ECE 356

US 2020 Elections Project Report

By: Andrew Hua, Jerry Yang, Gerald Zhang

Introduction:

This report intends to discuss the thought process, design choices, motivations, and more behind the creation of a Command Line Interface (CLI) application that services users to find information about the US 2020 Election. Information was acquired from Kaggle CSVs, which contained information about tweets, polling information, demographics, and post-election results. The report will first discuss the CLI application, with ideal requirements, proposed implementations, actual implementations, and any reasonings and afterthoughts. It will then follow up with a discussion on the ER design, with motivations and design choices, technical discussion of how the tables might end up looking, and anything else that may be interesting. Lastly, there will be a summary including successes, drawbacks, afterthoughts and closing thoughts.

CLI Application Design:

The ideal client requirements are to have fully fleshed out lookup on all attributes, relationships, and data for the US 2020 election in terms of results on a state and county level, polling on a state and county level, demographics for voting on a state and county level, differences in voting between the current election year and the previous (2016), and complete processing of tweet metadata with reference to how it may have impacted an election outcome in a state. Given this information, an ideal application should also be able to allow modification (specifically 'annotations') of existing data. Additionally, it would be useful for the client to also allow users to understand why votes may have swung one way or another in comparison to previous years, and if there are any demographics to explain it, leveraging both the data and specifically data-mining to coerce an explanation out from the information. The ideal client should also be able to receive any number of inputs as specific and dynamic as they'd like, and the client should be able to produce something of either mild to extreme coherency related to what the user is wishing to learn about, based on how specific and complex the inputted action was. It also has the freedom to find out specific placements or rankings of a certain property (i.e. county with most tweets, tweet with 3rd most likes, state with 2nd highest Hispanic population being polled) of applicable queries. The ideal client is also able to leverage proper location information from the tweet metadata to determine the county to which a person is located from.using a combination of either user-inputted tweet metadata of their location, or utilize the latitude and longitude to pinpoint which county the tweet was made from. It should also sanitize any string inputs to avoid nasty SQL injections of malicious intent.

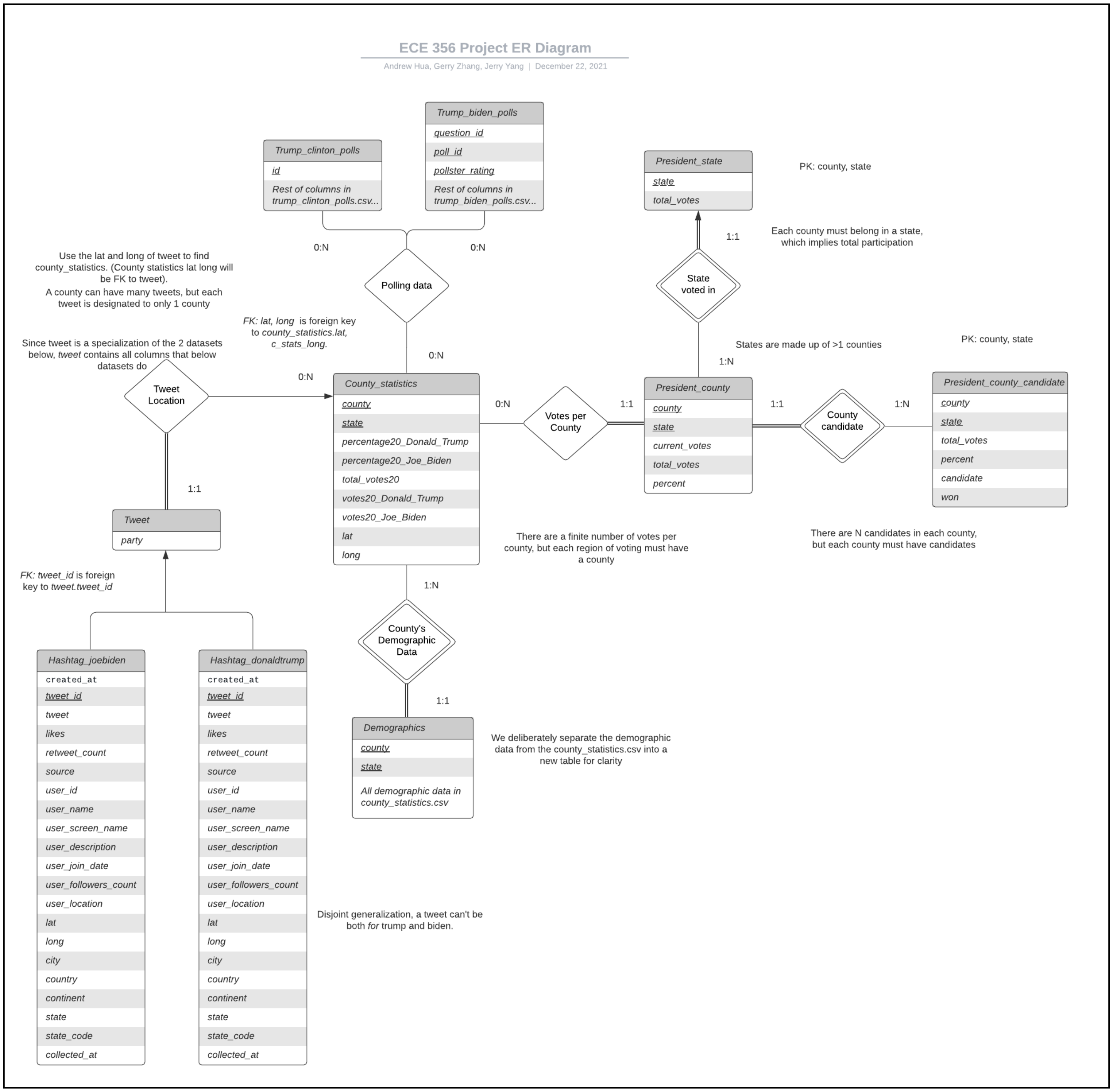
The proposed client will be a CLI application that will be able to take in specific commands to search up information about any of the specific areas that they would like to know about. This may either be county information, candidate information, historic information, or polling information. With a choice amongst any of these, the client should be able to accept more granular commands to allow a more definitive and specific understanding of what they would like to learn about. It would also be able to leverage data-mining to uncover patterns and reasons for why a statistic or specific query shows a certain result, allowing users to obtain at least a modicum of understanding what could've happened. The client itself is limited to only one 'query' (essentially one thing) that they would like to know about, to maximize the usage of arguments, flags, and keeping the overhead of having those at the parsing level. When it comes to ranking, the proposed is able to produce both the highest and lowest of a quantifiable query (i.e. tweet with most likes, county with smallest vote differential). It will also try to leverage the Kaggle CSV's of latitude-longitude information for counties to locate tweets in counties.This would be done using the various latitude-longitude information from the tweets CSV, cross-reference the values with a zip code amongst the OpenAddress Kaggle CSVs, and then use the corresponding row's zip code to find the county which it exists in the following Kaggle with zip codes to counties of all states in the US..

The actual client implemented was a Python CLI application, taking in a specific set of arguments, with various modifying flags to help make mundane queries more interesting. The compromise that was agreed upon for a lightweight client that users that are not of high technological competency can hope to use is a simple action-based query CLI that leverages flags to modify and alter whatever they seek to learn about. Actions were divided into separate domains of information, and came with flags depending on whether the action would (or could) require extra information to lessen the scope. There was also a successful integration of basic annotation (and removal of such) that can be attached to individual records of counties. Other successful parts of the client include itself as its independence from needing its existence (or even usage) on the *eceubuntu* servers and marmoset for a MySQL database is useful. Instead, the CLI application is able to leverage Docker and a MySQL instance along with various scripts to have it all locally setup and run. One mishap that was unfortunate was the omission of the county-finding queries that would be coupled with the latitude-longitude information from tweets and tables with corresponding latitude-longitude with zip codes, and zip codes to counties. Due to the sheer size of the table for complete efficacy of capturing possible zip codes (around 10GB), it was left out of the implementation. Other differences in proposed to actual implementation is the scope and granularity of queries (or specifically, just information about a certain topic they were looking for). The tradeoff specifically in this was that we covered and supported what we thought were more important queries and actions and left the completeness out due to short stock of time.In particular, the tables (and CSVs by extension) for polling information and tweets are unfortunately not utilized to a significant capacity. That is not to say however that they are not referenced, rather, they only were implemented with surface level querying complexity for information that was more relevant. It should also be known that some of the motivation behind using the available actions and queries were due to availability and stability of values in the CSVs. There are a few columns in some of the datasets which were difficult to leverage, such as lots of different tweet metadata being empty or null (often location parameters), to polling data being both inconsistent in info collected across the two elections and static information like seat name and ranked choice reallocation that did not provide useful information. As well, fields for similar information, for example 'state', would not be uniform in definition across the datasets. Lastly, no data-mining and more in-depth investigation of tweets and polling data was implemented due to time constraints.

The approach to the CLI is to have Python as the underlying infrastructure to read in commands, parse them, and appropriately assign queries and customize them to the specificity of the flags that are passed in. This limits the degree of freedom that the user might want, however, it also reduces errors and should simplify what the user may want out of the application. On the side of the database, a Docker container that is spun up will run an instance of MySQL, and our Python application will communicate with this MySQL database. The means of communication between the application and the MySQL instance was the Python library *mysql.connector*.A couple of scripts have been made in place to allow the downloading of the CSV's from Kaggle, which would then unzip them, and have them copied over to the Docker container. After which, an SQL file will contain table creation code to allow the CSV data insertion into the tables, and create the appropriate indices, primary keys, and foreign keys necessary to match our ER as best as possible.

On the Python side, it is able to parse the arguments passed into the CLI and based on what was passed, is able to run the selective query it wishes to run using an if-else structure that runs on the the type of arguments that were passed in. The query is then updated or is more complex based on whether more granular and complementary flags have been passed in with appropriate values.

ER Design:


ECE 356 Project ER Diagram
Andrew Hua, Gerry Zhang, Jerry Yang | December 22, 2021
Trump_clinton_polls
id
Rest of columns in trump_clinton_polls.csv...
Trump_biden_polls
question_id
poll_id
pollster_rating
Rest of columns in trump_biden_polls.csv...
0:N
0:N
Polling data
0:N
President_state
state
total_votes
PK: county, state
1:1
Each county must belong in a state, which implies total participation
State voted in
States are made up of >1 counties
1:N
PK: county, state
Use the lat and long of tweet to find county_statistics. (County statistics lat long will be FK to tweet).
A county can have many tweets, but each tweet is designated to only 1 county
FK: lat, long is foreign key to county_statistics.lat, c_stats_long.
Since tweet is a specialization of the 2 datasets below, tweet contains all columns that below datasets do
Tweet Location
0:N
County_statistics
county
state
percentage20_Donald_Trump
percentage20_Joe_Biden
total_votes20
votes20_Donald_Trump
votes20_Joe_Biden
lat
long
0:N
Votes per County
1:1
President_county
county
state
current_votes
total_votes
percent
1:1
County candidate
1:N
President_county_candidate
county
state
total_votes
percent
candidate
won
1:1
Tweet
party
There are a finite number of votes per county, but each region of voting must have a county
There are N candidates in each county, but each county must have candidates
FK: tweet_id is foreign key to tweet.tweet_id
1:N
County's Demographic Data
1:1
Demographics
county
state
All demographic data in county_statistics.csv
We deliberately separate the demographic data from the county_statistics.csv into a new table for clarity
Hashtag_joebiden
created_at
tweet_id
tweet
likes
retweet_count
source
user_id
user_name
user_screen_name
user_description
user_join_date
user_followers_count
user_location
lat
long
city
country
continent
state
state_code
collected_at
Hashtag_donaldtrump
created_at
tweet_id
tweet
likes
retweet_count
source
user_id
user_name
user_screen_name
user_description
user_join_date
user_followers_count
user_location
lat
long
city
country
continent
state
state_code
collected_at
Disjoint generalization, a tweet can't be both for trump and biden.

Our Entity-Relationship diagram was designed based on the framework of how the US election is set up. We chose to design it this way because if we want to extract useful, relevant information about the election, we need to know how the system works.

First, a little background information on how the US election works (since we're Canadians after all). The president of the United States is both the head of state and the head of government, unlike in Canada where the prime minister is the head of government and the Queen is head of state. This means the president has the power as the head of the executive and legislative branches of government so it's a very important role to be filled. In the US, there are two major political parties, the Democrats and the Republicans.

The electoral college is a group of representatives sent from each state that elect the next president of the United States. Electors are anonymously chosen representatives that cast the vote of their state based on which party their state voted for. There are 538 electors in the electoral college, 438 to represent the members of Congress and 100 to represent the Senators. This isn't to be confused with the fact that the electors are not the members of Congress or the Senators themselves, rather the 538 electors represent the number of seats in the US's legislative parliament. Each state will have electors equal to the number of senators and members of congress they have, this number is proportional to the population of the state - the more people in a state, the more seats in Congress that state has. The winner of the electoral college must get 270 electoral votes to win the presidency.

Now how does each elector choose who to vote for? The states are won using a first-past-the-post system, which is a winner-takes-all. The winner is determined by majority of votes in the given boundaries of the vote. For example, if the vote in Texas is 51% Democrat and 49% Republican, then all the votes of that state go to the Democrats. Now this paints a good picture on how to win the presidency - win in the right states that add up to 270 electoral votes, and secure the victory.

This leads us to our Entity-Relationship diagram, where we model the design after the American election system, we created entity sets such as *state, county, tweets,* and *polls*. We then broke these sets down to improve the clarity of our design.

From our datasets we were given Tweet data supporting each candidate (the tables *hashtag_joebiden, and hashtag_donaldtrump*). We made the design choice of creating a disjoint generalization of these two tables to make a generalized *Tweet* table that includes the same attributes as well as a party attribute to indicate which of the two candidates the tweet supports. The generalization simplifies the tweet information as both tables have the same attributes, and since a tweet can't be supportive of both Biden and Trump in the election, we chose a disjoint generalization. The new *Tweet* table includes a foreign key to the previous tables.

The *Tweet* table also includes metadata, such as the longitude and latitude of where the tweet was made. This information is valuable to us as we are able to examine trends and answer questions such as "which states have the largest ratio of supportive Trump tweets?"

Resultantly, we created a relationship set between our *Tweet* entity and *county_statistics*, where *county_statistics* includes voting and demographic information of each county. Through this relationship set we can leverage the longitude and latitude of the tweet and match it with the corresponding county and use *county_statistics* to examine voter and demographic trends. Data such as this is invaluable to understanding patterns in how people vote.

The main table we used for our database was the *county_statistics* table where it provided details about the number of votes in a county for each party. Since many counties make up a state, we created a relationship between *county_statistics* and *president_county*, which has relationship sets to determine who the candidate of a county is, and the state information of that county. This provides us with information on which state the vote came from.

Two other important relations we derived in our design were polling data and demographics. Although the polls may not be a clear indicator of the winner of the election, they do play a major part in pointing the direction of where the election is heading so that's why we felt it was information to incorporate. We also added demographic data because historically political parties held certain identities that drew the attention of specific demographics. This information is incredibly useful to help identify which part may win in what region, because of the strong presence of certain demographics.

The last key element in our ER design is a relationship set that connects the *county_statistics* with two more entity sets, one being a dataset that provides us with the winner and candidate of a county, and the other providing us with information regarding the state in which that county belongs.

Summary:

The CLI application that was created has its fair share of remarkable successes in what it can do (given the hefty roster of both ideal requirements and proposed requirements) and what could not be done (due to time, technical problems, or simply by design). While it could not find the exact information of all existing data on the available CSVs, it was able to uniquely provide information on more important and critical data, such as winners and losers of certain states and counties, demographic summaries on a county and state basis, and historical data to compare with. That is not to say that the CLI could not be improved - it is far from perfect and stands to improve with better and more definitive actionable arguments, as well as larger scope of options, and patterns. Understandably, the largest piece that is missing from the CLI is proper data-mining, which would allow users to effectively discover patterns within the datasets to truly get a more mathematical relationship between data points. The ER diagram and its design tries its best to respect the US electoral structure, as this is best suited to represent the classical needs of what a client would want to look for and understand in an application like this.

While our CLI and database are intended to represent a promising amount of information about what can swing a state and other observable patterns, it is not definitive proof that whatever findings may be uncovered can be directly correlated to a state's, and ultimately a candidate's, victory. Any findings are at most conjectures as the CLI serves to merely represent the data it has been provided. There are certainly more contributions and factors that are not present in the datasets we've been provided, even given our vast amount of knowledge that can be acquired in our era of digital information, it's possible that what we have only happens to represent a modicum of relevant data to which how an election is truly won. Ultimately, this was an interesting project to develop to uncover possible trends and patterns, to which we may not know with certainty how much it relates to the election outcome (perhaps Russia does), but proposed plenty of useful possibilities regardless.