Name of group members and CNetIDs

Matt Jackson	mbjackson	12351444
Sarik Goyal	sarik	12370047

Project Abstract

Recent advances in computational power have allowed for automated means of creating fairer Congressional maps and detecting partisan gerrymandering. Our project approaches this task in two ways: (1) a method for generating quasi-random Congressional district maps, and (2) a statistical process for estimating the expected share of districts each party is likely to win, given its statewide vote share in a recent election. By generating a random map and comparing its expected district breakdown by party to our statistical metric (and to the actual current delegation), we point towards a method that could* help determine ideal configurations for balanced, competitive Congressional maps. Our project also plots and exports a simple image of the new map for visual inspection, with districts colored from dark red for solidly Republican to to dark blue for solidly Democratic.

*...if iterated at scale, used on more granular population data, expanded to consider trends across election cycles, modified to consider Voting Rights Act and other legal concerns, etc. etc....

Overall structure of the software

Our project draws on Redistricting Data Hub, which houses (among other things) geospatial data, population estimates, and election results for state voting districts (VTDs) in every state. Using a modified version of Python code provided by RDH, we pull data from their API, then use custom functions to join population, precinct shape, 2020 presidential election vote totals, and a unique identifier (`"GEOID20"`) data by state, in shapefile format. (Due to time constraints and hiccups with the API, we stuck to six states that were relatively close in 2020: Arizona, Georgia, Nevada, North Carolina, Ohio, and Texas.) We also wrote a function to generate an adjacency list for each state's precincts, which we use later in district drawing. (*This pre-processing is done prior to running the project*.)

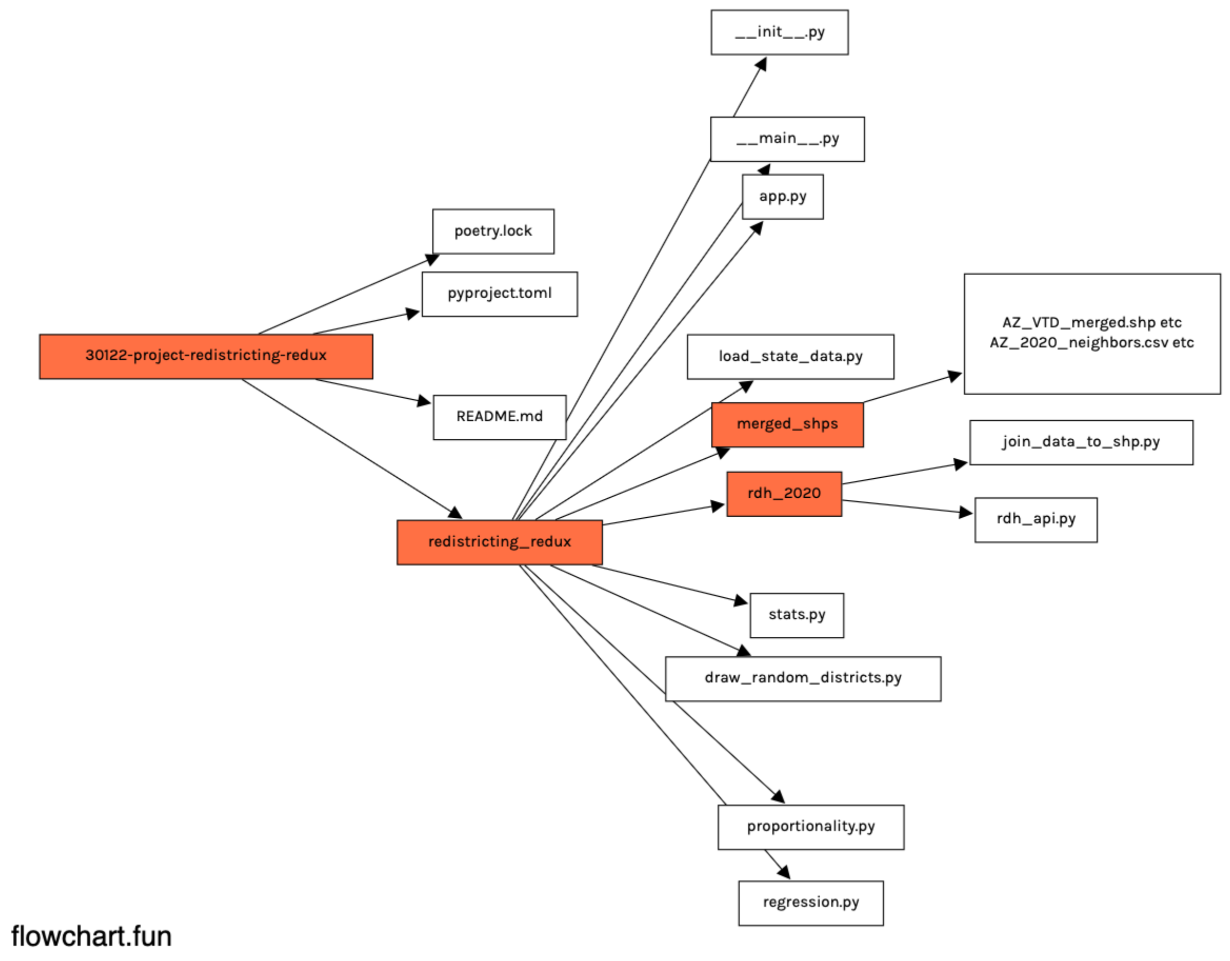
At runtime, we ask users to pick a state, then use `geopandas` to import that state's data as a GeoDataFrame. We settled on a draw method that picks *n* random precincts to start drawing *n* districts ("throwing darts at" the map); it then expands each district out into unclaimed space as far as possible, and attempts to swap precincts from overpopulated districts to underpopulated ones to achieve better population balance. (Our population balance function is very slow, and can get "stuck" above the desired threshold in some configurations, so users have the option to skip it or run it for only a few cycles.)

Once the user is satisfied with the population balance, we turn towards the modeling part of the project. Here, we explore how the statewide voteshare, the variance of these voteshares, and the degree to which voters are geographically clustered impact the partisan balance of a state's districts. We accomplish this by generating sample "states," modeled as square grids of districts, and seeing how varying those parameters affects the partisan balance of those grid districts. We generate training data for our model by creating several grids, then run a `sklearn`

`LinearRegression` on the results. (Since this process is also slow, we let users choose the number of trials, with a warning that a low number of trials may produce inaccurate results.)

Once the regression model outputs an expected seat share by party for the state, the program compares it to the partisan breakdown and expected seat share of the map it just created, as well as the current partisan breakdown of the state's actual Congressional delegation. Large discrepancies between these numbers may suggest that the new map has a partisan bias to it... or that the map in actual reality has been boosting one party beyond what we should expect.

Diagram of package structure (orange=folder, white=file)
(Due to time constraints with a 2-person team, `draw_random_districts/stats` and `proportionality`/ `regression` were not fully encapsulated as standalone modules within redistricting_redux.)



Description of code responsibilities for each member

Sarik: poetry shell maintenance, API calls to Redistricting Data Hub (modifications to `rdh_api.py, join_data_to_shp.py`), state dataframe joins/storage, all design and implementation of evaluative metrics (`proportionality.py`, `regression.py`), much of initial proposal and mid-project check-in

Matt: GeoDataFrame processing (`load_state_data.py`, `stats.py`), all design and implementation of district drawing and plotting (`stats.py`, `draw_random_maps.py`), organization of package structure and command line program flow (`app.py`), much of this paper (incl. flowcharts)

We also asked Ethan Arsht to help with creating a more accurate district-balancing function that preserves contiguity of districts. As of Project Fair time, we decided not to incorporate it. It is in the file `ethan_balance.py`. Thanks also to Cole von Glahn and Federico Dominguez Molina for their insights and advice on geopandas.

Short guide on how to interact with the application and what it produces

When run from the command line, the software uses print statements and input prompts to guide the user through picking a state, creating a map, balancing its population to a desired point, comparing it against a statistical model of the state based on 2020 presidential election vote shares, and creating a plot of districts shaded by 2020 presidential election partisan margin.

The output of the model, and of population and partisan margin data for the districts on the newly-drawn map, get printed to standard output.
The plot is a .png file that gets exported to a `maps` folder, from which it can be opened to view.

What the project tried to accomplish and what it actually accomplished

Our ambitions were vast: we hoped to create a new, multi-component fairness score, apply it to current real-world district maps, and use a custom algorithm to make new maps that maximized our fairness score. As we pared down to the more achievable minimum viable product described above, we lost a lot of validity and precision; our result is more of a demonstration / proof of concept than a usable tool. (The maps it outputs are definitely *extremely* illegal.)

Despite scaling back our project, we "accomplished" a lot in terms of learning and building our skills. For both parts of the project, we drafted, de-bugged, and improved several original functions and algorithms. We wanted to improve our skills with shapefile data and (geo)pandas selectors, and we did. (If we had to start over, we might incorporate `networkx`, treating precincts as nodes and viewing district drawing as a graph partition and/or graph coloring problem. At least one existing library, MGGG's `GerryChain`, does this.) The process of putting together our rather separate pieces of the project taught us a lot about how to effectively combine moving parts while keeping components distinct enough to allow for debugging.

We're especially proud that much of our code is easily extensible. With some adjustments to the columns of data we select, we could also assess the racial demographics of our random maps. When called in an interpreter, our map-drawing function lets us pick our own number of districts to, e.g. generate potential state legislative maps rather than Congressional ones. Though it didn't make our final project, we took exploratory steps with the more advanced visualization library like Folium, which could present maps with interactive or hover-over elements in a Web browser. Our functions for exporting the list of precinct neighbors from a state GeoDataFrame and/or affixing an adjacency list from a file could be tweaked to allow for export and import of .csv files of precinct-to-district assignments (which would allow user to "save" or "load" an old map without redrawing from scratch).

As for the modeling/metrics component, we hoped to create a metric that would reliably output an expected partisan balance given geographic and demographic properties of a state. Our result isn't as statistically rigorous as we might want. For example, while the model factors in geographic clustering, it doesn't account for *partisan* asymmetries in how voters are distributed in a state (i.e. that cities tend to be extremely blue, while rural areas tend to be more moderately red), just their geographic clustering.

Despite its shortcomings, our model did demonstrate one key insight: namely, that a variety of factors (such as said clustering) might make the expected partisan balance of a state's districts diverge from the proportion of statewide votes. Under the principle of "One person, one vote", it is tempting to believe that it is natural for a state that votes 60% Democratic to have 60% Democratic seats. While it is certainly reasonable to view proportionality as fair, our project shows that this relationship is not natural. In other words, our results imply that a mapmaker whose goal is achieving proportionality statewide would likely need to intentionally impose districts that take partisan election results into account.