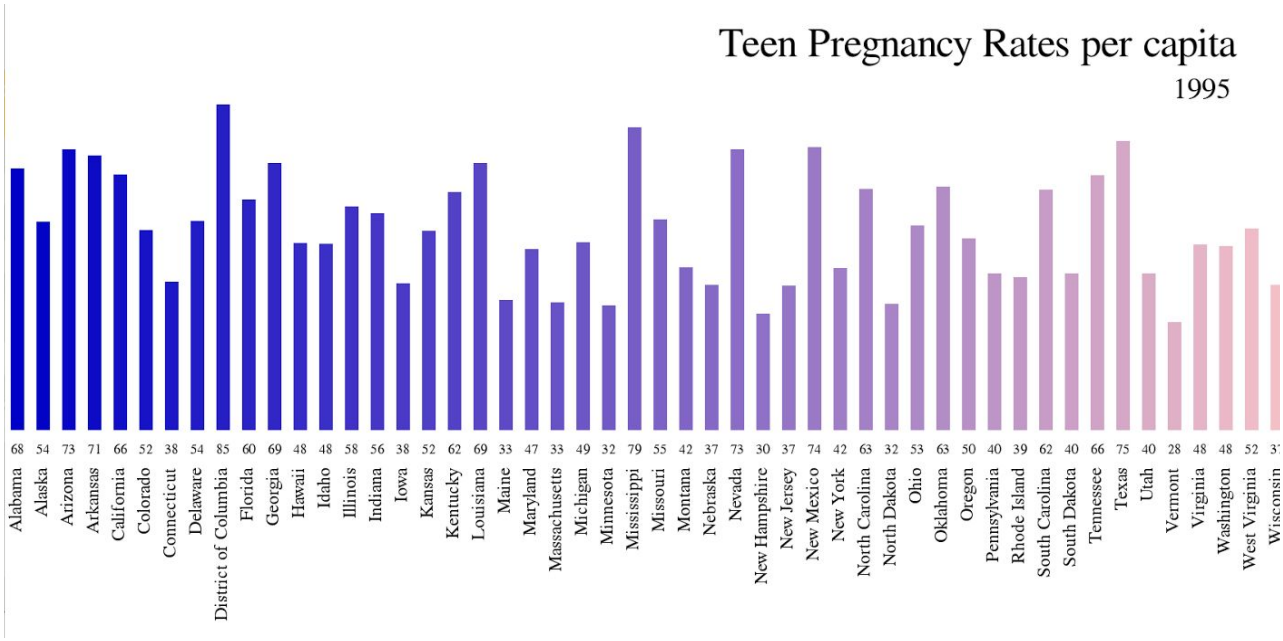
Mini Project 4 - Data Visualization
Emma Price - March Saper

Project Overview

Our project visualizes data using a bar graph with an amount per capita as the y axis and a state as the x axis. As the mouse moves, the bar graph changes to represent data from different years. Presently, the data that is being visualized is teen pregnancy per capita in all fifty states, including Washington D.C. from 1990 to 2014.

Results

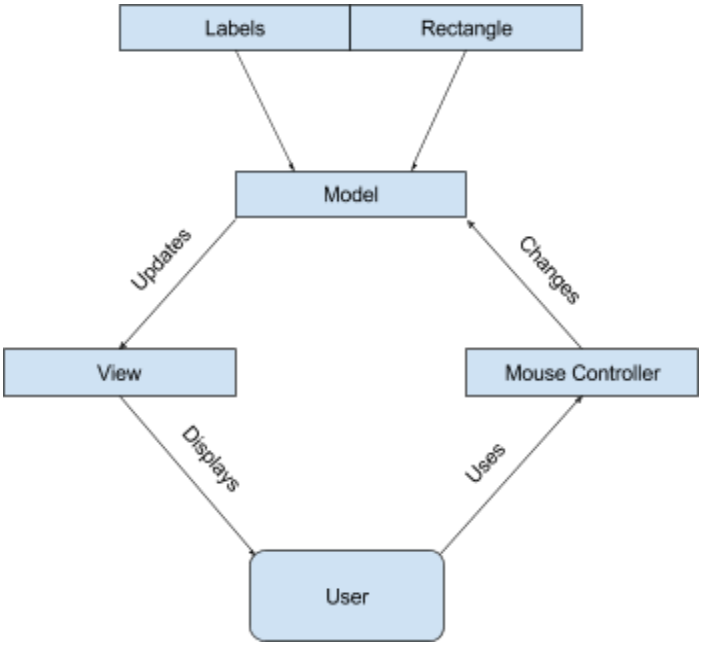


A screenshot of our program with data from the year 1995

Our program takes data from a csv file and uses the rates of teen (ages 15-19) pregnancy per capita (for each 1000 girls who are 15-19) to control the height of the bar in the bar graph displays a bar for each state, sorted alphabetically. Below each state name is the corresponding teen pregnancy rate. This information allows users to easily compare the rates across states by looking at the size of the bar and by look at the actual numbers. They can also easily interact with the program by moving their mouse to change which year of data they wish to display. The year that they are looking at is displayed beneath the title.

Some interesting findings from this data set: Washington D.C. has the highest teen pregnancy rates compared to all other states from 1990 through 1996, after which Missippi hold the lead until 2013, when Arkansas takes over. Unsurprisingly, the data has a general trend downward, likely due to an increase in the availability of birth control and an increase in contraceptive education. An interesting rise can be seen from 2005-2007, when there was a national debate over the idea of federally funded abstinence-only education.

Implementation



UML diagram

Above is our UML class diagram. We made use of the model, view, controller framework which constitutes our self-defined classes. In addition, we used two built-in classes from PyGame: rect and font. The Controller class purely tracks the position of the mouse. When the mouse position changes, it calls an update method on the Model class. The Model class initiates the necessary Rectangles and Labels. It also contains an update method that updates the year, bar height, and pregnancy rate accordingly. (And for some locations, a surprise.) The View class takes what is happening in the Model class and draws it, after initializing the surface on which to do so.

We made heavy use dictionaries in our code, both to organize the data and to create the necessary classes. Each state required four pieces of information it's values over time: rectangle object, color, label, and pregnancy rate. Each piece of information was placed into a separate dictionary with the corresponding state name as the key. These dictionaries at first gave us trouble, since dictionaries are not ordered, when they were drawn in the View class, the bars, labels, and numbers did not line up. Briefly we contemplated using a different data structure, but decided to keep using dictionaries and order them using the library "collections". For our purposes this worked very well.

Reflection

From a process perspective, we started simple by first creating a rectangle and then changing it's height based on the position of our mouse. We then placed this into the model, view, controller framework and started working with multiple rectangles. After this point, we imported our data set and began working with it. This process could have been improved by fully understanding the data-structure that we chose to store our data and objects etc. We had to spend some time later in the project going through our dictionaries and sorting them. We can improve our project by making it more modular, so that any csv file with one column of keys and another with values that change over time. Additionally, we would like to come up with new ways of viewing the data besides a bar graph, possibly a map, pie chart, circular graph, etc. Our project was appropriate to our skill level. Though it could be improved, it is functional and cleanly written

We planned to divide the work through pair programming. In actuality, we ended up programming individually, but together. We were each working on a separate part of the program, but working at the same time in the same location. This allowed us to focus on different problems within the program, but consult on another when necessary, but that was done at the expense of full understanding of the program. There were no issues that arose when working together. In the future we would like to fully commit to pair programming.