

Module 9

Data Cleaning

Andrew Jaffe
Instructor

Data

- We will be using multiple data sets in this lecture:
 - Salary, Monument, Circulator, and Restaurant from OpenBaltimore:
<https://data.baltimorecity.gov/browse?limitTo=datasets>
 - Gap Minder - very interesting way of viewing longitudinal data
 - Data is here - <http://www.gapminder.org/data/>
 - http://spreadsheets.google.com/pub?key=rMsQHawTObBb6_U2ESjKXYw&output=xls

Data Cleaning

In general, data cleaning is a process of investigating your data for inaccuracies, or recoding it in a way that makes it more manageable.

MOST IMPORTANT RULE - LOOK AT YOUR DATA!

Again - `table`, `summarize`, `is.na`, `any`, `all` are useful.

Data Cleaning

```
> table(c(0, 1, 2, 3, NA, 3, 3, 2, 2, 3), useNA = "ifany")
```

```
 0  1  2  3 <NA>  
1  1  3  4   1
```

```
> table(c(0, 1, 2, 3, 2, 3, 3, 2, 2, 3), useNA = "always")
```

```
 0  1  2  3 <NA>  
1  1  4  4   0
```

```
> tab <- table(c(0, 1, 2, 3, 2, 3, 3, 2, 2, 3), c(0, 1, 2, 3, 2, 3, 3, 4, 4, 3),  
+           useNA = "always")  
> margin.table(tab, 2)
```

```
 0  1  2  3  4 <NA>  
1  1  2  4  2   0
```

```
> prop.table(tab)
```

```
      0  1  2  3  4 <NA>
0  0.1 0.0 0.0 0.0 0.0 0.0
1  0.0 0.1 0.0 0.0 0.0 0.0
2  0.0 0.0 0.2 0.0 0.2 0.0
3  0.0 0.0 0.0 0.4 0.0 0.0
<NA> 0.0 0.0 0.0 0.0 0.0 0.0
```

```
> prop.table(tab, 1)
```

```
      0  1  2  3  4 <NA>
0  1.0 0.0 0.0 0.0 0.0 0.0
1  0.0 1.0 0.0 0.0 0.0 0.0
2  0.0 0.0 0.5 0.0 0.5 0.0
3  0.0 0.0 0.0 1.0 0.0 0.0
<NA>
```

Download Salary FY2012 Data

<https://data.baltimorecity.gov/Financial/Baltimore-City-Employee-Salaries-FY2012/7ymi-bvp3>

Download as a CSV and then read it into R as the variable `Sal`

Download Salary FY2012 Data

<https://data.baltimorecity.gov/Financial/Baltimore-City-Employee-Salaries-FY2012/7ymi-bvp3>

Download as a CSV and then read it into R as the variable `Sal`

```
> Sal = read.csv("data/Baltimore_City_Employee_Salaries_FY2012.csv", as.is = TRUE)
> colnames(Sal)[1] = "Name" # make uppercase
```

Data Cleaning

- `any()` - checks if there are any TRUES
- `all()` - checks if ALL are true

```
> Sal[1:4, ]
```

	Name	JobTitle	AgencyID
1	Aaron,Patricia G	Facilities/Office Services II	A03031
2	Aaron,Petra L	ASSISTANT STATE'S ATTORNEY	A29005
3	Abaineh,Yohannes T	EPIDEMIOLOGIST	A65026
4	Abdal-Rahim,Naim A	EMT FIREFIGHTER	A64215

	Agency	HireDate	AnnualSalary	GrossPay
1	OED-Employment Dev	10/24/1979	\$51862.00	\$52247.39
2	States Attorneys Office	09/25/2006	\$64000.00	\$59026.81
3	HLTH-Health Department	07/23/2009	\$57900.00	\$57129.79
4	Fire Department	03/30/2011	\$34146.00	\$35537.88

```
> any(is.na(Sal$Name))
```

```
[1] FALSE
```


Example of Cleaning:

For example, let's say gender was coded as Male, M, m, Female, F, f. Using Excel to find all of these would be a matter of filtering and changing all by hand or using if statements.

In R, you can simply do something like:

```
data$gender[data$gender %in% c("Male", "M", "m")] <- "Male"
```

Sometimes though, it's not so simple. That's where functions that find patterns come in very useful.

```
> table(gender)
```

```
gender
  F FeMAle FEMALE   Fm   M   Ma  mAle  Male  MaLe  MALE
75   82   74   89   89   79   87   89   88   95
Man  Woman
73   80
```

Find/Replace and Regular Expressions

- R can do much more than find exact matches for a whole string
- Like Perl and other languages, it can use regular expressions.
- What are regular expressions?
- Ways to search for specific strings
- Can be very complicated or simple
- Highly Useful

'Find' functions

`grep`: `grep`, `grepl`, `regexpr` and `gregexpr` search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

`grep(pattern, x, fixed=FALSE)`, where:

- `pattern` = character string containing a regular expression to be matched in the given character vector.
- `x` = a character vector where matches are sought, or an object which can be coerced by `as.character` to a character vector.
- If `fixed=TRUE`, it will do exact matching for the phrase anywhere in the vector (regular find)

```
> grep("Rawlings", Sal$Name) # These are the indices/elements where the pattern match occurs
```

```
[1] 11755 11756 11757 11758
```

`grep()` returns something similar to `which()` on a logical statement

```
> grep("Rawlings", Sal$Name)
```

```
[1] 11755 11756 11757 11758
```

```
> grep("Rawlings", Sal$Name, value = TRUE)
```

```
[1] "Rawlings Jr,Anthony H" "Rawlings,Kellye A"      "Rawlings,Paula M"  
[4] "Rawlings,Stephanie C"
```

```
> Sal[grep("Rawlings", Sal$Name), ]
```

	Name	JobTitle	AgencyID
11755	Rawlings Jr,Anthony H	AIDE BLUE CHIP	W02712
11756	Rawlings,Kellye A	EMERGENCY DISPATCHER	A99372
11757	Rawlings,Paula M	COMMUNITY AIDE	A04015
11758	Rawlings,Stephanie C	MAYOR	A01001

	Agency	HireDate	AnnualSalary	GrossPay
11755	Youth Summer	06/07/2011	\$1 \$1	
11756	Police Department	01/06/2003	\$4 \$5	
11757	R&P-Recreation	12/10/2007	\$1 \$9	
11758	Mayors Office	12/07/1995	\$1 \$1	

Grep Options

```
> head(grep("Tajhgh", Sal$Name, value = TRUE))
```

```
[1] "Reynold,Tajhgh J"
```

```
> grep("Jaffe", Sal$Name)
```

```
integer(0)
```

```
> length(grep("Jaffe", Sal$Name))
```

```
[1] 0
```

A bit on Regular Expressions

- <http://www.regular-expressions.info/reference.html>
- They can use to match a large number of strings in one statement
- `.` matches any single character
- `*` means repeat as many (even if 0) more times the last character
- `?` makes the last thing optional

Using Regular Expressions

- Look for any name that starts with:
 - Payne at the beginning,
 - Leonard and then an S
 - Spence then a capital C

```
> grep("Payne.*", x = Sal$Name, value = TRUE)
```

```
[1] "Payne,Brittany N"      "Payne-Cooke,Shelley F"  
[3] "Payne,Denise I"       "Payne El,Jackie"  
[5] "Payne,Gary W"         "Payne,James D"  
[7] "Payne,James R"        "Payne,Jasman T"  
[9] "Payne Johnson,Nickole A" "Payne,Jordan A"  
[11] "Payne,Karen V"        "Payne,Leonard S"  
[13] "Payne,Mary A"         "Payne,Micah W"  
[15] "Payne,Michael N"      "Payne,Walter"  
[17] "Ray Payne,Marion J"
```

```
> grep("Leonard.?S", x = Sal$Name, value = TRUE)
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S"
```

```
> grep("Spence.*C.*", x = Sal$Name, value = TRUE)
```

```
[1] "Spence Jr,Leonard C" "Spencer,Charles A"  "Spencer,Clarence W"  
[4] "Spencer,Michael C"
```


Replace

Let's say we wanted to sort the data set by Annual Salary:

```
> class(Sal$AnnualSalary)
```

```
[1] "character"
```

```
> sort(c("1", "2", "10")) # not sort correctly (order simply ranks the data)
```

```
[1] "1" "10" "2"
```

```
> order(c("1", "2", "10"))
```

```
[1] 1 3 2
```

Replace

So we must change the annual pay into a numeric:

```
> head(as.numeric(Sal$AnnualSalary), 4)
```

```
Warning: NAs introduced by coercion
```

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

R didn't like the \$ so it thought turned them all to **NA**.

`sub()` and `gsub()` can do the replacing part.

Replacing and subbing

Now we can replace the `$` with nothing (used `fixed=TRUE` because `$` means something in regular expressions):

```
> Sal$AnnualSalary <- as.numeric(gsub(pattern = "$", replacement = "", Sal$AnnualSalary,
+   fixed = TRUE))
> Sal <- Sal[order(Sal$AnnualSalary, decreasing = TRUE), ] # use negative to sort descending
> Sal[1:5, c("Name", "AnnualSalary", "JobTitle")]
```

	Name	AnnualSalary	JobTitle
1012	Bernstein, Gregg L	238772	STATE'S ATTORNEY
844	Bealefeld III, Frederick H	197700	EXECUTIVE LEVEL III
1120	Black, Harry E	180000	EXECUTIVE LEVEL III
12510	Sanchez, Alexander M	175000	EXECUTIVE LEVEL III
677	Barbot, Oxiris	170000	EXECUTIVE LEVEL III

Useful String Functions

Useful String functions

- `toupper()`, `tolower()` - uppercase or lowercase your data:
- `str_trim()` (in the `stringr` package) - will trim whitespace
- `nchar` - get the number of characters in a string
- `substr(x, start, stop)` - substrings from position start to position stop
- `strsplit(x, split)` - splits strings up - returns list!
- `paste()` - paste strings together - look at `?paste`

Paste

Paste can be very useful for joining vectors together:

```
> paste("Visit", 1:5, sep = "_")
```

```
[1] "Visit_1" "Visit_2" "Visit_3" "Visit_4" "Visit_5"
```

```
> paste("Visit", 1:5, sep = "_", collapse = " ")
```

```
[1] "Visit_1 Visit_2 Visit_3 Visit_4 Visit_5"
```

```
> paste("To", "is going be the ", "we go to the store!", sep = "day ")
```

```
[1] "Today is going be the day we go to the store!"
```

```
> # and paste0 can be even simpler see ?paste0  
> paste0("Visit", 1:5)
```

```
[1] "Visit1" "Visit2" "Visit3" "Visit4" "Visit5"
```

```
> paste(1:5, letters[1:5], sep = "_")
```

```
[1] "1_a" "2_b" "3_c" "4_d" "5_e"
```

```
> paste(6:10, 11:15, 2000:2005, sep = "/")
```

```
[1] "6/11/2000" "7/12/2001" "8/13/2002" "9/14/2003" "10/15/2004"  
[6] "6/11/2005"
```

```
> paste(paste("x", 1:5, sep = ""), collapse = "+")
```

```
[1] "x1+x2+x3+x4+x5"
```

Strsplit

```
> x <- c("I really", "like writing", "R code")  
> y <- strsplit(x, split = " ")  
> y[[2]]
```

```
[1] "like"    "writing"
```

```
> sapply(y, "[", 1) # on the fly
```

```
[1] "I"      "like" "R"
```

```
> sapply(y, "[", 2) # on the fly
```

```
[1] "really" "writing" "code"
```

Data Merging/Append

- Merging - joining data sets together - usually on key variables, usually "id"
- `merge()` is the most common way to do this with data sets
- `rbind/cbind` - row/column bind, respectively
 - `rbind` is the equivalent of "appending" in Stata or "setting" in SAS
 - `cbind` allows you to add columns in addition to the previous ways
- `reshape2` package also has a lot of information about different ways to reshape data (wide to long, etc) - but has a different (and sometimes more intuitive syntax)
- `t()` is a function that will transpose the data

Merging

```
> base <- data.frame(id = 1:10, Age = seq(55, 60, length = 10))  
> base[1:2, ]
```

```
  id  Age  
1  1 55.00  
2  2 55.56
```

```
> visits <- data.frame(id = rep(1:8, 3), visit = rep(1:3, 8), Outcome = seq(10,  
+      50, length = 24))  
> visits[1:2, ]
```

```
  id visit Outcome  
1  1     1   10.00  
2  2     2   11.74
```

```
> merged.data <- merge(base, visits, by = "id")  
> merged.data[1:5, ]
```

	id	Age	visit	Outcome
1	1	55.00	1	10.00
2	1	55.00	3	23.91
3	1	55.00	2	37.83
4	2	55.56	2	11.74
5	2	55.56	1	25.65

```
> dim(merged.data)
```

```
[1] 24 4
```

```
> all.data <- merge(base, visits, by = "id", all = TRUE)
> tail(all.data)
```

	id	Age	visit	Outcome
21	7	58.33	2	48.26
22	8	58.89	2	22.17
23	8	58.89	1	36.09
24	8	58.89	3	50.00
25	9	59.44	NA	NA
26	10	60.00	NA	NA

```
> dim(all.data)
```

```
[1] 26 4
```

Aside: Dates

You can convert date-like strings in the `Date` class (<http://www.statmethods.net/input/dates.html> for more info)

```
> circ = read.csv("data/Charm_City_Circulator_Ridership.csv", as.is = TRUE)
> head(sort(circ$date))
```

```
[1] "01/01/2011" "01/01/2012" "01/01/2013" "01/02/2011" "01/02/2012"
[6] "01/02/2013"
```

```
> circ$date <- as.Date(circ$date, "%m/%d/%Y") # creating a date for sorting
> head(circ$date)
```

```
[1] "2010-01-11" "2010-01-12" "2010-01-13" "2010-01-14" "2010-01-15"
[6] "2010-01-16"
```

```
> head(sort(circ$date))
```

```
[1] "2010-01-11" "2010-01-12" "2010-01-13" "2010-01-14" "2010-01-15"
[6] "2010-01-16"
```

Data Reshaping

Disclaimer: the `reshape` command in R is not remarkably intuitive.

- Wide - multiple measurements are variables / columns so that the data gets wider with more measurements
- Long - multiple measurements are rows so data gets longer with more measurements
- One example would be many ids with multiple visits

Example of Long/Wide

```
> head(wide)
```

```
  id visit1 visit2 visit3  
1  1   Good   Good   Bad
```

```
> head(long)
```

```
  id visit Outcome  
1  1     1    Good  
2  1     2    Good  
3  1     3    Bad
```

Data Reshaping

- Good resource: <http://www.ats.ucla.edu/stat/r/faq/reshape.htm>

```
> head(Indometh) # this is long
```

```
Subject time conc
1      1 0.25 1.50
2      1 0.50 0.94
3      1 0.75 0.78
4      1 1.00 0.48
5      1 1.25 0.37
6      1 2.00 0.19
```

Data Reshaping

```
> wide <- reshape(Indometh, v.names = "conc", idvar = "Subject", timevar = "time",  
+   direction = "wide")  
> head(wide)
```

	Subject	conc.0.25	conc.0.5	conc.0.75	conc.1	conc.1.25	conc.2	conc.3
1	1	1.50	0.94	0.78	0.48	0.37	0.19	0.12
12	2	2.03	1.63	0.71	0.70	0.64	0.36	0.32
23	3	2.72	1.49	1.16	0.80	0.80	0.39	0.22
34	4	1.85	1.39	1.02	0.89	0.59	0.40	0.16
45	5	2.05	1.04	0.81	0.39	0.30	0.23	0.13
56	6	2.31	1.44	1.03	0.84	0.64	0.42	0.24
	conc.4	conc.5	conc.6	conc.8				
1	0.11	0.08	0.07	0.05				
12	0.20	0.25	0.12	0.08				
23	0.12	0.11	0.08	0.08				
34	0.11	0.10	0.07	0.07				
45	0.11	0.08	0.10	0.06				
56	0.17	0.13	0.10	0.09				

Data Reshaping

```
> dim(Indometh)
```

```
[1] 66 3
```

```
> wide
```

```
  Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25 conc.2 conc.3
1         1         1.50      0.94      0.78  0.48      0.37  0.19  0.12
12        2         2.03      1.63      0.71  0.70      0.64  0.36  0.32
23        3         2.72      1.49      1.16  0.80      0.80  0.39  0.22
34        4         1.85      1.39      1.02  0.89      0.59  0.40  0.16
45        5         2.05      1.04      0.81  0.39      0.30  0.23  0.13
56        6         2.31      1.44      1.03  0.84      0.64  0.42  0.24
  conc.4 conc.5 conc.6 conc.8
1   0.11  0.08  0.07  0.05
12  0.20  0.25  0.12  0.08
23  0.12  0.11  0.08  0.08
34  0.11  0.10  0.07  0.07
45  0.11  0.08  0.10  0.06
56  0.17  0.13  0.10  0.09
```

Data Reshaping

- If you've reshaped a data set - to get it back, just reshape it again

```
> reshape(wide, direction = "long")[1:10, ]
```

```
      Subject time conc
1.0.25      1 0.25 1.50
2.0.25      2 0.25 2.03
3.0.25      3 0.25 2.72
4.0.25      4 0.25 1.85
5.0.25      5 0.25 2.05
6.0.25      6 0.25 2.31
1.0.5       1 0.50 0.94
2.0.5       2 0.50 1.63
3.0.5       3 0.50 1.49
4.0.5       4 0.50 1.39
```

Note the row name change

Data Reshaping - A Better Example

```
> TB <- read.xlsx(file = "data/indicator_estimatedincidencealltbper100000.xlsx",  
+ sheetName = "Data")  
> head(TB, 1)
```

```
TB.incidence..all.forms..per.population.per.year. X1990 X1991  
1 Afghanistan 168 168  
X1992 X1993 X1994 X1995 X1996 X1997 X1998 X1999 X2000 X2001 X2002 X2003  
1 168 168 168 168 168 168 168 168 168 168 168  
X2004 X2005 X2006 X2007 NA.  
1 168 168 168 168 NA
```

```
> TB$NA. <- NULL  
> head(TB, 1)
```

```
TB.incidence..all.forms..per.population.per.year. X1990 X1991  
1 Afghanistan 168 168  
X1992 X1993 X1994 X1995 X1996 X1997 X1998 X1999 X2000 X2001 X2002 X2003  
1 168 168 168 168 168 168 168 168 168 168 168  
X2004 X2005 X2006 X2007  
1 168 168 168 168
```

Data Reshaping - A Better Example

```
> colnames(TB) <- c("Country", paste("Year", 1990:2007, sep = "."))  
> head(TB, 1)
```

```
      Country Year.1990 Year.1991 Year.1992 Year.1993 Year.1994 Year.1995  
1 Afghanistan      168      168      168      168      168      168  
  Year.1996 Year.1997 Year.1998 Year.1999 Year.2000 Year.2001 Year.2002  
1          168      168      168      168      168      168  
  Year.2003 Year.2004 Year.2005 Year.2006 Year.2007  
1          168      168      168      168      168
```

Data Reshaping - More is better!

```
> TB.long <- reshape(TB, idvar = "Country", v.names = "Cases", times = 1990:2007,  
+   direction = "long", timevar = "Year", varying = paste("Year", 1990:2007,  
+   sep = "."))  
>  
> head(TB.long, 4)
```

```
                Country Year Cases  
Afghanistan.1990 Afghanistan 1990 168  
Albania.1990      Albania 1990 25  
Algeria.1990     Algeria 1990 38  
American Samoa.1990 American Samoa 1990 21
```

```
> rownames(TB.long) <- NULL  
> head(TB.long, 4)
```

```
                Country Year Cases  
1 Afghanistan 1990 168  
2 Albania 1990 25  
3 Algeria 1990 38  
4 American Samoa 1990 21
```