Data Wrangling

Preparing Your Data for Analysis

Data Wrangling

- Data wrangling is the process of going from messy data to data that can be analyzed

- It's not always fun but tools in *R* help to avoid a lot of headaches

Goal of Tidying

When *tidying* our goal is to end up with a row-by-column structure of our data, that has clearly named variables and valid values.

Stat1	Stat2	Stat3
30	101	0.530
32	111	0.670
19	131	0.499

Tidying Unstructured Data

Tidying Unstructured Data

- When scraping Web data, as we often do in sport, the data can be messy.

Tidying Unstructured Data

- When scraping Web data, as we often do in sport, the data can be messy.
- It is typical to need some programming to get the data into a nice row by column structure

Tidying Unstructured Data

- When scraping Web data, as we often do in sport, the data can be messy.
- It is typical to need some programming to get the data into a nice row by column structure
- String manipulation is a common task in this processed and can be tackled with the `stringr` package

Common Data Wrangling Steps

1. Manipulating strings
2. Selecting
3. Transforming
4. Reshaping
5. Validity check

Example: Match Statistics

Recall the example from the Zverev v Djokovic tennis match that we pulled from the ATP site using `RSelenium`. The extracted data is in a single string, so it is unstructured and needs to be tidied up.

Let's store those results in the object `match_stats`.

```
match_stats
```

```
[1] "ALEXANDER\nZVEREV\nNOVAK\nDJOKOVIC\nMATCH STATS YTD STATS\nSERVICE STATS\n7\nAces
\n1\n2\nDouble Faults\n3\n71%\n(32/45)\n1st Serve\n64%\n(43/67)\n84%\n(27/32)\n1st
Serve Points Won\n70%\n(30/43)\n69%\n(9/13)\n2nd Serve Points Won\n38%\n(9/24)\n0%
\n(0/0)\nBreak Points Saved\n40%\n(2/5)\n9\nService Games Played\n10\nRETURN STATS\n30%
\n(13/43)\n1st Serve Return Points Won\n16%\n(5/32)\n63%\n(15/24)\n2nd Serve Return
Points Won\n31%\n(4/13)\n60%\n(3/5)\nBreak Points Converted\n0%\n(0/0)\n10\nReturn
Games Played\n9\nPOINTS STATS\n80%\n(36/45)\nReturn Points Won\n58%\n(39/67)\n42%
\n(28/67)\nTotal Return Points Won\n20%\n(9/45)\n57%\n(64/112)\nTotal Points Won\n43%
\n(48/112)"
```

What Is Needed?

To get this string of match statistics into a data frame we can work with, we need to:

What Is Needed?

To get this string of match statistics into a data frame we can work with, we need to:

1. Identify the target structure (that is, variables and value types)
2. Split the string into the different variables
3. Extract values
4. Assign to variables in a data.frame
5. Convert values to appropriate types

Example: Target Structure

We have a set of statistics for each player. One option is a "long"" format with the following structure:

Statistic	Value	Player

Question: Target Structure

Suppose we instead wanted a "wide" format. How would that differ?

Question: Target Structure

Suppose we instead wanted a "wide" format. How would that differ?

Player	Stat 1	Stat 2	....
1			
2			

Example: Splitting

```r
library(stringr) # Load stingr

str_split(match_stats, "\n") # Split on return characters
```

```
## [[1]]
##  [1] "ALEXANDER"                   "ZVEREV"
##  [3] "NOVAK"                       "DJOKOVIC"
##  [5] "MATCH STATS YTD STATS"       "SERVICE STATS"
##  [7] "7"                           "Aces"
##  [9] "1"                           "2"
## [11] "Double Faults"               "3"
## [13] "71%"                         "(32/45)"
## [15] "1st Serve"                   "64%"
## [17] "(43/67)"                     "84%"
## [19] "(27/32)"                     "1st Serve Points Won"
## [21] "70%"                         "(30/43)"
## [23] "69%"                         "(9/13)"
## [25] "2nd Serve Points Won"        "38%"
## [27] "(9/24)"                      "0%"
## [29] "(0/0)"                       "Break Points Saved"
## [31] "40%"                         "(2/5)"
## [33] "9"                           "Service Games Played"
```

Group Data by Pattern

- Now that we have isolated some of the main elements of our data as a vector, we want to group data by type.
- We can use pattern-matching to separate strings by their pattern
- Several useful `stringr` packages for pattern matching include:

Group Data by Pattern

- Now that we have isolated some of the main elements of our data as a vector, we want to group data by type.
- We can use pattern-matching to separate strings by their pattern
- Several useful `stringr` packages for pattern matching include:

```
str_detect(x, pattern) # Test each element for presence of pattern
str_subset(x, pattern) # Subset x by where pattern is found
str_extract(x, pattern) # Extracts first occurrence of pattern
```

Regular Expressions

- By default, the `pattern` is assumed to be a *regular expression*.
- A *regular expression* describes a pattern in a string and is very powerful for pattern-finding.
- Find more about `regex` in R here

I want to find string patterns that include...	Regular Expression
Any single uppercase letter from A to Z	[A-Z
followed by	] (close the character class)
Any single lowercase letter from a to z	[a-z
followed by	]
Any single lowercase letter from a to z	[a-z
followed by	]
Any single digit from 0 to 9	[0-9
followed by	]
Any single lowercase vowel	[aeiou
(close the last character class)	]

Example: Using RegEx to Sort Data

Looking at our example, we can separate the stats by using a pattern that finds elements with at least one lower-case letter

```r
split <- str_split(match_stats, "\n")[[1]] # Save split vector

pattern <- "[a-z]"

stats <- str_subset(split, pattern) # Subset players and stat names
```

Example: Using RegEx to Sort Data

We use exclusion to get all the other values

```
values <- split[
    !str_detect(split, pattern) &
    !str_detect(split, "[A-Z]")
 ] # Get values
```

Practice: Using RegEx to Sort Data

There are a number of other ways we could isolate the statistic values from the other content of our string.

Find an alternative.

Practice: Using RegEx to Sort Data

There are a number of other ways we could isolate the statistic values from the other content of our string.

Find an alternative.

```
values <- str_subset(split, "[[0-9]\\)\\%]$")
```

Question: Using RegEx to Sort Data

Why didn't we just use the `[0-9]` regular expression to isolate the statistic values?

Question: Using RegEx to Sort Data

Why didn't we just use the `[0-9]` regular expression to isolate the statistic values?

Because the name of some statistics includes numbers, this wouldn't isolate the values.

```
str_subset(split, "[0-9]")
```

```
##  [1] "7"                            "1"
##  [3] "2"                            "3"
##  [5] "71%"                          "(32/45)"
##  [7] "1st Serve"                    "64%"
##  [9] "(43/67)"                      "84%"
## [11] "(27/32)"                      "1st Serve Points Won"
## [13] "70%"                          "(30/43)"
## [15] "69%"                          "(9/13)"
## [17] "2nd Serve Points Won"         "38%"
## [19] "(9/24)"                       "0%"
## [21] "(0/0)"                        "40%"
## [23] "(2/5)"                        "9"
## [25] "10"                           "30%"
## [27] "(13/43)"                      "1st Serve Return Points Won"
```

Example: Structuring Data Frame

We notice that some stats have just counts while others have percentages and ratios. We can deal with this by flagging counts and expanding the data frame based on the condition of being a count or percentage stat.

```
counts <- stats %in% c("Aces",
        "Double Faults",
        "Service Games Played",
        "Return Games Played")

data.frame(
    stat = rep(stats, ifelse(counts, 2, 4)),
    values = values
)
```

```
##                            stat   values
## 1                          Aces        7
## 2                          Aces        1
## 3                 Double Faults        2
## 4                 Double Faults        3
## 5                     1st Serve      71%
## 6                     1st Serve  (32/45)
## 7                     1st Serve      64%
```

Example: String Substitution

We will need to do some more tidying of the strings to get our `value` column into numeric values. String replace will be a big help. Here are some examples of removing percentage signs and parentheses using `str_replace`.

```
# We use 'all' to replace all instances
# The escapes \\ make sure () are treated as fixed
str_replace_all(values,"[\\(%\\)]", "")
```

```
##  [1] "7"      "1"      "2"      "3"      "71"     "32/45"  "64"
##  [8] "43/67"  "84"     "27/32"  "70"     "30/43"  "69"     "9/13"
## [15] "38"     "9/24"   "0"      "0/0"    "40"     "2/5"    "9"
## [22] "10"     "30"     "13/43"  "16"     "5/32"   "63"     "15/24"
## [29] "31"     "4/13"   "60"     "3/5"    "0"      "0/0"    "10"
## [36] "9"      "80"     "36/45"  "58"     "39/67"  "42"     "28/67"
## [43] "20"     "9/45"   "57"     "64/112" "43"     "48/112"
```

Practice: String Substitution

1. Use the `str_replace_all` function to prepare the `valuea` column of our data set for numeric conversion

2. Convert the values to numeric

3. Check that the first serve percentage won matches the proportion from the ratio form

Solution: String Substitution

```
match_stats <- data.frame(
    stat = rep(stats, ifelse(counts, 2, 4)),
    values = str_replace_all(values,"[\\(%\\)]", ""),
    stringsAsFactors = FALSE
)

match_stats <- match_stats %>%
  rowwise() %>%
  dplyr::mutate(
    values = ifelse(!str_detect(values, "/"), as.numeric(values),
      "/"(as.numeric(str_extract_all(values, "[0-9]+")[[1]])[1],
          as.numeric(str_extract_all(values, "[0-9]+")[[1]])[2]))
  )

subset(match_stats, stat == "1st Serve Points Won")
```

```
## # A tibble: 4 x 2
##                    stat     values
##                   <chr>      <dbl>
## 1 1st Serve Points Won 84.0000000
## 2 1st Serve Points Won  0.8437500
## 3 1st Serve Points Won 70.0000000
## 4 1st Serve Points Won  0.6976744
```

Tidying Structured Data

Sometimes we get data in a row by column format but there are still problems with data values. Some common issues with sports data are:

- Untidy strings
- Incorrect class
- Missing values
- Hidden missing values
- Bad labelling
- Transforming dates
- Alternative names/Misspelling

Manipulating Structured Data

- Many of the tools we need when working with data in `data.frames` come from the `dplyr` package.
- `dplyr` provides a grammar for data manipulation
- Install with the following command:

```
library(devtools)

install_github("hadley/dplyr") # Install dev version
```

Tools of `dplyr`

This is an overview of `dplyr` tools. We will apply these throughout the remainder of the tutorial.

Tool	Description
select	Column subsetting
filter	Row subsetting
mutate	Transform or create variables
summarise	Summarise variables (i.e., many values to one)
group_by	Apply tools by grouping variables
%>%	Pipe operator for chaining multiple commands

Reshaping Structured Data

- Sometimes we need to do more than change individual columns and rows
- When we want to *reshape* the structure of our data we can use `tidyr`
- The `tidyr` package provides a grammar for data reshaping

```r
library(devtools)
install_github("hadley/tidyr") # Install dev version
```

Tools of `tidyr`

This is an overview of `tidyr` tools. Like `dplyr`, we will illustrate these `tidyr` tools as we go through the tutorial.

Tool	Description
gather	Takes multiple columns, and gathers them into key-value pairs. Goes from wide to long format.
spread	Takes key-value pair and spreads them in to multiple columns. This goes from long to wide format.
separate	Breaks up a single column into multiple.

Objective: Scoring Surprising Event Results

- We will walk through a number of common tidying steps using a real-world example
- Suppose we want to measure which player had the most surprising Australian Open performance in the past 3 years
- Let's use the match result info from www.tennis-data.co.uk to try to get at this question

[1] Denis Istomin after upset of Novak Djokovic at 2017 AO

Importing the Data

- First, we need to read-in the data from the site for the years 2015 to 2017
- Each year is stored in a separate file with a URL that has the following pattern:

```
"http://www.tennis-data.co.uk/year/ausopen.csv"
```

Practice: String and Import

How would you use the URL pattern to get a single data frame of the results for AOs 2015 to 2017?

Practice: String and Import

How would you use the URL pattern to get a single data frame of the results for AOs 2015 to 2017?

Answer:

```
url <- "http://www.tennis-data.co.uk/year/ausopen.csv"

years <- sapply(2015:2017, function(x) sub("year", x, url))

data <- do.call("rbind", lapply(years, read.csv))
```

Checking for Messy Data

Checking for Messy Data

- With any data directly from the Web we need to be on guard for some messiness

Checking for Messy Data

- With any data directly from the Web we need to be on guard for some messiness
- The first step to diagnosing the messy issues, is to inspect each variable in the dataset.

Checking for Messy Data

- With any data directly from the Web we need to be on guard for some messiness
- The first step to diagnosing the messy issues, is to inspect each variable in the dataset.
- I recommend separating characters and numeric.

Checking for Messy Data

- With any data directly from the Web we need to be on guard for some messiness
- The first step to diagnosing the messy issues, is to inspect each variable in the dataset.
- I recommend separating characters and numeric.
- Sort and look at unique values for character type

Checking for Messy Data

- With any data directly from the Web we need to be on guard for some messiness
- The first step to diagnosing the messy issues, is to inspect each variable in the dataset.
- I recommend separating characters and numeric.
- Sort and look at unique values for character type
- Use summary on each numeric type

Inspect Classes

The code below evaluates the classes in our dataset. What do we conclude from this?

```r
# Check classes
table(apply(data, 2, class))
```

```
##
## character
##        40
```

Note: All character classes should make you suspect some need for class conversion

Inspect Variables

To learn more about the contents of each variable and any issues we need to address, I like to use the `ask` function of `gtools`. Here is how we can use it to inspect variables one at a time.

```
library(gtools) # For ask function

# Inspection loop
for(name in names(data)){
    print(name)
    print(sort(unique(data[,name])))
    ask()
}
```

Practice: Inspect Variables

Complete the inspection step in the previous slide. Determine:

1. What variables does the dataset contain?
2. Which variables are relevant to measuring event surprising event outcomes by player and year?
3. Are there any issues with those variables we need to address?

Solution: Inspect Variables

- The dataset contains winner and loser info for each match along with the pre-match Odds by several different bookmakers
- The 'Date', 'Winner', and one or more of the odds will be used to measure a player's event performance
- We need to create a 'Year' variable, check for duplicates/variants in spelling among Winner names, create a 'surprise score' for each win, and filter out 'Retirements' matches

Solution: Inspect Variables

- The dataset contains winner and loser info for each match along with the pre-match Odds by several different bookmakers
- The 'Date', 'Winner', and one or more of the odds will be used to measure a player's event performance
- We need to create a 'Year' variable, check for duplicates/variants in spelling among Winner names, create a 'surprise score' for each win, and filter out 'Retirements' matches

Let's get started...

Date Conversion

- Since we often will want to perform calculations with dates, we should convert them to a `Date` object. This is easy to do using the `lubridate` package.
- `lubridate` has conversion functions that are named according to the format of our input.

Function	Example
dmy	3/2/99
mdy	12302017
ymd	1981-10-21

Note that the delimiter is generally unimportant.

Convert Date and Make Year

In this code we will use `lubridate` and `dplyr` to convert the `Date` variable and create the variable `Year`.

```
library(dplyr) # dplyr for data manipulation
library(lubridate) # date manipulation
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```
data <- data %>%
    dplyr::mutate(
        Date  = dmy(Date),
        Year = year(Date)
    )
```


Note: `%>%` is a piping operator

Surprise Score

What should define a surprising event performance?

- A string of big upsets is one definition
- We can use the bookmaker odds to measure what a player was expected to do and compare that against their actual wins
- A more surprising result is one that exceeds expectations
- The sum of this 'surprise score' over all of a player's wins will be their event 'Surprise Score'

Which Odds?

There are several odds to choose from. Without knowing more about the bookmakers, we will choose one of the odds that is most complete across matches. To do this, we need to check for missing values.

```
data %>%
    select(B365W:AvgL) %>%
    summarise_all(
        funs(sum(is.na(.)))
    )
```

```
##   B365W B365L EXW EXL LBW LBL PSW PSL MaxW MaxL AvgW AvgL
## 1     0     0   1   1   1   1   1   1    0    0    0    0
```

Create Suprise Score

We will use the `B365W` odds for the winner and do the simple inversion to estimate the expected win chances for the player.

```
data <- data %>%
    dplyr::mutate(
        SurpriseScore = 1 - 1 / as.numeric( B365W)
    )
```

[1] Remember we needed to convert to numeric before making this calculation

Cleaning Names

- Before we can summarise results by `Winner` we need to check the validity of the names

- One of the most troublesome issues with sports data are inconsistent naming of players. This is a problem when you need to assign performance measures to the same individual, based on their name.

- Some of the "inconsistencies" you have to be prepared for are:

    - Misspellings
    - Differences in punctuation
    - Middle names
    - Multiple surnames
    - Abbreviations

Approximate `grep`

The `agrep` function performs approximate matching, and is a *very* useful function for cleaning up names in sports data. It looks at the distance between the input `x` and a `pattern`, using the Levenshtein edit distance.

```
agrep(pattern, x, max.distance = 0.1, costs = NULL, ...)
```

Most of the arguments are like the usual `grep` except for two: `max.distance` and `costs`.

`max.distance`: Numeric for the maximal distance

`costs`: Numeric cost for the Levenshtein edit distance

Example `agrep`

Here we look for possible inconsistencies in the `Winner` variable using `agrep`.

```
players <- sort(unique(as.character(data$Winner))) # Get unique playe

approx <- lapply(players, agrep, fixed = T, x = players)
# Compare each player against all others

players[sapply(approx, length) > 1]
```

```
## [1] "Bautista R." "Lopez F."    "Zverev A."   "Zverev M."
```

```
# Look for cases with multiple matches
```

Practice: Cleaning Names

Based on the results from our inspection in the previous slide, which changes do you think are needed to the `Winner` variable?

Practice: Cleaning Names

Based on the results from our inspection in the previous slide, which changes do you think are needed to the `Winner` variable?

Answer:

```
data$Winner[data$Winner == "Bautista Agut R."] <- "Bautista R."
```

Total Surprise Score

We are now ready to compute a total surprise score for each player and year. Using `summarise`, find the top 10 most surprising performances in the past 3 years.

Total Surprise Score

We are now ready to compute a total surprise score for each player and year. Using `summarise`, find the top 10 most surprising performances in the past 3 years.

```
summarise_wins <- data %>%
    filter(Comment != "Retired") %>% # Remember to remove retirements
    group_by(Year, Winner) %>%
    dplyr::summarise(
        TotalSurpriseScore = sum(SurpriseScore)
    )

summarise_wins[order(summarise_wins$TotalSurpriseScore, decreasing =
```

What About Losses?

What About Losses?

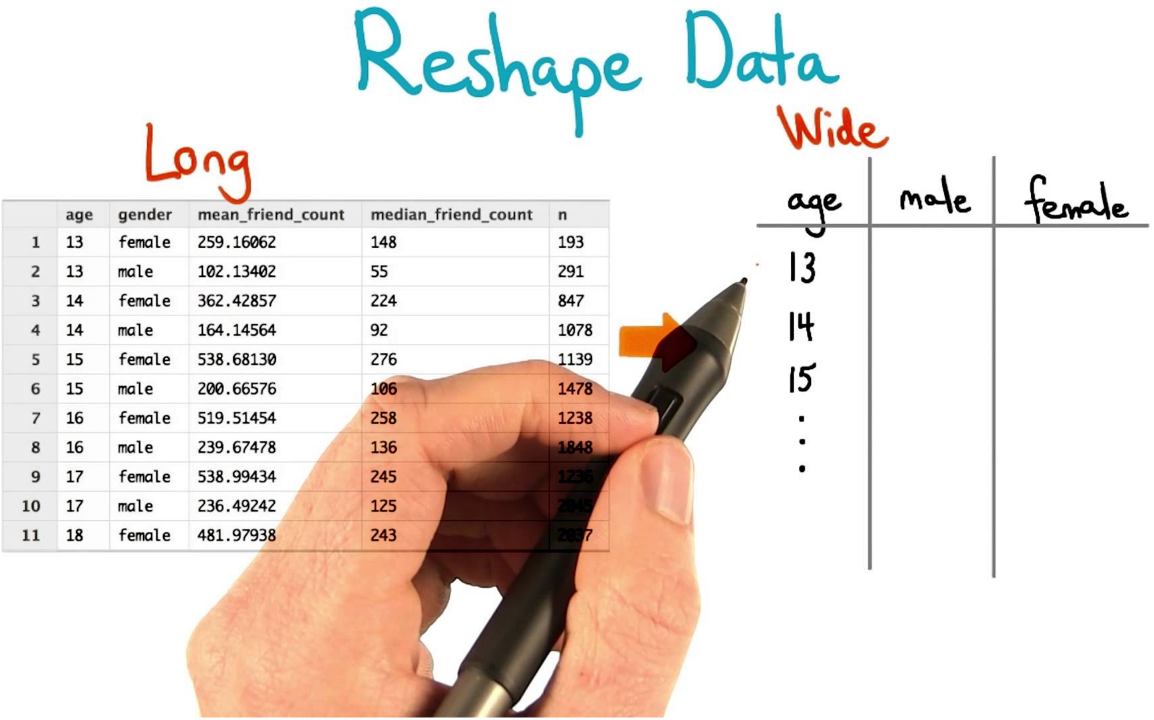
- Suppose we had wanted a score based on wins and losses, what would we need to do this?

What About Losses?

- Suppose we had wanted a score based on wins and losses, what would we need to do this?
- We would need to switch to a *long* format which means reshaping our data

Reshaping Data

In addition to transforming individual variables, we often will want to reshape our data from wide to long or long to wide formats.



	age	gender	mean_friend_count	median_friend_count	n
1	13	female	259.16062	148	193
2	13	male	102.13402	55	291
3	14	female	362.42857	224	847
4	14	male	164.14564	92	1078
5	15	female	538.68130	276	1139
6	15	male	200.66576	106	1478
7	16	female	519.51454	258	1238
8	16	male	239.67478	136	1848
9	17	female	538.99434	245	1236
10	17	male	236.49242	125	[illegible]
11	18	female	481.97938	243	[illegible]

Going from Wide to Long

To go from wide to long format, we can use the `tidyr` `gather` function. Here is an example.

```r
library(tidyr) # load tidyr for reshaping

data %>%
  gather("name", "value", x1, x2, x3)
```

Going from Wide to Long

To go from wide to long format, we can use the `tidyr` `gather` function. Here is an example.

```
library(tidyr) # load tidyr for reshaping

data %>%
  gather("name", "value", x1, x2, x3)
```

In the above, we stack the variables `x1`, `x2` and `x3`, creating a categorical variable `name` with the variable names and the column `value` with all of the grouped values.

Going from Long to Wide

To go from long to wide format, we can use the `tidyr` `spread` function. Here is an example.

```
data %>%
  spread(key = name, value)
```

Going from Long to Wide

To go from long to wide format, we can use the `tidyr` `spread` function. Here is an example.

```
data %>%
  spread(key = name, value)
```

In this example we undo with long format by providing the set of new columns to create with the variable supplied to `key`.

The values that will be inserted into those columns is indicated with the `value` variable.

Practice: Reshaping

Use the `tidyr` package to create a long format of our data that groups `Winner` and `Loser` into a single `player` column.

Practice: Reshaping

Use the `tidyr` package to create a long format of our data that groups `Winner` and `Loser` into a single `player` column.

```
data <- data %>%
  gather("Outcome", "Player", Winner, Loser)
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
head(data[,c("Date", "Outcome", "Player")])
```

```
##         Date Outcome       Player
## 1 2015-01-19  Winner  Berankis R.
## 2 2015-01-19  Winner  Dimitrov G.
## 3 2015-01-19  Winner  Anderson K.
## 4 2015-01-19  Winner    Chardy J.
## 5 2015-01-19  Winner     Lacko L.
## 6 2015-01-19  Winner Matosevic M.
```

Resources

- dplyr
- tidyr
- lubridate
- agrep
- regex