Methods for Freeing Public Election Data from PDFs

Kara Van Allen [1] , Tiffany Xiao [1] , Karen Santamaria [1] , Maggie X. Wang [1]

1 Statistical & Data Sciences Department, 1 Chapin Way, Northampton, MA, 01063

Abstract

This paper details our work in contributing to the open-source Open Elections project. The goal of the Open Elections project is to turn unstructured election data from government offices in the United States into a standard, free and usable form for the public. In order to make election data accessible to the general public, we created parsers, packages, helper programs, python packages, and data visualizations with extensive documentation. We successfully completed parsers for the election results of 31 different counties from California, Michigan, Missouri and Indiana. The parsers results and accuracy (at least one county's PDF format from each state was successfully parsed with 100% accuracy), along with their ability to be generalized further to other states are discussed in the paper. In addition, we built a Shiny application with an interactive data visualization. The purpose of our paper is to outline the procedure on how to successfully free election data from text-based PDFs and convert them into usable formats such as the standard Open Elections CSV format and interactive visualizations. Through our work with the election data, we discovered the extreme difficulty of meeting the goals of the Open Elections project due to the variations in the election data. Not only does our project contribute to Open Elections by providing effective parsers, a visualization and a starting point for future work, but it also reveals the severe lack of standardization of election data in the United States and a significant need for work similar to that of the Open Elections team.

Introduction

Since the country's inception, the United States has used elections to appoint its government officials. Today, there are over 500,000 elected officials in the United States [1]. Currently, there is an information gap in voting record data for the election of these officials at the state and local levels. Although United States citizens are given the right to access any election data through acts such as the Freedom of Information Act [2], election information is often not easily accessible or usable.

Firstly, election data is not always easy to find. Despite the advancement of technology and the popularity of the internet, not all government offices use the internet to publicize their election data. Additionally, there is no standard way for government offices to collect election data. Thus, election data can come in the form of PDFs, databases, images and other various formats. Therefore, elected offices of the United States lack a standardized format for publishing voting results.

The lack of a standard format for voting results poses a serious problem for those seeking to analyze election data. There is currently no free, extensive and easily accessible way of gathering election data from multiple election offices. Anyone who is interested in working with election data often struggle to gather usable election data.

This is particularly troublesome when people want to build visualizations, graphics or analyses based off of aggregated election results.

Open Elections is an open-source political-party-independent project started by Derek Willis, an interactive developer at the *New York Times*, to provide clean and usable data from state and local elections to those who need it regardless of that persons data literacy. From their website, when speaking about the goals of the Open Elections project, they state:

> "we want the people who work with election data to be able to get what they need, whether that's a CSV file for stories and data analysis or JSON usable for web applications and interactive graphics ... Our goal is to create the first free, comprehensive, standardized, linked set of election data for the United States, including federal and statewide offices." [3]

In order to achieve their goal, the Open Elections project has a team of volunteers that tackle every step of the process of getting the non-standardized election data from government offices to a standard form accessible on their GitHub repository [4]. This process entails many steps, starting with gathering the election data and putting them into their databases. As previously mentioned, the election data can have a variety of forms and sources thus the volunteers work on obtaining the election data which can mean anything from scraping from a website to calling a government office and asking for mailed documents. After gathering the election data, volunteers use a variety of techniques to convert the election data into the standard CSV format used by the Open Elections project. Depending on the skill sets of the volunteers involved in this process, this is done by methods such as manual data entry or python scripts.

Although the Open Elections project has a large team behind it, there is a copious amount of work that needs to be done in completing its goals. There are thousands of elected offices and years of election results that need to be transformed into their standard format. The goal of our project is to contribute to the Open Elections project by delineating methods that can be used to free the election data from PDFs effectively and then convert the data into usable forms such as the Open Elections standard CSV format and an interactive visualization.

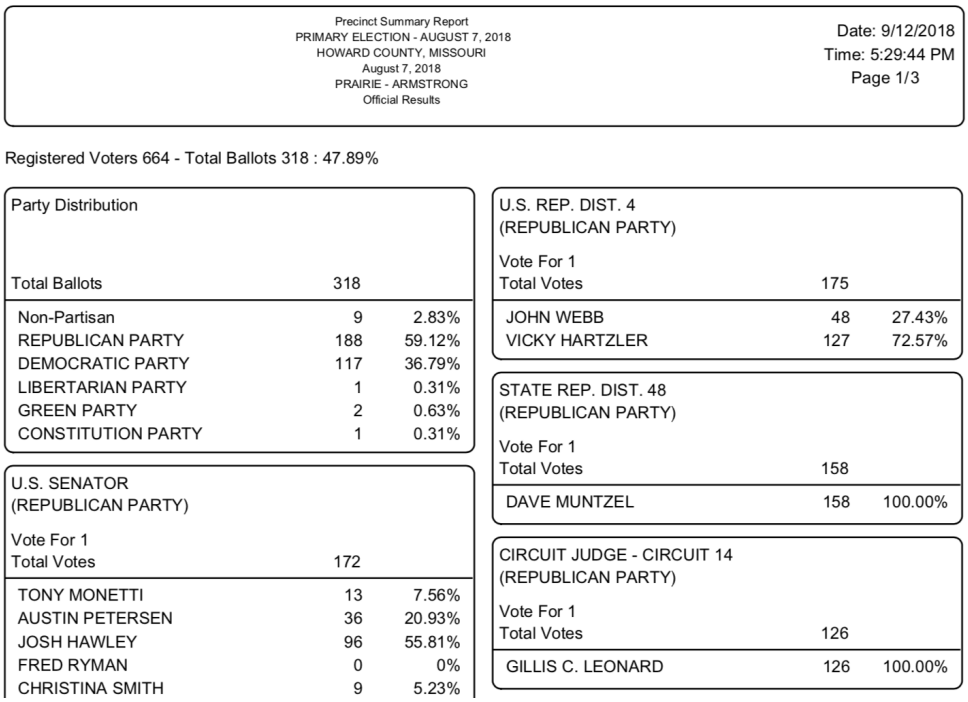
About the PDFs

While the information contained on the multiple PDF documents of voting data provided by state and local governments are valuable, the format they are stored in is inaccessible to the average data consumer. Each local government office publishes their own information in PDF files, many of them with distinct layouts. In order to make data accessible we created methods to extract PDF files into the common CSV format used by Open Elections.

We chose PDFs from California, Indiana, Michigan and Missouri as the PDFs we will convert into the standard CSV format. These states were selected because they contained counties with many precincts that publish their election data in PDFs that span hundreds of pages, which make manual data entry an extremely tedious task. In addition, we chose to work with only text-based PDFs (as opposed to image-based PDFs) and these states had many counties that only used text-based PDFs. Also, few to no volunteers had previously worked with the specific PDFs that we selected, thus there was a high demand for progress to be made in these untouched areas.

All PDFs are publicly available on the Open Elections' official GitHub repository [4]. Each PDF represents a particular county's voting results for a specific year. These PDFs all contain information on the precinct-level on the different candidates, their number of votes, the office they are running for, their party affiliation and the number

of votes they received. However, the format in which this information is displayed varies by state and for some states it varies by county. The PDFs we built parsers for are from California, Indiana, Michigan, and Missouri. Figures 1, Figure 2 and Figure 3 show excerpts of these PDFs. These figures demonstrate that each PDF can vary greatly from the others in how they display the election data. Figure 1 demonstrates a box based format, Figure 2 illustrates a table based format with vertical column names and Figure 3 shows a horizontal text based format.



Precinct Summary Report
PRIMARY ELECTION - AUGUST 7, 2018
HOWARD COUNTY, MISSOURI
August 7, 2018
PRAIRIE - ARMSTRONG
Official Results

Date: 9/12/2018
Time: 5:29:44 PM
Page 1/3

Registered Voters 664 - Total Ballots 318 : 47.89%

Party Distribution

Total Ballots	318	
Non-Partisan	9	2.83%
REPUBLICAN PARTY	188	59.12%
DEMOCRATIC PARTY	117	36.79%
LIBERTARIAN PARTY	1	0.31%
GREEN PARTY	2	0.63%
CONSTITUTION PARTY	1	0.31%

U.S. SENATOR
(REPUBLICAN PARTY)
Vote For 1

Total Votes	172	
TONY MONETTI	13	7.56%
AUSTIN PETERSEN	36	20.93%
JOSH HAWLEY	96	55.81%
FRED RYMAN	0	0%
CHRISTINA SMITH	9	5.23%

U.S. REP. DIST. 4
(REPUBLICAN PARTY)
Vote For 1

Total Votes	175	
JOHN WEBB	48	27.43%
VICKY HARTZLER	127	72.57%

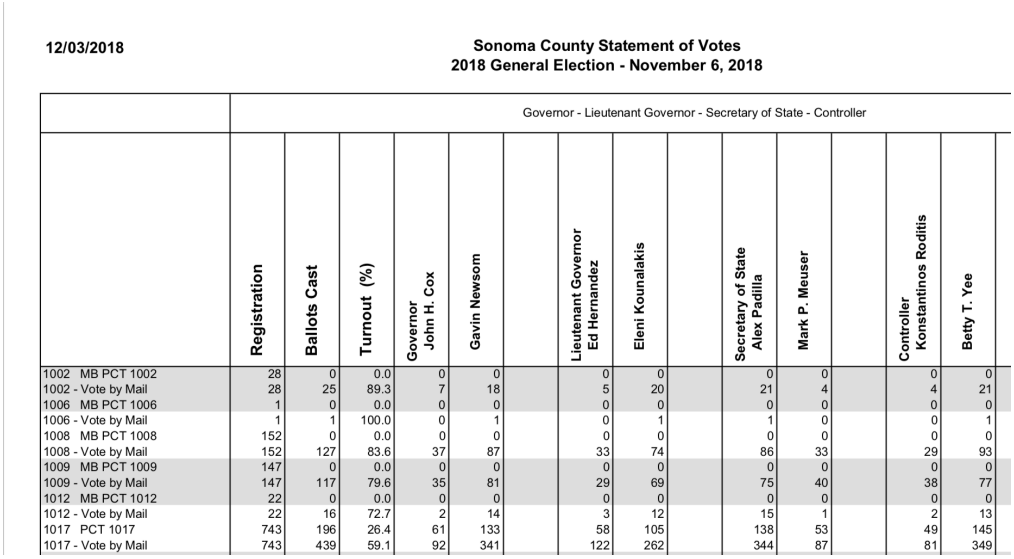
STATE REP. DIST. 48
(REPUBLICAN PARTY)
Vote For 1

Total Votes	158	
DAVE MUNTZEL	158	100.00%

CIRCUIT JUDGE - CIRCUIT 14
(REPUBLICAN PARTY)
Vote For 1

Total Votes	126	
GILLIS C. LEONARD	126	100.00%

Fig 1. Excerpt from Howard, MO 2018 General Election PDF.

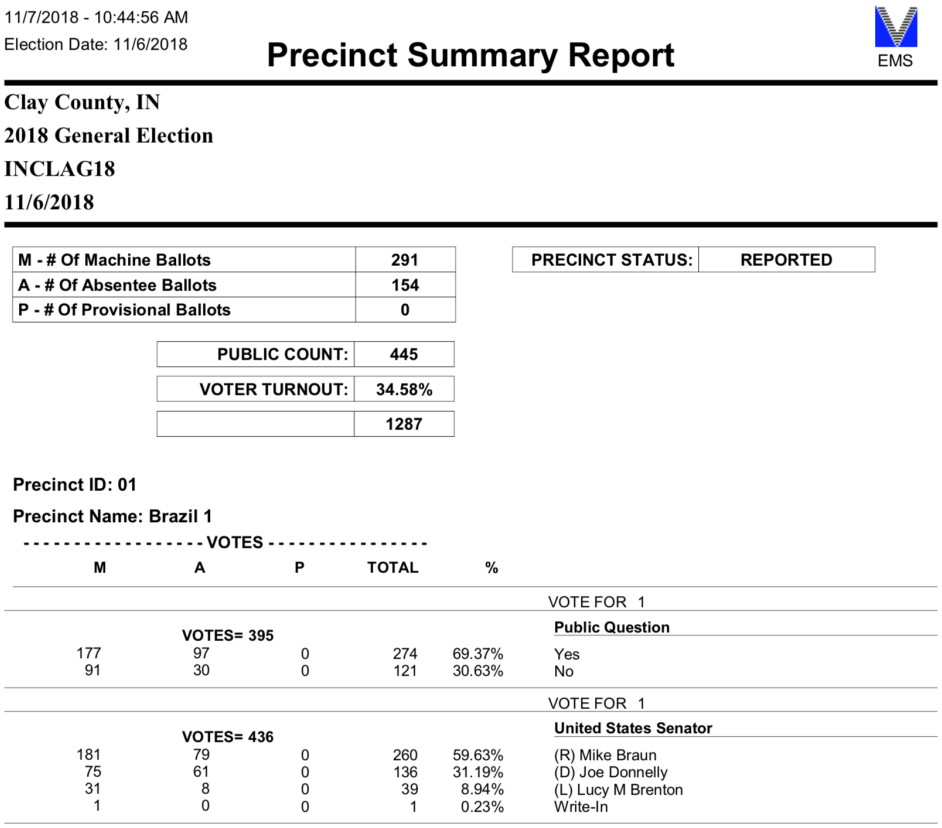


12/03/2018

Sonoma County Statement of Votes
2018 General Election - November 6, 2018

	Governor - Lieutenant Governor - Secretary of State - Controller														
	Registration	Ballots Cast	Turnout (%)	Governor John H. Cox	Gavin Newsom		Lieutenant Governor Ed Hernandez	Eleni Kounalakis		Secretary of State Alex Padilla	Mark P. Meuser		Controller Konstantinos Roditis	Betty T. Yee	
1002 MB PCT 1002	28	0	0.0	0	0		0	0		0	0		0	0	
1002 - Vote by Mail	28	25	89.3	7	18		5	20		21	4		4	21	
1006 MB PCT 1006	1	0	0.0	0	0		0	0		0	0		0	0	
1006 - Vote by Mail	1	1	100.0	0	1		0	1		1	0		0	1	
1008 MB PCT 1008	152	0	0.0	0	0		0	0		0	0		0	0	
1008 - Vote by Mail	152	127	83.6	37	87		33	74		86	33		29	93	
1009 MB PCT 1009	147	0	0.0	0	0		0	0		0	0		0	0	
1009 - Vote by Mail	147	117	79.6	35	81		29	69		75	40		38	77	
1012 MB PCT 1012	22	0	0.0	0	0		0	0		0	0		0	0	
1012 - Vote by Mail	22	16	72.7	2	14		3	12		15	1		2	13	
1017 PCT 1017	743	196	26.4	61	133		58	105		138	53		49	145	
1017 - Vote by Mail	743	439	59.1	92	341		122	262		344	87		81	349	

Fig 2. Excerpt from Sonoma, CA 2018 General Election PDF.



11/7/2018 - 10:44:56 AM
Election Date: 11/6/2018

Precinct Summary Report

EMS

Clay County, IN
2018 General Election
INCLAG18
11/6/2018

M - # Of Machine Ballots	291
A - # Of Absentee Ballots	154
P - # Of Provisional Ballots	0

PRECINCT STATUS:	REPORTED

PUBLIC COUNT:	445
VOTER TURNOUT:	34.58%
	1287

Precinct ID: 01
Precinct Name: Brazil 1

M	A	P	TOTAL	%	VOTES
					VOTE FOR 1
	VOTES= 395				**Public Question**
177	97	0	274	69.37%	Yes
91	30	0	121	30.63%	No
					VOTE FOR 1
	VOTES= 436				**United States Senator**
181	79	0	260	59.63%	(R) Mike Braun
75	61	0	136	31.19%	(D) Joe Donnelly
31	8	0	39	8.94%	(L) Lucy M Brenton
1	0	0	1	0.23%	Write-In

Fig 3. Excerpt from Clay, IN 2018 General Election PDF.

About the Visualization

In addition to parsers, we created an interactive visualization based on the data made available by the Open Elections project. The purpose of creating this visualization is to demonstrate how the data from the Open Elections project can be used to create meaningful visualizations that people can use to better understand the election data. Since our goal also involves creating work that other volunteers can use, we also created a website with our work and comprehensive descriptions about it. The website is hosted publicly on GitHub [5].

For our project, we focused on a visualization that depicts political parties' performance by state offices within a state. The visualization features the breakdown of votes by party for each elected state office within a county. It is a stacked bar plot, where each bar represents an elected state office and each stack represents the percentage won by a political party.

Methods

Turning Election Data from PDFs into a Standard CSV Format

Below describes our process in turning the text-based PDFs into the standard CSV format used by Open Elections. This process is can also be viewed in flowcharts provided in the Appendix Figures 5-6.

1. Locate Available Election Data

All the data (the PDFs) from the Open Elections project can be found on their GitHub page [4]. The PDFs we used were added to the GitHub repository by previous volunteers. Typically, volunteers of the Open Elections project pull the PDFs from elected offices' public websites.

2. Select Election Data to Transform

We focus on PDFs that are important to parse, consistent, and text-based. PDFs are important to programmatically parse when they are large since they would otherwise require many hours to process by hand. Also, PDFs need to be programmatically parsed when they are counties or states that have few volunteers assigned to them which means that the interest in manually parsing the PDFs is low. PDFs are consistent when they feature precincts that report election results in a way that can be exploited programmatically, such as having similarly formatted pages. Consistency is added when an entire state has a common way of reporting election data which further adds to the value of creating a programmatic parser for a set of PDFs.

Each parser we built were made to accommodate certain types of PDF formats. Thus, each parser has specific requirements to work on a PDF. In general, the main requirement for any PDF to be parsed using our parsers is that the PDF is text-based, since our parsers use various PDF text reading libraries. The parsers will not work on any image-based PDFs.

It is also important to select PDFs that have similar formats in order to effectively create a parser for them. For example, within the state of Indiana we were able to identity 24 counties with similar formats. Since many pages of the PDFs of multiple counties within the state of Indiana were very similar in format, we were able to select these counties and build a valuable parser to process the data from the counties' PDFs.

3. Transforming Election Data

The first step of transforming the election data to the standard CSV format used in the Open Elections project is to extract the text from the PDFs. We identified two main methods in order to extract the election data: parsing with a text-based PDF extraction package `pdftotext` [6] and parsing with a table-based PDF extraction package `tabula` [7]. For our project, we used the `pdftotext`[6] to parse counties from Indiana and Michigan and the `tabula` [7] to parse counties from California and Missouri.

Parsing with PdfToText

When extracting text from PDFs consisting of single-column PDFs with exclusively horizontal words (as opposed to vertical words [e.g. column names that are rotated 90 degrees]) it is advantageous to use `pdftotext`. This is because one can exploit information within the text of the PDF in order to create booleans to test for the presence of information. The most vital structure used when creating a parser that uses `pdftotext` [6] involves looping line by line and testing for conditions. The conditions tested for may very between PDF-type, however the `pdftotext` parser commonly features these checks:

- test for the "candidate line" which contains candidate name and number of votes
- test for lines that contain information outside of "candidate line" which may include, precinct, party, district, office

Parsing with tabula

On the other hand, `tabula` [7] works better on PDFs with more complicated table formatting options such as vertical or rotated columns and multiple horizontally-concatenated tables with different headers. In these cases, `tabula` [7] is superior because users can specify the area coordinates and page numbers they would like `tabula` to extract data from. Therefore, when page layouts become messier, one can execute multiple `tabula` extraction commands for designated areas to extract different types of information constituting a table. For example, one can opt to read in the table headers, table body, and metadata including time stamp and county information separately with `tabula` and then combine the various elements in the parser later on. The `tabula` package also dictates that it works better with tables with clear divisions between cells, either by white space or by cell borders, and that it needs to work with PDFs in which the same area on each or every few pages contains the same type of information.

After successfully extracting the text from the PDFs, one can begin work to manipulate the text into the standard CSV format used in the Open Elections project. Table 1 is an example of a table that follows the standard CSV format, with the proper column headers and five rows of election data from Blackford, Indiana.

Table 1. A sample CSV table for Blackford, IN that follows the standard Open Elections CSV format.

county	precinct	office	district	party	candidate	votes
Blackford	Harrison 1	U.S. Senate	NA	R	Mike Braun	206
Blackford	Harrison 1	U.S. Senate	NA	D	Joe Donnelly	142
Blackford	Harrison 1	U.S. Senate	NA	L	Lucy M. Brenton	40
Blackford	Harrison 1	U.S. Senate	NA	NA	Write in	0
Blackford	Harrison 1	State Secretary	NA	R	Connie Lawson	235

Since each PDF of election results leads to a CSV with the same formatting, we created two packages (`utils` and `table`) that are used regardless of the PDF's format. The `utils` package contains common functions that are useful when converting the text from the PDF into lists. The `table` package is used after manipulating the text extracted from the PDFs into lists, to turn the lists into a CSV with the standard Open Elections format.

Utils Package

The `utils` package contains functions that are useful across all the parsing work. This package includes the following functions:

- a function to convert a two column CSV into a dictionary,
- a function that changes an office name to a standard office name (e.g. "state senate" and "state senator" should be "State Senate" in the CSV),
- a function that checks if an inputted office string is an "accepted" office (an accepted office is an office that the client would like included in the CSV),
- a function that cleans an inputted string by removing spaces and lowercase the string,
- a function that rotates a PDF file 90 degrees (to read rotated column names),
- a function that cleans candidate names and adjusts capitalization.

Table Package

The `table` package contains `Table` and `Row` classes with functions to output data in the proper CSV format. The `Row` class is used to create an object that is representative of a row in the CSV and a `Table` represents a collection of rows that will output to a CSV. It has a function to add more rows to the table if it is not already in the list that was input, and a function to convert the list to a CSV.

After using these methods, one will have successfully parsed the data from a PDF into a CSV following the standard format used by Open Elections. The next step is to validate that the data has been parsed correctly.

4. Validate the Data

After processing the data into the Open Elections data format, one must ensure that the data has been transformed correctly. This step can be done in different ways, depending on the existing data (and its quality) on the PDFs parsed. If the data was already manually processed into CSV format one could use a validator script we built for this project that can calculate the percent similarity between the programmatically parsed CSV and the manually parsed CSV. This is done by using the python library `Pandas` [8] to compute the symmetric difference between the two CSVs. The percent similarity between CSVs is calculated by using the below formula where A denotes the number of lines in one CSV, B denotes the number of lines in another CSV, and C denotes the number of lines in the symmetric union of these two CSVs.

$$\frac{A + B - C}{A + B} * 100$$

Even between highly similar CSVs the validator may return a percent similarity of 0% because of differences in string formatting between the new and old CSV (e.g. capitalization differences). For this reason, for some states it is important to make sure that slight differences are changed to have a more precise measure of accuracy. For example, all instances of "Jane J. Doe" may need to be changed to "Jane J Doe" and "BLACKFORD 02" to "Blackford 2" when checking two CSVs against each other.

If there is no manually processed data available then manually checking between the PDF and the CSV may be necessary. A couple of common methodologies include making sure that all the precincts present with the PDF are also present in the CSV, checking that the total number of votes for candidates are accurate and ensuring that the office name is listed correctly. Also, if each page is similarly formatted, it may be possible to randomly select a few pages of the PDF for manual validation instead of the entire PDF document which may span hundreds of pages. Additionally, it could be possible to use other data such as state or county-wide data that has been extracted from other sources to cross-check the precinct level data. After validating the CSV, it can be used to conduct data analyses and create visualizations.

Creating a Visualization Using the Open Elections Data

A description of our methods in creating the visualization is below.

1. Locating and Selecting the Data

Our visualization is intentionally using the data from Open Elections. For our particular visualization, we used the 2016 New York General election results from the Open Elections GitHub repository [9].

2. Clean the Data

After getting the data from the Open Elections GitHub, the data needs to be cleaned before creating the visualization. We used the `Pandas` [8] library to build a python program that cleans the CSV. Our cleaning steps include the following:

- drop all NaN columns,
- drop duplicate rows,
- calculate a total percentage for each party in each elected office's results,
- pivot the table to have columns representing party results,
- add a column called `sum` that represents the sum of all party percentages aside from the Democratic and Republican parties (since our visualization focuses on these two majority parties, we only want to show a single percentage for the other parties)

3. Create the Visualization

Finally, the last step is to create the visualization. We created the visualization using R — we used `plotly` [10] to build the bar chart and `Shiny` [11] to host the `plotly` chart on an interactive application. `Shiny` allows one to build visualizations in interactive web applications by combining the power of R visualizations with the interactivity of html widgets. This will enable us to host the visualization on a website for others to view freely. Our `Shiny` application is hosted on shinyapps.io [12] and is featured on our website using an embedded iframe that is linked to the application.

Results

Parsers

The goal we set for our parsers was to make sure we were at least able to parse all the information from the PDF that would be needed for the CSV to be considered completed. This includes the following columns: `precinct`, `office`, `candidate`, `votes`, and `county`. All of the information for these columns are typically included in the PDFs. Optional columns include districts, candidate parties and extra voting types such as `mail`, `provisional`, `absentee` and more. We built four parsers that take different PDF formats and output a CSV with at least the minimum requirements for the standard CSV format (the code for the parsers can be found on our website [5]) . A general description of each parser, including the county formats that they work with, is detailed below.

Indiana Parser

As previously mentioned, the Indiana parser works with 24 different Indiana counties: `Adams`, `Bartholomew`, `Blackford`, `Boone`, `Clay`, `Clinton`, `Decatur`, `Dekalb`, `Delaware`, `Dubois`, `Greene`, `Hendricks`, `Huntington`, `Jasper`, `Jefferson`, `Kosciusko`, `Lawrence`, `Marshall`, `Morgan`, `Noble`, `Pulaski`, `Randolph`, `Shelby` and `Whitley`. Since the parser is a `pdftotext` parser, its main structure is reading the PDF line by line and relying on boolean statements that check for certain information. This specific parser checks for four things:

1. If the line contains an office name and a district number.
2. If the line contains a precinct name.
3. If the office is a state office.
4. If the line contains a candidate's name, party and number of votes.

It is important to note that the parser relies on the fact that the office name, district number, precinct name and state office for a row in the CSV always come before a candidate's name, party and number of votes. Since a line containing the candidate name, party and number of votes in the PDF represents a row in the CSV, once such a line has been detected a row is created in the CSV with the appropriate column values. After every line of a county's PDF has been read in, the parser will have finished building a list of values for the CSV which will then be converted into an actual CSV using the `table` package.

Michigan Parser

As another `pdftotext` parser, the Michigan parser is very similar to the Indiana parser. The Michigan completely parses the `Muskegon` county's format. Its structure and checks are the same as the Indiana parser.

California Parser

The California parser successfully parses the `Sonoma`, `Modoc` and `Plumas` county formats. As a `tabula` parser, the main structure of the parser is to convert the table format of the PDFs into the desired CSV format. The main parts of this process is to:

1. Split the table into three parts to help the `tabula` package read each part accurately. These three parts are:
    - The column names (which are candidate name, and each column value represents the number of votes the candidate won)
    - The row name which contains the name of the precinct and/or the type of vote
    - The table's actual values (without the row or column names)
2. Recreate the table structure within the program with the cleaned data.
3. Convert the table into the CSV format by reading in the table cell by cell, and creating a list of values for the CSV from those cells.

The California parser also includes functions that deal with rotating column names so that they can be properly read, merging tables on different pages together, checking if the table represents a state office and cleaning whitespace and percentage values in the table. The final step of the California parser uses the list of values for the CSV to create the CSV using the `table` package.

Missouri Parser

The Missouri parser completely parses the `Franklin`, `Christian` and `Cape Girardeau` county formats. It is also a `tabula` parser that relies on first creating a table format and then later converting into a CSV. The most significant difference between this parser and the California parser is how the tables on each page are split. The tables of the Missouri counties are split into the following three parts:

1. The office and party information
2. The table headers
3. The body of the table (including row names)

After splitting the table into these three parts, the Missouri parser follows the same steps as the California parser, with slightly different checks to account for small differences between the county formats (e.g. different locations for precinct names).

Again, it uses the `table` package to convert a list of values into a CSV after the entire table is read and cleaned.

Overall 31 files were parsed with varying degrees of accuracy. The table below shows the files that were parsed and the percentage of similarity to manually processed election results calculated using the validation script we built. Those that are marked NA have no manually processed election results. Again note that files may show less than 100% accuracy because of manual data entry errors due to differences in the way that the data is displayed (e.g., precinct names 'Randolph 01' vs. "Randolph 1") rather than an actual inaccuracy in the data parsed.

Table 2. Table detailing information about the 36 programmatically parsed PDF files with the number of lines parsed and percent similarity to the manually parsed CSV

date	state	election_type	county	number_of_lines	percent_valid
2018-11-06	CA	General	Sonoma	13024	100
2018-11-06	CA	General	Plumas	660	100
2018-11-06	CA	General	Modoc	484	100
2018-11-06	IN	General	Clay	506	100
2018-11-06	IN	General	Dubois	800	100
2018-11-06	IN	General	Morgan	972	100
2018-11-06	IN	General	Dekalb	731	100
2018-11-06	IN	General	Randolph	396	96
2018-11-06	IN	General	Hendricks	1857	95
2018-11-06	IN	General	Marshall	538	94
2018-11-06	IN	General	Noble	493	91
2018-11-06	IN	General	Whitley	639	91
2018-11-06	IN	General	Lawrence	627	89
2018-11-06	IN	General	Pulaski	268	88
2018-11-06	IN	General	Greene	606	87
2018-11-06	IN	General	Shelby	686	85
2018-11-06	IN	General	Jasper	456	83
2018-11-06	IN	General	Bartholomew	1286	83
2018-11-06	IN	General	Clinton	660	74
2018-11-06	IN	General	Rush	253	65
2018-11-06	IN	General	Delaware	1480	59
2018-11-06	IN	General	Huntington	454	38
2018-11-06	IN	General	Jefferson	515	37
2018-11-06	IN	General	Decatur	357	35
2018-11-06	IN	General	Putnam	570	7
2018-11-06	IN	General	Blackford	229	0
2018-11-06	IN	General	Boone	1036	0
2018-11-06	IN	General	Adams	316	0
2018-11-06	IN	General	Wabash	13	0
2018-11-06	IN	General	Kosciusko	1104	0
2018-11-06	IN	General	Hamilton	4112	0
2018-11-06	MI	General	Muskegon	1477	NA
2018-11-06	MO	General	Franklin	2401	NA
2018-11-06	MO	General	Christian	550	NA
2018-11-06	MO	General	Cape Girardeau	490	NA

Visualization

Figure 4 displays the visualization as it is on our website. The user can hover over the bars to see the percentage of votes the party has. Also, the user can select which county and elected offices to view in the bar chart.

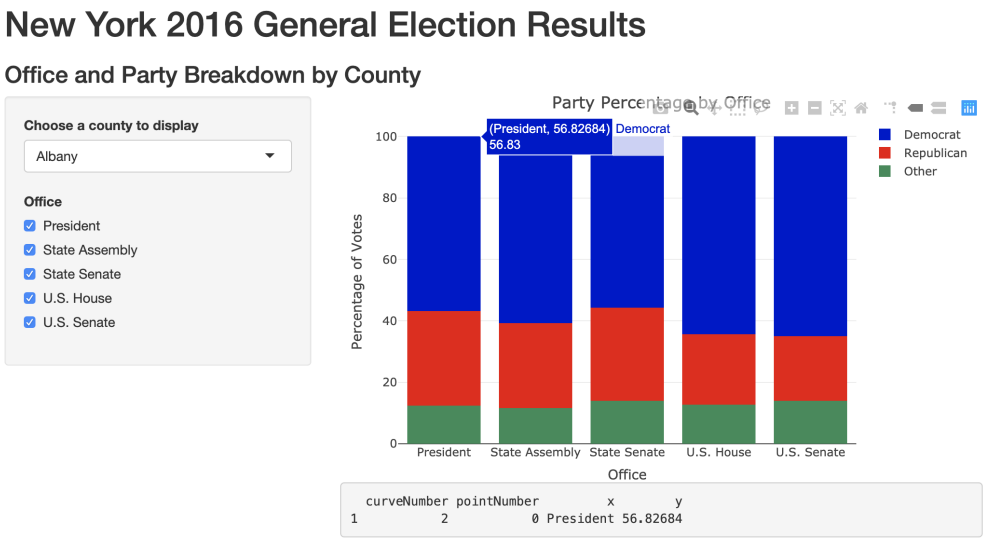


Fig 4. Screenshot of visualization as it is displayed on our website. In this screenshot, the cursor is hovering over the Albany's U.S. Senate office's Democratic party results. The popup details the party, the office and the percentage of votes won by the party.

Discussion

Limitations of Parsers

Currently, the four parsers that were created for this project can only parse the county formats they were built to parse. This is due to the fact that the parsers rely on specific attributes of the files. For example, a limitation of the `pdftotext` parsers is the reliance on indices to locate information (e.g. precinct names are the first index of the list that is created by the `pdftotext` text extraction for our particular counties). The `tabula` parsers also rely on specific attributes of the PDFs, such as the coordinates of the tables due to the nature of how the `tabula` package is meant to be used. However, these parsers have valuable cleaning and detection functions that can be used for other PDF formats. For example, the California parser features effective whitespace cleaning for the tables, since `tabula` sometimes improperly reads whitespace if the numbers are too close together in a table.

Limitations of the Visualization

The visualization is limited such that it is only able to show the CSV that has been loaded into the application. This means our visualization currently can only show the New York 2016 general results. In order to incorporate more states, one will need to follow the methods delineated for creating a visualization, and change the visualization such that it can allow switching between states. Furthermore, the visualization is only displaying one bar plot at a time — it may be more useful if one can show multiple bar plots at once, or if there were more interactive components.

Future Work

Running the parsers require beginner Python knowledge since the parsers will ask for the user to manually enter information such as the name of files or the coordinates of regions that were significant in a PDF. Furthermore, one who wishes to dive deeper into understanding the parsers will need to understand Python. With that in mind, our code is heavily documented along with READMEs and requirement files. In the future, it would be beneficial to create a graphical user interface (GUI) that removes the need of interacting with the Python program to enter things manually and thus make the parsers more accessible to new users.

In addition there are many other states and counties that have not been fully parsed. Therefore, modifying the previously created parsers to accommodate different PDF types can be a future endeavor. It may be possible to build a GUI that walks the user through the potential different formats of the PDFs and showing the user what code has been made to successfully parse the data from these formats. Also, image recognition technology can possibly be incorporated to identify the different PDF formats and recommend parsers for the format.

Based on the unique qualities of each elected office's statement of votes and the limited abilities of current PDF readers, we determined that is likely impossible to completely automate parsing all the statement of votes PDFs into the standard Open Elections CSV format. However, there is a large potential to build a guided GUI to expedite the parsing process by helping volunteers gain a better understanding of previous work done.

Conclusion

Despite the limitations of our parsers, they are still able to parse a large amount of PDFs effectively and serve as great starter code for future parsing work. Individuals who are interested in using these parsers for other counties may find it possible to use the same functions with a few changes. Unfortunately, there is no guarantee that the counties will continue to use similar PDF formats for their election votes. Perhaps the greatest finding of this project is that despite the fact that election results are supposed to be made available to the public, they are inaccessible in many ways (such as a lack of a standard format for hundreds of pages of election data).

Our project represents the workflow of someone who is interested in looking at election results at a large level — they first begin by gathering the election data made available by county governments and eventually create a visualization. The process required a significant amount of work by students who were trained in programming and data science concepts. Even with our skill sets, we still faced quite a few challenges in our work. Therefore, there is clearly a need for standardized election formats across the country. Furthermore, the lack of standardization and usability proves that the work that the volunteers at Open Elections do is integral to keeping election results truly accessible to the public. Contributions to the Open Elections project such as this project help make it possible for anyone with access to the internet find readable and comprehensive election data to learn more about the democratic processes that greatly impact their lives.

Acknowledgements

We would like to thank Derek Willis for providing his insight and expertise on the Open Elections data. We would also like to express our gratitude to Ben Baumer for his guidance and suggestions throughout this project. Finally, we wish to thank our peers

in the Fall 2019 Statistical & Data Sciences Capstone course who offered helpful opinions and advice on our project.

Appendix

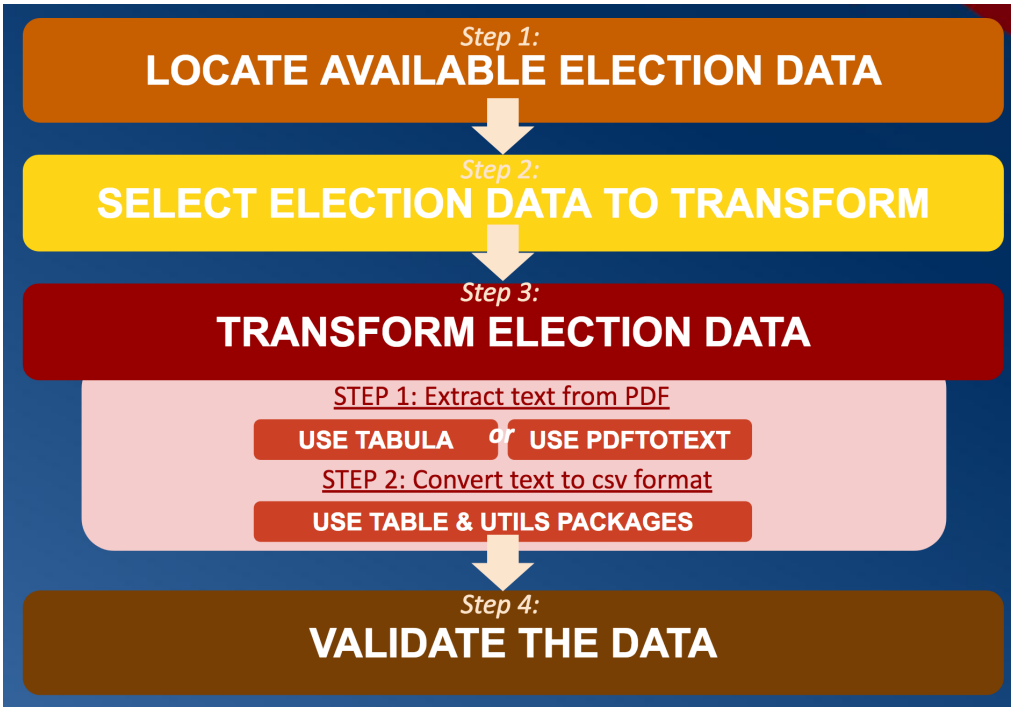


Fig 5. A broad view of the process to transform data from PDFs into a standard CSV format

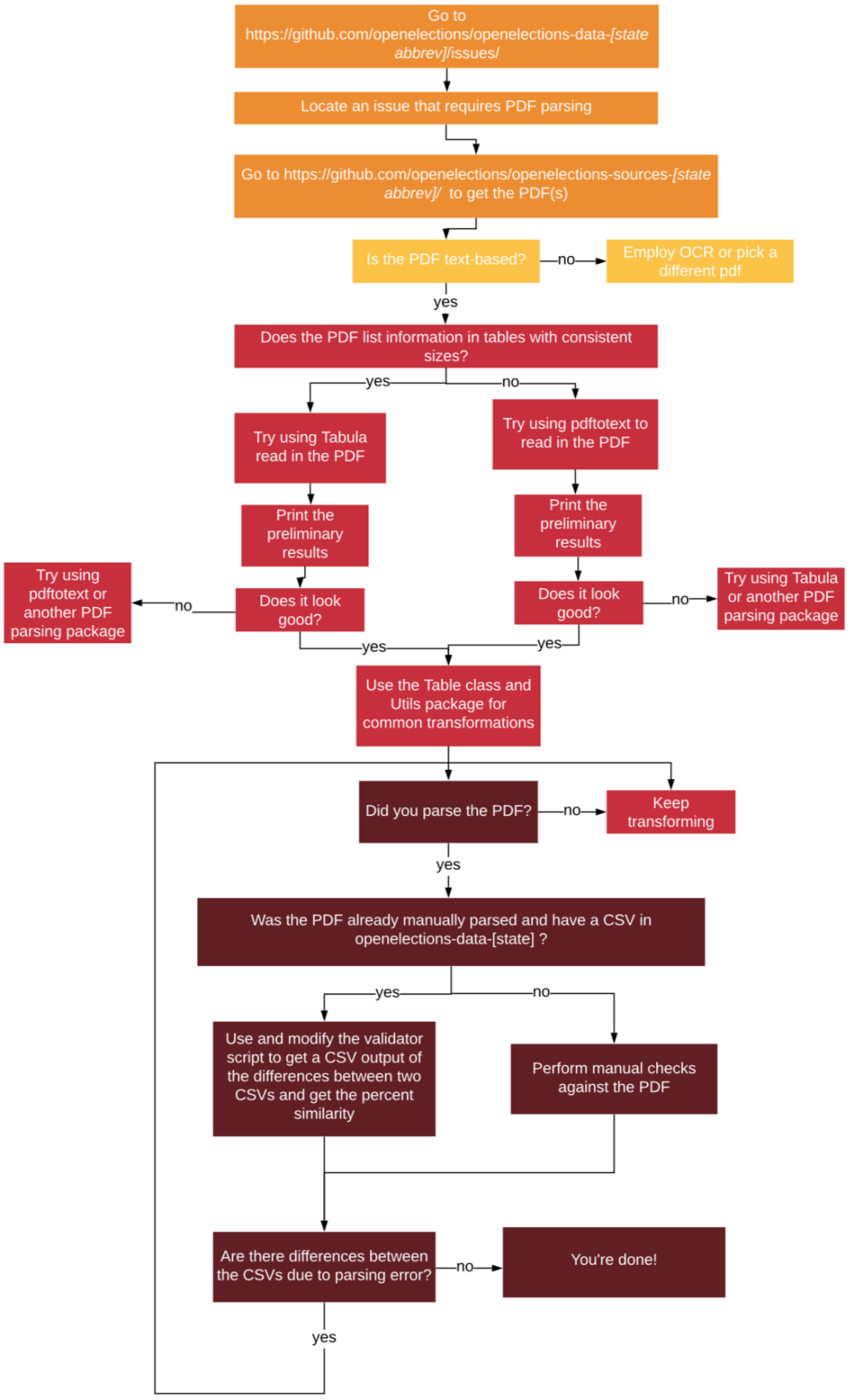


Fig 6. A detailed view of the process to transform data from PDFs into a standard CSV format. Following the steps outlined in the Figure 5, the orange boxes align with step 1, yellow boxes align with step 2, red boxes align with step 3, and brown boxes align with step 4.

References

1. Lawless JL. Becoming a candidate: Political ambition and the decision to run for office. Cambridge University Press; 2012.

2. Freedom of information act [Internet]. FEC.gov. Available: https://www.fec.gov/freedom-information-act/

3. Willis D. Welcome to openelections [Internet]. OpenElections. Available: http://openelections.net/about/

4. Willis D. The openelections project [Internet]. GitHub. Available: https://github.com/openelections

5. Xiao T. SDS capstone open elections github [Internet]. Available: https://sds-capstone.github.io/openelections/

6. Pdftotext [Internet]. xpdf. Glyph & Cog; Available: https://www.xpdfreader.com/pdftotext-man.html

7. Aristarán M, Tigas M, Merrill JB. Tabula-py [Internet]. PyPI. Available: https://pypi.org/project/tabula-py/

8. McKinney W. Data structures for statistical computing in python. In: Walt S van der, Millman J, editors. Proceedings of the 9th python in science conference. 2010. pp. 51–56.

9. Openelections. Openelections/openelections-data-ny [Internet]. GitHub. Available: https://github.com/openelections/openelections-data-ny/blob/master/2016/20161108__ny__general.csv

10. Sievert C. Plotly for r [Internet]. 2018. Available: https://plotly-r.com

11. Chang W, Cheng J, Allaire J, Xie Y, McPherson J. Shiny: Web application framework for r [Internet]. 2019. Available: https://CRAN.R-project.org/package=shiny

12. Shinyapps.io [Internet]. shinyapps.io. Available: https://www.shinyapps.io/