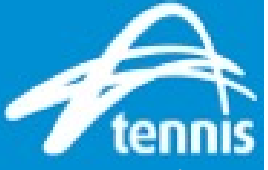
Getting Sports Data

Inspecting and Scraping Data from the Web



What Next?

If you've found the data you want on the Web, what comes next?

What Next?

If you've found the data you want on the Web, what comes next?

If you want to capture these or similar data repeatedly, you will want to retrieve it in a reproducible way.

Ways of Getting Sports Data

- Not always fun but a necessary part of sports analysis
- There are two major ways to get data from Web:

    1. Import a file directly

    2. Extract from HTML

Example: Import Data File

- Files that can be read with `read.table` or related functions can be directly imported from a URL.
- Here we extract the most recent Australian Open match results and betting odds using `read.csv`.

```
url <- "http://www.tennis-data.co.uk/2017/ausopen.csv"

read.csv(url)
```

Practice: URL Patterns & Importing

Consider the previous example. If we wanted the same data for the 2016 US Open, how do you think we could do that?

Practice: URL Patterns & Importing

Consider the previous example. If we wanted the same data for the 2016 US Open, how do you think we could do that?

1. Test a possible URL for the 2016 US Open
2. Import the file
3. Run a `str` on the dataset to determine what info it contains

Solution: URL Patterns & Importing

Changing the year and tournament names in the URL are enough to get the correct file.

```
url <- "http://www.tennis-data.co.uk/2016/usopen.csv"

usopen <- read.csv(url)
```

When we look at the variables, we find it contains the scores and betting odds for all 127 main draw matches for the US Open.

```
str(usopen) # Data contents
```

Scraping from a Website

If you can't directly import data from the Web--which is usually the case--you can still capture the data but you need to know whether it is *static* or *dynamic* data.

Scraping from a Website

If you can't directly import data from the Web--which is usually the case--you can still capture the data but you need to know whether it is *static* or *dynamic* data.

What is *static* data?

What is *dyanmic* data?

And how do you determine which type of data you have?

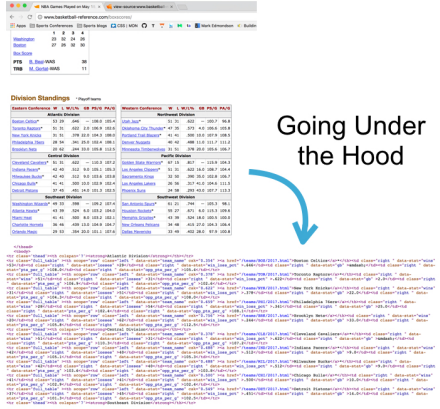
Static vs. Dynamic Data

We use different methods to get Web data depending on which type it is, so it is very important to be able to identify each type.

- *Static* web data is data you can see in the source code.
- If you can't see the data, the data is *dynamic*.

How Do I Know if Data is Static or Dynamic?

- You need to be able to inspect HTML and CSS
- This means being able to "View Source"
- Being able to identify CSS elements in source.



Web Developer Tools

Every modern browser has a suite of "developer tools". These include useful functions for Web scrapers, including:

1. Viewing the source code
2. Inspecting elements

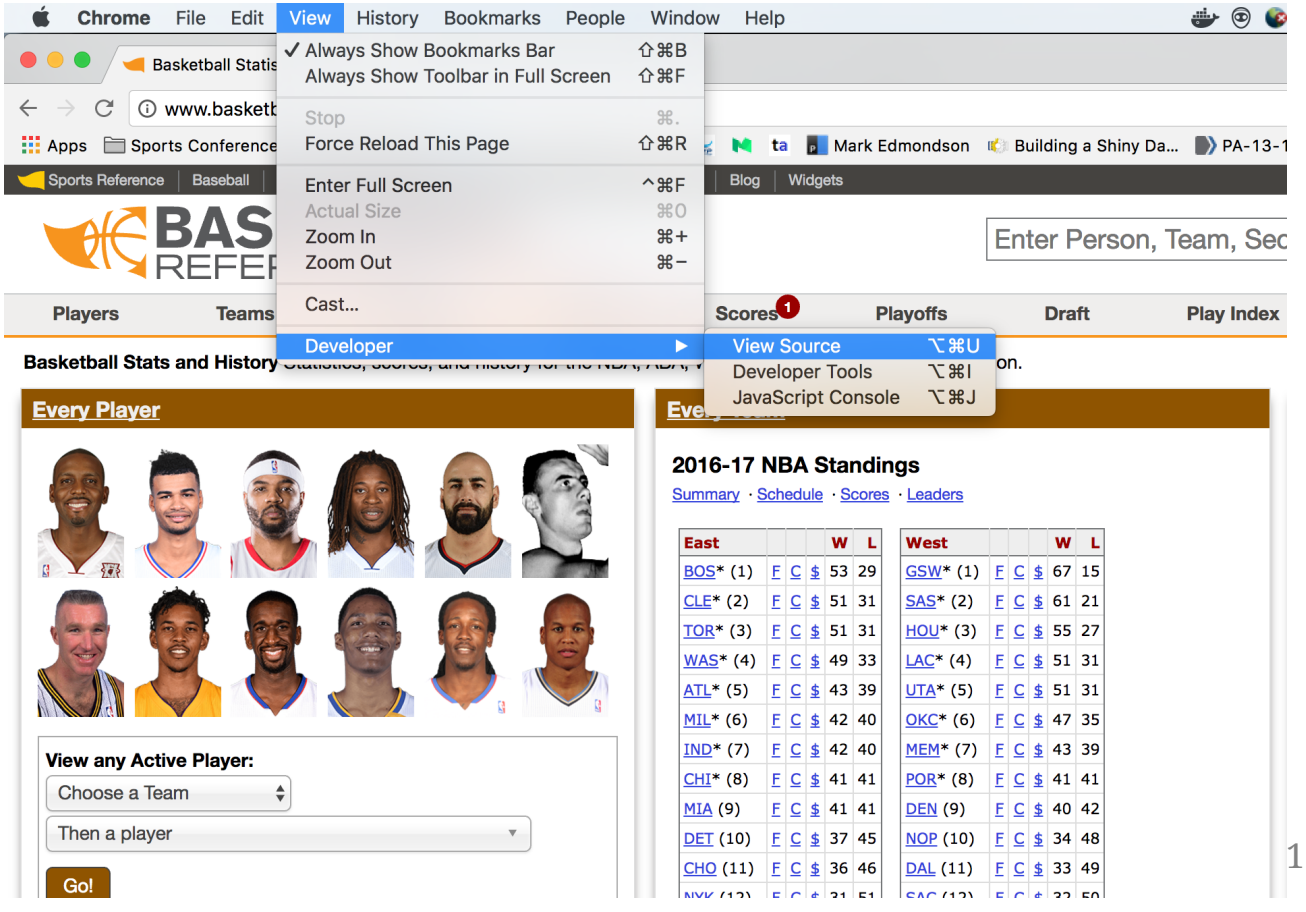
Web Developer Tools

Every modern browser has a suite of "developer tools". These include useful functions for Web scrapers, including:

1. Viewing the source code
2. Inspecting elements

I personally like Chrome's suite of Developer Tools.

Viewing Source in Chrome



Practice: Static or Dynamic?

Look at the source code for each of the following sites and determine whether they are examples of *static* or *dynamic* data.

Case 1. http://tennisabstract.com/reports/atp_elo_ratings.html

Case 2. http://www.espncricinfo.com/ci/content/stats/

Solution: Static or Dynamic?

Case 1 is static data.

Case 2 is dynamic data.

Finding Elements

- Whether you are working with static of dynamic data you need to be able to locate the elements that contain your data
- It is the information about this element which you will need to automate data capture
- CSS `class` and `id` fields are the most common ways to uniquely identify the element containing your data

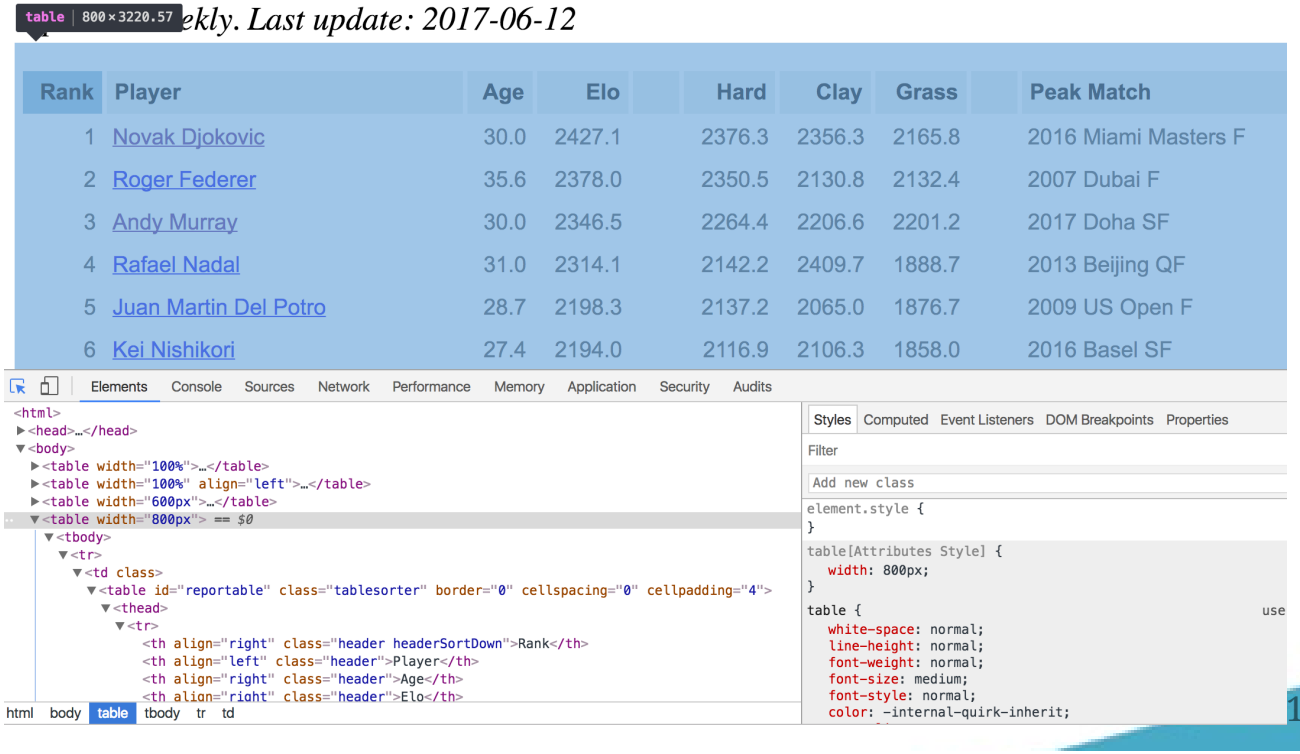
Practice: Inspect Element

Use the following static data example:
http://tennisabstract.com/reports/atp_elo_ratings.html

1. Find the CSS class or id that contains the Elo ratings data

Solution: Inspect Element

Here, I use the "inspect element" settings from Chrome's developer tools to learn about the CSS of the table containing the Elo ratings.



Solution: Inspect Element

- The CSS **class** for the table is "tablesorter"
- The CSS **id** is "reportable"

Scraping Static Data

There are a few options for extracting static HTML data.

1. `readLines` is an option if the data is *not* nicely formatted, in other words, when there is a lack of structure
2. More typically, the data is *nice* (e.g. if it is contained in a HTML table or other predictable tag) and we can use scraping packages like `rvest` or `RCurl` to get the data in a format we can work with.

Using `rvest`

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)
- Works with `magrittr` and allows piping commands with `%>%` operator

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)
- Works with `magrittr` and allows piping commands with `%>%` operator
- Allows some browsing functionality

Using `rvest`

- Suite of tools for scraping static Web data and putting them in easy-to-use objects (like data.frames)
- Works with `magrittr` and allows piping commands with `%>%` operator
- Allows some browsing functionality
- Authored by Hadley Wickham

Example: Scraping Box Scores

In this example, we will use `rvest` to extract the Eastern Division Standings.

First, we import the page content.

```
library('rvest')

# Creating object with the address
url <- 'http://www.basketball-reference.com/boxscores/'

#Reading the code from the site
webpage <- read_html(url)
```

Example: Scraping Box Scores

The `html_nodes` function is the work horse function for extracting specific elements of a site. We can specify the element we want using its CSS tag or using an XPATH selector.

```
# Using the CSS table tag to get all tables
data <- webpage %>%
   html_nodes(css = 'table') %>%
   html_table()

length(data) # List of multiple tables
```

```
## [1] 7
```

Example: Scraping Box Scores

Using an XPATH (XML Path Language) can help to make our extraction more specific, though the syntax is more opaque.

```
# Using an XPATH selector to get the specific table of interest
data <- webpage %>%
   html_nodes(xpath = '//*[@id="divs_standings_E"]') %>%
   html_table(header = T)

head(data[[1]])
```

```
##   Eastern Conference                 W                 L               W/L%
## 1  Atlantic Division Atlantic Division Atlantic Division Atlantic Divisior
## 2    Boston Celtics*                53                29               .646
## 3   Toronto Raptors*                51                31               .622
## 4    New York Knicks                31                51               .378
## 5 Philadelphia 76ers                28                54               .341
## 6      Brooklyn Nets                20                62               .244
##                   GB              PS/G              PA/G
## 1 Atlantic Division Atlantic Division Atlantic Division
## 2                  —             108.0             105.4
## 3                2.0             106.9             102.6
## 4               22.0             104.3             108.0
```

Practice: Static Data Extraction

The following site lists the Elo ratings of professional male tennis players:
Tennis Abstract Elo

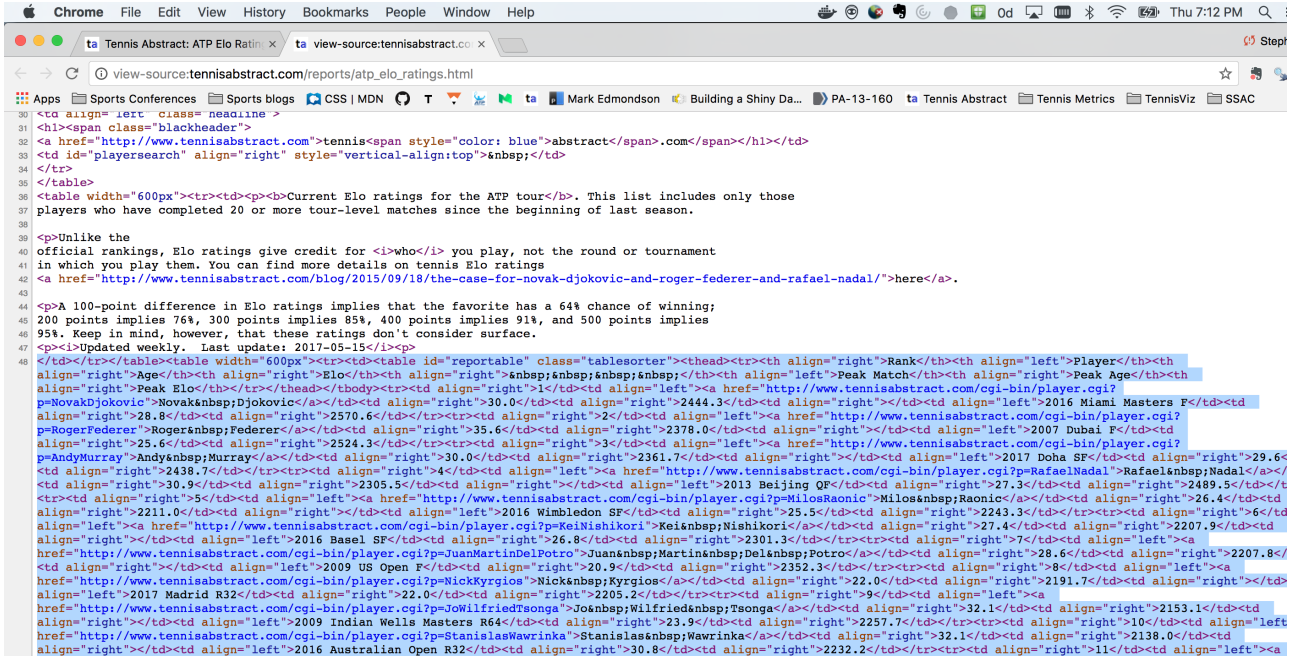
Practice: Static Data Extraction

The following site lists the Elo ratings of professional male tennis players:
Tennis Abstract Elo

1. Use your Web inspection tools to determine if the ratings are static data
2. Use `rvest` to scrape the data as efficiently as you can

[1] For a 'table' with class 'x' you can use 'table.x' as a shortcut

Solution: Elo Rating Extraction



Solution: Elo Rating Extraction

```
url <- "http://tennisabstract.com/reports/atp_elo_ratings.html"

page <- read_html(url)

# Use table class to extract Elo table
elo <- page %>%
    html_nodes("table.tablesorter") %>%
    html_table()

head(elo)
```

```
## [[1]]
##     Rank                     Player  Age    Elo     
## 1      1             Novak Djokovic 30.0 2427.1   NA 2376.3 2356.3 2
## 2      2              Roger Federer 35.6 2378.0   NA 2350.5 2130.8 2
## 3      3                Andy Murray 30.0 2346.5   NA 2264.4 2206.6 2
## 4      4               Rafael Nadal 31.0 2314.1   NA 2142.2 2409.7 1
## 5      5      Juan Martin Del Potro 28.7 2198.3   NA 2137.
## 6      6              Kei Nishikori 27.4 2194.0   NA 2116.9 2106.3 1
## 7      7               Milos Raonic 26.4 2185.0   NA 2125.1 2004.5 1
## 8      8              Dominic Thiem 23.7 2178.0   NA 1892.1 2217.6 1
## 9      9         Stanislas Wawrinka 32.2 2171.5   NA 2078.3 2138.2 1
## 10    10           Alexander Zverev 20.1 2156.2   NA 1959.9 2066.9 1
```

Dynamic Data & Automated Browsing

Dynamic Data & Automated Browsing

- Because dynamic data is created on-the-fly (in response to user interactions) we have to browse to get access to it

Dynamic Data & Automated Browsing

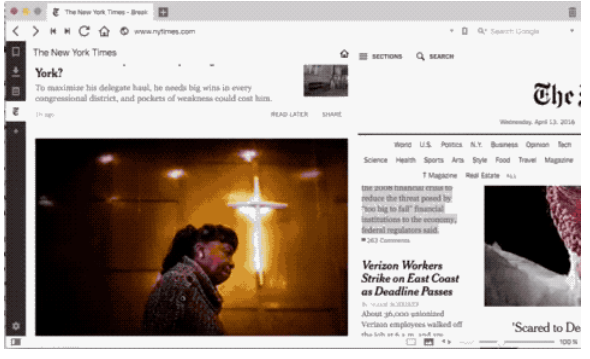
- Because dynamic data is created on-the-fly (in response to user interactions) we have to browse to get access to it
- Fortunately, we can automate browsing

Dynamic Data & Automated Browsing

- Because dynamic data is created on-the-fly (in response to user interactions) we have to browse to get access to it
- Fortunately, we can automate browsing
- We just need to find what instructions to give to mimic the browsing that generates the data and get familiar with tools that can implement these instructions

Scraping Dynamic Data with `RSelenium`

- We have to automate Web browsing to get dynamic data
- *Selenium* is software that allows automated Web browsing
- RSelenium is a package that provides Selenium functionality in R



RSelenium: Basic Steps

1. Set the Web driver (select browser and port)
2. Find the elements with the data
3. Extract the content
4. Parse the contents

Installing RSelenium

There are a few steps you need to get started with RSelenium.

1. Install a Selenium server which is a standalone java program and can be downloaded here:selenium-release.storage.googleapis.com
2. Run the Selenium server with the command: `java -jar selenium-server-standalone-x.xx.x.jar` where the `x.xx.x` will be the specific version (Default port is 4444)
3. Install RSelenium from CRAN

Example: Tennis Match Statistics

Consider the following match summary: 2017 Australian Open Final

Example: Tennis Match Statistics

If we inspect the page, we find that these stats are dynamic data. We also find that the main table of content has the id `detail`.

```
    <body id="top" class="tennis detailbody">

        <div id="detail" class="sport-tennis"><div id="detcon">
         <table class="detail">
             <thead>
             <tr>
                 <th class="header">
                     <div class="fleft">
                         <span class="flag fl_3473162"></span>ATP - SINGLES: <a href="#" onclick="window.open('/tennis/atp-singles/australian-open/');
return false;">Australian Open (Australia), hard - Final</a>

                     </div>

                 </th>
             </tr>
             </thead>
             <tbody>
             <tr>
                 <td class="hclean"></td>
             </tr>
             </tbody>
```

Using RSelenium

Below we activate the driver using a port that is not in use.

Note: You may need to activate javascript in the background for this driver to work.

```
# Running java -Dwebdriver.chrome.driver="chromedriver" -jar seleniu

library(RSelenium) # Load the package

# Match statistics URL
url <- "http://www.flashscore.com/match/Cj6I5iL9/#match-statistics;0"

# Establish remote driver using Chrome
remDr <-remoteDriver(port = 5556, browser = "chrome")
remDr$open(silent = TRUE)
remDr$navigate(url) # Navigate page
```

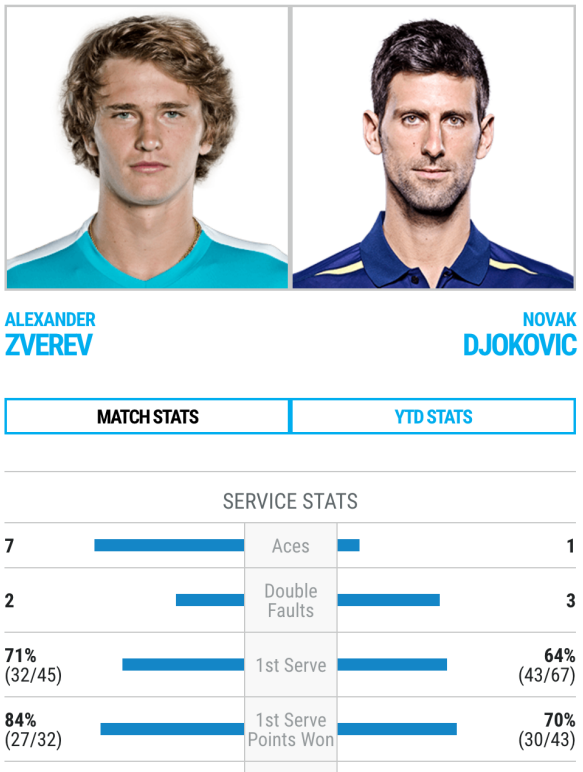
Using RSelenium

Next we extract the table of stats using the CSS `id` node.

```
# Get id element
webElem <- remDr$findElements(using = 'id', "detail")

#  Use getElementText to extract the text from this element
unlist(lapply(webElem, function(x){x$getElementText()}))[[1]]

remDr$close() # Close driver when finished
```

Practice: RSelenium

Take a look at the following match summary that you can find here.



Practice: RSelenium

Use what we've covered about `RSelenium` to extract the statistics for this match.

1. Start by inspecting the Web site
2. Determine which CSS element is most likely to contain the stats
3. Create a remote driver, navigate to that element, and check if the text for the match statistics are contained in the element

Solution: RSelenium

Inspection of the source code suggests that the Element with id *modalScoresMatchStatsTable* is likely to contain the statistics.

```
<div id="modalScoresContentContainer" class="modal-scores-tab-container">
    <div id="modalScoresMatchStats" class="modal-scores-match-stats">

        <div id="modalScoresMatchStatsTable" class="modal-scores-match-stats-table">
            <div class="modal-scores-match-stats-players">
                    <div class="match-stats-player-left">
        <div class="player-left-image">
            <a href="/en/players/alexander-zverev/z355/overview">
                <img src="/-/media/tennis/players/head-shot/2017/02/12/01/20/zverev_a-headshot-ao17.png" />
            </a>
        </div>
        <div class="player-left-name">
            <a href="/en/players/alexander-zverev/z355/overview">
                <span class="first-name">
                    Alexander
                </span>
                <span class="last-name">
                    Zverev
                </span>
            </a>
        </div>
    </div>
```

Solution: RSelenium

Now we navigate to the site.

```
# Match statistics URL
url <- "http://www.atpworldtour.com/en/players/novak-djokovic/D643/ov

# Establish remote driver using Chrome
remDr <- remoteDriver(port = 5556, browser = "chrome")
remDr$open(silent = TRUE)
remDr$navigate(url) # Navigate page
```

Solution: RSelenium

Then we find the `id` element of interest and extract the text it contains.

```
# Get id element
webElem <- remDr$findElements(using = "id",
    "modalScoresMatchStatsTable")

# Use getElementText to extract
# the text from this element
unlist(lapply(webElem, function(x) {
    x$getElementText()
}))[[1]]
```

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

Summary

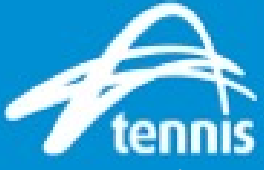
- Web data can be classed into three main categories: directly importable, static, or dynamic
- We can use source inspection and CSS selector tools to determine which data type we are working with and the site elements that contain the data
- We have seen how we use tools like `rvest` to capture static Web data
- For dynamic data, we can use automated browsing with `RSelenium`

Resources

- CSS and HTML crash course
- XPATH
- rvest
- RSelenium

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

```
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

```
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

```
library(devtools)

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

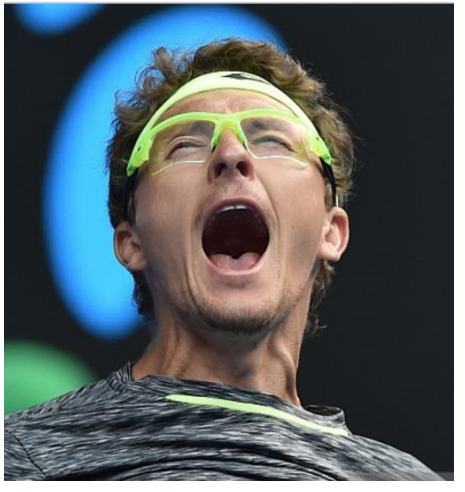
Tools of `tidyr`

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?

```
# 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?

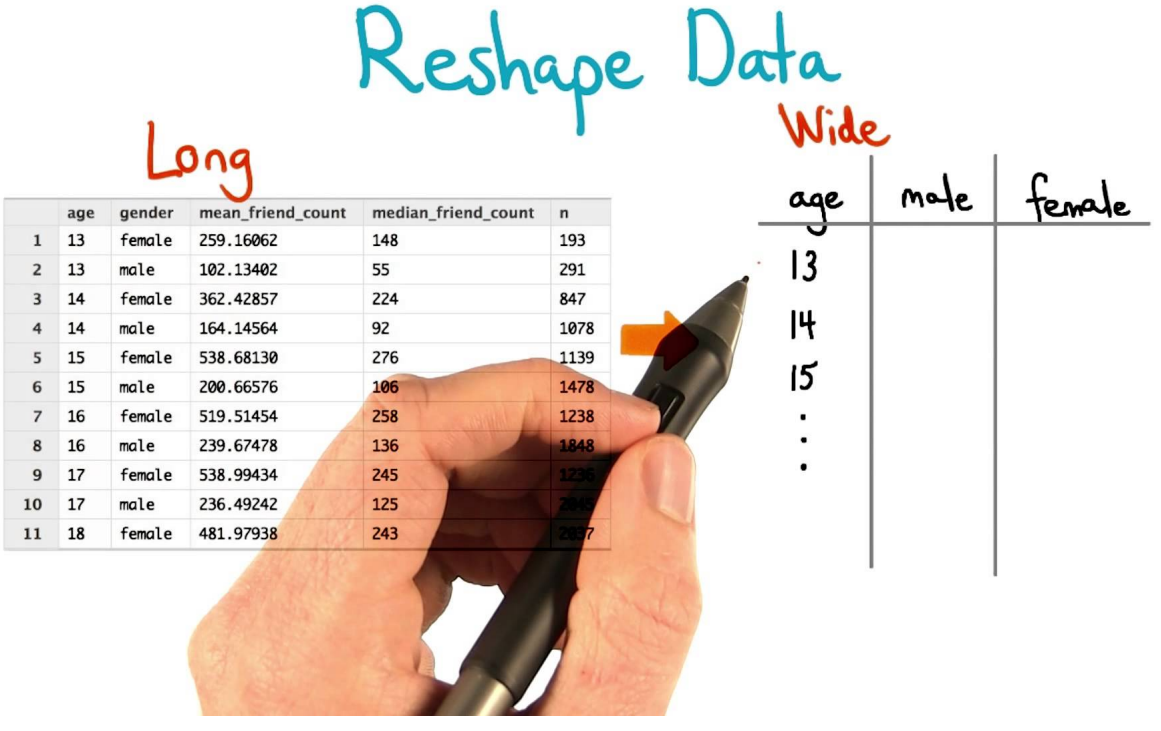
- Suppose we had wanted a score based on wins and losses, what would we need to do this?

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	2045
11	18	female	481.97938	243	2827

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)
```

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")])
```

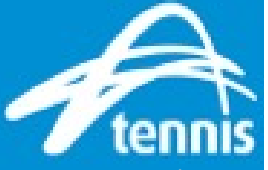
```
##         Date Outcome       Player
## 1 2015-01-19  Winner  Berankis R.
## 2 2015-01-19  Winner  Dimitrov G.
## 3 2015-01-19  Winner  Anderson K.
## 4 2015-01-19  Winner    Chardy J.
## 5 2015-01-19  Winner     Lacko L.
## 6 2015-01-19  Winner Matosevic M.
```

Resources

- dplyr
- tidyr
- lubridate
- agrep
- regex

Exploring

Tools for Exploratory Data Analysis



Data Exploration

Exploratory data analysis is detective work-numerical detective work-or counting detective work-or graphical detective work...Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone--as the first step. - John Tukey

EDA Starts With Specific Questions

- All exploration needs some direction
- Exploring aimlessly wastes time and rarely will get you where you need to be
- To avoid aimless exploration, you need to ask yourself what you are looking for? and what would be interesting to find?

Our Questions

In this tutorial we will use EDA to investigate some conventional wisdom in tennis. Here are 3 ideas commentators repeatedly say:

1. Second serve is more important than the first serve
2. Players who serve first have an advantage
3. The 7th game is the most important in a set

Match Data

To examine the first question we will use the `atp_matches` data set from the `deuce` package.

Load the data, limit to the years 2005 to 2015.

```
library(deuce)

data(atp_matches)

atp_matches <- atp_matches %>%
  dplyr::filter(year >= 2005 & year <= 2015)
```

Variable Documentation

Use the `help` function to learn about the contents of the `atp_matches`. What do these data include?

```
help("atp_matches", package = "deuce")
```

Second vs First Serve

The first question we will consider is the importance of the second serve versus the first serve.

- One implication of this statement is that we might expect the winner of a match to have outperformed on second serve compared to first serve
- This suggests focusing on the difference in winner and loser service stats

Service Points Won

For each match, we will calculate the proportion of second serve points won and first points won for the winner and loser of the match.

```
atp_matches <- atp_matches %>%
    dplyr::mutate(
      w_first = w_1stWon / w_svpt,
      w_second = w_2ndWon / w_svpt,
      l_first = l_1stWon / l_svpt,
      l_second = l_2ndWon / l_svpt
    )
```

Summarise First and Second Differential

Let's look at the difference in the winner and loser stats on each serve.

```
summary(with(atp_matches, w_first - l_first))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   -0.50    0.02    0.08    0.09    0.14    0.88  224474
```

Filtering

One problem with the previous summary is that we have included some matches with NA stats and 0-values stats, as well as matches ending in retirement. How would you correct this?

Filtering

One problem with the previous summary is that we have included some matches with NA stats and 0-values stats, as well as matches ending in retirement. How would you correct this?

```
summary(with(
  subset(atp_matches, !is.na(w_svpt) & w_svpt != 0
         & !is.na(l_svpt) & l_svpt != 0 & !Retirement),
             w_first - l_first))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.35260  0.01658  0.07684  0.08371  0.14330  0.61300
```

```
summary(with(
  subset(atp_matches, !is.na(w_svpt) & w_svpt != 0
         & !is.na(l_svpt) & l_svpt != 0 & !Retirement),
             w_second - l_second))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.36840 -0.01673  0.03355  0.03709  0.08689  0.51210
```

Charting

Charting

- Most of our describing and exploration in `R` happens with graphics

Charting

- Most of our describing and exploration in `R` happens with graphics
- A powerful package for graphing is `ggplot2`

Charting

- Most of our describing and exploration in `R` happens with graphics
- A powerful package for graphing is `ggplot2`
- `ggplot2` provides a grammar for building univariate, bivariate, and lattice plots

Charting

- Most of our describing and exploration in `R` happens with graphics
- A powerful package for graphing is `ggplot2`
- `ggplot2` provides a grammar for building univariate, bivariate, and lattice plots
- Also, many specialty graphics packages build on `ggplot2`

Overview of `ggplot2`

To install, use `install.packages('ggplot2')`.

Function.Type	Description
Function.Type	Description
aes	Relates variables to axes and aesthetic elements of our plot
geoms	Define how the variables will be displayed, that is, the type of chart
facet	Splits plot into rows and columns defined by a group variable
scales	Customizes the range and limits of aesthetics
themes	Further controls of the style and elements of the chart

Charting Serve Differentials

We could describe the difference in service stats with one of several univariate geoms: `geom_histogram`, `geom_density`, `geom_boxplot`.

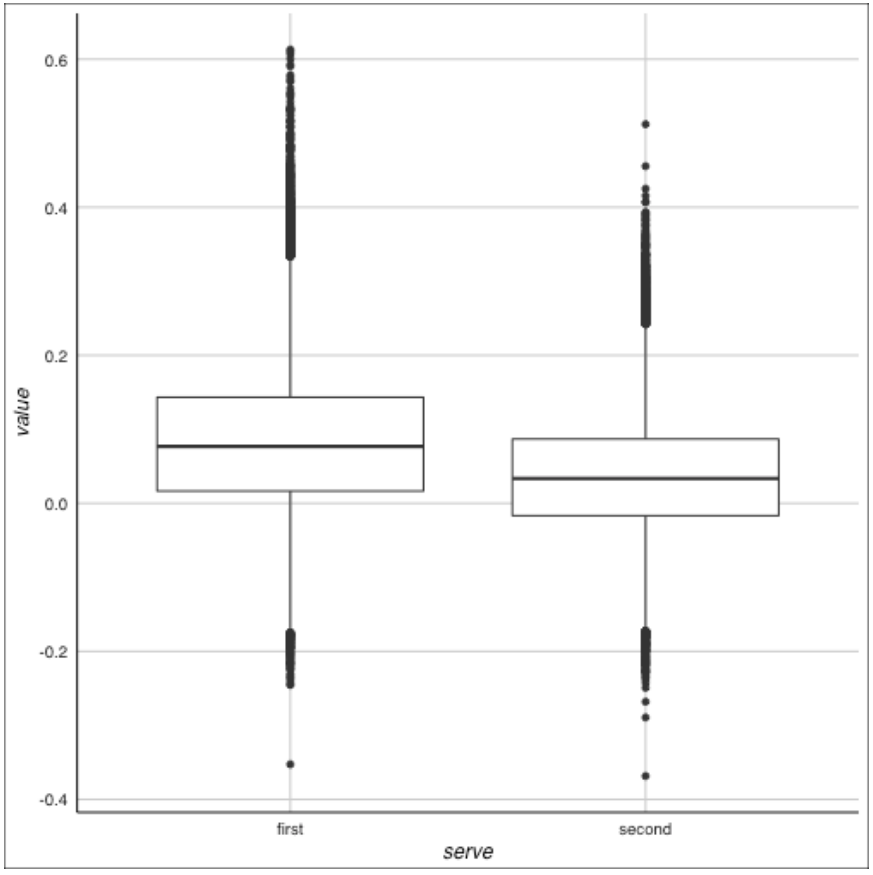
1. Reshape the data to long format with first and second serve differences stacked length-wise
2. Then use `geom_boxplot` to look at the difference in the differentials for the first and second.
3. What do you conclude about the comparative importance of first and second serve?

Solution: Charting Serve Differentials

```
# Prepare long format
serve_stats <- atp_matches %>%
  filter(!is.na(w_svpt), w_svpt != 0,
         !is.na(l_svpt), l_svpt != 0, !Retirement) %>%
  dplyr::mutate(
    first = w_first - l_first,
    second = w_second - l_second
  ) %>%
  gather("serve", "value", first, second)
```

Solution: Charting Serve Differentials

```
library(ggplot2)
library(ggthemes) # Extra themes

serve_stats %>% # We can use pipes with ggplot2!
  ggplot(aes(y = value, x = serve)) +
  geom_boxplot() +
  theme_gdocs()
```


0.6
0.4
0.2
0.0
-0.2
-0.4
value
first
second
serve

Interpretation

- We see a greater gap in the stats on the first serve compared to the second
- This suggests that it is a stronger differentiator between winning and losing

Serving First

- Next, let's consider the importance of serving first and what advantage that has for winning matches
- For this question, we will make use of the `gs_point_by_point` data which includes point-level data for several years of Grand Slam matches

```
data("gs_point_by_point")

gs_point_by_point <- gs_point_by_point %>%
  filter(Tour == "atp")
```

Practice: Exploring Serve Advantage

1. Sort the data by match and point number
2. Determine the percentage of players who served first that won the match
3. Determine the percentage who served second and won the match
4. Chart your results using a bar chart

Solution: Exploring Serve Advantage

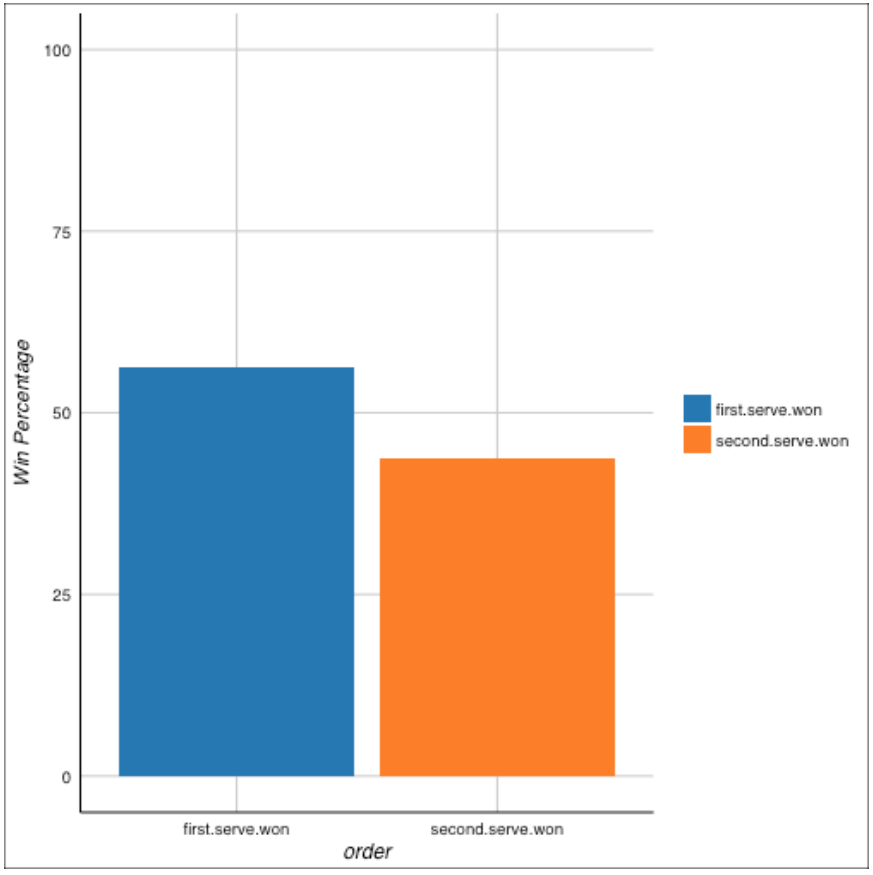
Here we sort and calculate the outcome with respect to the first and second server

```
# Sort data
gs_point_by_point <-
  gs_point_by_point[order(gs_point_by_point$match_id, gs_point_by_po

serve_advantage <- gs_point_by_point %>%
  group_by(match_id) %>%
  dplyr::summarise(
    first.serve.won = PointServer[1] == PointWinner[n()],
    second.serve.won = ifelse(PointServer[1] == 1, 2, 1) == PointWinn
  )
```

Solution: Exploring Serve Advantage

Now we summarise and reshape the data to prepate for charting.

```
serve_advantage <- serve_advantage %>%
  dplyr::summarise(
    first.serve.won = mean(first.serve.won),
    second.serve.won = mean(second.serve.won)
  ) %>%
  gather("order", "win", first.serve.won, second.serve.won)

serve_advantage %>%
  ggplot(aes(y = win * 100, x = order, fill = order)) +
  geom_bar(stat = "identity") +
  scale_y_continuous("Win Percentage", lim = c(0, 100)) +
  scale_fill_tableau(name = "") +
  theme_gdocs()
```



We find a near 10 percentage point advantage with serving first!

Practice: Game 7

Using the same point-level dataset and similar methods, investigating whether the "all-important game 7" is really that important for winning a set.

1. Determine how often the winner of the 7th game of a set won the set
2. Limit your analysis to "close" sets, which we will define as sets with 10 more games
3. Plot the percentage difference in set wins for winner's and loser's of game 7

Solution: Game 7

First we prepare the variables for summarising the set wins.

```
# Determine max games
game7 <- gs_point_by_point %>%
  group_by(match_id, SetNo) %>%
  dplyr::mutate(
    MaxGame = max(GameNo),
    SetWinner = PointWinner[n()]
)

# Filter and determine game 7 winner
game7 <- game7 %>%
  filter(MaxGame >= 10) %>%
  group_by(match_id, SetNo) %>%
  dplyr::mutate(
    Game7 = PointWinner[max(which(GameNo == 7))]
)
```

Solution: Game 7

Now, we summarise the set win percentages by game 7 status.

```
game7 <- game7 %>%
  group_by(match_id, SetNo) %>%
  dplyr::summarise(
    game7.winner = Game7[1] == SetWinner[1],
    game7.loser = ifelse(Game7[1] == 1, 2, 1) == SetWinner[1]
)

# Summarise and put in long format
game7 <- as.data.frame(game7) %>%
  dplyr::summarise(
    game7.winner = mean(game7.winner),
    game7.loser = mean(game7.loser)
  ) %>%
  gather("gamewinner", "setwin", game7.winner, game7.loser)
```

Solution: Game 7

Now we are ready to chart our findings.

```
game7 %>%
  ggplot(aes(y = setwin * 100, x = gamewinner, fill = gamewinner)) +
  geom_bar(stat = "identity") +
  scale_y_continuous("Set Win Percentage", lim = c(0, 100)) +
  scale_fill_solarized(name = "") +
  theme_gdocs()
```

Summary

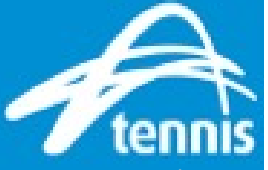
- Using common tools for data manipulation and graphics we have investigated 3 commonly held views in tennis
- Our observations provide support for only 1 of these 3 beliefs: the advantage of serving first

Resources

- ggplot2
- ggthemes
- Tukey and EDA
- EDA by Roger Peng

Modelling

Predictive Data Analysis in R



Sport Prediction

Predicting *who will win* is the holy grail of sports analytics and is a major research area of interest for sports statisticians.

Descriptive vs Predictive Models

Descriptive vs Predictive Models

- *Predictive modeling* usually refers to machine learning

Descriptive vs Predictive Models

- *Predictive modeling* usually refers to machine learning
- The goal of these techniques is to improve predictive performance

Descriptive vs Predictive Models

- *Predictive modeling* usually refers to machine learning
- The goal of these techniques is to improve predictive performance
- This is a very different goal from statistical models, like regression, where inference and interpretation are of primary importance

Descriptive vs Predictive Models

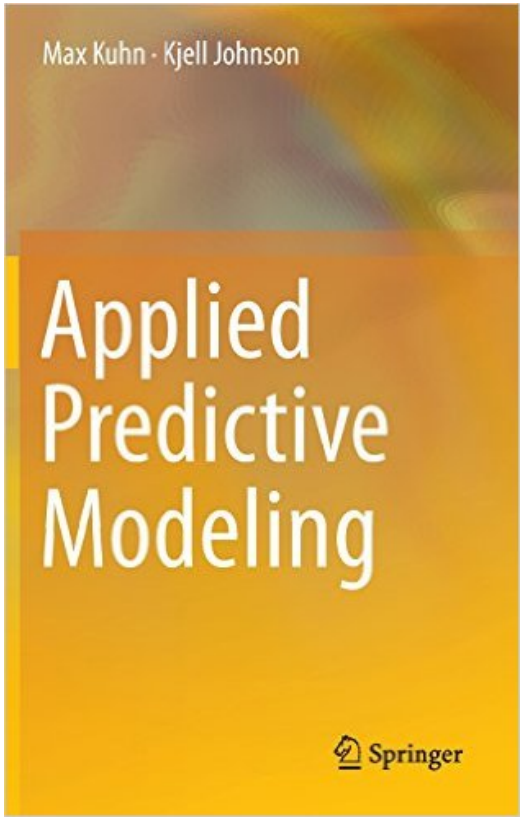
- *Predictive modeling* usually refers to machine learning
- The goal of these techniques is to improve predictive performance
- This is a very different goal from statistical models, like regression, where inference and interpretation are of primary importance
- Predictive models will often sacrifice interpretability for improved performance

Describe Before You Predict

- Because it is often a challenge to interpret the "how" of machine learning methods, it is good practice to first do some statistical modelling (e.g. `glm(y ~ x + ...)`)
- The reasons for this include:
    - Getting familiar with the interrelationships in your data
    - Identifying any issues not flagged during EDA
    - Developing some expectations for the predictive modelling results

Predictive Modelling with `caret`

- Once you are ready to develop your prediction model, the `caret` package, written by Max Kuhn, is a great resource for machine learning in `R`
- `caret` stand for Classification And REgression Training
- It includes 100+ predictive modelling methods
- It provides a unified & streamlines interface for building and evaluating models
- It allows parallel processing for faster computation
- You can install with `install.packages('caret')`



Modelling Steps

Modelling Steps

Step 1. Decide how to spend your data

Modelling Steps

Step 1. Decide how to spend your data

Step 2. Split the data into training and test

Modelling Steps

Step 1. Decide how to spend your data

Step 2. Split the data into training and test

Step 3. Conduct pre-processing (as needed)

Modelling Steps

Step 1. Decide how to spend your data

Step 2. Split the data into training and test

Step 3. Conduct pre-processing (as needed)

Step 4. Train the model with resampling

Modelling Steps

Step 1. Decide how to spend your data

Step 2. Split the data into training and test

Step 3. Conduct pre-processing (as needed)

Step 4. Train the model with resampling

Step 5. Evaluate the model with test data

Why Split Up the Data at All?

- Testing our data on independent samples is the strongest form of validation and evaluation
- If we trained and tested on the same data, we risk *overfitting* which is the machine-learning equivalent to a "Monday morning quarterback"
- We also resample among training for additional protection against overfitting

Using `createDataPartition`

We can use `createDataPartition` to split our data:

```
createDataPartition(y, times, p, list, ...)
```

Argument	Description
y	Outcome vector to balance sampling on
times	Number of partitions
p	Proportion of each partition allocated to training
list	Logical whether list is returned

Note: Using 70% of our data for training is typical

Example: Partitioning Data

In this example, we create one partition with 70% of our dataset allocated to training.

```
library(caret) # Load caret

# Returns matrix of indices for obs in training
train <- createDataPartition(
  y = data$outcome,
  times = 1,
  p = 0.7,
  list = F
)
```

Practice: Using CreateDataPartition

Modify the previous code to obtain 5 partitions for your training samples with 60% of observeations in each devoted to training.

Practice: Using CreateDataPartition

Modify the previous code to obtain 5 partitions for your training samples with 60% of observeations in each devoted to training.

Answer:

```
train <- createDataPartition(
  y = data$outcome,
  times = 5,
  p = 0.6,
  list = F
)
```

Pre-Processing

Before we split our data, we need to pre-process our data. The pre-processing can protect against some loss in model accuracy due to scale, skew, or high correlation.

Common pre-processing steps are:

1. Centering - Give all variables a common mean of 0
2. Standardizing - Give all variables a common scale
3. Remove highly correlated variables
4. Reduce dimension (when $n \sim p$)

The `train` Function

The main workhouse function for model training in `caret` is `train`. Here are the main arguments you need to know to get started.

Argument	Description
form	Formula (y ~ x)
data	Data frame of training data
method	Character of the ML method to be used
metric	Performance metric for summarizing
tuneLength	Sets granularity for tuning parameter if tuneGrid not specified
trControl	Control parameters for training
tuneGrid	Data frame that gives explicit range for tuning parameters

Practice: Pre-Processing

What function would you use to standardize your model features?

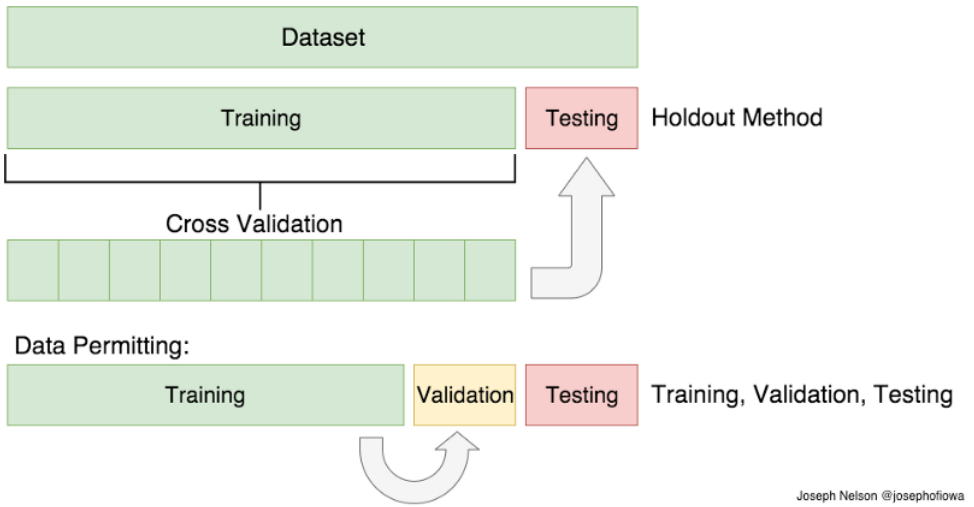
Practice: Pre-Processing

What function would you use to standardize your model features?

Answer: The `scale` function

Training Control

- The `trControl` argument is a list that controls a number of aspects of training including resampling and how we summarise performance with each resample.
- The resampling is an important additional measure to protect against overfitting when training



Example: Using `trControl`

In our example, we will use `trControl` to use 5-fold cross validation and a two-class summary for our performance measures.

```
ctrlSpecs <- trainControl(
    method = "repeatedCV",
    repeats = 5,
    summaryFunction = twoClassSummary, # function from caret
    classProbs = TRUE # Needed to use twoClassSummary
)
```

Grid Tuning

- The `tuneGrid` is a way to give a specific grid for the tuning parameters of the method
- You can use `expand.grid` to make a range of parameters
- To determine the parameters to tune and their variable names you can use `getModelInfo`

Using `getModelInfo`

- We can get a model of interest with `getModelInfo('model')` or get info from all models with `getModelInfo()`.
- This returns a list per model with information about the model type, parameters, grid function, etc.
- Here is an example with the `rpart` model.

```
getModelInfo('rpart')[['rpart']][1:4]
```

```
## $label
## [1] "CART"
##
## $library
## [1] "rpart"
##
## $type
## [1] "Regression"     "Classification"
##
## $parameters
##   parameter   class                 label
## 1        cp numeric Complexity Parameter
```

Performance Metrics

- We can use the `metric` argument to choose our performance metric for training evaluation
- There are many metrics for evaluating classification. In general, it is best to choose one when choosing among model approaches

Metric	Description
Accuracy	Proportion of exactly correct classifications
AUC	Area under the ROC curve
Sensitivity	The true positive rate (also called 'recall')
Specificity	The true negative rate
LogLoss	Prediction-weighted loss function

Setting Performance Metric

There is no one correct performance measure. In fact, multiple should be evaluated when testing. For training, the "log loss" is good all around measure. Here is how we can set our control specs to use it.

```
ctrlSpecs <- trainControl(
    method = "repeatedCV",
    repeats = 5,
    summaryFunction = mnLogLoss,
    classProbs = TRUE
)
```

Practice: Alternative Resampling

Look at the `method` options using the documentation on `trainControl`. Find an alternative resampling approach and modify the `ctrlSpecs` object to use this method.

Practice: Alternative Resampling

Look at the `method` options using the documentation on `trainControl`. Find an alternative resampling approach and modify the `ctrlSpecs` object to use this method.

In this alternative we use 10 bootstrap resamples:

```
ctrlSpecs <- trainControl(
    method = "boot",
    repeats = 10,
    summaryFunction = mnLogLoss,
    classProbs = TRUE
)
```

Models

There are more than 100 models to try in `caret`! Below is just a sample of some popular kinds. The full list is available with the online caret book.

Category	Description	Examples
Forest	Ensemble of multiple decision trees with bagging (boostrap aggregation)	rf, rfRules, cforest
Boosted	Incremental building of multiple classifiers, which is a kind of correlated ensembling	gbm, adaboost, C5.0
Support Vector Machines	Collection of regression lines that try to maximally separate classes	svmLinear, svmRadial

Random Forest

Let's have a look at each category and how we could train each in `caret`. Below we use the `rf` method to fit a random forest. The `tuneLength` is set to 10 to have a randomly generated grid for the forest parameters.

```
rfFit <- train(
    outcome ~ .,
    data = trainingData,
    method = "rf",
    tuneLength = 10,
    trControl = ctrlSpecs,
    metric = "logLoss"
)
```

Practice: Grid Tuning

Use `getModelInfo` to find the tuning parameters for the `rf` method. Create a data frame with 10 different options and modify the `train` function to use these in place of the random tuning with `tuneLength`.

Practice: Grid Tuning

Use `getModelInfo` to find the tuning parameters for the `rf` method. Create a data frame with 10 different options and modify the `train` function to use these in place of the random tuning with `tuneLength`.

```
# Find tuning parameters
getModelInfo("rf")[["rf"]]$parameters
```

```
##   parameter   class                         label
## 1      mtry numeric #Randomly Selected Predictors
```

```
grid <- data.frame(.mtry = seq(5, 25, length = 10))

# Modify train function
rfFit <- train(
    outcome ~ .,
    data = trainingData,
    method = "rf",
    tuneGrid = grid,
    trControl = ctrlSpecs,
    metric = "logLoss"
)
```

Evaluate the Model

Evaluate the Model

- We can see preformance metrics across the different tuning parameters using: `print` or `plot`

Evaluate the Model

- We can see preformance metrics across the different tuning parameters using: `print` or `plot`
- Also, the selected model can be extracted with `finalModel` and it will have all the class properties of the source method

Evaluate the Model

- We can see preformance metrics across the different tuning parameters using: `print` or `plot`
- Also, the selected model can be extracted with `finalModel` and it will have all the class properties of the source method

For example, with a random forest, we can look at feature importance:

```
library(randomForest) # Class methods for RF

importance(rfFit$finalModel) # Variable importance
```

Resources

- Web book on caret
- Applied Predictive Modeling
- Introduction to Statistical Learning

Blogging

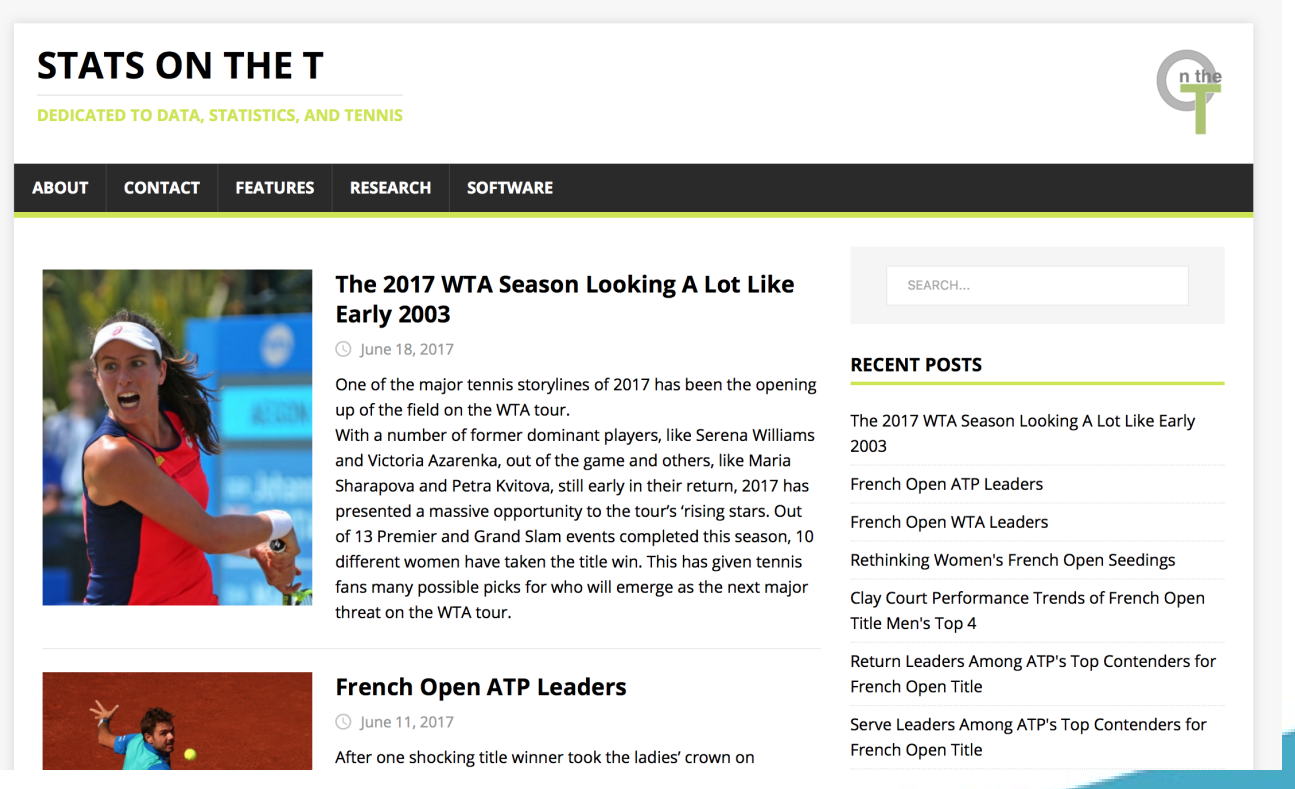
Sharing Your Work with the World

Why Blog?

Sharing your ideas with the world is one of the most joyful parts of sports analysis.

'On The T' Powered By Blogdown

on-the-t.com



Getting There

- Starting a blog might seem overwhelming but, if you know R and some basic CSS, it's easy
- Producing interesting content is the tough part about blogging
- R helps to let you focus on *content* by providing easy-to-use tools that solve the technical details of going from code to the Web
- *If you can write R markdown, you can create a blog!*

Blogdown

Blogdown

- Package for writing a blog in R

Blogdown

- Package for writing a blog in R
- Create content with R markdown

Blogdown

- Package for writing a blog in R
- Create content with R markdown
- Generate site with Hugo, a static site generator

Blogdown

- Package for writing a blog in R
- Create content with R markdown
- Generate site with Hugo, a static site generator
- Deploy on Github, Netlify, or other hosting services

Blogdown

- Package for writing a blog in R
- Create content with R markdown
- Generate site with Hugo, a static site generator
- Deploy on Github, Netlify, or other hosting services
- Authored by Yihui Xie (who is at the conference!)

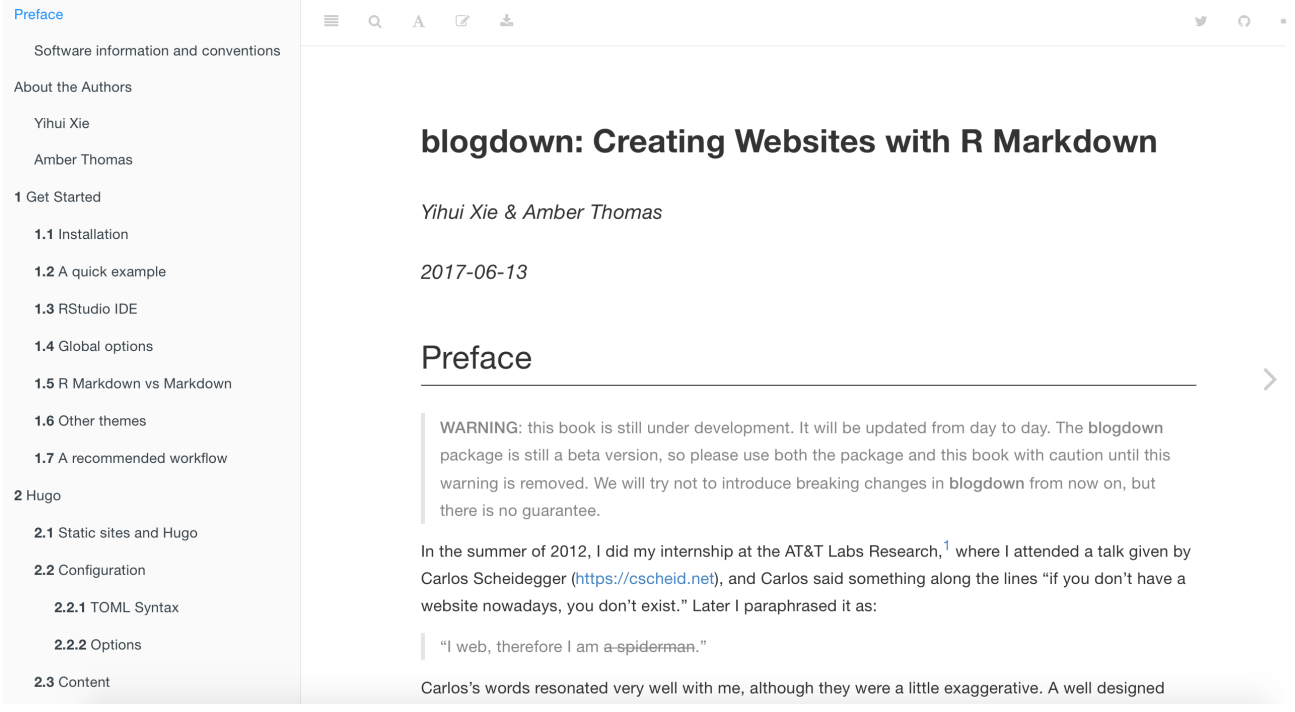
Getting Started

Here are the major functions you will use with `blogdown`.

Function	Description
install_hugo	Downloads and installs Hugo
install_theme	Downloads a Hugo theme from github
build_site	Compiles Rmd files and builds the site
html_page	Renders an Rmd file as an HTML that can be read by Hugo
hugo_cmd	Run Hugo commands
new_content	Creates new file in working directory
new_site	Creates enviroment for new site
serve_site	Preview working version of your site

Learning the Workflow

Yihui and Amber Thomas have a draft manual for using blogdown that will get you started.

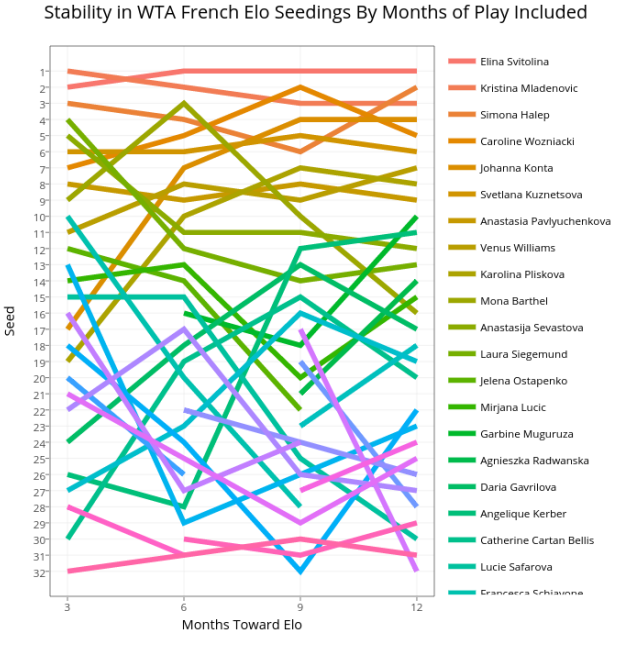


Worklow Overview

1. Install `blogdown` and run `install_hugo`
2. Use `new_site` to create your site directory
3. Setup your directory with `git` and push to `github`
4. Choose your own theme choosing from themes.gohugo.io and use `install_theme`
5. Configure your site by modifying `config` file
6. Create new pages with `new_post`
7. *Deploy!*

Adding Interactive Graphics

- All sports blogs should use charts! And you can make your blog charts especially interesting by making them interactive.
- For `R` users who already masters of `ggplot2`, the `plotly` package is a flexible and fast way to create interactive Web graphics



Plotly

Plotly is a free online service for creating and sharing Web graphics. With the `plotly` package (by Carson Sievert) you can transform `ggplots` to interactive charts on plotly.

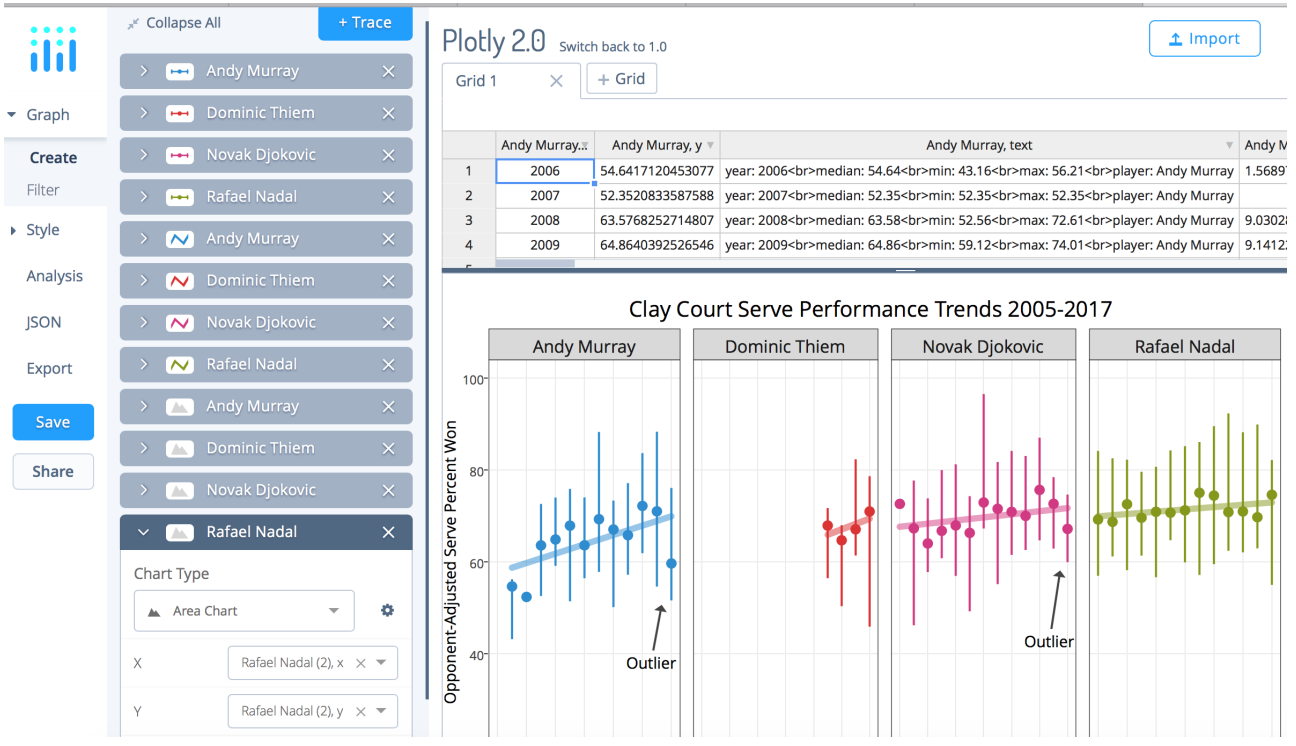
Getting Started with Plotly

Here are the basic steps to follow:

```
install.packages(plotly)

library(plotly)

# After creating a plotly account
Sys.setenv("plotly_username"="your_plotly_username")
Sys.setenv("plotly_api_key"="your_api_key")

<CREATE GGPLOT>

plotly_POST(gg1, filename = "my-new-plot") # Publish
```

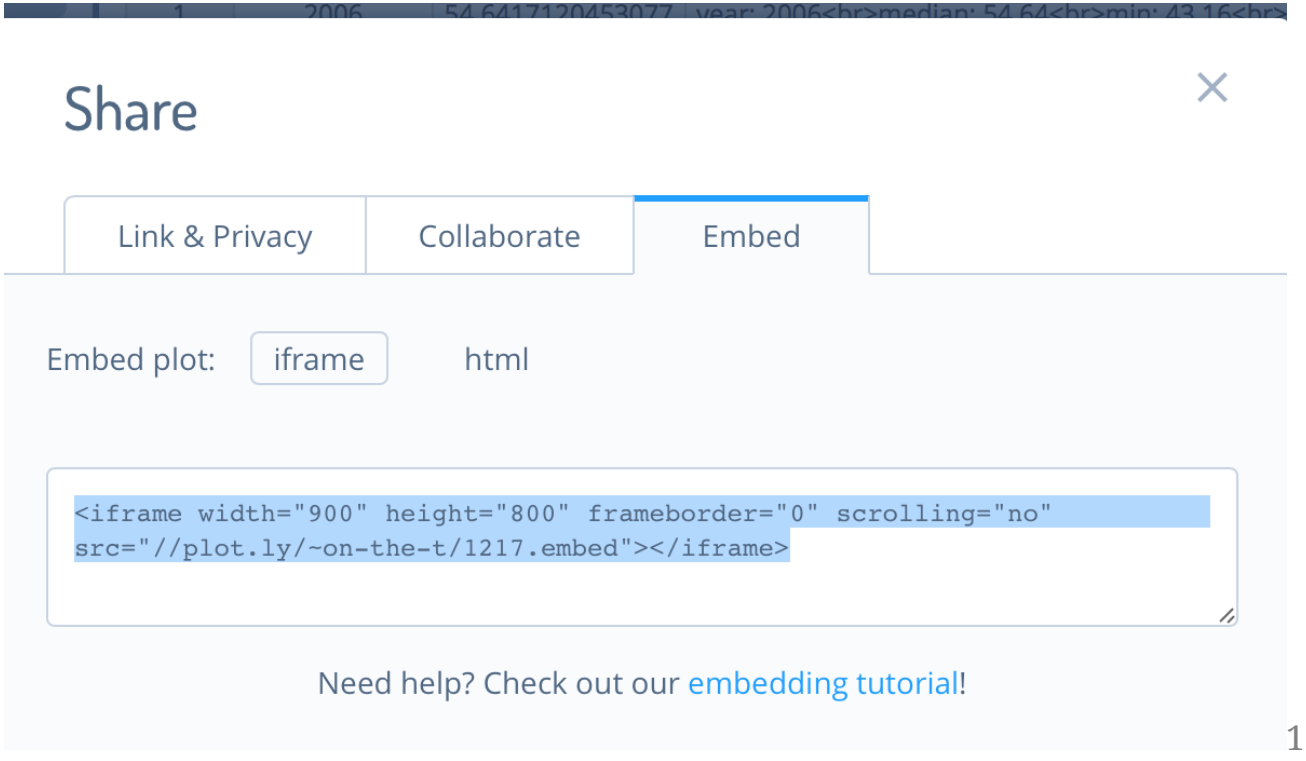
Plotly Site

Further customization can be done on the plotly site.



Plotly Embed

Under the `Share` link you can also find the code that you can use to embed your final plot in your R Markdown file.



Summary

- Doing sports analytics requires a lot of skills
- Many tools in `R` can make capturing, cleaning, and analyzing data much easier
- I hope much of what you have seen in this presentation will help you make your ideas about sports data into a reality

Resources

- blogdown
- plotly
- Git Commands

skovalchik@tennis.com.au

@StatsOnTheT