```
## Data

## Data Cleaning

## Dealing with Missing Data


## Finding Missing data

x = c(0, NA, 2, 3, 4)
x > 2

x != NA
x > 2 & !is.na(x)

(x == 0 | x == 2) # has NA
(x == 0 | x == 2) & !is.na(x) # No NA
x %in% c(0, 2) # NEVER has NA and returns logical

## Missing Data with Operations

x + 2
x * 2

## Tables and Tabulations

table(x)
table(x, useNA = "ifany")

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

## Creating Two-way Tables {.smaller}

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")

## Finding Row or Column Totals

margin.table(tab, 2)

## Proportion Tables {.smaller}

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

## Download Salary FY2014 Data

Sal = read.csv("Baltimore_City_Employee_Salaries_FY2014.csv",

head(Sal,2)
any(is.na(Sal$Name)) # are there any NAs?
```

~~~
# Recoding Variables

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

## Example of Recoding with `recode`: `car` package


library(car, quietly = TRUE)
x = rep(c("Male", "M", "m", "f", "Female", "female" ),
        each = 3)
car::recode(x, "c('m', 'M', 'male') = 'Male';
            c('f', 'F', 'female') = 'Female';")


## Example of Recoding with `revalue`: `plyr`

library(plyr)
plyr::revalue(x, c("M" = "Male", "m" = "Male",
                   "f" = "Female", "female" = "Female"))

set.seed(4) # random sample below - make sure same every time
gender <- sample(c("Male", "mAle", "MaLe", "M", "MALE", "Ma", "FeMAle",
"F", "Woman", "Man", "Fm", "FEMALE"), 1000, replace = TRUE)

table(gender)

# String functions

## Pasting strings with `paste` and `paste0`


paste("Visit", 1:5, sep = "_")
paste("Visit", 1:5, sep = "_", collapse = " ")
paste("To", "is going be the ", "we go to the store!", sep = "day ")
# and paste0 can be even simpler see ?paste0
paste0("Visit",1:5)

paste(1:5)
paste(1:5, collapse = " ")

## Splitting String: base R

x <- c("I really", "like writing", "R code programs")
y <- strsplit(x, split = " ") # returns a list
y

## Splitting String: `stringr`

library(stringr)
y2 <- str_split(x, " ") # returns a list
y2

## Using a fixed expression
~~~

```r
str_split("I.like.strings", ".")
str_split("I.like.strings", fixed("."))

suppressPackageStartupMessages(library(dplyr)) # must be loaded AFTER plyr
y[[2]]
sapply(y, dplyr::first) # on the fly
sapply(y, nth, 2) # on the fly
sapply(y, last) # on the fly

## 'Find' functions: Finding Indices

grep("Rawlings",Sal$Name)
which(grepl("Rawlings", Sal$Name))
which(str_detect(Sal$Name, "Rawlings"))

## 'Find' functions: Finding Logicals

head(grepl("Rawlings",Sal$Name))
head(str_detect(Sal$Name, "Rawlings"))

grep("Rawlings",Sal$Name,value=TRUE)
Sal[grep("Rawlings",Sal$Name),]

str_subset(Sal$Name, "Rawlings")
Sal %>% filter(str_detect(Name, "Rawlings"))

ss = str_extract(Sal$Name, "Rawling")
head(ss)
ss[ !is.na(ss)]

## Showing differnce in `str_extract` and  `str_extract_all`

head(str_extract(Sal$AgencyID, "\\d"))
head(str_extract_all(Sal$AgencyID, "\\d"), 2)

head(grep("^Payne.*", x = Sal$Name, value = TRUE), 3)

head(grep("Leonard.?S", x = Sal$Name, value = TRUE))
head(grep("Spence.*C.*", x = Sal$Name, value = TRUE))


head(str_subset( Sal$Name, "^Payne.*"), 3)

head(str_subset( Sal$Name, "Leonard.?S"))
head(str_subset( Sal$Name, "Spence.*C.*"))

## Replace

class(Sal$AnnualSalary)

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

## Replace

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

Sal$AnnualSalary <- as.numeric(gsub(pattern = "$", replacement="",
                              Sal$AnnualSalary, fixed=TRUE))
Sal <- Sal[order(Sal$AnnualSalary, decreasing=TRUE), ]
Sal[1:5, c("Name", "AnnualSalary", "JobTitle")]

## Replacing and subbing: `stringr` {.smaller}

dplyr_sal = Sal
dplyr_sal = dplyr_sal %>% mutate(
  AnnualSalary = AnnualSalary %>%
    str_replace(
      fixed("$"),
      "") %>%
    as.numeric) %>%
  arrange(desc(AnnualSalary))
check_Sal = Sal
rownames(check_Sal) = NULL
all.equal(check_Sal, dplyr_sal)

#Part 2

survey.messy <- read.csv("survey_messy.csv", header=TRUE)
survey.messy$TVhours

##What's happening?

str(survey.messy)

tv.hours.messy <- survey.messy$TVhours
tv.hours.messy
as.numeric(tv.hours.messy)

##That didn't work...

tv.hours.messy
as.numeric(tv.hours.messy)

head(tv.hours.messy, 40)
head(as.numeric(tv.hours.messy), 40)

num.vec <- c(3.1, 2.5)
as.factor(num.vec)
as.numeric(as.factor(num.vec))
as.numeric(as.character(as.factor(num.vec)))

as.character(tv.hours.messy)
as.numeric(as.character(tv.hours.messy))
typeof(as.numeric(as.character(tv.hours.messy)))  # Success!! (Almost...)

##A small improvement

tv.hours.strings <- as.character(tv.hours.messy)
tv.hours.strings

tv.hours.strings
```

```
# Use gsub() to replace everything except digits and '.' with a blank ""
gsub("[^0-9.]", "", tv.hours.strings)

tv.hours.messy[1:30]
tv.hours.clean <- as.numeric(gsub("[^0-9.]", "", tv.hours.strings))
tv.hours.clean

##Rebuilding our data

survey <- transform(survey.messy, TVhours = tv.hours.clean)
str(survey)

##A different approach

survey.messy <- read.csv("survey_messy.csv", header=TRUE,
stringsAsFactors=FALSE)
str(survey.messy)

survey <- transform(survey.messy,
                    TVhours = as.numeric(gsub("[^0-9.]", "", TVhours)))
str(survey)

##What about all those other `character` variables?

table(survey[["Program"]])
table(as.factor(survey[["Program"]]))

# Figure out which columns are coded as characters
chr.indexes <- sapply(survey, FUN = is.character)
chr.indexes
# Re-code all of the character columns to factors
survey[chr.indexes] <- lapply(survey[chr.indexes], FUN = as.factor)

str(survey)

##Another common problem

life.support <- as.factor(c("dialysis", "Ventilation", "Dialysis",
"dialysis", "none", "None", "nnone", "dyalysis", "dialysis",
"ventilation", "none"))
summary(life.support)

summary(life.support)

##For loops: a pair of examples

for(i in 1:4) {
  print(i)
}

phrase <- "Good Night, "
for(word in c("and", "Good", "Luck")) {
  phrase <- paste(phrase, word)
  print(phrase)
}
```

```r
##For loops: syntax


##Example

index.set <- list(name="Michael", weight=185, is.male=TRUE) # a list
for(i in index.set) {
  print(c(i, typeof(i)))
}

##Example: Calculate sum of each column

fake.data <- matrix(rnorm(500), ncol=5) # create fake 100 x 5 data set
head(fake.data,2) # print first two rows

col.sums <- numeric(ncol(fake.data)) # variable to store running column
sums
for(i in 1:nrow(fake.data)) {
  col.sums <- col.sums + fake.data[i,] # add ith observation to the sum
}
col.sums

colSums(fake.data) # A better approach (see also colMeans())

##while loops


day <- 1
num.days <- 365
while(day <= num.days) {
  day <- day + 1
}


##Example: apply()

colMeans(fake.data)
apply(fake.data, MARGIN=2, FUN=mean) # MARGIN = 1 for rows, 2 for columns
# Function that calculates proportion of vector indexes that are > 0
propPositive <- function(x) mean(x > 0)
apply(fake.data, MARGIN=2, FUN=propPositive)


##Example: lapply(), sapply()

lapply(survey, is.factor) # Returns a list
sapply(survey, FUN = is.factor) # Returns a vector with named elements


##Example: apply(), lapply(), sapply()

apply(cars, 2, FUN=mean) # Data frames are arrays
lapply(cars, FUN=mean) # Data frames are also lists
sapply(cars, FUN=mean) # sapply() is just simplified lapply()
```

```
##Example: tapply()

library(MASS)
# Get a count table, data broken down by Origin and DriveTrain
table(Cars93$Origin, Cars93$DriveTrain)

# Calculate average MPG.City, broken down by Origin and Drivetrain
tapply(Cars93$MPG.city, INDEX = Cars93[c("Origin", "DriveTrain")],
FUN=mean)


##Example: tapply()

tapply(Cars93[["Horsepower"]], INDEX = Cars93[c("Origin", "Type")],
FUN=mean)

any(Cars93$Origin == "non-USA" & Cars93$Type == "Large")

##Example: using tapply() to mimic table()

library(MASS)
# Get a count table, data broken down by Origin and DriveTrain
table(Cars93$Origin, Cars93$DriveTrain)

# This one may take a moment to figure out...
tapply(rep(1, nrow(Cars93)), INDEX = Cars93[c("Origin", "DriveTrain")],
FUN=sum)

##with()

with(Cars93, table(Origin, Type))

##Example: with()

any(Cars93$Origin == "non-USA" & Cars93$Type == "Large")
with(Cars93, any(Origin == "non-USA" & Type == "Large")) # Same effect!

with(Cars93, tapply(Horsepower, INDEX = list(Origin, Type), FUN=mean))
```