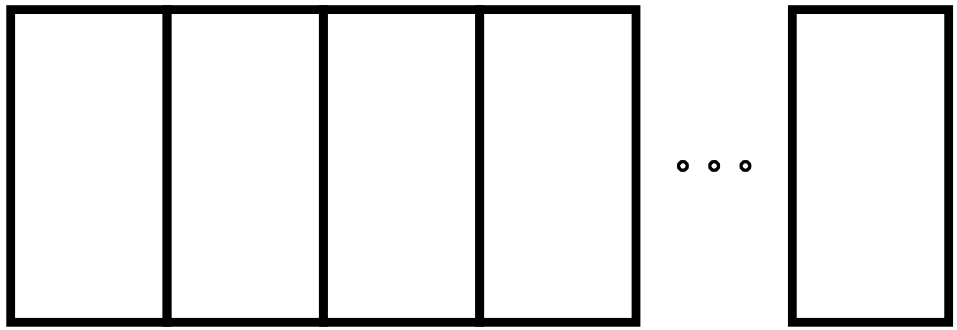
A vector is a one-dimensional data structure



- Vectors are indexed starting at 1
- A vector of length n has n cells

Vectors

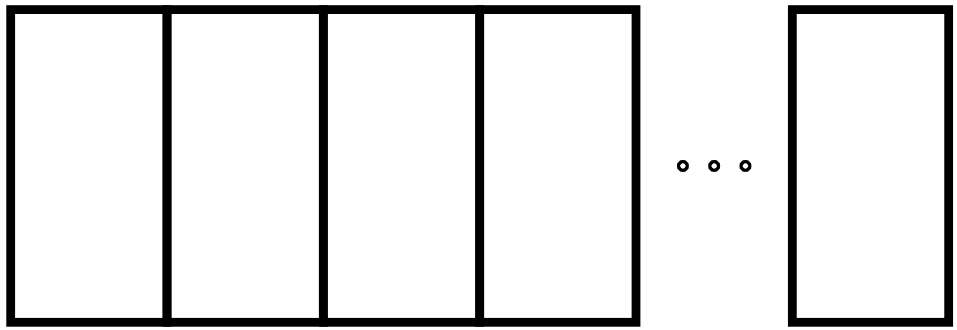
A vector is a one-dimensional data structure



- Vectors are indexed starting at 1
- A vector of length n has n cells
- Each cell can hold a single value

Vectors

A vector is a one-dimensional data structure



- Vectors are indexed starting at 1
- A vector of length n has n cells
- Each cell can hold a single value
- Vectors can only store data of the same type – either all strings or all numerical but not both

Working with Vectors in R

Working with Vectors in **R**

- We index vectors in **R** using "square bracket notation"

Working with Vectors in **R**

- We index vectors in **R** using "square bracket notation"
- Example:
  - you have a vector of numeric values called `testScores`

Working with Vectors in R

- We index vectors in **R** using "square bracket notation"
- Example:
  - you have a vector of numeric values called `testScores`
  - To retrieve the value in the third cell, type `testScores[3]`

Working with Vectors in **R**

- We index vectors in **R** using "square bracket notation"
- Example:
  - you have a vector of numeric values called `testScores`
  - To retrieve the value in the third cell, type `testScores[3]`
  - To retrieve all BUT the third value, type `testScoress[-3]`

Two-dimensional data in R

- Most (all?) of us are familiar with two-dimensional data like that in spreadsheets

Two-dimensional data in **R**

- Most (all?) of us are familiar with two-dimensional data like that in spreadsheets
- **R** has two built-in data structures for storing two-dimensional data

Two-dimensional data in **R**

- Most (all?) of us are familiar with two-dimensional data like that in spreadsheets
- **R** has two built-in data structures for storing two-dimensional data
  - Matrices

Two-dimensional data in **R**

- Most (all?) of us are familiar with two-dimensional data like that in spreadsheets
- **R** has two built-in data structures for storing two-dimensional data
  - Matrices
  - Data Frames

Two-dimensional data in **R**

- Most (all?) of us are familiar with two-dimensional data like that in spreadsheets
- **R** has two built-in data structures for storing two-dimensional data
  - Matrices
  - Data Frames
- In most instances, they behave the same

Two-dimensional data in **R**

- Most (all?) of us are familiar with two-dimensional data like that in spreadsheets
- **R** has two built-in data structures for storing two-dimensional data
  - Matrices
  - Data Frames
- In most instances, they behave the same
- Most functions will accept either a matrix or a data frame

Matrices versus data frames in R

Matrices versus data frames in R

- Matrices can only store data of one type

Matrices versus data frames in R

- Matrices can only store data of one type
  - Either all strings or all numbers, but not both

Matrices versus data frames in **R**

- Matrices can only store data of one type
  - Either all strings or all numbers, but not both
  - If you try to give it multiple types, **R** converts everything to string format

Matrices versus data frames in **R**

- Matrices can only store data of one type
  - Either all strings or all numbers, but not both
  - If you try to give it multiple types, **R** converts everything to string format
- Data frames can store data of multiple types

Matrices versus data frames in **R**

- Matrices can only store data of one type
  - Either all strings or all numbers, but not both
  - If you try to give it multiple types, **R** converts everything to string format
- Data frames can store data of multiple types
  - Ideal for classical data analysis where you might have a mix of numerical and string data

Matrices versus data frames in **R**

- Matrices can only store data of one type
  - Either all strings or all numbers, but not both
  - If you try to give it multiple types, **R** converts everything to string format
- Data frames can store data of multiple types
  - Ideal for classical data analysis where you might have a mix of numerical and string data
- data frames are the default for `read.csv()`

Matrices versus data frames in **R**

- Matrices can only store data of one type
  - Either all strings or all numbers, but not both
  - If you try to give it multiple types, **R** converts everything to string format
- Data frames can store data of multiple types
  - Ideal for classical data analysis where you might have a mix of numerical and string data
- data frames are the default for `read.csv()`
- can convert between the two using `as.matrix()` or `as.data.frame()` - but check what class your object is with `class()`

Working with matrices

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

Working with matrices

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

- Can also use "square bracket notation"

Working with matrices

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

- Can also use "square bracket notation"
- Inside the square brackets, the first position refers to the row(s) and the second to the column(s)

Working with matrices

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

- Can also use "square bracket notation"
- Inside the square brackets, the first position refers to the row(s) and the second to the column(s)
- If this matrix is called `friendSurvey`, the command to retrieve Josh's age is `friendSurvey[2,3]`

Working with data frames

Working with data frames

- Square bracket notation works for data frames as well

Working with data frames

- Square bracket notation works for data frames as well
- Data frames provide another option: dollar sign notation

Working with data frames

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

Working with data frames

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

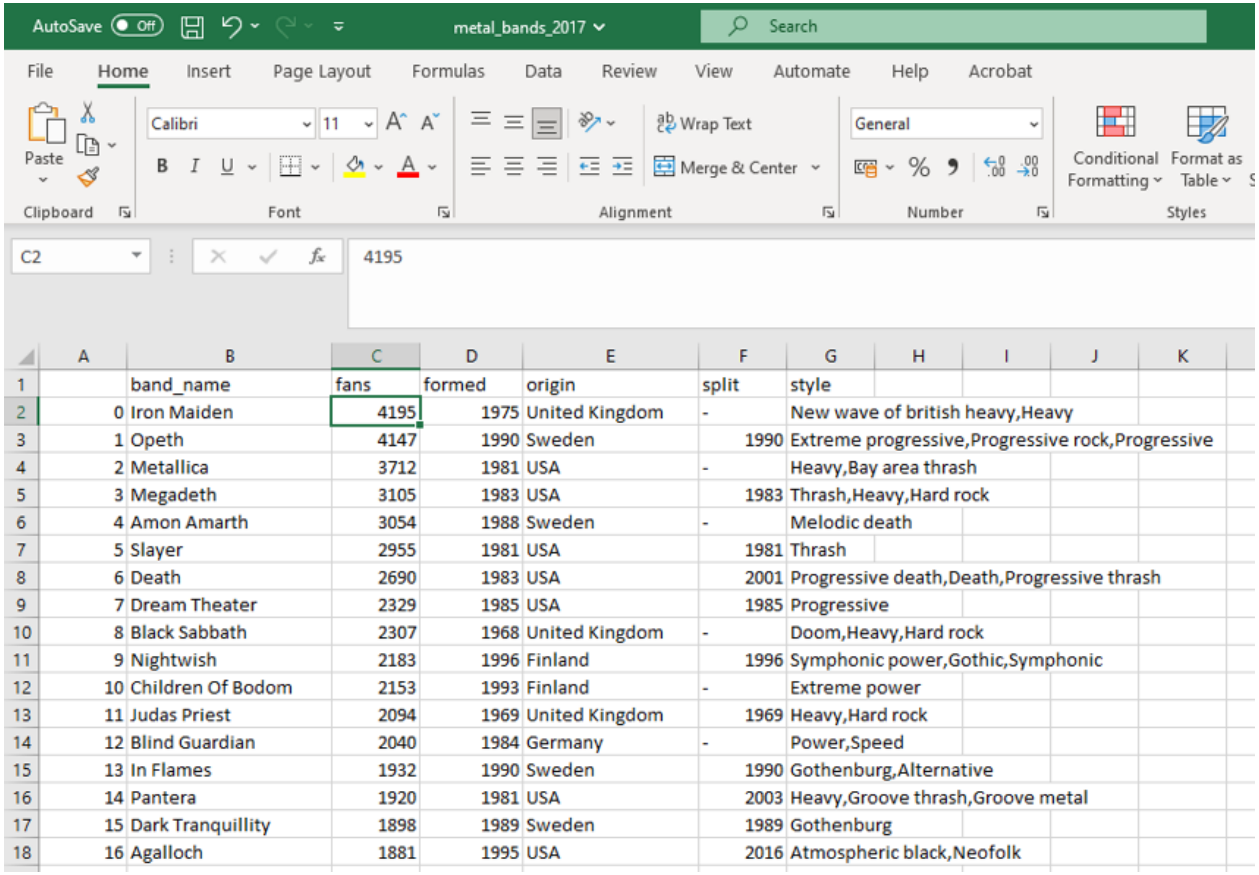
- To retrieve the 'sex' column as a vector, use `friendSurvey$sex`

Working with data frames

id	name	age	sex	handed	lastDocVisit
5012	Danielle	44	F	R	2012
2331	Josh	44	M	R	2008
1989	Mark	40	M	R	2010
2217	Emma	32	F	L	2012
2912	Sarah	33	F	R	2011

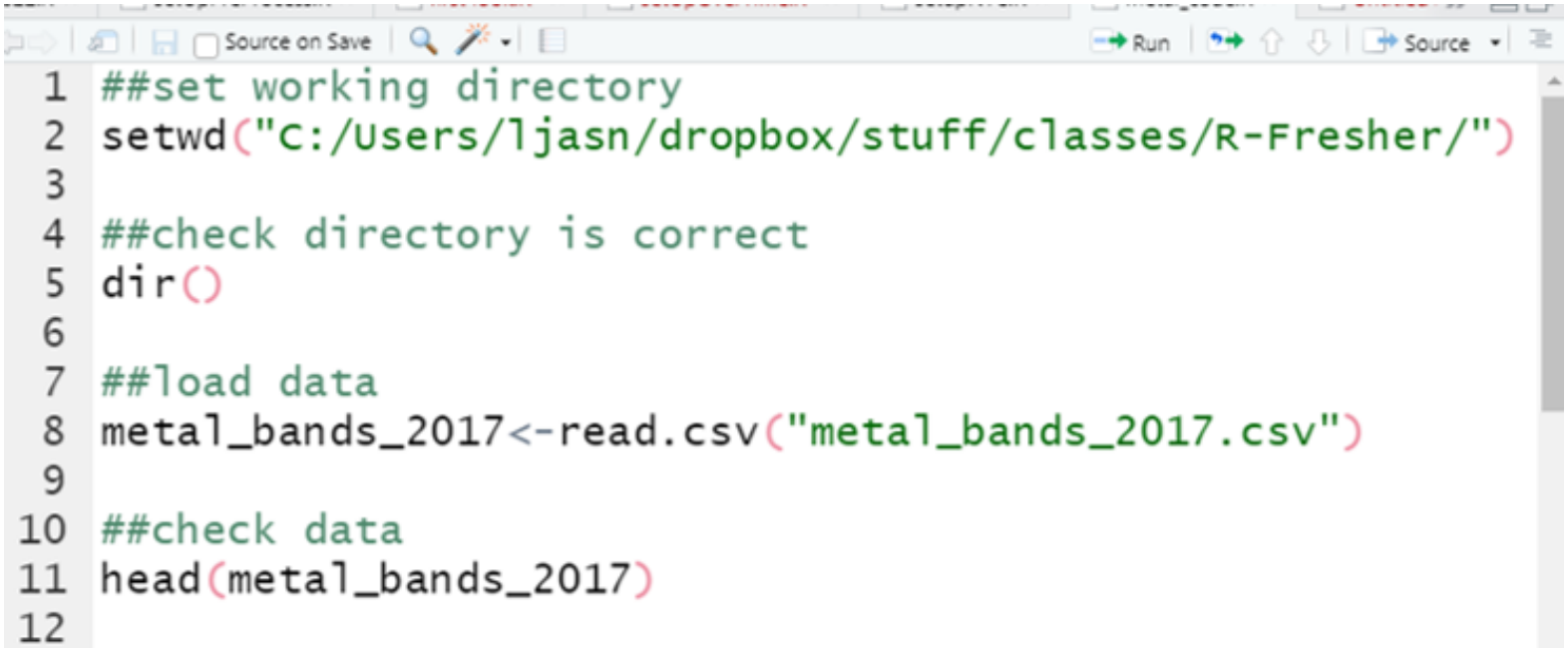
- To retrieve the 'sex' column as a vector, use `friendSurvey$sex`
- To retrieve Josh's age, use `friendSurvey$age[2]`

Example: Metal Bands



	A	band_name	fans	formed	origin	split	style
1		band_name	fans	formed	origin	split	style
2	0	Iron Maiden	4195	1975	United Kingdom	-	New wave of british heavy,Heavy
3	1	Opeth	4147	1990	Sweden	1990	Extreme progressive,Progressive rock,Progressive
4	2	Metallica	3712	1981	USA	-	Heavy,Bay area thrash
5	3	Megadeth	3105	1983	USA	1983	Thrash,Heavy,Hard rock
6	4	Amon Amarth	3054	1988	Sweden	-	Melodic death
7	5	Slayer	2955	1981	USA	1981	Thrash
8	6	Death	2690	1983	USA	2001	Progressive death,Death,Progressive thrash
9	7	Dream Theater	2329	1985	USA	1985	Progressive
10	8	Black Sabbath	2307	1968	United Kingdom	-	Doom,Heavy,Hard rock
11	9	Nightwish	2183	1996	Finland	1996	Symphonic power,Gothic,Symphonic
12	10	Children Of Bodom	2153	1993	Finland	-	Extreme power
13	11	Judas Priest	2094	1969	United Kingdom	1969	Heavy,Hard rock
14	12	Blind Guardian	2040	1984	Germany	-	Power,Speed
15	13	In Flames	1932	1990	Sweden	1990	Gothenburg,Alternative
16	14	Pantera	1920	1981	USA	2003	Heavy,Groove thrash,Groove metal
17	15	Dark Tranquillity	1898	1989	Sweden	1989	Gothenburg
18	16	Agalloch	1881	1995	USA	2016	Atmospheric black,Neofolk

Example: Metal Bands



Age of Band

```
20 ##age of band
21 ##this will throw an error - why?
22 metal_bands_2017$split-metal_bands_2017$formed
```

```
Error in metal_bands_2017$split -
metal_bands_2017$formed :  non-numeric argument to
binary operator
```

Age of Band

```
> ##look for error
> head(metal_bands_2017)
  X   band_name fans formed          origin split
1 0 Iron Maiden 4195   1975 United Kingdom     -
2 1        Opeth 4147   1990          Sweden  1990
3 2   Metallica 3712   1981             USA     -
4 3    Megadeth 3105   1983             USA  1983
5 4 Amon Amarth 3054   1988          Sweden     -
6 5       Slayer 2955   1981             USA  1981
                                          style
1                   New wave of british heavy,Heavy
2 Extreme progressive,Progressive rock,Progressive
3                              Heavy,Bay area thrash
4                            Thrash,Heavy,Hard rock
5                                    Melodic death
6                                           Thrash
> |
```

Age of Band

```
27 ##recoding
28 table(metal_bands_2017$split)
29 metal_bands_2017$split[metal_bands_2017$split=="-"]<-NA
30
31 ##age of band
32 ##try again with recoding
33 metal_bands_2017$split-metal_bands_2017$formed
34
```

Age of Band

```
27 ##recoding
28 table(metal_bands_2017$split)
29 metal_bands_2017$split[metal_bands_2017$split=="-"]<-NA
30
31 ##age of band
32 ##try again with recoding
33 metal_bands_2017$split-metal_bands_2017$formed
34
```

```
Error in metal_bands_2017$split -
metal_bands_2017$formed :  non-numeric argument to
binary operator
```

Age of Band

```
35 ##why still an error?
36 table(metal_bands_2017$formed)
```

Age of Band

```
> ##why still an error?
> table(metal_bands_2017$formed)

   - 1964 1965 1967 1968 1969 1970 1971 1972 1973 1974
   4    1    1    3    7    6    1    1    5    4    3
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985
   7   17   11   11   26   25   30   40   42   53   60
1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996
  53   53   64  102   98   77  100  157  132  169  160
1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007
 159  203  182  174  189  179  217  221  270  248  265
2008 2009 2010 2011 2012 2013 2014 2015 2016
 227  205  202  153  136  118   88   36    5
>
```

Age of Band

```
38 ##more recoding
39 metal_bands_2017$formed[metal_bands_2017$formed=="-"]<-NA
40
41 ##age of band
42 ##try again with recoding
43 metal_bands_2017$split-metal_bands_2017$formed
44
```

Age of Band

```
38 ##more recoding
39 metal_bands_2017$formed[metal_bands_2017$formed=="-"]<-NA
40
41 ##age of band
42 ##try again with recoding
43 metal_bands_2017$split-metal_bands_2017$formed
44
```

```
Error in metal_bands_2017$split -
metal_bands_2017$formed :  non-numeric argument to
binary operator
```

Age of Bands

```
45 ##check something else
46 class(metal_bands_2017$split)
47 class(metal_bands_2017$formed)
```

Age of Bands

```
> ##check something else
> class(metal_bands_2017$split)
[1] "character"
> class(metal_bands_2017$formed)
[1] "character"
>
```

Age of Bands

```
> ##try making everything numeric
> as.numeric(metal_bands_2017$split)-as.numeric(metal_bands_2017$f
ormed)
   [1] NA  0 NA  0 NA  0 18  0 NA  0 NA  0 NA  0 22  0 21  0 NA
  [20]  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0
  [39] NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 18
  [58]  0 NA  0 NA  0 NA  0 22  0 21  0 NA  0 NA  0 NA  0 NA  0
  [77] NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA
  [96]  0 NA  0 NA  0 NA  0 12  0 NA  0 28  0 NA  0 NA  0 NA  0
 [115] NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA
 [134]  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0
 [153] NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA
 [172]  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0
 [191] NA  0 NA  0 NA  0 30  0 NA  0 17  0 NA  0 24  0 NA  0 NA
 [210]  0 NA  0 NA  0 NA  0 NA  0 14  0 NA  0 NA  0 NA  0 NA  0
 [229] NA  0 13  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA
 [248]  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0
 [267] NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA
 [286]  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0  7  0 NA  0
 [305]  9  0 NA  0 NA  0 NA  0 NA  0 NA  0 16  0 NA  0 NA  0 NA
 [324]  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0
 [343] NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA
 [362]  0 19  0 NA  0 34  0 NA  0 NA  0 NA  0 NA  0 NA  0 NA  0
```

Age of Bands

```
52 ##save that as something
53 age<-as.numeric(metal_bands_2017$split)-as.numeric
   (metal_bands_2017$formed)
54
55 ##add it to the dataframe
56 metal_bands_2017<-cbind(metal_bands_2017,age)
57
58 ##save it!
59 write.csv(metal_bands_2017,file="metal_bands_2017_withAge
   .csv")
```

Testing Age vs Popularity

```
61 ##test whether older bands (who eventually split before 2017)
   more popular
62 cor.test(metal_bands_2017$age,metal_bands_2017$fans)
63
```

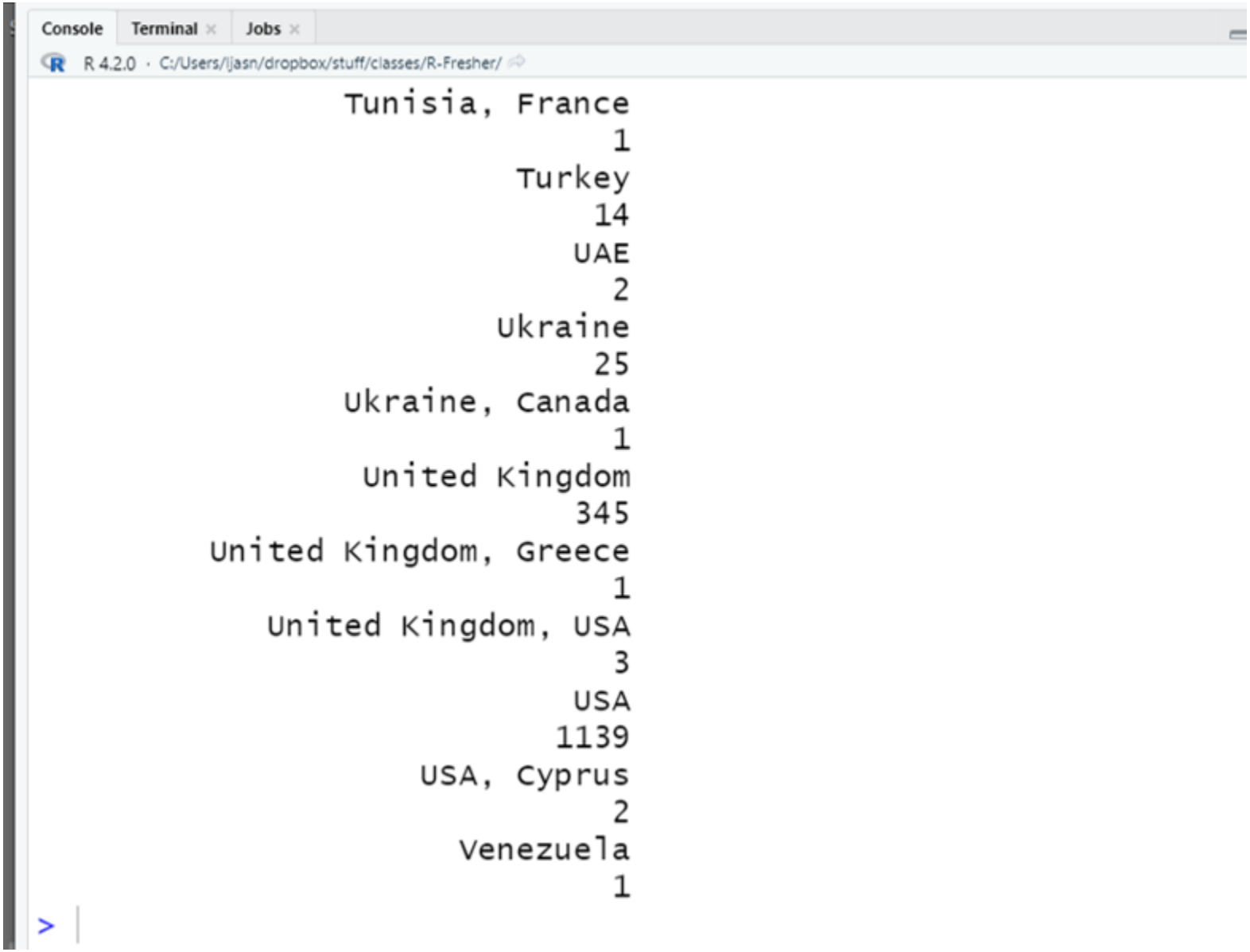
Testing Age vs Popularity

```
> cor.test(metal_bands_2017$age,metal_bands_2017$fans)

        Pearson's product-moment correlation

data:  metal_bands_2017$age and metal_bands_2017$fans
t = 3.3242, df = 2783, p-value = 0.0008983
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.02580601 0.09979738
sample estimates:
       cor
0.06288811

>
```

Are UK bands more popular?

```
64 ##are UK metal bands more popular than non-UK metal bands?
65
66 ##let's look at how 'origin' is coded
67 table(metal_bands_2017$origin)
68
```

Are UK bands more popular?



Are UK bands more popular?

```
68
69 ##recode using an if/else statement
70 fromUK<-ifelse(grepl("United Kingdom",metal_bands_2017$origin),1,0)
71
```

Are UK bands more popular?

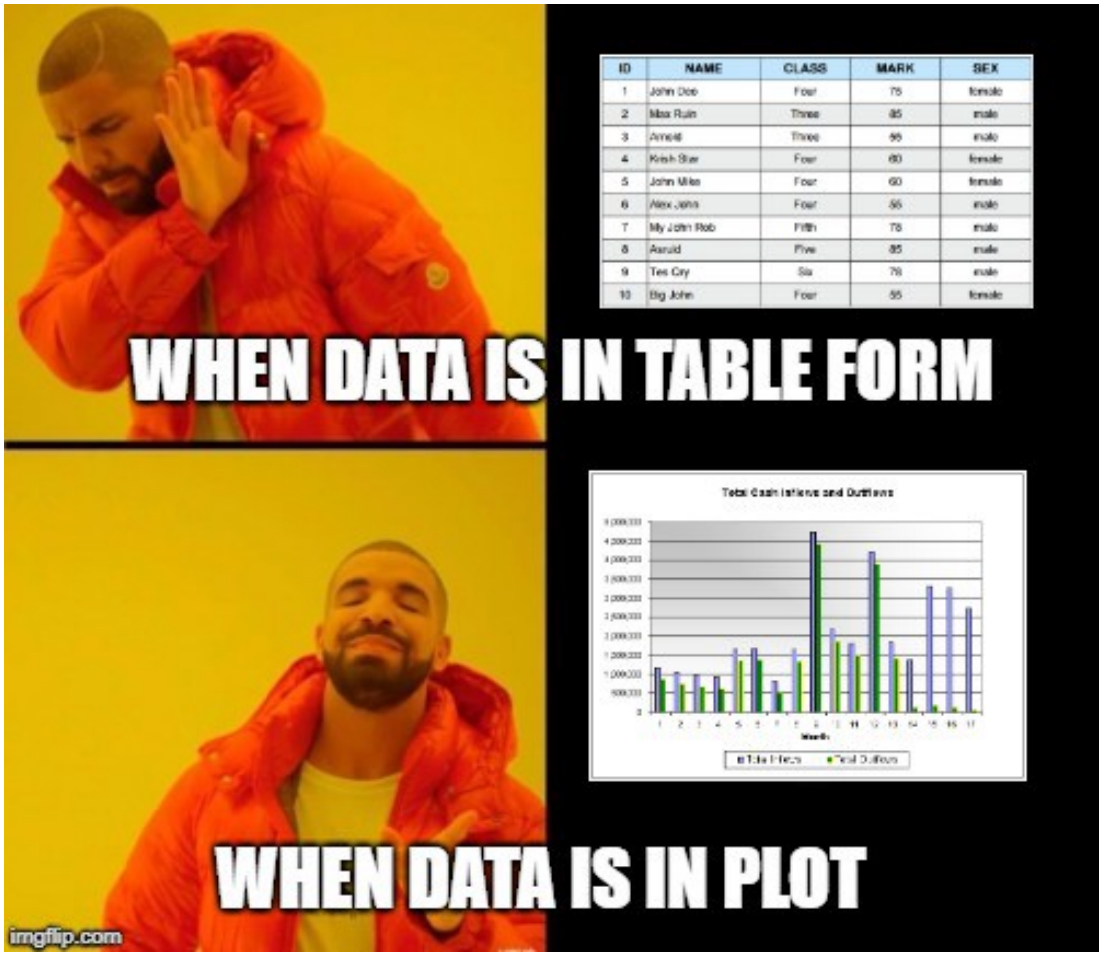
```
71
72 ##now we can run the test
73 t.test(metal_bands_2017$fans[fromUK==1],metal_bands_2017$fans[fromUK==0])
74
75
```

Are UK bands more popular?

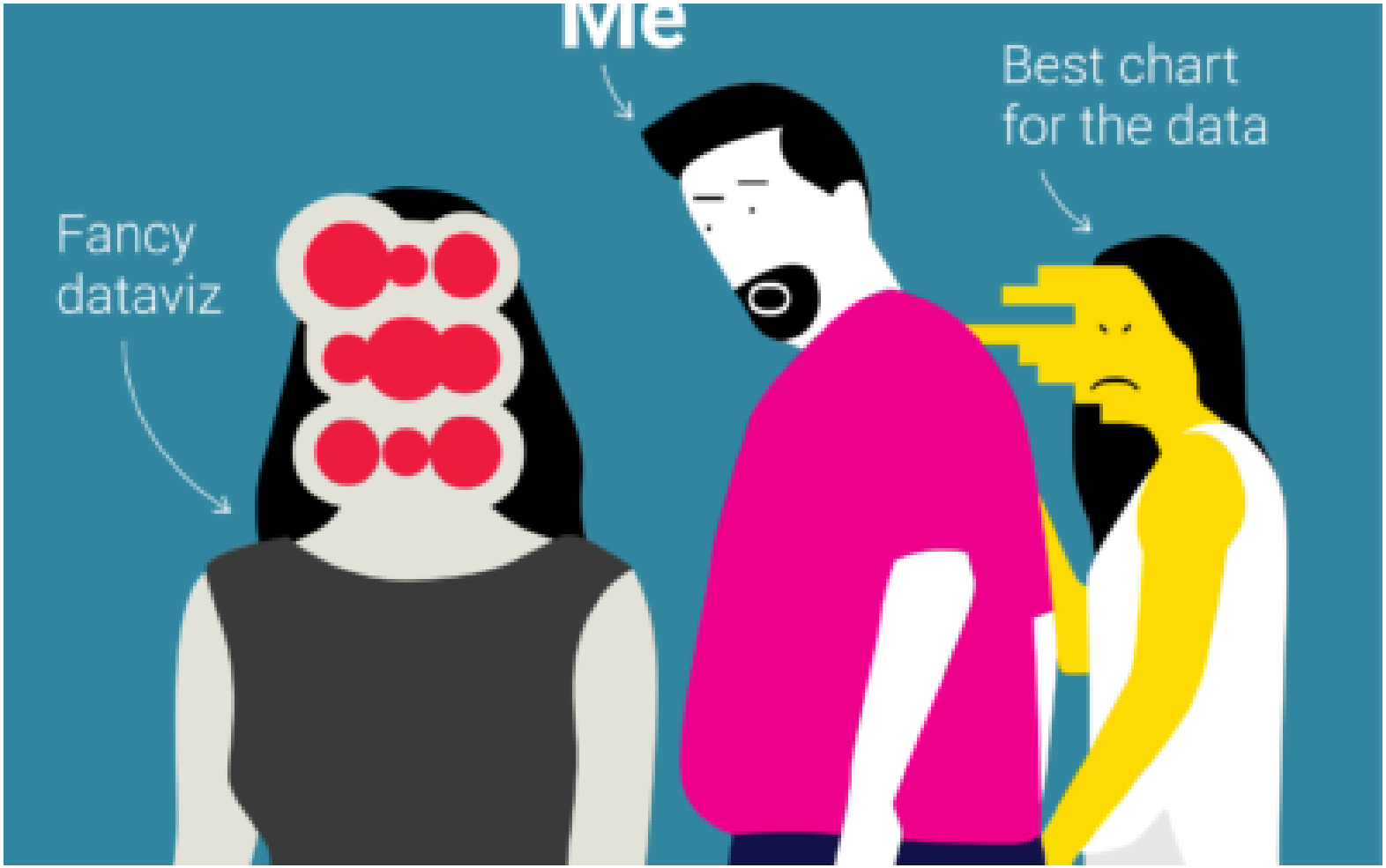
```
> ##now we can run the test
> t.test(metal_bands_2017$fans[fromUK==1],metal_bands_2017$fans[fromUK==0])

        Welch Two Sample t-test

data:  metal_bands_2017$fans[fromUK == 1] and metal_bands_2017$fans[fromUK == 0]
t = 2.027, df = 375.92, p-value = 0.04337
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  1.428692 93.980194
sample estimates:
mean of x mean of y
132.13277  84.42833

> |
```

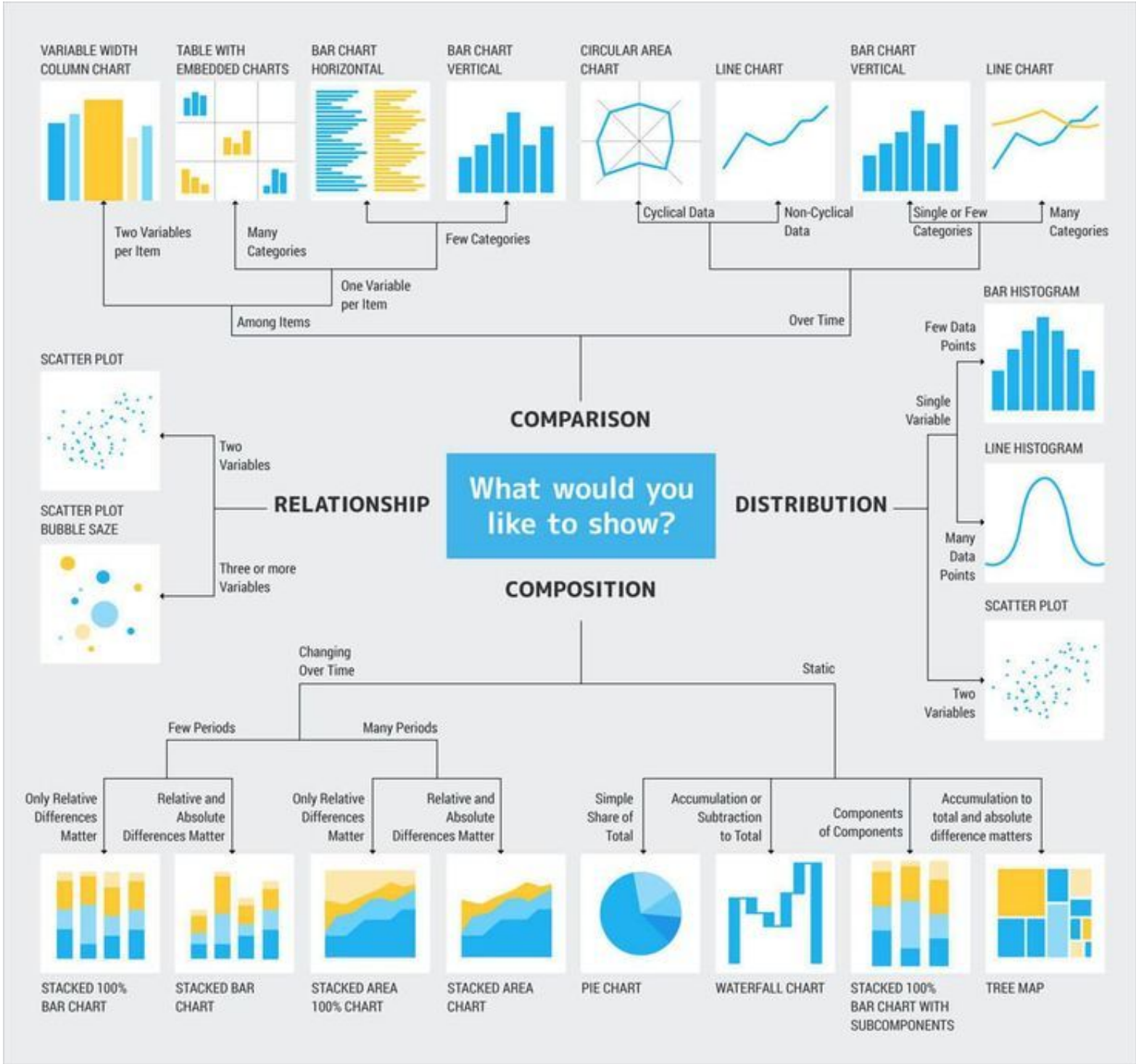
Visualizing Data

Data scientists possess a fundamental skill: not only do they understand data, they can communicate it.



Visualizing Data




What would you like to show?
COMPARISON
RELATIONSHIP
DISTRIBUTION
COMPOSITION
VARIABLE WIDTH COLUMN CHART
TABLE WITH EMBEDDED CHARTS
BAR CHART HORIZONTAL
BAR CHART VERTICAL
CIRCULAR AREA CHART
LINE CHART
BAR CHART VERTICAL
LINE CHART
Two Variables per Item
Many Categories
Few Categories
One Variable per Item
Among Items
Cyclical Data
Non-Cyclical Data
Single or Few Categories
Many Categories
Over Time
SCATTER PLOT
Two Variables
SCATTER PLOT BUBBLE SAZE
Three or more Variables
BAR HISTOGRAM
Few Data Points
Single Variable
LINE HISTOGRAM
Many Data Points
SCATTER PLOT
Two Variables
Changing Over Time
Static
Few Periods
Many Periods
Only Relative Differences Matter
Relative and Absolute Differences Matter
Only Relative Differences Matter
Relative and Absolute Differences Matter
Simple Share of Total
Accumulation or Subtraction to Total
Components of Components
Accumulation to total and absolute difference matters
STACKED 100% BAR CHART
STACKED BAR CHART
STACKED AREA 100% CHART
STACKED AREA CHART
PIE CHART
WATERFALL CHART
STACKED 100% BAR CHART WITH SUBCOMPONENTS
TREE MAP

ggplot2

The ggplot package starts with a blank canvas and builds a visualisation piece by piece.

ggplot2

The ggplot package starts with a blank canvas and builds a visualisation piece by piece.

In the first line of code, we give it the dataframe we want to plot from and specifications for x and y; R uses this to plot the two axes but no data yet.

ggplot2

The ggplot package starts with a blank canvas and builds a visualisation piece by piece.

In the first line of code, we give it the dataframe we want to plot from and specifications for x and y; R uses this to plot the two axes but no data yet.

Then we use a + sign to add our next line of code that builds a type of plot onto the axes.

ggplot2

The ggplot package starts with a blank canvas and builds a visualisation piece by piece.

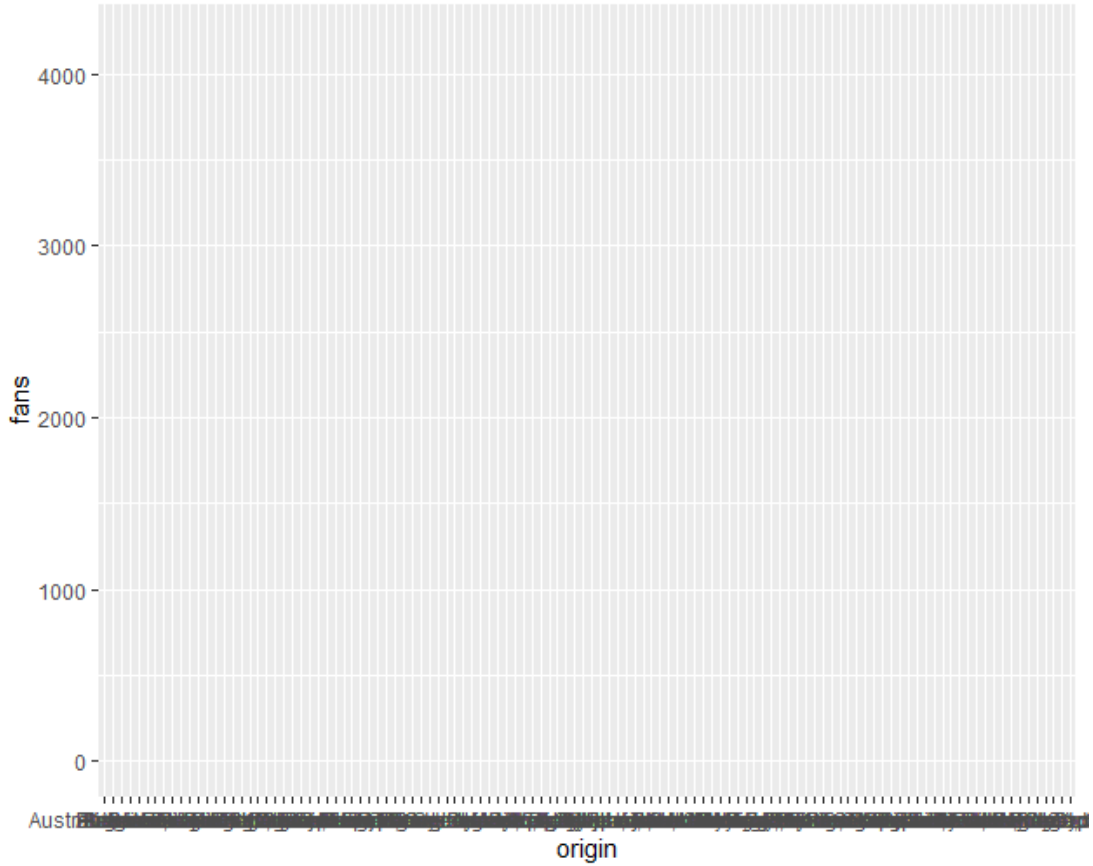
In the first line of code, we give it the dataframe we want to plot from and specifications for x and y; R uses this to plot the two axes but no data yet.

Then we use a + sign to add our next line of code that builds a type of plot onto the axes.

We specify the type of plot in the geom_ function.

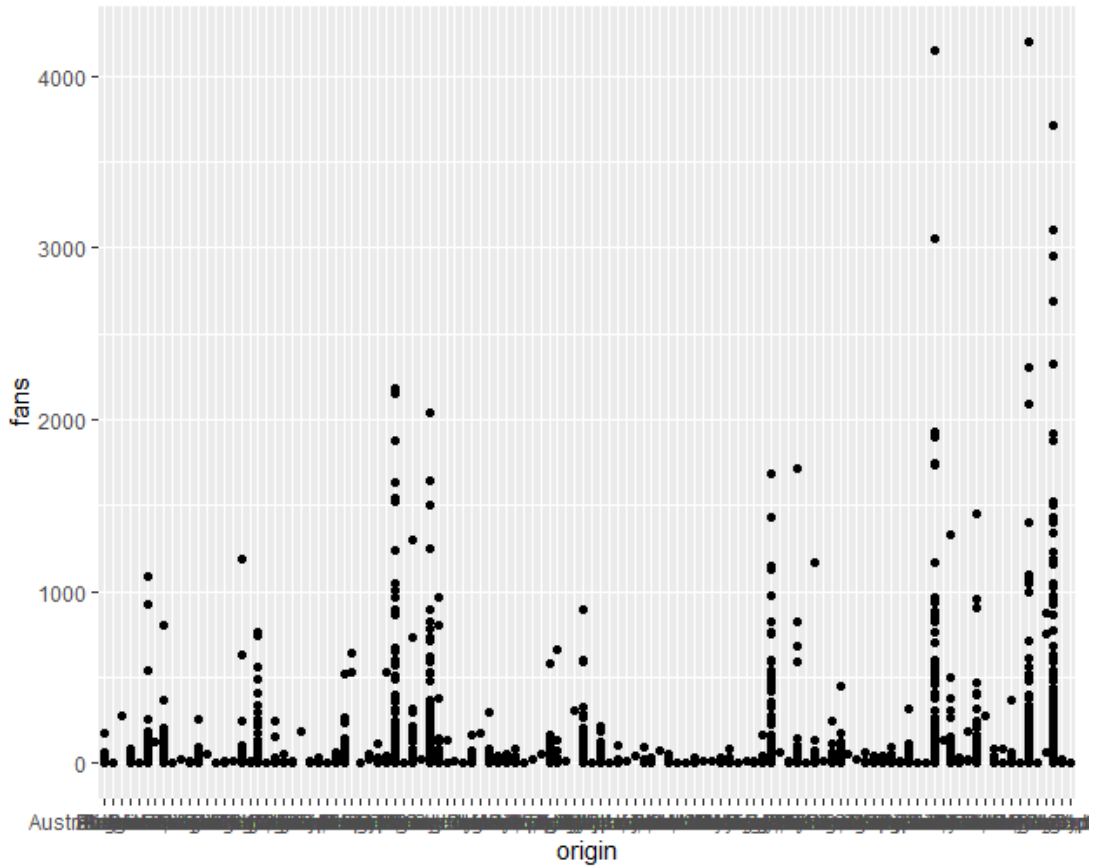
ggplot2

```
library(ggplot2)
# Plot number of fans by country of origin
ggplot(metal_bands_2017, aes(x = origin, y =
fans))
```



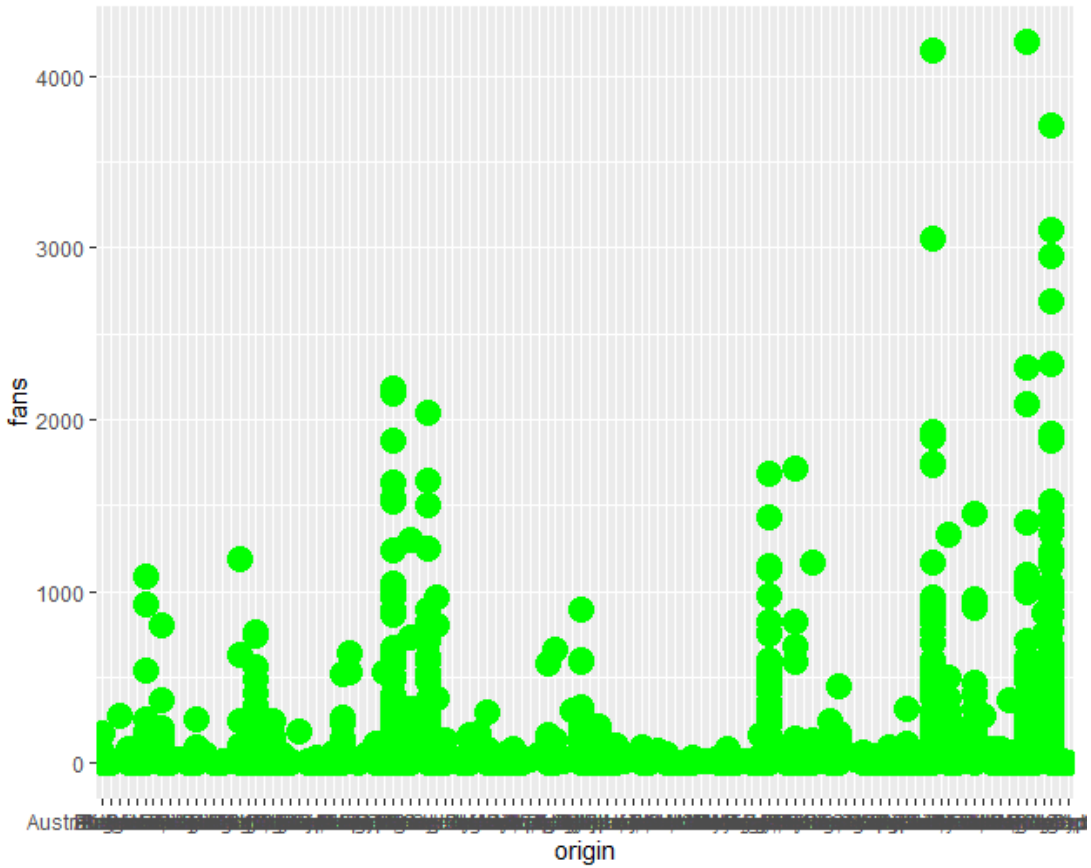
ggplot2

```
# now add the data points
ggplot(metal_bands_2017, aes(x = origin, y =
fans)) + geom_point()
```



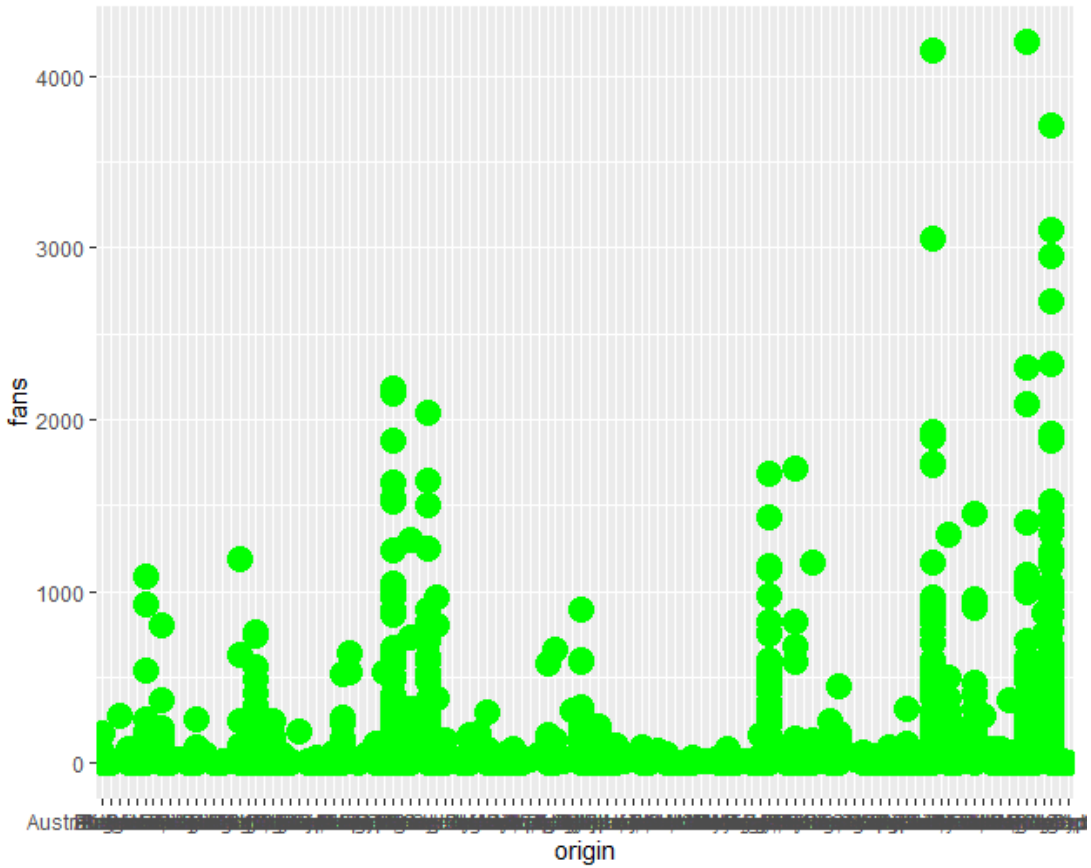
ggplot2

```
#set a static colour and size for the points
ggplot(metal_bands_2017, aes(x = origin, y =
fans)) + geom_point(col = ``green", size = 5)
```



ggplot2

```
#set a static colour and size for the points
ggplot(metal_bands_2017, aes(x = origin, y =
fans)) + geom_point(col = ``green", size = 5)
```

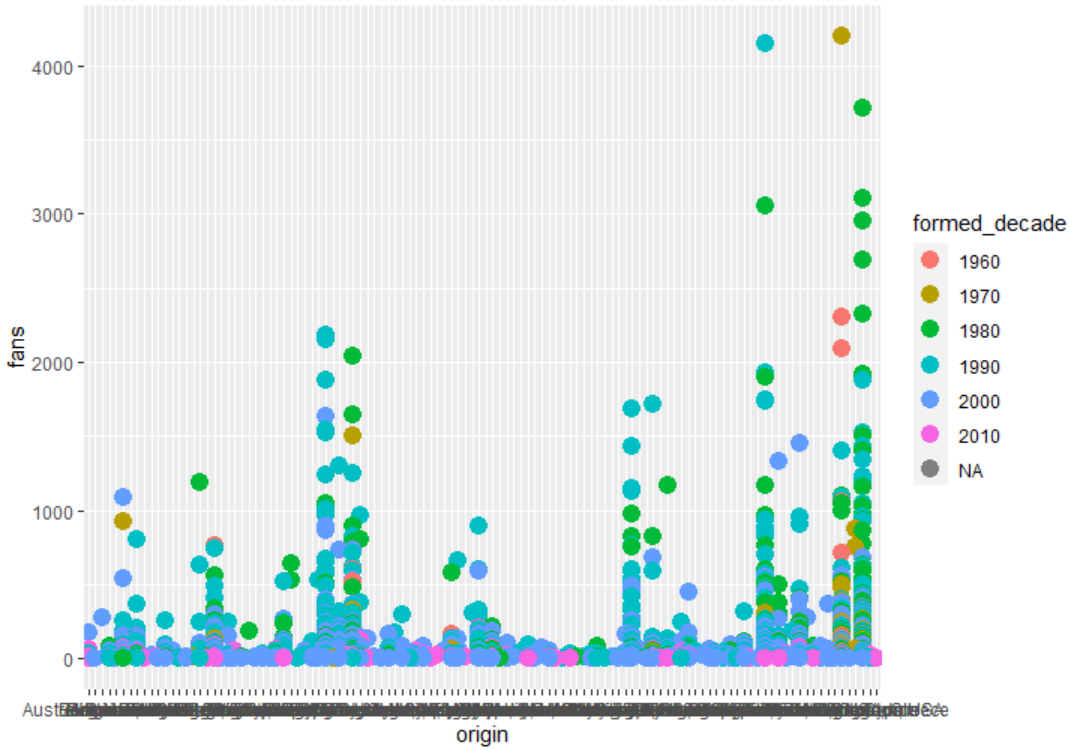


Recode more data

```
74
75 ##recode 'formed' by decade
76 formed_decade<-metal_bands_2017$formed
77
78 ##what do we do about NAs?
79 min(formed_decade)
80 min(formed_decade,na.rm=T)
81 max(formed_decade,na.rm=T)
82
83 ##recode the data
84 formed_decade[1960<=formed_decade&formed_decade<1970]<-1960
85 formed_decade[1970<=formed_decade&formed_decade<1980]<-1970
86 formed_decade[1980<=formed_decade&formed_decade<1990]<-1980
87 formed_decade[1990<=formed_decade&formed_decade<2000]<-1990
88 formed_decade[2000<=formed_decade&formed_decade<2010]<-2000
89 formed_decade[2010<=formed_decade&formed_decade<2020]<-2010
90
91 ##add it to the data frame
92 metal_bands_2017<-cbind(metal_bands_2017,formed_decade)
93
```

ggplot2

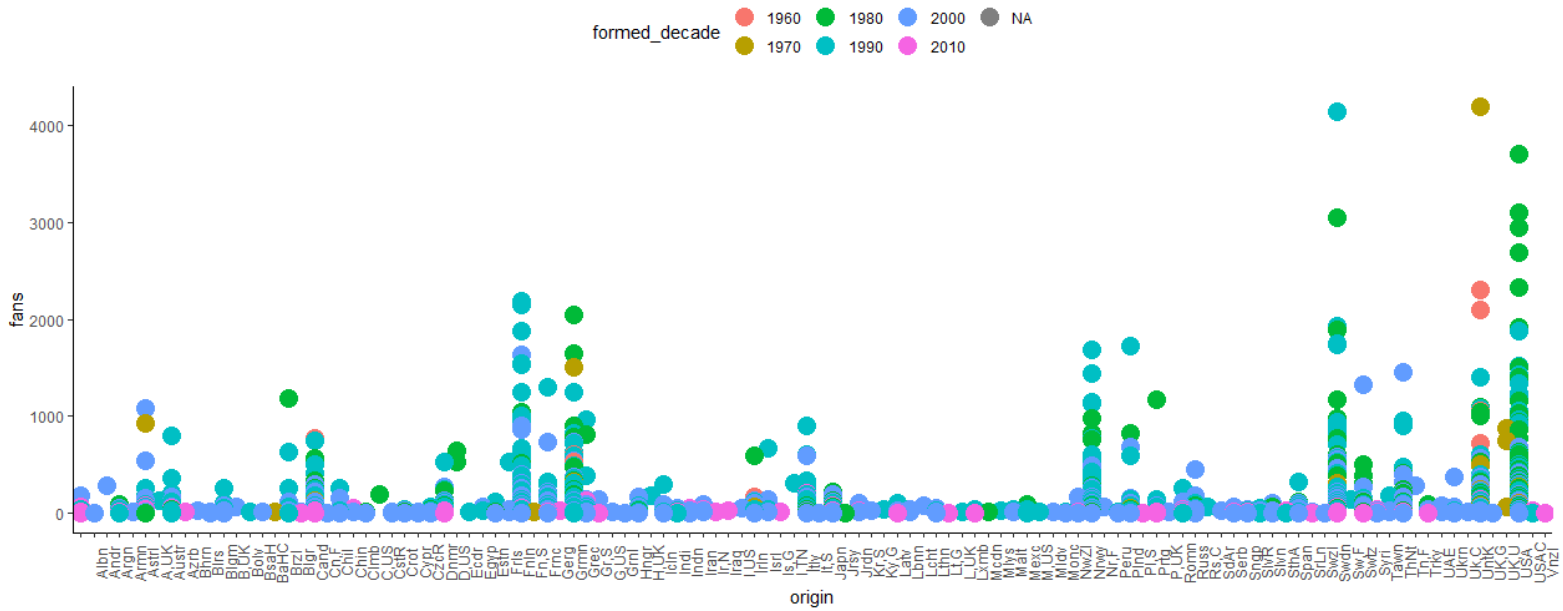
```
#set a static colour and size for the points
ggplot(metal_bands_2017, aes(x = origin, y =
fans)) + geom_point(aes(col = formed_decade), size
= 5)
```



ggplot2

```
#use the theme function to rotate the x axis text
#change legend placement
#abbreviate the labels

ggplot(metal_bands_2017, aes(x = origin, y =
fans))
+ geom_point(aes(col = formed_decade), size = 5)
+ scale_x_discrete(labels = abbreviate)
+ theme_classic()
+ theme(axis.text.x = element_text(angle = 90),
legend.position = ‘‘top")
```

ggplot2


formed_decade
1960
1970
1980
1990
2000
2010
NA
fans
4000
3000
2000
1000
0
origin

Which ggplot::geom_ do I choose?

Which ggplot::geom_ do I choose?

- When you've decided what kind of graph you want, you then need to translate that into ggplot syntax

Which ggplot::geom_ do I choose?

- When you've decided what kind of graph you want, you then need to translate that into ggplot syntax
- Some of them are intuitive, such as geom_bar or geom_boxplot, but I am guilty of repeatedly trying to type 'geom_scatter' which doesn't exist

Which ggplot::geom_ do I choose?

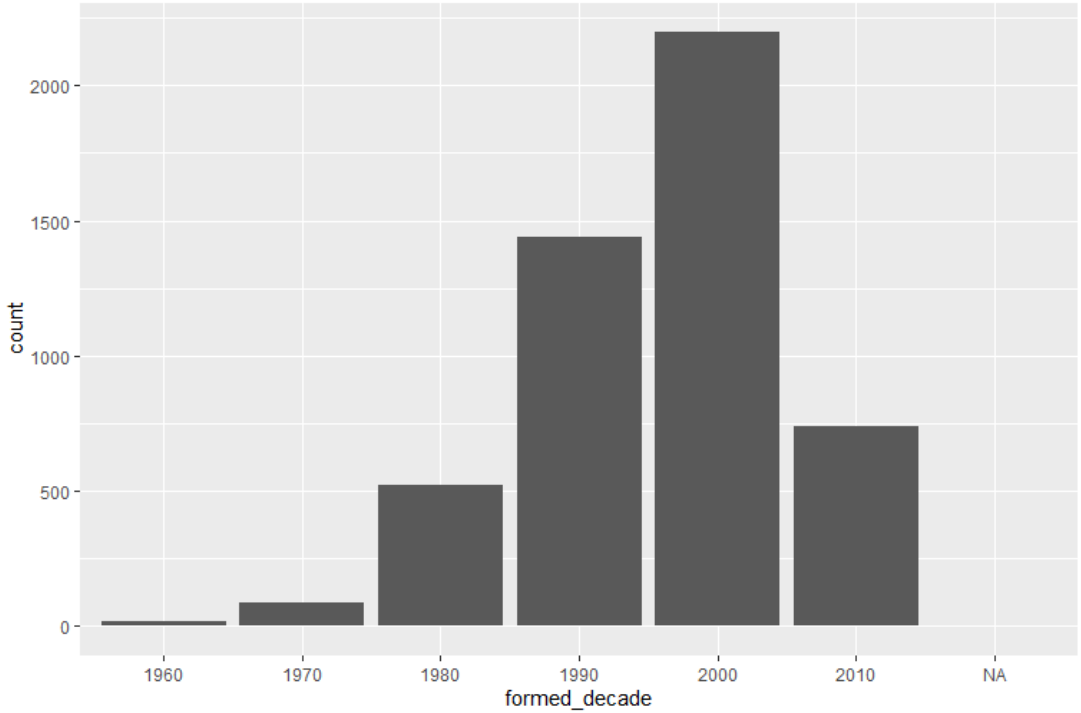
- When you've decided what kind of graph you want, you then need to translate that into ggplot syntax
- Some of them are intuitive, such as geom_bar or geom_boxplot, but I am guilty of repeatedly trying to type 'geom_scatter' which doesn't exist
- This is my data vis bible: https://www.r-graph-gallery.com/ggplot2-package.html as it has detailed explanations of how to plot lots of different graphs

Which ggplot::geom_ do I choose?

- When you've decided what kind of graph you want, you then need to translate that into ggplot syntax
- Some of them are intuitive, such as geom_bar or geom_boxplot, but I am guilty of repeatedly trying to type 'geom_scatter' which doesn't exist
- This is my data vis bible: https://www.r-graph-gallery.com/ggplot2-package.html as it has detailed explanations of how to plot lots of different graphs
- There are similar walk-throughs for 50 plots with helpful subheadings here.

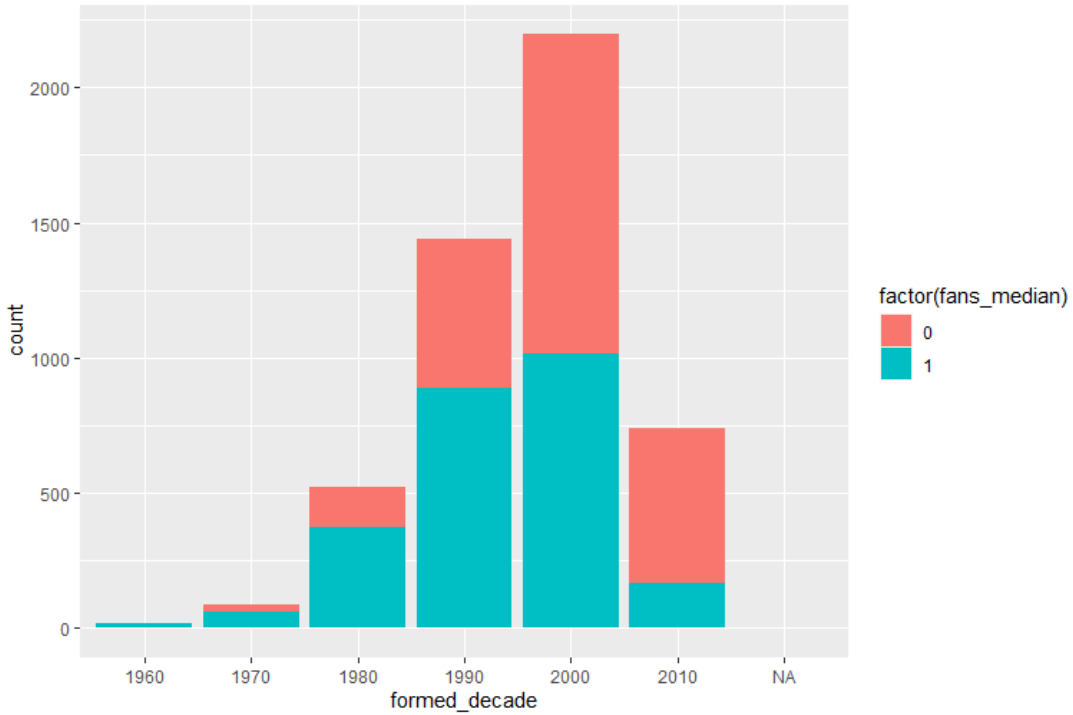
barplot

```
# example bar plot
ggplot(metal_bands_2017, aes(x = formed_decade)) +
geom_bar()
```



barplot

```
# example bar plot
ggplot(metal_bands_2017, aes(x = formed_decade,
fill=factor(fans_median))) +
geom_bar(position=‘‘stack")
```



Data visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                   required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
stat = <STAT>, position = <POSITION>) +                    Not required, sensible defaults supplied
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

last_plot() Returns the last plot.

ggsave("plot.png", width = 5, height = 5**)** Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Aes

Common aesthetic values.

color and **fill** - string ("red", "#RRGGBB")

linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotdash", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")

RStudio

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank() and **a + expand_limits()** Ensure limits include values across all plots.

b + geom_curve(aes(yend = lat + 1, xend = long + 1), curvature = 1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend = "butt", linejoin = "round", linemitre = 1**)** x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(alpha = 50)**)** - x, y, alpha, color, fill, group, subgroup, linetype, size

b + geom_rect(aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900)**)** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept = 0, slope = 1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**

b + geom_segment(aes(yend = lat + 1, xend = long + 1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin"**)** x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)** x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot() x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)** x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar() x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

both continuous

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_point() x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile() x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)** x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)** x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1**)** - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col() x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot() x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)** x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)** x, y, alpha, color, fill, group, linetype, size, weight

both discrete

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count() x, y, alpha, color, fill, shape, size, stroke

e + geom_jitter(height = 2, width = 2**)** x, y, alpha, color, fill, shape, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z)**)** x, y, z, alpha, color, group, linetype, size, weight

l + geom_contour_filled(aes(fill = z)**)** x, y, alpha, color, fill, group, linetype, size, subgroup

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)**)** x, y, alpha, color, fill, linetype, size, weight

h + geom_density_2d() x, y, alpha, color, group, linetype, size

h + geom_hex() x, y, alpha, color, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area() x, y, alpha, color, fill, linetype, size

i + geom_line() x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)** x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

j + geom_crossbar(fatten = 2**)** - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar() - x, ymax, ymin, alpha, color, group, linetype, size, width Also **geom_errorbarh()**.

j + geom_linerange() x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
      state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size