The More You Snow

Chloe Kurzon, Andrew Moran, and Jay Chakravarty

Final Project CS171 (Spring 2015)

Process Book

Table of Contents

CS171 Project Proposal

Background and Motivation

Snowfall has been mentioned by several leading environmental agencies as an important indicator of climate change globally. In light of recent snow events in Boston (i.e. Boston having its snowiest year on record), we thought it would be interesting to visualize changes in snow patterns over time throughout the U.S. to see if recent patterns are actually abnormal, and could therefore indicate real changes in the earth's climate. We want to create a visualization that makes it easy for users to compare changes in snow patterns for the entire U.S. and smaller regions over time, and relative to each other. To accomplish this task, we plan on making use of the NASA MODIS dataset, which uses satellite imagery to create a dataset that provides snowfall data on the latitudinal/longitudinal level. In addition to relaying interesting information about snowfall, we decided to pursue working with the NASA MODIS dataset because we thought it presented interesting challenges related to visualizing large datasets. This dataset is also particularly appropriate for this task because it provides the data in a format that will make it easy to create a comprehensive map-view of snowfall in the United States. Additionally, one of our team members, Andrew Moran, is currently involved in a research project related to the utilization of this dataset.

Project Objectives

Our main goal is to create a detailed visualization of snowfall in the US from satellite imagery data. We want a very interactive and intuitive interface that makes it easy to discover snowfall patterns over time. With current concerns of climate change, we thought that users can potentially predict weather patterns based on previous measures in snowfall amount, coverage, duration, etc. This could eventually extend to making comparisons and drawing conclusions of snowfall in other regions of the globe. We plan to achieve this using a map-based layout. Remaining in the geospatial domain will allow us to experience the added utility of superimposing a large dataset on a well-known geographical display. The user will be given the freedom to browse any region of interest and request more detail via added navigation features (zooming, panning, etc). Creating our own map interface will give us more insight on how other common navigation tools work (e.g Google maps, Open StreetMap). In addition to usability, we plan to exploit the benefits of representing data in a tile-based format. This format allows data to be queried and aggregated faster, providing a rich/high resolution display with the reduced time cost. Also, this representation will allow our visualizer to remain running smoothly without the loss of detail. Enhanced performance is a large benefit, especially is there is potential to use this visualizer for other datasets.

Data

Data will be acquired from the NASA MODIS site. The NASA MODIS is a satellite instrument that records satellite imagery data. This data is stored in a 3D array indexed by latitude, longitude and time (taken from one week of MODIS measurements). We may limit the time range to minimize the scope of the data we are working with. To start, we will be

visualizing global snowfall coverage. However, this can potentially be extended to other satellite imagery information such as vegetation estimates on land, phytoplankton populations in the ocean, etc. (Battle et al., unpublished, see attached)

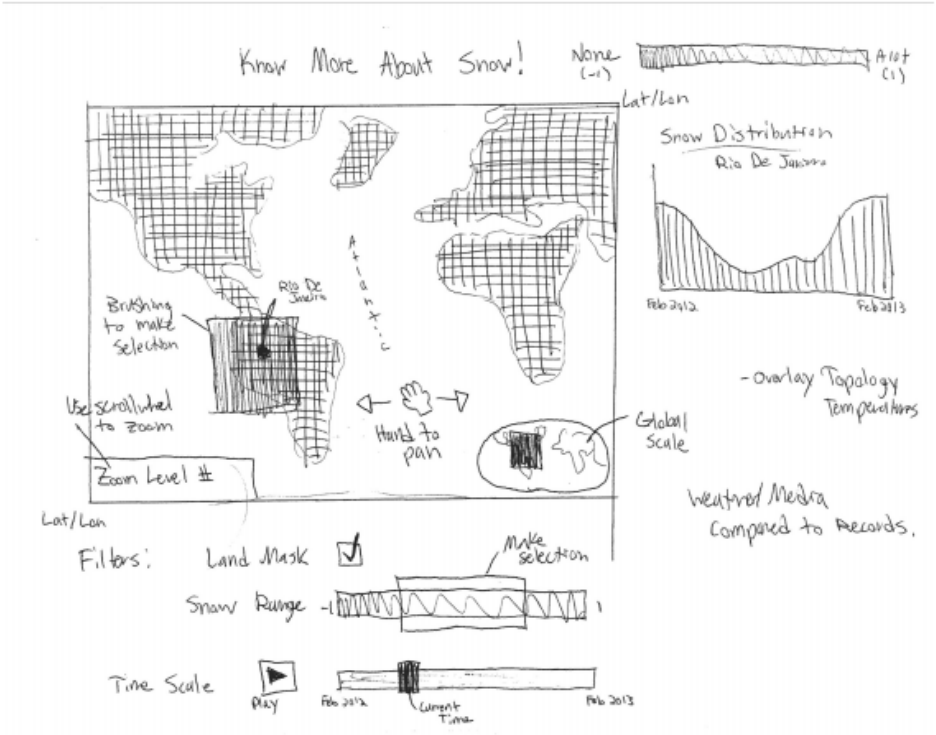
Data Processing

Not much data processing needs to be done besides reformatting the MODIS data into a tile format to help with zooming. We will be using numerical, array-based data so extracting information would not require additional processing. If we extend to non-integer based data, we will utilize Google Refine. (Battle et al., unpublished, see attached).

Visualization

For our visualization, we plan to have a map of the US (with the plan of expanding to a map of the entire world, if we have time). We will use a pixel-based method where each pixel is colored according to the amount of snowfall in that lat/lon region. In our snowfall example, pixels are colored on a gradient corresponding to a low or high NSDI (Normalized Snow Difference Index) value. Similar to a Google Maps view, users can navigate by zooming & panning. Additional widgets (e.g., scales, radio buttons) on the bottom of the screen will help filter pixels by various characteristics recorded in the dataset (e.g., whether it is a land/ocean pixel, min/max snowfall). The map view will also include an additional view on the side that includes a graph of historical snowfall data for a selected region. Additionally, we will include a table-view of the data set with columns sortable by continent, country, and snowfall level.

Initial Sketch:



Must-Have Features

- Zooming
- Panning
- Filtering - Choose min/max snowfall and only view regions that fall within this range; land/sea mask
- Selection/Brushing - change the view of a graph of the snowfall for the selected region.
- Tool-tips (hover over a pixel to view info specific to the tile/pixel)
- Timeline slider/play button
- Sortable table with data
- Only a map of U.S.

Optional Features

- Overlay height map data with NDSI data (similar to http://colorbrewer2.org/)
- Expand map to entire world
- Overlay of country borders
- Interchangeable MODIS data (Snowfall, vegetation, etc)
- Secondary map to show zoom level in relation to global scale
- Overlay of other relevant data sets (e.g. agriculture, wind speeds, weather-related diseases)
- Pre-set zoom features (e.g. person can select a country or continent from a drop down menu and view the zoomed version automatically on the map).
- Aggregate data by country and/or continent
- Small-secondary map that shows current zoomed-in position relative to the global map as a whole.
- Option to change the relative color scheme of a selected region (e.g. only scale based on the selected region vs. scaling for the whole globe).

Project Schedule

We plan to meet the milestones outlined on the course project page. Below is our project outline in accordance to key dates/milestones.

- W1 (Saturday, 04/11/15): Setup Repo for Development - Server/Client/Data
- W2 (Friday, 04/17/15): Simple Map-Based Visualization (color-coded by snowfall with tooltips), table with sortable columns **Milestone 1**
- W3 (Friday, 04/24/15): Fundamental Navigation (zooming,panning), time slider, filter by max/min snowfall **TF Review**
- W4 (Friday, 05/1/15): Feature Implementation (aggregation, filtering), additional views (graph of snowfall for selected region, separate view of zoomed in area).
- Tuesday, (05/5/15): any additional optional features we have not yet implemented, video
- **Final Submission**

4/7/15 - Meeting after class

[entry by amoran]

Today our team discussed about data representation/acquisition. Andrew met with an outside class member working on a related project about potentially integrating NASA MODIS data. It is up to the team's discretion how to format the dense dataset so it is easily readable/parable. We will come up with a basic JSON format and change it as we start developing our prototype. Meanwhile, we will continue to explore map visualizations. In particular, D3 implementations.

By Saturday:

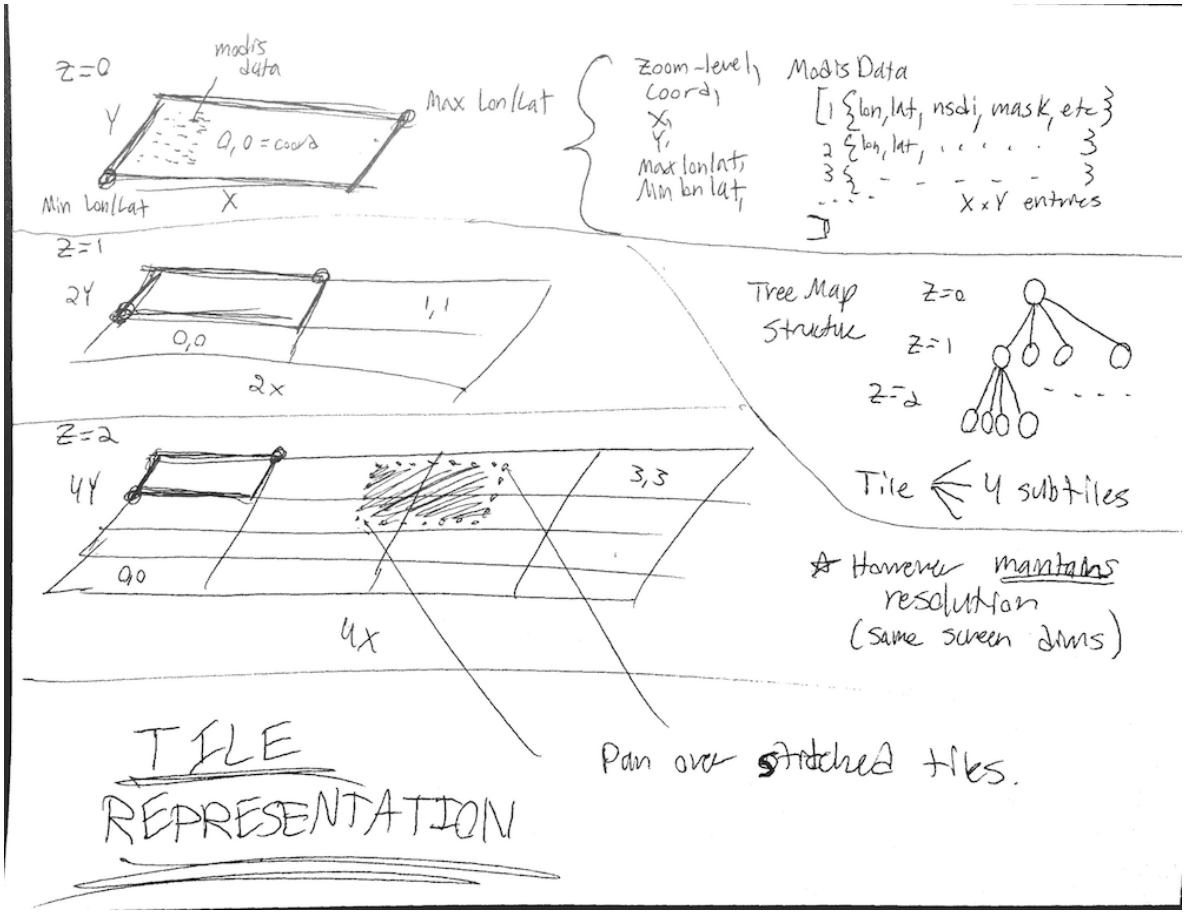
Andrew plans to have a data format template, a written script for JSON request from server, and a small sample ready. Potentially reach out to TF on suggestions for dense data rendering on browser (potential issues such as lagging, etc)

Chloe and Jay will explore current D3 map projections/tools and how to bind data to simple map layouts.

4/8/15 - Data/Tile Representation

[entry by amoran]

Continued to think about data/tile representation. Below is a sketch on how I picture the tile scheme working



It will be a recursive treemap representation where tiles remain the same size but become aggregated the higher level zoom level you are. Just to start of with, each tile will have a JSON format like the following:

```
id: {dataset: “modis”,
"zoom": 0,
"coord" : [0,0]},
“dims”: [“lat”,”lon”,”time”],
“resolution” : [1024,1024,0]
"min" : [0,0,0],
"max" : [1024, 1024,0],
"data": [
{ "lon" : 0,
```

```
  "lat" : 0,
  "avg_nsdi" : 0.5,
  "min_nsdi" : -1,
  "max_nsdi" : 1,
  "count_nsdi" : 100,
  "mask" : 0
},
{}, {}, {}, {}, {}          // ... 1024 X 1024 entries
],
“parent” : {#id}
“neighbors” : [{#id}, {#id}, {#id}, {#id} ]
```

link to retrieve data:
http://modis.csail.mit.edu:11001/fetch/0_1_0

tile id format for link: “.../fetch/<zoom>_<xcoord>_<ycoord>”

current implementation (1 tile):
Note: min max = range of tile boundaries ("min": [0, 0], "max": [3599, 1799])
lows/highs = upper/lower range of the given tile ("highs": [149, 149], "highs": [149, 149])
tile “0_0_0” has (-179,-89) as its lowest coord range
tile “0_23_11” has (179,89) as its highest coord range

from python:

```
“    return {
        'id':None,
        'attrs':[],
        'dtype':{},
        'data':[],
        'parent':[],
        'neighbors':[]
    }”
```

data sample:
"neighbors": [{"arrayname": "ndsi_agg_04_10_2015", "min": [0, 0], "max": [3599, 1799], "highs": [149, 149], "zoom": 0, "dims": ["longitude_e4ndsi_agg_04_10_2015", "latitude_e4ndsi_agg_04_10_2015"], "coords": [1, 0], "lows": [0, 0], "resolution": [50, 50]}, {"arrayname": "ndsi_agg_04_10_2015", "min": [0, 0], "max": [3599, 1799], "highs": [149, 149], "zoom": 0, "dims": ["longitude_e4ndsi_agg_04_10_2015", "latitude_e4ndsi_agg_04_10_2015"], "coords": [0, 1], "lows": [0, 0], "resolution": [50, 50]}],

"parent": [],

"dtype": {"min_land_sea_mask": "uint8", "min_lat": "double", "max_ndsi": "double", "longitude_e4ndsi_agg_04_10_2015": "int64", "min_ndsi": "double", "avg_ndsi": "double", "latitude_e4ndsi_agg_04_10_2015": "int64", "max_lat": "double", "max_land_sea_mask": "uint8", "min_lon": "double", "max_lon": "double", "avg_lat": "double", "avg_lon": "double"},

"attrs": ["avg_ndsi", "max_ndsi", "min_ndsi", "max_land_sea_mask", "min_land_sea_mask", "min_lat", "max_lat", "avg_lat", "max_lon", "min_lon", "avg_lon"],

"data": [
{"min_land_sea_mask": 1, "min_lat": -89.9748, "max_ndsi": 0.4697842395345295, "longitude_e4ndsi_agg_04_10_2015": 0, "min_ndsi": 0.12016576445557625, "avg_ndsi": 0.3397024633537475, "latitude_e4ndsi_agg_04_10_2015": 0, "max_lat": -89.7007, "max_land_sea_mask": 1, "min_lon": -179.998, "max_lon": -179.7068, "avg_lat": -89.84217620830057, "avg_lon": -179.84690677448958},
{"min_land_sea_mask": 0, "min_lat": -84.3, "max_ndsi": 0.9070968514112915, "longitude_e4ndsi_agg_04_10_2015": 0, "min_ndsi": 0.27715124031127036, "avg_ndsi": 0.7195162319970785, "latitude_e4ndsi_agg_04_10_2015": 19, "max_lat": -84.0001, "max_land_sea_mask": 0, "min_lon": -165.0, "max_lon": -164.7001, "avg_lat": -84.1502631182468, "avg_lon": -164.84991795994148}
],

"id": {"arrayname": "ndsi_agg_04_10_2015", "min": [0, 0], "max": [3599, 1799], "highs": [149, 149], "zoom": 0, "dims": ["longitude_e4ndsi_agg_04_10_2015", "latitude_e4ndsi_agg_04_10_2015"], "coords": [0, 0], "lows": [0, 0], "resolution": [50, 50]}

land sea mask definitions

Found this here:
http://lgge.osug.fr/~picard/enseignement/fichiers/Teledetection_M1/TPTemperatureSurface/MOD03.geolocation.fs

DN values:

0: Shallow Ocean (Ocean <5k from coast OR <50m deep).
1: Land (not anything else).
2: Ocean Coastlines and Lake Shorelines.
3: Shallow Inland Water (Inland Water < 5km from shore OR < 50m deep).
4: Ephemeral (intermittent) Water.
5: Deep Inland Water (Inland water > 5km from shoreline AND > 50m deep).
6: Moderate or Continental Ocean (Ocean > 5km from coast AND > 50m deep AND < 500m deep).
7: Deep Ocean (Ocean > 500m deep).

4/13/15 - First Meeting With TF

[entry by ckurzon]

- Daniel likes our project idea but is concerned about the difficulty with implementing the tile architecture. He thinks zooming with tiles will be extremely difficult
- Daniel suggests just implementing a global map with interactive and not implementing zoom unless we have extra time.
- Daniel is concerned about datasize. We need to make sure that on each load of the data, no more than 1 MB is loaded
- Daniel has concerns that the backend of our project is coming from an MIT Grad student who is not ultimately being graded for this project
- Need to remember that this project is focused on the visualization and front end, not on having a complex backend
- We realize during the meeting that we might not have a time component for our dataset (i.e. the data we are receiving might just be for one point in time).
    - This severely restricts our options for interactivity. Daniel is concerned about the number of views we can implement with only snowfall data for one point in time
    - Andrew is going to talk to the MIT grad student who is providing us the data to see if we can get a time component for it.
- Daniel has data concerns and states that we need to have a function to quickly request snowfall data by latitude and longitude by friday if he is going to feel comfortable with the state of our project
- Following the TF meeting, we decide to consider switching datasets and/or projects, pending Andrew's discussion with the MIT grad student who is providing us with data.

Update:

- Andrew talks to the MIT grad student who is providing us with this data, and ultimately decides that this dataset is not the best for our project considering the amount of time we have
- Team meeting is scheduled for tomorrow (Tuesday) to discuss potentially switching projects
- Daniel emails us a suggestion for a different dataset for snowfall data, that is on a much smaller scale than the one we were considering using before, from the NCDC: https://www.ncdc.noaa.gov/snow-and-ice/daily-snow/

4/14/15-Team Meeting and Design Studio

[entry by ckurzon]

- Team meeting to discuss switching project ideas because we don't think our current dataset is ideal for this project
- Consider switching to World Health Organization data, but ultimately decide to use the new dataset Daniel recommended and stick with our original sketches for thimplementation
- Unsure if we want to focus on snowfall data or cumulative snow data (can do both with new dataset)
    - Could possibly represent cumulative snow data with a 3D map
    - Could allow users to toggle between both
- To-Dos:
    - Andrew: By Wednesday night, Andrew will download all of data from NCDC and merge it into one tsv file that is organized in logical manner for parsing
    - Jay: will work on loading in the data
    - Chloe: make a draft of a static map visualization encoding the new dataset by meeting on friday
- Design studio feedback summary:
    - should probably stay away from 3D maps
    - Need to come up with a tool to compare different counties-possibly through a bar graph when brushing
    - Need to add a feature that allows users to aggregate by county or state

4/15/15- Map Projections

[entry by jchakravarty]

d3 map projections:

It is important that we find a d3 map projection appropriate to our dataset. One possible option is d3.geo.mercator() which displays the entire world without curvature.

The spherical Mercator projection is commonly used by tiled mapping libraries (such as OpenLayers and Leaflet). For an example displaying raster tiles with the Mercator projection, see the d3.geo.tile plugin.

Mercator Script:

```
<script type="text/javascript" src="d3/d3.v3.js"></script>
<script src="js/topojson.v0.min.js"></script>
<script>
var width = 960,
    height = 500;

var projection = d3.geo.mercator()
    .center([0, 5 ])
    .scale(900)
    .rotate([-180,0]);

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

var path = d3.geo.path()
    .projection(projection);

var g = svg.append("g");

d3.json("json/world-110m2.json", function(error, topology) {
    g.selectAll("path")
      .data(topojson.object(topology, topology.objects.countries)
          .geometries)
    .enter()
```

```
    .append("path")
    .attr("d", path)
});

</script>
```

However, now that we are using the tabular U.S. daily snowfall and snowdepth data from the National Oceanic and Atmospheric Administration, it will be more appropriate to use a map projection displaying solely the U.S.

Furthermore, given the area distortion potentially introduced by the mercator projection, it will create issues for choropleths (map in which areas are shaded or patterned in proportion to the measurement of a statistical variable being displayed on the map, such as, in our case, snowfall and snowdepth).

A better option may be d3.geo.conicEqualArea() which is more suitable to choropleths given that it preserves the relative areas of geographic features.

This projection also includes county lines which may be appropriate divisions by which the shade the map. Luckily, the data already includes the county of every weather station at which metrics have been recorded so the process of aggregating snowfall and snowdepth by county will be relatively straightforward.

4/15/15- Parsing the New Dataset
[entry by andrewmo]

- Due to foreseeable issues with the tiling schema and more overhead required to work with the complicated dataset, we decided to work with a new one.
- Thanks to Daniel's advice we found the below dataset to use for recorded snowfall: http://www.ncdc.noaa.gov/snow-and-ice/daily-snow/ (sample shown below)

State: AL
Lat Lon COOP# StnID State City/Station Name County Elev Apr 1 Apr 2 Apr 3 Apr 4 Apr 5 Apr 6 Apr 7 Apr 8 Apr 9 Apr10 Apr11 Apr12 Apr13 Apr14
33.59 -85.86 010272 AL ANNISTON ARPT ASOS CALHOUN 594 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 -9999.000

- Working with this dataset has its pros and cons.
  - Pros: Many attributes, dates back 10 years, many counties, lat/lon points
  - Cons: .txt file, needs to be reformatted to be utilized in d3. I decided to write a python script to convert this text file to JSON/CSV.

Parsing Implementation:

- Created a python script that grabs .txt from url
- I iterate through all desired months and years to get a unique snowfall/depth for that date. (update string url)
- First I want to remove unnecessary lines:
  - newlines, current state, headers
- Next, I thought that all I had to do was split each line by spacing, however, some values are names that are multiple words, etc. I discovered that each value had a fixed amount of spacing for that header field. EUREKA!
- Read so many characters for that line and pointer shows which header value you are reading.
- Something else was that not all months had the same number of days. The .txt file only showed days leading up to the end of the month. For consistency I made an empty placeholder for all possible days and did not populate the days field if it did not exist (e.g. Feb 30)
- Lastly I stripped any additional padding on either side of the desired field value and separated each line by a tab and wrote that line to a new file.
- Each file is about 115MB (Aggregated for all months all states for 10 years)

Added Files:

counties.json (not sure if necessary)

```
[
        {state: “Alabama”,
        counties: [ … ],
        abbrev: “AL”
        },
        …
]
```

(A mapping of states with their abbreviation and a list of all county names)

snfl.tsv (same idea for sndpth.tsv)

- .tsv instead of json so dont have to do with brackets, commas, spacing, etc
- uncollected data is empty represented as empty string “”
    - This happens for no ids for stations and days that do not have full 31 days.
- invalid data is shown as -9999.000
- not sure what range of values are just yet (most extreme)
- similar to previous psets need to aggregate by month/year/state/etc to get other insight

Below is a sample of the formatted data:

latitude		longitude		coop	id	state	station	county	elevation		year	month
day1	day2	day3	day4	day5	day6	day7	day8	day9	day10	day11	day12	day13
day14	day15	day16	day17	day18	day19	day20	day21	day22	day23	day24	day25	day26
day27	day28	day29	day30	day31								
34.74	-87.60	015749			AL	MUSCLE SHOALS AP			COLBERT		540	
2005	07	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000						

Look at https://github.com/mbostock/d3/wiki/CSV to utilize delimited files in d3

Worth mentioning

- Moving forward, Below are some example of county maps I believe can be useful.
    - http://vizhub.healthdata.org/us-health-map/
    - http://code.minnpost.com/simple-map-d3/
    - http://bl.ocks.org/mbostock/4060606
    - http://bl.ocks.org/rwaldin/6244803
    - http://scribu.net/blog/3d-maps-using-d3-and-three.js.html
- I think it is good to note down some libraries we should utilize for development (Libraries to use):
    - topojson
    - d3.js
    - d3-threeD.js
    - three.js

- colorbrewer
- bootstrap

4/16/15 - Loading Data
[entry by jchakravarty]

In order to upload the tsv file we will need to use:

d3.tsv(url[, accessor][, callback])
Issues an HTTP GET request for the comma-separated values (CSV) file at the specified url.

Using:

```
var downloaded_data;

d3.tsv(data/sndpth.tsv, function (error, data) {
          downloaded_data = data;
});
```

The data is available via console.log(downloaded_data);

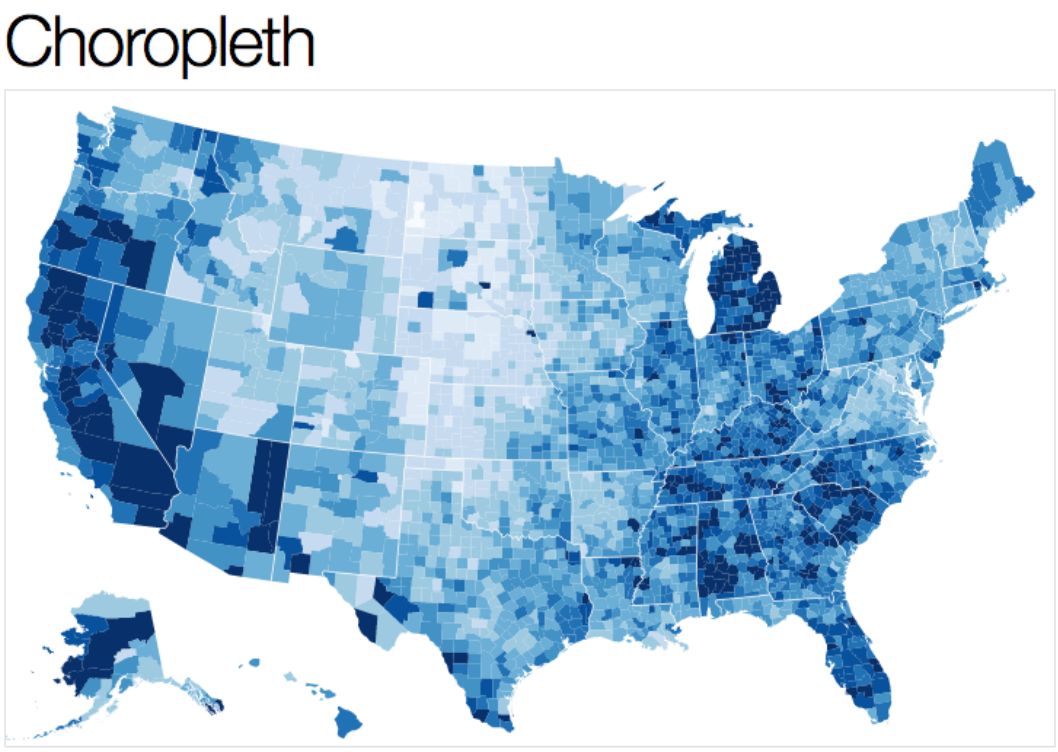
Once the TSV data has loaded we execute the following script within the d3.tsv function in order to build the object array which will eventually be populated with accumulated snowfall data for each county.

```
 data.forEach(function(d) {
    for (j = 0; j < county_state_list.length; j++) {
      if (county_state_list[j] == d.state + ", " + d.county) {
        listed = true;
      }
    }
    if (!listed) {
      county_state_list[county_state_list.length] = d.state + ", " + d.county;
    }
    listed = false;
 });
for (k = 0; k < county_state_list.length; k++) {
   aggregate_list[k] = {county: county_state_list[k], accumulation: 0};
 }
```

This constructed aggregate_list contains objects of the form {accumulation:0, county: “state, county-name”}. Once interaction with the slider occurs, the function year_value is called which will update the value stored under the county key to reflect an accumulated value.

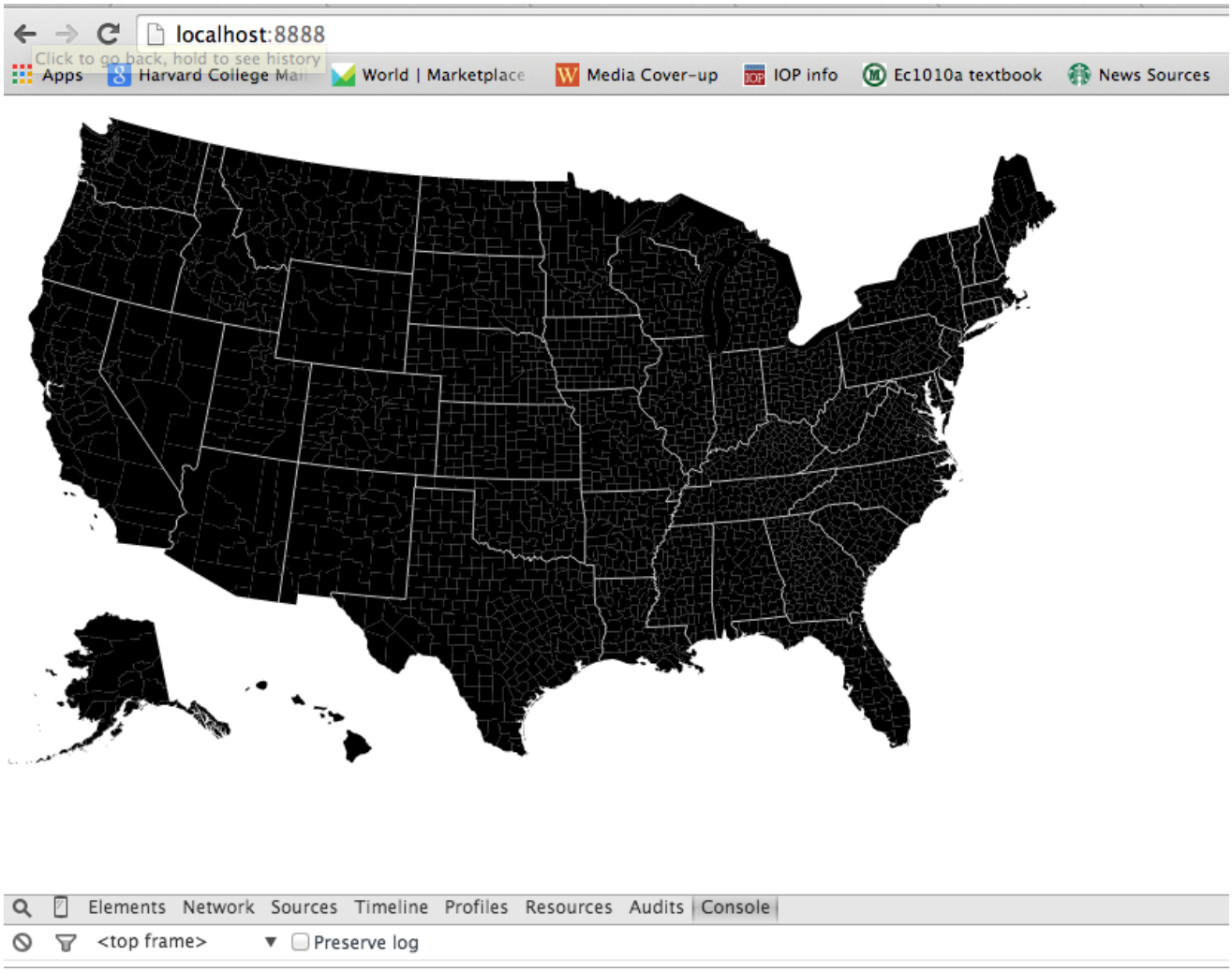
4/16/15-Working on Mapping Data
[entry by ckurzon]

Goal: To implement a choropleth U.S. map of a snapshot of the dataset that Andrew has parsed. The color coding should be based on the county's snowfall on the selected day. I want my map to look something like this:



Source: http://bl.ocks.org/mbostock/4060606

Methods: I first copy the code from this example. I noticed that the tsv file used has counties in the form of IDs. After looking at several other D3 choropleth maps, I discover that this ID scheme for counties is standard. I learn that this ID scheme is referred to as “FIPS”. This is a problem because our dataset only has county names and no county ids. I find a text file online that gives the FIPS for each individual county at the following link:
http://www2.census.gov/geo/docs/reference/codes/files/national_county.txt
I import this into excel and use a vlookup to match the entries in our dataset with their FIPS code.
After completing this, I then map the data using January 1st 2015 snowfall. The resulting map looks like this:



There’s clearly an issue with either the data or the color scale, but I was not able to figure it out at this time.

Remaining To-Dos:

- color scale of the map is not working for some reason, map is all just one color
- need to figure out how we want to deal with missing values in the dataset
- At the moment, some counties have multiple weather stations and we need to average out this weather station data so that each county only has one data point representing it.

4/17/15- Team Meeting

[entry by ckurzon]

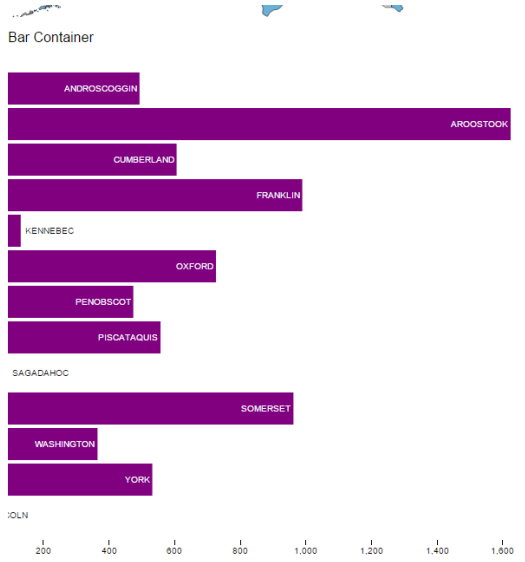
- We have a map display of a static portion of our visualization, and next need to work on color-coding it and on making the visualization interactive
- Task assignments due on Monday, 4/20/15:
    - Andrew: Fix the color scheme of the map implementation; create the layout for the svg elements (i.e., map out where the different views should go)
    - Jay: Make a function that aggregates the data by year, e.g. function takes in year as an argument, and then sums the snowfall data for each county in that year. Add a time slider.
    - Chloe: Make a static histogram of snowfall by county
- Will next meet on Sunday to do a progress check and see if people are running into issues
- Will combine all of our separate parts on monday/tuesday to have an interactive version by TF meeting on Wednesday.

4/19/2015 - Initial Bargraph

[entry by ckurzon]

For an additional view to the map, we thought it would be interesting to include a bar graph that allows users to compare counties. When a user clicks on a county, the graph will be updated with snowfall data for all of the counties in the same state as the selected county. Initially, this graph will aggregate snowfall by year for each county. Our inspiration for this graph came from:
http://vizhub.healthdata.org/us-health-map/

Screenshot of initial graph:



Remaining to-dos: Graph axiis need to be fixed. Additionally, the data aggregation is slow so we need to fix this. The graph also can be unwieldy for states with a large number of counties (e.g. Texas), so we need to figure out a way to manage this. As of right now, this graph is not interactive with the map

4/21/2015 - Slider

[entry by jchakravarty]

Insert the following piece of code at the top of the <body> section. This will create the interactable slider in HTML that calls the year_value() function whenever it is manipulated.

Time: 2005 <input type="range" name="points" min="2005" max="2015" step="1" value="0" id="slider-time" oninput="year_value();"> 2015

The year_value function stores the year currently selected by the slider in the variable year and then calls by_year, passing in year as the single argument:

```
function year_value(data) {
    d3.selectAll("input").each(function(d) {
        if(d3.select(this).attr("type") == "range") {
            year = (this).value;
        }
    })
    by_year(year);
};
```

by_year proceeds as follows:

```
function by_year(year) {
  var str_year = year.toString();
  var list = [];
  var listcount = 0;
  console.log(year);
  for (i = 0; i < downloaded_data.length; i++) {
    if (downloaded_data[i].year == str_year) {
      list[listcount] = downloaded_data[i];
      listcount = listcount + 1;
    }
  }
  for (i = 0; i < list.length; i++) {
    for (j = 0; j < aggregate_list.length; j++) {
      if (aggregate_list[j].county == list[i].county + " " + list[i].state) {
        if (parseInt(list[i].day1) > 0) {
          aggregate_list[j].accumulation = aggregate_list[j].accumulation + parseInt(list[i].day1);
        }
        if (parseInt(list[i].day2) > 0) { ...
```

Given that the argument year is passed in as an integer, it must be converted to a string, using the function toString(), in order to utilize conditional equality to filter only for station data points from the year selected by the slider.

The by_year function iterates across the entire dataset, constructing a list containing only data objects from the selected year.

The current script implements accumulation over the days of each month by hard-coding the keys (day1, day2, day3, etc.) This is not good style but was the only way of iterating through these keys to accumulate their values.

4/21/2015 - Team Meeting

[entry by ckurzon]

As of right now, this is what our visualization looks like:

Discussion:

Data:
Loading the data is very slow, as is aggregating the data. We did a sanity check on the values for our bar graph and map, and realized they don't match up. After further discovery, we realized that this is because we are incorrectly handling counties with multiple weather stations. For the map, we were just overwriting weather stations, which resulted in our aggregation only including one month's worth of data. For our bargraph, we realized we were adding up the data every weather station, which multiplies the amount of snow the county actually received. We realize that we need to correct how we deal with multiple weather stations.

Bar Graph:
We decided that the bar graph showing multiple counties is too awkward in cases where where states have a lot of counties. Instead, we decided to do a bar graph showing monthly snow data for the selected county (where the selected county is selected through clicking and/or brushing)

Post-Meeting Task Assignments:
Chloe- Chloe is going to manipulate the data in excel so that it can be easily loaded and corrects for the multiple weather stations. She is also going to change the bargraph to one for monthly data rather than county-level data.
Andrew- Andrew is going to bind data to the counties of the map (right now, the data isn't bound, it's just used for coloring the counties) and add interactivity between the map and graph.

4/21/2015 - Aggregating Data

[entry by ckurzon]

Problem: Our current dataset has daily values for every county for 10 years, in addition to multiple weather stations for some counties. It also has numerous extraneous columns that we aren't using for our project (e.g. latitude and longitude). As a result, it loads very slowly and aggregating the dataset is a long process. My goal is to make the dataset smaller and to deal with multiple weather stations. To handle multiple weather stations, we are going to take the average snow level for all weather stations of a given county that have snow recordings for the given day. This is going to be done in excel.

Step 1. Delete all extraneous columns (e.g. lat, longitude, weather station name)

Step 2. Sum all daily data for each weather station to get a monthly value for each station. I can then average these station values. Only summing if the value is >= 0 (because -9999 means there was no observation for that day)

Step 3. Average this monthly data. This was done by first creating a separate excel sheet with the name of the county, state, and month, and year, and making sure these rows were unique. I was then able to compute the average using the original sheet with an AVERAGEIFS function (checking to make sure that month, county, state, and year matched for the average).

Notes: It was impossible to run the AVERAGEIFs function on more than 3,000 rows of excel without crashing the computer, so I spent 1 ½ hours on a desktop running each chunk of rows separately (there were over 300,000 rows).

Final Data Set: Our resulting data set has five columns: county, state, month, year, and "monthly". "Monthly" is the monthly average for counties across all weather stations. Making these edits shrunk the dataset from over 100 MB to 4.4 MB, so it now loads much faster and is easier to aggregate.

4/22/2015 - Data Binding for Map, Time Slider

[entry by amoran]

In order to populate the map with the necessary data, we utilized three datasets:

- counties.json
- FipsCountyCodes.json
- us.json

The key idea of how d3 draws a US county map, multiple paths are drawn as Polygons and/or Multi-polygons for each county. Each county is assigned a unique id mentioned earlier known as a FIPS id. However, our dataset we already used did not contain FIPS information. Only state and county names. This served as a basis for a dictionary lookup in associated to the json file FipsCountyCodes. There needed to be a defined convention to correctly associate county names and states to their fips codes (re-represented in the json file, sample)

```
["02020", "AK, Anchorage"],
```

When clicking a county, there needs to be a unique key (or keys if brushing) that could be passed into the coordinated views of the bar and line graphs showing a breakdown of the aggregated data. This was done with event triggers of passing in fips codes.

Another functionality to incorporate is to update the time slider to properly update the map. We would like this to be done sooner than later to ensure the map is properly reading in data and to determine any thresholds/scales to make the viewing experience of snowfall data more meaningful. We hope to come up with a nice color code for the map soon.

4/23/2015 - Time Slider, Tool Tips

[entry by amoran]

To help see patterns faster and draw conclusions from the county map, we wanted to ensure that we got a time slider implemented correctly. The idea is that everytime someone clicks thicks/moves the slider, the map would update accordingly for the aggregated snowfall for that year. But after completing the first iteration of the timeslider, there were two main concerns.

- Performance

  Every step of the slider caused the visualization to pauses a second or two. We were actually filtering the data every time the slider changed. Since we already know that the data will be filtered per year, I decided to catch all possible variations of the map (11 from 2005-2015). This made the time slider move much faster! Its turned out that
- Propagation

  I noticed something a little off that once you use the slider multiple times in a row, its starts to slow downs a bit. After further research, most of the code to reproduce the map (path generation, color classes) were being repeated in the update visualization method after the data has been wrangled. Luckily this was removed and everything seemed to run smoothly.

Once hovering over a county, it would make sense to get information for that particular county. This can be done with pops ups and hoverovers. Initially, I decided to work with oaths since thats how counties were original being drawn in d3. However, I kept running into errors/artifacts when a fips ID wasn't found, unrecorded daya, etc. There were holes in the data that actually continued to draw through objects with the same amem. We found some resources below
d3 Tooltips
Color Brewer

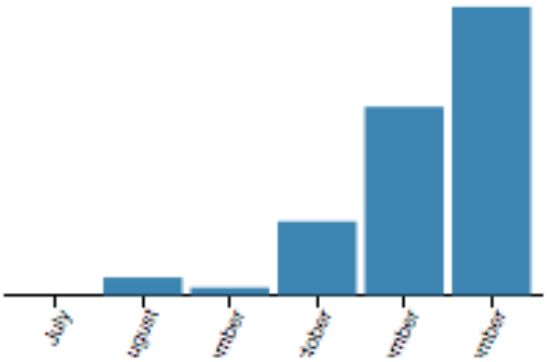
Eventually, we just customized divs with classes that displayed/filled when needed. This is one of the many ways in which I saw how the process of getting days but showing it to the user more effectively.

4/24/2015 - Monthly Bar Graph

[entry by ckurzon]

Motivation: We thought including a bar graph that shows the monthly data for counties in addition to the map which has yearly values, would be a good way for users to be able to interact with the data and observe which months had the most snow (indicating that there might have been large snowstorms in these months)

My goal is to create a bar graph that shows monthly snowfall for the county/counties selected. This will involve just creating a simple bar graph using the new monthly dataset. The only slightly tricky part of this graph is filtering for the counties by month and year, and aggregating the counties by month when multiple counties are selected. To efficiently filter by year, I nested the data so that the keys are the years and values is the data for each year. This allowed me to easily roll up the data by month and aggregate it. The resulting graph looks like this:



(The names are cut off because I need to change the margin size)

Remaining Issues: The graph loads quickly and interacts with the map well. I still need to fix the axiis and implement error checking for missing values (right now if we are missing data for a particular county, the graph breaks)

4/28/2015 - Daily Line Chart

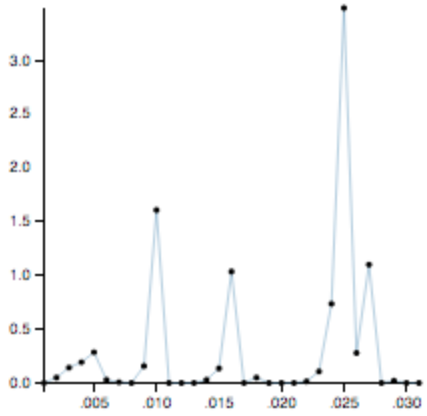
[entry by ckurzon]

Motivation: Since we have daily data for snowfall, we figured we should take advantage of this and let users see the daily data for a particular month for selected counties. This would allow users to more easily see when there were large snowstorms.

We decided that we're going to represent this daily data as a line chart. When the user clicks on a month in the bar graph, this line chart will be updated with the daily data for the selected counties for that month.

The biggest challenge for implementing this line chart is filtering the very large daily dataset down to only one month and only certain counties. To make this more efficient, I first nested the data by year and then by month. Then to filter, I used this nested dataset to quickly cut down the data by year and month before filtering for county. This dataset was also challenging because we still have the multiple weather station issues with the daily data (we only corrected this data issue in the aggregated monthly dataset). Excel is too slow to feasibly average the daily data by weather station, and we thought writing a python script to correct this would take too long. So my filterAndAggregate function also averages over counties after the data has been filtered.

Resulting graph:



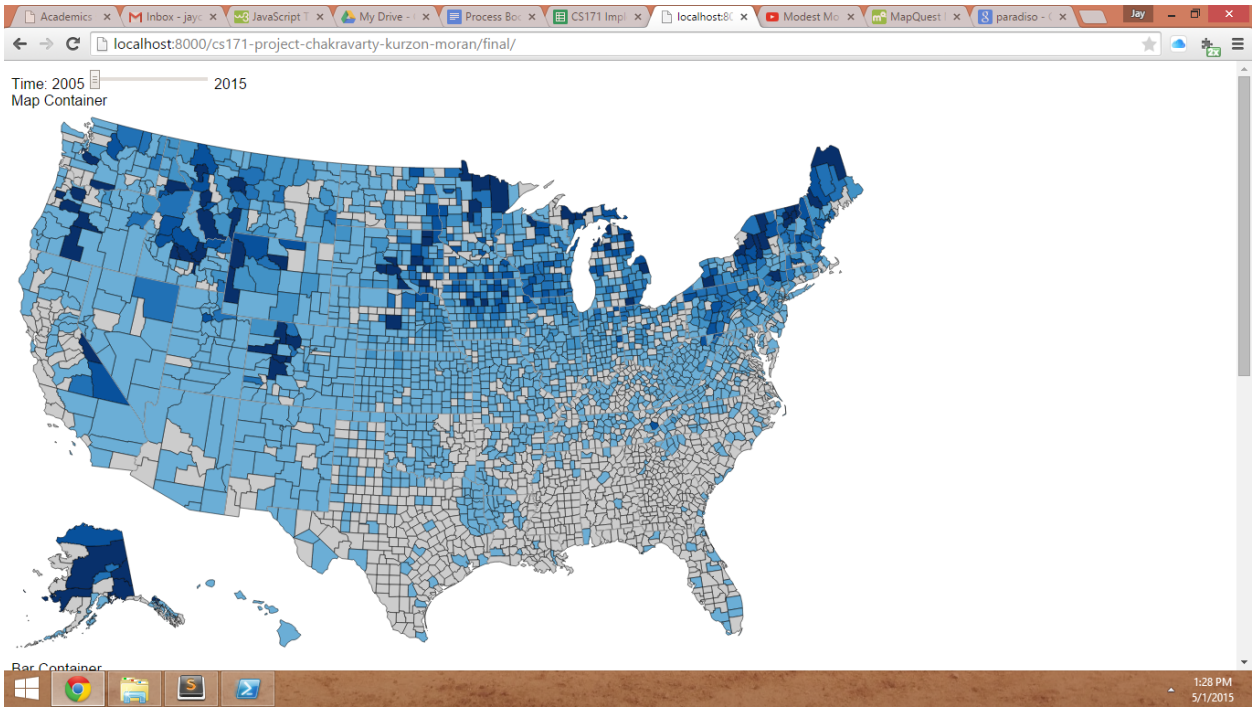
Using the nested filtering, the interaction between this line chart and the bar chart is surprisingly fast.

Remaining To-Dos: We want to allow users to toggle between viewing snow depth and snowfall data for this line chart. To do this, we'll need to decrease the size of the snowfall/depth daily data in order to load both of these datasets efficiently.

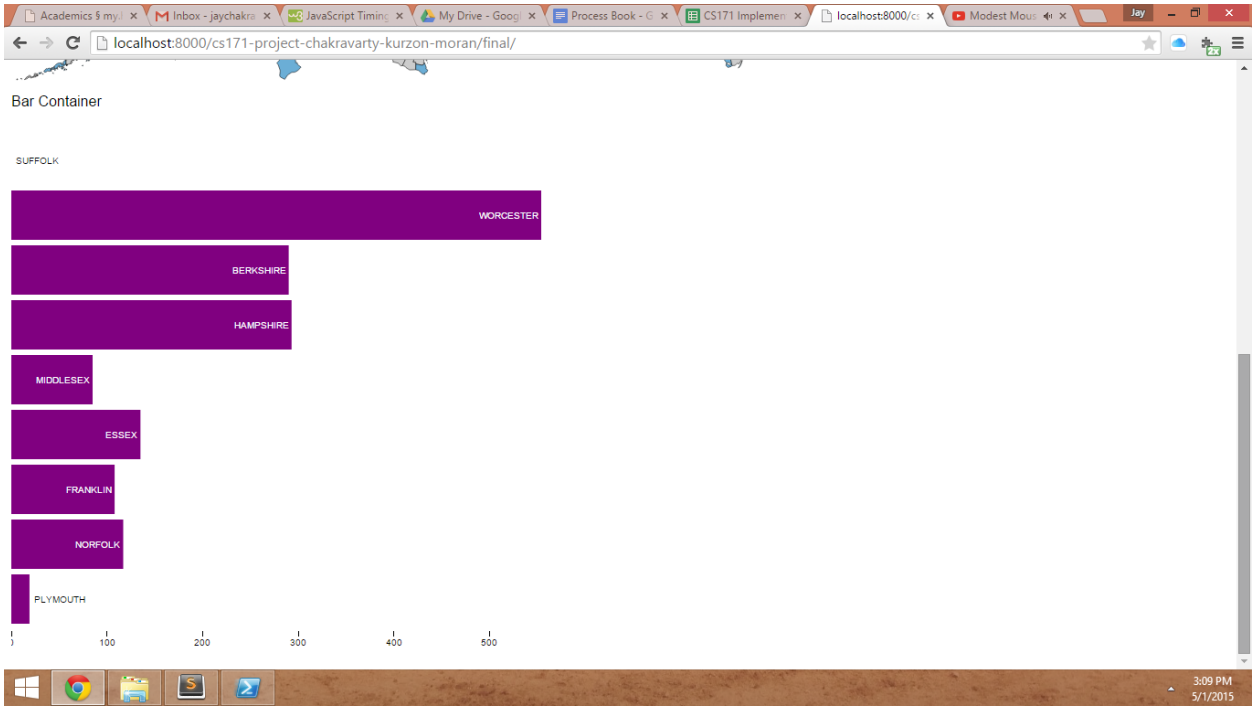
4/30/2015 - Feedback

[entry by jchakravarty]

Today we met with our TF Daniel to receive feedback on our project.



Currently our visualization includes a choropleth map with yearly accumulated snowfall encoded by the blue color gradient. Counties filled with darker shades of blue have more accumulated yearly snowfall. One of the current issues is missing values which are displayed in the map as uncolored counties. We might want to consider using interpolation to handle these values.



Our current visualization also includes a vertical bar chart located beneath the map. When the user clicks on a particular county, the bar chart displays the aggregated snowfall values for all counties in the same state as the selected county. The screenshot above displays the bar chart for the state of Massachusetts. For smaller states with fewer counties, the bar chart displays the aggregate data effectively, however for larger states with more counties the chart is not able to fit all the bins. We hope that moving the bar chart up toward the side of the map will create more available space for extending the dimensions of the chart such that it will be able to contain all bins. Additionally, we plan to add another chart in the same location as this one which will respond to brushing on the map. We are considering a bar chart with each bin corresponding to a month of the year, and the chart value representing the aggregate snowfall for the month across the region covered by the brush. This will require us to translate the brush location coordinates in terms of county boundaries.

Currently we have a slider which users can use to modify the year from which snowfall values are shown. The time slider experiences a delay which wel will correct by dividing our data by year and only reading in data for the selected year. We plan to also include a play button animation which progresses the slider over the years at regular intervals updating the map with data from the currently selected year.

The tool tips and hovering must be fixed such that the tool tip displays county name instead of county code. We are also considering adding tooltips to the monthly side graph. This along with a proper title and appropriate animation transitioning between the monthly and line graph will strengthen this the chart section of our visualization.

We had a team meeting after this TF meeting and assigned tasks using this spreadsheet: https://docs.google.com/a/college.harvard.edu/spreadsheets/d/1m-GDEZlct_Nx1dCIUQm7i5uPltWd6XACH7Wsmh-5TCg/edit?usp=sharing

4/31/2015 - Dealing With the Daily Data

[entry by ckurzon]

Problem: At the moment, we can't load daily data for both snow depth and snow fall efficiently because these datasets are too large. In order to show a daily line graph for both of these datasets, we need to shrink down this data size.

Solution: I shrunk the size of these datasets by almost half (over 100 MB to approx. 50 MB) in excel. I did this by deleting the extraneous data items we weren't using (e.g. lat/longitude), and by deleting any data items that were missing important information, and therefore not be included in the map anyways (e.g. data items that were missing county names). As a result, we are now able to load both datasets and a reasonable time

4/31/2015 - Play Button Animation Using D3 Slider with External Button

[entry by jchakravarty]

The play button animation makes use of this piece of code:

```
d3.select('#play').on('click', function () {
        if (p == false) {
            pos = 2004;
            setInterval(function () {if (pos <= 2014)
{document.getElementById('slider-button').click();
                                        newText(storyObject, pos.toString(), "1");
                                        }}, 3000);
        p = true;
        }
        else {
            document.getElementById('stop').click();
            p = false;
        }
    });
```

The d3.select allows this function to be called whenever the user selects the div with id=”play”. The var p is used to track whether the button is in the on or off position. It is initialized as false so that when the button is first clicked it calls the JavaScript setInterval function which generates clicks for the slider-button span element responsible for incrementing the d3 slider at regular intervals. The 3000 passed in It is this function which allows the animation to proceed as a sequence of slider positions. The setInterval function also calls the newText function which updates the storyline contained in the text object. Each time as the position increments the text becomes a new text file. In the case when the animation is already playing and var p is set to false, clicks are instead generated for the stop span which stops the movement of the slider.

5/1/2015 - Zooming

[entry by amoran]

Due to the size and vlomue of counties, I thought it was best to have a zooming and panning functionailty that allows users to view a focalized region of interest. I adopted mbostock's block #2206590. Click-To-Zoomapproach. This seemed great for counties which were hard to see and axtract information from the normal map scale.

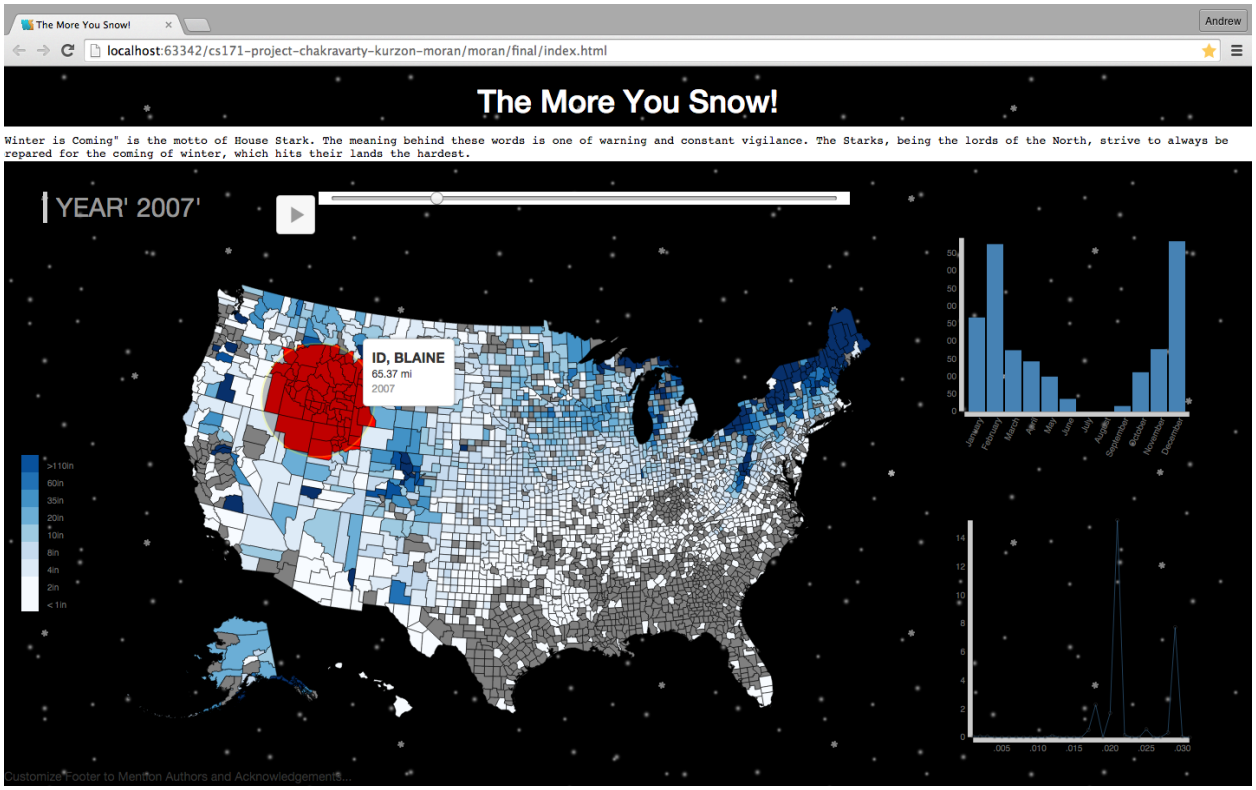
This unexpecdtley had a panning compoennet as well. So once zoomed, clicking on another county scales to that location (basically recetered your screen) What was more unassumiing was that it was hard to zoom back out to he normal scale initially. This required clicking on the smae couty you initally clocked (if first tim eentering zoom). Or the most centered county (ini which was breviously slecetd).

This was a feature that users and I wanted to implemnet becaose w=t was hard to define fine lindsa beteweee very closely assoicated storms and snow fall patterns.

One caveat is that zooming out can be dffictl if you exampd and brush for too whwich coveres all clickable space, of even forget the man count yyou origianlly clciekd on. That;s why I am thining of implmenetsing a map with an additonal afforadance (zooming scorll, button click) to reset the zoom.

5/1/203-5/3/2015 - Brushing

[entry by amoran]



One aspect we believed would be very useful while viewing a snow geography of the US is to select multiple counties and look at a particular region of aggregated snow data. This can be done in the form of brushing as we have seen in class and on previous psets. This produces a transparent element representing a desired region of interest on your visualization for selecting data points. One thing I would like to keep in mind is that D3 defaults to a brush which was meant to be for geometric 2D selection (like selecting a time range or on a grid). I thought it would be more intuitive to use a circle SVG that creates a localized region relative from a source county. This way, we can compare the snowfall of surrounding counties within a mile range while recalling a centralized location (e.g miles from a county as compared to a bounding box of lat,lons).

Implementing brushing required some additional magic to get it to perform properly and with the ideal functionality we wanted. As shown above, clcking a county is resevred for zooming. Thereofre, I utilizd the dragging functionality (whch is already the convention form brushing).

- Drag start - if you select a county however then start moving the mouse, the transparante circle brush appears.
- Drag move - The county you initially clicked on is the source and the distance the mouse is from the initially clicked county determines the radius of the brush

- Drag end - Releasing the mouse produces the final brush and the final list of selected counties.
- Scrollwheel - adjust the size of the radius while hovering over brush. This required some additional lookups in how D3 does scales.

When doing brushing, knowing which counties to select (and for every frame the mouse is moving) in an expensive process. As the region gets bigger when selecting on a growing region, we need to constantly on every move/drag event to check if any of the possible counties have left/entered the brush. I thought that sorting the counties in order from prxomity from a source county would help boost performace in the sense that once we reach a county that is outt of bounds of the radius, then all succeeding counties would then not be selected. However, this had little impact.

To get the prxomit for a county, I avearged all points of a counties path and stored it as d[XY] on in stantiation so its an easy lookup when continuing t prpgram.

```
        var dist = Math.sqrt(
//            Math.pow(d["XY"][0] - that.proximityStart[0], 2) +
//            Math.pow(d["XY"][1] - that.proximityStart[1], 2)
//        );
```

If within the radius range, store the fips id of the selected counties and bpass it in to the coordinated bar and line graphs;

5/4/2015 - Nonlinear Scale for Color Scheme

[entry by ckurzon]

Objective: As of right now, we have a linear color scale for our map, with only 5 colors and color changes for every 10 inches. This makes it difficult to see differences among counties with relatively low snowfall (for example, it's impossible to tell the difference between counties with 0 snowfall and 5 inches of snowfall). We want to implement a nonlinear scale for the map coloring, so we can see these changes more clearly.

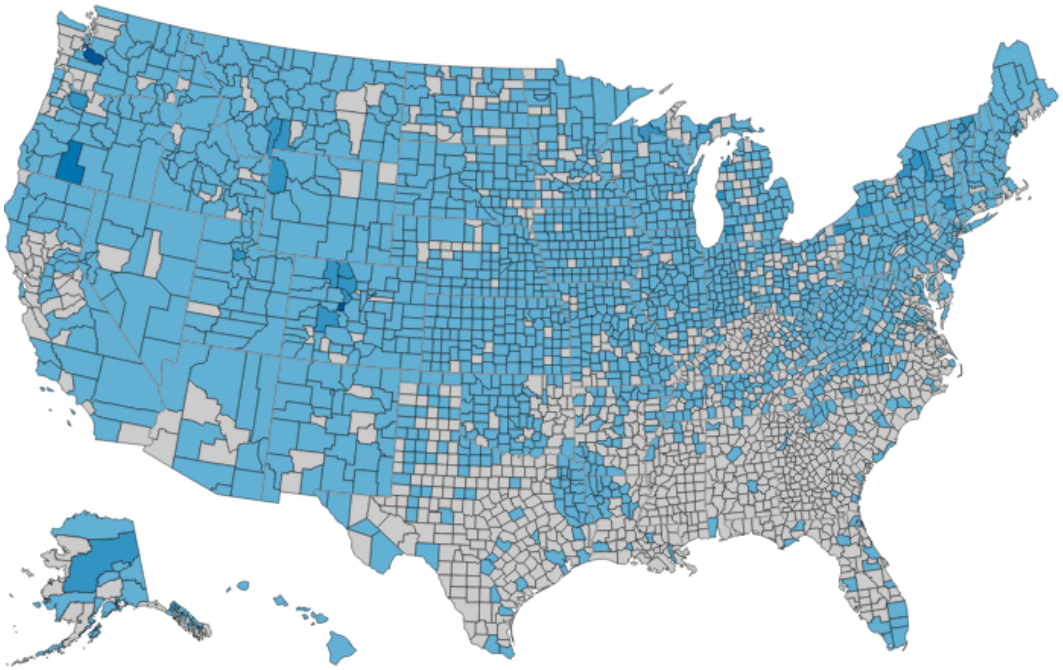
Methodology: I found this github library meant for creating nonlinear color scales: https://github.com/gka/chroma.js. More thorough description: https://vis4.net/blog/posts/mastering-multi-hued-color-scales/. Using the functionality from this github library, I was able to create a logarithmic color scheme with just a few lines of code:

This is what the code for creating the scale looks like:

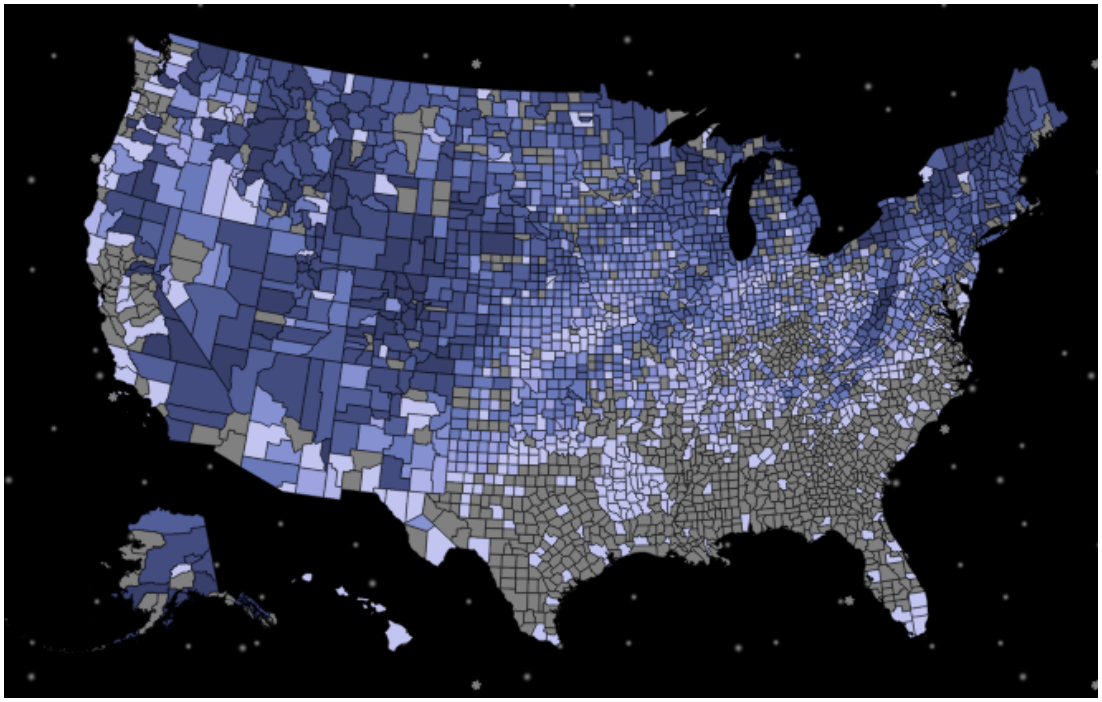
```
this.colorpow = chroma.scale(['eaf1fe', '76a2ee', '3475e4','094cbf','052356']).domain([1, 100], 8, 'log');
```

Results:

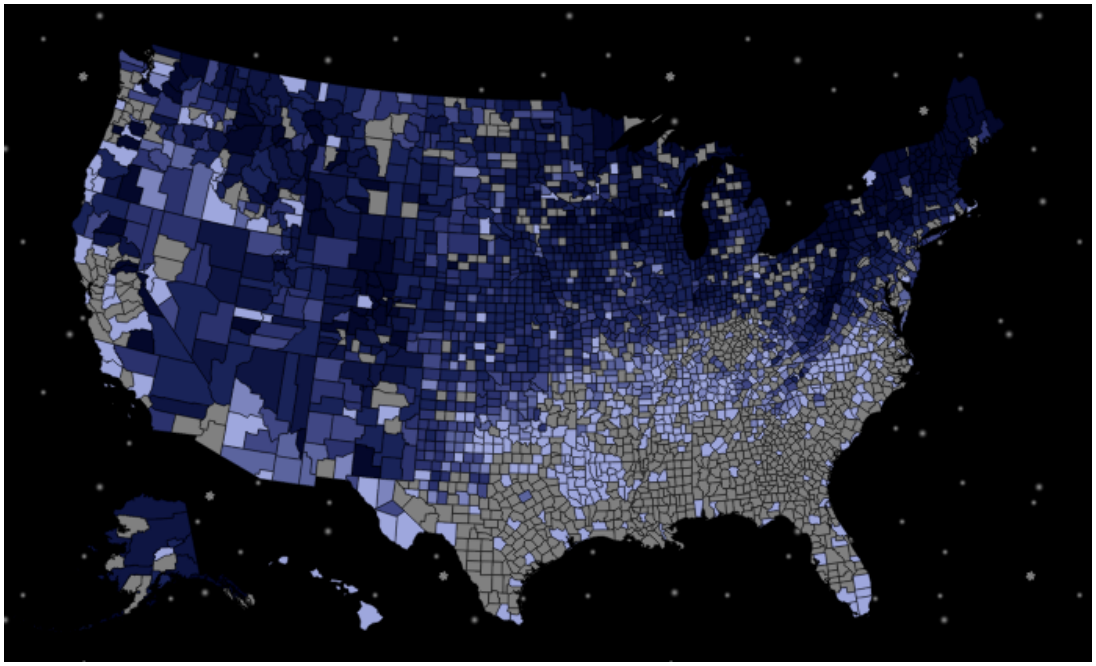
Original Map:



New Color Scheme 1:



New Color Scheme 2:



New Color Scheme 3 (scheme we chose to keep):