

ODU Big Data, Data Wrangling Boot Camp Software Overview and Design

Chuck Cartledge

September 23, 2018

Contents

List of Tables	ii
List of Figures	ii
1 Introduction	1
2 Software system design	3
2.1 Twitter software front and back ends	3
2.1.1 Details	3
2.1.2 Design limitations	5
2.2 NASA reports	7
2.3 Configuration file	10
3 References	12
A Database tables	13
A.1 Twitter related tables	13
A.2 NASA related tables	13
B Notational data structures	13
B.1 Twitter structures	13
B.2 Senate bill structures	14
C Software on each workstation	15
D Files	17

List of Tables

1	Frontend and backend algorithm cross matrix.	4
2	Configuration file entries.	10
3	Tables to support Twitter analysis.	13
4	Tables to support NASA analysis.	13
5	Notional plotting data structure.	14
6	Notional plotting data structure.	14

List of Figures

1	Notional data science data flow.	2
2	Twitter system design.	4
3	Image from the “checkPostgres.R” script.	18

List of Algorithms

1	Process configuration file.	5
2	Update database with new data.	6
3	Normalize text.	7
4	Evaluate text.	7
5	Update display with new data.	8
6	Plotting hash tag based Tweet sentiment.	9
7	NASA report processing.	9

1 Introduction

The tweet sentiment analysis software used as part of the Old Dominion University College of Continuing Education and Professional Development Big Data: Data Wrangling boot camp¹ will be used to provide boot-camp attendees with hands-on experience doing data-wrangling of textual data.

“We define such data wrangling as a process of iterative data exploration and transformation that enables analysis. . . . In other words, data wrangling is the process of making data useful.”

Kandel et al. [2]

In the boot-camp, we will be:

- Looking at tweets to conduct sentiment analysis relative to arbitrary hashtags,
- Extracting data from static web pages based on cascading style sheets (CSS), and
- Extracting data from a NASA textual archive using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH).

We will be focusing on data wrangling (see Figure 1) using the R programming language.

Each boot-camp workstation will have the same software load (see Section C), and almost fully functional software in R. The twitter software will be complete, in that it will:

- Retrieve tweets from Twitter,
- Place tweets in a PostGres database,
- Retrieve tweets from the database,
- Tokenize the tweets,
- Qualify the tweets as positive, negative, or neutral, and
- Plot the results in different ways.

The CSS software will be complete, in that it will:

- Download a “hard coded” web page,
- Extract a data field based on a CSS selector,
- “Wrangle” the data as necessary,

¹<https://www.odu.edu/cepd/bootcamps/data-wrangling>

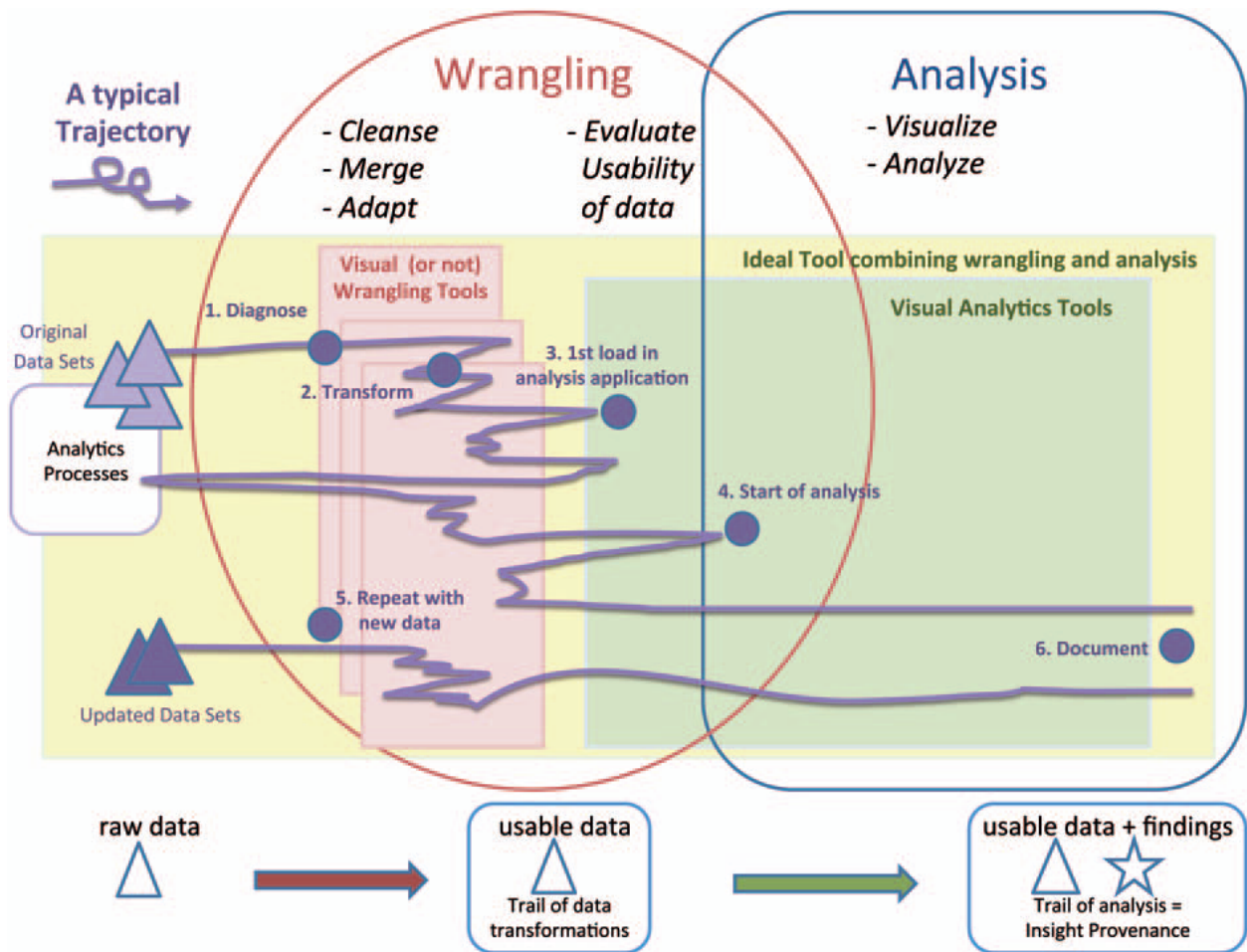


Figure 1: Notional data science data flow. Data wrangling requires domain specific knowledge to cleanse, merge, adapt, and evaluate raw data. Image from [2].

- Present the data.

The chrome browser and the SelectorGadget plugin² will be used to identify CSS selectors. The OAI-PMH software will be complete, in that it will:

- Download technical report meta data from the NASA Technical Reports Server³,
- Insert document IDs, report titles, and report descriptions into a PostGres database,
- Generate a static web page based on searching the PostGres data.

The code will be modified to display information about the reports based on how the textual data is wrangled.

Data wrangling will focus on:

1. Identifying problems with the tweet tokens,
2. Developing solutions to those problems, and
3. Reducing the number of problematic tokens.

The remaining sections layout in detail the overall system design, details of the major algorithms, database tables, and the configuration file used to control the system.

2 Software system design

2.1 Twitter software front and back ends

2.1.1 Details

The sytem is logically divided into three parts (see Figure 2):

1. A “backend” that gets tweets from Twitter or a data file.
2. A database to hold tweets from the backend.
3. A “frontend” that retrieves data from the data base for analysis and display.
4. The frontend and backend processes are controlled by the contents of a configuration file (see Section 2.3).

Details of the various algorithms used by the backend and frontend processes are outlined in this section.

²<https://chrome.google.com/webstore/detail/selectorgadget/mhjnkcfbdhnjickkkdbjoemdbfginb?hl=en>

³<https://ntrs.nasa.gov>

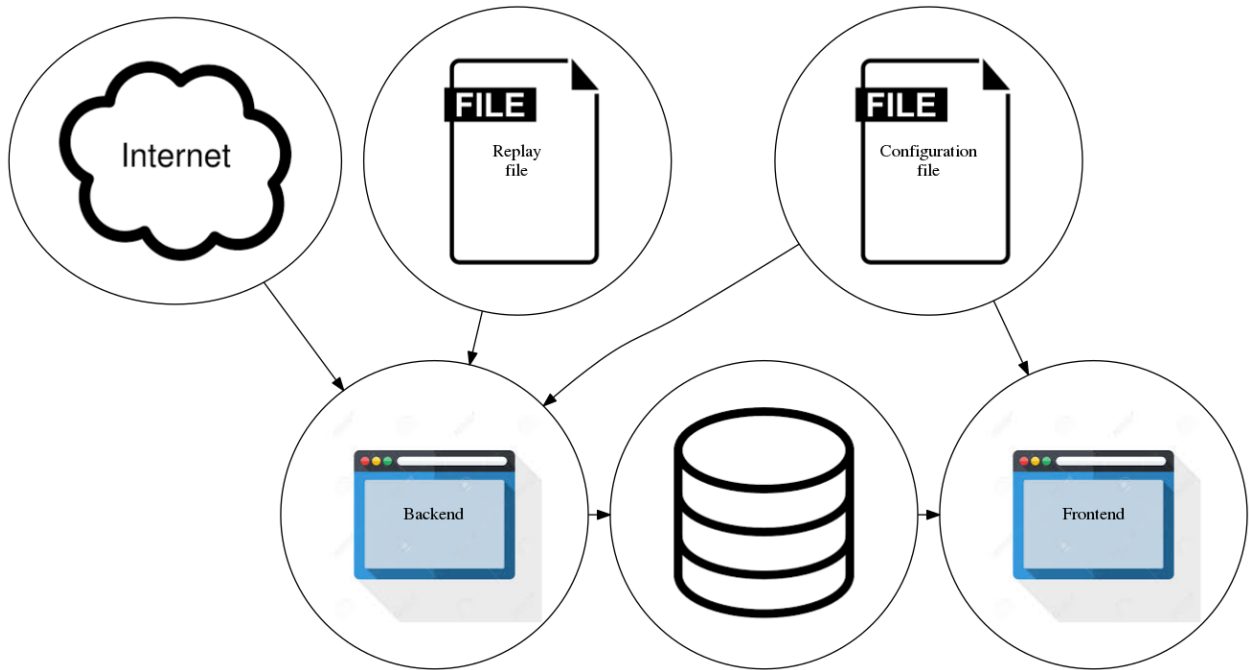


Figure 2: Twitter system design. Both front and back ends read data from a common configuration file, and use a shared library file of common functions.. The back end will receive data from the internet or from a replay file, based on directives in the configuration file and update the database with new data. The front end will connect to the database and retrieve data based on directives from the configuration file.

Table 1: Frontend and backend algorithm cross matrix. Algorithms that are used by both the front and back ends are recommended for a “utility” file or library that can be accessed by both ends.

Num.	Name	Back end	Front end
1	Process configuration file.	✓	✓
2	Update database with new data	✓	
3	Normalize text		✓
4	Evaluate text		✓
5	Update display with new data		✓
6	Plotting hash tag based Tweet sentiment		✓

```

Input: Location of configuration file
Assign default values;
while not at the end of the file do
|   read line;
|   if not a comment line then
|   |   get token;
|   |   get value;
|   |   if is a Hashtag then
|   |   |   add value to list of hashtags;
|   |   else
|   |   |   structure token value = value;
|   |   end
|   end
end

```

Result: A language specific data structure, values from file override defaults.

Algorithm 1: Process configuration file.

2.1.2 Design limitations

The current design polls Twitter for new tweets on a periodic basis. The entire list of search hash tags are polled, any returned tweets are stored in the database, and the system “sleeps” for a number of seconds (as per the configuration file). There are a number of factors that affect this processing cycle, including:

1. The number of hash tags being queried. Each poll takes a finite amount of time, even if no tweets are returned, so the more hash tags being queried, the longer it will take to service the complete list of tags.
2. Each tweet becomes a single row in the database. The more tweets that are returned from the query, the longer it takes to update the database with all the tweets.
3. Each tweet has a unique serial number. Each query includes the serial number of the earliest (the one furthest in the past) one of interest in order to get a complete and continuous tweet stream for the hash tag. The earliest acceptable tweet is updated after a successful query.
4. The no-cost query capability is limited to 100 tweets per query. If more than 100 tweets are created between queries, then the polling process will continue to fall further and further behind.

Because of these design and implementation limitations, if tweets are being created faster than the polling process can collect them, then the system will fall further and further behind.

```

Input: Language specific configuration structure
start = first time in data file;
if Offset = TRUE then
|   diff = now() - start;
else
|   diff = 0
end
time_end = start + SleepyTime + diff;
for Polls remaining do
|   if Live then
|   |   submit query to Twitter;
|   |   request data from Twitter;
|   |   for lines from Twitter do
|   |   |   extract time from JSON;
|   |   |   data = base64 encoding of entire JSON;
|   |   |   insert time and data into database;
|   |   |   if CollectionFile is not NULL then
|   |   |   |   append time and data to CollectionFile;
|   |   |   end
|   |   end
|   |   sleep SleepyTime;
|   else
|   |   read line from file;
|   |   parse line into time and data;
|   |   while time > time_end do
|   |   |   time_end = time_end + SleepyTime;
|   |   |   sleep SleepyTime;
|   |   end
|   |   insert time and data into database;
|   end
end

```

Result: An updated database.

Algorithm 2: Update database with new data.

Input: Text to be “normalized”, “stop word list”
cleansed = Null;
for *Text* **do**
| lower case Text;
| remove non-ASCII;
| stemming;
| **if** *Text* not in “stop word list” **then**
| | append Text to cleansed
| **end**
end
return *cleansed*;
Result: Normalized text
Algorithm 3: Normalize text.

Input: sourceText, baseLineText
numberOfSourceWords = number of words in baseLineText;
percentage = numberOfSourceWords / numberOfWordsInSourceText;
return *percentage*;
Result: Percentage of source text in baseLineText
Algorithm 4: Evaluate text.

The limitations imposed by a polling interface can be overcome by using a streaming interface⁴. A polling interface was used because it is simple to design, simple to implement, and simple to test. The back-end process could be replaced by a streaming interface without affecting the front-end process.

2.2 NASA reports

Processing the NASA reports is straight forward (see Algorithm 7).

⁴ R example:<http://bogdanrau.com/blog/collecting-tweets-using-r-and-the-twitter-streaming-api/>

Input: Language specific configuration structure
cleansedPositive = normalize positive words;
cleansedNegative = normalize negative words;
cleansedStopWords = normalize Stop words;
timeStart = minimum time from database ;
for *Polls remaining* **do**
 timeEnd = timeStart + SleepyTime;
 hash tag new data = NULL;
 lines = query database from timeStart to timeEnd;
 for *lines* **do**
 tweet = base64 decode of data;
 if *parse Tweet is GOOD* **then**
 extract text;
 extract hash tag from Tweet text;
 cleansedText = normalized text less cleansedStopWords;
 positive percentage = evaluate cleansedText vs. cleansedPositive;
 negative percentage = evaluate cleansedText vs. cleansedNegative;
 neutral percentage = 100 - positive percentage - negative percentage;
 update plotting information (hash tag, source, location);
 end
 end
 plot hash tag results;
 timeStart = timeEnd;
 sleep SleepyTime;
end
plot source information;
plot location information;
Result: An updated display.

Algorithm 5: Update display with new data.

Input: The previous/current plotting data, and new data

for *Each Tweet type* **do**

 set the lower left polygon point as the previous poll and the last previous type count;

 set the upper left polygon point as the previous poll and the last previous type count + next previous type count;

 set the lower right polygon point as the current poll and the current type type;

 set the upper right polygon point as the current poll and the current type count + next current type count;

 plot the polygon, filling it with then Tweet type color

end

for *Each Tweet type* **do**

 set previous Tweet count value to current Tweet count value;

end

Result: An updated display data structure, and display.

Algorithm 6: Plotting hash tag based Tweet sentiment. From the user's perspective, a stacked histogram is plotted. From a programatic perspective, each three filled polygons are plotted where the left and right edges are the poll number, and the vertical component is the number of Tweets per type (positive, neutral, and negative). The display will show the absolute number of Tweets, and the color bands will show the proportions of each type.

Input: The contents of the configuration file.

if *Reset the database* **then**

 create necessary database tables ;

 populate the database with report data ;

end

update the database tokens based on database data ;

define a search term ;

normalize the search term ;

search the database for documents that match the normalized term ;

create an html file based on the results ;

Result: An updated html file showing the query results.

Algorithm 7: NASA report processing.

2.3 Configuration file

Software processes are coordinated by control values in a shared configuration file.

1. A common configuration file to be used by both the data capture and the data presentation programs.
2. The file will default to a “well known” name in a “well known” location.
3. An alternative file can be passed in as a command line argument.
4. Any line in the file starting with a hashtag (#) will be treated as a comment and not processed.
5. File entries are case sensitive.
6. All entries are optional. Some are required for live operation capture.
7. If the same option appears more than once, the last option will be honored, **except for hashtags**. Hashtags will be treated as a collective.
8. “White space” separates each token from its value.

Table 2: Configuration file entries. The default stop word file will be provided (source: <http://xpo6.com/list-of-english-stop-words/>). It can be modified or replaced as needed.

Token	Meaning	Default
APIPrivateKey	Twitter private API key. Must be supplied for live operation.	(None)
APIPublicKey	Twitter public API key. Must be supplied for live operations.	(None)
CollectionFile	A file to collect raw Tweets during live operations.	(None)
ColorNegative	The color used to indicate negative tweets.	BLACK
ColorNeutral	The color used to indicate neutral tweets.	WHITE
ColorPositive	The color used to indicate positive tweets.	GREEN
Hashtag	This is the hashtag used to search Twitter without the leading hashtag (#) .	(None)
LexiconFile	A text file containing positive and negative words.	lexicon.csv

(Continued on the next page.)

Table 2. (Continued from the previous page.)

Token	Meaning	Default
Offset	Should the replay data be brought forward to current time? Accepted values are TRUE or FALSE.	FALSE
Poll	How many times to add new data to the database. If data is being replayed, the maximum number of database updates will be this value, or the end of data from the file. If live operations, then this is how many times Twitter will be polled for new data.	10
PostgresTable	The Postgres table containing the tweets.	tweets
PostgresUser	The Postgres user name used to access the database.	openpg
PostgresPassword	The Postgres password associated with the Postgres user.	new_user_password
PostgresTableNASA	The Postgres table containing NASA technical report related data.	NASAREports
ResetDatabase	Should the database be reset, and all previous data lost when the program starts. Accepted values are TRUE or FALSE.	FALSE
ResetDatabaseNASA	Should the NASA technical report database be reset, and all previous data lost when the program starts. Accepted values are TRUE or FALSE.	FALSE
SleepyTime	How many seconds between updates to the database. It is possible that no data will be added to the database if there isn't any Twitter activity for a hashtag.	5
SourceFile	The file containing the data to be replayed. If this option is not set, then the operation is assumed to be "live."	(None)
StopwordsFile	The file containing "stop words" that will not be considered in determining positive or negative sentiment.	stopword.txt

(Continued on the next page.)

Table 2. (Continued from the previous page.)

Token	Meaning	Default
ThresholdNegative	The percentage of words in a tweet considered negative for the tweet to be labeled negative.	0.33
ThresholdPositive	The percentage of words in a tweet considered positive for the tweet to be labeled positive.	0.33

(Last page.)

3 References

- [1] Simon Josefsson, *RFC 4648: The Base16, Base32, and Base64 Data Encodings*, RFC 4648, RFC Editor, October 2006.
- [2] Sean Kandel, Jeffrey Heer, Catherine Plaisant, Jessie Kennedy, Frank van Ham, Nathalie Henry Riche, Chris Weaver, Bongshin Lee, Dominique Brodbeck, and Paolo Buono, *Research directions in data wrangling: Visualizations and transformations for usable and credible data*, *Information Visualization* **10** (2011), no. 4, 271–288.

A Database tables

A.1 Twitter related tables

These are the database tables/data structures to support Twitter analysis:

Table 3: Tables to support Twitter analysis.

Column	Meaning
time	Unix seconds as extracted from the Tweet.
data	Base 64 encoding of the entire JSON Tweet.

A.2 NASA related tables

These are the database tables/data structures to support NASA analysis:

Table 4: Tables to support NASA analysis.

Column	Meaning
id	NASA document ID from NTRS.
title	Base 64 encoded report title.
description	Base 64 encoded report description.
tokens	“Normalized” tokens based on raw title and description.

“Base encoding of data is used in many situations to store or transfer data in environments that, perhaps for legacy reasons, are restricted to US-ASCII data. Base encoding can also be used in new applications that do not have legacy restrictions, . . .”

S. Josefsson [1]

Base 64 encoding ensures that data can pass cleanly through PostGres operations.

B Notational data structures

B.1 Twitter structures

These are the notational data structures used by the various processes.

Table 5: Notional plotting data structure. A multidimensional structure indexed by hashtag.

Name	Purpose
PositiveTweetSource	A dictionary/hash table to keep track of the number of positive Tweets by software source. This is for the entire polling period.
NegativeTweetSource	A dictionary/hash table to keep track of the number of negative Tweets by software source. This is for the entire polling period.
PositiveTweetLocation	A dictionary/hash table to keep track of the geographic location of a positive Tweet.
NegativeTweetLocation	A dictionary/hash table to keep track of the geographic location of a negative Tweet.

Table 6: Notional plotting data structure. This structure is indexed by hashtag.

Cell	Use
0	Number of positive Tweets.
1	Number of neutral Tweets.
2	Number of negative Tweets.

B.2 Senate bill structures

These are the notional data structures associated with the Senate Bills application:

1. Each bill is stored in a separate file on the disk. These files may, or may not be deleted when the R session ends. Hence, care must be taken with how the R script is executed. If the script is executed within an IDE, files may persist for the duration of that session. If the script is run using the CLI Rscript mechanism, then the files will be deleted when the Rscript session ends.
2. Internally, all information of interest is maintained in the list “sponsors” which is organized like this:

```
sponsors[[Billnumber]][1 = bill sponsor] [2 ...n cosponsors]
```

Party affiliation is included in the sponsor/cosponsor string.

C Software on each workstation

This section contains the assumptions about the operating system environment, and software load out for each work station.

1. Operating system: Windows 7
2. Database
 - (a) Name: PostgresSQL
 - (b) Version: 9.5.3
 - (c) Source: <http://www.postgresql.org/download/windows/> and <http://www.enterprisedb.com/products-services-training/pgdownload#windows>
 - (d) Superuser password: ODUBootcamp
 - (e) Misc: It may be necessary to manually start the PostGres server using these commands in a terminal window:

```
cd "\Program Files\PostgreSQL\9.5\bin"

.\pg_ctl -D "c:\Program Files\PostgreSQL\9.5\data" start
```
3. Software
 - (a) Chrome browser
 - Version: 63.0.3239.132
 - Available from: <https://www.google.com/chrome/browser/desktop/index.html>
 - (b) Java
 - Version: Java SE Development Kit 7u79 (assuming Windows 64 bit OS)
 - Available from: <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>
 - (c) pgAdmin
 - Version: 1.22.1
 - Available from: <https://www.pgadmin.org/download/>
 - (d) R
 - Version: 3.3.1
 - Available from: <https://cran.r-project.org/bin/windows/base/>
 - Packages:

- | | | |
|-------------|----------------|---------------|
| – bitops | – mapproj | – RPostgreSQL |
| – DBI | – maps | – rvest |
| – devtools | – methods | – SnowballC |
| – ggmap | – NLP | – streamR |
| – ggplot2 | – OAIHarvester | – tm |
| – htmltools | – openssl | – tools |
| – httr | – RCurl | – utils |
| – jsonlite | – rdom | – XML |
| – mapdata | – rjson | – xml2 |
| – mapplots | – ROAuth | |

(e) R-Studio

- Version: 0.99.903
- Available from: <https://www.rstudio.com/products/rstudio/download/>

(f) SelectorGadget

- Version: 1.1
- Available from Chrome web store: <https://chrome.google.com/webstore/detail/selectorgadget/mhjnkcfbdhnjickkkdbjoemdbfginb?hl=en>



(g) wget

- Version: 1.*
- Available from: <https://eternallybored.org/misc/wget/>

The PATH environment variable should be updated to include the location of the R interpreter.

D Files

A collection of miscellaneous files mentioned in the report.

- `installLibraries.R` – an R script to install all necessary libraries/packages from “the cloud” 
- `checkPostgres.R` – an R script to test the PostGres installation (see Figure 3). 

A complete collection of files (presentations, data, scripts, etc.) can be downloaded from the boot camp web site using this command:

```
wget -np -r http://www.cs.odu.edu/~ccartled/Teaching/2018-Fall/DataWrangling/
```

The Windows version of `wget` sometimes leaves “trashy” files behind, like “`index.html@C=D;O=A`” and so on. These files are not part of the boot camp web page, and can be removed or ignored. None of the boot camp scripts use, or process these files. The *nix version of `wget` does not leave trashy files.

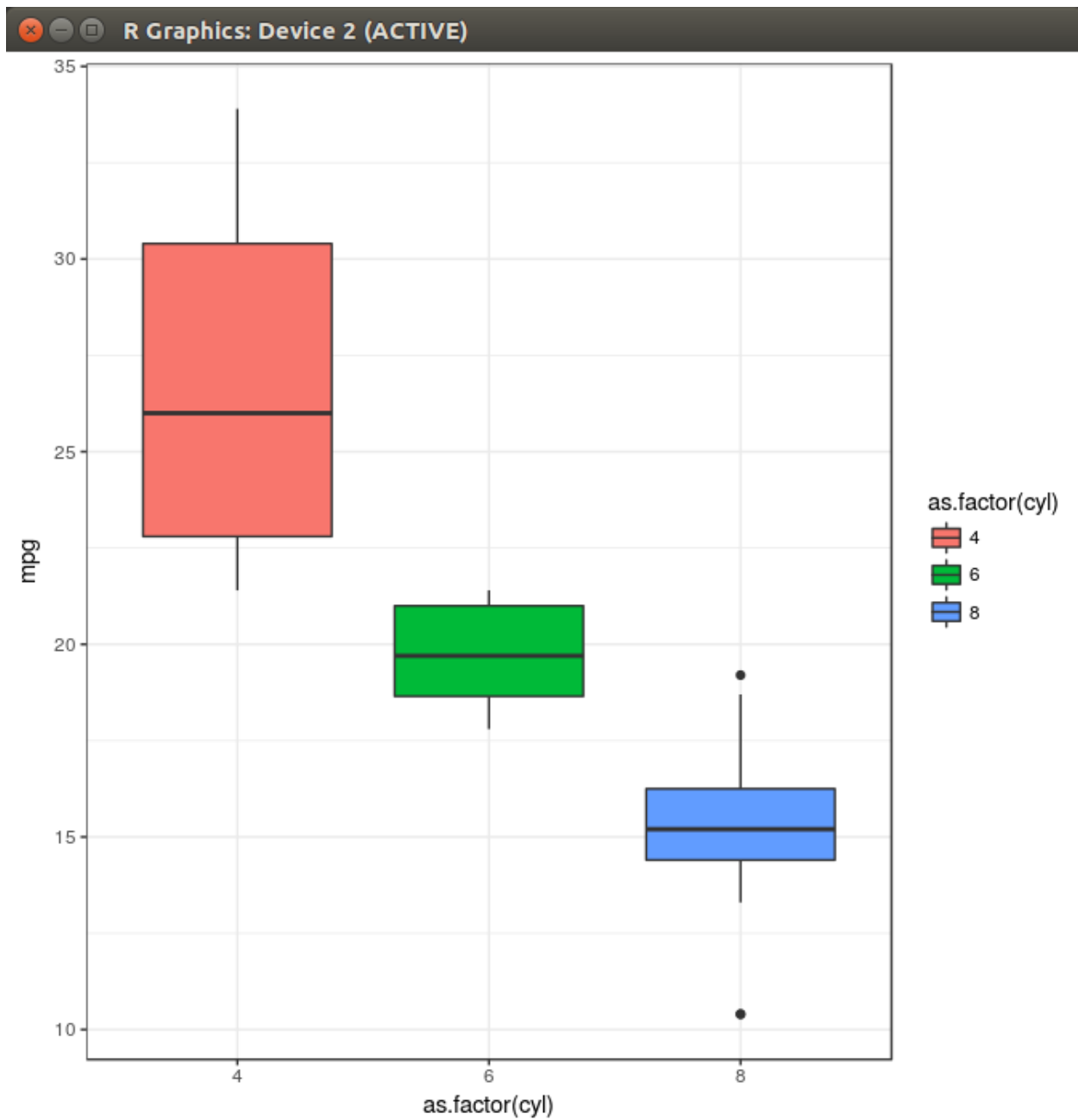


Figure 3: Image from the “checkPostgres.R” script. This image will be created (sans some of the image decorations) after successful execution of the “checkPostgres.R” script. The decorations will change based on how the script was executed.