

# The Political Methodologist

NEWSLETTER OF THE POLITICAL METHODOLOGY SECTION  
AMERICAN POLITICAL SCIENCE ASSOCIATION  
VOLUME 20, NUMBER 2, SPRING 2013

## Editors:

JAKE BOWERS, UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN  
*jwbowers@illinois.edu*

WENDY K. TAM CHO, UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN  
*wendycho@illinois.edu*

BRIAN J. GAINES, UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN  
*bjgaines@illinois.edu*

## Editorial Assistant:

ASHLY ADAM TOWNSEN, UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN  
*townsen5@illinois.edu*

## Contents

### Notes from the Editors

1

### Articles

2

Thomas J. Leeper: Crowdsourcing with R and the  
MTurk API . . . . . 2

Christopher Gandrud: GitHub: A tool for social  
data set development and verification in the  
cloud . . . . . 7

John Beielor: A Tutorial on Deploying and Using  
Amazon Elastic Cloud Compute Clusters . . 16

Taylor C. Boas and F. Daniel Hidalgo: Fielding  
Complex Online Surveys using rApache and  
Qualtrics . . . . . 21

Dino P. Christenson and David M. Glick: Crowd-  
sourcing Panel Studies and Real-Time Exper-  
iments in MTurk . . . . . 27

advanced use of MTurk: content analysis and a panel survey experiment. Christopher Gandrud then explains how GitHub offers a comprehensive data storage and collaboration environment to enhance data quality and reproducible research. He explains each of these concepts thoroughly with a case study on data set creation and management. Next, John Beielor gives a tutorial on how to move our computation to the cloud using Amazon's EC2 service when our data or computation needs exceed those of everyday computing environment. He provides examples of how to initiate and run an EC2 instance and explains issues with parallel computing in R and Python in the EC2 environment. Our last two articles focus on survey data collection in the cloud. Taylor Boas and Danny Hidalgo walk us through their implementation of a survey in Brazil using rApache and Qualtrics, demonstrating how to merge the statistical capabilities of R with the useful features in survey engines. Finally, Dino Christenson and David Glick revisit the many possibilities of research using MTurk by giving us a detailed account of how they implemented a panel study using MTurk.

We are immensely grateful to our contributors in this issue. We have learned a lot and are pleased to produce this issue as a reference source for our ever-changing computing and research needs. We hope that you will find this issue spurs your research into the realm of the clouds and beyond.

*The Editors*

This is our last issue as editors of *TPM*, and, we hope, our most exciting issue yet. We have great line-up of topics that we hope will provide new tools for everyone's toolbox. Thomas Leeper unveils the programming interface behind Amazon's Mechanical Turk. He provides two examples of

## Articles

### Crowdsourcing with R and the MTurk API

Thomas J. Leeper

Aarhus University  
thosjleeper@gmail.com

Political scientists are in constant need of human-generated data. Responses to survey questions and reactions to experimental stimuli, manual coding of visual, auditory, and textual data, and the recorded behavior of individuals engaged in naturalistic or artificial behaviors are the bread and butter of contemporary quantitative research on politics. Yet access to large numbers of humans capable of producing these data is often a major logistic and financial challenge for researchers doing everything from large coding projects to pretesting of questionnaires to the recruitment of participants for full research studies. This search for humans willing to generate the kinds of data social scientists demand has led to major recent interest in online data collection generally (e.g., Iyengar 2010; Iyengar and Vavreck 2011; Vavreck and Iyengar 2011) and, more recently, crowdsourcing platforms in particular (Schmidt 2010; Chen, Menezes, and Bradley 2011) to provide these kinds of data at low cost. Among these platforms, Amazon Mechanical Turk (MTurk) stands out as one of the largest and most useful platforms for political science research (Berinsky, Huber, and Lenz 2010).

Leveraging MTurk to move traditionally pencil-and-paper processes managed locally (like laboratory experiments or the hand-coding of materials by undergraduate research assistants) into a cloud-based process can dramatically lower the time and resource expenditure involved with such efforts, as well as streamline research workflow. This article advocates for and describes how to move these social science data needs into the cloud using an R package called MTurkR (Leeper 2012), while also encouraging the development of packages that connect researchers to potentially valuable sources of API-based data. Integrating data APIs into R enables researchers to work with data in a familiar programming environment, directly link data collection and data analysis (thereby eliminating the number of steps and amount of time involved), and aid the reproducibility of research by focusing on using (and making) publicly available data that can be readily accessed from the cloud using code

that can be shared to produce identical results.<sup>1</sup>

### MTurk: Introduction and Core Concepts

MTurk is a crowdsourcing platform designed to provide “human intelligence” for tasks that cannot be readily, affordably, or feasibly automated (Amazon.com 2012). The service provides researchers with useful infrastructure for the generation of common social science data. While many early adopters of MTurk as a data generation tool came from computer science (Mason and Suri 2011; Kittur, Chi, and Suh 2008), more recent attention has emerged among social scientists (Buhrmester, Kwang, and Gosling 2011; Berinsky, Huber, and Lenz 2010; Paolacci, Chandler, and Stern 2010). Use of MTurk in general reflects a clear move toward cloud-based research, yet substantial barriers to entry for sophisticated use of MTurk exist—namely the limited functionality of the service’s online graphical Requester User Interface (RUI) for anything other than linking to off-site survey tools and the difficulty (for most non-engineers) of using MTurk’s other access points. Frankly, using MTurk for complicated social science research tasks is quite challenging and that difficulty limits the ability of researchers to innovate and test the limits of the platform for generating useful social science data.

### Key Terms and Concepts

The service connects *requesters*, who are willing to pay *workers* to perform a given task or set of tasks at a given price per task. These “Human Intelligence Tasks” (HITs), are the core element of the MTurk platform. A HIT is a task that a requester would like one or more workers to perform. Every HIT is automatically assigned a unique HITId to identify it in the system. Performance of that HIT by one worker is called an *assignment*,<sup>2</sup> such that a given worker can only complete one assignment per HIT but multiple workers can each complete an assignment for that HIT (up to a maximum set by the requester). Multiple HITs can be grouped as a HITType, allowing a worker to complete multiple similar HITs (e.g., coding of several different texts) with ease.

MTurk operates on basic market principles of supply and demand. Workers can choose which HITs to complete and how many HITs they want to complete at any given time, depending on their own time, interests, and the payments

<sup>1</sup>Another approach is the *dvn* package, which provides access to The Dataverse Network repository API (Leeper 2013) or the *ROpenSci* project, which does the same for a large number of (mostly natural science) APIs.

<sup>2</sup>HITs, Workers, Requesters, and Assignments each have a unique ID, which become essential for using MTurkR.

that requesters offer.<sup>3</sup> A requester can offer as low as \$0.005 per assignment, but if other requesters offer HITs that add up to a higher hourly wage, workers can choose to take their labor elsewhere. Similarly, requesters can pay any higher amount they want per assignment, but that may not be cost-effective given market forces. MTurk also charges a surcharge equal to 10% of all worker payments.

Once a worker completes a HIT, the requester can *review* the assignment, determining whether the responses or answers provided by the worker are satisfactory. If so, an assignment can be *approved* and the requester pays the worker the predetermined per-assignment price for the HIT (and no more or no less; the price is fixed in advance). If the requester thinks the work merits additional compensation (or perhaps if workers are rewarded for completing all HITs of a given HITType), the requester can also pay a *bonus* of any amount to the worker at any point in the future. If work is unsatisfactory, the requester can *reject* work and thereby deny payment (but has to justify that rejection to the worker), freeing the assignment for completion by another worker.

While these are the basic functions of MTurk, additional functionality is hidden (or at least inconvenient) via the RUI. In particular, while the RUI provides some ability to control what types of workers are eligible to complete a HIT (based upon their HIT approval rate, country of residence, and a few other measures) through *qualification requirements*, the functionality provided by the RUI makes it difficult to organize large numbers of workers using these qualifications as well as create new qualifications. Paying bonuses to and contacting workers is similarly quite tedious. The Requester API, by contrast, provides full access to the underlying web application that runs MTurk and MTurkR accesses that API through a familiar programming environment.

## MTurkR Package

Before using MTurk or the MTurkR package, one needs to have an MTurk requester account, which can be created at <http://www.mturk.com>, and deposit money in that account.<sup>4</sup> To use MTurkR (or any tool for accessing the Requester API), one additionally needs to retrieve Amazon Access Keys from <https://aws-portal.amazon.com/gp/aws/securityCredentials>. The *keypair* is a linked “Ac-

cess Key ID” and a “Secret Access Key” that, in combination, allow MTurkR to access the API. In MTurkR, the keypair is a two-element character vector with the Access Key ID as the first element and the Secret Access Key as the second element. This keypair is used to authenticate API requests, which are HTTP communications sent from MTurkR running on a local workstation to the MTurk server.<sup>5</sup>

MTurkR provides access to every part of the MTurk API through a set of easy-to-use, but powerful R functions that provide both simplicity for the beginning requester and robust functionality for managing everything from a single survey-type HIT with a few hundred responses to a massive HITType with large numbers of HITs, assignments, and workers. The package additionally provides an array of novel tools for managing workers (i.e., with qualifications, email notifications, and bonus payments).<sup>6</sup>

MTurkR automates the request and authentication process, such that no knowledge of HTTP requests or authentication is required to use it.<sup>7</sup> One need only provide a keypair and know the particular operation to be performed. Once performed, the MTurk service verifies that the request is valid. The API then returns an XML *response*, which MTurkR (generally) converts into R data structures that can be directly used in analysis with no need for manual conversions to R-readable data.<sup>8</sup>

As a brief example, the simplest operation is to check the balance in one’s requester account. To do so (or before doing any operation with MTurkR), the keypair should be recorded with a call to `credentials()`, which takes two character strings as its parameters: (1) an AWS Access Key ID, and (2) an AWS Secret Access Key.<sup>9</sup> A call to `AccountBalance()` then queries the API and returns a simple character string showing the remaining balance in the requester account.

## Two Political Science Use Cases

By providing direct and complete access to the API, MTurkR removes rather than imposes (as the RUI does) limitations on what researchers can crowdsource. To describe how researchers might utilize MTurk, this section provides two examples of MTurk as a social science data platform using MTurkR code to demonstrate how to easily manage that data collection. In both cases, a data need—first,

<sup>3</sup>Workers also communicate about the quality of HITs and requesters on fora such as <http://turkopticon.differenceengines.com/>, <http://mturkforum.com/>, and <http://www.turkernation.com/>.

<sup>4</sup>Note: The API does not allow you to add funds to your account, which must therefore be done through the web interface: <https://requester.mturk.com/mturk/prepurchase>.

<sup>5</sup>To move further into the cloud, one could also run MTurkR remotely through a server-based implementation of R, such as RApache (<http://rapache.net/>), RStudio Server ([http://www.rstudio.com/ide/docs/server/getting\\_started](http://www.rstudio.com/ide/docs/server/getting_started)), or Amazon’s EC2 (<http://aws.amazon.com/ec2/>).

<sup>6</sup>MTurkR also includes a lightweight graphical user interface, which will not be discussed here.

<sup>7</sup>This functionality is provided by calls to the RCurl package (Lang 2012a).

<sup>8</sup>The XML parsing is provided by the XML package (Lang 2012b). The raw XML responses are stored, by default, in a tab-separated value log file in the user’s working directory.

<sup>9</sup>The function stores this as a two-element character vector, which is then referenced by default by all other MTurkR functions.

for human coding of newspaper articles and, second, participation in a panel experiment—starts the research workflow and that need is broken down into MTurk HITs, which are completed by workers and reviewed by the requester

via MTurkR, before the completed data are extracted from MTurk for analysis immediately in R. Figure 1 lays out this basic process with solid lines representing necessary actions and dashed lines showing optional actions.

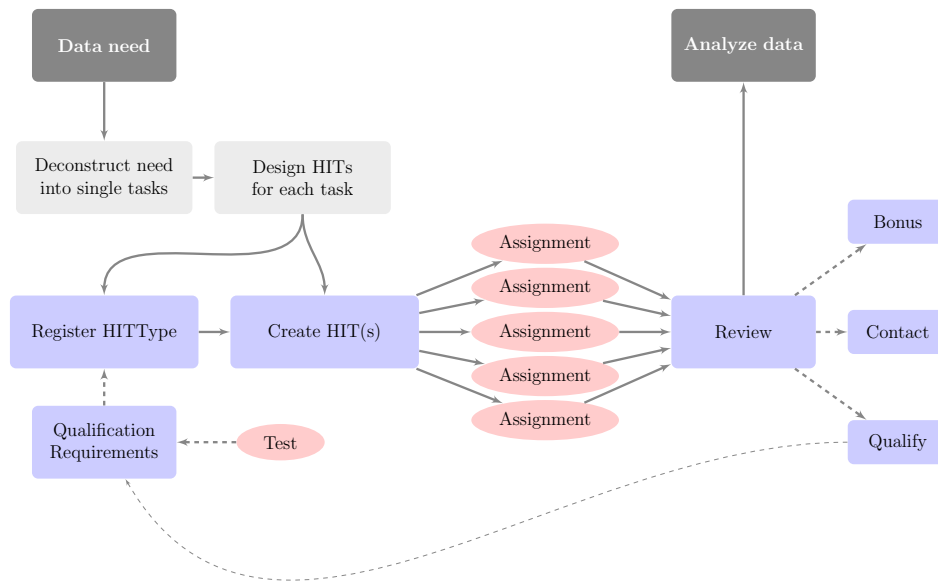


Figure 1: An MTurk Workflow

### First Use Case: Content Analysis

One of the most prevalent research techniques in political science is content analysis, be it coding of newspaper articles, images, speeches, campaign advertisements, or websites. While this is often done with an army of undergraduate research assistants or, more recently, via a number of automated techniques, crowdsourcing of human coding remains underutilized. Following the workflow described in Figure 1, a coding task consists of a need for a rectangular, item-by-attribute dataset for all of the items to be coded (items for our purposes are newspaper articles). Deconstructing this need into individual tasks, requires the construction of a coding sheet to be used on each item (a HIT will consist of coding one item), which can be written as an HTML form (or prepared in WYSIWYG form in the RUI).

To ensure quality, however, it may also be useful to restrict who can code the items to those who have demonstrated that they can accurately perform the task. Thus, in addition to designing each of the HITs, a qualification test should also be designed that will simulate the coding process and allow the requester to evaluate whether or not a particular worker qualifies to work on the coding.<sup>10</sup> Once the coding sheet and qualification test are written, implementing the coding process via MTurkR simply proceeds

left-to-right across the tasks in Figure 1.

First, a HITType is created with the qualification test to display all of the coding items together on the MTurk worker site. Then, each HIT is created by associating it with the HITType. Workers then complete the qualification test and, if successful, are eligible to complete assignments. Once assignments are completed, the requester can review those assignments (approving acceptable work and rejecting all else) and then extract the data and analyze. Workers who pass the qualification test but fail to perform well on the work can have their qualification revoked, preventing them from completing more of the assignments. The HIT-Type has four required and three optional characteristics, but good practice is to specify all of them:

- Title (required)
- Description (required)
- Reward (required)
- Duration (required)
- Keywords
- Assignment Auto-Approval Delay
- Qualification Requirements

<sup>10</sup>For technical reasons, this test needs to be written in the proprietary QuestionForm (XML) format, with the advantage being that MTurk can automatically score workers' qualification tests.

To register a HITType, these characteristics need to be defined in a call to `RegisterHITType()`. But, first, we will create a Qualification that tests workers' ability to code, along with an AnswerKey that MTurk will use to automatically score and qualify workers who complete the test.

```
## AnswerKey is a character string containing an AnswerKey
## QuestionForm is a character string containing a QuestionForm
newqual <- CreateQualificationType(name="Coding Test", description="
Test of coding ability", status="Active",
test.duration=seconds(hours=1),
test=QuestionForm,
answerkey=AnswerKey)
```

That QualificationRequirement can then be attached to a new HITType, along with the other parameters:

```
q1 <- GenerateQualificationRequirement(newqual$QualificationTypeId,
">=", 100, preview=TRUE)
register <- RegisterHITType(title="20-Question Survey",
description="Take a five-question survey about your political
opinions from researchers at Aarhus University,"
, reward=".25", duration=seconds(days=1, hours=8),
keywords="survey, question, answers, research, politics",
qual.req=q1)
```

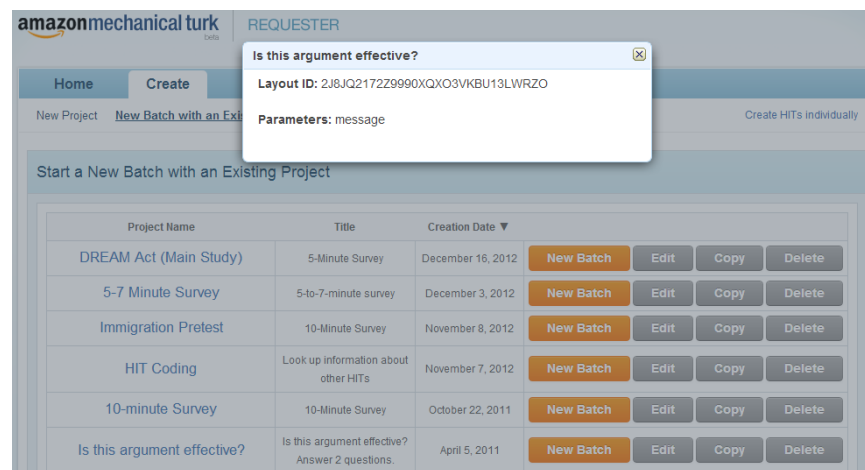


Figure 2: Retrieving a HIT LayoutId from the RUI

Creating a HIT using HITLayout parameters is also incredibly easy. The coding sheet template can be saved in the RUI with a placeholder of the style `#{message}` where the text of each article will be inserted. Once the HIT template is created in the RUI, you can retrieve its LayoutId from [https://requester.mturk.com/hit\\_templates](https://requester.mturk.com/hit_templates) by clicking on the "Project Name" on the left-hand side of the screen. A pop-up window will appear displaying the LayoutId and the name of any placeholders (see Figure 2). Those placeholders can be replaced with the relevant material when `CreateHIT()` is called, by specifying `hitlayoutid` and `hitlayoutparameters`. Assuming the newspaper texts are loaded into R in a list of character strings called `articles`, one can simply iterate through that list to create each HIT, placing the text of each article in the `message` layout parameter:

```
layout <- "2J8JQ2172Z9990XQX03VKBU13LWRZO"
hits <- vector(length=length(articles), mode="character")
for(i in 1:length(articles)){
  hits[i] <- CreateHIT(hit.type=register$HITTypeId, hitlayoutid=
layout, hitlayoutparameters=
GenerateHITLayoutParameter("message", articles[[i]]),
annotation=paste("Article_to_code", i),
expiration=seconds(hours=1),
assignments=2)$HITId}
```

This loop will return the HITId for each new HIT, which the above code stores in a character vector called `hits`. The HITId for each HIT (and the `annotation` value, which provides a private description of the HIT visible only to the requesters) can be retrieved at any time using `SearchHITs()`. Once a HIT is created, the simplest (and perhaps modal) management strategy is to simply let it run its course, with workers completing some or all of the available assignments before the HIT expires. But, it is also helpful to be able to make certain changes to HITs after they have been created

To delay the expiration of HIT (e.g., because not all assignments have been completed), `ExtendHIT(hit=hits$HITId[1], add.seconds=seconds(days=1))` extends the specified HIT(s) by the time specified in `seconds()`. If intercoder reliability appears to be low, a call to `ExtendHIT(add.assignments=1)` increases the number of available assignments for all specified HITs by one (or more) to help resolve disagreement.<sup>11</sup>

```
ExtendHIT(hit.type=register, add.assignments=1)
```

To instead expire a HIT early (e.g., because there is an anticipated problem with the HIT), simply call `ExpireHIT()` with one or more HITIds specified. At the completion

<sup>11</sup>MTurk also provides "review policy" functionality to automatically respond to agreements or disagreements between workers' coding. See MTurk documentation.



of data collection, the easiest method of reviewing workers' assignments is simply to retrieve all of the assignments and then approve them. `ApproveAllAssignments()` can be specified with either a `HITId` or a `HITTypeId`. Once a HIT and all of its assignment data are no longer needed, `DisposeHIT()` can optionally delete all data from the MTurk server.

```
a <- hits$HITId[1]
b <- GetAssignments(hit=a, return.all=TRUE)
c <- ApproveAllAssignments(hit=a)
```

## Second Use Case: Panel Experiments

The viability of MTurk as a platform for implementing social science experiments is well-understood, but the platform's comparative advantage for implementing complicated panel data collection is underappreciated, in part for technological reasons. The ability to recontact and pay large numbers of workers falls outside the functionality of the RUI. In this use case, the objective is to implement a three-wave panel experiment, where respondents are randomly assigned to a condition at *t1*, recontacted to participate in a follow-up wave with additional block-randomization at *t2*, and finally a second follow-up wave at *t3*. In contrast with the first use case, the first stages of the workflow here are relatively easy. First, an experimental survey instrument is constructed with any tool (e.g., Qualtrics or one's own HTML). Second, a single HIT is created (without a qualification test) to which workers respond by completing the survey-experiment. Note that when only one HIT is needed, the parameters normally assigned with `RegisterHITType()` can be specified in the `CreateHIT()` command.

```
newhit <- CreateHIT(question=GenerateExternalQuestion
  ("http://www.test.com/surveylink", 400)$string,
  title="20-Question Survey",
  description="Take a five-question survey about
    your political opinions.",
  reward=".25", duration=seconds(hours=4),
  expiration=seconds(days=7),
  assignments=1000,
  keywords="survey, question, answers, research,
    politics, opinion",
  auto.approval.delay=seconds(days=15),
  qual.req=GenerateQualificationRequirement
    ("Location","=", "US", preview=TRUE))
```

Once a sufficient number of responses are collected (i.e., assignments completed)—this can be checked with `HITStatus(hit=newhit$HITId)`, assignments can be reviewed such that only those who pass an attention check are approved and the remainder rejected:

```
review <- GetAssignments(hit=newhit$HITId)
correctcheck <- "7"
approve <- approve(assignments=review$AssignmentId[review$check
  ==correct])
reject <- reject(assignments=review$AssignmentId[!review$check
  ==correct])
```

After reviewing these assignments, MTurkR is leveraged via the three tasks at the right side of Figure 1: Bonus, Contact, and Qualify. To implement a panel, workers who completed the original HIT (the *t1* survey) are randomized to receive either a Democratic or Republican message in the *t2* survey, with separate links for each condition. These workers are then contacted to complete the *t2* survey and are paid a bonus if they complete it.

```
random <- rbinom(dim(approve)[1], 1, .5)
b1 <- "If you complete a follow-up survey, you will receive a $.50
  bonus.\n
  You can complete the survey at the following link:
  http://www.test.com/link1?WorkerId="
w1 <- approve$WorkerId[random==1]
t2cond1 <- ContactWorkers(subjects="Complete follow-up survey for
  $.50 bonus", msgs=apply(c1,FUN=function(worker)
  paste(b1,worker)), workers=w1)
b2 <- "If you complete a follow-up survey, you will receive a $.50
  bonus.\n
  You can complete the survey at the following link:
  http://www.test.com/link2?WorkerId="
w2 <- approve$WorkerId[random==2]
t2cond2 <- ContactWorkers(subjects="Complete follow-up survey for
  $.50 bonus",
  msgs=apply(c2,FUN=function(worker) paste(b2,worker)),
  workers=w2)
```

The process could be repeated for the *t3* survey. Paying bonuses for completing the *t2* and *t3* surveys is similar. The below example shows paying a single bonus, but replacing the single character strings with vectors of character strings allows multiple bonuses to be paid with one called to `GrantBonus()`.

```
bonus <- GrantBonus(workers="AZ3456EXAMPLE",
  assignments="123RVWYBAZW00EXAMPLE456RVWYBAZW00EXAMPLE",
  amounts="1.00", reasons="Thanks for completing the
  follow-up survey!")
```

Workers can optionally be granted qualifications, e.g., to allow those who respond to all panel waves to complete a future study or, alternatively, to prevent participants in this study from completing a similar study launched in the near future bringing the process back to the left side of Figure 1. With all data collected, analysis can proceed.<sup>12</sup>

## APIs, Cloud Data, and Political Science

As these two use cases have demonstrated, crowdsourcing (via MTurk) is a powerful addition to the political scientist's toolkit. By managing cloud-based data collection directly in R, this process is also relatively easy, done in a comfortable programming environment, and scientifically reproducible. But using MTurkR to connect with the MTurk API also shows that R can be an effective and easy-to-use way to work with cloud-based data. Currently, R has few packages that provide easy access to data APIs. One, `twitterR` provides access to Twitter data. Alone it may not be immediately useful, but when used in combination with the U.S. federal government's [Social Media Registry API](#), it should be relatively easy to track the Twitter activity of numerous government

<sup>12</sup>If all three panel waves were created as HITs, qualifications would restrict the *t2* and *t3* surveys to workers who had completed *t1* and the data could also be directly extracted from MTurk via MTurkR rather than having to be pooled from another survey tool.

agencies. A second R package, **RWeather** provides access to current weather from NOAA weather stations; a more useful package would tap the **wunderground.com** API, which is capable of providing robust, geocoded weather data going back decades. An enormous amount of data is also available from other APIs, such as those from **Bitly** to track social sharing of links and **YouTube Analytics API** to track liking, sharing, and commenting on YouTube videos, **The World Bank**, **U.S. Census Bureau**, the **Federal Register**, **OpenCongress** and **GovTrack**, several **Washington Post** APIs that track political contributions and White House visitors, and eventually the new **Congress.gov**, which aims to be a central repository for Congressional data available via APIs. Developing software that enables researchers to readily tap into these immense sources of data is an important task for political methodologists, who can develop tools that enable non-programmers to easily access these data.

## References

- Amazon.com. 2012. "Amazon Mechanical Turk Getting Started Guide".
- Berinsky, Adam J., Gregory A. Huber, and Gabriel S. Lenz. 2010. "Using Mechanical Turk as a Subject Recruitment Tool for Experimental Research." Unpublished paper, Massachusetts Institution of Technology.
- Buhrmester, Michael, Tracy Kwang, and Samuel D. Gosling. 2011. "Amazons Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data?" *Perspectives on Psychological Science* 6(1): 3–5.
- Chen, Jenny J., Natala J. Menezes, and Adam D. Bradley. 2011. "Opportunities for Crowdsourcing Research on Amazon Mechanical Turk." Unpublished paper, Amazon.com.
- Iyengar, Shanto. 2010. "Experimental Designs for Political Communication Research: From Shopping Malls to the Internet." In *Sourcebook for Political Communication Research: Methods, Measures, and Analytical Techniques*, eds. Erik Page Bucy, and R. Lance Holbert. New York: Routledge.
- Iyengar, Shanto, and Lynn Vavreck. 2011. "Online Panels and the Future of Political Communication Research." In *Handbook of Political Communication Research*, eds. Holli A. Semetko, and Margaret Scammell. New York: Sage Publications, 225–240.
- Kittur, Aniket, Ed H. Chi, and Bongwon Suh. 2008. "Crowdsourcing User Studies with Mechanical Turk." Paper presented at the CHI 2008, New York, New York, USA.
- Lang, Dustin Temple. 2012a. "RCurl: General network (HTTP/FTP/...) client interface for R." R package version 1.91–1.1.
- Lang, Dustin Temple. 2012b. "XML: Tools for parsing and generating XML within R and S-Plus." R package version 3.9–4.1.
- Leeper, Thomas J. 2012. "MTurkR: Access to Amazon Mechanical Turk Requester API." R package version 0.1.
- Mason, Winter, and Siddharth Suri. 2011. "Conducting Behavioral Research on Amazons Mechanical Turk." Unpublished paper, Yahoo! Research. <http://www.ncbi.nlm.nih.gov/pubmed/21717266>
- Paolacci, Gabriele, Jesse Chandler, and Leonard N. Stern. 2010. "Running Experiments on Amazon Mechanical Turk." *Judgment and Decision Making* 5(5): 411–419.
- Schmidt, Lauren A. 2010. "Crowdsourcing for Human Subjects Research." Paper presented at the CrowdConf 2010, San Francisco, CA.
- Vavreck, Lynn, and Shanto Iyengar. 2011. "The Future of Political Communication Research: Online Panels and Experimentation." In *Oxford Handbook of Public Opinion and Media Research*, eds. Robert Y. Shapiro, and Lawrence R. Jacobs. New York: Oxford University Press.

## GitHub: A tool for social data set development and verification in the cloud

**Christopher Gandrud**

Yonsei University

[christopher.gandrud@gmail.com](mailto:christopher.gandrud@gmail.com)

A new data set is created. Analyses are run. An article is published. The data set languishes. The data set's creators move on to other projects. Maybe someone else accesses the data and updates it for their own work. However,

these updates are never connected back to the original and the updates are not widely known about. So, multiple people end up wasting time making the same updates. Maybe another researcher finds a mistake in the original data set. They email the authors suggesting corrections. The original authors may or may not make the changes. If changes are made, there are no easily accessible records of them. Some researchers may even be reluctant to make their data available at all (partially) out of fear that someone may find a mistake and they will have to spend time making tedious corrections.

These are all examples of ways that the development and management of social science data sets do not take advantage of knowledge distributed in the social science community. These problems are partially caused by the data storage tools social scientists use. Despite rapid recent advances in social technologies—Twitter, Facebook, and so on—the tools many social scientists use do not make it easy, nor do they provide incentives to collaboratively develop data sets and verify their accuracy, especially for people not involved in creating the original data set.

In this brief article I show that GitHub<sup>1</sup> offers a comprehensive data storage service for social scientists creating and using original data sets. It has unique tools for *social data set development and accuracy verification*. Furthermore, GitHub fits directly into an active research workflow, particularly one that also includes R.<sup>2</sup>

GitHub was originally created and is widely used as a tool for software developers to work together on projects, especially open source projects, using the Git<sup>3</sup> version control system. It has been called the “Facebook of programmers” (Xie, 2011). Though GitHub is already often used by social scientists for statistical package development<sup>4</sup> it initially seems strange for me to suggest that this service would be useful for social scientists building and maintaining data sets. However, a software program and many social science data sets are fundamentally similar. *They are collections of text files*. GitHub is a means of storing, version controlling, and collaborating on text files. So if a social scientist’s original data is (a) in a **plain text format**, such as comma-separated values (CSV),<sup>5</sup> and (b) has accompanying variable description files also in a plain text format—e.g. TXT plain text or Markdown (MD),<sup>6</sup> they can take full advantage of GitHub’s features for social data set development and verification.<sup>7</sup>

In this article I begin by discussing the features that we want from a cloud service to store social science data: stable storage, version control, access, and collaboration. Then I examine how strong these features are in three methods widely used to store social science data. Though each method has its strengths, none of them encourage social data set development and verification.

A few brief notes before beginning: To get started with Git and GitHub see the GitHub Set Up page: <https://help.github.com/articles/set-up-git>. In this arti-

cle I give examples of how to take advantage of GitHub using their website and from the command line. However, there are also very good graphical user interface (GUI) versions of GitHub for Mac and Windows (see GitHub’s Set Up page for more information). I focus on GitHub, but you can also use other services that work with Git such as Bitbucket<sup>8</sup> for some of the same things. Nonetheless, I discuss GitHub because it has a much more comprehensive set of features and is more widely used. Finally, the tools I discuss in this article are suitable for the small to medium size data sets common in social science. Much larger data sets often cannot be efficiently stored in plain text files. GitHub is not a suitable storage service for very large data sets.

## What do we need from a cloud data storage service?

A cloud storage service for actively developed and used social science data needs to enable at least four things: *stable storage, version control, access, and collaboration*.<sup>9</sup> Obviously a cloud storage service needs to be a stable and reliable platform for data storage. The service needs to include version control—similar to track changes in a word processing program—so that the development of the data set can be understood and researchers are able to revert to old versions (see Bowers (2011), Healy (2011) and Fredrickson, Testa and Weidmann (2011)). Data stored on it needs to be accessible both to coauthors and to others who wish to reproduce analyses (Fomel and Claerbout, 2009), verify the data sets’ accuracy, or use the data sets in new projects (Kelly, 2006; King, 1995). Another key part of access is that the storage service makes it easy to both develop and use data sets as part of researchers’ workflows. Finally, a cloud data storage service should make collaboration easy. As long as there are no confidentiality or other strong reasons to limit access to a data set, collaboration should not be limited to coauthors. It should be possible to also collaborate with non-coauthors so that the data set can be improved by taking advantage of knowledge (and motivation) distributed throughout the social science community. Ideally, social data set development and verification keeps data sets more up-to-date and more accurate.

<sup>1</sup><https://github.com/>

<sup>2</sup>GitHub can be used to store and develop entire social science research projects, not just data. However, I do not directly address these other topics here.

<sup>3</sup><http://git-scm.com/>

<sup>4</sup>For example, the *Zelig* R package (Imai, King and Lau, 2008; Owen et al., 2012) is hosted on GitHub. See <https://github.com/IQSS/Zelig>.

<sup>5</sup>All major statistical programs as well as Microsoft Excel and Apple’s Numbers can save and open files in plain text formats like CSV.

<sup>6</sup><http://daringfireball.net/projects/markdown/>

<sup>7</sup>See Bowers (2011, 3) for a discussion of the advantages of storing research files in plain text format.

<sup>8</sup><https://bitbucket.org/>

<sup>9</sup>King (2007) discussed eight “requirements for effective data sharing.” His focus was on replication data sets for published results, rather than actively developing original data sets. His requirements, nonetheless largely overlap with mine here with the exception of “authorization” and “legal protection.” I discuss these below.



## Data storage: the social science status quo

Social science data is currently often stored in three ways. It may be stored locally on a researcher's computer, on a general purpose cloud storage service such as Dropbox<sup>10</sup> or Google Cloud Drive,<sup>11</sup> or on a specialized research hosting service, notably Dataverse.<sup>12</sup> Each of these data storage methods has strengths in terms of stable storage, access, version control, and collaboration. Nonetheless each are lacking in at least one important way, and none of them promote social data set development and verification.

**Local storage.** Local storage—data sets saved on individual researchers' computers—is the least robust form of data storage currently used. It is not a very stable form of storage. Version control with Git or some other program is possible with locally stored data. However, if the version control files are lost along with the data, the data are lost. Researchers with access to the computer can easily access the data, but access by coauthors is limited. Access by non-coauthors is very cumbersome. Files must be emailed or conveyed by hand (on some portable storage medium) in response to individual requests. These files are not automatically updated with new versions of the data. Because access is limited to users of the computer the data is stored on, collaboration, especially by non-coauthors is extremely limited.

**General purpose cloud storage.** Dropbox and other general purpose cloud storage services offer much more robust storage. These services generally work by syncing files stored on individual computers with those on cloud servers. Dropbox has a basic version control system. If you are using Dropbox for free, every time you save a file that version is stored and accessible for 30 days.<sup>13</sup> In addition you could use Git with a data set stored on Dropbox. Access is usually very easy. These services can be accessed via desktop programs, mobile apps, and websites. Because files stored on individual computers are automatically synced with cloud servers, it is very easy to incorporate them into a workflow. General purpose cloud storage services also make it possible to share files and folders via URL links. Collaboration with coauthors is very easy because folders can be shared. This means that official collaborators (those given permission to make changes to the folder, i.e. write access) can easily change files in the folder and these changes are automatically synced for all users.<sup>14</sup> However non-coauthors, without write access to the shared folder cannot easily sug-

gest specific changes. They must email their suggestions to one of the researchers with write access. This is not much better than a situation with locally stored files.

**Dataverse.** Many journals—*Political Analysis* for example—require data used in articles they publish to be uploaded to a specialized research data storage service like Dataverse. Dataverse is a stable form of storage. It does have some version control features. It saves each version that is uploaded to it. Old versions are downloadable. Dataverse is easily accessible for anyone with an internet connection. But it is difficult to access data as part of a research workflow, at least using the standard version.<sup>15</sup> Unlike Dropbox, for example, there is no way to automatically update a data set on Dataverse. You have to point and click to upload data files for each version. It is difficult to access data directly from a statistical program such as Stata or R. The data needs to be downloaded by pointing and clicking and then loaded into these programs. Finally, it is difficult to collaborate using Dataverse. Changes to a data set must be uploaded to the site manually, then manually downloaded by collaborators. There are also no direct ways for non-coauthors to suggest changes.

Dataverse is very good for what it is designed to do: store a snapshot of a data set for replicating specific published results (King, 2007). It is uniquely strong among all of the storage options discussed here for providing journals with branded replication data set archives and has built in mechanisms for ensuring that a journal is legally protected and that data users have proper authorization (King, 2007, 177–179). However, it is difficult to use Dataverse to store and access data as part of an *ongoing research workflow*. In addition it does not easily enable or incentivize social data set development and verification of its accuracy.

## Data Storage on GitHub

Though admittedly requiring a steeper learning curve, GitHub meets the four criteria for data set storage very well. In this section I will demonstrate how to use GitHub for data set storage, version control, access, and collaboration. Rather than giving a complete introduction, I focus on specific important aspects of each feature in Git and GitHub. In this section I give examples from a data set stored on GitHub. It is a data set of countries' Gallagher Electoral Disproportionality index across time that I put together from data used in Gallagher (1991, updated through 2011) and Carey and Hix (2011). The data set and more informa-

<sup>10</sup><https://www.dropbox.com/>

<sup>11</sup><https://drive.google.com/>

<sup>12</sup><http://thedata.org/>

<sup>13</sup>Old versions are stored for longer with paid accounts.

<sup>14</sup>This can create problems if multiple authors are making changes to the same files at the same time. For a discussion of file conflicts see Fredrickson, Testa and Weidmann (2011).

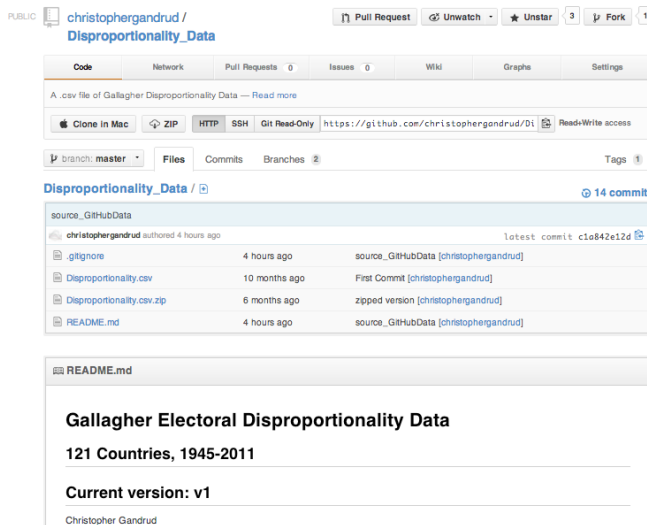
<sup>15</sup>It is possible to build a custom version of Dataverse with some R integration.

tion about it is available at: [http://christophergandrud.github.com/Disproportionality\\_Data/](http://christophergandrud.github.com/Disproportionality_Data/).

## Storing and version control

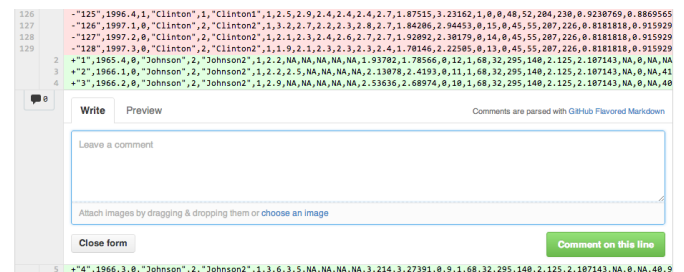
Version control is an integral part of how GitHub stores files. GitHub remotely stores files in what Git calls “repositories”, repos for short. You can think of repos as the folder containing a data set, including the data, description files, citation, copyright, and other legal information, as well as any source code used to create the data set. Repos could also include PDFs of papers where the data set was used and code for replicating the analyses in these papers. Git version controls files in repositories and GitHub hosts them remotely in the cloud.<sup>16</sup>

With a GitHub account you can store repositories for free on GitHub if they are in what GitHub calls “public repositories.” Anyone can see the contents of public repositories, including all previous versions.<sup>17</sup> Users can have an unlimited number of public repositories. There is a soft storage size limit of one gigabyte per repository. This should be more than enough for most social science data sets if they are stored in plain text formats. Each repository is given a webpage. Here is a portion of the Electoral Disproportionality repository’s main page:

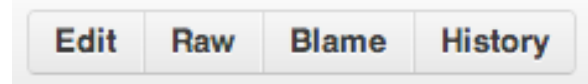


Git is a very powerful version control system, especially for text files. For these types of files, Git only saves the actual changes when you (1) **add** a file to the repository and then (2) **commit** a version of the file to the repository. This is different from the other cloud storage services we have discussed. They save the whole file for each version. With Git, each “commit” in a repo is given a unique SHA-1 number identifying it. The author of the commit is also saved.

Once you commit changes on your computer you can (3) **push** them to the remote GitHub version of the repo. A repo’s GitHub website allows you to view all of the changes that have been made to it. You can browse all versions of the text files, view highlighted changes, and comment on these changes. For example:



To see changes made to a file click on the **History** button located on the file’s GitHub webpage.



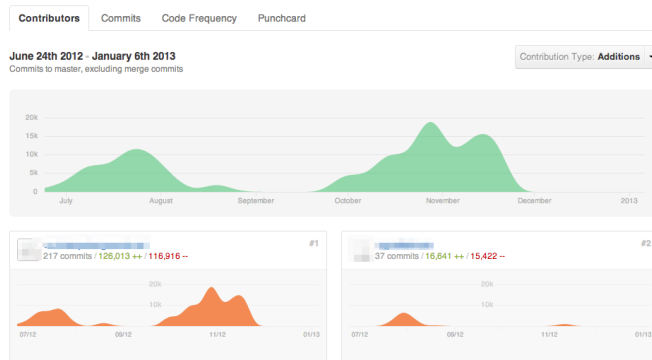
This will show the file’s entire commit history. Click an individual commit’s description to see and comment on the changes.

**Contributor analytics.** Because Git only commits changes and uniquely identifies the changer, it is possible to properly attribute every individuals’ contribution to a data set. On the GitHub website you can see every change that has been made to each line of a file and who made the change. Git calls this “blame.” To view who last changed each line of a file click on the **Blame** button on its GitHub page.

GitHub has graphical capabilities for organizing and displaying this data. One way to use these is by clicking on the **Graphs** button located near the top of each repository’s webpage. This will give you access to graphs such as the ones below (I’ve blurred the contributors’ names):

<sup>16</sup>Specifically, GitHub files are stored on Rackspace (<http://www.rackspace.com/>).

<sup>17</sup>Private repositories are available allowing only official contributors to see their contents. Private repos require a paid account.



**Binary files.** I mentioned that Git treats non-plain text—binary—files differently. Stata's DTA data format and PDFs are examples of binary file types. Rather than committing specific changes, a new version of the entire binary file is stored with each commit *if* they are changed. Binary files that are very very large can take up a lot of storage space when version controlled with Git.<sup>18</sup> This is also a problem for all of the other data storage methods with version control discussed here. For very large binary files you will need to use a totally different type of cloud storage system like Rackspace or Amazon's S3.<sup>19</sup>

If you absolutely must have very large binary files in a repository one solution to the space constraints problem is to have Git ignore them. You do this by including a text file called `.gitignore` in the repository. In `.gitignore` simply type the binary file's name. Git will not version control it. This unfortunately also means that the file will not be pushed to GitHub.

**Describe the data set with README files.** Each folder in a GitHub repository can contain a file called README.<sup>20</sup> README files for data sets can contain information about sources, variable descriptions, statements testifying that the collection and distribution of the data set violated no law, citation and copyright information, and so on. GitHub automatically displays the README file on the repository's website in full. If it is written in the Git Flavored Markdown<sup>21</sup> mark-up language and called README.md, it will also be automatically formatted. You can see part of the Disproportionality data set's README.md above in figure 1.

**Tagging versions.** Git can **tag** specific commits. Tags function as bookmarks for major versions of a Git repository. They are particularly useful for demarcating the version of a data set used in a particular publication, for ex-

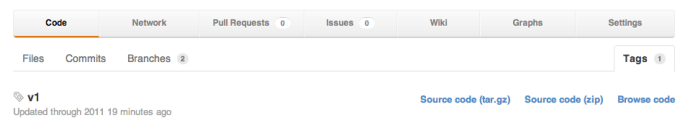
ample. Creating tags is simple. Imagine we want to tag the second major version of a data set, the one we used for a publication:

```
git tag -a v2 -m 'Citation Information'
```

The option `-a` means add, `v2` is the version number, and `-m` adds a message, in this case the publication's citation information. Now we simply **push** the tags to GitHub:

```
git push --tags
```

On GitHub there will be a list including the tag and an option to view and download this specific version of the data. For example:



## Accessing data

There are many ways to access data stored on GitHub and incorporate it into your workflow. The simplest way is to use the GitHub website to actually edit files and commit changes. This can be handy for small changes, especially from mobile devices. As I mentioned, changes can also be committed on your computer and pushed to GitHub. You can use the command line version of Git or the GUI version of GitHub to push changes. RStudio,<sup>22</sup> a program that integrates R and mark-up languages like L<sup>A</sup>T<sub>E</sub>X and Markdown, also includes Git and the ability to push a repo to GitHub. So it is possible to make changes to a repository, commit them, and push them to GitHub all within the same program. Here is a screenshot of this paper being written in RStudio. The Git functions are in the upper right pane.

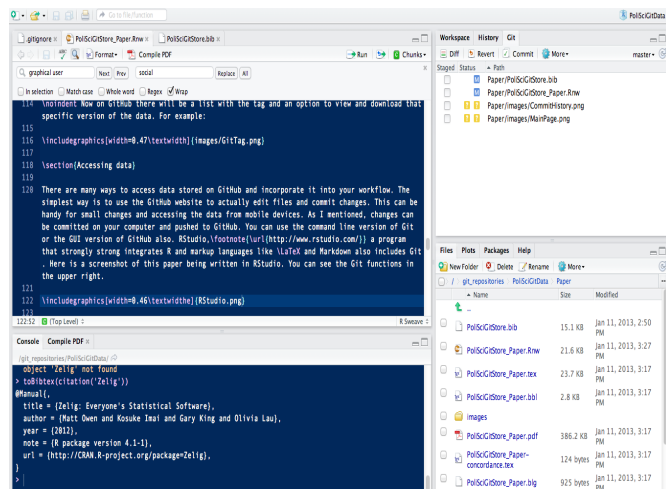
<sup>18</sup>Though it is important to note that previous commits are compressed, so the amount of storage space a committed binary file take up is less than the size of the original.

<sup>19</sup><http://aws.amazon.com/s3/>

<sup>20</sup>Actually each folder in a repository can contain one as well.

<sup>21</sup><http://github.github.com/github-flavored-markdown/>

<sup>22</sup><http://www.rstudio.com/>



**Cloning a repo.** Repositories can be downloaded in full. This is called cloning. You can do this by clicking the **Clone in . . .** button on a GitHub repository's website. Cloning can also be done in the command line using a repository's address. For example, the Disproportionality data's clone-able address is: [https://github.com/christophergandrud/Disproportionality\\_Data.git](https://github.com/christophergandrud/Disproportionality_Data.git).

```
git clone https://github.com/christophergandrud/
Disproportionality_Data.git
```

Note that in real life the address needs to be on the same line as the Git clone command. Also, before cloning a repository remember to change the working directory to the folder where you want to save it. In the Unix command line use the `cd` command to do this.

Once you have cloned the repository you can roll back to any previous version with the `checkout` command. For example if you want to roll back to a tag called "v2", type:

```
git checkout v2
```

If you have permission to make changes to the repository (see below) you can then push any changes you make back to GitHub.

**Access data directly from R.** Loading data stored on GitHub into R for use in statistical analysis is very easy.<sup>23</sup> I created a function that loads plain text formatted data from GitHub into an R data frame ready for analysis.<sup>24</sup> It's called `source_GitHubData` and is stored in a GitHub Gist<sup>25</sup> at: <https://gist.github.com/4466237>. It can be loaded into R using the `source_gist` command from the *devtools* package (Wickham and Chang, 2012).

```
# Load source_GitHubData
```

<sup>23</sup>You can of course also load data stored on the version of the repo located on your computer.

<sup>24</sup>The function is based on *devtools*' `source_url` command.

<sup>25</sup>Gists host code snippet. See: <https://gist.github.com/>.

<sup>26</sup>I used bitly (<http://bitly.com/>) to shorten the URL so that it would fit on the page more easily. The full URL is: [https://raw.githubusercontent.com/christophergandrud/Disproportionality\\_Data/master/Disproportionality.csv](https://raw.githubusercontent.com/christophergandrud/Disproportionality_Data/master/Disproportionality.csv).

<sup>27</sup>Dataverse uses a version of this standard.

```
# The functions' gist ID is 4466237
devtools::source_gist('4466237')
```

The main argument in `source_GitHubData` is `url`. You set this as the URL for the *raw* version of plain text data file you want to download. The raw version is the version with only the text file. You find this URL by clicking the **Raw** button on the file's GitHub page. It's next to the **Blame** button we saw earlier. If you use the URL for the most recent commit of the file, you will always download the most recent version whenever you use `source_GitHubData`. The raw URLs for each commit and tagged versions of the data are also accessible if you want to use a particular version.

The URL for the raw version of the Electoral Disproportionality data is <http://bit.ly/Ss6zD0>.<sup>26</sup> To download the data and put it in a data frame object called *Data* using `source_GitHubData` type:

```
# Download data
Data <- source_GitHubData(url='http://bit.ly/Ss6zD0')
```

Note that by default `source_GitHubData` loads CSV data. You can add the argument `sep = "\t"` for tab-separated (TSV) data files. You can also specify `header = TRUE` (default) or `header = FALSE`.

**Citing GitHub data.** When a researcher uses data they accessed via GitHub how can they cite it? A common practice for citing data is to cite the publication the data set was originally used in. However, this is incomplete in at least two ways. First, the version of the data used in the original publication may be different from that used later on. Second, it would be difficult to use this citation to acknowledge contributions made by contributors who did not work on the original data set, but contributed to later versions. One solution is to use the standard set by Altman and King (2007) (see also King, 2007, 183–184).<sup>27</sup> They propose that data set citations have:

- the author's name,
- the date,
- the data set's title,
- a unique global identifier (UGI),
- a universal numeric fingerprint (UNF),
- a bridge service.

The first three are self explanatory and shared with standard citations for other types of materials. Examples of UGI

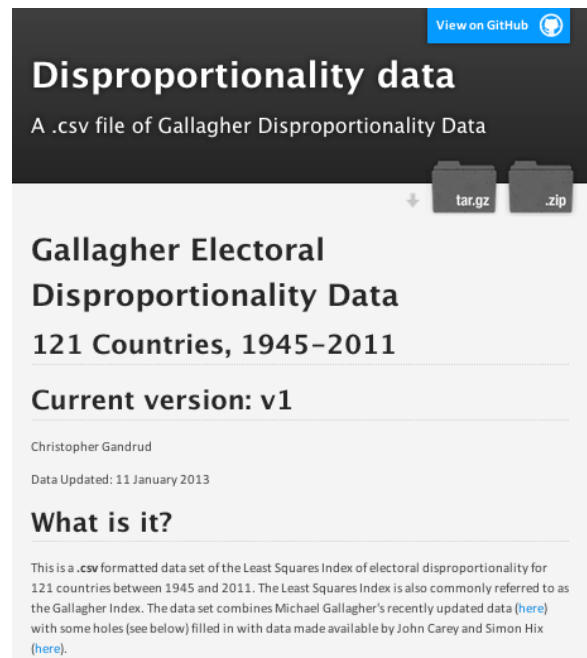
include Document Object Identifiers (DOI)<sup>28</sup> and the Handle System.<sup>29</sup> They uniquely identify the data set. UNF's uniquely number a particular version of the data set. A bridge service allows you to use the DOI and UNF to link to the actual data set. Most UGI include a bridge service and create UNFs. One place to register DOIs for free (with restrictions) is the German National Library for Science and Technology.<sup>30</sup>

You can use Altman and King's citation standards to cite specific versions—tags and commits—of a data set stored with Git on GitHub. Citation information can be displayed on the version's README.md file. This allows you to persistently cite the exact version of the data set you used to achieve a particular result. Because Git stores information about each contribution it is possible with this citation method to identify every person who has contributed to a particular version of a data set and exactly what their contribution was.

**Copyright and open data.** If you are creating an original data set that you want to and can make open—e.g. there are no confidentiality issues—while retaining authorship and enabling collaboration, it is worth considering how you want to copyright the data set.<sup>31</sup> Any copyright information can be placed in a repository's main README.md file or in a separate LICENSE.md file. This is a common practice for open source software stored on GitHub. For discussions of why and how to copyright data see Stodden (2009) and Creative Commons (2012).

**Showcase a data set with GitHub Pages.** Each public repository's GitHub website allows anyone full access to the data. However, these pages may be confusing for those without experience using a version control system. To solve this problem GitHub Pages<sup>32</sup> allows you to very easily create a simpler display webpage for the repository. These pages can be used to describe the data and include links to download the entire repository. You can of course also add links to specific files.

To create a page navigate to the repository's normal GitHub webpage. Then click **Settings** → **Automatic Page Generator**. By default it will load the README file as the content of the new page. You can change this, add a Google Analytics tracking ID<sup>33</sup> to gather information on who visits the page, and choose an aesthetic style before publishing it. Here is a sample of the Disproportionality data's page:



## Collaboration

Compared to the status quo social science data storage methods, GitHub is uniquely strong for enabling and incentivizing collaboration both between coauthors and non-coauthors who may be able to help develop and verify the data set.

**Coauthors.** Public repositories can have an unlimited number of what GitHub calls “collaborators.”<sup>34</sup> Collaborators have permission to push changes to the repository. There is one important practical issue to note. If multiple collaborators are actively working on a data set, they need to add an extra step to their GitHub commit process.<sup>35</sup> Each person needs to **pull** their collaborator's changes and resolve any merge conflicts before pushing committed changes to GitHub. Here is an example:

```
# Add new files to Git
git add .

# Commit the changes
git commit -a -m "A message"

# Pull collaborator's commits
git pull

# Push changes to GitHub
git push origin master
```

<sup>28</sup><http://www.doi.org/>

<sup>29</sup><http://www.handle.net/>

<sup>30</sup><http://datacite.org/TIB>

<sup>31</sup>Note that raw facts in a data set cannot be copyrighted, though the present layout and procedures used to gather the data can (Stodden, 2009, 39).

<sup>32</sup><http://pages.github.com/>

<sup>33</sup><http://www.google.com/analytics/>

<sup>34</sup>To add collaborators to a repository go to its webpage and click **Settings** → **Collaborators** and add the coauthors' GitHub usernames.

<sup>35</sup>This is also true if you make changes to both the GitHub version of the repository and the copy on your computer.

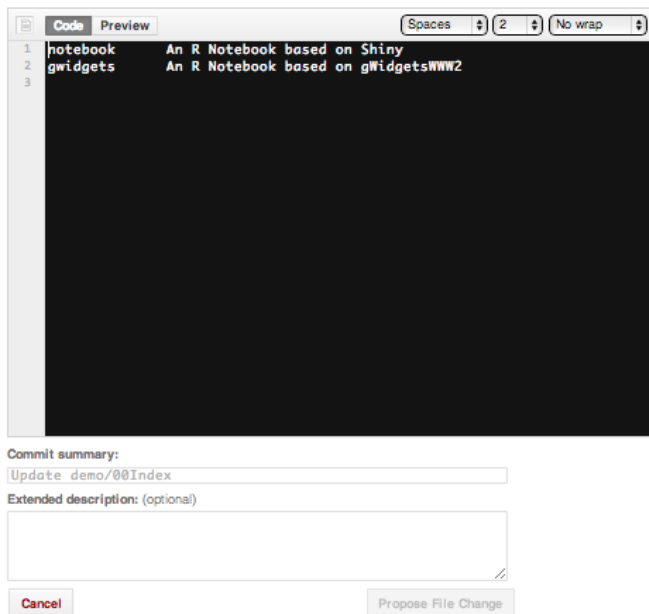


### Collaborating with non-Coauthors: Pull requests.

People who want to make changes to a repository, but are not collaborators can make “pull requests.” All they need are GitHub accounts. Pull requests are specific changes that non-collaborators suggest. Requesters can add comments about why they are suggesting a change. GitHub also has a discussion forum where anyone can discuss these comments.

After a pull request has been made it is up to the repository’s collaborators to decide whether or not to accept the requested changes. If a collaborator accepts the changes, they are made instantly and a full record of who made the changes is kept.

There are two ways to make a pull request. If someone notices a small error—e.g. a misspelling, a misclassification—or other improvement that they think should be made to a data set, they can make a pull request by navigating to the GitHub page for the specific file they think needs to be changed. Then they click the **Edit** button located next to the **Raw** button (see above). Clicking this button “forks” the repository, i.e. gives them a copy they can change. A record of the fork is created on GitHub. Requesters will then see window like this:



In this window they can make their proposed changes and add a comment about what the changes are and why they should be made before clicking **Propose File Change**.

For longer file changes, e.g. a major addition to the data set to bring it up-to-date, it is better to work with the forked repository rather than in the Edit window. To directly fork a data set’s repository click the **Fork** button on the upper right corner of its webpage. You can then make changes to the forked repository as if it were your own. When you

are ready to suggest the changes be included, you click the **Pull Request** button at the top center of the forked repository’s GitHub page. For more information on forking and pull requests see the GitHub article on the topic: <https://help.github.com/articles/using-pull-requests>.

**Issues.** GitHub repositories also have an “Issues” area that allows any other GitHub user to make a comment on the repo. These tend to be general suggestions for how to improve the files or questions about the repo that may be of general interest. Anyone can respond in the Issues area creating a discussion thread.

**Repo Wiki.** GitHub provides a very easy tool for creating nicely formatted repository wikis. For example, a data set repo’s wiki could include short articles with details about how the data set was created and what it has been used for. Non-official collaborators can add to repo wikis. Like pull requests, these changes are subject to approval by one of the repository’s collaborators.

**Follow a repository.** If you are a GitHub member you can follow a public repository, even if you aren’t a collaborator. This gives you a Facebook-style newsfeed<sup>36</sup> showing any updates made to the repository as well as discussions in the Issues area and pull requests.

**Passing on the data set.** Because of changing time commitments, professional interests, and so on, no one can maintain and update a data set forever. GitHub makes it easy to pass on control of a data set while maintaining its entire version history. Simply add the new data set maintainer as a collaborator. They will largely have the same privileges to manage the repository as you did. To completely transfer control the repositories’ owner can go to the repo’s GitHub page, click **Settings** → **Transfer**. The transferred repository will contain the entire commit history of the original repository.

**Selective incentives.** Why would people actively contribute to improving publicly available data sets, especially data sets primarily associated with other authors? What’s in it for them?

GitHub not only provides technology—particularly pull requests—for social data development and verification, but it also gives Olsonian selective incentives that can motivate people to do so (Olson, 1965). By keeping track of who made what change and providing numerous ways to quantify and visualize each member’s contributions it provides strong selective incentives to collaborate. Perhaps one day social scientists could bring GitHub’s descriptive statistics of their

<sup>36</sup>See: <https://www.facebook.com/help/327131014036297/>.

contributions, like the one below, to hiring and promotion committees.

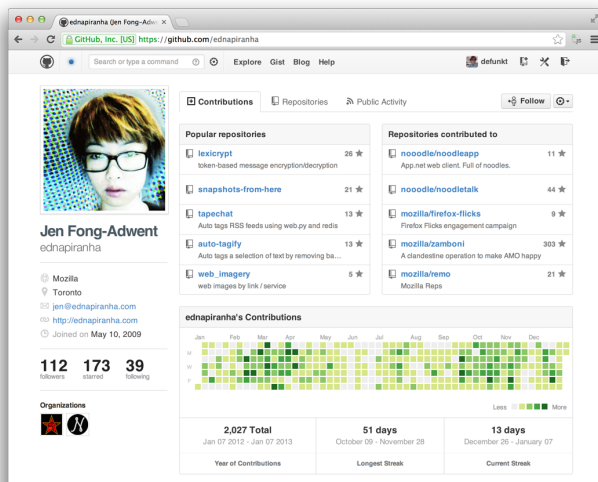


Image from Palmer 2013

## Conclusion

In this article I have tried to show that GitHub is a very good option for storing actively developed original social science data in the cloud. Journals wishing to host complete replication data sets for findings that they publish in a way that easily secures authorization and legal protection may want to continue to require data snapshots be deposited in a Dataverse-type archive.<sup>37</sup>

However, I hope to have demonstrated that GitHub is worth learning and using for developing and maintaining original social science data. It is at least as good as the alternatives in terms of stable storage, access, version control, and collaboration. In addition it is far better at enabling social data development and verification. GitHub has already enabled significant contributions to open source software development. We can easily use this service to both make social science data set collaboration easier and provide selective incentives to do so. Hopefully, this will improve the social science community's utilization of its collective knowledge to create more complete and robust data sets. Better data sets will allow us to better answer our research questions.

## References

- Altman, Micah and Gary King. 2007. "A Proposed Standard for the Scholarly Citation of Quantitative Data." *D-Lib Magazine* 13(3-4).
- Bowers, Jake. 2011. "Six Steps to a Better Relationship with Your Future Self." *The Political Methodologist* 18(2):2-8.
- Carey, John M. and Simon Hix. 2011. "The Electoral Sweet Spot: Low Magnitude Proportional Electoral Systems." *American Journal of Political Science* 55:383-397.
- Creative Commons. 2012. "Data." <http://wiki.creativecommons.org/Data>.
- Fomel, Sergey and Jon F Claerbout. 2009. "Reproducible Research." *Computing in Science & Engineering* 11(1):5-7.
- Fredrickson, Mark M., Paul F. Testa and Nils B. Weidmann. 2011. "Collaboration for Social Scientists, or Software is the Easy Part." *The Political Methodologist* 18(2):19-23.
- Gallagher, Michael. 1991. "Proportionality, Disproportionality, and Electoral Systems." *Electoral Studies* 10(1):33-41.
- Healy, Kieran. 2011. "Choosing your work flow applications." *The Political Methodologist* 18(2):9-18.
- Imai, Kosuke, Gary King and Olivia Lau. 2008. "Toward A Common Framework for Statistical Analysis and Development." *Journal of Computational and Graphical Statistics* 17(4):892-913.
- Kelly, Clint D. 2006. "Replicating Empirical Research in Behavioral Ecology: How and Why it Should be Done But Rarely Ever Is." *The Quarterly Review of Biology* 81(3):221-236.
- King, Gary. 1995. "Replication, Replication." *PS: Political Science and Politics* 28(3):444-452.
- King, Gary. 2007. "An Introduction to the Dataverse Network as an Infrastructure for Data Sharing." *Sociological Methods & Research* 36(2):173-199.
- Olson, Mancur. 1965. *The Logic of Collective Action: Public Goods and the Theory of Groups*. Harvard Economic Studies. Cambridge, MA: Harvard University Press.
- Owen, Matt, Kosuke Imai, Gary King and Olivia Lau. 2012. *Zelig: Everyone's Statistical Software*. R package version 4.1-1. <http://CRAN.R-project.org/package=Zelig>
- Palmer, Justin. 2013. "Introducing Contributors." <https://github.com/blog/1360-introducing-contributions>.
- Stodden, Victoria. 2009. "The Legal Framework for Reproducible Scientific Research." *Computing in Science & Engineering* 11(1):35-40.

<sup>37</sup>Enterprising journals could create GitHub organization accounts. This would require some more work to apply the appropriate legal protections as they are not 'in the box'.

Wickham, Hadley and Winston Chang. 2012. *devtools: Tools to make developing R code easier*. R package version 0.8. <http://CRAN.R-project.org/package=devtools>.

Xie, Yihui. 2011. “How to Become an Efficient and Collaborative R Programmer.” <http://yihui.name/en/2011/12/how-to-become-an-efficient-and-collaborative-r-programmer/>.

## A Tutorial on Deploying and Using Amazon Elastic Cloud Compute Clusters

John Beiler  
Pennsylvania State University  
[jub270@psu.edu](mailto:jub270@psu.edu)

### The Cloud and You

With the datasets analyzed by Political Scientists growing ever larger and analysis becoming more complex, it is often necessary to utilize more powerful computing resources. Research using large amounts of network data, event data, or textual data often pushes the limits of what an individual machine can accomplish in a reasonable period of time, if at all. The use of cloud resources allows tasks that take a large amount of time to be offloaded to a remote server in order to free up the user’s local machine. Alternatively, for tasks larger than one computer can handle, one can divide and distribute a job across a cluster of servers. While many universities offer high-performance computing resources, it is often the case that the user does not have free reign over what software is installed; it can take hours, days, or even weeks to have a required piece of software installed by the server administrators. Additionally, jobs on university resources are typically restricted to a certain run length, such as 24 hours. The use of a remote server that you rent, for as little as two cents per hour, enables whatever software is necessary to be installed when desired, and for jobs to be run as long as required. Amazon’s Elastic Compute Cloud (EC2) environment provides access to these cloud resources for the rental of a server or cluster of servers. Amazon EC2 allows for the creation of a cluster with up to 20 machines, each with multiple processing cores available. This is a large amount of computational power available on demand and at relatively low cost.

While EC2 offers quick and straightforward rental of computing resources, setting up and managing the servers comes with a rather steep learning curve. This article provides a brief introduction to the setup and use of EC2 resources. The focus is on the use of the **Starcluster** utility

for creating and managing EC2 clusters. Following this, I provide a brief overview of using R and Python in parallel on a server cluster. Code and examples for the routines presented below are hosted on [github](#).

### Starcluster and EC2

Before starting an analysis on an EC2 server, it is necessary to follow a few steps to set up the server. There are two primary components of an EC2 server: the instance, which refers to the hardware used, and the Amazon Machine Instance (AMI), which refers to the software deployed on the machine such as the operating system and other packages. In order to ease the deployment of an EC2 instance, **Starcluster** was developed by the **STAR program** at MIT.

The following sections walk through the installation and configuration of **Starcluster**. This entails installation of **Starcluster**, the creation of an Amazon Web Services (AWS) account, the creation of an Elastic Backed Stores (EBS) volume for the storage of user data, and finally the installation of software for analysis, such as *R*, and the attendant libraries and packages, such as *joblib* in Python and *snow* in *R*.

### Installing Starcluster

Since **Starcluster** is based on Python it is possible to easily install the utility using the `easy_install` method.<sup>1</sup> Unfortunately, `easy_install` does not come prepackaged with a Python distribution. To install `easy_install` in a Unix-like environment, download the appropriate Python egg from [the Python Package Index](#), change into the directory that contains the egg, and run the shell script. Installing **Starcluster** requires the presence of a C compiler. On OS X, XCode must first be installed from the App Store. Within XCode, the command-line tools must be installed by selecting in the menubar **XCode** → **Preferences** → **Downloads** and installing the **Command Line Tools**. On other Unix-like systems, if a C compiler is not already installed, one can be loaded using the package-management method used on that particular distribution. This will then allow the installation of **Starcluster** on the local machine.

```
#Change to the directory where the egg was downloaded
cd ~/Downloads
```

<sup>1</sup>This tutorial will assume the reader is working in a Unix-like environment such as Linux or OS X and has Python already installed. If on Windows, tutorials on installing Python and `easy_install` can be found [here](#) and [here](#).

```
#Execute the shell script sh setuptools-0.6c11-py2.7.egg
#Install starcluster sudo easy_install StarCluster
```

Following the install, typing `starcluster help` into the command-line will bring up dialogue asking the user to select an option for the configuration file. At this point type 2, which will save the `config` file in the `~/.starcluster` directory. The following section describes the information the `config` file should contain.

## Configuring Starcluster

With `Starcluster` installed, it is necessary to set up the configuration file for `Starcluster`. There are two basic parts to the configuration file: user information and instance information. The following sections provide guidance for adding the AWS user information to the `config` file, as well as adding the various templates and other options necessary to create an EC2 instance.

**Configuring User Information.** All configuration options for `Starcluster` are found in the `config` file located in the `~/.starcluster` directory. The default `config` provided by `Starcluster` has numerous comments and options. It is good to keep these in the file in order to see the available options, but in the interest of clarity I have provided a cleaner `config` with the configuration discussed in this article at the [github link](#) provided earlier.

The first step to configuring `Starcluster` is to create an **Amazon AWS** account. Once an account is created, navigate to the “Security Credentials” page, which can be found in the drop-down menu entitled “My Account/Console” at the top right corner of the page displayed immediately after login. Once on the “Security Credentials” page, in the “Access Credentials” section there is the option to create a personal access key. First create an access key, and then copy down the Access Key ID, the Secret Access Key, and the Account Number, which can be found toward the top of the page under the name used to register the account. This information should be placed in the `config` file in the section entitled **AWS Credentials and Connection Settings**. To open and edit the `config` file, execute the following commands:<sup>2</sup>

```
cd ~/.starcluster
vim config
#Or depending on your editor preferences
emacs config
#On Mac, the following will open the config file in TextEdit
open config
```

<sup>2</sup>It might be useful at this junction to point out that the use of a “programmer’s” editor will likely be necessary. When working on a remote server it often is not possible, or is very difficult, to use a graphical editor. Text-based editors such as vim or emacs come preloaded on almost every instance of Linux available. They are very powerful and useful, but come with a fairly steep learning curve.

<sup>3</sup>You can store data on the instance itself, but if you terminate the cluster the data is deleted. EBS storage allows you to terminate and restart clusters and keep the same data.

<sup>4</sup>For the examples used in this article a small amount of storage is necessary; 5 GiB should suffice.

Next, a pair of SSH keys must be created. SSH stands for secure shell, and is a method for “tunneling” securely from one computer to another. `Starcluster` uses SSH to allow to access to the EC2 instance. To create the key pair, execute the following commands:

```
#If the ~/.ssh directory does not exist
mkdir ~/.ssh
starcluster createkey aws_key -o ~/.ssh/aws_key.rsa
```

This will create a key in the `~/.ssh` directory on the local machine. This information should then be added to the `config` file in the **Defining EC2 Keypairs** section. The updated section should read as follows:

```
[key aws_key]
KEY_LOCATION=~/.ssh/aws_key.rsa
```

The next step is to create an Elastic Backed Storage (EBS) volume in order to store data in a persistent manner.<sup>3</sup> In the AWS management console, which is accessed by clicking the “My Account/Console” link at the top of the page after logging in to AWS, navigate to the EC2 section, followed by the “EBS Volumes” page under “My Resources.” Once on this page, create a new volume, with volume type of “standard” and the desired amount of storage<sup>4</sup>, and copy down the Volume ID to add to the **Configuring EBS Volumes** section of the `config` file. For this example, the EBS volume will be named `data` and will be mounted on the cluster at `/root/data`. This gives the following configuration:

```
[volume data]
VOLUME_ID = #INSERT ID
MOUNT_PATH = /root/data/
```

The final step is to uncomment, i.e., delete the “#” symbols around, the `ipcluster` plugin, which is located roughly around line 280 in the plugins section of the configuration file. After completing these steps the `config` file is properly setup with the basic user and configuration information. The next step is to define various server configurations that will be used.

**Configuring Cluster Templates.** There are three primary components to the setup of a server: the AMI used, the instance type, and the size. The individuals at the STAR program have generously provided public AMIs that have many of the components necessary for scientific research such as Python, OpenMPI, and the Sun Grid Engine, already present. The next section will cover creating your own AMI as an alternative to using the STAR AMIs. The second component necessary to create an EC2 instance is the instance type; Amazon offers numerous instance types with varying **configurations** and **prices**. For the purposes of this tutorial, the 64-bit `Starcluster` AMI will be used



on the M1 Extra Large instance type. This leads to the following configuration, defined in the **Defining Cluster Templates** section:

```
[cluster base]
KEYNAME = aws_key
CLUSTER_SIZE = 1
CLUSTER_USER = john
CLUSTER_SHELL = bash
NODE_IMAGE_ID = ami-999d49f0
NODE_INSTANCE_TYPE = m1.xlarge
VOLUMES = data
PLUGINS = ipcluster
```

While this setup is sufficient for many use cases, there are other situations that might require more memory or more nodes in the cluster. **Starcluster** allows for the definition of further templates in the **Defining Additional Cluster Templates** section. The following code defines a small cluster with the same basic characteristics as the **base** configuration, but starting two nodes instead of one.

```
[cluster basecluster]
EXTENDS = base
CLUSTER_SIZE = 2
```

The final step is to head back up to the beginning of the config file and define the **DEFAULT\_TEMPLATE** as **base**.

**Using the Cluster.** With all of the configuration options defined, the EC2 instance can finally be spun up and used. The following code will start a server named “mycluster” which can then be accessed using SSH. The first attempt to SSH into a server will be met with a long message about unknown hosts. This is nothing to worry about; just type “yes” and hit return.

```
starcluster start mycluster
starcluster sshmaster mycluster
```

Alternatively, a cluster following the **basecluster** configuration can be started by running:<sup>5</sup>

```
starcluster start mycluster -c basecluster
starcluster sshmaster mycluster
```

Users must issue the **starcluster terminate mycluster** command to shut the EC2 instance down. *It is important to note that if the instance is not explicitly terminated the instance will keep running and you will be charged for the uptime of the server.* Once connected to

the cluster operation is the same as any Unix command-line interface.<sup>6</sup> In order to exit the server and terminate the instance, the following commands are used:

```
#This will exit the EC2 instance and returns to the local
machine exit

starcluster terminate mycluster
```

**Adding Software and Creating AMIs.** Creating a custom AMI is optional for the use of an EC2 instance. If one desires, software and packages can be loaded to the instance each time it is started. Having an AMI with all of the software pre-loaded, however, can save a large amount of time and repetitive action. In addition, having this AMI created will ensure that the same software is installed on all nodes within a cluster. Thus, the following example shows how to load the libraries and software needed for the rest of this tutorial, such as **R** and various Python packages, and how to create an AMI that will allow this same software configuration to be loaded repeatedly. This section assumes that the reader wishes to create a custom AMI to be reused. If not, the commands used to install software should be issued on the instance used to run analyses. The following commands start a special type of EC2 instance configured for the creation of AMIs and SSHs into the instance as usual.

```
starcluster start -o -s 1 -i m1.xlarge -n ami-999d49f0
imagehost
```

```
starcluster sshmaster imagehost
```

The AMI that the custom AMI is built upon runs on the operating system Ubuntu, which is a specific distribution of Linux. Ubuntu uses the command **apt-get install** to install programs.<sup>7</sup> Before installing packages, however, it is necessary to tell Ubuntu to look in a different location for the newest version of **R**. In the file **/etc/apt/sources.list** add the line **deb http://cran.mtu.edu/bin/linux/ubuntu oneirc/**. This will allow the newest version of **R** to be installed instead of version 2.13, which is usually installed by using **apt-get**.

```
apt-get update
apt-get install r-base-core
```

```
easy_install pip
pip install pandas
pip install joblib
```

```
R
```

<sup>5</sup>I advise against having multiple, different instances running at once. It is far too easy to forget how many are running, which can lead to a rather large (and unexpected) bill at the end of the month.

<sup>6</sup>EC2 will close the connection if the session does not receive any input. This means that if a job or script is running, but nothing is typed into the terminal, EC2 will close the connection and the progress on the job will be lost. Thus, it is often a good idea to split jobs into **GNU Screen** sessions. This is done by typing **screen -S analysis**, which will create a screen session named **analysis**. To exit from a screen session simply use the command **Ctrl-a-d**, which indicates that the control key should be held down while pressing a then d. Then the screen session can be resumed using **screen -r analysis**.

<sup>7</sup>Reference is often seen to **sudo apt-get install**; the presence of **sudo** tells the machine to run the following command as the root user. This addition is unnecessary in this situation since the user is logged in to the instance as the root user already.



Once inside the R session, it is necessary to install several packages that will be of use later when performing analyses in parallel. The specific packages used are (Analytics 2012c), doParallel (Analytics 2012a), snow (Tierney, Rossini, Li and Sevcikova 2012), and doSNOW (Analytics 2012b). After these packages are installed, R can be exited, as can the instance itself.

```
install.packages('foreach')
install.packages('doParallel')
install.packages('snow')
install.packages('doSNOW')
q()

exit
```

Now back at the local machine's command line, it is time to create the custom AMI. Executing the following code will create an AMI with a unique AMI ID that can be placed in the config file in place of the STAR AMI that is currently in use. In other words, the new AMI ID would replace ami-999d49f0 in the configuration.

```
#Note instance ID
starcluster listinstances
starcluster ebsimage <INSTANCE-ID> analysis-image
#Note the new AMI ID that prints out

starcluster terminate imagehost
```

## Performing Analyses on a Cluster

The server has now been configured and R, along with other libraries and packages, has been installed. At this point R can be used as it is on any other machine, but with the potential for much more computational power. The features of EC2 can be utilized for either cluster or multicore processing, depending on the problem. One potential situation that can arise when analyzing big data is that a long job needs to be offloaded onto a remote server. In this situation multicore processing, which is the use of multiple cores on a single CPU within the machine, may be advantageous. If a specific task is too large for a single machine to handle, due to issues such as RAM limitations, cluster computation, or using multiple machines to perform the computations, may be warranted. It is important to note that the use of a cluster is not always necessary; sometimes a machine with a large amount of RAM is sufficient for the task and will allow for greater simplicity (Rowstron, Narayanan, Donnelly, O'Shea and Douglas, 2012). The various instance types available on EC2 allow for the selection of the proper setup for either situation.

Given these two different environments, multicore or cluster, different steps are required depending on which is

utilized for a given task.<sup>8</sup> This section focuses on “embarrassingly parallel” situations of the type commonly encountered in basic data cleaning, data subsetting, or data simulations, e.g. Monte Carlo simulations. I define “embarrassingly parallel” problems as those that can commonly be approached using a for-loop in a computer program. The running example used is a function that generates 100 draws from the uniform distribution 100 times, which are then transformed to the exponential distribution from which a mean is calculated. This function is then called 100 times.<sup>9</sup>

## Using R on EC2

**Multicore.** The following example makes use of the `foreach` library in R. The additions needed to run code in parallel in a multicore setting are to register a parallel backend, using `makeCluster()` and `registerDoParallel`, and the addition of `%dopar%` instead of `%do%` before the function being used, which will call the function in parallel instead of sequentially. In R,

```
library(foreach)
library(doParallel)

cl = makeCluster(3)
registerDoParallel(cl)

unif.trans = function(){
  results = matrix(nrow=100,ncol=100)
  for(i in 1:100){
    results[i,] = runif(100)
    exponential = -log(results)
  }
  return(mean(exponential))
}

x = foreach(i=1:100, .combine='c')
  %dopar% unif.trans()
mean(x)
```

**Cluster.** Performing analysis utilizing a cluster of computers uses a similar approach, but requires some communication between the various nodes within the cluster. R has some packages, such as `snow` and `snowfall`, that assist in this. The first step is to create a cluster with more than one node in it, such as defined by the `basecluster` template. The following code assumes that any other instances named `mycluster` have been terminated. Additionally, the following should be executed on the user's local machine in order to create the new EC2 instance.

```
starcluster start mycluster -c basecluster
#Note the IP Address of the instances in the cluster
starcluster listclusters
```

<sup>8</sup>An added level of complexity not covered is the combination of cluster and multicore computing. In short, one can create a job that shares work amongst nodes on a cluster, which is then further divided amongst multiple cores. To achieve this, one must simply combine the two different sets of code outlined in the multicore and cluster sections below.

<sup>9</sup>This is an admittedly trivial example, but it shows the basics of how a Monte Carlo simulation might proceed in a cluster or multicore environment.

The approach for analysis on a cluster environment is extremely similar to that for a multicore environment; the main difference lies in how the parallel backend is created. For the cluster setting, it is necessary to specify the IP addresses of the nodes included in the cluster.

```
library(snow)
library(doSNOW)
library(foreach)

#Replace MASTER and NODE001 with the
#appropriate IP Address
cl = makeCluster(c('MASTER.compute-1.
amazonaws.com',
'NODE001.compute-1.amazonaws.com'),
type='SOCK')
registerDoSNOW(cl)

unif.trans = function(){
  results = matrix(nrow=100,ncol=100)
  for(i in 1:100){
    results[i,] = runif(100)
    exponential = -log(results)
  }
  return(mean(exponential))
}

x = foreach(i=1:100, .combine='c')
  %dopar% unif.trans()
mean(x)
```

**snow** also comes packaged with parallel and cluster versions of the **apply** family of functions. A more detailed discussion of these can be found in the [snow documentation](#).

## Using Python on EC2

**Multicore.** The easiest approach for implementing embarrassingly parallel for-loops in a multicore situation in Python is the **Parallel** functionality of the **joblib** package. The code below illustrates the use of **joblib** in the same toy simulation used in the R examples. The heart of the code below is the call to the **Parallel** function. The **n\_jobs** argument tells the function how many cores to use; -1 indicates the use of all available cores. The **uniform\_trans** function is then called 100 times, with these 100 calls split across all available cores.

```
starcluster sshmaster mycluster
ipython

from joblib import Parallel, delayed
import numpy as np
import scipy.stats as stats

def uniform_trans():
  results = list()
  for i in xrange(100):
    results.append(stats.uniform.rvs
      (size=100))
  results = np.asarray(results)
  exponential = -np.log(results)
```

```
    return np.mean(exponential)

means = Parallel(n_jobs=-1)(delayed
  (uniform_trans)()
  for _ in xrange(100))
finalMean = np.mean(means)
```

**Cluster.** For analysis on a cluster using Python, the developers of **Starcluster** had the foresight to include the IPython (Perez and Granger, 2007) cluster plugin. This allows analysis to be easily forked to different nodes within a server. The main difference is that it is necessary to login to the EC2 instance with a different user than root, hence why the user **john** was defined in the **config** above. The basic functioning of the below code is to create a **Client** object, which contains the information about the available nodes in the cluster. The **nodes** object is then assigned all possible worker nodes. The asynchronous map function is then used to split the code between the nodes and collect the results in an asynchronous manner.<sup>10</sup> A final point of interest is that since the function is being sent to various worker nodes, it is necessary to import the appropriate packages within the function itself.

```
starcluster sshmaster mycluster -u john
ipython

from IPython.parallel import Client
import numpy as np

cluster = Client(packer='pickle')
nodes = cluster[:]

def uniform_transform(z):
  import scipy.stats as stats
  import numpy as np
  results = list()
  for i in xrange(100):
    results.append(stats.uniform.rvs
      (size=100))
  results = np.asarray(results)
  exponential = -np.log(results)
  return np.mean(exponential)

gather = nodes.map_async(uniform_transform,
  xrange(100))
means = gather.get()
mean = np.mean(means)
```

## Resources and Final Thoughts

While this article serves as an extremely brief introduction, there are many resources available for exploring parallel computation in R and Python in greater depth. Before explicating these resources, however, a final note on using parallel processing is in order. For small examples like the one used in this article, it is sometimes *slower* to run the process in parallel due to the scheduling and recombining of results. It is important to identify the bottleneck in your

<sup>10</sup>Further explanation of **map\_async** is located in the [IPython Documentation](#).

workflow. If you are trying to fit a model to a large amount of data and hitting memory limits, it is likely easier to use the high memory EC2 instance with 68 GB of memory.<sup>11</sup> On the other hand, if your data subsetting script is taking an hour or more to run, a cluster or multicore solution might be useful. In addition, some problems are not easily parallelized while others are not parallelizable at all. The type of algorithms that are easily parallelized, however, could serve as the subject of an entirely different article.<sup>12</sup>

If multicore or cluster computing is the best way forward for a given problem, there has been a copious amount of (digital) ink spilled outlining the various options available for parallel computation in both R and Python. In R there are the `foreach`, `snow`, and `snowfall` packages discussed in this article, in addition to the various implementations of `apply`.<sup>13</sup> There are also explicit implementations of MPI in R such as `Rmpi`, a good example of which, along with a less trivial usage of parallel processing than presented in this article, can be seen [here](#). In Python, MPI is also available, as is the `multithread` package. The easiest and most straightforward approach, however, is to make use of IPython and `joblib`. These two should cover almost any imaginable scenario. With this in mind, the aim of this article was not to provide an exhaustive tutorial on parallel computation; in reality this would devolve into a repetition of the documentation for the various implementations mentioned above. Rather, the hope is that this article has provided the reader with a working understanding of, and a quick-start guide for, 1) initiating and running an AWS EC2 instance and 2) utilizing an EC2 instance for the purposes

of parallel computing in R and Python.

## References

- Perez, Fernando and Brian E. Granger. 2007. "IPython: a System for Interactive Scientific Computing." *Computer Science Engineering* 9(3):21–29. <http://ipython.org>.
- Rowstron, Antony, Dushyanth Narayanan, Austin Donnelly, Greg O'Shea and Andrew Douglas. 2012. "Nobody ever got fired for using Hadoop on a cluster". In *HotCDP 2012 -1st International Workshop on Hot Topics in Cloud Data Processing*. Bern, Switzerland: <https://research.microsoft.com/pubs/163083/hotcbp12%20final.pdf>.
- Revolution Analytics. 2012a. *doParallel: Foreach parallel adaptor for the parallel package*. R package version 1.0.1. <http://CRAN.R-project.org/package=doParallel>.
- Revolution Analytics. 2012b. *doSNOW: Foreach parallel adaptor for the snow package*. R package version 1.0.6. <http://CRAN.R-project.org/package=doSNOW>.
- Revolution Analytics. 2012c. *foreach: Foreach looping construct for R*. R package version 1.4.0. <http://CRAN.R-project.org/package=foreach>.
- Tierney, Luke, A. J. Rossini, Na Li and H. Sevcikova. 2012. *snow: Simple Network of Workstations*. R package version 0.3-10. <http://CRAN.R-project.org/package=snow>.

## Fielding Complex Online Surveys using rApache and Qualtrics

Taylor C. Boas and F. Daniel Hidalgo

Boston University and Massachusetts Institute of Technology  
tboas@bu.edu and dhidalgo@mit.edu

## Introduction

Online surveys are increasingly popular in political science as an easy and inexpensive way to gather data and con-

duct experiments. Typically, scholars use an online survey engine, such as Qualtrics, SurveyGizmo, or SurveyMonkey, to administer the questionnaire and handle any experimental manipulation.<sup>1</sup> Compared to programming a survey from scratch and hosting it on one's own server, commercial survey packages present many advantages. They are user-friendly and reliable, offer an attractive graphical user interface, and may be available at no cost for scholars whose universities purchase site licenses. Their high levels of data security and built-in anonymity features should also make them more attractive to Institutional Review Boards than

<sup>11</sup>In fact, as I was writing this I received an email from Amazon announcing new types of instances that have 244 GiB of RAM and two Intel Xeon processors, which each have 8 cores for 16 total physical cores and 32 total threads. In reality this instance should be more than enough firepower for nearly any application that could arise in political science research.

<sup>12</sup>In short, if an algorithm contains the summation of results it is probably possible to run it in parallel.

<sup>13</sup>A good resource for a high level overview for some of these commands is Ryan Rosario's presentation on parallelizing R, available [here](#).

<sup>1</sup>Recruitment of subjects or respondents—a separate and prior step—might rely on Amazon.com's Mechanical Turk, Facebook advertisements, or a convenience sample drawn from one's university.

self-programmed and locally-hosted surveys.

Despite their many advantages, even the most powerful online survey engines may present limitations in terms of question wording customization, complex randomization, or other goals that a researcher may wish to accomplish. For example, in our research on Brazil, one task we were unable to accomplish within the survey engine Qualtrics was to pull up a list of candidates for city council from the respondent's municipality and randomly choose one of these names to be inserted into a survey question.

In this article, we describe a strategy for augmenting the capacity of online survey engines using rApache, a version of R that runs on the Apache web server. The approach involves routing respondents to a server running an R script that, via a web interface, administers an initial set of survey questions. Based on the answers to these questions, the R script conducts any necessary randomization and database lookups and then passes the results to the online survey engine via the redirect URL. No respondent data are retained by the server that is used for this preprocessing step.

We focus on our experience integrating rApache with Qualtrics, a powerful online survey engine that is oriented toward academic research and is commonly available through university site licenses.<sup>2</sup> Qualtrics has recently been used for a number of online surveys in political science, many of which involve experimental manipulation (Kamal et al., 2012; Kriner and Shen, 2012; Morey, Evaland and Hutchens, 2012; Nyhan and Reifler, 2011; Popescu and Toka, 2012; Sances, 2012; Shineman, 2012; Young and Hoffman, 2009; Zahedzadeh and Merolla, 2012). As we discuss below, some of these studies might have benefited from using the method described here.

## The Survey: Religion, Race, and Class in Brazilian Municipal Elections

In late September–early October 2012, we used Qualtrics to administer an online survey, including several experimental treatments, to 1820 registered voters in Brazil.<sup>3</sup> The survey was designed to explore issues of religion, race, and class in the country's October 7 municipal elections. Following the approach of Samuels and Zucco (2012*a,b*), we recruited respondents using Facebook advertisements and raffled off an iPad as an incentive for participating. The research design and results of the survey are described in greater detail elsewhere (Boas, 2013; Smith, 2013). Here, we focus on features that we wished to include in the online survey but were unable to implement natively (or could do so only awkwardly) in Qualtrics. We then describe how we were able to

accomplish these tasks using rApache on a virtual server.

A first set of survey questions sought to gauge the effect on vote intention of candidates using professional titles, such as "Pastor" or "Doctor," in their ballot names (a practice this is permitted under Brazilian electoral law, and quite common). Each question described a candidate for city council, including his or her party, age, marital status, and education level. Respondents in the treatment condition were given a version of the candidate's name that included the professional title, such as "Pastor Paulo" or "Dr. Carlos"; those in the control condition were given full legal name, without the title.

To increase the external validity of the survey experiment, we wanted to ask a subset of respondents about real candidates from their municipality that were running in the upcoming election. Doing so required two steps that were impossible in Qualtrics: gathering the respondent's state and municipality in such a manner that the data could be used elsewhere in the survey, and looking up a list of candidates from that municipality. Qualtrics offers a question type, Drill Down, that involves selecting items from nested drop-down menus, and we could have used this option to ask for the respondent's state and then municipality. However, if one wants to use the answer to a Drill Down question in display logic or as piped text elsewhere in the survey (e.g., showing certain questions only to residents of state capitals, or inserting municipality name into the question "How long have you lived in \_\_\_\_\_"), there is a limit on the number of categories that can be included. We are unaware of the exact limit (and Qualtrics technical support was unable to specify it), but Brazil's 5568 municipalities and the 3141 counties or county-equivalents in the United States both lie above it. We might have been able to circumvent this problem by asking respondents to input their postal code in a text box. However, we still would have run up against a second limitation: Qualtrics has no table-lookup feature that we could have used to match municipality name or postal code to a list of candidates.

A second part of the survey sought to have respondents characterize a large database of candidate photos in terms of race and social class. We started with 1000 photos and intended to show a randomly chosen photo to each of 3000 respondents, such that each photo was rated by three respondents.<sup>4</sup> Randomizing among 1000 photos would have been possible in Qualtrics, but exceedingly awkward; we would have had to create 1000 versions of the same question, each with a different image URL.

In a third part of the survey, we wanted to ask a random sample of 10% of respondents if they would be opposed to

<sup>2</sup>Given Qualtrics's reputation as one of the most capable online survey engines available (e.g., Leland, 2011), we assume that the limitations we have encountered while using it apply to most other packages as well.

<sup>3</sup>The survey was conducted jointly with Amy Erica Smith of Iowa State University and was approved by the Institutional Review Boards of each of our institutions.

<sup>4</sup>We ultimately recruited fewer respondents than initially expected, and thus had to reduce both the number of photos and the number of times that each was shown.

their son or daughter marrying a black person. To do this within Qualtrics, we would have had to randomize among this question and 9 copies of a blank question—not difficult, but rather cumbersome.<sup>5</sup>

A final challenge was specific to our compensation mechanism—a raffle prize for one respondent who completed the survey. In contrast to Mechanical Turk, which allows for directly compensating participants without obtaining their personal data, our approach required us to collect the names and email addresses of respondents who completed the survey and wished to be entered into the drawing for an iPad. However, to obtain exempt status from the IRB, names and emails could not be linked to survey responses. The solution was to collect personal data through a second Qualtrics survey to which interested respondents would be redirected after completing the first one. However, we were concerned that savvy users might copy the URL of the second survey and distribute it to friends, or use it themselves to enter the raffle multiple times with different email addresses.<sup>6</sup> Hence, we needed a unique identifier for each respondent that would be passed from the first survey to the second, but, for purposes of anonymity, not generated by or recorded in the first survey.

## The Solution: Preprocessing with rApache on a Virtual Server

We employed rApache as our server side solution. rApache is a module for Apache developed at Vanderbilt University that embeds the R interpreter inside a web server. rApache allows the Apache web server to interact with R, which is useful when one wants to use R's advanced statistical and data management tools to manipulate data received through a web interface. rApache requires Apache 2.2.x or above with an installation of R. In our particular case, we used an Ubuntu virtual server hosted by the firm Linode (<http://www.linode.com>), but any standard Linux-based web server should suffice. Installation instructions are documented in the rApache manual (<http://rapache.net/manual.html>), but some familiarity with Apache server administration is required.

Our questionnaire, a standard web form, initiates a GET request that passes values of the items entered into the form to R. The HTML web form would look something like the following:

```
<form method="GET" action="/r-scripts/SendToQualtrics.R">
Choose the state where you vote
<select name="state" id="state">
<option value="">Select a state</option>
<option value="AC" >Acre</option>
<option value="AL" >Alagoas</option>
```

```
<option value="AP" >Amapa</option>
<option value="AM" >Amazonas</option>
...
</select>
<input type="submit" value="Advance">
</form>
```

In this example, the user is presented with a drop-down menu and asked to choose the state where they vote. One could then conduct a survey experiment in which the content of the question prompt would depend on the respondent's state. For example, the researcher could ask respondents for an evaluation of their state governor's job performance, randomly assigning them to be presented with the governor's party affiliation. Much more complicated randomization schemes and data lookups could be accommodated—including the one we used, which depended upon the respondent's state and municipality—but this running example illustrates the basic method. While this particular example could have been implemented in Qualtrics using branching or display logic, it would have been cumbersome to do so for Brazil's 27 states.

The only necessary change to standard HTML is the need to specify the R file to be called once the form is submitted. Notice that in the `action` variable in the form header, we specified an R file called `SendToQualtrics.R`. This is the R file that will handle the data entered by the user. Upon clicking on the "Advance" button, the data entered into the form is sent to an R instance. The data arrives in R in the form of the of a list object called `GET`, with an element for each variable in the form.

R receives the data as a list object that can then be manipulated. In this particular case, R would launch and run the script `SendToQualtrics.R` which could operate on the data contained in the `GET` list object, where the name of each element is given by the name attribute of the form. The element in the list object would contain the "value" specified for that particular choice in the HTML (i.e., the state abbreviations). In the case of a respondent selecting Acre in the menu, R would receive the following web object:

```
> GET
$state
[1] "AC"
```

In this hypothetical case, one could have the server-side R script load a dataset containing the names and party affiliations of all state governors. The R script could then contain functions that operate on the `GET` list, such as:

<sup>5</sup>More generally, one cannot directly specify unequal probabilities of selection when randomizing among questions. Hence, assigning one-tenth of a sample to a treatment condition and the rest to control would require making 9 copies of the control question.

<sup>6</sup>Qualtrics has an HTTP Referrer Verification feature that could be used to block users not being redirected from the first survey. However, it would also block all those who had HTTP referral disabled on their browser for security or privacy reasons, and we were reluctant to do this.



```
getGovernorData <- function(state){
  treat <- sample(0:1, 1)
  gov_name <- governors$name[governors$state == state]
  gov_party <- governors$party[governors$state == state]
  question_name <- ifelse(treat == 1,
    paste(gov_name, "(", gov_party, ")",
      sep = ""),
    gov_name)
  URLEncode(question_name)}
```

This function accepts the state abbreviation passed by the web server, randomly assigns the respondent to a treatment or control condition, and then produces the name that will be presented in the survey prompt (with or without the party label, depending on treatment status). Because this customized text will be passed to Qualtrics in the form of “embedded” text in a URL, special characters must be encoded. This list includes accented characters such as “á”, which needs to be transformed into “%c3%a1”, as well as a variety of more common characters, such as a space (“%20”). The R function `URLEncode()` performs this transformation as necessary; we recommend using it regardless of whether one’s data include accented characters.

To pass the data to Qualtrics, we need to embed it in the URL that sends the user to the rest of the online questionnaire. Qualtrics provides a unique URL for each survey, of the form `http://www.qualtrics.com//SE/?SID=SV_545454`, where the `SID=` field specifies the survey number. Additional variables can be appended to this URL in the form `&var=value`. To construct the URL in R, the following code would suffice:

```
gov_name <- getGovernorData(GET$state)
forwardUrl <- paste("http://www.qualtrics.com//SE/?SID=
  SV_545454&gov=", gov_name, sep = "")
```

The last step is to forward the respondent to the Qualtrics website with the data embedded in the redirect URL. To do so, at the end of our R script we used the following code:

```
setContentype("text/html")
cat("<head>")
cat("<code>")
cat("</code>")
cat(paste("<meta HTTP-EQUIV=\"REFRESH\"
  content=\"0; url=\", forwardUrl, \">\", sep = ""))
cat("</head>")
cat("<html>")
cat("<body>")
cat("</body>")
cat("</html>")
```

This code asks R to generate a blank HTML website with the single purpose of immediately forwarding the respondent to the Qualtrics website and passing along the embedded data.

In our survey, we used preprocessing in `rApache` to handle each of the four randomization and wording customiza-

tion tasks described in the previous section. Through a similar process to the “governor” example above, we used the respondent’s state and municipality to look up a list of city council candidates with “Pastor” and “Doctor” titles, randomly choose one candidate from each list, assign the respondent to the treatment or control condition (ballot name with professional title or full legal name), and pass the resulting name and candidate biographical data to Qualtrics. This included full phrases, such as

`&marital=she%20is%20married_`

that were inserted directly into the survey question as piped text.

Our additional randomization tasks were straightforward. To choose a candidate photo for the respondent to evaluate, we had R sample from a vector of 1000 filenames, passing along the result in the form `&photo=photo1.jpg`. In order to ensure that each photo was shown no more than three times, we had our photo-choosing function check the server log; if the limit had been reached, another file was chosen. To display the photo in Qualtrics, we created a Text/Graphic item, and then, in HTML view, inserted the embedded data field into the image URL, e.g., ``. To select the 10% of respondents who would be asked about their son or daughter marrying a black person, we did `sample(c(1,rep(0,9)),1)` in R and passed along the result to Qualtrics as `&marry=`. We then used display logic to show the question only when `marry=1`.

Finally, in order to generate a unique identifier for each respondent, we used R’s system time function. The command `as.character(as.numeric(Sys.time()))` returns the number of seconds since 1970, including fractional seconds, with the level of precision determined by the operating system (5 decimal places for our server). Hence, we could generate a unique 15-digit code for each respondent with `as.character(as.numeric(Sys.time())*100000)`. This code was passed to the main survey, and, upon completion, to the raffle survey, where it was recorded.<sup>7</sup> As a result, we could eliminate anyone who entered the raffle via a “shortcut” process that did not begin with their visiting the Linode server and obtaining a unique code.

The opportunities for complex randomization and question wording customization go well beyond the specific examples outlined here. Qualtrics places no limits on the number of embedded data fields that can be passed in via the URL, and virtually all browsers can handle URLs of up to 2000 characters. One practical limitation concerns possible delays in executing the R script. Computationally-intensive tasks might generate long pauses between when the user hits the “submit” button on the web form and when she is redirected to the survey. The same is true when working

<sup>7</sup>To pass embedded data *out* of Qualtrics, one simply creates a customized End of Survey element in Survey Flow and includes the field in the redirect URL.

with large data files, since a new instance of R is started for each user and the data have to be loaded from scratch. That said, we found that rApache was remarkably fast, and the delay from executing our preprocessing tasks was virtually undetectable.

The major drawback to preprocessing survey respondents in rApache is that it significantly increases the number of moving parts in one's survey. We found that extensive testing was necessary in order to work out the kinks prior to launching, and we cannot rule out the possibility that some unknown problem arose for some respondents. Our ad click-to-completed survey ratio of 22:1 was much higher than the 7:1 figure obtained by Samuels and Zucco (2012*b*) for a similar survey in Brazil the previous year. We think the difference is largely attributable to our longer questionnaire and to people having less patience for surveys during election season, but it is possible that unknown bugs caused us to lose some respondents.

A final drawback is that it is necessary to "activate" (i.e., launch) the survey in Qualtrics in order to do testing on data read in via the URL. While responses generated during this testing period are easily discarded or separated from real data, they will count against any quota on one's account, in contrast to testing done via the "Preview" function.

## Alternative Solutions

While combining rApache with Qualtrics worked well for our project, there are other solutions for implementing complex randomization schemes and database lookups in an online survey. A powerful and flexible solution would be to build a survey engine using a web framework such as Django, built using Python, or Ruby on Rails, built using Ruby.<sup>8</sup> These web frameworks employ full-fledged programming languages, which would allow for the customization required by many social science online surveys. Furthermore, they are designed to work with database backends, thus allowing for rapid data lookups and data collection. Packages specifically designed for constructing online surveys are available, such as Django-crowdsourcing (<https://pypi.python.org/pypi/django-crowdsourcing>) and Surveyor for Rails (<https://github.com/NUBIC/surveyor>). Furthermore, these popular frameworks have been used for websites with millions of users and thus have proven robust and scalable for a variety of tasks. The chief drawback to this approach is that web frameworks require a good deal of expertise to use effectively and thus can be "overkill" for the kinds of surveys political scientists wish to field.

Another approach would be a pure R-based solution that relies on rApache to administer the survey and collect response data. This approach is facilitated by the use of the "Brew" package ([http://cran.r-project.org/web/](http://cran.r-project.org/web/packages/brew/index.html)

[packages/brew/index.html](http://cran.r-project.org/web/packages/brew/index.html)), which is a templating framework for mixing R code and text. Using Brew, one could write an HTML template with embedded R code. When users visit the web page, R dynamically generates the necessary customized HTML within the template and serves it to visitors. Of course, this solution would forgo the ease of use and reliability of commercial tools like Qualtrics. Other approaches to using R to dynamically generate web pages are covered in Verzani (2012).

Finally, a promising recent development is the release by the makers of RStudio of "Shiny" (<http://www.rstudio.com/shiny/>), an R package that allows for the fast and easy creation of web applications which can take advantage of R's statistical capabilities. The chief advantage of Shiny is that it does not require knowledge of HTML or JavaScript and is very easy to use. At present, it is primarily designed to serve HTML locally, but Shiny's developers have announced plans for a hosting service that would allow for the deployment of applications online.

## Conclusion

Using rApache to augment the capacities of online survey engines such as Qualtrics takes advantage of R coding skills that many quantitative researchers already possess in abundance, and it requires only a minimal amount of additional skills, such as HTML coding or server administration, that are less common. Even for those with the programming ability to design and host their own surveys, integrating with an existing survey engine might be attractive due to their proven reliability and data security, as well as the fact that they are often available for free through university site licenses.

Our article has described applications of the rApache-plus-Qualtrics method that were specific to our own survey, but potential uses are much broader. Kriner and Shen (2012), for instance, use an online survey experiment conducted in Qualtrics to test the hypothesis that public support for war is more greatly affected by news of casualties that are local. In the treatment condition, respondents read a news story about the death in Afghanistan of a U.S. soldier from their home state, which they had specified earlier in the survey. In the control condition, a randomly chosen different state is specified. State, of course, only imperfectly operationalizes the concept of "local." An ideal strategy might be to ask for the respondent's ZIP code early in the survey and then describe the casualty as being from the corresponding county—a data lookup task that would be impossible within Qualtrics but is easily accomplished using our method. The effects of a "home county" treatment might be even larger than those (already sizable and significant) that Kriner and Shen (2012) found for causalities

<sup>8</sup>A promising set of tools for the creation and administration of online surveys is "Shanks," developed by Mark Fredrickson (<https://github.com/markfredrickson/shanks>). These tools are written in Clojure and run on top of Google App Engine.

from one's home state.

Another potential application of our method involves assigning respondents to experimental conditions when one wants to end up with a specific number of completed surveys in each group—something that may be especially important for smaller- $N$  survey experiments. In their  $N = 60$  study, for instance, Zahedzadeh and Merolla (2012, 13) ended up with fewer respondents than desired in some experimental conditions because subjects dropped out after assignment but before completing the survey. Qualtrics is capable of ensuring that equal numbers of respondents are *assigned* to different conditions—using the “evenly present elements” option in the Randomizer—but it cannot take into account whether respondents in each condition actually complete the survey. Using our method, Qualtrics could communicate with the server running rApache upon survey completion, e.g., by loading one of two blank image files, `treat.jpg` or `control.jpg`, on the last page of the survey. Probabilities of assignment, which would be done in advance by rApache, could be adjusted each time by consulting the server log, which would record the number of times each image was loaded.

For our specific research on voting behavior and candidate evaluations in Brazil's 2012 local elections, preprocessing via rApache allowed us to substantially boost the external validity of a survey experiment by asking respondents about real candidates from their municipalities. It also permitted forms of randomization, such as choosing 1 out of 1000 candidate photos, that would have been exceedingly awkward to implement in Qualtrics. Hopefully other researchers can use the method we propose to combine the data-processing power of R with the reliability, data security, and graphical user interface of online survey engines.

## References

- Boas, Taylor C. 2013. “Vote for Pastor Paulo: Religious Ballot Names as Heuristics in Brazil.” Paper presented at the Annual Meeting of the Southern Political Science Association, Orlando, FL, January 3–5.
- Kamal, Mia, Jason Turcotte, Donyelle Davis and Christy Arrazattee. 2012. “An experiment in tolerance: How religious stereotypes shape attitudes of reciprocity and political engagement.” Paper presented at the Annual Meeting of the Southern Political Science Association, New Orleans, January 12.
- Kriner, Douglas L. and Francis X. Shen. 2012. “How Citizens Respond to Combat Casualties: The Differential Impact of Local Casualties on Support for the War in Afghanistan.” *Public Opinion Quarterly* 76(4):761–770.
- Leland, Eric. 2011. “A Few Good Online Survey Tools.” [www.idealware.org](http://www.idealware.org), February.
- Morey, Alyssa C., William P. Eveland Jr. and Mylah J. Hutchens. 2012. “The ‘Who’ Matters: Types of Interpersonal Relationships and Avoidance of Political Disagreement.” *Political Communication* 29(1):86–103.
- Nyhan, Brendan and Jason Reifler. 2011. “Opening the Political Mind? The effects of self-affirmation and graphical information on factual misperceptions.” Manuscript, Dartmouth College/Georgia State University.
- Popescu, Marina and Gabor Toka. 2012. “How can mass media help citizens make sense of the political world? Media systems and citizens cognitive political engagement in Europe.” Paper presented at the workshop “Advancing comparative political communication research: New frameworks, designs and data,” ECPR Joint Sessions of Workshops, Antwerp, April.
- Samuels, David and Cesar Zucco. 2012a. “The Power of Partisanship in Brazil: Evidence from Survey Experiments.” Working paper, University of Minnesota/Rutgers University.
- Samuels, David and Cesar Zucco. 2012b. “Using Facebook as a Subject Recruitment Tool for Survey-Experimental Research.” Working paper, University of Minnesota/Rutgers University.
- Sances, Michael W. 2012. “Is Money in Politics Harming Trust in Government? Evidence from Two Survey Experiments.” Unpublished Manuscript, Massachusetts Institute of Technology.
- Shineman, Victoria Anne. 2012. “Incentivizing Participation Increases Political Information: Evidence from a Randomized Field Experiment.” Paper prepared for presentation at the Annual Meeting of the American Political Science Association, New Orleans, Aug. 30–Sept. 2.
- Smith, Amy Erica. 2013. “ClericalWork: Clergy, Media, and Religious Polarization, Brazil 2008–2012.” Paper presented at the Annual Meeting of the Southern Political Science Association, Orlando, FL, January 3–5.
- Verzani, John. 2012. “gWidgetsWWW: Creating Interactive Web Pages within R.” *Journal of Statistical Software* 49(10):112.
- Young, Dannagal Goldthwaite and Lindsay H. Hoffman. 2009. “An experimental exploration of political knowledge acquisition from The Daily Show versus CNN student news.” Paper prepared for presentation at the Annual Meeting of the American Political Science Association, Toronto, Aug. 30–Sept. 2.
- Zahedzadeh, Giti and Jennifer Merolla. 2012. “How do negative political ads impact public trust in candidates?” Paper presented at the Annual Meeting of the Western Political Science Association, Portland, Oregon, March 22–24.

## Crowdsourcing Panel Studies and Real-Time Experiments in MTurk

Dino P. Christenson and David M. Glick

Boston University

*DinoPC@BU.edu and DMGlick@BU.edu*

While researchers conducting quick experiments and pilot studies currently appear to make the most use of Amazon's Mechanical Turk (MTurk) as a subject recruitment tool, it is capable of supporting more ambitious research designs, many of which would be otherwise infeasible for most researchers. Specifically, researchers with a modest budget can use MTurk to recruit participants for customized and flexible panel studies. Because it is a cloud based environment, it is easy to recontact respondents, which helps limit panel attrition. Moreover, when used in tandem with online survey software, panel waves can be quickly fielded around imminent and recent political events, rather than at constant intervals or other times determined well in advance.

Thus, MTurk's attributes allow researchers to affordably collect individual level data for pre-post comparisons that can be combined with real-time experimental treatments. In this piece we briefly discuss our own experience conducting panel studies in MTurk and provide some basic instructions for researchers looking to do the same. We utilize the design and data from one of our own recent studies to discuss how we took advantage of MTurk and suggest some avenues for future research.

### Implementing a Panel Study in MTurk

Last summer we implemented a five wave panel study to capture public opinion about health care, the Supreme Court, and politics in general around the Supreme Court's Affordable Care Act (ACA) decision (Christenson and Glick, 2012). We collected two waves of data prior to the ACA decision, two waves shortly after, and then reconnected with our participants for a final wave around the November elections. In all, we were able to independently conduct five extensive political surveys for approximately \$5,000 total. Employing an MTurk sample was not just inexpensive but, as we show below, also offered unparalleled flexibility for collecting panel data and implementing a real-time experimental treatment around a real world event.

We used MTurk for sample recruitment and re-contact while conducting the actual surveys using an online platform.<sup>1</sup> We have used both SurveyGizmo and Qualtrics in our work and found MTurk to be equally compatible with them. The first step in recruiting a panel is very similar to conducting a one-time survey in MTurk. You simply have

to post a public "HIT" (MTurk jargon for an assignment or "Human Intelligence Task") to attract participants and direct them to the survey. For this HIT we offered participants \$1 for a 15 minute survey in which they would "answer questions about politics and healthcare." Following Berinsky, Huber and Lenz (2012a), we restricted our survey to those in the United States with a 95% approval rating from other MTurk "requesters" (those who post assignments like us) on at least 50 previous assignments. Our initial HIT also told participants that "attentive and satisfactory completion of the initial survey will earn invitations to follow up surveys at higher pay rates." We also advised respondents that "we check responses to ensure that [they] read the questions and respond carefully." Such recruitment restrictions and clear wording regarding the length of the study lead to more reliable samples across the waves and less respondent drop-off on each survey.

The initial HIT must also include a link to take participants recruited in MTurk to online survey hosted elsewhere (e.g., Qualtrics or SurveyGizmo), the generic password for entering it, if password protected, and a box for participants to enter the unique code which serves as a receipt they will get at the end of the survey. We recommend explicitly telling participants both in the HIT and at the end of the survey to return to the HIT and enter the random number displayed at the end of the survey to claim their compensation and approval. This last part is critical because it is necessary for a) determining who satisfactorily completed the assignment for payment, and b) linking a participant's surveys to each other for the panel. This step is necessary in part because Amazon is the intermediary for all connections between researchers and participants, which provides anonymity to the participants. Researchers ("requesters") fund an Amazon account with a credit card and then Amazon deposits money in participants' accounts after assignments are approved. All the researcher knows about participants is their random ID number. Both SurveyGizmo and Qualtrics will produce a unique ID for the respondent on the last page of the survey, which also appears as a variable in the survey output. Respondents can then copy and paste or type this receipt into MTurk. In the ACA study, we recruited participants straight into our first wave. Alternatively, one can conduct a screener survey prior to the full survey to ensure that those taking the survey meet particular characteristics, which would be appropriate for oversampling from particular populations (e.g., based on demographic questions) or limiting the sample to particular populations (e.g., based on geographic regions) or launching side-by-side panels around different events without repeat participants. Such screener surveys generally cost only a few cents per respondent.

<sup>1</sup>The authors' names are listed alphabetically.

<sup>1</sup>We were able to conduct our first panel studies with few complications in part by utilizing the work of Berinsky, Huber and Lenz (2012a) and their supplemental documentation online (Berinsky, Huber and Lenz, 2012b,c).

At the end of the first wave you will have two spreadsheets: one downloaded from the survey software with all of the responses, and one downloaded from MTurk. To get the latter, log in as “requester,” then select “manage,” then click on the “batch” you want, and then choose “download csv.” The next step is deciding whom to approve and whom to keep in the panel. This requires matching rows in the survey data to rows in the MTurk participant file using the unique ID that participants entered into the MTurk HIT and which appears in the “answer” column at the far right of the MTurk export file. Matching with the unique identifier can be done in any statistical program or spreadsheet. We chose to compensate and approve anyone with a successful match and those who appeared to make a good faith effort to enter a proper unique ID (even those that may have made a typo or otherwise gotten confused). While you can compensate those without a perfect match, you can only keep those for whom you can match an MTurk entry to a survey entry in the panel.

In addition, you may want to include a few trick screener questions in the first survey to identify those who are not paying any attention and simply checking boxes (Berinsky, Margolis and Sances, 2012). Because some *Turkers* may be multitasking, expecting little oversight or exhausted from previous MTurk tasks, you are likely to have some survey satisficing.<sup>2</sup> For example, in the ACA study, embedded in a series of multiple choice political knowledge questions, such as “what government position does John Roberts hold,” we asked: “what government position (Senator, Chief Justice, Speaker of the House, Secretary of State, None of the Above) do you currently hold?” We deemed participants who failed all three screeners unsatisfactory and dropped them from the data and from the recruitment list for subsequent waves, since they were likely not paying attention at various points in the survey.

At the end of this process you will be able to create a simple master panel file which comprises MTurk ID numbers in the first column and their corresponding wave one survey IDs in the second column. As the panel progresses you will simply tack on additional columns of survey IDs after subsequent waves by matching to the MTurk IDs in the corresponding MTurk output files. The final step in the first wave, though it is not absolutely necessary, is giving all of those we wanted to invite back for the second wave a custom MTurk “qualification,” which we called “complete 1.” You can create and bulk-assign a custom qualification by uploading a spreadsheet of MTurk IDs within the “manage” function. MTurk will allow you to make this qualification a requirement to view the second wave HIT which will prevent

other *Turkers* from accidentally trying to join the panel.

The only difference between the first wave and subsequent waves is that instead of posting a public HIT to recruit, you will need to have Amazon send emails to your chosen participants inviting them to the next wave. You can do this automatically with a script. We followed Berinsky, Huber and Lenz (2012c) and used a simple Perl code to feed it a text file of MTurk IDs (all of the “live” participants from the first column of our master panel spreadsheet) and the contents of an email which will be sent from the address associated with your Amazon account to each participant.<sup>3</sup> You will want to include a subject line such as “invitation to follow up survey for \$1.50” and include a link to the proper survey and the password to enter it in the text (and, probably, a friendly and gracious note). We placed our own MTurk ID (which you will get when signing up) at the top and bottom of text file so that we would get the first and last emails to verify that the Perl script was working correctly.

For each wave after the first, you will need to create a new HIT so that participants can enter the new code they will get at the end of the survey. This HIT is no longer for recruiting but is still needed for collecting survey IDs to compensate participants and for continuing the panel. On that note, we found that participants occasionally had trouble accessing the HIT, and thus we ended up creating a main HIT as well as one or two backups (with smaller Ns) for those who got locked out of the primary HIT. Doing so simply means that you will have to download and append multiple CSVs from MTurk for a given wave. It is likely that a few people will recognize that they made a mistake entering their unique ID and email you. As long as they send a MTurk ID and a survey code, you can update your master sheet manually. We recommend double-checking all non-matches manually, as some will not match due to formatting issues, like leading or lagging spaces (we encountered approximately five per wave like this).

You can make the HIT private by including a custom qualification (above) as a requirement for accepting the HIT. Importantly, you will want to make sure you can view the HIT from your own account, so that you can copy the unique URL and include it on the last page of the survey and/or in the email, which points participants to the URL where they can enter their codes. In the design phase you will have a choice of whether to allow those who do not meet the requirements (including yourself) to still be able to view a HIT. The procedure for subsequent waves is the same, but with a new custom qualification, new private HIT, and new list of MTurk IDs for the Perl script to send the emails. By the end of the process you should have a master participant

<sup>2</sup>We recommend reading little, if anything, into the “time to completion” data from MTurk. This number simply represents how long it took for respondents to enter the survey receipt after opening the HIT. It is an accurate indicator of time only for those who open the HIT, go straight to the survey, and then go straight back to MTurk. If one refreshes or reopens the MTurk HIT for example, their time to completion could be a few seconds. This becomes even more relevant later when participants access the survey via an email and only go to the HIT at the end.

<sup>3</sup>You can also update your ID file partway through a wave to send a reminder email.



file matching MTurk IDs with each survey they completed and output files for each wave from the survey program, i.e., panel data.

## Panel Demographics and Attrition

The primary concern in conducting any study with a sample of *Turkers* is representativeness. As with any online environment, one expects a younger, more liberal, and more tech-

nology savvy participant pool than the population at large. These demographic traits and other features of MTurk samples are thoroughly described and analyzed in Berinsky, Huber and Lenz (2012a), and our initial ACA study sample largely bolsters their cross-sectional findings. Before turning to the evolution of our sample over the panel waves, we briefly describe our first wave sample demographics with an eye towards the potential for unrepresentativeness.

Table 1: Sample Demographics and Comparison with Other Surveys

Variable	Our Sample	Internet		Face to Face	
		BHL MTurk	ANES-P 2008-09	CPS 2008	ANES 2008
% Female	54.4	60.1	57.6	51.7	55.0
% White	79.0	83.5	83.0	81.2	79.1
% Black	7.9	4.4	8.9	11.8	12.0
% Hispanic	5.0	6.7	5.0	13.7	9.1
Age (years)	33.4	32.3	49.7	46.0	46.6
Party ID (mean 7 pt.)	3.2	3.5	3.9		3.7
Ideology (mean 7 pt.)	3.3	3.4	4.3		4.2
Education	50% Coll Grad 37% Some Coll	14.9 yrs	16.2 yrs	13.2 yrs	13.5 yrs
Income (median)	30-49K	45K	67.5K	55K	55K

*Traits for our sample from wave 1 (N=1242), “BHL MTurk” = Berinsky, Huber and Lenz (2012a), ANES-P = American National Election Panel Study (Knowledge Networks), CPS = Current Population Survey, ANES = American National Election Study, CPS and ANES are weighted. Data from all columns other than “Our Sample” reproduced from Table 3 in Berinsky, Huber and Lenz (2012a)*

Table 1 compares our (wave one) MTurk sample to that of another MTurk sample, a high quality internet panel study, the 2008-2009 American National Election Panel Study (ANES-P) conducted by Knowledge Networks, and two gold standard traditional surveys, the 2008 Current Population Survey (CPS) and the 2008 ANES. Overall, our sample appears to closely resemble the population with a few expected deviations, especially age and education. While not as representative as the field’s best national probability samples, it outperforms the typical convenience samples. Indeed, other than being a slightly younger sample, it is comparable to the high quality ANES-P population.

Panel studies, however, introduce additional opportunities to reduce the sample representativeness with every wave. Panel attrition may be the result of participants choosing to opt out of future surveys or difficulties faced by the researcher in re-contacting them. In either case, if those in particular demographic groups or with certain attitudes leave the panel, the reduction in sample size can affect representativeness and pose a problem for inference. Because it utilizes cloud computing, MTurk makes panel attrition less likely. For one, regardless of whether panelists move—a typical problem for long-term panel studies—they

are still easy to contact through their MTurk accounts. In addition, because we can screen for Turkers with a history of carefully completed (but unrelated) projects, we can sample those who are more likely to complete new tasks.

The numbers of successfully completed responses for the four waves conducted in the weeks surrounding the ACA decisions were 1242, 944 (76% of the successful completes from the previous wave), 856 (91%), and 751 (88%). We consider these numbers to be especially high given the tight time windows we were working with. They are in line with rates in other “carefully conducted” high quality surveys (Bartels, 1999). Because we wanted to make sure all of our responses in a respective wave had the same opportunities to be exposed to relevant information, our surveys were only in the field for a few days. In fact, our second wave, the one with the largest attrition, was only in the field for 48 hours, because we wanted to stop it prior to Supreme Court decision, the date of which was not known until moments before the announcement. Four months after our initial four waves, we were able to collect responses from 472 participants (63% response rate).

While the number of responses and the speed with which we were able to collect them are impressive in their own

rights, the seemingly random attrition may be even more important for statistical inference in a panel study. Figure 1 shows the evolution of our panel's demographics. Particularly in the first four waves, collected in a one-month period around the Supreme Court decision (highlighted as the event in the figure), our panel demographics remained stable as people fell out of the panel. While there were some deviations in the fifth wave, collected four months later, our data suggest that panel attrition was essentially more or less equal across the categories of respondent traits, such as race, gender, partisanship, and income. Interestingly, slight trends are evident in age and education, with 18 and 19 year olds and those with some college education consistently falling out slightly more often than the older and more educated, respectively. It is somewhat intuitive to expect that across five months, the older and more educated would be more reliable survey respondents. In all, however, we believe that the results of our study suggest that there is little danger in the panel attrition in MTurk samples across the typical demographics.

## Leveraging the Cloud

Turkers respond to survey requests incredibly quickly, which is especially valuable when conducting surveys intended to explore how individuals respond to real events in an observational setting. Indeed, the pre-post event design has the unique potential to capture any micro-level change in this context, which is one of several typical reasons to collect panel data (see Sharot, 1991). Moreover, because events in the real world unfold quickly and often change with the discovery of new information, related events or media coverage, it is important that the panels surrounding the event are tightly situated around it. MTurk makes it possible to launch a survey and collect hundreds of responses within hours. Even in latter waves of our panel, we registered dozens of responses in the first several minutes of posting the HIT. In our case, this allowed us to end waves in anticipation of the health care decision and start a new one right after it was announced. While we did not need to, it also would have allowed us to quickly go back into the field with another wave if, for example, President Obama had made a health care speech a few days after an adverse decision. Thus, along with MTurk's low cost and ease of use, the low panel attrition and quick response times of the MTurk sample provide unusual flexibility for researchers to quickly adapt their research design to real world events. In our study of the ACA decision, scheduled weekly panels would have been much less efficient than panels quickly and easily adaptable to the event's uncertain timing. Our case is not unique; for example, one who is interested in public opinion in response to natural disasters could time surveys around forecasts and collect data in tight windows as events unfold.

A related benefit of doing MTurk panel studies around

events is the ability to do what we are calling *real-time* survey experiments, in which one randomizes participants to realistic treatments related to the event as they are simultaneously getting political information about it in their natural environments. Relative to conducting artificial experiments with captive audiences, this approach dramatically increases external validity and offers a unique opportunity to combine experimental treatments with the unfolding events in the real world, thereby reconciling some of the known discrepancies in survey and experimental studies (Hovland, 1959). Indeed, there has been some noteworthy attention to the implications of captive audiences in political science (e.g., Arceneaux and Johnson, 2010; Gaines and Kuklinski, 2011; Gaines, Kuklinski and Quirk, 2007; Levendusky, 2011) and even to realistic experiments of attitude change in the longitudinal context (Druckman, Fein and Leeper, 2012). However, outside of using an MTurk sample with an online survey platform or a convenience sample, combining a large N panel design around imprecisely timed real world events with realistic experimental treatments would likely be too costly and/or would require sacrificing external validity due to its advanced planning demands.

Our approach using MTurk allows one to not only capture the micro-level change around the event, but also to leverage any effects of the event with a related experiment. That is, we might be interested in not only attitude change as a result of an event, like the ACA decision, but also with factors associated with the event that can be experimentally manipulated, such as media frames. In our case, we took advantage of news reports that Chief Justice Roberts switched sides for political reasons that broke within days of the decisions. While a typical experiment might expose participants to this story, and this story only, weeks or months after a decision to investigate the effect of a story about politicians in robes, MTurk enabled us to expose some participants to it as it was unfolding and as they were also self-selecting other news and information about the decisions. An example of a similar application would be studying the effects of media frames leading up to and/or after presidential debates. MTurk would allow a researcher to easily conduct a pre-wave and then expose participants to different media frames in the days after the debate to estimate their effect on people who are also exposed to other uncontrolled information.

To be sure, our health care study is but one example of using MTurk to conduct an inexpensive and flexible panel study around a foreseeable political event. It enabled us to ask dozens of questions of our choosing in each wave, to adopt the timing of our surveys to events as they unfolded, and to embed experiments by exposing participants to a little-known yet important news story as it broke. Supreme Court cases are but one potential application. As we mentioned above, whether one wants to collect pre-post data around precisely scheduled events, like presidential debates

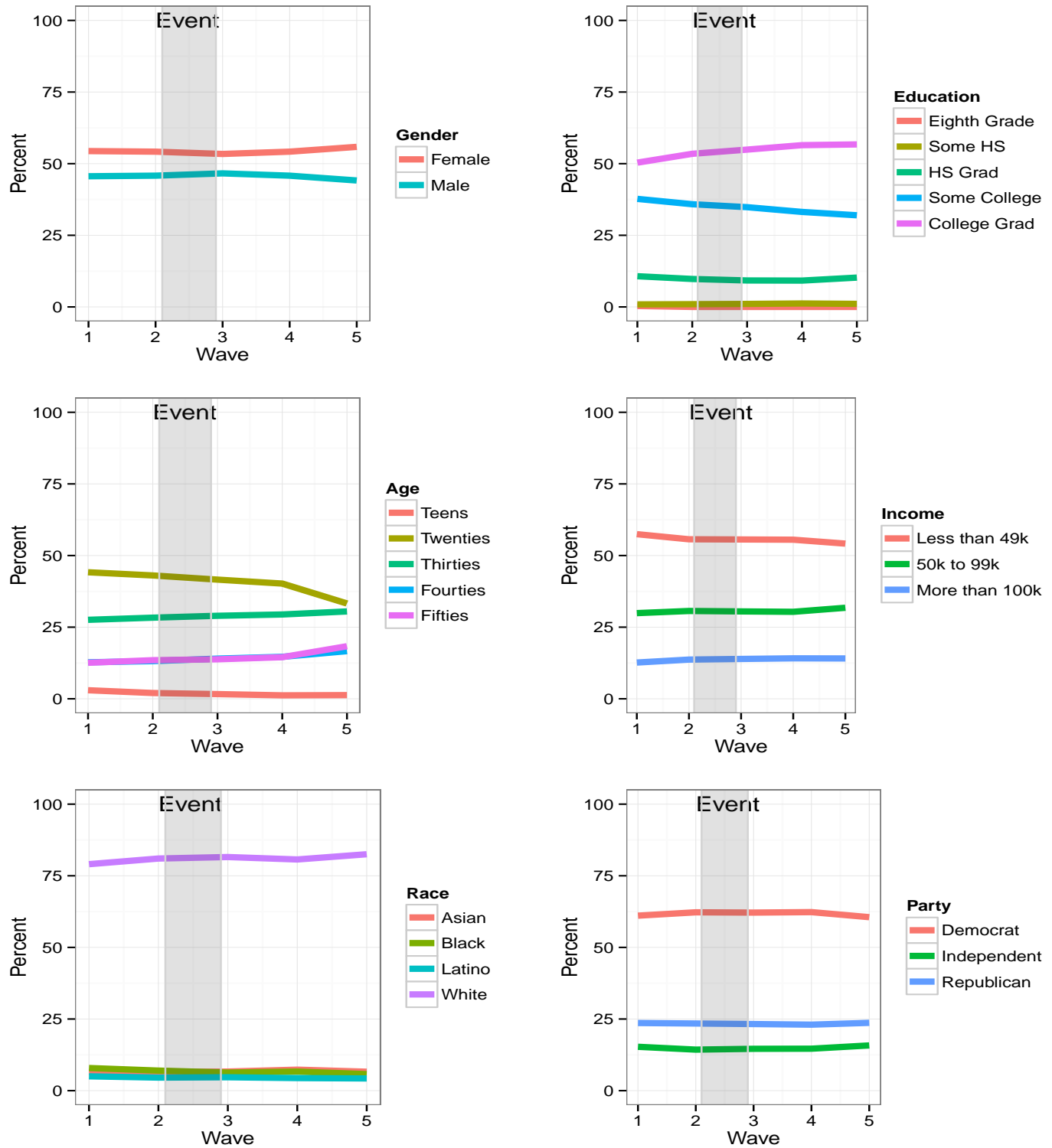
or the release of economic data, or those that can only be forecast a few days in advance, like natural disasters or major legislative votes, MTurk makes it easy to interact political events with experimental treatments in real-time. It also

allows one to simply conduct a standard panel design at a workable price. In all, MTurk makes sophisticated research designs possible for those with constrained research budgets and introduces little in the way of inferential bias.

## References

- Arceneaux, Kevin and Martin Johnson. 2010. "Does Media Fragmentation Produce Mass Polarization?" Temple University. Unpublished paper.
- Bartels, Larry M. 1999. "Panel Effects in the American National Election Studies." *Political Analysis* 8(1):1–20.
- Berinsky, Adam J., Gregory A. Huber and Gabriel S. Lenz. 2012a. "Evaluating Online Labor Markets for Experimental Research: Amazon.com's Mechanical Turk." *Political Analysis* 20(3):351–368.
- Berinsky, Adam J., Gregory A. Huber and Gabriel S. Lenz. 2012b. *Mechanical Turk Instructions*. Available at [https://docs.google.com/document/pub?id=1Bjqp0FSTmEG83S-vtbr0uFrdesq10SC27Xbp\\_iCI8Eg](https://docs.google.com/document/pub?id=1Bjqp0FSTmEG83S-vtbr0uFrdesq10SC27Xbp_iCI8Eg).
- Berinsky, Adam J., Gregory A. Huber and Gabriel S. Lenz. 2012c. *Recontacting Mechanical Turk Workers*. Available at [http://docs.google.com/View?id=dd4dxgxf\\_9g9jtdkfc](http://docs.google.com/View?id=dd4dxgxf_9g9jtdkfc).
- Berinsky, Adam J., Michele Margolis and Michael W. Sances. 2012. "Separating the Shirkers from the Workers? Making Sure Respondents Pay Attention on Internet Surveys." Presented at NYU CESS 5th Annual Experimental Political Science Conference.
- Christenson, Dino P. and David M. Glick. 2012. "Roberts's Health Care Decision Disrobed: The Microfoundations of the Court's Legitimacy." Presented at the Meeting of the Pacific Northwest Political Science Association, November 8–10, 2012. Portland, OR.
- Druckman, James N., Jordan Fein and Thomas J. Leeper. 2012. "A Source of Bias in Public Opinion Stability." *American Political Science Review* 106(2):430–54.
- Gaines, Brian J and James H. Kuklinski. 2011. "Experimental Estimation of Heterogeneous Treatment Effects Related to Self Selection." *American Journal of Political Science* 55(3):724–736.
- Gaines, Brian J., James H. Kuklinski and Paul J. Quirk. 2007. "The Logic of the Survey Experiment Reexamined." *Political Analysis* 15(1):1–20.
- Hovland, Carl I. 1959. "Reconciling Conflicting Results Derived From Experimental and Survey Studies of Attitude Change." *American Psychologist* 14(1):8.
- Levendusky, Matthew S. 2011. "Do Partisan Media Polarize Voters." University of Pennsylvania, Unpublished Paper.
- Sharot, Trevor. 1991. "Attrition and Rotation in Panel Surveys." *The Statistician* 40:325–331.

Figure 1: Sample Demographics by Panel Wave



*Event refers to the period between the second and third waves when the Supreme Court ACA decision was announced.*

University of Illinois at Urbana-Champaign  
Department of Political Science  
420 David Kinley Hall  
1407 W. Gregory Drive  
Urbana, IL 61801

*The Political Methodologist* is the newsletter of the Political Methodology Section of the American Political Science Association. Copyright 2012, American Political Science Association. All rights reserved. The support of the Department of Political Science at the University of Illinois in helping to defray the editorial and production costs of the newsletter is gratefully acknowledged.

Subscriptions to *TPM* are free for members of the APSA's Methodology Section. Please contact APSA (202-483-2512) if you are interested in joining the section. Dues are \$25.00 per year and include a free subscription to *Political Analysis*, the quarterly journal of the section.

Submissions to *TPM* are always welcome. Articles may be sent to any of the editors, by e-mail if possible. Alternatively, submissions can be made on diskette as plain ascii files sent to Wendy K. Tam Cho, 420 David Kinley Hall, 1407 W. Gregory Drive, Urbana, IL 61801. L<sup>A</sup>T<sub>E</sub>X format files are especially encouraged.

*TPM* was produced using L<sup>A</sup>T<sub>E</sub>X.



**President: Robert Franzese**

University of Michigan  
[franzese@umich.edu](mailto:franzese@umich.edu)

**Vice President: Kevin Quinn**

University of California at Berkeley, School of Law  
[kquinn@law.berkeley.edu](mailto:kquinn@law.berkeley.edu)

**Treasurer: Luke Keele**

Pennsylvania State University  
[ljk20@psu.edu](mailto:ljk20@psu.edu)

**Member-at-Large: Betsy Sinclair**

University of Chicago  
[betsy@uchicago.edu](mailto:betsy@uchicago.edu)

***Political Analysis* Editors:**

**Michael Alvarez and Jonathan Katz**

California Institute of Technology  
[rma@hss.caltech.edu](mailto:rma@hss.caltech.edu) and [jkatz@caltech.edu](mailto:jkatz@caltech.edu)