Lecture 4

Homework Review and Recoding I

2018 R Teaching Team

September 5, 2018

Acknowledgements

1. Mike Fliss & Sara Levintow!
2. stackoverflow (particularly user `David` for lecture styling - link)
3. R Markdown: The Definitive Guide - link Yihui Xie, J. J. Allaire, Garrett Grolemund
4. R & Rstudio Teams

This Lecture

Goals of Lecture

1. Discuss HW
2. Review factors, functions, and subsetting by coding!

We are going to review all that we learned this week by reading and working with the births dataset.

Overview of Lecture

1. Review Homework Structure/Study Question
2. Reading in Data
3. Subsetting and Recoding Data

Structure

Contains `R code` in grey boxes and `R output` followed by ##.

Overview of Lecture

1. **Review Homework Structure/Study Question**
2. Reading in Data
3. Subsetting and Recoding Data

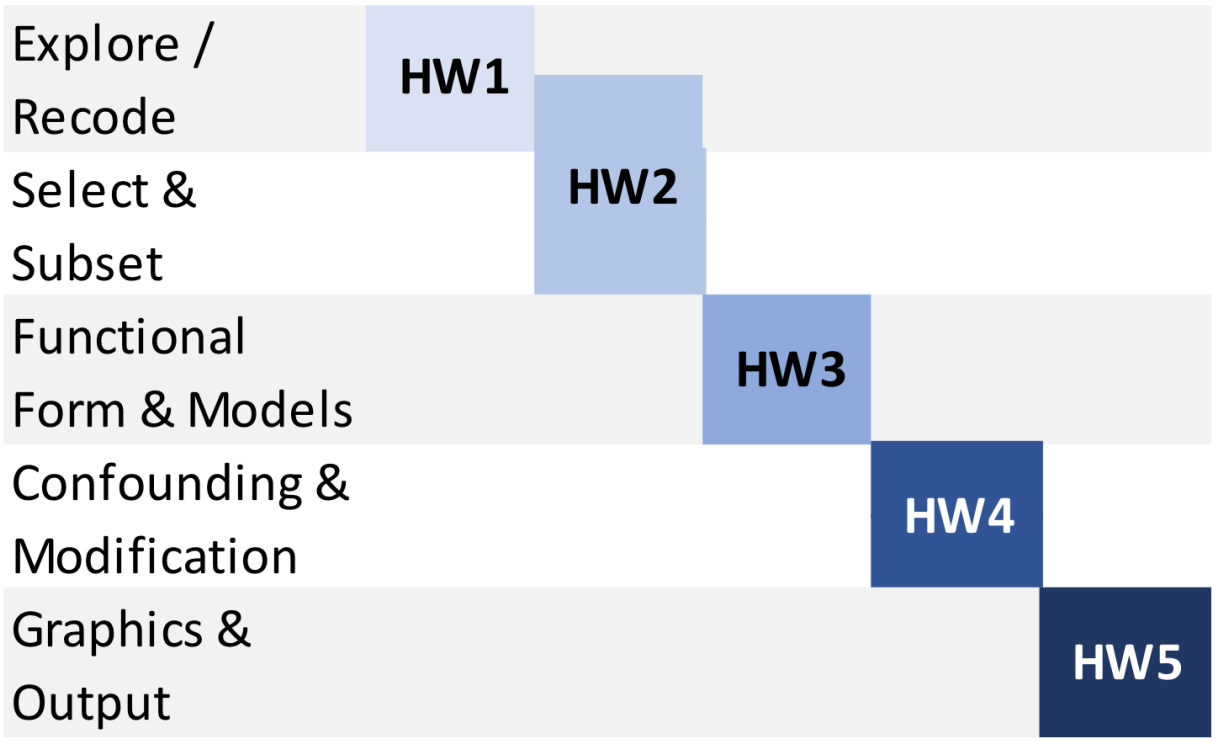
Homework Review

For our homework seriess, we will analyze the effect of **prenatal care** (exposure) on **preterm birth** (outcome). To estimate this effect we will need to do the basics of any epidemiology project:

1. Clean the data (i.e. "Datawrangling"" and making the data tidy)
2. Descriptive Statistics
    - 2.1 Univariate Analyses
    - 2.2 Bivariate Analyses
    - 2.3 Data Visualization
    - 2.4 More data visualization
3. Modeling

Visualizing the Homeworks

Prototypical Epi Analysis



Note a little overlap of HW2. We'll occasionally learn some useful tools "out of order" – slightly more advanced concepts that you'd often want to pull out right away. But generally we're working in order.

Motivating Question

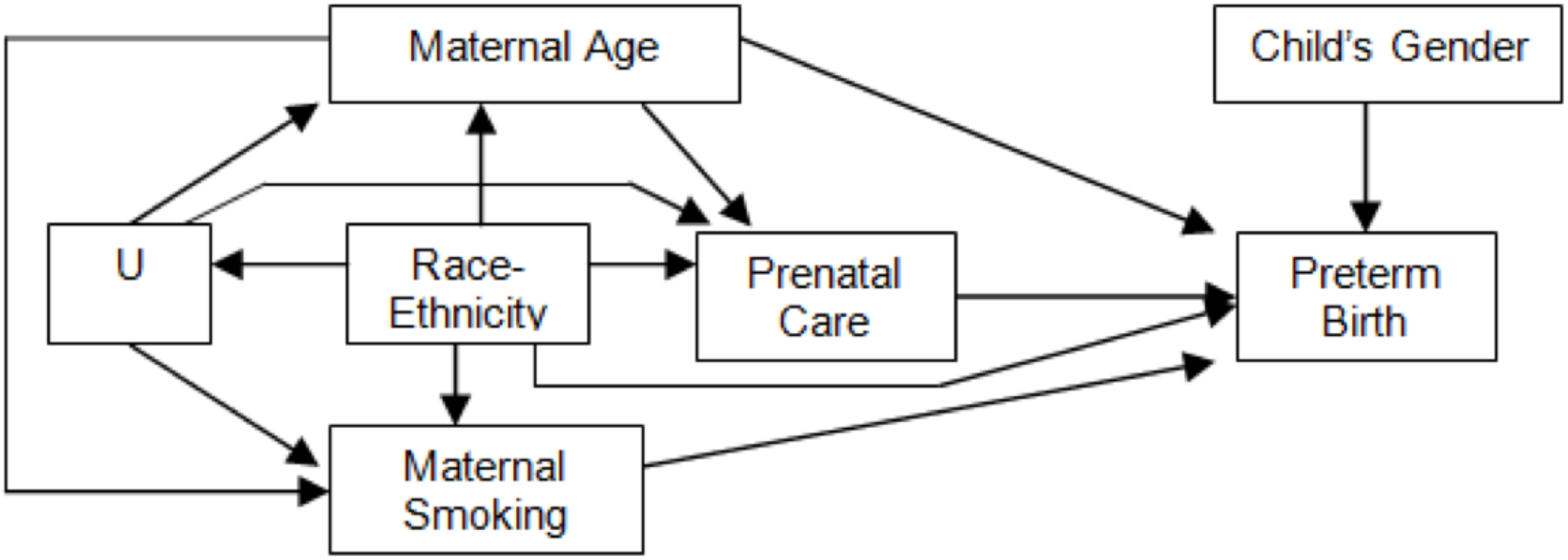
Does early prenatal care (PNC during or before the 5th month of pregnancy) reduce preterm birth (defined as birth before 37 weeks) when controlling for [obvious] confounders.

Punts – We will not cover:

- Forming a research question
- Core EPID concepts (i.e. confounding, EMM, etc)
- Biostat concepts (i.e. validity of models, distributions, etc)

This is all to say that we have made decisions for you in order to focus on the coding. You will cover these topics in EPID716, EPID718, EPID722, Bios454, +/- Bios665 but we welcome feedback/questions!

DAG



Directed Acyclic Graphs (DAGs) inform our variable selection and treatment in models (based on their status as mediators, confounders, effect measure modifiers, etc. We will not elaborate in this class!
Take the Epi sequence for more.

DAG from EPID 716 / Christy Avery

Relevant Variables I

EXPOSURE/OUTCOME

Mdif: Month Prenatal Care Began
Wksgest: Calculated Estimate of Gestation

COVARIATES

Mage: Maternal age
Mrace: Maternal Race
Methnic: Hispanic Origin of Mother
Cigdur: Cigarrette Smoking During Pregnancy
Cores: Residence of Mother – County

Throughout the homeworks we will be creating modified versions of these covariates and using them for plotting, modeling, and mapping!

Relevant Variables II

SELECTION CRITERIA

Plur: Plurality of birth (twins, triplets, etc.)
Wksgest: Calculated Estimate of Gestation
DOB: Date of birth of baby
Congenital Anomalies: multiple variables with congenital anomaly status
Sex: Infant sex
Visits: Total Number of Prenatal Care Visits

We will use these variables to determine the eligible birth observations.

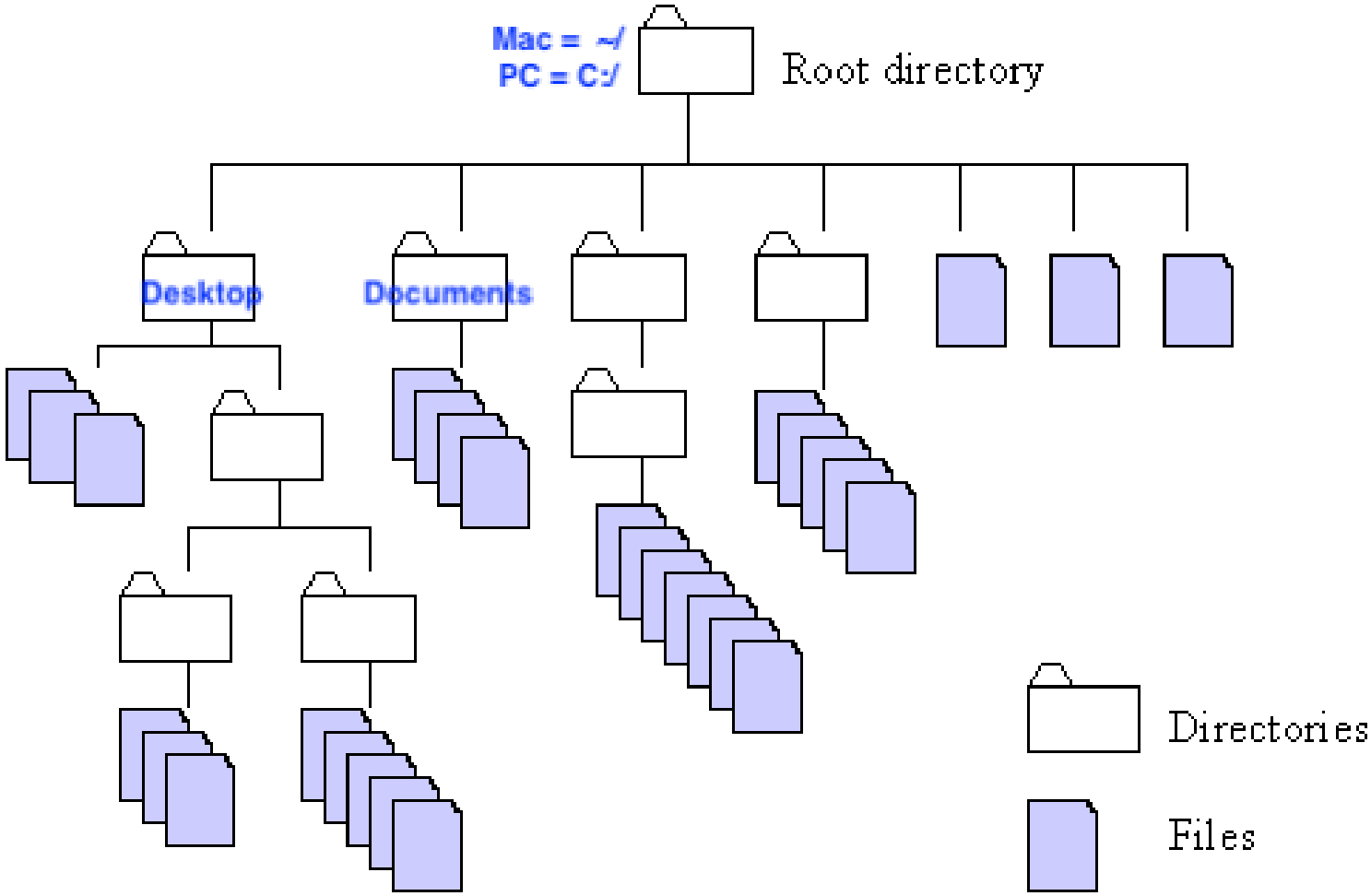
Overview of Lecture

1. Review Homework Structure/Study Question
2. **Reading in Data**
3. Subsetting and Recoding Data

Reading in Data/OS I

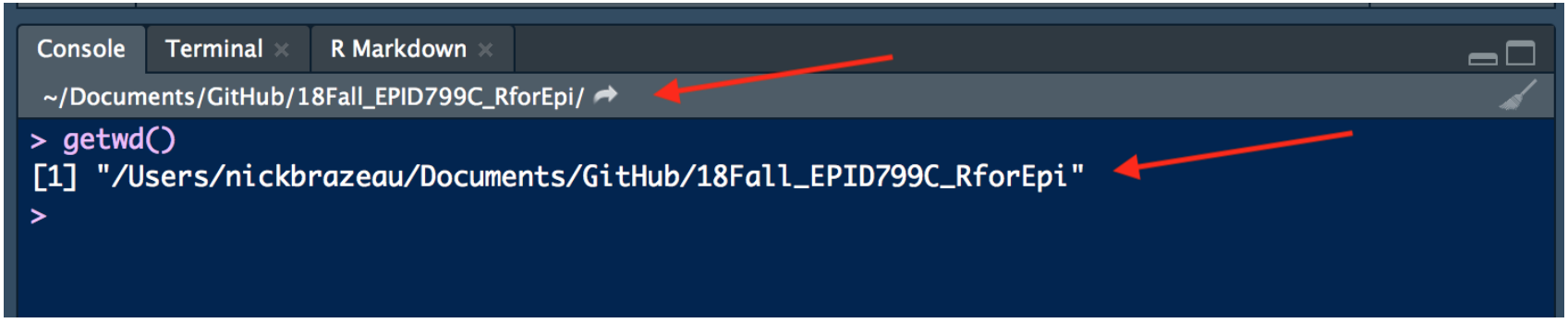
Before we read (import) data into R, let's review how OS File Systems are set up:

OS File System Structure



Reading in Data/OS II

Where R Thou



Changing "Levels"

```
setwd("~/Documents")
setwd("../")
```

Reading in Data/OS III

Organization

Project setup and file management are really important topics that we will cover later. Jenny Bryan has several great articles that are linked on the LearnR website

My advice

- Be careful with relative paths (i.e. "../../mydata.csv")
  - Consider writing out the full path from the root
- Use `Projects` (see Jenny Bryan above) *
- Use GitHub or another form of version control *

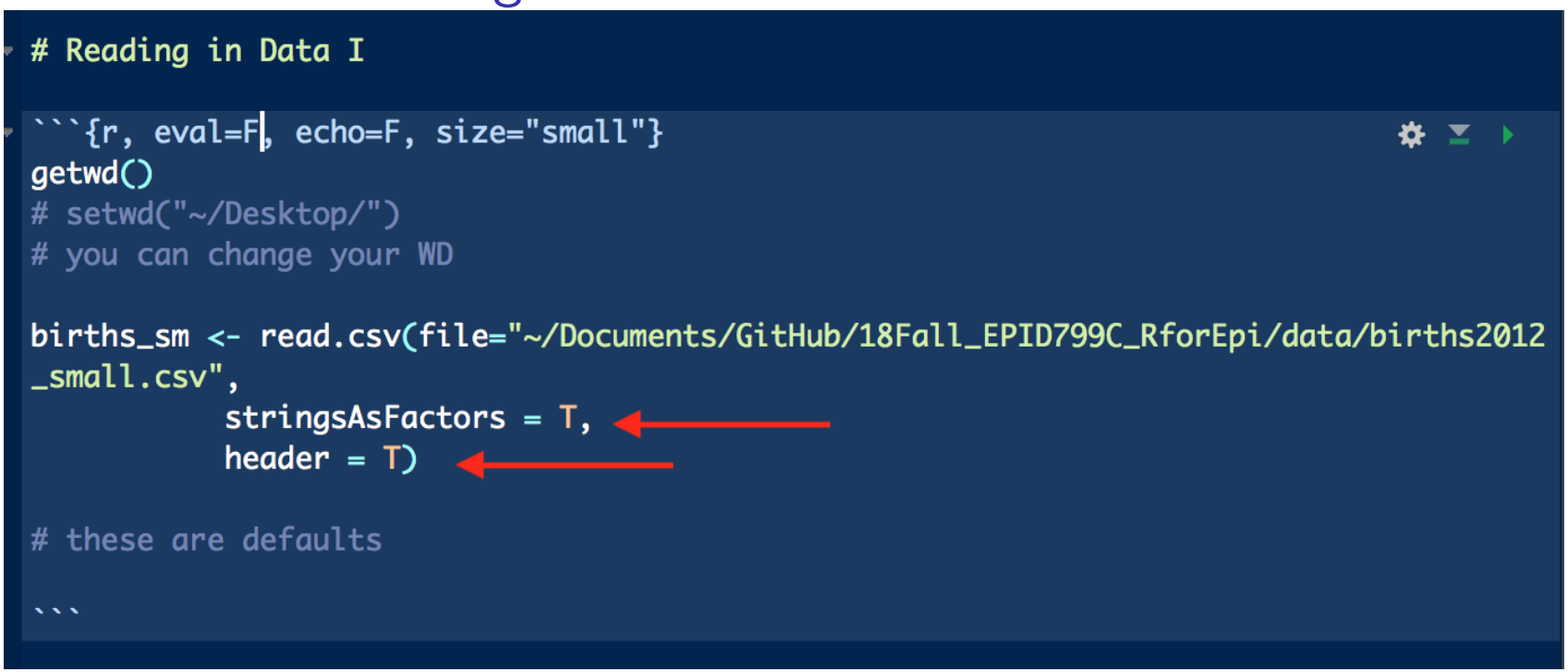
"*" *Advanced content*

Reading in Data IV

Code

```
births_sm <- read.csv(file="~/Documents/GitHub/18Fall_EPID799C_RforEpi/data/births2012_small.csv",
           stringsAsFactors = TRUE,
           header = TRUE)

# Above are default settings
# ?read.csv

colnames(births_sm) <- tolower(colnames(births_sm)) # time-saver
```

Note Default Settings



Reading in Data V

`stringsAsFactors = TRUE`

- We are saying, "Make all strings (i.e. characters) factors"
- (Will review factors in recoding section)

```
apply(births_sm, 2, is.factor)
# ^ you in a few weeks
```

```
##    mage     sex    mdif  visits wksgest   mrace  cigdur   cores     dob
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
##    bfed
##   FALSE
```

No strings/characters in this dataset but . . .

Reading in Data VI

When is `stringsAsFactors = TRUE` a problem?

Factors have additional attributes (i.e. levels) that make them behave differently than strings/characters or numerics. Normally, this is a good thing but can cause problems when:

- You don't expect a variable to be a factor
- Missing data was coded differently than you expected
  - Your datafarme has a mix of numerics & characters (i.e. you have age but someone coded missing as ".")

Reading in Data VII: SUMMARY

Factors do weird things

We are going to talk about some of them. . .

Recommended to always use `stringsAsFactors = FALSE`

- Characters are more flexible than factors
- You should make your own factors (NEXT!)

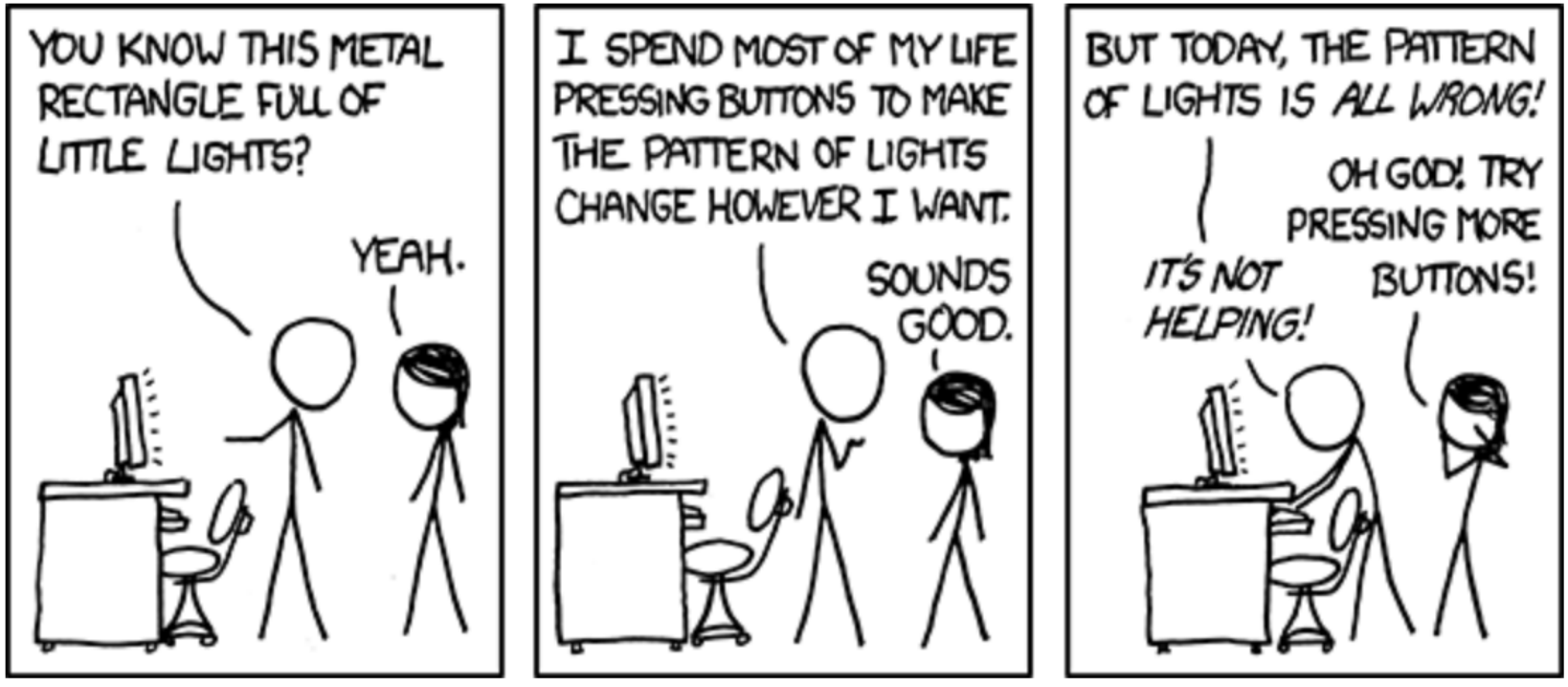
```
births_sm <- read.csv(file="~/Documents/GitHub/18Fall_EPID799C_RforEpi/data/births2012_small.csv",
           stringsAsFactors = FALSE,
           header = TRUE)

colnames(births_sm) <- tolower(colnames(births_sm))
```

Reading in Data VIII

Data of Other Types?

- SAS or STATA or SPSS
  - `haven::`
    - Saves your labels from a `SAS/SPSS/STATA` file
- SAS dataset that is "locked"
  - `sas7bdat::`
- Text files (or ambiguous-ish types)
  - `read.table or readr::read_tsv` – will discuss later

Checking In


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

Overview of Lecture

1. Review Homework Structure/Study Question
2. Reading in Data
3. **Subsetting and Recoding Data**
    - 3.1 **Code Breakout**
    - 3.2 Births Dataset & Your Homework :)

Code Breakout

Create ABC

Below, I have "simulated" a flat file that has three columns and 8 rows. These rows and columns are made up of the first 24 letters of the alphabet with columns named `first`, `second`, and `third` (8-letters each). We are going to start the lecture by playing around with the ABCs. We will then move to the births data set.

```
# this is me making the dataframe
# not tidy or pretty code
abc <- data.frame(matrix(letters[1:24],
                         ncol=3,
                         dimnames = list(NULL, c("first", "second", "third"))),
                  stringsAsFactors = FALSE)
```

On the next slide, I have provided some code working with subsetting. Before you run the code, think about what the output will be!

Code Breakout – Challenge Questions

```r
# print(abc)

abc[1, ]
abc[, 1]

identical(abc[, 1], abc$first)

#--------------------

# print(abc)

abc[1:3, ]
abc[c(1,4,5), ]
abc[, c(1,3)]

#--------------------

abc$first == "a"

abc$first

abc$first[ abc$first == "a" ]

#--------------------

abc == "a"

abc[ abc == "a" ]

abc[ abc == "a" ] <- "NFB"
# ^Change to your initials!
```

Code Breakout Answers I

```
print(abc)

abc[1, ]
abc[, 1]

identical(abc[, 1], abc$first)
```

```
##   first second third
## 1     a      i     q
## 2     b      j     r
## 3     c      k     s
## 4     d      l     t
## 5     e      m     u
## 6     f      n     v
## 7     g      o     w
## 8     h      p     x
```

```
##   first second third
## 1     a      i     q
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```

```
## [1] TRUE
```

Code Breakout Answers II

```
print(abc)
abc[1:3, ]
abc[c(1,4,5), ]
abc[, c(1,3)]
```

```
##   first second third
## 1     a      i     q
## 2     b      j     r
## 3     c      k     s
## 4     d      l     t
## 5     e      m     u
## 6     f      n     v
## 7     g      o     w
## 8     h      p     x
```

```
##   first second third
## 1     a      i     q
## 2     b      j     r
## 3     c      k     s
```

```
##   first second third
## 1     a      i     q
## 4     d      l     t
## 5     e      m     u
```

```
##   first third
## 1     a     q
## 2     b     r
## 3     c     s
## 4     d     t
## 5     e     u
## 6     f     v
## 7     g     w
## 8     h     x
```

Code Breakout Answers III

```
print(abc)

abc$first == "a"

abc$first

abc$first[ abc$first == "a" ]
```

```
##   first second third
## 1     a      i     q
## 2     b      j     r
## 3     c      k     s
## 4     d      l     t
## 5     e      m     u
## 6     f      n     v
## 7     g      o     w
## 8     h      p     x

## [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

## [1] "a" "b" "c" "d" "e" "f" "g" "h"

## [1] "a"
```

We are taking a vector of TRUE-FALSES (logicals) and "layering" that over our dataframe column (which is really a vector too) and only returning "positions" that evaluated to TRUE.

Code Breakout Answers IV

```r
print(abc)

abc == "a"

abc[ abc == "a" ]

abc[ abc == "a" ] <- "NFB"
# Change to your initials!

abc[1,1]
```

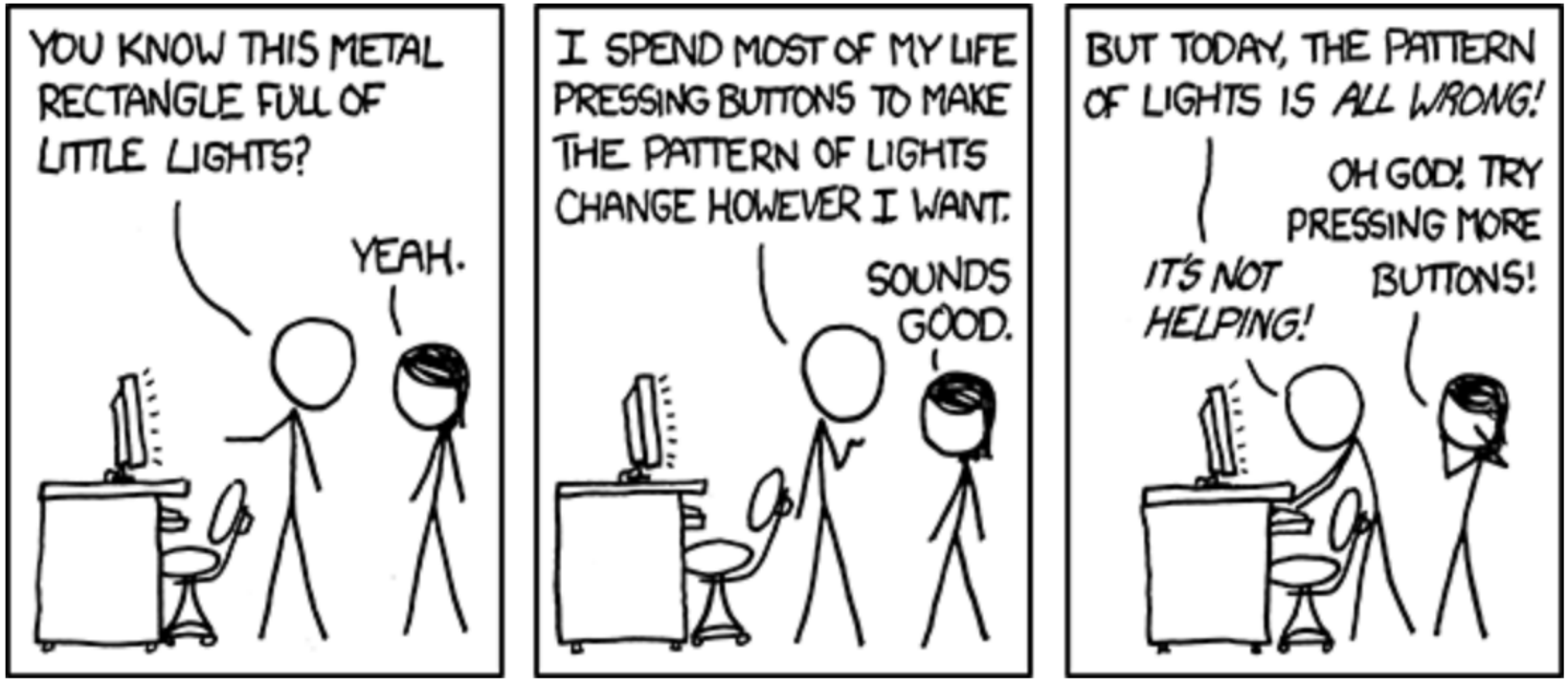
```
##   first second third
## 1     a      i     q
## 2     b      j     r
## 3     c      k     s
## 4     d      l     t
## 5     e      m     u
## 6     f      n     v
## 7     g      o     w
## 8     h      p     x
```

```
##      first second third
## [1,]  TRUE  FALSE FALSE
## [2,] FALSE  FALSE FALSE
## [3,] FALSE  FALSE FALSE
## [4,] FALSE  FALSE FALSE
## [5,] FALSE  FALSE FALSE
## [6,] FALSE  FALSE FALSE
## [7,] FALSE  FALSE FALSE
## [8,] FALSE  FALSE FALSE
```

```
## [1] "a"
```

```
## [1] "NFB"
```

Checking In


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

Overview of Lecture

1. Review Homework Structure/Study Question
2. Reading in Data
3. **Subsetting and Recoding Data**
    - 3.1 Code Breakout
    - 3.2 **Births Dataset & Your Homework :)**

Subsetting & Recoding

Of note, this section of the lecture borrows heavily from Hadley's second chapter: Subsetting

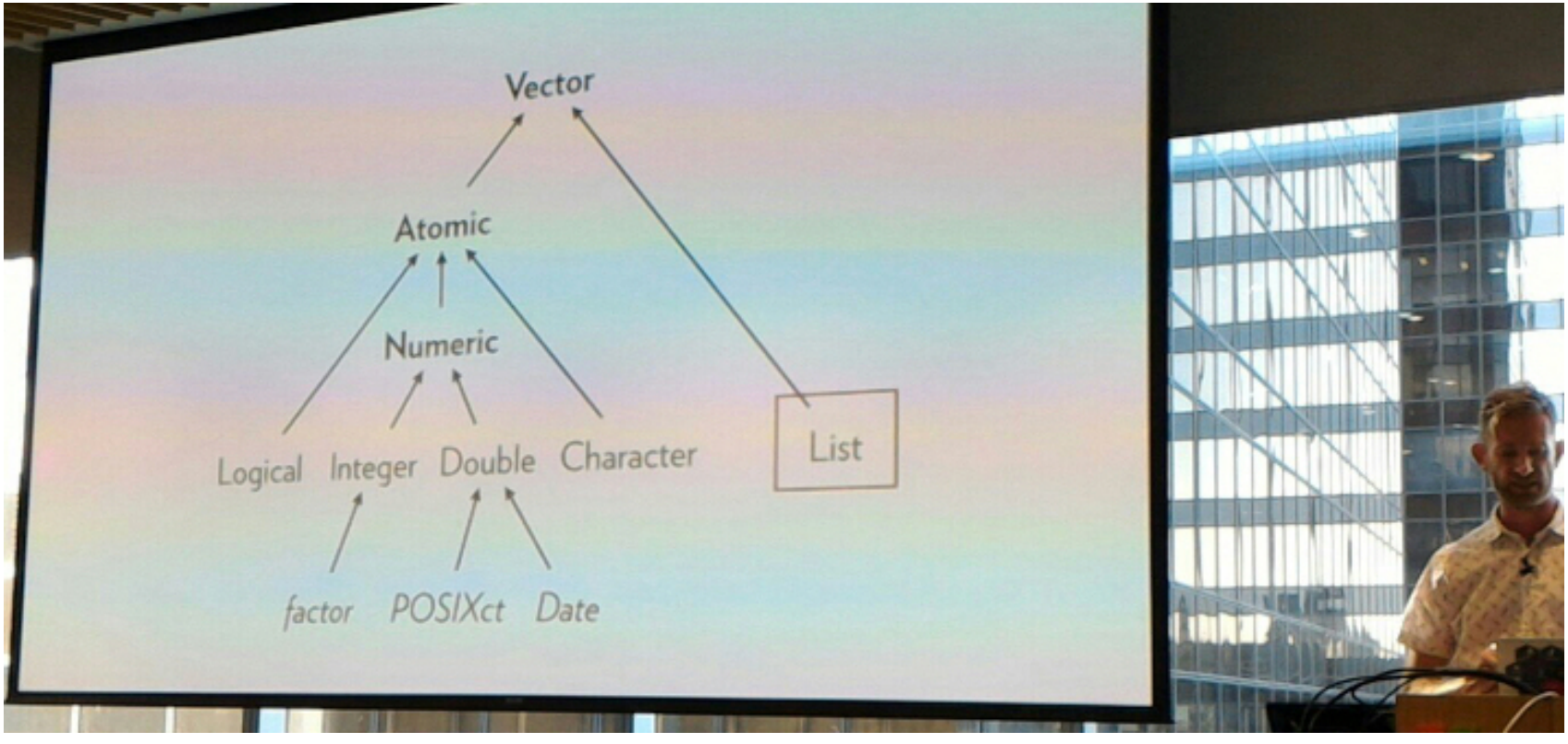
REMINDER: R is case sensitive (meaning you can't mix upper and lower cases). This is to say:
`A != a`
Be careful when evaluating characters, etc. Consider using `tolower()` to make all characters, colnames, etc. lowercase.

Subsetting & Recoding I

Hierarchy of Data & Vector Types

Advanced Content – We will circle back throughout the semester

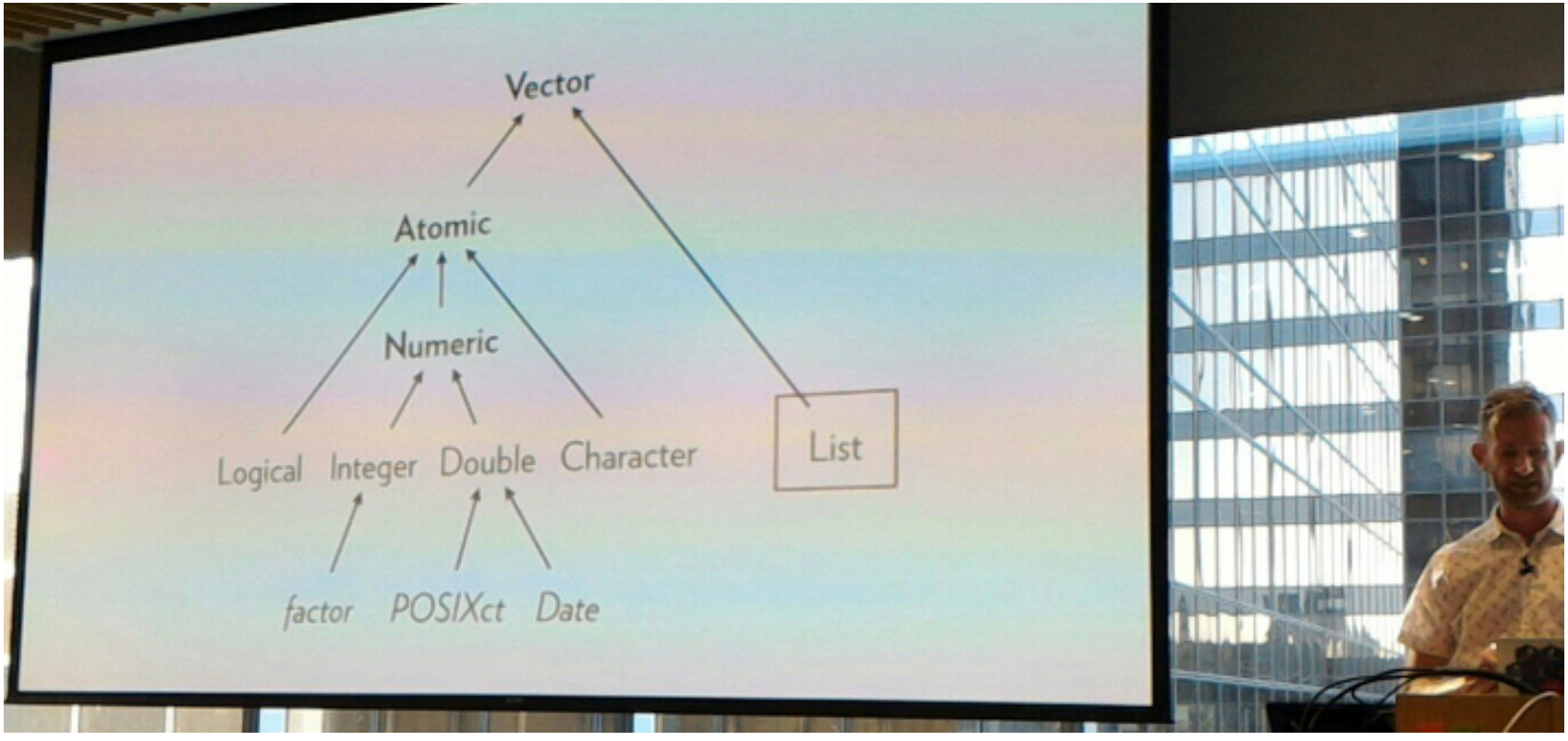
Two Types of Vectors: `Atomic Vectors` (homogenous) and `Lists` (heterogenous)



Subsetting & Recoding I

Hierarchy of Data & Vector Types

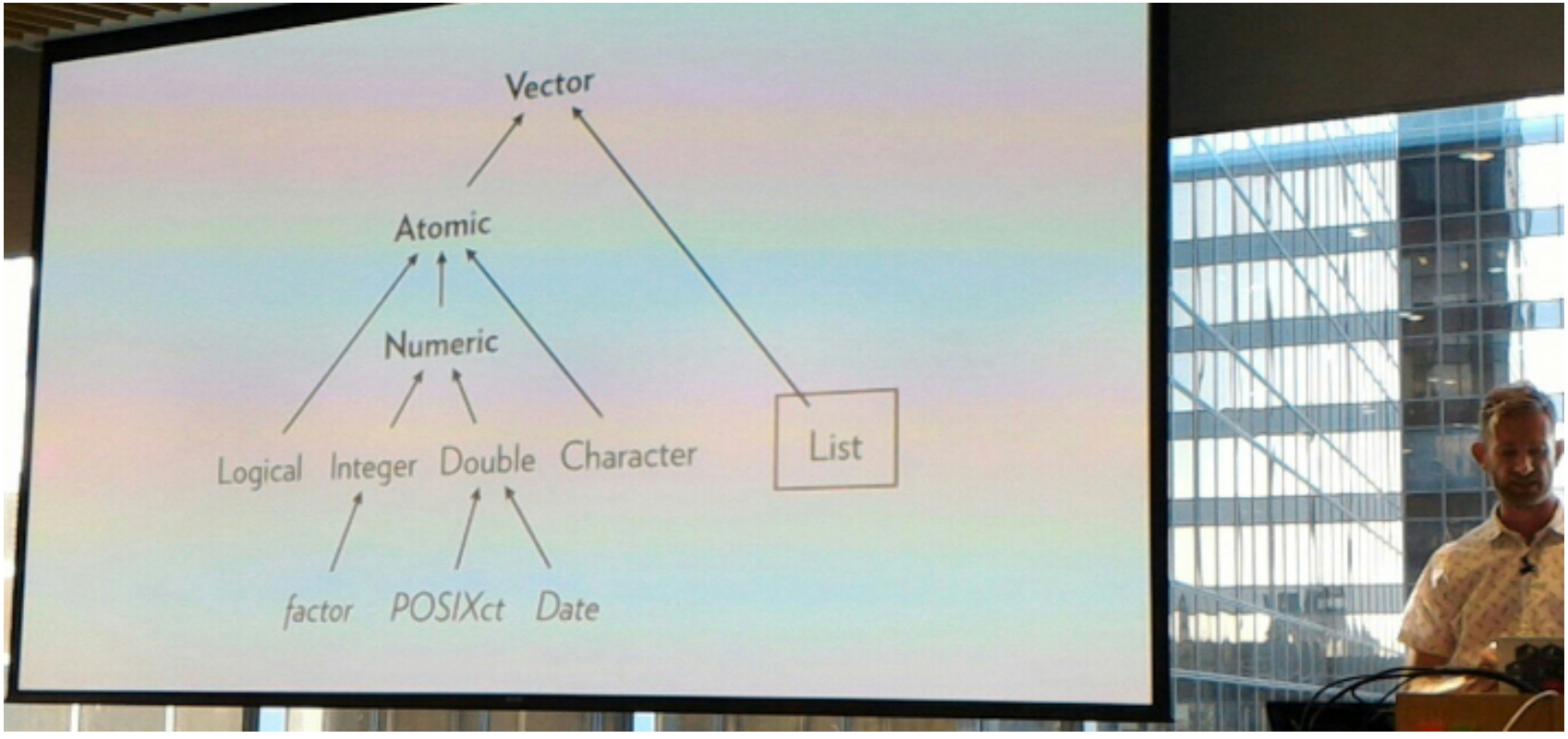
Three Types of (common) Atomic Vectors: `logical`, `numeric`, and `character`. Of note, `numeric` for our purposes encapsulates `double` and `integer`. There is a distinction for the `C` and `Fortran` code that is running under-the-hood in R.



Subsetting & Recoding I

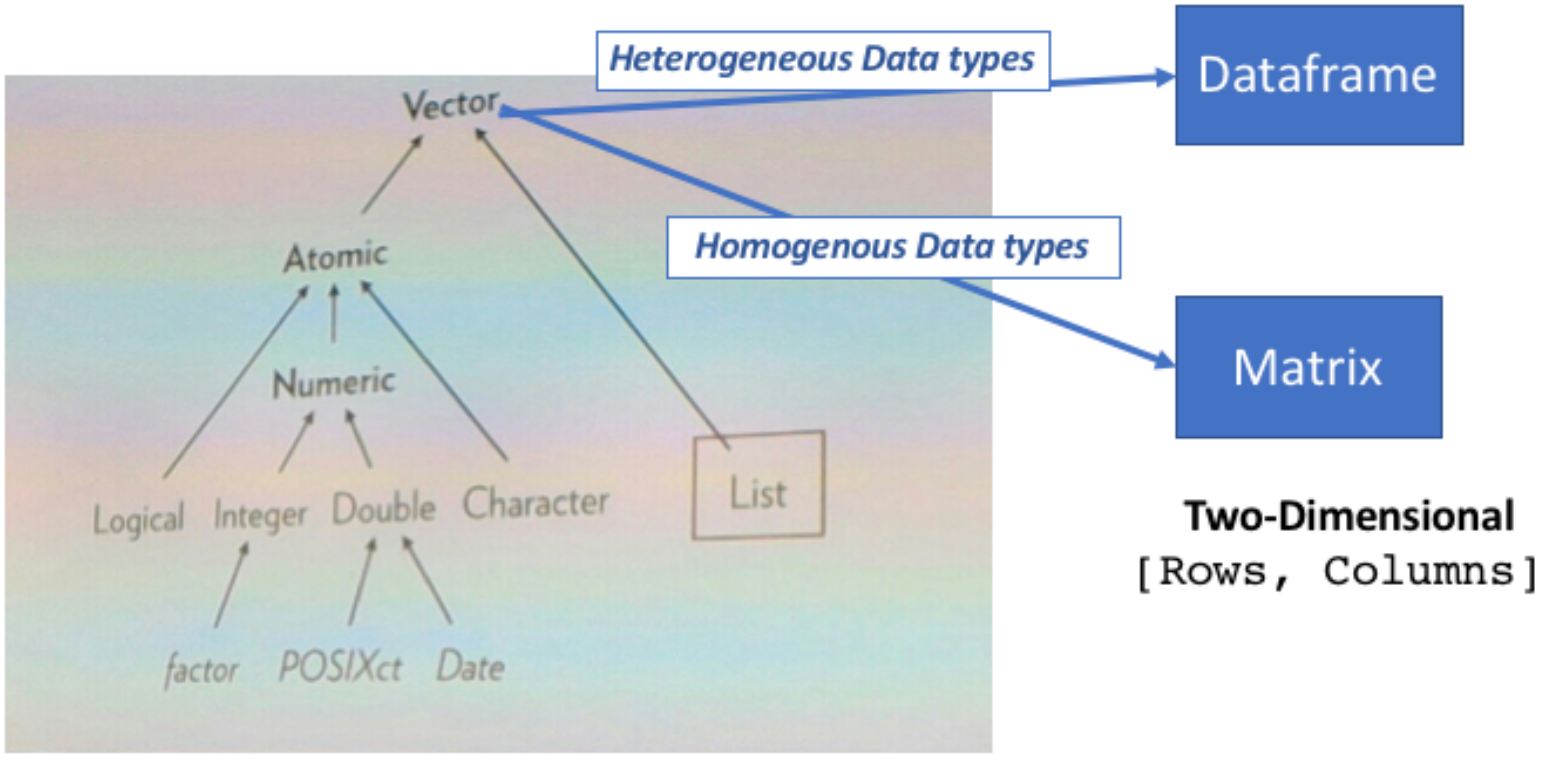
Hierarchy of Data & Vector Types

`Factors` are built on top of integers but contain special attributes that make them behave as categorical variables (i.e. have levels) instead of `integers`.
Dates and times since an origin are really just numerics (`lubridate` can handle dates differently).



Subsetting & Recoding I

Hierarchy of Data & Vector Types



Advanced R, Chapter 1

Therefore, matrices and dataframes are a series of vectors all of the same length. Matrices have to have vectors of the same type while dataframes can have a mix of types (i.e. numerics, factors, etc.)

Subsetting & Recoding II

As epidemiologists, data scientist, computer nerds, etc. etc. we spend a majority of our time cleaning messy data. This process of "wrangling (aka munging)" data into a useuable format is estimated to consume 60-80% of your time on any given project.

To clean data, we often use logical evaluations (i.e. `TRUE-FALSES`) and/or conditionals (if-else statements) to either fix old variables, make new variables, etc.

Subsetting & Recoding III

We can subset a vector in five ways:

(You did a lot of this above!)

Positive & Negative Integers

```
abc$first[c(3, 1)]
abc$first[-c(1,2)]
abc$first[c(3, -1)]
```

```
## [1] "c"   "NFB"
```

```
## [1] "c" "d" "e" "f" "g" "h"
```

```
## [1] "Your console will produce an error..."
```

Logicals

```
abc$first[c(TRUE, TRUE)]
abc$first[abc$first == "d"]
# conditionals that evaluate to logicals
```

```
## [1] "NFB" "b"
```

```
## character(0)
```

Subsetting & Recoding III

We can subset a vector in five ways (continued):

Nothing or Zero

These are "advanced" tricks. We will circle back.

```
abc$first[]
abc$first[0]
```

```
## [1] "NFB" "b"   "c"   "d"   "e"   "f"   "g"   "h"


## character(0)
```

Character Names

```
temp <- setNames(abc$first, LETTERS[1:8])
temp
temp[c("A", "B")]
```

```
##     A     B     C     D     E     F     G     H
## "NFB"   "b"   "c"   "d"   "e"   "f"   "g"   "h"


##     A     B
## "NFB"   "b"
```

Subsetting & Recoding IV

As we discussed above, **we can think of dataframes as collections of vectors that are of the same length**. This means the same rules of subsetting a vector can be applied to a dataframe but now with increased dimensionality. Here, I am using "increased dimensionality" to mean that we can subset multiple rows or columns at the same time.

Subsetting & Recoding V

```r
colnames(births_sm)
colnames(births_sm) %in% c("mage", "sex", "mdif")
# temp <- births_sm[, colnames(births_sm) %in% c("mage", "sex", "mdif")
dim( births_sm[1:100, colnames(births_sm) %in% c("mage", "sex", "mdif") ])
```

```
##  [1] "mage"    "sex"     "mdif"    "visits"  "wksgest" "mrace"   "cigdur"
##  [8] "cores"   "dob"     "bfed"
```

```
##  [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [1] 100   3
```

Subsetting & Recoding VII

Let's start with questions 4 and 5 from homework 1.

Question 4

"... using the selection operators [], create a smaller version of the births file called births_sample with only the first 1000 rows and with only these variables:"MAGE", "MDIF", "VISITS", "WKSGEST", "MRACE"."

Thoughts?

Subsetting & Recoding VII

Question 4

"... using the selection operators [], create a smaller version of the births file called births_sample with only the first 1000 rows and with only these variables:“MAGE”, “MDIF”, “VISITS”, “WKSGEST”, “MRACE”."

```
births_tiny <- births_sm[1:1000, ]
# check it out
dim(births_tiny)
```

```
## [1] 1000   10
```

```
colnames(births_tiny)
```

```
##  [1] "mage"    "sex"     "mdif"    "visits"  "wksgest" "mrace"   "cigdur"
##  [8] "cores"   "dob"     "bfed"
```

Subsetting & Recoding VIII

Question 5-ish

- Running the code on the "tiny" dataframe. You should run it on the full births dataframe.

Q5, Part i - iii

To paraphrase, look at a table of the `mdif` variable. Set the missing value (99) to `missing`.
As a reminder, the `mdif` variable is the month prenatal care began. We have defined early prenatal care as care started before the 5th month (main exposure).

Subsetting & Recoding IX

Q5, Part i - iii

To paraphrase, look at a table of the `mdif` variable. Set the missing value (99) to `missing`.

Check out the Varibale

```
# check it out
str(births_tiny$mdif)
```

```
##  int [1:1000] 2 2 7 6 4 4 4 2 1 2 ...
```

```
summary(births_tiny$mdif)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   5.363   4.000  99.000
```

```
table(births_tiny$mdif, useNA = "always")
```

```
##
##    1    2    3    4    5    6    7    8    9   88   99 <NA>
##   87  264  334  139   60   43   33    9    7   10   14    0
```

Subsetting & Recoding X

Q5, Part i - iii

To paraphrase, look at a table of the `mdif` variable. Set the missing values (99) to `missing`.

Set 99 to missing

```
births_tiny$mdif[births_tiny$mdif == 99] <- NA
# check it worked
summary(births_tiny$mdif)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   1.000   2.000   3.000   4.033   4.000  88.000      14
```

```
table(births_tiny$mdif, useNA = "always")
```

```
##
##    1    2    3    4    5    6    7    8    9   88 <NA>
##   87  264  334  139   60   43   33    9    7   10   14
```

```
# boxplot(births_tiny$mdif)
```

Subsetting & Recoding XI

Q5, Part iv

To paraphrase, make a new indicator variable for whether a neonate received prenatal care at or before 5 months (`pnc5` variable). Make the variable a numeric and a factor.

Making `pnc5` numeric – the LONG WAY

```
births_tiny$pnc5 <- NA #init
births_tiny$pnc5[births_tiny$mdif == 1] <- 1
births_tiny$pnc5[births_tiny$mdif == 2] <- 1
births_tiny$pnc5[births_tiny$mdif == 3] <- 1
births_tiny$pnc5[births_tiny$mdif == 4] <- 1
births_tiny$pnc5[births_tiny$mdif == 5] <- 1
births_tiny$pnc5[births_tiny$mdif == 6] <- 0
births_tiny$pnc5[births_tiny$mdif == 7] <- 0
births_tiny$pnc5[births_tiny$mdif == 8] <- 0
births_tiny$pnc5[births_tiny$mdif == 9] <- 0
births_tiny$pnc5[births_tiny$mdif == 88] <- 0

# man that's a lot of key strokes
```

Subsetting & Recoding XI

Q5, Part iv

To paraphrase, make a new indicator variable for whether a neonate received prenatal care before 5 months (`pnc5` variable). Make the variable a numeric and a factor.

Making `pnc5` numeric – the CONDITIONAL WAY

```
births_tiny$pnc5 <- ifelse(births_tiny$mdif <= 5, 1, 0)
```

Subsetting & Recoding XI

Making pnc5 numeric – the CONDITIONAL WAY

A Deeper Dive

```
head( births_tiny$mdif )
```

```
## [1] 2 2 7 6 4 4
```

```
head( births_tiny$mdif <= 5 )
```

```
## [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE
```

```
head( ifelse(births_tiny$mdif <= 5, 1, 0) )
```

```
## [1] 1 1 0 0 1 1
```

```
births_tiny$pnc5 <- ifelse(births_tiny$mdif <= 5, 1, 0)
```

Subsetting & Recoding XI

Q5, Part iv

To paraphrase, make a new indicator variable for whether a neonate received prenatal care before 5 months (`pnc5` variable). Make the variable a numeric and a factor.

Making `pnc5` numeric – the Long CONDITIONAL WAY

```
for(i in 1:length(births_tiny$mdif)){

  if(is.na(births_tiny$mdif[i])){
    births_tiny$pnc5[i] <- NA
  } else if( births_tiny$mdif[i] <= 5){
     births_tiny$pnc5[i] <- 1
    } else (
      births_tiny$pnc5[i] <- 0
  )

}
```

Subsetting & Recoding XI

Q5, Part iv

To paraphrase, make a new indicator variable for whether a neonate received prenatal care before 5 months (`pnc5` variable). Make the variable a numeric and a factor.

Make `pnc5` a factor

```
births_tiny$pnc5_f <- factor(births_tiny$pnc5,
                             levels=c(0,1),
                             labels=c("No Early PNC", "Early PNC"))
```

Subsetting & Recoding – FACTORS

Make pnc5 a factor

- Factors are categorical variables
- **Can be orderd (Ordinal)**
- Can be unordered (Disjoint Indicators)

Ordinal Variable

```
births_tiny$pnc5_f <- factor(births_tiny$pnc5,
                             levels=c(0,1),
                             labels=c("No Early PNC", "Early PNC"),
                             ordered = T)

levels(births_tiny$pnc5_f)
```

```
## [1] "No Early PNC" "Early PNC"
```

Subsetting & Recoding – FACTORS

Make pnc5 a factor

- Factors are categorical variables
- Can be orderd (Ordinal)
- **Can be unordered (Disjoint Indicators)**

Disjoint Indicator Variable

```
births_tiny$pnc5_f <- factor(births_tiny$pnc5,
                             levels=c(0, 1),
                             labels=c("No Early PNC", "Early PNC"))
levels(births_tiny$pnc5_f)
```

```
## [1] "No Early PNC" "Early PNC"
```

Subsetting & Recoding – FACTORS

Factors are Weird (on purpose)

```
str(births_tiny$pnc5_f)
```

```
##  Factor w/ 2 levels "No Early PNC",..: 2 2 1 1 2 2 2 2 2 2 ...
```

```
attributes(births_tiny$pnc5_f)
```

```
## $levels
## [1] "No Early PNC" "Early PNC"
##
## $class
## [1] "factor"
```

```
# levels(births_tiny$pnc5_f)
# sum(births_tiny$pnc5_f) # will throw error
head(births_tiny$pnc5_f == "No Early PNC")
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

```
sum(births_tiny$pnc5_f == "No Early PNC", na.rm = T)
```

```
## [1] 102
```

Subsetting & Recoding – FACTORS

From a Modeling Perspective

Remember disjoint indicator variables? In SAS this is the same as `CLASS` (I think? SAS users?)

Factors are Weird (on purpose)

```
temp <- lm(mage ~ pnc5_f,
    data = births_tiny)

broom::tidy(temp) # future you!
```

```
## # A tibble: 2 x 5
##   term            estimate std.error statistic   p.value
##   <chr>              <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)        25.7      0.601     42.8  6.46e-227
## 2 pnc5_fEarly PNC     1.71     0.635      2.69 7.18e-  3
```

Subsetting & Recoding – FACTORS

From a Modeling Perspective

Factors are Weird (on purpose)

By default, the first level you put in is the referent level. You can reassign the referent level with `relevel`.

```
births_tiny$pnc5_f <- relevel(births_tiny$pnc5_f, ref = "Early PNC")
temp <- lm(mage ~ pnc5_f,
    data = births_tiny)

broom::tidy(temp) # future you!
```

```
## # A tibble: 2 x 5
##   term               estimate std.error statistic p.value
##   <chr>                 <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)           27.4      0.204    134.    0
## 2 pnc5_fNo Early PNC    -1.71     0.635     -2.69 0.00718
```

Subsetting & Recoding XI

Making sure we coded pnc5 correctly

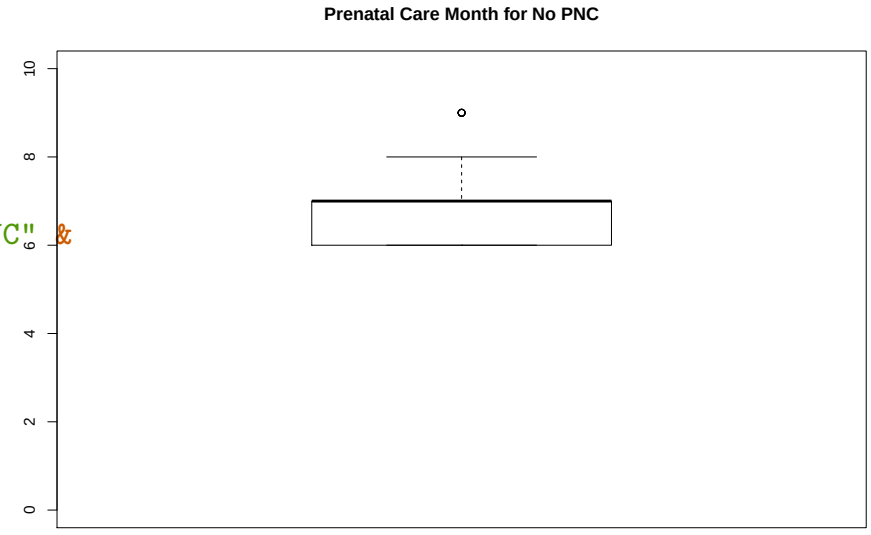
```
table(births_tiny$mdif,
      births_tiny$pnc5_f,
      useNA = "always")
```

```
##
##          Early PNC No Early PNC <NA>
##   1            87             0    0
##   2           264             0    0
##   3           334             0    0
##   4           139             0    0
##   5            60             0    0
##   6             0            43    0
##   7             0            33    0
##   8             0             9    0
##   9             0             7    0
##   88            0            10    0
##   <NA>          0             0   14
```

Subsetting & Recoding XI

Making sure we coded pnc5 correctly

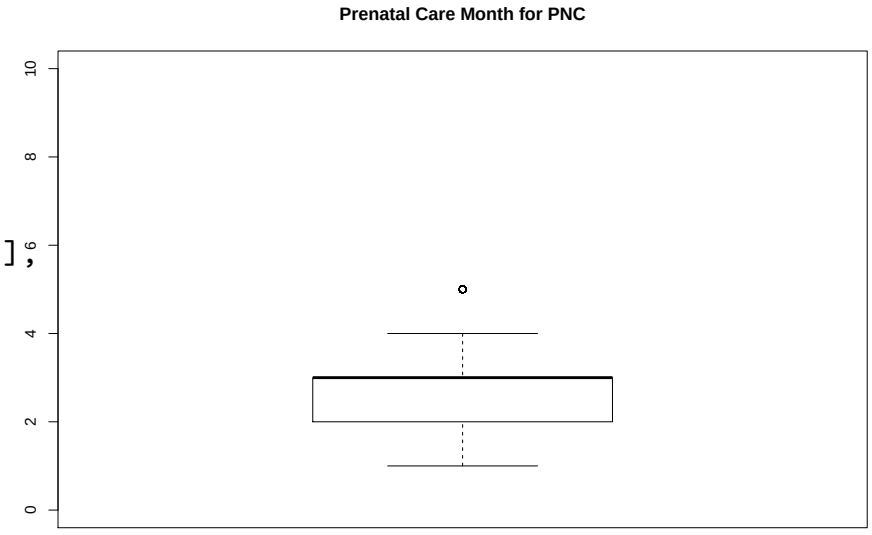
```
boxplot(
  births_tiny$mdif[births_tiny$pnc5_f == "No Early PNC" &
                     births_tiny$mdif != 88],
  ylim = c(0, 10),
  main = "Prenatal Care Month for No PNC"
  )
```



Subsetting & Recoding XI

Making sure we coded pnc5 correctly

```
boxplot(
  births_tiny$mdif[births_tiny$pnc5_f == "Early PNC" ],
  ylim = c(0, 10),
  main = "Prenatal Care Month for PNC"
  )
```



Always Check your Subset and Recoding

Useful functions to call:

- `mean`, `median`, `mode`, `sd`, `range`, `summary`
  - if you know you have missing set `na.rm=T`
- `table` (remember to set `use.NA = "always"`)
- `plot`, `boxplot`

Subsetting & Recoding XII

```
births_tiny$smoker_f <- factor(births_tiny$pnc5,
                               levels=c(0, 1),
                               labels=c("Nonsmoker", "Smoker"))
```

Challenge question

Why does the following code not produce the "expected" result?

```
mean( births_tiny$wksgest[ !is.na(births_tiny$smoker_f == "Smoker") ] )
mean( births_tiny$wksgest[ births_tiny$smoker_f == "Smoker" & !is.na(births_tiny$smoker_f) ] )
```

Subsetting & Recoding XII

Challenge question

Why does the following code not produce the "expected" result?
What is being "subsetted"?

```
sum( !is.na(births_tiny$smoker_f == "Smoker") )
```

```
## [1] 986
```

```
sum( births_tiny$smoker_f == "Smoker" & !is.na(births_tiny$smoker_f) )
```

```
## [1] 884
```

Subsetting & Recoding XII

Challenge question

Why does the following code not produce the "expected" result?
What is being "subsetted"?

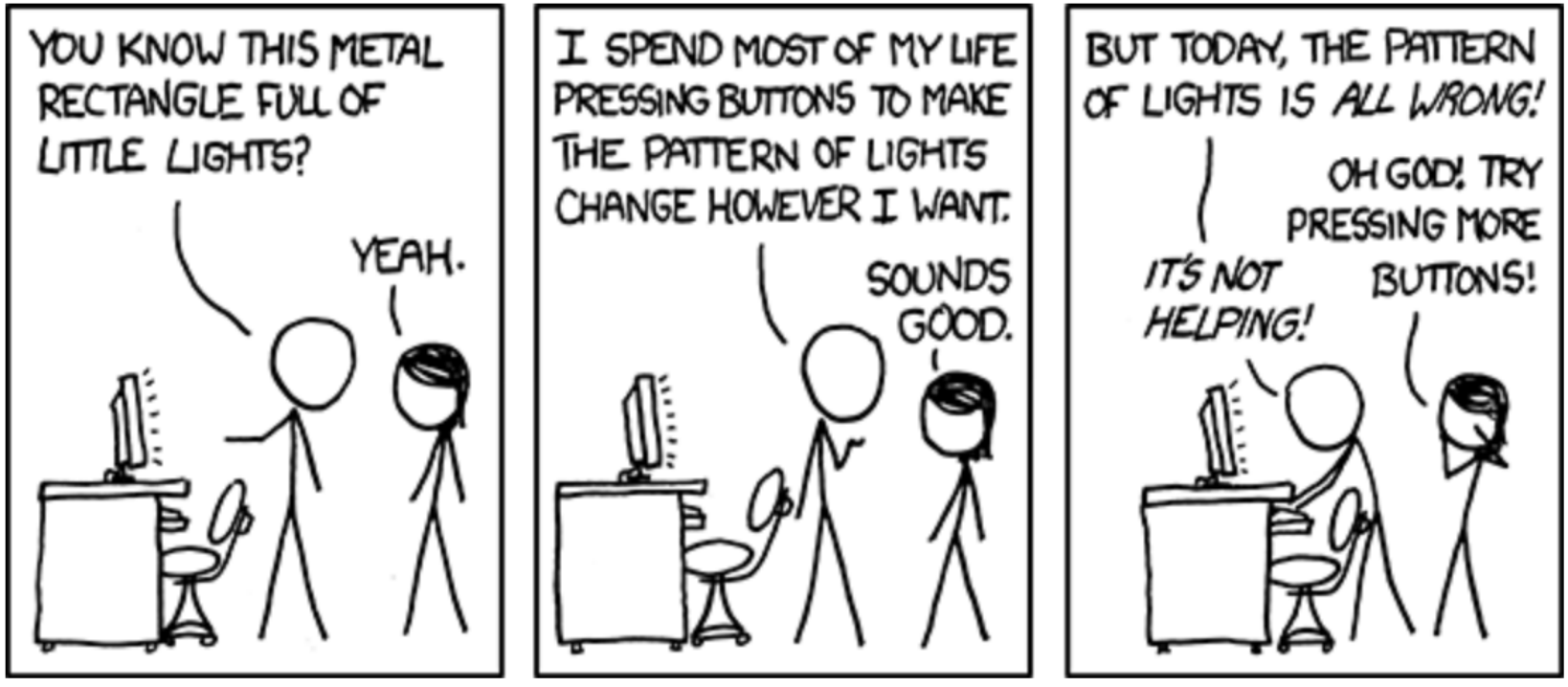
```
table( !is.na(births_tiny$smoker_f == "Smoker"), useNA = "always" )
```

```
##
## FALSE  TRUE  <NA>
##    14   986     0
```

```
table( births_tiny$smoker_f == "Smoker" & !is.na(births_tiny$smoker_f), useNA = "always" )
```

```
##
## FALSE  TRUE  <NA>
##   116   884     0
```

Checking In


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

Useful Function Vocabularly for Data Scientists

Advanced R

- See Hadley's Adv R Chap 3 here

R Team's Favorite (from Base::)

- `all`, `any`, `which`, `ifelse`
- `is.na`
- `str`, `class`, `typeof`
- `head`, `tail`, `dim`

Useful Subsetting & Recoding "Tricks"

Overloading summary

```
summary(births_tiny)
```

Missing Data-Code Known

```
births_temp <- read.csv(file="~/Documents/GitHub/18Fall_EPID799C_RforEpi/data/births2012_small.csv",
           stringsAsFactors = FALSE,
           header = TRUE,
           na.strings = "99") # 99 is missing here...
```

Looking Forward I

Of the many amazing packages within the `tidyverse`, the `readr::`, `dplyr::`, and `forcats::` packages deal with many of the issues we discussed today. We will be spending **A LOT** of time on the `tidyverse` shortly.

- `readr::`
  - `readr::read_csv`
    - default behavior – `stringsAsFactors = FALSE`
    - Tibbles instead of data.frames (discussed here)
- `dplyr::`
  - `dplyr::select` – select columns (to keep)
  - `dplyr::filter` – filter rows (to keep)
- `forcats::`
  - `forcats::fct_relevel`

Looking Forward II

Within the R community (and other data fields) there is a growing effort to have a common language for "tidy" data. Hadley Wickham wrote an excellent article here if you are interested.

We will be exploring these concepts throughout the semester but the basic gist is:

- All observations should have their own row
- All missing data should be coded as `NA`
- All variables should be properly formatted (i.e. factors)

We will also explore the difference between **long** and **wide** data formats.

I mention this only to pique your interest!

Looking Forward II

This lecture was created with `R` and `Rmarkdown`.

`Rmarkdown` is a powerful tool for creating reports, documents, manuscripts, websites, etc. and is a pillar of reproducible research. *It is my favorite feature of R and will be covered in the second-half of the semester.*

Appendix

Subsetting Lists I

From Hadley's Chap 3, "Subsetting a list works in the same way as subsetting an atomic vector. Using [will always return a list; [[and $" pull out items of a list.

You can think of $ as shorthand for [[and this is what we use mostly for dataframes to pull out column vectors.

Subsetting Lists II

Let's look at that...

```
abc_list <- lapply(abc, list)
str(abc_list)
```

```
## List of 3
##  $ first :List of 1
##   ..$ : chr [1:8] "NFB" "b" "c" "d" ...
##  $ second:List of 1
##   ..$ : chr [1:8] "i" "j" "k" "l" ...
##  $ third :List of 1
##   ..$ : chr [1:8] "q" "r" "s" "t" ...
```

Subsetting Lists III

Accessing elements... The main difference is if we want to perserve the list structure with `[` or simplify the structure with `[[`.

```
abc_list["first"]
str(abc_list["first"])
# ^ single element list
#
abc_list[["first"]]
str(abc_list[["first"]])
# ^ object within list
```

```
## $first
## $first[[1]]
## [1] "NFB" "b"   "c"   "d"   "e"   "f"   "g"   "h"


## List of 1
##  $ first:List of 1
##   ..$ : chr [1:8] "NFB" "b" "c" "d" ...


## [[1]]
## [1] "NFB" "b"   "c"   "d"   "e"   "f"   "g"   "h"


## List of 1
##  $ : chr [1:8] "NFB" "b" "c" "d" ...
```

Subsetting Lists III – Summary

From Hadley and @RLangTip,
"If list `x` is a train carrying objects, then `x[[5]]` is the object in car 5; `x[4:6]` is a train of cars 4-6."