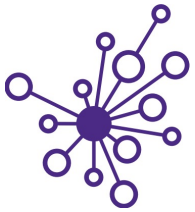
Knowledge and solutions
for a changing world

Be boundless



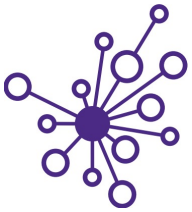
Advancing data-intensive
discovery in all fields

Data Science for Social Good

Documentation and programming style

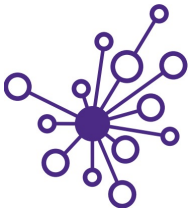
Dave Beck & Jose Hernandez
eScience Institute: putting the ***e*** back in Science.

Agenda



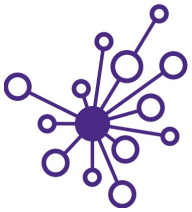
- Style
  - Intro (Review of Justification)
  - Survey of R Style Guide

Programming Style



"Good coding style is like correct punctuation:
you can manage without it,
butitsuremakesthingseasiertoread." - Hadley

Programming Style



- Why is it important that people can read your code? (SURF)
  - Sustainable
    - New version of R (4.x?)
  - Understandable
    - Is it doing what you claim?
  - Reusable
    - Can it be incorporated into a larger project?
  - Fixable

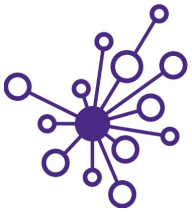
R Style

- There isn't only one 'style':
  - Google R Style Guide + Tidyverse Style Guide
    - *The OG style guide: (http://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html)*
    - *The Tidyverse Style Guide = Google's Guide ++*
    - https://style.tidyverse.org/
    - Present: Google defaults to the Tidyverse style guide with some exceptions. See this for details: https://google.github.io/styleguide/Rguide.html

R Style

- There isn't only one 'style':
  - Bioconductor Style Guide:
    - http://master.bioconductor.org/developers/how-to/coding-style/

Programming Style



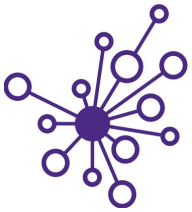
- Most important rule of any style
  - Consistency
    - If you make particular decision about a style guide, use it consistently
    - Always
    - Forever

Put some color in your style

- Tools for checking the style adherence
  - Many editors allow you to color your code
    - Sublime, Atom, vim, Visual Studio Code
  - Turn this ON if it isn't already!

```
1  from statistics import mean
2  import numpy.random as nprnd
3  from statistics import stdev
4  def MyFuNcTiOn(ARGUMENT):
5      m = mean(ARGUMENT)
6      s = stdev(ARGUMENT)
7      gt3sd = 0
8      lt3sd = 0
9      for m in ARGUMENT:
10         if m > m + (s * 2):
11             gt3sd += 1
12         elif m < m - (s * 2):
13             lt3sd += 1
14     return(gt3sd,lt3sd)
15 def AnotherFunction(anumber, anothernumber):
16     l = nprnd.randint(anothernumber, size = anumber)
17     return(MyFuNcTiOn(l))
18 a,b=AnotherFunction(anumber = 1000, anothernumber = 1000)
19 print('found %d random values greather than 2 * sd and %d less than 2 * sd' % (a, b))
```

No, sorry!
Those colors just don't work together.



- Play with the colors a bit…

- What does your brain do when you see something like below?

```
1  # This creates a data frame of rows of column names and the percent of missingness
2  missing_values <- gather(missing_values, key = "feature", value = "missing_pct")
3
4  # This creates a bar graph of the percent missing by column
5  missing_values %>%
6    ggplot(aes(x = reorder(feature,-missing_pct), y = missing_pct)) +
7    geom_bar(stat = "identity", fill = "red") +
8    coord_flip() + theme_bw()
9
```

No, sorry!
Those colors just don't work together.

- Play with the colors a bit…
  - Your colors should never be set so that code comments are "diminished in value"

Tools for style checking

- Tools…
  - *lintr* (https://github.com/jimhester/lintr)
    - R package offering static code analysis that checks adherence to a given style, syntax errors and possible semantic issues.

```
install.packages("lintr")
```

  - styler (https://styler.r-lib.org/)
    - Intended to be used interactively with RStudio, but adaptable.
    - Adheres to the tidyverse formatting rules.

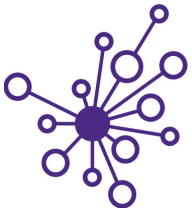
Getting' some style

- How to use them?
  - Use interactively in Rstudio:
  - Invoke explicitly in R console:
    - `lintr(file_name.R)`
    - e.g. `R_demo_1.py`

Ask a friend?

- Other ways to check your style:
  - Many editors support real time style checking!
    - RStudio
    - Visual Studio Code
    - Atom
  - Know and use style until it becomes muscle memory!

Let's play!



- Clone the demo repository
  - SSH:

```
git clone
git@github.com:jmhernan/programming_style_documen
tation.git
```

  - HTTPS:

```
git clone
https://github.com/jmhernan/programming_style_doc
umentation.git
```

Let's play!

- Clone the demo repository
- `Open directory using Rstudio: programming_style_documentation`
- Try one of the style tools, e.g.

```
install.packages("lintr")
lintr::lint("R_demo_1.R")
```

- This slide intentionally left blank

R Style Guide

- General
  - For assignment use <- and not =
  - Don't use ;

```
112  # This
113  x <- 23
114  y <- 12
115  z <- 20
116
117  # Not this
118  x <- ; y <- 12 ; z <- 20
```

R Style Guide

- Indentation
  - Two spaces!
    - Most editors can be set to convert a tab that you type to two spaces in the file
    - RStudio by default has tab = 2-spaces
    - What about lines that wrap?
      - Wrap and indent to opening of parens

R Style Guide

- Indentation

```
# Good
long_function_name <- function(a = "a long argument",
                               b = "another argument",
                               c = "another long argument") {
  # As usual code is indented by two spaces.
}

# Bad
long_function_name <- function(a = "a long argument",
  b = "another argument",
  c = "another long argument") {
  # Here it's hard to spot where the definition ends and the
  # code begins
}
```

R Style Guide

- Indentation

```
 9  if (this_is_one_thing &&
10      that_is_another_thing) {
11      do_something()
12  }
```

```
1
2  # Add a comment, which will provide some distinction in editors
3  # supporting syntax highlighting.
4  if (this_is_one_thing &&
5      that_is_another_thing) {
6    # Since both conditions are true, we can frobnicate
7    do_something()
8  }
```

R Style Guide

- Indentation

```
14  # This is preferred
15  my_list <- list(
16    1, 2, 3,
17    4, 5, 6
18  )
19
20  # Over this
21  my_list <- list(
22    1, 2, 3,
23    4, 5, 6
24    )
```

R Style Guide

- Curly braces
  - Do not place { on its own line
  - Place } on its own unless followed by else
  - Surround else with curly braces

```
120 if (a < d) {
121   a <- (b + c) * d
122 } else {
123   a <- d
124 }
```

R Style

- Maximum line length?
  - Coding lines? Keep it to 80 characters
    - Most editors can show you the line position
    - E.g. vim, Sublime, RStudio
  - Comments
    - No specific recommendations but.
    - > 80 characters
  - Why? My monitor is big!
    - Open two files side by side? History?
    - Some teams choose to use a different max

R Style Guide

- Line spacing
  - One blank line between functions
  - One blank line between logical groups in a function *(sparingly)*
  - *No recommendation about groups of related functions.*

R Style Guide

- Importing packages
  - Imports go at the top of a file after any comments
  - Libraries go on separate lines

```r
# Good
library(tidyverse)
library(stats)

# Bad
library(tidyverse); library(stats)
```

R General Tips

- Package loads
  - Packages should be grouped with a blank line separating each group in the following order:
    - Standard library imports
      - grid, parallel, …
    - Related third party imports
      - dplyr, ggplot2, readr, etc...
    - Local application

```
95  library(boot)
96  library(grid)
97  library(parallel)
98
99  library(dplyr)
100 library(ggplot2)
101
102 source('MyFunction.R')
```

R Style Guide

- Quotes
  - When should I use single?
  - When should I use double?

R Style Guide

- Quotes
  - Use double quotes, not single quotes for quoting text.
    - Except for single quotes, when text already contains a double quotes and no single quotes.

```
# Good
"Text"
'Text with "quotes"'
'<a href="http://style.tidyverse.org">A link</a>'

# Bad
'Text'
'Text with "double" and \'single\' quotes'
```

R Style Guide

- Whitespace
  - No trailing spaces at end of a line
  - Always put space after a comma, never before.

```
# Good
x[, 1]


# Bad
x[,1]
x[ ,1]
x[ , 1]
```

R Style Guide

- Whitespace
  - Parentheses
    - Do not put spaces inside or outside parentheses for regular function calls.
    - Place space before and after () when used with if, for, or while.

```r
# Good
mean(x, na.rm = TRUE)

# Bad
mean (x, na.rm = TRUE)
mean( x, na.rm = TRUE )
```

```r
# Good
if (debug) {
  show(x)
}

# Bad
if(debug){
  show(x)
}
```

R Style Guide

- Whitespace
  - Parentheses
    - Place a space after () used for function arguments:

```
# Good
function(x) {}

# Bad
function (x) {}
function(x){}
```

R Style Guide

- Whitespace
  - Always surround =, ==, +, -, <- with a single space

```r
# Good
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

R Style Guide

- Whitespace
  - Always surround infix operators =, ==, +, -, <- with a single space
  - There are a few exceptions, which should NEVER be surrounded by spaces.
    - ::, $, @, [, [[, ^, :

```r
# Good
sqrt(x^2 + y^2)
df$z
x <- 1:10


# Bad
sqrt(x ^ 2 + y ^ 2)
df $ z
x <- 1 : 10
```

R Style Guide

- Whitespace and functions
  - Surround = with a space as a function parameter argument

```r
# Preferred
complex <- function(real, imag = 0) {
  return(magic(r = real, i = imag))
}

# Over this
complex <- function(real, imag=0) {
  return(magic(r=real, i=imag))
}
```

R Style Guide

- Compound statements/run-on lines

```r
17  if (x & y > 10) { print("something") }
```

```r
105 iris[iris$Species == 'setosa', c("Sepal.Length", "Sepal.Width")]
```

```r
108 setosa_rows <- iris$Species == 'setosa'
109 sepal_cols <- c("Sepal.Length", "Sepal.Width")
110 setosa_df <- iris[setosa_rows, sepal_cols]
```

Remember this?

```
1  library(stats); library(ggplot2)
2  fUncTion1 <- function(x, w) {
3    if (length(x) != length(w)) {
4      stop("`x` and `w` must be the same length", call. = FALSE)
5    }
6    sum(w * x) / sum(w)
7  }
8  fUnctioN2=function(x,conf=0.95) {
9      se=sd(x)/sqrt(length(x))
10     alpha=1-conf
11     mean(x)+se*qnorm(c(alpha/2,1-alpha/2))
12 }
13 x <- runif(100)
14 w <- 1:100
15 print(paste0('The weighted mean is: ', fUncTion1(x,w)))
16 print(paste0('The mean confidence interval is: ',fUnctioN2(x)[1], ' and ', fUnctioN2(x)[2]))
```

R Style play time, part 2!

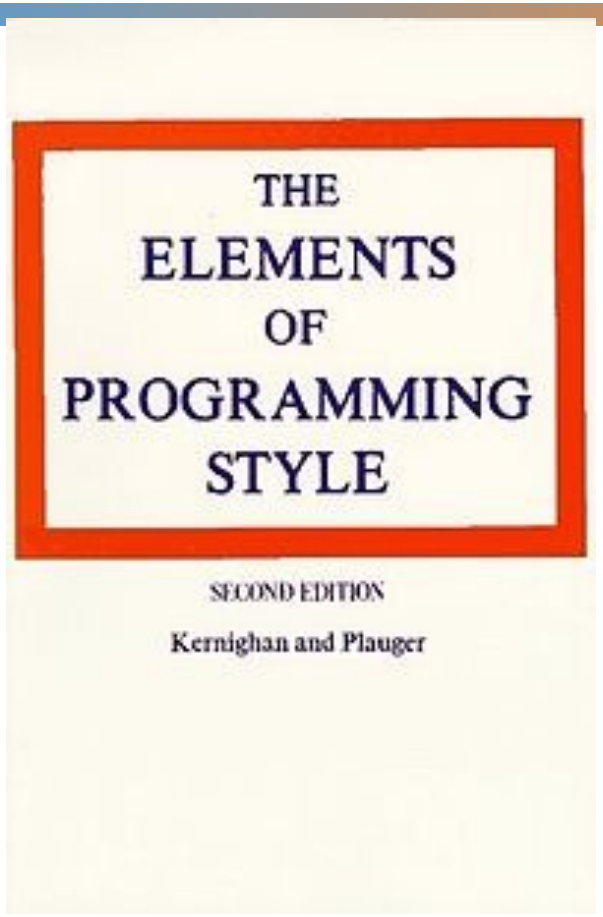
- Run lintr on the R file in RStudio,

  e.g. `lintr::lint("R_demo-1.R")`

- Try to clean things up using some of the rules discussed. Focus on “readability”, how many markers can you get rid of?

```r
library(stats); library(ggplot2)
fUncTion1 <- function(x, w) {
  if (length(x) != length(w)) {
    stop("`x` and `w` must be the same length", call. = FALSE)
  }
  sum(w * x) / sum(w)
}
fUnctioN2=function(x,conf=0.95) {
    se=sd(x)/sqrt(length(x))
    alpha=1-conf
    mean(x)+se*qnorm(c(alpha/2,1-alpha/2))
}
x <- runif(100)
w <- 1:100
print(paste0('The weighted mean is: ', fUncTion1(x,w)))
print(paste0('The mean confidence interval is: ',fUnctioN2(x)[1], ' and ', fUnctioN2(x)[2]))
```

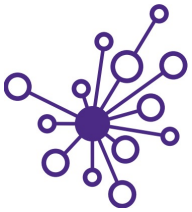
Elements of Programming Style



- 1974
- Fortran & PL/1[1]
- Most of the lessons are language free, e.g.
  - *Replace repetitive expressions by calls to a common [f]unction.*
  - *Choose variable names that won't be confused.*

1. Programming Language 1

Elements of Programming Style



- *Choose variable names that won't be confused.*

```
1  library(stats); library(ggplot2)
2  fUncTion1 <- function(x, w) {
3    if (length(x) != length(w)) {
4      stop("`x` and `w` must be the same length", call. = FALSE)
5    }
6    sum(w * x) / sum(w)
7  }
8  fUnctioN2=function(x,conf=0.95) {
9      se=sd(x)/sqrt(length(x))
10     alpha=1-conf
11     mean(x)+se*qnorm(c(alpha/2,1-alpha/2))
12 }
13 x <- runif(100)
14 w <- 1:100
15 print(paste0('The weighted mean is: ', fUncTion1(x,w)))
16 print(paste0('The mean confidence interval is: ',fUnctioN2(x)[1], ' and ', fUnctioN2(x)[2]))
```

R Style Guide

- Naming conventions
  - How you name functions, classes, and variables can have a huge impact on readability

```
1  library(stats); library(ggplot2)
2  fUncTion1 <- function(x, w) {
3    if (length(x) != length(w)) {
4      stop("`x` and `w` must be the same length", call. = FALSE)
5    }
6    sum(w * x) / sum(w)
7  }
8  fUnctioN2=function(x,conf=0.95) {
9      se=sd(x)/sqrt(length(x))
10     alpha=1-conf
11     mean(x)+se*qnorm(c(alpha/2,1-alpha/2))
12 }
13 x <- runif(100)
14 w <- 1:100
15 print(paste0('The weighted mean is: ', fUncTion1(x,w)))
16 print(paste0('The mean confidence interval is: ',fUnctioN2(x)[1], ' and ', fUnctioN2(x)[2]))
```

R Style Guide

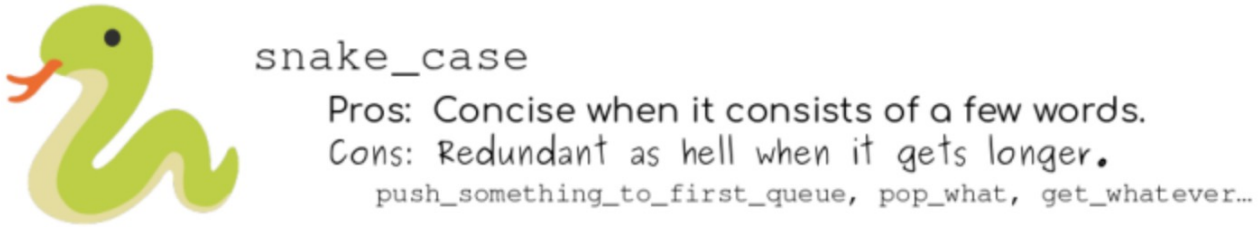
- Naming conventions
  - Avoid the following variable names:
    - Lower case L (l)
    - Upper case O (O)
    - Upper case I (I)
    - There are unacceptable, terrible, and awful. Why?
      - Can be confused with 1 and 0 in some fonts
      - Can be confused with each other (i.e. I and l)

R Style Guide

- Naming conventions
  - Module names should be short, lowercase
    - Underscores are OK if it helps with readability
  - Package names should be short, lowercase
    - Underscores are frowned upon and people will speak disparagingly behind your back if you use them

Tidyverse Style Guide

- Naming conventions
  - Variables and function names should be in lowercase.
  - Use an underscore (_) to separate words within a name.
    - Also known as snake_case



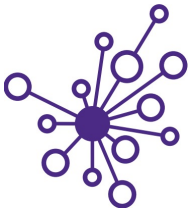
https://github.com/tidyverse/dplyr/blob/master/R/case_when.R

Google Recommendation

- Naming conventions
  - Function names should be in CapWords
    - SoNamedBecauseItUsesCapsForFirstLetterInEachWord
    - Also known as CamelCase
    - Notice no underscore!
  - Other objects
    - Moving away from dot!
    - dot.case

https://github.com/google/CausalImpact/blob/master/R/impact_analysis.R

Common Practices



- Naming conventions
  - Variables
    - Lowercase, with words separated by underscores as necessary to improve readability
    - mixedCase is permitted if that is the prevailing style
      - As seen in Bioconductor Coding Style
    - Easy habit to fall into… Very common in style guides for other languages.
      - If this is your thing, then be consistent

Tidyverse or Google or Other?

Why be consistent from day 1?

https://goo.gl/o57K7g

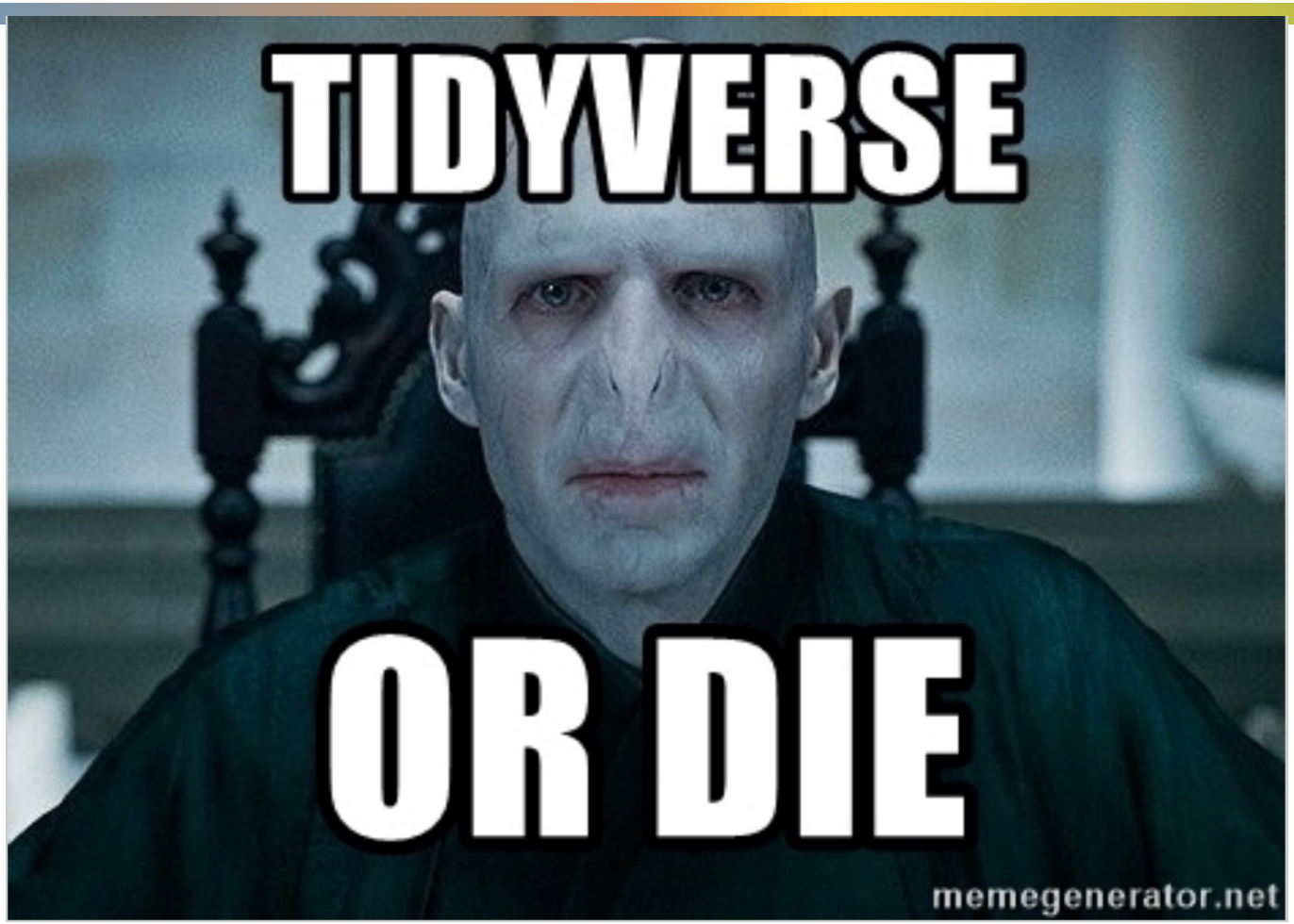
Tidyverse

- Programming Recommendations
  - These section are recommended
  - https://style.tidyverse.org/pipes.html
  - https://style.tidyverse.org/ggplot2.html

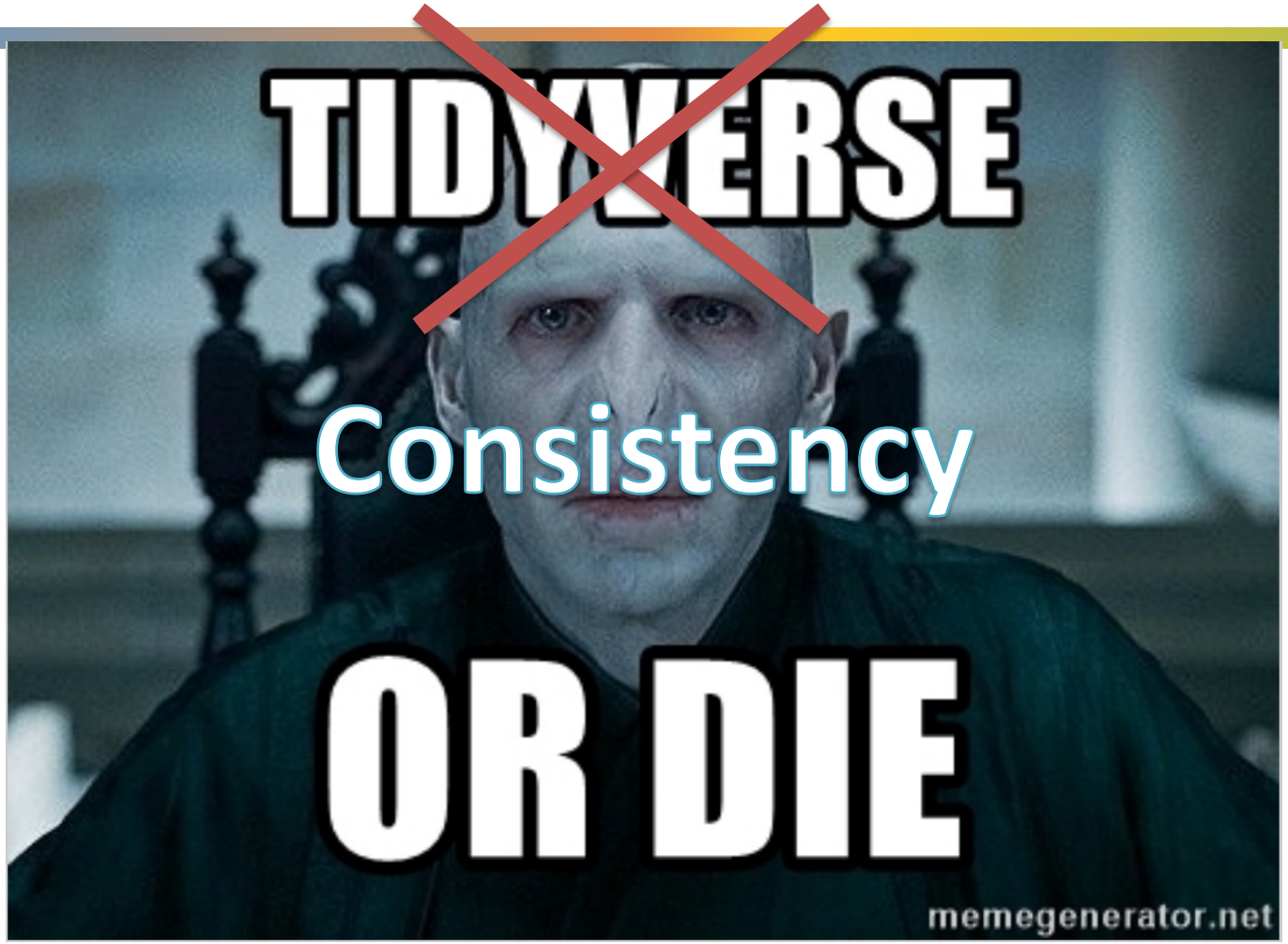
```r
# Good
iris %>%
  group_by(Species) %>%
  summarize_if(is.numeric, mean) %>%
  ungroup() %>%
  gather(measure, value, -Species) %>%
  arrange(value)

# Bad
iris %>% group_by(Species) %>% summarize_all(mean) %>%
ungroup %>% gather(measure, value, -Species) %>%
arrange(value)
```

```r
# Good
iris %>%
  filter(Species == "setosa") %>%
  ggplot(aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()

# Bad
ggplot(filter(iris, Species == "setosa"), aes(x = Sepal.Width, y = Sepal.Length)) +
  geom_point()
```

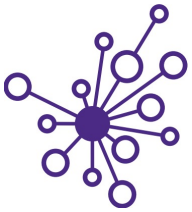
Tidyverse



Tidyverse



Documentation



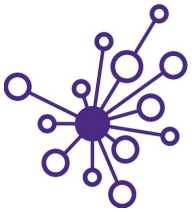
- Two types
  - Code readers
    - What the code is doing and why
      - E.g.

Code comments

  - Users
    - How to use your code
      - E.g.

README.md

Documentation



- .md
  - .md files are Markdown
  - Markdown is a lightweight text formatting language for producing mildly styled text
  - Ubiquitous (github.io, README.md, etc.)
  - E.g. Google markdown editor browser
    - http://dillinger.io
  - Tool: roxygen2 (https://github.com/r-lib/roxygen2)

Documentation

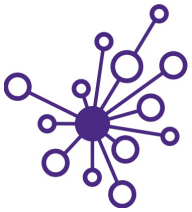
- What kind of stuff going in a repositories README.md?

https://github.com/kallisons/NOAH_LSM_Mussel_v2.0

Documentation

- Comments
  - Shell script
    - #
  - R
    - #

Documentation



Some examples of bad comments (from the ‘net)

```
%
% For the brave souls who get this far: You are the chosen ones,
% the valiant knights of programming who toil away, without rest,
% fixing our most awful code. To you, true saviors, kings of men,
% I say this: never gonna give you up, never gonna let you down,
% never gonna run around and desert you. Never gonna make you cry,
% never gonna say goodbye. Never gonna tell a lie and hurt you.
%
```

Don’t Rick Roll your readers!

```
% drunk, fix later
```

Uhm… Sigh.

```
%
% Dear maintainer:
%
% Once you are done trying to 'optimize' this routine,
% and have realized what a terrible mistake that was,
% please increment the following counter as a warning
% to the next guy:
%
% total_hours_wasted_here = 42
%
```

Funny is funny, but don’t troll. And what was the issue the writer encountered!

```
true = false;
% Happy debugging suckers
```

At least it is logical

http://stackoverflow.com/questions/184618/what-is-the-best-comment-in-source-code-you-have-ever-encountered

Documentation

- Good comments
  - Make the comments easy to read
  - Write the comments in English
  - Discuss the function parameters and results

```
211  % parameters:
212  %   sequence = character string of nucleotide letters (ATCG)
213  % returns:
214  %   geneStarts = vector of start index into sequence of start codon
215  %   geneEnds = vector of stop index into sequence of stop codons
216  %                value is the first base of the stop codon
217  function [ geneStarts, geneEnds ] = callGenesFromSequence(sequence)
218      ...
219  end
```

Documentation

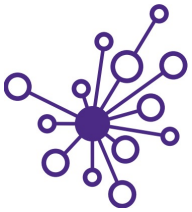
- Good comments
  - Don’t comment bad code, rewrite it!

```
223  % this is so terrible
224  % I can't find the bug here but, just subtract the length of x from
225  %   the result and divide by the length
226  function meanX = averageX(x)
227      meanX = sum(x) + length(x)
228  end
```

  - Then comment it

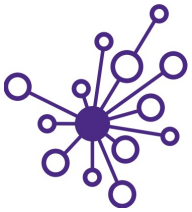
```
230  % parameters:
231  %   x = a vector of numerics
232  % returns:
233  %   meanX = the average of the vector x
234  function meanX = averageX(x)
235      meanX = sum(x) / length(x)
236  end
```

Documentation



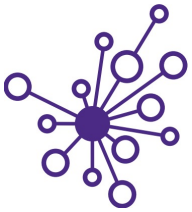
- Good comments
  - Some languages have special function headers

Documentation



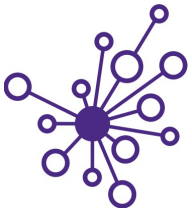
- Good comments
  - Some languages have special function headers
    - This example is fantastic!
    - It describes
      - Calling synopsis (example usage)
      - The input parameters
      - The output variables
      - Aimed at coders and users

Documentation



- Good comments
  - Some languages have special function headers
    - These comments should also describe **side effects**
      - Any global variables that might be altered
      - Plots that are generated
      - Output that is puked

Documentation / PEP8



- Good comments
  - Inline comments
    - Comments inline with the code

```
177   x = x + 1 # Increment x
```

    - Generally unnecessary (as above)
    - Inhibit readability

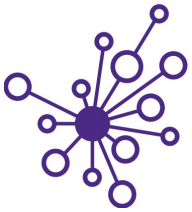
Documentation

- Good comments
  - Wrong comments are bugs

```
179     # Unit test for the sweepFiles function to test bounds
180     # checking on metadata parameters.
181     def test_sweepFiles_metadataType(self):
182         with self.assertRaises(TypeError):
183             io.sweepFiles('examples', metadata=41)
```

  - When updating code, don't forget to update the comments

Documentation



- Good comments
  - Don’t insult the reader

```
240   % compute the square root of the square of the distance
241   distance = sqrt(distanceSquared);
```

  - If they are reading your code… they aren’t that dumb

  - Corollary: don’t comment every line!

Documentation

- Good comments
  - Don’t comment every line!

```
57  # Find the square root of the 3D distance.
58  distance_squared <- (x2-X1)^2 + (y2-y1)^2 + (z2-z1)^2
59  # Compute the square root of the square of the distance.
60  distance <- sqrt(distance_squared)
61  # Make sure the distance is less than 3.5 Angstroms or error.
62  if (distance < 3.5) {
63    # Throw an error.
64    stop("interatomic distance is less than 3.5 Angstroms")
65  } else {
66    # Add the distance to the list of distances.
67    distances <- append(distances, distance)
68  }
```

Documentation

- Good comments
  - Problems with this code (other than excessive comments?)

```
57  # Find the square root of the 3D distance.
58  distance_squared <- (x2-X1)^2 + (y2-y1)^2 + (z2-z1)^2
59  # Compute the square root of the square of the distance.
60  distance <- sqrt(distance_squared)
61  # Make sure the distance is less than 3.5 Angstroms or error.
62  if (distance < 3.5) {
63    # Throw an error.
64    stop("interatomic distance is less than 3.5 Angstroms")
65  } else {
66    # Add the distance to the list of distances.
67    distances <- append(distances, distance)
68  }
```

Documentation

- Good comments
  - Problems with this code (other than excessive comments?)
    - What happens if I want to change the cutoff distance
      - I have to change the code (in 2 places)
      - I have to change the comment

```
57  # Find the square root of the 3D distance.
58  distance_squared <- (x2-X1)^2 + (y2-y1)^2 + (z2-z1)^2
59  # Compute the square root of the square of the distance.
60  distance <- sqrt(distance_squared)
61  # Make sure the distance is less than 3.5 Angstroms or error.
62  if (distance < 3.5) {
63    # Throw an error.
64    stop("interatomic distance is less than 3.5 Angstroms")
65  } else {
66    # Add the distance to the list of distances.
```

Documentation

- Good comments

```
71  # Find the square of the 3D distance and compute the sqrt,
72  #   then compare the distance to our cutoff (defined elsewhere)
73  #   and throw an error if the distance is too small
74  #   otherwise add the distance to our vector of distances.
75  distance_squared <- (x2 - X1)^2 + (y2 - y1)^2 + (z2 - z1)^2
76  distance <- sqrt(distance_squared)
77  if (distance < DISTANCE_CUTOFF) {
78    # Throw an error.
79    exception(
80      "DistanceViolation",
81      sprintf(
82        "interatomic distance is less than %.2f Angstroms",
83        DISTANCE_CUTOFF
84      )
85    )
86  } else {
```

- Note how the block is commented
- The code itself reads clearly enough
- We used an obviously marked constant whose value is displayed if an error is encountered

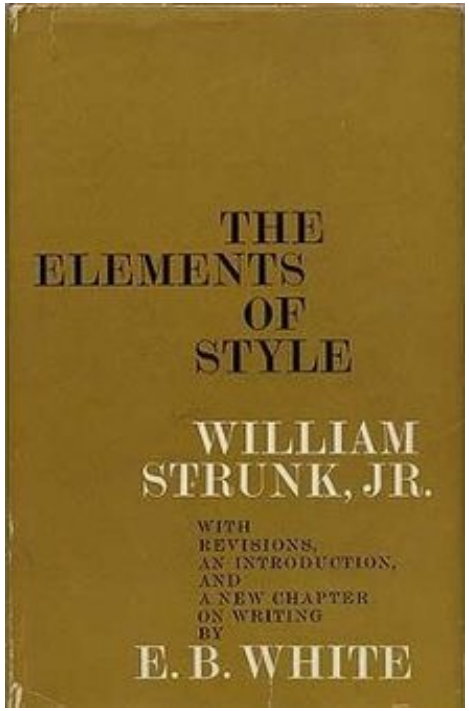
Documentation

- Good comments
  - Comments should be in sentence case. They should end with a period if they contain at least two sentences.

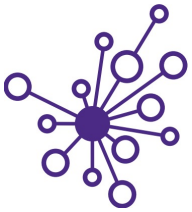
```
# Good

# Objects like data frames are treated as leaves
x <- map_if(x, is_bare_list, recurse)

# Do not use `is.list()`. Objects like data frames must be treated
# as leaves.
x <- map_if(x, is_bare_list, recurse)
```

Documentation / PEP8

- Good comments
  - Comments should be written in English, and follow Strunk and White.

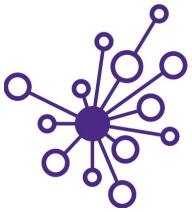


Documentation



- For modules and packages, list the classes, exceptions and functions (and any other objects) that are exported by the module, with a one-line summary of each.

Documentation / PEP 0257



- Docstrings
  - **Most importantly…** For functions and methods, it should summarize its behavior and document its arguments, return value(s), side effects, exceptions raised.

Feedback Survey