# A Tutorial for Using Twitter Data in the Social Sciences: Data Collection, Preparation, and Analysis

**Book** · January 2016

CITATIONS
6

READS
3,233

**2 authors**, including:

Andreas Jungherr
Universität Konstanz
**91** PUBLICATIONS **1,469** CITATIONS

SEE PROFILE

PASCAL JÜRGENS AND ANDREAS JUNGHERR

# A TUTORIAL FOR USING TWITTER DATA IN THE SOCIAL SCIENCES: DATA COLLECTION, PREPARATION, AND ANALYSIS

The manuscript was set using the Tufte LaTeX package available at `https://github.com/Tufte-LaTeX/tufte-latex` developed by Kevin Godby, Bil Kleb, and Bill Wood.

*V. 0.1, January 2016*

# *Abstract*

The ever increasing use of digital tools and services has led to the emergence of new data sources for social scientists, data wittingly or unwittingly produced by users while interacting with digital tools. The potential of these digital trace data is well-established. Still, in practice, the process of data collection, preparation and storage, and subsequent analysis can provide challenges. With this tutorial, we provide a guide for social scientists to the collection, preparation, and analysis of digital trace data collected on the microblogging service Twitter. This tutorial comes with a set of scripts providing researchers with a starter kit of code allowing them to search, collect, and prepare Twitter data following their specific research interests. We will start with a general discussion of the research process with Twitter data. Following this, we will introduce a set of scripts for data collection on Twitter. After this, we will introduce various scripts for the preparation of data for analysis. We then present a series of examples for typical analyses that could be run with Twitter data. Here, we focus on counts, time series, and networks. We close this tutorial with a discussion of challenges in establishing digital trace data as a normal data source in the social sciences.

# *Contents*

# *List of Figures*

# *List of Tables*

# *Using Twitter Data in the Social Sciences*

The ever increasing use of digital tools and services has led to the emergence of a new data source for social scientists. Increasingly multifaceted uses of digital services result in *digital trace data* (Freelon, 2014; Golder and Macy, 2012; Howison, Wiggins, and Crowston, 2011; Jungherr, 2015), data wittingly or unwittingly produced by users. Examples for the former are texts of tweets, Facebook posts, or pictures posted on a photo-sharing site. Examples for the latter are metadata of online interactions such as the location a message was posted at or the device it was posted with. These data promise a closer look at aspects of human behavior accompanied by the use of digital tools.

This promise has led researchers to propose various approaches as to how this new data source might be incorporated in social science research, such as *computational social science* (Cioffi-Revilla, 2010; Cioffi-Revilla, 2014; Conte et al., 2012; Gilbert, 2010; Lazer et al., 2009; Strohmaier and Wagner, 2014; Vespignani, 2012), *digital methods* (Rogers, 2013b), or *big data* (boyd and Crawford, 2012; González-Bailón, 2013; Lazer et al., 2014; Mahrt and Scharkow, 2013; Mayer-Schönberger and Cukier, 2013). Also, various key-journals in the social sciences dedicated special issues or sections to discussing this potential, such as *The ANNALS of the American Academy of Political and Social Science* (Shah, Cappella, and Neuman, 2015), *Journal of Communication* (Parks, 2014), *PS: Political Science & Politics* (Clark and Golder, 2015), or the *Social Science Computer Review* (Zúñiga, 2015).

But while the potential of this new data source is well-established, in practice, the process of data collection, preparation and storage, and subsequent analysis can provide challenges. Especially, as most social scientists, even those familiar with quantitative methods, often lack familiarity with methods and tools necessary to work through all necessary steps of the research process with digital trace data. One potential solution to this challenge is a lab-based approach with an interdisciplinary team of researchers collaborating on projects (King, 2011). In these teams, social scientists could, for example, be respon-

sible for the development of research questions and theory-driven operationalizations, computer scientists could focus on data collection and storage, while physicist or applied mathematicians could approach data analysis with advanced quantitative methods. While sounding promising on paper, actually establishing interdisciplinary teams like this is fraught with challenges. Unsurprisingly, only few successful examples for constellations like this exist. Yet, even when working in one of these, as of yet, largely imaginary all-star teams, social scientists interested in using digital trace data should be able to perform basic tasks in the collection, preparation, and analysis of digital trace data (Freelon, 2015).

While, in principle, digital trace data come in many shapes or forms, in practice, most researchers using them focus on data collected on the microblogging service Twitter. Most likely, this is due to the relative ease of access to Twitter data for researchers. Researchers using Twitter data should thus always critically discuss how their focus on data collected on this platform is driven by more than simply the relative convenience of collecting data on Twitter. Yet, even while some of Twitter's prominence as a research object might be driven by a scientific availability bias, it offers a promising research environment for social scientists (Jungherr, 2015; Jungherr, Schoen, and Jürgens, 2015; Rogers, 2013a).. Especially if we compare the level of access and detail Twitter provides to its data with the access and detail of data available on other digital services such as Google or Facebook, data collected on Twitter might be the closest researchers unaffiliated with cooperations can get to analyzing and understanding characteristics, measurement logic, and discovery potential of digital trace data in general.

With this tutorial, we provide a guide for social scientists to the collection, preparation, and analysis of digital trace data collected on the microblogging service Twitter. This tutorial comes with a set of scripts providing researcher with a starter kit of code, allowing them to search, collect, and prepare Twitter data following their specific research interests. We will start with a general discussion of the research process with Twitter data. Following this, we will introduce a set of scripts for data collection on Twitter. After this, we will introduce various scripts for the preparation of data for analysis. We then present a series of examples for typical analyses that could be run with Twitter data. Here, we focus on counts, time series, and networks. We close with a discussion of challenges in establishing digital trace data as a normal data source in the social sciences.

# *Establishing a Research Process for Digital Trace Data*

## *Using Digital Trace Data in the Social Sciences*

While the potential of digital trace data for the social sciences is well established and often discussed (Cioffi-Revilla, 2010; González-Bailón, 2013; Jungherr, 2015; Lazer et al., 2009), practical aspects of realizing this potential are often neglected. The integration of digital trace data in the social sciences faces a series of non-trivial challenges, such as stably linking digital trace data to concepts of interest for social scientists (Freelon, 2014; Golder and Macy, 2014; Howison, Wiggins, and Crowston, 2011; Jungherr, 2015; Lazer et al., 2014), concept validation (Freelon, 2014; González-Bailón and Paltoglou, 2015; Howison, Wiggins, and Crowston, 2011), data quality (Morstatter et al., 2013; Morstatter, Pfeffer, and Liu, 2014; Ruths and Pfeffer, 2014), or users' privacy (boyd and Crawford, 2012; King, 2011; Puschmann and Burgess, 2013). It is vital for researchers to address these questions appropriately in the design and the interpretation of their analyses. Yet, for the purposes of this tutorial, we will ignore these questions and focus only on practical aspects of establishing a research process starting with the collection of data on a digital service—in our case the microblogging service Twitter—the preparation of the collected data for analysis, and three basic analytical approaches—counts, time series, and networks. While this is only a small selection of potential analytical approaches using Twitter data and we do not adequately discuss theoretical and conceptual issues of Twitter-based research, this tutorial should provide a good basis to get you started. Yet, keep in mind that there is more to Twitter-based research than that what we address in these pages.

It has been argued that the use of digital trace data in research is best done in interdisciplinary teams. Working with these data requires expertise in large-scale continuous data collection, the storage of large data sets, the preparation of these data sets for analysis, various methods of quantitative data analysis, and the appropriate theoretical contexts. This skill set is unlikely to be found within individual researchers. Instead, this combination of skills is most likely to

be found in a formal or informal lab setting with various researchers contributing their specific expertise (King, 2011). Most likely, in these teams the role of social scientists will lie in the development of research questions, the development of operationalizations, and the contextualization of research results. Still, it would be a mistake for social scientists to avoid coding altogether (Freelon, 2014). In order to successfully work in an interdisciplinary research environment, social scientists interested in the use of digital trace data have to become code literate. Without a basic understanding of the research process with digital trace data, they will remain uncomfortably dependent on more technically minded members of their research team. More importantly, they will remain ignorant of the research process, design choices, and algorithms used by their colleagues and, therefore, even run the risk of misinterpreting results presented to them.

With this tutorial, we aim to provide social scientists interested in the use of digital trace data with an accessible guide for working with data collected on Twitter. In this, our tutorial follows the lead of a series of other tutorials on the collection of Twitter data. While these tutorial are excellent, our tutorial builds on those already available tutorials in three important aspects:

First, our tutorial focuses on the collection of data on Twitter through code. You can find a series of tools that offer out-of-the-box solutions for researchers interested in collecting data on Twitter. Three of the most popular tools are NodeXL[1] (Hansen, Shneiderman, and Smith, 2010), YourTwapperKeeper[2] (Bruns and Liang, 2012), and the *The Digital Methods Initiative Twitter Capture and Analysis Toolset* (DMI-TCAT)[3] (Borra and Rieder, 2014). Further tools can be found on a list curated by Deen Freelon[4]. Still, we believe a code-based approach is more flexible and offers researchers a more direct understanding of the data underlying their analyses.

[1] http://nodexl.codeplex.com

[2] https://github.com/540co/yourtwapperkeeper

[3] https://github.com/digitalmethodsinitiative/dmi-tcat

[4] http://socialmediadata.wikidot.com/

Alternatively to collecting data through Twitter's API yourself, there are options to buy data from data vendors licensed by Twitter. This might be your optimal choice if you are looking for historical data or want to make sure you are actually covering all messages identified by your selectors. For this, you could turn, for example, to Gnip[5] or DiscoverText[6].

[5] http://gnip.com

[6] http://discovertext.com

Second, we chose to use dedicated software for different aspects of the research process. We decided on the use of python for data collection and rudimentary data preparation, SQLite for data storage, and R for data analysis. Choosing three different tools leads to somewhat higher setup costs for you than the choice of only one tool for all tasks. Still, we believe the task-based choice of tools pays off in greater flexibility when performing specific tasks. Alternatives to our approach are readily available, such as the use of R not only for anal-

ysis but also for data collection (Barberá, 2014; Munzert et al., 2015), or the use of Java and JavaScript for data collection and MongoDB for data storage (Kumar, Morstatter, and Liu, 2014).

Third, we chart the research process with digital trace data through its essential steps: Data collection, data storage and preparation, and analysis. With a few exceptions (Kumar, Morstatter, and Liu, 2014), most tutorials focus on specific steps in this process. This leads to a wealth of tutorials documenting data collection through Twitter's API (Makice, 2009; Russell, 2014) or in the analysis of resulting data (McKinney, 2013; Segaran, 2007). The essential steps of data storage and data preparation for analysis are largely missing from these accounts. With this tutorial, we wish to bridge this gap by illustrating the complete research process with digital trace data using a specific research project as example.

For this example project, we will focus on Twitter messages commenting on politics during the fourth televised debate in the Republican primaries for the US presidential election 2016. For each step of the research process, we provide detailed example scripts allowing readers to perform the tasks described by us on their own machines. In our own research, we focus on Twitter use in political communication, during election campaigns (Jungherr, 2013; Jungherr, 2014; Jungherr, 2015; Jungherr, Schoen, and Jürgens, 2015; Jürgens and Jungherr, 2015; Jürgens, Jungherr, and Schoen, 2011) and in political participation (Jungherr and Jürgens, 2014a; Jungherr and Jürgens, 2014b). Thus the examples presented her are based on these interests. Still, it should be trivial for readers to adapt the example scripts provided by us to the requirements of their specific areas of interest.

We posted example scripts and functions in a GitHub repository dedicated to the tutorial[7]. The examples come with detailed documentations, explaining the use of the functions and example code. While we tested code and functions used in the tutorial, it is highly likely that we overlooked mistakes. Please open issues in our GitHub repository if you find mistakes or if you have advice on potential improvements to the tutorial. Feedback is always welcome.

[7] https://github.com/trifle/twitterresearch

## *Linking Data Access to Theoretical Concepts*

Social scientists routinely deal with phenomena that are hard to observe. They are familiar with numerous obstacles to scientific discovery and have devised methods to counteract them. For example, latent variables may be measured through standardized scales or biased access to a sample can be overcome through clever measurement and weighting. Computer-mediated communication adds another layer of opacity and distortion to the research process. In

addition to the careful elaboration required in empirical work, researchers dealing with digital trace data need to take into account that their data may be tainted by entirely new biases and limitations. Computer systems are designed to facilitate individual communication, not to enable representative observations. This has to be taken into account by researchers thinking about using data provided by digital services in their projects. Design choices, therefore, have to be consciously weighted and documented. Here, two types of issues arise: (A) those regarding sampling/representativity (Diaz et al., 2014; Lazer et al., 2014; Ruths and Pfeffer, 2014) and (B) those regarding the mediation of user behavior through platforms/online services (Jungherr, 2015; Jungherr, Schoen, and Jürgens, 2015). Both impact our ability to perform rigorous social science, since they weaken the bridge between theory and empirical measurements.

First and foremost, sampling remains a problematic task on the internet. On the one hand, there is the open internet that is composed of the myriads of web pages. We only have rough guesses with regards to their number, as there is no central index of all sites. Sampling relies on a well-defined population with a known distributions—something we can only claim about small portions of the web (Huberman, 2001). With the advent of closed platforms—such as Facebook, Twitter and others—a promising new opportunity emerged. The monolithic, centralized architectures of digital services create a well-defined population, in theory, enabling statistically sound samples. However, the corporations that run the platforms protect their growth by hedging network effects through lock-in. It is, therefore, not in their best interest to provide broad access to their content. Instead, they usually provide a programming interface (API) that comes with restrictions to its use. As long as researchers stay within the legal bounds of the platform's terms of service (TOS), the collection of digital trace data is limited to what is accessible through the official interface. Through their closed nature, online platforms enforce the priority of the API. Researchers must (1) consider what data are available, (2) construct a theoretically meaningful and useful sample that can be built from elements available in the API, (3) proceed to match the desired sample with the appropriate API methods, and (4) finally collect the data and verify its quality. In practice, these steps require both persistence and an intricate knowledge of the API. For example, it is absolutely crucial to know that Twitter's search function by default filters (that is, omits!) "irrelevant" content[8]. If one were to assume that search provides a comprehensive picture of user activity surrounding one keyword, the resulting research would be severely biased through Twitter's algorithm. This is the central reason why we so strongly recommend interdisciplinary teams and code

[8] https://dev.twitter.com/rest/public/search

literacy for social scientists.

In the same way that the scientists' observations of online behavior is filtered through a service's API, users can only express themselves within the limits of a platform's design. Whatever their original intentions might be, users need to translate them into actions that fit the pre-defined channels, modes, and interaction patterns defined by the platform. For example, as long as Facebook only allows positive feedback in the form of a *like*, negative feedback can only be provided as a text comment. This is the second layer of bias added by computer-mediated communication: Whatever digital traces we observe, any conclusion about their meaning needs to take into account the mediation through the platform's rules and algorithms (Gillespie, 2014; Rieder, 2004).

To illustrate this problem, imagine a study design that focuses on opinion leaders on Twitter. Traditionally, opinion leader research has relied on surveys and used individuals' psychological attributes along with self-reported measures of interpersonal communication. While surveying Twitter users is fraught with difficulties, digital trace data can serve as a readily available replacement measurement. Their theoretical usefulness is limited however, since the API does not provide any insight into the tweets that users read. We can only measure interaction in the form of retweets, directed messages, and mentions. The opinion leaders that we find are, therefore, not necessarily advice givers in the sense of the literature, but rather famous people that provoke information diffusion (retweets) or feedback (directed messages and mentions). Even if a study following this design drew from a rich literature with clearly defined concepts, it would end up measuring phenomena that differ more or less subtly from the original definitions, making for an empirically and theoretically challenging design. This raises the importance of conceptual work, critically linking established theories to newly available metrics based on digital trace data (Howison, Wiggins, and Crowston, 2011; Jungherr, 2015; Jungherr, Schoen, and Jürgens, 2015).

In our view, these challenges inherent in collecting and analyzing digital trace data emphasize the importance for social scientists to develop code literacy. Only by being able to write and read code, will you be able to directly interact with a service's API and thus be able to make and assess design choices while keeping in mind their consequences for the interpretation of patterns emerging subsequent analyses.

*Further Reading:*

James Howison and colleagues provide an overview of common issues in the use of digital trace data (Howison, Wiggins, and Crowston, 2011). Deen Freelon provides a valuable overview of the research process involved in using digital trace data in the social sciences (Freelon, 2015). Claudio Cioffi-Revilla provides a comprehensive introduction to *Computational Social Science* from a computer science perspective (Cioffi-Revilla, 2014). Also, make sure to read Lazer et al. (2014), a piece in which the authors offer a much needed critically discussion of challenges in using *big data* for research.

# *Before We Proceed*

## *What Does this Tutorial Offer, What Doesn't it Offer?*

Before we begin, let's start with a little expectation management. With this tutorial and the accompanying set of scripts, we aim to provide you with an easy to follow path through the research process of collecting, preparing, and analyzing Twitter data. What we will not, what we can not offer, is an introductory course to the underlying tools or analytical techniques. To be sure, our scripts should offer any novice the possibility to collect and analyze data on Twitter. But to go beyond the examples provided by us and to build on the scripts provided here, we recommend you spend some time learning the basics of python and R.

Luckily, there are many sources you can turn to for introductions to these tools. If you have not gained any familiarity with python try the website *code academy*[9]. The site offers a free introductory course to python. Another easy to follow general introduction to programming in python is *Learn Python the Hard Way*[10] (Shaw, 2014). If you are firm on the basics, move on to Wes McKinney's *Python for Data Analysis*. This book offers great advice on the use of python for data analysis (McKinney, 2013).

In general, we believe python to be an excellent choice for the collection and initial preparation of data, for subsequent analyses we recommend the use of the statistical programming language R. Robert I. Kabacoff offers an excellent overview of various uses of R (Kabacoff, 2015). For a more thorough approach, have a look at Norman Matloff's *The Art of R Programming: A Tour of Statistical Software Design* (Matloff, 2011). Of course, there are other resources you can turn to.

[9] https://www.codecademy.com/learn/python

[10] http://learnpythonthehardway.org

## *What Software Do You Need?*

In order to follow the provided examples, you will need a series of basic tools:

- A good text editor that can work with raw, unformatted text files: On Microsoft Windows, notepad++[11] is a popular choice, OS X users can use the free tool TextWrangler[12] or try one of the many commercial options (TextMate[13], BBEdit[14]). Atom[15] is a new but rapidly evolving free programming editor built by GitHub. Sublime Text[16] is a powerful and popular cross-platform editor (Windows, Mac, or Linux).
- A working copy of the programming language python[17]: There are two versions available that are both used widely: Version 2 (which will stay at version 2.7) and version 3 (which is currently at version number 3.5 and which is the basis for future developments of python). We will use version 3 since it has a more modern and consistent syntax. However, some tools and documentation are only available for the older variant.
- A way to install python libraries (packages sometimes called eggs or wheels): Some managed installations of Windows (such as those found at large universities) do not permit users to install their own software. In such cases, it might be easier to use your own machine. There are two commercial python distributions that offer free all-in-one packages which might be easier to install: Continuum Analytics' Anaconda[18] and Enthought Canopy[19].
- If you are using a Mac, you also have to install Apple's Xcode[20] program for python to work. This is easily done and provides you with a host of tools for software development on the Mac.
- We also recommend that you install the statistical programming language R (R Core Team, 2015). While, in theory, you could run your analyses using python, R is a much more flexible environment for data analysis than python. You can install R by simply following the information given on the website of the R-Project[21]. We also recommend that you use RStudio[22], a very helpful user interface for R. While, strictly speaking, RStudio is optional for the purposes of this tutorial, the free desktop version will definitely make your interactions with R easier.

[11] https://notepad-plus-plus.org
[12] http://www.barebones.com/products/textwrangler/
[13] https://macromates.com
[14] http://www.barebones.com/products/bbedit/
[15] https://atom.io
[16] http://www.sublimetext.com
[17] https://www.python.org
[18] https://www.continuum.io/why-anaconda
[19] https://www.enthought.com/products/canopy/
[20] https://developer.apple.com/xcode/
[21] https://www.r-project.org
[22] https://www.rstudio.com

Strictly speaking, everything beyond these elementary tools is optional.

## *Does Everything Work?*

Once you have everything set up, you can check if your installation of python is working by typing `python -V` in a terminal (on Windows: `start cmd.exe`, on OS X start `Terminal.app`) and pressing enter. You should see output similar to:

```
python -V
>>>Python 3.5.0
```

In addition to the programming language itself, we will use several readymade collections of code called libraries. The preferred way of installing them is using a little tool called pip. Sometimes it is already included with python (on linux, for example). Try typing `pip -V` at the command line:

```
pip -V
>>>pip 7.1.2 from /usr/local/lib/python3.5/site-packages (python 3.5)
```

If that command gives an error, it might be possible to install pip with the command `easy_install pip`. In order to verify that you can install libraries, try installing the library `ipython`:

```
pip install ipython
```

Once everything works, you are ready to use the code from this tutorial and to start creating your own programs.

### *Script Examples*

For this tutorial, we developed a series of scripts in python and R covering standard tasks in the collection, preparation, and analysis of digital trace data collected on Twitter. These scripts are available online in the GitHub repository `twitterresearch` dedicated to this tutorial[23]. You have a number of possibilities for using these scripts. First, you could simply download their most current version directly from the repository. Alternatively, you could create a GitHub account yourself[24] and clone the repository on your machine—for example, by using the program GitHub Desktop[25]. The benefit of this approach is that you can very easily update the scripts on your machine to their most current version. But remember to backup the changes you made to the scripts on your machine so you do not accidentally overwrite these changes by updating.

We will aim to keep these scripts current and provide updates. Still, it is possible that some of the scripts have become somewhat obsolete by the time you work through this tutorial. The scripts are based on the current implementation of Twitter's API and current versions of various software packages used in this tutorial. Future updates to Twitter's API or these software packages might render the

[23] https://github.com/trifle/twitterresearch

[24] https://help.github.com/articles/signing-up-for-a-new-github-account/

[25] https://desktop.github.com

code examples provided by us obsolete. Please let us know if you run into trouble by opening an issue in the GitHub repository.

We documented these scripts in the respective GitHub repository[26]. If you are familiar with the research process with Twitter data, you, therefore, could skip this tutorial and simply use our scripts and turn to the documentation online. For everyone else, here, we offer detailed examples for the required steps in Twitter-based research. In these examples, we show how and when our scripts might be used in research. This should allow you to adapt them to your own research interests and projects.

[26] https://github.com/trifle/twitterresearch

We published these scripts under a GPL V3 license. This means, you are free to use them in your work, modify them, and incorporate them in work building on them—provided this happens in a non-commercial context[27].

[27] http://www.gnu.org/licenses/gpl-3.0.en.html

## *Preparing Your Workspace*

Start the commando line of your system. Now, navigate to the directory you saved our example scripts in. You do this by using the command `cd` followed by the path of the respective directory. For an easy to follow tutorial on using the command line see Bradnam and Korf (2012).

Our scripts need a series of python modules to run. We list these in the file `requirements.txt`. To make sure, you have all required modules run the following command:

```
pip install -r requirements.txt
```

Now, you should be almost ready to go. You can always run your Twitter data collection from your command line. Alternatively, you could use IPython[28] to access python. We recommend this, as IPython offers a lot of functionality making your interactions with python smooth and painless. If you decide to install IPython run the following command:

[28] http://ipython.org

```
pip install -U ipython
```

Excellent, now you are truly ready to go!

## *Data Used in this Tutorial*

For the examples presented in our tutorial, we decided to focus on Twitter messages commenting on politics during the fourth televised

debate in the Republican primaries for the US presidential election 2016 on October 28, 2015[29]. We collected messages posted between October 27, 0:00 a.m. MST and November 3, 2015, 0:00 a.m. Mountain Standard Time (MST). The debate was transmitted from Boulder, Colorado. We, therefore, use the local timezone, MST, as reference timezone for our analysis.

We used the following selectors for identifying relevant messages posted between October 27 and November 3, 2015. First, we collected all messages posted by candidates standing in the Republican and Democratic primaries. We also collected tweets posted by the official account of the television station organizing the debate, CNBNC, and the sitting US-President, Barack Obama[30]. Second, we collected mentions and retweets of these accounts in messages posted by other users. Third, we collected mentions of these candidates, the television station, and the debate in hashtags or in keywords[31]. This resulted in a data set consisting of 805.630 messages. For another look at Twitter messages posted during the televised debate see Guess, Nagler, and Tucker (2015).

One of the big issues in working with digital trace data is the reproducibility of studies (King, 2011; Stodden, Leisch, and Peng, 2014). The central challenge being directly reproducing analyses presented by authors by using their code for data collection, preparation, and analysis. In this, the reproduction of studies is closely related to the replication of studies, which focuses on the replication of findings in similar but not necessarily the same contexts than the ones used in the original study (Peng, Dominici, and Zeger, 2006).

The first step in allowing a Twitter-based study to be reproduced is the comprehensive publication and annotation of code used in the analysis. This goes a long way. But even then, it might not be possible to exactly reproduce a study. The weak link in the reproduction of Twitter-based studies is the reproduction of the underlying data. Twitter does not allow the publication of datasets documenting full information collected through Twitter's API. Instead, Twitter allows for the publication of lists documenting the IDs of messages included in an analysis. These IDs can then be used to collect the respective messages by researchers trying to reproduce a specific study[32]. But even publishing a list of IDs of all tweets included in an analysis does not guarantee the exact reproduction of studies. The reason for this is that users might delete selected tweets or their complete accounts in the time between the original data collection and any reproduction attempt. Any tweets deleted between these dates will not be available for the reproduction of the study.

For this tutorial, we attempted to provide you with a dataset coming as close as possible to be reproducible. To this end, we extracted

[29] https://en.wikipedia.org/wiki/Republican_Party_presidential_debates,_2016

[30] We collected the messages posted by the following accounts: *Candidates, Republican*: JebBush, RealBenCarson, ChrisChristie, tedcruz, CarlyFiorina, GovMikeHuckabee, gov_gilmore, LindseyGrahamSC, bobbyjindal, johnkasich, governorpataki, RandPaul, governorperry, marcorubio, RickSantorum, realdonaldtrump, ScottWalker; *Candidates, Democrats*: lincolnchafee, HillaryClinton, MartinOMalley, BernieSanders, JimWebbUSA; *Media*: cnbc; *Sitting President*: BarackObama.

[31] We collected messages containing the following character strings either in hashtags or in keywords irrespective of capitalization: *Debate*: gopdebate, CNBCGOPDebate, CNBCDebate; *Candidates, Republican*: JebBush, RealBenCarson, ChrisChristie, tedcruz, CarlyFiorina, GovMikeHuckabee, gov_gilmore, LindseyGrahamSC, bobbyjindal, johnkasich, governorpataki, RandPaul, governorperry, marcorubio, RickSantorum, realdonaldtrump, ScottWalker; *Candidates, Democrats*: lincolnchafee, HillaryClinton, MartinOMalley, BernieSanders, JimWebbUSA; *Media*: cnbc; *Sitting President*: BarackObama, obama.

[32] https://dev.twitter.com/overview/terms/agreement-and-policy

all IDs of tweets identified in our original data collection. On November 4, 2015, we attempted to re-download these tweets through Twitter's API using this ID list. As expected not all tweets originally identified by us were available. About two percent of our originally identified messages had been deleted by their authors between our original data collection and our attempt at reproduction. This left us with 788.229 messages including retweeted tweets. These messages provide the basis for the analyses presented later in this tutorial. We published these IDs in our GitHub repository to allow you the direct reproduction of the results presented in this tutorial. Still, there is no guarantee that between the time of this writing and your reproduction attempt further tweets will not be deleted by their authors. Your results might thus still deviate somewhat from the results presented here.

### *Further Reading:*

The website *code academy*[33] offers a free introductory course to the programming language *python*[34]. Another general introduction to programming in python is *Learn Python the Hard Way*[35] (Shaw, 2014). Wes McKinney's *Python for Data Analysis* offers great advice on the use of python for data analysis (McKinney, 2013). Matthew A. Russell discusses the use of python in mining data from various social web services (Russell, 2014). Robert I. Kabacoff offers an excellent overview of various uses of the statistical programming language *R*[36] (Kabacoff, 2015). Norman Matloff provides more systematic introduction to programming in R (Matloff, 2011). For studies analyzing Twitter activity during political media events on television see for example Freelon and Karpf (2015), Jungherr (2014), Lin et al. (2014), Trilling (2015), and Vaccari, Chadwick, and O'Loughlin (2015).

[33] https://www.codecademy.com/learn/python

[34] https://www.python.org

[35] http://learnpythonthehardway.org

[36] https://www.r-project.org

# *Data Collection*

The first step in any Twitter-based research project in the social sciences should be the development of a theory-driven research question. This aspect, although of crucial importance, is not the focus of this tutorial. For a brief review of varying interests and approaches of studies addressing the use of Twitter during election campaigns—the topical frame of the examples presented below—see Jungherr (2016). The second step is the collection of data of interest. In this section, we offer various examples for data collection on Twitter. We start by briefly discussing Twitter's application programming interfaces (APIs) and by presenting our scripts for accessing them. We then discuss hashtag/keyword-based searches and account-based searches. These two approaches cover most data collection logics on Twitter.

## *Twitter APIs*

Twitter offers two types of application programming interfaces—API for short. The REST-APIs[37] allow developers access to read and write Twitter data. For researchers, these APIs are a valuable access-point to search for messages posted in the recent past, conforming with criteria—such as the use of specific keywords, hashtags, or user names. The Streaming APIs[38] allow developers access to Twitter's global stream of data. These APIs are valuable for researchers as they allow the capturing of messages corresponding with specific criteria in real time. In order to access these APIs, researchers have to create a Twitter application handling the requests to Twitter's database.

[37] https://dev.twitter.com/rest/public

[38] https://dev.twitter.com/streaming/overview

### *Authentication*

To get the necessary authentication information to access data through Twitter's API, you have to create a Twitter application. Twitter has made this process quick and painless. First, you have to have an active Twitter account. Once you have an account, visit Twitter's application registration page[39] and follow the steps listed there. If you run into trouble, a series of helpful tutorials walk you through the

[39] https://apps.twitter.com

process in greater detail—see for example Ojeda et al. (2014) and Russell (2014).

After you have created your application, you have generate four tokens allowing your scripts to collect data on Twitter. These are: *API-key*, *API-secret*, *Acces token*, and the *Access token secret*. These keys are vital and you should keep them as protected as your email passwords or ATM-PINs. Any interaction with Twitter's databases using these credentials can be tracked back to you.

Of the scripts provided in our tutorial the script `twitter_auth.py`[40] handles the authentication process between your data collection effort and Twitter's various APIs. For `twitter_auth.py` to work as intended, you have to store the credentials provided to you by Twitter in your local copy of the script `keys.yaml.template`[41]. Once you have done this, you are ready to go.

[40] https://github.com/trifle/twitterresearch/blob/master/twitter_auth.py

[41] https://github.com/trifle/twitterresearch/blob/master/keys.yaml.template

*REST APIs:*

As stated above, Twitter offers two distinct modes of data access: Real-time streams are collected through Streaming APIs while REST APIs[42] offer structured, albeit limited, access to Twitter's archives. There are two central characteristics that shape our strategies when interacting with Twitter:

[42] https://dev.twitter.com/rest/public

- A REST API uses single requests in order to retrieve (`GET`), publish (`POST/PUT`) or delete (`DELETE`) resources. Just like all http transfers, it is stateless—meaning that each transaction is isolated from others and stands for itself. With each request, we need to fully specify what we want to do. It is also our own duty to handle errors by retrying failed requests.
- Second, just like any other major platform on the web, Twitter needs to protect its resources against misuse. Twitter's APIs enforce a so-called *rate limit*, a maximum number of requests per time interval that users are allowed to perform. This rate limit has to be taken into account in running automated data collection through Twitter's APIs.

This shifts the burden of valid and reliable data collection to the researcher. We need to meticulously specify what data we are looking for, track which parts have been downloaded at any given time, and deal with numerous potential errors.

Our script `rest.py`[43] handles one-off and repeated calls to Twitter's REST APIs. It thus serves as the basis for our scripts querying Twitter's REST APIs.

[43] https://github.com/trifle/twitterresearch/blob/master/rest.py

With this script, we first and foremost tried to provide a clean and legible but solid blueprint. In contrast to most example scripts, it

handles rate limits and errors gracefully. While we optimized our example script with regard to legibility, this comes at the cost of some inefficiency. We purposefully omitted some optimizations which would greatly increase the codebase and decrease its legibility. Here are two hints for potential improvements:

- *Global rate limit:* Currently, there is a global rate limit counter; we pause our access as soon as Twitter tells us we have only four requests left. By doing this, we ignore the distinct (independent!) rate limits for different types of requests. For example, if you exhausted your rate limit for crawling an user's tweet archive, you can still perform requests for her friend list. If you implement a separate rate limit counter for every possible API endpoint, you can parallelize fetching multiple types of data.
- *Only one user, one thread:* All modules in this repository rely on the same user credentials defined in the single keyfile. It is possible to have several copies of the code fetch different sets of data with two or more different Twitter accounts.

*Streaming APIs:*

Streaming APIs[44] deliver a continuous stream of incoming tweets—either from a random sample or matching given criteria—and thus follow different paradigms than REST APIs. Instead of performing multiple independent requests, a stream is connected once. The connection stays alive as long as both sides keep it open and collects tweets in real-time as they are published. Under the hood, streams are still http connections, which means they have response headers, are initiated via `GET` or `POST` requests, and are terminated with a status code. There are some issues to watch out for:

[44] https://dev.twitter.com/streaming/overview

- Stream connections can exist for a very long time. But this does not necessarily mean that they are continuously transmitting data. For example, a stream filtering for rarely used words can go minutes or even hours without yielding a tweet. To keep the connection between server and client alive for long periods without necessarily transferring messages, Twitter periodically sends a *keepalive* signal.
- Just like REST requests, streams are rate-limited. Twitter restricts the amount of tweets delivered via the random sample stream and via tracking streams. For validity issues connected with Twitter's sample stream see Morstatter et al. (2013) and Morstatter, Pfeffer, and Liu (2014). A complete stream containing all tweets and a tracking stream that guarantees to deliver all matching tweets

are only available commercially via Twitter's data broker Gnip[45]. Take note that all free sample streams are identical: Connecting multiple times is prohibited and yields no additional data.

[45] https://gnip.com

- Even though streams are designed to stay open, there are numerous reasons why they might be closed. It is helpful to distinguish between reasons for stream termination: (1) Failures to connect result in an http error status code. For example, if you try to connect with invalid tokens, the connection ends with the status code `401`. (2) In contrast, failures during streaming try to send a failure explanation before disconnecting. The most common causes for disconnects are connection issues on the user's or on Twitter's side—such as a restart of the server delivering the stream.
- Twitter provides very helpful metadata while the stream is running. In between tweets, you will see lines containing metadata—such as keepalive data, delete notices broadcasting the IDs of deleted tweets, and warnings if your application is not fast enough to keep up with the stream. From the researcher's point of view, the most important metadata are limit notices. Lines such as this:

```
{'limit': {'timestamp_ms': '1443095140794', 'track': 16}}
```

  tell your application that as of the given timestamp your tracking stream has omitted 16 matching tweets. Short of buying access, it is not possible to know what exactly was omitted. Still, tracking limit notices allows researchers to roughly quantify the coverage of their tracking data.

Our script `streaming.py`[46] handles requests to Twitter's Streaming API. As before, in the script we put a premium on legibility and the robust handling of access limits, somewhat to the detriment of efficiency.

[46] https://github.com/trifle/twitterresearch/blob/master/streaming.py

Any application that connects to the Streaming API needs to somehow interweave connection handling and the actual processing of tweets. There are many ways to achieve this: It would be possible, for example, to set up two separate programs—potentially running on different machines—and pass the stream between them. This would be a modular and performant design, but would also introduce complexity and additional opportunities for failure. Instead, we opted to use a simple unified design.

In our script, you will find two functions for setting up either a random sample or a tracking stream. Each of these functions takes two functions as optional arguments: One function for handling tweets and one for handling non-tweet information. This design is

called *callback*—our stream handling code hands a tweet over to a processing function which then hands control back to the stream handler.

It is absolutely vital that the code processing tweets does so quickly. While the callback functions do their work, the main function is paused and cannot process incoming data. If your code takes too long, Twitter will disconnect the stream and—if that happens too often—disable your API access altogether. So make sure you only perform the absolute minimum of processing work while streaming. Given this, post-processing is best handled after data collection.

### *Hashtag/Keyword-Based Searches*

One common approach to data collection on Twitter is the collection of tweets using topically relevant character strings in keywords or hashtags. While there is some debate on how to interpret data collections based on keywords or hashtags, this is a conceptual question to be addressed in the justification of your choice of data collection and of little practical relevance in the collection of messages. Here, we provide you with a list of examples for the easy tracking of messages using specific character strings in keywords or hashtags.

First, access the command line on your machine. Now, access the directory you saved your code in using the `cd` command. You now have access to the functions defined in our example scripts. After this, start IPython by simply typing:

```
ipython
```

Let's say, your are interested in tracking messages containing the keywords *politics* and *election*. You, therefore, have to access Twitter's Streaming API to look for messages matching these criteria. The function `track_keywords` in our script `examples.py` helps you with this.

By importing the script `examples.py` you make our set of functions accessible from your workspace. Type in your command line:

```
import examples
```

Now, let's start tracking mentions of your words of interest on the Streaming API. Type:

```
examples.track_keywords()
```

Now, you should see a stream of messages crawling through your command line containing the words *politics* or *election*. You can stop the stream anytime by pressing `crtl-c`. You can save the messages identified by your query with the following function:

```
examples.save_track_keywords()
```

Based on this example, it should be easy for you to create scripts tracking character strings of interest to you on Twitter's Streaming API. For this, you can modify the respective functions to track keywords of interest to you. Simply exchange the keywords *politics* and *election* listed in the functions under `keywords` with a list of keywords of interest to you. You can list as many keywords as you like, up to a limit that depends on your specific API access privilege. If your list is too long, Twitter will return a `413 HTTP` error.

You can also identify character strings only contained in hashtags instead of keywords. To do so, you follow the procedure described above but precede the character strings of interest to you by the # character.

### *Account-Based Searches*

Instead of selecting messages based on character strings, we can also collect messages based on their authors. There are various strategies for defining lists of relevant users. The easiest way is selecting relevant users based on their profession or political role—such as politicians, candidates, journalists, or official accounts of media outlets (Graham et al., 2013). Another possibility is the selection of users exhibiting relevant behavior—such as using politically relevant words in their messages (Jungherr, 2015; Lin et al., 2013; Lin et al., 2014). No matter how you come up with your list of relevant accounts, our scripts help you in collecting their messages.

To illustrate our procedure, let's have a look at messages posted by Lawrence Lessig, a law professor and would-be candidate in the 2015/2016 Democratic primaries:

```
ipython
import examples
examples.print_user_archive()
```

What you see now is a list of all tweets available through Twitter's API posted by the requested account. While this is certainly interesting from an exploratory perspective, to further work with the

resulting data, it is helpful to save the results of your query on your hard drive. One possibility to do so is:

```
examples.save_user_archive_to_file()
```

Now, you should find a `.json` file in your working directory, containing all available tweets of the users specified by you in the function. It is important to note, though, that more often than not, you won't be able to collect all tweets posted by your selected users. Twitter limits the amount of messages available through the API to the last 3.200 messages posted by a user.

Finally, you are also able to track messages posted by users through Twitter's Streaming API:

```
examples.track_users()
```

Now, you should see a stream of messages crawling through your command line posted by *The New York Times* and *The Washington Post*. You can stop the stream anytime by pressing `crtl-c`. You can save the messages identified by your query with the following function:

```
examples.save_track_users()
```

As before, it should be easy for you to use these examples as a starting point and to adapt them to your research interests.

### *Downloading Lists of Tweets*

Finally, let's focus on an example showing you how to download a list of tweets based on their IDs. This comes in handy, if you are looking to reproduce a study, provided the authors published the IDs of tweets used in their analysis.

```
ipython
import examples
examples.print_list_of_tweets()
```

This provides you with three tweets identified by the IDs in our example script. You can supplement these IDs with a list of IDs of your choice. To save the thus identified tweets, you can easily adapt the function `save_user_archive_to_file` discussed above.

It is important to note, that while the function above is perfectly fine for collecting small numbers of tweets, for larger lists, we recom-

mend using our function `hydrate`. In the chapter *Data Analysis*, we give an example for how to use that function.

### *Further Reading:*

Here, we have only covered two of the most basic approaches for collecting data through Twitter's API. There are many other useful ways to access Twitter's APIs than simply those covered here by us. Russell (2014) offers a selection of very helpful example scripts. Have a look at them, if you feel your research interests are not adequately covered by the approaches discussed by us. Further helpful information can to found in Twitter's documentation to its REST[47] and Streaming[48] APIs.

[47] https://dev.twitter.com/rest/public

[48] https://dev.twitter.com/streaming/overview

# *Data Processing*

Data processing and handling usually do not figure very prominently in most descriptions of scientific workflows with digital trace data. Yet, this step takes up much of the time spent working with digital trace data and is crucial in ensuring the quality of data subsequent analyses are based on. Handling digital trace data involves countless small tasks that by themselves do not lead to big revelations, but nevertheless serve to safeguard the reliability, validity, and thus ultimately the trustworthiness of research. A rigorous approach to data processing also helps to decrease the workload for subsequent enquiries using the same data set. The processing of digital trace data takes raw information retrieved from an API, stores it in a reliable manner, and prepares it so that researchers can use it to answer specific research questions.

When working with digital trace data, reliable storage before data processing is crucial. No matter how sophisticated your data processing approach, be sure to keep an unaltered version of the data originally collected by you. Thus, you will always be able to turn to the original data. No matter how experienced you are, it is more than likely that during the analysis of your data, you will find that, for some reason or other, your processed data are not appropriate for your chosen analytical approach. In these cases, having a copy of your original data is crucial. Also, make sure to keep physical backups of your original data. This is very important as your data of interest might have become inaccessible between now and the time you originally collected them. Losing data to a hardware failure or human error might thus seriously damage your research project. While these practices seem like common sense. In practice and the hustle-and-bustle of research often conducted as just-in-time delivery, they are often neglected.

While we recommend redundant storage of your original data, the preprocessing of your data for analysis is best done by loading a copy of your data in a database. While at first, using a database might seem like unnecessarily steepening your learning curve for starting with the analysis of digital trace data, you will find that time invested

in developing some basic skills in the use of databases will pay off handsomely later in your research process. In most cases, databases allow you a faster and much more flexible access to summary statistics of your data than processing raw data. This gives you more flexibility and speed during the crucial early stages of exploring your data. Also, databases are very handy in exporting specific elements of your data set for analysis in dedicated analytical environments, such as R.

## *Getting Started with SQLite*

For the purposes of this tutorial, our database of choice is SQLite[49], an in-process database. SQLite does not need an external application and is transparent and intuitive as to where and how it stores its data. SQLite databases are files that can be stored in any directory you want. A program simply declares which database file it wants to use and an ORM (object relational mapper) handles all the details of storage. This makes the datasets portable and easy to handle. At the same time, the database allows us to search for specific content without needing to iterate over all the included items. There is one caveat: As with files containing raw data, we strongly recommend against accessing the database from multiple programs at the same time. SQLite offers better data consistency than raw files, but you should still separate data storage and analysis operations.

[49] https://www.sqlite.org

Digital trace data are not collected and structured based on a dedicated research design but come predefined based on criteria specified in the design of the respective platform's API. Especially studies involving hypothesis testing will need to perform some preprocessing before running actual analyses. If the steps involved are complicated and the computed values central to the research questions, it pays to specify these preprocessing steps already in the database. If, on the other hand, one only needs to do basic filtering, aggregation, grouping etc., then the database can take care of that.

In order to interact with the database, you can use either a special query language or a helper library, typically called ORM (object relational mapper). An ORM facilitates working with the database by abstracting many common tasks into methods that are easier to use and more consistent with your programming language of choice. We chose to use the peewee[50] library because it supports many different popular databases and provides an abstraction layer sticking closely to the SQL standard. The time spent to learn core concepts in peewee directly pays off as basic knowledge of SQL, a central programming language for the interaction with databases.

[50] http://docs.peewee-orm.com/en/latest/

Peewee offers another benefit. It works with most popular SQL-

based databases—among them PostgreSQL[51] (our favorite for large projects) and MySQL[52], both of which run as stand-alone server applications. This enables you to just download PostgreSQL, or another solution, and swap it against SQLite, in case you should you ever feel the need for larger or faster storage. Your code will continue to run without any major changes (apart from the specifics of database configuration).

[51] http://www.postgresql.org

[52] http://www.mysql.com

Getting started with SQLite is quick and easy. In all likelihood, OS X and Linux users already have a version of SQLite preinstalled on their systems. To make sure, please open your command line and just type:

```
sqlite3
```

This should start the version of SQLite installed on your system. To close the command line shell, just type:

```
.exit
```

Users of other systems should just visit the SQLite download page[53] and download binaries prebuilt for their systems, or get a copy of the sources and compile them themselves. To get some familiarity with SQLite, make sure to check out the official documentation on how to use SQLite from the command line[54]. For more information, you can also turn to introductory books offering you a more detailed account of the program (Allen and Owens, 2010; Kreibich, 2010).

[53] https://www.sqlite.org/download.html

[54] https://www.sqlite.org/cli.html

## *Using SQLite and peewee*

Both SQL and peewee have an incredibly broad range of functionality, which means we cannot provide a comprehensive overview. Instead, we direct you to peewee's very good and thorough documentation[55] as well as the blog[56] of its author, Charles Leifer, on which he frequently explains core concepts and gives helpful hints. The database module contains some helper functions for frequently used queries and hence should be useful reading as well. Here, we will only walk you through some basic operations in querying the database.

[55] http://peewee.readthedocs.org/en/latest/

[56] http://charlesleifer.com/blog/

The starting point of any query is a model. Models not only define the kind of data stored within them, they also provide a host of helper functions used for working with them. We thus have to create a database and define its underlying models before we can load

our data. In our example script `database.py` we present an example for such models. While this script will allow you to follow our subsequent examples, it might not be appropriate for your analysis of choice. So, you should think about adapting the scripts for your purposes.

In our example scripts, `database.py` handles the creation of a database and models relevant for the analysis of Twitter-based data. During this tutorial, you will use the script anytime you want to load tweets saved in a file in `json` or in `db` format, for exploratory analysis of your dataset, or for the export of summary statistics. To illustrate the workings of the script, we will walk you through some examples for its usage. But before you press on, you should definitely work through peewee's quickstart documentation[57] to gain some familiarity with peewee's notation and fundamentals of database handling.

[57] https://peewee.readthedocs.org/en/latest/peewee/quickstart.html

Now, let's collect some data! Start your engines and enter in your command line:

```
cd [your working directory with our example code]
ipython
import examples
examples.save_user_archive_to_database()
```

The last line loads all available messages posted by Lawerence Lessig and saves them in your working directory in `db` format. This process can take a few minutes. So bear with it. Be sure to clear out other files from previous runs from the working directory or to rename them. Otherwise, they will be overwritten. You will know that the process is completed once Ipython prompts you with an empty line starting with an `In` prompter. Of course, you can change this preset to any account you like by just changing the username listed in the function element `rest.fetch_user_archive`. Still, for the purposes of this tutorial let's use Lessig's tweets.

Once finished, you should find a file named `tweets.db` in your working directory. In our example, this file contains all available tweets and retweets posted by Lawrence Lessig. Now, we have to load them into a database. This process can seem a little complicated in the beginning, but we hope it will be clear once you have worked through it a few times. After all, nobody said digital trace data analysis would be all fun and games! We have documented this process in the script `database.py`. Here, we will walk you through the necessary steps. First, we have to load the python modules needed for our analysis:

```
import logging
import datetime
from dateutil import parser
from pytz import utc
import peewee
from playhouse.fields import ManyToManyField
```

Now, your workspace should be prepared for the next steps. Let's identify the database and connect to it:

```
db = peewee.SqliteDatabase('tweets.db', threadlocals=True)
db.connect()
```

After connecting to the database, we have to define the models establishing the structure of the database:

```
class BaseModel(peewee.Model):
    class Meta:
            database = db

class Hashtag(BaseModel):
    tag = peewee.CharField(unique=True, primary_key=True)

(...)
```

The first command defines our database object `db` as the basis for the following operations. The following commands define various `classes` with corresponding `fields`. Here, we show the code for the definition of the class `Hashtag` and its field `tag`. You should think of classes as database tables and fields as data columns.

After defining the structure of the database and its tables, we have to create functions that allow us to load data into their dedicated database fields:

```
def deduplicate_lowercase(l):
    lowercase = [e.lower() for e in l]
    deduplicated = list(set(lowercase))
    return deduplicated

def create_user_from_tweet(tweet):
    user, created = User.get_or_create(
        id=tweet['user']['id'],
        defaults={'username': tweet['user']['screen_name']},
```

```
        )
        return user

    (...)
```

Now, we have to define functions allowing us to summarize and query data loaded into the database:

```
    def database_counts():
        return {
            "tweets": Tweet.select().count(),
            "hashtags": Hashtag.select().count(),
            "urls": URL.select().count(),
            "users": User.select().count(),
        }

    def mention_counts(start_date, stop_date):
        mentions = Tweet.mentions.get_through_model()
        users = (User.select(User, peewee.fn.Count(mentions.id).alias('count'))
                 .join(mentions)
                 .join(Tweet, on=(mentions.tweet == Tweet.id))
                 .where(Tweet.date >= to_utc(start_date), Tweet.date < to_utc(stop_date))
                 .group_by(User)
                 .order_by(
            peewee.fn.Count(mentions.tweet).desc())
        )
        return users

    (...)
```

After running the preceding sections of the script, you have to use the following lines of code to actually set up the database and its underlying models:

```
    try:
        db.create_tables([Hashtag, URL, User, Tweet, Tweet.tags.get_through_model(
        ), Tweet.urls.get_through_model(), Tweet.mentions.get_through_model()])
    except Exception as exc:
        logging.debug(
            "Database setup failed, probably already present: {0}".format(exc))
```

You have to repeat these steps each time, you load a data set into a new database. While this configuration is completely sufficient to run

the examples in this tutorial, you might want to reconsider some of our choices if you find them impractical for your analytical interests. If this is the case, you have to adjust the `database.py` script according to your needs.

Now, let's have a quick look at how to perform summary statistics on the data in your database and how to query your database from your command line shell. For this, turn back to your open command line where you hopefully still have our database ready and loaded.

First, let's directly inspect the fields in our model by directly calling them using their names. Let's for example select the field `date` contained in the class `Tweet`:

```
Tweet.date
```

OK, not let's see how many tweets are in our database:

```
Tweet.select().count()
```

Now, let's see how many users posted the tweets contained in our database, were mentioned in tweets, or were retweeted by Lawrence Lessig:

```
User.select().count()
```

OK, let's turn to something a little more interesting. Let's look at the texts of all tweets in which Lawrence Lessig mentioned Donald Trump:

```
for tweet in Tweet.select().where(Tweet.text.contains("Trump")):
    print(tweet.text)
```

It could be that your milage of this request varies depending on when you are working through this tutorial. If querying for *Trump* might give you no results, depending mainly on how long the political fortunes of this controversial figure hold, you have to replace his name in the query with a term promising more successful results.

With this last example, you see that models might also serve as starting points for querying your data through a method called `select`. By calling `Model.select()`, we create a query that represents all model objects that are present in the database. The `select` statement stems from SQL and serves as a pre-filter defining fields we want to query. In theory, queries are slightly more efficient and faster if we specifically ask for a small selection of fields. We recommend,

however, not worrying about this until queries are prohibitively slow.

By layering queries, you can transpose core functionality into queries. Using composed queries makes them easier to understand and your code more readable. The key here is a function that (again, named after the corresponding SQL statement) is called `where()`. `Where` is called with statements that express the limitation that we want to filter on. For example, we could restate the query introduced above:

```
query = Tweet.select().where(Tweet.text.contains("Trump"))
```

The query itself is a generator object that only runs once we ask it for data. One can either iterate over the query, processing each tweet, or use a shortcut for getting the first object, such as:

```
first_tweet = query.get()
```

Accessing attributes of the class `Tweet` is then as trivial as typing

```
first_tweet.text
```

This simple example query illustrates the use of the operator `contains`. This is but one in a series of operators that are available for creating filter conditions. There are many more explained in peewee's documentation[58]. Make sure to familiarize yourself with the uses of these operators. This will provide you with rich analytical options for the summary and preparation for further analysis of your data.

[58] http://peewee.readthedocs.org/en/latest/peewee/querying.html#query-operators

Now, you have some basic understanding on how the data you collected on Twitter are stored in a database and accessible in preparation for your analysis. In the rest of the tutorial, we will not directly interact with databases but instead offer you functions defined in the script `examples.py` that automatically access the database according to their objectives. Still, especially for early explorative stages in a data set's analysis, directly querying a database from your command line shell offers easy and quick insights. Getting familiar with your database system of choice and peewee will thus pay off for you very quickly.

## *Issues to Keep in Mind While Working with Twitter Data*

While peewee or alternative ORMs (object relational mapper) might save you from learning SQL (a database focused programming lan-

guage) in order to work with your data, you still need to become comfortable with a few key-concepts:

- *Queries*: We interact with databases through queries. Queries can fetch specific or all records, insert new data, or change the way the database is structured.
- *Tables*: Just like an excel sheet or a R data frame, database tables represent a set of columns and rows. After defining a model, we create its corresponding table.
- *Foreign Keys /Many to Many Relationships*: Most of our models share links between them. For example, above, we defined a model named Tweet and a model named User. Each Tweet object must have exactly one author, who in turn is a User object. Because User and Tweet objects are stored in different tables, the database manages them via a feature called foreign keys. Instead of storing the entire information on User objects in the table that holds Tweet objects (which would duplicate informations on users repeatedly), it just stores a link. If Tweet object number 1 had User object number 1 with username *alpha* as its author, the database representation of the Tweets `User` field would contain the number 1. Many to many relationships are similar but allow storing a list of entities, such as a tweet with multiple hashtags.
- *Joins*: The database stores models in different tables. If we are interested in pieces of data in multiple tables, it needs to look at those tables in conjunction. A typical case would be searching for all tweets posted by one username. In this case, we would first look up the user with the given username, and then filter the tweets table by this ID. In SQL, such an operation is called a JOIN[59].
- *Transactions*: SQL databases store only data that correspond with their underlying model. They do so by checking any input first on its correspondence with the database's model and only accept it once it passes the test. The process of storing data is called a transaction. If it succeeds, the data is saved; if it fails, transactions are rolled back and the database returns to its previous state.

[59] http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

For this tutorial, we consciously decided on using a SQL database precisely because of this last characteristic. By only accepting data that conforms with its underlying previously defined models, SQLite helps identifying inconsistencies or errors in data sets that might otherwise remain undetected and potentially hinder further analysis. This is especially relevant when working with digital trace data. Often digital trace data are collected continuously and unsupervised

over long period of times. This introduces many opportunities for data corruption:

- The remote service may break and stop serving data at any point. This is a problem when the data collection is time-sensitive—because the content itself is volatile and may disappear (as users and/or platform operators delete it) or may become inaccessible (one example for this is Twitter's sample stream which is available as a real-time stream but cannot be accessed retroactively).
- The remote service may break and send wrong data. While rare, it is perfectly possible that any piece of data we receive is wrong—either in terms of content and/or of form. A good storage strategy should notify the researcher of such problems, instead of silently accepting erroneous data.
- The data format received may change. Most popular platforms changed the structure of their data, the methods for accessing their API, and other details multiple times over the last few years. So it is more than likely, that this will happen again.

Traditional (schema-based) databases can help with all three of these issues. They impose strict restrictions on the data that can be stored. In particular, we need to define the structure of the data in advance by designing data models. These models guarantee that anything that is stored in the database conforms to our expectations. It is not possible to store incomplete (for example a tweet missing its ID), duplicate (two tweets with the same ID) or wrong data (such as writing the tweet text into the ID field). By explicitly failing, the database makes errors visible that might have severe consequences for research but might otherwise go unnoticed.

In the previous chapters, we argued that a detailed understanding of the API, its methods, and objects is crucial for reliably collecting and storing data. Database models help a lot in this regard, as they represent an ideal abstraction layer. A best practice is to put your knowledge, assumptions, and validation logic into the models. The database will ensure that everything that gets stored conforms to these explicit expectations. Since subsequent analyses can rely on these guarantees, they do not need to concern themselves too deeply with the data collection logic and can instead focus on research questions.

It should be noted that, obviously, the correctness of the data depends on the correctness of the models. There are many cautionary examples where small misunderstandings of an API lead to mistakes in model design. In most cases, such errors quickly become apparent as the database complains about violated constraints. In some cases,

however, mistakes can be subtle or only manifest with high volumes of data. Here are two especially relevant examples:

- *Usernames are not Unique*: Twitter's data schemata have some implications that at first glance are not readily apparent. Before modeling the data structures used to store tweets, it pays to have a closer look at what the official documentation says about the uniqueness of objects. Tweets and users both have an ID field which contains a large number that is guaranteed to be unique. Ergo, there is no other user sharing that number and no other tweet sharing one ID. However, the same does not apply for other attributes, notably usernames (`screen_name`). Although Twitter does not allow users to pick a name that is already chosen, usernames become available again once the previous account using that name has been deleted or its author decides to rename it. In practice, this happens surprisingly often. The longer the timeframe of a data collection, the more likely it is that usernames collide. A robust solution to this problem is to always use user IDs, not usernames as identifiers for users. The database schema below does this by defining the user model with an unique ID, but allowing (non-unique) any username.

  The example dataset provided with the package actually contains seven cases where usernames collide. You can find them by inspecting the list of usernames. You first have to download the tweets used in this study from Twitter's API and load them into a database. We show you how to do this in the following chapter. Once you have done this, you could create a list of all usernames in the database's class `Users`:

```
usernames = [user.username for user in User.select("username")]
```

  Now, we create a counting object from the list:

```
from collections import
Counter namecount = Counter(usernames)
```

  We then show the ten usernames that occur the most often across all users. The number following the string corresponds to the number of unique users with this name.

```
print(namecount.most_common(10))
```

This should produce the following output:

```
Out[1]: [("halseycupcakes", 2), ("bootsntwinks", 2), ("kilIdrake", 2),
("junhoestan", 2), ("StephenWolfUNC, 2), ("JackofKent", 2),
("oooh_sebastian", 2), ("neumantj", 1),  ("JohnHollings", 1),
("Guidotoons", 1)]
```

- *Times and Timezones*: Date and time are crucial for the analysis of digital trace data. Yet, they are difficult to handle. The challenge lies in reliably transforming an universal reference time to a local time. To do this, we usually have to add a timezone offset to transform a universal reference time to a local time. During this step, it can also be necessary to add a daylight savings offset. The Twitter API returns date/time in UTC, the universal reference time (no timezone offset, no daylight saving time). However, depending on your object of study, users will see and use their local time when interacting with Twitter. In many cases, we thus need to convert the native UTC dates into the local timezone of interest. To do this without introducing unnecessary sources of error in your data, you should:

  - Always store and/or explicitly declare the timezone information of your data;
  - Convert your time/date information at the latest opportunity possible in your research process;
  - Store data in the most generic format possible (this usually means UTC).

  In our example scripts, we have to compromise on date handling since SQLite does not support timezones. One option is storing the `datetime` object as a string. But this prevents efficient date range queries. Instead, we chose to store UTC datetime without timezone information. So please keep in mind that you might need to convert both your queries and the resulting data. For example, if you are interested in tweets from New York City on January 1st of 2015, midnight to noon, you would first convert the range to UTC times:

```
from pytz import timezone
from pytz import utc
from datetime import datetime
```

While it is possible to use timezone abbreviations such as MST, some of those (such as CST) are ambiguous! While the `pytz` package will prevent you from using them, readers and users of your code might be misled. So be sure to be as explicit as possible in your code and its documentation. For our fictitious example, one possible transformation code might look like this:

```
ny = timezone("America/New_York")
start_date = datetime(2015, 1, 1, 0, tzinfo=ny)
stop_date = datetime(2015, 1, 1, 12, tzinfo=ny)
start_date_utc = utc.normalize(start_date)
stop_date_utc = utc.normalize(stop_date)
```

The source code to the database module contains some additional hints for using alternative storage solutions.

*Further Reading:*

To get background information on using SQLite you can try the official documentation online[60] or an introductory book (Allen and Owens, 2010; Kreibich, 2010). To get to grips with peewee try its documentation[61] or the blog[62] of its creator Charles Leifer, on which he frequently explains core concepts and gives helpful hints.

[60] https://www.sqlite.org/docs.html

[61] http://docs.peewee-orm.com/en/latest/index.html

[62] http://charlesleifer.com/blog/

# *Data Analysis*

After the collection and preparation of Twitter data, we will now briefly turn to examples for the analysis of digital trace data collected on Twitter. Examining the literature shows an staggering and ever increasing number of analytical approaches to Twitter data. We find approaches relying exclusively on digital trace data (Barberá and Rivero, 2015; Goel et al., 2015; González-Bailón et al., 2011; Lin et al., 2014; Theocharis et al., 2015), studies combining digital trace data with surveys (Barberá, 2015; Jungherr, Schoen, and Jürgens, 2015; Vaccari, Chadwick, and O'Loughlin, 2015), other quantitative metrics of political phenomena (Bastos, Mercea, and Charpentier, 2015; Jungherr, 2014; Trilling, 2015; Zeitzoff, 2011), or ethnographic approaches (Dubois and Ford, 2015; Jackson and Welles, 2015; Kreiss, 2014).

We also find strong variety in analytical methods. Some researchers include digital trace data in regression models explaining political behavior (Peterson, 2012; Vergeer and Hermans, 2013), perform qualitative content analyses of tweets (Graham et al., 2013; Jungherr and Jürgens, 2014b; Theocharis et al., 2015), try their hands at the (semi-)automated detection of sentiment contained in tweets (Frank et al., 2013; González-Bailón and Paltoglou, 2015), build models linking political characteristics or outcomes to patterns in digital trace data (Barberá, 2015; Metaxas, Mustafaraj, and Gayo-Avello, 2011), construct networks based on interactions between Twitter users (Conover et al., 2011; González-Bailón et al., 2011; Jürgens, Jungherr, and Schoen, 2011), identify temporal patterns in the distribution of tweets (Jungherr and Jürgens, 2013; Jungherr and Jürgens, 2014a), or create models of information diffusion online (Goel et al., 2015; Myers and Leskovec, 2014).

Of course, it is impossible for us to cover the whole variety of approaches available to you in this tutorial. Instead, we will focus on offering you examples on how to use the data collected by you over the course of this tutorial for three typical Twitter-based analyses: Counts, time series, and networks. These three approaches are very basic but offer you a first view of what might be in store for you on

your way through the garden of forking paths of potential analytical approaches. Once you have worked through the examples provided by us, we recommend you have a look at some of the studies listed in our bibliography. They offer exemplary cases of where you might take your own research using digital trace data.

### *Download the Data Used in the Following Analyses*

The analyses presented here were run using data documenting politically relevant tweets posted during the fourth televised debate in the Republican primaries for the US-presidential election 2016 on October 28, 2015[63]. To allow you to reproduce the examples presented here, we prepared a file containing all IDs of tweets originally collected by us between October 27 and November 3, 2015, `example_dataset_tweet_ids.txt`. As described above, files like this allow the reproduction of Twitter-based studies. You only have to download the listed tweets through Twitter's API and you are good to go. In our `examples.py` script, we provide you with a function taking care of this process for you.

[63] https://en.wikipedia.org/wiki/Republican_Party_presidential_debates,_2016

Before you start downloading the tweets listed in the reproduction file `example_dataset_tweet_ids.txt`, be warned that this can easily take a few hours, due to the amount of tweets listed there. While the function as presented by us allows you to pause your data collection and seamlessly pick it up at the first tweet not already collected by you, it can be a while until you are able to run the following examples with our data. So if you want to press on directly, it might be a better idea for you to collect an original data set—for example all tweets available for a select set of interesting accounts through our `save_user_archive_to_database()` function, introduced above. The following examples should run perfectly well with these data but will of course return different results. Also, keep in mind that the analytical choices made by us were made in the context of a large data set with many messages being posted at any given time. Depending on the specifics of your data collection, some of these choices might not return sensible results.

If you choose to reproduce our examples directly, this is how you get the necessary data. First, you prepare your workspace. Start the command line and type the by now familiar commands:

```
cd [your working directory with the file example_dataset_tweet_ids.txt]
ipython
import examples
examples.hydrate()
```

Now, you should see a dialogue in your console informing you on the count of messages left to download and a rough estimation of the expected run time. Don't be alarmed if your console keeps returning the line

```
ERROR: root: UNIQUE constraint failed: tweet.id
```

while fetching tweets. In the script, we made the choice to download retweets together with their source—the original tweet. In cases when an original tweets was retweeted more than once, our script tries to download the original tweet again but is informed that the original tweet is already in the database. This leads the script to return the error above. While loading our reproduction data set for the first time, seeing this error, therefore, just means the script is working as designed.

The function as given by us has default values for the name of the file containing the IDs and the filename the resulting data will be saved in—`tweet.db`. You can easily exchange the name of the file containing our example IDs for a file with IDs of interest to you. But be sure to change other occurrences of the default names used by us in the other example scripts in your code as well. Otherwise, you will run into trouble down the line. For the purposes of this tutorial, we thus recommend that you stick with the name `tweet.db`.

As before, if you want to cancel the collection, just hit `ctrl C`. You then can pick up the collection at any given time in the future. Well, this is it for now. Time to wait and catch a movie or read a book—*War and Peace* seems to be a popular choice.

## *Counts*

The most basic approach to the analysis of Twitter data is counting entities. You could count the mentions of actors be it by username, as hashtag, or keyword, mentions of specific character strings in hashtags or keywords, links to objects outside of Twitter, or the use of Twitter's usage conventions, such as @mentions, retweets, or hashtags. For a short list of usage conventions see Table 1, for a more comprehensive discussion see Jungherr (2015).

While counts are only a first step in understanding communication on Twitter, they have been used successfully by researchers to address various topics. Counts have been used to analyze dominant topics of conversation during events of interests (Jungherr and Jürgens, 2014a; Rogers, 2013a) or to identify prominent actors (Jürgens and Jungherr, 2015; Larsson and Moe, 2012). It could also be shown that the use of specific usage conventions changes de-

| Usage Convention | Description |
|---|---|
| @reply to another user | to publicly address other Twitter users, one precedes the text of a message with the username of the addressee and an @ (i.e. @username) |
| @mention of another user | one can also user the @username convention in the text of a messages instead of the beginning, this is called an @mention |
| RT verbatim | a retweet (RT) is a verbatim copy quote of another tweet, to do this one copies the tweets and precedes it with the character string *RT @username* |
| RT modified | one can also comment or modify a quote message, this is called a modified retweet |
| #keywords | to establish an explicit context for a tweet, one can use keywords preceded by the # sign, so called hashtags |
| links to other Web content (e.g. websites, pictures, videos et al.) | one can also post links in messages to content on the Web, these links are often shortened to accommodate Twitter's 140 character limit |

Table 1: Usage Conventions on Twitter

pending on the types of events Twitter-users were commenting on (Jungherr, 2015; Lin et al., 2014). Some researchers even went so far as to expect mentions of political actors to predict their electoral fortunes (Gayo-Avello, 2013). While ultimately the hope of being able *to predict elections with Twitter* might be far-fetched (Diaz et al., 2014; Huberty, 2015; Jungherr, Jürgens, and Schoen, 2012; Metaxas, Mustafaraj, and Gayo-Avello, 2011), we believe that by identifying Twitter users' objects of attention Twitter data provide valuable insights in the dynamics of political communication.

By mentioning an account, referring to a specific topic by using a hashtag, retweeting specific messages, or linking to specific content on the Web, Twitter users identify actors, topics, and objects of relevance to them. By aggregating these signals over specific time spans, we are able to gain insights in objects that a majority of relevant Twitter users found of relevance during that time. In the past, we have called this process *collective curating* (Jungherr and Jürgens, 2014b; Jungherr, 2015). By focusing our analysis on the results of this *collective curation process*, we are able to gain insights into the interests and dynamics of Twitter as a political communication space.

Here, we offer a few example scripts illustrating simple counts of entities based on the prepared data set described above. Counting items in the data set boils down to four basic operations: Filtering, grouping, counting, and sorting. Filtering narrows down the data set, most commonly to limit its date range. Grouping collects items belonging into some common category together—such as linking mentions of each user to their ID. The counting step in SQL then assigns a count value to the grouper variable. In other words, the user object is annotated with a *count* variable that contains the number of mentions received. Finally, the query is sorted by that count variable

so that the most mentioned user is returned first. How exactly we told the database to perform these steps is beyond the scope of this chapter—but feel free to have a look at the annotated `mention_counts` function in the `database.py` module.

The example module contains the relevant export functions that output the top users, retweets, hashtags, and URLs. We will illustrate their use by identifying prominent actors, topics, and objects in messages commenting on the fourth televised debate in the Republican primaries for the US-presidential election 2016 on October 28, 2015.

### *Identifying Prominent Actors*

Let's look at two different approaches at identifying popular users in our data set. In principle, we could identify prominence by the number of @replies, @mentions, retweets, hashtag mentions, or keyword mentions a user received. In the following examples we will identify the top 50 users according to their @replies or @mentions and retweets.

First, we have to prepare our database:

```
cd [name of the working directory you saved the reproduction database]
ipython
```

Now, you proceed as we discussed in the section on Data Preparation. After loading our reproduction data set in a database. Let's check the number of tweets and unique users:

```
Tweet.select().count()
User.select().count()
```

We find 788.229 messages and 291.589 unique users in our database. As discussed, your results might vary somewhat depending on the messages still available when you first downloaded the reproduction data set. Now, let's identify the users mentioned most often in @replies and @mentions:

```
cd [name of the working directory you saved our scripts]
import examples
examples.export_mention_totals()
```

Now, you'll find a new file in your working directory called `mention_totals.csv`. In the file you find a list of 50 accounts most often mentioned in the the tweets collected in the database. The first

column contains usernames while the second column contains their mention counts.

Table 2: Most Mentioned Users

| Usernames | Mentions |
|---|---|
| realDonaldTrump | 187866 |
| HillaryClinton | 62128 |
| BarackObama | 40819 |
| tedcruz | 38014 |
| CNBC | 30790 |
| marcorubio | 27526 |
| BernieSanders | 26828 |
| JebBush | 24901 |
| RealBenCarson | 23841 |
| DanScavino | 16985 |
| FoxNews | 15337 |
| RandPaul | 12500 |
| CarlyFiorina | 10497 |
| GOP | 8679 |
| POTUS | 7417 |
| RickSantorum | 7008 |
| CNN | 6390 |
| GovMikeHuckabee | 5844 |
| ChrisChristie | 5720 |
| Morning_Joe | 5268 |
| (...) | (...) |

Your output should return similar results to those reported in Table 2. Unsurprisingly, we find that in tweets collected based on their political relevance, accounts of political candidates, prominent politicians, campaign accounts, prominent consultants, media outlets, and journalists are dominating the conversation.

The function provided by us makes specific choices that you can freely adjust for the purposes of your analysis. First, we only included the 50 accounts most often referred to in messages contained in our database. You can easily include more accounts by changing the respective values in the first and tenth line of the function. Second, we provide a start and stop date between which mentions are aggregated. You have to adjust these values in line six and seven according to your analytical interests. Finally in line ten, we define the database operation invoked by the function: `database.mention_counts`. Here, we count mentions listed in Twitter's `user_mentions` field[64]. By calling a different database query here you can change our mention interpretation to one in accordance with the specific needs of your analysis.

[64] https://dev.twitter.com/overview/api/entities

One such alternative would be to identify the most retweeted accounts in our dataset. For this popular query, we provided you with a dedicated function `export_retweet_totals`. The use of the function is completely analogue to the one presented above. Go ahead and compare the results.

| Usernames | Retweets |
|---|---|
| realDonaldTrump | 25536 |
| DanScavino | 7457 |
| FoxNews | 4282 |
| Amaka_Ekwo | 3725 |
| Drudge_Report_ | 2779 |
| hdpdemirtas | 2471 |
| ussoccer_wnt | 2256 |
| CNNPolitics | 2180 |
| BarackObama | 2064 |
| HillaryClinton | 1884 |
| DailyPresRaps | 1840 |
| LastWeekTonight | 1513 |
| CNBC | 1362 |
| DefendingtheUSA | 1308 |
| DiamondandSilk | 1278 |
| man_vs_liberals | 1250 |
| tedcruz | 1218 |
| CNN | 1203 |
| StrengthenTheUS | 1111 |
| hannahkauthor | 1073 |
| (...) | (...) |

Table 3: Most Retweeted Users

When we compare the accounts listed in Table 3 with those listed in Table 2, we quickly see that although politicians, media accounts, and journalists are prominent in both lists, a greater variety of accounts was prominently retweeted than prominently mentioned. During televised media events, users apparently focus their attention measured by mentions on actors central to these events, while showing much less exclusive focus on these actors in their retweeting behavior. Different usage conventions seem to be associated with different behavior and thereby might also signify different aspects of attention towards politics. You should be careful to account for these differences in your operationalization and interpretation of your analysis.

### *Identifying Dominant Topics*

We can also focus on topics of interest to Twitter users during given time intervals. Here, we will illustrate two approaches: Identifying the most prominently used hashtags and identifying the most often retweeted tweets. Let's start by looking at prominent hashtags.

First, you have to prepare your workspace and load the database as discussed in previous sections of the tutorial. Then, if you haven't done so already, you have to import `examples.py` and select the working directory you want to save the results of your analysis in. Now, call the function `export_hashtag_totals`:

```
cd [name of the working directory you saved our scripts]
```

```
import examples
cd [name of the working directory you want to save results to]
examples.export_hashtag_totals()
```

The script creates the file `hashtag_totals.csv` in your working directory. Following the specifications in the function, this file contains the 50 most often used hashtags and their respective usage counts.

Table 4: Most Often Used Hashtags

| **Hashtags** | **Counts** |
|---|---|
| gopdebate | 48855 |
| trump2016 | 21631 |
| cnbcgopdebate | 19740 |
| tcot | 16860 |
| makeamericagreatagain | 14424 |
| cruzcrew | 12660 |
| pjnet | 11138 |
| obama | 9649 |
| feelthebern | 8467 |
| trump | 7837 |
| gop | 6991 |
| trumptrain | 6316 |
| uniteblue | 6040 |
| hillaryclinton | 5363 |
| money | 5252 |
| supernatural | 5127 |
| earn | 5111 |
| earnfromhome | 5021 |
| wakeupamerica | 5008 |
| hillary2016 | 4278 |
| (...) | (...) |

As Table 4 shows, the most prominent hashtags used in politically relevant messages reflect nicely the nature of the underlying event, the televised debate. We find the names of prominent candidates, hashtags referring to their campaigns and the Republican party, and the television station broadcasting the debate. But we also see hashtags not related to politics, such as the hashtag #*supernatural* most likely referring to the television series of the same name.

You might have recognized that all hashtags listed in Table 4 are lowercase. This is no accident. To account for many potential spelling variations in the use of hashtags, we decided to transform all hashtags to lowercase and count their appearance afterwards. This is done through the function `deduplicate_lowercase` in the example script `database.py`. If this choice might be not appropriate for your analysis, you should adjust the `database.py` script accordingly. Of course, you can also adjust the function's other parameters predefined by us. To do so, follow the approach described above.

Another view on prominent topics offers the analysis of prominent retweets. For this, prepare your workspace and call the function:

```
examples.export_retweet_text()
```

As with the examples before, this will create a new file in your working directory titled `retweet_texts.csv`. With the specification chosen for our example, this file contains the texts of the 50 most often retweeted tweets during the specified time interval. As before, you are of course free to adjust these specifications. The first column contains the respective tweets' texts, while the second column contains their retweet count.

Table 5: Most Prominent Retweets

| **Tweet Body** | **Retweet Count** |
|---|---|
| Bak böyle bir sey varmis hiç de söylemiyorsun, zalim @BarackObama https://t.co/01p3k55gLR | 2334 |
| This team taught all of America's children that playing like a girl means being a badass" - President @BarackObama. https://t.co/O3eYhNGTGq | 2025 |
| A couple of points… Yes, we have a boring show. At no point did we invite Donald Trump to appear on it. https://t.co/qjpg9FLboV | 1513 |
| @jebbush is a disgrace to the Bush name. lol. | 1001 |
| confirm this @realDonaldTrump https://t.co/rKpM6lKjeC | 847 |
| @BarbaraJensen1: @realDonaldTrump @OANN @GravisMarketing https://t.co/evyzpgIJ1V | 841 |
| @twillnurse: @realDonaldTrump saw this and had to share! He wants to be you! Love it!!!! U r his HERO! https://t.co/dL22Vuon6q So cute! | 812 |
| @Morning_Joe: Online poll: @realDonaldTrump 'best to handle economy' by far https://t.co/SZvSzNzoIk Very true, thanks! | 759 |
| Barack Obama singing Boyfriend by Justin Bieber https://t.co/jvgccBVkCC | 620 |
| Barack Obama singing SexyBack by Justin Timberlake https://t.co/O3U5EXM36m | 608 |
| @N_R_Mandela: @realDonaldTrump And you're still a negative loser and Trump is still a positive winner. I'm Black and proudly voting TRUMP! | 607 |
| Loved doing the debate last night on @CNBC. Check out all of the polls! Everyone agrees that Harwood bombed! | 594 |
| I miss you son @realDonaldTrump | 581 |
| @RhatPatriot: @FoxBusiness @realDonaldTrump Why not post the other polls where Trump has 40 percent and Carson is in the teens? Strange? | 530 |
| Trick-or-treat? #HappyHalloween https://t.co/0T8yc2vYyk | 521 |
| VIDEO fr @CNBC: Vice is nice? @SullyCNBC talks "sin stocks" with Barrier Fund manager https://t.co/BiL4SqbsZ7 https://t.co/nK5rlXgloO | 504 |
| @FreeStateYank: The only way anybody's gonna beat Trump is being better than he is.~@rushlimbaugh on @realDonaldTrump. | 502 |
| President Obama plays with Ella Rhodes in her elephant Halloween costume. https://t.co/Bi3GK3ZHvz | 501 |
| J Cole thoughts on Obama ðŞŤěðŞŤě http://t.co/NSXtmf1wjz | 495 |
| . @BarackObama Your time almost up, sneak this in for the hood. http://t.co/EOMD97RUw1 | 486 |
| (…) | (…) |

Analyzing the tweets listed in Table 5 shows some characteristics of Twitter communication during televised political events already

identified in other studies (Anstead and O'Loughlin, 2011; Freelon and Karpf, 2015; Jungherr, 2014; Trilling, 2015). Users predominantly retweet tweets containing quotes of candidates and prominent politicians, humorous comments, and tweets containing links to content on the web contextualizing the debate. Analyzing popular retweets thus offers a close view of content and issues dominating the political communication space on Twitter.

### *Identifying Dominant Objects*

Finally, let's look at prominent objects linked to by Twitter users in our database. Unfortunately, links provide a much more problematic object of analysis than usernames, hashtags, or retweets as their use is much less standardized than the use of other conventions.

Identifying and counting links raises a series of specific challenges. For example, URLs might contain spelling variations rendering them invalid, point to removed content, or multiple URLs might lead to the same content. In addition, Twitter's character limit led to the widespread use of so-called URL shortening services such as bit.ly[65], which create short versions of longer URLs. While these services have obvious benefits for users, they create further challenges in the analysis of Twitter-based data. Shortened URLs contain no information on their destinations unless opened in a browser. The Twitter API provides a dedicated field `expanded_url` in which it lists the destination of shortened links in tweets. For the purposes of this tutorial, we only use the information contained in this field. Yet, relying on the field `expanded_url` does not guarantee that all links are unshortened. Furthermore, while the information in the field `expanded_url` easily lends itself to quick analyses, you might want to also consider identifying links in the text body of a tweet itself and unshorten links yourself. This renders your analysis independent of Twitter's chosen algorithms and might raise your–and reviewers'—confidence in your reported results.

[65] https://bitly.com

To identify popular links prepare your workspace like you did for the preceding examples and call the function:

```
examples.export_url_totals()
```

Now, you should find a new file in your working directory titled `url_totals.csv`. As in the examples before, the file contains the 50 links most often used in the tweets in our database and their usage counts.

Examining Table 6 points to the issues raised above. Quite a few of the links here do not point to addresses on the Web but instead

Table 6: Most Often Used Links

| Link | Counts |
|---|---|
| https://twitter.com/asliaydintasbas/status/659446195081859073 | 2344 |
| https://twitter.com/realdonaldtrump/status/660597552023228416 | 1630 |
| http://ift.tt/psnbl7 | 1547 |
| http://ift.tt/1ucdor1 | 1376 |
| http://ift.tt/1qnk1to | 1308 |
| https://goo.gl/t4fpx2 | 1225 |
| https://twitter.com/uberfacts/status/588798277941878784 | 855 |
| https://twitter.com/realbencarson/status/659777986854559744 | 807 |
| http://www.therealstrategy.com/putin-exposes-obama/ | 736 |
| http://tedcruz.org | 713 |
| http://www.therealstrategy.com/strong-cities-network-next-step-of-nwo/ | 712 |
| http://www.mevee.com/lindseykroning | 635 |
| http://bit.ly/1jptbjb | 618 |
| http://www.therealstrategy.com/hillary-clinton-puts-another-nail-in-her-coffin/ | 592 |
| http://3tags.org/l/be2w | 574 |
| http://www.therealstrategy.com/breaking-obama-declares-himself-king/ | 554 |
| http://garyforbes.wix.com/blog | 551 |
| https://twitter.com/hillaryclinton/status/659555444202070016 | 519 |
| http://nbcnews.to/1jmdo6e | 507 |
| http://dlvr.it/cbpwdk | 505 |
| (...) | (...) |

contain links to tweets. Others have not been unshortened by Twitter. This shows that to get a true understanding of popular links, you can not entirely rely on an automated approach. Instead, you have to examine the results provided to you by the automated count and enrich the information by hand. This can be the unshortening of links or the identification of duplicate links to identical addresses.

Examining the content of the most prominent linked to sites, we largely see patterns known from research on tweeting behavior in the context of political television programs (Anstead and O'Loughlin, 2011; Freelon and Karpf, 2015; Jungherr, 2014; Trilling, 2015). Users point to content contextualizing comments by candidates in traditional and non-traditional media either in support or critique, users point to specific tweets by other Twitterers deemed relevant by them, and users point to humorous content. Analyzing objects linked to by Twitter users during politically relevant events might thus contain significant insights into the public negotiation of meaning during these events.

*Further Reading:*

In Jungherr (2015) you find a more extensive discussion on limits and potentials of using Twitter in researching political communication. Richard Rogers also offers an account of Twitter as an object of and and tool for research (Rogers, 2013a).

## *Time Series*

Another approach to the analysis of data collected on Twitter is the focus on temporal dynamics. For this, you have to extract entities of interest from your collected Twitter messages and transform them into time series. In principle, you can transform all entities in Twitter messages that you can count into time series. You only have to decide on time spans—or time bins—over which to aggregate counts of entities of interest to you. The length of these time bins depends on your research question. Sometimes, it might be appropriate to focus on daily aggregates. Shorter periods, such as hours or minutes might also be appropriate, depending on your interests.

Researchers have focused on time series of messages containing specific hashtags, time series of messages posted by specific users, and time series of messages containing specific usage conventions (Hanna et al., 2013; Jungherr, 2015; Lin et al., 2013; Lin et al., 2014). Here, we will show you how to extract time series like these from the messages in our data set.

### *Time Series of Hashtags Used in Messages*

Let's start by examining the time series of all messages identified by our selection criteria. Here, we will provide you with a guided walkthrough of our script `time_series_analysis_single_series.R`. While we discuss the steps of our analysis in detail, for the intricacies of R and the graphic package `ggplot2`, we recommend you turn to dedicated introductory books. For R see for example (Kabacoff, 2015; Matloff, 2011). For `ggplot2` see for example (Chang, 2013; Wickham, 2009) or the tool's online documentation[66].

[66] http://ggplot2.org

First, you have to identify the data of interest to you in your database. Then, you have to extract the resulting time series and transform them into a form you can read into R and perform your analysis on.

When building a time series from a stream of events—in our case tweets—you have to define a *bin width*. You might, for example, be interested in the count of messages per day, per hour, or per minute. After deciding on a bin width appropriate for your research question, you calculate for each bin over the time span of interest the aggregate count of messages falling in it. The helper function `objects_by_interval()` in the example script `database.py` takes care of this for you. Using this function, you can generate time series from any kind of data stored in the database.

To further make things easier, the `examples.py` script also contains six pre-made export functions that offer the most common use cases.

They are preconfigured to perform the analyses discussed in this section. So, simply calling them without any parameters, will produce the datasets required for the examples discussed below. These results will be saved as `csv` files, readable by R and other software. The specifications of these functions can be easily changed to accommodate your research interest. For example, if you want to change the aggregation interval from `day` to `hour`, you change the functions' `interval` specification from `day` to `hour`. Table 7 lists the export functions provided by us.

Table 7: Time Series Export Functions

| Function | Description | Predefined Specification |
|---|---|---|
| export_total_counts | Time series of all messages in database in given time bins | Daily aggregates |
| export_featureless_counts | Daily counts for tweets not containing mentions or URLs and are not retweets | Daily aggregates |
| export_hashtag_counts | Time series of hashtag mentions in given time bins | Daily aggregates of hashtag mentions of primary candidates |
| export_keyword_counts | Time series of keyword mentions in given time bins | Daily aggregates of keyword mentions of primary candidates |
| export_mention_counts | Time series of account mentions in given time bins | Daily aggregates of account mentions of primary candidates |
| export_user_counts | Time series of tweets posted by given accounts in given time bins | Daily aggregates of tweets posted by primary candidates |

As an exercise in adapting the function to your purposes, try exporting the hourly data for the hashtag *#gopdebate* using the function `export_hashtag_counts()`. Here is a hint, the function's default parameters are:

```
export_hashtag_counts(
    interval="day",
    hashtags=["Bush", "Carson", "Christie", "Cruz", "Fiorina", "Huckabee", "Kasich",
        "Paul", "Rubio", "Trump"])
```

After exporting the time series, let's briefly focus on case sensitivity. Automated coding of text requires very careful handling of case. Unless the information contained in upper- or lowercase spellings is important, we recommend converting text to lowercase for pattern matching. In the example code, we retain the original spelling in the database but convert it to lowercase for queries. One typical use can be found in the function `export_hashtag_counts`:

```
(...)
.where(peewee.fn.Lower(database.Hashtag.tag) == tag.lower())
(...)
```

There, we hand over the task of converting the tweet to lowercase to the database by using the helper function `peewee.fn.Lower`. This makes the procedure much more efficient, as we do not need to manually iterate over every item.

Now, after you have become familiar with the general use of our export functions, let's look at some example analyses using these data. Let's start by examining time series of all messages. To export the respective time series, start by preparing your workspace, loading and preparing the database containing the messages of interests, and activating the working directory you want to save your data in. Then type:

```
examples.export_total_counts()
```

You should find the time series of interest saved in the active working directory under the title `total_counts.csv`. Now, you are ready to read it into R. So start your copy of R or RStudio.

First, we have to make sure you have the necessary tools. For the following analyses you need to install two packages, `ggplot2` and `scales`. `ggplot2`[67] (Wickham, 2009) is a very powerful graphic package that allows R to plot data. `scales`[68] (Wickham, 2015) helps R to transform data to construct breaks and labels for plot axes and legends. Let's install these packages. Tell R to:

[67] http://ggplot2.org

[68] https://cran.r-project.org/web/packages/scales/index.html

```
install.packages(c("ggplot2","scales"))
```

You have to load any package you want to use in a specific analysis into your R workspace. This is easily done:

```
library(ggplot2)
library(scales)
```

Now, we have to load the data you want to analyze. First, you have to select the directory you have saved your data to:

```
setwd("/path_of_directory_you_saved_your_data_to")
```

Let's load your data:

```
message_counts_df<-data.frame(read.csv("name_of_your_data_file.csv"))
```

From this data frame, you now have to extract the columns of interest to you. For the first step in our analysis let's focus on the total volume of messages identified by our selectors. First, we have to extract a vector containing the dates of our observations.

```
dates<-as.POSIXct(message_counts_df[,1])
```

The command `as.POSIXct` tells R that the selected column contains time data. R has considerable abilities in dealing with time data once it has recognized the data as such. This identification process can sometimes be a bit complicated, though. For detailed discussions of this see Kabacoff (2015) and Teetor (2011).

Now, let's select the column in the data frame containing the daily counts of all messages collected by us:

```
all_messages<-message_counts_df[,2]
```

For plotting the time series of all messages, we need to combine both vectors in a new data frame:

```
plot_all_messages_df<-data.frame(dates,all_messages)
```

We recommend, you save plots in a dedicated subdirectory in the directory you hold your data. In our example, we have called this subdirectory `_plots`. To activate this subdirectory, type:

```
setwd("/path_of_directory_you_saved_your_data_to/_plots")
```

Now, let's create an object containing the necessary information to allow `ggplot2` to plot the data:

```
plot_all_messages<-ggplot(plot_all_messages_df,aes(x=dates,all_messages))+
    geom_line(stat="identity") +
    geom_point(size=2)+
    theme_bw()+
    xlab("")+
    theme(axis.text.x=element_text(angle=45,hjust=1))+
    ylab("Message Volume, Daily")
```

Let's have a look at what you have created:

```
plot_all_messages
```

Now, let's save the plot:

```
ggsave(file="Message Volume Daily.pdf", width = 170, height = 90, unit="mm", dpi=300)
dev.off()
```

Figure 1 shows a more or less stable baseline in the volume of messages just above 100.000 messages per day. The day of the debate, October 28, does not show a significant deviation from this pattern.

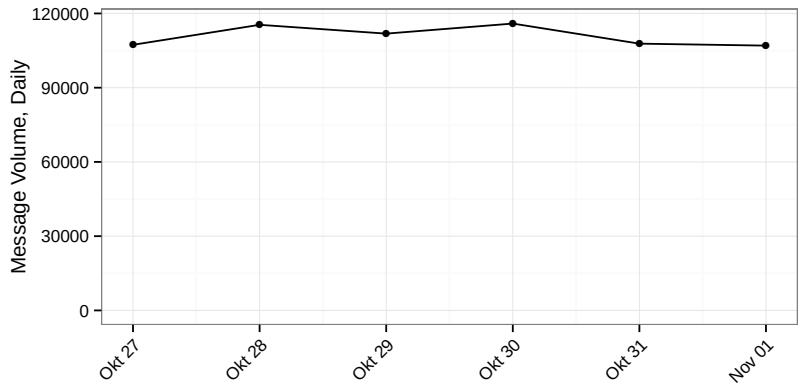


Figure 1: All Messages, Daily Aggregates

Let's have a look at the debate-specific hashtag #*GOPDebate* to see if the debate left clearer traces here. For this, you first have to export the respective time series by calling the function:

```
examples.export_hashtag_counts()
```

But be sure to adjust the function's specifications according to your interest.

The process of analyzing messages using the hashtag #GOPDebate is nearly identical to the process analyzing the volume of all messages. The only difference being that you load a different data set:

```
setwd("/path_of_directory_you_saved_your_data_to")
gopdebate_counts_df<-data.frame(read.csv("name_of_your_data_file.csv"))
```

In Figure 2, we see a clear spike in the volume of messages using the hashtag #GOPDebate on September 28, while the volume of

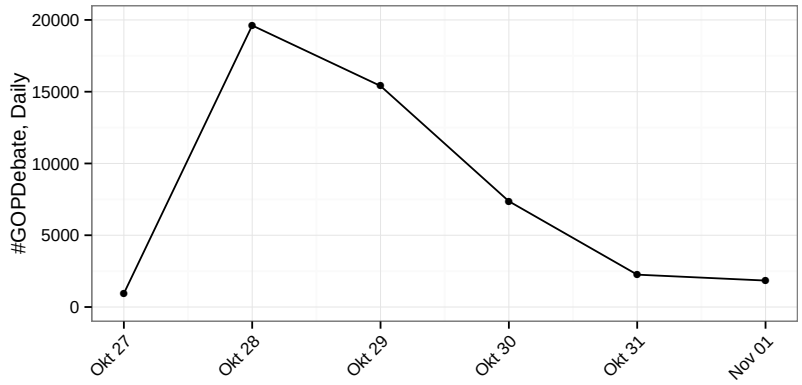


Figure 2: All Messages, Daily Aggregates

Twitter messages overall did not change significantly during the televised debate. Thus, by focusing only on temporal spikes in directly topically relevant messages—such as those identified by the hashtag #GOPDebate—we can identify underlying events driving the Twitter conversation (Jungherr, 2015; Jungherr and Jürgens, 2013; Jungherr and Jürgens, 2014a; Shamma, Kennedy, and Churchill, 2011). This pattern conforms with empirical findings focusing on the use of Twitter during televised candidate debates (Hanna et al., 2011; Jungherr, 2015). The pattern becomes even clearer once we focus on hourly counts of published messages.

For this analysis, you first have to export hourly aggregates of tweets and hourly mentions of #GOPDebate. Once you have done this, load your data sets containing the hourly counts of messages and hashtags. Point R to the directory you stored your data and load the respective data set:

```
setwd("/path_of_directory_you_saved_your_data_to")
message_counts_hourly_df<-data.frame(read.csv("tagcounts-hours.csv"))
```

As before, you have to tell R which column contains the necessary information on time:

```
dates_hourly<-as.POSIXct(strptime(message_counts_hourly_df[,1],"%Y-%m-%d %k")))
```

Here, the `as.POSIXct` command contains additional information. The command `strptime` contains information on the format time information or saved in the respective column. For more information on the command see the online documentation[69] or Kabacoff (2015). From here on, you follow the same procedure as with the previous

[69] https://stat.ethz.ch/R-manual/R-devel/library/base/html/strptime.html

time series documenting the counts of all messages and the mentions of the topically relevant hashtags.

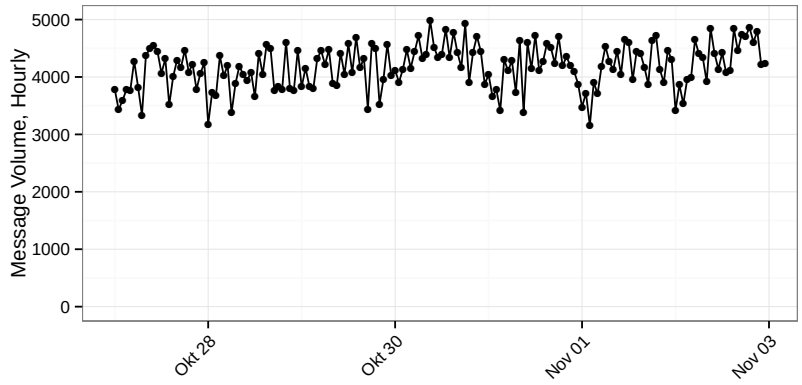


Figure 3: All Messages, Hourly Aggregates

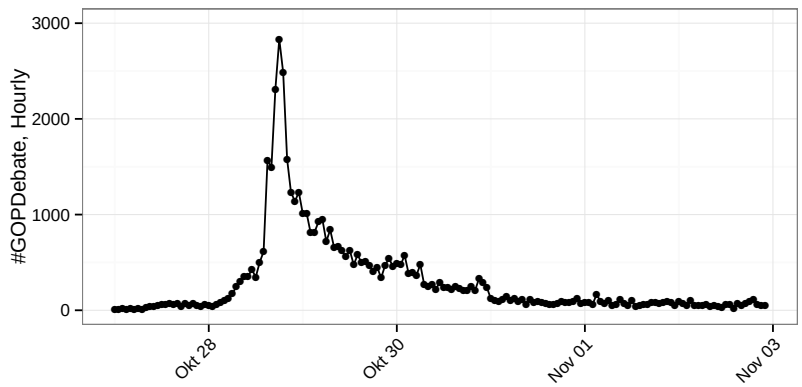


Figure 4: All Messages, Hourly Aggregates

Examining Figure 3 and Figure 4 shows that the total volume of collected messages fluctuates rather stably just above 4.000 hourly mentions. The televised debate leaves no directly identifiable deviation from this pattern. Contrast this with the plot documenting the mentions of the hashtag #GOPDebate. Here, we see the message volume increase steadily in the direct run-up to the debate at 5 p.m. SMT on September 28. During the debate itself, the volume spikes strongly. For the following day, we still see strong activity but well below the absolute spike experienced during the debate itself and the hours directly following it. Most likely, this increased volume reacts to the debate post-coverage. Two days after the debate the volume of messages using the hashtag returns to its baseline level before the debate. These patterns show that Twitter is highly reactive

to external political events and political media coverage. Given this reactivity, the analysis of the right set of topically relevant hashtags might provide significant insights into dynamics and topics of interest during political events (Jungherr, 2015; Jungherr and Jürgens, 2014b; Shamma, Kennedy, and Churchill, 2010).

Finally, let's look at another way to approach the identification of relevant televised events in Twitter messages. It has been shown that users change their behavior in reaction to important media events by posting fewer messages containing @mentions or URLs than during other time periods (Jungherr, 2015; Lin et al., 2014; Trilling, 2015). Let's see if this was also the case during this televised debate. To identify these messages, you have to call the function `export_featureless_counts`. In this function we count only messages containing no @mentions, @messages, or URLs. Let's go ahead and export the data:

```
examples.export_featureless_counts(interval="hour")
```

To analyze the resulting data adapt your R scripts used to analyze the previous hourly time series.

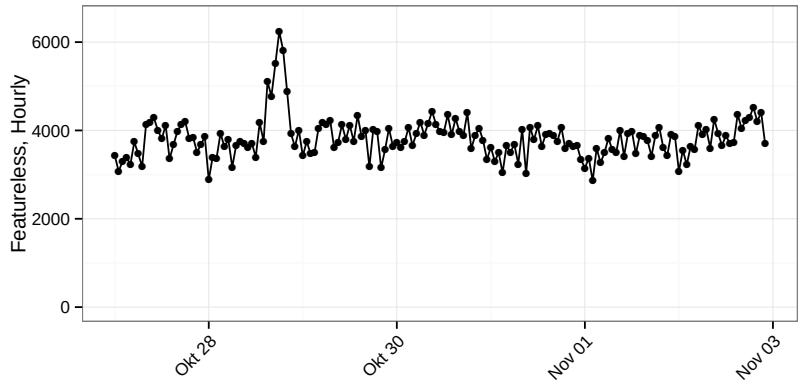


Figure 5: Messages Without @mentions, @messages, or URLs, Hourly Aggregates

Examining Figure 5, the televised debate stands out even more clearly than in mentions of the hashtag #GOPDebate. The level of messages containing neither @mentions, @messages, or URLs spikes briefly during the time of the debate but returns more quickly to its baseline level than mentions of the hashtag #GOPDebate. This illustrates the potential of focusing on messages like this to identify moments of concentrated collective attention on events outside of Twitter—such as media events.

These analyses are only the most elementary ways, you can ap-

proach time series based on message counts or hashtag mentions. These examples are designed to help you to get started with your own analyses. Make sure to dive deeper into more advanced approaches to the analysis of time series (Shumway and Stoffer, 2011). Also, there is a lot of room for improvement in the plotting of the data that you should check out by examining closer the options provided to you by `ggplot2` (Chang, 2013; Wickham, 2009).

### *Time Series of Messages Mentioning Candidates*

A very popular approach in the identification of relative attention paid to candidates during campaigns or political media events—such as televised debates—is the calculation of relative mention counts of politicians. For this, you first have to decide which type of mentions you are interested in. In the previous section, we focused on hashtag mentions. Here, we will focus on the mentions of candidates' Twitter accounts. Alternatively, you could use our function `export_hashtag_counts()` or `export_keyword_counts()` to identify mentions of candidates in hashtags or keywords. In this example, we will count mentions of the accounts of ten candidates who participated in the debate. These were: Jeb Bush, Ben Carson, Chris Christie, Ted Cruz, Carly Fiorina, Mike Huckabee, John Kasich, Rand Paul, Marco Rubio, and Donald Trump.

As before, your first step has to be the identification of candidate mentions in your database. You can do so by simply running the function as specified in our example script:

```
examples.export_mention_counts(interval="hour")
```

After a while, you will find a .csv file in your working directory containing the account mentions of the candidates listed above in hourly aggregates. When running the function it pays to remember that Twitter's API delivers mentions of users' account names already identified in the subfield `user_mentions` in the field `entities`[70]. Of course, you can always run your own character recognition on the text body of a tweet but for the purposes of this tutorial, we rely on the information returned by Twitter's API. Again, we have to decide how to handle variations in the spelling of usernames with regard to upper and lower cases. Here, it is important to remember that Twitter does not allow usernames to vary only by case. This means that while the name *realDonaldTrump* is in use, nobody can register the account *REALDonaldTrump*. For his reason, our function performs username matches completely in lower case. However, usernames can be freed (say, if an account is deleted) and assigned to a new

[70] https://dev.twitter.com/overview/api/entities

user—so be careful when extracting data over long time spans.

After exporting the relevant data, it is time again to start your copy of R or RStudio. Load the same packages as before— `ggplot2` and `scales`. In this step of the tutorial, we will show you how to plot multiple time series in one plot. For this, we have to transform the data in R. This can easily be done by using the package `reshape2`[71] (Wickham, 2007). So, please install and load it. Now, it's time to tell R your working directory of choice and load your data.

[71] https://cran.r-project.org/web/packages/reshape2/index.html

```
setwd("/path_of_directory_you_saved_your_data_to")
message_counts_hourly_df<-data.frame(read.csv("name_of_your_data_file.csv"))
```

To plot the time series of the ten Republican candidates who participated in the debate, you first need to create vectors in R containing their accounts' mention counts per hour:

```
JebBush<-message_counts_hourly_df$jebbush
BenCarson<-message_counts_hourly_df$realbencarson
ChrisChristie<-message_counts_hourly_df$chrischristie
TedCruz<-message_counts_hourly_df$tedcruz
CarlyFiorina<-message_counts_hourly_df$carlyfiorina
MikeHuckabee<-message_counts_hourly_df$govmikehuckabee
JohnKasich<-message_counts_hourly_df$johnkasich
RandPaul<-message_counts_hourly_df$randpaul
MarcoRubio<-message_counts_hourly_df$marcorubio
DonaldTrump<-message_counts_hourly_df$realdonaldtrump
```

Now it's time again to load the information on time and transform it into a format R understands as containing dates and time:

```
dates_hourly<-message_counts_hourly_df[,1]
dates_hourly<-as.POSIXct(strptime(dates_hourly,"%Y-%m-%d %k"))
```

To plot the data, you have to create a data frame containing the relevant vectors:

```
plot_all_candidate_mentions_hourly_df<-data.frame(JebBush,
    BenCarson,
    ChrisChristie,
    TedCruz,
    CarlyFiorina,
    MikeHuckabee,
    JohnKasich,
```

```
    RandPaul,
    MarcoRubio,
    DonaldTrump)
```

Now, you are ready to plot the data. To do this, first switch to your plot directory:

```
setwd("/path_of_directory_you_saved_your_data_to/_plots")
```

For plotting the data, we have to transform your data frame. This can be easily done using the `melt` command in the `reshape2` package:

```
plot_all_candidate_mentions_hourly_melted<-
    melt(plot_all_candidate_mentions_hourly_df)
```

Now, have a look at the resulting data frame to understand what you just did:

```
head(plot_all_candidate_mentions_hourly_melted)
```

It's time for a few final preparations. Such as telling R about the required date format and meaningful labels containing the full names of the candidates:

```
plot_all_candidate_mentions_hourly_melted_df<-data.frame(
    date=as.POSIXct(strptime(dates_hourly,"%Y-%m-%d %H")),
    names=factor(plot_all_candidate_mentions_hourly_melted$variable,
        levels=c("JebBush",
            "BenCarson",
            "ChrisChristie",
            "TedCruz",
            "CarlyFiorina",
            "MikeHuckabee",
            "JohnKasich",
            "RandPaul",
            "MarcoRubio",
            "DonaldTrump"), ordered=TRUE,
        labels=c("Jeb Bush",
            "Ben Carson",
            "Chris Christie",
            "Ted Cruz",
            "Carly Fiorina",
            "Mike Huckabee",
```

```
            "John Kasich",
            "Rand Paul",
            "Marco Rubio",
            "Donald Trump")),
    mentions_count=plot_all_candidate_mentions_hourly_melted$value
)
```

A final piece of information is needed before we can plot the data. Let's create a an object containing the information on the date and starting hour of the debate. This will allow us to mark that date in the resulting plots:

```
debate_start=as.POSIXct(strptime(c("2015-10-28 17"),"%Y-%m-%d %H"))
```

Now, let's plot the time series of the mentions of candidates participating in the debate:

```
plot_all_candidate_mentions_hourly<-ggplot(
    plot_all_candidate_mentions_hourly_melted_df,aes(x=date,y=mentions_count,group=names))+
        geom_line() +
        theme_bw()+
        facet_wrap(~names, nrow=5) +
        theme(axis.text.x=element_text(angle=45,hjust=1))+
        xlab("") +
        scale_x_datetime(
            breaks=date_breaks("1 day"),
            minor_breaks=date_breaks("8 hours"),
            labels=date_format("%m/%e", tz="CET"))+
        geom_vline(xintercept=as.numeric(as.POSIXct(debate_start)),linetype = 2)+
        ylab("Mention Volume, Hourly")
```

Call the object `plot_all_candidate_mentions_hourly` to check your results. You now should see a similar plot to the one presented here.

Figure 6 shows the mentions of the Twitter accounts of all candidates participating in the debate. The start of the debate on September 28 at 5 p.m. MST is marked by a dashed line. Mentions of Donald Trump dominate the plot, as he is mentioned much more frequently than the other candidates. Besides the total dominance of Donald Trump in the attention of Twitter users, another interesting question is whether candidate mentions spike in reaction to the debate. This pattern—if at all present—is somewhat obscured the scales in all plots are automatically adjusted for the highest maximum

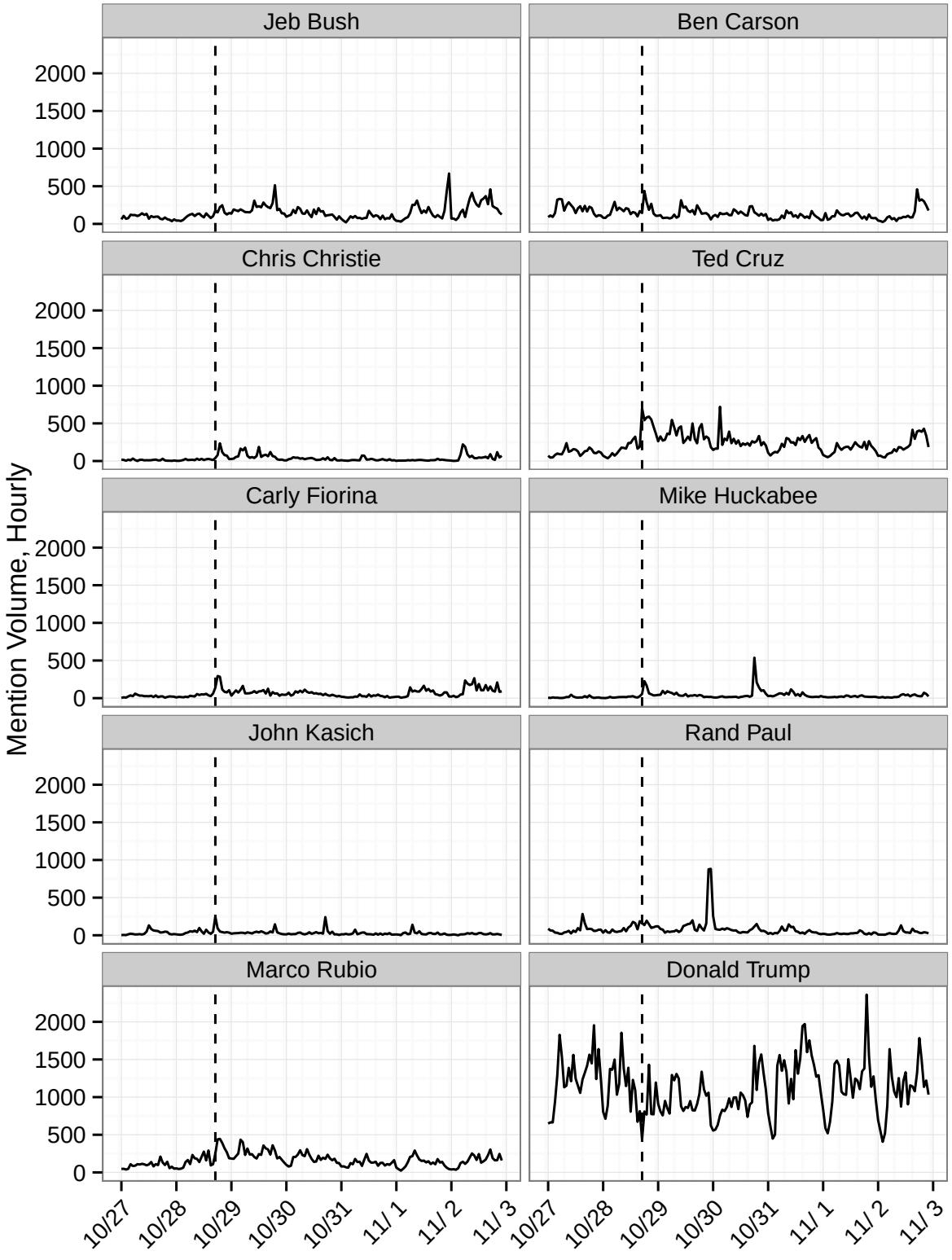


Figure 6: Candidate Mentions, Hourly Aggregates

value. In our case, this is the mention volume of Donald Trump.

A simple command allows you to plot candidate mentions more appropriately for each volume level: `facet_wrap(~names, nrow=5, scales = "free_y")`. Here is the adjusted code:

```
plot_all_candidate_mentions_hourly_scale_free<-ggplot(
    plot_all_candidate_mentions_hourly_melted_df,aes(x=date,y=mentions_count,group=names))+
        geom_line() +
        theme_bw()+
        facet_wrap(~names, nrow=5, scales = "free_y") +
        theme(axis.text.x=element_text(angle=45,hjust=1))+
        xlab("") +
        scale_x_datetime(
            breaks=date_breaks("1 day"),
            minor_breaks=date_breaks("8 hours"),
            labels=date_format("%m/%e", tz="CET"))+
        geom_vline(xintercept=as.numeric(as.POSIXct(debate_start)),linetype = 2)+
        ylab("Mention Volume, Hourly")
```

Figure 7 is somewhat misleading with regard to the overall mention levels but it allows us to assess if the debate led to an increase in mention volume of each participating candidate. We see indeed that the debate led to an increase in mention volume for nearly all candidates but it did so with varying intensity.

These examples allow you to ease into the analysis of time series of Twitter-based metrics. The examples focused on visualizing simple mention counts. In this, we only scratched the surface of the analytical potential provided to you by the data. Make sure to explore further analytical options based on your specific research interests.

*Further Reading:*

For a helpful introduction to time series analysis in the social sciences see Box-Steffensmeier et al. (2014). For a more general introduction see Shumway and Stoffer (2011). Also, Rob J. Hyndman and George Athanasopoulos offer and continuously update an excellent introductory textbook on the use of time series based forecasting online[72] (Hyndman and Athanasopoulos, 2013).

[72] https://www.otexts.org/book/fpp

Various software packages help in the analysis of time series in R and python. Beyond the obvious choices, a series of software packages helps for more exotic analyses. In the past, we found the following packages helpful for Twitter-based research: anomalous-acm[73] (Hyndman, Wang, and Laptev, 2015), AnomalyDetection[74], and CausalImpact[75] (Brodersen et al., 2015).

[73] https://github.com/robjhyndman/anomalous-acm

[74] https://github.com/twitter/AnomalyDetection

[75] https://google.github.io/CausalImpact/CausalImpact.html

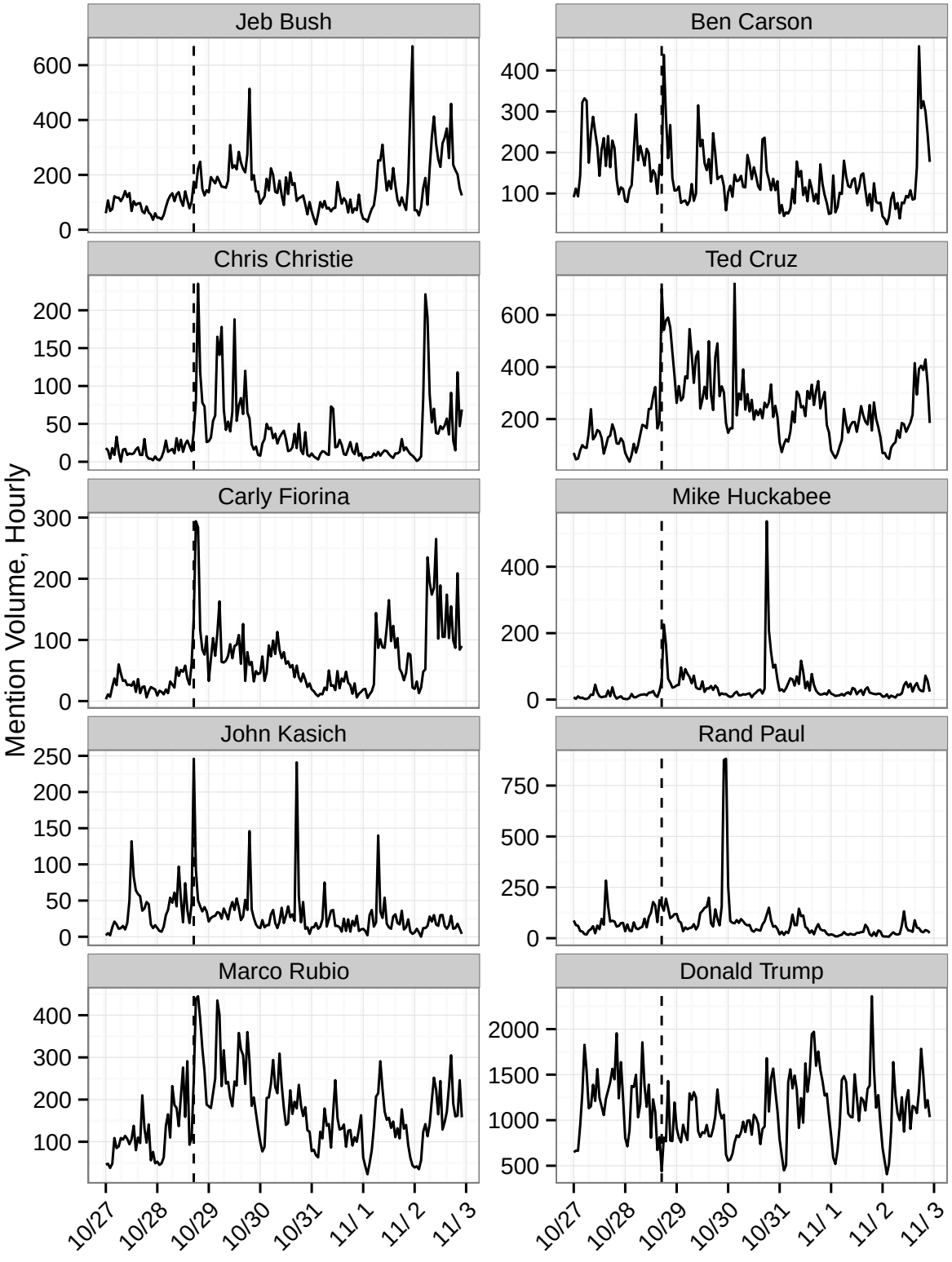


Figure 7: Candidate Mentions, Hourly Aggregates (Free Scales)

### *Network Analysis*

The analysis of networks has become one of the main areas of computational social science (Cioffi-Revilla, 2014; Howison, Wiggins, and Crowston, 2011). So it is not surprising that network analysis has become a prominent approach in the analysis of Twitter-based data. Various aspects of data collected on Twitter make them a natural fit for network analysis. You can construct networks between people based on their public interactions on Twitter, such as @replies, @mentions, or retweets (Conover et al., 2011; Jürgens, Jungherr, and Schoen, 2011). Or you can use the publicly available information on the accounts Twitter users follow and in turn are followed by others (Barberá, 2015). Alternatively, you could construct networks based on the shared use of hashtags or hyperlinks. While these approaches generate networks based on specific interactions, it is also possible to construct networks of networks, thereby combining information gained across various metrics (Bennett, Segerberg, and Walker, 2014; González-Bailón and Wang, 2016). There is also a choice to be made whether you want to look at static networks, based on interactions aggregated over a specific amount of time, or take a perspective focusing on dynamic changes between network slices over time.

Here, we will only focus on the most basic approach to network analysis with Twitter-based data. The analysis of network structures arising from the interactions between users through @replies or retweets. Still, make sure to examine papers using more advanced approaches to see if they are more appropriate to your research interests.

We can distinguish between three types of interactions between users: (1) a Tweet can be a retweet, that is, it references some other tweet while potentially adding new text; (2) it can also mention users by using their username prefixed with the @-character; (3) finally, if a username appears at the beginning of a tweet, it is marked as a reply to the user and only those that follow both sender and receiver will see it. Determining which (potentially multiple) of these features apply is as simple as checking the respective fields in the `json` objects representing tweets: `retweeted_status`, `entities["user_mentions"]`, and `in_reply_to_user_id`.

For further information, you should consult Twitter's API documentation[76]. The precise differences between the visibility of mentions and replies are explained here[77]. While networks can be composed from many different features, we will focus on the network of retweets and of replies since they are easy to extract and to interpret. The network of mentions can be constructed in the same manner, but is somewhat more complicated and substantially larger and as such

[76] https://dev.twitter.com/overview/api/tweets

[77] https://support.twitter.com/articles/14023

left as an exercise to the reader.

As with any form of statistical analysis, in the analysis of social networks you should be very careful to develop a meaningful theoretically driven operationalization. Digital trace data allow for the precise identification of very specific forms of interaction between members of the population documented by the specific data available to a researcher. Still, the interpretation of networks constructed based on these interactions is far from obvious (Howison, Wiggins, and Crowston, 2011). In this, digital trace data emphazise the importance of a point made by John Tukey over fifty years ago:

> "Far better an approximate answer to the right question, which is often vague, than an exact answer to the wrong question, which can always be made precise." (Tukey, 1962)

In using digital trace data for network analysis, you should, therefore, be very careful to explicitly state your theoretical frame and connect it through a meaningful operationalization with Twitter-based metrics. Without linking metrics to theoretical concepts, you will have a hard time interpreting the meaning of network metrics (Monge and Contractor, 2003). In practice, you best start your analysis by calculating and interpreting basic network characteristics first before proceeding to more advance modeling approaches.

Here, we chose the R library `igraph`[78] as basis for our examples of network analyses. The library not only provides a reliable, tested, and versatile environment for handling network data but also provides well-documented and powerful analytical primitives. It implements the fundamental building blocks from which empirical studies can be composed, documents their parameters, and lists the relevant citations. We will illustrate the use of `igraph` in the analysis of Twitter-based network data by the calculation of a small selection of elementary network metrics.

[78] http://igraph.org

We can differentiate between two important types of network metrics: Those that describe the whole network or *graph*, and those that relate to single elements. For a whole graph, one might be interested in its density—how strongly it is connected. For this, we can use a metric called *transitivity* (sometimes also called the *clustering coefficient*). This metric is defined as the probability that two neighbors of one node are also connected between themselves. Transitivity helps us to compare how connection-friendly the populations of different networks are. Another interesting aspect of graphs is their *degree distribution*. Degree represents the number of connections that one node has; it can be calculated either regardless of the edge direction or as the number of incoming (*indegree*) or outgoing (*outdegree*) edges. The degree distribution shows whether a graph is rather centralized (few

nodes with extremely high degrees, many nodes with low degrees) or rather egalitarian (nodes have roughly similar degrees).

Beyond global parameters, there are also local metrics that can be computed for either nodes or edges. *Centrality* measures are a prominent example. Using different aspects of a graph, they identify how central any given node is. *Betweenness centrality*, for example, looks at the shortest connections from any point to any other point in the network and tells us, how many of those quick connections go through one specific node. A high betweenness centrality value for a node means that anything that passes through the network would be either slower or not reach some regions at all, were that node to be removed. A close sibling, *closeness centrality*, represents how many steps are on average required to reach every other node. A high closeness centrality means that the average distance to others is very low. `igraph` can also calculate *global centrality* values for the whole graph—also see `igraph's centralize` function.

### *Extracting and Analyzing Retweet Networks*

Retweets can be seen as citations or endorsements whose primary function is information diffusion. The retweeting user deems a specific tweet as so relevant that she wishes to spread it to her followers. Retweet networks reveal a specific part of users' actual information sources, since reading a tweet is a prerequisite for retweeting. More importantly, retweet networks reflect a distribution of attention: Authors and tweets that are retweeted often receive a large share of attention, while others may not evoke much of a response.

The first step in constructing a network of retweets is to check whether the necessary data are available in the database. In the database as defined in our example script, the Tweet model has a foreign key relationship to itself through the field `retweet`. This means that tweets can be linked to each other with the retweet pointing to the original. Upon reading a tweet from `json`, we create the tweet, detect a potentially included original tweet, create it if necessary and store both. If the original tweet already exists, we only write one tweet to the database. While one could theoretically model the relationship the other way around—the original pointing to retweets—this would be less performant. This results in a data structure looking somewhat like this:

```
Original Tweet
Field: ID
Field (foreign key): User -> User object (with ID and username)
Field (foreign key): Retweet -> Tweet object (with ID, User etc.)
```

```
(...)
```

Now that you are familiar with the underlying data structure, let's look at the elements needed to create the network. At the lowest level, any network consists of nodes (also often called vertices) and edges (also often called links). The simplest form of edge merely designates some link between nodes, but omits information about the direction. An edge definition like this creates an undirected network. In our case, this would mean throwing away information, as we know the identity of retweeting users and authors of original tweets. Even more importantly, in our case, this operationalization would lead us to a questionable interpretation of the underlying network. Twitter has asymmetrical connections so we cannot treat incoming and outgoing connections as containing identical information. Hence, this tutorial always treats edges as having a direction, an origin and a target. This means in our database nodes need to have unique IDs and edges use these IDs to reference nodes as source and target. Now we could just devise any sort of ID and use that, but then we'd be hard pressed to say which node represents which Twitter user. In addition, we might want to store some additional information for each node. From this follows that we will (a) need to create and maintain some sort of mapping between IDs and the added attributes and (b) need this mapping *before* defining the edges, since the edges themselves will need to reference the nodes.

The mapping brings up an important point regarding IDs and large numbers. We will perform the analysis in R, and there are some pitfalls when working with very large numbers. The short story is that for largely historical reasons, there is a distinction between 32bit and 64bit numbers; Twitter started with IDs that fit within 32bit but are now so large they *require* 64bit. As a result there might be all sorts of places where IDs of tweets, users etc. are warped without us noticing. Unfortunately one such place are SQLite libraries. For example, importing tweets directly from a SQLite database overflows the ID integers and they might flip around to a negative number without any error warning you about this. Long story short, we re-number nodes to be on the safe side. As noted, it is important that we can link the nodes to users despite this re-ordering, so we use node attributes to store both the user ID and the username. You can have a look at our implementations for creating the mapping and re-ordering IDs in the file `network.py`.

Putting everything together, what we need is basically a list of all retweet -> original tweet combinations along with the linked objects (for getting IDs and usernames). Such a list is a perfect candidate for a SQL query using `group_by`. Starting from a query of tweets, we

first bring in the linked objects and then group by a *pair* of attributes: retweet->user->id *and* retweet->original tweet->user->id. This creates a de-duplicated list of tweets, where every combination of retweeting user and original author only appears once. The actual extraction is then as easy as iterating over the query and storing information on nodes and edges.

The last step in extracting the network is writing the network to a file. igraph supports a long list of file formats that can be read and written to. Coming from python (where we avoided igraph due to its somewhat advanced installation requirements), GraphML[79] is a well-defined, solid, XML-based file format that allows storing arbitrary attributes of nodes and edges. Python includes a XML parser library (etree)—based on that, the `network.py` file includes a very rudimentary writer for GraphML files.

[79] http://graphml.graphdrawing.org

To create a retweet network, just call the function `retweet_links` from our script `network.py`:

```
network.retweet_links()
```

Be patient, this might take a while. In the meantime, you can have a look at the helper function underlying this export:

```
write_graphml_file(nodes=nodes, edges=edges, filename="replies.graphml")
```

where `nodes` is a dictionary of nodes `{node_id: username}` and edges a list of edge tuples `(source_id, target_id)`.

Once you have created a `graphml`-file, the network can be analyzed using a wide number of applications. To get a first impression of the network structure, it is often useful to use the open source application Gephi[80]. While its analytic capabilities leave much to be desired, it is the fastest and easiest way to visualize a network. Visual representations are the result of applying one of the available layout algorithms that spread out nodes while trying to keep well-connected areas close together. A word of warning, not all layouts are deterministic. It is generally not a good idea to take the visual distance as hard evidence of similarity (there are better, formal methods for this, see below). The `ForceAtlas2` layout performs well with very large networks (this layout is also available[81] as a plug-in to R igraph, but needs to be installed manually.). `Yifan Hu` can be a good alternative, although you probably need to increase the *optimal distance* parameter to prevent overlapping. Once the network is spread out enough to reveal its structure, you can adjust the node size to reflect degree (number of total connections), indegree (number of incoming connections), as well as outdegree (number of outgoing connections).

[80] https://gephi.org

[81] https://github.com/analyxcompany/ForceAtlas2

We defined edges as links from a retweet to the original tweet. Hence large vertices—those receiving many connections—are frequently retweeted authors.

As an example for a more formal analysis of our retweet network in R, we provide an annotated example code in the file `networks.r`. After installing and loading the igraph library, network files can be loaded via the `read.graph` method by passing the corresponding format as a parameter, such as

```
rt.net <- read.graph("retweets.graphml", format="graphml")
```

The result of reading in the file is a network object which—along with the other means of working with networks—differs somewhat from the data frames that usually serve as data storage in R. Make sure to consult the igraph documentation[82] on the details. We can get some useful information about our network by using the summary function:

[82] http://igraph.org

```
summary(rt.net)
>>IGRAPH D -- 168617 291450 --
>>+ attr: label (v/c), user_id (v/c), id (v/c), id (e/c)
```

What we have here is an igraph network with roughly 170.000 users (nodes or vertices) and 290.000 retweet connections (edges). Note that the library prefers the term *vertex* over *node*. The "D" attribute signals a directed network comprising edges reaching from one node to another. The second line lists the attributes available for nodes, here called vertices, (v/c) and edges (e/c). Both have the required ID attribute, and nodes carry an attribute named `label` containing the username as well as the unique user ID as `user_id`. Should you ever need to inspect this data object or access it directly, nodes and edges are provided as a list by using `V(rt.net)` and `E(rt.net)` respectively. Attributes follow the usual dollar-notation in R:

```
V(rt.net)\$label
```

returns a list of all labels and so on. Setting new attributes works the same way. For example, setting the size of plotted vertices to 10 would be done like this:

```
V(rt.net)\$size $<$- 10
```

And adjusting the size to the number of incoming edges is as easy as:

```
V(rt.net)\$size $<$- degree(rt.net, mode="in")
```

Now that we have the network loaded and all the usual R functionality at our hands, we can slice and measure things any way we want. For example, we can easily find the most frequently retweeted users. First, let's store the indegree—the number of retweets a user received—in a node attribute named indegree

```
V(rt.net)$indegree <- degree(rt.net, mode="in")
```

Then we sort nodes by this attribute, get the labels, and print the first 10:

```
V(rt.net)[order(-indegree)]$label[1:10]
```

The result conforms with our expectations given the focus of our political data set that included media accounts. Still, the appearance of Kurdish politician Selahattin Demirtas (@hdpdemirtas[83]) and the US national women's soccer team (ussoccer_wnt[84]) are obviously due to news events unconnected with the primaries:

[83] https://twitter.com/hdpdemirtas

[84] https://twitter.com/ussoccer_wnt

```
 [1] "realDonaldTrump" "FoxNews"         "hdpdemirtas"     "DanScavino"      "ussoccer_wnt"
 [6] "BarackObama"     "DailyPresRaps"   "HillaryClinton"  "LastWeekTonight" "CNNPolitics"
```

Turning towards true network metrics, let's inspect two common indicators of vertex importance: Betweenness and closeness centrality. Both can be stored as attributes in the same way we saw above:

```
V(rt.net)$betweenness <- betweenness(rt.net)
V(rt.net)$closeness <- closeness(rt.net, mode="in")
```

Before we examine the results, let's step back for a moment and think about our expectations. Betweenness centrality measures the number of shortest paths that pass through one node. In a directed graph such as the retweet network, only those users that both receive *and* perform retweets to and from a large number of others will create such shortest paths. As such, we should expect high betweenness centrality with very active and somewhat recognized amateur users, but not from high visibility actors such as politicians and media outlets, whose role is defined by asymmetrical attention (Shirky, 2008,

p. 91). Closeness, on the other hand, measures a different aspect of centrality, namely the average distance to all other nodes. In a directed network, this can be both the distance *to* as well as *from* other nodes. The concept of outdegree closeness—the average number of retweets required to cite everyone else—does not lend itself to many meaningful interpretations. Its information flow model is flawed, because users are central if they retweet many authors that in turn retweet many further authors. The best description would be that outdegree closeness measures the breadth of a user's information sources. A much more interesting perspective is found in the opposite, indegree closeness, which measures how many retweets are on average required to bring one author's tweet to any other user. This direction mirrors the actual information flow, so we should expect high indegree closeness on accounts that are widely regarded as valuable sources.

There are a number of ways through which we can check whether these assumptions match the dataset—for example, we could look at the centrality distributions or compute aggregated values for hand-coded user types etc. For brevity's sake, we will content ourselves with a quick exploratory glance. To get a little bit more context than the information contained in the above shown top 10 list, we can extract the graph for any top N users by centrality and visualize the network. This way, we get not only the list of names but can also see the retweets between them. Assuming we already stored the centrality values on the vertices, all we need to do is to create a copy of the graph by deleting all vertices not in the top list:

```
top25.nodes <- V(rt.net)[order(-closeness)][1:25]
small.rt.net <- delete.vertices(rt.net, which(!V(rt.net) %in% top25.nodes))
l <- layout_with_fr(small.rt.net)
plot(small.rt.net, l)
```

The resulting graph indeed confirms that news outlets are amongst the most central accounts measured by indegree closeness and are at the same time the major source of retweets within the top 25 accounts.

### *Extracting and Analyzing @Reply Networks*

Replies are not the most popular Twitter feature, probably in part because of some confusion with regard to their visibility[85]. Nevertheless, apart from direct messages, they are the most focussed mode of communication. Replies differ from retweets insofar as the `json` object provided by Twitter's APIs does not contain both parts of the

[85] https://support.twitter.com/articles/14023

exchange. Instead, a reply-tweet has four fields pointing to the user (both screen_name and ID) that was addressed and potentially the ID of a specific tweet that the reply is directed at. Our tweet model reflects this structure by using a foreign key to the addressed user and a big integer field for the target tweet's ID. The rationale behind these two different approaches is that thus we able to create a user when merely knowing the ID and screen_name. In this case, using a foreign key which requires to have the target object present in the database is ok. However, we cannot use a foreign key to reference tweets that are not present in the dataset, so instead we simply store the ID. Hence our structure is:

```
Original Tweet
Field: ID
Field (foreign key): User -> User object (with ID and username)
Field (foreign key): Reply_to_User -> User object (with ID and username)
(...)
```

The hierarchy of models is less complex than that of retweets, since we can directly link from the tweet to both involved users. Apart from that, the extraction logic including the involved SQL queries is pretty much the same, and we won't repeat it here. You can generate the reply network by using the `reply_links()` function in `network.py` which will generate the appropriate GraphML file. Now, you have to load the network file into R:

```
reply.net <- read.graph("replies.graphml")
```

On this object, we can perform the same analytic steps as with the retweet network. Of course, the interpretations will differ quite a bit. Already in the `summary()` overview, we can see that the reply network is a lot smaller than that composed of retweets:

```
>>IGRAPH D -- 86057 132218 --
>>+ attr: label (v/c), user_id (v/c), id (v/c), id (e/c)
```

Moving on to the metrics, the differences in both meaning and results continue. A high indegree represents not necessarily increased reach but rather attention from many others—a definition that maps well onto famous actors. A glance at the network of the 25 most replied to users reveals that—much to our satisfaction—those accounts encompass most of the candidates and important media outlets. Interestingly enough, while there were practically no retweets

amongst this group, the candidates do address each other. Especially Clinton, Huckabee, Bush, and Trump form a connected group with Hillary Clinton at the center:

```
V(reply.net)$indegree <- degree(reply.net, mode="in")
top25.nodes <- V(reply.net)[order(-indegree)][1:25]
small.reply.net <- delete.vertices(reply.net, which(!V(reply.net) %in% top25.nodes))
l <- layout_with_fr(small.reply.net)
plot(small.reply.net, l)
```

Again, due to the directed nature of the graph, betweenness centrality is high for nodes with many mutual tweet exchanges and favors users with both high given and received attention. Closeness as measured by incoming links could be seen as a measure of how many steps a user is away from receiving the attention of some node, while the outdegree variant remains hard to make sense of.

Without going into detail, we would like to point to the network of the 25 nodes with the highest indegree closeness values. Here, a few new non-candidate, non-media accounts appear, most of which are recipients of tweets from Donald Trump but do not reciprocate. In this regard, the most prominent candidate by indegree differs markedly from his peers.

In closing, there are two very important caveats to be mentioned. For simplicity's sake, we simplified the analysis in two main regards: First, when constructing the networks, we omitted edge weights and treated any connection the same, regardless of how many tweets were its foundation. This is obviously an assumption that will not fit many concepts and, in general, many analyses will profit from incorporating communication frequency as edge weight. Second, the time frame of the data set was deliberately chosen to include a reasonable time span surrounding the debate but does not go beyond a point where entirely different topics dominate the conversations. When going beyond the example data, it is absolutely crucial that networks aggregated over a long time actually map to stable phenomena. If we were to aggregate over numerous volatile phases with entirely different dynamics, the credibility of our analysis would be seriously limited.

All in all, we could only give a very brief and exemplary overview of network analysis as a useful tool in analyzing the relational characteristics of digital trace data. We strongly recommend further exploring the recommended literature and the documentation accompanying the software packages used here.

*Further Reading:*

There are many excellent introductions to the analysis of social networks. For a general introduction to network analysis see Carrington, Scott, and Wasserman (2005) and Wasserman and Faust (1994). For a more technical approach see Kolaczyk (2009) and Newman (2010). For a detailed guide to the analysis of social networks with *R* see Kolaczyk and Csárdi (2014). For a conceptual discussion of social network analysis in the social sciences see Easley and Kleinberg (2010). For specific issues in constructing networks based on digital trace data see Howison, Wiggins, and Crowston (2011). For network visualizations see Katya Ognyanova's excellent tutorial[86].

[86] http://kateto.net/network-visualization

## *Prepare Your Own Data for Replication*

We started this tutorial by showing you how to collect data provided by other researchers for the reproduction of their analyses. It is only fitting to end this tutorial by showing you how to prepare your own data as to allow the reproduction of your analyses. At the time of this writing, editors and reviewers of scientific journals appear to be rather indifferent to demanding the publication of code or data files for the reproduction of published results based on digital trace data. This is bound to change in the near future. As we have shown, working with digital trace data entails a series of choices in the collection, preparation, and analyses of data, the consequences of which for the stability of results are not at all understood. Detailed documentation of your research approach and enabling other researchers to reproduce your data set and your analyses will, therefore, go a long way to establish trust in your results and also to allow an assessment of the consequence of your analytical choices in comparing your work with that of researchers opting for other approachers.

While we cannot support you in the documentation of your research process and code, in our script `examples.py` we provide you with the function `dehydrate`, which extracts the tweet IDs from tweets saved in your database and saves them in a `csv` file. This file then allows others to download the tweets used in your analysis through Twitter's API, as you did in the beginning of the tutorial. Here is one example how to use this function.

Let's start by downloading the tweet archive of Lawrence Lessig and save it as a .json file to your working directory:

```
cd [your working directory where you saved our code examples]
ipython
import examples
```

```
examples.save_user_archive_to_file()
```

Now, let's prepare your workspace to work with a database and load the tweets saved in the `json` file. For this, you proceed as discussed in the chapter on data preparation but make sure that you have no file called `tweets.db` already in your working directory. Once you have finished the setup of your database run the command

```
examples.import_json()
```

Now, let's export the IDs of the files contained in your database:

```
examples.dehydrate()
```

Excellent! You now should find a file named `dehydrated_tweet_ids.txt` in your working directory, containing the IDs of the tweets you collected from Lawrence Lessig's user archive. Upload this file to a data repository of your choice and you have successfully completed the first step in allowing your analyses based on this data set to be reproduced by other researchers.

# *This Is Where We Leave You*

Congratulations, you made it through our little tutorial! And yet, this can only be the beginning of your journey in finding your way working with digital trace data. We aimed to offer you a helping hand for the first stages of this journey. We hope our starter kit of scripts and functions provided you with some illustrative examples on how to approach the collection of digital trace data through Twitter's API, their preparation, and subsequent analysis. We tried to toe the line between making our code transparent and easy to use while avoiding to downplay complications inherent in the use of digital trace data for research.

While we sincerely hope this tutorial enables you to start working with digital trace data, we are very aware that we could only scratch the surface of this topic. You will have quickly realized that the examples presented here are strongly connected with our own research interests and past publications. We chose these examples for our familiarity with them and their ability to illustrate a small selection of elementary tasks in working with digital trace data. Unfortunately, our examples cannot come close to illustrating all potential analytical approaches open to you. Researchers use digital trace data in a stunning variety of analytical approaches and methods. An excellent start for gaining a more encompassing view of what is on offer for you would be scanning the papers listed in our bibliography.

Similarly, you will find that while our examples might help you in getting started with the collection of Twitter data, you will soon meet new challenges not covered by this tutorial or the code provided by us. You might meet these challenges by trying to run your code on an external server to ensure continuous data collection; by trying to develop a more sophisticated data selection process than the ones presented by us—for example, you might want to track the use of specific keywords or hashtags and then collect all messages posted by users thus identified; or by trying to track topics and users across platforms, therefore, having to develop data collection approaches appropriate for various platforms. Without a doubt, once you start thinking about your research interests, you will find the limits of

our examples and start running into these and other, more original challenges.

This goes to show that we can only help you in starting your journey. Your road ahead will be lined with a series of challenges common to all who work with this new data source and challenges unique to you and your research interests. As long as there are few standards in data collection, preparation, analysis, and the reporting of results for research using digital trace data, these challenges can not be avoided and are an integral part of the research process.

As there are few indicators that these standards will be established any time soon, we believe it is your choice how to deal with these challenges. You might follow approaches proposed by other researchers or you might develop your own solutions, deviating from practices proposed by others. All of this is fine and necessary in a nascent field. Still, for the field to evolve and to avoid the uncontrolled expansion of a perplexing undergrowth of ill-connected methods, you have to document your choices, sketch how they connect or deviate from others', and be transparent with regard to their consequences for your analysis. To be able to do so, you need at least a basic understanding of your code: How does your code interact with a service's API? How do you transform raw data in preparing them for analysis? How do you operationalize the theoretical concepts of interest to you, given the information available through a service's API? Given these choices, what do analytical metrics calculated on digital trace data truly tell you about the phenomenon of interest?

When first starting out, these demands might seem overly challenging, especially in face of the predominantly empiricistic bent of most studies using digital trace data in the analysis of social phenomena. But you will soon find that addressing these questions is the only way your work will be taking seriously in the center of social science research. And, what is more important, it is the only way you will be able to cut through the noise and truly understand your phenomenon of interest. Pointing out colorful patterns in networks constructed based on arbitrary signals identified in arbitrarily collected digital data traces might provide you with a nice paper illustrating some perceived potentials of the underlying data. But sooner or later proponents of the use of new data sources and methods have to engage with central questions and concerns of a field and show how these new data and methods actually improve our understanding. Proponents of the use of digital trace data and advanced computational methods in the social sciences are increasingly challenged to face these questions, and rightly so. We urge you to become part of this debate and not to stop short by collecting the low-hanging fruits of presenting surprising correlations, shiny plots, overly ambitious

predictions, and seemingly sophisticated models—all of which with little or no true meaning.

# *Bibliography*


[1] Grant Allen and Mike Owens. *The Definitive Guide to SQLite*. 2nd. New York, NY: Apress, 2010.

[2] Nick Anstead and Ben O'Loughlin. "The emerging Viewertariat and BBC Question Time: Television debate and real-time commenting online". In: *The International Journal of Press/Politics* 16.4 (2011), pp. 440–462. DOI: 10.1177/1940161211415519.

[3] Pablo Barberá. *streamR: Access to Twitter Streaming API via R*. CRAN, 2014. URL: https://cran.r-project.org/web/packages/streamR/index.html.

[4] Pablo Barberá. "Birds of the same feather tweet together: Bayesian ideal point estimation using Twitter data". In: *Political Analysis* 23.1 (2015), pp. 76–91. DOI: 10.1093/pan/mpu011.

[5] Pablo Barberá and Gonzalo Rivero. "Understanding the Political Representativeness of Twitter Users". In: *Social Science Computer Review* 33.6 (2015), pp. 712–729. DOI: 10.1177/0894439314558836.

[6] Marco T. Bastos, Dan Mercea, and Arthur Charpentier. "Tents, Tweets, and Events: The Interplay Between Ongoing Protests and Social Media". In: *Journal of Communication* 65.2 (2015), pp. 320–350. DOI: 10.1111/jcom.12145.

[7] W. Lance Bennett, Alexandra Segerberg, and Shawn Walker. "Organization in the crowd: peer production in large-scale networked protests". In: *Information, Communication & Society* 17.2 (2014), pp. 232–260. DOI: 10.1080/1369118X.2013.870379.

[8] Erik Borra and Bernhard Rieder. "Programmed method: developing a toolset for capturing and analyzing tweets". In: *Aslib Journal of Information Management* 66.3 (2014), pp. 262–278. DOI: 10.1108/AJIM-09-2013-0094.

[9] Janet M. Box-Steffensmeier et al. *Time Series Analysis for the Social Sciences*. New York, NY: Cambridge University Press, 2014.

[10] danah boyd and Kate Crawford. "Critical questions for Big Data: Provocations for a cultural, technological, and scholarly phenomenon". In: *Information, Communication & Society* 15.5 (2012), pp. 662–679. DOI: 10.1080/1369118X.2012.678878.

[11] Keith Bradnam and Ian Korf. *UNIX and PERL to the Rescue! A field guide for the life sciences (and other data-rich pursuits)*. Cambridge, UK: Cambridge University Press, 2012.

[12] Kay H. Brodersen et al. "Inferring causal impact using Bayesian structural time-series models". In: *Annals of Applied Statistics* 9.1 (2015), pp. 247–274. DOI: 10.1214/14-AOAS788.

[13] Axel Bruns and Yuxian Eugene Liang. "Tools and methods for capturing Twitter data during natual disasters". In: *First Monday* 17.4 (2012). URL: http://firstmonday.org/ojs/index.php/fm/article/view/3937/3193.

[14] Peter J. Carrington, John Scott, and Stanley Wasserman. *Models and Methods in Social Network Analysis*. Cambridge, UK: Cambridge University Press, 2005.

[15] Winston Chang. *R Graphics Cookbook*. Sebastopol, CA: O'Reilly Media, 2013.

[16] Claudio Cioffi-Revilla. "Computational social science". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.3 (2010), pp. 259–271. DOI: 10.1002/wics.95.

[17] Claudio Cioffi-Revilla. *Introduction to Computational Social Science: Principles and Applications*. Heidelberg, DE: Springer, 2014.

[18] William Roberts Clark and Matt Golder. "Big Data, Causal Inference, and Formal Theory: Contradictory Trends in Political Science? Introduction". In: *PS: Political Science & Politics* 48.1 (2015), pp. 65–70. DOI: 10.1017/S1049096514001759.

[19] Michael D. Conover et al. "Political polarization on Twitter". In: *ICWSM 2011: Proceedings of the 5th International AAAI Conference on Weblogs and Social Media*. Ed. by Nicolas Nicolov et al. Menlo Park, CA: Association for the Advancement of Artificial Intelligence (AAAI), 2011, pp. 89–96.

[20] Rosaria Conte et al. "Manifesto of computational social science". In: *The European Physical Journal Special Topics* 214.1 (2012), pp. 325–346. DOI: 10.1140/epjst/e2012-01697-8.

[21] Fernando Diaz et al. "Online and social media data as a flawed continuous panel survey". In: *Microsoft Research* (2014). URL: http://research.microsoft.com/en-us/projects/flawedsurvey/.

[22] Elizabeth Dubois and Heather Ford. "Qualitative Political Communication | Trace Interviews: An Actor-Centered Approach". In: *International Journal of Communication* 9 (2015), pp. 2067–2091.

[23] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge, UK et al.: Cambridge University Press, 2010.

[24] Morgan R. Frank et al. "Happiness and the patterns of life: A study of geolocated Tweets". In: *Nature Scientific Reports* 3.2625 (2013). DOI: 10.1038/srep02625.

[25] Deen Freelon. "On the interpretation of digital trace data in communication and social computing research". In: *Journal of Broadcasting & Electronic Media* 58.1 (2014), pp. 59–75. DOI: 10.1080/08838151.2013.875018.

[26] Deen Freelon. "On the cutting edge of Big Data: Digital politics research in the social computing literature". In: *Handbook of Digital Politics*. Ed. by Stephen Coleman and Deen Freelon. Northampton, MA: Edward Elgar, 2015, pp. 451–472. DOI: 10.4337/9781782548768.00038.

[27] Deen Freelon and David Karpf. "Of Big Birds and Bayonets Hybrid Twitter Interactivity in the 2012 Presidential Debates". In: *Information, Communication & Society* 18.4 (2015), pp. 390–406. DOI: 10.1080/1369118X.2014.952659.

[28] Daniel Gayo-Avello. "A meta-analysis of state-of-the-art electoral prediction from Twitter data". In: *Social Science Computer Review* 31.6 (2013), pp. 649–679. DOI: 10.1177/0894439313493979.

[29] Nigel Gilbert, ed. *Computational Social Science*. London, UK et al.: SAGE Publications, 2010.

[30] Tarleton Gillespie. "The Relevance of Algorithms". In: *Media Technologies: Essays on Communication, Materiality, and Society*. Ed. by Tarleton Gillespie, Pablo J. Boczkowski, and Kirsten A. Foot. Cambridge, MA: MIT Press, 2014, pp. 167–194. DOI: 10.7551/mitpress/9780262525374.001.0001.

[31] Sharad Goel et al. "The Structural Virality of Online Diffusion". In: *Management Science* (2015), pp. 1–17. DOI: 10.1287/mnsc.2015.2158.

[32] Scott A. Golder and Michael Macy. "Social Science with Social Media". In: *ASA Footnotes* 40.1 (2012), pp. 7–9.

[33] Scott A. Golder and Michael W. Macy. "Digital Footprints: Opportunities and Challenges for Online Social Research". In: *Annual Review of Sociology* 40 (2014), pp. 129–152. DOI: 10.1146/annurev-soc-071913-043145.

[34] Sandra González-Bailón. "Social science in the era of big data". In: *Policy & Internet* 5.2 (2013), pp. 147–160. DOI: 10.1002/1944-2866.POI328.

[35] Sandra González-Bailón and Georgios Paltoglou. "Signals of Public Opinion in Online Communication: A Comparison of Methods and Data Sources". In: *The ANNALS of the American Academy of Political and Social Science* 659.1 (2015), pp. 95–107. DOI: 10.1177/0002716215569192.

[36] Sandra González-Bailón and Ning Wang. "Networked discontent: The anatomy of protest campaigns in social media". In: *Social Networks* 44 (2016), pp. 95–104. DOI: 10.1016/j.socnet.2015.07.003.

[37] Sandra González-Bailón et al. "The dynamics of protest recruitment through an online network". In: *Nature Scientific Reports* 1.197 (2011). DOI: 10.1038/srep00197.

[38] Todd Graham et al. "Between broadcasting political messages and interacting with voters: The use of Twitter during the 2010 UK general election campaign". In: *Information, Communication & Society* 16.5 (2013), pp. 692–716. DOI: 10.1080/1369118X.2013.785581.

[39] Andy Guess, Jonathan Nagler, and Joshua A. Tucker. "Did Rubio win the last Republican debate? Here's what we learned from Twitter". In: *The Washington Post: Monkey Cage* (2015). URL: https://www.washingtonpost.com/news/monkey-cage/wp/2015/11/10/what-appened-on-twitter-during-the-last-republican-debate/.

[40] Alexander Hanna et al. "Mapping the political Twitterverse: Candidates and their followers in the Midterms". In: *ICWSM 2011: Proceedings of the 5th International AAAI Conference on Weblogs and Social Media*. Ed. by Nicolas Nicolov et al. Menlo Park, CA: Association for the Advancement of Artificial Intelligence (AAAI), 2011, pp. 510–513.

[41] Alexander Hanna et al. "Partisan alignments and political polarization online: A computational approach to understanding the French and US presidential elections". In: *PLEAD 2013: Proceedings of the 2nd Workshop Politics, Elections and Data*. Ed. by Ingmar Weber, Ana-Maria Popescu, and Marco Pennacchiotti. New York, NY: ACM, 2013, pp. 15–21. DOI: 10.1145/2508436.2508438.

[42] Derek Hansen, Ben Shneiderman, and Marc A. Smith. *Analyzing Social Media Networks with NodeXL: Insights from a Connected World*. Burlingham, MA: Morgan Kaufmann, 2010.

[43] James Howison, Andrea Wiggins, and Kevin Crowston. "Validity issues in the use of social network analysis with digital trace data". In: *Journal of the Association for Information Systems* 12.12 (2011), pp. 767–797.

[44] Bernardo A. Huberman. *The laws of the web: patterns in the ecology of information*. Cambridge, MA: The MIT Press, 2001.

[45] Mark Edward Huberty. "Can we vote with our tweet? On the perennial difficulty of election forecasting with social media". In: *International Journal of Forecasting* 31.3 (2015), pp. 992–1007. DOI: 10.1016/j.ijforecast.2014.08.005.

[46] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2013. URL: https://www.otexts.org/book/fpp.

[47] Rob J Hyndman, Earo Wang, and Nikolay Laptev. "Large-Scale Unusual Time Series Detection". In: *ICDM 2015: Proceedings of the International Conference On Data Mining Series*. 2015.

[48] Sarah J. Jackson and Brooke Foucault Welles. "Hijacking #myNYPD: Social Media Dissent and Networked Counterpublics". In: *Journal of Communication* 65.6 (2015), pp. 932–952. DOI: 10.1111/jcom.12185.

[49] Andreas Jungherr. "Tweets and votes, a special relationship: The 2009 federal election in Germany". In: *PLEAD 2013: Proceedings of the 2nd Workshop Politics, Elections and Data*. Ed. by Ingmar Weber, Ana-Maria Popescu, and Marco Pennacchiotti. New York, NY: ACM, 2013, pp. 5–14. DOI: 10.1145/2508436.2508437.

[50] Andreas Jungherr. "The logic of political coverage on Twitter: Temporal dynamics and content". In: *Journal of Communication* 64.2 (2014), pp. 239–259. DOI: 10.1111/jcom.12087.

[51] Andreas Jungherr. *Analyzing Political Communication with Digital Trace Data: The Role of Twitter Messages in Social Science Research*. Cham, CH: Springer, 2015.

[52] Andreas Jungherr. "Twitter Use in Election Campaigns: A Systematic Literature Review". In: *Journal of Information Technology & Politics* 13.1 (2016). DOI: 10.1080/19331681.2015.1132401.

[53] Andreas Jungherr and Pascal Jürgens. "Forecasting the pulse: How deviations from regular patterns in online data can identify offline phenomena". In: *Internet Research* 23.5 (2013), pp. 589–607. DOI: 10.1108/IntR-06-2012-0115.

[54] Andreas Jungherr and Pascal Jürgens. "Stuttgart's Black Thursday on Twitter: Mapping political protests with social media data". In: *Analyzing Social Media Data and Web Networks*. Ed. by Rachel Gibson, Marta Cantijoch, and Stephen Ward. New York, NY: Palgrave Macmillan, 2014, pp. 154–196.

[55] Andreas Jungherr and Pascal Jürgens. "Through a glass, darkly: Tactical support and symbolic association in Twitter messages commenting on Stuttgart 21". In: *Social Science Computer Review* 32.1 (2014), pp. 74–89. DOI: 10.1177/0894439313500022.

[56] Andreas Jungherr, Pascal Jürgens, and Harald Schoen. "Why the Pirate Party won the German election of 2009 or the trouble with predictions: A response to Tumasjan, A., Sprenger, T.O., Sander, P.G. & Welpe, I.M. "Predicting elections with Twitter: What 140 characters reveal about political sentiment"". In: *Social Science Computer Review* 30.2 (2012), pp. 229–234. DOI: 10.1177/0894439311404119.

[57] Andreas Jungherr, Harald Schoen, and Pascal Jürgens. "The mediation of politics through Twitter: An analysis of messages posted during the campaign for the German federal election 2013". In: *Journal of Computer-Mediated Communication* (2015). DOI: 10.1111/jcc4.12143.

[58] Pascal Jürgens and Andreas Jungherr. "The use of Twitter during the 2009 German national election". In: *German Politics* 24.4 (2015), pp. 469–490. DOI: 10.1080/09644008.2015.1116522.

[59] Pascal Jürgens, Andreas Jungherr, and Harald Schoen. "Small worlds with a difference: New gatekeepers and the filtering of political information on Twitter". In: *WebSci 2011: Proceedings of the 3rd International Web Science Conference*. Ed. by David

De Roure and Scott Poole. 21. New York: ACM, 2011. DOI: 10.1145/2527031.2527034.

[60] Robert I. Kabacoff. *R in Action: Data Analysis and Graphics with R*. 2nd ed. Shelter Island, NY: Manning Publications Co., 2015.

[61] Gary King. "Ensuring the data-rich future of the social sciences". In: *Science* 331.6018 (2011), pp. 719–721.

[62] Eric D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Heidelberg, DE: Springer, 2009.

[63] Eric D. Kolaczyk and Gábor Csárdi. *Statistical Analysis of Network Data with R*. Heidelberg, DE: Springer, 2014.

[64] Jay A. Kreibich. *Using SQLite: Small. Fast. Reliable. Choose Any Three.* Sebastopol, CA: O'Reilly Media, 2010.

[65] Daniel Kreiss. "Seizing the moment: The presidential campaigns' use of Twitter during the 2012 electoral cycle". In: *New Media & Society* (2014). DOI: 10.1177/1461444814562445.

[66] Shamanth Kumar, Fred Morstatter, and Huan Liu. *Twitter Data Analytics*. New York, NY: Springer, 2014.

[67] Anders Olof Larsson and Hallvard Moe. "Studying political microblogging: Twitter users in the 2010 Swedish election campaign". In: *New Media & Society* 14.5 (2012), pp. 729–747.

[68] David Lazer et al. "Computational social science". In: *Science* 323.5915 (2009), pp. 721–723. DOI: 10.1126/science.1167742.

[69] David Lazer et al. "The Parable of Google Flu: Traps in Big Data Analysis". In: *Science* 343.6176 (2014), pp. 1203–1205. DOI: 10.1126/science.1248506.

[70] Yu-Ru Lin et al. "Voices of victory: a computational focus group framework for tracking opinion shift in real time". In: *WWW 2013: Proceedings of the 22nd international conference on World wide web*. Ed. by Daniel Schwabe et al. Geneva, CH: International World Wide Web Conferences Steering Committee, 2013, pp. 737–748.

[71] Yu-Ru Lin et al. "Rising tides or rising stars? Dynamics of shared attention on Twitter during media events". In: *PLoS One* 9.5 (2014), e94093. DOI: 10.1371/journal.pone.0094093.

[72] Merja Mahrt and Michael Scharkow. "The Value of Big Data in Digital Media Research". In: *Journal of Broadcasting & Electronic Media* 57.1 (2013), pp. 20–33. DOI: 10.1080/08838151.2012.761700.

[73] Kevin Makice. *Twitter API: up and running*. Sebastopol, CA et al.: O'Reilly Media, 2009.

[74] Norman Matloff. *The Art of R Programming: A Tour of Statistical Software Design*. San Francisco, CA: No Starch Press, 2011.

[75] Viktor Mayer-Schönberger and Kenneth Cukier. *Big Data: A Revolution that Will Transform How We Live, Work, and Think*. New York, NY: Houghton Mifflin, 2013.

[76] Wes McKinney. *Python for Data Analysis*. Sebastopol, CA: O'Reilly Media, 2013.

[77] Panagiotis Takis Metaxas, Eni Mustafaraj, and Daniel Gayo-Avello. "How (not) to predict elections". In: *SocialCom 2011: The 3rd IEEE International Conference on Social Computing*. Washington, DC: IEEE, 2011, pp. 165–171. DOI: 10.1109/PASSAT/SocialCom.2011.98.

[78] Peter R. Monge and Noshir S. Contractor. *Theories of Communication Networks*. New York, NY: Oxford University Press, 2003.

[79] Fred Morstatter, Jürgen Pfeffer, and Huan Liu. "When is it Biased? Assessing the Representativeness of Twitter's Streaming API". In: *WWW 2014: Proceedings of the 23rd International Conference on World Wide Web*. Ed. by Chin-Wan Chung et al. New York, NY: ACM, 2014, pp. 555–556. DOI: 10.1145/2567948.2576952.

[80] Fred Morstatter et al. "Is the sample good enough? Comparing data from Twitter's Streaming API with Twitter's Firehose". In: *ICWSM 2013: Proceedings of the 7th international AAAI conference on weblogs and social media*. Ed. by Emre Kiciman et al. Menlo Park, CA: Association for the Advancement of Artificial Intelligence (AAAI), 2013, pp. 400–408.

[81] Simon Munzert et al. *Automated Data Collection with R: A Practical Guide to Web Scraping and Text Mining*. Chichester, West Sussex, UK: John Wiley & Sons, 2015.

[82] Seth Myers and Jure Leskovec. "The Bursty Dynamics of the Twitter Information Network". In: *WWW 2014: Proceedings of the 23rd International Conference on World Wide Web*. Ed. by Chin-Wan Chung et al. New York, NY: ACM, 2014, pp. 913–924. DOI: 10.1145/2566486.2568043.

[83] Mark E. J. Newman. *Networks: An Introduction*. Oxford, UK: Cambridge University Press, 2010.

[84] Tony Ojeda et al. *Practical Data Science Cookbook*. Birmingham, UK: PACKT Publishing, 2014.

[85] Malcolm R. Parks. "Big Data in Communication Research: Its Contents and Discontents". In: *Journal of Communication* 64.2 (2014), pp. 355–360. DOI: 10.1111/jcom.12090.

[86] Roger D. Peng, Francesca Dominici, and Scott L. Zeger. "Reproducible Epidemiologic Research". In: *American Journal of Epidemiology* 163.9 (2006), pp. 783–789. DOI: 10.1093/aje/kwj093.

[87] Rolfe Daus Peterson. "To tweet or not to tweet: Exploring the determinants of early adoption of Twitter by House members in the 111th Congress". In: *The Social Science Journal* 49.4 (2012), pp. 430–338. DOI: 10.1016/j.soscij.2012.07.002.

[88] Cornelius Puschmann and Jean Burgess. "The politics of Twitter data". In: *Twitter and Society*. Ed. by Katrin Weller et al. New York, NY: Peter Lang Publishing, 2013.

[89] R Core Team. *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing, 2015. URL: http://www.R-project.org.

[90] Bernhard Rieder. "Algorithmische Mediatoren: Software-Agenten als Gegenstand der Medienwissenschaften". In: *Medienjournal* 28.1 (2004), pp. 36–46.

[91] Richard Rogers. "Debanalizing Twitter: The transformation of an object of study". In: *WebSci 2013: Proceedings of the 5th Annual ACM Web Science Conference*. Ed. by Hugh Davis et al. New York, NY: ACM, 2013, pp. 356–365. DOI: 10.1145/2464464.2464511.

[92] Richard Rogers. *Digital Methods*. Cambridge, MA: The MIT Press, 2013.

[93] Matthew A. Russell. *Mining the Social Web*. 2nd ed. Sebastopol, CA: O'Reilly Media, 2014.

[94] Derek Ruths and Jürgen Pfeffer. "Social media for large studies of behavior". In: *Science* 346.6213 (2014), pp. 1063–1064. DOI: 10.1126/science.346.6213.1063.

[95] Toby Segaran. *Programming Collective Intelligence*. Sebastopol, CA: O'Reilly Media, 2007.

[96] Dhavan V. Shah, Joseph N. Cappella, and W. Russell Neuman. "Big Data, Digital Media, and Computational Social Science: Possibilities and Perils". In: *The ANNALS of the American Academy of Political and Social Science* 659.1 (2015), pp. 6–13. DOI: 10.1177/0002716215572084.

[97] David A. Shamma, Lyndon Kennedy, and Elizabeth F. Churchill. "Conversational shadows: Describing live media events using short messages". In: *ICWSM 2010: Proceedings of the 4rth International AAAI Conference on Weblogs and Social Media*. Ed. by Marti Hearst, William Cohen, and Samuel Gosling. Menlo Park, CA: Association for the Advancement of Artificial Intelligence (AAAI), 2010, pp. 331–334.

[98] David A. Shamma, Lyndon Kennedy, and Elizabeth F. Churchill. "Peaks and persistence: Modeling the shape of microblog conversations". In: *CSCW 2011: Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*. Ed. by Pamela Hinds et al. New York, NY: ACM, 2011, pp. 355–358. DOI: 10.1145/1958824.195887810.1145/1958824.1958878.

[99] Zed A. Shaw. *Learn Python the Hard Way: A Very Simple Introduction To The Terrifyingly Beautiful World Of Computers And Code*. 3rd ed. 2014.

[100] Clay Shirky. *Here comes everybody: the power of organizing without organizations*. New York: The Penguin Press, 2008.

[101] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. 3rd. Springer, 2011.

[102] Victoria Stodden, Friedrich Leisch, and Roger D. Peng, eds. *Implementing Reproducible Research*. Boca Raton, FL: Chapman & Hall/CRC, 2014.

[103] Markus Strohmaier and Claudia Wagner. "Computational Social Science for the World Wide Web". In: *IEEE Intelligent Systems* 29.5 (2014), pp. 84–88. DOI: 10.1109/MIS.2014.80.

[104] Paul Teetor. *R Cookbook*. Sebastopol, CA: O'Reilly Media, 2011.

[105] Yannis Theocharis et al. "Using Twitter to mobilize protest action: Online mobilization patterns and action repertoires in the Occupy Wall Street, Indignados, and Aganaktismenoi movements". In: *Information, Communication & Society* 18.2 (2015), pp. 202–220. DOI: 10.1080/1369118X.2014.948035.

[106] Damian Trilling. "Two Different Debates? Investigating the Relationship Between a Political Debate on TV and Simultaneous Comments on Twitter". In: *Social Science Computer Review* 33.3 (2015), pp. 259–276. DOI: 10.1177/0894439314537886.

[107] John W. Tukey. "The Future of Data Analysis". In: *The Annals of Mathematical Statistics* 33.1 (1962), pp. 1–67.

[108] Cristian Vaccari, Andrew Chadwick, and Ben O'Loughlin. "Dual screening the political: Media events, social media, and citizen engagement". In: *Journal of Communication* 65.6 (2015), pp. 1041–1061. DOI: 10.1111/jcom.12187.

[109] Maurice Vergeer and Liesbeth Hermans. "Campaigning on Twitter: Micro-blogging and online social networking as campaign tools in the General Elections 2010 in the Netherlands". In: *Journal of Computer-Mediated Communication* 18.4 (2013), pp. 399–419. DOI: 10.1111/jcc4.12023.

[110] Alessandro Vespignani. "Modelling dynamical processes in complex socio-technical systems". In: *Nature Physics* 8.1 (2012), pp. 32–39. DOI: 10.1038/nphys2160.

[111] Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge, UK et al.: Cambridge University Press, 1994.

[112] Hadley Wickham. "Reshaping Data with the reshape Package". In: *Journal of Statistical Software* 21.12 (2007), pp. 1–20. URL: http://www.jstatsoft.org/v21/i12/.

[113] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer, 2009.

[114] Hadley Wickham. *scales: Scale functions for Visualization*. 2015. URL: http://CRAN.R-project.org/package=scales.

[115] Thomas Zeitzoff. "Using Social Media to Measure Conflict Dynamics: An Application to the 2008-2009 Gaza Conflict". In: *The Journal of Conflict Resolution* 55.6 (2011), pp. 938–969. DOI: 10.1177/0022002711408014.

[116] Homero Gil de Zúñiga. "Citizenship, Social Media, and Big Data: Current and Future Research in the Social Sciences". In: *Social Science Computer Review* (2015). DOI: 10.1177/0894439315619589.

# *About the Authors*

## *Pascal Jürgens*

Pascal[87] is a research associate at the *Department of Mass Communication* at the *University of Mainz*, Germany. His research focuses on the diffusion of information, the fragmentation of media usage, political communication, online participation and protests, as well as computational and quantitative research methods. He has published in, among other places, the *Journal of Computer-Mediated Communictation*, *Social Science Computer Review*, and *Internet Research*. He may be reached on Twitter at @pascal[88] and per email at `pjuergen[at]uni-mainz.de`.

[87] http://www.medienkonvergenz.ifp.uni-mainz.de/team/pascal-juergens-m-a/

[88] https://twitter.com/pascal

## *Andreas Jungherr*

Andreas[89] is a research fellow at the *Chair for Political Psychology* at the *University of Mannheim*, Germany. His research focuses on the mediation of political life online, effects of the internet on political communication, and the use of digital trace data in the social sciences. He is author of the books *Analyzing Political Communication with Digital Trace Data: The Role of Twitter Messages in Social Science Research* (Springer: 2015) and *Das Internet in Wahlkämpfen: Konzepte, Wirkungen und Kampagnenfunktionen* (with Harald Schoen, Springer VS: 2013). His articles have appeared in, among other places, *Journal of Communication*, *Journal of Computer-Mediated Communication*, *Internet Research*, and *Social Science Computer Review*. He may be reached on Twitter at @ajungherr[90] and per email at `andreas.jungherr[at]gmail.com`.

[89] http://andreasjungherr.net/about/

[90] https://twitter.com/ajungherr

# *How to Cite*

You are free to use and adapt the scripts provided in our `twitterresearch` package in your research. If you do so, please cite the package by providing the following information:

Pascal Jürgens and Andreas Jungherr. 2016. *twitterresearch* [Computer software]. Retrieved from `https://github.com/trifle/twitterresearch`

If you want to cite this tutorial please provide the following information:

Pascal Jürgens and Andreas Jungherr. 2016. *A Tutorial for Using Twitter Data in the Social Sciences: Data Collection, Preparation, and Analysis*. Available at SSRN: `http://ssrn.com/abstract=2710146`