Web Scraping HTML Tables

Amber Brodeur & Olga Dobromyrova

January 2017

Contents

Introduction

Web scraping, or web crawling, is a useful technique for data scientists and analysts to extract data from the Internet. At times, the data needed to complete a project is not readily available or provided. If the data is available on the web, in most cases it can be extracted by scraping it. However, some websites have restrictions on web scraping and data cannot be scraped because of legality repercussions, like privacy laws. For example, it is against the user agreement to scrape data off from LinkedIn.[1] Violating the user agreement by scraping LinkedIn violates several state and federal statutes, including the Computer Fraud and Abuse Act enacted by Congress in 1986.[2] There are different types of files on the web, one type is HyperText Markup Language (HTML), which is, "A language that web pages are created in. HTML isn't a programming language, like Python – instead, it's a markup language that tells a browser how to layout content."[3] In this tutorial, we walk through the steps to scrape a HTML table on NFL data off from two ESPN websites.

Objective

The objective of this project is to scrape a HTML table, clean the data, and export the data into a csv file.

Dataset Description

The dataset extracted was the Total Touchdowns Leaders table from ESPN's NFL Player Statistics 2016 section on the following two websites:

http://www.espn.com/nfl/statistics/player/_/stat/scoring/sort/totalTouchdowns/qualified/false [4]

[1] https://www.linkedin.com/help/linkedin/answer/56347/prohibition-of-scraping-software?lang=en
[2] https://assets.documentcloud.org/documents/3011889/Linkedin-v-Does.pdf
[3] https://www.dataquest.io/blog/web-scraping-tutorial-python/
[4] http://www.espn.com/nfl/statistics/player/_/stat/scoring/sort/totalTouchdowns/qualified/false

http://www.espn.com/nfl/statistics/player/_/stat/scoring/sort/totalTouchdowns/qualified/false/count/41 [5]

The dataset is on NFL players' statistics from the 2016 regular football season. The data contains information about players, teams, and players' statistics. Players' statistics includes data like: rushing touchdowns, field goals, and total points per game.

Variables

The dataset consists of the following variables:

Variable	Description
RK	Index number of a player in the table.
Player	Player's name and position.
Team	Team name abbreviation.
RUSH	Rushing touchdowns.
REC	Receiving touchdowns.
RET	Return touchdowns.
TD	Total touchdowns.
FG	Field goals.
XP	Extra points.
2PT	Two-point conversions.
PTS	Total points.
PTS/G	Total points per game.

Dataset Extraction

Install the packages used in this project.

```
install.packages('rvest')
install.packages('tidyr')
install.packages('dplyr')
```

Here are short descriptions of the capabilities of each package:

- `rvest` includes the `xml2` and `httr` packages that help to download HTML and XML data
- `tidyr` helps to make data tidy by organizing information in variables (columns) and observations (rows)
- `dplyr` manipulates data easily and efficiently

Load the packages into the workplace.

```
library(rvest)
library(tidyr)
library(dplyr)
```

The `read_html` function is used to read the webpage.

```
website <- read_html('http://www.espn.com/nfl/statistics/player/_/stat/scoring/sort/totalTouchdowns/qua
```

Next, we used the `html_nodes` command, which specifies the webpage to be worked with and the type of node ("table" in our case). This function extracts the list of nodes of the specified type. Next, the data

[5] http://www.espn.com/nfl/statistics/player/_/stat/scoring/sort/totalTouchdowns/qualified/false/count/41

table is saved as a data frame object. We used the `html_table` command to specify the index of the table we wanted to extract. This index can be found manually by "trial and error" or by looking at the HTML document structure.

```
td.table <- html_nodes(website, 'table')
td <- html_table(td.table)[[1]]
head(td)
```

```
##   X1                    X2   X3         X4         X5         X6
## 1                              TOUCHDOWNS TOUCHDOWNS TOUCHDOWNS
## 2 RK                PLAYER TEAM       RUSH        REC        RET
## 3  1     David Johnson, RB  ARI         16          4          0
## 4  2 LeGarrette Blount, RB   NE         18          0          0
## 5  3   Ezekiel Elliott, RB  DAL         15          1          0
## 6  4      Jordy Nelson, WR   GB          0         14          0
##           X7      X8      X9     X10     X11     X12
## 1 TOUCHDOWNS SCORING SCORING SCORING SCORING SCORING
## 2         TD      FG      XP     2PT     PTS   PTS/G
## 3         20       0       0       1     122     7.0
## 4         18       0       0       0     108     6.0
## 5         16       0       0       0      96     6.0
## 6         14       0       0       0      84     5.0
```

The football data is on two webpages, so we extracted the data on both webpages by repeating all of the previous steps.

```
website1 <- read_html('http://www.espn.com/nfl/statistics/player/_/stat/scoring/sort/totalTouchdowns/qua
td.tbl1 <- html_nodes(website1, 'table')
td1 <- html_table(td.tbl1)[[1]]
head(td1)
```

```
##   X1                    X2     X3         X4         X5         X6
## 1                              TOUCHDOWNS TOUCHDOWNS TOUCHDOWNS
## 2 RK                PLAYER   TEAM       RUSH        REC        RET
## 3 41      Kyle Rudolph, TE    MIN          0          7          0
## 4         Doug Baldwin, WR    SEA          0          7          0
## 5    Christine Michael, RB GB/SEA          6          1          0
## 6      Kelvin Benjamin, WR    CAR          0          7          0
##           X7      X8      X9     X10     X11     X12
## 1 TOUCHDOWNS SCORING SCORING SCORING SCORING SCORING
## 2         TD      FG      XP     2PT     PTS   PTS/G
## 3          7       0       0       0      42     2.0
## 4          7       0       0       0      42     2.0
## 5          7       0       0       0      42     4.0
## 6          7       0       0       0      42     2.0
```

The data is now extracted from the second HTML table. Next, the two data frames are merged into one data frame. The data frames have identical structure, so we used the `rbind` command, which combines data frames by rows. To ensure the data was merged correctly, use the `View` command to view the complete data set, or `head` command to see the first few data entries.

```
td<- rbind(td, td1)
head(td, n=10)
```

```
##    X1                    X2   X3         X4         X5         X6
## 1                               TOUCHDOWNS TOUCHDOWNS TOUCHDOWNS
## 2  RK                PLAYER TEAM       RUSH        REC        RET
```

```
## 3   1     David Johnson, RB  ARI         16          4          0
## 4   2 LeGarrette Blount, RB   NE         18          0          0
## 5   3   Ezekiel Elliott, RB  DAL         15          1          0
## 6   4      Jordy Nelson, WR   GB          0         14          0
## 7          LeSean McCoy, RB  BUF         13          1          0
## 8   6   Devonta Freeman, RB  ATL         11          2          0
## 9   7     Antonio Brown, WR  PIT          0         12          0
## 10       DeMarco Murray, RB  TEN          9          3          0
##             X7      X8      X9     X10     X11     X12
## 1  TOUCHDOWNS SCORING SCORING SCORING SCORING SCORING
## 2          TD      FG      XP     2PT     PTS   PTS/G
## 3          20       0       0       1     122     7.0
## 4          18       0       0       0     108     6.0
## 5          16       0       0       0      96     6.0
## 6          14       0       0       0      84     5.0
## 7          14       0       0       1      86     5.0
## 8          13       0       0       0      78     4.0
## 9          12       0       0       0      72     4.0
## 10         12       0       0       0      72     4.0
```

```
View(td)
```

The data set was combined correctly; however, it looks messy and requires some data cleaning before saving it as a csv file.

First, we filtered out all rows with duplicate header names. Use the `View` command to observe the `td` table.

```
td <- td[td$X4 != "TOUCHDOWNS",]
td <- td[td$X2 != "PLAYER",]
head(td, n=10)
```

```
##    X1                    X2  X3 X4 X5 X6 X7 X8 X9 X10 X11 X12
## 3   1     David Johnson, RB ARI 16  4  0 20  0  0   1 122 7.0
## 4   2 LeGarrette Blount, RB  NE 18  0  0 18  0  0   0 108 6.0
## 5   3   Ezekiel Elliott, RB DAL 15  1  0 16  0  0   0  96 6.0
## 6   4      Jordy Nelson, WR  GB  0 14  0 14  0  0   0  84 5.0
## 7          LeSean McCoy, RB BUF 13  1  0 14  0  0   1  86 5.0
## 8   6   Devonta Freeman, RB ATL 11  2  0 13  0  0   0  78 4.0
## 9   7     Antonio Brown, WR PIT  0 12  0 12  0  0   0  72 4.0
## 10       DeMarco Murray, RB TEN  9  3  0 12  0  0   0  72 4.0
## 11      Latavius Murray, RB OAK 12  0  0 12  0  0   0  72 5.0
## 12           Mike Evans, WR  TB  0 12  0 12  0  0   2  76 4.0
```

```
View(td)
```

The variables are assigned names using the `names` command.

```
names(td) <- c("rank", "player", "team", "rush", "rec", "ret", "td", "fg", "xp", "2pt", "pts", "ppg")
head(td, n=10)
```

```
##    rank                player team rush rec ret td fg xp 2pt pts ppg
## 3     1     David Johnson, RB  ARI   16   4   0 20  0  0   1 122 7.0
## 4     2 LeGarrette Blount, RB   NE   18   0   0 18  0  0   0 108 6.0
## 5     3   Ezekiel Elliott, RB  DAL   15   1   0 16  0  0   0  96 6.0
## 6     4      Jordy Nelson, WR   GB    0  14   0 14  0  0   0  84 5.0
## 7            LeSean McCoy, RB  BUF   13   1   0 14  0  0   1  86 5.0
## 8     6   Devonta Freeman, RB  ATL   11   2   0 13  0  0   0  78 4.0
## 9     7     Antonio Brown, WR  PIT    0  12   0 12  0  0   0  72 4.0
```

```
## 10         DeMarco Murray, RB  TEN    9   3   0 12  0  0   0  72 4.0
## 11        Latavius Murray, RB  OAK   12   0   0 12  0  0   0  72 5.0
## 12             Mike Evans, WR   TB    0  12   0 12  0  0   2  76 4.0
```

Next, we handled the missing values in the variable `RK` (rank). Below, we fixed the missing values by filling the column `RK` with a consecutive sequence of index numbers.

```
td$rank <- 1:length(td$player)
head(td, n=10)
```

```
##    rank               player team rush rec ret td fg xp 2pt pts ppg
## 3     1     David Johnson, RB  ARI   16   4   0 20  0  0   1 122 7.0
## 4     2 LeGarrette Blount, RB   NE   18   0   0 18  0  0   0 108 6.0
## 5     3   Ezekiel Elliott, RB  DAL   15   1   0 16  0  0   0  96 6.0
## 6     4      Jordy Nelson, WR   GB    0  14   0 14  0  0   0  84 5.0
## 7     5      LeSean McCoy, RB  BUF   13   1   0 14  0  0   1  86 5.0
## 8     6   Devonta Freeman, RB  ATL   11   2   0 13  0  0   0  78 4.0
## 9     7     Antonio Brown, WR  PIT    0  12   0 12  0  0   0  72 4.0
## 10    8    DeMarco Murray, RB  TEN    9   3   0 12  0  0   0  72 4.0
## 11    9   Latavius Murray, RB  OAK   12   0   0 12  0  0   0  72 5.0
## 12   10        Mike Evans, WR   TB    0  12   0 12  0  0   2  76 4.0
```

The variable `player` needs to be separated into two variables. Both the player's name and the player's position are in the same column seperated by a comma. Below, we used the `separate` command to break-up the `player` column into two seperate columns named `player` and `position`. We used the command `head` to look at the first 10 rows of data.

```
td <- separate(td, player, c('player', 'position'), sep=', ', remove=TRUE)
head(td, n=10)
```

```
##    rank            player position team rush rec ret td fg xp 2pt pts ppg
## 3     1     David Johnson       RB  ARI   16   4   0 20  0  0   1 122 7.0
## 4     2 LeGarrette Blount       RB   NE   18   0   0 18  0  0   0 108 6.0
## 5     3   Ezekiel Elliott       RB  DAL   15   1   0 16  0  0   0  96 6.0
## 6     4      Jordy Nelson       WR   GB    0  14   0 14  0  0   0  84 5.0
## 7     5      LeSean McCoy       RB  BUF   13   1   0 14  0  0   1  86 5.0
## 8     6   Devonta Freeman       RB  ATL   11   2   0 13  0  0   0  78 4.0
## 9     7     Antonio Brown       WR  PIT    0  12   0 12  0  0   0  72 4.0
## 10    8    DeMarco Murray       RB  TEN    9   3   0 12  0  0   0  72 4.0
## 11    9   Latavius Murray       RB  OAK   12   0   0 12  0  0   0  72 5.0
## 12   10        Mike Evans       WR   TB    0  12   0 12  0  0   2  76 4.0
```

The data looks clean. Next, we check data types by observing the structure of the data frame.

```
str(td)
```

```
## 'data.frame':    80 obs. of  13 variables:
##  $ rank    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ player  : chr  "David Johnson" "LeGarrette Blount" "Ezekiel Elliott" "Jordy Nelson" ...
##  $ position: chr  "RB" "RB" "RB" "WR" ...
##  $ team    : chr  "ARI" "NE" "DAL" "GB" ...
##  $ rush    : chr  "16" "18" "15" "0" ...
##  $ rec     : chr  "4" "0" "1" "14" ...
##  $ ret     : chr  "0" "0" "0" "0" ...
##  $ td      : chr  "20" "18" "16" "14" ...
##  $ fg      : chr  "0" "0" "0" "0" ...
##  $ xp      : chr  "0" "0" "0" "0" ...
##  $ 2pt     : chr  "1" "0" "0" "0" ...
```

```
##  $ pts     : chr  "122" "108" "96" "84" ...
##  $ ppg     : chr  "7.0" "6.0" "6.0" "5.0" ...
```

All variables are of class character. There are two categorical variables and the players' statistics are numerical. Below, we converted `position` and `team` to factor variables, and all players' statistics are converted to class numerical. Two `for` loops were used; one to convert columns 3 and 4 into factor variables, and the other to convert columns 5-13 into numerical variables. The `class` command is used to check the classes of each variable in the data frame.

```
for(i in 3:4) {
  td[,i] <- as.factor(td[,i])
}

for(i in 5:13) {
  td[,i] <- as.numeric(td[,i])
}

class(td)
```

```
## [1] "data.frame"
```

The correct classes were assigned to all of the variables.

To save the data set as a csv file, we set the working directory and then used the `write.csv` command to export the data to the file path.

```
setwd("/Users/amberbrodeur/Desktop/Web Scrape")
write.csv(td, 'td.csv', row.names=FALSE)
```

Variable Summary

There are many ways to analyze the data we scraped. Below, we demonstrate a simple example by creating a distribution summary of points-per-game grouped by team.

```
td %>% select(team, ppg) %>%
  group_by(team) %>%
    summarise(min.ppg=min(ppg), max.ppg=max(ppg),  mean.ppg=mean(ppg)) %>%
      arrange(desc(max.ppg))
```

```
## # A tibble: 33 × 4
##      team min.ppg max.ppg mean.ppg
##    <fctr>   <dbl>   <dbl>    <dbl>
## 1     ARI       2       7 3.666667
## 2     DAL       1       6 3.000000
## 3      NE       2       6 4.000000
## 4     ATL       2       5 3.500000
## 5     BUF       2       5 3.333333
## 6      GB       4       5 4.500000
## 7     OAK       3       5 4.000000
## 8      SD       2       5 3.250000
## 9     CAR       2       4 2.666667
## 10 GB/SEA       4       4 4.000000
## # ... with 23 more rows
```

The output displays a distribution summary of the minimum points per game, maximum point per game, and mean points per game for each team.

Conclusion

This tutorial drove you through the steps to scrape an HTML table, clean the data, export the data as a csv file, and use the data for analysis. Well done!