

Government Data

Bill Hunt
@krues8dr

I'm here today to talk to you about some fun things you can do with government data. Depending on how you define "fun."

Government Data



I've spent the last four years focused on opening up legal data for citizen engagement. These are the two main projects I've been focused on: The State Decoded, which shows the law online, and Madison, which does the same for legislation. These projects are used all over the world today to help the people have a voice in their government. Now, to put these sites together, we had to solve a lot of different types of problems - technical, legal, and most importantly, culture.

Government Data

- Almost never available via an API
- Rarely standards-compliant
- Usually proprietary format (Microsoft Excel, etc.)
- Frequently “dirty”, Not normalized across sources
- Can’t easily be shared or bookmarked

With regard to the technical issues, we have to start with the fact that government data, on the whole, is just not very good. When you go to a government website to get data, most of the time you’re going to spend a lot of time digging for what you’re looking for. You learned about APIs in your reading already – but you’re not going to see many APIs right now for government data. Instead, you’re probably going to find an Excel spreadsheet or a PDF with a couple of rows of unhelpful headers at the top and filled with dozens of acronyms and abbreviations that are incomprehensible. Once you get past that, you’re going to discover that every file has a slightly different format, and there’s no simple way to compare data from different sources. The government mostly doesn’t use a consistent naming or numbering system between departments, so if you want to combine data from different places, it’s almost impossible to do so. A lot of government websites still use outdated systems for sharing these files, which frequently make it hard to share links to this data or bookmark it for later.

Open Data Projects

- Release clean, normalized data in standard formats
- Present accessible user interfaces & visualizations for existing data in way that people can understand
- Create new ways of viewing, searching, and interacting with the data - charts, tables, maps, APIs, etc.

Most open data projects seek to solve those problems. The majority of them present the information in a way that is intended to be easily understood. Many go further and allow users to interact with the data in an entirely new way. This can be a single blog post or news article where a journalist digs into a particular piece of data to find trends to share them. Or it may be an entire toolkit to let users manipulate data on their own to find or follow changes.

Open Data Projects

- **The State Decoded** takes XML from government sources (or their legal publishers), cleans it, and displays it in a user-friendly, accessible format.
- Allows users to comment on laws, and send those comments directly to lawmakers.



Going back to one of my own projects, The State Decoded puts the law online. In many places, you still can't view the law online, or it's behind a paywall or some terms and conditions that make it unusable, or the website is not accessible to users with visual impairment, or any number of other barriers to free use of the law. And even in places where you can get access to the law online, the official copy is almost always the printed version, not the digital version, but that's another talk. So, for our project, we get the legal code provided by government publishers as some really awful XML, transform it into cleaned up data, and then show that on a website. Along the way, we add inline definitions, cross-references, a search system, and lots of other useful things – all using the original legal code as a source. We then add a commenting system on top of the law web pages to let users leave comments for lawmakers.

Open Data Projects

How to Open Data:

1. Get some data.
2. Clean up the data.
3. Put it online.
4. ???

So, here are the basic steps that we've been using to create open data projects in the past. First you need to get some data. Then you're probably going to have to clean it up a *lot*. Then you put it online in some form. After that comes, well, a lot of complicated concerns that we'll come back to in a bit.

Sources of Data

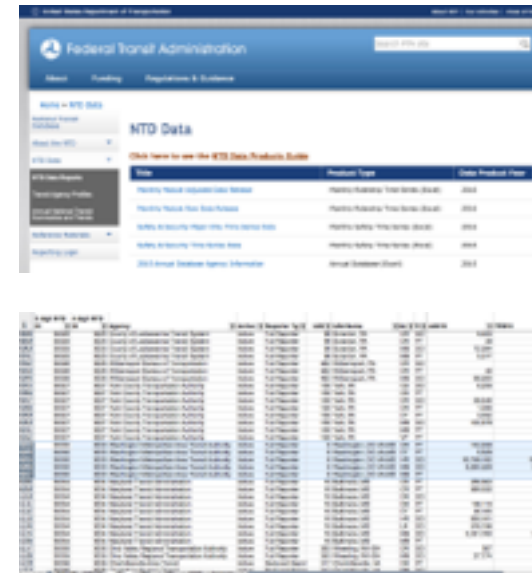
Data may be released by the government voluntarily, sometimes in press releases



So, if you need to get some data, what are your options? The first place to look is online. Frequently, agencies will post blog posts, tweets, and other media pushes voluntarily to gain attention for their services. This can be a handy way of getting information about specific facts that you're interested in. Here's an example from last week, comparing WMATA's ridership from the Inauguration and the Women's March on the following day – this tweet is from a Washington Post reporter, who got the data from a WMATA press release.

Sources of Data

Agencies may be required to report data due to regulations.



The screenshot shows the Federal Transit Administration (FTA) NTD Data website. The header includes the FTA logo and navigation links. The main content area displays a table titled "NTD Data" with columns for "State", "Provider Type", and "Data Period". The table lists various transit agencies and their corresponding data periods. Below the table, there is a link to "2013 Annual Data Report".

State	Provider Type	Data Period
Alabama	Heavy Rail	2013
Alabama	Light Rail	2013
Alabama	Bus Rapid Transit	2013
Alabama	Commuter Rail	2013
Alabama	Other	2013
Alaska	Heavy Rail	2013
Alaska	Light Rail	2013
Alaska	Bus Rapid Transit	2013
Alaska	Commuter Rail	2013
Alaska	Other	2013
Arizona	Heavy Rail	2013
Arizona	Light Rail	2013
Arizona	Bus Rapid Transit	2013
Arizona	Commuter Rail	2013
Arizona	Other	2013
Arkansas	Heavy Rail	2013
Arkansas	Light Rail	2013
Arkansas	Bus Rapid Transit	2013
Arkansas	Commuter Rail	2013
Arkansas	Other	2013
California	Heavy Rail	2013
California	Light Rail	2013
California	Bus Rapid Transit	2013
California	Commuter Rail	2013
California	Other	2013
Colorado	Heavy Rail	2013
Colorado	Light Rail	2013
Colorado	Bus Rapid Transit	2013
Colorado	Commuter Rail	2013
Colorado	Other	2013
Connecticut	Heavy Rail	2013
Connecticut	Light Rail	2013
Connecticut	Bus Rapid Transit	2013
Connecticut	Commuter Rail	2013
Connecticut	Other	2013
Delaware	Heavy Rail	2013
Delaware	Light Rail	2013
Delaware	Bus Rapid Transit	2013
Delaware	Commuter Rail	2013
Delaware	Other	2013
District of Columbia	Heavy Rail	2013
District of Columbia	Light Rail	2013
District of Columbia	Bus Rapid Transit	2013
District of Columbia	Commuter Rail	2013
District of Columbia	Other	2013
Florida	Heavy Rail	2013
Florida	Light Rail	2013
Florida	Bus Rapid Transit	2013
Florida	Commuter Rail	2013
Florida	Other	2013
Georgia	Heavy Rail	2013
Georgia	Light Rail	2013
Georgia	Bus Rapid Transit	2013
Georgia	Commuter Rail	2013
Georgia	Other	2013
Hawaii	Heavy Rail	2013
Hawaii	Light Rail	2013
Hawaii	Bus Rapid Transit	2013
Hawaii	Commuter Rail	2013
Hawaii	Other	2013
Idaho	Heavy Rail	2013
Idaho	Light Rail	2013
Idaho	Bus Rapid Transit	2013
Idaho	Commuter Rail	2013
Idaho	Other	2013
Illinois	Heavy Rail	2013
Illinois	Light Rail	2013
Illinois	Bus Rapid Transit	2013
Illinois	Commuter Rail	2013
Illinois	Other	2013
Indiana	Heavy Rail	2013
Indiana	Light Rail	2013
Indiana	Bus Rapid Transit	2013
Indiana	Commuter Rail	2013
Indiana	Other	2013
Iowa	Heavy Rail	2013
Iowa	Light Rail	2013
Iowa	Bus Rapid Transit	2013
Iowa	Commuter Rail	2013
Iowa	Other	2013
Kansas	Heavy Rail	2013
Kansas	Light Rail	2013
Kansas	Bus Rapid Transit	2013
Kansas	Commuter Rail	2013
Kansas	Other	2013
Kentucky	Heavy Rail	2013
Kentucky	Light Rail	2013
Kentucky	Bus Rapid Transit	2013
Kentucky	Commuter Rail	2013
Kentucky	Other	2013
Louisiana	Heavy Rail	2013
Louisiana	Light Rail	2013
Louisiana	Bus Rapid Transit	2013
Louisiana	Commuter Rail	2013
Louisiana	Other	2013
Maine	Heavy Rail	2013
Maine	Light Rail	2013
Maine	Bus Rapid Transit	2013
Maine	Commuter Rail	2013
Maine	Other	2013
Maryland	Heavy Rail	2013
Maryland	Light Rail	2013
Maryland	Bus Rapid Transit	2013
Maryland	Commuter Rail	2013
Maryland	Other	2013
Massachusetts	Heavy Rail	2013
Massachusetts	Light Rail	2013
Massachusetts	Bus Rapid Transit	2013
Massachusetts	Commuter Rail	2013
Massachusetts	Other	2013
Michigan	Heavy Rail	2013
Michigan	Light Rail	2013
Michigan	Bus Rapid Transit	2013
Michigan	Commuter Rail	2013
Michigan	Other	2013
Minnesota	Heavy Rail	2013
Minnesota	Light Rail	2013
Minnesota	Bus Rapid Transit	2013
Minnesota	Commuter Rail	2013
Minnesota	Other	2013
Mississippi	Heavy Rail	2013
Mississippi	Light Rail	2013
Mississippi	Bus Rapid Transit	2013
Mississippi	Commuter Rail	2013
Mississippi	Other	2013
Missouri	Heavy Rail	2013
Missouri	Light Rail	2013
Missouri	Bus Rapid Transit	2013
Missouri	Commuter Rail	2013
Missouri	Other	2013
Montana	Heavy Rail	2013
Montana	Light Rail	2013
Montana	Bus Rapid Transit	2013
Montana	Commuter Rail	2013
Montana	Other	2013
Nebraska	Heavy Rail	2013
Nebraska	Light Rail	2013
Nebraska	Bus Rapid Transit	2013
Nebraska	Commuter Rail	2013
Nebraska	Other	2013
Nevada	Heavy Rail	2013
Nevada	Light Rail	2013
Nevada	Bus Rapid Transit	2013
Nevada	Commuter Rail	2013
Nevada	Other	2013
New Hampshire	Heavy Rail	2013
New Hampshire	Light Rail	2013
New Hampshire	Bus Rapid Transit	2013
New Hampshire	Commuter Rail	2013
New Hampshire	Other	2013
New Jersey	Heavy Rail	2013
New Jersey	Light Rail	2013
New Jersey	Bus Rapid Transit	2013
New Jersey	Commuter Rail	2013
New Jersey	Other	2013
New Mexico	Heavy Rail	2013
New Mexico	Light Rail	2013
New Mexico	Bus Rapid Transit	2013
New Mexico	Commuter Rail	2013
New Mexico	Other	2013
New York	Heavy Rail	2013
New York	Light Rail	2013
New York	Bus Rapid Transit	2013
New York	Commuter Rail	2013
New York	Other	2013
North Carolina	Heavy Rail	2013
North Carolina	Light Rail	2013
North Carolina	Bus Rapid Transit	2013
North Carolina	Commuter Rail	2013
North Carolina	Other	2013
North Dakota	Heavy Rail	2013
North Dakota	Light Rail	2013
North Dakota	Bus Rapid Transit	2013
North Dakota	Commuter Rail	2013
North Dakota	Other	2013
Ohio	Heavy Rail	2013
Ohio	Light Rail	2013
Ohio	Bus Rapid Transit	2013
Ohio	Commuter Rail	2013
Ohio	Other	2013
Oklahoma	Heavy Rail	2013
Oklahoma	Light Rail	2013
Oklahoma	Bus Rapid Transit	2013
Oklahoma	Commuter Rail	2013
Oklahoma	Other	2013
Oregon	Heavy Rail	2013
Oregon	Light Rail	2013
Oregon	Bus Rapid Transit	2013
Oregon	Commuter Rail	2013
Oregon	Other	2013
Pennsylvania	Heavy Rail	2013
Pennsylvania	Light Rail	2013
Pennsylvania	Bus Rapid Transit	2013
Pennsylvania	Commuter Rail	2013
Pennsylvania	Other	2013
Rhode Island	Heavy Rail	2013
Rhode Island	Light Rail	2013
Rhode Island	Bus Rapid Transit	2013
Rhode Island	Commuter Rail	2013
Rhode Island	Other	2013
South Carolina	Heavy Rail	2013
South Carolina	Light Rail	2013
South Carolina	Bus Rapid Transit	2013
South Carolina	Commuter Rail	2013
South Carolina	Other	2013
South Dakota	Heavy Rail	2013
South Dakota	Light Rail	2013
South Dakota	Bus Rapid Transit	2013
South Dakota	Commuter Rail	2013
South Dakota	Other	2013
Tennessee	Heavy Rail	2013
Tennessee	Light Rail	2013
Tennessee	Bus Rapid Transit	2013
Tennessee	Commuter Rail	2013
Tennessee	Other	2013
Texas	Heavy Rail	2013
Texas	Light Rail	2013
Texas	Bus Rapid Transit	2013
Texas	Commuter Rail	2013
Texas	Other	2013
Utah	Heavy Rail	2013
Utah	Light Rail	2013
Utah	Bus Rapid Transit	2013
Utah	Commuter Rail	2013
Utah	Other	2013
Vermont	Heavy Rail	2013
Vermont	Light Rail	2013
Vermont	Bus Rapid Transit	2013
Vermont	Commuter Rail	2013
Vermont	Other	2013
Virginia	Heavy Rail	2013
Virginia	Light Rail	2013
Virginia	Bus Rapid Transit	2013
Virginia	Commuter Rail	2013
Virginia	Other	2013
Washington	Heavy Rail	2013
Washington	Light Rail	2013
Washington	Bus Rapid Transit	2013
Washington	Commuter Rail	2013
Washington	Other	2013
West Virginia	Heavy Rail	2013
West Virginia	Light Rail	2013
West Virginia	Bus Rapid Transit	2013
West Virginia	Commuter Rail	2013
West Virginia	Other	2013
Wisconsin	Heavy Rail	2013
Wisconsin	Light Rail	2013
Wisconsin	Bus Rapid Transit	2013
Wisconsin	Commuter Rail	2013
Wisconsin	Other	2013
Wyoming	Heavy Rail	2013
Wyoming	Light Rail	2013
Wyoming	Bus Rapid Transit	2013
Wyoming	Commuter Rail	2013
Wyoming	Other	2013

Now, if you need more general or more granular data, you can start looking at what else the agency publishes. Many government agencies are required to report data due to federal, state, or local regulations. Here's some of the first government data I worked on, as a federal contractor. The data here is from the FTA, showing information aggregated from local transit agencies. The FTA regulations require that every city's transit agency report specific pieces of information that are used to track safety and security – everything from accidents to the number of riders to miles of rail track to income. Data is recorded for monthly periods and broken down by the type of service – rail, bus, and so forth. Now if the data is not in this spreadsheet, there's a good chance the agency isn't keeping track of it. Or they keep track of it internally but are under no obligation to publish it by federal regulations. So how can we find out something that's not here?

Sources of Data

FOIA requests can be used by the public to get access to data.

Freedom of Information Act (FOIA), 5 U.S.C. § 552

We can use a FOIA request to ask the government to release information that they don't normally release. The term FOIA comes from the name of the act, the Freedom of Information Act, which was signed into law in 1966 by President Lyndon B. Johnson. Yes, it's been around that long. To make a FOIA request, you simply must send to the FOIA office of the agency in writing the information you want. That's really it.

Now, even with a proper FOIA request, an agency doesn't necessarily have to tell you the thing you want to know. For instance, if the agency doesn't keep track of the information in question, they don't have to start keeping track of it for you. There are also "exemptions" and "exclusions" which are categories of information they do not have to share with you – information that could impact national security, personal information that would violate privacy concerns, and so on. (1/2)

Sources of Data

FOIA requests can be used by the public to get access to data.

Freedom of Information Act (FOIA), 5 U.S.C. § 552

FOIA requests also can cost money and take a long time, which may make it infeasible for you to get the information you're looking for when you need it. You should generally tell them the timeline and your budget for the request when you make it. Agencies are often willing to work with you if you can narrow down what you're looking for in a way that's easy for them. And fees can be waived in cases where it's in the "public interest" – and not your personal profit. As always when asking for things from strangers, be kind, courteous, and respectful.

Sources of Data

An informal ask is sometimes the easiest method or the best place to start.



To that point – sometimes it’s easier to just ask for the data from an official directly rather than going through the whole FOIA request. A few years ago, a couple of legal hackers here in DC went to the D.C. Council’s General Counsel, a Mr. Dave Zvenyach, and asked for the entire DC code. After some back and forth, the code was released as public domain data and dccode.org was born, displaying the code using a modern interface. A little while later, over at OpenGov Foundation we asked city counselors and the mayor’s office to post bills online on our Madison project, which ended up becoming drafts.dc.gov and is still in use today. None of this was through official legal action or paying costly fees – just people talking to each other and getting excited about doing good work.

Sources of Data

- There are ongoing court cases over what is public domain and what isn't. (Building codes, etc.)
- The government may charge for access. These costs may be prohibitively high. (FOIA, PACER)
- Not everything is FOIA-able.

Now, things don't always go quite so smoothly. There are a lot of potential barriers between you and the data you want. First off, not all data that should be public domain necessarily is. For instance, though our legal codes are largely public domain data, there are currently pending court cases around law that is under copyright, such as building codes. You're going to be hearing a lot more about this in your next session of this course, so I won't go into this now.

Besides that, as I mentioned before, there might be costs charged by the agency for retrieving the data. One of the most well-known instances of this is PACER, the US Courts record system, which charges you per result returned for searches on its website. The biggest problem here is that you don't know how many results will be returned until after you search. There's currently an ongoing class action lawsuit around PACER, which contends that the fees are higher than they legally should be. (1/2)

Sources of Data

- There are ongoing court cases over what is public domain and what isn't. (Building codes, etc.)
- The government may charge for access. These costs may be prohibitively high. (FOIA, PACER)
- Not everything is FOIA-able.

I also have already mentioned that plenty of data is not FOIA-able. This can often be surprising, for instance we routinely find that records have been stored with personally identifiable information such as names, addresses, and even social security numbers – and this data may be hard or costly for agencies to clean.

As a side note, you might also find that the agencies have done a poor job of cleaning their data before providing it to you, putting you in a position where you have to remove personal data from your tools very quickly and with very little warning. I've seen this happen more than a few times. Most recently, I had a person contact our organization to ask us to remove a public document honoring them for an achievement, because they were facing harassment and didn't want to give their attackers more ammunition. Though we had no legal requirement to do so, the ethical thing to do was to simply remove the document.

Cleaning Data

- Most of the time, the first goal is to get the data into a database.
- Translate the data from the provided format to a usable format.
- Lightweight, verbose scripting languages are the easiest way to do this. (Python, Ruby, Javascript)
- Most languages already have libraries for common formats, such as CSV, Excel, XML, and JSON.

Now, this gives us a taste of what is needed to clean our data for presentation. At a more basic level, we generally have to do some work just to get our data into something useful – it's rare that we use the data in the same form that we received it in. Assuming we've received a Word document or spreadsheet or something, most of the time the first thing we want to do is get it into a database of some sort, so we can store it and manipulate it. (1/2)

Cleaning Data

- Most of the time, the first goal is to get the data into a database.
- Translate the data from the provided format to a usable format.
- Lightweight, verbose scripting languages are the easiest way to do this. (Python, Ruby, Javascript)
- Most languages already have libraries for common formats, such as CSV, Excel, XML, and JSON.

Even if we're not using a database, we'll want to use some sort of programming language to translate our data into something useful – another type of document or whatever. I recommend choosing a nice, general-purpose scripting language that's flexible such as Python, Ruby, or Javascript. These all have the advantage of having native libraries to work with many of the formats we'll be encountering, from CSVs and Excel files, to Word Documents, XML files, and JSON APIs. And by using a native library, we reduce the amount of code we have to write, and - more importantly – the number of things we might do wrong.

Cleaning Data: PDFs

PDFs are notoriously hard to extract data from

Often, paper documents scanned as images. Text content is lost in this process. OCR (optical character recognition) tools to extract text from the images. (e.g. Tesseract)

Content may be broken up in complicated ways preventing reassembling by tools.

Just a quick side note. There's a running joke in civic tech about how bad PDFs are, and this is well-warranted. Agencies scan paper documents as images, which are embedded in the PDF file. This is basically a photograph of the document, so the "text" is not digitally available. Which means, you can't copy and paste selections from it into other documents or extract meaning from it via automated means. Your computer doesn't "know" what the text says as it does with a standard document like a Word file. In these cases, you'll need to use an OCR (optical character recognition) tool to extract the text from the images. (e.g. Tesseract) These tools "read" the document and try to guess what the characters are. Even if the PDF has real text in it, it may be broken up in ways that prevent you from automatically getting that content back out. For instance, if you have a document with two columns, or a table of values, in the source of the document, those may be jumbled such that the data all runs together. This can make it almost impossible to get the content you want back out of the document.

A Simple Scraper

Python:

```
import requests
import json

response = requests.get('https://vacode.org/1-1.json')

law = json.loads(response.text)

print law['section_number'] + ': ' + law['catch_line']
```

Ok, so let's see what this actually looks like. Here's a very simple scraper. We're using Python as our language of choice to connect to the VA Code website and download a single law – section 1-1 – as a JSON file. We're using the Python requests library to do all the remote file getting, and the json library to turn the json into an object that Python understands. We then take that object and print out the section number and catch line of the law. You could just as easily turn this into another type of file here, put it into a database or web page, or really anything else.

A Less Simple Scraper

[illegible]

Here is a far less simple web scraper; this is one small piece of the code that gets legislation from the GPO FDSys and scrapes them for the GovTrack site.

A Less Simple Scraper

Python:

```
import json
import logging
import os
import re
import xmltodict

...
    fdsys_billstatus = utils.read(fn)
    return xmltodict.parse(fdsys_billstatus, force_list=('item', 'amendment',))

...

    bill_data = {
        'bill_id': bill_id,
        'bill_type': bill_dict.get('billType').lower(),
        'number': bill_dict.get('billNumber'),
        'congress': bill_dict.get('congress'),

        'url': billstatus_url_for(bill_id),
```

Drilling into the code a bit, it's really not that much more complicated than our previous example. Don't let the long names here scare you off – we're using a lot of the same libraries, we're still reading the file in, and we're doing some parsing. This time we're using an XML file instead of a JSON file, and instead of printing out the results, we're taking our resulting data and reformatting it into an object that's shaped slightly different for more processing later. Note that we're doing a little bit of cleaning in here as well - that call to the `lower()` function for the bill type is changing the type into all lowercase text. This helps to make it easier to compare to other data later, so we don't have to worry about the case not matching up, capital-B Bill versus all lowercase bill versus all capitals BILL.

Data Traversal

Ways of getting to the thing you're looking for in the data, once you know where it is.

- Object properties `laws['2-10']['title']`
- HTML selectors `soup.findAll('table.laws .title')`
- XML: XPath/XQuery (lxml) `soup.find('metadata/*/title')`

The most important thing to know here is how to get at the pieces of data you actually want to use. You may have a large spreadsheet with several hundred items, or a huge data file with dozens of properties, and you need a way to isolate just the pieces you care about. Here are a couple of the ways we do that. The most common way is to turn our data into a native object in the language of our choice – through something like the json translation example we saw in our scraper – and then traversing the properties of the object to get to the piece we care about. In this first example, we have an array of laws, and we're trying to get the title of section 2-10. In Python, this data structure in particular is called a dictionary, in other languages it might be called an associative array – but no matter what you call it, you'll have a way of drilling down through the items in it to get to the pieces you need. In most languages, the notation will be very similar to this.

Data Traversal

Ways of getting to the thing you're looking for in the data, once you know where it is.

- Object properties `laws['2-10']['title']`
- HTML selectors `soup.findAll('table.laws .title')`
- XML: XPath/XQuery (lxml) `soup.find('metadata/*/title')`

The next example is using a Python library called Beautiful Soup to parse an HTML document – which is just a normal web page like you'd find on any site – to get the title of all laws in a table of laws. This is looking for the HTML element that is a table and has a class of “laws”, and then for each of those finding the element with a class of “title”. Now, this method works great if you have a nicely marked up web page. If you don't you'll have to resort to uglier methods, like counting to the third table on the page and the fourth column in that table, or whatever. No matter how you go about it, just know that we have rather friendly ways of getting data out of web pages.

Now, HTML and XML are very closely related, so we can use a similar methodology to get data out of an XML file. XML has a large number of dedicated tools and methods for this, which include XPath and XQuery, which are ways of finding data in XML files. The lxml library provides these for us, and when used with Beautiful Soup in Python, we can again get our data similarly to how we do in an HTML file. In this last example, we're finding a title element that is nested under a metadata element, anywhere in our document.

Cleaning Data

- Look at a *lot* of data
- String/Number manipulation
- Regular Expressions (RegExp / regex / grep)
- Natural Language Processing (NLP)
- Look at your data some more
- Peer Review

Now, we've got our data, we've downloaded it somehow, and we can move through it programmatically. The last thing we need to do is clean it up. This is the hard part. First, we need to look at the data and see what's weird about it – if there are obvious inconsistencies, missing data points, or obviously wrong content. As I mentioned earlier, this is a good time to look around for protected information that you shouldn't be releasing as well. (1/4)

Cleaning Data

- Look at a *lot* of data
- String/Number manipulation
- Regular Expressions (RegExp / regex / grep)
- Natural Language Processing (NLP)
- Look at your data some more
- Peer Review

The most used tool in our toolbox for cleaning up data is String manipulation. You remember Strings from your data types reading? We can perform operations on strings to manipulate them in powerful ways. You saw the example a few slides back of changing a string into all-lowercase letters, this is a common one. You might also have to handle “special” characters – all those letters with umlauts and accents, symbols, anything that might cause problems in processing later. You might want to combine several strings into one, like combining first and last name into just a “name” field. Or you might need to split a string into several pieces, like in the State Decoded, we often receive data where the law section number and the law title are squeezed into one field, so we have to split out the number from the beginning of the line. You can largely work with numbers in the same way, and you’re probably familiar with doing this in spreadsheets already – adding several columns together, or averaging numbers, and so on. All of these can be handled using built-in features of the programming languages, and pretty much all languages have these features. (2/4)

Cleaning Data

- Look at a *lot* of data
- String/Number manipulation
- Regular Expressions (RegExp / regex / grep)
- Natural Language Processing (NLP)
- Look at your data some more
- Peer Review

Now in some cases, these law number-title pieces are too complicated to just split out by string manipulation. Maybe some laws have the word “Repealed” before the section number. Or maybe the section number has letters instead of just numbers in one chapter. Or maybe for whatever reason the publisher has used the section symbol before the number in one part, the word “section” in another, and the abbreviation “sec” in yet another. I’ve seen all of these, by the way. In these cases, we want to match the *pattern* of the text. We can describe rules for these patterns: if the number is preceded by the word repealed or a section symbol or the word section then ignore that bit and keep going. Regular Expressions are a pattern language we use to describe these sorts of rules, and find the bits of text we care about. You can think of it as a smarter version of string manipulation. You might all here these called “RegExp” or “regex”. Or “grep” which is actually a tool made for regular expression handling, not the language itself. (3/4)

Cleaning Data

- Look at a *lot* of data
- String/Number manipulation
- Regular Expressions (RegExp / regex / grep)
- Natural Language Processing (NLP)
- Look at your data some more
- Peer Review

Past that, we can do some analysis of the text itself to look for commonalities, which can reveal inconsistencies. For instance, we might want to know where in a legal code the legal definitions are, based on specific language like “A means B.” We can use a variety of Natural Language processing tools to help us do that. Now, this tends to be more of an analysis tool than a purely cleaning tool, but can be very useful for helping to identify trends in the data. In many cases the NLP may be the end goal itself, so that you can report on trends in text.

Once you’ve gone through all of this, look at your data more. You’re probably going to continue to find problems that you need to code around, over and over. It would often take me two weeks to clean up a legal code enough to be re-published in The State Decoded, and I still wouldn’t find everything. Just keep working through it until you’re satisfied.

Then find a friend who knows a little about the field and get them to kick the tires for you – a little unofficial peer review can go a long way towards better data quality.

This is probably the most important step in the process, so don’t skip it. (4/4)

Putting Data Online

- Static Prose, Tables, Charts: Medium, Wordpress, GitHub Pages
- Maps: Google Maps, MapBox, Leaflet (Javascript)
- Dynamic Websites & APIs: Python, Ruby, PHP, Javascript, etc.
- Interactive Tools: Javascript + API
- User Experience / User Interface (Jakob Nielsen, Edward Tufte)

So, we have *relatively* clean data. Now we want to put it online. We have to consider our audience first and what our goal here is. Do we just want to write about what we've found? Then we can just make a blog post somewhere like Medium or Wordpress or GitHub pages or anywhere else you like to blog. You can post tables and charts this way, or share spreadsheets of the data you've cleaned, whatever. But with this method, this data isn't going to change over time, you're presenting a snapshot of the data, as it is at this moment.

You could also embed some sort of map showing the data, if that helps to demonstrate what's happening. There are lots of tools for mapping, but here are a few of the more popular ones. There's a bit of a learning curve here, and you're going to need to write some Javascript to make these work, but you can still put them in a static blog post as we did above. (1/3)

Putting Data Online

- Static Prose, Tables, Charts: Medium, Wordpress, GitHub Pages
- Maps: Google Maps, MapBox, Leaflet (Javascript)
- Dynamic Websites & APIs: Python, Ruby, PHP, Javascript, etc.
- Interactive Tools: Javascript + API
- User Experience / User Interface (Jakob Nielsen, Edward Tufte)

Now, you may want to let people interact with the data in some way, or you might want your data to be updated over time as new data is published. For that, we're going to need to a dynamic site – that is, one that is backed by some sort of programming language, so we can change it as the data changes. There are many good programming language options to choose from, and here are some of the most popular ones. All of them have libraries that will allow you to build a dynamic website relatively quickly and easily, with a little programming knowledge. All of them can also allow you to build an API using libraries as well.

If you want to build interactive tools – say, charts and tables that users can manipulate by filtering data, or combining data sources – you're probably going to need some Javascript and an API. Any time you go to a website and you can filter data – say, only show me t-shirts that are blue, or votes on a bill by a particular political party, that's almost always using Javascript in your web browser and an API. (2/3)

Putting Data Online

- Static Prose, Tables, Charts: Medium, Wordpress, GitHub Pages
- Maps: Google Maps, MapBox, Leaflet (Javascript)
- Dynamic Websites & APIs: Python, Ruby, PHP, Javascript, etc.
- Interactive Tools: Javascript + API
- User Experience / User Interface (Jakob Nielsen, Edward Tufte)

The last thing you'll need is some knowledge of user experience and user interface. If you're going through all the trouble to get this data and to show it to people, you need to make sure that you're presenting it in a way that is clear and easily understood. For instance, one of the biggest mistakes I see developers making is using pie charts. Research has shown over and over that humans are really terrible at judging proportional sizes of slices of a circle, our brains just aren't wired for it. Similarly, the more labels and lines and text and rulers you add to a chart, the harder it is for anyone to read and understand it at a glance.

Jakob Nielsen and Edward Tufte are two of the most prominent writers on the topic of presenting information to humans in a way that's easy to use and understand – so I strongly suggest you spend some time getting familiar with them. And that's even if you're not going to be considering a full-time career as a programmer or web developer – that next slideshow presentation you give will benefit greatly from their tips. (3/3)

Digression: API Types

- SOAP (not used much these days) (XML, WSDL)
- XML-RPC (still used some)
- REST (most common today)
- GraphQL (new & trendy, but not used much yet)

I do want to take a quick step back here and dig in a bit on a concept from your reading, and that is “What is an API”? Because there are many types of APIs, and this gets glossed over too often. An API is most commonly a piece of software, running on a computer connected to the internet, that other computers can ask for data. Now, we can ask for data in a variety of different ways, but your API probably only knows how to respond to one flavor of ask. Here are the most common types – you’ll need to know what type you’re dealing with to be able to interact with an API.

SOAP is an older type of API, and if you mention it around us older developers we will frequently grumble and groan and scoff and make dismissive noises. SOAP stands for Simple Object Access Protocol, and it uses XML to both make requests and send responses. You also will frequently see the term WSDL in connection with SOAP, which stands for Web Service Definition Language – it’s a way for the API server to tell the person requesting more about what data it has available and how to get access to it, again this is all through XML. (1/3)

Digression: API Types

- SOAP (not used much these days) (XML, WSDL)
- XML-RPC (still used some)
- REST (most common today)
- GraphQL (new & trendy, but not used much yet)

XML RPC is similar to SOAP in a lot of ways, but it's much, much simpler. You're still using XML to make and receive requests.

REST is the most popular API method today, it's incredibly simple. Instead of sending complicated XML documents, you simply use a url to ask for data - the same way that you type web site urls into your browser to pull up google or gmail. This is the method we saw way back in the simple scraper example, where we requested a JSON file for a particular law from State Decoded. There are similar methods in REST as well for submitting new data to the server and updating existing data. This is all designed to be very web-browser-friendly, using the same methods that we had already been using to interact with web pages. As a result, developers *love* this. It does have a few major drawbacks. Often, a REST API is going to either give you way more or way less information than you need, which in both cases slows down your requests – because moving more data is slower, and having to ask repeatedly for more information is also very slow. (2/3)

Digression: API Types

- SOAP (not used much these days) (XML, WSDL)
- XML-RPC (still used some)
- REST (most common today)
- GraphQL (new & trendy, but not used much yet)

GraphQL is the newest API type and aims to solve these problems. It was created by Facebook in 2012, and like most of their projects it's designed to make working with data on the web (and mobile apps) easier. But as I said, it's really new and hasn't gained wide adoption yet. Personally, I haven't had an opportunity to use it in any professional projects yet, but I'm very excited about it – it's really, really well thought out. (3/3)

Sharing Your Work

- Write about how you solved the problem. (blog/documentation/README)
- Share your code. (GitHub)
- Share your data. (dat project, github, blog, etc.)
- **Use open licenses!** (GPL, MIT, Apache2, CC0)

Ok, end digression. So you've done all this fantastic work, and you've shared the results with the world. Now you should consider sharing the actual work as well. In the civic technology community, we have a very strong ethic of sharing and reusability.

The easiest way to share your work is to write down how you did it. Adding documentation on your methodology – or just adding comments to your programming code – is a fantastic way to let others build on your work.

An even better way is to share your tools, whatever you had to create to solve the problem. Most commonly, we share programming code using GitHub, which is a site that makes it easy to store your code and let others use it. And if your project is public and open source, it's free to host on GitHub. There are other ways to host your code, but GitHub is easily the most popular in the world right now. Every single project I work on is on GitHub right now. Now GitHub uses a tool called Git to move your code around, and Git is a little complicated. (1/2)

Sharing Your Work

- Write about how you solved the problem. (blog/documentation/README)
- Share your code. (GitHub)
- Share your data. (dat project, github, blog, etc.)
- **Use open licenses!** (GPL, MIT, Apache2, CC0)

Besides sharing your code, you should share your data. Telling people where to get the original sources you started with is great, but sharing your transformed and cleaned up data is also great. If you didn't already create an API or something, you can host your data itself in a variety of ways - just uploading a spreadsheet or two to your original post is great. You can also host data on GitHub, assuming it's not ridiculously large. There are also dedicated projects for hosting data, such as the dat project.

Finally, putting your code and data online is completely useless if you haven't licensed it so that people can use it. Open licenses allow other people to take your code and reuse it for a variety of purposes. For instance, that code we looked at for the GovTrack bill scraper also powered the Sunlight Foundation congress API and is used by the CATO Deepbills project as well. Everyone shares the code and contributes back to it to improve it over time. This helps to build a strong community and keep research moving forward. Here are a couple of the most popular license – GPL, MIT and Apache are great for software code, and CC0 is good for data and text. (2/2)

The ???

Protip: Think through these *before you start*, not *after you're done*.

- What are your goals for the project? How will you evaluate your outcomes? Who will it help? Is someone already working on this? Can you work with them? <http://bit.ly/2jmTqEK>
- How much time and energy are you willing to dedicate to the project? Will this be a one-time or ongoing effort?
- How will you sustain the project? Does it require money (hosting costs, paying for data)? Will you need to fundraise?
- How will you do outreach to spread knowledge and get participation? How can you engage with those outside of your immediate community?

Ok, so I know you're all excited to run out and start working on your big project to save the world now. Before you do that, you need to consider a lot of things. Unfortunately, most people don't think through this list of issues – and I personally have been as guilty as anyone of this. Now, I always use a list like this these days before I begin any new project.

Starting at the beginning, you should really think about the impact that you want to have from your project. What change do you want to see? Who will benefit from this work? Are there already other people trying to solve the problem, and can you work with them? I'm just going to say it straight here – someone probably has already had your idea, and with any luck, they've written about it somewhere. I've dropped a link here to a fantastic post by Josh Tauberer, which goes into great detail of why many projects to “fix democracy” simply don't work. (1/3)

The ???

Protip: Think through these *before you start*, not *after you're done*.

- What are your goals for the project? How will you evaluate your outcomes? Who will it help? Is someone already working on this? Can you work with them? <http://bit.ly/2jmTqEK>
- How much time and energy are you willing to dedicate to the project? Will this be a one-time or ongoing effort?
- How will you sustain the project? Does it require money (hosting costs, paying for data)? Will you need to fundraise?
- How will you do outreach to spread knowledge and get participation? How can you engage with those outside of your immediate community?

Next, you need to think realistically about how much time and energy you're willing to sink into this project. Are you prepared to keep a site up and running, updating the data many times a year indefinitely? Or do you just want to spend one weekend on this and never think about it again? Be honest with yourself.

Similarly, think about how you're going to sustain this project. Another spoiler – you're probably not going to get people to give you millions of dollars for an idea, that's largely a myth. Even the largest and longest-running non profits focused on government transparency are running out of money and making huge cuts to data and technology. If your project costs money to run – or if you're thinking about doing it full-time, you're going to have to hustle. (2/3)

The ???

Protip: Think through these *before you start*, not *after you're done*.

- What are your goals for the project? How will you evaluate your outcomes? Who will it help? Is someone already working on this? Can you work with them? <http://bit.ly/2jmTqEK>
- How much time and energy are you willing to dedicate to the project? Will this be a one-time or ongoing effort?
- How will you sustain the project? Does it require money (hosting costs, paying for data)? Will you need to fundraise?
- How will you do outreach to spread knowledge and get participation? How can you engage with those outside of your immediate community?

Last, how will you share your results with the world? How can you get the data out to people who need it? It's really hard to cut through all the noise on Twitter and Facebook and Reddit these days, so you need a solid outreach strategy. Finding experts in the field or journalists or community leaders that care about your topic will help immensely. You should probably talk to those people before you get started, again to see if there are other people already working on this. (3/3)

Bill Hunt
@krues8dr

There are many great civic technology volunteer organizations in DC, two of my favorites are Code for DC (<http://codefordc.org/>) and DC Legal Hackers (<http://dclegalhackers.org/>). You don't have to know how to program to help out! Just interest and passion are enough.