Harvardx Data Science Notes

Friday, January 31, 2020 8:57 AM

Ls() - show all objects in environment
Sqrt() - square root

Functions are evaluated from the inside out log(exp(1)) = 1

args(functionname) - shows you what arguments are needed n the function

Helps ?function()

- seq creates a list of numbers and sum adds them up.
- Names() - shows names of columns
- Length() - number of entries in a vector
- Class() - type of data
- Logical vectors must be either TRUE or FALSE
- Factors are different than characters and used to categorical data, levels() can show the categories in the column, they are stored this way because it is more efficient
- [] can be used to access columns instead off df$clmn, df["clmn"] , single brackets will keep the format as a DF, while [[]] will make the data a vector
- Indentical() - determines if 2 vectors are identical
- Table() returns the frequency of unique elements in a vector
- Names() applies names to vector
- Seq(1,10) consecutive numbers seq(1,10,2) sequence is 1,3,5,7,9
  x <- seq(0, 100, length.out = 5)
  produces the numbers 0, 25, 50, 75, 100.

  SORT, ORDER, and RANK
- Sort() sorts data from small to large
- Order shows you the index of each value in the vector
  index <- order(murders$total)
  murders$abb[index]
  Indexing allows you to sort data but have a different output of the sorted data, for example
  Sort by numbers but have state abbreviation display in the order
- Which.max or which.min is the index of where the lowest or highest value is within the vector
- Rank() ranks each value in the vector

 # Define the variable i to be the index of the smallest state
i <- which.min(murders$population)
 # Define variable states to hold the states
state <- murders$state
 # Use the index you just defined to find the state with the smallest population
state[i]

 # Define a variable ind to store the indexes needed to order the population values
ind <- order(murders$population)
 # Create a data frame my_df with the state name and its rank and ordered from least populous to most
my_df <- data.frame(states[ind], ranks[ind])

 # Use is.na to create a logical index ind that tells which entries are NA

```
ind<- is.na(na_example)
# Determine how many NA ind has using the sum function
sum(ind)

# Compute the average, for entries of na_example that are not NA
mean(na_example[!ind]
```

Basic Syntax / Functions

Saturday, February 1, 2020 9:27 PM

- Names() shows names of variables in a column
- length(unique (x)) - number of unique variables
- quantile(df$column, seq(.10, 0.90, 0.20)) - create a vector showing the 10th, 30th, 50th, 70th, and 90th percentiles
- Stedv - sd()
- Mad - median absolute deviation
- Replace first value of vector vector[1] <- n (n being what you want to replace it with)
- Use tidyverse to load ggplot2
- Reduce the number of significant digits globally by setting an option. For example, options(digits = 3) will cause all future computations that session to have 3 significant digits.
- Reduce the number of digits locally using round() or signif().
- Lapply returns a list
- Apply returns a vector

Indexing, Wrangling, and Plots

Friday, January 31, 2020 8:59 PM

Indexing is useful for vector analysis example ….

```
# Store the murder_rate < 1 in low
low <- murder_rate < 1
# Names of states with murder rates lower than 1
murders$state[low]
```

WHICH, MATCH, and %in%

WHICH

Which entries of a logical vector are TRUE, similar to filter() but can be applied to vectors

```
index <- which(murders$state == "Massachusetts")
```

MATCH

"match" looks for entries in a vector and returns the index needed to access them.

```
index <- match(c("New York", "Florida", "Texas"), murders$state)
```

Creating Dataframe

Friday, January 31, 2020 9:52 PM

DF turns characters into factors, to avoid this use stringsASFactors = FALSE which will keep them as characters

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
                     exam_1 = c(95, 80, 90, 85),
                     exam_2 = c(90, 85, 85, 90),
                     stringsAsFactors = FALSE)
```

Basic Plots

Saturday, February 1, 2020 10:54 AM

Plot() - scatter plot
Hist() - histogram
Boxplot() -
boxplot(rate~region, data = murders)

Conditionals, For Loops, Functions

Saturday, February 1, 2020 4:52 PM

Nchar() number of characters long
factorial(4) returns 4! = 4 × 3 × 2 × 1 = 24

Basic Conditionals (if else or ifelse()
If(condition)
{Expressions}
else{alternative expression}

Number of NA's
Sum(is.na(column)

Any() checks any elements of a vectors
All() checks all elements of a vector
function
FUNCTION
Creating a new function

```
Nameoffunction<- function(x){
  s <- sum(x)
  n <- length(x)
  s/n
}
```

```
# functions can have multiple arguments as well as default values
avg <- function(x, arithmetic = TRUE){
  n <- length(x)
  ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))
}
```

FOR LOOPS

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
```

compute_s_n(3) returns 6
compute_s_n(100) returns 5050

What if I wanted n to vary?
For(I in range of values)
{operations that use I, which is changing across the range of values}

```
# a very simple for-loop
for(i in 1:5){
  print(i)
```

```
# a for-loop for our summation
m <- 25
s_n <- vector(length = m) # create an empty vector
for(n in 1:m){
```

```
  s_n[n] <- compute_s_n(n)
}
```

 # creating a plot for our summation function

```
n <- 1:m
plot(n, s_n)
```

 # a table of values comparing our function to the summation formula

```
head(data.frame(s_n = s_n, formula = n*(n+1)/2))
```

 # overlaying our function with the summation formula

```
plot(n, s_n)
lines(n, n*(n+1)/2)
```

Other Functions to learn

Apply
Sapply
Tapply
Mapply
Spli
Cut
Quantile
Reduce
unique

Visualization Lesson

Saturday, February 1, 2020 9:33 PM

Cumulative distribution function CDF

make a table of category proportions
prop.table(table(heights$sex)) - shows percentage of each variable

Manual CDF

```
a <- seq(min(my_data), max(my_data), length = 100)    # define range of values spanning the dataset
cdf_function <- function(x) {    # computes prob. for a single value
   mean(my_data <= x)
}
cdf_values <- sapply(a, cdf_function)
plot(a, cdf_values)
```

- Smooth density plots are smooth histograms with very many small bins
- 95% are between 2 standard deviations from the mean

Normal Distribution
Code saved in R

- 68% of observations will be 1 Stdev from the mean
- 95% of observations will be 2 Stdev from the mean
- 99.7% of observations will be 3 Stdev from the mean

Code: Using pnorm to calculate probabilities
pnorm(a, avg, s)

Given male heights x:

```
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

We can estimate the probability that a male is taller than 70.5 inches with:

```
1 - pnorm(70.5, mean(x), sd(x))
```

1 - pnorm(7*12, 69, 3)

Code: Discretization and the normal approximation

```
# plot distribution of exact heights in data
plot(prop.table(table(x)), xlab = "a = Height in inches", ylab = "Pr(x = a)")
```

 # Determine the proportion of data between two numbers

```
# probabilities in actual data over length 1 ranges containing an integer
mean(x <= 81) - mean(x <= 79)  or  mean(x > 79 & x <= 81)
```

Normal approximation

```
pnorm(68.5, mean(x), sd(x)) - pnorm(67.5, mean(x), sd(x))
```

probabilities in normal approximation match well

```
pnorm(68.5, mean(x), sd(x)) - pnorm(67.5, mean(x), sd(x))
pnorm(69.5, mean(x), sd(x)) - pnorm(68.5, mean(x), sd(x))
pnorm(70.5, mean(x), sd(x)) - pnorm(69.5, mean(x), sd(x))

# probabilities in actual data over other ranges don't match normal approx as well
mean(x <= 70.9) - mean(x <= 70.1)
```

As seen in exercise 3, the normal approximation tends to underestimate the extreme values. It's possible that there are more seven footers than we predicted.

Quantiles

Sunday, February 2, 2020 4:48 PM

Definition of quantiles

Quantiles are cutoff points that divide a dataset into intervals with set probabilities. The Qth quantile is the value at which Q% of the observations are equal to or less than that value.

Using the quantile function

Given a dataset data and desired quantile q, you can find the qth quantile of data with:

```
quantile(data,q)
```

Percentiles

Percentiles are the quantiles that divide a dataset into 100 intervals each with 1% probability. You can determine all percentiles of a dataset data like this:

```
p <- seq(0.01, 0.99, 0.01)
quantile(data, p)
```

Quartiles

Quartiles divide a dataset into 4 parts each with 25% probability. They are equal to the 25th, 50th and 75th percentiles. The 25th percentile is also known as the *1st quartile*, the 50th percentile is also known as the *median*, and the 75th percentile is also known as the *3rd quartile*.

The summary() function returns the minimum, quartiles and maximum of a vector.

EXAMPLE

```
library(dslabs)
data(heights)
```

Use summary() on the heights$height variable to find the quartiles:

```
summary(heights$height)
```

Find the percentiles of heights$height:

```
p <- seq(0.01, 0.99, 0.01)
percentiles <- quantile(heights$height, p)
```

Confirm that the 25th and 75th percentiles match the 1st and 3rd quartiles. Note that quantile() returns a named vector. You can access the 25th and 75th percentiles like this (adapt the code for other percentile values):

```
percentiles[names(percentiles) == "25%"]
percentiles[names(percentiles) == "75%"]
```

Definition of qnorm

The qnorm() function gives the theoretical value of a quantile with probability p of observing a value equal to or less than that quantile value given a normal distribution with mean mu and standard deviation sigma:

```
qnorm(p, mu, sigma)
```

By default, mu=0 and sigma=1. Therefore, calling qnorm() with no arguments gives quantiles for the standard normal distribution.

```
qnorm(p)
```

Recall that quantiles are defined such that p is the probability of a random observation less than or equal to the quantile.

Relation to pnorm
The pnorm() function gives the probability that a value from a standard normal distribution will be less than or equal to a z-score value z. Consider:

pnorm(-1.96) $\approx$0.025

The result of pnorm() is the quantile. Note that:

qnorm(0.025) ≈ -1.96

qnorm() and pnorm() are inverse functions:

pnorm(qnorm(0.025)) =0.025

Theoretical quantiles
You can use qnorm() to determine the theoretical quantiles of a dataset: that is, the theoretical value of quantiles assuming that a dataset follows a normal distribution. Run the qnorm() function with the desired probabilities p, mean mu and standard deviation sigma.

Suppose male heights follow a normal distribution with a mean of 69 inches and standard deviation of 3 inches. The theoretical quantiles are:

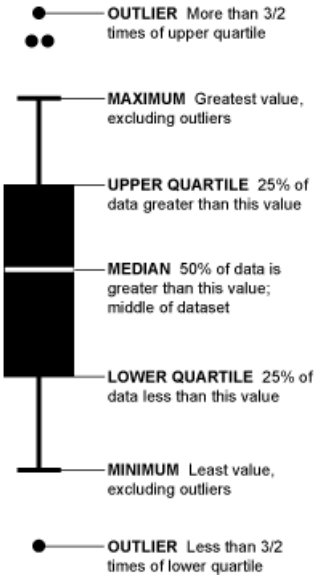
```
p <- seq(0.01, 0.99, 0.01)
theoretical_quantiles <- qnorm(p, 69, 3)
```
Theoretical quantiles can be compared to sample quantiles determined with the quantile function in order to evaluate whether the sample follows a normal distribution.

Quantile-Quantile Plots
```
# define x and z
library(tidyverse)
library(dslabs)
data(heights)
index <- heights$sex=="Male"
x <- heights$height[index]
z <- scale(x)

# proportion of data below 69.5
mean(x <= 69.5)

# calculate observed and theoretical quantiles
p <- seq(0.05, 0.95, 0.05)
observed_quantiles <- quantile(x, p)
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))

# make QQ-plot
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)

# make QQ-plot with scaled values
observed_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
plot(theoretical_quantiles, observed_quantiles)
```

```
abline(0,1)
```

Box and wisker plot

Sunday, February 2, 2020 6:04 PM



Ggplot basics

Friday, February 7, 2020 4:41 PM

Aes is aesthetic mapping

```
library(tidyverse)
library(dslabs)
data(murders)

ggplot(data = murders)

murders %>% ggplot()

p <- ggplot(data = murders)
class(p)
print(p)    # this is equivalent to simply typing p
p

library(tidyverse)
library(dslabs)
data(murders)

murders %>% ggplot() +
  geom_point(aes(x = population/10^6, y = total))

# add points layer to predefined ggplot object
p <- ggplot(data = murders)
p + geom_point(aes(population/10^6, total))

# add text layer to scatterplot
p + geom_point(aes(population/10^6, total)) +
  geom_text(aes(population/10^6, total, label = abb))

# no error from this call
p_test <- p + geom_text(aes(population/10^6, total, label = abb))

# error - "abb" is not a globally defined variable and cannot be found outside of aes
p_test <- p + geom_text(aes(population/10^6, total), label = abb)

# change the size of the points
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))

# move text labels slightly to the right
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1)

# simplify code by adding global aesthetic
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))
p + geom_point(size = 3) +
  geom_text(nudge_x = 1.5)
```

```
# local aesthetics override global aesthetics
p + geom_point(size = 3) +
   geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

Ggplot scales, labels and colors

Friday, February 7, 2020 4:58 PM

Code: Log-scale the x- and y-axis

```
# define p
library(tidyverse)
library(dslabs)
data(murders)
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))

# log base 10 scale the x-axis and y-axis
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")

# efficient log scaling of the axes
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10()
```

Code: Add labels and title

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Population in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")
```

Code: Change color of the points

```
# redefine p to be everything except the points layer
p <- murders %>%
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Population in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010")

# make all points blue
p + geom_point(size = 3, color = "blue")
```

Make sure to put color in quotes

```
# color points by region
p + geom_point(aes(col = region), size = 3)
```

Code: Add a line with average murder rate

```
# define average murder rate
r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  pull(rate)
```

 # basic line with average murder rate for the country

```
p + geom_point(aes(col = region), size = 3) +
   geom_abline(intercept = log10(r))    # slope is default of 1
```

 # change line to dashed and dark grey, line under points

```
p +
   geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
   geom_point(aes(col = region), size = 3)
```

Code: Change legend title

```
p <- p + scale_color_discrete(name = "Region")    # capitalize legend title
```

Change color of labels

```
murders %>% ggplot(aes(population, total, label= abb)) +
  geom_label()
```

Multiple colors by region

```
murders %>% ggplot(aes(population, total, label = abb, color = region)) +
  geom_label()
```

Rescale plot using log

```
p + scale_x_log10() + scale_y_log10()
```

Ggplot add on packages

Friday, February 7, 2020 5:05 PM

Code: Adding themes

```
# theme used for graphs in the textbook and course
library(dslabs)
ds_theme_set()
# themes from ggthemes
library(ggthemes)
p + theme_economist()  # style of the Economist magazine
p + theme_fivethirtyeight()  # style of the FiveThirtyEight website
```

Code: Putting it all together to assemble the plot

```
# load libraries
library(tidyverse)
library(ggrepel)
library(ggthemes)
library(dslabs)
data(murders)
# define the intercept
r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  .$rate

# make the plot, combining all elements
murders %>%
  ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col = region), size = 3) +
  geom_text_repel() +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Population in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in 2010") +
  scale_color_discrete(name = "Region") +
  theme_economist()
```

Ggplot Other plots

Friday, February 7, 2020 5:08 PM

Code: Histograms in ggplot2

```
# load heights data
library(tidyverse)
library(dslabs)
data(heights)
# define p
p <- heights %>%
  filter(sex == "Male") %>%
  ggplot(aes(x = height))

# basic histograms
p + geom_histogram()
p + geom_histogram(binwidth = 1)
# histogram with blue fill, black outline, labels and title
p + geom_histogram(binwidth = 1, fill = "blue", col = "black") +
  xlab("Male heights in inches") +
  ggtitle("Histogram")
```

Code: Smooth density plots in ggplot2

```
p + geom_density()
p + geom_density(fill = "blue")
```

Code: Quantile-quantile plots in ggplot2

```
# basic QQ-plot
p <- heights %>% filter(sex == "Male") %>%
  ggplot(aes(sample = height))
p + geom_qq()
# QQ-plot against a normal distribution with same mean/sd as data
params <- heights %>%
  filter(sex == "Male") %>%
  summarize(mean = mean(height), sd = sd(height))
  p + geom_qq(dparams = params) +
  geom_abline()

# QQ-plot of scaled data against the standard normal distribution
heights %>%
  ggplot(aes(sample = scale(height)) +
  geom_qq() +
  geom_abline()
```

Code: Grids of plots with the gridExtra package (shows charts next to each other)

```
# define plots p1, p2, p3
p <- heights %>% filter(sex == "Male") %>% ggplot(aes(x = height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue", col = "black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue", col = "black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue", col = "black")
# arrange plots next to each other in 1 row, 3 columns
library(gridExtra)
grid.arrange(p1, p2, p3, ncol = 3)
```

Change bin width of histogram

```
geom_histogram(binwidth =1)
```

Overlapping density plots

```
heights %>%
  ggplot(aes(height, group = sex, color = sex)) +geom_density()
```

Dplyr

Friday, February 7, 2020 9:15 PM

Dot placeholder

```
murders <- murders %>% mutate(murder_rate = total/population*100000)
summarize(murders, mean(murder_rate))

# calculate US murder rate, generating a data frame
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population) * 100000)
  us_murder_rate

# extract the numeric US murder rate with the dot operator
us_murder_rate %>% .$rate

# calculate and extract the murder rate with one pipe
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population * 100000)) %>%
  .$rate
```

SORTING in DPLYR using arrange()

```
# arrange by population column, smallest to largest
murders %>% arrange(population) %>% head()

# arrange by murder rate, smallest to largest
murders %>% arrange(murder_rate) %>% head()

# arrange by murder rate in descending order
murders %>% arrange(desc(murder_rate)) %>% head()

# arrange by region alphabetically, then by murder rate within each region
murders %>% arrange(region, murder_rate) %>% head()

# show the top 10 states with highest murder rate, not ordered by rate
murders %>% top_n(10, murder_rate)

# show the top 10 states with highest murder rate, ordered by rate
murders %>% arrange(desc(murder_rate)) %>% top_n(10)
```

ggplot Faceting

Saturday, February 8, 2020 1:07 PM

Facet paired with ggplot creates subsets of charts by column and by row
Facet keeps the scales of charts consistent
facet_grid(row~ column)

facet_wrap

```
# facet by continent and year
filter(gapminder, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(continent ~ year)

# facet by year only
filter(gapminder, year %in% c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(. ~ year)

# facet by year, plots wrapped onto multiple rows
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
  filter(year %in% years & continent %in% continents) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_wrap(~year)
```

Gpplot timeseries with geom_line

Saturday, February 8, 2020 1:15 PM

Code: Single time series

```
# scatterplot of US fertility by year
gapminder %>%
  filter(country == "United States") %>%
  ggplot(aes(year, fertility)) +
  geom_point()
# line plot of US fertility by year
gapminder %>%
  filter(country == "United States") %>%
  ggplot(aes(year, fertility)) +
  geom_line()
```

Code: Multiple time series

```
# line plot fertility time series for two countries- only one line (incorrect)
countries <- c("South Korea", "Germany")
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility)) +
  geom_line()

# line plot fertility time series for two countries - one line per country
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, group = country)) +
  geom_line()
# fertility time series for two countries - lines colored by country
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, fertility, col = country)) +
  geom_line()
```

Code: Adding text labels to a plot

```
# life expectancy time series - lines colored by country and labeled, no legend
labels <- data.frame(country = countries, x = c(1975, 1965), y = c(60, 72))
gapminder %>% filter(country %in% countries) %>%
  ggplot(aes(year, life_expectancy, col = country)) +
  geom_line() +
  geom_text(data = labels, aes(x, y, label = country), size = 5) +
  theme(legend.position = "none")
```

Ggplot additional (transform, stratify & boxplot)

Saturday, February 8, 2020 1:22 PM

```
# repeat histogram with log2 scaled x-axis
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2")



# boxplot of GDP by region in 1970
past_year <- 1970
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(region, dollars_per_day))
p + geom_boxplot()

# rotate names on x-axis
p + geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

Code: The reorder function (to reorder categories by value)

```
# by default, factor order is alphabetical
fac <- factor(c("Asia", "Asia", "West", "West", "West"))
levels(fac)

# reorder factor by the category means
value <- c(10, 11, 12, 6, 4)
fac <- reorder(fac, value, FUN = mean)
levels(fac)



# reorder by median income and color by continent
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%    # reorder
  ggplot(aes(region, dollars_per_day, fill = continent)) +    # color by continent
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  xlab("")
p

# log2 scale y-axis
p + scale_y_continuous(trans = "log2")

# add data points
p + scale_y_continuous(trans = "log2") + geom_point(show.legend = FALSE)
```

Boxplot advanced feature

```
 # arrange matching boxplots next to each other, colored by year
 p + geom_boxplot(aes(region, dollars_per_day, fill = factor(year)))
```

```
library(dplyr)
library(ggplot2)
library(dslabs)
dat <- us_contagious_diseases %>% filter(year == 1967 & disease=="Measles" & count>0 & !
is.na(population)) %>%
 mutate(rate = count / population * 10000 * 52 / weeks_reporting) %>%
 mutate(state = reorder(state, rate))
dat %>% ggplot(aes(state, rate)) +
 geom_bar(stat="identity") +
 coord_flip()
```

Density Plots

Saturday, February 8, 2020 1:41 PM

Key points

- Change the y-axis of density plots to variable counts using ..count.. as the y argument.
The case_when() function defines a factor whose levels	are defined by a variety of logical operations to group data.
- Plot stacked density plots using position="stack".
- Define a weight aesthetic mapping to change the relative weights of density plots - for example, this allows weighting of plots by population rather than number of countries.

Code: Faceted smooth density plots

```
# see the code below the previous video for variable definitions
# smooth density plots - area under each curve adds to 1
gapminder %>%
filter(year == past_year & country %in% country_list) %>%
mutate(group = ifelse(region %in% west, "West", "Developing")) %>% group_by(group) %>%
summarize(n = n()) %>% knitr::kable()
# smooth density plots - variable counts on y-axis
p <- gapminder %>%
filter(year == past_year & country %in% country_list) %>%
mutate(group = ifelse(region %in% west, "West", "Developing")) %>%
ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
scale_x_continuous(trans = "log2")
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```

Code: Add new region groups with case_when

```
# add group as a factor, grouping regions
gapminder <- gapminder %>%
mutate(group = case_when(
.$region %in% west ~ "West",
.$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
.$region %in% c("Caribbean", "Central America", "South America") ~ "Latin America",
.$continent == "Africa" & .$region != "Northern Africa" ~ "Sub-Saharan Africa",
TRUE ~ "Others"))
# reorder factor levels
gapminder <- gapminder %>%
mutate(group = factor(group, levels = c("Others", "Latin America", "East Asia", "Sub-Saharan Africa", "West")))
```

Code: Stacked density plot

```
# note you must redefine p with the new gapminder object first
p <- gapminder %>%
filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
ggplot(aes(dollars_per_day, fill = group)) +
scale_x_continuous(trans = "log2")
# stacked density plot
p + geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
facet_grid(year ~ .)
```

Code: Weighted stacked density plot

```
# weighted stacked density plot
gapminder %>%
filter(year %in% c(past_year, present_year) & country %in% country_list) %>%
group_by(year) %>%
mutate(weight = population/sum(population*2)) %>%
ungroup() %>%
```

```
ggplot(aes(dollars_per_day, fill = group, weight = weight)) +
scale_x_continuous(trans = "log2") +
geom_density(alpha = 0.2, bw = 0.75, position = "stack") + facet_grid(year ~ .)
```

Ecological Fallacy & geom_jitter

Saturday, February 8, 2020 1:43 PM

Key points

- The breaks argument allows us to set the location of the axis labels and tick marks.
- The *logistic* or *logit transformation* is defined as $f(p)=\log p1-p$, or the log of odds. This scale is useful for highlighting differences near 0 or near 1 and converts fold changes into constant increases.
- The *ecological fallacy* is assuming that conclusions made from the average of a group apply to all members of that group

```
# define gapminder library(tidyverse) library(dslabs) data(gapminder)
# add additional cases gapminder <- gapminder %>%
mutate(group = case_when(
.$region %in% west ~ "The West",
.$region %in% "Northern Africa" ~ "Northern Africa",
.$region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
.$region == "Southern Asia" ~ "Southern Asia",
.$region %in% c("Central America", "South America", "Caribbean") ~ "Latin America",
.$continent == "Africa" & .$region != "Northern Africa" ~ "Sub-Saharan Africa",
.$region %in% c("Melanesia", "Micronesia", "Polynesia") ~ "Pacific Islands")) # define a data frame with
group average income and average infant survival rate surv_income <- gapminder %>%
filter(year %in% present_year & !is.na(gdp) & !is.na(infant_mortality) & !is.na(group)) %>%
group_by(group) %>%
summarize(income = sum(gdp)/sum(population)/365,
infant_survival_rate = 1 - sum(infant_mortality/1000*population)/sum(population))
surv_income %>% arrange(income) # plot infant survival versus income, with transformed axes
surv_income %>% ggplot(aes(income, infant_survival_rate, label = group, color = group)) +
scale_x_continuous(trans = "log2", limit = c(0.25, 150)) +
scale_y_continuous(trans = "logit", limit = c(0.875, .9981),
breaks = c(.85, .90, .95, .99, .995, .998)) +
geom_label(size = 3, show.legend = FALSE)
```

Key points

- A dynamite plot - a bar graph of group averages with error bars denoting standard errors - provides almost no information about a distribution.
- By showing the data, you provide viewers extra information about distributions.
- Jitter is adding a small random shift to each point in order to minimize the number of overlapping points. To add jitter, use the geom_jitter() geometry instead of geom_point(). (See example below.)
- Alpha blending is making points somewhat transparent, helping visualize the density of overlapping points. Add an alpha argument to the geometry.

```
# dot plot showing the data
heights %>% ggplot(aes(sex, height)) + geom_point()
# jittered, alpha blended point plot
heights %>% ggplot(aes(sex, height)) + geom_jitter(width = 0.1, alpha = 0.2)
```

Slope Charts

Saturday, February 8, 2020 7:52 PM

Key points

- Consider using a slope chart or Bland-Altman plot when comparing one variable at two different time points, especially for a small number of observations.
- Slope charts use angle to encode change. Use geom_line() to create slope charts. It is useful when comparing a small number of observations.
- The Bland-Altman plot (Tukey mean difference plot, MA plot) graphs the difference between conditions on the y-axis and the mean between conditions on the x-axis. It is more appropriate for large numbers of observations than slope charts.

Code: Slope chart

```
library(tidyverse)
library(dslabs)
data(gapminder)
west <- c("Western Europe", "Northern Europe", "Southern Europe", "Northern America", "Australia and New Zealand")
dat <- gapminder %>%
filter(year %in% c(2010, 2015) & region %in% west & !is.na(life_expectancy) & population > 10^7)
dat %>%
mutate(location = ifelse(year == 2010, 1, 2),
location = ifelse(year == 2015 & country %in% c("United Kingdom", "Portugal"),
location + 0.22, location),
hjust = ifelse(year == 2010, 1, 0)) %>%
mutate(year = as.factor(year)) %>%
ggplot(aes(year, life_expectancy, group = country)) +
geom_line(aes(color = country), show.legend = FALSE) +
geom_text(aes(x = location, label = country, hjust = hjust), show.legend = FALSE) +
xlab("") +
ylab("Life Expectancy")
```

Code: Bland-Altman plot

```
library(ggrepel)
dat %>%
mutate(year = paste0("life_expectancy_", year)) %>%
select(country, year, life_expectancy) %>% spread(year, life_expectancy) %>%
mutate(average = (life_expectancy_2015 + life_expectancy_2010)/2,
difference = life_expectancy_2015 - life_expectancy_2010) %>%
ggplot(aes(average, difference, label = country)) +
geom_point() +
geom_text_repel() +
geom_abline(lty = 2) +
xlab("Average of 2010 and 2015") +
ylab("Difference between 2015 and 2010")
```

Tile Plots

Saturday, February 8, 2020 7:57 PM

Key points

- Vaccines save millions of lives, but misinformation has led some to question the safety of vaccines. The data support vaccines as safe and effective. We visualize data about measles incidence in order to demonstrate the impact of vaccination programs on disease rate.
- The **RColorBrewer** package offers several color palettes. Sequential color palettes are best suited for data that span from high to low. Diverging color palettes are best suited for data that are centered and diverge towards high or low values.
- The geom_tile() geometry creates a grid of colored tiles.
- Position and length are stronger cues than color for numeric values, but color can be appropriate sometimes.

Code: Tile plot of measles rate by year and state

```
# import data and inspect
library(tidyverse)
library(dslabs)
data(us_contagious_diseases)
str(us_contagious_diseases)
# assign dat to the per 10,000 rate of measles, removing Alaska and Hawaii and adjusting for weeks reporting
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
filter(!state %in% c("Hawaii", "Alaska") & disease == the_disease) %>%
mutate(rate = count / population * 10000 * 52/weeks_reporting) %>%
mutate(state = reorder(state, rate))
# plot disease rates per year in California
dat %>% filter(state == "California" & !is.na(rate)) %>%
ggplot(aes(year, rate)) +
geom_line() +
ylab("Cases per 10,000") +
geom_vline(xintercept=1963, col = "blue")
# tile plot of disease rate by state and year
dat %>% ggplot(aes(year, state, fill=rate)) +
geom_tile(color = "grey50") +
scale_x_continuous(expand = c(0,0)) +
scale_fill_gradientn(colors = RColorBrewer::brewer.pal(9, "Reds"), trans = "sqrt") +
geom_vline(xintercept = 1963, col = "blue") +
theme_minimal() + theme(panel.grid = element_blank()) +
ggtitle(the_disease) +
ylab("") +
xlab("")
```

Code: Line plot of measles rate by year and state

```
# compute US average measles rate by year
avg <- us_contagious_diseases %>%
filter(disease == the_disease) %>% group_by(year) %>%
summarize(us_rate = sum(count, na.rm = TRUE)/sum(population, na.rm = TRUE)*10000)
# make line plot of measles rate by year by state
dat %>%
filter(!is.na(rate)) %>%
ggplot() +
geom_line(aes(year, rate, group = state), color = "grey50",
show.legend = FALSE, alpha = 0.2, size = 1) +
geom_line(mapping = aes(year, us_rate), data = avg, size = 1, col = "black") +
```

```
scale_y_continuous(trans = "sqrt", breaks = c(5, 25, 125, 300)) +
ggtitle("Cases per 10,000 by state") +
xlab("") +
ylab("") +
geom_text(data = data.frame(x = 1955, y = 50),
mapping = aes(x, y, label = "US average"), color = "black") +
geom_vline(xintercept = 1963, col = "blue")

>
```

Monte Carlo Simulations

Sunday, February 9, 2020 2:42 PM

```
beads <- rep(c("red", "blue"), times = c(2,3))    # create an urn with 2 red, 3 blue
beads    # view beads object
sample(beads, 1)    # sample 1 bead at random

B <- 10000    # number of times to draw 1 bead
events <- replicate(B, sample(beads, 1))    # draw 1 bead, B times
tab <- table(events)    # make a table of outcome counts
tab    # view count table
prop.table(tab)    # view table of outcome proportions
```

To place bead back

```
events <- sample(beads, B, replace = TRUE)
prop.table(table(events))
```

The set.seed() function

Before we continue, we will briefly explain the following important line of code:

```
set.seed(1986)
```

If you are running R 3.6, you can revert to the original seed setting behavior by adding the argument `sample.kind="Rounding"`. For example:

```
set.seed(1)
set.seed(1, sample.kind="Rounding")
```

An important application of the mean() function

In R, applying the mean() function to a logical vector returns the proportion of elements that are TRUE. It is very common to use the mean() function in this way to calculate probabilities and we will do so throughout the course.

Suppose you have the vector beads from a previous video:

```
beads <- rep(c("red", "blue"), times = c(2,3))
beads
[1] "red" "red" "blue" "blue" "blue"
```

To find the probability of drawing a blue bead at random, you can run:

```
mean(beads == "blue")
[1] 0.6
```

This code is broken down into steps inside R. First, R evaluates the logical statement beads == "blue", which generates the vector:

FALSE FALSE TRUE TRUE TRUE

When the mean function is applied, R coerces the logical values to numeric values, changing TRUE to 1 and FALSE to 0:

0 0 1 1 1

The mean of the zeros and ones thus gives the proportion of TRUE values. As we have learned and will continue to see, probabilities are directly related to the proportion of events that satisfy a requirement.

HOW MANY MONTE CARLO ESTIMATES ARE ENOUGH?

Key points

- The larger the number of Monte Carlo replicates B, the more accurate the estimate.
- Determining the appropriate size for B can require advanced statistics.
- One practical approach is to try many sizes for B and look for sizes that provide stable estimates.

Code: Estimating a practical value of B

This code runs Monte Carlo simulations to estimate the probability of shared birthdays using several `B` values and plots the results. When `B` is large enough that the estimated probability stays stable, then we have selected a useful value of `B`.

```
B <- 10^seq(1, 5, len = 100)    # defines vector of many B values
compute_prob <- function(B, n = 22){    # function to run Monte Carlo simulation with each B
          same_day <- replicate(B, {
          bdays <- sample(1:365, n, replace = TRUE)
     any(duplicated(bdays))
   })
   mean(same_day)
}
prob <- sapply(B, compute_prob)    # apply compute_prob to many values of B
plot(log10(B), prob, type = "l")    # plot a line graph of estimates
```

Example:

```
# This line of example code simulates four independent random games where the Celtics either lose or win. Copy this example code to use within the `replicate` function.
simulated_games <- sample(c("lose","win"), 4, replace = TRUE, prob = c(0.6, 0.4))

# The variable 'B' specifies the number of times we want the simulation to run. Let's run the Monte Carlo simulation 10,000 times.
B <- 10000

# Use the `set.seed` function to make sure your answer matches the expected result after random sampling.
set.seed(1)

# Create an object called `celtic_wins` that replicates two steps for B iterations: (1) generating a random four-game series `simulated_games` using the example code, then (2) determining whether the simulated series contains at least one win for the Celtics.
    celtic_wins <- replicate(B, {
    simulated_games <- sample(c("lose","win"), 4, replace = TRUE, prob = c(0.6, 0.4))
    any(simulated_games=="win") })

# Calculate the frequency out of B iterations that the Celtics won at least one game. Print your answer to the console.
mean(celtic_wins)
```

Equations for probabilities

Sunday, February 9, 2020 2:56 PM

Key points

- *Conditional probabilities* compute the probability that an event occurs given information about dependent events. For example, the probability of drawing a second king given that the first draw is a king is:
  $$\Pr(\text{Card 2 is a king}|\text{Card 1 is a king})=3/51$$
- If two events A and B are independent, $\Pr(A|B)=\Pr(A)$.
- To determine the probability of multiple events occurring, we use the *multiplication rule*.

Equations

The multiplication rule for independent events is:

$$\Pr(A \text{ and } B \text{ and } C)=\Pr(A)\times\Pr(B)\times\Pr(C)$$

The multiplication rule for dependent events considers the conditional probability of both events occurring:

$$\Pr(A \text{ and } B)=\Pr(A)\times\Pr($$

We can expand the multiplication rule for dependent events to more than 2 events:

$$\Pr(A \text{ and } B \text{ and } C)=\Pr(A)\times\Pr(B|A)\times\Pr(C|A \text{ and } B)$$

The Addition Rule

By "facecard", the professor means a card with a value of 10 (K, Q, J, 10).

Key points

- The addition rule states that the probability of event A or event B happening is the probability of event A plus the probability of event B minus the probability of both events A and B happening together.
  $$\Pr(A \text{ or } B)=\Pr(A)+\Pr(B)-\Pr(A \text{ and } B)$$
- Note that $(A \text{ or } B)$ is equivalent to $(A|B)$.

Example: The addition rule for a natural 21 in blackjack

We apply the addition rule where A = drawing an ace then a facecard and B = drawing a facecard then an ace. Note that in this case, both events A and B cannot happen at the same time, so $\Pr(A \text{ and } B)=0$.

$$\Pr(\text{ace then facecard})=\frac{4}{52}\times\frac{16}{51}$$
$$\Pr(\text{facecard then ace})=\frac{16}{52}\times\frac{4}{51}$$
$$\Pr(\text{ace then facecard} \mid \text{facecard then ace})=\frac{4}{52}\times\frac{16}{51}+\frac{16}{52}\times\frac{4}{51}=0.0483$$

Combinations and Permutations

Friday, February 14, 2020 6:49 PM

Code: Introducing paste() and expand.grid()

```
# joining strings with paste
number <- "Three"
suit <- "Hearts"
paste(number, suit)
# joining vectors element-wise with paste
paste(letters[1:5], as.character(1:5))
```

Creates all possible combinations

```
# generating combinations of 2 vectors with expand.grid
expand.grid(pants = c("blue", "black"), shirt = c("white", "grey", "plaid"))
```

Code: Generating a deck of cards

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen",
"King")
deck <- expand.grid(number = numbers, suit = suits)
deck <- paste(deck$number, deck$suit)
# probability of drawing a king
kings <- paste("King", suits)
mean(deck %in% kings)
```

Code: Permutations and combinations

Correction: The code shown does not generate all 7 digit phone numbers because phone numbers can have repeated digits. It generates all possible 7 digit numbers without repeats.

```
library(gtools)
permutations(5,2)    # ways to choose 2 numbers in order from 1:5
all_phone_numbers <- permutations(10, 7, v = 0:9)
n <- nrow(all_phone_numbers)
index <- sample(n, 5)
all_phone_numbers[index,]
permutations(3,2)    # order matters
combinations(3,2)    # order does not matter
```

Code: Probability of drawing a second king given that one king is drawn

```
hands <- permutations(52,2, v = deck)
first_card <- hands[,1]
second_card <- hands[,2]
sum(first_card %in% kings)
sum(first_card %in% kings & second_card %in% kings) / sum(first_card %in% kings)
```

Code: Probability of a natural 21 in blackjack

```
aces <- paste("Ace", suits)
facecard <- c("King", "Queen", "Jack", "Ten")
facecard <- expand.grid(number = facecard, suit = suits)
facecard <- paste(facecard$number, facecard$suit)
hands <- combinations(52, 2, v=deck) # all possible hands
# probability of a natural 21 given that the ace is listed first in `combinations`
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
# probability of a natural 21 checking for both ace first and ace second
mean((hands[,1] %in% aces & hands[,2] %in% facecard)|(hands[,2] %in% aces & hands[,1] %in% facecard))
```

Code: Monte Carlo simulation of natural 21 in blackjack

Note that your exact values will differ because the process is random and the seed is not set.

```
# code for one hand of blackjack
hand <- sample(deck, 2)
hand
```

```
# code for B=10,000 hands of blackjack
B <- 10000
results <- 0
```

Probability Birthday Problem

Friday, February 14, 2020 6:58 PM

Key points

- duplicated() takes a vector and returns a vector of the same length with TRUE for any elements that have appeared previously in that vector.
- We can compute the probability of shared birthdays in a group of people by modeling birthdays as random draws from the numbers 1 through 365. We can then use this sampling model of birthdays to run a Monte Carlo simulation to estimate the probability of shared birthdays.

Code: The birthday problem

```
# checking for duplicated bdays in one 50 person group
n <- 50
bdays <- sample(1:365, n, replace = TRUE)   # generate n random birthdays
any(duplicated(bdays))   # check if any birthdays are duplicated
# Monte Carlo simulation with B=10000 replicates
B <- 10000
results <- replicate(B, {   # returns vector of B logical values
  bdays <- sample(1:365, n, replace = TRUE)
  any(duplicated(bdays))
})
mean(results)   # calculates proportion of groups with duplicated bdays
```

SAPPLY()

Key points

- Some functions automatically apply element-wise to vectors, such as sqrt() and *.
- However, other functions do not operate element-wise by default. This includes functions we define ourselves.
- The function sapply(x, f) allows any other function f to be applied element-wise to the vector x.
- The probability of an event happening is 1 minus the probability of that event not happening:

$$\Pr(\text{event}) = 1 - \Pr(\text{no event})$$

- We can compute the probability of shared birthdays mathematically:

$$\Pr(\text{shared birthdays}) = 1 - \Pr(\text{no shared birthdays}) = 1 - (1 \times 364/365 \times 363/365 \times \ldots \times 365 - n + 1365)$$

Code: Function for birthday problem Monte Carlo simulations

Note that the function body of compute_prob() is the code that we wrote in the previous video. If we write this code as a function, we can use sapply() to apply this function to several values of n.

```
# function to calculate probability of shared bdays across n people
compute_prob <- function(n, B = 10000) {
          same_day <- replicate(B, {
          bdays <- sample(1:365, n, replace = TRUE)
    any(duplicated(bdays))
  })
  mean(same_day)
}
n <- seq(1, 60)
```

Code: Element-wise operation over vectors and sapply

```
x <- 1:10
sqrt(x)   # sqrt operates on each element of the vector
y <- 1:10
```

```r
x*y   # * operates element-wise on both vectors
compute_prob(n)   # does not iterate over the vector n without sapply
x <- 1:10
sapply(x, sqrt)   # this is equivalent to sqrt(x)
prob <- sapply(n, compute_prob)   # element-wise application of compute_prob to n
plot(n, prob)
```

Code: Computing birthday problem probabilities with sapply

```r
# function for computing exact probability of shared birthdays for any n
exact_prob <- function(n){
  prob_unique <- seq(365, 365-n+1)/365   # vector of fractions for mult. rule
  1 - prod(prob_unique)   # calculate prob of no shared birthdays and subtract from 1
}
# applying function element-wise to vector of n values
eprob <- sapply(n, exact_prob)
# plotting Monte Carlo results and exact probabilities on same graph
plot(n, prob)   # plot Monte Carlo results
lines(n, eprob, col = "red")   # add line for exact prob
```

Monty Hall Problem (Discrete Probability)

Friday, February 14, 2020 7:47 PM

Key points

- Monte Carlo simulations can be used to simulate random outcomes, which makes them useful when exploring ambiguous or less intuitive problems like the Monty Hall problem.
- In the Monty Hall problem, contestants choose one of three doors that may contain a prize. Then, one of the doors that was not chosen by the contestant and does not contain a prize is revealed. The contestant can then choose whether to stick with the original choice or switch to the remaining unopened door.
- Although it may seem intuitively like the contestant has a 1 in 2 chance of winning regardless of whether they stick or switch, Monte Carlo simulations demonstrate that the actual probability of winning is 1 in 3 with the stick strategy and 2 in 3 with the switch strategy.
- For more on the Monty Hall problem, you can watch a detailed explanation here or read an explanation here.

Code: Monte Carlo simulation of stick strategy

```
B <- 10000
stick <- replicate(B, {
        doors <- as.character(1:3)
        prize <- sample(c("car","goat","goat"))    # puts prizes in random order
        prize_door <- doors[prize == "car"]    # note which door has prize
        my_pick  <- sample(doors, 1)    # note which door is chosen
        show <- sample(doors[!doors %in% c(my_pick, prize_door)],1)    # open door with no prize that isn't
chosen
        stick <- my_pick    # stick with original door
        stick == prize_door    # test whether the original door has the prize
})
mean(stick)    # probability of choosing prize door when sticking
```

Code: Monte Carlo simulation of switch strategy

```
switch <- replicate(B, {
        doors <- as.character(1:3)
        prize <- sample(c("car","goat","goat"))    # puts prizes in random order
        prize_door <- doors[prize == "car"]    # note which door has prize
        my_pick  <- sample(doors, 1)    # note which door is chosen first
        show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1)    # open door with no prize that isn't
chosen
        switch <- doors[!doors%in%c(my_pick, show)]    # switch to the door that wasn't chosen first or opened
        switch == prize_door    # test whether the switched door has the prize
})
mean(switch)    # probability of choosing prize door when switching
```

Sample Probability code example

Friday, February 14, 2020 8:02 PM

```
# Assign a variable 'n' as the number of remaining games.
n <- 6

# Assign a variable `outcomes` as a vector of possible game outcomes, where 0 indicates a loss and 1 indicates a win for the Cavs.
outcomes <- c(0,1)

# Assign a variable `l` to a list of all possible outcomes in all remaining games. Use the `rep` function on `list(outcomes)` to create list of length `n`.
l <- rep(list(outcomes), n)

# Create a data frame named 'possibilities' that contains all combinations of possible outcomes for the remaining games.
possibilities <- expand.grid(l)

# Create a vector named 'results' that indicates whether each row in the data frame 'possibilities' contains enough wins for the Cavs to win the series.
results <- rowSums(possibilities)>=4

# Calculate the proportion of 'results' in which the Cavs win the series. Print the outcome to the console.
mean(results)



# Let's assign the variable 'p' as the vector of probabilities that team A will win.
p <- seq(0.5, 0.95, 0.025)

# Given a value 'p', the probability of winning the series for the underdog team B can be computed with the following function based on a Monte Carlo simulation:
prob_win <- function(p){
  B <- 10000
  result <- replicate(B, {
    b_win <- sample(c(1,0), 7, replace = TRUE, prob = c(1-p, p))
    sum(b_win)>=4
    })
  mean(result)
}

# Apply the 'prob_win' function across the vector of probabilities that team A will win to determine the probability that team B will win. Call this object 'Pr'.
Pr <- sapply(p, prob_win)

# Plot the probability 'p' on the x-axis and 'Pr' on the y-axis.

plot(p, Pr)



# Given a value 'p', the probability of winning the series for the underdog team B can be computed with the following function based on a Monte Carlo simulation:
```

```
prob_win <- function(N, p=0.75){
    B <- 10000
    result <- replicate(B, {
      b_win <- sample(c(1,0), N, replace = TRUE, prob = c(1-p, p))
      sum(b_win)>=(N+1)/2
      })
    mean(result)
  }

# Assign the variable 'N' as the vector of series lengths. Use only odd numbers ranging from 1 to 25 games.
N <- seq(1, 25,2)

# Apply the 'prob_win' function across the vector of series lengths to determine the probability that team B will win. Call this object `Pr`.
Pr <-sapply(N, prob_win)

# Plot the number of games in the series 'N' on the x-axis and 'Pr' on the y-axis.
plot(N, Pr)
```

Continuous Probability

Friday, February 14, 2020 8:32 PM

Code: Cumulative distribution function

Define x as male heights from the **dslabs** heights dataset:

```
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

Given a vector **x**, we can define a function for computing the CDF of **x** using:

```
F <- function(a) mean(x <= a)
1 - F(70)    # probability of male taller than 70 inches
```

Plotting Probability Density dnorm() for the normal distribution

```
library(tidyverse)
x <- seq(-4, 4, length = 100)
data.frame(x, f = dnorm(x)) %>%
ggplot(aes(x, f)) +
geom_line()
```

Note that dnorm() gives densities for the standard normal distribution by default. Probabilities for alternative normal distributions with mean mu and standard deviation sigma can be evaluated with:

```
dnorm(z, mu, sigma)
```

Monte Carlo

Key points

- rnorm(n, avg, s) generates n random numbers from the normal distribution with average avg and standard deviation s.
- By generating random numbers from the normal distribution, we can simulate height data with similar properties to our dataset. Here we generate simulated height data using the normal distribution.

Code: Generating normally distributed random numbers

```
# define x as male heights from dslabs data
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
# generate simulated height data using normal distribution - both datasets should have n observations
n <- length(x)
avg <- mean(x)
s <- sd(x)
simulated_heights <- rnorm(n, avg, s)
# plot distribution of simulated_heights
data.frame(simulated_heights = simulated_heights) %>%
  ggplot(aes(simulated_heights)) +
  geom_histogram(color="black", binwidth = 2)
```

Code: Monte Carlo simulation of tallest person over 7 feet

```
B <- 10000
tallest <- replicate(B, {
  simulated_data <- rnorm(800, avg, s)    # generate 800 normally distributed random heights
  max(simulated_data)    # determine the tallest height
```

```
})
mean(tallest >= 7*12)   # proportion of times that tallest person exceeded 7 feet (84 inches)
```

Other Continuous Distributions

Key points

- You may encounter other continuous distributions (Student t, chi-squared, exponential, gamma, beta, etc.).
- R provides functions for density (d), quantile (q), probability distribution (p) and random number generation (r) for many of these distributions.
- Each distribution has a matching abbreviation (for example, norm() or t()) that is paired with the related function abbreviations (**d**, **p**, **q**, **r**) to create appropriate functions.
- For example, use rt() to generate random numbers for a Monte Carlo simulation using the Student t distribution.

Code: Plotting the normal distribution with dnorm

Use d to plot the density function of a continuous distribution. Here is the density function for the normal distribution (abbreviation norm()):

```
x <- seq(-4, 4, length.out = 100)
data.frame(x, f = dnorm(x)) %>%
  ggplot(aes(x,f)) +
  geom_line()
```

```
# Assign a variable 'female_avg' as the average female height.
female_avg <- 64

# Assign a variable 'female_sd' as the standard deviation for female heights.
female_sd <- 3

# To a variable named 'taller', assign the value of a height that is one SD taller than average.
taller <- female_avg+female_sd

# To a variable named 'shorter', assign the value of a height that is one SD shorter than average.
shorter <- female_avg-female_sd

# Calculate the probability that a randomly selected female is between the desired height range. Print this value to the console.
pnorm(taller, female_avg, female_sd) - pnorm(shorter, female_avg, female_sd)


# Determine the height of a man in the 99th percentile of the distribution.
qnorm(.99 ,male_avg, male_sd)
```

```
# The variable `B` specifies the number of times we want the simulation to run.
B <- 1000

# Use the `set.seed` function to make sure your answer matches the expected result after random number generation.
set.seed(1)
```

```
# Create an object called `highestIQ` that contains the highest IQ score from each random distribution of
10,000 people.
highestIQ <- replicate(B, {
sim <- rnorm(10000, 100, 15)
max(sim)
})
# Make a histogram of the highest IQ scores.
hist(highestIQ)
```

Random Variables

Saturday, February 22, 2020 8:53 AM

Key points

- Random variables are numeric outcomes resulting from random processes.
- Statistical inference offers a framework for quantifying uncertainty due to randomness.

Code: Modeling a random variable

```
# define random variable x to be 1 if blue, 0 otherwise
beads <- rep(c("red", "blue"), times = c(2, 3))
x <- ifelse(sample(beads, 1) == "blue", 1, 0)
# demonstrate that the random variable is different every time
ifelse(sample(beads, 1) == "blue", 1, 0)
ifelse(sample(beads, 1) == "blue", 1, 0)
ifelse(sample(beads, 1) == "blue", 1, 0)
```

Sampling models

Saturday, February 22, 2020 8:55 AM

Key points

- A sampling model models the random behavior of a process as the sampling of draws from an urn.
- The **probability distribution of a random variable** is the probability of the observed value falling in any given interval.
- We can define a CDF $F(a)=Pr(S \leq a)$ to answer questions related to the probability of S being in any interval.
- The average of many draws of a random variable is called its **expected value**.
- The standard deviation of many draws of a random variable is called its **standard error**.

Monte Carlo simulation: Chance of casino losing money on roulette

We build a sampling model for the random variable S that represents the casino's total winnings.

```
# sampling model 1: define urn, then sample
color <- rep(c("Black", "Red", "Green"), c(18, 18, 2)) # define the urn for the sampling model
n <- 1000
X <- sample(ifelse(color == "Red", -1, 1), n, replace = TRUE)
X[1:10]
# sampling model 2: define urn inside sample function by noting probabilities
x <- sample(c(-1, 1), n, replace = TRUE, prob = c(9/19, 10/19))    # 1000 independent draws
S <- sum(x)    # total winnings = sum of draws
S
```

We use the sampling model to run a Monte Carlo simulation and use the results to estimate the probability of the casino losing money.

```
n <- 1000    # number of roulette players
B <- 10000    # number of Monte Carlo experiments
S <- replicate(B, {
  X <- sample(c(-1,1), n, replace = TRUE, prob = c(9/19, 10/19))    # simulate 1000 spins
  sum(X)    # determine total profit
})
mean(S < 0)    # probability of the casino losing money
```

We can plot a histogram of the observed values of S as well as the normal density curve based on the mean and standard deviation of S.

```
library(tidyverse)
s <- seq(min(S), max(S), length = 100)    # sequence of 100 values across range of S
normal_density <- data.frame(s = s, f = dnorm(s, mean(S), sd(S))) # generate normal density for S
data.frame (S = S) %>%    # make data frame of S for histogram
  ggplot(aes(S, ..density..)) +
  geom_histogram(color = "black", binwidth = 10) +
  ylab("Probability") +
  geom_line(data = normal_density, mapping = aes(s, f), color = "blue")
```

What proportion of the list is less than or equal to a certain value

```
Avg<- sum(x)/length(x)
S <- sqrt(sum((x-avg)^2 / length(x))
```

NOTION FOR RANDOM VARIABLES

Key points

- Capital letters denote random variables (X) and lowercase letters denote observed values (x).
- In the notation $Pr(X=x)$, we are asking how frequently the random variable X is equal to the value x. For example, if $x=6$, this statement becomes $Pr(X=6)$.

Central Limit Theorem

Key points

- The Central Limit Theorem (CLT) says that the distribution of the sum of a random variable is approximated by a normal distribution.
- The expected value of a random variable, $E[X]=\mu$, is the average of the values in the urn. This represents the expectation of one draw.
- The standard error of one draw of a random variable is the standard deviation of the values in the urn.
- The expected value of the sum of draws is the number of draws times the expected value of the random variable.
- The standard error of the sum of independent draws of a random variable is the square root of the number of draws times the standard deviation of the urn.

Equations

These equations apply to the case where there are only two outcomes, a and b with proportions p and $1-p$ respectively. The general principles above also apply to random variables with more than two outcomes.

If we play a casino game over and over agin the casin wins X amount on average
Expected value of a random variable:

$ap+b(1-p)$

# of draws times the average of the numbers in the urn

Expected value of the sum of n draws of a random variable:

$n\times(ap+b(1-p))$

|| is absolute value

Standard deviation of an urn with two values:

$$|b-a|\sqrt{p(1-p)}$$

Standard error of the sum of n draws of a random variable:

$$\sqrt{n}\times|b-a|\sqrt{p(1-p)}$$

 # Calculate the expected outcome
(17*p_green)+ (-1*(1-p_green))

For multiple instances (n)
Sqrt(n) * (17*p_green)+ (-1*(1-p_green))
Expected value
n*(20-18)/38

If 1000 people bet on red on the roulette wheel casino is expected to win $50 with a standard error of $32
N <- 1000
Sqrt(n) *2* sqrt(90)/19

Probability of losing money
Pnorm(0, m, sd)

```
# Define the number of bets using the variable 'n'
n <- 10000

# Assign a variable `p_green` as the probability of the ball landing in a green pocket
p_green <- 2 / 38

# Assign a variable `p_not_green` as the probability of the ball not landing in a green pocket
p_not_green <- 1 - p_green

# Compute the standard error of 'Y', the mean outcome per bet from 10,000 bets.
abs(17-(-1))* sqrt(p_green*p_not_green) / sqrt(10000)
```

```
# We defined the average using the following code
avg <- 17*p_green + -1*p_not_green

# We defined standard error using this equation
se <- 1/sqrt(n) * (17 - -1)*sqrt(p_green*p_not_green)

# Given this average and standard error, determine the probability of winning more than $0. Print the result to the console.
1 - pnorm(0, avg, se)
```

Law of Large Numbers

Saturday, February 22, 2020 10:54 AM

Key points

- The law of large numbers states that as n increases, the standard error of the average of a random variable decreases. In other words, when n is large, the average of the draws converges to the average of the urn.
- The law of large numbers is also known as the law of averages.
- The law of averages only applies when n is very large and events are independent. It is often misused to make predictions about an event being "due" because it has happened less frequently than expected in a small sample size.

How Large is Large in CLT?

Central Limit Theorem

Key points

- The sample size required for the Central Limit Theorem and Law of Large Numbers to apply differs based on the probability of success.
- If the probability of success is high, then relatively few observations are needed.
- As the probability of success decreases, more observations are needed.
- If the probability of success is extremely low, such as winning a lottery, then the Central Limit Theorem may not apply even with extremely large sample sizes. The normal distribution is not a good approximation in these cases, and other distributions such as the Poisson distribution (not discussed in these courses) may be more appropriate.

Poisson distribution is used for very low porability events (example the lottery)

Central Limit Theorem

Saturday, February 22, 2020 11:10 AM

Using the CLT, we can skip the Monte Carlo simulation and instead compute the probability of the casino losing money using this approximation:
Mu = mean (expected)
Se = standard error

```
mu <- n * (20-18)/38
se <-  sqrt(n) * 2 * sqrt(90)/19
pnorm(0, mu, se)
```

From <https://rafalab.github.io/dsbook/random-variables.html>

CLT works better when the sample size is large

Interest Rates

Saturday, February 22, 2020 2:46 PM

Key points

- Interest rates for loans are set using the probability of loan defaults to calculate a rate that minimizes the probability of losing money.
- We can define the outcome of loans as a random variable. We can also define the sum of outcomes of many loans as a random variable.
- The Central Limit Theorem can be applied to fit a normal distribution to the sum of profits over many loans. We can use properties of the normal distribution to calculate the interest rate needed to ensure a certain probability of losing money for a given probability of default.

Code: Interest rate sampling model

```
n <- 1000
loss_per_foreclosure <- -200000
p <- 0.02
defaults <- sample( c(0,1), n, prob=c(1-p, p), replace = TRUE)
sum(defaults * loss_per_foreclosure)
```

Code: Interest rate Monte Carlo simulation

```
B <- 10000
losses <- replicate(B, {
  defaults <- sample( c(0,1), n, prob=c(1-p, p), replace = TRUE)
 sum(defaults * loss_per_foreclosure)
})
```

Code: Plotting expected losses

```
library(tidyverse)
data.frame(losses_in_millions = losses/10^6) %>%
  ggplot(aes(losses_in_millions)) +
  geom_histogram(binwidth = 0.6, col = "black")
```

Code: Expected value and standard error of the sum of 1,000 loans

```
n*(p*loss_per_foreclosure + (1-p)*0)    # expected value
sqrt(n)*abs(loss_per_foreclosure)*sqrt(p*(1-p))    # standard error
```

Code: Calculating interest rates for expected value of 0

We can calculate the amount x to add to each loan so that the expected value is 0 using the equation $lp+x(1-p)=0$. Note that this equation is the definition of expected value given a loss per foreclosure l with foreclosure probability p and profit x if there is no foreclosure (probability $1-p$).

We solve for $x=-lp1-p$ and calculate x:

```
x = - loss_per_foreclosure*p/(1-p)
x
```

On a $180,000 loan, this equals an interest rate of:

```
x/180000
```

Equations: Calculating interest rate for 1% probability of losing money

We want to calculate the value of x for which $Pr(S<0)=0.01$. The expected value $E[S]$ of the sum of $n=1000$ loans given our definitions of x, l and p is:

$$\mu_S=(lp+x(1-p))*n$$

And the standard error of the sum of n loans, $SE[S]$, is:

$$\sigma_S=|x-l|\sqrt{np(1-p)}$$

Because we know the definition of a Z-score is $Z=\frac{x-\mu}{\sigma}$, we know that $Pr(S<0)=Pr(Z<\frac{-\mu}{\sigma})$. Thus, $Pr(S<0)=0.01$ equals:

$$Pr(Z<\frac{-\{lp+x(1-p)\}\sqrt{n}}{\sqrt{np(1-p)}})=0.01$$

z<-qnorm(0.01) gives us the value of z for which $Pr(Z \leq z)=0.01$, meaning:

$$z=\frac{-\{lp+x(1-p)\}\sqrt{n}}{\sqrt{np(1-p)}}$$

Solving for x gives:

$$x=\frac{-lnp-znp(1-p)}{\sqrt{n}(1-p)+znp(1-p)}$$

Code: Calculating interest rate for 1% probability of losing money

```
l <- loss_per_foreclosure
z <- qnorm(0.01)
x <- -l*( n*p - z*sqrt(n*p*(1-p)))/ ( n*(1-p) + z*sqrt(n*p*(1-p)))\x
x/180000   # interest rate
loss_per_foreclosure*p + x*(1-p)   # expected value of the profit per loan
n*(loss_per_foreclosure*p + x*(1-p)) # expected value of the profit over n loans
```

Code: Monte Carlo simulation for 1% probability of losing money

Note that your results will vary from the video because the seed is not set.

```
B <- 100000
profit <- replicate(B, {
  draws <- sample( c(x, loss_per_foreclosure), n,
                   prob=c(1-p, p), replace = TRUE)
  sum(draws)
})
mean(profit)   # expected value of the profit over n loans
mean(profit<0)   # probability of losing money
```

Key points

- The Central Limit Theorem states that the sum of independent draws of a random variable follows a normal distribution. However, when the draws are not independent, this assumption does not hold.
- If an event changes the probability of default for all borrowers, then the probability of the bank losing money changes.
- Monte Carlo simulations can be used to model the effects of unknown changes in the probability of default.

Code: Expected value with higher default rate and interest rate

```
p <- .04
loss_per_foreclosure <- -200000
r <- 0.05
x <- r*180000
loss_per_foreclosure*p + x*(1-p)
```

Equations: Probability of losing money

We can define our desired probability of losing money, Z, as:

$$Pr(S<0)=Pr(Z<\frac{-E[S]}{SE[S]})=Pr(Z<z)$$

If μ is the expected value of the urn (one loan) and σ is the standard deviation of the urn (one loan), then $E[S]=n\mu$ and $SE[S]=\sqrt{n}\sigma$.

As in the previous video, we define the probability of losing money $z=0.01$. In the first equation, we

can see that:

$$z = -E[S]SE[S]$$

It follows that:

$$z = -n\mu n - - \sqrt{}\sigma = -n - - \sqrt{}\mu\sigma$$

To find the value of n for which z is less than or equal to our desired value, we take $z \leq -n\sqrt{}\mu\sigma$ and solve for n:

$$n \geq z2\sigma 2\mu 2$$

Code: Calculating number of loans for desired probability of losing money

The number of loans required is:

```
z <- qnorm(0.01)
l <- loss_per_foreclosure
n <- ceiling((z^2*(x-l)^2*p*(1-p))/(l*p + x*(1-p))^2)
n   # number of loans required
n*(loss_per_foreclosure*p + x * (1-p))   # expected profit over n loans
```

Code: Monte Carlo simulation with known default probability

This Monte Carlo simulation estimates the expected profit given a known probability of default $p = 0.04$. Note that your results will differ from the video because the seed is not set.

```
B <- 10000
p <- 0.04
x <- 0.05 * 180000
profit <- replicate(B, {
  draws <- sample( c(x, loss_per_foreclosure), n,
                   prob=c(1-p, p), replace = TRUE)
  sum(draws)
})
mean(profit)
```

Code: Monte Carlo simulation with unknown default probability

This Monte Carlo simulation estimates the expected profit given an unknown probability of default $0.03 \leq p \leq 0.05$, modeling the situation where an event changes the probability of default for all borrowers simultaneously. Note that your results will differ from the video because the seed is not set.

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
  new_p <- 0.04 + sample(seq(-0.01, 0.01, length = 100), 1)
  draws <- sample( c(x, loss_per_foreclosure), n,
                   prob=c(1-new_p, new_p), replace = TRUE)
  sum(draws)
})
mean(profit)   # expected profit
mean(profit < 0)   # probability of losing money
mean(profit < -10000000)   # probability of losing over $10 million
```

Sampling & Inference

Sunday, February 23, 2020 9:36 AM

Course overview

In this course, we will learn:

- *statistical inference*, the process of deducing characteristics of a population using data from a random sample
- the statistical concepts necessary to define *estimates* and *margins of errors*
- how to *forecast future results* and estimate the precision of our forecast
- how to calculate and interpret *confidence intervals* and *p-values*

Key points

- Information gathered from a small random sample can be used to infer characteristics of the entire population.
- Opinion polls are useful when asking everyone in the population is impossible.
- A common use for opinion polls is determining voter preferences in political elections for the purposes of forecasting election results.
- The *spread* of a poll is the estimated difference between support two candidates or options.

Sampling Model Parameters and Estimates (used to estimate parameter(p) example fatality rate

Key points

- The task of statistical inference is to estimate an unknown population parameter using observed data from a sample.
- In a sampling model, the collection of elements in the urn is called the *population.*
- A *parameter* is a number that summarizes data for an entire population.
- A *sample* is observed data from a subset of the population.
- An *estimate* is a summary of the observed data about a parameter that we believe is informative. It is a data-driven guess of the population parameter.
- We want to predict the proportion of the blue beads in the urn, the parameter p. The proportion of red beads in the urn is $1-p$ and the *spread* is $2p-1$.
- The sample proportion is a random variable. Sampling gives random results drawn from the population distribution.

Code: Function for taking a random draw from a specific urn

The **dslabs** package includes a function for taking a random draw of size n from the urn described in the video:

```
library(tidyverse)
library(dslabs)
take_poll(25)  # draw 25 beads
```

The Sample Average

Key points

Key points

- Many common data science tasks can be framed as estimating a parameter from a sample.
- We illustrate statistical inference by walking through the process to estimate p. From the estimate of p, we can easily calculate an estimate of the spread, $2p-1$.
- Consider the random variable X that is 1 if a blue bead is chosen and 0 if a red bead is chosen. The proportion of blue beads in N draws is the average of the draws $X_1, \ldots, X_N$.
- $\bar{X}$ is the *sample average*. In statistics, a bar on top of a symbol denotes the average. $\bar{X}$ is a random variable because it is the average of random draws - each time we take a sample, $\bar{X}$ is different.

$$\bar{X} = \frac{X_1+X_2+\ldots+X_N}{N}$$

- The number of blue beads drawn in N draws, $N\bar{X}$, is N times the proportion of values in the urn. However, we do not know the true proportion: we are trying to estimate this parameter p.

Ifstandard error is larger than the spread, the sample size is too small

Properties of Our Estimate

Key points

- When interpreting values of $\bar{X}$, it is important to remember that $\bar{X}$ is a random variable with an expected value and standard error that represents the sample proportion of positive events.
- The expected value of $\bar{X}$ is the parameter of interest p. This follows from the fact that $\bar{X}$ is the sum of independent draws of a random variable times a constant $1/N$.

$$\mathrm{E}(\bar{X}) = p$$

SE of the sample average

- As the number of draws N increases, the standard error of our estimate $\bar{X}$ decreases. The standard error of the average of $\bar{X}$ over N draws is:

$$\mathrm{SE}(\bar{X}) = \sqrt{p(1-p)/N}$$

  - In theory, we can get more accurate estimates of p by increasing N. In practice, there are limits on the size of N due to costs, as well as other factors we discuss later.
  - We can also use other random variable equations to determine the expected value of the sum of draws $\mathrm{E}(S)$ and standard error of the sum of draws $\mathrm{SE}(S)$.

$$\mathrm{E}(S) = Np$$

$$\mathrm{SE}(S) = \sqrt{Np(1-p)}$$

Code examples:

```
# `N` represents the number of people polled
N <- 25

# Create a variable `p` that contains 100 proportions ranging from 0 to 1 using the `seq` function
p <- seq(0,1, length = 100)

# Create a variable `se` that contains the standard error of each sample average
se <-sqrt(p*(1-p)/N)

# Plot `p` on the x-axis and `se` on the y-axis
plot(p, se)
```

```
# The vector `p` contains 100 proportions of Democrats ranging from 0 to 1 using the `seq` function
p <- seq(0, 1, length = 100)

# The vector `sample_sizes` contains the three sample sizes
sample_sizes <- c(25, 100, 1000)

# Write a for-loop that calculates the standard error `se` for every value of `p` for each of the three samples sizes `N` in the vector
`sample_sizes`. Plot the three graphs, using the `ylim` argument to standardize the y-axis across all three plots.
for  (N in sample_sizes) {
 se <-sqrt(p*(1-p)/N)
 plot(p, se, ylim = c(0, .01))
}
```

Our estimate for the difference in proportions of Democrats and Republicans is $d=\bar{X}-(1-\bar{X})$. Which derivation correctly uses the rules we learned about sums of random variables and scaled random variables to derive the expected value of d?

$\mathrm{E}[\bar{X} - (1-\bar{X})] = \mathrm{E}[2\bar{X} - 1] \;\; = 2\mathrm{E}[\bar{X}] - 1 \;\; = 2p - 1 \;\; = p - (1-p)$

Our estimate for the difference in proportions of Democrats and Republicans is $d = \bar{X} - (1 - \bar{X})$.

Which derivation correctly uses the rules we learned about sums of random variables and scaled random variables to derive the standard error of d?

Instructions 50 XP

Possible Answers

- $\mathrm{SE}[\bar{X} - (1 - \bar{X})] = \mathrm{SE}[2\bar{X} - 1] = 2\mathrm{SE}[\bar{X}] = 2\sqrt{p/N}$
- $\mathrm{SE}[\bar{X} - (1 - \bar{X})] = \mathrm{SE}[2\bar{X} - 1] = 2\mathrm{SE}[\bar{X} - 1] = 2\sqrt{p(1-p)/N} - 1$
- $\mathrm{SE}[\bar{X} - (1 - \bar{X})] = \mathrm{SE}[2\bar{X} - 1] = 2\mathrm{SE}[\bar{X}] = 2\sqrt{p(1-p)/N}$
- $\mathrm{SE}[\bar{X} - (1 - \bar{X})] = \mathrm{SE}[\bar{X} - 1] = \mathrm{SE}[\bar{X}] = \sqrt{p(1-p)/N}$

```
# `N` represents the number of people polled
N <- 25

# `p` represents the proportion of Democratic voters
p <- 0.45

# Calculate the standard error of the spread. Print this value to the console.

2*sqrt(p*(1-p)/N)
```

```
# Write a function called `take_sample` that takes `p` and `N` as arguements and returns the average value of a randomly sampled population.

take_sample <- function(p, N) {
  x <-sample(c(1, 0), N, replace = TRUE, prob = c(p, 1-p))
  mean(x)
}


# Use the `set.seed` function to make sure your answer matches the expected result after random sampling
set.seed(1)

# Define `p` as the proportion of Democrats in the population being polled
p <- 0.45

# Define `N` as the number of people polled
N <- 100

# Call the `take_sample` function to determine the sample average of `N` randomly selected people from a population containing a proportion of Democrats equal to `p`. Print this value to the console.
take_sample(p, N)
```

```
# Define `p` as the proportion of Democrats in the population being polled
p <- 0.45
```

```
# Define `N` as the number of people polled
N <- 100

# The variable `B` specifies the number of times we want the sample to be replicated
B <- 10000

# Use the `set.seed` function to make sure your answer matches the expected result after random sampling
set.seed(1)

# We generated `errors` by subtracting the estimate from the actual proportion of Democratic voters
errors <- replicate(B, p - take_sample(p, N))

# Calculate the mean of the absolute value of each simulated error. Print this value to the console.
mean(abs(errors))
```

```
# Define `p` as the proportion of Democrats in the population being polled
p <- 0.45

# Define `N` as the number of people polled
N <- 100

# The variable `B` specifies the number of times we want the sample to be replicated
B <- 10000

# Use the `set.seed` function to make sure your answer matches the expected result after random sampling
set.seed(1)

# We generated `errors` by subtracting the estimate from the actual proportion of Democratic voters
errors <- replicate(B, p - take_sample(p, N))

# Calculate the standard deviation of `errors` (or of the spread)
sqrt(mean(errors^2))
```

```
# Define `p` as a proportion of Democratic voters to simulate
p <- 0.45

# Define `N` as the sample size
N <- 100

# Use the `set.seed` function to make sure your answer matches the expected result after random sampling
set.seed(1)

# Define `X` as a random sample of `N` voters with a probability of picking a Democrat ('1') equal to `p`
X <- sample(c(1, 0), N, replace = TRUE, prob = c(p, 1-p))

# Define `X_bar` as the average sampled proportion
X_bar <- mean(X)

# Calculate the standard error of the estimate. Print the result to the console.
sqrt(X_bar*(1-X_bar)/N)
```

Create a plot of the largest standard error for N ranging from 100 to 5,000. Based on this plot, how large does the sample size have to be to have a standard error of about 1%?

```
N <- seq(100, 5000, len = 100)
p <- 0.5
se <- sqrt(p*(1-p)/N)
```

```
# Define `p` as the proportion of Democrats in the population being polled
p <- 0.45

# Define `N` as the number of people polled
N <- 100
```

Calculate the probability that the estimated proportion of Democrats in the population is greater than 0.5. Print this value to the console.
1-pnorm(0.5, p, (sqrt(p*(1-p)/N)))

# Define `N` as the number of people polled
N <-100

# Define `X_hat` as the sample average
X_hat <- 0.51

# Define `se_hat` as the standard error of the sample average
se_hat <- sqrt(X_hat*(1-X_hat)/N)

Calculate the probability that the error is 0.01 or larger
1- pnorm(.01, 0, se_hat) + pnorm(-0.01, 0, se_hat)

Central Limit Theorem in Practice

Sunday, February 23, 2020 1:28 PM

Key points

- Because $\bar{X}$ is the sum of random draws divided by a constant, the distribution of $\bar{X}$ is approximately normal.
- We can convert $\bar{X}$ to a standard normal random variable Z:

$$Z = \frac{\bar{X} - \mathrm{E}(\bar{X})}{\mathrm{SE}(\bar{X})}$$

- The probability that $\bar{X}$ is within .01 of the actual value of p is:

$$\Pr\left(Z \leq .01/\sqrt{p\left(1-p\right)/N}\right) - \Pr\left(Z \leq -.01/\sqrt{p\left(1-p\right)/N}\right)$$

- The Central Limit Theorem (CLT) still works if $\bar{X}$ is used in place of p. This is called a *plug-in estimate*. Hats over values denote estimates. Therefore:

$$\hat{\mathrm{SE}}\left(\bar{X}\right) = \sqrt{\bar{X}\left(1-\bar{X}\right)/N}$$

- Using the CLT, the probability that $\bar{X}$ is within .01 of the actual value of p is:

$$\Pr\left(Z \leq .01/\sqrt{\bar{X}\left(1-\bar{X}\right)/N}\right) - \Pr\left(Z \leq -.01/\sqrt{\bar{X}\left(1-\bar{X}\right)/N}\right)$$

Code: Computing the probability of $\bar{X}$ being within .01 of p

```
X_hat <- 0.48
se <- sqrt(X_hat*(1-X_hat)/25)
pnorm(0.01/se) - pnorm(-0.01/se)
```

Margin of Error

Key points

- The *margin of error* is defined as 2 times the standard error of the estimate $\bar{X}$.
- There is about a 95% chance that $\bar{X}$ will be within two standard errors of the actual parameter p.

A Monte Carlo Simulation for the CLT

Sunday, February 23, 2020 1:35 PM

Key points

- We can run Monte Carlo simulations to compare with theoretical results assuming a value of p.
- In practice, p is unknown. We can corroborate theoretical results by running Monte Carlo simulations with one or several values of p.
- One practical choice for p when modeling is $\bar{X}$, the observed value of X^ in a sample.

Code: Monte Carlo simulation using a set value of p

```
p <- 0.45    # unknown p to estimate
N <- 1000
# simulate one poll of size N and determine x_hat
x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
x_hat <- mean(x)
# simulate B polls of size N and determine average x_hat
B <- 10000    # number of replicates
N <- 1000    # sample size per replicate
x_hat <- replicate(B, {
  x <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
  mean(x)
})
```

Code: Histogram and QQ-plot of Monte Carlo results

```
library(tidyverse)
library(gridExtra)
p1 <- data.frame(x_hat = x_hat) %>%
  ggplot(aes(x_hat)) +
  geom_histogram(binwidth = 0.005, color = "black")
p2 <- data.frame(x_hat = x_hat) %>%
  ggplot(aes(sample = x_hat)) +
  stat_qq(dparams = list(mean = mean(x_hat), sd = sd(x_hat))) +
  geom_abline() +
  ylab("X_hat") +
  xlab("Theoretical normal")
grid.arrange(p1, p2, nrow=1)
```

Key points

- The spread between two outcomes with probabilities p and $1-p$ is $2p-1$.
- The expected value of the spread is $2\bar{X}-1$.
- The standard error of the spread is $2\hat{\mathrm{SE}}(\bar{X})$.
- The margin of error of the spread is 2 times the margin of error of $\bar{X}$.

Bias: Why Not Run a Very Large Poll?

Sunday, February 23, 2020 1:40 PM

Key points

- An extremely large poll would theoretically be able to predict election results almost perfectly.
- These sample sizes are not practical. In addition to cost concerns, polling doesn't reach everyone in the population (eventual voters) with equal probability, and it also may include data from outside our population (people who will not end up voting).
- These systematic errors in polling are called *bias*. We will learn more about bias in the future.

Code: Plotting margin of error in an extremely large poll over a range of values of p

```
library(tidyverse)
N <- 100000
p <- seq(0.35, 0.65, length = 100)
SE <- sapply(p, function(x) 2*sqrt(x*(1-x)/N))
data.frame(p = p, SE = SE) %>%
  ggplot(aes(p, SE)) +
  geom_line()
>
```

Confidence intervals

Sunday, February 23, 2020 6:41 PM

Key points

- We can use statistical theory to compute the probability that a given interval contains the true parameter p.
- 95% confidence intervals are intervals constructed to have a 95% chance of including p. The margin of error is approximately a 95% confidence interval.
- The start and end of these confidence intervals are random variables.
- To calculate any size confidence interval, we need to calculate the value z for which $\Pr\left(-z \leq Z \leq z\right)$ equals the desired confidence. For example, a 99% confidence interval requires calculating z for $\Pr\left(-z \leq Z \leq z\right) = 0.99$.
- For a confidence interval of size q, we solve for $z = 1 - \frac{1-q}{2}$.
- To determine a 95% confidence interval, use `z <- qnorm(0.975)`. This value is slightly smaller than 2 times the standard error.

Code: geom_smooth confidence interval example

The shaded area around the curve is related to the concept of confidence intervals.

```
data("nhtemp")
data.frame(year = as.numeric(time(nhtemp)), temperature = as.numeric(nhtemp)) %>%
  ggplot(aes(year, temperature)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Average Yearly Temperatures in New Haven")
```

Code: Monte Carlo simulation of confidence intervals

Note that to compute the exact 95% confidence interval, we would use qnorm(.975)*SE_hat instead of 2 *SE_hat.

```
p <- 0.45
N <- 1000
X <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))   # generate N observations
X_hat <- mean(X)   # calculate X_hat
SE_hat <- sqrt(X_hat*(1-X_hat)/N)   # calculate SE_hat, SE of the mean of N observations
c(X_hat - 2*SE_hat, X_hat + 2*SE_hat)   # build interval of 2*SE above and below mean
```

Code: Solving for Z with qnorm

```
z <- qnorm(0.995)   # calculate z to solve for 99% confidence interval
pnorm(qnorm(0.995))   # demonstrating that qnorm gives the z value for a given probability
pnorm(qnorm(1-0.995))   # demonstrating symmetry of 1-qnorm
pnorm(z) - pnorm(-z)   # demonstrating that this z value gives correct probability for interval
```

A Monte Carlo Simulation for Confidence Intervals

Key points

- We can run a Monte Carlo simulation to confirm that a 95% confidence interval contains the true value of p 95% of the time.
- A plot of confidence intervals from this simulation demonstrates that most intervals include p, but roughly 5% of intervals miss the true value of p.

Code: Monte Carlo simulation

Note that to compute the exact 95% confidence interval, we would use qnorm(.975)*SE_hat instead of 2 *SE_hat.

```
B <- 10000
inside <- replicate(B, {
  X <- sample(c(0,1), size = N, replace = TRUE, prob = c(1-p, p))
  X_hat <- mean(X)
SE_hat <- sqrt(X_hat*(1-X_hat)/N)
  between(p, X_hat - 2*SE_hat, X_hat + 2*SE_hat)   # TRUE if p in confidence interval
})
mean(inside)
```

Key points

- The 95% confidence intervals are random, but p is not random.
- 95% refers to the probability that the random interval falls on top of p.
- It is technically incorrect to state that p has a 95% chance of being in between two values because that implies p is random.

POWER

Key points

- If we are trying to predict the result of an election, then a confidence interval that includes a spread of 0 (a tie) is not helpful.
- A confidence interval that includes a spread of 0 does not imply a close election, it means the sample size is too small.
- Power is the probability of detecting an effect when there is a true effect to find. Power increases as sample size increases, because larger sample size means smaller standard error.

Code: Confidence interval for the spread with sample size of 25

Note that to compute the exact 95% confidence interval, we would use `c(-qnorm(.975), qnorm(.975))` instead of 1.96.

```
N <- 25
X_hat <- 0.48
(2*X_hat - 1) + c(-2, 2)*2*sqrt(X_hat*(1-X_hat)/N)
```

https://cran.rstudio.com/src/contrib/pdftools_2.3.tar.gz

p-vlaues

Friday, February 28, 2020 8:29 AM

Another way to ask are there more blue beads than red beads, is the spread greater than 0

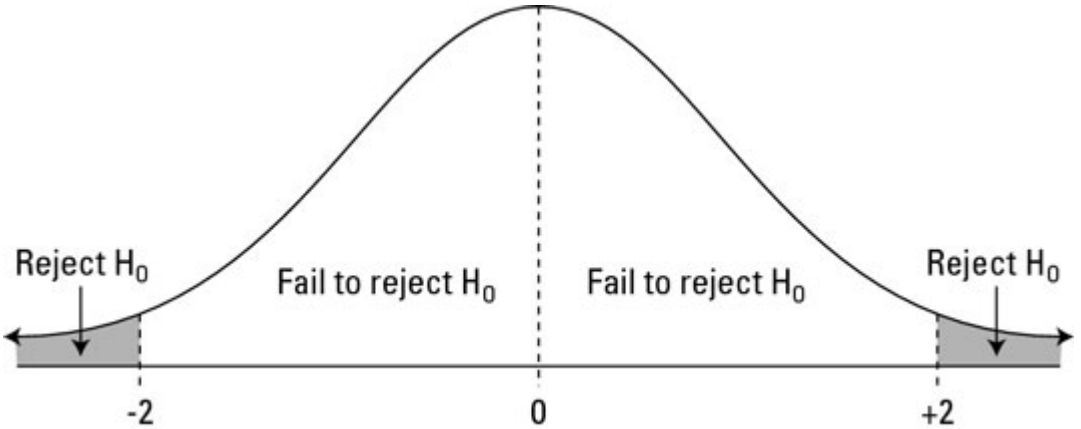
Key points

- The null hypothesis is the hypothesis that there is no effect. In this case, the null hypothesis is that the spread is 0, or $p=0.5$.
- The p-value is the probability of detecting an effect of a certain size or larger when the null hypothesis is true.
- We can convert the probability of seeing an observed value under the null hypothesis into a standard normal random variable. We compute the value of Z that corresponds to the observed result, and then use that Z to compute the p-value.
- If a 95% confidence interval does not include our observed value, then the p-value must be smaller than 0.05.
- It is preferable to report confidence intervals instead of p-values, as confidence intervals give information about the size of the estimate and p-values do not.

Code: Computing a p-value for observed spread of 0.02

```
N <- 100    # sample size
z <- sqrt(N) * 0.02/0.5    # spread of 0.02
1 - (pnorm(z) - pnorm(-z))
```

The p-value is the probability of observing a value as extreme or more extreme than the result given that the null hypothesis is true.
In the context of the normal distribution, this refers to the probability of observing a Z-score whose absolute value is as high or higher than the Z-score of interest.
Suppose we want to find the p-value of an observation 2 standard deviations larger than the mean. This means we are looking for anything with $|z| \geq 2$. $|z| \geq 2$
Graphically, the p-value gives the probability of an observation that's at least as far away from the mean or further. This plot shows a standard normal distribution (centered at $z=0$ with a standard deviation of 1). The shaded tails are the region of the graph that are 2 standard deviations or more away from the mean.



The right tail can be found with 1-pnorm(2). We want to have both tails, though, because we want to find the probability of any observation as far away from the mean or farther, in either direction. (This is what's meant by a two-tailed p-value.) Because the distribution is symmetrical, the right and left tails are the same size and we know that our desired value is just 2*(1-pnorm(2)).
Recall that, by default, pnorm() gives the CDF for a normal distribution with a mean of $\mu=0$ and standard deviation of $\sigma=1$. To find p-values for a given z-score z in a normal distribution with mean mu and standard deviation sigma, use 2*(1-pnorm(z, mu, sigma)) instead.

From <https://courses.edx.org/courses/course-v1:HarvardX+PH125.4x+1T2020/courseware/9cde20bcc44842e789fbe5bbcd79a7a2/c1cbebe9d71f4523b32f396fa8321121/3?activate_block_id=block-v1%3AHarvardX%2BPH125.4x%2B1T2020%2Btype%40video%2Bblock%4015063bbda04c42359f4d2f5785c40c1e>

UNIX

Sunday, March 1, 2020 2:30 PM

ls #list dir content
mkdir folder_name #create directory called "folder_name"
rmdir folder_name #remove an empty directory as long as it is empty
rm -r folder_name #remove dir that is not empty, "r" stands for recursive
cd: change dir
../ # two dots represents parent dir
. # single dot represents current workingdir
cd ~/projects # concatenate with forward slashes
cd ../.. # change to two parent layer beyond
cd - # whatever dir you were before
cd # return to the home dir

Key points
The mv command moves files.
[warning] mv will not ask you to confirm the move, and it could potentially overwrite a file.
The rm command removes files.
[warning] rm is permanent, which is different than throwing a folder in the trash.
Code
mv path-to-file path-to-destination-directory
rm filename-1 filename-2 filename-3

WARNING: rm is permanent

So, for example, if we want to move the file cv.tex tech from Resumes
to Reports, you could use the full paths like this--
mv ~/docs/resumes/cv.tex ~/docs/reports/.
You can also use relative paths, so you could do this--
cd ~/docs/resumes and then mv cv.tex to ../report/.
Or you could also do this-- cd ~/~/docs/reports/ then mv ../cv.tex/./

So, for example, to change the name from cv.tex resume.tex, we simply type,
first, we get to the directory they're in-- cd ~/~/docs/resumes and then we
move it.
mv cv.tex to resume.tex.

Note that we can combine a move and rename.
For example, here we're going to change cv.tex to resume.tex,
but we're going to put in another directory.
So first we go to the directory where it is--
cd ~/~/docs/resumes then we move cv.tex to the new directory with the file name
at the end-- ../reports/resume.tex.

We can also move entire directories.
So to move the resume directory into the reports directory,
we could do something like this--
mv ~/docs/resumes to ~/~/docs/reports/

Key points
less allows you to quickly look at the content of a file

Use q to exit the less page
use the arrows to navigate in the less page
Code
less cv.tex

Key points

- Ideally, files (code, data, output) should be structured and self-contained
- In a project, we prefer using relative paths (path relative to the default working directory) instead of the full path so that code can run smoothly on other individual's computers.
- It is good practice to write a README.txt file to introduce the file structure to facilitate collaboration and for your future reference.

Code

```
########### In terminal ###########
cd ~   # move to home directory
mkdir projects   # make a new directory called projects
cd projects   # move to ~/projects directory
mkdir murders   # make new directory called murders inside of projects
cd murders   # move to ~/projects/murders/
mkdir data rda   # make two new directories, one is data the other is rda folder
ls   # to check if we indeed have one data folder and one rda folder
pwd   # check the current working directory
mkdir figs   # make a directory called figs to store figures
########## In RStudio ########
# pick existing directory as new project
getwd()   # to confirm current working directory
save()   # save into .rda file, .RData is also fine but less preferred
ggsave("figs/barplot.png")   # save a plot generated by ggplot2 to a dir called "figs"
```

What does each of ~, ., .., / represent, respectively?
Home directory, Current directory, Parent directory, Root directory

Arguments

Key points

- Arguments typically are defined using a dash (-) or two dashes (--) followed by a letter of a word.
- r: recursive. For example, rm -r <directory-name>: remove all files, subdirectories, files in subdirectories, subdirectories in subdirectories, etc.
- Combine arguments: rm -rf directory-name
- ls -a: Shows all files in the directories including hidden files (e.g. .git file when initializing using git init) (a for all).
- ls -l: Returns more information about the files (i.e. l for long).
- ls -t: Shows files in chronological order.
- ls -r: Reverses the order of how files are shown.
- ls -lart: Shows more information for all files in reverse chronological order.

- **Getting Help:** Use man + command name to get help (e.g. man ls). Note that it is not available for Git Bash. For Git Bash, you can use command -- help (e.g. ls --help).
- **Pipes:** Pipes the results of a command to the command after the pipe. Similar to the pipe %>% in R. For example, man ls | less (and its equivalent in Git Bash: ls --help | less). Also useful when listing files with many files (e.g ls -lart | less).

- * means any number of any combination of characters. Specifically, to list all html files: ls *.html and to remove all html files in a directory: rm *.html.
- ? means any single character. For example, to erase all files in the form file-001.html with the numbers going from 1 to

999: rm file-???.html.

- Combined wild cards: rm file-001.* to remove all files of the name file-001 regardless of suffix.
- **Warning: Combining rm with the * wild card can be dangerous. There are combinations of these commands that will erase your entire file system without asking you for confirmation. Make sure you understand how it works before using this wild card with the rm command.**

Key points

- In Unix, variables are distinguished from other entities by adding a $ in front. For example, the home directory is stored in $HOME.
- See home directory: echo $HOME
- See them all: env
- See what shell is being used: echo $SHELL (most common shell is bash)
- Change environmental variables: (*Don't actually run this command though!*) export PATH = /usr/bin/

- In Unix, all programs are files. They are called executables. So, ls, mv, and git are all files.
- To find where these program files are, use which. For example, which git would return /usr/bin/git.
- Type ls /usr/bin to see several executable files. There are other directories that hold program files (e.g. Application directory for Mac or Program Files directory in Windows).
- Type echo $PATH to see a list of directories separated by ":".
- Type the full path to run the user-created executables (e.g ./my-ls).
- Regular file -, directory d, executable x.
- This string also indicates the permission of the file: is it readable? writable? executable? Can other users on the system read the file? Can other users on the system edit the file? Can other users execute if the file is executable?

- Be aware of common commands and know what they do.
- open/start - On the mac open filename tries to figure out the right application of the filename and open it with that application. This is a very useful command. On Git Bash, you can try start filename. Try opening an R or Rmd file with open or start: it should open with RStudio.
- nano - A bare-bones text editor.
- ln - create a symbolic link. We do not recommend its use, but you should be familiar with it.
- tar - archive files and subdirectories of a directory into one file.
- ssh - connect to another computer.
- grep - search for patterns in a file.
- awk/sed - These are two very powerful commands that permit you to find specific strings in files and change them.

Github

Sunday, March 1, 2020 7:48 PM

Key points

- Recap: there are four stages: working directory, staging area, local repository, and upstream repository
- Clone an existing upstream repository (copy repo url from clone button, and type "git clone <url>"), and all three local stages are the same as upstream remote.
- The working directory is the same as the working directory in Rstudio. When we edit files we only change the files in this place.
- git status: tells how the files in the working directory are related to the files in other stages
- edits in the staging area are not tracked by the version control system by default - we add a file to the staging area by git add command
- git commit: to commit files from the staging area to local repository, we need to add a message stating what we are doing by git commit -m "something"
- git log: keeps track of all the changes we have made to the local repository
- git push: allows moving from the local repository to upstream repository, only if you have the permission (e.g. if it is yours)
- git fetch: update local repository to be like the upstream repository, from upstream to local
- git merge: make the updated local sync with the working directory and staging area
- To change everything in one shot (from upstream to working dir), use git pull (equivalent to combining git fetch + git merge)

Code

```
pwd
mkdir git-example
cd git-example
git clone https://github.com/rairizarry/murders.git
cd murders
ls
git status
echo "test" >> new-file.txt
echo "temporary" >> tmp.txt
git add new-file.txt
git status
git commit -m "adding a new file"
git status
echo "adding a second line" >> new-file.txt
git commit -m "minor change to new-file" new-file.txt
git status
git add
git log new-file.txt
git push
git fetch
git merge
```

Key points

- Recap: two ways to get started, one is cloning an existing repository, the other is initializing our own
- Create our own project on our computer (independent of Git) on our own machine
- Create an upstream repo on Github, copy repo's url
- Make a local git repository: On the local machine, in the project directory, use git init. Now git starts

tracking everything in the local repo.

- Now we need to start moving files into our local repo and connect local repo to the upstream remote by `git remote add origin <url>`
- **Note:** The first time you push to a new repository, you may also need to use these `git push` options: `git push --set-upstream origin master`. If you need to run these arguments but forget to do so, you will get an error with a reminder.

Code

```
cd ~/projects/murders
git init
git add README.txt
git commit -m "First commit. Adding README.txt file just to get started"
git remote add origin "https://github.com/rairizarry/murders.git"
git push    # you may need to add these arguments the first time: --set-upstream origin master
```

Data Wrangling - Reading files

Saturday, March 7, 2020 11:42 AM

Read in using Tidy

```
# generate a full path to a file
filename <- "murders.csv"
fullpath <- file.path(path, filename)
fullpath

# copy file from dslabs package to your working directory
file.copy(fullpath, getwd())

# check if the file exists
file.exists(filename)


library(dslabs)
library(tidyverse)    # includes readr
library(readxl)

# inspect the first 3 lines
read_lines("murders.csv", n_max = 3)

# read file in CSV format
dat <- read_csv(filename)

#read using full path
dat <- read_csv(fullpath)
head(dat)

#Ex：
path <- system.file("extdata", package = "dslabs")
files <- list.files(path)
files

filename <- "murders.csv"
filename1 <- "life-expectancy-and-fertility-two-countries-example.csv"
filename2 <- "fertility-two-countries-example.csv"
dat=read.csv(file.path(path, filename))
dat1=read.csv(file.path(path, filename1))
dat2=read.csv(file.path(path, filename2))
```

- excel_sheets gives you the names of the sheets in the excel file

R Base read in functions

Read.csv
Read.table
Read.delim

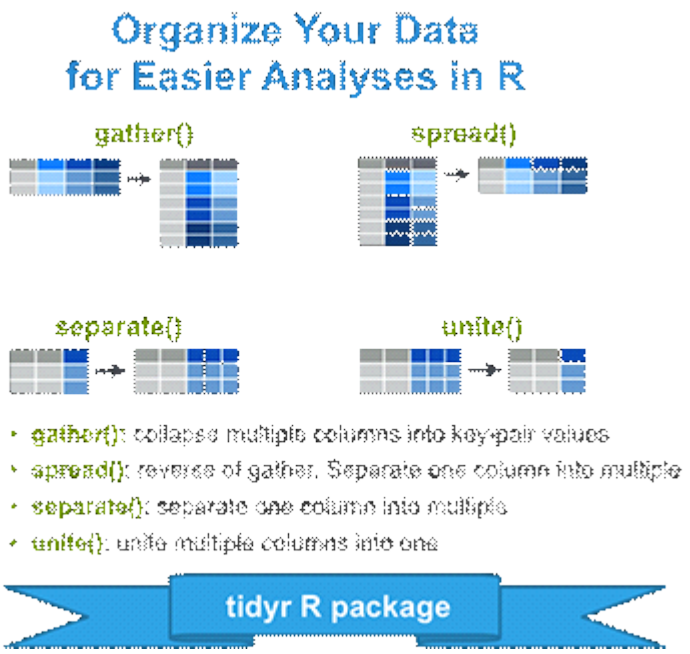
Will read in as factors, unless you include stringsAsFactors=FALSE
Also reads in as a DF not a tibble, use read_csv if you want a tibble

Reading Data from the internet

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/extdata/murders.csv"
dat <- read_csv(url)
download.file(url, "murders.csv")
tempfile()

tmp_filename <- tempfile()
download.file(url, tmp_filename) #download file and give it a temporary name
dat <- read_csv(tmp_filename) #read it in
file.remove(tmp_filename) #erase files that were downloaded
```

Tidy Date - Reshaping Data

Saturday, March 7, 2020 12:16 PM



In tidy data, each row represents an observation and each column represents a different variable.
In wide data, each row includes several observations and one of the variables is stored in the header.
Code

```
library(tidyverse)
library(dslabs)
data(gapminder)

# create and inspect a tidy data frame
tidy_data <- gapminder %>%
  filter(country %in% c("South Korea", "Germany")) %>%
  select(country, year, fertility)
head(tidy_data)

# plotting tidy data is simple
tidy_data %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_point()

# import and inspect example of original Gapminder data in wide format
path <- system.file("extdata", package="dslabs")
filename <- file.path(path,  "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)
select(wide_data, country, `1960`:`1967`)
```

RESHAPING DATA

Key points

- The **tidyr** package includes several functions that are useful for tidying data.
- The gather() function converts wide data into tidy data.
- The spread() function converts tidy data to wide data.

Code

```
# original wide data
library(tidyverse)
path <- system.file("extdata", package="dslabs")
filename <- file.path(path,  "fertility-two-countries-example.csv")
wide_data <- read_csv(filename)

# tidy data from dslabs
library(dslabs)
data("gapminder")
tidy_data <- gapminder %>%
filter(country %in% c("South Korea", "Germany")) %>%
select(country, year, fertility)

# gather wide data to make new tidy data
new_tidy_data <- wide_data %>%
gather(year, fertility, `1960`:`2015`)
head(new_tidy_data)

# gather all columns except country
new_tidy_data <- wide_data %>%
gather(year, fertility, -country)

# gather treats column names as characters by default
class(tidy_data$year)
class(new_tidy_data$year)

# convert gathered column names to numeric
new_tidy_data <- wide_data %>%
gather(year, fertility, -country, convert = TRUE)
class(new_tidy_data$year)

# ggplot works on new tidy data
new_tidy_data %>%
ggplot(aes(year, fertility, color = country)) +
geom_point(

# spread tidy data to generate wide data
new_wide_data <- new_tidy_data %>% spread(year, fertility)
select(new_wide_data, country, `1960`:`1967`)
```

Separate and Unite

Key points

- The separate() function splits one column into two or more columns at a specified character that separates the variables.
- When there is an extra separation in some of the entries, use fill="right" to pad missing values with NAs, or use extra="merge" to keep extra elements together.
- The unite() function combines two columns and adds a separating character.

Code

```
# import data
path <- system.file("extdata", package = "dslabs")
filename <- file.path(path, "life-expectancy-and-fertility-two-countries-example.csv")
```

```
raw_dat <- read_csv(filename)
select(raw_dat, 1:5)
# gather all columns except country
dat <- raw_dat %>% gather(key, value, -country)
head(dat)
dat$key[1:5]
# separate on underscores
dat %>% separate(key, c("year", "variable_name"), "_")
dat %>% separate(key, c("year", "variable_name"))
# split on all underscores, pad empty cells with NA
dat %>% separate(key, c("year", "first_variable_name", "second_variable_name"),
fill = "right")
# split on first underscore but keep life_expectancy merged
dat %>% separate(key, c("year", "variable_name"), sep = "_", extra = "merge")


# separate then spread
dat %>% separate(key, c("year", "variable_name"), sep = "_", extra = "merge") %>%
spread(variable_name, value)
# separate then unite
dat %>%
separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
unite(variable_name, first_variable_name, second_variable_name, sep="_")
# full code for tidying data
dat %>%
separate(key, c("year", "first_variable_name", "second_variable_name"), fill = "right") %>%
unite(variable_name, first_variable_name, second_variable_name, sep="_") %>%
spread(variable_name, value) %>%
rename(fertility = fertility_NA)
```

EXAMPLES:

You have a dataset on U.S. contagious diseases, but it is in the following wide format:

```
> head(dat_wide)
state year population HepatitisA Mumps Polio Rubella
Alabama 1990   4040587    86       19   76   1
Alabama 1991   4066003    39       14   65   0
Alabama 1992   4097169    35       12   24   0
Alabama 1993   4133242    40       22   67   0
Alabama 1994   4173361    72       12   39   0
Alabama 1995   4216645    75    2  38   0
```

You want to transform this into a tidy dataset, with each row representing an observation of the incidence of each specific disease (as shown below):

```
> head(dat_tidy)
state   year  population  disease  count
Alabama 1990    4040587 HepatitisA        86
Alabama 1991    4066003 HepatitisA        39
Alabama 1992    4097169 HepatitisA        35
Alabama 1993    4133242 HepatitisA        40
Alabama 1994    4173361 HepatitisA        72
Alabama 1995    4216645 HepatitisA        75
```

```
dat_tidy <- dat_wide %>%
  gather(key = disease, value = count, HepatitisA:Rubella)
```

You have successfully formatted marathon finish times into a tidy object called `tidy_data`. The first few lines are shown below.

```
age_group year  time
20        2015  03:46
30        2015  03:50
40        2015  04:39
50        2015  04:48
20        2016  03:22
```

Select the code that converts these data back to the wide format, where each year has a separate column.

```
tidy_data %>% spread(year, time)
```

You have the following dataset:

```
> head(dat)
state   abb region             var   people
Alabama  AL  South population 4779736
Alabama  AL  South            total      135
Alaska   AK   West population  710231
Alaska   AK   West            total       19
Arizona  AZ   West population 6392017
Arizona  AZ   West            total      232
```

You would like to transform it into a dataset where population and total are each their own column (shown below):

```
state      abb region population total
Alabama     AL  South       4779736   135
Alaska      AK   West        710231    19
Arizona     AZ   West       6392017   232
Arkansas    AR  South       2915918    93
California  CA   West  37253956  1257
Colorado    CO   West       5029196    65
```

```
dat_tidy <- dat %>% spread(key = var, value = people)
```

```
age_group,2015_time,2015_participants,2016_time,2016_participants
20,3:46,54,3:22,62
30,3:50,60,3:43,58
40,4:39,29,3:49,33
50,4:48,10,4:59,14
```

You read in the data file:

```
d <- read_csv("times.csv")
```

Which best makes the data tidy?

```
tidy_data <- d %>%
  gather(key = “key”, value = “value”, -age_group) %>%
```

```
  separate(col = key, into = c("year", "variable_name"), sep = "_") %>%
  spread(key = variable_name, value = value)
```

You are in the process of tidying some data on heights, hand length, and wingspan for basketball players in the draft. Currently, you have the following:

```
> head(stats)
key            value
allen_height     75
allen_hand_length 8.25
allen_wingspan     79.25
bamba_height     83.25
bamba_hand_length 9.75
bamba_wingspan    94
```

"tidy" format with columns "height", "hand_length" and "wingspan".

```
tidy_data <- stats %>%
  separate(col = key, into = c("player", "variable_name"), sep = "_", extra = "merge") %>%
  spread(key = variable_name, value = value)
```

```
co2_wide <- data.frame(matrix(co2, ncol = 12, byrow = TRUE)) %>%
    setNames(1:12) %>%
  mutate(year = as.character(1959:1997))
```

Use the `gather()` function to make this dataset tidy. Call the column with the CO2 measurements `co2` and call the month column `month`. Name the resulting object `co2_tidy`.

```
co2_tidy <- gather(co2_wide,month,co2,-year)
```

Graph this data

```
co2_tidy %>% ggplot(aes(as.numeric(month), co2, color = year)) + geom_line()
```

Original data

	major	gender	admitted
1	A	men	62
2	B	men	63
3	C	men	37
4	D	men	33
5	E	men	28
6	F	men	6
7	A	women	82
8	B	women	68
9	C	women	34
10	D	women	35
11	E	women	24
12	F	women	7

Your goal is to get the data in the shape that has one row for each major, like this:

major	men	women
A	62	82
B	63	68
C	37	34
D	33	35

```
E   28  24
F    6   7
```

dat_tidy <- spread(dat, gender, admitted)

```
   major gender        key value
1      A    men   admitted    62
2      B    men   admitted    63
3      C    men   admitted    37
4      D    men   admitted    33
5      E    men   admitted    28
6      F    men   admitted     6
7      A  women   admitted    82
8      B  women   admitted    68
9      C  women   admitted    34
10     D  women   admitted    35
11     E  women   admitted    24
12     F  women   admitted     7
13     A    men applicants   825
14     B    men applicants   560
15     C    men applicants   325
16     D    men applicants   417
17     E    men applicants   191
18     F    men applicants   373
19     A  women applicants   108
20     B  women applicants    25
21     C  women applicants   593
22     D  women applicants   375
23     E  women applicants   393
24     F  women applicants   341
```

Combine the key and gender and create a new column called `column_name` to get a variable with the following values: `admitted_men`, `admitted_women`, `applicants_men` and `applicants_women`. Save the new data as `tmp2`.

```
   major      column_name value
1      A     admitted_men    62
2      B     admitted_men    63
3      C     admitted_men    37
4      D     admitted_men    33
5      E     admitted_men    28
6      F     admitted_men     6
7      A   admitted_women    82
8      B   admitted_women    68
9      C   admitted_women    34
10     D   admitted_women    35
11     E   admitted_women    24
12     F   admitted_women     7
13     A   applicants_men   825
14     B   applicants_men   560
15     C   applicants_men   325
16     D   applicants_men   417
17     E   applicants_men   191
18     F   applicants_men   373
19     A applicants_women   108
20     B applicants_women    25
21     C applicants_women   593
22     D applicants_women   375
```

```
23      E applicants_women   393
24      F applicants_women   341
```

tmp2 <- unite(tmp, column_name, c(key, gender))

Combining Tables

Saturday, March 7, 2020 1:57 PM

Key points

- The join functions in the **dplyr** package combine two tables such that matching rows are together.
- left_join() only keeps rows that have information in the first table.
- right_join() only keeps rows that have information in the second table.
- inner_join() only keeps rows that have information in both tables.
- full_join() keeps all rows from both tables.
- semi_join() keeps the part of first table for which we have information in the second.
- anti_join() keeps the elements of the first table for which there is no information in the second.

Code

```
# import US murders data
library(tidyverse)
library(ggrepel)
library(dslabs)
ds_theme_set()
data(murders)
head(murders)
# import US election results data
data(polls_us_election_2016)
head(results_us_election_2016)
identical(results_us_election_2016$state, murders$state)
# join the murders table and US election results table
tab <- left_join(murders, results_us_election_2016, by = "state")
head(tab)
# plot electoral votes versus population
tab %>% ggplot(aes(population/10^6, electoral_votes, label = abb)) +
geom_point() +
geom_text_repel() +
scale_x_continuous(trans = "log2") +
scale_y_continuous(trans = "log2") +
geom_smooth(method = "lm", se = FALSE)
# make two smaller tables to demonstrate joins
tab1 <- slice(murders, 1:6) %>% select(state, population)
tab1
tab2 <- slice(results_us_election_2016, c(1:3, 5, 7:8)) %>% select(state, electoral_votes)
tab2
# experiment with different joins
left_join(tab1, tab2)
tab1 %>% left_join(tab2)
tab1 %>% right_join(tab2)
inner_join(tab1, tab2)
semi_join(tab1, tab2)
anti_join(tab1, tab2)
```

BINDING

Key points

- Unlike the join functions, the binding functions do not try to match by a variable, but rather just combine datasets.
- bind_cols() binds two objects by making them columns in a tibble. The R-base function cbind() binds columns but makes a data frame or matrix instead.

- The bind_rows() function is similar but binds rows instead of columns. The R-base function rbind() binds rows but makes a data frame or matrix instead.
  Code

```
bind_cols(a = 1:3, b = 4:6)
tab1 <- tab[, 1:3]
tab2 <- tab[, 4:6]
tab3 <- tab[, 7:9]
new_tab <- bind_cols(tab1, tab2, tab3)
head(new_tab)
tab1 <- tab[1:2,]
tab2 <- tab[3:4,]
bind_rows(tab1, tab2)
```

Set Operators

Key points

- By default, the set operators in R-base work on vectors. If **tidyverse/dplyr** are loaded, they also work on data frames.
- You can take intersections of vectors using intersect(). This returns the elements common to both sets.
- You can take the union of vectors using union(). This returns the elements that are in either set.
- The set difference between a first and second argument can be obtained with setdiff(). Note that this function is not symmetric.
- The function set_equal() tells us if two sets are the same, regardless of the order of elements.
  Code

```
# intersect vectors or data frames
intersect(1:10, 6:15)
intersect(c("a","b","c"), c("b","c","d"))
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
intersect(tab1, tab2)
# perform a union of vectors or data frames
union(1:10, 6:15)
union(c("a","b","c"), c("b","c","d"))
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
union(tab1, tab2)
# set difference of vectors or data frames
setdiff(1:10, 6:15)
setdiff(6:15, 1:10)
tab1 <- tab[1:5,]
tab2 <- tab[3:7,]
setdiff(tab1, tab2)
# setequal determines whether sets have the same elements, regardless of order (returns either TRUE or FALSE)
If false it will inidcate what rows are in one but not the other table
If setequal(1:5, 1:6)
setequal(1:5, 5:1)
setequal(tab1, tab2)
```

Web Scraping

Saturday, March 7, 2020 5:02 PM

- Web scraping is extracting data from a website.
- The **rvest** web harvesting package includes functions to extract nodes of an HTML document: html_nodes() extracts all nodes of different types, and html_node() extracts the first node.
- html_table() converts an HTML table to a data frame.

```
# import a webpage into R
library(rvest)
url <- "https://en.wikipedia.org/wiki/Murder_in_the_United_States_by_state"
h <- read_html(url)
class(h)
h

tab <- h %>% html_nodes("table")
tab <- tab[[2]]

tab <- tab %>% html_table
class(tab)

tab <- tab %>% setNames(c("state", "population", "total", "murders", "gun_murders", "gun_ownership",
"total_rate", "murder_rate", "gun_murder_rate"))
head(tab)
```

For the guacamole recipe page, we already have done this and determined that we need the following selectors:

```
h <- read_html("http://www.foodnetwork.com/recipes/alton-brown/guacamole-recipe-1940609")
recipe <- h %>% html_node(".o-AssetTitle__a-HeadlineText") %>% html_text()
prep_time <- h %>% html_node(".m-RecipeInfo__a-Description--Total") %>% html_text()
ingredients <- h %>% html_nodes(".o-Ingredients__a-Ingredient") %>% html_text()
```

You can see how complex the selectors are. In any case we are now ready to extract what we want and create a list:

```
guacamole <- list(recipe, prep_time, ingredients)
guacamole
```

Since recipe pages from this website follow this general layout, we can use this code to create a function that extracts this information:

```
get_recipe <- function(url){
  h <- read_html(url)
  recipe <- h %>% html_node(".o-AssetTitle__a-HeadlineText") %>% html_text()
  prep_time <- h %>% html_node(".m-RecipeInfo__a-Description--Total") %>% html_text()
  ingredients <- h %>% html_nodes(".o-Ingredients__a-Ingredient") %>% html_text()
  return(list(recipe = recipe, prep_time = prep_time, ingredients = ingredients))
}
```

and then use it on any of their webpages:

```
get_recipe("http://www.foodnetwork.com/recipes/food-network-kitchen/pancakes-recipe-1913844")
```

There are several other powerful tools provided by **rvest**. For example, the functions html_form(), set_values(), and submit_form() permit you to query a webpage from R. This is a more advanced topic not covered here.

String Processing

Saturday, March 7, 2020 5:14 PM

- **Remove** unwanted characters from text.
- **Extract numeric** values from text.
- **Find** and **replace** characters.
- **Extract specific** parts of strings.
- **Convert** free form text into more uniform formats.
- **Split strings** into multiple values.
- Use **regular expressions (regex)** to process strings.

STRING PARSING

Key points

- The most common tasks in string processing include:
- extracting numbers from strings
- removing unwanted characters from text
- finding and replacing characters
- extracting specific parts of strings
- converting free form text to more uniform formats
- splitting strings into multiple values
- The **stringr** package in the **tidyverse** contains string processing functions that follow a similar naming format (str_functionname) and are compatible with the pipe.

Code

```
# read in raw murders data from Wikipedia
url <- "https://en.wikipedia.org/w/index.php?
title=Gun_violence_in_the_United_States_by_state&direction=prev&oldid=810166167"
murders_raw <- read_html(url) %>%
html_nodes("table") %>%
html_table() %>%
.[[1]] %>%
setNames(c("state", "population", "total", "murder_rate"))
# inspect data and column classes
head(murders_raw)
class(murders_raw$population)
class(murders_raw$total)
```

Defining Strings: Single and Double Quotes and How to Escape

Key points

- Define a string by surrounding text with either single quotes or double quotes.
- To include a single quote inside a string, use double quotes on the outside. To include a double quote inside a string, use single quotes on the outside.
- The cat() function displays a string as it is represented inside R.
- To include a double quote inside of a string surrounded by double quotes, use the backslash (\) to escape the double quote. Escape a single quote to include it inside of a string defined by single quotes.
- We will see additional uses of the escape later.

Code

```
s <- "Hello!"    # double quotes define a string
```

```
s <- 'Hello!'    # single quotes define a string
s <- `Hello`    # backquotes do not
s <- "10"'    # error - unclosed quotes
s <- '10"'    # correct
# cat shows what the string actually looks like inside R
cat(s)
s <- "5'"
cat(s)
# to include both single and double quotes in string, escape with \
s <- '5'10"'    # error
s <- "5'10""    # error
s <- '5\'10"'    # correct
cat(s)
s <- "5'10\""    # correct
cat(s)
```

Defining Strings: Single and Double Quotes and How to Escape

Key points

- The main types of string processing tasks are detecting, locating, extracting and replacing elements of strings.
- The **stringr** package from the **tidyverse** includes a variety of string processing functions that begin with str_ and take the string as the first argument, which makes them compatible with the pipe.

Code

```
# murders_raw defined in web scraping video
# direct conversion to numeric fails because of commas
murders_raw$population[1:3]
as.numeric(murders_raw$population[1:3])
library(tidyverse)    # includes stringr
```

Case Study

Key points

- Use the str_detect() function to determine whether a string contains a certain pattern.
- Use the str_replace_all() function to replace all instances of one pattern with another pattern. To remove a pattern, replace with the empty string ("").
- The parse_number() function removes punctuation from strings and converts them to numeric.
- mutate_at() performs the same transformation on the specified column numbers.

Code

```
# murders_raw was defined in the web scraping section
# detect whether there are commas
commas <- function(x) any(str_detect(x, ","))
murders_raw %>% summarize_all(funs(commas))
# replace commas with the empty string and convert to numeric
test_1 <- str_replace_all(murders_raw$population, ",", "")
test_1 <- as.numeric(test_1)
# parse_number also removes commas and converts to numeric
test_2 <- parse_number(murders_raw$population)
identical(test_1, test_2)
murders_new <- murders_raw %>% mutate_at(2:3, parse_number)
murders_new %>% head
```

Examples

```
> head(dat)
# A tibble: 5 x 3
```

Month	Sales	Profit
<chr>	<chr>	<chr>
January	$128,568	$16,234
February	$109,523	$12,876
March	$115,468	$17,920
April	$122,274	$15,825
May	$117,921	$15,437

Which of the following commands could convert the sales and profits columns to numeric? Select all that apply.

dat %>% mutate_at(2:3, as.numeric)

String Processing Part 2

Saturday, March 7, 2020 5:28 PM

Key points

- In the raw heights data, many students did not report their height as the number of inches as requested. There are many entries with real height information but in the wrong format, which we can extract with string processing.
- When there are both text and numeric entries in a column, the column will be a character vector. Converting this column to numeric will result in NAs for some entries.
- To correct problematic entries, look for patterns that are shared across large numbers of entries, then define rules that identify those patterns and use these rules to write string processing tasks.
- Use suppressWarnings() to hide warning messages for a function.

Code

```
# load raw heights data and inspect
library(dslabs)
data(reported_heights)
class(reported_heights$height)
# convert to numeric, inspect, count NAs
x <- as.numeric(reported_heights$height)
head(x)
sum(is.na(x))
# keep only entries that result in NAs
reported_heights %>% mutate(new_height = as.numeric(height)) %>%
filter(is.na(new_height)) %>%
head(n=10)
# calculate cutoffs that cover 99.999% of human population
alpha <- 1/10^6
qnorm(1-alpha/2, 69.1, 2.9)
qnorm(alpha/2, 63.7, 2.7)
# keep only entries that either result in NAs or are outside the plausible range of heights
not_inches <- function(x, smallest = 50, tallest = 84){
inches <- suppressWarnings(as.numeric(x))
ind <- is.na(inches) | inches < smallest | inches > tallest
ind
}
# number of problematic entries
problems <- reported_heights %>%
filter(not_inches(height)) %>%
.$height
length(problems)
# 10 examples of x'y or x'y" or x'y\"
pattern <- "^\\d\\s*'\\s*\\d{1,2}\\.*\\d*'*\"*$"
str_subset(problems, pattern) %>% head(n=10) %>% cat
# 10 examples of x.y or x,y
pattern <- "^[4-6]\\s*[\\.|,]\\s*([0-9]|10|11)$"
str_subset(problems, pattern) %>% head(n=10) %>% cat
# 10 examples of entries in cm rather than inches
ind <- which(between(suppressWarnings(as.numeric(problems))/2.54, 54, 81) )
ind <- ind[!is.na(ind)]
problems[ind] %>% head(n=10) %>% cat
```

Regular Expressions (Regex)

Saturday, March 7, 2020 5:31 PM

Key points

- A regular expression (regex) is a way to describe a specific pattern of characters of text. A set of rules has been designed to do this specifically and efficiently.
- **stringr** functions can take a regex as a pattern.
- str_detect() indicates whether a pattern is present in a string.
- The main difference between a regex and a regular string is that a regex can include special characters.
- The | symbol inside a regex means "or".
- Use '\\d' to represent digits. The backlash is used to distinguish it from the character 'd'. In R, you must use two backslashes for digits in regular expressions; in some other languages, you will only use one backslash for regex special characters.
- str_view() highlights the first occurrence of a pattern, and the str_view_all() function highlights all occurrences of the pattern.

Code

```
# load stringr through tidyverse
library(tidyverse)
# detect whether a comma is present
pattern <- ","
str_detect(murders_raw$total, pattern)
# show the subset of strings including "cm"
str_subset(reported_heights$height, "cm")
# use the "or" symbol inside a regex (|)
yes <- c("180 cm", "70 inches")
no <- c("180", "70'"")
s <- c(yes, no)
str_detect(s, "cm") | str_detect(s, "inches")
str_detect(s, "cm|inches")
//d means contains digits

# highlight the first occurrence of a pattern
str_view(s, pattern)
# highlight all instances of a pattern
str_view_all(s, pattern)
```

Character Classes, Anchors and Quantifiers

Saturday, March 7, 2020 5:34 PM

Key points

- Define strings to test your regular expressions, including some elements that match and some that do not. This allows you to check for the two types of errors: failing to match and matching incorrectly.
- Square brackets define character classes: groups of characters that count as matching the pattern. You can use ranges to define character classes, such as [0-9] for digits and [a-zA-Z] for all letters.
- Anchors define patterns that must start or end at specific places. ^ and $ represent the beginning and end of the string respectively.
- Curly braces are quantifiers that state how many times a certain character can be repeated in the pattern. \\d{1,2} matches exactly 1 or 2 consecutive digits.

Code

```
# s was defined in the previous video
yes <- c("5", "6", "5'10", "5 feet", "4'11")
no <- c("", ".", "Five", "six")
s <- c(yes, no)
pattern <- "\\d"
# [56] means 5 or 6
str_view(s, "[56]")
# [4-7] means 4, 5, 6 or 7
yes <- as.character(4:7)
no <- as.character(1:3)
s <- c(yes, no)
str_detect(s, "[4-7]")
Can do the same for letters
str_detect(s, "[a-z]")

# ^ means start of string, $ means end of string
pattern <- "^\\d$"   #start of the string followed by one digit followed by the end of the string
yes <- c("1", "5", "9")
no <- c("12", "123", " 1", "a4", "b")
s <- c(yes, no)
str_view(s, pattern)
# curly braces define quantifiers: 1 or 2 digits
pattern <- "^\\d{1,2}$"   # find all of the numbers that are 1 digit or 2 digits
yes <- c("1", "5", "9", "12")
no <- c("123", "a4", "b")
str_view(c(yes, no), pattern)
# combining character class, anchors and quantifier
pattern <- "^[4-7]'\\d{1,2}\"$"  #starts the string, then an number between 4 and 7, ' symbol,  then one or two
digits, " symbol (\")
yes <- c("5'7\"", "6'2\"",  "5'12\"")
no <- c("6,2\"", "6.2\"","I am 5'11\"", "3'2\"", "64")
str_detect(yes, pattern)
str_detect(no, pattern)
```

Regex Search and Replace

Saturday, March 7, 2020 5:44 PM

Key points

- str_replace() replaces the first instance of the detected pattern with a specified string.
- Spaces are characters and R does not ignore them. Spaces are specified by the special character \\s.
- Additional quantifiers include *, + and ?. * means 0 or more instances of the previous character. ? means 0 or 1 instances. + means 1 or more instances.
- Before removing characters from strings with functions like str_replace() and str_replace_all(), consider whether that replacement would have unintended effects.

Code

The problems object is defined in the reported heights case study introduction video.

```
# number of entries matching our desired pattern
pattern <- "^[4-7]'\\d{1,2}\"$"
sum(str_detect(problems, pattern))
# inspect examples of entries with problems
problems[c(2, 10, 11, 12, 15)] %>% str_view(pattern)
str_subset(problems, "inches")
str_subset(problems, "'''")
# replace or remove feet/inches words before matching
pattern <- "^[4-7]'\\d{1,2}$"
problems %>%
str_replace("feet|ft|foot", "'") %>% # replace feet, ft, foot with '
 str_replace("inches|in|''|\"", "") %>% # remove all inches symbols
 str_detect(pattern) %>%
sum()
# R does not ignore whitespace
identical("Hi", "Hi ")
# \\s represents whitespace
pattern_2 <- "^[4-7]'\\s\\d{1,2}\"$"
str_subset(problems, pattern_2)
# * means 0 or more instances of a character
yes <- c("AB", "A1B", "A11B", "A111B", "A1111B")
no <- c("A2B", "A21B")
str_detect(yes, "A1*B")
str_detect(no, "A1*B")
# test how *, ? and + differ
data.frame(string = c("AB", "A1B", "A11B", "A111B", "A1111B"),
           none_or_more = str_detect(yes, "A1*B"),
           nore_or_once = str_detect(yes, "A1?B"),
           once_or_more = str_detect(yes, "A1+B"))

# update pattern by adding optional spaces before and after feet symbol
pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
problems %>%
 str_replace("feet|ft|foot", "'") %>% # replace feet, ft, foot with '
 str_replace("inches|in|''|\"", "") %>% # remove all inches symbols
 str_detect(pattern) %>%
 sum()
```

Groups with Regex

Saturday, March 7, 2020 7:14 PM

Key Points

- Groups are defined using parentheses.
- Once we define groups, we can use the function str_match() to extract the values these groups define. str_extract() extracts only strings that match a pattern, not the values defined by groups.
- You can refer to the ith group with \\i. For example, refer to the value in the second group with \\2.

Code

```
# define regex with and without groups
pattern_without_groups <- "^[4-7],\\d*$" #none or more digits
pattern_with_groups <-  "^([4-7]),(\\d*)$"
# create examples
yes <- c("5,9", "5,11", "6,", "6,1")
no <- c("5'9", ",", "2,8", "6.1.1")
s <- c(yes, no)
# demonstrate the effect of groups
str_detect(s, pattern_without_groups)
str_detect(s, pattern_with_groups)
# demonstrate difference between str_match and str_extract
str_match(s, pattern_with_groups)
str_extract(s, pattern_with_groups)
# improve the pattern to recognize more events
pattern_with_groups <-  "^([4-7]),(\\d*)$"
yes <- c("5,9", "5,11", "6,", "6,1")
no <- c("5'9", ",", "2,8", "6.1.1")
s <- c(yes, no)
str_replace(s, pattern_with_groups, "\\1'\\2")
# final pattern
pattern_with_groups <-"^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$"
^ start of string
[4-7] first digit 4-7
\\s* none or more white spaces
[,\\.\\s+] feet symbol is either comma, dot or white space
\\s* none or more white spaces
\\d* none or more digits

# combine stringr commands with the pipe
str_subset(problems, pattern_with_groups) %>% head
str_subset(problems, pattern_with_groups) %>%
str_replace(pattern_with_groups, "\\1'\\2") %>% head
```

Examples

```
animals <- c("cat", "puppy", "Moose", "MONKEY")
pattern <- "[a-z]"
str_detect(animals, pattern)
1]  TRUE  TRUE  TRUE FALSE


animals <- c("cat", "puppy", "Moose", "MONKEY")
pattern <- "[A-Z]$"
str_detect(animals, pattern)
```

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

```
animals <- c("cat", "puppy", "Moose", "MONKEY")
pattern <- "[a-z]{4,5}"
str_detect(animals, pattern)
[1] FALSE  TRUE  TRUE FALSE
```

```
animals <- c("moose", "monkey", "meerkat", "mountain lion")
pattern <- “mo*”
pattern <- “mo?”
> str_detect(animals, pattern)
[1] TRUE TRUE TRUE TRUE
```

```
> schools
[1] "U. Kentucky"              "Univ New Hampshire"       "Univ. of Massachusetts"     "University Georgia"
[5] "U California"             "California State University"
```

You want to clean this data to match the full names of each university:

```
> final
[1] "University of Kentucky"     "University of New Hampshire" "University of Massachusetts" "University of
Georgia"
[5] "University of California"   "California State University"
```

```
schools %>%
  str_replace("^Univ\\.?\\s|^U\\.?\\s", "University ") %>%
  str_replace("^University of |^University ", "University of ")
```

Testing and Improving Strings

Saturday, March 7, 2020 7:21 PM

Key points

- Wrangling with regular expressions is often an iterative process of testing the approach, looking for problematic entries, and improving the patterns.
- Use the pipe to connect **stringr** functions.
- It may not be worth writing code to correct every unique problem in the data, but string processing techniques are flexible enough for most needs.

Code

```
# function to detect entries with problems
not_inches_or_cm <- function(x, smallest = 50, tallest = 84){
inches <- suppressWarnings(as.numeric(x))
ind <- !is.na(inches) &
((inches >= smallest & inches <= tallest) |
(inches/2.54 >= smallest & inches/2.54 <= tallest))
!ind
}
# identify entries with problems
problems <- reported_heights %>%
filter(not_inches_or_cm(height)) %>%
.$height
length(problems)
converted <- problems %>%
str_replace("feet|foot|ft", "'") %>% #convert feet symbols to '
  str_replace("inches|in|''|\"", "") %>%  #remove inches symbols
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2") ##change format
# find proportion of entries that fit the pattern after reformatting
pattern <- "^[4-7]\\s*'\\s*\\d{1,2}$"
index <- str_detect(converted, pattern)
mean(index)

converted[!index]    # show problems

>
```

Separate with Regex - extract()

Saturday, March 7, 2020 7:43 PM

Key Point

- The extract() function behaves similarly to the separate() function but allows extraction of groups from regular expressions.

Code

```
# first example - normally formatted heights
s <- c("5'10", "6'1")
tab <- data.frame(x = s)
# the separate and extract functions behave similarly
tab %>% separate(x, c("feet", "inches"), sep = "'")
tab %>% extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")
# second example - some heights with unusual formats
s <- c("5'10", "6'1\"","5'8inches")
tab <- data.frame(x = s)
# separate fails because it leaves in extra characters, but extract keeps only the digits because of regex groups
tab %>% separate(x, c("feet","inches"), sep = "'", fill = "right")
tab %>% extract(x, c("feet", "inches"), regex = "(\\d)'(\\d{1,2})")

>
```

String Groups and Quantifiers

Saturday, March 7, 2020 7:45 PM

Four clear patterns of entries have arisen along with some other minor problems:

1. Many students measuring exactly 5 or 6 feet did not enter any inches. For example, **6'** - our pattern requires that inches be included.
2. Some students measuring exactly 5 or 6 feet entered just that number.
3. Some of the inches were entered with decimal points. For example **5'7.5''**. Our pattern only looks for two digits.
4. Some entires have spaces at the end, for example **5 ' 9**.
5. Some entries are in meters and some of these use European decimals: **1.6, 1,7**.
6. Two students added **cm**.
7. One student spelled out the numbers: **Five foot eight inches.**

Case 1

For case 1, if we add a '0 to, for example, convert all 6 to 6'0, then our pattern will match. This can be done using groups using the following code:

```
yes <- c("5", "6", "5")
no <- c("5'", "5''", "5'4")
s <- c(yes, no)
str_replace(s, "^([4-7])$", "\\1'0")
```

The pattern says it has to start (^), be followed with a digit between 4 and 7, and then end there ($). The parenthesis defines the group that we pass as **\\1** to the replace regex.

Cases 2 and 4

We can adapt this code slightly to handle case 2 as well which covers the entry **5'**. Note that the **5'** is left untouched by the code above. This is because the extra ' makes the pattern not match since we have to end with a 5 or 6. To handle case 2, we want to permit the 5 or 6 to be followed by no or one symbol for feet. So we can simply add **'{0,1}** after the ' to do this. We can also use the none or once special character **?**. As we saw previously, this is different from * which is none or more. We now see that this code also handles the fourth case as well:

```
str_replace(s, "^([56])'?$", "\\1'0")
```

Note that here we only permit 5 and 6 but not 4 and 7. This is because heights of exactly 5 and exactly 6 feet tall are quite common, so we assume those that typed 5 or 6 really meant either 60 or 72 inches. However, heights of exactly 4 or exactly 7 feet tall are so rare that, although we accept 84 as a valid entry, we assume that a 7 was entered in error.

Case 3

We can use quantifiers to deal with case 3. These entries are not matched because the inches include decimals and our pattern does not permit this. We need allow the second group to include decimals and not just digits. This means we must permit zero or one period **.** followed by zero or more digits. So we will use both **?** and *. Also remember that for this particular case, the period needs to be escaped since it is a special character (it means any character except a line break).

So we can adapt our pattern, currently **^[4-7]\\s*'\\s*\\d{1,2}$**, to permit a decimal at the end:

```
pattern <- "^[4-7]\\s*'\\s*(\\d+\\.?\\d*)$"
```

Case 5

Case 5, meters using commas, we can approach similarly to how we converted the x.y to x'y. A difference is that we require that the first digit is 1 or 2:

```
yes <- c("1,7", "1, 8", "2, " )
no <- c("5,8", "5,3,2", "1.7")
s <- c(yes, no)
```

```
str_replace(s, "^([12])\\s*,\\s*(\\d*)$", "\\1\\.\\2")
```
We will later check if the entries are meters using their numeric values.

Trimming

In general, spaces at the start or end of the string are uninformative. These can be particularly deceptive because sometimes they can be hard to see:

```
s <- "Hi "
cat(s)
identical(s, "Hi")
```
This is a general enough problem that there is a function dedicated to removing them: str_trim.
```
str_trim("5 ' 9 ")
```
To upper and to lower case

One of the entries writes out numbers as words: **Five foot eight inches**. Although not efficient, we could add 12 extra **str_replace** to convert **zero** to **0**, **one** to **1**, and so on. To avoid having to write two separate operations for **Zero** and **zero**, **One** and **one**, etc., we can use the str_to_lower() function to make all words lower case first:

```
s <- c("Five feet eight inches")
str_to_lower(s)
```
Putting it into a function

We are now ready to define a procedure that handles converting all the problematic cases.
We can now put all this together into a function that takes a string vector and tries to convert as many strings as possible to a single format. Below is a function that puts together the previous code replacements:

```
convert_format <- function(s){
 s %>%
  str_replace("feet|foot|ft", "'") %>% #convert feet symbols to '
  str_replace_all("inches|in|''|\"|cm|and", "") %>%  #remove inches and other symbols
  str_replace("^([4-7])\\s*[,\\.\\s+]\\s*(\\d*)$", "\\1'\\2") %>% #change x.y, x,y x y
  str_replace("^([56])'?$", "\\1'0") %>% #add 0 when to 5 or 6
  str_replace("^([12])\\s*,\\s*(\\d*)$", "\\1\\.\\2") %>% #change european decimal
  str_trim() #remove extra space
}
```
We can also write a function that converts words to numbers:

```
words_to_numbers <- function(s){
 str_to_lower(s) %>%
  str_replace_all("zero", "0") %>%
  str_replace_all("one", "1") %>%
  str_replace_all("two", "2") %>%
  str_replace_all("three", "3") %>%
  str_replace_all("four", "4") %>%
  str_replace_all("five", "5") %>%
  str_replace_all("six", "6") %>%
  str_replace_all("seven", "7") %>%
  str_replace_all("eight", "8") %>%
  str_replace_all("nine", "9") %>%
  str_replace_all("ten", "10") %>%
  str_replace_all("eleven", "11")
}
```
Now we can see which problematic entries remain:

```
converted <- problems %>% words_to_numbers %>% convert_format
remaining_problems <- converted[not_inches_or_cm(converted)]
pattern <- "^[4-7]\\s*'\\s*\\d+\\.?\\d*$"
index <- str_detect(remaining_problems, pattern)
remaining_problems[!index]
```

We are now ready to put everything we've done so far together and wrangle our reported heights data as we try to recover as many heights as possible. The code is complex but we will break it down into parts.

We start by cleaning up the height column so that the heights are closer to a feet'inches format. We added an original heights column so we can compare before and after.

Let's start by writing a function that cleans up strings so that all the feet and inches formats use the same x'y format when appropriate.

```
pattern <- "^([4-7])\\s*'\\s*(\\d+\\.?\\d*)$"
smallest <- 50
tallest <- 84
new_heights <- reported_heights %>%
 mutate(original = height,
        height = words_to_numbers(height) %>% convert_format()) %>%
 extract(height, c("feet", "inches"), regex = pattern, remove = FALSE) %>%
 mutate_at(c("height", "feet", "inches"), as.numeric) %>%
 mutate(guess = 12*feet + inches) %>%
 mutate(height = case_when(
   !is.na(height) & between(height, smallest, tallest) ~ height, #inches
   !is.na(height) & between(height/2.54, smallest, tallest) ~ height/2.54, #centimeters
   !is.na(height) & between(height*100/2.54, smallest, tallest) ~ height*100/2.54, #meters
   !is.na(guess) & inches < 12 & between(guess, smallest, tallest) ~ guess, #feet'inches
   TRUE ~ as.numeric(NA))) %>%
 select(-guess)
```

We can check all the entries we converted using the following code:

```
new_heights %>%
 filter(not_inches(original)) %>%
 select(original, height) %>%
 arrange(height) %>%
 View()
```

Let's take a look at the shortest students in our dataset using the following code:

```
new_heights %>% arrange(height) %>% head(n=7)
```

We see heights of 53, 54, and 55. In the original heights column, we also have 51 and 52. These short heights are very rare and it is likely that the students actually meant 5'1, 5'2, 5'3, 5'4, and 5'5. But because we are not completely sure, we will leave them as reported.

String Splitting

Saturday, March 7, 2020 7:46 PM

Key Points

- The function str_split() splits a string into a character vector on a delimiter (such as a comma, space or underscore). By default, str_split() generates a list with one element for each original string. Use the function argument simplify=TRUE to have str_split() return a matrix instead.
- The map() function from the **purrr** package applies the same function to each element of a list. To extract the ith entry of each element x, use map(x, i).
- map() always returns a list. Use map_chr() to return a character vector and map_int() to return an integer.

Code

```
# read raw murders data line by line
filename <- system.file("extdata/murders.csv", package = "dslabs")
lines <- readLines(filename)
lines %>% head()
# split at commas with str_split function, remove row of column names
x <- str_split(lines, ",")
x %>% head()
col_names <- x[[1]]
x <- x[-1]
# extract first element of each list entry
library(purrr)
map(x, function(y) y[1]) %>% head()
map(x, 1) %>% head()
# extract columns 1-5 as characters, then convert to proper format - NOTE: DIFFERENT FROM VIDEO
dat <- data.frame(parse_guess(map_chr(x, 1)),
                  parse_guess(map_chr(x, 2)),
                  parse_guess(map_chr(x, 3)),
                  parse_guess(map_chr(x, 4)),
                  parse_guess(map_chr(x, 5))) %>%
 setNames(col_names)

dat %>% head

# more efficient code for the same thing
dat <- x %>%
transpose() %>%
map( ~ parse_guess(unlist(.))) %>%
setNames(col_names) %>%
as.data.frame()
# the simplify argument makes str_split return a matrix instead of a list
x <- str_split(lines, ",", simplify = TRUE)
col_names <- x[1,]
x <- x[-1,]
x %>% as_data_frame() %>%
setNames(col_names) %>%
mutate_all(parse_guess)
```

EXAMPLE:

```
>schedule
day             staff
Monday          Mandy, Chris and Laura
```

```
Tuesday              Steve, Ruth and Frank
```

You want to turn this into a more useful data frame.

Which two commands would properly split the text in the “staff” column into each individual name? Select ALL that apply.

```
str_split(schedule$staff, ", | and ")
```

OR

```
str_split(schedule$staff, ",\\s|\\sand\\s")
```

```
> schedule
day       staff
Monday            Mandy, Chris and Laura
Tuesday Steve, Ruth and Frank
```

What code would successfully turn your “Schedule” table into the following tidy table?

```
> tidy
day     staff
<chr>   <chr>
Monday  Mandy
Monday  Chris
Monday  Laura
Tuesday Steve
Tuesday Ruth
Tuesday Frank
```

```
tidy <- schedule %>%
  mutate(staff = str_split(staff, ", | and ")) %>%
  unnest()
```

Extracting a Table from a PDF

Saturday, March 7, 2020 7:51 PM

One of the datasets provided in dslabs shows scientific funding rates by gender in the Netherlands:

```
library(dslabs)
data("research_funding_rates")
research_funding_rates
```

The data come from a paper published in the prestigious journal PNAS. However, the data are not provided in a spreadsheet; they are in a table in a PDF document. We could extract the numbers by hand, but this could lead to human error. Instead we can try to wrangle the data using R.

Downloading the data

We start by downloading the PDF document then importing it into R using the following code:

```
library("pdftools")
temp_file <- tempfile()
url <- "http://www.pnas.org/content/suppl/2015/09/16/1510159112.DCSupplemental/pnas.201510159SI.pdf"
download.file(url, temp_file)
txt <- pdf_text(temp_file)
file.remove(temp_file)
```

If we examine the object text we notice that it is a character vector with an entry for each page. So we keep the page we want using the following code:

```
raw_data_research_funding_rates <- txt[2]
```

The steps above can actually be skipped because we include the raw data in the dslabs package as well:

```
data("raw_data_research_funding_rates")
```

Looking at the download

Examining this object,

```
raw_data_research_funding_rates %>% head
```

we see that it is a long string. Each line on the page, including the table rows, is separated by the symbol for newline: \n.

We can therefore can create a list with the lines of the text as elements:

```
tab <- str_split(raw_data_research_funding_rates, "\n")
```

Because we start off with just one element in the string, we end up with a list with just one entry:

```
tab <- tab[[1]]
```

By examining this object,

```
tab %>% head
```

we see that the information for the column names is the third and fourth entires:

```
the_names_1 <- tab[3]
the_names_2 <- tab[4]
```

In the table, the column information is spread across two lines. We want to create one vector with one name for each column. We can do this using some of the functions we have just learned.

Extracting the table data

Let's start with the first line:

```
the_names_1
```

We want to remove the leading space and everything following the comma. We can use regex for the latter. Then we can obtain the elements by splitting using the space. We want to split only when there are 2 or more spaces to avoid splitting success rate. So we use the regex \\s{2,} as follows:

```
the_names_1 <- the_names_1 %>%
  str_trim() %>%
  str_replace_all(",\\s.", "") %>%
```

```
  str_split("\\s{2,}", simplify = TRUE)
the_names_1
```
Now let's look at the second line:
```
the_names_2
```

Here we want to trim the leading space and then split by space as we did for the first line:
```
the_names_2 <- the_names_2 %>%
  str_trim() %>%
  str_split("\\s+", simplify = TRUE)
the_names_2
```

Now we can join these to generate one name for each column:
```
tmp_names <- str_c(rep(the_names_1, each = 3), the_names_2[-1], sep = "_")
the_names <- c(the_names_2[1], tmp_names) %>%
  str_to_lower() %>%
  str_replace_all("\\s", "_")
the_names
```

Now we are ready to get the actual data. By examining the `tab` object, we notice that the information is in lines 6 through 14. We can use `str_split()` again to achieve our goal:
```
new_research_funding_rates <- tab[6:14] %>%
  str_trim %>%
  str_split("\\s{2,}", simplify = TRUE) %>%
  data.frame(stringsAsFactors = FALSE) %>%
  setNames(the_names) %>%
  mutate_at(-1, parse_number)
new_research_funding_rates %>% head()
```
We can see that the objects are identical:
```
identical(research_funding_rates, new_research_funding_rates)
```

Recoding

Saturday, March 7, 2020 7:52 PM

Key points

- Change long factor names with the recode() function from the **tidyverse**.
- Other similar functions include recode_factor() and fct_recoder() in the **forcats** package in the **tidyverse**. The same result could be obtained using the case_when() function, but recode() is more efficient to write.

Code

```
# life expectancy time series for Caribbean countries
library(dslabs)
data("gapminder")
gapminder %>%
filter(region=="Caribbean") %>%
ggplot(aes(year, life_expectancy, color = country)) +
geom_line()
# display long country names
gapminder %>%
filter(region=="Caribbean") %>%
filter(str_length(country) >= 12) %>%
distinct(country)
# recode long country names and remake plot
gapminder %>% filter(region=="Caribbean") %>%
mutate(country = recode(country,
'Antigua and Barbuda'="Barbuda",
'Dominican Republic' = "DR",
'St. Vincent and the Grenadines' = "St. Vincent",
'Trinidad and Tobago' = "Trinidad")) %>%
ggplot(aes(year, life_expectancy, color = country)) +
geom_line()
```

EXAMPLE

The remain and leave columns are both given in the format "48.1%": percentages out of 100% with a percent symbol.

Which of these commands converts the remain vector to a proportion between 0 and 1?

```
as.numeric(str_replace(polls$remain, "%", ""))/100
parse_number(polls$remain)/100
```

Changes "48.1%" to a decical between 0 and 1

Dates and Times

Saturday, March 7, 2020 8:22 PM

Key points

- Dates are a separate data type in R.The **tidyverse** includes functionality for dealing with dates through the **lubridate** package.
- Extract the year, month and day from a date object with the year(), month() and day() functions.
- Parsers convert strings into dates with the standard YYYY-MM-DD format (ISO 8601 format). Use the parser with the name corresponding to the string format of year, month and day (ymd(), ydm(), myd(), mdy(), dmy(), dym()).
- Get the current time with the Sys.time() function. Use the now() function instead to specify a time zone.
- You can extract values from time objects with the hour(), minute() and second() functions.
- Parsers convert strings into times (for example, hms()). Parsers can also create combined date-time objects (for example, mdy_hms()).

Code

```
# inspect the startdate column of 2016 polls data, a Date type
library(tidyverse)
library(dslabs)
data("polls_us_election_2016")
polls_us_election_2016$startdate %>% head
class(polls_us_election_2016$startdate)
as.numeric(polls_us_election_2016$startdate) %>% head
# ggplot is aware of dates
polls_us_election_2016 %>% filter(pollster == "Ipsos" & state =="U.S.") %>%
ggplot(aes(startdate, rawpoll_trump)) +
geom_line()
# lubridate: the tidyverse date package
library(lubridate)
# select some random dates from polls
set.seed(2)
dates <- sample(polls_us_election_2016$startdate, 10) %>% sort
dates
# extract month, day, year from date strings
data.frame(date = dates,
month = month(dates),
day = day(dates),
year = year(dates))
month(dates, label = TRUE)    # extract month label
# ymd works on mixed date styles
x <- c(20090101, "2009-01-02", "2009 01 03", "2009-1-4",
"2009-1, 5", "Created on 2009 1 6", "200901 !!! 07")
ymd(x)
# different parsers extract year, month and day in different orders
x <- "09/01/02"
ymd(x)
mdy(x)
ydm(x)
myd(x)
dmy(x)
dym(x)
now()    # current time in your time zone
now("GMT")    # current time in GMT
now() %>% hour()    # current hour
now() %>% minute()    # current minute
now() %>% second()    # current second
```

```r
# parse time
x <- c("12:34:56")
hms(x)
#parse datetime
x <- "Nov/2/2012 12:34:56"
mdy_hms(x)
```

Text Mining

Saturday, March 7, 2020 8:26 PM

Key points
The tidytext package helps us convert free form text into a tidy table.
Use unnest_tokens() to extract individual words and other meaningful chunks of text.
Sentiment analysis assigns emotions or a positive/negative score to tokens. You can extract sentiments using get_sentiments(). Common lexicons for sentiment analysis are "bing", "afinn", "nrc" and "loughran".
With the exception of labels used to represent categorical data, we have focused on numerical data, but in many applications data starts as text. Well known examples are spam filtering, cyber-crime prevention, counter-terrorism and sentiment analysis.

In all these examples, the raw data is composed of free form texts. Our task is to extract insights from these data. In this section, we learn how to generate useful numerical summaries from text data to which we can apply some of the powerful data visualization and analysis techniques we have learned.

Case study: Trump Tweets
During the 2016 US presidential election, then-candidate Donald J. Trump used his Twitter account as a way to communicate with potential voters. On August 6, 2016 Todd Vaziri tweeted about Trump that "Every non-hyperbolic tweet is from iPhone (his staff). Every hyperbolic tweet is from Android (from him)." Data scientist David Robinson conducted an analysis to determine if data supported this assertion. Here we go through David's analysis to learn some of the basics of text mining. To learn more about text mining in R we recommend this book.

We will use the following libraries

```
library(tidyverse)
library(ggplot2)
library(lubridate)
library(tidyr)
library(scales)
set.seed(1)
```

In general, we can extract data directly from Twitter using the rtweet package. However, in this case, a group has already compiled data for us and made it available at http://www.trumptwitterarchive.com.

```
url <- 'http://www.trumptwitterarchive.com/data/realdonaldtrump/%s.json'
trump_tweets <- map(2009:2017, ~sprintf(url, .x)) %>%
  map_df(jsonlite::fromJSON, simplifyDataFrame = TRUE) %>%
  filter(!is_retweet & !str_detect(text, '^"')) %>%
  mutate(created_at = parse_date_time(created_at, orders = "a b! d! H!:M!:S! z!* Y!", tz="EST"))
```

For convenience we include the result of the code above in the dslabs package:

```
library(dslabs)
data("trump_tweets")
```

This is data frame with information about the tweet:

```
head(trump_tweets)
```

The variables that are included are:

```
names(trump_tweets)
```

The help file ?trump_tweets provides details on what each variable represents. The tweets are

represented by the text variable:

```
trump_tweets %>% select(text) %>% head
```
and the source variable tells us the device that was used to compose and upload each tweet:

```
trump_tweets %>% count(source) %>% arrange(desc(n))
```
We can use extract to remove the Twitter for part of the source and filter out retweets.

```
trump_tweets %>%
  extract(source, "source", "Twitter for (.*)") %>%
  count(source)
```
We are interested in what happened during the campaign, so for the analysis here we will focus on what was tweeted between the day Trump announced his campaign and election day. So we define the following table:

```
campaign_tweets <- trump_tweets %>%
  extract(source, "source", "Twitter for (.*)") %>%
  filter(source %in% c("Android", "iPhone") &
         created_at >= ymd("2015-06-17") &
         created_at < ymd("2016-11-08")) %>%
  filter(!is_retweet) %>%
  arrange(created_at)
```
We can now use data visualization to explore the possibility that two different groups were tweeting from these devices. For each tweet, we will extract the hour, in the east coast (EST), it was tweeted then compute the proportion of tweets tweeted at each hour for each device.

```
ds_theme_set()
campaign_tweets %>%
  mutate(hour = hour(with_tz(created_at, "EST"))) %>%
  count(source, hour) %>%
  group_by(source) %>%
  mutate(percent = n / sum(n)) %>%
  ungroup %>%
  ggplot(aes(hour, percent, color = source)) +
  geom_line() +
  geom_point() +
  scale_y_continuous(labels = percent_format()) +
  labs(x = "Hour of day (EST)",
       y = "% of tweets",
       color = "")
```
We notice a big peak for the Android in early hours of the morning, between 6 and 8 AM. There seems to be a clear different in these patterns. We will therefore assume that two different entities are using these two devices. Now we will study how their tweets differ. To do this we introduce the tidytext package.

Text as data

The tidytext package helps us convert free from text into a tidy table. Having the data in this format greatly facilitates data visualization and applying statistical techniques.

```
library(tidytext)
```
The main function needed to achieve this is unnest_tokens(). A token refers to the units that we are considering to be a data point. The most common tokens will be words, but they can also be single characters, ngrams, sentences, lines or a pattern defined by a regex. The functions will take a vector of

strings and extract the tokens so that each one gets a row in the new table. Here is a simple example:

```
example <- data_frame(line = c(1, 2, 3, 4),
                      text = c("Roses are red,", "Violets are blue,", "Sugar is sweet,", "And so are you."))
example
example %>% unnest_tokens(word, text)
```

Now let's look at a quick example with a tweet number 3008:

```
i <- 3008
campaign_tweets$text[i]
campaign_tweets[i,] %>%
 unnest_tokens(word, text) %>%
 select(word)
```

Note that the function tries to convert tokens into words and strips characters important to twitter such as # and @. A token in twitter is not the same as in regular English. For this reason, instead of using the default token, words, we define a regex that captures twitter character. The pattern appears complex but all we are defining is a patter that starts with @, # or neither and is followed by any combination of letters or digits:

```
pattern <- "([^A-Za-z\\d#@']|'(?![A-Za-z\\d#@]))"
```

We can now use the unnest_tokens() function with the regex option and appropriately extract the hashtags and mentions:

```
campaign_tweets[i,] %>%
 unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
 select(word)
```

Another minor adjustment we want to make is remove the links to pictures:

```
campaign_tweets[i,] %>%
 mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&amp;", "")) %>%
 unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
 select(word)
```

Now we are ready to extract the words for all our tweets.

```
tweet_words <- campaign_tweets %>%
 mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&amp;", "")) %>%
 unnest_tokens(word, text, token = "regex", pattern = pattern)
```

And we can now answer questions such as "what are the most commonly used words?"

```
tweet_words %>%
 count(word) %>%
 arrange(desc(n))
```

It is not surprising that these are the top words. The top words are not informative. The tidytext package has database of these commonly used words, referred to as stop words, in text mining:

```
stop_words
```

If we filter out rows representing stop words with filter(!word %in% stop_words$word):

```
tweet_words <- campaign_tweets %>%
 mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&amp;", "")) %>%
 unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
 filter(!word %in% stop_words$word )
```

We end up with a much more informative set of top 10 tweeted words:

```
tweet_words %>%
  count(word) %>%
  top_n(10, n) %>%
  mutate(word = reorder(word, n)) %>%
  arrange(desc(n))
```

Some exploration of the resulting words (not show here) reveals a couple of unwanted characteristics in our tokens. First, some of our tokens are just numbers (years for example). We want to remove these and we can find them using the regex ^\d+$. Second, some of our tokens come from a quote and they start with '. We want to remove the ' when it's at the start of a word, so we will use str_replace(). We add these two lines to the code above to generate our final table:

```
tweet_words <- campaign_tweets %>%
  mutate(text = str_replace_all(text, "https://t.co/[A-Za-z\\d]+|&amp;", ""))  %>%
  unnest_tokens(word, text, token = "regex", pattern = pattern) %>%
  filter(!word %in% stop_words$word &
         !str_detect(word, "^\\d+$")) %>%
  mutate(word = str_replace(word, "^'", ""))
```

Now that we have all our words in a table, along with information about what device was used to compose the tweet they came from, we can start exploring which words are more common when comparing Android to iPhone.

For each word we want to know if it is more likely to come from an Android tweet or an iPhone tweet. We previously introduced the odds ratio, a summary statistic useful for quantifying these differences. For each device and a given word, let's call it y, we compute the odds or the ratio between the proportion of words that are y and not y and compute the ratio of those odds. Here we will have many proportions that are 0 so we use the 0.5 correction.

```
android_iphone_or <- tweet_words %>%
  count(word, source) %>%
  spread(source, n, fill = 0) %>%
  mutate(or = (Android + 0.5) / (sum(Android) - Android + 0.5) /
         ( (iPhone + 0.5) / (sum(iPhone) - iPhone + 0.5)))
android_iphone_or %>% arrange(desc(or))
android_iphone_or %>% arrange(or)
```

Given that several of these words are overall low frequency words we can impose a filter based on the total frequency like this:

```
android_iphone_or %>% filter(Android+iPhone > 100) %>%
  arrange(desc(or))

android_iphone_or %>% filter(Android+iPhone > 100) %>%
  arrange(or)
```

We already see somewhat of a pattern in the types of words that are being tweeted more in one device versus the other. However, we are not interested in specific words but rather in the tone. Vaziri's assertion is that the Android tweets are more hyperbolic. So how can we check this with data? Hyperbolic is a hard sentiment to extract from words as it relies on interpreting phrases. However, words can be associated to more basic sentiment such as as anger, fear, joy and surprise. In the next section we demonstrate basic sentiment analysis.

Sentiment Analysis

In sentiment analysis we assign a word to one or more "sentiment". Although this approach will miss context dependent sentiments, such as sarcasm, when performed on large numbers of words,

summaries can provide insights.

The first step in sentiment analysis is to assign a sentiment to each word. The tidytext package includes several maps or lexicons in the object sentiments:

sentiments
There are several lexicons in the tidytext package that give different sentiments. For example, the bing lexicon divides words into positive and negative. We can see this using the tidytext function get_sentiments():

get_sentiments("bing")
The AFINN lexicon assigns a score between -5 and 5, with -5 the most negative and 5 the most positive.

get_sentiments("afinn")
The loughran and nrc lexicons provide several different sentiments:

get_sentiments("loughran") %>% count(sentiment)
get_sentiments("nrc") %>% count(sentiment)
To start learning about how these lexicons were developed, read this help file: ?sentiments.

For the analysis here we are interested in exploring the different sentiments of each tweet, so we will use the nrc lexicon:

```
nrc <- get_sentiments("nrc") %>%
  select(word, sentiment)
```
We can combine the words and sentiments using inner_join(), which will only keep words associated with a sentiment. Here are 10 random words extracted from the tweets:

```
tweet_words %>% inner_join(nrc, by = "word") %>%
  select(source, word, sentiment) %>% sample_n(10)
```
Now we are ready to perform a quantitative analysis comparing Android and iPhone by comparing the sentiments of the tweets posted from each device. Here we could perform a tweet by tweet analysis, assigning a sentiment to each tweet. However, this somewhat complex since each tweet will have several sentiments attached to it, one for each word appearing in the lexicon. For illustrative purposes, we will perform a much simpler analysis: we will count and compare the frequencies of each sentiment appears for each device.

```
sentiment_counts <- tweet_words %>%
  left_join(nrc, by = "word") %>%
  count(source, sentiment) %>%
  spread(source, n) %>%
  mutate(sentiment = replace_na(sentiment, replace = "none"))
sentiment_counts
```
Because more words were used on the Android than on the phone:

```
tweet_words %>% group_by(source) %>% summarize(n = n())
```
for each sentiment we can compute the odds of being in the device: proportion of words with sentiment versus proportion of words without and then compute the odds ratio comparing the two devices:

```
sentiment_counts %>%
  mutate(Android = Android / (sum(Android) - Android) ,
        iPhone = iPhone / (sum(iPhone) - iPhone),
        or = Android/iPhone) %>%
```

```
  arrange(desc(or))
```
So we do see some difference and the order is interesting: the largest three sentiments are disgust, anger, and negative! But are they statistically significant? How does this compare if we are just assigning sentiments at random?

To answer that question we can compute, for each sentiment, an odds ratio and confidence interval. We will add the two values we need to form a two-by-two table and the odds ratio:

```
library(broom)
log_or <- sentiment_counts %>%
  mutate( log_or = log( (Android / (sum(Android) - Android)) / (iPhone / (sum(iPhone) - iPhone))),
        se = sqrt( 1/Android + 1/(sum(Android) - Android) + 1/iPhone + 1/(sum(iPhone) - iPhone)),
        conf.low = log_or - qnorm(0.975)*se,
        conf.high = log_or + qnorm(0.975)*se) %>%
  arrange(desc(log_or))

log_or
```
A graphical visualization shows some sentiments that are clearly overrepresented:

```
log_or %>%
  mutate(sentiment = reorder(sentiment, log_or),) %>%
  ggplot(aes(x = sentiment, ymin = conf.low, ymax = conf.high)) +
  geom_errorbar() +
  geom_point(aes(sentiment, log_or)) +
  ylab("Log odds ratio for association between Android and sentiment") +
  coord_flip()
```
We see that the disgust, anger, negative sadness and fear sentiments are associated with the Android in a way that is hard to explain by chance alone. Words not associated to a sentiment were strongly associated with the iPhone source, which is in agreement with the original claim about hyperbolic tweets.

If we are interested in exploring which specific words are driving these differences, we can back to our android_iphone_or object:

```
android_iphone_or %>% inner_join(nrc) %>%
  filter(sentiment == "disgust" & Android + iPhone > 10) %>%
  arrange(desc(or))
```
We can make a graph:

```
android_iphone_or %>% inner_join(nrc, by = "word") %>%
  mutate(sentiment = factor(sentiment, levels = log_or$sentiment)) %>%
  mutate(log_or = log(or)) %>%
  filter(Android + iPhone > 10 & abs(log_or)>1) %>%
  mutate(word = reorder(word, log_or)) %>%
  ggplot(aes(word, log_or, fill = log_or < 0)) +
  facet_wrap(~sentiment, scales = "free_x", nrow = 2) +
  geom_bar(stat="identity", show.legend = FALSE) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

Linear Regression - Correlation

Sunday, March 8, 2020 7:34 PM

Key points

- Galton tried to predict sons' heights based on fathers' heights.
- The mean and standard errors are insufficient for describing an important characteristic of the data: the trend that the taller the father, the taller the son.
- The correlation coefficient is an informative summary of how two variables move together that can be used to predict one variable using the other.

Code

```
# create the dataset
library(tidyverse)
library(HistData)
data("GaltonFamilies")
set.seed(1983)
galton_heights <- GaltonFamilies %>%
  filter(gender == "male") %>%
  group_by(family) %>%
  sample_n(1) %>%
  ungroup() %>%
  select(father, childHeight) %>%
  rename(son = childHeight)
# means and standard deviations
galton_heights %>%
   summarize(mean(father), sd(father), mean(son), sd(son))
# scatterplot of father and son heights
galton_heights %>%
   ggplot(aes(father, son)) +
   geom_point(alpha = 0.5)
```

Correlation Coefficient

Key points

- The correlation coefficient is defined for a list of pairs $(x_1, y_1), \ldots, (x_n, y_n)$ as the product of the standardized values: $\left(\frac{x_i - \mu_x}{\sigma_x}\right)\left(\frac{y_i - \mu_y}{\sigma_y}\right)$.
- The correlation coefficient essentially conveys how two variables move together.
- The correlation coefficient is always between -1 and 1.

Code (when is gets closer to 1 or -1 the scatter plot points get thinner - closer together like a dense line

```
rho <- mean(scale(x)*scale(y))
galton_heights %>% summarize(r = cor(father, son)) %>% pull®
```

Sample Correlation is a Random Variable

Key points

- The correlation that we compute and use as a summary is a random variable.
- When interpreting correlations, it is important to remember that correlations derived from samples are estimates containing uncertainty.
- Because the sample correlation is an average of independent draws, the central limit theorem applies.

Code

```
# compute sample correlation
R <- sample_n(galton_heights, 25, replace = TRUE) %>%
   summarize(r = cor(father, son))
R
```

```
# Monte Carlo simulation to show distribution of sample correlation
B <- 1000
N <- 25
R <- replicate(B, {
sample_n(galton_heights, N, replace = TRUE) %>%
summarize(r = cor(father, son)) %>%
pull(r)
})
qplot(R, geom = "histogram", binwidth = 0.05, color = I("black"))
# expected value and standard error
mean(R)
sd(R)
# QQ-plot to evaluate whether N is large enough
data.frame(R) %>%
ggplot(aes(sample = R)) +
stat_qq() +
geom_abline(intercept = mean(R), slope = sqrt((1-mean(R)^2)/(N-2)))
```

Anscombe's Quartet/Stratification

Key points

- Correlation is not always a good summary of the relationship between two variables.
- The general idea of conditional expectation is that we stratify a population into groups and compute summaries in each group.
- A practical way to improve the estimates of the conditional expectations is to define strata of with similar values of x.
- If there is perfect correlation, the regression line predicts an increase that is the same number of SDs for both variables. If there is 0 correlation, then we don't use x at all for the prediction and simply predict the average μy. For values between 0 and 1, the prediction is somewhere in between. If the correlation is negative, we predict a reduction instead of an increase.

Code

```
# number of fathers with height 72 or 72.5 inches
sum(galton_heights$father == 72)
sum(galton_heights$father == 72.5)
# predicted height of a son with a 72 inch tall father
conditional_avg <- galton_heights %>%
filter(round(father) == 72) %>%
summarize(avg = mean(son)) %>%
pull(avg)
conditional_avg
# stratify fathers' heights to make a boxplot of son heights
galton_heights %>% mutate(father_strata = factor(round(father))) %>%
ggplot(aes(father_strata, son)) +
geom_boxplot() +
geom_point()
# center of each boxplot
galton_heights %>%
mutate(father = round(father)) %>%
group_by(father) %>%
summarize(son_conditional_avg = mean(son)) %>%
ggplot(aes(father, son_conditional_avg)) +
geom_point()
# calculate values to plot regression line on original data
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m <- r * s_y/s_x
b <- mu_y - m*mu_x
```

```
# add regression line to plot
galton_heights %>%
ggplot(aes(father, son)) +
geom_point(alpha = 0.5) +
geom_abline(intercept = b, slope = m)
```

Example:

```
> lm(son ~ father, data = galton_heights)
Call:
lm(formula = son ~ father, data = galton_heights)
Coefficients:
(Intercept)    father
   35.71       0.50
```

For every inch we increase the father's height, the predicted son's height grows by 0.5 inches.

Baseball Data

Sunday, March 8, 2020 7:22 PM

Code: Scatterplot of the relationship between HRs and wins

```
library(Lahman)
library(tidyverse)
library(dslabs)
ds_theme_set()
Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR_per_game = HR / G, R_per_game = R / G) %>%
  ggplot(aes(HR_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```

Code: Scatterplot of the relationship between stolen bases and wins

```
Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(SB_per_game = SB / G, R_per_game = R / G) %>%
  ggplot(aes(SB_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```

Code: Scatterplot of the relationship between bases on balls and runs

```
Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(BB_per_game = BB / G, R_per_game = R / G) %>%
  ggplot(aes(BB_per_game, R_per_game)) +
  geom_point(alpha = 0.5)
```

Linear Models

Saturday, March 14, 2020 5:30 PM

```
# linear model for multiple variables, predict R_per_game based off of both BB_per_game & HR_per_game
#data will return intercept and the slope of the model
lm(formula = R_per_game ~ BB_per_game + HR_per_game, data = data)

#_____
_____

# Predictions with confidence intervals
galton_heights %>% ggplot(aes(father, son)) +
  geom_point() +
  geom_smooth()

OR

# Predictions with confidence intervals
model <- lm(son ~ father, data = galton_heights)
predictions <- predict(model, interval = c("confidence"), level = 0.95)
data <- as.tibble(predictions) %>% bind_cols(father = galton_heights$father)

ggplot(data, aes(x = father, y = fit)) +
  geom_line(color = "blue", size = 1) +
  geom_ribbon(aes(ymin=lwr, ymax=upr), alpha=0.2) +
  geom_point(data = galton_heights, aes(x = father, y = son))
#_____
#Predicted height compared to variable2
#can use fit, se.fit. df, or residual.scale = TRUE depending on desired output
x <- lm(formula = mother ~ daughter, data = female_heights)
df <- predict(x, df = TRUE)

#create linear model using  historical average singles to predict current singles
lm(formula = singles ~ mean_singles , data = final)

#create linear model using  historical average singles to predict current singles
lm(formula = singles ~ mean_singles , data = final)



fit <- Teams %>%
   filter(yearID %in% 1961:2001) %>%
   mutate(avg_attendance = attendance/G,
          rpg = R / G,
          HRpg = HR/G)%>%
   lm(avg_attendance ~ rpg+ HRpg+ W+yearID, data = .)

_____
 # using linear model named "fit" to predict avg attendance based on preset variables
   predict(fit, data.frame(rpg = 5, HRpg = 1.2, W = 80, yearID = 1960))
```

```
#using linear model to predict values and then comparing the prediction to an actual
 LM <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(avg_attendance = attendance/G,
         rpg = R / G,
         HRpg = HR/G)%>%
  lm(avg_attendance ~ rpg+ HRpg+ W+yearID, data = .)


 Teams %>%
    filter(yearID == 2002) %>%
  mutate(avg_attendance = attendance/G,
         rpg = R / G,
         HRpg = HR/G)%>%
   mutate(prediction = predict(LM, newdata=.))
```

```
 # plot RSS as a function of beta1 when beta0=25, Least Squared Estimate is point at the lowest
curve
 beta1 = seq(0, 1, len=nrow(galton_heights))
 results <- data.frame(beta1 = beta1,
                       rss = sapply(beta1, rss, beta0 = 36))
 results %>% ggplot(aes(beta1, rss)) + geom_line() +
  geom_line(aes(beta1, rss))



 B <- 1000
 N <- 50
 lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
    lm(son ~ father, data = .) %>%
    .$coef
 })
 lse <- data.frame(beta_0 = lse[1,], beta_1 = lse[2,])

 # Plot the distribution of beta_0 and beta_1
 library(gridExtra)
 p1 <- lse %>% ggplot(aes(beta_0)) + geom_histogram(binwidth = 5, color = "black")
 p2 <- lse %>% ggplot(aes(beta_1)) + geom_histogram(binwidth = 0.1, color = "black")
 grid.arrange(p1, p2, ncol = 2)

 # summary statistics
 sample_n(galton_heights, N, replace = TRUE) %>%
  lm(son ~ father, data = .) %>%
  summary %>%
  .$coef

 lse %>% summarize(se_0 = sd(beta_0), se_1 = sd(beta_1))
```

```
# plot predictions and confidence intervals
galton_heights %>% ggplot(aes(son, father)) +
  geom_point() +
  geom_smooth(method = "lm")

# predict Y directly
fit <- galton_heights %>% lm(son ~ father, data = .)
Y_hat <- predict(fit, se.fit = TRUE)
names(Y_hat)

# plot best fit line
galton_heights %>%
  mutate(Y_hat = predict(lm(son ~ father, data=.))) %>%
  ggplot(aes(father, Y_hat))+
  geom_line()
```

Bivariate Normal Distribution

Friday, March 13, 2020 6:11 PM

Key points

- When a pair of random variables are approximated by the bivariate normal distribution, scatterplots look like ovals. They can be thin (high correlation) or circle-shaped (no correlation).
- When two variables follow a bivariate normal distribution, computing the regression line is equivalent to computing conditional expectations.
- We can obtain a much more stable estimate of the conditional expectation by finding the regression line and using it to make predictions.

Code

```
galton_heights %>%
mutate(z_father = round((father - mean(father)) / sd(father))) %>%
filter(z_father %in% -2:2) %>%
ggplot() +
stat_qq(aes(sample = son)) +
facet_wrap( ~ z_father)
```

Variance Explained

Key points

- Conditioning on a random variable X can help to reduce variance of response variable Y.
- The standard deviation of the conditional distribution is $\mathrm{SD}\left(Y \mid X = x\right) = \sigma_y \sqrt{1-\rho^2}$, which is smaller than the standard deviation without conditioning σ_y.
- Because variance is the standard deviation squared, the variance of the conditional distribution is $\sigma_y^2 \left(1-\rho^2\right)$.
- In the statement "X explains such and such percent of the variability," the percent value refers to the variance. The variance decreases by ρ^2 percent.
- The "variance explained" statement only makes sense when the data is approximated by a bivariate normal distribution.

Conditional Expected Value

```
m = r * s_y/s_x
b = mu_y - (r * s_y/s_x)*mu_x
x = 60
m*x+b
```

Two Regression Lines

Friday, March 13, 2020 6:17 PM

Key point

There are two different regression lines depending on whether we are taking the expectation of Y given X or taking the expectation of X given Y.

Code

```
# compute a regression line to predict the son's height from the father's height
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m_1 <-  r * s_y / s_x
b_1 <- mu_y - m_1*mu_x
# compute a regression line to predict the father's height from the son's height
m_2 <-  r * s_x / s_y
b_2 <- mu_x - m_2*mu_y
```

Confounding

Saturday, March 14, 2020 11:11 AM

Key points

- Association is not causation!
- Although it may appear that BB cause runs, it is actually the HR that cause most of these runs. We say that BB are **confounded** with HR.
- Regression can help us account for confounding.

Code

```
# find regression line for predicting runs from BBs
library(tidyverse)
library(Lahman)
bb_slope <- Teams %>%
filter(yearID %in% 1961:2001 ) %>%
mutate(BB_per_game = BB/G, R_per_game = R/G) %>%
lm(R_per_game ~ BB_per_game, data = .) %>%
.$coef %>%
.[2]
bb_slope
# compute regression line for predicting runs from singles
singles_slope <- Teams %>%
 filter(yearID %in% 1961:2001 ) %>%
 mutate(Singles_per_game = (H-HR-X2B-X3B)/G, R_per_game = R/G) %>%
 lm(R_per_game ~ Singles_per_game, data = .) %>%
 .$coef  %>%
 .[2]
singles_slope
# calculate correlation between HR, BB and singles
Teams %>%
 filter(yearID %in% 1961:2001 ) %>%
 mutate(Singles = (H-HR-X2B-X3B)/G, BB = BB/G, HR = HR/G) %>%
 summarize(cor(BB, HR), cor(Singles, HR), cor(BB,Singles))

>
```

Stratification and Multivariate Regression

Saturday, March 14, 2020 11:12 AM

Key points

- A first approach to check confounding is to keep HRs fixed at a certain value and then examine the relationship between BB and runs.
- The slopes of BB after stratifying on HR are reduced, but they are not 0, which indicates that BB are helpful for producing runs, just not as much as previously thought.

Code

```
# stratify HR per game to nearest 10, filter out strata with few points
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
 mutate(HR_strata = round(HR/G, 1),
        BB_per_game = BB / G,
        R_per_game = R / G) %>%
 filter(HR_strata >= 0.4 & HR_strata <=1.2)

# scatterplot for each HR stratum
dat %>%
 ggplot(aes(BB_per_game, R_per_game)) +
 geom_point(alpha = 0.5) +
 geom_smooth(method = "lm") +
 facet_wrap( ~ HR_strata)

# calculate slope of regression line after stratifying by HR
dat %>%
 group_by(HR_strata) %>%
 summarize(slope = cor(BB_per_game, R_per_game)*sd(R_per_game)/sd(BB_per_game))

# stratify by BB
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
 mutate(BB_strata = round(BB/G, 1),
        HR_per_game = HR / G,
        R_per_game = R / G) %>%
 filter(BB_strata >= 2.8 & BB_strata <=3.9)
# scatterplot for each BB stratum
dat %>% ggplot(aes(HR_per_game, R_per_game)) +
 geom_point(alpha = 0.5) +
 geom_smooth(method = "lm") +
 facet_wrap( ~ BB_strata)

# slope of regression line after stratifying by BB
dat %>%
 group_by(BB_strata) %>%
 summarize(slope = cor(HR_per_game, R_per_game)*sd(R_per_game)/sd(HR_per_game))
```

Least Squares Estimates

Saturday, March 14, 2020 11:45 AM

Key points

- For regression, we aim to find the coefficient values that minimize the distance of the fitted model to the data.
- Residual sum of squares (RSS) measures the distance between the true value and the predicted value given by the regression line. The values that minimize the RSS are called the least squares estimates (LSE).
- We can use partial derivatives to get the values for β_0 and β_1 in Galton's data.

Code

```
# compute RSS for any pair of beta0 and beta1 in Galton's data
library(HistData)
data("GaltonFamilies")
set.seed(1983)
galton_heights <- GaltonFamilies %>%
 filter(gender == "male") %>%
 group_by(family) %>%
 sample_n(1) %>%
 ungroup() %>%
 select(father, childHeight) %>%
 rename(son = childHeight)
rss <- function(beta0, beta1, data){
   resid <- galton_heights$son - (beta0+beta1*galton_heights$father)
   return(sum(resid^2))
}
# plot RSS as a function of beta1 when beta0=25
beta1 = seq(0, 1, len=nrow(galton_heights))
results <- data.frame(beta1 = beta1,
                      rss = sapply(beta1, rss, beta0 = 25))
results %>% ggplot(aes(beta1, rss)) + geom_line() +
 geom_line(aes(beta1, rss))
```

The lm funtion (LM)

Key points

- When calling the lm() function, the variable that we want to predict is put to the left of the ~ symbol, and the variables that we use to predict is put to the right of the ~ symbol. The intercept is added automatically.
- LSEs are random variables.

Code

```
# fit regression line to predict son's height from father's height
fit <- lm(son ~ father, data = galton_heights) # variable to the right of ~ is used to predict the model
fit
# summary statistics
summary(fit)
```

LSE are Random Variables

Key points

- Because they are derived from the samples, LSE are random variables.

- β_0 and β_1 appear to be normally distributed because the central limit theorem plays a role.
- The t-statistic depends on the assumption that ϵ follows a normal distribution.

Code

```
# Monte Carlo simulation
B <- 1000
N <- 50
lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
    lm(son ~ father, data = .) %>%
    .$coef
})
lse <- data.frame(beta_0 = lse[1,], beta_1 = lse[2,])
# Plot the distribution of beta_0 and beta_1
library(gridExtra)
p1 <- lse %>% ggplot(aes(beta_0)) + geom_histogram(binwidth = 5, color = "black")
p2 <- lse %>% ggplot(aes(beta_1)) + geom_histogram(binwidth = 0.1, color = "black")
grid.arrange(p1, p2, ncol = 2)
# summary statistics
sample_n(galton_heights, N, replace = TRUE) %>%
  lm(son ~ father, data = .) %>%
  summary %>%
  .$coef

lse %>% summarize(se_0 = sd(beta_0), se_1 = sd(beta_1))
```

Advanced Note on LSE

Although interpretation is not straight-forward, it is also useful to know that the LSE can be strongly correlated, which can be seen using this code:

```
lse %>% summarize(cor(beta_0, beta_1))
```

However, the correlation depends on how the predictors are defined or transformed.

Here we standardize the father heights, which changes X_i to $X_i - \bar{X}$.

```
B <- 1000
N <- 50
lse <- replicate(B, {
    sample_n(galton_heights, N, replace = TRUE) %>%
    mutate(father = father - mean(father)) %>%
    lm(son ~ father, data = .) %>% .$coef
})
```

Observe what happens to the correlation in this case:

```
cor(lse[1,], lse[2,])
```

Predicted Variables are Random Variables

Saturday, March 14, 2020 3:18 PM

Key points

- The predicted value is often denoted as Y^, which is a random variable. Mathematical theory tells us what the standard error of the predicted value is.
- The predict() function in R can give us predictions directly.

Code

```
# plot predictions and confidence intervals
galton_heights %>% ggplot(aes(son, father)) +
  geom_point() +
  geom_smooth(method = "lm")

# predict Y directly
fit <- galton_heights %>% lm(son ~ father, data = .)
Y_hat <- predict(fit, se.fit = TRUE)
names(Y_hat)
# plot best fit line
galton_heights %>%
  mutate(Y_hat = predict(lm(son ~ father, data=.))) %>%
  ggplot(aes(father, Y_hat))+
  geom_line()
```

Tibbles & Do

Saturday, March 14, 2020 8:33 PM

Key points

- Tibbles can be regarded as a modern version of data frames and are the default data structure in the tidyverse.
- Some functions that do not work properly with data frames do work with tibbles.

Code

```
# stratify by HR
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR = round(HR/G, 1),
         BB = BB/G,
         R = R/G) %>%
  select(HR, BB, R) %>%
  filter(HR >= 0.4 & HR<=1.2)

# calculate slope of regression lines to predict runs by BB in different HR strata
dat %>%
  group_by(HR) %>%
  summarize(slope = cor(BB,R)*sd(R)/sd(BB))

# use lm to get estimated slopes - lm does not work with grouped tibbles
dat %>%
  group_by(HR) %>%
  lm(R ~ BB, data = .) %>%
  .$coef

# inspect a grouped tibble
dat %>% group_by(HR) %>% head()
dat %>% group_by(HR) %>% class()
```

Key points

- Tibbles are more readable than data frames.
- If you subset a data frame, you may not get a data frame. If you subset a tibble, you always get a tibble.
- Tibbles can hold more complex objects such as lists or functions.
- Tibbles can be grouped.

Code

```
# inspect data frame and tibble
Teams
as.tibble(Teams)
# subsetting a data frame sometimes generates vectors
class(Teams[,20])
# subsetting a tibble always generates tibbles
class(as.tibble(Teams[,20]))
# pulling a vector out of a tibble
class(as.tibble(Teams)$HR)
# access a non-existing column in a data frame or a tibble
Teams$hr
as.tibble(Teams)$hr
# create a tibble with complex objects
tibble(id = c(1, 2, 3), func = c(mean, median, sd))
```

Do

Key points

- The do() function serves as a bridge between R functions, such as lm(), and the tidyverse.
- We have to specify a column when using the do() function, otherwise we will get an error.
- If the data frame being returned has more than one row, the rows will be concatenated appropriately.

Code

```
# use do to fit a regression line to each HR stratum
dat %>%
  group_by(HR) %>%
  do(fit = lm(R ~ BB, data = .))

# using do without a column name gives an error
dat %>%
group_by(HR) %>%
do(lm(R ~ BB, data = .))
# define a function to extract slope from lm
get_slope <- function(data){
 fit <- lm(R ~ BB, data = data)
 data.frame(slope = fit$coefficients[2],
         se = summary(fit)$coefficient[2,2])
}
# return the desired data frame
dat %>%
 group_by(HR) %>%
 do(get_slope(.))

# not the desired output: a column containing data frames
dat %>%
 group_by(HR) %>%
 do(slope = get_slope(.))

# data frames with multiple rows will be concatenated appropriately
get_lse <- function(data){
 fit <- lm(R ~ BB, data = data)
 data.frame(term = names(fit$coefficients),
  slope = fit$coefficients,
  se = summary(fit)$coefficient[,2])
}
dat %>%
 group_by(HR) %>%
 do(get_lse(.))
```

From <https://courses.edx.org/courses/course-v1:HarvardX+PH125.7x+1T2020/courseware/90e85d216c7a4fef9ded45a947010154/43cef30e72dc46efa1dc82c35cd31139/?child=first>

broom

Saturday, March 14, 2020 8:45 PM

Key points

- The **broom** package has three main functions, all of which extract information from the object returned by **lm** and return it in a **tidyverse** friendly data frame.
- The tidy() function returns estimates and related information as a data frame.
- The functions glance() and augment() relate to model specific and observation specific outcomes respectively.

Code

```
# use tidy to return lm estimates and related information as a data frame
library(broom)
fit <- lm(R ~ BB, data = dat)
tidy(fit)
# add confidence intervals with tidy
tidy(fit, conf.int = TRUE)
# pipeline with lm, do, tidy
dat %>%
 group_by(HR) %>%
 do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
 filter(term == "BB") %>%
 select(HR, estimate, conf.low, conf.high)

# make ggplots
dat %>%
 group_by(HR) %>%
 do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
 filter(term == "BB") %>%
 select(HR, estimate, conf.low, conf.high) %>%
 ggplot(aes(HR, y = estimate, ymin = conf.low, ymax = conf.high)) +
 geom_errorbar() +
 geom_point()

# inspect with glance
glance(fit)
```

Building a Better Offensive Metric for Baseball

Sunday, March 15, 2020 4:02 PM

```
# linear regression with two variables
fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB/G, HR = HR/G,  R = R/G) %>%
  lm(R ~ BB + HR, data = .)
  tidy(fit, conf.int = TRUE)

# regression with BB, singles, doubles, triples, HR
fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB / G,
         singles = (H - X2B - X3B - HR) / G,
         doubles = X2B / G,
         triples = X3B / G,
         HR = HR / G,
         R = R / G) %>%
  lm(R ~ BB + singles + doubles + triples + HR, data = .)
coefs <- tidy(fit, conf.int = TRUE)
coefs

# predict number of runs for each team in 2002 and plot
Teams %>%
  filter(yearID %in% 2002) %>%
  mutate(BB = BB/G,
         singles = (H-X2B-X3B-HR)/G,
         doubles = X2B/G,
         triples =X3B/G,
         HR=HR/G,
         R=R/G)  %>%
  mutate(R_hat = predict(fit, newdata = .)) %>%
  ggplot(aes(R_hat, R, label = teamID)) +
  geom_point() +
  geom_text(nudge_x=0.1, cex = 2) +
  geom_abline()

# average number of team plate appearances per game
pa_per_game <- Batting %>% filter(yearID == 2002) %>%
  group_by(teamID) %>%
  summarize(pa_per_game = sum(AB+BB)/max(G)) %>%
  pull(pa_per_game) %>%
  mean

# compute per-plate-appearance rates for players available in 2002 using previous data
players <- Batting %>% filter(yearID %in% 1999:2001) %>%
  group_by(playerID) %>%
  mutate(PA = BB + AB) %>%
  summarize(G = sum(PA)/pa_per_game,
    BB = sum(BB)/G,
    singles = sum(H-X2B-X3B-HR)/G,
```

```
    doubles = sum(X2B)/G,
    triples = sum(X3B)/G,
    HR = sum(HR)/G,
    AVG = sum(H)/sum(AB),
    PA = sum(PA)) %>%
  filter(PA >= 300) %>%
  select(-G) %>%
  mutate(R_hat = predict(fit, newdata = .))

# plot player-specific predicted runs
qplot(R_hat, data = players, geom = "histogram", binwidth = 0.5, color = I("black"))

# add 2002 salary of each player
players <- Salaries %>%
  filter(yearID == 2002) %>%
  select(playerID, salary) %>%
  right_join(players, by="playerID")

# add defensive position
position_names <- c("G_p","G_c","G_1b","G_2b","G_3b","G_ss","G_lf","G_cf","G_rf")
tmp_tab <- Appearances %>%
  filter(yearID == 2002) %>%
  group_by(playerID) %>%
  summarize_at(position_names, sum) %>%
  ungroup()
pos <- tmp_tab %>%
  select(position_names) %>%
  apply(., 1, which.max)
players <- data_frame(playerID = tmp_tab$playerID, POS = position_names[pos]) %>%
  mutate(POS = str_to_upper(str_remove(POS, "G_"))) %>%
  filter(POS != "P") %>%
  right_join(players, by="playerID") %>%
  filter(!is.na(POS)  & !is.na(salary))

# add players' first and last names
players <- Master %>%
  select(playerID, nameFirst, nameLast, debut) %>%
  mutate(debut = as.Date(debut)) %>%
  right_join(players, by="playerID")

# top 10 players
players %>% select(nameFirst, nameLast, POS, salary, R_hat) %>%
  arrange(desc(R_hat)) %>%
  top_n(10)

# players with a higher metric have higher salaries
players %>% ggplot(aes(salary, R_hat, color = POS)) +
  geom_point() +
  scale_x_log10()

# remake plot without players that debuted after 1998
library(lubridate)
players %>% filter(year(debut) < 1998) %>%
```

```
 ggplot(aes(salary, R_hat, color = POS)) +
  geom_point() +
  scale_x_log10()
```

A way to actually pick the players for the team can be done using what computer scientists call **linear programming**. Although we don't go into this topic in detail in this course, we include the code anyway:

```
library(reshape2)
library(lpSolve)
players <- players %>% filter(debut <= "1997-01-01" & debut > "1988-01-01")
constraint_matrix <- acast(players, POS ~ playerID, fun.aggregate = length)
npos <- nrow(constraint_matrix)
constraint_matrix <- rbind(constraint_matrix, salary = players$salary)
constraint_dir <- c(rep("==", npos), "<=")
constraint_limit <- c(rep(1, npos), 50*10^6)
lp_solution <- lp("max", players$R_hat,
                  constraint_matrix, constraint_dir, constraint_limit,
                  all.int = TRUE)
```

This algorithm chooses these 9 players:

```
our_team <- players %>%
  filter(lp_solution$solution == 1) %>%
  arrange(desc(R_hat))
our_team %>% select(nameFirst, nameLast, POS, salary, R_hat)
  nameFirst    nameLast POS   salary R_hat
1    Jason     Giambi  1B 10428571  7.99
2    Nomar Garciaparra  SS  9000000  7.51
3     Mike     Piazza   C 10571429  7.16
4     Phil      Nevin  3B  2600000  6.75
5     Jeff       Kent  2B  6000000  6.68
```

We note that these players all have above average BB and HR rates while the same is not true for singles.

```
my_scale <- function(x) (x - median(x))/mad(x)
players %>% mutate(BB = my_scale(BB),
                   singles = my_scale(singles),
                   doubles = my_scale(doubles),
                   triples = my_scale(triples),
                   HR = my_scale(HR),
                   AVG = my_scale(AVG),
                   R_hat = my_scale(R_hat)) %>%
  filter(playerID %in% our_team$playerID) %>%
  select(nameFirst, nameLast, BB, singles, doubles, triples, HR, AVG, R_hat) %>%
  arrange(desc(R_hat))
  nameFirst    nameLast    BB singles doubles triples    HR  AVG R_hat
1    Jason     Giambi 3.317 -0.5315   0.754  -0.675 2.067 2.63  3.54
2    Nomar Garciaparra 0.284  1.7330   2.651   0.471 1.003 3.95  2.97
3     Mike     Piazza 0.596 -0.0499  -0.177  -1.335 2.682 1.70  2.56
4     Phil      Nevin 0.790 -0.6751   0.670  -1.137 2.103 1.09  2.07
5     Jeff       Kent 0.875 -0.2717   1.833   1.210 0.967 1.66  2.00
```

Regression Fallacy

Sunday, March 15, 2020 4:20 PM

Key points

- Regression can bring about errors in reasoning, especially when interpreting individual observations.
- The example showed in the video demonstrates that the **"sophomore slump"** observed in the data is caused by regressing to the mean.

Code

The code to create a table with player ID, their names, and their most played position:

```
library(Lahman)
playerInfo <- Fielding %>%
  group_by(playerID) %>%
  arrange(desc(G)) %>%
  slice(1) %>%
  ungroup %>%
  left_join(Master, by="playerID") %>%
  select(playerID, nameFirst, nameLast, POS)
```

The code to create a table with only the ROY award winners and add their batting statistics:

```
ROY <- AwardsPlayers %>%
  filter(awardID == "Rookie of the Year") %>%
  left_join(playerInfo, by="playerID") %>%
  rename(rookie_year = yearID) %>%
  right_join(Batting, by="playerID") %>%
  mutate(AVG = H/AB) %>%
  filter(POS != "P")
```

The code to keep only the rookie and sophomore seasons and remove players who did not play sophomore seasons:

```
ROY <- ROY %>%
  filter(yearID == rookie_year | yearID == rookie_year+1) %>%
  group_by(playerID) %>%
  mutate(rookie = ifelse(yearID == min(yearID), "rookie", "sophomore")) %>%
  filter(n() == 2) %>%
  ungroup %>%
  select(playerID, rookie_year, rookie, nameFirst, nameLast, AVG)
```

The code to use the spread function to have one column for the rookie and sophomore years batting averages:

```
ROY <- ROY %>% spread(rookie, AVG) %>% arrange(desc(rookie))
ROY
```

```
#> # A tibble: 99 x 6
#>   playerID  rookie_year nameFirst nameLast rookie sophomore
#>   <chr>           <int> <chr>     <chr>     <dbl>     <dbl>
#> 1 mccovwi01        1959 Willie    McCovey   0.354     0.238
#> 2 suzukic01        2001 Ichiro    Suzuki    0.350     0.321
#> 3 bumbral01        1973 Al        Bumbry    0.337     0.233
#> 4 lynnfr01         1975 Fred      Lynn      0.331     0.314
#> 5 pujolal01        2001 Albert    Pujols    0.329     0.314
#> 6 troutmi01        2012 Mike      Trout     0.326     0.323
#> # ... with 93 more rows
```

The code to calculate the proportion of players who have a lower batting average their sophomore year:

```
mean(ROY$sophomore - ROY$rookie <= 0)
```

```
#> [1] 0.677
```

The code to do the similar analysis on all players that played the 2013 and 2014 seasons and batted more than 130 times (minimum to win Rookie of the Year):

```
two_years <- Batting %>%
  filter(yearID %in% 2013:2014) %>%
  group_by(playerID, yearID) %>%
  filter(sum(AB) >= 130) %>%
  summarize(AVG = sum(H)/sum(AB)) %>%
  ungroup %>%
  spread(yearID, AVG) %>%
  filter(!is.na(`2013`) & !is.na(`2014`)) %>%
  left_join(playerInfo, by="playerID") %>%
  filter(POS!="P") %>%
  select(-POS) %>%
  arrange(desc(`2013`)) %>%
  select(nameFirst, nameLast, `2013`, `2014`)
two_years
#> # A tibble: 312 x 4
#>   nameFirst nameLast `2013` `2014`
#>   <chr>     <chr>     <dbl>  <dbl>
#> 1 Miguel    Cabrera   0.348  0.313
#> 2 Hanley    Ramirez   0.345  0.283
#> 3 Michael   Cuddyer   0.331  0.332
#> 4 Scooter   Gennett   0.324  0.289
#> 5 Joe       Mauer     0.324  0.277
#> 6 Mike      Trout     0.323  0.287
#> # ... with 306 more rows
```

The code to see what happens to the worst performers of 2013:

```
arrange(two_years, `2013`)
#> # A tibble: 312 x 4
#>   nameFirst nameLast `2013` `2014`
#>   <chr>     <chr>     <dbl>  <dbl>
#> 1 Danny     Espinosa  0.158  0.219
#> 2 Dan       Uggla     0.179  0.149
#> 3 Jeff      Mathis    0.181  0.2
#> 4 Melvin    Upton     0.184  0.208
#> 5 Adam      Rosales   0.190  0.262
#> 6 Aaron     Hicks     0.192  0.215
#> # ... with 306 more rows
```

The code to see the correlation for performance in two separate years:

```
qplot(`2013`, `2014`, data = two_years)
summarize(two_years, cor(`2013`,`2014`))
#> # A tibble: 1 x 1
#>   `cor(\`2013\`, \`2014\`)`
#>                       <dbl>
#> 1                     0.460
```

Measurement Error Models

Sunday, March 15, 2020 4:26 PM

Key points

- Up to now, all our linear regression examples have been applied to two or more random variables. We assume the pairs are bivariate normal and use this to motivate a linear model.
- Another use for linear regression is with **measurement error models**, where it is common to have a non-random covariate (such as time). Randomness is introduced from measurement error rather than sampling or natural variability.

Code

The code to use **dslabs** function rfalling_object to generate simulations of dropping balls:

```
library(dslabs)
falling_object <- rfalling_object()
```

The code to draw the trajectory of the ball:

```
falling_object %>%
  ggplot(aes(time, observed_distance)) +
  geom_point() +
  ylab("Distance in meters") +
  xlab("Time in seconds")
```

The code to use the lm() function to estimate the coefficients:

```
fit <- falling_object %>%
  mutate(time_sq = time^2) %>%
  lm(observed_distance~time+time_sq, data=.)
tidy(fit)
#> # A tibble: 3 x 5
#>   term        estimate std.error statistic  p.value
#>   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
#> 1 (Intercept)    56.9     0.580     98.0  1.56e-17
#> 2 time          -1.04     0.829    -1.25 2.36e- 1
#> 3 time_sq       -4.73     0.246    -19.2  8.17e-10
```

The code to check if the estimated parabola fits the data:

```
augment(fit) %>%
  ggplot() +
  geom_point(aes(time, observed_distance)) +
  geom_line(aes(time, .fitted), col = "blue")
```

The code to see the summary statistic of the regression:

```
tidy(fit, conf.int = TRUE)
#> # A tibble: 3 x 7
#>   term        estimate std.error statistic  p.value conf.low conf.high
#>   <chr>          <dbl>    <dbl>    <dbl>    <dbl>    <dbl>     <dbl>
#> 1 (Intercept)    56.9     0.580     98.0  1.56e-17    55.6      58.2
#> 2 time          -1.04     0.829    -1.25 2.36e- 1    -2.86     0.784
#> 3 time_sq       -4.73     0.246    -19.2  8.17e-10    -5.27    -4.19
```

Confounding - correlation is not causation

Saturday, March 21, 2020 11:20 AM

What can you do to determine if you are misinterpreting results because of a confounder?

- More closely examine the results by stratifying and plotting the data.

Correlation is Not Causation: Spurious Correlation

Key points

- Association/correlation is not causation.
- p-hacking is a topic of much discussion because it is a problem in scientific publications. Because publishers tend to reward statistically significant results over negative results, there is an incentive to report significant results.

Code

P-hacking :

- Looking for associations between an outcome and several exposures and only reporting the one that is significant.
- Trying several different models and selecting the one that yields the smallest p-value.
- Repeating an experiment multiple times and only reporting the one with the smallest p-value.

```
# generate the Monte Carlo simulation
N <- 25
g <- 1000000
sim_data <- tibble(group = rep(1:g, each = N), x = rnorm(N * g), y = rnorm(N * g))
# calculate correlation between X,Y for each group
res <- sim_data %>%
 group_by(group) %>%
 summarize(r = cor(x, y)) %>%
 arrange(desc(r))
res
# plot points from the group with maximum correlation
sim_data %>% filter(group == res$group[which.max(res$r)]) %>%
 ggplot(aes(x, y)) +
 geom_point() +
 geom_smooth(method = "lm")

# histogram of correlation in Monte Carlo simulations
res %>% ggplot(aes(x=r)) + geom_histogram(binwidth = 0.1, color = "black")
# linear regression on group with maximum correlation
library(broom)
sim_data %>%
 filter(group == res$group[which.max(res$r)]) %>%
 do(tidy(lm(y ~ x, data = .)))
```

Correlation is Not Causation: Outliers

Key points

- Correlations can be caused by **outliers**.
- The **Spearman correlation** is calculated based on the ranks of data.

Code, this takes out the impact of outliers

```
# simulate independent X, Y and standardize all except entry 23
set.seed(1985)
x <- rnorm(100,100,1)
y <- rnorm(100,84,1)
x[-23] <- scale(x[-23])
y[-23] <- scale(y[-23])
# plot shows the outlier
qplot(x, y, alpha = 0.5)
# outlier makes it appear there is correlation
cor(x,y)
cor(x[-23], y[-23])
# use rank instead
qplot(rank(x), rank(y))
cor(rank(x), rank(y))
# Spearman correlation with cor function
cor(x, y, method = "spearman")
```

Correlation is Not Causation: Reversing Cause and Effect

Key points

- Another way association can be confused with causation is when the **cause and effect are reversed**.
- As discussed in the video, in the Galton data, when father and son were reversed in the regression, the model was technically correct. The estimates and p-values were obtained correctly as well.
  What was incorrect was the **interpretation** of the model.

Code

```
# cause and effect reversal using son heights to predict father heights
library(HistData)
data("GaltonFamilies")
GaltonFamilies %>%
 filter(childNum == 1 & gender == "male") %>%
 select(father, childHeight) %>%
 rename(son = childHeight) %>%
 do(tidy(lm(father ~ son, data = .)))
```

Correlation is not Causation: Confounders

Key points

- If X and Y are correlated, we call Z a **confounder** if changes in Z causes changes in both X and Y.

Code

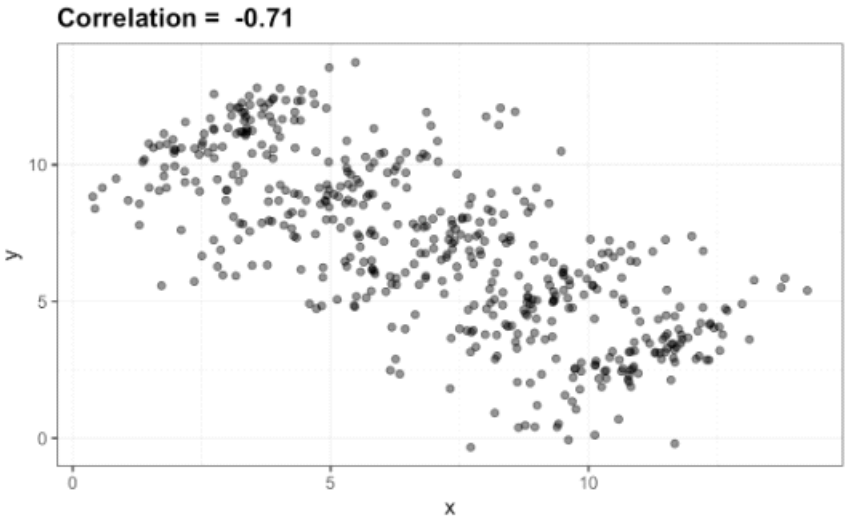
```
# UC-Berkeley admission data
library(dslabs)
data(admissions)
admissions
# percent men and women accepted
admissions %>% group_by(gender) %>%
 summarize(percentage =
           round(sum(admitted*applicants)/sum(applicants),1))

# test whether gender and admission are independent
admissions %>% group_by(gender) %>%
 summarize(total_admitted = round(sum(admitted / 100 * applicants)),
           not_admitted = sum(applicants) - sum(total_admitted)) %>%
 select(-gender) %>%
 do(tidy(chisq.test(.)))

# percent admissions by major
```

```
admissions %>% select(major, gender, admitted) %>%
  spread(gender, admitted) %>%
  mutate(women_minus_men = women - men)

# plot total percent admitted to major versus percent women applicants
admissions %>%
  group_by(major) %>%
  summarize(major_selectivity = sum(admitted * applicants) / sum(applicants),
            percent_women_applicants = sum(applicants * (gender=="women")) /
                                       sum(applicants) * 100) %>%
  ggplot(aes(major_selectivity, percent_women_applicants, label = major)) +
  geom_text()

# plot number of applicants admitted and not
admissions %>%
  mutate(yes = round(admitted/100*applicants), no = applicants - yes) %>%
  select(-applicants, -admitted) %>%
  gather(admission, number_of_students, -c("major", "gender")) %>%
  ggplot(aes(gender, number_of_students, fill = admission)) +
  geom_bar(stat = "identity", position = "stack") +
  facet_wrap(. ~ major)
admissions %>%
  mutate(percent_admitted = admitted * applicants/sum(applicants)) %>%
  ggplot(aes(gender, y = percent_admitted, fill = major)) +
  geom_bar(stat = "identity", position = "stack")
# condition on major and then look at differences
admissions %>% ggplot(aes(major, admitted, col = gender, size = applicants)) + geom_point()
# average difference by major
admissions %>%  group_by(gender) %>% summarize(average = mean(admitted))
```

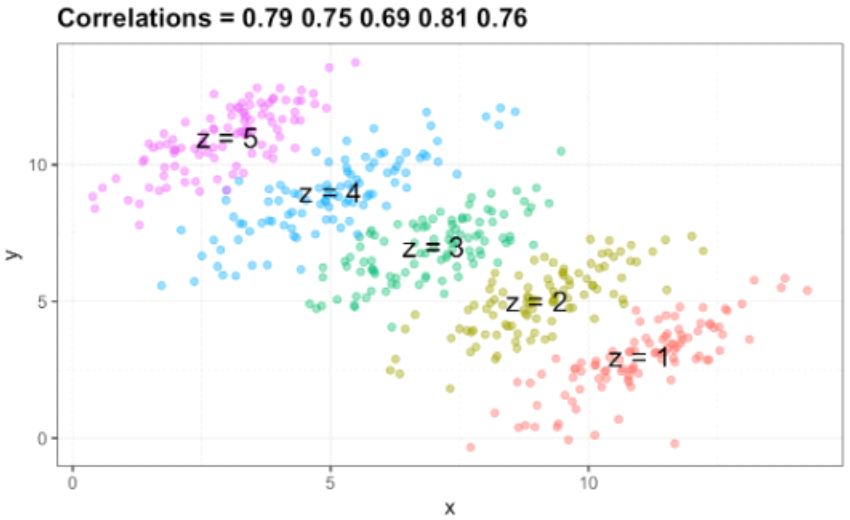
Simpson's Paradox

Key point

- Simpson’s Paradox happens when we see the sign of the correlation flip when comparing the entire dataset with specific strata.



You can see that X and Y are negatively correlated. However, once we stratify by Z (shown in different colors below) another pattern emerges:



It is really Z that is negatively correlated with X. If we stratify by Z, the X and Y are actually positively correlated as seen in the plot above.

Machine Learning

Saturday, March 21, 2020 6:40 PM

Notation

Key points

- $X_1,\ldots,X_p$ denote the features, Y denotes the outcomes, and $\hat{Y}$ denotes the predictions.
- Machine learning prediction tasks can be divided into **categorical** and **continuous** outcomes. We refer to these as **classification** and **prediction**, respectively.

Caret package, training and test sets, and overall accuracy

Key points

- To mimic the ultimate evaluation process, we randomly split our data into two — a training set and a test set — and act as if we don't know the outcome of the test set. We develop algorithms using only the training set; the test set is used only for evaluation.
- The createDataPartition() function from the **caret** package can be used to generate indexes for randomly splitting data.
- The simplest evaluation metric for categorical outcomes is overall accuracy: the proportion of cases that were correctly predicted in the test set.

Code

```
library(tidyverse)
library(caret)
library(dslabs)
data(heights)
# define the outcome and predictors
y <- heights$sex
x <- heights$height

# generate training and test sets
# times used to define how many random samples of indexes to return
#p used to define what proportion of the index
#list - if you wnt it to be retuned as a list or not
set.seed(2007)
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
test_set <- heights[test_index, ]
train_set <- heights[-test_index, ]
# guess the outcome
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE)
y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE) %>%
  factor(levels = levels(test_set$sex))

# compute accuracy
mean(y_hat == test_set$sex)
heights %>% group_by(sex) %>% summarize(mean(height), sd(height))
y_hat <- ifelse(x > 62, "Male", "Female") %>% factor(levels = levels(test_set$sex))
mean(y == y_hat)

# examine the accuracy of 10 cutoffs
cutoff <- seq(61, 70)
accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  mean(y_hat == train_set$sex)
```

```
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
y_hat <- factor(y_hat)
mean(y_hat == test_set$sex)
```

Confusion Matrix

Key points

- Overall accuracy can sometimes be a deceptive measure because of unbalanced classes.
- A general improvement to using overall accuracy is to study sensitivity and specificity separately. **Sensitivity**, also known as the true positive rate or recall, is the proportion of actual positive outcomes correctly identified as such. **Specificity**, also known as the true negative rate, is the proportion of actual negative outcomes that are correctly identified as such.
- A confusion matrix tabulates each combination of prediction and actual value. You can create a confusion matrix in R using the table() function or the confusionMatrix() function from the **caret** package.

High sensitivity means Y=1 -> Y_HAT=1 OR High sensitivity means Y_HAT=1 -> Y=1
High specificity means Y=0 -> Y_HAT=0

	Actually Positive	**Actually Negative**
Predicted positive	True positives (TP)	False positives (FP)
Predicted negative	False negatives (FN)	True negatives (TN)

Measure of	Name 1	Name 2	Definition	Probability representation
sensitivity	TPR	Recall	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$	$\Pr(\hat{Y}=1 \mid Y=1)$
specificity	TNR	1-FPR	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$	$\Pr(\hat{Y}=0 \mid Y=0)$
specificity	PPV	Precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$	$\Pr(Y=1 \mid \hat{Y}=1)$

FPR - false positive rate
TPR - True positive rate
PPV - positive predictive value

Code

```
# tabulate each combination of prediction and actual value
table(predicted = y_hat, actual = test_set$sex)
test_set %>%
  mutate(y_hat = y_hat) %>%
  group_by(sex) %>%
  summarize(accuracy = mean(y_hat == sex))
prev <- mean(y == "Male")
confusionMatrix(data = y_hat, reference = test_set$sex)
```

Balanced accuracy and F1 score

Key points

- For optimization purposes, sometimes it is more useful to have a one number summary than studying both specificity and sensitivity. One preferred metric is **balanced accuracy**. Because specificity and sensitivity are rates, it is more appropriate to compute the ***harmonic*** average. In fact, the **F1-score**, a widely used one-number summary, is the harmonic average of precision and recall.
- Depending on the context, some type of errors are more costly than others. The **F1-score** can be adapted to weigh specificity and sensitivity differently.
- You can compute the **F1-score** using the F_meas() function in the **caret** package.

Code

```
# maximize F-score
cutoff <- seq(61, 70)
F_1 <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  F_meas(data = y_hat, reference = factor(train_set$sex))
})
max(F_1)
best_cutoff <- cutoff[which.max(F_1)]
y_hat <- ifelse(test_set$height > best_cutoff, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
sensitivity(data = y_hat, reference = test_set$sex)
specificity(data = y_hat, reference = test_set$sex)

ggplot() + geom_line(aes(x = cutoff, y=F_1))
```

Key points

- A machine learning algorithm with very high sensitivity and specificity may not be useful in practice when prevalence is close to either 0 or 1. For example, if you develop an algorithm for disease diagnosis with very high sensitivity, but the prevalence of the disease is pretty low, then the precision of your algorithm is probably very low based on Bayes' theorem.

Example Code

Sunday, March 22, 2020 6:15 PM

```
library(dslabs)
library(dplyr)
library(lubridate)
data(reported_heights)

dat <- mutate(reported_heights, date_time = ymd_hms(time_stamp)) %>%
  filter(date_time >= make_date(2016, 01, 25) & date_time < make_date(2016, 02, 1)) %>%
  mutate(sex = factor(sex),
    type = ifelse(day(date_time) == 25 & hour(date_time) == 8 & between(minute(date_time), 15, 30),
"inclass","online")) %>%
  select(sex, type)

# define the outcome and predictors
y <- factor(dat$sex, c("Female", "Male"))
x <- dat$type

# view proportions of data to use for educated guess
inclass <- dat %>% filter( type == "inclass")
 prop.table(table(inclass))

online <- dat %>% filter( type == "online")
 prop.table(table(online))

# calc accuracy using my educated guess.  Example online has 67% Males so I know male is more likely
for online
 y_hat <- ifelse(x == "online", "Male", "Female") %>% factor(levels = levels(dat$sex))
 mean(y == y_hat) # this code calcs the accuracy

# Once you have y_hat you can create a table to compare y_hat to actual
table(y_hat, y)

#creates a confustion matrix of prediction compared to actual with a number of metrics
confusionMatrix(y_hat, y)
sensitivity(y_hat, y) #can also type the specific metric as a function from the caret package
```

```
library(caret)
data(iris)
iris <- iris[-which(iris$Species=='setosa'),]
y <- iris$Species

set.seed(2)    # if using R 3.6 or later, use set.seed(2, sample.kind="Rounding")
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
test <- iris[test_index,]
train <- iris[-test_index,]
```

```
cutoff <- seq(0, 10, by = .1 )
accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train$Sepal.Length > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff


accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train$Sepal.Width > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff

accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train$Petal.Length > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff

accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train$Petal.Width > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff


best_cutoff <- 4.7

accuracy <- map_dbl(best_cutoff, function(x){
  y_hat <- ifelse(test$Petal.Length > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
```

```
#_____using test

cutoff <- seq(0, 10, by = .1 )
accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(test$Sepal.Length > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff


accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(test$Sepal.Width > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff

accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(test$Petal.Length > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff

accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(test$Petal.Width > x, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
})
max(accuracy)
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff


#accuracy using best cutoffs from two variables

  y_hat <- ifelse(train$Petal.Width > 1.5 | test$Petal.Length > 4.7, "virginica", "versicolor") %>%
    factor(levels = levels(test$Species))
  mean(y_hat == test$Species)
```

ROC and precision-recall curves

Sunday, March 22, 2020 1:06 PM

Key points

- A very common approach to evaluating accuracy and F1-score is to compare them graphically by plotting both. A widely used plot that does this is the **receiver operating characteristic (ROC) curve**. The ROC curve plots sensitivity (TPR) versus 1 - specificity or the false positive rate (FPR).
- However, ROC curves have one weakness and it is that neither of the measures plotted depend on prevalence. In cases in which prevalence matters, we may instead make a **precision-recall plot**, which has a similar idea with ROC curve.

Code

```
p <- 0.9
n <- length(test_index)
y_hat <- sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
  factor(levels = levels(test_set$sex))
mean(y_hat == test_set$sex)
# ROC curve
probs <- seq(0, 1, length.out = 10)
guessing <- map_df(probs, function(p){
  y_hat <-
    sample(c("Male", "Female"), n, replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guessing",
       FPR = 1 - specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
})
guessing %>% qplot(FPR, TPR, data =., xlab = "1 - Specificity", ylab = "Sensitivity")
cutoffs <- c(50, seq(60, 75), 80)
height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       FPR = 1-specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
})
# plot both curves together
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(FPR, TPR, color = method)) +
  geom_line() +
  geom_point() +
  xlab("1 - Specificity") +
  ylab("Sensitivity")
library(ggrepel)
map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       cutoff = x,
       FPR = 1-specificity(y_hat, test_set$sex),
       TPR = sensitivity(y_hat, test_set$sex))
}) %>%
  ggplot(aes(FPR, TPR, label = cutoff)) +
  geom_line() +
  geom_point() +
  geom_text_repel(nudge_x = 0.01, nudge_y = -0.01)
```

```r
# plot precision against recall
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index),
                  replace = TRUE, prob=c(p, 1-p)) %>%
    factor(levels = c("Female", "Male"))
  list(method = "Guess",
    recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})
height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Female", "Male"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, test_set$sex),
    precision = precision(y_hat, test_set$sex))
})
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()
guessing <- map_df(probs, function(p){
  y_hat <- sample(c("Male", "Female"), length(test_index), replace = TRUE,
                  prob=c(p, 1-p)) %>%
    factor(levels = c("Male", "Female"))
  list(method = "Guess",
    recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})
height_cutoff <- map_df(cutoffs, function(x){
  y_hat <- ifelse(test_set$height > x, "Male", "Female") %>%
    factor(levels = c("Male", "Female"))
  list(method = "Height cutoff",
       recall = sensitivity(y_hat, relevel(test_set$sex, "Male", "Female")),
    precision = precision(y_hat, relevel(test_set$sex, "Male", "Female")))
})
bind_rows(guessing, height_cutoff) %>%
  ggplot(aes(recall, precision, color = method)) +
  geom_line() +
  geom_point()
```

Conditional Probabilities

Sunday, April 5, 2020 4:57 PM

Key points

- Conditional probabilities for each class:

$$p_k(x) = Pr(Y = k|X = x), for\, k = 1, \ldots, K$$

- In machine learning, this is referred to as **Bayes' Rule**. This is a theoretical rule because in practice we don't know $p(x)$. Having a good estimate of the $p(x)$ will suffice for us to build optimal prediction models, since we can control the balance between specificity and sensitivity however we wish. In fact, estimating these conditional probabilities can be thought of as the main challenge of machine learning.

Key points

- Due to the connection between **conditional probabilities** and **conditional expectations**:

$$p_k(x) = Pr(Y = k|X = x), \text{ for } k = 1, \ldots, K$$

we often only use the expectation to denote both the conditional probability and conditional expectation.

- For continuous outcomes, we define a loss function to evaluate the model. The most commonly used one is **MSE (Mean Squared Error)**. The reason why we care about the conditional expectation in machine learning is that the expected value minimizes the MSE:

$$\hat{Y} = E(Y|X = x) \text{ minimizes } E\{(\hat{Y} - Y)^2|X = x\}$$

Due to this property, a succinct description of the main task of machine learning is that we use data to estimate for any set of features. **The main way in which competing machine learning algorithms differ is in their approach to estimating this expectation.**

Linear Regression for Prediction

Saturday, April 11, 2020 9:30 AM

Key points

- Linear regression can be considered a machine learning algorithm. Although it can be too rigid to be useful, it works rather well for some challenges. It also serves as a baseline approach: if you can't beat it with a more complex approach, you probably want to stick to linear regression.

Code

```
library(HistData)
set.seed(1983)
galton_heights <- GaltonFamilies %>%
 filter(gender == "male") %>%
 group_by(family) %>%
 sample_n(1) %>%
 ungroup() %>%
 select(father, childHeight) %>%
 rename(son = childHeight)
y <- galton_heights$son
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- galton_heights %>% slice(-test_index)
test_set <- galton_heights %>% slice(test_index)
m <- mean(train_set$son)
# squared loss
mean((m - test_set$son)^2)
# fit linear regression model
fit <- lm(son ~ father, data = train_set)
fit$coef
y_hat <- fit$coef[1] + fit$coef[2]*test_set$father
mean((y_hat - test_set$son)^2)
```

Prediction Function

Key points

- The predict() function takes a fitted object from functions such as lm() or glm() and a data frame with the new predictors for which to predict. We can use predict like this:
  y_hat <- predict(fit, test_set)
- predict() is a generic function in R that calls other functions depending on what kind of object it receives. To learn about the specifics, you can read the help files using code like this:
  ?predict.lm # or ?predict.glm

Code

```
y_hat <- predict(fit, test_set)
mean((y_hat - test_set$son)^2)
# read help files
?predict.lm
?predict.glm
```

Q1 assessment Chapter 3.1
Create 100 linear models and calc the RMSE for each model, then calc the mean and stedv of the RMSE's from all the models

```
set.seed(1)
Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)
dat <- MASS::mvrnorm(n = 100, c(69, 69), Sigma) %>%
  data.frame() %>% setNames(c("x", "y"))

set.seed(1)
y <- dat$y
RMSE<-replicate(100, {
  test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
  test_set <- dat %>% slice(test_index)
  train_set <- dat %>% slice(-test_index)
fit <- lm(y ~ x, data = train_set)
y_hat <- predict(fit, test_set)
sqrt(mean((y_hat - test_set$y)^2))
})

RMSE
mean(RMSE)
sd(RMSE)
```

Q2 assessment, repeat Q1 but using datasets of different sizes (n)

```
n <- c(100, 500, 1000, 5000, 10000)

models <- function(n) {
Sigma <- 9*matrix(c(1.0, 0.5, 0.5, 1.0), 2, 2)
dat <- MASS::mvrnorm(n , c(69, 69), Sigma) %>%
  data.frame() %>% setNames(c("x", "y"))
replicate(100, {
  y <- dat$y
  test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
  test_set <- dat %>% slice(test_index)
  train_set <- dat %>% slice(-test_index)
  fit <- lm(y ~ x, data = train_set)
  y_hat <- predict(fit, test_set)
  sqrt(mean((y_hat - test_set$y)^2)) })}

set.seed(1)
df <- as.data.frame(sapply(n, models))

mean(df$V1)
mean(df$V2)
mean(df$V3)
mean(df$V4)
mean(df$V5)

sd(df$V1)
sd(df$V2)
sd(df$V3)
sd(df$V4)
sd(df$V5)
```

Q6

Create a data set using the following code.

```
set.seed(1)
Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.25, 0.75, 0.25, 1.0), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
        data.frame() %>% setNames(c("y", "x_1", "x_2"))
```

Note that `y` is correlated with both `x_1` and `x_2` but the two predictors are independent of each other, as seen by `cor(dat)`.

Set the seed to 1, then use the **caret** package to partition into a test and training set of equal size. Compare the RMSE when using just `x_1`, just `x_2` and both `x_1` and `x_2`. Train a single linear model for each (not 100 like in the previous questions).

Which of the three models performs the best (has the lowest RMSE)?

```
set.seed(1)
Sigma <- matrix(c(1.0, 0.75, 0.75, 0.75, 1.0, 0.25, 0.75, 0.25, 1.0), 3, 3)
dat <- MASS::mvrnorm(n = 100, c(0, 0, 0), Sigma) %>%
  data.frame() %>% setNames(c("y", "x_1", "x_2"))

set.seed(1)
y <- dat$y

  test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
  test_set <- dat %>% slice(test_index)
  train_set <- dat %>% slice(-test_index)
  x_1fit <- lm(y ~ x_1 , data = train_set)
  x_2fit <- lm(y ~ x_2 , data = train_set)
  bothfit <- lm(y ~ x_1 + x_2, data = train_set)


  y_hat1 <- predict(x_1fit, test_set)
  y_hat2 <- predict(x_2fit, test_set)
  y_hat3 <- predict(bothfit, test_set)


  sqrt(mean((y_hat1 - test_set$y)^2))
  sqrt(mean((y_hat2 - test_set$y)^2))
  sqrt(mean((y_hat3 - test_set$y)^2))
```

Regression for a Categorical Outcome

Sunday, April 12, 2020 12:45 PM

Key points

- The regression approach can be extended to categorical data. For example, we can try regression to estimate the conditional probability:

$$p(x) = Pr(Y = 1|X = x) = \beta_0 + \beta_1 x$$

- Once we have estimates β_0 and β_1, we can obtain an actual prediction $p(x)$. Then we can define a specific decision rule to form a prediction.

Code

```
library(dslabs)
data("heights")
y <- heights$height
set.seed(2) #if you are using R 3.5 or earlier
set.seed(2, sample.kind = "Rounding") #if you are using R 3.6 or later
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- heights %>% slice(-test_index)
test_set <- heights %>% slice(test_index)

#what is the probability for being female if you are 66 inches tall
train_set %>%
 filter(round(height)==66) %>%
 summarize(y_hat = mean(sex=="Female"))
#what is the probability for being female at all given heights
heights %>%
 mutate(x = round(height)) %>%
 group_by(x) %>%
 filter(n() >= 10) %>%
 summarize(prop = mean(sex == "Female")) %>%
 ggplot(aes(x, prop)) +
 geom_point()

lm_fit <- mutate(train_set, y = as.numeric(sex == "Female")) %>% lm(y ~ height, data = .)
p_hat <- predict(lm_fit, test_set)
y_hat <- ifelse(p_hat > 0.5, "Female", "Male") %>% factor()
confusionMatrix(y_hat, test_set$sex)$overall["Accuracy"]
```

Logistic Regression

Sunday, April 12, 2020 12:53 PM

Key points

- **Logistic regression** is an extension of linear regression that assures that the estimate of conditional probability $Pr\left(Y=1|X=x\right)$ is between 0 and 1. This approach makes use of the logistic transformation:

$$g\left(p\right)=log\frac{p}{1-p}$$

- With logistic regression, we model the conditional probability directly with:

$$g\{Pr\left(Y=1|X=x\right)\}=\beta_0+\beta_1 x$$

- Note that with this model, we can no longer use least squares. Instead we compute the **maximum likelihood estimate (MLE)**.
- In R, we can fit the logistic regression model with the function `glm()` (generalized linear models). If we want to compute the conditional probabilities, we want `type="response"` since the default is to return the logistic transformed values.

Code

```
heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(prop = mean(sex == "Female")) %>%
  ggplot(aes(x, prop)) +
  geom_point() +
  geom_abline(intercept = lm_fit$coef[1], slope = lm_fit$coef[2])
range(p_hat)
# fit logistic regression model
glm_fit <- train_set %>%
  mutate(y = as.numeric(sex == "Female")) %>%
  glm(y ~ height, data=., family = "binomial")
p_hat_logit <- predict(glm_fit, newdata = test_set, type = "response")
y_hat_logit <- ifelse(p_hat_logit > 0.5, "Female", "Male") %>% factor
confusionMatrix(y_hat_logit, test_set$sex)$overall[["Accuracy"]]
```

Plot actual, LM, and GLM

```
heights %>%
  mutate(x = round(height)) %>%
  group_by(x) %>%
  filter(n() >= 10) %>%
  summarize(actual = mean(sex == "Female")) %>%
  mutate(logistic = plogis(glm_fit$coef[1] + glm_fit$coef[2]*x),
         regression = lm_fit$coef[1] + lm_fit$coef[2]*x) %>%
  gather(method, p_x, -x) %>%
  ggplot(aes(x, p_x, color = method)) +
  geom_line() +
  geom_hline(yintercept = 0.5, lty = 5)
```

>

Case Study 2 or 7 (2 Predictors)

Sunday, April 12, 2020 1:26 PM

Key points

- In this case study we apply logistic regression to classify whether a digit is two or seven. We are interested in estimating a conditional probability that depends on two variables:

$$g\{p(x_1, x_2\} = g\{Pr\left(Y = 1|X_1 = x_1, X_2 = x_2\right)\} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

- Through this case, we know that logistic regression forces our estimates to be a **plane** and our boundary to be a **line**. This implies that a logistic regression approach has no chance of capturing the **non-linear** nature of the true $p\left(x_1, x_2\right)$. Therefore, we need other more flexible methods that permit other shapes.

Code

```
mnist <- read_mnist()
is <- mnist_27$index_train[c(which.min(mnist_27$train$x_1), which.max(mnist_27$train$x_1))]
titles <- c("smallest","largest")
tmp <- lapply(1:2, function(i){
  expand.grid(Row=1:28, Column=1:28) %>%
    mutate(label=titles[i],
           value = mnist$train$images[is[i],])
})
tmp <- Reduce(rbind, tmp)
tmp %>% ggplot(aes(Row, Column, fill=value)) +
  geom_raster() +
  scale_y_reverse() +
  scale_fill_gradient(low="white", high="black") +
  facet_grid(.~label) +
  geom_vline(xintercept = 14.5) +
  geom_hline(yintercept = 14.5)
data("mnist_27")
mnist_27$train %>% ggplot(aes(x_1, x_2, color = y)) + geom_point()
is <- mnist_27$index_train[c(which.min(mnist_27$train$x_2), which.max(mnist_27$train$x_2))]
titles <- c("smallest","largest")
tmp <- lapply(1:2, function(i){
  expand.grid(Row=1:28, Column=1:28) %>%
    mutate(label=titles[i],
           value = mnist$train$images[is[i],])
})
tmp <- Reduce(rbind, tmp)
tmp %>% ggplot(aes(Row, Column, fill=value)) +
  geom_raster() +
  scale_y_reverse() +
  scale_fill_gradient(low="white", high="black") +
  facet_grid(.~label) +
  geom_vline(xintercept = 14.5) +
  geom_hline(yintercept = 14.5)
fit_glm <- glm(y ~ x_1 + x_2, data=mnist_27$train, family = "binomial")
p_hat_glm <- predict(fit_glm, mnist_27$test)
y_hat_glm <- factor(ifelse(p_hat_glm > 0.5, 7, 2))
confusionMatrix(data = y_hat_glm, reference = mnist_27$test$y)$overall["Accuracy"]
mnist_27$true_p %>% ggplot(aes(x_1, x_2, fill=p)) +
  geom_raster()
mnist_27$true_p %>% ggplot(aes(x_1, x_2, z=p, fill=p)) +
  geom_raster() +
  scale_fill_gradientn(colors=c("#F8766D","white","#00BFC4")) +
  stat_contour(breaks=c(0.5), color="black")
p_hat <- predict(fit_glm, newdata = mnist_27$true_p)
mnist_27$true_p %>%
  mutate(p_hat = p_hat) %>%
  ggplot(aes(x_1, x_2,  z=p_hat, fill=p_hat)) +
  geom_raster() +
  scale_fill_gradientn(colors=c("#F8766D","white","#00BFC4")) +
  stat_contour(breaks=c(0.5),color="black")
p_hat <- predict(fit_glm, newdata = mnist_27$true_p)
mnist_27$true_p %>%
  mutate(p_hat = p_hat) %>%
```

```
ggplot() +
stat_contour(aes(x_1, x_2, z=p_hat), breaks=c(0.5), color="black") +
geom_point(mapping = aes(x_1, x_2, color=y), data = mnist_27$test)
```

Smoothing

Sunday, April 12, 2020 2:20 PM

Key points

- **Smoothing** is a very powerful technique used all across data analysis. It is designed to detect trends in the presence of noisy data in cases in which the shape of the trend is unknown.
- The concepts behind smoothing techniques are extremely useful in machine learning because **conditional expectations/probabilities** can be thought of as **trends** of unknown shapes that we need to estimate in the presence of uncertainty.

Code

```
data("polls_2008")
qplot(day, margin, data = polls_2008)
```

Bin Smoothing and Kernels

Key points

- The general idea of smoothing is to group data points into strata in which the value of $f(x)$ can be assumed to be constant. We can make this assumption because we think $f(x)$ changes slowly and, as a result, $f(x)$ is almost constant in small windows of time.
- This assumption implies that a good estimate for $f(x)$ is the average of the Y_i values in the window. The estimate is:

$$\hat{f}(x_0) = \frac{1}{N_0} \sum_{i \in A_0} Y_i$$

- In smoothing, we call the size of the interval $|x - x_0|$ satisfying the particular condition the window size, bandwidth or span.

Code

```
# bin smoothers
span <- 7
fit <- with(polls_2008,ksmooth(day, margin, x.points = day, kernel="box", bandwidth =span))
polls_2008 %>% mutate(smooth = fit$y) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth), color="red")
# kernel
span <- 7
fit <- with(polls_2008, ksmooth(day, margin,  x.points = day, kernel="normal", bandwidth = span))
polls_2008 %>% mutate(smooth = fit$y) %>%
 ggplot(aes(day, margin)) +
 geom_point(size = 3, alpha = .5, color = "grey") +
 geom_line(aes(day, smooth), color="red")
```

Local Weighted Regression (loess)

Key points

- A limitation of the bin smoothing approach is that we need small windows for the approximately constant assumptions to hold which may lead to imprecise estimates of f(x). **Local weighted regression (loess)** permits us to consider larger window sizes.
- One important difference between loess and bin smoother is that we assume the smooth function is locally **linear** in a window instead of constant.
- The result of loess is a smoother fit than bin smoothing because we use larger sample sizes to estimate our local parameters.

Code

```
polls_2008 %>% ggplot(aes(day, margin)) +
geom_point() +
geom_smooth(color="red", span = 0.15, method.args = list(degree=1))
```

Matrices

Saturday, April 18, 2020 3:33 PM

Key points

- The main reason for using matrices is that certain mathematical operations needed to develop efficient code can be performed using techniques from a branch of mathematics called **linear algebra**.
- **Linear algebra** and **matrix notation** are key elements of the language used in academic papers describing machine learning techniques.

Code

```
library(tidyverse)
library(dslabs)
if(!exists("mnist")) mnist <- read_mnist()
class(mnist$train$images)
x <- mnist$train$images[1:1000,]
y <- mnist$train$labels[1:1000]
```

- In matrix algebra, we have three main types of objects: **scalars**, **vectors**, and **matrices**.
  - **Scalar:** $\alpha = 1$
  - **Vector:** $X_1 = \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{N,1} \end{pmatrix}$
  - **Matrix:** $X = [X_1 X_2] = \begin{pmatrix} x_{1,1} & x_{1,2} \\ \vdots & \vdots \\ x_{N,1} & x_{N,2} \end{pmatrix}$
- In R, we can extract the dimension of a matrix with the function `dim()`. We can convert a vector into a matrix using the function `as.matrix()`.

Code

```
length(x[,1])
x_1 <- 1:5
x_2 <- 6:10
cbind(x_1, x_2)
dim(x)
dim(x_1)
dim(as.matrix(x_1))
dim(x)
```

Converting a Vector to a Matrix

Key points

In R, we can **convert a vector into a matrix** with the matrix() function. The matrix is filled in by column, but we can fill by row by using the byrow argument. The function t() can be used to directly transpose a matrix.
Note that the matrix function **recycles values in the vector** without warning if the product of columns and rows does not match the length of the vector.

Code

```
my_vector <- 1:15
# fill the matrix by column
mat <- matrix(my_vector, 5, 3)
mat
# fill by row
mat_t <- matrix(my_vector, 3, 5, byrow = TRUE)
mat_t
identical(t(mat), mat_t)
matrix(my_vector, 5, 5)
grid <- matrix(x[3,], 28, 28)
image(1:28, 1:28, grid)
# flip the image back
image(1:28, 1:28, grid[, 28:1])
```

Row and Column Summaries and Apply

Saturday, April 18, 2020 3:41 PM

Key points

- The function rowSums() computes the sum of each row.
- The function rowMeans() computes the average of each row.
- We can compute the column sums and averages using the functions colSums() and colMeans().
- The **matrixStats** package adds functions that performs operations on each row or column very efficiently, including the functions rowSds() and colSds().
- The apply() function lets you apply any function to a matrix. The first argument is the **matrix**, the second is the **dimension** (1 for rows, 2 for columns), and the third is the **function**.

Code

```
sums <- rowSums(x)
avg <- rowMeans(x)
data_frame(labels = as.factor(y), row_averages = avg) %>%
  qplot(labels, row_averages, data = ., geom = "boxplot")
avgs <- apply(x, 1, mean)
sds <- apply(x, 2, sd
```

Filtering Columns Based on Summaries

Key points

- The operations used to extract columns: x[,c(351,352)].
- The operations used to extract rows: x[c(2,3),].
- We can also use logical indexes to determine which columns or rows to keep: new_x <- x[,colSds(x) > 60].
- **Important note:** if you select only one column or only one row, the result is no longer a matrix but a **vector**. We can **preserve the matrix class** by using the argument drop=FALSE.

Code

```
library(matrixStats)
sds <- colSds(x)
qplot(sds, bins = "30", color = I("black"))
image(1:28, 1:28, matrix(sds, 28, 28)[, 28:1])
#extract columns and rows
x[ ,c(351,352)]
x[c(2,3),]
new_x <- x[ ,colSds(x) > 60]
dim(new_x)
class(x[,1])
dim(x[1,])
#preserve the matrix class
class(x[ , 1, drop=FALSE])
dim(x[, 1, drop=FALSE])
```

Which of the following lines of code would add the scalar 1 to row 1, the scalar 2 to row 2, and so on, for the matrix x

```
x <- x + seq(nrow(x))
```

OR

```
x <- sweep(x, 1, 1:nrow(x),"+")
```

Which of the following lines of code would add the scalar 1 to column 1, the scalar 2 to column 2, and so on, for the matrix `x`?

```
x <- sweep(x, 2, 1:ncol(x), FUN = "+")
```

rowMeans() - calc mean of rows
colMeans() - calc mean of cols

Indexing with Matrices and Binarizing the Data

Saturday, April 18, 2020 3:44 PM

Key points

- We can use logical operations with matrices:
  mat <- matrix(1:15, 5, 3)
  mat[mat > 6 & mat < 12] <- 0
- We can also binarize the data using just matrix operations:
  bin_x <- x
  bin_x[bin_x < 255/2] <- 0
  bin_x[bin_x > 255/2] <- 1

Code

```
#index with matrices
mat <- matrix(1:15, 5, 3)
as.vector(mat)
qplot(as.vector(x), bins = 30, color = I("black"))
new_x <- x
new_x[new_x < 50] <- 0
mat <- matrix(1:15, 5, 3)
mat[mat < 3] <- 0
mat
mat <- matrix(1:15, 5, 3)
mat[mat > 6 & mat < 12] <- 0
mat
#binarize the data
bin_x <- x
bin_x[bin_x < 255/2] <- 0
bin_x[bin_x > 255/2] <- 1
bin_X <- (x > 255/2)*1
```

Vectorization for Matrices and Matrix Algebra Operations

Saturday, April 18, 2020 3:45 PM

Key points

- We can scale each row of a matrix using this line of code:
  `(x - rowMeans(x)) / rowSds(x)`
- To scale each column of a matrix, we use this code:
  `t(t(X) - colMeans(X))`
- We can also use a function called `sweep()` that works similarly to `apply()`. It takes each entry of a vector and subtracts it from the corresponding row or column:
  `X_mean_0 <- sweep(x, 2, colMeans(x))`
- Matrix multiplication: `t(x) %*% x`
- The cross product: `crossprod(x)`
- The inverse of a function: `solve(crossprod(x))`
- The QR decomposition: `qr(x)`

Code

```
#scale each row of a matrix
(x - rowMeans(x)) / rowSds(x)
#scale each column
t(t(x) - colMeans(x))
#take each entry of a vector and subtracts it from the corresponding row or column
x_mean_0 <- sweep(x, 2, colMeans(x))
#divide by the standard deviation
x_mean_0 <- sweep(x, 2, colMeans(x))
x_standardized <- sweep(x_mean_0, 2, colSds(x), FUN = "/")
```

Distance

Saturday, April 18, 2020 3:59 PM

Distance

Key points

- Most clustering and machine learning techniques rely on being able to define distance between observations, using features or predictors.
- With high dimensional data, a quick way to compute all the distances at once is to use the function dist(), which computes the distance between each row and produces an object of class dist():
  d <- dist(x)
- We can also compute distances between predictors. If N is the number of observations, the distance between two predictors, say 1 and 2, is:
  $$dist(1,2)=\sqrt{\sum_{i=1}^{N}(x_{i,1}-x_{i,2})^2}$$
- To compute the distance between all pairs of the 784 predictors, we can transpose the matrix first and then use dist():
  d <- dist(t(x))

Code

```
library(tidyverse)
library(dslabs)
if(!exists("mnist")) mnist <- read_mnist()
set.seed(1995) # if using R 3.5 or earlier
set.seed(1995) # if using R 3.6 or later
ind <- which(mnist$train$labels %in% c(2,7)) %>% sample(500)
#the predictors are in x and the labels in y
x <- mnist$train$images[ind,]
y <- mnist$train$labels[ind]
y[1:3]
x_1 <- x[1,]
x_2 <- x[2,]
x_3 <- x[3,]
#distance between two numbers
sqrt(sum((x_1 - x_2)^2))
sqrt(sum((x_1 - x_3)^2))
sqrt(sum((x_2 - x_3)^2))
#compute distance using matrix algebra
sqrt(crossprod(x_1 - x_2))
sqrt(crossprod(x_1 - x_3))
sqrt(crossprod(x_2 - x_3))
#compute distance between each row
d <- dist(x)
class(d)
as.matrix(d)[1:3,1:3]
#visualize these distances
image(as.matrix(d))
#order the distance by labels
image(as.matrix(d)[order(y), order(y)])
#compute distance between predictors
d <- dist(t(x))
dim(as.matrix(d))
d_492 <- as.matrix(d)[492,]
```

```
image(1:28, 1:28, matrix(d_492, 28, 28)
```

Which of the following lines of code computes the Euclidean distance between each observation and stores it in the object `d`?

```
d <- dist(tissue_gene_expression$x)
```

KNN

Saturday, April 18, 2020 4:20 PM

Key points

- **K-nearest neighbors (kNN)** estimates the conditional probabilities in a similar way to bin smoothing. However, kNN is easier to adapt to multiple dimensions.
- Using kNN, for any point (X_1,X_2) for which we want an estimate of $p(X_1,X_2)$, we look for the ***k* nearest points** to (X_1,X_2) and take an average of the 0s and 1s associated with these points. We refer to the set of points used to compute the average as the **neighborhood**. Larger values of k result in smoother estimates, while smaller values of k result in more flexible and more wiggly estimates.
- To implement the algorithm, we can use the knn3() function from the **caret** package. There are two ways to call this function:

1. We need to specify a formula and a data frame. The formula looks like this: $outcome \sim predictor_1 + predictor_2 + predictor_3$. The predict() function for knn3 produces a probability for each class.
2. We can also call the function with the first argument being the matrix predictors and the second a vector of outcomes, like this:

```
x <- as.matrix(mnist_27$train[,2:3])
y <- mnist_27$train$y
knn_fit <- knn3(x,y)
```

Code

```
library(caret)

#fit a GLM and compute accuracy
fit_glm <- glm(y~x_1+x_2, data=mnist_27$train, family="binomial")
p_hat_logistic <- predict(fit_glm, mnist_27$test)
y_hat_logistic <- factor(ifelse(p_hat_logistic > 0.5, 7, 2))
confusionMatrix(data = y_hat_logistic, reference = mnist_27$test$y)$overall[1]

#fit knn model
knn_fit <- knn3(y ~ ., data = mnist_27$train)

#2nd approach for KNN
x <- as.matrix(mnist_27$train[,2:3])
y <- mnist_27$train$y
knn_fit <- knn3(x, y)

#fit KNN and compute accuracy
knn_fit <- knn3(y ~ ., data = mnist_27$train, k=5)
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$test$y)$overall["Accuracy"]
```

Overtraining and Over smoothing

Saturday, April 18, 2020 4:27 PM

Key points

- **Over-training** is the reason that we have higher accuracy in the train set compared to the test set. Over-training is at its worst when we set $k=1$. With $k=1$, the estimate for each (x_1,x_2) in the training set is obtained with just the y corresponding to that point.
- When we try a larger k, the k might be so large that it does not permit enough flexibility. We call this **over-smoothing**.
- Note that if we use the test set to pick this k, we should not expect the accompanying accuracy estimate to extrapolate to the real world. This is because even here we broke a golden rule of machine learning: **we selected the k using the test set**. **Cross validation** also provides an estimate that takes this into account.

Code

```
y_hat_knn <- predict(knn_fit, mnist_27$train, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$train$y)$overall["Accuracy"]
y_hat_knn <- predict(knn_fit, mnist_27$test, type = "class")
confusionMatrix(data = y_hat_knn, reference = mnist_27$test$y)$overall["Accuracy"]
#fit knn with k=1
knn_fit_1 <- knn3(y ~ ., data = mnist_27$train, k = 1)
y_hat_knn_1 <- predict(knn_fit_1, mnist_27$train, type = "class")
confusionMatrix(data=y_hat_knn_1, reference=mnist_27$train$y)$overall[["Accuracy"]]
#fit knn with k=401
knn_fit_401 <- knn3(y ~ ., data = mnist_27$train, k = 401)
y_hat_knn_401 <- predict(knn_fit_401, mnist_27$test, type = "class")
confusionMatrix(data=y_hat_knn_401, reference=mnist_27$test$y)$overall["Accuracy"]
#pick the k in knn
ks <- seq(3, 251, 2)
library(purrr)
accuracy <- map_df(ks, function(k){
  fit <- knn3(y ~ ., data = mnist_27$train, k = k)
y_hat <- predict(fit, mnist_27$train, type = "class")
  cm_train <- confusionMatrix(data = y_hat, reference = mnist_27$train$y)
  train_error <- cm_train$overall["Accuracy"]
  y_hat <- predict(fit, mnist_27$test, type = "class")
  cm_test <- confusionMatrix(data = y_hat, reference = mnist_27$test$y)
  test_error <- cm_test$overall["Accuracy"]

tibble(train = train_error, test = test_error)
  })
})
#pick the k that maximizes accuracy using the estimates built on the test data
ks[which.max(accuracy$test)]
max(accuracy$test)
```

Example Knn code

Saturday, April 18, 2020 9:16 PM

Previously, we used logistic regression to predict sex based on height. Now we are going to use knn to do the same. Set the seed to 1, then use the **caret** package to partition the **dslabs** heights data into a training and test set of equal size. Use the sapply() or map function to perform knn with k values of seq(1, 101, 3) and calculate F1 scores with the F_meas() function using the default value of the relevant argument.

```
data("heights")
set.seed(1)
test_index <- createDataPartition(heights$sex, times = 1, p = 0.5, list = FALSE)
train_set <- heights %>% slice(-test_index)
test_set <- heights %>% slice(test_index)
y <- train_set$sex
x <- train_set$height
ks <- seq(1, 101, 3)

F_1 <- map_dbl(ks, function(k){
  fit <- knn3(sex ~ height, data = train_set, k = k)
  y_hat <- predict(fit, test_set, type = "class")
  F_meas(data = y_hat, reference = factor(test_set$sex))
})
max(F_1)
which.max(F_1)
ks[16]


plot(ks,F_1)
```

Next we will use the same gene expression example used in the Comprehension Check: Distance exercises. You can load it like this:

```
library(dslabs)
data("tissue_gene_expression")
```

First, set the seed to 1 and split the data into training and test sets. Then, report the accuracy you obtain from predicting tissue type using KNN with k = 1, 3, 5, 7, 9, 11 using sapply() or map_df(). Note: use the createDataPartition() function outside of sapply() or map_df().

```
library(dslabs)
data("tissue_gene_expression")
dfy <- as.data.frame(tissue_gene_expression$y)
names(dfy) <- "tissue"
dfx <- as.data.frame(tissue_gene_expression$x)
DF <- data.frame(dfy, dfx)
```

```
set.seed(1)
test_index <- createDataPartition(DF$tissue, times = 1, p = 0.5, list = FALSE)
train_set <- DF %>% slice(-test_index)
test_set <- DF %>% slice(test_index)
ks <- c(1,3,5,7,9,11)


F_1 <- map_dbl(ks, function(k){
  fit <- knn3(tissue ~ ., data = train_set, k = k)
  y_hat <- predict(fit, test_set, type = "class")
  cm_test <- confusionMatrix(data = y_hat, reference = test_set$tissue)
  test_error <- cm_test$overall["Accuracy"]
})

F_1
```

k-fold Cross Validation

Sunday, April 19, 2020 2:29 PM

Key points

- For **k-fold cross validation**, we divide the dataset into a training set and a test set. We train our algorithm exclusively on the training set and use the test set only for evaluation purposes.
- For each set of algorithm parameters being considered, we want an **estimate of the MSE and then we will choose the parameters with the smallest MSE**. In k-fold cross validation, we randomly split the observations into k non-overlapping sets, and repeat the calculation for MSE for each of these sets. Then, we compute the average MSE and obtain an estimate of our loss. Finally, we can select the optimal parameter that minimized the MSE.
- In terms of how to select k for cross validation, **larger values of k are preferable but they will also take much more** computational time. For this reason, the choices of $k=5$ and $k=10$ are common.

>

Bootstrap

Sunday, April 19, 2020 3:54 PM

Key points

- When we don't have access to the entire population, we can use **bootstrap** to estimate the population median m .
- The bootstrap permits us to **approximate a Monte Carlo simulation** without access to the entire distribution. The general idea is relatively simple. We act as if the observed sample is the population. We then sample datasets (with replacement) of the same sample size as the original dataset. Then we compute the summary statistic, in this case the median, on this bootstrap sample.
- Note that we can use ideas similar to those used in the bootstrap in **cross validation**: instead of dividing the data into equal partitions, we simply bootstrap many times.

Code

```
n <- 10^6
income <- 10^(rnorm(n, log10(45000), log10(3)))
qplot(log10(income), bins = 30, color = I("black"))
m <- median(income)
m
set.seed(1995)
#use set.seed(1995, sample.kind="Rounding") instead if using R 3.6 or later
N <- 250
X <- sample(income, N)
M<- median(X)
M
library(gridExtra)
B <- 10^4
M <- replicate(B, {
  X <- sample(income, N)
  median(X)
})

p1 <- qplot(M, bins = 30, color = I("black"))
p2 <- qplot(sample = scale(M)) + geom_abline()
grid.arrange(p1, p2, ncol = 2)
mean(M)
sd(M)
B <- 10^4
M_star <- replicate(B, {
  X_star <- sample(X, N, replace = TRUE)
  median(X_star)
})
tibble(monte_carlo = sort(M), bootstrap = sort(M_star)) %>%
  qplot(monte_carlo, bootstrap, data = .) +
  geom_abline()

quantile(M, c(0.05, 0.95))
quantile(M_star, c(0.05, 0.95))
median(X) + 1.96 * sd(X) / sqrt(N) * c(-1, 1)
mean(M) + 1.96 * sd(M) * c(-1,1)
mean(M_star) + 1.96 * sd(M_star) * c(-1, 1)
```

Generative Models

Saturday, April 25, 2020 2:44 PM

Key points

- **Discriminative approaches** estimate the conditional probability directly and do not consider the distribution of the predictors.
- **Generative models** are methods that model the joint distribution and X (we model how the entire data, X and Y, are generated).

Naive Bayes

Saturday, April 25, 2020 2:46 PM

Code

```
# Generating train and test set
library("caret")
data("heights")
y <- heights$height
set.seed(2)
test_index <- createDataPartition(y, times = 1, p = 0.5, list = FALSE)
train_set <- heights %>% slice(-test_index)test_set <- heights %>% slice(test_index)
# Estimating averages and standard deviations
params <- train_set %>%
 group_by(sex) %>%
 summarize(avg = mean(height), sd = sd(height))
params
# Estimating the prevalence
pi <- train_set %>% summarize(pi=mean(sex=="Female")) %>% pull(pi)
pi
# Getting an actual rule
x <- test_set$height
f0 <- dnorm(x, params$avg[2], params$sd[2])
f1 <- dnorm(x, params$avg[1], params$sd[1])
p_hat_bayes <- f1*pi / (f1*pi + f0*(1 - pi))
```

Controlling Prevalence

Saturday, April 25, 2020 2:59 PM

Code

```
# Computing sensitivity
y_hat_bayes <- ifelse(p_hat_bayes > 0.5, "Female", "Male")
sensitivity(data = factor(y_hat_bayes), reference = factor(test_set$sex))
# Computing specificity
specificity(data = factor(y_hat_bayes), reference = factor(test_set$sex))
# Changing the cutoff of the decision rule
p_hat_bayes_unbiased <- f1 * 0.5 / (f1 * 0.5 + f0 * (1 - 0.5))
y_hat_bayes_unbiased <- ifelse(p_hat_bayes_unbiased > 0.5, "Female", "Male")
sensitivity(data = factor(y_hat_bayes_unbiased), reference = factor(test_set$sex))
specificity(data = factor(y_hat_bayes_unbiased), reference = factor(test_set$sex))
# Draw plot
qplot(x, p_hat_bayes_unbiased, geom = "line") +
 geom_hline(yintercept = 0.5, lty = 2) +
 geom_vline(xintercept = 67, lty = 2)
```

qda and lda

Saturday, April 25, 2020 2:59 PM

QDA

```
# Load data
data("mnist_27")
# Estimate parameters from the data
params <- mnist_27$train %>%
 group_by(y) %>%
 summarize(avg_1 = mean(x_1), avg_2 = mean(x_2),
       sd_1 = sd(x_1), sd_2 = sd(x_2),
       r = cor(x_1, x_2))
# Contour plots
mnist_27$train %>% mutate(y = factor(y)) %>%
 ggplot(aes(x_1, x_2, fill = y, color = y)) +
 geom_point(show.legend = FALSE) +
 stat_ellipse(type="norm", lwd = 1.5)
# Fit model
library(caret)
train_qda <- train(y ~., method = "qda", data = mnist_27$train)
# Obtain predictors and accuracy
y_hat <- predict(train_qda, mnist_27$test)
confusionMatrix(data = y_hat, reference = mnist_27$test$y)$overall["Accuracy"]
# Draw separate plots for 2s and 7s
mnist_27$train %>% mutate(y = factor(y)) %>%
 ggplot(aes(x_1, x_2, fill = y, color = y)) +
 geom_point(show.legend = FALSE) +
 stat_ellipse(type="norm") +
 facet_wrap(~y)
```

LDA

```
params <- mnist_27$train %>%
 group_by(y) %>%
 summarize(avg_1 = mean(x_1), avg_2 = mean(x_2),
       sd_1 = sd(x_1), sd_2 = sd(x_2),
       r = cor(x_1, x_2))
params <- params %>% mutate(sd_1 = mean(sd_1), sd_2 = mean(sd_2), r = mean(r))
train_lda <- train(y ~., method = "lda", data = mnist_27$train)
y_hat <- predict(train_lda, mnist_27$test)
confusionMatrix(data = y_hat, reference = mnist_27$test$y)$overall["Accuracy"]
```