SOCIAL DATA SCIENCE

Tidy Data, Data Manipulation & Functions

Sebastian Barfort

August 04, 2016

University of Copenhagen
Department of Economics

- dplyr
- tidyr
- purrr
- tidytext
- stringr

"Herein lies the dirty secret about most data scientists' work – it's more data munging than deep learning. The best minds of my generation are deleting commas from log files, and that makes me sad. A Ph.D. is a terrible thing to waste."

Source

DATA JANITOR

TECHNOLOGY

For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights

By STEVE LOHR AUG. 17, 2014

Email

Share

Tweet

Save

More

Technology revolutions come in measured, sometimes foot-dragging steps. The lab science and marketing enthusiasm tend to underestimate the bottlenecks to progress that must be overcome with hard work and practical engineering.

The field known as "big data" offers a contemporary case study. The catchphrase stands for the modern abundance of digital data from many sources — the web, sensors, smartphones and corporate databases — that can be mined with clever software for discoveries and insights. Its promise is smarter, data-driven decision-making in every field. That is why data scientist is the economy's hot new job.

Yet far too much handcrafted work — what data scientists call "data wrangling," "data

Monica Rogati, Jawbone's vice president for data science, with Brian Wilt, a senior data scientist. Peter DaSilva for The New York Times

Source

Raw data

The original source of the data

Often hard to use directly for data analysis

You should *never* process your original data

Processed data

Data that is ready for analysis

Data manipulation involves going from *raw* to *processed* data.

This can include merging, subsetting, transforming, etc.

All steps that take you from raw to processed data should be scripted

Introduce some tricks for working (efficiently) with data

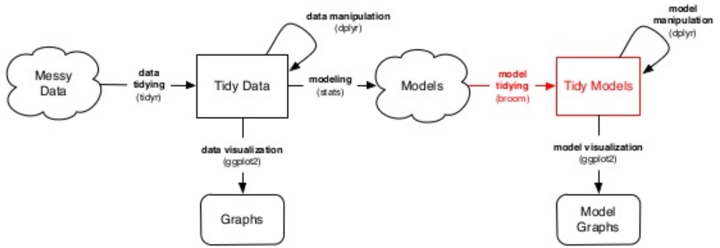
Introduce concept and tools for working with tidy data (`tidyr`)

Manipulate tidy data using `dplyr`

Iterate over elements using functions (`purrr`)

String processing (`stringr`, regular expressions)

WORKFLOW



Messy Data
data tidying (tidyr)
Tidy Data
data manipulation (dplyr)
modeling (stats)
Models
model tidying (broom)
Tidy Models
model manipulation (dplyr)
data visualization (ggplot2)
Graphs
model visualization (ggplot2)
Model Graphs

The Pipe

DIGRESSION: THE PIPE

magrittr::

%>%

The pipe operator `%>%` (RStudio has keyboard shortcuts, learn to use them!) let's you write sequences instead of nested functions

`x %>% f(y)` -> `f(x,y)`

`x %>% f(z, .)` -> `f(z, x)`

Read `%>%` as "then". First do this, *then* do this, etc...

It's implemented in R by a Danish econometrician

All the packages you will learn today work with the pipe.

INTUITION

```r
enjoy(cool(bake(shape(beat(append(bowl(rep("flour", 2),
"yeast", "water", "milk", "oil"), "flour", until =
"soft"), duration = "3mins"), as = "balls", style =
"slightly-flat"), degrees = 200, duration = "15mins"),
duration = "5mins"))
```

```r
bowl(rep("flour", 2), "yeast", "water", "milk", "oil") %>%
    append("flour", until = "soft") %>%
    beat(duration = "3mins") %>%
    shape(as = "balls", style = "slightly-flat") %>%
    bake(degrees = 200, duration = "15mins") %>%
    cool(buns, duration = "5mins") %>%
    enjoy()
```

source

Tidy data

`tidyr`

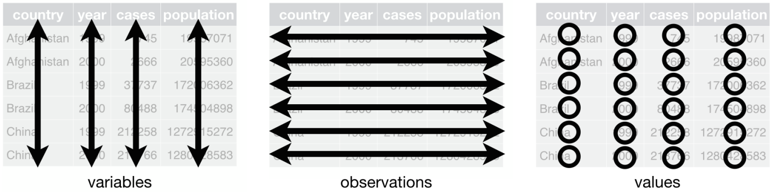
Tidy data: observations are in the rows, variables are in the columns

`tidyr`: take your messy data and turn it into a tidy format

Advantages of tidy data:

- Consistency
- Allows you to spend more time on your analysis
- Speed

TIDY DATA


country
year
cases
population
variables
observations
values

- `gather`: Reshape from wide to long
- `spread`: Reshape from long to wide
- `separate`: Split a variable into multiple variables.

(Also more complicated functions such as `nest` for nested data frames, but we won't go into detail with those here)

READ DATA

```
library("readr")
gh.link = "https://raw.githubusercontent.com/"
user.repo = "hadley/tidyr/"
branch = "master/"
link = "vignettes/pew.csv"
data.link = paste0(gh.link, user.repo, branch, link)
df = read_csv(data.link)
```

First five columns

religion	<$10k	$10-20k	$20-30k	$30-40k
Agnostic	27	34	60	81
Atheist	12	27	37	52
Buddhist	27	21	30	34

Question 1: What variables are in this dataset?

Question 2: How does a tidy version of this data look like?

The `gather` function

Problem: Column names are not names of a variable, but *values* of a variable.

Objective: Reshaping wide format to long format

To tidy such data, we need to **gather** the non-variable columns into a two-column key-value pair

gather

Three parameters

1. Set of columns that represent values, not variables.
2. Name of the variable whose values form the column names (key).
3. The name of the variable whose values are spread over the cells (value.

```
library("tidyr")
args(gather)
```

```
## function (data, key, value, ..., na.rm = FALSE, conve
##     factor_key = FALSE)
## NULL
```

gather PEW DATA

```
df.gather = df %>%
  gather(key = income,
         value = frequency,
         -religion)
```

religion	income	frequency
Agnostic	<$10k	27
Atheist	<$10k	12
Buddhist	<$10k	27
Catholic	<$10k	418
Don’t know/refused	<$10k	15

This

```
df %>%
  gather(key = income,
         value = frequency,
         2:11)
```

returns the same as

```
df %>%
  gather(key = income,
         value = frequency,
         -religion)
```

Billboard data

```
library("readr")
gh.link = "https://raw.githubusercontent.com/"
user.repo = "hadley/tidyr/"
branch = "master/"
link = "vignettes/billboard.csv"
data.link = paste0(gh.link, user.repo, branch, link)
df = read_csv(data.link)
```

```
df[1:5, 1:5]
```

year	artist	track	time	date.entered
2000	2 Pac	Baby Don't Cry (Keep...	4:22	2000-02-26
2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02
2000	3 Doors Down	Kryptonite	3:53	2000-04-08
2000	3 Doors Down	Loser	4:24	2000-10-21
2000	504 Boyz	Wobble Wobble	3:35	2000-04-15

```
df[1:5, 6:10]
```

wk1	wk2	wk3	wk4	wk5
87	82	72	77	87
91	87	92	NA	NA
81	70	68	67	66
76	76	72	69	67
57	34	25	17	17

Question: what are the variables here?

To tidy this dataset, we first gather together all the `wk` columns. The column names give the week and the values are the ranks:

```
billboard2 = df %>%
  gather(key = week,
         value = rank, wk1:wk76,
         na.rm = TRUE)
```

Not displaying the `track` column

year	artist	time	date.entered	week	rank
2000	2 Pac	4:22	2000-02-26	wk1	87
2000	2Ge+her	3:15	2000-09-02	wk1	91
2000	3 Doors Down	3:53	2000-04-08	wk1	81
2000	3 Doors Down	4:24	2000-10-21	wk1	76
2000	504 Boyz	3:35	2000-04-15	wk1	57

Are we done?

Let's turn the week into a numeric variable and create a proper date column

```
library("dplyr")
billboard3 = billboard2 %>%
  mutate(
    week = extract_numeric(week),
    date = as.Date(date.entered) + 7 * (week - 1)) %>%
  select(-date.entered) %>%
  arrange(artist, track, week)
```

What functions from `tidyr` did we use here?

year	artist	track	time	week
2000	2 Pac	Baby Don’t Cry (Keep...	4:22	1
2000	2 Pac	Baby Don’t Cry (Keep...	4:22	2
2000	2 Pac	Baby Don’t Cry (Keep...	4:22	3
2000	2 Pac	Baby Don’t Cry (Keep...	4:22	4
2000	2 Pac	Baby Don’t Cry (Keep...	4:22	5

After gathering columns, the key column is sometimes a combination of multiple underlying variable names.

```
library("readr")
gh.link = "https://raw.githubusercontent.com/"
user.repo = "hadley/tidyr/"
branch = "master/"
link = "vignettes/tb.csv"
data.link = paste0(gh.link, user.repo, branch, link)
df = read_csv(data.link)
```

iso2	year	m04	m514	m014	m1524	m2534	m3544
AD	1989	NA	NA	NA	NA	NA	NA
AD	1990	NA	NA	NA	NA	NA	NA
AD	1991	NA	NA	NA	NA	NA	NA
AD	1992	NA	NA	NA	NA	NA	NA
AD	1993	NA	NA	NA	NA	NA	NA

Question: what are the variables here?

ANSWER

The dataset comes from the World Health Organisation, and records the counts of confirmed tuberculosis cases by country, year, and demographic group. The demographic groups are broken down by sex (m, f) and age (0-14, 15-25, 25-34, 35-44, 45-54, 55-64, unknown).

```r
tb2 = df %>%
  gather(demo, n, -iso2, -year, na.rm = TRUE)
```

iso2	year	demo	n
AD	2005	m04	0
AD	2006	m04	0
AD	2008	m04	0
AE	2006	m04	0
AE	2007	m04	0

Is this dataset tidy?

SEPARATING THE demo VARIABLE

`separate` makes it easy to split a variable into multiple variables. You can either pass it a regular expression to split on or a vector of character positions. In this case we want to split after the first character.

```
tb3 = tb2 %>%
  separate(demo, c("sex", "age"), 1)
```

iso2	year	sex	age	n
AD	2005	m	04	0
AD	2006	m	04	0
AD	2008	m	04	0
AE	2006	m	04	0
AE	2007	m	04	0

There are times when we are required to turn long formatted data into wide formatted data. The `spread` function spreads a key-value pair across multiple columns.

`spread` IN ACTION

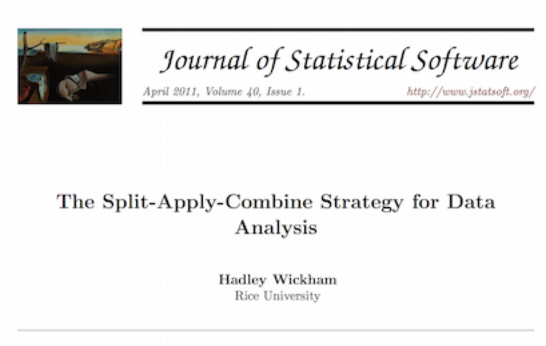
```
tb3.wide = tb3 %>% spread(age, n)
```

iso2	year	sex	014	04	1524	2534	3544
AD	1996	f	0	NA	1	1	0
AD	1996	m	0	NA	0	0	4
AD	1997	f	0	NA	1	2	3
AD	1997	m	0	NA	0	1	2
AD	1998	m	0	NA	0	0	1

Data Manipulation

THE SPLIT-APPLY-COMBINE APPROACH

Once you have your data stored in tidy form, you can easily apply a split-apply-combine strategy, where you break up a big problem into manageable pieces, opereate on each piece independently and then put the pieces back together

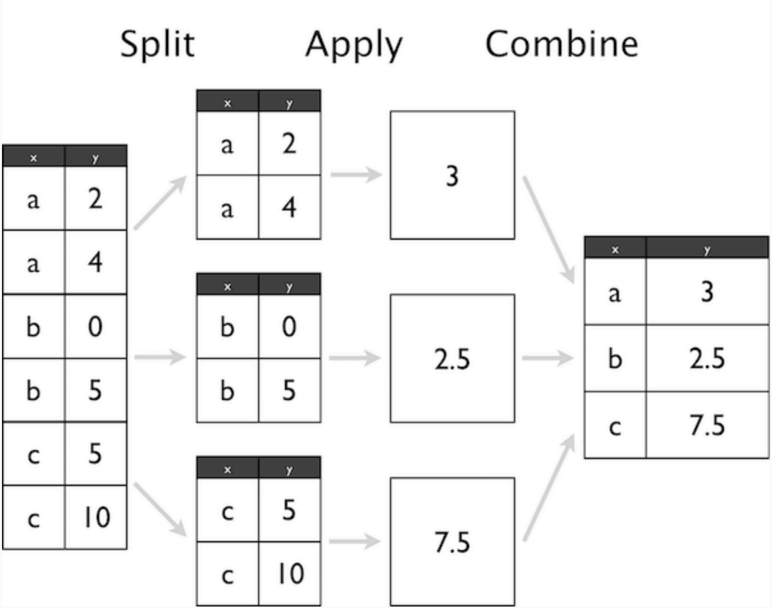


Journal of Statistical Software

April 2011, Volume 40, Issue 1. http://www.jstatsoft.org/

The Split-Apply-Combine Strategy for Data Analysis

Hadley Wickham
Rice University


Split
Apply
Combine
x
y
a
2
a
4
b
0
b
5
c
5
c
10
x
y
a
2
a
4
x
y
b
0
b
5
x
y
c
5
c
10
3
2.5
7.5
x
y
a
3
b
2.5
c
7.5

THE dplyr PACKAGE

dplyr: (efficiently) split-apply-combine for data frames

Verbs a verb is a function that takes a data frame as it's first argument

- filter: select rows
- arrange: order rows
- select: select columns
- rename: rename columns
- distinct: find distinct rows
- mutate: add new variables
- summarise: summarize across a data set
- sample_n: sample from a data set

In this part of the lecture we will work with the Danish federal budget proposal for 2016

```
library("readr")
library("dplyr")
gh.link = "https://raw.githubusercontent.com/"
user.repo = "sebastianbarfort/sds_summer/"
branch = "gh-pages/"
link = "data/finanslov_tidy.csv"
data.link = paste0(gh.link, user.repo, branch, link)
df = read_csv(data.link)
```

Some nice guy has already cleaned this data for you

Try yourself

```
View(df)
glimpse(df)
summary(df)
head(df)
```

`filter` return rows with matching conditions.

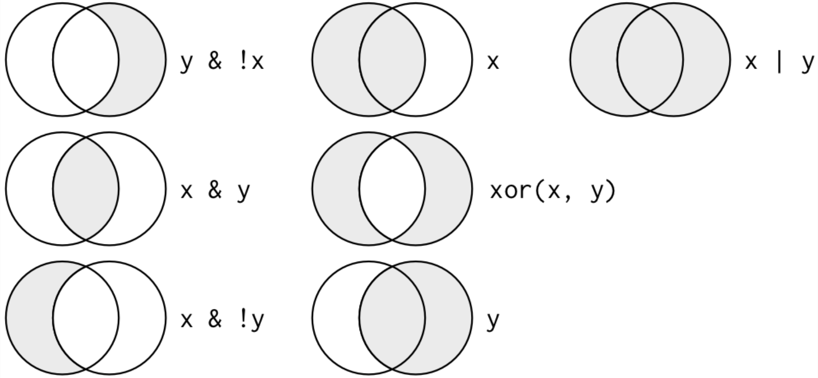
```
df.max.udgift = df %>%
  filter(udgift == max(udgift)) %>%
  select(paragraf, aar, udgift)
```

paragraf	aar	udgift
Beskæftigelsesministeriet	2018	132541.4

FILTERING DATA II

```
df.skat = df %>%
  filter(paragraf == "Skatter og afgifter") %>%
  select(paragraf, aar, udgift) %>%
  arrange(-udgift)
```

paragraf	aar	udgift
Skatter og afgifter	2014	14487.1
Skatter og afgifter	2015	14401.6
Skatter og afgifter	2016	14386.2
Skatter og afgifter	2014	185.9
Skatter og afgifter	2015	185.9


y & !x
x
x | y
x & y
xor(x, y)
x & !y
y

`mutate` let's you add new variables to your data frame

```
df.mutated = df %>%
  mutate(newVar = udgift/2) %>%
  select(newVar, udgift)
```

newVar	udgift
38.85	77.7
13.20	26.4
193.90	387.8
132.10	264.2
3.25	6.5

SAMPLING FROM A DATA FRAME

We can sample from a data frame using `sample_n` and `sample_frac`

```
df.sample.n = df %>%
  select(paragraf, aar, udgift) %>%
  sample_n(3)
```

paragraf	aar	udgift
Sundheds- og Ældreministeriet	2018	0.0
Social- og Indenrigsministeriet	2018	0.0
Social- og Indenrigsministeriet	2015	2.4

GROUPED OPERATIONS

So far, we have primarily learned how to manipulate data frames.

The `dplyr` package becomes really powerful when we introduce the `group_by` function

`group_by` breaks down a dataset into specified groups of rows. When you then apply the verbs above on the resulting object they'll be automatically applied "by group".

Use in conjunction with `mutate` (to add existing rows to your data frame) or `summarise` (to create a new data frame)

COMMON `mutate`/`summarise` OPTIONS

- `mean`: mean within groups
- `sum`: sum within groups
- `sd`: standard deviation within groups
- `max`: max within groups
- `n()`: number in each group
- `first`: first in group
- `last`: last in group
- `nth(n = 3)`: nth in group (3rd here)
- `tally`: count number in group

Which ministry has the largest expenses?

```
df.expense = df %>%
  group_by(paragraf) %>%
  summarise(sum.exp = sum(udgift, na.rm = TRUE)) %>%
  arrange(-sum.exp)
```

paragraf	sum.exp
Social- og Indenrigsministeriet	1231214.6
Beskæftigelsesministeriet	1146997.8
Uddannelses- og Forskningsministeriet	296539.2
Min. for Børn, Undervisning og Ligestilling	180955.1
Pensionsvæsenet	139058.0

Add `sum.exp` to existing data frame

```
df.2 = df %>%
  group_by(paragraf) %>%
  mutate(sum.exp = sum(udgift, na.rm = TRUE)) %>%
  select(paragraf, udgift, sum.exp)
```

paragraf	udgift	sum.exp
Dronningen	77.7	474.7
Medlemmer af det kongelige hus m.fl.	26.4	161.2
Folketinget	387.8	6137.6
Folketinget	264.2	6137.6
Folketinget	6.5	6137.6

You can group by several variables

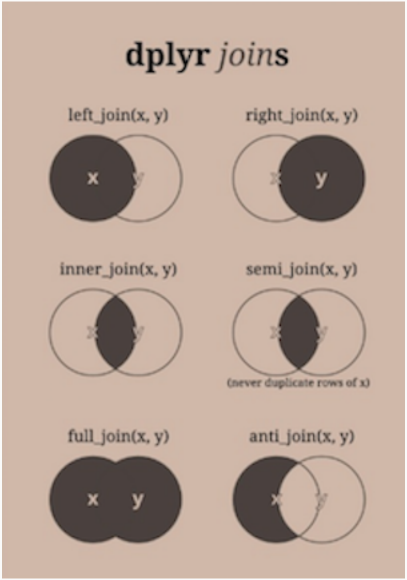
```
df.expense.2 = df %>%
  group_by(paragraf, aar) %>%
  summarise(sum.exp = sum(udgift, na.rm = TRUE)) %>%
  arrange(sum.exp)
```

paragraf	aar	sum.exp
Afdrag på statsgælden (netto)	2016	-77832.3
Afdrag på statsgælden (netto)	2015	-32519.9
Afdrag på statsgælden (netto)	2017	0.0
Afdrag på statsgælden (netto)	2018	0.0
Afdrag på statsgælden (netto)	2019	0.0

Let's first calculate yearly expenses at the `paragraf` level and then calculate mean expenses over the years.

```
df.expense.3 = df %>%
  group_by(paragraf, aar) %>%
  summarise(exp = sum(udgift, na.rm = TRUE)) %>%
  ungroup() %>%
  group_by(paragraf) %>%
  summarise(sum.exp = mean(exp, na.rm = TRUE))
```

paragraf	sum.exp
Afdrag på statsgælden (netto)	-14628.13333
Beholdningsbevægelser mv.	1540.65000
Beskæftigelsesministeriet	191166.30000
Dronningen	79.11667
Energi-, Forsynings- og Klimaministeriet	2433.18333


dplyr joins
left_join(x, y)
x
y
right_join(x, y)
x
y
inner_join(x, y)
x
y
semi_join(x, y)
x
y
(never duplicate rows of x)
full_join(x, y)
x
y
anti_join(x, y)
x
y

Look at this dataset

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

And this

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

```
ijsp = inner_join(superheroes, publishers)
```

name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934

```
ljsp = left_join(superheroes, publishers)
```

name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934
Hellboy	good	male	Dark Horse Comics	NA

```r
superheroes = superheroes %>%
  mutate(seblikes = (publisher == "Marvel"))
publishers = publishers %>%
  mutate(seb = (publisher == "Marvel"))
ij2 = inner_join(superheroes,publishers)
```

```
## Joining by: "publisher"
```

name	alignment	gender	publisher	seblikes	yr_founded	seb
Magneto	bad	male	Marvel	TRUE	1939	TR
Storm	good	female	Marvel	TRUE	1939	TR
Mystique	bad	female	Marvel	TRUE	1939	TR
Batman	good	male	DC	FALSE	1934	FAL
Joker	bad	male	DC	FALSE	1934	FAL
Catwoman	bad	female	DC	FALSE	1934	FAL

```r
ij2 = inner_join(superheroes, publishers,
                    by=c("publisher"="publisher",
                         "seblikes"="seb"))
```

name	alignment	gender	publisher	seblikes	yr_founded
Magneto	bad	male	Marvel	TRUE	1939
Storm	good	female	Marvel	TRUE	1939
Mystique	bad	female	Marvel	TRUE	1939
Batman	good	male	DC	FALSE	1934
Joker	bad	male	DC	FALSE	1934
Catwoman	bad	female	DC	FALSE	1934

```
fj = superheroes %>%
  full_join(publishers)
```

name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934
Hellboy	good	male	Dark Horse Comics	NA
NA	NA	NA	Image	1992

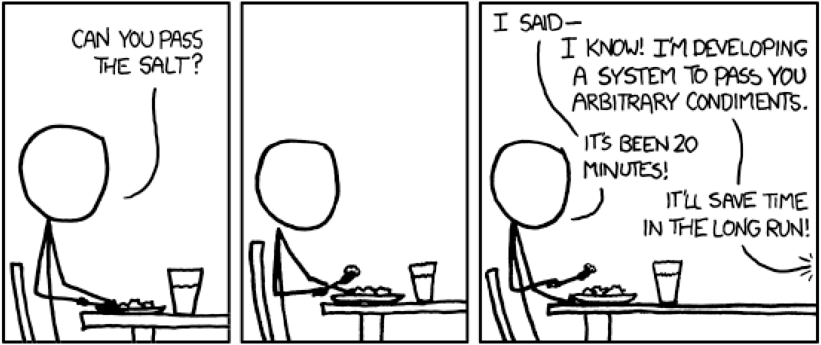
Functions and Iteration

Perhaps most important skill for being effective when working with data: write functions.

Advantages

1. You drastically reduce risk of making mistakes
2. When something exogenous changes, you only need to update code in one place
3. You can give your function an intuitive name that makes your code easier to read

You should write functions to increase your productivity.


CAN YOU PASS
THE SALT?
I SAID—
I KNOW! I'M DEVELOPING
A SYSTEM TO PASS YOU
ARBITRARY CONDIMENTS.
IT'S BEEN 20
MINUTES!
IT'LL SAVE TIME
IN THE LONG RUN!

TEMPLATE

```
my_function = function(input1, input2, ..., inputN)
  {
  # define 'output' using input1,...,inputN
  return(output)
  }
```

EXAMPLE

```r
add_numbers = function(x, y){
  z = x + y
  return(z)
}

add_numbers(2, 4)
```

```
## [1] 6
```

```r
add_numbers(2, 6)
```

```
## [1] 8
```

Now what

```
add_numbers(2, "y")
```

```
## Error in x + y: non-numeric argument to binary operat
```

```
add_numbers = function(x, y){
  if ( !is.numeric(x) || !is.numeric(y)) {
    warning("either 'x' or 'y' is not numeric")
    return(NA)
  }
  else {
    z = x + y
    return(z)
  }
}
```

TESTING

```r
add_numbers(2, 4)
```

```
## [1] 6
```

```r
add_numbers(2, ”y”)
```

```
## Warning in add_numbers(2, ”y”): either ’x’ or ’y’ is
```

```
## [1] NA
```

One important skill for being an effective data analyst was being able to write functions. A second is iteration.

Iteration helps you when you need to do the same thing to multiple inputs. For example, repeating the same function on lots of inputs.

Two iteration paradigms

1. Imperative programming (`for` loops, etc.)
2. Functional programming

Imagine that we have this data frame, called `df`

a	b	c	d
0.5694764	0.7846639	0.0344072	0.1333072
-0.3449869	1.1177386	0.4828237	0.2284890
1.3862865	-0.1598630	0.6252824	2.4510436
-2.0722838	0.1070453	0.0797690	0.5427930
-0.3469432	-0.6757737	0.4860079	0.0576001

And assume that we want to compute the mean of each column

THE `for` LOOP

We could iterate through each column, compute the mean and output the results

```
output = vector()
for (i in 1:ncol(df)){
  output[[i]] = mean(df[, i], na.rm = TRUE)
}
output
```

```
## [1] -0.1616902  0.2347622  0.3416580  0.6826466
```

String Processing