INFO/CS 3300
Final Exam
Due Thursday, May 18 at 11:59pm (no extensions, no late work)

You will find five HTML files in the zip file along with this file, which correspond to the five sections of this exam. You will answer all questions by editing these files. Some questions involve coding, some involve short written responses. When you edit Javascript code, the graders will appreciate if you add a comment stating what you changed or added. For short answer questions, there will be a `div` element in the HTML file for the section containing the problem. Clearly state the question number when writing your answers. When you are done, save your edits and package all files in a new zip file. Turn this zip file in through CMS.

There is no "drop lowest" policy for the final. You *must* make sure that the file you submit is correct. Once you submit, download your zip file from CMS and verify that it is the correct size, that it opens, and that it contains your final version. We have had problems with students submitting Mac shortcut files instead of the original zip file. These files will be about 300k.

This exam is due Thursday, May 18 at 11:59pm. There is no specific time limit. You do not need to finish it in one session. You are, in fact, strongly encouraged to work on it over the course of the week.

You may use online references such as APIs and all class notes from GitHub. Reading previous posts from question forums like StackOverflow is permitted, but "official" resources like the D3 docs are preferred, and more likely to be useful. **You may not communicate with any other person in any form.**

If you have any questions, please write to mimno@cornell.edu. Any corrections or clarifications resulting from such questions will be posted in a pinned note on Piazza. Start early. Upload to CMS frequently. If for some reason CMS is not accepting your submission, mail it as an attachment. Make sure that you have submitted the correct file. Timestamps from your computer are not admissible.

1. Debug code

As you know, "coding" is really mostly debugging. This exercise gives you a chance to test your bug detective skills.

a. The file `whyvis.html` contains a script that should display a table of summary statistics and 13 scatterplot of light blue dots, but is not working. There are 25 errors in this script. Edit this file to fix the errors. Include comments to describe your changes. Each "error" consists of a contiguous sequence of characters that must be edited. Some lines may contain more than one error; count these individually. Note that in some cases the same

type of error occurs on multiple lines of code (that is, the same error is made in consecutive lines). These count as multiple errors. All edits will be in the code, the data file `DatasaurusDozen.tsv` has no errors (but it may be worth looking at). The lack of axes or labels is not an error for this problem. If you're stuck, try adding console.log() statements, and use the Element inspector tool in your browser to see what you're actually creating in the page. (25 pts)

b. In the free response section of this file, comment on why we like to use visual displays of quantitative information. (5 pts)

2. Manufacturing jobs, time-series plots

We heard a lot about manufacturing jobs during the recent US election campaign, mostly with regard to international competition. But what about domestic competition? In this problem we'll look at data from the US Bureau of Labor Statistics for all manufacturing employment *by state* over the period from 1990 to 2016. You will modify the file `manufacturing.html`. Data (heavily processed by me) is in the file `manufacturing.txt`.

Remember how we constructed the approximation of the hammer throw trajectory by first calculating changes in velocity, and then adding those up to calculate positions? Here we'll do the opposite: start with position (i.e. job numbers) and then calculate a "velocity" of change by subtracting to find differences.

a. Add code to turn the raw input from the `d3.tsv()` call into usable data. You will need to convert the `Jobs` field into a number and the Date field into a javascript Date object (there may be other fields, you do not need to change these). Do not worry about timezones for Date objects. Next filter the data to only include four selected states, NY, TN, WA, and OH. (Useful code is provided.) Now apply three transformations to the data: (1) smooth each state's timeseries by substituting each value (except the two on the ends) with the average of the value and the immediately adjacent values; (2) calculate the "velocity" value *v* for each element by comparing it to the previous value; (3) smooth the velocity values in the same way. (5 pts)

b. Fill in the body of the `plotJobs()` function. This should create an `<svg>` element in the `#jobsPlot` div. You will need to define appropriate scales and axes, and add one `<path>` element for every state, showing the number of manufacturing jobs in that state during each month over 26 years. Use appropriate css styles (we will not clarify that, it's up to you) to differentiate the four states: NY, TN, WA, and OH. Add text elements to show us which line corresponds to which state. (5 pts)

c. Now create another view of the same data, showing the *rate* of change using the velocity value you calculated earlier. Fill in the body of the `plotJobChange()` function. This should create an `<svg>` element in the `#jobChangePlot` div. You will need to define appropriate

scales and axes, and add one `<path>` element for each state, showing the monthly change in the number of manufacturing jobs in that state during each month over 26 years. As before, differentiate NY, TN, WA, and OH and label each state's line in the right margin. Add text elements to show us which line corresponds to which state. (5 pts)

d. Answer the following questions in the provided spaces. (5 pts)

- How has NY manufacturing done relative to OH? What about TN? How has US manufacturing performed in the 1990s, the 2000s, and the 2010s?
- What question can you answer from the change-in-jobs plot that would be difficult, or impossible, to answer from the original jobs plot?
- Going the other way, what question can you answer from the original jobs plot that would be difficult, or impossible, to answer from the change-in-jobs plot?

3. Driving fatalities.

The number of traffic deaths in the US for 2016 was released recently, and it continues a surprising trend. This problem is similar to the previous one, but with more of a focus on data collection and curation. You will modify the file `fatalities.html`.

a. Compile a full dataset of traffic fatality counts. Combine data from wikipedia with the new 2016 value from the National Safety Council (see the HTML file for links). You may construct this as a separate file or as a JSON element in the HTML file. (5 pts)

b. Plot the time series using a `<path>` element in the `#plot` div. Use appropriate scales and axes. (5 pts)

c. In the provided field: What factor do you think might have caused the observed increase in fatalities? What additional information would you collect in order to test that hypothesis? (You do **not** actually have to collect any data for this problem, so be creative!) (5 pts)

4. Design choices in French election results

The `FrenchElection` directory contains five screenshots of choropleth maps of the recent French election. We looked at them briefly in class, but there's a lot to think about. All were done by professional graphic design/data visualization shops: two American (New York Times, Bloomberg), two British (the Guardian, the Financial Times), and one French (Le Monde). The race was between two candidates, Macron and Le Pen. France is divided into large regions, smaller departments, and, at the smallest visible level, communes. These include several small outlying islands and territories in the Atlantic and Indian oceans. All five maps show the same data.

What choices are these designs making? In the file `frenchelections.txt` discuss the following questions:

a. How does each map use color? Answer in terms of color values such as hue, saturation, and lightness. (Focus on the colors themselves, talk about how they are placed in the next question.) (4 pts)

b. How does each map display boundaries and vote data for administrative divisions (region, department, commune)? How does this choice affect the information that is presented? (4 pts)

c. How does each map deal with regions of high density and outlying regions? (4 pts)

d. What elements are used in the maps to help the reader recognize the meaning of different elements? Examples might include (but are not limited to) labels, guides, and annotations. For each map, say which of these elements are present and which are not. What does this choice tell you about the designers' expectations for the cultural background of their audience? (4 pts)

e. What is missing? What is a piece of information you would like to have that is not present in any of these maps? Why would that information help? (4 pts)

5. Non-linear linear regression; learning rates

We spent some time looking at linear regression models, and in each case were trying to fit a straight line to a set of points on an x,y plane. But what if the points line up not in a straight line but in a curve? Linear regression is no good, right? Wrong! In this problem you will work with a linear model that fits a curve, and determine the effect of a "learning rate" parameter in fitting this model. You will modify the file `learningrate.html`. Read the code and the comments carefully, but most edits will happen at the end of the file.

For a model to be a linear model we just need to have a set of input variables, each of which has a parameter. There's no reason that the input variables have to be independent of each other. In this case, we will use a model with three inputs: x, x^2, and a constant. Adding the square of x gives us a quadratic equation, so rather than just fitting a line we are fitting a parabola.

a. The code samples 100 points from the model $y = ax + bx^2 + c$, plus some added Gaussian noise. What does the curve associated with such a function look like? Previously we've been able to show the model as a <line> element, but for a curve we need to be more sophisticated. We'll approximate the curve by constructing a `<path>` based on a series of x,y pairs. In the function `displayCurve()`, create an array of objects containing `x` and `y` variables. The x variables should be evenly spaced between -6 and 6 with 0.2 between each

value. The y values should be the y value predicted for the associated x value by the model that was passed to this function as an argument. Add x/y scales and code to create a d3 function that maps this array into a string that you can pass to a `<path>` element, and use these to update the `fitPath` element. (See the comments in the code for details.) (5 pts)

b. *Sparklines* are small "word-sized" line graphs popularized by Edward Tufte. I want to use these to show you the effect of the *learning rate* parameter in an algorithm for fitting a linear model. The function `sparkline(g, values, label)` will create a sparkline in the element `g` showing a line representing `values`, and an indication of what the sparkline represents (`label`). Fill in the body of this function so that it works. Define an x scale with domain from 0 to 50, and a y scale appropriate for the "values" array. I want the x scale to remain fixed so that the line appears to "grow" as we add values, but I want the y scale to change dynamically with each update. The `sparklineHeight` global variable defines the height of the sparkline group elements. Set the range of the y scale to include some padding between these elements. Generate a function using d3.line() to map "values" into a path string. Add a <path> for the sparkline and a <text> element for the label. If this function is working, clicking the SVG element repeatedly should draw five sparklines showing model variables. Refer to the included screenshot file for reference. (5 pts)

c. The sparklines will now show you how the algorithm I implemented affects the parameters of the curved linear model. (Click the SVG repeatedly to run the algorithm.) I'm calculating the gradient with respect to the three parameters *a, b,* and *c*. I then update those parameters by adding that gradient multiplied by the `learningRate` variable. A larger value of this variable will move the parameters further each time, a smaller value will move them more slowly. The default learning rate is 0.01, defined on line 118. In the `<p>` tag at the top of the page, describe how changing the value of the learning rate affects how well we minimize error and learn values of the three model parameters. Reload the page several times for each value, since both the sampled points and the initial model will change each time. You do not need to write code for this part, but you should try at least one bigger and one smaller value. Interesting learning rate values might be between 1.0 and 0.00001. You might also try changing the `rateDecay` parameter, but this is not required. (5 pts)