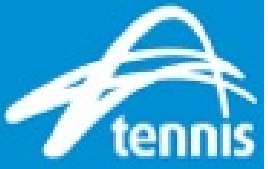
Sports Analytics with R

Melbourne R Ladies

June 15, 2017



About Me

- ISEAL Research Fellow at Victoria University
- Principal Data Scientist for Tennis Australia's Game Insight Group
- Tennis Blog: on-the-t.com
- @StatsOnTheT

What It's Like to Be a Sports Statistician

- Finding sports data
- Scraping sports data
- Data wrangling
- Exploratory data analysis
- Predictive modelling
- Blogging

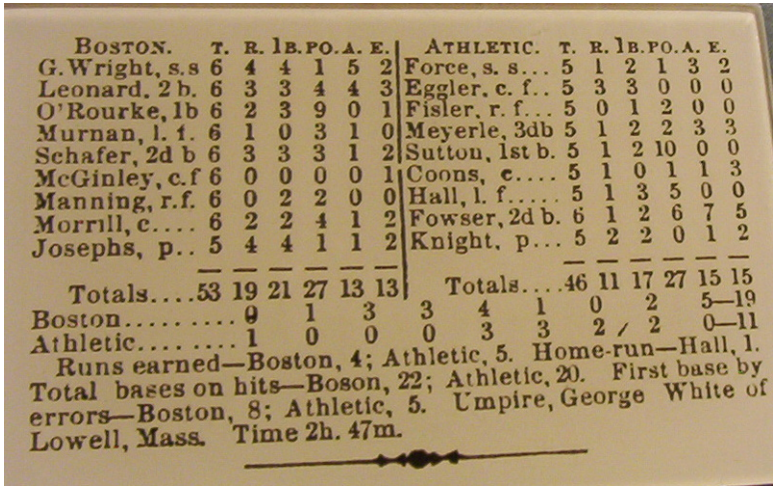
Finding Sports Data

Popular Types of Sports Data

- Box Scores
- Performance Metrics
- Tracking Data
- Wearable Data

Box Scores

A 'box score' is a generic term for the summary statistics reported about a single sporting event.



BOSTON.	T.	R.	1B.	PO.	A.	E.
G. Wright, s. s	6	4	4	1	5	2
Leonard, 2 b.	6	3	3	4	4	3
O'Rourke, 1b	6	2	3	9	0	1
Murnan, l. f.	6	1	0	3	1	0
Schafer, 2d b	6	3	3	3	1	2
McGinley, c. f	6	0	0	0	0	1
Manning, r. f.	6	0	2	2	0	0
Morrill, c.	6	2	2	4	1	2
Josephs, p.	5	4	4	1	1	2
Totals.	53	19	21	27	13	13

ATHLETIC.	T.	R.	1B.	PO.	A.	E.
Force, s. s.	5	1	2	1	3	2
Eggler, c. f.	5	3	3	0	0	0
Fisler, r. f.	5	0	1	2	0	0
Meyerle, 3db	5	1	2	2	3	3
Sutton, 1st b.	5	1	2	10	0	0
Coons, c.	5	1	0	1	1	3
Hall, l. f.	5	1	3	5	0	0
Fowser, 2d b.	6	1	2	6	7	5
Knight, p.	5	2	2	0	1	2
Totals.	46	11	17	27	15	15

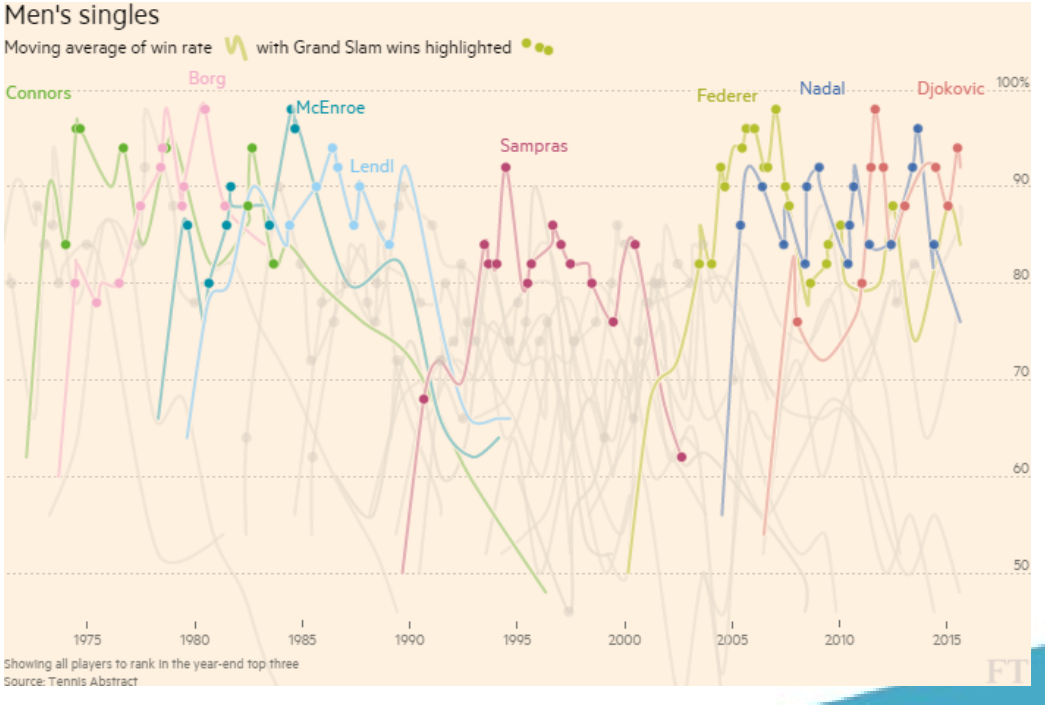
Boston	0	1	3	3	4	1	0	2	5—19
Athletic	1	0	0	0	3	3	2	2	0—11

Runs earned—Boston, 4; Athletic, 5. Home-run—Hall, 1. Total bases on hits—Boson, 22; Athletic, 20. First base by errors—Boston, 8; Athletic, 5. Umpire, George White of Lowell, Mass. Time 2h. 47m.

[1] Baseball box score from 1876.

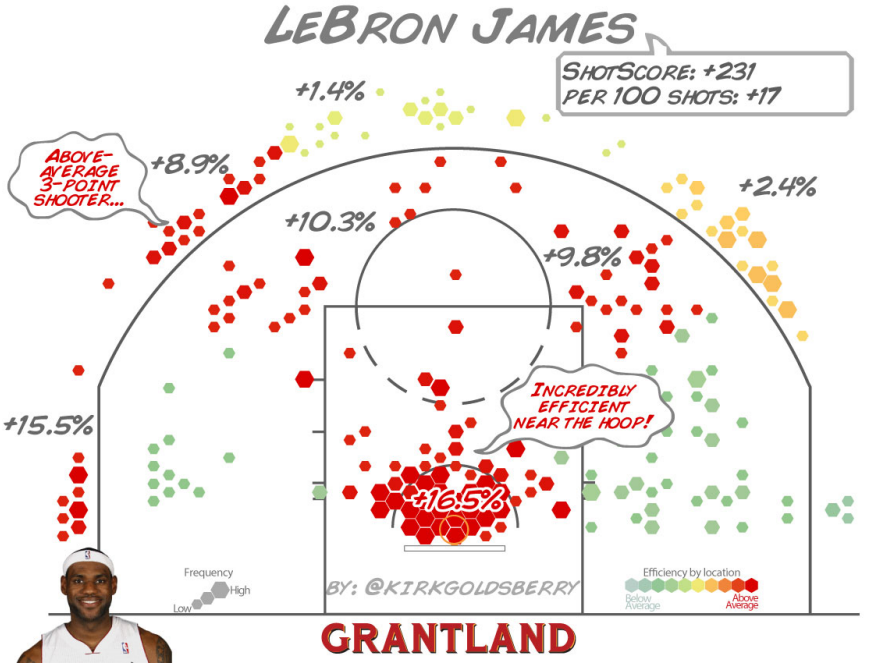
Performance Metrics

Performance metrics are usually repeated measures of team or players that are derived from historical box scores.



Tracking Data

'Tracking data' is spatio-temporal data of objects in a sporting event.

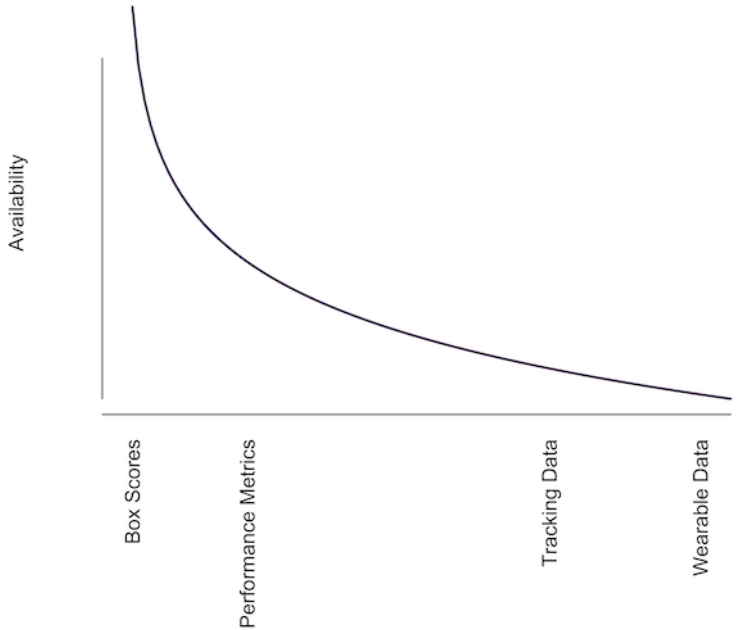


Wearables

- 'Wearables' are technologies that collect data about athletes when worn.
- Biometrics and movement are two common types of data collected.

Data Availability Curve

Sadly, not all types of sports data are equally available though the steepness of the curve depends on the particular sport.



Web Resources

R Resources

Sport	Package
General	stattleshipR
	SportsAnalytics
	odds.converter
Soccer	engsoccerdata
Baseball	pitchRx
	Lahman
Tennis	deuce
Cricket	cricketr!

More on `deuce`

More on `deuce`

- `deuce` is a package I created to make it easy to access large historical data on tennis matches

More on deuce

- deuce is a package I created to make it easy to access large historical data on tennis matches
- It combines data from multiple sites:
  - flashscore.com
  - ATP Tour
  - WTA Tour
  - Tennis Abstract

More on deuce

- deuce is a package I created to make it easy to access large historical data on tennis matches
- It combines data from multiple sites:
    - flashscore.com
    - ATP Tour
    - WTA Tour
    - Tennis Abstract
- Some of the data you can obtain from deuce includes:
    - Match results from Open Era (1968) to the present
    - Historical rankings
    - Player demographics
    - Point-level data for multiple years at Grand Slams
    - Shot level data from the Match Charting Project

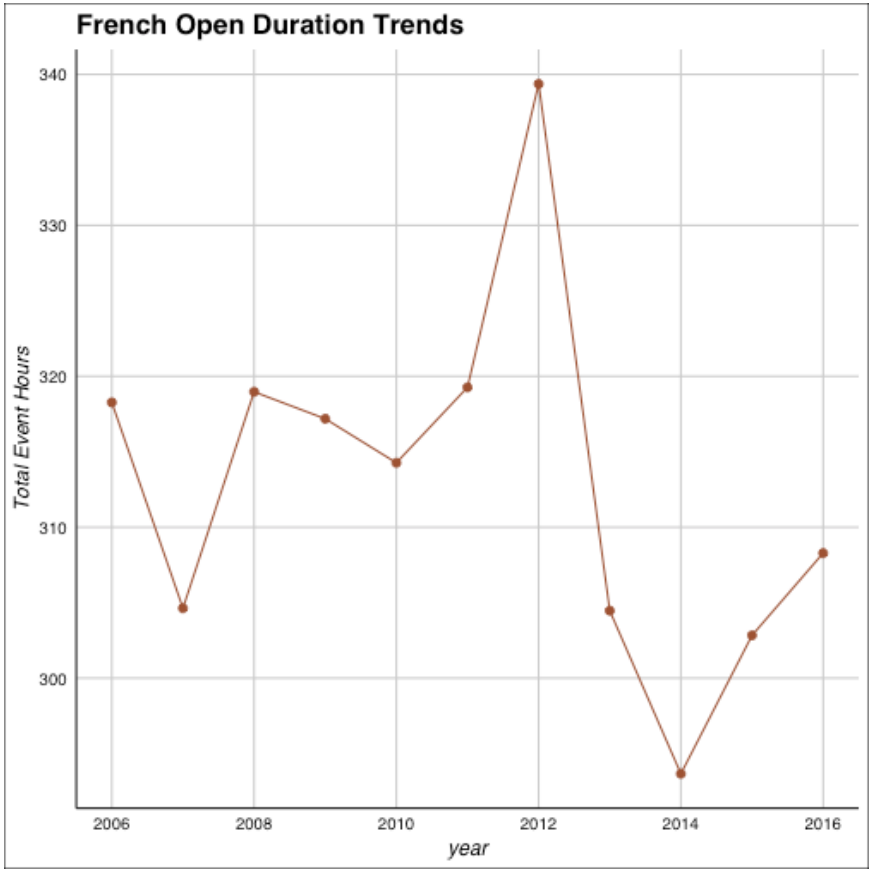
Installing deuce

You can install deuce using `devtools`. It may take several minutes because of the size of the datasets being transferred.

```
library(devtools)

devtools::install_github("skoval/deuce")
```

Illustrative Analysis: Match Duration

```
library(deuce)
library(dplyr)

data(atp_matches, package = "deuce")

french_minutes <- atp_matches %>%
  dplyr::filter(tourney_name == "Roland Garros",
         year >= 2006 & year < 2017)

french_minutes %>%
  group_by(year) %>%
  dplyr::summarise(
    total.time = sum(minutes, na.rm = T)
  ) %>%
  ggplot(aes(y = total.time / 60, x = year)) +
  geom_point(size = 2, col = "sienna") + geom_line(col = "sienna") +
  scale_y_continuous("Total Event Hours") +
  theme_gdocs() +
  ggtitle("French Open Duration Trends")
```


French Open Duration Trends
340
330
320
310
300
Total Event Hours
2006
2008
2010
2012
2014
2016
year

Scraping Sports Data

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)
```

Scraping from a Website

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

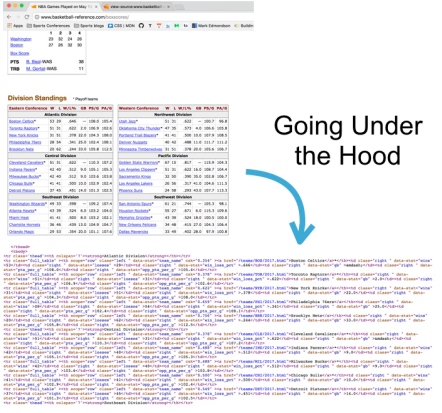
Scraping from a Website

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

- Web data is *static* if you can see it in the source code.
- If you can't, the data is *dynamically* generated and you will need automation to extract it.

How Do I Know if Data is Static or Dynamic?

- You need to be able to inspect HTML and CSS
- I recommend Google Chrome's developer tools
- Get familiar with "View Source"
- Try out the Selector Gadget



Example: 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/

Scraping Static Data with `rvest`

Scraping Static Data with `rvest`

- The `rvest` package is a suite of tools for scraping static Web data and putting them in easy-to-use objects (like data frames)

Scraping Static Data with `rvest`

- The `rvest` package is a 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

Scraping Static Data with `rvest`

- The `rvest` package is a 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

Scraping Static Data with `rvest`

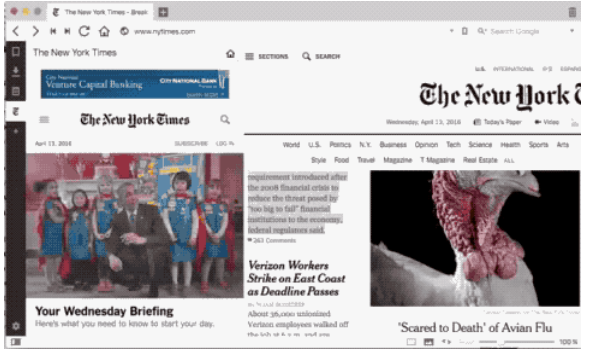
- The `rvest` package is a suite of tools for scraping static Web data and putting them in easy-to-use objects (like data frames)
- Works with `magrittr` and allows piping commands with `%>%` operator
- Allows some browsing functionality
- Authored by Hadley Wickham

Example: Scraping Elo Ratings

```
library(rvest)

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)
```

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

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.

```
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

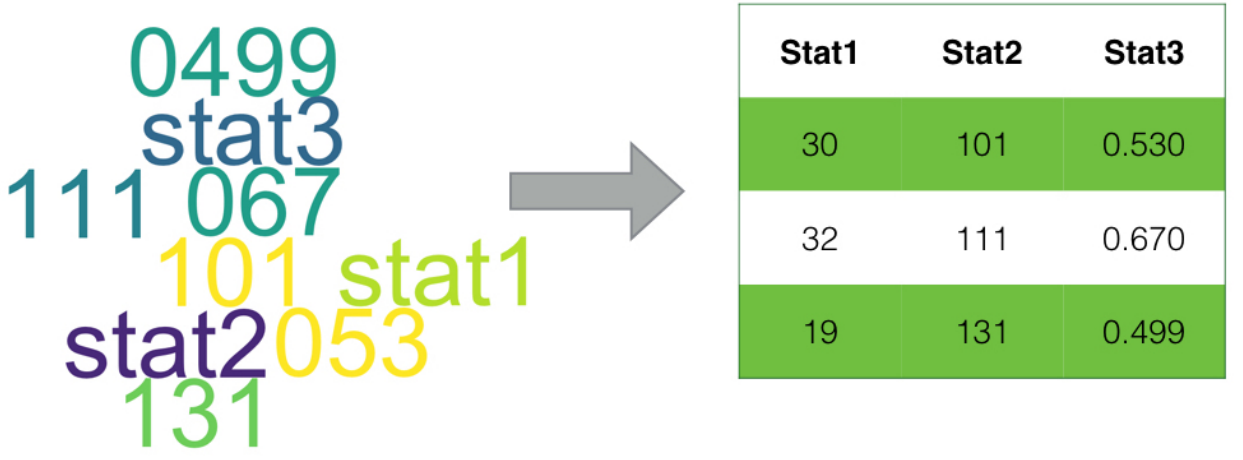
Next we extract the table of stats using the CSS `id` node.

```
# 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
```

Data Wrangling

Goal of Wrangling

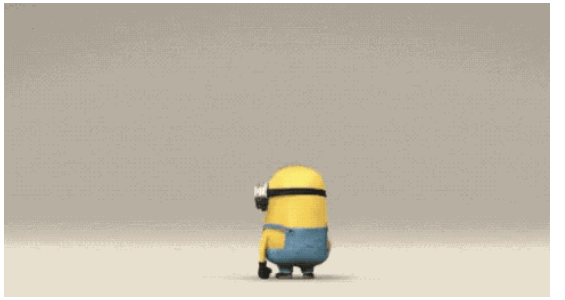
The purpose of *wrangling* is to go from untidy data to a tidy row-by-column structure of our data, that has clearly named variables and valid values.



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

R's Arsenal for Wrangling

- `stringr` for string manipulation
- `tidyr` a grammar for reshaping data
- `dplyr` a grammar for filtering and transforming data



Wrangling by Example

Consider this string of tennis match statistics scraped from the Web. How could we get this into a nice data structure?

```
match_stats # Example of untidy data
```

```
[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 Steps Do We Need?

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

Goal Structure

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

Statistic	Value	Player

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 combining `stringr` and regular expressions.
- Here, 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
```

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
```

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")

result <- data.frame(
    stat = rep(stats, ifelse(counts, 2, 4)),
    values = values
)

result
```

```
##                          stat   values
## 1                        Aces        7
## 2                        Aces        1
## 3               Double Faults        2
## 4               Double Faults        3
## 5                   1st Serve      71%
## 6                   1st Serve  (33/45)
```

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"
```

Reshaping a Data Frame

We frequently need to go between wide and long formats of our data. With `tidyr` we can use `gather` to go from wide to long and `spread` to go from long to wide.

Below is an example of reshaping our `match_stats` going from long to wide.

```
library(tidyr)

# Assign player to their stat
result <- result %>%
    group_by(stat) %>%
    dplyr::mutate(
      player = rep(c(1, 2), each = n() / 2)
    )

result %>%
  filter(stat %in% c("Aces", "Double Faults")) %>%
  tidyr::spread(stat, values)
```

```
## # A tibble: 2 × 3
##   player   Aces `Double Faults`
## *  <dbl> <fctr>          <fctr>
## 1      1      7               2
## 2      2      1               3
```

Exploring Our Data

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 By Asking 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?
- This means getting to know your variables with summary functions like: `str`, `table`, and `summary`

Exploratory Charting

Exploratory Charting

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

Exploratory Charting

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

Exploratory 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

Exploratory 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')`.

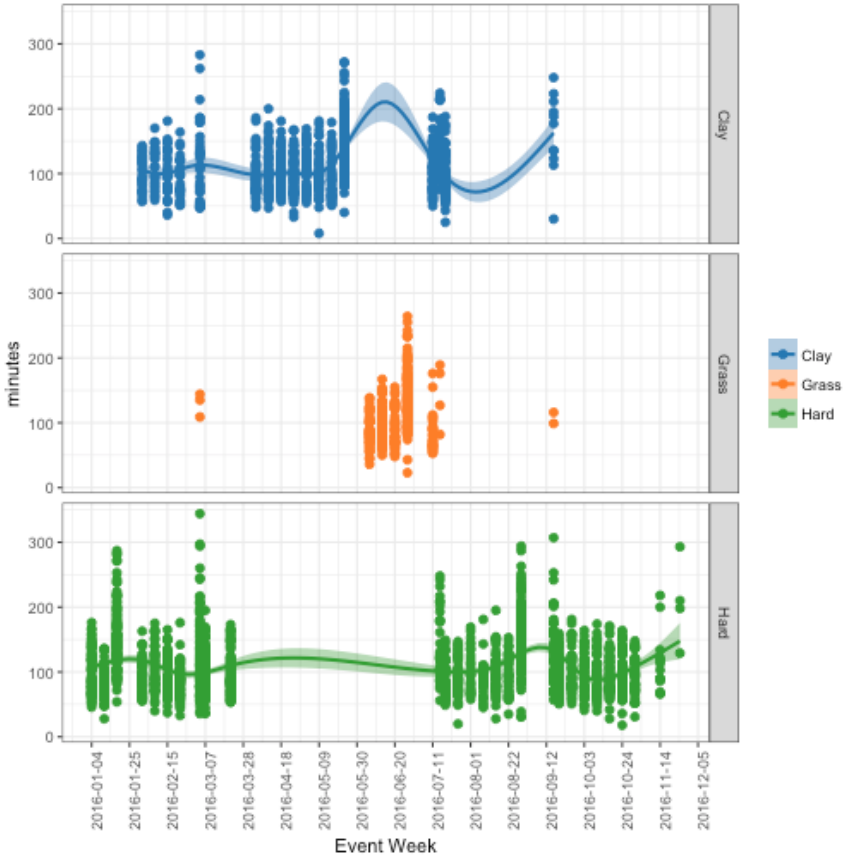
Function.Type	Description
aes	Relates variables to axes and aesthetic elements of our plot
geoms	Define how the variables will be displayed, that is, the type of chart
Function.Type	**Description**
facet	Splits plot into rows and columns defined by a group variable
scales	Customizes the range and limits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

Example: Match Duration Trends by Surface

In this example, we explore patterns in match duration by surface using a number of features in `ggplot2` and `ggthemes`.

```
data(atp_matches)
```

```
atp_matches %>%
  filter(!(minutes == 0 | minutes > 10 * 60), surface != "Carpet") %>
    ggplot(aes(y = minutes, x = tourney_date,
               col = surface, fill = surface)) +
  facet_grid(surface ~ .) + # Facet by row with no column facet
  geom_smooth() +
    geom_point(size = 2) +
  scale_color_tableau(name = "") +
  scale_fill_manual(name = "",
                    values = alpha(tableau_color_pal()(3), 0.5)) +
  scale_x_date("Event Week", date_breaks = "3 week") +
    theme_bw() +
  theme(axis.text.x = element_text(angle = 90))
```


minutes
300
200
100
0
Clay
Grass
Hard
2016-01-04
2016-01-25
2016-02-15
2016-03-07
2016-03-28
2016-04-18
2016-05-09
2016-05-30
2016-06-20
2016-07-11
2016-08-01
2016-08-22
2016-09-12
2016-10-03
2016-10-24
2016-11-14
2016-12-05
Event Week
Clay
Grass
Hard

3D Charting

- So far, we have only considered two-dimensional charts.
- For 3D data, we can use the package `plot3D`.
- But first, we need to extract the (x, y, z) coordinates for each serve along a set of time points.

Plotting Tennis Shots

In this example, we will look at some tracking data for about 60 tennis serves of ATP players. The data includes the x, y, and z locations of the flight of the serves.

```
positions[1:10,] # Serve position data
```

```
##     seconds         x         y        z id
## 1  1.532434 -11.39293 0.7152673 3.089983  1
## 2  1.532823 -11.36955 0.7105796 3.086929  1
## 3  1.533213 -11.34617 0.7058936 3.083875  1
## 4  1.533602 -11.32280 0.7012094 3.080821  1
## 5  1.533991 -11.29944 0.6965270 3.077767  1
## 6  1.534381 -11.27608 0.6918463 3.074713  1
## 7  1.534770 -11.25274 0.6871674 3.071659  1
## 8  1.535159 -11.22940 0.6824902 3.068605  1
## 9  1.535549 -11.20607 0.6778148 3.065551  1
## 10 1.535938 -11.18275 0.6731411 3.062497  1
```

Package `plot3D`

- The `plot3D` package provides a suite a functions for 3D plots: `points3D`, `lines3D`, `ribbon3D`, etc.
- You can install with `install.packages('plot3D')`

Plot3D Key Arguments

In addition to the basic variables `x`, `y`, `z`, here are some useful arguments when working with `plot3D`.

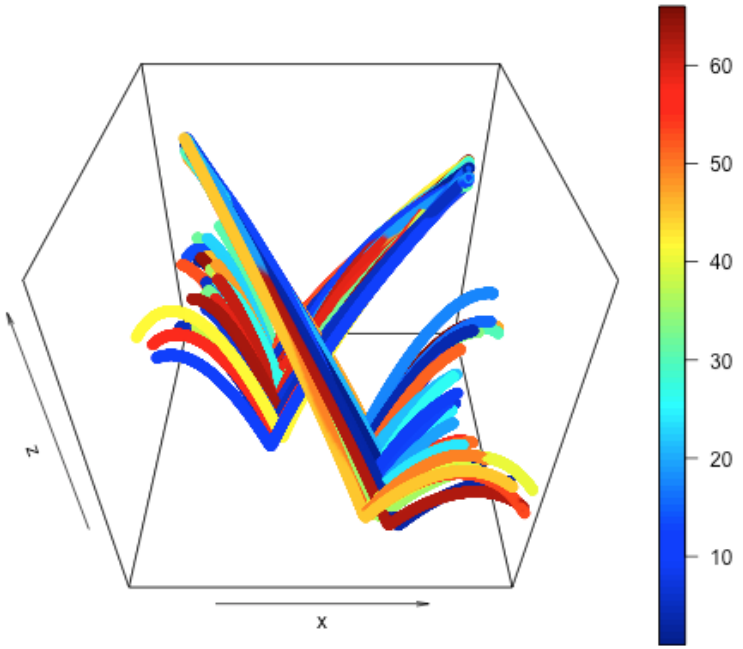
Argument	Description
colvar	Numeric group variable for coloring points
phi	Angle defining colatitude
theta	Angle defining azimuthal direction
col	Colors to use with colvar
bty	Style for the perspective box

| Argument | Description |

Plotting Serves

In this chart, we plot all serves using the `points3D`. We set `theta` to 0 so that we look perpendicularly along the length of the court.

```
library(plot3D) # Load plot3D

points3D(
    x = positions$x,
    y = positions$y,
    z = positions$z,
    colvar = positions$id,
    theta = 0
)
```


z
x
60
50
40
30
20
10

Prediction Models

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

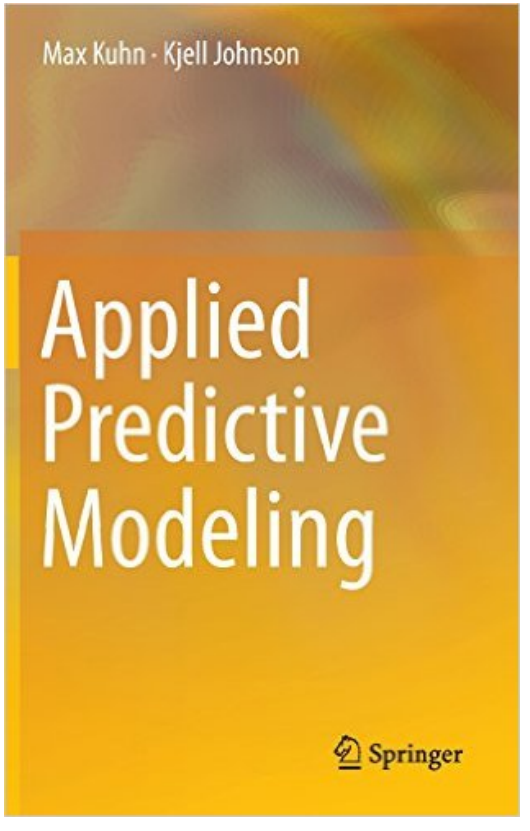
- *Predictive modeling* usually refers to machine learning
- The goal of these techniques is to improve predictive performance
- This is a very different goal from statistical models, like regression, where inference and interpretation are of primary importance
- Predictive models will often sacrifice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

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 $n \sim p$)

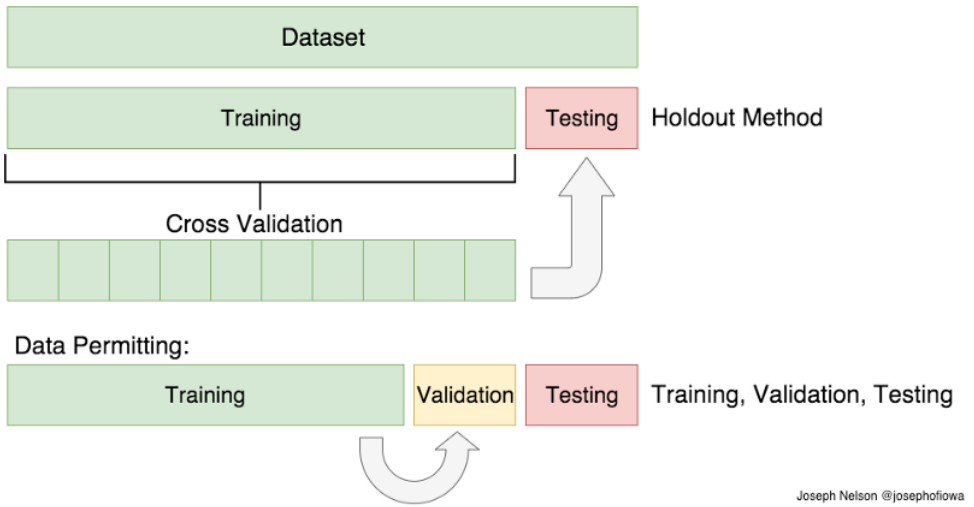
The `train` Function

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

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
)
```

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, adabag, ada, blackboost, C5.0
Support Vector Machines	Collection of regression lines that try to maximally separate classes	svmLinear, svmRadial

Category Description Examples

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.

```
set.seed(1115)

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

Evaluate the Model

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

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

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

Application: Emotion Detection

At GIG we have recently been using caret to create prediction models for player emotions based on Facial Action Units in match broadcasts.

How much does this expression indicate:

- Anxiety?
- Determination?
- Excitement?

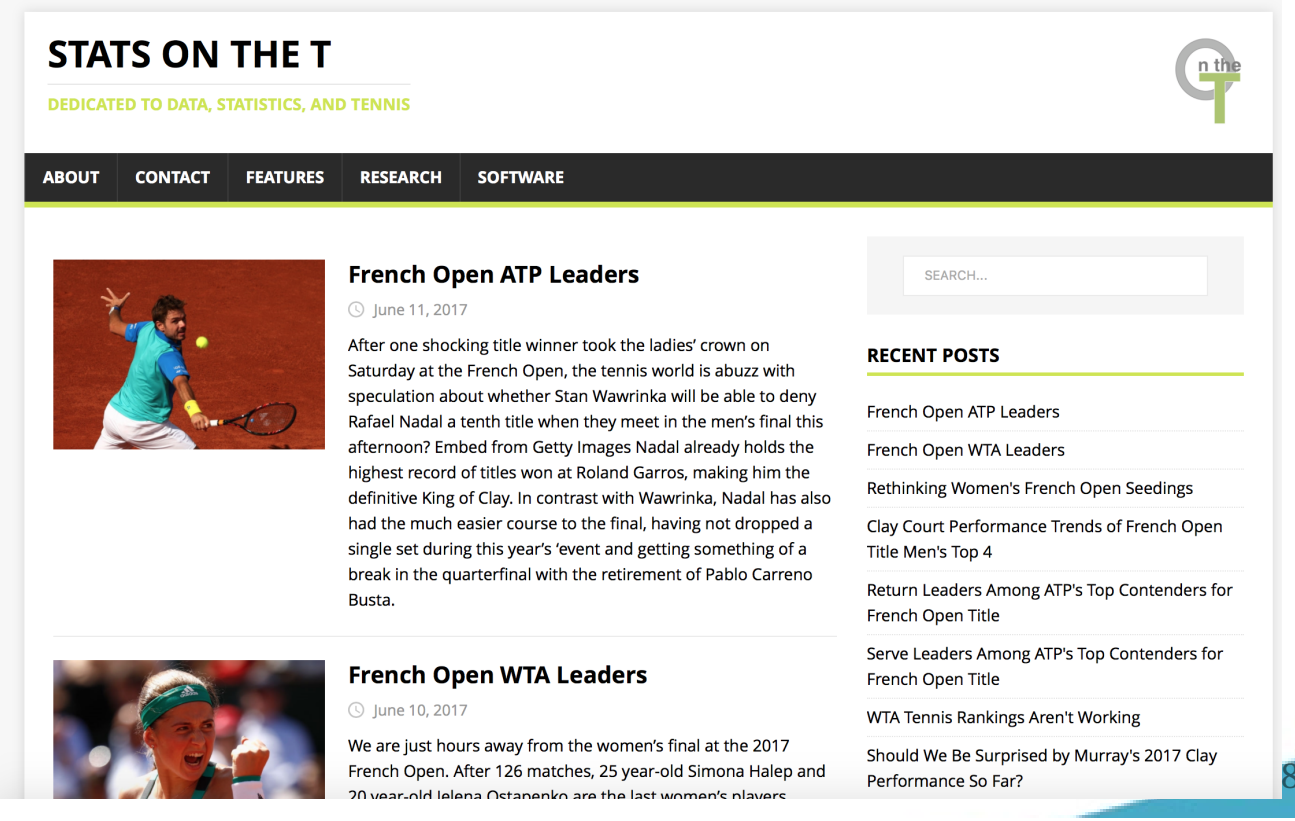
Blogging

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

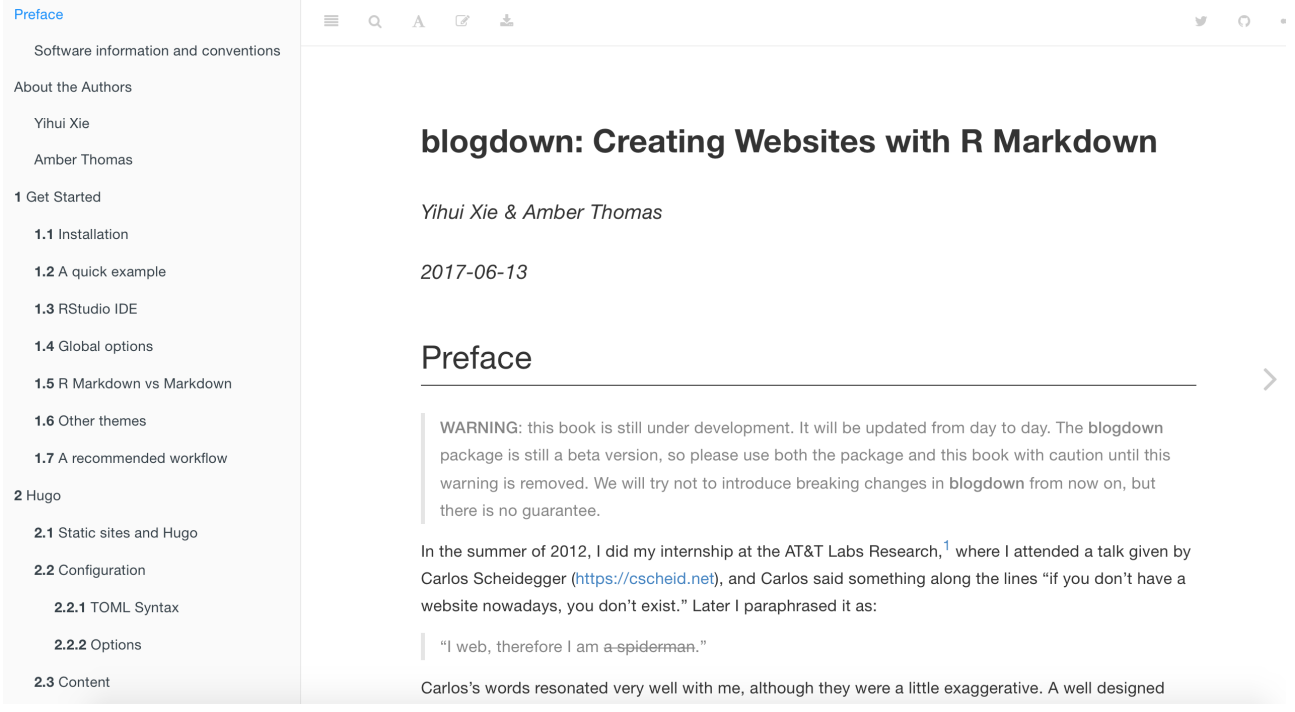
Here are the major functions you will use with `blogdown`.

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

| Function | Description |

Learning the Workflow

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

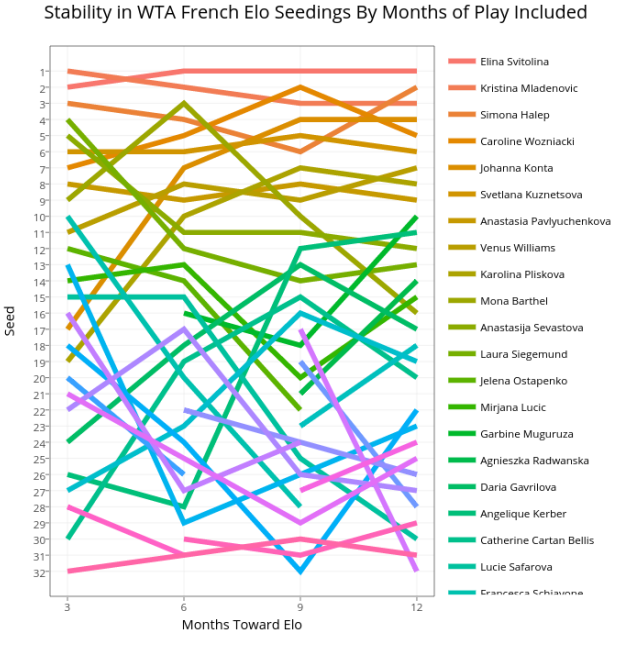


Worklow Overview

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.

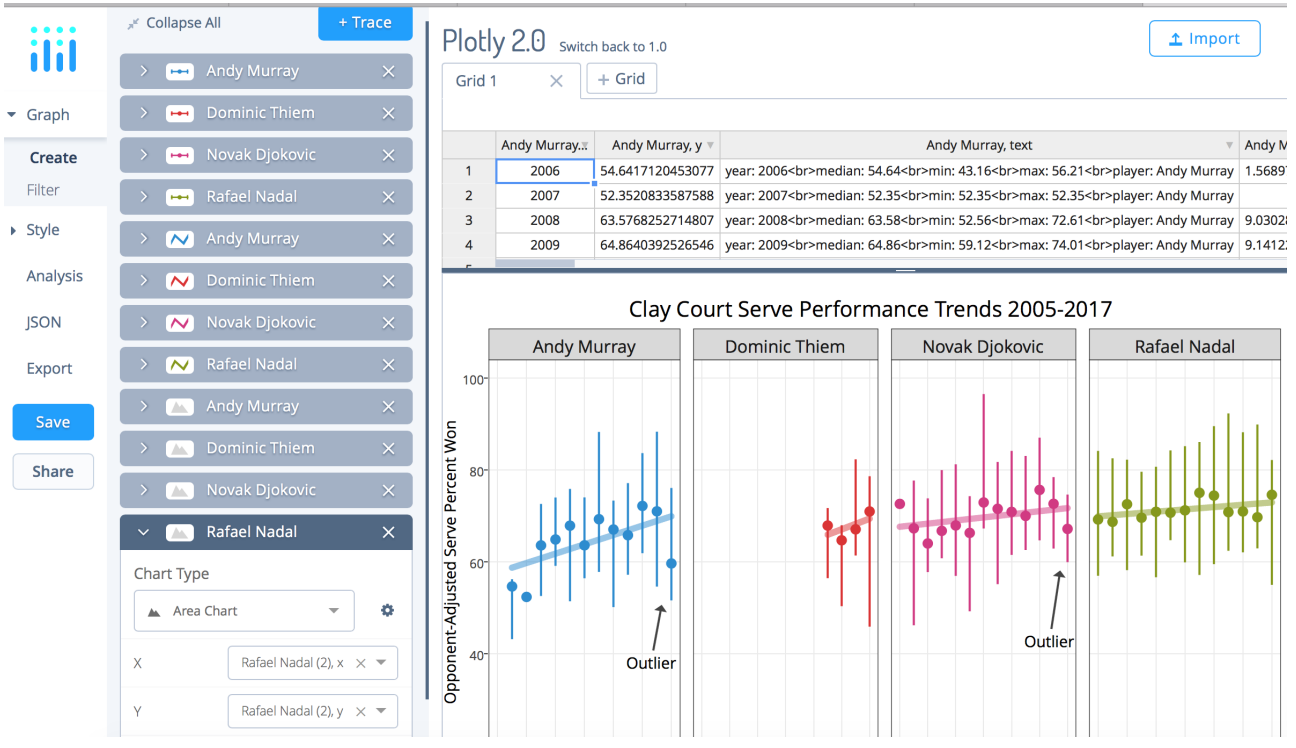
Getting Started with Plotly

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
```

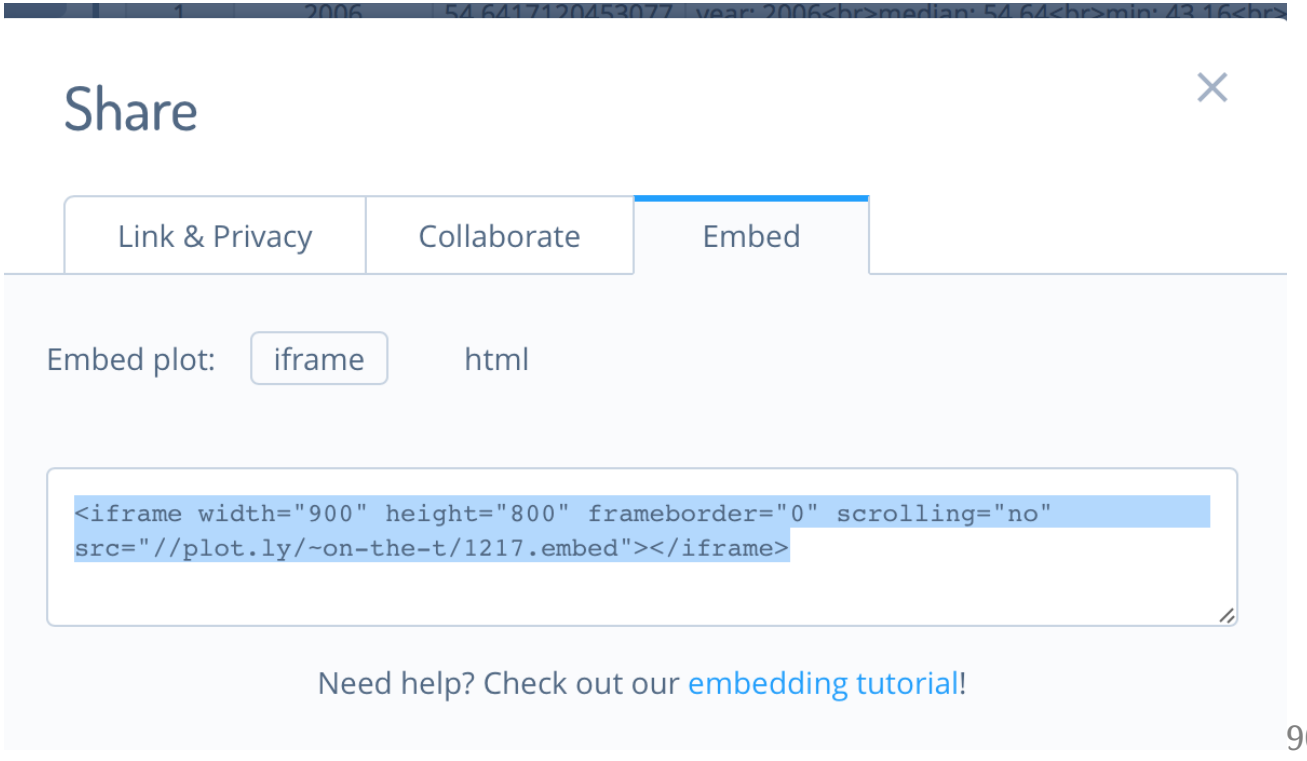
Plotly Site

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

skovalchik@tennis.com.au

@StatsOnTheT