# Managing data for experimental semantics and pragmatics

Judith Degen (Stanford), Judith Tonhauser (University of Stuttgart)

April 8, 2019

## Abstract

The current DMUC reports on the organization of an experimental semantics/pragmatics project that investigated the extent to which variability in projective content's projectivity is predicted by that content's at-issueness (Tonhauser, Beaver, & Degen, 2018). The project included four web-based experiments in which participants adjusted sliding scales to provide projectivity and at-issueness ratings for close to 300 items. The workflow and best practices we report here are generalizable to any sufficiently similar web-based study, as well as to in-lab studies and dependent measures that differ substantially from the reported ones, e.g., online measures like eye movements. We also acknowledge that experimental semantics/pragmatics projects increasingly include a computational cognitive modeling component in addition to standard experimental data analysis. While the current DMUC does not include a cognitive

model, we add some remarks on how to maintain development and documentation of cognitive models towards the end of this chapter. The following also describes best practices within the first author's interActive Language Processing Lab at Stanford (ALPS) as of the time of publication. We close with some reflections on what we would change, were we to start again from scratch.

## 1. Research hypothesis: the less at-issue a content is, the more it projects

The project whose organization is described in this DMUC was published in Tonhauser et al. (2018) and addressed the following questions:

1. Is there variability in the extent to which the projective content of a lexical expression projects?
2. If there is variability, is degree of projectivity predicted by a particular item's at-issueness?

For instance, the verb *regret* is typically taken to be a factive predicate. This means that even when embedded under an entailment-canceling operator like a polar question as in (1), the content of its complement – here *that Mike visited Alcatraz* – is taken to project, i.e., the speaker of (1) is taken to be committed to Mike having visited Alcatraz.

(1) Did Felipe regret that Mike visited Alcatraz?

But while intuitions are robust regarding the projection of some projective contents, there is evidence that others vary in the robustness with which they project (Karttunen, 1971; Smith & Hall, 2011; Xue & Onea, 2011). Our goal was to investigate this potential projection variability systematically and test the hypothesis, based on previous work (Abrusán, 2011; Beaver, Roberts, Simons, & Tonhauser, 2017; Simons, Tonhauser, Beaver, & Roberts, 2010), that the at-issueness of lexical content is implicated in its projectivity.

To address the research questions formulated above, we conducted four experiments: in Exps. 1a and 1b we collected projectivity ratings and at-issueness ratings for the projective contents of two sets of lexical expressions. In Exps. 2a and 2b we collected at-issueness ratings for the same contents as in Exps. 1a and 1b using a different at-issueness diagnostic. We found that there is indeed variability in projectivity both within and across lexical expressions, and that this variability is systematically predicted by at-issueness: the less at-issue a particular projective content is, the more it projects.

## 2. Summary of tools used for data management

All experimental files, data files, and analysis files are freely available in a github repository at https://github.com/judith-tonhauser/how-projective. All of the software used to generate the experiment (html/javascript/css), manage the connection between the experiment and Mechanical Turk (submiterator), and analyze the data (R), is open source. This means that, in the spirit of open science, all experiments and analyses reported in the paper are directly reproducible. The easiest way to view the experiments locally in a browser and to run the analyses is to clone the github repository locally. To do so, the following tools must be installed: git, R. For convenience, RStudio is also highly recommended.

Once the tools are installed, the github repository may be cloned from the command line:

```
git clone https://github.com/judith-tonhauser/how-projective.git
```

This will generate a copy of the repository on your local machine. To view an experiment, open one of the experiments/expXX/experiment.html files in your preferred browser[1] (where 'XX' should be replaced with one of '1a', '1b', '2a', '2b'). To run the analyses, open and run one of the files preprocessing.R, analysis.R, or graphs.R from results/expXX/rscripts/ in RStudio, depending on whether you want to recreate the preprocessing stage of the raw data, the statistical analysis, or the visualizations reported in the paper, respectively.

## 3. Data management: the organizational framework

In this section we present the workflow and organization of the project, and offer comments on the general workflow of any experimental semantics/pragmatics project we conduct.

[1] These experiments will not work with Internet Explorer.

## 3.1 The organizational framework for collaboration

While every experimental project starts with an idea, its organization starts with the creation of a github repository which serves to store experimental materials, data, analysis scripts, and documentation. For the current project, the repository linked above accompanies the paper. This is a tidied up version of the larger project repository, which continues to evolve as we continue to work on the broader project investigating the nature of projectivity.[2] Thus, the above linked repository serves as a way for us to share all files that are required to reproduce the studies reported in the paper, but not as a time-stamped record of the development of the project in general. In contrast, the repository linked in the footnote will allow the interested reader to meticulously follow the exact timeline of the project.

In this project (and in general for ALPS Lab projects), we used git to keep track of the project's progress. Git is a version control system for tracking changes. Originally it was developed for tracking changes in software development, but it has the fortunate broader application of tracking changes in any file, which makes it useful for tracking changes in research projects generally. We additionally used

[2] The broader project repository can be found at https://github.com/judith-tonhauser/projection-NAI-variability -- this repository is not as well documented for external eyes, but we're sharing it regardless.

a dedicated Slack[3] channel to exchange messages about the project and used Slack's github integration feature. This feature allows Slack to follow any changes made to the github repository and to automatically post as messages in Slack the commit messages created by the researcher when a change is made to the repository. This allows collaborators on the project to be notified when a change is made to the repository, and to see what the nature of the change is (provided the researcher making the change has written an informative commit message). Together, git/github/Slack comprise the collaboration framework used in this project and in all of ALPS Lab's projects. Note that for the reader interested in following the structure of the project, Slack is *not* necessary, though we enthusiastically recommend it as a tool for collaborative research projects to manage communication.

## 3.2 Project repository structure

The project repository, like ALPS Lab repositories in general, has the following contents, which we encourage the reader to follow in either the online version of the repository or in their locally cloned repository:

1. A *README* markdown file in the root directory of the repository, which contains information about the repository's contents.

[3] Slack is a collaboration tool that is essentially a messaging system and allows for (groups of) users to create channels dedicated to specialized topics or research projects.

2. An *experiments* directory, which stores all the files required to run the experiments. This directory is further divided into sub-directories: one for each experiment, named appropriately, and a *_shared* directory that contains javascript libraries required for any of the experiments to run.
3. A *results* (or *analysis*) directory, which stores all the files required to analyze the collected data. This directory is further divided into sub-directories that mirror the structure of the *experiments* directory, i.e., each experiment in *experiments* has its corresponding directory in *results*. It also contains files with helper functions required by all analysis scripts (in this case, aptly named *helpers.R*) as well as a separate directory for any analyses spanning multiple experiments.
4. A *paper* (or *writing*) directory, which contains the files that generate the final submitted version of the paper. When this is a more general *writing* directory, it may contain multiple writing projects, organized as separate sub-directories.

Additional directories a project repository may routinely contain:

5. A *data* directory, if the project is complex and the researchers want to keep all relevant data files in one place.
6. A *models* directory, if the project encompasses a computational cognitive modeling (separate from the statistical analysis) component, as is frequently the case in ALPS Lab.

7. A *talks* directory, where slides from talks given about the project are stored.

Some projects may also encompass corpus searches. Depending on the status of these searches – whether they constitute the main studies of the project or whether they are used to extract information to be used in the analysis of the main studies – they may either merit their own top-level directory or be included in the *experiments* or *analysis* directories. Finally, each repository also contains a *.gitignore* file, which specifies files that git should ignore (i.e., not track changes in, and not push to the online github repository). More on this below.

We consider this to be an optimal project repository structure and have not yet encountered a project that doesn't benefit from this organizational structure. Cleanly separating the files used to generate the experiments from the analysis of the data collected in said experiments is useful because it naturally separates two aspects of the replicability pipeline: a) the reproducibility of the *experimental methodology* (including the exact task, materials, trial structure, and instructions) and b) the reproducibility (and hopefully replicability) of the *data analysis*. An external party interested in reproducing the analysis but not re-running the experiment may therefore directly operate only on the *results* directory without having to wade through the experiments directory. In contrast, an external party interested only in seeing exactly what participants' task was can easily navigate to the experiment .html file corresponding to the experiment of interest without having to search very hard.

In Sections 4 and 5 we describe the contents of the *experiments* and the *results* directories and the associated workflows, which we consider the main components of the project.

## 4. Data collection (*experiments* directory)

Each experiment in the project was run via Amazon's web service Mechanical Turk in the following way: the experiment was coded as an external website using html/javascript/css and uploaded to JT's university web space to be accessed through an external URL.[4] Using the submiterator tool (now superseded by supersubmiterator), which provides an intuitive wrapper around the Mechanical Turk command line tools, the external website was posted as a HIT to Mechanical Turk and ran until all the data was collected (no longer than a few hours for any of the experiments). Again using submiterator, the data file was downloaded from Mechanical Turk, participant IDs anonymized, and the data reformatted in such a

[4] The experiments have since been taken off of JT's former university web space. We provide a lasting example of Experiment 1a at https://web.stanford.edu/~jdegen/exp1a/experiment.html -- this is the same experiment that is viewable by the reader by opening the file experiments/exp1a/experiment.html in a browser on your local machine if you have cloned the repository as described in Section 2.

way that the resulting .csv file contained one data point per row with all the relevant information for analysis in R. This data file was then copied into the corresponding data sub-directory in the results directory, which concluded the data collection process.

## 4.1 Experiment files

We find it useful to label the directories corresponding to the individual experiments that were run in ascending order (e.g., by prefacing the directory names with '1_', '2_', etc.). This serves as a reminder of the chronological order in which the experiments were run. In this case, there is only a small number of experiments so this may seem less relevant, but for projects that require a lot of piloting, this ascending labeling is extremely useful. Alternatively, we have found it useful in other projects to include a *pilots* and *main* sub-directory that contains the pilot experiments and main experiments to be reported, respectively.

Each of the sub-directories in experiments has the same structure and consists of

- an *experiment.html* file that specifies the elements of the experiment and can be opened in a browser to view the experiment;
- a *js* directory that contains an *experiment.js* javascript file that determines the logical flow of the experiment;
- and a *css* directory that contains *.css* style files that determine stylistic elements of the experiment.

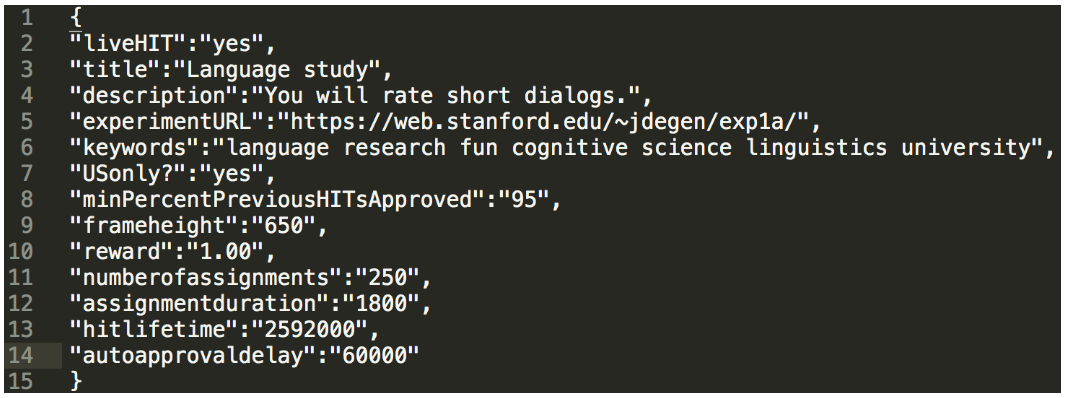
While generating experiments directly using html/javascript/css initially involves a learning curve for the user uninitiated in web programming, we highly recommend it over using the Mechnical Turk templates or out-of-the box services like Qualtrics. The reason for this is simple: coding up one's own experiments allows for far more flexibility and control regarding tasks and experimental design than out-of-the-box services do. While the currrent experiment involved a simple continuous slider rating task in a two-block design with randomization of trials within blocks, we have used the same general infrastructure to run truth-value judgment studies, response time studies, self-paced reading studies, perception studies, free and forced production studies, studies involving drag-and-drop functionality, and many others. Example templates for such studies within the same framework used for this study are provided at https://github.com/thegricean/LINGUIST245B. We encourage interested readers to use the files we have provided as the basis for their own experiments. The internet abounds with good web programming tutorials to aid in the process of modifying the experiments.

## 4.2 Interfacing with Mechanical Turk

Once the experiments were ready to be deployed, we used submiterator, a python program originally written by Dan Lassiter and extended by Erin Bennett, to post the experiment to Mechanical Turk, subsequently retrieve the data, and anonymize participants' worker IDs.[5] This tool has since been superseded by supersubmiterator, written by Sebastian Schuster, which has the following convenient features (features not in common with submiterator are bolded):

1. One simple command each for a) posting the experiment, b) retrieving the results, and c) reformatting the Mechanical Turk results file (which comes back as one participant's data per row) into a .csv file with one data point per row for easy analysis in R with already anonymized worker IDs.
2. **No interaction with the no longer supported Mechanical Turk Command Line Tools necessary. Simply specify a .config file with all relevant information about the HIT (see supersubmiterator documentation for more details, and see Figure 1 for an example .config file for Experiment 1a).**
3. **Built-in batch support which makes sure that the total number of assignments (participants) is spread over batches of no more than 9 each. This is relevant because Amazon charges a 40% fee for HITs with more than 9 assignments, but only 20% for HITs with up to 9 assignments. The reformatting command automatically generates one merged data file for analysis.**

[5] A Mechanical Turk worker ID is a unique identifier associated with a participant's, linked to their email address, and thus provides personal identifying information. These IDs must therefore never be posted online.



```
{
"liveHIT":"yes",
"title":"Language study",
"description":"You will rate short dialogs.",
"experimentURL":"https://web.stanford.edu/~jdegen/exp1a/",
"keywords":"language research fun cognitive science linguistics university",
"USonly?":"yes",
"minPercentPreviousHITsApproved":"95",
"frameheight":"650",
"reward":"1.00",
"numberofassignments":"250",
"assignmentduration":"1800",
"hitlifetime":"2592000",
"autoapprovaldelay":"60000"
}
```

*Figure 1. Example .config file for posting experiments to Mechanical Turk with (super)submiterator.*

Running supersubmiterator requires only installing the tool. Running submiterator additionally requires installing the no longer supported Mechanical Turk Command Line Tools (highly discouraged). Additionally, any requirements associated with Mechanical Turk studies generally – including having an Amazon Payments account and a Mechanical Turk requester account – are necessary. Because using supersubmiterator means that multiple HITs with 9 participants are generated, the issue of how to prevent participants from taking the experiment multiple times is relevant. Fortunately, the Unique Turker service provides just that functionality. Setup notes are provided on the first author's LINGUIST 245B "Methods in Psycholinguistics" github repository along with setup notes for the rest of the web-based experiment pipeline used in ALPS Lab and useful Mechanical Turk related tips.

For the current project, the experiments for which were run before the existence of supersubmiterator, the steps involved in running each experiment were:

1. Program experiment as external website and copy it to web space.
2. Create *mturk* directory inside the *expXX* directory where all Mechanical Turk related files are stored and from where the submiterator command will be called.
3. Create *experiment.config* file (see example in Figure 1) in *mturk*.
4. Run from command-line to post experiment to Mechanical Turk:
   ```
   submiterator posthit experiment
   ```
5. Run from command-line to retrieve data from Mechanical Turk:
   ```
   submiterator getresults experiment
   ```
6. Run from command-line to reformat data for worker ID anonymization and easy analysis in R:
   ```
   submiterator reformat experiment
   ```
7. Copy generated data file *experiment.csv* to its corresponding *data* directory in *results* directory.

Step 7 is necessary because the project repository's *.gitignore* file specifies that all files inside of directories called *mturk* are to be ignored. This is a safety measure to ensure that deanonymized participant data does not end up visible to the public on the internet. It does introduce one step of potential human error if the wrong file is copied, or copied to the wrong place. We consider this preferable to the potential for exposing deanonymized data.

## 5. Data analysis (*results* directory)

For this project, and in ALPS Lab generally, we used the statistical software package R (R Core Team, 2017) in conjunction with the highly recommended developer environment RStudio (RStudio Team, 2016) to analyze the data. Useful packages we used in this project (1-4 we use routinely):

1. *lme4* (Bates, Maechler, & Dai, 2009) for conducting mixed effects analyses.
2. *brms* (Bürkner, 2017) for conducting Bayesian mixed effects analyses.
3. *tidyverse* (Wickham, NA) for tidy data wrangling in R.
4. *ggplot2* (Wickham, 2009) for data visualization.
5. *lsmeans* (Lenth, 2016) for computing pairwise comparisons.

The directory structure of *results* roughly mirrors that of *experiments*, in that each experiment receives its own directory in *results*. Each experiment directory further contains sub-directories *data*, *graphs*, and *rscripts*, which cleanly separates data files (e.g., the *experiment.csv* file generated by `submiterator reformat`), R analysis files,[6] and graphs generated in the analysis process.

[6] Researchers may of course use different statistical software packages for data analysis. For example, for someone who uses python wanting to replicate our workflow, the only difference should be in the *rscripts* directory. We have no recommendations for researchers using SPSS or other dropdown menu software packages except to switch to R or python, for the simple reason that they offer more control, increased reproducibility of analysis steps, and comparatively easy identification of analysis errors.

The rscripts directory contains separate R scripts for preprocessing the raw data from Mechanical Turk (mostly for the purpose of performing exclusions, *preprocessing.R*), creating useful visualizations that are saved to the *graphs* sub-directory (*graphs.R*), and analyzing the data using mixed effects models (*analysis.R*). All scripts should contain enough comments to allow the reader to reproduce the analyses reported in the paper and to follow what was done at each step. This particular way of separating the scripts is not necessary, though it can be helpful to separate preprocessing, visualization, and analysis.

## 6. Concluding remarks

We believe the above reported organizational framework for web-based studies is a useful one: it cleanly separates relevant components of the reproducibility pipeline and openly stores all of said components for others to reproduce and build off of, with the exception of Mechanical Turk related files that contain confidential participant information.

### 6.1 What would we change?

In general, we consider this an instance of a well-organized and easily shareable project that we recommend as a model to others. However, given the rapid pace at which discussions regarding best practices in open and transparent science are progressing, we have already identified one key issue that we would handle differently, were we to start again from scratch: preregistration. The Open Science

Framework makes it easy to preregister hypotheses, experimental design, and analyses (Foster & Deardorff, 2017). The literature suggests that preregistration reduces the potential for many pitfalls in scientific practice, including avoiding problematic researcher degrees of freedom in fiddling with exclusion criteria, determining stopping criteria for running participants, and dreaming up exploratory analyses that are framed as planned analyses in the resulting papers (Roettger, 2019; Simmons, Nelson, & Simonsohn, 2011). Indeed, in current follow-ups to this project we have pre-registered our hypotheses (see OSF preregistration at https://osf.io/hn8px/), and the policy we have implemented in ALPS Lab is that any hypothesis-driven experiment must be preregistered. This excludes experiments that only serve norming purposes or are acknowledged to be exploratory experiments that serve a subsequent hypothesis generation purpose.

### 4.2 Generalization to other types of studies

The organizational framework we reported in this paper generalizes to any project using web-based experiments. Experience suggests that it also easily generalizes to projects that use computational cognitive modeling or corpus searches in addition to (or instead of) behavioral experiments. It also generalizes to lab-based experiments and experiments conducted in the field – storing experimental scripts is a general property of the framework and is not specific to web-based or even more specifically Mechanical Turk experiments. An issue arises when proprietary software is used, e.g. EPrime. In this case, less material can be shared. We consider this a reason to use open source software whenever possible.

A different problem arises for studies that require storing very large datasets, as is often the case with eye-tracking or ERP studies. Github has limits on how much data can be stored within one repository: no one file can be larger than 100MB; repositories have a hard limit of 100GB; and it is recommend that repositories be kept under 1GB in size. One option is to use Git Large File Storage (Git-LFS). Another option is to use databases. And yet another option is to upload downsampled or otherwise compressed data files and keep the raw data files backed up locally.

### 4.3 Conclusion

This DMUC described data management practices for an experimental semantics/pragmatics project investigating the projectivity of various projective contents via web-based Mechanical Turk experiments. All project materials are openly accessible here.

## References

Abrusán, M. (2011). Predicting the presuppositions of soft triggers. *Linguistics and Philosophy*, *34*(6), 491–535. https://doi.org/10.1007/s10988-012-9108-y

Bates, D., Maechler, M., & Dai, B. (2009). lme4: linear mixed-effects models using S4 classes. R package version 0.999375-31.

Beaver, D. I., Roberts, C., Simons, M., & Tonhauser, J. (2017). Questions Under Discussion: Where Information Structure Meets Projective Content. *Annual Review of Linguistics*, *3*, 265–284. https://doi.org/10.1146/annurev-

linguistics-011516-033952
Bürkner, P.-C. (2017). brms: An R Package for Bayesian Multilevel Models Using {Stan}. *Journal of Statistical Software*, *80*(1), 1–28. https://doi.org/10.18637/jss.v080.i01
Foster, E. D., & Deardorff, A. (2017). Open Science Framework (OSF). *Journal of the Medical Library Association*, *105*(2), 203–206. https://doi.org/10.5195/JMLA.2017.88
Karttunen, L. (1971). Implicative Verbs. *Language*, *47*(2), 340–358.
Lenth, R. V. (2016). Least-Squares Means: The R Package lsmeans. *Journal of Statistical Software*, *69*(1), 1–33. https://doi.org/10.18637/jss.v069.i01
R Core Team. (2017). R: A language and environment for statistical computing. Retrieved from https://www.r-project.org
Roettger, T. B. (2019). Researcher degrees of freedom in phonetic research. *Laboratory Phonology: Journal of the Association for Laboratory Phonology*, *10*(1). https://doi.org/10.5334/labphon.147
RStudio Team. (2016). RStudio: Integrated Development ENvironment for R.
Simmons, J. P., Nelson, L. D., & Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, *22*(11), 1359–1366. https://doi.org/10.1177/0956797611417632
Simons, M., Tonhauser, J., Beaver, D., & Roberts, C. (2010). What projects and why. *Semantics and Linguistic Theory*, *20*, 309–327. Retrieved from http://elanguage.net/journals/salt/article/download/20.309/1326
Smith, E. A., & Hall, K. C. (2011). Projection diversity: Experimental evidence. *Proceedings of the Workshop on Projective Meaning*. Retrieved from http://www.ling.osu.edu/~kchall/Smith_Hall_ESSLLI_presentation_2011.pd f
Tonhauser, J., Beaver, D. I., & Degen, J. (2018). How Projective is Projective Content? Gradience in Projectivity and At-issueness. *Journal of Semantics*, (August), 1–48. https://doi.org/10.1093/jos/ffy007
Wickham, H. (n.d.). tidyverse: Easily Install and Load the "Tidyverse."
Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. Retrieved from http://ggplot2.org
Xue, J., & Onea, E. (2011). Correlation between presupposition projection and at-issueness: An empirical study. *Proceedings of the ESSLLI 2011 Workshop on Projective Meaning*, 171–184.