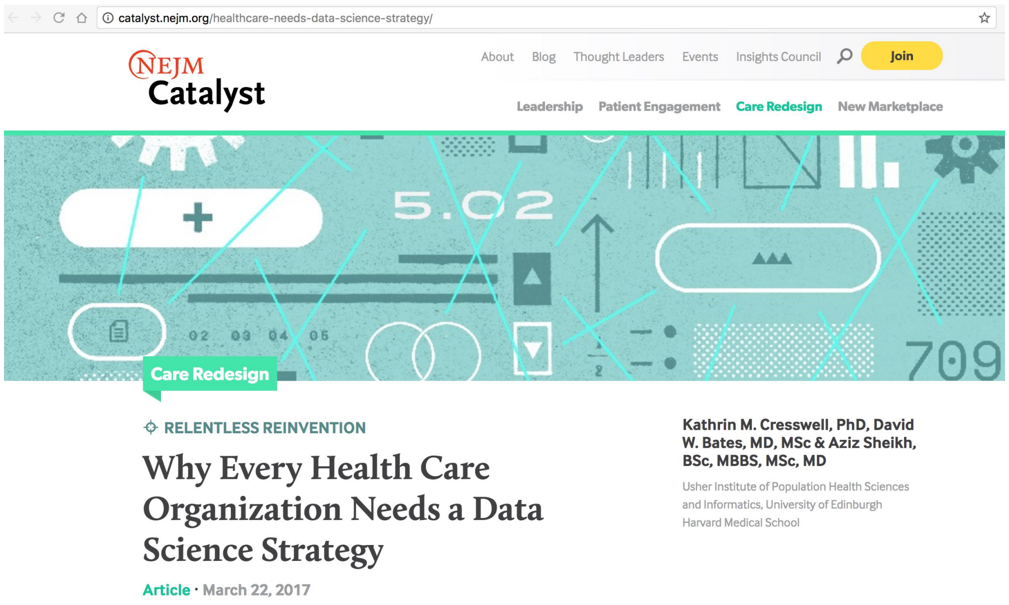
Lecture 2: The Data and the Mess

David Shilane

Data Science in Health Care



RELENTLESS REINVENTION

Why Every Health Care Organization Needs a Data Science Strategy

Article · March 22, 2017

Kathrin M. Cresswell, PhD, David W. Bates, MD, MSc & Aziz Sheikh, BSc, MBBS, MSc, MD

Usher Institute of Population Health Sciences and Informatics, University of Edinburgh
Harvard Medical School

Why Focus On Health Care?

Health care is a field with many laudable goals. It’s also a great place for data scientists, with:

- Opportunities to extend and improve people’s lives
- A Huge Industry: 17% of the USA’s GDP
- Myriad applications of essentially every method and technology in data science.

Data Science Roles in Health Care

There are many avenues to work as a data scientist in health care:

- Scientists
- Biostatisticians
- Health economists
- Developing products from data
- Everything else (marketing, operations, finance, etc.)

Areas of Health Care

Populations	Conditions	Medical Practice
Infants	Heart Disease	Laboratory Tests
Children	Diabetes	Diagnoses
Expecting Mothers	Cancer	Vaccines
Elderly	Asthma	Medicines
Differently Abled	Arthritis	Devices
Geographic	Fractures	Interventions
Vulnerable	Mental Health	Therapies
Impoverished	Dental	Wellness
Homeless	Nutrition	Nutrition

The Patient’s Privacy Is Paramount

Federal regulations mandate strong control of a patient’s personal information.

- HIPAA (the Health Insurance Portability and Accountability Act) mandates strong legal protections.
- There are many restrictions on sharing Personal Health Information.
- Security breaches have legal and financial ramifications.

Data Security Best Practices

Many of these practices apply to other fields of Data Science, too.

- Keep sensitive data in secure, centralized repositories.
- Use unique passwords (and a password manager) to lock your files or folders.
- Do not send personally identifying information by e-mail.
- Work with de-identified data files whenever possible.
- Keep identifying information separate from de-identified files.

Deep Dive #1: The 2014 NYC Community Health Survey

- https://www1.nyc.gov/site/doh/data/data-sets/community-health-survey-public-use-data.page
- The original data on the site is in SAS format.
- The course website includes a copy of the data in CSV format along with a full description of the survey (PDF).

Reading Files in R

File Extension	Library	Function	Notes
CSV	data.table	fread	Most efficient
TXT	data.table	fread	Most efficient
XLS	readxl	read_xls	Can specify the sheet name
XLSX	readxl	read_xlsx	Can specify the sheet name
XLS	excel.link	xl.read.file	Can open password protected files
XLSX	excel.link	xl.read.file	Can open password protected files
JSON	rjson	fromJSON	
SPSS (.sav)	foreign	read.spss	Very slow
SAS	sas7bdat	read.sas7bdat	Very slow
Postgre SQL	RPostgreSQL	dbGetQuery	Requires ports and security protocols

Reading In the Survey’s Data

```r
library(data.table)
# This assumes that you've set your working directory to
# the source file's location.
the.file <- "NYC 2014 Community Health Survey.csv"
dat <- fread(input = the.file)
```

Sample Size, Number of Variables

```
dim(dat)
```

```
[1] 8562  175
```

```
dat[, .N]
```

```
[1] 8562
```

```
ncol(dat)
```

```
[1] 175
```

Exploring Some Variables

```
dat[, .N, keyby = maritalstatus14]
```

```
   maritalstatus14    N
1:              NA   74
2:               1 3266
3:               2 1038
4:               3  734
5:               4  489
6:               5 2486
7:               6  475
```

```
dat[, .N, keyby = tolddepression]
```

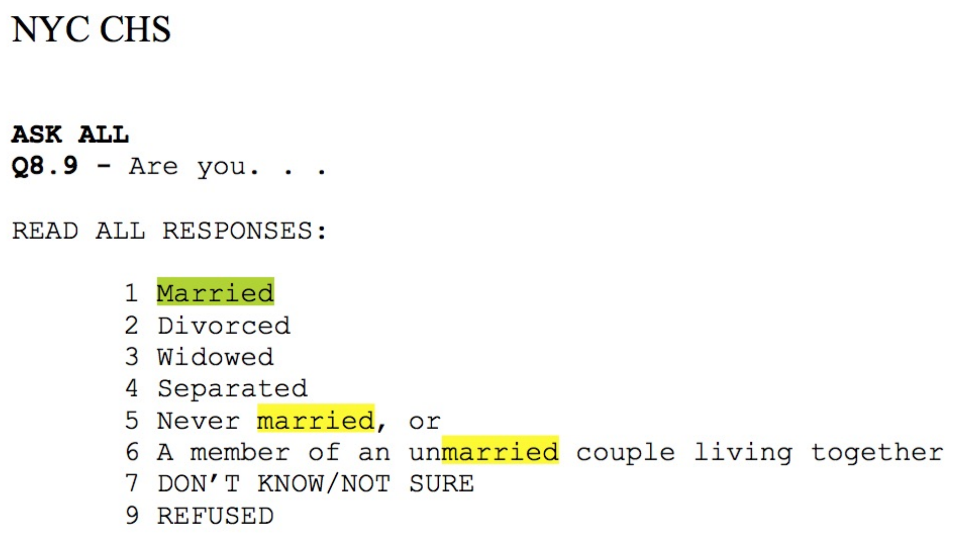
```
   tolddepression    N
1:             NA   36
2:              1 1268
3:              2 7258
```

What do these values mean?

Data Dictionaries

- Data sets should include documentation describing all of the variables used.
- The meaning of each value should be clearly explained.
- Any intricacies or special cases should be noted.
- Unfortunately, it's rare to even have a data dictionary, and the qualities of them can vary considerably.

Marital Status



NYC CHS

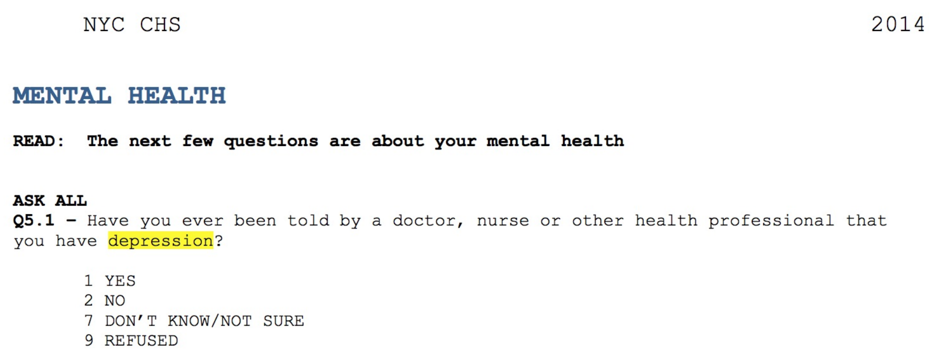
ASK ALL
Q8.9 - Are you. . .

READ ALL RESPONSES:

1 Married
2 Divorced
3 Widowed
4 Separated
5 Never married, or
6 A member of an unmarried couple living together
7 DON'T KNOW/NOT SURE
9 REFUSED

- This is reasonably straightforward. However, the file never mentions the specific name of the variable!

The Depression Status Variable



NYC CHS 2014

MENTAL HEALTH

READ: The next few questions are about your mental health

ASK ALL
Q5.1 – Have you ever been told by a doctor, nurse or other health professional that you have depression?

```
1 YES
2 NO
7 DON'T KNOW/NOT SURE
9 REFUSED
```

- This is also reasonably straightforward. However, the file never mentions the specific name of the variable!

Why Numeric Categories?

- It made sense in the old days of limited storage.
- Perhaps database designers would say it’s a good hashing practice.
- That may be so, but cryptically encoding variables **creates a barrier to understanding their meaning**. The best encoding is one that is self-explanatory to anyone who uses the data.

On The Other Hand: Security

- Cryptically encoded variables are, well, cryptic.
- If the data's security is breached, this provides additional protection.
- However, you could ensure security with other protocols (e.g. a password for the file) while maintaining the clarity for those with access.

Data Cleaning

- This is the process of making the data more amenable to analysis.
- **Investigation** is paramount.
- You never know what you might find.

Don’t Say I Didn’t Warn You

The Lament Of The Data Scientist

“80-90% of my time on a project is spent on cleaning up messy data sets. I would really like to spend more time doing analyses.”

– Origin: Every data scientist out there.

Cleaning Marital Status

- An approach based on **merging** tables to update the values.

```
old.marital.status.name <- "maritalstatus14"
marital.status.name <- "Marital Status 14"
marital.status.values <- c("1: Married", "2: Divorced",
    "3: Widowed", "4: Separated", "5: Never Married", "6: Cohabiting")
marital.status.map <- data.table(maritalstatus14 = 1:6,
    `Marital Status 14` = marital.status.values)

orig.dat <- dat
dat <- merge(x = orig.dat, y = marital.status.map, by = old.marital.status.name,
    all.x = TRUE, all.y = FALSE)
```

The Clean Variable

```
dat[, .N, marital.status.name]
```

```
   Marital Status 14    N
1:              <NA>   74
2:        1: Married 3266
3:       2: Divorced 1038
4:        3: Widowed  734
5:      4: Separated  489
6:  5: Never Married 2486
7:     6: Cohabiting  475
```

Cleaning Depression Status

```
dat[, depression.cleaned := 1*(tolddepression == 1)]
dat[, .N, keyby = c("tolddepression", "depression.cleaned")]
```

```
   tolddepression depression.cleaned    N
1:             NA                 NA   36
2:              1                  1 1268
3:              2                  0 7258
```

Changing the Variables' Names

```r
the.year = "14"
names.14 <- names(dat)[grep(pattern = the.year, x = names(dat))]
new.names <- gsub(pattern = the.year, replacement = "",
    x = names.14)
setnames(x = dat, old = names.14, new = new.names)
the.names <- data.table(Old = names.14, New = new.names)
the.names[1:5]
```

```
                 Old            New
1:  maritalstatus14  maritalstatus
2: insuredgateway14 insuredgateway
3:         insure14         insure
4:            pcp14            pcp
5:     sickadvice14     sickadvice
```

Why did I take out the year? In anticipation of aggregating the same variables from different years of data from the NYC CHS.

Common Errors and Messes

These examples came from a recent analysis:

- Typos: How many ways can people spell Brooklyn?

```
 [1] "Bbrooklyn"          "BK"                 "Booklyn"
 [4] "Brooklkyn"          "Brooklny"           "Brooklun"
 [7] "Brookly"            "brooklyn"           "Brooklyn"
[10] "BROOKLYN"           "Brooklyn NY"        "BROOKLYN NY"
[13] "BROOKLYN,"          "BROOKLYN, NY 11238" "Brookyln"
[16] "Brookyn"            "Broolyn"            "Bropoklyn"
[19] "brroklyn"
```

- Different Formats

```
[1] "Male"      "Female"    "M"         "F"         "m"         "f"
[7] "masculine" "fem"
```

What to Do About Messy Data

Every small mistake in entering the data eventually works its way to you. What should you do about it?

- For small issues, it’s better to clean them up yourself.
- For ongoing efforts of data collection, your team might want to refine its processes.
- Dropdown menus, limits on variables… any effort at prevention can help.
- The more other members of your team are aware of these issues, the more they will understand the efforts you are taking to help them.

One Bad Apple

Consider a data set of families tracking the number of cars and children in each:

```
num.children.name <- "num_children"
dat.family[1:5, get(num.children.name)]
```

```
[1] "3" "4" "0" "4" "1"
```

What turned the number of children into a character variable?

Finding the Bad Apples

```
dat.family[, sort(unique(num_children))]
```

```
[1] "0"   "1"   "2"   "3"   "4"   "NA"
```

```
w <- which(dat.family[, num_children] == "NA")
print(w)
```

```
[1] 893
```

The value of num_children at index 893 is "NA", a character. This should have been coded as **NA**, a missing value. One single mistaken value converted the whole vector from a numeric to a character.

Correcting the Mistake

```
dat.family[w, num_children := NA]
dat.family[, num_children := as.numeric(num_children)]
dat.family[, is.numeric(num_children)]
```

```
[1] TRUE
```

```
dat.family[, summary(num_children)]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
  0.000   1.000   2.000   1.979   3.000   4.000       1 
```

Off By One Errors

Show 10 entries

Search:

	Name	City	State
1	Karen Fenton	Philadelphia	PA
2	Mohamed Saidi	New York	NY
3	Steve	and Thomas Chang	Miami

Showing 1 to 3 of 3 entries

Previous 1 Next

This mistake impacted one row of data. However, other off-by-one errors can impact every row or every column!

Using Your Knowledge

Some values of a variable *should be* impossible for a specific domain:

- Negative Ages for people – caused by a typo in their year of birth (e.g. 2103).
- Patients who are prescribed a drug that they are allergic to.
- Patients with two prescriptions that interact badly to create complications in combination.
- Surgical operations on *the healthy limb*.
- Always be on the lookout for results or combinations of values that would seem to be impossible. You might discover coding errors – or identify far more serious problems in need of remediation. There are numerous opportunities to extend lives – or otherwise improve the process – by closely examining healthcare data.

Real Situations, No Clear Solutions

- Patients with multiple medical visits on known dates.
- We asked them: “When was the month and year of your last visit to the ER?”
- Then we tabulated the counts for those who responded “Unknown”:

```
              Year: Known Year: Unknown
Month: Known         6218            21
Month Unknown        1308           403
```

What would you do to clean up this mess?

One Possible Resolution

- Month Unknown, Year Known: Estimate the month by splitting the difference between the date recorded and either a) the last visit or b) January 1st.
- Month Known, Year Unknown: This seemed like a less sure sign of a recent ER visit. We didn't count these as recent visits.
- Both Unknown: Don't count these as recent ER visits.
- Since the goal was to track ER visits after the patient's baseline treatment, we made a judgment call to only count visits with a known year.

Deep Dive #2 – Tremendously Messy Data

- We will be examining the file **Simulated Blood Test and Newborn Weight.xlsx**.
- These data – which are simulated – reflect some work on a real study conducted long ago.
- **Scenario**: A new blood test has been proposed that may be a marker for predicting low weights at birth for babies. The researchers thought that elevated levels of the blood test might be a predictor of low birthweights.

However, before you can do the analysis, it will be necessary to understand, organize, and clean up the data. Let's dig in!

Challenges of Spreadsheets

- Spreadsheet files can have more than one tab of data.
- You don't necessarily know what the structure will be… which tabs will be useful, how large they are relative to others, etc.
- For small files, you can open up the file in a spreadsheet program. Larger files may not have that luxury.

The Structure of XLSX Files

- First, we can try to understand the different tabs:

```
library(readxl)
the.path <- "Simulated Blood Test and Newborn Weight.xlsx"
sheet.names <- excel_sheets(path = the.path)
print(sheet.names)
```

```
[1] "Elevated" "Control"
```

- It appears that there are different data sets stored as tabs in the sheet: one for the elevated cohort, and the other for the control group.

Reading XLSX Data

```r
elevated <- read_excel(path = the.path, sheet = "Elevated")
control <- read_excel(path = the.path, sheet = "Control")
```

- The **read_excel** command formats the data as **tibble** data.frames. We can convert them to **data.table** objects as follows:

```r
library(data.table)
elevated <- setDT(x = elevated)
control <- setDT(x = control)
```

- Using **setDT** is the most efficient way to perform this conversion when the data is already a list, data.frame, or data.table object.

- Other data types (e.g. a matrix) can be converted using **as.data.table**. This is not as efficient but works on a wider array of objects.

A First Glance at the Elevated Data

The **datatable** function in the **DT** library is an excellent way to display tables in HTML. It should not be confused with the **data.table** function and library for data processing.

```r
library(DT)
datatable(data = elevated)
```

Show 10 entries   Search:

	X__1	Baby	X__2	X__3	X__4	X__5	X__6	X__7	X__8	X__9	X__10	Mo
1	Subject ID#	Blood Test	Premature	Gestational Age	Date of Birth	Sex of Newborn	Blood Type	Height	Weight	Temperature - C	Blood Pressure	Age c Moth
2	1	22.6	yes		8-7-2013	M	O	40 cm	2468 g	98.7 F	89/50	19
3	2	22.8	no	38 wks	3-25-2013	M	O	48 cm	7.2 lbs	36.7 C	73/48	23
4	3	26.4	yes	29.2 wks	4-9-2013	F	n/a	41 cm	3.2 lbs	99.1 F	83/48	26
5	4	26.9	yes	36.9 wks	12-24-2014	M	O	39cm	6 lbs 11oz	99.3 F	95/43	33
6	5	26.7	yes	35 wks	8-22-2014	F	n/a	36 cm	2561 g	37.1 C	71/44	29
7	6	20.6	yes	33 wks	5-29-2013	M	B	41.7cm	2.5 kg	99.0 F	72/45	37
8	7	22.8	yes		2-16-2014	F	B	40 cm	2.9 lbs	37.2 C	76/45	21
9	8	TNP	no		10-5-2013			47.5 cm	8 lbs 7oz	36.7 C	74/41	
10	9	TNP	yes		9-26-2013			41 cm	4.4 lbs	98.3 F	75/48	

Showing 1 to 10 of 101 entries   Previous 1 2 3 4 5 … 11 Next

The Control Data

You can also view the data by clicking on the **Environment** portion of RStudio or by printing portions in the console.

```
datatable(data = control)
```

Show 10 entries

Search:

	X__1	Baby	X__2	X__3	X__4	X__5	X__6	X__7	X__8	X__9	X__10	Mo
1	Subject ID#	Blood Test	Premature	Gestational Age	Date of Birth	Sex of Newborn	Blood Type	Height	Weight	Temperature	Blood Pressure	Age c Moth
2	1	17.2	no		12-17-2013	male	n/a	45.6cm	7.9 lbs	37.2 C	80/49	32
3	2	13.5	Blood test performed at least 1 month after birth	28.8 wks	5-11-2014	Male	AB	NAcm	3.2 lbs	99.4 F	79/51	26
4	3	10.1	yes		4-30-2013	Male	O	39 cm	4 lbs 5oz	37.4 C	77/41	41
5	4	17.7	yes	34.3 wks	9-8-2013	Female	n/a	39.8cm	4 lbs 13oz	98.6 F	77/42	41
6	5	11.2	yes		7-14-2013	male	n/a	39 cm	5.1 lbs	98.8 F	59/50	37
7	6	17.2	Same patient as above									
8	7	12.7	No newborn data						4.3 lbs	99.8 F	80/47	
9	8	16.5	No newborn data						6.6 lbs	37.2 C	73/45	
10	9	10.5	yes	33.2 wks	4-7-2013	Female	B	39.5 cm	4.4 lbs	99.4 F	89/49	20

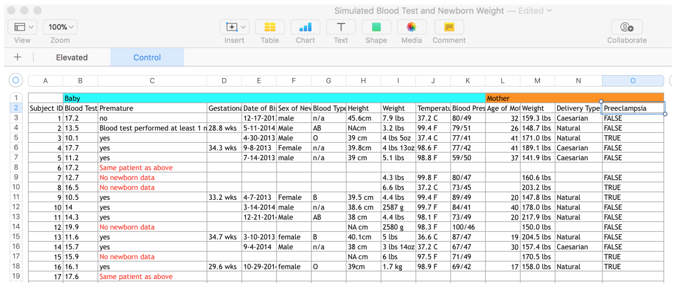
Showing 1 to 10 of 101 entries

Previous 1 2 3 4 5 … 11 Next

Strange Variable Names

- The header names are mostly default values (X__1, X__2, etc.) assigned for empty values.
- Only **Baby** and **Mother** appear to have been written in for the header.
- Meanwhile, the first row has character values that look much closer to typical column names. The remainder of the rows look like medical data.

Looking at the Spreadsheet



Simulated Blood Test and Newborn Weight — Edited

Elevated | Control

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1		Baby										Mother			
2	Subject ID	Blood Test	Premature	Gestationa	Date of Bi	Sex of Nev	Blood Type	Height	Weight	Temperatu	Blood Pres	Age of Mot	Weight	Delivery Type	Preeclampsia
3	1	17.2	no		12-17-201	male	n/a	45.6cm	7.9 lbs	37.2 C	80/49	32	159.3 lbs	Caesarian	FALSE
4	2	13.5	Blood test performed at least 1 n	28.8 wks	5-11-2014	Male	AB	NAcm	3.2 lbs	99.4 F	79/51	26	148.7 lbs	Natural	FALSE
5	3	10.1	yes		4-30-2013	Male	O	39 cm	4 lbs 5oz	37.4 C	77/41	41	171.0 lbs	Natural	TRUE
6	4	17.7	yes	34.3 wks	9-8-2013	Female	n/a	39.8cm	4 lbs 13oz	98.6 F	77/42	41	189.1 lbs	Caesarian	FALSE
7	5	11.2	yes		7-14-2013	male	n/a	39 cm	5.1 lbs	98.8 F	59/50	37	141.9 lbs	Caesarian	FALSE
8	6	17.2	Same patient as above												
9	7	12.7	No newborn data						4.3 lbs	99.8 F	80/47		160.6 lbs		FALSE
10	8	16.5	No newborn data						6.6 lbs	37.2 C	73/45		203.2 lbs		TRUE
11	9	10.5	yes	33.2 wks	4-7-2013	Female	B	39.5 cm	4.4 lbs	99.4 F	89/49	20	147.8 lbs	Natural	TRUE
12	10	14	yes		3-14-2014	male	n/a	38.6 cm	2587 g	99.7 F	84/41	40	178.0 lbs	Natural	FALSE
13	11	14.3	yes		12-21-201	Male	AB	38 cm	4.4 lbs	98.1 F	73/49	20	217.9 lbs	Natural	FALSE
14	12	19.9	No newborn data					NA cm	2580 g	98.3 F	100/46		150.0 lbs		FALSE
15	13	11.6	yes	34.7 wks	3-10-2013	female	B	40.1cm	5 lbs	36.6 C	87/47	19	204.5 lbs	Natural	FALSE
16	14	15.7	yes		9-4-2014	Male	n/a	38 cm	3 lbs 14oz	37.2 C	67/47	30	157.4 lbs	Caesarian	FALSE
17	15	15.9	No newborn data					NA cm	6 lbs	97.5 F	71/49		170.5 lbs		TRUE
18	16	16.1	yes	29.6 wks	10-29-201	female	O	39cm	1.7 kg	98.9 F	69/42	17	158.0 lbs	Natural	TRUE
19	17	17.6	Same patient as above												

- It's much easier to visually inspect a file.
- You can get a better sense of the structure.
- However, it's not always easy – or even possible – to look through a file directly. Relying on it will only take you as far as spreadsheet programs can go. Programming with R can take you further, but you'll have to learn to inspect data without the benefit of opening the spreadsheet.

Numerous Problems

- If you look carefully, you will begin to see that the construction of this data set was not especially systematic.
- I have identified **27** different issues, and there may be more.
- It is a useful record for people to read through, but much of the data is not set up to analyze.
- Carefully cleaning the data will be a necessary step before any analysis is possible.

Why Not Clean Up the Spreadsheet Directly?

- It may not be possible to open large files as a spreadsheet. R's memory limits greatly outperform those of spreadsheet programs.
- Spreadsheets are **fundamentally unaccountable**. It would be nearly impossible to keep a record of how you cleaned a data set within it.
- R can also enable many computations that are not easily performed in a spreadsheet. Yes, you could try to set up macros or spreadsheet programs, but R is better equipped to do this productively.

Because of the limitations of spreadsheet programs, it is worthwhile to use R to systematically investigate, understand, and organize data.

Issue #1: Headers in Multiple Rows

- We saw through investigation in both R and the spreadsheet that the real column names are in the second row.
- However, the **Baby** and **Mother** headings in Row 1 provide useful information. They mark the beginning of **sections of columns** pertaining to the baby and the mother.
- We will use the **skip** parameter to re-read the data:

```
elevated <- setDT(read_excel(path = the.path, sheet = "Elevated",
    skip = 1))
control <- setDT(read_excel(path = the.path, sheet = "Control",
    skip = 1))
```

Examining the Updated Data

```
datatable(data = elevated[1:5, ])
```

Show 10 entries Search:

	Subject ID#	Blood Test	Premature	Gestational Age	Date of Birth	Sex of Newborn	Blood Type	Height	Weight	Tem
1	1	22.6	yes		8-7-2013	M	O	40 cm	2468 g	98.7 F
2	2	22.8	no	38 wks	3-25-2013	M	O	48 cm	7.2 lbs	36.7 C
3	3	26.4	yes	29.2 wks	4-9-2013	F	n/a	41 cm	3.2 lbs	99.1 F
4	4	26.9	yes	36.9 wks	12-24-2014	M	O	39cm	6 lbs 11oz	99.3 F
5	5	26.7	yes	35 wks	8-22-2014	F	n/a	36 cm	2561 g	37.1 C

Showing 1 to 5 of 5 entries Previous 1 Next

Issue #2: Data Spread Across Multiple Tables

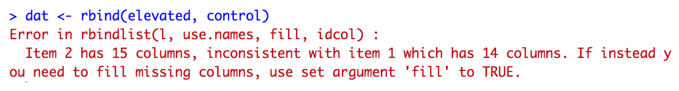
- The **elevated** and **control** tables each contain pieces of the overall data.
- Any clean-up operation or analysis would have to be repeated across each table.
- Unifying the data into a single data.table object will best enable clean-up and analysis.

With that said, **unification may or may not be the best approach**. Whether or not to do so depends on your goals, the similarities and differences of the tables, and the trade-offs between unified and segmented analyses. Evaluating these circumstances should play a role in your choices about the designs.

Combining Sources of Information – Classical Methods

- **rbind** will bind the rows of vectors, matrices, or data.frames.
- **cbind** will bind the columns of vectors, matrices, or data.frames.
- **data.frame(x, y, …)** or **data.table(x, y, …)** will create a new data frame by combining the columns of each object supplied.

Since we want to combine the rows of **elevated** and **control**, then binding the rows would make sense. However…



```
> dat <- rbind(elevated, control)
Error in rbindlist(l, use.names, fill, idcol) :
  Item 2 has 15 columns, inconsistent with item 1 which has 14 columns. If instead y
ou need to fill missing columns, use set argument 'fill' to TRUE.
```

Learning From Mistakes

- In this case, the error message is informative.
- The two data.table objects have differing numbers of columns.
- Using **fill = TRUE** will fill in the non-overlapping columns with missing **NA** values to allow for binding.

```
dat <- rbind(elevated, control, fill = TRUE)
print(names(dat))
```

```
 [1] "Subject ID#"      "Blood Test"       "Premature"
 [4] "Gestational Age" "Date of Birth"    "Sex of Newborn"
 [7] "Blood Type"       "Height"           "Weight"
[10] "Temperature - C" "Blood Pressure"   "Age of Mother"
[13] "Weight__1"        "Delivery Type"    "Temperature"
[16] "Preeclampsia"
```

A Faster Way to Bind Rows

- The function **rbindlist** in the **data.table** package accomplishes the same tasks as **rbind**.
- **rbindlist** handles memory more efficiently and can be faster for large sample sizes – albeit with similar performance in a small sample size like we have here.
- Relative to rbind, using **rbindlist** can also be more flexible to alternative structures like a list of data.frame or data.table objects.

```
dat <- rbindlist(l = list(elevated, control), fill = TRUE)
```

Issue #3: Lost Connection to Cohorts

- The original cohorts were separated by tabs in the spreadsheet. However, there is no surefire identifier for the cohorts in the combined data.

- This can be solved by creating a column for the cohort prior to binding:

```
elevated[, cohort := "Elevated"]
control[, cohort := "Control"]
```

```
dat <- rbindlist(l = list(elevated, control), fill = TRUE)
dat[, .N, by = "cohort"]
```

```
     cohort   N
1: Elevated 100
2:  Control 100
```

Issue #4: Columns `Temperature` and `Temperature - C`

- The variable for temperature had different names in each cohort:

```r
names.with.pattern <- function(x, pattern) {
    return(x[grep(pattern = pattern, x = x)])
}
pattern.temp <- "Temp"
names.with.pattern(x = names(elevated), pattern = pattern.temp)
```

```
[1] "Temperature - C"
```

```r
names.with.pattern(x = names(control), pattern = pattern.temp)
```

```
[1] "Temperature"
```

- We can further confirm this by checking for rates of 100% missingness in the full data within different cohorts:

```r
temp.names <- names.with.pattern(x = names(dat), pattern = pattern.temp)
mean.missing <- function(x) {
    return(mean(is.na(x)))
}
dat[, lapply(X = .SD, FUN = "mean.missing"), .SDcols = temp.names,
    by = cohort]
```

```
     cohort Temperature - C Temperature
1: Elevated               0        1.00
2:  Control               1        0.08
```

Solution: Unify the Names Prior to Binding

- Calling **rbindlist**, we will bind the rows based on matching column names of the data.
- With **fill = TRUE**, then mismatched column names will create separate columns.
- Changing the names first will resolve this problem.

```
setnames(x = elevated, old = "Temperature - C", new = "Temperature")
dat <- rbindlist(l = list(elevated, control), fill = TRUE)
dat[, .(Missingness_Temp = mean.missing(Temperature)), by = cohort]
```

```
     cohort Missingness_Temp
1: Elevated             0.00
2:  Control             0.08
```

Design Choices – When to Bind

- We initially said that cleaning would be easier by binding the rows prior to any other cleaning steps.
- However, this turned out to be a hasty choice. We discovered tasks – like naming the cohorts and matching the column names – that are better performed prior to binding.
- While there are no absolute prescriptions, observing these kinds of trends across multiple projects can inform how you approach your work.

Issue #4: Unique Identifiers

- It is always a good practice to provide each subject of a data set with a unique identifier. Was this done in the study? Let's investigate:

```
id.name <- "Subject ID#"

# Overall
dat[, .(`Number of Rows` = .N, `Unique Identifiers` = length(unique(get(id.name))))]
```

```
   Number of Rows Unique Identifiers
1:            200                100
```

```
# By Cohort
dat[, .(`Number of Rows` = .N, `Unique Identifiers` = length(unique(get(id.name)))),
    by = "cohort"]
```

```
     cohort Number of Rows Unique Identifiers
1: Elevated            100                100
2:  Control            100                100
```

```
dat[1:5, get(id.name)]
```

```
[1] 1 2 3 4 5
```

- The Identifiers are unique in each cohort but overlap across the cohorts.

Creating Unique Identifiers

A wide range of solutions might suffice here. A few possibilities include:

- Create a second identifier with a set of random strings:

```
library(stringi)
dat[, `Random ID` := stri_rand_strings(n = .N, length = 16)]
```

- Combine Cohort and Subject ID:

```
dat[, `Cohort ID` := sprintf("%s: %d", cohort, get(id.name))]
print(dat[1:5, .(`Subject ID#`, `Random ID`, `Cohort ID`)])
```

```
   Subject ID#        Random ID   Cohort ID
1:           1 A42DbJ5OfYoQZbbh Elevated: 1
2:           2 OOxqgfjpJAUoOiNR Elevated: 2
3:           3 ngOhAZVWGA6Tj5L9 Elevated: 3
4:           4 L9au6vYbR8CV9Siv Elevated: 4
5:           5 BoPlwvbNRJg9BDs9 Elevated: 5
```

- Or create unique identifiers earlier in the process.

Identifiers: Practices for Data Collection

- The practice used by the clinical team was good for counting the number of subjects in each cohort
- However, it created practical problems for uniquely identifying and tracking the subjects when the data were combined.
- Creating unique identifiers (that ideally maintain the privacy of the subject) should be an important first step in creating records.

This becomes increasingly important when the information about a subject cannot be tracked in a single row of a spreadsheet. Databases need to be able to record and extract all of the information for a single subject without any confusion in the process.

Issue #5: Blood Test is Not Numeric

- After a quick look at a few rows of data, you might want to summarize the Blood Test variable:

```r
blood.test.name <- "Blood Test"
dat[, summary(get(blood.test.name))]
```

```
   Length     Class      Mode 
      200 character character 
```

```r
dat[1:10, get(blood.test.name)]
```

```
 [1] "22.6" "22.8" "26.4" "26.9" "26.7" "20.6" "22.8" "TNP"  "TNP"  "16.9"
```

- Blood Test is listed as a character variable. Meanwhile, we see some values labeled as "TNP" in place of a numeric result.

- As a reminder, adding a single character value will coerce a numeric vector into a character vector.

So what do we do? And what is TNP?

The Mystery of TNP

- In the real spreadsheet I worked with, the bottom of the file included a note: **TNP – Test Not Performed**.
- In other settings, you would likely have to ask a member of your team.
- It is important to understand mysterious values because what they represent might inform how you handle them in cleaning the data.

Cataloging the Culprits of Character Coercion

- Numeric variables may have many unique values. It might be difficult to visually identify which records caused the coercion to character values.
- However, there is a coding solution:

```r
identify.character.coercion.culprits <- function(x) {
    w1 <- which(is.na(x))

    options(warn = -1)
    y <- as.numeric(x)
    options(warn = 0)
    w2 <- which(is.na(y))

    the.indices <- w2[!(w2 %in% w1)]
    the.culprits <- unique(x[the.indices])
    return(the.culprits)
}
the.culprits <- identify.character.coercion.culprits(x = dat[,
    get(blood.test.name)])
```

Handling the Culprits

- A Test Not Performed clearly indicates what should be a missing value.
- After setting the TNP values to missing, we can then switch the Blood Test to a numeric variable:

```
dat[get(blood.test.name) %in% the.culprits, eval(blood.test.name) := NA]
dat[, eval(blood.test.name) := as.numeric(get(blood.test.name))]

dat[, summary(get(blood.test.name))]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
   8.20   14.30   17.80   18.56   22.90   30.10      16 
```

Issue #6: Unusual Values in the Premature Variable

- Babies are considered premature if they are born before 37 weeks of gestational age.

- The Premature column should be an indicator of a premature birth. Let’s investigate:

```
premature.name <- "Premature"
dat[1:5, get(premature.name)]
```

```
[1] "yes" "no"  "yes" "yes" "yes"
```

```
dat[, unique(get(premature.name))]
```

```
[1] "yes"
[2] "no"
[3] "Blood test performed at least 1 month after birth"
[4] "Same patient as above"
[5] "No newborn data"
```

- It would appear that this variable is not as simple as yes or no. Let’s dig in further:

Issue #7: Blood Tests A Month After Birth?

- What does that mean? After conferring with the team, we had no definitive answer, but there was a decent theory: Not every pregnant mother has the blood test before giving birth. The team responsible for pulling data from the health records pulled the first test available, which happened to be long after the baby was born.

- A blood test after birth may have **no predictive value** for predicting the weight of the baby at birth. (On the other hand, depending on the blood test, there may be some correlation.)

- How to handle this matter is a delicate question. We decided to remove these blood tests (setting them to NA):

```
late.blood.test.value <- "Blood test performed at least 1 month after birth"
dat[get(premature.name) == late.blood.test.value, eval(blood.test.name) := NA]
```

Issue #8: Information in the Wrong Field

- Why was the timing of the blood test recorded in the field about Prematurity?
- Unfortunately, we did not have a chance to discuss this matter with the clinical team that put the data together.
- In practice, this can be a big problem for larger operations. Information needs to be recorded in the right place to be useful. Perhaps the timing of the blood test should have been a separate column of the data set. It makes no sense to overwrite the Prematurity value with this information here.

When you see these kinds of issues, you have an opportunity to change the processes. Discuss these issues with your team. Show them some examples. Contribute to better designs and systems. Over time, it pays off.

Adding a Field for Blood Test Timing

```
timing.name <- "On Time Blood Test"
dat[!is.na(get(blood.test.name)), eval(timing.name) := TRUE]
dat[get(premature.name) == late.blood.test.value, eval(timing.name) := FALSE]
```

Issue #9: Same Patient As Above?

```
same.patient.value <- "Same patient as above"
dat[get(premature.name) == same.patient.value, .N]
```

```
[1] 8
```

- This is another mystery that likely requires discussion with the team. Why did some patients have multiple IDs in the study?
- Ideas: Twins? Multiple Blood Tests? A mistake?
- We were unable to get a clear answer. Not knowing what else to do, we decided to remove these rows from the study.

```
orig.dat <- copy(dat)
dat <- dat[get(premature.name) != same.patient.value, ]
dat[, .N]
```

```
[1] 192
```

Copies of data.frame Objects

- With classical **data.frame** objects, any assignment of a new object based on an old one creates a copy of the values in memory:

```r
k <- 3
x <- data.frame(id = 1:k, value = rnorm(n = k))
y <- x
y$value[1] <- 17
print(x)
```

```
  id     value
1  1 0.5792581
2  2 1.4203508
3  3 1.7647737
```

```r
print(y)
```

```
  id     value
1  1 17.000000
2  2  1.420351
3  3  1.764774
```

- **x** and **y** are distinct objects here. Making a change in **y** did not affect the values of **x**.

Linked data.table Objects

- With **data.table** objects, an assignment of a new table based on an old one **points to the same set of memory**.

```r
k <- 3
x <- data.table(id = 1:k, value = rnorm(n = k))
y <- x
y[1, value := 7]
print(x)
```

```
   id      value
1:  1  7.0000000
2:  2 -0.2507511
3:  3 -0.6659089
```

```r
print(y)
```

```
   id      value
1:  1  7.0000000
2:  2 -0.2507511
3:  3 -0.6659089
```

- Because they are linked, changes in **y** will impact the values of **x**.
- This is a shortcoming of **data.table**. Its gains in speed and efficiency come at a price in situations like this one. Be careful!

Copies of data.table Objects

- The earlier problem can be avoided using the **copy** function:

```
x <- data.table(id = 1:k, value = rnorm(n = k))
y <- copy(x = x)
y[1, value := 7]
print(x)
```

```
   id       value
1:  1 -0.29082298
2:  2 -0.06158006
3:  3 -1.11303485
```

```
print(y)
```

```
   id       value
1:  1  7.00000000
2:  2 -0.06158006
3:  3 -1.11303485
```

Issue #10: No Newborn Data?

- Let's take a closer look at the records with "No newborn data" in the Premature column:

```
no.newborn.value <- "No newborn data"
dat[get(premature.name) == no.newborn.value, .N]
```

```
[1] 29
```

```
library(DT)
datatable(data = dat[get(premature.name) == no.newborn.value, ], rownames = FALSE)
```

Show 10 entries Search:

Subject ID#	Blood Test	Premature	Gestational Age	Date of Birth	Sex of Newborn	Blood Type	Height	Weight	Tempera
7	12.7	No newborn data						4.3 lbs	99.8 F
8	16.5	No newborn data						6.6 lbs	37.2 C
12	19.9	No newborn data					NA cm	2580 g	98.3 F
15	15.9	No newborn data					NA cm	6 lbs	97.5 F
20	8.2	No newborn data					NAcm	3.7 lbs	99.1 F
22	17.9	No newborn data					NAcm	2435 g	97.9 F
30	10.2	No newborn data					NA cm	2461 g	98.2 F
33	17.3	No newborn data					NAcm	1869 g	98.2 F
34	13.5	No newborn data					NAcm	2.2 kg	98.4 F
36	13.8	No newborn data					NAcm	2.5 lbs	37.3 C

Showing 1 to 10 of 29 entries Previous 1 2 3 Next

Some Newborn Data

- By direct examination, we see that “No newborn data” corresponds to missing values in some fields: **Premature, Gestational Age, Date of Birth, Sex of Newborn, and Blood Type**. The **Height** measurements all look strange, too (an issue to deal with later).

- However, other values are measured, like the **Weight, Temperature, and Blood Pressure**.

- For now, we will choose not to exclude these records since they measure the main factors about the blood test and weight of the newborn baby. However, greater investigation and discussions with other members of the team are warranted in this situation.

Issue #11: High Rates of Missingness in Gestational Age

- Let's calculate the proportion of missing cases for Gestational Age.

```
gestational.age.name <- "Gestational Age"
dat[, mean(is.na(get(gestational.age.name)))]
```

```
[1] 0.5625
```

- With such a high rate of missingness, it may be difficult to perform certain analyses, such as a linear regression of birth weights that incorporates Gestational Age and other factors. Any row with at least one missing covariate would be dropped.

- Bringing missingness to the attention of the team may help to identify other sources of this information. However, you may have to deal with the fact that the data includes high rates of missingness. You might decide to change the analysis – dropping the variable, or considering a subgroup analysis for the measured cases – in light of this observation.

Issue #12: Non-Numeric Data

- The measured values all have a unit “wks” for weeks included.
- Some measurements have a space between the number and the unit, while others don’t.
- Here is an approach to cleaning the data using **gsub** to replace the units and **trimws** to remove lagging spaces before numeric conversion:

```r
old.gestational.age.name <- gestational.age.name
gestational.age.name <- "Gest_Weeks"
pattern.weeks <- "wks"

dat[, eval(gestational.age.name) := gsub(pattern = pattern.weeks, replacement = "", x = get(old.gestational.age.name))]

dat[, eval(gestational.age.name) := as.numeric(trimws(x = get(gestational.age.name), which = "both"))]

dat[, summary(get(gestational.age.name))]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
  26.00   31.00   34.00   33.53   35.92   42.00     108 
```

New Columns or Old?

- In converting the values of Gestational Age into a usable numeric form, we have a choice: should we make the changes in their existing column or create a new one? Your choice would depend on some factors:
- Old column: No duplication, maintains column names and structure.
- New column: Easier to cross-check your work for accuracy, maintains both the old and new forms for later use, prevents coding errors that assume the wrong structure at that stage.

There is no universal answer here, but it's worth thinking through your choices about the designs.

Issue #13: Misclassified Premature Births?

- Are there newborns marked premature with gestational ages of 37 or more weeks?

```
threshold.premature <- 37
premature.yes.value <- "yes"

misclass.1 <- dat[get(premature.name) == premature.yes.value &
    get(gestational.age.name) >= threshold.premature, .(ID = get(id.name),
    Premature = get(premature.name), `Gestational Weeks` = get(gestational.age.name))]
misclass.1
```

```
   ID Premature Gestational Weeks
1: 70       yes              39.9
```

- Are there newborns marked NOT premature with gestational ages of less than 37 weeks?

```
premature.no.value <- "no"

misclass.2 <- dat[get(premature.name) == premature.no.value &
    get(gestational.age.name) < threshold.premature, .(ID = get(id.name),
    Premature = get(premature.name), `Gestational Weeks` = get(gestational.age.name))]
misclass.2
```

```
Empty data.table (0 rows) of 3 cols: ID,Premature,Gestational Weeks
```

- The mistaken cases should be investigated to determine which indicator is correct. Although the gestational weeks sounds more precise, it could be a typo. Likewise, Prematurity may have been miscalculated. Without further examination, we can’t know.

Issue #14: Formatting Birthdates

- Let's examine some of the dates of birth for the babies:

```
birthday.name <- "Date of Birth"
dat[, str(get(birthday.name))]
```

```
 chr [1:192] "8-7-2013" "3-25-2013" "4-9-2013" "12-24-2014" ...
```

```
NULL
```

```
dat[1:5, get(birthday.name)]
```

```
[1] "8-7-2013"   "3-25-2013"  "4-9-2013"   "12-24-2014" "8-22-2014"
```

- The birthdates are stored as characters with the month, day, and then year.

- This can be difficult to work with. For instance, if you sort the dates, you'll get the following output:

```
dat[1:5, sort(get(birthday.name))]
```

```
[1] "12-24-2014" "3-25-2013"  "4-9-2013"   "8-22-2014"  "8-7-2013"
```

This does not match up with any kind of consistent ordering of the actual dates.

Dates are Notoriously Difficult to Standardize

If we consider a date, such as the 30th of August of 1908, it can be numerically coded in numerous ways:

- 1908-8-30
- 1908-08-30
- 8-30-1908
- 08-30-1908
- 8-30-08
- 08-30-08
- 30-8-1908
- 30-08-1908
- 30-8-08
- 30-08-08

The current practice in American is to refer to dates by the month, day, and then year, **mm-dd-yyyy**, usually without leading zeros. This diverges from the standards of other countries.

My Approach to Dates

- Being able to order dates and compute the time from one to another are fundamental to many analyses.

- The **YYYY-mm-dd** format is the only way to maintain a proper order. This requires including all leading zeros. August 30th of 1908 would be **1908-08-30**.

- Meanwhile, I tend to structure dates as character vectors rather than any of the specialized forms (like POSIXct). This is possibly an unconventional choice, but it's not a bad default. Any further calculation that requires conversion can be done in the moment while maintaining a form that is sufficiently flexible to work with many forms and packages. The **lubridate** package can be useful, but no single set of tools is comprehensive for the many challenges dates convey.

- In this matter (and in all aspects of coding and data science), I would encourage you to examine other paradigms and develop an approach that works best for you.

Reformating the Birthdates

```r
## Converts dates to YYYY-mm-dd format stored in a
## character variable.  The format parameter is the
## current format for the date x.
convert.date <- function(x, format = "%m/%d/%Y") {
    require(lubridate)

    y <- as.character(ymd(as.Date(x = x, format = format)))
    return(y)
}
old.birthdate.name <- "Date of Birth"
birthdate.name <- "Birthdate"
```

```r
dat[, eval(birthdate.name) := convert.date(x = get(old.birthdate.name), format = "%m-%d-%Y")]
datatable(data = dat[, .(`Date of Birth`, Birthdate)], rownames = FALSE)
```

Show 10 entries Search:

Date of Birth	Birthdate
8-7-2013	2013-08-07
3-25-2013	2013-03-25
4-9-2013	2013-04-09
12-24-2014	2014-12-24
8-22-2014	2014-08-22
5-29-2013	2013-05-29
2-16-2014	2014-02-16
10-5-2013	2013-10-05
9-26-2013	2013-09-26
11-21-2013	2013-11-21

Showing 1 to 10 of 192 entries Previous 1 2 3 4 5 … 20 Next

Issue #15: Variations of Categorical Values

- Let's take a look at the **Sex of Newborn** variable:

```
sex.of.newborn.name <- "Sex of Newborn"
dat[, .N, keyby = sex.of.newborn.name]
```

```
   Sex of Newborn  N
1:           <NA> 44
2:              F 37
3:         FEMALE  5
4:         Female 20
5:              M 51
6:           MALE  3
7:           Male 18
8:         female  7
9:           male  7
```

- Each category was entered with slight variations in the values. This is a common problem when data **are entered manually** without limits on the input. Every typo works its way back to you.

Making Updates

- Here is one approach to handling the issue:

```r
old.female.values <- c("F", "FEMALE", "Female", "female")
old.male.values <- c("M", "MALE", "Male", "male")
female.value <- "Female"
male.value <- "Male"
```

```r
dat[get(sex.of.newborn.name) %in% old.female.values, eval(sex.of.newborn.name) := female.value]
dat[get(sex.of.newborn.name) %in% old.male.values, eval(sex.of.newborn.name) := male.value]
dat[, .N, keyby = sex.of.newborn.name]
```

```
   Sex of Newborn  N
1:           <NA> 44
2:         Female 69
3:           Male 79
```

Issue #16: Representation of Missing Data in Blood Type

- Let's take a look at the distribution of Blood Types:

```
blood.type.name <- "Blood Type"
dat[, .N, keyby = blood.type.name]
```

```
   Blood Type  N
1:       <NA> 44
2:          A 20
3:         AB  6
4:          B 19
5:          O 43
6:        n/a 60
```

- The standard categories (A, B, AB, and O) are present and reasonably in line with our expectations about their distribution.
- However, missing data is represented both by (a recognized form of missingness) and by **n/a**, which is not recognized as missing.

Coding Missing Data

- We can solve the problem here:

```
na.values <- c("n/a", "NA", "N/A")
dat[get(blood.type.name) %in% na.values, eval(blood.type.name) := NA]
dat[is.na(get(blood.type.name)), .N]
```

```
[1] 104
```

- Another approach is to specify the forms of missing data **when the file is read**. The **read_excel** file has a parameter called **na** that allows one to specify a character vector (e.g. **na.values** above). Likewise, data.table's **fread** function has a parameter called **na.strings**, and most other methods for reading files have a similar approach.

- If you specify the forms of NA values up front, it will save you the work of recoding each column individually.

Issue #17: Non-Numeric Height

- Let's take a look at some of the heights:

```
height.name <- "Height"
dat[1:5, get(height.name)]
```

```
[1] "40 cm" "48 cm" "41 cm" "39cm"  "36 cm"
```

- Similar to our work on Gestational Age, we will need to remove the units. Let's write a function (that could potentially be applied to both variables).

```
remove.units <- function(x, units) {
    y <- gsub(pattern = units, replacement = "", x = x)
    y.trimmed <- trimws(x = y, which = "both")
    return(y.trimmed)
}
```

```
dat[, eval(height.name) := remove.units(x = get(height.name), units = "cm")]
dat[1:5, get(height.name)]
```

```
[1] "40" "48" "41" "39" "36"
```

Issue #18: Non-Numeric Values for Height Remain

- Using our earlier function, let's identify the non-numeric values that remain in the Height column:

```
dat[, identify.character.coercion.culprits(x = get(height.name))]
```

```
[1] "NA"   "n/a"
```

- It will be necessary to convert these records to missing values:

```
dat[get(height.name) %in% na.values, eval(height.name) := NA]
dat[is.na(get(height.name)), .N]
```

```
[1] 41
```

```
dat[, identify.character.coercion.culprits(x = get(height.name))]
```

```
character(0)
```

- Could we have resolved this issue at the reading step? Yes, but only if we included the values **"NA cm"** and **"NA cm"** along with **"n/a"**.

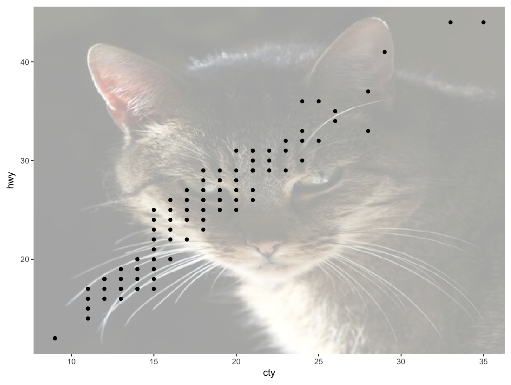
Why NA cm? Where Did That Come From?

- It is unlikely that a person wrote that in directly.
- However, perhaps the Height column was created by concatenating a numeric measurement with a unit.
- If so, then it’s worth examining the process that generated these data. A data engineer upstream could implement more safeguards for handling missing values – or at least perform a greater review of the information before passing it along. Fixing these problems upstream means that you won’t have to clean them up repeatedly.

And Now It's Time for a Short Break

- Check out the library **cats** with the **add_cat** function for plots.

```
library(cats)
library(ggplot2)
set.seed(18)
ggplot(mpg, aes(cty, hwy)) + add_cat(bw = FALSE) + geom_point()
```



Issue #19: Weight in Many Forms

- The weights of the newborns were recorded with multiple types of units:

```
weight.name <- "Weight"
dat[1:10, get(weight.name)]
```

```
 [1] "2468 g"      "7.2 lbs"     "3.2 lbs"     "6 lbs 11oz" "2561 g"
 [6] "2.5 kg"      "2.9 lbs"     "8 lbs 7oz"   "4.4 lbs"     "2563 g"
```

- Any kind of analysis will require standardization into a numeric form and consistent units.

- For standardization purposes, measuring weight in kilograms would make the most sense.

A Plan

- The weights in pounds have two forms: decimal pounds, and (x pounds, y ounces).
- We will first convert the ounces into a decimal fraction of a pound, and then address the pounds for both types.
- Then we will convert the grams into kilograms.

The result will be a new column called **Weight of Newborn (kg)** in standard units for all of the measures.

Reformatting lbs and oz

- First, some preliminaries

```r
pattern.oz <- "oz"
pattern.lbs <- "lbs"
pattern.kg <- "kg"
pattern.g <- "g"
```

- Then some reformatting functions:

```r
replace.and.convert.to.numeric <- function(x, pattern, which = "both") {
    y <- gsub(pattern = pattern, replacement = "", x = x)
    y.trimmed <- trimws(x = y, which = which)
    return(as.numeric(y.trimmed))
}
```

```r
reformat.lbs.oz <- function(x, pattern.lbs, pattern.oz, which = "both"){
  require(data.table)
  the.pieces <- strsplit(x = x, split = pattern.lbs)
  tab <- as.data.table(t(setDT(the.pieces)))
  setnames(x = tab, old = names(tab), new = c(pattern.lbs, pattern.oz))
  tab[, eval(pattern.lbs) := as.numeric(trimws(x = get(pattern.lbs), which = "both"))]
  tab[, eval(pattern.oz) := replace.and.convert.to.numeric(x = get(pattern.oz), pattern = pattern.oz)]

  tab[, result := get(pattern.lbs) + get(pattern.oz)/16]
  return(tab[, result])
}
```

A Test Case

```r
x <- c("3 lbs 2oz", "4 lbs 1 oz")
reformat.lbs.oz(x = x, pattern.lbs = pattern.lbs, pattern.oz = pattern.oz,
    which = "both")
```

```
[1] 3.1250 4.0625
```

So far, so good! Now let's put it to use.

Converting the lbs and oz cases to kg:

```r
new.weight.name <- "Weight of Newborn (kg)"
lbs.to.kg.divider <- 2.2
w.oz <- dat[, grep(pattern = pattern.oz, x = get(weight.name))]
```

```r
dat[w.oz, eval(new.weight.name) := reformat.lbs.oz(x = get(weight.name), pattern.lbs = pattern.lbs, pattern.oz = pattern.oz, which =
  "both") / lbs.to.kg.divider]
```

- Let's take a look at these cases:

```r
case.tab.lbs.oz <- dat[w.oz, .(`Original Weight` = get(weight.name),
    `Calculated Weight in lbs` = lbs.to.kg.divider * get(new.weight.name),
    `Calculated Weight in kg` = get(new.weight.name))]
datatable(data = case.tab.lbs.oz, rownames = FALSE)
```

Show 10 entries   Search:

Original Weight	Calculated Weight in lbs	Calculated Weight in kg
6 lbs 11oz	6.6875	3.03977272727273
8 lbs 7oz	8.4375	3.83522727272727
7 lbs 8oz	7.5	3.40909090909091
4 lbs 7oz	4.4375	2.01704545454545
4 lbs 14oz	4.875	2.21590909090909
5 lbs 3oz	5.1875	2.35795454545455
2 lbs 14oz	2.875	1.30681818181818
8 lbs 3oz	8.1875	3.72159090909091
5 lbs 10oz	5.625	2.55681818181818
8 lbs 4oz	8.25	3.75

Showing 1 to 10 of 56 entries   Previous 1 2 3 4 5 6 Next

Converting the Decimal lbs Cases

- We have to work on the lbs cases that **do not include** the unit of ounces.

```
w.lbs.all <- dat[, grep(pattern = pattern.lbs, x = get(weight.name))]
w.lbs.only <- w.lbs.all[!(w.lbs.all %in% w.oz)]
```

- Then we can drop the units and make a numeric conversion:

```
dat[w.lbs.only, eval(new.weight.name) := replace.and.convert.to.numeric(x = get(weight.name), pattern = pattern.lbs) /
  lbs.to.kg.divider]
```

- Then we can check the calculation:

```
case.tab.lbs <- dat[w.lbs.only, .(`Original Weight` = get(weight.name),
    `Calculated lbs` = lbs.to.kg.divider * get(new.weight.name),
    `Calculated kg` = get(new.weight.name))]
datatable(data = case.tab.lbs, rownames = FALSE)
```

Show 10 entries Search:

Original Weight	Calculated lbs	Calculated kg
7.2 lbs	7.2	3.27272727272727
3.2 lbs	3.2	1.45454545454545
2.9 lbs	2.9	1.31818181818182
4.4 lbs	4.4	2
1.9 lbs	1.9	0.863636363636364
4.6 lbs	4.6	2.09090909090909
6.1 lbs	6.1	2.77272727272727
9.2 lbs	9.2	4.18181818181818
9.6 lbs	9.6	4.36363636363636
9.2 lbs	9.2	4.18181818181818

Showing 1 to 10 of 87 entries Previous 1 2 3 4 5 … 9 Next

This also looks good!

The Grams and the Order of Operations

- The unit for grams **g** is a subset of the unit of kilograms **kg**.
- If we search for **g** with a function like **grep**, then all of the **kg** cases will also be flagged.
- It would make sense to process the **kg** cases first.

Convert kg Cases to Numeric Values

```r
w.kg <- dat[, grep(pattern = pattern.kg, x = get(weight.name))]
dat[w.kg, eval(new.weight.name) := replace.and.convert.to.numeric(x = get(weight.name), pattern = pattern.kg)]
```

- Then check the values:

```r
case.tab.kg <- dat[w.kg, .(`Original Weight` = get(weight.name),
    `Calculated Weight` = get(new.weight.name))]
datatable(data = case.tab.kg, rownames = FALSE)
```

Show 10 entries   Search:

Original Weight	Calculated Weight
2.5 kg	2.5
2.5 kg	2.5
2.5 kg	2.5
1.9 kg	1.9
1.2 kg	1.2
2.6 kg	2.6
2.0 kg	2
0.8 kg	0.8
2.1 kg	2.1
1.7 kg	1.7

Showing 1 to 10 of 20 entries   Previous 1 2 Next

That also looks good!

Now the Grams

```r
w.g.all <- dat[, grep(pattern = pattern.g, x = get(weight.name))]
w.g.only <- w.g.all[!(w.g.all %in% w.kg)]
g.to.kg.divider <- 1000
```

```r
dat[w.g.only, eval(new.weight.name) := replace.and.convert.to.numeric(x = get(weight.name), pattern = pattern.g) / g.to.kg.divider]
```

- Then we can check the cases:

```r
case.tab.g <- dat[w.g.only, .(`Original Weight` = get(weight.name),
    `Calculated Weight, g` = get(new.weight.name) * g.to.kg.divider,
    `Calculated Weight, kg` = get(new.weight.name))]
datatable(data = case.tab.g, rownames = FALSE)
```

Show 10 entries Search:

Original Weight	Calculated Weight, g	Calculated Weight, kg
2468 g	2468	2.468
2561 g	2561	2.561
2563 g	2563	2.563
1768 g	1768	1.768
3234 g	3234	3.234
2535 g	2535	2.535
1565 g	1565	1.565
1873 g	1873	1.873
2012 g	2012	2.012
1385 g	1385	1.385

Showing 1 to 10 of 29 entries Previous 1 2 3 Next

Lo and Behold, Standardized Weights

- We now have all of the weights of the newborns standardized in kilograms:

```
dat[, summary(get(new.weight.name))]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.4545  1.7663  2.2080  2.2551  2.5909  4.3636 
```

- The work here enumerated all of the cases observed, but each situation might reveal slight variations.

- In this circumstance, standardizing all of the weights was a ton of work! This would be a good opportunity for you to show those who collect the data what can be entailed. You might encourage them to give more thought to the manner in which the data are collected.

Issue #20: Unstandardized Temperatures

- Some of the temperatures are measured in Fahrenheit and others in Celsius:

```
old.temp.name <- "Temperature"
dat[1:5, get(old.temp.name)]
```

```
[1] "98.7 F" "36.7 C" "99.1 F" "99.3 F" "37.1 C"
```

- We will have to undertake a similar process of numeric conversion and standardization to Celsius:

```
temp.name <- "Temperature (C)"
temp.F.to.C <- function(F) {
    C <- (F - 32) * (5/9)
    return(C)
}
pattern.f <- "F"
pattern.c <- "C"

wf <- dat[, grep(pattern = pattern.f, x = get(old.temp.name))]
wc <- dat[, grep(pattern = pattern.c, x = get(old.temp.name))]
```

Temperature Conversions

```
dat[wf, eval(temp.name) := temp.F.to.C(replace.and.convert.to.numeric(x = get(old.temp.name), pattern = pattern.f))]

dat[wc, eval(temp.name) := replace.and.convert.to.numeric(x = get(old.temp.name), pattern = pattern.c)]
dat[, summary(get(temp.name))]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  35.90   36.78   37.06   37.03   37.28   37.83 
```

Issue #21: Blood Pressure Extraction

- The measurements for Blood Pressure are in the **x/y** form, with **x** as the **Systolic** and **y** as the **Diastolic** pressure.

```
bp.name <- "Blood Pressure"
dat[1:5, get(bp.name)]
```

```
[1] "89/50" "73/48" "83/48" "95/43" "71/44"
```

- To numerically analyze the results, we would need to extract the two readings and convert them to numeric form:

```
extract.bp.readings <- function(x, split, new.names) {
    require(data.table)
    the.pieces <- strsplit(x = x, split = split)
    tab <- as.data.table(t(setDT(the.pieces)))
    tab <- tab[, lapply(X = .SD, FUN = "as.numeric")]
    setnames(x = tab, old = names(tab), new = new.names)
    return(tab)
}
```

Performing the Extractions

```r
bp.reading.names <- c("Systolic BP", "Diastolic BP")
bp.readings <- extract.bp.readings(x = dat[, get(bp.name)],
    split = "/", new.names = bp.reading.names)

dat <- data.table(dat, bp.readings)
dat[, lapply(X = .SD, FUN = "summary"), .SDcols = bp.reading.names]
```

```
   Systolic BP Diastolic BP
1:       51.00        33.00
2:       73.00        42.00
3:       78.00        46.00
4:       77.69        45.25
5:       83.00        48.00
6:      100.00        55.00
```

Issue #22: Missingness of Mother's Age

- Let's take a look at the summary of the Mother's Age:

```
age.of.mother.name <- "Age of Mother"
dat[, mean(is.na(get(age.of.mother.name)))]
```

```
[1] 0.2291667
```

```
dat[, summary(get(age.of.mother.name))]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
  17.00   22.00   28.00   28.45   34.25   42.00      44 
```

- There is a reasonably high rate of missingness, which might be of concern.
- Otherwise, this variable looks reasonably well structured. All of the values are numeric and reasonably in line with the ages at which mothers give birth.

Issue #23: Weight__1's Name

- The weight of the mother was coded as **Weight__1**. In the original spreadsheet, it was listed as** Weight**. Since a previous column (the weight of the newborn) had the same name, the reading program automatically assigned** Weight__1** here to distinguish it.

- Let's change the name of the variable to something more meaningful:

```
old.mother.weight.name <- "Weight__1"
mother.weight.name <- "Weight of Mother (kg)"
dat[1:5, get(old.mother.weight.name)]
```

```
[1] "167.7 lbs" "184.6 lbs" "161.8 lbs" "184.0 lbs" "216.1 lbs"
```

```
# setnames(x = dat, old = old.mother.weight.name, new =
# mother.weight.name)
```

- However, before changing the name, we realize…

Issue #24: Mother's Weight is Non-Numeric and in Pounds

- We will need to convert the weights to numeric kilograms, in line with our work on the newborn weights.
- Fortunately, there are fewer cases here – every record is consistently labeled with the same units.
- To check the consistency, we'll choose to create a new column here – rather than changing the **Weight__1** column to a more meaningful name.

```
dat[, eval(mother.weight.name) := replace.and.convert.to.numeric(x = get(old.mother.weight.name), pattern = pattern.lbs) /
  lbs.to.kg.divider]
dat[, summary(get(mother.weight.name))]
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  46.82   68.12   74.84   74.50   80.42  108.59 
```

Checking the Case Conversions

```
case.check.mother.weight <- dat[, .(Original = get(old.mother.weight.name),
    `Calculated (lbs)` = lbs.to.kg.divider * get(mother.weight.name),
    `Calculated (kg)` = get(mother.weight.name))]
datatable(data = case.check.mother.weight, rownames = FALSE)
```

Show 10 entries Search:

Original	Calculated (lbs)	Calculated (kg)
167.7 lbs	167.7	76.2272727272727
184.6 lbs	184.6	83.9090909090909
161.8 lbs	161.8	73.5454545454545
184.0 lbs	184	83.6363636363636
216.1 lbs	216.1	98.2272727272727
158.9 lbs	158.9	72.2272727272727
173.9 lbs	173.9	79.0454545454545
193.2 lbs	193.2	87.8181818181818
181.0 lbs	181	82.2727272727273
199.2 lbs	199.2	90.5454545454545

Showing 1 to 10 of 192 entries Previous 1 2 3 4 5 … 20 Next

Issue #25: Missingness in Delivery Type

- Let’s check on the Delivery Type variable:

```
delivery.type.name <- "Delivery Type"
dat[, .N, keyby = delivery.type.name]
```

```
   Delivery Type  N
1:          <NA> 44
2:     Caesarian 50
3:       Natural 98
```

- There is also a high rate of missingness in these data.

- Otherwise, the categories are clear and understandable.

Issue #26: Missingness in Preeclampsia

- Let’s check on the Delivery Type variable:

```
preeclampsia.name <- "Preeclampsia"
dat[, .N, keyby = preeclampsia.name]
```

```
   Preeclampsia   N
1:           NA 100
2:        FALSE  67
3:         TRUE  25
```

- There is also a high rate of missingness in these data.

- Otherwise, the categories are clear and understandable.

Issue #27: What Does Missingness Mean?

- For Delivery Type, does a missing value mean we truly don't know, or should we assume a default of a natural delivery? After all, a Caesarian delivery is a major surgery, which would presumably have associated records.
- What about missingness in Preeclampsia? Does missingness signify a lack of knowledge, or should we assume a default of FALSE? After all, preeclampsia is a major complication with life-threatening implications.
- Ultimately these questions cannot be fully answered just by looking at the data. It requires a greater understanding of the measurements, the process used to collect data, and the most likely alternatives. These are all factors to consider before forging ahead with an analysis. When you have a chance, ask members of your team – or outside experts – for help. Ultimately, it will be important to justify your assumptions, and the results could hinge on the interpretations.

And On and On

- Cleaning this data set turned out to be a major undertaking in its own right.
- Even when a cleaning issue is easy to diagnose, it's not necessarily so easy to fix. Converting multiple formats for weight turned out to require a considerable degree of effort.
- Ultimately, all of this work was important. Without the cleaning, it would have been difficult to answer simple questions about the average scores, correlations, or regression effects.

Engineer or Scientist?

Data cleaning heavily emphasizes the engineering component of data science. You can’t get to the scientific investigation without first creating the structures that enable analyses.

Meanwhile, creating a data structure that enables your work utilizes different skills than analyzing the data. Sometimes you have to be an engineer before you can be a scientist. But sometimes you have to go back and forth to get it right.

Speaking Up

- As you can see, data cleaning can involve a lot of work. There is a reason why our **Lament of the Data Scientist** is such a common observation in our community.
- Meanwhile, a significant portion of your work in cleaning the **might not be readily apparent to or appreciated by other members of your team**.
- It is important to communicate about what you're seeing. Make sure that others have reasonable expectations for what your work will entail. Better yet, set reasonable expectations for other members of your team. There is an opportunity to improve upon the accuracy, quality, and timeliness of the information that your organization needs. Advocate for these changes, and your organization will run more smoothly in the long run.
- Better yet, being thorough in the beginning pays dividends down the road. Issues of data quality will eventually have to be addressed. If you don't do this up front, then your later work may require fundamental changes, up to and including starting over. If productivity, accuracy, and quality are your goals, then learning how to clean data well can help you get there.