Ernie Tsang(301211038), Jeremy Lo (301143659)

Audience

The intended audience for this visualization has not changed. Our team is still focused on targeting a few key demographics with the visualization.

Young jobseekers and new graduates are still the primary audience. These people are typically between 22 and 27, with a post-secondary education, looking to buy a permanent home in the north, south, or central regions of Surrey. With their youth and inexperience in the workforce, introductory salaries ranging from $40,000 to $65,000 CAD annually is expected -- certainly not in the territory of hedge fund managers or venture capitalists. Though lifestyles in this age bracket are subject to change, it is assumed that the number of children for this primary audience will be between 0 and 3.

For the secondary audience, realtors and investors aged 30 to 50 with an interest in Surrey properties for the purpose of profit before residence.

Questions and Answers

With the shape of our current data, we've begun to start exploring some of the informations beyond price. Thanks to the presence of some general population data and summative listing data, a new question and answer has come about: *is the change of population related to the changing number of listings in 3 of Surrey's regions*?

After some digging with the visualizations (one for number of active listings, the other for population) there does not seem to be any relation between the two. As each year progresses, the number of active listings present in all 3 regions of Surrey seems to oscillate between the summer and winter months: Many listings become available around April/June, and vanish as the year comes to a close to ring in the coming new year. Population, on the other hand, happens to maintain a very linear growth across all 3 regions, save for a strange spike in North Surrey in 2010.

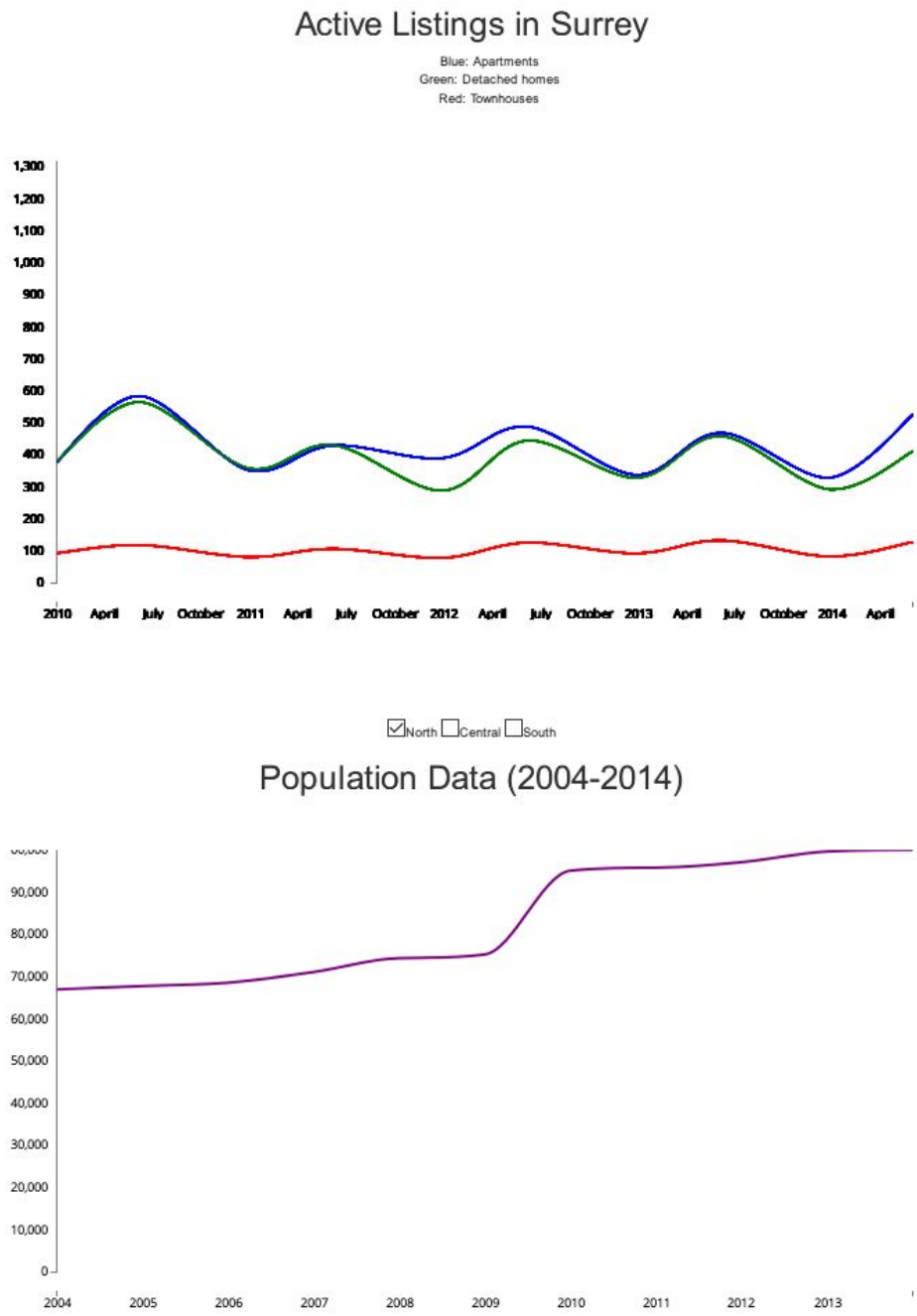
Data Mapping

The previous view's bar-like layout has been scrapped in favour for 2 line graphs.

For the first, the y-axis is mapped to the number of active listings for any given data point, ranging from the lower 100's to upwards of 1300 (central Surrey seems to be a hotspot for detached homes). The x-axis maps time, ranging from January of 2010 to June of 2014, where our listing data sheet ended. Overall, the first line graph gives a spread of 4 years of real estate listing data across the north, central, and south Surrey regions for detached homes, townhouses, and apartments.

The second graph is another line graph, but this time measuring the population of the 3 regions. Similarly to the previous, the x-axis is used for time, but here, it's measuring annually between 2004 and 2014, giving it a wider scope than the active listing data. This may be a cause for future confusion, so some pruning or brushing implementation may be in order to remedy this. To measure actual population, the y-axis is used, ranging from just above 20,000 to 100,000.

Some interactivity is introduced to both of the visualizations. Hovering any line will brush it to alter the colour to **black**, while also increasing the line's weight. Filtering, although currently buggy, is present to hide and show data from each of the regions. It is possible to view data from any 1, 2, or all 3 regions at once for both population and active listings from a single set of checkboxes (image on next page).



Visual Idiom

Because the data we're dealing with is exploring *change over time*, line graphs were deemed to be the optimal choice. Scatterplots, while good at revealing relations (and lack thereof), unfortunately are difficult for us to incorporate across the dimensions of population, listing number, *and* time all at once. With that in mind, a vertical stacking of the graphs reveals change in our desired dimensions, while keeping all information relatively close together. Optimally, this is viewed on a display with a slightly larger height (1080p minimum) in order to leverage the layout of the checkboxes for filtering both visualisations.

Ernie Tsang(301211038), Jeremy Lo (301143659)

Code Overview

NOTE: If the checkboxes do not work, try refreshing the visualisation until checking and unchecking changes the graphs. This is an intermittent bug that we have not been able to reproduce reliably.

index.html

Not much to say about this one. There are 2 svg elements, labeled with appropriate IDs for later reference in the code. Checkboxes with the labels of the region are placed and appropriately tagged with IDs for usage with our JS and jQuery components. Some bootstrap dependencies were brought in for future CSS functionality, but for now, their usage is minimal. D3 and our style sheet are included in the head, and the remaining scripts (the 2 visualizations plus our jQuery dependency) are included along the bottom of the body.

HouseholdVis.js

What started as a secondary practice file (the first being written following a tutorial) ended up as the main logic for our Active Listings visualization. First and foremost, a jQuery call is used to ensure that the document is ready to be manipulated. The visualization is defined with some preset metadata to make appending easier down the line (MARGIN holds all the margin data for our scaling ranges, WIDTH and HEIGHT serve as global boundaries). With the visualization defined, now begins the helper functions.

Helper functions

- **`getDateRange(d)`** - Takes an input data set and hunts for a “Date” column, harvesting all dates and returning them in a list, not necessarily ordered.
- **`returnDataSet(d, desired_region, desired_type)`** - Scans a dataset *d* and returns a list of entries matching the region and listing type. Used to fetch listings from north/central/south Surrey, matching apartments, townhouses, and detached home listings.
- **`date_sort_asc(date1, date2)`** - Sample taken from a coding tutorial. Passed in as a parameter for calls on `sort()` to allow our JS to sort lists of `Date()` items.
- **`sort_obj_by_date(a,b)`** - Another snippet taken from a tutorial, to sort objects that *contain* `Date()` items, in ascending order.
- **`fixDataRow(d)`** - Pulls in the data set and formats the “Date” column to hold `Date()` objects rather than strings, as well as converting the “Households” category to numbers. Returns d after running its course.
- **`setupFilters(svg, points)`** - From the in-class tutorial. Listens for a click on any of the checkboxes, and calls the draw function again if so.

- **`drawLineGraph(svg, points)`** - The main event. Uses `getDateRange()` to retrieve and process dates into a list. This list is then sorted, with the first and last items being saved to be put into the `xScale` domain. 9 arrays are filled with respective entries suiting their region and type criteria, and sorted internally using their dates. As clarified above, the `xAxis` uses time, so the scale provided makes its declaration easy. `yAxis` on the other hand is linear and only needs to be bound by the min/max values found in our data.
  - **`LineGen(d)`** is created as a small helper to make data entry in line creation more elegant (saves 6 lines of code per line definition), compacting the x/y definitions and the interpolation statement.
  - **`visAppend(region, data_set, type, width)`** similarly is used as another shortener. The act of appending lines to vis is quite large, and by specifying a region, a data set, the type of the data, and some line width, the lines can be attached to the svg. The region will give us filtering options, while the type will dictate what colour the line is (defined in the CSS)

After this, some jQuery logic is called to do a read on all of the checkboxes. If any of their states are different from the last call of **`drawLineGraph`**, then certain lines will be hidden or shown. This also applies to lines in the population vis. With the helpers defined, the only thing left to do is actually call the d3.get(), fix the rows, check for errors, and call **`drawLineGraph()`** as well as **`setupFilters()`**. Because the `setupFilters` function continues to call `drawLineGraph` whenever a checkbox is clicked, the jQuery check for hiding and showing lines is called again. Since the logic for hiding and showing is class-based, there's no need to include this code anywhere else.

PopulationVis.js

This JS file supports the population visualization, and is called after PopulationVis`.js`. This way, certain functions from the previous file can be used as helpers here. A lot of the functionality remains the same, however, the resource csv files are slightly different in format. That meant that certain helpers had to be tweaked or rewritten.

Helper functions

- **`fixPopRow(d)`** - Changed version of the row fixer in PopulationVis. Converts the population data to numbers, and the year numbers to `Date()` objects.
- **`popFilters(svg, points)`** - Modified version of the `setupFilters()`, re-draws the population graph on each checkbox click.
- **`returnPopulationSet(d, desired_region)`** - Because the population data set lacks a 'type' field, this function operates using only an input data set and a desired region. It will iterate through the data and return a list of all entries belonging to one region.
- **`drawPopGraph(svg, points)`** - A shorter version of the `drawLineGraph()`, mostly because there's only 3 lines rather than 9. Dates are again pulled in and sorted to give basis for our scales and axes, and arrays are built to house information for the population of each of the 3 regions using our helper functions. Another `LineGen` is defined, as well as another vis-appender helper function, which operate on the `#population` svg rather than the `#visualisation` one.

As stated before, because line hiding and showing is called in the PopulationVis.js file, none of the hiding/revealing logic is needed in this new file.

Bibliography and Resources

Line Chart Tutorial

http://code.tutsplus.com/tutorials/building-a-multi-line-chart-using-d3js--cms-22935

Sorting Date Objects

https://onpub.com/how-to-sort-an-array-of-dates-with-javascript-s7-a109

Sorting Objects by Date Key

http://stackoverflow.com/questions/19430561/how-to-sort-a-javascript-array-of-objects-by-date

Population data source

http://www.surrey.ca/business-economic-development/1418.aspx

jQuery

https://jquery.com/download/