Data Manipulation in R

Data Manipulation Objectives:

* Learn how to summarize data efficiently
* Understand how to convert between wide form and long form data as well as the uses for each type of data
* Recognize "tidy" data and use tidyr and dplyr functions to work towards "tidy" data
* Understand how to work with dates and times in R

# Reading in Data

## Setup

# Setup.R file  
source("https://bit.ly/2i8sicn")  
##   
## Attaching package: 'dplyr'  
## The following objects are masked from 'package:stats':  
##   
## filter, lag  
## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union  
##   
## Attaching package: 'lubridate'  
## The following object is masked from 'package:base':  
##   
## date

## Data in Excel

Formats xlsx and csv - what's the difference?

* File extensions xls and xlsx are proprietary Excel formats, binary files
* csv is an extension for Comma Separated Value files. They are text files - directly readable.
* Example: Gas prices in midwest since 1994

### Gas Prices in the Midwest

midwest <- read.csv("https://bit.ly/2hwWiQ8")  
head(midwest)  
## Year.Month Week.1 X Week.2 X.1 Week.3 X.2 Week.4 X.3  
## 1 End Date Value End Date Value End Date Value End Date Value  
## 2 1994-Nov 28-Nov 1.122  
## 3 1994-Dec 5-Dec 1.086 12-Dec 1.057 19-Dec 1.039 26-Dec 1.027  
## 4 1995-Jan 2-Jan 1.025 9-Jan 1.046 16-Jan 1.031 23-Jan 1.054  
## 5 1995-Feb 6-Feb 1.045 13-Feb 1.04 20-Feb 1.031 27-Feb 1.052  
## 6 1995-Mar 6-Mar 1.053 13-Mar 1.042 20-Mar 1.048 27-Mar 1.065  
## Week.5 X.4  
## 1 End Date Value  
## 2   
## 3   
## 4 30-Jan 1.055  
## 5   
## 6

str(midwest)  
## 'data.frame': 212 obs. of 11 variables:  
## $ Year.Month: Factor w/ 212 levels "","1994-Dec",..: 1 3 2 8 7 11 4 12 10 9 ...  
## $ Week.1 : Factor w/ 86 levels "","1-Apr","1-Aug",..: 86 1 52 18 65 69 26 10 56 31 ...  
## $ X : Factor w/ 197 levels "","0.905","0.918",..: 197 1 19 7 12 13 21 29 42 31 ...  
## $ Week.2 : Factor w/ 86 levels "","10-Apr","10-Aug",..: 86 1 28 78 41 45 2 70 32 7 ...  
## $ X.1 : Factor w/ 206 levels "","0.919","0.921",..: 206 1 17 14 12 13 27 39 45 34 ...  
## $ Week.3 : Factor w/ 86 levels "","15-Apr","15-Aug",..: 86 1 52 18 65 69 26 10 56 31 ...  
## $ X.2 : Factor w/ 199 levels "","0.91","0.929",..: 199 1 11 9 9 15 28 40 38 29 ...  
## $ Week.4 : Factor w/ 85 levels "22-Apr","22-Aug",..: 85 82 51 17 64 68 25 9 55 30 ...  
## $ X.3 : Factor w/ 201 levels "0.883","0.921",..: 201 29 9 14 13 15 32 44 34 27 ...  
## $ Week.5 : Factor w/ 31 levels "","29-Apr","29-Aug",..: 31 1 1 16 1 1 1 9 1 27 ...  
## $ X.4 : Factor w/ 74 levels "","0.955","1.023",..: 74 1 1 5 1 1 1 18 1 11 ...

The variables are all factors, which are numeric variables with labels. Instead, they should be character and numeric variables. In addition, the first row is made up of labels, rather than actual data.

There is clearly some work to be done...

read.csv is really just a wrapper for read.table with certain parameters set:

read.csv  
## function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",   
## fill = TRUE, comment.char = "", ...)   
## read.table(file = file, header = header, sep = sep, quote = quote,   
## dec = dec, fill = fill, comment.char = comment.char, ...)  
## <bytecode: 0x126a4248>  
## <environment: namespace:utils>

So to properly read in this data, it's probably best to use read.table directly

?read.table

##### Your Turn

Have a look at the parameters of read.table (?read.table) to solve the following problems:

1. Read the first two lines of the file into an object called midwest\_names
2. Read everything EXCEPT the first two lines into an object called midwest\_data

##### Solutions

1. Read the first two lines of the file into an object called midwest\_names

midwest\_names <- read.table("https://bit.ly/2hwWiQ8",   
 nrows = 2, sep = ",",   
 stringsAsFactors = FALSE)

1. Read everything EXCEPT the first two lines into an object called midwest\_data

midwest\_data <- read.table("https://bit.ly/2hwWiQ8",   
 skip = 2, sep = ",",   
 stringsAsFactors = FALSE)

###### End Solutions

## Demo: Data Cleaning

In order to demonstrate reading data in to R, some quick data cleaning is necessary.

First, all of the gas prices are collected in the values variable.

values <- c(midwest\_data$V3, midwest\_data$V5, midwest\_data$V7,   
 midwest\_data$V9, midwest\_data$V11)

Then, all of the dates are collected in the dates variable, with the year, month, and end date pasted together separated by "-".

dates <- c(paste(midwest\_data$V1, midwest\_data$V2, sep = "-"),   
 paste(midwest\_data$V1, midwest\_data$V4, sep = "-"),  
 paste(midwest\_data$V1, midwest\_data$V6, sep = "-"),  
 paste(midwest\_data$V1, midwest\_data$V8, sep = "-"),  
 paste(midwest\_data$V1, midwest\_data$V10, sep = "-"))

Missing/empty values are then removed from both variables.

dates <- dates[!is.na(values)]  
values <- values[!is.na(values)]

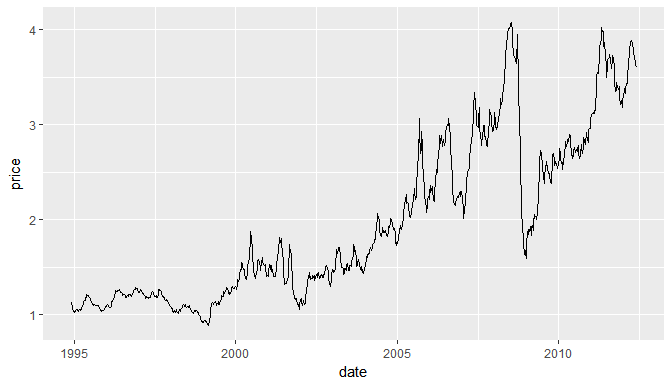
Finally, the dates are converted to Date variable types using the ymd function (more on this later). The dates and price variables (the values collected above) are combined into a data frame and then ordered chronologically.

library(lubridate)  
dates <- ymd(dates)  
  
midwest\_gas <- data.frame(date = dates, price = values)  
midwest\_gas <- midwest\_gas[with(midwest\_gas, order(date)), ]

A simpler way to reorganize data structured like this will be shown later.

With the tidy version of the midwest gas dataset constructed, it is fairly easy to plot the data:

library(ggplot2)  
qplot(date, price, data = midwest\_gas, geom = "line")



## Reading Excel Data

library(readxl)  
download.file(url = "https://bit.ly/2i6oyFi",  
 destfile = "midwest.xlsx",  
 mode = "wb")  
  
midwest2 <- read\_excel("midwest.xlsx")  
# Fix name issues  
names(midwest2) <- make.names(names(midwest2))  
head(midwest2, 4)  
## # A tibble: 4 × 11  
## Year.Month Week.1 X Week.2 X  
## <chr> <chr> <chr> <chr> <chr>  
## 1 <NA> End Date Value End Date Value  
## 2 1994-Nov <NA> <NA> <NA> <NA>  
## 3 1994-Dec 43074 1.0860000000000001 43081 1.0569999999999999  
## 4 1995-Jan 42737 1.0249999999999999 42744 1.046  
## # ... with 6 more variables: Week.3 <chr>, X <chr>, Week.4 <chr>, X <chr>,  
## # Week.5 <chr>, X <chr>

## foreign Package

Other file formats can be read using the functions from package foreign

* SPSS: read.spss
* SAS: read.xport
* Minitab: read.mtp

##### Your Turn

The NHANES (National Health and Nutrition Survey) publishes data in the SAS xport format:

<https://wwwn.cdc.gov/Nchs/Nhanes/Search/nhanes13_14.aspx>

1. Scroll to the bottom, choose one of the datasets (Demographics, Examination, etc.). Download the Data file (XPT)
2. Use read.xport() to load the file into R

##### Solutions

download.file("https://wwwn.cdc.gov/Nchs/Nhanes/2013-2014/DEMO\_H.XPT",   
 destfile = "DEMO.XPT", mode = "wb")  
library(foreign)  
demographic.vars <- read.xport("DEMO.XPT")

# Summarizing Data

## Baseball Data

* The plyr package contains the data set baseball
* seasonal batting statistics of all major league players (through 2007)

data(baseball, package = "plyr")  
help(baseball, package = "plyr")  
## starting httpd help server ...  
## done  
head(baseball)  
## id year stint team lg g ab r h X2b X3b hr rbi sb cs bb so  
## 4 ansonca01 1871 1 RC1 25 120 29 39 11 3 0 16 6 2 2 1  
## 44 forceda01 1871 1 WS3 32 162 45 45 9 4 0 29 8 0 4 0  
## 68 mathebo01 1871 1 FW1 19 89 15 24 3 1 0 10 2 1 2 0  
## 99 startjo01 1871 1 NY2 33 161 35 58 5 1 1 34 4 2 3 0  
## 102 suttoez01 1871 1 CL1 29 128 35 45 3 7 3 23 3 1 1 0  
## 106 whitede01 1871 1 CL1 29 146 40 47 6 5 1 21 2 2 4 1  
## ibb hbp sh sf gidp  
## 4 NA NA NA NA NA  
## 44 NA NA NA NA NA  
## 68 NA NA NA NA NA  
## 99 NA NA NA NA NA  
## 102 NA NA NA NA NA  
## 106 NA NA NA NA NA

* Goal: create career summary statistics for each player
* Plan: subset on a player, and compute statistics

ss <- subset(baseball, id == "sosasa01")  
head(ss, 3)  
## id year stint team lg g ab r h X2b X3b hr rbi sb cs bb  
## 66822 sosasa01 1989 1 TEX AL 25 84 8 20 3 0 1 3 0 2 0  
## 66823 sosasa01 1989 2 CHA AL 33 99 19 27 5 0 3 10 7 3 11  
## 67907 sosasa01 1990 1 CHA AL 153 532 72 124 26 10 15 70 32 16 33  
## so ibb hbp sh sf gidp  
## 66822 20 0 0 4 0 3  
## 66823 27 2 2 1 2 3  
## 67907 150 4 6 2 6 10  
mean(ss$h / ss$ab)  
## [1] 0.2681506

There should be an **automatic** way to calculate this.

## for loops

* Idea: repeat the same (set of) statement(s) for each element of an index set
* Setup:
  + Introduce counter variable (sometimes named i)
  + Reserve space for results
* Generic Code:

result <- rep(NA, length(indexset))  
for (i in indexset) {  
 ... some statements ...  
 result[i] <- ...  
}

### for loops for Baseball

* Index set: player id
* Setup:

players <- unique(baseball$id)  
n <- length(players)  
  
ba <- rep(NA, n)  
  
for (i in 1:n) {  
 career <- subset(baseball, id == players[i])  
 ba[i] <- with(career, mean(h / ab, na.rm = TRUE))  
}  
  
summary(ba)  
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 0.0000 0.1831 0.2459 0.2231 0.2699 0.5000 6

* Index set: player id
* i = 0:

players <- unique(baseball$id)  
n <- length(players)  
  
ba <- rep(NA, n)  
  
head(ba)  
## [1] NA NA NA NA NA NA

* Index set: player id
* i = 1:

players <- unique(baseball$id)  
  
ba <- rep(NA, length(players))  
  
for (i in 1:1) {  
 career <- subset(baseball, id == players[i])  
 ba[i] <- with(career, mean(h / ab, na.rm = TRUE))  
}  
  
head(ba)  
## [1] 0.3371163 NA NA NA NA NA

* Index set: player id
* i = 2:

players <- unique(baseball$id)  
  
ba <- rep(NA, length(players))  
  
for (i in 1:2) {  
 career <- subset(baseball, id == players[i])  
 ba[i] <- with(career, mean(h / ab, na.rm = TRUE))  
}  
  
head(ba)  
## [1] 0.3371163 0.2489226 NA NA NA NA

* Index set: player id
* i = 3:

players <- unique(baseball$id)  
  
ba <- rep(NA, length(players))  
  
for (i in 1:3) {  
 career <- subset(baseball, id == players[i])  
 ba[i] <- with(career, mean(h / ab, na.rm = TRUE))  
}  
  
head(ba)  
## [1] 0.3371163 0.2489226 0.2018073 NA NA NA

##### Your Turn

* MLB rules for the greatest all-time hitters are that players have to have played
  + at least 1000 games
  + with at least 1000 at-bats in order to be considered
* Extend the for loop above to collect the additional information
* Introduce and collect data for two new variables: games and atbats

### Solution

library(dplyr)  
data(baseball, package = "plyr")  
  
  
players <- unique(baseball$id)   
ba <- rep(NA, length(players))   
games <- rep(NA, length(players))   
atbats <- rep(NA, length(players))   
for (i in 1:length(players)) {   
 career <- subset(baseball, id == players[i])   
 ba[i] <- with(career, mean(h / ab, na.rm = TRUE))   
 games[i] <- with(career, sum(g, na.rm = TRUE))  
 atbats[i] <- with(career, sum(ab, na.rm = TRUE))  
}

Using forloops can be difficult and tedious. - Household chores (declaring variables, setting values each time) distract from real work - Indices are error-prone - Loops often result in slow code because R can compute quantities using entire vectors in an optimized way

Fortunately, R makes this process much easier with the summarize function

## Summarize

A special function: summarise or summarize

This function takes a data frame worth of data and computes summary functions using the entire data frame.

library(dplyr)  
  
baseball <- read.csv("https://bit.ly/2iFyIwL")  
  
summarize(baseball,  
 ba = mean(h / ab, na.rm = TRUE),  
 games = sum(g, na.rm = TRUE),  
 hr = sum(hr, na.rm = TRUE),  
 ab = sum(ab, na.rm = TRUE))  
## ba games hr ab  
## 1 0.2339838 1580070 113577 4891061

When passed a subset of the data, the summarize function computes statistics for only that portion of the data frame.

summarize(subset(baseball, id == "sosasa01"),   
 ba = mean(h / ab, na.rm = TRUE),  
 games = sum(g, na.rm = TRUE),  
 hr = sum(hr, na.rm = TRUE),  
 ab = sum(ab, na.rm = TRUE))  
## ba games hr ab  
## 1 0.2681506 2354 609 8813

## dplyr + Summarize

The dplyr package facilitates splitting the dataset into sub-datasets (groups), applying functions to those sub-groups, and then recombining the function results into a new dataset. This process is very powerful when creating summary statistics.

careers <- summarize(group\_by(baseball, id),  
 ba = mean(h / ab, na.rm = TRUE),  
 games = sum(g, na.rm = TRUE),  
 homeruns = sum(hr, na.rm = TRUE),  
 atbats = sum(ab, na.rm = TRUE))  
  
head(careers)  
## # A tibble: 6 × 5  
## id ba games homeruns atbats  
## <fctr> <dbl> <int> <int> <int>  
## 1 aaronha01 0.3010752 3298 755 12364  
## 2 abernte02 0.1824394 681 0 181  
## 3 adairje01 0.2363071 1165 57 4019  
## 4 adamsba01 0.2096513 482 3 1019  
## 5 adamsbo03 0.2378073 1281 37 4019  
## 6 adcocjo01 0.2751690 1959 336 6606

## Pipes

Rather than nesting functions(inside(each(other))), use a pipe %>%:

careers <- baseball %>%  
 group\_by(id) %>%  
 summarize(  
 ba = mean(h / ab, na.rm = TRUE),  
 games = sum(g, na.rm = TRUE),  
 homeruns = sum(hr, na.rm = TRUE),  
 atbats = sum(ab, na.rm = TRUE)  
 )

Pipes make code more readable and easier to understand.

a %>% function(b)  
is the same as  
function(a, b):

the piped variable is substituted for the first function argument.

If the argument passed in is not the first argument, use .:  
b %>% function(a, .)  
is the same as  
function(a, b)

##### Your Turn

* Find some summary statistics for each of the teams (variable team)
  + How many different (unique) players has the team had?
  + What was the team's first/last season?
* Challenge: Find the number of players on each team over time. Does the number change?

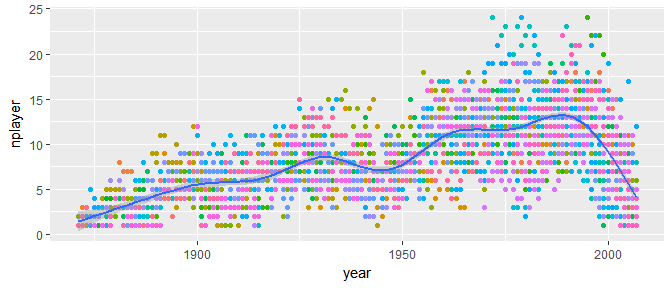
##### Solutions

1. Find some summary stats for each of the teams (team)
   * How many unique players has the team had?
   * What was the team's first/last season?

baseball %>%   
 group\_by(team) %>%   
 summarize(  
 nplayer = length(unique(id)),   
 first = min(year),   
 last = max(year))  
## # A tibble: 132 × 4  
## team nplayer first last  
## <fctr> <int> <int> <int>  
## 1 ALT 1 1884 1884  
## 2 ANA 29 1997 2004  
## 3 ARI 43 1998 2007  
## 4 ATL 133 1966 2007  
## 5 BAL 158 1954 2007  
## 6 BFN 11 1879 1885  
## 7 BFP 1 1890 1890  
## 8 BL1 5 1872 1874  
## 9 BL2 6 1882 1889  
## 10 BL3 3 1890 1891  
## # ... with 122 more rows

1. Challenge: Find the number of players on each team over time. Does the number change?

players.per.year <- baseball %>%   
 group\_by(team, year) %>%   
 summarize(nplayer = n())   
## `geom\_smooth()` using method = 'gam'



# Transforming and Analyzing Data

## Data Housekeeping

80% of data analysis is spent on the process of cleaning and preparing the data.

[](https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html)

“Tidy datasets are all alike but every messy dataset is messy in its own way.” – Hadley Wickham

### French Fries Data

During a ten week sensory experiment, 12 individuals were asked to assess taste of french fries on several scales:

How \_\_\_\_ do the fries taste?

* potato-y
* buttery
* grassy
* rancid
* paint-y

This experiment involved

* 3 different oils
* 2 replicates
* 12 individuals
* 10 weeks
* 5 qualities to rate

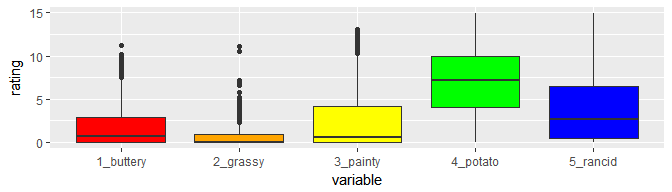
Each week, each individual tasted 6 batches of french fries.

french\_fries <- read.csv("https://bit.ly/2hOI890")  
head(french\_fries)  
## time treatment subject rep potato buttery grassy rancid painty  
## 1 1 1 3 1 2.9 0.0 0.0 0.0 5.5  
## 2 1 1 3 2 14.0 0.0 0.0 1.1 0.0  
## 3 1 1 10 1 11.0 6.4 0.0 0.0 0.0  
## 4 1 1 10 2 9.9 5.9 2.9 2.2 0.0  
## 5 1 1 15 1 1.2 0.1 0.0 1.1 5.1  
## 6 1 1 15 2 8.8 3.0 3.6 1.5 2.3

#### Issues

This format is not ideal for data analysis

library(ggplot2)  
  
qplot("1\_buttery", buttery, data = french\_fries, fill = I("red"), geom = "boxplot") +  
 geom\_boxplot(aes(x = "2\_grassy", y = grassy), fill = I("orange")) +  
 geom\_boxplot(aes(x = "3\_painty", y = painty), fill = I("yellow")) +  
 geom\_boxplot(aes(x = "4\_potato", y = potato), fill = I("green")) +  
 geom\_boxplot(aes(x = "5\_rancid", y = rancid), fill = I("blue")) +  
 xlab("variable") + ylab("rating")



## Tidy Data

Data which follows these rules is "tidy" and easier to work with in R:

* Each variable in the data set is placed in its own column
* Each observation is placed in its own row
* Each value is placed in its own cell

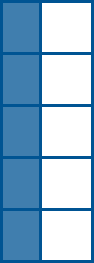
### Ideal Data

The french fry data is in **wide format**

Wide Format

Wide Format

and should be in **long format** for plotting



Long Format

This is done using the gather function.

### Gathering

When gathering, specify the **keys** (identifiers) and the **values** (measures).

#### Keys/Identifiers:

* Identify a record (must be unique)
* Example: Indices on an random variable
* Fixed by design of experiment (known in advance)
* May be single or composite (may have one or more variables)

#### Values/Measures:

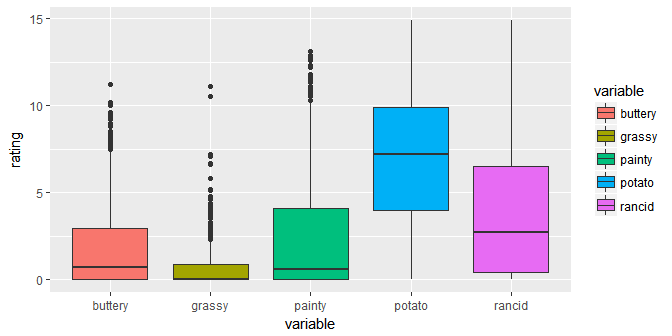
* Collected during the experiment (not known in advance)
* Usually numeric quantities

#### French Fry Data

library(tidyr)  
  
french\_fries\_long <- gather(french\_fries,   
 key = variable,   
 value = rating,   
 potato:painty)  
  
head(french\_fries\_long)  
## time treatment subject rep variable rating  
## 1 1 1 3 1 potato 2.9  
## 2 1 1 3 2 potato 14.0  
## 3 1 1 10 1 potato 11.0  
## 4 1 1 10 2 potato 9.9  
## 5 1 1 15 1 potato 1.2  
## 6 1 1 15 2 potato 8.8

Plotting is much easier with long format data:

qplot(variable, rating, data = french\_fries\_long,   
 fill = variable, geom = "boxplot")



## Spread

In certain applications, a wide dataset may be preferable (e.g. to display in a table).

head(french\_fries\_long)  
## time treatment subject rep variable rating  
## 1 1 1 3 1 potato 2.9  
## 2 1 1 3 2 potato 14.0  
## 3 1 1 10 1 potato 11.0  
## 4 1 1 10 2 potato 9.9  
## 5 1 1 15 1 potato 1.2  
## 6 1 1 15 2 potato 8.8

The spread function (from tidyr) is used to do this:

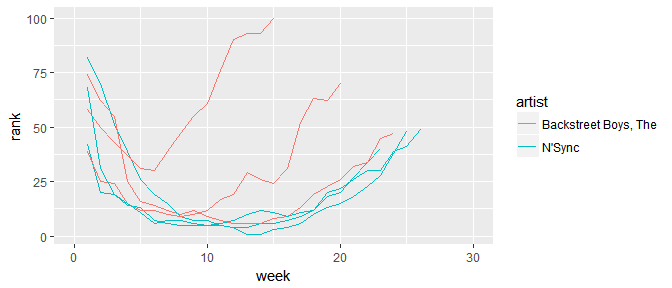
french\_fries\_wide <- spread(french\_fries\_long,   
 key = variable, value = rating)  
  
head(french\_fries\_wide)  
## time treatment subject rep buttery grassy painty potato rancid  
## 1 1 1 3 1 0.0 0.0 5.5 2.9 0.0  
## 2 1 1 3 2 0.0 0.0 0.0 14.0 1.1  
## 3 1 1 10 1 6.4 0.0 0.0 11.0 0.0  
## 4 1 1 10 2 5.9 2.9 0.0 9.9 2.2  
## 5 1 1 15 1 0.1 0.0 5.1 1.2 1.1  
## 6 1 1 15 2 3.0 3.6 2.3 8.8 1.5

##### Your Turn

Read in the billboard top 100 music data

billboard <- read.csv("https://bit.ly/2hQGDao")

1. Use tidyr to convert this data into a long format.
2. Use ggplot2 to create this time series plot:



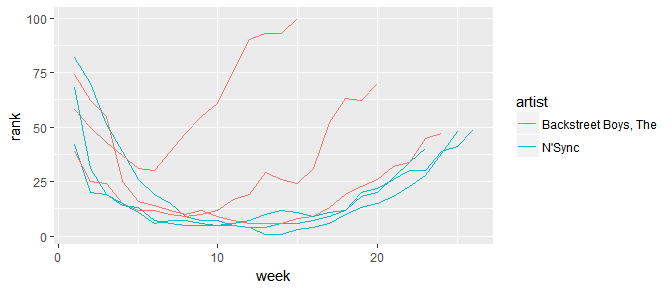
##### Solutions

1. Use tidyr to convert this data into a long format.

long\_billboard <- gather(billboard,   
 key = week, value = rank,   
 X1:X76)  
  
# Convert weeks to numeric variables  
long\_billboard$week <- long\_billboard$week %>%  
 gsub("X", "", .) %>%  
 as.numeric()  
  
# Get rid of NAs:  
long\_billboard <- long\_billboard %>% na.omit()

1. Use ggplot2 to create this time series plot:

qplot(x = week, y = rank,   
 colour = artist, group = track,   
 data = long\_billboard, geom = "line")



###### End Solutions

## Split-Apply-Combine

* *Split* a dataset into many smaller sub-datasets
* *Apply* some function to each sub-dataset
* *Combine* the results of the function calls  
  into one dataset

### Using dplyr

library(dplyr)  
  
french\_fries\_rating <- french\_fries\_long %>%  
 # SPLIT:  
 group\_by(variable) %>%   
 # APPLY + COMBINE:  
 summarize(rating = mean(rating, na.rm = T))   
  
french\_fries\_rating  
## # A tibble: 5 × 2  
## variable rating  
## <chr> <dbl>  
## 1 buttery 1.8236994  
## 2 grassy 0.6641727  
## 3 painty 2.5217579  
## 4 potato 6.9525180  
## 5 rancid 3.8522302

## dplyr verbs

There are five primary dplyr **verbs**, representing distinct data analysis tasks:

* filter: Keep only a subset of a data frame
* arrange: Reorder the rows of a data frame
* select: Select particular columns of a data frame
* mutate: Add new columns computed from existing cols
* summarize: Create collapsed summaries of a data frame

### Filter

french\_fries %>%  
 filter(subject == 3, time == 1)  
## time treatment subject rep potato buttery grassy rancid painty  
## 1 1 1 3 1 2.9 0.0 0.0 0.0 5.5  
## 2 1 1 3 2 14.0 0.0 0.0 1.1 0.0  
## 3 1 2 3 1 13.9 0.0 0.0 3.9 0.0  
## 4 1 2 3 2 13.4 0.1 0.0 1.5 0.0  
## 5 1 3 3 1 14.1 0.0 0.0 1.1 0.0  
## 6 1 3 3 2 9.5 0.0 0.6 2.8 0.0

### Arrange

french\_fries %>%  
 arrange(desc(rancid)) %>%  
 head  
## time treatment subject rep potato buttery grassy rancid painty  
## 1 9 2 51 1 7.3 2.3 0 14.9 0.1  
## 2 10 1 86 2 0.7 0.0 0 14.3 13.1  
## 3 5 2 63 1 4.4 0.0 0 13.8 0.6  
## 4 9 2 63 1 1.8 0.0 0 13.7 12.3  
## 5 5 2 19 2 5.5 4.7 0 13.4 4.6  
## 6 4 3 63 1 5.6 0.0 0 13.3 4.4

### Select

french\_fries %>%  
 select(time, treatment, subject, rep, potato) %>%  
 head  
## time treatment subject rep potato  
## 1 1 1 3 1 2.9  
## 2 1 1 3 2 14.0  
## 3 1 1 10 1 11.0  
## 4 1 1 10 2 9.9  
## 5 1 1 15 1 1.2  
## 6 1 1 15 2 8.8

### Mutate

french\_fries %>%  
 mutate(rancid2 = rancid^2) %>%  
 select(time, treatment, subject, rancid, rancid2) %>%  
 head  
## time treatment subject rancid rancid2  
## 1 1 1 3 0.0 0.00  
## 2 1 1 3 1.1 1.21  
## 3 1 1 10 0.0 0.00  
## 4 1 1 10 2.2 4.84  
## 5 1 1 15 1.1 1.21  
## 6 1 1 15 1.5 2.25

### Summarize

french\_fries %>%  
 group\_by(time, treatment) %>%  
 summarize(mean\_rancid = mean(rancid),   
 sd\_rancid = sd(rancid))  
## Source: local data frame [30 x 4]  
## Groups: time [?]  
##   
## time treatment mean\_rancid sd\_rancid  
## <int> <int> <dbl> <dbl>  
## 1 1 1 2.758333 3.212870  
## 2 1 2 1.716667 2.714801  
## 3 1 3 2.600000 3.202037  
## 4 2 1 3.900000 4.374730  
## 5 2 2 2.141667 3.117540  
## 6 2 3 2.495833 3.378767  
## 7 3 1 4.650000 3.933358  
## 8 3 2 2.895833 3.773532  
## 9 3 3 3.600000 3.592867  
## 10 4 1 2.079167 2.394737  
## # ... with 20 more rows

##### Your Turn

This dataset contains information on over 300,000 flights that departed from New York City in the year 2013.

flights <- read.csv("https://bit.ly/2hzhAfW")

1. Using dplyr and the pipe operator (%>%), create a data frame consisting of the average arrival delay (arr\_delay) based on the destination airport (dest).  
   Sort this data frame in descending order, so the destination airport with the largest delay is first.
2. Find out the most used airports for each airline carrier.

##### Solutions

1. Create a data frame consisting of the average arrival delay (arr\_delay) based on the destination airport (dest).  
   Sort this data frame in descending order, so the destination airport with the largest delay is first.

flights %>%   
 group\_by(dest) %>%   
 summarize(avg\_delay = mean(na.omit(arr\_delay))) %>%   
 arrange(desc(avg\_delay))  
## # A tibble: 105 × 2  
## dest avg\_delay  
## <fctr> <dbl>  
## 1 CAE 41.76415  
## 2 TUL 33.65986  
## 3 OKC 30.61905  
## 4 JAC 28.09524  
## 5 TYS 24.06920  
## 6 MSN 20.19604  
## 7 RIC 20.11125  
## 8 CAK 19.69834  
## 9 DSM 19.00574  
## 10 GRR 18.18956  
## # ... with 95 more rows

1. Find out the most used airports for each airline carrier.

by.origin <- flights %>%   
 group\_by(carrier, origin) %>%   
 summarize(count = n()) %>%   
 ungroup() %>% group\_by(carrier) %>%   
 filter(count == max(count)) %>%   
 rename(mostused = origin) %>%  
 mutate(kind = "origin")  
  
by.dest <- flights %>%   
 group\_by(carrier, dest) %>%   
 summarize(count = n()) %>%   
 ungroup() %>% group\_by(carrier) %>%   
 filter(count == max(count)) %>%   
 rename(mostused = dest) %>%   
 mutate(kind = "dest")  
  
mostused <- bind\_rows(by.origin,by.dest)  
## Warning in bind\_rows\_(x, .id): Unequal factor levels: coercing to character  
  
mostused %>% group\_by(carrier) %>%   
 filter(count == max(count)) %>% ungroup() %>%  
 arrange(desc(count))   
## # A tibble: 19 × 4  
## carrier mostused count kind  
## <fctr> <chr> <int> <chr>  
## 1 UA EWR 46087 origin  
## 2 EV EWR 43939 origin  
## 3 B6 JFK 42076 origin  
## 4 DL LGA 23067 origin  
## 5 MQ LGA 16928 origin  
## 6 AA LGA 15459 origin  
## 7 9E JFK 14651 origin  
## 8 US LGA 13136 origin  
## 9 WN EWR 6188 origin  
## 10 VX JFK 3596 origin  
## 11 FL LGA 3260 origin  
## 12 AS EWR 714 origin  
## 13 AS SEA 714 dest  
## 14 F9 LGA 685 origin  
## 15 F9 DEN 685 dest  
## 16 YV LGA 601 origin  
## 17 HA JFK 342 origin  
## 18 HA HNL 342 dest  
## 19 OO LGA 26 origin

###### End Solutions

## Dates and Times

Dates are deceptively hard to work with.

**Example**: 02/05/2012. Is it February 5th, or May 2nd?

Other things are difficult too:

* Time zones
* POSIXct format in base R is challenging

The **lubridate** package helps tackle some of these issues.

### Basic Lubridate Use

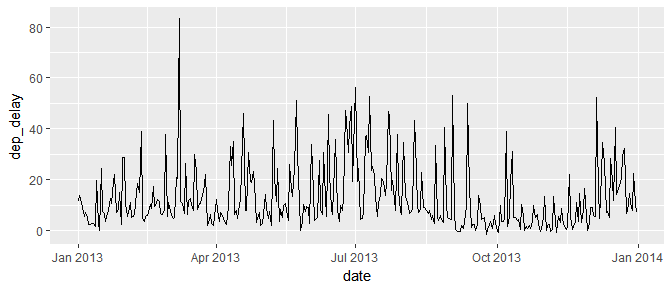
library(lubridate)  
  
now()  
today()  
now() + hours(4)  
today() - days(2)  
## [1] "2017-01-03 10:09:42 CST"  
## [1] "2017-01-03"  
## [1] "2017-01-03 14:09:42 CST"  
## [1] "2017-01-01"

### Parsing Dates

ymd("2013-05-14")  
mdy("05/14/2013")  
dmy("14052013")  
ymd\_hms("2013:05:14 14:50:30", tz = "America/Chicago")  
## [1] "2013-05-14"  
## [1] "2013-05-14"  
## [1] "2013-05-14"  
## [1] "2013-05-14 14:50:30 CDT"

##### Your Turn

1. Using the flights data, create a new column Date using lubridate. Paste together the columns year, month, and day in order to do this. See the paste function.
2. Use dplyr to calculate the average departure delay for each date.
3. Plot the date versus the average departure delay



##### Solutions

1. Using the flights data, create a new column Date using lubridate.

flights$date <- paste(flights$year, flights$month, flights$day,   
 sep = "-") %>%  
 ymd() # Convert from character to date format

1. Use dplyr to calculate the average departure delay for each date.

delay.dat <- flights %>%   
 group\_by(date) %>%   
 summarize(dep\_delay = mean(dep\_delay, na.rm = TRUE))

1. Plot the date versus the average departure delay

qplot(date, dep\_delay, geom = "line", data = delay.dat)

